BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER SYSTEMS
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FAULT-TOLERANT SYSTEMS DESIGN AUTOMATION

AUTOMATIZACE NÁVRHU SYSTÉMŮ ODOLNÝCH PROTI PORUCHÁM

PHD THESIS
DISERTAČNÍ PRÁCE

AUTHOR **Ing. JAKUB LOJDA**
AUTOR PRÁCE

SUPERVISOR **prof. Ing. LUKÁŠ SEKANINA, Ph.D.**
ŠKOLITEL

BRNO 2023

Abstract

If a digital system is required to maintain a high level of reliability, it must withstand the presence of naturally-emerging failures. Many of such systems utilize *Field Programmable Gate Arrays* (FPGAs). One of the approaches to increase the system's reliability is the insertion of the so-called *Fault Tolerance* (FT) mechanisms. It is, however, a significant challenge to design systems to be FT. In this thesis, an approach is designed and researched, capable of automatically transforming an unhardened design into its FT version. The thesis emphasizes the generality of such a process, which allows for the reusability of the methods among various description formats, languages, and abstraction levels. This thesis describes the proposed method and its main aspects: the source code modification approaches, design strategies, and acceleration of FT parameters measurement. Last but not least, design flows that target the minimization of required measurements are proposed, which significantly accelerates the complete automated design of the FT system.

Several cases were experimentally studied during the research presented in this thesis. Multiple circuits described in different languages were targeted with various reliability metrics to cover multiple scenarios. The first steps use a robot controller written in C++ as a target for evaluating the source code manipulations and the so-called critical bits representation of an FPGA design. After that, our C++ benchmark circuits were used instead of the robot controller. At first, a strategy based on the *Multiple-choice Knapsack Problem* (MCKP) was used to automatically select the most suitable hardening from available hardening schemes (e.g., Triple Modular Redundancy, or N-modular Redundancy). The proposed design strategy found a solution with 18 % fewer critical bits while even lowering the design size overhead compared to the previous approach with the static allocation of FT mechanisms. After that, means of FT mechanism insertion were implemented for VHDL. VHDL benchmarks were also used with the MCKP strategy to find solutions with the best *Median Time to Failure* (a.k.a. t50). For the actual case study, circa 25 % savings in the area were achieved compared to the reference design to which the FT mechanisms were assigned statically and manually. The method allows the user to constrain the available chip area and obtain the result optimal on reliability for this given area (under assumptions specified in the thesis). Also, system recovery was tested, which further improved the t50 results by 70 %. Finally, a comprehensive case was studied on a real circuit, the FPGA reconfiguration controller. This presents a method of finding a Pareto-frontier of optimal designs considering multiple criteria (i.e., power consumption, size, and *Mean Time to Failure* – MTTF). The method exploits the principles of dynamic partial reconfiguration.

Abstrakt

Pokud je požadováno, aby digitální systém dosáhl vysoké úrovně spolehlivosti, musí zachovat funkčnost i v případě přítomnosti přirozeně se objevujících poruch. Mnoho takových systémů využívá hradlová pole FPGA (z angl. *Field Programmable Gate Array*). Jedním z přístupů ke zvýšení spolehlivosti systému je začlenění mechanismů odolnosti proti poruchám (OPP; angl. *Fault Tolerance*). Není však snadné navrhovat systémy tak, aby byly OPP. V této disertační práci je navržen, prozkoumán a popsán automatický způsob transformace popisu systému do jeho podoby zvyšující OPP. Prezentovaný výzkum klade důraz na obecnost tohoto procesu, který umožňuje znovupoužitelnost metod mezi odlišnými formáty popisu, různými jazyky a úrovněmi abstrakce. Tato práce zkoumá navrhovanou metodu a její hlavní aspekty: metody úpravy zdrojového kódu, strategie návrhu OPP a akceleraci měření dosažené úrovně OPP. V neposlední řadě práce prezentuje postup návrhu, který

cílí na minimalizaci požadovaných měření parametrů, což výrazně urychluje automatický návrh systému OPP.

Během výzkumu prezentovaného v této práci bylo experimentálně studováno několik případů. Různé obvody popsané v odlišných jazycích byly optimalizovány dle rozdílných metrik spolehlivosti tak, aby během výzkumu bylo pokryto více scénářů. První kroky ve výzkumu využívají řídicí jednotku robota napsanou v C++ jako cíl pro vyhodnocení manipulace se zdrojovým kódem. Optimalizace se zaměřuje na procentuální zastoupení tzv. kritických bitů (z angl. *critical bits*) na FPGA. Následně byly místo řídicí jednotky robota použity naše testovací obvody, rovněž popsané v C++. K automatickému přiřazení nejvhodnějších mechanismů OPP (např. třímodulové redundance, z angl. *Triple Modular Redundancy* – TMR; nebo *N-modular Redundancy* – NMR) byla nejprve použita strategie založená na *Multiple-choice Knapsack Problem* (MCKP). Navrhovaná strategie nalezla řešení snižující počet kritických bitů o 18 % a zároveň snížila velikost obvodu (obojí ve srovnání s předchozím přístupem se statickou alokací mechanismů OPP). Poté byly implementovány prostředky pro vkládání mechanismů OPP do VHDL kódů. Testovací obvody popsané ve VHDL byly použity rovněž se strategií MCKP k nalezení řešení s nejlepším *mediánem času do selhání* (též známým jako t50). Pro tuto případovou studii bylo dosaženo cca 25 % úspory velikosti obvodu ve srovnání s referenčním návrhem, ve kterém byly mechanismy OPP přiřazeny staticky a ručně. Prezentovaná metoda totiž umožňuje uživateli omezit oblast na čipu, která je pro daný systém dostupná a získat výsledek o optimální spolehlivosti pro tuto danou oblast (za předpokladů blíže specifikovaných v této práci). Rovněž byla testována obnova systému, která dále zlepšila výsledky t50 o 70 %. Nakonec byla provedena komplexní případová studie na reálném obvodu – řadiči rekonfigurace FPGA. V této případové studii se v praxi představuje způsob nalezení Paretovy fronty optimálních obvodů zohledňujících více kritérií, tj. spotřeba energie, velikost a střední doba do poruchy (z angl. *Mean Time to Failure* – MTTF). Metoda také umí využít principů dynamické částečné rekonfigurace FPGA pro obnovu systémů.

Keywords

Fault-tolerant System Design Automation, Electronic Design Automation, Fault-tolerant System Design Flow, Redundancy Allocation and Insertion, FPGA, VHDL, C++.

Klíčová slova

Automatizace návrhu systémů odolných proti poruchám, automatizace návrhu elektronických systémů, postup návrhu systémů odolných proti poruchám, alokace a vkládání redundance, FPGA, VHDL, C++.

Reference

LOJDA, Jakub. *Fault-Tolerant Systems Design Automation.* Brno, 2023. PhD thesis. Brno University of Technology, Faculty of Information Technology. Supervisor prof. Ing. Lukáš Sekanina, Ph.D. and doc. Ing. Zdeněk Kotásek, CSc.

Fault-Tolerant Systems Design Automation

Declaration

I hereby declare that this Ph.D. thesis was prepared as an original work by the author under the supervision of Mr. Prof. Ing. Lukáš Sekanina, Ph.D. and Mr. Doc. Ing. Zdeněk Kotásek, CSc. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Jakub Lojda
April 17, 2023

Acknowledgements

I want to thank Mr. Doc. Ing. Zdeněk Kotásek, CSc., who guided me during the research for many years and provided valuable information, insights, practical advice, and help. I also thank him for sharing his wisdom and (not only technical) knowledge.

In 2022, Mr. Doc. Kotásek passed away. I also want to thank Mr. Prof. Ing. Lukáš Sekanina, Ph.D. for kindly helping me during this difficult time. He became my supervisor and provided advice during the ongoing research and completion of my Ph.D. thesis, for which he deserves a big thank you.

Also, thank all the Department of Computer Systems colleagues who provided valuable feedback and ideas. Last but not least, I would also like to thank my family for their colossal support. Without any of these, the thesis would not have been created. Thank you!

Contents

1 Introduction **3**
- 1.1 Thesis Goals . . . 4
- 1.2 The Proposed Approach . . . 4
- 1.3 Thesis Outline . . . 5

2 Research Background **8**
- 2.1 Field Programmable Gate Arrays . . . 8
  - 2.1.1 FPGA's Structure . . . 9
  - 2.1.2 Single-Event Effects in FPGAs . . . 10
- 2.2 Hardware Description and Synthesis for FPGAs . . . 10
  - 2.2.1 Hardware Description Languages . . . 11
  - 2.2.2 High-Level Synthesis . . . 12
- 2.3 Dependable Systems Design . . . 13
  - 2.3.1 Fault Avoidance . . . 14
  - 2.3.2 Fault Tolerance . . . 14
  - 2.3.3 Calculation with Reliability Metrics . . . 16
- 2.4 Computer-Aided Design of Fault-Tolerant Systems . . . 17
  - 2.4.1 Modifying the Code to Support FT . . . 17
  - 2.4.2 Strategy for Selection of FT Mechanisms . . . 19
  - 2.4.3 Fault Tolerance Evaluation . . . 19
- 2.5 Open Problems . . . 20

3 Research Summary **21**
- 3.1 Methodology . . . 21
- 3.2 Fault Tolerance Evaluation Approaches . . . 22
  - 3.2.1 FT Evaluation Platform utilizing Robot Controller . . . 22
  - 3.2.2 FT Evaluation Framework . . . 24
- 3.3 Research and Results . . . 24
  - 3.3.1 FT Mechanisms Insertion into the C++ Source Code: First Experiments . . . 24
  - 3.3.2 FT Mechanisms Insertion into the C++ Source Code: Multiple FT Mechanisms . . . 28
  - 3.3.3 FT Evaluation: Accelerated Testbeds, FPGA Design Metrics . . . 31
  - 3.3.4 FT Mechanism Selection Strategy: Multiple-Choice Knapsack Problem 36
  - 3.3.5 FT Mechanisms Insertion into the VHDL Source Code: Hardening Methods . . . 41
  - 3.3.6 FT Mechanism Selection Strategy: Multiple Objectives, Real System Case Study . . . 47

3.4 FT Design Automation Overview . . . 53
3.4.1 Design Flow . . . 53
3.4.2 FT Mechanisms Insertion . . . 56
3.4.3 FT Mechanism Selection Strategy . . . 57
3.4.4 FT Evaluation . . . 58
3.5 Publications . . . 59
3.5.1 Selected Publications Summary . . . 59
3.5.2 Author's Contributions to The Selected Publications . . . 64
3.5.3 Other Topic-Related Publications . . . 65
3.6 Research Projects, Grants . . . 68

4 Conclusions **70**
4.1 Thesis Main Contributions Summary . . . 71
4.2 Future Research Possibilities . . . 71

Bibliography **73**

Selected Papers **80**

A Data Types and Operations Modifications: a Practical Approach to Fault Tolerance in HLS **81**

B Redundant Data Types and Operations in HLS and their Use for a Robot Controller Unit Fault Tolerance Evaluation **88**

C Majority Type and Redundancy Level Influences on Redundant Data Types Approach for HLS **95**

D FT-EST Framework: Reliability Estimation for the Purposes of Fault-Tolerant System Design Automation **100**

E Automatic Design of Reliable Systems Based on the Multiple-choice Knapsack Problem **109**

F Automatic Design of Fault-Tolerant Systems for VHDL and SRAM-based FPGAs **114**

G Automatically-Designed Fault-Tolerant Systems: Failed Partitions Recovery **119**

H Automated Design and Usage of the Fault-Tolerant Dynamic Partial Reconfiguration Controller for FPGAs **128**

Appendices **145**

I List of Used Abbreviations **146**

Chapter 1

Introduction

Certain electronic systems are required to maintain a high level of reliability. This is because their failure might endanger human health, e.g., in the case of medical equipment [25]. Other electronic systems must remain functional because of substantial economic losses in case of their potential failure [18]. These include, for example, power grid control. Also, space probe controllers and space rover computers must remain functional in harsh environments. After these systems leave our planet, their reparation is nearly impossible, and their failure can lead to an unnecessary loss of effort and money. It also delays the scientific knowledge these space probes would discover if they continued to work correctly. This also counts for cases where the space probe accomplishes its mission, but it would still be interesting to prolong its operation, as it could lead to another discovery. In other words, an extension of these electronic systems' lifetime is always beneficial, such as in the case of the Voyager 1 and 2 spacecrafts [17].

Many of these electronic systems (controllers, accelerators, decoders, etc.), use the so-called *Static Random Access Memory* (SRAM) based *Field Programmable Gate Arrays* (FPGAs) in which the configuration information is stored in SRAM [19, 63]. These FPGA chips hold highly-configurable structures in one package, allowing to implement complex reconfigurable digital systems. SRAM-based FPGAs thus allow to easily modify or completely change the implemented circuit by re-programming the FPGA. Various approaches exist for increasing the robustness and reliability of FPGA designs. This research targets the so-called *Fault Tolerance* (FT), which accepts the existing risk of a failure and tries to isolate and hide (i.e., mask) its impacts in such a way that the system still performs its function [28].

However, for complex systems, it is a great challenge to consider all the FT approaches during the design of a system. This is why an effort exists, which exploits the automated insertion of FT approaches into already existing systems [41, 76, 36]. This thesis presents research on methods that allow automatic (or semi-automatic) modification of an existing FPGA design into its FT form. As opposed to existing approaches, the methods presented in this thesis target the highest-possible generality – the ability to use the presented methods throughout various description languages and at different levels of abstraction. As a part of this research, all the techniques needed for such an automated design are implemented into a newly created toolkit and evaluated. All the emerging limitations are removed on an ongoing basis with the main research goal: to make the automated design possible and practically usable. The methods will be experimentally evaluated on various case studies on an ongoing basis as the method emerges. The first steps will utilize a robot controller. After that, artificial benchmark circuits will be used. And finally, a real system (the so-

called *FPGA reconfiguration controller*) will be hardened in a case study with optimization of power consumption and size overhead.

1.1 Thesis Goals

This thesis aims to propose, implement and evaluate methods that allow an automated (or, at least, semi-automated) modification of an existing FPGA design into a new one showing requested FT properties. Emerging methods will be experimentally evaluated. To accomplish this overall objective of the thesis, the following goals are established:

Goal 1. Select at least two languages usable for HW system description. Design the means to insert FT mechanisms into HW descriptions in these two languages. Implement these means and thoroughly test them.

Goal 2. Find an algorithmic strategy that suggests selecting the proper FT incorporation methods to the given places of the description code. Implement and test these strategies.

Goal 3. Design a tool that allows acceleration of the evaluation of FT parameters. This tool should minimize the needed user interactions and maximize the speed of measurement (testing).

Goal 4. Use the means from the previous goals to complete an automated design flow for the transformation of unhardened systems into their hardened versions.

1.2 The Proposed Approach

The approach proposed in this thesis tries to develop, implement and evaluate a method for automated design of FT systems. In the literature, various approaches target the automated insertion of FT into existing systems. They are, however, always targeting a specific type of technology [76], certain language [36] or a given level of abstraction [41] of the circuit description. At the time of writing and according to the knowledge of the author of the thesis, none of them was dealing with the automated design of FT systems in a general way.

Suppose that a system is composed of several components and its description is available in a *Hardware Description Language* (HDL). Standard implementations of these components do not support any FT mechanism. Various types of FT mechanisms exist, which can be introduced into a system component **by modifying** it. When a support for FT is introduced into a component, its reliability can be increased; however, the implementation cost is higher. Other design parameters might also be changed, such as power consumption or heat dissipation. The reliability increase is not always in relation to the implementation cost (and the other parameters). Thus, the consequences of incorporating an FT mechanism into a component must be **measured**. Numerous possible configurations exist for the whole system. They typically exhibit different functional and non-functional parameters. Their suitability is highly dependent on the project needs (the optimized parameters). These configurations must be **selected strategically** to find the best configuration that suits the project's specific needs. To compose such a general method of automated design, multiple tasks must be fulfilled:

The means to modify the system's description code are needed to transform the system into its FT variant. These means will be separately implemented into individual modules of

the design method. This allows the design method to support a new description language by simply adding a new FT insertion module for the new language. These means of FT incorporation will be used in a smart way to achieve the best result (e.g., best reliability for a given system size). The optimal selection of the FT incorporation mechanisms will be performed by strategy modules, which are SW components of the proposed FT design automation method that allocate FT mechanisms according to optimized parameters. The generic design of the strategy modules will allow using various modules for FT incorporation mechanisms (for different description languages) with the same strategy module.

A highly accelerated testing framework is desired as the strategy modules will need to receive feedback regarding the reliability parameters. In addition, performing these reliability tests on the target platform greatly maximizes the measurement's credibility. The testing framework will include a tool to perform measurements (i.e., tests). It will minimize the user interactions to fully support the automated design techniques for increasing FT.

The complete automated flow of FT system design will use the FT incorporation modules and strategy modules alongside the FT measurement tool. A complete design flow is proposed, which consumes a system description and produces an FT version of the input system. A simplified diagram of the design flow is in Figure 1.1.

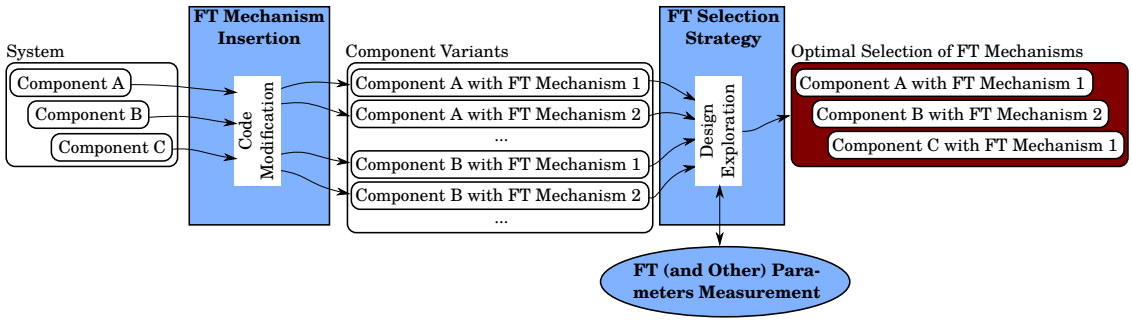

Figure 1.1: The simplified diagram of the proposed FT system design automation with its main steps.

Part of this thesis is an experimental work to support the research and accomplish the overall goal of this thesis. All the experimental work is done on a selected type of FPGA utilizing the Xilinx Virtex 5 technology [79]. However, the complete method is designed to be independent of the FPGA technology assuming the target technology allows the measurement of the targeted (i.e., optimized) parameters. For this reason, the FT measurement tool proposed in this thesis will also have the platform-specific parts isolated in specific modules to allow its porting by reprogramming only these technology-specific parts.

1.3 Thesis Outline

This thesis is structured as a collection of selected published research papers, including a brief summary of the main research results that are important for the thesis goals. Chapter 2 presents a brief introduction to FPGAs, HW description, dependability, and finally, state of the art for computer-aided design of FT systems. Chapter 3 of this thesis presents the main research contributions backed by the selected research papers. At the end of this chapter, these selected papers are also listed alongside the other topic-related papers and

grants. Chapter 4 concludes the thesis and presents the recapitulation of the research contributions, and suggests the possibilities to continue this research.

The research presented in this thesis comprehends several branches, which might be challenging to follow for a reader. For this reason, a *mind map* is included in Figure 1.2, which displays the main goals and results of the research. The research conducted within the Ph.D. study also led to several publications that are not part of this thesis. Despite this, they are shown in the presented mind map, as their further investigation and research might be interesting. These related but not included publications are marked by the blue hatching on the mind map.

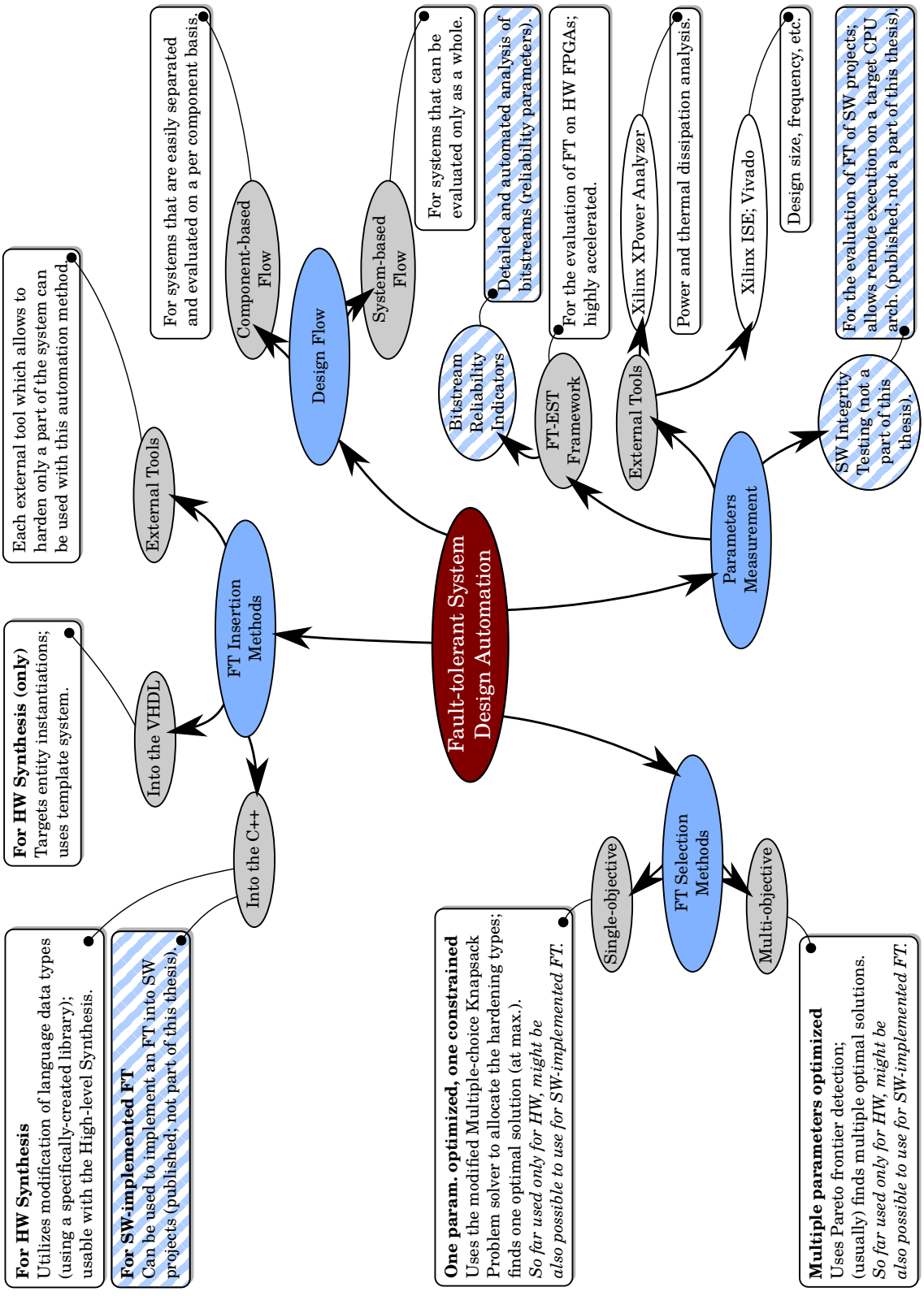

Figure 1.2: The mind map of the research made within this Ph.D. thesis; parts hatched in light blue color were researched, although their results are not included in this thesis, following their paths might be interesting as future research.

Chapter 2

Research Background

The following chapter summarizes the current state of knowledge in relevant research areas and presents the background context of the research. As the research of this thesis has been performed with Xilinx's FPGAs, we will focus on FPGAs of this company.

2.1 Field Programmable Gate Arrays

Field Programmable Gate Arrays (FPGAs) are complex digital circuits containing programmable logic blocks, various pre-designed components and the programmable interconnection network [37]. FPGA's structure is generic, and a designer programs its function after manufacture. Thanks to this approach, FPGAs are a very affordable alternative to the *Application Specific Integrated Circuits* (ASICs). The FPGA's function is derived from the so-called *configuration bitstream*, which is stored on the FPGA in the *configuration memory*. The configuration bitstream is a data string that holds the configuration of the internal blocks on the FPGA and the interconnections among them – in other words, the complete design. The configuration bitstream is *synthesized* from a design description using synthesis tools, such as Xilinx *Integrated Synthesis Environment* (ISE) [80] or Vivado Design Suite [81]. Currently, most design representations are written in the so-called *Hardware Description Language* (HDL); however, graphical representations are also possible [22]. A huge advantage of FPGA is the possibility to modify or completely change its configuration. This allows us to deliver a new upgraded design to an already deployed FPGA or, for example, evaluate FPGA's design functionality easily.

The configuration process is held through a configuration interface. Two types can be distinguished: 1) *External Interface*, such as *Joint Test Action Group* (JTAG) interface, *Serial Peripheral Interface* (SPI), or Xilinx's SelectMAP. 2) With the *Internal Interface*, reconfiguration is possible from the inside of the FPGA itself. These include, for example, the Xilinx's *Internal Configuration Access Port* (ICAP) [78].

The configuration bitstream of an FPGA can be loaded *fully*, typically in the case of the first configuration of an FPGA after its power-on or reset. It can also be loaded *partially*, in which only a specific portion of the FPGA's configuration is re-written with a prepared partial bitstream. Such an approach also lowers the time needed to transfer the configuration. This ability of an FPGA is called the *partial reconfiguration*. Certain FPGA types also allow the so-called *dynamic reconfiguration* to modify the configuration memory while the design part on the FPGA that is not being overwritten is still performing its

function. With partial dynamic reconfiguration through the internal interface, an FPGA design can modify itself from within the FPGA while still performing its function [78].

2.1.1 FPGA's Structure

A generic structure of an FPGA is constructed of the so-called *Programmable Logic Blocks* (PLBs), *Input/Output* (I/O) blocks, *Block Random Access Memories* (BRAMs), *Digital Signal Processing* (DSP) components, a set of control and management components, and programmable interconnection fabric [37, 32]. Modern FPGAs contain additional complex components, such as processors, interfaces and acceleration engines (e.g., for *Artificial Intelligence*). Together, these components are arranged in a matrix, as shown in Figure 2.1. Different manufacturers developed different structures of PLBs, such as Altera's *Logic Element* (LE) or Xilinx's *Configuration Logic Block* (CLB).

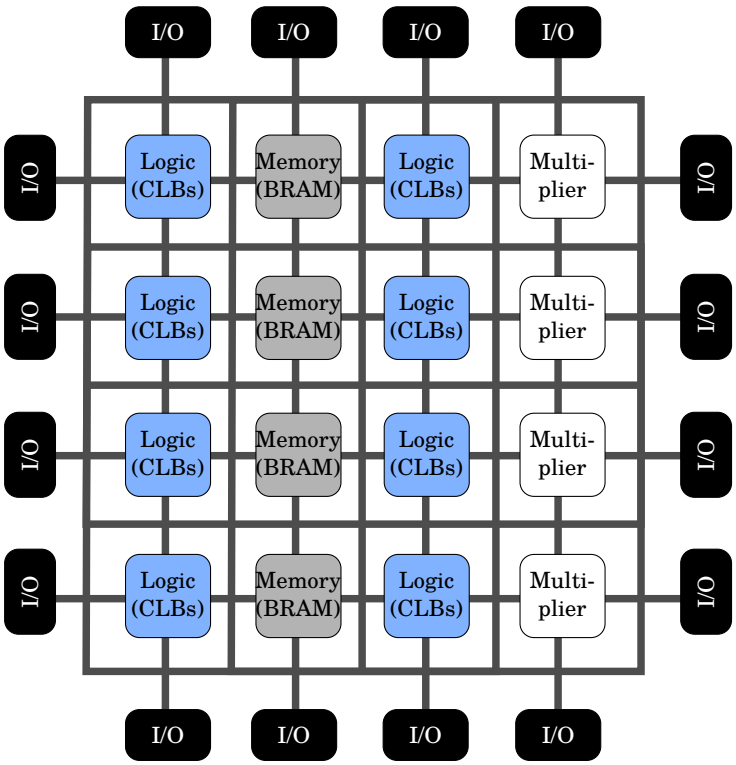

Figure 2.1: Example of a simplified FPGA structure [37].

In Xilinx FPGAs, CLBs are composed of slices containing *Look-Up Tables* (LUTs), storage elements, multiplexers, and carry logic. Specific slices might also support additional functions, such as storing data using distributed RAM and shifting data with 32-bit registers [79].

Three approaches to store the configuration are currently used: 1) *Static Random Access Memory* (SRAM), which is volatile; 2) antifuse technology, which is non-volatile and can be programmed only once; and 3) *Electrically Erasable Programmable Read-Only Memory* (EEPROM) or Flash technology, which is also non-volatile, but can be reprogrammed [32]. The research presented in this thesis primarily targets FPGAs utilizing the SRAM to store the configuration, the so-called SRAM-based FPGAs.

2.1.2 Single-Event Effects in FPGAs

FPGAs are often utilized in applications with increased demands on reliability. Examples include medical equipment for ultrasound, MRI, and CT scanners [82, 50]. Other applications are additionally required to withstand an increased occurrence of faults while still maintaining increased reliability. Aerospace scientific applications can serve as an example [19, 63] that must withstand the cosmic rays and high-energy protons [55].

In such environments, the so-called *Single-Event Effects* (SEE) can occur. A SEE is caused by a particle with high energy [55]. To explain the effect of SEE on FPGAs, an FPGA can be viewed in a double-layer model [57], which is illustrated in Figure 2.2. This model consists of 1) the user logic layer and 2) the configuration memory layer.

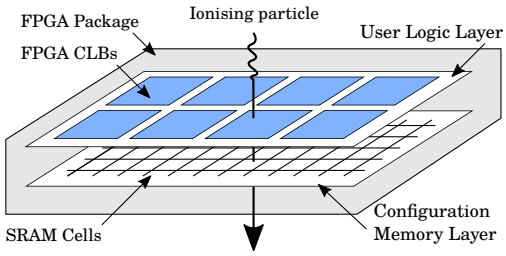

Figure 2.2: The double-layer FPGA model through which an ionizing particle is passing [57].

Both layers from the double-layer model [57] are prone to manifestation of SEEs as charged particles pass through them. On the user logic layer, a risk of *Single-Event Transients* (SETs) [74] exists. SETs introduce *transient errors*, which appear on signal levels connecting the components implemented in the user logic. These errors might eventually be propagated to the primary outputs and observed as incorrect results. If such erroneous results are not explicitly stored by the circuit implemented on the FPGA, their effects disappear as the circuit continues its work. It is, however, important to note that the design structure of the implemented user logic is not altered.

A high-energy particle that passes through the FPGA's configuration memory layer creates a risk of *Single-Event Upset* (SEU) [60]. This event results in a bit flip of a configuration bit. This phenomenon possibly results in an instantly altered design, as the configuration memory stores the implemented design. This affects LUTs contents, *Flip-flop* (FF) contents, interconnections, and also contents of the BRAM, which is graphically illustrated in Figure 2.3.

However, not each SEU (i.e., bit flip) necessarily changes the behavior of the design. This is because the affected bit might be located in a currently unused part of the configuration bitstream, or the implemented design might contain redundant components, thus effectively *masking* the SEU impacts. From the utilization perspective, the configuration bits can be classified into 1) un-used and 2) utilized in the design. Another classification is into A) insensitive bits, whose flipping is not observable on components (or FPGA's) primary outputs, and B) sensitive (i.e., critical) bits that are susceptible to the SEU phenomenon.

2.2 Hardware Description and Synthesis for FPGAs

Before a bitstream is generated for an FPGA, the target circuit must be described, synthesized and validated. For this purpose, *Hardware Description Languages* (HDLs) are mainly

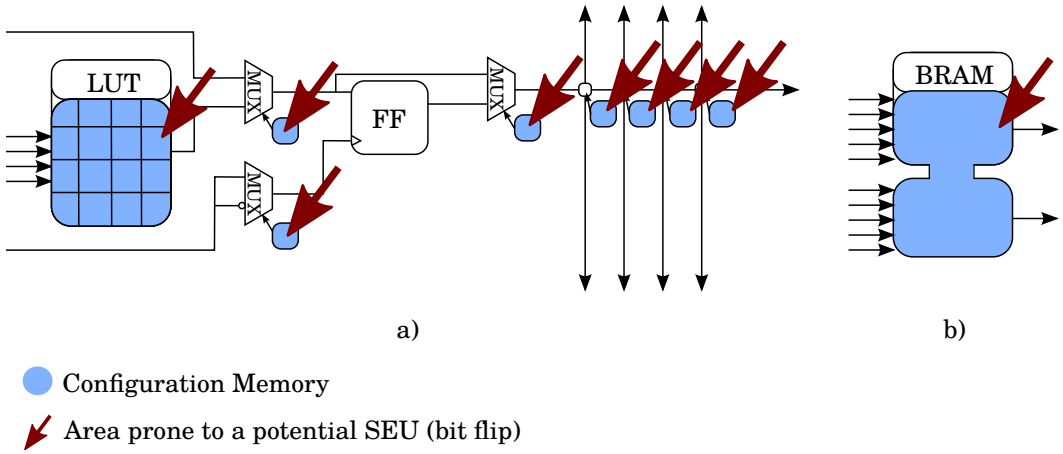

Figure 2.3: Places in the configuration bitstream in (a) CLBs and (b) BRAMs in which an SEU can affect the implemented design (slightly varies among models; illustrated for the Xilinx Virtex technology) [53].

used. Nonetheless, experimental graphical programming approaches are also available [22]. They, however, internally use textual representation in an HDL.

In recent decades, another approach, the so-called *High-Level Synthesis* (HLS), gained popularity. This is mainly because of the growing complexity of digital systems, still supported by the increasing level of integration on a chip [12]. The HLS aims at shortening the development and design process to lower the *Time to Market* and make it error-free and easier to verify [20]. HLS is enabled by describing the HW in a higher-level programming language, such as C, C++, or even Python.

2.2.1 Hardware Description Languages

Many HDLs exist; however, the most popular are the VHDL and Verilog [21]. The case study experiments presented in this research are based on the VHDL. That's why the following text will very briefly describe the VHDL principles.

VHDL combines approaches of conventional languages such as Pascal or C, logic description languages such as ABEL-HDL, netlist languages such as EDIF, and additional features for precise time modeling of events [71]. Because of the *dual nature* of VHDL, the same language can be used to describe the design and also create a testbench to simulate surrounding components. This, however, implies that circuit descriptions with correct syntax exist that can only be simulated and cannot be synthesized into an FPGA design [14].

A design in VHDL can be described as a module with data inputs and outputs. This module is called *entity* in the VHDL nomenclature. The design can be further hierarchically defined by other modules (i.e., entities) with their interconnections. This approach to circuit description is called *structural*. At the bottom of the such hierarchy, the entity must be described directly – with the *functional description* approach for simpler combinational logic or the *behavioral* approach for more complex components [5].

Firstly, a design in HDL starts with a specification, from which the circuit architecture is later constructed. After writing a module where the circuit is described, the designer checks the correctness according to the specification. This is performed on a test bench using a simulation tool. After the verification, the synthesis tool converts the design into

a logic-level *netlist* which is subsequently optimized (logic minimization, timing, etc.). After that, the place and route tool is utilized to map the synthesized logic onto available elements in the FPGA, including their routing, which utilizes the interconnection fabric switches. After the design is fully placed and routed, timing analysis is performed. The design is transformed into the configuration bitstream if the timing specification is still met. The resulting bitstream is stored and eventually written to the FPGA's configuration memory to start the FPGA's operation. Graphical representation of this process is shown in Figure 2.4 [24].

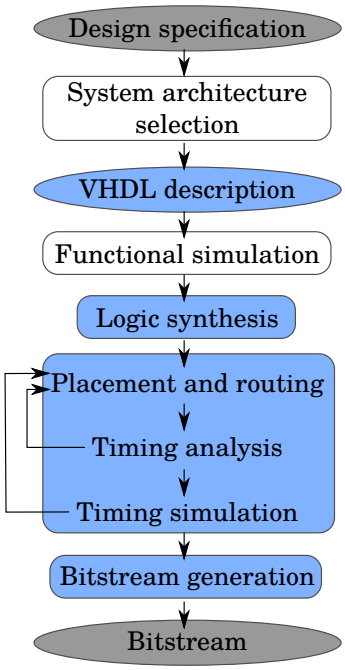

Figure 2.4: HDL design flow [24].

2.2.2 High-Level Synthesis

Many *Higher-Level Programming Languages* (HLLs) can be used to describe synthesizable hardware [52]. Currently, often the C/C++, and rarely Python, Java, or MATLAB languages are utilized for this purpose [52]. Practically, the ability to use an HLL to describe hardware depends on the existence of a synthesis design toolkit for the given HLL. Current design toolkits usually require additional information to be added to the HLL program (i.e., a description) to synthesize working designs efficiently. For example, they modify the basic data types of a description to specify their bit width [66].

Additional design speed and size optimization is achieved by applying the so-called *pipelining* and *unrolling* to specific program loops. In the pipelining process, a loop is transformed into a pipeline. A new input data set processing is started through the pipeline during each the so-called *Initiation Interval* (II). This allows starting a new loop before the previous cycle has ended. In the unrolling process, a loop can be partially or fully unrolled (the full unroll applies to static repetitions only). This results in hardware replication and decreases the number of program cycles needed to execute a loop [20].

The HLS design toolkit transforms the description into an HDL (such approaches then partly utilize the traditional HDL design synthesis flow) or (rarely) into a bitstream. As

the support of the C/C++ language is very common in HLS, the thesis also partly focuses on C++ in combination with HLS [52].

The HLS design flow begins with a circuit described in HLL. This time, the description is compiled at first. During this compilation, the *Control-Data Flow Graph* (CDFG) model is obtained, which represents the data and control dependencies. Then, the steps of allocation, binding, and scheduling are carried out. These steps are very tightly entangled. During their execution the appropriate hardware resources are allocated, their operation is scheduled, and finally assigned to available hardware resources. As for the latter, the operations are assigned to functional units, while variables to storage elements and data transfers to buses. Finally, the *Register-Transfer Level* (RTL) representation is generated. This HLS design flow is depicted in Figure 2.5 [16].

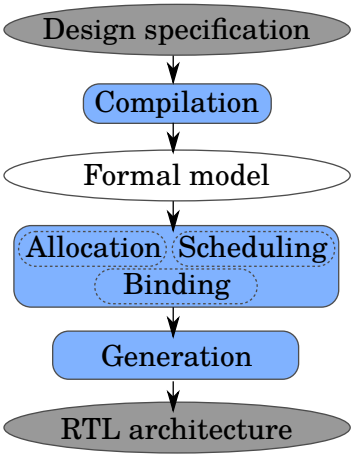

Figure 2.5: HLS design flow [16].

2.3 Dependable Systems Design

Specific electronic systems are required to keep their failure rate at a minimum. There are many reasons why designers must design these systems with dependability in mind. The first is the risk of endangering human health and huge financial or tangible and intangible losses. The economic losses are also the result of a failure of a system whose reparation is not possible or very difficult to ensure. This is related to space probes especially. For example, the loss of the on-board computer system of a space probe was involved in circa 6 % of failed investigations launched towards planet Mars between 1960 and 2020 [48]. Such a failure is very costly, because the mission prices are in the magnitude of billions US Dollars. For example, the costs of development and launch of the current NASA's Perseverance Mars rovers are circa 2.45 billion US Dollars [73]. A premature failure of such a probe is not only an economic loss but also poses unnecessary delays in scientific progress.

Generally, a system is considered dependable if it allows performing its function as required and when required; dependability can be used as a term for a time-related quality characteristic of a system. The dependability property can be divided into availability, reliability, recoverability, maintainability, and maintenance support performance [30].

Definition 1. *Dependability of an item is the ability to perform as and when required. Dependability includes availability, reliability, recoverability, maintainability, and mainte-*

nance support performance, and, in some cases, other characteristics such as durability, safety and security. Dependability is used as a collective term for the time-related quality characteristics of an item [30].

In this thesis, the focus is directed toward reliability and availability. Reliability is generally the ability to perform the function as required and without failure for a given time interval and given conditions.

Definition 2. *Reliability of an item is the ability to perform as required, without failure, for a given time interval, under given conditions. The time interval duration can be expressed in units appropriate to the item concerned, e.g. calendar time, operating cycles, distance run, etc., and the units should always be clearly stated. Given conditions include aspects that affect reliability, such as: mode of operation, stress levels, environmental conditions, and maintenance [30].*

Reliability can be quantified using various metrics, from which the most frequently used ones in this thesis are *Time to Failure* (TTF), *Mean Time to Failure* (MTTF) [30] and *Median Time to Failure* (a.k.a. t50) [72]. The t50 parameter defines the time in which the probability of the fully functioning system is equal to 50 %.

Availability expresses the fraction of time a system can operate according to the requirement [67].

Definition 3. *Availability of an item is the ability to be in a state to perform as required. Availability depends upon the combined characteristics of the reliability, recoverability, and maintainability of the item, and the maintenance support performance [30].*

The demand for designing a system that will perform the given function under stated conditions and period is called the *System Reliability Problem*. Maintaining the functionality requires increasing the reliability of the system. Two main approaches to increase reliability exist: 1) *Fault Avoidance* (FA) and 2) *Fault Tolerance* (FT); a combination of these approaches is usually required in practice, as the FA approach alone is not enough for the highest reliability requirements [11].

2.3.1 Fault Avoidance

The *Fault Avoidance* approach targets to *avoid* faults. This is achieved through the design of components for a long-lasting life, strict manufacturing standards, and thorough quality testing after manufacture [11]. Thus, compared to the standard design, the FA approach differs in selecting highly-reliable components. However, the highly-reliable components are usually more expensive, and their parameters are worse compared to the currently available top consumer-grade parts [40]. When using this approach alone, it is unnecessary to alter the design. Only the parts have to be properly selected.

2.3.2 Fault Tolerance

Fault Tolerance does not avoid the emergence of faults. It masks them instead. After a potential fault occurs, it is masked. Thus, the fault's consequences are not observable through the system outputs, and the system functions according to the specification. The number of faults the FT approach withstands is usually limited.

Definition 4. *A system is considered fault-tolerant if, even during the presence of faults, all of the following conditions are met:*

- *data processing of the system is not stopped nor altered due to a fault,*
- *the results provided by the system are correct,*
- *the results provided by the system are obtained at the specified time [28].*

FT is achieved through the systematic incorporation of special structures into the design. This involves *redundancy*. Generally, three types of redundancy can be used to achieve FT: 1) *spatial*, 2) *temporal*, and 3) *information* (a.k.a. *data*) [67].

Spatial Redundancy

The spatial redundancy uses additional components to introduce redundant structures to the system. During this, the designer has to ensure that the added redundancy effectively increases the system's fault tolerance. One of the most common types of FT utilizing spatial redundancy is the *Triple Modular Redundancy* (TMR) [47]. Graphical representation of the TMR structure for a component is in Figure 2.6. The TMR can be applied to a complete system (or a subsystem); such an approach is called the *Coarse-grained TMR* (CGTMR). If the TMR is applied to each component of a system (or a subsystem) individually, the resulting architecture is called the *Fine-grained TMR* (FGTMR). A TMR architecture can be generalized into the *n-modular* redundancy.

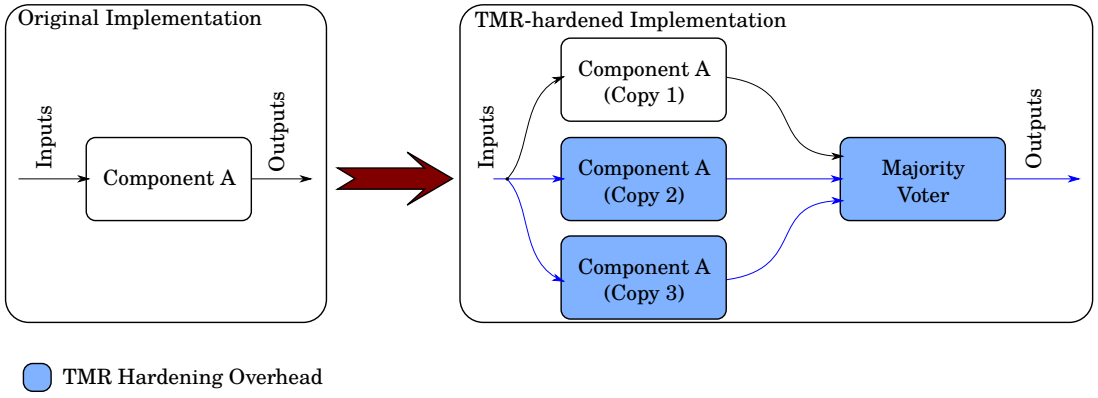

Figure 2.6: Graphical representation of an introduction of TMR mechanisms for a component.

To achieve better results, especially for longer mission times, the spatial redundancy can be improved by using the so-called *restore mechanisms* (a.k.a. *repair* or *renew*). For example, on FPGAs, the introduction of restore mechanisms eliminates the accumulation of faults in the bitstream, which is a serious problem. The partial reconfiguration property of an FPGA can be used to rewrite and repair the bitstream, which is called configuration memory scrubbing [26]. The reconfiguration can be performed from within the FPGA design by using the *Dynamic Partial Reconfiguration* (DPR), for which a reconfiguration controller is used. The reconfiguration controller that was designed and developed previously in our research group is called the *Generic Partial Dynamic Reconfiguration Controller* (GPDRC) [69].

Temporal Redundancy

The temporal redundancy approach targets the repetition of a computation. The main idea is that a transient fault disappears after a certain time. Thus, making repetitive equivalent computations might provide enough results to detect the correct ones (i.e., to dismiss the erroneous results calculated while a transient fault was present in the system) [6].

Information Redundancy

The information (a.k.a. data) redundancy for FT improvement uses added information, which is kept in the system. Error-detection codes can detect faults, while error-correction codes provide the ability to correct an erroneous result eventually [67].

2.3.3 Calculation with Reliability Metrics

The *Mean Time to Failure* (MTTF) is the most used reliability metric in this thesis. For a system, it expresses the mean value of time in which the system fails. It is a statistical value and, thus, it should be calculated from a set of *Time to Failure* (TTF) measurements of a representative size [30].

Another reliability metric is the *failure rate*, denoted as λ. It expresses the probability of failure. A typical failure rate forms the so-called *bathtub curve* during a lifetime of a system, which is depicted in Figure 2.7 [28].

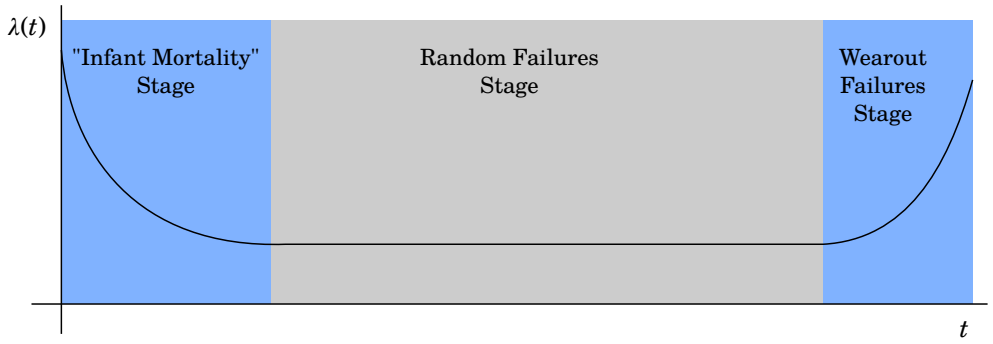

Figure 2.7: The failure rate λ as a function of time t; it forms the so-called bathtub curve [28].

The failure rate of a complete system can be approximated by the sum of failure rates of individual components as shown in Equation 2.1.

$$\lambda_{sys} = \sum_{\forall c \in C} \lambda_c \tag{2.1}$$

It is also possible to convert the mentioned MTTF metric to the failure rate using Equation 2.2, which will be extensively used in the presented research.

$$MTTF = \frac{1}{\lambda} \tag{2.2}$$

2.4 Computer-Aided Design of Fault-Tolerant Systems

As the complexity of digital systems grows, new approaches have to be developed to increase design productivity. The same applies to the design of FT systems. A general approach has been established in past decades [36], [4]: the system is designed as *unhardened* (i.e., not incorporating any of the FT mechanisms), and the computer-aided design methods are employed to incorporate FT mechanisms to the design.

Generally, the incorporation of FT into a system consists of three steps: 1) a design modification enabling to incorporate the FT; 2) a selection of the most suitable FT approach for a given partition of the system; and 3) measurement of the achieved design parameters (reliability parameters, area overhead, power consumption etc.).

2.4.1 Modifying the Code to Support FT

Throughout the literature (e.g., [36, 76, 4]), two general approaches to computer-aided insertion of FT structures into digital systems can be identified. It is possible to distinguish them as 1) *modified synthesis* and 2) *modified description*.

In the case of the modified synthesis approach, the synthesis mechanisms themselves are modified to produce FT systems on the output, although a description of an unhardened system is provided on their input. An example of such an approach is the *TLegUp* [41], which is an extended version of the *LegUp* open-source HLS tool, targeting the C language [10]. The flow of utilizing the modified synthesis approach (e.g., the TLegUp) is in Figure 2.8.

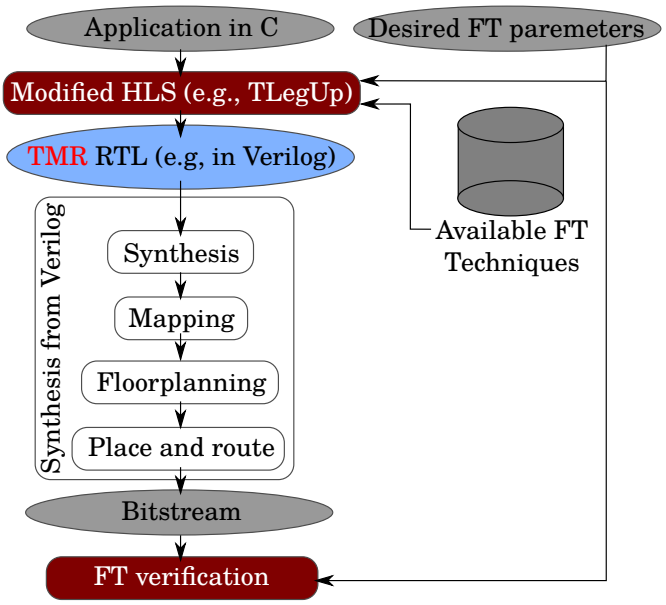

Figure 2.8: TLegUp design flow (i.e., the modified synthesis approach to automated FT insertion; simplified) [1].

The advantage of this approach is a direct access to the synthesis's internal representation of the design. Moreover, the higher control over the optimization processes inside the synthesis tool is a plus. This is because the optimization process can be modified to avoid removing the introduced FT mechanisms. However, if a designer decides to extend a synthesis tool to incorporate FT, he or she quickly encounters a fundamental requirement of an access to the source code of the synthesis tool software. This is an essential disadvantage

for experimental research and development. However, it can be overcome by focusing on open-source synthesis tools.

The modified description approach instead keeps the components of the design flow intact; however, it adds a new element to the design flow. An example of such a design flow is in Figure 2.9, in which the added element is highlighted by a red color. An example of such an approach can be the commercially-available Xilinx TMRTool [76], which modifies the synthesized design during the design process in the Xilinx ISE flow. Another example is the *BYU-LANL TMR Tool* (BL-TMR) [9]. Similarly, the TMRG [36] modifies the structures directly in the Verilog language and outputs the Verilog code with FT structures incorporated. All these mentioned tools focus exclusively on TMR.

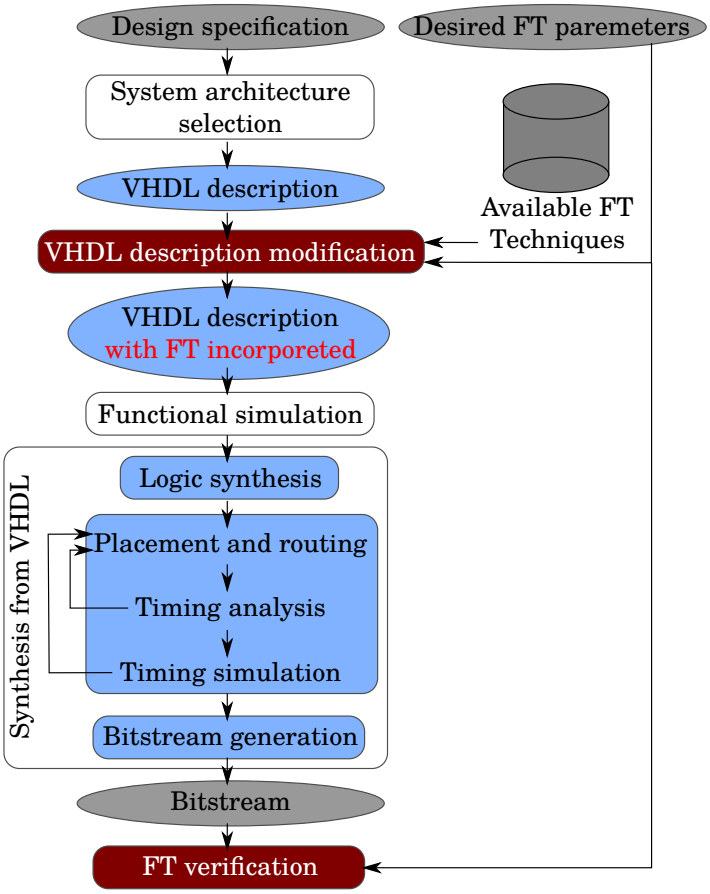

Figure 2.9: The modified description approach to the automated FT insertion utilized in the Xilinx ISE design flow [24].

The method of modifying the description has one advantage. The method's developer can treat the design flow steps as black boxes. It is, however, essential to ensure the added redundancy does not interfere with the optimization processes running the synthesis tool. In this thesis, the method of modified description is used, as one of the goals is to investigate the possibilities of supporting multiple description languages and abstraction levels, which is straightforward to implement if the approach of modified description is used.

2.4.2 Strategy for Selection of FT Mechanisms

A system is usually composed of multiple components, each of which might have different requirements on FT. The suitability of a given FT mechanism is expressed as the gained reliability improvement compared to the overhead associated with the application of the FT mechanism. The problem of selecting the best FT mechanism for each component is called the *redundancy allocation problem* throughout the literature [42, 83]. This is a hard optimization problem [13] which is solved by design space exploration methods, for example, evolutionary algorithms [35].

In [31] and [75], the genetic algorithm to allocate the redundancy, particularly the *Non-dominated Sorting Genetic Algorithm II* (NSGA-II) is used. The method utilizing the *Improved Surrogate Constraint* approach is proposed in [56], targeting minimization of the method's computation requirements. Other approaches include penalty-guided artificial bee colony algorithm [83], variable neighborhood search meta-heuristic method [42], and particle swarm optimization algorithm [34]. In [4] the BL-TMR FT mechanisms insertion tool is extended with design space exploration and parameter optimization.

2.4.3 Fault Tolerance Evaluation

A candidate redundancy allocation must be evaluated during the FT mechanism insertion to obtain feedback for the design space exploration algorithm. One possibility is to utilize the so-called *functional verification*, the aim of which is to run the system with faults and, through examination of its primary outputs, examine the *observability* of a fault on the primary outputs. Simply described, if a fault causes the system to produce incorrect results or alters the system's timing, such a system is not tolerant to the given fault.

One possibility to perform functional verification is to simulate the design alongside simulated faults [8, 54]. However, this approach requires a lot of computation power. Nonetheless, the simulation allows practically limitless possibilities to simulate fault types and their unconstrained timing [7]. The accuracy, however, depends on the level of the simulation. Another possibility for functional verification of FT properties is the execution of the system on the actual hardware (i.e., FPGA in our case). However, in a standard environment, waiting for faults to naturally appear in an FPGA during the testing procedure is impossible because of the enormous time demands of such an approach. Thus, the fault occurrence must be artificially increased to test and examine the candidate solution in the presence of faults. Such an artificial manifestation of faults is called the *fault injection* (a.k.a. *SEU injection*) [70].

An important advantage in design test and evaluation is if the testing method does not require modifying the design. Thus, the design under test is equivalent to one later deployed. This requirement is met by Alderighi et al. [3, 2]. They inject faults into real hardware utilizing an FPGA. Similarly, in [2], a platform called *FLIPPER* is presented, which utilizes two FPGAs, and the test is performed on the target hardware. One FPGA board controls the experiment process, and the other contains the design under test. Contrasted to FLIPPER, [46] presents an approach in which the fault injection is performed using circuits designed inside the same FPGA. In [65], the method utilizing the RapidSmith library [39] is presented, which is then further demonstrated in [38]. Liu et al. [43] propose a method for fault injection, which can modify and observe the values of signals in a design. The communication is performed through the JTAG interface. In [64], multiple fault models are supported. However, the disadvantage of this approach is that the design must be modified to inject faults into it. Benso et al. [7] propose the approach of fault

modeling combined with design simulation. Similarly, in [8, 54], evaluation platform toolkits are presented that also utilize simulation. And finally, in [61], an evaluation platform is presented, which was designed, and developed previously in our research group. Part of the platform runs on a personal computer, while the design under test is executed on the target hardware. This platform targets final design testing while monitoring impacts on the mechanical parts controlled by the design under test. However, it has to be noted that massive acceleration of the evaluation is needed, which is not supported by this platform, to conduct research on automated FT design.

2.5 Open Problems

Although various approaches to automated design of FT systems exist in the literature, they always target a specific type of technology [76], specific language [36], or a given level of abstraction [41] of the description. The current state of the research in the field of FT system design automation brings the following questions:

- According to the author's knowledge, the effort to solve the problem of automated FT system design **in a general way** has not been addressed in the literature. Would it be possible to propose a uniform approach to frame the automated design flow? If so, what would this approach look like?
- Is it possible to obtain the solution(s) that is (are) optimal on selected parameters (e.g., size, power consumption and reliability) automatically (in a finite and reasonable time)?
- As the measurement of candidate solutions seems to be the most time-consuming part of the design, is it possible to accelerate the evaluation?
- What is the best trade-off between the number of FT measurements that have to be accomplished and the quality of FT analysis?

These are the questions that appear behind the research presented in this thesis. And this thesis tries to answer them through experimental work. It also tries to prove the possibilities by designing and implementing the method in the form of a toolkit, as presented in the following chapter.

Chapter 3

Research Summary

This chapter describes the methods used to solve the overall thesis goal. The main part focuses on a summary of the most important results achieved during the research of this thesis goal.

3.1 Methodology

This thesis aims to design, implement and present a method for the automated transformation of an unhardened design to its hardened version. The newly-emerging method should be, however, as general as possible. In other words, it should allow working with various description languages and levels of abstraction. Moreover, it should be possible to use the method for different FPGA technologies. For this reason, the main part of this method was decided to work on the level of description code. To work on the description code level, it is, however, essential to design and implement means for introducing FT mechanisms into the code. That's why in the first stage of research, the means to introduce FT mechanisms into existing code will be designed and implemented as a library and (or) a tool. C++ language support, in combination with the HLS, was chosen for this. These mechanisms will be thoroughly evaluated and tested to show their feasibility. Later in this thesis, the introduction of FT mechanisms into the VHDL code will also be designed and implemented.

Task 1. It will be necessary to design and implement the means for introducing FT mechanisms into the already-existing description. The C++ and the VHDL languages were selected as two representative languages.

A selection strategy will be developed to select the best FT mechanism from the available FT mechanisms. These FT selection strategies implement a way to assign the best FT mechanism to each part of the description code. We cannot talk about components here, as, for example, the C++ language does not describe any components, as opposed to VHDL. Such a design separates the insertion of FT mechanisms (i.e., description code specifics) from their selection (i.e., reliability and design space exploration specifics). This separation aims the ability to extend the set of supported languages easily. A new language can be supported just by providing a new language manipulating module into the design flow while completely re-using the existing selection strategy methods.

Task 2. Finding and implementing methods for algorithmic selection of the FT mechanisms per each part of the description code will be necessary.

The current approaches to evaluate the level of FT are not prepared for the automated usage. Their configuration is not straightforward. Moreover, the evaluation is the most time-consuming part of the automated FT design, according to our first preliminary experiments. For this reason, a new approach that will use all the available acceleration means must be designed and implemented as a framework. All the technology-specific parts of this framework (such as the communication interface) will be isolated into specific components to allow straightforward porting of this framework to different FPGA technologies of various manufacturers.

Task 3. A new approach to FT evaluation will be designed and implemented, which allows a straightforward configuration and utilization of several techniques to accelerate the evaluation.

As a final stage of the research, all the previously mentioned means will be merged into a design flow. Two design flow types will be presented (i.e., the so-called component-based flow and system-based flow), and the user's task will be to select the most appropriate one based on the structure of the description code. This is because certain description languages do not allow splitting the design into components (e.g., C++). Other languages (e.g., VHDL) might allow the separation of the system into components. However, a low level of code writing discipline might effectively disable the user from separating the system into individual components.

By proposing the complete design flows, the overall goal of this thesis – a general way of an automated method for FT systems – will be fulfilled.

Task 4. A new design flow for the automated design of FT systems will be composed of the previously mentioned modules and tools.

3.2 Fault Tolerance Evaluation Approaches

In this thesis, two design evaluation approaches will be used. In the first research stage, an external platform for evaluating FT properties will be used (Section 3.2.1). This external platform targets the evaluation of faults that impact the electro-mechanical systems. It must simulate the environment for the design under test; thus, the space for acceleration techniques is limited. Its purpose was to enable the author to start the research and to present the first results and feasibility of the FT mechanisms insertion methods.

Later in this thesis, the newly created framework for accelerated evaluation will be presented. This framework does not allow for the simulation of a real-time environment of the tested circuit; it, however, utilizes several acceleration techniques. One of the essential acceleration techniques is the autonomous execution of the evaluation and parallel evaluation of multiple design instances on one FPGA. This framework is introduced as part of the thesis research and, thus, will be presented later with the other results (Section 3.3.3).

3.2.1 FT Evaluation Platform utilizing Robot Controller

This evaluation platform was designed by J. Podivínský [62]. The evaluation platform includes a prepared maze simulation environment and a reference implementation of a robot controller in the VHDL language. We are primarily interested in the robot's controller implemented in an FPGA. The specific implementation of the robot controller uses the so-called left-hand algorithm, i.e., the robot chooses the option to turn left at every opportunity

to turn in the corridors. However, for the goal to be accessible to the robot, it is necessary to follow specific rules during the creation of the maze [62].

The evaluation platform uses a robot controller to represent the design under test. The robot controller is part of a simulated robot aiming to search a path through a maze. The maze's structure is not known in advance to the robot controller – the robot must utilize its simulated sensors to detect obstacles and its simulated motors to navigate the maze in real time. The evaluation platform functionality is based on the so-called *fault injection and functional verification*. This approach executes the design under test and its reference (i.e., *golden*) model. An artificial fault is injected into the design under test. At the same time, its output data is monitored and verified, whether it matches the output data of the reference model. The tested design must be provided with input data. These are provided by virtual sensors that are part of a computer simulation.

The environment is divided into two parts. The first part, running on a PC, cares about starting the experiment, its evaluation (data verification), and the simulation of the design environment under test. The second part is executed directly on the target FPGA. Both of these parts are connected using Fast Ethernet and JTAG interfaces. Fault injection into the design under test uses DPR so that the design can be functional during the injection (which is suitable for fault injection during the design run). The Ethernet interface is used to monitor the experimental system. The scheme of the evaluation platform is shown in Figure 3.1.

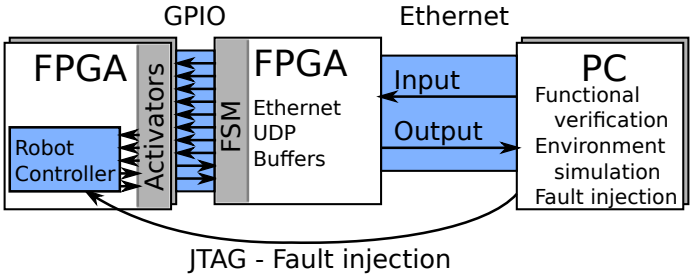

Figure 3.1: The evaluation platform which will be used in the first experiments with FT insertion methods [62].

During one verification run, the expected data is compared with the data received from the design under the test. The desired result is computed in the reference model on the personal computer. After the first difference between the tested system and the golden model is observed, the time of this failure is recorded. This is marked as an „electronic failure." Still, the experiment continues to determine whether such a failure also results in a mechanical manifestation (depending on the tested application, e.g., impact, jam, finding a target by a different path, etc.).

After verification runs with the evaluation platform are finished, the user obtains a list of faults injected in each verification run. It also includes the evaluated system's behavior under the presence of faults. It separately monitors the precise correctness of the data received from the robot controller on the FPGA (called the *electronic point of view*; these are the results we are interested in this thesis mainly). It also verifies whether the simulated robot achieved the desired target in the simulated maze (i.e., from the *mechanical point of view*). Note that the robot can fulfill its mission from the mechanical point of view even if some incorrect data are observed from the electronic point of view.

3.2.2 FT Evaluation Framework

The approaches to FT evaluation currently available to us are not prepared for usage with automated design flow. Firstly, the evaluation itself must be automated as much as possible. It must minimize the user interactions needed during the evaluation. Also, current approaches do not fully utilize the potential of acceleration techniques. For this reason, a new approach will be presented later in this thesis to improve the critical properties needed in the automated design flow. Because the Evaluation Framework (described in Paper D) is a part of the proposed FT system design automation method, it will be presented with the other author's results in Section 3.3.3.

3.3 Research and Results

As the research of FT system design automation covers multiple areas, the results are divided into 1) FT mechanisms insertion, 2) selection strategy for FT mechanisms, and 3) FT evaluation. These areas were, however, researched in such an order that allowed the author to systematically and incrementally verify the method proposed in this thesis. For this reason, the conducted research is presented chronologically. The summary of the proposed design toolkit is communicated in separate Section 3.4. The research presented in this chapter is supported by eight publications, which represent the core of the research.

3.3.1 FT Mechanisms Insertion into the C++ Source Code: First Experiments

The following section presents the functionality of the approach to modify a description code in combination with the selected HLS tool. And also to prove that it is beneficial to choose the appropriate FT mechanism for each component (of a system) individually, which is one of the key ideas proposed in this thesis for the automated design of FT systems.

Research Steps

- Choose one of the design synthesis flows for FPGAs and prepare an experimental system on which the FT insertion mechanisms will be tested.
- Design and implement a method for automated insertion of redundant structures implementing FT into a design.
- Perform the synthesis and evaluate how the reliability achieved through the FT mechanism insertion depends on various synthesis settings.
- Divide the experimental design into smaller parts and evaluate the effect of FT insertion on individual parts separately.

Design Flow for the First Experiments For initial experiments, it is necessary to start with the choice of synthesis flow. Next, we will incorporate mechanisms into this flow that modify the design description so that the resulting system is FT. The HLS approach was chosen for the initial research. The synthesis is performed for the description in the C++ language, and the resulting circuit is then described in the VHDL language at the RTL level. The specific tool used in this research is the Catapult C University Version (UV) 8.2b [51]. The design flow then looks as shown in Figure 3.2.

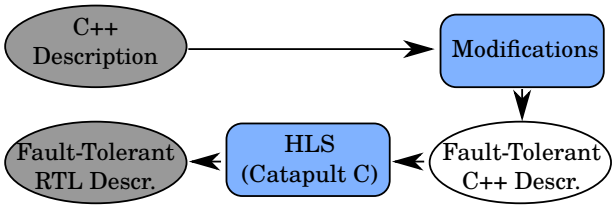

Figure 3.2: The selected flow utilized for the FT insertion in combination with the selected Catapult C HLS.

Principles of Redundant Data Types In the C++ language description, three targets of FT mechanism insertion can be generally distinguished: 1) data storage elements (variables); 2) operations with data (arithmetic and logic operations); and 3) flow control statements (e.g., if – then – else). The proposed method is based on modifying language data types, thus the name *Redundant Data Types* (RDTs). The RDTs target FT insertion into the storage elements and data operations. The concept of RDTs is very similar to the Algorithmic C Data Types (AC Datatypes) [66], which are used specifically with HLS to specify the bit width of synthesized variables and operations. In this case, the principle is used to specify an FT mechanism on a per-variable basis. Specified FT mechanisms are inserted into the data path corresponding to hardened variables during the synthesis. It is important to highlight that it allows targeting FT mechanisms into specific parts of the code. The simplified example of usage is shown in Figure 3.3.

	Original code	Modified code	Preprocessed result (semantically)
1	`int a;`	`triple<int> a;`	`int a_x, a_y, a_z;`
2	`int b;`	`triple<int> b;`	`int b_x, b_y, b_z;`
3	`int c;`	`triple<int> c;`	`int c_x, c_y, c_z;`
4	`b = 7;`	`b = 7;`	`b_x = 7; b_y = 7; b_z = 7;`
5	`c = 8;`	`c = 8;`	`c_x = 8; c_y = 8; c_z = 8;`

6	`a = b + c;`	`a = b + c;`	`a_x = b_x + c_x;`
			`a_y = b_y + c_y;`
			`a_z = b_z + c_z;`
			`vote(&a_x, &a_y, &a_z);`

7	`/* a = 15 */`	`/* a = 15 */`	`/* a_x, a_y and a_z = 15 */`

Figure 3.3: An example of a C++ program code (a) before and (b) after the incorporation of RDTs and (c) semantic meaning after preprocessing.

The RDT is seen as a standard data type in the C++ language. For the implementation of new data types into the C++ language, a unique approach called the *C++ templates* is used. Each RDT then represents one FT mechanism (e.g., TMR), while the basic standard C++ data types can be considered *simplex*. If we take the TMR as an example, each RDT triplicates the associated data path and storage elements and adds voters after each operation with the data type. This is achieved as follows. The variables in the code are triplicated. Each operation (and its associated operator) can be identified as 1) *unary* (e.g., the ++ operator), 2) *binary* (e.g., the + operator), or 3) *ternary* (three operands; for certain cases used to shorten the if – then – else statement). For the unary operators, the operation is triplicated, and a voter method (in the terminology of C++) is added. For

the binary operators, collision of RDTs of the same or different redundancy types might occur: (a) *intra-datatype*: two RDTs of the same redundancy type (e.g., TMR RDT + TMR RDT); (b) *inter-datatype*: two RDTs of different redundancy types (e.g., TMR RDT + duplex RDT); and (c) with *original datatype*: RDT and a standard data type (e.g., TMR RDT + simplex). Example of these interconnections is shown in the original Paper A in Figure 1. 3) For the ternary operator, adding a possibility to cast the stored value to a Boolean value ensures the possibility to *decide* the ternary operator.

The Impact of the Synthesis Settings At first, an object of the case study must be selected. Because the evaluation platform [62] will be used in this preliminary stage of experiments, the object of the case study will be the robot controller on an FPGA. This evaluation platform was described previously in Section 3.2.1. A new robot controller was implemented using HLS, which incorporates the FT mechanisms inserted using the proposed approach. The choice of the robot system mainly depends on the chosen evaluation platform.

In the first stage, we monitor the impacts of the HLS pipelining and unrolling techniques on the resources consumed, which were briefly described in Section 2.2.2. The pipelining and unrolling techniques are part of the Catapult C [51] HLS tool. The optimization impacts are monitored for the new robot controller system, which is re-implemented using C++ but still uses the same navigation algorithm (i.e., the left-hand algorithm). Two versions of the implementation are prepared: one in which each variable remained intact (i.e., the input design as a reference) and the identical robot system, in which all variables were transformed to the TMR RDT (i.e., triplication of data paths). The results were synthesized using the Catapult C University Version [51] into the VHDL RTL and subsequently into a bitstream using the Xilinx ISE 14.7 [80] for the Virtex 5 [79]. A comparison of consumed resources for the Virtex 5 FPGA is in Figure 3.4. The first two *noopt* variants have neither pipelining nor unrolling applied. For the *noopt-area*, the synthesis goal was set to the area minimization, while the *noopt-latency* has latency minimized. The *pipeline1-area* has the main function pipelined with the initiation interval set to 1 with the minimized area; the *unroll2-area* has applied unrolling to the main function with the level of parallel execution set to 2 and area minimized.

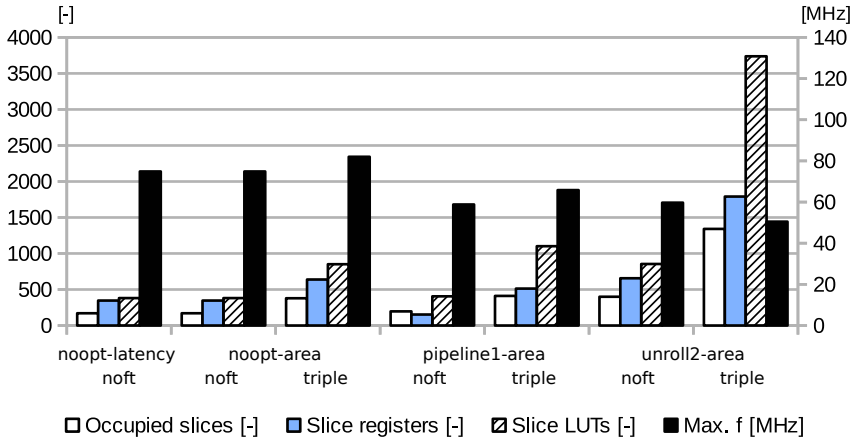

Figure 3.4: Comparison of resources consumed for the HLS-generated robot controllers.

Figure 3.4 shows that the pipelining and unrolling do not significantly affect the resources consumed compared to the simplex versions of systems. After the application of the triplication RDT, the *noopt* and *pipelined* designs consume $2.2 - 2.7$ times more LUTs. The design with *unrolling* applied stands out in this trend, and the triplication consumes 4.3 times more LUTs. This is because the unrolling executes a loop in parallel. More resources are required, which deepens even more with the triplication applied.

Lowering the number of failed runs was also examined for the combination of the triplication RDT and optimization settings. The evaluation platform was used in this experiment, which was briefly described in Section 3.2.1. The *noopt-latency* was discarded from the next experiments, as the latency goal did not affect the synthesized design. Figure 3.5 shows severe impacts on the signal correctness and timing (even a correctly found target is considered a failure here if the path and timing were different). Note that 1000 verification runs were performed for each version of the robot controller. In each of them, a single SEU fault was injected into the utilized parts of LUTs of the design under test before the robot is instructed to start its movement.

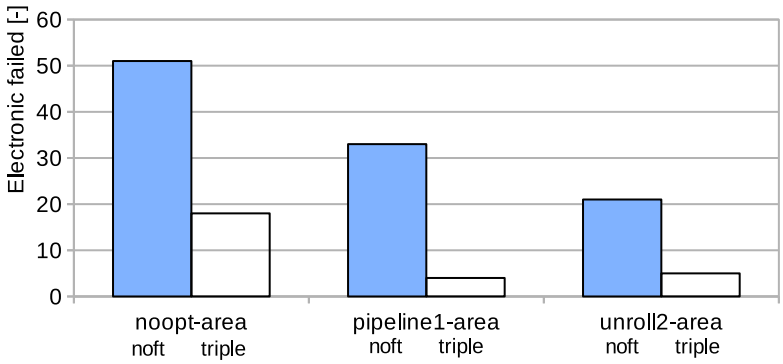

Figure 3.5: The number of failing runs (out of 1000) of the system for each FT implementation.

The most significant improvement in minimizing the number of failed runs is achieved for the pipelined design (87.9 %), followed by the unrolled design (76.2 %) and noopt-area (64.7 %). These experiments show that the RDT method can effectively improve the reliability of a system.

Hardening of System Components Another critical assumption to the FT system design automation is that one best hardened configuration can be identified for each component (which is a part of the description in the C++ source code). In other words, each allocation of FT mechanism on a component (part of description code) provides a different improvement in FT parameters for a different price (e.g., size, power consumption, etc.). This is examined in the following text. At first, the robot controller system was modeled in the *Universal Modeling Language* (UML) *Activity Diagram* (AD) to visualize the parts of the source code that are relevant. The UML AD of the robot controller can be found in the original Paper B in Figure 4. Seven blocks in the AD are identified, which can be considered FT relevant parts of the robot controller.

Each of these parts is identified by a number. Seven versions of a robot controller were prepared, in which always precisely one part was hardened using the TMR RDT (i.e., RDT triple). HLS at this time is set to the pipelining with an initiation interval of 1. The

reason is that this optimization setting showed the highest percentage decrease of failed runs in the previous experiment when the RDT is applied to the robot controller. This time, 2000 verification runs were performed for each version of the experimental system. During one test (i.e., verification scenario), one artificial SEU is injected before the robot starts. In our case, evaluation was performed on an FPGA from Xilinx utilizing the Virtex 5 technology [79]. The FPGA is located on the ML506 evaluation board [77]. The results are summarized in Table 3.1.

Table 3.1: The evaluation of resources overhead, and the improvement in lowering the number of failed runs compared to the unhardened reference values (of an unmodified design); an extended version of the table is in Paper B in Table 1.

Robot Version	Ref.	1	2	3	4	5	6	7
LUTs bits [–]	21952	17408	55744	12800	15744	47552	15872	35840
Slices [–]	196	147	370	135	165	379	147	250
Failures [‰]	33.0‰	27.0‰	13.5‰	30.5‰	37.5‰	15.5‰	29.5‰	17.0‰
Improvement in lowering failed runs number [%]	–	18.2%	59.1%	7.6%	-13.6%	53.0%	10.6%	48.5%
Area overhead [%]	–	-25.0%	88.8%	-31.1%	-15.8%	93.4%	-25.0%	27.6%

Table 3.1 shows that designs with higher overhead (2, 5, and 7) decreased the number of failed runs. The remaining designs (1, 3, 4, and 6) are smaller than the reference design. However, designs 1, 3, and 6 show lowered number of failed runs. This is due to the use of various pipelining block sizes, which the synthesis selects. Also, as the operations are associated with variables of different bit widths, the sizes of the resulting design are not directly proportional to the number of operations hardened, as shown in Table 3.1.

Key Publications

(A) Data Types and Operations Modifications: a Practical Approach to Fault Tolerance in HLS, LOJDA Jakub, PODIVÍNSKÝ Jakub, KOTÁSEK Zdeněk, KRČMA Martin. In: *Proceedings of IEEE East-West Design & Test Symposium.* Novi Sad: IEEE Computer Society, 2017, pp. 273-278. ISBN 978-1-5386-3299-4.

(B) Redundant Data Types and Operations in HLS and their Use for a Robot Controller Unit Fault Tolerance Evaluation, LOJDA Jakub, PODIVÍNSKÝ Jakub, KOTÁSEK Zdeněk. In: *Proceedings of IEEE East-West Design & Test Symposium.* Novi Sad: IEEE Computer Society, 2017, pp. 359-364. ISBN 978-1-5386-3299-4.

3.3.2 FT Mechanisms Insertion into the C++ Source Code: Multiple FT Mechanisms

The next step in this research is to verify the essential parameters of the proposed design automation method, which were previously chosen only empirically. For example, what is the impact of using the word-based voter on reliability parameters of the resulting circuit? Or is the accuracy of measured reliability parameters sufficient? These questions are answered in the experimental work presented in the following section.

Research Steps

- Verify the functionality of the RDT in the case of multiple SEUs and with a fair measurement reflecting the design size.
- Extend the library of RDTs (i.e., FT mechanisms for C++ algorithms).
- Examine the effect of the type of voter (bit- vs. word-based).
- Examine the precision of the reliability measurement.

Multiple SEUs for a Fair Measurement To design a practically usable FT system, the added redundancy must be efficiently used to increase its reliability. A question of how to measure this property emerges. As charged particles cause SEUs, it is essential to assume that the whole bitstream is subject to the manifestation of SEUs. The rate of particles achieving the FPGA's surface is usually given in particles/cm^2/s, which is called the *radiation flux* [68]. This demonstrates the necessity of considering the design size throughout the reliability measurement. To illustrate this on a hypothetical example, a *dead area* (i.e., component performing a dummy computation) added to a system efficiently decreases the percentage of critical bits. However, the system's reliability is not enhanced in such cases because the absolute number of critical bits remains the same; only the circuit size increases. To address this, the unit of *fault injection intensity* in inj/s/bit was proposed in this part of the research. The unit represents the number of fault injections per time per circuit size, which is very similar to the radiation flux rate. This unit considers the bitstream area under the fault injection, i.e., larger designs receive more SEUs with a shorter interval between and vice versa for equivalent fault injection intensity.

Voter types in RDTs, n-MR Redundancy Techniques In the first steps of the research, a word-based voter was utilized in the RDTs. A question arises about whether a bit-oriented voter would be smaller in terms of chip area and what consequences would the bit-based voter have on reliability. Unlike a bit-based voter, which computes the majority function per each bit of the voted word, a word-based voter selects the results based on the complete word representation. The latter has, on the contrary, a straightforward implementation in the C++ language. Furthermore, the following experiment implemented additional FT mechanisms from the n-MR group of approaches into the RDTs.

The following experiment still utilizes the evaluation platform described in Section 3.2.1. In this experiment, repeated fault injection of a given frequency according to the fault intensity is performed. Based on preliminary experiments, the fault injection is selected to 2.0×10^{-6} inj/s/bit. The fault injection intensity represents a trade-off between the time needed to achieve the critical accumulation of faults in the design, measurement accuracy, and throughput of the fault injector. This is mainly because the evaluation platform used in this measurement does not allow the clock gating of the tested circuit while a fault is injected, thus limiting the frequency of fault injections by the real-time throughput of the injector. This is because of complicated synchronization with the simulated environment. The number of experiments was set to $0.1 * LUTs\_bits$, where $LUTs\_bits$ represents the number of bits of the bitstream used to implement LUTs, as LUTs are the main target of the fault injection experiments. This is how the number of experiments also reflects the size of the measured circuit to statistically maintain certain accuracy of the reliability measurement. Bit sizes, the percentage of failed runs (i.e., showing an electronic failure –

runs during which discrepancies were detected on the outputs of the robot control unit), and the MTTF were measured. Measurements were performed for all the bit- and word-based voters in combination with the triple (i.e., TMR), quadruple (i.e., 4-MR), and quintuple (i.e., 5-MR). The results are shown in Table 3.2. A detailed overview of the consumed resources, including the maximal frequency of the implementations, is listed in the original Paper C in Table II. The resulting MTTF and the representation of failed runs are visualized again in Figure 3.6.

Table 3.2: The overview of the parameters of testing and the results obtained.

RDT Applied to the Robot Controller Unit Algorithm		Parameters of Testing			Results Obtained	
		LUTs bits [b]	Num. of Runs [-]	Fault Rate [inj/s/bit]	Failed Runs [%]	MTTF [s]
	noft (no RDT)	19392	1940	2e-6	21.24	131.05
Word Majority	triple	48704	4871	2e-6	18.99	139.40
	quadruple	73216	7322	2e-6	20.88	138.14
	quintuple	122880	12288	2e-6	21.34	141.06
Bit Majority	triple_bit	24480	2448	2e-6	18.91	128.68
	quadruple_bit	26784	2679	2e-6	20.87	132.88
	quintuple_bit	37632	3764	2e-6	25.05	130.08

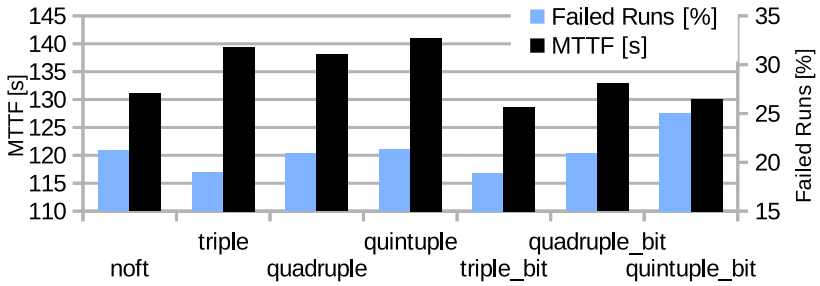

Figure 3.6: Reliability parameters for robot units utilizing TMR, 4-MR and 5-MR RDTs in combination with word- and bit-based majority voters.

As the chart shows for the RDTs with a word-based voter, the TMR RDT decreased the number of failed runs (i.e., the runs detected as an electronic failure by the evaluation platform). In the 4-MR RDT, the percentage of failed runs was also reduced (compared to the reference) but is still higher than for the TMR. This is because the 4-MR occupies more area yet still requires three out of four results (or bits) to be correct to work correctly. For the 5-MR, the percentage of failed runs is slightly higher than for the reference unit. On the other side, for the RDTs utilizing bit-based voters, the TMR RDT achieved the best percentage of failed runs, only 18.9 %. In opposition, its MTTF decreased, suggesting that the scatter of TTF values is higher in this case. The 4-MR, similarly to the RDT with the word-based voter, indicates an increased percentage of failed runs due to the nature of the 4-MR structure. For the 5-MR, the bit-based majority function provides worse results. The experiments can be concluded that if the percentage of failed runs is essential, the bit-based voter is more suitable. If the MTTF is important, then the word-based voter

achieves higher MTTF values. Each RDT thus provides different performance in terms of mission time and failure rate percentage.

Measurement Precision The number of measurements was previously set to one-tenth of the number of tested bits (i.e., $Tests\_num = 0.1 * LUTs\_bits$). This was an empiric selection. For this reason, a retrospective analysis of obtained difference is performed to decide whether this amount of verification runs is enough. Figure 3.7 shows the percentage difference between the achieved final measurements (in which the $Tests\_num = 0.1 * LUTs\_bits$) and measurements with a lower precision achieved by decreasing the number of verification runs. The monitored parameter for this analysis is the ratio of runs with an electronic failure. It is calculated by taking only a certain amount of first results (i.e., as if the number of runs was smaller). And the difference is then calculated in percentage points. In this chart, the number of verification runs is related to the size of the design (more precisely, its LUT contents, which were the target of SEU fault injection).

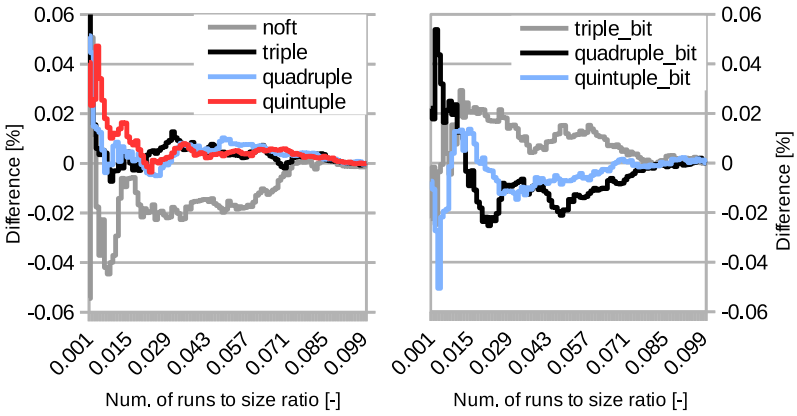

Figure 3.7: The retrospectively-calculated differences in the failure rate for smaller numbers of verification runs.

As can be observed, starting from the ratio of 0.073 (i.e., $0.073 * LUTs\_bits$), the differences from all the retrospectively-calculated differences stay below the 0.01 % and remain at a level as the ratio rises to its final value of 0.1. Assuming the average converges to the ideal value, the number of verification runs is sufficient to ensure that the differences of results stay below the 0.01 %.

Key Publications

(C) Majority Type and Redundancy Level Influences on Redundant Data Types Approach for HLS, LOJDA Jakub, PODIVÍNSKÝ Jakub, KOTÁSEK Zdeněk, KRČMA Martin. In: *2018 16th Biennial Baltic Electronics Conference (BEC)*. Tallinn: IEEE Computer Society, 2018, pp. 1-4. ISBN 978-1-5386-7312-6.

3.3.3 FT Evaluation: Accelerated Testbeds, FPGA Design Metrics

At this point, the speed of testing has become the limiting factor in continuing the research. Moreover, for the automation method to be practically applicable, the testing needs to be faster in orders of magnitude compared to using the previous approach. For this reason,

the work on a new testing approach was initiated, targeting explicitly accelerated and automated measurement of reliability parameters on FPGAs. The main goal is to accelerate the testing and allow (at least semi-)automated preparation of tests which will be part of the proposed FT design automation approach. The design automation approach must evaluate the suitability of the incorporated measures, and using as much information as possible is essential in this process. This section provides additional details to the evaluation framework, which was briefly mentioned in Section 3.2.2.

Research Steps

- Design and implement a new FT testing and measurement framework, which will be suitable for automated testing of candidate designs. It will also significantly accelerate the evaluation.
- Demonstrate its functionality; primarily focus on measuring the sensitive (i.e., critical) bits ratio.
- Examine to what extent the bitstream coverage influence the resulting precision.

Fault Tolerance Estimation Framework For FT design automation to be practical and usable, it is essential to evaluate the candidate solutions quickly. This is especially important to accelerate the FT system design process, as the evaluation requires non-negligible periods of time. For this reason, the *Fault Tolerance Estimation Toolkit* (FT-EST) was designed and implemented as a part of this thesis, which targets the testing of FPGA designs. It bases on the automated generation of the so-called testbeds. The testbeds are generated to incorporate several acceleration techniques to minimize the design evaluation and testing time. Also, testbeds and the FT-EST design use a modular architecture, allowing for easy extensibility and modifications of its components. A testbed comprises the testing core and the tested designs (a.k.a. designs under tests).

The complete FT-EST comprises an HW design, executed on an FPGA, and an SW part, which runs on a PC. Naturally, both parts must be connected, which is abstracted in a separate communication module, easing to port the communication abilities among FPGA manufacturers and their specific transfer protocols. The FT-EST incorporates several acceleration techniques, including 1) Many instances of a tested design can be evaluated simultaneously, multiplying the time efficiency. 2) Stimulation data are generated directly on the FPGA, eliminating communication bottlenecks between the designs under tests and the stimuli generator. 3) The output data are also examined on the FPGA itself for the same reason as in point (2).

Besides the acceleration techniques, the FT-EST is also designed to minimize the configuration changes needed to adapt the FT-EST to a tested FPGA design. However, specific parts of the FT-EST might be modified to fine-tune an experiment. For this reason, the FT-EST is highly modular, and each module is strictly isolated from the rest of the system. This allows altering the default setup rather straightforwardly.

The testing core is completely written in VHDL, utilizing the so-called VHDL generics to describe dynamically-generated parts of the testing core. The complete overview of an FPGA running the FT-EST testbed and the SW on a PC is shown in Figure 3.8. The complete description of the FT-EST testbed (including its modules) and the SW part running on a PC is shown in the original Paper D in Sections VI.C and VI.D.

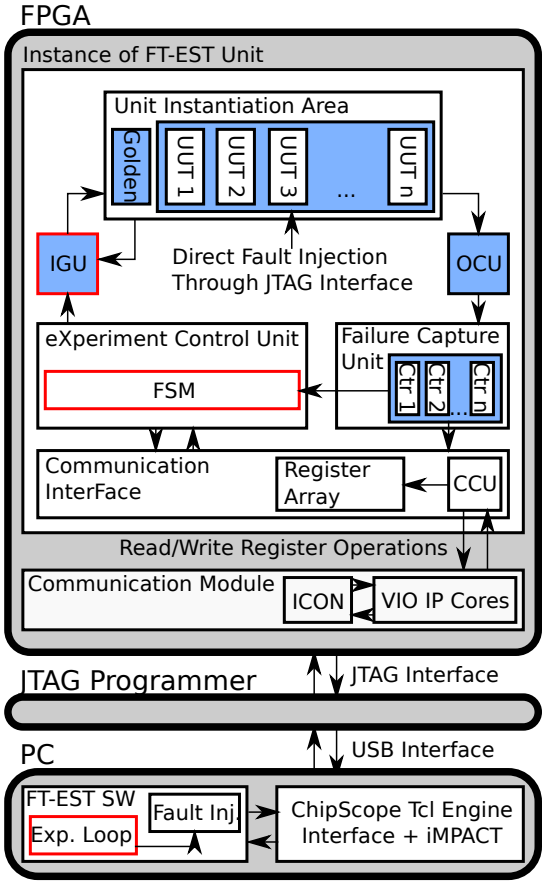

Figure 3.8: Diagram of the FT-EST architecture; the parts highlighted in blue are dynamically and entirely automatically generated; the parts highlighted in red can be modified by the designer to specify the experiment setup.

First Automatically-Generated Testbeds As the first benchmarking circuits, simple addition, subtraction, and CRC-8 algorithms are selected and implemented. For their implementation, the C++ language is used in combination with the RDT approach, described previously in this thesis. A thorough evaluation of such simpler designs allows to further verify the RDT approach. An overview of the selected benchmarks is in Table 3.3. For each benchmark, two versions are prepared: 1) the simplex implementation, which does not incorporate any FT techniques (i.e., as the reference), and 2) the triplicated implementation utilizing the RDT on each variable. These designs are synthesized using the Catapult C University Version (UV) 8.2b [51] and after adding the FT-EST core with the Xilinx ISE 14.7 [80].

The triplication, however, leads to the primary outputs of the design also being triplicated. For this reason, a VHDL-implemented voter of a given width is always added to unite the results. This voter is also subject to fault injections and size measurements, as illustrated in Figure 3.9.

For each examination of the design under test, precisely one fault injection is performed into occupied LUTs. This is accomplished until all the bits are tested. The input generation is configured to cycle through all the possible combinations (i.e., using the counter) with a step of 43. This step was selected purely empirically based on adequate time requirements

Table 3.3: An overview of the benchmarks selected for the purposes of evaluation.

Benchmark	Inputs	Outputs
Addition	A: 16-bit unsigned int.	$A + B$: 16-bit unsigned int.
	B: 16-bit unsigned int.	
Subtraction	A: 16-bit signed int.	$A - B$: 16-bit signed int.
	B: 16-bit signed int.	
CRC-8	A: 32-bit data	$CRC_8(A)$: 8-bit checksum

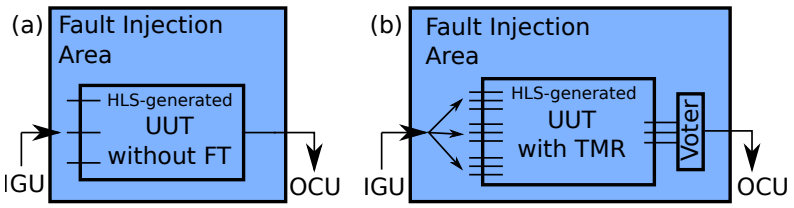

Figure 3.9: The testing for (a) a simplex component and (b) the component triplicated using RDTs.

and demands on the toggle rate of the primary input bits (i.e., specific numbers, when added, do not change the bit values of least significant bits, rendering them untested). The output comparison monitors each discrepancy, including time shifts in timing. The experiment control tests each of the LUT-occupied bits, thus allowing us to obtain the specific list of critical bits (i.e., without statistical approximation). The results are shown in Table 3.4. The total number of injections and the number of injections that led to a discrepancy in primary outputs are presented. From these, the percentage of critical bits is calculated. The application of RDTs led to a lower rate of critical bits for each benchmark. The reduction of critical bits is very dependent on the benchmarks. For example, for the CRC-8, RDTs reduced the critical bit percentage from 34 % to 13 %. Very similar numbers were obtained for the addition benchmark. However, for the subtraction, the number of critical bits lowered from 4 % to 3 %. Nonetheless, the RDTs prove to lower the critical bit percentage. Please note that the triplication using RDTs is not equivalent to the TMR, as only the data path is triplicated alongside other specific components. This makes the resource consumption sometimes smaller than three times the simplex version (as opposed to the TMR).

SEU Coverage vs. Accuracy As cycling through all the occupied bits is very time-consuming for larger designs, it should be statistically possible to test only a random portion of them. This would supposedly lower the accuracy of the measurement; however, it also significantly reduces the time needed to measure a design. This is why in the next experiment, such an idea is tested. Only a random portion of targeted bits of the bitstream is covered, while the achieved accuracy is monitored. The accuracy is calculated using the exact values obtained from the previous full test (i.e., in which all the bits are tested). In this experiment, the percentage of tested bits is lowered, and bits are selected uniformly at random to obtain the implication on accuracy. The data is calculated based on 1000 runs

Table 3.4: The number of SEUs that caused an output mismatch.

Algorithm	FT mechanism	LUT bits total [b]	Num. of inj. [-]	Num. of disturbances [-]	Sensitive bits [%]
Addition	none (simplex)	4288 b	4288	890	20.76 %
Addition	TMR	8320 b	8320	225	2.70 %
Subtraction	none (simplex)	4288 b	4288	178	4.15 %
Subtraction	TMR	8320 b	8320	278	3.34 %
CRC-8	none (simplex)	4800 b	4800	1658	34.54 %
CRC-8	TMR	6592 b	6592	879	13.33 %

for each combination of coverage vs. design. Table 3.5 shows the deviation in the detected percentage of critical bits (i.e., in percentage points).

Table 3.5: The deviation of the estimations for various SEU coverage settings (less is better).

SEU cove-rage	Deviation Range of the Estimation [% points]					
	Addition simplex	Addition TMR	Subtraction simplex	Subtraction TMR	CRC-8 simplex	CRC-8 TMR
60 %	-1.63 – 1.63	-0.40 – 0.46	-0.89 – 0.70	-0.46 – 0.63	-1.94 – 1.71	-0.97 – 1.10
30 %	-2.47 – 2.57	-0.78 – 0.98	-1.20 – 1.91	-1.10 – 0.91	-3.64 – 3.38	-2.46 – 1.99
10 %	-5.36 – 6.06	-1.50 – 1.74	-2.52 – 2.61	-1.90 – 1.71	-6.21 – 6.29	-3.78 – 4.26
5 %	-9.60 – 11.00	-1.98 – 2.58	-3.69 – 4.71	-2.62 – 3.15	-9.54 – 9.63	-5.14 – 5.78
1 %	-16.10 – 16.60	-2.70 – 6.91	-4.15 – 12.20	-3.34 – 8.68	-19.96 – 21.70	-11.80 – 18.50

As observed, the accuracy of results estimated with the lower bit coverage is higher for larger designs. However, this remains true only within one type of tested design. While the accuracy keeps at a decent level, the time required is significantly lower. Please note that if only 10 % of bits is covered, then only 10 % of time is necessary to perform the measurement. This is important to keep the automated design at a decent level by allowing the exploration of more candidate solutions and, thus, to find a possibly better solution during the design.

Extended Methods of Data Analysis and Visualization The number of critical bits is often used as the indicator of reliability. It is important to study different parameters, such as critical bit chunk sizes, positions, etc. The number of erroneous transactions per fault is also an essential factor. This motivated the research to extend further the analysis of measurements obtained by the FT-EST framework.

The data obtained through the FT-EST are often three-dimensional (depending on the measurement). For example, for each data stimulus (e.g., binary combination number, one transaction, etc.) and each bit flip of the bitstream, the number of erroneous results (or transactions) is measured. From this, the three dimensions can be identified: 1) stimuli transactions (input number intervals, transaction types, operation types, etc.); 2) bits of the bitstream to which the fault injection is performed; and 3) the type of erroneous outputs

(error types, number of erroneous outputs, number of mismatching bits, etc.). An extended FT-EST data analysis method is presented in [45]. The author of this thesis developed this as part of the FT design automation toolkit. However, it is not a part of this thesis to maintain it less extensive.

Key Publications

(D) FT-EST Framework: Reliability Estimation for the Purposes of Fault-Tolerant System Design Automation, LOJDA Jakub, PODIVÍNSKÝ Jakub, ČEKAN Ondřej, PÁNEK Richard, KOTÁSEK Zdeněk. In: *Proceedings of the 2018 21st Euromicro Conference on Digital System Design.* Praha: IEEE Computer Society, 2018, pp. 244-251.

3.3.4 FT Mechanism Selection Strategy: Multiple-Choice Knapsack Problem

In this research stage, the automated insertion of FT mechanisms is accompanied by accelerated measurement of reliability parameters. So far, FT techniques have been applied to each part of the design based purely on the empirical decision. This is why it's time to find an algorithmic method to individually select the proper FT mechanisms for each part of the design to maximize the FT achieved and minimize the size of the resulting design. This is an essential part of the automation technique, based on the previously-confirmed hypothesis that each part of the circuit reacts with a different improvement in reliability with a given FT mechanism.

Research Steps

- Find and describe a deterministic method to assign the best FT technique to each part of the design while satisfying a given constraint (e.g., design size).
- Perform theoretical calculations of the number of critical bits in a design with the selected method.
- Implement this method into the FT design automation toolkit and perform measurements of the number of critical bits in a design on a real HW to prove the concept.
- Compare this method to the previously used approach, in which the same FT technique was consistently applied to each part of the design.

Using Multiple-choice Knapsack Problem for FT Assignment Let's find a method for automated allocation of hardening type to particular components of a system. This must be performed while one parameter is optimized (e.g., minimization of the number of critical bits) and another constraint is considered (e.g., available chip area). It is undoubtedly beneficial if a method has a well-documented formal base. At first sight, this problem is very similar to the *Knapsack Problem* (KP) [49]. Let's suppose we have a hypothetical knapsack with a maximal load capacity. Also, let's assume we have a set of items, each with its profit and weight. We aim to select such items to maximize the sum of profits and keep the sum of weights under the load capacity of the knapsack. Intuitively, one can see that the items represent component variants of a system, and the weights are the sizes of the components. Their profits then represent the benefit of hardening, and the total load

capacity of a knapsack is the available area on the chip. However, one can also see that the KP does not correspond fully to the hardening allocation needed in this research. It is because the items are in one set in the original KP definition. Thus the solution is not constrained to select precisely one variant of each system component. A variant of the KP, called *Multiple-choice Knapsack Problem* (MCKP) [33], however, exists, which addresses precisely this issue. The MCKP divides the items into sets, and from each set, precisely one item must be selected while still searching for the highest sum of profits and keeping the sum of weights given by the load capacity as the constraint.

To define the MCKP precisely, m sets (e.g., classes) of objects $N_1 \dots N_m$ and a hypothetical knapsack of capacity c exist. For each object $j \in N_i$, a parameter of *weight* w_{ij} and *profit* p_{ij} is stated ($i \in \mathbb{N}, i \leq m$). A solver aims to select precisely one item from each class of objects $N_i, i = 1, \dots, m$ while maximizing the sum of profits, subject to the load capacity c of the hypothetical knapsack. The binary variable x_{ij} is introduced to define the MCKP formally. The x_{ij} takes on value 1 if the j is chosen in the class N_i and 0 otherwise. The MCKP is then formally defined as the maximization problem by Equation 3.1a, constrained with Equations 3.1b, 3.1c and 3.1d [33].

$$\text{maximize} \quad \sum_{i=1}^{m} \sum_{j \in N_i} p_{ij} x_{ij} \tag{3.1a}$$

$$\text{subject to} \quad \sum_{i=1}^{m} \sum_{j \in N_i} w_{ij} x_{ij} \leq c, \tag{3.1b}$$

$$\sum_{j \in N_i} x_{ij} = 1, \qquad i = 1, \dots, m, \tag{3.1c}$$

$$x_{ij} \in \{0, 1\}, \qquad i = 1, \dots, m, j \in N_i \tag{3.1d}$$

Then, the load capacity c of the knapsack represents the available area; the item weight w_{ij} represents the size of variant j of component i ($j \in N_i$; N_i is a set of variants of component i); the item profit p_{ij} is the FT mechanism hardening gain. The sets of items N_i consist of variants of a component (e.g., TMR, duplex, etc.), $i = 1 \dots m$, m is the number of components of the system. One minor implementation detail must be, however, considered. For the critical bits optimization, minimization of a *handicap* is more straightforward than maximizing the *profit*. This is because the handicap can be directly the number of critical bits without any need to recalculate the values between flipped intervals. The graphical representation of the MCKP used to solve the FT mechanism allocation is presented in Figure 3.10.

Theoretical Calculations To demonstrate the MCKP method, its usage will be divided into two stages. At first, FT systems will be automatically designed. However, the parameters of these systems presented at the end of this stage are considered only theoretical (i.e., obtained by calculations). This is because the MCKP uses approximated parameters calculated from preliminary measurements of individual system components. The parameters of the resulting circuits are, thus, only an approximation and demonstrate data that the MCKP uses internally to find the solution. The final parameters of the resulting systems are obtained after accurate measurements, which are presented later in the next paragraphs about the practical measurements.

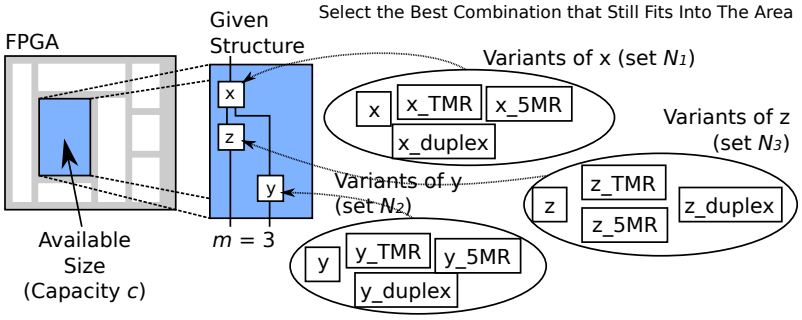

Figure 3.10: Graphical representation of hardening allocation mapped to the MCKP.

We have the triple, quadruple and quintuple RDTs to manipulate descriptions in the C++ language described and designed as a part of this thesis. The target of the following case study is to demonstrate the usage of an MCKP solver to minimize critical bits number while keeping the design under a given size constraint. A simple system utilizing serial and parallel connections between its components is used as a benchmarking system for this purpose. It performs an artificial task: it calculates the number of one bits in the sum of two 16-bit digits and one constant. It also produces a CRC-8 checksum from the intermediate results. Each of the components is designed in the C++ language to allow the usage of RDTs. The system diagram is shown in Figure 3.11.

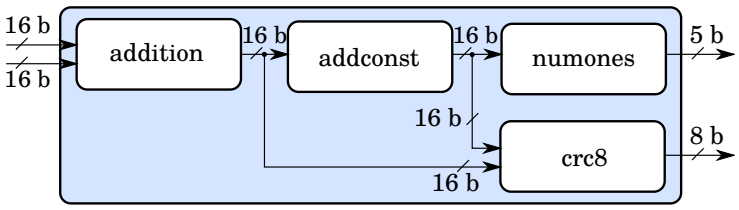

Figure 3.11: The benchmarking system used to demonstrate the usage of the MCKP solver to minimize the number of critical bits.

To accelerate the automated design (further to the techniques already presented in the previous parts of this thesis), this research shows the possibility of minimizing the number of measurements needed during the design. This is done by measuring each component exclusively (including all their hardened variants), which is later used to calculate the parameter of the overall system completely in SW. This is significantly faster than repeated measurement of a complete system each time a solver needs to evaluate a new candidate solution. Thus, in the first phase, critical bits numbers and sizes for each component (including its hardened variants) are measured on the target HW. And after that, the data of these components are used to calculate the overall critical bit number and the size of the resulting system configuration (i.e., according to a candidate hardening allocation during the design). Data acquired for the calculations within the following case study are shown in detail in the original Paper E, Table 1.

Such acquired data are subsequently used to calculate critical bits numbers and sizes of emerging candidate systems by the MCKP solver during its operation. The operation of the MCKP solver is then very fast, as no further measurements are required. A straightforward

MCKP solver, implemented for this purpose by the author[1] is used, which performs these calculations. The solver's output is the estimated size, the estimated number of critical bits, and the selection of component variants that meet the given area constraint. The MCKP solver itself is implemented to always find the optimal solution in a deterministic way. Nonetheless, the data the solver works with are approximated, which must be kept in mind. The data obtained from the MCKP solver are visualized in the chart in Figure 3.12.

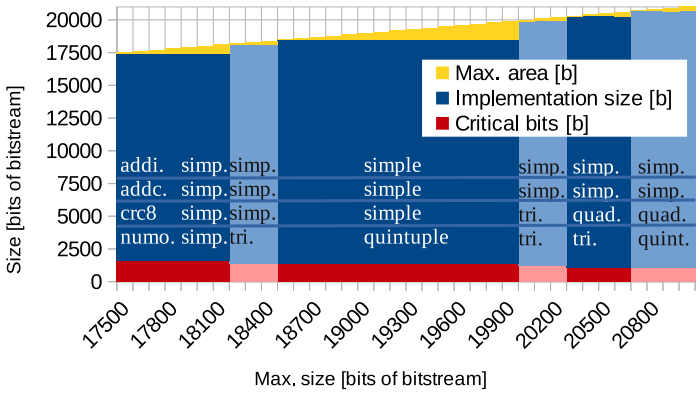

Figure 3.12: The calculated theoretical values (i.e., implementation size and the number of critical bits on the y-axis) for the system configurations obtained from the MCKP solver by changing the available chip area (i.e., max. area on the x-axis).

As seen in Figure 3.12, the x-axis shows the maximal size of the available area that the MCKP solver was provided with. The yellow color also demonstrates this on the y-axis. The height of the blue color reflects the size of the resulting (i.e., automatically designed) system. The size of the system (i.e., blue) must always fit into the available area (i.e., yellow). As observed, the solver constantly changes the FT mechanisms allocation after the available area is enough to provide a better allocation (i.e., allocation with fewer critical bits). The two shades of blue distinguish between different FT mechanisms allocations, which are described directly in the corresponding parts of the chart. The red part of the system size on the y-axis demonstrates the representation of critical bits in the system. It is desired to minimize this red area, which is paid for by adding more of the blue area.

At first sight, it is evident that the MCKP solver utilizes all the provided area. A critical observation is that providing more space decreases the absolute number of critical bits. This is a good indication that the MCKP solver is a promising strategy. Moreover, it can be seen that the MCKP solver selects to harden only the crc8 and numones components, as the other two components' hardening does not bring us closer to the chosen goal of critical bits minimization.

Practical Measurements After the examination of the theoretical output of the MCKP solver, it is essential to test the actual results by synthesizing the proposed systems and measuring the critical bits numbers on the target Virtex 5 platform. These are presented in the chart in an identical form to the previous theoretical results in Figure 3.13.

[1]The MCKP solver needs to aggregate the resulting parameters of a current candidate system. Thus, it was easier to implement a straightforward solver that approximates the target parameters on demand in specific methods in its program, which allows the user to aggregate various types of metrics. It also allows the user to set whether a parameter is maximized or minimized.

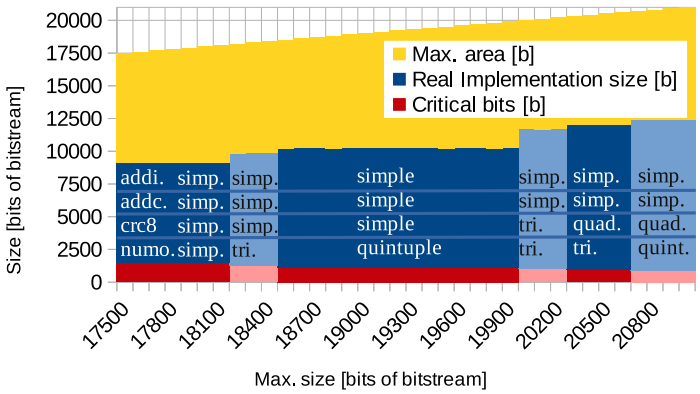

Figure 3.13: The values measured on the real target Virtex 5 platform (i.e., synthesized implementation size and the measured number of critical bits on the y-axis) for the system configurations obtained from the MCKP solver by changing the available chip area (i.e., max. area on the x-axis).

The first important outcome to notice is a significant difference in a system's estimated and final size. It is, however, important that the sizes remain in relation (i.e., a more extensive system in the theoretical calculation also remained larger after the measurement). It is assumed that the synthesis, which is a black box for us, works differently for small components and larger systems. Possibly because the more extensive system provides more space for optimization; nonetheless, such a discrepancy could be calibrated by applying coefficients to the estimation. The measured numbers of critical bits of the resulting designs also directly relate to the theoretically calculated results. Moreover, the measured representations of critical bits are slightly smaller than the theoretically-calculated ones, which is actually a positive outcome. Again, the precision of the estimation could be calibrated by applying coefficients to the estimated data. It is essential to notice that the numbers of critical bits fall (even in the absolute domain), although the system is becoming larger as the FT mechanisms add more redundancy. Therefore we see that the added redundancy is used efficiently to increase the FT of the system. This is important for the MCKP design strategy to be considered correct.

Comparison to the Previous Approach So far, in this thesis, the hardening mechanisms were selected purely empirically. This is why this MCKP solver approach is compared to a *naive* approach, always utilizing the same hardening type per component. The comparison of approaches is evaluated on the previously-used benchmark, which was presented in Figure 3.11. The comparison using this benchmark is in Figure 3.14.

The systems on the left side, which always had a constant hardening type per each component (e.g., triple for each component, quadruple for each component, etc.), have significant steps in their size among them. This might be a problem for applications where a designer intends to utilize the available area fully. This is not the case for the systems designed with the assistance of the MCKP solver (six FT designs were selected for this comparison, see Figure 3.14 on the right), which increase their size in a very fine-grained fashion. Moreover, the systems designed by the MCKP solver follow the trend of critical bit minimization. This is not always true for the naive strategy used to create systems on

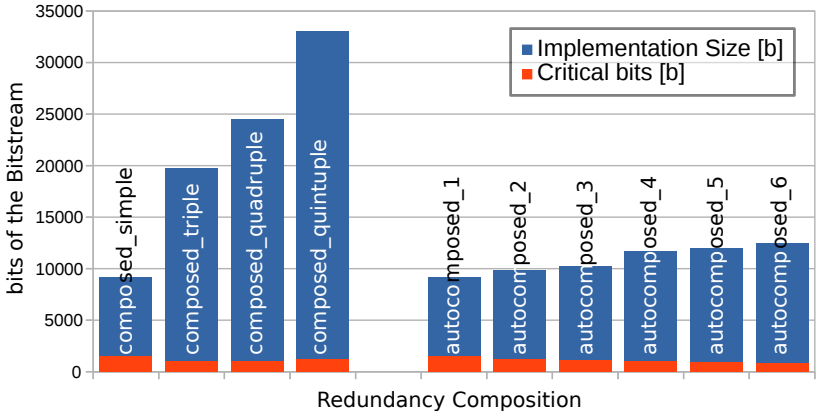

Figure 3.14: Selected FT implementations of the benchmarking system from Figure 3.11 created by a naive approach (four FT designs on the left) and MCKP solver (six FT designs on the right).

the left side of the chart. The results on the right side achieve equivalent or lower numbers of critical bits, yet their overhead is smaller.

Key Publications

(E) Automatic Design of Reliable Systems Based on the Multiple-choice Knapsack Problem, LOJDA Jakub, PODIVÍNSKÝ Jakub, ČEKAN Ondřej, PÁNEK Richard, KRČMA Martin, KOTÁSEK Zdeněk. In: *Proceedings - 2020 23rd International Symposium on Design and Diagnostics of Electronic Circuits and Systems, DDECS 2020*. Novi Sad: Institute of Electrical and Electronics Engineers, 2020, pp. 1-4. ISBN 978-1-7281-9938-2.

3.3.5 FT Mechanisms Insertion into the VHDL Source Code: Hardening Methods

So far, this research has dealt with designs implemented using the C++ language in combination with the HLS. At this point, it is possible to switch the design language by simply replacing the part of the design method that modifies the design codes. In the following demonstration, the VHDL is used. Modifying a design in a VHDL is more complicated than the previously used C++ codes. It can be slightly simplified by using the so-called VHDL generics and partly focusing only on the modified structures of the code. Also, the first possibilities to incorporate the so-called repair mechanisms into the resulting designs are examined during this research stage.

Research Steps

- Design and develop mechanisms to manipulate the source code in the VHDL language in order to introduce FT mechanisms into the design description.
- Evaluate the functionality on a benchmark by focusing on the TTF and circuit size parameters.
- Firstly, detect the diverging components suitable for recovery using the DPR technology.

- Secondly, find and exploit the possibility of utilizing the proposed design method and toolkit to automatically incorporate self-reparation techniques into the detected components of the resulting design.

Manipulating the VHDL Code To extend the FT design automation approach presented in this thesis with the ability to harden VHDL designs, the means to modify VHDL code must be added. These must insert FT structures into the selected part of the code. In the following approach, targeting *entities* (in the sense of the VHDL terminology) was sufficient. Thus, these entities define partitions of the system, and the system is a chosen *top* entity within the VHDL project. To specify the desired modification, special code comments are written around the instantiation of an entity.

After the comments specifying particular FT mechanisms are placed in the VHDL source code around the entities, an application written explicitly for this purpose is executed, which modifies the corresponding code to realize the specified FT mechanism. Each time a file in the project is automatically modified, a backup is created to allow revert. The modifications are based on a set of templates, each template holding the structure of one FT mechanism (e.g., TMR, the addition of a DPR controller, etc.). This template-based design is intentional, as it easily incorporates new FT mechanisms and uses them in the automated design process. The templates are supplemented by procedures that provide the necessary information to fill the templates. These procedures search the complete VHDL project directory and are implemented on a class-based approach. This allows the user to easily extend them alongside the newly added templates (and potentially newly demanded variables, which need to be filled in the template).

The code modification process always targets one VHDL file (i.e., *top* file for a given part of a system). Multiple executions of the process are needed if multiple files are intended to be modified. The process starts by dividing the original input VHDL file into the so-called *tokens*, each containing a code snippet. These also include metadata, which can be obtained during the tokenization process, such as identification (i.e., instantiation code intended for modification, *don't care* instantiation, remaining parts of code, etc.). After that, the instances identified as intended for modification are processed. The corresponding procedures are called, which correspond to the intended hardening type. These search for entity declaration, signals, their direction, bit width (which must be enumerated for certain cases), etc. The clock signal is also detected automatically based on its characteristics and name. This is important for structures utilizing the clock signal for auxiliary components (e.g., clocked TMR voter, DPR controller for component renewal, etc.). After all the required variables are detected, the corresponding template is filled, and a new entity is created, which contains the original entity and the added structure. This is called the *out-of-place modification*, as it makes a new VHDL file containing the new entity. The newly created entity is then referenced in the place of the original entity, which is called the *in-place modification*. Also, the special comments are disabled to avoid chained application of the same FT technique again – for such cases when the designer executes the modification flow multiple times per one file. This complete modification flow is graphically illustrated in Figure 3.15. It is important to note that this method of FT mechanism insertion is compatible with the previously proposed strategy selection method (i.e., the MCKP-based strategy, which will be used in the subsequent experimental demonstration) to form a fully automated toolkit of the complete FT system design.

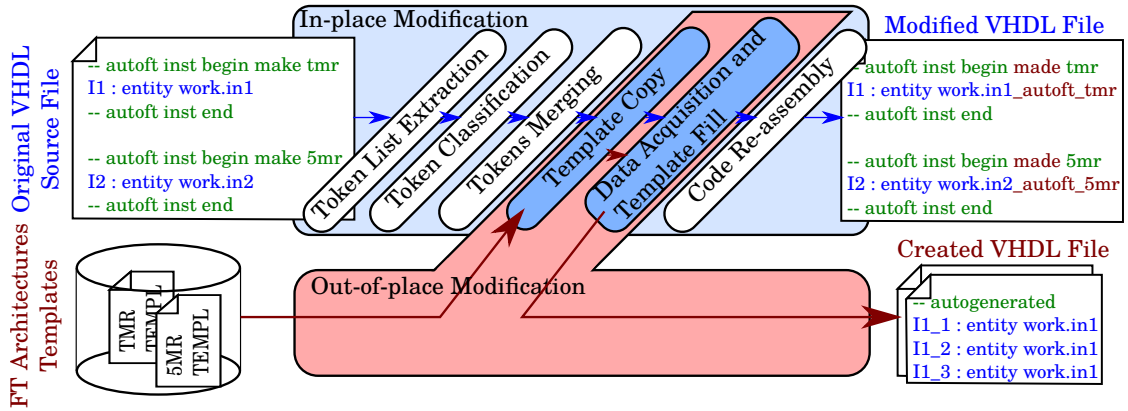

Figure 3.15: The VHDL code modification flow based on the template system and procedures to fill the templates.

First Experiments with VHDL Designs In this case study on the automated design of the FT system described in the VHDL code, the objective is to maximize the median of the TTF parameter. The median value of TTF is also called the *t50* parameter. It can also be viewed as the time of the mission, in which the probability that a system is still correctly functioning is exactly 50 % (i.e., no failure is observed on the primary outputs of a system). In this case, where the TTF parameter is examined, a minor modification in the measurement and fault injection setup is required. This involves the configuration of the presented FT-EST toolkit by using the *Linear Feedback Shift Register* (LFSR) as an input generator and a register-readable *stopwatches* in the place of the failure capture unit (one per each instance of the tested unit). One TTF measurement is stopped after each of the instances tested in the testbed is in a failure state (i.e., one or more failures were observed on its primary outputs). After one measurement is stopped, the SW reads the time of the first failure observation measured by the *stopwatches*, which is done per each instance of the tested unit. The SW was set up to fault injection intensity of 2.0×10^{-5} inj/s/bit.

The same benchmarking design is implemented in VHDL, which was previously used to demonstrate the usage of the MCKP solver on system design. Its block diagram is in Figure 3.11. During the preparation stage of the design (i.e., measurement of reliability parameters of various versions of a given component), TTF was targeted. The TTF for each component is presented in the original Paper F in Table 1 and Figure 4.

The parameters of candidate solutions are, again, calculated in the SW using the data from the measurement of the components. In the case discussed in Section 3.3.4, it was a simple task, as the number of critical bits was chosen as the only criterion. Now, there is the t50. Thus, the t50 of a system must be calculated from individual t50 parameters of the components used in the system. The t50 of the resulting system of dependent partitions is approximated by equations used to calculate the MTTF. At first, the *failure rate* (i.e., λ) is expressed based on previously-presented Equation 2.2 (i.e., the approximation lies mainly in fact, that the median TTF is used in the place of the mean TTF). After this is performed for each used component from the set C_{used}, the individual failure rates are added, as shown in Equation 2.1. Subsequently, the resulting sum of failure rates is approximated back to the t50 using the already-utilized Equation 2.2.

This approximation technique was incorporated into the previously-described MCKP solver to speed up the design space exploration. Three area constraints (i.e., bitstream

sizes) were tested with this approach: 20 000 bits, 25 000 bits, and 30 000 bits. Then, three system configurations were obtained and again measured using the FT-EST to get their accurate parameters. These parameters are presented in Table 3.6. The table also contains the measurements for reference systems utilizing a static allocation approach (i.e., the same hardening type per each component in a system).

Table 3.6: Automatically-designed FT systems (for benchmark in Figure 3.11, reimplemented in VHDL) using the VHDL code manipulators and the MCKP solver approach; the results are compared to the reference manually-designed systems utilizing static allocation; all the data are measured on Virtex 5 technology, and the measurements are performed under the fault injection intensity of 2.0×10^{-5} inj/s/bit.

System Name	FT Techniques				Bitstream Area [b]	t50 [ms]
	addition	addconst	crc8	numones		
MCKP Approach						
auto_20000	simplex	simplex	5-MR	simplex	18 624	43 198
auto_25000	simplex	simplex	5-MR	TMR	22 400	49 935
auto_30000	simplex	simplex	5-MR	5-MR	25 856	49 675
Static Allocation						
ref_simplex	simplex	simplex	simplex	simplex	9 152	23 559
ref_TMR	TMR	TMR	TMR	TMR	24 704	42 173
ref_5-MR	5-MR	5-MR	5-MR	5-MR	37 376	55 900

Table 3.6 shows that the MCKP solver targeted only the most failure-prone components (i.e., the crc8 and the numones). The crc8 obtained the highest hardening all the time, as it was the most failure-prone. Also, the resulting size is slightly smaller than the constraint enforces. This is because the solver works with the approximation to speed up the design, thus not continuously synthesizing the complete system per each possible configuration. It uses the same approach as presented earlier in Section 3.3.4: the size is the sum of the components sizes. This is, however, not always precise, as the synthesis performs specific optimizations when the system is larger. The smallest automatically-designed system is only 75.34 % in size of the reference system utilizing TMR for each component. However, the smaller automatically-designed system achieves yet better t50 time. The second automatically-created system is still circa 9 % smaller, compared to the reference TMR, but also its t50 is more than 7 s longer. The third automatically-designed system has the t50 nearly equivalent to the second system. It is, however, more extensive. This sub-optimal configuration is caused by the imprecise approximation of the t50 parameters, leading the MCKP solver to choose not-so-appropriate variants of components. The accumulation of faults might also cause this. For a more extensive system, the area exposed to faults is more significant, thus effectively worsening the problem of accumulation of faults, which cannot be solved by redundancy only (but also component repair must be implemented). This third system is, however, not completely bad compared to the reference system, as it is more than 30 % smaller than the reference manually-designed 5-MR system and its t50 is only 11 % smaller.

Detection of Diverging Components The FT mechanisms based on increased redundancy effectively increase resiliency against faults but only for shorter mission times. This

is because the faults are accumulating in the FPGA configuration bitstream, and logically, after a certain number of faults, the redundancy ceases to be efficient for masking them. For this reason, allowing the so-called renewal of the bitstream is interesting. This approach can repair faults and thus helps to eliminate their accumulation. To maintain the system functioning, masking is still used, but the flipped bits in the bitstream are searched and repaired to their previous state using DPR while the masking techniques (e.g., TMR) perform their function.

The MCKP solver was also extended to provide information on which part of the system should incorporate the recovery using DPR. This selection is based on the difference from the average values of the optimized parameter (i.e., t50 in this case). Using the function *avg_param(S)*, the arithmetic average t50 for all components of a final FT system utilizing allocated FT mechanisms is calculated, as shown in Equation 3.2. The S is a set of FT system's components c that already include selected FT mechanisms. Please note that the *card(S)* denotes the cardinality of the set S (i.e., the number of elements in the set; for S it is the number of components in the system). After that, the user must specify a difference between the optimized parameter and its average. Using this difference, a significance is determined, which results in a threshold, and all components crossing this threshold will be selected to apply recovery mechanisms. The difference is a real number from the interval of $\langle 0; 1 \rangle$, with 0 specifying all the components with their t50 below average for recovery mechanisms. On the other hand, for example, the difference of 0.5 specifies only such components to be suitable for recovery mechanisms whose t50 is under half of the average t50 time. Equation 3.3 demonstrates how to calculate the significance (i.e., *sig*) from the difference (i.e., *diff*). Equation 3.4 selects the proper comparison operator based on whether the optimization parameter is minimized or maximized. The set R in Equation 3.5 then contains the parts suitable for the recovery mechanism.

$$\mathbf{avg\_param}(S) = \frac{1}{\mathbf{card}(S)} \times \sum_{\forall c \in S} \mathbf{param}(c) \tag{3.2}$$

$$sig = \begin{cases} 1 - diff, & \textit{if } \mathbf{param} \textit{ is maximized}, \\ 1 + diff, & \textit{if } \mathbf{param} \textit{ is minimized}. \end{cases} \tag{3.3}$$

$$op = \begin{cases} <, & \textit{if } \mathbf{param} \textit{ is maximized}, \\ >, & \textit{if } \mathbf{param} \textit{ is minimized}. \end{cases} \tag{3.4}$$

$$R = \{\, c \mid c \in S,\ \mathbf{param}(c)\ op\ sig \times \mathbf{avg\_param}(S) \,\} \tag{3.5}$$

For the following case study, the difference is empirically set to 0.4 of the average. Also, in our case study, we maximize the t50 parameter. This means that the significance parameters will be 0.6, meaning that every partition that has its t50 worse than the 0.6 of the average t50 for the current system composition is denoted as suitable for recovery using DPR.

Incorporation of a DPR Controller to the Repair The recovery is performed by adding a new auxiliary component – the reconfiguration controller. Multiple components might be suitable for recovery. Thus, the reconfiguration controller is arranged only once to the *top* module of the hardened system. The controller subsequently implements the rewrite of chosen parts of the bitstream based on the previously described selection. It also allows rewriting of the component on demand. This might minimize the system's power consumption if the components themselves can self-detect their failure, thus not requiring permanently rewriting their partial bitstreams. This is usually the case of components hardened using the n-MR techniques, as a specifically-extended variant of a voter can detect redundant modules providing different results. The reconfiguration controller also connects to the internal reconfiguration interface (i.e., ICAP for Xilinx FPGAs) and external storage to save the bitstreams (e.g., a flash chip). As the research of the reconfiguration controller itself is not part of this thesis, the description is very brief, and further details can be found in [69] or publications of my colleague Richard Pánek [59, 58], who is also the co-author of the publications in which this research was presented.

The DPR controller is responsible for rewriting the bitstream part of a failing component module (e.g., module 1 of the TMR of component A). Two approaches to DPR repair are compared: a) The reconfiguration controller is placed on the same FPGA, and thus, the created system is fully implemented on one FPGA. The reconfiguration controller is then, however, itself prone to failure. b) The reconfiguration controller is placed outside the FPGA in a radiation-hardened shell to maintain its operational state. This second case is simulated in our approach. It still uses the same reconfiguration controller on the FPGA. However, it is omitted from the size calculations and fault injection, leaving it always functional. These approaches are graphically demonstrated in the original Paper G in Figure 6.

In the previous experiment (on the beginning of this Section 3.3.5), the automatically-designed system that was constrained to the 30 000 bits achieved sub-optimal results. One of the causes might be the accumulation of faults for a larger system, as a more extensive system must withstand a higher number of faults (considering absolute numbers) because a larger area is exposed to the manifestation. This 30 000 bit system is selected because it demonstrates that the results can be further improved by providing new FT mechanisms to the FT design automation method (such as component repair).

In Figure 3.16, the impact of various hardening schemes on TTF is demonstrated. The first three boxes are from the previous experiment (the beginning of this Section 3.3.5) for reference purposes. These are the systems utilizing static allocation of components. The automatically-created 30 000 bit system is in the middle part of the chart. For this system, components significantly deviating in the t50 are determined using Equations 3.2, 3.3, 3.4, and 3.5. The one marked component is treated using reparation mechanisms in the third part of Figure 3.16. This is done for (a) a system that self-contains the reconfiguration controller applied to the one marked component (the crc8) and (b) a system in which the marked component is repaired using an external always-functioning reconfiguration controller. In both these cases, the 5-MR of the marked component crc8 was downgraded to the TMR to save space while the reconfiguration controller repairs the individual modules of this crc8 component.

As can be observed in Figure 3.16, although the 30 000 bit system without DPR has nearly equivalent median TTF (i.e., t50) to the system with DPR controller on FPGA, it has a stronger scatter towards higher values. This is a positive behavior, as the lower TTFs remain nearly the same. It is important to note that the added DPR controller is also

subject to fault injection, and the design of the DPR controller itself does not incorporate any FT mechanisms. Utilizing a reconfiguration controller outside the fault injection area is much more efficient in improving MTTF. Also, the TTFs have many outliers towards higher values, which is also a good message. However, it is essential to highlight that the DPR repair was still targeted toward only one system component.

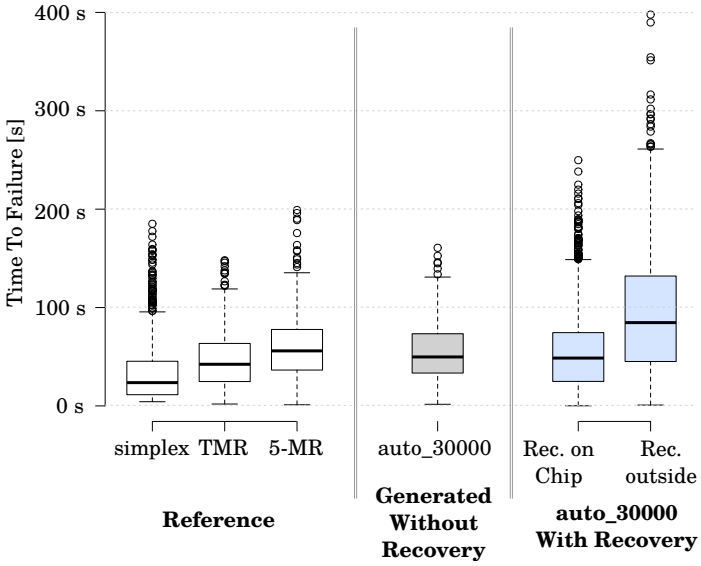

Figure 3.16: TTF values of the system without DPR recovery, with DPR on a chip and outside of the chip (i.e., outside of fault injection area), and reference measurements for the static allocation of TMR and 5-MR to each component (all measured on Virtex 5).

Key Publications

(F) Automatic Design of Fault-Tolerant Systems for VHDL and SRAM-based FPGAs, LOJDA Jakub, PÁNEK Richard, KOTÁSEK Zdeněk. In: *Proceedings - 2021 24th Euromicro Conference on Digital System Design, DSD 2021*. Palermo: Institute of Electrical and Electronics Engineers, 2021, pp. 549-552. ISBN 978-1-6654-2703-6.

(G) Automatically-Designed Fault-Tolerant Systems: Failed Partitions Recovery, LOJDA Jakub, PÁNEK Richard, KOTÁSEK Zdeněk. In: *2021 IEEE East-West Design and Test Symposium, EWDTS 2021 - Proceedings*. Batumi: Institute of Electrical and Electronics Engineers, 2021, pp. 26-33. ISBN 978-1-6654-4503-0.

3.3.6 FT Mechanism Selection Strategy: Multiple Objectives, Real System Case Study

In this research stage, the complete flow can be built using the presented components to automate the design of FT systems. Two important aspects, however, must be yet addressed to allow its usage in practice. The first is the possibility to optimize multiple parameters, i.e., not only one reliability parameter and one constraint, as was the case for the MCKP solver approach. Various parameters, such as reliability, design size, and power consumption, must be considered in practice. Thus, the method must also support them. The second aspect is that the method has been used only on an artificial benchmark design.

This time, an actual practical design must be used to evaluate the method, which will be further utilized. And the complete FT design automation method and toolkit, as a result of detailed experimentations, will be presented in one comprehensive article.

Research Steps

- Extend the method by supporting a multi-objective optimization.
- Present the complete design process in a case study on an actual practical design.
- Test and measure parameters of the automatically-designed system in a real scenario.
- Summarize the complete FT design automation method and its implementation in the form of the toolkit.

Optimization on Multiple Criteria In practice, the design process must consider multiple criteria. It is evident that, for example, a space probe has a given power budget (allocated by its solar panels and battery, etc.). Thus, the design process of systems for such a space probe must consider the reliability and size of the resulting design and the power consumption. Multi-objective design space exploration is thus required in the proposed FT design automation method when a real-world problem is targeted.

For multi-objective optimization, finding only one (i.e., the best) solution is impossible. This is because the objectives are usually conflicting. Hence, a set of so-called non-dominated solutions is a typical result in the case of multi-objective optimization. A trade-off must be selected among them. A well-known way to graphically display such a set of non-dominated solutions is the so-called Pareto-frontier [23].

This research utilizes full design space exploration to perform multi-objective optimization and determine an exact solution. Determining the reliability and other parameters of each candidate solution is unrealistic, as it requires measuring each possible combination of component variants in the system. However, after this method is combined with the approach allowing one to approximate the resulting parameters, the evaluations are requested only in the preliminary stage. After that, all the design-space creation and exploration are performed purely and very quickly in SW. For example, the MTTF can be approximated using the already-presented Equations 2.2 and 2.1; the power consumption can be approximated as a sum of individual parts' consumption, etc. Of course, some parameters cannot be easily approximated (e.g., the maximal frequency of an FPGA design). For such cases, the fastest way to obtain such a parameter has to be examined (e.g., executing the synthesis process partly and extracting the frequency report from there). This approach to full design-space exploration will be used in the following case study.

GPDRC as a Design from Practice *Generic Partial Dynamic Reconfiguration Controller* (GPDRC) is an actual implementation of a reconfiguration controller, which was previously used in this thesis to demonstrate components recovery. It was implemented in our research group, and thus, the source codes of the GPDRC are available. At the same time, the GPDRC is an unhardened component in a repaired system from previous experiments (Section 3.3.5 and [59]), as none of the FT approaches was implemented into its structure. This makes it an excellent candidate to demonstrate FT design automation on it. The GPDRC is used in the following case study, which targets maximization of the MTTF time while lowering the resulting system's size and power consumption.

The GPDRC consists of 10 components, each of which has a specific function. The names of the components are self-explanatory. Details of the GPDRC are out of the scope of this thesis and can be found in [69], or more compactly in Paper H in Section 5.1. The GPDRC connects to the ICAP interface of a Xilinx FPGA and to a flash memory chip, in which particular components' partial bitstreams are prepared. After the GPDRC is instructed by the *Module in Error* signal (of a proper bit width according to the number of modules), the GPDRC fetches the bitstream from the flash and reconfigures the affected component of the system. The GPDRC must be pre-configured to contain the addresses of bitstreams in the flash memory. The block diagram of the GPDRC is in Figure 3.17.

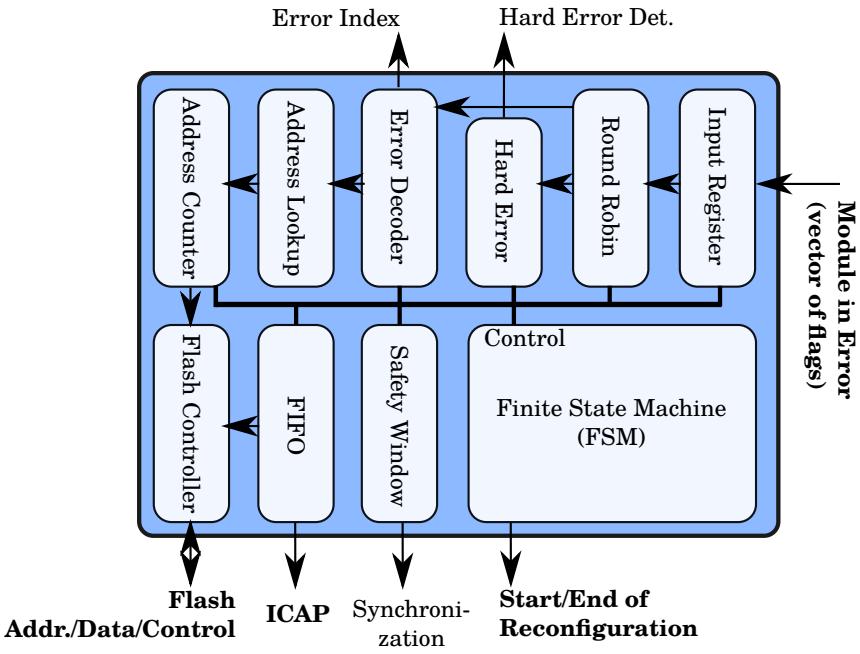

Figure 3.17: Block diagram of the GPDRC design (simplified) [69].

At first, the influences of hardening must be measured per individual components of the GPDRC. It is, however, challenging to prepare stimuli generators per each component of the system and correctly simulate the environment for each component. This research solves this by keeping the system whole and performing multiple system measurements in which only one component is always hardened. Each component is tested in its natural environment, and the stimuli generator is implemented only once per the complete system, as graphically demonstrated in Figure 3.18.

Thus, multiple systems (i.e., GPDRCs in our case study) are prepared, in which always one component is hardened. In addition, the original (i.e., reference – simplex) system is added. These systems are then called data-acquisition systems. Please keep in mind that in the following text, the notation $sys\_a_1b_1$ represents the system in which the component a is hardened by the FT mechanism 1 and the component b is also hardened by the FT mechanism 1, while the rest of the system components are without hardening (i.e., original – simplex). Therefore, using this notation, the previously-mentioned data-acquisition systems can be marked as $sys\_c_m, \forall c \in C, \forall m \in M$ (where C denotes the set of components, and M denotes the set of available FT mechanisms). The data-acquisition systems are extended by the mentioned one original reference simplex system $sys_{simplex}$. The MTTF is measured for each of these data-acquisition systems. After this data-acquisition stage of measurements

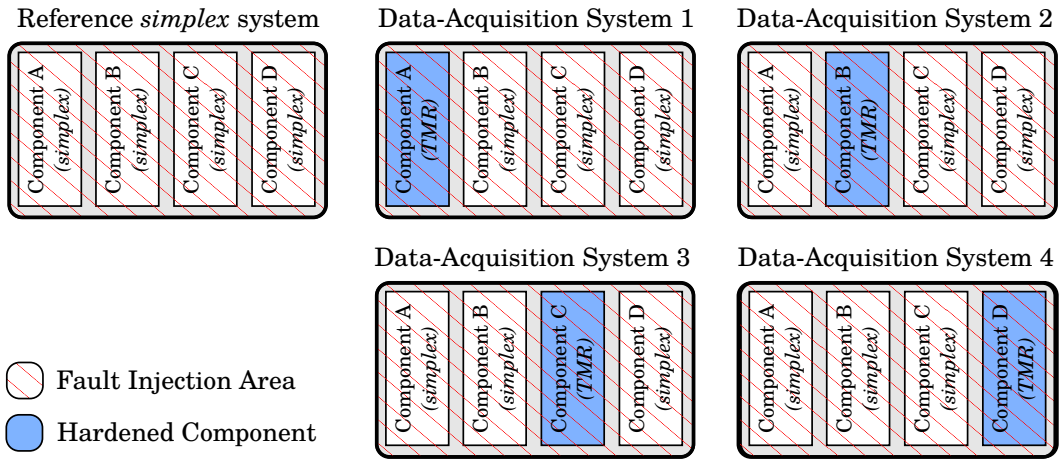

Figure 3.18: Illustration of the data acquisition for more complex systems by testing components inside the complete system (overcoming thus the need to model stimuli generators per each component).

is finished, it is possible to calculate the benefits of each particular modification. For example, for MTTF, this can be performed in three steps based on well-known equations, briefly described in the following text.

a) Particular *failure rates* of the data-acquisition systems, $\lambda_{simplex}$ and $\lambda_{sys\_c_m}$, $\forall c \in C$, $\forall m \in M$, are calculated using Equation 2.2.

b) It is known, that λ_{sys} of a system is a sum of λ_c, $\forall c \in C_{sys}$. This is formally described in Equation 2.1. Using this equation, the differences $\delta\lambda_{c_m}$ can be calculated for each FT mechanism and a system component. This $\delta\lambda_{c_m}$ then expresses the failure rate difference after applying the FT mechanism $m \in M$ to the component $c \in C$.

c) After that, the $\delta\lambda_{c_m}$ values are used to calculate the system λ_{sys} for systems in which multiple components are hardened by various FT mechanisms (i.e., not only one, as in the case of data-acquisition systems). The system failure rate calculation is performed by adding the sum of differences corresponding to the hardened systems $\delta\lambda_{c_m}$ and the $\lambda_{simplex}$. After that, this failure rate is converted back to the MTTF using Equation 2.2.

In more detail, such a calculation is described in the original Paper H, Section 4.2, alongside an overview of particular equations used in the analysis, including the error analysis of the estimation. It is important to note that this approach significantly lowers the number of measurements. Recall that a smaller number of systems is measured in advance, and then the SW interpolates all the possible variants. For example, a 10-component system in which each component can be hardened or unhardened (i.e., two states) would involve $2^{10} = 1024$ evaluations. If the proposed approach is used, only 10 data-acquisition systems are measured alongside the 1 reference (i.e., simplex) system. Measurement of the original number of systems would require months or years. With this accelerated approach, the measurement time is in the magnitude of weeks, thus, making the automated design possible.

The measured data-acquisition systems are presented in the original Paper H in Table 2. Using these data and the described interpolation process, the design space can be constructed and explored on Pareto-optimal solutions. The entire design space with such optimal solutions is also shown in the original Paper H in Figure 11. This figure reveals that the solutions form clusters on the chart. The first combination of parameters (i.e., MTTF

vs. power consumption) has 22 Pareto-optimal solutions, which are small or medium in size (as demonstrated by the color). The second combination (i.e., the MTTF vs. design size) possesses 26 Pareto-optimal solutions. These solutions are relatively power-efficient. The third pair (i.e., the power consumption vs. design size) has only 3 optimal solutions. This is because the size and power consumption are slightly dependent. From these optimal solutions, one solution was selected to analyze its properties further and test the design with a real scenario. As the design space construction utilizes the approximated data, the solution selected was again measured to validate the outcome.

Table 3.7: Configuration of the selected hardened GPDRC and its parameters.

	Configuration of Components														
GPDRC Variant	*input_register*	*roundrobin*	*errdec*	*harderr*	*safetyw*	*lookup_ent*	*adrcnt*	*fifo_ent*	*fsm*	*flash_controller*	Size [bits]	**MTTF (Estimated) [ms]**	**MTTF (Measured) [ms]**	**Power Consump. [mW]**	Frequency [MHz]
simplex	sim.	sim.	sim.	sim.	sim.	sim.	sim.	sim.	sim.	sim.	18 688	–	**24 703**	**20.47**	171.880
selected	**TMR**	**TMR**	sim.	**TMR**	**TMR**	sim.	**TMR**	sim.	sim.	sim.	33 984	**28 934**	**29 021**	**25.38**	130.016
CGTMR (ref.)	The whole GPDRC in coarse-grained TMR										60 672	–	**21 485**	**35.24**	170.445

Table 3.7 shows that the difference between the measured and approximated MTTF value is only 0.3 %, which is very promising. Table 3.7 also lists the components on which the TMR is applied to achieve higher MTTF times. Also, the simplex and CGTMR systems are added for comparison. The frequency, as a representative of a non-optimized parameter, shows that the selected system is circa 25 % slower than the original simplex system. As observed, the CGTMR keeps its speed, so the CGTMR approach does not prolong the *critical path*.

Testing on a Real Scenario Finally, the hardened GPDRC is built and measured. Although the measurement was performed on the target FPGA platform, it was estimated in a faked environment created by the FT-EST stimuli generator. Thus, it is essential to test the GPDRC in the natural environment. This means running the GPDRC alongside another system, which the GPDRC will harden (i.e., using bitstream renew). For this purpose, the benchmarks from the ITC'99 set [15] are used. Specifically, the *b01*, *b05*, and *b12* are used based on the available space on the FPGA. The *b01* represents an FSM that compares serial data flows, while the *b05* is a circuit elaborating memory contents. The *b12* is a simple guess-a-sequence game. These benchmarks are hardened using the TMR, and the GPDRC is used to renew their partial bitstreams. The block diagram of the system, which will be measured on MTTF, is displayed in Figure 3.19. In this case, the measurement is also performed by the FT-EST, but the renewed system is included this time, which is equal to the actual scenario of the GPDRC usage.

The measurement of the actual scenarios is shown in Table 3.8. Each measurement always includes the benchmark circuit in TMR repaired using a simplex version of the GPDRC. Then, the same benchmark in TMR, but this time repaired using the GPDRC, automatically hardened using the FT design automation method. Thus, the table allows comparing the benefits of the hardened GPDRC design. The column *MTTF (Testbed)* shows the actual value measured under the fault intensity of 2×10^{-5} inj/s/bit. The column *MTTF (Orbit)* presents the approximated MTTF on the orbital trajectory of the planet Earth in the height of 555.6 km with a 2.5-inch shielding made of aluminum. This approximation is calculated from the MTTF values measured using the testbed, based on the information obtained from [27]. Also, the design size corresponds to the GPDRC hard-

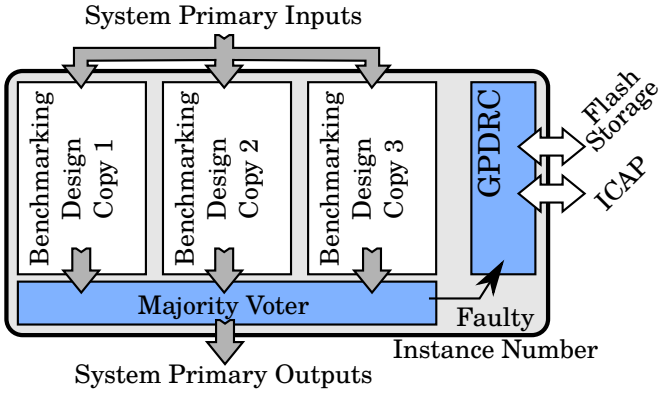

Figure 3.19: Block diagram of the resulting system, which demonstrates the benefits of the hardened GPDRC.

ening and the mean time between fault injections, which is based on the constant fault intensity. The best improvement in MTTF was achieved when the hardened GPDRC is used with the b12 benchmark. In this case, the MTTF increased by 11.7 %, while the design size overhead was 20.1 %. Also, the GPDRC effectiveness depends on the target benchmark that is repaired using the DPR. It is mainly its functionality but also its size, which is reflected in the results.

Table 3.8: The measurements of the GPDRCs in their natural environment on Virtex 5 technology, alongside the selected benchmark circuits from the ITC'99 set [15].

Benchmark Name	GPDRC Version	**Size [bits]**	Fault Intensity [inj/s/bit]	Mean Time Between FIs [ms]	**MTTF (Testbed) [ms]**	**MTTF (Orbit) [days]**	**FIs to Failure [-]**	**Difference (*simplex*)** **MTTF [%]**	**Size [%]**	**FIs to Fail. [%]**
b01 in TMR	simplex	**21 120**	2×10^{-5}	2 367.4	**371 477**	**17.69**	157	–	–	–
	selected	**36 224**	2×10^{-5}	1 380.3	**388 085**	**18.48**	271	**+4.5**	**+71.5**	**+79,0**
b05 in TMR	simplex	**61 248**	2×10^{-5}	816.4	**85 877**	**4.09**	105	–	–	–
	selected	**75 776**	2×10^{-5}	659.8	**89 673**	**4.27**	136	**+4.4**	**+23.7**	**+29.5**
b12 in TMR	simplex	**69 184**	2×10^{-5}	722.7	**158 613**	**7.55**	219	–	–	–
	selected	**83 072**	2×10^{-5}	601.9	**177 149**	**8.44**	294	**+11.7**	**+20.1**	**+34.2**

Next, the improvement in MTTF for future larger systems is presented. Unfortunately, it is currently impossible to measure larger systems on the Virtex 5 FPGAs utilized in this research, as space is the main limit factor. For this reason, the results in Table 3.9 are an extrapolation of the percentage improvement. The extrapolation targets the percentage improvement instead of the actual MTTF in seconds. This is to abstract from the need to measure a system with the simplex GPDRC to calculate the actual MTTF in seconds. Values for hypothetical systems of 100, 250 and 500 kbits are presented. The most important assumption is that with a larger system, the GPDRC overhead (in percentage) decreases as the MTTF grows further.

Key Publications

(H) Automated Design and Usage of the Fault-Tolerant Dynamic Partial Reconfiguration Controller for FPGAs, LOJDA Jakub, PÁNEK Richard, SEKANINA Lukáš, KOTÁSEK Zdeněk. In: *Microelectronics Reliability, vol. 2023, no. 144, pp. 1-16.* ISSN 0026-2714.

Table 3.9: Extrapolation of the percentage improvement in MTTF for larger designs with the simplex and hardened GPDRCs, based on the data previously measured on the Xilinx Virtex 5 FPGA technology.

Hypothetical Benchmark Size [bits]	GPDRC Version	**Size with GPDRC [bits]**	**Diff. (simplex) MTTF [%]**	**Size [%]**
100 000	simplex	**118 688**	–	–
	selected	**133 984**	**+13.9**	**+12.9**
250 000	simplex	**268 688**	–	–
	selected	**283 984**	**+29.1**	**+5.7**
500 000	simplex	**518 688**	–	–
	selected	**533 984**	**+54.4**	**+2.9**

3.4 FT Design Automation Overview

This section provides a brief overview of the proposed method devoted to the automated design of FT systems on FPGAs.

3.4.1 Design Flow

One of the goals of this research was to present an approach to FT design automation that would be general enough to cover multiple design languages and, thus, various levels of abstraction. This is why the design flow is currently proposed in two variants based on the properties of the input (i.e., original) design. Both of these flows, however, use code modification, FT mechanism selection strategy, and measurement.

It is also important to note that none of these flows use the evaluation of each candidate solution on the HW. In the flow representation, it could be said that the candidate evaluation was partly migrated to the SW. This is important, as it minimizes the number of measurements for larger systems, as the accurate FT measurement is still the most time-consuming part of the design. Thus, the automated FT design method is also suitable for larger systems.

Component-based Design Flow The first design flow targets designs that allow the separation of its components. This is, for example, a VHDL design, for which each component can be separated from the rest of the system without much effort and modification to the design itself. Moreover, to use this flow, each component must be tested and measured on all decision parameters separately. This mainly involves preparing multiple stimuli generators that authentically simulate each component's environment. Also, it's required to measure its size and power consumption (when used as a decision parameter). If these conditions are met, components are separated and measured individually after the available hardening mechanisms are applied to them.

Subsequently, data obtained during this measurement are used to model and approximate the individual candidate compositions of the resulting design. The approximated data serves for the FT mechanism selection strategy, aiming to select one or more optimal solutions. This approach comes from the finding (presented in Section 3.3.1) that to achieve a (power-, area-, etc.) efficient FT system, each component of the system must be treated using the hardening that suits the given component the most. The graphical representa-

tion of this design flow is shown in Figure 3.20. This flow corresponds to the one used in Section 3.3.4.

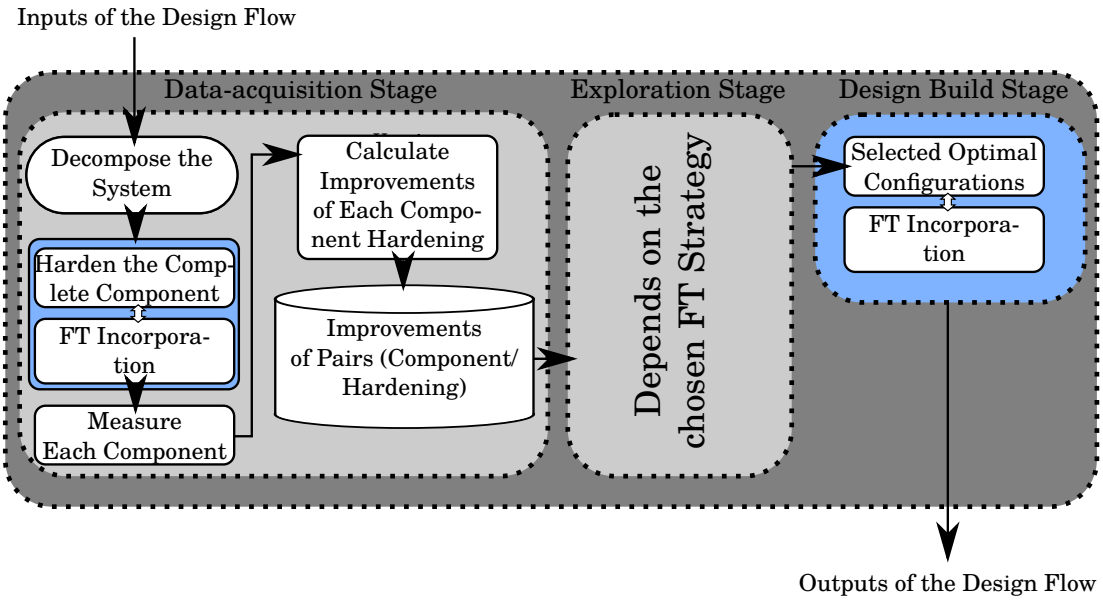

Figure 3.20: Design flow of the component-based approach.

Overview of the Component-based Design Flow

Intended for: Designs that can be separated into individual components

Advantages and Disadvantages

\+ Saves time for measurements (per each component and hardening, only a component itself is measured)

\- More demanding on creation of accurate stimulation data for components (which is manual work)

Compatibility with Presented Methods

FT Insertion: (From thesis-presented) VHDL; any other method targeting a component-separable description method

FT Strategy: Single-objective; Multi-objective

Measurement Methods: Reliability (FT-EST for MTTF, t50, critical bits ratio); power consumption and heat dissipation (external tool: Xilinx XPower Analyzer [29]); size (external tool: Xilinx ISE [80])

System-based Design Flow As opposed to the previous, the system-based flow targets designs whose structure is not easily separated. For example, this might be a complex design requiring much effort to construct the stimuli generators per each component. Or a system without a properly structured design that is not easily separated into components. Last but not least, this flow is also used with systems that utilize the algorithm-based

description (e.g., C++ code in combination with HLS), which does not allow for easily separating the algorithms and functions into components.

This flow keeps the system as a whole. Each part of the system (component, algorithm function, etc.; the appropriate granularity is left on the user) is hardened with each possible FT mechanism. The critical fact is that only one selected part is always hardened within the system. This way, the set of data-acquisition systems is prepared. These are measured, and the data obtained during these measurements are used to calculate and subsequently model the other candidate solutions, which already contain multiple parts treated with hardening. The graphical flow of this approach is in Figure 3.21. This approach was practically used in Section 3.3.6.

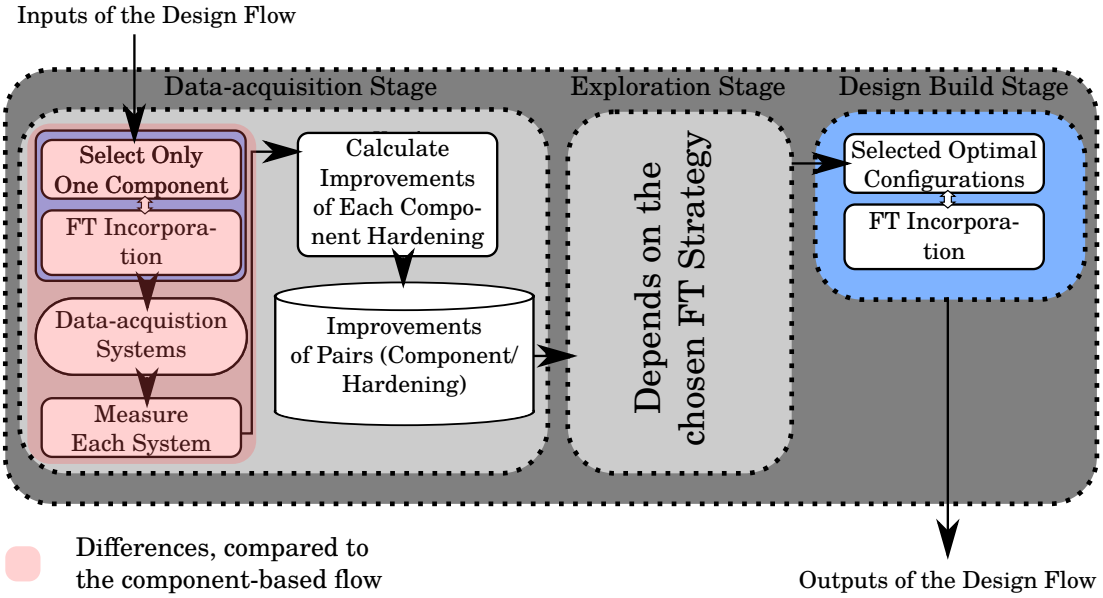

Figure 3.21: Design flow of the system-based approach.

Overview of the System-based Flow

Intended for: Designs that can not be separated into individual components

Advantages and Disadvantages

\+ Saves designer work (no need to analyze and design stimuli generators per each component)

\- More demanding on measurement time (a larger area is always tested – per each component and hardening pair, even the rest of the system is tested)

Compatibility with Presented Methods

FT Insertion: (From thesis-presented) C++, VHDL; any other method

FT Strategy: Single-objective; Multi-objective

Measurement Methods: Reliability (FT-EST for MTTF, t50, critical bits ratio);
power consumption and heat dissipation (external tool:
Xilinx XPower Analyzer [29]);
size (external tool: Xilinx ISE [80])

3.4.2 FT Mechanisms Insertion

To incorporate resiliency against faults, the design description code must be modified. This can be performed manually, but such a manual modification of static structures is time-consuming and increases the risk of incorporating design errors. For this reason, tools to modify the source code in a given language are presented. This thesis presents the two following code modifications, which must always be selected based on the language in which the design is written. This does not rule out the possibility of using an external tool for other language support.

C++ Code Modification The approach to the C++ code hardening presented in this thesis is based on data type modifications. These allow the incorporation of FT mechanisms into the C++ on the algorithm-level description. Such a modified algorithm is then synthesized using the HLS flow, and the resulting design (usually on RTL) is obtained, which can then be synthesized to the bitstream and uploaded to the FPGA.

The newly-created data types are provided in a library and are easily used by text-replacing the previously used data types in the code. The user of this approach can then precisely target the modifications toward a specific part of the code. This approach was presented in Section 3.3.1.

Overview of FT Insertion into a Design Described in C++

Intended for: Insertion of redundant data paths into an HLS-generated design

Advantages and Disadvantages

\+ Straightforward usage (only data types are modified in source code, after a special library is added)

\- Targets only the data paths, the control path (if, while constructs, etc.) remains unhardened

Compatibility with Presented Methods

FT Strategy: Single-objective; Multi-objective

Measurement Methods: Reliability (FT-EST for MTTF, t50, critical bits ratio);
power consumption and heat dissipation (external tool:
Xilinx XPower Analyzer [29]);
size (external tool: Xilinx ISE [80])

VHDL Code Modification This code modification targets the designs in VHDL, specifically its entity instantiations. The modifications to the design are written in the form of special comments around the entity instantiation, which is intended to be hardened. This way, the hardening is targeted toward specific components. After running the tool, the

given entities' names are changed to reference a newly created project file, which copies the original entity while inserting the given FT mechanism (e.g., TMR).

The VHDL modification is based on a template system to allow a straightforward extension of FT mechanisms applicable to the design entities by adding templates and support modification procedures (if needed). It was implemented in Python 3. The VHDL modification tries to abstract the context-sensitivity of the VHDL grammar to simplify the implementation. More details about this approach can be found in Section 3.3.5.

Overview of FT Insertion into a Design Described in VHDL

Intended for: Insertion of redundant structures into VHDL source code

Advantages and Disadvantages

\+ Straightforward usage (only comments are inserted into the source code)

\+ Straightforward addition of new approaches (template-based system)

\- Without further extension, it targets mostly spatial redundancy approaches only

Compatibility with Presented Methods

FT Strategy: Single-objective; Multi-objective

Measurement Methods: Reliability (FT-EST for MTTF, t50, critical bits ratio); power consumption and heat dissipation (external tool: Xilinx XPower Analyzer [29]); size (external tool: Xilinx ISE [80])

3.4.3 FT Mechanism Selection Strategy

The ability to modify a code is not enough to automate the FT design process. This is because the user still has to decide which hardening mechanism to apply to which component or part of a system. The strategies are algorithmic ways to solve this problem. This thesis presented two approaches, which must be selected based on the number of optimized parameters (e.g., design reliability, size, heat dissipation, etc.).

Single-objective optimization with Constraints In specific scenarios, the resulting design is optimized on one parameter (e.g., reliability) and constrained on another parameter (e.g., available chip area). For such situations, this approach is preferred. It bases on the *Multiple-choice Knapsack Problem* (MCKP), which has a well-known and documented formal base usable to solve this problem. This method always provides one optimal solution. Details were presented and discussed in Section 3.3.4.

Overview of the Single-objective Opt. Strategy

Intended for: Designs optimized on only one parameter (e.g., MTTF), while another parameter is constrained (e.g., size)

Advantages and Disadvantages

\+ The one optimal solution is found

- In practice, multiple parameters are optimized

Compatibility with Presented Methods

FT Insertion:	(From thesis-presented) C++, VHDL; any other method
Measurement Methods:	Reliability (FT-EST for MTTF, t50, critical bits ratio); power consumption and heat dissipation (external tool: Xilinx XPower Analyzer [29]); size (external tool: Xilinx ISE [80])

Multi-objective Optimization This strategy approach targets cases in which multiple parameters are optimized. In most cases, this approach will be used. This is because, in practice, nearly always multiple objectives are considered (e.g., reliability, power consumption, and size). This method utilizes a full design space search, which always targets optimal solutions. It is also important to note that for multi-objective optimization, finding only one (best) optimal solution is almost always impossible. Thus, a set of optimal solutions is provided by this method, and the user is required to choose a trade-off solution. The provided solutions are Pareto-optimal, and the user selects the best solution based on their preferences. This approach was used in Section 3.3.6.

Overview of the Multi-objective Opt. Strategy

Intended for:	Designs optimized on multiple parameters (e.g., MTTF, size and power consumption)

Advantages and Disadvantages

\+ Most usable for practice

\- Multiple optimal solutions are found, from which the user must select (given by the fundamental limits of multi-objective optimization)

Compatibility with Presented Methods

FT Insertion:	(From thesis-presented) C++, VHDL; any other method
Measurement Methods:	Reliability (FT-EST for MTTF, t50, critical bits ratio); power consumption and heat dissipation (external tool: Xilinx XPower Analyzer [29]); size (external tool: Xilinx ISE [80])

3.4.4 FT Evaluation

Last but not least, the FT level must be measured, which is the most time-consuming part of the complete flow. For this reason, this thesis proposed an automated testbed generator, which accepts a design and provides a fully-functional autonomous testbed, runnable on a target FPGA. Such a testbed is then controlled from a PC, and the measurements are collected and stored on a PC hard drive. On these measurements, statistical methods are applied to get relevant results.

To this moment, the measurement of critical bits number, sensitive bits ratio, MTTF, and the number of erroneous results per bitstream bit flip was measured using this approach.

A change of measured parameter is always straightforward, as the complete generic testbed is written in VHDL with a rich module structure, as was presented in Section 3.3.3.

Overview of the FT Evaluation Method

Intended for: Automated generation of accelerated and autonomous testbeds for FPGAs

Advantages and Disadvantages

\+ Saves time with preparation of measurements
\+ Heavily accelerated (multiple instances at once, at-speed testing, etc.)
\- For each design, a suitable stimuli generator must be prepared manually

Compatibility with Presented Methods

FT Insertion: (From thesis-presented) C++, VHDL; any other method
FT Strategy: Single-objective; Multi-objective

3.5 Publications

During the research on the topic of this thesis, numerous publications were presented, which were (co-)authored by the author of this thesis. The following section presents the summary of these publications closely related to the thesis. Subsequently, other publications still associated with this thesis's topic, whose results are not directly used in this thesis, are listed.

3.5.1 Selected Publications Summary

The research presented in this thesis is based on the following selected publications.

Paper A

LOJDA Jakub, PODIVÍNSKÝ Jakub, KOTÁSEK Zdeněk and KRČMA Martin. **Data Types and Operations Modifications: a Practical Approach to Fault Tolerance in HLS**. In: *Proceedings of IEEE East-West Design & Test Symposium*. Novi Sad: IEEE Computer Society, 2017, pp. 273-278. ISBN 978-1-5386-3299-4.

Abstract: Some types of electronic systems are working in the environment with an increased occurrence of faults such as space, aerospace or medical systems. Faults in these systems can lead to the failure of the whole system and can cause high economical losses or endanger human health. Fault tolerance is one of the techniques, the goal of which is to avoid such situations. This paper presents an approach to fault-tolerant data-paths design that is based on the modification of High-level Synthesis (HLS) input specification. The description and evaluation of the impacts of some HLS optimization methods are demonstrated in the paper as well. Higher reliability is achieved through the modification of input description in the C++ programming language, which the HLS synthesis tools are based on. Our work targets SRAM-based FPGAs that are prone to Single Event Upsets (SEUs). For the evaluation of the proposed method we use our evaluation platform, which allows

us to analyze fault tolerance properties of the Design Under Test (DUT). The evaluation platform is based on functional verification in combination with fault injection.

Author Participation: 52 %

Citations: 2

Conference Rank: *unknown*

Paper B

LOJDA Jakub, PODIVÍNSKÝ Jakub and KOTÁSEK Zdeněk. **Redundant Data Types and Operations in HLS and their Use for a Robot Controller Unit Fault Tolerance Evaluation**. In: *Proceedings of IEEE East-West Design & Test Symposium*. Novi Sad: IEEE Computer Society, 2017, pp. 359-364. ISBN 978-1-5386-3299-4.

Abstract: Some environments (e.g. space, aerospace or medical systems) require electronic systems to withstand an increased occurrence of faults. Moreover, the failure of these electronic systems can cause high economical losses or endanger human health. Fault tolerance is one of the techniques, the goal of which is to avoid such situations. This paper presents an approach to evaluate the degree of importance of individual system partitions when High-Level Synthesis (HLS) methodology is used. The importance of individual partitions was evaluated by the usage of our approach to fault-tolerant data-paths design which is based on the HLS input specification modification. The partitions are formed by sets of variables and operations. A brief description of the approach to fault tolerance in HLS is shown in the paper as well. Our experiments are evaluated using an SRAM-based FPGA evaluation platform which allows us to analyze fault tolerance properties of the Design Under Test (DUT). In the evaluation platform, functional verification in combination with fault injection is utilized.

Author Participation: 57 %

Citations: 0

Conference Rank: *unknown*

Paper C

LOJDA Jakub, PODIVÍNSKÝ Jakub, KOTÁSEK Zdeněk and KRČMA Martin. **Majority Type and Redundancy Level Influences on Redundant Data Types Approach for HLS**. In: *2018 16th Biennial Baltic Electronics Conference (BEC)*. Tallinn: IEEE Computer Society, 2018, pp. 1-4. ISBN 978-1-5386-7312-6.

Abstract: Due to the increasing demand for reliable computation in environments that require electronic systems to withstand an increased occurrence of faults (e.g. space, aerospace and medicine), new techniques of the so-called Fault Tolerance insertion arise. From another perspective, today's systems have become incredibly large and complex. New methodologies (e.g. High-Level Synthesis) are used to reduce time to market and simplify the verification of the resulting system. In our research we focus on an implementation of Fault Tolerance into complex systems with the usage of High-Level Synthesis. In our approach, we

are using newly designed Data Types that introduce redundancy on the functional level of an algorithm. In this student paper, our previously presented technique is extended by another means of redundancy and also by a new type of voting component. The systems incorporating various levels of redundancies using our approach are experimentally tested on the application of a robot controller. The paper also briefly presents the evaluation process and investigates its correct settings. The results show that the bit-based majority function is more suitable for usage with our Redundant Data Types.

Author Participation: 45 %

Citations: 0

Conference Rank: *unknown*

Paper D

LOJDA Jakub, PODIVÍNSKÝ Jakub, ČEKAN Ondřej, PÁNEK Richard and KOTÁSEK Zdeněk. **FT-EST Framework: Reliability Estimation for the Purposes of Fault-Tolerant System Design Automation**. In: *Proceedings of the 2018 21st Euromicro Conference on Digital System Design*. Praha: IEEE Computer Society, 2018, pp. 244-251. ISBN 978-1-5386-7376-8.

Abstract: The complexity of today's systems is growing along with the level of chip integration. This results in higher demand for reliability techniques; it also increases the difficulty of incorporating reliability in such systems. For this purpose, we are working on a method to automate reliability insertion; however, for this method, it is necessary to have feedback on the result. In this paper, one component of the automation flow enabling the estimation of the resulting reliability - Fault Tolerance ESTimation (FT-EST) framework - is presented along with an improvement for accelerating the time necessary to reach the estimation. For the purpose of evaluation, we are using our Redundant Data Types approach, which enables us to intentionally insert reliability in a particular operation. The estimation utilizes the concept of fault injection. The results indicate, that the concept of Redundant Data Types is functional, however, also suggest its future improvements (e.g. for the operation of subtraction).

Author Participation: 35 %

Citations: 0

Conference Rank: B1 (Qualis)

Paper E

LOJDA Jakub, PODIVÍNSKÝ Jakub, ČEKAN Ondřej, PÁNEK Richard, KRČMA Martin and KOTÁSEK Zdeněk. **Automatic Design of Reliable Systems Based on the Multiple-choice Knapsack Problem**. In: *Proceedings - 2020 23rd International Symposium on Design and Diagnostics of Electronic Circuits and Systems, DDECS 2020*. Novi Sad: Institute of Electrical and Electronics Engineers, 2020, pp. 1-4. ISBN 978-1-7281-9938-2.

Abstract: This paper evaluates the practical usage of the Multiple-choice Knapsack Problem (MCKP) solver to automatically select the proper fault mitigation method for each component to maximize the overall fault tolerance of the whole system. The usage of the MCKP is placed into the context with our fault tolerance automation toolkit, the goal of which is to completely automate the process of fault-tolerant system design on a very general level. To achieve our goal, we present our research on Field Programmable Gate Arrays (FPGAs) for which we have developed the specific components in order to support their fault-tolerant design automation. In our particular case study, the MCKP method on the partitioned system was able to find the solution with 18 % less critical bits compared to our previous approach, while even lowering the circuit size. The results indicate that by splitting the system into smaller components and applying the MCKP method, considerably better results in terms of critical bits representation can be achieved.

Author Participation: 40 %

Citations: 1

Conference Rank: B3 (Qualis)

Paper F

LOJDA Jakub, PÁNEK Richard and KOTÁSEK Zdeněk. **Automatic Design of Fault-Tolerant Systems for VHDL and SRAM-based FPGAs**. In: *Proceedings - 2021 24th Euromicro Conference on Digital System Design, DSD 2021*. Palermo: Institute of Electrical and Electronics Engineers, 2021, pp. 549-552. ISBN 978-1-6654-2703-6.

Abstract: This paper presents and evaluates the possibility of automatic design of fault-tolerant systems from unhardened systems. We present an overview of our toolkit with its three main components: 1) fault-tolerant structures insertion (which we call helpers); 2) fault-tolerant structures selection (called guiders); and 3) automatic testbed generation, incorporating advanced acceleration techniques to accelerate the test and evaluation. Our approach is targeting complete independence on the HW description language and its abstraction level, however, for our case study, we focus on VHDL in combination with fine-grained n-modular redundancy. In the case study part of this paper, we proved that it is undoubtedly beneficial to select a proper fault tolerance method for each partition separately. Three experimental systems were developed with the usage of our method. Two of them achieved better reliability parameter while even lowering their chip area, compared to static allocation of equivalent fault tolerance technique type. In the case study, we target the best median time to failure, the so-called t50, however, our method is not dependent on this parameter and arbitrary optimization target can be selected, as soon as it is measurable.

Author Participation: 60 %

Citations: 0

Conference Rank: B1 (Qualis)

Paper G

LOJDA Jakub, PÁNEK Richard and KOTÁSEK Zdeněk. **Automatically-Designed Fault-Tolerant Systems: Failed Partitions Recovery**. In: *2021 IEEE East-West Design and Test Symposium, EWDTS 2021 - Proceedings*. Batumi: Institute of Electrical and Electronics Engineers, 2021, pp. 26-33. ISBN 978-1-6654-4503-0.

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.

Author Participation: 60 %

Citations: 1

Conference Rank: *unknown*

Paper H

LOJDA Jakub, PÁNEK Richard, SEKANINA Lukáš and KOTÁSEK Zdeněk. **Automated Design and Usage of the Fault-Tolerant Dynamic Partial Reconfiguration Controller for FPGAs**. *Microelectronics Reliability, vol. 2023, no. 144, pp. 1-16.* ISSN 0026-2714.

Abstract: This article presents a new design automation method for Fault-Tolerant (FT) systems implemented on dynamically reconfigurable Field Programmable Gate Arrays (FPGAs). The method aims at minimizing the human interactions needed to incorporate FT mechanisms into an existing system. It starts with a source code of an original unhardened circuit. It continues by automated manipulation of the source code, algorithmic strategic selection of suitable FT techniques, design space exploration of candidate FT implementations, and selection of the resulting implementation. The method also includes efficient evaluation of achieved FT parameters performed on the target HW. As a novel approach working on the level of HW description languages is employed, the code modification is separated, which differentiates our method from others. The case study utilizing this method targets the design of an experimental FT dynamic partial reconfiguration controller for an FPGA. This controller is helpful for the restoration of faulty components due to a single-event upset on an FPGA. We used the method to generate a set of Pareto-optimal controllers concerning the design's Mean Time to Failure (MTTF) parameter, power consumption, and size. Then, the FT controller is connected to several benchmark circuits, and the reliability parameters are evaluated at the entire system level. Our results show

that by replacing the standard reconfigurable controller with our automatically-designed FT controller for one specific benchmark, the design size increased by 20.1 %, and MTTF increased by 11.7 %. However, the efficiency is highly dependent on the target system size, MTTF, and circuit functionality. We also estimate that a complex system defined by half a million configuration bits would improve MTTF by more than 50 %.

Author Participation: 45 %

Citations: 0

Impact Factor: 1.418

3.5.2 Author's Contributions to The Selected Publications

All the selected publications were created in the Dependable Systems Research Group at the Department of Computer Systems, the Faculty of Information Technology, Brno University of Technology. Although the publications were made in cooperation with colleagues, the author added the main contribution to these selected publications, which is also the main contribution to this thesis. The following list names the main contributions per each of the selected publications.

- **Paper A** – design and implementation of fault tolerance insertion methods and principles of their connections for designs written in C++; preparation and performing the experiments; evaluation of measured data; writing most of the paper.
- **Paper B** – design, implementation and performing the experiments; evaluation of measured data; writing most of the paper.
- **Paper C** – implementation of additional fault tolerance mechanisms for the insertion; preparation and performing the experiments; evaluation of measured data incl. their accuracy; design of fault injection unit, which considers the size of the design under test; writing most of the paper.
- **Paper D** – design and implementation of the framework for automated creation of testbeds; preparation of benchmarking circuits; preparation and performing the experiments; evaluation of measured data; writing most of the paper.
- **Paper E** – the transformation of the fault tolerance allocation problem to the multiple-choice knapsack problem; implementation of a simple solver; preparation and performing the experiments on benchmark circuits; evaluation of measured data and improvements in critical bits representation; writing most of the paper.
- **Paper F** – design and implementation of new fault tolerance insertion methods for designs described in VHDL; preparation and performing the experiments; evaluation of measured data; writing most of the paper.
- **Paper G** – preparation and performing the experiments; evaluation of measured data; writing most of the paper.
- **Paper H** – summary of the research; incorporation of the power consumption estimation into the method; implementation of the multi-criteria optimization; design of the concepts of the experimental case study; case study data measurement; writing most of the paper.

3.5.3 Other Topic-Related Publications

The following list contains the rest of the author's publications. These are thematically related to the thesis topic. The contributions in them are, however, not fundamental to the thesis. However, the experiences gained in work on these publications were significant to progress on the main thesis topic.

2023

- PÁNEK Richard and LOJDA Jakub. **The Fault-tolerant Single-FPGA Systems with a Self-repair Reconfiguration Controller**. In: *2023 IEEE 14th Latin America Symposium on Circuits and Systems (LASCAS)*. Quito: Institute of Electrical and Electronics Engineers, 2023, pp. 104-107. ISBN 978-1-6654-5705-7.

2021

- LOJDA Jakub, PODIVÍNSKÝ Jakub, ČEKAN Ondřej and KOTÁSEK Zdeněk. **Accelerating Tests of Arithmetic Circuits Through On-FPGA Stimuli Generation and Their Reduction**. In: *International Conference on Electrical, Computer, Communications and Mechatronics Engineering, ICECCME 2021*. Mauritius: Institute of Electrical and Electronics Engineers, 2021, pp. 1628-1633. ISBN 978-1-6654-1262-9.

- PÁNEK Richard, LOJDA Jakub, PODIVÍNSKÝ Jakub and KOTÁSEK Zdeněk. **Reliability Analysis of the FPGA Control System with Reconfiguration Hardening**. In: *Proceedings - 2021 24th Euromicro Conference on Digital System Design, DSD 2021*. Palermo: Institute of Electrical and Electronics Engineers, 2021, pp. 553-556. ISBN 978-1-6654-2703-6.

- LOJDA Jakub, PÁNEK Richard, PODIVÍNSKÝ Jakub, ČEKAN Ondřej, KRČMA Martin and KOTÁSEK Zdeněk. **Testing Embedded Software Through Fault Injection: Case Study on Smart Lock**. In: *2021 IEEE 22nd Latin American Test Symposium, LATS 2021*. Punta del Este: Institute of Electrical and Electronics Engineers, 2021, pp. 80-85. ISBN 978-1-6654-2057-0.

2020

- LOJDA Jakub, PÁNEK Richard, PODIVÍNSKÝ Jakub, ČEKAN Ondřej, KRČMA Martin and KOTÁSEK Zdeněk. **Analysis of Software-Implemented Fault Tolerance: Case Study on Smart Lock**. In: *2020 IEEE East-West Design and Test Symposium, EWDTS 2020 - Proceedings*. Varna: Institute of Electrical and Electronics Engineers, 2020, pp. 24-28. ISBN 978-1-7281-9899-6.

- PODIVÍNSKÝ Jakub, LOJDA Jakub, PÁNEK Richard, ČEKAN Ondřej, KRČMA Martin and KOTÁSEK Zdeněk. **Evaluation Platform For Testing Fault Tolerance: Testing Reliability of Smart Electronic Locks**. In: *2020 IEEE 11th Latin American Symposium on Circuits & Systems (LASCAS)*. San José: IEEE Circuits and Systems Society, 2020, pp. 1-4. ISBN 978-1-7281-3427-7.

- LOJDA Jakub, PÁNEK Richard, PODIVÍNSKÝ Jakub, ČEKAN Ondřej, KRČMA Martin and KOTÁSEK Zdeněk. **Hardening of Smart Electronic Lock Software**

against Random and Deliberate Faults. In: *Proceedings - Euromicro Conference on Digital System Design, DSD 2020*. Kranj: Institute of Electrical and Electronics Engineers, 2020, pp. 680-683. ISBN 978-1-7281-9535-3.

- PÁNEK Richard, LOJDA Jakub, PODIVÍNSKÝ Jakub and KOTÁSEK Zdeněk. **Reliability Analysis of Reconfiguration Controller for FPGA-Based Fault Tolerant Systems: Case Study**. In: *2020 International Symposium on VLSI Design, Automation, and Test (VLSI-DAT) : proceedings of technical papers*. Hsinchu: IEEE Computer Society, 2020, pp. 121-124. ISBN 978-1-7281-6083-2.

2019

- KRČMA Martin, KOTÁSEK Zdeněk and LOJDA Jakub. **Detecting hard synapses faults in artificial neural networks**. In: *20th IEEE Latin American Test Symposium (LATS 2019)*. Santiago de Chile: IEEE Computer Society, 2019, pp. 1-6. ISBN 978-1-7281-1756-0.

- PODIVÍNSKÝ Jakub, LOJDA Jakub and KOTÁSEK Zdeněk. **Extended Reliability Analysis of Fault-Tolerant FPGA-based Robot Controller**. In: *20th IEEE Latin American Test Symposium (LATS 2019)*. Santiago: IEEE Computer Society, 2019, pp. 97-100. ISBN 978-1-7281-1756-0.

- LOJDA Jakub, PODIVÍNSKÝ Jakub and KOTÁSEK Zdeněk. **Reliability Indicators for Automatic Design and Analysis of Fault-Tolerant FPGA Systems**. In: *20th IEEE Latin American Test Symposium (LATS 2019)*. Santiago: IEEE Computer Society, 2019, pp. 93-96. ISBN 978-1-7281-1756-0.

- ČEKAN Ondřej, PODIVÍNSKÝ Jakub, LOJDA Jakub, PÁNEK Richard, KRČMA Martin and KOTÁSEK Zdeněk. **Smart Electronic Locks and Their Reliability**. In: *Proceedings of the 7th Prague Embedded Systems Workshop*. Roztoky u Prahy: Czech Technical University, 2019, pp. 4-5. ISBN 978-80-01-06607-2.

- ČEKAN Ondřej, PODIVÍNSKÝ Jakub, LOJDA Jakub, PÁNEK Richard, KRČMA Martin and KOTÁSEK Zdeněk. **Testing Reliability of Smart Electronic Locks: Analysis and the First Steps Towards**. In: *Proceedings of the 2019 22nd Euromicro Conference on Digital System Design*. Kalithea: Institute of Electrical and Electronics Engineers, 2019, pp. 506-513. ISBN 978-1-7281-2861-0.

2018

- PODIVÍNSKÝ Jakub, LOJDA Jakub and KOTÁSEK Zdeněk. **An Experimental Evaluation of Fault-Tolerant FPGA-based Robot Controller**. In: *Proceedings of IEEE East-West Design & Test Symposium*. Kazan: IEEE Computer Society, 2018, pp. 63-69. ISBN 978-1-5386-5710-2.

- LOJDA Jakub and KOTÁSEK Zdeněk. **Automatizace návrhu spolehlivých systémů a její dílčí komponenty**. In: *Počítačové architektury & diagnostika 2018*. Stachy: University of West Bohemia in Pilsen, 2018, pp. 5-8. ISBN 978-80-261-0814-6.

- PODIVÍNSKÝ Jakub, LOJDA Jakub, ČEKAN Ondřej and KOTÁSEK Zdeněk. **Evaluation Platform for Testing Fault Tolerance Properties: Soft-core Processor-based Experimental Robot Controller**. In: *Proceedings of the 2018 21st Euromicro Conference on Digital System Design*. Praha: IEEE Computer Society, 2018, pp. 229-236. ISBN 978-1-5386-7376-8.

- LOJDA Jakub and KOTÁSEK Zdeněk. **Fault Tolerance in HLS for the Purposes of Reliable System Design Automation**. In: *Proceedings of the 6th Prague Embedded Systems Workshop*. Roztoky u Prahy: Faculty of Information Technology, Czech Technical University, 2018, pp. 31-32. ISBN 978-80-01-06456-6.

- LOJDA Jakub, PODIVÍNSKÝ Jakub and KOTÁSEK Zdeněk. **Fault Tolerance Properties of Systems Generated with the Use of High-Level Synthesis**. In: *Proceedings of IEEE East-West Design & Test Symposium*. Kazan: IEEE Computer Society, 2018, pp. 80-86. ISBN 978-1-5386-5710-2.

- PODIVÍNSKÝ Jakub, LOJDA Jakub and KOTÁSEK Zdeněk. **FPGA-based Robot Controller: An Experimental Evaluation of Fault Tolerance Properties**. In: *INFORMAL PROCEEDINGS 21st IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems*. Budapešť, 2018, pp. 9-12.

- PÁNEK Richard, LOJDA Jakub, PODIVÍNSKÝ Jakub and KOTÁSEK Zdeněk. **Partial Dynamic Reconfiguration in an FPGA-based Fault-Tolerant System: Simulation-based Evaluation**. In: *Proceedings of IEEE East-West Design & Test Symposium*. Kazaň: IEEE Computer Society, 2018, pp. 129-134. ISBN 978-1-5386-5710-2.

2017

- LOJDA Jakub and KOTÁSEK Zdeněk. **A Basic Approach to Fault Tolerance of Data Paths of HLS-synthesized Systems and its Evaluation**. In: *Proceedings of the 5th Prague Embedded Systems Workshop*. Roztoky u Prahy: Faculty of Information Technology, Czech Technical University, 2017, pp. 79-80. ISBN 978-80-01-06178-7.

- LOJDA Jakub and KOTÁSEK Zdeněk. **Automatizace návrhu systémů odolných proti poruchám pomocí vysokoúrovňové syntézy**. In: *Počítačové architektury & diagnostika 2017*. Smolenice: Slovak University of Technology in Bratislava, 2017, pp. 59-62. ISBN 978-80-972784-0-3.

- KRČMA Martin, KOTÁSEK Zdeněk and LOJDA Jakub. **Comparison of FPNNs Models Approximation Capabilities and FPGA Resources Utilization**. In: *Proceedings of IEEE 13th International Conference on Intelligent Computer Communication and Processing*. Cluj-Nappoca: IEEE Computer Society, 2017, pp. 125-132. ISBN 978-1-5386-3368-7.

- PODIVÍNSKÝ Jakub, ČEKAN Ondřej, LOJDA Jakub, ZACHARIÁŠOVÁ Marcela, KRČMA Martin and KOTÁSEK Zdeněk. **Functional Verification Based Platform for Evaluating Fault Tolerance Properties**. *Microprocessors and Microsystems*, vol. 52, no. 5, 2017, pp. 145-159. ISSN 0141-9331.

- PODIVÍNSKÝ Jakub, LOJDA Jakub, ČEKAN Ondřej, PÁNEK Richard and KOTÁSEK Zdeněk. **Reliability Analysis and Improvement of FPGA-based Robot Controller**. In: *Proceedings of the 2017 20th Euromicro Conference on Digital System Design*. Vídeň: IEEE Computer Society, 2017, pp. 337-344. ISBN 978-1-5386-2145-5.

- KRČMA Martin, LOJDA Jakub and KOTÁSEK Zdeněk. **Triple Modular Redundancy Used in Field Programmable Neural Networks**. In: *Proceedings of IEEE East-West Design & Test Symposium*. Novi Sad: IEEE Computer Society, 2017, pp. 1-6. ISBN 978-1-5386-3299-4.

2016

- LOJDA Jakub and KOTÁSEK Zdeněk. **A Systematic Approach to the Description of Fault-tolerant Systems**. *Proceedings of the 4th Prague Embedded Systems Workshop*. Roztoky u Prahy, 2016.

- KRČMA Martin, KOTÁSEK Zdeněk, LOJDA Jakub and KAŠTIL Jan. **Comparison of FPNNs models approximation capabilities and resources utilization**. In: *Proceedings of the Work in progress Session held in connection with DSD 2016*. Limassol: Johannes Kepler University Linz, 2016, pp. 1-2. ISBN 978-3-902457-46-2.

- PODIVÍNSKÝ Jakub, ČEKAN Ondřej, LOJDA Jakub and KOTÁSEK Zdeněk. **Functional Verification as a Tool for Monitoring Impact of Faults in SRAM-based FPGAs**. In: *Proceedings of the 2016 International Conference on Field Programmable Technology*. Xi'an: IEEE Computer Society, 2016, pp. 293-294. ISBN 978-1-5090-5602-6.

- LOJDA Jakub, PODIVÍNSKÝ Jakub, KRČMA Martin and KOTÁSEK Zdeněk. **HLS-based Fault Tolerance Approach for SRAM-based FPGAs**. In: *Proceedings of the 2016 International Conference on Field Programmable Technology*. Xi'an: IEEE Computer Society, 2016, pp. 301-302. ISBN 978-1-5090-5602-6.

- KRČMA Martin, KOTÁSEK Zdeněk and LOJDA Jakub. **Implementation of Fault Tolerant Techniques into FPNNs**. In: *Proceedings of the 2016 International Conference on Field Programmable Technology*. Xi'an: IEEE Computer Society, 2016, pp. 297-298. ISBN 978-1-5090-5602-6.

- PODIVÍNSKÝ Jakub, ČEKAN Ondřej, LOJDA Jakub and KOTÁSEK Zdeněk. **Verification of Robot Controller for Evaluating Impacts of Faults in Electromechanical Systems**. In: *Proceedings of the 19th Euromicro Conference on Digital Systems Design*. Limassol: IEEE Computer Society, 2016, pp. 487-494. ISBN 978-1-5090-2816-0.

3.6 Research Projects, Grants

2020

- *Design, Optimization and Evaluation of Application Specific Computer Systems*, BUT, FIT-S-20-6309, 2020-2022, start: 2020-03-01, end: 2022-12-31

2019

- *Electromobile charger - 4th stage*, KPB INTRA s.r.o., 2019, start: 2019-04-01, end: 2019-08-31

2018

- *SECREDAS - Product Security for Cross Domain Reliable Dependable Automated Systems*, ECSEL JU, 8A18014, Proposal ID 783119-2, 2018-2021, start: 2018-05-01, end: 2021-04-30

2017

- *Advanced parallel and embedded computer systems*, BUT, FIT-S-17-3994, 2017-2019, start: 2017-03-01, end: 2019-12-31

2016

- *IT4Innovations excellence in science*, MŠMT CR, LQ1602, 2016-2020, start: 2016-01-01, end: 2020-12-31

2015

- *Algorithms, Design Methods, and Many-Core Execution Platform for Low-Power Massive Data-Rate Video and Image Processing*, Artemis JU, 7H14002, 621439, 2014-2017, start: 2014-04-01, end: 2017-06-30
- *Architecture of parallel and embedded computer systems*, BUT, FIT-S-14-2297, 2014-2016, start: 2014-03-01, end: 2016-12-31

Chapter 4

Conclusions

The main goal of this research thesis was to examine the possibilities of automated transformation of digital system descriptions into their hardened versions. The approach was targeted towards generality. It should apply to various design description languages, design strategies, and abstraction levels. To achieve the generality of the process, four goals were defined:

Goal 1. *FT Mechanisms Incorporation:* To achieve the main research target, a possibility to automatically modify a design description must exist within the design automation method (and its toolkit implementation).

The Goal 1 was addressed by creating the *Redundant Data Types* – particular data types for the C++ language and HLS, which allow a selection of a hardening mechanism on a per-variable basis. These new data types were presented in Section 3.3.1 and the corresponding Paper A. Not only one FT incorporation method was proposed. In Section 3.3.5 of this thesis and the corresponding Paper F, the approach to modify VHDL descriptions was presented, which was later practically demonstrated.

Goal 2. *FT Strategy Methods:* Only the source code modification is not enough. A specific strategy (or logic) must be incorporated into the design automation method, which selects the proper modifications based on the target parameters of the design.

This Goal 2 was addressed by implementing two selection strategies. The first strategy targets cases in which only one target parameter of the design is optimized while another parameter is constrained (e.g., FT parameters are optimized within a constrained design size). This strategy uses the MCKP solver internally to select the best solution; it was presented in Section 3.3.4 and Paper E. The second strategy allows the optimization of multiple target parameters simultaneously (e.g., FT parameter, size, and power consumption). In such a case, however, multiple optimal solutions might be provided (according to their Pareto-optimality). This second mechanism of FT selection strategy was presented in Section 3.3.6 and its corresponding Paper H.

Goal 3. *FT Evaluation:* To base the automated design on accurate data, measurement of FT properties on the target platform is critical. It is, however, the most time-consuming part of the complete design flow. For this reason, the FT evaluation must use all the available means to accelerate the measurement.

Goal 3 was solved by designing and implementing the FT-EST framework. This framework allows the measurement of FT properties of FPGA designs. The main concept is

based on semi-automatically generated testbeds, which are subsequently synthesized and uploaded to the target FPGA. The testbeds are prepared for communication with the SW part on a PC, which stores the data. Later, statistical calculations are performed to quantify the achieved level of FT. This FT-EST framework was described in Section 3.3.3 and its corresponding Paper D.

Goal 4. *Design Flow:* The complete design flow is proposed based on the previously mentioned modules. However, specific cases might require using a particular flow variant.

The Goal 4 is straightforward. The decomposition is performed for systems that can be decomposed (i.e., partitioned) on individual components, and individual component parameters are measured. Data obtained after the measurement are then used to interpolate all the possible solutions, thus, eliminating the number of measurements, which significantly accelerates the automated design. This approach to design flow was presented in Section 3.3.5 and Papers F and G. However, partitioning is impossible for certain design types or code modification approaches. For example, a system implemented as a single unit in C++ (and synthesized using HLS) does not allow a simple system decomposition. For this reason, another flow is proposed, which allows for keeping the system as a whole. This is the design flow used in Section 3.3.6 and Paper H.

4.1 Thesis Main Contributions Summary

The following contributions of this thesis can be considered the most important:

Contribution 1. *The automated design method* is the first contribution of this thesis. The automated design method includes various modules which can be used and extended in further experimentation. Besides this, the method was implemented in the form of a toolkit.

Contribution 2. *Proving that a general approach to the automated design of FT systems is possible* is another significant contribution of this thesis. Multiple approaches to automated FT design always utilize their specific solutions. Knowing that a unified approach is possible is beneficial because such an approach allows the re-usability of the incorporated modules (i.e., design methods).

Contribution 3. *Several experimental works* and data measurements were performed during the design of this method. These experiments provided data needed to complete the automated process and various other knowledge presented in the included papers.

4.2 Future Research Possibilities

This thesis deals with the automated design of FT systems, a relatively broad concept. During the research, various branches emerged. Some of them were dead ends. The others were interesting. However, they moved away from the original target of the research. The following text presents the noteworthy ones, which I believe would be beneficial to research further and may lead to interesting results.

1. *SW-implemented FT*: It might be interesting to utilize this method for the so-called SW-implemented FT. Specific steps towards this were already performed, which were not part of this thesis. These include the creation of a testing platform that executes target binaries

on a remote system of a possibly different CPU architecture, which was presented in [44]. The design automation toolkit might utilize this testing platform. The redundant data types approach might be usable for FT mechanism insertion, except their usage would not be applied in combination with HLS but instead on a program intended for CPU execution. The strategy selection methods might be reused.

2. *Extension of FT mechanisms:* It might be interesting to extend the methods of FT mechanism incorporation by newly emerging approaches that target a specific efficiency (e.g., area-efficient FT mechanisms).

3. *Testing of new FT mechanisms:* It would be interesting to find out whether the complete design automation toolkit could be used differently – to help in the creation and testing of new means of FT. This is because it includes everything from testing to FT incorporation mechanisms, which would save a significant amount of work for a beginning researcher.

Bibliography

[1] Agiakatsikas, D., Lee, G., Mitchell, T., Cetin, E. and Diessel, O. From C to Fault-Tolerant FPGA-Based Systems. In: IEEE. *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 2018, p. 212–212. ISBN 978-1-5386-5522-1.

[2] Alderighi, M., Casini, F., d'Angelo, S., Mancini, M., Pastore, S. et al. Evaluation of Single Event Upset Mitigation Schemes for SRAM-based FPGAs Using the FLIPPER Fault Injection Platform. In: IEEE. *Defect and Fault-Tolerance in VLSI Systems, 2007. DFT'07. 22nd IEEE International Symposium on*. 2007, p. 105–113. ISBN 978-0-7695-2885-4.

[3] Alderighi, M., D'Angelo, S., Mancini, M. and Sechi, G. R. A Fault Injection Tool for SRAM-based FPGAs. In: IEEE. *On-Line Testing Symposium, 2003. IOLTS 2003. 9th IEEE*. 2003, p. 129–133. ISBN 0-7695-1968-7.

[4] Anwer, J., Platzner, M. and Meisner, S. FPGA Redundancy Configurations: An Automated Design Space Exploration. In: IEEE. *2014 IEEE International Parallel & Distributed Processing Symposium Workshops*. 2014, p. 275–280. DOI: 10.1109/IPDPSW.2014.37. ISBN 978-1-4799-4116-2.

[5] Ashenden, P. J. *The VHDL Cookbook, First Edition*. University of Adelaide, South Australia, 1990.

[6] Aviziens, A. Fault-Tolerant Systems. *IEEE Transactions on Computers*. 1976, C-25, no. 12, p. 1304–1312. DOI: 10.1109/TC.1976.1674598.

[7] Benso, A., Bosio, A., Di Carlo, S. and Mariani, R. A Functional Verification based Fault Injection Environment. In: IEEE. *Defect and Fault-Tolerance in VLSI Systems, 2007. DFT'07. 22nd IEEE International Symposium on*. 2007, p. 114–122. ISBN 978-0-7695-2885-4.

[8] Bernardeschi, C., Cassano, L., Domenici, A. and Sterpone, L. Accurate Simulation of SEUs in the Configuration Memory of SRAM-based FPGAs. In: IEEE. *Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2012 IEEE International Symposium on*. 2012, p. 115–120. ISBN 978-1-4673-3044-2.

[9] Brigham Young University. *BYU EDIF Tools Homepage* [online]. Brigham Young University [cit. 2022-08-02]. Available at: http://reliability.ee.byu.edu/edif/.

[10] Canis, A., Choi, J., Aldham, M., Zhang, V., Kammoona, A. et al. LegUp: An open-source high-level synthesis tool for FPGA-based processor/accelerator systems.

ACM Transactions on Embedded Computing Systems (TECS). ACM New York, NY, USA. 2013, vol. 13, no. 2, p. 1–27.

[11] Carvalho, T. R. de et al. The fault avoidance and the fault tolerance approaches for increasing the reliability of aerospace and automotive systems. In: *SAE Brasil 2005 Congress and Exhibit*. 2005, 2005-01-4157.

[12] Cavin, R., Lugli, P. and Zhirnov, V. Science and Engineering Beyond Moore's Law. *Proceedings of The IEEE - PIEEE*. january 2012, vol. 100, p. 1720–1749.

[13] Chern, M.-S. On the computational complexity of reliability redundancy allocation in a series system. *Operations Research Letters*. 1992, vol. 11, no. 5, p. 309–315. DOI: https://doi.org/10.1016/0167-6377(92)90008-Q. ISSN 0167-6377. Available at: https://www.sciencedirect.com/science/article/pii/016763779290008Q.

[14] Cofer, R. and Harding, B. F. *Rapid System Prototyping with FPGAs: Accelerating the Design Process*. Elsevier, 2006. ISBN 978-0-7506-7866-7.

[15] Corno, F., Reorda, M. and Squillero, G. RT-level ITC'99 benchmarks and first ATPG results. *Design and Test of Computers, IEEE*. Jul 2000, vol. 17, no. 3, p. 44–53. DOI: 10.1109/54.867894. ISSN 0740-7475.

[16] Coussy, P., Gajski, D. D., Meredith, M. and Takach, A. An introduction to high-level synthesis. *IEEE Design & Test of Computers*. IEEE. 2009, vol. 26, no. 4, p. 8–17.

[17] Croswell, K. Voyager still breaking barriers decades after launch. *Proceedings of the National Academy of Sciences*. National Acad Sciences. 2021, vol. 118, no. 17, p. e2106371118.

[18] Dalal, S. and Chhillar, R. S. Case studies of most common and severe types of software system failure. *International Journal of Advanced Research in Computer Science and Software Engineering*. Citeseer. 2012, vol. 2, no. 8.

[19] Fallahlalehzari, F. *How does the Mars Perseverance rover benefit from FPGAs as the main processing units?* [online]. Aldec, 2021 [cit. 2022-09-13]. Available at: https://www.aldec.com/en/company/blog/188--how-does-the-mars-perseverance-rover-benefit-from-fpgas-as-the-main-processing-units.

[20] Fingeroff, M. *High-level Synthesis: Blue Book*. Xlibris Corporation, 2010.

[21] Foster, H. *Part 6: The 2020 Wilson Research Group Functional Verification Study* [online]. Siemens Digital Industries Software, december 2020 [cit. 2022-07-06]. Available at: https://blogs.sw.siemens.com/verificationhorizons/2020/12/16/part-6-the-2020-wilson-research-group-functional-verification-study/.

[22] FPGAwars. *Digital Design for OpenSource FPGAs made easy: icestudio.io* [online]. icestudio.io, august 2022 [cit. 2022-09-08]. Available at: https://github.com/FPGAwars/icestudio.

[23] Grierson, D. E. Pareto multi-criteria decision making. *Advanced Engineering Informatics*. 2008, vol. 22, no. 3, p. 371–384. DOI: https://doi.org/10.1016/j.aei.2008.03.001. ISSN 1474-0346. Collaborative Design and

Manufacturing. Available at:
https://www.sciencedirect.com/science/article/pii/S1474034608000281.

[24] Hauck, S. and DeHon, A. *Reconfigurable computing: the theory and practice of FPGA-based computation*. Elsevier, 2010.

[25] Hegde, V. Reliability in the medical device industry. *Handbook of Performability Engineering*. Springer. 2008, p. 997–1009.

[26] Heiner, J., Sellers, B., Wirthlin, M. and Kalb, J. FPGA partial reconfiguration via configuration scrubbing. In: IEEE. *2009 International Conference on Field Programmable Logic and Applications*. 2009, p. 99–104.

[27] Hiemstra, D. M., Battiston, G. and Gill, P. Single Event Upset Characterization of the Virtex-5 Field Programmable Gate Array Using Proton Irradiation. In: *2010 IEEE Radiation Effects Data Workshop*. 2010, p. 4–4. DOI: 10.1109/REDW.2010.5619490.

[28] Hlavička, J., Racek, S., Golan, P. and Blažek, T. *Číslicové systémy odolné proti poruchám. 1st edition, Prague, Published by: ČVUT, 330 s.* 1992.

[29] Inc., X. *Xilinx XPower Analyzer* [online]. Xilinx Inc., 2013 [cit. 2021-12-07]. Available at: https://www.xilinx.com/html_docs/xilinx14_5/isehelp_start.htm#xpa_c_overview.htm.

[30] International Electrotechnical Commission. *IEC 60050 - International Electrotechnical Vocabulary, Area 192: Dependability* [online]. International Electrotechnical Commission, 2022 [cit. 2022-07-08]. Available at: https://www.electropedia.org/iev/iev.nsf/index?openform&part=192.

[31] Kanagaraj, G., Ponnambalam, S. and Jawahar, N. A Hybrid Cuckoo Search and Genetic Algorithm for Reliability–Redundancy Allocation Problems. *Computers & Industrial Engineering*. Elsevier. 2013, vol. 66, no. 4, p. 1115–1124.

[32] Kastensmidt, F. and Rech, P. FPGAs and parallel architectures for aerospace applications. In: *Soft Errors and Fault-Tolerant Design*. Springer, 2016.

[33] Kellerer, H., Pferschy, U. and Pisinger, D. The Multiple-choice Knapsack Problem. In: *Knapsack Problems*. Springer, 2004, p. 317–347.

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

[35] Kim, H. and Kim, P. Reliability–redundancy allocation problem considering optimal redundancy strategy using parallel genetic algorithm. *Reliability Engineering & System Safety*. Elsevier. 2017, vol. 159, p. 153–160.

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

[37] Kuon, I., Tessier, R., Rose, J. et al. FPGA architecture: Survey and challenges. *Foundations and Trends® in Electronic Design Automation.* Now Publishers, Inc. 2008, vol. 2, no. 2, p. 135–253.

[38] Kuuhn, J. M., Schweizer, T., Peterson, D., Kuhn, T. and Rosenstiel, W. Testing Reliability Techniques for SoCs with Fault Tolerant CGRA by Using Live FPGA Fault Injection. In: IEEE. *Field-Programmable Technology (FPT), 2013 International Conference on.* 2013, p. 462–465. ISBN 978-1-4799-2198-0.

[39] Lavin, C., Padilla, M., Lundrigan, P., Nelson, B. and Hutchings, B. Rapid Prototyping Tools for FPGA Designs: RapidSmith. In: *Field-Programmable Technology (FPT), 2010 International Conference on.* Dec 2010, p. 353–356. DOI: 10.1109/FPT.2010.5681429.

[40] Lee, D. S. *Commercial Field-Programmable Gate Arrays for Space Processing Applications.* Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2017.

[41] Lee, G., Agiakatsikas, D., Wu, T., Cetin, E. and Diessel, O. TLegUp: A TMR code generation tool for SRAM-based FPGA applications using HLS. In: IEEE. *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM).* 2017, p. 129–132. ISBN 978-1-5386-4037-1.

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

[43] Liu, M., Zeng, Z., Su, F. and Cai, J. Research on Fault Injection Technology for Embedded Software based on JTAG Interface. In: IEEE. *Reliability, Maintainability and Safety (ICRMS), 2016 11th International Conference on.* 2016, p. 1–6. ISBN 978-1-5090-2714-9.

[44] Lojda, J., Pánek, R., Podivínský, J., Čekan, O., Krčma, M. et al. Testing Embedded Software Through Fault Injection: Case Study on Smart Lock. In: *2021 IEEE 22nd Latin American Test Symposium (LATS).* 2021, p. 1–6. DOI: 10.1109/LATS53581.2021.9651770.

[45] Lojda, J., Podivínský, J. and Kotásek, Z. Reliability Indicators for Automatic Design and Analysis of Fault-Tolerant FPGA Systems. In: *2019 IEEE Latin American Test Symposium (LATS).* 2019, p. 1–4. DOI: 10.1109/LATW.2019.8704593. ISBN 978-1-7281-1756-0.

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

[47] Lyons, R. E. and Vanderkulk, W. The use of triple-modular redundancy to improve computer reliability. *IBM journal of research and development.* IBM. 1962, vol. 6, no. 2, p. 200–209.

[48] Malaya Kumar Biswal M. and Ramesh Naidu Annavarapu. A Study on Mars Probe Failures. In: *AIAA Scitech 2021 Forum*. 2021. DOI: 10.2514/6.2021-1158. Available at: https://arc.aiaa.org/doi/abs/10.2514/6.2021-1158.

[49] Martello, S. and Toth, P. *Knapsack Problems: Algorithms and Computer Implementations*. J. Wiley & Sons, 1990. Wiley-Interscience series in discrete mathematics and optimization. ISBN 9780471924203.

[50] Mehra, D. R. High Speed CT Image Reconstruction using FPGA. *International Journal of Computer Applications*. may 2011, vol. 22, p. 7–10. DOI: 10.5120/2574-3550.

[51] Mentor Graphics. *Catapult HLS* [online]. Mentor Graphics, july 2022 [cit. 2022-07-08]. Available at: https://www.mentor.com/hls-lp/catapult-high-level-synthesis/.

[52] Nane, R., Sima, V.-M., Pilato, C., Choi, J., Fort, B. et al. A survey and evaluation of FPGA high-level synthesis tools. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*. IEEE. 2015, vol. 35, no. 10, p. 1591–1604.

[53] Neuberger, G., Kastensmidt, F. and Reis, R. Improving Fault Tolerance to Radiation Effects in Integrated Systems. may 2014, p. 1–12.

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

[55] O'Bryan, M. and Campola, M. *Single Event Effects* [online]. NASA Goddard Space Flight Center, october 2019 [cit. 2022-07-12]. Available at: https://radhome.gsfc.nasa.gov/radhome/see.htm.

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

[57] Padovani, R. *Reconfigurable FPGAs for Space – Present and Future, Presentation on the MAPLD conference, Washington, DC*. 2005.

[58] Pánek, R., Lojda, J., Podivínský, J. and Kotásek, Z. Reliability Analysis of Reconfiguration Controller for FPGA-Based Fault Tolerant Systems: Case Study. In: *2020 International Symposium on VLSI Design, Automation, and Test (VLSI-DAT) : proceedings of technical papers*. IEEE Computer Society, 2020, p. 121–124. DOI: 10.1109/VLSI-DAT49148.2020.9196269. ISBN 978-1-7281-6083-2. Available at: https://www.fit.vut.cz/research/publication/12101.

[59] Pánek, R., Lojda, J., Podivínský, J. and Kotásek, Z. Reliability Analysis of the FPGA Control System with Reconfiguration Hardening. In: *Proceedings - 2021 24th Euromicro Conference on Digital System Design, DSD 2021*. Institute of Electrical

and Electronics Engineers, 2021, p. 553–556. DOI: 10.1109/DSD53832.2021.00089. ISBN 978-1-6654-2703-6. Available at: https://www.fit.vut.cz/research/publication/12489.

[60] Petersen, E. *Single Event Effects in Aerospace*. John Wiley & Sons, Ltd, 2011. 1-12 p. ISBN 9781118084328. Available at: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118084328.ch1.

[61] Podivínský, J., Lojda, J., Čekan, O. and Kotásek, Z. Evaluation Platform for Testing Fault Tolerance Properties: Soft-core Processor-based Experimental Robot Controller. In: IEEE. *2018 21st Euromicro Conference on Digital System Design (DSD)*. 2018, p. 229–236. DOI: 10.1109/DSD.2018.00051. ISBN 978-1-5386-7377-5.

[62] Podivínský, J., Čekan, O., Lojda, J., Zachariášová, M., Krčma, M. et al. Functional Verification Based Platform for Evaluating Fault Tolerance Properties. *Microprocessors and Microsystems*. 2017, vol. 52, no. 5, p. 145–159. DOI: 10.1016/j.micpro.2017.06.004. ISSN 0141-9331. Available at: https://www.fit.vut.cz/research/publication/11318.

[63] Ratter, D. FPGAs on Mars. *Xcell J.* 1st ed. 2004, vol. 50, no. 8, p. 11.

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

[65] Schweizer, T., Peterson, D., Kühn, J. M., Kuhn, T. and Rosenstiel, W. A Fast and Accurate FPGA-based Fault Injection System. In: IEEE. *Field-Programmable Custom Computing Machines (FCCM), 2013 IEEE 21st Annual International Symposium on*. 2013, p. 236–236. ISBN 978-0-7695-4969-9.

[66] Siemens EDA. *Algorithmic C (AC) Datatypes Reference Manual* [online]. Siemens, august 2022 [cit. 2022-09-12]. Available at: https://github.com/hlslibs/ac_types/blob/master/pdfdocs/ac_datatypes_ref.pdf.

[67] Somani, A. K. and Vaidya, N. H. Understanding fault tolerance and reliability. *Computer*. 1st ed. IEEE Computer Society. 1997, vol. 30, no. 04, p. 45–50.

[68] SOOS, C. *SEU effects in FPGA: How to deal with them?, Presentation on the 1st Combined R2E Workshop & School-Days, European Organization for Nuclear Research (CERN)*. 2009.

[69] Straka, M., Kaštil, J. and Kotásek, Z. Generic Partial Dynamic Reconfiguration Controller for Fault Tolerant Designs Based on FPGA. In: *NORCHIP 2010*. IEEE Computer Society, Nov 2010, p. 1–4. DOI: 10.1109/NORCHIP.2010.5669477. ISBN 978-1-4244-8971-8.

[70] Straka, M., Kaštil, J. and Kotásek, Z. SEU Simulation Framework for Xilinx FPGA: First Step Towards Testing Fault Tolerant Systems. In: *14th EUROMICRO Conference on Digital System Design*. IEEE Computer Society, 2011, p. 223–230. ISBN 978-0-7695-4494-6.

[71] Synario Design Automation. *VHDL Reference Manual*. Synario Design Automation, 1997.

[72] Texas Instruments Incorporated. *Reliability Terminology* [online]. Texas Instruments Incorporated, 2022 [cit. 2022-07-27]. Available at: https://www.ti.com/support-quality/reliability/reliability-terminology.html.

[73] The Planetary Society. *Cost of Perseverance* [online]. The Planetary Society, 2022 [cit. 2022-07-08]. Available at: https://www.planetary.org/space-policy/cost-of-perseverance.

[74] Velazco, R., McMorrow, D. and Estela, J. *Radiation Effects on Integrated Circuits and Systems for Space Applications*. 1st ed. January 2019. ISBN 978-3-030-04659-0.

[75] Wang, Z., Chen, T., Tang, K. and Yao, X. A Multi-objective Approach to Redundancy Allocation Problem in Parallel-series Systems. In: IEEE. *2009 IEEE Congress on Evolutionary Computation*. 2009, p. 582–589. ISBN 978-1-4244-2959-2.

[76] Xilinx Inc. *TMRTool: The Induay's First Development Tool to Automatically Generate Triple Module Redundancy (TMR) for Space-grade Re-programmable FPGAs* [online]. Xilinx Inc. [cit. 2021-04-13]. Available at: https://www.xilinx.com/products/design-tools/tmrtool.html.

[77] Xilinx Inc. *ML505/ML506/ML507 Evaluation Platform User Guide* [online]. Xilinx Inc., may 2011 [cit. 2022-07-27]. Available at: https://www.xilinx.com/support/documentation/user_guides/ug347.pdf.

[78] Xilinx Inc. *Virtex-5 FPGA Configuration User Guide* [online]. Xilinx Inc., november 2011 [cit. 2022-09-17]. Available at: https://www.xilinx.com/support/documentation/user_guides/ug191.pdf.

[79] Xilinx Inc. *Virtex-5 FPGA User Guide* [online]. Xilinx Inc., march 2012 [cit. 2019-03-26]. Available at: https://www.xilinx.com/support/documentation/user_guides/ug190.pdf.

[80] Xilinx Inc. *ISE Design Suite* [online]. Xilinx Inc., july 2022 [cit. 2022-07-08]. Available at: https://www.xilinx.com/products/design-tools/ise-design-suite.html.

[81] Xilinx Inc. *Vivado ML Overview* [online]. Xilinx Inc., september 2022 [cit. 2022-09-12]. Available at: https://www.xilinx.com/products/design-tools/vivado.html.

[82] Xilinx Inc. *Xilinx Medical Applications* [online]. Xilinx Inc., july 2022 [cit. 2022-07-12]. Available at: https://www.xilinx.com/applications/medical.html.

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

Selected Papers

Paper A

Data Types and Operations Modifications: a Practical Approach to Fault Tolerance in HLS

LOJDA Jakub, PODIVÍNSKÝ Jakub, KOTÁSEK Zdeněk, KRČMA Martin

In: *Proceedings of IEEE East-West Design & Test Symposium*. Novi Sad: IEEE Computer Society, 2017, pp. 273-278. ISBN 978-1-5386-3299-4.

Available at: https://ieeexplore.ieee.org/document/8110113

Data Types and Operations Modifications: a Practical Approach to Fault Tolerance in HLS

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

Abstract

Some types of electronic systems are working in the environment with an increased occurrence of faults such as space, aerospace or medical systems. Faults in these systems can lead to the failure of the whole system and can cause high economical losses or endanger human health. Fault tolerance is one of the techniques, the goal of which is to avoid such situations. This paper presents an approach to fault-tolerant data-paths *design that is based on the modification of High-level Synthesis (HLS) input specification. The description and evaluation of the impacts of some HLS optimization methods are demonstrated in the paper as well. Higher reliability is achieved through the modification of input description in the C++ programming language which the HLS synthesis tools are based on. Our work targets SRAM-based FPGAs that are prone to Single Event Upsets (SEUs). For the evaluation of the proposed method we use our evaluation platform, which allows us to analyze fault tolerance properties of the Design Under Test (DUT). The evaluation platform is based on functional verification in combination with fault injection.*

1. Introduction

Nowadays, electronic systems are used in various devices which play an important role in our everyday lives. The increase of chip-level integration results in a higher susceptibility to faults. The number of digital systems with a high demand on reliability, such as medicine, space, industry, is growing as well. In these cases especially, reliability is very important because the consequences of failure can result in injury or heavy financial losses or can endanger human health. One of the main approaches to increase reliability is the so-called *fault tolerance* [10]. Fault tolerance accepts the fact a fault can appear, but the goal of this approach is to keep the system functional even in the presence of faults. Techniques based on various types of redundancy are used for this purpose. Many fault tolerance methodologies exist, which combine and improve these basic methods (e.g. the HW and *temporal* redundancy is combined in approach shown in [9]).

The HLS is a set of methods transforming a digital circuit description into its RTL representation. The architecture of a typical HLS-generated circuit is composed of the so-called *data-path* and *control-path* (the controller) [4]. The HLS tools usually allow the designer to explore the state space of various RTL realizations very effectively and easily. The main decisions, such as setting a degree of parallel computation of a programming loop (*unrolling*) or *pipelining* a programming loop, are still the designer's responsibility. These two optimization techniques are not nearly all used in the process of accelerating the resulting system, but we have chosen these as we believe these are the most important ones. The *unrolling* basically allows parallel execution of individual loop iterations. The parameter of *degree of parallel execution* expresses the number of iterations executed in parallel. The *pipelining* allows the designer to create a pipelined version of the loop. When the *pipelining* is applied, each *Initiation Interval* (II) the execution of one iteration is started.

These days a lot of effort in the research of *fault tolerance* in HLS is dedicated to *data-path* hardening. Specifically modified versions of HLS methodologies are usually used for this purpose. The method described in [7] is directed against transient faults that last for several clock cycles. Another heuristic algorithm based on two-phase *resource binding* was published in [8]. A heuristic-based method that was published in [14] en-

978-1-5386-3299-4/17/$31.00

ables designers to choose a trade-off between resources consumed, resulting system latency and redundancy. The authors of [2] present an approach to error detection of arithmetic oriented *data-paths*.

This paper is organized as follows. Our fault tolerance method based on data types modifications is proposed in Section 2. The proposed method is demonstrated on the robot controller described in Section 3. The results of our experiments are summarized in Section 4. Section 5 concludes the paper and presents future plans.

2. HLS-based Fault Tolerance Methodology

Our approach is to apply modifications to the specification as the input of HLS. The modifications should produce the resulting RTL description fault-tolerant. Our method is based on the modification of the *C++* language. There are three types of locations the modification can be done at the level of *C++* language: 1) data types (*storage elements*); 2) arithmetic and logic operations and 3) flow control statements.

In this research we focus on the modification of *storage elements* and *operations* of the input description. We developed a new class of *Data Types* (DTs), which we call *Redundant Data Types* (RDTs), that are able to incorporate redundancy for all the operations and storages associated with the corresponding variables. The concept of RDTs is very similar to that of the Algorithmic C Datatypes [11], which are widely used in HLS as a principle to specify a bit width of a particular data type. In this case, this concept is used to specify a redundancy mechanism. This way we were able to achieve redundancy on certain parts of the input digital system description only. An RDT is associated with the previously used DT which we call the *original* DT in relation to the particular RDT. We show this concept on the *Triple Modular Redundancy* (TMR) principle, although it is not limited to TMR only. If a user intends to add a redundancy to a particular part of the system, simply replacing the previously used DT by the RDT of a desired redundancy in this part of the system is enough. An example of the usage of RDTs is shown in Figure 1.

In the following text, TMR is used as an example of the construction principles of the RDTs. Each instance of the TMR RDT contains three nested instances of the *original* DT. If necessary, support methods to extend the behavior of the operators can be added. In the case of TMR, one method implementing the voter is included. In the *C++* language, 1) unary, 2) binary and 3) ternary operators can be distinguished from the

	Original code	Modified code	Preprocessed result (semantically)
1	`int a;`	`triple<int> a;`	`int a_x, a_y, a_z;`
2	`int b;`	`triple<int> b;`	`int b_x, b_y, b_z;`
3	`int c;`	`triple<int> c;`	`int c_x, c_y, c_z;`
4	`b = 7;`	`b = 7;`	`b_x = 7; b_y = 7; b_z = 7;`
5	`c = 8;`	`c = 8;`	`c_x = 8; c_y = 8; c_z = 8;`

6	`a = b + c;`	`a = b + c;`	`a_x = b_x + c_x;`
			`a_y = b_y + c_y;`
			`a_z = b_z + c_z;`
			`vote(&a_x, &a_y, &a_z);`

7	`/* a = 15 */`	`/* a = 15 */`	`/* a_x, a_y and a_z = 15 */`

Figure 1. An example of a C++ program code before/after its modification and after the code is preprocessed by a C++ preprocessor.

arity point of view. For each unary operator, a new operator is constructed that is composed of three operations, each for one of the nested instances. After that, the voter method is called to choose the majority result, which is then written back to each nested instance of the original DT. For binary operators there are three cases to be distinguished when considering operations with RDTs. These include a) *intra-data type* operations – RDT vs. RDT of equivalent redundancy types (e.g. TMR vs. TMR subsystem); b) *inter-data type* operations – RDT vs. RDT of different redundancy types – (e.g. TMR vs. *duplex* subsystem); and c) *original-data type* operations – RDT vs. it's *original* (*unhardened*) DT (e.g. TMR vs. *unhardened* subsystem). These three cases are schematically illustrated in Figure 2. For the ternary (*conditional*) operator, it is enough to provide a way to decide which value should be considered in place of the (*conditional*) operator. Therefore, the solution is to add an ability to cast the RDT to the Boolean DT.

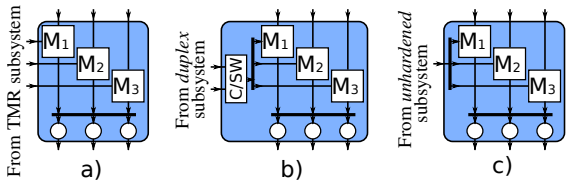

Figure 2. Three types of cases that can be distinguished when considering binary operations, intra-DT operation between two TMR subsystems (a), inter-DT operation between system with TMR and duplex hardening (b) and original-DT operation between TMR and unhardened subsystems (c).

The advantage of our approach includes the ease of its use with any HLS tool and its ability to ensure

fault tolerance for a specific part of the system that corresponds to a particular variable and its associated operations on the description level. Another benefit includes an automatic ability to interface fault-tolerant parts with the unmodified remainder of the system.

The method is intended to be a part of a larger system that would make the modifications automatically with the impacts of these changes in mind. The methodology will have to be aware of the importance of each component to assign the proper fault tolerance methodology.As the input description is in the form of an executable code, a possible option could be to involve a profiler tool, which can be used to determine the frequency of function calls. This might be a good guide to find out the functions with variables in order to apply fault tolerance to. Figure 3 shows the proposed flow.

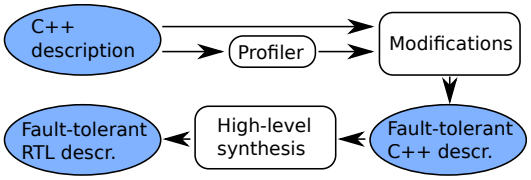

Figure 3. The new approach to FT design.

3. Case Study: Robot Controller

To demonstrate and evaluate our approach, an experimental electro-mechanical system has been developed which is composed of the robot which searches a path through a maze and its electronic controller. The robot controller unit was developed according to the HLS methodology flow using the *C++* language. The HLS tool we used in our experiments is the *Catapult C University Version* [6]. The unit is based on the left hand algorithm, that is, in case of a crossroad the robot always follows the wall of the maze on its left side.

In our previous papers (e.g. [13]) the evaluation platform for checking the impact of faults was presented. Our evaluation platform uses *functional verification* [12] as a tool for monitoring the impacts of faults injected into an electronic controller implemented into FPGA. In case of the fault injection, the verified circuit must operate in FPGA, so we do not use classical simulation-based functional verification, but modified FPGA-based functional verification.

Nowadays, many electronic systems are a part of electro-mechanical systems, where a mechanical part is controlled by its electronic controller. The trend is to move more functionality to electronic controllers because it results in lower costs on device operation. As an example, the results published in [3] and [1] can serve, where moving more functionality to electronic controllers results in a lower weight of the aircraft saving costs on aircraft operation. Based on these facts, our evaluation platform is able to use an electro-mechanical application as an experimental system, which allows us to monitor the impact of faults not only on electronic controller, but also on the controlled mechanical parts.

The main components which our evaluation platform is composed of, are shown in Figure 4. The two main parts are a computer and an FPGA development board. We use the ML506 board with the *Virtex 5* FPGA, which allows us to implement a verified electronic controller in FPGA and inject faults directly into the FPGA. The fault injector is one of the components which is running on the computer. Our fault injector [15] is based on the partial reconfiguration and uses JTAG interface for communication with the configuration memory. The platform is designed to evaluate the impact of faults on the electro-mechanical application, so the simulation of the mechanical part is important and also runs on the computer. We use the *Player/Stage* [5] simulation environment to simulate the robot and its environment. The electronic controller implemented into FPGA is connected with the simulation of mechanical part through Ethernet interface. The software part of the verification environment also runs on the computer and performs the evaluation of impacts of injected faults both on the electronic and mechanical parts.

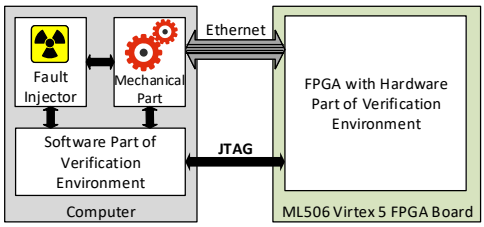

Figure 4. The evaluation platform architecture.

4. Experiments and Results

The previously mentioned robot controller implementation was used as an experimental electronic system in our experiments. In the first stage of the experiments, we evaluated the impact of some of the main optimization and acceleration techniques of the HLS methods to the resulting system's susceptibility

to SEUs. The other parameters monitored were the number of slices, slice registers and slice *Lookup tables* (LUTs) occupied.

Parts of Table 1 labeled as *noft* summarize the resource requirements for each of the four robot controller units synthesized with a different parameters set. The first and the second set of parameters, denoted as *noopt-area* and *noopt-latency*, include area and latency optimizations with no additional requirements added. As can be seen in Figure 5, the results are almost equal, which may be caused by a relatively small design size. The third one, *pipeline1-area*, includes the main loop pipelined with II set to 1 and the overall goal set to the area. The fourth one, *unroll2-area*, contains the main loop partially unrolled with the level of parallel computation set to 2. As can be seen, the unrolled loop requires more resources as the pipelined one, but it is slightly faster.

Table 1. Resources consumed for each version of the HLS synthesized robot controllers.

Version		Occupied slices [–]	Slice reg. [–]	Slice LUTs [–]	Max. frequency [MHz]	LUT bits [–]
noopt-latency	*noft*	170	346	381	74.85	19392
noopt-area	*noft*	170	346	381	74.85	19392
	triple	378	638	851	82.01	48704
	TMR	546	1038	1143	74.85	58176
pipeline1-area	*noft*	196	152	405	58.82	21952
	triple	411	512	1101	65.81	67264
	TMR	540	456	1215	58.82	65856
unroll2-area	*noft*	399	656	854	59.70	48704
	triple	1341	1791	3738	50.48	224256
	TMR	1224	1968	2562	59.70	146112

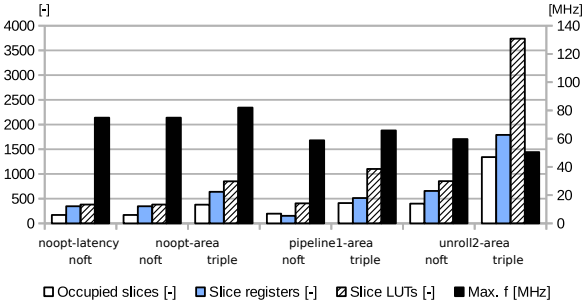

Figure 5. Comparison of resources consumed for each version of the HLS synthesized robot controllers.

The first experiments with fault injection were targeted to evaluate the resilience against faults of proposed versions of the robot controller. In these experiments only three versions (*noopt-area*, *pipeline1-area* and *unroll2-area*) were taken into account, because the detailed analysis showed that *noopt-area* and *noopt-latency* led to a very similar VHDL description.

The three resulting robot controller units were examined for their susceptibility to SEUs. Exactly 1000 verification runs were performed with each version of the three robot controller units. Before the experiments started, a list of configuration bits that were used as a content of LUTs was generated for each version of the robot controller bitstream. These bits were then used in the processes of SEU injections. The scenario of each verification run was as follows:

1. the robot controller unit was reinstated into its initial state, the maze map as well as its starting and target positions were the same for all the verification runs,
2. the *Player/Stage* simulation environment was started, the robot was placed on the starting position,
3. one SEU was injected into a bit that was utilized as an LUT content bit, the bit was selected *uniformly at random* from all bits utilized as contents of LUTs, the bit remained in a faulty state during the whole verification run,
4. the robot was started and the ability of the robot to reach the target position was monitored and eventually the reason of its failure was observed.

Results of these experiments are summarized in Table 2. The first row shows the number of verification runs without any impact on the electronic controller, while the second row indicates the number of faults that cause discrepancy on the output of the electronic controller. The third row enumerates the reliability improvement of the units with triplication applied. The reliability improvement was calculated using Equation 1 for each pair of units with corresponding HLS settings set s.

$$reliab\_improv_s = \frac{fails\_noft_s - fails\_triple_s}{fails\_noft_s} * 100 \quad (1)$$

Table 2 also shows that electronic failure sometimes led to “Goal not reached” or “Goal reached”. It should be noted that sometimes the robot reached the goal position although its electronic system failed. The table shows that the *noopt-area* version of robot controller is

the most prone to injected faults. The number of faults which led to electronic failure is 51 which is 5.1% of injected faults.

Table 2. Impact of faults on various versions of robot controller without and with fault tolerance method applied.

Monitored impact	*noopt-area*		*pipeline1-area*		*unroll2-area*	
	noft	*triple*	*noft*	*triple*	*noft*	*triple*
Electronic OK [–]	949	982	967	996	979	995
Electronic failed [–]	51	18	33	4	21	5
Reliability improvement [%]	–	64.7%	–	87.9%	–	76.2%
Goal not reached [–]	50	16	32	4	19	4
Collision with wall [–]	4	2	5	1	4	1
Goal reach. alth. el. fail. [–]	1	2	1	0	2	1

The next stage of our experiments was targeted to applying the proposed methodology to each variable of the robot controller algorithm specification (in tables and charts labeled as *triple* version). We evaluated 1) a resource consumption, 2) a susceptibility of modified robot controllers to the faults and the comparison of the results with the versions without any fault tolerance modifications.

The comparison of the resource consumption of the *noft* and *triple* versions (with the proposed method applied) is available in Table 1. The synthesis tool used was the *Xilinx Integrated Synthesis Environment* (ISE) [16]. It is evident that the robot controller hardened by the proposed methodology consumes more resources. In comparison with the complete triplication of the robot controller (rows of Table 1 labeled as *TMR*) that were synthesized using three copies of the corresponding *noft* version, less resources are consumed, although in some cases such as in the case of the *unroll2-area* the resource utilization increased. However, it is important to keep in mind the complete triplication comprises three copies of the *control-path*, while the proposed method is *data-path* only.

From the reliability point of view the same experiments were prepared. In this stage, we also injected one single fault during one verification run. The scenario of each verification run, including the maze map, was identical to that of the experiments mentioned in the first phase. Based on 1000 verification runs we can say that the proposed methodology leads to a lower sensitivity on injected faults as is shown in Figure 6. The most significant contribution can be seen in the case of the *pipeline1-area* where our methodology led to an improvement of 87.9%. In case of *unroll2-area* the improvement is 76.2%. The smallest improvement of 64.7% was achieved in case of *noopt-area*. Based on these experiments we can conclude that the proposed methodology leads to improvement of the fault tolerance against SEUs with the best efficiency for pipelined designs. However, this method needs to be combined with another approach considering the *control-path*, which would provide even better resilience to faults.

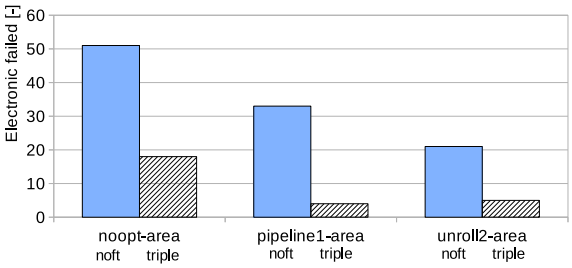

Figure 6. Number of faults that led to failure of the electro-mechanical system for each of the versions.

5. Conclusions and Future Research

In this paper we introduced a newly emerging approach to easily achieve a certain level of fault tolerance with the usage of HLS. In our experiments, robot controller system is modeled in the C++ language. After that, the model is modified using our approach and synthesized using HLS with selected settings sets. Resulting systems, which have all data-path components and data elements triplicated, are evaluated using single fault injections. The experiments monitor the impacts of injected faults, both on robot controllers without proposed fault tolerance methodology applied, and also on robot controllers hardened against faults. The experiments show that single faults injected into utilized LUTs of the hardened robot controller had smaller impact on the robot controller and its behavior. In our experiments, resource consumption was also analyzed. The proposed methodology leads to a greater consumption of resources, the comparison of each of the versions with the TMR triplication the corresponding robot controller was provided as well. The objective of our research is to improve this principle to make it generally usable and show its usability on other applications or benchmarks.

The next step in our research would be to involve the evaluation of the impact of these modifications on different parts of the system. The main idea is that each part of the system deserves a different level of reliability, based on its function and thus a different level of fault tolerance. Therefore, it is not necessary to apply the same level of fault tolerance to every part of the

system and it is very likely that another configuration of fault tolerance that has at least the same level of dependability with less resources consumed exists.

Acknowledgements

This work was supported by The Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPU II); project IT4Innovations excellence in science - LQ1602 and BUT project FIT-S-14-2297.

References

[1] J. Bennett, A. Jack, B. Mecrow, D. Atkinson, C. Sewell, and G. Mason. Fault-tolerant Control Architecture for an Electrical Actuator. In *Power Electronics Specialists Conference, 2004. PESC 04. 2004 IEEE 35th Annual*, volume 6, pages 4371–4377, June 2004.

[2] K. A. Campbell, P. Vissa, D. Z. Pan, and D. Chen. High-Level Synthesis of Error Detecting Cores Through Low-Cost Modulo-3 Shadow Datapaths. In *Proceedings of the 52nd Annual Design Automation Conference*, DAC '15, pages 161:1–161:6, New York, NY, USA, 2015. ACM.

[3] S. Cutts. A Collaborative Approach to the More Electric Aircraft. In *Power Electronics, Machines and Drives, 2002. International Conference on (Conf. Publ. No. 487)*, pages 223–228, June 2002.

[4] M. Fingeroff. *High-Level Synthesis Blue Book*. Xlibris Corporation, 2010.

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

[6] M. Graphics. Catapult HLS. `<https://www.mentor.com/hls-lp/catapult-high-level-synthesis/>`, 2017. Accessed: 2017-07-07.

[7] T. Inoue, H. Henmi, Y. Yoshikawa, and H. Ichihara. High-Level Synthesis for Multi-Cycle Transient Fault Tolerant Datapaths. In *2011 IEEE 17th International On-Line Testing Symposium*, pages 13–18, July 2011.

[8] M. Kaneko and Y. Tsuboishi. Constrained Binding and Scheduling of Triplicated Algorithm for Fault Tolerant Datapath Synthesis. In *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1448–1451, June 2014.

[9] F. L. Kastensmidt, G. Neuberger, L. Carro, and R. Reis. Designing and Testing Fault-Tolerant Techniques for SRAM-based FPGAs. In *Proceedings of the 1st conference on Computing frontiers*, pages 419–432. ACM, 2004.

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

[11] Mentor Graphics. AC Datatypes v3.7. `<https://www.mentor.com/hls-lp/downloads/ac-datatypes>`, June 2016. Accessed: 2016-12-14.

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

[13] J. Podivinsky, O. Cekan, J. Lojda, and Z. Kotasek. Verification of Robot Controller for Evaluating Impacts of Faults in Electro-mechanical Systems. In *Digital System Design (DSD), 2016 19th Euromicro Conference on*, pages 487–494. IEEE, 2016.

[14] A. Shastri, G. Stitt, and E. Riccio. A Scheduling and Binding Heuristic for High-Level Synthesis of Fault-Tolerant FPGA Applications. In *2015 IEEE 26th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 202–209, July 2015.

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

[16] Xilinx. ISE Design Suite. `<https://www.xilinx.com/products/design-tools/ise-design-suite.html>`, 2017. Accessed: 2017-07-07.

Paper B

Redundant Data Types and Operations in HLS and their Use for a Robot Controller Unit Fault Tolerance Evaluation

LOJDA Jakub, PODIVÍNSKÝ Jakub, KOTÁSEK Zdeněk

In: *Proceedings of IEEE East-West Design & Test Symposium*. Novi Sad: IEEE Computer Society, 2017, pp. 359-364. ISBN 978-1-5386-3299-4.

Available at: https://ieeexplore.ieee.org/document/8110127

Redundant Data Types and Operations in HLS and their Use for a Robot Controller Unit Fault Tolerance Evaluation

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

Abstract

Some environments (e.g. space, aerospace or medical systems) require electronic systems to withstand an increased occurrence of faults. Moreover, the failure of these electronic systems can cause high economical losses or endanger human health. Fault tolerance is one of the techniques, the goal of which is to avoid such situations. This paper presents an approach to evaluate the degree of importance of individual system partitions when High-Level Synthesis (HLS) methodology is used. The importance of individual partitions was evaluated by the usage of our approach to fault-tolerant datapaths *design which is based on the HLS input specification modification. The partitions are formed by sets of variables and operations. A brief description of the approach to fault tolerance in HLS is shown in the paper as well. Our experiments are evaluated using an SRAM-based FPGA evaluation platform which allows us to analyze fault tolerance properties of the Design Under Test (DUT). In the evaluation platform, functional verification in combination with fault injection is utilized.*

1. Introduction

In our everyday lives we meet various types of electronic systems. Some of them are used just for entertainment purposes, others help us to make our life easier, but some of them are very important and safety critical because they control processes the failure of which can result in injury, heavy financial losses or can endanger human health. Medical equipment, space and aerospace control systems or automotive safety assistants can serve as examples of safety critical systems. It is very important to protect these systems against faults and ensure their reliable operation in every situation.

The technique called *fault tolerance* [8] is one of main approaches to increase electronic systems reliability. Fault tolerance accepts the fact a fault can appear, but the goal of this approach is to keep the system functional even with the presence of faults. Fault tolerance is a widely used technique which is usually based on various types of redundancy. The most common are *area* and *time redundancy*. *Area redundancy* usually uses n-copies of the same functional unit and a comparator to guarantee the proper function. On the other hand, *time redundancy* is based on repeated computation and the results from the independent runs are then compared.

The main subject of our research in the field of fault tolerant systems design and evaluation are SRAM-based *Field Programmable Gate Arrays* (FPGAs). FPGAs are composed of reconfigurable blocks and an interconnection network. The SRAM memory, in which the configuration bitstream is saved, is sensitive to *Single Event Upsets* (SEUs), which are caused by charged particles [16]. The goal of the research presented in this paper is to evaluate the importance of various high level *storage elements* and associated *operations* using their impact on reliability improvement. Knowing the importance of various partitions is essential in ensuring the highest level of reliability in cases where the remaining chip-area is limited. The fault tolerance method is based on the modification of an input algorithm before it is processed by HLS. For the evaluation, the approach of SEU injections in combination with our evaluation platform is used.

In this paper, the concept of HLS can be understood as a set of methods transforming a *high-level* de-

978-1-5386-3299-4/17/$31.00

scription to its implementation on the *Register Transfer Level* (RTL). The description is made on a high level of abstraction, usually in the form of an algorithm described in one of the higher-level programming languages (e.g. *C++*). The resulting RTL implementation is dependent on the configuration of the HLS. The HLS tools usually incorporate an ability to effectively explore the state space of all possible configurations. There are various parallelization techniques in HLS, although we only consider the most important the loop acceleration techniques such as *loop pipelining* and *loop unrolling*. These are usually fully exposed to the designer. The HLS resulting RTL is usually composed of the so-called *control-path*, which is usually in the form of a *Finite State Machine* (FSM), and the so-called *data-path*, which contains all the data processing hardware such as *Arithmetic Logic Units* (ALUs).

These days a lot of effort in the research of fault tolerance in HLS is dedicated to *data-path* hardening. The authors of [5] present a heuristic algorithm for searching an optimal assignment of operations to *data-paths* while considering the maximal cycle length of the transient fault. The authors of [14] show that in most cases, 100% fault coverage is not necessary. This fact allows them to implement a higher degree of freedom into the phases of *scheduling* and *binding* and save HW resources. The authors of [7] present a *two-phase* resource binding heuristic algorithm with considerable processor time and memory usage reduction in the phase of system design. An approach to error detection of arithmetic oriented *data-paths* is presented in [1]. The authors of [13] show a method of detecting *multi-cycle* transient faults while connecting the higher-level synthesis with the lower (*physical*) level and reducing the overhead. A new approach for the fault-tolerant HLS controller is shown in [6]. The authors of this method show that their approach requires less overhead resources than using the TMR. All the methods presented rely on a modification of the HLS methodology, but in our approach we are trying to strictly separate from the HLS tool itself while keeping all the benefits of the HLS.

This paper is organized as follows. An overview of our fault tolerance method based on data types modifications is proposed in Section 2. The experimental system and evaluation platform are presented in Section 3. The case study and experimental results are summarized in Section 4. Section 5 contains the conclusion of the paper and presents our plans for future research.

2. HLS-based Fault Tolerance Approach

In our approach we modify the input specification of the HLS to achieve a fault-tolerant system at the output of the HLS methodology. As this approach works at the level of abstraction of the input specification, it profits from all the benefits of the HLS. This idea was already presented in our previous work [10], although in this paper the method is used to evaluate the importance of various partitions of the DUT.

In the *C++* language code, three places to make modifications can be distinguished: 1) *data types*; 2) *arithmetic and logic operations*; and 3) *flow control statements*. This research is focused on the *data types* (DTs) and *operations* modifications. The method of creating new DTs, which we call the *Redundant Data Types* (RDTs), will be shown on the well known principle of the *Triple Modular Redundancy* (TMR). RDTs are using already existing (in the following text referenced as *original*) DTs, where each RDT expresses one method of fault tolerance (e.g. the RDT *triple* expresses TMR). This approach allowed us to modify the semantics of corresponding DT operations and to implement fault tolerance methods into these operations. As a result, all the operations whose operands include at least one RDT are modified according to the particular fault tolerance method of the RDT(s).

In the phase of a new RDT creation, three types of operations considering their *arity* must be addressed: 1) *unary*; 2) *binary*; and 3) *ternary*. While in the case of *unary* operations there is no need to consider another DTs existence, in the case of *ternary* (*conditional*) operations, however, an ability to cast the RDT variable to the *Boolean* DT must be added to provide the ternary operator with a *Boolean* value in order to evaluate the *conditional expression*. In the case of *binary* operations, operands of multiple combinations of DTs or RDTs may arise. These combinations include: a) *intra-data type* operations – RDT vs. RDT of equivalent redundancy types (e.g. TMR vs. TMR subsystem); b) *inter-data type* operations – RDT vs. RDT of different redundancy types – (e.g. TMR vs. *duplex* subsystem); and c) *original-data type* operations – RDT vs. it's *original* (*unhardened*) DT (e.g. TMR vs. *unhardened* subsystem). Examples of the combinations of RDT and DT operations are shown in Figure 1. Each time a new RDT is added, all existing RDTs must be updated to address all of the newly arisen *inter-data type* operation combinations.

With our approach it is relatively easy to incorporate fault tolerance into systems already described, although the decision on which RDT to apply to which

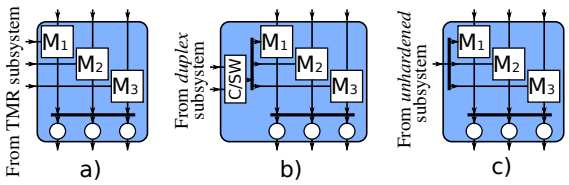

Figure 1. Three types of cases that can be distinguished when considering binary operations, intra-DT operation between two TMR subsystems (a), inter-DT operation between system with TMR and duplex hardening (b) and original-DT operation between TMR and unhardened subsystems (c).

variable might not be a trivial problem in cases where the whole system contains hundreds or even thousands of program variables. For such cases, our goal is to develop an automated methodology that would contain guides to calculate weights for algorithm operations and thus corresponding variables to estimate the impact of a particular modification of the input code. We assume that when a limited amount of HW resources is a concern, the selection of the amount of redundancy for each component of the system should be in correlation with its *importance*. To express the changes made in DTs of the input algorithm, an extension of *Activity Diagram* (AD) from *Unified Modeling Language* (UML), which is described in its original form in [2], could be used. For each action of the AD, the extension assigns a corresponding set of variable instance names utilized, their *original* DTs and eventually the RDTs replacing them. An example of the extended AD is shown in Figure 2. The example contains one action with two corresponding *original* variables and their DTs. The corresponding RDT if applied is listed for each variable on the right side of the vertical line. In this example of an AD, each variable is *hardened* with the `triple` RDT.

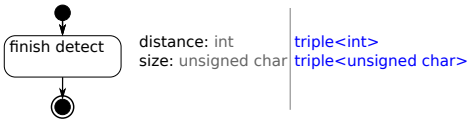

Figure 2. An example of the extended UML AD, to each action variable instances and corresponding DTs are assigned; RDTs if applied are listed on the right side of the line.

3. Experimental platform

The objective of our research comprehends not only the development and improvement of fault tolerance methodologies, but their evaluation as well. In our previous papers (e.g. [12]) the evaluation platform for checking the impact of faults was presented. Our evaluation platform uses *functional verification* [11] as a tool for monitoring the impact of faults injected into an electronic controller implemented into an FPGA. The main task of the functional verification is to check whether a verified circuit satisfies its specification. It compares outputs of a verified circuit running in the RTL simulator with a reference model. When fault injection is required, the DUT must be implemented into an FPGA. In this case the classical simulation-based functional verification is not used.

The architecture of our evaluation platform is shown in Figure 3. The two main parts are a computer and an FPGA development board with the robot controller. We use the ML506 board with the Virtex 5 FPGA, which allows us to inject faults directly into the FPGA. Another FPGA board (in the middle) serves as a bridge between the *Ethernet* interface and the *General Purpose Input Output* (GPIO) ports. The fault injector is one of the components running on the computer. Our fault injector [15] is based on the *Partial Dynamic Reconfiguration* (PDR) [17]. It reads part of the configuration bitstream from the configuration memory through the JTAG interface, then the requested number of specified bits of the bitstream is inverted and the modified bitstream is configured back to the configuration memory. The evaluation platform is able to use an electro-mechanical application as an experimental system, which allows us to monitor the impact of faults not only on the electronic controller, but also on the controlled mechanical parts. It should be noted that the simulation of the mechanical part is important and also runs on the computer. The electronic controller implemented into the FPGA is connected with the simulation of the mechanical part through the Ethernet interface. An evaluation of the impact of injected faults, both on the electronic and mechanical parts, is performed in the software of the verification environment, which runs on the computer.

We use a robot for the searching path through a maze and its electronic controller as an experimental electro-mechanical system. Our robot controller is implemented in VHDL which can be synthesized and configured in the FPGA. So, there are two possibilities on how to evaluate the fault tolerance methodologies: 1) apply a fault tolerance methodology to the implemented controller; or 2) create a new robot controller

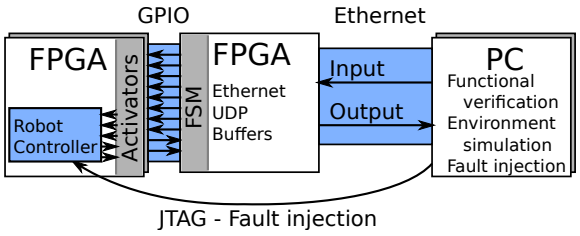

Figure 3. The architecture of our evaluation platform.

according to the evaluated fault tolerance methodology. The second approach is used in this work and we have implemented a new HLS-based robot controller using the HLS flow.

4. Case Study and Experimental Results

For the purpose of our approach evaluation, a robot controller was implemented according to HLS methodology. The input specification is written in the *C++* language and is based on the so-called *left-hand* algorithm which in the case of a crossroad in a maze always follows the *wall* on its left side. The *Player/Stage* [3] simulation environment is used to simulate the robot which has four sensors on its chassis, each facing one of the sides of the World.

In this research we are trying to evaluate the importance of each particular component of the robot controller unit and thus find out components that have the greatest potential in adding SEU resiliency if made fault tolerant. Our previous research was targeted towards an evaluation of the robot controller design fully hardened with our method when considering various HLS settings, while in this paper, the HLS settings remain constant and resilience against SEUs is always evaluated for designs with only one particular partition hardened.

An RDT named *triple*, which is based on the TMR principle, was used. For the purpose of evaluation, seven new robot controller unit versions were created using the *Catapult C University Version* tool [4] and synthesized with the *Xilinx Integrated Synthesis Environment* (ISE) [18]. Each controller of these versions has the proposed methodology applied to a different set of variables. These seven sets of variables are *mutually disjoint*. Since placing all of the seven extended ADs for each of the robot controllers to this paper would be very space-consuming, Figure 4 shows the extended AD of the robot controller unit having each variable *hardened*. The figure also highlights the seven variable sets with their proposed modifications. It is important to note that for each of the seven controller designs, only the corresponding set of variables was *hardened*. That is, for the controller unit version 1, only the set of variables marked by 1 (i.e. the `x_goal` and the `y_goal` variable) was *hardened*, while leaving the remaining variables *unhardened* for a particular design. Similarly, the other six versions were created. The HLS setup used during the synthesis is based on the *pipeline1-area* HLS settings set from our previous research which was submitted to [9]. The *pipeline1-area* settings comprehend the whole design *pipelined* with an *Initiation Interval* (II) of 1. As this HLS setup turned out to be the most sensitive to our approach (i.e. with the best reliability improvement gain when each variable used the *triple* RDT from all of the setups tested), we decided to use it in this research in order to achieve the best possible resolution when distinguishing among reliability improvements of various versions.

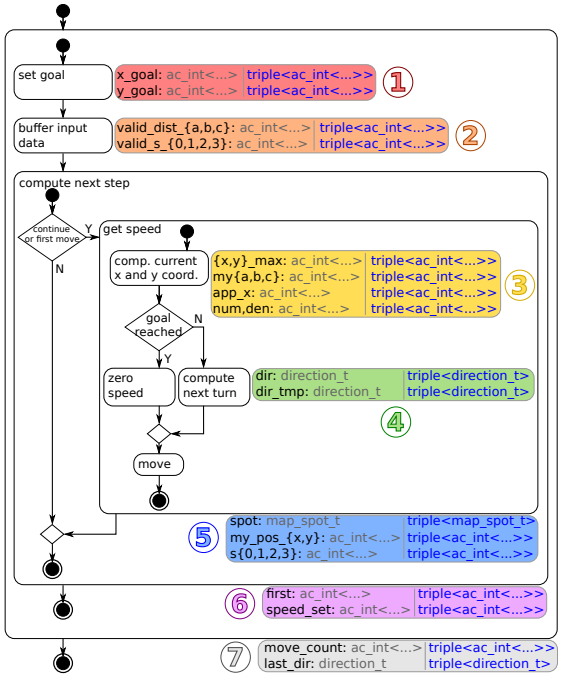

Figure 4. The extended UML AD of the robot controller having each variable hardened by the triple RDT with the variables divided into seven sets.

For each version of this robot controller, 2 000 eval-

uation runs were performed. The scenario of each run was as follows:

1. the robot controller unit was reinstated into its initial state, the maze map as well as its starting and target positions remained constant for all of the evaluation runs,
2. the *Player/Stage* simulation environment was started, the robot was placed on the starting position,
3. one SEU was injected into a bit of the bitstream, the bit was selected *uniformly at random* from all bits utilized as LUTs contents, the bit remained in a faulty state during the whole verification run,
4. the ability of the robot to reach the target position was monitored.

Table 1 shows the comparison of reliability gained for each robot controller version with the corresponding area overhead, which were calculated using the reference values of the unhardened version. The table also shows the numbers of *unary*, *binary* and *ternary* operations and numbers of *inter-*, *intra-* and *original-DT* operations associated with the hardened part using the *triple* RDT. The reliability improvement and the area overhead of each unit i were calculated using Equation 1 and Equation 2, respectively. The *reference* values of the unhardened version from our previous research were used for the calculation.

$$reliab\_improv_i = \frac{failures_{ref} - failures_i}{failures_{ref}} * 100 \quad (1)$$

$$area\_overhead_i = \frac{slices_{ref} - slices_i}{slices_{ref}} * 100 \quad (2)$$

As can be seen in Table 1, the designs with the higher overhead (numbers 2, 5 and 7) have achieved a higher level of reliability. In the other cases, our fault tolerant designs are even smaller than the *reference* unhardened design. In the cases of 1, 3 and 6, the smaller designs perform still better than the *reference* design. We believe this interesting behavior is caused by various pipelined settings the HLS tool chooses (i.e. various sizes of pipelined blocks the buffering registers are inserted in between). Further research is needed to find out the exact reason for this behavior, as it leads to significant reliability improvement when considering the chip-area consumed. It is important to note that

Table 1. The experimental evaluation of resources overhead, operations hardened and reliability gained in comparison with the unhardened reference values.

Robot Version	Ref.	1	2	3	4	5	6	7
LUTs bits [–]	21952	17408	55744	12800	15744	47552	15872	35840
Slices [–]	196	147	370	135	165	379	147	250
Failures [‰]	33.0‰	27.0‰	13.5‰	30.5‰	37.5‰	15.5‰	29.5‰	17.0‰
RDT *unary*	0	0	7	22	4	4	2	2
ops. *binary*	0	6	7	32	7	9	5	2
[–] *ternary*	0	0	0	0	0	0	0	0
RDT *inter-DT*	0	0	0	0	0	0	0	0
ops. *intra-DT*	0	0	0	30	4	0	0	2
[–] *orig.-DT*	0	6	14	24	7	13	7	2
Reliability improv. [%]	–	18.2%	59.1%	7.6%	-13.6%	53.0%	10.6%	48.5%
Area overhead [%]	–	-25.0%	88.8%	-31.1%	-15.8%	93.4%	-25.0%	27.6%

the operations are associated with variables of various bit-width lengths, thus the operations complexity varies. Therefore, the size of the resulting design does not correlate with the number of operations *hardened*. As part of our future research, we would like to try our approach with a different HLS tool that would allow us to to specify the synthesis parameters in a more detailed way. Our intention is to also research a method to estimate the importance of variables from the input algorithm without the need for the long process of the synthesis and fault injection.

5. Conclusions and Future Research

In this paper a newly emerging approach to fault tolerance of HLS-synthesized systems was briefly explained and evaluated after its application to a robot controller unit. The robot controller *C++* description was partitioned and into each partition a set of variables was assigned. Finally, each robot controller version, which included the corresponding set hardened by our approach, was evaluated. The experimental evaluation was performed using our evaluation platform, which incorporates injection into the utilized contents of FPGA LUTs. The main contribution of the paper is to demonstrate a way to evalate the importance of particular operation in HLS.

In the near-future, we would like to try our approach with a different HLS tool (possibly with some *open-source* alternative). As part of our future work, we would like to incorporate various fault tolerance methods and evaluate the impact of hardening other places in the *C++* code the fault tolerance could be applied to (mainly the control statements). And finally, from the results investigated, we would also like to focus on

the creation of a methodology to automate the whole process of design modifications and the proper form of redundancy selection.

Acknowledgements

This work was supported by The Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPU II); project IT4Innovations excellence in science - LQ1602 and BUT project FIT-S-14-2297.

References

[1] K. A. Campbell, P. Vissa, D. Z. Pan, and D. Chen. High-Level Synthesis of Error Detecting Cores Through Low-cost Modulo-3 Shadow Datapaths. In *Proceedings of the 52nd Annual Design Automation Conference*, DAC '15, pages 161:1–161:6, New York, NY, USA, 2015. ACM.

[2] M. Fowler. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3 edition, 2003.

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

[4] M. Graphics. Catapult HLS. <https://www.mentor.com/hls-lp/catapult-high-level-synthesis/>, 2017. Accessed: 2017-07-07.

[5] T. Inoue, H. Henmi, Y. Yoshikawa, and H. Ichihara. High-Level Synthesis for Multi-Cycle Transient Fault Tolerant Datapaths. In *2011 IEEE 17th International On-Line Testing Symposium*, pages 13–18, July 2011.

[6] T. Iwagaki, Y. Ishimori, H. Ichihara, and T. Inoue. Designing Area-Efficient Controllers for Multi-Cycle Transient Fault Tolerant Systems. In *2015 20th IEEE European Test Symposium (ETS)*, pages 1–2, May 2015.

[7] M. Kaneko and Y. Tsuboishi. Constrained Binding and Scheduling of Triplicated Algorithm for Fault Tolerant Datapath Synthesis. In *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1448–1451, June 2014.

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

[9] J. Lojda, J. Podivinsky, Z. Kotasek, and M. Krcma. Data Types and Operations Modifications: A Practical Approach to Fault Tolerance in HLS. In *Paper submitted to EWDTS 2017, East-West Design & Test Symposium 2017*.

[10] J. Lojda, J. Podivínský, M. Krčma, and Z. Kotásek. HLS-based Fault Tolerance Approach for SRAM-based FPGAs. In *Proceedings of the 2016 International Conference on Field Programmable Technology*, pages 297–298. IEEE Computer Society, 2016.

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

[12] J. Podivinsky, O. Cekan, M. Simkova, and Z. Kotasek. The Evaluation Platform for Testing Fault-Tolerance Methodologies in Electro-Mechanical Applications. In *Digital System Design (DSD), 2014 17th Euromicro Conference on*, pages 312–319. IEEE, 2014.

[13] A. Sengupta and D. Kachave. Generating Multi-Cycle and Multiple Transient Fault Resilient Design During Physically Aware High Level Synthesis. In *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 75–80, July 2016.

[14] A. Shastri, G. Stitt, and E. Riccio. A Scheduling and Binding Heuristic for High-Level Synthesis of Fault-Tolerant FPGA Applications. In *2015 IEEE 26th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 202–209, July 2015.

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

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

[17] XILINX. Partial Reconfiguration User Guide. <http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_1/ug702.pdf>, Apr. 2012. Accessed: 2016-09-15.

[18] Xilinx. ISE Design Suite. <https://www.xilinx.com/products/design-tools/ise-design-suite.html>, 2017. Accessed: 2017-07-07.

Paper C

Majority Type and Redundancy Level Influences on Redundant Data Types Approach for HLS

LOJDA Jakub, PODIVÍNSKÝ Jakub, KOTÁSEK Zdeněk, KRČMA Martin

In: *2018 16th Biennial Baltic Electronics Conference (BEC)*. Tallinn: IEEE Computer Society, 2018, pp. 1-4. ISBN 978-1-5386-7312-6.

Available at: https://ieeexplore.ieee.org/document/8600951

Majority Type and Redundancy Level Influences on Redundant Data Types Approach for HLS

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

***Abstract*—Due to the increasing demand for reliable computation in environments that require electronic systems to withstand an increased occurrence of faults (e.g. space, aerospace and medicine), new techniques of the so-called Fault Tolerance insertion arise. From another perspective, today's systems have become incredibly large and complex. Methodologies like High-Level Synthesis are used to reduce time to market and simplify the verification of the resulting system. In our research we focus on an implementation of Fault Tolerance into complex systems with the usage of High-Level Synthesis. In our approach, we are using newly designed Data Types that introduce redundancy on the functional level of an algorithm. In this student paper, our previously presented technique is extended by another means of redundancy and also by a new type of voting component. The systems incorporating various levels of redundancies using our approach are experimentally tested on the application of a robot controller. The paper also briefly presents the evaluation process and investigates its correct settings. The results show that the bit-based majority function is more suitable for usage with our Redundant Data Types.**

***Keywords*—High-Level Synthesis, Redundant Data Type, Level of Redundancy, Voter Component, CatapultC, Fault Tolerance.**

I. Introduction

Some electronic systems require a very high level of reliability. The reason may be because the repair of these systems is very costly or in some cases even impracticable. Another reason for high reliability demand are systems whose failure would cause high economical losses or even could endanger human health. Two main approaches to high reliable systems construction exist. The first is the so-called *Fault Avoidance* (FA) [1], which is based on a strict selection of reliable components, thus increasing the overall system reliability. The second approach is the so-called *Fault Tolerance* (FT) [2]. The FT technique accepts that the system is composed of non-reliable components while trying to hide this fact and propagate the correct result in a prescribed time, even in the presence of faults. FT is based on an incrementation of redundancy, which can be spatial, temporal or information. Basically, a fault can be distinguished as *permanent* or *transient* (i.e. occurring only for a certain period of time). In our work, we focus on simulation of permanent faults mainly, as these faults have the potential to accumulate during the system operation.

As today's systems are becoming incredibly large and complex, methodologies such as *High-Level Synthesis* (HLS) are becoming popular. The movement to the higher layer of abstraction helps to reduce time to market and simplify the verification of the resulting system. HLS in this paper is understood as a collection of methods transforming a description in a higher-level programming language into its equivalent *Register Transfer Level* (RTL) representation. In this research, our goal is to bring the advantages of using unmodified HLS in the process of FT systems design. In this paper, the combination of the spatial and temporal redundancy is used to increase component reliability. The Catapult C [3] synthesis tool in collaboration with the Xilinx ISE tool [4] is utilized in this research. The Catapult C is set up with all optimizations off (i.e. no loop pipelining or unrolling is active), as the influences of those settings were, among others, studied in our paper [5].

Generally, two approaches incorporating FT into HLS can be distinguished: 1) HLS methods modifications; and 2) description source modifications. The method in [6] focuses on modified *data-paths* synthesis with concurrent error detection ability. The authors of [7] show a method of detecting *multi-cycle* transient faults. Another approach to error detection is presented in [8]. The authors of [9] present a heuristic algorithm for searching an optimal assignment of operations to *data-paths* while considering transient faults. In opposition to all the methods mentioned, the following approach moves the problem of reliability to a higher abstraction level (i.e. the function level). The authors of [10] developed a new data type that introduces the so-called *self-checking* (i.e. the error detection technique) into *data-paths* of HLS generated systems. The authors also consider the suitability of moving such problem to a higher level of abstraction in the context of the complexity of today's systems. In the paper [11], the authors evaluated this method on an application of the FIR filter. Generally, moving to a higher level of abstraction is important, as the level of chip-integration rises. In our research, we focus on developing a reliability insertion approach easily usable with today's modern HLS tools.

This paper is organized as follows. An overview of our FT method based on Redundant Data Types is proposed in Section II. Our experimental system setup and evaluation platform are presented in Section III. The experimental results are summarized in Section IV. Section V concludes the paper and suggests our plans for future research.

II. Redundant Data Types Method

Our method is based on the modification of the input source code before it is processed by HLS. Newly created Data Types (DTs) are used as a means of redundancy insertion. The redundancy is inserted to the source code by replacing the original DT name with the name of newly created so-called *Redundant Data Type* (RDT). RDT then incorporates redundancy to all the operators and storage elements associated with the corresponding variable instance. Each of RDTs represents one method of redundancy insertion, for example, the RDT *triple* represents the well known *Triple Modular Redundancy* (TMR); the RDT *quadruple* represents *Quadruple Modular Redundancy* (4MR); and the *quintuple* represents *Quintuple*

978-1-5386-7312-6/18/$31.00 ©2018 IEEE

Modular Redundancy (5MR). Each RDT is connected to its so-called *original* data type, which implements the original data operations. The *original* data type is usually (but not limited to) one of the base types of the programming language used. In our research, RDTs are implemented using *C++ templates*.

In our previous work [12], we also used RDTs to evaluate importance of particular operation groups in a circuit. In our paper [5], three general types of *C++* modifications were identified: 1) variables (storage elements), 2) operators (arithmetic and logic operations) and 3) flow control statements. In this research, we focus on the first two types of modifications. The storage element multiplication is achieved by instantiating the variables of the *original* data type by the desired number of times. Then for unary operators, one RDT operation is performed on each instance of the *original* data type according to its original implementation. In order to allow automatic interconnections of subsystems using different reliability methods (i.e. binary operation of two non-equivalent RDTs, for example, the TMR and *duplex*), three cases must be distinguished for binary operations: a) *intra-data type* operations – RDT vs. RDT of equivalent redundancy types (e.g. TMR vs. TMR); b) *inter-data type* operations – RDT vs. RDT of different redundancy types – (e.g. TMR vs. *duplex*); and c) *original-data type* operations – RDT vs. its *original* (*unhardened*) DT (e.g. TMR vs. *unhardened* subsystem). These cases are schematically illustrated in Figure 1. For the ternary operator (i.e. the conditional operator) compatibility, the RDT must be able to cast its value to the *Boolean* data type. Each operation then includes an additional method that ensures self-synchronization (e.g. a majority function in the case of xMR).

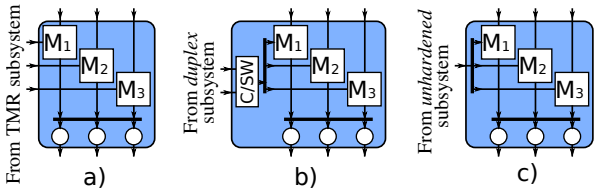

Figure 1: Three types of cases that can be distinguished when considering binary operations, (a) *intra-DT* operation between two TMR subsystems; (b) *inter-DT* operation between system with TMR and *duplex* hardening; and (c) *original-DT* operation between TMR and *unhardened* subsystems, [5].

As this approach works at the functional level, it allows for better compatibility between different HLS tools. It also benefits from the possibility of validation of implemented redundancy techniques before the development moves to the RTL. It is also easier to maintain the source code as the redundancy techniques are separated from the original code.

III. Experimental platform

The evaluation platform for testing FT properties presented in our previous work [13] is used for evaluation of the proposed methodology. Lots of real electronic systems are working together with some mechanical part. The mechanical part is usually controlled by its electronic controller. This is the reason why our evaluation platform monitors impact of faults not only on the electronic part, but also on the mechanical part. Our evaluation platform is based on the concept of the functional verification in combination with the artificial fault injection. A verified circuit (Design Under Test – DUT) is operating on *Field Programmable Gate Array* (FPGA), which allows us to inject faults directly into FPGA.

The architecture of the evaluation platform is shown in Figure 2. An electronic controller running on FPGA communicates with the simulation of a mechanical part which is running on a PC. The communication between FPGA and PC is accomplished through the Ethernet interface which is transformed to input signals for DUT. This transformation is done on the second FPGA. The verification environment which is also running on the computer monitors the communication between DUT and the mechanical part. The communication is compared with the reference model and a fault is reported in the case of a difference detection.

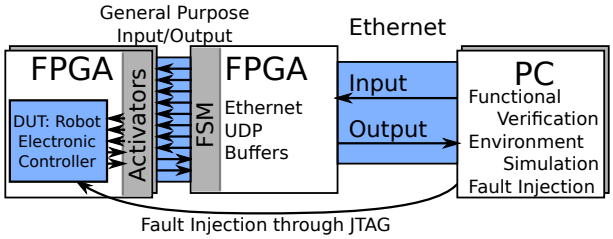

Figure 2: The architecture of our evaluation platform, [12].

The last tool working on PC is the fault injector. We use our previously published fault injector [14] which is able to inject permanent faults into a specified area of the FPGA. A permanent fault is simulated by flipping a configuration bit. Currently we are able to find a relation between bits of bitstream and the functional unit implemented on a specified place on the FPGA. For the experiments efficiency, it is very important to just inject faults into the utilized area of FPGA. Unfortunately, the current approach is only restricted on *Look-up Tables* (LUTs), so we just inject faults into the occupied LUTs. The fault injector is able to inject faults according to the specified strategy. It is possible to inject single faults during one run of the system or multiple faults within a various period. In this paper we use multiple injection with a specified rate.

The robot is exploring the maze in the Player/Stage simulation environment and its electronic controller is implemented into FPGA. The same experimental system is used in this paper, however, the robot controller is implemented with respect to the proposed methodology.

IV. Case Study and Experimental Results

For our experiments, the robot controller was fully implemented in the C++ language. Three redundancy methods were selected in combination with two majority function types: the TMR, 4MR and 5MR. For the purpose of these experiments, we decided to utilize the word-majority function (i.e selecting the word with the largest representation) and the bit-majority function (i.e for a particular bit position the value is selected by the majority representation). Generally, six RDTs (i.e. one RDT for each combination of redundancy and majority function type) were implemented according to the method previously mentioned in this paper. The overview of the RDTs used in our experiments can be seen in Table I.

TABLE I: The overview of the RDTs we have implemented as a part of our experimental work.

		Redundancy Method		
		TMR	4MR	5MR
Majority	Word	*triple*	*quadruple*	*quintuple*
	Bit	*triple_bit*	*quadruple_bit*	*quintuple_bit*

For each RDT, one robot controller unit was synthesized. Each robot controller unit was created by applying equivalent RDT to each of 30 variables in the controller's source code. Moreover, one reference robot controller unit without our approach applied (i.e. *noft*) was synthesized. The resource consumptions for each synthesized robot controller unit can be seen in Table II. As can be seen, the units with the *bit majority function* in their voter consume significantly less resources than the units with the *word majority*.

TABLE II: The overview of resource consumptions for each synthesized robot controller unit.

RDT Applied to the Robot Controller Unit Algorithm		Resource Consumption				
		Occupied Slices [-]	Slice Regs [-]	Slice LUTs [-]	Max. f [MHz]	LUTs bits [b]
noft (no RDT)		338	708	634	249.5	19392
Word Majority	triple	611	999	1109	273.4	48704
	quadruple	647	1093	1479	239.2	73216
	quintuple	999	1560	2261	195.0	122880
Bit Majority	triple_bit	535	982	730	274.7	24480
	quadruple_bit	530	1102	755	309.3	26784
	quintuple_bit	596	1357	925	284.6	37632

A. Fault Injection Intensity

Redundancy in a circuit does not necessarily improve its resilience against faults. If we actually injected one permanent fault into our DUT per verification run, the results obtained would potentially be distorted by the occupied area of the circuit tested. The same situation occurs if we injected a constant number of faults per run. It is obvious that the injection ratio should reflect the size of the DUT. This corresponds to the fact that the failure manifestation probability is in direct proportion with the circuit area occupied. For this reason we suggest incorporating the size of circuit into the unit of fault injection rate. For this purpose we chose the unit of *bit* of the bitstream as the metrics for the circuit area occupancy. The resulting fault injection unit is $injection/s/bit$.

Moreover, in our research it is important to have an ability to compare results among various versions of experimental design implementations. For this purpose, we made a series of experiments to evaluate the impact of failure rate on the resulting *Mean Time To Failure* (MTTF). As the experiments' execution is very time consuming, we set the number of 500 verification runs for the *quintuple* RDT. The results for the failure rates of $4.5e$-6 to $0.5e$-6 can be seen in Figure 3. With a decreasing failure rate the number of failed runs also decreases while the MTTF increases slightly.

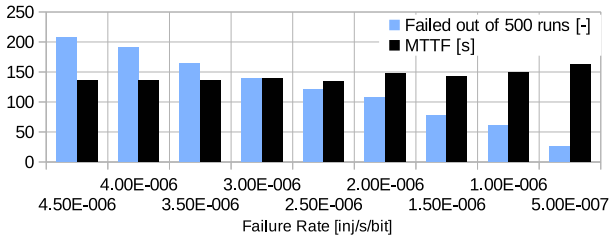

Figure 3: The impact of various failure rates on the number of failures and the resulting Mean Time To Failure (MTTF) for the controller utilizing *quintuple* RDTs.

As one maze exploration lasts for $204s$, we decided to choose the failure rate of $2.0e$-6 inj/s/bit. In this particular case, the MTTF of $148s$ was achieved, which is rational considering it is the 5MR implementation. As can be seen, while the failure rate is reduced, the number of failed runs decreases thus resulting in a less precise computation of the MTTF metrics because each run of the robot was limited to $324s$ (i.e. $204s$ + $120s$) in order to speed up the process of controller circuit evaluation (e.g. in case the robot is looped). It is important to note that the failure rate was selected only to make a fair comparison (according to the resolution scale provided by our evaluation platform) rather than to precisely simulate a real failure rate of any environment (e.g. the space).

B. Effect of Redundancy Level and Voter Type

We decided to examine $LUTs\_bits_{unit} \times 0.1$ number of runs per one robot controller unit as we assume that the number of verification runs should also be in direct proportion with the size of the DUT. The injection strategy was to inject faults into LUTs bits with the chosen fault rate of $2e$-6 inj/bit/s. Each fault injection bit was selected *uniformly at random* from all the utilized LUTs content bits. The parameters of the testing with the exact results obtained are shown in Table III.

TABLE III: The overview of the parameters of testing and the results obtained.

RDT Applied to the Robot Controller Unit Algorithm		Parameters of Testing			Results Obtained	
		LUTs bits [b]	Num. of Runs [-]	Fault Rate [inj/s/bit]	Failed Runs[%]	MTTF [s]
noft (no RDT)		19392	1940	2e-6	21.24	131.05
Word Majority	triple	48704	4871	2e-6	18.99	139.40
	quadruple	73216	7322	2e-6	20.88	138.14
	quintuple	122880	12288	2e-6	21.34	141.06
Bit Majority	triple_bit	24480	2448	2e-6	18.91	128.68
	quadruple_bit	26784	2679	2e-6	20.87	132.88
	quintuple_bit	37632	3764	2e-6	25.05	130.08

For a better illustration, the results obtained are also shown in the chart in Figure 4. First, it is important to note that, the larger the circuit is, the more faults per second are injected, because the *fault rate* is related to the number of bits of bitstream, and, thus, the comparison is more fair. However, the difference between the *noft* and reliable units looks smaller from this perspective. As can be seen the results indicate that for the *triple*, the number of failed runs is lower which is the expected situation. For the *quadruple*, the percentage of failed runs is lower than for the unhardened unit, but still higher than for the *triple* unit. We assume this phenomenon is caused by the fact the 4MR redundancy occupies more area while the majority function still has to obtain three correct results (bits) out of four to select the correct output. The MTTF confirms this assumption as for the *triple* and *quadruple*, the MTTF is nearly equivalent. For the *quintuple* robot controller unit, the percentage of failed runs is slightly higher than for the *noft*. The MTTF suggests this is caused by a higher scatter of values and the fact the robot was tested exactly for the time period of one maze traversal plus 120 seconds. As can be seen the *triple_bit* controller unit achieved the failed runs ratio of $18.91\,\%$, which is the best result. In contrast, the MTTF decreased which would also suggest the scatter of values is higher in this case. For the *quadruple_bit*, the same phenomenon of increasing failed runs ratio can be observed. For the *quintuple_bit*, the percentage of failed runs increased and the MTTF decreased. In this case the bit-based majority function does not bring better results although the resulting circuit is still much smaller than its word majority version.

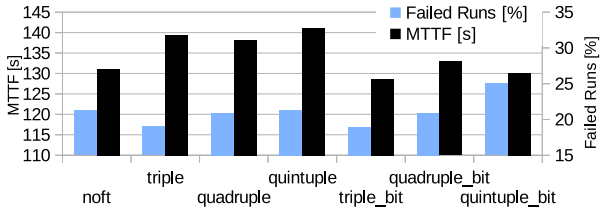

Figure 4: The overview of the results obtained through experimentation for each robot controller unit.

The conclusion of our experiments is that the bit majority function is more suitable for the purpose of usage with RDTs as it causes less overhead while keeping an almost equivalent reliability. In general, each RDT satisfies different requirements in terms of mission time or failure rate percentage.

C. Number of Verification Runs

As the number of verification runs was chosen purely empirically, we included a retrospective evaluation of the achieved precision. As a basic method of evaluation, we assume the higher the number of verification runs is, the more precise the results are. Therefore, we set the original results as a reference value and retrospectively calculated the results we would obtain if we set the number of verification runs lower. The reference value was then subtracted from each of the retrospectively calculated failure rates. By examining the deviation we are then able to get an idea of the achieved accuracy of our results. The differences in the failure rate ratios for each controller unit are shown in the chart in Figure 5. The number of verification runs for each controller unit is related to its number of LUTs bits (e.g. 0.1 in the chart represents $0.1 \times LUTs\_bits_{unit}$ verification runs).

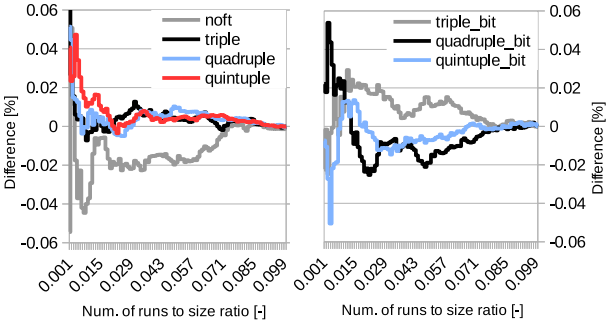

Figure 5: The retrospectively calculated failure rate differences for different numbers of verification runs.

As can be seen in Figure 5, starting from the ratio of 0.073 (i.e. $0.073 \times LUTs\_bits_{unit}$ runs) the difference from the result obtained at $0.1 \times LUTs\_bits_{unit}$ runs is kept under 0.01%. The important feature is that the differences stay at the same level after this point. Assuming the average converges to the ideal value, this suggests that the number of verification runs is sufficient considering the purpose of our evaluation.

V. Conclusion and Future Research

This paper briefly describes the approach of RDTs for usage with HLS and classifies this method in the context of other approaches. The main part of this paper evaluates the effect of the majority function type (i.e. word- and bit-based). The results show that the bit-based majority function leads to smaller circuits while keeping reasonable reliability. In addition, the paper also briefly explains the techniques utilized behind the process of the evaluation and selection of the evaluation parameters. The selected number of verification runs was retrospectively reviewed to verify its suitability.

As a part of our future research, we would like to find a key to select the proper redundancy method for a particular subsystem (i.e. function, expression etc.) and automate the process of this selection and source code modification.

Acknowledgements

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

References

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

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

[3] M. Graphics, "Catapult HLS," https://www.mentor.com/hls-lp/catapult-high-level-synthesis/, 2017, accessed: 2017-07-07.

[4] Xilinx, "ISE Design Suite," https://www.xilinx.com/products/design-tools/ise-design-suite.html, 2017, accessed: 2017-07-07.

[5] 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. 273–278.

[6] A. Antola, V. Piuri, and M. Sami, "High-level Synthesis of Data Paths with Concurrent Error Detection," in *Defect and Fault Tolerance in VLSI Systems, 1998. Proceedings., 1998 IEEE International Symposium on*, Nov 1998, pp. 292–300.

[7] A. Sengupta and D. Kachave, "Generating Multi-Cycle and Multiple Transient Fault Resilient Design During Physically Aware High Level Synthesis," in *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, July 2016, pp. 75–80.

[8] K. A. Campbell, P. Vissa, D. Z. Pan, and D. Chen, "High-Level Synthesis of Error Detecting Cores Through Low-cost Modulo-3 Shadow Datapaths," in *Proceedings of the 52nd Annual Design Automation Conference*, ser. DAC '15. New York, NY, USA: ACM, 2015, pp. 161:1–161:6.

[9] T. Inoue, H. Henmi, Y. Yoshikawa, and H. Ichihara, "High-Level Synthesis for Multi-Cycle Transient Fault Tolerant Datapaths," in *2011 IEEE 17th International On-Line Testing Symposium*, July 2011, pp. 13–18.

[10] C. Bolchini, F. Salice, D. Sciuto, and L. Pomante, "Reliable system specification for self-checking data-paths," in *Design, Automation and Test in Europe*, March 2005, pp. 1278–1283 Vol. 2.

[11] C. Bolchini, A. Miele, F. Salice, D. Sciuto, and L. Pomante, "Reliable system co-design: the FIR case study," in *19th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 2004. DFT 2004. Proceedings.*, Oct 2004, pp. 433–441.

[12] J. Lojda, J. Podivinsky, and Z. Kotasek, "Redundant data types and operations in HLS and their use for a robot controller unit fault tolerance evaluation," in *2017 IEEE East-West Design Test Symposium (EWDTS)*, Sept 2017, pp. 359–364.

[13] J. Podivinsky, O. Cekan, J. Lojda, and Z. Kotasek, "Verification of Robot Controller for Evaluating Impacts of Faults in Electro-mechanical Systems," in *Digital System Design (DSD), 2016 19th Euromicro Conference on*. IEEE, 2016, pp. 487–494.

[14] 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.

Paper D

FT-EST Framework: Reliability Estimation for the Purposes of Fault-Tolerant System Design Automation

LOJDA Jakub, PODIVÍNSKÝ Jakub, ČEKAN Ondřej, PÁNEK Richard, KOTÁSEK Zdeněk

In: *Proceedings of the 2018 21st Euromicro Conference on Digital System Design*. Praha: IEEE Computer Society, 2018, pp. 244-251.

Available at: https://ieeexplore.ieee.org/document/8491824

2018 21st Euromicro Conference on Digital System Design

FT-EST Framework: Reliability Estimation for the Purposes of Fault-Tolerant System Design Automation

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

***Abstract*—The complexity of today's systems is growing along with the level of chip integration. This results in higher demand for reliability techniques; it also increases the difficulty of incorporating reliability in such systems. For this purpose, we are working on a method to automate reliability insertion; however, for this method, it is necessary to have feedback on the result. In this paper, one component of the automation flow enabling the estimation of the resulting reliability – Fault Tolerance ESTimation (FT-EST) framework – is presented along with an improvement for accelerating the time necessary to reach the estimation. For the purpose of evaluation, we are using our Redundant Data Types approach, which enables us to intentionally insert reliability in a particular operation. The estimation utilizes the concept of fault injection. The results indicate, that the concept of Redundant Data Types is functional, however, also suggest its future improvements (e.g. for the operation of subtraction).**

***Keywords*—Fault-Tolerant, Fault Tolerance Property Estimation, FT-EST, Verification, High-Level Synthesis, Redundant Data Type.**

I. Introduction

In recent decades, electronic systems have dominated the field of controlling many important processes. For example, autonomous vehicles are becoming more and more popular. Furthermore, electronic systems handle the processes of airplane control, space aviation, etc. Many medical systems are dependent on the reliable operation of their computer controllers, as a failure of these controllers might endanger the state of health of the patient. Moreover, as the complexity of systems used in a critical environment is still growing, it is important to focus on the aspects of their reliability. Reliability improvement can be achieved through two main approaches. The first one, called Fault Avoidance (FA) [1], consists of the selection of components with prescribed quality, which enhances the overall reliability of the system. The second approach – Fault Tolerance (FT) [2] – modifies the architecture of the system in such way that it becomes reliable; however, the system remains composed of non-reliable parts.

Today's systems are designed according to new methodologies operating on a high level of abstraction. This helps designers to abstract from the details and simplify the design process. High-Level Synthesis (HLS) can serve as an example of this type of new methodology. In this paper, we use the term "HLS" to mean a collection of methods transforming a description in a higher-level programming language (e.g. C++) into its equivalent Register Transfer Level (RTL) representation in the form of another language (e.g. VHDL). Field Programmable Gate Arrays (FPGAs), towards which our method is targeted, are particularly prone to Single Event Upsets (SEUs). SEUs are caused by charged particles (e.g. a heavy ion or a proton) [3]. Charged particles traveling through an FPGA have the potential to change the state of the SRAM configuration memory, thus disrupting the functionality of the design. A small change in the SRAM configuration memory might result in a major change in the behavior of the design. For this reason, reliable systems operated in a hard environment must implement measures to eliminate the impact. The current demands on the FPGA technology tend to lower the number of bits of the bitstream sensitive to faults.

This paper is organized as follows: Related work is presented in Section II. An overview of our method for the verification of the final system is proposed in Section III. Our Fault Tolerance ESTimation (FT-EST) framework is put into the context of FT system design automation in Section IV. The method for inserting redundancy during the HLS, which is the subject of our experimentation, is presented in Section V. The proposed FT-EST framework is described in Section VI. Finally, the experimental setup and the results are summarized in Section VII. The generator, which is able to produce stimuli, is presented in Section VIII as a part of our future research. Section IX concludes the paper and mentions our future plans.

II. Related Work

During the testing of the resilience of systems against faults, waiting for faults to appear naturally is not feasible. Therefore, some special techniques were developed in order to artificially accelerate the fault occurrence. An accurate simulation method for the emulation of the effects of SEUs in the configuration memory of FPGAs is presented in [4]. This approach combines simulation and topological analysis of the design mapped on the FPGA. An analytical algorithm is able to identify the electrical effects induced into the resources of the circuit affected by an SEU. Another simulation-based injection technique, called "script-based fault injection technique", is presented by the authors of [5]. A TCL script-based automated fault injection methodology built around the target simulator, which can take designs in both RTL and netlist levels of abstraction, is proposed. The use of functional verification for FT evaluation is presented in [6]. The authors use standard verification language for fault modeling and faults are injected

978-1-5386-7377-5/18/$31.00 ©2018 IEEE
DOI 10.1109/DSD.2018.00053

during the verification run in the simulation environment. These simulator-based techniques prevent the need for designers to use an expensive FPGA board, but there is the problem that the design is not evaluated on a real FPGA.

Multi-platform fault injection based on the use of a boundary scan through the Joint Test Action Group (JTAG) interface is presented in [7]. This technique uses JTAG for observing and modifying signals in design. An FPGA-based fault injection tool, which is presented in [8], supports several synthesizable fault models of digital systems and is implemented using VHDL. However, the fault injection requires the addition of some extra gates and wires to the original design, and, thus, modifying the original VHDL. One weak point of this approach is the difference between the tested device and the device which will be manufactured.

In [9], [10], techniques which are based on fault injection into a real FPGA board without changing the original design were presented. These techniques are based on Partial Dynamic Reconfiguration (PDR), which allows them to read the configuration bitstream, inverse bits and write the affected bitstream back to the FPGA. In [10], the authors present FLIPPER. This fault injection platform is composed of two boards with FPGAs – the main board and the Design Under Test (DUT) board. The fault injection is controlled by the main board, which is driven by the software application running on a PC. The authors in [11] focus on the speed of the fault impact evaluation, where the fault injection is fully controlled by a part of the design on the FPGA. The communication with a PC is used only for the initial configuration of the fault injection process. The FPGA-based fault injection method is presented also by the authors of [12], which is demonstrated in [13]. This technique is implemented in Java and is based on the RapidSmith library [14]. The authors provide a command line interpreter that can operate in batch or interactive mode, and a graphical interface to specify the locations of permanent faults.

III. The Verification of the Final System

The evaluation of the impact of faults on an FPGA-based system lay within the scope of our previous research [15]. We proposed an evaluation platform based on modified functional verification. In our evaluation platform, we move the verified design to the FPGA, which allows us to inject faults directly into the target device. Together with the evaluation platform, an evaluation process composed of three phases was developed. The first phase is classical simulation-based functional verification. The second phase focuses on monitoring the impact of faults on the electronic part of a verified system. The third phase monitors the impact of faults on the mechanical part. Many electronic systems control some kind of mechanical part, which is the reason that our evaluation platform allows us to evaluate the impact of faults on the mechanical part. The main part of our evaluation platform is a fault injector, which allows us to inject artificial faults directly into the FPGA. We use a previously-developed fault injector [16], which is based on PDR. The faults are injected through the JTAG interface into a specified bit of the bitstream. The robot in the maze and its robot controller were used as an experimental electro-mechanical system in our previous case study. The design of the whole system allowed us to implement various types of robot controllers (hard-coded, soft-core processor, neural network, etc.). Various experiments with various fault-injection strategies have been done and presented in previous papers.

During the experiments with a developed evaluation platform, we identified some disadvantages which we plan to solve in our new evaluation approach. The evaluation of impact of the faults is very time-consuming, as many time-consuming verification runs must be performed. The evaluation platform is very advantageous in the case of the final evaluation of the whole system before it is manufactured. However, a designer needs an evaluation which is done in a short time during the development process. At the end of the development cycle, a final, full evaluation with all parts of the system (electronic and mechanical) can be performed. We identified the fact that, during the development process, fast evaluation and the identification of weakness points in the design are necessary. The designer must harden these weakness points against faults and perform another evaluation. These are the reasons why we present a new evaluation approach.

IV. Parameter Estimation for the Purposes of FT Design Automation

The general idea of our research is to propose a method and build a platform to automate the process of the insertion of FT properties into systems that were designed without FT principles in mind. The demand for such a platform is based on the constantly-increasing complexity of modern electronic systems, which makes it significantly complicated to incorporate FT properties into these systems. Moreover, chip-level integration is growing constantly, which increases the probability of SEU manifestation. This is why our previous research targeted the areas of HLS and FT, as the combination of both these principles aims to solve both problems. Our method allows the insertion of FT properties at the algorithm level without the need for significant changes of the source description code. Even given this fact, however, the designer has to insert its knowledge into the process of the decision as to which FT method to apply to which component.

A. The General Process of FT System Design

The common approach to FT system design consists of these stages [17]: 1) delimitation of the desired parameters of the system, 2) selection of fault detection mechanisms, 3) selection of recovery algorithms, 4) evaluation of the FT properties. The approach of FT system design starts with a clear delimitation of the desired reliability parameters, which are called *reliability indicators*. First of all, the set of observed indicators I is determined. A threshold value is specified for each indicator. The rest are based on an iteration process, which starts with the *nondurable* system s_0. This process includes the modification of the current version of the system $s_{i+1} = FTmodif(s_i)$ and also the evaluation of the result for each of the selected reliability indicators $\forall j, j \in I, r_j = indicator_j(s_{i+1})$. The process of the design ends with the iteration that produces the system s_x, which fulfills the threshold value for each of the selected reliability indicators $j \in I$. The actual process is illustrated in Figure 1.

B. FT System Design Automation

The structure of the automation framework will follow the previously mentioned process of FT system design. The mod-

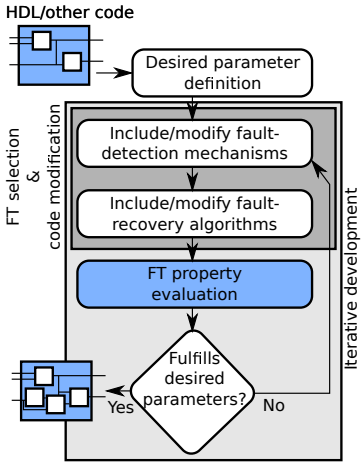

Figure 1: The process of designing an FT system from a nondurable system.

ification of the system involves text manipulation operations (on the high-level description), the time requirement of which is not critical, as well as the synthesis process. These text modifications can be further simplified by utilizing the power of the language used for the system description (e.g. for a plain VHDL, generics can be used; for HLS, C++ templates can be used; etc.). So far, the process of the evaluation of the resulting reliability indicators has been the most time-consuming part of our research. It is also obvious, the evaluation is performed for each design during each iteration, which implies the need for the acceleration. In our previous work, we evaluated our manually modified circuit descriptions (i.e. robot controller units) by a verification environment that tracks the behavior of the simulated mechanical part of the system as well. In this paper, we introduce this new framework that abstracts some details (e.g. the mechanical part simulation) and introduces new acceleration techniques such as parallel evaluation or acceleration of the most evident bottlenecks by moving them from the PC to the FPGA.

V. Implementation and Usage of Redundant Data Types

In our previous work [18], we introduced an approach to introduce an arbitrary level of redundancy into an arbitrary operation on the algorithm level. The general use of Redundant Data Types (RDTs) includes: 1) selection of FT methods and their implementation in the form of RDTs, 2) source code modification, 3) FT property evaluation.

Each method of FT is implemented in the form of one new RDT (e.g. Triple Modular Redundancy [TMR] is implemented as a new RDT called *triple*). The selection and implementation of RDTs can be prepared in advance. Then, the method of using RDTs involves the modification of the HLS input source code. The transformation of a nondurable system into its FT version is achieved through the substitution of ordinary Data Types (DTs) for RDTs. The particular variables to which this substitution is applied, determine its FT method. This principle allows us also to modify the operations performed on these variables, thus allowing us to introduce not only information redundancy, but also a combination of temporal and spatial redundancy. As a result, such an RDT has the potential to introduce almost arbitrary FT method (e.g. various modifications of the TMR principle, *duplex*, etc.). With the usage of RDTs, it is possible to significantly reduce the modifications needed to make the source code incorporate FT measures and separate the FT method from the source code. This is useful not only for the purposes of manual modifications of the source code, but also for the automated approach. In the case of an automated approach, RDTs are prepared in advance and the FT automation tool is then focused on the FT method selection, while the source code modification involves text substitution of a specific part of the code.

Each of the RDTs represents one method of FT. To maintain the functional equivalence of the code, it is necessary to keep the behavior of the DT that was previously used in the place of an RDT, which represents the FT method on the architectural level (i.e. the number of instances and the interconnections, etc.). In our research, we solve this problem by parametrization of the RDT – the name of the previously used DT is passed as a parameter to the RDT. In the context of a particular RDT instance, we call this previously used DT an *original* DT. Different subsystems utilizing different methods of FT can be interconnected. On the algorithm level, this is mainly done through an operation execution. To keep the functionality of the dynamic interconnection of subsystems according to the FT method, three interconnection possibilities must be allowed: a) *intra-data type* – RDT vs. RDT of equivalent redundancy types (e.g. TMR vs. TMR); b) *inter-data type* operations – RDT vs. RDT of different redundancy types – (e.g. TMR vs. *duplex*); and c) *original-data type* operations – RDT vs. its *original* (*unhardened*) DT (e.g. TMR vs. *unhardened* subsystem). The interconnections are schematically illustrated in Figure 2. The rules to establish these interconnections are part of the *binary* operator definitions of the particular language (e.g. in our research, we use the C++ language for its ability to easily incorporate new DTs with the usage of the C++ *templates* [19].

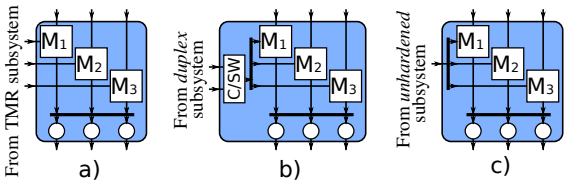

Figure 2: Three types of cases that can be distinguished when considering binary operations: (a) *intra-DT* operation between two TMR subsystems; (b) *inter-DT* operation between a system with TMR and one with *duplex* hardening; and (c) *original-DT* operation between TMR and *unhardened* subsystems.

VI. The Acceleration of FT Parameter Estimation

When designing FT systems, the designer (or a design automation system) needs to have feedback of the reliability of the current variation of the circuit. The sooner the designer obtains the information on how the current circuit performs in the terms of reliability, the better the results have the potential to be. However, in our research, the evaluation of the circuit has been the most time-consuming task so far. In the following text,

a novel approach we call FT-EST framework is described. The FT-EST framework utilizes various acceleration techniques to evaluate the reliability.

A. Iteration Types

For better understanding, let's introduce the concepts of iteration types in the FT-EST framework: 1) ***test cycle***: During the test cycle, all the selected input stimuli are tested against their golden output values and compared to equivalence. The output from one iteration of this type is whether A) the circuit meets its functionality, and, thus, we assume it was not corrupted by a fault, B) the functionality was corrupted, and, thus, we definitely know the fault manifested in the form of a failure. The output from this iteration can also be the number of failures (i.e. the number of mismatching output transactions) the fault has caused since the beginning of this iteration. 2) ***SEU cycle***: During the SEU cycle, each of the selected bits of the bitstream is attacked by a fault injection, which we simulate by a bit-flip, and tested by the test cycle to obtain the effect of the fault. During this cycle, one whole component is tested.

B. Acceleration Techniques

The framework incorporates acceleration techniques to accelerate the evaluation of the provided circuit and make the process of the evaluation more autonomous:

1) The framework is prepared to evaluate many instances of the circuit simultaneously.
2) It is possible to perform the generation of the stimuli input data on an FPGA to eliminate any bottlenecks between the FPGA and the PC.
3) The comparison of the output data is also carried out on the FPGA.
4) After each test cycle, only the Units Under Test (UUTs) are refreshed to their original bitstreams, which reduces the reconfiguration time.

Of course, the acceleration with the usage of multiple instance evaluations simultaneously is limited by the space provided on the FPGA, thus, it depends upon the FPGA area consumption of the UUT. The area of the UUT also directly specifies the number of bits of the bitstream that correspond to this unit, and, therefore, have to be tested by an SEU injection during the SEU cycle. As can be seen, the speed of the evaluation is in indirect proportion to the size of the circuit and even in a quadratic ratio. The speed of the evaluation is also dependent on the number of input transactions the UUT has to process correctly to be evaluated as resistant to the particular SEU tested in this particular test cycle.

C. The Hardware Architecture of the Framework

The FT-EST framework HW part is a modular system written in the VHDL language. A simplified diagram of the framework is displayed in Figure 3. The FT-EST framework was designed to require minimal user interactions during the setup, so it will be usable in the process of automatic FT system design. The parts of Figure 3 that are highlighted in red are the only parts of the system that a designer (or possibly another process) has to modify in order to alter the experiment flow. The parts of the system highlighted in blue are dynamically generated based on the configuration, which includes basic information about the UUT (e.g. the number of its input and output pins) and the number of instances of the UUTs the framework has to prepare.

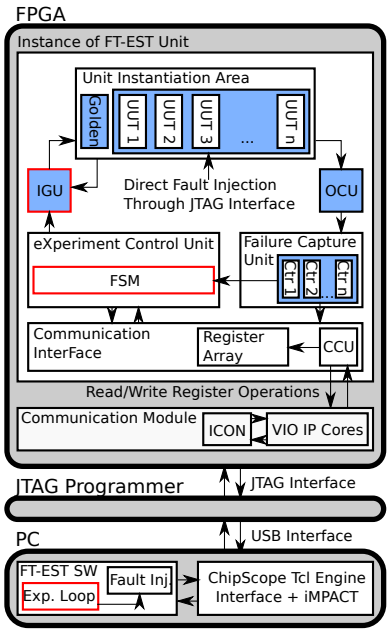

Figure 3: Simplified architecture of the FT-EST system; the parts highlighted in blue are dynamically and fully automatically generated, while the parts highlighted in red are to be provided by the designer to specify the experiment setup.

The system is composed of various modules implemented on the FPGA:

1) ***Input Generation Unit (IGU)****:* This unit generates the input stimuli. It is one of the main parts of the experiment specification, as the selection of stimuli has a significant impact on the meaning of the experiment. For the simplest UUTs (such as some proofs of concept), the basic configurations of this module can include an ordinary counter, which incrementally generates all the possible values from a given range. Another possibility is to include a random generation of stimuli. The maximum-length Linear Feedback Shift Register (LFSR), the FPGA implementation of which is discussed in [20], can be used for this purpose, as this variation of the LFSR cycles through each of the possible bit configurations except all zeros, thus avoiding the state in which a circuit would be tested for equivalent stimuli multiple times during one test cycle. Another possibility is to pre-generate the most suitable set of stimuli on the PC and utilize a BlockRAM memory to store this set on the FPGA. However, this approach is dependent on the capacity of the BlockRAMs available to the FT-EST framework.

2) ***Unit Instantiation Area (UIA)****:* The UUTs are being instantiated in this area. The number of instances is configurable, and the smallest possible configuration includes one instance of the UUT and one instance of the golden unit. The golden unit serves as a reference unit that is not subject to

fault injection, and, thus, always provides the correct results.

3) ***Output Compare Unit (OCU)****:* In this unit, analysis of outputs of the results obtained from the instantiated UUTs is performed. It compares the results obtained from the golden unit and each of the UUTs and creates a vector of differences. This unit makes the actual decision on whether a particular UUT failed or passed through the current test stimulus.

4) ***Failure Capture Unit (FCU)****:* This unit monitors the vector of differences and records the number of output mismatches. It is composed of n counters, where n is equal to the number of the UUT instances. The counters are addressable and readable by the Communication InterFace (CIF) module, which allows the PC SW to continuously read the current stats.

5) ***eXperiment Control Unit (XCU)****:* Controls the experiment process flow. The configuration of the Finite State Machine (FSM) contained inside this module is dependent on the reliability parameters measured. For example, for some applications, it might be feasible to stop the evaluation after each of the counters in the FCU module, which detects a system failure, holds a non-zero value.

6) ***Communication InterFace (CIF)****:* Serves as a platform-independent interface to the internal registers holding the configuration of the experiment. Its platform independence is the key to the portability of the framework.

7) ***Communication Module (CM)****:* It contains the interface, which is controlled by the PC. In our specific experimentation, we focus on the Xilinx FPGAs; thus, we used the Xilinx specific implementation of the JTAG interface utilizing the ChipScope Pro Integrated CONtroller (ICON) core [21] and the Virtual Input/Output (VIO) core [22]. These IP cores connect together and provide data signals inside of the FPGA controllable from the PC.

D. The Software Part of the Framework

The system is also composed of a PC SW, which controls the HW counterpart through high-level commands issued through the communication registers:

1) ***FT-EST Software****:* On the PC, there is SW we developed to control the experiment on the HW counterpart and to report the results. The structure is fully modular, so it is possible to completely switch the target technology without impacting the functionality. The only limitation is that the target platform must support a way to inject faults into the configuration bitstream, such as PDR [23]. The target platform must also support partial bitstream creation with its relocation and also must support a way to communicate with the platform (such as the Ethernet, JTAG, etc.). The main experiment controlling loop inside this module has to be adapted according to the experiment strategy.

2) ***Fault Injector****:* We use the previously developed fault injector [16], which was integrated into the FT-EST SW. In our case, the fault injection is based on the PDR. Part of our injector is also a toolkit, which allows us to select particular parts of the bitstream, based on the location of the instantiated component and allows us to filter out just the Look-Up Tables (LUTs) configuration bits.

3) ***Tcl Engine Interface****:* To communicate with the HW part of our solution, we use a ChipScope Engine Tcl Interface [24], which allows us to set the address registers and read and write data registers of the CIF through the CM on the HW side. We also use an iMPACT tool to download the bitstream data and renew the states of the UUTs after each iteration of the SEU cycle.

E. Making More Instances of Identical Bitstreams

The acceleration techniques of our approach involve, among others, multiple instantiations of the same UUT. For this reason, we need to ensure that all of these UUTs are synthesized equally and that each n-th bit of the partial bitstream has an equal function between different instances of the UUT. It is also necessary to transfer the tested design to the final practical implementation without disturbing its properties, because the same system synthesized multiple times might have different properties (e.g. based on the area on the target FPGA). We believe the solution to this problem can be solved by using the automated bitstream relocation technique [25], which would also allow us to transfer the final realization of the component to the final implementation without major changes to its architecture even on the lowest level of abstraction, considering that the same platform is used for the final implementation.

VII. The Experiments and Results

For our experiments, we decided to utilize the FT-EST framework to evaluate our previously presented approach of RDTs. A specific configuration of the FT-EST platform is discussed later in this text. For this evaluation, we specified three testing algorithms, which were afterwards translated into the VHDL using the HLS and then instantiated in the FT-EST framework and evaluated.

A. Benchmark Algorithm/Circuit Selection

We decided to put simple unsigned addition and signed subtraction as the first two algorithms. Each of these operations is performed on two 16-bit vectors, with the output being 16-bit as well. This approach should evaluate the correctness of the principle of a particular RDT operation, as one particular operation implementation is addressed, and, thus, it is possible to isolate the other influences, which is not possible in a large design with multiple operations. For the third algorithm, a Cyclic Redundancy Check (CRC) was selected. For this test, we utilized its 8-bit version – CRC-8 – with an input data of 32 bits in length. The overview of the benchmark circuits we selected is summarized in Table I.

TABLE I: An overview of the algorithms/circuits selected for the purposes of benchmarking.

Algorithm	Inputs	Outputs
Addition	A: 16-bit unsigned int. B: 16-bit unsigned int.	$A + B$: 16-bit unsigned int.
Subtraction	A: 16-bit signed int. B: 16-bit signed int.	$A * B$: 16-bit signed int.
CRC-8	A: 32-bit data	$CRC_8(A)$: 8-bit checksum

B. The Synthesis of the Circuits and the HW Platform

At first, the algorithms were implemented in a plain C++ language. For each algorithm, two implementations were made: 1) a *simplex* implementation, which did not involve any FT techniques to serve as a reference unit, and 2) a TMR implementation, which had each variable sanitized using the approach of RDTs. These implementations were then put into the HLS. In this experiment, we are using the Mentor Graphics CatapultC University Version (UV) 8.2b [26]. During the HLS, we turned off all the acceleration techniques, such as *pipelining* or *unrolling*. Also, optimization was turned off, to make sure the synthesis process does not remove the redundancy we intentionally inserted into the algorithm code. As a result, we obtained a VHDL implementation of the algorithms. Before the evaluation, these VHDL implementations were instantiated as a UUT in the UIA of the FT-EST unit. For the TMR implementations a voter was added behind the component to perform the final voting. For the simplex version, the UUT was instantiated without additional components. The architectures for both these cases are shown in Figure 4.

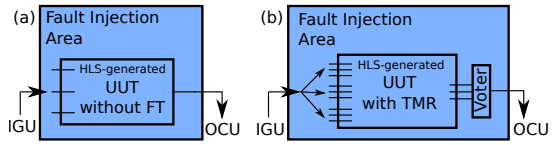

Figure 4: The architecture of the testing for (a) a simplex component and also for (b) the TMR component utilizing the concept of RDTs.

Finally, the prepared FT-EST unit was synthesized using the Xilinx Integrated Synthesis Environment (ISE) 14.7 [27]. The resulting bitstreams were then evaluated on the ML506 board [28] utilizing the Virtex 5 technology.

C. FT-EST Configuration Parameters

During the evaluation, we set the FT-EST framework to generate a permanent bit-flip for each bit of the utilized contents of the LUTs. Each of the circuits consumes a 32-bit width input data, thus, for each SEU, the FT-EST tested the UUT through a range from 0 to $2^{32} - 1$ at a step of 43 resulting in approximately 100 millions of combinations. This settings was chosen on an experimental basis considering the amount of faults detected and the time spent. For this particular experimentation, we omitted the state behavior of the UUTs, as the UUTs act as combinational circuits. For such simple circuits, we just omit this fact, as the state space of the tested stimuli is so large that such a failure state would very likely manifest itself in a form of a failure (i.e. output data disruption).

D. FT Property Estimation Results

The experimental results were obtained through the steps described in the previous text. We focused on the number of output disturbances. We traced the cases in which, for a given SEU, the UUT propagated one or more results that were not equivalent to the results of the golden unit. Table II shows the actual numbers of injections and also the numbers of injections that caused an output mismatch as well as the percentage of sensitive bits. As can be seen, the application of the RDT approach utilizing a TMR led to a better reliability (i.e. a lower percentage of sensitive bits) for each benchmark algorithm. However, the impact of the RDT is dependent on the operation. For example, in the case of the CRC-8, RDTs managed to lower the percentage of sensitive bits from 34% to 13%. Similarly for the addition. Although, in the case of subtraction, this approach was less efficient lowering the sensitive bits from 4% to 3%. We believe this dissimilarity is caused by the fact that the CRC-8 actually uses more than only one operation to compute, thus, resulting in more opportunities for the RDT to show its effect. Also, the TMR versions of the UUTs do not utilize exactly three times the bits of the simplex versions. This is caused by the approach of RDTs, as it is a data-path oriented approach, and, thus, it leaves the control-path untreated.

TABLE II: The number of SEUs that caused an output mismatch.

Algorithm	FT method	LUT bits total [b]	Num. of inj. [-]	Num. of disturbances [-]	Sensitive bits [%]
Addition	none (simplex)	4288 b	4288	890	20.76 %
Addition	TMR	8320 b	8320	225	2.70 %
Subtraction	none (simplex)	4288 b	4288	178	4.15 %
Subtraction	TMR	8320 b	8320	278	3.34 %
CRC-8	none (simplex)	4800 b	4800	1658	34.54 %
CRC-8	TMR	6592 b	6592	879	13.33 %

We were interested not only in the number of manifested failures but we also monitored the number of mismatches each particular SEU produced (i.e. the number of output mismatches during one test cycle). For example, various SEUs caused the CRC-8 simplex UUT to produce various numbers of error outputs, however, the results show, that the application of RDTs reduced the median of erroneous outputs per SEU to less then one half of the median of the simplex unit. One exception is the subtraction, for which this is not true. However, for this case, one can see the TMR version produced always nearly equivalent number of faults. Unfortunately, for now we are not able to observe the actual data the UUT propagated but the fact the number of failures is always the same might suggest one case, the treatment of which would significantly improve the robustness of the subtraction operation in the RDT. The results for each UUT are summarized in a boxplot chart in Figure 5.

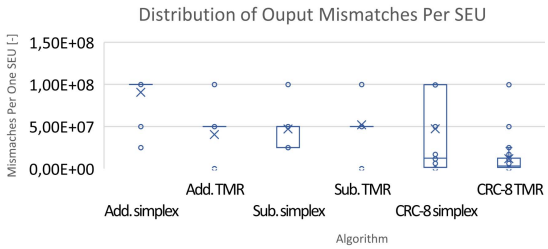

Figure 5: The distribution of mismatched outputs quantities caused by one SEU during one test cycle (test cycles that did not show any output errors are omitted; median value is marked by a cross).

E. The SEU Coverage

As another part of our experiments, we wanted to evaluate whether the SEU coverage could be lowered without significant impact on the precision of the estimation. For example, if we evaluated only one half of the utilized LUTs content bits selected uniformly at random from all the bits available, would the resulting estimation still hold the certain level of accuracy? For this evaluation, we utilized detailed logs of the previous experimental runs to simulate this behavior. We made 1000 runs per benchmark unit and SEU coverage. Each run simulated one of the selected SEU bit coverages. The results were compared to determine the spread of the accuracy. The results in Table III show the dispersion interval of the deviations. The deviation is calculated in the unit of percentage points. As can be seen, lowering the SEU coverage impacts the accuracy of the evaluation but the accuracy might still be useful as an estimation during the FT design. The estimation accuracy seems to be higher for larger UUTs, although this is true only within one type of the algorithm (e.g. FT-EST achieved better accuracy for the subtraction TMR than for the subtraction simplex). It is also important to note, that 10% SEU coverage results in a ten times shorter evaluation time, which has significant impact on the FT design automation runtime and allows to explore more configurations of the state space.

TABLE III: The deviation of the estimations for various SEU coverage settings (less is better).

SEU cove-rage	Deviation Range of the Estimation [% points]					
	Addition simplex	Addition TMR	Subtraction simplex	Subtraction TMR	CRC-8 simplex	CRC-8 TMR
60 %	1.63 – -1.63	0.46 – -0.40	0.70 – -0.89	0.63 – -0.46	1.71 – -1.94	1.1 – -0.97
30 %	2.57 – -2.47	0.98 – -0.78	1.91 – -1.20	0.91 – -1.1	3.38 – -3.64	1.99 – -2.46
10 %	6.06 – -5.36	1.74 – -1.50	2.61 – -2.52	1.71 – -1.9	6.29 – -6.21	4.26 – -3.78
5 %	11.0 – -9.6	2.58 – -1.98	4.71 – -3.69	3.15 – -2.62	9.63 – -9.54	5.78 – -5.14
1 %	16.6 – -16.1	6.91 – -2.7	12.2 – -4.15	8.68 – -3.34	21.7 – -19.96	18.5 – -11.8

VIII. Stimuli Generation Method

As part of our future research, we would like to utilize a method to generate the stimuli for more complex systems as stimuli generation is a very important process in checking the correct behavior of any system. Obtaining a stimulus or a set of stimuli that adequately covers the entire state space can greatly reduce the overall time for testing the system.

The universal stimuli generator is based on the theory of grammar systems. For these purposes, we have designed our own grammar system – probabilistic constrained grammar (already introduced in the paper [29]), which is based on probabilistic context-free grammar. Probabilistic context-free grammar is a common context-free grammar that has a defined probability for its production rules with which they are applied. Probabilistic constrained grammar extends the probabilistic context-free grammar about constraints which are capable of modifying the probability values during a generation process (the application of production rules). This makes it possible to control the application of production rules to ensure the suitability and validity of the stimulus for the system.

The architecture our universal stimuli generator is based on two input structures which define grammar and constraints. The Generator Core performs the leftmost derivations (application of production rules) and takes into account/applies the defined constraints. After replacing all non-terminal symbols of a defined grammar, the output is a string which contains only terminal symbols representing the resulting stimulus.

Our probabilistic context-free grammar is described primarily by a set of production rules. Our conventions for symbols of grammar are as follows: Non-terminal symbols are in capital letters, while we consider any string in single quotation marks to be terminal symbols. We always consider a non-terminal marked with an *S* character to be a start symbol. Each production rule always has a non-terminal symbol on its left side, while its right side consists of a combination of non-terminal and terminal symbols or ϵ (*eps*). The symbols are separated by spaces. In parentheses after each production rule, we can specify the percentage value of the probability with which the rule will be applied. If the probability value definition is missing, it is automatically calculated with respect to the other defined probabilities of the rule. Each rule must end with a dot. For a more efficient write, the production rules can be merged using a comma. An example of the definition of production rules for the probabilistic context-free grammar that generates a simple linear equation is shown below:

```
S -> LEFT ' = ' RIGHT EX .
LEFT -> LEFT OP LEFT, VAL .
RIGHT -> RIGHT OP RIGHT, VAL .

VAL -> NUM, NUM 'x', NUM '(' VAL ')' .
EX -> OP 'x', eps(0%) .
OP -> ' + ', ' - ' .
NUM -> '1', '2', '3', '4', '5', '6', '7' '8', '9' .
```

The proposed grammar would generate inequality without the *EX* non-terminal (*2 = 6*), therefore, it is necessary to add a rule that adds the variable *x* to the equation at any cost. However, this solution is not ideal, because it decreases the space of all possible solutions. This means, for example, that it is not able to generate the equation *2x = 2*, because it always attaches an extra *x*: *2x = 2 - x*. This problem can be solved by a constraint through which we have expanded this grammar to gain more expressive power. To add the constraint, we have already defined the second rule *EX* replaces with *eps* in the grammar, which will serve as the second replacement option.

The constraints are defined by the keyword cons followed by 5 parameters. The first parameter specifies the activator, a rule that causes a change of probability after a replacement. The second parameter is a rule that gets a new probability. The third parameter is the new probability value. The fourth parameter specifies a rule that cancels the set probability through this constraint after a replacement. Through the last parameter, the number of replacements of the rule under the forth parameter before the cancellation of set probability can be specified. For the grammar above, we add this constraint:

```
cons(VAL -> NUM 'x',EX -> eps,100);
```

This constraint has only three parameters which will cause the *EX -> eps* rule to have a probability value set to 100% after the application of the *VAL -> NUM 'x'* rule. If *x* is generated at any time during the generation process using the rule, the *EX* non-terminal symbol will always be replaced by *eps*. If this rule is not applied and the variable *x* is not generated, the rule *EX -> OP 'x'*, which will still have the probability at 100%, is used. Using these constraints, we are able to define and generate more complex input stimuli.

IX. Conclusions and Future Research

This paper describes a novel approach, which we call FT-EST framework, to accelerate the estimation of the impact of SEUs. The FT-EST framework was used to evaluate the previously presented approach of RDTs, which serve as an instrument to incorporate redundancy on the algorithm level before its processing by the HLS. The size of the UUT is limited only by the FPGA capacity and by its testability or how good test coverage is achievable by the chosen stimuli generator. As for latent faults, the detection is possible through the bitstream read-back, although, this requires a detailed information of which bit of the bitstream covers the storage function, which we do not have implemented at the moment. The results we obtained indicate that the concept of RDTs is functional, however, suggests its future improvements (e.g. for the operation of subtraction). With the ability to evaluate UUTs at a high speed the development of the FT automation tool will be much easier. In addition, this paper also briefly explains the techniques we plan to utilize to generate the stimuli inputs for systems requiring more complex input transactions. As a part of our future research, we would like to utilize the stimuli generation technique and to incorporate the presented approach to a larger system, which will utilize the outputs of the evaluation during the FT design automation (i.e. to select the proper FT method for a particular component or partition).

Acknowledgements

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

References

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

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

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

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

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

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

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] J. Podivinsky, O. Cekan, J. Lojda, M. Zachariasova, M. Krcma, and Z. Kotasek, "Functional verification based platform for evaluating fault tolerance properties," *Microprocessors and Microsystems*, vol. 52, pp. 145 – 159, 2017.

[16] 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.

[17] J. Hlavička, S. Racek, P. Golan, and T. Blažek, *Číslicové systémy odolné proti poruchám. 1st edition, Prague, Published by: ČVUT, 1992, 330 s*.

[18] 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.

[19] D. Vandevoorde and N. M. Josuttis, *C++ Templates*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[20] A. K. Panda, P. Rajput, and B. Shukla, "FPGA Implementation of 8, 16 and 32 Bit LFSR with Maximum Length Feedback Polynomial Using VHDL," in *2012 International Conference on Communication Systems and Network Technologies*, May 2012, pp. 769–773.

[21] 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.

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

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

[24] Xilinx Inc., "ChipScope Pro 11.4 Software and Cores User Guide," https://www.xilinx.com/support/documentation/sw_manuals/xilinx11/chipscope_pro_sw_cores_ug029.pdf, Dec. 2009, accessed: 2018-02-15.

[25] A. Laleve, P. H. Horrein, M. Arzel, M. Hbner, and S. Vaton, "AutoReloc: Automated Design Flow for Bitstream Relocation on Xilinx FPGAs," in *2016 Euromicro Conference on Digital System Design (DSD)*, Aug 2016, pp. 14–21.

[26] M. Graphics, "Catapult HLS," https://www.mentor.com/hls-lp/catapult-high-level-synthesis/, 2017, accessed: 2017-07-07.

[27] Xilinx Inc., "ISE Design Suite," https://www.xilinx.com/products/design-tools/ise-design-suite.html, 2017, accessed: 2017-07-07.

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

[29] O. Cekan and Z. Kotasek, "A probabilistic context-free grammar based random test program generation," in *2017 Euromicro Conference on Digital System Design (DSD)*, Aug 2017, pp. 356–359.

Paper E

Automatic Design of Reliable Systems Based on the Multiple-choice Knapsack Problem

LOJDA Jakub, PODIVÍNSKÝ Jakub, ČEKAN Ondřej, PÁNEK Richard, KRČMA Martin, KOTÁSEK Zdeněk

In: *Proceedings - 2020 23rd International Symposium on Design and Diagnostics of Electronic Circuits and Systems, DDECS 2020.* Novi Sad: Institute of Electrical and Electronics Engineers, 2020, pp. 1-4. ISBN 978-1-7281-9938-2.

Available at: https://ieeexplore.ieee.org/document/9095576

Automatic Design of Reliable Systems Based on the Multiple-choice Knapsack Problem

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

***Abstract*—This paper evaluates the practical usage of the Multiple-choice Knapsack Problem (MCKP) solver to automatically select the proper fault mitigation method for each component to maximize the overall fault tolerance of the whole system. The usage of the MCKP is placed into the context with our fault tolerance automation toolkit, the goal of which is to completely automate the process of fault-tolerant system design on a very general level. To achieve our goal, we present our research on Field Programmable Gate Arrays (FPGAs) for which we have developed the specific components in order to support their fault-tolerant design automation. In our particular case study, the MCKP method on the partitioned system was able to find the solution with 18% less critical bits compared to our previous approach, while even lowering the circuit size. The results indicate that by splitting the system into smaller components and applying the MCKP method, considerably better results in terms of critical bits representation can be achieved.**

Keywords*—*Fault-Tolerant System Design, Electronic Design Automation, Multiple-choice Knapsack Problem, Fault Tolerance Property Estimation, Verification, High-Level Synthesis.

I. Introduction

As electronic systems penetrate into areas with increased reliability demand, the pressure on designers to make such systems reliable arises. Generally, two main approaches to reliable system design exist: 1) *Fault Avoidance* (FA) [1], which is based on the better selection of proper and more reliable components and does not change the structure or interconnections of the system; and 2) *Fault Tolerance* (FT) [2], which accepts unreliable components as a fact and tries to solve the problem of higher reliability with modification of the system structure. Our research is based on the FT approach. Also, growing complexity of electronic systems led to strategies and design flows that maintain the *Time To Market* (TTM) on a reasonable level. *High-level Synthesis* (HLS) is a good example of such a design flow. Generally, HLS allows a designer to utilize the description written in one of the higher-level programming languages and transform it to its *Register Transfer Level* (RTL) implementation in VHDL or Verilog. In our research we focus on FT design automation and also on its combination with HLS, because our concept of FT design automation is general, and thus is able to cover different design flows.

So far, our research has been targeting SRAM-based FPGAs. SRAM-based FPGAs store their configuration bitstream in an SRAM memory, and thus are prone to the so-called *Single Event Upset* (SEU) bit-flips. A benefit of using FPGAs is also their good usability in the process of FT design testing, because the concepts can be easily tested on a real HW with the usage of the so-called *fault injection*. During the test of an FT circuit, the approach of fault injection is usually combined with the so-called *functional verification* in order to detect the failure of the tested unit. The bits that cause a discrepancy on the output pins during the verification are called *critical bits*, sometimes in the literature also referred to as *sensitive bits*. The percentage of critical bits in an FPGA design is usually understood as a quantified measurement of FT of the design.

The usual step in the process of FT design is to assign a suitable combination of FT techniques to harden individual blocks against faults that would lead to higher FT of the whole circuit. Solutions to these reliability allocation problems can also be found in literature. The *Improved Surrogate Constraint* (ISC) method was applied to the system reliability allocation problems with a mix of components in paper [3]. The authors of [4] proposed the penalty guided artificial bee colony algorithm. The paper [5] presents the use of the variable neighborhood search meta-heuristic method. The use of dynamic self-adaptive multi-objective particle swarm optimization method is proposed in [6]. The usage of the genetic algorithm was examined in [7] and [8], where the use of *Non-dominated Sorting Genetic Algorithm II* (NSGA-II) was presented. The experiments show that the NSGA-II can find a number of promising solutions of the reliability allocation problem. Most of the presented work is a separate solution to this problem without a broader concept. In our research, we bring the integration into the complex tool which targets the automation of FT design process. As the fundamental algorithm for our experiments, we decided to transform the problem of redundancy allocation to the MCKP [9].

This paper is organized as follows: Section II introduces the concept of our FT design automation toolkit. The use of MCKP for FT strategy selection is proposed in Section III. The case study and experimental results are presented and discussed in Section IV. Section V concludes the paper and presents plans for our future research.

II. Fault-Tolerant Design Automation

In our research, by FT design automation, we mean the transformation of the so-called *unhardened system* description to its FT version. Our aim is to research FT design methods that allow automatic FT selection and insertion while keeping the methods as much general as possible, with the ability to specialize on particular design flow through addition of special modules. The structure of our platform comes from the ordinary process of FT system creation, that is: 1) specification of the required parameters, and 2) iterative modification and evaluation until the specific requirements are met.

Our FT design automation platform consists of various components and each component targets particular phase or task during the transformation of the system. The first is the component implementing the FT strategy selection, which is the main topic of this paper. The FT strategy selection decides which type of FT method to use for which part of the system. The so-called *helpers* are used during this process. The helpers include libraries or possibly modification

978-1-7281-9938-2/20/$31.00 ©2020 IEEE

scripts, that are able to incorporate redundancy into a particular component of the system. So far, we have been developing specific helpers for the usage with HLS, which we call the *Redundant Data Types* (RDTs) [10]. The RDTs act as new data types in the algorithm description and *decorate* the resulting system to decrease its critical bit representation. After each iteration, the draft of the system is evaluated. The evaluation is performed on the real FPGA HW. We use the *Fault Tolerance ESTimation* (FT-EST) framework [11] to automatically build test-benches. The data obtained through the FT-EST framework can be further examined by the FPGA bitstream-specific analysis [12] to obtain numeric quantification of FT indicators.

The general overview of the traditional approach with a designer making manual operations and the connections to our automatic flow utilizing processes, can be seen in Figure 1.

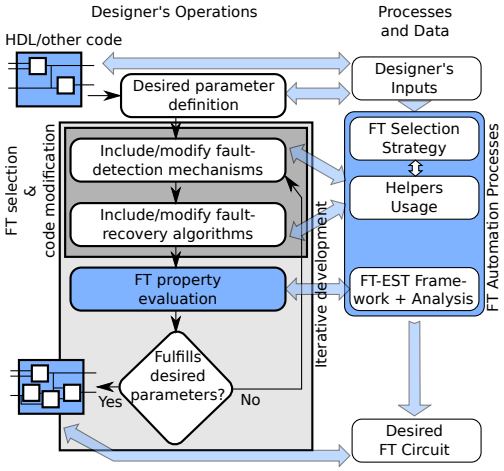

Figure 1: Design of FT system from an unhardened system with the designer's operations mapped to the processes of our automation platform.

III. Fault Tolerance Strategy Selection

For our testing, we decided to map the problem of redundancy selection to the MCKP. The MCKP is a special type of the Knapsack Problem (KP). KP and its variations belong to the so-called combinatorial optimization problems [13]. The KP as an optimization is an NP-hard problem. Let's suppose we have a *knapsack* of a particular load *capacity* and a set of objects. Each object has attached the values of the so-called *profit* and *weight*. Then, the KP objective is to solve the problem of the best selection of the objects to achieve the best value (i.e. profit) in the knapsack, while keeping the load below its maximum capacity [14]. The MCKP extends the original KP by distinguishing the objects into classes. In addition, in MCKP, exactly one object from each class must be selected. The general objective is equivalent – to maximize the profit while keeping the load below the given capacity [9]. The MCKP can be mapped to the problem of redundancy selection. However, it is important to note that for our purposes, we slightly modify the MCKP problem to prefer the units with the *smallest handicap*, instead of the highest profit. The handicap is actually the number of critical bits, in our terms and the capacity is the chip area available (e.g. bits of the bitstream). We decided to use critical bits as the metrics, as their representation determines the SEU resistance of a design. Other metrics such as time required to compute a result may be also used for a component, in such case, however, components in the system would have to use a communication protocol because of timing differences. Also the final throughput of the complete system would have to by analyzed separately. Graphical illustration of the usage of MCKP for redundancy selection can be seen in Figure 2.

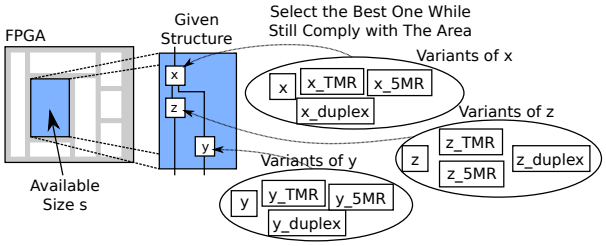

Figure 2: The graphical representation of FT system and the selection strategy based on MCKP.

In our research we chose to evaluate the weights and profits of each component, and then solve the MCKP fully in SW. After that, create the composed unit according to the results obtained from the MCKP solver. This is the reason why our FT strategy has actually just one iteration on the level of the FT automation toolkit because the state-space exploration is performed in the MCKP fully in SW. For this method to work, it is necessary to evaluate each variant of each component to obtain the input parameters for the MCKP solver.

IV. The Case Study and Experimental Results

In our case study, we designed an electronic system that is suitable for our demonstration purposes. The system literally computes the number of 1 bits (i.e. high bits) in the sum of three numbers, of which one is a static constant. The system also computes a CRC-8 checksum of data obtained after the first and second additions. The system can be partitioned into four components: 1) addition; 2) addition of a constant; 3) number of high bits computation; and 4) CRC-8 checksum computation. The block diagram can be seen in Figure 3.

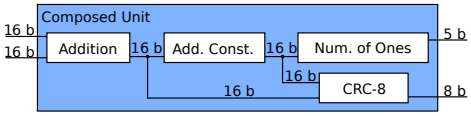

Figure 3: The schematic of the system with its components.

A. Components of Benchmark System and Their Implementation Properties

The component *addition* sums two 16-bit unsigned numbers and provides the result also on 16 bits. The component *Add. Constant* which will be referred to as *addconst* further in the text, adds a constant number to its one 16-bit input. The component *crc8* computes the *Cyclic Redundancy Check* (CRC) out of the 32 bits wide vector. The output is 8 bits wide. The last component, *Number of Ones*, provides an unsigned 5-bit number representing the quantity of high bits in its 16 bits wide input vector. This component will be referred to as *numones*.

For the implementation of the components, we used a traditional HLS design flow utilizing the C++ language. Each component was implemented in four variations incorporating different amount of redundancy in their data-path level using the RDT approach: 1) *simple* (no redundancy); 2) *triple* (data-path triplicated); 3) *quadruple*, (four data-paths); and 4) *quintuple* (five data-paths). Component descriptions were synthesized using the Mentor Graphics Catapult C HLS tool [15].

For each of these circuits, the FT-EST test bench was created. Test benches were synthesized using the Xilinx Integrated Synthesis Environment (ISE) 14.7 [16]. After that, each bit of utilized LUT contents was exhaustively tested on the ML506 evaluation board using Virtex 5 technology. The overview of the synthesized components can be seen in Table I. As can be seen, each RDT (i.e. each FT insertion method) has different efficiency on a component. This property actually suggests that each unique component suits different FT method that achieves the best parameters.

TABLE I: Component Variations and Their Properties

Component Name	Number of Inputs [b]	Num. of Outputs [b]	Used LUT bits [b]	Critical bits [b]/ KP Handicap [-]	Critical bits Repr. [%]
addition_simple	16 b; 16 b	16 b	4288	162	3.78
addition_triple	3x 16 b; 3x 16 b	3x 16 b	8320	163	1.96
addition_quadruple	4x 16 b; 4x 16 b	4x 16 b	10304	195	1.89
addition_quintuple	5x 16 b; 5x 16 b	5x 16 b	14528	232	1.6
addconst_simple	16 b	16 b	4224	104	2.46
addconst_triple	3x 16b	3x 16 b	8096	130	1.61
addconst_quadruple	4x 16b	4x 16 b	9952	171	1.72
addconst_quintuple	5x 16b	5x 16 b	14144	198	1.4
crc8_simple	32 b	8 b	4800	977	20.35
crc8_triple	3x 32 b	3x 8 b	6592	819	12.42
crc8_quadruple	4x 32 b	4x 8 b	6976	712	10.21
crc8_quintuple	5x 32 b	5x 8 b	7360	971	13.19
numones_simple	16 b	5 b	4096	380	9.28
numones_triple	3x 16 b	3x 5 b	4800	122	2.54
numones_quadruple	4x 16 b	4x 5 b	5312	125	2.35
numones_quintuple	5x 16 b	5x 5 b	5184	120	2.31

To connect components of various redundancy levels, VHDL bit-based voter was utilized. If there is a requirement to ensure that no component is shared on the system, *pblocks* in the Xilinx PlanAhead software [17] can be used.

B. MCKP Reaction to Area Available on the FPGA

In the first phase of our experiments, the behavior of the MCKP for various chip area settings was tested. We selected the number of LUT bits to represent the chip area occupied. Empirically, the interval from 17500 to 21100 with the step of 100 bits was selected. For each chip-area threshold, the MCKP solver was executed and the resulting configuration was observed. As the granularity of the MCKP reaction to different thresholds is dependent on the size of available components, we obtained six different FT configurations. The results including the actual configurations are shown in Figure 4. As can be seen, the MCKP tries to completely utilize the given area. Another important observation is that with the increasing size of the system, the absolute number of critical bits decreases, indicating the MCKP strategy targets the optimal configurations. Especially important observation is that not only the critical bit representation is lowering but also the absolute value of critical bits decreases while the circuit increases in size.

Furthermore, as can be seen, the MCKP selects to harden only the *crc8* and *numones* components. As you can see in Table I, this is caused by inefficiency of the selected type of redundancy methods for the first two components (i.e. the *addition* and *addconst*).

C. Real Implementation Results

From the previous step, six system configurations were obtained. These systems will be referred to by the name

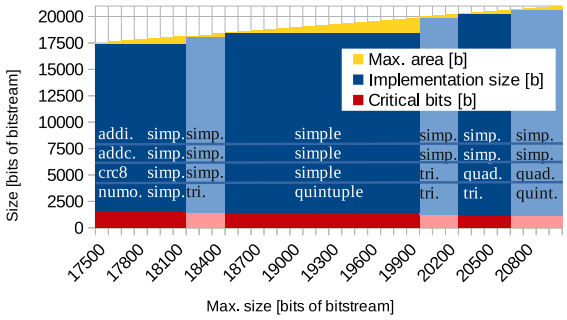

Figure 4: The theoretical computed values for the system configurations obtained by changing the available area (i.e. size) on the chip.

autocomposed_1 to *6*. We further added four reference systems that were composed without the partitioning, just by using equivalent redundancy method on the complete system. These units will be further referred to as *composed_simple*, *composed_triple*, *composed_quadruple* and *composed_quintuple*. In fact, the *autocomposed_1* system is equivalent to the *composed_simple* system, as for the smallest chip area setting, only *simple* units are selected by the MCKP solver. As a result, 9 unique systems were obtained and evaluated with the usage of the FT-EST framework on the exhaustive test of all LUT bits. The results are shown in Table II. As can be seen, after the synthesis, the autocomposed units utilize less LUT bits than indicated by the estimated values provided by a sum of components. Obviously, the synthesis is able to optimize better for larger systems, compared to the much smaller components. The numbers of critical bits, however, follow the general trend and are nearly equivalent to the values determined by the MCKP-based method.

TABLE II: Parameters of the Synthesized Systems

System Name	Num. of Inputs [b]	Num. of Outputs [b]	Used LUT bits [b]	Critical bits [b]/ KP Handicap [-]	Critical bits Repr. [%]
autocomposed_1	16 b; 16 b	5 b; 8 b	9120	1518	16.64
autocomposed_2			9856	1255	12.73
autocomposed_3			10240	1147	11.2
autocomposed_4			11684	1044	8.96
autocomposed_5			12000	968	8.07
autocomposed_6			12416	860	6.93
composed_simple			9120	1518	16.64
composed_triple			19684	1049	5.34
composed_quadruple			24448	1060	4.34
composed_quintuple			33056	1261	3.81

As can be seen, the largest autocomposed solution is still 37% smaller than the composition in which the entire system was hardened with the usage of the triple RDT. For this system, the number of critical bits is yet 18% smaller. In addition, the autocomposed units are built of the optimal components that decrease the number of critical bits with the growing size. This is not always the case for the unpartitioned units, where for example the *composed_quintuple* system contains more critical bits than its smaller counterparts. Important properties of the autocomposed and naively composed systems after their synthesis are shown in Figure 5.

Although a significant difference in the estimated and real sizes can be seen, the important fact for us is that the estimation of critical bits (i.e. the overall KP handicap) remained nearly equivalent after the systems were composed and synthesized.

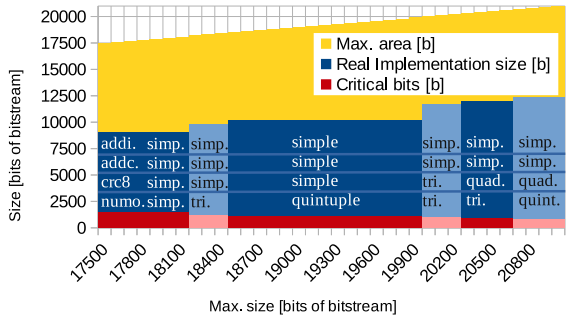

Figure 5: Properties of the synthesized configurations obtained by changing the available area (i.e. size) on the chip.

D. Comparison of Partitioned vs. Homogeneous FT Selection

As can be observed in Figure 6, systems on which equivalent redundancy method was applied to each component (i.e. left part of the chart) have significant steps (i.e. low granularity) in their implementation sizes according to utilized redundancy method. This increases the overhead of the solution, as for a given space on an FPGA, implementation that is smaller than the given space must be selected, resulting in an unused FPGA area. On the other hand, systems on which each component can be hardened using different redundancy method (i.e. right part of the chart) have better granularity among the resulting sizes of the system. Furthermore, the automatic method based on the MCKP solver follows the trend of lowering the absolute number of critical bits, which is not always the case with the *composed* units. Also, the assumption, that each component must be hardened using its most suitable redundancy method can be also confirmed from the chart. It is obvious, that the *autocomposed* units achieve equivalent or better results while occupying significantly smaller area.

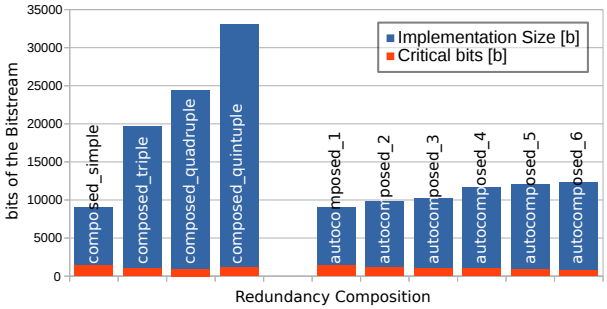

Figure 6: Comparison of partitioned (i.e. autocomposed) systems with systems hardened using equivalent redundancy method (i.e. manually composed systems).

V. Conclusions and Future Research

In this paper, the possibility to utilize the MCKP solver to assign redundancy types to components of a system was presented. The usability was put into the context of the main goal of our research which is the automation of FT design process. In our case study, the method was presented on an artificially created system which was built of four components. The MCKP solver in combination with our FT design automation toolkit was able to optimize the selection and its results contained 18% less critical bits compared to our previous naive selection. In addition, the circuits assembled according to the MCKP solver were much smaller because the solver dismisses the sub-optimal configurations. Furthermore, the results indicate that dividing the system into smaller components and searching the best solution for each component leads to considerably smaller circuit while achieving even better results.

In future, the function blocks of the automation toolkit will have to be programmably interconnected into a platform because, at the moment, a designer must execute them separately. The integration would result in the complete platform, whose user interactions would be in fact minimal, while still allowing to replace and use its blocks as in a toolkit.

Acknowledgements

This work was supported by The Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPU II) project IT4Innovations excellence in science – LQ1602, the Brno University of Technology under number FIT-S-20-6309 and the JU ECSEL Project SECREDAS (Product Security for Cross Domain Reliable Dependable Automated Systems), Grant agreement No. 783119.

References

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

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

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] G. Kanagaraj, S. Ponnambalam, and N. Jawahar, "A Hybrid Cuckoo Search and Genetic Algorithm for Reliability–Redundancy Allocation Problems," *Computers & Industrial Engineering*, vol. 66, no. 4, pp. 1115–1124, 2013.

[8] Z. Wang, T. Chen, K. Tang, and X. Yao, "A Multi-objective Approach to Redundancy Allocation Problem in Parallel-series Systems," in *2009 IEEE Congress on Evolutionary Computation*. IEEE, 2009, pp. 582–589.

[9] H. Kellerer, U. Pferschy, and D. Pisinger, "The Multiple-choice Knapsack Problem," in *Knapsack Problems*. Springer, 2004, pp. 317–347.

[10] 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.

[11] 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.

[12] J. Lojda, J. Podivinsky, and Z. Kotasek, "Reliability Indicators for Automatic Design and Analysis of Fault-Tolerant FPGA Systems," in *2019 IEEE Latin American Test Symposium (LATS)*, March 2019, pp. 1–4.

[13] B. Korte and J. Vygen, *Combinatorial Optimization: Theory and Algorithms*, ser. Algorithms and Combinatorics. Springer Berlin Heidelberg, 2007.

[14] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*, ser. Wiley-Interscience series in discrete mathematics and optimization. J. Wiley & Sons, 1990.

[15] M. Graphics, "Catapult HLS," https://www.mentor.com/hls-lp/catapult-high-level-synthesis/, 2017, accessed: 2017-07-07.

[16] Xilinx Inc., "ISE Design Suite," https://www.xilinx.com/products/design-tools/ise-design-suite.html, 2017, accessed: 2017-07-07.

[17] N. Dorairaj, E. Shiflet, and M. Goosman, "PlanAhead Software as a Platform for Partial Reconfiguration," vol. 55, no. 84, 2005, pp. 68–71.

Paper F

Automatic Design of Fault-Tolerant Systems for VHDL and SRAM-based FPGAs

LOJDA Jakub, PÁNEK Richard, KOTÁSEK Zdeněk

In: *Proceedings - 2021 24th Euromicro Conference on Digital System Design, DSD 2021*. Palermo: Institute of Electrical and Electronics Engineers, 2021, pp. 549-552. ISBN 978-1-6654-2703-6.

Available at: https://ieeexplore.ieee.org/document/9556374

2021 24th Euromicro Conference on Digital System Design (DSD)

Automatic Design of Fault-Tolerant Systems for VHDL and SRAM-based FPGAs

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 evaluates the possibility of automatic design of fault-tolerant systems from unhardened systems. We present an overview of our toolkit with its three main components: 1) fault-tolerant structures insertion (which we call *helpers*); 2) fault-tolerant structures selection (called *guiders*); and 3) automatic testbed generation, incorporating advanced acceleration techniques to accelerate the test and evaluation. Our approach is targeting complete independence on the HW description language and its abstraction level, however, for our case study, we focus on VHDL in combination with fine-grained n-modular redundancy. In the case study part of this paper, we proved that it is undoubtedly beneficial to select a proper fault tolerance method for each partition separately. Three experimental systems were developed with the usage of our method. Two of them achieved better reliability parameter while even lowering their chip area, compared to static allocation of equivalent fault tolerance technique type. In the case study, we target the best median time to failure, the so-called t50, however, our method is not dependent on this parameter and arbitrary optimization target can be selected, as soon as it is measurable.**

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

I. Introduction

Certain types of electronic systems must be able to maintain high level of reliability. The various reasons for this exist. For instance a control system of a medical equipment must remain stable otherwise a human health would be endangered. Another group of systems cannot be repaired because it is difficult or even impossible to access them. This group includes, for example, satellites, space research probes or space rovers. The design of all of these systems must, therefore, reflect the demand for the high reliability. Generally, reliable systems must be able to perform their task while delivering correct results in prescribed time. One well-known approach to reliable system design is the so-called *Fault Tolerance* (FT) [1]. This approach is based on FT enhancement of the system, while the components are considered naturally unreliable. The architecture of the system is, however, designed and configured in such way, that a failing component does not influence the correctness of produced results nor their timing requirements.

In our research, we focus primarily on FT of commercially available *Field Programmable Gate Arrays* (FPGAs) that are storing their configuration bitstream in the SRAM memory. These are, especially in the area with increased radiation, prone to the so-called *Single Event Upsets* (SEUs). SEUs have potential to flip a configuration bit, thus, changing the implemented design function and possibly the correctness of results. Primarily, we research the possibilities in the FT design automation. Our previous publications [2], [3], [4] presented the possibilities of automatic incorporation of FT structures into algorithms written in a higher programming language, synthesized using the *High-Level Synthesis* (HLS) Design Flow [5]. In this new paper, we present a new method of incorporating FT structures into VHDL language. Description code modification algorithms are strictly separated from allocation algorithms. This is different from the related work FT design automation tools and allows to operate our FT automation toolkit on various description languages of various levels of abstraction while re-using most of the toolkit. As opposed to behavioral-level C++ design in [3], this paper is primarily focused towards designs described in the structural-level VHDL. This research aims to abstract from the description language and bring the FT system design automation in a comprehensive way, which should also be the contribution of our research.

Tools to insert a particular redundancy method exist. Some of them are available only commercially, such as the Xilinx TMRTool [6], which works as a part of the synthesis process, during which it modifies the synthesized design. Another tool is the *BYU-LANL TMR Tool* (BL-TMR) [7], which is not strictly commercial as the TMRTool. The tool targeting Verilog, called TMRG [8], works on the description-code level. It focuses on *Triple Modular Redundancy* (TMR) exclusively.

Approaches to solve the reliability allocation problem can also be found in literature. For example, the genetic algorithm was used for this purpose in [9] and [10], where the use of *Non-dominated Sorting Genetic Algorithm II* (NSGA-II) was used to find a number of promising solutions. The authors of paper [11] present a combination of previously mentioned BL-TMR insertion tool with design space exploration, while targeting various optimization goals.

After a design is hardened, it must be properly tested to ensure its compliance with its specification. In the papers [12], [13], techniques of fault injection into a real FPGA board are shown. There is no need to modify the original design, which is an important advantage. The paper [13] presents the platform called FLIPPER. This platform utilizes two FPGAs, one running the *Design Under Test* (DUT) and the other acting as a controller. The paper [14] presents evaluation platform, which was previously developed in our research group. It runs on a PC and evaluates data captured from an FPGA. This platform is, however, more suitable for the final testing, not for the massively accelerated evaluations, that are necessary in the process of FT system design automation.

This paper is organized as follows: Section II shows the principles of our FT design automation toolkit and its main concepts. The experiments setup and results are presented in Section III. Finally, the Section IV concludes the paper.

II. Fault-Tolerant Design Automation

The following section presents our FT design automation. Our approach is based on the traditional flow, which incorporates manual iteration-oriented improvement of the system

978-1-6654-2703-6/21/$31.00 ©2021 IEEE
DOI 10.1109/DSD53832.2021.00088

while addressing its weakest points. In the automated flow, there is a description of the original system and the target specification available at the beginning. The specification might include, for example, the percentage of critical bits of FPGA bitstream or a *Time to Failure* (TTF). At first, the system must be partitioned. At the moment, partitions are created based on instances of VHDL entities. For the description language, the so-called *helpers* are built. These allow to incorporate FT into a partition of the system. Subsequently, the so-called *guider* must select the most appropriate FT technique for each partition, following the reliability specifications. The last part of the automated flow is testing. This part is crucial according to our previous experiences. The testing and parameters measurement are usually performed in high quantities, making it very time-consuming part of the design flow. The context of the traditional and the automated flow can be observed in Figure 1.

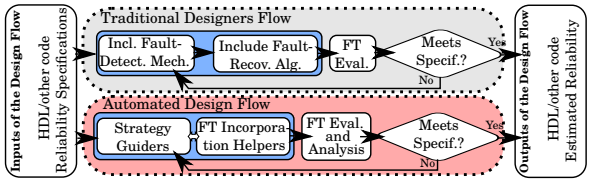

Figure 1: Traditional and Automatic Flows for FT System Design.

A. The Helpers: Fault Tolerance Incorporation

In this paper, we use the newly created helpers for VHDL, which allow us to harden specific entity instances. We consider these as partitions, in the sense of the previously established terminology. Special code comments must be written around the instantiation, which instruct the helpers to make the specific modification. Our VHDL helpers are based on a group of generic templates, which simplifies the addition of new architectures. These are, however, limited by the encapsulation of the entity instance, as these are currently considered as black boxes. The group of templates is supplemented by additional procedures that search for necessary data and use this data to fill a generic template. At first, the VHDL helper divides the original VHDL file into code-block tokens delimited by the special code comments and identifies the tokens (e.g. instantiation block, *don't care* block, etc.). After that, the instances marked for modification are selected and the whole VHDL project is searched for basic pieces of the source description code. These include, for example, entity declarations, signals, etc. These are then parsed to obtain additional information, such as signal directions, bit widths etc. for filling the generic template. Also the clock signal name is detected, in order to route this signal to an optional auxiliary component, such as a scrubbing unit, in the template. After the template is filled, the previous instantiation is modified to refer to this newly filled template. The modification flow is displayed in Figure 2.

B. The Guiders: Fault Tolerance Strategy

It is important to have a strategy to select proper FT techniques for each partition, while meeting the given constraints (e.g. chip area). Such strategy, in our toolkit, is called the *guider*. The guider basically solves the allocation of redundancy techniques. It selects the appropriate FT techniques for the partitions, in order to strengthen FT of the system, while considering one or more constraints. Unachievable constraints cause the design process to stop without a candidate solution.

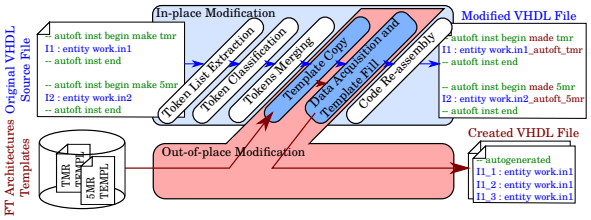

Figure 2: Simplified Code Example with VHDL Modification Flow, Automated using VHDL Helpers.

In our toolkit, we utilize a form of the so-called *Multiple-Choice Knapsack Problem* (MCKP) [15] solver in place of the main guider. The MCKP is a specific variant of the general *Knapsack Problem* (KP). The KP is one of the so-called combinatorial optimization problems [16], the target of which is to maximize the value of items put into a hypothetical knapsack of a given load capacity. The MCKP variant constrains the items that are put into the knapsack. The items are divided to classes and from each class, exactly one item must be selected. As can be seen, the solution to this problem is convertible to our problem of FT technique selection: we have classes of different implementations for each partition. Each implementation has different value (i.e. benefit in the form of increased FT) and different weight (i.e. chip area, power demand, etc.).

C. Fault Tolerance Evaluation

Testing and evaluation of a component or a system is performed relatively often during the design flow. This makes it the most time-consuming part of the complete FT design flow. For this reason, we developed our *Fault Tolerance Estimation* (FT-EST) framework [4], in which we stressed its acceleration possibilities to expedite the evaluation. A test design consists of a test controller and the tested units. We call this complete formation a testbed.

The generated testbed has a fixed structure, although the components are very configurable. The main part of a testbed includes the so-called *Input Generation Unit*, which generates the so-called stimuli for the testing. These can be streams of data (e.g. generated using a counter or a *Linear Feedback Shift Register* (LFSR) unit) or transactions of data. The outputs of tested units are compared against the golden (i.e. reference) unit and the differences are captured. The running tested unit is paused through clock-gating. Fault Injector [17] artificially and permanently changes utilized bits of *Look-Up Tables* (LUTs) in specified times, based on the required fault intensity. Detailed description of our FT-EST testbed generator can be obtained from our previous publication [4].

III. The Case Study and Experimental Results

In the following section, a case study utilizing our FT system design automation toolkit will be presented, based on hardening of an artificially constructed system. Also the parameters of the resulting systems will be discussed alongside with the design flow.

A. Toolkit Setup

In our experiments, we prepared the helpers to include the TMR and *5-Modular Redundancy* (5-MR) techniques. We also utilize the MCKP guider. We focus on the minimization

of the median time to failure, also called the *t50* parameter. This parameter defines the time of 50% probability that the system is still fully functioning. The t50 quantification is more useful for our measurement, as it tends to remove extreme values, opposed to the classical *Mean Time To Failure* (MTTF), which utilizes the mathematical average. Also, the MTTF (i.e. the period from system start, for which the fault masking is possible) tends to lower with the added modular redundancy for longer mission times [18]. We chose to precisely evaluate each partition in advance and then estimate the resulting system parameters inside the MCKP solver. After the solver finishes, the best matched system is then evaluated precisely. We use the FT-EST framework to generate our testbeds. As the test stimuli generator, we use various bit-width variants of LFSRs utilizing corresponding maximal-length polynomials to produce a pseudo-random sequence of all the possible combinations. The fault model includes permanent faults of utilized bits of LUTs. Faults are injected into the precisely selected part of the FPGA configuration bitstream. Their intensity is derived from this bitstream size of the tested design, based on the fault injection intensity unit – *injection/s/bit*. To measure the results, testbeds for each partition and each system are synthesized using the Xilinx *Integrated Synthesis Environment* (ISE) 14.7 and prepared using the Xilinx PlanAhead 14.7. Testbeds are run on the ML506 board [19] with the Virtex 5 technology.

B. Toolkit Input: Benchmark System

We prepared a benchmark system composed of four hypothetical partitions: 1) addition, 2) constant addition, 3) *Cyclic Redundancy Check on 8 bits* (CRC-8) computation; and 4) number of high bits detection. Connection of these components including their input and output bit widths can be observed in Figure 3. The system was described in VHDL.

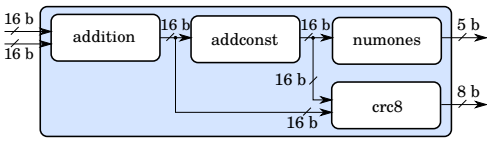

Figure 3: Benchmark System Structure.

C. The Helpers: Variants Generation

With the usage of helpers, we created two hardened variants for each partition of the system. The overview of the partitions, including their real measured parameters, can be seen in Table I. As can be observed, each partition has a different size. After the application of each technique, the FPGA synthesis surely optimized the larger partitions better, as the sizes of hardened partitions do not correspond to the theoretically predicted overheads (i.e. more than triple for TMR and quintuple for 5-MR techniques). The so-called majority voters, which are an integral part for both the TMR and 5-MR architectures, were both included in the timing analysis and subject to fault injection (as other parts of the complete component or system). As can be seen, the t50 parameter improved for each partition except the addconst TMR version, for which it was nearly 20% worse, compared to the simplex t50. This might be caused by the internal structure of the implementation or the nature of the computation itself. The addconst holds the added constant in its implementation. This, if hit, results in a logically functioning design. The results are, however, computed from a different constant values, rendering them incorrect. As can be seen from the results, the effectiveness varies among the partitions, thus supporting the need to methodically select the proper FT method for each partition.

TABLE I: System Partitions with Their Size and Reliability Parameters under Fault Injection Intensity of $2e{-}5\,\mathrm{inj/s/bit}$

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 box plot chart displayed in Figure 4 illustrates the scatter on the measured values for each partition. As can be seen, the benefit of an FT technique is very fluctuating among different circuit types. Also, the simplex minimum time to failure (i.e. the worst measured case) is always better, compared to the TMR and 5-MR versions. This means that the dispersion rates of hardened partitions (at least towards minimum values) are higher. Also, for the crc8 and numones partitions, the middle 50% interquartile range is concentrated nearer the median, indicating lower variability of these results. This indicates that the TMR and 5-MR work better on these partitions. The addconst was the only component visibly deviating in efficiency of FT techniques, specifically for TMR. As can be seen, the 5-MR version has a slightly better median value, although the difference is nearly negligible. Nonetheless, for the 5-MR version of this partition, the variability of the middle 50% is also smaller, similarly but not so obvious as for the numones and crc8 partitions.

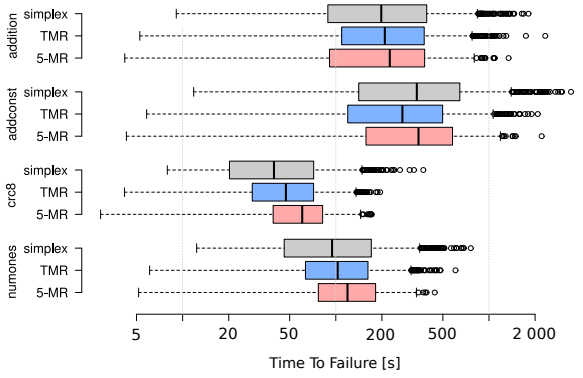

Figure 4: Box Plot Chart of Time to Failure for Each Partition and Their Hardened Variants.

D. The Guiders: Automatic Composition of Systems

Three systems were automatically composed using our methods. Methods were configured to minimize the t50 parameter while not exceeding a given chip area. These area limits were based on the bitstream area that was subject to the fault injection. This was 20 000, 25 000 and 30 000 bits. The overview of synthesized systems, including their parameters, can be seen in Table II. We also created two additional

homogeneous reference systems, each of which is utilizing one type of FT technique applied to each partition.

TABLE II: Automatically and Manually Composed Systems (as a Reference) with Reliability Parameters under Fault Injection Intensity of $2e{-}5\,\mathrm{inj/s/bit}$

System Name	FT Techniques				Bitstream Area [b]	t50 [ms]
	addition	addconst	crc8	numones		
auto_20000	simplex	simplex	5-MR	simplex	18 624	43 198
auto_25000	simplex	simplex	5-MR	TMR	22 400	49 935
auto_30000	simplex	simplex	5-MR	5-MR	25 856	49 675
ref_simplex	simplex	simplex	simplex	simplex	9 152	23 559
ref_TMR	TMR	TMR	TMR	TMR	24 704	42 173
ref_5-MR	5-MR	5-MR	5-MR	5-MR	37 376	55 900

As can be observed, the guider based on the MCKP solver targeted the mostly failure-prone partitions: the crc8 and the numones. Incorporation of FT techniques into the remaining two partitions was evaluated as not sufficiently effective. The crc8 partition was the most error-prone, and thus the highest hardening was allocated for this partition in all the three cases. The smallest automatically composed system occupied approximately $18\,\mathrm{kbits}$, that is only 75.34% size of the reference system size for which the TMR was manually assigned to each partition. Despite this, the automatically composed system shows slightly better t50 parameter than for the manually created reference, thus, saving circa 25% of area. The second automatically composed system is still by 9.33% smaller than the manually created TMR one, yet its t50 is more than $7\,\mathrm{s}$ longer. For the last automatically created system, the t50 is nearly equivalent to the previous, second one. The size of the third system is, however, larger. This wrong choice of partitions by the MCKP solver is apparently caused by imprecise estimation of system t50 from the components t50 times, thus, confusing to solver to choose sub-optimal selection of FT techniques. Nevertheless, this third system is still more than 30% smaller compared to the manually created 5-MR system and its t50 is only by 11% worse.

IV. Conclusions

This paper presents a novel approach to FT system design, which is able to work in various abstraction levels with various language description formats. We implemented our solution in the form of a toolkit with each part of the toolkit specializing on a different task of the FT system design automation. New template-based approach to helpers for incorporating FT techniques into VHDL was presented alongside with the usage of MCKP solver as the guider for the redundancy allocation. Our automatic testbed generation framework was also briefly described. We modified it to monitor, detect and report the time of the first failure observation. The experimental evaluation and illustration of our approach was presented in the Section III. In this section the experimentation is performed on our artificial benchmark circuit. During our experiments, we proved that it is undoubtedly beneficial to select FT method for each partition separately. Three automatically generated versions of the experimental system were developed with the usage of our method. Two of them achieved better reliability parameter while even lowering their chip area, compared to static allocation of equivalent FT technique type.

Acknowledgements

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

References

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

[2] 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.

[3] J. Lojda, J. Podivinsky, O. Cekan, R. Panek, M. Krcma, and Z. Kotasek, "Automatic Design of Reliable Systems Based on the Multiple-choice Knapsack Problem," in *2020 23rd International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, 2020, pp. 1–4.

[4] 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.

[5] M. Fingeroff, *High-level synthesis blue book*. Xlibris Corporation, 2010.

[6] 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.

[7] "Byu edif tools homepage," http://reliability.ee.byu.edu/edif/, 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. Kanagaraj, S. Ponnambalam, and N. Jawahar, "A Hybrid Cuckoo Search and Genetic Algorithm for Reliability–Redundancy Allocation Problems," *Computers & Industrial Engineering*, vol. 66, no. 4, pp. 1115–1124, 2013.

[10] Z. Wang, T. Chen, K. Tang, and X. Yao, "A Multi-objective Approach to Redundancy Allocation Problem in Parallel-series Systems," in *2009 IEEE Congress on Evolutionary Computation*. IEEE, 2009, pp. 582–589.

[11] J. Anwer, M. Platzner, and S. Meisner, "FPGA Redundancy Configurations: An Automated Design Space Exploration," in *2014 IEEE International Parallel Distributed Processing Symposium Workshops*, 2014, pp. 275–280.

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

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

[14] 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.

[15] H. Kellerer, U. Pferschy, and D. Pisinger, "The Multiple-choice Knapsack Problem," in *Knapsack Problems*. Springer, 2004, pp. 317–347.

[16] B. Korte and J. Vygen, *Combinatorial Optimization: Theory and Algorithms*, ser. Algorithms and Combinatorics. Springer Berlin Heidelberg, 2007.

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

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

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

Paper G

Automatically-Designed Fault-Tolerant Systems: Failed Partitions Recovery

LOJDA Jakub, PÁNEK Richard, KOTÁSEK Zdeněk

In: *2021 IEEE East-West Design and Test Symposium, EWDTS 2021 - Proceedings.* Batumi: Institute of Electrical and Electronics Engineers, 2021, pp. 26-33. ISBN 978-1-6654-4503-0.

Available at: https://ieeexplore.ieee.org/document/9580996

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-

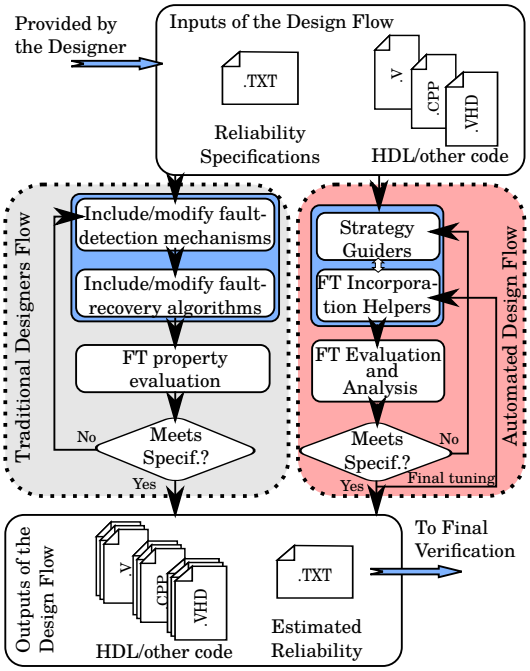

Figure 1: FT System Design with the Original and Automated Design Flows.

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.

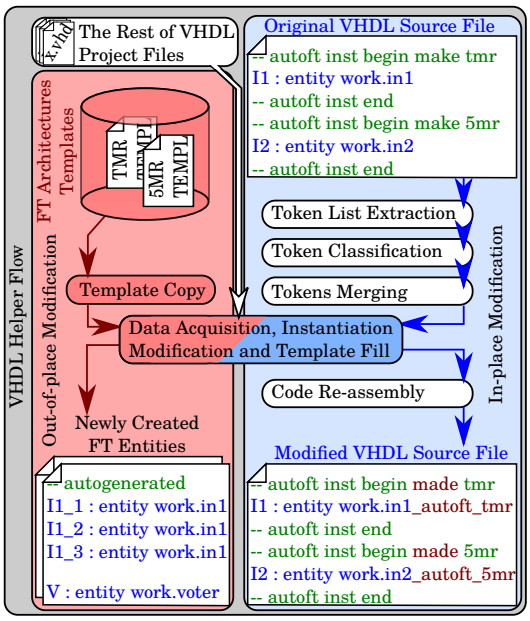

Figure 2: VHDL Helpers Code Modification Flow, Including a Simplified Code Example of Usage.

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.

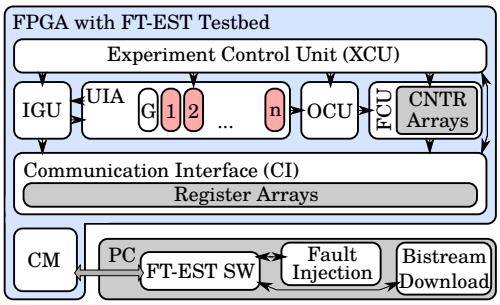

Figure 3: Structure of Automatically Generated Testbed Architecture, Red Parts are Subject to Fault Injection.

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.

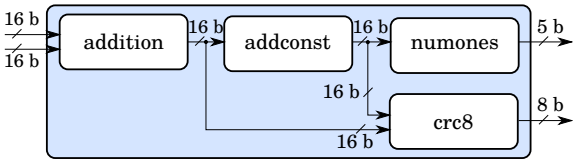

Figure 4: Diagram of the Benchmark System.

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.

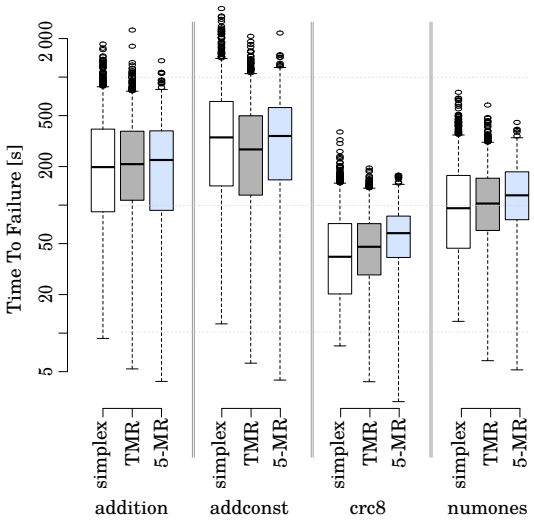

Figure 5: Time to Failure for Each Partition Version on a Box Plot Chart [6].

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

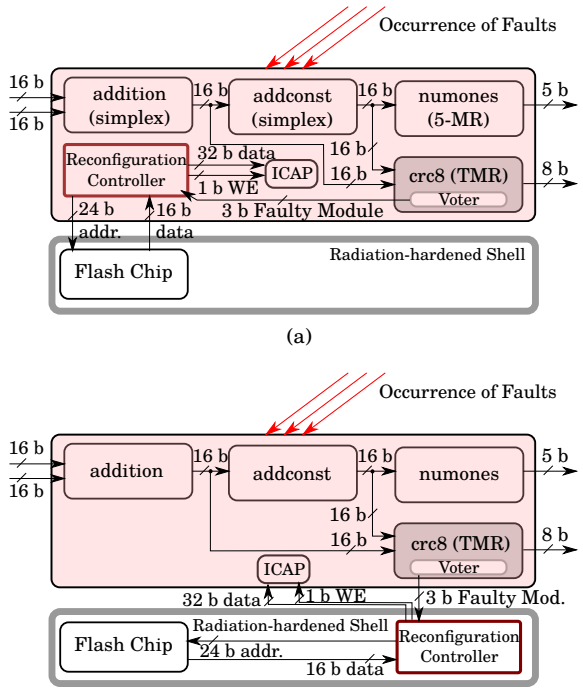

Figure 6: Recovered Systems with PDR controller (a) Inside the System on the FPGA; and (b) Outside the FPGA.

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.

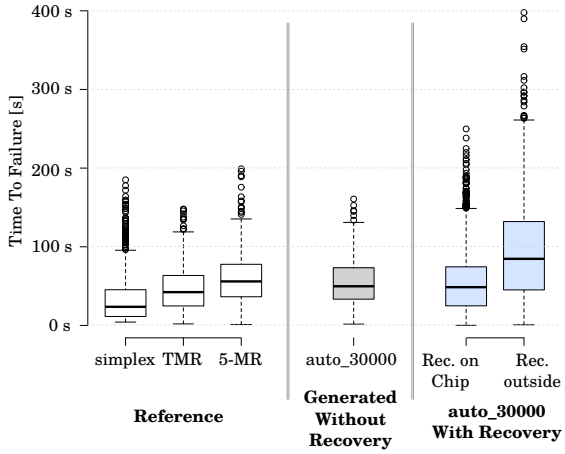

Figure 7: Time to Failure for Each of the Systems (i.e. Reference, Auto-generated [6] and Auto-generated with PDR Controller) on a Box Plot Chart.

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.

Paper H

Automated Design and Usage of the Fault-Tolerant Dynamic Partial Reconfiguration Controller for FPGAs

LOJDA Jakub, PÁNEK Richard, SEKANINA Lukáš, KOTÁSEK Zdeněk

In: *Microelectronics Reliability, vol. 2023, no. 144, pp. 1-16.* ISSN 0026-2714.

Available at: https://doi.org/10.1016/j.microrel.2023.114976

Microelectronics Reliability 144 (2023) 114976

Contents lists available at ScienceDirect

Microelectronics Reliability

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

Automated design and usage of the Fault-Tolerant dynamic partial reconfiguration controller for FPGAs☆

Jakub Lojda *, Richard Panek, Lukas Sekanina, Zdenek Kotasek

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

ARTICLE INFO

Keywords:
Fault-Tolerant system design
Electronic design automation
Dynamic partial reconfiguration
Redundancy allocation and insertion
FPGA
VHDL

ABSTRACT

This article presents a new design automation method for Fault-Tolerant (FT) systems implemented on dynamically reconfigurable Field Programmable Gate Arrays (FPGAs). The method aims at minimizing the human interactions needed to incorporate FT mechanisms into an existing system. It starts with a source code of an original unhardened circuit. It continues by automated manipulation of the source code, algorithmic strategic selection of suitable FT techniques, design space exploration of candidate FT implementations, and selection of the resulting implementation. The method also includes efficient evaluation of achieved FT parameters performed on the target HW. As a novel approach working on the level of HW description languages is employed, the code modification is separated, which differentiates our method from others. The case study utilizing this method targets the design of an experimental FT dynamic partial reconfiguration controller for an FPGA. This controller is helpful for the restoration of faulty components due to a single-event upset on an FPGA. We used the method to generate a set of Pareto-optimal controllers concerning the design's Mean Time to Failure (MTTF) parameter, power consumption, and size. Then, the FT controller is connected to several benchmark circuits, and the reliability parameters are evaluated at the entire system level. Our results show that by replacing the standard reconfigurable controller with our automatically-designed FT controller for one specific benchmark, the design size increased by 20.1%, and MTTF increased by 11.7%. However, the efficiency is highly dependent on the target system size, MTTF, and circuit functionality. We also estimate that a complex system defined by half a million configuration bits would improve MTTF by more than 50%.

1. Introduction

The ever-growing level of integration allows the implementation of increasingly complicated systems. However, this complexity makes designing highly-reliable systems a significant challenge. For this reason, we propose a new method and a toolkit that automates the transformation of unreliable *Field Programmable Gate Array* (FPGA) designs into hardened ones. We cover the complete process of the design. The proposed method combines (1) manipulation of the source code description; (2) strategic selection of the available manipulation techniques to increase the resilience against faults after the described design is synthesized into the HW; (3) thorough autonomous testing on the target platform HW and measurement of the critical metrics; (4) interpretation of the measured data and selection of optimal solutions. Our method combines these approaches to generate hardened systems by modifying unhardened systems while minimizing user interactions.

Our research has the following contributions.

(1) We target the complete process of *Fault Tolerance* (FT) design automation, starting with the unhardened system and providing multiple optimized hardened versions (in case multiple criteria are optimized) or one hardened version (in case a single criterion is optimized). This includes the testing methods and estimation of final parameters.

(2) We target toolkit architecture independent of the description language and its level of abstraction. In fact, most of the methods from the literature are usable within our approach by implementing them as toolkit components.

(3) Our testing approaches stress the speed of the test. This is important as testing is the most time-consuming part of the design.

(4) We support a single-objective and multi-objective design space exploration optimizing criteria such as power consumption, MTTF, and design size.

☆ This article is dedicated to the memory of Zdenek Kotasek, an outstanding scientist who passed away in 2022 and co-supervised most of this research.

* Corresponding author.
E-mail addresses: ilojda@fit.vut.cz (J. Lojda), ipanek@fit.vut.cz (R. Panek), sekanina@fit.vut.cz (L. Sekanina).
URLs: https://www.fit.vut.cz/person/ilojda/ (J. Lojda), https://www.fit.vut.cz/person/ipanek/ (R. Panek), https://www.fit.vut.cz/person/sekanina/ (L. Sekanina), https://www.fit.vut.cz/person/kotasek/ (Z. Kotasek).

https://doi.org/10.1016/j.microrel.2023.114976
Received 3 June 2022; Received in revised form 27 January 2023; Accepted 28 March 2023

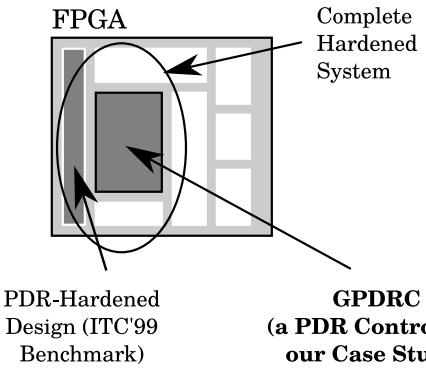

Fig. 1. The relation of the GPDRC to the hardened system.

(5) To demonstrate the automated design flow usage, we present a case study on a particular design, the *Generic Partial Dynamic Reconfiguration Controller* (GPDRC) [1]. The role of GPDRC is to ensure that dynamic partial reconfiguration of the FPGA is performed as requested. In the case study, this FPGA component is hardened against faults. Then, the FT controller is connected to several benchmark circuits, and the reliability parameters are evaluated at the entire system level.

The rest of this article is structured as follows. Section 2 describes the concepts of our research. Section 3 deals with the research background and related work. Our FT design automation flow is described in Section 4. Section 5 presents our case study circuit, the so-called reconfiguration controller. A set of Pareto-optimal FT controller implementations is generated. In Section 6, one selected representative reconfiguration controller is evaluated in detail when connected with benchmark circuits on the FPGA. Section 7 concludes the article and presents ideas for future research.

2. Objectives and research concept

Our research goal is the automation of the FT design process for FPGAs supporting *Dynamic Partial Reconfiguration* (DPR). To demonstrate the automated design flow usage, we present a case study on a GPDRC circuit, which serves as a starting point. The relation of the GPDRC to the hardened system is displayed in Fig. 1.

This GPDRC is hardened and later utilized in experimental measurements on a real FPGA HW. To achieve this goal, we propose a design flow and practically implement tasks of this flow with tools from our toolkit. An overview of the experimental part presented in this article is displayed in Fig. 2.

2.1. The proposed approach

Based on the research in the field of FPGA FT systems, we propose the design approach that will be explained in detail in Section 4 and practically used in a case study in Section 5. The design automation can be divided into three main steps, which include *generation*, *design space construction and exploration*, and *FT design build*. The inputs of the automated design flow are:

(I) the original unhardened description of an FPGA circuit (e.g., in VHDL);
(II) a set of monitored parameters, for example, *Mean Time to Failure* (MTTF), design size and power consumption;
(III) a set of FT mechanisms that will be used to reach the monitored parameters, for example, *Triple Modular Redundancy* (TMR).

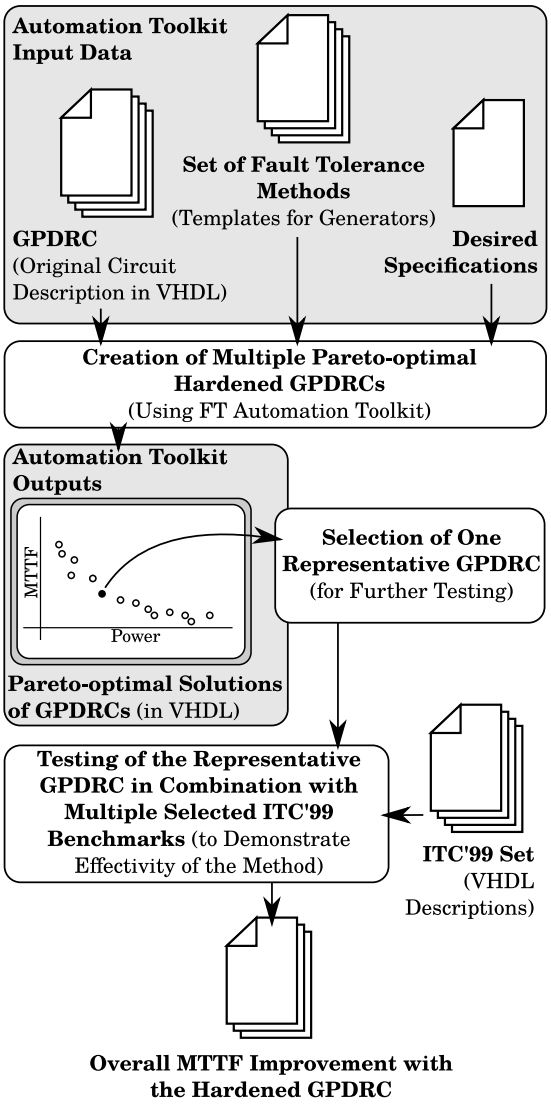

Fig. 2. Overview of the experimental part of this article.

During the **generation**, a set of temporary circuits is generated using given FT methods. These circuits are generated using scripts that perform the modifications in commonly used *Hardware Description Languages* (HDLs) and insert FT mechanisms into particular parts of the code. This set of circuits is evaluated in terms of desired parameters, and the obtained data serve as a base for the next step. Measurement of FT properties is performed using our unique tool for the automated generation of testbeds, which runs autonomously on a target FPGA and utilizes techniques for the massive acceleration of the evaluation.

In the **design space construction and exploration**, the data obtained in the previous step are used to construct the design space of possible solutions completely in SW. This is critical, as no further parameter measurement is needed, significantly accelerating the design space exploration and, thus, the design automation process.

Through the **FT design build**, the selected (i.e., the most suitable) design configurations are built using the scripts for FT mechanism insertion (which were used in the generation step). Also, corresponding parameters of the selected solutions are provided to the designer.

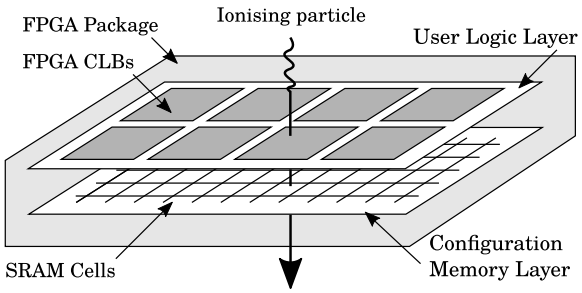

Fig. 3. The double-layer FPGA model through which an ionizing particle is passing [7].

2.2. Assumptions

In this article, we target consumer-grade FPGAs with SRAM configuration memory. We utilize fault masking in combination with system component restore, which is achieved through the DPR function of an FPGA.

The principles of our design automation flow are limited to neither the target FPGA technology nor the description format. However, the current implementation in the form of a toolkit targets only systems and algorithmic logic described in VHDL and C++ languages. This is because, so far, we have implemented algorithms for language modifications for these two languages. The FPGA used for our case study is based on the Xilinx Virtex-5 technology. This is because we utilize specific tools previously implemented in our lab to inject artificial faults and detect utilized parts of the Xilinx Virtex-5 configuration bitstream [2]. However, the design automation method is designed to be independent of the target technology. It is versatile enough, as it mostly works at the language level. A potential reuse for another FPGA technology is straightforward as it only requires replacing fault injection components and FPGA communication protocols according to the target FPGA family.

3. Background and related work

Many computer systems are data-oriented, meaning they process a significant amount of data. Scientific applications can serve as an example. FPGAs can help to accelerate such data processing. For example, the *National Aeronautics and Space Administration* (NASA) Perseverance rover [3] carries multiple Xilinx FPGAs on board [4]. These accelerate the data processing throughout the mission of the Perseverance rover, which is searching for signs of life on Mars.

3.1. Field Programmable Gate Arrays

FPGAs are integrated circuits that can be user-configured after production with the help of a *Hardware Description Language* (HDL) [5]. FPGA's configuration is prepared in the form of the so-called *configuration bitstream*, which is transferred to FPGA and stored in the so-called *configuration memory* [6]. The FPGA can therefore be viewed in a double-layer model [7]. The first layer represents the *user logic*, which consists of *Configurable Logic Blocks* (CLBs) and a programmable interconnection network. The central functional units are *Look-Up Tables* (LUTs); others include registers, multiplexers, and other more complex elements, including, for example, memory blocks. The configuration bitstream determines the specific setting of this logic, i.e., the current FPGA configuration. The second layer is the *configuration memory* – this memory stores the configuration of the user logic. One commonly used configuration memory type in commercially-available FPGAs is the SRAM. A graphical representation of this double-layer model can be seen in Fig. 3.

The undeniable advantage of FPGAs is the ability to upgrade or even completely re-program their function, even for already deployed systems. This process of changing the FPGA configuration is called a *reconfiguration*. There are two approaches to reconfiguration; the first is *external*, i.e., controlled from the outside, and the second is *internal*, i.e., held directly from within the FPGA logic. Advanced applications might also utilize the so-called *Dynamic Partial Reconfiguration* (DPR) of the FPGA [8]. *Partial* reconfiguration means it is possible to change the configuration of only a specific part of the circuit on the FPGA. This has the advantage of a faster change in functionality over reconfiguring the entire FPGA. This is due to transferring a shorter bitstream to the configuration memory. *Dynamic* reconfiguration provides the advantage of changing the FPGA configuration while the implemented circuit runs continuously. Thanks to DPR, it is possible to change only a particular part of the implemented circuit while the rest continues to perform the required function. In addition, the whole process can be controlled directly from the logic of the FPGA.

FPGAs have various interfaces for accessing the configuration memory. It is possible to both upload the configuration and read it back through them. If the FPGA allows it, some are also usable for DPR. Finally, the FPGA has a special internal interface accessible directly from its logic, which can only be used for DPR. For example, for Xilinx Virtex-5 FPGAs, this is *Internal Configuration Access Port* (ICAP) [9]. Through it, it is possible to transmit a bitstream at a data width of up to 32 bits and with a clock frequency of 100 MHz. Therefore, the maximum transfer rate is 3.2 Gbit/s.

3.2. Single-Event Effects on FPGAs

An ionizing charged particle passing through the FPGA poses a risk on both layers of the FPGA's double-layer model in the form of the so-called *Single-Event Effect* (SEE). Firstly, it has the potential of the so-called *Single-Event Transient* (SET) [10]. In the case of the SET phenomenon, a *transient error* appears on signal levels, which connect the components implemented as the user logic. The errors can also be propagated to the primary outputs and produce incorrect results. If the structure of the user logic does not explicitly store the incorrect result, such an erroneous state disappears over time. Nevertheless, the structure of the user logic remains intact.

Secondly, an ionizing particle passing through the configuration memory layer creates a risk of a *Single-Event Upset* (SEU) [11]. After an SEU phenomenon, a configuration bit in the FPGA configuration memory is flipped. This is a severe problem, as a change in the configuration memory instantly changes the structure of the user logic. The design implemented in the FPGA is immediately and permanently altered, and its function thus differs. FPGA configuration bits whose flipping is observable by inconsistencies on the primary outputs of the design are called *critical* or *sensitive*.

3.3. Mitigation of SETs

In general, the reliability of an FPGA system can be increased by two main approaches. (1) In *Fault Avoidance* (FA) [12], the effort to avoid failures of an FPGA is put into selecting the FPGA type. For example, specific FPGA product ranges can withstand increased radiation by utilizing special radiation-hardened packaging. The number of types of such FPGAs is, however, limited. Moreover, these hardened types' performance (e.g., the capacity or a maximal clock speed) is usually worse than the state-of-the-art commercially available types. (2) On the opposite side, the so-called *Fault Tolerance* (FT) [13] approach accepts the risks of possible failures but mitigates the consequences of failing components through masking. This is usually achieved by inserting redundant structures, thus creating a form of a backup. The most popular method of FT is the *Triple Modular Redundancy* (TMR) [14] and its variations, which comprise copying the system multiple times and adding a majority voter to detect the majority result (i.e., the correct result). Graphical representations of the FA and FT approaches are shown in Fig. 4(a) and (b), respectively.

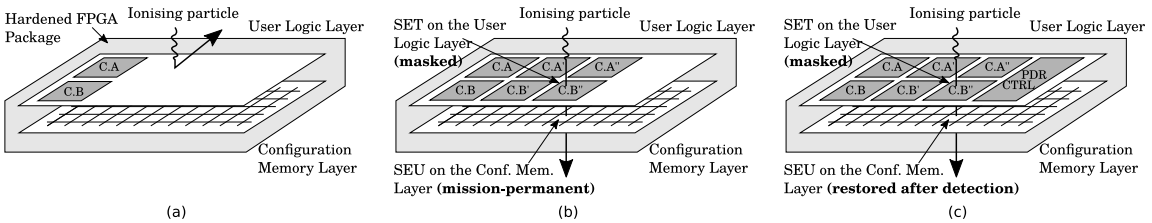

Fig. 4. Graphical illustration of (a) FA; (b) FT without restore; and (c) FT with a restore with the usage of the RC.

3.4. Mitigation of SEUs

The number of SEUs maskable by the FT approach is limited. This is because SEUs accumulate in the system configuration memory, which is unfavorable for systems with an extended mission time. The redundancy needed on the user logic layer to maintain a certain level of FT rises dramatically as the mission time increases. The so-called *restore* (i.e., repair) can solve this problem if a failing component is detectable [15]. During the restoration, the configuration memory layer below the failing component is returned to its original state, thus effectively restarting the component and eliminating the SEU effects accumulated in the user logic layer. The DPR property of the FPGA is used to implement the restore mechanism. The process of restoring function on an FPGA is known as memory *scrubbing* [16]. In the simplest case, the configuration memory is only cyclically overwritten by the correct configuration, i.e., the so-called *golden bitstream*. More sophisticated approaches use configuration memory readback. In addition, each configuration must be provided with a correction code. Thanks to the code calculation, it is possible to determine whether there is a fault in the given section of the configuration memory and reconstruct only the affected parts by reconfiguration. A critical component is the *Reconfiguration Controller* (RC), which must provide and manage everything related to FPGA reconfiguration. The RC can be added to the system or the FPGA itself. A non-volatile memory type, such as flash memory storage, significantly less sensitive to SEUs, is utilized to store the golden bitstreams. It is also possible to use the unused part of the configuration memory of the FPGA itself to keep the golden bitstream and to reallocate them from one part of the memory to another for reconfiguration [17]. Fig. 4(c) shows a graphical representation of the restore principle.

3.5. Testing

The correct functioning of the implemented measures must be tested during a project's design and final verification stages. For FPGAs, SEUs can be artificially inserted into the configuration memory. This is because waiting for a fault to appear naturally is impossible for time reasons. Also, the fault manifestation process must be controlled to ensure a certain testing quality. Such an artificial insertion of faults is called the *Fault Injection* (FI). While one or multiple faults are injected, the system's output data is viewed to monitor the faults' impacts. Such an approach is called the *functional verification*. Many strategies to inject faults exist, for example, *single* fault injection, which is used to detect the criticality of a bit; *periodic* injection with a given interval, etc. Selecting the correct injection strategy to measure the desired parameter is essential. For example, the radiation effects are simulated through periodic injection into a randomized bitstream position. The rate at which particles achieve the FPGA's external surface is given in the *radiation flux* unit, which is in particles/cm^2/s [18]. Already from this unit is evident that the intervals between randomized injections must be based on the tested chip area. This is because a more extensive system covers a larger area of the FPGA. Thus particle penetration and SEU manifestation have a higher probability for a more extensive system, considering the equivalent environment, i.e., equivalent flux, device, and exposure time (test length). For this reason, we introduced the unit inj/s/bit [19], which represents the number of injections per time per size in bits. This takes account of the area (i.e., the configuration size in bits), resulting in a smaller injection interval for larger designs and vice versa.

Testing is thoroughly discussed in the literature. The authors of [20] show how to observe and modify signals in the HW-implemented design. This is performed through the *Joint Test Action Group* (JTAG) interface. In [21], various fault models are supported. Additional support gates must be, however, incorporated into the tested design. A simulated approach is presented in [22,23]. In [24], the simulation approach is extended with fault modeling. The authors of [25] utilize the RapidSmith software [26] and demonstrate their approach in [27]. Another approach, in which faults are injected directly by a component on an FPGA itself, is presented in [28]. The authors of [29] show fault injection into a real FPGA HW. Later, in paper [30], the authors present their platform called FLIPPER. It is a two-FPGA platform, with one FPGA running the tested design and the other FPGA controlling the testing procedure. A platform for testing fault-tolerance properties developed in our group is presented in paper [31]. However, this platform is suitable for the final testing of resilience against faults. It runs partially on a PC, and the tested design runs on a target HW. Nonetheless, to allow automated testing, we developed the *FT Estimation Tool* (FT-EST) testbed generator framework [32], which is intended for use in our design automation toolkit. It allows massively-accelerated evaluations on multiple FPGAs simultaneously to complete the test in a reasonable time.

3.6. Computer-aided design of FT systems

The growing complexity of FT system design created pressure to, at least partially, automate the process. Automation brings new benefits to the field of FT system design. At first, the automated process is easier to check on potential algorithmic errors. Secondly, it eliminates human factor errors, which potentially prolong the design process before the verification process detects them. Naturally, designing a complex system to be FT is challenging. The automated design of FT systems is composed of two main components: (1) manipulation of the description in such a way as to enhance resilience against faults; (2) proper allocation of the available manipulation techniques.

The literature suggests two main approaches. (1) One possibility is manipulating the synthesis tool to produce hardened designs. The TLegUp [33] can serve as an example. It bases on the *High-Level Synthesis* (HLS) tool LegUP [34]. With TLegUp, the user can generate TMR designs directly from their description in the C language. Another example is the so-called Xilinx TMRTool [35], which modifies the design during its synthesis. The advantage of such an approach is the direct access of the automation algorithms to the internal representation of the circuit. For example, the optimization algorithms are aware of the added redundancy, which can be straightforwardly kept in the internal representation. However, suppose a designer intends to add a new FT technique or modify the current techniques. In that case, they immediately find the fundamental requirement to access the synthesis tool's source code and modify it to accomplish their intention. This can be solved by exclusively focusing on open-source equipment, yet still, modifying such tools remains a great challenge.

(2) As opposed to the previous, this approach enhances a design's resiliency by modifying the design's source description. This method looks more straightforward as it strictly separates the functionality and focuses exclusively on source manipulation and technique allocation. Also, one significant advantage of such an approach is the possibility of achieving (at least partly) independence of the language and abstraction level. For example, the allocation algorithm can be reused, while the manipulation and synthesis components are replaced in the automated FT design toolkit. One crucial requirement is to precisely manipulate the descriptions in a given language, which is (as we found in our previous research) rather challenging for particular languages (e.g., VHDL) and might also be straightforward for other languages (e.g., C++ with HLS). One representative of such an approach is the TMRG [36]. It modifies descriptions in the Verilog language. An alternative is the RASP-TMR [37], which also focuses on TMR and Verilog. Another representative is the so-called *BYU-LANL TMR Tool* (BL-TMR) [38], which is also non-commercial and modifies structures in the *Electronic Design Interchange Format* (EDIF). In [39], an extension that combines this approach with allocation algorithms was presented. Our previously published VHDL generators [40] also allow us to modify VHDL entity instantiations and interconnections. The VHDL generators are based on a templating system, allowing a straightforward extension of available FT methods. However, these tools only partially represent the source description modification approach. This is because these source-code manipulators do not provide allocation functionality.

The allocation is an essential part of the FT design. It analyzes the data (e.g., system structure, measurements, etc.). And after that, each component (or subsystem) is provided with the best setup of code manipulation to achieve the desired level of reliability. The allocation of the available resiliency-enhancement techniques is formulated as an optimization problem in the literature. In [41], the *Improved Surrogate Constraint* method and its computational speed were presented. In [42], the so-called *Particle Swarm Optimization* is proposed to solve the problem. Another method, the so-called *Neighborhood Search Metaheuristic* method, is proposed in [43]. In [44,45], a genetic algorithm is used. Specifically, the so-called *Non-dominated Sorting Genetic Algorithm II* (NSGA-II) helps to find promising solutions. The authors of [46] present the *Penalty-guided Artificial Bee Colony Algorithm*.

However, none of these has explored a way to completely automate the FT design process from its beginning to the end, including the automated measurements on a target HW platform during the design. This is a highly complex task and requires large-scale implementation of various components. The research presented in this article, however, tries its best to address this challenge comprehensively; for this, it uses the approach of modified description. Table 1 presents and compares the available methods to the one shown in this article from their subject elaboration point of view. The first six columns are self-explanatory. The table's seventh column (Search Alg.) focuses on whether the approach provides the exact solution or whether it is a heuristic method, providing a solution that satisfies the requirements but might not offer the optimal combinations of parameters. The eighth column indicates whether the approach is single- or multi-objective. The upper part of Table 1 targets the codes manipulation tools, while the center part targets redundancy allocation methods. The last part deals with approaches combining multiple techniques to achieve the automated design of FT systems. The generalized approach presented in this article is characterized in the last row of Table 1. Also, please note that none of the related research analyzes the resulting designs on the target HW platform while being easily usable for different platforms and languages.

4. Automated design of FT systems

Our design flow starts with the specifications of input data. Two fundamental inputs are requested: (1) the *unhardened* (i.e., original or simplex) system which is to be hardened; (2) the information on which parameters are to be minimized or maximized. This can be, for example, the MTTF, power consumption, design size, etc. The designer is then provided with multiple Pareto-optimal system configurations if multiple criteria are specified or a single solution in the case of a single criterion.

The automated design flow starts with the unhardened version of a system. This might be a previously designed system from a project stage in which the necessity of FT was not considered. It might also be a newly developed system for which the manual incorporation of FT would be complex. At the beginning of the design, the designer must also specify the design objectives (e.g., FT, size, power consumption). These parameters can be arbitrarily selected based on the project's requirements and needs. So far in our research, we have been optimizing the number of critical bits [48], MTTF, the median time to failure [49] and, in this article, also the power consumption of the resulting design.

After the specification is provided, the automated design can start. At first, in a process called *Generation*, our automated flow generates solutions that serve for data acquisition. In each solution, precisely one component is hardened using one FT method. In addition, one reference solution, which corresponds to the original description, is added. These data-acquisition solutions are analyzed to measure their parameters. The level of FT can be measured directly on the targeting HW using our *FT Estimation Tool* (FT-EST) [32]. External tools are also supported to provide the necessary information about the candidate solution, such as estimating the power consumption.

In the following process, called *Design Space Construction and Exploration*, data obtained in the previous phase are used to interpolate the parameters of the other possible configurations. This approach significantly accelerates the design-space exploration, as no further measurements are required. This is because the previously measured data serve as a base for the calculation. Full design space exploration can be performed. Heuristic approaches of single- or multi-objective design space exploration can also be utilized for large systems, saving CPU time. The efficient exploration methods are essential, as the entire design space contains many sub-optimal solutions.

One or multiple suitable configurations are stored after discovering them during the design-space exploration. These configurations are then used to build the actual FT system in the process called *FT Design Build*. This uses the same generator scripts as the first process (i.e., the Generation process). During this stage, any possible final one-time modifications to the design can also be performed. Our previous research used this stage to incorporate system repair [49]. Afterward, the optimal FT solutions are handed over to the designer. The graphical overview of this flow from the beginning to end, where the final systems are obtained, can be seen in Fig. 5.

4.1. FT design generation

The automated design flow depends on the possibility of modifying the source description. In our approach, we stress the re-usability of the flow components among different HDLs and various levels of abstraction. For this purpose, components of the flow called *generators* were developed. The generators modify the source code description to produce FT design after the modified code is synthesized using traditional synthesis methods (e.g., the HDL synthesis) or even new methods, such as the *High-Level Synthesis* (HLS). So far, we have been using generators to modify C++ on the algorithm level [47] and VHDL on the *Register-Transfer Level* (RTL) [40]. The nature of these modifications is based on the abstraction level of the description language. An essential part of the generator function is the insertion of FT structures. The set of generators for a given description language might cover various FT architectures (e.g., TMR, duplex, 5-MR). Another fundamental property of the generators is targeting the modification towards only a part of the source code. This allows the construction of design space to select later the FT methods based on a fine-grained approach.

Table 1
Comparison of the related research; the first part targets code manipulation methods; the second part focuses on redundancy allocation approaches; and the third part describes approaches that target design automation, including the approach presented in this article, which is on the last row.

	Approach name	Author	Focus	FT method	Language		Search	Objectives
					Input	Output	Alg.	number
Code Manipulation	BL-TMR [38]	BYU[a], LANL[b]	Design Manipulation	TMR	EDIF	EDIF	–	–
	TMRG [36]	CERN	Design Manipulation	TMR	Verilog	Verilog	–	–
	RASP-TMR [37]	A. R. Khatri et al.	Design Manipulation	TMR	Verilog	Verilog	–	–
	VHDL Generators [40]	J. Lojda et al.	Design Manipulation	Not Limited/ templates	VHDL	VHDL	–	–
	Redundant Data Types [47]	J. Lojda et al.	Design Manipulation	Not Limited/ templates	C++/HLS	C++/HLS	–	–
Redundancy Allocation	Improved Surrogate Constraint [41]	J. Onishi et al.	Reliability Allocation	–	–	–	Exact	Multiple
	Multi-objective Particle Swarm Optimization [42]	K. Khalili-Damghani et al.	Reliability Allocation	–	–	–	Meta-heuristic	Multiple
	Variable Neighborhood Search Algorithm [43]	Y. C. Liang et al.	Reliability Allocation	–	–	–	Meta-heuristic	Single
	Cuckoo Search Genetic Algorithm [44]	G. Kanagaraj et al.	Reliability Allocation	–	–	–	Meta-heuristic	Multiple
	Non-dominated Sorting Genetic Algorithm II [45]	Z. Wang et al.	Reliability Allocation	–	–	–	Meta-heuristic	Multiple
	Artificial Bee Colony Algorithm [46]	W. C. Yeh et al.	Reliability Allocation	–	–	–	Meta-heuristic	Multiple
	Multiple-choice Knapsack Problem Solver [48]	J. Lojda et al.	Reliability Allocation	–	–	–	Exact	Single
Design Automation	TMRTool [35]	Xilinx/AMD	Part of Synthesis Flow	TMR	NGC	NGC	–	–
	TLegUp [33]	G. Lee et al.	Part of Synthesis Flow	TMR	C	Verilog	–	–
	BL-TMR+Space Explo-ration+Multiple Voters [39]	J. Anwer et al.	Design Mani-pulation+Reliability Allocation	TMR (Var. Voter Types)	EDIF	Bitstream	Exact	Multiple
	Fault Tolerant Design Automation Toolkit (this work)	**J. Lojda et al.**	**Design Mani-pulation+Reliability Allocation**	**Not Limited/ templates**	**Not Limited/ modular**	**Not Limited/ modular**	**Exact**	**Multiple**

[a]Brigham Young University.
[b]Los Alamos National Laboratory.

In this article, we practically utilize the generators for VHDL. These allow targeting the specific FT architecture towards a VHDL entity. After the generators are executed, the VHDL code is examined, and the selected parts are modified. Special code comments are used to instruct a specific FT method. These surround the entity instantiation, and during one execution of the generators, one or multiple modifications can be performed. A template system is used to specify FT methods. A graphical representation of the VHDL generators' flow can be seen in Fig. 6.

In the first stage, the data-acquisition systems are generated using our generators. In these systems, always precisely one component c is hardened using one of the FT methods m for each component $c \in C$ and each method $m \in M$, while C is the set of system's components and M is the set of available FT methods. This results in $|C| \cdot |M|$ data-acquisition systems.

4.2. Design space construction and exploration

Another essential component of the design flow is the *Design Space Construction and Exploration*. Candidate solutions are explored and evaluated in this part. Again, this component is strictly isolated from the rest of the design flow components. This is useful to achieve the reusability of the design space construction and exploration for multiple description languages on various abstraction levels. The global strategy of the design flow can be easily modified by replacing exploration methods. It is helpful to utilize a method with a well-studied and known mathematical base. This makes it possible to estimate the resulting quality of the conforming solution in advance.

In practice, a very high number of measurements (assessing reliability parameters) of the candidate solutions would be required. Even for smaller systems, this might lead to a number of measurements in the magnitude of thousands, significantly prolonging the design

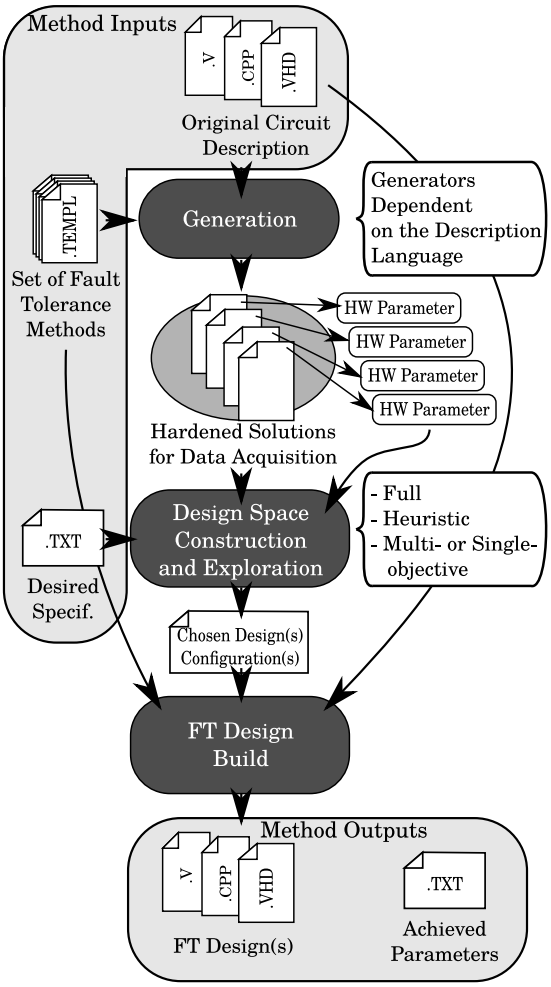

Fig. 5. The graphical overview of the automated flow for FT systems design.

time. For this reason, attention should be paid to estimating most of the parameters of candidate solutions from the measurement of the data-acquisition solutions. This is, however, not possible for specific parameters, such as the achievable maximal frequency of a design, which is not easily interpolated. Nonetheless, the maximal frequency can be read from the synthesis process. However, this prolongs the design process, as it requires executing the synthesis tool for each candidate solution. Nevertheless, this applies to designs for which frequency is essential and is part of the design space construction.

Please note that in the following text, notation $sys\_a_1b_1$ denotes the system in which the component a is hardened by the method 1 and the component b is also hardened by the method 1, while the rest of the components of the system are left simplex. In the previous stage (i.e., the generation stage), the data-acquisition systems were prepared. These data-acquisition systems can be denoted as sys_c_m, $\forall c \in C$, $\forall m \in M$. In addition to them, the *simplex* solution is also added. The parameters of these solutions are measured. Let us define a process of MTTF calculation for each of the remaining candidate solutions whose MTTF actually was not measured directly. The following procedure is used.

(a) The resulting *failure rates* of the simplex and data-acquisition systems, $\lambda_{simplex}$ and $\lambda_{sys\_c_m}$, $\forall c \in C$, $\forall m \in M$, are calculated using Eq. (1).

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

(b) Let λ_{sys} of a system be a sum of λ_c, $\forall c \in C_{sys}$. This is formally described in Eq. (2).

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

It is evident that using this relation, the difference $\delta\lambda_{c_m}$ can be calculated for each pair of an FT method and a system component, which expresses the difference in the failure rate after the application of the FT method m to the component c. Please note that these combinations are represented by the previously-measured data-acquisition systems, including the simplex system, which serves as a reference point to calculate the differences.

(c) These $\delta\lambda_{c_m}$ values can be subsequently utilized to calculate the λ_{sys} for systems with multiple components hardened by various FT methods. This is done by summing the corresponding differences $\delta\lambda_{c_m}$ and the $\lambda_{simplex}$. And after that, the resulting failure rate value can be converted using Eq. (1) back to the MTTF of the candidate system.

The overview of such calculation for the MTTF can be seen in Eq. (3). Such a calculation is more straightforward for power consumption, as the overall system consumption is simply a sum of the components' power requirements.

$$\lambda_{simplex} = \frac{1}{MTTF_{simplex}} \tag{3a}$$

$$\lambda_{sys\_a_1} = \frac{1}{MTTF_{sys\_a_1}}$$

$$\vdots$$

$$\delta\lambda_{a_1} = \lambda_{sys\_a_1} - \lambda_{simplex} \tag{3b}$$

$$\delta\lambda_{b_1} = \lambda_{sys\_b_1} - \lambda_{simplex}$$

$$\vdots$$

$$\lambda_{sys\_a_1b_1} = \lambda_{simplex} + \delta\lambda_{a_1} + \delta\lambda_{b_1} \tag{3c}$$

$$MTTF_{sys\_a_1b_1} = \frac{1}{\lambda_{sys\_a_1b_1}}$$

$$\vdots$$

Such an approach significantly accelerates the design-space exploration, especially if MTTF is measured. For example, in the case of full design space exploration, 1 FT method and 10-component system, $2^{10} = 1024$ solutions would have to be measured. However, only 11 measurements would have to be performed with the in-SW approximation. This represents only 1% of the required time to measure because the design space construction and exploration can be considered negligible compared to the measurement time. This is essential as it allows the design automation to base on accurate measured data. This would be impossible without such an approach, as measuring all the candidate solutions would take months or years. With this approach, the measurement time lies in the magnitude of weeks. And these techniques make the automated design possible.

It is, however, essential to analyze the precision of such an estimation. Such error analysis of uncertainty comes from the estimation process. And, because it is necessary to know the limits of the MTTF estimation, we focus on the *standard error* (a.k.a. *standard deviation of the mean*) as the common way to enumerate error in measurements [50]. The standard deviation is denoted by σ, while the standard deviation of the mean is denoted by $\bar{\sigma}$. Their relation can be seen in Eq. (4), where N equals the number of measurements.

$$\bar{\sigma} = \frac{\sigma}{\sqrt{N}} \tag{4}$$

During the error analysis, the calculation of the estimated MTTF is followed. For each operation with the MTTF, the corresponding procedure is performed with the standard error, which is measured initially (as well as the MTTF of the data-acquisition systems) and which is finally estimated (as well as the MTTF of the candidate solutions).

(a) In Eq. (3a), the MTTF of the reference simplex system and the data-acquisition systems are used to calculate their failure rate (i.e., λ).

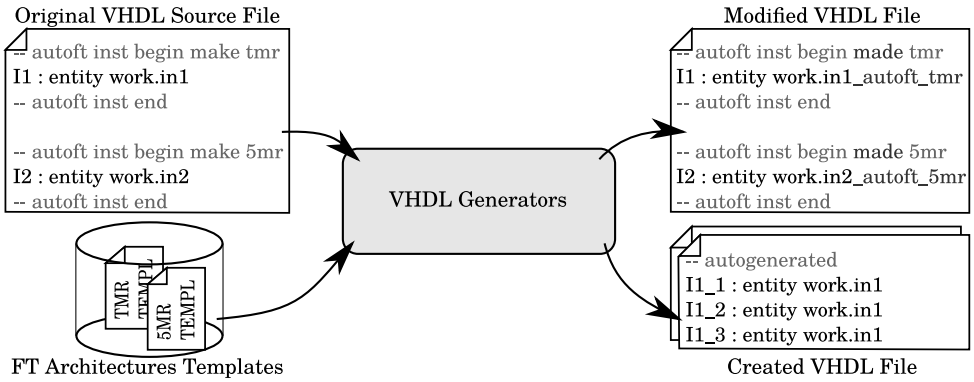

Fig. 6. The creation of one candidate solution using the VHDL generators, including a short example of original and modified VHDL code.

This is done by dividing the value 1 by the MTTF value. According to [50], the standard deviation of a division (for independent measures) equals the root sum of squares of the individual relative uncertainties. We use this equation to calculate the standard deviation of the mean (i.e., standard error). We assume that the $MTTF > 0$ and $\bar{\sigma}_{MTTF} \geq 0$. Moreover, we assume that the deviation of a constant (i.e., 1) equals 0. After the substitution, it can be further simplified into the form shown in the last part of Eq. (5a).

(b) After that, the failure rate of the simplex system is subtracted from the failure rates of the data-acquisition systems as shown in Eq. (3b). According to [50], the standard deviation of a sum (for independent measures) equals the root sum of squares of the individual absolute uncertainties. The deviation for one of the data-acquisition systems can be seen at the end of Eq. (5b).

(c) And finally, the failure rates are summarized, and the value is returned to the MTTF. For this operation, the Eqs. (5a) and (5b) are used again. The result (after simplification) can be seen in Eq. (5c). Because the inputs of these equations include the MTTF value itself, naturally, the expected standard error of the measurement can be calculated only after the input measurement (i.e., data acquisition) is performed.

$$\frac{\sigma_f}{f} = \sqrt{\left(\frac{\sigma_x}{x}\right)^2 + \left(\frac{\sigma_y}{y}\right)^2} \tag{5a}$$

$$\frac{\bar{\sigma}_{\lambda_a}\sqrt{N}}{\frac{1}{MTTF_a}} = \sqrt{\left(\frac{\bar{\sigma}_{MTTF_a}\sqrt{N}}{MTTF_a}\right)^2 + \left(\frac{0}{1}\right)^2}$$

$$\boxed{\bar{\sigma}_{\lambda_a} = \frac{\bar{\sigma}_{MTTF_a}}{{MTTF_a}^2}}$$

$$\sigma_f = \sqrt{(\sigma_x)^2 + (\sigma_y)^2} \tag{5b}$$

$$\dots$$

$$\boxed{\bar{\sigma}_{\delta\lambda_{a1}} = \sqrt{\left(\frac{\bar{\sigma}_{MTTF_{simplex}}}{{MTTF_{simplex}}^2}\right)^2 + \left(\frac{\bar{\sigma}_{MTTF_{a_1}}}{{MTTF_{a_1}}^2}\right)^2}}$$

$$\boxed{\bar{\sigma}_{\lambda_{sys}} = \sqrt{\left(\bar{\sigma}_{\lambda_{simplex}}\right)^2 + \sum_{\forall c \in hardened}\left(\bar{\sigma}_{\delta\lambda_{c1}}\right)^2}} \tag{5c}$$

$$\boxed{\bar{\sigma}_{MTTF_{sys}} = \frac{\bar{\sigma}_{\lambda_{sys}}}{{\lambda_{sys}}^2}}$$

This presented error analysis will be later used in the text to enumerate the actual standard error values of the estimations.

For multi-objective optimization, deciding which conforming solution is the best is impossible. For this purpose, a chart with detected Pareto frontier [51] is plotted. The full design space search or a heuristic method can be utilized in place of the design-space exploration method. For example, in our previous research [48], we used the Multiple-choice Knapsack Problem solver to explore the state space. This approach, however, was limited to only one searched parameter and one constrained parameter.

4.3. Evaluation of design parameters

Although we try to minimize the number of measurements, we still need to measure the actual parameters for a smaller number of systems on the real target hardware. This is to obtain base data for the design space construction. This includes the evaluation of each modified variant. Thus at least $|C| \cdot |M| + 1$ times, where C is the set of components, and M is the set of FT methods implemented in generators (the +1 is for the reference simplex system). Differences (i.e., improvement or deterioration) from the original values can be calculated for each component based on the overall measured values for the whole system.

We developed the FT-EST framework, which aims to generate the so-called *testbeds* automatically. These allow accelerating FT parameters measurement. Such a testbed is configured into the FPGA alongside the tested unit and performs accelerated measurement of the tested unit. Most of the testbed functionality is implemented in HW to maximize the throughput. Our approach stressed the acceleration of the evaluation, which is still very important to keep the measurement time at a reasonable level. Parallel unit evaluation of multiple instances is possible, but depends on the size of the tested unit and the space available on the FPGA. The testbed can measure the number of critical bits of a design. It is also able to measure the MTTF. The structure of the testbed is described in generic VHDL. Modifications and extensions to the measured parameters are straightforward, thanks to the modular construction and the abstract data access layer.

The modular structure of the testbed can be seen in Fig. 7. An essential part of the FT-EST testbed is the *Input Generation*, which generates the testing stimuli for the tested units in the *Unit Instantiation Area*. The clock signal to this area is gated, allowing the computer SW to pause a design clock at a given time. This facilitates the fault injection at the exact time and enables the injection of faults in an arbitrarily short interval (i.e., even shorter interval than the fault injection script run time). Results from the tested units are checked against the reference unit in the *Output Compare* block. Potential mismatches are captured in the *Failure Capture* block. Data from this block are addressable through the *Communication Interface*. So are addressable

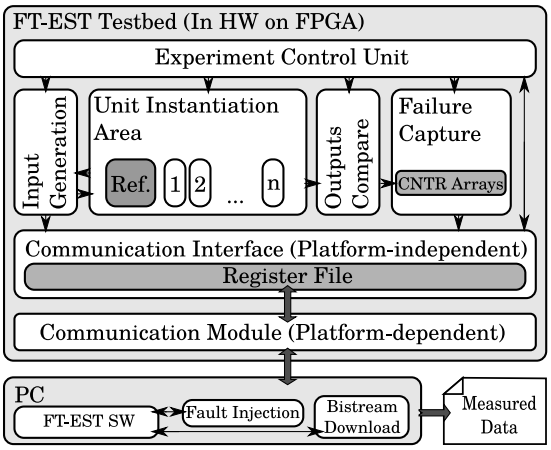

Fig. 7. Block diagram of the testbed, which is generated using the FT-EST tool.

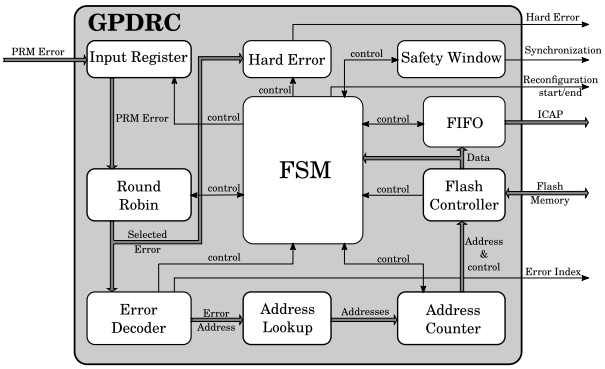

Fig. 8. Block diagram of the GPDRC with interface.

the status registers and command register, which provide control of the experiment process. The platform-specific communication is then isolated in the *Communication Module*, as each manufacturer offers different means of communication. For example, in this article, we use the Xilinx ChipScope Pro *Integrated Controller* (ICON) [52] and the *Virtual Input/Output* (VIO) [53] IP cores for the USB JTAG communication. On the PC, a script starts the autonomous experiment on the FPGA and downloads and stores the measured data into a file. For the artificial creation of faults in the FPGA configuration bitstream, a fault injector that can inject faults at the FPGA run time is used [2]. It was previously developed in our research group. Besides the injection, it also utilizes the RapidSmith software to detect the occupied part of LUTs in the configuration bitstream. It helps achieve higher accuracy by accelerating the test, as the unused configuration bits are ignored. This article uses the FT-EST framework to generate all the FT measuring testbeds. The FT-EST was explicitly configured to measure the MTTF.

For the parameters unrelated to FT, any external tool can be used whose output is accessible by a program (i.e., from a script). In this work, we use the *Xilinx XPower Analyzer* [54] to estimate the power consumption. This is a helpful tool, as it precisely calculates the power consumption based on the accurate 0–1 switch ratio. The analysis for the switch ratio is obtained in advance by simulating the design in the *Xilinx ISE Simulator* (ISim) [55].

5. Case study: Automated design of FT RC

A reconfiguration controller was selected to demonstrate the automated design of its fault-tolerant version. The goal is to design a resilient controller with the following features: as fault-tolerant as possible (i.e., the largest MTTF) and the lowest power consumption. Pareto-optimal solutions are sought using the previously described tool for the automated design of an FT system.

The *Generic Partial Dynamic Reconfiguration Controller* (GPDRC) [1] was chosen for the experiments. The GPDRC we use in the case study is representative of a hard-coded controller that can be placed on the same FPGA with the hardened circuit. However, at the same time, it is prone to SEU. The susceptibility to SEU of the controller itself is summarized in [56]. It was found that the original GPDRC has approximately 8.8% of critical bits. A failed reconfiguration controller can lead to the failure of the whole system. This is due to its access to the internal interface for FPGA configuration. Therefore, in a loss, it may expose incorrect data to this interface, causing unexpected system behavior. Thus, the FT of the controller itself is crucial for the entire resilient system.

5.1. GPDRC implementation analysis

At first, we will analyze the circuit design that is an object of our case study before we use the automated FT design tool. The GPDRC is directly implemented in FPGA logic. Its block diagram is shown in Fig. 8. There are 10 basic functional components of which the controller is composed:

Finite State Machine (FSM) is the fundamental component that controls the operation of the entire controller and, thus, all other components. This control is provided by responding to incoming status signals from other components and setting control signals leading to other components.

Input Register is a component that captures and stores all detected faults for further processing. The detected fault may also correspond to the reconfiguration request of the given module.

Round Robin provides a step-by-step selection of stored module reconfiguration requests so that they are all serviced.

Hard Error component will identify any faults that cannot be repaired. These are so-called permanent (irreparable) configuration memory faults.

Error Decoder converts the selected reconfiguration request to a specific address in the conversion table.

Address Lookup poses a table with addresses on which individual bitstreams are stored in memory with golden bitstreams. In our case, it is an external flash memory.

Address Counter takes care of the sequential issuance of addresses for reading data from memory with the golden bitstream.

Flash Controller provides an interface for external flash memory with golden bitstreams. It, therefore, generates the appropriate signals for reading data from a given memory. This component must always be adapted to the specific HW used.

FIFO is a component that synchronizes data from memory with golden bitstreams and then exposes it to the ICAP. As a result, the reconfiguration is performed, i.e., the relevant configuration is written to the FPGA configuration memory, and the required function is restored.

Safety Window is used to provide demanded time for the modules to be synchronized before reconfiguring the module that was being restored.

The controller interface is as follows. At the input, the controller waits for a reconfiguration request via the *PRM Error* bus. Each request has its signal, so it is possible to receive multiple requests simultaneously. The controller also has two-way communication with flash memory. Finally, there are outputs necessary for the reconfiguration, such as data to ICAP and a signal indicating the end of the current reconfiguration process, i.e., *synchronization*. Then there are statistical outputs. These include identifying whether the fault is permanent, i.e., *Hard Error*. *Reconfiguration start/end* signals inform about the actual reconfiguration process used for recovery. The *Error Index* indicates which component is being restored by reconfiguration.

A summary of the controller's operation based on its interface and components is as follows. The controller waits for the arrival and subsequent selection of the reconfiguration request. Subsequently, the range of addresses at which the relevant golden bitstream is stored is determined based on the selected request. These addresses are gradually issued for the memory, and the applicable data are then read from it. The read bitstream data is used both to store in the FPGA configuration memory and to control the reconfiguration itself. In addition to the configuration itself, the bitstream also provides the control instructions and information about its structure, such as the data size of individual blocks [57].

5.2. Experiment setup

In our experiments, we search for FT GPDRC designs. We stress the chip area, the MTTF, and the GPDRC design's power consumption. Our GPDRCs are targeting the Xilinx Virtex-5 technology [58]. For the design stage and evaluation of the designs, we utilize the ML506 evaluation boards [59] featuring the *XC5VSX50TFF1136* Xilinx FPGA and *JS28F256P30T95* Intel flash memory. This golden bitstream memory, located directly on the evaluation board, is 16 bits wide and 32 MB in size. We use the FT-EST framework to measure the MTTF of designs. A periodic FI strategy is used. We empirically selected the fault intensity to 2×10^{-5} inj/s/bit. We keep the fault intensity constant among the tested designs, making the actual mean time between FIs reflect the design size. For GPDRC testing, the *Input Generation* component of the testbed simulates continuous requests for reconfiguration. The GPDRC under test is not connected to the FPGA's internal reconfiguration port. The interfacing is emulated in an additional component instead. This is to prevent data inconsistencies if a failing GPDRC would damage the contents of the FPGA configuration memory, including the FT-EST testbed core. It also allows testing multiple GPDRCs at once, as the FT-EST provides parallel design evaluation to accelerate the process, while the FPGA's internal reconfiguration port is the only one.

The Xilinx XPower Analyzer estimates the power consumption of particular designs. Our GPDRC is composed of 10 components. The FT insertion generators will modify these parts of the source code. In the experiment, only one generator is used: the TMR. This means that a component in the GPDRC is either in the simplex version (i.e., the original description) or a TMR version (created by the generator). The design space exploration targets the size, MTTF, and power consumption and generates the Pareto frontier of optimal solutions. We generate the Pareto frontier for each parameter pair to visualize them in the two-dimensional space.

5.3. Initial data acquisition

During the initial data acquisition, the influences of generators for each combination generator vs. component must be evaluated. In our case, we evaluate the complete GPDRC, in which one component is hardened using generators. The enhancement of parameters gained by the modification is then compared with the reference original (i.e., unhardened) GPDRC. This way, the gain for each parameter is calculated. Such a procedure is helpful, especially in cases where splitting the design would require preparing multiple different Input Generators, which is time-consuming. Moreover, by keeping the tested component with its surrounding components, we do not risk potential inaccuracy in the measurement. This is 10 components and 1 TMR generator, resulting in 10 data-acquisition systems and their evaluations. Besides these, another 1 evaluation is needed to measure the parameters of the original simplex GPDRC. For reference purposes, we also created and measured one extra design, the so-called *Coarse-Grained TMR* (CGTMR), in which the GPDRC is triplicated as a whole, i.e., on the system level, not on the component level. This also had to be measured. Thus we performed 12 measurements of various GPDRC configurations. The MTTF is a statistical value. A vast number of FI runs must be done for one MTTF measurement to obtain a statistically-accurate value. For our case, we empirically set the number of experiments to equal the number of the tested unit's chosen FI bits.

Table 2 summarizes data-acquisition measurements. The TMR generator efficiency varies among the different components of the GPDRC. However, the size overhead is not always in direct proportion to the MTTF, which can be understood as an indicator of FT (i.e., a higher MTTF represents a higher level of FT). For example, the *1tmr_6*, in which the *lookup_ent* component is hardened, has the size of 20 608 bits and the MTTF is 24 234 ms. For *1tmr_2* the MTTF is 24 929 ms, which is nearly by 3% better, but its size overhead is even yet smaller by 6%. Another highlight is the *1tmr_5*, which achieves the highest gain in MTTF; however, the design size increase is not the highest one compared to the rest of the table.

Similarly, as seen in Table 2, the MTTF is not in direct proportion to the power consumption. Hence, the power consumption is not directly proportional to the size overhead. This can be best observed by comparing the simplex and *1tmr_3* and *1tmr_6* systems, in which *errdec* and *lookup_ent* components are hardened. Despite these components being hardened, their power consumption lowers. We believe that this is because the static power consumption (e.g., generated by the leakage current) is not dependent on the active chip area [60]. The current leaking through "a stack" of transistors depends on the number of transistors that are currently turned off in this stack; this is the so-called stacking effect and is, in general, purposefully used to lower the leakage current [61]. It shows that our measured parameters cannot be trivially derived from each other. This is why a multi-objective search of optimized variants is needed in our case. The reference CGTMR is significantly larger. Yet its MTTF is even smaller than for the other data-acquisition systems. This is because the TMR can lower the MTTF for extensive mission times for specific designs. Thus, for this system, the CGTMR is the worst approach.

Besides the optimized parameters, Table 2 displays the maximum frequency of a design as a representative of the parameter that was not optimized throughout the design process. As can be observed, the frequency remains nearly constant while the components are being hardened using the TMR. Two exceptions are the roundrobin and errdec components. They significantly decrease the maximal frequency after their hardening, which certainly affects the critical path of the design, thus the frequency.

To analyze the precision of the resulting estimations, standard errors are calculated using the relations deduced in Eq. (5a). At first, the chart in Fig. 9 presents the standard errors of the MTTF for each of the data-acquisition systems. As observed on the chart, the standard errors remain in the same magnitude. The primary source of the differences in standard errors are various responses of the components on their hardening, resulting in different scatters of MTTF values.

However, it is more interesting to examine the standard error of the MTTF for complete systems. As is obvious from Eqs. (5a), the MTTF standard error of a system depends on the number of hardened components in the system. To study this, we examine the standard error for MTTF of systems in which the first; the first and second; the first, second, and third, etc., components are hardened. It is 10 components in total. The systems are named $sys_{\mathbf{1-x}}$, in which the x is from the interval $\langle 1; 10 \rangle$. As shown in the chart in Fig. 10, the standard error of

Table 2
Measurements of the complete GPDRC components acquired on the real Xilinx Virtex-5 FPGA as a primary base for the decision-making process of the design space exploration (the 1tmr variants) and a reference (the simplex and CGTMR variants). MTTF was measured using our FT-EST framework, and the power consumption was measured using the Xilinx XPower Analyzer toolkit.

Tested GPDRC	Description	Size [bits]	Fault intensity [inj/s/bit]	Mean time between FIs [ms]	Number of measurements [-]	MTTF (Measured) [ms]	Power Consumption [mW]	Frequency [MHz]
simplex	original GPDRC	**18 688**	2×10^{-5}	2 675.5	18 712	**24 703**	**20.47**	171.880
1tmr_1	input_register in TMR	**19 392**	2×10^{-5}	2 578.4	19 392	**24 760**	**21.76**	171.880
1tmr_2	roundrobin in TMR	**19 328**	2×10^{-5}	2 586.9	19 328	**24 929**	**22.63**	136.221
1tmr_3	errdec in TMR	**19 136**	2×10^{-5}	2 612.9	19 136	**24 599**	**17.06**	136.631
1tmr_4	harderr in TMR	**19 648**	2×10^{-5}	2 544.8	19 648	**24 793**	**21.15**	170.271
1tmr_5	safetyw in TMR	**25 536**	2×10^{-5}	1 958.0	25 536	**26 719**	**20.64**	171.880
1tmr_6	lookup_ent in TMR	**20 608**	2×10^{-5}	2 426.2	20 608	**24 234**	**19.65**	151.768
1tmr_7	adrcnt in TMR	**24 832**	2×10^{-5}	2 013.5	24 832	**26 161**	**21.08**	164.328
1tmr_8	fifo_ent in TMR	**30 784**	2×10^{-5}	1 624.2	30 784	**26 415**	**29.28**	171.880
1tmr_9	fsm in TMR	**38 976**	2×10^{-5}	1 282.8	38 976	**21 181**	**23.56**	165.413
1tmr_10	flash_controller in TMR	**21 632**	2×10^{-5}	2 311.4	21 632	**24 064**	**27.22**	171.880
CGTMR	the whole GPDRC in coarse-grained TMR	**60 672**	2×10^{-5}	824.1	60 676	**21 485**	**35.24**	170.445

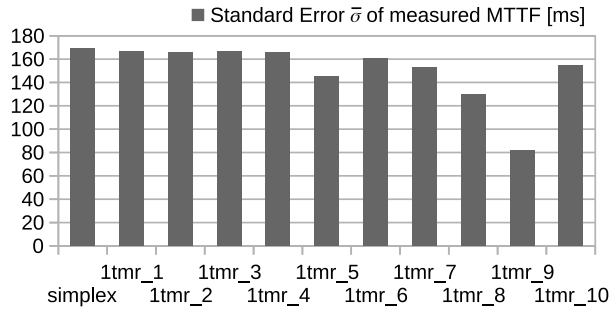

Fig. 9. Calculated standard error of measured MTTF values for the data-acquisition systems.

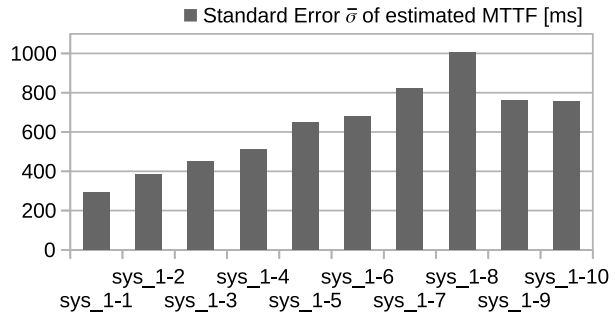

Fig. 10. Calculated standard error of estimated MTTF values for the candidate systems.

the estimated MTTF increases as the number of hardened components grows. But still, for our experiments, the maximal standard error keeps below the 1000 ms, which is a relatively positive fact for the precision of the estimated MTTFs.

5.4. Complete design space exploration

Each of the 10 components is exclusively in either simplex (i.e., original) or hardened TMR state. This results in $2^{10} = 1024$ GPDRC configurations. The overwhelming majority of them are surely sub-optimal. But there are certainly such configurations that are perspective at least by one of the monitored parameters. We monitor the size overhead, MTTF, and power consumption and explore the whole state-space of configured solutions. We save time by calculating these values based on the values obtained during the initial data acquisition. We show each pair of the parameters in two-dimensional charts in Fig. 11. The Pareto-optimal solutions are marked in the chart. These solutions are optimal for a certain application. In the case of multiple criteria, an appropriate solution must be selected later based on the user's preferences (e.g., minimal power consumption or minimal overhead).

As can be observed in Fig. 11, the solutions create clusters on the chart. The granularity is 10 components and 2 possible configurations per component (i.e., the original and TMR). The first combination of parameters (i.e., the MTTF vs. power consumption) reveals 22 Pareto-optimal solutions. These solutions are small or medium, demonstrated by the green and yellow colors. In the second chart (i.e., the MTTF vs. design size), 26 Pareto-optimal solutions were found. In this case, the clusters are more concentrated. None of these solutions belong to the power-intensive solutions, as the absence of the red color suggests. In the third chart (i.e., the power consumption vs. design size), only three Pareto-optimal solutions were found. This is because the power consumption is significantly more dependent on the design size. In this case, the clusters are not so evident. The solution that will be selected later in this article for further experimentation and the reference solution *simplex* are denoted in the charts. In the third chart, only the reference solution (i.e., *simplex*) is Pareto-optimal from the further investigated systems. However, the selected solution's MTTF is excellent, as the red color suggests higher values (i.e., better MTTF).

6. Usage of the FT GPDRC

For directly unmanaged missions, it is essential to have control systems that are highly independent. Such behavior is also required for the fault correction subsystem itself. Therefore, the RC must take care of everything related to mitigating faults. In addition, it must be resilient on its own. We have chosen to detect errors at the application level, where at the same time, these errors caused by configuration memory faults are masked by TMR until the correct function is restored.

Our second experiment aims at the practical evaluation of the previously-designed GPDRC in an experimental system. We assume that for each system, a certain GPDRC exists that corresponds to the characteristics of the hardened system and the required mission duration. This makes the effectiveness of each GPDRC highly dependent on the system that the GPDRC restores. This time, the power consumption will not be part of the evaluation. This is because the power consumption measurement in the simulated environment does not work on the bitstream level, making it impossible to simulate reconfiguration using the ICAP. Nonetheless, the power consumption was considered during the search in Section 5.

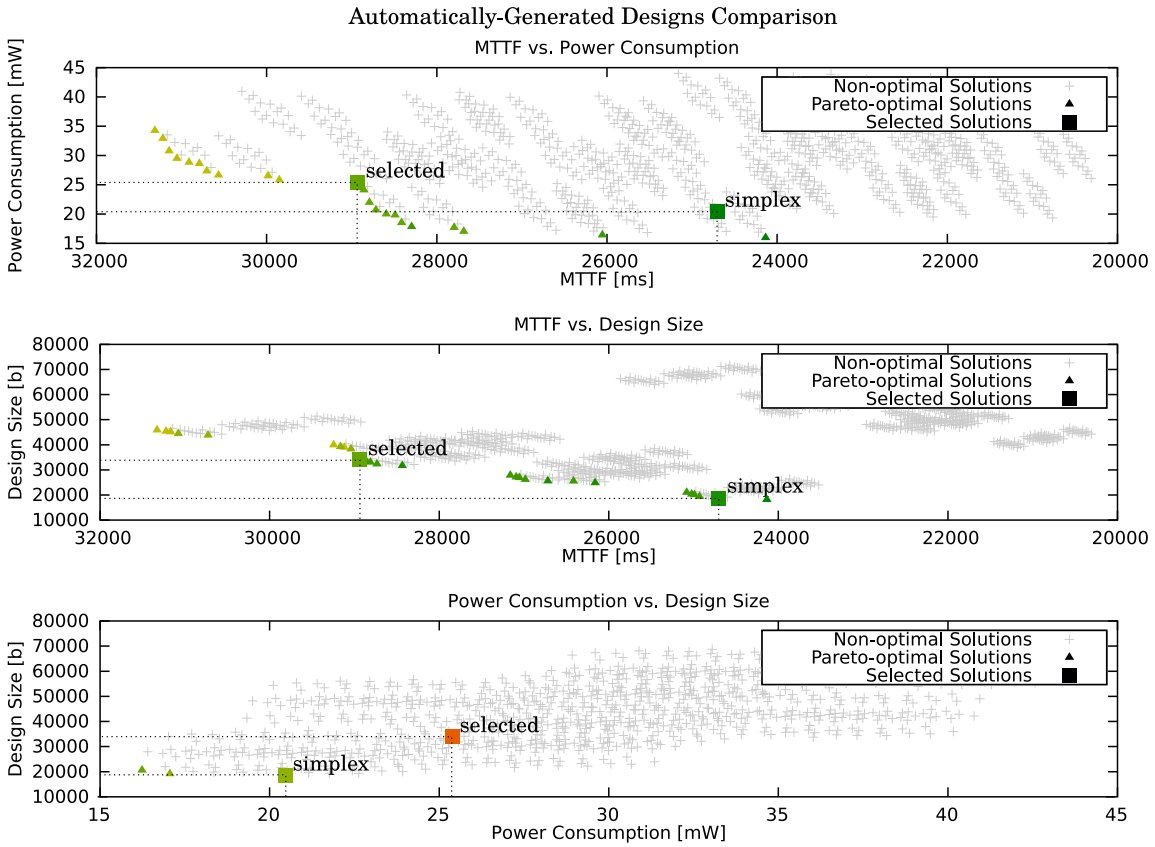

Fig. 11. The design space of solutions obtained by the exploration method with indicated Pareto-optimal solutions per each combination of the target parameters. The color of Pareto-optimal and selected solutions always reflects the value of the third parameter (i.e., design size for the first chart), varying from red for the largest through yellow to green for the lowest value. Please note that for the power consumption and the design size, the lower is better; for the MTTF, the higher is better.

6.1. System preparation for the reconfiguration

Based on FT and other requirements, such as non-interruption of the system even during repair, reconfiguration cannot be used without modifying the system itself. In our case, we expect an uninterrupted system run, i.e., the system will be fully functional even in a fault and subsequent recovery. TMR is used to mask the fault until the system recovers. The application of TMR to the system is the first necessary modification. The modified majority voter is used. Compared to the standard one, it must be able to determine which of the modules provides a different value. This information is brought to the GPDRC input, which must also be added to the FPGA area. This scenario was tested in our previous research [62], where other issues related to increasing system FT through reconfiguration were addressed.

Secondly, it is necessary to ensure that the modules are synchronous; therefore, it is possible to determine the majority of their outputs. The easiest way is to use the *reset* signal of individual modules, ensuring that they are reset to the same default state. However, it is not always possible to use this option. If the modules need to maintain their internal state, then a reset is unsatisfactory, and synchronization is required. When synchronizing, the current internal state of the undamaged modules must be loaded into the newly repaired module. Therefore, it is necessary to know the system's functionality thoroughly to identify what needs to be synchronized. It is also required to ensure that the modules can provide these values to others and initialize themselves with current values. Either the modules themselves must provide this functionality, or another component must be added to ensure the process. In our present scenario, all modules are reset to synchronize them after restoring any of them using reconfiguration.

6.2. The selected GPDRC

In this experiment, we primarily target the MTTF vs. power consumption tradeoffs (i.e., the first chart in Fig. 11). For this reason, we selected one representative GPDRC from the Pareto-optimal solutions in this set. This selected solution has the power consumption of 25.38 mW, MTTF of 29 021 ms and bitstream size of 33 984 bits. Besides this, we added the original simplex and the CGTMR versions of GPDRCs as a reference. The overview of these selected GPDRCs can be seen in Table 3. The last column of the table displays the maximum frequency at which the design can be executed as a representative of a parameter that is not optimized throughout the automated FT design. The selected solution is nearly 25% slower than the original input design. Surprisingly, the CGTMR version keeps its speed. This means the critical path is not manifesting in the coarse-grained reference approach. However, comparing the design size and the MTTF of the CGTMR version to the others, the CGTMR is significantly below the optimal solutions. The results would be, however, very different if the optimizations were targeted towards the frequency, for example. Please note that the estimated MTTF used during the automated design search is very close to the actual measured MTTF after the system was synthesized. The difference from the real measured value is below 0.3%, which is within the magnitude of the precision expected from the previously presented error analysis.

6.3. Benchmarking systems

To demonstrate the function of the selected GPDRCs, we need to utilize benchmarking systems that our selected GPDRCs will harden.

Table 3
Parameters and configuration of components for the selected and simplex GPDRCs and comparison to the CGTMR GPDRC.

GPDRC variant	Configuration of Components										Size [bits]	MTTF (Estimated) [ms]	MTTF (Measured) [ms]	Power Consump. [mW]	Frequency [MHz]
	input_register	roundrobin	errdec	harderr	safetyw	lookup_ent	adrcnt	fifo_ent	fsm	flash_controller					
simplex	sim.	sim.	sim.	sim.	sim.	sim.	sim.	sim.	sim.	sim.	18 688	–	**24 703**	**20.47**	171.880
selected	**TMR**	**TMR**	sim.	**TMR**	**TMR**	sim.	**TMR**	sim.	sim.	sim.	33 984	**28 934**	**29 021**	**25.38**	130.016
CGTMR (ref.)	The whole GPDRC in coarse-grained TMR										60 672	–	**21 485**	**35.24**	170.445

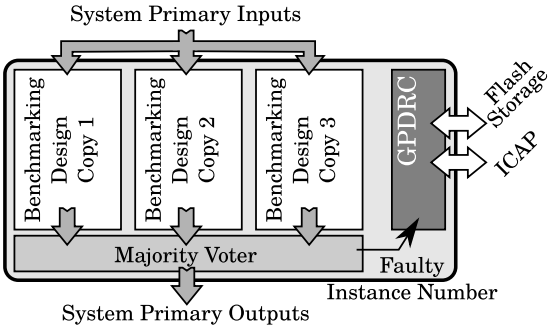

Fig. 12. Block diagram of the benchmarking system, to which the GPDRC controller was added.

For this, we selected three benchmarks from the *ITC'99* benchmarking design set [63]. We took advantage of the fact that the ITC'99 set can also be obtained in the VHDL format. The first is the *b01*, which implements a finite-state machine that compares serial data flows. With the FI size of only 704 configuration bits, this is representative of a relatively small design. The second selected benchmark is the *b05*, which implements a circuit elaborating memory contents. It spreads over 12 736 bits. The third selection is the *b12*, which implements a simple game, *the guess a sequence* for one player. This benchmark represents a more extensive system, as the size of 16 384 bits suggests.

6.4. Experiment setup

To implement a restoration mechanism into the system, we must first incorporate redundant blocks into the system. This redundancy serves as a masking mechanism for the failing block and thus serves as a backup until the failing block is restored. In our case, we also utilize the TMR method. The primary outputs of the triplicated blocks are compared in the voter component, and the failing block is detected based on the results of the majority function. The GPDRC is then requested to reconfigure the failing block, effectively restoring its function. Most of the designs, however, hold an internal state. This is why it is necessary to synchronize all the blocks before the system is considered in fault-less condition again. In our case, we utilize the reset signal to keep the benchmarking designs in a synchronized state. The already-described VHDL generator tool produces the TMR version of the benchmarking system. The overview of the system with the GPDRC incorporated can be seen in Fig. 12. After the systems are created, again, we use the FT-EST framework to generate testbeds and set the fault intensity to 2×10^{-5} inj/s/bit. The ML506 evaluation boards are again used to execute the tests on the real HW. The testbed features the tested system that incorporates the GPDRC controller. The FT-EST's golden system does not include any DPR restore mechanism, serving only for reference purposes. The GPDRC-hardened system is under fault injection, which outputs data that is then compared to the original unhardened reference system with the clock-precise timing.

6.5. Results

For our measurement of the MTTF value, we selected the number of experiments to equal one-tenth of the number of the selected FI bits (i.e., area). The results of the measured MTTF parameter for the selected benchmarks with selected GPDRCs can be seen in Table 4 in column *MTTF (Testbed)*. The estimated value for an orbital trajectory in the height of 555.6 km with 2.5-inch aluminum shielding is shown in column *MTTF (Orbit)*. These values are estimated based on the information presented in [64]. The design size reflects the size of the GPDRC, as observed. So does the mean time between FI, which depends on the total system size. The results are divided into three main parts, depending on the benchmarking system. For the *b01* benchmarking system with the reference simplex GPDRC, the MTTF is 371 s. Using the *selected* GPDRC, the design size increased by 71.5%, while the MTTF increased by 4.5%. It is important to note that the GPDRC is nearly 23 times larger than the *b01* benchmark system. For the *b05* benchmark, the hardened GPDRC also proves to be beneficial. The 23.7% overhead in design size increases MTTF by 4.4%. As can be seen, the best enhancement in MTTF was achieved for the *b12* benchmark. The MTTF increased by 11.7%, while the design size increased by 20.1% after the selected GPDRC was incorporated in place of the simplex one. Also, each measurement shows the mean values of absolute injection numbers leading to a failure. These keep in the magnitude of hundreds for our benchmarks. As can be observed, one GPDRC brings various effectiveness, which is dependent on the characteristics of the benchmark (e.g., its size and functionality).

We also want to estimate the resulting improvement for future larger systems. However, the exact measurement is not currently possible, as we are limited by the FPGA area currently available on our equipment. For this reason, we extrapolate the percentage improvement of MTTF in Table 5. This extrapolation is calculated for hypothetical 100, 250 and 500 kbit systems. The critical fact is that, with larger systems, the GPDRC overhead percentage decreases, while the MTTF is yet expected to increase.

7. Conclusions and future work

This article presented a new automated flow for FT systems design on dynamically reconfigurable FPGAs. Subsequently, the practical usage of our implemented automation toolkit was shown. In the first part of the experimental case study, we utilized the automation flow to prepare a set of RCs that are Pareto-optimal concerning the MTTF, power consumption, and size parameters. Then, one selected solution was evaluated in benchmark systems on a real HW. For this, benchmarking designs from the ITC'99 set were utilized.

The results show that by changing a standard RC with our automatically-designed FT version, for one specific application, the design size increased by 20.1%, and the MTTF increased by 11.7%. However, the efficiency is highly dependent on the target system size, MTTF, and function. Our experiments demonstrate the suitability of GPDRCs for shorter mission times. We also estimate that a complex system defined by half a million configuration bits would gain an MTTF improvement of more than 50%.

As a part of our future research, we would like to repeat this process for an RC of a different structure, for example, a simple microprocessor

Table 4
Parameters of the simplex and selected hardened GPDRCs, measured on benchmark systems on the Xilinx Virtex-5 FPGA technology and estimated for equivalent technology with a 2.5-inch Al shielding on 555.6 km-high orbital trajectory.

Benchmark name	GPDRC version	Size [bits]	Fault intensity [inj/s/bit]	Mean time between FIs [ms]	MTTF (Testbed) [ms]	MTTF (Orbit) [days]	FIs to failure [-]	Difference (*simplex*)		
								MTTF [%]	Size [%]	FIs to Fail. [%]
b01 in TMR	simplex	**21 120**	2×10^{-5}	2 367.4	**371 477**	**17.69**	157	–	–	–
	selected	**36 224**	2×10^{-5}	1 380.3	**388 085**	**18.48**	271	**+4.5**	**+71.5**	**+79,0**
b05 in TMR	simplex	**61 248**	2×10^{-5}	816.4	**85 877**	**4.09**	105	–	–	–
	selected	**75 776**	2×10^{-5}	659.8	**89 673**	**4.27**	136	**+4.4**	**+23.7**	**+29.5**
b12 in TMR	simplex	**69 184**	2×10^{-5}	722.7	**158 613**	**7.55**	219	–	–	–
	selected	**83 072**	2×10^{-5}	601.9	**177 149**	**8.44**	294	**+11.7**	**+20.1**	**+34.2**

Table 5
Extrapolated values for larger designs with the simplex and hardened GPDRCs, based on previously measured data on the Xilinx Virtex-5 FPGA technology.

Hypothetical benchmark size [bits]	GPDRC version	Size with GPDRC [bits]	Diff. (simplex)	
			MTTF [%]	Size [%]
100 000	simplex	**118 688**	–	–
	selected	**133 984**	**+13.9**	**+12.9**
250 000	simplex	**268 688**	–	–
	selected	**283 984**	**+29.1**	**+5.7**
500 000	simplex	**518 688**	–	–
	selected	**533 984**	**+54.4**	**+2.9**

core with a program code. This would extend the set of available RCs. Also, such experiments would yield the knowledge needed to select the proper RC for a given system and target environment. We also want to design a new GPDRC that allows renewing itself using the same DPR technology. This would supposedly provide even better results. This is because the GPDRCs themselves are prone to the accumulation of failures.

CRediT authorship contribution statement

Jakub Lojda: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Data curation, Writing – original draft, Visualization. **Richard Panek:** Software, Validation, Investigation, Data curation, Writing – original draft, Visualization. **Lukas Sekanina:** Resources, Writing – review & editing, Supervision, Funding acquisition. **Zdenek Kotasek:** Resources, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This work was supported by the Brno University of Technology, Czechia under project number FIT-S-23-8141 and the Czech Science Foundation Project 21-13001S.

References

[1] M. Straka, J. Kastil, Z. Kotasek, Generic partial dynamic reconfiguration controller for fault tolerant designs based on FPGA, in: NORCHIP 2010, 2010, pp. 1–4, http://dx.doi.org/10.1109/NORCHIP.2010.5669477.
[2] M. Straka, J. Kastil, Z. Kotasek, SEU simulation framework for xilinx FPGA: First step towards testing fault tolerant systems, in: 14th EUROMICRO Conference on Digital System Design, IEEE Computer Society, 2011, pp. 223–230.
[3] NASA, Mars 2020 Perseverance Rover, NASA, 2020, URL https://mars.nasa.gov/mars2020/. (Accessed 11 April 2021).
[4] F. Fallahlalehzari, How does the Mars perseverance rover benefit from FPGAs as the main processing units? Aldec (2021) URL https://www.aldec.com/en/company/blog/188--how-does-the-mars-perseverance-rover-benefit-from-fpgas-as-the-main-processing-units. (Accessed 11 April 2021).
[5] I. Kuon, R. Tessier, J. Rose, FPGA Architecture: Survey and Challenges, in: Foundations and trends in electronic design automation, 2008, Published, sold, and distributed by now Publishers, URL https://books.google.cz/books?id=AdK2OWDP7L0C.
[6] P. Athanas, Embedded Systems Design with FPGAs, Springer, New York, 2013.
[7] R. Padovani, Reconfigurable FPGAs for Space – Present and Future, Presentation on the MAPLD Conference, Washington, DC, 2005.
[8] K. Vipin, S.A. Fahmy, FPGA dynamic and partial reconfiguration: A survey of architectures, methods, and applications, ACM Comput. Surv. 51 (4) (2018) http://dx.doi.org/10.1145/3193827.
[9] Xilinx Inc., Partial reconfiguration user guide, 2013, https://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/ug702.pdf. (Accessed: 15 September 2021).
[10] R. Velazco, D. McMorrow, J. Estela, Radiation Effects on Integrated Circuits and Systems for Space Applications, Springer, 2019, http://dx.doi.org/10.1007/978-3-030-04660-6.
[11] E. Petersen, Single Event Effects in Aerospace, John Wiley & Sons, Ltd, 2011, pp. 1–12, http://dx.doi.org/10.1002/9781118084328.ch1, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781118084328.ch1. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118084328.ch1.
[12] J.-C. Geffroy, G. Motet, Design of Dependable Computing Systems, Springer Science & Business Media, 2013.
[13] I. Koren, C.M. Krishna, Fault-Tolerant Systems, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007.
[14] R.E. Lyons, W. Vanderkulk, The use of triple-modular redundancy to improve computer reliability, IBM J. Res. Dev. 6 (2) (1962) 200–209.
[15] C. Bolchini, A. Miele, 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, 2007, pp. 87–95.
[16] J. Heiner, B. Sellers, M. Wirthlin, J. Kalb, FPGA partial reconfiguration via configuration scrubbing, in: 2009 International Conference on Field Programmable Logic and Applications, 2009, pp. 99–104, http://dx.doi.org/10.1109/FPL.2009.5272543.
[17] R. Giordano, D. Barbieri, S. Perrella, R. Catalano, G. Milluzzo, Configuration self-repair in Xilinx FPGAs, IEEE Trans. Nucl. Sci. 65 (10) (2018) 2691–2698, http://dx.doi.org/10.1109/TNS.2018.2868992.
[18] C. SOOS, SEU effects in FPGA: how to deal with them?, in: Presentation on the 1st Combined R2E Workshop & School-Days, European Organization for Nuclear Research (CERN), 2009.
[19] J. Lojda, J. Podivinsky, Z. Kotasek, M. Krcma, Majority type and redundancy level influences on redundant data types approach for HLS, in: 2018 16th Biennial Baltic Electronics Conference, BEC, 2018, pp. 1–4, http://dx.doi.org/10.1109/BEC.2018.8600951.

[20] M. Liu, Z. Zeng, F. Su, 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.

[21] S. Rudrakshi, V. Midasala, S. Bhavanam, Implementation of FPGA based fault injection tool (FITO) for testing fault tolerant designs, IACSIT Int. J. Eng. Technol. 4 (5) (2012) 522–526.

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

[23] T. Nidhin, A. Bhattacharyya, R. Behera, T. Jayanthi, 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.

[24] A. Benso, A. Bosio, S. Di Carlo, 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.

[25] T. Schweizer, D. Peterson, J.M. Kühn, T. Kuhn, 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, p. 236.

[26] C. Lavin, M. Padilla, P. Lundrigan, B. Nelson, B. Hutchings, Rapid prototyping tools for FPGA designs: RapidSmith, in: Field-Programmable Technology (FPT), 2010 International Conference on, 2010, pp. 353–356, http://dx.doi.org/10.1109/FPT.2010.5681429.

[27] J.M. Kuuhn, T. Schweizer, D. Peterson, T. Kuhn, 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.

[28] C. López-Ongil, M. Garcia-Valderas, M. Portela-García, L. Entrena, Autonomous fault emulation: A new FPGA-based acceleration system for hardness evaluation, IEEE Trans. Nucl. Sci. 54 (1) (2007) 252–261.

[29] M. Alderighi, S. D'Angelo, M. Mancini, G.R. Sechi, A fault injection tool for SRAM-based FPGAs, in: On-Line Testing Symposium, 2003. IOLTS 2003. 9th IEEE, IEEE, 2003, pp. 129–133.

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

[31] J. Podivinsky, J. Lojda, O. Cekan, 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, http://dx.doi.org/10.1109/DSD.2018.00051.

[32] J. Lojda, J. Podivinsky, O. Cekan, R. Panek, Z. Kotasek, FT-EST framework: Reliability estimation for the purposes of fault-tolerant system design automation, in: 2018 21st Euromicro Conference on Digital System Design, DSD, 2018, pp. 244–251, http://dx.doi.org/10.1109/DSD.2018.00053.

[33] G. Lee, D. Agiakatsikas, T. Wu, E. Cetin, 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, http://dx.doi.org/10.1109/FCCM.2017.57.

[34] A. Canis, J. Choi, M. Aldham, V. Zhang, A. Kammoona, T. Czajkowski, S. Brown, J. Anderson, Legup: An open-source high-level synthesis tool for FPGA-based processor/accelerator systems, ACM Trans. Embed. Comput. Syst. (TECS) 13 (2013) http://dx.doi.org/10.1145/2514740.

[35] Xilinx Inc., TMRTool: The industry's first development tool to automatically generate triple module redundancy (TMR) for space-grade re-programmable FPGAs, 2021, https://www.xilinx.com/products/design-tools/tmrtool.html. (Accessed 13 April 2021).

[36] S. Kulis, Single event effects mitigation with TMRG tool, J. Instrum. 12 (01) (2017) C01082, URL http://stacks.iop.org/1748-0221/12/i=01/a=C01082.

[37] A.R. Khatri, A. Hayek, J. Borcsok, RASP-TMR: An automatic and fast synthesizable verilog code generator tool for the implementation and evaluation of TMR approach, Int. J. Adv. Comput. Sci. Appl. 9 (8) (2018).

[38] BYU EDIF tools homepage, 2021, http://reliability.ee.byu.edu/edif/. (Accessed 13 April 2021).

[39] J. Anwer, M. Platzner, S. Meisner, FPGA redundancy configurations: An automated design space exploration, in: 2014 IEEE International Parallel Distributed Processing Symposium Workshops, 2014, pp. 275–280, http://dx.doi.org/10.1109/IPDPSW.2014.37.

[40] J. Lojda, R. Panek, Z. Kotasek, Automatic design of fault-tolerant systems for VHDL and SRAM-based FPGAs, in: 2021 24th Euromicro Conference on Digital System Design, DSD, 2021, pp. 549–552, http://dx.doi.org/10.1109/DSD53832.2021.00088.

[41] J. Onishi, S. Kimura, R.J. James, Y. Nakagawa, Solving the redundancy allocation problem with a mix of components using the improved surrogate constraint method, IEEE Trans. Reliab. 56 (1) (2007) 94–101.

[42] K. Khalili-Damghani, A.-R. Abtahi, M. Tavana, A new multi-objective particle swarm optimization method for solving reliability redundancy allocation problems, Reliab. Eng. Syst. Saf. 111 (2013) 58–75.

[43] Y.-C. Liang, Y.-C. Chen, Redundancy allocation of series-parallel systems using a variable neighborhood search algorithm, Reliab. Eng. Syst. Saf. 92 (3) (2007) 323–331.

[44] G. Kanagaraj, S. Ponnambalam, N. Jawahar, A hybrid cuckoo search and genetic algorithm for reliability–redundancy allocation problems, Comput. Ind. Eng. 66 (4) (2013) 1115–1124.

[45] Z. Wang, T. Chen, K. Tang, X. Yao, A multi-objective approach to redundancy allocation problem in parallel-series systems, in: 2009 IEEE Congress on Evolutionary Computation, IEEE, 2009, pp. 582–589.

[46] W.-C. Yeh, T.-J. Hsieh, Solving reliability redundancy allocation problems using an artificial bee colony algorithm, Comput. Oper. Res. 38 (11) (2011) 1465–1473.

[47] J. Lojda, J. Podivinsky, Z. Kotasek, M. Krcma, Data types and operations modifications: A practical approach to fault tolerance in HLS, in: 2017 IEEE East-West Design and Test Symposium, EWDTS, 2017, pp. 1–6, http://dx.doi.org/10.1109/EWDTS.2017.8110113.

[48] J. Lojda, J. Podivinsky, O. Cekan, R. Panek, M. Krcma, Z. Kotasek, Automatic design of reliable systems based on the multiple-choice knapsack problem, in: 2020 23rd International Symposium on Design and Diagnostics of Electronic Circuits Systems, DDECS, 2020, pp. 1–4, http://dx.doi.org/10.1109/DDECS50862.2020.9095576.

[49] J. Lojda, R. Panek, Z. Kotasek, Automatically-designed fault-tolerant systems: Failed partitions recovery, in: 2021 IEEE East-West Design and Test Symposium, EWDTS, 2021, pp. 1–8, http://dx.doi.org/10.1109/EWDTS52692.2021.9580996.

[50] J. Taylor, Introduction to Error Analysis: The Study of Uncertainties in Physical Measurements, University Science Books, 1997, URL https://books.google.cz/books?id=giFQcZub80oC.

[51] C. Coello, C. Dhaenens, L. Jourdan, Advances in Multi-Objective Nature Inspired Computing, Vol. 272, Springer, 2009, http://dx.doi.org/10.1007/978-3-642-11218-8.

[52] Xilinx Inc., LogiCORE IP ChipScope Pro Integrated Controller (ICON) Documentation, Xilinx Inc., 2011, https://www.xilinx.com/support/documentation/ip_documentation/chipscope_icon/v1_05_a/chipscope_icon.pdf. (Accessed 15 February 2018).

[53] Xilinx Inc., ChipScope pro VIO documentation, 2009, https://www.xilinx.com/support/documentation/ip_documentation/chipscope_vio.pdf. (Accessed 15 February 2018).

[54] Xilinx Inc., Xilinx XPower Analyzer, Xilinx Inc., 2013, https://www.xilinx.com/html_docs/xilinx14_5/isehelp_start.htm#xpa_c_overview.htm. (Accessed 07 December 2021).

[55] Xilinx Inc., ISim User Guide, Xilinx Inc., 2012, https://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/plugin_ism.pdf. (Accessed 07 December 2021).

[56] R. Panek, J. Lojda, J. Podivinsky, Z. Kotasek, Reliability analysis of reconfiguration controller for FPGA–based fault tolerant systems: Case study, in: 2020 International Symposium on VLSI Design, Automation and Test, VLSI-DAT, 2020, pp. 1–4, http://dx.doi.org/10.1109/VLSI-DAT49148.2020.9196269.

[57] Xilinx Inc., Virtex-5 FPGA Configuration User Guide, Xilinx Inc., 2017, https://www.xilinx.com/support/documentation/user_guides/ug191.pdf. (Accessed 22 November 2017).

[58] Xilinx Inc., Virtex-5 FPGA User Guide, Xilinx Inc., 2012, https://www.xilinx.com/support/documentation/user_guides/ug190.pdf. (Accessed 26 March 2019).

[59] Xilinx Inc., ML506 Evaluation Platform User Guide, 2011, UG347 (V3. 1.2).

[60] A. Razzaq, A. Ye, Static power model for CMOS and FPGA circuits, IET Comput. Digit. Tech. 15 (4) (2021) 263–278, http://dx.doi.org/10.1049/cdt2.12021, arXiv:https://ietresearch.onlinelibrary.wiley.com/doi/pdf/10.1049/cdt2.12021. URL https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/cdt2.12021.

[61] S. Mukhopadhyay, C. Neau, R.T. Cakici, A. Agarwal, C.H. Kim, K. Roy, Gate leakage reduction for scaled devices using transistor stacking, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 11 (4) (2003) 716–730.

[62] R. Panek, J. Lojda, J. Podivinsky, Z. Kotasek, Reliability analysis of the FPGA control system with reconfiguration hardening, in: 2021 24th Euromicro Conference on Digital System Design, DSD, 2021, pp. 553–556, http://dx.doi.org/10.1109/DSD53832.2021.00089.

[63] F. Corno, M. Reorda, G. Squillero, RT-level ITC'99 benchmarks and first ATPG results, IEEE Des. Test Comput. 17 (3) (2000) 44–53, http://dx.doi.org/10.1109/54.867894.

[64] D.M. Hiemstra, G. Battiston, P. Gill, Single event upset characterization of the Virtex-5 field programmable gate array using proton irradiation, in: 2010 IEEE Radiation Effects Data Workshop, 2010, p. 4, http://dx.doi.org/10.1109/REDW.2010.5619490.

Jakub Lojda (Ph.D. student, FIT BUT) was born in 1991. In 2015 he graduated (MSc.) at the Department of Computers Systems of the Faculty of Information Technology, Brno University of Technology (BUT). In 2015 he started his Ph.D. studies at the Department of Computers Systems (DCSY). His scientific research is focused on the design of fault-tolerant systems on FPGAs. His research target is the automation of the complete fault-tolerant system design process.

Richard Panek (Ph.D. student, FIT BUT) was born in 1990. In 2015 he graduated (MSc.) at the Department of Computers Systems of the Faculty of Information Technology, Brno University of Technology (BUT). There, in 2016, he obtained another MSc degree. In the same year, he started his Ph.D. studies at the Department of Computers Systems (DCSY). His scientific research is focused on fault-tolerant control systems that utilize the dynamic partial reconfiguration of FPGAs.

Lukas Sekanina (Senior Member, IEEE) received the Ing. and Ph.D. degrees from the Brno University of Technology, Brno, Czech Republic, in 1999 and 2002, respectively. He was a Visiting Professor with Pennsylvania State University, Erie, PA, USA, in 2001. He received the Fulbright Scholarship to work with the NASA Jet Propulsion Laboratory, Caltech, in 2004. He is currently a Full Professor and the Head of the Department of Computer Systems, Faculty of Information Technology, Brno University of Technology. He has coauthored over 200 papers, mainly on evolvable hardware, evolutionary computation, and approximate computing, and one patent. He served as an Associate Editor for the IEEE Transactions on Evolutionary Computation, from 2011 to 2014, the Genetic Programming and Evolvable Machines Journal, and the International Journal of Innovative Computing and Applications.

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

Appendices

Appendix I

List of Used Abbreviations

AD	Activity Diagram
ASIC	Application Specific Integrated Circuit
BL-TMR	Brigham Young University and Los Alamos National Laboratory TMR Tool
BRAM	Block Random Access Memory
CDFG	Control-Data Flow Graph
CGTMR	Coarse-grained TMR
CLB	Configuration Logic Block
DPR	Dynamic Partial Reconfiguration
DSP	Digital Signal Processing
DUT	Design Under Test
EEPROM	Electrically Erasable Programmable Read-Only Memory
FA	Fault Avoidance
FF	Flip Flop
FGTMR	Fine-Grained Triple Modular Redundancy
FPGA	Field Programmable Gate Array
FT	Fault Tolerance
FT-EST	Fault Tolerance Estimation Toolkit
GPDRC	Generic Partial Dynamic Reconfiguration Controller
HDL	Hardware Description Language
HLL	Higher-Level Programming Language
HLS	High-Level Synthesis

ICAP	Internal Configuration Access Port
II	Initiation Interval
ISE	Integrated Synthesis Environment
I/O	Input/Output
JTAG	Joint Test Action Group
KP	Knapsack Problem
LE	Logic Element
LFSR	Linear Feedback Shift Register
LUT	Look-Up Table
MCKP	Multiple-choice Knapsack Problem
MTTF	Mean Time to Failure
NSGA-II	Non-Dominated Sorting Genetic Algorithm II
PLB	Programmable Logic Block
RDT	Redundant Data Type
RTL	Register-Transfer Level
SEE	Single-Event Effect
SET	Single-Event Transient
SEU	Single-Event Upset
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
TMR	Triple Modular Redundancy
TTF	Time to Failure
t50	Median Time to Failure
UML	Universal Modeling Language