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 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-

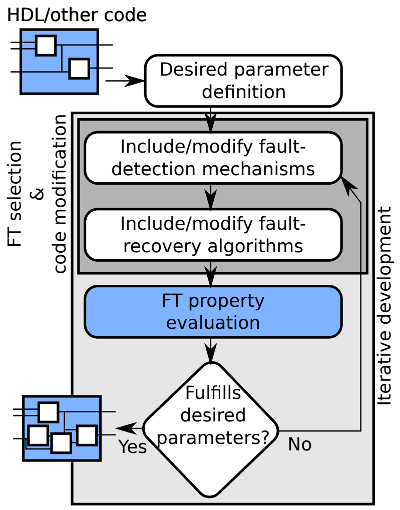
HDL/other code
Desired parameter definition
FT selection & code modification
Include/modify fault-detection mechanisms
Include/modify fault-recovery algorithms
Iterative development
FT property evaluation
Fulfills desired parameters?
Yes
No

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].

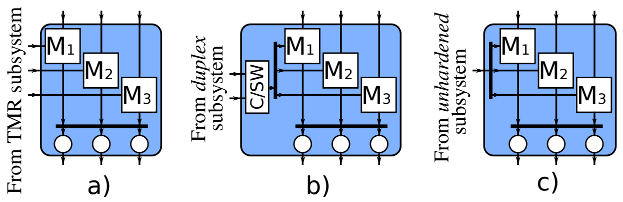

Figure 2: Three types of cases that can be distinguished when considering binary operations: (a) *intra-DT* operation between two TMR subsystems; (b) *inter-DT* operation between a system with TMR and one with *duplex* hardening; and (c) *original-DT* operation between TMR and *unhardened* subsystems.

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.

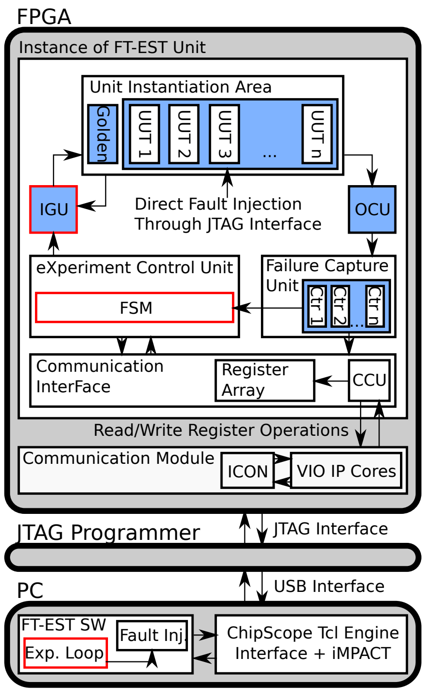

Figure 3: Simplified architecture of the FT-EST system; the parts highlighted in blue are dynamically and fully automatically generated, while the parts highlighted in red are to be provided by the designer to specify the experiment setup.

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.

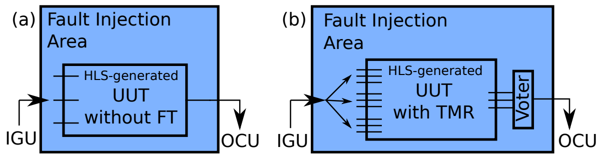

Figure 4: The architecture of the testing for (a) a simplex component and also for (b) the TMR component utilizing the concept of RDTs.

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.

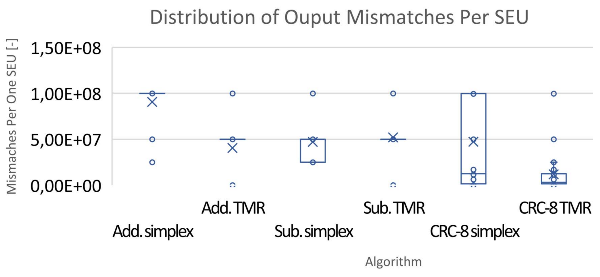

Figure 5: The distribution of mismatched outputs quantities caused by one SEU during one test cycle (test cycles that did not show any output errors are omitted; median value is marked by a cross).

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	Deviation Range of the Estimation [% points]					
cove-rage	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.