Evaluation of the SEU Faults Coverage of a Simple Fault Model for Application-Oriented FPGA Testing

Jaroslav Borecký, Robert Hülle, Petr Fišer
Department of Digital Design, Faculty of Information Technology
Czech Technical University in Prague, Technická 9, Prague, Czech Republic
{borecjar, hullerob, fiserp}@fit.cvut.cz

***Abstract*—Testing of FPGA-based designs persists to be a challenging task because of the complex FPGA architecture with heterogeneous components, and therefore a complicated fault model. The standard stuck-at fault model has been found insufficient. On the other hand, very precise FPGA fault models have been recently devised. However, these models are often excessively complex and require a lot of resources (run-time, memory) to manipulate with. In this paper, we propose a simple yet efficient combined fault model comprising bit-flips in look-up tables and stuck-at faults in the rest of logic. On top of this model, a dedicated SAT-based application-oriented ATPG has been designed. The main contribution of this paper is the evaluation of efficiency of the fault model with the respective ATPG by exhaustive hardware emulation of all possible SEUs in the configuration memory that may influence the functionality of the circuit implemented in the FPGA. We show that the obtained fault coverage reaches up to more than 99%, which makes the method applicable in practice. Even though combinational circuits are assumed only, the method can be used to quickly test safety-critical combinational cores.**

***Index Terms*—FPGA, SEU, fault model, application-oriented testing, ATPG, SAT**

I. Introduction

Field-Programmable Gate Array (FPGA) chips are becoming increasingly popular even in one-purpose applications where reconfiguration is not required, due to their low cost, high availability, and simplicity of the design process. Unfortunately, FPGAs are more susceptible to radiation than ASICs. The main reason for this susceptibility is a large amount of SRAM cells in the FPGA configuration memory, which are highly prone to Single-Event Upsets (SEUs) [1]–[4]. A SEU in the configuration memory may alter the design structure and thus the functionality of the implemented circuit. Hence, testing of applications implemented in FPGAs is increasingly important. This is emphasized by the fact that FPGAs are being increasingly deployed in safety-critical space applications, where the SEU rate is increased, while the test/repair capabilities are limited [5], [6].

Generally, there are two approaches to test FPGA-based designs. One is the *manufacture-oriented* testing, where the FPGA fabric is tested, without considering any application implemented in it. In this approach, the FPGA interconnect and logic (CLBs, ALMs, BRAMS) are tested separately, taking advantage of the regular structure of FPGAs [7]–[13]. This always involves multiple reconfigurations of the FPGA. The whole FPGA chip is tested in this way, even though some FPGA resources cannot affect the functionality of the implemented design (unused FPGA resources, redundant logic, etc.). This makes such an approach impractical for in-field testing, as the FPGA must be reconfigured for the test purposes, and the implemented design must be reloaded into the FPGA after the test is executed. Also, false fault positives may be indicated by the test, since the above-mentioned unused FPGA resources are covered by the test.

In the second approach, the *application-oriented* testing, the implemented design is tested only, without considering the unoccupied FPGA resources [14]–[22]. Many of these approaches also require reconfiguration for the test purposes [17], [19], [21], involve complicated fault models [15], [20], or a complex testing infrastructure that must be embedded in the FPGA [16], [22].

Recently we have published an application-oriented testing approach *not needing any FPGA reconfiguration* [23]. It can be advantageously used in in-field testing, where reconfiguration is not practical, or is even impossible in case of flash-based [24], or antifuse-based [25] FPGAs. For the sake of completeness, basic principles of this method will be shortly reviewed. Then its thorough experimental evaluation will be presented. The main novel principles of [23] are:

- In contrast to most of previously mentioned approaches, this method *does not* take advantage of the regular FPGA structure. Instead, a dedicated Automated Test Patterns Generator (ATPG) is used to produce test patterns for a given circuit implemented in the FPGA.
- The method is applicable to any FPGA type/family/vendor, provided the gate-level description (e.g, EDIF netlist) of the implemented design is available. This fortunately holds for all leading FPGA vendors.
- As the standard stuck-at fault model has been found insufficient for FPGA testing [14], [15], [26] and precise fault models are often excessively complex and require a lot of resources (run-time, memory) to process [27]–[29], a simplified fault model was been offered. It consists of SEUs in Look-Up Tables (LUTs) and stuck-at faults in the rest of logic, including the interconnect. The combination of SEU faults in LUTs and stuck-at faults allows to test the vast majority of faults affecting the occupied logic. The stuck-at fault model allows to efficiently test routing faults, independently of the interconnect implementation.

- A SAT-based ATPG able to easily accommodate both fault models is provided.

The work [23] presents just a fault model and an ATPG tailored to it. No implementation platform has been given. It is assumed that the test is generated off-line and then applied to the on-line tested FPGA by any means, e.g., using a simple built-in self-test (BIST) structure. One of the drawbacks of the method is the need to suspend the application during the test. However, the test is typically short enough, thus this needs not cause serious problems.

Building on top of this previous work, the main contribution of this paper is the experimental evaluation of the fault model and the proposed ATPG. By *exhaustive hardware emulation* of all SEUs that may affect the functionality of the implemented circuit, we show that more than 99% of SEUs are detected by our approach. The undetected faults are typically caused by unpredictable behavior, due to, e.g., antennas or a sequential behavior induced by a SEU in the FPGA switching matrices. We have also validated the fault model by constraining it either to the stuck-at faults at circuit signals or bit-flips in LUTs only. We show that these simplest fault models are not sufficient enough, whereas the combined model is acceptable. As a result, the simple fault model can be judged as efficient enough to be applied in practice.

The rest of the paper is organized as follows: after this Introduction, a review of the related work is given in Section II. The fault model used is discussed in Section III. Basic principles of the SAT-based ATPG used are briefly described in Section IV. The main contributions of this paper – the fault emulation platform and the fault coverage evaluation method – are described in Sections V and VI, respectively. Experimental results are shown in Section VII. Section VIII concludes the paper.

II. Related Work

Application-oriented FPGA testing has been tackled for two decades already [14]. It brings some benefits over the manufacture-oriented (application-independent) testing, like the testing speed and no need to test unused resources.

The first hints of insufficiency of a standard stuck-at fault model were presented in [14]. The fact that faults in the unused logic should not be accounted has been pointed out. Still, the interconnection faults were neglected.

A fault model for application-oriented FPGA testing was proposed in [15]. The authors have correctly observed that a special fault model must be used for LUTs: even though stuck-at faults in LUTs are considered too, they are assumed to be also in LUT parts that are seemingly redundant. Particularly, stuck-at faults in LUTs corresponding to an inversion of a configuration memory bit are included in the fault model. These are, indeed, bit-flips assumed in [23] and in this paper. Apart from these LUT faults, also stuck-at faults in the rest of logic are considered. This is exactly what is covered by our fault model too. However, their transformation of bit-flips to stuck-at faults is rather cumbersome. The SAT-based ATPG used in this paper enables a native support of bit-flip faults.

A genetic algorithm (GA) was used to generate test patterns for SEUs in [20]. The same fault model as in [15] was used there. Since fault simulation is used for the GA fitness computation, there is no need to transform bit-flips to stuck-at faults. Actually, the GA does not need to know details of the FPGA structure.

A different testing strategy was used in [17], [19]. Specific fault models for interconnection and LUTs are used as well. However, this approach needs multiple reconfigurations of the FPGA in order to test all types of resources.

A SAT-based approach to application oriented testing primarily targeting feedback bridging faults was presented in [21]. Again, multiple test configurations were needed.

Another conceptually different approach to fault modeling was used in [27], [28]. A logic-level model of all SEU-induced faults in the FPGA application logic was constructed for simulation purposes. The emphasis was put on interconnect faults. Particularly, faults affecting routing resources were interpreted as equivalent logic effects in the application netlist. The model is accurate, and this accuracy is paid by the model complexity and therefore long processing run-times.

The paper [29] presents an application-oriented ATPG detecting untestable faults in FPGA. It is based on formal methods (model-checking). Test patterns for detectable faults are generated as model-checking counterexamples. An accurate fault model for both logic components and routing structures is used. Despite of the accuracy, no detailed knowledge of the FPGA architecture is needed. The approach can be used on-line, using dynamic reconfiguration. All the accuracy and flexibility is however paid by very long processing run-times; even small circuits having tenths of LUTs need tenths of minutes to compute the test.

A recent paper [22] presents an on-line and on-demand FPGA testing approach, where unused parts of the FPGA are selectively tested before they are possibly used after reconfiguration. Only routing resources are tested, as they represent the vast majority of configuration bits. The fault model we use assumes faults in LUTs and other resources (carry chains, multiplexers, etc.) as well.

In contrast to all approaches to application-oriented testing, a similar task can be performed by the memory read-back instead. Unfortunately, this feature is supported in the recent FPGA devices only. However, old and robust devices are preferred in mission-critical space applications. The method [23] is general and can be used for any FPGA device. Next, the memory read-back checks the whole content of the configuration memory, regardless of its usage. Therefore, false positives may be reported, when a SEU occurs in an unused part of the FPGA. One remedy can be the *scrubbing* technique [30]. Here the configuration memory content is periodically refreshed by data stored in a permanent (or more robust) memory. But again, support for this technique needs not be present in older FPGA families.

III. The Fault Model Used

The fault model used in [23] and in this paper will be shortly summarized, for completeness.

The main resources present in contemporary FPGA devices are typically:

1) look-up tables (LUTs) of different sizes, typically contained in Configurable Logic Blocks (CLBs) or Adaptive Logic Modules (ALMs), together with flip-flops and multiplexers,
2) device specific primitives, such as fast carry chains (dedicated XOR gates),
3) interconnect and switch-boxes,
4) I/O and other communication blocks, and
5) special complex features, such as block-RAMs, DSP blocks, CPUs, etc.

The last two categories of resources will not be considered in this paper, since their testing involves dedicated approaches, as shown, e.g., in [31], [8], [32], [33].

Moreover, combinational circuits only will be assumed in this paper for simplicity. Sequential circuits testing would involve using a sequential ATPG (which could be feasible for smaller designs) or using special design-for-testability (DFT) approaches [34].

In order to test the first three types of resources, the fault model comprises of *stuck-at* and *bit-flip* faults. The motivation for this choice was the fact, that neither the stand-alone simple stuck-at fault model nor the bit-flip fault model is sufficient enough [15], [26]. The secondary motivation was the simplicity of the model.

In this combined model, all bit-flips in LUTs will be covered (from its principle) and we assume that faults in the other parts of the combinational logic (MUXes, XORs, and also most of faults in the interconnect) will be covered by the stuck-at model.

In order to apply this fault model and submit the implemented design to the ATPG, a *primitive-level* description of the design is needed. Fortunately, this is completely possible; most commercial FPGA design tools support this feature, e.g., in the form of an EDIF file describing the mapped netlist. It is expected that only minor changes will be made in the place&route step.

Note that our approach is much different from standard application-oriented FPGA testing techniques. The interconnect is tested as a *logical* part of the implemented circuit, not the physical one. Particularly, the switch-boxes are not tested at the transistor level [12] or just by verification of the point-to-point connection without considering the connected logic [7], [10], [17]. We even do not resort to modeling the interconnect resources at logic level, as in [27], [28]. For our purposes, it is not required.

The drawback of our approach is that some SEU faults may escape by unpredictable behavior caused by, e.g., antennas (disconnections) and possible SEU-induced combinational (or even sequential) feedbacks. However, testing these SEU consequences is a difficult problem in any scenario, as it requires a deep analysis of the FPGA behavior.

Most importantly, combinational elements, like MUXes and XORs, are tested explicitly, not in the FPGA family dependent manner [11]. This makes the proposed approach *universal and flexible*. Provided a gate-level description of the mapped design can be obtained, the method can be used for any FPGA family from any vendor.

IV. SAT-Based ATPG Algorithm Used

The widely used structural ATPG algorithms [35]–[37] are typically based on the stuck-at fault model and are difficult to modify to accept other (or more complex) fault models. For this reason, the proposed ATPG is based on the Boolean Satisfiability solving problem (SAT) [38]–[42], making it flexible enough to accommodate the combined fault model efficiently [23].

A. The Basic Algorithm

Basic concepts of SAT-based ATPG have been introduced already in 1990's [38]. Many significant enhancements have been proposed since then [39]–[42]. However, the original principal concept is sufficient for our experiment, as we do not target the minimality of the test and the ATPG run-time; the aim of this paper is to make the *proof-of-concept* of the simple combined fault model.

The basic principle of SAT-based ATPG algorithms is as follows: a *miter*, i.e., conceptual hardware, is constructed from the tested circuit. The miter consists of the fault-free circuit and its copy with the targeted fault. Respective outputs of these circuits are then XOR-ed and the outputs of these XOR gates are OR-ed, to produce the miter output. The output of this conceptual circuit will equal to one for such input vectors, that detect the fault (i.e., at least one of the outputs of the fault-free and faulty circuits differ). For simplicity, only the logic affected by the tested fault is copied.

A SAT formula in a conjunctive normal form (CNF) is then constructed from this miter by the Tseitin transformation [43] and this CNF is submitted to a SAT solver. The satisfiability proof then equals to the test vector detecting the targeted fault. If the formula is unsatisfiable, the fault is proven redundant (undetectable).

The overall algorithm, in its most basic form, executes a loop, where some yet undetected fault is picked, a test vector is generated for it, and faults covered by this vector are dropped from the fault list. This loop is repeated until all faults are covered or marked as undetectable. For details see [38].

When the combined fault model is used (see Section III and Subsection IV-B), the fault list comprises of both types of faults – bit-blips (BF) in LUTs and stuck-at faults (S@) at the netlist signals. As seen from the experiments, the ordering of these fault sets does make a difference. For details see Section VII.

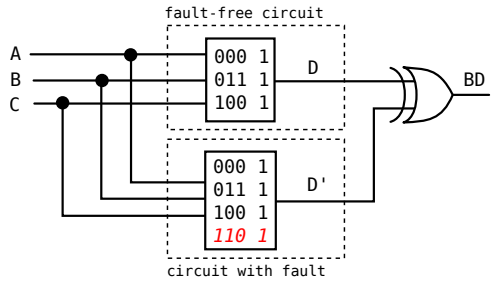

Fig. 1. Example of a conceptual model of a bit-flip fault in a LUT. The output of the circuit must be 1 to detect the fault. The bit-flip is indicated by italics

B. Fault Modeling

Generally, a fault of any kind is modeled as a modification of the faulty region of the miter. A stuck-at fault is modeled by a constant value of the affected signal, which is then represented as a unit clause in the CNF [38].

Modeling a LUT bit-flip is more complicated, but similarly straightforward: the fault is modeled by flipping the output value for a particular LUT input vector that excites the fault. For example, let us have a LUT described by 1-minterms $\{000, 011, 100\}$ and a bit-flip at address `110` which can be described by a minterm $\{110\}$. An illustrative example is shown in Figure 1. Let us note that the LUT content is modeled in the miter as a sum-of-minterms (SOM). Therefore, injecting a bit-flip involves just a modification of this SOM. For more details see [23].

V. Hardware Fault Emulation Platform

The main contribution of this paper is the *experimental evaluation* and consequently a *validation* of the proposed fault model. For this purpose, we have designed an FPGA-based *emulation platform*, where SEU faults can be injected into the bitstream and their *real effects*, i.e., their manifestation can be evaluated. Both the fault injection and response evaluation are performed in the FPGA.

The emulation platform was developed for the Zybo board with the Zynq-7010 device. The Zynq device contains the Processing System (PS) and Programmable Logic (PL) with the same architecture as an Artix-7 FPGA. The Processing System is equipped with a dual-core ARM CPU and several peripherals. In our emulation platform, the PL part is used for SEU emulation (configuration, fault injection) and executing all proposed tests. Both parts and their interaction are shown in Figure 2.

The platform has been designed in such a way, so that multiple tested circuits with multiple test vectors sets can be stored in external memory (SD card) and then processed in a batch mode.

A. Processing System

Processor Configuration Access Port (PCAP): is used for configuration of the whole Programmable Logic by a bitstream with the circuit under test and also serves for the fault injection.

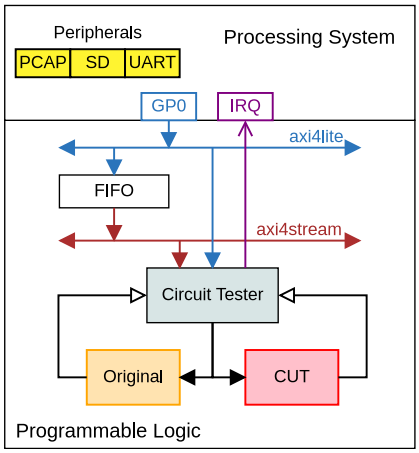

Fig. 2. Block scheme of the Hardware Emulator embedded in Zynq

SD card: contains test vector files and bitstreams for all tested circuits.

B. Programmable Logic

Original: is the original circuit, which is used as a reference, to produce fault-free responses.

Circuit Under Test (CUT): is the original circuit placed at a specific location in the PL, in order to inject SEU faults into it.

Circuit Tester: contains a Gray code counter generating a trivial test, i.e., all possible values at the primary inputs of the tested circuit, an AXI-stream control unit for inserting test vectors from a FIFO (which is fed by the *axi4lite* interface), and a comparator unit which detects a possible mismatch of the Original and the CUT logic.

Similar fault injection methods based on the same principles have been proposed in the literature. They use the internal or processor configuration access port and change the configuration memory to simulate the occurrence of a SEU in an FPGA. In [44], a fault injection platform that uses a Linear Feedback Shift Register (LSFR) to inject a fault into a random location is described. Approaches presented in [45], [46] use embedded processors to control the fault injection process.

All the above approaches serve the same purpose and could have been used for the experimental evaluation as well. However, we have opted for our previously developed solution [47], since significant architectural changes would have to be involved.

VI. Emulation-Based Fault Coverage Evaluation Method

The input to the process is the BLIF format [48], mainly due to the ATPG tool used [23]. Figure 3 shows the flow of the whole evaluation process.

The process starts with creating a VHDL file. A source BLIF file is converted to VHDL by our in-house tool, in a

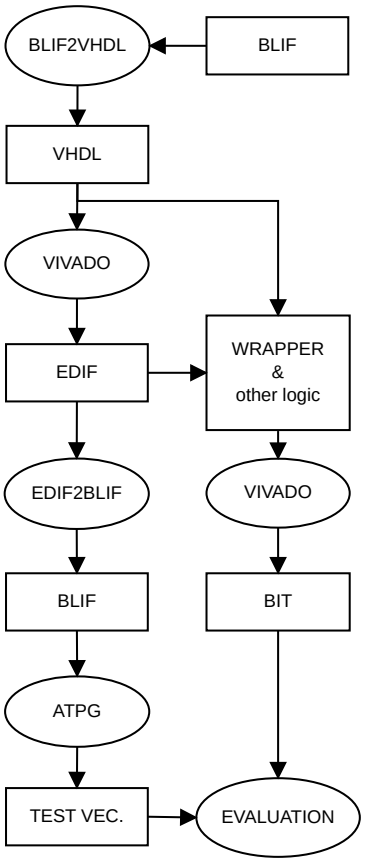

Fig. 3. The process flow diagram

straightforward way (the netlist of SOPs is converted to logic expressions). Then the process flow is split into two branches. The left branch shows the process of test vectors generation and the right branch shows the test design creation for an FPGA.

Test vectors generation: The next step is the synthesis using the Vivado tool that generates an EDIF file. This file is then converted by our in-house tool to BLIF with specific comments, so that the ATPG will be able to distinguish LUTs from the other logic. This converted file is used for the test vectors generation by the ATPG described in Section IV.

Test design creation: The whole test design is composed of the circuit wrapper created from the tested circuit VHDL file and its respective mapped EDIF file. This wrapper is merged with the tester logic which is described in Section V in detail. The final design is processed by the Vivado tool that generates a bitstream for the FPGA.

Both outputs are used for the final evaluation.

VII. Experimental Results

A mix of two standard benchmark sets LGSynth‘91 (MCNC) [49] and ISCAS‘89 [50] has been used in our experiments. We have exercised 83 circuits from these sets, those having up to 22 inputs. This limitation was necessary because of the need for the application of the exhaustive test, i.e., all 2^n vectors, where n is the number of the CUT inputs.

In order to compute the fault coverage of the ATPG-produced test, we first needed to determine the set of SEU faults that can affect the functionality of the implemented design. For this purpose, an exhaustive (trivial) test for each circuit was generated for each tested circuit using a Gray counter (see Subsection V-B).

During a more detailed testing it was discovered that certain faults occurring in LUTs are not covered. Moreover, it was found that the coverage of faults in routing resources also depends on the order of the test vectors. Undetected faults in LUTs are caused by the place&route tool in Vivado, which may merge two LUTs into a dual-output one, and its behavior (in terms of SEU effects) is slightly different. This fact is unfortunately not reflected in the exported EDIF file; the experiments were performed when dual-output LUTs were separated manually and all SEU faults in LUTs were covered. Also, a sequential behavior induced by SEUs in switch-boxes can happen. Therefore, also test vectors produced by all ATPG scenarios (fault models) were applied. We must admit that some SEU faults may have escaped even in this testing scenario. However, we believe that their number will be negligible.

Next, vectors produced by the ATPG were applied to the CUT, for different fault models. Particularly, the stuck-at (S@), bit-flip (BF) fault models, and their combination were used. In the case of the combined fault model, we have tried out two scenarios: either the S@ or BF faults were tested first (see Section IV).

The results for all fault models are shown in Table I. Average fault coverage values are calculated over all benchmark circuits for the specific fault models and the last column shows the total number test vectors produced by the ATPG, for all 83 circuits. The results show that the stuck-at fault model itself is insufficient, while the combination of the stuck-at and bit-flip model is the best option. The “Full” row indicates the total number of vectors applied to achieve the complete fault coverage.

Note that there is a slight difference in the ordering of fault models (S@, BF vs. BF, S@); the S@, BF scenario performs better, in terms of the test length and fault coverage. This issue will be discussed in the Conclusions.

TABLE I
Average coverage of various fault models

Fault Model	Avg. coverage	Sum of vectors
S@,BF	99.14%	24,502
BF,S@	98.95%	25,313
S@	69.46%	4,398
BF	99.01%	25,359
Full	100.00%	20,947,008

Fault coverages for all the 83 tested circuits, for the S@

fault model and the combined model (S@, BF) are shown in Figures 4 and 5. The circuits were sorted by the fault coverage in ascending order, for better visualization. We can see that the stuck-at fault model achieves the fault coverage ranging between 48% and 94% and the combination of the stuck-at with bit-flip fault model ranges between 92% and 100%, while the 100% fault coverage is achieved for most of circuits.

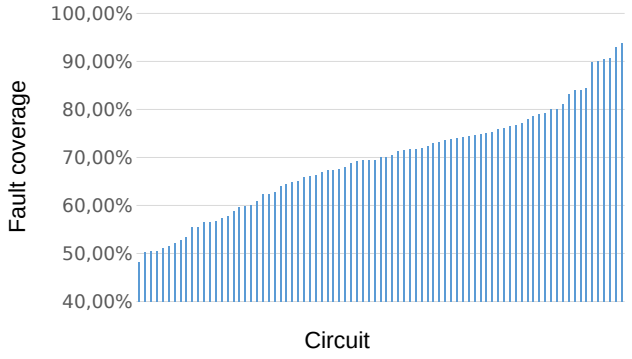

Fig. 4. Fault coverages for the S@ fault model

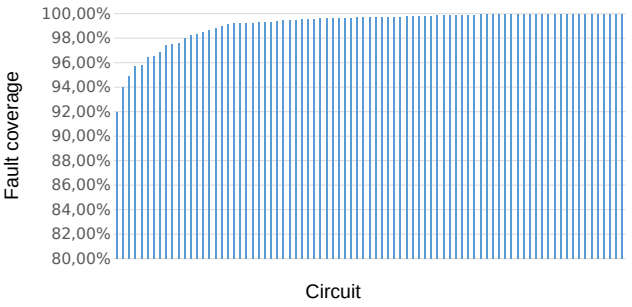

Fig. 5. Fault coverages for the combined S@, BF fault model

Detailed results for three fault models for some selected circuits are shown in Table II. After the circuit name, the numbers of its inputs and outputs follow. The "ADF" column shows the total number of detectable faults. Results for each fault model follow, showing the numbers of detected faults, the numbers of test vectors produced by the ATPG, and the respective fault coverages.

VIII. Conclusions and Discussion

A SAT-based ATPG for application-oriented FPGA testing employing a combined fault model has been presented. The proposed fault model comprises of stuck-at and bit-flip faults. The efficiency of such a fault model has been verified by an exhaustive hardware emulation, by which the real coverage of faults that may affect the functionality of the implemented design was determined. As a result, the *average* fault coverage reaches up to more than 99%, which makes the proposed application-oriented FPGA testing approach applicable in practice.

The experimental results indicate that the proposed fault model reflects the internal structure of the FPGA relatively precisely. While the stand-alone stuck-at fault model has been proven highly insufficient for FPGA testing, and the bit-flip fault model is not capable of covering faults outside of LUTs, the combined model is able to cover a vast majority of faults. Note that faults located in switch-boxes are assumed to be covered by the stuck-at model, since in our scenario (i.e., the application-oriented testing), a switch-box represents just a connection, no matter how complex its "internals" are.

Two test generation scenarios for this fault model were examined, particularly test generation for stuck-at faults first and then for SEUs, and the reverse scenario. The experiments have shown that the first one yields better fault coverage, even though fewer test vectors are generated. This can be explained by the fact that test vectors targeting stuck-at faults probably cover many SEU faults as well, unlike the opposite.

Note that all the experiments were performed using circuits with a small number of inputs (n), for the purpose of hardware emulation, where the exhaustive test (comprising 2^n vectors) is applied to obtain the total number of possible faults. The experiments served as a proof-of-concept. In practice, there is no limitation of the circuit size and the number of its inputs and the scalability is determined by the ATPG algorithm only. Also, note that the proposed concept can be used with *any* SAT-based ATPG, as the ATPG algorithm modification is straightforward.

The proposed method benefits from its simplicity. The test generation can be very fast, since any SAT-based ATPG can be used, including very advanced ATPGs. Just a very slight modification must be involved, in order to accommodate the bit-flip fault model. The test application is also fast, compared to more sophisticated and more precise approaches. Particularly, the number of test vectors is similar to that for ASIC testing. Note that this data (test generation and application time) was not not studied in this paper, since we have used just our in-house ATPG, whose efficiency is far below the standard of advanced SAT-based ATPGs, in terms of the run-time, memory consumption, and the test vectors count. However, it was sufficient enough for the purposes of this paper.

Most importantly, no reconfiguration is needed, even though the application must be temporarily suspended when tested. However, this needs not be a problem in practice, because of the short test time.

Even though the method is limited to combinational circuits, it can be extended to sequential ones by using a sequential ATPG. In any case, it can be applied to quickly and efficiently test safety-critical combinational or small sequential cores of FPGA-based designs.

Acknowledgment

The authors acknowledge the support of the OP VVV MEYS funded project CZ.02.1.01/0.0/0.0/16 019/0000765 "Research Center for Informatics" and the CTU project SGS17/213/OHK3/3T/18. Computational resources were supplied by the project "e-Infrastruktura CZ" (e-INFRA

TABLE II
DETAILED RESULTS AND A COMPARISON OF FAULT MODELS

Benchmark circuit				ATPG (S@, BF)			ATPG (S@)			ATPG (BF)		
Name	Ins	Outs	ADF[1]	DF[2]	Vectors	%	DF[2]	Vectors	%	DF[2]	Vectors	%
5xp1	7	10	961	961	104	100.00	667	20	69.41	960	108	99.90
9sym	9	1	552	552	271	100.00	279	33	50.54	551	270	99.82
alcom	15	38	1838	1824	111	99.24	1726	55	93.91	1825	117	99.29
alu1	12	8	286	279	69	97.55	229	23	80.07	279	83	97.55
alu3	10	8	1030	1029	202	99.90	690	34	66.99	1030	204	100.00
apla	10	12	2139	2138	254	99.95	1680	53	78.54	2138	259	99.95
b11	8	30	1518	1517	85	99.93	1368	36	90.12	1517	89	99.93
b12	15	9	914	902	184	98.69	630	28	68.93	903	190	98.80
b2	15	17	18925	18111	1795	95.70	13353	284	70.56	17920	1809	94.69
b7	8	30	1521	1520	85	99.93	1367	36	89.88	1519	89	99.87
br1	12	8	1542	1537	225	99.68	1185	42	76.85	1538	235	99.74
br2	12	8	1363	1359	176	99.71	1042	46	76.45	1360	174	99.78
c17	5	2	81	81	23	100.00	64	9	79.01	81	24	100.00
clip	9	5	1522	1522	213	100.00	1111	48	73.00	1521	222	99.93
clpl	11	5	352	351	91	99.72	199	16	56.53	349	92	99.15
cmb	16	4	513	507	169	98.83	313	21	61.01	504	170	98.25
cu	14	11	651	647	171	99.39	466	29	71.58	647	173	99.39
dk27	9	9	1057	1053	203	99.62	713	41	67.46	1055	209	99.81
duke2	22	29	6541	6384	590	97.60	5444	122	83.23	6363	609	97.28
ex7	16	5	1073	1071	213	99.81	696	27	64.86	1070	211	99.72
f51m	8	8	728	728	125	100.00	505	22	69.37	728	130	100.00
gary	15	11	8401	8136	705	96.85	7095	172	84.45	8078	728	96.16
in2	19	10	6121	5863	746	95.79	4905	116	80.13	5834	747	95.31
intb	15	7	16930	15920	3939	94.03	11218	581	66.26	15761	4307	93.10
misex1	8	7	741	741	114	100.00	477	19	64.37	741	119	100.00
misex3c	14	14	8148	7496	1101	92.00	6001	187	73.65	7591	1103	93.16
mux	21	1	477	475	258	99.58	249	33	52.20	475	258	99.58
opa	17	61	9455	9443	419	99.87	8567	108	90.61	9442	438	99.86
pcle	19	9	714	709	164	99.30	523	28	73.25	708	166	99.16
pdc	16	40	9330	9165	886	98.23	7846	172	84.09	9161	905	98.19
pm1	16	13	803	762	99	94.89	619	32	77.09	754	101	93.90
risc	8	31	1664	1653	84	99.34	1505	37	90.44	1655	89	99.46
s1488	14	25	9932	9577	934	96.43	7382	145	74.33	9238	1002	93.01
s208	19	10	1032	1029	281	99.71	747	48	72.38	1025	283	99.32
s208_1	18	9	931	926	289	99.46	606	35	65.09	925	298	99.36
s27	7	4	171	171	65	100.00	113	9	66.08	171	67	100.00
s386	13	13	2430	2425	354	99.79	1685	56	69.34	2425	362	99.79
sao2	10	4	1964	1960	326	99.80	1257	48	64.00	1961	327	99.85
sex	9	14	974	971	123	99.69	677	26	69.51	972	124	99.79
t1	21	23	2496	2482	408	99.44	1780	69	71.31	2478	421	99.28
t3	12	8	1343	1341	243	99.85	914	42	68.06	1341	255	99.85
t481	16	1	506	506	193	100.00	302	25	59.68	501	210	99.01
ts10	22	16	2368	2308	313	97.47	1878	73	79.31	2312	323	97.64
Z5xp1	7	10	879	878	112	99.89	616	24	70.08	878	112	99.89
Z9sym	9	1	531	531	272	100.00	268	33	50.47	531	269	100.00
Summary for all 83 circuits:					24,502	99.14		4,398	69.46		25,359	99.01

[1] All detectable faults
[2] Detected faults

LM2018140) provided within the program Projects of Large Research, Development and Innovations Infrastructures.

REFERENCES

[1] J. J. Wang, "Radiation effects in FPGAs," in *9th Workshop on Electronics for LHC Experiments*, Oct. 2003, pp. 34–43.

[2] R. C. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies," *IEEE Transactions on Device and Materials Reliability*, vol. 5, no. 3, pp. 305–316, 2005.

[3] R. Velazco, P. Fouillat, and R. Reis, *Radiation Effects on Embedded Systems*. Springer Netherlands, 2007.

[4] M. Nicolaidis, *Soft Errors in Modern Electronic Systems*, ser. Frontiers in Electronic Testing. Springer Publishing Company, Incorporated, 2010.

[5] F. Bubenhagen, B. Fiethe, T. Lange, H. Michalik, and H. Michel, "Reconfigurable platforms for data processing on scientific space instruments," in *2013 NASA/ESA Conference on Adaptive Hardware and Systems (AHS-2013)*, 2013, pp. 63–70.

[6] F. Rittner, R. Glein, T. Kolb, and B. Bernard, "Broadband FPGA payload processing in a harsh radiation environment," in *2014 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2014, pp. 151–158.

[7] H. Michinishi, T. Yokohira, T. Okamoto, T. Inoue, and H. Fujiwara, "A test methodology for interconnect structures of LUT-based FPGAs," in *The Fifth Asian Test Symposium*, Nov 1996, pp. 68–74.

[8] W. K. Huang, F. J. Meyer, N. Park, and F. Lombardi, "Testing memory modules in SRAM-based configurable FPGAs," in *International Workshop on Memory Technology, Design and Testing*, August 1997, pp. 79–86.

[9] C. Stroud, S. Wijesuriya, C. Hamilton, and M. Abramovici, "Built-in self-test of FPGA interconnect," in *International Test Conference*, Oct 1998, pp. 404–411.

[10] M. Renovell, J. Figueras, and Y. Zorian, "Test of RAM-based FPGA: methodology and application to the interconnect," in *15th IEEE VLSI Test Symposium*, Apr 1997, pp. 230–237.

[11] M. Renovell, J. Portal, J. Figuras, and Y. Zorian, "Minimizing the number of test configurations for different FPGA families," in *8th Asian Test Symposium (ATS'99)*, 1999, pp. 363–368.

[12] E. Chmelar, "Minimizing the number of test configurations for FPGAs," in *IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, Nov 2004, pp. 899–902.

[13] J. Smith, T. Xia, and C. Stroud, "An automated BIST architecture for testing and diagnosing FPGA interconnect faults," *Journal of Electronic Testing: Theory and Applications*, vol. 22, no. 3, p. 239–253, June 2006.

[14] M. Renovell, M. Portal, J., P. Faure, J. Figueras, and Y. Zorian, "Analyzing the test generation problem for an application-oriented test of FPGAs," in *IEEE European Test Workshop*, May 2000, pp. 75–80.

[15] M. Rebaudengo, S. Reorda, Matteo, and M. Violante, "A new functional fault model for FPGA application-oriented testing," in *17th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, 2002, pp. 372–380.

[16] M. Rozkovec, J. Jeníček, and O. Novák, "Application dependent FPGA testing method," in *13th Euromicro Conference on Digital System Design: Architectures, Methods and Tools (DSD)*, Sept 2010, pp. 525–530.

[17] M. Tahoori, E. McCluskey, M. Renovell, and P. Faure, "A multi-configuration strategy for an application dependent testing of FPGAs," in *22nd IEEE VLSI Test Symposium*, April 2004, pp. 154–159.

[18] M. Renovell, "Some aspects of the test generation problem for an application-oriented test of SRAM-based FPGAs," *Journal of Circuits, Systems and Computers*, vol. 12, no. 02, pp. 143–158, Feb. 2003.

[19] M. Tahoori, "Application-dependent testing of FPGAs," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 9, pp. 1024–1033, Sept. 2006.

[20] C. Bernardeschi, L. Cassano, M. G. Cimino, and A. Domenici, "GABES: a genetic algorithm based environment for SEU testing in SRAM-FPGAs," *Journal of Systems Architecture*, vol. 59, no. 10, Part D, pp. 1243 – 1254, 2013.

[21] A. Cilardo, "New techniques and tools for application-dependent testing of FPGA-based components," *IEEE Transactions on Industrial Informatics*, vol. 11, no. 1, pp. 94–103, 2 2015.

[22] D. Cozzi, S. Korf, L. Cassano, J. Hagemeyer, A. Domenici, C. Bernardeschi, L. Sterpone, and M. Porrmann, "OLT(RE)^2: An on-line on-demand testing approach for permanent radiation effects in reconfigurable systems," *IEEE Transactions on Emerging Topics in Computing*, vol. 6, no. 4, pp. 511–523, June 2018.

[23] R. Hülle, P. Fišer, J. Schmidt, and J. Borecký, "SAT-ATPG for application-oriented FPGA testing," in *15th Biennial Baltic Electronics Conference*, Oct. 2016, pp. 83–86.

[24] S. Rezgui, J. J. Wang, E. C. Tung, B. Cronquist, and J. McCollum, "New methodologies for set characterization and mitigation in flash-based FPGAs," *IEEE Transactions on Nuclear Science*, vol. 54, no. 6, pp. 2512–2524, Dec 2007.

[25] Microsemi, "Axcelerator family FPGAs," Microsemi, Tech. Rep., March 2012.

[26] J. Borecký, M. Kohlík, P. Kubalík, and H. Kubátová, "Fault models usability study for on-line tested FPGA," in *14th Euromicro Conference on Digital System Design (DSD)*, Aug 2011, pp. 287–290.

[27] C. Bernardeschi, L. Cassano, A. Domenici, and L. Sterpone, "Accurate simulation of SEUs in the configuration memory of SRAM-based FPGAs," in *IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, Oct. 2012, pp. 115–120.

[28] ——, "ASSESS: a simulator of soft errors in the configuration memory of SRAM-based FPGAs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 9, pp. 1342–1355, Sept. 2014.

[29] ——, "UA^{2}TPG: an untestability analyzer and test pattern generator for SEUs in the configuration memory of SRAM-based FPGAs," *Integration, the VLSI Journal*, vol. 55, pp. 85–97, Sept. 2016.

[30] C. Carmichael, M. P. Caffrey, and A. E. S. Salazar, "Correcting single-event upsets through Virtex partial configuration," Xilinx Corporation, Tech. Rep. XAPP216 v1.0, Tech. Rep., 2000.

[31] M. Renovell, J. Portal, J. Figueras, and Y. Zorian, "SRAM-based FPGA's: testing the LUT/RAM modules," in *International Test Conference*, Oct 1998, pp. 1102–1111.

[32] M. D. Pulukuri and C. E. Stroud, "Built-in self-test of digital signal processors in Virtex-4 FPGAs," in *41st Southeastern Symposium on System Theory*, March 2009, pp. 34–38.

[33] M. Wegrzyn, F. Novak, A. Biasizzo, and M. Renovell, "Functional testing of processor cores in FPGA-based applications," *Computing and Informatics*, vol. 28, pp. 97–113, 2009.

[34] M. Renovell, P. Faure, J. Portal, J. Figueras, and Y. Zorian, "IS-FPGA: a new symmetric FPGA architecture with implicit scan," in *International Test Conference*, 2001, pp. 924–931.

[35] H. Fujiwara and T. Shimono, "On the acceleration of test generation algorithms," *IEEE Transactions on Computers*, vol. C-32, no. 12, pp. 1137–1144, Dec 1983.

[36] M. Schulz, E. Trischler, and T. Sarfert, "SOCRATES: a highly efficient automatic test pattern generation system," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 7, no. 1, pp. 126–137, Jan 1988.

[37] R. Ubar, L. Jurimagi, E. Orasson, G. Josifovska, and S. Oyeniran, "Double phase fault collapsing with linear complexity in digital circuits," in *Euromicro Conference on Digital System Design (DSD)*, Aug 2015, pp. 700–705.

[38] T. Larrabee, "Test pattern generation using Boolean satisfiability," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 11, no. 1, pp. 4–15, Jan. 1992.

[39] F. Flores, Paulo, C. Neto, Horácio, and P. Marques-Silva, João, "An exact solution to the minimum size test pattern problem," *ACM Transactions on Design Automation of Electronic Systems*, vol. 6, no. 4, pp. 629–644, Oct. 2001.

[40] S. Eggersglüß and R. Drechsler, "Robust algorithms for high quality test pattern generation using Boolean satisfiability," in *IEEE International Test Conference*, Nov. 2010, pp. 1 –10.

[41] S. Eggersglüß, R. Krenz-Baath, A. Glowatz, F. Hapke, and R. Drechsler, "A new SAT-based ATPG for generating highly compacted test sets," in *15th IEEE Design and Diagnostics of Electronic Circuits and Systems*, April 2012, pp. 230–235.

[42] H. Chen and J. Marques-Silva, "A two-variable model for SAT-based ATPG," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 12, pp. 1943–1956, Dec 2013.

[43] G. Tseitin, "On the complexity of derivation in propositional calculus," in *Automation of Reasoning*, ser. Symbolic Computation, J. Siekmann and G. Wrightson, Eds. Springer Berlin Heidelberg, 1983, pp. 466–483.

[44] G. L. Nazar and L. Carro, "Fast single-FPGA fault injection platform," in *2012 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, Oct 2012, pp. 152–157.

[45] I. Villalta, U. Bidarte, G. Santos, A. Matallana, and J. Jiménez, "Fault injection system for SEU emulation in Zynq SoCs," in *Design of Circuits and Integrated Systems*, Nov 2014, pp. 1–6.

[46] L. Sterpone and M. Violante, "A new partial reconfiguration-based fault-injection system to evaluate SEU effects in SRAM-based FPGAs," *IEEE Transactions on Nuclear Science*, vol. 54, no. 4, pp. 965–970, Aug 2007.

[47] J. Borecký, "Dependable systems design methods for FPGAs," Ph.D. dissertation, Faculty of Information Technology, Czech Technical University in Prague, 2015.

[48] University of California, Berkeley, "Berkeley logic interchange format (BLIF)," 2005.

[49] S. Yang, "Logic synthesis and optimization benchmarks user guide: Version 3.0," MCNC Technical Report, Tech. Rep., Jan. 1991.

[50] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," in *IEEE International Symposium on Circuits and Systems (ISCAS'89)*, May 1989, pp. 1929–1934 vol.3.