Unexcitability Analysis of SEUs Affecting the Routing Structure of SRAM-based FPGAs[*]

Cinzia Bernardeschi Luca Cassano Andrea Domenici
Luca Sterpone

November 30, 2016

Abstract

Testing SEUs in the configuration memory of SRAM-based FPGAs is very costly due to their large configuration memory, therefore it is necessary to optimize the generation of test patterns. In particular, in order to reduce the effort required of automatic test pattern generators, it is useful to identify early the unexcitable faults, i.e., those faults that cannot be excited by any combination of input signals. In this paper, the unexcitability of SEUs affecting the configuration bits controlling the routing resources of SRAM-based FPGAs is considered. Since this part of the configuration memory contains the largest number of configuration bits, its testing is particularly onerous. Faults in the routing resources are modeled considering the actual electrical behavior of the affected interconnections, thus the resulting fault model is more accurate then the classical open/short model usually considered. This paper introduces a methodology to prove the unexcitability of these faults. The methodology has been implemented in a tool based on a formal specification language (SAL) and a model checker (SAL-SMC). Results from the application of the tool to some circuits from the ITC'99 benchmark are reported.

Keywords Model Checking, SAL, Single Event Upset, SRAM-based FPGA, Untestability Analysis

1 Introduction and Related Work

SRAM-based FPGAs are increasingly being employed in safety critical applications such as avionic or space ones, where a harsh radiation environment causes a high incidence of *Single Event Upsets (SEUs)* that may corrupt the functionality of the device [1, 6]. The industrial use of electronic devices in safety-critical systems is regulated by application-related safety standards that impose strict safety requirements on the system. In particular safety standards, such as ISO 26262-5 [8], CENELEC 50129 [5], and IAEA NS-G-1.3 [7], require in-service testing activities for safety-related systems.

[*] submitted to GLSVLSI 2013.

Automatic test pattern generation (ATPG) for integrated circuits is a hard task, since in modern *Very Large Scale of Integration (VLSI)* systems the total number of faults that need to be detected may be very large. In FPGA-based systems, faults in the configuration memory must be tested, in addition to those in user resources. A number of these faults may be demonstrated to be untestable, thus reducing the effort required of ATPG tools. Moreover, demonstrating the untestability of faults in a VLSI design provides a way to assess the degree of testability of the system.

A number of works addressing various aspects of the analysis of untestability of faults in digital systems can be found in the literature. In [13] and [11] a new subclass of untestable faults, called *register enable stuck-on* is defined and a method for generating property specification language (PSL) assertions for proving the untestability of this class of faults is presented. In these papers stuck-at faults on the clock-enable signals of registers at the register transfer level (RTL) are addressed. The same authors propose in [12] a hierarchical untestability identification method. The method addresses untestable faults in functional units, such as adders and multiplexers, at the RTL level. In [16] a preprocessing method for accelerating SAT-based ATPGs by eliminating untestable faults is presented. The method takes into account the stuck-at fault model and it addresses only *easy-to-classify* untestable faults. In [10] two algorithms (FILL and FUNI) for untestability demonstration of stuck-at faults are presented. FILL identifies large subsets of illegal states in synchronous sequential circuits, and FUNI finds untestable faults that require illegal states previously found by FILL to be detected. Untestability analysis of SEUs in the configuration bits controlling logic components of SRAM-based FPGAs was proposed in [3], assuming the fault model proposed in [14], much more accurate than the stuck-at fault model.

As shown in [17], a more accurate fault model than the open/short (usually assumed for interconnection faults) has to be considered for SEUs in the configuration memory controlling routing resources of FPGA-based systems. This detailed fault model, together with the large number of configuration bits, makes the issue of fault untestability more relevant.

The present work introduces a methodology to prove the unexcitability of SEUs in the configuration memory of SRAM-based FPGAs controlling routing resources. The tool relies on the *Electrical Effects Static Analyzer* (E^2STAR) [4] tool for the identification of the effects of the considered SEUs, and on the SAL environment [2] for the modeling of netlists and for the proof of the unexcitability of faults. The tool implements the accurate fault model proposed in [17]. The main contribution of this paper consists in presenting the first automatic tool for the analysis of the unexcitability of SEUs in the routing structure, at the post place-and-route netlist level, and with an accurate model of the effects of SEUs.

The remainder of this paper is organized as follows: in Section 2 the considered fault model is presented; Section 3 introduces some background information on the E^2STAR tool and the SAL environment; Section 4 describes the proposed approach; Section 5 reports results from the application of approach to some circuits from the ITC'99 benchmark; Section 6 concludes the paper.

2 Effects of SEUs in the Routing Structure

An FPGA is an array of programmable logic blocks, interconnected through a programmable routing architecture and communicating with the output through programmable I/O pads [9]. Logic blocks may be simple combinational or sequential functions, called *soft* logic blocks, e.g., lookup tables, multiplexers and flip-flops, or more complex structures, called *hard* logic blocks, e.g., memories, adders, and micro-controllers. The routing architecture in an FPGA consists of wires of various length and programmable switches that form the desired connections among logic blocks and I/O pads. Finally, the I/O architecture is composed of I/O pads disposed along the perimeter of the device, each one implementing one or more communication standards.

The programming of an FPGA device consists in downloading a programming code, called a *bitstream*, in its configuration memory. The bitstream determines the hardware structure of the system to be implemented in the FPGA, and thus the functionality performed by the system. In particular the bitstream determines the functionalities performed by programmable logic blocks, the internal connections among logic blocks and the external connections among logic blocks and I/O pads.

Each net of a circuit implemented in an FPGA device is realized by the connection of logic modules through *Switchboxes*, i.e. programmable routing components that can be configured through *Programmable Interconnect Points (PIPs).* An SEU occurring in a configuration bit controlling a PIP may alter or interrupt the propagation of one or more signals to the logic block at the downstream of the effect.

To show the model of SEUs [17], let us consider the original interconnection illustrated in Figure 1 that provides the implementation of two different routing nets using the two PIPS $A \rightarrow B$ and $C \rightarrow D$, respectively. Depending on the position and the electrical properties of the affected PIP, a SEU in the routing structure can cause the following topological modifications (also shown in Figure 1): (i) *Open*, where the PIP is not programmed any more and thus a the corresponding routing segment ($A \rightarrow B$) is deleted; (ii) *antenna*, where a new routing segment (unused $\rightarrow B$) is added between an unused input node and a used output node; (iii) *conflict*, where a new routing segment ($A \rightarrow D$) is added between a used input node and a used output node; and (iv) *bridge*, where an existing routing segment is deleted ($C \rightarrow D$) and a new routing segment ($A \rightarrow D$) is added between a used input node and the output node of the deleted routing segment. Effects of SEUs in PIPs involving unused connections [17] are not considered as they do not cause a faulty behaviour.

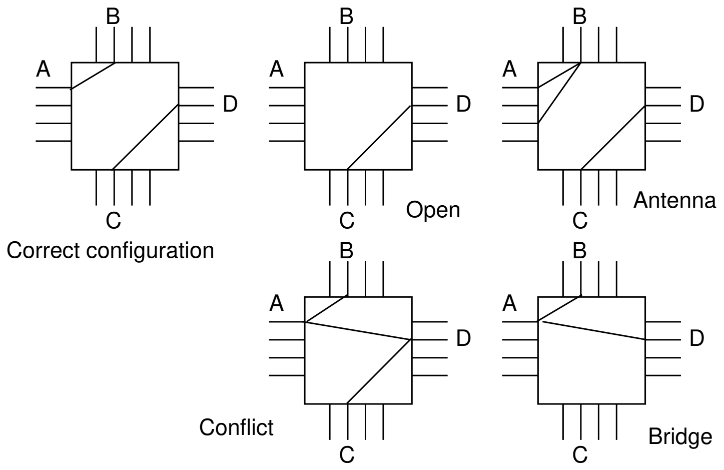

Figure 1: Effects of SEUs in the routing components of an FPGA.

3 Background

In this section we provide background information about E^2STAR [4] and the SAL environment [2].

3.1 The E^{2}STAR Tool

The purpose of the E^2STAR tool is to provide an accurate model of SEU effects in the FPGA configuration memory. E^2STAR not only reports information about the routing topology modifications induced by an SEU, but it also determines the generated electrical effect and the corresponding propagation points in the routing and logic resources used by the circuit mapped on the FPGA.

In order to characterize the logical effects induced by routing faults, fault injection experiments were carried out on a physical prototype of an FPGA application, and the corresponding outputs were observed. Then, logical faults were simulated in a behavioral model of the same application, and by comparing the outputs of the prototype with those of the simulation it was possible to map routing faults to logical effects [17].

For each given configuration memory bit the E^2STAR tool is able to find the routing segments through which an SEU in the configuration bit is propagated and the logic effects that the SEU will have in the various propagation routing segments. The following logical effects associated with the routing faults shown in Fig. 1 have been detected:

- Stuck-at: A node is stuck at a constant logic value.
- Bridge: Two nodes exchange their values.

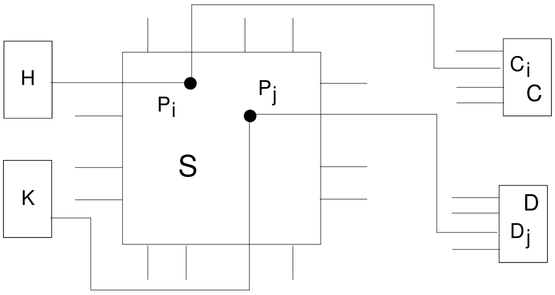

Figure 2: Routing example.

- Wired-AND (Wired-OR): The value of a node C is the AND (OR) of the values of two nodes A and B.
- Wired-MIX: The values of two nodes A and B are mixed as follows: If the values A and B are equal, A and B keep their correct values, otherwise A takes the zero logic value and B takes the one logic value.

With reference to Fig. 2, if S is a switch box, C and D are two components directly connected to S, C_i and D_j are the input pins of C and D connected to S through the PIPs P_i and P_j, respectively, the five possible effects of an SEU in the configuration bit controlling P are modeled as follows:

- A stuck-at on P_i is modeled by setting the logic signal on C_i at the corresponding fixed value.
- A bridge between P_i and P_j is modeled by exchanging the logic values on C_i and D_j.
- A Wired-AND (Wired-OR) between P_i and P_j is modeled by setting the logic signals on C_i and on D_j to the AND (OR) of the outputs of P_i and P_j.
- A Wired-MIX between P_i and P_j is modeled by setting the logic signals on C_i to 1 and on D_j to 0 if the output of P_i and P_j are different, while leaving C_i and D_j unaltered otherwise.

It may be observed that a given SEU in the configuration bit associated with a PIP can propagate to different routing segments, and that the same SEU can have different effects on the routing segments through which it propagates.

3.2 The SAL Environment

The Symbolic Analysis Laboratory (SAL) is a framework for combining different tools for the specification and verification of concurrent systems [2].

The SAL language is a strongly-typed description language. Supported types are Booleans, scalars, integers and integer subranges, records, arrays, and abstract datatypes. Expressions consist of constants, variables, applications of Boolean, arithmetic, bit-vector operations, array and record selection and update. New types can be defined and conditional expressions and user-defined functions are supported.

The basic concept in the SAL specification language is the `Module`. A SAL module is a self-contained specification of a transition system. A module consists of a `State` and an `Initialization` on the state and a list of `Transitions` on the state. The state is defined by four disjoint sets of `Input`, `Output`, `Global`, and `Local` variables. The input and global variables are *observed*, i.e., they can only be read. The output and local variables are *controlled*, i.e., they can be both read and written. Each SAL variable has two values, the *current* value (denoted, e.g., by `x`) and the *next* value (denoted, e.g., by `x'`), valid in the current and the next state (respectively) of the module.

The initialization is used to specify an initial value for all or some of the controlled state variables of the module. Like initializations, a `Definition` is a simple assignment of a value to a controlled variable. Transitions are assignments of a value to a next-state variable. A guarded command is composed of a guard, i.e., a Boolean condition defined on state variables, and one or more transitions. The transitions can be performed only if the guard is satisfied.

The SAL Symbolic Model Checker (SAL-SMC) uses LTL (Linear Temporal Logic) as an assertion language [15]. LTL formulas state properties about each linear path induced by a transition system. Typical LTL operators are:

- $\mathbf{G}(p)$ states that p is always true.
- $\mathbf{F}(p)$ states that p will eventually be true.
- $\mathbf{U}(p, q)$ states that p is true until a state is reached where q is true.
- $\mathbf{X}(p)$ states that p is true in the next state.

For a formal definition of LTL see [15]. Typical properties expressed with LTL formulas are *safety*, in the form $\mathbf{G}(\neg\chi)$, stating that the undesired condition χ is never satisfied, and *liveness*, in the form $\mathbf{G}(\mathbf{F}(\psi))$ stating that the desired condition ψ will eventually be satisfied. An expression of the form $M \vdash F$ means that a SAL specification M is a *model* of the LTL formula F, i.e., property F holds for the system specified by M.

4 The Proposed Approach

The language provided by SAL has been used to model FPGA-based systems starting from a description of the circuit at the netlist level before the place-and-route phase. Each netlist is described by a SAL `MODULE`. Inputs and outputs of the system are modeled by SAL `Input` and `Output` variables. Each component

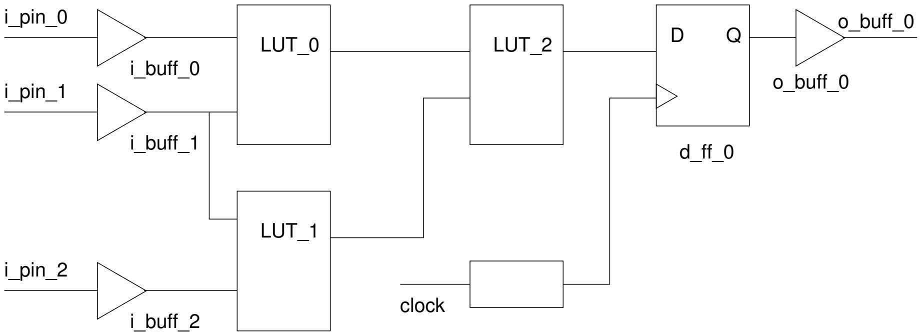

Figure 3: An example netlist.

in the netlist is modeled as a SAL `Local` variable that represents the output of the component itself.

The behavior of each component except flip-flops is described by `Definitions`. The behavior of an input buffer can be described as an assignment between a local variable, modeling the buffer, and an input variable, modeling the associated input pin. Similarly, the behavior of an output buffer can be described as an assignment between two local variables, one modeling the buffer and the other modeling the connected component. The behavior of LUTs is described by the corresponding logic functions. To show how we modeled the behavior of components we refer to the simple example of netlist shown in Fig. 3, where LUT_0 implements the OR function, while LUT_1 and LUT_2 implement the AND function. Examples of SAL models for input and output buffers are shown below:

```
i_buff_0 = i_pin_0;
o_buff_0 = d_ff_0;
```

The lookup tables of the circuit can be modeled as follows:

```
LUT_0 = i_buff_0 OR i_buff_1;
LUT_1 = i_buff_0 AND i_buff_1;
LUT_2 = LUT_0 AND LUT_1;
```

Multiplexers can be described by a conditional clause, as follows (`s` represents the select signal for the multiplexer):

```
y IN IF(s = FALSE) THEN {x1} ELSE {x2} ENDIF;
```

Flip-flops are described by `Transitions`. D-flip-flops can be described as a simple assignment between the next value of the flip-flop and the current value of its input:

```
d_ff_0' = LUT_2;
```

Other types of flip-flops (FDC, FDP, FDCE and FDPE) can be described by a conditional clause.

4.1 Unexcitability Theorems for SEUs in the Routing Structure

The analysis carried out by the E^2STAR tool makes it possible to model the effects of SEUs in the routing structure as observed at the netlist (functional) level, instead of modeling them explicitly, i.e., at the level of the routing structure itself.

Proving that a routing fault is unexcitable means proving that the signal propagated by a given faulty routing line is always the same as the signal that the line would have propagated if no fault had occurred. More precisely, with reference to Figure 2, let P_i and P_j be two PIPs in a switchbox, H and C two components connected through P_i, with signals going from H (*source*) to C (*destination*), and let K and D be two components similarly connected through P_j. According to the caused logical effect, an SEU s in the configuration bits controlling P_i and P_j is unexcitable if:

- stuck-at 0 in P_i: s is unexcitable if the output of H is always 0.
- stuck-at 1 in P_i: s is unexcitable if the output of H is always 1.
- bridge between P_i and P_j: s is unexcitable if the output of H always equals the output of K.
- wired-AND between P_i and P_j: s is unexcitable if (i) the output of H always equals H AND K and (ii) the output of K always equals H AND K.
- wired-OR between P_i and P_j: s is unexcitable if (i) the output of H always equals H OR K and (ii) the output of K always equals H OR K.
- wired-MIX between P_i and P_j: s is unexcitable if (i) the output of H always equals the output of K or (ii) the output of H is always 1 and the output of K is always 0.

Further, let $\hat{H}$ and $\hat{K}$ be the SAL variables modeling H and K respectively. The unexcitability theorems associated with the six possible logical effects of an SEU in the configuration bits controlling P_i and P_j are shown in Table 1 (see the discussion about logical effects in Section 3.1). Note that unexcitability theorems are safety theorems of the form $\mathbf{G}(\neg(fault\_excitation\_condition))$.

The table shows that in order to prove a given fault as unexcitable, we try to prove that the components feeding the putatively affected components through a switchbox will never produce the combinations of values that excite the fault.

It may be observed that a given SEU in the configuration bit associated with a PIP can propagate to different routing segments, and that the same SEU can have different logical effects on the routing segments through which it propagates. In such a case the unexcitability theorem associated with the SEU will be composed of the disjunction of a number of unexcitability lemmas, each associated with one of the propagation segments of the SEU itself, since an SEU

Table 1: Unexcitability theorems for routing faults.

s-a-0 on P_i	$C \vdash \mathbf{G}(\neg(\hat{H}))$
s-a-1 on P_i	$C \vdash \mathbf{G}(\neg(\neg\hat{H}))$
bridge between P_i and P_j	$C \vdash \mathbf{G}(\neg(\hat{H} \neq \hat{K}))$
Wired-AND between P_i and P_j	$C \vdash \mathbf{G}(\neg((\hat{H} \neq (\hat{H} \wedge \hat{K})) \vee (\hat{K} \neq (\hat{H} \wedge \hat{K}))))$
Wired-OR between P_i and P_j	$C \vdash \mathbf{G}(\neg((\hat{H} \neq (\hat{H} \vee \hat{K})) \vee (\hat{K} \neq (\hat{H} \vee \hat{K}))))$
Wired-MIX between P_i and P_j	$C \vdash \mathbf{G}(\neg((\hat{H} \neq \hat{K}) \wedge (\neg\hat{H} \vee \hat{K})))$

with multiple propagation segments is excited if and only if at least one of the faulty propagation segments is excited.

As an example, let us consider a PIP P. Let Y_1, Y_2, Y_3 and Y_4 be four destination components connected to P. Let us suppose that an SEU F in P propagates as a stuck-at-0 towards Y_1, as a stuck-at-1 towards Y_2, and as a wired-mix between Y_3 and Y_4. Let X_1, X_2, X_3, and X_4 be the netlist components generating the signals directed to Y_1, Y_4, Y_3, and Y_4, respectively, and let $\hat{X}_1$ $\hat{X}_2$, $\hat{X}_3$, and $\hat{X}_4$ be the corresponding SAL variables. The unexcitability theorem associated with F is:

$$C \vdash \mathbf{G}(\neg(\hat{X}_1 \vee (\neg\hat{X}_2) \vee ((\hat{X}_3 \neq \hat{X}_4) \wedge (\neg\hat{X}_3 \vee \hat{X}_4))))$$

The theorem above is expressed in SAL as follows:

```
F: THEOREM
  circuit |-
    G(NOT((X1=TRUE) OR
          (X2=FALSE) OR
          (X3/=X4 AND (X3=FALSE OR X4=TRUE))));
```

4.2 The Execution Flow

The overall execution flow of the proposed tool is shown in Fig. 4. The netlist description file contains an intermediate description of the netlist, generated by a parser designed for the EDIF language. The routing fault list file contains the list of the routing faults and the associated logical effects and propagation routing segments for each effect. This file is produced by the E^2STAR tool starting from the description of the netlist after the place and route phase. In this way we are able to interact with commercial CAD tools and thus the proposed tool could be integrated into the standard design process of FPGA-based systems.

The first step performed by the proposed tool is the creation of the SAL specification of the netlist under analysis starting from the netlist description file. Then, for each possible SEU in in the routing fault list an unexcitability theorem is specified. At the end of these two steps a SAL specification of the

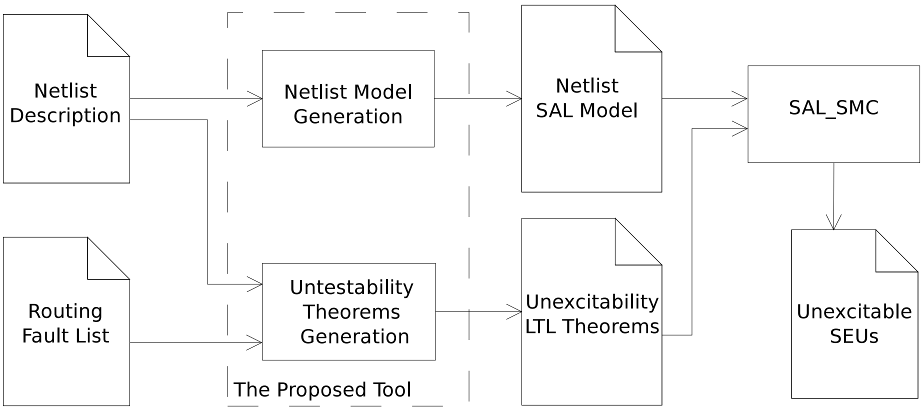

Figure 4: The execution flow of the proposed tool.

netlist and the list of the LTL unexcitability theorems associated with all the SEUs in the configuration bits controlling routing components of the system are created.

Finally the SAL-SMC model checker is invoked to prove each unexcitability theorem, one at a time. If a theorem is proved, the corresponding fault is demonstrated to be unexcitable and it is logged. These last steps are repeated until all theorems have been processed by SAL-SMC. At the end of this analysis a report file containing the number and the list of the unexcitable faults is created.

5 Experimental Results

We applied the tool to some circuits from the ITC'99 benchmark. These designs provide various test cases with circuits characterized by single clock signal, no tri-state buses or internal memories. We synthesized the VHDL code of the circuits using the Xilinx ISE CAD tool and mapping the benchmark circuits on the Xilinx Virtex-II SRAM-based FPGAs. The characteristics of the circuits in terms of name, number of Look-Up Tables (LUTs), Flip-Flop (FFs), MUXs, Input and Output buffers are shown in Table 2.

In table 3 we show the results from the application of the E^2STAR tool to the considered circuits. The table shows the number of identified critical configuration memory bits (Faults) and the number of propagation points classified by logical effects (s-a-0, s-a-1, W-AND, W-Mix, Bridge). From Table 3 it can be noticed that the number of propagation points per SEU is much higher than the actual number of SEUs. In particular the number of fault propagation points is on average 5.3 times larger than the number of faults. Moreover, as it is also discussed in [17], the largest number of effects of SEUs is stuck-at 0 and stuck-at

Table 2: Characteristics of the benchmarks.

Circuit	LUTs	FFs	MUXs	IBuffs	OBuffs
b01	5	5	0	3	2
b02	4	4	0	2	1
b03	22	30	0	5	4
b06	8	8	0	3	6
b07	135	53	15	2	8
b08	23	21	0	10	4
b09	35	29	0	2	1
b10	40	17	1	12	6
b11	77	32	9	8	6
b13	106	59	11	11	10

1.

Results of the analysis of the considered circuits with the proposed tool are shown in Table 4. The table shows the circuit name (Circuit), the total number of SEUs affecting the routing structure (RF), the number of unexcitable SEUs affecting the routing structure (RUn), and the time (in minutes) needed by the proposed tool to carry out the analysis. Moreover, the second part of Table 4 shows the percentage of unexcitable SEUs affecting the routing structure (RUn%) and compares it with the percentage of unexcitable SEUs affecting logic resources (LUn%) calculated using the tool presented in [3]. The computer used for the experiments was equipped with an Intel Core i5 (QuadCore) 2.67 GHz, 256 KB L1 Cache, 1 MB L2 Cache, 8MB L3 Cache, 4 GB RAM.

The experiments show that, in spite of the very large number of propagation points (that make SEUs much easier to excite) four out of ten circuits have a number of unexcitable SEUs in configuration bits controlling routing resources. The average unexcitability is 1.1%, with a peak of about 4.4% for b13. If we look at the second part of Table 4, we can see that SEUs affecting routing resources are much easier to excite than faults affecting logic resources. This may appear obvious, if we take two points into account: (i) the excitation of an SEU in a configuration bit controlling an LUT depends on the values of all the inputs of the LUT while, as it has been discussed in Section 4.1, the excitation of an SEU in a configuration bit controlling a PIP depends on the value of one or two signals; and (ii) as we previously discussed, each SEU in the routing structure has a very large number of propagation points. Moreover we can argue that stuck-at effects seem to be much easier to excite than wired-AND, wired-MIX and Bridge effects. In fact among the considered circuits, the ones showing a number of unexcitable SEUs are those circuits with the highest number of wired-AND, wired-MIX and Bridge effects.

The time required for the analysis ranges from a few seconds up to a few

Table 3: Effects of SEUs in the routing structure.

Circuit	Faults	s-a-0	s-a-1	W-AND	W-Mix	Bridge
b01	547	708	2,944	5	7	0
b02	304	118	339	5	7	102
b03	5,910	8,105	21,661	1,423	1,431	2,320
b06	566	372	790	0	18	305
b07	10,431	18,331	47,762	4,085	3,911	4,739
b08	2,689	3,074	8,061	464	496	1,217
b09	3,872	6,569	15,948	567	512	1,908
b10	3,942	4,603	10,727	482	692	1,498
b11	10,104	14,059	35,749	3,537	3,536	4,480
b13	7,203	10,390	27,720	1,143	1,387	3,602

minutes for very small circuits. The analysis time for medium size circuits increases to about a few hours while the required time for larger circuits is about one day. We believe that these times are reasonable if we take three points into account: (i) this analysis should be performed just once during the design of the system; (ii) by demonstrating the unexcitability of faults, the results of this analysis can reduce the time required for the generation of test patterns, thus producing an benefit for the design of the system; and (iii) the proposed analysis addresses a very accurate fault model for SEUs in the configuration bits controlling routing resources, and this makes the number of faults that have to be analysed very large.

Moreover, if we look at the unexcitability analysis from the fault tolerance point of view, we can see that the proposed tool is able to identify a subset of those SEUs to which the system is not sensitive, thus allowing an early assessment of the robustness of the system to SEUs in the configuration memory.

6 Conclusions and Future Work

We have proposed a tool for the unexcitability analysis of SEUs affecting the configuration memory controlling routing resource of SRAM-based FPGAs. The tool implements a fault model that is more accurate than the open/short model usually assumed for the analysis of interconnections in digital circuits. The application of the tool to some circuits from the ITC'99 benchmark has shown that, in spite of the large number of propagation points, also SEUs in routing components can be demonstrated to be unexcitable. Such an analysis may have a twofold benefit for system designers: on the one hand it can reduce the effort required of ATPG tools; on the other hand it can express a measure of sensitivity to SEUs of the system.

As future work we plan to analyze the unpropagability, detecting those faults

Table 4: Results from the application of the tool

Circuit	RF	RUn	T(min)	RUn%	LUn%
b01	547	0	2.45	0	0
b02	304	0	0.80	0	0.11
b03	5,910	170	219.36	2.87	48.42
b06	566	0	1.33	0	7.69
b07	10,431	385	1,863.53	3.69	31.33
b08	2,689	0	37.10	0	4.16
b09	3,872	0	93.29	0	27.45
b10	3,942	0	30.70	0	32.27
b11	10,104	13	1,347.54	0.12	21.56
b13	7,203	321	251.91	4.45	39.72

that cannot be tested because they are always masked before reaching the output of the system. Further, the counterexamples produced by SAL-SMC for excitable faults can be used to generate test patterns. We also plan to substitute the SAL model checker with an on-the-fly model checking tool, in order to improve the scalability of the tool.

References

[1] R. Baumann. Radiation-induced Soft Errors in Advanced Semiconductor Technologies. *IEEE Transactions on Device and Materials Reliability*, 5(3):305 – 316, September 2005.

[2] S. Bensalem, V. Ganesh, Y. Lakhnech, C. Muoz, S. Owre, H. Rueß, J. Rushby, V. Rusu, H. Sadi, N. Shankar, E. Singerman, and A. Tiwari. An Overview of SAL. In *Proceedings of the Fifth NASA Langley Formal Methods Workshop (LFM 2000)*, pages 187–196, 2000.

[3] C. Bernardeschi, L. Cassano, and A. Domenici. SEU-X: a SEu UnuXecitbility prover for SRAM-FPGAs. In *Proceedings of the 18th IEEE International On-Line Testing Symposium (IOLTS2012)*, June 2012.

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

[5] European Committee for Electrotechnical Standardization (CENELEC). EN 50129: Railway applications - Communications, signaling and processing systems - Safety related electronic systems for signaling, February 2003.

[6] P. Graham, M. Caffrey, J. Zimmerman, D. E. Johnson, P. Sundararajan, and C. Patterson. Consequences and Categories of SRAM FPGA Configuration SEUs. In *Proceedings of the 6th Military and Aerospace Applications of Programmable Logic Devices (MAPLD'03)*, September 2003.

[7] International Atomic Energy Agency (IAEA). NS-G-1.3: Instrumentation and Control Systems Important to Safety in Nuclear Power Plants, 2002. IAEA Safety Standards Series.

[8] International Organization for Standardization (ISO). 26262-5: Road vehicles - Functional safety - Part 5. Product development: hardware level, December 2009. Draft.

[9] I. Kuon, R. Tessier, and J. Rose. Fpga architecture: Survey and challenges. *Foundations and Trends Electronic Design Automation*, 2(2):135–253, Feb. 2008.

[10] D. Long, M. Iyer, and M. Abramovici. FILL and FUNI: Algorithms to Identify Illegal States and Sequential Untestable Faults. *ACM Transaction on Design Automation of Electronic Systems*, 5(3):632–657, 2000.

[11] J. Raik, H. Fujiwara, R. Ubar, and A. Krivenko. Untestable Fault Identification in Sequential Circuits Using Model-Checking. In *Proceedings of the 17th Asian Test Symposium (ATS'08)*, pages 21–26, 2008.

[12] J. Raik, A. Rannaste, M. Jenihhin, T. Viilukas, R. Ubar, and H. Fujiwara. Constraint-Based Hierarchical Untestability Identification for Synchronous Sequential Circuits. In *Proceedings of the 16th European Test Symposium (ETS'11)*, pages 147–152, 2011.

[13] J. Raik, R. Ubar, A. Krivenko, and M. Kruus. Hierarchical Identification of Untestable Faults in Sequential Circuits. In *Proceedings of the 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools (DSD'07)*, 2007.

[14] M. Rebaudengo, M. Sonza Reorda, and M. Violante. A new functional fault model for FPGA application-oriented testing. In *Proceedings of the 17th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT 2002)*, pages 372 – 380, 2002.

[15] K. Rozier. Linear Temporal Logic Symbolic Model Checking. *Computer Science Review*, 5(2):163 – 203, 2011.

[16] D. Tille and R. Drechsler. A Fast Untestability Proof for SAT-based ATPG. In *Proceedings of the 12th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS'09)*, pages 38–43, 2009.

[17] M. Violante, N. Battezzati, and L. Sterpone. *Reconfigurable Field Programmable Gate Arrays for Mission-Critical Applications*. Springer Science & Business Media, 2011.