Failure Probability of SRAM-FPGA Systems with Stochastic Activity Networks

Cinzia Bernardeschi, Luca Cassano and Andrea Domenici
Department of Information Engineering, University of Pisa, Italy
Email: {cinzia.bernardeschi, luca.cassano, andrea.domenici}@ing.unipi.it

***Abstract*—We describe a simulation-based fault injection technique for calculating the probability of failures caused by SEUs in the configuration memory of SRAM-FPGA systems. Our approach relies on a model of FPGA netlists realised with the Stochastic Activity Networks (SAN) formalism.**

We validate our method by reproducing the results presented in other studies for some representative combinatorial circuits, and we explore the applicability of the proposed technique by analysing the actual implementation of a circuit for the generation of Cyclic Redundancy Check codes.

***Index Terms*—SRAM-FPGA; Failure Probability; Single Event Upset; Simulation.**

I. Introduction and Related Works

Radiations in the atmosphere are responsible for introducing *Single Event Upsets* (SEU) in digital devices [1]. SEUs have particularly adverse effects on FPGAs using SRAM technology, as they may alter a bit in the configuration memory, causing a permanent error [2].

Programming an SRAM-FPGA device consists in downloading a programming code, called a *bitstream*, in its configuration memory, that determines the internal interconnections and the programmable logic block functions of the system to be implemented in the FPGA. Interconnections are realized internally by routing switches and by *I/O buffers* between input or output pins and the logic blocks. The commonest programmable logic blocks are *lookup tables* (LUT), small memories whose contents are defined by configuration bits. SEUs in the configuration memory of an SRAM FPGA can change the structure, and thus the functionality, of the system implemented in the device.

Some authors (e.g.,[3]) classify reliability analysis techniques for FPGA-based systems in three categories: radiation testing, analytical models and fault injection techniques. Radiation testing [4], [5] aims at measuring the reliability of a physical prototype of the circuit by exposing it to radiation fluxes. Analytical models [3], [6], [7], [8] use mathematical formalism, such as Binary Decision Diagrams or Probabilistic Transfer Matrices, to realize a model of the system and then analyse its reliability. Fault injection techniques use a prototype [9] or a simulated model [10], [11] of the system to inject faults and find if they are propagated to the output or not.

In this paper we present a simulation method for failure probability analysis, based on the SAN formalism [12] and developed with the Möbius tool [13]. Our approach relies on the model of systems implemented on SRAM-FPGAs presented in [14] for signal probability analysis. This model is based on the netlist-level representation of an FPGA system produced in the synthesis phase, before the place and route phase. At this level, the elements visible in the model are I/O buffers, LUTs, flip-flops, and multiplexers.

We extended this model introducing an accurate model of single and multiple event upsets that may affect the configuration memory of the FPGA chip at any clock cycle during the simulation. This method aims at providing accuracy, flexibility, and controllability. We point out that the main objective of this method (and associated tool) is studying the system failure probability depending on where in the circuit SEUs occur, on the basis of input signal probabilities and SEU probabilities whose values are assumed to be available from other sources.

The remainder of this paper is organized as follows: In Section II the SAN formalism and the Möbius tool are briefly illustrated and then a model of fault-free FPGA-based systems is presented; in Section III the adopted fault model, the SAN model for fault simulation and the simulation setup are presented; Section IV reports on the application of the approach to a simple FPGA-based system; Section V concludes the paper.

II. The FPGA Model

We introduce the main characteristics of the SAN formalism and we briefly illustrate the fault-free FPGA model, referring to [14] for a more detailed discussion.

A. The SAN Formalism

Stochastic Activity Networks [12] are an extension of the Petri Nets (PN) formalism. SANs are directed graphs with four disjoint sets of nodes: *places*, *input gates*, *output gates*, and *activities*. The topology of a SAN is defined by its input and output gates and by two functions that map input gates to activities and pairs (*activity*, *case*) (see below) to output gates, respectively. Each input (output) gate has a set of *input* (*output*) places. The activities replace and extend the *transitions* of the PN formalism. Each SAN activity may be either *instantaneous* or *timed*. The duration of each timed activity is expressed via a *time distribution* function. Any instantaneous or timed activity may have mutually exclusive outcomes, called *cases*, chosen probabilistically according to the *case distribution* of the activity. As in PNs, the state of a SAN is defined by its *marking*. The marking of each place is a non-negative integer called its *number of tokens* or, in *extended places*, a real value or a complex data structure.

SANs enable the user to specify any desired enabling condition and firing rule for each activity. This is accomplished by associating an *enabling predicate* and an *input function* to each input gate, and an *output function* to each output gate. The enabling predicate is a Boolean function of the marking of the gate's input places. The input and output functions compute the next marking of the input and output places, respectively, given their current marking. If these predicates and functions are not specified, the standard PN rules are assumed.

Graphically, places are drawn as circles, input (output) gates as left-pointing (right-pointing) triangles, instantaneous activities as narrow vertical bars, and timed activities as thick vertical bars. Cases are drawn as small circles on the right side of activities. Gates with default (standard PN) enabling predicates and firing rules are not shown.

A popular software tool to build and analyze SAN models is Möbius [13], that provides a framework for model-based evaluation of system dependability and performance. In this tool, properties of interest are specified with *reward functions* [15]. A reward function specifies how to measure a property on the basis of the SAN marking. Evaluation of reward functions can be made at specific time instants, over periods of time, or when the system reaches a steady state. A desired confidence level is associated to each reward function. At the end of a simulation the Möbius tool is able to evaluate for each reward function whether the desired confidence level has been attained or not, thus ensuring a high accuracy of the results.

B. Description of the Model

The FPGA model is split into a number of modules: *System Manager*, *Input Vector*, *Combinatorial Logic* and *Sequential Logic*. Each module is a SAN and communicates with the other modules through *shared places*, a Möbius extension to SANs that enables communication among subsystems.

The System Manager module orchestrates the activity of the other modules of the system according to the following steps: (i) an *input vector*, i.e., an n-tuple of the input signal values, is applied to the input lines; (ii) the combinatorial part of the system is executed; (iii) a clock tick arrives and the sequential part of the system is executed. These steps are repeated until all input vectors have been applied.

The Input Vector module generates an input vector that is applied to the input lines of the FPGA.

The Combinatorial Logic module models the combinatorial part of the system, i.e., lookup tables, multiplexers, and input/output buffers. The Sequential Logic module models the flip-flops. Various types of flip-flops can be modeled besides the basic D-edge-triggered. Combinatorial and sequential elements are represented as a SAN model, called `Generic_Component` (see Figure 1). One replica of the `Generic_Component` is generated for each element. Places `spA` and `spB` are used to control the execution of a component. The output gate `OG0` implements the functionality of the component. When the `execute` activity of a component completes, the function specified in gate `OG0` is executed, and a token is added to `spB`.

Three shared extended places (`input_lines`, `output_lines`, and `internal_lines`) encode the value of the signals on the input, output, and internal connections of the FPGA.

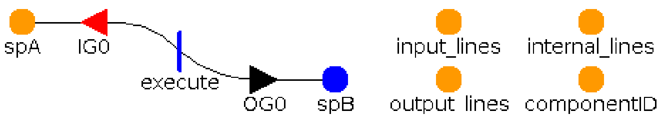

Fig. 1. Generic_Component module.

The logical connections are specified in a *connectivity matrix*, a data structure accessed by the input and output functions of the model. This way, the connections are not hardwired in the SAN models, and can be set up from a text file generated with CAD tools, such as the Xilinx ISE tool [16], on the basis of the specification of the FPGA-based system.

As discussed in [14], the proposed model can deal with circuits whose combinatorial part has reconvergent fanouts, with an accuracy that depends only on the number of simulation batches, whereas other methods either ignore reconvergent fanouts or consider a limited number of them.

III. Fault Model and Simulation

In this work we take into account single and multiple event upsets in the configuration memory of an FPGA-based system described at netlist level, considering LUTs and input and output buffers. SEUs may occur at any clock cycle in the simulation. In our fault model a SEU in the configuration bit of an input/output buffer causes the buffer to hold the current value of the output when the fault occurred. Thus, after a SEU occurred, a buffer becomes insensitive to further changes of the input signal. A SEU in the configuration memory of a lookup table causes the lookup table to change the implemented function. Our model is able to specify which configuration bit is faulty, and a LUT generates an incorrect output only when the input values are those associated to the faulty configuration bit. This contrasts with other simpler models where a faulty LUT is assumed to be faulty for all possible input values. In the following we will use the term *component* to indicate only LUTs and buffers, leaving out flip-flops and multiplexers.

In order to model the occurrence of event upsets in the components of the FPGA at a random clock cycle, we extended the `Generic_Component` module (Figure 1) of the model described in [14] as shown in Figure 2. The extension consists in adding an execution path to simulate behaviour in the presence of faults and a mechanism to choose probabilistically between correct or faulty execution, as explained below.

The following parameters must be specified: (i) the failure probability of LUTs, p_L; (ii) the failure probability of buffers, p_B; (iii) the maximum number of faults to be injected, F; (iv) the number of clock cycles N; and (v) the input signal probabilities P_i.

Place `faulty` specifies whether the current component is faulty or not. The `num_faults` place is shared among all

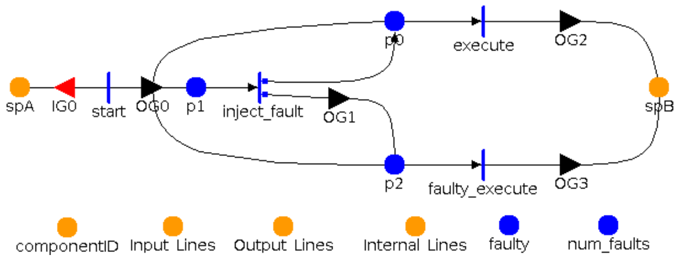

Fig. 2. The Extended_Generic_Component module.

the components of the netlist and specifies the current number of faults already occurred.

The spA place is a shared array of Booleans whose i-th element specifies whether the i-th component is currently activated. When spA[i] is true the i-th component is executed. First, start is enabled. When it completes, the output gate OG0 executes the following algorithm: (i) If faulty is true, then the current component is already faulty, and a token is placed in p2; (ii) If faulty is false and the marking of num_faults is less than F, then the current component is still unfaulty and more faults can occur in the circuit, thus a new fault could be injected and a token is placed in p1; (iii) If faulty is false and the marking of num_faults equals F, then the current component is unfaulty, no more faults can occur and a token is placed in p0.

When a token is placed in p1 the inject_fault activity becomes enabled. It is responsible for deciding whether a fault has to be injected or not in the current component. With a probability $1-p_L$ (or $1-p_B$, according to the type of component) a token is placed in p0 and the current component remains unfaulty. With a probability p_L (p_B) the inject_fault activity enables output gate OG1 and the current component is damaged. When executed, OG1 increments the marking of num_faults to specify that a new fault occurred, sets faulty to true to specify that the current component is faulty and, if the current component is a LUT, it randomly chooses which configuration bit of the LUT has to be corrupted. At the end of the execution of OG1 a token is placed in p2.

At this point if the current component is still unfaulty, the execute activity is enabled, and when it terminates, the output gate OG1 is executed. OG1 implements the normal behaviour of the component. If the current component was already faulty, or became faulty right now, the activity faulty_execute is enabled. When the activity terminates the output gate OG2 is executed. OG2 implements the failure mode of the current component that has been described at the begining of the section.

Our analysis was aimed at measuring the failure probability of an FPGA-based system, by calculating the percentage of clock cycles in which the expected output signals and the actual ones were different for at least one value. Given the high level of controllability and observability of the proposed tool, it is also possible to compute the individual failure probability of any output and internal signals of the system.

The simulations are performed according to the following

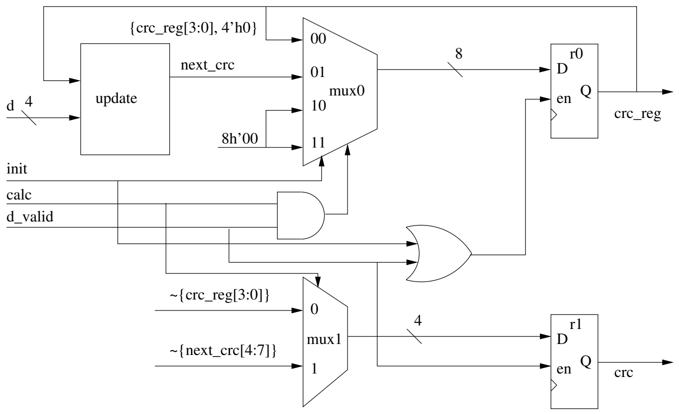

Fig. 3. A circuit to generate CRC code (adapted from [18]).

schema: (i) Two copies, called *Golden Copy* (GC) and *The Device Under Test* (DUT), of the above described FPGA model are instantiated in the simulator from the same netlist description; (ii) random input vectors are generated according to the specified signal probabilities of the input and fed to the GC and DUT; (iii) the GC executes with no faults, while the DUT injects faults according to the specified fault probabilities and maximum number of injectable faults; (iv) at each clock cycle the output values of the GC and the DUT are compared, using a reward function that returns 1 if they are different for at least one value.

IV. A Simulation Example

To validate our model, we considered combinatorial circuits presented in various related works, and we checked that we were able to reproduce the same results. The circuit failure probability computed with our model corresponds, for example, with that reported in [17]; specifically, after 10000 simulation runs, we obtained a relative error with respect to the reference results of $7 \cdot 10^{-3}$. This cross-validation exercise reinforced our confidence in the correctness of the model.

A. Analysis of a Circuit for CRC generation

As a case study, we consider the FPGA implementation of a circuit for the generation of IEEE 802.3 CRC codes [18]. A simplified schematic of a 4-bit data bus circuit that generates 8-bit CRC codes is shown in Figure 3. In the figure, d is the 4-bit data bus, init, calc, and d_valid are control signals, and update is a combinatorial network that computes the next state for the upper output register. This example is only meant to show how a system can be simulated by varying several parameters.

The Verilog code for the circuit, available from the Xilinx web site, was compiled for the Virtex 6 device into a netlist with the Xilinx ISE tool. The resulting netlist has 8 input signals, 12 output signals, 20 I/O buffers, 17 LUTs, and 19 flip-flops. The signal values during the experiments were as follows: Control pins load_init and reset are always low; calc is always high; d_valid switches between high and low levels at each clock cycle; input pins d[3:0] are high with probability P_i, where i is the index of the input pin.

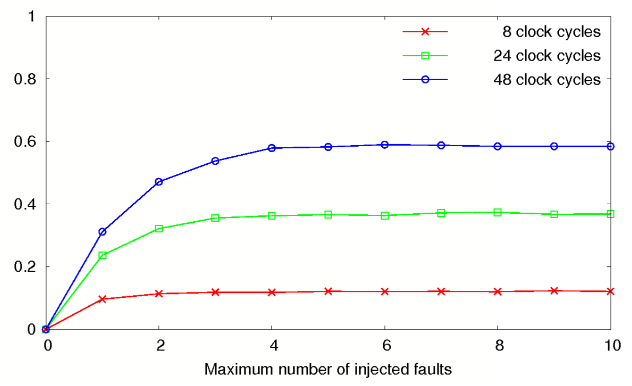

Fig. 4. Failure probability vs. maximum number of faults that can be injected.

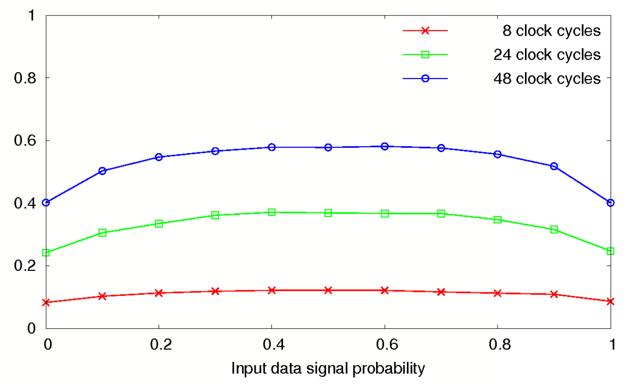

Fig. 5. Failure probability vs. input signal probability of data signals.

The following plots show some of the parameters that can be varied and the measurements that can be taken. Results were computed with a confidence level of 0.95, executing between 10^4 and 10^5 simulation runs. In a first scenario we calculated the failure probability of the system for 8, 24 and 48 clock cycles, letting F vary, setting p_L and p_B to 0.001 and P_i to 0.5. The resulting failure probability is shown in Figure 4. In a second scenario we varied the input signal probabilities, setting p_L and p_B to 0.001, and F to 5. Figure 5 shows the failure probability of the circuit calculated for 8, 24 and 48 simulated clock cycles and 5 injectable faults.

V. Conclusions and Future Work

We have described a simulation-based technique for failure probability calculation of systems implemented in SRAM-FPGA technology in presence of SEUs.

This technique has the high degree of flexibility, controllability and observability typical of simulation-based techniques. The maximum number of faults and the failure probability of each type of components can be specified, and the approach can be applied to FPGA chips regardless of the particular vendor and model.

The proposed approach also has a high accuracy that makes it possible to reproduce in detail the failure behaviour of a lookup table or a buffer affected by a SEU. Both purely combinatorial and synchronous systems can be simulated, taking into account circuits with reconvergent fanouts, which is not possible in most analytical approaches. Single or multiple event upsets can be injected during the simulation.

As further work, SEUs affecting user resources (such as flip-flops) and routing will be studied.

References

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

[2] P. Graham *et al.*, “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.

[3] L. Sterpone and M. Violante, “A new analytical approach to estimate the effects of SEUs in TMR architectures implemented through SRAM-based FPGAs,” *IEEE Transactions on Nuclear Science*, vol. 52, no. 6, pp. 2217 – 2223, December 2005.

[4] E. Fuller *et al.*, “Radiation Testing Update, SEU Mitigation, and Availability Analysis of the Virtex FPGA for Space Re-configurable Computing, presented at the IEEE Nuclear and Space Radiation Effects Conference,” in *Proceedings of the 3rd Military and Aerospace Programmable Logic Devices International Conference (MAPLD’00)*, 2000.

[5] M. Ceschia *et al.*, “Ion beam testing of ALTERA APEX FPGAs,” in *Proceedings of the IEEE Radiation Effects Data Workshop*, 2002, pp. 45 – 50.

[6] G. Asadi and M. B. Tahoori, “An Analytical Approach for Soft Error Rate Estimation of SRAM-based FPGAs,” in *Proceedings of the 7th Military and Aerospace Programmable Logic Devices International Conference (MAPLD’04)*, 2004.

[7] O. Heron *et al.*, “On the reliability evaluation of SRAM-based FPGA designs,” in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL’05)*, August 2005, pp. 403 – 408.

[8] S. Krishnaswamy *et al.*, “Accurate reliability evaluation and enhancement via probabilistic transfer matrices,” in *Proceedings of the conference on Design, automation and test in Europe (DATE ’05)*, March 2005, pp. 282 – 287 Vol. 1.

[9] 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, August 2007.

[10] M. Violante *et al.*, “Simulation-based analysis of SEU effects in SRAM-based FPGAs,” *IEEE Transactions on Nuclear Science*, vol. 51, no. 6, pp. 3354 – 3359, December 2004.

[11] P. Samudrala *et al.*, “Selective Triple Modular Redundancy (STMR) based single-event upset (SEU) tolerant synthesis for FPGAs,” *IEEE Transactions on Nuclear Science*, vol. 51, no. 5, pp. 2957 – 2969, 2004.

[12] W. H. Sanders and J. F. Meyer, “Stochastic Activity Networks: formal definitions and concepts,” *Lectures on formal methods and performance analysis: first EEF/Euro summer school on trends in computer science*, pp. 315–343, 2002.

[13] G. Clark *et al.*, “The Möbius modeling tool,” in *9th Int. Workshop on Petri Nets and Performance Models*. Aachen, Germany: IEEE Computer Society Press, September 2001, pp. 241–250.

[14] C. Bernardeschi *et al.*, “Analysis of FPGAs using the SAN Formalism,” in *Proceedings of the Mosharaka International Conference on Communications, Networking and Information Technology (MIC-CNIT2010)*, 2010, in press.

[15] W. H. Sanders and J. F. Meyer, “A unified approach for specifying measures of performance, dependability, and performability,” in *Dependable Computing for Critical Applications*, A. Avižienis *et al.*, Eds. Springer-Verlag Heidelberg, 1991, pp. 215–237.

[16] “ISE Design Suite Software Manuals and Help,” http://www.xilinx.com/support/documentation/sw_manuals, 2010.

[17] G. Asadi and M. B. Tahoori, “An Accurate SER Estimation Method Based on Propagation Probability,” in *Proceedings of the conference on Design, Automation and Test in Europe (DATE ’05)*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 306–307.

[18] C. Borrelli, “IEEE 802.3 Cyclic Redundancy Check,” application note: Virtex Series and Virtex-II Family, XAPP209 (v1.0), March 23, 2001, Xilinx, Inc.