ASSESS: A Simulator of Soft Errors in the Configuration Memory of SRAM-Based FPGAs*

Cinzia Bernardeschi, Luca Cassano, Andrea Domenici and Luca Sterpone †‡

March 30, 2019

Abstract

In this paper a simulator of soft errors (SEUs) in the configuration memory of SRAM-based FPGAs is presented. The simulator, named ASSESS, adopts fault models for SEUs affecting the configuration bits controlling both logic and routing resources that have been demonstrated to be much more accurate than classical fault models adopted by academic and industrial fault simulators currently available. The simulator permits the propagation of faulty values to be traced in the circuit, thus allowing the analysis of the faulty circuit not only by observing its output, but also by studying fault activation and error propagation. ASSESS has been applied to several designs, including the miniMIPS microprocessor, chosen as a realistic test case to evaluate the capabilities of the simulator. The ASSESS simulations has been validated comparing their results with a fault injection campaign on circuits from the ITC'99 benchmark, resulting in an average error of only 0.1%.

Keywords: Computer-aided design, reliability and testing, simulation, single event upset, SRAM FPGA.

1 Introduction

Soft errors are caused by high-energy particles that change the stored charge in memory cells of electronic devices [7]. In particular, a bit flip, or *Single Event Upset* (SEU), in a memory cell may bring the system to an erroneous state until the cell is rewritten. SEUs, therefore, are usually transient as they are likely to be overwritten at the next clock cycle, but they may have permanent, or at

*Postprint. Published in: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (Volume: 33, Issue: 9, Sept. 2014). DOI: https://doi.org/10.1109/TCAD.2014.2329419.

†C. Bernardeschi, L. Cassano and A. Domenici are with Dipartimento di Ingegneria dell'Informazione, University of Pisa.

‡Luca Sterpone is with Dipartimento di Automatica ed Informatica, Politecnico di Torino.

least long term, effects in configurable devices such as SRAM-based FPGAs, since the functions and the interconnections of these devices are controlled by a configuration memory [26]. The sensitivity to SEUs is relevant in a growing number of highly reliable FPGA applications, operating in environments where radiations can increase the likelihood of soft errors [23].

In particular, FPGAs based on SRAM technology store the configuration data in SRAM cells, which are rewritable and thus make it possible to reconfigure the device. The frequency of reconfigurations depends on the application. Some devices may never be reconfigured, as the need for reconfiguration does not arise during their operational life, while others may be reconfigured occasionally for repair or upgrades, and yet others may undergo frequent reconfigurations to change mode of operation. However, even in this last case, long periods of operation separate consecutive reconfigurations, therefore SEUs may corrupt a device's functionality over long period of times, until the reconfiguration is performed [23]. Besides, as the portion of configuration bits actually used in the design increases, the probability that a soft error will alter the logical behavior of the system becomes an issue in FPGAs using SRAM technology [21].

The main techniques for the analysis of the effects of SEUs in the configuration memory of SRAM-based FPGAs rely on accelerated radiation testing, fault injection and fault simulation. Radiation testing [9] and fault injection [36] are generally accurate and fast but they can be applied only late in the design process, when a prototype of the system is available. On the other hand, fault simulation [24] can be applied soon during the design of the system and at different levels of abstraction, but it is generally less accurate and slower. Moreover, while radiation testing and fault injection allow the analysis of the system's behavior only by observing its output, fault simulation makes it possible to study fault activation and fault propagation, giving designers a deeper insight into the circuit and the causes of system failures.

Current simulation tools and methodologies represent SEU effects on logic resources as stuck-at faults, and effects on routing elements as open, short, or bridge faults. Although these fault models can be effectively used when analyzing the effects of faults in ASIC circuits or in the user resources of an FPGA design, they have been shown not to be accurate enough for the analysis of the configuration memory in SRAM-based FPGAs, so that more accurate fault models have been proposed [29, 37]. In particular, the stuck-at model cannot describe a fault in the configuration bit of a LUT, which is activated only when the LUT inputs take the combination of values corresponding to that bit [29]. Similarly, faults in the routing configuration memory may cause a wider range of effects than open, short, or bridge, as has been discussed in [37] and briefly reported in Sec. 3.2.

The novelty of the simulation framework presented in this paper with respect to currently available academic and commercial tools is the adoption of the accurate fault model for SEUs proposed in [29, 37], leading to more accurate results than those of the other fault simulators, which, to the best of our knowledge, do not yet support this model. The additional feature of ASSESS is the use of a formal specification language, the *Stochastic Activity Networks*

(SAN) [32], supported by the Möbius tool [18], for modeling the FPGA-based system and the fault injection module. SANs are an extension of stochastic Petri nets, used to evaluate performance and dependability of complex systems [28]. The Möbius tool provides a flexible and easy to use environment to define and analyze SAN-based models, and in particular it makes it possible to specify *reward functions* that quantify dependability-related properties of the modeled system.

The ASSESS tool is highly configurable, interoperates seamlessly with the standard development process and enables designers to measure different reliability-related properties of FPGA-based applications. The FPGA applications are described at the netlist level, which shows the circuit components with their logical connections but not the physical routing. Faults affecting routing resources are interpreted as equivalent logic effects in the netlist: This is made possible by another tool, E^2STAR [14]. E^2STAR is able to identify the topological modifications induced by an SEU in the routing structure of the FPGA according to the position of the configuration bit as well as to electrical properties. Moreover, E^2STAR is able to determine the logic effects induced by these topological modifications, thus allowing ASSESS to map on the pre-place and route logic netlist the effects of the SEUs occurring in the routing structure.

Dependability-related measures that can be estimated include failure probability, i.e., the probability of a system failure in a given number of clock cycles given the probability of the occurrence of SEUs in the configuration memory of the system; and SEU observability, i.e., whether a system failure is observed, assuming a specific bit flip in the configuration memory. Moreover, ASSESS allows designers to detect the fault activation and to trace the error propagation. In this way designers can understand the causes of the failures of the system and better analyze the effectiveness of fault tolerance techniques. We point out that we do not propose fault simulation as an alternative to fault injection, but as an additional tool for designers to analyze the susceptibility of the system to faults, as early as possible during the design process, thus getting to the final fault injection or radiation experiment on the prototype with a thoroughly analyzed system.

Preliminary work from the same authors related to ASSESS are: 1) [12], where a netlist simulator without fault injection is presented; 2) [11, 10], where faults in the logic were introduced; and 3) [14] where routing faults were taken into account. With respect to the previously published papers, the current paper presents the following new technical material: 1) re-engineering of the whole simulation environment in order to make it configurable, e.g., able to change operational mode (fault-free, deterministic and stochastic fault injection) and performed measures; 2) redesign of the fault injection SAN module to support both stochastic and deterministic fault injection, and of the system manager SAN module in charge of orchestrating the simulation; 3) design of reward functions for additional dependability-related measures; 4) comparison of results obtained by ASSESS with those obtained using the traditional stuck-at fault model; 5) figures from the analysis of fault activation, error propagation and failure probability on a set of circuits from the ITC'99 benchmark; and 6)

simulation and analysis of the miniMIPS microprocessor, representing a realistic industrial case application.

The remainder of this paper is organized as follows. Section 2 discusses related work on SEU analysis and simulation; Section 3 introduces the FPGA technology and the adopted fault models; Section 4 briefly presents the SAN formalism and Möbius environment; Section 5 describes the simulation framework, the architecture of the ASSESS tool and some SAN models composing ASSESS; Section 6 discusses the dependability-related measures performed by ASSESS; a discussion about results obtained by simulating some benchmark circuits is in Section 7; conclusions and future work are in Section 8.

2 Related Work

In this section we focus on the techniques and tools specifically addressing the analysis of the effects of SEUs in SRAM-based FPGAs that can be found in the literature.

The sensitivity to SEUs of SRAM-based FPGA systems can be analyzed according to four main approaches: *accelerated radiation ground testing*, *fault injection boards*, *analytical computation*, and *fault simulation*.

Accelerated radiation ground testing [21, 9, 22, 17, 16] aims at emulating the effects of SEUs by exposing a prototype of the FPGA-based system to a flux of radiations, originated by either a radioactive source or a particle accelerator. While being exposed to the radiations flux, the prototype is fed with a set of input patterns, and its behavior is monitored. Drawbacks of these techniques are: 1) The impossibility of injecting SEUs only in the configuration memory of the FPGA, since the whole chip area will be irradiated (including logic resources); 2) a possibility that the device be permanently damaged after the experiment; and 3) high cost.

Several fault injection boards have been developed in order to evaluate the impact of SEUs in the configuration memory of circuits mapped on SRAM-based FPGAs [36, 38, 4, 2]. These boards emulate the occurrence of SEUs by modifying the bitstream of the target system whose dynamic behavior is then evaluated. Fault injection can be performed either before downloading the bitstream on the device under test, or at run time exploiting partial dynamic reconfiguration. Unlike radiation testing experiments, fault injection makes it possible to focus the analysis on SEUs in the configuration memory of the FPGA, leaving out any other resources. Moreover, fault injection avoids the risk of damaging the device under analysis. The major drawbacks of SEU injection boards are high costs, complex usability, and strong chip and vendor dependence. Both fault injection and radiation testing also have the drawback that they need a physical prototype of the system.

Analytical approaches are reported in [34, 35, 5, 25]. In [34] and [35], a model based on the structure of the design implemented on the FPGA is built, and the topological modifications induced by SEUs in each configuration bit are deduced, thus discovering which SEUs affect the design. In [5], sensitive

paths to SEUs are identified by combining the error probability of all nodes of the circuit with the error propagation probability of each path of the circuit. Finally, in [25] an accurate probabilistic model to estimate the reliability of SRAM-based FPGA system is presented. Given the probability of occurrence of an SEU, the model is able to estimate the probability of having a system failure after a given amount of time. The drawback of these approaches is that, since the analysis is carried out without respect to the input patterns fed into the system, they are able to provide a worst case analysis but they cannot give information about the behavior of the system in its normal operating conditions.

Although a large number of fault simulators for digital circuits can be found in the literature, very few simulation approaches targeting the analysis of the effects of SEUs have been proposed. Moreover, if we look for simulators that specifically address the FPGA technology we find an even smaller number of works. In [33, 15], two simulators of SEUs affecting digital circuits have been proposed. Both simulators work at the gate-level representation of the circuit, thus ensuring accurate results, but neither one takes into account any details specific of the FPGA technology. To the best of our knowledge, the only simulator targeting SEUs in FPGAs is *SST* [24]. SST is a set of TCL scripts able to modify the HDL description of the circuit in order to emulate the effects of SEUs, and then to interact with standard Register Transfer Level (RTL) simulators, such as ModelSim [27]. Since SST works on the RTL representation of the system, it can only emulate the effects of SEUs in logic resources, e.g., flip-flops and memories, but it is not able to reproduce the effects of SEUs in the configuration memory.

3 Effects of SEUs in the Configuration Memory of SRAM-based FPGAs

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 [26].

The basic logic element may have several different structures, which in many cases are variations of the pattern shown in Figure 1: The logic block has n m-input LUTs implementing the required functions, a carry chain logic that can be used to implement arithmetic functions, a number of multiplexers that allow selecting among the outputs of the LUTs and those of the carry chain logic, and a number of flip-flops.

A common architectural pattern is based on a regular layout where conductive *tracks* (or *wires*) are grouped into *channels* running in two orthogonal directions and forming a grid whose meshes enclose the logic blocks. The areas where vertical and horizontal channels cross are called *switch boxes* and contain the programmable circuitry that connects tracks of one channel to tracks of the same or the other channel.

Figure 2 shows an overall representation of an FPGA architecture. More

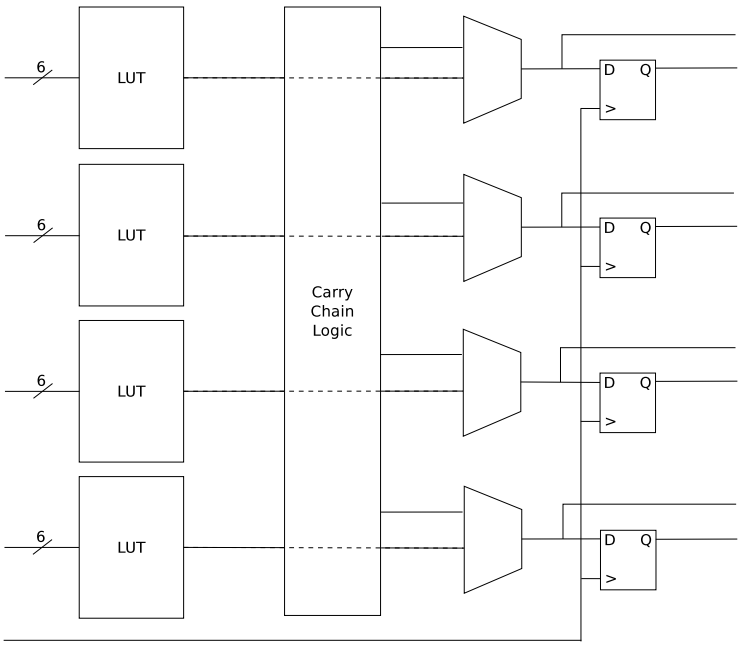


Figure 1: Basic logic block structure.

in detail, the figure shows the regular structure of the logic blocks, the routing architecture, and the configuration memory. The configuration memory bits control and program both logic and routing resources of the FPGA.

After an FPGA chip has been selected, a tool synthesizes the RTL description of a system into a *netlist*, choosing the logic blocks to implement the RTL design and defining their logical connections; then the *place and route* phase defines the location of the various components of the netlist on the target device and then their physical interconnections; and finally the bitstream is generated from the placed and routed netlist and downloaded into the configuration memory of the chip.

The bitstream determines the functions of the programmable logic blocks, the internal connections among logic blocks and the external connections among logic blocks and I/O pads.

3.1 Logical Faults

ASSESS adopts the model for SEUs affecting the configuration memory of LUTs presented in [29]. A LUT affected by an SEU in its configuration memory will produce an incorrect value only when the pattern of its input values is the one associated with the faulty configuration bit, while for every other input pattern the faulty LUT will behave correctly. Figure 3(a) shows an SEU causing a bit flip in the configuration bit associated with input (0000). In this case,

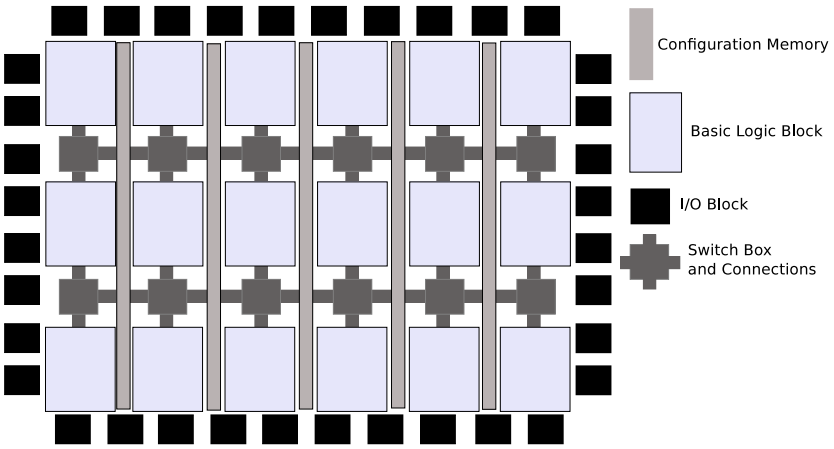
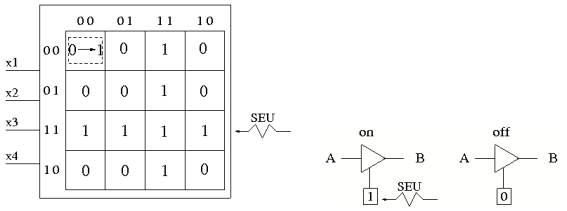


Figure 2: Overall structure of an FPGA device.



(a) SEU in a lookup table. (b) SEU in an I/O buffer.

Figure 3: Effects of SEUs in the logic components of an FPGA.

the logic function implemented by the LUT changes from the correct function $y = x_1 \cdot x_2 + x_3 \cdot x_4$ to the faulty function $y_f = x_1 \cdot x_2 + x_3 \cdot x_4 + \bar{x_1} \cdot \bar{x_2} \cdot \bar{x_3} \cdot \bar{x_4}$. The results of y and y_f are different only when the values of the input signals are (0000). This behavior cannot be simulated with the stuck-at fault model, which does not "look inside" the LUT, and thus assumes that the output is fixed at a value V that is faulty for all inputs such that $y \neq \mathrm{V}$.

An SEU in the configuration bit of an I/O buffer causes an undesired connection or disconnection between two wires. Fig. 3(b) shows a 1 to 0 bit flip causing a disconnection between point A and B.

3.2 Routing Faults

An extensive treatment of the fault model for the effects of SEUs in the routing structure that is assumed in this paper is found in [37].

In order to discuss the effects of SEUs in the interconnect, let us first consider the organization of the switch boxes. *Programmable interconnect points* (PIPs) are used to connect the input and output ports of switch boxes, providing the routing paths to connect the I/O buffers and logic components of the

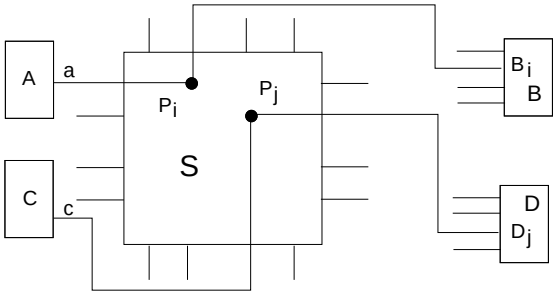


Figure 4: An example of a switch box.

FPGA circuit. A PIP may consist in a single pass transistor controlled by one configuration bit, or in more complex structures controlled by groups of bits. The various structures accomplish different functions, such as connecting two wires in the same channel, or two wires in orthogonal channels, or establishing multiple connections. In particular, even if an SEU modifies only one switch box configuration bit, single and multiple effects may be originated for PIPs of the switch box.

Depending on the position and the electrical properties of the affected PIPs, an SEU in the routing structure can cause the following topological modifications.

- Open. The PIP corresponding to a net is not programmed anymore.
- Input antenna. A new PIP is added between an unused input node and a used output node. The new PIP can influence the behavior of the output node.
- Output antenna. A new PIP is added between a used input node and an unused output node. The new PIP does not influence the behavior of the implemented circuit.
- Conflict. A new PIP is added between an input node and an already used output node.
- Bridge. A PIP is disabled while a new PIP is instantiated between a used input node and the output nodes used by the previous net.

By comparing the results of electrical fault injection with behavioral simulations, it has been possible to identify the logical effects of the topological modifications induced by SEUs into the routing resources, thus defining an equivalent logical model associated with such SEUs.

With reference to Fig. 4, if S is a switch box, B and D are two components directly connected to S, B_i and D_j are the input pins of B and D connected to S through the PIPs P_i and P_j, respectively, the five possible logical effects of an SEU in the configuration bit controlling the two PIPs are described as follows.

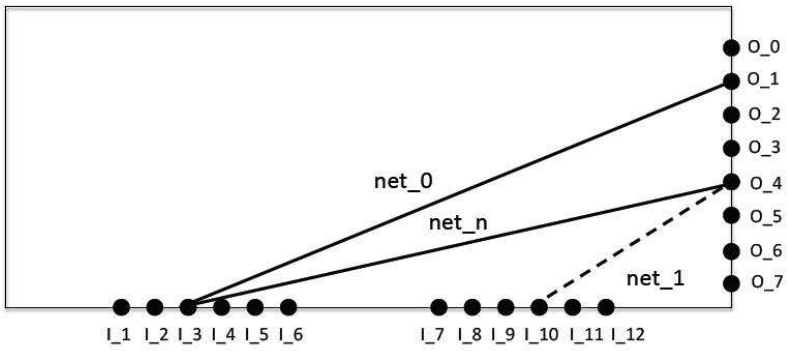


Figure 5: The bridge effect.

- Stuck-at 0 on B_i: The logic signal on B_i is set at zero.
- Stuck-at 1 on B_i: The logic signal on B_i is set at one.
- Logical bridge between B_i and D_j: The logic values on B_i and D_j are exchanged.
- Wired-AND (Wired-OR) between B_i and D_j: The logic signals on B_i and on D_j are set to the value a AND (OR) c.
- Wired-MIX between B_i and D_j: B_i is set to 1 and D_j is set to 0 if $a \neq c$, while B_i and D_j are left unaltered otherwise.

In detail, considering the picture in Fig. 5, two interconnections are initially configured (net_0 and net_1). The scenario that may happen once a configuration memory bit is changed is the bridge phenomenon. In the case represented in Fig. 5, the interconnection segment corresponding to net_1 is disabled while a new segment corresponding to the interconnection net_n is instantiated between a used input node and the output nodes of the previously used net_1 as shown in Fig. 5. The behavior of the circuit is therefore modified. The characteristics of the topological modification depend on the electrical characteristics of the segment's input and output nodes. In detail, the schema in Fig. 5 is a simplification of a possible realistic scenario. The electrical behavior of the nodes represented between I_1 and I_6 is different from the ones belonging to the nodes from I_7 to I_12. A realistic routing switch box contains several hundreds of routing interconnection points having different electrical behavior. The main characteristics of the node are related to its physical characteristics. In general a node can be unidirectional or bidirectional. The bidirectional nodes are generally used to support extra switch box connections, for instance long lines between different routing switch boxes. A long line may have several input and output points, depending on where the routing is topologically configured by the place and route algorithm. If a bridge phenomenon occurs, it may interfere

with different interconnection nodes with different electrical behavior that may lead to a wired-and or wired-or characteristic, if the node is a starting point of a long line. If the node is not a starting point of a long line, the phenomenon is described through the wired-mix mode. Since the node is an intermediate point without a tied-on configuration, its electrical behavior can be floating and depend on the signal value passing through the node according to the model we described, which has been electrically validated in [8].

Note 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.

4 The SAN Formalism and the Möbius tool

Stochastic Activity Networks (SANs) [32] are an extension of Petri Nets (PN). A SAN is a directed graph 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, one that maps input gates to activities, and one that maps pairs (*activity*, *case*) (see below) to output gates. Each input (output) gate has a set of *input* (*output*) places. Activities may be instantaneous or timed, i.e., with a duration, and may have mutually exclusive outcomes, called *cases*, chosen probabilistically according to the *case distribution* of the activity. Cases can be used to model probabilistic behaviors, e.g., the failure probability of a component. An activity *completes* when its execution terminates.

Enabling condition and firing rule for each activity are specified 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, given the current marking. If these predicates and functions are not specified for some activity, the standard PN rules are assumed.

Places are drawn as circles, input (output) gates as left-pointing (right-pointing) triangles, and instantaneous (timed) activities as thin (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.

Möbius [20] is a software tool that provides a hierarchical modeling paradigm, allowing complex models to be built from individual components, and support for customized measures of system properties. SAN models can be composed by means of *Join* and *Rep* operators. Join is used to compose two or more SANs. Rep is used to construct a model consisting of a number of replicas of a SAN. Models composed with Join and Rep interact via *place sharing*, an extension to the SAN formalism that enables different SAN submodels to communicate through *shared places*. Moreover, places may be complex data structures (*extended* places).

Properties of interest are specified with *reward functions* [31]. A reward func-

tion specifies how to measure a property, represented by a *performance variable*, on the basis of the SAN marking and of the executed activities. Measurements can be conducted at specific time instants, over periods of time, or when the system reaches a steady state.

SANs allow a simulator designer to focus on the definition of the execution flow of the various components of the simulator before defining their detailed behavior. The Möbius tool translates this execution flow into a C++ implementation on top of which the application-specific components are developed. The tool made it possible to develop a flexible simulator that can be easily customized for different types of experiments and analyses. To this purpose, Möbius provides the means to define system properties of interest using reward functions and to automate experiment design with an interactive environment, enabling users to specify the desired statistical properties and let the system produce the code performing the experiments.

5 The ASSESS Simulation Environment

Fig. 6 shows the overall structure of the simulation framework and how it fits in the standard design process of FPGA-based systems.

After the hardware description language specification of the system has been synthesized into a netlist in EDIF format, a parser translates the netlist into an intermediate description of its topology and of its components' functionality. The parser also produces the list of SEUs occurring in configuration bits associated with the logic resources and of their effects, according to the fault model presented in Section 3.1.

After the place and route phase, E^2STAR produces a file containing the list of the SEUs in the configuration bits associated with the routing elements of the FPGA that can alter the structure of the implemented system, according to the fault model presented in Section 3.2. For each SEU, the file contains the list of its effects on the logic components (see Section 5.4). Nevertheless, ASSESS could also be used to simulate only SEUs affecting the logic resources of the FPGA. In this last case, the simulation can be carried out before the place and route phase and E^2STAR is not required.

The simulator is configured for the different types of experiments by specifying a number of parameters that will be described in Section 5.1 and by defining reward functions that compute values for properties of interest (Section 6.1).

By configuring dedicated simulation parameters, ASSESS can generate detailed reports on input data and results. In particular, a detailed SEU report shows, for each injected SEU, whether the SEU has been propagated to at least one output pin.

Algorithm 1 summarizes the overall execution flow of the simulation environment for the three simulation modes (*fault-free*, *stochastic fault injection* and *deterministic fault injection*) that will be presented in the following.

In the stochastic fault simulation mode, the execution flow (excluding the steps involving the EDIF parser and E^2STAR) is automatically repeated as

```
call the EDIF parser;
load the netlist description;
load the list of SEUs in logic components;
if routing faults simulation required then
   call E^2STAR;
   load the list of SEUs in routing components;
end if
load the configuration;
generate/load test patterns;
if fault-free simulation then
   for each clock cycle n do
      simulate;
   end for
else if stochastic fault injection mode then
   for each clock cycle n do
      inject an SEU with the configured SEU probability;
      simulate;
      evaluate reward functions;
   end for
else if deterministic fault injection mode then
   for each SEU do
      inject the SEU;
      for each clock cycle do
         simulate;
         evaluate reward functions;
      end for
      clear fault;
   end for
end if
collect and output results.
```

Algorithm 1: The overall behavior of the proposed SEU simulation environment.

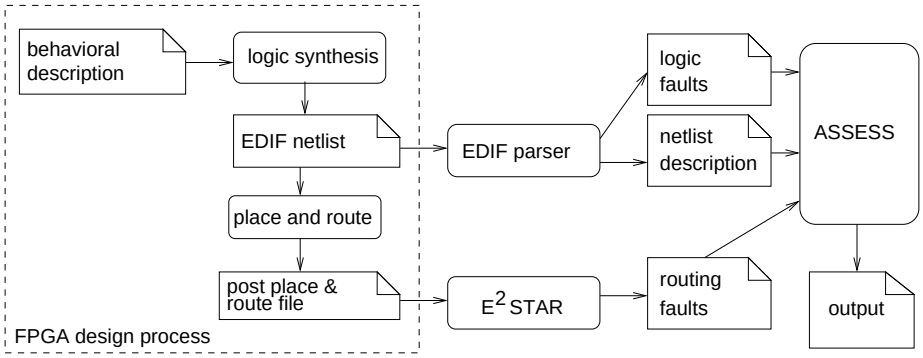


Figure 6: Flow Diagram of the Simulation Environment.

many times as needed to achieve the required statistical confidence.

5.1 The ASSESS Tool

The high-level structure of ASSESS is shown in Figure 7. The simulator is composed of three subsystems: the *SEU Injector*, the *Input Pattern Generator*, and the *Netlist Simulator*.

The SEU Injector reads two lists of configuration memory faults (for logic and for routing resources) and alters the behavior of the simulated circuit according to the corresponding failure modes.

The Input Pattern Generator provides the signals at the input pins at each clock cycle. Signal values may be taken from externally generated test patterns or chosen stochastically according to user-provided signal probabilities.

The Netlist Simulator is the core of the tool, and implements a generic model of execution of digital circuits. In particular, its subsystems Combinational Logic and Sequential Logic simulate the behavior of combinational and sequential elements, respectively.

Three modes of operation are possible: *fault-free simulation*, *deterministic fault injection*, and *stochastic fault injection*. With deterministic fault injection, all possible faults are injected, one at a time, each one at the beginning of a simulation run. A simulation run is the application of a test pattern, i.e., a sequence of test vectors, to the modeled circuit. This operation mode is used to analyze the sensitivity of the system to each fault, and to have a detailed and complete picture of the activability, propagability and criticality of each fault.

With stochastic fault injection, the user sets the maximum number of faults to be injected and a fault is injected with a specified probability q (SEU probability) at each clock cycle. Faults are chosen at random. This operation mode is used to produce statistical measures of the failure probability of the system and of the fault activability and propagability.

As shown in Figure 7, a configuration file is used to control the simulator. Table 1 shows the main parameters of the simulator. Fault-free simulation is per-

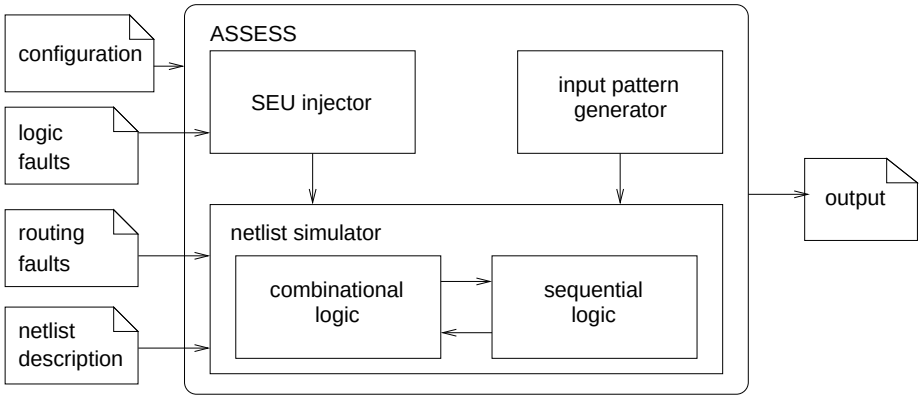


Figure 7: High-level structure of the ASSESS tool.

Table 1: Main parameters of the simulator

PARAMETER	DEFINITION
MAX_CLOCKS	limit to clock cycles to simulate
MAX_SEUS	limit to injected SEUs
STOCHASTIC_INJ	true for stochastic fault injection
ROUTING_FAULTS	true for routing faults injection
q	SEU probability per clock cycle
STOCHASTIC_INPUT	true for stochastic TP generation
p_i	signal probability of i-th primary input
SAVE_FAULTS	true if a detailed output report is generated
SAVE_INPUTS	true if applied TP are saved in output file

formed when MAX_SEUS is zero. Deterministic simulation is performed when STOCHASTIC_INJ is false. Configuration parameter ROUTING_FAULTS enables simulation of routing faults, in addition to logic faults. Stochastically generated input patterns can be saved in order to be reused in further experiments when SAVE_INPUTS is true. Finally, a detailed output report about the activation, propagation to flip-flops and detection of each fault is generated if SAVE_FAULTS is true; otherwise, a report containing only the number of system failures is generated.

The internal structure of the tool rests on three levels of abstraction: 1) A generic SAN model of digital circuit behavior, consisting of a set of interacting SANs; 2) a set of procedures, attached to the SAN model, that specialize its behavior for the typical logical blocks of FPGA netlists, stored in a library accessed by the Möbius environment; and 3) the definitions of the actual systems to be simulated, which comprise the reward model(s), and descriptions of the specific circuit. A user of the ASSESS tool only needs to provide the information

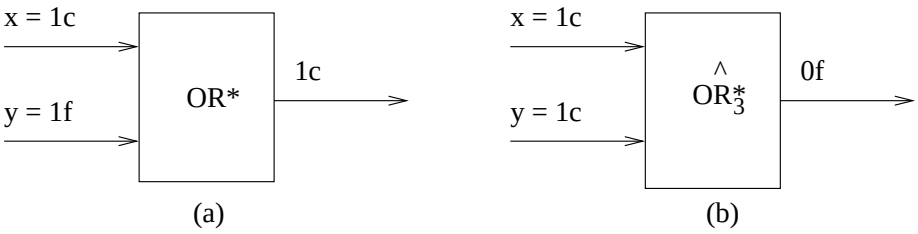


Figure 8: Tracking function: (a) correct component (b) faulty component for input combination $x = 1$, $y = 1$.

required at level 3), in the configuration files.

The SAN model is translated by the Möbius tool into a core set of C++ modules, that in turn call the procedures at the second abstraction level. These procedures fulfill many tasks, including the definition of input predicates for various SAN gates and simulating specific classes of circuit components, such as LUTs, multiplexers, and various kinds of flip-flops. At the third abstraction level, a specific circuit is represented by a set of data structures.

5.2 Tracking functions of components

The simulator uses a four-valued logic [30] to propagate faults through the circuit. Let $B = \{0, 1\}$ be the set of standard Boolean values and $D = \{0_c, 0_f, 1_c, 1_f\}$ be the domain of the four-valued logic. D represents correct and faulty Boolean values, namely *zero correct* (0_c), *one correct* (1_c), *zero faulty* (0_f) and *one faulty* (1_f).

Each component of the netlist implements a Boolean function $f : B^n \to B$. For such function, its tracking function $f^* : D^n \to D$, extends the semantics of f to the four-valued domain D. For a given n-tuple of inputs $(d_1, \cdots, d_n)$ in D^n, the tracking function f^* applies f to the actual inputs and to the input that would have been applied in absence of faults. Then f^* compares the two results and returns a correct or a faulty output accordingly.

Let us now consider the propagation of values through faulty components. For each possible fault i of the component the simulator uses a *faulty* function $\hat{f}_i : B^n \to B$ that describes the behavior of the component in presence of that fault. This behavior may be given in the form of truth table or as an expression. The tracking function $\hat{f}_i^* : D^n \to D$ of a faulty function $\hat{f}_i$ compares the output of the faulty component with possibly faulty inputs to the output of the correct component with correct inputs. If the two are equal, the result is taken as correct, otherwise it is tagged as faulty.

For example, consider a correct OR gate with inputs x and y, where x is correct and equals 1. In this case, the output will be 1 no matter what the other input is. Therefore, if input y is faulty, the output will nevertheless be correct (1). If, instead, the correct value of x were 0, an incorrect 1 value of y would produce an incorrect output value of 1.

Fig. 8 (a) shows the output of the two-input correct LUT implementing the OR operator. A correct 1 input and a faulty 1 input generate a correct output. The value is tagged as correct (1_c) and it is propagated to the next component. Fig. 8 (b) shows the output of the two-input LUT implementing the OR function with a fault in the configuration bit associated with the input $x = 1$ and $y = 1$. If we number each fault with a label i equal to the integer value of the input combination activating the fault, the label of this fault is 3. Two 1_c inputs activate the fault in the component and generate a faulty output, because the resulting output differs from the output that should have been produced by a correct component with $(1, 1)$, and the faulty value (0_f) is propagated.

The simulator uses tracking functions to trace the propagation of faults and to determine whether they reach the output or not, and, if not, to find which components mask or propagate the fault.

5.3 Modeling the circuit

A *component* is an input or output buffer, a flip-flop, a multiplexer, or a LUT. Each component has an identifier. The circuit's topology is represented by a connectivity matrix C with a row for each component and a number of columns equal to the maximum number of input pins per component. Each element of the matrix contains the identifier and the type of the component connected to the corresponding pin, and a flag used for routing faults injection. Thus, each row holds the information needed to find the inputs to a component.

The values of the circuit's signals at each clock cycle are stored in data structures defined in the Möbius model (Sec. 5.5), namely places Input Lines, Internal Lines, and Output Lines. The signals take values of the *extended Boolean* type D discussed in section 5.2.

A *functions table* T holds encodings of the functions of all the components. In particular, the function of each LUT is encoded as an array of Boolean values representing the LUT's truth table.

The current test pattern is stored in a matrix TP, and the current number of generated test vectors is n_v.

5.4 Modeling faults

A data structure F keeps the lists of logic and routing faults.

An array L of Boolean values indicates for each component if it is faulty or not. A single Boolean value is sufficient to model logic faults in buffers, multiplexers and flip-flops, as these components have a single mode of failure.

Faults in LUTs require a more complex representation. Each LUT may be affected by multiple faults, and a fault is represented by the input that activates the fault. Then, array L also has a pointer to the *fault set* injected in each LUT. The fault set is a set of input configurations.

The E^2STAR tool produces a list of all possible routing faults. Each fault is characterized by the list of its logical effects and the components involved by each effect. For example, an E^2STAR output might contain the following entry:

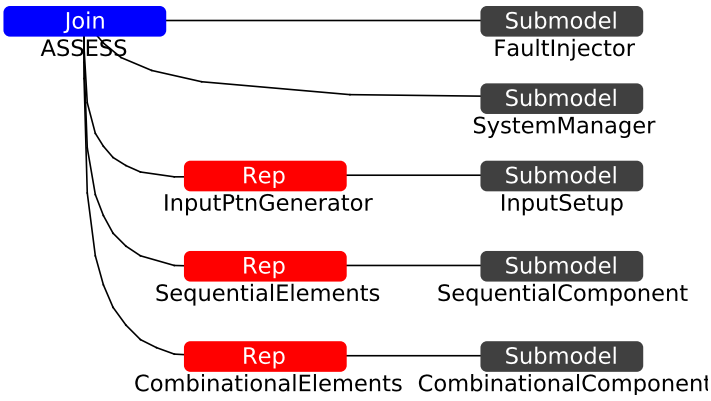


Figure 9: Möbius model of the ASSESS tool.

```
37 2;
0 bridge 22 0 21 0;
1 stuck-at-0 16 0;
```

This entry means that routing fault number 37 has two effects: a bridge between input pin 0 of component 22 and input pin 0 of component 21, and a stuck-at-0 on input pin 0 of component 16.

The simulator maintains the list of routing faults in an array R whose elements hold pointers to the respective effects, each defined by its type (as described in Section 3.2) and its affected components.

A *fault propagation matrix* P with the same structure as the connectivity matrix is used to identify the component input pins affected by routing faults, by storing a pointer to the injected fault.

It may be observed that the actual (faulty) signal value at a given pin cannot be computed in advance for all types of faults, but in general it must be computed when each component is executed for the actual state of the circuit. As an example, if the fault type in element $(22, 0)$ of the fault propagation matrix is a bridge between pin 0 of component 22 and pin 0 of component 21, the value applied to pin 0 of component 22 depends on the actual value of both signals.

5.5 SAN Models

Fig. 9 shows the Möbius model of the tool, obtained by composing SAN submodels with *Join* and *Rep* operators.

The Combinational Logic subsystem of Figure 7 corresponds to the CombinationalElements node in Figure 9, which is the replication of the SAN submodel for the class of combinational components, such as LUTs, I/O buffers and multiplexers. The Sequential Logic subsystem corresponds to the SequentialElements node, the replication of the SAN submodel for storage elements, i.e., different

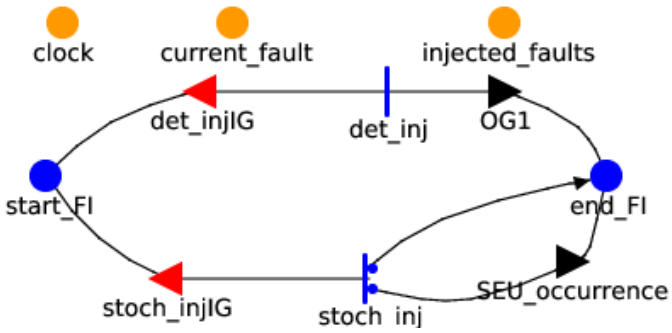


Figure 10: The FaultInjector SAN model.

kinds of flip-flops. The SEU Injector and the Input Pattern Generator subsystems correspond to the FaultInjector and InputPtnGenerator Möbius nodes. In particular, the InputPtnGenerator node has a replica of the InputSetup submodel for each input pin. Finally, submodel SystemManager co-ordinates the other submodels.

The SAN submodules interact through shared places. The main shared places are *Input Lines*, *Output Lines*, and *Internal Lines*, holding the values of the input, output, and internal signals, respectively. Other places hold global variables, such as the numbers of executed clock cycles, of injected faults, and of generated test patterns.

In the following, we describe the FaultInjector and SystemManager models. The other models have been introduced in previous work [12].

5.5.1 The FaultInjector Model

Places start_FI and end_FI (Figure 10) contain a Boolean to indicate if the fault injector is enabled for execution or has terminated its execution, respectively. Depending on the value of a simulation parameter, either the det_injIG or the stoch_injIG input gate enabled its activity. In the first case, the simulator works in deterministic fault injection mode. Place current_fault identifies the fault being simulated with its position in the data structure F. At the beginning of a simulation run (at the first clock cycle), the fault simulated in the previous run is cleared by restoring the affected component to the normal state. The new fault is then injected by updating the relevant data structures and updating the current_fault place. In the rest of the run, OG1 just moves a token to place end_FI.

In the stochastic fault injection mode, the probability of SEU occurrence q is read from a simulation parameter. One case of activity stoch_inj selects, with probability q, gate SEU_occurrence, which chooses randomly one fault. A fault is defined by a component identifier, and additionally by an input configuration if the component is a LUT, or by a routing fault identifier. Array L (Section 5.4)

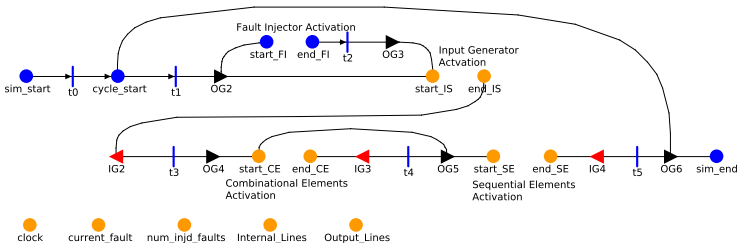


Figure 11: The SystemManager SAN model.

or the fault propagation matrix P are then updated accordingly, and the number of injected faults is increased. No fault is injected if the predetermined number of faults (MAX_SEUS) has been reached. The other case of activity stoch_inj, selected with probability $1 - q$, simply skips fault injection.

5.5.2 The SystemManager Model

The SystemManager SAN model (shown in Fig. 11) orchestrates the execution of the simulation process by co-ordinating the interactions between the various submodules of the simulator. The System manager starts the execution of the FaultInjector through the place named start_FI. Then the System manager waits for the termination of the FaultInjector, notified to the System manager trough the place end_FI. A similar mechanism is used to interact with the InputSetup (start_IS, end_IS places), the CombinationalElements (start_CE, end_CE places) and the SequentialElements (start_SE, end_SE places).

Place cycle_start in Fig. 11 enables the simulation of one clock cycle. Depending on the value of the STOCHASTIC_INJ and MAX_SEUS parameters, transition t_1 starts the FaultInjector, then the InputSetup module is activated.

After InputSetup has generated the input vector for the current clock cycle, the System manager enables the CombinationalElements and then the SequentialElements models.

CombinationalElements is repeatedly activated by transition t_4 (through the output function implemented in gate OG5) until the signal values at the input of the combinational components are stable.

Transition t_5 is enabled when the clock cycle terminates. The output gate OG6 checks if the current simulation run must terminate or a next simulation step must be enabled, depending on the conditions required in each simulation mode.

6 Measurements

The simulator has been used to evaluate different properties. The *failure probability* of a circuit is the fraction of simulation runs in which a faulty output is produced, assuming known fault probabilities. In order to evaluate failure

probability, random input vectors are generated according to specified signal probabilities of the input, faults are injected according to specified fault probabilities, and at each clock a reward function returns 1 if at least one output value is in the set $\{0_f, 1_f\}$.

The *fault observability* analysis consists in studying the propagation of faults to the output of the system, and can be evaluated as the fraction of system failures observed in a series of simulation runs over the total number of injectable faults in the configuration bits. In each simulation run a test pattern is stochastically generated, and at the end of the simulation the list of applied test patterns and associated discovered faults is available. From this list a set of test patterns able to detect all the observed faults can be obtained.

Moreover, regardless of the type of fault injection, the simulator allows *error propagation* and *fault activation* to be estimated. The error propagability analysis consists in measuring the fraction of simulation runs in which at least one flip-flop in the system stores a faulty value, meaning that the fault has been propagated from the fault location to at least one memory element. The fault activability analysis consists in measuring the fraction of simulation runs in which the component affected by the fault produces an incorrect value.

The simulator can be easily extended by adding new measures, in order to analyze different dependability-related properties. For example, in [6], the simulator was extended in order to estimate the probability of occurrence of a subclass of failures, called *catastrophic failures*, that lead the considered system (the battery management system of an electric car) to a dangerous working condition. Further, the simulator has been used in [12] to compute signal probability, computed as the fraction of clock cycles in which a given signal is high, and transition density, computed as the fraction of clock cycles in which a signal makes a transition. Finally, in [13], the simulator was used to assess the fault coverage of test patterns produced by a genetic algorithm-based automatic test pattern generation environment.

6.1 Reward Functions

Reward functions are used in Möbius to collect data during the simulation. In particular, in order to compute the total number of failures observed at the primary outputs, the performance variable *sys_failures*, with the reward function shown in Figure 12, is defined. This function checks, at each clock cycle, if a faulty value is present at an output pin and returns 1 or 0 accordingly. The Möbius-generated code accumulates the values returned by the function in the associated *sys_failures* performance variable, thus counting the number of system failures. The function is executed whenever the marking changes, throughout the simulation. Since the reward function must check for failures only once per clock cycle, the check is made only when all the sequential elements are enabled, which happens exactly once per cycle. This condition holds when all the elements of start_SE have the value 1 and it is implemented by function *check_cycle*.

Similarly, we calculate the probability of a fault to be activated and to

```
double sys_failures_rf()
{ if (check_cycle())
      return check_failures();
   else return 0;
}
bool check_cycle()
{ for (i = 0;
          i < System_Manager->start_SE->length(); i++)
      if (System_Manager->start_SE->Index(i) == 0)
         return false;
   return true;
}
double check_failures()
{ for (int i = 0; i < N_OUTPUT_PINS; i++)
      if ((out_pin[i] == ZERO_F )
          || (out_pin[i] == ONE_F))
         return 1;
   return 0;
}
```

Figure 12: The sys_failures_rf reward function.

be propagated to flip-flops. In particular, with the *sys_activation_rf* reward function we check whether the output of the faulty component is faulty or not. Finally, with the *sys_propagation_rf* reward function we check whether the output of at least one flip-flop is faulty or not. These functions are similar in structure to the one shown in Fig. 12.

Data of interest can also be collected by input and output functions of the gates in the SAN model. In the deterministic simulation mode, a data structure *log_faults* is used, where it is reported whether each fault has been activated, propagated to a flip-flop and propagated to the output of the system. The output gate OG6 of the SystemManager SAN model checks the output pins and the internal lines and updates *log_faults* accordingly.

7 Experimental results

This section presents results from three sets of experiments, one to evaluate the accuracy of the tool, one to show one of its possible applications, and one to evaluate its applicability to large and complex systems. The first two sets are based on the simulation of circuits from the ITC'99 benchmark [19], which provide a diversified set of test cases composed of sequential circuits with a single clock signal, no tristate buses or internal memories, modeled at the RTL level. The number of LUTs and flip-flops for the circuits studied in this work range between 4 and 152 and between 4 and 59, respectively. The VHDL code of the circuits was synthesized with the Xilinx ISE CAD tool. The target device was the Xilinx Virtex-II XC2VP30, that was also used to perform fault injection on a

Table 2: Characteristics of the benchmarks

Circuit	LUTs	FFs	MUXs	IOBs	Function
b01	9	5	0	5	Compare serial flows
b02	4	4	0	3	Recognize binary coded decimal numbers
b03	76	37	0	9	Resource arbiter
b06	9	8	0	9	Interrupt handler
b07	152	51	20	10	Count points on a straight line
b08	40	21	0	14	Find inclusions in sequences of numbers
b09	53	28	0	3	Serial-to-serial converter
b10	52	24	0	18	Voting system
b11	147	38	14	14	Scramble string with var. cipher
b13	106	59	11	21	Interface to meteo sensors

prototype board. The characteristics of the designs used in the experiments are shown in Table 2, which reports for each circuit the number of Look-Up Tables (LUTs), Flip-Flop (FFs), multiplexers, Input and Output buffers (IOBs), and a synthetic description of the circuit's purpose. 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.

Table 3 shows the number F_l of faults in LUTs and buffers and the results of the analysis performed with E^2STAR on the considered circuits, i.e., the number F_r of SEUs affecting configuration bits for routing resources identified by the tool, and the number of affected nodes classified by logical effect: Stuck-at-0, Stuck-at-1, Wired-And, Wired-Mix, and Bridge. Wired-Or effects were not observed. As previously discussed (Section 3.2), the number of propagation points per SEU in the configuration bits controlling the routing structure is much higher than the actual number of SEUs.

In the third set of experiments, the miniMIPS RISC processor, synthesized for the Xilinx Virtex 6 device family, has been studied.

7.1 Validation of ASSESS

Each circuit was simulated by applying 10000 randomly generated test vectors and performing an exhaustive deterministic fault injection. For each circuit, the same test vectors and faults were also applied to its prototype on the fault injection board.

The fault injection setup is the following: A test manager running on the host computer reads the lists of faults and test patterns, builds the faulty bitstreams, loads each of them onto the FPGA and starts the execution of the device driver, a control circuit that feeds the test patterns to the implemented circuit and

Table 3: Effects of SEUs

Circuit	F_l	F_r	SA0	SA1	W-AND	W-Mix	Bridge
b01	129	547	708	2,944	5	7	0
b02	55	304	118	339	5	7	102
b03	963	5,910	8,105	21,661	1,423	1,431	2,320
b06	113	566	372	790	0	18	305
b07	1,730	10,431	18,331	47,762	4,085	3,911	4,739
b08	518	2,689	3,074	8,061	464	496	1,217
b09	695	3,872	6,569	15,948	567	512	1,908
b10	678	3,942	4,603	10,727	482	692	1,498
b11	1,790	10,104	14,059	35,749	3,537	3,536	4,480
b13	1,237	7,203	10,390	27,720	1,143	1,387	3,602

reports the results to the test manager, which compares them with the expected results obtained from a golden copy.

Table 4 shows the two sets of results: The total number F_t of faults (in LUTs, buffers, and routing elements) and, for both sets, the number of detected faults (D_{sim} for simulation and D_{fi} for fault injection) and the fault observability (O_{sim} and O_{fi}).

The comparison between columns D_{sim} and D_{fi} shows that the proposed simulation method can accurately reproduce the effects of SEUs affecting any configuration bit of an SRAM-based FPGA system. In particular, the comparison with results obtained by fault injection shows that our simulator has an error ranging between 0.0% and 0.5% for this benchmark (0.1% on average).

In order to show the accuracy of ASSESS and of the adopted fault models, a modified version of the ASSESS tool, built with the same architecture but adopting the stuck-at model currently assumed in commercial and academic tools, was used to simulate the above mentioned benchmark (Table 5) applying the same input patterns used in the previous experiment. The simulations produced the number D_{sa} of detected failures with respect to the number F_{sa} of possible stuck-at faults, at zero and at one, in LUTs and buffers, giving the fault observabilities (O_{sa}) reported in the table.

Results from this experiment show that traditional fault simulators adopting the stuck-at fault model obtain much different fault detection values than ASSESS. In particular, considering the fault detection values achieved with fault injection as a reference, it can be observed that the error obtained with stuck-at fault simulators, which ranges between 0.0% and 57.1% (15.2% on average), is much higher than the error obtained using ASSESS.

Table 4: Comparison between simulation and fault injection

Circuit	F_t	D_{sim}	D_{fi}	O_{sim} (%)	O_{fi} (%)
b01	676	676	676	100.0	100.0
b02	359	352	350	98.0	97.5
b03	6,873	3,285	3,278	47.8	47.7
b06	679	670	670	98.7	98.7
b07	12,161	927	927	7.6	7.6
b08	3,207	157	157	4.9	4.9
b09	4,567	2,081	2,080	45.5	45.5
b10	4,620	3,548	3,545	76.8	76.7
b11	11,894	7,521	7,519	63.2	63.2
b13	8,440	2,517	2,515	29.8	29.7

Table 5: Estimated SEU Observability, Stuck-at Model

Circuit	F_{sa}	D_{sa}	O_{sa}
b01	28	28	100.0%
b02	24	24	100.0%
b03	170	78	45.9%
b06	36	36	100.0%
b07	324	17	5.1%
b08	108	23	2.1%
b09	112	42	37.6%
b10	140	107	76.2%
b11	322	259	80.5%
b13	254	69	27.1%

7.2 Examples of ASSESS Applications

In the following, we show how ASSESS can be used to measure dependability-related properties, including some that could not be measured with fault injection or radiation experiments. The failure, fault activation and error propagation probabilities of three circuits from the above mentioned benchmark were measured as functions of operating time, in a single-fault and in an accumulated-faults scenario, with realistic assumptions on SEU rate and clock frequency.

Circuits b08, b09, and b10 were simulated with stochastic fault injection for operating times ranging from 100,000 to 1,000,000 clock cycles, in steps of 100,000, using randomly generated test vectors. The SEU rate of 82 s^{-1} reported in [3] and a working frequency of 1 MHz were assumed, resulting in an SEU rate q of $8,2 \cdot 10^{-5}$ per clock cycle, and in operating times between

100 milliseconds and 1 second. The Möbius tool was configured to compute the performance variables with a confidence level of 0.95 and a confidence interval of 0.1, which required between 1,000 and 5,000 simulation runs for each value of operating time. We chose the 0.95 confidence level and 0.1 confidence interval since they allowed us to obtain results with a 5% average error in a reasonable time. Nevertheless, these parameters can be relaxed or strengthened in order to speed up the analysis or obtain more accurate results, respectively.

For each of the three properties both single fault analysis, in which just one SEU was injected in the system during the operating time, and fault accumulation analysis, in which multiple SEUs were injected in the system, were carried out. Fig.s 13(a), 13(b), and 13(c) refer to the single fault scenario and show, for each value of operating time, the probability that a fault is activated, the probability that an error reaches a flip-flop and that the fault corrupts the output of the system respectively. Similarly, Fig.s 14(a), 14(b), and 14(c) show the same properties in the fault accumulation scenario.

The analysis of fault activation and fault propagation gives designers a deeper knowledge of the system from the point of view of dependability. For example, knowing which faults have been activated and how they propagate in the circuit could help designers in the generation of test patterns much more than the analysis of the system at the input/output level. Moreover, knowing which parts of the circuit are more prone to fault activation and propagation could help designers in the development of selective fault tolerance and hardening techniques. Finally, knowledge of the probability of an error to be propagated to a flip-flop could be used to implement more efficient reset and state restoration mechanisms to be used in conjunction with configuration memory scrubbing to recover from faulty states.

Failure probability and fault observability are useful when the focus is on studying the robustness of the system to faults: Knowing the working frequency of the system and the SEU rate, the mean time to failure can be estimated, helping designers to determine the optimal scrubbing frequency for a self-correcting system. Moreover, analyzing the failure probability of the system could be used to assess the effectiveness of fault tolerance techniques. Finally, the fault coverage achieved by a set of test patterns can be assessed through analysis of the system's fault observability.

In order to better highlight how ASSESS could be used to get a deeper insight into the internal behavior of the circuit in the presence of SEUs, we report in Fig. 15 the probability of SEUs of being propagated to flip-flop 0, flip-flop 1, output pin 0, and output pin 1 in circuit b08.

The computations to perform the stochastic fault injection experiments (shown in Fig.s 13 to 15) took about ten minutes for each point of the plots.

The most time-demanding analysis was the validation experiment. The exhaustive deterministic fault injection simulations took between 2 and 5 minutes for the small circuits (b01, b02 and b06), about one hour for the medium sized ones (b03, b08 and b09), and between one and five days for the largest circuits (b07, b11 and b13). This was expected since, as discussed in Section 3, the number of SEUs (both in the logic and in the routing structure) that must be

Table 6: Characteristics of the miniMIPS RISC processor

Functional Unit	LUTs	FFs
Register Bank	1,745	992
BUS Controller	38	1
Instruction Decoding	410	207
Instruction Extraction	66	66
Execution + ALU	953	238
Memory Access	108	107
Address Calculation	97	32
Branch Prediction	1,780	564
Bypass Unit	168	0
Coprocessor System	326	129
total	5691	2336

simulated with such an accurate fault model is very large.

The bounds on the scalability of the tool strongly depend on the number of faults to be simulated. Taking a 6-input LUT as an example, with the stuck-at fault model only two faults have to be considered (the stuck-at-0 and stuck-at-1 on the LUT output, while the stuck-at faults on its inputs are analyzed when simulating the outputs of the upstream LUTs), while, with the SEU fault model considered in the paper, 64 SEUs must be simulated (one for each of the 2^6 configuration bits of the LUT).

7.3 The miniMIPS RISC processor

In order to show how ASSESS can be used to analyze large circuits, we performed a set of experiments on the miniMIPS RISC processor [1]. Moreover, in order to show that ASSESS can be applied to systems synthesized for any FPGA device family, we synthesized the miniMIPS processor for the Virtex 6 family. The synthesized circuit is composed of 5691 LUTs, 2336 flip-flops, and 138 I/O buffers. Table 6 reports the characteristics of the miniMIPS RISC processor.

The number F_l of faults in LUTs and buffers, the number F_r of SEUs in configuration bits for routing resources, and the number of affected nodes classified by logical effect, are shown in Table 7, which summarizes the analysis performed with the E^2STAR tool.

In Fig. 16, we show the probability that some components of the processor propagate a faulty value to their outputs after the occurrence of 1 to 10 SEUs. These experiments were carried out by simulating 10000 randomly generated test patterns. The analysis took about half an hour for each point.

Finally, a deterministic SEU injection was performed, using 10000 previously generated test patterns on four functional units of the processor: In this way we also show that ASSESS can be used to simulate deterministically generated

Table 7: Effects of SEUs in the miniMIPS Processor
W-A: Wired-AND; W-B: Wired-Mix; B: Bridge

Circuit	F_l	F_r	SA0	SA1	W-A	W-M	B
Register Bank	24,430	123,808	160,323	453,021	421	390	93
BUS Controller	531	2,699	3,832	6,492	0	3	0
Inst. Decoding	5,744	30,135	51,239	52,532	34	31	12
Instr. Extraction	925	4,685	6,097	6,130	3	9	0
Execution + ALU	13,534	68,632	87,562	88,532	81	89	3
Memory Access	1,523	7,662	9,932	15,032	7	18	1
Addr. Calculation	1,453	6,882	8,907	11,945	5	4	0
Branch Prediction	24,332	126,294	163,462	249,993	90	16	8
Bypass Unit	2,491	11,914	15,353	15,843	13	3	0
Coprocessor	4,564	23,129	31,032	58,302	49	53	11

Table 8: Results from deterministic SEU injection on some functional units of the miniMIPS processor

Functional Unit	#Clock Cycles	Inj. SEUs [%]	Det. SEUs [%]
Instruction Extraction	10,000	100%	8.91%
Memory Access	10,000	100%	25.70%
Address Calculation	10,000	100%	40.42%
Bypass Unit	10,000	100%	36.92%

input patterns, in addition to stochastically generated ones. The fraction of detected faults is reported in Table 8.

8 Conclusions and Further Work

This paper has presented ASSESS, a digital circuit simulator designed for the study of soft errors in the configuration memory of SRAM-based FPGAs. A detailed model of the circuits and of the logical effects of SEUs in their configuration memory has been developed around a formal model based on Stochastic Activity Networks. The fault simulator is proposed as an additional tool to be used during the design of an FPGA-based system, in order to get to the final fault injection or radiation experiment with a deeper knowledge of the system's behavior in the presence of faults.

The accuracy of the dependability measures computed with the developed SAN-based simulator has been validated against results obtained with a fault injection board.

Currently available commercial and academic fault simulators model faults at a much coarser detail level than the one adopted in ASSESS, which achieves

significantly more accurate results in terms of fault observability and failure probability estimation.

The simulator can be used to evaluate a number of dependability-related properties, using various policies to generate faults and test vectors. In particular, both faults and test vectors can be generated stochastically according to user-supplied probabilities, or deterministically. Simulations are configured through a wide range of parameters characterizing the simulated circuit and the statistical properties of the sought results.

As future work we plan to extend the simulator in order to consider a larger spectrum of faults, such as hard errors and multiple cell upsets. Further, abstraction techniques will be studied to improve the scalability of the simulator, through the generation of a reduced model of the states of the system.

References

[1] miniMips overview page at OpenCores. [Online] Available: `http://opencores.org/project,minimips,overview`

[2] Miguel Angel Aguirre, Jonathan Noel Tombs, Vicente Baena, Fernando Muñoz, Antonio Jesus Torralba, A. Fernández-León, and F. Tortosa-López. Ft-Unshades: a New System for Seu Injection, Analysis and Diagnostics Over Post Synthesis Netlist. In *Proc. of the 8th Military and Aerospace Programmable Logic Devices Int. Conf. (MAPLD'05)*, 2005.

[3] M. Alderighi, F. Casini, S. D'Angelo, M. Mancini, A. Marmo, S. Pastore, and G. R. Sechi. A Tool for Injecting SEU-Like Faults into the Configuration Control Mechanism of Xilinx Virtex FPGAs. In *Proc. 18th IEEE Int. Symp. Defect and Fault Tolerance in VLSI Systems (DFT '03)*, pages 71–78, Washington, DC, USA, 2003. IEEE Computer Society.

[4] M. Alderighi, F. Casini, S. D'Angelo, M. Mancini, S. Pastore, and G.R. Sechi. Evaluation of Single Event Upset Mitigation Schemes for SRAM based FPGAs using the FLIPPER Fault Injection Platform. In *Proc. 22nd IEEE Int. Symp. Defect and Fault-Tolerance in VLSI Systems (DFT '07)*, pages 105–113, Sept. 2007.

[5] Ghazanfar Asadi and Mehdi B. Tahoori. An analytical Approach for Soft Error Rate Estimation of SRAM-based FPGAs. In *Proc. Military and Aerospace Applications of Programmable Logic Devices (MAPLD)*, pages 2991–2994, 2004.

[6] F. Baronti, C. Bernardeschi, L. Cassano, A. Domenici, R. Roncella, and R. Saletti. Mitigation Techniques of Single Event Upsets in the Control Logic of a Charge Equalizer for Li-ion Batteries. In *39th Annu. Conf. of the IEEE Industrial Electronics Society (IECON 2013)*, pages 6756–6761. IEEE, 2013.

[7] R.C. Baumann. Radiation-induced Soft Errors in Advanced Semiconductor Technologies. *IEEE Trans. Device Mater. Rel.*, 5(3):305–316, Sept. 2005.

[8] M. Bellato, P. Bernardi, D. Bortolato, A. Candelori, M. Ceschia, A. Paccagnella, M. Rebaudengo, M.S. Reorda, M. Violante, and P. Zambolin. Evaluating the effects of SEUs affecting the configuration memory of an SRAM-based FPGA. In *Design, Automation and Test in Europe Conf. and Exhibition, 2004*, volume 1, pages 584–589, Feb. 2004.

[9] M. Bellato, M. Ceschia, M. Menichelli, A. Papi, J. Wyss, and A. Paccagnella. Ion beam testing of SRAM-based FPGA's. In *Proc. 6th European Conf. on Radiation and Its Effects on Components and Systems*, pages 474–480, Sept. 2001.

[10] C. Bernardeschi, L. Cassano, and A. Domenici. Failure Probability and Fault Observability of SRAM-FPGA Systems. In *Int. Conf. Field Programmable Logic and Applications (FPL2011)*, pages 385–388, Sept. 2011.

[11] C. Bernardeschi, L. Cassano, and A. Domenici. Failure Probability of SRAM-FPGA Systems with Stochastic Activity Networks. In *Proc. 14th IEEE Symp. on Design and Diagnostics of Electronic Circuits and Systems*, Apr. 2011.

[12] C. Bernardeschi, L. Cassano, A. Domenici, and P. Masci. A Tool for Signal Probability Analysis of FPGA-Based Systems. In *Proc. 2nd Int. Conf. Computational Logics, Algebras, Programming, Tools, and Benchmarking*, 2011.

[13] Cinzia Bernardeschi, Luca Cassano, Mario G.C.A. Cimino, and Andrea Domenici. GABES: A genetic algorithm based environment for SEU testing in SRAM-FPGAs. *J. of Systems Architecture*, 59(10, Part D):1383–1254, 2013.

[14] Cinzia Bernardeschi, Luca Cassano, Andrea Domenici, and Luca Sterpone. Accurate Simulation of SEUs in the Configuration Memory of SRAM-based FPGAs. In *IEEE Int. Symp. Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT 2012)*, pages 115–120, Oct. 2012.

[15] W. Calienes Bartra and R. Reis. SET and SEU simulation toolkit for LabVIEW. In *12th European Conf. Radiation and Its Effects on Components and Systems (RADECS2011)*, pages 829–836, Sept. 2011.

[16] Carl Carmichael, Earl Fuller, Joe Fabula, and Fernanda D. Lima. Proton testing of SEU mitigation methods for the Virtex FPGA. In *Proc. IEEE Microelectronics Reliability and Qualification Workshop*, Pasadena, CA, Dec. 2001.

[17] M. Ceschia, M. Bellato, A. Paccagnella, S. C. Lee, C. Wan, A. Kaminski, M. Menichelli, A. Papi, and J. Wyss. Ion beam testing of Altera Apex

FPGAs. In *Proc. 2002 IEEE Radiation Effects Data Workshop*, pages 45–50, July 2002.

[18] G. Clark, T. Courtney, D. Daly, D. D. Deavours, S. Derisavi, J. M. Doyle, W. H. Sanders, and P. G. Webster. The Möbius modeling tool. In *9th Int. Workshop on Petri Nets and Performance Models*, pages 241–250, Aachen, Germany, Sept. 2001. IEEE Computer Society Press.

[19] F. Corno, M. Sonza Reorda, and G. Squillero. RT-Level ITC'99 Benchmarks and First ATPG Results. *IEEE Des. Test*, 17:44–53, July 2000.

[20] Daniel D. Deavours, Graham Clark, Tod Courtney, David Daly, Salem Derisavi, Jay M. Doyle, William H. Sanders, and Patrick G. Webster. The Möbius framework and its implementation. *IEEE Trans. Softw. Eng.*, 28(10):956–969, Oct. 2002.

[21] Earl Fuller, Michael Caffrey, Phil Blain, Carl Carmicheal, Noor Khalsa, and Anthony Salazar. Radiation test results of the Virtex FPGA and ZBT SRAM for space based reconfigurable computing. In *Proc. 2nd Military and Aerospace Programmable Logic Devices Int. Conf. (MAPLD'99)*, 1999.

[22] Earl Fuller, Michael Caffrey, Anthony Salazar, Carl Carmichael, and Joe Fabula. Radiation Testing Update, SEU Mitigation, and Availability Analysis of the Virtex FPGA for Space Reconfigurable Computing. In *Proc. 3rd Military and Aerospace Programmable Logic Devices Int. Conf. (MAPLD'00)*, 2000.

[23] P. Graham, M. Caffrey, J. Zimmerman, D. E. Johnson, P. Sundararajan, and C. Patterson. Consequences and Categories of SRAM FPGA Configuration SEUs. In *Proc. 6th Military and Aerospace Applications of Programmable Logic Devices (MAPLD'03)*, pages 1–9, Sept. 2003.

[24] Daniel González Gutiérrez. Single event upsets simulation tool functional description. Technical Report TEC-EDM/DGG-SST2, ESA-ESTEC, 2000.

[25] O. Heron, T. Arnaout, and H.-J. Wunderlich. On the reliability evaluation of SRAM-based FPGA designs. In *Proc. Int. Conf. Field Programmable Logic and Applications (FPL'05)*, pages 403–408, Aug. 2005.

[26] Ian Kuon, Russell Tessier, and Jonathan Rose. FPGA Architecture: Survey and Challenges. *Found. Trends Electron. Des. Autom.*, 2(2):135–253, February 2008.

[27] Mentor Graphics Corporation. *ModelSim SE Reference Manual*, 2008.

[28] Ali Movaghar and J. F. Meyer. Performability modelling with stochastic activity networks. In *Real-Time Systems Symp., Austin, TX, USA*, pages 215–224, 1984.

[29] M. Rebaudengo, M. Sonza Reorda, and M. Violante. A new functional fault model for FPGA application-oriented testing. In *Proc. 17th IEEE Int. Symp. Defect and Fault Tolerance in VLSI Systems (DFT 2002)*, pages 372–380, 2002.

[30] J. Paul Roth. Diagnosis of Automata Failures: A Calculus and a Method. *IBM J. of Research and Development*, 10(4):278 –291, July 1966.

[31] William H. Sanders and John F. Meyer. A unified approach for specifying measures of performance, dependability, and performability. In Algirdas Avižienis, Herman Kopetz, and Jean-Claude Laprie, editors, *Dependable Computing for Critical Applications*, pages 215–237. Springer-Verlag Heidelberg, 1991.

[32] William H. Sanders and John F. Meyer. Stochastic activity networks: formal definitions and concepts. In *Lectures on formal methods and performance analysis: first EEF/Euro summer school on trends in computer science*, pages 315–343. Springer-Verlag New York, Inc., New York, NY, USA, 2002.

[33] S. Schulz, G. Beltrame, and D. Merodio-Codinachs. Smart behavioral netlist simulation for SEU protection verification. In *9th European Conf. Radiation and Its Effects on Components and Systems (RADECS2008)*, pages 406–411, Sept. 2008.

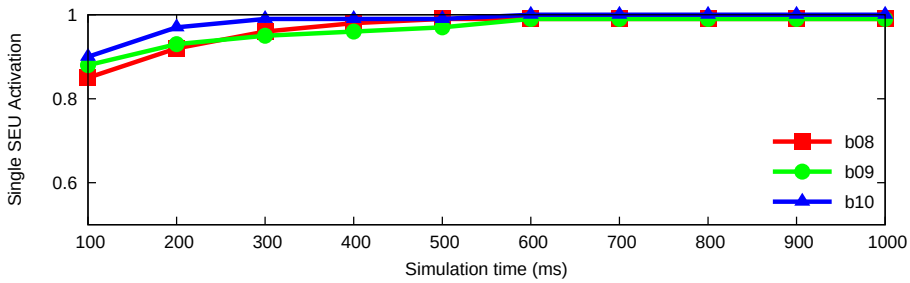
[34] L. Sterpone and M. Violante. A new analytical approach to estimate the effects of SEUs in TMR architectures implemented through SRAM-based FPGAs. *IEEE Trans. Nucl. Sci.*, 52(6):2217–2223, Dec. 2005.

[35] L. Sterpone and M. Violante. Analysis of the robustness of the TMR architecture in SRAM-based FPGAs. *IEEE Trans. Nucl. Sci.*, 52(5):1545–1549, Oct. 2005.

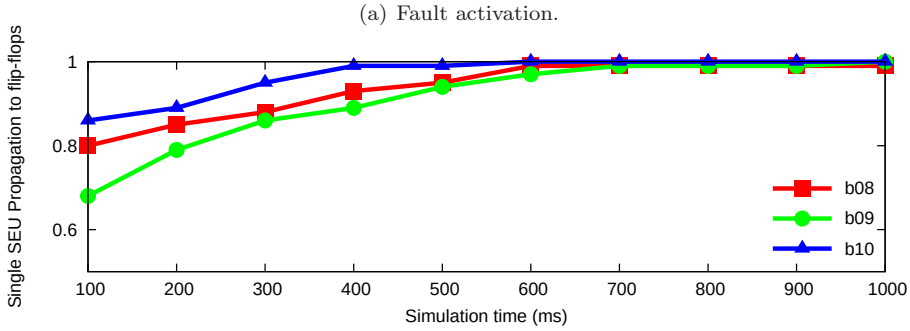
[36] M. Straka, J. Kastil, and Z. Kotasek. SEU Simulation Framework for Xilinx FPGA: First Step towards Testing Fault Tolerant Systems. In *Proc. 14th Euromicro Conf. Digital System Design (DSD2011)*, pages 223–230, Sept. 2011.

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

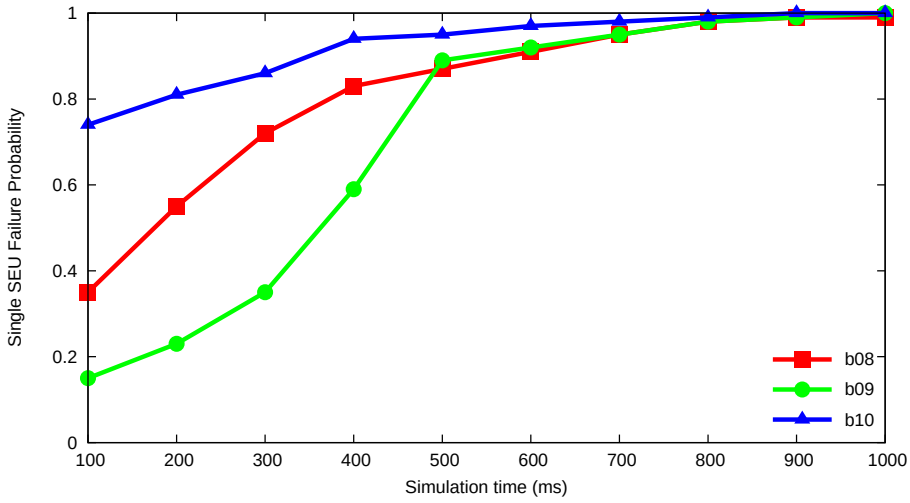
[38] M. Wirthlin, E. Johnson, N. Rollins, M. Caffrey, and P. Graham. The reliability of FPGA circuit designs in the presence of radiation induced configuration upsets. In *Proc. 11th Ann. IEEE Symp. Field-Programmable Custom Computing Machines (FCCM 2003)*, pages 133–142, Apr. 2003.



(a) Fault activation.

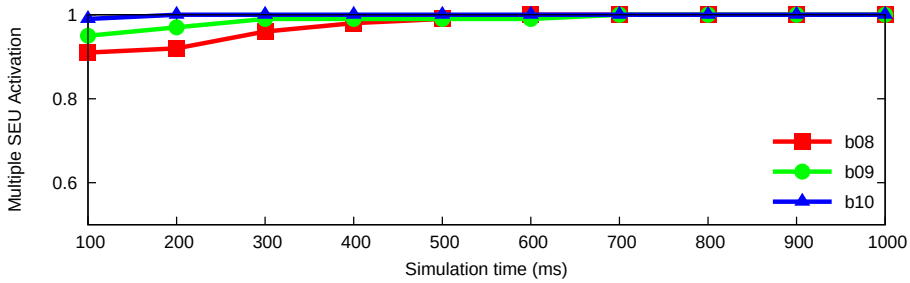


(b) Error propagation to flip-flops.

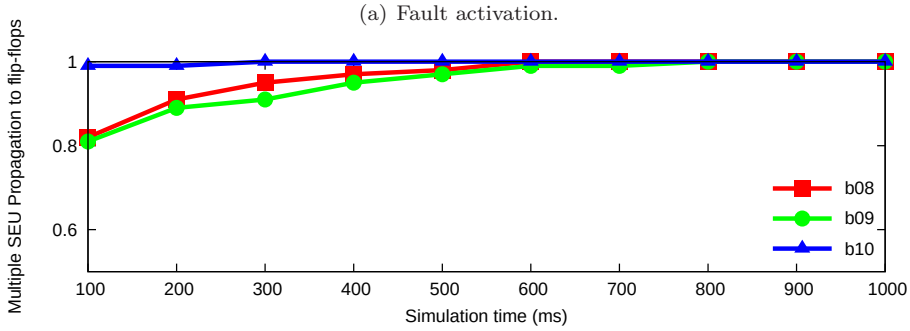


(c) Failure probability.

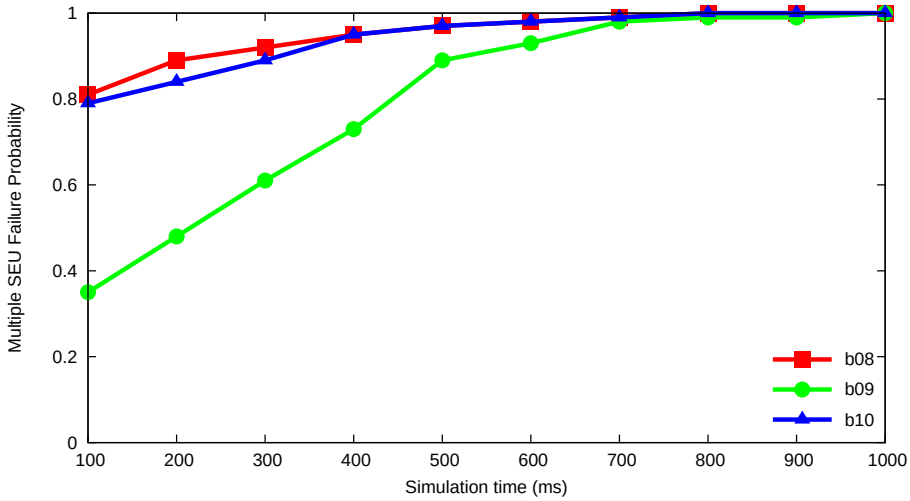
Figure 13: Results from fault simulation in the single SEU scenario.



(a) Fault activation.



(b) Error propagation to flip-flops.



(c) Failure probability.

Figure 14: Results from fault simulation in the SEU accumulation scenario.

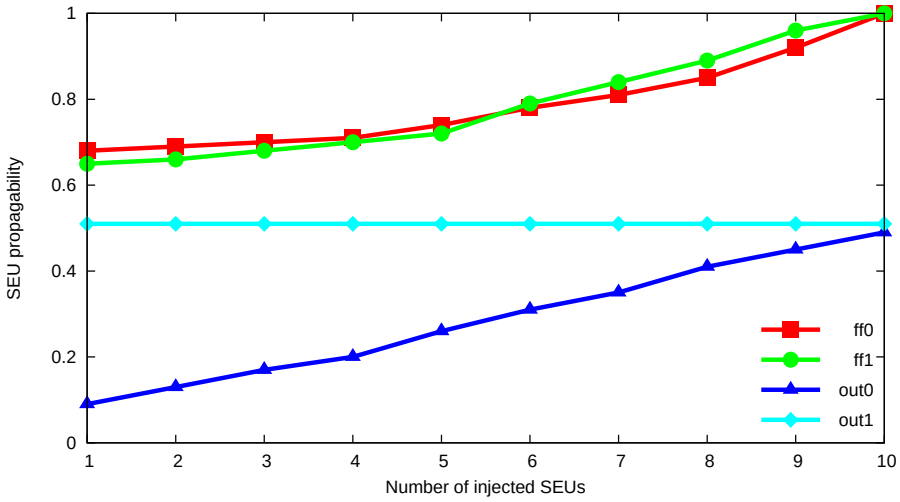


Figure 15: Probability of an SEU of being propagated to flip-flops 0 and 1 and to output pins 0 and 1 in circuit b08.

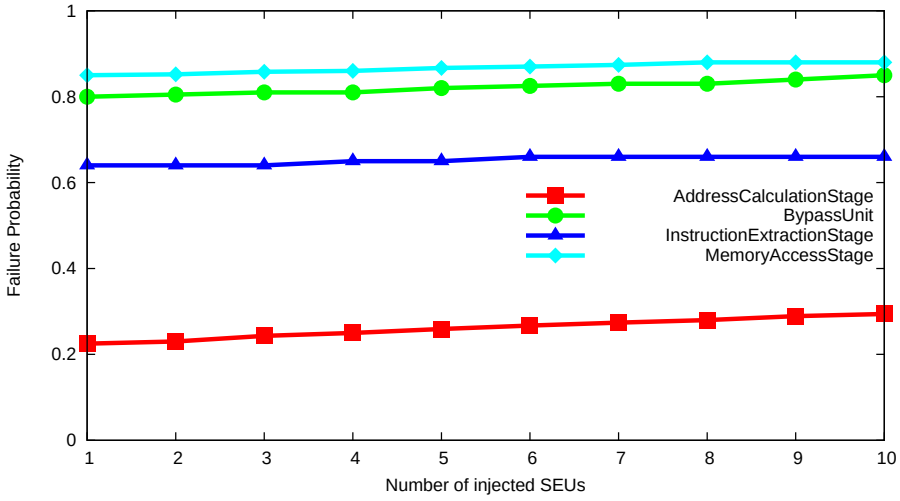


Figure 16: Probability of four miniMIPS processor components of propagating a faulty value.