Exploiting Circuit Emulation for Fast Hardness Evaluation

P. Civera, L. Macchiarulo, M. Rebaudengo, M. Sonza Reorda, and M. Violante

***Abstract*—Hardware designers need effective techniques for early evaluation of the hardening mechanisms adopted in safety-critical VLSI circuits. We propose field-programmable gate-array based circuit emulation for performing fault-injection campaigns. Experimental results show that the new technique is about four orders of magnitude faster than simulation-based fault injection.**

***Index Terms*—Fault-injection, FPGA, safety-critical applications, single-event upset (SEU).**

I. Introduction

THE DESIGN of safety-critical VLSI circuits requires techniques and tools for early evaluation of the correctness and effectiveness of the mechanisms adopted to harden circuits against SEU effects. In particular, designers are interested in tools for performing fault-injection campaigns when a prototype of the designed circuit is not yet available, and thus when radiation testing cannot be exploited. In this context, the adopted fault model is often the (single/multiple) bit-flip in the circuit storage elements, i.e., registers and embedded memories. Figures obtained exploiting this model can be used to predict the circuit error rate [1].

Simulation-based fault injection offers a promising solution to the above problem since it allows early evaluation of the system dependability when only a model of the system is available. Moreover, it is very flexible in terms of fault model, since any fault model can potentially be supported and faults can be injected in any module of the system. The major drawback of this approach is the high CPU time required for performing circuit simulation.

In the last years, several approaches to simulation-based fault injection have been proposed. Earlier works, such as [2], proposed the exploitation of switch-level simulators for analyzing error propagation through a circuit. To match the constantly increasing complexity of the target circuits, a probabilistic and a deterministic approach have been proposed.

The method proposed in [3] follows the former approach: the authors exploit device-level simulation to estimate the probability that ion–device interactions produce erroneous signals capable of propagating to memory elements and logic simulation to analyze how wrong information propagates among the circuit. More recently, a further method has been proposed, based on the probabilistic description of error propagation in VLSI circuits, formulated and solved as a set of linear equations [4].

The alternative approach is based on deterministic simulation. Inspired by the widespread adoption of VHDL simulator in VLSI design centers and the high level of efficiency of modern simulation algorithms, several authors proposed the use of VHDL simulators to perform fault-injection campaigns, and several approaches have been presented (e.g., [5]–[8]) for speeding up the process. The approach presented in [9] is based on probabilistic and deterministic models allowing the identification of the most sensible elements among the gates in a combinational circuit. As a result, fault injection is performed on a potentially small subset of the circuit gates, thus saving simulation time.

In this paper, we propose an alternative solution for reducing the time spent during fault-injection campaigns, which is best suited to study the effects of transient errors affecting the circuit memory elements. Our fault-injection technique is able to perform injection campaigns in a fraction of the time simulation-based approaches require, while still supporting most of their positive features. In particular, we use low-cost commercial off-the-shelf field-programmable gate-array (FPGA) devices for efficiently emulating an instrumented version of circuit under analysis. The instrumentation technique allows easily modifying and observing the content of the circuit memory elements. Please note that the approach is not intended to evaluate the effect of faults on FPGA devices. Conversely, it exploits FPGAs for effective circuit emulation.

Several works already explored the use of FPGAs for speeding up fault simulation [10], [11] of permanent single stuck-at faults. In [12], the extension of their usage to fault injection is proposed, but the approach is based on reprogramming the FPGA once for each fault. The reconfiguration, which can be only partial, results in a time overhead reducing fault-injection efficiency. In [13], a new method based on a mutant generation on an FPGA circuit has been presented. This method avoids circuit reconfiguration but could introduce a significant area overhead.

When compared with previous FPGA-based methods, our solution does not require FPGA reconfiguration (either complete or partial) for each fault experiment, thus attaining a much higher efficiency in terms of elapsed time. Moreover, our solution allows one not only to efficiently perform fault injection but also to effectively support the observation of faulty behavior (e.g., for the observation of latent faults). A prototypical version of the proposed fault-injection environment has been developed,

Manuscript received July 17, 2001. This work was supported in part by the Italian Space Agency (ASI).

P. Civera and L. Macchiarulo are with the Dipartimento di Elettronica, Politecnico di Torino, Torino, Italy (e-mail: civera@polito.it; luca@vlsilab01.polito.it).

M. Rebaudengo, M. Sonza Reorda, and M. Violante are with the Dipartimento di Automatica e Informatica, Politecnico di Torino, Torino, Italy (e-mail: reba@polito.it; sonza@polito.it; violante@polito.it).

Publisher Item Identifier S 0018-9499(01)10713-6.

showing that the whole fault-injection process can be highly automated and easily introduced into existing VLSI design flows. Experimental results showed that our approach is four orders of magnitude faster than state-of-the-art VHDL simulation-based fault-injection techniques.

II. The Fault-Injection System

A. Background and Assumptions

According to requirements coming from some European space and car industries [14], the fault model we consider is the (single/multiple) bit-flip in the circuit storage elements, since it closely matches SEU effects. As pointed out in [14], SEUs are the major source of concerns; therefore, in this paper we do not consider more complex effects, such as single-event transients.

In this work, we considered a typical fault-injection environment composed of three modules: a *Fault List Manager* in charge of generating the list of faults to be injected; a *Fault Injection Manager* for orchestrating the selection of a new fault, its injection in the system, and the observation of the resulting behavior; and a *Result Analyzer* that analyzes the output data produced by the system during each fault-injection experiment, categorizes faults according to their effects, and produces statistical information.

Fault effects are classified according to the following categories.

1) *Silent:* the output produced by the faulty circuit and the content of all its memory elements at the end of the simulation correspond to the ones of the fault-free circuit.
2) *Latent:* the output produced by the faulty circuit corresponds to the fault-free one, but the content of their memory elements at the end of the simulation do not match. As a consequence, the fault is still active in the circuit and may later produce wrong outputs.
3) *Failure:* the output produced by the faulty circuit does not match the one of the fault-free circuit, and the circuit error detection mechanisms (EDMs) do not detect the fault.
4) *Detected:* the fault is detected by any EDMs existing in the system.

B. FPGA-Based Fault-Injection Manager

As far as simulation-based fault-injection approaches are considered, the fault-injection manager is normally the most critical module, since it may require a huge amount of time for performing its task when the size of the circuit, or the number of faults to be injected, or the length of the input sequence to be considered become large. In our approach, this module exploits an FPGA board that emulates the gate-level system with and without faults. The FPGA board is driven by a host computer where the other modules and the user interface run.

To efficiently determine the behavior of the circuit when a fault arises, the FPGA board emulates an instrumented version of the circuit, which allows both the injection of each fault and the observation of the corresponding faulty behavior.

The fault-injection manager is composed of the following modules.

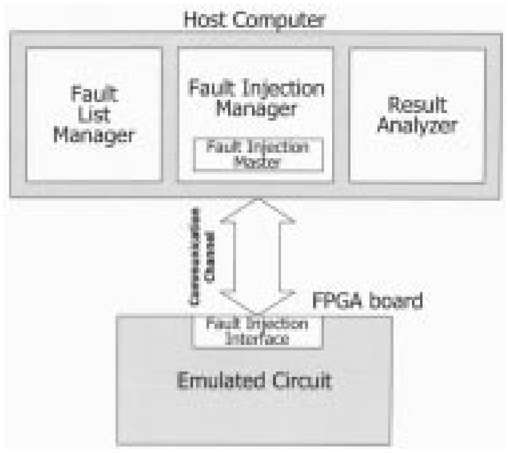

Fig. 1. Architecture of the proposed fault-injection system.

1) Fault Injection Master: It is implemented as a software module running on the host computer, and it is in charge of:

1) downloading to the FPGA board the instrumented circuit description;
2) setting up the environment for the fault-injection campaign;
3) repeatedly accessing the fault list, selecting a fault, and sending the information for its injection (in terms of fault location) to the FPGA board;
4) launching the circuit emulation by providing the input stimuli and the triggering signal for the injection of the fault;
5) retrieving from the board the information about the faulty system behavior.

2) Fault-Injection Interface: It is implemented as a hardware module on the FPGA board, and it is in charge of receiving and executing the commands issued by the fault-injection master.

The architecture of the whole environment is summarized in Fig. 1.

C. Circuit Instrumentation

A software tool *(Instrumenter)* is in charge of modifying the gate-level circuit description and creating an instrumented version of the circuit that will be then downloaded on the FPGA board. The Instrumenter is also in charge of the synthesis of the fault-injection interface.

The circuit modifications we introduce allow transient faults to be easily injected in the circuit flip-flops; moreover, the method implements an effective way to trigger the occurrence of faults at the proper injection time and for supporting the observation of the faulty behavior.

The architecture we devised, reported in Fig. 2, is based on adding to the original circuit a register (named `Mask Chain`) storing the binary information about which flip-flop(s) should be affected by the fault and an *ad hoc* logic for performing the fault injection. The signal `inject` controls the fault injection: at the injection time (i.e., when the fault-injection master asserts the `inject` signal, as described in the following section), the fault-injection master asserts it to force the occurrence of the selected bitflip(s).

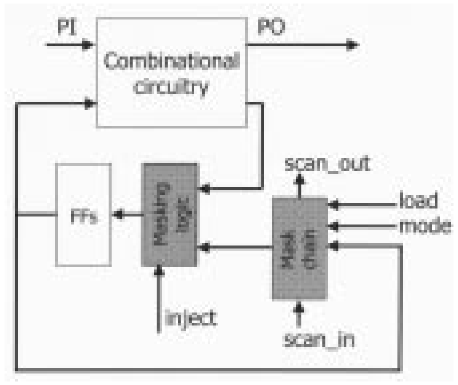

Fig. 2. Instrumented version of the circuit.

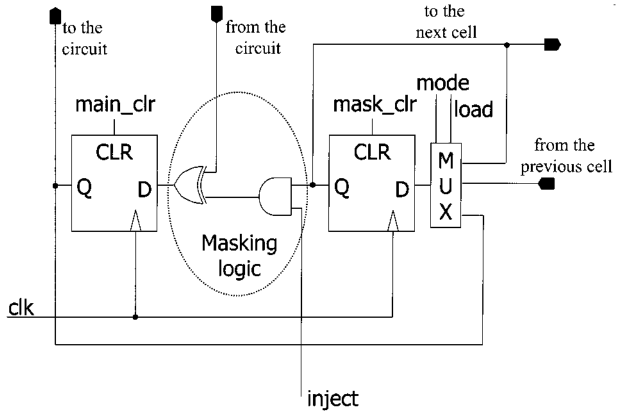

Fig. 3. Schematic of the flip-flop cell used to instrument the circuit under analysis.

The modules added to the circuit are the following.

1) Module `Mask Chain`*:* Each bit in `FFs` is associated to a bit in `Mask Chain`, which is a parallel- and serial-load register (the load and mode signals control the operation of the register). The register operates as follows.

During the experiment initialization phase, it works as a shift register and loads a bitstream coming over the `scan_in` signal. The register holds a "1" in the positions corresponding to the flip flop(s) to be flipped at the injection time, "0"s elsewhere.

At the injection time, each bit in `Mask Chain` set to one produces a bit-flip, i.e., the corresponding `FFs` module complements the value coming from the `Combinational Circuitry`.

The `Mask Chain` may load the content of the `FFs` module. The state of the circuit can then be read through the `scan_out` signal by operating the module as a shift register. This feature may be exploited at the end of each experiment to better classify fault effects.

2) Module `MaskingLogic`*:* It is the combinational logic in charge of possibly complementing the output of the `Combinational Circuitry` that is loaded into the `FFs` module. The behavior of `Masking logic` depends on the contents of the `Mask Chain` module and on the `inject` signal.

Each memory element is implemented as described in Fig. 3; the figure includes one element of `FFs`, the corresponding element in the `Mask Chain`, and the `Masking logic`.

```
for every fault $F_j$ {
  reset the circuit
  load Mask Chain
  i = 0
  while( $T_i$ < $T_{MAX}$) {
    apply vector $V_i$
    if( $T_i$ == Injection Time ($F_j$) )
      inject = 1
    else
      inject = 0
    emulate the circuit for one clock cycle
    observe output
    classify fault effects
    i = i+1
  }
  read circuit FFs
}
```

Fig. 4. Pseudocode of the fault-injection procedure.

D. The Fault-Injection Campaign

The *fault-injection campaign* is composed of several *fault-injection experiments*, one for each fault we intend to inoculate in the system under analysis.

The fault-injection campaign is composed of the following steps.

1) The fault list manager generates the list of faults to be injected during the campaign.
2) The circuit description is instrumented according to the transformations described in Section II-C.
3) The FPGA device is loaded with the instrumented circuit description.
4) The FPGA-based system is exploited to simulate the fault-free circuit: the output values at each clock cycle are recorded. The obtained output trace is the reference trace we use to classify fault effects.
5) The fault-injection manager executes the procedure described in Fig. 4. In particular, it executes the following phases during each fault-injection experiment.
   a) *Initialization:* the mask representing the fault to be injected is loaded in the `Mask Chain` module.
   b) *Emulation and injection:* the emulated circuit is fed with a stream of input vectors, and the fault is injected at the proper time; the output at each clock is recorded and sent to the host.
   c) *Latent fault analysis*: at the end of each fault-injection experiment, the content of the `FFs` module is loaded in the `Mask chain` module and then sent to the fault-injection master through the `scan_out` signal.

Our proposed method guarantees the accessibility of every memory element in the circuit during fault injection and allows a detailed result analysis. Moreover, it allows a time resolution of one clock cycle: the fault-injection master can inject faults at any clock cycle by triggering the `inject` signal at that cycle and observing the system behavior at any clock cycle.

III. Implementation

We developed a prototype of the fault-injection system described in the previous section using the ADM-XRC development board [15], equipped with a Xilinx Virtex V1000E FPGA. The development board supports a peripheral component interconnect (PCI) bus and is inserted in a standard Pentium-class PC, which manages the board as a memory-mapped device.

To modify the emulated circuit description to support fault injection, we developed the Instrumenter tool that automatically inserts the circuit structures for implementing the instrumented model. In particular, each memory element in the circuit is replaced with the cell outlined in Fig. 3. The memory cells are connected to form a parallel-/serial-load register, and their behavior can be controlled through ad hoc control signals.

The hardware/software fault injection manager is composed of the following modules.

1) The Fault-Injection Master: It amounts to 500 lines of C++ code; it is in charge of implementing the software module (running on the host), implementing the procedure described in Fig. 4.

2) The Fault-Injection Interface: It has been mapped on the Xilinx FPGA along with the circuit under evaluation; it is a simple finite-state machine that recognizes and executes the commands issued by the fault-injection master over the PCI bus. We used a memory-mapped protocol to control the operations of the Xilinx, providing a series of addresses to remotely guide from the host PC all steps of the fault-injection experiments. In particular, the fault-injection interface captures the PCI bus cycles directed toward the ADM-XRC board. It executes one of the following commands depending on the requested operation (read or write) and the specified address.

1) *Reset:* the instrumented circuit clock is disabled and all the flip-flops are set to the reset state.
2) *Load Mask Chain:* the scan chain is loaded with the values specifying the fault to be injected. This phase takes a number of clock periods to complete, which depends on the number of circuit flip-flops.
3) *Apply Vector:* the circuit primary inputs (PIs) are fed with a given vector.
4) *Step:* the circuit executes one step and computes the output values and the next state, depending on the given input vectors and the current state.
5) *Fault Injection:* the `inject` signal is activated. The circuit outputs and state will possibly exhibit a faulty behavior.
6) *Output Read:* the values on the circuit primary outputs (POs) are read.
7) *State Read:* the content of the circuit flip-flops is loaded in the `Mask Chain` and sent to the fault-injection master.

The instrumented circuit, including the fault-injection interface, is described in an automatically generated VHDL code. We then used typical development tools to obtain the stream needed to program the Xilinx device. Synplicity is used to map the description to the Xilinx device, while the proprietary tool Alliance is then employed to perform place, route, and timing analysis of the implemented design. The final result is a valid Xilinx configuration bit stream, which can be used to program the device.

After programming has taken place, the behavior of the system is completely controlled by the fault-injection master, which uses the basic functionality offered by the aforementioned addressing scheme to flexibly direct all phases of the fault-injection experiments.

The entire procedure can be modified according to different fault-injection schemes. For example, it is easy to sample the simulation output on given periods only (instead of comparing the entire output trace) or to support the injection of multiple bit-flips.

IV. Performance Analysis

In this section, we evaluate the performance of our FPGA-based approach from a theoretical point of view. Assuming an infinite bandwidth between the host computer and the FPGA board, the upper bound T_{FI} to the time for performing an injection campaign is defined as follows:

$$T_{\mathrm{FI}} = f \cdot (n_{\mathrm{FF}} \cdot (t_a + t_{\mathrm{clock}}) + N \cdot (2 \cdot t_a + t_{\mathrm{clock}})) \qquad (1)$$

where

f	number of faults injected during the fault-injection campaign;
N	number of vectors composing the workload used during injection experiments;
n_{FF}	number of memory elements in the circuit under analysis;
t_a	accounts for the time required to exchange information among the host computer and the FPGA board. It is thus the time required for transferring one input vector (output response) word to (from) the FPGA board and the host computer, as well as the time required for writing/reading the content of one bit in the `Mask chain`;
t_{clock}	period of the clock signal of the emulated circuit. It thus represents the time required for evaluating one input vector, as well as the time required for shifting the `Mask chain` by one position.

Equation (1) is the upper bound to the injection time since the product $n_{\mathrm{FF}} \cdot (t_a + t_{\mathrm{clock}})$ accounts for the time required for shifting the whole `mask chain`. During shift operations, we set up the information concerning the next fault to be injected, and we read the content of the circuit flip-flops at the end of the previous injection experiment. Moreover, the term $N \cdot (2 \cdot t_a + t_{\mathrm{clock}})$ is obtained under the hypothesis that the circuit outputs are observed at every clock cycle, thus by summing the following terms:

$N \cdot t_a$	time for transferring N input vectors from the host computer to the FPGA board;
$N \cdot t_a$	time for transferring N output responses from the FPGA board to the host computer;
$N \cdot t_{\mathrm{clock}}$	time spent by the emulated circuit to process N input vectors.

Under the hypothesis of having an infinite bandwidth between the host computer and the FPGA board, (1) suggests that the duration of fault-injection campaigns depends on the number of circuit memory elements, their clock frequency, the number of injected faults, and the number of vectors in the workload. Therefore, the duration does not depend on the circuit size (i.e., the number of gates composing the circuit).

In case of limited bandwidth, (1) should be modified in order to account for the reduced performance of the communication channel. When the implementation described in Section III is considered, which exploits a PCI bus to transfer a 32-bit word between the host computer and the FPGA board in a time equal to t_{pci}, T_{FI} becomes

$$T_{\text{FI}} = f \cdot \begin{pmatrix} 2 \cdot n_{\text{FF}} \cdot (t_{\text{pci}} + t_{\text{clock}}) + N \cdot t_{\text{pci}} \\ \cdot \left(\left\lceil \frac{n_{\text{PI}}}{32} \right\rceil + \left\lceil \frac{n_{\text{PO}}}{32} \right\rceil \right) + N \cdot t_{\text{clock}} \end{pmatrix} \tag{2}$$

where

t_{pci}	time for transmitting one 32-bit word over the PCI bus connecting the host computer with the FPGA board;
n_{PI}	number of circuit input ports;
n_{PO}	number of circuit output ports.

The communication protocol is half-duplex; as a result, two shift operations of whole `mask chain` are required: one for reading the content of the circuit flip-flops and one for setting up the information concerning the new fault to be injected.

As a result, the duration of fault-injection campaigns depends on the number of memory elements in the circuit, the number of circuit input and output ports, the number of injected faults, and the number of vectors in the workload. In our implementation, t_{pci} is significantly higher than t_{clock}; thus the injection time is dominated by the communication time. As a result, the efficiency of our approach could be further improved by adopting a faster communication channel between the host computer and the FPGA device.

V. Experimental Results

In this section, we describe the experiments we performed in order to assess the effectiveness of our approach. The goal of the experiments is twofold: to measure the speedup provided by our approach with respect to alternative ones and to show the versatility of the proposed approach in terms of supported fault model.

We selected a set of benchmark circuits and then performed several fault-injection campaigns exploiting different tools: the FPGA-based tool we described in this paper and a register transfer (RT)-level simulation-based tool. For each campaign, we recorded its duration. The results (in terms of fault classification) are the same no matter the adopted approach.

All the simulation experiments have been performed on a Sun UltraSparc running at 300 MHz and equipped with 256 MB of RAM.

A. Adopted Benchmark Circuits

To compare the time performance of the different approaches, we considered some neutral circuits coming from the ITC'99 benchmark suite [16]. The adopted benchmarks represent different types of circuits in terms of size and characteristics. However, they lack some of the characteristics typical of safety-critical circuits; in particular, they do not embed any EDM. We thus developed a new benchmark, b14_TMR, which implements a triple modular redundancy (TMR) scheme by using three copies of the b14 benchmark and a voter. Additional outputs signal the occurrence of an error inside the circuit. The circuit's characteristics are reported in Table I.

TABLE I
Benchmark Characteristics

Name	Gates	PIs	POs	FFs
b14	4,787	33	54	245
b15	8,922	37	70	449
b14_TMR	15,406	33	58	735
b17	24,195	38	97	1,415

TABLE II
FPGA Occupation

Name	Original version [%]	Instrumented version [%]	Overhead [%]
b14	7	10	42.8
b15	15	18	20.0
b14_TMR	22	28	27.3
b17	40	44	10.0

TABLE III
Effects of Single Bit-Flip on Adopted Benchmark Circuits

Name	Silent [%]	Latent [%]	Failure [%]	Detected [%]
b14	0,0	42,9	57,1	0,0
b15	0,0	79,4	20,6	0,0
b14_TMR	56,5	28,3	0,0	15,2
b17	1,2	87,9	10,9	0,0

Table II reports the percent occupation of the Xilinx V1000E device when the benchmark circuits are mapped on it for being emulated. We also reported the percent occupation of the instrumented version we used to perform the fault-injection experiments and the overhead with respect to the original version. This overhead mainly depends on two factors: the size of the fault-injection interface, which is independent of the circuit size, and that coming from the substitution of the memory elements with the cells of the `Mask Chain`, which depends on the number of flip-flops. It is worthwhile to underline that the overhead figures are important only during the selection of a suitable FPGA device. The final circuit does not include any structure for fault-injection purposes; as a result, our method does not introduce any overhead in the final product.

As far as timing is concerned, in the current version we did not stress the optimization process, and the Synplicity-Alliance tool chain allowed an operation frequency equal to 20 MHz. Faster emulation frequencies could be obtained by letting the optimization tool run for a longer period.

Table III summarizes the effects of the injection of 100 000 randomly selected faults (in terms of both memory element and injection time) when the workload is composed of 100 functional vectors; fault effects have been classified according to the categories introduced in Section II-A.

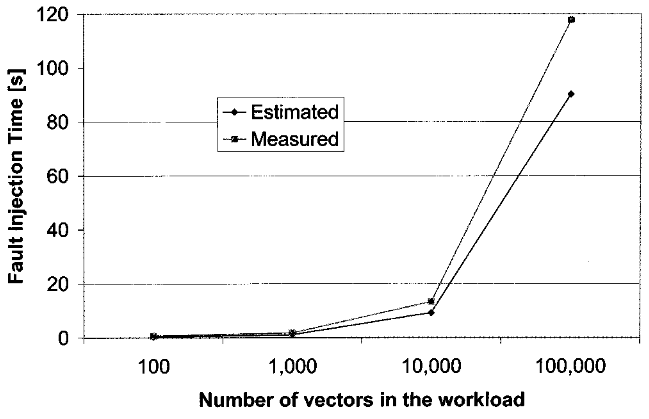

Fig. 5. Comparing expected and measured injection time.

B. Validation of the Theoretical Analysis

We performed some preliminary experiments to assess the soundness of the theoretical performance analysis of Section IV. The b17 benchmark has been used for these experiments; 1000 faults have been injected for each experiment, while the workload was composed of a number of random vectors ranging from 100 to 100 000. Fig. 5 plots the time for performing the experiments we measured as well as that computed by means of (2). As one can observe, the measured data match the foreseen data well.

C. Comparison With RT-Level VHDL Simulation

Comparisons with an RT-level VHDL simulator are intended to assess the benefits stemming from our FPGA-based approach when it is inserted in a typical industrial design environment. We thus compared our FPGA-based tool with a fault-injection environment based on a commercial VHDL simulator. This scenario reflects the current design practice, where VHDL simulators are often used to inject transient faults on RT-level models of the designed systems. Faults are injected into VHDL signals and variables that will correspond to memory elements after the synthesis process. The tool we adopted for the purpose of this paper, described in [17], has been developed at our institution and is based on the Modeltech VHDL simulator, version 5.1. All the experiments have been performed by injecting 100 000 randomly selected faults and considering a workload composed of the 100 functional vectors already used in the experiments of Section V-A. Table IV reports the time required by our FPGA-based tool and compares it with an RT-level simulation-based environment.

The experimental results show that the attained speedup is about four orders of magnitude, thus demonstrating the very high efficiency of the proposed approach. Analyzing the results, we can also state that the attained speedup is relatively independent of the circuit size, as anticipated by the theoretical analysis of Section IV.

D. Supporting Multiple Bit-Flips

To evaluate the versatility of the proposed approach in terms of supported fault model, we performed further fault-injection campaigns. In particular, by exploiting the FPGA-based tool, we analyzed the effects of multiple bit-flips injected in the B14_TMR memory elements. This analysis simulates a harsh environment where faults are likely to accumulate in the circuit memory elements. The results we obtained are reported in Table V. Each fault-injection campaign is composed of 100 000 faults, while the workload is that used in Section V-A. Thanks to the efficiency of the FPGA-based approach, we injected the faults in less than 4 min. As expected, the benchmark circuit is able to tolerate the presence of a single fault, while faults with greater multiplicity may originate failures, thus requiring a more complex hardening mechanism.

TABLE IV
FPGA-BASED APPROACH VERSUS RT-LEVEL VHDL SIMULATION

Name	FPGA Time [sec]	VHDL Time [sec]	Speed-up
b14	10	291,170	29,117
b15	14	434,091	31,274
b14_TMR	25	356,650	14,186
b17	14	625,065	45,859

TABLE V
FAULT EFFECT ANALYSIS FOR THE B14_tmr BENCHMARK

Number of affected memory elements	Silent [%]	Latent [%]	Failure [%]	Detected [%]
1	56.53	28.26	0.00	15.21
2	46.98	23.48	0.16	29.38
3	38.46	19.22	0.42	41.90
4	31.06	15.53	0.73	52.68

VI. CONCLUSION

This paper presented a method for performing fault-injection campaigns that is based on the adoption of an FPGA device for speeding up fault-injection experiments in VLSI circuits. A major novelty in the proposed approach lies in the circuit instrumentation approach adopted to inject transient faults in the circuit emulated through the FPGA. The proposed transformations introduce an area overhead that ranges from 10% to 42% for the considered benchmark circuits. They allow a significant reduction in the time required to perform the experiments, thanks to the fact that they do not require FPGA reconfiguration. By exploiting a prototypical version of the described fault-injection environment, we were able to evaluate its effectiveness with respect to a state-of-the-art alternative approach. Speedup factors higher than four orders of magnitude have been observed when comparing our method with alternative state-of-the-art VHDL simulation-based fault-injection tools. Moreover, the approach showed a high degree of versatility, allowing the injection of multiple bit-flips in the circuit memory elements.

VII. FUTURE WORK

We are currently working toward improved versions of the approach presented here. Two main improvements are under way.

1) Adoption of a more efficient communication architecture. As pointed out in Section IV, the communication channel between the FPGA board and the host computer is critical to obtain high efficiency. We are currently designing a new version of the FPGA board, where high-speed RAM memory modules are used to store workloads and output responses. Preliminary results show that by minimizing the amount of information exchanged over the PCI bus, we can further improve the fault-injection system performance.
2) Extension of the method for allowing the performance of fault injection within processor cores. Commercial off-the-shelf processors are becoming widely used even for safety- or mission-critical applications. As a result, it is becoming increasingly important to have the possibility of analyzing the effects of faults inside the processor core. Our approach is suitable for performing such an analysis provided that a synthesizable description of the processor core is available. The method is particularly suitable for injecting faults in memory elements that cannot be accessed by traditional software-implemented fault-injection techniques, e.g., in the registers implementing the processor pipeline.

We are also evaluating the possibility of partitioning designs on more than one FPGA device in order to address larger VLSI designs as well.

References

[1] R. Velazco, S. Rezgui, and R. Ecoffet, "Predicting error rate for microprocessor-based digital architectures through C.E.U. (code emulating upsets) injection," *IEEE Trans. Nucl. Sci.*, vol. 47, no. 6, pp. 2405–2411, 2000.

[2] B. L. Bhuva, J. J. Paulos, R. S. Gyurcsik, and S. E. Kerns, "Switch-level simulation of total dose effects on CMOS VLSI circuits," *IEEE Trans. Nucl. Sci.*, vol. 8, pp. 933–938, 1989.

[3] N. Kaul, B. L. Bhuva, and S. E. Kerns, "Simulation of SEU transients in CMOS IC," *IEEE Trans. Nucl. Sci.*, vol. 38, pp. 1514–1520, Dec. 1991.

[4] M. P. Baze, S. Buchner, W. G. Bartholet, and T. A. Dao, "An SEU anlysis approach for error propagation in digital VLSI CMOS ASICs," *IEEE Trans. Nucl. Sci.*, vol. 42, pp. 1863–1869, Dec. 1995.

[5] E. Jenn, J. Arlat, M. Rimen, J. Ohlsson, and J. Karlsson, "Fault injection into VHDL models: The MEFISTO tool," *Proc. Fault Tolerant Computing (FTCS-24)*, pp. 66–75, 1994.

[6] T. A. Delong, B. W. Johnson, and J. A. Profeta III, "A fault injection technique for VHDL behavioral-level models," *IEEE Design Test Comput.*, pp. 24–33, Winter 1996.

[7] D. Gil, R. Martinez, J. V. Busquets, J. C. Baraza, and P. J. Gil, "Fault injection into VHDL Models: Experimental validation of a fault tolerant microcomputer system," *Depend. Comput.*, pp. 191–208, 1999.

[8] J. Boué, P. Pétillon, and Y. Crouzet, "MEFISTO-L: A VHDL-based fault injection tool for the experimental assessment of fault tolerance," *Proc. Fault-Tolerant Computing (FTCS-28)*, pp. 168–173, 1998.

[9] L. W. Massengill, A. E. Baranski, D. O. Van Nort, J. Meng, and B. L. Bhuva, "Analysis of single-event effects in combinational logic-simulation of the AM2901 Bitslice Processor," *IEEE Trans. Nucl. Sci.*, vol. 47, pp. 2609–2615, Dec. 2000.

[10] S. A. Hwang, J. H. Hong, and C. W. Wu, "Sequential circuit fault simulation using logic emulation," *IEEE Trans. Computer-Aided Design*, vol. 17, pp. 724–736, Aug. 1998.

[11] K. T. Cheng, S. Y. Huang, and W. J. Dai, "Fault emulation: A new methodology for fault grading," *IEEE Trans. Computer-Aided Design*, vol. 18, no. 10, pp. 1487–1495, Oct. 1999.

[12] L. Antoni, R. Leveugle, and B. Fehér, "Using run-time reconfiguration for fault injection in hardware prototypes," in *Proc. IEEE Int. Symp. Defect and Fault Tolerance in VLSI Systems*, 2000, pp. 405–413.

[13] R. Leveugle, "Fault injection in VHDL descriptions and emulation," in *IEEE Int. Symp. Defect and Fault Tolerance in VLSI Systems*, 2000, pp. 414–419.

[14] I. Gonzales and L. Berrojo, "Supporting fault tolerance in an industrial environment: The AMATISTA approach," in *Proc. IEEE Int. On-Line Test Workshop*, 2001, pp. 178–183.

[15] ADM-XRC PCI Mezzanine card User Guide Version 1.2. ALPHA DATA Parallel Systems Ltd. [Online]. Available: http://www.alpha-data.co.uk/

[16] F. Corno, M. Sonza Reorda, and G. Squillero, "RT-Level ITC 99 benchmarks and first ATPG results," *IEEE Design Test Comput.*, pp. 44–53, July–Aug. 2000.

[17] B. Parrotta, M. Rebaudengo, M. Sonza Reorda, and M. Violante, "New techniques for accelerating fault injection in VHDL descriptions," in *Proc. IEEE Int. On-Line Test Workshop*, 2000, pp. 61–66.