Autonomous Fault Emulation: A New FPGA-Based Acceleration System for Hardness Evaluation

Celia López-Ongil, *Member, IEEE*, Mario García-Valderas, *Member, IEEE*, Marta Portela-García, *Student Member, IEEE*, and Luis Entrena

***Abstract*—The appearance of nanometer technologies has produced a significant increase of integrated circuit sensitivity to radiation, making the occurrence of soft errors much more frequent, not only in applications working in harsh environments, like aerospace circuits, but also for applications working at the earth surface. Therefore, hardened circuits are currently demanded in many applications where fault tolerance was not a concern in the very near past. To this purpose, efficient hardness evaluation solutions are required to deal with the increasing size and complexity of modern VLSI circuits. In this paper, a very fast and cost effective solution for SEU sensitivity evaluation is presented. The proposed approach uses FPGA emulation in an autonomous manner to fully exploit the FPGA emulation speed. Three different techniques to implement it are proposed and analyzed. Experimental results show that the proposed Autonomous Emulation approach can reach execution rates higher than one million faults per second, providing a performance improvement of two orders of magnitude with respect to previous approaches. These rates give way to consider very large fault injection campaigns that were not possible in the past.**

***Index Terms*—Fault emulation, fault injection, fault tolerance, FPGA, reliability testing, SEU.**

I. Introduction

SAFETY-CRITICAL VLSI circuits working in radiation environments require designs hardened against SEU effects. As technology evolves towards smaller transistor sizes and reduced power supply, this requirement becomes a concern in many application areas, even at sea level [1]. New technologies also enable more complex circuits and higher operation frequencies that pose new challenges in the efficient design of hardened circuits.

A crucial task in the design of a safety critical circuit is the estimation of sensitivity to SEU effects, which in turn allows predicting the circuit error rate [2]. Estimation of SEU sensitivity is required, first of all, to identify weak areas that must be hardened and then to validate the correctness and effectiveness of the safety critical design. To this purpose, it is essential to have techniques and tools that can provide accurate estimations of SEU sensitivity in a fast way. Such tools may allow a deeper exploration of the design space in order to satisfy reliability requirements with reduced design effort and cost.

Fault injection has been widely accepted to carry out SEU sensitivity analysis of ICs. This can be performed in several ways. Among them, a typical one is the injection of faults by exposing the circuit to radiation [3], [4]. This approach produces very realistic results, but requires expensive equipment and a manufactured circuit in order to perform a test. Therefore, it is mainly targeted to the certification of the final hardened circuit and cannot be used in early design stages. Other approaches, such as laser fault injection [5] or electromagnetic interference [6] suffer the same problem.

For an estimation of SEU sensitivity before the circuit is manufactured, simulation-based fault injection has been traditionally used [7]–[10]. Simulation-based fault injection can support many fault models and provides high flexibility in order to configure fault injection campaigns. Most techniques are built on top of commercial HDL simulators. Fault injection is achieved with the use of HDL simulator commands or modifying the HDL model to include the fault injection capability. The major drawback of this technique is the huge computational effort required to perform circuit simulation under a large number of faulty conditions. As new generations of circuits can include millions of gates in a single circuit, the capability of analyzing a significant number of faults and obtaining a comprehensive estimation of SEU sensitivity using simulation-based fault injection is reduced [11].

In the last years, emulation-based fault injection has emerged as a means to accelerate fault injection experiments. Emulation-based fault injection uses FPGA devices in order to emulate the behaviour of the circuit under test. Modern FPGAs include millions of equivalent gates and allow the emulation of large circuits at high speed. It must be remarked that the purpose of emulation-based fault injection is evaluating the SEU sensitivity of an ASIC design, not an FPGA design. SEU effects appear in the memory elements of circuits, which are the same in the ASIC and in the FPGA prototype. In the context of emulation-based fault injection, FPGAs are used as a means to perform SEU sensitivity evaluation of any digital design, whatever the target technology, by taking advantage of FPGA emulation speed.

Fault injection in FPGAs is performed either by reconfiguring the device to introduce the faulty behaviour [12] or by modifying the original circuit adding extra hardware to alter the state of the circuit [13]. In any case, a very intensive interaction is required between the emulator and the host computer. The host controls the injection and evaluation for every fault. This introduces a performance bottleneck due to the communication between the emulator and the host computer, which prevents

Manuscript received April 19, 2006; revised September 21, 2006. This work was supported by the Directorate of Research of Madrid Community Government, Spain (Code 07/0052/2003 2) and by the European Commission and Spanish Government under MEDEA+ Project (PARACHUTE-2A701) and PROFIT Project (CIRCE-FIT-330100-2005-60).

The authors are with Universidad Carlos III de Madrid, Departamento Tecnología Electrónica, 28911 Madrid, Spain (e-mail: celia@ing.uc3m.es; mgvalder@ing.uc3m.es; mportela@ing.uc3m.es; entrena@ing.uc3m.es).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNS.2006.889115

taking full advantage of the FPGA capabilities for fast hardware emulation. In [11], fault tolerance evaluation of microprocessors is performed, including fault injection and fault classification in the FPGA, but no further speedup of the fault injection process has been reported.

This paper proposes efficient mechanisms to overcome this performance bottleneck and achieve an extremely efficient fault injection system. To this purpose, an Autonomous Emulation system and several techniques for its practical implementation are proposed. The key of the proposed Autonomous Emulation is to minimise the communication between emulator and host during the fault injection execution and to optimize fault injection and fault classification tasks executed within the hardware. This is accomplished by including fault injection control functions inside the FPGA, generating or storing the stimuli and the fault list within the emulation hardware and storing also the fault classification in the emulation hardware. Thus, communication is only needed at the beginning of the process, to download the fault injection campaign configuration to the emulation board, and at the end, to retrieve the results.

Once the interaction between the host and the FPGA is minimised, the fault injection mechanisms can be optimized to reduce the time required to evaluate each fault, taking full advantage of FPGA emulation speed. In many cases, a fault effect can be categorized just a few clock cycles after the fault is injected. Then, if testbench application is stopped as soon as the fault can be classified and if circuit state is reloaded just before the injection time, a dramatic reduction in the total number of clock cycles used for a fault injection campaign is obtained. Therefore, there is no need to run the complete testbench for each fault, allowing huge savings in computational effort. Using this approach, the efficiency of emulation can be increased typically by about two orders of magnitude with respect to existing approaches, reaching rates in the order of 1 $\mu\mathrm{s/fault}$. These rates give way to consider very large fault injection campaigns that were not possible in the past.

The paper is organized as follows. Section II reviews the fundamentals of emulation-based fault injection. Then, the proposed Autonomous Emulation system is described in Section III. Section IV presents the experimental results. Finally, Section V states the conclusions of this work.

II. Transient Fault Emulation

The fault tolerance evaluation process consists in checking the response of a circuit in the presence of faults, by comparing the behaviour of both the fault free and the faulty circuit. To this purpose, faults are injected into a model of the circuit while it is executing a workload, to check their effects. This paper is focused on the behavioural effects produced by Single Event Upsets (SEU). The most commonly used fault model for SEU effects is the bit-flip, which affect circuit memory elements [16].

According to the circuit response after the injection of a fault, faults can be classified into several categories. At least, three basic categories are generally considered:

- A fault that induces wrong circuit behaviour is classified as a *failure*. A wrong behaviour usually means wrong or delayed values at the circuit outputs.

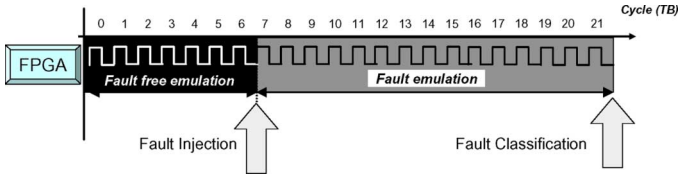


Fig. 1. Emulation of a fault in an FPGA.

- A fault that does not produce a faulty behaviour during the whole testbench, but leaves the circuit in a different state than the fault free circuit, is classified as a *latent* fault. Although it has not produced an observable error at the checking time, it could produce it in the future.
- A fault whose effects disappear completely is considered a *silent* fault.

Injecting a fault according to the bit-flip model involves changing the value of a memory element. The fault may produce different effects if applied to a different memory element or if injected in a different clock cycle. If just single faults are considered, the fault tolerance evaluation problem is two-dimensional. Being F the number of flip-flops in a circuit, and C the number of cycles of the testbench, the complete set of single faults is thus composed of FxC faults: a fault can be injected in any of the flip-flops, and for every flip-flop, it can be injected at any clock cycle. For each fault, the following basic steps are typically performed in order to classify the fault effect. These steps are graphically represented in Fig. 1.

1) Fault-free emulation in the hardware: run the testbench until the fault injection time.
2) Fault injection: inject the fault, by flipping the value of the faulty flip-flop.
3) Fault emulation in the hardware: continue testbench execution until the end of the testbench is reached.
4) Fault classification: check the circuit state in order to classify the fault effect.

If the testbench is completely applied to check every possible fault, the total number of clock cycles needed to perform the whole fault emulation process is C multiplied by the number of faults (FxC) that makes FxC^2. For large circuits as well as for long testbenches, the evaluation process can become extremely time-consuming. The usual approach to handle this complexity, regardless of the method used to perform the evaluation, is to sample the fault set in order to obtain statistical results. The main objective of this work is to reduce the time required for each fault evaluation, allowing much larger fault sets.

FPGA emulation has proven to be an effective method to reduce fault evaluation time [12]–[15], [17], [18]. In this case, an FPGA is used to emulate circuit behaviour under the control of a host that implements the fault injection campaign. FPGA-based emulation techniques have already been proposed to solve the fault tolerance evaluation problem, as an alternative to simulation solutions. These solutions profit from hardware speed, but maintain software flexibility.

Fault injection in an FPGA-based emulation system can be achieved by taking advantage of FPGA reconfiguration capabilities [12]. In this case, partial reconfiguration is used to modify flip-flop values. Applying emulation scheme shown in Fig. 1,

testbench execution is stopped when injection point is reached and the host reconfigures the FPGA in order to inject the fault. Finally, in the FPGA, execution is resumed up to the end of the testbench or until a failure is detected. Configuration readback can be used to retrieve the circuit state and to classify latent and silent faults in the host. In [12], readback is used to retrieve the circuit state at the injection time, and the fault is injected through the manipulation of the GSR signal (General Set Reset) of a Xilinx Virtex device. Every design flip-flop needs to be manipulated to preserve its value, except the injected flip-flop, whose value is flipped. Injection time with these techniques is usually high, as reconfiguration is quite a slow process. In [12], injection time is estimated to be about 100 ms for the worst case applied to a small Virtex device (XCV50). On the other hand, this approach depends heavily on the specific reconfiguration capabilities provided by a particular FPGA technology.

A different approach consists in using FPGA general purpose logic to support fault injection. In the instrumented circuit technique [13], [14], faults are injected by means of specific hardware located in the flip-flops of the circuit, named *instruments*, and connected in a scan-path chain, named *fault mask*. The fault mask determines the set of flip-flops to be affected by fault injection and is downloaded via scan path. At the required time, fault injection is activated from the host and executed by means of the instruments located at every flip-flop. A modification of this approach is presented in [15], where some optimization is proposed with respect to output comparison and fault injection. Instruments introduce an area overhead of one flip-flop and some additional logic per original flip-flop. Instrumented circuit technique has given rates ranging from 100 μs/fault and 830 μs/fault for short testbenches (less than 1,000 cycles) and long fault lists (100,000 faults) [13], [14]. On the other hand, in [15] fault rates are around 1 s/fault for long testbenches (10,000 cycles) and short fault lists (1,000 faults). However, these solutions require an intensive communication between the host computer and the FPGA for every fault emulated. Communication is needed to download the fault mask, to activate fault injection, to apply the stimulus and to retrieve output values. In [13], [14], fault classification is performed in the host, comparing correct outputs (generated by simulation) with faulty circuit outputs uploaded from the emulator. In [11], fault injection and fault classification is executed in the FPGA; however, no further optimization is done, giving rates of 5 ms/fault. Thus, the resulting emulation speed is dominated by the communication speed.

III. Autonomous Emulation System

Emulation-based techniques exploit the capability of an FPGA to emulate circuit behaviour at hardware speed. Since fault injection requires analyzing circuit behaviour for each injected fault, a large speed-up can be initially expected. However, with existing emulation-based solutions, the emulation process is interrupted every time the emulator needs to wait for the host to apply the stimuli, to inject a fault or to check the output values.

Autonomous emulation speeds up fault injection by minimising the communication between the FPGA and the host during the fault evaluation process execution. To this purpose, the critical control tasks of the fault injection process are implemented within the FPGA.

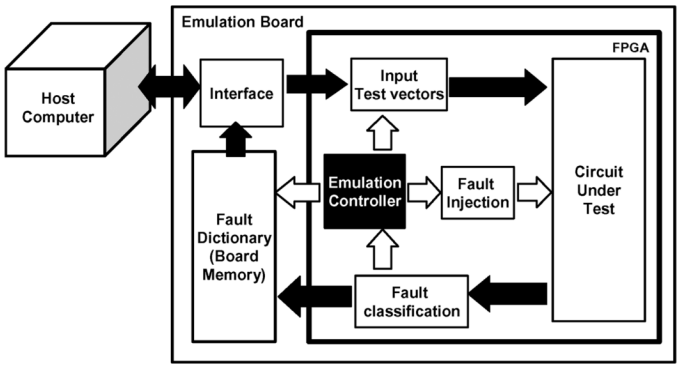


Fig. 2. Autonomous fault emulation system.

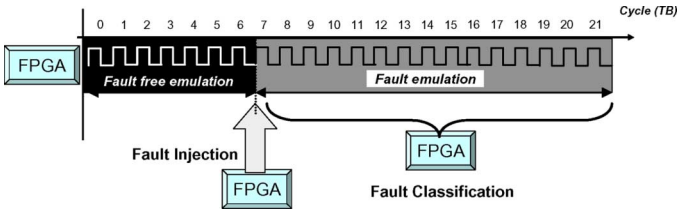


Fig. 3. Emulation of a fault in the Autonomous system.

Fig. 2 shows the general structure of the Autonomous Emulation system. The system uses an emulation board, which contains an FPGA, on-board RAM and an interface to the host. In addition to the circuit under test, the FPGA includes a testbench application module, a fault injection module, a fault classification module and an emulation controller. The emulation controller manages the whole process, initializing the circuit, applying input test vectors, injecting faults and enabling the comparison between expected outputs and faulty circuit outputs.

Available local RAM is used to store testbench inputs and fault classification results. Testbench inputs are stored in embedded RAM so that they can be applied to the circuit under test with no restrictions of bus widths and without off-chip delays. External RAM could be used if available embedded RAM is not enough. This will not imply a reduction in performance, given the current working frequencies of these memories. Fault classification is stored in external RAM (onboard RAM), accessible by the host computer after the fault injection campaign is finished.

With the Autonomous Emulation system, a typical fault injection campaign is executed as follows. First, the FPGA is configured. This step defines the circuit under test, the testbench and the fault list. Then, the FPGA emulation controller repeats, for each fault, the following steps (Fig. 3):

1) Set the circuit to the state previous to fault injection.
2) Fault injection.
3) Fault emulation until the fault is classified or until the testbench ends.

When the fault emulation campaign is finished, the host can upload the fault classification. Note that communication with the host takes place only at the beginning of the process, to download the fault injection campaign information, and at the

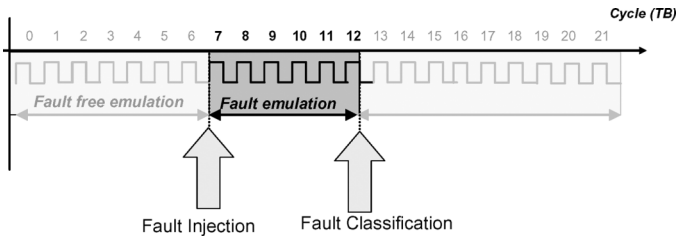


Fig. 4. Hardware cycles employed for the emulation of a fault in the Autonomous Emulation system.

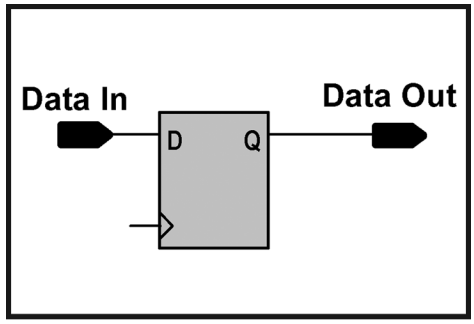
end, to retrieve the results. All tasks related to fault injection and classification are executed at hardware speed, and the FPGA is never idle waiting for the host to set up the next step in the process.

A. Optimization of Fault Emulation

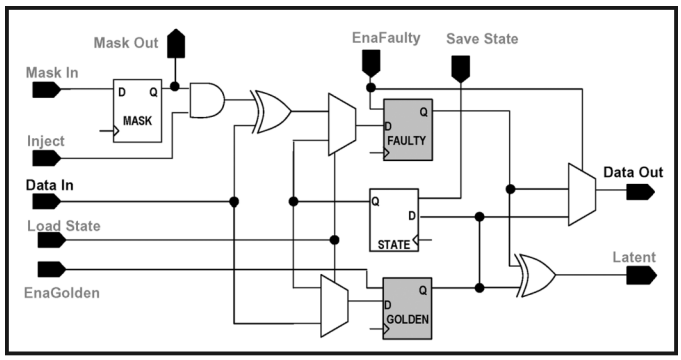
Once the host-FPGA communication is minimised, the time dedicated to the emulation of every fault can also be optimized. As described above, the emulation of a fault consists of three steps. The first step, fault free emulation in the Fig. 3, is performed in order to set the circuit to the required state before fault injection. This state can also be set without the need of emulation if it is stored elsewhere. In this case, the circuit can be set to the required state by loading circuit flip-flops with already saved values. Therefore, if state restoring takes fewer clock cycles than fault free emulation, the evaluation time is reduced.

On the other hand, fault emulation is executed after fault injection and it stops when a failure is detected (outputs are different in golden and faulty circuit) or when the testbench finishes. In the case of a silent fault, fault emulation could be stopped as soon as the fault effect disappears. However, this condition is difficult to apply for solutions based on host-FPGA communication. Faulty circuit state (in the FPGA) should be compared with golden circuit state (in the host) at every testbench cycle. Since this checking implies long periods of time for every testbench cycle, silent faults are usually classified at the end of the testbench, although its effect could have disappeared soon after fault injection. If emulation is managed in the system hardware, state comparison can be done for every testbench cycle and fault emulation can be stopped immediately after a fault becomes silent. This optimization allows a large reduction in the time required to emulate a fault which is finally classified as failure or silent. With these optimizations, the evaluation of each fault can be performed by applying the testbench just from the fault injection cycle to the cycle where the fault is classified (Fig. 4). The testbench will be applied up to the end only if the fault is latent. Therefore, the emulation time can be reduced notably.

In this work, the autonomous emulation paradigm allows a range of possibilities. Three techniques are proposed with different levels of optimization. They are named Time-Multiplexed, State-Scan and Mask-Scan. The three techniques are based on modifying the circuit under evaluation to support Autonomous Emulation. The main modification of the circuit under study is the substitution of all the internal flip-flops by



(a)



(b)

Fig. 5. (a) Original flip-flop. (b) Flip-flop replacement for the Time-Multiplexed emulation technique.

instrumented versions with fault injection capabilities. All techniques are based on the principle of Autonomous Emulation but every one may be useful for different applications.

B. Time-Multiplexed Technique

In Time-Multiplexed technique, every circuit flip-flop is replaced by the logic structure shown in Fig. 5. The idea underlying this structure is having two circuits working simultaneously, one to run the golden circuit and another one to run the faulty circuit. In order to save resources, these two circuits share their combinational logic by running in alternate clock cycles, performing time multiplexing. Full duplication is also possible, avoiding time multiplexing, at the expense of more combinational logic.

The flip-flop replacement contains a *golden* flip-flop and a faulty flip-flop, which are used to store the golden state and the faulty state, respectively. This way, the circuit states can be compared at every test cycle [*Latent* output in Fig. 5(b)], allowing the immediate detection of silent faults. Additional flip-flops are added to allow fault injection in the faulty circuit (*mask*) and to store the required circuit state (*state*) for next fault.

The *mask* flip-flop sets whether the value of the *faulty* flip-flop will be changed or not by the next fault injection. The different *mask* flip-flops of the circuit form a scan chain (*mask in* input and *mask out* output), named fault mask, so that the emulation controller can select where to inject faults (*Inject* input).

The *state* flip-flop is used to store the golden circuit state needed for the next fault injection. To this purpose, the *SaveState* signal is activated at the appropriate testbench cycle. When the current fault has been classified, the faulty and golden flip-flops

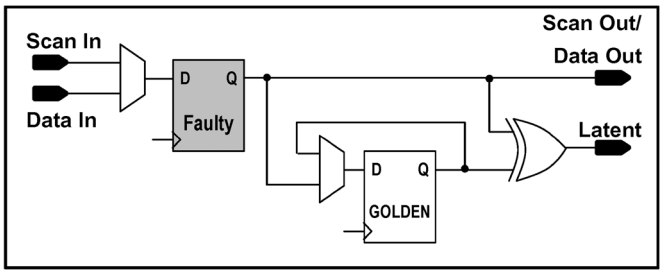


Fig. 6. Flip-flop replacement for the State-Scan emulation technique.

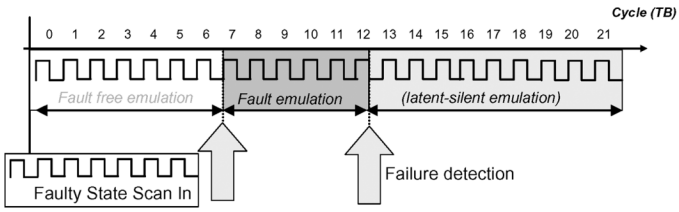


Fig. 7. Hardware cycles employed for the emulation of a fault in the State Scan technique.

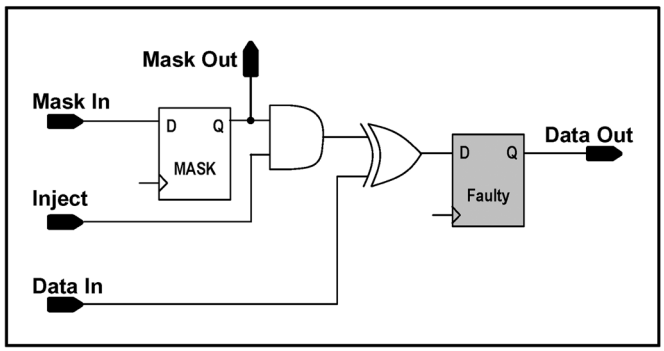


Fig. 8. Flip-flop replacement for the Mask-Scan emulation technique.

reload their values from the *state* flip-flop in order to start the next fault.

In this technique, all optimizations described above are implemented. Emulation can be restored from a previous state, so it is not necessary to restart it every time from the beginning. Also, emulation can be stopped immediately after a fault is classified as failure or silent, without waiting until the end of the testbench.

This technique implies extra logic resource usage, as shown in Fig. 5(b). The main overhead is in the number of flip-flops. However, current FPGA devices include a large amount of flip-flops, so that this is not crucial in many cases.

C. State-Scan Technique

State-Scan technique is a simplification of Time-Multiplexed technique devised to reduce area overhead. In State Scan technique, all original flip-flops in the circuit are now connected through a scan-path chain, so that the whole circuit state can be loaded into the circuit. Therefore, no fault mask is included. Fault injection is carried out by directly downloading a faulty circuit state. To this purpose, the fault list is processed in the host in order to generate the circuit state for every fault to be evaluated. Consequently, no *state* flip-flop is included either. The emulator stores the different faulty states in onboard RAM. These states are then inserted into the circuit by the emulation controller via scan-path. Although more RAM is required, two flip-flops are removed with respect to Time Multiplexed technique. Therefore, the expected area overhead, in terms of flip-flops, is now 100% instead of 300% (Time-Multiplexed technique).

Fig. 6 shows the flip-flop replacement to implement this technique. A single flip-flop substitutes the *faulty*, *state* and *mask* flip-flops. This flip-flop stores the faulty state of the circuit. In order to detect silent faults, a *golden* flip-flop is added. The golden flip-flop stores the final state of the circuit after a golden run. To this purpose, a golden run is first executed by the emulator at the beginning of the process and the final circuit state is stored in the golden flip-flops. Then, the circuit state is compared with the golden run state at the end of every fault emulation to determine whether the fault is latent or silent.

State Scan technique loads the state of the circuit at the injection time through a scan chain. Fault free emulation is substituted by faulty state scan in (Fig. 7) which is faster when the circuit has few flip-flops in relation with the number of testbench cycles. The number of cycles required to load the state is determined by the number of flip-flops. On the other hand, emulation is only stopped when a failure is detected. Silent faults can not be detected until the end of the testbench execution.

With this technique, the flip-flop number in the circuit under evaluation is only doubled with respect to the original circuit. On the other hand, a larger amount of RAM is usually required to store faulty circuit states for the fault injection campaign.

D. Mask-Scan Technique

Mask-Scan technique is another simplification of Time-Multiplexed technique, but it does not need so much RAM as the State-Scan technique. In Mask-Scan technique, only a *mask* flip-flop is added to every original flip-flop (Fig. 8). The *mask* flip-flops form a scan chain, with a mechanism similar to the instrumented circuit technique [13]. Neither *golden* nor *state* flip-flops are included in this technique. Circuit state is not stored for time optimization. Therefore, no RAM blocks are required. Mask Scan introduces less area overhead at the expense of increasing emulation time.

In this case, testbench application is always performed from the beginning, because there is no specific mechanism to load a particular state. Emulation is only stopped when a failure is detected or at the end of the testbench.

With the flip-flop replacement shown in Fig. 8, faults can only be classified into failure and non-failure categories. Silent and latent faults can not be distinguished unless the final circuit state is sent to the host for comparison, as the golden circuit state is not stored locally. A third flip-flop would be needed to be able to make the distinction.

E. Comparison of Techniques

In order to compare the time savings that can be obtained with the three proposed techniques, Figs. 9–12 show graphically

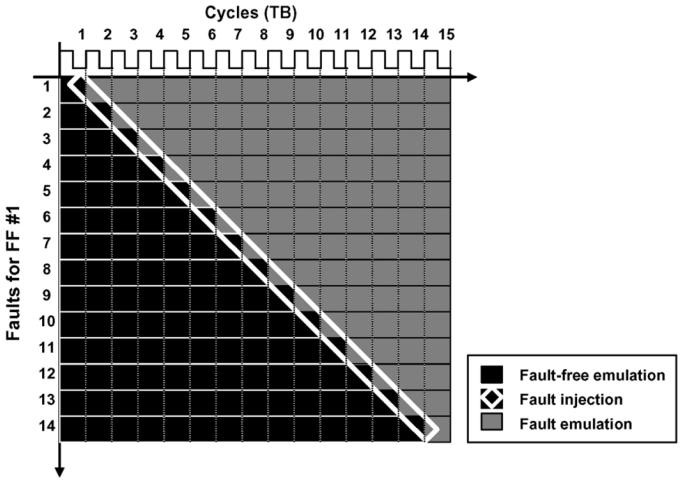


Fig. 9. Emulation of faults for FF1.

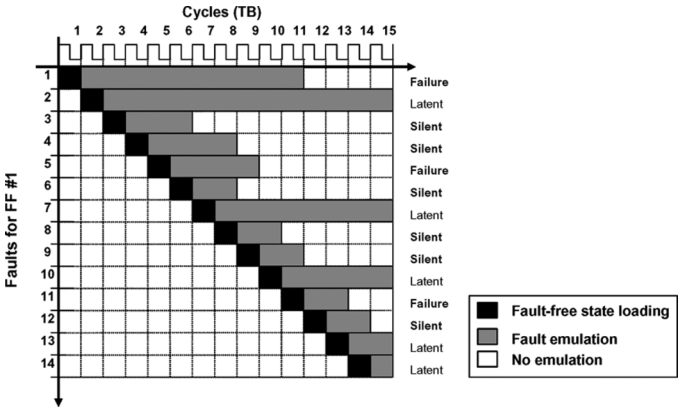


Fig. 10. Emulation of faults for FF1 with Time Multiplexed Technique.

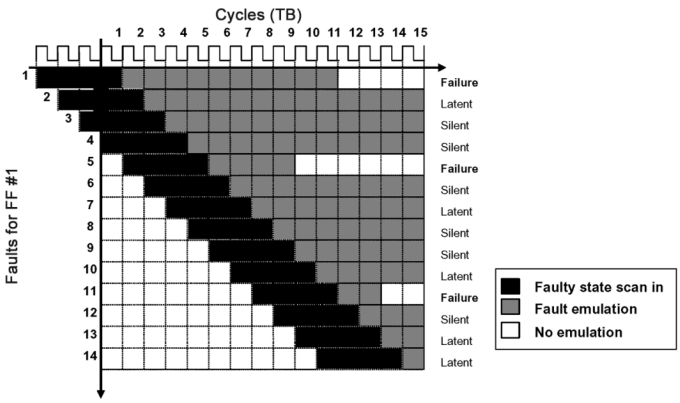


Fig. 11. Emulation of faults for FF1 with State Scan Technique.

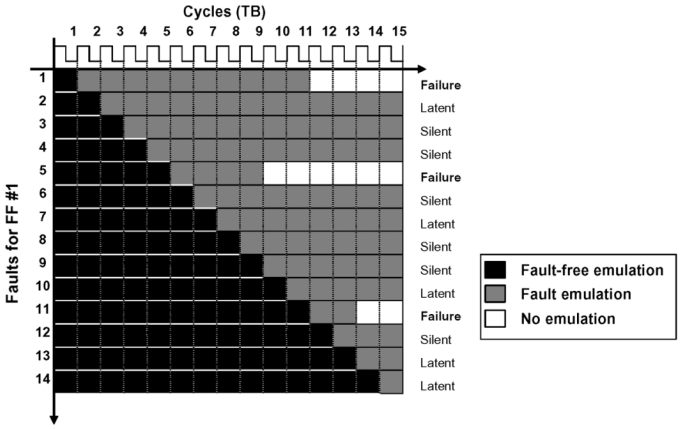


Fig. 12. Emulation of faults for FF1 with Mask Scan Technique.

the clock cycles needed for the emulation of faults injected in a single flip-flop FF1 of a generic circuit. In the figures, horizontal axis represents emulation clock cycles and vertical axis stands for number of faults.

Fig. 9 shows the general case of fault emulation with no optimization. Fault free emulation (black area) is executed until the injection time (white border box) and then, fault emulation (grey area) is run until the end of the testbench. Classification is performed at the end of the testbench execution. The total emulation time is proportional to the shaded area of the figure.

With Time Multiplexed technique (Fig. 10), fault free emulation is not required. Also, fault emulation is stopped as soon as the fault can be classified. Thus, the testbench is completely executed only for latent faults. The total emulation time is proportional to the shaded area in the figure, whereas the white area shows the time savings.

In State Scan technique, less area overhead is obtained but more time is required for fault emulation. Fault injection and state restoring is done via scan path from onboard memory at hardware speed (Fig. 11). Fault emulation is executed until the fault is classified as a failure or until the end of the testbench. Silent and latent faults are emulated until the end of the testbench. The time savings depend on the balance between fault free emulation and faulty state scan in (black area).

Finally, in Mask Scan technique (Fig. 12) circuit state cannot be restored, so the full testbench must be executed. Fault injection takes one clock cycle for the application of fault mask and fault emulation is performed in the same terms as in State Scan technique. Although this technique implies the smallest amount of area overhead, the execution of the emulation campaign is slower.

IV. Experimental Results

The proposed techniques for Autonomous Emulation have been tested with several benchmark circuits, in order to analyse the effectiveness of every technique. In addition, a real circuit from aerospace industry has also been evaluated. The experiments have been performed on a Celoxica RC1000 board [19], which includes a Xilinx Virtex-2000E [20] and 8 MBytes of onboard RAM. Both FPGA and RAM are accessible through the PCI bus. The circuit examples were described in VHDL at RT level. Leonardo Spectrum and Xilinx ISE were used as synthesis and back-end tools, respectively.

Table I shows the main characteristics of the circuits used in the experiments. For each circuit, the number of inputs, outputs and flip-flops are reported. B12, b14 and b15 circuits are from the ITC'99 benchmark suite [21]. In particular, circuit b12 is a one-player game (guess a sequence), circuit b14 is a subset of the Viper processor and circuit b15 is a subset of the 80386 processor. CIRCUIT_A is a real application from aerospace industry, where hardness is a crucial aspect. These circuits have

TABLE I
CHARACTERISTICS OF CIRCUITS UNDER TEST

	#Inputs	#Outputs	#Flip-flops
B12	5	6	119
B14	32	54	215
B14_TMR	32	54	323
B15	36	70	418
CIRCUIT_A	32	68	484
CIRCUIT_A_TMR	32	68	582

been chosen to test the performance and applicability of the Autonomous Emulation system and the advantages and disadvantages of the three proposed techniques. Besides, partially hardened versions of b14 and CIRCUIT_A have been included in order to check the system performance with fault tolerant circuits. Circuit b14_TMR has been obtained from circuit b14 by applying Triple Modular Redundancy on the primary outputs. CIRCUIT_A _TMR has been obtained from CIRCUIT_A by applying Triple Modular Redundancy on some selected critical outputs and internal signals. It must be noted that CIRCUIT_A was designed with a robust architecture, typical of aerospace applications.

A Celoxica RC1000 board used in the experiments includes a small sized FPGA device. Every circuit tested fit within this device. More complex circuits could be analyzed with bigger FPGAs, already available in the market. Also, additional on-board RAM can be used if needed for larger stimuli set and larger sets of faults.

The design flow for the experiments has been automated with tools developed by the authors, which interact with commercial simulation and synthesis tools. First, a netlist of the circuit under test (CUT) is obtained with a commercial synthesis tool. Then, the netlist is instrumented and linked to the emulator using automatic tools. The resulting circuit is re-synthesised and downloaded into the FPGA using the vendor back-end tool. A software application, developed by the authors, is used to start the fault injection campaign. In this moment, FPGA takes the control of fault injection campaign and executes all the fault emulation autonomously. Finally the mentioned software application uploads the classification results form the FPGA.

Input test vectors applied to ITC benchmarks are randomly generated through the simulation of an LFSR. Testbench stimuli are stored in the FPGA within embedded RAM. Although random test vectors provide higher number of latent faults than functional testbenches, these ones are difficult to obtain for those benchmarks. The influence of the testbench length in emulation time and area overhead has been analyzed by applying to the ITC benchmarks two sets of 160 and 600 stimulus vectors, respectively. For CIRCUIT_A, a functional testbench of 1500 cycles is used. In all the experiments the complete list of single faults is injected, i.e., all bit-flips in any flip-flop and at any clock cycle. Multiple bit-flips could be applied directly, just generating fault masks with multiple bits set.

A. Fault Classification

The fault classification results obtained with the Autonomous Emulation system are summarized in Table II. For each circuit, the total number of injected faults and the percentage of failure, latent and silent faults are presented. It must be noted that the Autonomous Emulation system provides a complete list of the classification result for each fault, so that a more detailed analysis can be obtained by post processing the results.

TABLE II
FAULT CLASSIFICATION FOR b12, b14 AND b14_TMR, b15, CIRCUIT_A AND CIRCUIT_A_TMR CIRCUITS

Circuits	# cycles	# faults	Fault Classification		
			%F	%L	%S
B12	160	19,040	29.8	55.6	14.6
	600	71,400	33.3	53.4	13.3
B14	160	34,400	49.2	4.4	46.4
	600	129,000	59.6	1.9	38.5
B14_TMR	160	51,680	16.0	3.2	80.7
	600	193,800	23.0	1.3	75.7
B15	160	66,880	19.9	52.7	27.4
	600	250,800	18.6	54.6	26.8
CIRCUIT_A	1,500	726,000	21.4	47.7	31.0
CIRCUIT_A_TMR	1,500	873,000	14.1	38.7	47.2

Circuit b14 presents a large number of silent and failure faults whereas in circuits b12 and b15 most faults are latent. As it was expected, for the partially hardened circuits (B14_TMR and CIRCUIT_A_TMR), failure faults decrease considerably in favour of silent faults, with respect to non-hardened versions. Note that hardened versions of b14 and CIRCUIT_A apply TMR on circuit outputs and some selected signals.

B. Emulation Performance

Emulation times are shown in Table III. For each circuit, the emulation time using the Mask Scan, State Scan and Time Multiplexed techniques are reported. In all cases, the clock frequency was set to a moderate frequency of 25 MHz for the sake of comparison. However, the clock frequency can be tuned to the maximum allowable frequency for each example, as reported by the FPGA synthesis tool. In the case of the ITC benchmarks, results for 160 and 600 testbench clock cycles are given. The average emulation time per fault is also listed.

Circuit b14 has already been evaluated with the Instrumented Circuit technique [13], obtaining a time average of 100 μs/fault, with a testbench of similar length. Comparing this result with the time average shown here [Table III(b)] it can be concluded that the proposed techniques provide a performance improvement up to two orders of magnitude.

According to the results, the Time Multiplexed technique is the fastest. It may become slightly slower when there is a large amount of latent faults and the testbench is large. The reason is that Time Multiplexed technique uses two clock cycles for each testbench cycle (to execute the golden and faulty circuit). With latent faults, the testbench must be run until the end, taking twice the number of cycles and making the process slower. This is the case of circuit b12 for the 600 cycle testbench, which shows 53.4% of latent faults. On the other hand, this example is well suited for the State Scan technique, as it has few flip-flops.

The performance benefits of Time Multiplexed technique are outstanding when there is a majority of faults classified as failure or silent. Therefore, hardened circuits are expected to produce a better performance in Time Multiplexed technique, as failure and latent faults become silent due to the hardening structures. For instance, fault emulation for circuit b14_TMR runs about 40 times faster with Time Multiplexed technique. For Mask Scan and State Scan, the effect is the opposite. Transforming failures

TABLE III(a)
TIME RESULTS FOR b12 CIRCUIT

B12	Execution time (ms)		Time average (µs/fault)	
	160	600	160	600
Mask Scan	108.70	1,450	5.7	20.3
State Scan	142.00	771	7.5	10.8
Time Mux	80.80	971	4.2	13.6

TABLE III(b)
TIME RESULTS FOR b14 CIRCUIT

B14	Execution time (ms)		Time average (µs/fault)	
	160	600	160	600
Mask Scan	141.11	2,200.0	4.1	17
State Scan	386.40	2,100.0	11.2	16
Time Mux	19.90	83.2	0.57	0.64

TABLE III(c)
TIME RESULTS FOR b14 HARDENED CIRCUIT

B14_TMR	Execution time (ms)		Time average (µs/fault)	
	160	600	160	600
Mask Scan	310.70	4,200	6.04	21.6
State Scan	838.90	4,400	16.3	23.06
Time Mux	24.00	99	0.47	0.51

TABLE III(d)
TIME RESULTS FOR b15 CIRCUIT

B15	Execution time (ms)		Time average (µs/fault)	
	160	600	160	600
Mask Scan	386.5	5,600	5.8	21.8
State Scan	1,300.0	6,800	19.9	27.1
Time Mux	226.8	3,300	3.4	13.1

TABLE III(e)
TIME RESULTS FOR CIRCUIT_A AND CIRCUIT_A_TMR

CIRCUIT A	Execution time (ms)		Time average (µs/fault)	
	Non-hardened	TMR	Non-hardened	TMR
Mask Scan	38,500	-	53	-
State Scan	31,900	-	44	-
Time Mux	23,800	23,300	33	27

into silent faults makes the evaluation slower, as for silent faults the testbench must be run until the end in these techniques. This aspect is shown in the results for circuits b14 and CIRCUIT_A, compared to their hardened versions.

Comparing Mask Scan and State Scan techniques, they differ in the time employed to reach the injection state. State Scan uses a fixed number of cycles to load the circuit state, equal to the number of circuit flip-flops. Mask Scan needs to run the testbench from the beginning to the injection cycle. Since faults are injected in every clock cycle, the mean time for this technique to reach the injection point is half the testbench length. Therefore, State Scan will be faster if the number of flip-flops is smaller than half the testbench length. This is demonstrated in the results of the experiments, comparing the rates for the two techniques and the two testbenches used. For circuits b12 and b14, State Scan is faster with the long testbench and slower with the short testbench. For b14_TMR and b15, where the number of flip-flops is higher, Mask Scan technique is better for the two testbenches.

TABLE IV
AREA RESULTS FOR FFS IN THE MODIFIED CUT

FFs	*Original*	*Mask_Scan*		*State_Scan*		*Time_Mux*	
		#FFs	*%*	*#FFs*	*%*	*#FFs*	*%*
b14	215	430	100%	430	100%	860	300%
b14_TMR	323	646	100%	646	100%	1,192	300%
b12	119	238	100%	238	100%	475	300%
b15	418	836	100%	836	100%	1,672	300%
Circuit_A	484	968	100%	968	100%	1,941	300%
Circuit_A_TMR	582	*1,164*	100%	*1,164*	100%	2,538	300%

TABLE V
AREA RESULTS FOR LUTS IN THE MODIFIED CUT

LUTs	*Original*	*Mask_Scan*		*State_Scan*		*Time_Mux*	
		#LUTs	*%*	*#LUTs*	*%*	*#LUTs*	*%*
b14	1,172	1,648	40.6%	1,555	32.7%	3,945	236.6%
b14_TMR	2,126	2,191	3.1%	1,982	-6.8%	4,448	109.2%
b12	362	673	85.9%	622	71.8%	1,233	240.6%
b15	2,322	3,795	63.4%	3,037	30.8%	7,368	217.3%
Circuit_A	932	2,167	132.5%	1,422	52.6%	4,385	370.5%
Circuit_A_TMR	955	1,632	70.9%	1,590	66.5%	4,767	399.2%

Fault classification has influence in the performance of every technique. It cannot be decided *a priori* which technique is going to be better for a given circuit. However, a decision can be taken according to the expected fault classification. Time Multiplexed will be usually better, especially when most faults are expected to be silent, like in hardened design. When a large amount of faults are expected to be latent, State-Scan or Time Multiplexed technique may be the faster depending on the existing relationship between the number of flip-flops and the testbench length [17]. This may be the case, for instance, when the testbench does not fully exercise the circuit functionality.

C. Area Overhead

In this section, the area overhead results are presented in order to complete the evaluation of the proposed techniques. As expected, each technique produces different overheads in terms of LUTs, FFs, Block RAM and onboard RAM. Thus, each technique has a different area-speed trade-off. This trade-off can be exploited by selecting the fastest technique that fits in the available device and offering the possibility of emulating larger circuits within the same device by using other techniques. The synthesis results for the three techniques implemented in the autonomous system developed are shown in Tables IV–IX. The number of LUTs and FFs and the relative area overhead (reported by the synthesis tools) are shown for the various circuits. Testbench lengths have negligible influence in FPGA area overhead. Therefore, area results are reported for b12, b14, b15 and b14_TMR with a testbench of 160 cycles. For each example, the area of the circuit under test after flip-flop replacement (Modified CUT) is shown along with the area for the complete emulation system including the emulation controller and the CUT. The required onboard and embedded RAM blocks (in Kbits) are presented for the complete emulation circuit.

The overhead due to the modification of the circuit under evaluation is proportional to the number of flip-flops in the original circuit. Time Multiplexed technique implies the highest area

TABLE VI
AREA RESULTS FOR FFS IN THE COMPLETE EMULATION SYSTEM

FFs		Original	Mask_Scan		State_Scan		Time_Mux	
			#FFs	%	#FFs	%	#FFs	%
b14	160	215	520	141.9%	539	150.7%	1,077	400.9%
	600		524	143.7%	545	153.5%	1,035	381.4%
b14_TMR	160	323	785	143.0%	804	148.9%	1,483	359.1%
	600		789	144.3%	810	150.8%	1,473	356.0%
b12	160	119	327	174.8%	365	206.7%	564	373.9%
	600		331	178.2%	372	212.6%	568	377.3%
b15	160	418	962	130.1%	991	137.1%	1,919	359.1%
	600		966	131.1%	998	138.8%	1,926	360.8%
Circuit_A		484	1,102	127.7%	1,099	127.1%	2,038	321.1%
Circuit_A_TMR		582	1,378	136.8%	1,297	122.9%	2,456	322.0%

TABLE VII
AREA RESULTS FOR LUTS IN THE COMPLETE EMULATION SYSTEM

LUTs		Original	Mask_Scan		State_Scan		Time_Mux	
			#LUTs	%	#LUTs	%	#LUTs	%
b14	160	1,172	1,586	35.3%	1,684	43.7%	4,255	263.1%
	600		1,592	35.8%	1,694	44.5%	4,291	266.1%
b14_TMR	160	2,126	2,073	-2.5%	2,099	-1.3%	4,861	128.6%
	600		2,079	-2.2%	2,110	-0.8%	4,702	121.2%
b12	160	362	710	96.1%	758	109.4%	1,481	309.1%
	600		716	97.8%	773	113.5%	1,496	313.3%
b15	160	2,322	3,523	51.7%	3,315	42.8%	7,845	237.9%
	600		3,529	52.0%	3,328	43.3%	7,793	235.6%
Circuit_A		932	1,821	95.4%	1,831	96.5%	4,798	414.8%
Circuit_A_TMR		955	2,070	116.8%	2,090	118.8%	5,184	442.8%

TABLE VIII
RAM REQUIREMENTS FOR ON-BOARD RAM IN THE COMPLETE EMULATION SYSTEM

Board RAM		Available (kbits)	Mask_Scan		Time Mux		State_Scan	
			# kbits	%	# kbits	%	# kbits	%
b14	160	65,536	33.0	0.1%	67	0.1%	7,289	11.1%
	600		126.0	0.2%	252	0.4%	26,573	40.5%
b14_TMR	160	65,536	50.5	0.1%	101	0.2%	19,380	29.6%
	600		189.3	0.3%	378	0.6%	72,675	110.9%
b12	160	65,536	18.6	0.0%	37	0.1%	2,200	3.4%
	600		69.7	0.1%	139	0.2%	8,200	12.5%
b15	160	65,536	65.0	0.1%	130	0.2%	27,431	41.9%
	600		244.5	0.4%	489	0.7%	102,867	157.0%
Circuit_A		65,536	708.9	1.1%	1,418	2.2%	385,687	588.5%
Circuit_A_TMR		65,536	851.5	1.3%	1,703	2.6%	513,281	783.2%

TABLE IX
RAM REQUIREMENTS FOR FPGA RAM IN THE COMPLETE EMULATION SYSTEM

FPGA RAM		Available (kbits)	Mask_Scan		Time Mux		State_Scan	
			# kbits	%	# kbits	%	# kbits	%
b14	160	640	13.40	2.1%	5.30	0.8%	13.40	2.1%
	600		50.40	7.9%	18.75	2.9%	50.40	7.9%
b14_TMR	160	640	13.40	2.1%	5.30	0.8%	13.40	2.1%
	600		50.40	7.9%	18.80	2.9%	50.40	7.9%
b12	160	640	2.00	0.3%	0.78	0.1%	2.00	0.3%
	600		7.60	1.2%	2.90	0.5%	7.60	1.2%
b15	160	640	16.56	2.6%	5.60	0.9%	16.51	2.6%
	600		62.20	9.7%	21.10	3.3%	62.20	9.7%
Circuit_A		640	146.48	22.9%	46.9	7.3%	146.48	22.9%
Circuit_A_TMR		640	146.48	22.9%	46.9	7.3%	146.48	22.9%

overhead for the circuit and for the complete emulation system. The area overhead due to the emulation controller depends on the number of flip-flops, the testbench length and number of circuit inputs and outputs, but it is small compared to the circuit size. Regarding to RAM requirements, they are very important in the State-Scan technique for the board RAM blocks. In some cases, it has been necessary to split the fault injection experiments because RAM required was larger than available.

Although the implementation of the Autonomous Emulation system represents a significant resource overhead, the size of current FPGA devices allows the successful implementation of the emulator for a wide range of circuits and applications within a single FPGA. For very large circuits, Autonomous Emulation can be implemented on multiple FPGAs.

D. Considerations on Scalability

The techniques proposed may be used for a wide range of circuits and applications, The benchmarks used to demonstrate the techniques may be considered small or medium size circuits, In order to use the proposed techniques with larger circuits and longer testbenches, the two main factors affecting the emulator implementation are the size of the circuit to test and the length of the testbench. A possibility that can be always considered to support larger circuits is the use of a hardware platform with a larger FPGA device or several of them, and a larger on-board memory.

Regarding testbench length, several possibilities may be considered when the testbench vectors exceed the available storage memory. Input stimuli are usually very repetitive, so it is fairly easy to use compression algorithms to increase the storage capability. It is also possible to generate input stimuli with a circuit, like a LFSR, instead of explicitly storing vector values. Finally, it is always possible to split the test in several parts and perform the injection campaign in several steps.

Two examples of larger size have been implemented using Time-Multiplexed technique. The first one is the CIRCUIT_A, but with a testbench of 100,000 cycles. A simple compression technique has been implemented to allow the storage of the complete vector set. The second example is a CORDIC core [22]. A 150,000 vector set has been tested, using an LFSR for vector generation. In both examples, the fault set has been split in several sections due to the limitations of the result storage memory; different sections imply different ranges of flip-flops and clock cycles in which faults are injected. These results, shown in Table X and Table XI, are provided to demonstrate the capability of implementing large fault injection campaigns, even with a small size hardware platform.

Table X shows that large fault injection campaigns have been carried out with a very high emulation speed. Both circuits have been emulated at 25 MHz. Several million faults have been emulated in a few minutes, whereas experiments shown in the literature usually test just a few thousand faults.

Regarding the fault classification of CIRCUIT_A experiment, a large increment in silent faults can be observed in relation with previous experiments of this circuit. This increment is caused by the application of a much longer testbench. CIRCUIT_A was designed using robust communication protocols, but they require some time to be effective and to be able to cancel faults.

Table XI shows that the resources used by the emulators are not unaffordable, specially taking into account that the hardware platform used could be considered small. CIRCUIT_A takes a higher amount of memory because test vectors are stored in RAM, whereas in the CORDIC emulator, test vectors are generated with an LFSR.

TABLE X
RESULTS FOR LARGE FAULT INJECTION CAMPAIGNS

Circuit	# inputs	# flip-flops	# cycles	# faults	Fault Classification			Speed (µs/fault)	Emulation time (s)
					%F	%L	%S		
CIRCUIT_A	32	484	100,000	48,400,000	8.46	0.02	91.52	1.55	75.2
CORDIC	51	865	150,000	129,750,000	83.92	0.01	16.07	1.07	140

TABLE XI
EMULATOR RESOURCE USAGE

	# flip-flops	# LUT4	# FPGA RAM	#board RAM
CIRCUIT_A	1,893	4,361	617.4 Kbits	92.3 Mbits
CORDIC	4,050	5,178	0	247.5 Mbits

V. CONCLUSIONS

This paper presents a new solution for improving the performance of SEU emulation in FPGAs. An Autonomous Fault Emulation system is proposed, which executes in the FPGA most of the tasks involved in a fault injection campaign. This approach allows taking full advantage of FPGA emulation speed and saving execution time by optimizing the fault injection process. Execution rates higher than 10^6 faults per second can be reached, providing a performance improvement of two orders of magnitude with respect to existing approaches. These rates give way to consider very large fault injection campaigns that were not possible in the past and to tackle the estimation of SEU sensitivity for large circuits. Moreover, flexibility is maintained because the fault injection campaign can still be configured from software running in a host, but fault emulation process is autonomously performed within the FPGA once the fault list is generated. Detailed analysis of the results could be done because the emulation process stores all the required information (fault classification, fault latencies, etc.) in on-board RAM, which is finally uploaded to the host.

Autonomous Emulation defines a tool in which several techniques are possible with different tradeoffs in terms of performance and area overhead. In this work, three fault emulation techniques have been presented and compared experimentally. The best solution depends on the circuit under test and the testbench, but in most cases Time Multiplexed technique is one order of magnitude faster than the two other techniques, State Scan technique is better for circuits with small number of flip-flops and long testbenches, and Mask Scan technique is better in the opposite case. Regarding resource usage, Time Multiplexed uses the highest amount for FPGA resources, but with less RAM blocks, whereas State Scan has the smallest area overhead.

The results demonstrate that Autonomous Emulation is a time and cost effective solution for transient fault emulation, due to the popularization of low cost FPGAs, with a large amount of available resources. It can be easily scaled to multi-FPGAs to support the efficient evaluation of very large circuits. The proposed approach is technology independent, since it does not rely on any particular FPGA configuration mechanism.

REFERENCES

[1] International Technology Roadmap for Semiconductors, 2001 Edition.

[2] 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, Dec. 2000.

[3] J. Karlsson, P. Folkesson, J. Arlat, Y. Crouzet, G. Leber, and J. Reisinger, "Application of three physical fault injection techniques to the experimental assessment of the MARS architecture," in *Proc. Int. Working Conf. Dependable Computing for Critical Applications*, Urbana/Champaign, IL, Sep 1995, pp. 150–161.

[4] R. Velazco, S. Karoui, T. Chapuis, D. Benezech, and L. H. Rosier, "Heavy ion test results for the 68020 microprocessor and the 68882 coprocessor," *IEEE Trans. Nucl. Sci.*, vol. 39, no. 3, pp. 436–440, Jun. 1992.

[5] R. Velazco, T. Calin, M. Nicolaidis, S. C. Moss, S. D. LaLumondiere, V. T. Tran, and R. Koga, "SEU-hardened storage cell validation using a pulsed laser," *IEEE Trans. Nucl. Sci.*, vol. 43, no. 6, pp. 2843–2848, Dec. 1996.

[6] F. Vargas, D. L. Cavalcante, E. Gatti, D. Prestes, and D. Lupi, "On the proposition of an EMI-based fault injection approach," in *Proc. 11th IEEE Int. On-Line Testing Symp.*, Jul. 2005, pp. 207–208.

[7] E. Jenn, J. Arlat, M. Rimen, J. Ohlsson, and J. Karlsson, "Fault injection into VHDL models: the MEFISTO tool," in *Proc. FTCS-24, Int. Symp. Fault Tolerant Computing*, 1994, pp. 66–75.

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

[9] V. Sieh, O. Tschäche, and F. Balbach, "VERIFY: evaluation of reliability using VHDL-models with embedded fault descriptions," in *Proc. 27th Int. Symp. Fault Tolerant Computing*, Jun. 1997, pp. 32–36.

[10] L. Berrojo, F. Corno, L. Entrena, I. González, C. López, M. Sonza, and G. Squillero, "An industrial environment for high-level fault-tolerant structures insertion and validation," in *IEEE VLSI Test Symp.*, Monterey, CA, May 2002, pp. 229–236.

[11] F. Lima, S. Rezgui, L. Carro, R. Velazco, and R. Reis, "On the use of VHDL simulation and emulation to derive error rates," in *Proc. 6th Conf. Radiation and its Effects on Components and Systems (RADECS'01)*, Grenoble, France, Sep. 2001.

[12] L. Antoni, R. Leveugle, and B. Feher, "Using run-time reconfiguration for fault injection in HW prototypes," in *Proc. IEEE Int. Symp. Defect and Fault Tolerance in VLSI Systems*, 2002, pp. 245–253.

[13] P. Civera, L. Macchiarulo, M. Rebaudengo, M. Sonza Reorda, and M. Violante, "FPGA-based fault injection techniques for fast evaluation of fault tolerante in VLSI circuits," in *Proc. Forum on Programmable Logic (FPL)*, Belfast, Northern Ireland, Aug. 2001.

[14] ——, "Exploiting circuit emulation for fast hardness evaluation," *IEEE Trans. Nucl. Sci.*, vol. 48, no. 6, pp. 2210–2216, Dec. 2001.

[15] A. Ejlali, B. M. Al-Hashimi, and S. Ghassem Miremadi, "Fast observation architecture for FPGA-based SEU analysis," in *Proc. 10th European Test Symp. (ETS'05)*, Talinn, Estonia, May 2005.

[16] I. González 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.

[17] C. López-Ongil, M. García-Valderas, M. Portela-García, and L. Entrena-Arrontes, "Autonomous transient fault emulation on FPGAs for accelerating fault grading," in *Int. On-Line Testing Symp.*, Saint-Raphael, France, Jul. 2005, pp. 43–48.

[18] M. Aguirre, J. N. Tombs, F. Muñoz, V. Baena, A. Torralba, A. Fernández-León, F. Tortosa, and D. González-Gutiérrez, "A FPGA based hardware emulator for the insertion and analysis of single event upsets in VLSI designs," in *Proc. Radiation Effects on Components and Systems Conf. (RADECS)*, Madrid, Spain, Sep. 2004.

[19] "*Celoxica RC1000 Hardware Reference Manual,*" ver. 2.3, 2004.

[20] [Online]. Available: www.xilinx.com

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

[22] [Online]. Available: www.opencores.org