A Methodology to Emulate Single Event Upsets in Flip-Flops using FPGAs through Partial Reconfiguration and Instrumentation

Felipe Serrano, Juan Antonio Clemente and Hortensia Mecha

***Abstract*—This paper presents a methodology to emulate Single Event Upsets (SEUs) in FPGA flip-flops (FFs). Since the content of a FF is not modifiable through the FPGA configuration memory bits, a dedicated design is required for fault injection in the FFs. The method proposed in this paper is a hybrid approach that combines FPGA partial reconfiguration and extra logic added to the circuit under test, without modifying its operation. This approach has been integrated into a fault-injection platform, named NESSY (*Non intrusive ErrorS injection SYstem*), developed by our research group. Finally, this paper includes results on a Virtex-5 FPGA demonstrating the validity of the method on the ITC'99 benchmark set and a Feed-Forward Equalization (FFE) filter. In comparison with other approaches in the literature, this methodology reduces the resource consumption introduced to carry out the fault injection in FFs, at the cost of adding very little time overhead (1.6 μs per fault).**

I. Introduction and Related Work

In the last few years, radiation effects on embedded systems, such as Aplication Specific Integrated Circuits (ASICs) and Field Programmable Gate Arrays (FPGAs) are becoming a major concern for the semiconductor industry due to the growing use of these kinds of devices in radiation hazardous environments and the increase in device count per chip [1], [2]. Radiations may cause soft errors known as Single Event Effects (SEEs), which alter the operation of the microelectronic components of embedded systems.

Several hardening techniques and manufacturing methodologies offer different trade-offs between SEE tolerance and flexibility/adaptability in the final implementation of the circuit. Typically, the most widespread alternative has been to manufacture digital circuits as ASICs. Important reasons justify this choice, such as power, area and performance. In addition, their typical operation frequency for highly reliable systems (in the order of just a few hundreds of MHz), makes these systems essentially vulnerable to Single Event Upsets (SEUs) on their sequential elements (flip-flops (FFs) and memory cells), while keeping the vulnerability of their combinatorial elements very low with respect to other technologies. Indeed, recent studies [3] indicate that, for technologies under 100 nm, the vulnerability of combinatorial logic exceeds that of the sequential one only for high-frequency systems (typically, on the order of several GHz [4], [5]).

This work was supported in part by the Spanish MCINN project TIN2013-40968-P, and by the "José Castillejo" grant.

Felipe Serrano is with INDRA Systems, Madrid, Spain, e-mail: felise01@ucm.es.

Juan A. Clemente and H. Mecha are with the Computer Architecture Department, Facultad de Informática, Universidad Complutense de Madrid (UCM), Spain, e-mails: ja.clemente@fdi.ucm.es, horten@ucm.es.

An interesting alternative to ASIC implementations is the utilization of reconfigurable devices, and especially FPGAs. The reason is that they present run-time reconfiguration, which makes it possible to change the functionality of the circuit *"on the fly"*. This feature is potentially interesting to save weight and space since it allows to multiplex several designs in one FPGA. However, these kinds of devices are usually more vulnerable to SEUs, because of the enormous size of their *configuration memory*. These are Static Random Access Memories (SRAM) that describe the functionality at the structural level of the circuit(s) implemented on the FPGA. For instance, a medium-sized FPGA, such as a Xilinx^{TM}Virtex-5 XC5VLX110T, contains 29,124,608 configuration bits, which represent 84% of the bits vulnerable to SEUs in a circuit implemented in this device (including both FFs (69,120 bits) and BRAMs (5,328,000 bits)) [6].

A common technique to study SEU effects on digital circuits consists in testing them under radiation [7]–[10]. However, because of the high cost and tidiness of these experiments [11], alternative approaches, such as fault simulation and emulation techniques have become such a popular alternative.

Simulation-based fault injection approaches [12] need a huge computational effort since they just simulate the execution of the circuit at the desired level (behavioral, structural...). Hence their capability for analyzing a significant number of faults on circuits with millions of gates is limited. Hence, emulation-based fault injection has emerged to accelerate the experiments. These techniques may target either ASICs and FPGA-based systems, but they have in common the fact that they use FPGAs to emulate the circuit under test.

Emulation-based fault injection methodologies can be classified into instrumentation-based [13]–[16] and reconfiguration-based [17]–[25]. The next two subsections elaborate on these two types of methodologies in greater details.

A. Instrumentation-based Fault Injection

These techniques involve adding fault injector circuits called *saboteurs* to each fault site [13]–[16]. They do not involve any time penalty additional to the execution of the testbench on the target circuit, once the modifications needed on the circuit have been made. However, their main limitation is that they introduce some area overhead, which may limit their applicability on large circuits.

The implementation of the *saboteurs* can be achieved by performing certain modifications in the initial netlist, or by using the commercial tools. For instance, the method presented in [13] needs 150 μs on average per bitflip, since it uses the serial Joint Test Action Group (JTAG) interface to communicate between the host computer and the FPGA.

Other two interesting instrumentation-based approaches are described in [14], [15] and [16]. Authors in [14] present the Hardware Description Language (HDL)-based fault injection tool *NETlist Fault Injection* (NETFI). This tool injects faults at the register-transfer level of a processor by adding extra hardware to the sensitive registers of a target processor. This approach is based on the modification of the built-in libraries of Xilinx$^{\text{TM}}$ [15]. It adds some extra hardware per FF, as well as a complex controller that steers the fault-injection process on the whole circuit. This controller requires as much memory as needed to store n m-bit words, n being the number of FFs in the design, and m being the number of entries in the testbench. In addition, extra hardware to implement the controller itself is also needed. Hence, even though no precise information is provided in [14] about the area overhead that this approach introduces, everything suggests that it is considerable.

On the other hand, in [16], three different techniques are presented, so-called *time-multiplexed*, *state-scan* and *mask-scan*, which offer different trade-offs between area overhead and performance. At best, they can achieve fault injection times on the order of just a few μs. However, these three techniques also involve an important area overhead. In this case, since all of them are very well documented in [16], it has been possible to compare them with the approach presented in this paper. More details about this point will be provided in Section IV.

B. Reconfiguration-based Fault Injection

Reconfiguration-based approaches consist of modifying the appropriate information in the configuration memory of the FPGA in order to inject a fault [17]–[25]. In these techniques, the reconfiguration process is the speed bottleneck, instead of the extra logic added by instrumentation-based ones. These techniques may be based either on total or partial reconfiguration.

Approaches based on total reconfiguration use the Global Set/Reset (GSR) line of the device for this purpose. This line sets/resets *all the FFs on the device* depending on the polarity of this line. This polarity is determined for each FF in the configuration memory through a configuration memory bit named INIT0/INIT1 [26]. Hence, they involve reading the state of all the FFs in the FPGA [19] (by means of a *Capture&Readback* operation [26]), modifying the INIT0/INIT1 bit of some of them according to the retrieved information, pulsing the GSR line, and restoring the configuration memory of its initial status. This process is very slow since it necessarily involves iterating on all the INIT0/INIT1 bits of all the FFs in the circuit. Indeed, [19] reports 916 seconds to inject 3000 bitflips in one FF and [17] reports 3.5 seconds needed for a FF fault injection in a Xilinx$^{\text{TM}}$Virtex-II FPGA.

Another interesting reconfiguration-based system is *Fault Tolerant - University of Seville Hardware DEbugging System* (FT-UNSHADES), which implements a *read-modify-write* approach [20], [21]. The reported time to inject and evaluate a fault is on the order of 100 ms (in this case, no information was found only for a single FF fault injection) and it uses the JTAG debugging port to this end. An enhanced version of this tool, named FT-UNSHADES2 [22], [23], speeds up the communications by using PCIExpress, rather than USB transactions. This allows reaching fault injection rates of up to 10,000 faults per second. In addition, it features fault injection on the reconfiguration memory of the FPGA, via the Select Map port. However, for what concerns fault injections on FFs, they continue using the approach based on the manipulation of the GSR line and total configuration (as previously described).

However, approaches based on partial reconfiguration can greatly speed up this process. Typically, they use the FPGA Internal Configuration Access Port (ICAP), which speeds up the fault injection process up to 12 times with respect to equivalent total reconfiguration ones [24]. This idea has been used for other fault injection tools targeting the whole configuration memory of the FPGA. For instance, the authors in [25] report only 4.6 ms per fault injection and evaluation.

However, even though partial reconfiguration through the ICAP port is potentially very interesting for carrying out fault injection on FPGA FFs, only a few works have explored this possibility. Among them, one can highlight [18]. Nevertheless, it is based on manipulating the GSR line of the FPGA, which affects all the FFs of the FPGA simultaneously. For that reason, previous (and time consuming) readback and total reconfiguration operations are needed in order to control how this signal is switched to each one of the FFs so SEUs are emulated in a controlled way. For the sake of performance, the methodology presented in this paper follows a different approach: using the local SR and REV signals to the FPGA slices. This is one of the key contributions of the approach presented in this paper. The following subsection will elaborate this point in greater detail.

C. The Presented Hybrid Fault Injection Approach for FFs

The main contribution of this paper is a technique for fault emulation in FFs using FPGAs, and exploiting partial reconfiguration. It is a hybrid technique that takes the best of both instrumentation-based and reconfiguration-based SEU-emulation approaches. Hence, it offers a good trade-off between area overhead and performance, making it adequate for comprehensive fault-injection campaigns on all types of circuit (small or large). On the one hand, it features a considerably lower area overhead with respect to other instrumentation-based systems [13]–[16]. On the other hand, it speeds up the fault injection process by up to 12 times thanks to partial reconfiguration, as opposed to to other reconfiguration-based ones [17]–[24]. Basically, *our approach injects bitflips in the FPGA FFs by manipulating the local resets of the FPGA slices*. This is a key difference with respect to the approaches based on *capture&readback* and total reconfiguration [22], [23], where the global GSR line is used.

It is also extremely important to point out that the methodology presented in this paper does not target injecting bitflips on FPGA configuration bits. Instead, it targets an ASIC-based technology. Hence the methodology of this paper uses partial reconfiguration to carry out fault injections in the content of FFs, and FPGAs are used just to emulate the SEUs. Thus, even though it introduces changes in the original circuit, neither these changes alter its functionality, nor it introduces new sensitive zones in the circuit, or modifies the existing ones. In fact, typically, instrumentation-based techniques share this approach (such as [16], for instance).

This methodology has been integrated into NESSY (*Non intrusive ErrorS injection SYstem*) [27], a fault-injection tool developed by our research group that targeted the SRAM-based configuration memory of Xilinx™FPGAs. Thus, the approach presented in this paper makes NESSY capable of performing either of the two following types of SEU emulation campaigns: in the configuration memory of the FPGA (only for circuits to be implemented in this target technology), and in the circuit FFs. This means that NESSY has been extended to support two modes of operation. For the first case, NESSY will use the approach described in [27], which targets circuits implemented in FPGA technology. However, for fault injection in FFs, the methodology presented in this paper will be used instead. Both of them are complementary, and this potentially allows to quantify the tolerance against SEUs of different digital circuits, implemented either in an FPGA or as an ASIC.

The presented methodology uses the Xilinx™Virtex-5 XC5VLX110T FPGA as target device. However, it is easily portable to other Xilinx™Virtex device (such as Virtex-4 or Virtex-6 [28], [29]) since all of them feature the same kind of FFs, with identical input/output interfaces. In addition, the information of the bitstream in different devices is organized similarly, in a number of frames with different sizes. This information is easily provided to NESSY by means of a configuration file.

The remainder of the paper is organized as follows: First of all, Section II describes the presented fault injection methodology for FFs and the instrumented hardware needed. Next, Section III explains the modifications introduced in order to incorporate this methodology in a partial reconfiguration based fault injection tool, like NESSY. Section IV presents results and finally, Section V concludes this paper.

II. The Fault-Injection Methodology in FPGA Flip-Flops

For Xilinx™Virtex-5 FPGAs, the information stored in a FF is not modifiable from the configuration memory of the device unless a reset of the FPGA's sequential logic is carried out [26], [30]. Instead, for each FF, a pair of attributes named $SRHIGH$ (*Set/Reset to HIGH*) and $SRLOW$ (*Set/Reset to LOW*) in the Xilinx™documentation [31], are accessible. Both attributes are complementary. Thus, they can only take the following values: $SRHIGH = 1$, $SRLOW = 0$ and $SRHIGH = 0$, $SRLOW = 1$. They are in fact accessible through a single bit in the configuration memory, named $SRHIGH/SRLOW$ in the following.

The purpose of these attributes is to set the functionality of the input signals SR (*Set/Reset*) and REV (*REVerse*) of the

TABLE I
Xilinx™Virtex-5 FF truth table. The output Q signal is updated synchronously with the input clock signal C (not shown in the table for simplicity)

SRHIGH=0; SRLOW=1			*SRHIGH=1; SRLOW=0*		
SR	*REV*	*Q (t+1)*	*SR*	*REV*	*Q (t+1)*
0	0	*Q (t)* [a]	0	0	*Q (t)* [a]
0	1	1 [b]	0	1	0 [b]
1	0	0 [b]	1	0	1 [b]
1	1	0 [b]	1	1	0 [b]

[a] Updated only if CE = '1'
[b] Updated either if CE = '0' or '1'

FPGA FFs, as indicated in Table I. Thus, if $SRHIGH = 0$ ($SRLOW = 1$) then SR can be used to initialize the FF to 0 (it acts as a *reset* signal) and REV, to initialize it to 1 (it acts as a *set* signal); whereas if $SRHIGH = 1$ ($SRLOW = 0$) the FF behaves in the opposite way (SR would be a *set*; and REV, a *reset*). Unless stated otherwise, the Xilinx™synthesis tools use the Virtex-5 FFs under the operation: $SRHIGH = 0$, $SRLOW = 1$.

The Xilinx™synthesis tools allow to instantiate the FFs existing in the FPGA by means of *primitives* in the source code. A comprehensive list of all the available primitives available for instantiating Virtex-5 FPGA FFs can be found in [32]. They basically differ in the number of input entries that are accessible from the circuit source code (essentially, a *reset* and/or *set* input signals). Among all the available ones, $FDRSE$ (a D-FF with synchronous *reset* and *set*) is the one that most directly implements a FF in the FPGA. The reason is that, since $SRHIGH = 0$ and $SRLOW = 1$, the *reset* input is connected to SR and the *set* one, to REV. Other FF primitives are implemented in a similar way, but set the SR and/or REV signals to '0' in order to comply with the primitive specifications. In any case, all of them have in common that they instantiate the same (and unique) kind of FF available in the FPGA.

The proposed methodology uses the information in Table I and adds little hardware instrumentation around all the FFs of the circuit under test in order to emulate SEUs just modifying the $SRHIGH/SRLOW$ bit. This new hardware (Figure 1) includes a new input signal inj, $inj = 1$ indicating that a SEU must be emulated in any of the FFs of the circuit under test. This signal is part of the testbench of the circuit. Thus, it is possible to control the exact clock cycle when the SEU has to be emulated. Note that Figure 1 shows the particular case for the $FDRSE$ primitive.

In order to specify *in which FF* (or FFs) the SEU (or SEUs) must be emulated, NESSY triggers the attributes $SRHIGH = 1$, $SRLOW = 0$ of the target FF(s) by means of partial reconfiguration while keeping the remaining ones to $SRHIGH = 0$, $SRLOW = 1$. Of course, in order to do this, it is not necessary to carry out the *Capture&Readback* / Modify of *all* the FFs on the FPGA. Instead, NESSY identifies the *frame(s)* that contain(s) the bit(s) that has/have to be modified and writes the modified version of that/those frame(s) onto the configuration memory of the FPGA. A frame is the

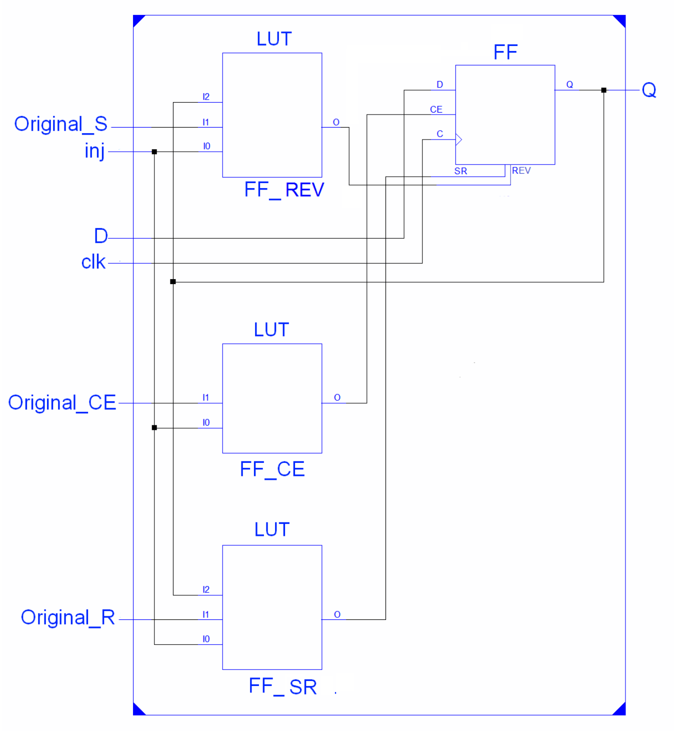


Fig. 1. RTL model of a modified $FDRSE$ FF, for SEU emulations using NESSY

smallest addressable segment of the Virtex-5 configuration memory space, and it contains 1280 configuration bits. Thus, this approach guarantees that the additional overhead introduced due to the partial reconfiguration is very low. This point will be discussed in deeper detail in Section IV.

The exact location of this configuration bit in the circuit bitstream has been obtained for all the FFs in the FPGA by reverse engineering on a simple design with one FF: The $SRHIGH$ and $SRLOW$ attributes were set both to '0'/'1' and to '1'/'0' by using the Xilinx^{TM}FPGA Editor tool, then the two resulting bitstreams were compared, and finally it was discovered that they differed in just one bit. This infomation allowed obtaining the exact position of this configuration bit, for any FF located anywhere in the FPGA. This methodology has been necessary because that information is not available in the Xilinx^{TM}documentation [26].

The extra logic added to each FF primitive is implemented in three Look-Up Tables (LUTs), depicted in Figure 1, and named FF_REV, FF_CE and FF_SR, respectively. Each one of them implements the following equations:

$$FF_{REV} = inj \cdot Q + \overline{inj} \cdot original\_S \tag{1}$$

$$FF_{CE} = \overline{inj} \cdot original\_CE \tag{2}$$

$$FF_{SR} = inj \cdot \overline{Q} + \overline{inj} \cdot original\_R \tag{3}$$

where $original\_CE$, $original\_R$ and $original\_S$ are the signals of the FF originally connected to the Clock Enable (CE), $reset$ and set inputs, respectively. This is true for the $FDRSE$ one, where these three inputs are accessible from the source code. In case a primitive without *reset* and/or *set* was instantiated in the code, it is replaced by the $FDRSE$, by setting the missing $original\_reset$ and/or $original\_set$ signals to '0' in order not to alter its original behaviour.

According to (1), (2) and (3), if no SEU is injected ($inj = 0$), the FFs work under normal operation ($FF_{CE} = original\_CE$); otherwise, all the FFs of the circuit under test are updated according to their specific values of the $SRHIGH$/ $SRLOW$ configuration bits, and the FF_{REV} and FF_{SR} inputs following the information in Table 1. Thus, since the presented methodology makes the target FF work under the configuration $SRHIGH = 1$, $SRLOW = 0$; according to equations (1) and (2), if $inj = 1$, then $FF_{SR} = \overline{Q}$ and $FF_{REV} = Q$. Hence if $Q = 0$, then $FF_{SR} = 1$ and $FF_{REV} = 0$, which triggers $Q(t+1)$ to 1 (bitflip) in the target FF, and to 0 (no change) in the remaining ones. Following the same reasoning, if $Q = 1$, then $FF_{SR} = 0$ and $FF_{REV} = 1$, which triggers $Q(t + 1)$ to 0 (bitflip) in the target FF, and to 1 (no change) in the remaining ones. Note that, if SEUs are to be injected in several FFs, then this is done in parallel with all the involved FFs.

As a consequence of this hardware instrumentation, the area required to implement a modified design is greater than its original version. The increase factor with respect to the original circuit greatly depends on the number of FFs of each circuit. A discussion about this point is included in Section IV.

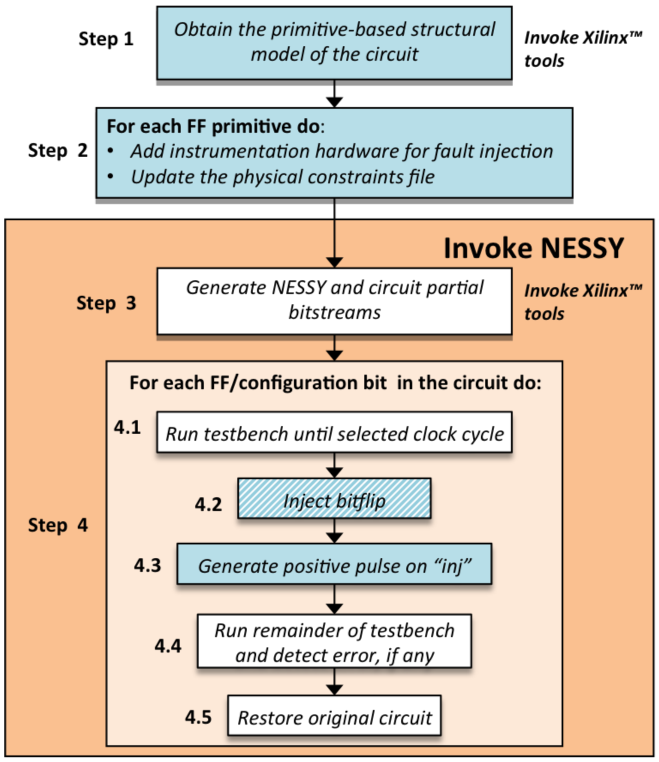
III. Fault Injection Flow

As hinted above, this methodology has been integrated into NESSY so two complementary SEU emulation modes are now possible:

- Exhaustively in the FPGA configuration memory bits that implement the circuit under test, as described in [27], in a non-intrusive way. This methodology is suitable to test the SEU tolerance of FPGA designs.
- Selectively in the FFs of the circuit under test, using the methodology presented in this paper. Even if it introduces small modifications in the original circuit, it is suitable to test the SEU tolerance of digital circuits when manufactured as ASICs, since these modifications do not alter the behaviour of the circuit at the register-transfer level.

The flowchart in Figure 2 depicts how the presented methodology has been integrated in NESSY. Steps 3, 4.1, 4.4 and 4.5 were already implemented in the previous version of NESSY, whereas Steps 1, 2 and 4.3 are new. These steps are activated/deactivated, depending on the SEU emulation mode that is used. Finally, Step 4.2 has been updated from the previous version in order to support both types of SEU emulations. This is indicated in the figure through the striped colouring. A detailed description of this flowchart is provided below.

First, the Xilinx^{TM}ISE tool is invoked (Step 1 in Figure 2) in order to generate a new *.vhd* file containing a structural



Step 1
Obtain the primitive-based structural model of the circuit
Invoke Xilinx™ tools
Step 2
For each FF primitive do:
• Add instrumentation hardware for fault injection
• Update the physical constraints file
Invoke NESSY
Step 3
Generate NESSY and circuit partial bitstreams
Invoke Xilinx™ tools
Step 4
For each FF/configuration bit in the circuit do:
4.1
Run testbench until selected clock cycle
4.2
Inject bitflip
4.3
Generate positive pulse on "inj"
4.4
Run remainder of testbench and detect error, if any
4.5
Restore original circuit

Fig. 2. The presented fault injection methodology integrated in NESSY

model of the original circuit. This model describes the implementation of the circuit in terms of Xilinx^{TM}primitives, which allows interaction with the FPGA FFs at the register level.

Next (Step 2), our approach identifies the FF primitives in this structural model and for each one of them, it adds the extra logic described in Figure 1, taking into account that all of them have a by-default $SRHIGH = 0$, $SRLOW = 1$ initialization. This step also updates the physical constraints file (.ucf) in order to force the physical placement of the FFs. This is needed because, in Xilinx^{TM}FPGAs, the local resets of the FFs are shared between the four FFs located in the same FPGA slice. This fact is taken into account in order to place the FFs in different FPGA slices from each other.

In Step 3, NESSY is invoked to generate two bitstreams: one with the circuit under test and the other one with the static part of NESSY. This is explained in greater details in [27].

Finally, Step 4 describes the fault injection methodology itself. The testbench is firstly executed (Step 4.1) until the clock cycle where the fault is injected (remember that this is indicated in the testbench by means of the activation of the inj input signal to '1'). Next, the bitflip (Step 4.2) is injected. In the previous version of NESSY, this was achieved by just modifying the desired bit in the configuration memory and by writing the corresponding frame onto the FPGA by using the ICAP port [27]. In the updated version, it modifies the $SRHIGH/SRLOW$ bit corresponding to the FF (or FFs) that is/are to be modified. The positive pulse inj signal is then generated (Step 4.3) to flip the content of the FF (or FFs) selected in Step 4.2. Once this has been made, the remainder of the testbench is executed and, if a fault has been detected, it is reported (Step 4.4). Finally, the original circuit is restored (Step 4.5). Note that the $SRHIGH/SRLOW$ bit is not restored until this step. The reason is that it does not affect the remainder of the testbench execution while the inj signal is '0'. In addition, this does not generate unnecessary delays that an additional frame restoration would involve.

TABLE II
FEATURES OF THE BENCHMARKS THAT WERE EVALUATED

Circuit	*Circuit functionality*	*[#] of conf. bits*	*FF count*
b01	FSM that compares serial flows	46,080	5
b02	FSM that recognizes BCD numbers	46,080	4
b03	Resource arbiter	46,080	30
b04	Compute min and max	92,160	66
b05	Elaborate the contents of a memory	138,240	34
b06	Interrupt handler	46,080	9
b07	Count points on a straight line	46,080	49
b08	Find inclusions in number sequences	46,080	21
b09	Serial to serial converter	46,080	28
b10	Voting system	46,080	17
b11	Scramble string with variable cipher	92,160	31
b12	1-player game (guess a sequence)	138,240	121
FFE	Finite impulse response filter	1,797,120	622

IV. RESULTS AND DISCUSSION

This section presents SEU emulation results on the FFs of circuits taken from the ITC'99 benchmark set [33] and on a Feed-Forward Equalization (FFE) filter featuring 16 filtering steps. For ITC'99 designs, only those from *b01* to *b12* have been selected for fault injection. The remaining ones were not tested as, in its current version, NESSY cannot run fault injection on microprocessors. On the other hand, the FFE filter has been selected because it is a realistic application that is widely used in aerospace contexts and because it is considerably larger than the ITC'99 circuits. Table II summarizes the main characteristics of the benchmarks, as well as their number of configuration bits and FF count.

A. Statistical Validation

A statistical methodology was carried out in order to validate the presented methodology. This decision was made since a validation against a radiation beam was not possible in this case. The reason is that data issued from radiation ground tests cannot mimic the fault injection results obtained by the presented methodology at all, hence they cannot be directly cross-checked. Indeed, the presented methodology only injects SEUs in the content of FFs, but experimental data regarding sensitivity of FPGAs would show the sensitivity of the entire configuration memory.

In fact, the configuration memory and the programmable logic of modern FPGAs constitute the same silicon chip. Hence, they cannot be physically separated in order to only radiate the programmable logic (which contains the FFs and the user memory) thereby excluding the configuration memory from the radiation beam. Hence, in order to make such experimental validation, it would be necessary either:

- To have an implementation of the validated circuit in silicon (i.e., an ASIC), whose only elements sensitive to SEUs are its FFs and memory cells. Such a circuit could be introduced in a radiation chamber. However, for the case of the circuits tested in this paper, the only option would be to make ad-hoc implementations for all of them, which has an unaffordable financial cost.
- A 100% radhard implementation of an FPGA whose FFs have not been hardened. However, this is also unfeasible because commercial radhard FPGAs are hardened at all levels: programmable logic and routing, user memory (BlockRAMs and FFs) and embedded logic (multipliers, processors...).

This motivates that purely fault-injection experiments have to be carried out in this case. As indicated in [34], the test space for any fault-injection experiment comprises three axes: a) the number of fault locations, b) the number of test vectors and c) the number of program cycles of operation. However, even for small circuits, exhaustive fault-injection experiments may involve a combinatorial explosion of test cases that cannot be managed with an average computer. For this reason, the methodology presented in this paper does not carry out exhaustive fault injections either.

If a fault emulation approach is not exhaustive in any of said axes, the obtained results need to be statistically validated. Thus, for this experiment: a) A number of SEUs have been emulated exhaustively in all the FFs of each circuit; b) For the ITC'99 *b01* - *b12* circuits, gold vectors provided by the designers [35] have been used as testbenches. For the FFE, it was elaborated an ad-hoc testbench that is representative of all its functionality. And c) NESSY allows emulating SEUs in any clock cycle of the testbench, as previously described in Figure 2.

The concepts presented in [36] were used to validate the presented methodology. The sampling error equation given in this reference returns the number of SEUs (n) that need to be injected in a system with N test cases, a given confidence level and a given margin of error. For these experiments, $N = NUM\_FFs * NUM\_cycles\_testbench$.

As a first experiment, a confidence level of 90% and a margin error of 5% have been set. For these parameter values, the equation in [36] returns 73.21 SEUs per FF for *b01* - *b12*. Hence, 74 SEUs per FF were injected in these circuits. Note that we speak in terms of *number of SEUs per FF*. The *total* number of SEUs (n) to be injected in a given circuit is directly proportional to its number of FFs and the number of testbench clock cycles. However, since all the testbenches of *b01* - *b12* have 100 cycles, normalizing n by the number of FFs of each circuit always returns the same value (73.21). Repeating this experiment for the FFE, this value was 228 SEUs per FF.

These SEUs were emulated and it was obtained the percentage among them that resulted into an error at the output of the circuit. The first bar in Figure 3 shows these results.

Experiments in bars 2 and 3 in Figure 3 aim at gradually attain an error margin of 0% (regardless of the confidence level). For this purpose, 100 SEUs for *b01* - *b12* and 1000 SEUs for the FFE need to be emulated according to Equations (1) and (2) in [36]. *This means that, from the statistical point of view, these results have the same validity as an exhaustive fault-injection campaign*. Finally, the experiment with 1000 SEUs per FF was also repeated for *b01* - *b12* (bar 3 in Figure 3).

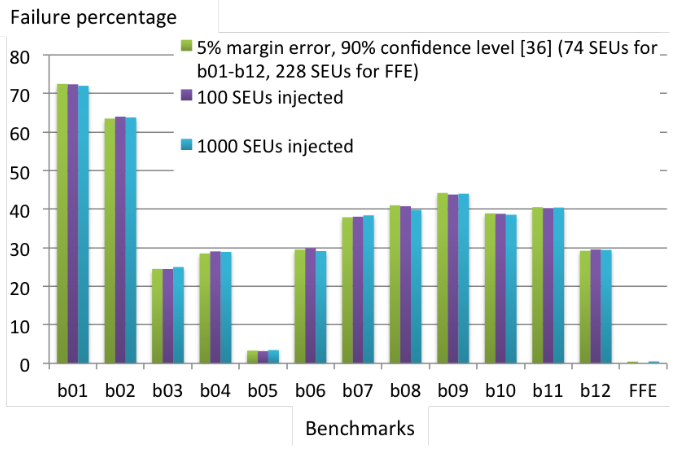


Fig. 3. Failure percentage for the ITC'99 *b01-b12* benchmarks and a FFE filter, for different number of fault injections per FF

As the figure shows, for a given circuit, the obtained SEU sensitivities are very similar. In fact, the gradient of the regression line that best fits to the three bars is, on average, 0.00027, which is negligible. This demonstrates that the results issued from bars 1 and 2 were already very accurate. Thus, the presented methodology can be safely extrapolated to much larger circuits, where an exhaustive fault-injection experiment is unfeasible.

Finally, in comparison with [16] one can verify that the obtained percentage of detected errors for *b12* (29.41%) is very similar to that of [16] (29.8%), even though we both use completely different testbenches to obtain the results. This finding further supports the correctness of the presented approach and the selected experimental setup.

B. Design Overhead

This subsection presents a comparison between the presented methodology and the ones presented in [16], in terms of design overhead. The same circuits as in the previous subsection were used.

Figures 4 and 5 depict the resource consumption of the modified circuits when using the presented methodology, in comparison with the so-called *Mask-scan*, *State-scan* and *Time-mux* ones presented in [16]. The proposed methodology does not increase nor decrease the [#] FFs with respect to the original circuits (Figure 4, the *+% FFs overhead* of all the circuits is always 0%). However, the *Mask-scan* and *State-scan* approaches double the [#] FFs needed, whereas the *Time-mux* one multiplies this number by four. In addition, note that the proposed approach *does not decrease* the number of used FFs for the original circuits either. This proves that the synthesis tool does not replicate nor remove FFs during the modification of the circuits.

The design overhead introduced in terms of [#] LUTs (Figure 5) depends on the methodology used. Thus, *Time-mux* in-

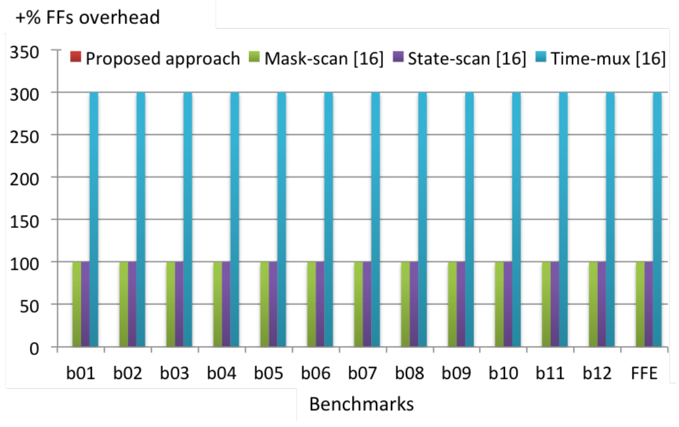


Fig. 4. FFs overhead of the modified circuits, when using the presented method, in comparison with the methodologies in [16]

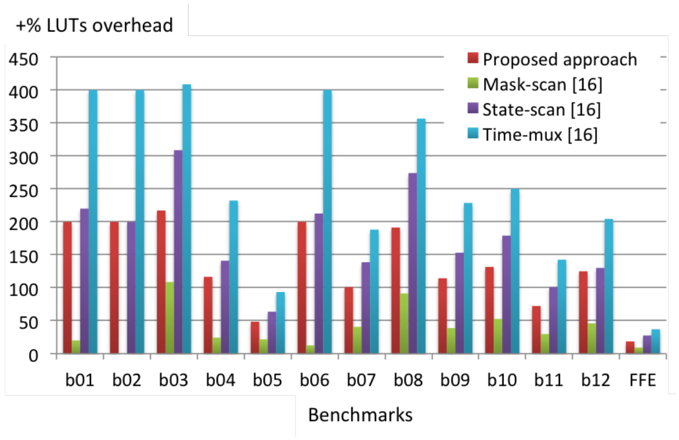


Fig. 5. LUTs overhead of the modified circuits, when using the presented method, in comparison with the methodologies in [16]

troduces an average overhead of +275.38% with respect to the original version of the circuits. This overhead is +176.75% and +40.51% for *State-scan* and *Mask-scan*, respectively, whereas for the presented approach, it is +143.11%. Thus, the presented methodology clearly reduces the resource consumption with respect to *Time-mux* and *State-scan*, whereas for *Mask-scan*, it reduces the [#] FFs needed by half, while increasing by 49.18% the [#] LUTs needed by this technique. Nevertheless, it is important to remember that the presented approach has an additional advantage: it has been integrated in the NESSY tool [27], which was originally designed to inject faults in the configuration memory of SRAM-based FPGAs. Hence, with this extension, NESSY is now able to quantify the tolerance of digital circuits against SEUs, both when they are implemented as an ASIC or configured in an FPGA.

Finally, it is also interesting to observe the existing strong correlation between the percentages displayed in Figure 5 and the resource consumption of the original circuits. In general terms, the larger the original circuit is, the less significant is the overhead, hence the small FFE bars.

C. Time Overhead

The time overhead introduced by the proposed approach has also been evaluated.

On the one hand, the execution time of the circuit under test with NESSY (Step 4 in Figure 2) depends on the size of the testbench (set to 100 or 1000 input values, depending on the benchmark), the speed of the platform (constant at 100 MHz) and the time overhead introduced in order to reconfigure a Virtex-5 frame using the ICAP port (1.6 μs, see [24]). In our experiments, a circuit running a testbench of 1000 entries takes 10 μs to be completed in NESSY [27]. Thus, in this case the reconfiguration time penalty is 16% the initial execution time of the circuit on the platform. This may seem an important limitation. However, as the size of the testbench increases, this overhead becomes decreasingly less significant with respect to the execution time of the circuit under test. Thus, for instance, let us imagine a fault injection campaign on a processor that runs at 400 MHz a testbench that takes 1 second to be completed. In that case, the fault injection time of 1.6 μs is negligible with respect to the total tesbench execution time.

On the other hand, the time needed to carry out Steps 1 and 2 of Figure 2 (which are carried out before the fault injection campaign) depends on the number of FFs of the circuit. For the experiments in this section, it was always less than 1 second. Thus, in comparison with the time needed by the Xilinx^{TM}placement and routing tools (on the order of minutes, or even hours), our approach introduces a negligible time overhead.

Hence, the methodology proposed in this paper is suitable for fault injection campaigns in large circuits, not only in terms of execution time, but also because of the little area overhead, as shown in Figures 4 and 5.

V. Concluding Remarks

This paper has presented a new methodology for SEU emulation in the embedded FFs of Xilinx^{TM}Virtex-5 FPGAs, although it is valid for any other Xilinx^{TM}Virtex device. It has been integrated into a fault-injection tool named NESSY, whose previous version [27] could only emulate SEUs into the FPGA configuration memory. The presented methodology is a hybrid technique that combines the features of both instrumentation-based and reconfiguration-based fault injection approaches. Hence, it achieves a good trade-off between resource consumption and time overhead with respect to other approaches existing in the literature. In addition, it extends the previous functionality of NESSY, and thus it potentially allows to carry out complementary experiments depending on the target technology: ASIC or FPGA.

References

[1] A. Dixit and A. Wood, “The Impact of New Technology on Soft Error Rates,” in *Reliability Physics Symposium (IRPS), 2011 IEEE International*, April 2011, pp. 5B.4.1–5B.4.7.

[2] E. Ibe, H. Taniguchi, Y. Yahagi, K. Shimbo, and T. Toba, “Impact of Scaling on Neutron-Induced Soft Error in SRAMs From a 250 nm to a 22 nm Design Rule,” *Electron Devices, IEEE Transactions on*, vol. 57, no. 7, pp. 1527–1538, July 2010.

[3] N. Mahatme, S. Jagannathan, T. Loveless, L. Massengill, B. Bhuva, S.-J. Wen, and R. Wong, “Comparison of Combinational and Sequential Error Rates for a Deep Submicron Process,” *Nuclear Science, IEEE Transactions on*, vol. 58, no. 6, pp. 2719–2725, Dec 2011.

[4] M. Ebrahimi, A. Evans, M. Tahoori, R. Seyyedi, E. Costenaro, and D. Alexandrescu, "Comprehensive Analysis of Alpha and Neutron Particle-Induced Soft Errors in an Embedded Processor at Nanoscales," in *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*, March 2014, pp. 1–6.

[5] B. Gill, N. Seifert, and V. Zia, "Comparison of Alpha-Particle and Neutron-Induced Combinational and Sequential Logic Error Rates at the 32nm Technology Node," in *Reliability Physics Symposium, 2009 IEEE International*, April 2009, pp. 199–205.

[6] Xilinx Corporation, "Virtex-5 Family Overview," online at: http://www.xilinx.com/support/documentation/data_sheets/ds100.pdf, 2009.

[7] C. Bender, P. Sanda, P. Kudva, R. Mata, V. Pokala, R. Haraden, and M. Schallhorn, "Soft-Error Resilience of the IBM POWER6 Processor Input/Output Subsystem," *IBM Journal of Research and Development*, vol. 52, no. 3, pp. 285–292, May 2008.

[8] K. Morgan, M. Caffrey, P. Graham, E. Johnson, B. Pratt, and M. Wirthlin, "SEU-Induced Persistent Error Propagation in FPGAs," *Nuclear Science, IEEE Transactions on*, vol. 52, no. 6, pp. 2438–2445, Dec 2005.

[9] L. Wissel, E. Cannon, D. Heidel, M. Gordon, and K. Rodbell, "Alpha-Particle, Carbon-Ion and Proton-Induced Flip-Flop Single-Event-Upsets in 65 nm Bulk Technology," *Nuclear Science, IEEE Transactions on*, vol. 55, no. 6, pp. 3375–3380, Dec 2008.

[10] T. Criswell, P. Measel, and K. L. Wahlin, "Single Event Upset Testing with Relativistic Heavy Ions," *Nuclear Science, IEEE Transactions on*, vol. 31, no. 6, pp. 1559–1561, Dec 1984.

[11] R. Velazco, G. Foucard, and P. Peronnard, "Combining Results of Accelerated Radiation Tests and Fault Injections to Predict the Error Rate of an Application Implemented in SRAM-Based FPGAs," *Nuclear Science, IEEE Transactions on*, vol. 57, no. 6, pp. 3500–3505, Dec 2010.

[12] L. Berrojo, F. Corno, L. Entrena, I. Gonzalez, C. Lopez, M. Sonza Reorda, and G. Squillero, "An Industrial Environment for High-Level Fault-Tolerant Structures Insertion and Validation," in *VLSI Test Symposium, 2002. (VTS 2002). Proceedings 20th IEEE*, 2002, pp. 229–236.

[13] M. Ebrahimi, A. Mohammadi, A. Ejlali, and S. G. Miremadi, "A Fast, Flexible, and Easy-to-Develop FPGA-Based Fault Injection Technique," *Microelectronics Reliability*, vol. 54, no. 5, pp. 1000–1008, 2014.

[14] W. Mansour and R. Velazco, "An Automated SEU Fault-Injection Method and Tool for HDL-Based Designs," *Nuclear Science, IEEE Transactions on*, vol. 60, no. 4, pp. 2728–2733, Aug 2013.

[15] F. Faure, P. Peronnard, and R. Velazco, "Thesic+: A Flexible System for SEE Testing," in *The Conference on Radiation and its Effects on Components and Systems (RADECS)*, Sept 2012.

[16] C. Lopez-Ongil, M. Garcia-Valderas, M. Portela-Garcia, and L. Entrena, "Autonomous Fault Emulation: A New FPGA-Based Acceleration System for Hardness Evaluation," *Nuclear Science, IEEE Transactions on*, vol. 54, no. 1, pp. 252–261, Feb 2007.

[17] R. Leveugle, L. Antoni, and B. Fehér, "Dependability Analysis: a New Application for Run-Time Reconfiguration," in *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, April 2003, pp. 7 pp.–.

[18] L. Antoni, R. Leveugle, and B. Feher, "Using Run-Time Reconfiguration for Fault Injection Applications," *Instrumentation and Measurement, IEEE Transactions on*, vol. 52, no. 5, pp. 1468–1473, Oct 2003.

[19] D. de Andres, J.-C. Ruiz, D. Gil, and P. Gil, "Fault Emulation for Dependability Evaluation of VLSI Systems," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 16, no. 4, pp. 422–431, April 2008.

[20] M. A. Aguirre, J. N. Tombs, V. Baena-Lecuyer, J. L. Mora, J. M. Carrasco, A. Torralba, and L. G. Franquelo, "Microprocessor and FPGA Interfaces for in-System co-Debugging in Field Programmable Hybrid Systems," *Microprocessors and Microsystems*, vol. 29, no. 23, pp. 75 – 85, 2005, special Issue on FPGA Tools and Techniques.

[21] M. Aguirre, J. N. Tombs, F. Muñoz, V. Baena, H. Guzman, J. Napoles, A. Torralba, A. Fernandez-Leon, F. Tortosa-Lopez, and D. Merodio, "Selective Protection Analysis Using a SEU Emulator: Testing Protocol and Case Study Over the Leon2 Processor," *Nuclear Science, IEEE Transactions on*, vol. 54, no. 4, pp. 951–956, Aug 2007.

[22] J. Mogollon, H. Guzman-Miranda, J. Napoles, J. Barrientos, and M. Aguirre, "FTUNSHADES2: A Novel Platform for Early Evaluation of Robustness Against SEE," in *Radiation and Its Effects on Components and Systems (RADECS), 2011 12th European Conference on*, Sept 2011, pp. 169–174.

[23] H. Guzman-Miranda, J. Napoles, J. Mogollón, J. Barrientos, L. Sanz, and M. Aguirre, "FT-UNSHADES2: A platform for early evaluation of ASIC and FPGA dependability using partial reconfiguration," in *Jornadas SARTECO (La Sociedad de Arquitectura y Tecnologa de Computadores)*, Sept 2012.

[24] E. McDonald, "Runtime FPGA Partial Reconfiguration," in *the IEEE Aerospace Conference*, 2008, pp. 1–7.

[25] L. Sterpone and M. Violante, "A New Partial Reconfiguration-Based Fault-Injection System to Evaluate SEU Effects in SRAM-Based FPGAs," *Nuclear Science, IEEE Transactions on*, vol. 54, no. 4, pp. 965–970, Aug 2007.

[26] Xilinx Corporation, "Virtex-5 FPGA Configuration User Guide, UG191 (v 3.11)," online at: http://www.xilinx.com/support/documentation/user_guides/ug191.pdf, 2012.

[27] V. Alaminos, F. Serrano, J. A. Clemente, and H. Mecha, "NESSY: An Implementation of a Low-cost Fault-injection Platform on a Virtex-5 FPGA," in *The Conference on Radiation and its Effects on Components and Systems (RADECS)*, Sept 2012.

[28] Xilinx Corporation, "Virtex-4 FPGA Configuration User Guide," online at: http://www.xilinx.com/support/documentation/user_guides/ug071.pdf, 2009.

[29] ——, "Virtex-6 FPGA Configuration User Guide," online at: http://www.xilinx.com/support/documentation/user_guides/ug360.pdf, 2014.

[30] M. C. Carl Carmichael and A. Salaza, "Correcting Single-Event Upsets Through Virtex Partial Configuration (XAPP216 (v1.0))," 2000.

[31] Xilinx Corporation, "Virtex-5 FPGA User Guide, UG190 (v5.4)," online at: http://www.xilinx.com/support/documentation/user_guides/ug190.pdf, 2012.

[32] ——. (2011) Virtex-5 libraries guide for schematic designs, ug622 (v 13.1). online at: http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_1/virtex5_scm.pdf.

[33] F. Corno, M. Reorda, and G. Squillero, "RT-Level ITC'99 Benchmarks and First ATPG Results," *Design Test of Computers, IEEE*, vol. 17, no. 3, pp. 44–53, Jul 2000.

[34] H. Quinn, D. Black, W. Robinson, and S. Buchner, "Fault simulation and emulation tools to augment radiation-hardness assurance testing," *Nuclear Science, IEEE Transactions on*, vol. 60, no. 3, pp. 2119–2142, June 2013.

[35] CAD Group, Politecnico di Torino, "http://www.cad.polito.it/downloads/tools/itc99.html," 2014.

[36] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert, "Statistical Fault Injection: Quantified Error and Confidence,," in *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*