IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 31, 2019, accepted August 25, 2019, date of publication September 6, 2019, date of current version September 20, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2939858

ACME: A Tool to Improve Configuration Memory Fault Injection in SRAM-Based FPGAs

LUIS ALBERTO ARANDA, ALFONSO SÁNCHEZ-MACIÁN, AND JUAN ANTONIO MAESTRO, (Senior Member, IEEE)
ARIES Research Center, Universidad Antonio de Nebrija, 28040 Madrid, Spain
Corresponding author: Luis Alberto Aranda (laranda@nebrija.es)

ABSTRACT Circuits in harsh environments, as space, tend to suffer severe problems caused by radiation. In this scenario, where the behavior of the system can be jeopardized, it is critical to produce fault tolerant circuits that can operate correctly. An important task in this scenario is to effectively test the new fault tolerant designs to guarantee their correct operation. There exist several and diverse methods to achieve this task, from actual test flights to the use of particle accelerators. Fault injection emulation is one of the most popular methods, due to its low cost, availability and convenience. There are a number of tools to perform fault injection using a field-programmable gate array (FPGA) as a supporting platform for this task. However, most of these tools are very dependent on the FPGA version and technology, with limited capability to control the injection process in a precise way. In this paper we present ACME (Automatic Configuration Memory Error-injection), a new tool able to pinpoint fault injections in specific areas of the design under test, with great control and precision of the process. In addition, the methodology to configure the tool and make it work with new FPGA families is also provided.

INDEX TERMS Configuration memory, emulation, fault injection, fault tolerance, FPGA, reliability.

I. INTRODUCTION

While the performance vs. area / power consumption ratio has been the main trade-off to explore in consumer electronics, reliability has recently become a major concern. The miniaturization of devices, which implies the use of smaller geometries, makes systems more prone to suffer errors, both permanent and temporary [1]. While at ground level this is becoming an issue [2], the problem is really prominent at higher altitudes, especially in space applications. Space is a harsh environment, where there are multiple sources of radiation [3]. In this way, cosmic rays, solar flares and the radiation trapped in the Earth's magnetosphere present numerous problems to the electronics on board satellites and other spacecraft. These problems may induce permanent failures in the devices, which make them inoperative for the rest of the mission [4]. One example of this is the total dose, which happens when transistors have received too much radiation over time, inducing charge in their channels that permanently disables the device [5]. Another example is the displacement damage, which happens when an energetic particle hits the silicon substrate and removes one of the atoms forming the structure [6]. This produces breakage points that may end up making the devise unusable. Another kind of errors are those that are temporary, also called soft errors [7]. This happens when energetic particles traverse the devices, producing direct or indirect ionization problems. In this way, free charge is generated along the ion track, which due to diffusion and other phenomena induce a logic bit flip in the device. In other words, a logic '0' may be flipped into a logic '1' and vice versa, thus corrupting data and altering the behavior of the circuit. Soft errors may present different behaviors. Single Event Upsets (SEU) are bit flips induced in storage elements, as memory cells and registers. If an individual particle produces several simultaneous bit flips in adjacent positons, then that would be a Multiple Bit Upset (MBU). Single Event Transients (SET) are bit flips produced in the combinational logic, which induce a transient pulse that propagates to the rest of the circuit, possibly being registered at some stage. Single Event Functional Interruptions (SEFI) are bit flips that

The associate editor coordinating the review of this manuscript and approving it for publication was Yongquan Sun.

affect the control structure of the components, e.g. the state machine of memories, thus producing unexpected behaviors.

In this situation, it is necessary to utilize circuits that can effectively operate in these adverse circumstances. There are two alternatives to achieve this. The first one is to produce circuits that are radiation-hardened by process. This means that the circuits are physically hardened, in order to support higher doses of radiation. This solution comes with some problems [8], specifically the higher manufacturing costs (over one order of magnitude respect to the equivalent commercial components) and the reduction of functionality. The other alternative is to protect circuits by design [8]. This means to use commercial components that are modified adding redundancy so that they can detect and sometimes correct errors. This can be achieved using standard techniques, usually based on massive redundancy, like Dual Modular Redundancy (DMR) and Triple Modular Redundancy (TMR), in order to detect and correct errors, respectively. This usually comes at large area and power consumption overheads, which many times are unfeasible due to the strict constraints typical of space applications. Another option is to use ad-hoc protection techniques, which add redundancy in a selective way taking advantage of algorithmic properties of the circuit. An example of this is the algorithm-based fault tolerance (ABFT) approach [9]. In any case, both standard and ad-hoc protection techniques imply the modification of the design, which forces the need to intensively test the new design, to guarantee its functional correctness and the expected fault tolerance levels. Therefore, the need to count with efficient test methods is a must in any situation in which reliability is an issue.

In order to effectively test circuits and verify their correct behavior, several alternatives are available. One possibility is to perform in-flight characterization, which implies installing the circuit in an actual test mission and measure its behavior. This, although very realistic, is unfeasible most of the times due to the low availability of flight opportunities and the large associated cost. Another possibility is to perform physical experiments at ground facilities, trying to replicate the conditions of the space environment. Usually, this is conducted at particle accelerators that can perform the test in an accelerated way using beams with different kinds of ions and high energetic neutrons [10]. Another option is to use alternative test methods, as the use of laser beams to replicate the impact of particles in the device [11]. In all cases, these facilities are usually expensive and many times present availability issues [12]. Therefore, although they are more convenient and accessible than actual test flights, they also present problems. Due to this, tests based on fault emulation are quite popular. Fault emulation is not based on physical interaction with the device, but on inducing artificial bit flips using instrumentation methods. A common approach is to use an FPGA to hold the design under test, and then introduce faults by flipping bits in the configuration memory [13]. In this way, the design is altered and its behavior can be assessed using different benchmarks and test cases. This solution, although less realistic than the previous ones, presents low cost and high availability. Hence, it is a popular approach that can also be complemented with some kind of physical tests. One of the main problems of the emulation solutions is that they are very dependent on the FPGA architecture. In this way, controlling the experiments, by being aware of which bits need to be modified in order to affect a specific part of the design under test, implies a deep knowledge of the FPGA resources and technology. This is usually achieved by costly reverse engineering processes on the board [14], which allow to understand and control the aforementioned emulation experiments. But this knowledge is difficult to extrapolate to other FPGAs since each internal architecture differs from one device to another. In this paper we present a tool that allows to optimize the fault injection process in an FPGA, being able to pinpoint specific areas of the FPGA to inject errors, thus saving time in the process and gaining more control of the experiments.

The rest of this paper is structured as follows. Section II introduces the main tools for error injection in electronic circuits. In Section III, the motivation behind the creation of the tool is explained. The proposed tool is described in detail in Section IV. In Section V, the tool is tested with several design examples, presenting the results of these tests at the end of that section. Finally, Section VI concludes the paper.

II. RELATED WORK

In this section, the main tools for error injection in electronic circuits are described. Two main scenarios exist to validate a design against errors. First, when working with designs that are going to be implemented in Application Specific Integrated Circuits (ASICs), the errors to be injected are those affecting the user logic. They produce effects such as Single Event Upsets (SEUs) in the user memories (e.g. a microprocessor cache structure) or Single Event Transients (SETs) in combinational logic. However, there is a second scenario based on the use of reconfigurable FPGAs. In this case, both configuration memory and user memory errors can be injected.

To inject errors in the user memory, it is possible to use simulation-based and instrumentation-based methods. These methods may be supported by the execution of a testbed with the errors already injected in the design, the use of a saboteur [15], scan-chain-based approaches [16], or specific tools such as the SEUs Simulation Tool (SST) to produce random bit flips during simulation time [17]. In order to emulate SEUs in the registers of an FPGA, the FT-UNSHADES [18] can be used. This tool is oriented to inject errors in the flip-flops of Xilinx SRAM-based FPGAs by using reconfiguration. User-memory injection in FPGAs is an interesting approach to prototype a solution that will be later implemented in an ASIC. If, on the other hand, the final design will be implemented in an FPGA, the Xilinx Soft Error Mitigation (SEM) IP Controller [19], FT-UNSHADES2 [20], FLIPPER [21], or other works [22]–[24] can be explored to extend the injection tests to the configuration memory bits.

In an FPGA, errors in the user logic occur with less probability than configuration memory errors. This is because the configuration memory of an FPGA has a higher cross-section compared to the user memory, which makes it more prone to errors. Besides, an error in the configuration memory has a higher impact as the effect can be a modification of the design such as a change in the routing or in the Boolean expression of a Look-up Table (LUT). Hence, the study of the behavior of a design that will be deployed in an SRAM-based FPGA against configuration memory errors is highly advisable for space applications.

Regarding the previously mentioned platforms to inject errors in the configuration memory of an FPGAs, it should be mentioned that these works normally use fault injection over the configuration bits of the board. Unfortunately, as the number of configuration bits is usually too big and, consequently, it may take too long to test every bit, statistical fault injection is usually performed. This approach involves some drawbacks that are explained in detail in the following section.

III. MOTIVATION

Given the configuration memory failure model of an FPGA explained before, this section will describe the different approaches to perform an injection campaign in an efficient way and justify the need for a tool that can control the process.

In order to inject in the configuration memory, a first option is to follow the procedure explained in [13], which basically consists in performing an exhaustive campaign in all the configuration bits of the FPGA. After each injection, the behavior of the design is compared to the golden, counting how many times a wrong output has been produced. The number of injections that did actually produce an error divided by the total number of configurations bits in the board (that is equal to the total number of performed injections) would represent the percentage of errors induced in the design. This can be used as a figure of merit of the sensitivity of the FPGA with the tested design implemented in it. However, this is not exactly true, because since the injections are not targeted, many of them would have affected configuration bits on the board that are not part of the design. In this case, an absence of error at the output does not have to mean that the injected bit is not sensitive, just that it is not part of the design. Therefore, a scale factor should be applied to account for this fact and obtain the normalized sensitivity. This scale factor is simple, just the quotient of the total number of slices in the board divided by the number of slices in the design. This represent how large the design is respect to the total capacity of the board (all details can be found in [13]). As an example, let us assume that the obtained sensitivity (injections that have produced an error divided by the total number of injections) is 8%. Let us also assume that the design under test occupies just 10% of the board capacity. Then the scale factor would be 10, and therefore the normalized sensitivity would be ten times 8%, which is 80%. This method is quite simple, but it has a huge drawback: it is based on exhaustive injections, so all the bits in the configuration memory have to be tested. This is not always possible due to the high number of configuration bits in most FPGAs. For example, a Nexys 4 DDR Artix-7 FPGA has 30,607,152 configuration memory bits and a larger FPGA such as the Kintex UltraScale KCU105 has 69,338,912 bits. This large number of injections is likely unfeasible in most of the cases.

As mentioned in Section II, another option, especially for complex designs, is to perform statistical injections. This means to inject randomly in a limited number of the configuration bits, enough to obtain a representative behavior of the injections. In this case, when statistical injections are performed, the concept of confidence level and error margin have to be taken into account. This comes defined in equation (1), proposed in [25].

$$n = \frac{N}{1 + e^2 \cdot \frac{N-1}{t^2 \cdot p \cdot (1-p)}} \tag{1}$$

where N is the total number of configuration bits in the design, e is the error margin of the results, and t is a factor associated to the confidence level, in other words, the probability that the obtained result is within the specified error margin. For example an error margin of 1% ($e = 0.01$) and a confidence level of 95% ($t = 1.96$) means that the obtained result is 95% likely to have an error of 1% or less (the relation of a given confidence level and the related t can be found in [25]). p is a statistical parameter in the [0, 1] range, with a worst-case value of 0.5. Finally, n is the number of injections that are required on N in order to achieve the specified error margin and confidence level.

As an example, let us consider a soft processor implemented in an FPGA. A typical value of N would be around $6 \cdot 10^6$ essential bits. In order to achieve an error margin of 1% and a confidence level of 95%, we would need a number of injections of $n = 9{,}604$. Therefore, just with that number of injections, we would get the results in the expected error margin. What is the problem with this approach? That we need to have the list of the configuration bits essential to the design (N), in order to select the random subset in which to perform the injections (n). If that is the case, then the method can be directly used. But if the target FPGA and the associated design framework does not provide this possibility, then the process becomes quite inconvenient. Let us assume a board, with for example $12 \cdot 10^6$ configurations bits, in which a microprocessor with $6 \cdot 10^6$ essential bits (N) is implemented. If the latter list of bits is not known, then a random but blind injection campaign has to be performed in the board. Since in this case N is half of the total configuration bits, then around half of the injections would actually affect bits in N. In other words, if 9,604 injections are performed, statistically only 4,802 of them would actually be effective, what would lead to a worsening of the error margin. If we want to keep it at the expected level, then we would have to perform double of the number of injections, i.e. 19,208. This means that not having a clear idea of which of the bits in the configuration memory are essential to the design, implies a large time overhead in

the injection process, having to perform more injections that the minimum required.

There are some ways to obtain the list of configuration bits associated to a design. For example, Xilinx provides some methods to generate this list [26]. But even if we are in the situation in which we actually know the list of essential bits in a design, we still may have problems in some situations. Let us assume that we do have the list of $6 \cdot 10^6$ essential bits of the previous design. That would allow an optimal injection campaign based on the previously described statistical method. But let us imagine that we want to test the behavior of a new protection technique devised for a specific component of the processor, i.e. the translation lookaside buffer (TLB). Of course we want to exercise the TLB integrated with the whole processor, in order to have a functional design and be able to run a benchmark. In this situation, even if we have the list of essential bits of the processor, what we also need is the subset of the bits that are essential to the TLB. Let us assume that these are just 300,000 of the total $6 \cdot 10^6$. Now, our N would be these 300,000, and to keep the error margin and confidence level selected above, we would need a number of injections of $n = 9{,}307$, according to (1). But if we do not have the list of these 300,000 bits, we would need to perform the injection campaign in the overall $6 \cdot 10^6$ essential bits. And to guarantee that statistically 9,307 of them affect the TLB, we would need to perform 20 times this number of injections, i.e. 186,140 injections (since the whole design is 20 times larger than the TLB). This number of injections would be unfeasible in most of the cases. And even if it is feasible, it would take a huge time to complete the injection campaign, leading to an inefficiency of the method.

Therefore, it is fundamental to have a tool that can provide the list of essential bits associated to a design. And if this list is already provided by the design framework, then we would need the mentioned tool to provide the subset of the essential bits associated to a submodule of the design. With this, we can perform injection campaigns targeted to the specific parts that we want to test, and optimize the run time with the minimal number of injections required by the statistical procedure. In the following, we will present ACME, a tool able to provide this functionality.

IV. DESCRIPTION OF THE TOOL

ACME (Automatic Configuration Memory Error-injection) is an open-source tool created by the ARIES Research Center [27] designed to translate the configuration memory essential bits of an SRAM-based FPGA region into injection addresses for the Xilinx SEM IP Controller. In this way, when the design under test (DUT) is part of a bigger system (e.g. the TLB of a microprocessor), ACME enables the possibility of injecting errors in the essential bits of that subsystem with the SEM IP.

The SEM IP is a module designed by Xilinx to protect the configuration memory of their 7-series or UltraScale FPGAs against soft errors by using Error-Correcting Codes (ECCs). It has the additional capability of injecting errors

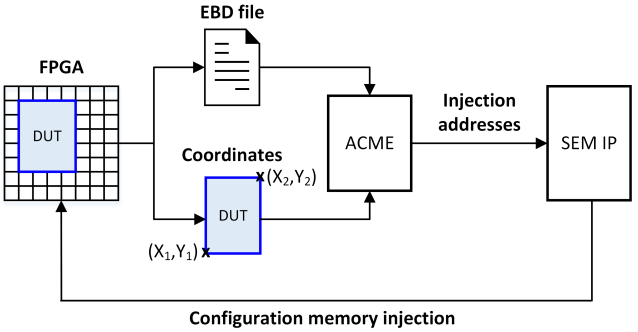


FIGURE 1. Integration of the ACME tool into the reliability analysis of an FPGA design.

into a bit of a particular frame of the configuration memory. However, given a design, Xilinx only provides a list of the essential bits of the complete design summarized in the so-called EBD files. Unlike the bitstream that contains all the bits of the design, an EBD file is an ASCII file that has 32 characters per line that are either zero or one. A zero represents a non-essential bit and a one represents an essential bit. In other words, an EBD file contains a subset of the device configuration bits that are associated with the design. Therefore, injections in the configurable logic blocks and the routing of the FPGA could be performed. The main problem is that an EBD file is just a collection of zeros and ones. Therefore, a mechanism to extract and translate these characters into injection addresses for the SEM IP is needed.

Fig. 1 illustrates the integration of the proposed ACME tool into the reliability analysis of an FPGA design in which the SEM IP is used as a fault injector.

It can be observed in the previous figure that ACME requires two inputs from the user to generate the injection addresses for the SEM IP:

1) An EBD file that lists all the essential bits in the bitstream
2) The coordinates of the FPGA region in which the DUT is placed

Using these two inputs, ACME can extract the EBD file lines corresponding to the essential bits of the DUT and convert them to SEM IP injection addresses by implementing some equations published by Xilinx [28], [29]. These equations and the internal functioning of ACME will be explained in subsection IV-B. However, an usage example is presented first for a better understanding of the tool.

A. ACME: EXAMPLE OF USE

Let us suppose that we want to obtain the injection addresses of the TLB of a microprocessor deployed in an SRAM-based FPGA to be able to inject faults with the Xilinx SEM IP. First, the implementation of the microprocessor into the desired FPGA target architecture has to be performed to determine the number of resources used by the TLB. Then, a placement block (pBlock) that encloses the resources related with the TLB has to be drawn in the floorplan view of the FPGA as illustrated in Fig. 2.

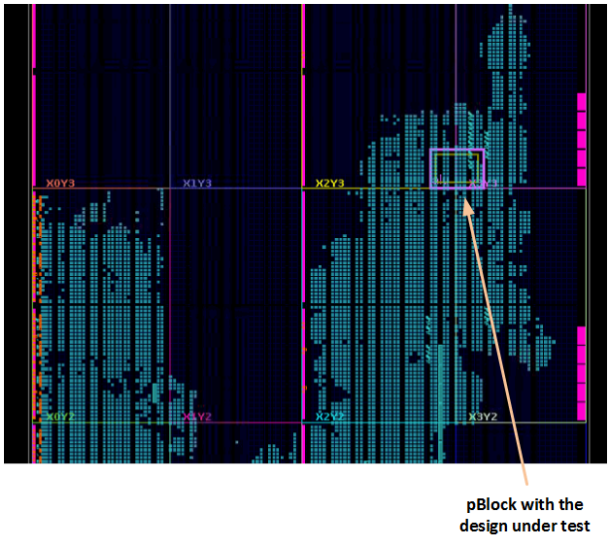


FIGURE 2. Floorplan of a design implemented in an FPGA in which a pBlock is containing the design under test (DUT).

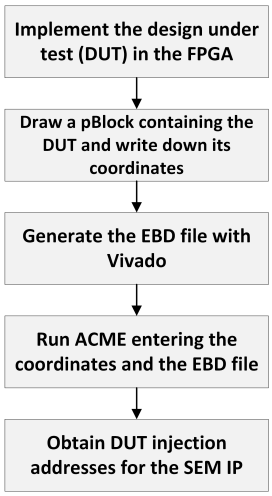
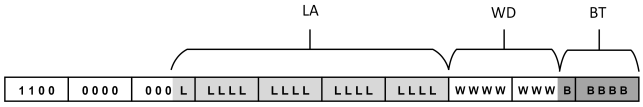


FIGURE 3. ACME usage flow chart.



Where:
- LA = linear frame address (17 bits)
- WD = word address (7 bits)
- BT = bit address (5 bits)

FIGURE 4. Error injection command.

A pBlock is a collection of cells inside one or more rectangular regions that group related logic and assign it to a physical region of the FPGA. Drawing a pBlock as the one shown in the previous figure can be done by selecting two pairs of coordinates in the floorplan view of the FPGA. These coordinates have to be noted down since they will be used as input parameters for the ACME tool. Using these coordinates, ACME will be able to extract the EBD file lines corresponding to that specific FPGA region defined by the pBlock. It should be mentioned that it is highly advisable to exclude from the pBlock those resources that are not part of the TLB. In this way, the injection of faults in other modules is avoided and the total running time of the campaign reduced. This pBlock exclusion and the generation of the mentioned EBD file can be done by adding the following commands in the Xilinx constraints file of the design project.

```
set_property EXCLUDE_PLACEMENT true   [get_pblocks MY_PBLOCK]
set_property BITSTREAM.SEU.ESSENTIALBITS yes [current_design]
```

The second command enables an algorithm already implemented in the Vivado Design Suite which automatically creates the EBD file together with the bitstream of the design. Once the EBD file is generated and the pBlock coordinates noted down, ACME can be executed to obtain the injection addresses of the TLB module that can be sent to the SEM IP to perform the fault injection campaign. For the sake of clarity, ACME usage steps explained before have been summarized in Fig. 3.

In the next subsection, the equations that ACME implements to generate the SEM IP injection addresses from the EBD file are explained in detail.

B. ACME: INTERNAL FUNCTIONING

The explanation of the internal behavior of ACME has been divided into two steps.

1) Translation from FPGA region to EBD file range
2) Conversion of the EBD lines to injection addresses

The first step is performed to extract the EBD file lines corresponding to the essential bits of the design under test (DUT). As mentioned in the previous subsection, the coordinates of a pBlock containing the DUT are required for it. ACME implements this step based on the linear correlation that exists between an FPGA region and the EBD lines. In this way, each FPGA clock region can be defined by a linear slope-intercept equation that uses the coordinates of the pBlock to obtain its EBD lines. These equations vary depending on the selected FPGA. If the target FPGA is not currently supported by ACME, it can be characterized and added to the tool by following the steps enumerated in the Appendix of this paper.

In the second step, the resulting EBD lines are converted to SEM IP injection addresses by using some equations published by Xilinx for their 7-series [28] and UltraScale [29] families. Each injection address has to be sent to the SEM IP by means of its monitor interface, which can be connected through an USB to UART converter to a serial terminal in a computer managed by the user. Once the connection is established, different commands can be sent to control the SEM IP behavior. One of these commands is the injection command, which has to be introduced when the SEM IP is in idle state. The structure of this command is shown in Fig. 4.

This command tells the SEM IP where it has to perform the bit flip, so the understanding of the linear frame address distribution is necessary. As mentioned before, an EBD file is an ASCII file with 32 characters (zeros or ones) per line. Each EBD file line represents a word of a configuration memory frame for the selected device. In the 7-series family for example, each frame consists of 101 words, i.e. 101 lines of the EBD file. Hence, equations given by (2) can be used to obtain the 17 bits of the linear frame address (LA), the 7 bits of the word (WD), and the 5 bits that comprise the bit address (BT) (see Fig. 4).

$$\begin{aligned} LA &= \frac{EBD\ line - (EBD\ line \bmod 101)}{101} - 1 \\ WD &= EBD\ line \bmod 101 \\ BT &= 31 - character\ position \end{aligned} \tag{2}$$

where *EBD line* is the line number of the EBD file in which an essential bit is found, *character position* is the position of each one in the file line, and mod represents the modulo operation. As mentioned before, each EBD line has 32 characters, these are the 32 bits of a word in which the least significant bit is on the far right of the line. Therefore, BT can be easily obtained by subtracting 31 to each character '1' found in the EBD line.

As an example of how to use these equations to obtain error injection commands as the one shown in Fig. 4, let us suppose that in line 134 of the EBD file we have the following 32 characters: 1000 0000 1000 0000 0000 0000 0000 0000. Therefore, we will obtain two injection commands since we have two essential bits in that line of the EBD file.

LA and WD can be calculated by using (2), obtaining LA = 0, and WD = 33 (which is 0100001 in binary). Then, we need to calculate the 5 bits of BT to complete each injection command. The first '1' is in the most significant bit position (*character position* = 0), so BT = 31 (which is 11111 in binary). The second '1' is in *character position* = 8, so BT = 23 (which is 10111 in binary). Finally, concatenating the 7 bits of WD and the 5 bits of BT the following hex numbers can be obtained:

$$010000111111 = 0 \times 43F \tag{3}$$

$$010000110111 = 0 \times 437 \tag{4}$$

Then, according to Fig. 4, the commands to be introduced should be C00000043F and C000000437 in hex format to inject in the two essential bits of that part of the design.

Similarly, this procedure can be applied to obtain the error injection commands for an UltraScale device by using the equations published in [29].

Combining step 1) explained at the beginning of this subsection with the previous equations, the error injection addresses for the SEM IP tool can be generated to inject errors only in those essential bits that belong to a specific submodule of the design. In the next section, the proposed tool will be tested with several designs to highlight the benefits that can be derived from its use.

V. TESTING THE TOOL

In this section, the design under test and the fault injection experiments performed to test ACME are explained below.

A. DESIGN UNDER TEST: THE RISC-V PROCESSOR

The design under test that will be used to demonstrate the ACME tool is the RISC-V processor. RISC-V is an open instruction set architecture (ISA) freely available to academia and industry developed by the Computer Science Division of the EECS Department at the University of California, Berkeley. This open-source ISA is nowadays attracting a lot of attention across the semiconductor industry mainly because the hardware/software collaborative ecosystem created around it. RISC-V is an extensible ISA with 32-bit and 64-bit address space variants and support for multicore or manycore implementations [30]. A RISC-V core has been chosen for the experiments because it is suitable for native hardware implementations and it follows the open-source philosophy of the tool presented in this paper.

In order to demonstrate the functionality of ACME, two experiments will be performed. The first one will consist in exercising the RISC-V core, while the second one will inject into a much smaller module, the instruction TLB (iTLB), in order to show the control on injections that ACME provides. Results will be compared with a scenario in which ACME is not available. In both experiments, the fault injection campaigns have been performed in a Kintex UltraScale FPGA KCU105 evaluation board with the SEM IP, which is controlled by a MATLAB script executed in an Intel x64 architecture running Windows 7 that sends the injection commands through a Digilent USB-UART peripheral module (PMOD). The correct/incorrect behavior of the microprocessor in the presence of errors is verified by executing two well-known benchmarks. The Dhrystone benchmark, which only contains integer operations, and the Whetstone benchmark, used to measure the floating-point arithmetic performance.

B. INJECTION CAMPAIGN ON THE RISC-V CORE

The full design that we are using comprises not only the RISC-V core, but also a set of peripherals, interfaces and routing. As described previously, an EBD file of the full design (which has 14,739,515 essential bits) can be obtained using Xilinx design tools. But let us assume that we are only interested in injecting in the RISC-V core (which has only 6,665,452 essential bits), in order to characterize its reliability, but not in the peripherals and the rest of the hardware. Using expression (1), we deduce that 9,604 is the number of injections to perform in order to get results with 95% confidence level and 1% error margin. Using ACME, we are able to obtain the specific bits of the EBD file that correspond to the RISC-V, discarding the rest of them. Therefore, we can guarantee that 100% of the injections will actually affect modules within the core. The whole injection process has taken 20 and 32 hours for the Dhrystone and the Whetstone benchmarks respectively, and the results are depicted in Table 1.

TABLE 1. Reliability results for the first experiment (RISC-V core).

	Without ACME		With ACME	
	Dhrystone	Whetstone	Dhrystone	Whetstone
No errors	9544 (99.4%)	9515 (99.1%)	9306 (96.9%)	9257 (96.4%)
Errors	60 (0.6%)	89 (0.9%)	298 (3.1%)	347 (3.6%)
Injections	9604 (100%)	9604 (100%)	9604 (100%)	9604 (100%)

TABLE 2. Reliability results for the second experiment (RISC-V iTLB).

	Without ACME		With ACME	
	Dhrystone	Whetstone	Dhrystone	Whetstone
No errors	9350 (99.4%)	9317 (99.1%)	8804 (93.6%)	8924 (94.9%)
Errors	55 (0.6%)	88 (0.9%)	601 (6.4%)	481 (5.1%)
Injections	9405 (100%)	9405 (100%)	9405 (100%)	9405 (100%)

Now, let us focus on a scenario without ACME. In this case, the full EBD file would have to be used, with all the essential bits including those of the peripherals, interfaces and routing. The experiment is repeated, obtaining the results shown in Table 1. Now, since there is no control in this case of where errors are injected, many of them would have stricken outside the core, decreasing the actual number of effective injections. These out-of-the-core injections have a limited impact in the RISC-V core operation, which leads to a lower number of produced errors, as can be seen in Table 1. In other words, time and effort has been devoted on a set of injections outside the region that we are testing. To solve this, and get results that are equivalent to the scenario using ACME, we would need to perform more injections. In a rough approach, the size of the RISC-V core is approximately 45% the size of the full design. Therefore, we would need a total number of 21,237 injections to guarantee that, statistically, 9,604 will actually affect the RISC-V core. But this would also imply an execution time of 45 and 71 hours for the Dhrystone and the Whetstone benchmarks respectively, slowing down the whole injection campaign. Therefore, not using ACME would imply either i) getting results with worse quality or ii) needing more time to perform the experiment. The latter can be unfeasible in many situations in which a large number of injection campaigns is desired.

C. INJECTION CAMPAIGN ON THE RISC-V ITLB

The previous experiment has showcased the convenience of ACME in order to perform fully-controlled injection campaigns. The benefit increases as the target area to inject is smaller respect to the total design under test area. To demonstrate this, let as assume now that we want to characterize the RISC-V instruction TLB (iTLB), e.g. to test a specific protection technique designed ad-hoc for this module. In this case, using ACME, the specific essential bits of the iTLB can be extracted. The number of essential bits, in this case, is 452,749. Therefore, the total number of statistical injections is 9,405 according to expression (1) to get results with 95% confidence level and 1% error margin. Again, all the performed injections are guaranteed to affect essential bits of the iTLB. The injection campaign took 19.5 hours for the Dhrystone benchmark and 31 hours for the Whetstone benchmark, and the obtained results can be seen in Table 2.

Now, let us try to perform a similar experiment without using ACME. In this case, we cannot distinguish which of the bits in the original EBD file correspond to the iTLB and which to other parts of the RISC-V. The results in this case, using the same number of injections, are also depicted in Table 2. Now, since the iTLB is a smaller fraction of the full RISC-V design (around 3%), fewer injections would have actually affected the iTLB. Therefore, the obtained results would provide less quality than in the case of using ACME. In order to solve this, as happened in the previous experiment, more injections would have to be performed, in order to guarantee that the same number of them affects the iTLB. Statistically, 306,185 would be the number of injections in this case, which represents an increment of 32.5× respect to the original number. But this would also imply a significant increment on the injection time, 638 hours for the Dhrystone benchmark and 1,021 hours for the Whetstone in this case. This impact on time may be unfeasible in many experiments, thus leading to shorter injection campaigns and therefore results with lower quality. In summary, the smaller the module to characterize is, the higher benefit using ACME, since the injection process will be more controlled and efficient.

But this is not the only problem. Without ACME, many of the injected errors have not affected the actual iTLB, but instead other modules of the RISC-V that are also critical. These may have induced errors in the system, counted in the total figures of Table 2. But these do not represent an actual malfunctioning of the iTLB, because they have been produced by other modules not currently under test. In other words, we are not strictly characterizing the reliability of the iTLB, since there are other modules being affected. This is one of the main advantages of ACME: not only we reduce the campaign time by guaranteeing that all the injections actually affect the target area, but we also avoid injecting in other modules of the design whose misbehavior can distort the overall experimental results.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, ACME, a new tool to pinpoint fault injections in specific regions of Xilinx SRAM-based FPGAs is presented. As illustrated in the experimental results, the main advantages of using the proposed tool are the acceleration of the injection campaign up to 32.5× and the ability to exclude other modules of the design by injecting only in those bits related to the current circuit under test. In this way, injection side-effects that modify the experimental results can be avoided and the number of injections to be performed can be reduced. As a result, better-quality statistical fault injection campaigns can be done with a more reduced number of injections.

APPENDIX
EXTENDING THE TOOL

Currently, ACME supports a subset of FPGA architectures such as the ZedBoard, the Basys3 Artix-7, the Nexys 4 DDR, the ZC706, and the KCU105 UltraScale. For further extensions of the tool to other target architectures, the reverse engineering process explained down below has to be followed to identify the correlation between each FPGA region and its corresponding EBD file lines.

1) A small design (e.g. an inverter) is placed in one of the four corners of an FPGA clock region
2) An EBD file is generated for that location
3) The '1s' in the EBD file (the essential bits of the inverter) are searched and the EBD line numbers in which these '1s' appear are stored
4) Steps 1-3 are repeated for the remaining three corners of the selected FPGA clock region
5) Since the correlation between FPGA region and EBD line is linear, a slope and an intercept is calculated by using the EBD line numbers and the clock region coordinates obtained in the previous steps
6) The slope and intercept for that clock region are added to the ACME source code as defined constants
7) All the previous steps are repeated for every clock region to characterize the entire device

Once the board is characterized and the constant values added to ACME, the tool has to be compiled to create its executable file, which can now be used to generate the injection addresses for the desired FPGA target architecture.

REFERENCES

[1] 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," *IEEE Trans. Electron Devices*, vol. 57, no. 7, pp. 1527–1538, Jul. 2010.

[2] J. L. Autran and D. Munteanu, "Radiation and COTS at ground level," *Microelectron. Rel.*, vol. 55, nos. 9–10, pp. 2147–2153, 2015.

[3] J. W. Howard and D. M. Hardage, "Spacecraft environments interactions: Space radiation and its effects on electronic systems," NASA, Washington, DC, USA, Tech. Rep. TP-1999-209373, Jul. 1999.

[4] S. Cristoloveanu and V. Ferlet-Cavrois, "Introduction to SOI MOSFETs: Context, radiation effects, and future trends," in *Radiation Effects and Soft Errors in Integrated Circuits and Electronic Devices*, R. D. Schrimpf and D. M. Fleetwood, Eds. Singapore: World Scientific, 2004.

[5] T. R. Oldham and F. B. McLean, "Total ionizing dose effects in MOS oxides and devices," *IEEE Trans. Nucl. Sci.*, vol. 50, no. 3, pp. 483–499, Jun. 2003.

[6] J. R. Srour, C. J. Marshall, and P. W. Marshall, "Review of displacement damage effects in silicon devices," *IEEE Trans. Nucl. Sci.*, vol. 50, no. 3, pp. 653–670, Jun. 2003.

[7] T. Karnik, P. Hazucha, and J. Patel, "Characterization of soft errors caused by single event upsets in CMOS processes," *IEEE Trans. Depend. Sec. Comput.*, vol. 1, no. 2, pp. 128–143, Jun. 2004.

[8] R. C. Lacoe, J. V. Osborn, R. Koga, S. Brown, and D. C. Mayer, "Application of hardness-by-design methodology to radiation-tolerant ASIC technologies," *IEEE Trans. Nucl. Sci.*, vol. 47, no. 6, pp. 2334–2341, Dec. 2000.

[9] K.-H. Huang and J. A. Abraham, "Algorithm-based fault tolerance for matrix operations," *IEEE Trans. Comput.*, vol. C-33, no. 6, pp. 518–528, Jun. 1984.

[10] R. Koga, W. A. Kolasinski, M. T. Marra, and W. A. Hanna, "Techniques of microprocessor testing and SEU-rate prediction," *IEEE Trans. Nucl. Sci.*, vol. NS-32, no. 6, pp. 4219–4224, Dec. 1985.

[11] B. Alpat, R. Battiston, M. Bizzarri, D. Caraffini, E. Fiori, A. Papi, M. Petasecca, and A. Pontetti, "The radiation sensitivity mapping of ICs using an IR pulsed laser system," *Microelectron. Rel.*, vol. 43, no. 6, pp. 981–984, 2003.

[12] J. S. Melinger *et al.*, "Critical evaluation of the pulsed laser method for single event effects testing and fundamental studies," *IEEE Trans. Nucl. Sci.*, vol. 41, no. 6, pp. 2574–2584, Dec. 1994.

[13] N. A. Harward, M. R. Gardiner, L. W. Hsiao, and M. J. Wirthlin, "Estimating soft processor soft error sensitivity through fault injection," in *Proc. IEEE 23rd Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, May 2015, pp. 143–150.

[14] A. Ullah, P. Reviriego, and J. A. Maestro, "An efficient methodology for on-chip SEU injection in flip-flops for Xilinx FPGAs," *IEEE Trans. Nucl. Sci.*, vol. 65, no. 4, pp. 989–996, Apr. 2018.

[15] L. A. B. Naviner, J.-F. Naviner, G. G. dos Santor, Jr., E. C. Marques, and N. M. Pavia, Jr., "FIFA: A fault-injection–fault-analysis-based tool for reliability assessment at RTL level," *Microelectron. Rel.*, vol. 51, pp. 1459–1463, Sep./Nov. 2011.

[16] P. Civera, L. Macchiarulo, M. Rebaudengo, M. S. Reorda, and M. Violante, "New techniques for efficiently assessing reliability of SOCs," *Microelectron. J.*, vol. 34, no. 1, pp. 53–61, Jan. 2003.

[17] D. Gonzalez, "Single event upset simulation tool functional description," Eur. Space Agency, Paris, France, ESA Rep. TEC-EDM/ DCC-SST2, 2004.

[18] C. López-Ongil, L. Entrena, M. Garcia-Valderas, M. Portela, M. A. Aguirre, J. Tombs, V. Baena, and F. Munoz, "A unified environment for fault injection at any design level based on emulation," *IEEE Trans. Nucl. Sci.*, vol. 54, no. 4, pp. 946–950, Aug. 2007.

[19] *Soft Error Mitigation Controller LogiCORE IP Product Guide (PG036)*, Xilinx, San Jose, CA, USA, Sep. 2015.

[20] J. M. Mogollon, H. Guzmán-Miranda, J. Nápoles, J. Barrientos, and M. A. Aguirre, "FTUNSHADES2: A novel platform for early evaluation of robustness against SEE," in *Proc. 12th Eur. Conf. Radiat. Effects Compon. Syst.*, Sevilla, Spain, Sep. 2011, pp. 169–174.

[21] M. Alderighi, F. Casini, S. D'Angelo, S. Pastore, G. R. Sechi, and R. Weigand, "Evaluation of single event upset mitigation schemes for SRAM based FPGAs using the FLIPPER fault injection platform," in *Proc. IEEE 22nd Int. Symp. Defect Fault-Tolerance VLSI Syst. (DFT)*, Rome, Italy, Sep. 2007, pp. 105–113.

[22] L. Sterpone and M. Violante, "A new partial reconfiguration-based fault-injection system to evaluate SEU effects in SRAM-based FPGAs," *IEEE Trans. Nucl. Sci.*, vol. 54, no. 4, pp. 965–970, Aug. 2007.

[23] G. L. Nazar and L. Carro, "Fast single-FPGA fault injection platform," in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Nanotechnol. Syst. (DFT)*, Austin, TX, USA, Oct. 2012, pp. 152–157.

[24] U. Legat, A. Biasizzo, and F. Novak, "Automated SEU fault emulation using partial FPGA reconfiguration," in *Proc. IEEE 13th Symp. Design Diagnostics Electron. Circuits Syst. (DDECS)*, Vienna, Austria, Apr. 2010, pp. 24–27.

[25] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert "Statistical fault injection: Quantified error and confidence," in *Proc. Conf. Design, Automat. Test Eur.*, Nice, France, 2009, pp. 502–506.

[26] R. Le, "Soft error mitigation using prioritized essential bits, Version 1.0," Xilinx, San Jose, CA, Appl. Note XAPP538, 2012.

[27] J. A. Maestro. (2019). ARIES Research Center Website. Universidad Nebrija. Accessed: Jul. 3, 2019. [Online]. Available: http://www.nebrija.es/aries/acme.htm

[28] Xilinx. (2016). *7 Series—SEM IP—How to Use the SEM IP Error Report to Look Up Bit Error Locations Using Essential Bit Data in an EBD File? Xilinx, AR# 67337*. Accessed: Jul. 2, 2019. [Online]. Available: https://www.xilinx.com/support/answers/67337.html

[29] Xilinx. (2016). *UltraScale—SEM IP—How to Use the SEM IP Error Report to Look Up Bit Error Locations Using Essential Bit Data in the EBD File? Xilinx, AR# 67086*. Accessed: Jul. 2, 2019. [Online]. Available: https://www.xilinx.com/support/answers/67086.html

[30] A. Waterman and K. Asanovic, *The RISC-V Instruction Set Manual, Volume I: User-Level ISA*, RISC-V Foundation, Document Version 20190621-Draft, Jun. 2019.

LUIS ALBERTO ARANDA received the B.Sc. degree in industrial engineering and the M.Sc. degree in robotics from the Universidad Carlos III de Madrid, Spain, in 2012 and 2015, respectively, and the Ph.D. degree (Hons.) in industrial engineering from the Universidad Antonio de Nebrija, Madrid, Spain, in 2018.

He was a Project Engineer with Zeus Creative Technologies S.L., developing various computer vision projects, from 2013 to 2014. He was responsible for both hardware and software design and implementation. He is currently with the ARIES Research Center, Universidad Antonio de Nebrija. He is the author of several technical publications in journals and international conferences. His current research interests include reconfigurable computing for space applications, computer vision, and robotics.

ALFONSO SÁNCHEZ-MACIÁN received the M.Sc. and Ph.D. degrees in telecommunications engineering from the Universidad Politécnica de Madrid, Madrid, Spain, in 2000 and 2007, respectively.

He was a Lecturer and a Researcher at several universities, such as the Universidad Politécnica de Madrid; the IT Innovation Centre, University of Southampton, Southampton, U.K.; and the Universidad Antonio de Nebrija, Madrid, where he is currently a part of the ARIES Research Center. He was with numerous national and multinational companies as a Project Manager and a Senior Consultant for IT projects. His current research interests include fault-tolerance and reliability, performance evaluation of communication networks and knowledge representation, and reasoning in distributed systems.

JUAN ANTONIO MAESTRO (M'07–SM'15) received the M.Sc. degree in physics and the Ph.D. degree in computer engineering from the Universidad Complutense de Madrid, Madrid, Spain, in 1994 and 1999, respectively.

Since 2004, he has been directed the Electronic Design and Space Technology Research Group, Universidad Antonio de Nebrija, where he has also recently founded the ARIES Research Center (www.nebrija.es/aries), devoted to the aerospace research and innovation in electronic systems. His current activities are oriented to the space industry, with several projects on the protection of digital circuits against the effects of radiation, including microprocessors, memories, and auxiliary systems. He also collaborates with institutions, such as the European Space Agency, Stanford University, University College Dublin, and/or the Harbin Institute of Technology. He is the author of numerous technical publications in journals and international conferences. His current research interests include computer architecture, digital design, fault-tolerance, reliability, small satellites, and space applications.