IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 4, 2021, accepted October 9, 2021, date of publication October 13, 2021, date of current version October 21, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3119633

Reliability Analysis of ASIC Designs With Xilinx SRAM-Based FPGAs

LUIS ALBERTO ARANDA[1], OSCAR RUANO[1], FRANCISCO GARCIA-HERRERO[1], AND JUAN ANTONIO MAESTRO[2], (Senior Member, IEEE)

[1]ARIES Research Center, Universidad Antonio de Nebrija, 28040 Madrid, Spain
[2]Department of Computer Architecture and Automatics, Computer Science Faculty, Complutense University of Madrid, 28040 Madrid, Spain

Corresponding author: Luis Alberto Aranda (laranda@nebrija.es)

ABSTRACT There are many platforms and tools based on field-programmable gate array (FPGA) devices oriented to facilitate the reliability estimation of digital designs, but they are usually focused only on configuration memory errors since the configuration memory represents the majority of the memory elements in an FPGA. However, an FPGA-based platform could also be exploited to support the emulation of transient and permanent errors for designs intended to work in application-specific integrated circuits (ASICs) or radiation-hardened devices such as antifuse FPGAs. In this context, the obtention of a particular set of bits to flip is required to be able to emulate these error models. The main difficulty of this approach lies in determining the mentioned set of bits, which is due to the unavailability of a public description of the bitstream and the lack of FPGA architecture details. To help with this issue, we present a methodology to determine specific configuration memory bits from SRAM-based FPGAs that, when flipped, emulate permanent or transient upsets in any flip-flop element of the design under test. This methodology is proved in recent FPGA technologies and provides great control and precision in reliability experiments for harsh environments.

INDEX TERMS ASIC, configuration memory, emulation, fault injection, FPGA, reliability.

I. INTRODUCTION

Hardware-based fault emulation is a widely used approach to test the reliability of designs intended to work in harsh environments such as space, nuclear, or, more recently, quantum processors [1]–[3]. Compared to accelerated beam tests carried out in radiation facilities [4] or physical reliability tests perfomed in rooms at cryogenic temperatures, fault emulation approaches provide great versatility to test different designs in a short amount of time while simultaneously keeping costs low. Besides that, fault emulation can reduce timing requirements and provides more realistic results than simulation-based set-ups, which can be used to perform an accurate analysis of the design's behavior [5]. These designs may be deployed in field-programmable gate array (FPGA) or application-specific integrated circuits (ASICs), hence the error model differs and the platform's hardware has to be carefully selected.

Over the years, FPGAs have been traditionally used as fault emulation platforms. In particular, static random access memory (SRAM)-based FPGAs tend to be the preferred technology due to their low cost, accessibility, and reconfigurable capabilities [6]. SRAM-based FPGAs can be conceptually split into two layers. An application layer containing the bits that are dynamically managed by the user's design (e.g. the bits stored in flip-flop elements), and a configuration layer including the bits related to the configuration of the logic, memory, and routing resources (e.g. the structure of the design). Therefore, the FPGA error model, which is mainly focused on persistent errors, can be emulated by flipping the design-related configuration bits. Similarly, the ASIC error model, which comprehends both transient and permanent errors, can be emulated by inverting the bits in the application layer. The latter are particularly difficult to locate and modify since, as mentioned before, these bits are dynamic. For this reason, we propose a different approach to emulate the ASIC error model in an SRAM-based FPGA device through the testing of a selected group of configuration memory bits. These bits are static, but their modification will lead to the same behavior as modifying the bits in the application layer.

ASIC devices or radiation-hardened FPGAs such as antifuse FPGAs are sensitive to permanent errors (i.e. stuck-at

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

faults) and transient events that may eventually be registered by a flip-flop element creating a single-event upset (SEU) or may be masked by the subsequent logic elements of the design. On the other hand, a configuration memory bit flip typically produces persistent errors that modifies the structure or the routing of the design under test (DUT). Therefore, transient and permanent errors in flip-flop elements associated to ASIC or rad-hard FPGA designs could be emulated in an SRAM-based FPGA by flipping the configuration bits related to the flip-flop inputs. The two main difficulties of this approach are:

1) Configuration memory errors lead to persistent errors that can only be removed by reprogramming the FPGA or by flipping the bit again to its initial state. Therefore, permanent errors could be easily emulated, but transient errors should be carefully introduced and removed during the fault injection campaign to emulate the desired behavior.
2) Identifying the configuration bits to emulate both permanent and transient errors is a time-consuming and non-trivial task.

The first point is a matter of adjusting the duration of the bit flip by controlling the design clock signal and it is related to the fault injection campaign itself. In contrast, the second point is related to the lack of a configuration bit identification mechanism available. In this paper, a methodology to facilitate the ASIC or rad-hard FPGA error emulation in an SRAM-based FPGA platform is presented. It is based on the previously mentioned approach of emulating both transient and permanent errors in flip-flop elements through configuration memory bit flips. To test our idea, the reliability of several designs is assessed through fault injection. In these experiments, a set of configuration bits for every design flip-flop is translated into injection addresses for the Xilinx Soft Error Mitigation (SEM) IP Core [7]. This fault injector is then used to perform the reliability campaign in a Xilinx FPGA device.

The rest of the paper is organized as follows. Section II presents other fault injection approaches and platforms and explains the motivation behind the methodology. The methodology details are described in Section III. In Section IV, some example designs are tested following the methodology to evaluate its benefits. Finally, Section V summarizes the contributions of this work.

II. EXISTING FAULT EMULATION PLATFORMS

Nowadays, there are different approaches to increase the fault tolerance of a system intended to work in harsh environments. Hardening-by-process strategies are based on modifying the manufacturing process to increase the reliability of the device. Conversely, there are techniques based on adding some sort of redundancy to the original design that can be applied to commercial off-the-shelf (COTS) devices. Some well-known examples are triple modular redundancy (TMR), error correction codes (ECC), or techniques based on the system knowledge [8], [9].

Once the design under test has been protected through one or several of the above techniques, it is required to test the reliability level achieved by the protection using fault injection platforms to generate bit flips into the design's memory elements emulating different environments. A schematic diagram of these types of platforms is presented in Fig. 1 for illustrative purposes.

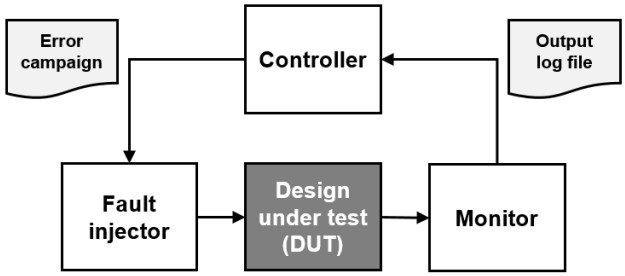


FIGURE 1. Schematic diagram of fault injection environment based on hardware emulation.

The controller in Fig. 1 manages the fault injection campaign, while the fault injector carries out the bit flips at the specific locations of the target DUT told by the controller. During the campaign, the effects of each introduced bit flip are typically logged in a file or stored for later processing.

In recent years, both simulation and emulation-based fault injection platforms have been documented in the literature. Simulation environments or tools such as XCEPTION [10], SST [11], MEFISTO [12], or VERIFY [13] are some examples. They all try to reproduce the behavior of a hardware circuit under specific undesirable malfunctions from its high-level description by using a commercial or ad-hoc simulator. However, the physical behavior of a complex circuit is usually difficult to mimic and requires to constrain the platform to analyze specific cases of interest. In contrast, emulation-based platforms may be used to test different scenarios just by selecting the resources and applying the conditions of the harsh environment under test. But, in this case, the difficulty lies in identifying those resources and modifying them in a way that imitate the physical effect. Some examples of emulation-based fault injection systems for SRAM-based FPGAs are the FT-UNSHADES [14], FLIPPER [15], SPFFI [16], XRTC [17], NESSY [18], or FIJI [19]. One major limitation of these hardware emulation platforms is that the ASIC error model is not usually supported. In fact, some of these platforms are focused on a specific technology or device and quickly become obsolete due to the lack of portability. For a detailed comparison between the different tools we refer to [20].

Apart from the above platforms, Xilinx and Intel (Altera) vendors have opted for providing proprietary methods to perform fault injection in their corresponding devices. Both alternatives are integrated into their development environments and are focused on facilitating user interaction.

- Intel provides the Fault Injection Debugger as part of its Quartus II hardware design software. It is capable of performing fault injection in the FPGA configuration memory by toggling one bit per injection thus emulating an SEU in runtime [21].
- Xilinx's Vivado Design Suite includes the SEM IP Core that is able to perform fault injection in the FPGA's configuration memory as the Intel tool. The user can select the specific configuration memory bit to flip by introducing its physical or linear address [7].

The previous tools allow the user to inject configuration memory errors to test its designs. However, adapting the fault injection campaign to test the desired error model and DUT regions is complex and time-consuming since the FPGA architecture, the bitstream, and the configuration memory details are not provided by the vendor.

In order to help reliability designers with this issue, we presented a tool named ACME (automatic configuration memory error-injection) in [22]. ACME determines the configuration bits associated with the DUT by using the information of the static bits collected in the Xilinx essential bit data (EBD) files. In this manner, the FPGA error model can be exercised with the Xilinx SEM IP fault injector.

In this present work, we introduce a methodology to enable the injection of transient and permanent errors in flip-flop resources to emulate the ASIC error model. The main idea behind this work lies in reusing the experimental set-up already created for ACME and extending it to support the ASIC error model. This new approach requires logic location files (.ll files) as input for the translation to SEM IP injection addresses because EBD files do not contain explicit information about flip-flop resources.

For clarification purposes, Fig. 2 is presented as an overview of a dual fault injection emulation procedure in which both FPGA and ASIC error models can be assessed depending on the user's needs. For the FPGA error model, ACME can be used as translation tool. For the ASIC error model, the methodology presented below can be followed to deal with the difficulty of locating the dynamic bits in the FPGA application layer. As can be observed in this figure, the SEM IP is kept as the fault injector since the ASIC error model is mimicked by flipping the configuration memory bits related to the flip-flop elements of the DUT.

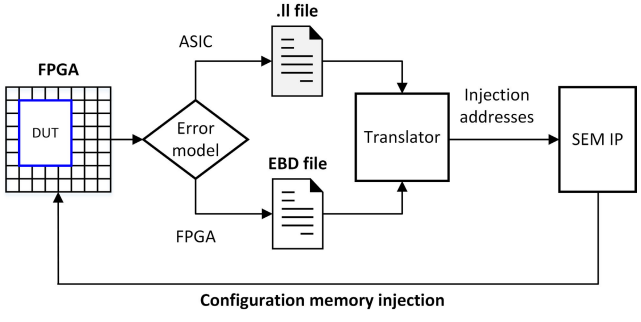


FIGURE 2. Integration of the ASIC emulation in an FPGA fault injection procedure.

III. METHODOLOGY DESCRIPTION

To create a state-of-the-art portable set-up able to emulate transient and permanent errors in Xilinx FPGAs, SEM IP has been applied. The SEM IP [7] is a standalone module designed by Xilinx to mitigate the effects of soft errors in the FPGA configuration memory. It can also be controlled via serial port to inject errors into a particular configuration memory bit by providing its physical or linear address. The SEM IP is well-documented and has been upgraded to support newer UltraScale and UltraScale+ devices, ensuring the continuity of the emulation platform and the portability between FPGA families. However, it presents an important limitation related to its use as a fault injector. The main question that arises when the user develops an experimental set-up based on the SEM IP is: how should I know the proper bits to test? This question cannot be answered by Xilinx since it implies uncovering proprietary information, and a "blind" statistical injection campaign is usually recommended. Therefore, it is fundamental to have a methodology to obtain the desired bits and gain control over the injection campaign to deal with this gap. As mentioned in the previous section, we have already developed ACME as a tool to enable the emulation of persistent errors in the configuration memory with the SEM IP following the FPGA error model. In this present work, we propose an approach to emulate the ASIC error model by modifying a specific subset of configuration bits of the design flip flops. The main idea is to locate the configuration bits associated to the set/reset input of each flip-flop to emulate both permanent and transient errors of the ASIC error model:

- Permanent errors could be emulated by triggering and holding the set/reset signal indefinitely. While the set/reset signal is triggered, the value stored and outputted by the affected flip-flop will remain constant (and will be equal to its initialization value). Therefore, the initialization value has to be manipulated as well to emulate both logic '0' and logic '1' scenarios.
- Transient errors of any duration could be emulated by triggering and releasing the set/reset signal at any moment. While the set/reset signal is triggered, the content of the affected flip-flop will not be updated by new input values. Once the set/reset signal is released by correcting the injected error, the flip-flop returns to its normal behavior.

In this manner, the difficulty of locating the dynamic bits in the application layer related to the flip-flop elements is reduced and the injection of permanent and transient errors in these memory elements is enabled to continue filling the mentioned gap. With this approach, the reliability of the sequential elements of any design can be evaluated as well as its combinational part by triggering a flip-flop upset and observing its propagation along the combinational elements.

A. CONFIGURABLE LOGIC BLOCK ARCHITECTURE

In Xilinx FPGAs, the configurable logic block (CLB) is the core resource for implementing general-purpose combinatorial and sequential circuits [23]. It contains look-up tables (LUTs), flip-flops, and other logic elements.

In the UltraScale family, for example, a CLB consists of 16 flip-flops labeled AQ, AQ2, BQ, BQ2, ...to HQ, HQ2 from bottom to top (see Fig. 3). In every CLB, there are two global set/reset (GSR) inputs, each of them dedicated to 8 of the 16 flip-flops, and one initialization (INIT) attribute per flip-flop (i.e. 16 INIT values in total).

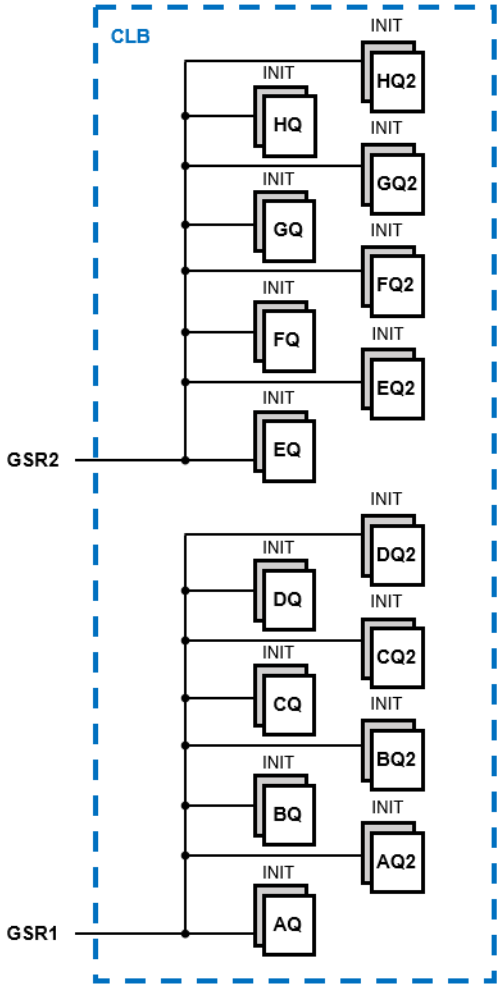


FIGURE 3. GSR signals and INIT attributes in a CLB of the UltraScale family. The remaining connections of the flip-flops have been omitted for clarity.

The INIT value (a logic '0' or a logic '1') can be different for each flip-flop and it is set after the configuration of the device. The INIT values are stored in a different storage element associated with each flip-flop. The initialization values are only released and stored in the flip-flop when the GSR signal is triggered. This means that both a logic '0' or '1' could be generated by altering these values. For example, if the default INIT value of a flip-flop is '0' then, a stuck-at-0 or a transient '0' could be generated at any moment by triggering the GSR signal. Similarly, a stuck-at-1 or a transient '1' could be generated in the same flip-flop by first performing a bit flip in the INIT value and then triggering the GSR signal. This procedure has the advantage of testing the two possible logic values in a flip-flop without even knowing the INIT value. In addition, the procedure can be directly applied to other Xilinx families such as the 7-series since their CLBs also contain the GSR and INIT signals and work in the same way.

The status of the GSR signal and the INIT values are stored in the configuration memory cells of the FPGA and, therefore, they could be modified by the Xilinx SEM IP to generate events in the flip-flops. The main difficulty is, again, to determine the precise injection addresses of the GSR and INIT values to perform the mentioned bit flips. To deal with this difficulty, we present a methodology and an example of use in the next subsection.

B. METHODOLOGY STEPS AND EXAMPLE OF USE

The purpose of this section is to present an experimental methodology to determine the injection addresses that have to be sent to the Xilinx SEM IP to alter the GSR and the INIT values of the flip-flops. Using these addresses, the ASIC error model can be emulated in an FPGA by releasing the initialization value of the flip-flop or its opposite.

In order to determine the addresses of the GSR and the INIT values, a CLB of the FPGA device has to be characterized. This means that sixteen INIT and two GSR addresses have to be found experimentally. The good thing is that, once a particular CLB is characterized, certain fields in the injection addresses are constant over the entire FPGA device, and the rest of the fields can be easily obtained from the logic location file of the DUT as will be described later.

The first step in the methodology is to create the design under test. A simple design is recommended during the first time characterizing a CLB of the target FPGA. Once the FPGA is characterized, any other DUT can be chosen. In our case, we have chosen an 8-bit register as a design to characterize the FPGA since its output values are directly related to its inputs and any change can be easily monitored using LED indicators. The second step is to obtain the logic location file of the DUT, file that has to be processed to determine specific fields of the SEM IP injection address. Most of the fields such as the row, column, and word addresses can be directly obtained from this file. However, during the first time following the methodology, the minor and the bit fields have to be experimentally determined. At all other times, the values can be loaded and reused. Finally, after determining the values for each field, they can be merged and collected in a text file. This text file can then be used by the SEM IP to perform the desired fault injection campaign.

For clarification purposes, the steps of the proposed methodology to emulate the ASIC error model in an SRAM-based FPGA are detailed in Fig. 4. It can be observed in this flowchart that the fifth step changes depending on whether or not it is the first time following the methodology.

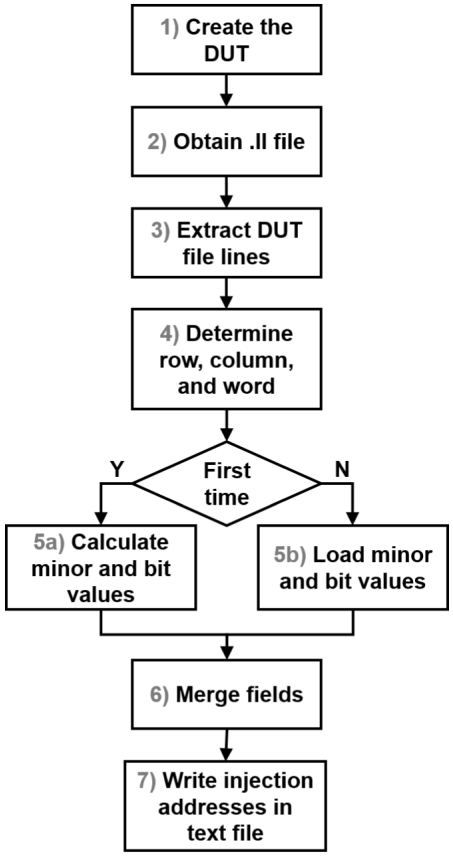


FIGURE 4. Steps of the proposed experimental methodology to generate the SEM IP injection addresses of the GSR signal and the INIT values from a logic location file.

This is representing the experimental characterization that has to be done once for the first time. Then, with the FPGA characterized, the methodology can be automated by reusing the previous values.

The steps of the methodology are explained below in detail following the mentioned 8-bit register example.

1) CREATE THE DUT

As mentioned before, a CLB of the target FPGA device has to be first characterized. To do so, an 8-bit register has been chosen as a design under test. This example design will be used to find the GSR and INIT addresses of the CLBs of a Kintex-UltraScale KCU105 FPGA. After the characterization of this device is done, the test register can then be replaced by the desired DUT and most of the steps explained below can be automated.

To facilitate the characterization of the CLB, the reset and the D inputs of the register flip-flops have been mapped to slide switches. Similarly, each Q output has been connected to an LED indicator for visual inspection. This will help us to monitor the status of each individual flip-flop more easily.

2) OBTAIN .LL FILE

Once the DUT is created, the next step is to generate its logic location file. A logic location file (.ll file) is an ASCII file provided by Xilinx containing information about all the nodes in the design from where a readback operation can be performed. This means that the logic location file can be taken as a reference to obtain information about the flip-flop elements. However, the information in this file is not enough to determine the complete injection address of the GSR signal and the INIT values associated with each flip-flop. Therefore, and as mentioned before, some experimental tests have to be performed to characterize the CLB. The logic location file of any DUT can be generated together with its bitstream by activating the *logic_location_file* option in the bitstream settings of Xilinx's Vivado project.

3) EXTRACT DUT FILE LINES

Continuing with the 8-bit register example, the .ll file lines obtained for this design are presented in Fig. 5. These lines can be easily extracted by searching the name of the design under test (i.e. DUT in this case).

```
Bit   48879104 0x00042584 1856 SLR0 0 Block=SLICE_X37Y149 Latch=AQ Net=DUT/Q[0]
Bit   48879108 0x00042584 1860 SLR0 0 Block=SLICE_X37Y149 Latch=BQ Net=DUT/Q[2]
Bit   48879112 0x00042584 1864 SLR0 0 Block=SLICE_X37Y149 Latch=CQ Net=DUT/Q[4]
Bit   48879116 0x00042584 1868 SLR0 0 Block=SLICE_X37Y149 Latch=DQ Net=DUT/Q[6]
Bit   48879120 0x00042584 1872 SLR0 0 Block=SLICE_X37Y149 Latch=AQ2 Net=DUT/Q[1]
Bit   48879124 0x00042584 1876 SLR0 0 Block=SLICE_X37Y149 Latch=BQ2 Net=DUT/Q[3]
Bit   48879128 0x00042584 1880 SLR0 0 Block=SLICE_X37Y149 Latch=CQ2 Net=DUT/Q[5]
Bit   48879132 0x00042584 1884 SLR0 0 Block=SLICE_X37Y149 Latch=DQ2 Net=DUT/Q[7]
```

FIGURE 5. Logic location file fragment showing the eight lines (one line per flip-flop) of an 8-bit register named DUT.

It can be observed that each flip-flop in the register has an associated file line. In each line, the following information is listed from left to right: bit offset, frame address, frame offset, super-logic region (SLR) name, SLR number, and information about the block, latch and net names. Among these fields, only the frame address and the frame offset are required to generate the injection addresses of the GSR and the INIT values for the SEM IP. With the information from these fields, the SEM IP can be used to emulate both transient and permanent events in the configuration memory of the FPGA as will be described later.

4) DETERMINE ROW, COLUMN, AND WORD

As explained in its manual [7], the SEM IP supports both linear and physical addressing formats. The physical format is related to the physical location of the bit to flip, while the linear format is based on an linear organization of the FPGA frames and does not provide any information about type and physical location of the frame.

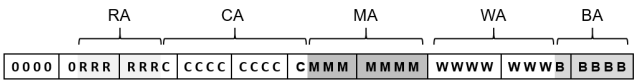
Analyzing the fields of the frame address in the .ll file (see Table 1 [24]) it can be observed that these files contain

TABLE 1. Fields of the frame address in a logic location file [24].

Field	Bit index
Row address	[22:17]
Column address	[16:7]
Minor address	[6:0]

physical information and thus, it is easier to directly provide it to the SEM IP using the physical injection address format.

The SEM IP physical addressing format contains 40 bits which are distributed on the fields shown in Fig. 6.



Where:
- RA = row address (6 bits)
- CA = column address (10 bits)
- MA = minor address (7 bits)
- WA = word address (7 bits)
- BA = bit address (5 bits)

FIGURE 6. SEM IP physical injection address. Figure adapted from [7].

Therefore, the information in the .ll file has to be translated to fill in the row, column, minor, word, and bit fields shown in Fig. 6. The row, column and minor values are related to the frame address presented before while the word and bit fields are related to the frame offset that will be described later. Taking the frame address in the example presented in Fig. 5, 0×00042584 gives $RA = 000010$, $CA = 0001001011$, and $MA = 0000100$. On the other hand, the word and bit fields can be calculated using the frame offset in the .ll file and knowing that a word is made up of 32 bits. With this information, the translation from frame offset to physical word and bit addresses can be done as presented in equations (1) and (2).

$$WA = floor\left(\frac{frameoffset}{32}\right) \tag{1}$$

$$BA = frameoffset \bmod 32 \tag{2}$$

In the example case, this gives $WA = 58 = 0111010$ for every flip-flop of the 8-bit register and, for example, $BA = 0 = 00000$ for the Q[0] flip-flop. However, both the minor and the bit addresses obtained directly from the .ll file are not valid since they are related to the readback node. Therefore, the correct values will have to be found experimentally. But, these values obtained directly from the .ll file can be taken as a starting point for these experimental tests.

5) CALCULATE OR LOAD MINOR AND BIT VALUES

As mentioned before, the minor and the bit values for the GSR and the INIT signals have to be experimentally determined for every flip-flop inside the CLB. These values are unique for the target FPGA and once they are known, they can be reused to test different designs in the same device since they are constant over the entire FPGA. This fact is also true for other Xilinx FPGA families such as the 7-series. The values that will change depending on the physical location of the DUT are the row, column, and word fields, and they can be easily calculated from the .ll file as explained in the previous step.

The GSR signal address can be found by knowing that, when triggered, it forces the flip-flop to output the initialization (INIT) value. For example, if we input a constant '1' in every flip-flop of the register and the initialization value is a logic '0', then we expect to see the 8 LED indicators connected to the Q outputs of the flip-flops go off when the GSR signal is triggered. Therefore, an exploratory fault injection campaign can be performed by taking $MA = 0000100 = 4$ as a starting point and testing each of the 32 bits inside this and adjacent minor frames with the SEM IP. The GSR minor and bit values will be those that turn off the 8 LED indicators simultaneously when the register input is a constant '1' and the initialization values are all '0'.

In order to find the minor and bit addresses of the INIT values, the procedure is similar to the one explained for the GSR signal. In this case, the INIT value is only released (and observed at the Q output) when the reset (or set) signal is triggered. Therefore, the main idea is to leave the register in a permanent reset state and to perform again an exhaustive fault injection campaign in each of the 32 bits inside $MA = 0000100 = 4$ and adjacent minor frames with the SEM IP. In this way, the INIT minor and bit values will be those that turn on a particular flip-flop when the default INIT value is '0'.

Once the minor and bit values are identified for these 8 flip-flops, the same process has to be repeated for the remaining 8 flip-flops of the top-half of the CLB controlled by the GSR2 signal (see Fig. 3). Then, the CLB will be completely characterized.

For reference purposes, the characterization values obtained for the Kintex-UltraScale KCU105 device are summarized in Table 2. These values are constant over the entire FPGA and are valid for any design implement in the FPGA. This means that the procedure does not have to be repeated again for other designs. It can be noticed in this table that the configuration memory bit of the INIT values is located at the next minor address from the one obtained directly from the .ll file (i.e. $MA = 4 + 1 = 5$). That makes sense since the addresses presented in the .ll file are related to the flip-flop readback nodes.

TABLE 2. Minor and bit values obtained after the characterization of a CLB for the Kintex-UltraScale device. These values are constant throughout the FPGA.

Field	Minor address	Bit address
GSR1	7	7
GSR2	6	13
INIT	.ll minor address + 1	.ll bit address

6) MERGE FIELDS

With the row, column, and word obtained from the .ll file and the minor and bit values obtained empirically, the SEM IP physical addresses can then be created by merging each field. The creation of future SEM IP physical addresses for other designs can now be automated developing a script or a simple application.

7) WRITE INJECTION ADDRESSES IN TEXT FILE

All the injection addresses for each flip-flop can be collected in a text file and sent line by line to the SEM IP through its

Algorithm 1 Pseudocode for Flip-Flop Placement

Input: The name **n** of the design under test
Output: A constraints file containing the commands to place half of the flip-flops at AQ and half at EQ

```
F_p ← (emptyfile)
OpenImplementedDesign()
all_flipflops ← GetCells(n)
l ← Length(all_flipflops)
h ← l/2
for i = 1 → l do
    if i <= h then
        file_line ← SetFlipflop(AQ)
        F_p ← Write(file_line)
    else
        file_line ← SetFlipflop(EQ)
        F_p ← Write(file_line)
    end if
end for
return F_p
```

monitor interface to inject in every design flip-flop. In this way, the two possible scenarios in a flip-flop can be emulated as follows:

- *Default value*: inject a bit flip using only the GSR address.
- *Opposite value*: inject first in the INIT address and then (without correcting the error) inject another bit flip in the GSR address.

It should be clarified that, for example, the GSR1 signal is common to the eight bottom-half flip-flops. Therefore, and in order to modify the content of one design flip-flop with each injection, the design under test has to be constrained to use at most two flip-flops per CLB (one from the bottom-half and one from the top-half). The creation of the physical constraints for the flip-flop placement can be done implementing a TCL (tool command language) script. For the sake of completeness, the pseudocode of this script is detailed in Algorithm 1.

The script opens an empty file to write the constraint commands to place half of the flip-flops at AQ and the other half at EQ (see Fig. 3). It receives as input the name of the design under test to get the flip-flop cells from the implemented design. Then, a constraint line is written for each flip-flop element to place it at AQ or EQ. This file can then be used in the Vivado project to perform the mentioned placement. It should be remarked that this fact is not a limitation since there are abundant flip-flop elements in current FPGA devices but, in large designs where the use of two flip-flops per CLB may be a limiting factor, an option is to divide it into submodules and test each of them separately with the appropriate input values. After evaluating the reliability of the design, the flip-flop constraint can be removed and the final ASIC or rad-hard FPGA design manufactured or implemented as usual. Therefore, this constraint will not affect the deployment of the circuit since it is only used in the FPGA platform to assure the reliability of the design.

The methodology explained in this section is valid to perform exhaustive fault injection campaigns, when the expected campaign runtime is reasonable, or statistical campaigns, when the design is too large to be completely covered. Besides, the experimental set-up may be configured to perform permanent or transient errors in the flip-flops depending on the user needs. For the sake of simplicity, this paper covers a detailed explanation of an experimental set-up to emulate permanent errors. However, transient errors can also be introduced with the SEM IP by using a clock management scheme such as the one presented in [25] to stop the master clock of the DUT, inject the error, and then resume the clock. In any case, the proposed methodology is independent of the campaign and the desired error to test.

As just mentioned, an experimental set-up to emulate permanent errors is presented in the next section as a practical example of the methodology described. This exercise will illustrate how to perform exhaustive fault injection campaigns in different example designs to demonstrate the control and precision achieved with the methodology.

IV. TESTING THE METHODOLOGY

In order to test the methodology, a serial-in serial-out (SISO) shift register and a 10-taps finite impulse response (FIR) filter have been implemented in a Kintex-UltraScale KCU105 FPGA to illustrate different behaviors.

The expected behavior of the shift register in the presence of errors will be merely based on its structure. This means that any injected error will always imply a visible error in the output since the introduced bit flip will be shifted to the right and eventually outputted by the design (see Fig. 7 (a)). Conversely, the behavior of the FIR filter in Fig. 7 (b) is more complex to predict since the injected error may be masked by subsequent multiply-accumulate operations depending on the flip-flop affected, the dynamic range of the input, and the coefficient values of the filter.

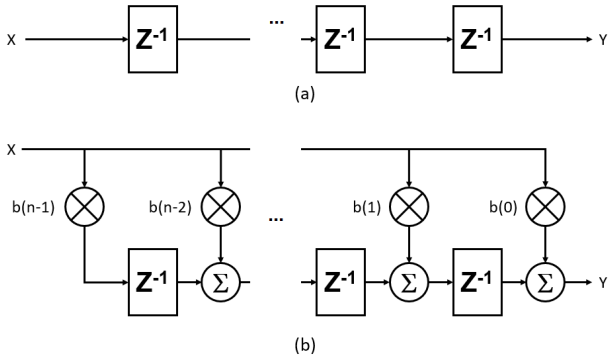


FIGURE 7. Structural comparison between the (a) SISO shift register and the (b) FIR filter.

For illustrative purposes, the reliability of these designs against permanent errors have been tested by conducting

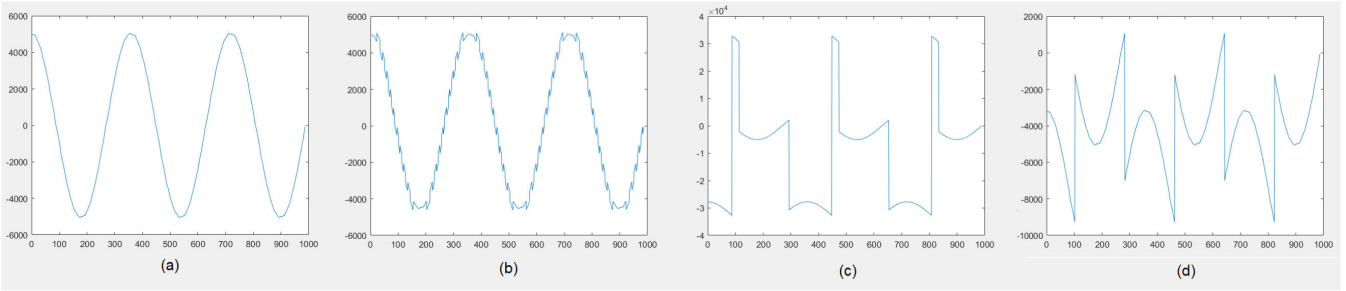


FIGURE 8. Output waveforms generated by the FIR filter during the fault injection campaign when the affected flip-flop stores (a) the most significant bit (MSB) of the first tap, (b) an intermediate bit from the sixth tap, (c) an intermediate bit from the ninth tap, and (d) the MSB of the final tap.

fault injection campaigns with the SEM IP. The flip-flop injection addresses have been previously obtained following the proposed methodology. To control the campaign, a C++ script running in a computer transmits the addresses through the serial port to the IP.

Exhaustive fault injection campaigns[1] have been conducted to test both designs. In particular, a single-error fault injection campaign has been carried out. The procedure is as follows:

1) The circuit under test is exercised in the absence of errors with a set of inputs to obtain the golden model.
2) A logic '0' is created in one of the flip-flops of the design by using the SEM IP and the injection addresses obtained from the methodology.
3) The circuit under test is exercised using the same set of inputs and the current output is compared with the previously obtained golden (error-free) output.
4) The behavior of the circuit is logged and the previously injected error is removed.
5) Steps 2 to 4 are repeated creating the opposite error scenario (a logic '1').
6) A new flip-flop is selected and the process is repeated from step 2 to 5 until all the flip-flops are tested.

Since there are two scenarios per flip-flop element (logic '0' and logic '1'), the experimental results obtained from the previous procedure have to be processed following the classification in Table 3.

TABLE 3. Flip-flop classification depending on the behavior observed during the fault injection campaign.

Logic '0' scenario	Logic '1' scenario	Flip-flop classification
output = golden	output = golden	**Non-critical**
output = golden	output $\neq$ golden	**Critical**
output $\neq$ golden	output = golden	**Critical**
output $\neq$ golden	output $\neq$ golden	**Critical**

Based on the procedure and the Table explained before, the selected designs under test can be exercised. The results obtained for the shift register and the FIR filter are summarized in Table 4 together with the amount of flip-flops used by each design.

TABLE 4. Error rate and resource usage for the SISO shift register and the FIR filter.

Design	Flip-flops	Critical flip-flops
Shift register	128	128 / 128 (100%)
FIR filter	176	167 / 176 (94.87%)

It can be observed that the error rate is exactly 100% in the shift register, meaning that all the flip-flops in the design are critical and any introduced error leads to an erroneous outcome. This behavior is expected for designs where the input values are just stored in flip-flops (and eventually outputted) because the error cannot be masked by subsequent logic or arithmetic operations.

On the other hand, the 10-taps FIR filter shows a different behavior. In these experiments, a sine wave that covers all the dynamic range is used as input to exercise the filter. It can be seen that 9 out of 176 flip-flops are not critical according to the classification shown in Table 3. This implies that some of the injected errors do not affect any of the input samples. As mentioned before, this percentage depends on several parameters such as the flip-flop affected, the dynamic range of the input used, and the coefficient values of the filter, and it is not expected to reach 100% because the FIR filter performs multiply-accumulate operations at each tap that may mask the injected bit flip.

In order to provide insight about this masking behavior, a more detailed analysis can be done by studying the output waveforms generated by the FIR filter after each injection. Fig. 8 shows some examples of these waveforms. It can be seen in Fig. 8 (a) that errors in the flip-flops from the first taps are likely to produce negligible effects in the output waveform. However, if we inject the error closer to the output tap, the effects are more visible. In Fig. 8(b), the error is injected in an intermediate bit of the sixth tap. In this case the shape of the sinewave is still present, but it has been distorted by a sawtooth artifact. Finally, in Fig. 8(c) and Fig. 8(d) the error is injected in bits from the final taps. This creates wrapping artifacts in the sinewave due to

[1]Exhaustive campaign means that bit flips are injected in all the design flip-flops testing both the logic '0' and the logic '1' scenarios.

extreme positive or negative values (see the vertical axes in figures 8(a) to (d)). This is just an example of a visual analysis that can be done with the results obtained from the fault injection campaing. The proposed methodology gives information about the flip-flop tested and the type of error introduced (logic '0' and/or logic '1') and, therefore, provides full control over the fault injection campaign to perform in-depth studies.

Finally, it should be remarked that these designs have been tested against permanent errors for illustrative purposes, but any design can be tested in a similar fashion using the methodology and set-up detailed. The methodology is independent of the circuit structure and knowing its hierarchy is not mandatory for a ''black-box'' study but, in large designs, the reliability analysis of a particular submodule may be needed. In these cases, the name of the specific module in the hierarchy should be known since it is required to find and extract from the logic location file the specific flip-flop addresses associated to that module under test. Besides, the flip-flop addresses generated are valid to perform statistical fault injection campaigns (by selecting a subset of the generated addresses) and could be used to introduce transient errors of any duration just by controlling the design clock and using the SEM IP to remove the injected error at a specific moment. These modifications do not affect the steps of the methodology since they are related to the VHDL set-up.

V. CONCLUSION

This paper presents a methodology to extract the configuration memory bit addresses associated with the flip-flop elements of a design implemented in a Xilinx SRAM-based FPGA. With these injection addresses, the ASIC error model can be mimicked by emulating transient and permanent errors in the FPGA to assess the reliability of the design. In this manner, ASIC or rad-hard FPGA designs intended to work in harsh environments can be evaluated a priori in a short amount of time and in a cost-effective way.

The proposed methodology is proved in recent FPGA technologies and has been designed to work together with Xilinx's SEM IP Core. However, it can be considered as a reference procedure for other FPGA vendors and injection tools. The proposed FPGA-based set-up can be integrated and merged with other platforms to create a richer fault injection environment in which both FPGA and ASIC error models can be tested.

REFERENCES

[1] R. C. Baumann, ''Soft errors in advanced computer systems,'' *IEEE Des. Test. Comput.*, vol. 22, no. 3, pp. 258–266, May/Jun. 2005.

[2] K. Nagatani, S. Kiribayashi, Y. Okada, K. Otake, K. Yoshida, S. Tadokoro, T. Nishimura, T. Yoshida, E. Koyanagi, M. Fukushima, and S. Kawatsuma, ''Emergency response to the nuclear accident at the Fukushima Daiichi nuclear power plants using mobile rescue robots,'' *J. Field Robot*, vol. 30, no. 1, pp. 44–63, Jan. 2013.

[3] J. Preskill, ''Quantum computing in the NISQ era and beyond,'' *Quantum*, vol. 2, p. 79, Aug. 2018, doi: 10.22331/q-2018-08-06-79.

[4] P. Maillard, M. J. Hart, P. Chang, Y. P. Chen, M. Welter, R. Le, R. Ismail, J. Barton, and E. Crabill, ''Single-event evaluation of Xilinx 16nm UltraScale+ single event mitigation IP,'' in *Proc. IEEE Radiat. Effects Data Workshop (REDW)*, Jul. 2018, pp. 1–5.

[5] A. Ullah, E. Sanchez, L. Sterpone, L. A. Cardona, and C. Ferrer, ''An FPGA-based dynamically reconfigurable platform for emulation of permanent faults in ASICs,'' *Microelectron. Rel.*, vol. 75, pp. 110–120, Aug. 2017.

[6] I. Herrera-Alzu and M. Lopez-Vallejo, ''design techniques for Xilinx Virtex FPGA configuration memory scrubbers,'' *IEEE Trans. Nucl. Sci.*, vol. 60, no. 1, pp. 376–385, Feb. 2013.

[7] *Soft Error Mitigation Controller V4.1.*, Xilinx, San Jose, CA, USA, 2018.

[8] A. Kourfali and D. Stroobandt, ''In-circuit fault tolerance for FPGAs using dynamic reconfiguration and virtual overlays,'' *Microelectron. Rel.*, vol. 102, Nov. 2019, Art. no. 113438.

[9] O. Ruano, J. A. Maestro, and P. Reviriego, ''A methodology for automatic insertion of selective TMR in digital circuits affected by SEUs,'' *IEEE Trans. Nucl. Sci.*, vol. 56, no. 4, pp. 2091–2102, Aug. 2009.

[10] J. Carreira, H. Madeira, and J. G. Silva, ''Xception: A technique for the experimental evaluation of dependability in modern computers,'' *IEEE Trans. Softw. Eng.*, vol. 24, no. 2, pp. 125–136, Feb. 1998.

[11] O. Ruano, J. A. Maestro, P. Reyes, and P. Reviriego, ''A simulation platform for the study of soft errors on signal processing circuits through software fault injection,'' in *Proc. IEEE Int. Symp. Ind. Electron.*, Jun. 2007, pp. 3316–3321.

[12] E. Jenn, J. Arlat, M. Rimen, J. Ohlsson, and J. Karlsson, ''Fault injection into VHDL models: The MEFISTO tool,'' in *Predictably Dependable Computing Systems*. Berlin, Germany: Springer, 1995, pp. 329–346.

[13] V. Sieh, O. Tschache, and F. Balbach, ''VERIFY: Evaluation of reliability using VHDL-models with embedded fault descriptions,'' in *Proc. IEEE 27th Int. Symp. Fault Tolerant Comput.*, Jun. 1997, pp. 32–36.

[14] H. Guzman-Miranda, J. N. Tombs, and M. A. Aguirre, ''FT-UNSHADES-uP: A platform for the analysis and optimal hardening of embedded systems in radiation environments,'' in *Proc. IEEE Int. Symp. Ind. Electron.*, Jun. 2008, pp. 2276–2281.

[15] M. Alderighi, F. Casini, S. D'Angelo, M. Mancini, D. M. Codinachs, S. Pastore, C. Poivey, G. R. Sechi, G. Sorrenti, and R. Weigand, ''Experimental validation of fault injection analyses by the FLIPPER tool,'' *IEEE Trans. Nucl. Sci.*, vol. 57, no. 4, pp. 2129–2134, Aug. 2010.

[16] G. Cieslewski, A. Jacobs, A. George, and A. Gordon-Ross, ''Multibit fault injection for field-programmable gate arrays with simple, portable fault injector,'' *J. Aerosp. Inf. Syst.*, vol. 11, pp. 1–13, 07 2014.

[17] 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.*, May 2015, pp. 143–150.

[18] F. Serrano, J. A. Clemente, and H. Mecha, ''A methodology to emulate single event upsets in flip-flops using FPGAs through partial reconfiguration and instrumentation,'' *IEEE Trans. Nucl. Sci.*, vol. 62, no. 4, pp. 1617–1624, Aug. 2015.

[19] C. Fibich, P. Roessler, S. Tauner, M. Matschnig, and H. Taucher, ''A FPGA-based demonstrator for safety-critical applications,'' in *Proc. Austrochip Workshop Microelectron. (Austrochip)*, Oct. 2017, pp. 35–40.

[20] Ó. Ruano, F. García-Herrero, L. A. Aranda, A. Sánchez-Macián, L. Rodriguez, and J. A. Maestro, ''Fault injection emulation for systems in FPGAs: Tools, techniques and methodology, a tutorial,'' *Sensors*, vol. 21, no. 4, p. 1392, Feb. 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/4/1392

[21] *Debugging Single Event Upsets Using Fault Injection Debugger*, Altera, San Jose, CA, USA, 2014.

[22] L. A. Aranda, A. Sanchez-Macian, and J. A. Maestro, ''ACME: A tool to improve configuration memory fault injection in SRAM-based FPGAs,'' *IEEE Access*, vol. 7, pp. 128153–128161, 2019.

[23] *UltraScale Architecture Configurable Logic Block User Guide, UG574*, Xilinx, San Jose, CA, USA, 2017.

[24] *UltraScale Architecture Configuration User Guide*, Xilinx, San Jose, CA, USA, 2020.

[25] H. Liang, X. Xu, Z. Huang, C. Jiang, Y. Lu, A. Yan, T. Ni, Y. Ouyang, and M. Yi, ''A methodology for characterization of SET propagation in SRAM-based FPGAs,'' *IEEE Trans. Nucl. Sci.*, vol. 63, no. 6, pp. 2985–2992, Dec. 2016.

LUIS ALBERTO ARANDA received the B.Sc. degree in industrial engineering and the M.Sc. 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 worked as 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, both in journals and international conferences. His research interests include reconfigurable computing for space applications, computer vision, and robotics.

OSCAR RUANO received the M.Sc. and Ph.D. degrees in computer engineering from the Universidad Antonio de Nebrija, 2005 and 2011, respectively.

He has worked as a Lecturer and a Researcher in several Spanish universities, such as Universidad Nebrija and Universidad Francisco de Vitoria. He has developed his activity in the Space field, with different research projects on fault tolerance optimization against radiation effects in microelectronic circuits. He is the author of several technical publications, both in journals and international conferences and a series of patents. Also, he has worked with different multinational companies in the IT consultancy field, as Accenture. His research interests include computer architecture, digital design, fault-tolerance, and reliability.

FRANCISCO GARCIA-HERRERO received the B.Sc. degree in telecommunication engineering from the Escuela Politecnica Superior de Gandia, Spain, in 2008, and the M.S. and Ph.D. degrees in electrical engineering from the Universitat Politècnica de València, Spain, in 2010 and 2013, respectively.

He has worked as a Lecturer and a Researcher at several universities, including the European University Miguel de Cervantes and the Universitat Politècnica de València. He is currently an Associate Professor and a Researcher with the Universidad Antonio de Nebrija. His research interests include hardware and algorithmic optimization of error-control decoders and fault-tolerance electronics in communication and storage systems.

JUAN ANTONIO MAESTRO (Senior Member, IEEE) 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.

He is currently a Full Professor in the computer architecture with the Universidad Complutense de Madrid. Previously, he directed the Electronic Design and Space Technology Research Group, Universidad Nebrija, Madrid, where he also founded the ARIES Research Center, 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 as the European Space Agency, Stanford University, University College Dublin or the Harbin Institute of Technology, among others. He is the author of numerous technical publications in journals and international conferences. His research interests include computer architecture, digital design, fault-tolerance, reliability, small satellites, and space applications.