ACME-2: Improving the Extraction of Essential Bits in Xilinx SRAM-based FPGAs

Luis Alberto Aranda, Oscar Ruano, Francisco Garcia-Herrero, and Juan Antonio Maestro.

***Abstract*—Fault emulation in field-programmable gate arrays (FPGAs) is a popular alternative to test the reliability of a design due to its low cost and high availability compared to traditional radiation-based approaches. Fault emulation is an instrumentation method based on inducing artificial bit flips on the design under test (DUT) to modify the content of its memory elements or the structure of the design itself. A major limitation of error emulation in FPGAs lies in the determination of the proper bits to test, which is directly related to the reliability results obtained and the execution times of the fault injection campaign. The Automatic Configuration Memory Error-injection (ACME) tool helps with this process in Xilinx FPGAs by obtaining the essential bits of an FPGA region where the DUT is. In this paper, we present ACME-2, an improved version of the tool where the translation from the FPGA region to design essential bits has been modified with architectural information of the FPGA. In this revision of the tool, redundant input/output routing bits that distort the reliability results are minimized. For the example designs used in the experiments, these modifications have led to an improvement in the precision of the reliability results as well as a reduction of the fault injection campaign runtime over the previous version of the tool.**

***Index Terms*—Configuration memory, essential bit, fault injection, FPGA, reliability.**

I. Introduction

In recent times, field-programmable gate arrays (FPGAs) are gradually becoming the predominant architecture to be used for the implementation of digital circuits in space applications. In particular, space missions in which a small satellite is involved tend to use FPGAs to reduce the overall cost of the spacecraft [1]. In addition to this benefit, most FPGAs also support reconfiguration after the deployment, so the functionalities of the device can be modified on the fly. Some FPGAs such as those based on static random access memories (SRAM) provide low cost, high processing speed, and can be reprogrammed an almost unlimited number of times [2]. These features make them interesting alternatives for the mentioned low-cost space applications. A major disadvantage of using SRAM-based FPGAs for space applications is that space is a harsh environment and these memories may suffer from radiation effects such as single-event upsets (SEUs). The type of error induced depends on the component affected by the incident radiation. The configuration memory of these FPGAs is made of SRAM cells, which occupy most of the area available in the device. Therefore, configuration memory upsets should be expected more frequently than user memory errors in SRAM-based FPGAs used for space missions.

In order to deal with configuration memory errors in SRAM-based FPGAs, several approaches may be followed. From shielding or radiation-hardening methods, which are usually expensive in terms of weight and cost, to low-cost redundancy-based protection techniques —e.g. dual or triple modular redundancy (DMR or TMR) schemes—, or ad-hoc techniques, which are usually difficult and time-consuming to implement. Typically, in small satellite applications where the cost is a major constraint, the redundancy-based approach is employed.

There are different ways to effectively test circuits implemented in an FPGA to verify their correct behavior before deploying the system. One possibility is to perform physical experiments at ground facilities, trying to replicate the conditions of the space environment. This is usually conducted at particle accelerators or by using laser beams to replicate the impact of particles in the device. In both cases, these facilities are usually expensive and many times present availability issues. Due to this, tests based on fault emulation are quite convenient. These experiments are based on inducing artificial bit flips using instrumentation methods. In FPGAs, this procedure can be done by flipping bits in the configuration memory [3]. In this way, the design is altered and its behavior can be assessed using different benchmarks and test cases.

To inject errors in the configuration memory, it is possible to use the Xilinx Soft Error Mitigation (SEM) IP Controller [4], FT-UNSHADES2 [5], or other works [6]–[8]. Automatic Configuration Memory Error-injection (ACME) is a tool based on a different approach that we developed for Xilinx FPGAs [9]. In contrast to the mentioned error injection tools, ACME is a tool oriented to help with the extraction of the configuration bits of the configurable logic block (CLB) tiles. Then, by using this list of bits and the SEM IP fault injection tool, any design implemented in the SRAM-based FPGA can be tested.

In our previous work, it is explained how the relationship between essential bit data (EBD) files and placement blocks (pBlocks) is used to generate this list. This relationship is based on several parameters obtained through reverse engineering which approximate a linear behavior. This approximation was demonstrated to be good enough to characterize isolated modules implemented in the FPGA. However, in complex designs where a module is part of a larger design, it was detected that the approximations lead to less accurate results and undesirable side-effects due to injections in other FPGA resources that are not part of the design under test and injections in redundant input/output routing. In this paper, we present a revision of the tool named ACME-2 in which architectural information of the FPGA is used to increase the

L. A. Aranda, O. Ruano, and F. Garcia-Herrero are with ARIES Research Center, Universidad Antonio de Nebrija, Madrid 28040, Spain (e-mail: laranda@nebrija.es; oruano@nebrija.es; fgarciahe@nebrija.es).

J. A. Maestro is with the Department of Computer Architecture and Automatics, Computer Science Faculty, Complutense University of Madrid, Madrid 28040, Spain (e-mail: jamaestro@ucm.es).

precision of the tool during the extraction of design essential bits. This represents an improvement over the previous ACME version in terms of accuracy of the reliability results and reduction of the fault injection campaign runtime.

The rest of the paper is organized as follows. Section II highlights the limitations of the first version of ACME and the main differences with ACME-2. The internal details of ACME-2 and its equations are described in Section III. In Section IV, both versions of the tool are tested with an example design to show the improvements of ACME-2 over the previous version. Finally, Section V concludes the paper.

II. Overview of ACME and ACME-2

ACME is an open-source tool presented in our previous work [9] designed to translate the configuration memory essential bits of an SRAM-based FPGA region into injection addresses for the Xilinx SEM IP Controller. ACME helps in the reliability analysis of a design under test (DUT) implemented in an SRAM-based FPGA by providing the injection addresses of those bits that have a higher probability to provoke a malfunction in the design. These bits are called "essential" by Xilinx and are grouped in the EBD file generated by the synthesis tool (e.g. Vivado). Unlike the bitstream, EBD files only contain 32 ASCII characters per line related to the CLB tiles. These characters are either zero or one. A zero represents a non-essential bit and a one represents an essential bit. The main problem of using an EBD file to perform a configuration memory fault injection is that the EBD file is just a collection of zeros and ones without any straightforward relationship with the SEM IP. ACME fills this gap by extracting and translating these characters into injection addresses that can be interpreted by the SEM IP.

In addition to the EBD file, ACME requires the slice coordinates of a rectangular shape containing the FPGA slices in which the resources of the DUT are placed. This requirement increases the precision of the tool by delimiting the range of the EBD lines that correspond to the DUT and that will be eventually translated into injection addresses by using some equations provided by Xilinx in [10, 11]. To fulfill this requirement, a pBlock that encloses the DUT resources can be drawn in the floorplan view of the FPGA during the implementation step of the hardware design flow.

The first version of ACME makes use of both the EBD file and the pBlock slice coordinates of the design to determine the injection addresses for the SEM IP. However, it was found out that the use of the slice-based coordinate system is not adequate to determine the exact range of EBD lines to translate. This coordinate system requires to estimate through reverse engineering procedures a slope and intercept values for each FPGA clock region to implement linear approximations that eventually accumulate error, leading to imprecise EBD line ranges. This means that neighboring FPGA frames are also translated into injection addresses. For example, in designs in which the DUT is placed close to other modules, this will imply injecting with the SEM IP in resources that belong to these other modules, or even undesirable injection side-effects if the design is close to the SEM IP itself. In designs where the DUT is isolated, this also implies injecting several times in the same input/output routing resources and thus producing the same effects. This phenomenon is intrinsic to FPGA devices and will occur when the DUT is deployed in a real platform. However, when the focus is on using an FPGA to test the capabilities of a protection technique, redundant injections are undesirable events that distort the final reliability results.

In order to deal with this limitation and increase the precision of the tool, we decided to go deeper into the FPGA architecture and use the grid-based tile coordinate system to upgrade ACME to ACME-2. Unlike the slice-based coordinate system, whose origin is at the bottom-left corner of the device, the origin of the grid-based coordinate system is located at the top-left corner of the FPGA. This means that the first version of ACME has to be internally modified to support a different coordinate system, but the inputs received from the user will remain the same. For clarification purposes, an overview of the integration of the ACME-2 tool in reliability analysis is depicted in Fig. 1.

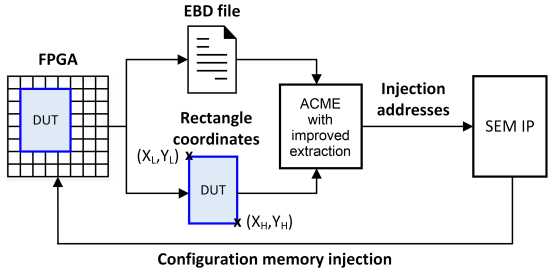

Fig. 1. Integration of the ACME-2 tool into the reliability analysis.

In essence, ACME-2 is a modified version of the previous ACME tool presented in [9] in which the input coordinate system has been changed. From the user point of view, this change does not imply any additional effort since the procedure is similar, and the grid coordinates of the rectangle containing the tiles of the DUT can be obtained by using Vivado and looking into the rectangle properties of the pBlock. Therefore the impact on the user interaction is minimal. However, from the point of view of the internal functioning of the tool, there have been several changes that are explained in detail in the following section.

III. Improving the Extraction of Essential Bits

The extraction of the EBD file lines related to the DUT is the first operation that ACME performs with the information provided by the user. The second step is to translate each essential bit present in the extracted EBD lines into an injection address compatible with the Xilinx SEM IP core. As mentioned in the previous section, the translation is done by implementing some equations published by Xilinx (see [9] for a detailed explanation of the equations). Therefore, the accuracy of the tool is related to the correctness of the EBD lines extracted in the first step.

In the first version of ACME, the linear correlation between EBD file lines and pBlock coordinates was inferred by reverse engineering procedures and then approximated by

a linear slope-intercept equation. This approximation is good enough to characterize the reliability of an isolated module implemented in the FPGA. However, there are more complex and realistic scenarios where the module of interest is part of a larger design and is placed next to other modules. In these cases, increasing the precision of the tool will be meaningful and beneficial since the quality of the results and the fault injection campaign runtime will be improved.

In order to increase the precision of the tool, it has been verified for different Xilinx FPGA families that the grid coordinates of the CLB tiles have a direct relationship with the EBD file lines. This is reasonable since each line of the EBD file represents the classification (in essential or non-essential) of the 32 bits belonging to a word of configuration RAM [10, 11]. Hence, the grid coordinate system can be used to extract the correct amount of lines in the EBD file that correspond to a specific tile or group of tiles thus avoiding any approximation.

An FPGA is divided into frames, which are a collection of CLBs from a clock region. As mentioned before, each line in an EBD file represents a word of 32 bits. Therefore, a specific number of consecutive EBD lines will represent a frame. For example, in Xilinx UltraScale devices this number is equal to 123. The difficult part here is to properly identify the starting line/word where the frame begins and, more importantly, its relationship with a tile in the FPGA. To determine this relationship, several experiments were performed placing a test design in different tiles along the FPGA. During these experiments, the behavior illustrated in Fig. 2 was observed for the Kintex-UltraScale KCU105 FPGA:

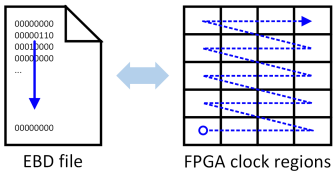

Fig. 2. Relationship between the frames of the Kintex-UltraScale KCU105 FPGA and its EBD file. Reading the file from top to bottom is similar to moving the design following the arrow in a zigzag basis.

Fig. 2 shows the relationship between the EBD file lines and the physical placement inside the FPGA. The essential bits of a design placed at the bottom-left frame correspond to the EBD lines located at the beginning of the file, while the essential bits of a design placed at the top-right frame of the UltraScale device will appear at the end of the EBD file. Once the behavior is known, the next step is to determine the total number of frames in the FPGA (F_t) as a starting point to infer a relationship between tile coordinates and EBD lines. To do that, the expression in equation (1) can be used:

$$F_t = \frac{\textit{Total EBD lines} - \textit{Dummy lines}}{W_f} \tag{1}$$

Where *Dummy lines* is the number of padding and header lines at the beginning of the EBD file (e.g. 141 for an UltraScale device and 109 for the 7-series family), and W_f the number of words per frame (e.g. $W_f = 123$ for an UltraScale device while $W_f = 101$ for the 7-series family).

As shown in Fig. 2, the frames of the KCU105 UltraScale FPGA increase when the tile coordinate in the X-axis increases. Therefore, the number of frames per row of clock regions (F_y) can also be obtained as follows:

$$F_y = \frac{F_t}{C_y} = \frac{\textit{Total EBD lines} - \textit{Dummy lines}}{W_f \cdot C_y} \tag{2}$$

Where C_y is the number of clock regions in the vertical axis. In the UltraScale example, $C_y = 5$ (see Fig. 2) so the number of frames per row of clock regions is $F_y = 5236$.

Finally, the number of frames per X tile coordinate (F_x) can be calculated by dividing F_y by the total number of CLB tile columns in the horizontal axis (C_x).

$$F_x = \frac{F_y}{C_x} = \frac{\textit{Total EBD lines} - \textit{Dummy lines}}{W_f \cdot C_x \cdot C_y} \tag{3}$$

The C_x value can be obtained by counting the number of horizontal CLB sites in the floorplanning view. In the UltraScale example, the total number of CLB tile columns in the horizontal axis is $C_x = 238$ and the number of frames per X tile coordinate is $F_x = 22$. With this information, and knowing that there is a horizontal offset of $X_0 = 50$ (the leftmost CLB site), the equations presented in (4) can be used to determine the starting line in the EBD file for both the first and last frames of the rectangle containing the DUT:

$$\begin{aligned} \textit{First frame} &= [(X_L - X_0) \cdot F_x + F_y \cdot Y_0] \cdot W_f \\ \textit{Last frame} &= [(X_H - X_0) \cdot F_x + (F_x - 1) + F_y \cdot Y_0] \cdot W_f \end{aligned} \tag{4}$$

Where X_L and X_H are the X grid coordinates of the rectangle containing the design tiles (see Fig. 1). The location of the design in the vertical axis is introduced in (4) through the vertical offset (Y_0) parameter. This value indicates the horizontal row (or rows) of clock regions where the design is. For example, it can be observed in Fig. 2 that the UltraScale has 5 horizontal rows of clock regions. In this device, $Y_0 = 0$ if the design is placed at the bottom row of clock regions, and increases by one when moving it up to the next horizontal row of clock regions. The maximum value ($Y_0 = 4$) will be reached at the top row of clock regions. If the rectangle containing the design is placed in between two or more horizontal rows of clock regions, equations (4) will have to be applied for each of them obtaining multiple pairs of first-last frame values. Each pair indicates the beginning and the end of a fragment of consecutive EBD file lines that belong to the design.

In the first version of ACME, the vertical coordinates of the pBlock are only used to determine the clock regions where the design is. Therefore, a range of words inside the frame cannot be selected, so all the 123 words (in an UltraScale device) are analyzed looking for essential bits. This means that essential bits that belong to neighbor modules may be flipped if those bits/resources are located in the same frame as the design under test. In ACME-2, this issue is fixed by implementing equation (5), where Y is the maximum value of the Y coordinate in the clock region.

$$\begin{aligned} \textit{First word} &= (Y - Y_H) \cdot 2 \\ \textit{Last word} &= (Y - Y_L) \cdot 2 + 1 \end{aligned} \tag{5}$$

During the placement experiments, it was observed in the EBD file that the starting word inside the frame decreased when the tile coordinate in the Y-axis increased. Moreover, it was also detected that each tile in the UltraScale FPGA consists of two words. Therefore, the Y coordinates of the rectangle containing the tiles of the design (Y_L, Y_H) can be used to obtain the range of words for the frames calculated with (4). Equations (5) illustrate how to calculate the first and last words of the range.

As a final remark, it should be mentioned that the procedure explained in this section for the UltraScale device can also be followed in a similar way to obtain the proper values for any 7-series or UltraScale+ FPGA. If the device has an uneven distribution of clock regions (e.g. Zynq SoCs), there will be as many pairs of equations (4) as dissimilar horizontal rows of clock regions. Each pair of equations will have its own C_x and F_x constants and will have to be applied depending on the vertical coordinates of the rectangle containing the design.

IV. Experimental Results

The improvements implemented in ACME over the previous version have been measured by testing two designs. The first design is a hardware implementation of the coordinate rotation digital computer (CORDIC) algorithm. A powerful algorithm widely used in space applications to compute the discrete cosine transform or to generate signals for heterodyne transceivers [12]. For this work, an open-source implementation of the algorithm for rectangular to polar coordinates conversion has been selected [13]. In addition to this design, the NOEL-V RISC-V processor from the Gaisler GRLIB IP Library has also been tested [14]. The reliability of the processor has been measured by executing the Dhrystone benchmark to exercise different parts of the core and comparing the final output values against an error-free scenario. The benchmark is run and its outcome is logged in a text file thanks to the Gaisler GRMON3 hardware monitor/debugger.

Both CORDIC and RISC-V designs have also been tested in a protected version by implementing a DMR scheme. In theory, any design protected with a DMR scheme can detect around 100% of the injected errors thanks to a redundant copy and a comparator (in practice this percentage is lower due to errors affecting the comparator). Therefore, it can be used to measure the accuracy of the reliability results obtained with each version of ACME.

Both unprotected and DMR-protected CORDIC and RISC-V designs have all been placed at the bottom-left and the top-right clock regions of the UltraScale FPGA device. The purpose of this is to measure the linear approximation effects in the first version of ACME under its best (bottom-left) and worst-case (top-right) scenarios, and to illustrate that these effects do not occur in ACME-2. These scenarios have been used to compare both ACME and ACME-2 versions in terms of reliability results provided and fault injection campaign runtime. To do so, the injection addresses of the designs have been extracted with both versions of the tool and then, different fault injection campaigns were conducted with the SEM IP.

In these experiments, one configuration memory bit flip at a time is performed to determine its effect on the design outcome. If the outcome mismatches the correct ("golden") output in the absence of errors then the injection is classified as "error" and, for the DMR schemes, if the comparator detects the error it is also classified as "detected". After collecting this information, the bit flip is corrected by the SEM IP to return the design to its initial state and the following address on the list is tested in the same manner. In the case of the CORDIC designs, the campaign finishes when all the injection addresses are tested, meaning that exhaustive fault injection campaigns are performed. However, in the RISC-V core, statistical fault injection campaigns are performed due to its large number of essential bits. The reliability results obtained for the CORDIC are presented first in Table I together with the campaign runtime. The results for the RISC-V processor will be described later in Table II. The values presented in both tables are in range format indicating the worst and the best-case scenarios mentioned before.

TABLE I
Comparison between ACME versions for the CORDIC designs

CORDIC unprotected	**ACME**	**ACME-2**
Total injections	166,387–177,556	44,640–46,450
- Errors	2.22–9.58%	2.54–2.67%
Runtime (hours)	6.7–7.4	1.8–2.0
CORDIC DMR-protected	**ACME**	**ACME-2**
Total injections	373,016–380,430	99,745–101,001
- Errors	2.00–17.71%	5.79–6.11%
- Detected	98.81–99.89%	99.82–99.95%
Runtime (hours)	15–15.5	3.9–4.2

The first thing that can be noticed in Table I is that ACME generates more injection addresses than ACME-2. This difference has a direct impact on the campaign runtime since more bits to flip implies more iterations. The higher amount of essential bits extracted by the first version of ACME is mainly because it cannot determine a range of words inside a frame, so the injections are performed in all the words of each frame. This means that the redundant input/output routing resources outside the pBlock are being injected. This is illustrated in Fig. 3. In this figure, the pBlock (purple box) containing the logic and routing resources of the DMR CORDIC (labeled MY_DUT) is partially depicted together with some white squares for illustrative purposes. These white squares highlight some redundant resources that are extracted by the first version of ACME, while ACME-2 only extracts resources inside the pBlock area. In other words, the bits extracted with ACME-2 are focused on the logic resources of the design plus the minimal input/output routing resources that belong to the DUT. This means that the translation process of ACME-2 is faster since the external routing, which depends on the design placement, is not evaluated.

This phenomenon can also be noticed in the percentage of errors. The range in ACME is wider than ACME-2 (e.g. 2.22–9.58% vs. 2.54–2.67%). This means that ACME-2 is more accurate than ACME. The injection addresses generated by ACME-2 are not "region dependent" and, as mentioned before, only include the design resources plus the minimal input/output resources. This is better for measuring the intrin-

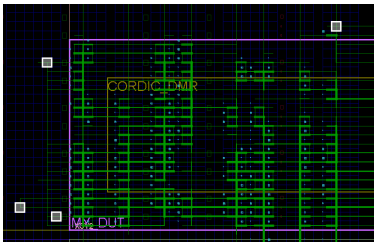

Fig. 3. Layout view with part of the pBlock (purple box) containing the logic and routing resources of the CORDIC design extracted by ACME-2. White squares around the pBlock illustrate redundant resources extracted by the first version of ACME, thus slowing the injection process.

sic reliability of a design regardless of its location inside the FPGA and also for characterizing a protection technique since injections in redundant input/output routing resources lower the final percentage of detected errors (see the DMR case).

In order to measure the precision of both tools, a larger design (i.e. a RISC-V processor) has been constrained in a way that occupies entire clock regions and has been placed filling the bottom-left clock regions following the dashed blue arrow in Fig. 2 to create the best-case scenario. The worst-case scenario is similarly created starting from the top-right clock region following the arrow in the opposite direction. In this manner, the previously mentioned phenomenon of redundant input/output routing observed in the CORDIC is minimized. In this case, the processor contains more than 17 million essential bits and exhaustive campaigns are unfeasible, so 10,000 statistical injections have been performed.

TABLE II
COMPARISON BETWEEN ACME VERSIONS FOR THE RISC-V CORE

RISC-V unprotected	**ACME**	**ACME-2**
- Errors	11.21–23.09%	23.59–25.20%
RISC-V DMR-protected	**ACME**	**ACME-2**
- Errors	16.11–40.31%	42.32–48.03%
- Detected	94.24–99.91%	98.75–99.89%
Statistical injections	10,000	10,000
Runtime (hours)	25	25

As can be seen in Table II, the best cases for ACME (23.09, 40.31, and 99.91%) are close to ACME-2 ranges. Therefore, it can be concluded that the precision of both tools is similar or slightly better for ACME-2. In the worst cases, ACME suffers from loss of accuracy due to the mentioned interpolation approximations and, therefore, the probability of affecting a design resource decreases. Finally, the runtime for all these experiments was around 25 hours, which is higher than the one for the CORDIC designs. That is because the next injection is not done until the processor has outputted the Dhrystone benchmark result.

As a final remark, it should be mentioned that in some cases the user may want to test the reliability of the whole design, meaning that the redundant input/output routing resources removed by ACME-2 should be tested. In this case, ACME-2 could still be useful to extract the bits of other modules in which the user does not want to inject (e.g. the SEM IP) and then test the rest of the bits of the whole design.

V. CONCLUSIONS

In this paper, ACME-2, an improved version of the ACME tool is presented and detailed. The ACME-2 tool allows hardware designers to obtain the configuration memory essential bits of a design, which can be used to measure its intrinsic tolerance to configuration memory soft errors. This newer version provides more accurate reliability results while, at the same time, reduces the overall fault injection campaign runtime. This improvement is particularly beneficial to enable the possibility to test large designs in which the campaign runtime is a major constraint (e.g. soft processors). In addition, ACME-2 facilitates the testing of specific submodules of a design providing high precision and avoiding injecting in resources that belong to other surrounding submodules. In this manner, both the behavior of a submodule in the presence of errors and the protection coverage provided by a fault-tolerant technique could be better studied.

REFERENCES

[1] F. Benevenuti et al., "Experimental applications on SRAM-based FPGA for the NanosatC-BR2 scientific mission," in *2019 IEEE Int. Parallel and Distributed Processing Symp. Workshops (IPDPSW)*, Rio de Janeiro, Brazil, 2019, pp. 140-146.

[2] F. Siegle, T. Vladimirova, J. Ilstad and O. Emam, "Availability analysis for satellite data processing systems based on SRAM FPGAs," *IEEE Trans. on Aerospace and Electronic Systems*, vol. 52, no. 3, 977-989, 2016.

[3] N. A. Harward et al., "Estimating soft processor soft error sensitivity through fault injection," in *2015 IEEE 23rd Annual Int. Symposium on Field-Programmable Custom Computing Machines*, Vancouver, BC, 2015, pp. 143-150.

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

[5] J. M. Mogollon et al., "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.

[6] 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.

[7] 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.

[8] U. Legat et al., "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.

[9] L. A. Aranda, A. Sánchez-Macián 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.

[10] Xilinx, "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, 2016. [Online]. Available: https://www.xilinx.com/support/answers/67337.html [Accessed: July 02, 2019].

[11] Xilinx, "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, 2016. [Online]. Available: https://www.xilinx.com/support/answers/67086.html [Accessed: July 02, 2019].

[12] L. A. Aranda, F. Garcia-Herrero, L. Esteban, A. Sánchez-Macián and J. A. Maestro, "Radiation Hardened Digital Direct Synthesizer With CORDIC for Spaceborne Applications," *IEEE Access*, vol. 8, pp. 83167-83176, 2020.

[13] R. Herveille, "CORDIC core," *OpenCores*, 2013. Available: https://opencores.org/projects/cordic [Accessed: Nov 04, 2020].

[14] "NOEL-V Processor," *Gaisler*, 2021. Available: https://www.gaisler.com/index.php/products/processors/noel-v [Accessed: May 04, 2021].