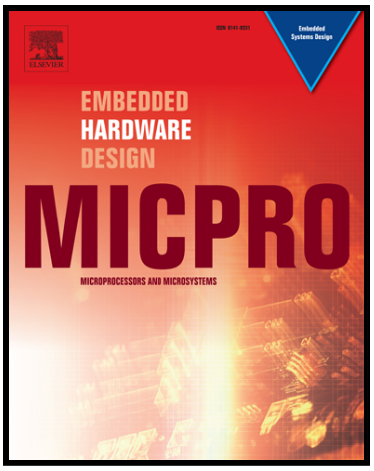


Soft Error Susceptibility Analysis Methodology of HLS Designs in SRAM-based FPGAs

Jorge Tonfat , Lucas Tambara , André Santos ,
Fernanda Lima Kastensmidt

PII: S0141-9331(17)30217-X
DOI: 10.1016/j.micpro.2017.04.016
Reference: MICPRO 2544

To appear in: *Microprocessors and Microsystems*

Received date: 5 August 2016
Revised date: 3 April 2017
Accepted date: 26 April 2017

Soft Error Susceptibility Analysis Methodology of HLS Designs in SRAM-based FPGAs

Jorge Tonfat[a,b], Lucas Tambara[a], André Santos[a], and Fernanda Lima Kastensmidt[a]

[a] Instituto de Informática – PGMICRO, Universidade Federal do Rio Grande do Sul (UFRGS), Av. Bento Gonçalves, 9500 Porto Alegre, Brazil
[b] Space Research Institute, Austrian Academy of Sciences, Schmiedlstraße 6, A-8042 Graz, Austria
jorge.tonfat@oeaw.ac.at
{latambara, afdsantos, fglima}@inf.ufrgs.br

Abstract. SRAM-based FPGAs are attractive to critical applications due to their reconfiguration capability, which allows the design to be adapted on the field under different upset rate environments. High level Synthesis (HLS) is a powerful method to explore different design architectures in FPGAs. In this paper, the HLS tool from Xilinx is used to generate different design architectures and then analyze the probability of errors in those architectures. Two different case studies scenarios are investigated. First, it is evaluated the influence of control flow and pipeline architectures combined with the use of specialized DSP blocks in the FPGA. The number of errors classified as silent data corruption and timeout according to the architectures and DSP blocks usage is analyzed. Moreover, more possibilities of HLS designs are explored such as data organization, aggressive pipeline insertion and the implementation of the algorithm in a soft processor like the Microblaze from Xilinx. These architectures are strongly optimized in performance and the least susceptible design under soft errors is investigated. All case-study designs are evaluated in a 28nm SRAM-based FPGA under fault injection. The dynamic cross section, soft error rate and mean work between failures are calculated based on the experimental results. The proposed characterization method can be used to guide designers to select better architectures concerning the susceptibility to upsets and performance efficiency.

Keywords: FPGA; Soft Error; Fault Injection; HLS

1 Introduction

Integrated circuits operating in high radiation environments are well known to be susceptible to errors due to particle ionization. However, circuits operating on Earth can experience transient faults too caused by the interaction of low and high-energy neutrons with the silicon [1]. Field Programmable Gate Arrays (FPGAs) are flexible components that can be customized and reconfigured to implement a large variety of designs. FPGAs are attractive devices for use in particles accelerators, automotive industry, aircrafts and satellites systems due to their high density and capability of integrating many designs into a single chip and still achieving high performance due to the process parallelism.

Transient ionization may occur when a single radiation-ionizing particle strikes the silicon, creating a transient voltage pulse, or a Single Event Effect (SEE). This effect can change the stored values in the sequential logic, known as Single Event Upsets (SEU), or change the logical value of a node in the combinational logic, known as Single Event Transient (SET). SRAM-based FPGAs are mainly susceptible to Single Event Upset (SEU) in their configuration memory bits and embedded memory cells (BRAMs). When energy is collected by a sensitive junction of an SRAM cell transistor that composes the configuration memory bits of the SRAM-based FPGA bitstream, a bit-flip (SEU) can occur. This bit-flip can change the configuration of a routing connection or the configuration of a LUT or the configuration of the BRAM. This bit-flip has a persistent effect, which can only be corrected when a new bitstream is loaded to the FPGA. Bit-flips can also occur in the flip-flop of the Configurable Logic Block (CLB) used to implement the user's sequential logic. In this case, the bit-flip has a transient effect and the next load of the flip-flop can correct it. Thus, the majority of the errors observed in harsh environments come from bit-flips in the configuration memory bits. Fault injection in the bitstream can be performed in the laboratory, and it is useful for analyzing the logical vulnerability of a design to bit-flips in the configuration memory.

Depending on the design’s architecture, more or less configuration bits are used and more or less susceptible bits may be responsible for provoking an error in the design output. However, not only the number of used configuration bits determines the sensitivity of a design. The masking effect of the application algorithm plays an important role and for that there are tradeoffs in the architecture such as area, performance, execution time in clock cycles and types of resources utilized that may directly contribute to the soft error rate analysis in FPGAs.

This work investigates a set of HLS optimizations to generate designs less suceptible to SEU in SRAM-based FPGAs. Two different case studies scenarios are analyzed. First, it is evaluated the influence of a control flow and pipeline architectures combined with the use of specialized DSP blocks in the FPGA. The number of errors classified as Silent Data Corruption (SDC) and timeout according to the architecture and DSP blocks usage is analyzed. Moreover, more possibilities of HLS designs are explored such as data organization, agressive pipeline insertion and the implementation of the algorithm in a soft processor like the Microblaze from Xilinx. These architectures are strongly optimized in performance and the least suceptible design under soft errors is investigated. All case-study designs are evaluated under fault injection and results are compared for different design architectures of a matrix multiplication algorithm. Although this algorithm is fairly simple, the first objective is to investigate the different directive parameters in HLS tool in simple algorithms in order to have full controllability of the generated architecture and comparison among them. Once the proposed flow is validated to simple algorithms, the work can be extended to more complex ones. Static and dynamic cross sections were estimated and also the Mean Workload Between Failures (MWBF) metric is discussed.

Previous works on HLS focus on the generated design quality in terms of area, performance or power consumption but not in analysing their susceptibility under upsets in SRAM-based FPGAs [2,3]. So, it is mandatory to investigate an approach to analyze the impact of using different architectures provided by HLS tools in the error rate of the design in order to guide designers to select better architectures concerning the susceptibility to upsets and performance efficiency. One of the main contributions of this work is a methodology that designers can follow to predict the error rate of a design synthesized by HLS tools in the early stages of the development. Moreover, authors discuss the susceptibility of different architectures under soft errors and show that with similar area and performance, it is possible to find more or less critical designs due to its masking effect capability and different resources usage. Also, this methodology can be useful to analyze the effectiveness of some mitigation techniques as the one presented in [4] based on software techniques for microprocessors without hardware overhead.

This paper is organized as follows: section 2 describes the proposed methodology to analyze the susceptibility of HLS designs. Section 3 presents the HLS design flow and the case study scenarios explored in this work. Section 4 shows the fault injection results for the selected case study designs and a detailed analysis of all the metrics used to characterize the designs under soft errors. Finally, section 5 presents the conclusions and future work.

2 Proposed Methodology

The susceptibility analysis of a design implemented in an SRAM-based FPGA under soft errors is not a straightforward task. It depends on the characteristics of the design and the susceptibility of the underlying FPGA platform. This section presents the metrics to analyze the susceptibility and the methods to obtain them.

2.1 Metrics for Estimating Susceptibility

Soft Error Rate (SER) is a metric normally adopted to characterize the susceptibility of a design under faults. However, in order to be able to calculate the SER of a design, many parameters must be taken into account. We first define each one of these parameters that we suggest using in our proposed method to calculate SER in early stages of the HLS design development. The first group of parameters includes *Area* and *Performance*. The area of an implemented design can be expressed in terms of the number of used resources such as LUTs, flip-flops, BRAM blocks, DSP blocks, etc. Also, it is possible to express the area in terms of configuration frames. A configuration frame is the smallest addressable memory segment of the configuration memory (bitstream). Since each frame is related to a specific resource and position in the floorplan [5], it is possible to calculate the number of configuration frames used by a design. The performance of a design can be expressed in terms of the execution time, operational frequency and the processed workload. The execution time can be defined by the number of clock cycles to perform the operation. According to the FPGA device and design architecture, a maximum clock frequency is achieved. A higher clock frequency is obtained by reducing the delay of the longest combinational paths. Another important parameter is the workload processed by the design. The workload is the amount of data computed in one execution. In terms of reliability, the performance information is helpful to know how much time the design is exposed to soft errors during the execution of the implemented function.

The second group of parameters includes the *Essential bits*, *Error* and *Critical bits*. The essential bits are defined by Xilinx [6], and they refer to the amount of configuration bits associated to a design mapped in a certain FPGA. Essential bits are a subset of the total configuration bits, and they depend on the area of the implemented design. The errors are defined as any deviation from the expected behavior. They can be classified as SDC and timeout errors. An SDC error is manifested as an incorrect data result of the circuit and timeout errors are manifested as an absence of a data result due to a problem in the execution of the circuit. Timeout errors can occur when the Finite State Machine (FSM) of a circuit goes to an invalid state or executes an invalid state transition. In the case of microprocessors and Digital Signal Processors (DSPs), timeout errors occur due a program crash or program exception [7]. The critical bits are also defined by Xilinx [6] as the amount of configuration bits that once flipped they cause an error in the expected design behavior (SDC or timeout). The critical bits are a subset of the essential bits.

The third group of parameters includes radiation measurements such as static and dynamic cross sections. The static cross section (σ_{static}) is an intrinsic parameter of the device usually expressed in terms of area (usually cm^2/device or cm^2/bit) and is related to the minimum susceptible area of the device to a particle species (e.g. neutron, proton, heavy ion, etc.). The expression to obtain the static cross section of a device is defined in (1) and the static cross section per bit is defined in (2):

$$\sigma_{static-device} = \frac{N_{SEU}}{\Phi_{particle}} \tag{1}$$

$$\sigma_{static-bit} = \frac{\sigma_{static-device}}{N_{bit}} \tag{2}$$

Where N_{SEU} is the number of SEU in the configuration memory bits, $\Phi_{particle}$ is the particle fluence. The fluence is measured by particle per cm^2, and it is calculated by multiplying the particle flux by the time the device has been exposed to that flux. The cross section per bit is calculated by dividing the static cross section by the total number of bits in the device (N_{bit}). In SRAM-based FPGAs, it is important the static cross section per bit of the configuration memory and the user embedded memory (BRAM).

The dynamic cross section (σ_{dynamic}) is defined as the probability that a particle generates an error in the design. The expression to obtain the dynamic cross section is:

$$\sigma_{dynamic} = \frac{N_{ERROR}}{\Phi_{particle}} \tag{3}$$

Where N_{ERROR} is the number of errors observed in the design behavior and $\Phi_{particle}$ is also the particle fluence.

The fourth group of parameters includes the metrics used based on the radiation measurements and estimations in the laboratory. They are *Soft Error Rate* (SER) and *Mean Workload Between Failures* (MWBF). SER is expressed in Failure in Time (FIT) units, and is defined as the expected amount of errors per 10^9 device operation hours in a determined radiation environment. It can be obtained multiplying the dynamic cross section with the particle flux in the radiation environment. The metric MWBF was proposed in [8], and it evaluates the amount of data (workload) processed correctly by the design before the appearance of an output error. It is defined as:

$$MWBF = \frac{w}{\sigma_{dynamic} * flux * t_{exec}} \tag{4}$$

Where w is the workload processed in one execution, σ_{dynamic} is the dynamic cross section, *flux* is the particle fluence per unit time and t_{exec} is the execution time of the design.

2.2 Xilinx Analysis Tools

By using the Xilinx Vivado design tool, we can obtain the following metrics: area in terms of resource utilization after the design is placed and routed, the performance of the design in terms of clock cycles and execution time, and also the essential bits of a particular design. The essential bits are calculated using a proprietary algorithm of the Xilinx tool after the bitstream is generated. Excluding essential bits, all the other mentioned metrics can be obtained with tools from other FPGA manufacturers.

2.3 Fault injection Method

Fault injection (FI) by emulation is a well-known method to analyze the reliability of a design implemented in an SRAM-based FPGA [9-12,23]. As mentioned before, SEUs in the configuration memory bits of SRAM-based FPGAs are the main reliability threat. Therefore, fault injectors usually emulate bit-flips (SEU) in the configuration memory bits by loading a corrupted bitstream into the FPGA. The original bitstream configured into the FPGA can be corrupted by a circuit or a computer tool by flipping one of the bitstream bits, one at a time. This flip emulates an SEU in the configuration memory cells.

The corrupted bitstream can be generated outside the FPGA or it can be corrupted inside the FPGA by using the Internal Configuration Access Port (ICAP) in the case of Xilinx FPGAs. When the corrupted bitstream is generated outside the FPGA, a custom computer program is usually used to alter the bitstream file and to program the FPGA with the corrupted bitstream. This method can inject an SEU or MBU in any bit position of the bitstream and it is not required any additional circuit on the device. The major drawback is that the time to inject the fault depends on how fast the full bitstream can be loaded from the computer. Usually the full bitstream is transfered using the JTAG interface and it can take many seconds to complete the transfer.

The configuration and readback time depends on the bitstream size, the configuration interface speed and the interface data width as shown in eq. 5.

$$\mathrm{ConfigurationTime} \approx \frac{\mathrm{BitstreamSize}}{\mathrm{InterfaceSpeed} * \mathrm{InterfaceDataWidth}} \qquad (5)$$

As an example for the Xilinx Virtex UltraScale+ FPGA family, the JTAG interface has a maximum speed of 66 MHz and a data width of one bit [13,14]. The smallest device in the family takes 3.24 seconds and the largest device takes 13.74 seconds to be configured with a JTAG interface. On the other hand in this FPGA family, the ICAP interface has a maximum speed of 200 MHz and a data width of 32 bits. In this case, the smallest device takes 0.03 seconds and the largest device takes 0.14 seconds. The trend is that newer devices will have larger bitstream sizes and the interface's speed will not keep the same increasing rate. As a result, the configuration time will increase.

On Xilinx devices, fault injectors can take advantage of the dynamic partial reconfiguration capabilities to reduce the time to inject bit-flips. Reconfiguring only a portion of the bitstream reduces the configuration time as shown in eq. 5. Following the example of the Xilinx Virtex UltraScale+ FPGA, reconfiguring only one configuration frame (the smallest portion of the bitstream) takes approximately 45 microseconds using the JTAG interface and 0.4 microseconds using the ICAP for all the devices sizes in that family.

After the FPGA is loaded with the bitstream, it is possible to modify only a portion of the configuration memory bits, so the corrupted bitstream can be generated inside the FPGA by a controlling logic and the reconfiguration can be done by the ICAP. Examples found on the literature are FLIPPER [9], FT-UNSHADES [10], FIRED [11] and the fault injector developed in our group [23]. In [9], the main objective is to predict radiation test experiments using a fault injector that mimics the effects of SEUs by injecting in random positions bit-flips and accumulating them until the appearance of an error. In [10] they present a similar approach where the bit-flips are injected at some point of the DUT execution time. However, before injecting the bit-flip, the DUT is stopped and then resumed. In [11], the authors present a fault injector used to find the interaction of cumulative faults also emulating the effects of a radiation test experiment. In our work, the objective is to obtain the number of critical bits to predict the result of radiation test experiments. All of these works uses the dynamic partial reconfiguration feature of Xilinx FPGAs to inject the bit-flips. This feature is used by fault injectors to speed up the fault injection campaign. This method is faster than a full reconfiguration of the device because only a small portion of the configuration memory needs to be modified to insert the bit-flip and the communication with a computer is limited to only download the fault injection data results and to receive some control signals.

On both methods explained before, the output of the design under test (DUT) can be constantly monitored to analyze the effect of the injected fault into the design. If the data output of the DUT is known, a comparison with a fault-free version of the output is used to detect an error. Another approach is to implement a “golden” design (that is a copy of the DUT) that receives the same input stimulus as the DUT and the result of both designs are compared as shown in [12].

If an error is detected, this means that this configuration memory bit is a critical bit. It is possible to inject faults in all the configuration bits and obtain the critical bits of the design. The entire fault injection campaign can spend from few hours to days depending on the amount of bits that are going to be flipped and the connection to the fault injection control circuit. The fault injection method provides us the error analysis and classification, and also the critical bits. A detailed analysis of fault injection platforms can be found in [15].

2.4 Accelerated Radiation Method

The accelerated radiation method consists in exposing the device in front of a particle accelerator beam and monitor the design behavior. This method is employed to obtain the device sensitivity to a determined range of particles and also to obtain a soft error rate of a circuit. In the case of FPGAs, the test can be static or dynamic. The basic procedure for the static test consists on programming the FPGA with a known (golden) bitstream, to irradiate the device and continuously readback the bitstream from the FPGA, to compare with the golden bitstream and to count the number of bit-flips. This process is generally controlled by an external device connected to the FPGA. The readback time depends on the same parameters as the configuration time (eq. 5). The bitstream size and the configuration interface speed and data width determine the smallest readback interval possible. However, the read-back interval (the time between two readback procedures) also depends on the particle flux and Linear Energy Transfer (LET). The LET is the quantity of energy that a particle can transfer to the silicon. The LET and flux define the SEU bit-flip rate. As an example, an Artix-7 FPGA exposed to carbon-12 with a flux of 182 particles/(cm^2•s) and LET of 3.26 MeV•cm^2/mg generates a bit-flip rate of 2.89 bit-flips/s. These data was obtained during a heavy ion test performed at the Laboratório Aberto de Física Nuclear of the Universidade de São Paulo (LAFN-USP), Brazil in March 2016 [16]. In this scenario, the readback interval should be keep as small as possible to avoid accumulating a large amount of bit-flips in the same readback process. With fewer bit-flips per readback, it is possible to analyze the sensitivity to MBUs with a higher confidence as shown in [17].

From the static test, it is possible to calculate the static cross section of the FPGA configuration memory. The static cross section is usually given per bit. In case the target FPGA has already been tested for a particular particle, the static cross section can be found in papers or in datasheets. The static cross section per bit for many Xilinx FPGAs can be obtained from the Xilinx Reliability Report [18]. In this work, the static cross section of each circuit is estimated using the static cross section per bit from the Xilinx Reliability Report under neutron and the number of configuration bits (frames) the design uses.

The dynamic test analyzes the design output mapped into the FPGA, and from this test is obtained the dynamic cross section and soft error rate. In this case, the expected soft error rate is much lower than the obtained from the static test. In this work, we calculate the dynamic cross section by a method proposed in [19]. The details are shown in next section.

3 Exploring High Level Synthesis tools

High Level Synthesis (HLS) tools are suitable to explore different design architectures to implement a desired algorithm. These tools offer the possibility to generate custom hardware accelerators for intensive data computing tasks such as DSP algorithms, off-loading these tasks from embedded processors. One of the benefits achieved by these tools is the reduced developing time that allows the rapid exploration of different design architectures.

The starting point of HLS tools are algorithms described in a high-level software language (e.g. C, C++, SystemC). HLS tools transform these high-level descriptions into hardware cycle-accurate descriptions using Hardware Description Languages (HDLs) (e.g. VHDL or Verilog).

A high-level synthesis extracts the control and data flow graphs from the source code to implement the hardware design using defaults and user-applied directives. Such directives guide the high-level synthesis process in specific objectives such as execution performance, area usage, or power consumption.

It is possible to find several HLS tool vendors and from the academia. Some of them are open-source such as the LegUp tool [20] from University of Toronto. In this work, the Xilinx Vivado HLS tool [21] is used as it is well integrated into the Xilinx design flow and offers support for different Xilinx FPGAs.

The Vivado HLS tool requires a validated algorithm source code and a set of constraints and directives to generate an RTL synthesizable netlist. The design flow is presented in Fig. 2. The synthesis directives are used to guide the generation of the synthesizable HDL description targeting area and performance. The algorithm source code does not require major modifications. The guiding directives are applied to operations, conditional statements, loops and functions of the source code. The directives used in this work are: the *pipeline* directive that allows the concurrent execution of operations within a loop or function reducing the computation latency; the *array partition* directive that can split large arrays into many small arrays to improve the data access removing the memory bottleneck; the *inline* directive that removes the hierarchy of a function to allow logic optimization across the function boundaries; the *unroll* directive which allows to unroll of loops to generate multiple instances of the operations within the loop [21]. Moreover, another feature of Vivado HLS is the possibility to create hybrid designs with portions of code running on a soft or hard-core processor communicating with custom hardware accelerators generated by HLS.

In order to evaluate the effects of using High Level Synthesis (HLS) in the design for SRAM-based FPGAs in the susceptibility to soft errors, we use the method proposed in Section 2. The idea is to have a straightforward method to predict the susceptibility and in early design phase being able to select the designs that will present the highest reliability and efficiency when synthesized into SRAM-based FPGAs.

The evaluated algorithm is the standard matrix multiplication. The algorithm is shown in Fig. 1.

```
void matrixmul(

      mat_a_t a[MAT_A_ROWS][MAT_A_COLS],

      mat_b_t b[MAT_B_ROWS][MAT_B_COLS],

      result_t res[MAT_A_ROWS][MAT_B_COLS])

{ // Iterate over the rows of the A matrix (Outer Loop)

   Row: for(int i = 0; i < MAT_A_ROWS; i++) {

      // Iterate over the columns of the B matrix (Middle Loop)

      Col: for(int j = 0; j < MAT_B_COLS; j++) {

         res[i][j] = 0;

         // Do the inner product of a row of A and col of B (Inner Loop)

         Product: for(int k = 0; k < MAT_B_ROWS; k++) {

            res[i][j] += a[i][k] * b[k][j]; }}}}
```

Fig. 1. Standard Matrix multiplication algorithm.

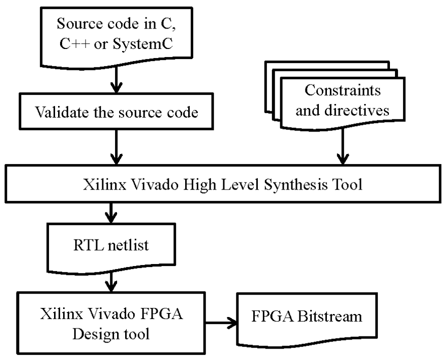

Source code in C, C++ or SystemC
Validate the source code
Constraints and directives
Xilinx Vivado High Level Synthesis Tool
RTL netlist
Xilinx Vivado FPGA Design tool
FPGA Bitstream

Fig. 2. FPGA Design flow using Xilinx High Level Synthesis (HLS) tool.

3.1 Case study scenario 1: Exploring control flow versus pipeline architecture with and without DSP blocks

In this first analysis, the matrix multiplication algorithm is implemented using 8-bit integer 6x6 input matrixes generating a 6x6 16-bits matrix output. Two architectural versions were generated using the Xilinx HLS tool from the algorithm C source code. The first version of the matrix multiplication design is generated by using no optimization directives in the tool. So the generated circuit implements the datapath logic for the inner product loop (Fig. 1) and a FSM to control the data flow. The second version of the matrix multiplication design uses the pipeline directive into the outer loop, so the three iteration loops are unrolled, and many instances of the hardware inner product are generated. The generated circuit implements a pipeline version of the inner product loop hardware (Fig. 1) and it presents a pipeline of 37 stages depth. Each architectural version was implemented using two different synthesis implementation directives in the FPGA. One uses the DSP48E resources to implement the adders and multipliers of the FPGA and the other not, consequently the adders and multipliers are implemented by Lookup Tables (LUTs). The four generated designs are labeled: *No opt., No DSP48E*; *No opt., with DSP48E*; *Pipeline opt., No DSP48E*; and *Pipeline opt., with DSP48E*.

The inputs matrixes A and B are stored in two embedded memories (BRAM) and the output matrix C is stored in another BRAM for all the architectures. Each element of the input matrixes is fixed to the value 0x55 that resembles a checkerboard pattern. So, the output matrix is also fixed for all the executions. The checkerboard pattern allow us to exercise most of the datapath of the matrix multiplication circuit, avoiding a fault masking due to the input pattern.

3.2 Case study scenario 2: Evaluating the data organization, pipeline and unroll features of HLS Designs

In this second analysis, the matrix multiplication algorithm is evaluated using the pipeline directive from the Xilinx HLS tool but applied at different levels of the matrix multiplication algorithm. Also, the unroll feature and a different data organization of the input matrixes are applied to the algorithm. Additionally, it is included a version of the algorithm implemented in a soft processor from Xilinx, the Microblaze (MB). It is a 32-bit RISC processor with a 5-stage pipeline. For all the design architectures, the input and output matrixes are 32x32 floating-point half precision (16-bits) stored in BRAM memories.

The first HLS design is similar to the *No opt. with DSP48E* version presented in the first analysis. None optimization is applied so it is labeled as *HLS - no opt*. Then, the pipeline directive is applied to the three different loops of the matrix multiplication generating three different designs. The labels of these versions are: *HLS – pipeline IL* (inner loop), *HLS – pipeline ML* (middle loop), *HLS- pipeline OL* (outer loop). In addition, the unroll directive was applied to the inner loop generating the design labeled *HLS – unroll IL* (inner loop). Finally, the array partition and inline directives were applied together with the pipeline directive in the middle loop generating the version labeled *HLS – pipeline ML+AP* (array partition).

All these HLS generated versions target the improvement of the execution time of the matrix multiplication algorithm. Each of these designs has different architecture, area and execution time; however all of them process the same input matrixes.

4 Experimental Results

The designs were synthesized into SRAM-based FPGA Artix-7 XC7A100TCSG324 from the Xilinx 7-series family. The configuration frame size varies among FPGA families; and in the case of the Xilinx Artix-7 FPGA, each frame has 101 words of 32-bits [5].

Table 1 and Table 2 show the area results and the performance achieved in both analyses. The area results are expressed in terms of FPGA resources (absolute values and in percentage of total available resources in the device) and configuration frames and bits used by the design. Performance is presented in terms of execution time measured in clock cycles and the processed workload. The workload is the amount of data computed by the design in one execution and is given in bits.

In case study scenario 1, the workload is two matrixes of 36 elements each and each element is an 8-bit data that gives us 36x8x2 bits of workload per execution. The versions with the pipeline optimization use more resources and have a better execution time than the versions without optimization since the HLS tool unrolls and pipeline the loops, generating more hardware instances and incrementing the parallelism. The purpose of DSP blocks in our case studies is to implement the additions and multiplications of the matrix multiplication algorithm instead of implementing these operations in general purpose LUT logic. So, it is possible to see that the use of DSP resources reduces the number of LUTs, as expected.

Table 1. Case study 1 area details and performance results.

	Area				Performance	
Matrix Mult. Version	**# LUTs**	**# FFs**	**# DSP**	**#Configuration bits**	**Exec. time (clk cycles)**	**Workload (bits)**
No opt., No DSP48E	155 0.24%	70 0.06%	0 0%	694,224 2.99%	733	576

No opt., with DSP48E	50 0.08%	38 0.03%	1 0.42%	784,216 3.38%	733	576
Pipeline opt., No DSP48E	18,910 29.83%	5,735 4.52%	0 0%	6,479,424 27.91%	36	576
Pipeline opt., with DSP48E	1,117 1.76%	1,327 1.05%	198 82.50%	7,559,328 32.56%	36	576

In case study scenario 2, the workload is two matrixes of 1024 elements each and each element is a 16-bit data. In this analysis the DSP blocks are enabled by default so each design version uses the number of DSP blocks that the synthesized design requires.

The Microblaze version has one of the smallest areas, but it presents the highest execution time. The used resources shown in Table 2 are for the standard configuration of the Microblaze. It is a 32-bit 5-stage pipeline Reduced Instruction Set Computer (RISC) soft processor. The matrix multiplication is implemented in software following the algorithm show in Fig. 1. Among the pure pipeline versions, the version with pipeline in the outer loop (*HLS – pipeline OL*) has the highest number of resources used. This design version is similar to the *Pipeline opt., with DSP48E* in the case study 1and it pipelines and unrolls all the three loops of the matrix multiplication algorithm. Finally, the fastest design version but also the one with more area is the design version *HLS - pipe-line ML+AP* where is combined the pipeline in the middle loop and an input partitioning of the matrix to increase the data throughput.

Table 2. Case study 2 area details and performance results.

	Area	Performance

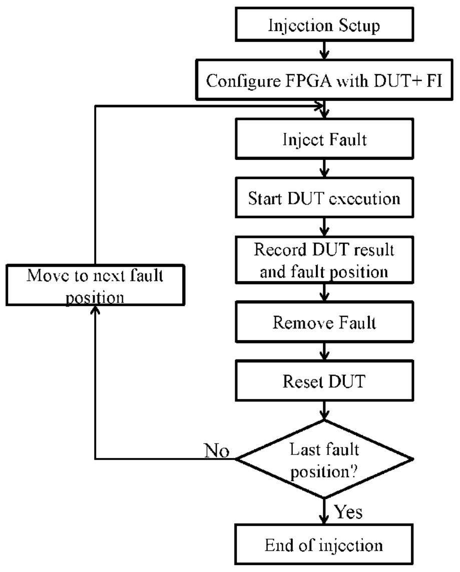
Matrix Mult. Version	# LUTs	# FFs	# DSP	#Configuration bits	Exec. time (clk cycles)	Workload (bits)
Microblaze	1,159	1,452	0	462,816	41,611,263	32,768
	1.83%	1.15%	0%	1.99%		
HLS – no opt.	755	944	5	435,879	366,354	32,768
	1.19%	0.74%	2.08%	1.88%		
HLS - pipe-line IL	899	1078	5	435,871	134,886	32,768
	1.42%	0.85%	2.08%	1.88%		
HLS - pipe-line ML	2,397	3,374	10	1,000,132	19,613	32,768
	3.78%	2.66%	4.17%	4.31%		
HLS - pipe-line OL	9,320	6,021	10	2,602,129	17,473	32,768
	14.70%	4.75%	4.17%	11.21%		
HLS - pipe-line ML+AP	11,787	13,924	160	4,606,244	4,982	32,768
	18.59%	10.98%	66.67%	19.84%		
HLS – unroll IL	985	1,092	5	435,867	173,127	32,768
	1.55%	0.86%	2.08%	1.88%		

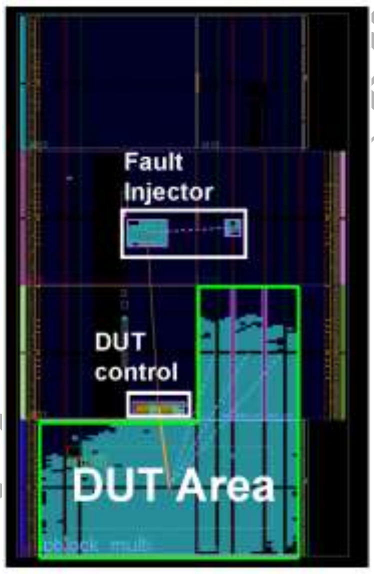
4.1 Obtaining critical bits from fault injection

The fault injection platform used in this work injects bit-flips in the configuration memory bits of the FPGA using the dynamic partial reconfiguration feature and the ICAP, so the corrupted bitstream is generated inside the FPGA. The fault injection platform is composed of a computer running a fault injection campaign script, a fault injection core that controls the Internal Configuration Access Port (ICAP) to read and write frames of the FPGA bitstream, and a DUT control unit. The DUT control unit handles the execution of the DUT and also it is responsible for the comparison of the DUT result with the correct (golden) result. This platform was developed in our research group and it has been used in many related works as [23] and [24].

The critical bits of each design version are obtained by an exhaustive fault injection in all the configuration bits of the injection area (DUT area) one at a time. The fault injection campaign is defined by the flow diagram of Fig. 3a. The first step is to setup the injection campaign that includes the injection area and the type of fault injection campaign. One bit-flip is injected per design execution. The next step in the flow diagram is to configure the FPGA with the DUT, the fault injector (FI) core and the DUT control unit. Then, the first fault is injected, and it is done before the execution of the DUT. Afterwards, the DUT starts executing and when the DUT execution is finished, the DUT result is analyzed. If a mismatch occurs between the DUT output and the golden execution data, the injected bit is classified as a critical bit and it is reported to the computer to be saved in the report file. The error can be either a SDC or a timeout. The DUT result analysis and fault position are saved. Finally, the fault is removed and the DUT is taken to its initial good known condition, prepared for the next fault injection. The fault removal process is explained with details later on the text. The fault injection process is done until all the configuration bits of the DUT area are evaluated. The required time to complete a fault injection campaign depends on the DUT area (number of configuration frames) and the time to inject and remove a fault. The time to inject a fault is constant because for each bit-flip injected, the fault injector needs to read the configuration frame where the bit-flip is going to be injected, then store the frame temporally in a BRAM block to modify the desired bit position and finally write the frame back to the configuration memory. As presented in eq. 5, the write and readback time from the configuration memory is constrained by the configuration data the data bandwidth of the used configuration port. So, in this case, reading and writing one frame (3,232 bits) using the ICAP port (32-bits @ 100 MHz) takes approximately 4 microseconds. In the final implementation of the fault injector, in order to read and write one single frame, it is necessary to read and write two frames because the ICAP architecture has an internal frame buffer. In most of Xilinx FPGA families, the ICAP port has the same characteristics and the frame size vary roughly in the range between 1,000 and 3,000 bits.

So, the injection time with our approach is around the value calculated for this device (Artix-7).





(b)

(a)

Fig. 3. The fault injection methodology in (a) and the FPGA floorplan in (b).[24]

We have two approaches to remove a fault and take the design to a good condition state. The first one is to insert again a fault in the same bit, restoring its original value, and then the DUT is reset and the function is executed again to verify that the fault was removed successfully. This is the fastest approach, as it will have the same delay as the bit-flip injection. However, if the generated output of the DUT is still incorrect, a second approach is used. The monitor PC reconfigures the whole FPGA to ensure that all configuration bits are restored. This process can take several seconds to complete as mentioned before. The latter approach is the main reason why some of our fault injection campaigns last more than one day. This approach can be improved by using dynamic partial reconfiguration only on the DUT. Instead of loading the full bitstream from the monitor PC, it is possible to load only the portion that corresponds to the DUT, in this case the matrix multiplication design. To achieve this, the DUT should be implemented in the FPGA as a dynamic reconfigurable module (DRM) and the partial bitstream can be generated using the Xilinx proposed design flow [22].

In the monitor PC, it is defined the injection area, as shown on the floorplan in Fig. 3b. The DUT area is the injection area where the matrix multiplication designs are placed, and this area remains constant for the evaluated designs in each case study. In this area are only injected the configuration bits related to CLBs (LUTs, user FFs and interconnection) and clock distribution interconnection. Since the input interface of the matrix multiplication designs can be different as the used in this work, the BRAM configuration bits are not corrupted by fault injection. For the versions, which include DSP resources (DSP48E), the correspondent DSP configuration bits were added to the injection area. The fault injector is placed in a different area of the FPGA to avoid fault injections that can disrupt its functionality. It is also shown a DUT control block that is also outside the injection area. This block analyzes the correctness of the output matrix. An execution error is defined when the matrix multiplication results stored in the memory differ from the golden results or when the application does not finish in the expected execution time. So, errors are classified as SDC or timeout errors. The SDC can affect a single data element in the result matrix, or it can affect a set of them. The timeout is detected using a watchdog timer implemented in the DUT control unit that sets a maximum time to receive the DUT result. In this work the maximum time was set to three times of the execution time.

In Table 3 and Table 4, the critical bits and the essential bits are compared for both case study scenarios. From fault injection results, it can be observed that there is a small fraction (less than 18%) of the essential bits that are critical bits.

The proposed method presented by Velazco R, et. al. [19] was used to estimate the neutron dynamic cross section. In this method, the dynamic cross section is calculated by multiplying the static cross section by masking upset effect probability as shown in eq. 6. The masking upset effect probability can be considered by using the information of essential bits or by the critical bits achieved by fault injection.

$$\sigma_{dynamic-est} = \sigma_{static} * \frac{\#bits_{essential} \vee \#bits_{critical}}{bits_{injected}} \qquad (6)$$

Table 3. Case study 1 Xilinx essential bits and critical bits comparison.

Matrix Mult. Version	**Essential bits from Xilinx Tool (% of config. bits in design area)**	**Critical bits from FI (% of Essential bits)**
No opt., No DSP48E	34,862 (5.02 %)	2,434 (6.98 %)
No opt., with DSP48E	18,208 (2.32 %)	1,740 (9.56 %)
Pipeline opt., No DSP48E	3,494,735 (53.94 %)	170,929 (4.89 %)
Pipeline opt., with DSP48E	834,967 (11.05 %)	66,758 (8.00 %)

In case study scenario 1, the version implemented with the pipeline optimization and without using the DSP resources have more essential bits and critical bits than the other designs since it uses more resources than the other versions.

In case study scenario 2, it is observed that the Microblaze version has more critical bits than the HLS designs that have similar area like the *HLS – no opt.* and the *HLS - pipeline IL* versions. This is an interesting characteristic that shows the role of the architecture vulnerability factor (AVF) in the susceptibility of a design. The design with more critical bits is the *HLS - pipeline OL*, even though the design with more essential bits is the *HLS - pipeline ML+AP.*

Table 4. Case study 2 Xilinx essential bits and critical bits comparison.

Matrix Mult. Version	Essential bits from Xilinx Tool (% of config. bits in design area)	Critical bits from FI (% of Essential bits)
Microblaze	273,645 (59.13 %)	48,085 (17.57 %)
HLS – no opt.	200,653 (46.03 %)	11,733 (5.85 %)
HLS - pipeline IL	226,690 (52.01 %)	9,818 (4.33 %)
HLS - pipeline ML	601,085 (60.10 %)	47,164 (7.85 %)
HLS - pipeline OL	1,683,011 (64.68 %)	204,866 (12.17 %)
HLS - pipeline ML+AP	2,874,704 (62.41 %)	94,692 (3.29 %)
HLS – unroll IL	240,459 (55.17 %)	17,633 (7.33 %)

4.2 Obtaining dynamic cross section, SER and MWBF from essential bits and critical bits

According to our proposed methodology, we should analyze the estimated dynamic cross section, SER and MWBF based on essential bits and critical bits. The results from both case study scenarios are presented in Table 5 and 6. The SER is calculated using the average neutron flux at sea level, i.e. 13 n/(cm^2×h) [25].

Table 5. Case study scenario 1 estimated dynamic cross section, SER and MWBF based on Xilinx essential bits and critical bits.

	From Xilinx report	From essential bits			From critical bits		
Matrix Mult. Version	σ_{static} (cm^2)	$\sigma_{dynamic}$ (cm^2)	SER @ NYC (FIT)	MWBF (bits)	$\sigma_{dynamic}$ (cm^2)	SER @ NYC (FIT)	MWBF (bits)
No opt., No	4.6E-08	2.4E-10	3.17	8.9E+19	1.701E-11	0.22	1.3E+21

DSP48E							
No opt. with DSP48E	4.9E-08	1.3E-10	1.65	1.7E+20	1.216E-11	0.16	1.8E+21
Pipeline opt., No DSP48E	4.6E-08	2.4E-08	317.57	1.8E+19	1.195E-09	15.53	3.7E+20
Pipeline opt. with DSP48E	5.4E-08	5.8E-09	75.87	7.6E+19	4.666E-10	6.07	9.5E+20

Table 6. Case study scenario 2 estimated dynamic cross section, SER and MWBF based on Xilinx essential bits and critical bits.

	From Xilinx report	**From essential bits**			**From critical bits**		
Matrix Mult. Version	**σ_{static} (cm^2)**	**$\sigma_{dynamic}$ (cm^2)**	**SER @ NYC (FIT)**	**MWBF (bits)**	**$\sigma_{dynamic}$ (cm^2)**	**SER @ NYC (FIT)**	**MWBF (bits)**
Microblaze	3.2E-09	1.9E-09	24.87	1.1E+16	3.4E-10	4.37	6.5E+16
HLS – no opt.	3.0E-09	1.4E-09	18.23	1.8E+18	8.2E-11	1.07	3.0E+19
HLS - pipeline IL	3.0E-09	1.6E-09	20.60	4.2E+18	6.9E-11	0.89	9.8E+19
HLS - pipeline ML	7.0E-09	4.2E-09	54.62	1.1E+19	3.3E-10	4.29	1.4E+20
HLS - pipeline OL	1.8E-08	1.2E-08	152.94	4.4E+18	1.4E-09	18.62	3.6E+19
HLS - pipeline ML+AP	3.2E-08	2.0E-08	261.22	9.1E+18	6.6E-10	8.60	2.8E+20
HLS – unroll IL	3.0E-09	1.7E-09	21.85	3.1E+18	1.2E-10	1.60	4.3E+19

The use of essential bits to estimate the susceptibility of a design in terms of dynamic cross section or MWBF gives a good approximation comparing those metrics calculated by using fault injection critical bits.

For the case study 1, we would like to point out that by only analyzing the dynamic cross section one can say: by using the DSP block the dynamic cross section can reduce from 0.4x to 0.7x according to the architecture, and that by using pipeline optimization the cross section can increase from 40x to 70x according to the use or not of DSP resources.

However, the MWBF gives us the susceptibility analysis taking into account not only the area bits but also the performance (execution time and work-load) of the application. When the trade-off of the performance is consid-ered, one can see that the differences between designs are drastically re-duced. The best design in terms of MWBF is the version without optimization and using DSP as expected because it presents the lowest number of essen-tial bits and critical bits. Although this design version has the highest MWBF, the pipeline version using DSP may also be in some cases a good solution as well because it presents a MWBF reduction in only half while reducing the absolute execution time value in 20x, which can be interesting at system level. The work presented in [26] shows also a comparison of MWBF be-tween designs with different architectures, however they show that not al-ways the design with the lowest dynamic cross section (or lowest number of critical bits) is best option in terms of MWBF.

For the case study 2 results we can consider the Microblaze option as the worst in terms of MWBF. The high number of critical bits and the lowest per-formance are the reasons of its low MWBF. On the other side, among the HLS versions the *HLS - pipe-line ML+AP* has the best MWBF obtained from critical bits. This is an example where the performance improvement is high-er than the increase in the number of critical bits by increasing the area and can give the best results.

4.3 Classifying the errors obtained in the fault injection campaign

Although the use of essential bits can early determine the trend behavior of a design under soft errors, the fault injection is necessary to classify the errors and to help designers to analyze the efficiency of fault tolerant techniques once they are applied in the design. Errors are classified as SDC and timeout. The SDC can affect a single data word in the result matrix, or it can affect a set of them.

For the case study scenario 1, Fig. 4. presents the error classification of the four designs obtained by fault injection. SDC are classified in three categories. The set of SDC errors spans from one erroneous data value of the output matrix to all erroneous data values (36) of the output matrix. One can see that the non-optimized versions have a different error pattern when compared to the pipeline optimized versions. The non-optimized versions have more percentage of errors related to the control flow of the algorithm because the control finite state machine in those architecture versions occupies a larger area of the architecture, which leads to a high number of timeouts errors and errors in all data values of the output matrix. The pipeline versions have more errors related to the data flow, which leads to a high number of errors in single or few data values of the output matrix. These results are mainly driven by the architecture of the design, which in the case of the pipeline optimization explores more the parallelism of the algorithm.

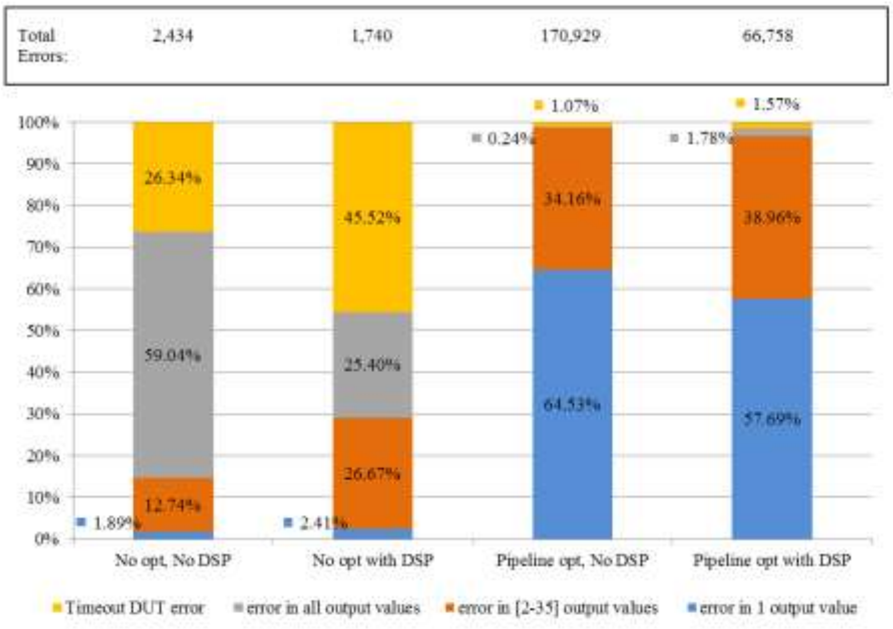


Fig. 4. Error classification for the case study scenario 1 design versions.

For the case study scenario 2, Fig.5 presents the error classification of the Microblaze and HLS design versions. The *Microblaze* version clearly presents errors related more with control flow leading to timeout or all output values errors. This is in agreement with other works found on the literature [26]. On the other side, the *HLS – pipeline OL* version presents more data flow related errors. The reason is that this design is the most parallelized version with less control flow of data.

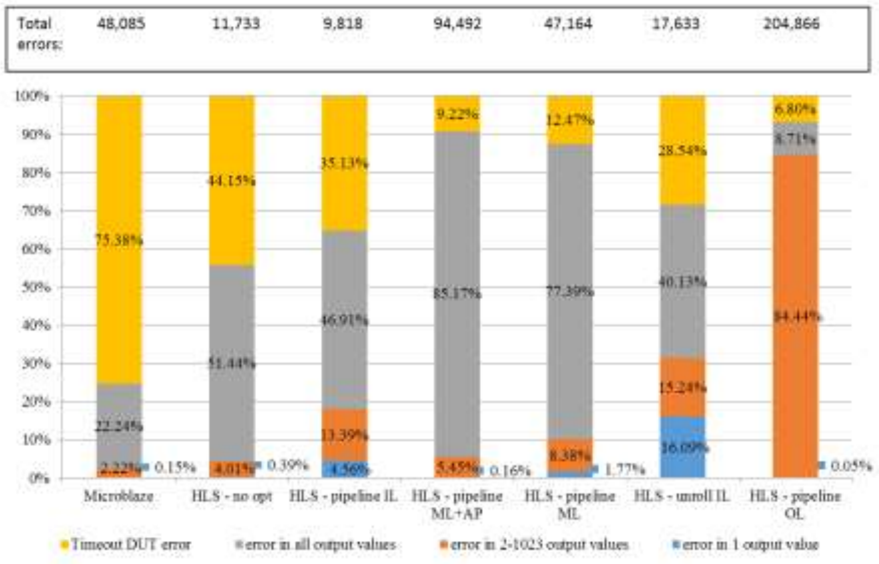


Fig. 5. Error classification for the case study scenario 2 design versions.

5 Conclusions

This work has presented a method to analyze the soft error susceptibility of designs generated by HLS and investigated different architectures under soft errors. One of the main contributions is a methodology that designers can follow to predict the error rate of a design synthesized by HLS tools in the early stages of the development. Moreover, authors discuss the susceptibility of different architectures under soft errors and show that with similar area and performance, it is possible to find more or less critical designs due to its masking effect capability and different resources usage. In a first case study scenario, it is compared control-flow style designs against data-flow style designs. In a second case study scenario, it is evaluated different types of data-flow styles designs and also a soft-processor approach. The method takes into account not only the susceptible area details of the design but also the efficiency of the design architecture to process data. A comparison between essential bits and critical bits obtained by fault injection is done. Even that metrics obtained from essential bits are a good approximation to estimate the soft error rate of a design; the fault injection method gives us a better analysis of the errors. Hence, it is shown that the architecture has a key influence on the error type. It was also demonstrated that it is possible to have a design that obtains a better performance with a low dynamic cross section that leads to a higher MWBF.

As future works, we intent to extend the work to more complex algorithms and to compare the predictive metrics to the ones calculated at radiation-based fault injection. Also, this methodology can be extended to analyze the effectiveness of some fault masking techniques that reduces the number of critical bits such as Triple Modular Redundancy (TMR).

References

1. F. Wang, V.D. Agrawal, Single Event Upset: An Embedded Tutorial, in: 21st Int. Conf. VLSI Des. (VLSID 2008), IEEE, 2008: pp. 429–434.
2. S. Skalicky, C. Wood, M. Lukowiak, M. Ryan, High level synthesis: Where are we? A case study on matrix multiplication, in: 2013 Int. Conf. Reconfigurable Comput. FPGAs, IEEE, 2013: pp. 1–7.
3. S. Windh, X. Ma, R.J. Halstead, P. Budhkar, Z. Luna, O. Hussaini, W.A. Najjar, High-Level Language Tools for Reconfigurable Computing, Proc. IEEE. 103 (2015) 390–408.
4. Gawkowski, P., Rutkowski, T., Sosnowski, J.: Improving fault handling software techniques. In: 2010 IEEE 16th International On-Line Testing Symposium. pp. 197–199 (2010).
5. Xilinx Inc. 7 Series FPGAs Configuration - User Guide, UG470 (v1.10), 2015.
6. Xilinx Inc. Soft Error Mitigation Using Prioritized Essential Bits, XAPP538 (v1.0), 2012.
7. Quinn, H., Fairbanks, T., Tripp, J.L., Manuzzato, A.: The reliability of software algorithms and software-based mitigation techniques in digital signal processors. In: IEEE Radiation Effects Data Workshop (2013).
8. P. Rech, L.L. Pilla, P.O.A. Navaux, L. Carro, Impact of GPUs Parallelism Management on Safety-Critical and HPC Applications Reliability, in: 2014 44th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Networks, IEEE, 2014: pp. 455–466.
9. M. Alderighi, F. Casini, M. Citterio, S. D'Angelo, M. Mancini, S. Pastore, G.R. Sechi, G. Sorrenti, Using FLIPPER to predict irradiation results for VIRTEX 2 devices, in: Proc. Eur. Conf. Radiat. Its Eff. Components Syst. RADECS, 2008: pp. 300–305.

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

11. Nunes, J.L., Cunha, J.C., Zenha-Rela, M.: On the Effects of Cumulative SEUs in FPGA-Based Systems. In: 2016 12th European Dependable Computing Conference (EDCC). pp. 89–96 (2016).

12. E. Johnson, M. Caffrey, P. Graham, N. Rollins, and M. Wirthlin, "Accelerator validation of an FPGA SEU simulator," IEEE Trans. Nucl. Sci., 50 (2003) 2147–2157.

13. Xilinx Inc. Virtex UltraScale+ FPGA Data Sheet: DC and AC Switching Characteristics. DS923 (v1.0), 2016.

14. Xilinx Inc. UltraScale Architecture Configuration – User Guide, UG570 (v1.7), 2017.

15. H. Quinn, M. Wirthlin, "Validation Techniques for Fault Emulation of SRAM-based FPGAs," IEEE Trans Nucl Sci 62 (2015) 1487–1500.

16. Aguiar, V. a. P., Added, N., Medina, N.H., Macchione, E.L. a., Tabacniks, M.H., Aguirre, F.R., Silveira, M. a. G., Santos, R.B.B., Seixas, L.E.: Experimental setup for Single Event Effects at the São Paulo 8UD Pelletron Accelerator. Nucl. Instruments Methods Phys. Res. Sect. B Beam Interact. with Mater. Atoms. 332, 397–400 (2014).

17. Wirthlin, M., Lee, D., Swift, G., Quinn, H.: A Method and Case Study on Identifying Physically Adjacent Multiple-Cell Upsets Using 28-nm, Interleaved and SECDED-Protected Arrays. IEEE Trans. Nucl. Sci. 61, 3080–3087 (2014).

18. Xilinx Inc. Device Reliability Report, UG116 (v9.4), 2015.

19. R. Velazco, G. Foucard, P. Peronnard, Combining Results of Accelerated Radiation Tests and Fault Injections to Predict the Error Rate of an Application Implemented in SRAM-Based FPGAs, IEEE Trans. Nucl. Sci. 57 (2010) 3500–3505.

20. A. Canis, J. Choi, M. Aldham, V. Zhang, A. Kammoona, J.H. Anderson, S. Brown, T. Czajkowski, LegUp: high-level synthesis for FPGA-based processor/accelerator systems, in: Proc. 19th ACM/SIGDA Int. Symp. F. Program. Gate Arrays - FPGA '11, ACM Press, New York, New York, USA, 2011: p. 33.

21. Xilinx Inc. Vivado Design Suite - User Guide - High-Level Synthesis. UG902 (v2014.3) Oct. 1, 2014.

22. D. Dye. Partial Reconfiguration of Xilinx FPGAs Using ISE Design Suite. WP374 (v1.2) 2012.

23. J. Tarrillo, J. Tonfat, L. Tambara, F.L. Kastensmidt, R. Reis, Multiple fault injection platform for SRAM-based FPGA based on ground-level radiation experiments, in: 2015 16th Latin-American Test Symp., IEEE, 2015: pp. 1–6.

24. J. Tonfat, L. Tambara, A. Santos, F. Kastensmidt, Method to Analyze the Susceptibility of HLS Designs in SRAM-Based FPGAs Under Soft Errors, in: 2016: pp. 132–143.

25. JEDEC (2006). Measurement and Reporting of Alpha Particle and Terrestrial Cosmic Ray-Induced Soft Errors in Semiconductor Devices JEDEC Standard [Online]. Available at: http://www.jedec.org/sites/default/files/docs/jesd89a.pdf.

26. L.A. Tambara, P. Rech, E. Chielle, F.L. Kastensmidt, Analyzing the Failure Impact of Using Hard- and Soft-Cores in All Programmable SoC under Neutron-Induced Upsets, in: 2015 15th Eur. Conf. Radiat. Its Eff. Components Syst., IEEE, 2015: pp. 1–5.