A generic methodology to compute design sensitivity to SEU in SRAM-Based FPGA

A Generic Methodology to Compute Design Sensitivity to SEU in SRAM-based FPGA

Mahsa Mousavi
and Hamid Reza Pourshaghaghi
Technical University of Eindhoven
Email: m.mousavi, h.r.pourshaghaghi@tue.nl

Mohammad Tahghighi
Hong Kong University of
Science and Technology
Email: mtahghighi@ust.hk

Roel Jordans
and Henk Corporaal
Technical University of Eindhoven
Email: r.jordans, h.corporaal@tue.nl

***Abstract*—Recently, SRAM-based FPGAs are widely used in aeronautic and space systems. As the adverse effects of radiations in space are much higher than in the Earth, developing fault tolerant techniques play crucial roles for the use of electronics in space. However, fault tolerance techniques might introduce additional penalties in area, power, performance and design time. In order to compromise between the overhead introduced by these techniques and system fault tolerance, a generic methodology for calculating design sensitivity to Single-Event Upset (SEU) is proposed in this paper. Separate schema and test-bench for evaluating effects of SEU in various types of FPGA memory are proposed in which both the raw device error rate and the vulnerability characteristic of the specific application mapped on the device are taken into account. Experimental results show that using our model in order to selectively add Triple Modular Redundancy (TMR) improve the design robustness only 18% less than full TMR while roughly introduces 69% less redundancy compared to full TMR for Fast Fourier Transform (FFT).**

***Index Terms*—Fault tolerance, FPGA, Single-Event Upset (SEU), Raw device error rate, Design vulnerability, Simulation-based injection**

I. Introduction

The interest to use Commercial Off-The-Shelf (COTS) Static-RAM Field Programmable Gate Arrays (FPGAs) in aeronautics and space missions have been significantly increased because of the features offered by them. During past years, the Europian Space Agency (ESA) has strongly stressed the need for new generation of high-performance general purpose digital signal processing platforms for future space missions based on COTS [1],[2]. Operation capacity, performance and relatively low power consumption combined with reconfigurable properties make FPGAs ideal processing devices to be used in satellites and space systems. However, COTS SRAM-based FPGAs are not manufactured for operation in space [3] and therefore the real challenge is how to develop a radiation-resistant solution to ensure the error-free function of COTS in deep-space.

Charged particles striking the silicon substrate in space may cause an error in FPGAs and make them unreliable processing platform. Single-event upsets (SEUs) are a major cause of error in contemporary FPGAs, among other radiation-induced faults [4]. An SEU is a change of memory cell state caused by one single ionizing particle e.g. ions, electrons, photons, etc, striking a sensitive node in a microelectronic device. Because this type of the fault does not reflect a permanent error of a device, it is termed as soft or transient error. Device error rate due to single event upsets is determined by the device circuit and environment characteristics. With shrinking the size of transistors, the probability of SEU in a memory cell leads to an error will increase as smaller deposit charge is capable to induce error in the transistors. On the other hand, as the size of memory cell decreases, the number of high energetic particles strike one cell decreases. The combined effect is that the error rate in one memory cell usually remains roughly constant or it even decreases. However, the error rate for an entire device exponentially increases for new generations as the total number of bits in a device exponentially increases according to the Moore's Law. Thus, fault detection isolation and recovery (FDIR) techniques are highly required for new generation devices.

A variety of fault tolerance techniques are already developed and exist, from special radiation-hardened circuit designs (e.g., [5]) to reliability-oriented place and route algorithm (RoRA) and to architectural redundancy (e.g., [6]-[8]), also scrubbing of CM (e.g., [9]-[12]). However, many of these approaches introduce significant penalties for performance, power consumption, logic utilization, and/or design time. Thus, the benefits of applying fault-tolerant techniques versus their extra cost must be carefully weighed by FPGA designers. A good estimation of design sensitivity to soft errors provides circuit designers with some key criteria to balance between fault tolerance features added to the design and overhead costs. The currently existing techniques and previously presented fault models for estimating FPGA design sensitivity to SEU still lack a comprehensive, scalable and clear technique.
This paper, therefore, presents a generic model for estimating of FPGA design sensitivity based on the fact that not all the faults affect or change final output results of a particular design. The main contributions of this paper, among others, include:

- proposing a generic model for estimating FPGA design sensitivity
- incorporating both raw device probability and design architecture in our model
- contemplating of SEU effects on all different types of FPGA memories i.e. configuration memory (CM), BRAMs, and flip-flops in our model

- proposing two different test benches for evaluating SEU effects on most important FPGA memory types
- experimentally evaluating the proposed model with a common signal processing application in digital systems i.e. Fast Fourier Transform (FFT)

The rest of paper is organized as follows: Section II gives a brief overview of the related work and fault-tolerant techniques that have been published. The proposed methodology to calculate design sensitivity in FPGA designs is described in section III. In section IV, test bench implementation for simulation-based fault injection is explained and educational examples are summarized in Section V. Finally, section VI conclude the paper.

II. RELATED WORK

A lot of efforts has been already spent looking into soft-error effects in SRAM-Based FPGAs: simulation-based, radiation-based, modeling and analysis-based approaches [13]-[24]. The first two approaches mainly exploit Fault-Injection strategies where the process of fault injection is achieved either using simulation tools or radiation equipment.

In radiation-based methods [13]-[14], a prototype model of a system under test is exposed to a flux of radiations generated either by radioactive sources or particle accelerators. The prototype FPGA under test then is continuously stimulated by a given set of input stimuli and the system outputs are compared with expected output values. Radiation-based methods are very expensive and capable of damaging the design under the test permanently. They are mainly used for device characterization, not for evaluation of the vulnerability of a particular mapped design.

The analysis-based methods [15],[16] make use of the analytical approach to evaluate SEU effects. These methods only require synthesis tool and software program and remove the necessity of any physical implementations. In these methods, first the probability of error in different design nodes is computed by modeling the error rate based on fault-behavior in different FPGA elements such as LUTs, multiplexers, programmable interconnect point (PIP) and etc. Next, the error propagation probability is computed using gate-level net-list. The vulnerability of a particular design is calculated afterward based on obtained error probability in nodes and probability of error propagation. The major drawback in these methods is in the accuracy of calculations, as only the model of some particular error behaviors is taken into account. In addition, these techniques are not easily applicable for various FPGA types as detailed information and implementation characteristic of FPGA devices are also required.

In simulation-based methods [17]-[24], SEU effects are estimated with flipping bits of CM while running a design on FPGA.

M. Wirthlin *et al.* [17] assessed the reliability of the FPGA circuit in the presence of configuration upsets. They developed an SEU simulator in order to artificially inject upsets into the CM and measure their impact on different FPGA designs. The simulator is based on the SLAAC-1V computing board consisting of three Virtex V1000 devices. Implementation results on two FPGAs are compared, one generates the golden results and the other produce results in presence of upsets.

TABLE I: Memory distribution in kintex7, Virtex V1000 and a typical design mapped on kintex7

	CM	BRAM	FF
Kintex7	81%	17%	0.5%
Virtex V1000	97%	2.2%	0.4%
Typical design	36%	62%	2%

L. Sterpone *et al.* [22] exploit from partial reconfiguration to develop a system for injecting faults into CM. A host computer controls and collects SEU simulated results and loads a list of SEU to be injected into the on-chip memory. Then hardwired Power-PC microprocessor reads the SEU list through PLB bus and executes the partial reconfiguration fault injection.

M. Violante *et al.* [23] proposed a method for calculating the probability (is named circuit cross section in the paper) of system failure due to SEU in CM by combining the physical error rate which only depends on the FPGA device and the probability of errors alter the correct output of the circuit. The former is calculated by radiation-based injection and latter is calculated by simulation-based error injection.

Pual Graham *et al.* [19] investigate SEU effects in different FPGA configurable components. In a particular type of FPGA, they determine how much percentage of a total design failure is because of SEUs. The configurable components are categorized into five main type including mux select line, programmable interconnect point, buffer enables, LUT and control bit. It is been shown that most of the observed failures are due to routing structures.

In spite of radiation and analysis based methods, which consider all sensible design elements to SEU, the simulation-based papers only assess effects of SEU on CM. The reason is that in most of FPGAs CM contributes in a large number of total memories (i.e. 97%). However, this contribution of CM decreases in new generations of FPGAs to 80%. Table I compares the contribution of different memory elements for an older FPGA i.e. Virtex V1000 and a new one i.e. Kintex 7. Moreover, this contribution rate decreases even more when only actual memory usage for a design is taken into account. For example, from Xilinx report [25] summarized in table I, for a typical design only 36% of the memory bits belong to configuration bits.

One other major drawback of previous simulation-based techniques is that any obtained error rate calculation results cannot be referred for comparison purposes between SEU sensitivity of two different designs because SEU effect is only considered for CM and not for the entire design. Therefore, for any designs that consume more FFs or BRAMs, considerable parts of the design sensible to SEU are not accounted in SEU sensivity. For example, the results in [17] show that the LFSR (linear feedback shift register) sensitivity to configuration SEU is less than that of the multiplier design. This is because LSFR mainly consists of flip-flops and uses less combination logic and routing in comparison with the multiplier. In order to

compare these two design in a fair manner, flip-flop sensitivity to SEU is also required to be evaluated.

III. METHODOLOGY

Figure 1 shows the process of how a failure is generated in a design. When high energetic particles hit the design, it may lead to SEU in memory elements of design, shown as error in fig. 1. In this situation, *failure* only happens if this error propagates through the given design and becomes visible at the final output of design. The key is that not all high energetic particles struck a design leads to SEU as well as not all SEUs lead to failure. Raw device error probability (ϵ_{dev}) indicates the probability of an SEU occurring in a device, which is perceived to be unconstrained from the circuit running on that device. The multiplication of ϵ_{dev} and percentage of device resources utilized by a given design will indicate raw design error probability (ϵ_{des}). The Design Vulnerability Factor (DVF) is defined as the probability of an happened error in a design resulting in a visible error in the final output which depends only on design architecture.

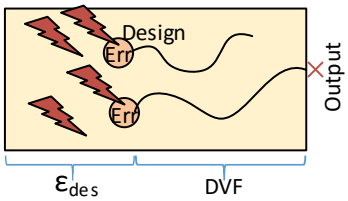


Fig. 1: Process of failure generation

The probability of failure for a design implemented in FPGA is defined as Design Sensitivity (DS), which is dependent on both device characterization and design architecture. Since not all SEUs lead to a failure in the final output of a given design, ϵ_{des} is not sufficient to estimate the sensitivity of the design to errors. Thus, considering only ϵ_{des} to determine design sensitivity will be an excessively high estimate and probably introduce the risk of increasing design costs by adding unnecessary fault-tolerant features. Thus DS is defined based on ϵ_{des} and DVF as follows:

$$DS = \epsilon_{des} * \ DVF, \ 0 \leq DS \leq 1 \tag{1}$$

Since a design composes of various resources, in order to calculate DS in (1), the multiplication of partial ϵ_{des} and partial DVF are obtained for each type of the resources in a design, separately. The probability that the resources with the same type in the design become erroneous is defined as partial ϵ_{des} and partial DVF is the probability that the error in that type of resources leads to visible failure in the output. Since SEU only affects the state of a memory, the partial multiplications are only calculated for various types of memory in FPGA which are the configuration memory (CM), block RAMs (BRAM), distributed RAMs (DiRAM) and flip-flops (FF). CM is the largest amounts of FPGA total memory bits and stores the functionality of a design. Block RAMs, distributed RAMs, and flip-flops contain the state of a design. Since the probability of occurring simultaneous error in more than one memory element caused by a high energy particle is so small and negligible, DS could therefore be derived by summation of multiplication of partial ϵ_{des} and partial DVF for all types of design resources affected by the fault. Considering the partial calculation, (1) for design L can be rewritten as:

$$\begin{aligned} DS_L = & Pr(Err\ in\ CM_L)Pr(failure|Err\ in\ CM_L) \\ & + Pr(Err\ in\ BRAM_L)Pr(failure|Err\ in\ BRAM_L) \\ & + Pr(Err\ in\ DiRAM_L)Pr(failure|Err\ in\ DiRAM_L) \\ & + Pr(Err\ in\ FF_L)Pr(failure|Err\ in\ FF_L) \\ & ,0 \leq DS_L \leq 1 \end{aligned} \tag{2}$$

where $Pr(A)$ is the probability of A happening and $Pr(A|B)$ is the probability of A happening given B has already occurred. For instance, in (2), $Pr(ErrinCM_L)$ represents partial ϵ_{des} and is interpreted as the probability of fault occurring in CM that belongs to design L. Likewise $Pr(failure|ErrinCM_L)$ represents partial DVF corresponding to CM and is defined as the probability of failure in the output of design, given that fault has been occurred in the CM of design L. In (2), partial ϵ_{des} as well as partial DVF need to be computed separately for various types of FPGA memory. Note that partial ϵ_{des} is depending on the cross-section of the memory which varies for each memory types. Similarly, DVF corresponding to different memory types need to calculated separately because faults are modeled differently and their impacts on design functionality are also different when they happen in each of dissimilar memory types. For example, SEUs in CM is modeled as permanent faults, considering there exists no mechanism for correcting them when they are faulty. However, the SEUs in FFs are modeled as transient faults since new healthy data might be written to those FFs in next clock cycles. Besides, faults in CM change the functionality of a circuit while faults in FFs only impact on the state of a design. In the following subsections, we discuss partial ϵ_{des} and DVF calculation methods for each of different memory types.

A. Raw Design Error Rate

When high energy particles such as galactic cosmic rays, cosmic solar particles and trapped protons in radiation belt hit silicon devices, electric charge is deposited which can produce a wide range of effects. These effects are expressed by Linear Energy Transfer (LET), in $MeV * cm^2/mg$ referring to that amount of energy transferred along the particle path per one material volumetric mass density. LET depends on radiations amount and the material transferred. These effects can damage an electronic device temporarily or permanently. Permanent effects like single-event latch-up (SEL) or single-event gate rupture (SEGR) are beyond the scope of this paper. Permanent faults also require mitigation schemes like shielding, but may not be always unavoidable. Instead, temporary effects can be fixed and the functionality of electronic circuits can be recovered by applying fault-tolerance techniques. Examples of temporal faults are single-event transient (SET), single-event upset (SEU) and multiple-event upset (MEU).

TABLE II: the cross-section (cm^2/bit) of various memory types for LET=20 $MeV * cm^2/mg$ [28] and [27]

Model	CM	BRAM	FF
28nm Kintex7	$4.12e-9$	$7.94e-9$	$5.0e-9$
Zynq-7000	$1.40e-9$	$2.14e-9$	$2e-9$

TABLE III: ϵ_{des} for typical design implemented in the 28-nm Kintex-7 FPGA based on cross-section presented in table II

Model	CM	BRAM	FF
Used bits	7Mb	15Mb	0.5Mb
ϵ_{des}	2.88e-2	1.19e-1	2.5e-3

A SET is a temporary voltage spike accrued in combinational logic of the FPGA. SET does not result in operation failure of circuit running on the device except when it is captured with flip-flops. An SEU is a single bit flip in the memory elements of the circuit including flip-flops, latches and SRAM cells which potentialy affects the correct function of a circuit. An MBU is the state change of two or more adjacent memory bits induced by a single particle strike. From [26], MBU rate in FPGAs is negligible. Configurations of circuitry in FPGAs are stored in SRAM cells and that makes FPGAs more vulnerable to SEU than ASIC chips.

Memory cross-section (σ) is the probability of an upset occurring per particle fluence which is obtained from the ratio of the number of observed SEUs by the particle fluence (particle per cm^2) received by the component under test during the test period:

$$\sigma = (number\ of\ SEUs)/fluence \tag{3}$$

In order to calculate σ whole FPGA device is irradiated to a specified fluence and after stopping the beam, the value of memory cells are compared with predefined values. For each specific memory type, the number of SEUs is divided by total fluence and then is normalized by the total number of bits to obtained the σ per bit. σ is related to particle LET as well as memory bit characterization such as memory type, implementation technology, and memory layout. That being defined, cross sections are different for each particular memory types i.e. CM, block RAM and flip-flop and they are also different for various FPGA models. Table II shows the cross-section of various memory types for Kintex-7 and Zynq-7000 Xilinx FPGA series used from [28] and [27] experimental results, respectively.

The ϵ_{dev} associated with memory type M is calculated as follows:

$$\epsilon_{dev_M} = (\sum_{total} b_M) * \sigma_M \tag{4}$$

where $\sum_{total} b_M$ is the total number of bit in memory type M and σ_M is the cross-section per bit of that memory type.

Similarly and for memory type M, $Pr(Err\ in\ M)$ or equally ϵ_{des_M} for a specific design L is calculated as follows:

$$\epsilon_{des_M} = (\sum_{L} b_M) * \sigma_M \tag{5}$$

where $\sum_L b_M$ is the total number of bits in memory type M that contribute in design L. Table III shows design raw error of a typical design which uses $7Mb$ configuration bits, $0.5Mb$ flip-flops and $15Mb$ block RAMs. The resources utilization numbers are obtained from Xilinx report [25] for a typical design implemented in the 28-nm Kintex-7 FPGA.

B. CM-Wise Design Vulnerability Factor

As already mentioned, in SRAM-based FPGAs design circuitry is stored in CM which is a static random-access memory (SRAM). SRAM memories are vulnerable and very sensible to SEUs and an upset in CM bits can affect or change the functionality of a design e.g. a look-up table (LUT), or routing between FPGA nodes. Figure 2 illustrates one example of such failures is a test design. Figure 2a shows a simple circuit consisting of a 2-input LUT (yellow square) and a routing switch (white square) within an FPGA. The LUT configuration bits were set to the values which implement an AND gate and inputs are connected to net A and B using a routing switch. The small yellow squares in routing switch represent configuration bits. The configuration bits of paths (N2, E1),(S2, E2) are active and connect the nets A and B to the inputs of LUT. Two scenarios are described in fig. 2b and fig. 2c in which SEU changes the circuit and causes a failure in the output result. In fig. 2b an SEU changes one bit of LUT from zero to one, exchanging AND gate functionality with an XNOR gate. In fig. 2b SEU in the routing switch configuration bit leads the path (N2, E1) to be disconnected. Consequently, the LUT input is disconnected from net A.

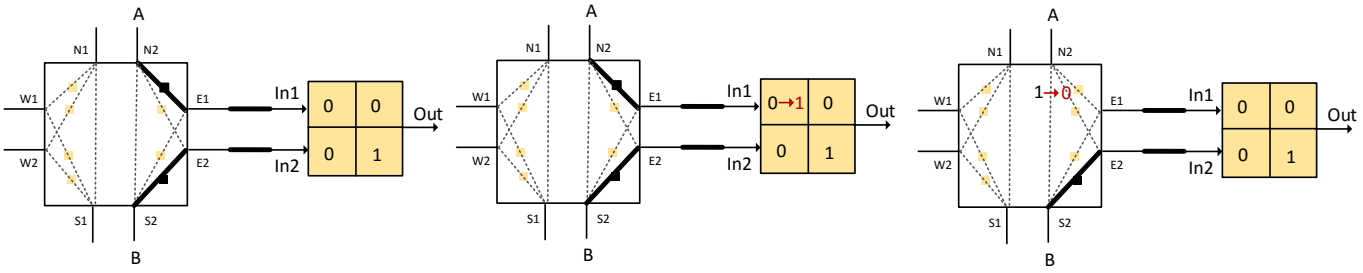
Essential bits are fractions of CM bits that determine the circuitry of a given design in FPGAs. If an essential bit is upset it follows that the design circuitry is changed, however the upset might not have an effect on the design functionality. Those subset of essential bits that result in a failure in design if they change are defined as critical bits. In 2, the probability of failure happening in design while an error occurs in CM $Pr(failure|Err\ in\ CM_L)$ is defined as CM-wise Design Vulnerability Factor (CMDVF) and is calculated as follows:

$$CMDVF = (critical\ bits\ number)/(essential\ bits\ number) \tag{6}$$

the essential bits are identified by Xilinx essential bits technology. However, the list of critical bits is required to be generated by validating the correct design behavior while moving an upset through all the essential bits in the design. The SEUs are modeled with bit-flip in the CM. Since the content of the CM will not change after FPGA are programmed, an upset remains until a correction mechanism modify it or FPGA is reprogrammed again.

C. Flip Flop-Wise Design Vulnerability Factor

In FPGAs, design states are stored in flip-flops which can also be influenced by SEUs. Same as configuration memories, not all the errors in flip-flops lead to a failure. For example, if a flip-flop is upset before its faulty bit of information is updated with a healthy value for being used towards the final



(a) A sample configuration (b) SEU in LUT (c) SEU in routing switch

Fig. 2: SEU impacts on various FPGA configurable components

output of circuit, in this case, the flip-flop upset does not have any impacts on circuit functionality.

Bit-flip is used to model SEU in flip-flops. This is a similar model used for CM with only one difference that flip-flop state could also be changed during execution of the circuit. This implies that injecting faults into flip-flops last only for one clock cycle. Furthermore, the fault may produce different effects if injected in different clock cycles. If F is the number of flip-flops in a circuit and C is the number of clock cycles of the design latency, the complete set of single errors is therefore FxC. The subset of these errors that result in a failure in design output is defined as effective errors. In 2, the $Pr(failure|Err\ in\ FF_L)$ is defined as flip-flop-wise Design Vulnerability Factor (FFDVF) and is calculated as follows:

$$FFDVF = (effective\ errors\ number)/(F * C) \quad (7)$$

The effective number of errors are computed by comparing output response of both 1) error-free and 2) erroneous circuit while we allow the circuit running in presence of errors. A faultless behavior of a design is evaluated by injecting errors in flip-flops. Errors are injected in all flip-flops, and for each of them at any desired clock cycles. Assuming fault is injected, if outputs of the circuit is different from expected outputs, then the error is counted as an effective error.

The method presented in this section is also used for fault modeling of BRAMs and distributed RAMs as they react to errors similar to flip-flop. BRAM and distributed RAM design vulnerability factors $Pr(Rfailure|Err\ in\ BRAM_L)$ and $Pr(Rfailure|Err\ in\ BRAM_L)$ are therefore called BDVF and DiDVF, respectively.

IV. IMPLEMENTATION

In this section, the schemes and test bench implementations for calculating partial vulnerability factor corresponding to each type of memory are presented.

A. CM Test Bench Implementation

We implemented a test bench system in Xilinx ZYNQ-7000 FPGA to quantify critical bits. Figure 3 depicts the block diagram of our test bench system. The test bench controller is executed on embedded ARM processor of ZYNQ to perform the evaluation process in the algorithm 1. The Design Under Test (DUT) represents the target design implemented in the

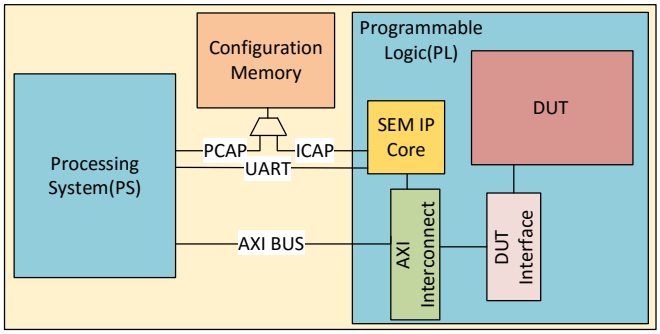


Fig. 3: Block diagram of the CM test bench system

programmable logic (PL). The test bench controller provides the inputs and valid signal for DUT or reads its generated output via AXI bus and DUT interface. The Xilinx Soft Error Mitigation IP core (SEM IP) implemented in PL is used to inject faults into the FPGA CM. The SEM IP Controller is an automatically configured, pre-verified solution to detect, correct and inject soft errors in CM of Xilinx FPGAs. The SEM IP can access and modify CM through ICAP interface and partial reconfiguration. The test bench controller communicates with SEM IP by sending commands through UART interface.

Fault injection is applied to all essential bits of design one by one in test bench. A list of essential bits is generated by the Xilinx Vivado Design Suite tool for the entire circuit implemented in FPGA, including the SEM IP. In order to determine the essential bits of DUT, Vivado tool is forced to map the design to a specific physical location of FPGA die. Then the list of design essential bits is extracted from the range of CM addresses corresponding to the design physical location. The test bench controller flow executed for each of the essential bit of design is explained in algorithm 1. In this test bench controller, first, the DUT is executed to obtain the golden output for a provided input. After that and to evaluate whether the essential bit is critical or not, an upset is injected into it by means of sending a command that includes the physical addresses of the essential bit to SEM IP core. When SEM IP confirmed that the essential bit value is toggled, then the same inputs are provided by the test bench controller for DUT and the design execution is initiated again. After that,

ALGORITHM 1: Test bench controller flow for calculating critical bits number

```
INPUTS: essential bits location, input-sets
OUTPUTS: critical bits number
for i = 0, i < input − setsnumber, i++ do
    current input vector = read first input vector from input-set(i);
    current essential bit= read the location of the first essential bit in essential bits list;
    for all essential bits do
        Provide current input vector for DUT;
        Initiate execution of DUT;
        Wait until DUT output vector is ready;
        Store generated output vector as golden output;
        Inject error into current essential bit;
        Provide the same input vector for DUT;
        Initiate execution of DUT;
        Wait until DUT output vector is ready;
        Store generated-output as output;
        if output ≠ goldenoutput then
            Add the number of critical bits;
        end
        Flip back the value of current essential bit;
        current input vector = read next input vector;
        current essential bit= read the location of next essential bit;
    end
end
```

the output of the DUT in presence of the fault is compared with the golden output. If they are different, the evaluated bit is accounted as a critical bit. Finally, the value of the infected bit is flipped back by sending a new injection commend with the same physical address.

A critical bit is input-dependent, which means a bit may be accounted as a critical bit for some inputs while for others may not. Thus, an average is taken across all the critical bits number for different input sets. Each input set composed of a set of input vectors that are randomly generated or generated based on input vectors used in a real application.

B. FF Test Bench Implementation

Algorithm. 2 depicts the schema for evaluating the design sensitivity in presence of errors in flip-flops. The evaluation is based on simulating the behavior of the circuit using HDL model of the design. Error injection is achieved by modifying the HDL model to include fault injection capability. Thus, all the flip-flops in the HDL model are replaced with modified versions enabling us to inject faults.

Figure 4 shows the standard implementation of flip-flops and the modified flip-flop which has an extra XOR gate. In the modified flip-flop the input is XORed with an inject signal. When the inject signal is high the value of flip-flop is flipped, otherwise it is associated only with input. The

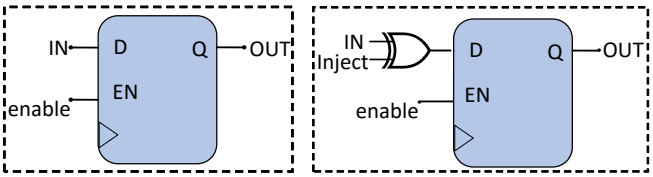


Fig. 4: Standard flip-flop and modified inject-able flip-flop

inject signal must be in the input list of the top module of the design so that test bench module can access it and control it when a fault required to be injected. Algorithm. 2 presents the flow of calculating FFDVF. Since it is very time-consuming to manually modify codes in big designs, for the sake of fault-injection, the net-list of the design generated with an HDL synthesizer tool is automatically parsed and modified. A sample-list file is also generated which contains inputs, expected outputs, target flip-flop and the clock cycle of injection for each simulation run. The modified net-list and sample-list of the design are used by test bench that performs the evaluation process as presented in following steps:
1) Read inputs, expected output, flip-flop number N and clock cycle number C from sample-list. 2) Run the test bench until the moment error has to be injected. 3) Inject error by activating the inject signal bit associated with the flip-flop N for one clock cycle. 4) Continue simulation until the testbench is completely executed. 5) Compare the circuit output with the expected output.
For large circuits and for long test benches, the evaluation process can become extremely long and time-consuming. The usual approach to handle this complexity is to sample the errors set in order to obtain statistical results. Moreover, in order to make calculated effective errors number independent from the inputs, the number of effective errors is calculated for some various input sets and an average is taken out of them. Thus, some sample-lists are generated that each of them composes of the list of test vectors and each test vector consists of the inputs vector, corresponding golden outputs, target flip-flop to be injected into and injection clock cycle.

The proposed test bench is also used for calculating the effective errors number in BRAM and DiRAM.

V. EDUCATIONAL EXAMPLEs

The DS is assessed for various designs i.e. a streaming Fast Fourier Transform (FFT) application implemented in radix-2 architecture and 32-bit Adder. The DS calculations are obtained for the system using FPGA Zynq-7000 device. According to the proposed methodology ϵ_{des} and DVF are required to obtain DS. Table IV shows partial ϵ_{des} which are calculated based on the cross-sections in table II for Zynq-7000. For a fair comparison, the cross-section for the same LET is used for all the designs. For different use cases based on the environment specification which circuit is executed in, specific LET is required to be considered.
In addition, for each design, partial vulnerability factor associated to each memory type is calculated and represented

ALGORITHM 2: Test-bench controller flow for calculating number of effective errors

INPUT: design HDL model, sample-lists
OUTPUT: critical bits number
Take design HDL model;
Generate synthesized net-list;
Generate fault inject-able net-list;
for $i = 0, i < sample - listsnumber, i++$ **do**
    current input vector = read first inputs vector from sample-list(i);
    current golden result = read first golden outputs vector from sample-list(i);
    current flip-flop number = read first flip-flop number from sample-list(i);
    current injection cycle = read first injection cycle from sample-list(i);
    for *all sample errors* **do**
        Run simulation until current injection cycle;
        Inject fault into current flip-flop ;
        Resume simulation until it finishes;
        if $outputs \neq goldenoutputs$ **then**
            add the number of effective errors;
        end
        Reset Simulation;
        current input vector = read next input vector;
        current golden result = read next golden vector;
        current flip-flop number = read next flip-flop number;
        current injection cycle = read next injection cycle;
    end
end

TABLE IV: Partial ϵ_{des} for various designs based on cross-section in table II

Design		CM	FF	DiRAM
FFT	used bits	172388	48944	18179
	ϵ_{des}	2.41e-4	9.89e-5	2.55e-5
ADDER	bits	5108	NA	NA
	ϵ_{des}	7.15e-6	NA	NA

in table V. The injections for CM show number of essential bits which are injected in and failures show the number of critical bits. The calculated DVFs for both designs are aligned with the Xilinx report [29] which estimates that no more than 0.11 fraction of configuration bits are critical, even when the whole FPGA is used. In addition and as expected this result is similar to results reported in [24] for ADDER implemented in Zynq-7000.

Configuration bits corresponding to routing elements are more sensible than the ones used in LUTs, as roughly 80% of failures are due to SEU in routing elements [19]. Given the results, DVF of streaming FFT application that uses more routing elements is higher than ADDER circuit. For the FF and DiRAM, the injections show the number of error injected and failures is equal to effective errors. From the table, the FF-wise DVF is more than CM-wise, a coherent outcome since FF stores the user data and that makes it more susceptible to error than CM bits. However, because the FFT architecture rounds the calculation in middle stages of its calculation, some of the errors are inevitably masked. The DS values in table VI are obtained by multiplying the ϵ_{des} with the DVF corresponding to each memory type and add them all. For instance the DS for FFT is calculated from sum of partial product of CM i.e $0.126e-4$, partial product of FF i.e $0.366e-4$ and partial product of DiRAM i.e. $0.071e-4$. The partial product results show that the contribution of flip-flops is 2.904 times more than CM in FFT DS, although ϵ_{des} from table IV related to flip-flops is roughly 2.4 times less than configuration bits. It is certified the importance of considering the DVF in our proposed model.

TABLE V: Partial DVF for various designs

Design		CM	FF	DiRAM
FFT	Injections	172388	48944	18179
	Failures	9001	18057	5173
	DVF	0.0522	0.37	0.28
ADDER	injections	5223	NA	NA
	Failures	119	NA	NA
	DVF	0.0227	NA	NA

TABLE VI: DS for various designs implemented in Zynq-7000 FPGA

Design	DS
FFT	0.563e-4
ADDER	0.162e-6

In addition, these results indicate that flip-flops used in FFT are more sensible to SEUs than CM which guide designers to use the budget of adding redundancy more for flip-flops than CM. For example, one of the common mitigation techniques is Triple Modular Redundancy (TMR) [30] in which three copies of a module are implemented and correct output is selected by majority voting. If TMR is applied for FFT, about 200% of plain design, redundancy is added to design which is significant and in some cases may violate system constraints like area. In order to balance between system constraints and required fault tolerance, TMR redundancy can be applied for most sensitive component which is flip-flop according to proposed DS calculation. Thus, as shown in table VII with adding 61% (of plain design), redundancy to the system, the design sensitivity is reduced only 18% less than full TMR.

TABLE VII: Evaluating the importance of considering DS in fault tolerance schema

Design		Full TMR	DS-Wise TMR
FFT	sensitivity reduction	98%	80%
	redundancy	200%	61%
Design-1	sensitivity reduction	95%	76%
	redundancy	200%	59%

The proposed DS can be also used for selectively adding redundancy to various component of a complex design. For instance, consider design-1 consists of an adder and FFT which both of them located on the critical path. Table VI indicates that FFT is roughly 500 times more sensitive than ADDER. Thus, the budget for adding redundancy is selectively used for FFT flip-flops to achieve fault tolerance requirement with less redundancy. Thus, as shown in table VII with adding 70% less redundancy compared to full TMR, the design sensitivity is reduced only 19% less than full TMR.

VI. CONCLUSION

Fault tolerance techniques for protecting digital system designs and specially FPGA-based systems against SEU mainly come with additional overheads. Given the extensive interest in using COTS devices in automotive and space missions, it is essential to use fault-tolerance techniques for processing components in a system vulnerable to SEU effects. In order to compute the SEU sensitivity of a design implemented in FPGA, a generic model was presented in this paper. This model empowers digital designers to tolerably compare the criticality of various parts of a system to make use of fault tolerance features. Toward this, all of the system parameters that assess the sensitivity of a design were taken into account in this paper. Those system parameters include device characterization, design architecture and environment specification as well as various elements of the design which are sensible to SEU, i.e. CM, BRAM, distributed RAM and flip-flops. Schemes to calculate the parameters of the proposed methods were also presented followed by fault modeling and fault injection techniques. Results were certified the importance of using proposed design sensitivity to select proper fault tolerance schema.

REFERENCES

[1] Synthesis: European space next generation processor for onboard payload data processing application, Technical Report. TEC-EDP/2007.35/RT, European Space Agency, 2007.

[2] R. Trautner. ESA's roadmap for next generation payload data processors. Proc. DASIA Conference, 2011.

[3] Srinivasan, S., Krishnan, R., Mangalagiri, P., Yuan, X., Narayanan, V., Irwin, M.J., and Sarpatwari, K., 2008. Toward Increasing FPGA Lifetime. Dependable and Secure Computing. IEEE Transactions on 5, 2, pp. 115-127.

[4] D.M. Hiemstra, and V. Kirischian, 2012. Single Event Upset Characterization of the Virtex-6 Field Programmable Gate Array Using Proton Irradiation. In Radiation Effects Data Workshop (REDW), 2012 IEEE, pp. 1-4.

[5] T.Calin, M.Nicolaidis, and R.Velazco. Upset Hardened Memory Design for Submicron CMOS Technology. IEEE Transactions on Nuclear Science, Vol. 43, No. 6, December 1996.

[6] T.J.Slegel, et al. IBM's S/390 G5 Microprocessor Design. IEEE Micro, pp 12-23, March/April, 1999.

[7] Shubhendu S. Mukherjee, Mike Kontz, and Steven K. Reinhardt. Detailed Design and Implementation of Redundant Multithreading Alternatives. Proceedings of the 29th Annual International Symposium on Computer Architecture (ISCA), May 2002.

[8] Todd M. Austin. DIVA: A Reliable Substrate for Deep Submicron Microarchitecture Design. 32nd Annual International Symposium on Microarchitecture (MICRO), November 1999.

[9] Carmichael, M. Caffrey, and A. Salazar. Correcting Single-Event Upsets Through Virtex Partial Configuration. Technical Report. Xilinx. 2000.

[10] R. Santos, S. Venkataraman, A. Das, and A. Kumar. Criticality-aware scrubbing mechanism for SRAM-based FPGAs. In IEEE International Conference on Field Programmable Logic and Applications (FPL14), 2014.

[11] R. Santos, Sh. Venkataraman, A. Kumar. Generic scrubbing-based architecture for custom error correction algorithms. International Symposium on Rapid System Prototyping (RSP), 2015, pp 112-118.

[12] R. Santos, Sh. Venkataraman, A. Kumar. Dynamically adaptive scrubbing mechanism for improved reliability in reconfigurable embedded systems. 52nd ACM/EDAC/IEEE Design Automation Conference (DAC), 2015, PP 1-6.

[13] M. Bellato, M. Ceschia, M. Menichelli, A. Papi, J. Wyss, and A. Paccagnella. Ion beam testing of SRAM-based FPGAs. in Proc. 6th European Conf. Radiation and Its Effects Components and Systems, Sep. 2001, pp. 474-480.

[14] C. Carmichael, E. Fuller, J. Fabula, and F. Lima. Proton testing of SEU mitigation methods for the Virtex FPGA. in Proc. Military and Aerospace Applications Programmable Logic Devices, Washington, D.C., Sep. 2001.

[15] G.H Asadi and M. B. Tahoori. Soft error mitigation for SRAM-based FPGAs in High-Performance Information Systems. in Proc. 23rd IEEE VLSI Test Symp. , 2005, pp. 207-212.

[16] G. Asadi and M. B. Tahoori, Soft error rate estimation and mitigation for SRAM-based FPGAs. in Proc. 13th ACM Int. Symp. Field-Programmable Gate Arrays. Monterey, CA, Feb. 2005, pp. 149-160.

[17] M. Wirthlin, E. Johnson, N. Rollins, M. Caffrey, and P. Graham. The reliability of FPGA circuit designs in the presence of radiation induced configuration upsets. in Proceedings of the 2003 IEEE Symposium on Field-Programmable Custom Computing Machines.

[18] M. Gokhale, P. Graham, E. Johnson, N. Rollins, and M.Wirthlin. Dynamic reconfiguration for management of radiation-induced faults in FPGAs. in Proc. 18th Int. Parallel and Distributed Processing Symp., Santa Fe, NM, Apr. 2004, pp. 145-150.

[19] P. Graham, M. Caffrey, J. Zimmerman, D. E. Johnson, P. Sundararajan, and C. Patterson. Consequences and categories of SRAM FPGA configuration SEUs. presented at the Military and Aerospace Applications Programmable Logic Devices Int. Conf., Washington, DC, Sep. 2003.

[20] F. Lima, C. Carmichael, J. Fabula, R. Padovani, and R. Reis. A fault injection analysis of Virtex FPGA TMR design methodology in Proc. 6th European Conf. Radiation Effects Components and Systems, Grenoble, France, 2001, pp. 275-282.

[21] M. Rebaudengo, M. S. Reorda, and M. Violante. Simulation-based analysis of SEU effects on SRAM-based FPGAs. in Proc. 12th Int. Conf. Field-Programmable Logic and Applications, Montpellier, France, Sep. 2002, pp. 607-615.

[22] L. Sterpone and M. Violante. A New Partial Reconfiguration-Based Fault-Injection System to Evaluate SEU Effects in SRAM-Based FPGAs. IEEE Transaction on Nuclear Science, VOL. 54, NO. 4, August 2007.

[23] M. Violante, L. Sterpone, M. Ceschia, D. Bortolato, P. Bernardi, M. Sonza Reorda, and A. Paccagnella. Simulation-Based Analysis of SEU Effects in SRAM-Based FPGAs. IEEE Transaction on Nuclear Science, VOL. 51, NO. 6, DECEMBER 2004

[24] I. Villata, U. Bidarte, U. Kretzschmar, A. Astarloa, J. Lazaro. Fast and Accurate SEU-Tolerance Characterization Method for Zynq SoCs. Field Programmable Logic And Applications (FPL), 2014

[25] K. Chapman SEU Mitigation Techniques for SRAM-based FPGAs. TWEPP2015, 30th September 2015.

[26] L. Harten, R. Jordans, H. Pourshaghaghi, 2017. Necessity of Fault Tolerance Techniques in Xilinx Kintex 7 FPGA Devices for Space Missions: A Case Study. Euromicro Conference on Digital System Design (DSD), 2017, PP 299-306.

[27] M. Amrbar, F. Irom, S. M. Guertin, G. Allen, 2015. Heavy Ion Single Event Effects Measurements of Xilinx Zynq-7000 FPGA. IEEE Radiation Effects Data Workshop (REDW), 2015, PP 1-4.

[28] D. S. Lee, G. R. Allen, G. Swift, M. Cannon, M. Wirthlin, J. S. George, R. Koga, K. Huey, 2015. Single-Event Characterization of the 20 nm Xilinx Kintex UltraScale Field-Programmable Gate Array under Heavy Ion Irradiation. IEEE Radiation Effects Data Workshop (REDW), 2015, PP 1-6.

[29] K. Chapman, 2010. Virtex-5 SEU Critical Bit Information Extending the capability of the Virtex-5 SEU controller. Feb. 2010

[30] ESA Requirements and Standards Division ESTEC. Techniques for radiation effects mitigation in ASICs and FPGAs handbook.