Emulation-Based Hierarchical Fault-Injection Framework for Coarse-to-Fine Vulnerability Analysis of Hardware-Accelerated Approximate Algorithms

Ioannis Chadjiminas, Ioannis Savva, Christos Kyrkou, Maria K. Michael, Theocharis Theocharides
KIOS Research Center, Department of Electrical and Computer Engineering
University of Cyprus
Nicosia, Cyprus
{chadjiminas.ioannis, savva.giannis, kyrkou.christos, mmichael, ttheocharides}@ucy.ac.cy

***Abstract*— This paper proposes a hierarchical fault injection emulation framework tailored to the structure of complex and large application-specific circuits, that performs vulnerability analysis of the system for single event upsets (SEUs) at different design granularities in real-time. In particular, the framework allows for efficient probabilistic modelling of the SEU impact, making it particularly applicable for hardware-accelerated approximate applications such as multimedia, computer vision and image/signal processing, due to its high processing speed and real-time capabilities. The framework is emulated on an FPGA-based platform and evaluated using a depth computation kernel, both in standalone manner as well as within a robotic obstacle avoidance application.**

I. Introduction

Soft errors have become an increasingly worrying issue for embedded hardware systems. Usually modeled as single event upsets (SEUs), these adverse phenomena cause the value of a signal to be temporarily inverted, which, if registered, can lead to undesired results at the application level [1]. Emerging research suggests that the output of many applications that rely on approximate algorithms such as computer vision, remain tolerable even when affected by upsets during operation [2]. These applications, by nature, typically produce outputs that are computed based on dynamic input and feedback, and approximate computational kernels as well as type and amount of input data. Such applications are typically employed in embedded and cyber-physical systems, and as such the need for reliable and secure operation is ensured by techniques such as redundancy and hardening; however, this can often lead to prohibitive costs in terms of area, complexity, performance and energy budget. As such, there is evident need to understand the impact of soft errors on the targeted application output, in an effort to budget only the necessary hardware resources for radiation hardening or redundancy. Hence, traditional simulation-based fault-injection methods for vulnerability analysis methods become impractical, as a large amount of input data is required to cover as many conditions as possible, and under different scenarios, in order to get accurate statistical results which increases the required time for analysis.

There is an abundance of relevant research works that attempt to study circuit vulnerability in the presence of SEUs. The majority of these works focus on high-level simulation and fault modeling using state transition models, formal and mathematical models, technology models, and application-specific workloads. While simulation-based techniques can provide high controllability and observability, they are time consuming techniques especially for large circuits [3]. In contrast, emulation-based fault injection techniques can be used as an alternative way to speed up the fault injection campaigns and provide good observability and controllability [4-8]. Nonetheless, the study of fault injection requirements for probabilistic approximate applications has received little attention. However, existing emulation based techniques incur either prohibitive times for large input data streams or require a lot of memory which may not be available on portable embedded platforms.

Our previous work [9] demonstrated how the severity of SEUs can change at different application layers thus emphasizing the need for hierarchical vulnerability analysis. Here, we therefore propose an emulation-based hierarchical fault injection mechanism that can provide a complete coarse-to-fine vulnerability analysis for very large circuits with long workloads. Through this hierarchical approach we address the problem of rapid but selective fault injection in a scalable manner, while also being able to model different fault-injection rates at different granularities. These features allow the framework to perform vulnerability analysis for circuits used for approximate applications, where both the use and the output of a component occur probabilistically based on the input data. The fault injection mechanism has been implemented on a Field Programmable Gate Array (FPGA) platform as a custom circuit, and it is capable to achieve very high fault injection rates in order to get accurate and real-time statistical results. The proposed mechanism is capable of identifying the most sensitive components and subsequently most sensitive registers and bits of these components dynamically. In addition, the system is standalone and does not require interfacing with a host PC neither the fault locations to be pre-stored as they are dynamically generated and injected based on a probabilistic model.

The paper is organized as follows. Section II provides a brief overview of existing emulation-based fault injection approaches. Section III outlines the proposed hierarchical fault injection mechanism. Section IV presents the experimental platform and analysis results. Finally, Section V concludes the paper.

II. Emulation-Based Related Work

Our focus in this paper is on emulation-based fault injection techniques. Such techniques are either reconfiguration based [4] or use a specific circuit [5-8], In the first case, the fault is implemented within the application circuit and is loaded on the reconfigurable hardware platform (e.g. FPGA) as part of the implementation. Although that limits area overheads, the fault injection speed is a limiting factor, since the design needs to be reconfigured and re-loaded (or through partial reconfiguration) after the injection of a new

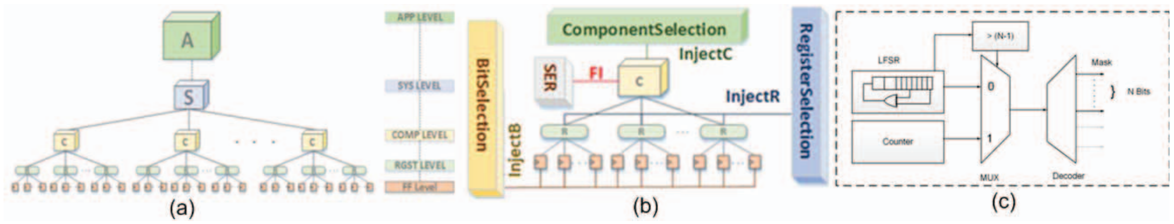

Figure 1: (a) Hierarchical Hardware Realized Algorithms. (b) Fault Injection Mechanism. (c) Base Architecture for Selection Mechanisms.

fault. Circuit-based techniques are more suitable when speed is required, as they implement the circuit-under-analysis (CUA) and the fault injection mechanism on the same fabric. The downside is the associated hardware overhead. A circuit-based fault injection framework can be built with a number of different ways, either via collaboration with a host PC or fully autonomous in hardware. The former is flexible but very time consuming and restricts the full exploitation of the hardware platform. In the latter case the entire fault injection list (location and time) needs to be saved on the FPGA memory [8], or an exhaustive fault injection needs to be performed. Further, these techniques utilize scan-chains, and thus performance is inversely proportional of the workload length [5]. In contrast to existing works, the proposed framework offers probabilistic fault distribution over the target system design at different levels of granularity. In addition, as we only use FPGA technology as an emulation engine for analysis purposes in this work, we assume that faults can occur only in the emulated design, in contrast to existing works, such as [10-11], which study the vulnerability of the SRAM-based FPGA fabric.

III. Proposed Framework Overview

In this section we describe the concept and the hierarchical-architecture of hardware implemented algorithms. In addition, we will show the design of the overall proposed fault injection framework which is optimized to this hierarchical structure in order to perform fast fault injection experiments and rapidly determine the most SEU sensitive components, registers and bits of a system.

A. Hardware-Realized Algorithms and Applications.

A typical structure of hardware-based algorithms (Fig 1-a) can be abstracted at five basic levels of hierarchy (focusing only on the memory elements); Flip-Flop (FF) which corresponds to all memory bits, Register (REG) corresponding to groups of bits, Component (CMP) corresponding to groups of registers, System (SYS) which correspond to group of components that constitute the hardware realized algorithm, and Application (APP) level that uses the hardware-realized algorithm as a kernel. These applications typically compute data in frequent time intervals (*computing windows*) to obtain a meaningful output. This distinguishes them from traditional applications where a result may be produced at every cycle.

B. Proposed Fault Injection Framework

The proposed Fault Injection Mechanism (FIM) is tailored to the hierarchical nature of hardware-implemented algorithms and allows the designers to restrict the injection of faults at different layers of the hierarchy of the system thus offering analysis at different granularity. The FIM consists of four different fault injection sub mechanisms, as shown in Fig.1-b. We briefly describe them here but more details can be in [9]. The first mechanism controls the soft error rate (SER) and determines how frequently the signal FI (*fault_inject*) should be asserted (which initiates the injection of a fault). The selection of a component where the fault will be injected is done by the component selection mechanism by asserting the appropriate signal *injectC* of the selected component. Concurrently, the register selection mechanism asserts the corresponding signal *injectR* inside the selected component which targets a single register. At the same time the bit that will be inverted from that register is selected through the bit selection mechanism. All selection mechanisms utilize variations of the basic structure shown in the Fig. 1-c which consists of a counter, an LFSR and a decoder. The mechanism operates in four different modes to allow analysis at the desired abstraction level:

1. **System mode:** Faults are injected randomly to all the memory bits of the system to determine the overall impact of SEUs to the system and application level.

2. **Component mode:** Faults are injected randomly to all the memory bits of a single component to determine the impact of each component at the system and application level.

3. **Register mode:** Faults are injected randomly to all the memory bits of a single register to determine the impact of each register at the system and application level.

4. **Bit mode:** Faults are injected exclusively to a single bit to determine its impact at the system and application level.

The benefits offered to the designers by the hierarchical SEU evaluation are twofold. Firstly, it provides the capability to perform the fault injection experiments at different granularities. Thus, the designers may choose to identify and protect either the most critical components, registers or bits depending on the hardware overhead, reliability factor, and power constraints of the CUA. Another benefit is that through this hierarchical analysis, the FIM provides a faster way to detect the critical bits. A traditiitional approach in order to find the critical bits of the system injects a fault at a single memory bit each time, and then tracks its effect on the system and application level. Once the effect of this memory bit has been determined, they inject a fault to another memory bit and repeat the same process. This must be perfomed for all memory bits in the system in order to find the criticality of each memory bit. Hence, as the complexity is proportional to the total bits in the circuit, this process becomes prohibitive for very large circuits with long workloads. However, our framework takes advantage the fact that many faults may not propagate to the output of the system. In other words, components and registers may not have an impact to the ouput of the system, thus they are excluded for the bit analysis. Starting from component level, our framework injects faults at the components of the system and then tracks their effect to the output of the system. From this fault injection campaign, many components, that do not affect the output, can be excluded from the next analysis at the lower level. This is repeated for the registers and bits of the system, in order to exclude resilient bits, thus avoiding injecting faults to all bits.

C. Framework Flexibility

The sensitivity of a system node against SEUs is determined by either its functionality or by its size. It is necessary to factor in both cases in order to provide the means for accurate analysis. To do so we implement two different probability models that impact how faults are injected in the system. The component weighted version identifies critical components based only on their functionality, whereas the bit-weighted version also identifies critical components based on their size as well. The two different approaches can be used to also test the system at different granularity levels.

1) Component Weighted

In the component weighted version of the mechanism the probability that a fault will be injected to a component is the same for all the components in the system irrespective of the total number of bits in each component. Hence, faults are injected with different probabilities at each FF but with the same probability at each component. We implement a circuit that distributes faults in equal probability using a single LFSR. As shown in the Fig.2-a, the length of the LFSR is equal to the logarithm of the number of components N_C, in the system. Thus, the LFSR generates a specific number for each component and a decoder is used to activate the fault injection mechanism of each component so that faults are injected. Under this scheme (Fig. 2-a) the probability P_F^{ij} of a flip-flop in register R_j within component C_i to be injected with a fault is proportional to the number of bits (N_B^j) in each register R_j, and number of registers (N_R^i) in each component C_i as shown in (1). The probability P_C^i of a component C_i being injected with a fault is the same as shown in (1).

$$P_F^{ij} = \frac{1}{N_C} \times \frac{1}{N_R^i} \times \frac{1}{N_B^j}\ , P_C^i = \frac{1}{N_C} \qquad (1)$$

2) Bit Weighted

In the previous approach the number of bits contained in each component is not considered and thus the probability of a fault to be injected is the same for all the components. However, the probability of a fault to be injected on a single component depends on the number of registers and flip-flops therein. Hence, the modified bit weighted version of this mechanism also considers the number of bits contained in each component so that faults are injected with a different probability at each component but with the same probability in each flip flop. An LFSR is used that has the same length as the total number of bits in the system. A range is selected for each component depending on how many bits it has over the total amount and whenever the LFSR-generated number falls into that range, it is selected. Similarly the same process is followed for the registers in the system. As a result, in this version (Fig. 2-b), the probability P_F^{ij} for each flip flop in the system to be injected with a fault is equal, 1 over the total number of bits, as shown in (2). The LFSR length increases logarithmically; resulting in important hardware savings.

$$P_F^{ij} = \frac{1}{\sum_{i=1}^{N_C} \sum_{j=1}^{N_R^i} N_B^j}\ , P_C^i = \frac{1}{\sum_{j=1}^{N_R^i} N_B^j} \qquad (2)$$

IV. Experimental Evaluation and Discussion

The framework is emulated on a Kintex-7 FPGA platform and evaluated using a disparity estimation (DE) kernel, both in standalone manner as well as within an obstacle avoidance

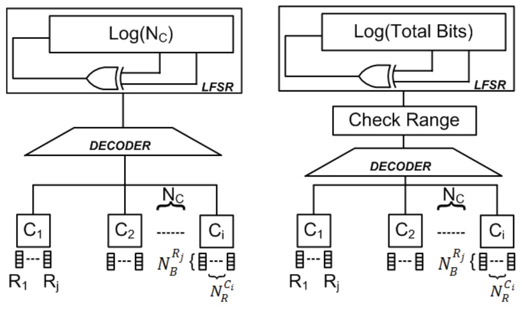

Figure 2 (a) Component-Weighted Scheme (b) Bit-Weighted Scheme

application. The fault injection mechanism was implemented alongside two instances of a hardware implementation of a disparity estimation algorithm (DE) [9]. One instance was injected with faults while the other instance was running fault-free. Both instances received the same camera feed and operate at the video rate which is much lower than the operational frequency achieved by the design (thus all operational conditions were identical). The percentage of affected pixels (error) is measured by comparing the two outputs. In addition, an obstacle avoidance (OA) algorithm is also implemented which decides the direction of movement based on the produced disparity map. More details can be found in [9]. For the OA we consider a wrongly computed output as a wrong decision regarding the direction. The architecture of the DE kernel is comprised of 12 major components, briefly listed here due to space limitations: an *address generator (ag),* a *scan-line buffer (scnbf)*, *serial-in parallel-out (dtpr)* register structure, the pixel cost units (*pcn&pco*), the *box-filter (bfltr),* a simple *Winner-Takes-All (wta)* minimization approach module, two multipliers (*m4&m6*), a summation unit (*cs*), and two synchronization units (*s1 & s2)*. The OA module comprises of eight registers, eight adders, and a comparator. The total number of registers in the system is 1813 corresponding to ~130,000 flip-flops. The overall experimental approach is summarized in Fig. 3. We performed two different types of vulnerability analysis at each case; analysis at the component level (component-weighted and bit-weighted), and analysis at the register level (bit-weighted only), targeting individual registers of each component. We used ten fault injection rates, ranging from 0.0144% to 2.10%, for each emulation.

A. Component Level: Component- and Bit-Weighted Analysis

Fig. 4 and Fig. 5, show the criticality (reflected in the percentage reduction of the quality of the output shown on the Y-axis) of each component, when faults are randomly injected within a component, and when faults are directed randomly in the component's registers for the two case studies (DE and OA) respectively. The analysis is done for various fault injection rates (each bar corresponds to different fault rate). At the component-level, the framework can be used to extract vulnerability information regarding the components of the CUA, and determine their overall criticality and impact with regards to the quality of the output based both on their functionality (component-weighted) and size (bit-weighted approach). Clearly, for our case study, the address generator (*ag*) component shows the highest vulnerability, even at low fault rates, where an SEU can drastically impact the quality of the output of the DE and also affect the decision of the OA.

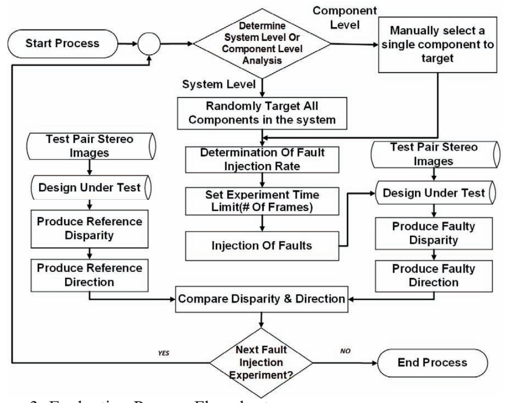
Start Process
Determine System Level Or Component Level Analysis
Component Level
Manually select a single component to target
System Level
Randomly Target All Components in the system
Test Pair Stereo Images
Determination Of Fault Injection Rate
Test Pair Stereo Images
Design Under Test
Set Experiment Time Limit(# Of Frames)
Design Under Test
Produce Reference Disparity
Injection Of Faults
Produce Faulty Disparity
Produce Reference Direction
Produce Faulty Direction
Compare Disparity & Direction
YES
Next Fault Injection Experiment?
NO
End Process

Figure 3: Evaluation Process Flowchart

B. Register Level: Bit-Weighted Analysis

Among the most important parameters however in determining the vulnerability of a component, is also its size with regards to the system; for example, the box filter (*bfltr*) in our case, in the component-weighted emulation, seems among the most critical components. However, if we look each register of the entire DE module independently, the registers comprising the box filter may not be as critical as the component-weighted approach suggests. This happens because the component-weighted approach does not consider the size of a component and can thus underestimate the vulnerability of a large component while at the same time can overestimate the vulnerability of a small component. The proposed framework therefore, allows for detailed, fine-grain quantitative analysis, at all the registers of the entire module, by using the bit-weighted approach. Fig. 6 illustrates the criticality of the individual registers of the top three most critical components, sorted in terms of individual register vulnerability; address generator (*ag*), box filter (*bfltr*) and summation unit (*cs*). The results are shown for low fault injection rates (0.0144%), medium (0.1298%) and high (2.1%). We only show the top three components but results for all registers were obtained.

C. Overheads

We compare the hardware resources of the DE module with and without the fault injection mechanism, using a Kintex-7 FPGA. The fault injection framework requires approximately 45% extra LUTs and slice register resources, since for every register, a multiplexer is also required in addition to the fault injection mechanism. There a slight increase in the critical path (by the delay of 2-1 multiplexer), as faults are injected within at most 2 cycles.

V. Conclusions

This paper presented a hierarchical fault injection emulation framework that can provide dynamic reliability analysis for hardware-accelerated approximate algorithms at different levels of granularity. Future work will focus on introducing component usage probabilities in the fault injection process to model the probabilistic nature of the approximate applications.

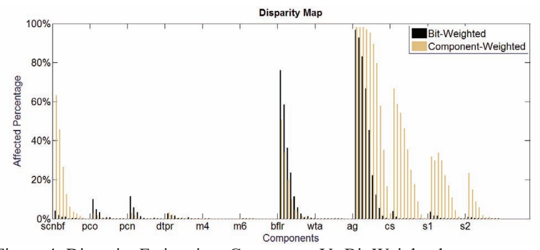

Figure 4: Disparity Estimation: Component Vs Bit Weighted

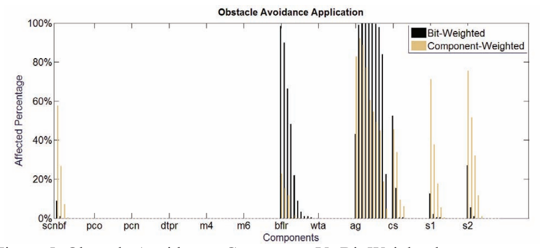

Figure 5: Obstacle Avoidance: Component Vs Bit Weighted

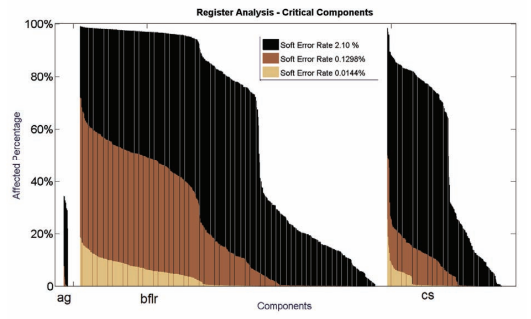

Figure 6: Disparity Estimation: Register Criticality

References

[1] Baumann, R., "Soft errors in advanced computer systems," in *Design & Test of Computers, IEEE* , vol.22, no.3, pp.258-266, May-June 2005.

[2] Nowroth, D.; Polian, I.; Becker, B., "A study of cognitive resilience in a JPEG compressor," *IEEE International Conference on Dependable Systems and Networks*, pp.32-41, 24-27 June 2008.

[3] M. Maniatakos, M. Michael, C. Tirumurti, Y. Makris, "Revisiting Vulnerability Analysis in Modern Microprocessors," *IEEE Transactions on Computers,* vol.64, no.9, pp.2664-2674, Sept. 1 2015.

[4] U. Legat, A. Biasizzo, F. Novak, "Automated SEU fault emulation using partial FPGA reconfiguration", *IEEE 13th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, 2010, pp.24-27, 14-16 April 2010.

[5] Lopez-Ongil, C.; Garcia-Valderas, M.; Portela-Garcia, M.; Entrena, L., "Autonomous Fault Emulation: A New FPGA-Based Acceleration System for Hardness Evaluation", *IEEE Transactions on Nuclear Science*, vol.54, no.1, pp.252-261, Feb. 2007.

[6] Riefert, A.; Muller, J.; Sauer, M.; Burgard, W.; Becker, B., "Identification of critical variables using an FPGA-based fault injection framework", *IEEE VLSI Test Symposium,* pp.1-6, April 29-May 2 2013.

[7] B. Rahbaran, A. Steininger, T. Handl, T., "Built-in fault injection in hardware - the FIDYCO example", IEEE International Workshop in Electronic Design, Test and Applications, pp.327-332, 28-30 Jan. 2004.

[8] 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," *IEEE International Symposium on Industrial Electronics,* pp.2276-2281, June 30 2008-July 2 2008.

[9] I. Chadjiminas, C. Kyrkou, T. Theocharides, M.K. Michael, C. Ttofis, "In-field vulnerability analysis of hardware-accelerated computer vision applications," *25th International Conference on Field Programmable Logic and Applications (FPL)*, pp.1-4, 2-4 Sept. 2015.

[10] C. Bolchini,F. Castro, A. Miele, "A Fault Analysis and Classifier Framework for Reliability-Aware SRAM-Based FPGA Systems," *24th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems,* pp.173-181, 7-9 Oct. 2009.

[11] A. Sari, D. Agiakatsikas, M. Psarakis, "A soft error vulnerability analysis framework for Xilinx FPGAs" In *Proceedings of the 2014 ACM/SIGDA international symposium on Field-programmable gate arrays* (FPGA '14). ACM, New York, NY, USA, 237-240.