SCHIFI: Scalable and flexible high performance FPGA-based fault injector

Suman Sau, Maha Kooli, Giorgio Di Natale, Alberto Bosio, Amlan Chakrabarti

SCHIFI: SCalable and flexible High performance FPGA-based Fault Injector

Suman Sau[12], Maha Kooli[1],
Giorgio Di Natale[1], Alberto Bosio[1]
[1]LIRMM CNRS-UM
<firstname.lastname>@lirmm.fr

Amlan Chakrabarti[2]
[2]University of Calcutta
acakcs@caluniv.ac.in

***Abstract*—One of the consequence of the scaling down of latest technologies, is that digital circuits are more prone to be affected by faults caused by physical manufacturing defects, environmental perturbations (*e.g.*, radiations, electromagnetic interference), or aging-related phenomena. Understanding the behavior of the whole system in the presence of faults affecting digital circuits is crucial for designing and validating fault tolerance techniques. Fault injection is a well-known and widely used approach for estimating the behavior of a system in the presence of errors. Among the different techniques, FPGA-based fault injection is one of the most popular since it allows a low-cost, rapid and accurate approach compared to simulation- and hardware based fault injectors. In this paper we present a flexible FPGA fault injector able to inject in the memory hierarchy of a given system.**

Keywords—Reliability, FPGA, Fault Injector, Hardware Faults, Memories

I. INTRODUCTION

System reliability has become an important design aspect for computer systems due to the aggressive technology miniaturization, which introduces a large set of different failure sources for hardware components [1][2][3]. The hardware system can be affected by faults caused by physical manufacturing defects, environmental perturbations (*e.g.*, radiations, electromagnetic interference), or aging-related phenomena [4]. Faults propagate through the different hardware structures composing the full system (see Fig. 1). However, faults can be masked during this propagation either at the technological or at architectural level [5][3]. When a fault reaches the software layer of the system, it can corrupt data, instructions or the control flow. These errors may impact the correct software execution by producing erroneous results or prevent the execution of the application leading to abnormal termination or application hang.

The system reliability includes both the software and the hardware reliability. In order to target the overall system reliability, we have to consider: (i) faults affecting the software layer and (ii) the faults affecting the hardware layer. In the literature, several methods and tools have been proposed to accurately evaluate the impact of the faults affecting either the software layer (*e.g.*, Mutation Testing) or the hardware layer (*e.g.*, Fault Injection).

This work has been supported by the joint FP7 Collaboration Project CLERECO (Grant No. 611404).

In this work, we mainly focus on the faults affecting the hardware layer that can actually really impacts on the software layers. The idea is to target a subset of all possible fault locations to speed up the fault injection campaign.

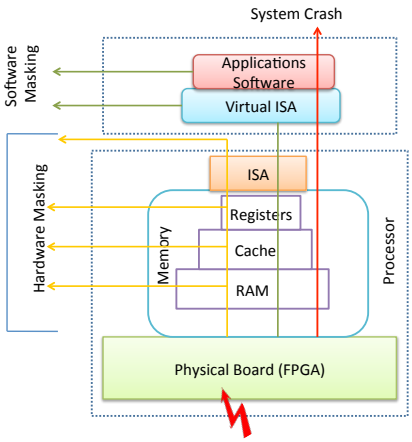

Fig. 1. System Layers and Fault Propagation

For this purpose, we present in this paper a scalable and flexible high performance fault injector based on the use of FPGA board. The proposed tool main aims at giving to the user the possibility to easily build its own fault injector in order to target the desired set of fault locations. We present a set of preliminary results carried out on the open sourceAmber project[6], that is a complete embedded system implemented on the Xilinx Spartan-6 SP605[7] FPGA development board.

The remainder of the paper is organized as follows. Section II present the background and state-of-the-art about fault injections techniques. Section III discusses the architecture of the proposed fault injector. Section IV presents preliminary experimental results. Finally, Section V concludes the paper.

II. BACKGROUND

To evaluate the system reliability, fault injection [8] is a well-known and powerful technique to observe the impact of generated errors on the system behavior. It is based on the realization of controlled experiments to evaluate the system behavior in the presence of artificial faults.

Hardware-based fault injection technique allows injecting physical faults (*e.g.*, bit flip fault, stuck at fault) in the target

hardware system. It uses either a manufactured processor prototype, a simulation of the processor architecture, or an implementation of the processor on an FPGA board. However, they are expensive in terms of execution time and injection facilities (*i.e.*, the way how the faults are really injected in to the hardware). In addition, it is difficult to control the injection instant and location, which limits the fault evaluation.

A. *Physical Fault-Injection Techniques*

The physical fault injection techniques apply external perturbations on the circuit under test to evaluate the reliability [9]. Particle radiations, laser beams or pin forcing are used to create realistic faults:

- *Radiation Methods:* These methods test the device in its real environment by exposing it to the particle radiation [10][11][12]. The advantage of this method is that the device is put in realistic conditions and the fault is correctly modeled. However, using this method it is difficult to control the fault location and instant.
- *Laser Methods:* The laser beams generate a photon material interaction instead of a particle material interaction. These methods can better control the fault location. Researchers prove that both the radiation and laser methods offer accurate results [13][14][15].
- *Pin Forcing:* The pin values at the input/output of the device are directly modified [16][17] to create the same effect of the radiation and the laser methods. This solution is cheaper, but it is only applied on simple circuits [9].

B. *Simulation-based Fault-Injection Techniques*

The simulation-based fault injection techniques do not operate on the physical device, but they employ a model of the device described using a simulation language, such as VHDL [18][19][20]. They can inject faults in the VHDL model either at run-time or at compile-time. Compared to the physical fault injection techniques, the simulation-based techniques are cheaper in term of set-up and can better control where the fault is injected. In addition, they present no risk to damage the hardware system under evaluation. However, they create a computational overhead depending on the complexity of the device under evaluation [21].

C. *FPGA-based Fault-Injection Techniques*

The FPGA-based fault injection techniques implement the device on an FPGA board and perform the fault injection campaigns on the different device components. Compared to the simulation-based techniques, they can precisely control where the fault is injected. Furthermore, they offer a faster fault evaluation [21]. The fault injection process applies one of the following mechanisms:

- *Reconfiguration Mechanism:* The bits of the FPGA board are reconfigured to inject the fault in the specified location [22][23]. The fault injection takes place either at run-time or at compile-time. However, the reconfiguration process creates a time overhead.
- *Instrumentation Mechanism:* Additional circuit elements in the different components of the system are built to inject the fault in the target location, which are called Saboteurs [24][25]. The activation of these elements generates the fault. The instrumentation mechanism is therefore faster than the reconfiguration mechanism.

III. Proposed Fault Injector Architecture

In modern processors-based computation systems, the concept of memory hierarchy is now always implemented. Actually, the main idea behind the memory hierarchy is to reduce as much as possible the need for transferring data from and to the main memory (RAM) and the processor. For this reason, cache memories have been introduced to store data/instructions in order to accelerate their future requests by the processor. To be cost-effective and to enable efficient use of data/instructions, the caches are relatively small compared to the RAM. This cache-based architecture introduces a sort of hardware redundancy for increasing performances. This means that the same variable can have, at the same time, several copies in different locations. It can reside the RAM (the data segment or the stack), the data cache or/and the CPU (the registers), as shown in Fig. 2. The same description can be applied to the register bank embedded in to the processor itself.

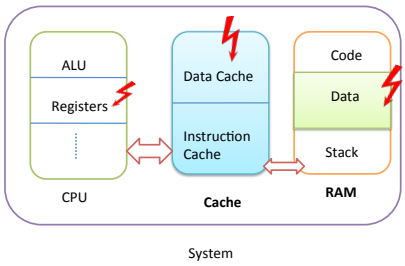

Fig. 2. Data Location in System

In this work we mainly target the components of the memory hierarchy as possible locations for faults that could reach the software layer (*e.g.*,by corrupting a variable or an instruction). The target fault model is the well-known Single Event Upset (SEU) that corresponds to the inversion of the logic value stored in a single memory element (*i.e.,* the single bit). It models soft errors caused by electromagnetic interference, external radiations (such as thermal neutrons, cosmic rays creating energetic neutrons and protons and alpha particles from package decay)[9].

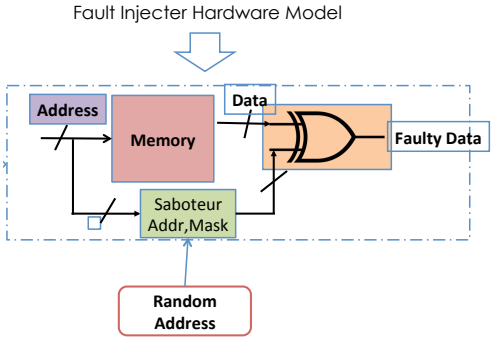

Fig. 3. Fault Injector Hardware Model

In Fig. 3, the hardware block diagram of the implemented Fault Injector (FI) is shown. We simple design a saboteur mask for select the desired memory address and flip a random bit on the data corresponding to that address. Bit flip is done by simply using a XOR gate as shown in the Fig. 3. The memory block depicted in the Fig. 3 can be the RAM, the cache or the register bank. The user can actually specify what is the target.

The Injection mechanism is managed by a Finite State Machine depicted in Fig. 4. “S0” is the FI initial state. When the user set the start command, it goes to “S1” state. In this state depending on the saboteur mask result, the next state will be decided. If the Address of the target memorymatches with the random Address for fault injection with other satisfied conditions, then next state will be“S4” otherwise next state will be “S3”.Actually, the state “SA” corresponds to the activation of the injection. Here, the output of the saboteur is set to logic ‘1’, so that the target bit is flipped by the XOR gate as shown in the Fig. 3. On the other hand, in the state “S3” the injection is not activated. Here, the output of the saboteur is set to logic ‘0’ so that the XOR does not invert the target bit. Next state of both “S3” and “S4” will be the “S0”. From “S0”, if the user set the end command, the next state will be “S3”.

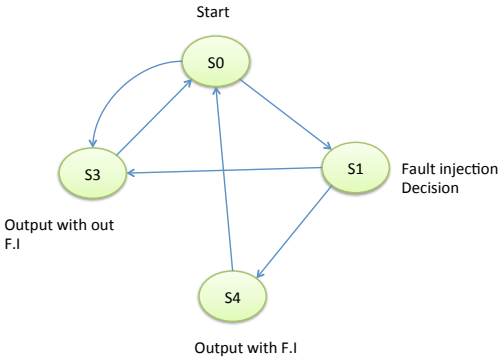

Fig. 4. State Diagram Graph of the FSM controlling the Fault Injection Block

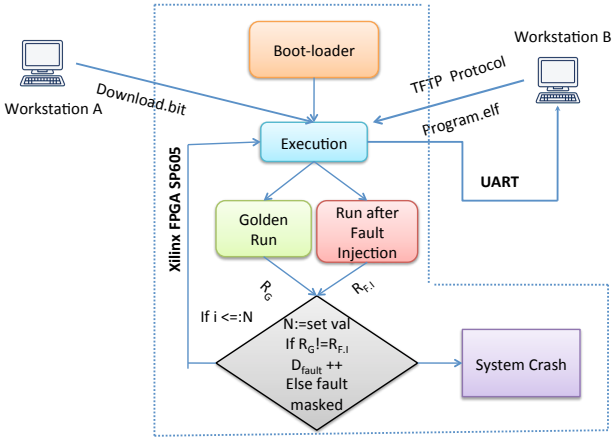

Fig. 5. Work flow diagram of the FPGA-based Fault Injector

The Fig. 5 sketches the full flowchart of a complete fault injection campaign. First of all the proposed fault injector is written in VHDL, synthesized by using the Xilinx ISE14.4[7]. The resulting bitstream is then download on the target board Xilinx Spartan 605[7].

The Workstation A, we designed a fault injector with amber core[6] where it is synthesized to generate the bit file. The download.bit file is downloaded to FPGA board via the JTAG port from Workstation A. From another Workstation B, the same board is connected using two different ways: Ethernet and Serial connection.

The target application corresponding to a given software C code is compiled to obtain the executable (.elf) file. The executable is downloaded on the Amber main memory implemented on the FGPA through the Ethernet connection. The Trivial File Transfer Protocol (TFTP) is exploited for the download. After that, the injection campaign automatically starts. First of all a golden run is done to store the expected application outputs. The, at each injection, a random address (i.e., the address can be a RAM, cache or register location) and mask are generated to inject the fault. After the execution of the faulty application, the system compares the obtained outputs with the goldenoutputs.The comparison result is classified into three categories:

- *Masked:* The software produces correct results. The faults is masked.
- *Silent Data Corruption:* The application outputs are different from the fault free outputs.
- *Detected:* The fault is detected by the application.
- *Crash/Unresponsive:* The application stops working or it never stops.

The Serial connection is used to monitor the Fault Injector behavior by using a HyperTerminal. The injections run up to anuser defined value N.

Finally, the Fig. 6 shows the real Lab setup implementing the flow above described.

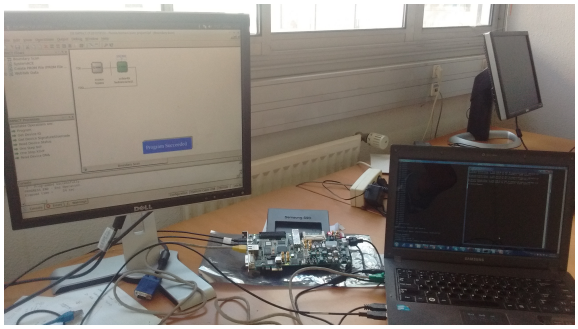

Fig. 6. Real experimental setup with FPGA

IV. Preliminary Results

In this section we present a set of preliminary results carried out by the proposed fault injector. The target application is a simple matrix multiplication program with

twoversions: (1) simple version, (2) duplicated version with a correction mechanism.The target application has been implemented in C programming language

The Table 1 gives the results obtained by the injector on the considered applications. For each application, we implement three different cache sizes. The goal was to see the impact of the cache size on the global system reliability (*i.e.,* the impact on the running application). For each sub column, we report the percentage of Silent Data Corruption. This is the only outcome since we target only the RAM containing the data segment of the application; therefore the instructions are not involved.

TABLE I. RESULTS

Application	**Silent Data Corruption%**		
	32 Kbytes	***64 Kbytes***	***128 Kbytes***
Standard	68.83	66.47	64.69
Duplicated	29.39	27.6	23.86

As we can see, increasing the cache size positively impacts the reliability of the whole system since the percentage of SDC decreases. It is also important to mention that the difference of SDC between the standard and the duplicated version is due to the capability to correct many of the injected faults. Finally, the whole injection campaigns were of 10k injections for each configuration. The required injection time is on the order of minutes.

V. CONCLUSIONS

In this work, we presented a scalable and flexible high performance fault injector based on the use of FPGA board. The proposed tool main aims at giving to the user the possibility to easily build its own fault injector in order to target the desired set of fault locations. We carried out a set of preliminary results carried out on the open source Amber project [6], that is a complete embedded system implemented on the Xilinx Spartan-6 SP605 [7] FPGA development board.

REFERENCES

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

[2] S. Borkar, T. Karnik, and V. De, "Design and reliability challenges in nanometer technologies," in *Proceedings of the 41st Annual Design Automation Conference*, ser. DAC '04, 2004, pp. 75–75.

[3] S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. Austin, "A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor," in *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 36, 2003, pp. 29–

[4] M. Kooli, A. Bosio, P. Benoit, and L. Torres, "Software testing and software fault injection," in *10th International Conference on Design & Technology of Integrated Systems in Nanoscale Era, DTIS 2015, Napoli, Italy, April 21-23, 2015*, 2015, pp. 1–6.

[5] R. Vadlamani, J. Zhao, W. Burleson, and R. Tessier, "Multicore Soft Error Rate Stabilization Using Adaptive Dual Modular Redundancy," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '10, 2010, pp. 27–32.

[6] http://opencores.org/amber

[7] http://www.xilinx.com/

[8] M. Kooli and G. Di Natale, "A survey on simulation-based fault injection tools for complex systems," in *Proceedings of the 9th International Conference on Design & Technology of Integrated Systems in Nanoscale Era, DTIS 2014, Santorini, Greece, May 6-8, 2014*, pp. 1–6.

[9] M. Nicolaidis, *Soft errors in modern electronic systems*. Springer Science & Business Media, 2010, vol. 41.

[10] J. Karlsson, P. Liden, P. Dahlgren, R. Johansson, and U. Gunneflo, "Using heavy-ion radiation to validate faulthandling mechanisms," *IEEE micro*, pp. 8–11, 1994.

[11] S. Duzellier and G. Berger, "Test facilities for see and dose testing," in *Radiation Effects on Embedded Systems*. Springer, 2007, pp. 201–232.

[12] R.Ecoffet,"In-flightanomaliesonelectronicdevices,"in*Radiation Effects on Embedded Systems*. Springer, 2007, pp. 31–68.

[13] R. Velazco, B. Martinet, and G. Auvert, "Laser injection of spot defects on integrated circuits," in *Test Symposium, 1992.(ATS'92), Proceedings., First Asian (Cat. No. TH0458-0)*. IEEE, 1992, pp. 158–163.

[14] P. Fouillat, V. Pouget, D. Lewis, S. Buchner, and D. McMorrow, "Investigation of single-event transients in fast integrated circuits with a pulsed laser," *International journal of high speed electronics and systems*, vol. 14, no. 02, pp. 327–339, 2004.

[15] F. Miller, N. Buard, T. Carrie`re, R. Dufayel, R. Gaillard, P. Poirot, J.-M. Palau, B. Sagnes, and P. Fouillat, "Effects of beam spot size on the correlation between laser and heavy ion seu testing," *IEEE transactions on nuclear science*, vol. 51, no. 6, pp. 3708–3715, 2004.

[16] D. Powell, E. Martins, J. Arlat, and Y. Crouzet, "Estimators for fault tolerance coverage evaluation," *Computers, IEEE Transactions on*, vol. 44, no. 2, pp. 261–274, 1995.

[17] J. Arlat, A. Costes, Y. Crouzet, J.-C. Laprie, and D. Powell, "Fault injection and dependability evaluation of fault-tolerant systems," *Computers, IEEE Transactions on*, vol. 42, no. 8, pp. 913–923, 1993.

[18] J. Pontes, N. Calazans, and P. Vivet, "An accurate single event effect digital design flow for reliable system level design," in *Proceedings of the Conference on Design, Automation and Test in Europe, DATE*. EDA Consortium, 2012, pp. 224–229.

[19] E. Jenn, J. Arlat, M. Rimen, J. Ohlsson, and J. Karlsson, "Fault injection into vhdl models: the mefisto tool," in *Fault-Tolerant Com- puting, 1994. FTCS-24. Digest of Papers., Twenty-Fourth International Symposium on*. IEEE, 1994, pp. 66–75.

[20] V. Sieh, O. Tschache, and F. Balbach, "Verify: Evaluation of reli- ability using vhdl-models with embedded fault descriptions," in *Fault-Tolerant Computing, 1997. FTCS-27. Digest of Papers., Twenty-Seventh Annual International Symposium on*. IEEE, 1997, pp. 32–36.

[21] M. Ebrahimi, A. Mohammadi, A. Ejlali, and S. G. Miremadi, "A fast, flexible, and easy-to-develop FPGA-based fault injection technique," *Microelectronics Reliability 2014*, vol. 54, no. 5, pp. 1000–1008.

[22] D. De Andre´s, J. C. Ruiz, D. Gil, and P. Gil, "Fault emulation for dependability evaluation of vlsi systems," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 16, no. 4, pp. 422–431, 2008.

[23] L. Sterpone and M. Violante, "A new partial reconfiguration- based fault-injection system to evaluate seu effects in sram-based fpgas," *Nuclear Science, IEEE Transactions on*, vol. 54, no. 4, pp. 965–970, 2007.

[24] S.-A. Hwang, J.-H. Hong, and C.-W. Wu, "Sequential circuit fault simulation using logic emulation," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 17, no. 8, pp. 724–736, 1998.

[25] P. Civera, L. Macchiarulo, M. Rebaudengo, M. S. Reorda, and M. Violante, "Exploiting circuit emulation for fast hardness eval- uation," *Nuclear Science, IEEE Transactions on*, vol. 48, no. 6, pp. 2210–2216, 2001.