Working Conditions-Aware Fault Injection Technique

Ihsen Alouani, Smail Niar, Mohamed Jemai, Fadi Kuradi, Mohamed Abid

Working-Conditions-Aware Fault Injection Technique

Ihsen Alouani[1] , Smail Niar[1], Mohamed Jemai[1], Fadi Kurdahi[2] and Mohamed Abid[3]

1 : University of Valenciennes et du Hainaut Cambrésis, 59300 Valenciennes - France.
2 : Center for Embedded Computer Systems, Univ. of California, Irvine, USA.
3 : CES, National Engineering School of Sfax, Tunisia.

Contact : ihsen.alouani@gmail.com, smail.niar@univ-valenciennes.fr

Abstract
With new integration rates, the circuits sensitivity to environmental and working conditions has increased dramatically. Thus, presenting reliable, less consuming energy and error resilient architectures is being one of the major problems to deal with. Besides, evaluating robustness and effectiveness of the proposed architectures is also an urgent need. In this paper, we present an extension of SimpleScalar simulation tool having the ability to inject faults in a given cache architecture. We tried to focus on transient errors which are due to cosmic rays (soft errors) or to voltage scaling and high temperatures. During the fault injection process, there is no software traps and the target application isn't modified. Transient errors are modeled by a bit flip. The bit flip may occur in any part of the cache arrays and also in the cache blocks' address. In this paper we present an overview on the existing fault injection techniques. Then, we propose a fuzzy system which models the relationship between working conditions (the supply voltage and temperature) and the error probability. The impact of the injected faults on the cache performance as well as the experimental results are also shown. The results show that the impact of faults depends also on the application itself.

Keywords: Dependability, fault tolerant, fault injection, voltage scaling.

1. Introduction

The electronic systems dependability is becoming a growing issue especially with new process technologies. Besides, evaluating the dependability of a processing architecture is a complex task. The complexity of this task increases with the electronic circuits' performance requirements in power consumption and rapidity. Modeling analytically the phenomena of occurring faults is very difficult because of the complexity and relative randomness of these phenomena. Fault Injection is defined by [1] as the dependability validation technique that is based on the realization of the controlled experiments where the observation of the system behavior in presence of faults, is explicitly induced by the deliberate introduction (injection) of faults into the system. It consists in the accomplishment of controlled experiments where the observation of the system's behavior in presence of faults is induced explicitly by the introduction (injection) of faults in the system. In the literature, there are four categories of fault injectors: Hardware-based, software-based, simulation-based and emulation-based fault injectors:
The hardware-based fault injection technique uses specific additional hardware in order to inject faults in the target system's hardware. Depending on the faults and their locations, hardware-implemented fault injection methods fall into two categories: [2]
1) Hardware fault injection with contact: The fault is injected using physical contact with the target system by introducing voltage or current pulses into the circuit.
2) Hardware fault injection without contact: The fault injector has no physical contact with the target system. Instead, an external source produces some physical phenomenon, such as heavy ion radiation and electromagnetic interference, causing spurious currents inside the target chip.

The software-based fault injection technique involves the modification of the software running on the tested system. Several kinds of faults may be injected: register and memory errors, dropped or replicated network packets and erroneous flags. [2]
The simulation-based fault injection technique is different from the two first ones as it involves the construction of a simulation model of the analyzed system architecture, including a detailed model of the used processor.
Finally, the fourth method is the emulation-based fault injection technique: In order to make use of simulation benefits and take into account the effects due to real working conditions. The target system is implemented onto FPGA. The synthesized circuit is connected to a computer which role is to control the fault injection triggers and display the results.
In our case, we opted for the simulation-based fault injection approach. The idea is to inject faults in a given architecture at the simulation level in order to measure its effectiveness and its level of accuracy. This method incurs a low cost and does not require any special-purpose hardware. These features make the fault injection process more flexible. In fact, any signal value can be easily corrupted and the results of the corruption are easily observable regardless of the location of the corrupted signal. This flexibility facilitates error modeling and injection. We extended SimpleScalar3.0 Mips processor simulator [3] in order to make a simulation environment equipped with a fault injector. Presently, our fault injection tool has been applied on different benchmarks of Mibench [4] in order to study the cache system behavior under aggressive working conditions. We note that the majority of works already done on fault injection were focusing on modeling faults, injecting them in hardware or software. Generally, choosing the time and the location in wich the fault injection is triggered is made either randomly or in a deterministic manner. The problem is that this choice may not correspond to the fault occurring phenomenon in a real process. For these reasons, we propose in this paper a fuzzy system that predicts the error probability in terms of supply voltage and temperature. The Fuzzy logic control is an alternative to traditional notions of logic. It has emerged as a profitable tool for the controlling and steering of systems and complex industrial processes, as well as for other expert systems and applications. As the nature of the studied phenomena is more fuzzy than deterministic, the fuzzy logic tool corresponds to the features of our problem.
This paper is organized as follows. The proposed fault injector and the proposed predictor of error probability are discussed in section 2. Section 3 shows the impact of injected faults on the cache system performance. Finally, section 4 concludes the paper and opens horizons to future work.

2. Fault Injection Process

A. The Proposed Fault Injector

In this paper, we are focusing on transient errors that are triggered by environmental conditions such as power-line fluctuation, electromagnetic interference, or cosmic radiations (soft errors). Generally, transient errors are modeled by a bit flipping even if it can cause a permanent stuck at 0 or stuck at 1 [2]. In terms of triggering time, the transient errors are generally considered as randomly occurring phenomena. The majority of related works in the literature dealing with this type of faults uses either deterministic or random fault injection. The deterministic method consists in injecting faults in preselected times before running. However, the random technique randomly carries out the injection during runtime. In our case, we are modeling transient errors using a bit flip in the cache arrays. The error may occur either in data bloc, tag or address bits. If a bit flip occurs in a data block, it leads automatically to an execution failure if the affected bloc is accessed after fault injection. However, in the case of either tag or address, it can produce an execution failure or simply produce a miss and increase the latency of the cache access. In this work, we propose a pseudo-random process of fault injector triggering. Our fault injection mechanism is based on fault rates of transient errors. These rates are deducted from the percentage of saved supply voltage and the working temperature by a proposed fuzzy system of error probability prediction. In addtion to the triggering time, the second issue our paper proposes to study is the location where errors has to be injected within the cache. In the literature, the fault injection location as well as the fault injection time, is either considered as random location, or deterministic choice, i.e. pre-defined locations [2]. In our case, we chose to inject faults in the same address of the cache access. This can be considered as “the worst case” methodology. In fact, choosing a given address to inject fault doesn’t lead necessarily and immediately to an error because the address may not be used immediately. However, if we inject a fault in the selected line address to access, it is more likely that this bloc would be needed in the

execution.

B. Error Probability Prediction System

The majority of works already done on fault injection were focusing on modeling faults, injecting them in hardware or software. They generally choose either random or deterministic methods to define time and location of the fault injection process. The problem is that this choice may not represent correctly how faults occur in reality. In fact, supposing that faults are occurring randomly may imprecisely model the phenomena behind faults taking place. Faults occur randomly in time, but they are not random phenomena in terms of working conditions. Thus, the whole fault injection tool may be far from reality and in a context of evaluating fault resilient architecture, it can lead to refusing an efficient architecture or accepting a weak one. Hence, the need to build a model that mimes the real phenomenon of occurring faults is being imperative.

The figure below represents the fault injection mechanism that we propose to design. The inputs are temperature and saved percentage of supply voltage which is the reduction in Vdd per cent. In fact, in a context of saving energy and reducing power consumption, reducing the supply voltage leads to increasing the cache failure probability. Besides, as shown in [6], the temperature of the circuit has an important impact on error probability. As there is a great difficulty in modeling analytically the phenomenon of fault occurring, we have though to use a fuzzy system that is able to integrate the interference between working conditions and describe the behavior of fault occurring mechanism in the reality. Fuzzy Logic was initiated in 1965 [7] by Lotfi Zadeh. Basically, Fuzzy Logic (FL) is a multivalued logic that allows intermediate values to be defined between conventional evaluations like true/false, yes/no, high/low…etc. Notions like rather tall or very fast can be formulated mathematically and processed by computers, in order to apply a more human-like way of thinking in the programming of computers [4].

Our proposed fuzzy system's role is to estimate the error probability value in terms of the chip working conditions and communicate it with the simulator.

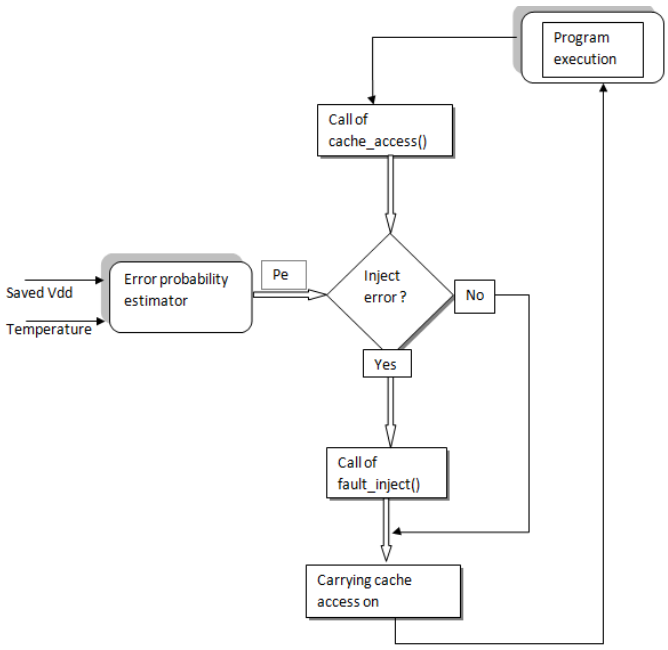

FIG. 1 – Diagram of fault injection process.

Given a set of rules describing the behavior of the error probability taken from [5], we have to build a fault injection approach that models the way error probability evolves while working conditions like

temperature and saved power are changing. So, we have to give a model that takes temperature and saved power as inputs and gives the probability of error as an output. In the proposed system, the modeling of the relationship between the probability of errors and the working conditions (Temperature and Voltage) is based on observed behavior of error probability under changed running parameters. Based on these observations we proposed a fuzzy logic system that transforms inputs from numbers to fuzzy sets admitting a number of membership functions. After that, basing on fuzzy rules, the system extracts the output using a given defuzzification method (centroïd in our case). Then the system construction is presented as follows (figure 2).

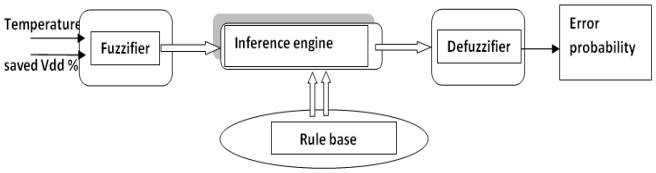

FIG. 2 – General form of the proposed fuzzy system.

In our case, we have two inputs ("temperature" and "saved supply voltage percentage") and one output ("error probability"). The range of temperature we have chosen is between 20°C and 120°C distributed on three fuzzy sets: mild, medium and aggressive. However, for the saved power percentage, we divide the range to 5 fuzzy sets: low, mild, medium, aggressive and destructive. The choice we made about the number of fuzzy sets can be explained by referring to [6] which shows that, below the nominal Vdd, the impact of the voltage scaling on cache failure probability is more important than the temperature one. Thus, while in our case we do not surpass the nominal supply voltage, the voltage scaling impact has to be more precisely described in the fuzzy system design. The output variable which is the error probability for the SRAM is placed in a range between 10^{-6} and 10^{-1} and this choice is chosen in harmony with the approximation for SRAM error probabilities maid in [9]. It is consisting of 5 fuzzy sets which are: low, mild, medium, high, aggressive, very aggressive and destructive. The rules we have implemented are shown in the table below.

Temperature	Saved power percentage	Error probability
Mild	Mild	Low
Medium	Mild	Low
Aggressive	Mild	Mild
Mild	Medium	Mild
Medium	Medium	Medium
Aggressive	Medium	Medium
Mild	High	Medium
Medium	High	High
Aggressive	high	Aggressive
Mild	Aggressive	High
Medium	Aggressive	High
Aggressive	Aggressive	Aggressive

TAB. 1 – Rules implemented in the fuzzy logic system.

We implemented the fuzzy inference system using Fuzzy Logic Toolbox of Matlab R2007a and we got the following variations of error probability in terms of temperature and saved Vdd percentage.

Figure 3 illustrates the exponential growth of the cache failure probability while reducing supply voltage. We suppose in our case that the frequency is maintained constant. However, in reality, changing cycle time leads to different probability of failure [6]. This means that our predictor targets a precised architecture with given performance and has to be changed with other cases.

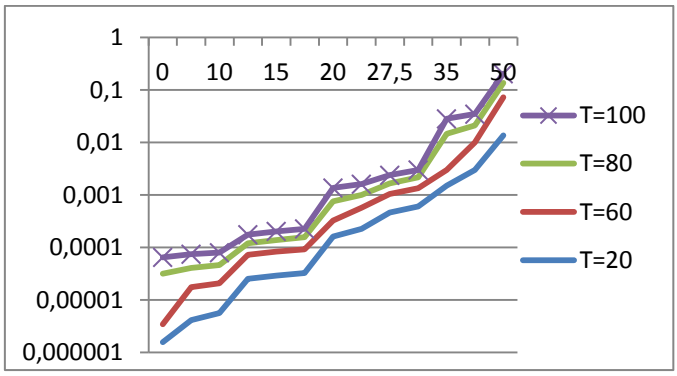

FIG. 3 – Error probability in cache in terms of saved supply voltage percentage and temperature. The range of error probabilities are obtained for 32-nm technology for a 32kB cache [9].

Increasing the percentage of the gain means a reduction in the supply voltage. Therefore, as shown in the figure 3, varying the saved Vdd percentage from 0% to 50% leads to multiplying the error probability by 10^4. We also note that increasing the running temperature increases the error probability.

3. The Impact of Injected Faults on The Cache Performance

In order to measure the impact of error rate on cache performance, we varied error probabilities in a range from 10^{-6} to 10^{-3} which is in accordance with [9]. We represented its impact on some cache performance as shown in figure 4 below.

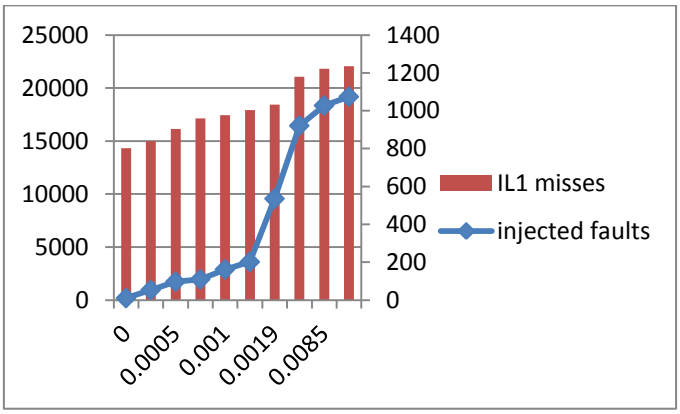

FIG. 4– The impact of the injected errors on the first level instruction cache (IL1). The probability varies from 0 to 0.01. The IL1 size is 16 KB. Basic-math benchmark (158M Instructions) was used.

From this figure we can notice the increase in instruction cache misses from 800 to more than 1200 misses while injecting faults. The increasing number of misses means an overhead in time and power which represents a degradation in cache performance. Figure 4 gives also the progress of the injected faults number while increasing the error probabilities.

After studying the impact of injecting errors on cache behavior in terms of data and misses, we modified the SimpleScalar based simulator sim-wattch in order to get information about the impact of injected errors on the cache power and energy consumption. Figure 4 represents the variation of instruction cache power consumption, and instruction per cycle (IPC) while increasing the error

probability in the Rijndael benchmark from Mibench. We note that injecting faults degrades the processor performance and increases the power consumption. This is due to the raise of the occurring misses' number. In fact, its direct impact is to decrease the executed instructions per cycle. This is due to the lost time in additional misses. In parallel with the IPC decrease, we note that the power consumption of the cache is increasing while rising fault injection probability.

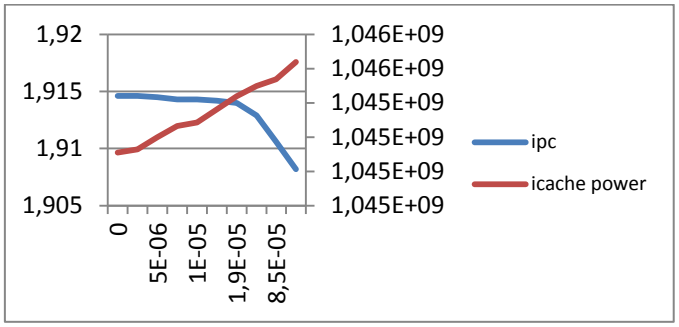

FIG. 5–Variation of IPC and the instruction cache power consumption for the Rijndael benchmark (39M instructions). The IL1 and DL1 size is set to 16KB.

The same behavior is recorded with Basic Math benchmark. In the figure 6, we represent the total power consumption per cycle as well as instruction per cycle evolution in terms of injected fault probability.

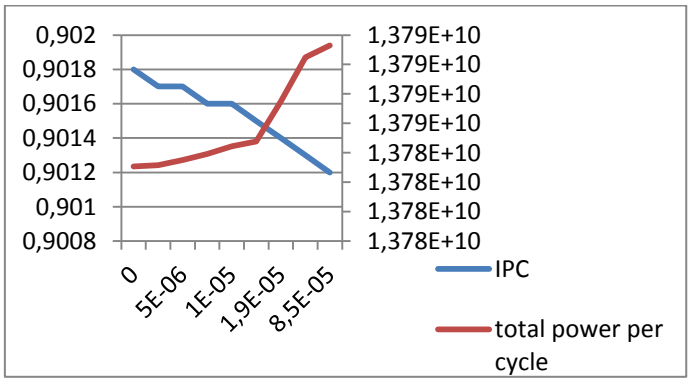

FIG. 6–Variation of the IPC and the instruction cache power consumption for the Rijndael benchmark (39M instructions). The IL1 and DL1 size is set to 16KB.

From the figure above we validate the remarks done for Rijndael application and we can note the increase of the total power consumption with the rising number of injected faults. In fact, the additional number of cache misses creates a need to additional power consumed to access next level caches.

After studying the impact of injected faults on power consumption and processor performance, we tried to show the impact of the cache size on the power consumption while the fault injection probability is maintained. The following figure shows the evolution of cache power consumption and IPC in terms of the cache size. We fixed the error probability and varied the first level cache size from 8 kB to 128kB. We note that the IPC is increasing with cache size and this is explained by the decrease of the misses number. At the same time, spite of the diminution of the cache misses, the power consumption increased. In fact, raising the cache size leads to increasing the cache power consumption. So, a designer has to look for the appropriate tradeoffs between reducing power consumption which result in raising the error probability and increasing the cache size to deal with performance degradation which result in increasing power consumption.

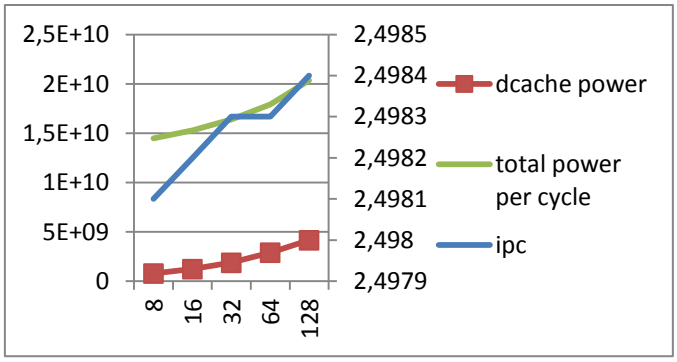

FIG. 7–Variation of data cache power, total power per cycle and IPC in terms of cache size (from 8kB to 128kB) 2-way associative cache. The error probability is fixed and it's equal to 10^{-4}

4. Conclusion

In this paper, we persent a fault injection environment at the simulation level. It is implemented in C language as an extension of SimpleScalar simulator. It simulates transient faults occurring in a cache architecture in runtime. The work focuses on transient faults because of their increasing impact especially with new transistors integration rates and process technologies. The injected errors are injected during cache access and modeled by a bit flip in the cache arrays. The fault injection process is random in time. However, it is controllable via an error probability that is estimated using a proposed fuzzy logic predictor. The latest, which is the main contribution of this paper, is an inference system that predicts the error probabilitiy using both estimated circuit temperature and saved supply voltage percentage. The current version can be improved in several ways. We are currently working on a possible extension aiming to integrate fault tolerant architectures and implementing a fault injection process in the pipeline stages and the processor's registers. We also intend to generalize the approach by integrating the running frequency as an input to the error probability predictor.

Bibliographie

1. A. Bonso, P. Prinetto, "Fault Injection Techniques and Tools for Embedded Systems Reliability Evaluation".
2. Haissam Ziade , Rafic Ayoubi, and Raoul Velazco , "A Survey on Fault Injection Techniques "The International Arab Journal of Information Technology, Vol. 1, No. 2, July 2004.
3. T. Austin, E. Larson, and D. Ernst, "SimpleScalar: an infrastructure for computer system modeling," IEEE Computer, vol. 35, 2002.
4. M. Guthaus, J. Ringenberg, et al., "A free, commercially representative embedded benchmark suite," in Proc. IEEE WWC 2001.
5. G. Swift et al., "Single-event upset susceptibility testing of the Xilinx Virtex II FPGA," in MAPLD, 2002.
6. A. Khajeh, A. Gupta, N. Dutt, F. Kurdahi, A. Eltawil, K. Khouri, and M. Abadir, "TRAM: A Tool for Temperature and Reliability".
7. L.A. Zadeh, "Fuzzy Sets, Information and Control", 1965.
8. L.A. Zadeh, "Making computers think like people," IEEE. Spectrum, 8/1984.
9. A. Sampson, W. Dietl, E. Fortuna, D Gnanapragasam, L. Ceze, D. Grossman, "EnerJ: Approximate Data Types for Safe and General Low-Power Computation".