

Soft Error Effect and Register Criticality Evaluations: Past, Present and Future

Régis Leveugle, Laurence Pierre, Paolo Maistri, Renaud Clavel
TIMA Laboratory (Grenoble INP, UJF, CNRS)
Grenoble, France
{Regis.Leveugle, Laurence.Pierre, Paolo.Maistri, Renaud.Clavel}@imag.fr

Abstract—Evaluating the robustness of digital circuits with respect to soft errors has become an important part of the design flow for many applications. The identification and hardening of the most critical registers is often necessary, while limiting the induced overheads. At the same time, the complexity of the circuits and the time to market pressure continue increasing. In this context, we briefly review the evolution of the techniques allowing a designer to evaluate the robustness early in the design flow and to identify the most critical elements. We discuss some limitations, then we propose new approaches to improve this process and we show some preliminary results.

I. INTRODUCTION

The dependability analysis of modern embedded systems is becoming a major concern for designers. Particle strikes, electromagnetic interferences or other signal integrity problems result in soft errors, i.e. logic errors in some registers while no damage exists in the circuit. Soft errors can be due to spurious commutations in flip-flops (SEUs – Single Event Upsets – are a well known example and correspond to single bit-flips) or to erroneous signals latched after propagation in the combinatorial parts (SETs – Single Event Transients). In the following, we will focus on evaluating the consequences of soft errors with respect to the application, no matter their origin. Also, we will focus on soft errors in random logic parts, i.e. in flip-flops (soft errors in memory blocks are often easier to mitigate). Only synchronous digital circuits will be considered. However, no assumption will be made on the functionality of the circuit.

The designer goal is in general to accurately evaluate the failure rate (FIT) as perceived by the system user, then protect the most critical elements when necessary. Since not all soft errors lead to system failure, derating factors have to be applied to the raw Soft Error Rate (SER). Analyzing the application-level robustness with respect to soft errors is usually carried out at design time by means of fault injections, either by simulation or by hardware emulation. The main limitation is the length of such a fault injection campaign. The circuit can contain hundreds of thousands of flip-flops and the application computations can last millions of cycles. A complex error model not limited to single bit flips easily leads to millions of possible error configurations. In consequence a complete fault injection campaign is often not affordable and the number of experiments has to be reduced, even when each experiment is accelerated by using for example hardware emulation.

Several approaches can be used to reduce the experimental time. One approach is to randomly select a given subset of the possible errors. Another approach consists in analytically

identifying errors that will be harmless and whose injection is therefore useless; this is often called fault pruning. Considering the various outcomes that may be expected from a fault injection campaign, it appears that not all optimization approaches are well suited in all cases. Also, general limitations still subsist.

The first part of this paper, and especially section III, aims at discussing the progress but also remaining limitations of current practice. In particular, limitations are discussed with respect to the possible expected outcomes, summarized in section II. In the second part of the paper, i.e. section IV, two new approaches based on formal modelling (Petri nets) and extended formal verification techniques are proposed to overcome some of the limitations in the near future.

II. EXPECTED EXPERIMENTAL OUTCOMES

All early robustness evaluations do not seek the same kind of answer. This point is important, because some techniques may be very efficient with respect to some outcomes, but completely inadequate with respect to others. The main types of outcomes that can be expected are:

- classification of faults/errors – Injected faults or errors are classified with respect to a list of potential effects defined by the designer. These effects may include application failure modes, error tolerance or detection (when mechanisms exist), or just nothing (silent errors, without any consequence on the application). Such a classification can be used to evaluate the intrinsic robustness level of a circuit and/or to validate the protection mechanisms implemented in the circuit.

- quantification of derating factors – Timing and architectural derating factors can be evaluated using fault injection campaigns. In that case, the classification is generally defined in order to compute the percentage of errors propagating to output errors. Other approaches (analytical or simulation-based) have also been published for various types of blocks or circuits (e.g. [1, 2]) but are out of the scope of this paper. In particular, approaches targeting high performance microprocessors are often not applicable to other circuits, due to the lack of some information or the unavailability of architectural simulators.

- identification of error propagation paths – Classifying the errors does not give any insight into how they propagate within the circuit from their origin to the recorded effect. In order to identify the best positions in the circuit where protection must be added it is necessary to store more detailed data during the experiments [3].

- identification of critical locations – When selective (or "pragmatic") hardening is the goal, the identification of error propagation paths allows the designer to pinpoint efficient locations where propagations can be stopped. Another possibility is to avoid using sensitive cells to implement registers that may be the origin of the most critical consequences (e.g. using specific hardened flip-flops on those locations). In that case, it is necessary to order the list of flip-flops with respect to the probability that an error in them will result in a critical event for the application. This will be called flip-flop or register grading.

- proof of a given set of properties – In that case, an expected system property and/or the efficiency of detection or tolerance mechanisms must be guaranteed for all errors in the expected error set.

In the following, we will discuss the pertinence or limitation of current techniques with respect to these five main outcomes.

III. FROM PAST TO PRESENT

Several types of approaches have been used to evaluate at design time the dependability level of a circuit. We will only focus here on approaches that can be used to analyze early in the design flow (on behavioral descriptions) the functional effects of soft error propagations. Other practices used to analyze gate-level, electrical-level or even physical-level descriptions will not be discussed. Also, all techniques used to qualify a circuit once a prototype has been manufactured will not be mentioned.

In the last decades, a lot of work has been done to propose fault injection techniques that can be applied early in the design process. The first approach consisted in injecting the faults in high level descriptions (most often, VHDL models) of the circuit or system. The pioneer work in [4], and then [5] or [6], considers the injection of several types of faults in the VHDL model of a circuit at several abstraction levels and using various techniques based either on the modification of the initial VHDL description or on the use of simulation primitives. Such techniques have then been developed in various works; the modifications in the initial circuit description can be either behavioral, generating a "mutant" [3], or structural, adding a "saboteur" block [4] or "saboteur" process [7]. Simulations are used to evaluate the impact of the faults on the circuit behavior. Techniques were proposed to optimize such simulation-based campaigns, by pruning the transient faults having no functional consequences [8, 9]. However, the main drawback related to the use of simulations remains the huge amount of time required to run the experiments when many faults have to be injected in a complex circuit.

To cope with the slow speed imposed by simulation, it has been proposed to take advantage of hardware prototyping [10-13]. However the increase in speed highly depends on the experimental set-up and on the communication throughput required between the emulation platform and the host computer [14]. Emulated fault injection can use either circuit instrumentation [3, 15, 16] or run-time reconfiguration of a reconfigurable platform [17-20]. In the first case, the circuit is modified before implementation on the reconfigurable hardware, so that the errors in the selected model can be forced during the application run. In the second case, errors are forced by partially reconfiguring the prototype during the execution.

Instrumentation has been used in many implementations and experiments that will not be listed here. Basically, the various approaches can be grouped in three categories. The circuit prototype can be fully controlled from an external computer [15], or on the opposite all injection controls can also be implemented in hardware; this is called "autonomous emulation" [16]. An intermediate approach consists in using a SoPC in order to take advantage of an integrated processor core to obtain a good trade-off between speed and flexibility [21].

With respect to the various types of outcomes, simulation is of course the most flexible and general approach. It has also a very big advantage when complex classification is required. As an example, the failure of the system may be related to the incapacity of globally providing a given number of results in a given time, without functional constraints on the order of the results or the specific time at which they are issued. In such a situation, simulation-based functional verification environments may be very efficient in determining whether the expected behavior is met or not. Using emulation implies a complex post-processing of the execution traces that may considerably reduce the speed advantage. On the opposite, if simple assertions have to be monitored, emulation can lead to several orders of magnitude improvements in experimental time when doing classification, or derating factor estimations. Autonomous emulation can in particular achieve very high speed for classification; it has however strong limitations for other expected outcomes, e.g. propagation path analysis that requires storing a large amount of data.

In all cases, the main limitation remains related to the experimental time that often makes exhaustive campaigns unfeasible in a time compatible with the design constraints. This leads in most cases to perform only partial analyses based on a randomly (and often arbitrary) selected set of faults or errors. Such a statistical fault injection (SFI) has been very extensively used in the literature. Recently, an approach has been proposed to quantify the error on the results with a given confidence level, or conversely to evaluate the number of injections to perform in order to achieve a given error/confidence level [22]. SFI can then be very useful in doing quick classification or derating factor estimations. Unfortunately, this approach does not address all possible expected outcomes of a fault injection campaign. In the case of a large number of errors and workload cycles, only a very small proportion of the registers are actually perturbed at a few cycles. This means that such results cannot help in identifying the most critical registers or clock cycles. Flip-flop grading remains possible when the random selection is only used to reduce the number of injection cycles in each flip-flop; however, in that case, the efficiency of SFI is noticeably reduced and the required number of experiments remains very large. Error propagation paths are also only partially exercised, so decisions on the best hardening positions are hard to make. Finally, SFI cannot guarantee that a given property always holds; this can at best be assessed with a given margin of error.

Associating emulation with fault pruning or fault collapsing [23] can further reduce the experimental time without decreasing the quality of the results. Since fault pruning identifies faults (or errors) that will not have any consequence on the application, fault injection campaigns can be shortened but the pruned faults can still be taken into account when computing the classification statistics, derating factor

estimations or register grading. Fault pruning has also no impact on error propagation path analysis and do not preclude property proofs. Unfortunately, available fault pruning techniques often do not allow a sufficient reduction in experimental time. In particular, they are only efficient on some types of architectures. They often rely on the notion of variable living time [24]. This notion can be efficiently used when pruning faults in processor instruction and data memories. On the opposite, aliveness is very difficult to evaluate for flip-flops, especially in parallel architectures because registers are written and read very often (possibly at each clock cycle) with no direct correspondence with their actual role in the computation of the final result. In this case, the classical definition of living times is not sufficient to actually compute the criticality of the data in the registers. New fault pruning techniques must therefore be developed; one will be proposed and illustrated in section IV.A.

Although improving fault pruning can help in reducing the gap between available experimental time and exhaustive fault injections, other approaches have to be considered to more specifically tackle some of the expected outcomes. Using formal verification techniques has been recently proposed [25, 26]. Such techniques may help in proving that properties remain valid even in case of errors, or in evaluating the percentage of cases in which the property holds, without simulating or emulating the circuit. Of course, this requires extending the current formal verification methods that do not assume any soft error occurrence. Also, efficient approaches have to be worked out to avoid the explosion of the processing time required to make the proof. Studies are ongoing based on theorem proving as well as model-checking techniques. Preliminary results using theorem proving are summarized in section IV.B.

Other approaches are currently being studied, including Register Transfer Level probabilistic propagation analyses, but will not be discussed in this paper.

IV. TOWARDS FUTURE

A. Improved fault pruning, towards propagation models

As previously mentioned, parallel architectures are a difficult case for usual fault pruning techniques. More generally, all registers in a circuit that may latch useless data at some cycles will reduce the efficiency of the fault pruning. Such a case is quite frequent for example in pipelined architectures, in which some pipeline flip-flops may contain meaningless bits at some processing steps, but will be clocked because of the architecture regularity. Identifying the real living time of data in a register therefore not only implies identifying the writing and reading times but also requires analyzing the final use of the data.

We therefore developed a new dynamic fault pruning approach, taking advantage of a formal model of the system under evaluation. For our experiments, we have chosen to use Timed Petri Nets to model the circuit. However, some other models could lead to similar results. The main advantage of such a model is its ability to identify, at each cycle, the places in which a value is currently propagating. Registers are modeled by such places and it is therefore possible to monitor the propagation of the useful (symbolic) data in the system registers, independently of the active clock edges leading to

write a new value in these registers. Registers associated at some execution cycles to empty places are removed from the potential error location list for the identified clock cycles.

We only focus here on identifying, from a functional point of view, the register idle cycles. The specific case of redundant registers in for example a TMR architecture may lead to extend the fault pruning but is a subject for further work.

An experiment has been carried out on an implementation of the Advanced Encryption Standard, protected against soft errors and previously evaluated by fault injections [27]. The algorithm is iterative, made of several similar rounds, after a loading phase and before a termination phase. The implemented design resembles a systolic architecture, but the computation is not as regular and all registers are not used in the same way depending on the execution stage. Most registers are read and written at each clock cycle making classical fault pruning techniques almost useless. However, only part of the data processed in the systolic core have an impact on the data selected as results by the controller.

TABLE I. SINGLE-BYTE FAULT INJECTIONS IN ONE AES FULL ROUND.

Injection	Result Class				Total Faults
	Silent	Undet	False Positive	Detected	
In all cycles	254	508	256	512	1530
In live cycles only	0	508	0	512	1020

A fault injection campaign has been performed on the basis of the pruned error location list and compared with the previous exhaustive fault injection campaign. Data are summarized in Table I for a whole single round of AES. Since the specific architecture uses an error detecting scheme based on parity code, the fault injection may lead to four different outcomes: the fault is silent (no effect on the application), undetected, detected, or a false detection alarm is raised. Fault injections lead with the reduced error list to the same number of undetected and detected faults; at the same time, no case of silent fault or false positive is recorded. The overall number of experiments is thus significantly reduced (by 33% in this case), without reducing the quality of the result.

Furthermore, avoiding false positives is very important to reduce the experimental times. As a matter of fact, discriminating a false positive in a classical fault injection process requires running the workload until the end even in case of error detection, to compare the final result with the expected one. It is therefore not possible to stop the experiment just after detection, thus a noticeable experiment duration increase. If errors leading to false positives can be identified a priori, only silent errors due to functional masking will require running the experiments until the end of the workload, further reducing the experimental times. Although it is illustrated here on classification, this technique may be used to accelerate fault injection campaigns for the five types of outcomes summarized in section II.

For this case study, the model was generated by using the non-commercial simulation tool HPetriSim. This software allows easy design of PN models, which can then be simulated by the tool itself to analyze their behaviour. It was chosen out of several comparable tools because it provides the (few) required functionalities, it is easy to learn, and because of its operating environment; however, other tools can be used as well. The AES model was manually generated in few hours, including the time needed to learn the tool and to improve the layout of the model for better readability. This model is illustrated in Figure 1. Once the model was defined, the time needed to run the simulation and take note of the utilisation rate of the functional units was less than a minute, while avoiding 33% of the injections. Of course, in the future, dealing with more complex models would require developing a more automatic analysis. Also, most time was spent for this case study in tuning the model of the control part of the design. In the future, the model generation may be part of the default design flow and used for formal verification of the architecture. The PN model may also be generated by an automatic tool performing the translation from HDL descriptions, either at RT or higher level, e.g. Transaction level SystemC descriptions for early system analysis, or on the opposite at gate level. From RTL descriptions, a tool such as Vsyml [28] could be used to obtain the transfer function (transition and output functions) of the circuit. Boolean abstraction can be used to obtain the same type of description from netlists. The global flow we are currently developing is illustrated in Figure 2.

Another perspective of this approach is to go beyond fault pruning and to fully exploit the generated model in order to analyze the error propagations and to identify the most critical registers. Work is on-going, based on Colored Timed Petri Nets.

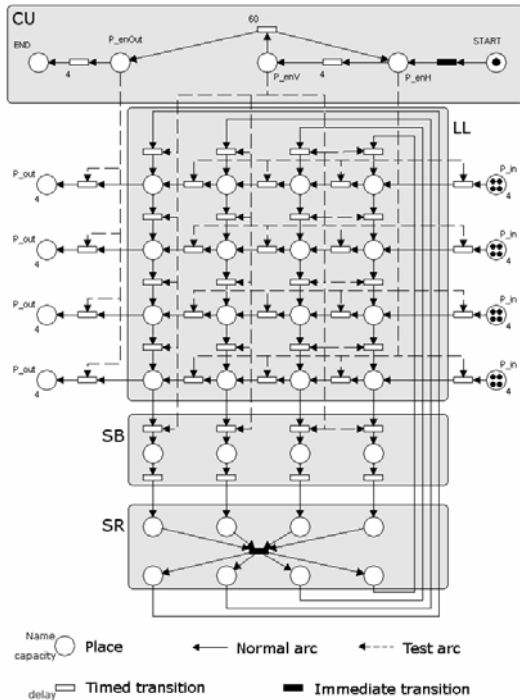


Figure 1. Petri Net of the AES data path (CU=simplified controller; LL=linear layer; SB=SubBytes; SR= ShiftRows).

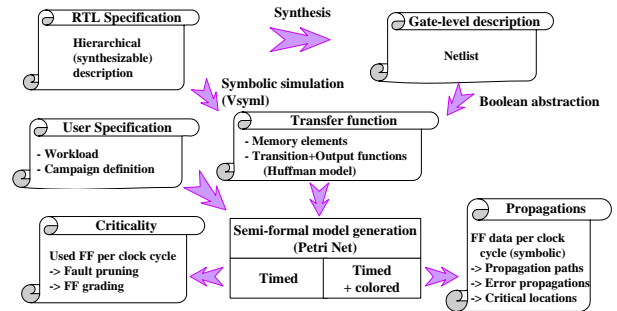


Figure 2. Evaluated flow using formal models.

B. Application of theorem proving techniques

As previously mentioned, recent papers describe preliminary solutions to apply formal verification methods in the context of dependability evaluation. They consider various problems, and most of them make use of model checking or symbolic simulation techniques. The approach of [26] focuses on *measuring the quality of fault-tolerant designs* and works by comparing fault-injected models with a golden model, all obtained by symbolic simulation for a given number of cycles. The model checker SMV is used in [29] to *identify latches that must be protected* in an arbitrary circuit: latches are fault-injected one at a time and SMV checks whether the formal specification of the original circuit still holds in each case. The goal of [30] is to *analyze the effects of transient faults*, using both symbolic simulation and model checking; fault injection is characterized by modifying the premises of the expected properties to picture the faults, and counter examples generated by the model checker are used to interpret their effects. The purpose of [31] is the *validation of mechanisms implemented in software for handling transient hardware faults*: bit-flips in data memory locations are emulated by manipulating the variables in the program, and fault injection is characterized by specific symbolic execution rules.

All these approaches are interesting but they share the same drawback: soft errors are enumerated and processed separately by applying the same procedure to all of them. Our aim is to provide solutions that avoid duplicating similar verifications for each error individually. To that goal, we investigated the use of the ACL2 theorem prover [32] which is a highly automated induction-based tool that supports first-order logic without quantifiers.

Theorem provers in general, and ACL2 in particular, have already been used in the context of fault-tolerant systems, more precisely for the verification of fault-tolerant protocols in distributed systems in which faulty processors may send conflicting information (e.g. Byzantine agreement protocols [33]). However to our knowledge, this is the first time results are reported regarding the use of ACL2 in the framework of hardware dependability analysis.

Principles. We consider synchronous circuits described in VHDL at the RT level and we target the *verification of reliability properties in presence of transient faults*. Using the

specialized tool previously mentioned [28], we parse the VHDL code and we get the transition and output functions (for a Mealy machine):

$$\begin{aligned}\delta &: I \times S \rightarrow S \\ \lambda &: I \times S \rightarrow O\end{aligned}$$

where I , O and S refer to the sets of inputs, outputs, and states (memory elements).

We formalize in ACL2 the fault model that corresponds for the on-going experiments to the presence of a single or multiple-bit error in a single register of the circuit. Thanks to the expressiveness of the ACL2 logic, we can give a kind of meta-characterization of the *fault injection function* f as the conjunction of the following properties:

- f takes as parameter a state s and returns a state $f(s)$
- $f(s)$ is different from s (injection is actual)
- there is only one memorizing element which differs from s to $f(s)$

Then, considering for instance the case of a device that has a property of auto-correction in one clock cycle in the presence of single faults (e.g. a TMR architecture), we can verify theorems such as the one below:

For any initial error-free state S_0 , if a fault is injected after n clock cycles ($n > 0$) then it will be corrected one clock cycle later i.e., the resulting state will be equivalent to the resulting state without fault injection:

$$\delta(i, f(\delta^n(i, S_0))) = \delta(i, \delta^n(i, S_0))$$

where i is an input sequence, and i is the current input.

The proof of such a theorem in ACL2 takes few seconds, depending on the complexity of the device.

Hierarchical decomposition. Complex hardware systems are typically described hierarchically as the interconnection of simpler components. We can take advantage of this hierarchical construction to perform hierarchical verifications, thus considerably improving the efficiency of the method. To that goal, we reason as follows with a component C_1 enclosed in a component C_2 :

From the VHDL description of component C_1 , we know the sets I_1 , O_1 , S_1 (deduced from the input/output ports and local signals declarations), and the transition and output functions

$$\begin{aligned}\delta_1 &: I_1 \times S_1 \rightarrow S_1 \\ \lambda_1 &: I_1 \times S_1 \rightarrow O_1\end{aligned}$$

For this component, we also have an error model specified by a function f_1 . Using all these characteristics, and locally to component C_1 , we determine:

- a predicate Sp_1 which is the state recognizer for C_1 i.e., $Sp_1(s) = \text{true} \Leftrightarrow s \in S_1$
- a predicate $Sreach_1$ which is the recognizer for the reachable (error-free) states of C_1 . For instance, in the case of a TMR architecture, reachable states are such that the contents of the three registers are identical
- a set \mathcal{P}_1 of fault-tolerance properties (theorems) for C_1

The definitions of the functions are local to C_1 . The outside world, in particular component C_2 , only knows the existence of δ_1 , λ_1 , Sp_1 , $Sreach_1$ and f_1 , and can use theorems \mathcal{P}_1 to infer

other properties. Component C_2 is characterized by similar constituents, and its properties \mathcal{P}_2 are deduced from \mathcal{P}_1 (and possibly from the properties of all other components contained in C_2). CPU times are significantly optimized through hierarchical proofs.

Example: consider the case of an ATM (Automated Teller Machine) that encloses three registers

- Rcode that memorizes the card code,
- Rin that contains the code input by the user, to be compared to the valid code,
- Rn that stores the current number of attempts,

if all the registers (component C_1) are implemented as TMR registers, we can prove various properties about the fault injection function f_2 for the ATM (component C_2), among them:

- $f_2 : S_2 \rightarrow S_2$
- $\forall s, f_2(s) \neq s$
- $Sreach_2(s) \Rightarrow \delta_2(i, f_2(s)) = \delta_2(i, s)$
- $Sreach_2(s) \Rightarrow start_op(i, f_2(s)) = start_op(i, s)$
where $start_op$ is the function associated with the output that triggers banking operations.

The complete set of theorems is proved in 10.58 seconds on an Intel Core2 Duo (3.0 GHz) under Linux.

If the registers are instantiated as registers that are only equipped with an error detection mechanism, we have other properties, for instance:

- $f_2 : S_2 \rightarrow S_2$
- $\forall s, f_2(s) \neq s$
- $Sreach_2(s) \Rightarrow \neg start_op(i, f_2(s))$
i.e., banking operations cannot start if an error occurs
- $Sreach_2(s) \Rightarrow e_detect(f_2(s))$
i.e., errors are detected.

The complete set of properties is proved in 2.34 seconds. CPU times are independent of the size of the registers (which are seen as integers) and of the maximum number of attempts (which is a *generic* of the VHDL description and remains generic in the proofs). Other results are obtained if other solutions are chosen (for instance only Rcode and Rin have a TMR architecture).

We are currently improving the methodology to deal with other kinds of fault models/properties, not limited to circuits with fault masking architectures or error detection mechanisms.

V. CONCLUSION

Several new approaches are under study in order to improve the efficiency of the current dependability evaluation techniques. Although a lot of progress has been done during the last decades, noticeable limitations remain and have been discussed in this paper. More powerful (and costly) emulation machines are not a sufficient answer to the increasing complexity of the analyses to be performed. Among the approaches currently under consideration, two examples have been summarized and some examples are shown. We expect that several complementary approaches, not limited to fault injection techniques, will be used in the future to meet all the designer expectations with respect to soft error effect evaluations.

ACKNOWLEDGMENT

The on-going work is partially supported by the French Government in the frame of the ASTER project (Minalogic) and by the French National Research Agency ANR in the frame of the FME3 project (ANR-07-SESU-006). Régis Leveugle is also supported by a CISCO Research Award.

REFERENCES

- [1] S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, T. Austin, "Measuring architectural vulnerability factors", *IEEE MICRO*, vol. 23, no. 6, Nov.-Dec. 2003, pp. 70-75
- [2] X. Li, S. V. Adve, P. Bose, J. A. Rivers, "SoftArch: an architecture-level tool for modeling and analyzing soft errors", 2005 International Conference on Dependable Systems and Networks (DSN), 2005, pp. 496-505
- [3] R. Leveugle, K. Hadjiat, "Multi-level fault injections in VHDL descriptions: alternative approaches and experiments", *Journal of Electronic Testing: Theory and Applications*, vol. 19, no. 5, Oct. 2003, pp. 559-575
- [4] E. Jenn, J. Arlat, M. Rimen, J. Ohlsson, J. Karlsson, "Fault injection into VHDL models: the MEFISTO tool", 24th Symposium on Fault-Tolerant Computing (FTCS), 1994, pp. 66-75
- [5] S. Svensson, J. Karlsson, "Dependability evaluation of the THOR microprocessor using simulation-based fault injection", Technical report n° 295, Chalmers University of Technology, Department of Computer Engineering, Sweden, November 1997
- [6] J. Boué, P. Pétilion, Y. Crouzet, "MEFISTO-L: a VHDL-based fault injection tool for the experimental assessment of fault tolerance", 28th Symposium on Fault-Tolerant Computing (FTCS), 1998, pp. 168-173
- [7] G. C. Cardarilli, F. Kaddour, A. Leandri, M. Ottavi, S. Pontarelli, R. Velazco, "Bit-flip injection in processor-based architectures: a case study", 8th IEEE International On-Line Testing workshop, Isle of Bendor, France, July 8-10, 2002, pp. 117-127
- [8] B. Parrotta, M. Rebaudengo, M. Sonza-Reorda, M. Violante, "New techniques for accelerating fault injection in VHDL descriptions", 6th IEEE Int. On-Line Testing workshop, Palma de Mallorca, Spain, July 3-5, 2000, pp. 61-66
- [9] L. Berrojo, I. Gonzalez, F. Corno, M. Sonza-Reorda, G. Squillero, L. Entrena, C. Lopez, "New techniques for speeding up fault-injection campaigns", Design, Automation and Test in Europe Conference (DATE), March 4-8, 2002, pp. 847-852
- [10] R. Leveugle, "Behavior modeling of faulty complex VLSIs: why and how?", The Baltic Electronics Conference, Tallinn, Estonia, October 7-9, 1998, pp. 191-194
- [11] R. Leveugle, "Towards modeling for dependability of complex integrated circuits", 5th IEEE Int. On-Line Testing workshop, Rhodes, Greece, July 5-7, 1999, pp. 194-198
- [12] J. Abke, E. Böhl, C. Henno, "Emulation based real time testing of automotive applications", 4th IEEE International On-Line Testing workshop, Capri, Italy, July 6-8, 1998, pp. 28-31
- [13] E. Böhl, W. Harter, M. Trunzer, "Real time effect testing of processor faults", 5th IEEE International On-Line Testing workshop, Rhodes, Greece, July 5-7, 1999, pp. 39-43
- [14] R. Leveugle, "A low-cost hardware approach to dependability validation of IPs", "The IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, San Francisco, California, USA, October 24-26, 2001", IEEE Computer Society Press, Los Alamitos, California, 2001, pp. 242-249
- [15] P. Civera, L. Macchiarulo, M. Rebaudengo, M. Sonza Reorda, M. Violante, "Exploiting FPGA for accelerating fault injection experiments", 7th IEEE International On-Line Testing workshop, Taormina, Italy, July 9-11, 2001, pp. 9-13
- [16] M. G. Valderas, M. P. Garcia, R. F. Cardenal, C. Lopez Ongil, and L. Entrena, "Advanced Simulation and Emulation Techniques for Fault Injection", IEEE International Symposium on Industrial Electronics (ISIE 2007), IEEE Computer Society, 2007, pp. 3339-3344
- [17] L. Antoni, R. Leveugle, B. Fehér, "Using run-time reconfiguration for fault injection in hardware prototypes", IEEE Int. Symposium on Defect and Fault Tolerance in VLSI Systems, 2000, pp. 405-413
- [18] L. Antoni, R. Leveugle, B. Fehér, "Using run-time reconfiguration for fault injection applications", IEEE Transactions on Instrumentation and Measurement, vol. 52, no. 5, October 2003, pp. 1468-1473
- [19] D. de Andres, J. C. Ruiz, D. Gil, P. Gil, "Run-time reconfiguration for emulating transient faults in VLSI systems", 2006 International Conference on Dependable Systems and Networks (DSN), 2006, pp. 291-300
- [20] L. Sterpone, M. Violante, "A New Partial Reconfiguration-Based Fault-Injection System to Evaluate SEU Effects in SRAM-Based FPGAs", IEEE Transactions on Nuclear Science, vol. 54, issue 4, August 2007, pp. 965-970
- [21] P. Vanhauwaert, R. Leveugle, P. Roche, "A flexible SoPC-based fault injection environment", 9th IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems (DDECS), Prague, Czech Republic, April 18-21, 2006, pp. 192-197
- [22] R. Leveugle, A. Calvez, P. Vanhauwaert, P. Maistri, "Statistical fault injection: how much is sufficient?", 2nd IFIP International Workshop on Dependable Circuit Design (DECIDE), Playa del Carmen, Mexico, November 27-29, 2008
- [23] A. Benso, M. Rebaudengo, L. Impagliazzo, P. Marmo, "Fault-list collapsing for fault-injection experiments", IEEE Reliability and Maintainability Symposium, 1998, pp. 383 - 388
- [24] L. Berrojo, I. González, F. Corno, M. Sonza Reorda, G. Squillero, L. Entrena, and C. López Ongil, "An Industrial Environment for High-Level Fault-Tolerant Structures Insertion and Validation", 20th IEEE VLSI Test Symposium, IEEE Computer Society, 2002, pp. 229-236.
- [25] R. Leveugle, "A new approach for early dependability evaluation based on formal property checking and controlled mutation", 11th IEEE International On-Line Testing symposium, St Raphaël, France, July 6-8, 2005, pp. 260-265
- [26] U. Krautz, M. Pflanz, C. Jacobi, H. W. Tast, K. Weber, H. T. Vierhaus, "Evaluating coverage of error detection logic for soft errors using formal methods", Design, Automation and Test in Europe Conference (DATE), March 6-10, 2006, pp. 176-181
- [27] P. Maistri, P. Vanhauwaert, R. Leveugle, "Evaluation of Register-Level Protection Techniques for the Advanced Encryption Standard by Multi-Level Fault Injections", 22nd IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT 2007), 2007, pp. 499-507
- [28] F. Ouchet, D. Borriore, K. Morin-Allory, L. Pierre, "High-level symbolic simulation for automatic model extraction", Proc. IEEE Symposium on Design and Diagnostics of Electronic Systems, April 2009
- [29] S. Seshia, W. Li, S. Mitra, "Verification-guided soft error resilience", Design, Automation and Test in Europe Conf. (DATE'07), April 2007
- [30] A. Darbari, B. Al-Hashimi, P. Harrod, D. Bradley, "A New Approach for Transient Fault Injection using Symbolic Simulation", International On-Line Testing Symposium (IOLTS), 2008
- [31] D. Larsson, R. Hähnle, "Symbolic Fault Injection", 4th International Verification Workshop, July 2007
- [32] M. Kaufmann, P. Manolios, J. Moore, "Computer Aided Reasoning: an Approach". Kluwer Academic Pub., 2002
- [33] W. D. Young, "Comparing verification systems: Interactive Consistency in ACL2", 11th Annual Conference on Computer Assurance, June 1996