Alternatives to Fault Injections for Early Safety/Security Evaluations

Regis Leveugle[1], Michele Portolan[1,2], Stefano Di Carlo[2], Alessandro Savino[2], Giorgio Di Natale[1] and Alberto Bosio[3]

[1] TIMA (Univ. Grenoble Alpes - CNRS - Grenoble INP*), Grenoble, France
Email: {name.surname}@univ-grenoble-alpes.fr
[2] Politecnico di Torino, Control and Computer Engineering Department, Torino, Italy
Email: {name.surname}@polito.it
[3] Lyon Institute of Nanotechnology, Ecole Centrale de Lyon, France
Email: {name.surname}@ec-lyon.fr

***Abstract*—Functional Safety standards like ISO 26262 require a detailed analysis of the dependability of components subjected to perturbations. Radiation testing or even much more abstract RTL fault injection campaigns are costly and complex to set up especially for SoCs and Cyber Physical Systems (CPSs) comprising intertwined hardware and software. Moreover, some approaches are only applicable at the very end of the development cycle, making potential iterations difficult when market pressure and cost reduction are paramount. In this tutorial, we present a summary of classical state-of-the-art approaches, then alternative approaches for the dependability analysis that can give an early yet accurate estimation of the safety or security characteristics of HW-SW systems. Designers can rely on these tools to identify issues in their design to be addressed by protection mechanisms, ensuring that system dependability constraints are met with limited risk when subjected later to usual fault injections and to e.g., radiation testing or laser attacks for certification.**

I. Introduction

In the recent years, many application domains have seen functional safety added to their classical list of design constraints [1]. In addition to traditional areas such as space or avionics, the ISO 26262 standard for automotive defined many safety requirements, increasingly important to go towards autonomous vehicles [2]. Similar standards are evolving in other areas such as railways, medical devices, or industrial devices and machinery (ISO/IEC/DO standards).

To focus by example on automotive, providing sufficient evidence for an Automotive Safety Integrity Level (ASIL) qualification requires a time- and resource-consuming process, today involving state-of-the-art fault injection approaches. When security is (also) a constraint, similar approaches are required to evaluate the robustness versus fault-based attacks that take advantage of computation errors to recover secrets [3], [4], [5].

No matter the final goal (safety, security, etc.) engineers need to get an evaluation of the effect of errors in their circuit as early as possible. In some cases, classical fault injection approaches applied at the Register-Transfer-Level (RTL) can be replaced by other approaches in order to early evaluate the level of robustness achieved, taking into account a given hardware architecture and/or an application software.

This paper summarizes the challenges related to a fast but accurate early evaluation of dependability in a cross-layer scenario, from hardware to software, discussing the most usual up-to-date approaches. The paper starts by presenting state-of-the-art fault injections techniques using either simulation or emulation and then moves to several complementary alternative approaches that try to avoid costly fault injections at each modification of the global system but are still accurate with respect to a given application. The presented approaches cover both pure hardware blocks and microprocessor based systems including the executed software. Results come from previous works performed in the presenter's teams and in the framework of the European FP7-CLERECO project [6].

Remaining challenges and perspectives of the presented approaches, even outside the scope of pure dependability evaluation, will be drawn.

II. Metrics and Definitions

This section provides a short glossary of reliability-related terminology to guarantee a common understanding of the terms used in this paper. The taxonomy of dependable computing defined in [7] is used as a reference.

Dependability is a global concept representing the extent to which a system is expected to operate in compliance to its specifications. According to [7], the concept of dependability covers different aspects including:

- *Availability*: readiness for correct service;
- *Reliability*: continuity of correct service;
- *Safety*: absence of catastrophic consequences for the user(s) and the environment;
- *Integrity*: absence of improper system alterations;
- *Maintainability*: ability to undergo modifications and repairs.
- *Security*: the state of being free from malicious dangers or threats.:

When considering a dependability threat, the following concepts must be considered:

*Institute of Engineering Univ. Grenoble Alpes

- *Failure*: an event that occurs when the delivered service deviates from the correct service. The deviation can be caused by incorrect design, environmental factors or malicious actions;
- *Error*: part of the total state of the system that may lead to its subsequent service failure;
- *Fault*: Adjudged or hypothesized cause of an error. A fault is active when it causes an error; otherwise it is dormant.

A. System-level reliability metrics

Several reliability metrics have been defined in the literature and Table I summarizes the most used ones. Most of them are applicable at the system level as well as at the component level.

B. Dependability threats

The metrics that characterize the dependability threats are necessary to understand and identify the weak points of the system. As previously explained the dependability threats are generally described by the concepts of faults, errors and failures as depicted in Figure 1. Dependability threats metrics measure the probability of occurrence of these events, and the relations among these events.

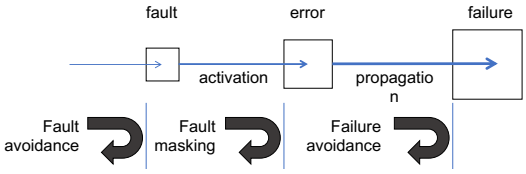

Fig. 1. The chain of dependability threats

1) Faults: Following the taxonomy of [7], faults can be classified according to different properties summarized in Figure 2. This includes the phase of creation or occurrence, the system boundaries, the phenomenological cause, the dimension, the objective, the intent, the capability and the persistence. Examples of faults that can affect the hardware of a computing system are: manufacturing defects (e.g., open or short circuits, parametric failures), physical deterioration (e.g., wear-out effects like Negative-Bias Temperature Instability - NBTI, electromigration, Time-Dependent Dielectric Breakdown - TDDB, Hot Carriers Injection - HCI) and physical interference (e.g., soft-errors and electromagnetic interference -EMI).

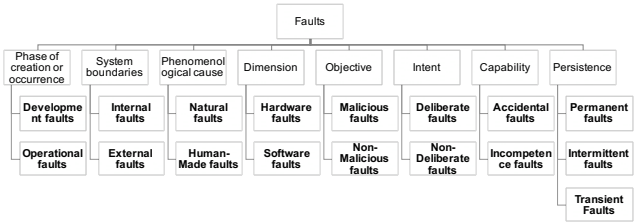

Fig. 2. Classification of faults (based on [7])

When considering hardware faults, the persistence (i.e., permanent, intermittent and transient) is among the most considered properties. The concept of intermittent faults [11] has been added to the used taxonomy in order to take into account the different nature and impact of these faults. This paper focuses on soft-errors. Table II lists the metrics that are typically used for the measurement of transient fault occurrences in a computing system.

2) Errors: When faults are activated, they result in errors. However, a large part of faults are not activated but dropped. The possible outcomes of a single-bit fault in different states have been classified in [14] and are represented in Figure 3. Table III presents the metrics that are most commonly used for

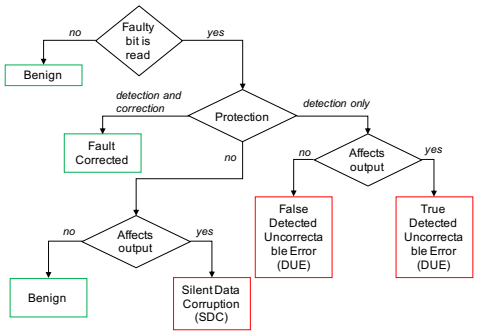

Fig. 3. Classification of the possible outcome of a faulty bit from [14]

the characterization of errors. It includes metrics that are used to characterize the probability of occurrence of errors and also to characterize the probability of activation of a fault (which results in an error).

C. Failures

A failure of the system happens when the delivered service deviates from correct service. The way the system deviates from a correct service is the failure mode of the system. Figure 4 represents the set of failure modes of a computing system as defined in [7].

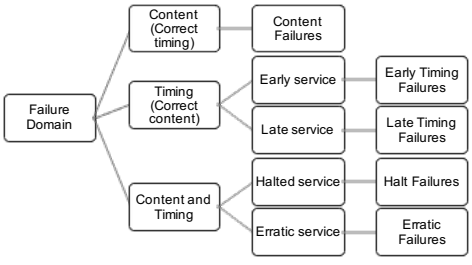

Fig. 4. Failure modes of a computing system from [7]

Besides its mode, a failure can be characterized by:

- Its *detectability*: if the failure of the system is detected and signaled;
- Its *consistency*: if the failure is perceived identically by all system users;

TABLE I
SUMMARY OF BASIC RELIABILITY METRICS.

Name	Description
Failure rate (λ)	Number of failures per unit of time. If the failure rate λ is constant, reliability can be modeled using an exponential distribution
Executions per Failure (EPF)	A measure of the number of times an application must be executed before observing a system failure [8]. It is computed as: $EPF = EIT/\lambda$ where EIT (Executions in Time) is the number of executions of an application in 10^9 hours of device operation. It enables a joint analysis of reliability and performance.
Failures In Time (FIT)	A measure of the failure rate in 10^9 device hours: $FIT = \lambda_{hours} * 10^9$
Mean Time To Failure (MTTF)	The arithmetic mean time to failure of a system. Usually expressed in hours. For instance, if a system's MTTF is 2 years, then on average a failure occurs every 2 years. It is a basic measure of reliability for non-repairable items. For non-repairable items with constant failure rate: $MTTF = 1/\lambda$
Mean Time To Repair (MTTR)	The mean time to repair an error once it is detected. It measures service interruption. This time is determined by the repair and recovery mechanisms that a system is equipped with. The smaller the MTTR the higher the reliability of the system. It is a basic measure of the maintainability of repairable items
Mean Time Between Failure (MTBF)	Arithmetic mean (average) time between failures of a system. Usually expressed in hours. Basic measure of reliability for repairable items. MTBF is calculated as: $MTBF = MTTF + MTTR$
Mean Work to Failure (MWTF)	Captures the average amount of work between two errors and is useful to compare the reliability of different workloads [9]
Mean Instructions to Failure (MITF)	Expresses the average number of committed instructions in a microprocessor between two errors [10]

TABLE II
METRICS ON TRANSIENT FAULTS

Name	Description
Single Event Upset (SEU) rate	Measures of the occurrences of SEU per unit of time. Expressed in FIT/Mb or SEU/Mbit/h. Depends on technology and, environment.
Multiple Bit Upset (MBU) rate	Measures the occurrences of MBU per unit of time. A MBU is defined as "any event or series of events that cause more than one bit to be upset during a single measurement" [12].
Single Event Functional Interrupt (SEFI) rate	Measures the occurrences of SEFI per unit of time. A SEFI is "a soft error that causes the component to reset, lock-up or otherwise malfunction in a detectable way, but does not require power cycling of the device (off and back on) to restore operability" [13].
Single Event Transient (SET) rate	Measures the occurrences of SET per unit of time. A SET is defined as a "momentary voltage excursion (voltage spike) at a node in an integrated circuit caused by a single energetic particle strike" [13].
Single Event Latch-Up (SEL) rate	Measures the occurrences of SEL per unit of time. A SEL is an "abnormal high-current state in a device caused by the passage of a single energetic particle through sensitive regions of the device structure and resulting in the loss of device functionality" [13].

TABLE III
METRICS ON ERRORS AND VULNERABILITY TO ERRORS

Name	Description
Silent Data Corruption (SDC) rate	Probability of occurrence of Silent Data Corruption. Form of error where a fault induces the system to generate erroneous outputs [15].
Detected Unrecoverable Error (DUE) rate	Probability of faults that are detected but cannot be recovered.
Time Vulnerability Factor (TVF)	Measures the fraction of time during which a device is susceptible to radiation-induced upsets [16].
Architectural Vulnerability Factor (AVF)	Measures the vulnerability of a hardware structure to faults. Defined as the probability that a fault in that particular structure will result in an error [17].
Program Vulnerability Factor (PVF)	Captures the architecture-level fault masking inherent in a program, allowing software designers to make quantitative statements about a program's tolerance to soft-errors [18].
Hardware Vulnerability Factor (HVF)	Quantifies the vulnerability of hardware structures to errors [18].
Hard-Fault Architectural Vulnerability Factor (H-AVF)	Measures the probability to commit an erroneous state due to hard faults in a microprocessor structure [19].

- Its *severity* and its *consequences*: if the failure has a minor impact or catastrophic consequences.

The notion of failure severity is heavily dependent on the application domain and there are no generic definitions that can be applied to all application domains.

III. FAULT-INJECTION-BASED DEPENDABILITY ANALYSIS

The baseline solution to evaluate the dependability of a system is to submit it to a controlled set of faults and observe its behavior. As explained in the previous section, not all faults necessarily result in a failure: the role of fault injection campaigns, is therefore to directly observe and quantify the ratio of faults being activated and ending up in a system failure. The principle, depicted in Figure 5, is quite simple: while the System Under Investigation (SUI) is subjected to a test bench, a Fault Injection Mechanism perturbs its behavior and an Acceptance Check verifies if the SUI is still inside specification or if any failure occurred.

Even though the principle is simple, the actual implementation of such a scheme is extremely complex.

First, there is the fault model: which faults must be injected?

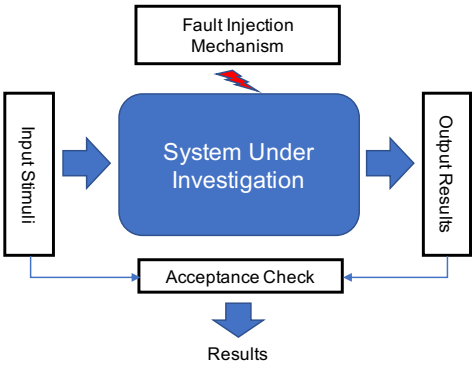

Fig. 5. General Setup of a Fault Injection Campaign

At which abstraction level? The usual considered levels are:

- Physical injection (e.g.: through controlled radiation testing);
- Transistor level (Spice simulations);
- Register Transfer level (RTL);
- Micro-architecture level;
- Software level.

Of course the lower the abstraction level is, the more precise the fault model is. Therefore, observability and debug capability will be extremely high. However, in the meantime, the simulation will take longer. The higher the abstraction level is, the faster the simulation will be, but the results will loose precision. Usually, information collected from lower levels are used to parameterize higher-level campaigns. In the following sections, we will present each of these levels and their relationships in more details.

Another important point to consider is the injection space. Faults can occur at any target location inside the system and at any time. Therefore, a simulation must be done for each pair (Target, Time) to fully characterize the system's behavior. The total set grows extremely fast: for instance, if a 1000-gates systems is subjected to a 100ns test bench, an exhaustive fault injection campaign on all gates with a fault each nanosecond would require $1000 \times 100 = 10K$ simulation runs. Developing efficient yet reliable fault models and exploiting the advantages of each abstraction level is therefore a priority to avoid hitting the "Combinatorial Wall".

A. Fault Injections and Abstraction Models

This section overviews, for each abstraction level, relevant fault injection approaches in an effort to point out advantages and drawbacks of each of them.

1) Physical fault injections: Physical fault injection uses external physical sources to introduce faults into the system's hardware (i.e., the actual target system) [20].

There are two main groups of physical fault injection techniques: by physically perturbing the external interface of the device (e.g., voltage/current pulses to the external pins of an integrated circuit) or by exploiting energetic sources (e.g., heavy ions, radiations, electromagnetic or laser beams) to internally affect the normal behavior of the device.

The advantage of physical fault injection is its accuracy in terms of realistic fault effect, and the ability to access some locations that are not accessible by other techniques. These techniques are also suitable for systems requiring high time-resolution for hardware triggering and monitoring. Nevertheless, they require: (i) the availability of the target device (thus not allowing early reliability analysis); (ii) special hardware infrastructures to inject the faults and to observe the results of the injection campaign. These infrastructures are usually very expensive; (iii) the high risk to damage the system while injecting the faults.

2) Simulation-Based Fault Injection at Transistor, Gate, RTL level: Perturbations on a circuit can be analyzed at various levels, with different representations of the fault and different levels of accuracy. Taking as an example the impact of a particle on a logic element, the lowest level of analysis will consider the electric charges generated in the silicon and the effect of the charge transfers between the electrodes of a transistor. This is analyzed using detailed technological models of a transistor (TCAD simulators) [21]. Such simulations are very time consuming and limited to cells with only a few transistors. They are used to create electrical models of the perturbations, i.e., shapes of current pulses that can then be used to analyze propagations in larger cells with electrical simulators such as SPICE. The results can then be compiled in a characterization library for e.g., standard cells in a given technology. This library allows simulations at gate level for blocks using these standard cells, taking into account both the logical effect of the expected perturbations (previously mentioned models such as SETs or SEUs) and the timing characteristics when appropriate. At each increasing level, a designer can analyze the perturbation effect on a larger circuit with a given simulation effort, but with a reduced accuracy with respect to the physical phenomena.

Even gate-level simulations can only be done very late in the design process and remain very time-consuming. Earlier analyses require more abstract fault models, that can be applied on higher level circuit descriptions, available earlier in the design process. In that case, timing information cannot be considered, nor propagations in the gate networks that are not yet defined. The usual fault model is therefore single or multiple bit-flips in registers and memories, corresponding to either direct perturbations of these elements or the sampling of wrong values due to SETs. Although the analysis can only be functional, simulations on Register-Transfer Level (RTL) descriptions can be faster by at least two orders of magnitude compared with gate-level simulations and they can give insights into critical parts of the circuit at an earlier stage of the design. Similar simulations can even be done before, on system-level (e.g., Transfer-Level - TLM) descriptions [22] but in that case .even the actual registers in the final circuit are unknown and the accuracy of the results is noticeably lower.

3) Simulation-Based Fault Injection at Micro-architectural level: Micro-architectural simulators are designed to simulate the full-system cycle in an accurate fashion [23]. More in detail, a micro-architectural simulator provides a software

model of the hardware architecture able to mimic the hardware behavior cycle-accurate level fully comparable with low level models, such RTL [24]. It means that the execution can emulate the pipeline of the hardware architecture and the In-Order or Out-of-Order execution. The hardware components that are usually described in a micro-architectural model are all memory data structures while all computational and control blocks are only functionally implemented. Nowadays, micro-architectural simulators hold a dominant role in early and accurate reliability assessments of several array-based micro-architectural structures that occupy the majority of the chip's area. The explanation lays on (i) the early availability in the design phase, (ii) the fast execution time compared with lower level models, (iii) the configuration flexibility of the model, (iv) the observability of the behavior at several levels of the system stack, (v) the accurate support of important Instruction Set Architectures (ISA).

Several fault injection architectures built on top micro-architectural level simulators have been proposed to analyze the software behavior in the presence of faults [25], [8], [26], [27], [28].

One of the main reasons is that fault injection approaches require small modifications of the simulators and, when statistically significant numbers of fault injections are performed [29], [30], fault injection delivers very accurate reports on the faulty behavior of hardware components. Moreover, compared to lower level fault injection approaches it allows execution of large portions of workloads to study the effect of faults to the final program output.

Most of the fault models introduced in Section II (transient, intermittent and permanent faults) can be analyzed at this level as far as they are applied to micro-architectural memory array components. Moreover, the high simulation throughput enables the analysis of multiple faults in several combinations, i.e., multiple faults of any type and any duration in a single structure or multiple faults on different structures. One of the main criticisms on such methods is the limitation to the memory structures of the system and the inability of dealing with asynchronous systems. In fact, all models relies on the synchronous behavior of the modeled system.

4) Software-Based Fault Injections : Software-Based Fault Injection (SBFI) requires software modifications/instrumentations in order to be able to model the impact of faults affecting the hardware layer of the system. The main advantage of SBFI is the capability to target a full application as well as Operating System, which cannot be simulated using the simulation-based fault injection. On the other hand, SBFI suffers for the limited controllability that limits the injection of the faults only to the locations accessible by software instructions.

Figure 6 sketches how physical faults reach and manifest themselves at software layer. A straightforward way to model faults is to map them into a set of fault models that affect the instructions and the data at software layer. Examples of software-level fault models are the Wrong Data in Operand (WDat) and the Instruction Replacement (InstR) [31], [32], [33]. They model the effect of transient/permanent faults occurring either in the memory segment storing the data of the program (WDat) or in the memory segment storing the code (InstR). SBFI can be classified depending on how the fault is injected into: compile-time (i.e., static) or run-time (i.e., dynamic).

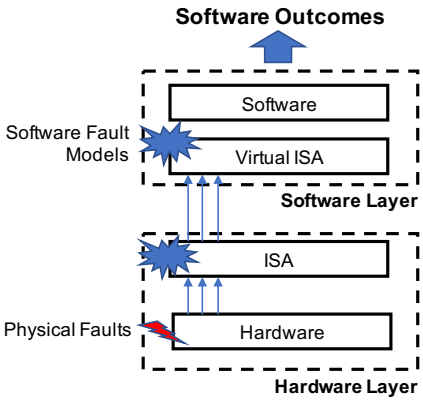

Fig. 6. Fault Propagation through System Layers

a) Compile-time : At compile-time, the fault is injected by modifying the software executed by the system. The software modification can be done at different levels: source code, intermediate representation or even at byte-level code. In [34] the concept of mutation testing is applied to mutate the source code (e.g., C, C++,) in order to inject a software fault. Each mutation corresponds to one fault model. [35], LLFI [36], [37] and KULFI [38] propose a fault injection environment able to inject software fault models into the LLVM intermediate code level of the application. The main idea is still the code mutation but at lower level than original source-code. In this way these approaches are language independent. Similar concepts applied at byte-code are presented in [39], [40], [41]. Here, the use of the Javassist toolkit [42] allows to be as independent as possible from the source code and to easily manipulate the byte-code of the loaded classes at run-time.

b) Run-time: At run-time, the fault is injected during the execution of the software without modifying its source code. FERRARI [43] uses traps and system calls in order to modify the execution state of the target application in UNIX systems. In [44] the fault injection mechanism is based on the interaction between two parallel processes: the fault injection (thread) and the target program process. XCEPTION [45], Faust [46] and [47] use advanced debugging and performance monitoring features of the actual processor in order to perform fault injections as realistically as possible.

B. Multi-Level Approaches

As seen in the previous subsections, fault simulations can be performed at different abstraction levels (physical, transistor, gate, RTL, micro-architecture and software). Low abstraction levels provide high accuracy in terms of simulation results, while higher levels allow simulating bigger systems in reasonable time. However, in some cases, fault effects must be

simulated at low abstraction levels to be representative of the physical phenomenon, while the outcome must be obtained from the whole system. To solve these cases, researchers proposed fault simulators that integrate co-simulation (at lower level for the fault, at higher level for the system). The terminology in this domain is not always uniform, and the reader can find this concept under different names: multi-level, mixed-mode, hybrid or multi-scale fault simulation.

Different approaches have been proposed for simulating the effects of permanent faults (to target defects) and transient faults (to target radiation effects or fault attacks). Concerning the first category, a multi-level fault simulator operating at switch- and gate-levels is described in [48]. In this approach, low/high resistances are used to model on/off states of transistors. [49] uses event-based mixed-level fault simulation to simulate the effect of manufacturing defects more accurately while maintaining a tolerable simulation time, while [50] targets bridging faults (at electrical and switch levels) while simulating the system at gate level.

In the second category, timing precision is required. All proposed solutions follow the same approach: the simulation is initiated at high level, up to moment where the fault is injected. At that moment, a low-level simulation is executed (for the same circuit described at low level), its state is copied from the previous simulation, and it is executed for the duration of the fault. Finally, the state is copied from the low-level to the high-level simulator to finish the simulation. Existing simulators cover RTL and transistor levels [51], gate and RTL levels [52], transaction level [53], and micro-architecture level [27].

C. Do it Smartly

As mentioned in the previous sections, simulations for fault injections can be extremely time-consuming (days for a simple circuit with exhaustive injections of the simplest single bit-flip fault model, up to weeks or months even on a server farm for a SoC described at RT-Level with a complex application to run). In order to reduce the Time-to-Market, it is mandatory to improve the efficiency of fault injections. Three complementary ways have been exploited: fault pruning, statistical fault injection and emulation-based experiments.

Fault pruning is a process identifying on an analytical basis faults or errors that cannot have any effect on the global computation. It is related to the AVF analysis of microprocessors for e.g., unused bits in instruction coding. It can be related to other properties in pure hardware blocks [54]. It can also be based on some abstract model of the architecture, using e.g., Petri Nets. These approaches are used before the final list of errors to inject is decided.

Once the fault/error model to consider is defined, the injection space size is the product of the number of potential errors by the number of potential occurrence instants. This space is in general huge so exhaustive injections are not feasible in the available timeframe. Many publications report an arbitrary number of injected faults that should be sufficient to trust the results. A better approach is to compute the required number of injections for a Gaussian distribution and a targeted level of confidence. In the case of a large injection space, the number of injections can be drastically reduced while keeping control over the trust in the results [55], [29]. Fault pruning can still help in reducing the final list.

When the injection list is finalized, the experimental time can be reduced by exploiting a hardware prototype of the circuit rather than a software simulation. Even compared with RTL simulation, the campaign duration can be reduced by several orders of magnitude. Setting up the prototype of course requires some effort but the experiments are noticeably accelerated. This was early proposed in [56] based on industrial emulators available at that time. The approach was then exploited on FPGA boards, taking advantage of dynamic partial reconfiguration capabilities to reduce the injection times without adding any hardware modifications in the circuit under analysis [57], [58], [59], [60]. More recently, large industrial SoC emulators have also been used [61], [55].

IV. Alternative Approaches and Cross Layer Analysis

From the previous sections it is clear that, while fault injection remains an unavoidable step to obtain a complete characterization of a system, it has several drawbacks when systematically used during the development steps of a system. Regardless of the considered abstraction layer, it is computationally expensive; therefore, its usage during design iterations would significantly affect the project deadline and cost. Moreover, dependability results would arrive extremely late in the development cycle, so any design change (typically, to harden the most sensitive parts) would require extremely long design iterations. For all these reasons, alternative approaches have been proposed to provide dependability estimations early enough in the design flow, allowing for easy iterations. The goal is not to obtain precise dependability figures, for which fault injection remains the only reliable approach, but rather to provide the designers with tools and metrics to guide the development phase by identifying the most critical elements.

An approach that gives excellent results and that has been widely used is the so-called Register Data Lifetime (RDL) [62]: instead of analyzing the whole injection space, the attention is focused on the periods when data contained inside a register are actually used. The principle is very simple: if a fault affects data stored into a register, it will have a potential effect, i.e., it will be activated, only when this data is read. Therefore, RDL focuses on identifying the "alive" periods of the register, i.e., the time between the moment it is written and the moment it is read, and the "dead" ones, i.e., the periods in which a register contains useless data. Based on those periods, it is possible to define the *Criticality* of a register as: $Criticality(R) = (\text{R Lifetime})/(\text{Total cycles}) \times 100$, where $RLifetime$ represents the sum of the alive periods of the register. This metric is extremely useful because it can identify the most critical parts of a system as soon as it is available, allowing the designer to proactively harden them. Moreover, RDL can be computed on different abstraction levels. In [63],

authors evaluated it at the micro-architecture level on a Leon 3 processor [64]: by combining the information given in the data sheet and an analysis of the RTL code, it is possible to construct a simplified model of the processor that is able to track dependencies and information flow inside the processor. This model is simpler and much faster than a standard ISA simulator because it is simply tracking data dependencies and is not concerned with the actual calculation results. The input of the model is a trace extracted from an RTL simulation for a given test-bench. To validate the model estimates, a full fault injection campaign was performed, showing that the criticality estimations are quite precise. Example results are depicted in Figure 7. The estimated criticality values are extremely close to the reference. It is also worth noticing that this approach requires only one simulation of the target test-bench, while fault injection requires a simulation for each fault configuration.

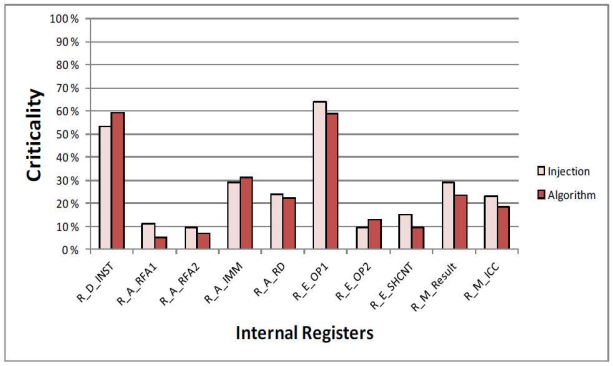

Fig. 7. Criticality comparison between Fault Injection and RDL, from [63]

By itself, criticality is an abstract value that has no direct relationships with the design: its value is in using it as a metric for identifying sensitive areas. For instance, Figure 8 shows another data presentation, with a Registers Binning based on criticality levels: each pie regroups the internal registers of the Leon 3, and coloring identifies the most critical registers for a given criticality level. This type of information can be directly used by designers to guide the hardening selection based on the dependability levels required by the final system.

In [63], the speedup when compared to an emulation-based fault injection campaign is around 40 to 50x, allowing several iterations during the device cycle. Even though the performances are extremely good, this micro-architecture level approach demands a good knowledge of the target processor, and a significant development time to obtain and validate the RDL model. To overcome this limitation, RDL can also be applied directly at the RTL level as described in [65]. When looking at RTL descriptions, scaling is an immediate problem: direct monitoring of all signals for all registers is not feasible in a timely and resource-efficient way. For this reason, [65] starts by identifying the "Logic Cones", i.e., the internal logic whose value has an effect on a given functional output, from which it is possible to extract a subset of meaningful signals

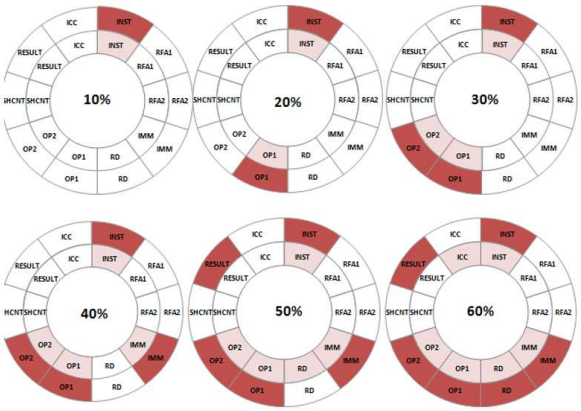

Fig. 8. Register Binning based on Criticality Levels, from [63]

to be monitored. This usually amounts to less than 1% of the total signals. As with the micro-architecture level approach, a single reference simulation is enough to compute RDL values. Once more, RDL-computed criticality is extremely close to reference results obtained through traditional fault injection campaigns, as can be seen for instance in Figure 9.

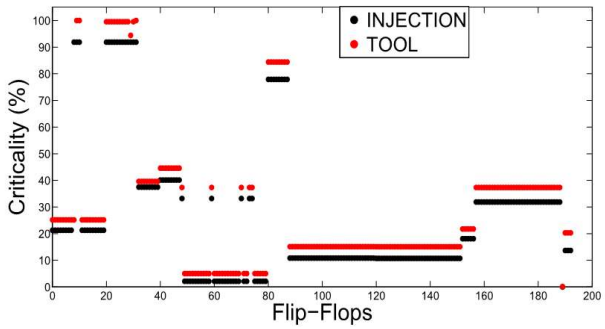

Fig. 9. Comparison with fault injections for a CRC running on Leon3 [65]

Once more the need of having just one execution run allows for extremely good performances: even when compared to emulation-based fault injection, the tool provides almost 10x speedup even though it implies the RTL simulation of a complete processor running a software. If the comparison is done with a simulation-based fault injection, the speedup can be reach [65] 400x. RDL criticality evaluation can therefore be directly used during the traditional design phase for an early identification of the most critical elements without seriously impacting development time.

A very similar set of approaches have been developed in the computer architecture community around the concept of Architectural Correct Execution (ACE) analysis to compute the AVF of a microprocessor using architecture level simulators [17], [66], [67]. These approaches are complex. They require significant modifications to the simulators to track resources during the execution of the program. Therefore, they are

limited to the analysis of small programs. Apart for the complexity, accuracy is a general limitation of these approaches. An 7x AVF over-estimation is reported in [68]. Even with refined approaches, which require additional complexity in the simulation, ACE analysis still provides 3x overestimation. This has a detrimental effect on the system leading to system over-design [67].

Similar approaches have been also proposed at the software level to analyze the impact of faults into the data of a software application and therefore compute the system's PVF. [69] proposes to use the variable lifetime combined with the data dependency graph to identify critical variables of an application to be protected. In [33] and [35], the authors presented an analytical methodology to measure the vulnerability of the memory components of a microprocessor-based computing system, based on the data and the instruction lifetime evaluation and residence. The approach considers only the software-layer of the system, which makes it usable during early design stages when the hardware architecture is not fully defined. To consider the hardware memory hierarchy (i.e., RAM, Caches, Register Files) at software level, the paper proposes a memory subsystem emulator that can be easily configured to support different memory and cache features. The approach works with LLVM, and it instruments the original source code with assertions that allow calculating the lifetime of every single variable, by emulating their residence in the memory hierarchy (L2 cache, L1 cache, registers). Their results showed that the methodology can be applied on very complex applications to evaluate their reliability.

Most of the approaches presented so far work at a single abstraction level and therefore are constrained by the limitations of the considered layer. Overall, a cross-layer holistic evaluation approach has several advantages compared to more traditional single layer techniques.

The first attempt to model the contribution of the software to the AVF of the system is provided in three seminal papers by Sridharan and Kaeli [70], [71], [18]. They introduce the concept of Program Vulnerability Factor (PVF) to quantify the portion of the AVF that can be attributed to the executed software. This concept has been further extended in [18] with the introduction of the concept of the System Vulnerability Stack. The System Vulnerability Stack is a significant advance towards the definition of a cross-layer system reliability model. However, its main drawback is that it oversimplifies the definition of the layers. In particular, the basic assumption is that the layers are statistically independent. This allows to compute the AVF of the system simply as the product of the vulnerability factors of each layer. Moreover, the layers are not further split into their composing components, preventing a fine-grained analysis of the architecture of the system.

Another interesting solution that considers the impact of the application software running on embedded microprocessors was discussed in [72]. Similarly, to ACE analysis, it is based on the use of program traces. One of the main contribution of this model is to first introduce stochastic AVF analysis. Nevertheless, being based on program traces, it suffers from inaccuracies due to the fact that it cannot capture important masking effects introduced during dynamic execution of the software. Moreover, it is limited to bare metal applications. An extension of this model based on Bayesian probability propagation has been proposed in [73].

V. Beyond simulations: stochastic models and design space exploration

It is clear from the previous sections that both fault injection techniques and alternative approaches have positive and negative aspects that must be carefully analyzed whenever choosing the best approach to evaluate the reliability of a target system. In fact, with the increasing adoption of cross-layer reliability techniques, there is an increasing interest into stochastic models able to combine the benefit of fault-injection techniques at different abstraction levels, analytical approaches such as RDE or ACE analysis and stochastic models able to cope with the complexity of the target design together.

Modern SoCs and CPSs show a tight integration and interaction between hardware and software built upon an integration capacity of billions of transistors. Moreover, on top of the hardware a complex software stack including an operating system and complex application is usually executed and must be taken into account when performing the reliability analysis. In such scenario, reliability is not the only design constraint to be optimized anymore: power, energy and performance are equal partners when accounting for the system specifications. Speaking of dependability, in a cross-layer resilient system, mitigation techniques may work at physical and circuit level, mitigating low-level faults, while hardware redundancy can be used to manage errors at the hardware architecture layer and software implemented error detection and correction mechanisms can manage those errors that escaped the lower layers of the stack [74]. Since software and hardware are linked each other, it is not surprising anymore that also the actual workload has direct impact on data and control flow.

Recently, Henkel et al. [75] proposed a multi-layer dependability approach to improve the design flow of a generic electronic system. It analyzes the system hierarchically, following the way faults propagate from the bottom layer up to the application layer. The underline idea is to apply protection mechanisms at different layers in the system design. For each layer, the paper proposes to resort to a design space exploration (DSE) procedure to select the set of protection mechanisms. The selection covers faults that escaped the protection mechanisms implemented in the lower hierarchical levels. Since the exploration is done by moving in a hierarchical order,the approach is biased toward low-level protection mechanisms because they are applied first. Therefore, it does not allow to fully explore the space of possible design options.

To go toward the definition of a full framework for cross-layer early reliability estimation, the FP7-EU Clereco project [6] has worked to develop a full framework named SyRA (System Reliability Analyzer) able to analyze the impact of radiation induced soft errors in the memory arrays of a complex computing cores, such as microprocessor and GPUs

[73], [76], [77]. SyRA can support designers in the early phases of the design, considering all layers of a system from the hardware up to the application software (including the operating system).

Figure 11 shows the architecture of the implemented framework. SyRA exploits a multi-level hybrid Bayesian model to describe the target system and to estimate different reliability metrics. The construction of the system is based on simulations at the different abstraction levels, thus combining into a single model the benefits of fault injection at all abstraction levels and the power of Bayesian stochastic models. This allows designers to speed up the analysis and therefore to cope with the complexity of the simulation of the full system stack. SyRA can compute several reliability metrics including AVF, FIT and EPF and the whole framework scales efficiently with the complexity of the system. Experimental results show that, on average, it is 68% faster than full micro-architecture level fault injection and two orders of magnitude faster than RTL fault injection while maintaining a comparable accuracy [24].

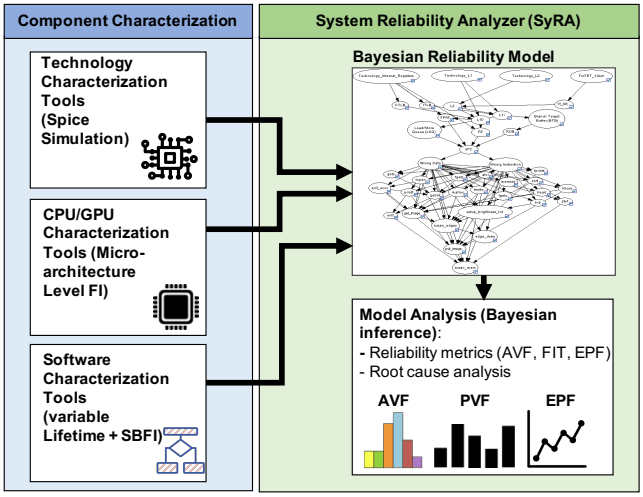

Fig. 10. Overview SyRA cross-layer reliability framework. The component characterization toolset integrates a set of characterization tools for technologies [78], CPUs [26], GPUs [25] [79] and software routines [32], [35]. The tools are used to build the Bayesian Reliability model that is at the core of our System Reliability Analyzer (SyRA) [77], [76].

The complete tool-chain developed to build the model is described in [76], [77] and an example of the accuracy of the analysis performed by SyRA is reported in Figure 11. The proposed framework scales efficiently with the complexity of the system. On average it is 68% faster than full micro-architecture level fault injection and two orders of magnitude faster than RTL fault injection while maintaining a comparable accuracy [24].

The potential of this type of approach is even more evident looking at the design space exploration (DSE) problem. From a designer perspective, performing DSE requires analyzing several configurations of the system, weighting them considering all key constraints, e.g., dependability as well as power consumption, area, etc., and delivering the best solution possible. In [80], the authors clearly showed that performing

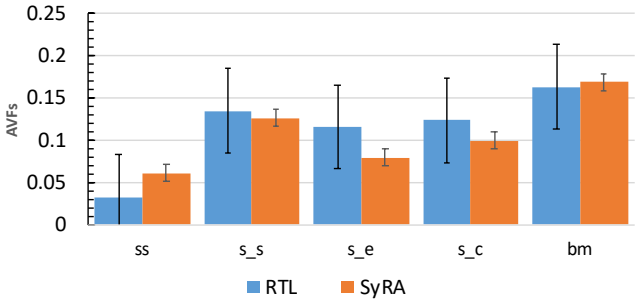

Fig. 11. Adapted from [77]. Results obtained using SyRA to estimate the AVF of five different applications executed an ARM Cortex A9. The figure compares estimation provided by SyRA with those obtained using precise RTL fault injection. The full experimental setup is described in [77].

the required simulation campaign for every new product in the early stages of the design might not be affordable. This is clearly only feasible if fast and accurate reliability estimation models are available. As shown in [81], the fast metric computation capabilities reached by the model proposed in [77], together with the quality of its estimations, enables the creation of optimization engines to support the DSE solutions.

VI. Conclusion

TO EDIT *********

Fault Injection works, but it is slow and costly. Even "smart on big machines" is not enough. Difficulty in modeling/simulating environment. Multi-level modeling can give better performances. More analytical analyses can help without reducing accuracy under some hypotheses A few words on perspectives on triggering approximate computing (without of course repeating the paper, but just the basic ideas) Other potential use of the proposed tools?

As mentioned in the introduction, the presented approaches aim at evaluating the effect of errors, no matter their origin. They are often used for safety analyses, extended to availability or reliability constraints. They are also meaningful in terms of security evaluations with respect to fault-based attacks. In that case, the fault/error model may have to be adapted to represent the expected attacker capabilities but the general experimental process can remain unchanged.

References

[1] D. Smith and K. Simpson, *Functional safety*. Routledge, 2004.

[2] ISO, "Road vehicles – Functional safety," 2011.

[3] C. Giraud and H. Thiebeauld, "A survey on fault attacks," in *Smart Card Research and Advanced Applications VI*, J.-J. Quisquater, P. Paradinas, Y. Deswarte, and A. A. El Kalam, Eds. Boston, MA: Springer US, 2004, pp. 159–176.

[4] R. Leveugle, "Early analysis of fault-based attack effects in secure circuits," *IEEE Transactions on Computers*, vol. Vol.56, no. No.10, October, pp. 1431–1434, 2007. [Online]. Available: https://hal.archives-ouvertes.fr/hal-00178989

[5] R. Leveugle, P. Maistri, P. Vanhauwaert, F. Lu, G. Di Natale, M. . Flottes, B. Rouzeyre, A. Papadimitriou, D. Hély, V. Beroulle, G. Hubert, S. De Castro, J. . Dutertre, A. Sarafianos, N. Boher, M. Lisart, J. Damiens, P. Candelier, and C. Tavernier, "Laser-induced fault effects in security-dedicated circuits," in *2014 22nd International Conference on Very Large Scale Integration (VLSI-SoC)*, Oct 2014, pp. 1–6.

[6] Cross-layer early reliability evaluation for the computing continuum (EU-FP7 GA no. 611404). [Online]. Available: https://cordis.europa.eu/project/rcn/109949/factsheet/en

[7] A. Avizienis, J. . Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, Jan 2004.

[8] A. Vallero, S. Di Carlo, S. Tselonis, and D. Gizopoulos, "Microarchitecture level reliability comparison of modern gpu designs: First findings," in *Performance Analysis of Systems and Software (ISPASS), 2017 IEEE International Symposium on*. IEEE, 2017, pp. 129–130.

[9] G. A. Reis, J. Chang, N. Vachharajani, R. Rangan, D. I. August, and S. S. Mukherjee, "Design and evaluation of hybrid fault-detection systems," in *ACM SIGARCH Computer Architecture News*, vol. 33, no. 2. IEEE Computer Society, 2005, pp. 148–159.

[10] C. Weaver, J. Emer, S. S. Mukherjee, and S. K. Reinhardt, "Techniques to reduce the soft error rate of a high-performance microprocessor," in *ACM SIGARCH Computer Architecture News*, vol. 32, no. 2. IEEE Computer Society, 2004, p. 264.

[11] C. Constantinescu, "Intermittent faults and effects on reliability of integrated circuits," in *2008 Annual Reliability and Maintainability Symposium*. IEEE, 2008, pp. 370–374.

[12] R. Reed, M. Carts, P. Marshall, C. Marshall, O. Musseau, P. McNulty, D. Roth, S. Buchner, J. Melinger, and T. Corbiere, "Heavy ion and proton-induced single event multiple upset," *IEEE Transactions on Nuclear Science*, vol. 44, no. 6, pp. 2224–2229, 1997.

[13] JEDEC solid state technology association, "Jesd89a, jedec standard: Measurement and reporting of alpha particle and terrestrial cosmic ray-induced soft errors in semiconductor devices," 2006.

[14] S. S. Mukherjee, J. Emer, and S. K. Reinhardt, "The soft error problem: An architectural perspective," in *11th International Symposium on High-Performance Computer Architecture*. IEEE, 2005, pp. 243–247.

[15] S. Mukherjee, *Architecture design for soft errors*. Morgan Kaufmann, 2011.

[16] A. Bramnik, A. Sherban, and N. Seifert, "Timing vulnerability factors of sequential elements in modern microprocessors," in *2013 IEEE 19th International On-Line Testing Symposium (IOLTS)*, July 2013, pp. 55–60.

[17] 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. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36.* IEEE, 2003, pp. 29–40.

[18] V. Sridharan and D. R. Kaeli, "Using hardware vulnerability factors to enhance avf analysis," *ACM SIGARCH Computer Architecture News*, vol. 38, no. 3, pp. 461–472, 2010.

[19] J. C. Smolens, B. T. Gold, J. Kim, B. Falsafi, J. C. Hoe, and A. G. Nowatryk, "Fingerprinting: bounding soft-error-detection latency and bandwidth," *IEEE Micro*, vol. 24, no. 6, pp. 22–29, Nov 2004.

[20] N. Song, J. Qin, X. Pan, and Y. Deng, "Fault injection methodology and tools," in *Proceedings of 2011 International Conference on Electronics and Optoelectronics*, vol. 1. IEEE, 2011, pp. V1–47.

[21] G. Hubert and L. Artola, "Single-event transient modeling in a 65-nm bulk cmos technology based on multi-physical approach and electrical simulations," *IEEE Transactions on Nuclear Science*, vol. 60, no. 6, pp. 4421–4429, Dec 2013.

[22] K.-J. Chang and Y.-Y. Chen, "System-level fault injection in systemc design platform," in *2007 Proc. 8th Int. Symposium on Advanced Intelligent Systems*. Citeseer, 2007, pp. 354–359.

[23] A. Patel, F. Afram, S. Chen, and K. Ghose, "Marss: a full system simulator for multicore x86 cpus," in *2011 48th ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2011, pp. 1050–1055.

[24] A. Chatzidimitriou, M. Kaliorakis, D. Gizopoulos, M. Iacaruso, M. Pipponzi, R. Mariani, and S. Di Carlo, "Rt level vs. microarchitecture-level reliability assessment: Case study on arm (r) cortex (r)-a9 cpu," in *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE, 2017, pp. 117–120.

[25] A. Vallero, D. Gizopoulos, and S. Di Carlo, "Sifi: Amd southern islands gpu microarchitectural level fault injector," in *2017 IEEE 23rd International Symposium on On-Line Testing and Robust System Design (IOLTS)*. IEEE, 2017, pp. 138–144.

[26] M. Kaliorakis, S. Tselonis, A. Chatzidimitriou, N. Foutris, and D. Gizopoulos, "Differential fault injection on microarchitectural simulators," in *2015 IEEE International Symposium on Workload Characterization*. IEEE, 2015, pp. 172–182.

[27] M. Kaliorakis, S. Tselonis, A. Chatzidimitriou, and D. Gizopoulos, "Accelerated microarchitectural fault injection-based reliability assessment," in *2015 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS)*, Oct 2015, pp. 47–52.

[28] F. G. Previlon, B. Egbantan, D. Tiwari, P. Rech, and D. R. Kaeli, "Combining architectural fault-injection and neutron beam testing approaches toward better understanding of gpu soft-error resilience," in *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*, Aug 2017, pp. 898–901.

[29] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert, "Statistical fault injection: Quantified error and confidence," in *2009 Design, Automation Test in Europe Conference Exhibition*, April 2009, pp. 502–506.

[30] P. Vanhauwaert and R. Leveugle, "Efficiency of probabilistic testability analysis for soft error effect analysis: A case study," in *2009 4th International Conference on Design Technology of Integrated Systems in Nanoscal Era*, April 2009, pp. 236–240.

[31] S. Di Carlo, A. Vallero, D. Gizopoulos, G. Di Natale, A. Gonzalez, R. Canal, R. Mariani, M. Pipponzi, A. Grasset, P. Bonnot, F. Reichenbach, G. Rafiq, and T. Loekstad, "Cross-layer early reliability evaluation: Challenges and promises," in *2014 IEEE 20th International On-Line Testing Symposium (IOLTS)*, July 2014, pp. 228–233.

[32] A. Vallero, S. Tselonis, N. Foutris, M. Kaliorakis, M. Kooli, A. Savino, G. Politano, A. Bosio, G. Di Natale, D. Gizopoulos *et al.*, "Cross-layer reliability evaluation, moving from the hardware architecture to the system level: A clereco eu project overview," *Microprocessors and Microsystems*, vol. 39, no. 8, pp. 1204–1214, 2015.

[33] M. Kooli, G. D. Natale, and A. Bosio, "Memory-aware design space exploration for reliability evaluation in computing systems," *Journal of Electronic Testing*, 03 2019.

[34] R. Natella, D. Cotroneo, J. A. Duraes, and H. S. Madeira, "On fault representativeness of software fault injection," *IEEE Transactions on Software Engineering*, vol. 39, no. 1, pp. 80–96, Jan 2013.

[35] M. Kooli, G. Di Natale, and A. Bosio, "Cache-aware reliability evaluation through llvm-based analysis and fault injection," in *2016 IEEE 22nd International Symposium on On-Line Testing and Robust System Design (IOLTS)*, July 2016, pp. 19–22.

[36] Q. Lu, M. Farahani, J. Wei, A. Thomas, and K. Pattabiraman, "Llfi: An intermediate code-level fault injection tool for hardware faults," in *2015 IEEE International Conference on Software Quality, Reliability and Security*, Aug 2015, pp. 11–16.

[37] J. Wei, A. Thomas, G. Li, and K. Pattabiraman, "Quantifying the accuracy of high-level fault injection techniques for hardware faults," in *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, June 2014, pp. 375–382.

[38] V. C. Sharma, A. Haran, Z. Rakamaric, and G. Gopalakrishnan, "Towards formal approaches to system resilience," in *2013 IEEE 19th Pacific Rim International Symposium on Dependable Computing*, Dec 2013, pp. 41–50.

[39] R. L. de Oliveira Moraes and E. Martins, "Jaca - a software fault injection tool," in *2003 International Conference on Dependable Systems and Networks, 2003. Proceedings.*, June 2003, pp. 667–667.

[40] E. Martins, C. M. F. Rubira, and N. G. M. Leme, "Jaca: a reflective fault injection tool based on patterns," in *Proceedings International Conference on Dependable Systems and Networks*, June 2002, pp. 483–487.

[41] B. P. Sanches, T. Basso, and R. Moraes, "J-swfit: A java software fault injection tool," in *2011 5th Latin-American Symposium on Dependable Computing*, April 2011, pp. 106–115.

[42] S. Chiba and M. Nishizawa, "An easy-to-use toolkit for efficient java bytecode translators," in *Proceedings of the 2Nd International Conference on Generative Programming and Component Engineering*, ser. GPCE '03. Berlin, Heidelberg: Springer-Verlag, 2003, pp. 364–376. [Online]. Available: http://dl.acm.org/citation.cfm?id=954186.954208

[43] G. A. Kanawati, N. A. Kanawati, and J. A. Abraham, "Ferrari: a flexible software-based fault and error injection system," *IEEE Transactions on Computers*, vol. 44, no. 2, pp. 248–260, Feb 1995.

[44] F. Kaddachi, M. Kooli, G. Di Natale, A. Bosio, M. Ebrahimi, and M. Tahoori, "System-level reliability evaluation through cache-aware software-based fault injection," in *2016 IEEE 19th International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, April 2016, pp. 1–6.

[45] J. Carreira, H. Madeira, and J. G. Silva, "Xception: a technique for the experimental evaluation of dependability in modern computers," *IEEE Transactions on Software Engineering*, vol. 24, no. 2, pp. 125–136, Feb 1998.

[46] A. Benso, S. Di Carlo, G. Di Natale, P. Prinetto, I. Solcia, and L. Tagliaferri, "Faust: fault-injection script-based tool," in *9th IEEE On-Line Testing Symposium, 2003. IOLTS 2003.*, July 2003, pp. 160–.

[47] A. Jin, J. Jiang, J. Hu, and J. Lou, "A pin-based dynamic software fault injection system," in *2008 The 9th International Conference for Young Computer Scientists*, Nov 2008, pp. 2160–2167.

[48] W. Meyer and R. Camposano, "Active timing multilevel fault-simulation with switch-level accuracy," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 10, pp. 1241–1256, Oct 1995.

[49] M. B. Santos and J. P. Teixeira, "Defect-oriented mixed-level fault simulation of digital systems-on-a-chip using hdl," in *Design, Automation and Test in Europe Conference and Exhibition, 1999. Proceedings (Cat. No. PR00078)*, March 1999, pp. 549–553.

[50] Greenstein and Patel, "E-proofs: A cmos bridging fault simulator," in *1992 IEEE/ACM International Conference on Computer-Aided Design*, Nov 1992, pp. 268–271.

[51] F. Lu, G. D. Natale, M.-L. Flottes, and B. Rouzeyre, "Multilevel ionizing-induced transient fault simulator," *Information Security Journal: A Global Perspective*, vol. 22, no. 5-6, pp. 251–264, 2013. [Online]. Available: https://doi.org/10.1080/19393555.2014.891280

[52] A. L. Sartor, P. H. E. Becker, and A. C. S. Beck, "Simbah-fi: Simulation-based hybrid fault injector," in *2017 VII Brazilian Symposium on Computing Systems Engineering (SBESC)*, Nov 2017, pp. 94–101.

[53] R. Baranowski, S. Di Carlo, N. Hatami, M. E. Imhof, M. A. Kochte, P. Prinetto, H.-J. Wunderlich, and C. G. Zoellin, "Efficient multi-level fault simulation of hw/sw systems for structural faults," *Science China Information Sciences*, vol. 54, no. 9, p. 1784, Aug 2011. [Online]. Available: https://doi.org/10.1007/s11432-011-4366-9

[54] L. Berrojo, I. Gonzalez, F. Corno, M. S. Reorda, G. Squillero, L. Entrena, and C. Lopez, "New techniques for speeding-up fault-injection campaigns," in *Proceedings 2002 Design, Automation and Test in Europe Conference and Exhibition*, March 2002, pp. 847–852.

[55] J. Daveau, A. Blampey, G. Gasiot, J. Bulone, and P. Roche, "An industrial fault injection platform for soft-error dependability analysis and hardening of complex system-on-a-chip," in *2009 IEEE International Reliability Physics Symposium*, April 2009, pp. 212–220.

[56] S. Mir, B. Charlot, and B. Courtois, "Extending fault-based testing to microelectromechanical systems," in *European Test Workshop 1999 (Cat. No.PR00390)*, May 1999, pp. 64–68.

[57] L. Antoni, R. Leveugle, and M. Feher, "Using run-time reconfiguration for fault injection in hardware prototypes," in *17th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 2002. DFT 2002. Proceedings.*, Nov 2002, pp. 245–253.

[58] L. Sterpone and 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, no. 4, pp. 965–970, Aug 2007.

[59] S. Di Carlo, P. Prinetto, D. Rolfo, and P. Trotta, "A fault injection methodology and infrastructure for fast single event upsets emulation on xilinx sram-based fpgas," in *2014 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, Oct 2014, pp. 159–164.

[60] S. D. Carlo, G. Gambardella, P. Prinetto, F. Reichenbach, T. Løkstad, and G. Rafiq, "On enhancing fault injection's capabilities and performances for safety critical systems," in *2014 17th Euromicro Conference on Digital System Design*, Aug 2014, pp. 583–590.

[61] A. Benso, A. Bosio, S. D. Carlo, and R. Mariani, "A functional verification based fault injection environment," in *22nd IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT 2007)*, Sep. 2007, pp. 114–122.

[62] J. Lee and A. Shrivastava, "Static analysis of register file vulnerability," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 4, pp. 607–616, April 2011.

[63] K. Chibani, M. Ben-Jrad, M. Portolan, and R. Leveugle, "Fast accurate evaluation of register lifetime and criticality in a pipelined microprocessor," in *2014 22nd International Conference on Very Large Scale Integration (VLSI-SoC)*, Oct 2014, pp. 1–6.

[64] C. GAISLER, "Leon 3," [Online] www.gaisler.com/index.php/products/processors/leon3, May 2019.

[65] K. Chibani, M. Portolan, and R. Leveugle, "Application-aware soft error sensitivity evaluation without fault injections — application to leon3," in *2016 16th European Conference on Radiation and Its Effects on Components and Systems (RADECS)*, Sep. 2016, pp. 1–4.

[66] X. Li, S. V. Adve, P. Bose, and J. A. Rivers, "SoftArch: an architecture-level tool for modeling and analyzing soft errors," in *2005 International Conference on Dependable Systems and Networks (DSN'05)*, June 2005, pp. 496–505.

[67] N. J. Wang, A. Mahesri, and S. J. Patel, "Examining ACE analysis reliability estimates using fault-injection," in *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ser. ISCA '07. New York, NY, USA: ACM, 2007, pp. 460–469.

[68] N. J. George, C. R. Elks, B. W. Johnson, and J. Lach, "Transient fault models and AVF estimation revisited," in *2010 IEEE/IFIP International Conference on Dependable Systems Networks (DSN)*, June 2010, pp. 477–486.

[69] A. Benso, S. Di Carlo, G. Di Natale, P. Prinetto, and L. Taghaferri, "Data criticality estimation in software applications," in *International Test Conference, 2003. Proceedings. ITC 2003.*, vol. 1, Sep. 2003, pp. 802–810.

[70] V. Sridharan and D. Kaeli, "Eliminating microarchitectural dependency from architectural vulnerability," in *High Performance Computer Architecture, 2009. HPCA 2009. IEEE 15th International Symposium on*, Feb 2009, pp. 117–128.

[71] V. Sridharan and D. R. Kaeli, "Using pvf traces to accelerate avf modeling," in *Proceedings of the IEEE Workshop on Silicon Errors in Logic System Effects*, 2010.

[72] A. Savino, S. Di Carlo, G. Politano, A. Benso, A. Bosio, and G. Di Natale, "Statistical reliability estimation of microprocessor-based systems," *IEEE Transactions on Computers*, vol. 61, no. 11, pp. 1521–1534, 2012.

[73] A. Vallero, A. Savino, S. Tselonis, N. Foutris, M. Kaliorakis, G. Politano, D. Gizopoulos, and S. Di Carlo, "Bayesian network early reliability evaluation analysis for both permanent and transient faults," in *2015 IEEE 21st International On-Line Testing Symposium (IOLTS)*, July 2015, pp. 7–12.

[74] J. Henkel, L. Bauer, N. Dutt, P. Gupta, S. Nassif, M. Shafique, M. Tahoori, and N. Wehn, "Reliable on-chip systems in the nano-era: Lessons learnt and future trends," in *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, May 2013, pp. 1–10.

[75] J. Henkel, L. Bauer, H. Zhang, S. Rehman, and M. Shafique, "Multi-layer dependability: From microarchitecture to application level," in *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2014, pp. 1–6.

[76] A. Vallero, A. Savino, G. Politano, S. D. Carlo, A. Chatzidimitriou, S. Tselonis, M. Kaliorakis, D. Gizopoulos, M. Riera, R. Canal, A. Gonzalez, M. Kooli, A. Bosio, and G. D. Natale, "Cross-layer system reliability assessment framework for hardware faults," in *2016 IEEE International Test Conference (ITC)*, Nov 2016, pp. 1–10.

[77] A. Vallero, A. Savino, A. Chatzidimitriou, M. Kaliorakis, M. Kooli, M. Riera, M. Anglada, G. Di Natale, A. Bosio, R. Canal, A. Gonzalez, D. Gizopoulos, R. Mariani, and S. Di Carlo, "SyRA: Early System Reliability Analysis for Cross-Layer Soft Errors Resilience in Memory Arrays of Microprocessor Systems," *IEEE Transactions on Computers*, vol. 68, no. 5, pp. 765–783, May 2019.

[78] M. Riera, R. Canal, J. Abella, and A. Gonzalez, "A detailed methodology to compute soft error rates in advanced technologies," in *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*. EDA Consortium, 2016, pp. 217–222.

[79] S. Tselonis and D. Gizopoulos, "Gufi: A framework for gpus reliability assessment," in *2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, April 2016, pp. 90–100.

[80] E. Cheng, S. Mirkhani, L. G. Szafaryn, C.-Y. Cher, H. Cho, K. Skadron, M. R. Stan, K. Lilja, J. A. Abraham, P. Bose, and S. Mitra, "CLEAR: Cross-Layer Exploration for Architecting Resilience - Combining Hardware and Software Techniques to Tolerate Soft Errors in Processor Cores," in *Proceedings of the 53rd Annual Design Automation Conference*, ser. DAC '16. New York, NY, USA: ACM, 2016, pp. 68:1–68:6. [Online]. Available: http://doi.acm.org/10.1145/2897937.2897996

[81] A. Savino, A. Vallero, and S. Di Carlo, "ReDO: Cross-Layer Multi-Objective Design-Exploration Framework for Efficient Soft Error Resilient Systems," *IEEE Transactions on Computers*, vol. 67, no. 10, pp. 1462–1477, Oct 2018.