Microprocessors and Microsystems 39 (2015) 1204–1214

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

Cross-layer reliability evaluation, moving from the hardware architecture to the system level: A CLERECO EU project overview

A. Vallero [a], S. Tselonis [b], N. Foutris [b], M. Kaliorakis [b], M. Kooli [c], A. Savino [a], G. Politano [a], A. Bosio [c], G. Di Natale [c], D. Gizopoulos [b], S. Di Carlo [a,*]

[a] *Department of Control and Computer Engineering, Politecnico di Torino, Torino, Italy*
[b] *Department of Informatics and Telecommunications, University of Athens, Athens, Greece*
[c] *LIRMM (Université Montpellier/CNRS UMR 5506), Montpellier, France*

ARTICLE INFO

Article history:
Received 23 December 2014
Revised 11 May 2015
Accepted 4 June 2015
Available online 20 June 2015

Keywords:
Reliability evaluation
Fault injection
Statistical models

ABSTRACT

Advanced computing systems realized in forthcoming technologies hold the promise of a significant increase of computational capabilities. However, the same path that is leading technologies toward these remarkable achievements is also making electronic devices increasingly unreliable. Developing new methods to evaluate the reliability of these systems in an early design stage has the potential to save costs, produce optimized designs and have a positive impact on the product time-to-market.

CLERECO European FP7 research project addresses early reliability evaluation with a cross-layer approach across different computing disciplines, across computing system layers and across computing market segments. The fundamental objective of the project is to investigate in depth a methodology to assess system reliability early in the design cycle of the future systems of the emerging computing continuum. This paper presents a general overview of the CLERECO project focusing on the main tools and models that are being developed that could be of interest for the research community and engineering practice.

1. Introduction

Most things we rely on in our everyday life contain electronic-based information and have enough computing power to run embedded software applications, which connect to the Internet and remote advanced computing services to get access to virtually unlimited resources. This future computing continuum, composed of a wide set of heterogeneous platforms, promises to be a fertile environment to engineer advanced services with high added value.

Radiation effects, wear-out, aging and variability throughout the operational period of a system, extreme scaling processes that move towards 12 nm manufacturing process nodes and beyond, the high design complexity, and a fast time-to-market demand are expected to make system components extremely unreliable. As an example, the single bit error rate of a six-transistor SRAM that is in the order 1.5×10^{-6} for a 22 nm technology is expected to increase up to 5.5×10^{-5} in 16 nm technology and 2.6×10^{-4} in 12 nm technology [1].

From a reliability perspective, system designers have to meet precise reliability requirements. These requirements are highly domain dependent and are influenced by the criticality of the considered system or component (e.g., aerospace and medical applications require very low failure rates). Reliability is therefore increasingly driving several design decisions at the technology, hardware and software level.

Error management solutions at all design/implementation levels are feasible. Technology can be hardened [2–8], hardware architectures may include redundancy [9–19], and finally all software layers may implement error detection and recovery mechanisms [20–26]. On the one hand, this enables designers to apply cross-layer holistic design approaches to manage errors in their systems. On the other hand, this enlarges the design space making design optimization complex.

Nowadays, the dominant approach to design reliable systems consists in worst-case design. However, it is well known that several reliability-oriented design decisions lead to costs in terms of area, complexity, performance and energy budget [27]. Reliability engineers and system architects need to be provided with adequate tools to cope with this complexity and to take design decision able to enable reliability targets to be met with minimum cost.

[*] Corresponding author.Tel.:+39 011 090 7080; fax:+39 011 090 7099.
E-mail address: stefano.dicarlo@polito.it (S. Di Carlo).

http://dx.doi.org/10.1016/j.micpro.2015.06.003
0141-9331/© 2015 Elsevier B.V. All rights reserved.

Moreover, these decision must be taken as early as possible in the design process when redesign and optimizations are still affordable. Products failing to reach the reliability objectives in the late stage of the design may lead to commercial failure with severe economical consequences.

Current reliability analysis approaches strongly rely on massive and time-consuming RTL fault injection campaigns, which are becoming a bottleneck due the increasing complexity of computing systems. Simulating a complete system composed of microprocessors and accelerators embedding several tens of processing cores and memory blocks, and executing complex applications is becoming prohibitive. Fault injection at the RTL level can require several months of CPU time. This strongly impacts the project TTM and poses a serious threat on the success of a product in case the target reliability levels are not reached and redesign of part of the system is required. Moreover RTL fault injection requires a full system already designed and can be applied only in the late stages of the design process. At these stages, design modifications to improve reliability are excessively costly.

The FP7 Collaboration Project CLERECO addresses early system reliability evaluation with a cross-layer approach [28–30]. The fundamental objective of the project is to investigate methodologies to accurately perform system reliability analysis focusing on the early stages of the design cycle for the future systems of the emerging computing continuum [31].

This paper presents an overview of the CLERECO project at the end of the first year of its research activity. It focuses on the tools that are being developed that could be of interest for the research community and engineering practice. Given the limited space, the paper does not provide detailed descriptions of all developed models and tools. The emphasis of the paper is instead to present CLERECO's perspective on the way system reliability analysis can be performed with a cross-layer approach considering the main layers that constitute a modern digital system.

The paper is organized as follows: Section 2 introduces the cross-layer approach to evaluate the system reliability. Section 3 overviews CLERECO's general methodology to perform system reliability analysis, and Sections 4 and 5 describe tools to evaluate the hardware and software contribution to the system reliability. Eventually, Section 6 presents final considerations and future perspectives for the system reliability estimation.

2. A cross-layer approach for system reliability evaluation

Performing cross-layer system reliability analysis, requires a deep understanding of the layers where faults appear in the system, how faults generate errors, and how errors propagate across layers, eventually impacting the final mission of the system.

Fig. 1 provides a graphical representation of how faults may be generated and propagated in a system. Following the Computing Community Consortium Visioning Study on Cross-Layer Reliability [32], a system can be seen as a stack go three main layers:

1. the technology layer that accounts for the raw technology used to build its hardware blocks,
2. the hardware layer that accounts for the hardware blocks and their architectures built on top of the technology, and
3. the software layer that includes the system and user applications executed on the hardware platform.

The technology used to build hardware components is the main root of hardware faults due to physical fabrication defects, aging or degradation (e.g., NBTI), environmental stress (e.g., radiations), and fabrication variability, etc. Within CLERECO we focus on how these faults propagate through the other layers composing the system. After a raw fault manifests in a hardware block, it can be propagated through the different hardware structures composing the system. Several masking effects can mitigate the impact of these faults. We define as *vulnerability factor* the conditional probability of a component to produce an erroneous result given the occurrence of a raw error in one of the lower layers of the design hierarchy. Several vulnerability factors do exist in a system.

Faults can be mitigated at the technology level by timing effects that prevent erroneous values to be sampled by memory elements (Time Vulnerability Factor – TVF) [33–35], or by logic masking effects (Cell Vulnerability Factor – CVF). Faults that manage to cross the technology layer and enter the hardware architecture layer can still be masked both at the micro-architecture level (μ Architecture Vulnerability Factor – μAVF) or at the architecture level (Architecture Vulnerability Factor – AVF) [36–38]. Finally those faults that are not masked at the hardware layer enter the software layer of the system by corrupting either data or instructions of software applications. These errors can damage the correct software execution by producing erroneous results if the computation is completed, or by preventing the execution of the application by causing exceptions, interrupts, abnormal terminations or applications hang-up. Nevertheless, the software stack can also play an important role in masking errors, introducing a further error masking effect (Software Vulnerability Factor – SVF), which may further improve the system reliability [39–46].

Performing system reliability analysis means calculating the different vulnerability factors associated with the components of a system, and then understanding how all masking effects work together and how they influence the behavior of the system. Fig. 2 provides a high-level view of the CLERECO cross-layer reliability evaluation flow. The key concept exploited in CLERECO is to analyze the three system layers separately computing different vulnerability factors for the different blocks. Vulnerability factors are then statistically combined in order to infer reliability measures at the system level. Analyzing the layers in isolation has the main advantage to reduce the complexity of the analysis focusing on the peculiar masking effects each layer can provide. As reported in Fig. 2, each layer defines an interface with the upper layer, which in turn sets how faults can be propagated from one layer to the next one. For each layer, in CLERECO, we devise to identify a set of tools and models able to perform this characterization.

Among the three layers composing the system, the technology layer is probably the most well studied layer. Studying faults that may affect new technologies such as FinFET [47], starts from the definition of predictive models [48] for the technology and requires the development of models for the basic cells (e.g., memory cells, boolean gates, etc.) that need to be analyzed. Resorting to these models, electrical simulations (e.g., Spice, TCAD) can be used to compute failure probability and to derive the TVF that will be required for the analysis of the next layers of the stack.

In this paper we focus on the vulnerability factors introduced by microprocessors and software routines and on the statistical models used to combine those vulnerability factors.

3. System level reliability modeling

Early system reliability analysis requires the identification of a proper high level statistical model enabling to represent the system and its vulnerability factors and to perform statistical reasoning.

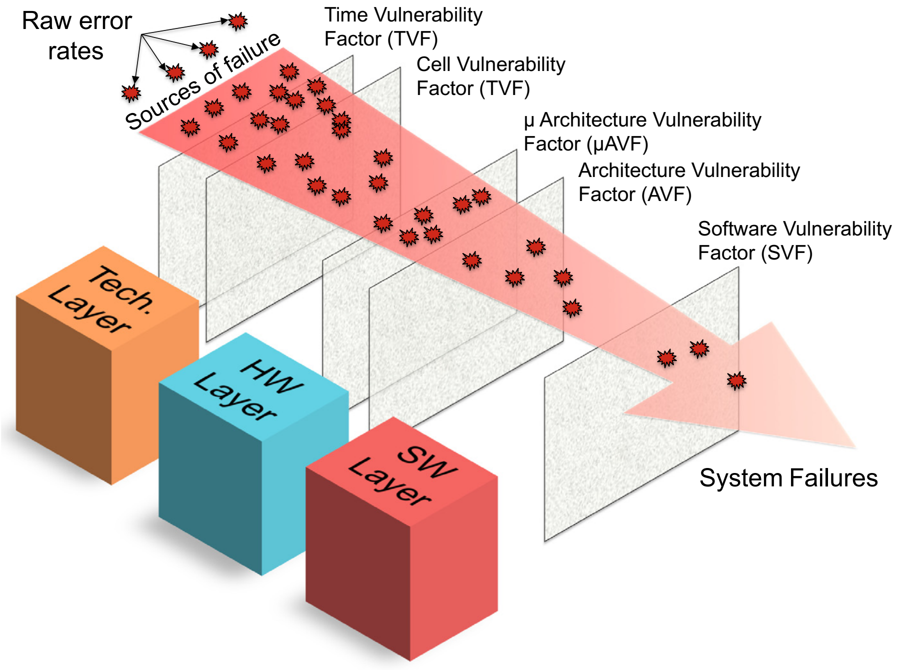

Raw error rates
Sources of failure
Time Vulnerability Factor (TVF)
Cell Vulnerability Factor (TVF)
μ Architecture Vulnerability Factor (μAVF)
Architecture Vulnerability Factor (AVF)
Software Vulnerability Factor (SVF)
Tech. Layer
HW Layer
SW Layer
System Failures

Fig. 1. Cross-layer error propagation. Faults manifests in the technology and then propagate through the hardware and software layers. During this propagation different masking effects may block the error propagation thus reducing their impact on the final system's reliability.

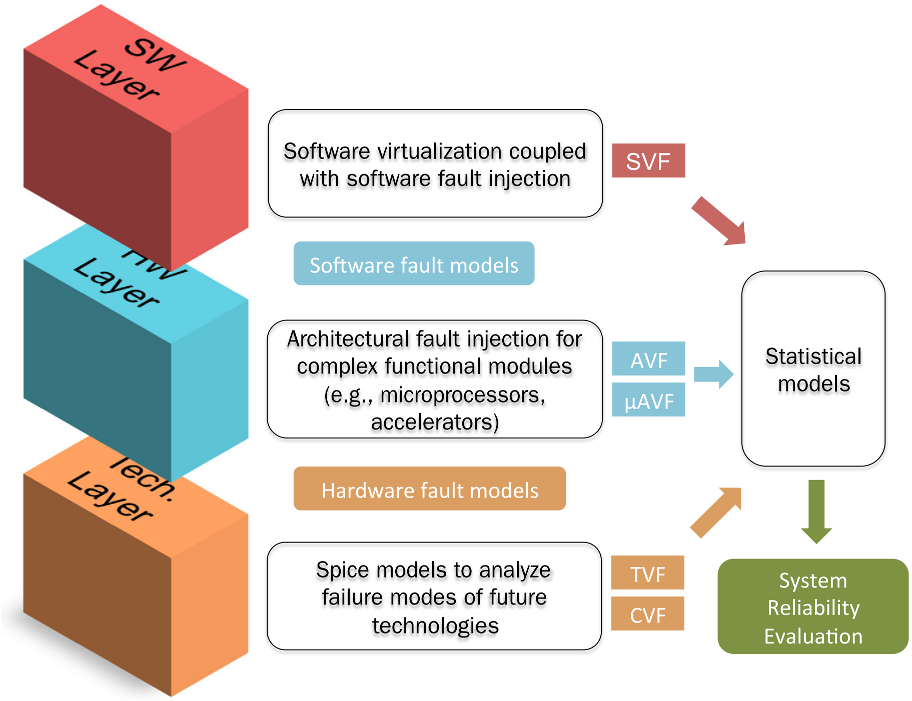

SW Layer
HW Layer
Tech. Layer
Software virtualization coupled with software fault injection
SVF
Software fault models
Architectural fault injection for complex functional modules (e.g., microprocessors, accelerators)
AVF
μAVF
Statistical models
Hardware fault models
Spice models to analyze failure modes of future technologies
TVF
CVF
System Reliability Evaluation

Fig. 2. CLERECO cross-layer reliability evaluation flow.

Fault Tree Analysis (FTA) is a very common statistical reliability analysis [49]. FTA is a top down, deductive failure analysis in which an undesired state of a system is analyzed using Boolean logic to combine a series of lower-level events. It is mainly used in the fields of safety engineering and reliability engineering to understand how systems can fail. Another very similar technique that is usually employed to statistically investigate the reliability of a system is Reliability Block Diagram (RBD). The most fundamental difference between FTD and RBD is that RBD works in the "success space", and thus looks at system successes combinations, while FTD works in the "failure space" and looks at system failure combinations. Both FTD and RBD do not enable to model reliability interactions among components or subsystems, or to represent system reliability configuration changes [50].

Markov chains represent a significant alternative to FTD or RBD analysis [51]. A Markov chain is a random process that undergoes transitions from one state to another on a state space. The probability distribution of the next state only depends on the current state (Markov property). Markov chains have several modeling issues when applied to reliability analysis. First, the whole system is modeled as a set of states, which may explode in complex systems. Second, the Markov property limits the possibility to fully analyze the propagation of errors among states.

Recently Bayesian Networks (BN) are gaining interest in modeling system reliability in hardware devices [52]. BNs are a statistical model to represent multivariate statistical distribution functions. They can model relationships among random variables and their respective probability density functions by means of conditional probability functions. They main advantage with respect to the previous techniques is the degree of freedom they have to define input causes of failure. The system can be described in terms of blocks and not just states. Blocks can be studied locally to populate the model, leaving the analysis of the interaction of the blocks to a high-level statistical reasoning. Bayesian networks have been selected in CLERECO as basic model to build early system reliability analysis.

Fig. 3 shows a very simplified example of Bayesian system modeling. It is important to highlight that this is not an example of a real model. Real models can account for up to hundreds of nodes and arcs. However, this simplified example can be used to introduce the basic modeling concepts and to show how statistical reasoning on system reliability can be implemented.

System modeling using a BN starts by identifying the networks nodes. Network nodes model the components of the system. They can be split into technology nodes, hardware nodes and software nodes following the three layers introduced in Fig. 2. Each node is associated to a state. The state space can be either discrete or continue. For the sake of simplicity in this example we simply consider a binary state space including the *failure* and *success* state.

The technology layer plays an active input role. Technology nodes are the root of the system model. Each node identifies a specific technology process used to build a certain hardware block embedded in the system. Each technology node is characterized by the raw failure probability with respect to the target fault model (e.g., soft error rate). It can be either provided by manufacturers or estimated through electrical simulations on predictive models.

Hardware nodes are intermediate nodes of the network that define the hardware resources assembled together to compose the hardware infrastructure. Each hardware node is connected to a technology node to model the relationship between the hardware architecture and the underlying technology. Moreover, hardware nodes can be connected to other hardware nodes to model error propagations or masking between components. Each intermediate node is characterized by a Conditional Probability Table (CPT). This table considers all possible combinations of states of the parents nodes (instantiations). For each distinct instantiation of parent nodes, the CPT defines the probability of the node to be in a certain state (e.g., failure or success) given the instantiation of the parents nodes.

In order to decouple the analysis of the software nodes from the one of the hardware nodes special attention is required to define the interface between the two layers. In CLERECO we have identified a set of high level Software Fault Models (SFMs), which model how hardware errors propagate to software (see Section 5). They mainly rely on alterations that have an impact on the Instruction Set Architecture (ISA) of the microprocessor. They are modeled as additional nodes that represent an intermediate layer between the hardware nodes and the software nodes (e.g., wrong data in operand node in Fig. 3).

Eventually, the software nodes are the final players of the system model. Errors generated in the technology can be propagated up to the software. At this level they can further propagate during the computation, mainly based on how the software manipulates data or on the way the flow of execution follows its proper path. Software nodes represent software functions or portions of software functions. Arcs at this level model propagation of errors

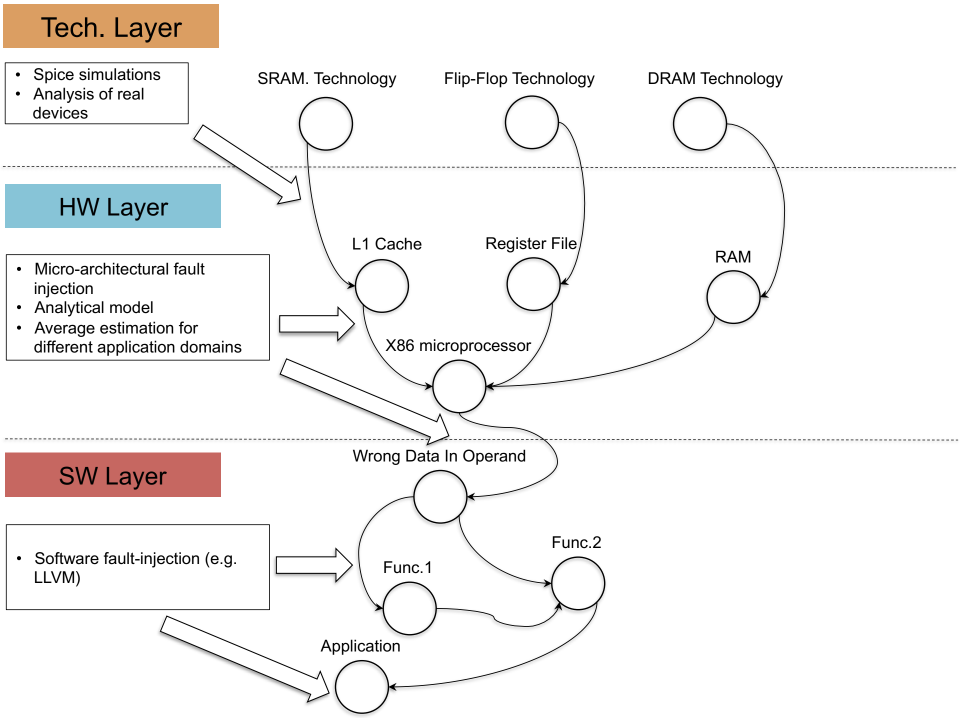


Fig. 3. System level reliability evaluation using Bayesian a Bayesian model of the system. Nodes of the model represent system resources (e.g., technology, hardware, software), while edges represent how these resources influence each other when faults arise in the system.

among different portions of the software. Again each node must be characterized using a CPT expressing the failure probability of the node given the instantiations of the parent nodes.

Once a Bayesian system reliability model is built, the model can be used to reason about the overall reliability properties of the system.

Bayesian networks provide full representations of probability distributions over their variables. That implies that they can be conditioned upon any subset of their variables, supporting any direction of reasoning. For example, one can perform diagnostic reasoning, i.e., reasoning from symptoms to cause, such as when we observe a failure in the application, we can update our belief about the contribution of each node to this failure, thus identifying those nodes that most likely contribute to the failure. Note that this reasoning occurs in the opposite direction to the network arcs. Differently, one can perform predictive reasoning, starting from the information about causes (i.e., raw technology failure rates) to new beliefs about effects (i.e., application failures), following the directions of the network arcs. Statistical reasoning resorting to Bayesian models is a well known statistical approach and the reader may refer to [53] for more detailed descriptions.

By resorting to the proposed high-level statistical reasoning, system designer are provided with a tool enabling to perform early estimation of the overall system reliability. Moreover, using diagnostic reasoning, weak components can be probabilistically identified. This provides means to drive the reliability design effort toward the most critical components of the system thus optimizing the overall system. Nevertheless, computing the conditional probabilities that populate the Bayesian model is still a complex task that requires dedicated tools to be completed. These probabilities represent the vulnerability factors introduced in Section 1. Next sections overview two of the main tools developed in CLERECO to accomplish this task.

4. Evaluating the hardware contribution to system's reliability through micro-architectural simulation

Functional modules such as microprocessors, accelerators (e.g., GPUs, APUs) and memory controllers represent the most complex hardware components of modern digital system. Therefore, they are likely to provide a major impact on the hardware vulnerability factors of a system. For this reason creating tools able to characterize them in the framework of the presented system reliability model is one of the most critical tasks considered in CLERECO.

In general, there are two categories of tools that enable to study these complex modules:

1. RTL-simulators, and
2. micro-architectural simulators.

RTL simulators enable to consider several circuit-level characteristics facilitating accurate hardware reliability estimations. However, their low simulation throughput is a limiting factor.

Micro-architectural simulators have the ability of executing faster simulations than RTL simulators. It is widely known that many important components modeled in micro-architectural simulators have a very direct relation to the actual hardware implementation. Such components are mainly storage-related components like DRAMs, SRAMs (caches), registers and register files, buffers and queues. In a micro-architectural simulator these structures are implemented as single or double-dimensional arrays of variables in the programming language used to implement the simulator. It is widely known that many important components modeled in micro-architectural simulators have a very direct relation to the actual hardware implementation. Fault injection analysis through micro-architectural simulators for these structures very closely model the hardware components and a major effort has been devoted in the project for the development of these type of tools. On the other hand, control logic blocks and functional components are very simply implemented on architectural simulators and different approaches are under investigation to compensate for this inaccuracy.

MAFin is the CLERECO fault injector framework created on top of the MARSSx86 micro-architecture level simulator [54]. MARSSx86 [55] is a full system simulator built on top of PTLsim simulator [56] and incorporating the QEMU emulator. PTLsim simulates the details of an x86 microprocessor while QEMU provides to MARSSx86 its full system capabilities.

We selected MARSSx86 as the base of the development of our fault injector tool because it is a full system simulator that models accurately x86 architectures (cycle accurate), it is publicly available and regularly supported. Moreover, MARSSx86 models both a complex out-of-order and a simpler in-order (Atom-like model) single core architecture, as well as multicore x86 architectures. Thus, the features and capabilities of the original MARSSx86 model coupled with our extensions cover the characterization of microprocessors employed in both High Performance Computing and Embedded Computing and based on the widespread x86 ISA.

The fault injection infrastructure provides users with the capability of tracing the propagation of a fault in a hardware structure at the micro-architectural level, till its manifestation at the ISA, operating system or application level [54].

Fig. 4 summarizes all structures at the micro-architectural level that can be studied through fault injection. A complete reliability study for all storage arrays is fully supported. These storage arrays are significantly more vulnerable to faults than control logic (in particular for transient faults). Moreover, the issue queues, i.e., the data-structures which facilitate the out-of-order execution in modern microprocessors can be studied as well.

4.1. Fault models

The fault injection infrastructure enables injection of both single faults and multiple faults. Single faults can be transient, intermittent and permanent. Multiple faults can be every possible combination of single faults in the form of spatial faults or temporal faults. For instance, spatial faults may represent the effect of a single particle strike that flips the state of multiple storage elements on a contiguous rectangle or square. Temporal faults may instead be the effect of multiple but independent single-event upsets that are distributed over time.

Transient faults are modeled by flipping (XOR) the value of a randomly selected bit in a randomly selected clock cycle during simulation. Intermittent faults are modeled by setting the state of storage elements to 1 (OR) or 0 (AND), in a randomly selected cycle, for a random period. Permanent faults are modeled by setting persistently to 1 (OR) or to 0 (AND) the value of a randomly selected storage element for the entire simulation time.

In general, the set of fault models considered by the injector is flexible enough to reproduce a wide set of real fault models identified during the characterization of the technology layer.

4.2. Fault injection framework

Fig. 5 shows the general architecture of MaFIN.

For the development of our tool, we extended the original version of MARSSx86 by adding data arrays in caches as well as by making all the necessary extensions to support comprehensive fault injection campaigns. Our extensions increase the simulation time but these modifications are necessary to realize fault injections in these important missing arrays. Overhead in simulation

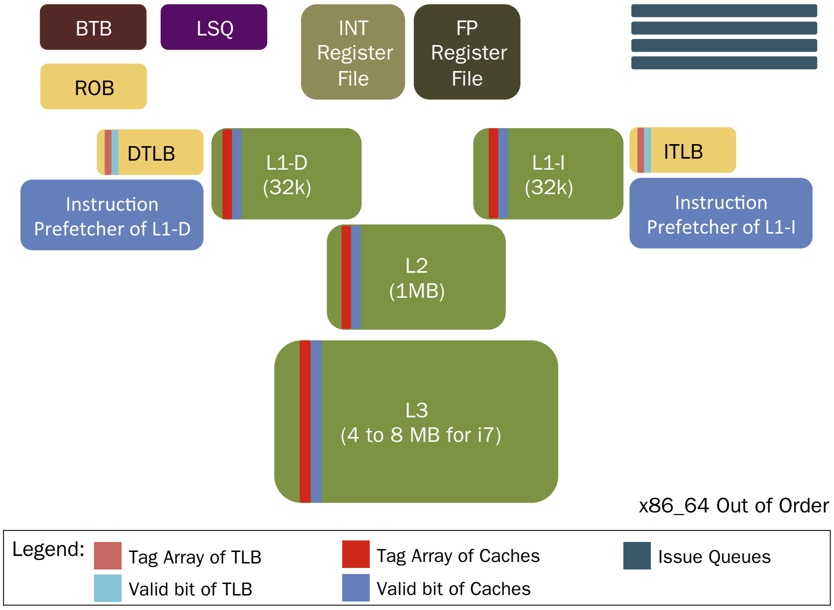


Fig. 4. Structures at the micro-architectural level that can be studied through fault injection. All structures can be arbitrary sized in order to meet the requirements of the final design.

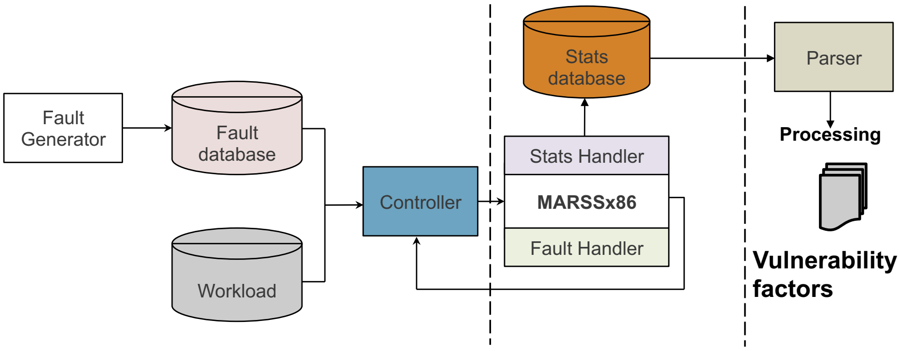


Fig. 5. MaFIN high-level block diagram.

time due to the fault injection infrastructure was kept to a minimum to enable fast vulnerability factors computation. To perform a micro-architectural fault injection campaign for a given workload, four steps are required to obtain the desired results.

Step 1. This is a preliminary step used to create a so called checkpoint `chk_x` for the given workload. A checkpoint includes the output as well as other parameters (e.g., execution time) related to the execution of the workload without injecting any fault. `chk_x` represents the golden execution used to classify the effect of the injected faults as reported in Step 4.

Step 2. After `chk_x` is calculated, the Fault Generator can perform its task. It is executed every time a new injection campaign must be performed. Its main goal is to generate a database of faults (fault database) according to the fault injection requirements. More specifically, every single injection experiment is characterized by a fault mask that is built according to the target fault model. The fault mask embeds several fault attributes including:

1. `processor_id`: the targeted processor in a multicore architecture.
2. `module_id`: the targeted micro-architecture module in the processor.
3. `fault_mask`: the set of bits that may change the value in a storage array.
4. `fault_type`: transient, intermittent and permanent. It determines the type of bitwise operation that is executed between the fault masks and the targeted storage arrays (i.e., for stuck-at-0 AND, for stuck-at-1 OR, for bit flip XOR). In the case of multiple faults,different sets of fault mask bits do exist.
5. `duration`: used in the case of intermittent faults to specify for how many clock cycles the fault is active.
6. `activation cycle`: the point in time during the simulation in which the fault is injected in the targeted structure.

As soon as a fault mask for a single experiment is generated, it is stored into a database containing all fault masks needed for the fault injection campaign. The Fault mask generation process is very flexible since it allows to generate a large set of fault models.

Step 3. At this point, the fault injection campaign is ready to start. The Fault Handler and the Stats Handlers are responsible for two independent processes:

1. The Fault Handler manages the fault injection campaign, i.e., it manages the fault injection experiments. Each fault injection experiment consists in the execution of the

workload during which faults are injected according to fault masks. In details, the set of fault mask attributes are passed through the Injection Interface from the fault mask database to the extended MARSSx86 simulator.

2. The Stats Handler manages the collection of information related to fault injection experiments required to classify the outcome of the system. In particular, it collects the following files for any given fault injection experiment: (i) the output of the application, (ii) the file that includes the redirected `std_err`, (iii) the file of statistics, (iv) the file with logs.

   The injection campaign ends when all fault injection experiments have been executed to the end. At this point, the most time consuming part of the overall process is complete.

Step 4. Last step is to establish the final outcome of each fault injection experiment so that statistics about fault injection can be collected. More specifically the Parser is responsible for comparing the files provided by the Stats Handler with the checkpoint `chk_x`. The outcome of each experiment is classified according to the categories reported in Table 1 and vulnerability factors for the different structures can be computed based on these results and used to feed the presented system reliability model.

5. Evaluating the software contribution to system's reliability through software virtualization

The software stack plays an important role in masking errors, thus enabling to improve the system reliability. In order to decouple the analysis of software masking probabilities from the target hardware architecture, therefore enabling refuse of computed statistics, we need to investigate methods and tools to:

- describe the software independently from the target hardware architecture, and
- study how errors in the hardware resources propagate through the software routines and possibly impact the correct behavior of the applications.

Fig. 6 summarizes the main concepts used in CLERECO to analyze the software vulnerability factor of a system.

When analyzing software applications independently from the target hardware layer, the Instruction Set Architecture (ISA) used to encode the program, which represents the main link between the hardware and the software domain, is still undefined. It cannot therefore be exploited to analyze the fault propagation through the software stack.

In CLERECO, we rely on the concept of software virtualization as an efficient, flexible and cost saving solution to enable the abstraction of the ISA from the hardware layer. The software vulnerability factor will depend on a Virtual Instruction Set Architecture (VISA) used to describe complex programs, as well as on a set of Software Fault Models describing the way hardware errors can manifest in a program.

Resorting to these two basic building blocks, fault injection of software fault models in a virtual environment can be efficiently exploited to analyze how faults propagate through the software application, and how they impact the correctness of the results. It is worth remembering that performing fault injection campaigns at the software layer is far less computational intensive than performing similar campaigns at the architectural or RTL level. This enables us to characterize realistic applications in a limited computational time.

Table 1
Set of categories in which results of a fault injection experiment can be classified.

Hangs	The application does not terminate within a reasonable time interval. (we have set this interval to 3× of the execution time of the fault-free case)
SDC	The output of application has been corrupted
DUE	An unexpected exception, assertion, or segmentation fault, deadlock or interrupt occurred. Simulator crashes, either during simulation or emulation phase, are also clustered into this category
Masked	No mismatch at the application output

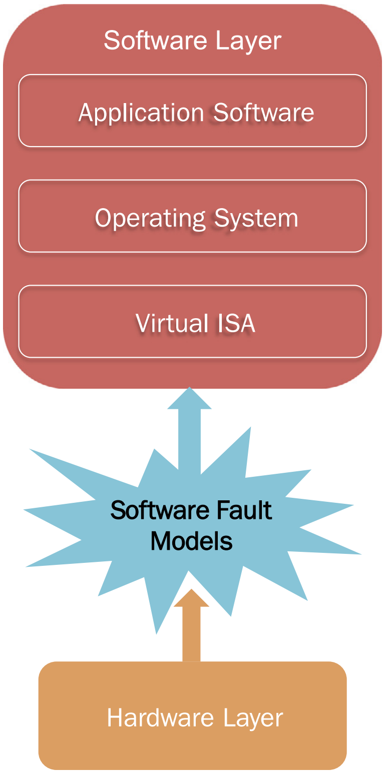


Fig. 6. Software stack.

5.1. Virtual instruction set architecture

Different virtualization systems implementing VISAs are available in the literature: Java [57], .NET [58]/Mono [59], and LLVM [60].

Java is widely used in web-based applications. However, it has the disadvantage of not being really suitable for both High Performance Computing and Embedded Applications. Moreover, the Java virtual machine is restricted to the Java programming language, thus limiting the spectrum of software that can be analyzed.

The .NET framework consists of a virtual machine able to execute programs written using the Common Language Infrastructure (CLI) defined by Microsoft and standardized by ISO and ECMA. To the best of our knowledge, no fault injection environment is actually available for this framework.

LLVM (Low Level Virtual Machine) is a framework that uses virtualization with Virtual Instruction Sets to perform complex analysis of full software applications on different architectures.

LLVM is a compiler framework designed to support transparent, life-long program analysis and transformation for arbitrary programs, by providing high-level information to compiler transformations at compile-time, link-time, run-time, and in idle-time between runs. In addition to the full tool chain required for software design (e.g., compiler, optimizer), LLVM provides a set of additional tools explicitly devoted to perform investigation of different software properties.

LLVM uses the Intermediate Representation (IR) as a form to represent code in the compiler. It symbolizes the most important aspect of the framework, because it is designed to host mid-level analysis and transformations found in the optimizer section of the compiler. The LLVM IR is independent from the source language and the target machine. It is easy for a front-end to generate, and expressive enough to permit important optimizations to be performed for real targets.

Given these premises LLVM represents a very promising virtualization platform for the analysis of the Software Vulnerability Factor of complex applications. It has therefore been selected as target VISA for the CLERECO project.

5.2. Software fault models

Research approaches that try to consider the impact of software in the reliability of a full system still start from low level hardware faults [61,62], trying to propagate them through the hardware architecture to the software layers in order to evaluate their impact on the final system outcome [63–65]. This propagation method requires complex and time consuming simulations of hardware models that do not enable to analyze complex software stacks.

CLERECO software analysis approach starts from a set of software fault models defined at the VISA level that can be directly linked to the effect of faults arising at the hardware level. Table 2 reports a preliminary list of identified Software Fault Models considered in CLERECO. Software fault models can be grouped in three main categories:

- *Data fault models*: they enable to model errors corrupting data processed by a software application. They include: (i) *Wrong Data in a Operand*, (ii) *Not-accessible Operand*, and (iii) *Operand Forced Switch*.
- *Code fault models*: they enable to model errors that corrupt the set of instructions composing a program. They include: (i) *Instruction Replacement*, (ii) *Faulty Instruction*, (iii) *Control Flow Error*.
- *System fault models*: they enable to model both timing errors and communication/synchronization errors during the software execution. They include: (i) *External Peripheral Communication Error*, *Signaling Error*, *Execution timing Error*, *Synchronization Error*.

5.3. LLVM based fault injector

LLVM already comprises two projects aimed at developing LLVM based fault injectors: (i) LLFI [66,67] and (ii) KULFI [68]. Nevertheless, both projects do not fit CLERECO requirements. Their main limitation is the set of considered fault models, which are mainly limited to bit-flips in the microprocessor and do not consider high level fault models as the one defined in Table 2. Moreover, both LLFI and KULFI define the outcome of the fault simulation as the impact of the hardware faults on the whole system while in CLERECO we are interested on evaluating the impact of the defined software fault models on the software execution, decoupling this analysis from the hardware architecture.

Table 2
Software Fault Models.

Software Fault Model	Description
Wrong Data in a Operand	An operand of the ISA instruction changes its value
Not-accessible Operand	An operand of the ISA instruction cannot change its value
Source Operand Forced Switch	An operand is used in place of another
Instruction Replacement	An instruction is used in place of another
Faulty Instruction	The instruction is executed incorrectly
Control Flow Error	The control flow is not respected (control-flow faults)
External Peripheral Communication Error	An input value (from a peripheral) is corrupted or not arriving
Signaling Error	An internal signaling (exception, interrupt, etc.) is wrongly raised or suppressed
Execution timing Error	An error in the timing management (e.g. PLL) interferes with the correct execution timing
Synchronization Error	An error in the scheduling processes causes an incoherent synchronization of processes/tasks

We therefore designed an ad hoc fault injection infrastructure on top of the LLVM virtualization framework. The tool is able to process the following information items:

- The original target source code written in any programming language supported by LLVM [69] (e.g., C, C++, Objective-C, Fortran, Python).
- An input file containing the fault injection parameters: the set of software fault models to inject and the corresponding number of simulations to perform, as well as the list of variables to monitor after the injection and their corresponding location.
- A set of parameters that enable to tune how the simulation results are compared against a golden execution to identify classes of faulty software behaviors.

As an output the tool provides a set of statistics on the identified *software faulty behaviors* as classified in Table 1.

Fig. 7 presents the structure of the fault injector environment. Starting from the original source code, the tool generates the LLVM code that is used in the whole process of injection and analysis. Three main steps are then performed.

Step 1. Software fault models are injected into the LLVM code respecting the input parameters. A LLVM file representing a faulty program is generated for each fault.

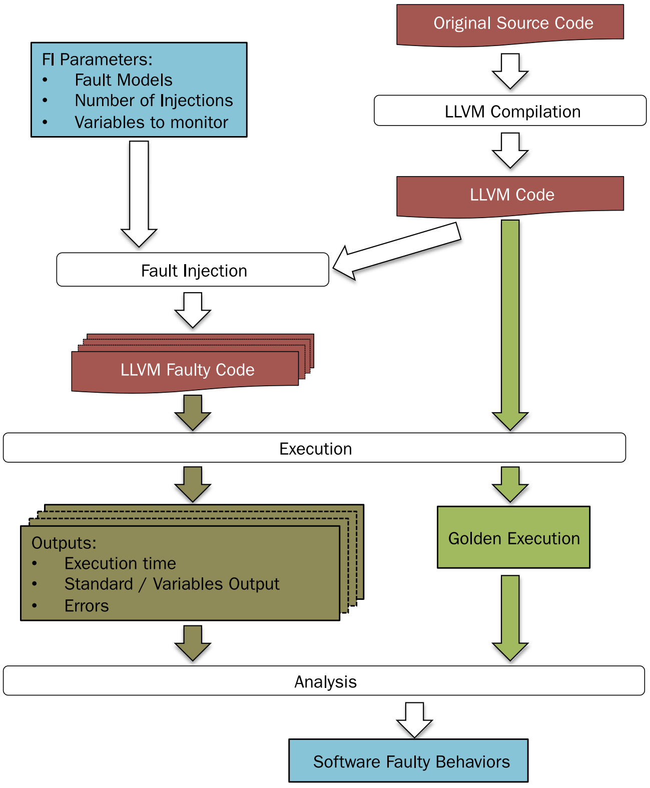


Fig. 7. Design of the fault simulator.

Step 2. The faulty programs are executed and each output is saved in a log file. Also the original LLVM code is executed and its output is redirected to a golden file.

Step 3. The final step is the analysis, where the log files corresponding to the outputs of the fault injection are compared with the golden output in order to evaluate the different software behaviors. Based on these statistics, vulnerability factors for the different software functions can be easily computed and eventually used to populate the system level reliability model proposed in Section 3.

6. Conclusions

Reliability is a key challenge for the next generation computing systems, and its precise evaluation in the early stage of the design process is pivotal for the design of high optimized and efficient future systems.

Current tools and models are still not mature to provide early reliability evaluations for a large set of applications as the ones that will be expected in the upcoming computing continuum. By closing this gap, significant improvements in the products performance and quality will be expected.

In this paper we have presented a very high level overview of the preliminary achievements obtained by the FP7 CLERECO project. We are aware that, due to limits in space several technical details on the specific methodology could not be included. Nevertheless, the paper should give the reader an indication on the roadmap followed in the project to implement cross-layer early reliability evaluation tools. Results presented in this paper are related to the first year of activity of the project. Several activities are still on-going and new tools and models are expected to be delivered in the upcoming months in order to build a full framework enabling reliability evaluation starting from the technology up to the full system.

To conclude we would like to emphasize that CLERECO solutions are not intended to replace reliability validation techniques employed at the end of the design to assess the final reliability of a product before its commercialization (e.g., stress tests, radiation tests, etc.). Instead they work at the beginning of the design cycle to help reliability engineers taking decisions able to optimize the designed systems and to increase the probability of success of the designed products.

Acknowledgments

This paper has been fully supported by the 7th Framework Program of the European Union through the CLERECO Project, under Grant Agreement 611404.

References

[1] S.R. Nassif, N. Mehta, Y. Cao, A resilience roadmap, in: Proceedings of the Conference on Design, Automation and Test in Europe, European Design and Automation Association, 2010, pp. 1011–1016.

[2] S. Krishnamohan, N.R. Mahapatra, Analysis and design of soft-error hardened latches, in: Proceedings of the 15th ACM Great Lakes symposium on VLSI, 2005, pp. 328–331.

[3] M. Hosseinabady, P. Lotfi-Kamran, G. Di Natale, S. Di Carlo, A. Benso, P. Prinetto, Single-event upset analysis and protection in high speed circuits, in: Eleventh IEEE European Test Symposium, 2006, ETS '06., IEEE, 2006, pp. 29–34.

[4] R. Rodríguez-Montañés, D. Arumí, S. Manich, J. Figueras, S. Di Carlo, P. Prinetto, A. Scionti, Defective behaviour of an 8t sram cell with open defects, in: 2010 Second International Conference on Advances in System Testing and Validation Lifecycle (VALID), 2010, pp. 81–86, http://dx.doi.org/10.1109/VALID.2010.19.

[5] E. Taylor, Overview of new and emerging radiation resistant materials for space environment applications, in: Aerospace Conference, 2011 IEEE, 2011, pp. 1–11, http://dx.doi.org/10.1109/AERO.2011.5747389.

[6] H. Villacorta, V. Champac, S. Bota, J. Segura, Finfet sram hardening through design and technology parameters considering process variations, in: 2013 14th European Conference on Radiation and Its Effects on Components and Systems (RADECS), 2013, pp. 1–7, http://dx.doi.org/10.1109/RADECS.2013.6937372.

[7] Z. Diggins, N. Gaspard, N. Mahatme, S. Jagannathan, T. Loveless, T. Reece, B. Bhuva, A. Witulski, L. Massengill, S.-J. Wen, R. Wong, Scalability of capacitive hardening for flip-flops in advanced technology nodes, IEEE Trans. Nucl. Sci. 60 (6) (2013) 4394–4398, http://dx.doi.org/10.1109/TNS.2013.2286272.

[8] M. McLain, D. Hughart, D. Hanson, M. Marinella, Effects of ionizing radiation on taox-based memristive devices, in: Aerospace Conference, 2014 IEEE, 2014, pp. 1–9, http://dx.doi.org/10.1109/AERO.2014.6836501.

[9] H. Ando, Y. Yoshida, A. Inoue, I. Sugiyama, T. Asakawa, K. Morita, T. Muta, T. Motokurumada, S. Okada, H. Yamashita, Y. Satsukawa, A. Konmoto, R. Yamashita, H. Sugiyama, A 1.3 GHz fifth generation sparc64 microprocessor, in: Proceedings of the 40th annual Design Automation Conference, 2003, pp. 702–705.

[10] C. Zambelli, M. Indaco, M. Fabiano, S. Di Carlo, P. Prinetto, P. Olivo, D. Bertozzi, A cross-layer approach for new reliability-performance trade-offs in mlc nand flash memories, in: Design, Automation Test in Europe Conference Exhibition (DATE), 2012, 2012, pp. 881–886, http://dx.doi.org/10.1109/DATE.2012.6176622.

[11] J. Guo, L. Xiao, Z. Mao, Q. Zhao, Novel mixed codes for multiple-cell upsets mitigation in static rams, IEEE Micro 33 (6) (2013) 66–74, http://dx.doi.org/10.1109/MM.2013.125.

[12] M. Fabiano, M. Indaco, S.D. Carlo, P. Prinetto, Design and optimization of adaptable BCH codecs for NAND flash memories, Microprocess. Microsyst. 37 (4–5) (2013) 407–419, http://dx.doi.org/10.1016/j.micpro.2013.03.002. <http://www.sciencedirect.com/science/article/pii/S0141933113000471>.

[13] B. Maric, J. Abella, M. Valero, Analyzing the efficiency of l1 caches for reliable hybrid-voltage operation using edc codes, Very Large Scale Integration (VLSI) Systems, IEEE Trans. 22 (10) (2014) 2211–2215, http://dx.doi.org/10.1109/TVLSI.2013.2282498.

[14] J. Guo, L. Xiao, Z. Mao, Q. Zhao, Enhanced memory reliability against multiple cell upsets using decimal matrix code, Very Large Scale Integration (VLSI) systems, IEEE Trans. 22 (1) (2014) 127–135, http://dx.doi.org/10.1109/TVLSI.2013.2238565.

[15] J. Walker, M. Trefzer, S. Bale, A. Tyrrell, Panda: a reconfigurable architecture that adapts to physical substrate variations, IEEE Trans. Comput. 62 (8) (2013) 1584–1596 (cited By 4) <http://www.scopus.com/inward/record.url?eid=2-s2.0-84880147752&partnerID=40&md5=5b62cc78d6c464cefa96d4d069d253f9>.

[16] A.-M. Rahmani, K. Vaddina, K. Latif, P. Liljeberg, J. Plosila, H. Tenhunen, High-performance and fault-tolerant 3d noc-bus hybrid architecture using arb-net-based adaptive monitoring platform, IEEE Trans. Comput. 63 (3) (2014) 734–747 (cited By 1) <http://www.scopus.com/inward/record.url?eid=2-s2.0-84897489220&partnerID=40&md5=b75a3fe16782f14f93b72b022d2e6518>.

[17] A. Benso, S. Di Carlo, G. Di Natale, P. Prinetto, A watchdog processor to detect data and control flow errors, in: 9th IEEE On-Line Testing Symposium, 2003, IOLTS 2003, IEEE, 2003, pp. 144–148.

[18] S.D. Carlo, G.D. Natale, R. Mariani, On-line instruction-checking in pipelined microprocessors, in: 17th Asian Test Symposium, 2008, ATS '08, 2008, pp. 377–382, http://dx.doi.org/10.1109/ATS.2008.47.

[19] S. Di Carlo, G. Gambardella, P. Prinetto, D. Rolfo, P. Trotta, A. Vallero, A novel methodology to increase fault tolerance in autonomous fpga-based systems, in: 2014 IEEE 20th International On-Line Testing Symposium (IOLTS), 2014, pp. 87–92, http://dx.doi.org/10.1109/IOLTS.2014.6873677.

[20] Y. Huang, C. Kintala, Software implemented fault tolerance technologies and experience, in: Proceedings of the 23rd International Symposium on Fault-Tolerant Computing, 1993, pp. 2–9, (cited By 33). <http://www.scopus.com/inward/record.url?eid=2-s2.0-0027796299&partnerID=40&md5=e0dbd36e41c3d62d1d09e9a44fb9b24b>.

[21] A. Benso, S. Di Carlo, G. Di Natale, P. Prinetto, L. Tagliaferri, Control-flow checking via regular expressions, in: Proceedings 10th Asian Test Symposium, 2001, IEEE, 2001, pp. 299–303.

[22] A. Benso, S. Di Carlo, G. Di Natale, P. Prinetto, L. Taghaferri, Data criticality estimation in software applications, in: Test Conference, 2003, Proceedings, ITC 2003, International, vol. 1, 2003, pp. 802–810, http://dx.doi.org/10.1109/TEST.2003.1270912.

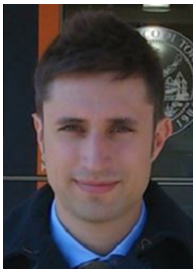
[23] A. Benso, S. Di Carlo, G. Di Natale, P. Prinetto, L. Tagliaferri, C. Tibaldi, Promon: a profile monitor of software applications, in: 8th IEEE International Workshop on Design and Diagnostics of Electronic Circuits and Systems 2005, DDECS 2005, IEEE, 2005, pp. 81–86.

[24] A. Piotrowski, D. Makowski, G. Jabłoński, S. Tarnowski, A. Napieralski, Hardware fault tolerance implemented in software at the compiler level with special emphasis on array-variable protection, in: Proceedings of The 15th International Conference Mixed Design of Integrated Circuits and Systems, MIXDES 2008, 2008, pp. 115–119, (cited By 0). <http://www.scopus.com/inward/record.url?eid=2-s2.0-56349110001&partnerID=40&md5=97a98a388e6768530e72097c363c7f27>.

[25] A. Piotrowski, D. Makowski, G. Jabónski, A. Napieralski, The automatic implementation of software implemented hardware fault tolerance algorithms as a radiation-induced soft errors mitigation technique, in: IEEE Nuclear Science Symposium Conference Record, 2008, pp. 841–846, (cited By 2). <http://www.scopus.com/inward/record.url?eid=2-s2.0-67649211970&partnerID=40&md5=b15279ca10edb0b1bbda3e918cc5c40e>.

[26] O. Goloubeva, M. Rebaudengo, M. Reorda, M. Violante, Software-Implemented Hardware Fault Tolerance, Springer, 2006 (cited By 40) <http://www.

scopus.com/inward/record.url?eid=2-s2.0-84889800114&partnerID=40&md5=a82b5c68512e252785f5b8540f221c6a>.
[27] A. DeHon, N. Carter, H. Quinn, Final Report for ccc Cross-Layer Reliability Visioning Study, March 2011. <http://www.relxlayer.org/FinalReport?action=AttachFile&do=get&target=finalreport.pdf>.
[28] CLERECO Consortium, Cross-Layer Early Reliability Evaluation for the Computing Continuum Official Website, 2013. <http://www.clereco.eu>.
[29] S. Di Carlo, A. Vallero, D. Gizopoulos, G. Di Natale, A. Grasset, R. Mariani, F. Reichenbach, Cross-layer early reliability evaluation for the computing continuum, in: 2014 17th Euromicro Conference on Digital System Design (DSD), 2014, pp. 199–205. <10.1109/DSD.2014.65>.
[30] 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, T. Loekstad, Cross-layer early reliability evaluation: challenges and promises, in: On-Line Testing Symposium (IOLTS), 2014 IEEE 20th International, 2014, pp. 228–233, http://dx.doi.org/10.1109/IOLTS.2014.6873704.
[31] D. Buchholz, J. Dunlop, The Future of Enterprise Computing: Prepare for Compute Continuum, May 2011. <http://goo.gl/KYb0H8>.
[32] Computing Community Consortium, Ccc Visioning Study on Cross-Layer Reliability, 2015. <http://xlayer.org/Home>.
[33] A. Bramnik, A. Sherban, N. Seifert, Timing vulnerability factors of sequential elements in modern microprocessors, in: 2013 IEEE 19th International On-Line Testing Symposium (IOLTS), 2013, pp. 55–60, http://dx.doi.org/10.1109/IOLTS.2013.6604051.
[34] N. Seifert, N. Tam, Timing vulnerability factors of sequentials, IEEE Trans. Dev. Mater. Reliab. 4 (3) (2004) 516–522, http://dx.doi.org/10.1109/TDMR.2004.831993.
[35] M. Ghahroodi, M. Zwolinski, R. Wong, S.-J. Wen, Timing vulnerability factors of ultra deep-sub-micron cmos, in: 2011 16th IEEE European Test Symposium (ETS), 2011, pp. 202–202, http://dx.doi.org/10.1109/ETS.2011.40.
[36] S.S. Mukherjee, C. Weaver, J. Emer, S.K. Reinhardt, T. Austin, A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor, in: Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture, IEEE Computer Society, 2003, p. 29.
[37] D. Ernst, S. Das, S. Lee, D. Blaauw, T. Austin, T. Mudge, N.S. Kim, K. Flautner, Razor: circuit-level correction of timing errors for low-power operation, IEEE Micro 24 (6) (2004) 10–20.
[38] V. Sridharan, D.R. Kaeli, Using hardware vulnerability factors to enhance avf analysis, SIGARCH Comput. Archit. News 38 (3) (2010) 461–472, http://dx.doi.org/10.1145/1816038.1816023.
[39] A. Savino, S. Carlo, G. Politano, A. Benso, A. Bosio, G. Di Natale, Statistical reliability estimation of microprocessor-based systems, IEEE Trans. Comput. 61 (11) (2012) 1521–1534, http://dx.doi.org/10.1109/TC.2011.188.
[40] S.K.S. Hari, S.V. Adve, H. Naeimi, P. Ramachandran, Relyzer: exploiting application-level fault equivalence to analyze application resiliency to transient faults, SIGPLAN Not. 47 (4) (2012) 123–134, http://dx.doi.org/10.1145/2248487.2150990.
[41] L. Rashid, K. Pattabiraman, S. Gopalakrishnan, Towards understanding the effects of intermittent hardware faults on programs, in: 2010 International Conference on Dependable Systems and Networks Workshops (DSN-W), 2010, pp. 101–106, http://dx.doi.org/10.1109/DSNW.2010.5542613.
[42] R. Vadlamani, J. Zhao, W. Burleson, R. Tessier, Multicore soft error rate stabilization using adaptive dual modular redundancy, in: Design, Automation & Test in Europe Conference & Exhibition (DATE), 2010, IEEE, 2010, pp. 27–32.
[43] M.-L. Li, P. Ramachandran, U. Karpuzcu, S.K.S. Hari, S. Adve, Accurate microarchitecture-level fault modeling for studying hardware faults, in: IEEE 15th International Symposium on High Performance Computer Architecture, 2009, HPCA 2009, 2009, pp. 105–116, http://dx.doi.org/10.1109/HPCA.2009.4798242.
[44] V. Sridharan, D. Kaeli, Eliminating microarchitectural dependency from architectural vulnerability, in: IEEE 15th International Symposium on High Performance Computer Architecture, 2009, HPCA 2009, 2009, pp. 117–128, http://dx.doi.org/10.1109/HPCA.2009.4798243.
[45] M.-L. Li, P. Ramachandran, S.K. Sahoo, S.V. Adve, V.S. Adve, Y. Zhou, Understanding the propagation of hard errors to software and implications for resilient system design, SIGOPS Oper. Syst. Rev. 42 (2) (2008) 265–276, http://dx.doi.org/10.1145/1353535.1346315.
[46] A. Benso, S. Di Carlo, G. Di Natale, P. Prinetto, Static analysis of seu effects on software applications, in: Test Conference, 2002, Proceedings, International, 2002, pp. 500–508, http://dx.doi.org/10.1109/TEST.2002.1041800.
[47] P. Mishra, A. Muttreja, N.K. Jha, Finfet circuit design, in: Nanoelectronic Circuit Design, Springer, 2011, pp. 23–54.
[48] Y. Cao, Predictive Technology Model for Robust Nanoelectronic Design, Springer, 2011.
[49] R.M. Sinnamon, J.D. Andrews, Fault tree analysis and binary decision diagrams, in: Reliability and Maintainability Symposium, 1996 Proceedings, International Symposium on Product Quality and Integrity, Annual, IEEE, 1996, pp. 215–222.
[50] S. Distefano, A. Puliafito, Dynamic reliability block diagrams vs dynamic fault trees, in: Reliability and Maintainability Symposium, 2007, RAMS'07, Annual, IEEE, 2007, pp. 71–76.
[51] C. Ciufudean, B. Satco, C. Filote, Reliability markov chains for security data transmitter analysis, in: The Second International Conference on Availability, Reliability and Security, 2007, ARES 2007, IEEE, 2007, pp. 886–894.
[52] S. Zhai, S.Z. Lin, Bayesian networks application in multi-state system reliability analysis, Appl. Mech. Mater. 347 (2013) 2590–2595.
[53] D. Heckerman, D. Geiger, D.M. Chickering, Learning Bayesian networks: the combination of knowledge and statistical data, Mach. Learn. 20 (3) (1995) 197–243.
[54] N. Foutris, M. Kaliorakis, S. Tselonis, D. Gizopoulos, Versatile architecture-level fault injection framework for reliability evaluation: a first report, in: On-Line Testing Symposium (IOLTS), 2014 IEEE 20th International, IEEE, 2014, pp. 140–145.
[55] A. Patel, F. Afram, S. Chen, K. Ghose, Marss: a full system simulator for multicore x86 cpus, in: Proceedings of the 48th Design Automation Conference, ACM, 2011, pp. 1050–1055.
[56] M.T. Yourst, Ptlsim: a cycle accurate full system x86–64 microarchitectural simulator, in: IEEE International Symposium on Performance Analysis of Systems & Software, 2007, ISPASS 2007, IEEE, 2007, pp. 23–34.
[57] T. Lindholm, F. Yellin, G. Bracha, A. Buckley, The Java Virtual Machine Specification, February 2013. <http://docs.oracle.com/javase/specs/jvms/se7/html/>.
[58] Microsoft Corporation, December 2014. <http://msdn.microsoft.com/enus/library/vstudio/w0x726c2>.
[59] Xamarin, Mono Project, December 2014. <http://www.mono-project.com>.
[60] C. Lattner, V. Adve, Llvm: a compilation framework for lifelong program analysis & transformation, in: International Symposium on Code Generation and Optimization, 2004, CGO 2004, IEEE, 2004, pp. 75–86.
[61] M. Bushnell, V.D. Agrawal, Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits, vol. 17, Springer, 2000.
[62] S. Mukherjee, Architecture Design for Soft Errors, Morgan Kaufmann, 2011.
[63] N.J. Wang, S.J. Patel, Restore: Symptom-based soft error detection in microprocessors, IEEE Trans. Dependable Secure Comput. 3 (3) (2006) 188–201.
[64] H. Cha, E.M. Rudnick, J.H. Patel, R.K. Iyer, G.S. Choi, A gate-level simulation environment for alpha-particle-induced transient faults, IEEE Trans. Comput. 45 (11) (1996) 1248–1256.
[65] S. Mirkhani, M. Lavasani, Z. Navabi, Hierarchical fault simulation using behavioral and gate level hardware models, in: Proceedings of the 11th Asian Test Symposium, 2002, (ATS'02), IEEE, 2002, pp. 374–379.
[66] A. Thomas, K. Pattabiraman, LLFI: an intermediate code level fault injector for soft computing applications, in: Workshop on Silicon Errors in Logic – System Effects (SELSE), 2013.
[67] J. Wei, A. Thomas, G. Li, 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 (DSN), 2014, pp. 375–382, http://dx.doi.org/10.1109/DSN.2014.2.
[68] V. Sharma, A. Haran, Z. Rakamaric, G. Gopalakrishnan, Towards formal approaches to system resilience, in: 2013 IEEE 19th Pacific Rim International Symposium on Dependable Computing (PRDC), 2013, pp. 41–50, http://dx.doi.org/10.1109/PRDC.2013.14.
[69] C. Lattner, The LLVM Compiler Infrastructure. <www.llvm.org>.



Alessandro Vallero received the MS degree in electronic engineering from University of Illinois at Chicago, US, and Politecnico di Torino, Italy. He has been a PhD student in the Department of Control and Computer Engineering since 2014.

Sotiris Tselonis graduated from Department of Electrical and Computer Engineering of University of Patras and received his MSc degree in Microelectronics from the Department of Informatics & Telecommunications of University of Athens, attending the division of Design of Integrated Circuits. Currently, he is a PhD student in the same department and his research area is related to reliability of computer architectures and hardware testing. He is participating in CLERECO project (FP7) and in DIaSTEMA project.

Nikos Foutrisis a PhD student at the Department of Informatics & Telecommunications, of University of Athens. His research interests are in the area of dependable computer architecture and silicon debug where he has published 11 papers on top conferences in computer architecture and hardware testing. As a part of his industrial experience he did his internship on Intel Labs. Finally, Foutris have participated in other EU-funded projects such as TRAMS and CLERECO (FP7).

Manolis Kaliorakisreceived his degree in Electrical and Computer Engineering from Democritus University of Thrace (DUTH) and his master degree in Microelectronics focused on the design of integrated circuits from the Department of Informatics & Telecommunications of University of Athens. Currently, he is a PhD student in the same department and his research interests concern reliability in computer architecture and hardware testing. He is participating in CLERECO project (FP7).

Maha Kooli received her engineer diploma in computer science and applied mathematics in 2013 from the engineering school ENSEEIH-INP Toulouse, Franc. Currently, she is in the second year of PhD. Her research interest is related to reliability, software systems and security.

Alessandro Savino received the MS degree in computer engineering and the PhD degree in information technologies from the Politecnico di Torino, Italy, where he has been a postdoc in the Department of Control and Computer Engineering since 2009. His main research topics are microprocessor test and software-based self-test as well as bioinformatics and image processing. He is member of the IEEE.

Gianfranco Politano received the MS degree in computer engineering and the PhD degree in information technologies from the Politecnico di Torino, Italy, where he has been a postdoc in the Department of Control and Computer Engineering since 2011. His main research topics are system reliability and machine learning techniques. He is a member of the IEEE and the IEEE Computer Society.

Alberto Bosio received the PhD in Computer Engineering from Politecnico di Torino in Italy in 2006. Currently he is an associate professor in the Laboratory of Informatics, Robotics and Microelectronics of Montpellier (LIRMM)-University of Montpellier 2 in France. He has published articles in publications spanning diverse disciplines, including memory testing, fault tolerance, diagnosis and functional verification. He is an IEEE member.

Giorgio Di Natale received the PhD in Computer Engineering from the Politecnico di Torino in 2003 and the HDR (Habilitation à Diriger les Recherches) in 2014 from the University of Montpellier II (France). He is currently a researcher for the National Research Center of France at the LIRMM laboratory in Montpellier. His research interests include: hardware security and trust, reliability, fault tolerance, test. He is the Action Chair of the COST Action IC1204 (TRUDEVICE) on Trustworthy Manufacturing and Utilization of Secure Devices. He is the chair of the European group of the TTTC, Golden Core member of the Computer Society and Senior member of the IEEE.

Dimitris Gizopoulos (IEEE Fellow) is Associate Professor at the Department of Informatics and Telecommunications, University of Athens where he leads the Computer Architecture Laboratory. His research focuses on dependable computer architecture, and in particular selftesting, on-line testing, fault tolerance, and silicon validation for microprocessors, embedded processors, multicore/multithreaded processors and GPUs. Gizopoulos has published more than 130 papers in peer reviewed IEEE Transactions, journals and IEEE and ACM conference proceedings. He is author of a book and editor of a second on test technology. He is or has been Associate Editor of IEEE Transactions on Computers, IEEE Transactions on VLSI Systems, IEEE Design & Test of Computers Magazine, and Springer's Journal of Electronic Testing: Theory and Applications, as well as guest editor for several special issues in IEEE Transactions and Magazines. He has served as a member of the Steering, Organizing and Program Committees, and as General and Program Chair of several IEEE and ACM conferences. Gizopoulos is a Fellow of IEEE (class of 2013), a Golden Core Member of IEEE Computer Society, and a member of ACM.

Stefano Di Carlo received the MS degree in computer engineering and the PhD degree in information technologies from the Politecnico di Torino, Italy, where he has been an assistant professor in the Department of Control and Computer Engineering from 2008 to 2014 and now is a tenured Associate professor. His research interests include DFT, BIST, and dependability as well as bioinformatics and system biology. He is a golden core member of the IEEE Computer Society and a senior member of the IEEE.