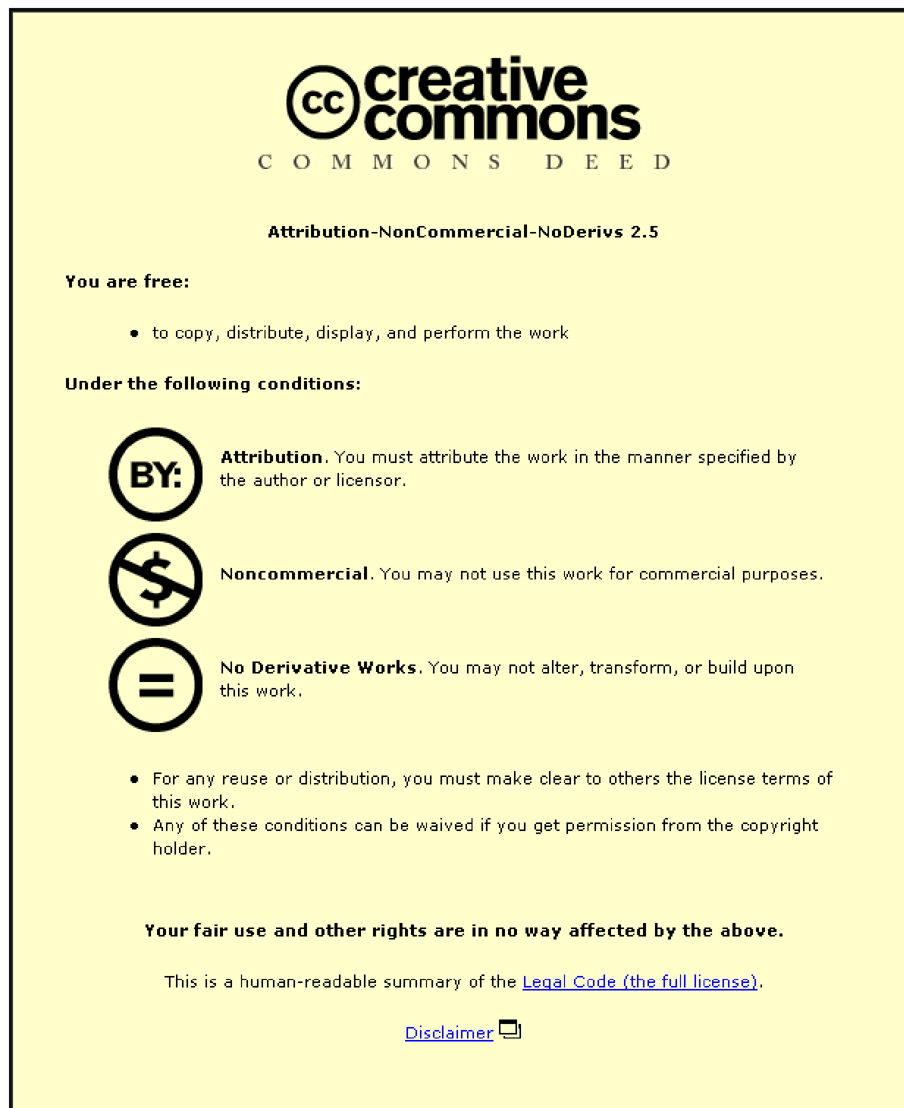
Loughborough University

This item was submitted to Loughborough's Institutional Repository (https://dspace.lboro.ac.uk/) by the author and is made available under the following Creative Commons Licence conditions.



For the full text of this licence, please go to: http://creativecommons.org/licenses/by-nc-nd/2.5/

Study of the Effects of SEU-Induced Faults on a Pipeline-Protected Microprocessor

Emmanuel Touloupis, *Member*, *IEEE*, James A. Flint, *Member*, *IEEE*,
Vassilios A. Chouliaras, *Member*, *IEEE*, and David D. Ward, *Member*, *IEEE*

Abstract—This paper presents a detailed analysis of the behavior of a novel fault-tolerant 32-bit embedded CPU as compared to a default (non-fault-tolerant) implementation of the same processor during a fault injection campaign of single and double faults. The fault-tolerant processor tested is characterized by per-cycle voting of microarchitectural and the flop-based architectural states, redundancy at the pipeline level, and a distributed voting scheme. Its fault-tolerant behavior is characterized for three different workloads from the automotive application domain. The study proposes statistical methods for both the single and dual fault injection campaigns and demonstrates the fault-tolerant capability of both processors in terms of fault latencies, the probability of fault manifestation, and the behavior of latent faults.

Index Terms—Fault injection, fault modeling and simulation, SEU, soft error, microprocessor test, fault tolerance.

1 Introduction

Major contributors of faults within electronic integrated circuit (IC) components include ionizing radiation, electromagnetic interference (EMI), and electromigration.

Radiation-induced faults were observed in the 1950s during nuclear tests; however, the first specific report of errors caused by alpha particles were published as late as 1979 [1]. The primary source of alpha particles within an IC are the package materials (mold compound, underfill, solder, and so forth); however, a number of elements that occur in the fabrication process, such as uranium and thorium, also contribute. A source of particles (for example, protons, neutrons, pions, and muons) not in the control of the semiconductor manufacturer is background radiation, which includes extraterrestrial cosmic rays [2], [3]. When charged particles travel through a semiconductor, they progressively lose energy while ionizing the medium. Electron-hole pairs are generated as a result, which, in turn, move due to the electric field inside an individual transistor. The resulting current may alter data in combinational and sequential circuits, thus generating the so-called Single-Event Upsets (SEUs), as they are more widely known. SEUs can occur directly as a bit flip on memories and registers or indirectly through transient pulses propagating in combinational circuits, usually termed a Single-Event Transient (SET), that may again affect memory elements. It must be noted that the rate of radiation-induced SEUs is not negligible in terrestrial applications and has been examined in detail by Ziegler [4].

When an event causes multiple faults, it is termed a "multiple-bit upset" (MBU) [2]. MBUs are becoming more common in newer silicon processes as the feature density is higher, so more transistors are potentially affected per unit area of influence following a particle strike.

The increasing hostility of the electromagnetic environment, contributed substantially by the ubiquitous adoption of wireless technologies (Wi-Fi, mobile telephones, and so forth), is also a significant threat to reliability [5]. Externally generated EMI is often coupled via the tracks on the printed circuit board to the IC, although it is possible for it to directly influence the silicon die at higher frequencies. Apart from external EMI, other electronic subsystems or even lines internal to the IC package can be problematic. Crosstalk within multilayer devices, in particular, is known to be a significant source of errors.

IC technology continues to follow Moore's law [6]. A major contributor to this progress is the improvement of lithography and processes where ever-decreasing feature sizes are becoming feasible. With the trend toward submicron technologies, there is an increase in the occurrence of soft errors [7], [8], [9], [10]. Decreasing the power supply voltage has reduced immunity to particle strikes because the total charge used for storing each bit is lower. In addition, the increasing clock frequency in newer devices presents problems. As the frequency increases, the errors observed will be dominated by transient faults originating in combinational logic rather than SEUs on sequential logic [11], [12]. The increasing clock frequency will tend to increase the occurrence of multiple-bit errors since the duration of the transient pulse may overlap more than one clock edge. As the complexity of on-chip circuits continues to increase, higher currents flow through the power supply lines, consequently increasing the susceptibility to electromigration [13]. The electromigration problem

- *E. Touloupis is with the Microelectronics Group, InAccess Networks SA, 12, Sorou Str, 15125, Maroussi, Athens, Greece. E-mail: etoul@inaccessnetworks.com,*
- *J.A. Flint and V.A. Chouliaras are with the Department of Electronic and Electrical Engineering, Loughborough University, Ashby Road, Loughborough, Leicestershire LE11 3TU, UK. E-mail: james.flint@ieee.org, V.A.Chouliaras@lboro.ac.uk.*
- *D.D. Ward is with the Electrical Group, MIRA Ltd., Watling Street, Nuneaton CV10 0TU, UK. E-mail: david.ward@mira.co.uk.*

Manuscript received 31 Mar. 2006; revised 12 Dec. 2006; accepted 5 June 2007; published online 28 June 2007.
Recommended for acceptance by M. Gokhale.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0124-0306.
Digital Object Identifier no. 10.1109/TC.2007.70766.

0018-9340/07/$25.00 © 2007 IEEE Published by the IEEE Computer Society
Authorized licensed use limited to: LOUGHBOROUGH UNIVERSITY. Downloaded on April 30,2010 at 13:22:25 UTC from IEEE Xplore. Restrictions apply.

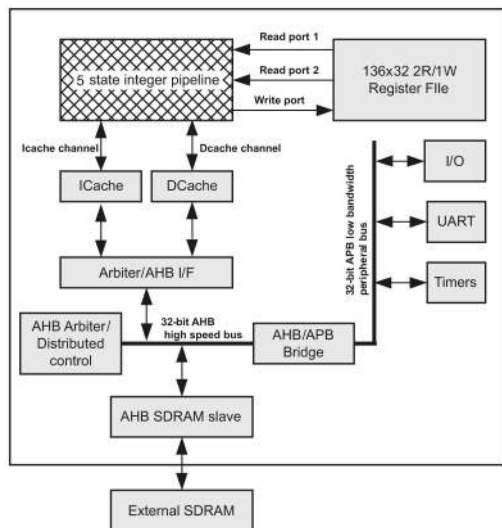


Fig. 1. Top-level architecture of the LEON2 processor.

is compounded by the reduction in feature sizes and the consequential increase in current density. Furthermore, the increased number of metal layers makes crosstalk between the interconnection lines more probable as the distance between them decreases.

As more industrial sectors adopt microprocessors in safety-critical applications such as drive-by-wire systems in the automotive industry, interest in the behavior in the presence of SEUs is increasing. The purpose of this current paper is to provide a very detailed analysis of a microprocessor which has been replicated at the pipeline level for the purpose of improving its reliability and availability. The performance of the processor is considered under the influence of both single and multiple nonconcurrent faults and in the context of three workloads, which have been selected to represent a typical application of the microprocessor in the automotive sector. It should be emphasized that the majority of techniques presented here are generic and could be applied to any type of processor as a means of revealing subtle responses to the effects of SEU and multiple faults, including concurrent and nonconcurrent faults.

In Section 2, the processor core is discussed, along with its single-CPU equivalent, which is used for comparative purposes. In Section 3, the fault injection technique used in this study is presented and, in Section 4, the results of the fault injection campaign are discussed in detail. Finally, in Section 5, we conclude with the benefits of applying this approach for the microprocessor in question.

2 Fault-Tolerant Processor

In this work, we have used LEON2 [14], an open source configurable processor, which is described by using the VHDL hardware description language. It is a five-stage pipeline processor that implements the Sparc V8 instruction set and is equipped with various peripherals that interconnect through two types of the AMBA bus (AHB and APB). A simplified top-level architecture of the processor is shown in Fig. 1.

Apart from the normal LEON2 architecture, a fault-tolerant pipeline microarchitecture proposed previously is tested [15]. Compared to the approach of Gaisler [16], which uses the TMR configuration for all of the flip-flops of the pipeline unit in the fault-tolerant version of LEON2, this new configuration triplicates the whole pipeline in order to offer additional protection against transient or permanent faults on the combinational parts of its circuit. Any single-bit fault that propagates to any of the pipeline units' outputs is detected and the "faulty" pipeline is temporarily disabled until correct register data are copied from one of the remaining "healthy" pipelines and then is reintroduced in the system. During this action, the system operates as a standard self-checking pair and the program execution is not interrupted. This redundant architecture also has the ability to mask many types of multiple faults and can remain fail silent when the faults cannot be corrected by entering into the error mode. The key concept of this architecture is the ability to not only detect but also to eliminate faults whenever it is possible. The features of this fault-tolerant microarchitecture make it suitable for applications that have very high reliability and availability requirements. The fact that the whole pipeline is triplicated allows the presence of one or even several permanent faults, depending on their location. In this case, the fault is masked whenever it propagates to the pipeline outputs.

This pipeline configuration introduces an area overhead of around 26.6 percent and also a performance penalty of 23.7 percent in the maximum clock frequency as compared to a non-fault-tolerant processor configuration. It must be noted that the above results are only indicative of a specific ASIC implementation.

All of the fault injection experiments have targeted only the pipelined execution unit (shown shaded). The sensitivity of this part of the processor is of great interest since it contains a significant amount of memory elements that constitute the registers of each pipeline stage and are therefore susceptible to SEUs. Soft faults in these parts of the processor are becoming a significant proportion of the overall error rate. In fact, the error rate of flip-flop and latches is equal to the error rate of SRAMs in 90 nm technologies, with a tendency to increase rapidly in future technologies [10]. Considering the fact that the pipeline unit is the "heart" of the processor and is responsible for its correct, continuous, and reliable operation, the study of the effects of single-bit and multiple-bit faults on the overall failure rate is critical. The impact of soft faults on larger memory arrays, such as caches and register files, has been explored in many cases in previous work (for example, [17], [18]). Furthermore, the level of the problem in this case is well known and there exist several methods for protecting them. For that reason, we have chosen to focus on the effects of faults on the pipeline and present a comparison between a standard and a fault-tolerant pipeline configuration. Such results can give an indication of the extent of the problem and the need to develop methods of protecting the control parts of the processor. These conclusions are of great interest to the safety-critical systems industry which assigns the level of required fault tolerance according to the safety integrity level of the system under study.

Authorized licensed use limited to: LOUGHBOROUGH UNIVERSITY. Downloaded on April 30,2010 at 13:22:25 UTC from IEEE Xplore. Restrictions apply.

3 Fault Injection Setup and Experiments

This section describes the fault injection environment that has been used and explains the various parameters of the fault injection experiments.

3.1 Background

A number of different software and hardware techniques have been previously reported in the literature. For example, [19] presents a detailed analysis of transient fault injection experiments on a fault-tolerant dual-processor configuration, [17] addresses the impact of SEUs on the data cache memory, and [18] presents a method for analyzing the susceptibility of different parts (instruction unit, data cache, instruction cache, and register file) of a pipelined processor. In [20], fault injection experiments are performed on a deeply pipelined out-of-order microprocessor in order to detect its most vulnerable portions. Experiments were repeated on an improved configuration, where these portions are protected by using low-overhead fault-tolerant techniques. The effects of transient and permanent faults on the program control flow on a RISC processor is examined in [21].

In many cases, fault injection experiments have been used for demonstrating the efficiency in error rate prediction of a fault injection platform or method. Such an example is the testing of an architecture based on an 80C51 microcontroller [22] by using the Code-Emulated Upset (CEU) technique [23]. In [24], the effectiveness of a fault list reduction technique is demonstrated, with results of fault injection on combinational parts of microprocessors.

In most of the aforementioned work, the analysis of the effect of faults on the targeted system is not the main objective. For this reason, the results presented do not cover in full detail many interesting subjects, such as fault latencies or fault propagation. Furthermore, there is a gap in the knowledge with regard to the effects of multiple faults on microprocessor systems, which this current paper seeks to address.

3.2 Fault Injection Environment

A popular method of injecting faults is the use of software methods (for example, [23], [25], [26], [27], [28], [29]). The main advantage of these techniques, often referred to as Software-Implemented Fault Injection (SWIFI), is that they are easy to implement and adapt to a target system. They are also cost effective since they do not require extra hardware. Furthermore, they are usually fast since they do not introduce significant delay to the execution of the target applications. Although they are built around a specific system, their principles can also be transferred relatively easily to other systems. A limitation in the context of this current paper is the fact that SWIFI cannot inject faults into non-programmer-visible locations. There are a significant number of flip-flops, registers, signals, and microarchitectural states that cannot be accessed through the instruction set but are equally sensitive to SEUs. Since the faults are injected through the software, which executes on the target system, the results obtained are unrealistic since the workload usually affects the fault-handling mechanisms (the extra fault injecting routines change the system's microarchitectural state). Furthermore, the time resolution in SWIFI is coarse. For example, it is impossible to inject a fault during the execution of an instruction, only between instructions.

Another popular method of injecting faults is the injection of physical faults on the actual target system hardware. This can be achieved through pin-level fault injection (for example, [30], [31]), heavy-ion radiation (for example, [32], [33]), EMI [34], and laser fault injection [35]. The major advantage of these approaches is that the environment is realistic (although much harsher than the real world) and the results obtained can give accurate information on the behavior of the system under such conditions. However, they require special hardware and instruments, which are usually very expensive. Furthermore, these experiments are complex to set up and control and the internal signals can only be monitored in real time if they are connected off the chip. There are also techniques that emulate faults on the actual hardware, with the use of built-in logic of the chip (for example, scan chains) [36]. In other cases, faults are injected in an FPGA prototype of the system under test, with the use of additional logic [37], [38]. In these cases, the fault injection experiments can be significantly accelerated. However, the implementation of such a scheme can be time consuming and not portable since it will heavily depend on the system under test.

A simulation-based method has been chosen in this current work. Although this approach is time consuming since simulation time is considerably longer than real-time execution, it is favored because it allows the testing of fault-tolerant systems very early in the design stage. If a VHDL or Verilog description of the system is available, testing through simulation can be performed in great detail and is potentially very accurate since it gives realistic emulation of faults and detailed monitoring of their consequences on the system. The Hardware Description Languages (HDLs) and the existing simulation environments provide a variety of tools in order to perform the fault injection and record the results. Most of the simulation-based approaches use HDL; however, there are some cases where other languages like C or C++ are used. Many different examples can be found in the literature (for example, [39], [40], [41], [42], [43]). In summary, the choice of a simulation-based method allows fast and easy implementation of the fault injection platform but limits the number of experiments due to its high computational requirements. However, as will be demonstrated in the following sections, we have managed to perform a satisfactory number of simulations by using this method.

In our system, the main fault injection support is implemented through a nonsynthesizable VHDL entity that can have access to all registers and alter their contents at specified times based on the idea of "saboteurs" [40]. This entity is "transparent" since it does not interfere or affect, in any way, the microprocessor model during its simulated operation. As can be seen in Fig. 2, faults are always synchronous with the clock and the time of appearance is expressed in clock cycles. When the cycle in which the fault is to be injected is reached, the fault injector masks the selected register inputs and creates a set of erroneous inputs for that register. At the same time, it changes the select signal of the multiplexer so that the erroneous inputs are latched into the registers. The characteristics of each fault

Authorized licensed use limited to: LOUGHBOROUGH UNIVERSITY. Downloaded on April 30,2010 at 13:22:25 UTC from IEEE Xplore. Restrictions apply.

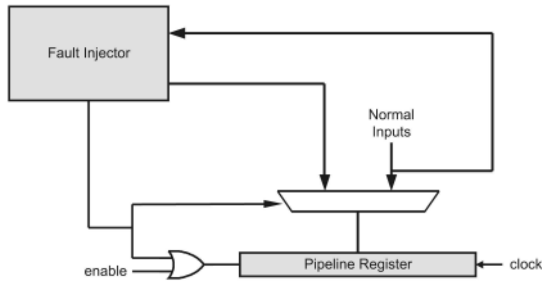


Fig. 2. Fault injection mechanism.

(time of occurrence and location) are read by the fault injector from a local file that is created before the beginning of the simulation. When the fault or faults are injected, the fault injector again reads the fault file to pick the next set of faults. Since the whole injection process is performed within one clock cycle, the fault injector is able to inject faults that occur in consecutive clock cycles. If there are no other faults to be injected, the fault injector does not interfere again in the operation of the pipeline until the end of the simulation run. It is important to note that this mechanism injects each fault or set of faults on the fly, without altering the state of the microprocessor or the program sequence and without stalling the CPU.

Our system also makes extensive use of the Foreign Language Interface (FLI) of the chosen commercial VHDL simulator (ModelSim) in order to implement various entities that deal with the fault injection, monitoring, and control of the simulation while gathering data about the state of the simulated system. The use of interfaces to routines that are written in different programming languages (in this case, C), offers great benefits when developing a fault injection tool (for example, easier file I/O handling and more efficient use of mathematical functions). At the same time, it is possible to have full visibility of all VHDL objects (for example, signals) and to control the simulator functions.

The steps of the fault injection campaign are given as follows:

1. Perform a golden run in order to obtain information such as possible fault locations, microarchitectural parameters related to the executed program (contents of registers and caches at the end of the execution), program execution time, and correct program outputs.
2. Generate a list of faults that define the time (clock cycle) and location (register bit) of the injection. The list is stored as a simple text file, which is read at the start of the simulation by an FLI-implemented entity.
3. Define the number of faults to be injected in each simulation and the total number of simulations for the campaign. Any number of faults can be injected per (simulation) time unit.
4. Run the simulations.
5. Analyze the output data. The output files contain raw numerical data which are subsequently parsed by simple PERL scripts.

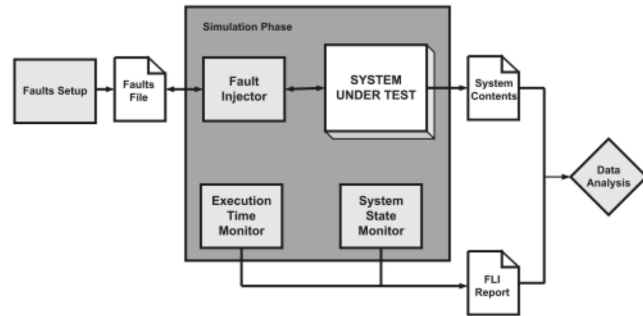


Fig. 3. Fault injection environment.

The fault injection environment is depicted in Fig. 3. It must be noted that this same process can be applied to any system that can run a simulation of a microprocessor (such as LEON2).

3.3 Fault Model

The most common fault model in the literature is the single-bit flip of a state element, which is also used in this work. Multiple faults have only been considered when testing fault-tolerant memories, but, in these cases, these faults are always assumed to have occurred concurrently. Nonconcurrent fault pairs have been used in [44] purely for a theoretical analysis of the effectiveness of design diversity in redundant systems. However, the time of occurrence is not modeled in detail. With increasing clock frequencies, radiation and EMI-induced errors may occur in several consecutive clock cycles. Furthermore, these errors may occur in different locations, depending on the different fault-propagation paths. In this work, we model nonconcurrent multiple faults by using the normal distribution [45]. We are interested in modeling a set of faults occurring around a specific time within a specific time interval (program execution time). The normal distribution provides this specific time through the mean μ, whereas the standard deviation σ determines the level of distribution around μ. In that sense, the normal distribution is an ideal choice compared to other popular distributions like the Poisson distribution, which is more suitable when the notion of fault rate is used.

3.4 Workload

A number of different benchmark applications have been used in these fault injection experiments:

- *mtx4x4*. This program multiplies two 4×4 integer matrices and stores the result at a specified memory location.
- *bitcnt*. This algorithm is used for testing the bit manipulation abilities of the processor by counting the number of bits in an array of integers using five different methods. This is part of the automotive and industrial control category of the MiBench embedded benchmark suite [46] and a few minor modifications have been made in order to adapt its I/O operations to the VHDL simulation environment.
- *qsort*. This program sorts a number of strings by using a well-known quicksort algorithm. It is also a part of

Authorized licensed use limited to: LOUGHBOROUGH UNIVERSITY. Downloaded on April 30,2010 at 13:22:25 UTC from IEEE Xplore. Restrictions apply.

TABLE 1
Execution Times of the Benchmark Applications

Program	Duration (clock cycles)
mtx4x4	3150
bitcnt	14740
qsort	46800

the automotive and industrial control category of the MiBench benchmark suite, but it has been modified to reduce its runtime and to adapt its I/O operations to the VHDL simulation environment.

The execution times of each application are shown in Table 1 and their dynamic instruction distribution is shown in Fig. 4. The addition of several more test programs would be desirable; however, the amount of simulation time required is somewhat restrictive. The workloads chosen were selected for their diversity within the set available. Judging from the obtained results and also previously published work, we consider that our choice of workloads, in the context of this paper's subject, raises many interesting issues that are analytically discussed in the following sections.

3.5 General Setup

As already mentioned, fault injection experiments have been performed on two microprocessor architectures. The first was the LEON2 microprocessor without any modifications (hereafter known as the single pipeline architecture) and the second was a fault-tolerant version of LEON2, with redundancy on the pipelined execution unit (hereafter known as the redundant pipeline architecture).

There were two main categories of fault injection campaigns: one in which single faults were injected in each simulation and one where double faults were injected following the fault model described in the previous section to define the time of injection for each pair of faults. For each pair, a random value for μ was selected and the time of injection for each fault was defined by using a preselected value of σ. Due to the long duration and the number of the VHDL simulations, a selection of values for σ has been used. Since the processor being investigated has a 5-stage pipeline, it is particularly interesting to observe its behavior when both faults occur concurrently or within a small time difference. By setting $\sigma = 0.833$, 99.7 percent of the faults occur within five clock cycles. The fault distributions used are summarized in Table 2.

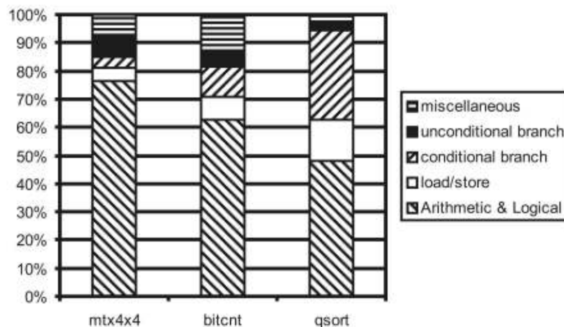


Fig. 4. Dynamic instruction distribution for the benchmarks used.

TABLE 2
Fault Occurrence Classification

Concurrent	The two faults are concurrent
$\sigma = 0.833$	99.7% of fault pairs occur within 5 clock cycles
$\sigma = 10$	99.7% of fault pairs occur within 10 clock cycles
$\sigma = 510$	99.7% of faults pairs occur within 3060 clock cycles
Uniform	The time of the two faults is chosen by using a uniform distribution

The register to be affected by each fault is selected randomly from the available registers/flops in the selected pipeline stage. The probability of one register being selected depends on the number of bits that it consists of. The registers of the pipeline have been grouped according to the stage to which they belong. As a result, there are five groups for each stage (fetch, decode, execute, memory, and write back) and one group of special purpose registers.

The output data for such a scheme consists of information about the correctness of results, the presence of latent faults inside the microarchitecture after the end of the program execution, the status of the processor at the end of the simulation, and the execution time. The simulation categories, which are similar to those used in [18], are defined as follows:

- *No effect*. The program terminates normally, the results are correct, and the contents of the pipeline registers and the register file are identical to those in the golden run.
- *Latent*. The program terminates normally, the results are correct, but the contents of the pipeline registers and/or the register file are not the same as those in the golden run.
- *Wrong result*. The program terminates normally, but the results are incorrect.
- *Timed out*. The program failed to terminate within a predefined time limit and the simulation was halted externally. Since our work focuses on hard real-time applications, this time limit has been kept short (400 clock cycles).
- *Exception*. The processor detected an erroneous condition and created a trap, forcing it into the error mode.

Exhaustive tests are infeasible in this type of complex system. The total number of possible faults depends on space (the targeted bit) and time (the time of injection), thus becoming very large, even for small benchmark programs. For each benchmark program, two types of fault injection experiments have been performed:

1. *Single fault*: one set of 6,000 simulations, with one fault injected in each. The simulations are divided equally, with 1,000 simulations for each of the six register groups fe, de, ex, and so forth.
2. *Double fault*: one set of 2,000 simulations for each register group and for every different level of the distribution chosen (see Table 2), resulting in a total number of 60,000 simulations. In every simulation, a

Authorized licensed use limited to: LOUGHBOROUGH UNIVERSITY. Downloaded on April 30,2010 at 13:22:25 UTC from IEEE Xplore. Restrictions apply.

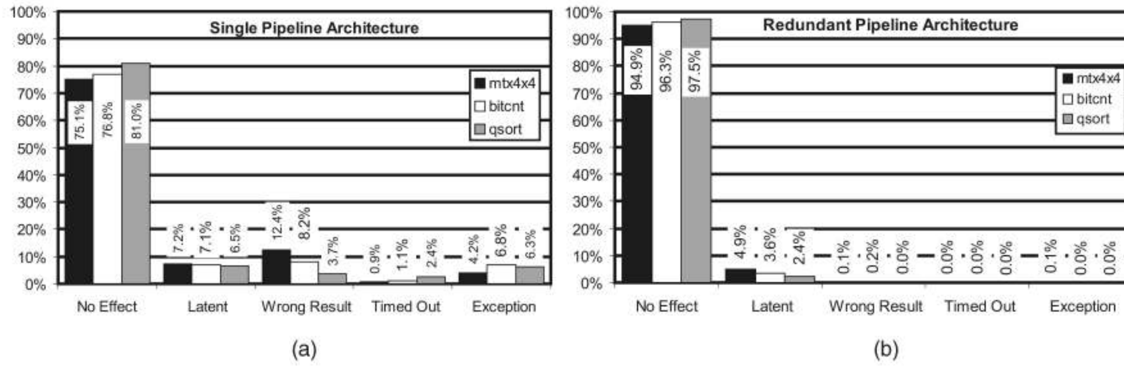


Fig. 5. Performance of the two architectures under single fault injection.

fault pair is injected into the selected register group, which means that both faults occur on the same pipeline stage.

It must be noted that several hours were required to perform such large sets of simulations on an average Windows or Linux-based PC.

Through convergence tests, we observed that the final results were almost identical to an experiment containing a much smaller number of simulations. This allows us to be confident that the chosen number of simulations was sufficient to obtain an accurate result.

The data obtained from each campaign gives an indication of the behavior of each pipeline stage. In order to get the overall view of the pipeline behavior, the total probability theorem can be used. This means that each simulation result category is calculated by combining the figures obtained for each stage and the probability of each stage being hit by a fault. The latter depends on the number of bits that each stage (register group) contains. The probability for each category is thus calculated as

$$P_C = \frac{N_{fe}}{N_{total}} P_{fe} + \frac{N_{de}}{N_{total}} P_{de} + \frac{N_{ex}}{N_{total}} P_{ex} + \frac{N_{me}}{N_{total}} P_{me} + \frac{N_{wr}}{N_{total}} P_{wr} + \frac{N_{sregs}}{N_{total}} P_{sregs},$$

where P_C is the probability that a simulation will end in category C, N_{fe}, N_{de}, N_{ex}, N_{me}, N_{wr}, and N_{sregs} are the numbers of bits of each register group, N_{total} is the total number of bits in the pipeline unit, and P_{fe}, P_{de}, P_{ex}, P_{me}, P_{wr}, and P_{sregs} are the probabilities for each register group for category C. These figures are measured directly from the simulation results.

In the presentation of our results, all of the above probabilities are expressed in percentages.

4 Fault Injection Results and Analysis

This section presents and discusses the results obtained from the fault injection experiments.

4.1 Single Faults

The results from the fault injection experiments performed on the single and the redundant pipeline architecture are shown in Figs. 5a and 5b, respectively.

4.1.1 General Behavior

The dependence of the performance on the processor's workload in the single architecture is very clear. It can be seen that *mtx4x4* has the worst performance in producing wrong results. This is due to the fact that this application benchmark is computationally intensive, with the extensive use of the arithmetic logic unit (ALU) and very few references to the memory. This means that the vast majority of the ALU operations in this benchmark are directly associated with the actual data due to the lack of loops within the code, having, as a result, higher rates in producing wrong results. This explains the improved rate observed in *bitcnt*, which is also ALU intensive. The *qsort* benchmark is based on many comparisons in order to sort an array of strings. The proportion of ALU-related instructions is smaller and the proportion of load/store instructions is higher, resulting in a generally better performance. As can be observed in these graphs, there is a substantial improvement in fault tolerance in the case of the redundant pipeline architecture. Overall, a very important characteristic of the redundant architecture, apart from its obvious fault-tolerant properties, is the minimal dependency on the application that is executed.

The generation of exceptions (for example, illegal memory reference and illegal operator code (opcode)) is also dependent on the workload in the single architecture. However, the percentage of generated exceptions is quite low in an architecture without any additional fault detection mechanism, such as the one that has been used for these simulations. The mechanisms that generate exceptions are the only means of soft fault detection in this case and the obtained results show that they are inadequate when high data integrity is a requirement. Exceptions are mainly generated when faults occur on the fetch stage or on the special registers. In the case of the redundant architecture, practically, no exceptions are observed since all of the faults are masked.

The probability of a simulation being "Timed Out" increases when the executed program uses recursive algorithms and many loops. A fault may increase the number of loops performed, thus delaying the termination of the program. This is consistent with the obtained results, where it is seen that *qsort*, which uses a recursive algorithm, has an increased percentage of "Timed Out" simulations, whereas *mtx4x4* has the lowest. Again, the redundant

Authorized licensed use limited to: LOUGHBOROUGH UNIVERSITY. Downloaded on April 30,2010 at 13:22:25 UTC from IEEE Xplore. Restrictions apply.

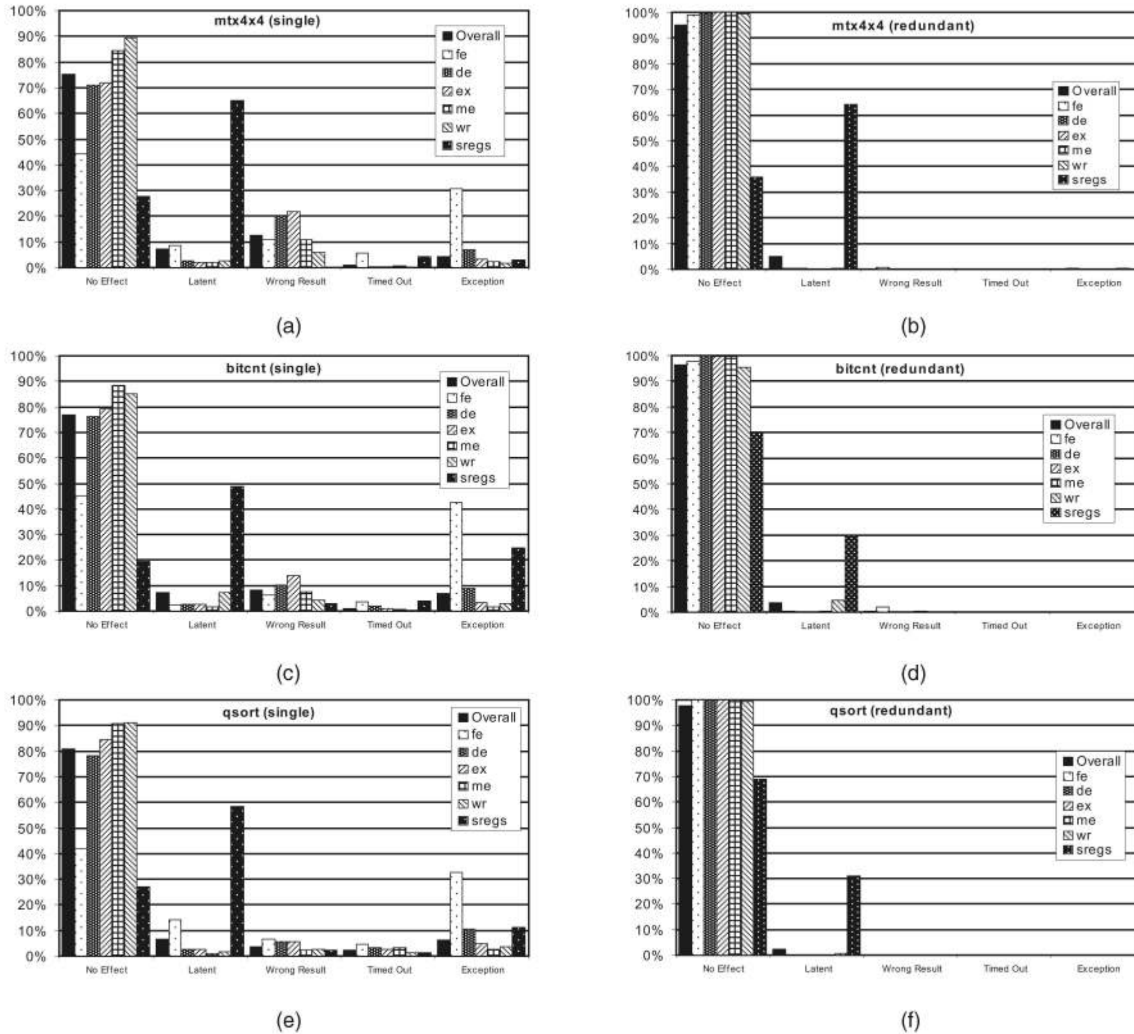


Fig. 6. Pipeline stages behavior in the single and redundant pipeline architecture.

architecture does not allow faults to alter the correct execution sequence.

The probability of a fault remaining latent within the microarchitecture can be affected by many different factors. A fault may propagate to the register file and remain there without being revealed. In other cases, a fault in the program counter or on loop controlling data may alter the execution sequence, thus changing the microarchitectural state at the end, without, however, causing implications on the correctness of results. Another class of latent faults is those that occur on unused special registers. Each of these factors depends on the workload; however, the overall rate of latent faults seems to be the same for the three benchmarks used in the experiments. This is clarified when examining the contribution of each stage in the generation of latent faults for each different benchmark program. The rate of latent faults is reduced in the redundant pipeline architecture due to the fact that latent faults in this case can only remain in the pipeline registers since no single fault can propagate to any output signal without being detected and masked. Latent faults in the redundant architecture originate in faults injected in (or propagated to) special registers only. It can be argued that latent faults in this architecture cannot cause a system failure since, even if they propagate to the pipeline output at any point, they will be detected.

4.1.2 Pipeline Stages Contribution

Fig. 6 depicts the contribution of each pipeline stage to the overall pipeline behavior for the different benchmarks. The first observation, as already mentioned, concerns the high rate of latent faults in special registers. This is expected since the majority of special registers are written during the boot sequence and they are not changed or, in some cases, even accessed during the execution of the application. The second observation concerns the fetch stage, where there is a high rate of generated exceptions. This can be explained by the fact that the main register of the fetch stage is the program counter and a fault in one of its bits can create an illegal memory reference. The decode stage seems to have a similar behavior, but to a lesser extent. The special registers contribute an important percentage to the generated exceptions in the *bitcnt* benchmark. This is due to the fact that, with *bitcnt*, faults on special registers have a higher probability of propagating to the pipeline's output. The reason for that is explained in the next section, where propagation probabilities are presented. Faults in the

Authorized licensed use limited to: LOUGHBOROUGH UNIVERSITY. Downloaded on April 30,2010 at 13:22:25 UTC from IEEE Xplore. Restrictions apply.

TABLE 3
Comparison of Results for the *mtx4x4* Benchmark

	Results (%)					Results from [18] (%)				
	N.E.	La.	W.R.	T.O.	Exc.	N.E.	La.	W.R.	T.O.	Exc.
fe	44.4	8.5	11	5.7	30.6	65.48	3.93	6.56	5.89	18.14
de	70.9	2.5	11.6	0.3	7	74.07	0.38	12.42	1.85	11.28
ex	72.5	2	21.8	0.4	3.3	83.51	2.21	10.32	2.12	1.84
me	84.5	1.9	10.8	0.5	2.3	85.19	3.11	5.09	2.26	4.35
wr	89.4	2.8	5.9	0.1	1.8	84.32	7.96	2.29	1.41	4.02
sregs	27.7	65	0	4.4	2.9	N/A	N/A	N/A	N/A	N/A

execution stage are mainly responsible for the generation of wrong results. This is more apparent in the two ALU intensive benchmarks (*mtx4x4* and *bitcnt*). Finally, faults in the fetch stage appear to be the main cause of "Timed Out" simulations. As explained previously, faults on the program counter have a direct effect on the program executing sequence and can significantly delay termination.

4.1.3 Comparison of Results

It is difficult to compare these results with other reported work, mainly because of the lack of experiments on the same microprocessor. The only relevant analysis made on the LEON2 processor was carried out by Rebaudengo et al. [18]; however, this used a hardware-based fault injection platform. Making a full comparison is challenging as the detailed fault injection mechanism and the exact processor configuration are unknown. Furthermore, the registers are grouped according to the pipeline stage to which they belong, without information about the special registers.

Table 3 compares the fault injection results produced by the method discussed in this paper and those produced by the hardware technique in [18] for single fault injection experiments for the *mtx4x4* benchmark. The first column identifies the pipeline stage at which faults were injected (FE = fetch, DE = decode, EX = execute, ME = memory return, WR = write-back, and SREGS = special registers), whereas the second row from the top identifies the effects of these experiments. NE stands for No Effect, La = latent error, WR = wrong result, TO = processor time-out, and EXC = processor exception. In comparison with [18], the generation of wrong results occurs less frequently, but there is a slightly higher number of timed-out executions.

Despite the challenge in producing comparative results, the overall trends are comparable and the results have been summarized here for completeness and as a benchmark for the work in the current paper.

4.2 Propagation Probabilities and Latencies

Fig. 7 shows the probability of a fault propagating to the output for the whole pipeline unit and for each stage separately. All faults in the fetch stage immediately affect the outputs since both the program counter and the branch flag are connected to the instruction cache. It can be noted that the propagation probability is similar for every benchmark program in each stage, except for the special registers. Faults in a special register can remain dormant for a long period of time and manifest their presence to an output only when special conditions are met. For example, a fault in the trap base address (TBA) register will only be activated if a trap is caused during the execution of a program. This example is highlighted in these fault injection experiments.

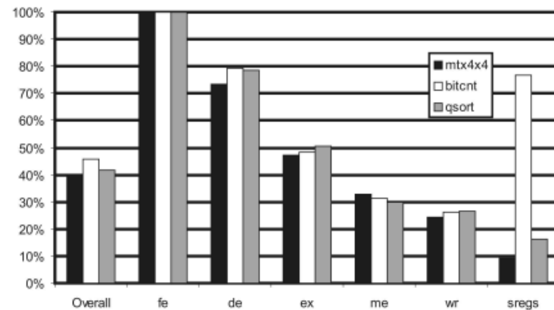


Fig. 7. Probability of a fault propagating to the pipeline output.

As can be seen, the propagation probability in the special registers is much higher in the *bitcnt* benchmark compared with the other two benchmarks. This is due to the fact that, during the execution of *bitcnt*, "window_overflow" and "window_underflow" traps [47] are generated several times. Since the TBA register is the largest among the special registers, faults of this type are frequent in the fault injection experiments, thus increasing the propagation rate in this case.

A significant number of fault injection simulations showed a delayed termination in the execution of the program in the case of the single pipeline architecture. A delay of even a few clock cycles can be very important in hard real-time applications and their study is of great importance. Some statistics related to this issue are shown in Table 4.

4.3 Double Faults

The results obtained for the double fault injection experiments are shown in Fig. 8. The following sections discuss the results for each category and present a summary of conclusions.

4.3.1 No Effect

As would be expected in both processor configurations, fewer double fault simulations finish without any effects than in the single fault case (Figs. 8a and 8b).

For the single processor case, this reduction is approximately the same for each benchmark and varies between 10 percent and 15 percent, depending on the value of σ. The effect of interfault occurrence time is not particularly large; however, in general, the performance deteriorates as the time between faults increases. The level of deterioration is approximately the same for all of the benchmarks.

The equivalent reduction for the redundant pipeline architecture varies between approximately 1 percent and 13 percent. In this case, the behavior of the system seems to be

TABLE 4
Delayed Simulations in the Single Pipeline Architecture for Single Faults

	fe	de	ex	me	wr	sregs	Total
mtx4x4	489	97	73	21	15	0	695 out of 6000
bitcnt	396	60	65	40	23	21	605 out of 6000
qsort	315	41	23	9	6	33	427 out of 6000

Authorized licensed use limited to: LOUGHBOROUGH UNIVERSITY. Downloaded on April 30,2010 at 13:22:25 UTC from IEEE Xplore. Restrictions apply.

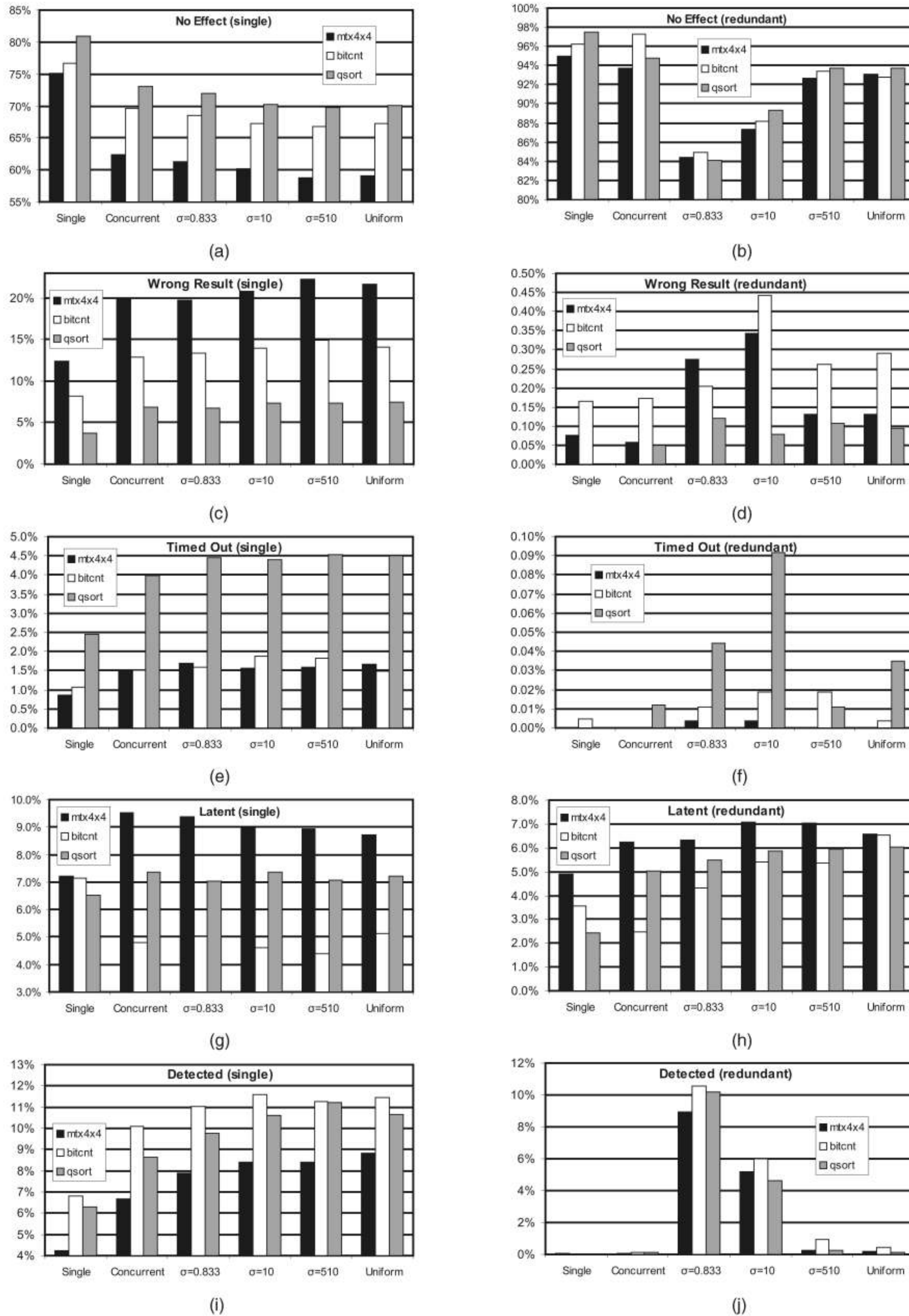


Fig. 8. Performance of the two architectures under double, nonconcurrent fault injection.

much more dependent on interfault occurrence time. For concurrent faults, the rate of effectless fault pairs is the highest since, in many cases, they manifest as single faults (for example, in most cases, two concurrent faults on the same register have the same effect as a single fault on that register). However, when the two faults occur at different times, the probability of disrupting the normal processor operation increases. As can be observed, small interfault occurrence

Authorized licensed use limited to: LOUGHBOROUGH UNIVERSITY. Downloaded on April 30,2010 at 13:22:25 UTC from IEEE Xplore. Restrictions apply.

times tend to have the most severe impact on this architecture. This is expected because of the critical period that starts after the detection of the first fault and lasts for two clock cycles. A second fault occurring during that period may disrupt the normal execution by either triggering a fault detection mechanism or causing a failure. As the interfault occurrence time increases, the effects of the fault pairs are less intense. Another important observation in this graph is that the behavior of this architecture does not depend heavily on the workload since all benchmarks have approximately the same percentages in all of the categories.

4.3.2 Wrong Result

These occur at a much higher rate in the single pipeline architecture when double faults are injected (Fig. 8c). The percentages for different values of σ do not change the results very much; however, there is a general trend of increasing in sympathy with an increase in the interfault occurrence time. Compared to the results of the single fault injection campaigns, the incorrect result generation rate is almost doubled for some benchmarks.

The results for the redundant pipeline architecture are quite different (Fig. 8d). For the two benchmarks (*mtx4x4* and *bitcnt*), the percentage increases with a peak at $\sigma = 10$, but remains reasonably low. For *qsort*, the percentage is even lower and remains unchanged for different levels of interfault timing distribution. An apparently paradoxical situation can be observed for *mtx4x4* and, to a lesser extent, for *bitcnt* in the category of concurrent faults. The result in this case is lower than that obtained for single fault injection. This can be explained by the fact that the presence of two concurrent faults raises the probability of at least one being detected before either creates any type of failure. Hence, in some cases, the two concurrent faults are equivalent to a more "easily detectable" single fault.

4.3.3 Timed Out

The percentage of timed-out simulations does not appear to depend on the interfault occurrence time in the single pipeline architecture (Fig. 8e). Both for the single and the double fault injection campaign, the *qsort* benchmark has a higher rate of timed-out simulations due to the high number of conditional branch instructions that it contains, which, when affected by a fault, may result in a delayed program termination (for example, the program may remain in a loop for more time than initially intended). This is also observed in the redundant pipeline architecture (Fig. 8f), where the timed-out percentage for $\sigma = 10$ is as high as the average "Wrong Result" percentage for this benchmark.

4.3.4 Latent Faults

The percentages of simulations that finish normally with correct results but latent faults are found in the final microarchitectural state are slightly increased in all cases compared to the results obtained from the single fault injection (Figs. 8g and 8h). There is no general trend that characterizes the behavior of the two architectures in relation to the interfault occurrence time since no significant variation is observed for different values of σ.

4.3.5 Exception

The number of simulations where the faults are detected tends to increase in the single pipeline architecture for higher values of σ (Fig. 8i). The relative increment is kept the same among the three different benchmarks. Compared to the results obtained from the single fault injection campaign, the increment in the detection of fault conditions by the mechanisms of the LEON2 processor varies between the benchmark programs. In particular, it is significantly augmented with *mtx4x4*, whereas it is almost doubled with *qsort* and *bitcnt*. It must be noted that the detection in the single pipeline architecture is performed by LEON's own fault detection mechanisms. On the other hand, fault detection is performed mainly by the proposed fault-tolerant mechanism [15] in the case of the redundant pipeline architecture. In this case, only faults that are not correctable are detected (Fig. 8j). When the fault pairs are concurrent, very few cases are uncorrectable. However, when the second fault occurs within a few cycles of the first, it is difficult for both faults to be masked. Hence, the detection percentages rise to 9 percent or 10 percent for $\sigma = 0.833$. As σ increases, fewer fault pairs become uncorrectable and the percentage drops. As shown in the relevant graph, this behavior is not dependent on the benchmark program since the percentages in all three are very similar.

5 Conclusion

Some interesting conclusions about the effects of multiple faults on this microprocessor can be extracted from the results. Concurrent multiple faults are not much worse when compared with single faults. Especially when they occur in the same location (for example, two bits on the same register), they are equivalent to a single fault. However, in the case of nonconcurrent faults, the impact depends heavily on the specific features of the microarchitecture under study. In our case, we can observe that the behavior of the single pipeline is not dependent on the fault interoccurrence time. The redundant architecture, however, is due to its microarchitecture, which is more sensitive to nonconcurrent faults with small interoccurrence time. It must be noted that, despite this trend, the redundant pipeline offers a substantial improvement in the system's overall reliability. Finally, we can observe that, for large values of σ and for the uniform distribution, the probability of failure (wrong result or timed-out execution) in both cases is doubled, as would be expected.

This paper has presented a very detailed analysis of fault effects on two pipeline processor configurations. The results have revealed very subtle features of both the LEON2 processor itself and its behavior in the presence of faults in its pipeline unit. A substantial improvement in reliability has been demonstrated for a proposed redundant architecture by exploring the effects of injection of both single and double faults. Triplication as an approach to the protection of the pipeline has often been neglected due to the impact on overhead. Nevertheless, our work has demonstrated clear benefits of following this approach, particularly in the context of handling multiple faults.

Authorized licensed use limited to: LOUGHBOROUGH UNIVERSITY. Downloaded on April 30,2010 at 13:22:25 UTC from IEEE Xplore. Restrictions apply.

A new fault model for the injection of multiple faults has been applied for the double fault injection. This model aims at representing a radiation-induced soft fault that may be nonconcurrent. The validity and usefulness of this model has been proven since different behavior has been observed when changing the time between the occurrence of the two faults. Although this was more apparent in the redundant architecture due to its nature, a dependency has also been observed in many cases in the single pipeline architecture. Although this paper has contributed results for a particular processor, it raises more general questions about exercising the failure modes in highly integrated multiprocessor system-on-chip designs. By observing the data collected in a study of this type, it is possible to observe weaknesses in the fault-handling capabilities, with a view to improving performance.

Acknowledgments

This work was supported by MIRA Ltd., UK and the Department of Electronic and Electrical Engineering of Loughborough University, United Kingdom.

References

[1] T.C. May and M.H. Woods, "Alpha-Particle-Induced Soft Errors in Dynamic Memories," *IEEE Trans. Electron Devices,* vol. 26, no. 1, 1979.
[2] J.F. Ziegler, H.W. Curtis, H.P. Muhlfeld, C.J. Montrose, B. Chin, M. Nicewicz, C.A. Russell, W.Y. Wang, L.B. Freeman, P. Hosier, L.E. LaFave, J.L. Walsh, J.M. Orro, G.J. Unger, J.M. Ross, T.J. O'Gorman, B. Messina, T.D. Sullivan, A.J. Sykes, H. Yourke, T.A. Enger, V. Tolat, T.S. Scott, A.H. Taber, R.J. Sussman, W.A. Klein, and C.W. Wahaus, "IBM Experiments in Soft Fails in Computer Electronics (1978-1994)," *IBM J. Research and Development,* vol. 40, no. 1, pp. 3-18, 1996.
[3] R.C. Baumann, "Soft Errors in Advanced Semiconductor Devices—Part I: The Three Radiation Sources," *IEEE Trans. Device and Material Reliability,* vol. 1, no. 1, pp. 17-22, 2001.
[4] J.F. Ziegler, "Terrestrial Cosmic Ray Intensities," *IBM J. Research and Development,* vol. 42, no. 1, pp. 117-139, 1998.
[5] "EMC for ICs," http://www.ic-emc.org/, 2007.
[6] G.E. Moore, "Cramming More Components onto Integrated Circuits," *Electronics,* vol. 38, no. 8, 1965.
[7] A.H. Johnston, "Scaling and Technology Issues for Soft Error Rates," *Proc. Fourth Ann. Research Conf. Reliability,* Apr. 2000.
[8] E. Dupont, M. Nicolaidis, and P. Rohr, "Embedded Robustness IPs for Transient-Error-Free ICs," *IEEE Design and Test of Computers,* vol. 40, no. 3, pp. 56-70, May/June 2002.
[9] C. Constantinescu, "Trends and Challenges in VLSI Circuit Reliability," *IEEE Micro,* vol. 23, no. 4, pp. 14-19, July-Aug. 2003.
[10] R. Baumann, "Soft Errors in Advanced Computer Systems," *IEEE Design and Test of Computers,* vol. 22, no. 3, pp. 258-266, May/June 2005.
[11] S. Buchner, M. Baze, D. Brown, D. McMorrow, and J. Melinger, "Comparison of Error Rates in Combinational and Sequential Logic," *IEEE Trans. Nuclear Science,* vol. 44, no. 6, pp. 2209-2216, 1997.
[12] F. Irom and F.F. Farmanesh, "Frequency Dependence of Single-Event Upset in Advanced Commercial PowerPC Microprocessors," *IEEE Trans. Nuclear Science,* vol. 51, no. 6, pp. 3505-3509, 2004.
[13] P.-C. Li and T.K. Young, "Electromigration: The Time Bomb in Deep-Submicron ICs," *IEEE Spectrum,* vol. 33, no. 9, pp. 75-78, Sept. 1996.
[14] "The LEON2 Processor User's Manual," http://www.gaisler.com, 2007.
[15] E. Touloupis, J.A. Flint, V.A. Chouliaras, and D.D. Ward, "A Fault-Tolerant Processor Core Architecture for Safety-Critical Automotive Applications," *SAE 2005 Trans. J. Passenger Cars: Electronic and Electrical Systems,* pp. 1-6, Feb. 2006.
[16] J. Gaisler, "A Portable and Fault-Tolerant Microprocessor Based on the SPARC v8 Architecture," *Proc. Int'l Conf. Dependable Systems and Networks (DSN '02),* pp. 409-415, 2002.
[17] F. Faure, R. Velazco, M. Violante, M. Rebaudengo, and M.S. Reorda, "Impact of Data Cache Memory on the Single-Event-Upset-Induced Error Rate of Microprocessors," *IEEE Trans. Nuclear Science,* vol. 50, no. 6, pp. 2101-2106, 2003.
[18] M. Rebaudengo, M.S. Reorda, and M. Violante, "Accurate Analysis of Single Event Upsets in a Pipelined Microprocessor," *J. Electronic Testing: Theory and Applications,* vol. 19, no. 5, pp. 577-584, 2003.
[19] D. Gil, R. Martínez, J.V. Busquets, J.C. Baraza, and P.J. Gil, "Fault Injection into VHDL Models: Experimental Validation of a Fault Tolerant Microcomputer System," *Proc. Third European Dependable Computing Conf. (EDCC '99),* pp. 191-208, 1999.
[20] N.J. Wang, J. Quek, T.M. Rafacz, and S.J. Patel, "Characterizing the Effects of Transient Faults on a High-Performance Processor Pipeline," *Proc. 2004 Int'l Conf. Dependable Systems and Networks (DSN),* 2004.
[21] M. Rimén, J. Ohlsson, and J. Karlsson, "Experimental Evaluation of Control Flow Errors," *Proc. 1995 Pacific Rim Int'l Symp. Fault Tolerant Systems (PRFTS),* 1995.
[22] R. Velazco, S. Rezgui, and H. Ziade, "Assessing the Soft Error Rate of Digital Architectures Devoted to Operate in Radiation Environment: A Case Studied," *J. Electronic Testing: Theory and Applications,* vol. 19, no. 1, pp. 83-90, 2003.
[23] R. Velazco, S. Rezgui, and R. Ecoffet, "Predicting Error Rate for Microprocessor-Based Digital Architectures through C.E.U. (Code Emulating Upsets) Injection," *IEEE Trans. Nuclear Science,* vol. 47, no. 6, pp. 2405-2411, 2000.
[24] M.S. Reorda and M. Violante, "Efficient Analysis of Single Event Transients," *J. Systems Architecture,* vol. 50, no. 5, pp. 239-246, 2004.
[25] Z. Segall, D. Vrsalovic, D. Siewiorek, D. Yaskin, J. Kownacki, J. Barton, D. Rancey, A. Robinson, and T. Lin, "FIAT: Fault Injection Based Automated Testing Environment," *Proc. 18th Int'l Symp. Fault-Tolerant Computing (FTCS-18),* pp. 102-107, 1988.
[26] W. Kao, R.K. Iyer, and D. Tang, "FINE: A Fault Injection and Monitoring Environment for Tracing the UNIX System Behavior under Faults," *IEEE Trans. Software Eng.,* vol. 19, no. 11, pp. 1105-1118, Nov. 1993.
[27] S. Han, K.G. Shin, and H.A. Rosenberg, "DOCTOR: An Integrated Software Fault Injection Environment for Distributed Real-Time Systems," *Proc. Int'l Computer Performance and Dependability Symp. (IPDS '95),* pp. 204-213, 1995.
[28] G.A. Kanawati, N.A. Kanawati, and J.A. Abraham, "FERRARI: A Flexible Software-Based Fault and Error Injection System," *IEEE Trans. Computers,* vol. 44, no. 2, pp. 248-260, Feb. 1995.
[29] J. Carreira, H. Madeira, and J. Gabriel Silva, "Xception: A Technique for the Experimental Evaluation of Dependability in Modern Computers," *Software Eng.,* vol. 24, no. 2, pp. 125-136, 1998.
[30] J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J.-C. Fabre, J.-C. Laprie, E. Martins, and D. Powell, "Fault Injection for Dependability Validation: A Methodology and Some Applications," *IEEE Trans. Software Eng.,* vol. 16, no. 2, pp. 166-182, 1990.
[31] H. Madeira, M.Z. Rela, F. Moreira, and J.G. Silva, "RIFLE: A General Purpose Pin-Level Fault Injector," *Proc. First European Dependable Computing Conf. (EDCC '94),* pp. 199-216, 1994.
[32] J.K.U. Gunneflo and J. Torin, "Evaluation of Error Detection Schemes Using Fault Injection by Heavy-Ion Radiation," *Proc. 19th Int'l Symp. Fault Tolerant Computing,* pp. 340-347, 1989.
[33] J. Karlsson, P. Liden, P. Dahlgren, R. Johansson, and U. Gunneflo, "Using Heavy-Ion Radiation to Validate Fault-Handling Mechanisms," *IEEE Micro,* vol. 14, no. 1, pp. 8-11, 13-23, Feb. 1994.
[34] J. Karlsson, P. Folkesson, J. Arlat, Y. Crouzet, G. Leber, and J. Reisinger, "Application of Three Physical Fault Injection Techniques to the Experimental Assessment of the MARS Architecture," *Proc. Fifth IFIP Working Conf. Dependable Computing for Critical Applications (DCCA-5),* pp. 267-287, 1995.
[35] J.R. Samson, W. Moreno, and F. Falquez, "A Technique for Automated Validation of Fault Tolerant Designs Using Laser Fault Injection (LFI)," *Proc. 28th Int'l Symp. Fault Tolerant Computing,* pp. 162-167, 1998.
[36] J. Aidemark, J. Vinter, P. Folkesson, and J. Karlsson, "GOOFI: Generic Object-Oriented Fault Injection Tool," *Proc. Int'l Conf. Dependable Systems and Networks (DSN '01),* 2001.

Authorized licensed use limited to: LOUGHBOROUGH UNIVERSITY. Downloaded on April 30,2010 at 13:22:25 UTC from IEEE Xplore. Restrictions apply.

[37] N.A. Kanawati, G. Kanawati, and J. Abraham, "Dependability Evaluation Using Hybrid Fault/Error Injection," *Proc. IEEE Int'l Computer Performance and Dependability Symp. (IPDS '95)*, p. 0224, 1995.

[38] P.L. Civera, L. Macchiarulo, M. Rebaudengo, M. SonzaReorda, and M. Violante, "Exploiting FPGA for Accelerating Fault Injection Experiments," *Proc. IEEE On-Line Testing Workshop (IOLTW '01)*, pp. 9-13, 2001.

[39] G.S. Choi and R.K. Iyer, "FOCUS: An Experimental Environment for Fault Sensitivity Analysis," *IEEE Trans. Computers*, vol. 41, no. 12, pp. 1515-1526, Dec. 1992.

[40] E. Jenn, J. Arlat, M. Rimén, J. Ohlsson, and J. Karlsson, "Fault Injection into VHDL Models: The MEFISTO Tool," *Proc. 24th Ann. Int'l Symp. Fault-Tolerant Computing*, pp. 66-75, 1994.

[41] T.A. Delong, B.W. Johnson, and J.A. Profeta III, "A Fault Injection Technique for VHDL Behavioral-Level Models," *IEEE Design and Test of Computers*, vol. 13, no. 4, pp. 24-33, Winter 1996.

[42] K.K. Goswami, R.K. Iyer, and L.T. Young, "DEPEND: A Simulation-Based Environment for System Level Dependability Analysis," *IEEE Trans. Computers*, vol. 46, no. 1, pp. 60-74, Jan. 1997.

[43] B. Parrotta, M. Rebaudengo, M. SonzaReorda, and M. Violante, "New Techniques for Accelerating Fault Injection in VHDL Descriptions," *Proc. Int'l On-Line Test Workshop (IOLTW '00)*, pp. 61-66, 2000.

[44] S. Mitra, N.R. Saxena, and E.J. McCluskey, "A Design Diversity Metric and Analysis of Redundant Systems," *IEEE Trans. Computers*, vol. 51, no. 5, pp. 498-510, May 2002.

[45] E. Touloupis, J.A. Flint, V.A. Chouliaras, and D.D. Ward, "Modelling Multiple Faults in Fault-Tolerant Processor Architectures," *IEE Electronics Letters*, vol. 41, no. 21, pp. 1162-1163, 2005.

[46] M.R. Guthaus, J.S. Ringenberg, D. Ernst, T.M. Austin, T. Mudge, and R.B. Brown, "MiBench: A Free, Commercially Representative Embedded Benchmark Suite," *Proc. Fourth IEEE Ann. Workshop Workload Characterization*, Dec. 2001.

[47] *The SPARC Architecture Manual Version 8*, http://www.sparc.org/standards/V8.pdf, 2007.

Emmanuel Touloupis received the diploma in electrical engineering and computer technology from the University of Patras, Greece, in 2002 and the PhD degree from Loughborough University, United Kingdom, in 2006 for his work on the design and testing of fault-tolerant embedded microarchitectures. He is currently a microelectronics engineer with InAccess Networks. His main research interests include embedded processor architectures, fault-tolerant computing, system-on-chip design, communication networks, and safety-critical systems. He is a member of the IEEE.

James A. Flint received the MEng and PhD degrees in electronic and electrical engineering from Loughborough University in 1996 and 2000, respectively. He was with the automotive industry as a project engineer. He became a lecturer in wireless systems engineering at Loughborough University in 2001 and was named a senior lecturer in 2006. He has acted as a consultant of numerous high-profile companies in the wireless and automotive sectors. His research interests include fault-tolerant signal processing, novel acoustic and electromagnetic transducers, metamaterials, and electromagnetic compatibility. He is a Chartered Engineer in the United Kingdom. He is a member of the IEEE.

Vassilios A. Chouliaras received the BSc degree in physics and laser science from Heriot-Watt University, Edinburgh, in 1993 and the MSc degree in VLSI systems engineering from the University of Manchester Institute of Science and Technology (UMIST) in 1995. He was an ASIC design engineer with INTRACOM and a senior research and design engineer/processor architect with ARC International. He is currently a senior lecturer in the Department of Electronic and Electrical Engineering at Loughborough University, where he leads the research in CPU architecture and microarchitecture, SoC modeling, and software parallelization. His research interests include superscalar and vector CPU microarchitecture, high-performance embedded CPU implementations, performance modeling, custom instruction set design, and Electronic Design Automation (EDA). He is a member of the IEEE.

David D. Ward received the BA (with honors) and MA degrees in natural sciences (physics and theoretical physics) from Churchill College, University of Cambridge and the PhD degree from the University of Nottingham. He was a research assistant in the Department of Electrical and Electronic Engineering at the University of Nottingham, developing a 3D numerical model of the electromagnetic effects of the lightning return stroke. In 1991, he joined the staff of the Electromagnetic Compatibility (EMC) Department at MIRA Ltd., UK. He is currently with the Electrical Engineering Department, MIRA, where he is responsible for advanced engineering. He is active in consultancy and technology development in a number of areas concerned with vehicle electronics, related systems and functional safety, including EMC, embedded software, communications systems, and intelligent transportation systems. He is a member of the IEEE.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.

Authorized licensed use limited to: LOUGHBOROUGH UNIVERSITY. Downloaded on April 30,2010 at 13:22:25 UTC from IEEE Xplore. Restrictions apply.