Characterization of a RISC-V Microcontroller Through Fault Injection

Dario Asciolla, Luigi Dilillo, Douglas Almeida dos Santos, Douglas Melo, Alessandra Menicucci, Marco Ottavi

Title: Characterization of a RISC-V Microcontroller Through Fault Injection

Author(s): Dario Asciolla, Luigi Dilillo, Douglas Santos, Douglas Melo, Alessandra Menicucci, Marco Ottavi, Sergio Saponara and Alessandro De Gloria

DOI: 10.1007/978-3-030-37277-4_11

Published: 21 March 2020

Document version: Post-print version (Final draft)

Please cite the original version:
Dario Asciolla, Luigi Dilillo, Douglas Santos, Douglas Melo, Alessandra Menicucci, Marco Ottavi, Sergio Saponara and Alessandro De Gloria. "Characterization of a RISC-V Microcontroller Through Fault Injection," Applications in Electronics Pervading Industry, Environment and Society, 2020, pp. 91-101, doi: 10.1007/978-3-030-37277-4_11.

Characterization of a RISC-V Microcontroller through Fault Injection

Dario Asciolla*, Luigi Dilillo*, Douglas Santos†,
Douglas Melo†, Alessandra Menicucci‡ and Marco Ottavi§
*LIRMM University of Montpellier, Montpellier, France
†Lab. of Embedded and Distributed Systems University of Vale do Itajaí, Itajaí, Brasil
‡Delft University of Technology,
Faculty of Aerospace Engineering, Space Systems Engineering, Delft, The Netherlands
§Department of Electronic Engineering, University of Rome Tor Vergata, Italy

Abstract

This article reports the results of fault injection on a microcontroller based on the RISC-V (Riscy) architecture. The fault injection approach uses fault simulation based on Modelsim and targets a set of 1000 fault injected per microcontroller block and per benchmarck. The chosen benchmarks are the Dhrystone and CoreMark that may represent generic workloads. The results show certain block are more prone to fault than others, as also confirmed by a vulnerability analysis that correlates the number of observed faults and the rate of access to the blocks.

I. Introduction

The space environment interaction with electronics represents an important challenge for satellite missions. Ionizing particles and electromagnetic radiations affect electronic devices by inducing faults in specific circuit areas that may lead to system failures. These failures can be temporary, with the occurrence of the so-called Soft Errors, or permanent with the occurrence of Hard Errors [1]. Moreover, the exposition of electronics to radiation induces to premature aging, with the deterioration of performance and/or functionality of the systems. For this reason, the space industry resorts to hardening techniques that are generally based on hardware and software redundancy, with the generation of custom devices. This type of solutions, while effective, are energy and hardware greedy and leads to costs several times higher than for conventional COTS (Commercial Off The Shelf) electronics.

Completely programmable hardware platforms such as FPGA make the development of a system extremely flexible but, at the same time, require ad-hoc designing and therefore they are not available to a broad programming community. On the other side, the use of standard processors ISA architectures allows many developers to design applications, but the closeness of the architectures does not allow the designers to make easy and cheap modifications to the underlying hardware.

RISC-V allows developers to combine the advantages of both worlds [2], providing flexibility to both hardware and software.

The purpose of this paper is characterizing a RISC-V core, through an extensive simulation-based fault injection campaign with the target of identifying the most critical modules within the core. For this purpose, the study of sequential modules is fundamental, especially for COTS components, because they can suffer from bit flips [3]. The data stored in the memory element can be corrupted, with Single Event Upsets (SEUs), after the interaction with ionizing particles and electromagnetic radiations in general.

II. Platform

The chosen RISC-V platform is the Parallel Ultra Low Power (PULP) Platform. It is been designed from Integrated Systems Laboratory (IIS) of ETH Zürich and Energy-efficient Embedded Systems (EEES) group of the University of Bologna [4]. It is an open-source platform and very useful for our purposes, because it is possible to access all parts of the system. In this specific case, from this platform, the Pulpino microcontroller has been chosen.

Riscy core architecture overview [5] is shown in Fig. 1. For the characterization campaign, we are focusing over all sequential modules inside the core.

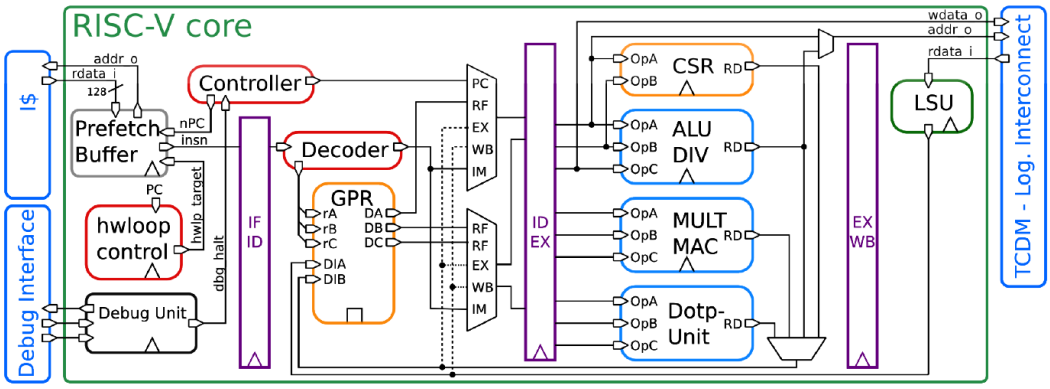


Fig. 1. RISC-V Riscy core Architecture Overview

III. Fault injection environment

For replicating the typical effects of space radiation environment on electronics [6], a simulation-based fault injection technique was chosen. This technique allows full access to the entire processor without any architectural modifications. The first step of the procedure, a golden simulation, with no injected fault , is performed to obtain the reference data to be used for the detection of mismatches caused by fault injections. The Riscy core fault injection is performed for all sequential subsystems. For each sequential subsystem, 1000 simulations are performed, 1 fault injected per simulation run. The following steps [7] summarize the tasks executed for each simulation:

1) Selection of a flip-flop, in a certain sequential subsystem, where the fault will be injected. This is done selecting, in a random way, from a list that contains all signals, in the VHDL code, which implement registers. Each signal corresponds to each bit of the register.
2) Selection of a random instant when the fault will be injected. In order to avoid a fault during the logging process, the fault is injected before the reporting process.
3) Simulation runs until the chosen injection instant.
4) Injection of the fault by forcing a bit flip in the target sequential element.
5) Simulation runs until the end of the algorithmic benchmark.
6) Making a copy of the register file content.
7) Storing the print out of the program results.

Data obtained during the simulation campaign are used to classify fault effects that can be summarized in five categories [7] that are listed below:

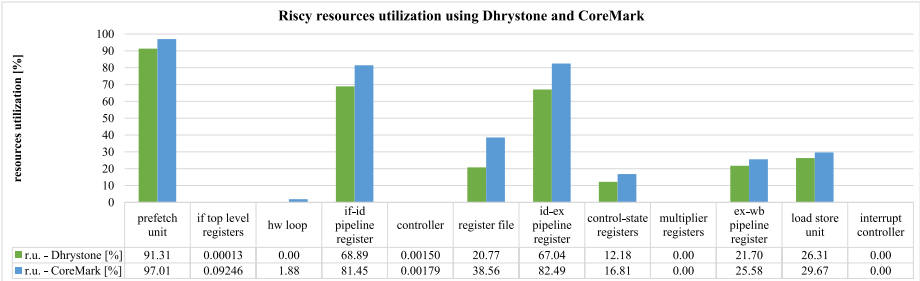
- *No Effect* - The simulation finishes obtaining the correct result from the program and the content of the register file is equal to the reference one.
- *Latent* - The simulation finishes obtaining the right result, but the content of the register file is not equal to the reference.
- *Wrong result* - The system has a failure and the simulation finishes obtaining the wrong program result.
- *Timed out* - The simulation takes an abnormal amount of time to finish the program execution compared to the reference.
- *Exceptions* - The core generates exceptions during the simulation.

Latent errors potentially can propagate and lead to a system failure in the future, but these errors may also be masked by the normal core functioning.

IV. Simulation Results

This section presents and discusses the results of the fault injection campaign and the measure of the utilization of sequential modules. These simulations are useful for vulnerability estimation. The chosen benchmarks, for this campaign, are Dhrystone and CoreMark, they offer a generic workload that can cover almost all operations that a core can execute.

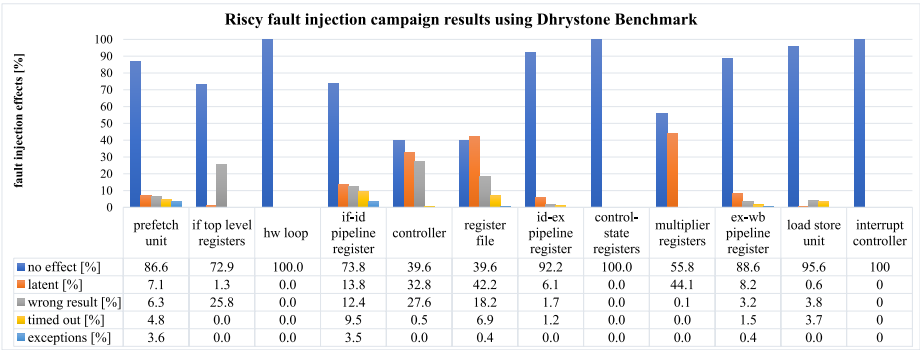
The resources utilization has been measured using a Modelsim simulation running a TCL script. Coremark is configured to perform 1 cycle while Dhrystone 1000 cycles. In this study, the focus is over the sequential parts inside modules that are the target of the characterization through fault injection. A simulation both for Dhrystone and CoreMark is performed obtaining the resources utilization for the entire simulation time.



	prefetch unit	if top level registers	hw loop	if-id pipeline register	controller	register file	id-ex pipeline register	control-state registers	multiplier registers	ex-wb pipeline register	load store unit	interrupt controller
r.u. - Dhrystone [%]	91.31	0.00013	0.00	68.89	0.00150	20.77	67.04	12.18	0.00	21.70	26.31	0.00
r.u. - CoreMark [%]	97.01	0.09246	1.88	81.45	0.00179	38.56	82.49	16.81	0.00	25.58	29.67	0.00

Fig. 2. RISC-V Riscy sequential resources utilization

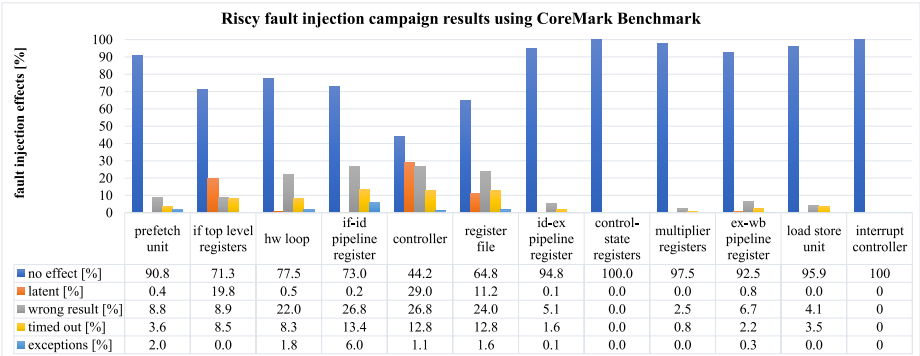
The workload is similar for both simulations, as shown in Fig. 2, for Dhrystone and CoreMark. In Fig. 3 are shown the results of the simulation campaign using Dhrystone as workload. As mentioned above, for each microcontroller block, 1000 simulations were performed, with a fault has been injected for each run. This procedure is repeated for each block, obtaining the results showed in the plot. Fig. 4 shows the results of the simulation campaign, using



	prefetch unit	if top level registers	hw loop	if-id pipeline register	controller	register file	id-ex pipeline register	control-state registers	multiplier registers	ex-wb pipeline register	load store unit	interrupt controller
no effect [%]	86.6	72.9	100.0	73.8	39.6	39.6	92.2	100.0	55.8	88.6	95.6	100
latent [%]	7.1	1.3	0.0	13.8	32.8	42.2	6.1	0.0	44.1	8.2	0.6	0
wrong result [%]	6.3	25.8	0.0	12.4	27.6	18.2	1.7	0.0	0.1	3.2	3.8	0
timed out [%]	4.8	0.0	0.0	9.5	0.5	6.9	1.2	0.0	0.0	1.5	3.7	0
exceptions [%]	3.6	0.0	0.0	3.5	0.0	0.4	0.0	0.0	0.0	0.4	0.0	0

Fig. 3. Fault injection campaign results using Dhrystone Benchmark

CoreMark as workload, with the same procedure used above. Like in the analysis concerning the other benchmark, it can be noticed that the most critical modules are the controller and the register file, which present a large number of latent errors and wrong results. The controller is again accessed with lower frequency w.r.t. the register file, but it displays high vulnerability.



	prefetch unit	if top level registers	hw loop	if-id pipeline register	controller	register file	id-ex pipeline register	control-state registers	multiplier registers	ex-wb pipeline register	load store unit	interrupt controller
no effect [%]	90.8	71.3	77.5	73.0	44.2	64.8	94.8	100.0	97.5	92.5	95.9	100
latent [%]	0.4	19.8	0.5	0.2	29.0	11.2	0.1	0.0	0.0	0.8	0.0	0
wrong result [%]	8.8	8.9	22.0	26.8	26.8	24.0	5.1	0.0	2.5	6.7	4.1	0
timed out [%]	3.6	8.5	8.3	13.4	12.8	12.8	1.6	0.0	0.8	2.2	3.5	0
exceptions [%]	2.0	0.0	1.8	6.0	1.1	1.6	0.1	0.0	0.0	0.3	0.0	0

Fig. 4. Fault injection campaign results using CoreMark Benchmark

V. Vulnerability Estimation

Results obtained from the fault injection campaigns present similar trends for the two workloads. In this section, we try to calculate the vulnerability of each block of the RISC-V Riscy core, by introducing the following equation:

$$v = \begin{cases} \frac{f}{u} \times c, & \text{if } u > 0 \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

- v resource vulnerability.
- f failure rate. It is equal to the number of wrong results normalized to the number of simulations;
- u resource utilization. For each module, it is equal to the number of clock cycles of activity over the total number of clock cycles.
- c normalization constant equal to 100.

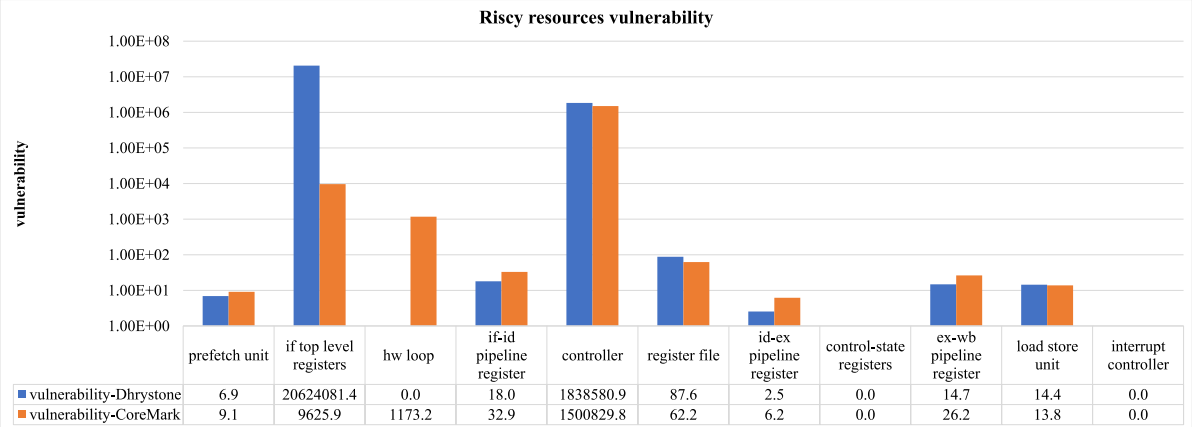
This approach allows to extrapolate a general evaluation of the block vulnerability that is independent of the used benchmark algorithm. Since it is based on the correlation between the amount of detected failures and the actual use of the blocks of the microcontroller. Fig. 5 shows the results of the vulnerability associated with each block.

The plot uses a logarithmic axis to easily visualize the results.

It can be noticed that there is a correlation between the vulnerability calculated from both campaigns. The subsystems that show a high vulnerability are the instruction fetch registers, the controller and the register file.

Lower but significant vulnerability magnitude is showed for the prefetch unit, instruction fetch-instruction decode pipeline register, the instruction decode-execution pipeline register, ex-wb pipeline register and the load store unit.

The limitation of the adopted vulnerability model is due to the occurrence of system failures that are observed also in blocks that are not supposed to be used by the algorithm.



	prefetch unit	if top level registers	hw loop	if-id pipeline register	controller	register file	id-ex pipeline register	control-state registers	ex-wb pipeline register	load store unit	interrupt controller
vulnerability-Dhrystone	6.9	20624081.4	0.0	18.0	1838580.9	87.6	2.5	0.0	14.7	14.4	0.0
vulnerability-CoreMark	9.1	9625.9	1173.2	32.9	1500829.8	62.2	6.2	0.0	26.2	13.8	0.0

Fig. 5. RISC-V Riscy sequential Resources Vulnerability

VI. Conclusion

From simulation results, showed in Fig. 3 for Dhrystone workload and in Fig. 4 for CoreMark workload, the most critical sequential modules are the controller and the register file. Despite the fact that the controller is used with lower frequency than the register file, it causes a large number of failures when it undergoes to fault injection. Exceptions are caused by faults injected in modules inside the instruction fetch stage, in the instruction decoder stage and in the execution-write back pipeline register. CoreMark stimulates the usage of the hardware loop module and we noticed a relevant system failures caused by faults injected inside this module.

The interrupt controller and the control-state registers don't cause any failure when faults are injected.

The most used resources are the prefetch unit, the instruction fetch-instruction decode pipeline register and the instruction decode-execute pipeline register. There are modules that are never used during the benchmark execution like the hardware loop for Dhrystone and the multiplier registers and the interrupt controller for both benchmarks (in the run simulations).

From the study of vulnerability, showed in Fig. 5, the most critical module result to be the controller module.

The Vulnerability can be used to estimate, for each module, the system failure rate when executing other software. This can be done simply making the product between the tabled vulnerability values and the utilization value measured for the given application. These represents an important information for the design of fault-tolerant Risc-V core, since it can be used to evaluate the best redundancy techniques in terms of time usage and hardening impact for each composing block, on the base of its vulnerability.

References

[1] Cristiano Calligaro, Umberto Gatti, *Rad-hard Semiconductor Memories*, ser. Series in Electronic Materials and Devices, 2018.
[2] S. Di Mascio, A. Menicucci, G. Furano, C. Monteleone, and M. Ottavi, "The case for risc-v in space," in *Applications in Electronics Pervading Industry, Environment and Society*, S. Saponara and A. De Gloria, Eds. Cham: Springer International Publishing, 2019, pp. 319–325.

[3] L. Dilillo, G. Tsiligiannis, V. Gupta, A. Bosser, F. Saigné and F. Wrobel, “Soft errors in commercial off-the-shelf static random access memories,” *Journal of Semiconductor Science and Technology*, vol. vol 32, 2016.
[4] Pulp-Platform, “Project info,” [Online]. Available: https://pulp-platform.org/projectinfo.html, Jun. 2017.
[5] ——, “PULPino: Datasheet,” PULPino: Datasheet, Jun. 2017.
[6] V. Gupta, A. Bosser, F. Wrobel, F. Saigne L. Dusseau, A. Zadeh, L. Dilillo, “Mtcube project: See ground-test results and in-orbit error rate prediction,” *The 4S Symposium, Small Satellites Systems and Services Symposium, Valletta, Malta*, 2016.
[7] Rodrigo Travessini, Paulo R. C. Villa, Fabian L. Vargas, Eduardo Augusto Bezerra, “Processor core profiling for seu effect analysis,” *in Test Symposium (LATS)*, 2018.