Prepared for submission to JINST

Towards Single-Event Upset detection in Hardware Secure RISC-V processors

N. Jonckers[a] **B. Engelen**[a] **K. Appels**[a] **S. De Raedemaeker**[a] **L. Mariën**[a] **J. Prinzie**[a]

[a] *KU Leuven, Dept. Electrical Engineering,*
Kleinhoefstraat 4, B-2440 Geel, Belgium

E-mail: nain.jonckers@kuleuven.be

Abstract: Single-event effects and hardware security show close similarities in terms of vulnerabilities and mitigation techniques. Secure processors address physical attacks from the outside, such as external laser stimulation, to compromise the program and extract sensitive information from the systems. To overcome this vulnerability, secure extensions to the hardware architecture are often built into modern processor cores. Given the limited design resources often found in space or high-energy physics experiment development teams, this article addresses the extent to which secure hardware architectures can be a reliable source of processor SEU detection.

Keywords: Radiation-hard electronics, Digital electronic circuits, Radiation damage to electronic components, Single-Event Effects

1 Introduction

1.1 Processing systems in harsh environments

The development of custom Application Specific Integrated Circuits (ASICs) for High-Energy Physics (HEP) has been the subject of numerous R&D decisions in recent decades. Typically, analog front ends are connected to digital readout chipsets that are very specific to a particular detector. While the digital readout systems in detectors used to be simple, they have become more complex over the years and cannot be reconfigured. Until now, microprocessors are not incorporated and used in detectors, but rather custom logic. Processor systems, on the other hand, offer the benefits of standardized organization and a high degree of reconfigurability: software programmability, standardized interconnect busses, and the use of an IP block library. The HEP community is currently exploring the possibility of using these processors for radiation-resistant control, monitoring, data acquisition, and data processing within detectors. The architecture of these processors should be open source to allow deep control of software and hardware implementation for optimal radiation resistant design and verification [1].

Space applications also require fault tolerance. Examples include general-purpose processors for running Linux-like operating systems, on-board computers for real-time operating systems, microcontrollers for distributed simple control applications, data acquisition systems, data processing systems without an operating system, and artificial intelligence processors optimized for large matrix computations with special vector instructions. The advantages of using RISC-V processors for radiation-tolerant systems are that the instruction set and most of the associated cores are open source, allowing developers to become familiar with the details of the architecture and perform detailed fault injection simulations to develop specialized radiation-tolerant IP. The RISC-V architecture is well-supported by academia and industry with a wide range of IP cores, software, and development tools, enabling rapid and efficient adoption in the community.[2].

1.2 Single-event effects and hardware security

Non-destructive Single-Event Effects (SEEs) are soft errors stimulated by high-energy particles that lead to bit flips in a logic system. SEEs can manifest as Single-Event Transients (SETs) in combinatorial logic or Single-Event Upsets (SEUs) in sequential logic such as flip-flops. SETs can be latched if the transient propagates to a flip-flop and occurs during a clock transition. As such, SEEs can cause erroneous operation in a microprocessor, resulting in incorrect calculations, unpredictable program execution, or severe crashes. The probability of SETs that manifesting themselves in errors increases linearly with the clock frequency due to the increased number of clock edges that occur[3].

Hardware security has a broad context in the design of computer architectures [4]. Many implementations today provide secure extensions to prevent malicious physical attacks to corrupt or steal data from the system. Certain attacks attempt to compromise the program through laser or EMI stimulation aimed at inducing errors in the system to extract information. Thus, if secure extensions are able to detect these errors, they can potentially be useful in radiation environments.

2 Ibex RISC-V Core

The Ibex core is an open-source 32-bit RISC-V processor, initially designed by ETH Zurich and the University of Bologna, and now maintained by lowRISC. It implements either the 32-bit integer (`I`) or 32-bit embedded (`E`) RISC-V base standard. On top of this base standard, the core also implements the multiplication and division (`M`), compressed instructions (`C`) and bit manipulation (`B`) extensions. These extensions and the base instruction set are implemented in a two-stage pipeline with an optional third "write-back" stage [5].

Since the parameters of the Ibex core are highly parametrisable, many different "flavors" of this core are possible. The developers define four different parametrized configurations, namely: *"micro"* (RV32EC), *"small"* (RV32IMC w. 3-cycle multiplier), *"maxperf"* (RV32IMC w. 1-cycle multiplier, branch target ALU and writeback stage) and *"maxperf-pmp-bmfull"* (RV32IMCB w. 1-cycle multiplier, branch target ALU, writeback stage and 16 PMP regions) [6].

This research focuses itself on the **"small"** configuration since it was the only configuration that, at the time of writing, was fully verified.

2.1 Security features

Aside from the `RV32IMC` RISC-V specification, the *"small"* configuration of the Ibex core also implements several security features. These features allow the core to detect malicious tampering at runtime, significantly increasing it's hardware security to external attacks. To signal a possible security alert, the Ibex core provides three output signals, namely `alert_major_internal_o`, `alert_minor_o` and `alert_major_bus_o` that respectively generate a trigger if there was a possibly malicious major, minor or system bus event [7].

Since these features make use of a.o. Error Correcting Code (ECC), register glitch detection, dual core lockstepping, etc. [7], they may also be suited to detect SEUs and/or SETs stimulated by radiation. This hypothesis is further investigated in this paper and will turn out, given a minor modification, to be true.

3 Test methodology

In order to validate the applicability of Ibex's security mechanisms for SEE detection, a framework was developed to inject faults into the processor core. In this section, we describe the Register Transfer Level (RTL) simulation framework to inject bitflips into the code of the Ibex core. We focussed at injecting bitflips into flipflops (i.e. SEUs) since SETs could become negligible in downscaled CMOS nodes.

3.1 Test flow

When injecting faults into flipflops at RTL-level, each flipflop must be identified to acquire its hierarchical name. A generic synthesis was performed (= elaboration) using Cadence® Genus™ to obtain a list of flipflop register (as indicated in figure 1). During simulation this list is used to target the flipflops to stimulate SEUs.

Aside from this flipflop list, the simulator also needs an application software programme that runs on the Ibex processor. Ideally this programme should use the majority of the RISC-V

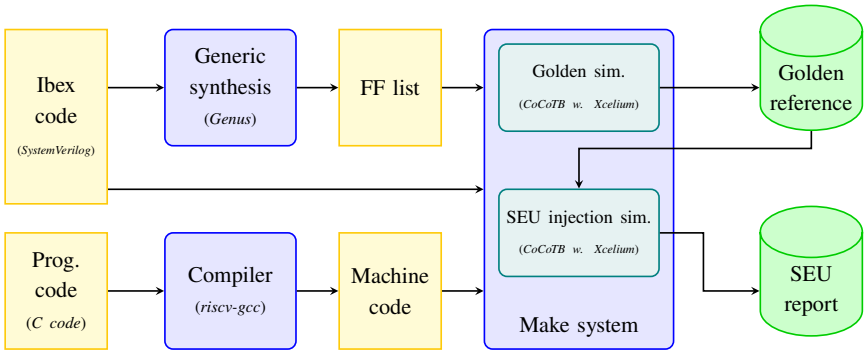

Figure 1: Block diagram of the test flow

instructions so that a good coverage is guaranteed when injecting bitflips. To obtain this goal, we used the Dhrystone (version 2) benchmark. Since Dhrystone is a benchmark programme, it should use (almost) all RISC-V instructions supported by the Ibex core to give a representative benchmark figure. This benchmark is also relatively lightweight and easy to set up. It is not as accurate when compared to more modern benchmarks like Coremark®, but this is hardly a concern for the presented SEU simulations [8, 9]. This Dhrystone programme is compiled (using GCC, the GNU Compiler Collection) into machine code which can be loaded into the (virtual) instruction memory of the Ibex core.

Once both the flipflop list and the machine code are generated, these two outputs, together with the Ibex SystemVerilog RTL-code, are used by the simulator to launch a simulation run.

The CoCoTB python library, together with Cadence® Xcelium™ as a backend simulator was used to implement the test framework. We use the CoCoTB python library to make the injection of bitflips easier. Since CoCoTB allows us to call any net from the top-level module using the `_id("net-name", extended=False)` method, we can easily inject bitflips by XOR-ing them with the required bit index: $net = net \hat{} (1 << bit_n)$. In addition, the concurrent programming nature of CoCoTB allows us to design a highly structured and clear to read testbench [10].

3.2 Golden reference model

Simulations are performed in two passes. In the first pass, a golden fault-free reference model is generated, followed by a batch of SEU runs. The golden reference model will be the reference to the corresponding SEU injection run simulations. In order to monitor the state of the processor core, a 32-bit Cyclic Redundancy Check (CRC) checksum is calculated on all the output signals on a cycle-by-cycle basis. The golden CRC database is generated during the golden reference simulation. This checksum can then be compared during the SEU injection run to asses whether a stimulated SEE propagates to the output.

3.3 SEU run

Once the golden reference model is generated, we can rerun the simulation and inject bitflips using the extracted flipflop list described in section 3.1. As shown in figure 2, we run the simulation for a random time and then pick a random flipflop to stimulate a bitflip. Afterwards, we give the SEU $check\_interval = 10$ cycles to propagate to the output, and then we compare the current CRC to the golden reference CRC for that clock cycle. If no CRC error was found, the simulation continues and a new SEU is stimulated. Otherwise, the simulation is terminated and the results are logged. Fault injection runs are massively dispatched in parallel to achieve statistical relevant data.

4 Results

When running an SEU injection simulation, we classify four possible outcomes. (i: Detected) The SEU leads to a CRC error but is flagged by the Ibex alert signals; (ii: Undetected) the SEU leads to a CRC error but is not flagged by the Ibex alert signals; (iii: Silent) the SEU does not lead to a CRC error and no alert was raised; or (iv: False positive) the SEU does not lead to a CRC error but an alert was raised.

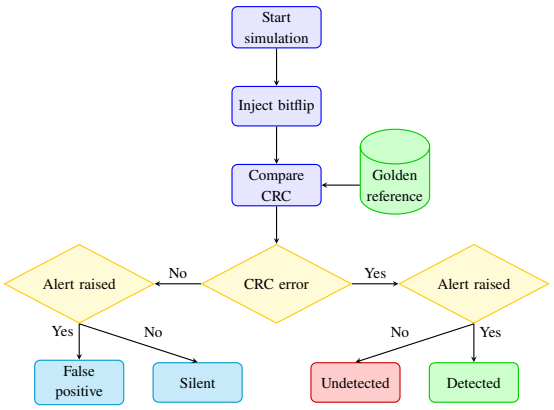

Figure 2: Single-Event Upset injection simulation flow.

Table 1 shows the total number of injected SEUs per logical block of the Ibex core, together with the false positives, detected and undetected bitflips. An injection is considered a false positive if the Ibex core put any alert signal high but there was no error observed at the output. If any alert signal went high and an error was observed, or when there was no error and no alert went hight, then the injection is considered as detected. When an error was observed but the core did not assert any alert signal, only then is the injection considered as undetected.

In these results, we can clearly see that the Ibex core does not detect all faults injected, namely some injected faults in the register file are not detected. After some code tracing, we found out that if we change the code below in the source file `ibex_core.sv`, that we are able to detect all faults in the register file as well.

```
// Calculate errors - qualify with WB forwarding to avoid xprop into the alert
↪ signal
assign rf_ecc_err_a_id = |rf_ecc_err_a & rf_ren_a & ~rf_rd_a_wb_match;
assign rf_ecc_err_b_id = |rf_ecc_err_b & rf_ren_b & ~rf_rd_b_wb_match;
// Combined error
assign rf_ecc_err_comb = instr_valid_id & (rf_ecc_err_a_id | rf_ecc_err_b_id);
```

→

```
// Calculate errors - qualify with WB forwarding to avoid
↪ xprop into the alert signal
assign rf_ecc_err_a_id = |rf_ecc_err_a;
assign rf_ecc_err_b_id = |rf_ecc_err_b;
// Combined error
assign rf_ecc_err_comb = (rf_ecc_err_a_id | rf_ecc_err_b_id);
```

It seemed that some enable signals inhibited the correct functionality of the alert signals sometimes. If we therefore remove these signals from the combined alert signal, then we detect all errors in the register file. Unfortunately, this also comes at a cost since we now have an increased number of false positives, which is not desirable. Table 2 shows the SEU injection results with the above code fix.

Table 1: Results of fault injection per logical block of the Ibex core

Ibex block	**Injected SEUs**	**Detected**	**Undetected**	**Silent**	**False positives**
Fetch	16721	4400	0	10782	1539
Decode	5754	230	0	5516	8
Execute	3702	0	0	3699	3
LSU	3224	171	0	1555	1498
CSR	22590	155	0	19354	3081
Regfile	56537	13295	**8686**	30694	3862
Lockstep core	173601	0	0	73997	99604
Others	242	0	0	241	1
Total	282371	18251	8686	145838	109596
Total (%)	100	6.46	3.08	51.65	38.81

Table 2: Results of fault injection per logical block of the Ibex core - with code fix

Ibex block	**Injected SEUs**	**Detected**	**Undetected**	**Silent**	**False positives**
Fetch	14995	4037	0	0	10958
Decode	5098	221	0	0	4877
Execute	3165	0	0	0	3165
LSU	2888	158	0	0	2730
CSR	20419	124	0	0	20295
Regfile	50796	19839	0	0	30957
Lockstep core	156316	0	0	0	156316
Others	230	0	0	0	230
Total	253907	24379	0	0	229528
Total (%)	100	9.60	0.0	0.0	90.40

Finally, we have also validated the area overhead of these security features. We have done this by synthesizing the Ibex core with and without security features in a 180 nm CMOS bulk technology. This resulted in an estimated area of $0.62mm^2$ with security features and $0.27mm^2$ without. In other words, the Ibex security features come at a cost of roughly 130% area increase.

5 Conclusions

The hardware secure Ibex RISC-V core was tested to see if it could alert SEUs. The RTL code of the Ibex core was embedded in a CoCoTB simulation framework that implements the Dhrystone benchmark as an application program. Thanks to various protections such as a lockstepped core, a double program counter, and ECC protection in the register file, alerts were triggered when SEEs were injected into the design. With a small code change, the core was able to detect all injected SEUs, but at the cost of an increased number of false positives. From this, we can conclude hardware secure features can be used for radiation hardness applications.

References

[1] M. Andorno et al. “Rad-hard RISC-V SoC and ASIP ecosystems studies for high-energy physics applications”. In: *Journal of Instrumentation* 18 (2023). Publisher: IOP Publishing, p. C01018. DOI: 10.1088/1748-0221/18/01/C01018.

[2] Gianluca Furano et al. “A European Roadmap to Leverage RISC-V in Space Applications”. In: *2022 IEEE Aerospace Conference (AERO)*. 2022, pp. 1–7. DOI: 10.1109/AERO53065.2022.9843361.

[3] Kruckmeyer K Baumann R. *Radiation Handbook for Electronics*. Texas Instruments, 2019.

[4] Nachiketh Potlapally. “Hardware security in practice: Challenges and opportunities”. In: *2011 IEEE International Symposium on Hardware-Oriented Security and Trust*. IEEE. 2011, pp. 93–98.

[5] *Ibex documentation*. https://ibex-core.readthedocs.io/en/latest. Valid for commit 45b7272.

[6] *Ibex Github repository*. https://github.com/lowRISC/ibex. Valid for commit 45b7272.

[7] *Ibex security features*. https://ibex-core.readthedocs.io/en/latest/03_reference/security.html. Valid for commit 45b7272.

[8] Reinhold P Weicker. “Dhrystone benchmark: rationale for version 2 and measurement rules”. In: *AcM SIGPLAn notices* 23.8 (1988), pp. 49–62.

[9] Alan R Weiss. “Dhrystone benchmark”. In: *History, Analysis,„Scores “and Recommendations, White Paper, ECL/LLC* (2002).

[10] *CoCoTB documentation*. https://docs.cocotb.org/en/stable/index.html. Valid for version 1.7.2.