ORSHIN

D3.1

RISC-V processor with countermeasures against microarchitectural attacks

Project number:	101070008
Project acronym:	**ORSHIN**
Project title:	Open-source ReSilient Hardware and software for Internet of thiNgs
Project Start Date:	1st October, 2022
Duration:	36 months
Programme:	HORIZON-CL3-2021-CS-01
Deliverable Type:	Demonstrator
Reference Number:	CL3-2021-CS-01 / D3.1 / 1.0
Workpackage:	WP3
Due Date:	Jun 2025 - M33
Actual Submission Date:	30th June 2025
Responsible Organisation:	KUL
Editor:	Frank Piessens
Dissemination Level:	PU
Revision:	1.0
Abstract:	This deliverable describes Proteus, a RISC-V processor designed for microarchitectural security research.
Keywords:	side-channel attacks, microarchitectural attacks, RISC-V, hardware security, Proteus, ProSpeCT, architectural mimicry, Libra

Funded by the European Union under grant agreement no. 101070008. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

Editor

Frank Piessens(KUL)

Contributors (ordered according to beneficiary numbers)

Marton Bognar, Lesly-Ann Daniel, Job Noorman, Frank Piessens, Hans Winderix (KUL)

Reviewers

Aurélien Francillon (ECM)
Jan Pleskac (TRPC)

Disclaimer

Executive Summary

The ORSHIN project has as a goal to build secure and open-source hardware. In Task 3.1, we designed formal models for reasoning about microarchitectural side-channel leakage and built several countermeasures. This deliverable describes the RISC-V processor, Proteus, we built to allow experimenting with these countermeasures. It also contains details of the countermeasures and their implementations. We chose RISC-V due to its open nature and ease of extensibility with custom instructions and registers, which are used in all of our security extensions.

Proteus is a software-defined RISC-V CPU built with the explicit goal of supporting security research by allowing users to quickly develop, validate, and evaluate hardware extensions. Much of the core functionality of Proteus is implemented using a plugin system, simplifying the process of enabling or modifying certain functionalities. It features textbook implementations of an in-order and an out-of-order pipeline, making it possible to evaluate extensions in different deployment scenarios aligning with embedded or high-end settings. To support robust security validation, we propose to closely integrate several different security testing and verification tools with the processor, together with benchmark suites for performance evaluation.

We used Proteus to build three security extensions in this project: ProSpeCT, Architectural Mimicry, and Libra. ProSpeCT proposes a formally verified defense mechanism against all known Spectre attack variants. Architectural Mimicry extends the processor with new “mimic” instructions that imitate the side effects of regular instructions without affecting the program state, enabling novel linearization and balancing techniques in software. Libra introduces a novel program transformation, “instruction folding”, which, together with minor hardware changes, enables eliminating control flow leakage.

The code of the base Proteus processor, the extensions and the evaluation code are all publicly available under permissive licenses. Two of the extensions have also successfully participated in artifact evaluation, where an artifact evaluation committee has determined that both satisfy the highest requirements for reusability.

Table of Content

List of Figures

Chapter 1

Introduction

One of the goals of the ORSHIN project, and specifically of WP3 is building prototypes for mitigations of microarchitectural side channels. This report describes the motivation behind us building a RISC-V processor for this purpose, and the security extensions we built on it as part of this project.
In contrast to the relatively well-established practices of developing and evaluating software-based countermeasures through codebases like the Linux kernel and software benchmarks, research into hardware changes is still fairly experimental. Early suggestions for hardware changes often remained theoretical, and making precise claims was difficult as hardware manufacturers are historically closed-off and do not openly publish their designs. A partial solution to this challenge is the use of architectural simulators such as gem5 [1] or Simics [15]. While these enable conducting performance evaluations on effects such as cache or TLB hit rates, they do not give a good indication of the incurred complexity of the hardware. Evaluating the changes to the area and the critical path is essential when considering which mitigations to deploy on real hardware, but this can only be measured by changing a real hardware design. Moreover, these simulation tools are far less suited to reason about attacks exploiting physical phenomena in hardware such as physical fault injection or power side-channel attacks.

RISC-V. The ratification of the open RISC-V instruction set in 2019 [18] has caused a noticeable surge in the development of open-source processors. RISC-V can be freely used in processor designs and was designed to target CPUs across the computing spectrum, from small microcontrollers to desktop and server CPUs. In addition, the instruction set was designed to be easily extensible, which encourages research proposals introducing new instructions to the standard. Today, RISC-V processors are appearing in commercial products by companies such as Microchip and Milk-V, and already in 2023 almost 10% of AI accelerators and 5% of microcontrollers shipped were RISC-V chips [20].
Open-source RISC-V processors are also increasingly being used in research. Popular low-end targets include the Ibex [13] chip (previously known as zero-riscy [5]), part of the openTitan project [14] designing a root-of-trust for embedded systems; and CVA6 [26]. There are also out-of-order RISC-V cores enabling research for high-end CPUs such as BOOM [27] and XiangShan [25], the latter of which is also being commercialized.[1] Security research into these CPUs is also rapidly evolving with proposals to perform fuzzing [9, 3] or verification [11, 2] on these chips. At the same time, security vulnerabilities have been shown on these commercial RISC-V CPUs [7, 21], underlining the importance of security research on this platform.

[1] https://milkv.io/ruyibook

Proteus. Despite the rising popularity of open-source RISC-V chips and tools operating on them to validate security properties, these largely exist in isolation. The goal of Proteus is specifically to enable quick prototyping of security extensions, including thorough security validation and performance evaluation tools. We tackle this goal from multiple directions. First, Proteus features a plugin system and many configurable options, making it possible to evaluate extensions in different configurations and to develop extensions as new plugins. The available configurations notably include an in-order and an out-of-order pipeline, the latter with a configurable number of parallel execution units. Second, we develop an extensive software ecosystem around the CPU which enables security validation and verification, as well as performance evaluation. As Proteus is written in SpinalHDL [16], which generates Verilog code, it is possible to apply techniques that operate on Verilog code or require the design to run on an FPGA. Compared to other out-of-order cores like BOOM, Proteus is relatively small, making it feasible to run it on affordable FPGAs.

Outline. In this document, we first describe the Proteus processor in Chapter 2, then detail the security extensions built as part of the ORSHIN project in Chapter 3. Finally, we conclude in Chapter 4.

Chapter 2

The Proteus core

This chapter briefly introduces the Proteus RISC-V processor. The code of the processor and the necessary infrastructure to run simulations and flash it to an FPGA can be found on GitHub, along with a hands-on tutorial on its Wiki for researchers and developers wanting to work with Proteus:
https://github.com/proteus-core/proteus/
Aside from the three ORSHIN-related projects described in Chapter 3, Proteus has been used in other projects [19], even by completely independent research teams [10], showing its versatility and usability.

2.1 Pipelines

Proteus features implementations of an in-order and an out-of-order pipeline which share large parts of the codebase. These pipeline implementations enable experimenting with a given security extension on different designs, the in-order pipeline representing a low-end IoT device, and the out-of-order design representing a more complex and performant microprocessor. Most of the functionality is implemented using a plugin system similar to that of VexRiscv [17]. This plugin system enables easy configuration of different components, such as optionally including the RISC-V extension for multiplications and divisions, and it allows reusing many of the same building blocks in the in-order and the out-of-order pipeline.
The plugins can either hook into a pipeline stage and define operations on the pipeline registers (e.g., defining a new arithmetic operation), introduce new pipeline registers and define how the decoder should populate these for different instructions, or provide functionality across the whole pipeline (cf. Figure 2.1). Plugins can also cooperate through interfaces called *services*. Services can be invoked across the pipeline to provide data or operations for other components. An example is the ALU service, which can be invoked from the branch operation plugin to calculate the target addresses of branch instructions.

2.1.1 In-order pipeline

The in-order pipeline is flexible in the number and order of stages: the plugins implementing different parts of the functionality can be assigned to arbitrary stages. For the published version, these are configured in a textbook 5-stage setting with fetch, decode, execute, memory, and writeback stages. Figure 2.1 shows the structure of the pipeline with two plugins highlighted. The stages are indicated at the bottom of the figure with the pipeline registers forwarding data from one stage

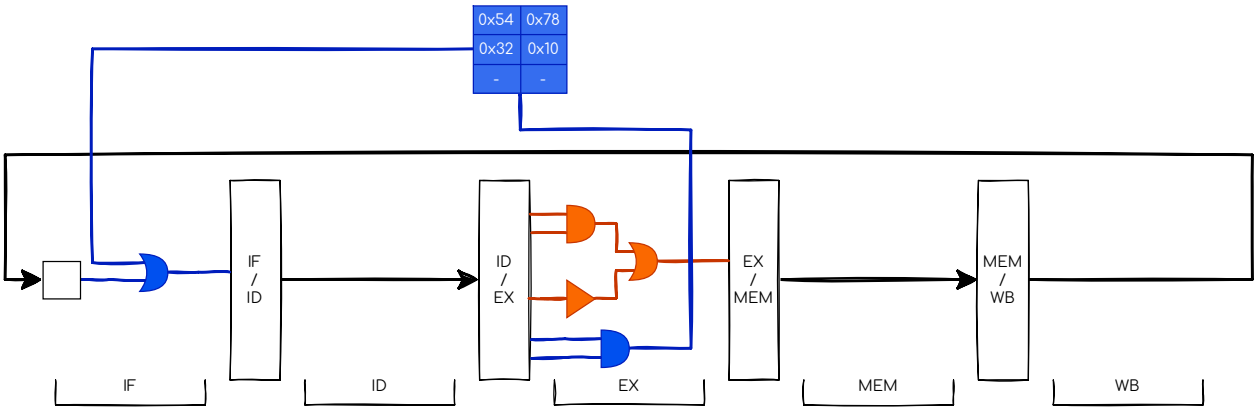

Figure 2.1: The in-order pipeline of Proteus showcasing the plugin system. The blue plugin spanning multiple stages is a branch target predictor (BTB), while the orange plugin represents an arithmetic operation. The plugins are only illustrations, they do not represent the real logic gates used.

to the next positioned between the stages. Plugins can either add functionality to a single stage, such as the orange plugin representing an arithmetic operation that performs a transformation on the values in the pipeline registers; or span across multiple stages and components, such as the blue plugin representing a branch predictor which analyzes and stores branch targets from the execute stage and influences the instruction fetching logic in the first stage.

2.1.2 Out-of-order pipeline

The out-of-order design is based on Tomasulo's algorithm [8] and features a reorder buffer (ROB) with a configurable number of entries performing register renaming; a number of execution units wrapped in reservation stations handling instruction dependencies; and load buffers performing load operations after the target addresses are calculated in an execution unit. The results of executed instructions and completed loads are communicated on the common data bus (CDB) to allow dependent instructions to start executing immediately after the dependency is resolved. Instructions marked as ready in the ROB are retired in program order, and stores and CSR operations are also performed in this stage.

2.2 Software infrastructure

To ease development with Proteus and the reproduction of results, the processor ships with a Docker container providing extensive functionality. This container is set up to contain the necessary toolchain to compile and run programs on the simulated processor, and also provides different benchmarks and evaluation tools (e.g., providing security testing and hardware measurement costs).

Chapter 3

Security extensions

This chapter provides a brief introduction of the security extensions of Proteus developed as part of the ORSHIN project. These extensions were implemented to show the feasibility and enable the evaluation (performance overhead, security guarantees, hardware cost) of these designs. For further information about the details of the developed security models and their security guarantees, we refer to Deliverable 3.2 and the publications [4, 22, 23].
The source code for the extensions and the evaluations are publicly available in the following repositories:

- https://github.com/proteus-core/prospect
- https://github.com/proteus-core/ami
- https://github.com/proteus-core/libra

Notably, the code for ProSpeCT and Libra went through the artifact evaluation process of USENIX Security '23 and ACM CCS '24 respectively, earning the highest level of badges, showing that the evaluators independently reproduced our evaluation results.

3.1 ProSpeCT

ProSpeCT [4] is a countermeasure against all known Spectre [12] variants, which works by preventing secret values from being forwarded to unsafe instructions during speculation.

Implementation. This extension was the first to make use of the out-of-order pipeline and implemented both speculation tracking and secret taint tracking. To mark secret sections of the memory, CSRs storing the boundary addresses of secret regions were introduced. Most of the modifications relate to the reservation stations, making sure that the secret and speculation taints are correctly propagated, and that instructions cannot start executing if they are speculative and use a secret value.

Evaluation. The design and implementation were evaluated in three ways: The hardware cost was estimated by synthesizing the design to an FPGA. The performance overhead was measured using the SpectreGuard [6] benchmarks with the protection applied to secrets grouped in one memory region. Finally, the developer effort required to manually mark secrets was measured by annotating cryptographic primitives in HACL* [28].

3.2 Architectural Mimicry

Architectural Mimicry introduces the concept of mimic execution, a new processor mode that executes instructions without writing their results to the register file to protect against attackers that observe the program's side-channel leakage. It also adds new *activating* instructions to toggle this processor mode and qualifiers to existing instructions to make their behavior depend on the processor mode. These changes allow writing performant and secure balanced or linearized code.

Implementation. The extension was added to both the in-order and the out-of-order pipeline. Both implementations make use of three internal registers for keeping track of the processor state and activation conditions, which have to be updated by the activating instructions enabling mimic mode. Then, based on the instruction qualifiers and the processor mode, the retirement stage decides whether the result should be committed.
On the out-of-order pipeline, instruction dependencies need to be tracked explicitly, as forwarding the value of a mimicked instruction or not waiting for the result of a mimicked instruction could lead to either functional or security issues. To keep track of activation changes, a queue of activating instructions was also introduced in the ROB.

Evaluation. Both the security and the performance evaluation were conducted on a subset of the Winderix benchmarks [24]. This was the first publication to introduce a more rigorous framework that tests the leakage of the benchmarks during simulation with different inputs. The hardware cost was also estimated by synthesizing the design to an FPGA.

3.3 Libra

Libra proposes a code transformation technique called *folding* that interleaves the instructions of secret-dependent branches in the compiled binary. This transformation, combined with changes in the hardware, ensures that secret-dependent branches can be executed without leaking the secrets.

Implementation. Libra adds a new instruction called the level-offset branch, which informs the hardware about the upcoming folded secret-dependent branch to enable its correct and secure execution. In the pipeline, the fetching unit needs to be modified to take the new code layout into account and to prevent instruction accesses from leaking secrets. The only other required change was disabling the branch target buffer (BTB) in secret-dependent regions.

Evaluation. Similar to Architectural Mimicry, the work used a subset of the Winderix benchmark suite [24] for both security and performance evaluation and used a similar approach for validating the security. The hardware cost was estimated by synthesizing the design to an FPGA.

Chapter 4

Summary and Conclusion

This deliverable described Proteus, a RISC-V processor developed as part of the ORSHIN project to enable rapid prototyping and evaluation of security extensions. The processor features a plugin system providing most of the functionality, and implementations of in-order and out-of-order pipelines, allowing to evaluate extensions that target heterogeneous devices. We also briefly described three security extensions, ProSpeCT, Architectural Mimicry, and Libra, that were developed during the project and that offer diverse security guarantees in the context of microarchitectural security by extending the Proteus core.
All of our code for the base processor and the extensions is publicly available, including a tutorial and the evaluation performed in the publications. Moreover, ProSpeCT and Libra have successfully participated in artifact evaluation at their conferences, where members of a committee have successfully reproduced our results independently.

Bibliography

[1] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, August 2011.

[2] Niklas Bruns, Vladimir Herdt, and Rolf Drechsler. Processor verification using symbolic execution: A risc-v case-study. In *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2023.

[3] Sadullah Canakci, Chathura Rajapaksha, Leila Delshadtehrani, Anoop Mysore Nataraja, Michael Bedford Taylor, Manuel Egele, and Ajay Joshi. Processorfuzz: Processor fuzzing with control and status registers guidance. In *IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2023, San Jose, CA, USA, May 1-4, 2023*, pages 1–12, 2023.

[4] Lesly-Ann Daniel, Marton Bognar, Job Noorman, Sébastien Bardin, Tamara Rezk, and Frank Piessens. Prospect: Provably secure speculation for the constant-time policy. In *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*, pages 7161–7178, 2023.

[5] Pasquale Davide Schiavone, Francesco Conti, Davide Rossi, Michael Gautschi, Antonio Pullini, Eric Flamand, and Luca Benini. Slow and steady wins the race? a comparison of ultra-low-power risc-v cores for internet-of-things applications. In *2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, pages 1–8, 2017.

[6] Jacob Fustos, Farzad Farshchi, and Heechul Yun. Spectreguard: An efficient data-centric defense mechanism against spectre attacks. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*, 2019.

[7] Lukas Gerlach, Daniel Weber, Ruiyi Zhang, and Michael Schwarz. A security RISC: Microarchitectural attacks on hardware RISC-V cpus. In *44th IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA, May 21-25, 2023*, pages 2321–2338, 2023.

[8] John L. Hennessy and David A. Patterson. *Computer Architecture, Fourth Edition: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., 2006.

[9] Jaewon Hur, Suhwan Song, Dongup Kwon, Eunjin Baek, Jangwoo Kim, and Byoungyoung Lee. Difuzzrtl: Differential fuzz testing to find CPU bugs. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*, pages 1286–1303, 2021.

[10] Jennifer Jackson, Minmin Jiang, and David Oswald. Cheri-crypt: Transparent memory encryption on capability architectures. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2025(2):268–292, 2025.

[11] Tobias Jauch, Alex Wezel, Mohammad R. Fadiheh, Philipp Schmitz, Sayak Ray, Jason M. Fung, Christopher W. Fletcher, Dominik Stoffel, and Wolfgang Kunz. Secure-by-construction design methodology for cpus: Implementing secure speculation on the rtl. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, 2023.

[12] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*, pages 1–19, 2019.

[13] lowRISC. Ibex risc-v core. https://github.com/lowRISC/ibex, 2017.

[14] lowRISC. Opentitan. https://opentitan.org/, 2019.

[15] P.S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner. Simics: A full system simulation platform. *Computer*, 35(2):50–58, 2002.

[16] Charles Papon. SpinalHDL: Scala based HDL. https://github.com/SpinalHDL/SpinalHDL, 2016.

[17] Charles Papon. VexRiscv: A FPGA friendly 32 bit RISC-V CPU implementation. https://github.com/SpinalHDL/VexRiscv, 2018.

[18] RISC-V Foundation. The risc-v instruction set manual, volume i: User-level isa, document version 20191214-draft, 2019.

[19] Thomas Van Strydonck, Job Noorman, Jennifer Jackson, Leonardo Alves Dias, Robin Vanderstraeten, David F. Oswald, Frank Piessens, and Dominique Devriese. Cheri-tree: Flexible enclaves on capability machines. In *8th IEEE European Symposium on Security and Privacy, EuroS&P 2023, Delft, Netherlands, July 3-7, 2023*, pages 1143–1159, 2023.

[20] The SHD Group. Risc-v market report: Application forecasts in a heterogeneous world. https://theshdgroup.com/wp-content/uploads/2024/01/RISC-V-Market-Analysis-2024-Abridged-Report-updated.b.pdf, 2024.

[21] Fabian Thomas, Lorenz Hetterich, Ruiyi Zhang, Daniel Weber, Lukas Gerlach, and Michael Schwarz. RISCVuzz: Discovering architectural CPU vulnerabilities via differential hardware fuzzing. https://ghostwriteattack.com/, 2024.

[22] Hans Winderix, Marton Bognar, Lesly-Ann Daniel, and Frank Piessens. Libra: Architectural support for principled, secure and efficient balanced execution on high-end processors. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024*, pages 19–33, 2024.

[23] Hans Winderix, Marton Bognar, Job Noorman, Lesly-Ann Daniel, and Frank Piessens. Architectural mimicry: Innovative instructions to efficiently address control-flow leakage in data-oblivious programs. In *IEEE Symposium on Security and Privacy, SP 2024, San Francisco, CA, USA, May 19-23, 2024*, pages 3697–3715, 2024.

[24] Hans Winderix, Jan Tobias Mühlberg, and Frank Piessens. Compiler-assisted hardening of embedded software against interrupt latency side-channel attacks. In *IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 667–682, 2021.

[25] Yinan Xu, Zihao Yu, Dan Tang, Guokai Chen, Lu Chen, Lingrui Gou, Yue Jin, Qianruo Li, Xin Li, Zuojun Li, Jiawei Lin, Tong Liu, Zhigang Liu, Jiazhan Tan, Huaqiang Wang, Huizhe Wang, Kaifan Wang, Chuanqi Zhang, Fawang Zhang, Linjuan Zhang, Zifei Zhang, Yangyang Zhao, Yaoyang Zhou, Yike Zhou, Jiangrui Zou, Ye Cai, Dandan Huan, Zusong Li, Jiye Zhao, Zihao Chen, Wei He, Qiyuan Quan, Xingwu Liu, Sa Wang, Kan Shi, Ninghui Sun, and Yungang Bao. Towards developing high performance RISC-V processors using agile methodology. In *55th IEEE/ACM International Symposium on Microarchitecture, MICRO 2022, Chicago, IL, USA, October 1-5, 2022*, pages 1178–1199, 2022.

[26] F. Zaruba and L. Benini. The cost of application-class processing: Energy and performance analysis of a linux-ready 1.7-ghz 64-bit risc-v core in 22-nm fdsoi technology. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 27(11):2629–2640, Nov 2019.

[27] Jerry Zhao, Ben Korpan, Abraham Gonzalez, and Krste Asanovic. SonicBOOM: The 3rd beneration berkeley out-of-order machine. In *Fourth Workshop on Computer Architecture Research with RISC-V*, volume 5, pages 1–7, 2020.

[28] Jean Karim Zinzindohoué, Karthikeyan Bhargavan, Jonathan Protzenko, and Benjamin Beurdouche. Hacl*: A verified modern cryptographic library. In *ACM Conference on Computer and Communications Security (CCS)*, pages 1789–1806, 2017.