Proteus: An Extensible RISC-V Core for Hardware Extensions

Marton Bognar[1]*, Job Noorman[1], Frank Piessens[1]

[1]imec-DistriNet, KU Leuven, 3001 Leuven, Belgium

Abstract

We present Proteus, an open-source RISC-V processor designed to allow rapid prototyping and evaluation of hardware extensions. This goal of making it easy to extend the processor's functionality and change its microarchitecture is enabled by making Proteus configurable in many parameters and using a plugin system for its functionality. Building on these plugins, Proteus features textbook implementations of an in-order and an out-of-order pipeline. We implement Proteus in SpinalHDL, which generates Verilog code that can run in a simulator or be synthesized to an FPGA, enabling rapid development and testing. This paper briefly introduces the processor's design and showcases hardware security extensions implemented on Proteus.

Introduction

RISC-V is an easily extensible instruction set architecture with openly available hardware designs. These properties make it a popular target for developing new hardware extensions, either in the microarchitecture or by extending the ISA and adding new instructions.

Our goal with creating and open-sourcing Proteus [1] is to ease this development process via a RISC-V processor focused on modularity and extensibility. Using Proteus simplifies implementing and evaluating hardware extensions and also enables a more straightforward comparison between them. The implementation is written in SpinalHDL [2], a hardware description language embedded in Scala, which speeds up the prototyping process by simplifying the code compared to lower-level implementations. SpinalHDL can generate Verilog and VHDL code, which can run in a simulator or be synthesized to an FPGA. Proteus also comes with a Newlib implementation [3], which enables the execution of existing C code, such as benchmark programs, on the processor after a recompilation step.

The processor features a textbook five-stage in-order pipeline and a dynamic out-of-order pipeline with multiple execution units, both implementing the RV32IM instruction set with CSR registers and corresponding instructions. Most of the functionality in these pipelines is implemented by (common) plugins, inspired by the plugin-based design of VexRiscv [4].

Description

In-order pipeline

The in-order pipeline consists of a sequence of stages, with plugins providing the functionality in each stage by adding logic between its pipeline registers. The number of stages is not fixed, and they can be reordered as long as their dependencies are satisfied. Proteus comes with a textbook five-stage implementation featuring *fetch*, *decode*, *execute*, *memory*, and *writeback* stages. For instance, the *fetch* stage is implemented by a single plugin (`Fetcher`), while the *execute* stage includes logic from multiple plugins, such as arithmetic shifting or address calculation. The 'M' (multiplication and division) and 'Zicsr' (control and status registers) RISC-V standard extensions are implemented as plugins that can optionally be included in the design – also in the out-of-order pipeline.

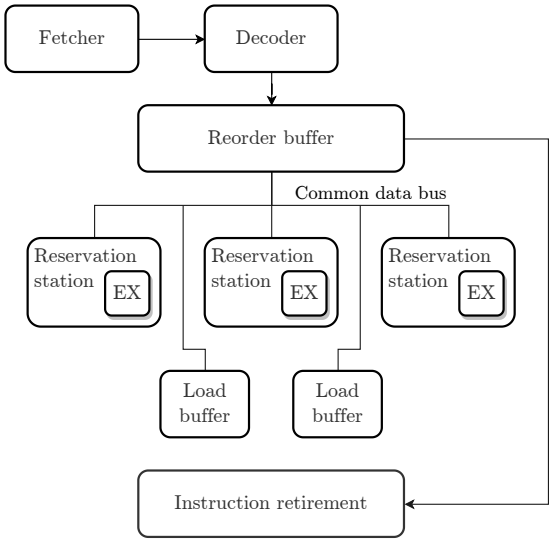

Figure 1: *Schematic of the out-of-order pipeline.*

Out-of-order pipeline

The out-of-order pipeline implementation is based on Tomasulo's algorithm as described by Hennessy and Patterson [5]. Figure 1 shows a schematic view of the

*Corresponding author: `marton.bognar@kuleuven.be`

design. After fetching and decoding, instructions are placed into the reorder buffer (ROB), which dispatches them to reservation stations. These reservation stations resolve instruction dependencies through register rewriting, execute instructions in an execution unit (EX), and broadcast this result on the common data bus. After a reservation station calculates the target address of a load operation, it forwards this to a load buffer, which connects to the memory bus. All reservation stations and load buffers can execute in parallel. When instructions are marked as ready in the ROB, they are retired in program order; register writes and memory stores are committed in this final retirement stage. The functionality of the first two pipeline stages, the execution units, the load buffers, and the retirement stage is provided by plugins that are also used in the in-order pipeline, showing the flexibility and reusability of the plugin architecture.

The out-of-order design is configurable by the number and type of execution units, load buffers, and ROB entries. It also features a branch target predictor with a configurable replacement policy. The parallelism and the branch predictor make this design more similar to higher-end microarchitectures, enabling the development of extensions for more complex systems.

Synthesis

Proteus uses an AXI4 memory bus, making it ideal both for simulation with software-defined memory and for synthesis to an FPGA with a connected memory module. FPGA synthesis also provides information about the impact of modifications and extensions on the hardware's area, power usage, and critical path. Table 1 summarizes these parameters when synthesizing the base in-order and out-of-order pipelines for an Artix-7 XC7A35TI board. The configuration for the latter contains 16 ROB entries, 5 execution units, and 3 load buffers. Proteus (including extensions) has been flashed to multiple physical FPGA boards: a Zynq UltraScale+ XCZU9EG and an Arty A7-35T XC7A35TICSG324-1L [6].

Design	LUTs	FFs	CP	Dynamic power
In-order	2,903	1,972	19.6 ns	15 mW
Out-of-order	16,775	11,913	29.5 ns	52 mW

Table 1: *Synthesis results for the two pipelines.*

Extensions

This section briefly introduces hardware extensions that have been implemented on Proteus. While these are all security extensions, we believe that the extensibility of Proteus is not limited to security.

CHERI

Capability Hardware Enhanced RISC Instructions [7] (CHERI) add support for hardware capabilities that enable fine-grained access control in the processor.

A version of the in-order pipeline has been extended to add support for most of version 8 of the 32-bit CHERI-RISC-V specification (without compressed capabilities), including storing capabilities in the register file and implicit memory access [6].

CHERI-TrEE

CHERI-TrEE [6] combines the CHERI capabilities with enclaved execution in one design. It implements an advanced form of enclaved execution by building on primitives provided by CHERI, extending the provided mechanisms where necessary.

ProSpeCT

ProSpeCT [8] is a defense against all known Spectre attack variants with a formally proven design. This prototype has been implemented on the out-of-order pipeline, as Spectre attacks generally require out-of-order and speculative execution. ProSpeCT requires no additions or changes to the ISA except adding new CSR registers to store the boundaries of memory marked as containing secret data. This secret data is tracked through the hardware, ensuring that it does not leak during speculative execution.

References

[1] Proteus developers. *Proteus: a configurable RISC-V core*. `https://github.com/proteus-core/proteus`.

[2] Charles Papon. *SpinalHDL: Scala based HDL*. `https://github.com/SpinalHDL/SpinalHDL`.

[3] Proteus developers. *Newlib: stdlib implementation for Proteus*. `https://github.com/proteus-core/newlib`.

[4] Charles Papon. *VexRiscv: A FPGA friendly 32 bit RISC-V CPU implementation*. `https://github.com/SpinalHDL/VexRiscv`.

[5] John L. Hennessy and David A. Patterson. *Computer Architecture, Fourth Edition: A Quantitative Approach*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2006. ISBN: 0123704901.

[6] Thomas Van Strydonck et al. "CHERI-TrEE: Flexible enclaves on capability machines". In: *EuroS&P - 8th IEEE European Symposium on Security and Privacy*. IEEE, 2023.

[7] Robert N.M. Watson et al. "CHERI: A Hybrid Capability-System Architecture for Scalable Software Compartmentalization". In: *2015 IEEE Symposium on Security and Privacy*. 2015, pp. 20–37. DOI: 10.1109/SP.2015.9.

[8] Lesly-Ann Daniel et al. "ProSpeCT: Provably Secure Speculation for the Constant-Time Policy". In: *32nd USENIX Security Symposium (USENIX Security 23)*. Aug. 2023.