Extended Abstract: A Flexible Simulation Environment for RISC-V

Karsten Emrich[1], Conrad Foik[1], Johannes Kappes[2], Sebastian Prebeck[2], Daniel Mueller-Gritschneder[1], Wolfgang Ecker[2], Ulf Schlichtmann[1]

[1]Technical University of Munich, Germany [2]Infineon Technologies
daniel.mueller@tum.de, wolfgang.ecker@infineon.com

Abstract

In this paper, we present a flexible simulation environment, well-suited for rapid prototyping of RISC-V embedded software. The environment supports both purely functional and performance simulations. It is based on an open-source instruction set simulator (ISS), called ETISS, which features a plugin mechanism to easily extend its functionality. This allows, for instance, to retrieve and modify the simulator state, enabling an early evaluation of the chosen processor architecture. Our approach utilizes this extensibility to automatically generate behavioral models for ETISS, based on abstract instruction set architecture (ISA) descriptions. This enables a quick adaptation of ETISS to an ISA variant, and eases the integration of our approach into existing design flows, e.g., to generate and validate custom RISC-V cores. We also outline how to extend ETISS to work as a performance estimator, using abstract descriptions of the targeted microarchitectures.

Introduction

One of the key advantages of the RISC-V instruction set architecture (ISA) is its high level of flexibility. Through standardized instruction sets and the ability to define custom instructions, the ISA is designed to be adaptable. In addition, due to its open-source character, vendors are free to implement their own microarchitectures. To fully utilize the opportunities offered by RISC-V's flexibility, designers typically rely on rapid prototyping of the processor, to enable early software development and design space exploration.

A common approach for early prototyping is the use of instruction set simulators (ISSs). ISSs simulate the functional behavior of a processor at ISA-level, and thus offer high simulation speeds, as required for efficient prototyping. Popular examples of ISSs for the RISC-V architecture are QEMU [1] and Spike [2].

Besides ISSs, performance simulators are frequently used during early prototyping, if the performance of the embedded software on the target processor needs to be analyzed. Examples of such simulators are TQSIM [3] and RISCV-VP [4], which extend the ISS concept by adding non-functional timing models of the targeted microarchitecture.

While there exist both ISSs and performance simulators for RISC-V, they usually target a specific variant and are not easily adaptable. As such, they do not support the flexibility offered by RISC-V.

In this paper, we present a flexible simulation environment well suited for rapid prototyping of RISC-V and validation of customized RISC-V hardware implementations, supporting both purely functional as well as performance simulations. Our approach is based on an ISS, called ETISS [5]. In the following, we will first present how to automatically generate behavioral models for ETISS from an abstract ISA description, thus enabling quick adaptation of the ISS to new ISA variants for functional simulations. Then we outline how to extend ETISS to work as a performance simulator. Similar to the first step, we use an abstract description of the microarchitecture to quickly adapt to different core variants.

Functional Simulation

Extending an ISS usually requires intricate knowledge of its internal interfaces and program structure. In ETISS, however, all major components apart from the main simulation loop are provided by plugins, to simplify the extension of the simulator. The most interesting plugin type for our use case is the `CPUArch` plugin, which specifies the entire behavioral model for a specific target architecture in ETISS.

While writing a plugin is easier than extending a monolithic codebase, it still requires knowledge of simulator-specific code layout and interfaces. With our tool M2-ISA-R, we designed a generalized metamodel to represent functional and structural components of a given architecture, as well as tools to parse an easy-to-use, consumer-agnostic description language and a generator for ETISS `CPUArch` plugins. M2-ISA-R is implemented fully in Python 3 and is split into frontends to consume models, the metamodel classes itself, and backends to generate models.

As the main description language we chose CoreDSL 2 [6] for its C-like instruction behavior modeling and ease of parsing and extensibility. The authors of CoreDSL 2 also provide ready-to-use descriptions of all basic RISC-V instruction sets.

This work was supported in part by the German Federal Ministry of Education and Research (BMBF) within the project Scale4Edge under contract nos. 16ME0127 and 16ME0132-40.

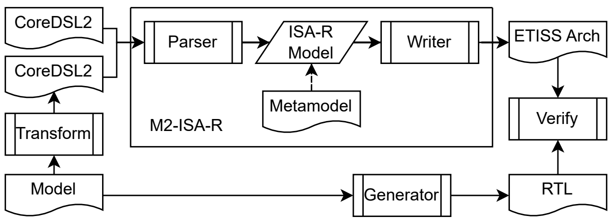

Figure 1: *M2-ISA-R tool flow*

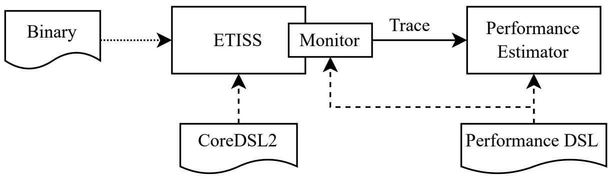

Figure 2: *Performance simulation setup for ETISS*

M2-ISA-R includes a frontend to parse CoreDSL 2 based on the ANTLR4 parser generator framework. The M2-ISA-R architecture description metamodel consists of a set of Python classes, roughly divided into architectural and behavioral descriptions of a target architecture. Finalizing the M2-ISA-R tool flow are backends to produce a model description. The most important backend is the ETISS architecture plugin writer, which closes the gap from CoreDSL 2 to a usable simulator.

Existing design tools can also be extended to directly output CoreDSL 2 descriptions. This approach facilitates broader possibilities of code reuse. Having a common source model is desirable for HW-SW-Codesign, because the functional model generated this way is then always coherent with a corresponding hardware design, enabling cross-verification applications.

A schematic representation of the entire flow is shown in Figure 1. The flow allows specifying ISA models in a consumer-agnostic way to facilitate reuse across tools while also abstracting away all simulator-specific details.

The focus of current research on M2-ISA-R and the CoreDSL 2 language is to generate also non-instruction behavior like trap entry, virtual memory management and semihosting directly from CoreDSL 2 descriptions.

Performance Estimation

Our approach extends ETISS to work as a performance simulator by using its plugin mechanism mentioned above. An instruction monitor is added to ETISS to generate a trace of the executed instructions. The trace is then passed to an estimator, which calculates the performance based on the trace and a non-functional timing model of the considered microarchitecture. Figure 2 illustrates this approach, omitting the generator flow presented above for readability.

To flexibly describe the timing model of the performance estimator, the CorePerfDSL language is introduced [7]. CorePerfDSL offers an isolated, non-functional description of a microarchitecture's timing behavior, including essential aspects, such as structural, data and control hazards. In addition, the description is highly flexible, allowing for the fast definition of multiple processor variants. CorePerfDSL also allows the specification of the required trace, which is used to specify the setup of the instruction monitor.

Table 1 shows the results for several performance simulations using the presented setup. To illustrate the flexibility of the approach 12 microarchitectural variants of an artificial, but characteristic, single-issue five-stage RISC-V processor have been considered. The performance estimates are given in cycles-per-instruction (CPI) and seem well-founded for the corresponding variants. The simulation speeds of 6-15 million instructions per second (MIPS) are reasonable for this kind of simulation.

Table 1: *Results from performance simulation*

Architecture	Data Forwarding	Branch prediction	Est. target perf. (CPI)	Simulation speed (MIPS)
Harvard	No	No	1.59	15.07
		Static	1.52	13.97
		Dynamic	1.48	12.27
	Yes	No	1.23	15.09
		Static	1.15	13.92
		Dynamic	1.11	12.72
von Neumann	No	No	1.68	6.67
		Static	1.60	6.44
		Dynamic	1.57	6.13
	Yes	No	1.36	6.75
		Static	1.30	6.42
		Dynamic	1.27	6.08

References

[1] Fabrice Bellard. "QEMU, a Fast and Portable Dynamic Translator". In: *USENIX Annual Technical Conference.* 2005, p. 41.

[2] *Spike RISC-V ISA Simulator.* https://github.com/riscv-software-src/riscv-isa-sim. Accessed: 2022-05-13.

[3] Shin-haeng Kang, Donghoon Yoo, and Soonhoi Ha. "TQSIM: A fast cycle-approximate processor simulator based on QEMU". In: *Journal of Systems Architecture* 66-67 (2016), pp. 33–47.

[4] Vladimir Herdt, Daniel Große, and Rolf Drechsler. "Fast and Accurate Performance Evaluation for RISC-V using Virtual Prototypes". In: *Design, Automation Test in Europe Conference and Exhibition (DATE).* 2020, pp. 618–621.

[5] Daniel Mueller-Gritschneder et al. "The Extendable Translating Instruction Set Simulator (ETISS) Interlinked with an MDA Framework for Fast RISC Prototyping". In: *International Symposium on Rapid System Prototyping (RSP).* 2017, pp. 79–84.

[6] MINRES Technologies. *CoreDSL 2.0.* https://www.minres.com/work/coredsl/. Accessed: 2023-03-15. 2022.

[7] Conrad Foik, Daniel Mueller-Gritschneder, and Ulf Schlichtmann. "CorePerfDSL: A Flexible Processor Description Language for Software Performance Simulation". In: *Forum on Specification & Design Languages (FDL).* 2022, pp. 1–8.