Surfer:
A Waveform Viewer as Dynamic as RISC-V

Lucas Klemmer[1], Frans Skarman[2], Oscar Gustafsson[2] and Daniel Große[1]

[1]Institut for Complex Systems, Johannes Kepler University Linz, Austria
[2]Department of Electrical Engineering, Linköping University, Sweden
lucas.klemmer@jku.at, frans.skarman@liu.se, oscar.gustafsson@liu.se, daniel.grosse@jku.at

Abstract

A growing ecosystem of available cores, ISA extensions, accelerators, and software creates new challenges for EDA tools. We believe that EDA tools have to be as dynamic and extensible as RISC-V itself to be able to optimally support developers handling RISC-V's dynamic ecosystem. In this paper, we present an extension to the Surfer waveform viewer. Surfer is explicitly developed with extensibility and customization in mind, which allows a range of new applications. We present Surfer's ability to utilize the Waveform Analysis Language *(WAL) to provide a CPU pipeline abstraction and other debugging information inside the waveform viewer.*

Introduction

Two of the most valuable aspects of RISC-V [1] are its open license, and its customizable and extendable nature. These two features have led to the explosive growth of a community that spans across commercial, academic, and even hobbyist groups. In just a few years, this community has created a large collection of available cores [2], ISA extensions [3], accelerators, and software. However, this fast pace also creates challenges, especially regarding the tools, on which further progress relies. Hardware tooling often still lacks behind its counterparts in the software domain. One example of this are waveform viewers, which are one of the most used tools of hardware designers and verification engineers alike. GTKWave [4], while being undoubtedly battle-proven, has changed little in the last few years. With rising design sizes and a growing number of highly custom designs, working in GTKWave can feel tedious. This is due to a lack of many quality-of-life features such as automatic detection and abstraction of buses, pipelines, a lack of integration with other tools, and a generally dated user interface.

In this paper, we extend Surfer, a new and open-source waveform viewer specifically designed with extensibility and customization in mind. Surfer has many features that make it easy to extend and integrate with other tools. As an example, we present that Surfer can connect to a waveform analysis program specified in the *Waveform Analysis Language* (WAL) [5]. The waveform analysis program provides an abstraction over the pipeline stages of a RISC-V processor [6]. Further, we are working on adding entirely new features such as connecting to simulators or waveform analyzers over network connections or deeply integrating Surfer into other applications via a remote control protocol. Surfer is available natively on all major platforms and the web through a version compiled to WebAssembly.

Surfer

Surfer [7] is an open source waveform viewer designed to be snappy and *extensible* to support new use cases. One example of this extensibility is the ease of adding translators which translate from the bit vectors present in a waveform to the actual values they represent. Adding a translator simply requires implementing two functions:

- `variable_info()` which tells Surfer what structure a translated signal will have
- `translate()` which performs the translation from bit vector to a more interesting value

In case of the RISC-V instruction translator, `variable_info()` tells Surfer that the structure of a translated value is a single field containing the disassembled instruction. `translate()` then takes each bit vector and runs it through a disassembler to get the human-readable RISC-V mnemonic.

For this work, Surfer being extensible to support many waveform sources is also important. In addition to just loading waves from waveform files like VCD and FST, Surfer can load waves from different programs via TCP. This allows it to interact with external programs that can dynamically generate waves such as an interactive simulator, or in this work, WAL.

Waveform Analysis based CPU Abstraction in Surfer

In this section, we present how programs in WAL can be used to define a powerful abstraction layer over RISC-V CPUs for use inside Surfer. The foundation of the proposed Surfer integration is the *Domain Specific Language* (DSL) for specifying pipelines of RISC-V cores, presented in [6]. This DSL allows creating an abstraction of the pipeline that tracks instructions flowing through the pipeline and allows easy injection of additional virtual signals [8] for debug and analysis.

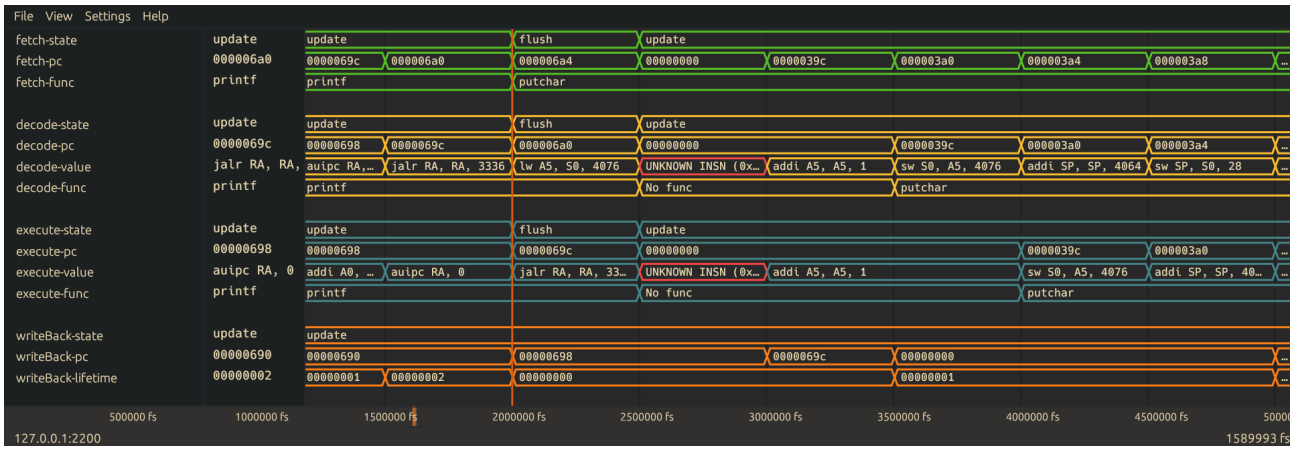

Figure 1: *Virtual Signals injected into a waveform of VexRiscv. The injected signals contain information about the pipeline stages, instruction lifetime, and even link instructions to their location in the binary.*

Figure 1 shows Surfer connected to a WAL program that analyzes the pipeline (four stages shown in green, yellow, blue, and orange) of a VexRiscv core. All displayed signals are "virtual" signals injected into the waveform by the WAL program. These signals are derived from other "real" signals or from external data, e.g., from debug information of an ELF file. Our example includes virtual signals that show the current *pipeline stage state*, the function to which the currently executed instruction belongs, a disassembled instruction, and the *lifetime* of an instruction, i.e., how long it took from entering the pipeline until completion.

```
(load "trace.fst")
(import rvnm)
(call rvnm.ranges "program.elf")
(defsig [decode-function 64]
  (call rvnm.find_func top.decode.pc))
(defsig [execute-function 64]
  (call rvnm.find_func top.execute.pc))
(server-start 1234)
(repl)
```

Listing 1: *Injecting new signals containing the function names to which the currently executing instructions belong.*

Listing 1 presents a simple WAL program that injects two new debugging signals into the loaded waveform. Two pipeline stages *decode* and *execute* both keep track of the *pc* of the currently contained instruction. For both stages, a new signal is injected that will contain the name of the function to which the instruction belongs. The new signals will compute their values by calling a small external Python function that analyzes the supplied ELF file using the *nm* command. Then, the program starts a TCP server that allows Surfer to connected to it and it opens a REPL, to keep the program alive while Surfer is still connected. Inside this REPL, new signals can be injected anytime and further waveform analysis is possible.

Outlook

We are working on integrating a remote control interface to Surfer that gives external tools the possibility to control all aspects of Surfer including adding and removing signals, setting the format and style of signals, adding markers, and zooming to specific time points. In addition, the protocol will also feature a set of drawing commands that will open up new possibilities for debugging, for example by highlighting assertion failures from simulators. Finally, we plan to integrate more waveform formats including high-level traces produced by RISC-V virtual prototypes at the TLM level.

Acknowledgements

This work has partially been supported by the LIT Secure and Correct Systems Lab funded by the State of Upper Austria.

References

[1] Andrew Waterman and Krste Asanović. *The RISC-V Instruction Set Manual; Volume I: Unprivileged ISA*. SiFive Inc. and CS Division, EECS Department, University of California, Berkeley. 2019.

[2] Alexander Dörflinger et al. "A comparative survey of open-source application-class RISC-V processor implementations". In: *CF'21*. Virtual Event, Italy, 2021, pp. 12–20.

[3] Enfang Cui, Tianzheng Li, and Qian Wei. "RISC-V Instruction Set Architecture Extensions: A Survey". In: *IEEE Access* 11 (2023), pp. 24696–24711.

[4] *GTKWave Waveform Viewer*. `https://github.com/gtkwave/gtkwave`.

[5] Lucas Klemmer and Daniel Große. "WAVING Goodbye to Manual Waveform Analysis in HDL Design with WAL". In: *IEEE Transactions on Computer Aided Design of Circuits and Systems* (2024). (accepted).

[6] Lucas Klemmer and Daniel Große. "A DSL for Visualizing Pipelines: A RISC-V Case Study". In: *RISC-V Summit Europe*. 2023.

[7] Frans Skarman et al. *Surfer 0.1.0*. 2024. DOI: `10.5281/ZENODO.10653540`.

[8] Lucas Klemmer and Daniel Große. "Towards a Highly Interactive Design-Debug-Verification Cycle". In: *ASP-DAC*. 2024, pp. 692–697.