Versatile Hardware Analysis Techniques

From Waveform-based Analysis to Formal Verification

Lucas Klemmer
Institute for Complex Systems
Johannes Kepler University Linz, Austria
lucas.klemmer@jku.at

I. Introduction

This thesis advances the field of hardware analysis with works ranging from performance analysis methods based on waveforms to formal verification. We provide various techniques related to design, debug, and verification of functional and non-functional properties, and present several works with both theoretical foundations and practical applications on real-world examples.

II. Waveform-based Analysis

In both, the design phase and the verification phase of a digital system, waveforms are heavily used. When the design matures, the verification plan is followed and advanced verification techniques, e.g., assertion-based methods together with coverage-based solutions, are employed [1], [2]. Along this highly iterative process, waveforms demonstrating expected behavior or unexpected behavior (e.g., in case of a failed assertion or a violated timing constraint) have to be analyzed and understood. For this task, waveform viewers are utilized. Waveform viewers are software tools which allow viewing signal values over time. Besides selecting the radix of each signal and grouping signals together, the user can zoom in and out, can jump to the next time point where the value of a signal changes, can determine the time difference between two cursors, etc. However, while all these features help in understanding and debugging, waveform viewing is a highly manual and tedious process.

So far, most research was either concentrated on specific design understanding approaches, for example to limit the manual analysis of waveforms to a minimum, or it has been confined to the generation of "better" waveforms, e.g., by employing formal methods, reducing their length, or minimizing the signals involved in a failing trace. While these approaches have introduced automation in general, there has been almost no progress for automating the analysis of waveforms.

A. Waveform Analysis Language

In [3]–[5] we presented WAL, our open-source *Waveform Analysis Language* that allows writing programs that walk over waveforms for performing analyses, and introduce the primary concepts behind the language. This most importantly includes notions of simulation time, signals, design hierarchy, and structural similarities as first class citizens of WAL. Even in this first publication, WAL is powerful enough to perform a wide range of tasks which were much more difficult to achieve before, such as complex bus analysis and visualization, as well as reconstruction of the control flow graph of software running on a RISC-V processor.

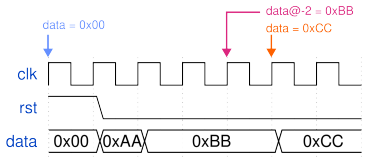

Fig. 1. WAL programming principle

```
1   (load "waveform.vcd")
2   (print data "␣" INDEX)        ;; prints 0 0
3   (step 10)
4   (print data "␣" INDEX)        ;; prints 33 10
5   (print data@-2 "␣" INDEX)     ;; prints 22 10
```

Listing 1. WAL program from Fig. 1

The idea behind WAL is shown in Fig. 1. After starting WAL and loading this waveform, the *INDEX* points to the start of the waveform (blue arrow). If we now evaluate the expression *data* we get the value *0x00*. Evaluating *(step 10)* moves the index forward by 10 timestamps (orange arrow). Note, that the index is not incremented for each rising edge of the *clk* signal, but whenever **any** signal is changed. Now, evaluating the same *data* expression results in the value *33*. Finally, we can move the index locally for just one expression using the *expression@offset* syntax. Using this syntax, the *expression* is evaluated at *INDEX* + *offset* and after the evaluation the index is restored to its previous value. Thus, evaluating *data@-2* at *INDEX* = *10* results in the value *22* (magenta arrow).

The steps presented above can be automated using the WAL code shown in Listing 1. First, the waveform is loaded on Line 1. Then, the value of the *data* signal is read and printed together with the current INDEX on Line 2. Next, the index is moved in Line 3, and the values of *data* and INDEX are printed again on Line 4. Finally, the value of *data* is printed again on Line 5, but this time the signal is read two timestamps before the current INDEX.

B. Processor Performance Analysis

The WAL programming language presented in [5] provides a new way to analyze hardware designs based on waveforms. However, WAL is only the programming language, and thus we needed to prove its effectivity on some real-world examples. In [6], we analyzed a wide range of RISC-V processors for important performance metrics such as instructions per cycle and and major pipeline behavior (e.g. stalls). Most importantly, our analysis is based on a very generic analysis

library which requires only a few lines of code that glue the library to a specific processor. In [7], we extended this approach by analyzing a commercial RISC-V processor leveraging the introduced analysis library. Additionally, we also determined the instruction runtime of single instructions for a wide variety of processor configurations with WAL. Finally, in [8], we present an extensible and flexible cache analysis framework.

C. Design Understanding and Debugging

In [9] we presented an online pipeline viewer which makes understanding complex processor pipelines much easier. By creating a custom pipeline DSL in WAL, the viewer can be easily adapted to various processors and architectures. We also use the viewer successfully in our computer architecture lecture.

D. HDL Integration

In [10], we collaborated with a team of researchers from Sweden who are developing Spade, a modern HDL. By integrating Spade and WAL, we were able to develop a system which allows bundling analysis code with Spade libraries. The main idea behind the integration is, that Spade data types can be annotated with analysis passes, which makes the Spade compiler emit the required analysis code. Now, users of Spade libraries can get valuable insights into their design without having to do any work themselves.

E. Design-Debug Workflow

Simulating larger hardware designs is a lengthy process that severely increases the time it takes to try and test changes. In [11], we presented a highly interactive hardware design-debug-verification workflow based on virtual signals, a methodology to inject new signals into existing waveforms. These virtual signals are WAL expressions, which, from the outside, look and behave exactly like regular signals. Using virtual signals, it is often possible to get extremely fast feedback on bug fixes without having to run the full simulation.

III. Analysis using Formal Methods

A. SUBLEQ Microcode Verification

SUBLEQ is a Turing-complete type of a one-instruction set computer that trades performance for implementation size. In [12], we proposed a virtual prototype of a microcoded RISC-V processor that uses the SUBLEQ instruction for the complete microcode. In [13], we presented a framework for formal microcode verification which we used to verify the microcode from [12]. Using the VP and the verification framework, we implemented and verified all instructions from RISC-V's RV32I base instruction set.

B. Generating Simulation Stimuli

In [14], we generated simulation stimuli by creating two distinct but semantically equivalent RISC-V programs. This is done with the help of a formal processor model, which is queried to find an equivalent program to another given program. The resulting two programs activate entirely different logic, but should produce the same result and end up in exactly the same state. This property allows using the two programs to uncover bugs, by running them on the same processor and looking for mismatches. This constitutes a novel verification approach for processors.

C. Gatelevel Netlist Optimization

Even with processors optimized for a low gate count, further optimizations are almost always possible. For example, in a specific application, a RISC-V processor might never execute some of the available instructions. In this case, it is possible to completely remove the gates that implement these instructions without compromising the correctness of the application. In [15], [16], we proposed an open-source gate-level optimization methodology to reduce netlists based on user-provided assumptions (in essence external don't cares), such as the restricted set of instructions. This approach leverages formal model checking to prove that some gates can be safely eliminated given the user-provided assumptions.

References

[1] H. D. Foster, A. C. Krolnik, and D. J. Lacey, *Assertion-based design*. Springer Science & Business Media, 2004.

[2] A. B. Mehta, *SystemVerilog Assertions and Functional Coverage*. Springer, 2019.

[3] **L. Klemmer** and D. Große, "WAL: a novel waveform analysis language for advanced design understanding and debugging," in *ASP-DAC*, 2022, pp. 358–364.

[4] D. Große and **L. Klemmer**, "Get the most out of your waveforms – from non-functional analysis to functional debug via programs on waveforms," in *Tutorial at Forum on specification & Design Languages*, 2023.

[5] **L. Klemmer** and D. Große, "WAVING goodbye to manual waveform analysis in HDL design with WAL," *IEEE Transactions on Computer Aided Design of Circuits and Systems*, 2024, (accepted).

[6] **L. Klemmer** and D. Große, "Waveform-based performance analysis of RISC-V processors: late breaking results," in *DAC*, 2022, pp. 1404–1405.

[7] **L. Klemmer**, E. Jentzsch, and D. Große, "Programmable analysis of RISC-V processor simulations using WAL," in *Design and Verification Conference and Exhibition Europe*, 2022.

[8] **L. Klemmer** and D. Große, "An extensible and flexible methodology for analyzing the cache performance of hardware designs," in *Forum on Specification and Design Languages*, 2024.

[9] **L. Klemmer** and D. Große, "A DSL for visualizing pipelines: A RISC-V case study," in *RISC-V Summit Europe*, 2023.

[10] F. Skarman, **L. Klemmer**, O. Gustafsson, and D. Große, "Enhancing compiler-driven HDL design with automatic waveform analysis," in *FDL*, 2023.

[11] **L. Klemmer** and D. Große, "Towards a highly interactive design-debug-verification cycle," in *ASP-DAC*, 2024.

[12] **L. Klemmer** and D. Große, "An exploration platform for microcoded RISC-V cores leveraging the one instruction set computer principle," in *ISVLSI*, 2022, pp. 38–43.

[13] **L. Klemmer**, S. Gurtner, and D. G. **(Best Paper Award)**, "Formal verification of SUBLEQ microcode implementing the RV32I ISA," in *FDL*, 2022, pp. 1–8.

[14] **L. Klemmer** and D. Große, "EPEX: processor verification by equivalent program execution," in *GLSVLSI*, 2021, pp. 33–38.

[15] **L. Klemmer**, D. Bonora, and D. Große, "Large-scale gatelevel optimization leveraging property checking," in *Design and Verification Conference and Exhibition Europe*, 2023.

[16] D. Große, **L. Klemmer**, and D. Bonora, "Using formal verification methods for optimization of circuits under external constraints," in *DATE*, 2024.