Formally Verifying Many RISC-V Implementations with One Page of Code

Steven Hoover
Redwood EDA
Shrewsbury, MA, USA
steve.hoover@redwoodeda.com

Ákos Hadnagy
Delft University of Technology
Delft, Netherlands
akos.hadnagy@gmail.com

ABSTRACT

WARP-V is a CPU generator written using Transaction-Level Verilog (TL-Verilog) that implements RISC-V (among other ISAs). WARP-V has been formally verified using riscv-formal, an open-source formal verification framework for RISC-V. Timing-abstraction and transaction-level design embodied in TL-Verilog are showing significant benefits for hardware modeling, but this work is the first demonstration of their benefits for verification modeling. The formal verification of all RISC-V configurations of WARP-V is accomplished in a single page of code.

KEYWORDS

formal verification, open source, RISC-V, hardware description language, digital logic, high-level modeling, transaction-level design.

1 BACKGROUND

WARP-V [1] is a RISC-V core written in Transaction-Level Verilog (TL-Verilog) [2][3]. The timing-abstract modeling of TL-Verilog enables WARP-V to achieve an unprecedented level of architectural flexibility and scalability. Readers unfamiliar with TL-Verilog's timing abstract and transaction-level design methodology are referred to [2] and [3], respectively. The TL-Verilog tool flow generates a variety of implementations from this flexible model.

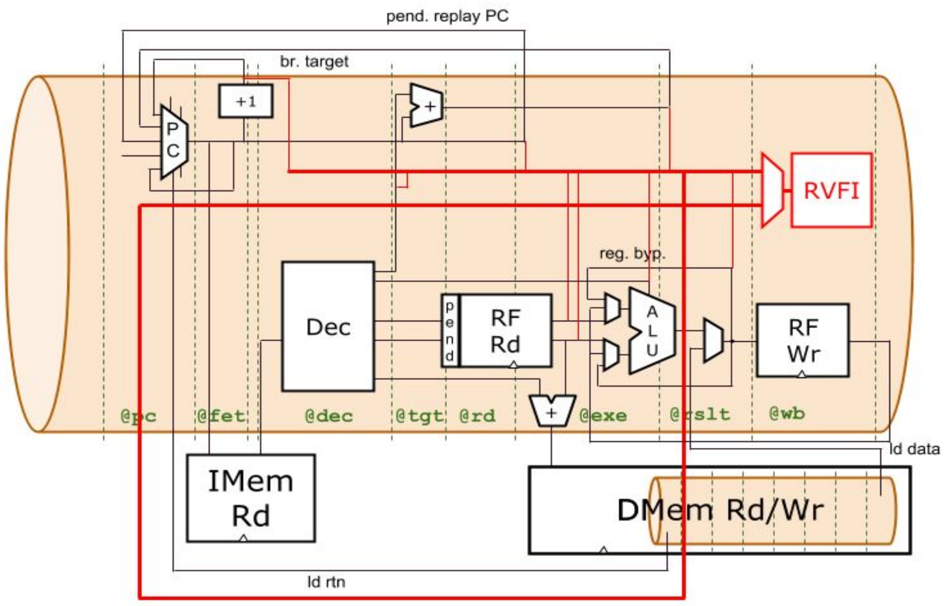

Fig. 1. WARP-V Microarchitecture with RVFI

Fig. 1 depicts the WARP-V microarchitecture, and, in red, the riscv-formal interface (RVFI) and the connections into it. Pipeline stages (labeled in green) in Fig. 1 are "virtual" stages. A particular configuration allocates virtual stages to the physical pipeline stages of the implemented microarchitecture. All virtual stages can be mapped to a single physical stage for a low-frequency microcontroller, or to seven physical stages for a mid-range high-frequency CPU. TL-Verilog tools generate the actual sequential logic implied from the staging.

The primary verification vehicle for WARP-V is an infrastructure called riscv-formal [4], an open-source Verilog model-checking framework for RISC-V. WARP-V was brought to life using a single 11-instruction test program. The remaining verification was entirely done formally using riscv-formal.

2 VERIFICATION

Traditionally, verification of a flexible CPU generator is challenging. A verification model must be generated that matches the flexibility of the hardware model.

To utilize riscv-formal, a test harness is required to present information about each retiring instruction to RVFI. This information must be presented to RVFI in the same clock cycle. Signals in the design that hold this information, however, are distributed across the CPU pipeline. So, the test harness must stage signals an appropriate number of cycles to present them in unison to RVFI.

Load instructions are particularly challenging. Before presenting the load to RVFI, all information characterizing the load must be available. This includes the load result from memory, which is available in WARP-V only after the load instruction has left the CPU pipeline. Therefore, in order to present the load instruction to RVFI, its information must be carried along with the load instruction as it goes to memory and returns into the CPU pipeline to write into the register file (depicted as the red loop through memory in Fig. 1.)

The staging and recirculation of signals are achieved by utilizing the transaction-level design features of TL-Verilog. RVFI logic is simply instantiated in the WARP-V model. As with TL-Verilog hardware logic in WARP-V, the riscv-formal checkers fit naturally into the CPU pipeline context and benefit from the transaction flow context that recirculates returning load data. These contexts provide automatic staging and recirculation of the signals needed as input to RVFI along the red path in Fig. 1.

The only modeling required to connect the hardware model with the verification model is the interface signal hookup and a bit of combinational logic to map available signals in the model to the expectations of RVFI. Also, a statement to select either the instruction from the CPU pipeline or the one recirculated with the load data is needed. This is the red multiplexer in Fig. 1.

3 RESULTS

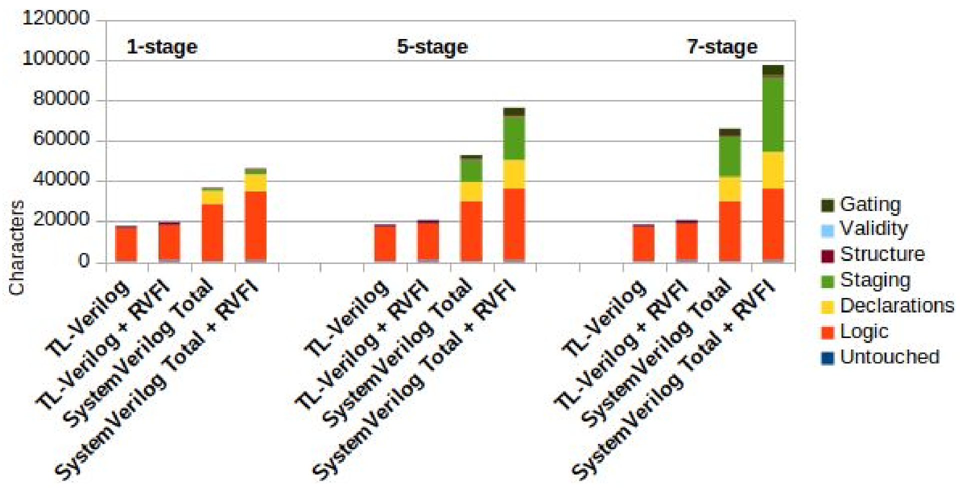

Fig. 2. Code-size comparison among various configurations

Fig 2. illustrates the benefits of transaction-level design in TL-Verilog versus RTL methodology in SystemVerilog by comparing TL-Verilog code size and generated SystemVerilog code size for various staging configurations with and without the verification harness. The bars labeled "TL-Verilog" are nearly equivalent code resulting from some preprocessing of the source code. The harness can be seen to be minimal in the TL-Verilog code, but it is much more significant in SystemVerilog. It grows with pipeline depth, even becoming more significant than the TL-Verilog source code.

4 SUMMARY

A single compact code base generates multiple different and larger SystemVerilog models with their formal verification harness.

REFERENCES

[1] https://github.com/stevehoover/warp-v

[2] S. F. Hoover, "Timing-Abstract Circuit Design in Transaction-Level Verilog," 2017 IEEE International Conference on Computer Design (ICCD), Boston, MA, 2017, pp. 525-532.

[3] S. F. Hoover and A. Salman. "Top-Down Transaction-Level Design with TL-Verilog." CoRR abs/1811.01780 (2018): n. pag.

[4] https://github.com/cliffordwolf/riscv-formal