RISC-V based Virtual Prototype: An Open Source Platform for Modeling and Verification*

Vladimir Herdt[1] Daniel Große[1,2] Hoang M. Le[1] Rolf Drechsler[1,2]
[1]Institute of Computer Science, University of Bremen, 28359 Bremen, Germany
[2]Cyber-Physical Systems, DFKI GmbH, 28359 Bremen, Germany
{vherdt,grosse,hle,drechsle}@informatik.uni-bremen.de
Find our github link on http://www.systemc-verification.org/riscv-vp

I. Introduction

RISC-V, being an open and free *Instruction Set Architecture* (ISA), is gaining huge popularity as processor ISA in Internet-of-Things (IoT) devices. We propose an open source RISC-V based *Virtual Prototype* (VP) demonstrator (available at http://www.systemc-verification.org/riscv-vp). Our VP is implemented in standard compliant SystemC using a generic bus system with TLM 2.0 communication. At the heart of our VP is a 32 bit RISC-V (RV32IMAC) *Instruction Set Simulator* (ISS) with support for compressed instructions. This enables our VP to emulate IoT devices that work with a small amount of memory and limited resources. Our VP can be used as platform for early SW development and verification, as well as other system-level use cases. We support the GCC toolchain, provide SW debug, coverage measurement capabilities and support the FreeRTOS and Zephyr operating systems. Our VP is designed as configurable and extensible platform. For example we provide the configuration for the RISC-V HiFive1 board from SiFive.

II. RISC-V VP Overview

A. Architecture Overview

Fig. 1 shows an overview of the general architecture of our VP. Besides the ISS, targeting the RISC-V RV32IMAC instruction set, we provide a TLM bus with an essential set of peripherals, including the RISC-V specific CLINT and PLIC peripherals. All components are attached to the the TLM bus at specific non-overlapping address ranges to enable routing of transactions. Timer interrupts are processed by the RISC-V specific *Core Local Interrupt Controller* (CLINT) and can be configured through memory mapped IO. Interrupts from other components are handled by the RISC-V specific *Platform Level Interrupt Controller* (PLIC). The PLIC collects and prioritizes interrupts and then routes them to the ISS. An ELF loader is provided to parse and load an executable RISC-V ELF file into the memory and setup the program counter in the ISS accordingly.

B. Performance Optimizations

We have implemented two optimization techniques in our VP that result in significant simulation speed-ups. The first optimization is a direct memory interface to fetch instructions and perform load/store operations from/to the (main) memory more efficiently. The second is a temporal decoupling technique with local time quantums to reduce the number of costly context switches, especially, in the ISS.

C. Extension and Configuration

It is very easy to add additional components (i.e. peripherals/controllers) and attach them to the bus system at a new address range, or change the address mapping of the existing components. This allows for an easy (re-)configuration of the VP. Support for additional RISC-V ISA extensions (beyond IMAC) can be added inside the ISS by extending the decode and execute functions accordingly. In general the compact implementation size makes the VP very manageable and thus suitable as foundation for different application areas.

* This work was supported in part by the German Federal Ministry of Education and Research (BMBF) within the project CONFIRM under contract no. 16ES0565 and by the University of Bremens Central Research Development Fund and by the University of Bremen's graduate school SyDe, funded by the German Excellence Initiative.

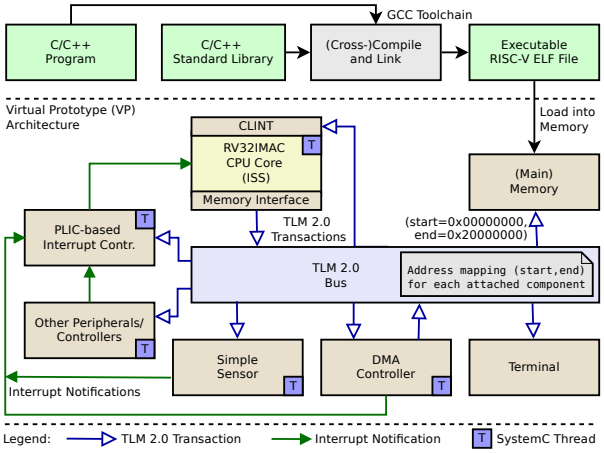

Fig. 1. RISC-V VP architecture overview

D. HiFive1 Configuration Example

As an example we provide the configuration matching the HiFive1 board from SiFive. The HiFive1 is based around a SoC that integrates a RISC-V RV32IMAC core with several peripherals and memories. Interrupts are processed by the CLINT and PLIC peripherals as specified by the RISC-V ISA. An SPI flash memory is provided to store the application code and a small writable memory (DTIM) to hold the application data.

E. Eclipse-based Application SW Debugging

Our VP provides SW debug capabilities in combination with the Eclipse IDE by implementing the GDB *Remote Serial Protocol* (RSP) interface. Debugging for example enables to step through the SW line by line, set (conditional) breakpoints, obtain and even modify variable values and also display the RISC-V disassembly (with the ability to step through the disassembly). Debugging can also be extremely helpful on the VP level to investigate errors, due to the deterministic and reproducible SW execution on the VP.

F. Timing Model

We provide a simple and configurable instruction-based timing model in the ISS and by following the TLM 2.0 communication standard, transactions can be annotated with optional timing informations to obtain a more accurate timing model of the executed software. We are working on integration of more precise timing models.

References

[1] V. Herdt, D. Große, H. M. Le, and R. Drechsler, "Verifying Instruction Set Simulators using Coverage-guided Fuzzing," in *DATE*, 2019.
[2] ——, "Extensible and Configurable RISC-V based Virtual Prototype," in *FDL*, 2018, pp. 5–16.
[3] ——, "Early Concolic Testing of Embedded Binaries with Virtual Prototypes: A RISC-V Case Study," in *DAC*, 2019.