MicroRV32 - A SpinalHDL based RV32I Implementation Suitable for FPGAs

Sallar Ahmadi-Pour[1] Vladimir Herdt[1,2] Rolf Drechsler[1,2]
[1]Group of Computer Architecture, University of Bremen, 28359 Bremen, Germany
[2]Cyber-Physical Systems, DFKI GmbH, 28359 Bremen, Germany
{sallar,vherdt,drechsler}@uni-bremen.de

Visit http://www.systemc-verification.org/risc-v to find our most recent RISC-V related approaches.

I. Introduction

We propose a demonstration of a lightweight RISC-V implementation called MicroRV32 that is suitable for FPGAs. The entire design flow is based on open source tools. The core itself is implemented in the modern Scala-based SpinalHDL hardware description language. For the FPGA flow, the IceStorm suite is utilized. On the iCE40 HX8K FPGA the design requires about 50% of the resources and can be run at a maximum clock frequency of 34.02 MHz. Beside the core, the design also includes basic peripherals and software examples. MicroRV32 is particularly suitable as a lightweight implementation for research and education. The complete design flow can be executed on a Linux system by means of open source tools which makes the platform very accessible.

II. MicroRV32 Overview

The demonstrator can be described in two parts: the system on a chip (SoC) platform, written in SpinalHDL, and the hardware setup using the platform on a FPGA development board with additional probing and stimuli circuits. Developing the SoC platform was aligned to the specifications our open source RISC-V *Virtual Prototype* (VP) (available at https://github.com/agra-uni-bremen/riscv-vp). The platform was tested with the RISC-V Unit Tests and comes with software examples. By utilizing the open source IceStorm suite for iCE40 FPGAs and the modern open source SpinalHDL hardware description language the platform is very accessible and can be used with a Linux system.

A. Architecture Overview

Fig. 1 shows a top level view of the architecture of the MicroRV32 platform. The architecture consists of the RISC-V RV32 core and peripherals interconnected through a memory mapped bus. A memory holds the executed program and acts as random access memory (RAM) for the processor. The initial program is loaded into the memory at synthesis time. For additional peripherals a module representing LEDs and a module to shutdown the platform were added. The shutdown peripheral is used to terminate the processor by transitioning it into a defined end state. To slow down the execution of the processor a clock divider is provided. The platform outputs the internal state of the processor in order to make the execution cycles visible and traceable.

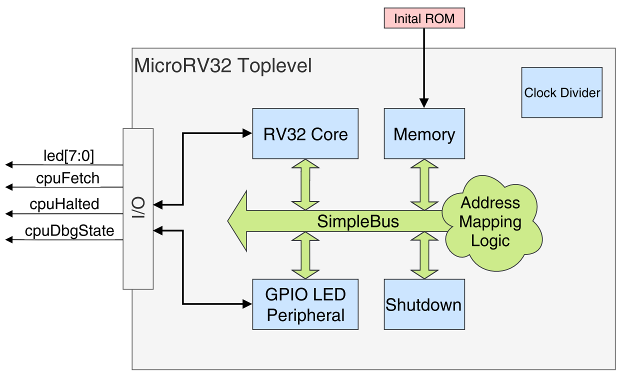

Fig. 1. Architecture overview

This work was supported in part by the German Federal Ministry of Education and Research (BMBF) within the project Scale4Edge under contract no. 16ME0127 and within the project VerSys under contract no. 01IW19001.

B. Demonstrator Overview

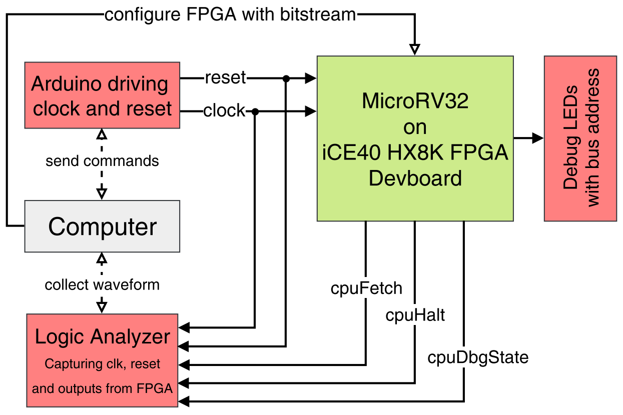
configure FPGA with bitstream
Arduino driving clock and reset
reset
clock
MicroRV32 on iCE40 HX8K FPGA Devboard
Debug LEDs with bus address
send commands
Computer
collect waveform
Logic Analyzer
Capturing clk, reset and outputs from FPGA
cpuFetch
cpuHalt
cpuDbgState

Fig. 2. Demonstrator overview

Fig. 2 shows the hardware setup of the demonstrator for the MicroRV32 platform. The SoC platform runs on a iCE40 HX8K FPGA development board (center, green) and is connected to components to control the FPGA inputs (here clock and reset) and to show and trace the outputs (SoC outputs). With an Arduino microcontroller (red, top left) the FPGA inputs are driven. A logic analyzer (red, bottom left) collects the traces of inputs and outputs as a waveform. Additional LEDs (red, top right) are used to show the the bus address of the memory mapped bus. Through serial commands from a computer the microcontroller drives the *reset* and *clock* lines of the FPGA.

On the iCE40 HX8K FPGA from Lattice Semiconductor the MicroRV32 Platoform achieves a maximum frequency of 34.02 MHz at a device utilization around 50%.

C. Future Work

We plan to investigate different directions to further improve our platform and the VP-based integration:

- Consider formal methods and comprehensive simulation-based techniques to validate the platform and in particular the RISC-V core.
- Investigate cross-level methodologies between the VP and RTL descriptions for verification, simulation and modeling purposes.
- Integrate further peripherals and RISC-V instruction set extensions into the platform to provide support for powerful operating systems.

References

[1] V. Herdt, D. Große, P. Pieper, and R. Drechsler, "RISC-V based virtual prototype: An extensible and configurable platform for the system-level," *JSA*, 2020.
[2] S. Tempel, V. Herdt, and R. Drechsler, "An effective methodology for integrating concolic testing with SystemC-based virtual prototypes," in *DATE*, 2021.
[3] V. Herdt, D. Große, E. Jentzsch, and R. Drechsler, "Efficient cross-level testing for processor verification: A RISC-V case-study," in *FDL*, 2020.
[4] V. Herdt, D. Große, and R. Drechsler, *Enhanced Virtual Prototyping: Featuring RISC-V Case Studies*. Springer, 2020.