Chipyard: A RISC-V Development Framework

Alexander Lukens Karl Hallsby Faculty Advisor: Dr. Jia Wang

April 9, 2021

Alex Lukens and Karl Hallsby are undergraduate students researching Chipyard [1] and RISC-V. RISC-V is a open-source Reduced Instruction Set Computing (RISC) instruction set architecture. It is sponsored by many companies, and is under heavy research at University of California at Berkeley. It was specifically designed to have a minimal core instruction set, that allows for string consutomization. This allows RISC-V to scale as needed in appropriate situations. Chipyard removes some of the manual aspects of designing a RISC-V CPU design, and instead moves the chip designer's responsibilities to determining the feature-set desired. This promises to make development of these designs both quicker and more correct. These promises also come at the price of Chipyard being under *very* active development. The API of Chipyard is not considered stable, and some features for working with FPGAs is not yet implemented.

RISC-V is a newer revision of older versions of earlier versions of the RISC instruction set[1]. This revision defines a very small core set of integer-based instructions allowing for a very basic CPU to be constructed. RISC-V also defines a variety of standardized extensions, enabling hardware support for floating-point arithmetic, atomic instructions, SIMD[2] instructions, and mixed-multiply-divide instructions.

Chipyard is a framework for designing, building, testing, and refining RISC-V CPU designs. It uses a combination of Verilog and Scala to provide its utilities. The hardware is described using Verilog, and Scala is used to parameterize and automate the generation of these Verilog modules to build full-fledged processors. To aid in testing, Chipyard also provided utilities to generate FPGA bitstreams, `mcs` files, among other data formats.

In this project we went in a variety of directions. We started by building some of the already-defined CPU designs and creating custom CPU designs using Scala. Then, we used the tools provided in Chipyard to test these designs against the RISC-V ISA standard, to ensure the generated designs were compliant with the RISC-V standard. We moved onto testing the standard processor designs using an FPGA both as an accelerator, and as a general-purpose testing platform. Lastly, we also uploaded a program written in a high-level language (C) to the FPGA, and ran it on the generated RISC-V CPU.

There are a variety of next steps that this work can go in. The first is a re-design of the computer architecture classes here at IIT. Because RISC-V is open-source and is relatively license-free, there are few legal concerns when it comes to CPU design and building. This open nature is quite useful for tracking changes made to underlying CPU designs too. Because RISC-V is currently being used in industry, there is strong documentation available for use. Lastly, other universities already use Chipyard and RISC-V for computer architecture courses, meaning materials are already available.

From a research perspective, this project does require additional work to integrate the Arty FPGA with the Chipyard framework. Another possible direction to take this is to design a GPIO device that allows for a single-step bus cycle, similar to the SANPER-1. Lastly, a feature we did not have any time to investigate was the VLSI-generation workflow.

Overall, Chipyard provides many tools that greatly enhance and simplify the development of larger RISC-V CPU designs. In addition, to make subsequent development easier, we are currently preparing a documentation manual that covers everything required to begin building and using Chipyard. There do exist similar documentation materials on the Internet, but they are scattered and sometimes contradictory. Preparing this documentation manual will allow us to curate and present the information required to set everything required up, and then move forward with the discussion.

[1] Not to be confused with the acronym of RISC (Reduced Instruction Set Computing)
[2] Single Instruction Multiple Data