Chipyard SoC Design Framework

Alexander Lukens

Karl Hallsby

Illinois Institute of Technology

Last Edited: March 23, 2021

Contents

Chapter 1

Setup

1.1 Introduction

This document is intended to serve as a record of the work performed for the ECE 497 special project supervised by Professor Jia Wang during the Spring 2021 semester. In this document, we will specify how our project repository was created, outline issues we ran into, and provide guidance on how to better setup the Chipyard Framework.

1.2 Obtaining Dependencies

1.2.1 Project Environment

The first obstacle in utilizing the Chipyard Framework is creating a project environment and obtaining all of the Chipyard dependencies. In my case, I decided to use Ubuntu 20.04 LTS running in a Virtualbox virtual machine with 4 cores, 8GB of ram, and a 250GB disk image. Much of this space will be utilized, as the entire RISC-V toolchain and Xilinx Vivado suite require a large amount of disk space.

1.2.2 Chipyard Dependencies

To gather the Chipyard dependencies, I followed the Chipyard documentation closely. Specifically, the documentation section 1.4 outlines how to prepare the Chipyard framework for use under Ubuntu. At this point (and before cloning the repositories) you should create a specific project folder somewhere in your home directory where all of the Chipyard files will live.

To alleviate any issues that may occur due to misconfigured environment variables, it is a smart idea to add the line `source /path/to/chipyard/env.sh` to your `.bashrc` file in your home directory. After adding this to your `.bashrc` file, reboot and continue.

1.2.3 Xilinx Vivado Suite Installation

The Xilinx Vivado Suite is important to be installed if any work regarding an FPGA is to be conducted. In my case, I downloaded the "offline installation" version of the Xilinx Unified Installer (version 2020.2) so that the actually installation process will complete faster. When conducting the installation, be sure to select "Vitis" instead of just selecting "Vivado". Installing Vitis will install the complete Xilinx package, including Vivado, and is useful for implementing FPGA projects.

1.3   Other Useful Projects

1.3.1   Freedom E SDK

This repository is maintained by SiFive, and provides several useful tools for designing, uploading, and debugging software to FPGA devices. This repository is specifically meant for use with SiFive IP, but can still be utilized for Chipyard projects with some modification.

1.3.2   Freedom Tools

This repository is maintained by SiFive, and is used to generate several tools that will be used during this project, such as the GCC cross-compiler for RISC-V (and many extension sets of RISC-V), OpenOCD, which assists users in debugging their FPGA designs, RISC-V QEMU, and other useful software. These tools take a considerable amount of time and disk space to compile so it is best to run the make command with `-j`nproc`` to parallelize compiling

Chapter 2

Repository Deep Dive

2.1 Verilator Simulator

2.1.1 About

The primary way to simulate SoCs' designed using the Chipyard framework is via Verilator simulations the directory for verilator is `/path/to/chipyard/sims/verilator`. A base/example simulation can be run by using `make` in the verilator directory. Custom Chipyard configs can be simulated by running `make CONFIG=*your custom config*`. Running the make command produces a simulator executable in the verilator directory. For example, if your project name was "TestConfig", running `make CONFIG=TestConfig` would create an executable called simulator-chipyard-TestConfig in the verilator directory. Custom RISCV code can be run by using the command `./simulator-chipyard-TestConfig /path/to/riscv/executable`.

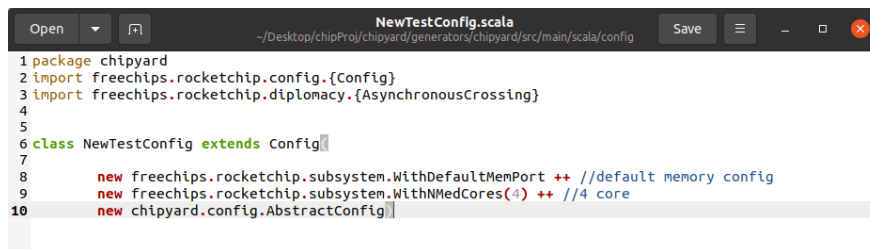### 2.1.2 Generators

Chipyard Generator

SHA3 Accelerators

2.1.3 Custom Configurations

Custom Configs can be created in the directory `chipyard/generators/chipyard/src/main/scala/config/`. For example, I created a new scala file called `NewTestConfig.scala` in the directory, allowing me to create a simulator from a class inside the NewTestConfig.scala file. Example Configs can be found in `RocketConfigs.scala` in the same directory.

2.1.4 FPGA Implementation

About


```scala
1 package chipyard
2 import freechips.rocketchip.config.{Config}
3 import freechips.rocketchip.diplomacy.{AsynchronousCrossing}
4
5
6 class NewTestConfig extends Config(
7
8     new freechips.rocketchip.subsystem.WithDefaultMemPort ++ //default memory config
9     new freechips.rocketchip.subsystem.WithNMedCores(4) ++ //4 core
10    new chipyard.config.AbstractConfig)
```

Figure 2.1: NewTestConfig.scala