FireSim/FPGA - Progress Report

Milan Lagae

Institution/University Name: Vrije Universiteit Brussel
Faculty: Sciences and Bioengineering Sciences
Course: Master Thesis

Contents

1 Introduction

The list of resources can be found in the section §2. First, the attempt at using the FireSim tool in section §4.1 and the following section on executing manual commands with Verilator, in section §4.2. The TLDR can be found in section §3.

2 Resources

The following list of resources where used during the course of this research, the most important are listed below:

- https://fires.im/asplos-2023-tutorial/
  - Very good collection of slides (+ videos)!
- https://docs.fires.im/en/latest/Advanced-Usage/Debugging-in-Software/RTL-Simulation.html
- https://chipyard.readthedocs.io/en/latest/Simulation/Software-RTL-Simulation.html
- https://chipyard.readthedocs.io/en/stable/Chipyard-Basics/Configs-Parameters-Mixins.html

3 TLDR

3.1 Firesim

Config execution using firesim manager:

- (re)started from example config files
- issue with sudo password & parallelism (Fabric package)
  - fixed by removing parallel annotation & setting password through ENV
- infrasetup command works for default midasexamples_gcd
- executing error when executing custom config: "Scala: File Not Found Exception"

3.2 Verilator

Manual command execution using make & verilator:

- running starter config
- compiling examples binaries in test directory
- running simple example binaries using Verilator.

/

4 Metasimulation

This section describes on how to get metasimulation working on FireSim on a (non-AWS) local host in using the open source software simulator Verilator.

4.1 FireSim & Verilator

4.1.1 Installation

Installation can be done by following the tutorials on the Firesim & Chipyard documentation website, installation will take some time and the host platform is required to be a Linux x86 machine for easy of support, see website for more info.

4.1.2 Sourcing

After the installation is complete, the following commands have to be executed to source the required binaries and set the proper environment variables.

Start of with sourcing the conda environment for the correct shell, shown in Listing 1.

```
eval "$(/home/firesim/setup/miniforge3/bin/conda shell.bash hook)"
```

Listing 1: Extending class.

After the shell is sourced, execute the command shown in Listing 2.

```
conda activate base
```

Listing 2: Extending class.

Proceed to change directory to the following directory: chipyard/sims/firesim and source the following two files:

1. env.sh
2. sourceme-manager.sh

Sourcing the first file shouldn't give an error, but the second one will if you are running on a bare metal non-AWS f1 instance. The error message can be ignored (for now), not relevant to our configuration.

To check if the sourcing has been successful, the which command can be used to check if the firesim binary is available: which firesim.

4.1.3 Config

Firesim works based on the use of *.yaml configuration scripts. For the metasimulation use case the configuration files that are important are the following two:

- config_runtime.yaml
- config_build_recipes.yaml

These files can be copied from the sample-backup-config directory in the chipyard repo. The config_runtime.yaml file should be modified with the following values as shown in Listing 8. Relevant sections are displayed, remaining values are the default values indicated in the documentation.

The metasimulation will be done on the same hosts we are managing firesim from. This configuration for the metasimulation slots is specified in the config_runtime.yaml file.

The config_build_recipes.yaml can stay the same as the default value with the midasexamples_gcd recipe specified and referenced in the config_build_recipes.yaml in Listing 3.

```
default_hw_config: midasexamples_gcd
```

Listing 3: Config parameter value.

4.1.3.1 Digression

With the above files configured, and using the default configuration, the firesim infrasetup can be executed. Executing the command in the current configuration will display the error message shown in Listing 4.

```
(...)
2025-11-12 20:10:30,529 [flush ] [INFO ] Fatal error: Needed to prompt for a connection or sudo password (host: 192.168.17.133), but input would be ambiguous in parallel mode 2025-11-12 20:10:30,529 [flush ] [INFO ] Aborting. 2025-11-12 20:10:30,544 [flush ] [INFO ] Fatal error: One or more hosts failed while executing task 'instance_liveness'
(...)
```

Listing 4: Extending class.

This error message has been tracked back to the Fabric Python library that Firesim uses to run functions in parallel on hosts.

This should be revisited if time allows or a better solution is found. For now the solutions is done by modifying the Python run file as follows.

From the method: instance_liveness remove the annotation @parallel, in the file: firesim_topology_with_passes in the directory: runtools and add the following code shown in Listing 5 in the method. The code looks for an environment variable called: FIRESIM_PWD, which must be specified for each command that is run, and than set, so the ssh Fabric package does not complain about (sudo/parallelism) requiring a password.

```
1 # Set the password from retrieving a env variable.
2   env.password = os.environ.get("FIRESIM_PWD")
3   # Local setup, security not a consideration
4   rootLogger.info("""Password: {}""".format(env.password))
```

Listing 5: Extending class.

NOTE: This is not secure, but since this is a local setup, that is not a consideration. Following firesim commands can than be executed as follows:

```
FIRESIM_PWD=<user-pwd> firesim <command>
```

Listing 6: Extending class.

4.1.4 Setup & Workload

After everything is setup, the firesim infrasetup command can be run, with the first example gcd there should be no errors.

In case of errors: a potential issue might be a missing directory named: sim_slot_0, in the home directory of the user used to execute the command.

Otherwise the connection is successful. Now we modify the config to execute a named config from the config directory. Change the config_runtime.yaml file and modify the default_hw_config parameter to specify a configuration in the config_build_recipes.yaml.

If that is done, the infrasetup command can be executed again.

FROM this point on, file not found exception

Continuing from this point, running the config, on my end I got the file not found exception. No solution has been found at the moment of writing this.

4.2 Chipyard & Verilator

The subsection describes on how to execute Chipyard core configuration using the Verilator software simulator, by executing manual make commands.

The tutorial-starter-config specified in Listing 7 and used earlier will be re-used. Proceed to the sims/verilator directory in the chipyard repository. Executing the following command to compile your config: make CONFIG=<config-name>. This command should execute without a problem.

```
1 // Tutorial Phase 1: Configure the cores, caches
2 class TutorialStarterConfig extends Config(
3  // CUSTOMIZE THE CORE
4  new freechips.rocketchip.rocket.WithNHugeCores(4)
5  new freechips.rocketchip.subsystem.WithNBanks(4) ++
6  new chipyard.config.AbstractConfig
7 )
```

Listing 7: Modified Tutorial Starter Config

After the config has been successfully compiled, the binary that is to be executed must be compiled. Proceed to the test/ directory inside the Chipyard repository.

There are different binaries that can be executed, for this use case, the mt-hello.o was chosen. For the list of options, the README.md file inside of the test/ directory describes on how to compile specific binaries.

Having compiled the mt-hello.o file to a mt-hello.riscv binary, it can executed on the core configuration. With the binary compiled execute the following command: make CONFIG=<config-name> BINARY=<path-to-binary> run-binary-hex

The last run-binary-hex specifies to run the binary using the fastmem option, more info is available in the documentary. Specifying the fastmem option, noticeably increases the speed of the simulation. When executing the mt-hello.riscv binary, the output will look like something in Figure 1.

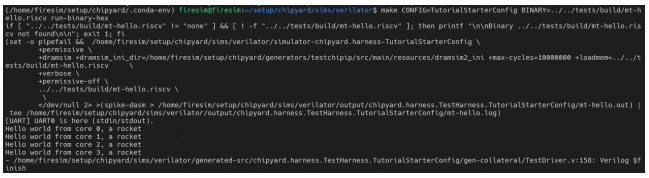

Figure 1: Executing mt-hello.riscv binary, with fast memory

4.3 Appendix

```
1
2  run_farm:
3    base_recipe: run-farm-recipes/externally_provisioned.yaml
4    recipe_arg_overrides:
5      default_platform: EC2InstanceDeployManager
6      default_simulation_dir: /home/firesim
7      default_fpga_db: /opt/firesim-db.json
8
9      run_farm_hosts_to_use:
10       - localhost: four_metasims_spec
11
12     run_farm_host_specs:
13       - four_metasims_spec:
14           num_fpgas: 0
15           num_metasims: 4
16           use_for_switch_only: false
17
18
19 metasimulation:
20     metasimulation_enabled: true
21     # vcs or verilator. use vcs-debug or verilator-debug for waveform generation
22     metasimulation_host_simulator: verilator
23     # plusargs passed to the simulator for all metasimulations
24     metasimulation_only_plusargs: "+fesvr-step-size=128 +max-cycles=100000000"
25     # plusargs passed to the simulator ONLY FOR vcs metasimulations
26     metasimulation_only_vcs_plusargs: "+vcs+initreg+0 +vcs+initmem+0"
27
28 # DOCREF START: target_config area
29 target_config:
30     topology: no_net_config
31     no_net_num_nodes: 1
32     link_latency: 6405
33     switching_latency: 10
34     net_bandwidth: 200
35     profile_interval: -1
36
37     default_hw_config: midasexamples_gcd
38     plusarg_passthrough: ""
39
40 (...)
```

Listing 8: Config Runtime File