

# FireSim/FPGA - Progress Report

Milan Lagae

Institution/University Name:  
Faculty:  
Course:

Vrije Universiteit Brussel  
Sciences and Bioengineering Sciences  
Master Thesis

## Contents

1	Introduction .....	1
2	Resources .....	1
3	TLDR .....	1
3.1	Firesim .....	1
3.2	Verilator .....	1
4	Metasimulation .....	2
4.1	FireSim & Verilator .....	2
4.1.1	Installation .....	2
4.1.2	Sourcing .....	2
4.1.3	Config .....	2
4.1.3.1	Digression .....	2
4.1.4	Setup & Workload .....	2
4.2	Chipyard & Verilator .....	3
5	Custom Tile/Cores .....	3
6	Running Proper Workloads .....	3
6.1	Appendix .....	4

## 1 Introduction

The list of resources can be found in the section §2. First trying to run using the FireSim tool in section §4.1 and the last section on executing manual commands 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 & parrallelism (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 is about getting metasimulation working in FireSim using the open source simulator Verilator.

### 4.1 FireSim & Verilator

#### 4.1.1 Installation

Installing 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.

#### 4.1.2 Sourcing

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

First 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. source-manager.sh

Sourcing the first file shouldn't give an error, but the second one will if you are running on a baremetal 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 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.

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 set, 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 and local setup.

This should be revisited if time allows or a better solution is found. For now the solution 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 and directory: runtools and add the following code 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 then 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(f"Password: {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 then 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.

If that is the case 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

Continuining 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

Since FireSim makes use of Verilator under the hood, I decided to perform manual command executions.

We will make use of the config we specified earlier. 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: Tutorial Starter Config

After this we can proceed to run a binary, before we can run an RISC-V binary, we must first compile a binary to run. Go into the `test/` directory inside the Chipyard repository. You can decide which binary you want to run, to compile assembly code, read the `README.md` file inside of the directory, it describes on how to compile all/or a specific binary. I will assume you compiled the `mt-hello.o` file to a `mt-hello.riscv` binary.

With the binary compiled proceed to 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.

[illegible]

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

## 5 Custom Tile/Cores

In progress

## 6 Running Proper Workloads

In progress

## 6.1 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 # DOCTREF 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