FireSim

Using On-Premises FPGAs and Distributed Metasimulation

https://fires.im

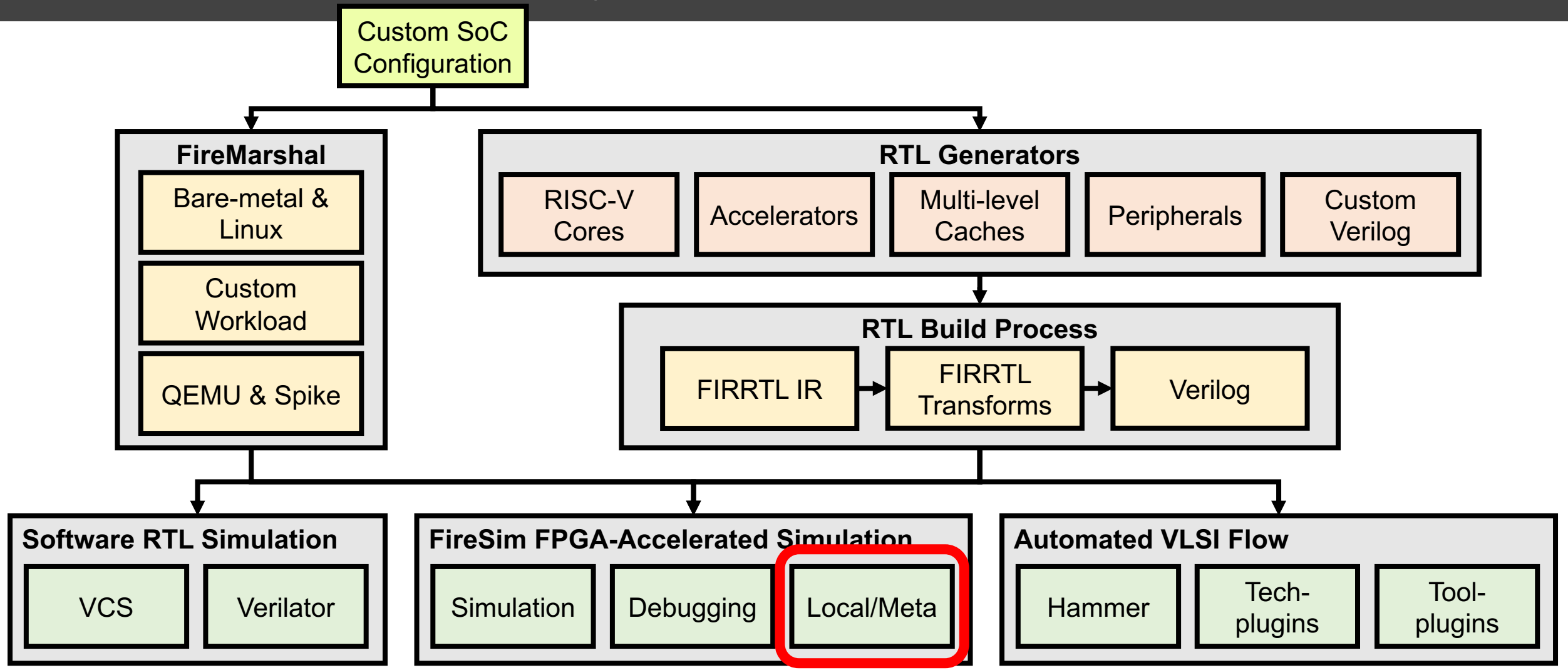
@firesimproject

ISCA 2023 Tutorial

Speaker: Sagar Karandikar

Berkeley Architecture Research

Tutorial Roadmap



Agenda

- Using On-Premises FPGAs
  - Case Study: How to build/run simulations locally with your own FPGA?
- Build Farms, Run Farms, Bit Builders, Deploy Managers Deep Dive
  - What are they and how do they configure the manager?
- Distributed Metasimulation
  - Scale-out software simulation

Some of our most requested questions…

Is anyone looking into adding support for other FPGA targets in addition to AWS F1?

I'd assume the U200 should be relatively straightforward since it's very close to the fpga that AWS uses.

hello all,

We are modifying firesm for our purpose. This involves new widget development (chisel + driver), integration in firesim, simulations (target + midas levels) and than running on f1. However, this has the potential of excessive AWS usage for simulations, buildafis, debugging and bug fixes etc. resulting in significant increase in AWS cost.

I was wondering if there may by some alternative (more economical) approach to this, like doing most of the development locally and than using AWS when we want to deploy at scale for large designs / configurations. While going through Midas / firesim code, I saw numerous mentions of Zynq so got curious about this approach.

Folks:
Has anyone tried to do bitstream builds using a local Vivado build, instead of on a build farm on AWS? Is there a Dockerfile or something like that that can create the build farm image so I can duplicate that on our internal servers?

Hi Guys,

I've been playing around with the awesome FireSim repo, and in particular trying to build and launch experiments for an on-premises alveo card using Vitis.

Hi,

I've read several threads of people interested in using FireSim on local VCU118, XC706 or various Alveo cards but never saw that such support had been implemented. What is the latest status of FireSim for local local Xilinx FPGA boards with XDMA PCIe support? Are there any examples to study?

Some of our most requested questions…

Is anyone looking into adding support for other FPGA targets in addition to AWS F1?

I'd assume the U200 should be relatively straightforward since it's very close to the fpga that AWS uses.

hello all,

We are modifying firesm for our purpose.
running on f1. However, this has the pote
cost.

I was wondering if there may by some alt
deploy at scale for large designs / config

Folks:
Has anyone tried to do bitstream build
farm image so I can duplicate that on

Hi Guys,

I've been playing around with the awesome FireSim repo, and in particular trying to build and launch experiments for an on-premises alveo card using Vitis.

Hi,

I've read several threads of people interested in using FireSim on local VCU118, XC706 or various Alveo cards but never saw that such support had been implemented.
What is the latest status of FireSim for local local Xilinx FPGA boards with XDMA PCIe support? Are there any examples to study?

"AWS EC2 F1 FPGAs are great but how do I use the on-premises FPGAs that I have?"

Support for On-Premises FPGAs

- Previously: Added support for Xilinx Alveo U250 FPGAs
  - (was experimentally released in 1.14.0)
- Integrates seamlessly with existing FireSim collateral + tooling
- Few line change to target on-premises FPGA vs AWS EC2 F1 FPGAs

Case Study: How to build and run simulations locally?

Building a U250 bitstream

- Creating a new build recipe
  - Use the `bit_builder_recipe` field to build a Vitis U250 bitstream
  - Everything else is shared from AWS EC2 F1 to Vitis!

```
firesim_rocket_singlecore_no_nic:
   DESIGN: FireSim
   TARGET_CONFIG: FireSimRocketConfig
   PLATFORM_CONFIG: BaseVitisConfig
   deploy_triplet: null
   platform_config_args:
      fpga_frequency: 60
      build_strategy: TIMING
   post_build_hook: null
   metasim_customruntime_config: null
   bit_builder_recipe: bit-builder-recipes/vitis.yaml
```

Single-core Rocket configuration with single DRAM channel

Building a U250 bitstream

- Running the bitstream build
  - Use the externally provisioned build farm to use a local machine
  - Everything else is the same!
- Run `firesim buildbitstream`

```
build_farm:
  base_recipe: build-farm-recipes/externally_provisioned.yaml
  recipe_arg_overrides:
    default_build_dir: <PATH TO USER BUILD DIRECTORY>
    build_hosts_to_use:
      - localhost

builds_to_run:
  - firesim_rocket_singlecore_no_nic
```

Building a U250 bitstream

- Expect to see a HWDB entry in `deploy/built-hwdb-entries/*`
- Similar format to AWS EC2 case, only has an `xclbin` instead of `agfi`

```
firesim_rocket_singlecore_no_nic:
  xclbin: <PATH TO XCLBIN FILE>
  deploy_triplet_override: FireSim-FireSimRocketConfig-BaseVitisConfig
  custom_runtime_config: null
```

- Support for sharing `xclbins` through URI
  - Store on-premises bitstreams in publicly accessible location (e.g. AWS S3)
  - Share bitstreams amongst multiple users

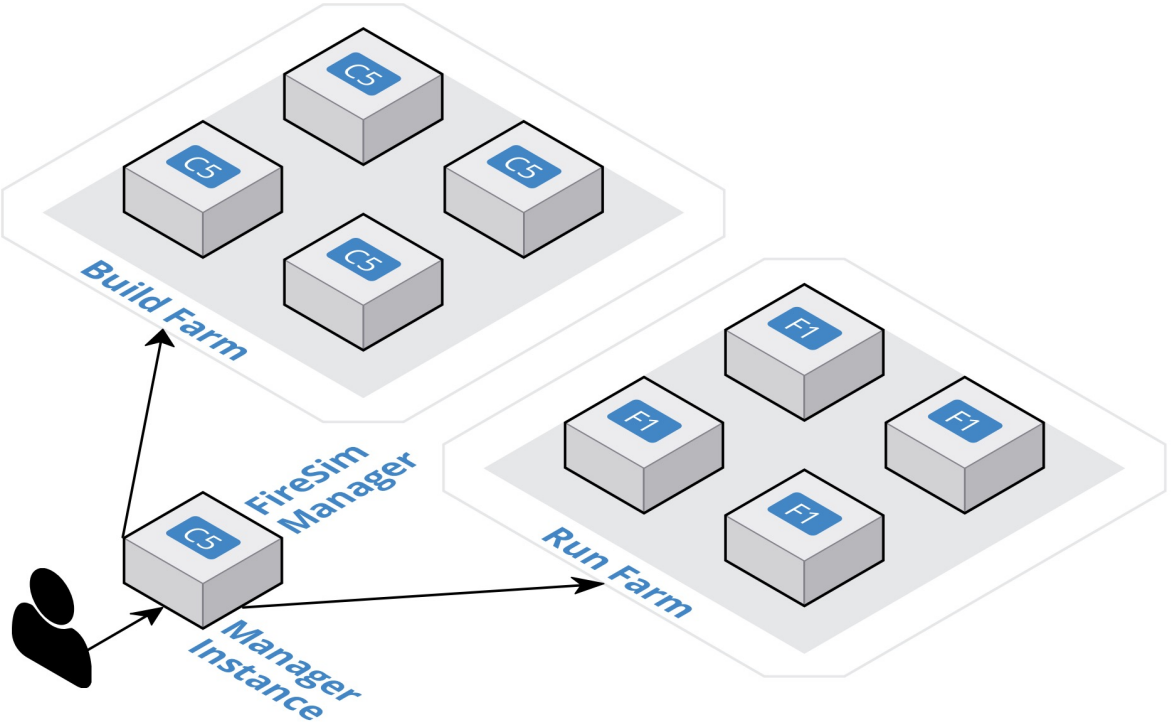
Running a U250 bitstream

- Uses externally provisioned run farm to target local FPGAs
  - In this case, a local machine with 4 U250s
  - Use `VitisInstanceDeployManager` to setup run farm hosts for U250s
- Use the HWDB entry created in the prior section
- Same process as before!
  - `launchrunfarm`, `infrasetup`, `runworkload`, `terminaterunfarm`
  - Attach to running screen session to interact
  - Have results automatically copied back

```
run_farm:
  base_recipe: run-farm-
recipes/externally_provisioned.yaml
  recipe_arg_overrides:
     default_platform: VitisInstanceDeployManager
     default_simulation_dir: <PATH TO SIM DIR>
     run_farm_hosts_to_use:
       - localhost: four_fpga_spec


target_config:
     topology: no_net_config
     no_net_num_nodes: 1
     link_latency: 6405
     switching_latency: 10
     net_bandwidth: 200
     profile_interval: -1

     default_hw_config:
firesim_rocket_singlecore_no_nic

workload:
     workload_name: linux-uniform.json
```

Behind the Scenes: Build + Run Farms

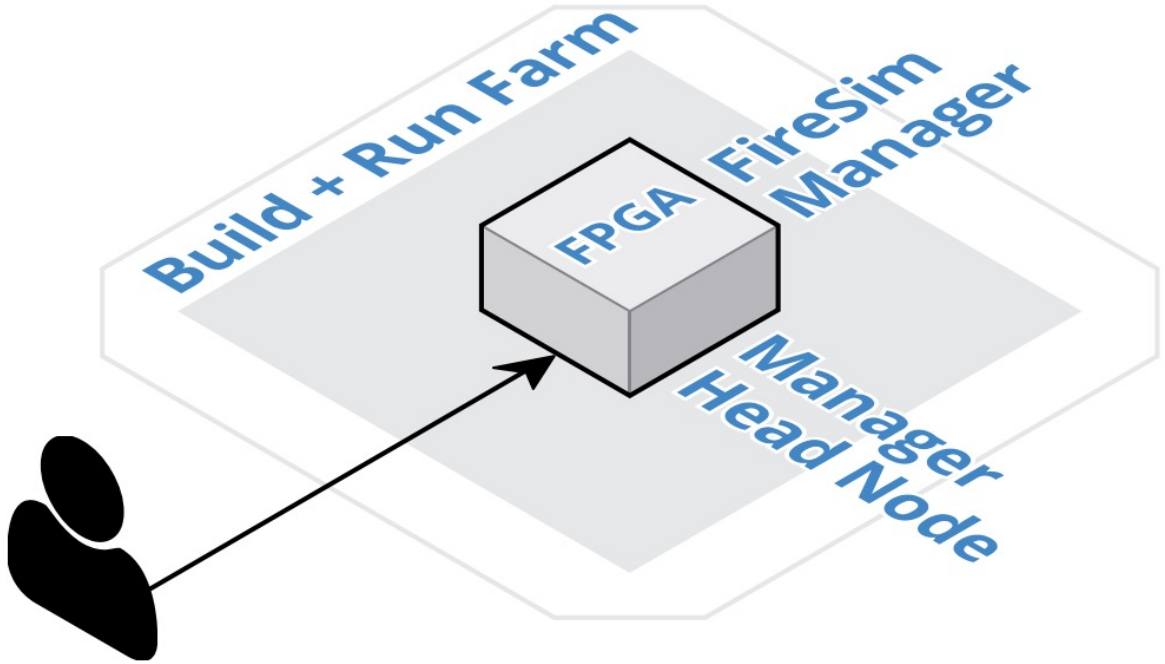
- Manager rearchitected for maximum configurability
  - Target different clouds/clusters
  - Convenient defaults for AWS EC2 and set of unmanaged machines (typical pre-setup cluster)



Default Distributed AWS EC2 Setup

Behind the Scenes: Build + Run Farms

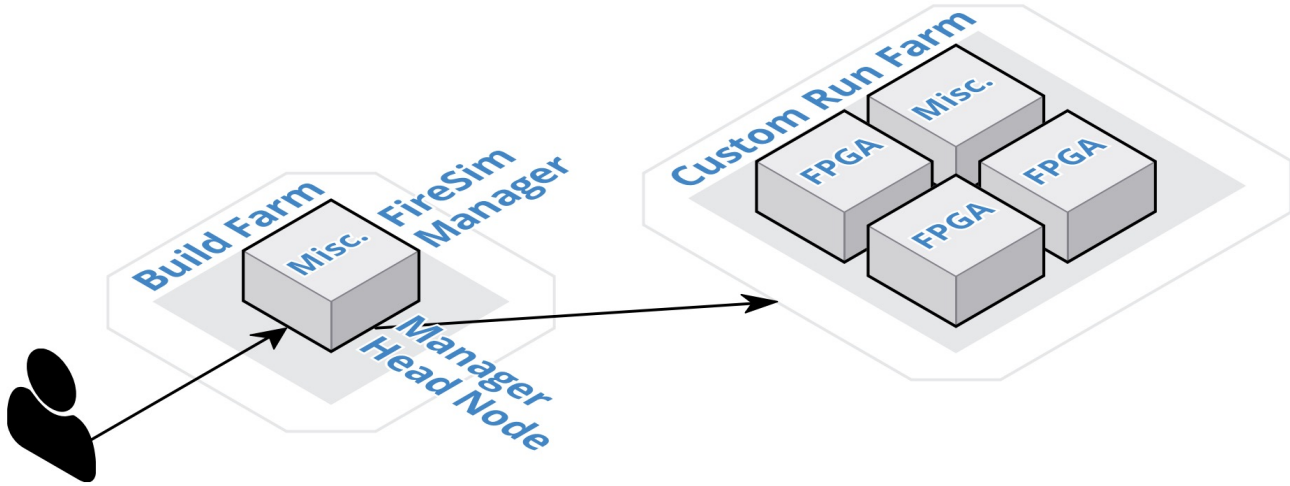
- Manager rearchitected for maximum configurability
  - Target different clouds/clusters
  - Convenient defaults for AWS EC2 and set of unmanaged machines (typical pre-setup cluster)



Completely Local Setup

Behind the Scenes: Build + Run Farms

- Manager rearchitected for maximum configurability
  - Target different clouds/clusters
  - Convenient defaults for AWS EC2 and set of unmanaged machines (typical pre-setup cluster)



Mixed Setup:
Local Builds + Distributed Simulations

Behind the Scenes: Build + Run Farms

- In `config_<build/runtime>.ini`
  - `base_recipe` sets type of build/run farm
  - You can modify its defaults by
    - Modifying the recipe file directly
    - Overriding using `recipe_arg_overrides`

```
run_farm:
  # managerinit replace start
  base_recipe: run-farm-recipes/aws_ec2.yaml
  # Uncomment and add args to override defaults.
  # Arg structure should be identical to the args given
  # in the base_recipe.
  #recipe_arg_overrides:
  #  <ARG>: <OVERRIDE>
  # managerinit replace end

metasimulation:
  metasimulation_enabled: false
  # vcs or verilator. use vcs-debug or verilator-debug for waveform g
  metasimulation_host_simulator: verilator
  # plusargs passed to the simulator for all metasimulations
  metasimulation_only_plusargs: "+fesvr-step-size=128 +dramsim +max-c
  # plusargs passed to the simulator ONLY FOR vcs metasimulations
  metasimulation_only_vcs_plusargs: "+vcs+initreg+0 +vcs+initmem+0"

target_config:
    # Set topology: no_net_config to run without a network simulation
    topology: example_8config
    no_net_num_nodes: 2
    link_latency: 6405
    switching_latency: 10
    net_bandwidth: 200
    profile_interval: -1

    # This references a section from config_hwdb.yaml for fpga-accele
    # or from config_build_recipes.yaml for metasimulation
    # In homogeneous configurations, use this to set the hardware con
    # for all simulators
    default_hw_config: firesim_rocket_quadcore_nic_l2_llc4mb_ddr3

    # Advanced: Specify any extra plusargs you would like to provide
    # booting the simulator (in both FPGA-sim and metasim modes). Thi
```

`config_runtime.yaml` example

Behind the Scenes: Build + Run Farms

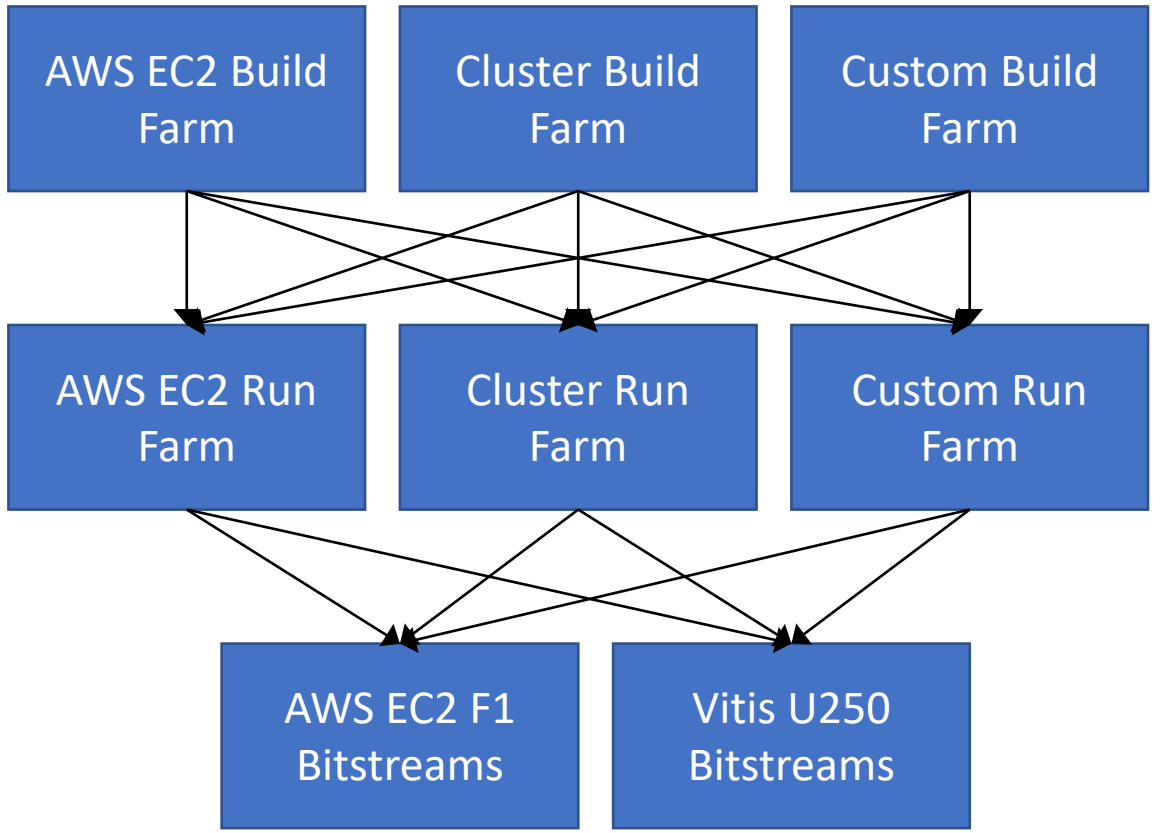
- Two types of default build/run farm types
- AWS EC2 (`aws_ec2.yaml`)
  - Default build/run farms used on AWS EC2
  - Fully distributed builds and simulations
  - Equivalent functionality to pre-1.14.0
- Externally Provisioned (`externally_provisioned.yaml`)
  - Use a pre-setup cluster of machines (including running locally)
  - Just needs FPGA platform (i.e. Vitis), number of FPGAs, and IP/hostname

Behind the Scenes: Bit Builders + Deploy Managers

- Notice how nothing was mentioned about type of FPGA used!
- Target different FPGA platforms as well!
  - AWS EC2 F1 or Vitis Alveo U250 FPGAs
- This is done by
  - Bit Builders – abstract bitstream build process
  - Deploy Managers - abstract setup of run farm hosts for FPGA platform
- You can see this in `config_build_recipes.yaml` and a specific run farm recipe (i.e. `aws_ec2.yaml`)

Behind the Scenes: Maximum Configurability

- Manager rearchitected for maximum configurability
  - Target different clouds/clusters
  - Target different bitstreams
  - And any combinations of them!

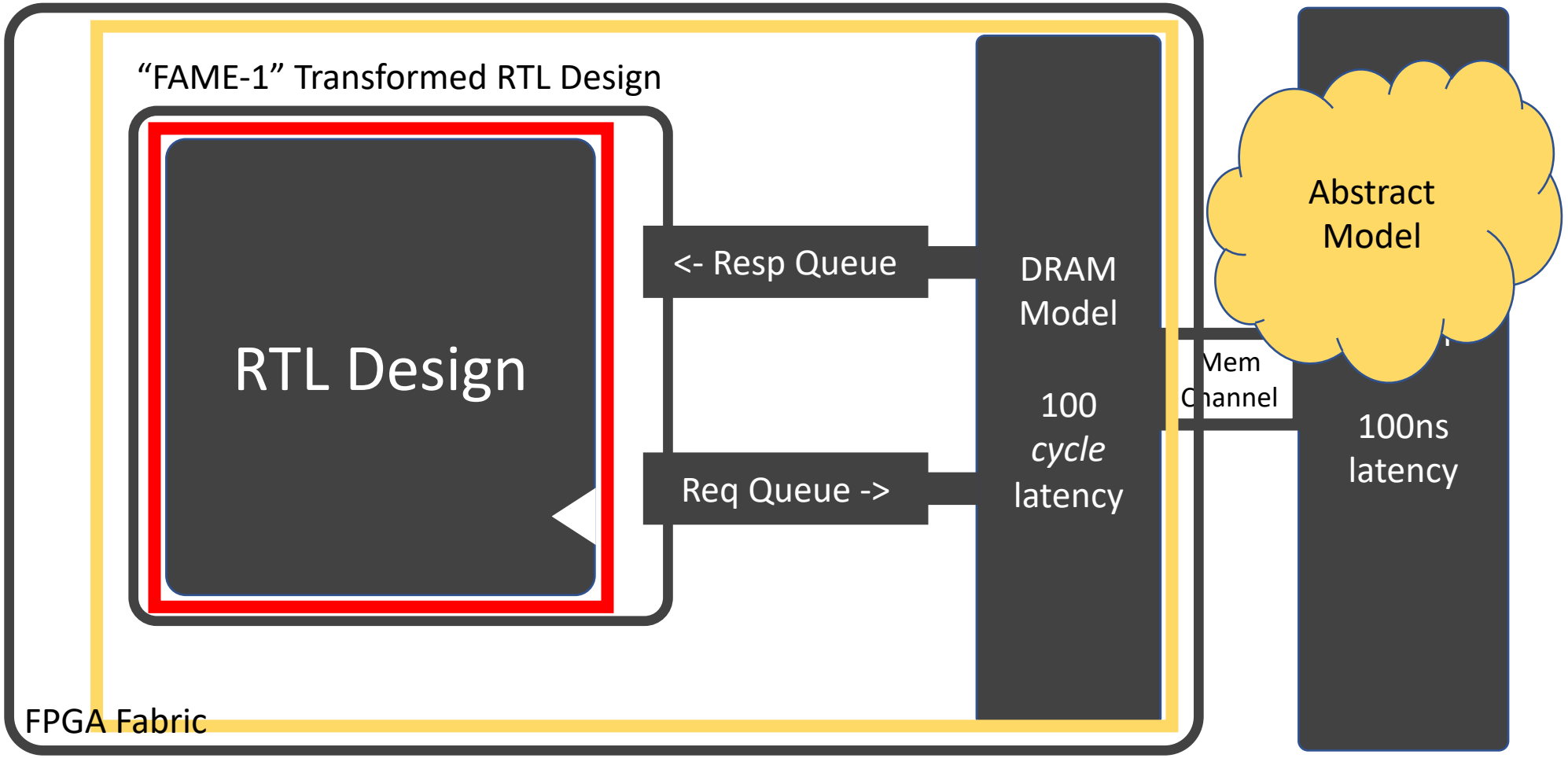


"Gah! My FireSim simulation breaks,
how do I do FireSim SW-level metasimulation again?"

Metasimulation Recap

Target-Level SW Simulation

Metasimulation



Metasimulation Recap

- Software RTL Simulation
  - Target design transformed by Golden Gate
  - Host-FPGA interfaces/shell emulated using abstract models
  - Uses existing FireSim models (i.e. DRAM, UART)

But how do I run it?

Running Metasimulations

- Original `make` API
  - In `$FDIR/sim`

```
$ make
    EMUL=<verilator|vcs>
    DESIGN=FireSimNoNIC
    run-asm-test-debug
```

- Issues
  - What are the `make` variables/targets I need to pass in?
  - How do I run multiple tests in parallel? Bash script it myself?
  - How do I run my existing FireMarshal workload with this?

Running Metasimulations

- Original `make` API
  - In `$FDIR/sim`

```
$ make
    EMUL=<
    DESIGN
    run-a
```

Better yet! Just have the FireSim manager do everything!

- Issues
  - What are the `make` variables/targets I need to pa
  - How do I run multiple tests in parallel? Bash script?
  - How do I run my existing FireMarshal workload with this?

Running In Metasimulation Mode

- In `config_runtime.yaml` use the `metasimulation` mapping
  - `enabled`: FPGA simulation → SW RTL metasimulation
  - `host_simulator`: Choose to run Verilator/VCS w/ and w/o waveforms
  - `*plusargs`: Extra non-FireSim specific arguments to pass to simulator
- Has same features as FPGA simulations!
  - Use arbitrary Run Farms
  - Automatic copying of results
  - Use FireMarshal workloads
  - Quickly do FPGASim ↔ MetaSim
  - Same performance results

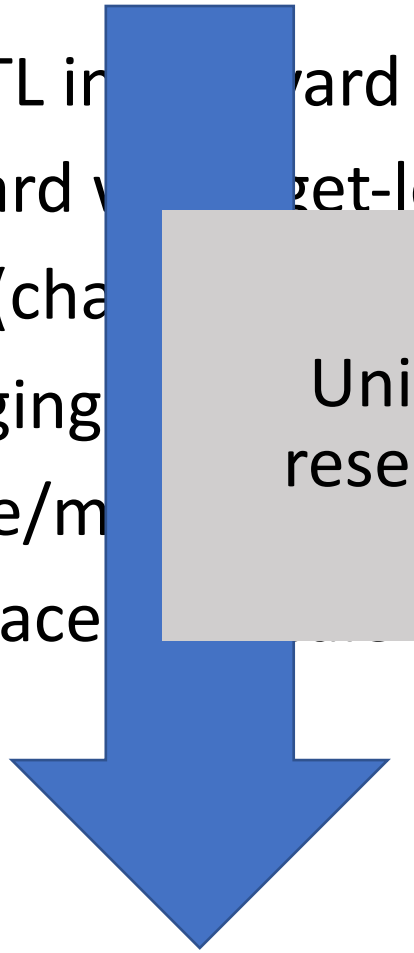
```
metasimulation:
    metasimulation_enabled: true
    metasimulation_host_simulator: verilator
    metasimulation_only_plusargs: …
    metasimulation_only_vcs_plusargs: …
```

Example Workflow

1. Write default RTL in Chipyard
2. Debug in Chipyard w/ target-level simulation
3. Port to FireSim (change config. files, use FireMarshal workload)
4. DSE and debugging w/ single/multi-node metasimulations
5. Testing w/ single/multi-node FPGA simulations using Vitis U250s
6. Scale-out to datacenter scale with AWS EC2 F1

Example Workflow

1. Write default RTL in …yard
2. Debug in Chipyard w… …get-level simulation
3. Port to FireSim (cha… …kload)
4. DSE and debugging …s
5. Testing w/ single/m… …s U250s
6. Scale-out to datace…



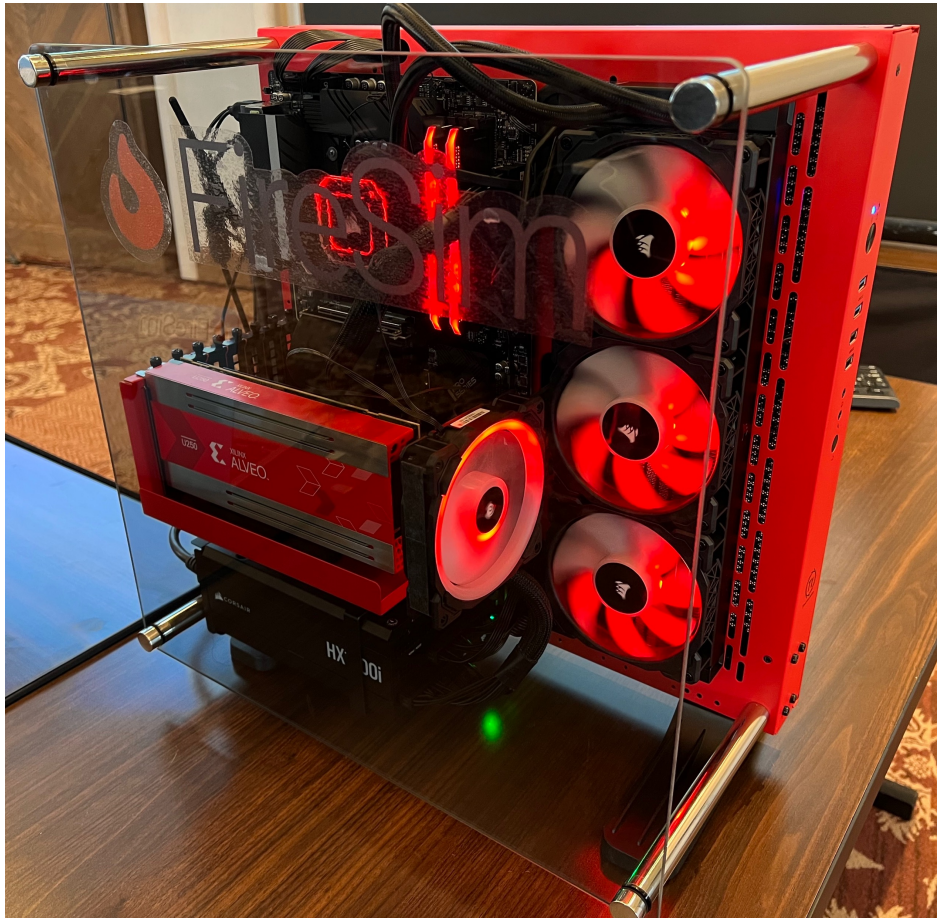
Unified workflow for agile research of RISC-V systems!

Demo Time!

FireStation v1 Machine Specs

- Intel Core i7 13700K
  - Liquid cooler (w/RGB)
- 32 GB DDR4 (w/RGB)
- **Xilinx Alveo U250 (active)**
- Motherboard spec'd for:
  - 2 U250 + GPU
  - OR
  - 3 U250
- 1500W PSU to support multi-FPGA/GPU
- Thermaltake Core P3 Red Case
- Ubuntu 18.04

$1500 without FPGAs or GPUs



Porting to new Xilinx FPGAs should be easy!

FireSim Interfaces

MMIO

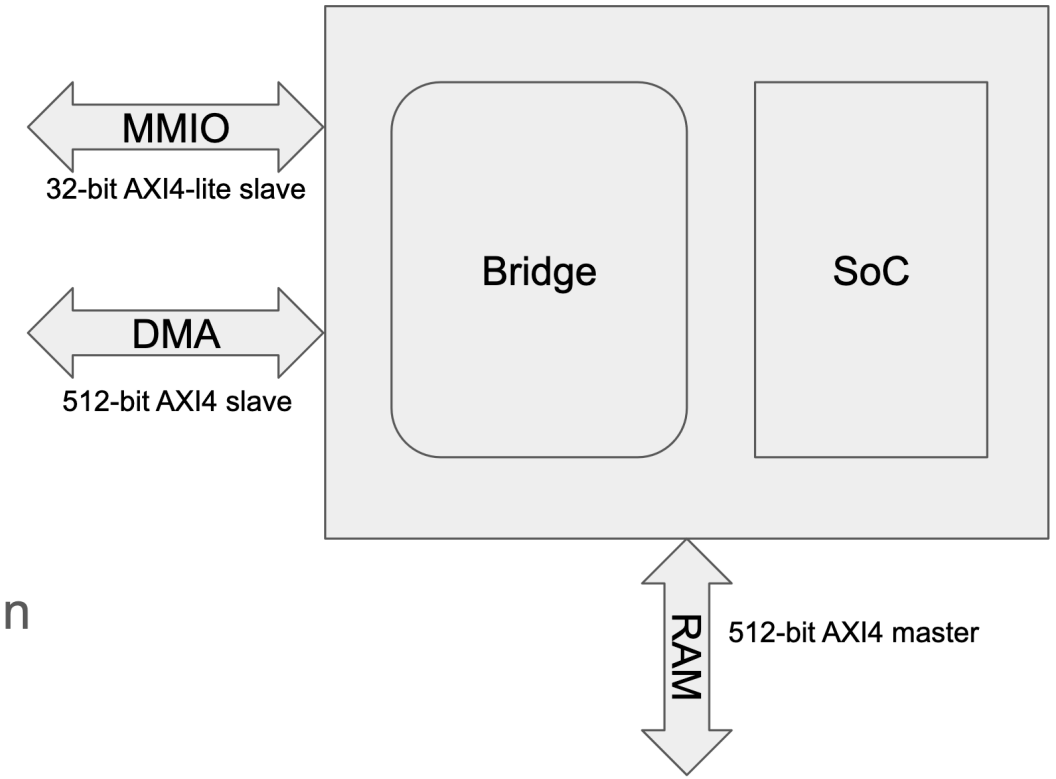
- Used by `simif_t::read()` and `simif_t::write()`

DMA

- Used by `simif_t::push()` and `simif_t::pull()`

RAM

- Primarily used by FASED for main memory simulation



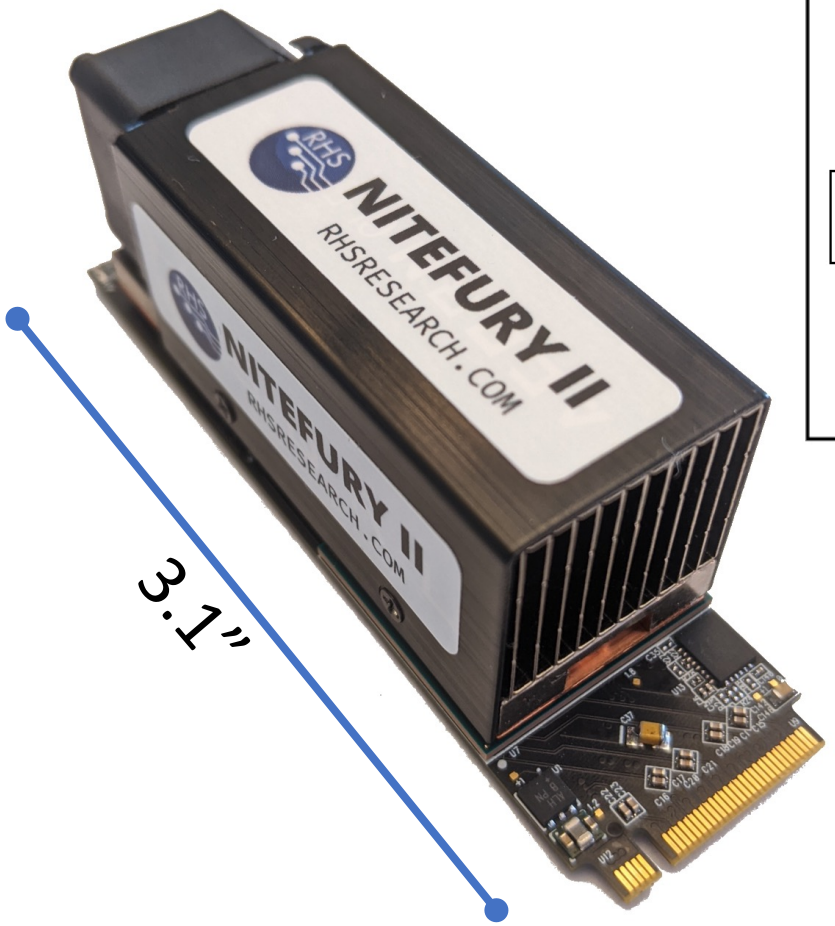
Credit: David Christoph Metz & Magnus Själander (NTNU), *First FireSim/Chipyard Workshop, co-located with ASPLOS 2023.*

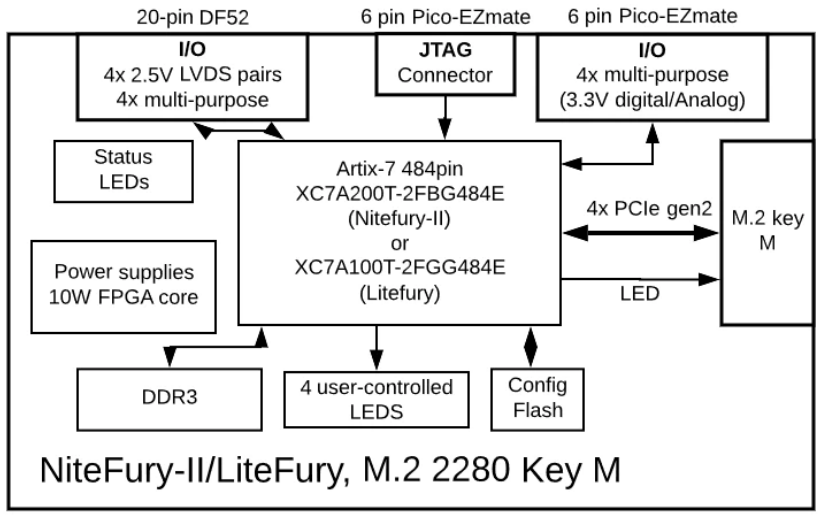
Support for two new FPGAs added to FireSim in 2 days: (#1) Xilinx VCU118

- Similar to U250 / F1, but “dev board” format
- Highly requested by users
- Usual PCIe + DRAM setup available to FPGA
- Less than 24 hours from “decide to do it” to “booting Linux”

Support for two new FPGAs added to FireSim in 2 days: (#2) RHS Research Nitefury II

- M.2 Form-factor board originally developed for Bitcoin mining
  - Xilinx XC7A200T-2FBG484E
  - 1GB DDR3
- A few reports on the internet of successful usage in an M.2 to Thunderbolt enclosure w/ regular XDMA IP/software stack
- So we bought a few!





RHS Research Nitefury II vs. Prior FireSim Host Platforms

- Previous host platforms supported by FireSim are cloud platforms or require $8-10K per-board
  - Also, require desktop/server with PCIe
- Instead:
  - $150 FPGA
  - $150 M2 <-> Thunderbolt enclosure (cheaper ones may work, untested)
  - Linux Laptop

On-premises FPGA support now available!

- High-level of automation/reproducibility enabled by FireSim on AWS F1 cloud now extended to local/on-prem FPGAs:
  - Went from new machine with no FPGA attached to working FPGA-accelerated simulation in 1 hour and 40 mins
- New in latest release: Xilinx VCU118, Xilinx Alveo U250 / U280 XDMA-shell, RHS Research Nitefury II
- Use existing FireSim features at-scale and locally!
  - Cycle-accurate simulation
  - Debugging
    - Integrated logic analyzers, trace dumps, synth. assert/prints, co-simulation
  - Software support
    - FireMarshal workload management
  - … and more!

Summary

- Customize how/what/where you build/run things
  - Local Builds → Fully Distributed AWS EC2 Builds
  - Local Simulations → Fully Distributed AWS EC2 Simulation
  - And everything in between
- Target both local and AWS EC2 FPGAs
- Distributed SW RTL metasimulations
  - Debug RTL using unified infrastructure
  - Use FireSim modeling + features in SW RTL simulation

Check out https://docs.fires.im/ for more usage details