Hammer Tutorial

Nayiri Krzysztofowicz

UC Berkeley

nayiri@berkeley.edu

Berkeley Architecture Research

CHIPYARD

Components of VLSI Flows

Design

- RTL
- IP cores
- Floorplan
- Constraints
- …

Tech

- Intel 16
- TSMC N5
- SkyWater 130nm
- …

Tools

- Synopsys VCS
- Cadence Genus
- OpenROAD
- …

This Tutorial

Design

- Tiny RISC-V Rocket Core

Tech

- SkyWater 130nm

Tools

- Open-source
  - Yosys
  - OpenROAD
  - Magic
  - Netgen

Goals

- Hammer applications
- Fixing traditional VLSI flows: One size doesn't fit all
- Overview of Hammer's abstractions
- Get you started with a TinyRocketConfig in Sky130 + OpenROAD
- Under the hood: plugins, hooks, etc.

How things will work

```
# command 1
> echo “Chipyard Rules!”

# command 2
> do_this arg1 arg2
```

Terminal Section

```
# Technology Setup
# Technology used is Sky130
vlsi.core.technology: sky130
# Technology paths
technology.sky130.sky130A: ""
```

Inside-a-File Section

How to use Hammer in Chipyard

- Hammer integrated under `chipyard/vlsi/`
- Supports a tapeout-verified commercial flow
- Newly supports a fully open-source flow

```
chipyard/
  vlsi/
    Makefile
    example-designs/
      sky130-openroad.yml
      sky130-rocket.yml
    example-openroad.yml
    example-sky130.yml
    example-vlsi-sky130
    hammer/
    hammer-cadence-plugins/
    hammer-synopsys-plugins/
    hammer-<tech>-plugin
    tutorial.mk
build.sbt
```

Hammer + Sky130 Example

- Prerequisites in tutorial README
  - E.g. access to CAD tools, PDK download, etc.
- Sky130 PDK
  - Open-source 130nm PDK from SkyWater Technology
- OpenRAM SRAM Macros
- Yosys Open Synthesis Suite
- OpenROAD Place-and-Route Tool

Getting Started: Tool/Tech Setup

- Get PDK and EDA tools
  - sky130A – Skywater 130nm PDK
  - sky130_sram_macros – OpenRAM SRAMs
  - Yosys – open-source synthesis tool
  - OpenROAD – open-source P&R tool (not downloaded for this tutorial)

```
> wget https://raw.githubusercontent.com/ucb-bar/chipyard/tutorial-testing/openroad-setup.sh

> source openroad-setup.sh
```

Getting Started: Hammer Steup

- Initialize Hammer plugins

```
> cd ~/chipyard-morning
> ./scripts/init-vlsi.sh sky130 openroad
> source env.sh
```

Wrapper around `git submodule init` to clone the hammer repository (do this once)

- Define the Hammer environment into the shell

```
> cd vlsi
> export HAMMER_HOME=$PWD/hammer
> source $HAMMER_HOME/sourceme.sh
```

Do this each time you use Hammer in a new terminal

Getting Started: Makefile

- Tutorial configurations

```
ifeq ($(tutorial),sky130-openroad)
  tech_name    ?= sky130
  CONFIG       ?= TinyRocketConfig
  TOOLS_CONF   ?= example-openroad.yml
  TECH_CONF    ?= example-sky130.yml
  DESIGN_CONF  ?= example-designs/sky130-openroad.yml
  EXTRA_CONFS  ?= $(if $(filter $(VLSI_TOP),Rocket),
                   example-designs/sky130-rocket.yml, )
  INPUT_CONFS  ?= $(TOOLS_CONF) $(TECH_CONF) $(DESIGN_CONF) $(EXTRA_CONFS)
  VLSI_OBJ_DIR ?= build-sky130-openroad
endif
```

```
chipyard/
  vlsi/
    Makefile
    example-designs/
      sky130-openroad.yml
      sky130-rocket.yml
    example-openroad.yml
    example-sky130.yml
    example-vlsi-sky130
    hammer/
    hammer-cadence-plugins/
    hammer-synopsys-plugins/
    hammer-<tech>-plugin
    tutorial.mk
build.sbt
```

Getting Started: Tools, PDK

- Fill in the path to your PDK + OpenRAM SRAM library

```
# Technology Setup
vlsi.core.max_threads: 16
# Technology used is Sky130
vlsi.core.technology: sky130
# Specify PDK and SRAM install directories
technology.sky130. sky130A: "/home/centos/tutorial-installs/sky130A"
technology.sky130.openram_lib: "/home/centos/tutorial-
installs/sky130_sram_macros"
```

```
chipyard/
  vlsi/
    Makefile
    example-designs/
      sky130-openroad.yml
      sky130-rocket.yml
    example-openroad.yml
    example-sky130.yml
    example-vlsi-sky130
    hammer/
    hammer-cadence-plugins/
    hammer-synopsys-plugins/
    hammer-<tech>-plugin
    tutorial.mk
build.sbt
```

Getting Started: Tools, PDK

- Select the EDA tool plugins to use

```
# Tool options. Replace with your tool plugin of choice.
# Yosys options
vlsi.core.synthesis_tool: "yosys"
vlsi.core.synthesis_tool_path: ["hammer/src/hammer-vlsi/synthesis"]
vlsi.core.synthesis_tool_path_meta: "append"
# OpenROAD options
vlsi.core.par_tool: "openroad"
vlsi.core.par_tool_path: ["hammer/src/hammer-vlsi/par"]
vlsi.core.par_tool_path_meta: "append"
```

These tools should already be on your PATH

```
chipyard/
  vlsi/
    Makefile
    example-designs/
      sky130-openroad.yml
      sky130-rocket.yml
    example-openroad.yml
    example-sky130.yml
    example-vlsi-sky130
    hammer/
    hammer-cadence-plugins/
    hammer-synopsys-plugins/
    hammer-<tech>-plugin
    tutorial.mk
build.sbt
```

Getting Started: Design

- Set the target clock period

```
# Override configurations in ../example-sky130.yml

# Specify clock signals
# Relax the clock period for OpenROAD to meet timing
vlsi.inputs.clocks: [
  {name: "clock_clock", period: "30ns", uncertainty: "1ns"}
]
```

```
chipyard/
  vlsi/
    Makefile
    example-designs/
      sky130-openroad.yml
      sky130-rocket.yml
    example-openroad.yml
    example-sky130.yml
    example-vlsi-sky130
    hammer/
    hammer-cadence-plugins/
    hammer-synopsys-plugins/
    hammer-<tech>-plugin
    tutorial.mk
build.sbt
```

Berkeley Architecture Research

Getting Started: Design

- Define the floorplan/macro placement

```
# Placement Constraints
vlsi.inputs.placement_constraints:
  - path: "ChipTop"
    type: toplevel
    x: 0
    y: 0
    width: 4000
    height: 2500

  # Place data cache SRAM instances
  - path: "ChipTop/system.tile_prci_domain.tile_reset_domain.tile.
           dcache.data.data_arrays_0.data_arrays_0_ext.mem_0_0"
    type: hardmacro
    x: 50
    y: 100
    orientation: r0
```

```
chipyard/
  vlsi/
    Makefile
    example-designs/
      sky130-openroad.yml
      sky130-rocket.yml
    example-openroad.yml
    example-sky130.yml
    example-vlsi-sky130
    hammer/
    hammer-cadence-plugins/
    hammer-synopsys-plugins/
    hammer-<tech>-plugin
    tutorial.mk
build.sbt
```

Getting Started: Design

- Running flow only on Rocket core

```
# Specify clock signals
# Rocket/RocketTile names clock signal "clock" instead of
"clock_clock"
vlsi.inputs.clocks: [
  {name: "clock", period: "5ns", uncertainty: "1ns"}
]

# Placement Constraints
# Rocket/RocketTile requires a much smaller footprint
vlsi.inputs.placement_constraints:
  - path: "Rocket"
    type: toplevel
    x: 0
    y: 0
    width: 2500
    height: 1500
```

```
chipyard/
  vlsi/
    Makefile
    example-designs/
      sky130-openroad.yml
      sky130-rocket.yml
    example-openroad.yml
    example-sky130.yml
    example-vlsi-sky130
    hammer/
    hammer-cadence-plugins/
    hammer-synopsys-plugins/
    hammer-<tech>-plugin
    tutorial.mk
build.sbt
```

Building the Design

Let's elaborate a TinyRocketConfig:

```
> make buildfile tutorial=sky130-openroad
```

This does the following:

- Runs the generator for Top and Harness
  - For the Top, MacroCompiler maps memories to Sky130's SRAMs

```
chipyard/
  vlsi/
    generated-src/<long-name>/
      … .v, .json, etc.
    build/<long-name>-ChipTop/
      sram_generator_output.json
      hammer.d
      inputs.yml
```

<long-name> should be chipyard.TestHarness.TinyRocketConfig

This will take a while!

Building the Design

Let's elaborate a TinyRocketConfig:

```
> make buildfile tutorial=sky130-openroad
```

This does the following:

- Runs the generator for Top and Harness
  - For the Top, MacroCompiler maps memories to Sky130's SRAMs
- Hammer generates a set of ExtraLibrarys for each selected SRAM

```
chipyard/
  vlsi/
    generated-src/<long-name>/
      … .v, .json, etc.
    build/<long-name>-ChipTop/
      sram_generator_output.json
      hammer.d
      inputs.yml
```

The first time Hammer is run with Sky130, the PDK is hacked into build/<long-name>-ChipTop/tech-sky130-cache

Building the Design

Let's elaborate a TinyRocketConfig:

```
> make buildfile tutorial=sky130-openroad
```

This does the following:

- Runs the generator for Top and Harness
  - For the Top, MacroCompiler maps memories to Sky130's SRAMs
- Hammer generates a set of ExtraLibrarys for each selected SRAM
- Hammer generates a set of Make targets implementing the VLSI flow graph and the input Hammer IR to the flow

```
chipyard/
  vlsi/
    generated-src/<long-name>/
      … .v, .json, etc.
    build/<long-name>-ChipTop/
      sram_generator_output.json
      hammer.d
      inputs.yml
```

You should reference these targets when/if you want to manually run flow steps

Building the design

More on SRAM mapping...

```
> ls generated-src/chipyard.TestHarness.TinyRocketConfig/*mem*

generated-src/chipyard.TestHarness.TinyRocketConfig/chipyard.TestHarness.TinyRocketConfig.harness.mems.conf
generated-src/chipyard.TestHarness.TinyRocketConfig/chipyard.TestHarness.TinyRocketConfig.memmap.json
generated-src/chipyard.TestHarness.TinyRocketConfig/chipyard.TestHarness.TinyRocketConfig.mems.hammer.json
generated-src/chipyard.TestHarness.TinyRocketConfig/chipyard.TestHarness.TinyRocketConfig.top.mems.conf
generated-src/chipyard.TestHarness.TinyRocketConfig/chipyard.TestHarness.TinyRocketConfig.top.mems.v
```

Flow Execution

User input (Make)

```
> make syn tutorial=sky130-openroad -n

./example-vlsi-sky130 -p ~chipyard/vlsi/example-openroad.yml -p ~chipyard/vlsi/example-sky130.yml
-p ~chipyard/vlsi/build/chipyard.TestHarness.TinyRocketConfig-ChipTop/inputs.yml
-p ~chipyard/vlsi/build/chipyard.TestHarness.TinyRocketConfig-ChipTop/sram_generator-output.json
--obj_dir ~chipyard/vlsi/build/chipyard.TestHarness.TinyRocketConfig-ChipTop syn


> make par tutorial=sky130-openroad -n

./example-vlsi-sky130
-p ~chipyard/vlsi/build/chipyard.TestHarness.TinyRocketConfig-ChipTop/syn-rundir/syn-output-full.json
-o ~chipyard/vlsi/build/chipyard.TestHarness.TinyRocketConfig-ChipTop/par-input.json
--obj_dir ~chipyard/vlsi/build/chipyard.TestHarness.TinyRocketConfig-ChipTop syn-to-par

./example-vlsi-sky130 -p ~chipyard/vlsi/build/chipyard.TestHarness.TinyRocketConfig-ChipTop/par-input.json
--obj_dir ~chipyard/vlsi/build/chipyard.TestHarness.TinyRocketConfig-ChipTop par
```

Hammer tool commands


```
> export tutorial=sky130-openroad
```

Running the VLSI Flow

Run Yosys synthesis

```
# synthesize only the Rocket core
> make syn tutorial=sky130-openroad
VLSI_TOP=Rocket

# synthesize the entire SoC (VLSI_TOP=ChipTop)
> make syn tutorial=sky130-openroad
```

```
chipyard/
  vlsi/
    generated-src/<long-name>/
      … .v, .json, etc.
    build/<long-name>-ChipTop/
      sram_generator_output.json
      hammer.d
      inputs.yml
      syn-rundir/
      par-rundir/
```

P&R takes a long time!
Lots of tool log output too…

Running the VLSI Flow

More on Synthesis output...

```
> ls build-sky130-openroad/chipyard.TestHarness.TinyRocketConfig-ChipTop/syn-rundir/
ChipTop.mapped.blif
ChipTop.mapped.hier.v
ChipTop.mapped.sdc
ChipTop.mapped.v
ChipTop.synth_check.rpt
ChipTop.synth_stat.txt
config_db_tmp.json
syn-output-full.json
syn-output-history.yml
syn-output.json
syn.tcl
```

Running the VLSI Flow

Run Yosys synthesis and OpenROAD P&R:

```
> make syn tutorial=sky130-openroad
> make par tutorial=sky130-openroad
```

Result netlists, GDS, etc. appear in here

```
chipyard/
  vlsi/
    generated-src/<long-name>/
      … .v, .json, etc.
    build/<long-name>-ChipTop/
      sram_generator_output.json
      hammer.d
      inputs.yml
      syn-rundir/
      par-rundir/
```

P&R takes a long time!
Lots of tool log output too…

Running the VLSI Flow

Run Yosys synthesis and OpenROAD P&R:

```
> make syn tutorial=sky130-openroad
> make par tutorial=sky130-openroad
```

(Optional) View the final OpenROAD database:

```
> cd build/<long-name>-ChipTop/par-rundir
> ./generated-scripts/open_chip
```

*Can only run this in an interactive session

```
chipyard/
  vlsi/
    generated-src/<long-name>/
      … .v, .json, etc.
    build/<long-name>-ChipTop/
      sram_generator_output.json
      hammer.d
      inputs.yml
      syn-rundir/
      par-rundir/
```

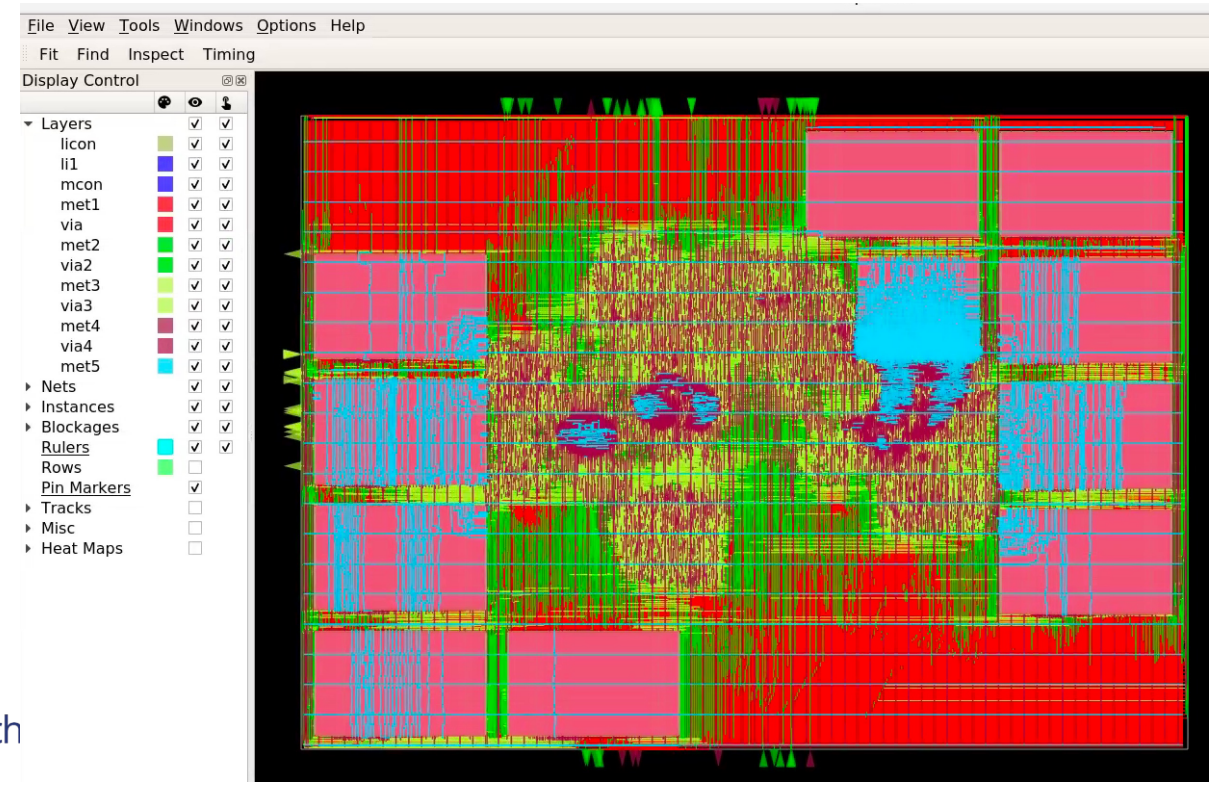
Running the VLSI Flow

More on Synthesis output...

```
> ls build-sky130-openroad/chipyard.TestHarness.TinyRocketConfig-ChipTop/par-rundir/
ChipTop.mapped.sdc                  post_floorplan_design
clock_constraints_fragment.sdc      post_global_placement
config_db_tmp.json                  post_global_route
enter                               post_init_design
floorplan.tcl                       post_initial_global_placement
generated-scripts                   post_io_placement
par.tcl                             post_macro_placement
pin_constraints_fragment.sdc        post_place_tapcells
post_add_fillers                    post_power_straps
post_clock_tree                     post_resize
post_detailed_placement             post_sky130_set_wire_rc
post_detailed_route                 power_straps.pdn
post_dummy_step
post_extraction
```

Running the VLSI Flow

```
> cd build/<long-name>-ChipTop/par-rundir
> ./generated-scripts/open_chip
```



VLSI Flow Control

Start after a step:

```
> make redo-par HAMMER_EXTRA_ARGS="--start-after-step detailed_route"
```

Run only a single step:

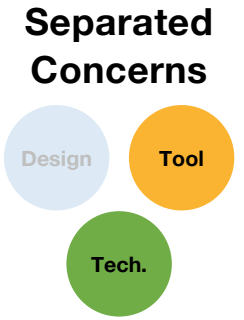
```
> make redo-par HAMMER_EXTRA_ARGS="--only-step power_straps"
```

Full list of options are on the Hammer documentation website.

Goals

- Hammer applications
- Fixing traditional VLSI flows: One size doesn't fit all
- Overview of Hammer's abstractions
- Get you started with a TinyRocketConfig in Sky130 + OpenROAD
- Under the hood: plugins, hooks, etc.

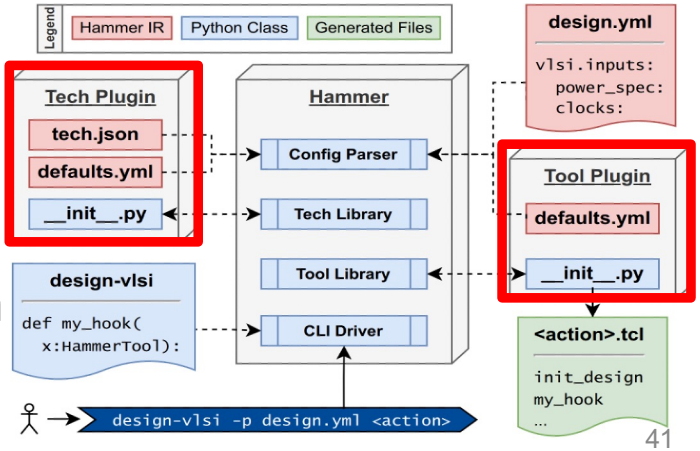
Under the Hood: Plugins



- Two types of plugin: Tool and Technology
  - `<tool>` is usually of the format `<action>/<name>`
  - e.g. `par/innovus` or `syn/dc`

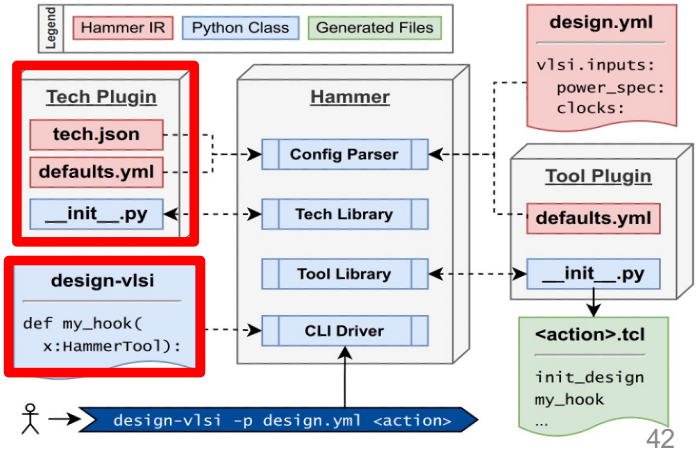
- Tool plugins contain:
  - `<tool>/defaults.yml` – overridable default settings for the tool
  - `<tool>/__init__.py` – Reusable python methods, implement Hammer APIs

- Technology plugins contain:
  - `<name>.tech.json` – pointers to relevant PDK files
  - `defaults.yml` – overridable default settings
  - `<name>/<tool>/__init__.py` – Reusable python methods



Under the Hood: “Hooks”

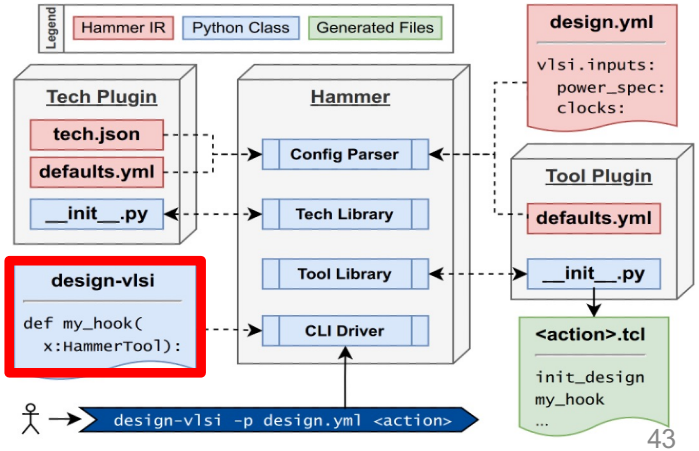
- The “Magic Tcl scripts” aren’t going away soon
  - Foundry reference flow, previous tapeouts
  - A lot of expertise captured in these scripts
- Hooks enable insertion of custom Python and Tcl scripts within the Hammer-generated flow
  - Quick-and-dirty
  - Cleanly allows “hacks” and workarounds
  - Allows prototyping of future APIs
  - Upstreamed hooks available for future re-use



“Hooks” – example-vlsi

- Example placeholders for hooks
  - For: setting non-default tool settings, custom fiducial/endcap placement, etc.
  - Can also be defined by technology plugins. ASAP7 example:
    - `asap7_scale_gds_script` method in ASAP7’s `__init__.py`
    - Inserted as a hook after `write_design`. Runs a Python script (`scale_gds.py`)

```
def example_tool_settings(x: hammer_vlsi.HammerTool) -> bool:
  if x.get_setting("vlsi.core.technology") == "sky130":
    x.append('''
# only route in met1 to met4
set_db route_design_bottom_routing_layer 2
set_db route_design_top_routing_layer 5
''')
  return True
```



Everyone can use Hammer

How: provide sensible defaults with methods to override

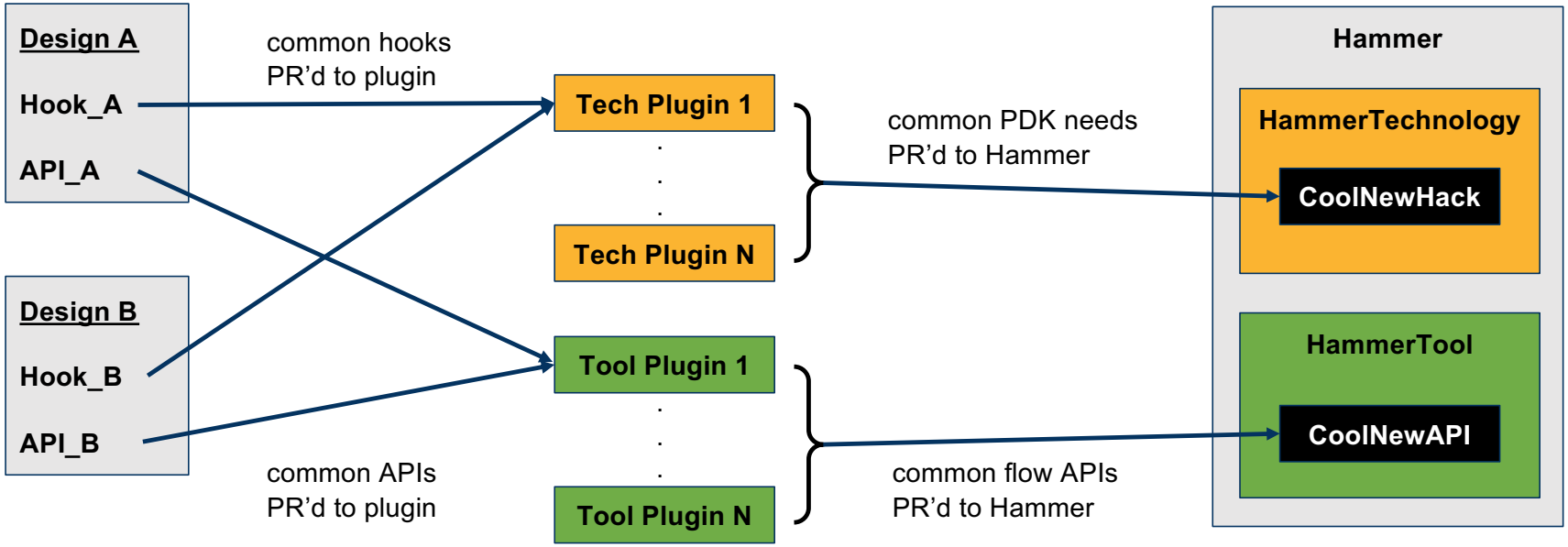
Sensible default	Override method
A default set of flow steps for every action (syn, par, etc.)	Hooks - inject your own steps anywhere
Auto-generated timing (SDC) & power (CPF) constraints	Use your own custom SDC and CPF files
Auto-generated power meshes from high-level parameters	Use foundry-provided or your own mesh generator
Auto-generated Makefile implementing flow graph	Running Hammer via command line, custom Makefiles

Result: gets you 80-90% of the way there out of the box

- Easily learn the VLSI flow, get early design feedback
- Chipyard examples with ASAP7, Sky130

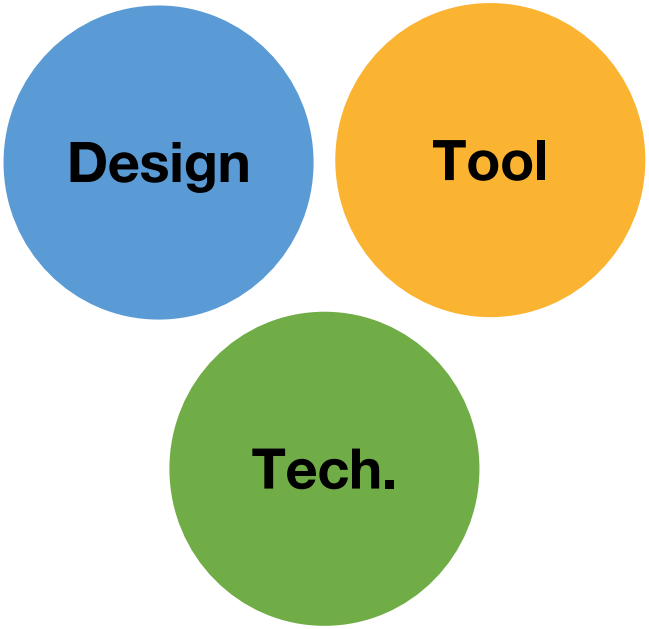
Everyone can contribute to Hammer

Modular design + override features = community contributions



Summary

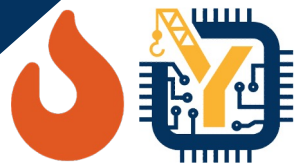
- Physical design is hard—there are good reasons why most people try to avoid it.
  - Chips are growing in complexity
  - Un-natural evolution of the EDA/PDK stack
- Hammer helps separate design, tool, and technology concerns
  - Enables re-use
  - Enables advanced abstractions and generators
- Easy power and area evaluation
  - Using Hammer, open source PDK, commercial EDA



Future of Hammer

- More tool integration: metrics parsing, constraints feedback
- Aspect-Oriented Chisel Floorplanning
  - Higher-level abstractions
    - Hierarchical partitioning, power strap/pin alignment
    - BAG IP collateral generation/integration
  - Reconfiguring Chisel designs based on physical design feedback
    - e.g. change clock constraints based on timing metrics
- Abutment/partition-based hierarchical flow
- True multi-clock & power domain (for DVFS, etc.)
- Dev work is cyclical -> typically happens alongside tapeouts
  - Lots of ideas, help always wanted

Learn More



- Github: https://github.com/ucb-bar/hammer/
- Documentation: https://hammer-vlsi.readthedocs.io/
- Chipyard-specific documentation: https://chipyard.readthedocs.io/en/latest/VLSI/index.html
- User mailing list: hammer-users@googlegroups.com
- Plugin access requests: hammer-plugins-access@lists.berkeley.edu
  - Cadence, Synopsys, and Mentor
- UCB Digital Design labs: https://github.com/EECS150/asic_labs_sp22
  - full lab releases coming soon!

Berkeley Architecture Research

Coming up…

FPGA Prototyping