Golden Gate

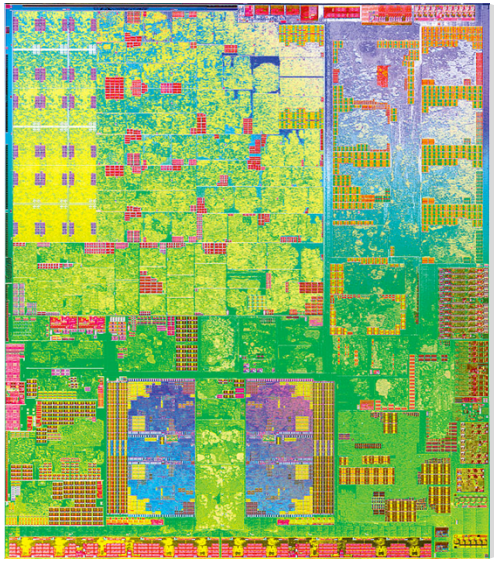
Bridging The Resource-Efficiency Gap Between ASICs and FPGA Prototypes

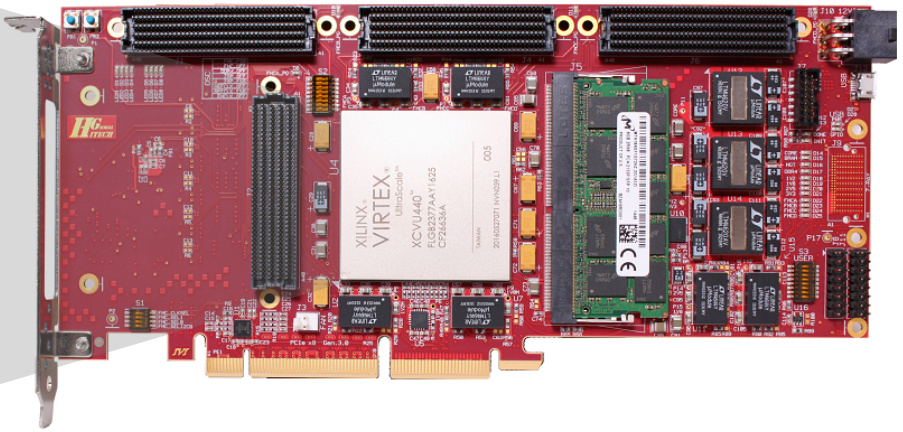
Albert Magyar, David Biancolin, Jack Koenig, Sanjit Seshia, Jonathan Bachrach, Krste Asanović

Two major challenges of FPGA simulation

- Labor-intensive
- Chip might not fit

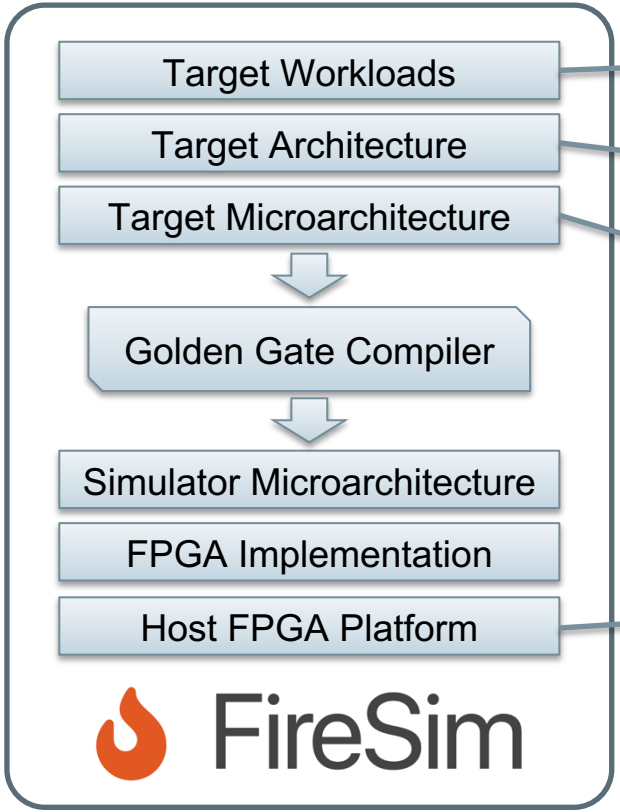
FireSim



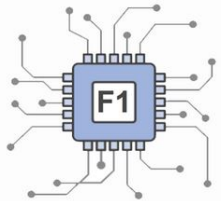


Karandikar et al., “FireSim: FPGA-Accelerated Cycle-Exact Scale-Out System Simulation in the Public Cloud,” ISCA ‘18.

FireSim: The easy button for FPGA simulation



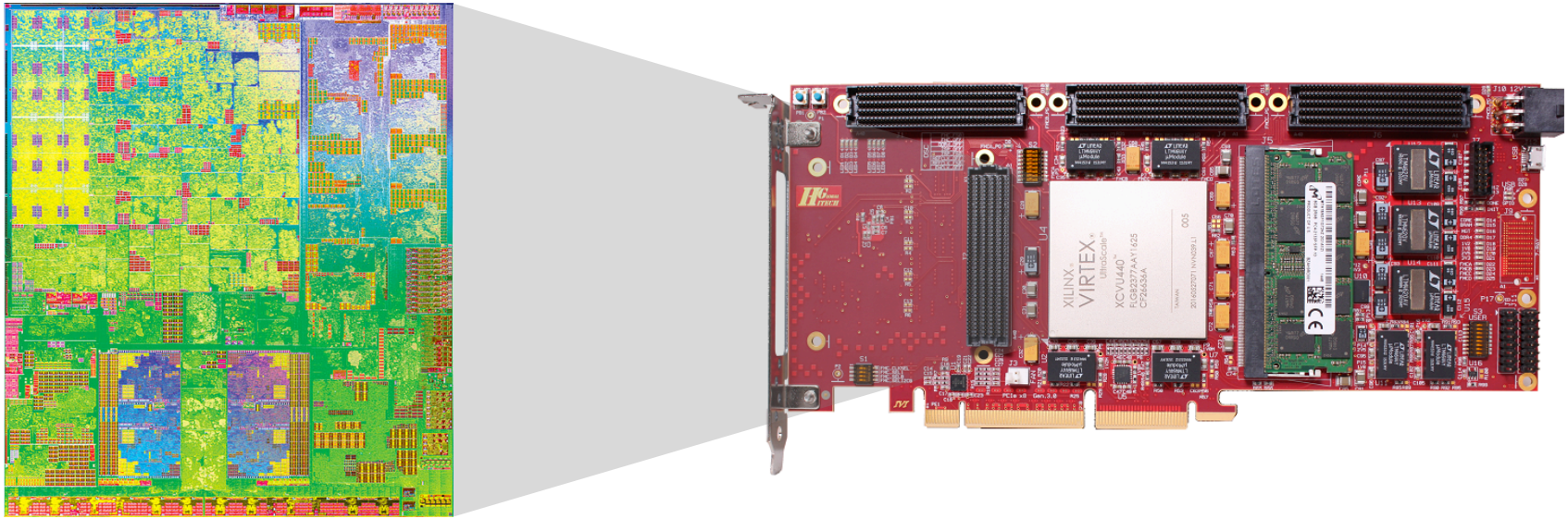
Flexible SoC generators
User accelerator designs
Architectural experiments



“Batteries included” for the full stack!

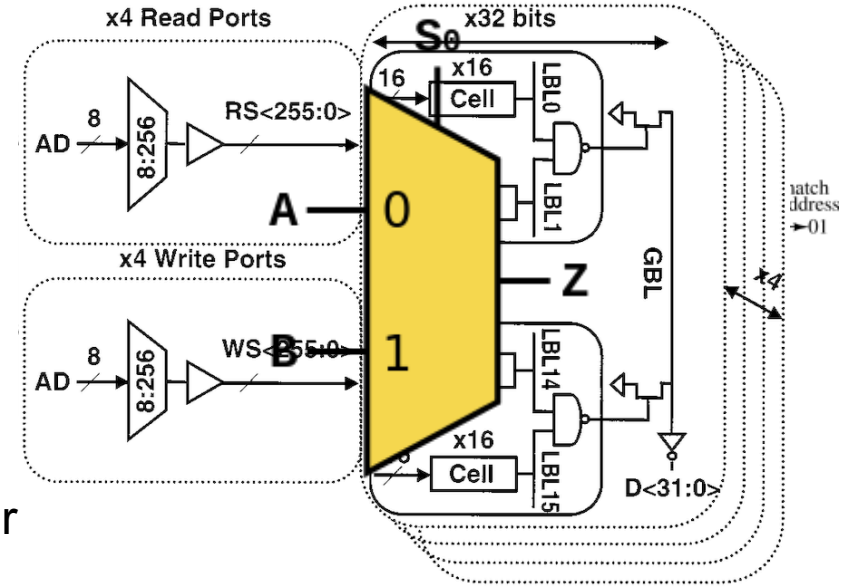
Two major challenges of FPGA simulation

- Labor-intensive
- Chip might not fit



Why the chip won’t fit

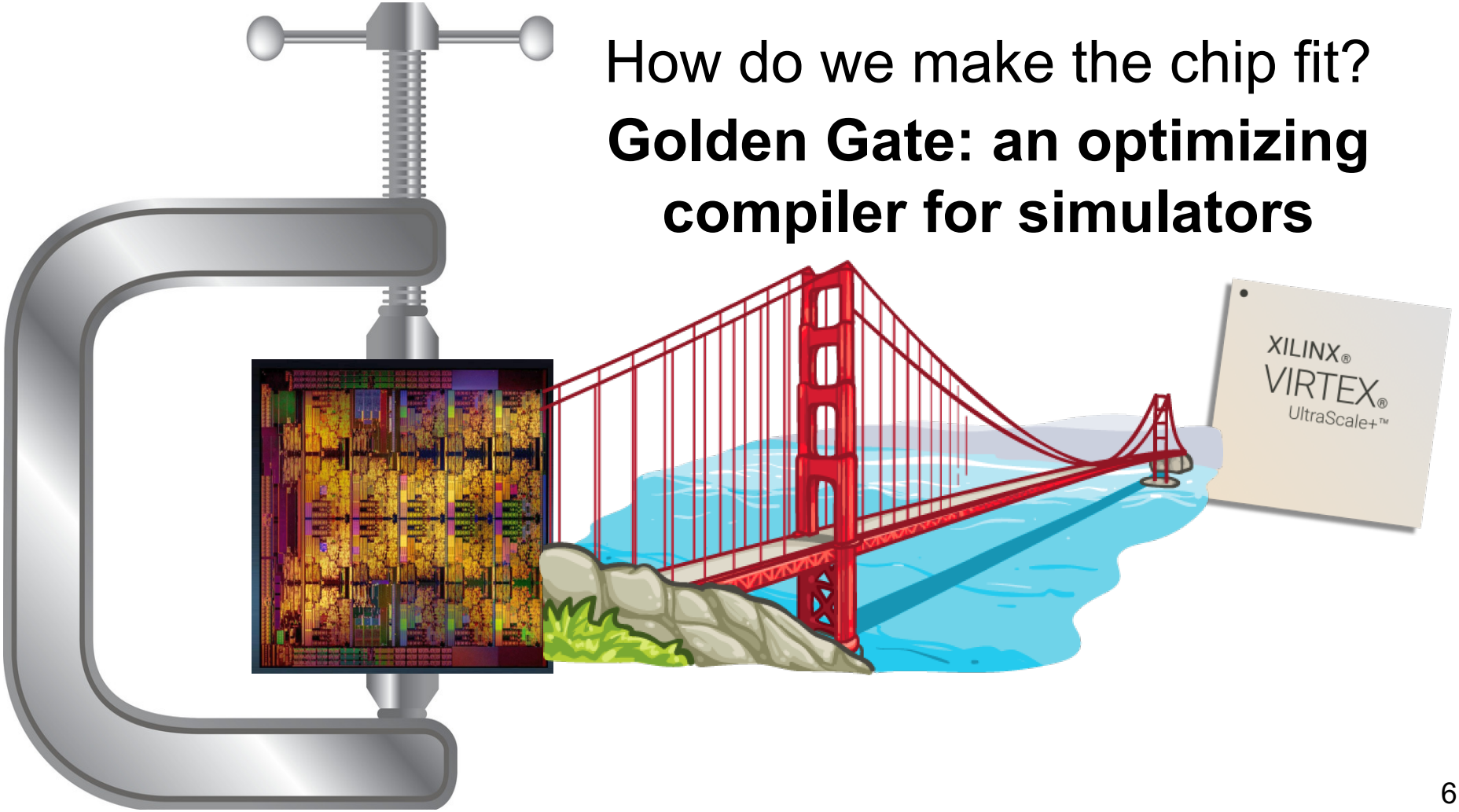
- Common ASIC structures map poorly
  - Highly-ported RAMs
  - Content-addressable memories
  - Multiplexers
- Abundant memory resources are under



Making the chip fit often means buying bigger FPGA!

How do we make the chip fit?

Golden Gate: an optimizing compiler for simulators



Golden Gate: a **hardware compiler** framework

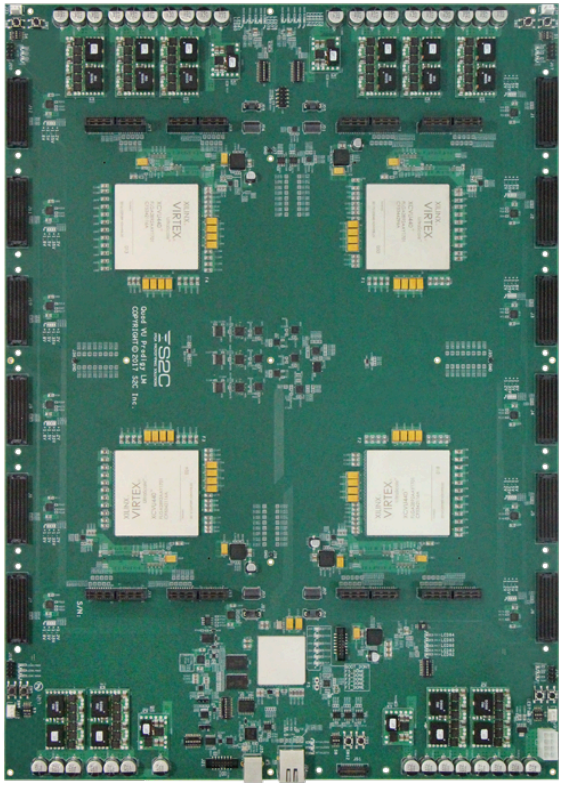
- Operating on **concrete RTL** target designs
- Producing cycle- and bit-exact FPGA simulators
- …structured as a network of **communicating actors**
- …relying on **decoupling** to ease per-cycle synchronization
- With a reusable API for FPGA-centric resource optimizations

With a basic optimization, we fit **50% more** out-of-order cores per FPGA!

- Introduction
- Prior work in increasing FPGA capacity
- Golden Gate: an optimizing compiler for FPGA simulators
- Case study: adding an optimization to Golden Gate
- Verification of complex simulation models
- Conclusion

- Introduction
- **Prior work in increasing FPGA capacity**
- Golden Gate: an optimizing compiler for FPGA simulators
- Case study: adding an optimization to Golden Gate
- Verification of complex simulation models
- Conclusion

Partitioning to solve capacity “cliffs”



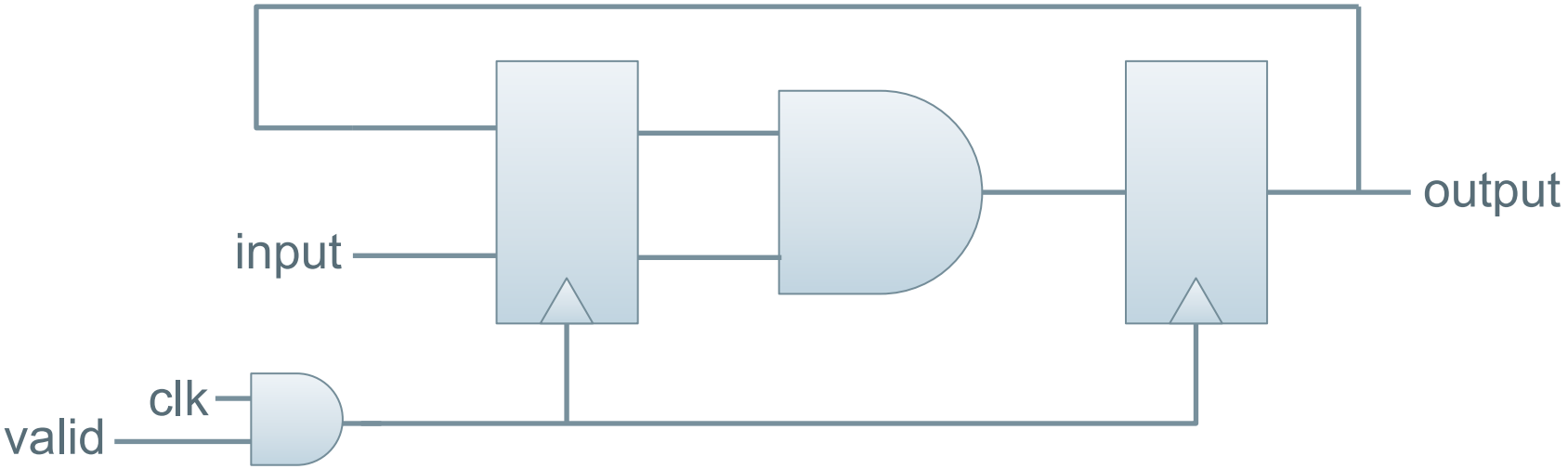
- Split design across multiple FPGAs
- Each FPGA is still under-utilized!
- Well put in HAsim†: “in order to maximize capacity of the multi-FPGA scenario we must first maximize utilization of an individual FPGA.”

† Pellauer et al., “HAsim: FPGA-Based High-Detail Multicore Simulation Using Time-Division Multiplexing” in HPCA 2011.

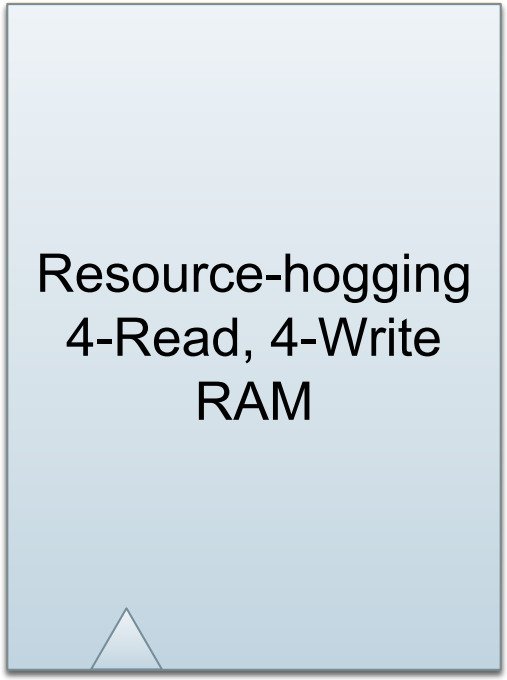
Decoupling

- **FPGA prototype:** one host FPGA clock = one simulated cycle
- **Decoupled simulator:** target and host time advance *independently*
  - Each target cycle may take multiple FPGA host cycles to simulate
- Software RTL simulators take this idea to the extreme

Clock gating: the simplest form of decoupling



You can save resources with decoupling



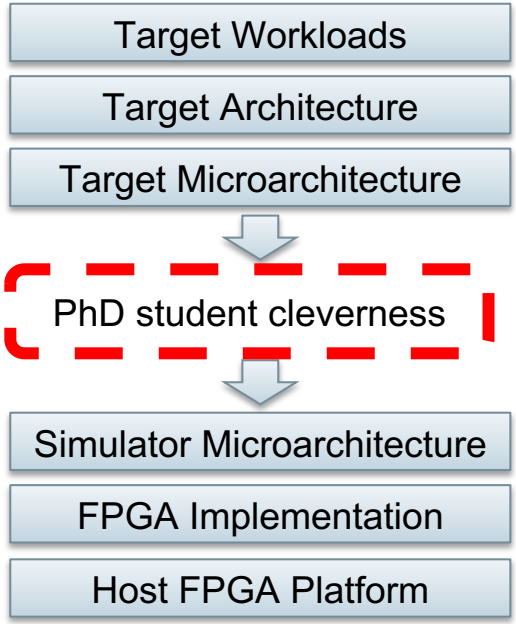
With 4 host cycles to simulate 1 target cycle ➡ trade space for time!

Tradeoff: it now takes 4 host cycles to simulate one target cycle, but we save FPGA resources

Decoupling enables optimizations that can significantly reduce utilization

No tools to apply them automatically

Where prior work falls short

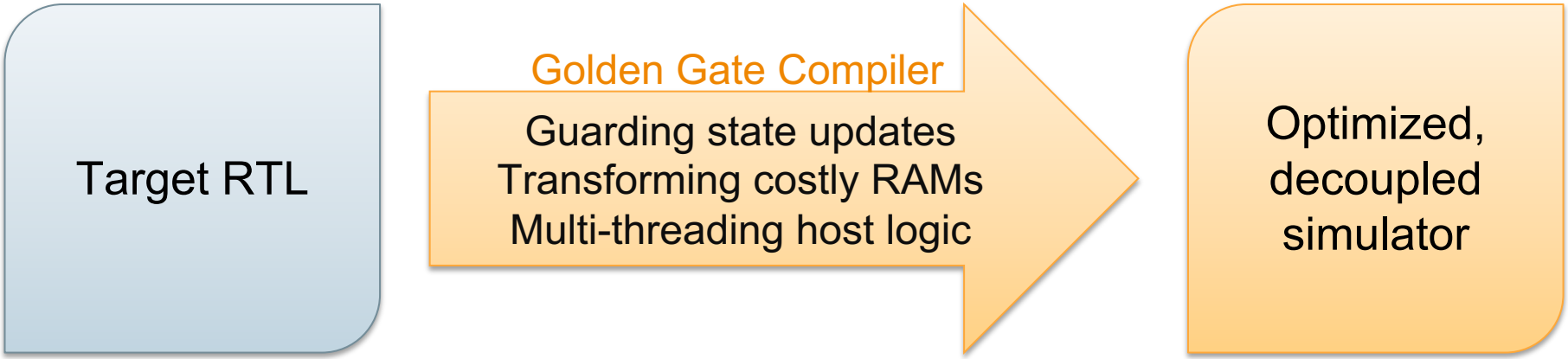


Conceptual simulation stack

- Paper idea: conceptual improvement in simulator architecture and/or microarchitecture arch
- Paper artifact: “artisanal” simulator based on idea
- Different goals: why write a compiler for RTL if most users don’t have working RTL to start with?

- Introduction
- Prior work in increasing FPGA capacity
- **Golden Gate: an optimizing compiler for FPGA simulators**
- Case study: adding an optimization to Golden Gate
- Verification of complex simulation models
- Conclusion

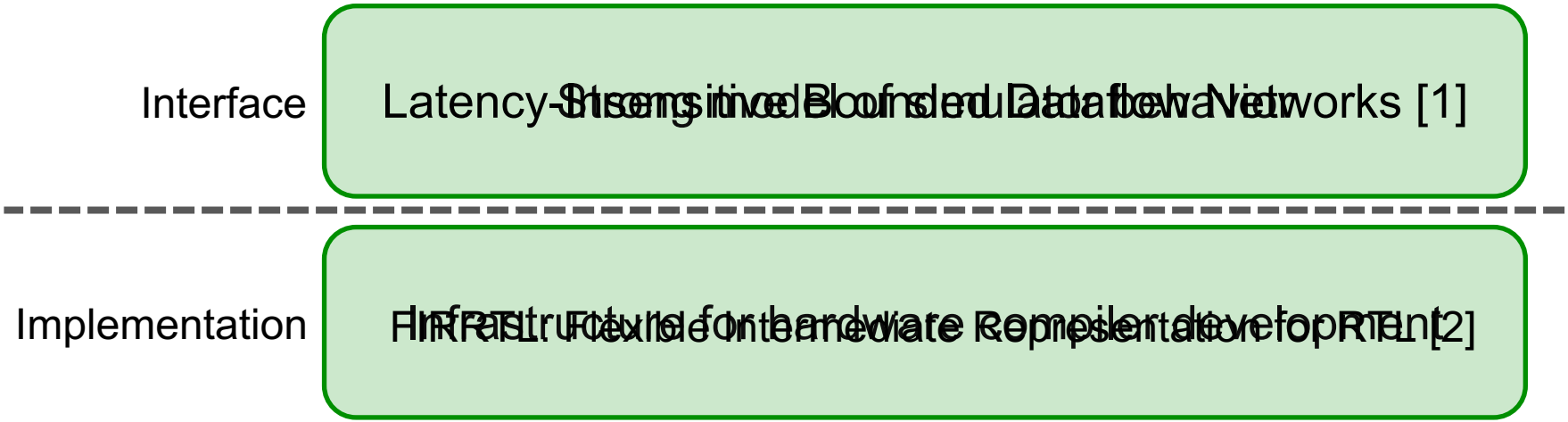
A compiler framework for FPGA simulators



These compiler passes are not RTL-preserving

- The generated simulator no longer implements the target's RTL semantics
- But it must simulate them in a cycle-exact manner!

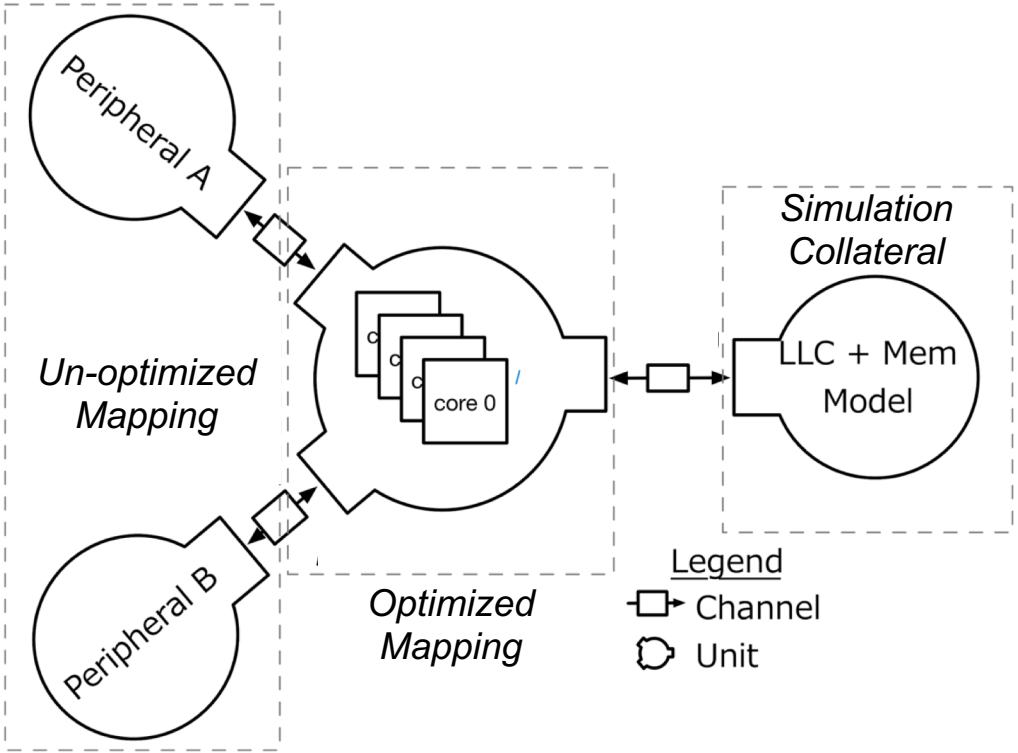
Building blocks for Golden Gate



[1] Vijayaraghavan et al., “Bounded Dataflow Networks and Latency-Insensitive Circuits,” MEMOCODE ‘09.

[2] Izraelevitz et al., “Reusability is FIRRTL Ground: Hardware Construction Languages, Compiler Frameworks, and Transformations,” ICCAD ’17.

Golden Gate models simulator as a dataflow network



Dividing target into multiple models enables composable optimizations!

Latency-Insensitive Bounded Dataflow Networks*

BDNs: Bounded Dataflow Networks

- General design technique to avoid synchronous design constraints
- Replace synchronously timed signals with decoupled ***channels***

Latency-Insensitive BDNs (LI-BDNs)

- Conform to a set of properties on both token values and the conditions under which tokens must be produced/accepted
- **As a simulator:** properties prescribe the behavior of tokens modeling inputs and outputs of components that are simulated.

* Vijayaraghavan et al., “Bounded Dataflow Networks and Latency-Insensitive Circuits,” MEMOCODE ‘09.

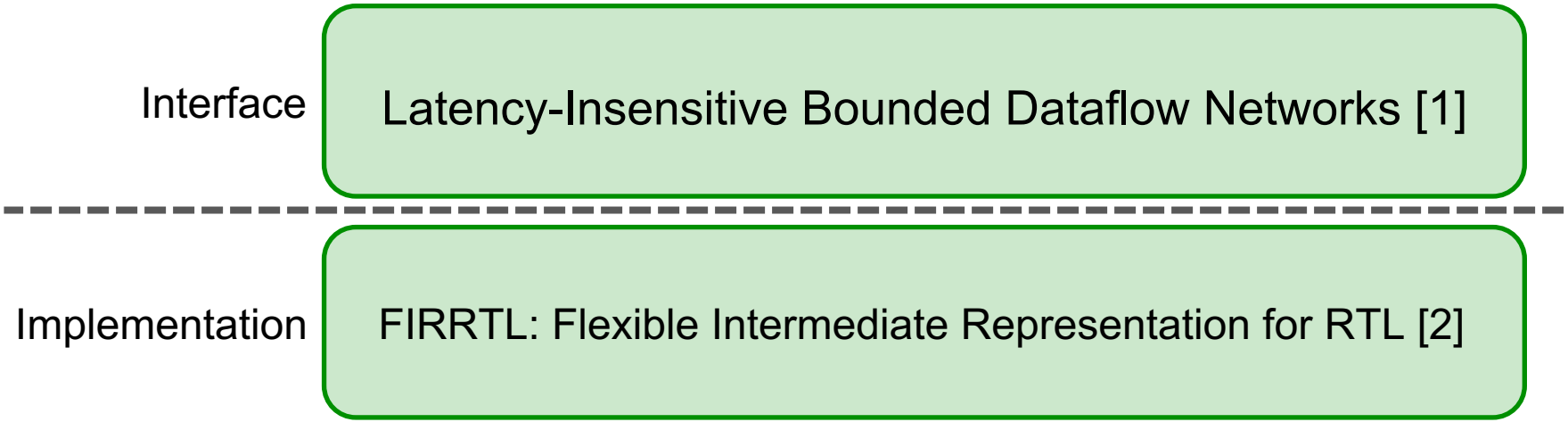
Compiler pass: RTL block to **unoptimized** LI-BDN

- Model the value of a given I/O on a particular cycle with a *token*
- Replace I/O with *token queues*
- Analyze netlist to find combinational I/O dependencies
- Transform RTL to a set of guarded atomic actions
  - Update target state when per-cycle synchronization is complete
  - I/O tokens are processed according to LI-BDN properties

LI-BDN structure guarantees freedom from deadlock and defines equivalence of two simulator components!

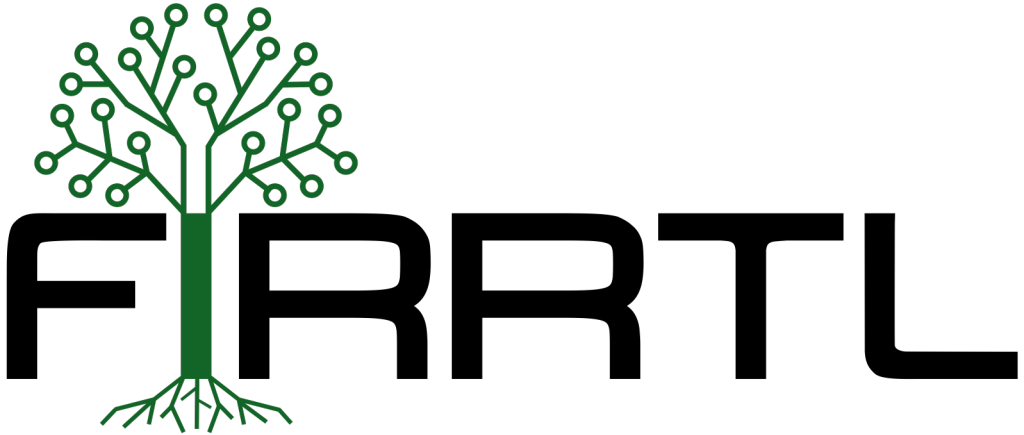
Helpful framework for inserting resource-optimized simulator components!

Building blocks for Golden Gate



[1] Vijayaraghavan et al., “Bounded Dataflow Networks and Latency-Insensitive Circuits,” MEMOCODE ‘09.
[2] Izraelevitz et al., “Reusability is FIRRTL Ground: Hardware Construction Languages, Compiler Frameworks, and Transformations,” ICCAD ’17.

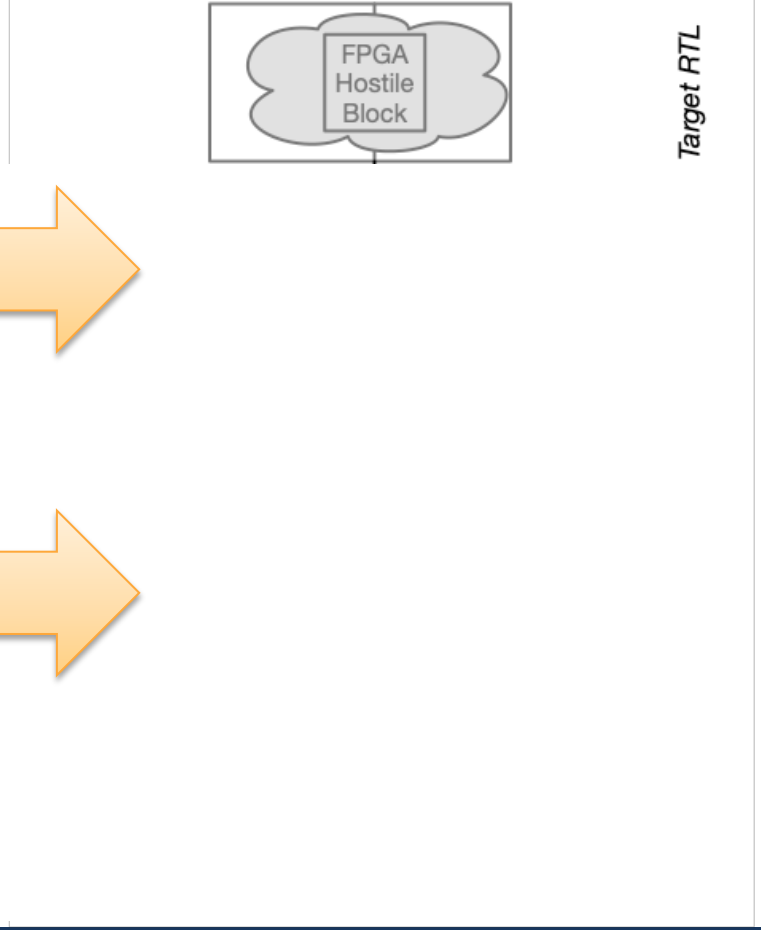
FIRRTL hardware compiler framework (ICCAD ‘17)



- Extensive suite of tools for writing hardware compiler passes
- Aimed at helping separate RTL from low-level implementation details

Makes writing CAD tools for chip design accessible to a wide audience!

Golden Gate is structured as an extensible compiler

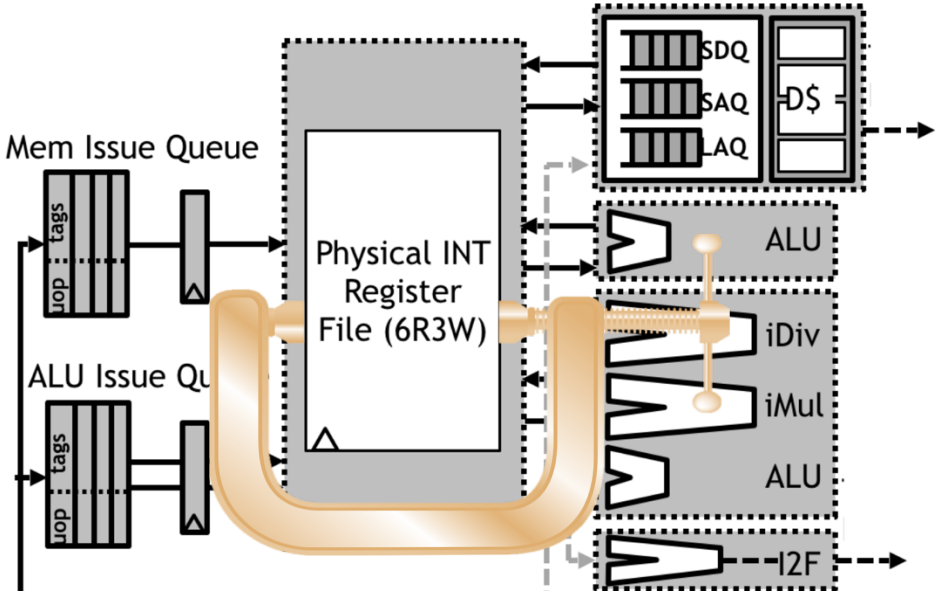


Sequence of FIRRTL passes

Optimizations fit in reusable framework

- Introduction
- Prior work in increasing FPGA capacity
- Golden Gate: an optimizing compiler for FPGA simulators
- **Case study: adding an optimization to Golden Gate**
- Verification of complex simulation models
- Conclusion

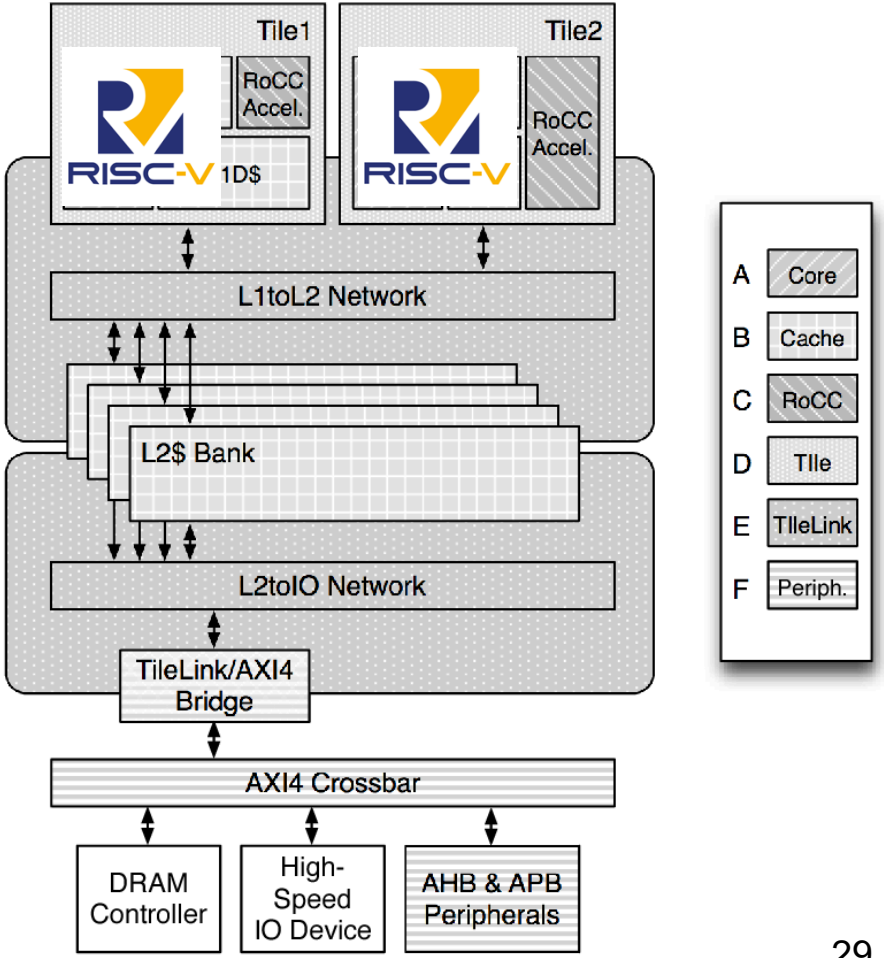
Case study: implementing an optimizing transform



Application: optimizing highly-ported register files in BOOM, an open-source RISC-V out-of-order core for the Rocket Chip Generator

The Rocket Chip Generator

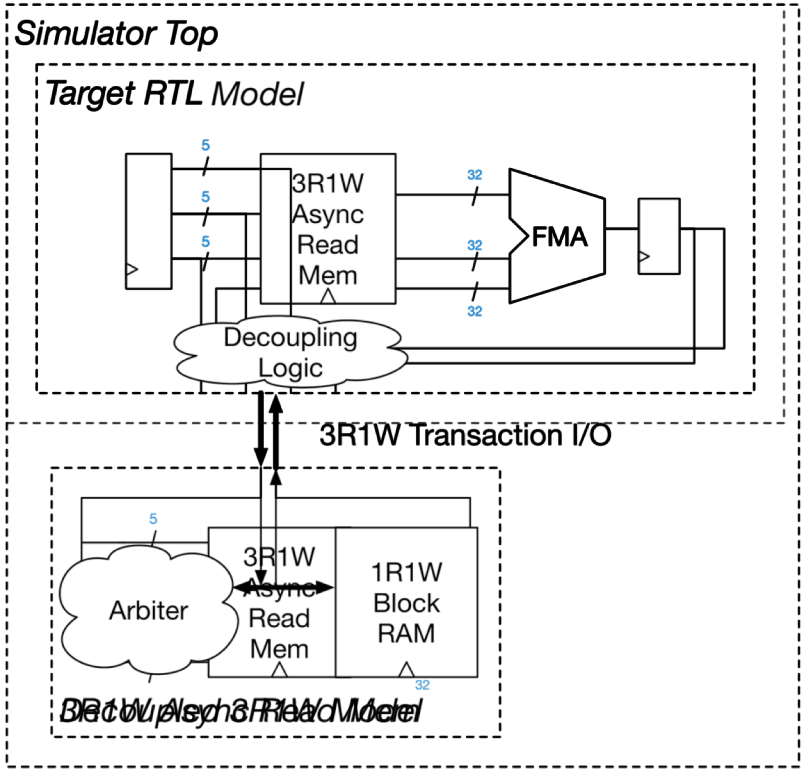
- Parameterizable SoC Generator [1]
- Cache-coherent TileLink network
- Variable number of cores
  - **Rocket:** 5-stage in-order
  - **BOOM:** parameterized out-of-order [2]



[1] Asanović et al., “The Rocket Chip Generator,” Berkeley Tech Report, 2016.
[2] Celio et al. “The Berkeley Out-of-Order Machine (BOOM): An Industry-Competitive, Synthesizable, Parameterized RISC-V Processor,” Berkeley Tech Report, 2015.

How the multi-ported memory optimization works

- Create multi-model simulator hierarchy
- Extract memory that is problematic for QoR
- Generate an FPGA-optimized memory model
  - Models exact target memory
  - Resource-efficient underlying BRAM
- Mapped independently from rest of circuit



How do we know this optimization works?

While FPGA simulation helps with pre-silicon verification, it brings new challenges.

A **functional** bug in the simulator can manifest as:

- An apparent **functional** bug in the target
- A **timing** irregularity in the target
- **Nondeterminism** of execution or **host deadlock**

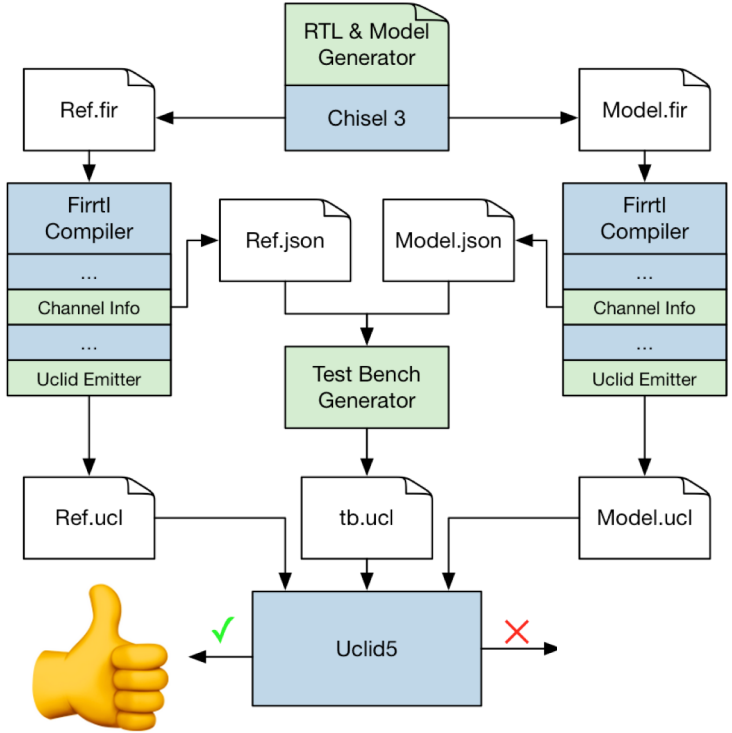
LIME: Automatic checking of decoupled models

LIME: Automatic checking of decoupled models

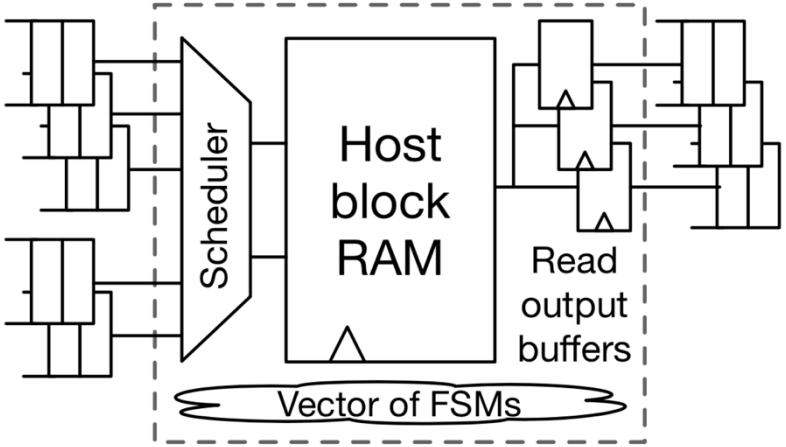
- Checks LI-BDN properties with BMC
- Ensures model is cycle-accurate
- Targets UCLID5 modeling system
- Used to verify multi-port RAM model

Inputs: reference RTL & model RTL

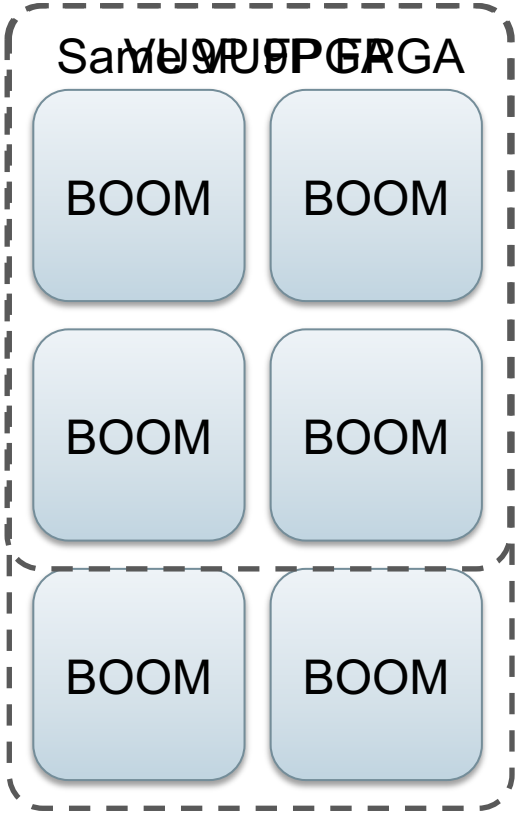
Output: counterexample waveforms (if any)



Results of optimizing register files



Underlying 1R1W implementation maps efficiently to FPGA block RAMS (BRAMs)

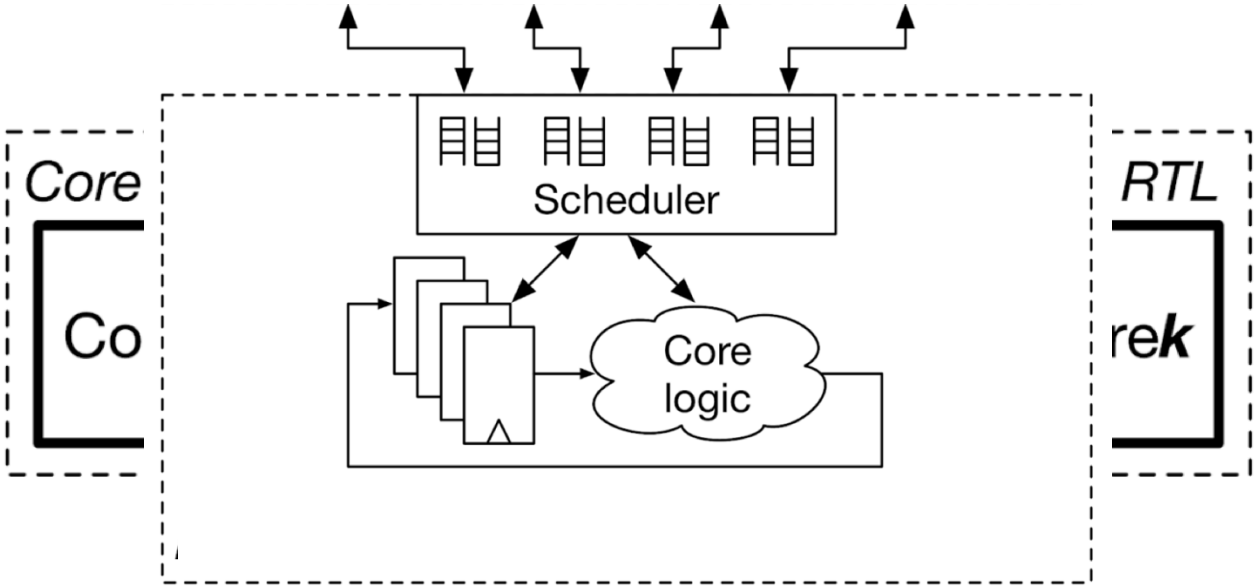


4 cores ➡6 cores

Results of optimizing register files

- **33% less LUT utilization per core**
- **Ample slack in BRAM count**

Future work: multi-threading to save resources



Why not borrow from software simulators and time-multiplex one copy of host logic to simulate N copies of a target block?

[1] Z. Tan et al, "RAMP Gold : A High-Throughput FPGA-Based Manycore Simulator," DAC '10.
[2] M. Pellauer et al., "HASim: FPGA-Based High-Detail Multicore Simulation Using Time-Division Multiplexing," HPCA '10.
[3] Z. Tan et al., "A Case for FAME: FPGA Architecture Model Execution," ISCA '10.

- Introduction
- Prior work in increasing FPGA capacity
- Golden Gate: an optimizing compiler for FPGA simulators
- Case study: adding an optimization to Golden Gate
- Verification of complex simulation models
- **Conclusion**

Conclusion

- We present Golden Gate, a compiler framework for FPGA simulators
- It includes a spec for simulators structured as dataflow networks
- We provide an API for heterogenous compiler passes
- Golden Gate open-sourced as part of FireSim @ https://fires.im

As a case study, we present a multi-cycle RAM optimization that significantly increases simulation capacity of a large Xilinx FPGA!