GreenRio: A Modern RISC-V Microprocessor Completely Designed with An Open-source EDA Flow

Yifei Zhu, Guohua Yin, Xinze Wang, Qiaowen Yang,
Zhengxuan Luan, Yihai Zhang, Minzi Wang, Peichen Guo,
Xinlai Wan, Shenwei Hu, Dongyu Zhang, Yucheng Wang,
Wei Wei Chen, Lei Ren, Zhangxi Tan
RIOS Lab, Tsinghua-Berkeley Shenzhen Institute, Tsinghua University
2022.11.03

Outline

- Background
- Overview
- Frontend Flow
- Backend Flow
- Experience & Feedback on Open-source Tools
- Future work

Background

- Open source EDA tools are evolving at an amazing rate, but community lacks complex designs for references
- We want to design an iterative complex and advanced design to stress test Open source EDA
- Here comes GreenRio！
  - GreenRio 1.0: 2022.09 (released)
  - GreenRio 2.0: 2022.11 (work in progress)
  - …

Background

- Comparison of some typical cores hardened by Open EDA

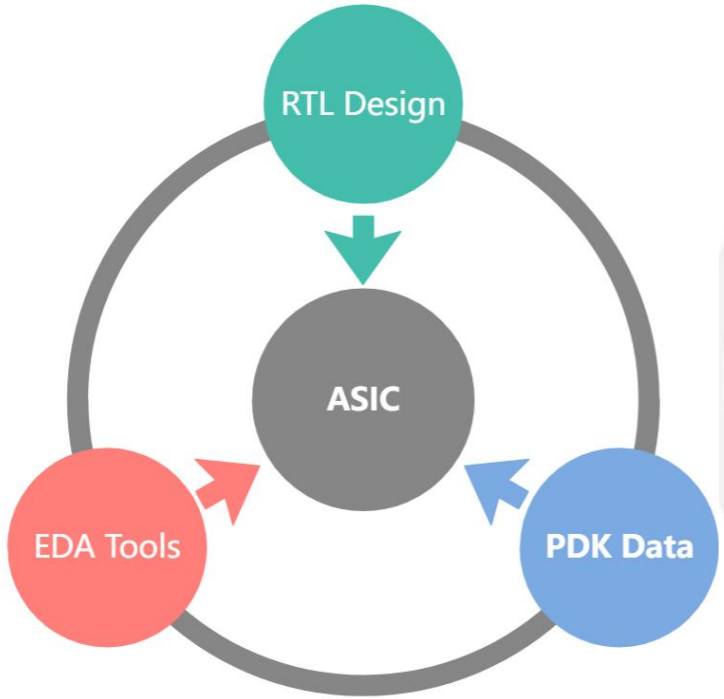
Design	GreenRio	EH1	biriscv	picorv32a	ibex
ISA	RV64	RV32	RV32	RV32	RV32
pipeline stage	7	9	6 or 7	6	2 or 3
issue width	dual	dual	dual	single	single
execution feature	out-of-order	in-order	in-order	in-order	in-order
gate count (K)	53-120*	100	67	17	20
Efabless tape-out	√	×	×	√	√

- Compared to other **OpenMPW & chipIgnite** designs(600+), GreenRio is one of the **most** advanced microarchitectures

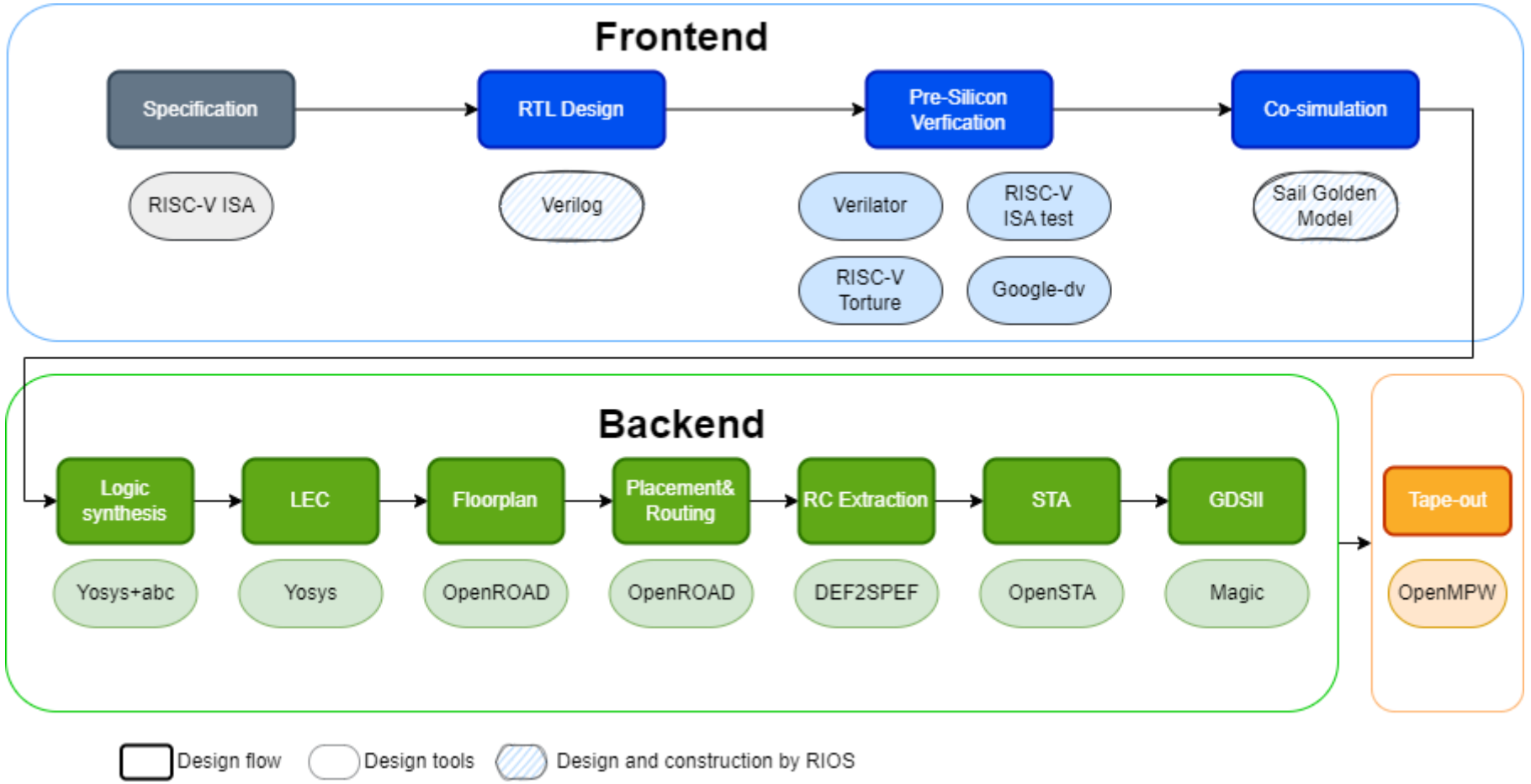
*: depending on configurations for instruction buffer, ROB buffer, etc.

Open Silicon Implementation Flow

- Open-source soft core IP & designs
  - Rios series CPU core: GreenRio
- Open-source hard core IP
  - SRAM blocks compiled by OpenRAM
- Open-source backend EDA tools
  - OpenLane
- Open-source PDK
  - Skywater 130nm
- Open-source silicon production
  - Open Multi Project Wafer (OpenMPW)
  - Efabless Chipignite



Overview



GreenRio Open-Silicon Flow

Frontend Flow

Design, Simulation, and Verification

Choose a Development Language：System Verilog versus Verilog

- The pass rate of open-source tools for some **SV-based** cores

	preprocessor		open synthesis tool	open verification tool
test cores*	Surelog*-UHDM*	sv2v*	Yosys	Verilator
ariane	0/1	0/1	0/1	1/1
black-parrot	0/7	0/7	0/7	4/7
earlgrey	0/1	0/1	0/1	0/1
fx68k	0/1	1/1	0/1	1/1
ibex	1/1	1/1	0/1	1/1
rsd	0/1	1/1	0/1	1/1
scr1	0/1	1/1	0/1	0/1
swerv	0/1	1/1	0/1	1/1
TNoC	0/1	0/1	0/1	0/1
picorio1.0	0/1	0/1	0/1	0/1
total tests pass	1/16	5/16	0/16	9/16

We chose Verilog for compatibility

From https://chipsalliance.github.io/sv-tests-results/

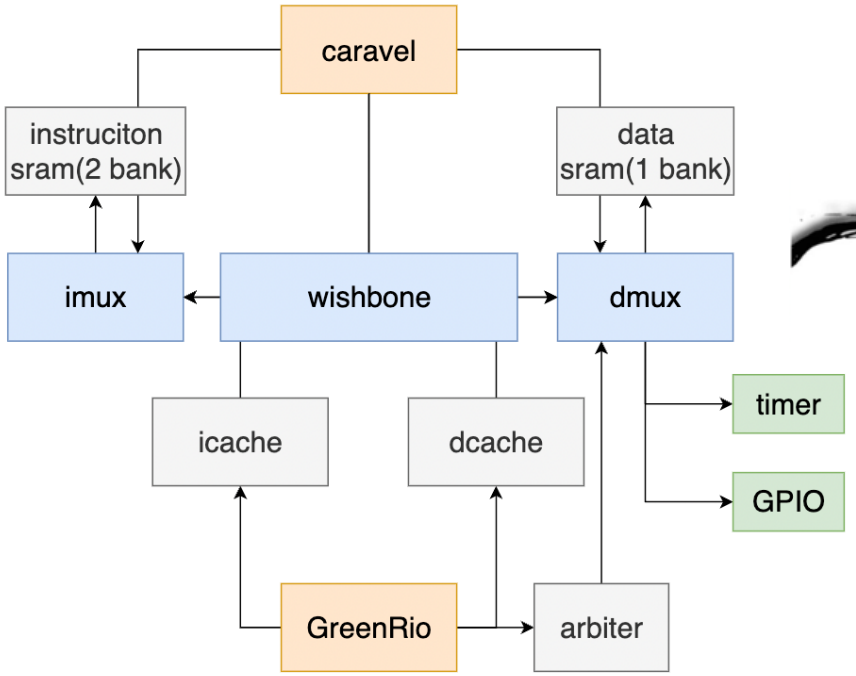
*test cores: some open cores which are written in System Verilog

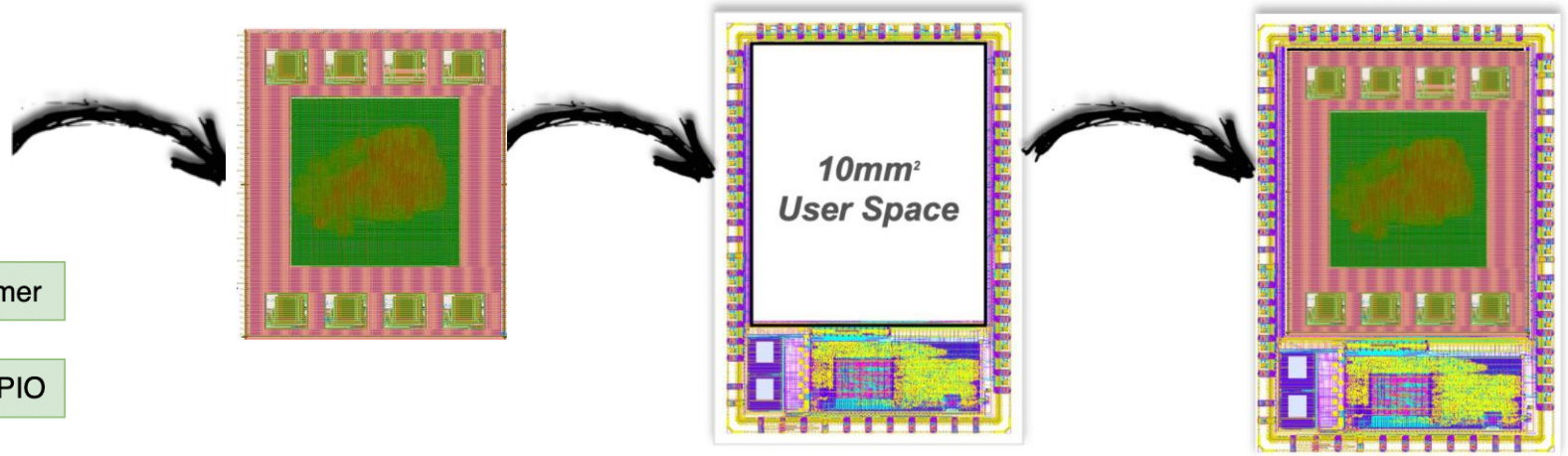
*Surelog: A frontend that can convert System Verilog to UHDM file

*UHDM (Universal Hardware Data Model) : A tool that can convert UHDM files to Verilog

*Sv2v: A tool that can convert System Verilog to Verilog

SoC Design





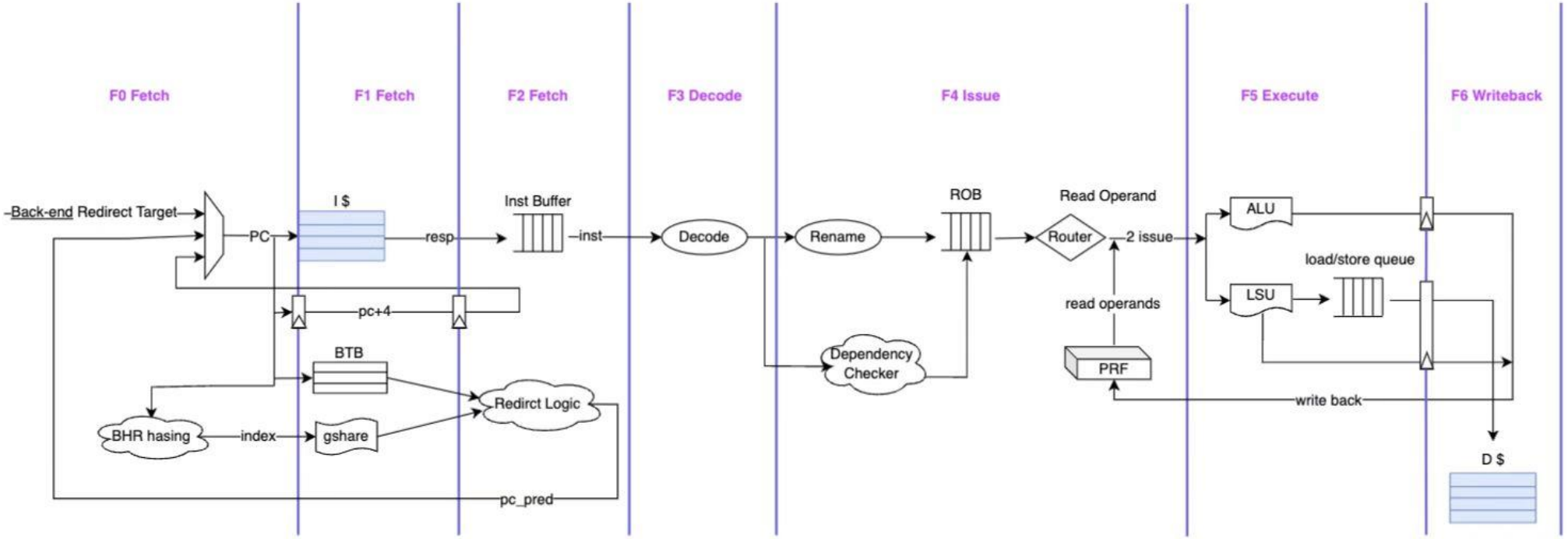
OUR DESIGN

CARAVEL

INTEGRATED CHIP

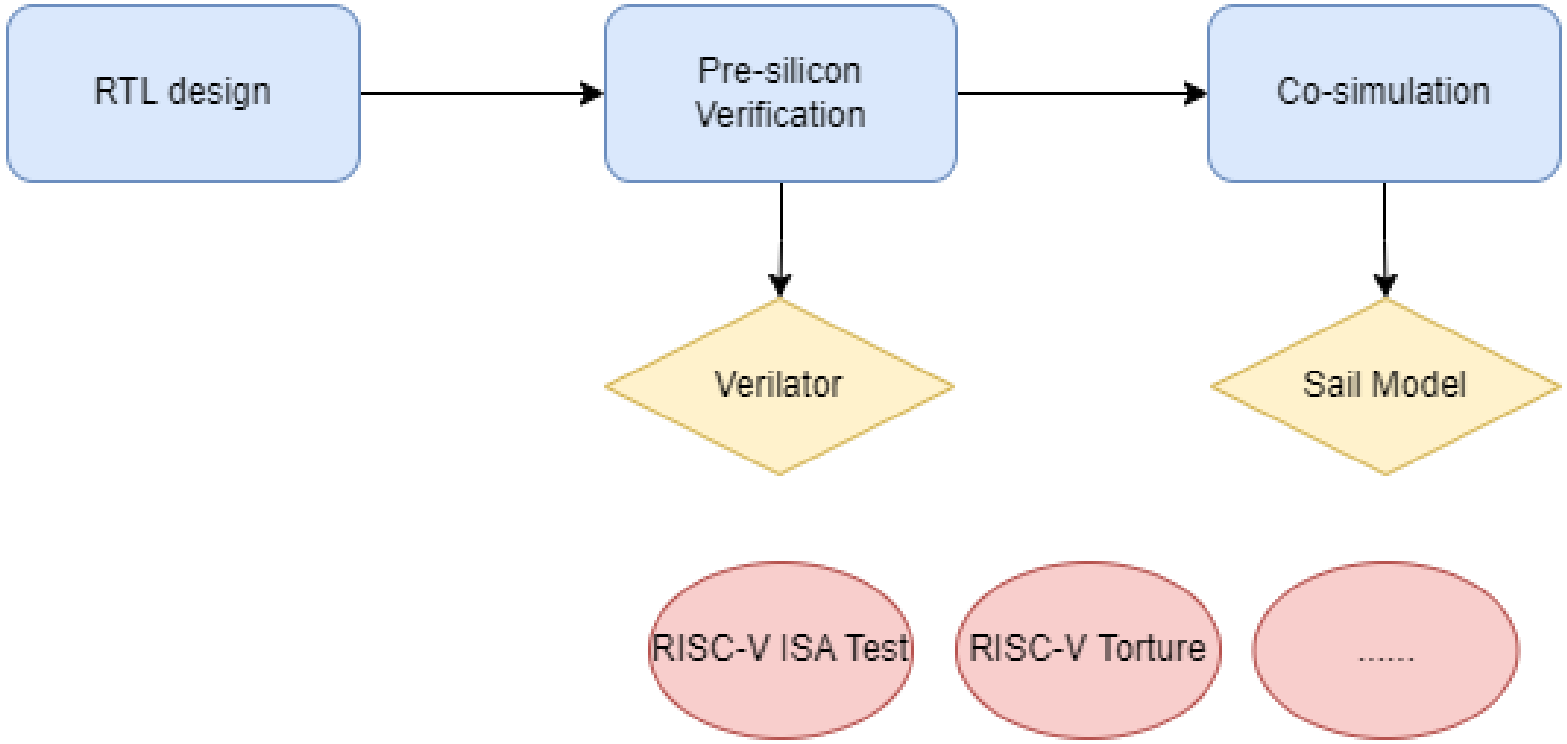
Microarchitecture Design

- GreenRio1.0
  - 7-stage RISC-V 64-bit core
  - Frontend stage : F0-F3　　Backend stage:F4-F6



Core Verification

- Pre-silicon Verification: Use Verilator for basic functional verification
- Co-simulation: Use Sail for transaction level verification

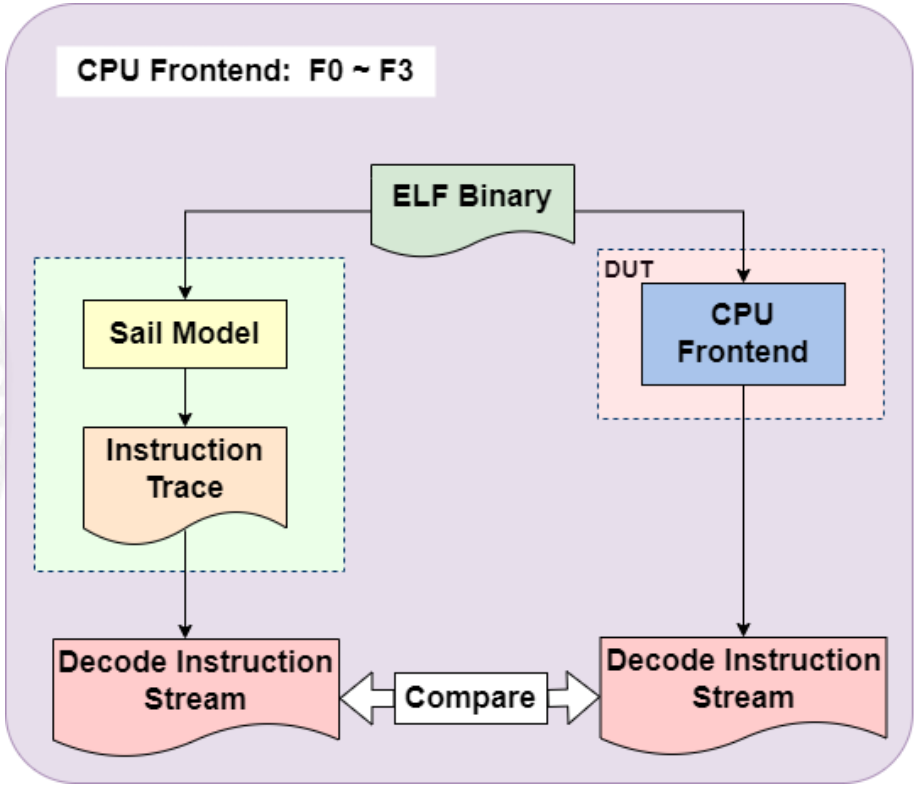


What & Why Sail?

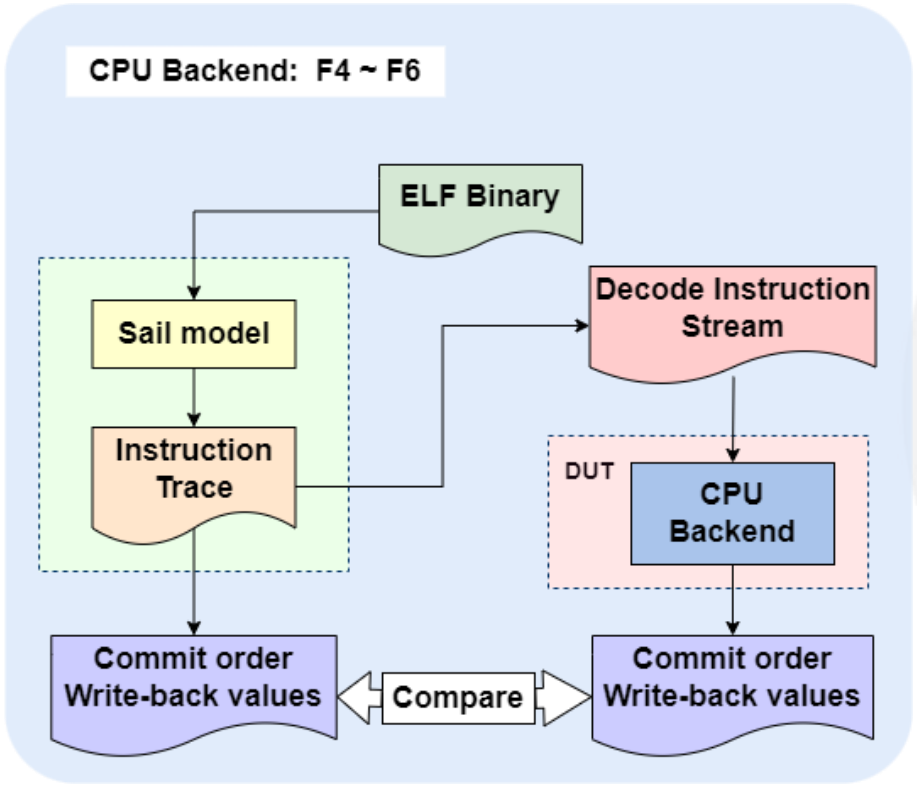
- Golden architectural model for instruction semantics
- Directly compile spec into golden model
  - Reduces possibility of human error
- Acceptable transaction level verification
- Slow C model generation, but acceptable
- Sail reduces the possibility of human error and has some acceptable drawbacks

Co-simulation Flow

- Frontend co-sim



- Backend co-sim



Backend Flow

Experience, Analysis and Feedback

Experience: A Journey of Discovery

- First tape-out project
- Using only Open EDA tools
- A series of trials and tribulations
- 5+ solutions tried within 10 days

1st Attempt: Flatten Everything

- Let OpenLane do everything!
  - Fixed area (2920x3520nm)
  - 8 SRAM blocks, GreenRio core, and SoC
  - Automatic floorplan, placement, and routing ….

Result

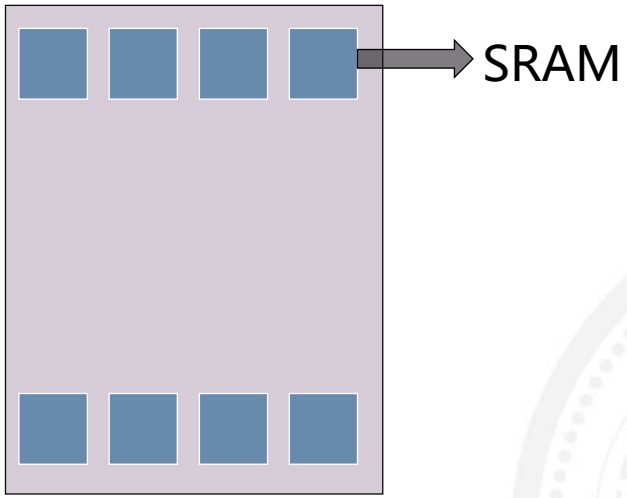
- Placement step failed

Discussion

- Robustness of algorithms?
  - Many big macros
  - *Automatic floorplan in complex scenarios*
- Needs artificial support

2nd Attempt: Manual Macro Placement

- Manual Floorplan
  - Specify the positions of SRAMs
  - EDA handles everything else



Result

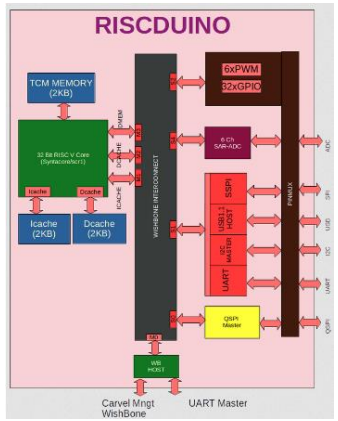
- Routing takes too long:
  - More than 10 hours
- SIGINT signal sent

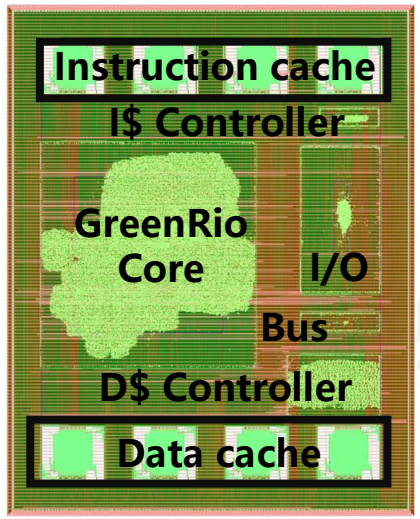
Discussion

- Routing algorithm efficiency?
  - *Big solution space*
- Auto placement optimizations?
  - Difficulties may be caused by inappropriate placement

3rd Attempt: Learning from Riscduino

- Harden hierarchical design
  - Take riscduino as a reference
    - A picorv32a-based SoC
  - Divide submodules by function
  - Harden submodules
  - Harden the full chip with macros





Result

- Flow finished
  - Completed within 5 hours
- DRC errors
  - Errors found by Magic & KLayout

Discussion

- Millions of SRAM DRC errors
  - **Can be neglected** *(from official)*
- Routing space too small?
  - More than 2000 wires between macros
  - Only ~3 mm² blank space
  - GreenRio is much larger than picorv32a

4th Attempt: Improvement of the 3rd One

- Adjustments based on Attempt 3
  - Reduce function blocks: integrate the controllers with SRAMs
  - Bypass SRAMs for DRC check

Result

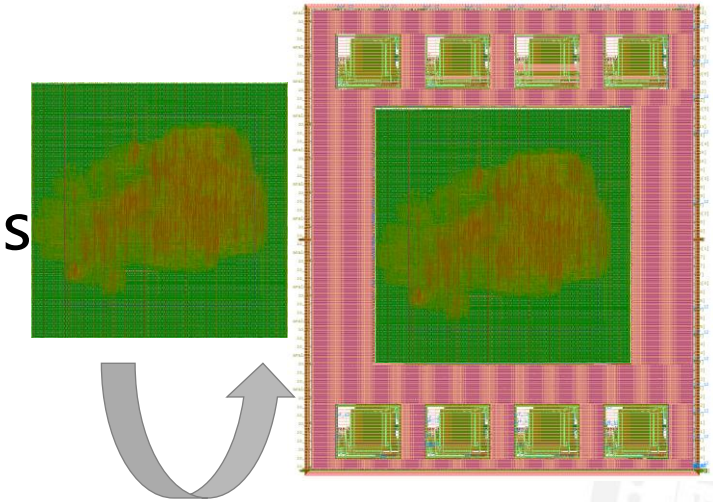
- DRC errors
  - Errors found by Magic & KLayout
- Re-floorplan and iterative testing
  - Number of DRC errors changed
  - Fewer errors (<100)

Discussion

- routing space still limited?
  - Still many macros: 11
  - Still many macro-macro wires
    - More than 1500 wires

Final Attempt: Learning from Failures

- Flatten and hierarchy
  - Flatten all modules except SRAMs
  - Integration: flattened modules and SRAMs



Result

- Flow finished in 4.3 hours
- No DRC or LVS errors found
- Passed precheck & tape-out check

Discussion

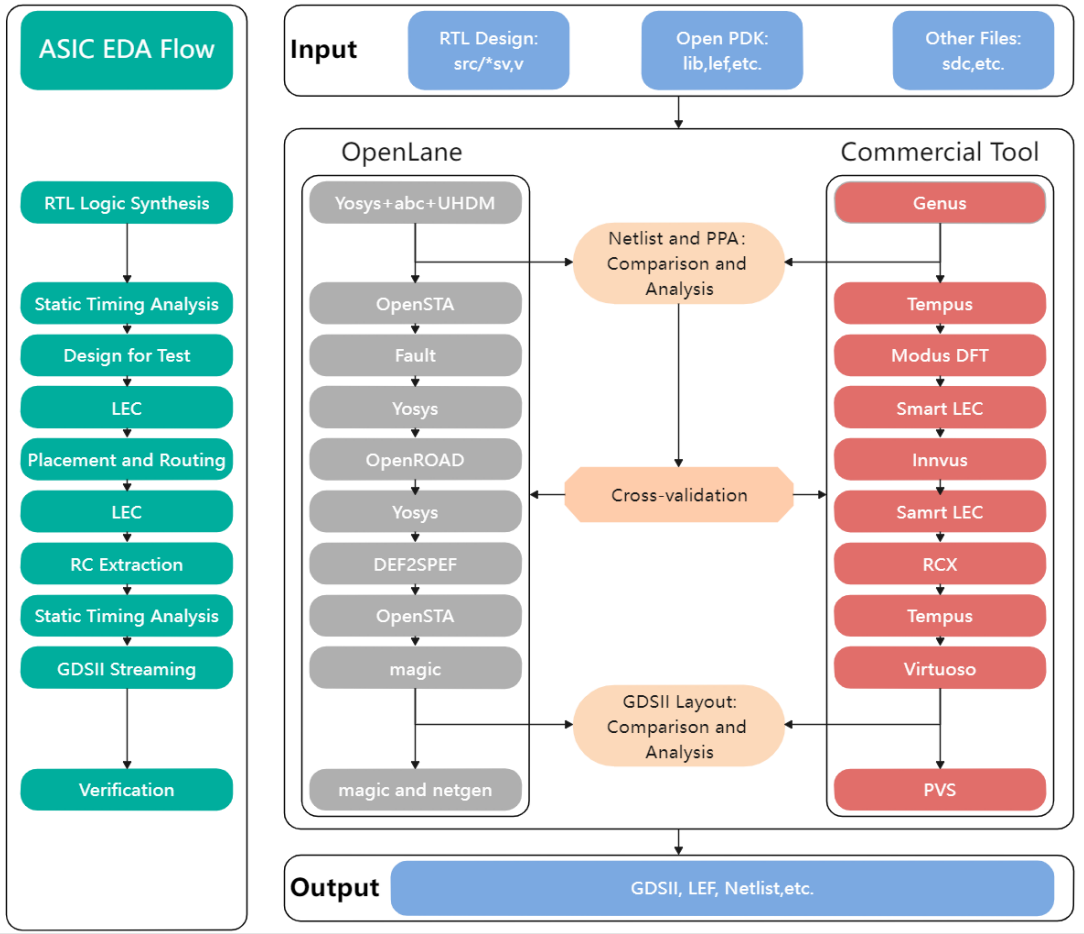
- Reduce macro-macro wires
  - Only 1000 wires between macros
  - 4.5 mm² blank area
- Solution range of the routing algorithm

Analysis: Where Did This Flow Fail Before?

- What needs to be done manually
  - Proper Marco partition
  - Reasonable Floorplan
- Expectations and reflections on tools
  - Automatic floorplan for complex designs
  - Optimized placement
  - Routing algorithm efficiency within large solution space
  - DRC-avoidant routing algorithm

Comparison: Open vs Proprietary

OpenLane versus Cadence EDA flow for hardening SoC without SRAMs



Comparison of Key Indicators

- The following data are obtained based on the same clock frequency (80MHz) and timing corner (tt_025C_1v80) without timing violations

	OpenLane	Proprietary EDA	Gap*1
synthesis run time	6m12s	4m4s	1.5X
gate count after synthesis	53 K	33 K	1.6X
placement & routing time*2	1h58m	43m	2.7X
die area (mm²)	2.02	1.24	1.6X
placement density	32%	45%	1.4X
dynamic power	48.6mW	23.1mW	2.1X
best clock period*3	80MHz	110MHz	1.4X

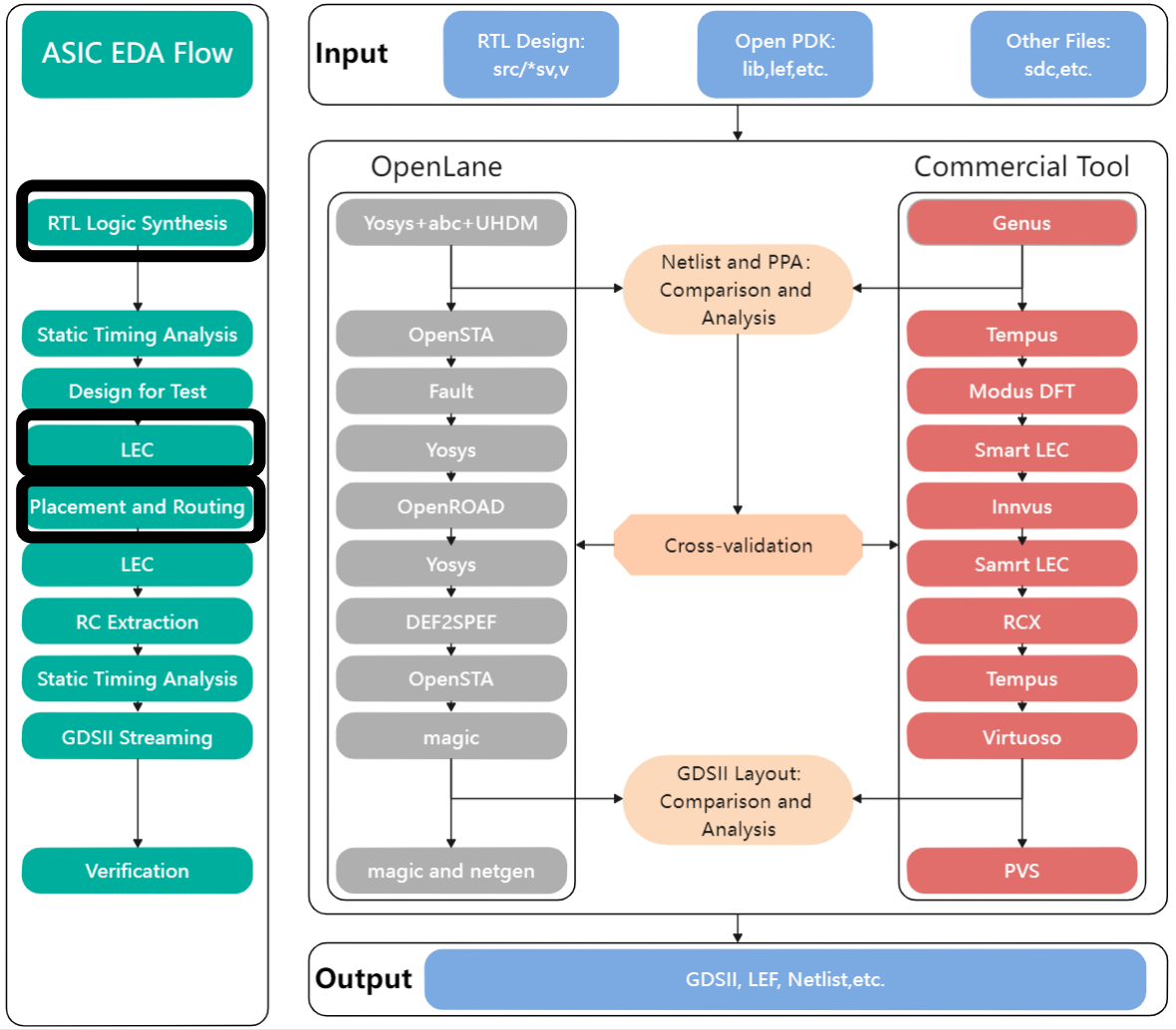
*1: Represents the gap between open and proprietary tools. Using the proprietary tool as a benchmark

*2: The number of cores for routing is 16 (both OpenLane and proprietary EDA)

*3: No setup or hold violations

Feedback on Open Source Tools

- **What is missing?**
  - Complete support for SV
    - Support more open designs
  - Available LEC
    - LEC function is abnormal
  - Efficient PnR algorithms
    - Higher density
    - Larger routing scale
- **What to expect?**
  - Optimize PPA
  - Multi-threading



Future Work

GreenRio IP

GreenRio 1.0: 2022.09 (released)

dual-issue, out-of-order

GreenRio 2.0: 2022.11 (WIP)

Multi-core coherence, Linux-compatible

GreenRio 3.0: 2023.04 (WIP)

GPU, Chrome OS, H-mode

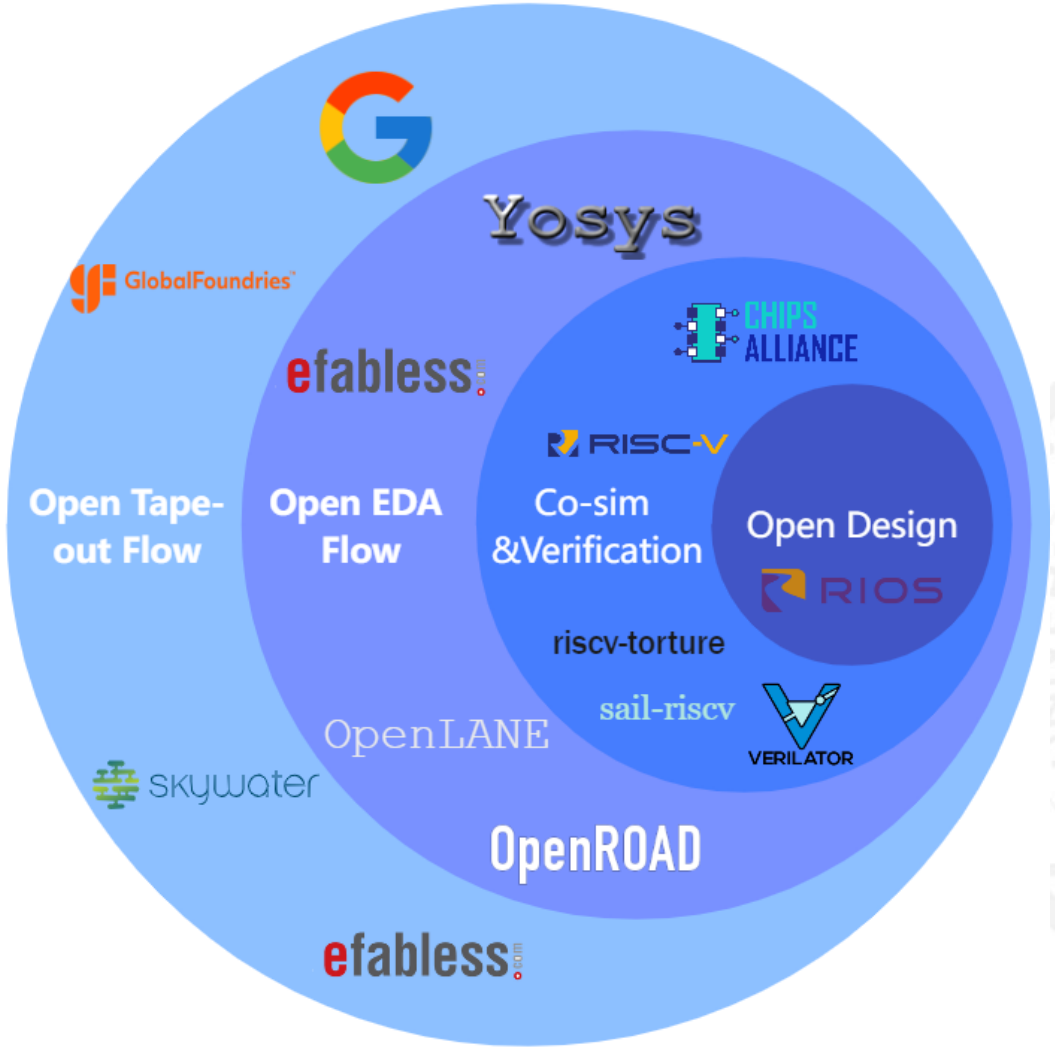
Open Source Silicon Flow

IR analysis tool design and optimization

Smarter PnR algorithms

AI-based PPA optimization

Build ecological environment



Acknowledgements

- This project would not have been possible without

Johan Euphrosine, Google
Tim Ansell, Google
Clifton W, RiVAI

RIOS