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, Mingzi Wang, Peichen Guo,

Xinlai Wan, Shenwei Hu, Dongyu Zhang, Yucheng Wang,

Wei Chen, Lei Ren, Zhangxi Tan

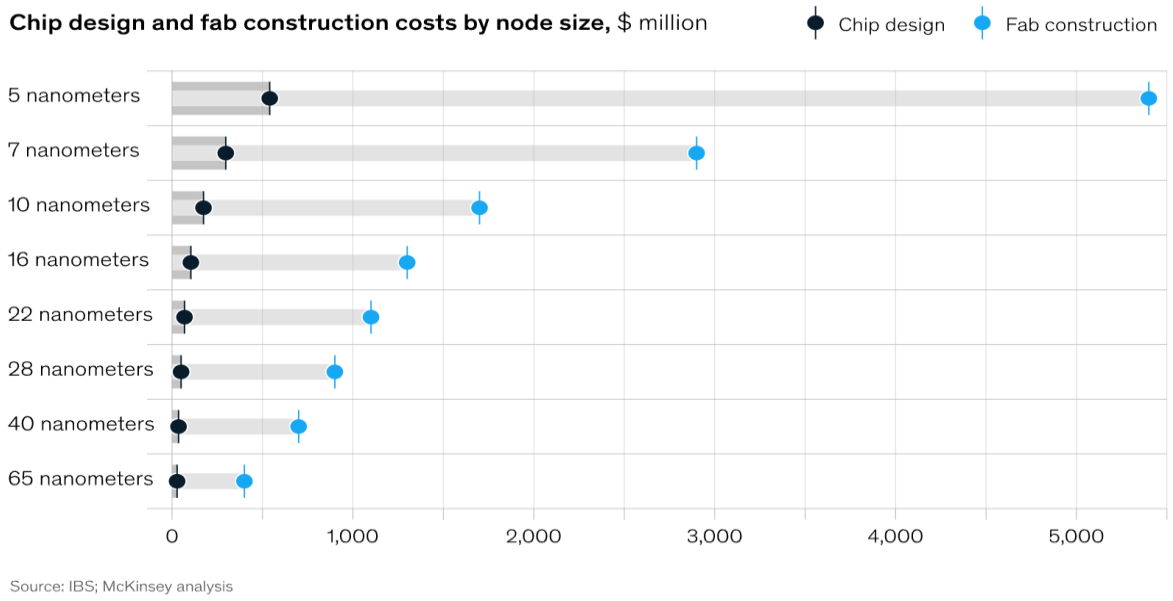
RISC-V International open-source Laboratory, Tsinghua-Berkeley Shenzhen Institute (RIOS Lab, TBSI)

Outline

- Background
- GreenRio: Front-end Design
- GreenRio: Back-end Design
- Comparison between Proprietary and Open EDA Tools
- Outlook on Future Open-source RISC-V Design

Background

- Increasing complexity of IC design in advanced processing technologies



Pains cost, difficulties, risks, policies, ...

Expectations

1. Bridging the knowledge gap between classroom and industry
2. Promote RISC-V and OpenEDA's echo system

→ Threshold for Open CPU design

→ Agile development in fully open source mode

- Open source EDA tools are evolving but the community lacks designs **for references**

OpenEda's iteration

RISC-V chip's development

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	√	×	×	√	√

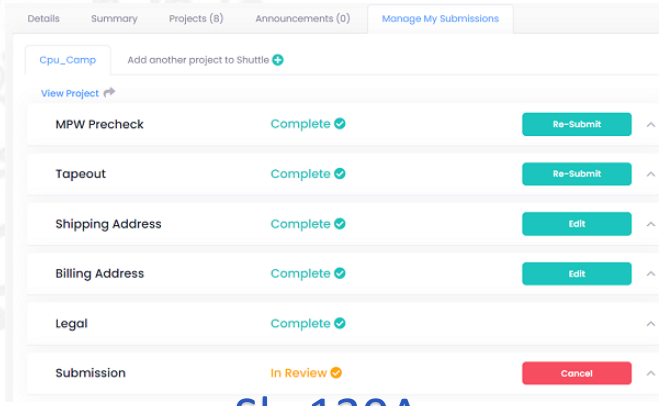
Motivations

Can the open toolchains be used to develop a modern RISC-V core?

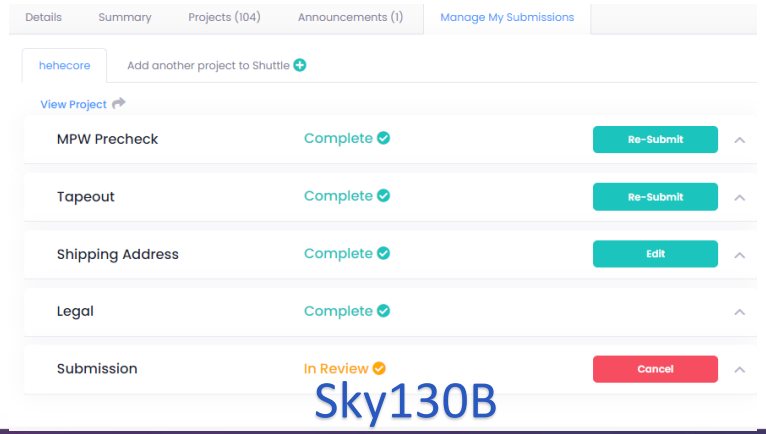
Limitations

1. Gate counts, area
2. Features of modern processors

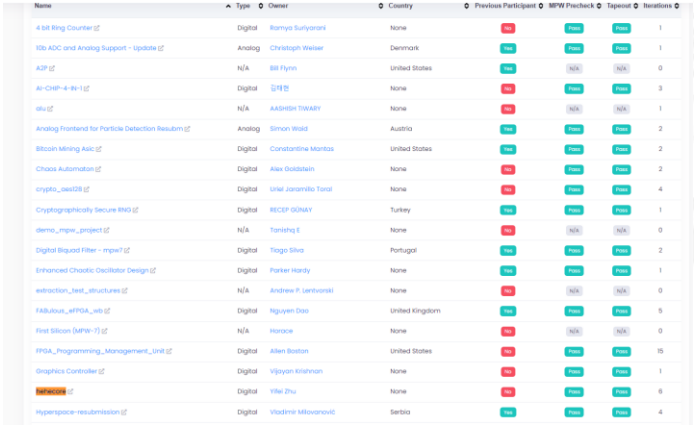
- Taped out in the Chipignite and OpenMPW programs



Sky130A

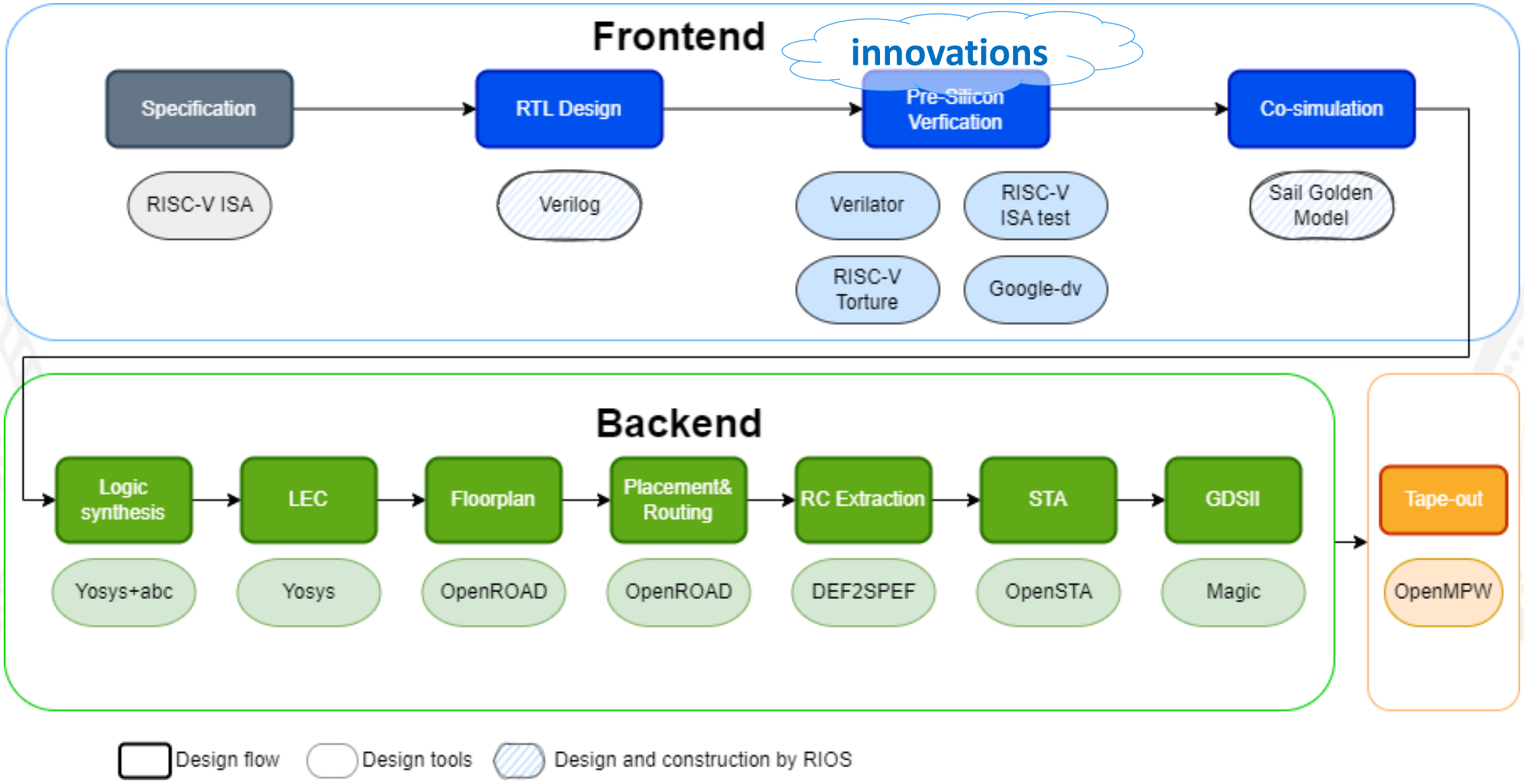


Sky130B



Overview of Our work

whole signoff flow



Frontend Flow

Specification, Simulation, and Verification

Choose a Development Language:

System Verilog versus Verilog

- The translation 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

- Formal verification tools like "*Conformal*"
  - open ecological chain of chip design.

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

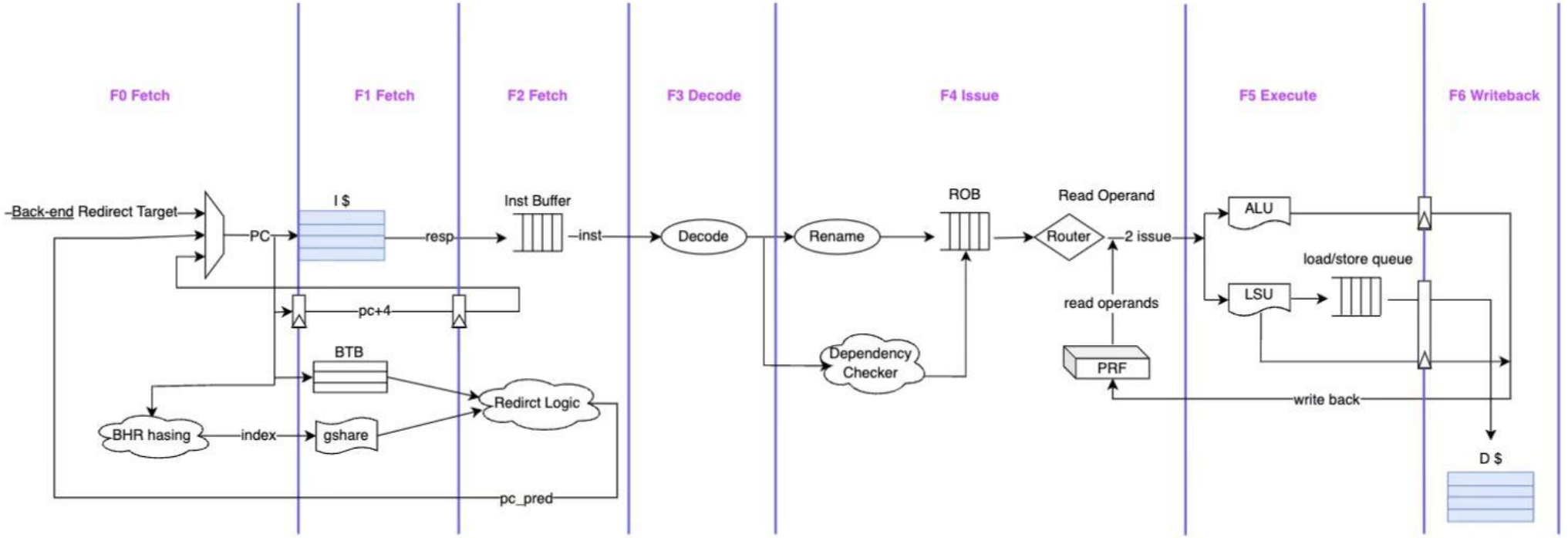
*UHDM (Universal Hardware Data Model) : A tool that can convert UHDM files to Verilog

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

GreenRio1.0

7-stage RISC-V 64-bit core

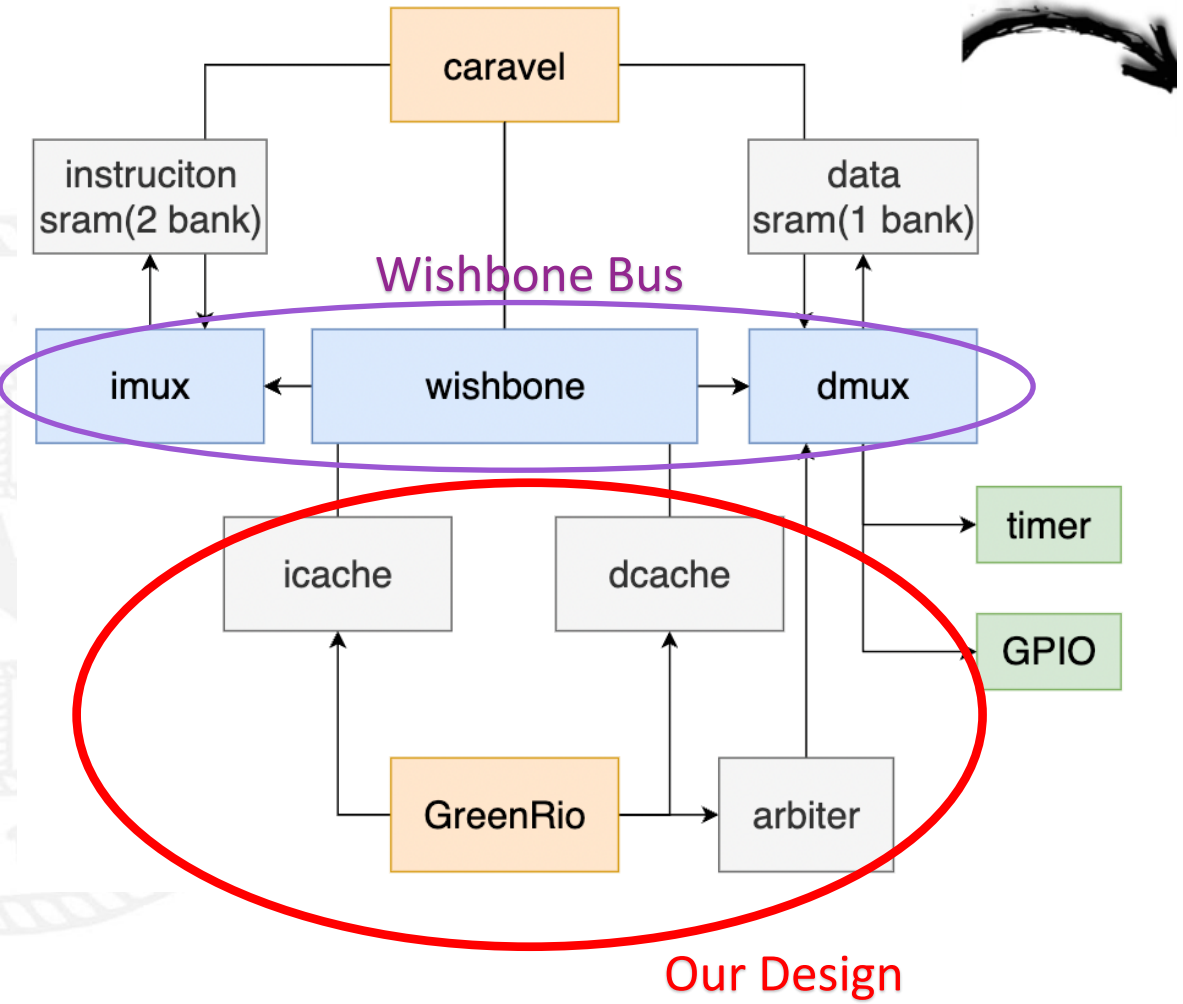
Frontend stage : F0-F3 Backend stage:F4-F6

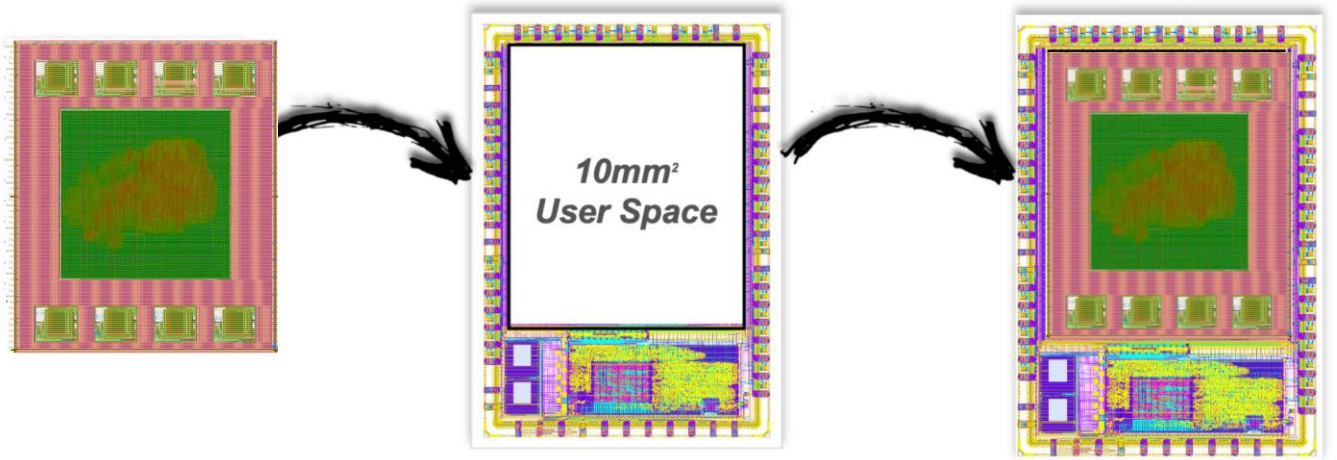


feature list

1. Dynamic branch prediction
2. Out-of-Order Execution
3. RISC-V I Extension, M mode
4. Register Renaming
5. Dual Issue
6. Nonblocking Dcache

SOC of GreenRio





OUR DESIGN

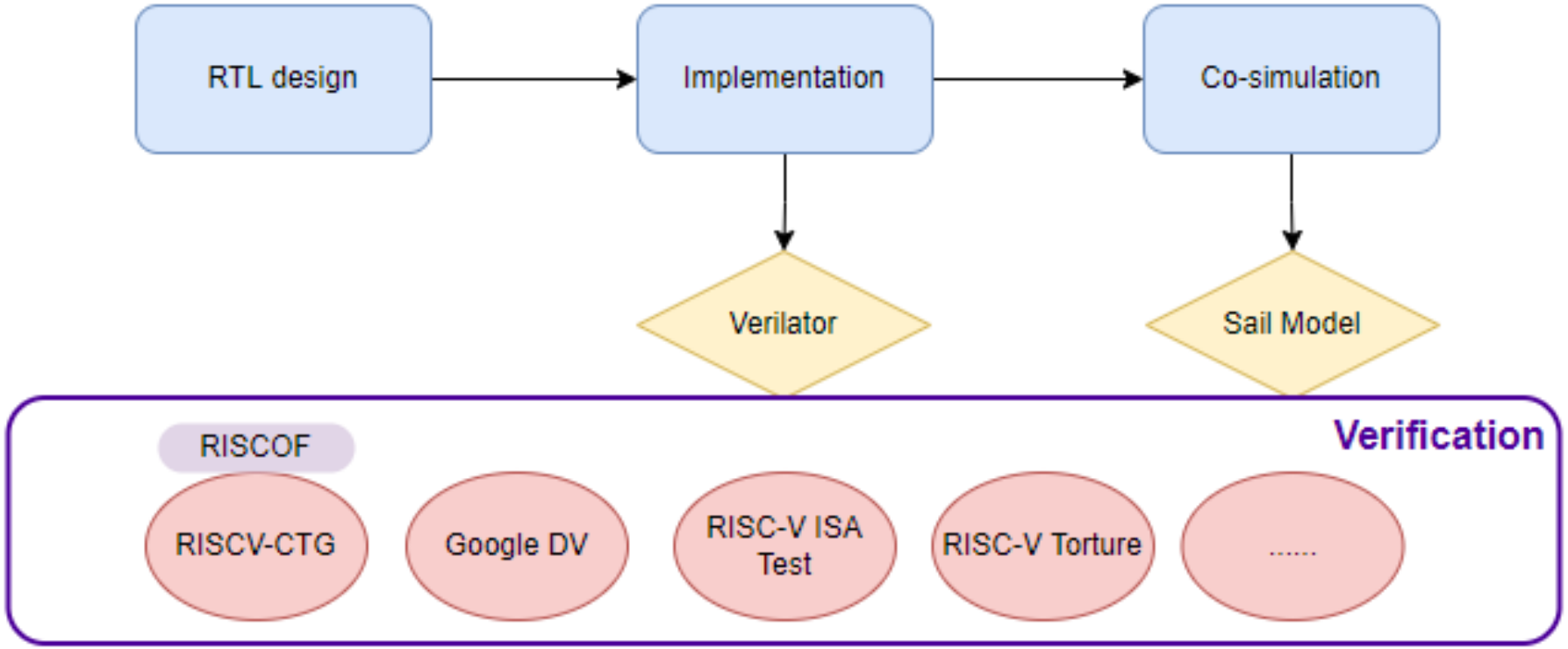
CARAVEL

INTEGRATED CHIP

https://github.com/daniel-santos-7/leaf-mpw6

Design Implementation and Verification

- Syntax Checking and RTL simulation: Verilator & Lint
- Verification: Open-source benchmarks for RISCV architectures
- Co-simulation: RISCV Sail model



Random Test Generator “Talon”

- Adding some randomness and being more flexible to adjust the test config

Google DV

input config

input dsl

config / dsl (interpreted by ”antlr” in python)

computational.config 81 Bytes

```
test:
sequence [U]{1000}
isa RV32IMC
dependency rd=rs1[3:0];rs1=rs2[3:0]
endtest
```

computational.talon 176 Bytes

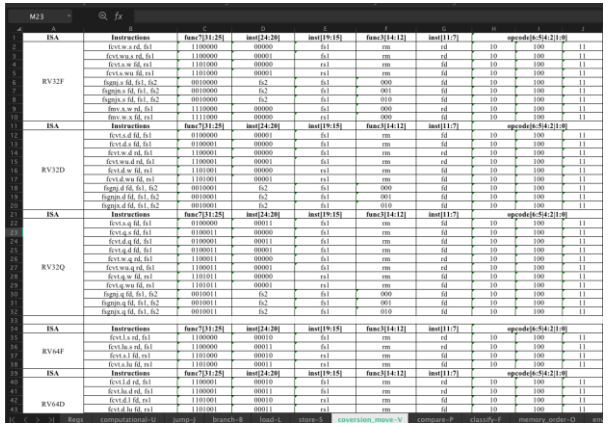
```
sequence sq "sort" ["U"] [1000]

test_case test1 sq ["RV32IMC"]

dependency dep "rd=rs1[3:0];rs1=rs2[3:0]"
let test1 add_dependency dep

let test1 add_checkpoint "at" -1
```

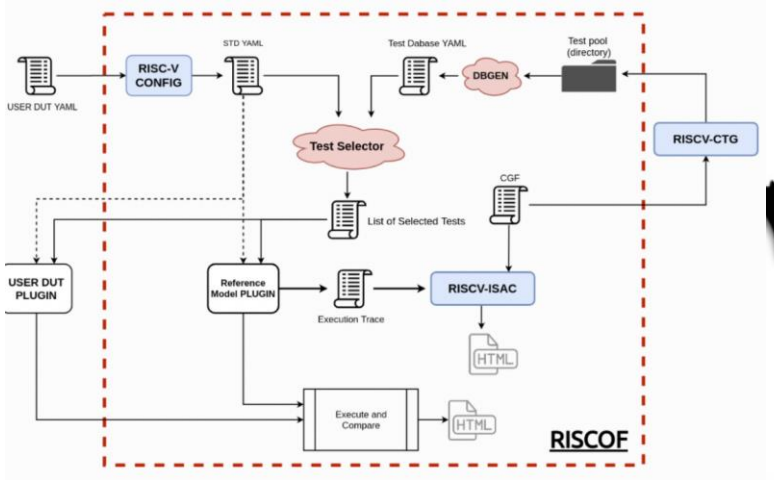
input encoding table



output.c

```
RVTEST_CASE(0,"//check ISA:=regex(.*32.*);check
    // random seed: 1
    asm volatile ("addi t2, a6, -0x5fc\n\t");
    asm volatile ("addi s8, t5, 0x662\n\t");
    asm volatile ("addi ra, t6, -0x500\n\t");
    asm volatile ("addi t3, zero, 0x5d9\n\t");
    asm volatile ("addi s4, t1, -0xae\n\t");
```

- Portable to the RISCOF infrastructures.



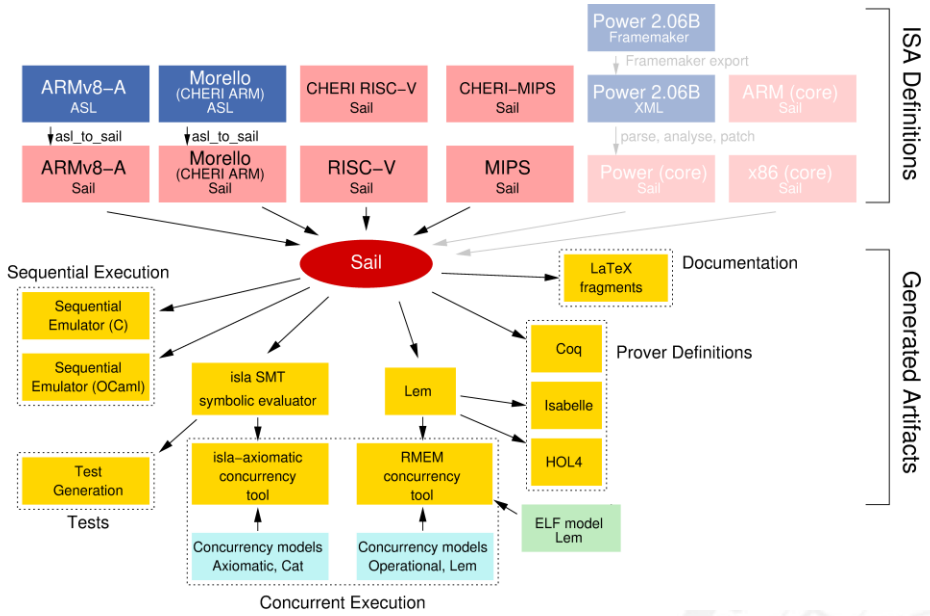
Talon

https://gitlab.com/picorio/software/talon-opensource

Co-simulation with RISCV Sail Model

What & Why Sail?

- Accurate transaction level abstraction
- Domain-Specific Language designed for expressing the ISA semantics
- RISC-V Sail: golden reference for RISC-V architecture
  - Reduces the possibility of human error
- Friendly to ISA extension
- Output clear information and registers' status
- Drawback: Slow C model compilation, but acceptable




```
/* the assembly abstract syntax tree (AST) clause for the ITYPE instructions */

union clause ast = ITYPE : (bits(12), regbits, regbits, iop)

/* the encode/decode mapping between AST elements and 32-bit words */

mapping encdec_iop : iop <-> bits(3) = {
  RISCV_ADDI  <-> 0b000,
  RISCV_SLTI  <-> 0b010,
  RISCV_SLTIU <-> 0b011,
  RISCV_ANDI  <-> 0b111,
  RISCV_ORI   <-> 0b110,
  RISCV_XORI  <-> 0b100
}

mapping clause encdec = ITYPE(imm, rs1, rd, op) <-> imm @ rs1 @ encdec_iop(op) @ rd @ 0b0010011

/* the execution semantics for the ITYPE instructions */

function clause execute (ITYPE (imm, rs1, rd, op)) = {
  let rs1_val = X(rs1);
  let immext : xlenbits = EXTS(imm);
  let result : xlenbits = match op {
    RISCV_ADDI  => rs1_val + immext,
    RISCV_SLTI  => EXTZ(rs1_val <_s immext),
    RISCV_SLTIU => EXTZ(rs1_val <_u immext),
    RISCV_ANDI  => rs1_val & immext,
    RISCV_ORI   => rs1_val | immext,
    RISCV_XORI  => rs1_val ^ immext
  };
  X(rd) = result;
  true
}

/* the assembly/disassembly mapping between AST elements and strings */

mapping itype_mnemonic : iop <-> string = {
  RISCV_ADDI  <-> "addi",
  RISCV_SLTI  <-> "slti",
  RISCV_SLTIU <-> "sltiu",
```

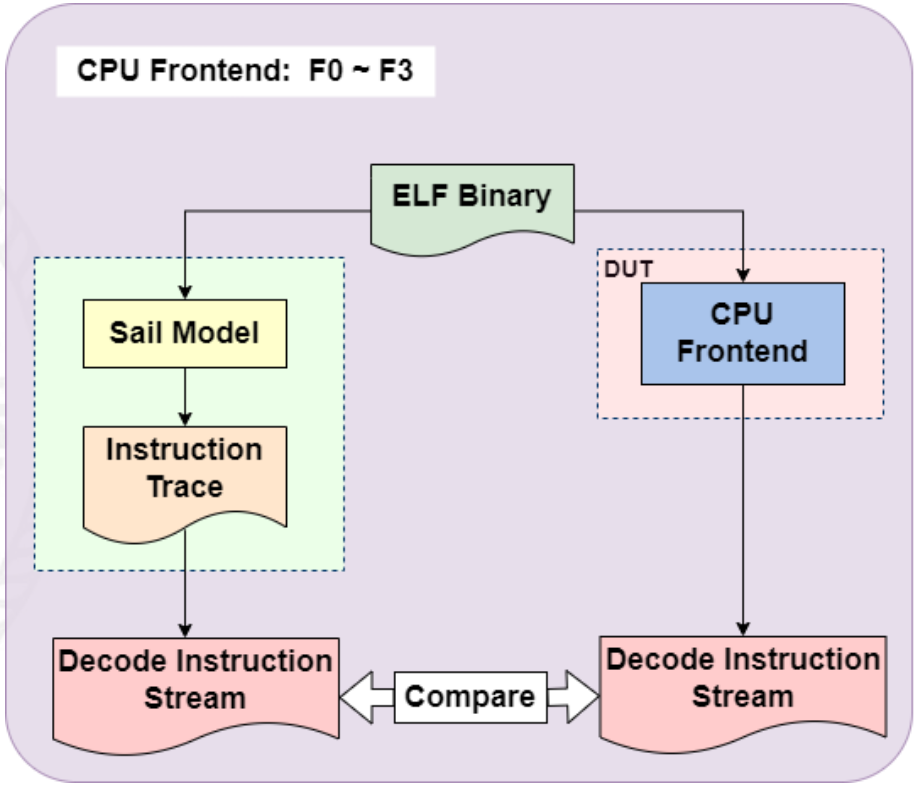
Vs spike

https://github.com/riscv/sail-riscv

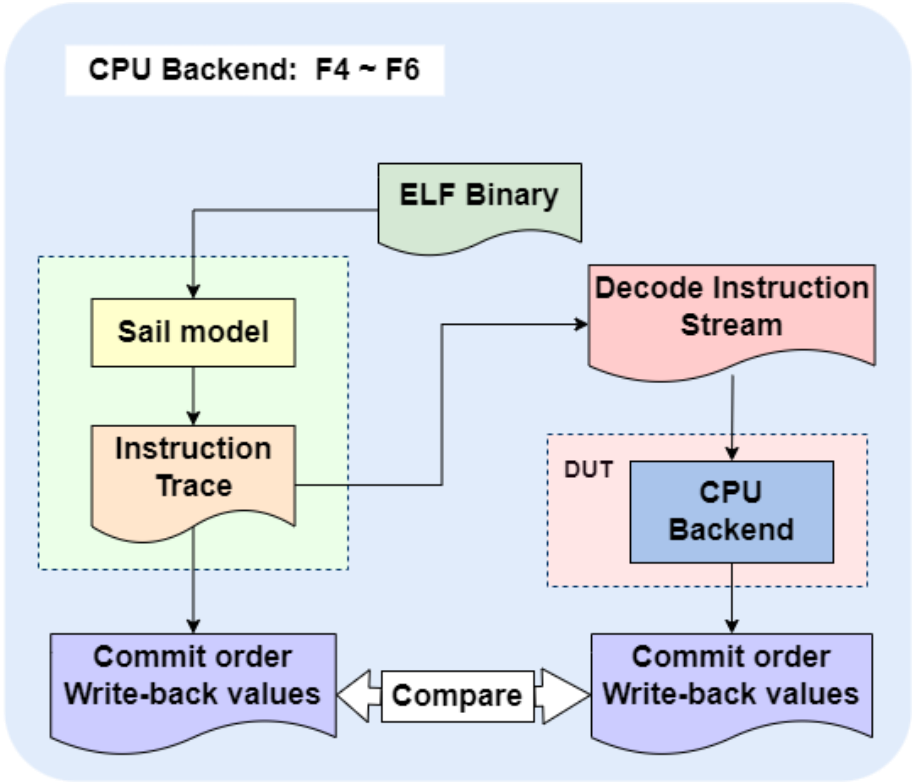
Co-simulation with RISCV Sail Model

Verify them respectively by comparing the results generated by RTL and Sail

- Frontend co-sim



- Backend co-sim



Backend Flow

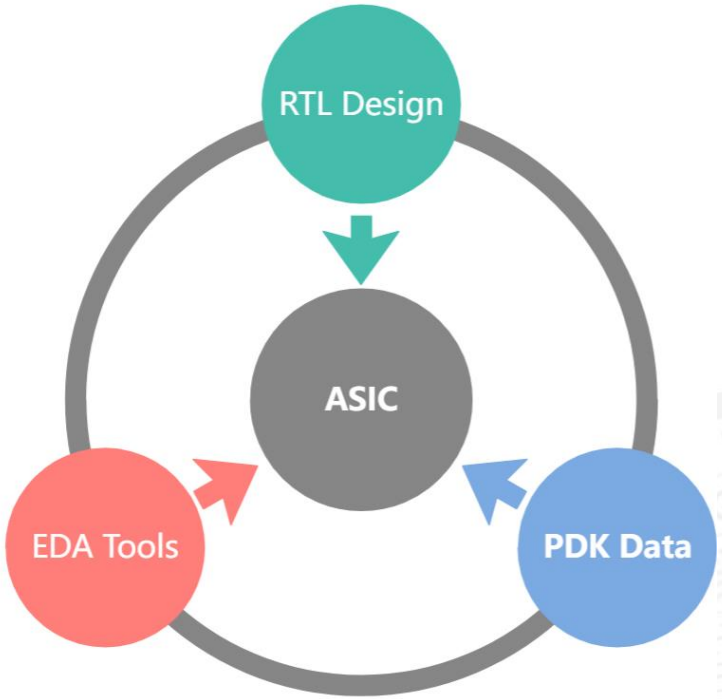
Experience, Analysis and Feedback

Technology dependent

Synthesis, floorplan, pnr

Open Silicon Implementation Flow

- Open-source backend EDA tools
  - OpenLane: an automated flow performing full ASIC implementation
- Open-source PDK
  - Skywater 130nm (Sky130A & Sky130B)
- Open-source external IP
  - SRAM blocks compiled by OpenRAM
- Open-source silicon production
  - Open Multi Project Wafer (OpenMPW)
  - Efabless Chipignite



Experience: A Journey of Discovery

- First tape-out project
- Openlane and open pdk
- 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

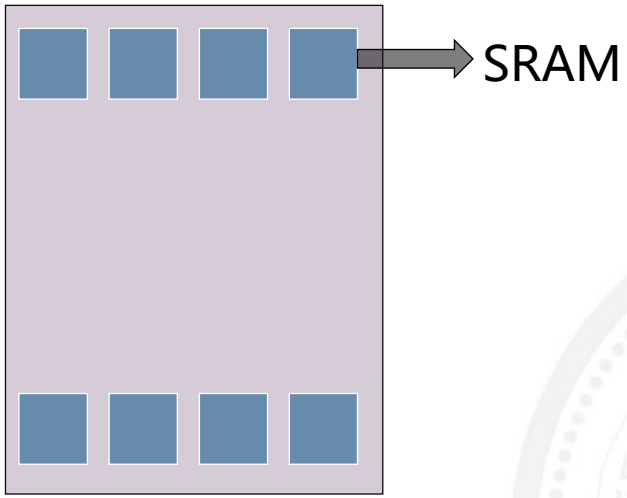
Detailed placement failed

Discussion

- Floorplan
  - 8 sram macros from Openram
- Needs manual participation

2nd Attempt: Manual Macro Placement

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



Result

- **Routing** time is too long:
  - More than 10 hours
- SIGINT signal sent

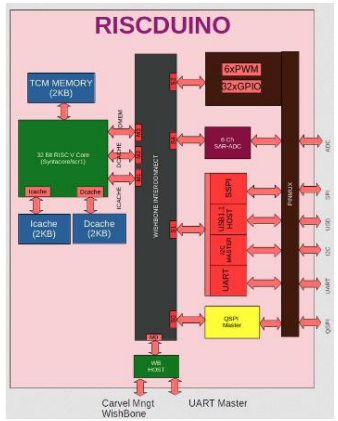
add further human intervention

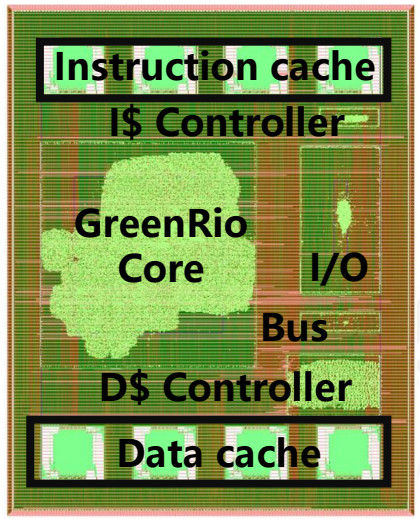
Discussion

- Routing algorithm efficiency?
  - *convergence is very time-consuming*
- Auto placement optimizations?
  - Difficulties may be caused by inappropriate placement

3rd Attempt: Learning from Riscduino

- Harden hierarchical design
  - Take riscduino as a reference
    - Single 32 bit RISC-V Core based SOC
  - Divide into 13 submodules
  - Harden submodules
  - Harden the full chip with macros





Result

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

Discussion

- SRAM DRC errors
  - **Can be waived** *(from official)*
- Routing space too small?
  - Only ~3 mm² blank space
  - More than 2000 wires between macros

4th Attempt: Improvement of the 3rd One

- Adjustments based on Attempt 3
  - Reduce the number of submodules: integrate the controllers with SRAMs
  - Waive 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)

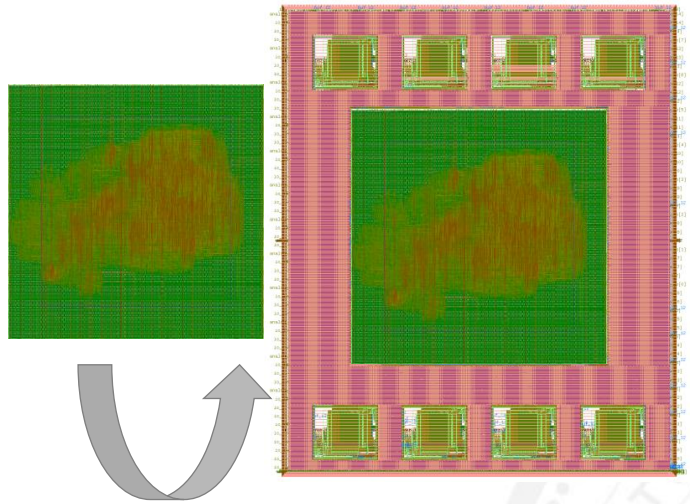
Discussion

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

⇨ adjust the number of submodules, find a successful strategy

Final Attempt: Learning from Failures

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



Result

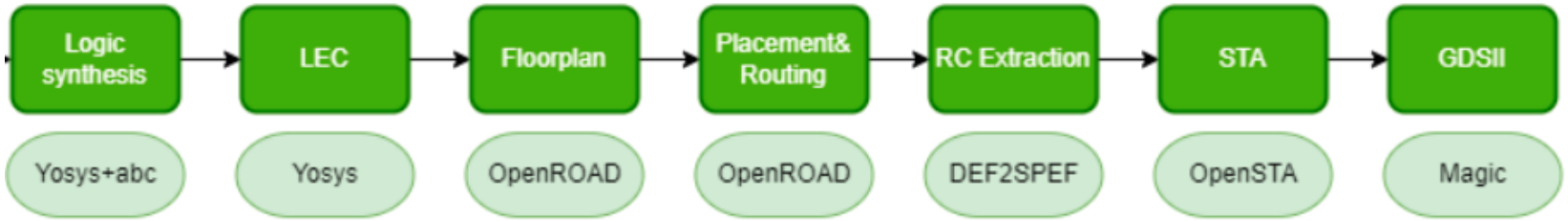
- Flow finished in 4.3 hours
- No DRC or LVS errors found
- Passed precheck & tape-out check

Discussion

- more effective routing algorithm
- Greenrio2: backend optimization exploration

Analysis: Feedback of OpenEda Tools

- What needs to be done manually
  - Proper Marco partition
  - Reasonable Floorplan
- Expectations and reflections on OpenLane
  - Automatic floorplan for complex designs
  - Smarter placement and routing algorithms
  - Trials and errors are annoying
- OpenLane is based on several components
  - OpenROAD, Yosys, Magic, Netgen, CVC, SPEF-Extractor, KLayout



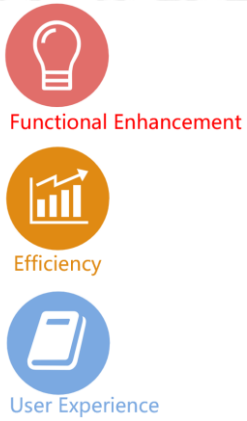
Comparison between Proprietary and Open EDA Tools

Comparison: Open vs Proprietary

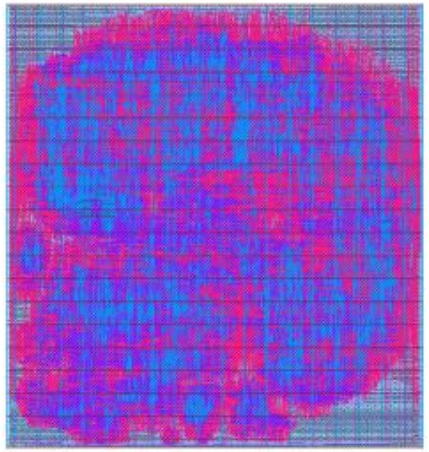
- Comparison with Proprietary EDA Tools: genius & innovas

	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

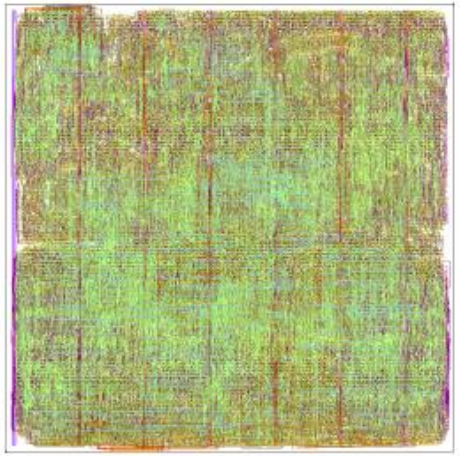
- Limitations of open-source EDA tools



1. System Verilog syntax support
2. PnR with high density
3. Logic equivalence check (LEC)
4. Antenna violation fixing
5. PPA optimization
6. Multi-thread acceleration
7. Early check mechanism
8. Tutorial and document



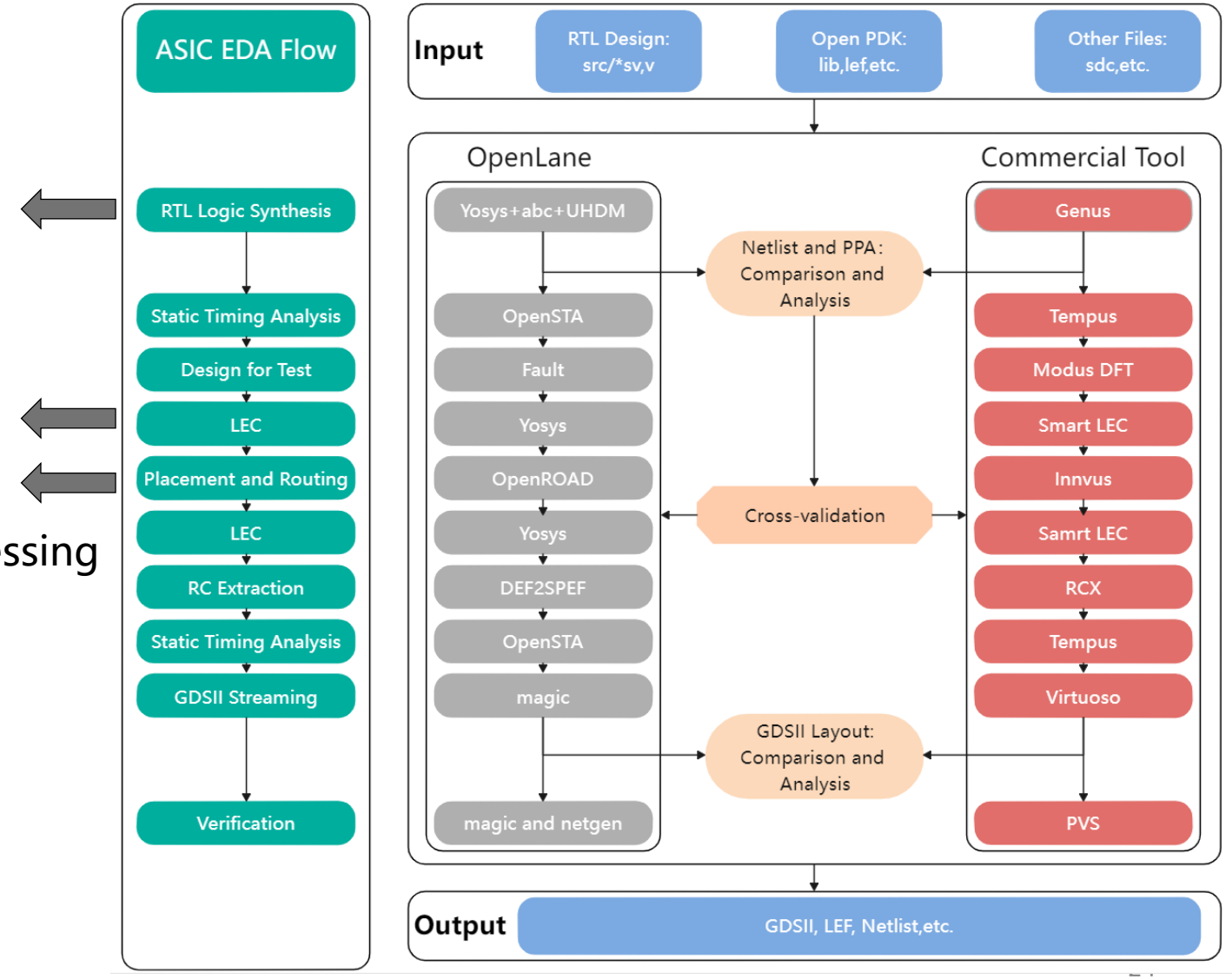
(a) Layout generated by OpenLane



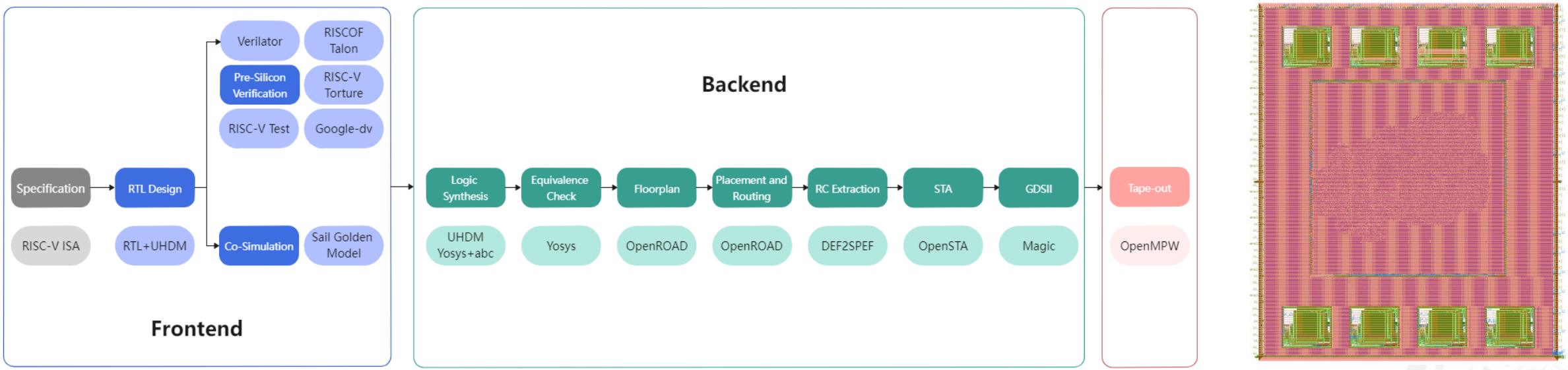
(b) Layout generated by Commercial EDA

Comparison: Open vs Proprietary

- **What is missing?**
  - Complete support for SV
    - Extensive ip usage
  - Available LEC
    - LEC function is abnormal
  - Efficient PnR algorithms
    - Higher density
  - Parameter searching
    - Early termination and endless processing
    - Parameter searching and indications
- **Accelerate Silicon Research**
  - Speed up with multi-threading
  - PPA optimization



Conclusion: Outlook on Future Open-source RISCV Design



Through this RISC-V developing experience

- Great usability of open-source EDA tools

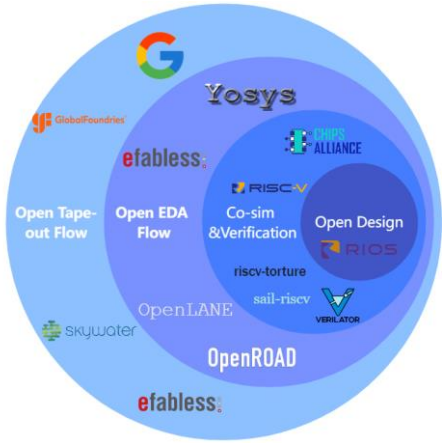
  High speed simulation, RISC-V Sail in verification, Automation of OpenLane…

- Several improvements to facilitate iteration

  Test generation, Design exploration, PPA optimization, …

Conclusion: Outlook on Future Open-source RISCV Design

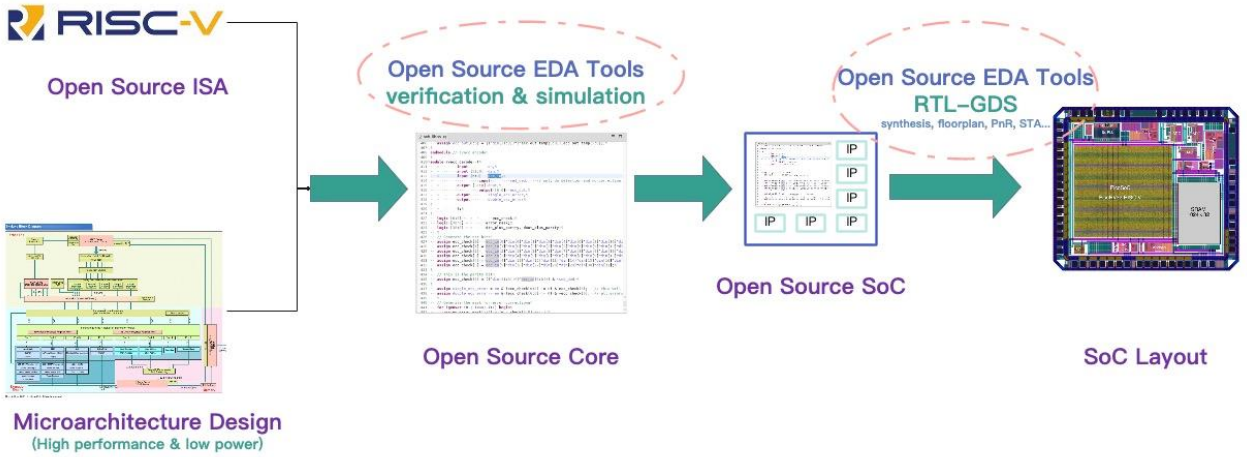
- **The RISC-V ISA has sparked a boom in open source hardware field**



→ “Open Chip”

（1）Instruction set
（2）Micro-architecture implementation
（3）Design flow with open tools (OpenEDA)

- **The development of RISCV core is closely related OpenEDA tools**



OpenEDA

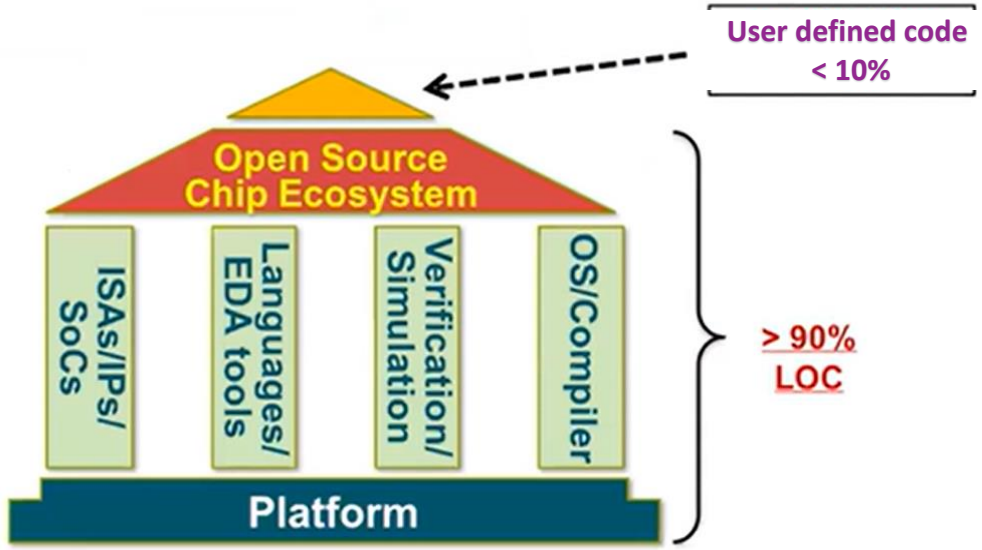
facilitate the whole process

Fully open source chip design strategy

Conclusion: Outlook on Future Open Source RISC-V Design

A full-stack open-source design methodology for modern processors

Prove the feasibility of designing high performance chip in open source mode



- Reduce open chips' the labor and cost of IP
- Accelerate the customized chip development
- Form open source chip ecology
- Promote the reform of Warehouse-scale computers(WSC) industry

In future...

bring a new perspective on future RISC-V architecture development

Greenrio 1.0
2022.09
Released

→

Greenrio 2.0
2022.11
Work in progress

→

...

Thank you

RIOS