GreenRio: A Modern RISC-V Microprocessor Completely Designed with A 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,
WeiWei Chen, Lei Ren, Zhangxi Tan
RIOS Lab, Tsinghua-Berkeley Shenzhen Institute, Tsinghua University
yifei.z@rioslab.org

Abstract—**The booming open-source EDA ecosystem brings transparency and reproducibility to the VLSI field, lowering the threshold for CPU design. However, the open design flow is relatively new and, due to the nature of open source resources, is accompanied by many hardships. This paper seeks to remedy this and proposes a full-stack design methodology for modern processors relying purely on open-source tools. To this end, the paper describes our design process for GreenRio, a 64-bit, dual-issue, out-of-order RISC-V microprocessor completely designed with a open-source EDA flow. Based on our tape-out experiences in the efabless Chipignite shuttle[1] and OpenMPW programs, we analyzed multiple open-source EDA tools used to develop GreenRio and compared the open versions of these tools to their proprietary versions. Furthermore, we also discussed their current limitations as well as future possible optimizations. We hope our methodology can bring a new perspective towards agile modern architecture development.**

Index Terms—**OpenEDA, OpenLane, ASIC, RISC-V Processor, Sail**

I. Introduction

In recent years, we have witnessed increasing complexity in IC designs of advanced processing technologies. The sky-rocketing costs, difficulties, and risks of design have put silicon implementations out of reach of system innovators. This crisis of design and innovation has brought renewed attention to the hardware design process itself [1]. Thus, it is becoming more important to lower the threshold for CPU design and enable agile development. In 2021, Edwards shared a tape-out project using only OpenLane and OpenPDK [2], making open-source chip manufacturing possible. Against this backdrop, an end-to-end CPU design flow in a fully open-source manner is becoming increasingly attractive.

Table 1 shows comparisons of features implemented by GreenRio versus other famous designs submitted to OpenEda. Compared to GreenRio, the other designs lack typical structural components of modern processors such as out-of-order execution, renaming mechanisms, reorder buffers, etc. Considering the lack of implementation of these features in other designs we believe that GreenRio has been one of the most complex designs in the efabless silicon projects. It also has a high level of micro-architectural complexity, making it one of the most advanced processors that openEDA has run so far and will be a good reference point for future projects that wish to use the open EDA flow.

In this paper, we use exclusively open-source resources to construct a full-stack modern architecture design methodology. We designed a seven-stage RISC-V core called “GreenRio”, which supports dynamic branch prediction, out-of-order execution, and has a non-blocking data cache. Fig. 1 illustrates the whole design flow.

The rest of the paper is organized as follows. Section 2 describes the frontend design of our core, including RTL implementation and verification. Section 3 evaluates the backend design with both open and proprietary EDA tools. Section 4 gives the outlook on future agile design with some improvment schemes proposed. Section 5 summarizes the main conclusions of this work.

II. frontend Design

The frontend design includes core specifications, RTL implementaion and verification. Generally engineers simulate the core’s functions in an RTL Simulator and have to verify the sanity of the design. In this sections, we introduced tools used in simulation and verification. And in order to test our core throughly, a test generator designed by ourself was also described.

A. System Verilog to Verilog

We choose to write our core in System Verilog (SV) which is close to being completely supported by mainstream dedicated EDA software. However, for most open-source tools such as Yosys[6] or Verilator, SV support is still limited. We believe that syntax support for SV is essential as mainstream IP is now more often than not written in SV than Verilog. Currently, in order to quickly adapt to an open-source backend flow, there is a strong need for automatic translation of hardware programming language into Verilog. Otherwise RTL has to be manually developed using Verilog or Chisel.

[1]https://github.com/b224hisl/rioschip2

[6]https://github.com/YosysHQ/yosys

TABLE I
COMPARISON OF TYPICAL CORES COMPLETED BY OPENEDA

	Picorv32a[2]	**EH1**[3]	**ibex**[4]	**biriscv**[5]	**GreenRio**
ISA	RV32	RV32	RV32	RV32	RV64
pipline stage	6	9	2 or 3	6 or 7	7
issue width	single	dual	single	dual	dual
execution feature	in-order	in-order	in-order	in-order	out-of-order
gate-count level (K)	17	10	20	67	53
Efabless tape-out	Y	N	N	Y	Y

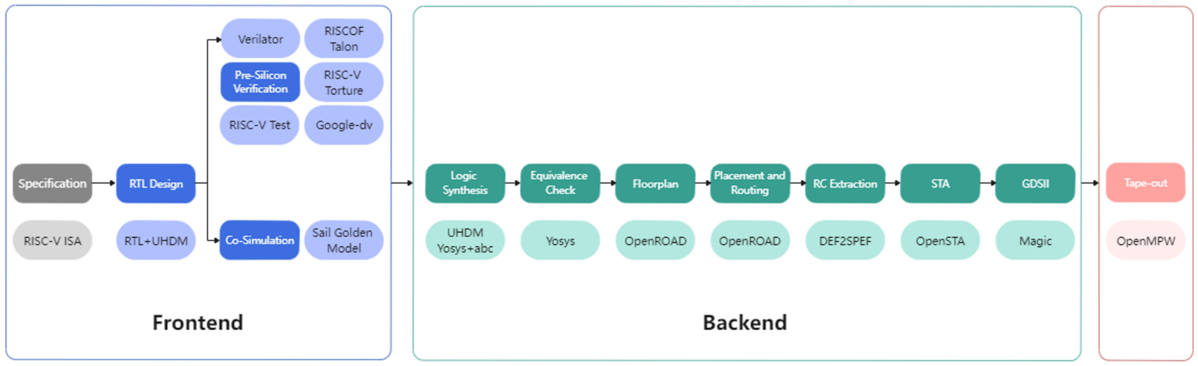


Fig. 1. Full-Stack Modern Processor Design Flow in Open-source Mode

We have explored two open-source tools for translating SV to V: Surelog+UHDM[7] and Sv2v[8]. Surelog has good syntax parsing capabilities for SV. It does the transformation through an intermediate UHDM-format file. However, the process is very time-consuming and does not guarantee correctness for some complex syntactic structures, such as ”enums” or ”breaks” in for-loops. And sv2v doesn’t have such problems.

Although there are reliable tools for the translation of SV to Verilog there is still a lack of open-source tools for formal verification between SV and V. These verification tools are fundamental to the open ecological chain of chip design and even proprietary tools like Cadence’s conformal are still not perfect for a few syntax cases. We expect future open-source EDA tools to be able to fill the gap.

B. *Simulation and Lint*

When developing a complex hardware structure, there are two bottlenecks that greatly increase development time: syntax checking and RTL simulation.

Verilator[9] is an open-source Verilog HDL simulator that instantiates a user’s top-level module. And Verilator is integrated with a lint system [3] to check the codes for syntax errors.

After stimulis are fed into user-defined modules, Verilator generates waveforms that can be visualized by Gtkwave. This process is fast and allows multi-threaded acceleration.

However, the lint checker is not flexible enough to specify some modules which are templated from FPGA constructors. Its inapplicability adds inconvenience our design process.

C. *Verification*

To achieve functional robustness, we have to run various tests on the core. Meanwhile, we expect a golden model to reflect correct architecture states in each CPU cycle for reference.

Currently, there are multiple open-source verification resources provided for RISC-V architectures. For example, the RISC-V Tests repository[10] has ISA coverage tests and typical benchmarks. There is also the RISC-V Torture repository[11] can be used to test more complex patterns.

However, due to the variable instruction parameters and infinite instruction combinations, it is necessary to generate a large number of tests that hit different edge cases in order to verify the sanity of our design and iteratively refine the model. For this purpose, we need an automatic test generator to generate pseudorandom instruction streams to cover specific edge cases. Google-dv[12] is an example of a test generator with such functionality. But it needs lots of prerequisites and has poor support for Verilog.

We have developed an open-source random architectural test generator called Talon [13] in order to meet our needs for a controllable automatic test generator. Talon uses YAML for user-controlled test parameters for readability and reusability [4]. Furthermore, Talon is integrated with RISCOF[14] for further architectural testability.

[7]https://github.com/chipsalliance/UHDM-integration-tests
[8]https://github.com/zachjs/sv2v
[9]https://github.com/verilator/verilator

[10]https://github.com/riscv-software-src/riscv-tests
[11]https://github.com/ucb-bar/riscv-torture
[12]https://github.com/google/riscv-dv
[13]https://gitlab.com/picorio/software/talon-opensource
[14]https://github.com/riscv-software-src/riscof

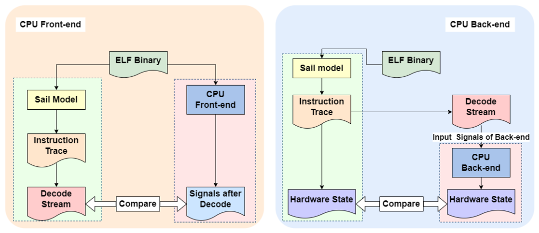
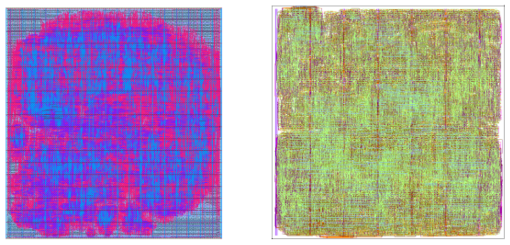


Fig. 2. COSIM Unit Test Environment integrated with Sail



(a) Layout generated by OpenLane (b) Layout generated by Commercial EDA

Fig. 3. Design layout

Besides, to verify as much of the hardware functionality as possible before fabricating the ASIC, co-simulation (COSIM) tools provide project architects with a simulation environment at a very high level of abstraction, manipulating simulated hardware with software.

Following this trend, we built a co-simulation unit test environment. We integrated Sail model into it, because the reference model has to be golden enough for transaction-based simulation. Sail is a Domain-Specific Language designed for expressing the ISA semantics of distinct computer architecture[15]. The RISC-V Sail model[16] is the current golden reference model for modern RISC-V computer architecture ISA design. The model allows for more flexibility in ISA extensions and provides a clear method to dump information from its instruction stream. These features make Sail a more suitable reference model for a core's COSIM verification than a more barebones model such as spike[17].

We split the CPU into frontend and backend, and verified them respectively by comparing the results generated by RTL and Sail. Fig. 2 shows the COSIM environment.

III. BACKEND DESIGN

Regardless of implementation, the frontend is technology independent. Once the frontend has been implemented, the core is then hardened with a specific PDK. Backend design includes logic synthesis, floorplan, layout, CTS, routing, static timing analysis, etc. In this sections, we evaluated the backend design with both open and proprietary EDA tools respectively and proposed optimization points of open EDA tools.

A. OpenLane and OpenPDK

Shalan et al. [5] introduced OpenLane[18], an automated RTL to GDSII flow performing full ASIC implementation steps from RTL down to GDSII. In July 2020, Google/Skywater launched the OpenPDK project [6], which is available to the public for use on custom silicon design. In a large-scale design, external IPs are usually used to speed up the process. For example, openRAM [7], an open-source memory IP, serves as Icache and Dcache in our processor design.

Through running OpenLane, we concluded that an automatic search of signoff parameters is a necessity. Openlane uses Tool Command Language (TCL) to configure various parameters. The suitability of parameters, especially those related to the floorplan and processes, is paramount to the completion of the whole process. As improper settings will cause early termination of the program. Therefore, to reduce the time cycle of backend implementation, it is important to design a configuration scheme based on automatic parameter searching.

B. Comparison with Proprietary EDA Tools

In our work, we harden our 64-bit RISC-V CPU "GreenRio" with both Openlane and proprietary EDA tools. Fig. 3 shows layout images generated by open EDA(a) and proprietary EDA flow(b). Specific statistical results are shown in Table II.

TABLE II
COMPARISON BETWEEN OPEN AND PROPRIETARY EDA TOOLS

Timing Corner: sky130_fd_sc_hd_tt_025C_1v80
Timing Closure: No hold & setup violation
CPU cores for routing: 16vCPU

	OpenLane	**proprietary EDA**	**Gap**
synthesis run Time	6m12s	4m04s	1.5X
gate count	53K	33K	1.6X
placement & routing time	1h58m	43m	2.7X
die area(mm^2)	2.02	1.24	1.6X
leakage power	209nW	152nW	1.4X
placement density	32%	45%	1.4X
best clock period	80MHz	110MHz	1.4X

From the experimental results, we can generally conclude that open EDA tools have higher potential. Under the same clock frequency, the area optimization performance has reached 60% of that of proprietary ones. Furthermore, improving the parallel computing capability can facilitate agile and high-quality development of modern processors.

We also explore some other features for comparison. The can be summarized as follows:

- **SystemVerilog Syntax Support**: OpenLane only supports Verilog. This is an issue due to the increasing popularity of SV. Thus, the translation of SV to Verilog is crucial.
- **PnR Algorithm Optimization**: Openlane's PNR algorithm should enhance its performance under higher density. The current algorithm has difficulties passing

[15]https://github.com/rems-project/sail
[16]https://github.com/riscv/sail-riscv
[17]https://github.com/riscv-software-src/riscv-isa-sim
[18]https://github.com/The-OpenROAD-Project/OpenLane

the routing congestion check once the density exceeds 40%. The correct parameter combination of floorplan, placement, and routing needs several iterations of trial and error, lengthening the development schedule.

- **Logic Equivalence Check (LEC)**: Compared with the proprietary tool *Cadence Conformal*, the open-source tool does not always provide correct verification results.
- **Antenna Violation Fixing**: When comparing data between the two sets of EDA flows used to harden GreenRio, we can see that OpenLane has antenna violations exceeding the number reported by proprietary tools. We believe that improving the violation repair algorithm will be of great benefit to reducing the difference in reported violations between the two tools.
- **PPA optimization**: The PPA obtained by Openlane is still lagging behind that provided by the proprietary EDA tool. Under the same clock frequency, the area of optimization performance for GreenRio hardened by OpenLane has only reached 60% of that of proprietary EDA tools.
- **Multi-thread Configuration**: Only the routing stage has multithreading support, resulting in a very slow flow.
- **Tutorial and Documentation**: The specification related to the backend's config file is brief and obscure leading it to be hard to understand. We believe this is not conducive to increasing the popularity of open-source tools.

IV. Outlook on Agile Design

For agile development in the IC field, it is important to improve the intelligence of EDA tools, thereby further increasing the degree of flow automation. It is also important to learn from successful experiences of open-source projects in the software industry and study the open-source design modes and methods.

Compared with proprietary EDA softwares, open EDA has excellent usability. We can see highlights of openEDA tools, such as the high-speed simulation of Verilator with Gtkwave, speeding up the frontend simulation; the great potential of Sail in co-simulation verification; and the high-level automation and availability integrated into OpenLane.

There are several key improvements to be considered for future agile development:

- Cover more verification patterns with fewer test cases.
- Achieve more effective design explorations with right hints.
- Build a smarter backend iteration system with the help of Artificial Intelligence (AI).

The first improvement can be achieved through the development of a constrained random instruction that uses specific mathematical algorithms. The last two improvements can be achieved through the use of AI, which has found success in many fields and can be used in designing PPA optimization. Feedback from the backend can be used to help iterate during frontend module development. Ziegler [8] has introduced SyntunSys, which has been used to deal with the problem of design space parameter search in industrial VLSI. Google has built a parameter tuning tool called Vizier[19] which has been used in OpenLane's parameter tuning [9]. With so many tools being developed a modern processor designed in an open-source manner with full-stack optimization will eventually be achieved.

V. Conclusion

In this work, we have used an open-source toolchain to complete design stages of a modern processor, GreenRio. GreenRio is one of the most complex processors that openEDA has run so far making it a good design to use to look at comparisons between open-source toolchains versus their proprietary versions.

In summary, the open-source flow's support and acceleration for some specific computational tasks can be further optimized. In addition, the lack of a unified process standard has led to relatively large differences in the quality of open-source IPs, leading to low reusability. If more advanced process design kits can be combined, the open-source mode can be adapted to a wider range of scenarios.

The ecology of open-source silicon has developed rapidly. Both the efabless shuttle[20] and the open EDA tools reduce the cost of silicon production and shorten the iterative loss during scientific inquiry. The boom in open EDA tools also has pedagogical implications, lowering the threshold for CPU design and bridging the knowledge gap between classroom and industry.

VI. ACKNOWLEDGMENT

We would like to express our deepest appreciation to Johan Euphrosine of Google for his patient guidance in this project.

References

[1] A. B. Kahng, "Open-Source EDA: If We Build It, Who Will Come?," 2020 IFIP/IEEE 28th International Conference on Very Large Scale Integration (VLSI-SOC), 2020, pp. 1-6, doi: 10.1109/VLSI-SOC46417.2020.9344073.
[2] R. T. Edwards, M. Shalan and M. Kassem, "Real Silicon Using Open-Source EDA," in IEEE Design & Test, vol. 38, no. 2, pp. 38-44, April 2021, doi: 10.1109/MDAT.2021.3050000.
[3] Snyder W. Verilator: Open simulation-growing up[J]. DVClub Bristol, 2013.
[4] Y. F. Zhu, X. Wang, "RISC-V Vector Sail Model and Test Generation" in Vector & Machine Learning
[5] M. Shalan and T. Edwards, "Building OpenLANE: A 130nm OpenROAD-based Tapeout-Proven Flow : Invited Paper," 2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD), 2020, pp. 1-6.
[6] Google/Skywater-pdk: https://github.com/google/skywater-pdk
[7] M. R. Guthaus, J. E. Stine, S. Ataei, Brian Chen, Bin Wu and M. Sarwar, "OpenRAM: An open-source memory compiler," 2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2016, pp. 1-6, doi: 10.1145/2966986.2980098.
[8] Matthew M. Ziegler, Jihye Kwon, Hung-Yi Liu and Luca P. Carloni. 2021. Online and Offline Machine Learning for Industrial Design Flow Tuning: (Invited - ICCAD Special Session Paper). In 2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD). IEEE Press, 1–9. https://doi.org/10.1109/ICCAD51958.2021.9643577
[9] Golovin D, Solnik B, Moitra S, et al. Google vizier: A service for black-box optimization[C]//Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining. 2017: 1487-1495.

[19]https://github.com/google/vizier
[20]https://github.com/efabless/caravel_user_project