ChatCPU: An Agile CPU Design & Verification Platform with LLM

Xi Wang
xi.wang@seu.edu.cn
National ASIC Center, School of Integrated Circuit, Southeast University
National Center of Technology Innovation for Electronic Design Automation
Nanjing, Jiangsu, China

Gwok-Waa Wan*
Sam-Zaak Wong
Layton Zhang
{gwokwaa,samzaak,layton}@nctieda.com
National Center of Technology Innovation for Electronic Design Automation
Nanjing, Jiangsu, China

Tianyang Liu
Qi Tian
Jianmin Ye
{liutianyang,qi_tian,jianmin_y}@seu.edu.cn
National ASIC Center, School of Integrated Circuit, Southeast University
Nanjing, Jiangsu, China

Abstract

The increasing complexity of semiconductor designs necessitates agile hardware development methodologies to keep pace with rapid technological advancements. Following this trend, the Large Language Models (LLMs) emerge as a potential solution, providing new opportunities in hardware design automation. However, existing LLMs exhibit challenges in HDL design and verification, especially for complicated hardware systems. Addressing this need, we introduce ChatCPU, the first end-to-end agile hardware design and verification platform with LLM. ChatCPU streamlines the ASIC design and verification process, guiding it from initial specifications to the final RTL implementations with enhanced design agility. Incorporating the LLM fine-tuning and the processor description language design for CPU design automation, ChatCPU significantly enhances the hardware design capability using LLM. Utilizing ChatCPU, we developed a 6-stage in-order RISC-V CPU prototype, achieving successful tape-out using SkyWater 130nm MPW project with Efabless, which is currently the largest CPU design generated by LLM. Our results demonstrate a remarkable improvement in CPU design efficiency, accelerating the design iteration process by an average of 3.81X, and peaking at 12X and 9.33X in HDL implementations and verification stages, respectively. The ChatCPU also enhances the design capability of LLM by 2.63X as compared to base LLama2. These advancements in ChatCPU represent a significant milestone in LLM-driven ASIC design and optimization.

CCS Concepts: • **Computing methodologies** → **Artificial intelligence**; • **Computer systems organization**; • **Hardware** → **Electronic design automation**;

Keywords: ChatCPU, LLM, Agile Hardware Design, Verification

ACM Reference Format:
Xi Wang, Gwok-Waa Wan, Sam-Zaak Wong, Layton Zhang, Tianyang Liu, Qi Tian, and Jianmin Ye. 2024. ChatCPU: An Agile CPU Design & Verification Platform with LLM. In *61st ACM/IEEE Design Automation Conference (DAC '24), June 23–27, 2024, San Francisco, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3649329.3658493

*Corresponding author: Gwok-Waa Wan

DAC '24, June 23–27, 2024, San Francisco, CA, USA

ACM ISBN 979-8-4007-0601-1/24/06
https://doi.org/10.1145/3649329.3658493

1 Introduction

In the era of big data, the surge in data-intensive workloads such as machine learning, graph analytics, blockchain, etc. shows the ever-increasing demand for enhanced computing capabilities. This need, alongside advancements in CMOS scaling and domain-specific architectures, has catalyzed an escalating expansion in the complexity of modern silicon designs. As a consequence, the process of constructing hardware prototypes for such systems from the ground up has become a formidable challenge, entailing substantial time and effort dedicated to implementations and verification.

Targeting this challenge, many efforts have been devoted to agile hardware designs to accelerate iterations. Recent approaches have leveraged High-Level Synthesis (HLS) [6] and modern Hardware Description Languages (HDLs) like Chisel and SpinalHDL [2], which can subsequently be compiled to RTL. However, these methods aim to enhance agility via additional layers of abstraction, resulting in difficulties of design verification and fine-grained control over the performance, power, and area (PPA) [12, 16].

Innovations in Machine Learning (ML) and Artificial Intelligence (AI) offer new avenues for agile hardware design [10]. Current ML applications target specific design stages such as the design space exploration, physical design, etc., yet their integration across the entire processor design cycle spanning from hardware specifications to HDL implementation is still emerging [15, 17, 18]. The processor design and verification process, particularly, remains labor-intensive, requiring significant expertise and extensive validation. The advent of Large Language Models (LLMs) [13] presents an opportunity in this domain. Recent efforts demonstrate the potential of LLMs in automating hardware implementations [3, 4]. However, the quality of LLM-generated hardware designs poses challenges in synthesis and functional verification, often leading to increased debugging and verification efforts. Moreover, existing LLM-based silicon generations remain tiny-scale, and LLMs exhibit limited capability in complex system designs like CPUs.

Therefore, we introduce **ChatCPU**, an agile CPU design and verification platform with fine-tuned LLM. ChatCPU features a hierarchical CPU design methodology and modularized HDL generation paradigm. We also designed a new Processor Description Language (PDL) dedicated to clear and unified CPU architecture descriptions, improving prompt management by eliminating ambiguity in design specifications. We further introduce an Agile Model Generator (AMG) for automated CPU reference model generation based on the CPU design description with PDL. Integrating the AMG and PDL, we propose a Module-interchangeable Verification Paradigm (MVP) to address the verification challenges of complicated hardware system designs using LLMs. Attributed to the enhanced intelligence and automation, ChatCPU substantially reduces design iteration time and effort, streamlining the lab-to-fab process of

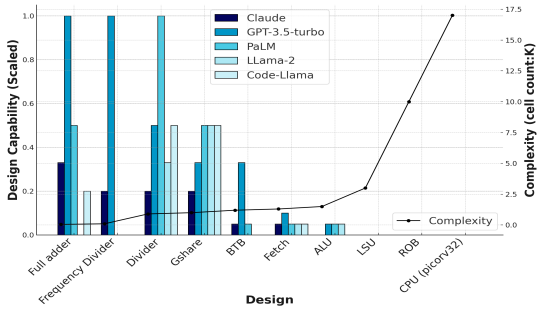


Figure 1. Hardware Design Capability with Different Complexities

hardware development. As a validation of the proposed framework, we designed *Saber6*, a 6-stage in-order RISC-V CPU with ChatCPU and implemented the silicon prototype with OpenEDA flow [14] and SkyWater 130nm MPW project. As a case study, we utilized ChatCPU to design and implement a RISC-V CPU prototype with the OpenEDA flow [14] and the MPW project, demonstrating the potential of ChatCPU to revolutionize CPU design and verification.

This research makes six key contributions.

- We extensively analyze the RTL design capability of existing LLMs and summarize the challenges of LLM-based design.
- We introduce ChatCPU, the first CPU design and verification platform with LLM, featuring a high degree of extensibility and scalability. ChatCPU is capable of autonomously driving end-to-end ASIC design.
- We construct the CPU design dataset and fine-tune the LLama 2 to enhance the capability of RTL generation.
- We design a processor design language and an agile model generator to automate the generations of CPU designs and reference models in a homologous manner.
- We propose a module-interchangeable verification paradigm to drive complete CPU verification flow upon each CPU module generated by LLM, based on the co-simulation support between modularized RTL design and CPU reference model.
- We design and verify a RISC-V CPU from scratch with ChatCPU, and fabricate the silicon prototype with OpenEDA and SkyWater 130 OpenPDK [7] in the Efabless MPW project.

The rest of this paper is organized as follows. Section 2 presents the motivation of ChatCPU and related work. Section 3 describes the detailed ChatCPU design and workflow. Section 4 presents the evaluations of the ChatCPU, and Section 5 concludes the paper.

2 Background

2.1 LLM in Agile Hardware Design

Recent advancements in LLMs such as BERT, GPT-2, and GPT-3 have showcased significant capabilities in Natural Language Processing (NLP) and text generation. These models utilize contextual information effectively, offering the potential for generating intricate hardware designs from high-level natural language inputs [9]. Initiatives like ChipGPT [4] and ChipChat [3] take advantage of LLMs for hardware design. However, ChipGPT focuses more on optimizing prompts rather than tackling the broader challenges of agile hardware design [19]. Conversely, ChipChat, while advancing LLMs in chip design, has only demonstrated results on a small scale, approximately 1,000 logic gates, which pales in comparison to modern processors that incorporate billions of gates. This disparity raises questions about the scalability and practicality of these methods for more complex designs.

In addition to design generation, verification remains a crucial aspect of AI-driven hardware design paradigms. For example, a recent study explored AI-driven CPU design based solely on external input-output observations with extensive I/O operations for circuit matching, rather than formal program code. This approach, while effective within its scope, results in a 'black-box' design process, leading to potential redundancies and challenges in design verification [5]. Current LLMs, despite their advancements, still produce hardware designs that are prone to errors and may not pass synthesis and functional checks, thereby inducing overhead in verification. Given the high costs associated with design verification in modern CPUs, it is imperative that agile hardware design not only automates HDL generation but also streamlines the verification.

2.2 LLM Hardware Design Analysis

To assess the efficacy of LLMs in hardware design generation, we conducted an evaluation using Claude, GPT3.5, PaLM, LLama2, and Code-LLama. Our test involved generating ten different open-source hardware designs written in Verilog, with complexity ranging from a simple adder to a sophisticated CPU. Each LLM received identical design specifications, and we monitored their results of synthesis and functionality verification. If errors were detected, the LLMs were tasked with debugging and regenerating the code. We quantified the design capability (C) of each LLM using the metric,

$$C = \frac{1}{N} \tag{1}$$

where N represents the number of iterations needed by the LLM to pass both synthesis and verification.

Following this metric, a higher C value indicates superior design capability. We allowed a maximum of 20 iterations for each design. If an LLM failed to produce a viable design within these iterations, its design capability for that particular design was recorded as zero. As depicted in Figure 1, GPT3.5 demonstrated the highest design capability among the evaluated LLMs. However, we observed a consistent decline in design capability across all LLMs as design complexity increased, particularly evident when transitioning from simpler designs to those comprising over 17,000 cells, such as the PicoRV32 CPU. Notably, none of the LLMs succeeded in generating functional Load Store Units (LSU), Reorder Buffers (ROB), or the PicoRV32 CPU designs, underscoring the challenges LLMs face in handling complex system designs.

2.3 Problem Formulation

Building upon the results and analysis from Section 2.2, we identify and define three primary challenges (*C1–C3*) in LLM-based hardware design:

- **C1. Limited HDL Generation Capability**. The hardware designs generated by LLMs frequently fail to meet synthesis and functional verification standards, indicating a gap in the quality of generated code.
- **C2. Limitations in Complex System Design**: While LLMs show promising capability in generating designs for simpler

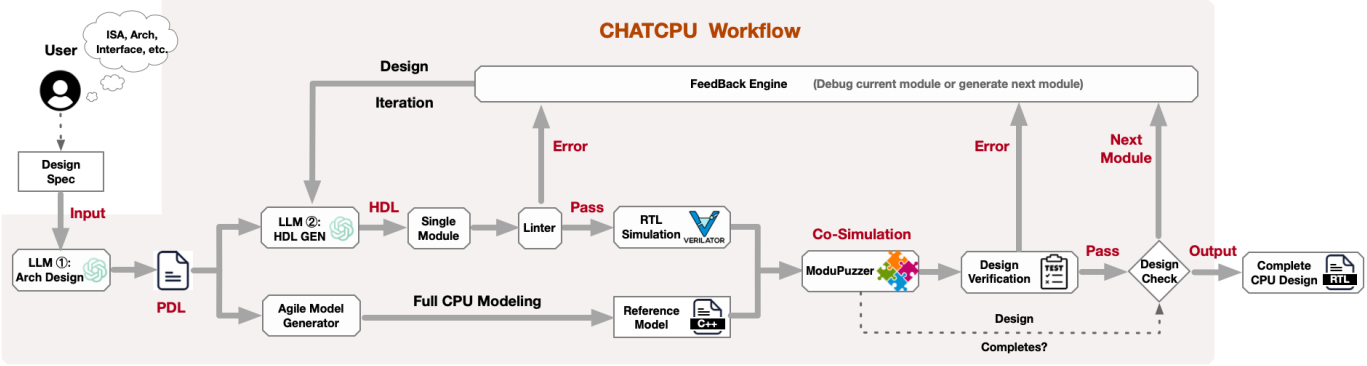


Figure 2. ChatCPU Architecture and Workflow

modules, their effectiveness diminishes with increasing complexity. This limitation becomes particularly evident in intricate hardware systems where the success rate of effective design generation by LLMs significantly decreases.

- **C3. Design Verification Challenges**: LLMs struggle to adequately address bugs within complex, interconnected modules. The limitations in token and chat history further constrain LLMs from comprehensively covering both the complete codebase and the intricate logic required in large-scale HDL designs.

3 Design & Philosophy

3.1 Solutions & Strategies

In response to the hardware design challenges (*C1–C3*) of LLMs identified in Section 2.3, we propose the ChatCPU design, incorporating the following key solutions:

- **S1. LLM Fine-Tuning**: To improve the quality of HDL code generation, we refine LLMs using high-quality training datasets derived from existing open-source hardware designs. This involves data collection, cleaning, and labeling to enable more accurate and effective HDL generation.
- **S2. Hierarchical System Design with Divide & Conquer Approach**: Addressing the complexity of CPU system design, we decompose it into smaller, manageable hardware modules. This hierarchical structuring allows for the distribution of complex design tasks across specialized LLMs, each assigned specific roles, thereby enhancing design efficiency and manageability.
- **S3. Processor Description Language (PDL)-Based Design Generation**: We introduce a novel PDL to standardize CPU design specifications. This unified language serves as the foundation for automated design and reference model generation, facilitating both the design and verification.
- **S4. Module-Interchangeable Verification Paradigm (MVP)**: To tackle the verification challenges of intricate designs produced by LLMs, we implement the MVP methodology. This approach integrates each generated module into a pre-verified reference model, enabling a comprehensive design verification flow.

3.2 HDL Selection

HLS offers the convenience of abstracting complex RTL design details through high-level languages like C/C++, but it limits control over hardware implementations and PPA tradeoffs. Besides, tracing issues from HLS to the original code is challenging, complicating debugging and verification. Similarly, while Chisel and SpinalHDL have their advantages, they present limitations in design verification, such as variable renaming and inadequate assertion support, respectively. Therefore, in this work, we select traditional HDLs (Verilog/SystemVerilog) for ChatCPU, leveraging their robustness and transparency in design and verification processes.

3.3 LLM Fine-Tuning

To enhance LLM capabilities in RTL code generation, we utilized Google Big Query to collect open-source RTL designs from GitHub, forming our primary dataset. This dataset was then processed with GPT4 to generate appropriate commands for each RTL segment, followed by deduplication and removal of irrelevant data. We trained these question-answer pairs using the Llama2-70B model, aiming to align RTL code with its functional descriptions. This approach enables the LLM to comprehend not only the RTL syntax but also the underlying functional context. Furthermore, we incorporated our PDL design specifications and detailed examples into the training set. This addition aims to boost the LLM proficiency in generating HDL and PDL content based on design specifications. During the fine-tuning phase, we employed the Low-Rank Adaptation (LoRA) technique, which targets a limited number of parameters, assuming a low "intrinsic rank" of weight changes during model tuning. This method optimizes the low-rank decomposition of weight matrices in self-attention modules, thereby streamlining the optimization process from higher to lower dimensions.

3.4 ChatCPU Workflow

As shown in Figure 2, ChatCPU employs a hierarchical design and modular generation flow with two fine-tuned LLMs featuring distinct roles. The first LLM (LLM ①) is tasked with translating design specifications into high-level architecture designs using our Processor Description Language (PDL), detailing aspects such as module lists, pipeline stages, and pin-to-pin connections. This PDL file is then fed into the Agile Model Generator (AMG) to automatically produce a cycle-accurate CPU reference model for design verification. Meanwhile, the second LLM (LLM ②) processes the PDL for

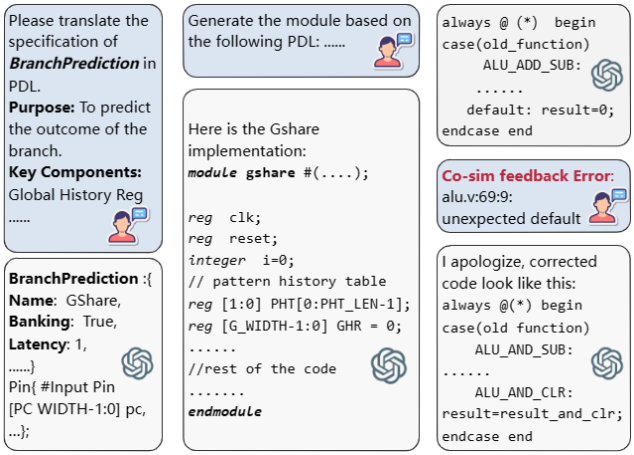


Figure 3. Prompt Examples of ChatCPU. From left to right are examples of PDL generation, HDL generation, and debugging.

each module, iteratively generating detailed HDL implementations. After a module passes linter checks, it is then verilated to a corresponding RTL model for co-simulation with the CPU reference model. This process, part of the MVP, allows for cross-validation of functionality. If the verification is successful, the design checker prompts LLM ① via a feedback engine to proceed with the next module generation. Any errors identified during code linting or design verification are reported back to LLM ② for module redesign.

Figure 3 provides examples of prompt interactions in ChatCPU. Part (a) of the figure illustrates how LLM ① receives a specification input and produces a corresponding PDL output for a Branch Prediction Unit (BPU) design. Part (b) demonstrates the design generation based on the provided PDL in LLM ②. Lastly, part (c) of the figure shows a scenario where issues are encountered during the verification process, highlighting the interactive and iterative nature of the design and verification cycle in ChatCPU.

3.5 Processor Description Language

Within ChatCPU, the processor description language (PDL) is innovatively crafted to encapsulate the timing, functionality, pipeline architecture, and pin-to-pin interfaces of each CPU module. PDL effectively translates CPU design specifications, originally in natural language, into a unified, machine-readable format. This comprehensive representation in the PDL file enables it to act as a foundational input for both the generation of HDL implementations and the construction of the CPU reference model. Figure 3(a) provides an illustrative example of PDL usage in CPU design specification.

3.6 Agile Model Generator

As depicted in Figure 4, the agile model generator (AMG) within ChatCPU initiates its process by interpreting PDL input to define the CPU timing behavior, functional characteristics, and pipeline structure. AMG facilitates interactions between modules through a well-defined timing interface, ensuring precise representation of timing relations across different modules.

The core of AMG operation is a decoupled timing and functional modeling mechanism, utilizing an event-based simulation approach to enhance both configurability and scalability. In this system, an

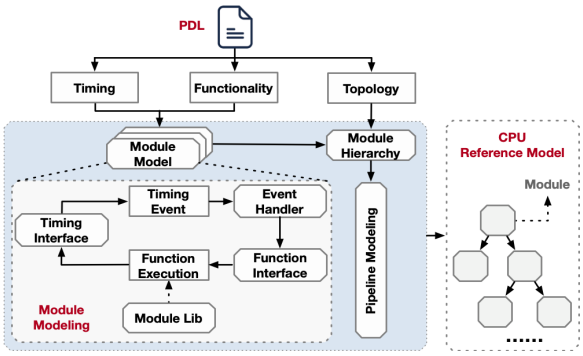


Figure 4. Agile Model Generator Workflow

event handler correlates timing events with their corresponding functional simulations via a functional interface. AMG module library houses functional implementations for simulating the behavior of classical modules in modern CPU designs. Upon completing the functional simulation, the timing interface is signaled to schedule subsequent events. The final stage of AMG's workflow involves connecting the established modules in a pipeline, culminating in the generation of a complete reference model for the CPU.

3.7 MVP Verification Flow

Verification plays a pivotal role in ASIC design, particularly for designs generated by LLMs. Addressing the verification challenge (*C3*) outlined in Section 2.3, we introduce the Module-interchangeable Verification Paradigm (MVP), specifically tailored for LLM-based CPU design. An example of this flow is shown in Figure 5.

The CPU reference model serves as the cornerstone for design verification. The reference model generated by ChatCPU offers a modularized golden model for both module-level and comprehensive CPU verification. We first ventilate each generated module to a respective RTL model for module-level simulation. The verification process starts with each LLM-generated module being translated into an RTL model using Verilator, a tool that converts RTL designs into C++ for simulation. Module-level verification is then conducted through unit tests, with results cross-validated against the equivalent module in the CPU reference model. Modules successfully passing unit tests, akin to fitting pieces in a jigsaw puzzle, are then integrated into the reference model. This integration is facilitated by co-simulation techniques such as Verilog Procedural Interface (VPI), Direct Programming Interface (DPI), or Transaction-Level Modeling (TLM), ensuring seamless module interchangeability.

Given the unified design language provided by PDL, the interchanged modules align perfectly with the reference model. The subsequent co-simulation of the verified RTL model with the reference model allows for comprehensive functionality evaluations. This includes ISA tests, torture tests, benchmarks, and real-world application simulations. As the CPU reference model is pre-verified for functional correctness, any discrepancies identified during co-simulation can be accurately attributed to the newly integrated RTL module. This targeted approach significantly streamlines the debugging process, overcoming the limitations of LLM-generated designs related to token and chat history constraints. The MVP approach thus offers a modular yet holistic verification strategy,

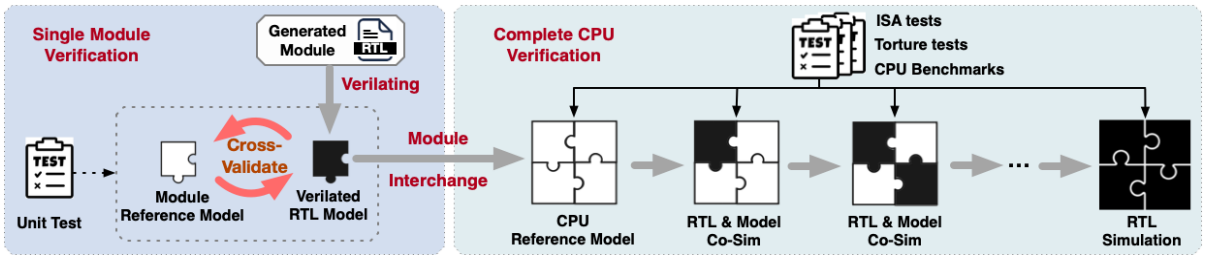


Figure 5. Example of Module-interchangeable Verification Flow: From Single Module to Complete CPU

ideally suited for the LLM-driven, modularized CPU design process, effectively addressing the complexity of verifying intricate systems.

4 Evaluations

Our experimental environment is delineated in Table 1. To validate our methodologies, we designed a 6-stage in-order RISC-V CPU: *Saber6*, with ChatCPU from scratch and completed the tapeout with SkyWater 130nm MPW project, which is so far the largest and most complicated CPU design (80K gates) generated by LLMs. The layout and detailed PPA information can be found in Figure 12.

To validate our methodologies, we designed a 6-stage in-order RISC-V CPU, named *Saber6*, using ChatCPU. This CPU, comprising approximately 80,000 gates, represents the largest and most complex CPU design generated by LLMs to date. The tapeout was completed with the SkyWater 130nm MPW project, and the layout and detailed PPA information are illustrated in Figure 12.

Table 1. Experimental Environment Specifications

Server	NVIDIA Server@Teax,US
CPU	AMD EPYC 7V13 64-Core Processor
GPU	4xA100(80GB,PCIE4.0)
Memory	256GB
CUDA	12.0
Disk	1024GB
OS	Ubuntu 22.04

4.1 Design Capability Analysis

To quantify the enhancement in design capability achieved through LLM fine-tuning, we utilized the RTLLM benchmark [11], comparing the performance of the default LLama2 and the fine-tuned LLM ② in ChatCPU. The design capability metric, as defined in Equation 1, is represented in Figure 6. Results indicate a significant improvement, with ChatCPU achieving an average design capability 2.63X higher than the base LLama2. Notably, ChatCPU was capable of generating all tested hardware modules, while the basic LLama2 was limited to smaller-scale designs.

Further, to assess the versatility of our fine-tuned LLM, we tasked LLM② with optimizing various RTL designs, as shown in Figure 7. These designs, sourced from the OpenLane project [1], included diverse hardware modules such as GCD, USB interface, Zipdiv, Picorv32, and a co-processor for encryption. After collecting baseline power and area statistics using the OpenLane EDA flow, we directed the fine-tuned LLM to optimize these designs without altering their functionality. The optimization yielded significant reductions in power and area, averaging 16.70% and 13.69%, respectively, with peaks reaching up to 36.36% and 27.72%.

4.2 Design Agility Analysis

To evaluate the acceleration in design processes offered by ChatCPU, we conducted an experiment involving 12 graduate students, divided into four groups. One group utilized ChatCPU (G1), while the others did not use any LLMs (G2-G4). The time cost for each group to complete identical CPU design tasks is visualized in Figure 8. G1 with access to ChatCPU demonstrated markedly faster completion times, especially in code correctness and verification phases. On average, ChatCPU users experienced a 3.81X time saving, with notable efficiency gains of up to 12X in RTL coding and 9.33X in design verification. These results validate the advancements our methodology offers in expediting code generation, improving verification success rates, and enhancing overall development efficiency.

4.3 Design Verification and Iterations

Our analysis also focused on the impact of the MVP on design capability. We assessed the capability of ChatCPU in generating various CPU modules in Saber6, comparing scenarios with MVP enabled and disabled (using only unit tests). The results, shown in Figure 9, reveal that while MVP leads to more iterations in simpler hardware designs (due to its thorough CPU verification flow), it significantly reduces the number of iterations required for complex designs. This reduction is particularly notable in tightly coupled modules of CPU frontend, backend, and full CPU designs, where the MVP comprehensive verification process is critical. Conversely, without MVP, even a fine-tuned LLM struggles with the challenges of cross-module verification in complex systems.

Additionally, we evaluated the impact of employing PDL in ChatCPU. Comparing the design capability with and without PDL usage (Figure 10) demonstrates that using PDL enhances design capability by an average of 2.06X over human-written specifications.

4.4 Design Complexity Analysis

Finally, to further assess design complexity, we compared the cell count of Saber6 with other LLM-generated hardware designs, including entries from the Efabless AI Design Contest [8]. As depicted in Figure 11, Saber6 exhibits a significantly higher level of complexity than the other designs in terms of the cell count. This comparison underscores the effectiveness of ChatCPU in facilitating the generation of complex hardware systems, highlighting its potential to revolutionize agile hardware design methodologies.

5 Conclusion

In this work, we first investigated the LLM-based hardware designs and summarized the associated challenges. Targeting these challenges, we proposed the ChatCPU platform to enhance the agile

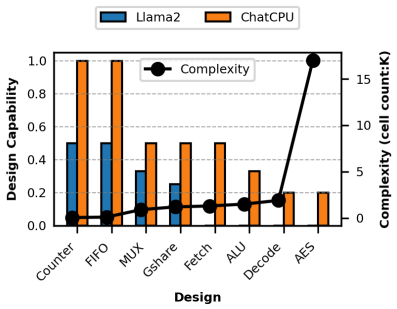


Figure 6. Impacts of LLM Fine-Tuning

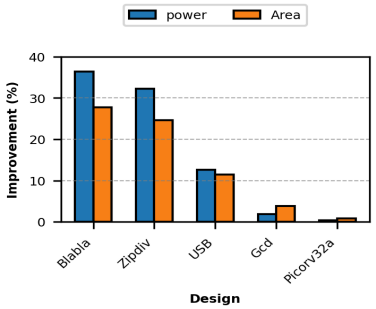


Figure 7. RTL Design Optimization

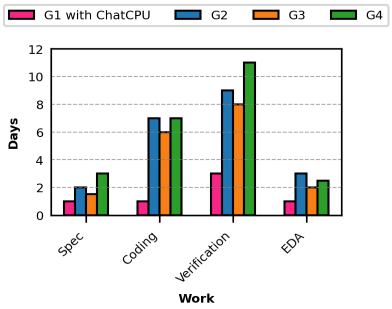


Figure 8. Time Cost Distribution

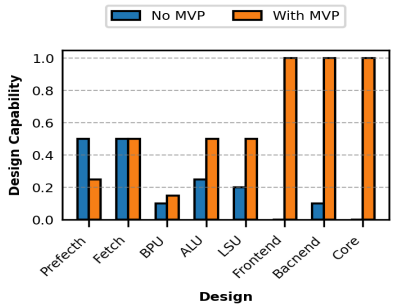


Figure 9. Verification Analysis

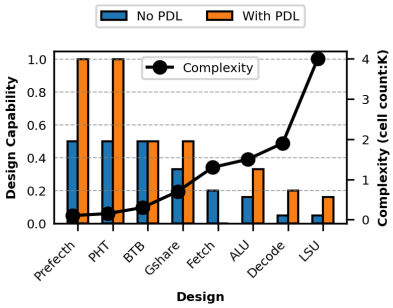


Figure 10. Impacts of PDL

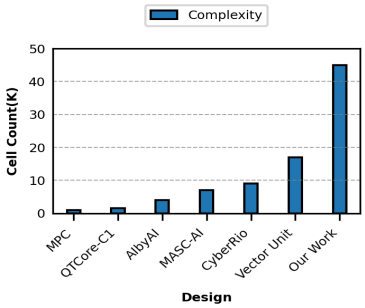
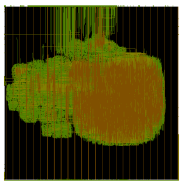


Figure 11. Complexity Comparisons



Feature	Parameters
PDK	Skywater 130nm
Gate Count	80,000
Pipeline	6
Frequency	100MHz
Total Power	19.2mW
Area	0.18mm^2

Figure 12. Silicon Layout and Information.

CPU design and verification using LLM. We proposed a hierarchical CPU design methodology with fine-tuned LLM, featuring distinct roles in the modularized CPU generation paradigm. We also introduced a Module-interchangeable Verification Paradigm (MVP) to effectively verify the hardware designs generated by LLMs. ChatCPU also integrates the OpenEDA flow to enable automated PPA optimizations, achieving an impressive boost in RTL-GDS efficiency by an average of 3.81X. Finally, we designed the largest RISC-V CPU generated by LLMs, with a scale of 80K gates using ChatCPU as a silicon prototype to validate our methodologies. exploit the transformative potential of LLMs in complex CPU designs.

References

[1] 2023. OpenLane Project. https://github.com/The-OpenROAD-Project/OpenLane.

[2] Jonathan Bachrach, Huy Vo, Brian Richards, Yunsup Lee, Andrew Waterman, Rimas Avižienis, John Wawrzynek, and Krste Asanović. 2012. Chisel: constructing hardware in a scala embedded language. In *DAC*.

[3] Jason Blocklove, Siddharth Garg, Ramesh Karri, and Hammond Pearce. 2023. Chip-Chat: Challenges and Opportunities in Conversational Hardware Design. *arXiv preprint arXiv:2305.13243* (2023).

[4] Kaiyan Chang, Ying Wang, Haimeng Ren, Mengdi Wang, Shengwen Liang, Yinhe Han, Huawei Li, and Xiaowei Li. 2023. ChipGPT: How far are we from natural language hardware design. *arXiv preprint arXiv:2305.14019* (2023).

[5] Shuyao Cheng, Pengwei Jin, Qi Guo, Zidong Du, Rui Zhang, Yunhao Tian, Hu, et al. 2023. Pushing the Limits of Machine Design: Automated CPU Design with AI. *arXiv preprint arXiv:2306.12456* (2023).

[6] Philippe Coussy and Adam Morawiec. 2010. *High-level synthesis*. Vol. 1. Springer.

[7] R Timothy Edwards. 2020. Google/SkyWater and the Promise of the Open PDK. In *Workshop on Open-Source EDA Technology*.

[8] efabless. 2023. Efabless AI Generated Open-Source Silicon Design Challenge. https://efabless.com/ai-generated-design-contest.

[9] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, et al. 2020. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155* (2020).

[10] Guyue Huang, Jingbo Hu, Yifan He, Jialong Liu, Mingyuan Ma, Zhaoyang Shen, Juejian Wu, Yuanfan Xu, et al. 2021. Machine learning for electronic design automation: A survey. *ACM TODAES* (2021).

[11] Yao Lu, Shang Liu, Qijun Zhang, and Zhiyao Xie. 2023. RTLLM: An Open-Source Benchmark for Design RTL Generation with Large Language Model. (2023).

[12] Zohar Manna and Richard Waldinger. 1979. Synthesis: Dreams→ Programs. *IEEE Transactions on Software Engineering* 4 (1979), 294–328.

[13] Katharine Sanderson. 2023. GPT-4 is here: what scientists think. *Nature* 615, 7954 (2023), 773.

[14] Mohamed Shalan and Tim Edwards. 2020. Building OpenLANE: a 130nm openroad-based tapeout-proven flow. In *ICCAD*.

[15] Nan Wu, Yuan Xie, and Cong Hao. 2022. IronMan-Pro: Multiobjective Design Space Exploration in HLS via Reinforcement Learning and Graph Neural Network-Based Modeling. *IEEE TCAD* (2022).

[16] Yinan Xu, Zihao Yu, Dan Tang, Guokai Chen, Lu Chen, Lingrui Gou, Yue Jin, Qianruo Li, Xin Li, Zuojun Li, et al. 2022. Towards developing high performance RISC-V processors using agile methodology. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, Chicago, IL, USA, 1178–1199.

[17] Xiaoling Yi, Jialin Lu, Xiankui Xiong, Dong Xu, Li Shang, and Fan Yang. 2023. Graph representation learning for microarchitecture design space exploration. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, IEEE, San Francisco, CA, USA, 1–6.

[18] Jianwang Zhai and Yici Cai. 2023. Microarchitecture Design Space Exploration via Pareto-Driven Active Learning. *IEEE TVLSI* (2023).

[19] Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2022. Large language models are human-level prompt engineers. *arXiv preprint arXiv:2211.01910* (2022).