LLM-Assisted RTL Generation and PPA Optimization for Domain-Specific Hardware Accelerators

Azahar Bernad
Department of Computer Science, University of New Hampshire, Durham, NH, USA.
helloazahar@unh.edu

Marissa Glass
Department of Electrical Engineering and Computer Science, University of Missouri, Columbia, MO, USA.
hellomarissa@missouri.edu

Niyaga Suwarno
Department of Computer Science, University of Central Florida, Orlando, FL, USA.
suwarnoniyaga@ucf.edu

Abstract

Large language models are rapidly moving from general software assistance toward specialized engineering domains, and register transfer level design has become a consequential test case for their maturity. This paper examines the system-level implications of using large language models to generate and optimize register transfer level code for domain-specific hardware accelerators, with particular attention to power, performance, and area outcomes. Rather than treating model output as an isolated artifact, the analysis positions generated hardware descriptions inside a socio-technical pipeline that includes specification interpretation, architectural exploration, verification, synthesis, physical implementation, deployment, and long-term maintenance. The paper argues that the central challenge is not simply producing syntactically valid Verilog or VHDL, but producing designs that are robust across process corners, understandable to human engineers, governable under institutional review, and sustainable in energy and economic terms. It discusses the trade-offs between aggressive optimization and verification burden, the role of domain adaptation and benchmark design, the infrastructure required for traceability and reproducibility, and the policy questions raised by automated hardware generation. Case illustrations from accelerator design, edge inference, and cloud-scale deployment show that large language models can compress early-stage design effort while also introducing new forms of technical debt, security risk, and fairness concern. The paper concludes that responsible adoption requires hybrid workflows, rigorous verification, transparent model reporting, and governance mechanisms that treat hardware generation as a matter of public consequence rather than a purely private productivity gain.

Keywords

large language models, register transfer level, hardware accelerators, power performance area, electronic design automation, verification, domain-specific architecture, technical governance, sustainability, robustness.

1. Introduction

The design of domain-specific hardware accelerators has become a defining activity of contemporary computing. As general-purpose scaling encounters physical and economic limits, architects increasingly turn to specialized pipelines for machine learning inference, cryptography, signal processing, graph analytics, and scientific simulation. This shift has been described as a new golden age for computer architecture, in which domain knowledge and hardware co-design matter more than ever [17]. At the same time, large language models have demonstrated surprising competence in code generation, program synthesis, and multi-step reasoning over textual specifications [1], [2], [3], [4]. The convergence of these trends has produced a practical question with deep systemic consequences: can large language models meaningfully assist in generating register transfer level code and in optimizing power, performance, and area for domain-specific accelerators?

The optimistic answer is that these models can lower the barrier to entry for hardware design, accelerate exploration, and help engineers navigate repetitive or well-understood portions of the design space. The cautious answer is that hardware is unforgiving. A software bug can be patched after deployment; a hardware defect may require respins, recall, or costly workarounds. A design that is merely functional may still fail because it consumes too much power, misses timing, exceeds area budgets, or cannot be verified within schedule. The use of large language models therefore cannot be evaluated only by whether they emit plausible code. It must be evaluated by whether they improve the overall system of specification, synthesis, verification, integration, deployment, and governance.

This paper argues that the meaningful unit of analysis is not the model alone but the socio-technical workflow in which the model is embedded. That workflow includes human architects, verification engineers, physical design specialists, toolchain vendors, cloud providers, standards bodies, and organizational decision makers. A generated hardware description becomes consequential only after it enters synthesis, place-and-route, timing analysis, power estimation, and formal or simulation-based verification. If those downstream stages are ignored, apparent gains in generation speed can be offset by hidden verification debt, brittle optimization, and institutional risk. The paper therefore focuses on system-level trade-offs, infrastructure requirements, deployment realities, sustainability, fairness, and policy implications.

2. From Natural Language to Register Transfer Level: Capabilities and Limits

Large language models are effective at pattern completion, code translation, and constrained generation because they learn statistical regularities across enormous corpora [1], [2]. In software domains, models such as Codex and related systems have shown that natural language prompts can be converted into executable code with reasonable success rates [3]. Program synthesis benchmarks have further clarified that performance depends heavily on problem structure, prompt formulation, and the availability of tests or examples [4]. Hardware description languages, however, differ from conventional software in ways that matter for model reliability. Register transfer level code describes concurrent hardware behavior, timing relationships, resource sharing, and physical constraints. A syntactically valid module may still imply a latch, create a combinational loop, violate clock domain assumptions, or synthesize into an unexpectedly large circuit.

Benchmarks such as RTLLM and VerilogEval have attempted to measure model performance on register transfer level tasks with more discipline than informal demonstrations allow [6], [10]. They reveal that models can produce recognizable modules for common patterns such as arithmetic units, finite state machines, and simple interfaces, but they also expose weaknesses

in handling complex specifications, hierarchical design, and timing-sensitive logic. Studies benchmarking automated Verilog generation similarly show that pass rates vary dramatically across task categories and that models often require iterative repair [9]. Domain-adapted models such as ChipNeMo indicate that specialization on chip design corpora can improve relevance and reduce some errors, but domain adaptation does not eliminate the need for verification [7]. The result is not a simple replacement of hardware engineers but a reconfiguration of their work toward specification, review, and constraint management.

A further limitation is that language models do not intrinsically understand physical reality. They do not directly experience power density, wire delay, clock skew, thermal gradients, or process variation. They can be conditioned on reports from synthesis and simulation tools, but those reports are proxies rather than embodied constraints. This creates a representational gap between textual plausibility and silicon feasibility. The gap is especially dangerous for domain-specific accelerators, which often depend on unusual dataflows, memory hierarchies, and quantization schemes that are underrepresented in public code corpora. Consequently, model usefulness is highest where the design pattern is common and the specification is precise, and lowest where the architecture is novel, the constraints are implicit, or the cost of failure is extreme.

3. Domain-Specific Accelerator Architectures and PPA Trade-Offs

Domain-specific accelerators are designed around a narrow set of operations, data types, and memory access patterns. This specialization can deliver order-of-magnitude gains in performance and energy efficiency, as seen in tensor processing units and other neural network accelerators [19]. However, specialization also increases sensitivity to design choices. A matrix multiplication engine optimized for one dataflow may be inefficient for another; a memory hierarchy tuned for one tensor shape may stall for another; a quantization strategy that reduces area may degrade accuracy or require additional calibration. Power, performance, and area are therefore not independent metrics but coupled outcomes of architectural decisions. Improving one often degrades another, and the appropriate balance depends on deployment context, workload distribution, and economic constraints.

The physical limits of scaling make these trade-offs more acute. Dark silicon research showed that not all transistors can be active simultaneously under thermal and power constraints [16], and energy efficiency has become a first-order design objective [18]. Efficient processing of deep neural networks requires careful attention to data reuse, on-chip memory, parallelism, and precision [20]. Large language models can assist by proposing candidate architectures, estimating relative costs, and generating parameterized register transfer level components. Yet the search space is enormous, and the cost function is noisy because early estimates may diverge from post-layout results. A model that optimizes for a proxy metric can produce designs that look favorable in simulation but fail in implementation. This is why PPA-aware generation must be connected to tool feedback loops rather than treated as a one-shot text generation problem.

There is also a human factors dimension. Domain-specific accelerator design often involves tacit knowledge about the target use case, the software stack, the compiler, and the deployment environment. This knowledge is not fully captured in prompts or public benchmarks. If large language models are used without mechanisms for eliciting and preserving that tacit knowledge, they may generate plausible but misaligned hardware. The most promising approaches combine model suggestions with structured architectural templates, constraint checkers, and expert review. In this hybrid model, the language model

acts as a generative assistant rather than an autonomous designer, and PPA optimization becomes a collaborative process of proposing, measuring, and refining.

4. LLM-Assisted RTL Generation in PPA Optimization Workflows

The most direct way to integrate large language models into PPA optimization is to treat them as generators of candidate register transfer level implementations that are then evaluated by synthesis and analysis tools. A recent study on power, performance, and area optimization aware Verilog code generation shows that models can be guided toward PPA objectives when prompts and feedback incorporate synthesis results [5]. This is important because it moves beyond functional correctness alone and begins to address the engineering reality that hardware must meet non-functional constraints. The same study also illustrates a central tension: aggressive PPA guidance can improve targeted metrics while making verification harder, because the generated code may use unusual constructs, rely on tool-specific behavior, or obscure design intent.

Another integration path is to use models for design-space exploration rather than final code generation. Here the model proposes parameter choices, microarchitectural variants, or interface contracts, and engineers use conventional tools to evaluate them. Benchmarks such as RTLLM and VerilogEval provide useful testbeds for this kind of assistance because they compare generated designs against reference behavior [6], [10]. However, benchmarks often simplify timing, power, and physical constraints. A design that passes a functional benchmark may still be unsuitable for a real accelerator. Therefore, PPA-aware workflows require richer evaluation harnesses that include synthesis reports, timing closure estimates, power analysis, and verification coverage. Without such harnesses, model performance metrics can create a false sense of readiness.

A third path is iterative repair and co-optimization. The model generates an initial design, tools report violations or inefficiencies, and the model proposes revisions. This loop resembles program synthesis with execution feedback, but hardware feedback is slower and more expensive than software test execution [4]. Synthesis and place-and-route can take minutes to hours, and power analysis may require detailed activity traces. The economic viability of LLM-assisted PPA optimization therefore depends on reducing iteration cost through surrogate models, early estimation, and caching of design knowledge. It also depends on governance: every iteration should be traceable, reproducible, and reviewable. Otherwise, the organization may accumulate a large set of generated variants whose provenance and rationale are unclear, creating a new form of technical debt.

5. Toolchain Integration, Verification, and Robustness

Verification is the dominant cost in many hardware projects, and large language models do not remove that cost automatically. In fact, generated code can increase verification burden if it is opaque, inconsistent, or difficult to constrain. Formal verification, simulation, emulation, and post-silicon validation all require assertions, coverage models, and corner-case analysis. A model can help write testbenches, generate assertions, or propose corner cases, but it can also generate tests that reflect the same misunderstandings as the design. Independent verification and human review remain essential. The safety literature has long warned that optimizing for measurable objectives can produce unintended behavior when the objective is incomplete [11]. Hardware generation is vulnerable to the same failure mode because functional pass rates and PPA estimates are incomplete proxies for real-world correctness.

Conversational hardware design tools such as Chip-Chat illustrate both the promise and the risk of interactive generation [8]. They lower the friction of producing initial designs, but they also make it easy to generate more code than a team can verify. The resulting verification debt is a form of hidden technical debt in machine learning systems, where rapid prototyping creates downstream maintenance and integration costs [14]. In hardware, this debt can be especially severe because late discovery of a defect may require mask respins or architectural workarounds. Robustness also includes security: generated hardware may contain unintended side channels, weak access control, or vulnerable cryptographic primitives. Because large language models are trained on public data, they may reproduce insecure patterns or omit security-critical constraints that are not explicit in the prompt.

Reliability at scale introduces additional concerns. The tail at scale shows that rare latency outliers can dominate user experience in large distributed systems [15]. Similarly, rare hardware defects or corner-case failures can dominate field returns and operational cost. A design that works in nominal simulation may fail under voltage droop, temperature variation, aging, or adversarial workload. LLM-assisted flows must therefore include stress testing, formal properties, and diversity in verification. The goal is not to prove that a model is always correct, but to build a system in which model errors are detected before they become physical failures. This requires an engineering culture that treats generated code as untrusted until verified, and an infrastructure that makes verification efficient enough to keep pace with generation.

6. Governance, Security, and Institutional Infrastructure

The governance of LLM-assisted hardware generation involves questions of accountability, traceability, and control. If a model proposes an accelerator design that later fails, who is responsible? The engineer who accepted it, the organization that deployed it, the model provider, or the team that curated the training data? Clear governance requires audit trails that record prompts, model versions, generated artifacts, tool settings, and human approvals. Model cards and related documentation practices can help by specifying intended use, limitations, and evaluation conditions [13]. However, hardware introduces additional dimensions such as process node, foundry constraints, and export-control classification. These dimensions are not typically captured in general model documentation, so domain-specific reporting frameworks are needed.

Security is another governance concern. Hardware is a foundation of trust in computing, and malicious or accidental modifications can undermine entire systems. Large language models may be susceptible to prompt injection, data poisoning, or adversarial prompts that steer generation toward vulnerable designs. The broader artificial intelligence safety literature provides concepts for specifying and mitigating such risks, but hardware-specific threat models are still maturing [11]. Organizations should adopt least-privilege access to design tools, isolate generated code before review, and require security review for interfaces that handle sensitive data. They should also consider supply-chain risks: if generated register transfer level code incorporates third-party intellectual property or copied patterns, licensing and provenance may become unclear.

Institutional infrastructure is equally important. Universities, research laboratories, and companies need shared benchmarks, reproducible toolchains, and open evaluation protocols. Benchmarks such as RTLLM and VerilogEval are steps in this direction, but they must evolve to include PPA, security, and verification coverage [6], [10]. Without shared infrastructure, each organization will invent its own ad hoc evaluation, and claims about model capability

will remain difficult to compare. Governance also includes fairness in access. If high-quality LLM-assisted design flows are available only to well-resourced firms, the benefits of specialization may concentrate, while smaller actors face higher barriers. Public research funding, open-source tooling, and educational programs can help distribute capability more broadly.

7. Deployment, Sustainability, and Fairness

Deployment context shapes what counts as a good accelerator. An edge device may prioritize energy efficiency and area, while a cloud accelerator may prioritize throughput, latency, and utilization. A safety-critical system may prioritize predictability and verification, while a consumer device may prioritize cost. Large language models can help tailor designs to these contexts, but they can also encode biases from training data that favor common commercial patterns over underrepresented use cases. The stochastic parrots critique warns that large models can reproduce and amplify biases present in their corpora [12]. In hardware, this may manifest as poor support for non-English specifications, unusual numerical formats, or application domains with less public code. Fairness therefore involves not only who gets access to the technology but also whose design requirements are legible to the model.

Sustainability is both a design objective and a cost of using the technology. Training and operating large language models consume significant energy, and using them to generate many hardware variants can multiply compute demand. At the same time, better accelerators can reduce energy consumption in deployment, especially for machine learning inference and data-intensive workloads [18], [20]. The net sustainability effect depends on whether model-assisted design leads to more efficient hardware, longer hardware lifetimes, and reduced respins, or to excessive exploration and premature obsolescence. Organizations should measure the full lifecycle impact, including model training, tool runs, fabrication, deployment, and disposal. A narrow focus on generation speed may obscure larger environmental costs.

Deployment also raises maintenance and adaptability questions. Hardware often remains in service for years, while models and toolchains evolve quickly. A design generated by a model may be difficult to modify if the rationale is not documented or if the code style is unfamiliar to human engineers. This creates a tension between rapid generation and long-term maintainability. The tail at scale again offers a lesson: systems must be designed for rare but consequential events, not only average-case performance [15]. Similarly, accelerators must be robust to workload shifts, software updates, and security patches. LLM-assisted design should therefore prioritize understandable, modular, and well-documented artifacts over maximally compact or exotic solutions.

8. Policy Implications and Cross-Domain Comparisons

Policy questions arise when automated hardware generation becomes economically significant. Export controls, intellectual property, liability, and national security all intersect with chip design. If a model can generate advanced accelerator logic from a natural language prompt, then the barrier to designing sensitive hardware may fall, raising concerns about proliferation and misuse. At the same time, overbroad restrictions could impede research, education, and legitimate innovation. Policymakers need nuanced frameworks that distinguish between general design assistance and the production of controlled capabilities. They also need to consider transparency: should model providers disclose training data sources, evaluation results, and known failure modes for hardware tasks? Should organizations be required to maintain audit trails for safety-critical or security-critical designs?

Cross-domain comparisons are instructive. In software engineering, AI-assisted coding has already changed developer workflows, but hardware has higher stakes at the physical layer. In biotechnology, dual-use research raises similar questions about automation, access, and governance. In civil infrastructure, engineering codes and certification processes provide examples of how safety-critical artifacts are regulated. Hardware design may need analogous certification regimes for certain application domains, such as medical devices, automotive systems, and critical infrastructure. These regimes would not necessarily prohibit LLM assistance, but they would require evidence of verification, traceability, and human accountability.

There is also an economic policy dimension. Domain-specific accelerators are central to competitiveness in artificial intelligence, cloud computing, and edge devices. Governments may invest in open benchmarks, shared fabrication access, and workforce training to ensure that LLM-assisted design benefits a broad ecosystem. They may also support research on verification, formal methods, and security analysis, because these are the bottlenecks that limit safe adoption. The alternative is a landscape in which generation outpaces verification, and organizations accumulate hidden risks. A balanced policy approach would encourage innovation while strengthening the public infrastructure needed to evaluate and govern it.

9. Case Illustrations and Forward-Looking Perspectives

Consider an edge inference accelerator for a specialized sensor. A small team might use a large language model to generate initial register transfer level modules for data movement, arithmetic, and control. The model can propose a parameterized design based on a textual specification, and the team can use synthesis reports to refine area and power. Benchmarks such as RTLLM and VerilogEval offer starting patterns, but the team must still verify sensor-specific interfaces, timing, and power states [6], [10]. If the model generates an elegant but opaque state machine, the team may save days in coding but spend weeks in verification. The net benefit depends on whether the organization has strong verification infrastructure and whether the design can be decomposed into well-understood modules.

Consider a cloud-scale accelerator for recommendation models. Here the design space is vast, and PPA optimization must balance throughput, latency, memory bandwidth, and utilization. A model-assisted flow could explore dataflow choices, buffer sizes, and precision formats, using feedback from performance models and synthesis tools [5]. The team might use conversational design tools to quickly generate variants, as suggested by Chip-Chat [8]. However, the deployment environment includes scheduling, multi-tenancy, and tail latency concerns that are not captured in isolated hardware benchmarks [15]. The accelerator must also fit into a software stack that compiles diverse models. This requires cross-layer co-design, not just RTL generation. The most effective use of language models may be in generating glue logic, testbenches, and documentation, while architects focus on high-level dataflow and memory strategy.

Forward-looking perspectives suggest that future systems will combine language models with formal methods, differentiable hardware generators, and reinforcement learning from tool feedback. Models may become specialized agents that call synthesis, simulation, and verification tools, then explain results to engineers. They may also be used to generate security assertions, power intent, and physical constraints. However, autonomy should be bounded by verification and governance. A mature ecosystem would include standard interfaces for model-tool interaction, reproducible evaluation suites, and clear liability frameworks. It would also include education that trains engineers to critically assess

generated hardware, not merely accept it. The goal is not to remove human expertise but to relocate it toward specification, architecture, and oversight.

10. Conclusion

Large language models offer genuine opportunities to accelerate register transfer level generation and PPA optimization for domain-specific hardware accelerators, but their value depends on the surrounding system. Models can propose code, explore design variants, generate testbenches, and help document intent, yet they cannot replace synthesis, verification, physical analysis, or human accountability. The central risk is that generation speed outpaces verification capacity, producing hidden technical debt, security vulnerabilities, and unsustainable design practices. The central opportunity is a hybrid workflow in which models handle well-bounded generative tasks while engineers and institutions provide constraints, review, and governance.

Realizing that opportunity requires investment in benchmarks, tool integration, traceability, security analysis, and reproducibility. It also requires attention to fairness and sustainability, because the benefits of automated hardware design should not concentrate only among well-resourced actors, and the environmental costs of both model training and hardware production must be measured. Policy frameworks should encourage innovation while demanding transparency and accountability for safety-critical and security-critical applications. If these conditions are met, LLM-assisted RTL generation can become a productive part of a broader movement toward domain-specific computing. If they are ignored, it may become another source of brittle automation in a field where physical failure is expensive and trust is difficult to recover.

References

1. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. Advances in Neural Information Processing Systems, 30, 5998–6008.

2. Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., ... & Amodei, D. (2020). Language models are few-shot learners. Advances in Neural Information Processing Systems, 33, 1877–1901.

3. Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., ... & Zaremba, W. (2021). Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374.

4. Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., ... & Sutton, C. (2021). Program synthesis with large language models. arXiv preprint arXiv:2108.07732.

5. Thorat, K., Zhao, J., Liu, Y., Hasan, A., Peng, H., Xie, X., ... & Ding, C. (2025, September). LLM-VeriPPA: Power, Performance, and Area Optimization aware Verilog Code Generation with Large Language Models. In 2025 ACM/IEEE 7th Symposium on Machine Learning for CAD (MLCAD) (pp. 1-7). IEEE.

6. Lu, Y., Liu, S., Zhang, Q., & Xie, Z. (2024). RTLLM: An open-source benchmark for design RTL generation with large language model. In 2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC) (pp. 722–727). IEEE.

7. Liu, M., Ene, N., Kiran, R., Das, D., Shah, A., Hofmann, A., ... & Kan, E. C. (2023). ChipNeMo: Domain-adapted LLMs for chip design. arXiv preprint arXiv:2311.00176.

8. Blocklove, J., Garg, S., Karri, R., & Pearce, H. (2023). Chip-Chat: Challenges and opportunities in conversational hardware design. In 2023 ACM/IEEE 5th Workshop on Machine Learning for CAD (MLCAD) (pp. 1–6). IEEE.

9. Thakur, S., Ahmad, B., Fan, Z., Pearce, H., Tan, B., Karri, R., Dolan-Gavitt, B., & Garg, S. (2023). Benchmarking large language models for automated Verilog RTL code generation. In 2023 Design, Automation & Test in Europe Conference & Exhibition (DATE) (pp. 1–6). IEEE.

10. Liu, M., Pinckney, N., Khailany, B., & Ren, H. (2023). VerilogEval: Evaluating large language models for Verilog code generation. In 2023 IEEE/ACM International Conference on Computer-Aided Design (ICCAD) (pp. 1–8). IEEE.

11. Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., & Mané, D. (2016). Concrete problems in AI safety. arXiv preprint arXiv:1606.06565.

12. Bender, E. M., Gebru, T., McMillan-Major, A., & Shmitchell, S. (2021). On the dangers of stochastic parrots: Can language models be too big? In Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency (pp. 610–623). ACM.

13. Mitchell, M., Wu, S., Zaldivar, A., Barnes, P., Vasserman, L., Hutchinson, B., ... & Gebru, T. (2019). Model cards for model reporting. In Proceedings of the Conference on Fairness, Accountability, and Transparency (pp. 220–229). ACM.

14. Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., ... & Dennison, D. (2015). Hidden technical debt in machine learning systems. Advances in Neural Information Processing Systems, 28, 2503–2511.

15. Dean, J., & Barroso, L. A. (2013). The tail at scale. Communications of the ACM, 56(2), 74–80.

16. Esmaeilzadeh, H., Blem, E., St. Amant, R., Sankaralingam, K., & Burger, D. (2011). Dark silicon and the end of multicore scaling. In 2011 38th Annual International Symposium on Computer Architecture (ISCA) (pp. 365–376). IEEE/ACM.

17. Hennessy, J. L., & Patterson, D. A. (2019). A new golden age for computer architecture. Communications of the ACM, 62(2), 48–60.

18. Horowitz, M. (2014). Computing's energy problem (and what we can do about it). In 2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC) (pp. 10–14). IEEE.

19. Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., ... & Yoon, D. H. (2017). In-datacenter performance analysis of a tensor processing unit. In 2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA) (pp. 1–12). ACM/IEEE.

20. Sze, V., Chen, Y.-H., Yang, T.-J., & Emer, J. S. (2017). Efficient processing of deep neural networks: A tutorial and survey. Proceedings of the IEEE, 105(12), 2295–2329.