arXiv:2603.05489v1 [cs.AR] 5 Mar 2026

NL2GDS：面向开源芯片设计的 LLM 辅助接口

Max Eland [1], **Jeyan Thiyagalingam**[2], **Dinesh Pamunuwa**[1], **和 Roshan Weerasekera**[1]
布里斯托大学电气、电子与机械工程学院（EEME），英国布里斯托尔

[2]英国科学技术设施委员会（STFC）卢瑟福·阿普尔顿实验室，
迪德科特；通讯作者：roshan.weerasekera@bristol.ac.uk

摘要

硬件设计日益复杂，高级规格与寄存器传输级（RTL）实现之间的差距不断扩大，这些因素阻碍了快速原型设计和系统设计。我们推出 NL2GDS（自然语言到版图），这是一个创新框架，利用大语言模型（LLM）将自然语言硬件描述转换为可综合 RTL，并通过开源 OpenLane ASIC流程生成完整的 GDSII 版图。NL2GDS 采用模块化流水线，用于捕捉非正式的设计意图、借助多个大语言模型引擎生成并验证硬件描述语言（HDL），以及协调自动化综合与版图生成。在 ISCAS'85 和 ISCAS'89 基准设计上的评估结果表明，与基线设计相比，该框架最多可实现 36% 的面积缩减、35% 的延迟缩减和 70% 的功耗节省，彰显了其推动 ASIC设计普及并加速硬件创新的潜力。

关键词： 自动化ASIC设计流程 · 生成式人工智能 · 硬件综合 · 大语言模型（LLM） ·自然语言到硬件 · 开源 ·寄存器传输级与GDSII生成

1 引言

自然语言处理（NLP）和大语言模型（LLM）的最新进展表明，以通俗语言表达的用户意图可以转化为可执行的软件制品 [1, 2]。在软件工程领域，自然语言编程已展现出降低入门门槛、提高生产力并实现特定领域自动化的潜力。将这些能力拓展至电子设计自动化（EDA）和专用集成电路（ASIC）设计，有望通过为复杂的后端流程提供直观的接口，从根本上重塑硬件开发工作流。

ASIC设计本质上是一个多阶段工作流，涵盖从高层硬件规范到可用于制造的物理版图。典型阶段包括寄存器传输级（RTL）设计、逻辑综合、布局规划、布局、布线、时序收敛和版图验证。传统上，这些阶段依赖专有EDA工具链，其成本往往高得令人难以承受，限制了许多中小企业（SME）和教育机构的使用。OpenLANE [3, 4] 等开源EDA框架已成为可行的替代方案，提供端到端ASIC实现流程。OpenLANE支持自动化综合、布局规划、布局、时钟树综合和布线，可显著降低硅原型设计的成本和资源需求。

在本工作中，我们介绍**NL2GDS**，这是一个可直接根据自然语言规范为数字设计生成芯片版图的框架。NL2GDS将基于大语言模型（LLM）的前端与OpenLane物理设计流程 [3, 4]相结合。用户可以用通俗语言指定硬件需求，系统会自动将其转换为可综合硬件描述语言，为OpenLane进行配置，并生成完整的GDSII版图。通过抽象底层实现细节，NL2GDS加快了从概念到版图的进程，支持快速原型设计，并服务于教育应用。整体工作流程如图1所示。

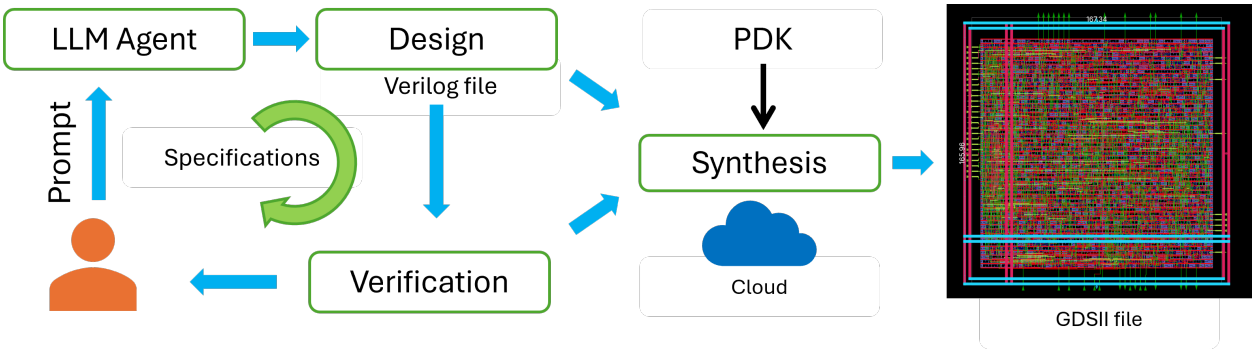


图 1：NL2GDS 流程

我们的工作标志着在推动ASIC设计民主化、实现敏捷硬件创新以及扩大硅原型设计可及性方面迈出了重要一步

1. **自然语言到ASIC流程：**介绍 NL2GDS，这是首个利用开源 OpenLANE 流程，将自然语言硬件规范转换为可综合RTL和完整 GDSII 版图的框架。2. **模块化大语言模型驱动流程：**采用多智能体架构，结合检索增强生成（RAG） [5] 和思维链推理，自动生成硬件描述语言、纠正错误并优化设计参数。3. **端到端自动化：**将硬件描述语言生成、验证、综合和版图设计整合到完全自动化的基于云的流程中，缩短周转时间并降低 ASIC 设计门槛。4. **性能提升：**在 ISCAS'85 [6] 和 ISCAS'89 [7] 基准测试中，面积缩减最高达 36%、延迟缩减达 35%、功耗节省达 70%，表现与门级优化设计相比具有竞争力。

本文其余部分安排如下。第 2 节回顾相关工作。第 3 节介绍系统架构，第 4 节描述实现。第 5 节报告案例研究，第 6 节总结全文并展望未来工作。

2 相关工作与研究动机

大语言模型的快速发展激发了研究界将自然语言驱动的自动化应用于硬件设计的浓厚兴趣 [8]。先前的工作主要涵盖：（1）基于大语言模型的硬件描述语言生成以及感知EDA工具的迭代优化；（2）端到端EDA流程编排。尽管这些工作证明了将大语言模型用作设计副驾驶的可行性，但它们都没有直接解决这一挑战：使用开源工具，直接根据自然语言描述生成可制造的 GDSII 版图。下文将回顾这些领域，并指出促使我们开展本研究的空白。

2.1 基于大语言模型的硬件描述语言生成

Chip-Chat [9] 提出了一种用于硬件设计的对话式接口，展示了如何通过与大语言模型进行自然语言交互来指导硬件描述语言代码的综合与修改。该系统强调人在环中的设计，使设计能够通过对话进行迭代优化，并凸显了大语言模型作为设计副驾驶的潜力。Chip-Chat 指出，将自然语言意图映射为硬件结构面临的关键挑战，包括在设计对话中保持上下文，以及将用户反馈整合到设计流程中。尽管该框架主要作为概念验证而实现，但它为交互式、由大语言模型驱动的硬件设计奠定了基础，也展示了利用对话系统提高设计可及性并加快设计迭代的可行性。 [8] 提出的工作介绍了一种由大语言模型辅助的硬件设计工作流，其中包括寄存器传输级生成、测试平台综合和调试支持。他们的工作表明，大语言模型能够自动化许多前端设计任务，但寄存器传输级仍是自动化的上限。
Nakkab等人于 [10] 提出一种分层提示策略，由大语言模型将硬件规范分解为子模块，并逐一综合各个组件。这提升了设计模块化程度和正确性，从而能够根据自然语言输入生成高质量的寄存器传输级代码。尽管取得了这些进展，这些方法仍仅关注前端设计，没有一种整合下游物理设计或 GDS生成。

2.2 端到端电子设计自动化流程编排

近期研究开始将大语言模型与完整的 RTL 到 GDS 流程连接起来。其中最值得关注的是 MCP4EDA [11]，它引入了模型上下文协议（MCP）接口，使大语言模型能够通过结构化工具调用来调用开源电子设计自动化工具（Yosys、OpenLane、KLayout 等）。MCP4EDA 展示了对话式设计自动化的潜力：用户可以通过自然语言请求综合、仿真和版图操作，而大语言模型则可以提供综合脚本、版图检查或指标查询。这标志着首个有据可查的尝试，旨在赋予大语言模型对完整流程的操作控制能力。然而，MCP4EDA 仍是一个工具编排系统，而非真正的自然语言到版图综合引擎。它假定已有寄存器传输级实现，并依赖用户意图来指导流程命令。它既不会根据高层次自然语言生成完整设计，也不会对目标电路执行版图驱动优化。

2.3 研究动机与贡献

尽管进展迅速，现有文献仍显示出两个主要缺口：**生成式流程缺乏后端集成**，以及**缺乏直接的自然语言到 GDS 综合**。现有系统能够生成寄存器传输级代码，却无法将自然语言转换为可制造版图，也无法将版图级反馈（时序、布线、设计规则检查）纳入生成循环。此前没有研究展示过通过端到端的大语言模型驱动流程，将自然语言行为描述自动转换为完整 GDSII 版图的能力。这些缺口表明，我们需要一种方法，将（i）高层自然语言推理、（ii）硬件描述语言综合、（iii）基于 OpenLane 的后端物理设计，以及（iv）工具感知的迭代优化结合起来，并统一于单一自动化流程中。

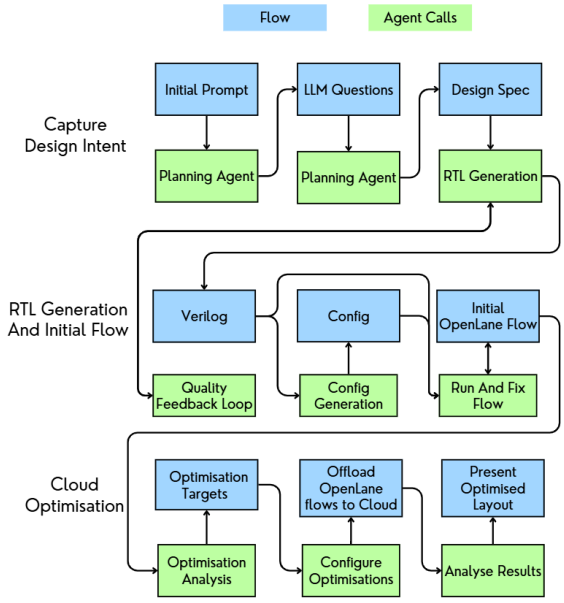


图 2：NL2GDS 框图

与以往工作不同，我们的系统首次提供了基于 OpenLane 流程的端到端自然语言到 GDSII 设计流程。具体而言，我们的贡献如下：

- **完整生成式综合**：我们直接根据自然语言提示生成寄存器传输级代码、约束和物理设计配置。
- **面向后端的迭代优化**：大语言模型读取 OpenLane 指标，并对寄存器传输级代码和流程配置进行迭代优化。
- **统一的对话式接口**：不同于基于 MCP 的编排 [11]，我们的系统在单一的自然语言循环中集成了生成、工具调用和物理优化。
- **普及芯片设计**：即使用户缺乏硬件描述语言或电子设计自动化专业知识，也能仅凭高层次描述生成可制造版图。

通过这种方式，我们的系统弥合了自然语言规范与可制造芯片版图之间长期存在的脱节，拓展了大语言模型驱动的设计自动化前沿，并解决了以往自然语言到RTL和仅工具编排框架的关键局限。

3 系统概述

NL2GDS 架构（见图 2）是一个完全模块化、由人工智能驱动的平台，可自动完成整个 RTL到GDSII硬件设计流程。我们利用大语言模型驱动的智能体进行编排，并开发了特定的工具调用，使每个智能体能够与 OpenLane流程和云端处理资源交互。该系统由四个紧密集成的核心组件组成，每个组件都针对传统基于脚本的电子设计自动化的特定障碍。

3.1 基于网页的前端

图 3 展示了首个专为 OpenLane 打造的浏览器界面，可与 KLayout 和 OpenROAD 的内部图形用户界面无缝集成。该界面使用 Streamlit Python 库开发，消除了命令行界面/脚本编写的复杂性，从而实现更快的设计迭代与优化。我们降低了入门知识门槛，同时仍能产出高质量、可流片的设计。该平台已完全 Docker 化并在云端运行，使用户无论操作系统或硬件如何，都能从任意地点访问该工具。

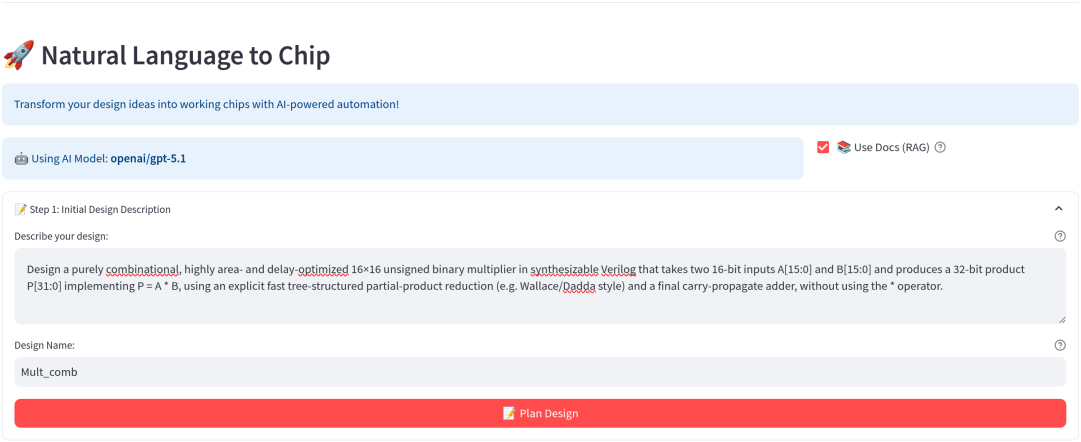


图 3：NL2GDS 前端初始提示

3.2 智能体方法

NL2GDS 系统利用一组专门的智能体来自动化设计流程的每个阶段，与单次大语言模型调用相比，能够生成质量更高的输出。这些智能体与 OpenLane 交互，从综合、布局规划、时序以及功耗、性能和面积（PPA）日志中收集上下文信息。每次 OpenLane 运行都会生成数千个文件，因此大语言模型对于有效综合这些知识至关重要。这些智能体经过专门优化，能够理解用户意图、生成硬件描述语言、修复 OpenLane流程特有的错误，并提供布局后优化建议。它们具备检索增强生成上下文检查、文件写入/读取权限，以及与 OpenLane流程各个不同部分交互等工具调用能力。

3.3 检索增强生成

检索增强生成使大语言模型能够访问其固定上下文窗口之外的相关信息。NL2GDS 系统利用检索增强生成访问项目专属文档和成功设计，从而改进 OpenLane 内配置文件的生成。基础大语言模型并不充分了解控制 OpenLane 流程的约 800 个设计参数。通过为其提供对这些参数的定向访问，我们在处理新设计时大幅提高了可靠性和创造性。这也使系统能够针对 OpenLane错误代码修复错误，并通过智能重新生成来解决这些问题。

3.4 云端后端

整个流程完全自动化，无需设计人员介入，即可从自然语言生成可投片版图。系统优先节省用户时间，并自动修复流程中出现的错误。系统利用云计算实现快速设计探索，涵盖参数扫描和版图优化。设计人员设定功耗、性能和面积目标，系统则协调并行运行，以迭代方式优化设计。这使缺乏本地计算资源的用户也能以极低成本进行硅原型设计。

4 实现

此实现解决了智能体式电子设计自动化流程中常见的三个主要瓶颈：Verilog 生成、缺少流程上下文，以及阻碍优化进展的计算资源限制。

4.1 思维链

思维链（CoT）过程是智能体应用的基础。任务不会拘泥于初始响应，而是被拆解为更小、更易管理的组成部分，从而能够完成一系列子任务，共同实现更广泛的目标。这种方法将线性工作流转变为自适应流程，并通过将任务序列拆分为明确的任务，更好地控制工具调用。系统在规划、Verilog 生成、验证、自动错误修正和优化过程中均采用了思维链。事实证明，思维链方法能够显著提升人工智能系统的性能。例如，思维链提示显著提高了代码质量，在 HumanEval 基准测试中，Pass@1准确率相对提升了 63% [12]。

我们采用经过优化的规划智能体来捕捉用户意图。整个流程从高层次的设计提示开始，随后由大语言模型围绕功能、输入和输出、架构以及 PPA 优化优先级，向用户提出有针对性的问题。大语言模型不会接管设计流程，而是与用户协作，直到充分理解需求。随后，大语言模型会为 Verilog生成智能体量身定制一份符合其能力的设计规范。这种方法已证明能够对精确的规范作出快速响应。对于 Verilog 生成，我们实现了一个反馈循环，由独立智能体执行额外的逻辑检查，并自动调用 Verilator 的代码检查模式。此流程可以识别可能导致 OpenLane流程无法执行的编码错误、不良实践、兼容性问题及其他错误。只有代码经过全面检查并确认无误后，才会提交给用户验证，然后再进行综合。我们使用 VerilogEval数据集中的 10 个具有挑战性的测试用例子集 [13] 对此实现进行了验证。该子集重点关注需要生成复杂有限状态机的问题（例如 fsm_hdlc、conwaylife 和 fsm_ps2data）、处理宽向量运算（popcount255），以及实现特定硬件算法（例如 gshare分支预测器和 lfsr32）。

最初使用 Gemini 2.5 Pro，在采用和未采用 NL2GDS 架构的情况下，分别对该子集进行了思维链实现评估。与基准模型相比，Gemini 2.5 Pro 与 NL2GDS 的结合使该子集中的测试平台检查成功率提高了 11%。随后，选取 VerilogEval数据集中的提示词 256 进行演示。该提示词被认为是最具挑战性的有限状态机任务之一，包含 200,000 次测试平台检查，可用于展示截至撰写本文时该方法在主流大语言模型中的通用性。表1 将基线成功率定义为测试平台检查成功的百分比，并将其与对每个大语言模型应用 NL2GDS 架构后取得的性能进行比较。结果表明，NL2GDS 能够持续改进所有模型的 Verilog 生成效果。

表1：采用和未采用 NL2GDS 思维链时生成的 Verilog 的测试平台通过率。

模型	基线通过率（%）	采用 NL2GDS 思维链（%）
Gemini 2.5 Pro	3.79	83.70
GPT-5	17.52	100.00
Sonnet 4.5	99.88	100.00
Grok-4 Fast	0.00	18.01
MiniMax M2	0.00	95.62

结果表明，代码生成能力有了显著提升，尤其是在遵循有关功能和输入/输出命名的精确指令方面，从而能够根据自然语言提示满足未公开测试平台的要求。此外，即使基础模型未能生成代码，该模型仍生成了语法正确的代码，展现出强大的自我纠错能力。

4.2 NL2GDS 流程中的检索增强生成

大语言模型对 OpenLane EDA 流程缺乏充分理解。由于缺少语法知识，也不了解各个参数对设计的影响，它们难以生成启动该流程所需的可综合配置文件。为了解决这一问题，我们使用了针对特定设计的检索增强生成系统，将相关的 OpenLane 文档、参数指南和成功示例注入提示中。

对于初始运行，我们从生成的 Verilog 和设计规范中解析指标，并将其输入一个初始算法，该算法检测复杂度以及为实现功耗、性能和面积所需的关键设计规范参数。该过程查询成功的设计，以生成领域特定的初始配置。流程由大语言模型编排并固定，直至完成。至此，我们已获得开始优化所需的日志。

我们通过检测算子定义关键设计问题的提取：

$$\mathcal{I} = f_{\text{detect}}(\mathcal{M}, \mathcal{E}) = \{i_1, i_2, \ldots, i_n\} \tag{1}$$

其中 $\mathcal{I}$ 编码 s 从提取的指标($\mathcal{M}$)和错误($\mathcal{E}$中检测到的时序、面积、布线及设计规则检查违规). f_{detect} 的实现包含启发式规则，这些规则将负的建立时间裕量或保持时间裕量识别为时序违规，将高布局利用率识别为潜在的面积拥塞，并将非零的设计规则检查/版图与原理图一致性检查计数识别为物理或连接性错误。每个检测到的问题都会按其类别、严重程度以及在流程中的位置进行编码。这些结构化问题（$\mathcal{I}$）连同当前配置（$\mathcal{C}$）以及（若已指定）用户定义的优化目标（$\mathcal{G}$），被用于生成定向查询：

$$\mathcal{Q} = f_{\text{query}}(\mathcal{I}, \mathcal{C}, G) = \{q_1, q_2, \ldots, q_k\} \tag{2}$$

查询的制定旨在检索与特定问题直接相关的文档和经过验证的解决方案，例如“OpenLane 时序优化 CLOCK_PERIOD 违规”或“通过 FP_CORE_UTIL 实现面积缩减”。对于每个查询，系统会呈现被引用最频繁的前 $\mathcal{N}$ 个参数说明、文档片段和配置值。

$$\mathcal{D}_{q_j} = \text{Retrieve}(q_j, \mathcal{N}), \quad \mathcal{D}_{\text{RAG}} = \bigcup_{j=1}^{k} \mathcal{D}_{q_j} \tag{3}$$

检索到的文档的组合集合 $\mathcal{D}_{\text{RAG}}$，连同运行指标、日志、配置状态和优化历史，被纳入提示词，以使大语言模型能够生成语法有效且具备设计感知的配置更改。

$$\mathcal{O}_{RAG} = \text{Concat}\left(\mathcal{M},\, \mathcal{E},\, \mathcal{H},\, \mathcal{C},\, \mathcal{D}_{\text{RAG}},\, \mathcal{G}\right) \tag{4}$$

4.3 并行性

传统上，OpenLane 设计为通过命令行作为独立的操作系统进程运行。NL2GDS 在很大程度上依赖 Python 来编排整个流程；然而，依赖 Python 的全局解释器锁（GIL）可能会限制 CPU 密集型工作负载的真正并行性 [14]。由于 OpenLane 本身是一个外部二进制程序，我们将其与 Python 的运行时限制解耦。鉴于存在大量读写操作，我们采用 I/O 并行方法进行编排。这种方法只需处理数据传输，而不必执行计算密集型的电子设计自动化任务。随后，我们将每次 OpenLane 运行作为子进程启动，使其在多核服务器或云节点上同时运行；每个进程都消耗大量资源并独立推进，不受 Python 线程模型的限制。

在 OpenLane 的无符号串行/并行乘法器（SPM）设计上共执行了 100 个流程。在串行模式下，每次运行完成该流程需要 132 秒。并行实现的平均运行时间为每个流程 37.5 秒，相比串行执行速度提升了 3.46 倍。此外，所有设计配置均由优化代理自动生成并启动，该代理会分析目标日志以识别潜在改进。若计入设计人员通常分析日志并修改配置所需的人工时间，整体生产力提升将更为显著。

5 评估

本节介绍 NL2GDS 流程的 PPA 优化能力，以及其可生成设计的复杂度。评估采用开源 Skywater 130nm 工艺设计套件。一组 ISCAS Verilog 基准设计（包括组合电路和时序电路）作为评估基础。我们将既有的 ISCAS基准电路与通过 NL2GDS 流程生成的设计相结合，作为 OpenLane 物理设计工具链的输入。这使我们能够依据硅面积、关键路径延迟和估计功耗等基本实现指标，系统地刻画每项设计。通过将这些可量化指标作为评估依据，我们建立了一个与本研究更广泛目标直接契合的稳健框架。该评估框架聚焦于与我们主要目标一致的可衡量结果：仅凭自然语言提示即可与专有基准测试竞争、缩短布局时间，并以显著更低的成本实现优化设计。

5.1 ISCAS 与 NL2GDS

通过提供指令以生成功能等效的 ISCAS 电路，对 NL2GDS 所生成设计的质量进行了评估。每个 NL2GDS 设计都被明确要求保持相同的输入与输出规格及功能，并由专用测试平台加以验证。ISCAS 电路先经过 OpenLane流程处理，随后使用 NL2GDS 生成功能等效的设计。为评估性能，我们采用了 $\mathcal{R}$ 所给出的 NL2GDS 性能与 ISCAS 性能之比。比值低于 1 表明，在相应指标上 NL2GDS 优于 ISCAS 实现。

$$\mathcal{R} = \frac{\text{NL2GDS}}{\text{ISCAS}} \tag{5}$$

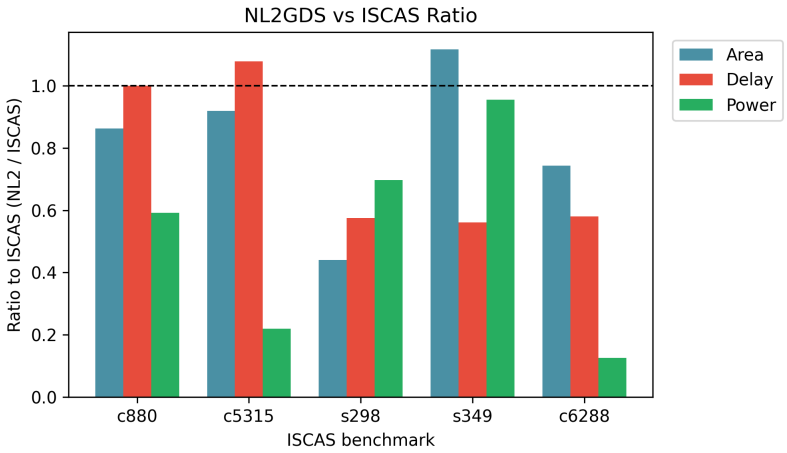
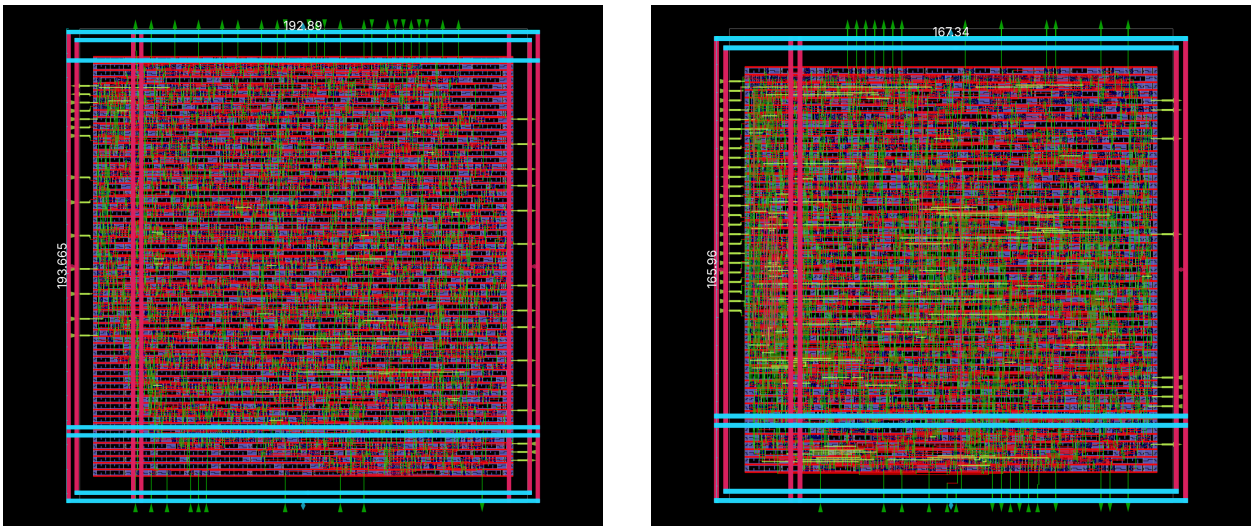


图 4：NL2GDS 与 ISCAS 在面积、延迟和功耗指标方面的比较

NL2GDS 系统不仅生成了设计，还通过自我优化为每个设计确定了优化后的布局。ISCAS 设计经过高度优化，并以门级 Verilog 编写。在全部 15 个实例中，除两个实例外，NL2GDS 在每一项设计的所有 PPA 指标上都取得了更好的结果。例如，图 5a 和 5b 展示了来自 ISCAS基准电路 [6] 的两个 c6288 设计；与 ISCAS基准实现相比，NL2GDS 系统将芯片面积减少了 25.7%。

5.2 NL2GDS 自我优化

从这里开始，我们扩展了测试案例，纳入了更复杂的设计。表 3 展示了 OpenLane 运行次数，以及从初始提示到完成优化后的布局所需的总时间。用户通过回答大语言模型的询问，向其指定完整功能，从而无需网表。结果表明，与传统开发方法所需的时间相比，快速优化仅需其中很小一部分时间即可实现。此外，整个工作流程产生的计算成本和大语言模型相关成本都极低。



（a）ISCAS 输入　　（b）NL2GDS 生成的 Verilog

图 5：在 OpenROAD 中使用（a）ISCAS 输入和（b）NL2GDS 生成的 Verilog代码，为 16×16 组合乘法器生成的版图（芯片尺寸：192.89um × 193.665um）和（167.34um × 165.96um）。

表 2：基线与优化后的NL2GDS设计的综合结果

设计	面积（A），单位：μm²			延迟（t_d）单位：ps			功耗（P），单位：μW)		
	基准-line	**NL2GDS**	**ΔA (%)**	**基础版-line**	**NL2GDS**	**Δt (%)**	**基础版-line**	**NL2GDS**	**ΔP (%)**
组合式8位算术逻辑单元（C880）	7229	4650	-35.68	15284	9955	-34.86	77	32	-58.44
16位错误检测与纠正器（C2670）	5833	3883	-33.43	15046	9877	-34.35	266	81	-69.55
9位算术逻辑单元（C5315）	67600	62500	-7.54	18284	12955	-29.15	127	112	-11.81
时序交通信号灯控制器（S298）	1727	1564	-9.44	6970	5737	-17.69	98	64	-34.69
4×4加移位乘法器（S349）	5863	4472	-23.73	8141	5607	-31.13	206	132	-35.92
数字分数乘法器（S838）	29576	19683	-33.45	12268	9758	-20.46	3319	1974	-40.52
16×16乘法器（C6288）	61317	27771	-54.71	21607	17765	-17.78	1505	1213	-19.40
16×16乘法器（流水线）	90979	58071	-36.17	11720	7584	-35.29	4521	2409	-46.72

表 3：NL2GDS 每项设计的总运行次数和成本。

设计	Runs	时间 [分钟]	大语言模型 +计算成本 [$]
C880	75	20	0.48
C2670	100	14	0.32
C5315	72	26	0.53
S298	85	12	0.13
S349	75	18	0.27
S838	75	26	0.51
16x16乘法器（C6288）	50	28	0.56
16x16乘法器（流水线）	48	30	0.56

基准电路的设计指标如表 2 所示，展示了使用 NL2GDS 系统进行快速设计探索的能力。基线是 OpenLane流程首次成功运行的结果，而优化后的设计则是在表 3 所示的云端运行次数之后获得的、具有功耗、性能和面积指标的设计。

所有设计的芯片面积均有所减少，降幅从 7.54%（9位ALU）到 54.71%（16×16 乘法器）不等。较大的算术电路，尤其是乘法器，受益最大，降幅超过 35%；而较小的面向控制的设计则仅有适度提升。时序也有显著改善，降幅介于 17.69% 至 35.29% 之间。功耗节省相当可观，幅度从 11.81% 到 69.55% 不等。乘法器和 ALU 等算术密集型设计的功耗通常降低了 35% 以上，这证实了 NL2GDS 在降低动态功耗方面的有效性。

5.3 乘法器案例研究

这两种乘法器设计展示了快速构建不同架构原型的能力。其中一种设计实现为纯组合电路，直接针对ISCAS基准。随后，系统在保持功能不变的前提下，按照指示引入流水线化以提高速度，并接受面积和开关功耗增加作为权衡。结果表明，时序设计的时钟周期缩短了 10ns；由于增加了流水线电路，功耗增加了一倍，面积则扩大至原来的 2.09 倍。这些结果符合预期的设计权衡。完成这些测试所需的总时间仅为一小时，其中包括自然语言输入、流片生成，以及针对两种设计进行的 98 次优化运行。此外，大语言模型和云服务的总费用为 1.12 美元。这些发现凸显了 NL2GDS 系统的巨大潜力：设计人员和研究人员无需编写代码，即可高效构建原型、评估设计权衡，并全面考察各项指标。

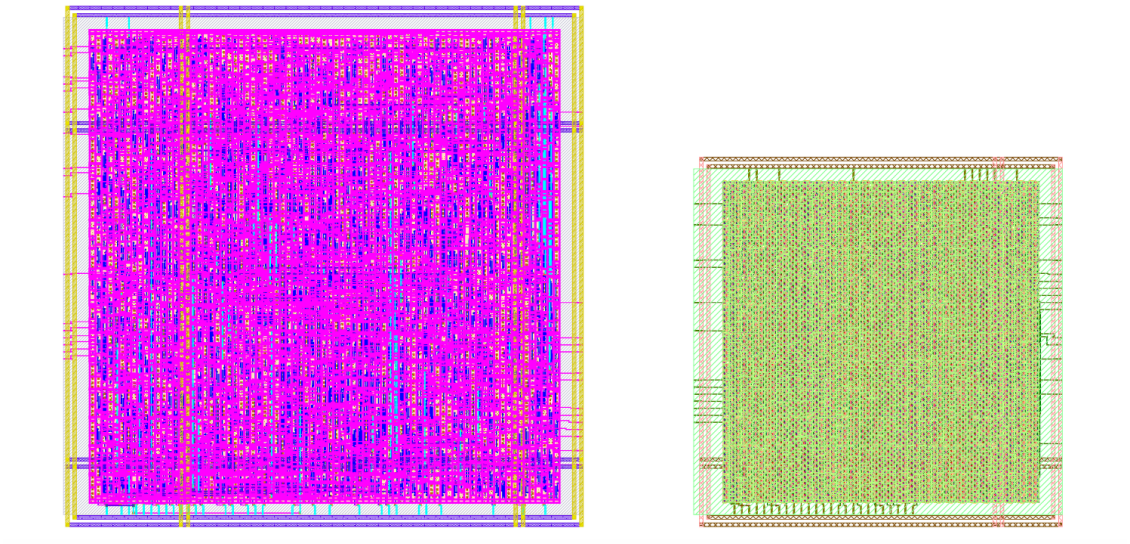


图6：在 KLayout 中生成的流水线与组合式 16×16乘法器的 GDSII 版图

6 结论

所提出的 NL2GDS 流程展示了将自然语言规格快速转换为 GDSII 版图的能力，弥合了高层设计意图与物理实现之间的差距。它能够生成可靠的 Verilog代码，提升最先进模型的性能，并引入了一种用于生成优化配置文件的新型基于RAG的系统——这是大语言模型此前尚未实现的能力。结合为实现与 OpenLane流程无缝交互而新开发的工具调用，NL2GDS 能够创建经过优化的云端设计，其性能可与门级优化的ISCAS基准相媲美。

实验结果证实，NL2GDS 能够有效实现紧凑、节能且高性能的设计。面积缩减幅度为 25.29% 至 54.43%，功耗节省幅度为 11.81% 至 69.55%；其中，乘法器和 ALU 等算术密集型电路的缩减幅度超过 35%。关键路径延迟也显著降低，降幅为 25.29% 至 44.16%，这表明 NL2GDS 不仅降低了动态功耗，还提升了时序性能。总而言之，NL2GDS 代表了一种变革性的自动化硬件设计方法，尤其适用于对效率和快速部署有较高要求的边缘人工智能和物联网应用。

7 致谢

我们衷心感谢布里斯托大学暑期实习计划为 Max Eland 提供的资助支持。

本研究还得到了 i-EDGE 项目的支持。该项目获得了欧盟（资助编号 101092018）、瑞士联邦教育、研究与创新国务秘书处（SERI）以及英国研究与创新署（UKRI）通过英国政府地平线欧洲资助保障机制提供的资助（资助编号 10061130and 10063023）。

参考文献

[1] M. Chen、J. Tworek、H. Jun、Q. Yuan、H. P. de Oliveira Pinto、J. Kaplan、H. Edwards、Y. Burda、N. Joseph、G. Brockman、A. Ray、R. Puri、G. Krueger、M. Petrov、H. Khlaaf、G. Sastry、P. Mishkin、B. Chan、S. Gray、N. Ryder、M. Pavlov、A. Power、L. Kaiser、M. Bavarian、C. Winter、P. Tillet、F. P. Such、D. Cummings、M. Plappert、F. Chantzis、E. Barnes、A. Herbert-Voss、W. H. Guss、A. Nichol、A. Paino、N. Tezak、J. Tang、I. Babuschkin、S. Balaji、S. Jain、W. Saunders、C. Hesse、A. N. Carr、J. Leike、J. Achiam、V. Misra、E. Morikawa、A. Radford、M. Knight、M. Brundage、M. Murati、K. Mayer、P. Welinder、B. McGrew、D. Amodei、S. McCandlish、I. Sutskever 和 W. Zaremba，《评估基于代码训练的大语言模型》，2021 年 7 月，arXiv:2107.*03374 [*cs.LG*]*。 [在线]。可访问：http://arxiv.org/abs/2107.03374[2] J. Austin、A. Odena、M. Nye、M. Bosma、H. Michalewski、D. Dohan、E. Jiang、C. Cai、M. Terry、Q. Le 和 C. Sutton，《使用大语言模型进行程序合成》，2021 年 8 月，arXiv:2108.*07732 [*cs*]*。 [在线]。可访问：http://arxiv.org/abs/2108.07732[3] M. Shalan 和 T. Edwards，《构建 OpenLANE：基于 130纳米 OpenROAD 的经流片验证流程》，载于《第 39 届计算机辅助设计国际会议论文集》，美国，线上会议，2020 年 11 月，第 1–6 页，doi：10.1145/3400302.3415735。[4] Efabless，《OpenLane》。 [在线]。可访问：https://efabless.com/openlane[5] P. Lewis、E. Perez、A. Piktus、F. Petroni、V. Karpukhin、N. Goyal、H. Küttler、M. Lewis、W. Yih、T. Rock- täschel、S. Riedel 和 D. Kiela，《面向知识密集型自然语言处理任务的检索增强生成》，2020 年，arXiv:2005.*11401 [*cs.CL*]*。 [在线]。可访问：https://arxiv.org/abs/2005.11401[6] F. Brglez 和 H. Fujiwara，《10 个组合基准电路的中性网表及其 Fortran 目标仿真器》，载于《电路与系统国际研讨会》，1985 年，第 695–698 页。 [在线]。可访问：https://cir.nii.ac.jp/crid/1572261550741925504[7] F. Brglez、D. Bryan 和 K. Kozminski，《时序基准电路的组合特征》，载于《IEEE 电路与系统国际研讨会》，美国俄勒冈州波特兰，1989 年，第 1929–1934 页，doi：10.1109/IS- CAS.1989.100747。[8] K. Xu、R. Qiu、Z. Zhao、G. L. Zhang、U. Schlichtmann 和 B. Li，《LLM 辅助的高效硬件设计自动化》，2024 年 10 月，arXiv:2410.*18582 [*eess*]*。 [在线]。可访问：http://arxiv.org/abs/2410.18582[9] J. Blocklove、S. Garg、R. Karri 和 P. Pearce，《Chip-Chat：对话式硬件设计的挑战与机遇》，载于《ACM/IEEE 第五届面向计算机辅助设计的机器学习研讨会（MLCAD）》，美国，2023 年 9 月，第 1–6 页，doi：10.1109/mlcad58807.2023.10299874。[10] A. Nakkab、S. Q. Zhang、R. Karri 和 S. Garg，《罗马不是一步建成的：基于大语言模型芯片设计的分层提示》，2024 年，arXiv:2407.*18276 [*cs.AI*]*。 [在线]。可访问：https://arxiv.org/abs/2407.18276[11] Y. Wang、W. Ye、Y. He、Y. Chen、G. Qu 和 A. Li，《MCP4EDA：大语言模型驱动的模型上下文协议 RTL 到 GDSII 自动化及后端感知综合优化》，2025 年 7 月，arXiv:2507.*19570 [*cs*]*。 [在线]。可访问：http://arxiv.org/abs/2507.19570[12] G. Yang、Y. Zhou、X. Chen、X. Zhang、T. Y. Zhuo 和 T. Chen，《神经代码生成中的思维链：面向轻量级语言模型及源于轻量级语言模型》，2024 年，arXiv:2312.*05562 [*cs.SE*]*。 [在线]。可访问：https://arxiv.org/abs/2312.05562[13] N. Pinckney、C. Batten、M. Liu、H. Ren 和 B. Khailany，《重新审视 VerilogEval：更新的大语言模型、上下文学习与规范到 RTL 任务》，2024 年，arXiv:2408.*11053 [*cs.SE*]*。 [在线]。可访问：https://arxiv.org/abs/2408.11053[14] W. Ruys、H. Lee、B. You、S. Talati、J. Park、J. Almgren-Bell、Y. Yan、M. Fernando、G. Biros、M. Erez、M. Burtscher、C. J. Rossbach、K. Pingali 和 M. Gligoric，《深入探讨 Python 中基于任务的并行性》，载于《IEEE 国际并行与分布式处理研讨会专题会议（IPDPSW）》，2024 年，第 1147–1149 页，doi：10.1109/IPDPSW63119.2024.00187。