借助分层形式化指导，弥合自然语言规范与寄存器传输级之间的语义鸿沟

Yan Tan、Xiangchen Meng 和 Yangdi Lyu†微电子学方向，香港科技大学（广州）

†通讯作者：yangdilyu@hkust-gz.edu.cn

arXiv:2611.30766v1 [cs.AR] 23 Nov 2026

摘要——尽管大语言模型（LLM）在根据自然语言规范生成寄存器传输级（RTL）代码方面展现出潜力，但面对控制逻辑和时序行为复杂的设计时，其可靠性仍然有限。这源于一个根本性的语义鸿沟：高层自然语言往往没有充分明确实现所需的关键细节，迫使大语言模型推断未明示的假设。本文提出分层形式化指导，这是一种新颖的框架，通过将形式化属性作为自然语言规范与RTL生成之间的中间语义层，弥合这一语义鸿沟。该框架在三个层面应用形式化指导：生成前进行属性驱动的规约增强，生成后进行属性一致性检查，以及检查后进行反例引导的细化。在RTLLM基准测试和多种开源IP设计上的实验表明，该框架实现了更高的功能正确性，并降低了采样输出轨迹的多样性。此外，我们还表明，分层形式化指导能够灵活增强通用大语言模型和领域专用模型，显著提升其生成可靠性。

索引词—**大语言模型生成的Verilog、规范细化、形式化指导、RTL生成**

I. 引言

现有研究表明，大语言模型能够成功为简单模块生成语法有效的Verilog [1], [2], [3], [4]。然而，当逻辑复杂或Verilog代码较长时，大语言模型仍难以生成在功能上正确的设计。一个关键挑战在于高层自然语言（NL）意图与实现关键的寄存器传输级细节之间存在语义鸿沟。自然语言规范通常描述高层功能意图，而寄存器传输级必须严格定义硬件在每个时钟周期、每种状态和每个输入条件下的行为。因此，大语言模型难以将高层意图与实现关键的寄存器传输级细节联系起来，例如周期精确的时序约束、状态更新、接口协议和边界情况行为。

这种语义抽象鸿沟使自然语言到RTL生成不再只是简单的代码转换任务。某个规范对人类设计者而言可能看似清晰，但仍可能未定义重要的RTL行为。例如，高层除法器规范可能只定义被除数、商和余数之间的数学关系。然而，它可能遗漏除以零等边界情况行为、就绪-有效握手等接口协议，或确切的时序约束和输出有效窗口。当生成RTL时，大语言模型必须推断这些未明确说明的语义，从而导致不同的假设，并基于同一描述生成不一致或存在缺陷的RTL实现。

现有的智能体式和反馈驱动方法利用编译器消息、仿真日志或迭代修复来改进生成的寄存器传输级代码 [5], [6], [7], [8]。然而，这些方法主要修正已生成RTL候选方案中的实现错误。当底层自然语言规范不完整、含糊或规定不充分时，修复单个RTL候选方案无法解决潜在的语义不确定性。因此，仅依靠生成后RTL修复不足以弥合语义抽象鸿沟，也无法捕捉关键的行为约束。

为弥合这一差距，我们使用 SystemVerilog 断言（SVA）作为中间层，将有关时序、状态更新和接口行为的候选周期级规则形式化。在 RTL生成之前，框架从规范中推导这些属性，将其转换为结构化的自然语言约束，并将原始规范和这些明确的约束一并提供给生成器。这会揭示未指定的行为，并限制语义自由度。由于规范、生成的属性和寄存器传输级之间仍可能存在不一致，框架会针对当前的 SVA 集对每个 RTL候选方案进行模型检查。当某条 SVA 属性被违反时，框架会分析形式化检查器返回的反例（CEX）轨迹。随后，框架会判断不一致是由 RTL错误、规范中的遗漏或歧义，还是错误的 SVA 属性导致的。然后，框架会修订相应的产物。这种循环内检查仅评估与 SVA 集的一致性，而最终的功能正确性则通过固定的黄金参考独立验证。

我们的主要贡献总结如下：

- 我们提出了分层形式化指导，这是一种通过在规约增强、候选模型检查和迭代式失败归因过程中应用形式化指导，来弥合自然语言规范与寄存器传输级生成之间鸿沟的框架。
- 我们开发了一种属性驱动的规约增强机制，用于揭示隐含的 RTL 级需求，并在代码生成之前限制语义自由度。
- 我们引入了一种反例引导的失败归因与细化机制，将形式化失败映射到

针对性细化，包括寄存器传输级修复、规范消歧和属性纠正。

- 我们通过实验证明，所提出的框架在各种基准测试和源自 IP 的设计上提升了 RTL生成的正确性、语义一致性和可靠性。

II. 相关工作

A. 基于大语言模型的 *RTL* 生成

近期研究通过基准测试、领域专用训练和推理时改进探索了基于大语言模型的寄存器传输级生成。VerilogEval [4] 和 OpenLLM-RTL [3] 等基准测试提供了用于衡量 Verilog 生成质量的评估任务和数据集。RTLCoder [6], 、OriGen [9],、BetterV [10], 、ChipSeek-R1 [11], 和 CodeV-R1 [12],等以训练和数据为中心的方法，通过硬件专用语料库、数据增强、自我反思或基于奖励的优化来改进 RTL生成。

与专注于生成器的技术并行，一些方法在推理过程中引入了反馈驱动的改进。AutoChip [13] 利用编译器和工具反馈进行迭代式 HDL 生成，而 RTLFixer [14] 则执行检索辅助调试以进行 RTL 修复。除了调试信息外，高级编程语言也被用于修复 RTL：通过生成参考模型和测试平台来指导功能纠正 [15]。然而，这些反馈循环严格地在代码生成之后运行，未触及底层的规格说明歧义。

B. 基于大语言模型的硬件生成形式化方法

以往工作主要将形式化方法用于基于大语言模型的 RTL 生成中的断言生成和生成后验证。AssertLLM [16] 和 Spec2Assertion [17] 根据自然语言规范生成断言。Assertion- Forge [18]结合规范与 RTL 中的信息，以改进断言生成。这些研究表明，大语言模型能够将非正式意图转换为可检查属性。

在验证方面，FormalRTL [19] 使用可执行的 C 参考模型和形式等价性检查来验证并改进生成的 RTL。VeriSure [20]结合轨迹分析、断言检查和布尔等价性，以发现并修复 RTL 特定部分中的错误。这些工作主要将形式化方法用于断言生成或生成后检查。相比之下，我们的框架将形式化属性集成到所有生成阶段，在代码生成之前解决规范歧义，并利用形式化

根据验证失败情况，共同改进寄存器传输级设计、规范和属性集。

III. 方法

A. 研究动机

自然语言提示通常描述高层功能意图，却将必要的实现要求隐含起来。在生成寄存器传输级代码时，大语言模型不得不自行推断这些未明确说明的实现细节，

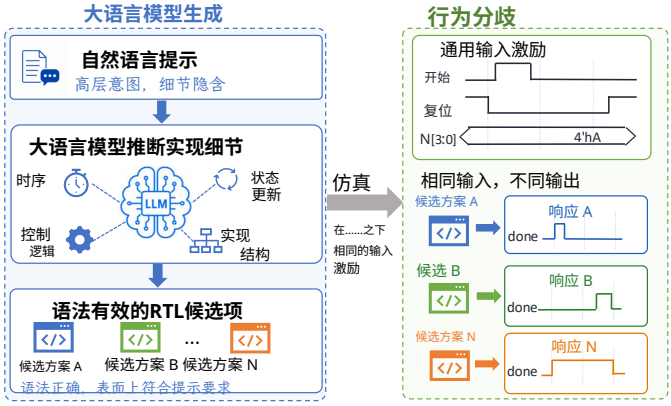


图1：规范不足的提示可能导致语法正确但行为分歧的寄存器传输级候选方案。

例如周期精确时序、状态更新机制、控制逻辑行为和结构划分。因此，对描述不充分的提示进行重复采样，会生成多个语法有效的RTL候选方案。它们看似符合提示要求，却在仿真中表现出严重的功能差异。

图1展示了这种分歧的一个示例。在相同输入激励的驱动下，不同的生成候选方案会产生相互冲突的输出轨迹响应。由于这种行为分歧源于根本性的规范歧义，而非简单的语法错误，因此仅依靠生成后RTL修复无法解决其潜在的不确定性。因此，生成框架必须将未明确说明的周期级规则形式化，以便在代码生成开始之前缩小语义空间。

B. 概述

图2展示了分层形式化指导的整体工作流程。该框架由三层组成。第1层从原始规范中推导候选SVA属性，并将其转换为结构化约束，以增强提示。第2层根据增强规范生成候选寄存器传输级方案，并依据SVA属性评估每个候选方案。当某个属性失败时，形式化工具会生成一条反例轨迹。第3层利用该轨迹定位失败原因，并在寄存器传输级方案、规范或属性集之间进行针对性细化。根据诊断结果，该框架要么直接重新验证经过修复的寄存器传输级候选方案，要么依据更新后的规范和SVA属性重新生成寄存器传输级方案。*SVA* 检查在这一生成与细化循环中进行，而最终的功能正确性则使用独立的黄金参考单独评估。

C. 属性提取与规范增强

在第1层中，该框架从自然语言规范中提取一组初始的 SVA属性。每个属性都将必须在寄存器传输级定义的关键行为形式化，例如复位逻辑、时序约束、状态

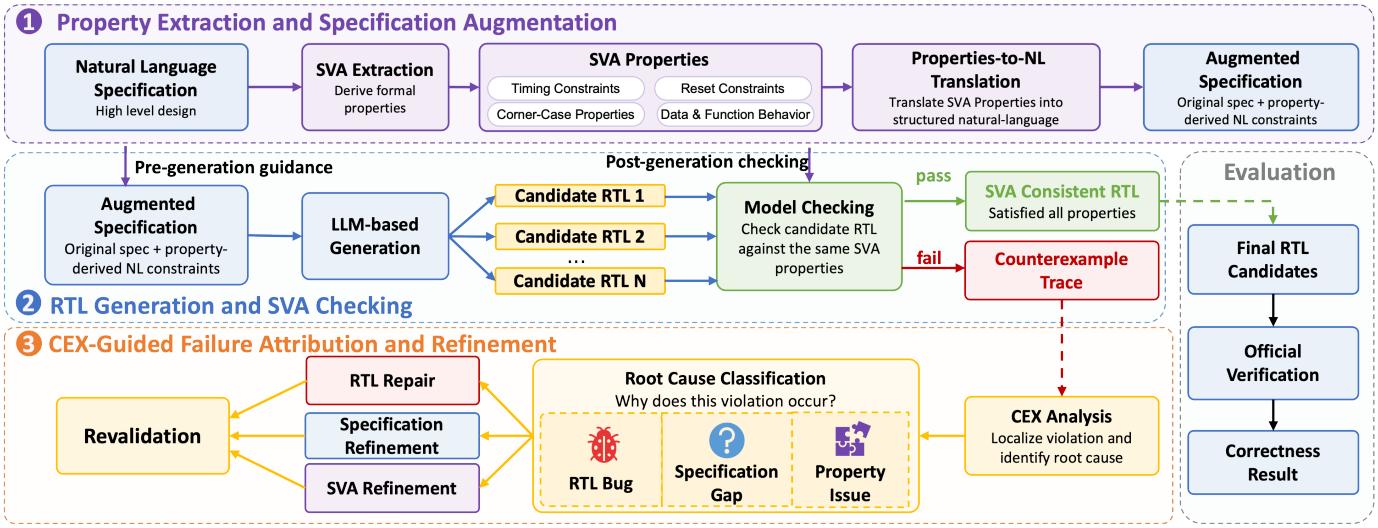


图2：分层形式化指导框架概览。

转换、接口握手和边界情况，从而为大语言模型提供明确的实现要求。

完成基本检查后，我们将保留的 SVA 属性转换为简洁的自然语言约束，用于寄存器传输级生成。该转换保留每个属性的条件、时序关系和所需响应。随后，将生成的约束追加到原始提示中，形成用于寄存器传输级生成的增强规范。原始规范提供设计功能和接口，而追加的约束则描述更具体的 RTL 级需求。图3以一个无符号除法器为例说明了这一过程。在该示例中，关于商余数一致性和除零处理的属性被转换为自然语言约束，并追加到原始规范中。

D. 寄存器传输级生成与 *SVA* 检查

第1层生成的增强规范为寄存器传输级生成提供了更具体的 RTL 级需求。然而，大语言模型生成器在寄存器传输级生成过程中仍可能出现实现错误。因此，第2层检查生成的寄存器传输级代码是否满足这些需求。该框架将生成的 Verilog 与 SystemVerilog断言集合一同提交给自动化模型检查器。如果所有属性均通过检查，则当前生成完成。如果任一属性失败，形式化工具将返回失败的属性及反例轨迹。我们将失败的属性和反例传递给第3层，以进行失败分析和反例引导的细化。

E. CEX 引导的失败归因与细化

对于每个未通过 SVA 检查的 RTL 候选方案，分析器都会接收规范、当前 RTL 候选方案、违反的 SVA 属性以及反例轨迹。反例提供与该违反相关的具体输入、状态和 RTL 行为。分析器会评估这些诊断信号，以隔离故障根因，并针对以下三类潜在对象执行定向更新：

•RTL 实现错误：当候选 RTL 未能实现规范中已明确规定的要求时，该框架会直接修复或重新生成候选 RTL。

• 规格说明歧义：当反例揭示出原始规范未充分定义的寄存器传输级行为时，框架会澄清提示，然后根据更新后的规范重新生成寄存器传输级设计。

• 属性错误：当某个 SVA 与原始规范冲突、遗漏必需条件或错误地限制了合法行为时，框架会同时修订该属性及其翻译后的提示约束，然后重新生成寄存器传输级设计。

如果单条轨迹揭示了多个工件中的缺陷，框架会同时更新所有受影响的组件。每个寄存器传输级候选方案都会被独立细化。每次细化后，框架都会针对当前属性集重新运行 SVA 检查。当候选方案满足当前属性、不再取得进一步进展，或达到预设的迭代上限时，细化循环将停止。

图4展示了使用实现迭代长除法算法的无符号除法器模块进行反例引导的细化过程。在此任务中，在第1层规约增强期间，框架提取出一个关于操作数对齐的候选形式化属性，并将其转换为明确的提示要求，以指导模型匹配操作数位宽。尽管这一表述对人类设计者而言很清晰，但其中包含语义鸿沟。在第2层RTL生成期间，大语言模型逐字翻译操作数位宽要求，通过对 temp remainder[15:8]进行切片，_ 使其匹配除数位宽。在减去除数之前对部分余数进行切片会引入功能错误，并产生错误结果。在第2层模型检查期间，SVA 检查捕获了这一功能违规，而反例提供了触发故障的逐周期精确输入和状态转换。在

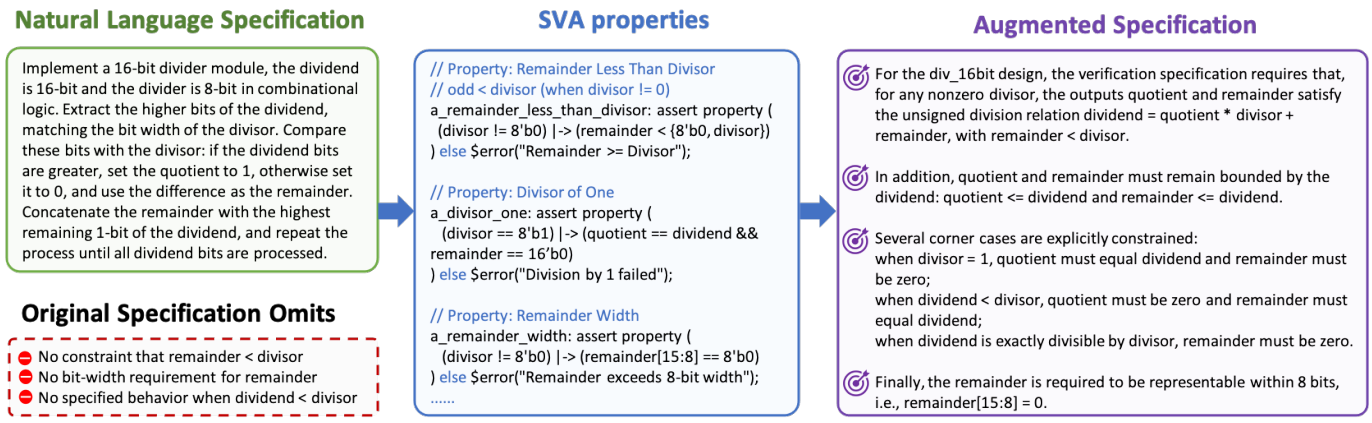


图3：基于形式属性的规范增强用于无符号除法器。

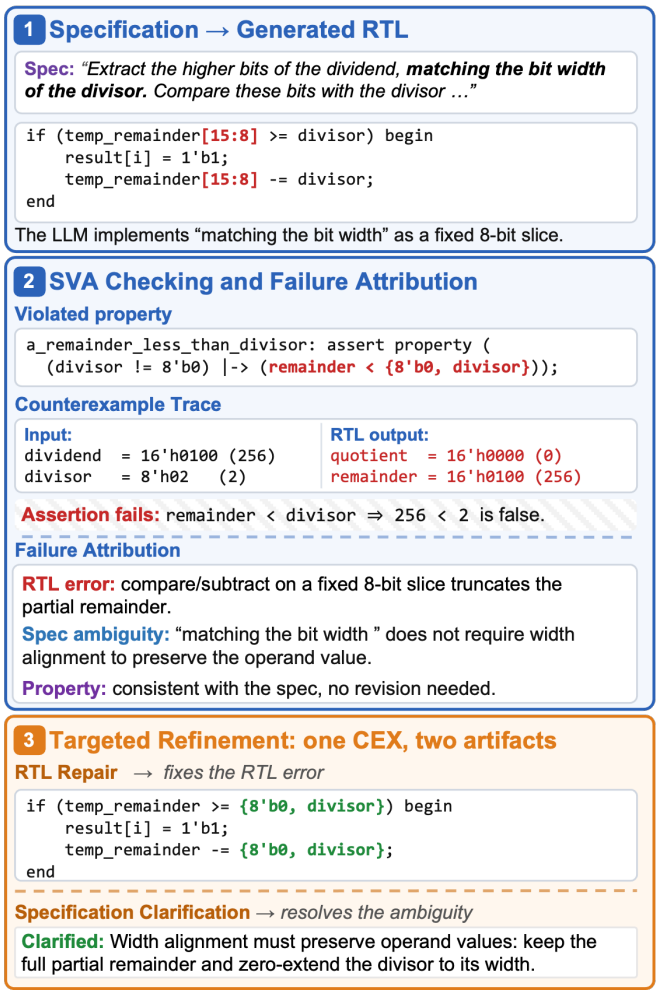


图4：无符号除法器的反例引导的细化示例。

第3层基于这一诊断反馈，通过明确相应的位宽处理要求并保留完整的临时余数，对寄存器传输级设计进行重新配对，同时零扩展较窄的除数。该示例表明，尽管生成前的规约增强可以减少未充分规范的行为，但无法保证这些要求在寄存器传输级中得到正确实现。因此，生成后的 SVA 检查和反例引导的细化仍然是必需的。

IV. 实验

A. 实验设置

我们在四个涵盖不同寄存器传输级任务的公开基准数据源上评估分层形式化指导：来自 RTLLM [21] 和 VerilogEval [4], 的从规范到寄存器传输级设计的生成任务、来自 CVDP [22], 的 RTL 代码生成任务，以及来自 ChipVerilog [23] 的 IP 级 RTL 生成任务。我们评估了具有代表性的先进 RTL 生成方法，包括 OriGen [9], 、RTLCoder、 [6] CodeV-R 1 [12],, 以及来自 DeepSeek [24],、通义千问 [25], 和 GPT 系列 [26] 的通用大语言模型。

在生成过程中，我们使用温度 0.7 和核采样 0.9 生成十个寄存器传输级候选方案。对于每个任务，使用相同的大语言模型生成一次 SVA 集合，并将同一 SVA 集合用于全部十个寄存器传输级候选方案。每个候选方案的 CEX 引导细化限制为一次迭代。只有通过相应基准测试官方验证流程的候选方案才被视为正确：VerilogEval、RTLLM 和 CVDP 使用官方测试平台；ChipVerilog 使用官方测试平台，并针对参考实现进行等价性检查。功能正确性使用标准的*pass@k* 指标进行报告：

$$\text{pass@k} := \mathbb{E}_{\text{Problems}}\left[1 - \frac{\binom{n-c}{k}}{\binom{n}{k}}\right] \quad (1)$$

其中， $n = 10$表示生成样本的总数， c 表示通过官方验证流程的候选样本数， k 表示选定代码样本的数量。仿真使用 Synopsys VCS [27],执行，而形式化检查使用 Yosys [28]执行。本地模型推理在单个 NVIDIA A30 GPU 上进行，

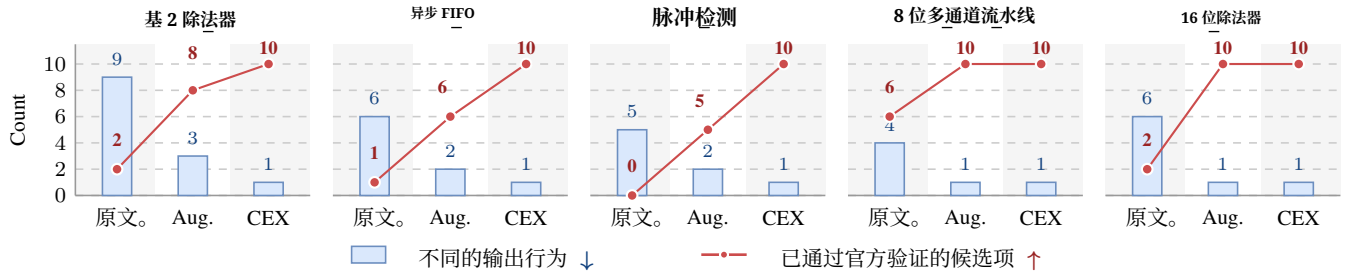


图 5：代表性 RTLLM 困难任务上的行为收敛与功能正确性。柱状图表示不同输出行为的数量，折线表示通过官方验证的候选实现数量。

而通过 API 托管的模型（如 GPT）则通过其官方 API 访问。

B. 行为差异分析

为分析候选实现的多样性，我们针对五个代表性 RTLLM 任务分别生成了十个寄存器传输级候选实现。我们使用行为数量衡量输出多样性，并使用通过数量衡量验证成功情况。如图 5 所示，在采用原始自然语言规范（Orig.）时，每个任务产生了 4 至 9 种不同的行为，而只有 0 至 6 个候选实现通过了官方验证。该结果表明，自然语言规范可能导致不同的 RTL行为和不正确的实现。

随后，我们通过增强规范（Aug.）加入了明确的寄存器传输级要求。应用增强规范后，行为数量降至 1 至 3 种，而官方验证通过数量增加至 5 至 10 个。我们进一步应用反例引导的细化（CEX），以解决规范、形式化规则与生成的寄存器传输级实现之间的不匹配。经过一轮细化后，五个任务均仅表现出一种观测行为，且全部十个候选实现都通过了官方验证。这些结果表明，明确的寄存器传输级约束能够在生成前缩小生成空间，而反例引导的细化则能在生成后减少剩余的不一致。

表 I：使用 DeepSeek-V4 在 FPU 和 OR1200 上生成的 SVA 属性及寄存器传输级生成结果的质量。时间表示每项任务的平均耗时。

指标	**FPU**		**OR1200**	
平均每项任务的 SVA 数量	18.3		17.3	
语法有效率	85.8%		91.0%	
非空洞率	91.9%		88.8%	
设置	**Pass@1**	**时间（秒）**	**Pass@1**	**时间（秒）**
原始规范	5.6%	94.8	12.2%	88.5
原始规范 +CEX	5.6%	142.6	18.1%	137.5
增强规范	11.1%	102.5	29.3%	91.6
增强规范 + CEX	22.2%	165.1	40.2%	145.9

*C. SVA*属性的有效性

表I报告了使用 DeepSeek-V4-Flash 生成的 SVA 属性的质量及其对寄存器传输级生成的影响。我们使用两个指标评估 SVA 质量：语法有效性，即被形式化工具接受的生成属性所占的比例；以及非空洞性，即在任务设置下，至少被一次形式化运行或仿真轨迹触发的可执行属性所占的比例。对于下游寄存器传输级生成，我们报告 Pass@1 和每项任务的平均时间，其中包括大语言模型推理和形式化检查。

生成的属性具有较高的有效率和非空洞率。在 FPU 上，语法有效率为 85.8%，在 OR1200 上为 91.0%；非空洞率分别为 91.9% 和 88.8%。将这些属性用作指导时，还能提高 RTL 的正确性。规约增强使 FPU 上的 Pass@1 从 5.6% 提升至 11.1%，使 OR1200 上的 Pass@1 从 12.2% 提升至 29.3%。在增强后的规格上应用反例引导的细化时，效果显著更好。在 OR1200 上，使用原始规范时，它将 Pass@1 从 12.2% 提升至 18.1%；使用增强规范时，则从 29.3% 提升至 40.2%。这些结果表明，模型生成的 SVA属性能够为寄存器传输级生成提供有效的语义约束，尤其适用于控制和状态密集型设计。

D. 正确性比较

表 II 总结了五个大语言模型在 FPU、OR1200 和 CVDP 基准测试上，在三种设置下的 pass@k 结果：基础模型、采用基于 SVA 的规格增强（+SVA）的基础模型，以及采用 CEX 引导细化的完整框架（+SVA+CEX）。分层形式化指导提升了五个模型在三个基准测试上的表现。规约增强使 FPU、OR1200 和 CVDP 上的平均 pass@1 分别提高了 6.67 个百分点、11.46 个百分点和 7.75 个百分点。CEX 细化又分别带来了 8.22 个百分点、9.54 个百分点和 3.05 个百分点的提升。这些结果表明，两个阶段提供了互补的增益。规约增强在 RTL生成之前限制了语义歧义，而 CEX 细化则处理生成之后残留的不一致问题。

该改进在各基准测试套件中均表现一致。在 FPU 上，平均 pass@1 从 4.89% 提升至 19.78%；在 OR1200 上，从 16.99% 提升至 37.99%；在 CVDP 上，则从 39.3% 提升至 50.09%。在两个 IP 级基准测试上，增益更为显著，FPU 达到 14.89 个百分点，OR1200 达到 21.00 个百分点。这些结果表明，我们的方法在规模更大、复杂度更高的寄存器传输级设计上依然有效。尽管 CVDP 的基础模型准确率更高，其增益仍达到 10.79 个百分点。其

表 II：通过 @k 在 FPU（9 项任务）、OR1200（38 项任务）和 CVDP（76 项任务）基准套件上的结果s.

模型	设置	FPU			OR1200			CVDP		
		pass@1	pass@5	pass@10	pass@1	pass@5	pass@10	pass@1	pass@5	pass@10
通义千问 3.6	基础模型	2.22	11.11	22.22	29.48	40.97	42.11	31.15	43.23	50.00
	基础模型 +SVA	13.33	33.95	44.44	34.53	45.27	50.53	35.38	60.39	69.23
	基础模型 +SVA + CEX	17.78	43.25	55.56	42.52	52.41	54.74	36.41	61.85	70.51
DeepSeek-Coder	基础模型	0.00	0.00	0.00	0.45	2.25	4.51	13.64	21.78	24.24
	基础模型 +SVA	3.33	10.19	11.11	6.32	20.16	27.07	27.56	36.77	39.74
	基础模型 +SVA + CEX	4.45	10.85	11.11	11.73	33.73	40.60	32.56	40.61	42.31
DeepSeek-V4	基础模型	5.55	11.07	11.11	12.18	34.98	40.60	49.36	55.97	56.41
	基础模型 +SVA	11.11	25.31	33.33	29.32	54.99	58.65	50.64	57.25	57.69
	基础模型 +SVA + CEX	22.22	46.34	55.56	40.15	57.52	58.65	55.64	63.65	64.10
GPT-4o	基础模型	4.45	10.85	11.11	11.28	31.47	36.09	44.49	57.97	61.54
	基础模型 +SVA	7.78	16.67	22.22	26.17	49.71	54.13	48.85	63.10	67.95
	基础模型 +SVA + CEX	11.11	26.85	33.33	32.93	54.60	58.65	50.00	63.74	67.95
GPT-5.4	基础模型	12.22	28.40	33.33	31.58	50.46	54.13	56.94	63.70	65.28
	基础模型 +SVA	22.22	54.32	66.67	45.94	69.39	73.68	72.78	77.78	79.17
	基础模型 +SVA + CEX	43.33	78.13	88.89	62.63	72.08	73.68	75.83	80.56	81.94
SVA的平均增益		+6.67	+15.80	+20.00	+11.46	+15.88	+17.32	+7.75	+8.52	+9.21
SVA的平均增益 + CEX		+14.89	+28.80	+33.34	+21.00	+22.04	+21.78	+10.79	+11.54	+11.82

表 III：分层形式化指导在多种大语言模型和专用 RTL 生成系统中的兼容性与性能提升。

模型	设置	VerilogEvalMachine		VerilogEvalHuman		RTLLM 2.0	
		pass@1	pass@5	pass@1	pass@5	pass@1	pass@5
RTLCoder	基础 7B	61.20	76.50	41.60	50.10	43.50	48.10
	+SVA+CEX	65.26	79.80	49.04	57.17	62.90	66.27
OriGen	基础 7B	74.10	82.40	54.40	60.10	50.90	60.90
	+SVA+CEX	78.23	85.55	61.90	67.20	69.10	74.40
CodeV-R1	基础 8B	76.50	84.10	69.90	79.30	72.90	86.10
	+SVA+CEX	80.14	87.25	77.14	86.35	89.70	97.21
Qwen3.6	基础版 35B	74.13	80.05	55.32	67.27	52.34	60.70
	+SVA+CEX	81.89	90.87	69.62	85.29	75.74	87.18
DeepSeek-V4 Base 284B		76.43	83.07	65.19	75.62	52.98	70.54
	+SVA+CEX	79.86	91.47	71.28	87.86	70.21	90.68
平均配对增益（百分点）		**+4.60**	**+5.76**	**+8.51**	**+10.30**	**+19.01**	**+17.88**

细化增益的幅度还取决于基础模型。在 FPU 上，GPT-5.4 从 CEX 阶段获得了 21.11 个百分点的提升，而 DeepSeek-Coder 仅提升了 1.12 个百分点。尽管形式化工具提供了详细的反例轨迹，性能增益仍取决于模型分析这些执行轨迹、推理复杂逻辑并生成正确修复方案的能力。

E. 跨大语言模型架构的泛化性

我们在 VerilogEval 和 RTLLM 2.0 上，使用三种 RTL 专用方法（OriGen、RTLCoder 和 CodeV-R1）以及两种通用大语言模型（通义千问3.6 和 DeepSeek-V4）评估了我们的方法。对于每种方法，我们保留其原有的生成流程，并使用相同的基础模型进行 SVA 生成、RTL生成和反例分析。表 III 展示了五种方法均取得了一致的改进。在 VerilogEval-Machine、VerilogEval-Human 和 RTLLM 2.0 上，平均 pass@1 增益分别为 4.60 个百分点、8.51 个百分点和 19.01 个百分点。该改进同样适用于最先进的领域专用模型。CodeV-R1 在 RTLLM 2.0 上将结果从 72.90% 提升至 89.70%

而 OriGen 在 VerilogEval-Human 上提升了 7.50 个百分点，在 RTLLM 2.0 上提升了 18.20 个百分点。这些结果表明，分层形式化指导与模型无关，并且能够无缝集成现有的反馈驱动方法和生成后修复方法。

五、结论

本文提出了分层形式化指导，这是一种旨在弥合高层自然语言规范与实现关键的寄存器传输级细节之间语义鸿沟的框架。其核心思想是以分层特定的形式应用形式化指导：在生成前施加结构化约束，在生成后进行基于断言的正确性检查，并在验证后利用基于反例的细化信号。对形式化属性的分层使用缩小了大语言模型的生成空间，根据形式化断言验证生成的寄存器传输级代码，并确定应当细化的是寄存器传输级代码、规范，还是属性。在 RTLLM 困难任务和源自 OpenCores 的 IP设计中，分层形式化指导提升了多个大语言模型的功能正确性和行为收敛性。这些结果表明，形式化属性不仅可以充当生成后的检查器，还可以作为主动指导机制，以实现更可靠的寄存器传输级代码生成。

参考文献

[1] S. Thakur等，“面向自动化Verilog寄存器传输级代码生成的大语言模型基准测试”，载于*2023*年欧洲设计、自动化与测试会议及展览会（*DATE*）。IEEE，2023年，第1–6页。[2] E. Dehaerne、B. Dey、S. Halder和S. De Gendt，“面向设计与验证自动化的Verilog自动补全深度学习框架”，*arXiv*预印本 *arXiv:2304.13840*，2023年。[3] S. Liu、Y. Lu、W. Fang、M. Li和Z. Xie，“OpenLLM-RTL：面向大语言模型辅助设计RTL生成的开放数据集与基准（特邀）”，载于*2024*年*IEEE/ACM*国际计算机辅助设计会议（*ICCAD*）。ACM，2024年。

[4] M. Liu、N. Pinckney、B. Khailany和H. Ren，“VerilogEval：评估用于Verilog代码生成的大语言模型”，载于2023年IEEE/ACM国际计算机辅助设计会议（ICCAD）。IEEE，2023年，第1–8页。[5] C.-T. Ho、H. Ren和B. Khailany，“Verilogcoder：采用基于图的规划和抽象语法树（AST）波形追踪工具的自主Verilog编码智能体”，arXiv预印本 arXiv:2408.08927，2024年。[6] S. Liu、W. Fang、Y. Lu、Q. Zhang、H. Zhang和Z. Xie，“RTLCoder：凭借我们的开源数据集和轻量级方案，在设计RTL生成方面超越GPT-3.5”，载于2024年IEEE大语言模型辅助设计研讨会（LAD）。IEEE，2024年，第1–5页。[7] Y. Zhao、H. Zhang、H. Huang、Z. Yu和J. Zhao，“Mage：用于自动化RTL代码生成的多智能体引擎”，载于2025年第62届ACM/IEEE设计自动化会议（DAC），2025年，第1–7页。[8] Y. Wei、Z. Huang、L. He、L. Huang、T.-J. Lin和W. W. Xing，“Vflow：发现用于Verilog生成的最优智能体工作流”，载于2026年第31届亚洲及南太平洋设计自动化会议（ASP-DAC），2026年，第355–361页。[9] F. Cui、C. Yin、K. Zhou、Y. Xiao、G. Sun、Q. Xu等，“OriGen：通过代码到代码增强和自反思提升RTL代码生成”，arXiv预印本 arXiv:2407.16237，2024年。[10] Z. Pei、H.-L. Zhen、M. Yuan、Y. Huang和B. Yu，“BetterV：通过判别式引导实现受控Verilog生成”，arXiv预印本 arXiv:2402.03375，2024年。[11] Z. Chen、K. Chang、Z. Li、X. He、C. Chen、C. Li、M. Wang、H. Xu、Y. Han和Y. Wang，“ChipSeek-R1：通过分层奖励驱动强化学习，利用大语言模型生成超越人类的RTL”，arXiv预印本 arXiv:2507.04736，2025年。[12] Y. Zhu、D. Huang等，“Qimeng-codev-r1：推理增强的Verilog生成”，《神经信息处理系统进展》，第38卷，第154 266–154 300页，2026年。[13] S. Thakur、J. Blocklove、H. Pearce、B. Tan、S. Garg和R. Karri，“Autochip：利用大语言模型反馈实现HDL生成自动化”，arXiv预印本 arXiv:2311.04887，2023年。[14] Y. Tsai、M. Liu和H. Ren，“RTLFixer：使用大语言模型自动修复RTL语法错误”，载于《第61届ACM/IEEE设计自动化会议论文集》，2024年，第1–6页。[15] Y. Tan、X. Meng、Z. Jiang和Y. Lyu，“Autoverifix：自动纠错并提升大语言模型生成的Verilog代码的功能正确性”，载于2026年第31届亚洲及南太平洋设计自动化会议（ASP-DAC），2026年，第526–532页。[16] Z. Yan、W. Fang、M. Li、M. Li、S. Liu、Z. Xie和H. Zhang，“AssertLLM：通过多大语言模型从设计规范生成硬件验证断言”，载于《第30届亚洲及南太平洋设计自动化会议论文集》，2025年，第614–621页。[17] F. Wu、E. Pan、R. Kande、M. Quinn、A. Tyagi、D. Kebo、J. Rajendran和J. Hu，“Spec2Assertion：使用具备渐进式正则化的大语言模型自动进行RTL前断言生成”，arXiv预印本 arXiv:2505.07995，2025年。[18] Y. Bai、G. B. Hamad、S. Suhaib和H. Ren，“AssertionForge：通过规范与RTL的结构化表示增强形式化验证断言生成”，载于2025年IEEE国际大语言模型辅助设计会议（ICLAD）。IEEE，2025年，第85–92页。[19] K. Li、M. Li、X. Wen、S. Zhao、J. Wu、J. Huang和Q. Xu，“FormalRTL：大规模经验证的RTL综合”，arXiv预印本 arXiv:2603.08738，2026年。[20] J. Liu、T. Zhou和T. Jiang，“Veri-sure：用于正确RTL代码生成的契约感知多智能体框架，结合时序追踪与形式化验证”，arXiv预印本 arXiv:2601.19747，2026年。[21] Y. Lu、S. Liu、Q. Zhang和Z. Xie，“RTLLM：面向设计RTL生成的大语言模型开源基准”，载于2024年第29届亚洲及南太平洋设计自动化会议（ASP-DAC）。IEEE，2024年，第722–727页。[22] N. Pinckney、C. Deng、C.-T. Ho、Y.-D. Tsai、M. Liu、W. Zhou、B. Khailany和H. Ren，“综合性Verilog设计问题：用于评估大语言模型和智能体在RTL设计与验证方面能力的新一代基准数据集”，2025年。 [在线]。可访问：https://arxiv.org/abs/2506 14074 。[23] Y. Tan、J. Du、X. Meng和Y. Lyu，“ChipVerilog：面向基于大语言模型的Verilog RTL生成的大规模OpenCores衍生基准”，2026 [ ] 2607 13079 。在线。可访问：[24] 4 https://arxiv.org/abs/ 。DeepSeek-AI，“DeepSeek-V：迈向高效的百万词元上下文智能”，2026年。[25] Qwen团队，“Qwen3.8-Max：编程与协作的新标杆”，2026年8月。 [在线]。可访问：https://qwen.ai/blog?id=qwen3.8 [26] OpenAI，《GPT-5.3 模型》，https://developers.openai.com/api/docs/models/gpt- 5.3，2026年，访问时间：2026年5月。[27] 新思科技公司，*VCS*：功能验证解决方案，新思科技公司，产品文档。[28] C. Wolf，《Yosys 开源综合套件》，https://yosyshq.net/yosys/，2013年。