计算机研究与发展
Journal of Computer Research and Development
https://crad.ict.ac.cn
63(6): 1409−1422, 2026

Faver: 基于函数抽象验证的 RTL 生成 ReAct 推理框架

穆嘉楠[1†] 史茗宇[1,2†] 叶 靖[1,2] 晁志腾[1] 李华伟[1,2]

[1]（处理器芯片全国重点实验室（中国科学院计算技术研究所） 北京 100190）

[2]（中国科学院大学 北京 100190）

（chaozhiteng@ict.ac.cn）

Faver: Function-Abstracted Verification for RTL Generation Framework with ReAct Reasoning

Mu Jianan[1], Shi Mingyu[1,2], Ye Jing[1,2], Chao Zhiteng[1], and Li Huawei[1,2]

[1]（*State Key Lab of Processors* (*Institute of Computing Technology*, *Chinese Academy of Sciences*), *Beijing* 100190）

[2]（*University of Chinese Academy of Sciences*, *Beijing* 100190）

Abstract Register transfer level (RTL) code generation based on large language models (LLMs) has emerged as a promising research direction, as it targets the least automated stage in current chip design workflows. However, the substantial semantic gap between high-level specifications and RTL, combined with limited training data, poses significant challenges for generation accuracy. A natural approach is to leverage human design experience by integrating design and verification, yet RTL test data are even scarcer than design data, making this strategy less LLM-friendly. In contrast, LLMs exhibit stronger capabilities in high-level languages such as Python or C, which are more suitable for functional specification and appear promising for RTL verification. Nevertheless, a large semantic gap remains, with notable differences in spatiotemporal granularity between high-level languages and hardware code. Verifying RTL using Python or similar high-level languages requires the LLM not only to understand high-level functional semantics but also to ensure that low-level timing and operational details match the circuit behavior, which is nontrivial. To address this challenge, we propose function-abstracted verification for RTL generation with ReAct reasoning—Faver, a middleware framework that streamlines RTL verification in LLM-based workflows. By combining LLM-friendly code structures with rule-based templates, Faver decouples circuit verification details, enabling the LLM to focus on functionality itself. Experimental results on both SFT and open-source models show that Faver can improve RTL generation accuracy by up to 14%.

Key words ReAct paradigm; LLM; RTL generation; verifiable middleware; prompt engineering

摘 要 基于大语言模型（LLM）的寄存器传输级（RTL）代码生成是近年来受到广泛关注的研究方向，因为它旨在突破当前芯片设计流程中自动化程度最低的环节。然而，高层规格描述与 RTL 之间存在显著语义鸿沟。同时训练数据有限，导致现有模型在生成正确性方面仍面临较大挑战。一种自然的思路是借鉴人工设计经验，将设计与验证相结合，但 RTL 测试数据相较设计数据更加稀缺，使该方法对 LLM 并不友好。相比之下，LLM 在 Python/C 等高层语言中表现出更强能力，这类语言更适合功能描述，似乎更有潜力用于 RTL 验证。但高层语言与 RTL 之间仍存在巨大的语义差异，并且在时空粒度上存在明显差别。

收稿日期：2026-03-11；**修回日期**：2026-04-23

基金项目：中国科学院战略性先导科技项目（XDB0660103）；国家自然科学基金项目（92473203, U25A20486）

This work was supported by the Strategic Priority Research Program of Chinese Academy of Sciences (XDB0660103) and the National Natural Science Foundation of China (92473203, U25A20486).

基于高层语言对RTL进行验证,需要LLM不仅理解高层功能语义,还必须确保底层时序行为与电路执行一致,这并非易事。针对上述问题,提出基于函数抽象验证的RTL生成ReAct推理框架—Faver,作为一种函数抽象可验证中间件,用于简化LLM驱动的RTL生成与验证流程。通过将更适合LLM处理的代码结构与基于规则的模板相结合,Faver对电路验证细节进行解耦,使LLM能够专注于功能本身。实验结果表明,在SFT模型和开源模型上,Faver可将RTL生成正确率提升最高达14%。

关键词 ReAct范式;大语言模型;RTL生成;可验证中间件;提示工程

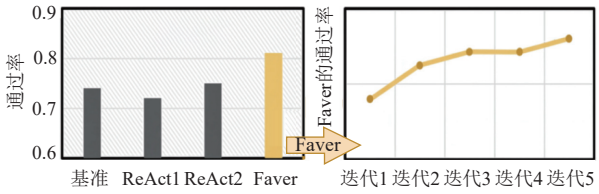
中图法分类号 TP311.13; TP309

DOI: 10.7544/issn1000-1239.202660164 **CSTR:** 32373.14.issn1000-1239.202660164

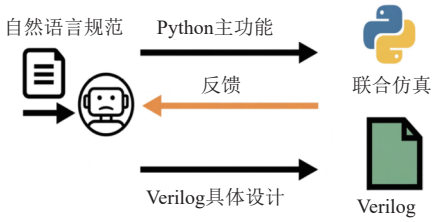
寄存器传输级(register transfer level, RTL)设计是将功能规格转化为RTL代码的过程,是集成电路(IC)设计流程中自动化程度最低、人工投入最多的关键环节之一。近年来,随着大语言模型(large language model, LLM)技术的快速发展,利用LLM实现RTL设计自动化逐渐成为研究热点。已有研究通过系统构建训练数据集,设计针对性的训练策略,不断提升专用模型和商业模型的RTL生成能力[1-9]。

从框架层面来看,现有基于LLM的RTL生成方法大体可分为2类。早期方法通常直接根据自然语言规格生成RTL代码,其生成结果虽然具备一定可用性,但功能正确性往往缺乏保障[1,4]。为进一步提高生成准确率,后续研究引入了类似ReAct(reasoning and acting)的迭代推理机制,即在人工编写测试平台(testbench, TB)的支持下,结合EDA工具反馈对生成结果进行迭代修正[3,10]。围绕测试平台生成,自验证和多智能体验证闭环,相关工作也从不同角度探索了自动反馈机制[11-13]。这类方法在一定程度上提升了RTL生成质量,但其整体流程仍然依赖人工提供验证环境。实际上,编写一个功能完备且时序正确的测试平台本身就是一项复杂且耗时的工作,因此现有方法仍未真正实现RTL设计过程的自动闭环。

基于此,本文关注如下问题:如何构建一种无需人工编写测试平台、无需人工干预的自验证RTL生成框架,使LLM能够在自动迭代过程中持续修正设计错误,并实现更高的功能正确率。围绕这一问题,一个直观的思路是直接让LLM承担验证职责。具体而言,一种做法是将LLM本身作为判别器,直接判断生成的RTL是否满足功能规格;另一种做法是让LLM进一步生成RTL测试平台,包括参考模型和测试激励,再据此完成验证与反馈。图1(a)对这2类朴素ReAct构造方式及其效果进行了直观展示。从图1(a)可以看出,ReAct1和ReAct2相较于基线方法均未表现出显著提升。其原因在于,RTL验证涉及大量复杂的时序行为和结构细节,使得“LLM作为判别器”的方式难以给出稳定可靠的判断;而构建Verilog级测试平台,尤其是周期精确的黄金参考模型,其复杂度和专业门槛往往接近于直接完成RTL设计本身。



(a) 不同ReAct思路的构造效果



(b) Faver:带有函数抽象可验证中间件的ReAct框架

注:基准方案表示LLM直接生成RTL;ReAct1表示LLM直接进行功能判断;ReAct2表示LLM使用Verilog测试平台辅助判断。相比之下,Faver具有更高的通过率,且可在多轮迭代中实现提升。

Fig. 1 Performance comparison of ReAct approaches and the Faver framework in RTL generation

图1 ReAct方法及Faver框架在RTL生成中的性能比较

相比之下,使用Python或C等高层语言构建功能级测试程序是一种更具吸引力的方案。一方面,高层语言验证更符合人工验证中的常见习惯,能够从功能层面对设计行为进行直接刻画;另一方面,相比Verilog等硬件描述语言,LLM在Python或C代码生成方面具备更强能力,相关训练数据也更加丰富。此外,借助Python-RTL协同仿真框架,还可以对高层参考模型与RTL实现进行精确比对,而不必依赖LLM自身完成最终判定[14-15]。更重要的是,将设计代码和验证代码分别生成,有助于降低两者在分布上的耦合程度,从而减少“自我验证”带来的误判风险,

提升验证鲁棒性。

然而，现有高层语言辅助验证方法通常仍将 Python 或 C 语言作为测试平台或辅助程序使用，尚未显式建模 RTL 的周期级执行语义，因此在时序一致性验证方面仍存在不足。我们的进一步实验也发现，若不对验证过程进行专门建模，仅依赖 LLM 直接生成 Python 测试程序，往往会产生较多误判，对 RTL 生成正确率的提升十分有限。问题的根源在于，硬件和软件虽然可以描述相同功能，但二者在表达粒度上存在本质差异：硬件系统强调时钟驱动、状态演化和周期级行为，而软件程序更侧重输入到输出的函数映射关系。这种“时空粒度不一致”使得硬件—软件协同验证面临 2 方面关键挑战。

1）硬件中的许多变量天然具有时间相关性。在 RTL 设计中，输入通常表现为由时钟驱动的比特流，模块输出不仅由当前周期输入决定，还受寄存器状态及历史输入影响。然而，标准 Python 或 C 语言并不具备时钟、寄存器和复位等原生语义。虽然已有工作尝试借助特定领域语言支持高层验证[16]，但这类语言在公开数据中相对稀缺，不利于 LLM 学习和泛化。

2）测试激励的构造本身具有较高难度。高质量的电路测试激励不仅要覆盖设计的主要功能空间，还要兼顾时序语义、输入依赖关系以及比特级边界约束。对于 LLM 而言，仅凭自然语言规格直接生成兼具功能覆盖性和数据精确性的测试激励，并不是一项容易完成的任务。

针对上述问题，本文提出基于函数抽象验证的 RTL 生成 ReAct 推理框架—Faver。图 1(b)给出了 Faver 的整体框架，LLM 在根据自然语言规格生成 Verilog 设计的同时，生成 Python 主功能验证代码；随后，借助联合仿真对高层功能代码与 RTL 实现进行比对，并将验证结果反馈给 LLM，从而形成自动闭环的 ReAct 迭代过程。通过在 RTL 设计与高层验证代码之间引入函数抽象层，Faver 使 LLM 能够充分发挥其在高层语义建模和程序生成方面的优势，以更适合自身能力的方式完成 RTL 验证。

与图 1(a)中的 2 种朴素 ReAct 方案相比，Faver 无需人工参与即可实现更稳定、更显著的正确率提升。基于 Python 的验证框架，Faver 将 ReAct 推理过程中的“推理验证”与依赖外部协同仿真工具的“事实验证”结合起来，从而建立了一种有效的面向 LLM 的“设计—验证协同”RTL 生成框架。与已有工作不同，本文的关键不在于简单引入 Python 辅助验证，而在于通过函数抽象机制，将 RTL 的周期级执行行为映射为适合 LLM 处理的高层 Python 模型，并通过 Python–Verilog 协同仿真实现逐周期对齐验证。实验结果表明，Faver 在多个测试集和不同模型上均能显著提升 RTL 生成的正确性。

本文的主要工作如下：

1）提出了一种面向 LLM 的 RTL 设计—验证协同框架 Faver，使模型能够通过生成高层语义验证代码实现对电路设计的自动验证，从而摆脱对人工编写测试平台的依赖。

2）设计了一种函数-类抽象模板，将硬件设计中的时钟语义、寄存器语义和状态更新过程映射为事件驱动的 Python 函数表示，从而有效缩小硬件验证与软件验证之间的时空粒度差异。

3）在多个测试集和多种 LLM 上的实验结果表明，Faver 能够稳定提升基于 LLM 的 RTL 生成正确率，在不同模型上最高可带来 14% 的性能提升。

1 相关工作与背景知识

1.1 RTL 设计与验证背景

RTL 是现代数字硬件设计中的基础抽象，用于描述数据与控制信号在时钟周期之间的传递与变换过程。RTL 通常通过硬件描述语言（HDL）进行表达，其中 Verilog 是当前应用最广泛的描述语言之一。RTL 设计主要由 2 类结构组成：一类是由时钟驱动的寄存器单元，用于保存状态并决定电路的时序行为；另一类是寄存器之间的组合逻辑，用于执行算术与逻辑运算，从而实现复杂功能。

与传统软件程序不同，RTL 设计的正确性不仅体现在功能输出上，还体现在周期级执行行为是否符合设计预期。因此，在实际工程中，RTL 验证通常需要同时依赖高层参考模型与被测设计 DUT（device under test）的协同仿真，通过逐周期对比验证其行为一致性。同时，SystemVerilog 断言等机制也被广泛用于处理器功能验证和硬件化验证场景[17]。此外，为保证验证独立性，设计代码与验证代码通常由不同模块或团队实现，从而降低验证偏差带来的风险。这种“设计—验证协同”的开发模式为后续自动化方法提供了重要参考。

1.2 ReAct 范式与 LLM 闭环优化背景

ReAct 是一类面向复杂任务求解的 LLM 推理范式，其核心思想是在“推理—行动—反馈”之间交替迭代：模型在生成中间推理过程的同时，调用外部工

具或执行动作,并根据返回结果不断修正后续决策。与一次性生成式方法相比,ReAct能够更有效地利用环境反馈,在多步任务中逐步逼近正确解,因此已被广泛应用于代码生成与交互式问题求解等场景。

对于RTL生成任务而言,ReAct同样具有天然吸引力。RTL设计并非单次生成即可完成,而通常需要经历“生成—验证—修正”的多轮迭代过程。因此,基于ReAct构建自动闭环的RTL生成框架成为一个自然方向。然而,在RTL场景中,验证环节本身具有较高复杂性:若缺乏稳定且语义一致的验证机制,ReAct框架中的反馈信号可能存在误判,从而削弱闭环优化效果。因此,如何设计高质量的验证机制,成为ReAct在RTL生成中发挥作用的关键。

1.3 基于LLM的RTL代码生成

近年来,LLM的快速发展为RTL代码生成自动化带来了新的可能。通过监督微调(SFT)、强化学习(RL)以及推理增强等技术,多种模型已表现出一定的RTL生成能力[4-5,18-27]。例如,VeriReason通过结合监督学习与强化学习(GRPO),并引入测试平台反馈机制,提升了Verilog代码生成的功能正确性[22]。ScaleRTL通过构建大规模RTL推理数据集并引入测试时扩展策略,在多个基准上实现了较高精度(如在VerilogEval-Human上达到80.4%的pass@1)[23]。

此外,一些工作还尝试通过推理增强与多轮生成策略进一步提升模型性能,例如利用链式推理(Chain-of-Thought)或测试时自修正机制,使模型在生成过程中不断优化其输出。

总体来看,现有方法主要通过增强模型的生成能力来提升RTL代码质量,但其验证过程通常依赖人工构建的测试平台或简单反馈信号,缺乏统一、稳定的自动验证机制。

1.4 基于LLM的自验证与自修正方法

随着RTL生成研究的深入,越来越多工作开始关注“生成—验证—修正”的闭环优化机制。例如,VeriAssist通过生成测试平台和测试用例,并结合仿真反馈对RTL代码进行迭代修正,从而提升设计正确率[11]。

代码修复与调试问题也与LLM驱动的自动程序修复研究密切相关[24]。在调试与修复方向,RTLFixer针对LLM生成Verilog代码中大量语法错误的问题,提出基于ReAct提示与检索增强(RAG)的自动修复框架,在VerilogEval基准上显著提升了语法通过率[25]。HDLDebugger则通过构建高质量训练数据与检索增强机制,实现了较高的代码修复成功率,并验证了其在工业级HDL调试任务中的有效性[26]。此外,HLSDebugger进一步将LLM调试思路扩展到HLS代码逻辑缺陷定位与修复场景[27]。

总体而言,这类方法表明验证反馈对于提升RTL生成质量至关重要。然而,其反馈信号主要来源于TB、编译错误或低层调试信息,缺乏统一的语义表达形式,在复杂时序行为下的稳定性仍然有限。

1.5 基于高层语言辅助的RTL生成与验证

另一类研究尝试利用高层语言(如C或Python)提升RTL生成与验证能力。BetterV通过构建Verilog–C对,并利用跨语言翻译任务实现知识迁移,从而增强模型对RTL语义的理解能力[28]。近期的工作AutoVeriFix提出生成Python参考模型与测试平台,并利用其对Verilog代码进行迭代调试,从而显著提升功能正确性[29]。

尽管上述方法展示了高层语言在RTL生成与验证中的潜力,但其通常将Python或C语言作为测试平台生成工具或辅助验证手段,并未显式建模RTL的周期级执行语义。因此,在需要精确刻画时序行为的场景中,其验证能力仍存在一定局限。

1.6 小结:本文定位与差异

综上所述,现有研究已从“直接生成RTL代码”逐步发展到“生成—验证—修正”的闭环框架,并探索了TB反馈、高层语言辅助等多种路径。然而,这些方法仍存在以下不足:一是验证机制依赖TB或低层反馈信号,缺乏统一的语义表达;二是高层语言方法未能显式建模RTL的周期级执行语义;三是验证结果难以稳定指导生成过程。

与现有工作不同,本文提出的Faver并非简单引入高层语言辅助验证,而是通过函数抽象机制,将RTL的周期级执行行为映射为高层Python模型,并结合Python–Verilog协同仿真实现逐周期对齐验证。在此基础上,Faver进一步构建结构化验证反馈机制,使验证结果能够有效指导RTL生成过程,从而实现更加稳定的性能提升。

2 Faver的ReAct框架

如图2所示,Faver可以被视为一个由生成器和验证器组成的闭环系统。生成器负责根据自然语言规格生成RTL设计;验证器则基于自动构建的高层参考模型,对生成结果进行验证并输出反馈信号。系统运行遵循如下规则:

1)若生成结果通过验证,则输出当前设计;

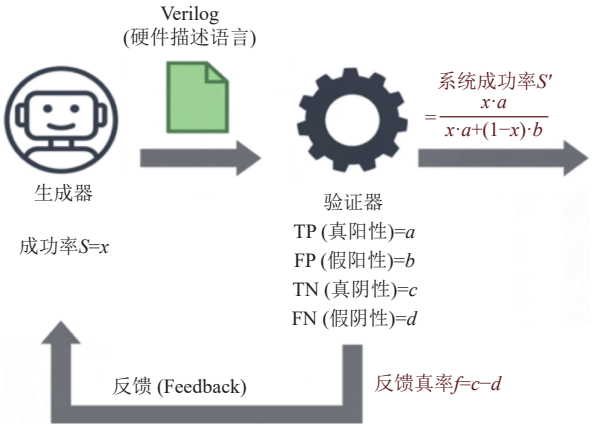


Fig. 2 ReAct framework for LLM RTL design in Faver

图 2 Faver 中面向 LLM RTL 设计的 ReAct 框架

2)若设计未通过验证，则将验证反馈返回生成器，并重新生成设计；

3)若连续 N 次验证失败，则从已有候选结果中选取一个作为最终输出。

在这一框架中，验证器的作用不仅在于判断当前设计是否正确，更在于为后续迭代提供有效反馈，从而影响生成过程的优化方向。

为便于说明验证器在闭环中的作用，可以考虑如下简化分析。设生成器一次生成正确设计的概率为 x，验证器对正确设计判定为通过的概率为真阳率 a，对错误设计误判为通过的概率为假阳率 b。若暂时忽略反馈对后续生成过程的影响，则系统在单轮下输出正确结果的概率主要取决于生成器本身的正确率以及验证器对正确结果的保留能力和对错误结果的过滤能力。在这一理想化情形下，较高的真阳率和较低的假阳率通常有助于提高系统整体输出正确设计的概率。

基于上述分析，本文构建了 Faver，通过自动生成 Python 参考模型及执行 Python–Verilog 协同仿真并形成结构化验证报告，实现对 RTL 设计的迭代修正。本文后续实验表明，在当前任务设置下，Faver 能够提供具有较高有效性的验证反馈，并最终带来整体生成正确率的稳定提升。

3 Faver 构建方法

为实现高效且准确的 RTL 设计-验证协同机制，本文提出了 Faver，该 ReAct 框架使 LLM 能够专注于使用 Python/C 等高层语言编写验证代码，从而获得高质量反馈以提升 RTL 生成效果。如图 3 所示，Faver 主要包括 4 个关键步骤：

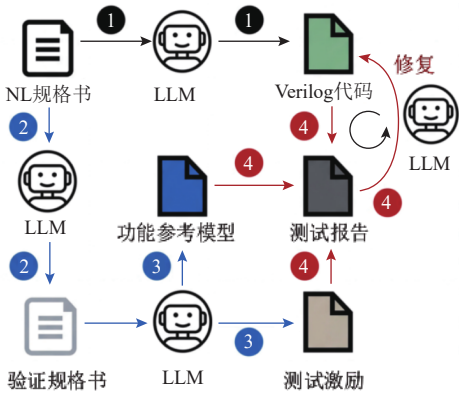


Fig. 3 The main workflow of Faver

图 3 Faver 的主要工作流

1)LLM 读取设计规格，生成 RTL 代码。

2)LLM 读取设计规格，生成验证规格(如图 4 所示)。

3)LLM 依据验证规格，编写功能参考模型(如图 5 所示)，生成测试激励(如图 6 所示)。

4)执行 Python–Verilog 协同仿真，生成验证报告，并将其反馈给 LLM 以进行迭代优化，其过程如图 7 所示。下文将详细介绍上述的具体方法。

3.1 验证语义与适用范围

首先，本文中的 Python 验证模型主要执行的是基于周期抽象(cycle abstraction)的功能一致性验证。即验证 RTL 设计在时钟周期级别上的输入—输出行为是否与参考模型一致，而不是对底层门级时序进行精确建模。

其次，为了在 Python 中表达时序行为，本文采用了一种函数抽象机制，将 RTL 中的时钟驱动执行过程映射为事件驱动的函数调用：

1)将每一个时钟上升沿抽象为一次 step 函数调用；

2)将寄存器、计数器和状态变量映射为 Python 中的类成员变量，用于保存跨周期状态；

3)通过 reset 函数实现电路初始状态的建模；

4)在 step 函数中统一实现状态更新与输出计算，从而模拟 RTL 的周期级执行过程。

在此基础上，Python–Verilog 协同仿真框架对同一组测试激励同时驱动 Python 参考模型与 Verilog 被测设计(DUT)运行，并在统一时序基准下逐周期采集两者输出进行对齐比较。若两者结果一致，则判定当前周期行为匹配；若出现不一致，则进一步分析其属于功能错误、时序错位还是边界条件问题，并将分析结果整理为结构化报告，用于后续迭代优化。

需要说明的是，上述基于 step 函数的周期抽象

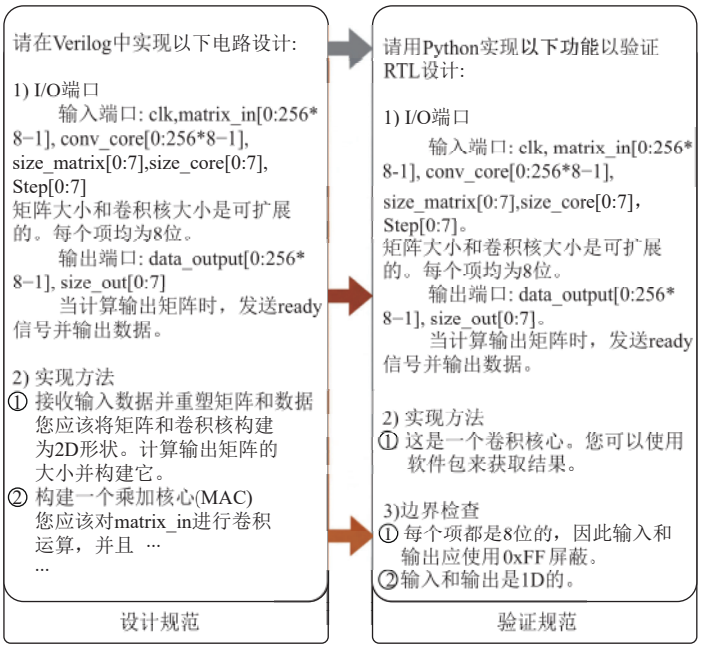


Fig. 4 Verification specification generation

图 4 验证规范生成

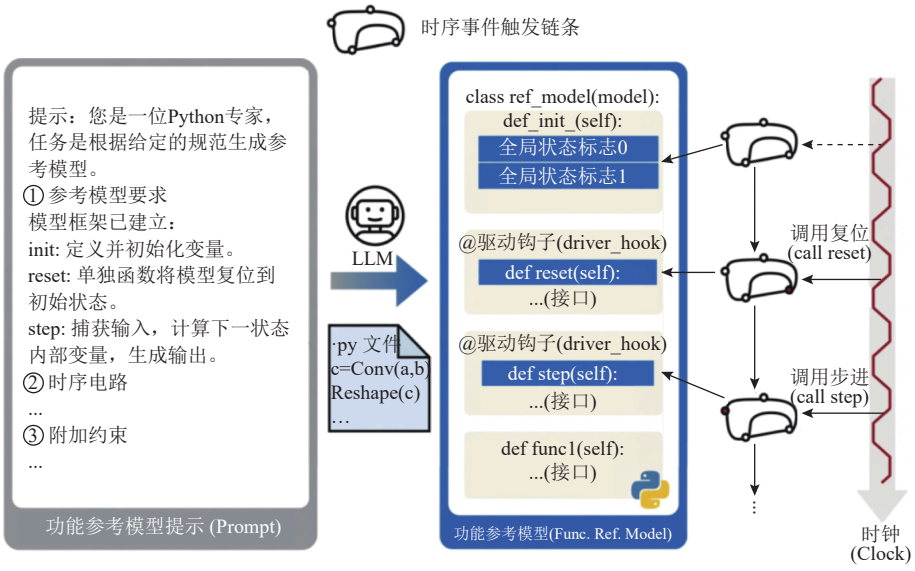


Fig. 5 Reference model generation in Faver

图 5 Faver 中的参考模型生成

主要适用于单时钟域、状态更新边界清晰的 RTL 设计，例如基础时序逻辑和有限状态机(FSM)。对于多时钟域、复杂流水线停顿控制、动态握手协议等场景，简单的每周期一次 step 调用可能不足以完整表达其全部时序语义，因此当前 Faver 框架在这些任务上的适用性仍然有限。

3.2 验证规格生成

我们并不希望 LLM 逐字逐句地在 Python 中复现电路拓扑，而是希望它从更高层次的功能视角编写验证代码。因此，我们不是直接将电路设计规格输入 LLM 来生成验证代码，而是首先将其转换为更高层的验证规格。

为确保由这种高层功能描述生成的验证代码与原始设计保持一致，本文提出了 3 个关键技术组成部分。下面结合图 4 所示的卷积核设计示例进行说明。

1)保留输入/输出端口

我们保留与原设计一致的输入/输出端口定义。输入中的时钟信号将在后续生成参考模型时被提取出来并进行专门处理。

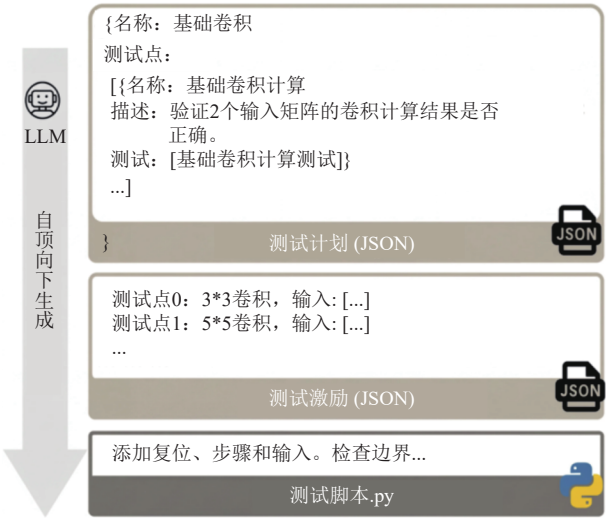


Fig. 6 Top-down test stimulus generation

图 6 自顶向下的测试激励生成

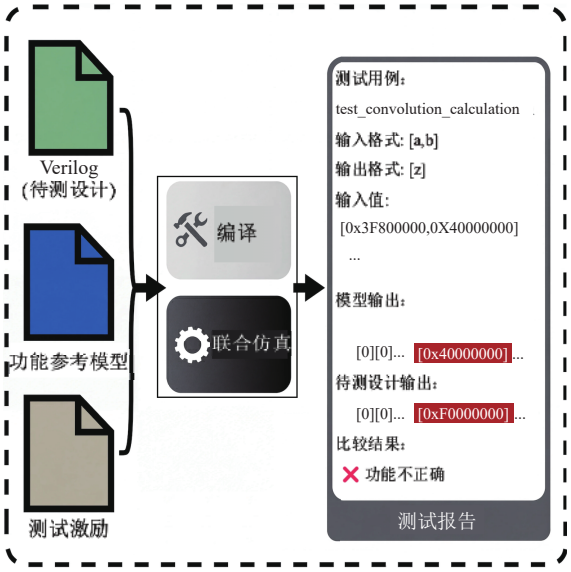


Fig. 7 Process of test reports generation and Verilog code iteration generation

图 7 测试报告生成与 Verilog 代码迭代生成流程

2)功能抽象

在 Verilog 中，功能通常是通过各类时序单元按特定拓扑连接而实现的；而在软件编程中，重点则更多放在“输入如何经过处理得到输出”这一功能映射关系上。因此，LLM 需要基于设计规格对相应功能进行总结与抽象。如图 4 所示，在卷积核的 RTL 设计规格中，详细描述了该设计如何通过乘法器与加法器的互连实现卷积运算。通过阅读该规格，LLM 可以归纳出：“这是一个卷积功能”。在此基础上，它便能够编写简洁的 Python 代码来实现这一功能。

3)边界分析

RTL 代码与 Python 代码在数据规格和表示格式上的差异可能会引入边界条件错误。为避免这类问题，我们要求 LLM 分析 RTL 设计中的边界条件，并在验证规格中将其显式列出。

例如，如图 4 所示，在卷积核设计中，每个数据元素都受到最多 8 位位宽的约束，因此在生成测试输入时必须考虑这一限制。此外，Python 能够较为灵活地处理高维张量，而 Verilog 语法并不具备这一能力，它仅支持固定长度的一维数组。因此，在 Python 与 Verilog 之间进行输入/输出端口匹配时，需要额外注意数据格式转换问题。

3.3 基于时间变量提取类模板的参考模型生成

我们希望 LLM 能够直接生成可与 Verilog 进行协同仿真的 Python 函数，而 Verilog 设计通常包含时钟输入以及相关操作，如寄存器变量和复位逻辑。然而，原生 Python 并不具备这些语义结构，而用于嵌入这些概念的专用领域语言(DSL)不仅数据稀缺，而且通常对 LLM 并不友好。为解决这一问题，Faver 采用了如下设计。

1)用于时钟变量提取的类模板

如图 5 中的功能参考模型所示，其核心机制是一个类模板，主要包括以下内容：

① 将寄存器映射为全局变量。将寄存器变量映射为全局变量(类级变量)，并在构造函数中初始化，从而模拟类似存储器的与时序相关的行为。

② 统一函数接口。提供一个外部的 step 函数，用于接收输入，执行设计功能并输出结果。

③ 复位函数。提供独立的 reset 函数，可由外部调用，用于重新初始化输出、全局变量以及其他有状态组件。

④ 附加函数。保留若干函数桩，由 LLM 补全具体逻辑，从而将设计划分为多个模块，并通过统一的 step 接口进行访问。

2)将时钟视为事件

如图 5 中最右边的时钟事件序列所示，我们将时钟输入视作一种“事件”，从而避免在 Python 代码中直接处理时钟信号。具体而言，每一个时钟上升沿都对应一次“调用 step”操作，并携带一个输入向量，因此整个电路运行过程可以被表示为一系列“call step”事件。在类内部，功能逻辑通过 step 函数被调用，它负责获取输入并返回输出。

3)填充类模板

在构建好类模板之后，我们再让 LLM 补全其中所需的具体逻辑。

① Faver 首先提供一个类模板，其中包括：在 init

函数中的全局变量声明、reset 函数、用于跟踪 RTL 设计时序输入序列的 step 函数，以及若干内部函数桩。

② 随后，结合验证规格和适当的提示信息，LLM 编写卷积函数，并将其嵌入 step 函数中；同时生成用于处理边界条件的 reshape 函数，并将其纳入 step 的执行流程。

③ 最终，LLM 生成一个能够与 Verilog 无缝协同仿真的、基于 Python 的卷积参考模型。

3.4 自顶向下的测试激励生成

测试激励生成的核心挑战在于：如何在高层语义覆盖和功能相关性与底层数据流精确性之间取得平衡。从高层语义角度来看，测试激励需要具备有意义的时序功能，并能够充分覆盖设计的功能空间，这正是 LLM 较为擅长的方面；然而，在细粒度数据流层面，如何保证设计被正确复位，以及如何准确处理边界条件，则是 LLM 较难可靠完成的任务。

图 6 的核心思想是基于自顶向下的设计思想来结合 LLM 与基于规则的生成器各自的优势。

1）高层测试规划。利用 LLM 制定测试计划，明确每个测试用例所针对的功能点，从而保证对功能空间的充分覆盖。

2）时序数据生成。由 LLM 为每个测试用例生成时间序列数据，确保时序输入与目标功能具有较强相关性。

3）基于规则的激励细化。在底层数据流层面，使用专门的规则代码对测试激励进行修正和完善，具体包括：插入基于规则的复位操作，确保电路进入有效工作状态；对数据流执行边界检查与修正。

如图 6 所示，在卷积电路验证的示例中，首先要求 LLM 从高层生成覆盖不同卷积规格的测试用例，以保证较广的功能覆盖。接着，在每个测试用例中，由 LLM 提供逐时刻的数据流，以验证预期功能。

随后，基于规则的代码从规格中提取复位模式（如同步/异步、高有效/低有效），并将相应的复位信号附加到各个测试用例中。同时，它还依据规格中的位宽约束和格式约束，对数据流进行边界检查与修正。经过上述处理后，最终得到一组高质量且精确的测试激励。

3.5 协同仿真、反馈和迭代

如图 7 所示，我们采用协同仿真来检测设计与参考模型之间是否存在不一致。对于同一组测试激励，Python 参考模型与 Verilog DUT 在统一仿真框架下同步执行，并逐周期输出结果。若两者结果一致，则判定验证通过，并接受该设计。若出现不匹配，则系统进一步对输出序列和关键波形进行比较，判断错误属于纯功能错误、时序不匹配还是边界条件问题。随后，我们将比较结果解析为统一的数据序列描述，并整合为结构化验证报告，反馈给 LLM 以进行迭代优化。

基于此，Faver 将 ReAct 推理过程中的“推理验证”与依赖外部协同仿真工具的“事实验证”高效结合。为避免系统陷入无限循环，我们设定了迭代次数阈值。具体而言，一旦验证通过，即将当前 RTL 作为最终输出；否则继续重复上述过程。当迭代轮数达到预设阈值后，系统将从此前生成的候选结果中随机选取一个作为输出。

综上，Faver 并非简单将 Python 作为辅助测试脚本使用，而是通过“验证规格—函数抽象参考模型—自顶向下测试激励—协同仿真反馈”这一完整流程，将高层语义建模与周期级行为验证结合起来，从而为 RTL 生成提供更稳定的闭环反馈。

4 实现与实验评估

Faver 基于 Toffee[30] 实现。Toffee 是构建在 Verilator 和 Picker[31] 之上的 Python/Verilog 协同仿真平台。Faver 的自动化调用框架、模型模板以及提示词生成模块均采用 Python 实现。为避免系统陷入无限迭代，本文将连续失败后的输出阈值设为 5。

在实验中，我们选取了 3 种具有代表性的模型作为生成器，包括 Qwen2.5-7B-SFT、DeepSeek-R1-0528[2] 和 Kimi K2[32]。其中，Qwen2.5-7B-SFT 基于我们自行收集的数据进行监督微调（SFT）训练。具体而言，我们在一个经过筛选的数据集上进行了训练，该数据集包含超过 5 000 组自然语言描述与对应 RTL 代码实现的配对样本。训练过程中，我们采用 Hugging Face Transformers 作为训练框架，并使用 LoRA[33] 方法在尽量保持性能的同时降低计算开销。

实验评测采用学术界 2 个公开 Verilog 代码生成基准：RTLLM[34] 和 VerilogEval[35]。其中，VerilogEval 主要评估自然语言到 RTL 代码生成的功能正确性，RTLLM 则进一步涵盖更复杂的 RTL 设计任务。

为全面评估模型性能，本文采用 2 类指标。第 1 类是基础生成指标，即 pass@k，用于衡量模型在无验证辅助情况下的生成能力。具体而言，pass@1 表示首个生成样本通过语法和功能检查的比例，pass@5 表示 5 次独立采样中至少存在 1 个正确设计的比例。第 2 类是系统级指标，用于衡量设计—验证协同闭环

下的整体性能。具体定义如下：

1）sys_sel_pass@1：指设计—验证协同系统在最多进行 N 次生成后输出的单个设计的通过率，其中本文实验中 N=5；

2）sys_inner_pass@5：指在最多 5 次内部迭代过程中，若存在任一生成结果通过检查，则记为成功。

需要说明的是，pass@k 与 sys_* 指标对应不同评价维度：前者反映模型的基础生成能力，后者反映验证闭环下的系统整体输出性能。尤其是，sys_inner_pass@5 在语义上与 pass@5 相近，但由于其基于带反馈的迭代生成过程，与独立采样的 pass@k 不完全等价。后文将分别从基础生成性能和系统级性能 2 个维度展开分析。

4.1 总体结果与前沿方法对比

为全面评估本文方法的性能，表 1 和表 2 分别从基础生成能力与设计—验证协同框架 2 个维度进行了对比分析。其中，表 1 采用 pass@k 指标，反映模型在无验证辅助情况下的 RTL 生成能力；表 2 采用系统级指标（sys_sel_pass@1 与 sys_inner_pass@5），用于衡量引入验证反馈后的整体性能。

Table 1 Comparison of Basic Generation Capability of RTL

表 1 RTL 基础生成能力对比

方案	模型	VerilogEval		RTLLM	
		pass@1	pass@5	pass@1	pass@5
基础模型	DeepSeek-R1-0528	0.83	0.88	0.74	0.77
	Kimi K2	0.74	0.79	0.67	0.77
	GPT-4O	0.60	0.71	0.42	0.66
	Qwen2.5-Coder-32B-Instruct	0.48	0.58	0.48	0.68
微调模型	CodeV-CL-7B	0.45	0.60	0.40	0.62
	RTLCoder-Mistral-7B	0.37	0.46		0.48
	BetterV	0.49	0.50		
	CraftRTL-SC2-15B	0.68	0.72	0.49	0.66
	Qwen2.5-7B-SFT			0.40	0.49

首先，从表 1 可以看出，近年来通过模型结构优化、数据增强以及推理策略改进，基础模型在 RTL 生成任务上已取得一定进展。例如，DeepSeek-R1-0528 与 Kimi K2 在 2 个基准上均表现出较高的 pass@1 与 pass@5 指标。

其次，表 2 展示了设计—验证协同框架的性能对比。可以观察到，相比于直接生成或简单验证辅助方法，引入验证反馈能够显著提升 RTL 的 pass@k。例如，在 RTLLM 数据集上，DeepSeek-R1-0528 的 pass@1 和 pass@5 分别为 74% 和 77%，而引入 Faver 后，其 sys_sel_pass@1 提升至 83%，sys_inner_pass@5 提升至 89%。值得注意的是，Faver 的 sys_sel_pass@1 比原始模型的 pass@5 更高。对于 Qwen2.5-7B-SFT，Faver 模型的 sys_inner_pass@5 相比原模型提升了 14%。总体而言，Faver 为基础模型带来了约 10% 的 pass@k。进一步地，与现有代表性方法相比，Faver 在多个模型与数据集上均取得了更优的结果。例如，在 VerilogEval 数据集上，Faver-DeepSeek-R1-0528 在 sys_sel_pass@1 和 sys_inner_pass@5 上分别达到 0.85 和 0.90，均优于对应的 Python Verify 与 VeriAssist 方法；在 RTLLM 数据集上，Faver 同样保持了稳定优势。而相比 AutoVeriFix，Faver 进一步引入了函数抽象机制，对 RTL 的周期级执行语义进行显式建模，使 Python 参考模型能够在统一时序语义下进行逐周期对齐验证，从而提升验证结果的稳定性与可靠性。

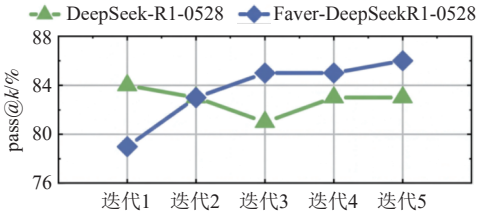
进一步地，在图 8 中，我们比较了 Faver-DeepSeek-

Table 2 Performance Comparison of Design-Verify Collaborative Framework

表 2 设计—验证协同框架性能对比

方案	模型	VerilogEval		RTLLM	
		sys_sel_pass@1	sys_inner_pass@5	sys_sel_pass@1	sys_inner_pass@5
验证辅助设计	DeepSeek-R1-0528+Python Verify	0.83	0.86	0.74	0.80
	Kimi K2+Python Verify	0.74	0.81	0.69	0.77
	VeriAssist[11]+Claude-3	0.64[a]	0.70[b]		0.65[b]
	VeriAssist[11]+GPT-4	0.68[a]	0.78[b]		0.76[b]
	ScaleRTL-32B		0.90[b]		0.78[b]
	AutoVeriFix+ GPT-4		0.84[b]		0.84[b]
本文方案	Faver-DeepSeek-R1-0528	0.85	0.90	0.83	0.89
	Faver-Kimi K2	0.81	0.83	0.74	0.83
	Faver-Qwen2.5-7B-SFT			0.51	0.63

注：对于本文方案，报告的是系统级指标 sys_sel_pass@1 和 sys_inner_pass@5；对于部分外部方法，由于原文未报告同口径的系统级指标，表中采用其论文中公开的 pass@1 和 pass@5 结果，其中上标 a 表示 pass@1，上标 b 表示 pass@5，相关结果主要用于参考比较。



注：数据采集自 VerilogEval。

Fig. 8 pass@k of RTL within Faver framework over five iterations

图 8 Faver 框架下 RTL 在 5 轮迭代中的 pass@k

R1-0528 与 DeepSeek-R1-0528 在 VerilogEval 数据集上、验证迭代过程中各轮生成结果的 pass@k 变化情况。结果表明，原始 DeepSeek-R1-0528 的 pass@k 呈现出较为随机的波动，而 Faver-DeepSeek-R1-0528 在验证反馈信号的作用下，pass@k 表现出持续而稳定的提升趋势。

4.2 函数抽象验证机制分析

为进一步分析 Faver 性能提升的来源，本文构建了多种对比基线，并在相同实验设置下进行比较，包括：基础模型（不使用验证机制）、基线-V（由 LLM 生成 Verilog testbench 并进行仿真验证）、基线-L（由 LLM 直接判断设计是否满足规格）、基线-P（由 LLM 生成 Python 验证程序）以及完整的 Faver 方法。相关结果如图 9 所示。

从图 9 可以看出，在不同模型与数据集设置下，Faver 均显著优于所有对比方法，而基础模型仅表现出有限甚至不稳定的性能提升。例如，基线-V 在部分任务中能够提供一定程度的验证反馈，但由于 testbench 本身生成难度较高，其整体效果不稳定；基线-L 直接依赖 LLM 进行判别，其判断结果往往缺乏一致性；基线-P 虽然引入了高层语言验证，但由于缺乏统一的执行语义建模，其验证信号仍然存在较大噪声。

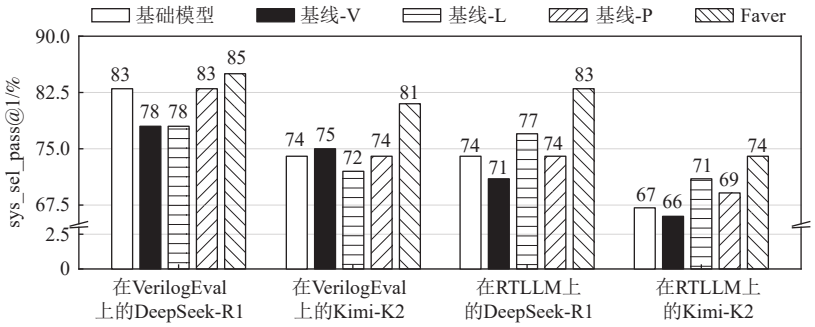


Fig. 9 Analysis results on function abstract verification

图 9 对于函数抽象验证的分析结果

上述结果表明，性能提升并非来源于“使用 Python 本身”，而是来源于对验证过程的结构化建模。具体而言，Faver 通过函数抽象机制，将 RTL 中的时钟驱动执行过程映射为 Python 中的 step 函数调用，使寄存器状态与周期行为得到显式表达。同时，协同仿真框架在统一时序基准下逐周期对齐 Python 参考模型与 Verilog DUT 的输出，从而能够提供更加准确和一致的验证信号。

4.3 消融实验

为进一步分析 Faver 各组成部分的贡献，本文设计了消融实验，并选取 DeepSeek-R1-0528 和 Kimi K2 两个模型，在 RTLLM 与 VerilogEval 两个基准上进行评测。实验中比较了以下配置：基础模型（不使用验证机制）、Faver1（直接使用 Python 编写参考模型与测试激励，但不采用函数抽象机制）、Faver2（采用函数抽象参考模型，但不使用分层测试激励生成机制）以及完整的 Faver 方法。相关结果如图 10 所示。

从图 10 可以看出，Faver 在所有设置下均优于其消融版本，且整体趋势在不同模型与数据集上保持一致。具体而言，Faver1 相较于基础模型仅表现出有限提升，说明“直接使用 Python”并不足以稳定提升 RTL 生成质量；在引入函数抽象参考模型后（Faver2），系统性能出现明显提升；而进一步引入完整的测试激励生成机制后，完整 Faver 在各项指标上均达到最优表现。

这些结果表明，Faver 的性能提升并非来源于单一组件，而是来源于“参考模型构建 + 测试激励生成 + 协同反馈机制”的协同作用。

4.4 验证器统计结果与说明

为进一步分析验证机制的行为，本文统计了验证器在实验过程中的判别结果，包括真阳率（TP）、假阳率（FP）、真阴率（TN）和假阴率（FN），相关结果如图 11 所示。

可以看出，在当前实验设置下，验证器的 TP 高于 FP，同时 TN 也高于 FN，说明其能够在整体上提供较为有效的判别信号。这意味着，在多数情况下，验证器能够正确识别生成结果的正确性，并为后续迭代提供有意义的反馈。

需要强调的是，上述统计结果仅反映验证器在当前任务上的判别能力。由于验证反馈会进一步影响后续生成过程，其整体作用不仅取决于单次判别的准确性，还与反馈的稳定性以及生成器对反馈的利用方式密切相关。因此，这些结果主要作为经验性观察，用于辅助理解验证机制的行为。

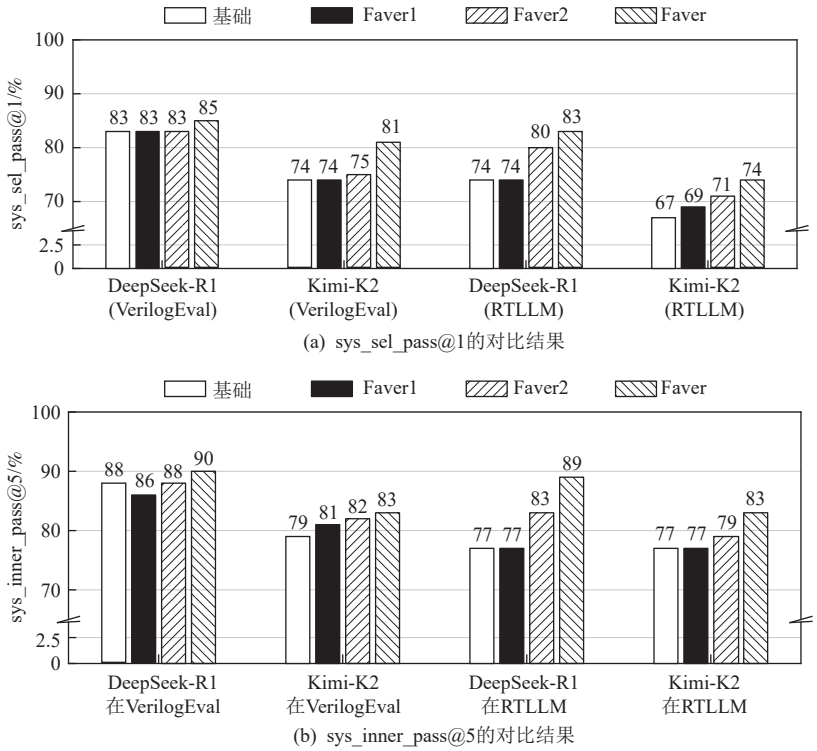


Fig. 10 Ablation study for Faver

图 10 Faver 的消融实验

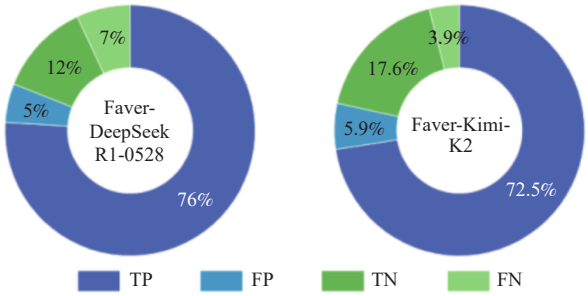


Fig. 11 TP, FP, TN, FN of the verifier for Faver on VerilogEval dataset

图 11 Faver 在 VerilogEval 数据集上验证器的 TP、FP、TN 和 FN

4.5 时序电路与组合电路对比分析

为进一步分析 Faver 在不同类型电路上的表现，本文将测试样例划分为组合逻辑电路与时序电路，并分别统计其性能，相关结果如图 12 所示。从图 12 可以看出，在 2 个数据集上，Faver 在组合逻辑电路与时序电路上均优于基线方法。其中，在组合逻辑电路上的性能提升稳定，而在时序电路上的提升更显著。

这一现象说明，函数抽象验证机制更适用于功能语义清晰、状态规模适中且周期行为可显式展开的设计场景。在组合逻辑电路中，设计行为主要由输入直接决定，参考模型较易构建；而在时序电路中，由于存在跨周期状态依赖与复杂控制逻辑，参考模型的构建难度更高，从而对验证效果产生一定影响。

总体而言，Faver 在 2 类电路上均表现出良好的泛化能力，同时其性能差异也反映了不同电路类型在验证建模难度上的内在差异。

5 讨 论

本文提出了 Faver，一个基于函数抽象验证的 RTL 生成 ReAct 推理框架。针对 LLM 驱动的 RTL 设计任务，构建了一种将高层语义验证与电路周期级行为结合的自动闭环生成-验证机制。Faver 通过函数-类抽象模板，将硬件设计中的时钟语义、寄存器状态及周期级操作映射为事件驱动的 Python 函数，从而有效缩小了硬件验证与软件验证之间的时空粒度差异，使 LLM 能够在理解高层功能语义的基础上完成准确的 RTL 验证。需要说明的是，当前框架主要适用于单时钟域、基础时序逻辑和有限状态机等电路，对于多时钟域或复杂流水控制电路仍存在一定局限，这也是未来的重要研究方向。

同时，本文中关于 TP、FP、TN、FN 的分析仅用于直观说明验证器质量与系统性能之间的关系，其隐含假设是各轮生成过程相互独立，且验证结果不会影响后续生成行为。而在实际的 Faver 框架中，验

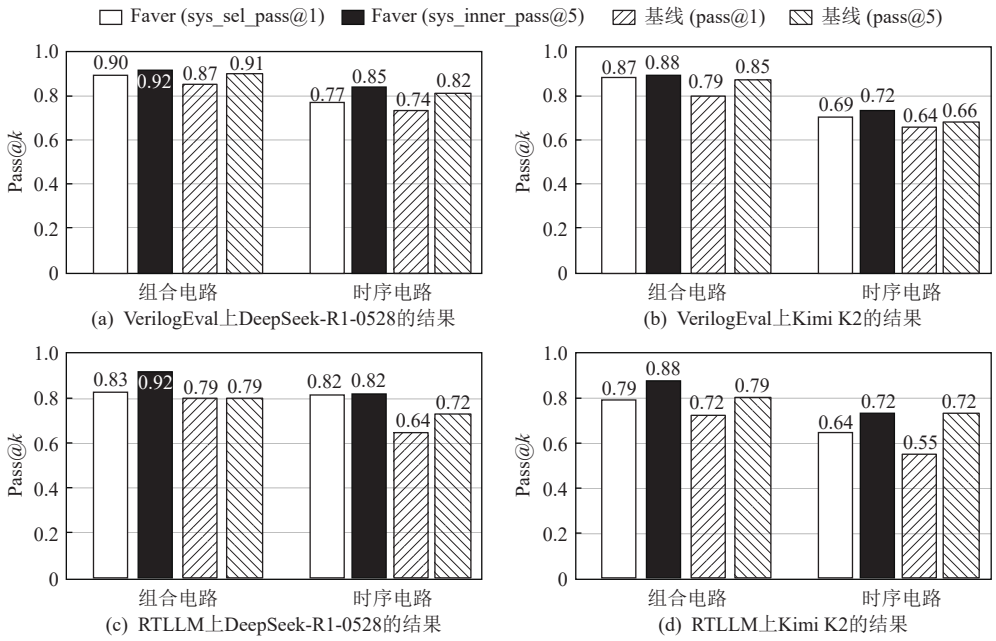


Fig. 12 Performance comparison across different circuit types

图 12 不同电路类型上的性能对比

证器输出的反馈会进一步参与下一轮生成，因此验证器并非一个静态分类器，而是闭环优化过程中的反馈模块。此时，验证器的作用不仅体现在单次判别准确率上，还体现在其反馈是否能够为后续生成提供稳定、有效的优化信号。针对上述问题，一种可行的缓解思路是引入异源验证机制，例如使用不同模型或不同提示策略分别生成 RTL 与参考模型，从而降低错误相关性。此外，可以结合外部验证信号（如人工构建的测试用例、形式验证工具或已有黄金模型）对生成结果进行补充校验，以进一步提升验证可靠性。总体而言，上述问题反映了基于 LLM 的生成—验证闭环在“语义同源性”方面的固有限制，而如何构建更加独立且鲁棒的验证机制，是未来值得进一步研究的重要方向。

6 总结与未来工作

本文提出了 Faver，一个基于函数抽象验证的 RTL 生成 ReAct 推理框架。Faver 通过函数-类抽象模板，将硬件设计中的时钟语义、寄存器状态及周期级操作映射为事件驱动的 Python 函数，从而有效缩小了硬件验证与软件验证之间的时空粒度差异，使 LLM 能够在理解高层功能语义的基础上完成准确的 RTL 验证。在多种 SFT 模型和开源模型上的实验结果表明，Faver 显著提升了 RTL 生成的正确性，系统输出通过率在不同模型上可提升最高 14%。同时，通过 ReAct 协同推理机制，Faver 将“推理验证”和“事实验证”有机结合，实现了无需人工编写测试平台即可进行自动化自验证的 RTL 生成。综上所述，Faver 为 LLM 驱动的 RTL 自动化生成提供了一条可行且高效的技术路线，同时验证了函数抽象和协同推理在提升生成正确性方面的重要作用。未来工作可以进一步探索该方法在更大规模电路设计、复杂状态机及多模块并行加速器上的应用，以提升 LLM 在硬件设计自动化中的泛化能力和实用性。

作者贡献声明：穆嘉楠负责 Faver 整体方案设计并撰写论文；史茗宇负责部分算法思路和实验统计；晁志腾负责部分算法思路并修改论文；叶靖、李华伟提出指导意见并修改论文。穆嘉楠和史茗宇为共同第一作者。

参 考 文 献

[1] Chang Kaiyan, Wang Ying, Ren Haimeng, et al. ChipGPT: How far are we from natural language hardware design[J]. arXiv preprint arXiv: 2305.14019, 2023

[2] Guo Daya, Yang Dejian, Zhang Haowei, et al. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning[J]. arXiv preprint arXiv: 2501.12948, 2025

[3] Ho C T, Ren H, Khailany B. Verilogcoder: Autonomous Verilog coding agents with graph-based planning and abstract syntax tree (AST)-based waveform tracing tool[C]//Proc of the 39th AAAI Conf on Artificial Intelligence. Philadelphia: AAAI, 2025: 300−307

[4] Liu Mingjie, Ene T D, Kirby R, et al. Chipnemo: Domain-adapted LLMs for chip design[J]. arXiv preprint arXiv: 2311.00176, 2023

[5] Pei Zehua, Zhen Hui-Ling, Yuan Mingxuan, et al. Betterv: Controlled Verilog generation with discriminative guidance[J]. arXiv preprint arXiv: 2402.03375, 2024

[6] Liu Shang, Fang Wenji, Lu Yao, et al. Rtlcoder: Fully open-source and efficient LLM-assisted RTL code generation technique[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2025, 44(4): 1448−1461

[7] Thakur S, Ahmad B, Fan Zhenxing, et al. Benchmarking large language models for automated Verilog RTL code generation[C]// Proc of the 2023 Design, Automation & Test in Europe Conf & Exhibition. Piscataway, NJ: IEEE, 2023: 1−6

[8] Thakur S, Ahmad B, Pearce H, et al. Verigen: A large language model for Verilog code generation[J]. ACM Transactions on Design Automation of Electronic Systems, 2024, 29(3): 1−31

[9] Zhao Yang, Huang Di, Li Chongxiao, et al. Codev: Empowering LLMs with HDL generation through multi-level summarization[J]. arXiv preprint arXiv: 2407.10424, 2024

[10] Zhao Yujie, Zhang Hejia, Huang Hanxian, et al. Mage: A multi-agent engine for automated RTL code generation[J]. arXiv preprint arXiv: 2412.07822, 2024

[11] Huang Hanxian, Lin Zhenghan, Wang Zixuan, et al. Towards LLM-powered Verilog RTL assistant: Self-verification and self-correction[J]. arXiv preprint arXiv: 2406.00115, 2024

[12] Qiu Ruidi, Zhang G L, Drechsler R, et al. Correctbench: Automatic testbench generation with functional self-correction using LLMs for HDL design[C]//Proc of the 2025 Design, Automation & Test in Europe Conf. Piscataway, NJ: IEEE, 2025: 1−7

[13] Zhao Yujie, Wu Zhijing, Zhang Hejia, et al. Pro-v: An efficient program generation multi-agent system for automatic RTL verification[J]. arXiv preprint arXiv: 2506.12200, 2025

[14] Snyder W. Verilator and systemperl[C]//Proc of North American SystemC Users' Group, Design Automation Conf. New York: ACM, 2004: 122−148

[15] Williams S, Baxter M. Icarus Verilog: Open-source Verilog more than a year later[J]. Linux Journal, 2002, 2002(99): 3−3

[16] Rosser B J. Cocotb: A Python-based digital logic verification framework[C]//Proc of Micro-electronics Section Seminar. Geneva: CERN, 2018

[17] Zhang Ziqing, Shi Kan, Xu Shuoxiang, et al. Design of SystemVerilog assertions hardware towards efficient processor functional verification[J]. Journal of Computer Research and Development, 2024, 61(6): 1436−1449 (in Chinese)
(张子卿, 石侃, 徐烁翔, 等. 面向处理器功能验证的硬件化 SystemVerilog 断言设计[J]. 计算机研究与发展, 2024, 61(6): 1436−1449)

[18] Ma Ruiyang, Yang Yuxin, Liu Ziqian, et al. Verilogreader: LLM-aided hardware test generation[C]//Proc of the 2024 IEEE LLM Aided Design Workshop. Piscataway, NJ: IEEE, 2024: 1−5

[19] Liu Mingjie, Tsai Y D, Zhou Wenfei, et al. Craftrtl: High-quality synthetic data generation for Verilog code models with correct-by-construction non-textual representations and targeted code repair[J]. arXiv preprint arXiv: 2409.12993, 2024

[20] Hurst A, Lerer A, Goucher A P, et al. GPT−4O system card[J]. arXiv preprint arXiv: 2410.21276, 2024

[21] Qwen Team. Qwq-32b: A 32b inference model enhanced with large-scale reinforcement learning[EB/OL]. 2025[2025-03-10]. https://www.alibabagroup.com/document-1834731580057190400

[22] Wang Y, Sun G, Ye W, et al. VeriReason: Reinforcement learning with testbench feedback for reasoning-enhanced Verilog generation[J]. arXiv preprint arXiv: 2505.11849, 2025

[23] Deng C, Tsai Y D, Liu G T, et al. ScaleRTL: Scaling LLMs with reasoning data and test-time compute for accurate RTL code generation[C]//Proc of 2025 ACM/IEEE 7th Symp on Machine Learning for CAD (MLCAD). Piscataway, NJ: IEEE, 2025: 1−9

[24] Xu Pengyu, Kuang Boyu, Su Mang, et al. Survey of large-language-model-based automated program repair[J]. Journal of Computer Research and Development, 2025, 62(8): 2040−2057 (In Chinese)
(许鹏宇, 况博裕, 苏铓, 等. 基于大语言模型的自动代码修复综述[J]. 计算机研究与发展, 2025, 62(8): 2040−2057)

[25] Tsai Y D, Liu M, Ren H. RTLFixer: Automatically fixing RTL syntax errors with large language model[C]//Proc of the 61st ACM/IEEE Design Automation Conf. Piscataway, NJ: IEEE, 2024: 1−6

[26] Yao X, Li H, Chan T H, et al. HDLDebugger: Streamlining HDL debugging with large language models[J]. ACM Transactions on Design Automation of Electronic Systems, 2025, 30(6): 1−26

[27] Wang J, Liu S, Lu Y, et al. HLSDebugger: Identification and correction of logic bugs in HLS code with LLM solutions[C]//Proc of 2025 IEEE/ACM Int Conf On Computer Aided Design (ICCAD). Piscataway, NJ: IEEE, 2025: 1−9.

[28] Pei Z, Zhen H, Yuan M, et al. BetterV: Controlled Verilog generation with discriminative guidance[C]//Proc of the 41st Int Conf on Machine Learning. 2024.

[29] Tan Y, Meng X, Jiang Z, et al. AutoVeriFix: Automatically correcting errors and enhancing functional correctness in LLM-generated Verilog code[C]//Proc of 2026 31st Asia and South Pacific Design Automation Conference (ASP-DAC). Piscataway, NJ: IEEE, 2026: 526−532

[30] XS-MLVP. Toffee[EB/OL]. 2025[2025-03-10]. https://github.com/XS-MLVP/toffee

[31] XS-MLVP. Picker[EB/OL]. 2025[2025-03-10]. https://github.com/XS-MLVP/picker

[32] Kimi Team, Bai Yifan, Bao Yiping, et al. Kimi K2: Open agentic intelligence[J]. arXiv preprint arXiv: 2507.20534, 2025

[33] Hu E J, Shen Y, Wallis P, et al. LoRA: Low-rank adaptation of large language models[J]. arXiv preprint arXiv: 2106.09685, 2021

[34] Lu Yao, Liu Shang, Zhang Qijun, et al. RTLLM: An open-source benchmark for design RTL generation with large language model[C]// Proc of the 29th Asia and South Pacific Design Automation Conf. Piscataway, NJ: IEEE, 2024: 722−727

[35] Liu Mingjie, Pinckney N, Khailany B, et al. VerilogEval: Evaluating large language models for Verilog code generation[J]. arXiv preprint arXiv: 2309.07544, 2023



Mu Jianan, born in 1997. PhD, assistant professor. His main research interest includes intelligent and high-reliability design of domain-specific processors.

穆嘉楠, 1997 年生。博士, 特别研究助理。主要研究方向为专用处理器智能化与高可靠设计。



Shi Mingyu, born in 2003. Undergraduate. His main research interests include integrated circuit design and electronic design automation.

史茗宇, 2003 年生。本科生。主要研究方向为集成电路设计、电子设计自动化。



Ye Jing, born in 1988. Associate professor. His main research interests include VLSI testing, hardware security, and cryptographic circuits.

叶　靖, 1988 年生。副研究员。主要研究方向为 VLSI 测试、硬件安全、密码电路。



Chao Zhiteng, born in 1998. PhD, special research assistant. His main research interest includes integrated circuit testing and verification.

晁志腾, 1998 年生。博士，特别研究助理。主要研究方向为集成电路测试与验证。



Li Huawei, born in 1974. Professor. Her main research interests include testing of very large-scale integrated (VLSI) circuits, approximate computing architectures, and machine learning accelerators.

李华伟, 1974 年生。研究员。主要研究方向为超大规模集成电路测试、近似计算体系结构、机器学习加速器。