计 算 机 研 究 与 发 展
Journal of Computer Research and Development

https://crad.ict.ac.cn
63(6): 1437−1450, 2026

本期推荐

SegTracer: 大语言模型辅助的多 FPGA 原型系统时延探测工具设计与实现

张聪武[1,2] 王亚洲[1,2] 王攀宇[1,2] 陈明宇[1,2] 包云岗[1,2] 张 科[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 100049）

（zhangcongwu@ict.ac.cn）

SegTracer: LLM-Assisted Design and Implementation of a Latency-Probing Tool for Multi-FPGA Prototyping Systems

Zhang Congwu[1,2], Wang Yazhou[1,2], Wang Panyu[1,2], Chen Mingyu[1,2], Bao Yungang[1,2], and Zhang Ke[1,2]

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

[2]（*University of Chinese Academy of Sciences*, *Beijing* 100049）

Abstract Large language models (LLMs) are increasingly being incorporated into chip and FPGA development workflows. However, existing studies have mainly focused on RTL generation, assertion generation, or localized debugging, while systematic case studies covering end-to-end engineering workflows remain limited. Using SegTracer, a data-path latency probing tool for multi-FPGA prototyping systems, as a case study, we investigate the role of a human-led, LLM-assisted, verification-driven development paradigm in practical IP implementation. During the design and deployment of SegTracer, LLMs are used for requirements refinement, solution comparison, design document consolidation, SpinalHDL implementation, generation of register-access and verification scripts, and troubleshooting during localized debugging; human developers are responsible for key mechanism definition, literature verification, clock-domain-crossing and handshake-semantics handling, waveform analysis, and experimental interpretation. Technically, SegTracer employs independent probe packets together with a packet generation and monitoring module and a unified timestamp module. Through stack-based recording on the forward path, in-place subtraction on the return path, and centralized endpoint statistics, it achieves segmented latency measurement and distribution analysis without relying on global clock synchronization. Simulation and FPGA board-level experiments show that the tool can support latency decomposition and tail-bottleneck localization in both QSFP direct-link and PCIe hardware-software collaborative scenarios. The case study demonstrates that large language models can significantly improve design iteration, script development, and joint debugging and verification efficiency, while human leadership remains essential in key aspects such as domain-specific semantic understanding, clock-domain-crossing handling, timing correctness, and result interpretation.

Key words Role-Shell architecture; segment latency measurement; in-place computation; clock domain crossing (CDC); tail latency capture; LLM-assisted design

摘 要 大语言模型正逐步进入芯片与 FPGA 研发流程，但现有工作多集中于 RTL 生成、断言生成或局部调试，缺少结合真实工程闭环的系统性案例。以多 FPGA 原型系统中的数据通路时延探测工具 SegTracer 为对象，研究人工主导、大模型辅助、验证驱动的研发方式在实际 IP 实现中的作用。围绕 SegTracer 的设

收稿日期：2026-03-17；**修回日期**：2026-05-06

通信作者：张科（zhangke@ict.ac.cn）

计与落地，大模型被用于需求重构、方案比较、设计文档收敛、SpinalHDL 实现、寄存器访问与验证脚本生成，以及局部调试中的错误排查；人工负责关键机制定义、文献核验、跨时钟域与握手语义处理、波形分析和实验归因。技术上，SegTracer 采用独立探测包、包生成监控模块和统一时间戳模块协同工作，通过去程压栈、返程原位减法和端点集中统计，实现了不依赖全局时钟同步的分段时延测量与分布分析。仿真与 FPGA 上板实验表明，该工具能够在 QSFP 直连和 PCIe 软硬件协同场景下完成时延分解与尾部瓶颈定位。案例结果表明，大模型有助于提升设计收敛、脚本开发和联调验证效率，但在领域语义理解、跨时钟域处理、时序正确性和结果解释等关键环节仍需人工主导。

关键词 Role-Shell 架构；分段时延测量；原位计算；跨时钟域；尾部延迟捕获；大模型辅助设计

中图法分类号 TP391.72；TP302.7

DOI: 10.7544/issn1000-1239.202660162 **CSTR:** 32373.14.issn1000-1239.202660162

近年来，大语言模型(large language model, LLM)(以下简称“大模型”)在自然语言理解、代码生成与交互式问题求解方面快速进展，推动其从通用软件开发逐步进入芯片设计的多个环节，包括规格理解与文档整理、RTL/HDL 代码框架生成、验证脚本与断言生成、调试辅助以及流程自动化等。已有研究围绕 RTL 生成评测[1-4]、规格到断言转换[5-6]、调试修复与 Agent 化工作流[7-8]等方向提出了多种方法与基准；面向 Agent 化设计流程、高层综合辅助以及 EDA 流程自动化的研究也正在出现[9-11]，初步证明了大模型在“单点任务自动化”上的潜力。然而，芯片/IP 设计往往不是孤立的代码生成问题：在真实工程中，需求抽象、文档收敛、接口语义澄清、实现取舍、仿真联调、综合实现、板级验证与结果归因构成跨阶段闭环，且关键错误常源于隐含约束(如跨时钟域语义、握手时序闭环与资源、时序权衡)。因此，相比继续讨论模型在某一局部任务上的指标表现，一个更具工程与研究价值的问题是：在一个真实而复杂的芯片设计项目中，大模型究竟能够如何参与，又应如何与人工形成有效协同。

本文以一个面向云 FPGA Role-Shell 架构的分段时延探测工具为案例对象来回答上述问题。随着云端 FPGA 在 SmartNIC、异构加速与多租户计算环境中的部署，Role-Shell 隔离架构已成为常见组织方式[12]：平台侧 Shell 提供 PCIe、网络、DMA、时钟与管理等基础设施，用户侧 Role 承载可定制应用逻辑[13-17]。该架构在提升可运维性的同时，也引入了异构时钟域、跨时钟域(clock domain crossing, CDC)缓冲以及基于 valid/ready 的背压机制，使数据通路时延从相对稳定的传播/流水线延迟演变为受排队与拥塞传播影响的动态延迟。当系统出现抖动或尾延迟事件时，开发者往往难以判断瓶颈究竟位于 Role、Shell 还是两者边界的 CDC 结构内，形成难以观测与责任界定的“时延黑盒”。

传统测量手段在该场景下面临明显限制：依赖 IEEE 1588 等全局同步的方案在宏观节点间测量较成熟[18-19]，但在密集插桩与跨时钟域路径中会受到采样不确定性与高扇出时间戳分发网络的制约[20-25]；片上逻辑分析仪[26]等方式更适合离线短窗口波形抓取，难以长期在线捕获罕见尾事件；带内遥测[27-31]等在线方法虽可携带路径元数据，但在 FPGA 内部高性能数据通路中可能带来额外解析与修改开销，并且难以自然适配“跨时钟域边界+透明插桩”的需求。由此，如何在不依赖全局时钟同步的前提下，以低侵入方式实现 Role-Shell 数据通路的分段时延测量与尾延迟分布捕获，成为一个兼具工程价值与研究意义的问题。

针对上述问题，本文实现了分段时延探测架构 SegTracer，并以其研发过程作为人机协同案例进行总结。SegTracer 采用“探针极简、计算集中”的思路：在数据通路分段节点插入统一时间戳模块(unified timestamp module, UTM)，对专用探测包进行透明打点，并在返回路径执行“去程压栈、返程原位减法”的闭环计算，从而在同一局部时钟域内抵消初始相位差，实现免全局同步的分段测量；在链路端点部署包生成与监控(packet generation and monitoring, PGM)模块，完成探测包注入、远端反射、集中统计与分布分析，并对对数直方图记录宽量程时延分布以捕获尾延迟。基于 Xilinx VU37P 平台的实验表明，该工具能够在硬件直连场景下实现周期级分段时延剖析，并在 PCIe 软硬件协同场景中有效定位毫秒级尾部排队瓶颈。

更重要的是，本文关注的不仅是该工具的技术结果，也关注其形成过程中的大模型介入方式与可复用经验。本案例采用多种大模型与 Agent 工具贯

穿问题重构、文献候选生成与人工核验、设计文档收敛、SpinalHDL 代码框架生成、仿真脚本搭建与联调排查等环节；同时也暴露出大模型在概念类比与隐含约束理解上的典型失效模式，例如将独立探测包误解为带内遥测，误读返程原位减法语义，忽略背压导致的模块内等待时间等。本文据此总结了“短大纲先锁关键前提—文档与实现联动回写—以仿真与上板证据驱动纠偏”的协同策略，指出大模型更适合作为增强工程师能力的协同工具，而非替代最终责任的自动设计器。

本文主要贡献如下：

1）设计并实现了面向多 FPGA Role-Shell 系统的细粒度时延探测工具 SegTracer。该 IP 通过统一时间戳模块（UTM）与包生成监控模块（PGM）协同工作，采用独立探测包、去程压栈、返程原位减法和端点集中统计机制，实现了不依赖全局时钟同步的分段时延测量、集中统计与尾部分布分析。

2）以 SegTracer 的研发闭环为载体，系统呈现了一个覆盖问题重构、设计文档收敛、SpinalHDL 实现、仿真联调、板级验证与实验分析的人机协同芯片设计过程，展示了人工主导、大模型辅助、验证驱动的研发方式在真实 FPGA/IP 项目中的可行性。

3）总结了大模型在需求重构、材料整理、代码框架生成、脚本编写与调试支持中的有效介入点，以及其在领域概念理解、CDC 处理、握手时序、状态一致性与实验归因等环节中的典型偏差与纠偏路径，为后续复杂 FPGA/IP 项目中引入大模型提供了可复用经验。

1 动机与相关工作

1.1 Role-Shell 云 FPGA 架构与分段时延“黑盒”

云端 FPGA 在 SmartNIC、异构加速与多租户计算中的应用，通常采用 Role-Shell 隔离架构：平台侧 Shell 提供 PCIe、网络接口、DMA、时钟与管理等公共基础设施，用户侧 Role 承载可定制的应用逻辑。该组织方式有利于平台复用与部署运维，但也在数据通路上引入了清晰的结构边界与异构时钟域，从而增加了分段性能观测与责任归因的难度。

在典型实现中，Role 与 Shell 通过 AXI/AXI4-Stream 等接口互连[23-24]，并在边界处使用异步 FIFO、同步器等结构完成 CDC 交换[20-22] 因此，数据通路时延往往不再由固定逻辑级数决定，而会受到 valid/ready 背压传播和局部排队的显著影响：当下游阻塞或主机侧软件服务不及时，等待可能出现在 Role 内部、Shell 侧接口，或两者边界的跨域结构中。对开发者而言，虽然端到端时延升高可以被观测到，但各分段的具体贡献及责任来源并不清晰。

这一问题在云端场景中尤为突出，因为系统性能往往不仅取决于平均时延，更受高分位尾部事件影响[32-33]。因此，开发者真正关心的并不是“总时延是否升高”，而是“延迟发生在哪一段、由哪一侧引起、是否与跨域或背压有关”。这就要求测量方法不仅能给出端到端结果，还应支持分段归因、低侵入部署、长期在线运行和尾部分布刻画。

如图 1 所示，Role-Shell 数据路径中的典型观测边界与不确定性来源，包括跨时钟域、背压传播与排队点，正是本文关注的主要测量对象与技术难点。

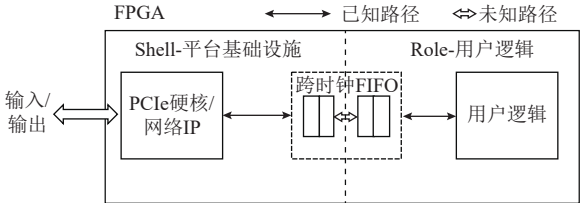


Fig. 1 Illustration of Role-Shell isolation architecture in FPGA

图 1 FPGA 内的 Role-Shell 隔离架构示意图

1.2 分段时延测量与剖析方法：分类与局限

围绕网络与片上系统的时延测量，既有工作形成了多条技术路线。为避免逐篇罗列，本文按“同步依赖、在线性、侵入性与分段归因能力”对相关方法进行分类，并总结其在 Role-Shell 场景下的主要局限。

1）基于全局时间基准的时间戳测量。以 IEEE 1588/PTP[18-19] 等为代表的全局同步方案，适合跨设备或宏观节点间的端到端测量与对时。在 FPGA 内部若直接依赖统一时间基准，需要面对 2 类额外代价：其一，时间戳语义必须跨越多个 CDC 边界，采样与同步链路会带来拍级不确定性；其二，在芯片内部大范围分发高位宽时间戳会形成高扇出时钟样布线网络[29-32]，增加布局布线压力并影响时序收敛。因此，此类方法更自然适配“宏观端到端测量”，但并不总能经济地支撑 Role-Shell 边界内的高密度分段插桩与长期在线观测。

2）片上调试/波形抓取与离线分析。 片上逻辑分析仪与调试基础设施（如 ILA）能以较高信号可见性捕获局部波形，是工程联调的常用手段[26]。但其本质上更偏离线：持续捕获窗口受片上存储资源限制，难以长期在线等待罕见尾事件；为获得可解释波形往往需要插入较多观测信号与触发逻辑，可能带

来额外资源占用与实现扰动。对于需要“分段统计+尾部分布”而非“短窗口波形”的场景，这一路线往往不足以作为常态化观测手段。

3)带内遥测与可编程数据面在线监测。带内遥测类方法通过在报文中携带或更新元数据，在线采集路径状态、队列信息与时间戳，已在网络测量与可编程交换等场景验证有效[27-31]。然而在高速硬件数据通路中，带内字段的解析、更新、重封装与状态写回会带来一定的资源与时序压力。本文关注 Role-Shell 云 FPGA 内部的分段测量与责任界定。若在普通业务流中嵌入元数据，可能会扰动原始的排队行为，从而改变长尾时延的真实分布。同时，传统带内遥测通常面向“网络路径级”而非“跨时钟域边界的细粒度分段”，在 CDC 与局部时钟差异并存的情况下并不总能直接套用。

综合上述分类可以看到：现有方案各有优势，但在本文关注的“Role-Shell 边界内区段级归因+长期在线+尾部分布刻画”组合目标下，常见不足集中在 3 点：1)对全局同步或统一时间基准的依赖较强；2)对高速数据通路的侵入性与实现压力较高；3)对长尾事件的长期捕获与宽量程分布分析支持不足。因此，本文案例对象需要一种更适配该场景的测量范式：既能在不显著扰动业务流的前提下在线运行，又能将“分段测量与尾部分布分析”落到可验证的工程实现上。相关工作与本文工作对比如表 1 所示。

1.3 大模型辅助芯片设计：能力边界与“闭环案例”缺口

近年来，围绕大模型辅助芯片设计，已在 RTL 生成评测[1-4]、规格到断言转换[5-6]、调试修复支持[7-8]以及 Agent 化流程探索[9-11]等方面取得了一定进展。这些工作说明，大模型能够在文档整理、代码框架生成、验证脚本编写和局部调试中提供有效帮助。

但对于真实 FPGA/IP 项目而言，挑战往往不在某一步能否生成，而在跨阶段闭环，即接口语义与隐含约束(例如 CDC 语义、背压闭环、时序与资源权衡)如果在早期被误解，会在后续实现与验证中被放大；此外，大模型可能产生概念类比错误或引用幻觉，导致看似合理但不满足工程前提的设计分支。由此，现有研究中相对不足的一环是：以一个具备明确工程约束、需要多轮文档—实现—验证迭代，并最终通过仿真与上板证据闭环确认的项目为对象，系统复盘“人工主导、大模型辅助”的协同方式，纠偏机制与可复用经验。本文不提出新的大模型结构、训练方法或 benchmark，也不以提升单一自动化任务指标为目标；本文定位为以真实云 FPGA/IP 工程为对象的 human-AI 协同案例研究。与侧重“单环节能力评测”或“流程框架原型”的既有研究相比，本文强调 3 点：

1)工程载体与闭环证据。给出可综合、可部署并通过仿真与上板实验验证的分段时延探测工具，实现从问题定义到结果归因的闭环。

2)协同边界与纠偏机制。总结大模型在文档、代码框架、验证与调试支持中的有效介入点，以及在隐含约束理解上的典型失效模式与纠偏方法。

3)面向复用的经验沉淀。将“人工主导—模型辅助—证据驱动验证”的协同范式具体化为可操作流程，为后续在复杂 FPGA/IP 项目中引入大模型提供参考。

Table 1 Comparison of Related Work and Our Work

表 1 相关工作与本文工作对比

工作	主要关注任务	成果形态	是否关注跨阶段工程闭环	是否包含实现/硬件验证	与本文差异
VerilogEval[1]	Verilog/RTL 生成评测	benchmark	否	否	聚焦代码生成能力评测
RTLLM[2] OpenLLM-RTL[3]	RTL 生成、数据与评测资源	benchmark/dataset	否	否	聚焦生成任务与资源建设
AssertLLM[5]	规格到断言生成	方法/验证流程	否	部分	聚焦验证语义转换
LLM4SecHW[7] MEIC[8]	调试、缺陷分析与修复支持	方法/数据集/调试流程	否	部分	聚焦调试辅助与错误分析
Spec2RTL-Agent[9]	规格到 RTL 的 Agent 流程	workflow/Agent framework	部分	部分	更偏流程自动化，不是完整工程案例复盘
本文	分段时延探测工具案例	human-AI case study+工程实现	是	是	关注真实 FPGA/IP 项目中的全过程协同与落地验证

2 人机协同设计过程：从问题重构到实现收敛

在复杂 FPGA/IP 的设计中，研发效率与逻辑正确性的平衡是核心挑战。如图 2 所示，本次 SegTracer 的开发经历了从传统范式到协同范式的认知转变。传统的手工 Verilog 开发(范式 *A*)由于需要人工处理繁琐的门级连线及编写复杂的 Testbench，往往导致

研发周期长达数周，且极易遗留深层逻辑缺陷。而尝试直接利用零样本大模型生成代码（范式 *B*）则表现出明显的“逻辑失真”与“编译死锁”现象，由于缺乏全局设计约束，模型在多轮提示词微调中往往因上下文丢失而无法收敛。

为此，本研究实践了一套“人工主导、大模型/Agent 辅助”的协同设计方法学（范式 *C*）。该方法学将研发过程重构为文档收敛、代码编译、仿真纠偏与特性进阶 4 个阶段，实现了仅用 3 天即完成从构思到板级验证的工程闭环。

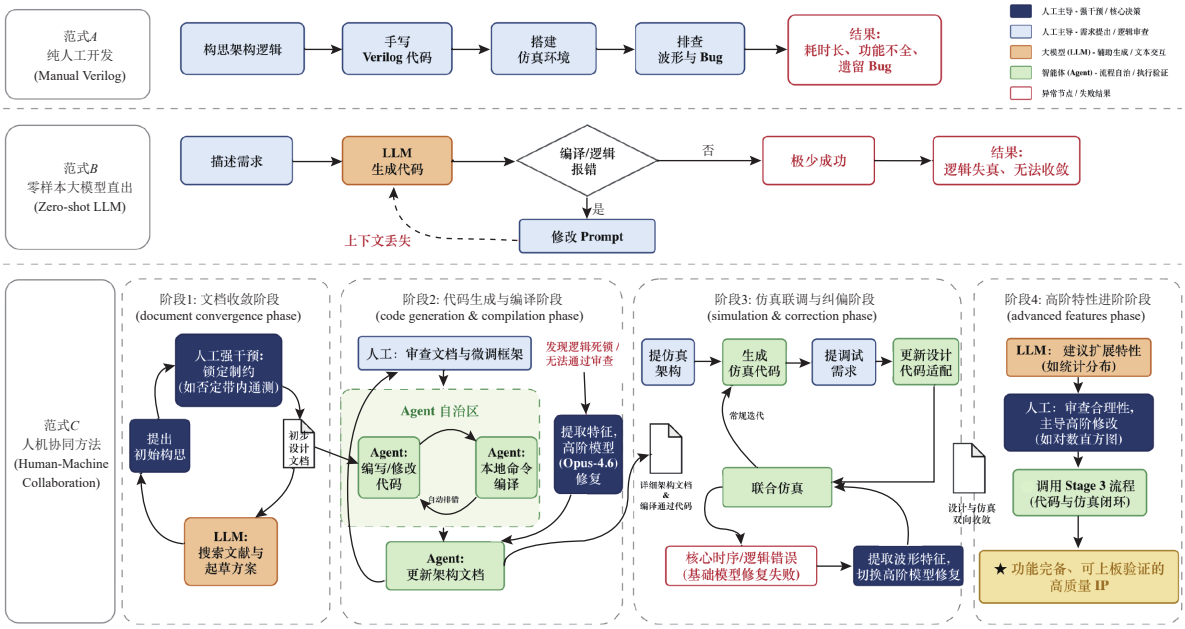


Fig. 2 Timeline of LLM-assisted collaborative design and staged output

图 2 大模型辅助的协同设计时间线与阶段性产出

2.1 阶段 1：文档收敛环与核心机制的强干预

在真实工程中，缺乏文档约束的代码生成必然导致逻辑发散。如图 2 的“文档收敛阶段”所示，该阶段的核心在于利用人工的领域经验对大模型的发散性进行收敛。

在 SegTracer 的设计初期，大模型主要负责搜索文献与起草初始方案。然而，大模型容易将局部关键词套入不完全匹配的现成范式。如表 2 第 3 行所示，模型曾将本系统误解为类似“带内遥测”的方案，试图将探测信息附着于普通业务流；在测量机制上，模型默认采用了常规软件思维的“端点统一计算”（表 2 第 4 行）。这些概念偏差若不加以阻断，将直接

Table 2 Key Stages, Typical Deviations, and Design Impacts in Human–AI Collaboration

表 2 人机协同中的关键阶段、典型偏差与设计影响

序号	阶段/关键问题	大模型的主要作用	典型偏差或局限	人工纠偏与验证方式	对最终系统的影响
①	问题重构	扩展术语、整理背景、重组问题表述	容易把局部关键词套入不完全匹配的现成范式	人工重新收敛研究边界，明确“Role-Shell 分段测量、低侵入、免全局同步”的目标	将原始工程直觉收敛为可实施、可写作的问题定义
②	文献候选生成与相关工作组织	提供候选关键词、候选文献方向和章节框架	可能出现错误题名、作者、年份或不相关引用	对候选文献逐条人工核验并重建引用链	形成可靠的相关工作与论文定位
③	探测包形式	组织协议描述、生成方案草稿	曾将系统误解为类似带内遥测，将探测信息附着于普通业务流	明确采用独立探测包，与普通业务流解耦	奠定了后续反射、闭环测量和低扰动实验的基础
④	闭环测量机制	提供时间槽位组织和流程草案	曾将机制误写为“去返程分别记录、端点统一计算”	明确固定为“去程压栈、返程原位减法”，并据此统一包头和槽位语义	决定了 SegTracer 的核心测量机制
⑤	UTM 组织与时延语义	参与模块拆分和接口草案生成	初始实现倾向于单打点，难以区分模块内部等待与跨模块传播	明确要求进入/离开双打点，并修正 tsIdx 返程语义	使系统具备分段责任界定能力
⑥	RTL 框架与跨文件实现	生成代码框架、接口模板、局部补全	容易局部合理但全局不一致，忽略 CDC、握手和边界条件	人工逐段检查，结合编译、仿真和接口约束持续修正	推动代码从“可见框架”收敛到可运行实现
⑦	回包反射与合法性判断	给出反射逻辑、状态判断和字段更新草案	初始代码对反射优先级、返回包合法性和异常路径考虑不足	人工增加反射优先级与 tsIdx 等状态一致性检查	提高了探测闭环稳定性与统计健壮性
⑧	长尾统计机制	提出记录分布信息的方向	初始方案偏向均值/极值或线性桶，难兼顾量程与资源	基于实验目标与资源约束收敛到对数直方图与子桶方案	使系统具备宽量程尾时延表征能力

导致系统偏离低侵入的设计初衷。具体示例如下：

示例 1. 纠正“带内遥测”误解，明确独立探测包

本文需要澄清一个关键前提：本系统发送的是完全独立的探测包，而不是嵌入到现有业务流中的元数据。

具体来说：

1）探测包由 PGM 模块主动生成，不依赖于任何业务流量。

2）探测包与业务包可以共享同一物理通道（AXI-Stream），但两者是独立的报文流。

3）探测包可以设置为低优先级而业务包为高优先级。这样当链路拥塞时，探测包会经历排队和反压，从而感知真实负载——这正是我们想要的测量效果。

请基于“独立探测包”这个前提，重新调整你的设计方案，不要再提及“嵌入业务流”或“带内遥测”。

在此环节，人工必须实施“强纠偏”：显式否定带内遥测，严格界定“独立探测包”前提，并锁定“去程压栈、返程原位减法”的物理闭环机制。这一过程确立了详尽的初步设计文档，为后续的 Agent 代码生成提供了稳固的逻辑基石。

示例 2. 纠正返程计算逻辑，明确原位减法

关于 UTM 的时间戳记录和返程计算逻辑，请仔细理解以下要点：

1）每个 UTM 在去程记录 2 个时间戳：进入时刻和离开时刻，分别占用 2 个连续槽位。进入时记录当前时间到 slot[tsIdx]，tsIdx++；离开时记录当前时间到 slot[tsIdx]，tsIdx++。经过 N 个 UTM 后 tsIdx=2N。

2）返程时，UTM 需要在 2 个时刻分别执行原位减法：进入 UTM 时，用当前本地时间减去 slot[tsIdx−1]（去程离开该 UTM 的时刻），结果覆盖 slot[tsIdx−1]，tsIdx 不变；离开 UTM 时，用当前本地时间减去 slot[tsIdx−2]（去程进入该 UTM 的时刻），结果覆盖 slot[tsIdx−2]，然后 tsIdx=2。

3）经过所有 UTM 后 tsIdx 归零，可用此条件判断回包完整性。请不要再简化为“只在离开时做一次减法”或“返回端点后统一计算”。

2.2 阶段 2：代码生成与编译的 Agent 化闭环

当核心机制在文档层面收敛后，开发进入如图 2 所示的“代码生成与编译阶段”。在本阶段，人工的角色从“代码编写者”转变为“架构微调者与审查者”。

通过将设计文档输入 Agent 工具，系统能够快速生成基于 SpinalHDL 的代码大框架。相比于传统的 Verilog 手写过程，Agent 的优势在于其能够调用本地命令排查语法错误，实现底层的自动迭代。在此过程中，人工无需干预底层连线与基础语法，而是将精力集中于审查 Agent 生成的代码架构文档是否与初始设计文档的预期保持一致。这一“文档驱动框架、Agent 解决语法、人工审查逻辑”的分工，有效提升了 RTL 框架的构建效率。

2.3 阶段 3：仿真驱动的逻辑纠偏与模型路由决策

代码通过编译并不等同于逻辑正确，仿真联调是验证系统时序的关键。如图 2 的“仿真联调与逻辑纠偏阶段”所示，人工首先提出仿真架构并增设调试寄存器，随后由 Agent 迭代生成 Testbench。

在联调过程中，大模型在处理复杂硬件时空关系上的局限性充分暴露。当涉及 CDC 握手与返程原位减法等硬核逻辑时，初始模型（GLM-4.7）生成的代码在波形层面出现核心逻辑错误，且多次尝试修复失败（表 2 第 6、7 行）。此时，协同方法学进入关键的“模型路由决策”节点：人工通过阅读仿真波形提取核心错误特征，辅以详尽的时序说明，主动切换至推理能力更强的高阶模型（如 Claude Opus-4.6）进行针对性修复。这种由人工主导的“验证发现问题—提取波形特征—调度高阶模型”的纠偏机制，是突破深层硬件逻辑瓶颈的必要手段。

2.4 阶段 4：高级特性进阶与方法学沉淀

在系统逻辑基本跑通后，研发进入高阶特性完善阶段。如图 2 最后一步所示，为了捕获长尾分布，大模型建议增加直方图统计（表 2 第 8 行）。然而，人工结合 FPGA 的真实资源约束与时延量程评估后，主导将其优化为“对数直方图”，并再次利用高阶模型完成代码落地，最终实现了高质量 IP 的上板收敛。

综合上述 4 个阶段的闭环实践，针对复杂数字 IP 的研发，本文沉淀出以下人机协同设计方法论。

1）短文档先行与边界约束。摒弃“直接对话生成代码”的习惯，优先通过 1~2 页的设计大纲锁定核心机制与接口约束，并明确“非什么（如非带内遥测）”，以阻断模型的路径依赖。

2）多轮拆解与状态机演进。将复杂的时序逻辑拆解为单一维度的子步骤，并在文档与实现之间进行多轮同步回写，避免在错误的底层代码上进行局部补丁叠加。

3）能力路由与验证驱动。明确不同 Agent/大模型的能力边界。常规模型用于框架生成与文档梳理，遇到深层时序与资源权衡问题时，必须依赖人工提取仿真波形特征，并调度高阶模型进行精准攻坚。

3 案例对象：SegTracer 的系统架构与实现

第 2 节复盘了 SegTracer 的形成过程。本节在此基础上给出经多轮纠偏与验证后收敛形成的系统架构、探测协议与核心模块实现。换言之，以下内容并非独立于协同过程之外的预设方案，而是人机协同设计闭环最终沉淀下来的工程结果。

3.1 设计目标与总体架构

SegTracer 采用“探针极简、计算集中”的总体思路，由 2 类功能互补的模块构成：一类是透明插入数据通路分段节点的 UTM，用于在本地时钟域内完成透明打点与返程原位减法；另一类是部署于链路端点的 PGM，负责探测包注入、远端反射、统计维护和延迟分布分析。

该架构的核心思想不是向全局分发统一时间戳，而是让探测包在去程沿路径逐级记录局部时间信息，在返程再次经过相同节点时直接执行本地减法，从而得到各分段区段对应的往返周期差值。UTM 负责“局部观测”，PGM 负责“端点控制与集中分析”，这样的职责划分既降低了分段节点插桩的实现压力，也使统计与实验控制能够集中在链路端点完成。

如图 3 所示，从系统功能上看，SegTracer 关注的并非一般意义上的端到端延迟测量，而是面向 Role-Shell 数据通路内部的分段时延剖析。因此，系统设计强调 2 点：一是通过轻量级探针实现对关键节点的透明插桩，尽量降低对原始数据路径的扰动；二是通过端点集中统计与分布记录，使系统不仅能够给出分段延迟值，而且能够支持尾部分布刻画和责任边界分析。

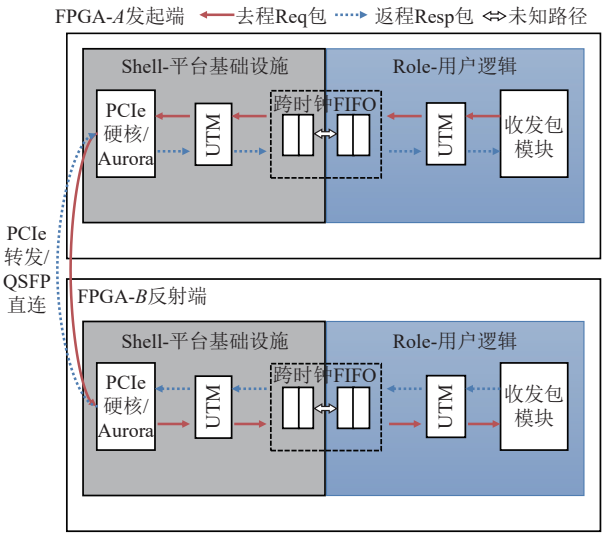


Fig. 3 Physical deployment of SegTracer system

图 3 SegTracer 系统物理部署

3.2 探测协议与原位闭环测量机制

为了兼容 AXI4-Stream 等标准流接口，并尽量避免对正常业务流造成扰动，SegTracer 采用专用探测包而非直接修改普通业务报文。如图 4 所示为探测包头部示意，其预留了固定长度的自定义遥测区域，用于保存探测标识、源/目的 ID、动态指针 tsIdx 以及若干时间戳槽位。

系统最终确立了“去程压栈、返程原位减法”的闭环测量机制。该机制并非大模型直接生成的默认方案，而是经由人工对 CDC 物理约束与测量时序边界进行显式定义后，最终收敛的软硬件协同产物。去程时，探测包在经过某个 UTM 节点时，由本地自由运行计数器对进入和离开该节点的时刻分别采样，

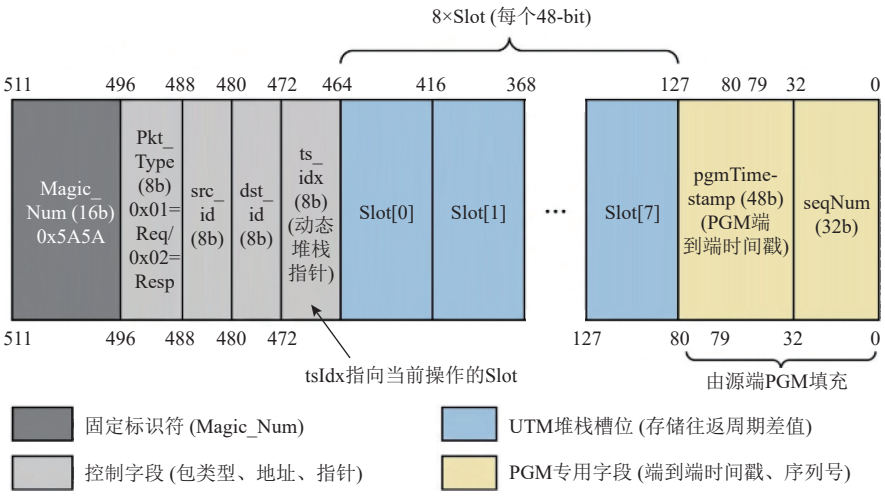


Fig. 4 Custom telemetry header field layout of the probe packet

图 4 探测报文自定义遥测头部字段布局

并按 *tsIdx* 指示的位置压入包头槽位；探测包到达远端后，由 PGM 硬件仲裁器以最高优先级将其切换为响应包并原路反射。

返程时，响应包再次经过相同 UTM 节点，UTM 读取此前在去程写入的本地计数值，与当前计数器值直接在本地时钟域内相减，并将结果覆盖回原槽位。这样，局部初始相位偏移可在同一时钟域内被抵消，从而得到该分段节点对应的周期差值。在实现上，每个被观测子段对应固定数量的时间槽位；去程阶段 UTM 分别对进入与离开事件写入对应槽位，返程阶段再按相同槽位索引完成本地减法并原位回写，从而保持包头语义与 RTL 行为一致。

核心测量关系可表示为：

$$\Delta C_i = C_i^{\mathrm{ret}} - C_i^{\mathrm{fwd}} = \frac{t_i^{\mathrm{ret}} - t_i^{\mathrm{fwd}}}{P_i}, \tag{1}$$

其中 C_i^{fwd} 与 C_i^{ret} 分别表示第 i 个节点在去程与返程阶段采样得到的本地计数值，P_i 为该节点的本地时钟周期。由于 2 次采样均在同一局部时钟域内完成，相减后不再依赖全局同步时钟，因此能够支持跨域路径中的分段测量。

需要指出的是，独立探测包及“去程压栈、返程原位减法”的协议语义，并非设计初期即已完全明确，而是在前述协同过程中经多轮纠偏后逐步固定下来的关键前提。正是这些前提的明确，才使探测包头布局、tsIdx 槽位语义以及返程阶段的本地闭环计算能够在实现层面保持一致。

3.3 核心模块实现与工程落地

在具体实现上，UTM 被设计为透明插入数据通路中的轻量级探针，其主要功能是在尽量固定的流水线级数下完成报文识别、槽位寻址、本地计数采样和返程减法操作。由于探测包需要反映模块内部排队和反压带来的等待时间，UTM 在节点的进入与离开阶段均执行一次打点，从而不仅能够观测传播延迟，也能够反映模块内部的动态等待时间。

PGM 位于链路端点，负责主动发起探测、对远端请求进行快速反射，并对回收包中的各段时间信息进行集中统计。最终实现中，发起与反射功能被合并在同一模块中，以便每个端点既可作为发起方，也可作为反射方。为降低反射引入的附加误差，PGM 在仲裁上赋予反射路径更高优先级。除记录最小值、最大值、累计值和平均统计量外，系统还实现了对数直方图统计逻辑，从而能够在有限资源下覆盖从纳秒级到毫秒级的宽量程时延分布。

从实现层面看，统一 UTM、进入/离开双打点以及 PGM 的发起—反射合并等细节，也是在协同设计与验证迭代中逐步收敛形成的工程取舍。这些取舍既体现了前述设计文档与代码实现之间的联动修正，也直接影响了系统在后续实验中的测量精度、可部署性与统计能力。

从工程实现看，SegTracer 具有较好的可落地性。如表 3 所示，单个 UTM 探针资源开销较低，可在多个分段节点处进行密集部署；集中式 PGM 虽承担更多控制与统计功能，但整体资源占用仍处于可接受范围内，并可支持在线统计与分布记录。由于使用了寄存器语义来实现相应的记录，可以看出 BRAM 的使用为 0，这与预期也是一致的。综上，SegTracer 不仅可作为实验性原型，也具备作为云 FPGA 场景诊断工具的应用潜力。

Table 3 Post-Synthesis Resource Utilization of Hardware Modules

表 3 综合后硬件模块资源占用

模块	LUT	FF	BRAM	工作频率/MHz
UTM（Role）	2 287	2 350	0	50
UTM（Shell）	2 288	2 362	0	250
PGM	19 754	26 361	0	50
总资源（芯片）	1 303 680	2 607 360	2 016	
UTM 占比/%	0.18	0.09	0	
PGM 占比/%	1.52	1.01	0	

SegTracer 的时延测量基于各 UTM 所在时钟域的本地自由运行计数器。探测包去程与返程经过同一 UTM 时，分别记录该计数器的值，并在返程时原位相减。由于 2 次采样均在同一局部时钟域内完成，相减结果直接给出经过的时钟周期数，不受跨时钟域相位差或全局同步误差的影响。

因此，单个 UTM 的测量精度等于该时钟域的时钟周期。在本实验平台中，Role 域工作频率为 50 MHz，对应周期 20 ns；Shell 域工作频率为 250 MHz，对应周期 4 ns。由于测量基于本地自由运行计数器且去、返程在同一时钟域相减，单个 UTM 的测量精度即等于该时钟域的时钟周期，无需额外校准。

与基于排队论或机器学习的时延估计模型不同，SegTracer 提供的是实测值，不依赖任何先验参数或假设。这一特性使其天然适合作为时延模型的训练标签或验证基准，例如研究者可将 SegTracer 在 FPGA 原型上测得的逐段时延分布作为 ground truth，用于校准其分析模型或机器学习模型的预测误差。

为评估 SegTracer 在高频及密集插桩场景下的时

序收敛能力，对 UTM 模块在 Xilinx VU37P(−2 速度等级)上进行了独立综合。目标频率 400 MHz 下最差建立时间裕量 WNS(worst negative slack)为正，关键路径(位选择操作)组合逻辑延迟约 0.4 ns，理论支持 500 MHz 以上。

密集插桩场景下，各 UTM 串行插入数据通路但相互独立，彼此之间不形成组合逻辑级联，因此时序收敛压力主要源于总资源利用率而非单一路径深度。基于 Xilinx UltraScale+架构的布线资源与设计规模关系保守估算：当插桩数量不大于 200 时，250 MHz 工作频率下时序可保证收敛，且留有超过 20% 的裕量(WNS > 0.5 ns)；若插桩数量增至 400，建议将工作频率适当降至 200 MHz 或采用更精细的物理布局约束(如按区域分组)。此外，高负载反压不会导致时序额外恶化(UTM 在非探测包周期透明直通)。

4 实验评估与案例验证

本节通过仿真与上板实验验证 SegTracer 的技术有效性，重点考察其在不同场景下的分段时延测量能力、责任界定能力以及长尾分布捕获能力。对于本文而言，这些实验结果不仅用于评估系统性能，也用于检验前文协同过程中形成的关键设计决策是否真正转化为可测量、可解释的硬件行为。

4.1 实验平台与测试场景

为验证该案例产物的技术有效性，本文在 Xilinx Virtex UltraScale+ VU37P 平台上完成了仿真与上板测试。如图 5 所示为实验平台，该平台由主机与 2 块 FPGA 加速卡组成，2 块 FPGA 既可通过 QSFP 直连形成纯硬件往返路径，也可通过 PCIe 与主机交互构成软硬件协同测试环境。系统内部划分为 Role 域与 Shell 域，两者通过跨时钟域 FIFO 相连，从而构成与云 FPGA Role-Shell 架构一致的测试对象。

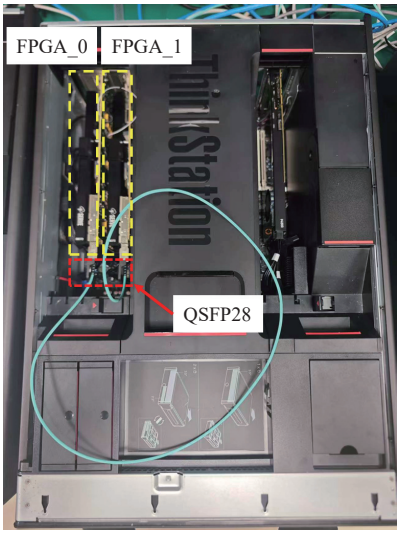


Fig. 5 Prototype of experimental platform

图 5 实验平台原型

为覆盖不同系统条件，实验设置了 3 类典型场景。第 1 类为纯硬件 QSFP 直连场景，用于建立低时延基准并验证分段测量机制的基本正确性；第 2 类为 PCIe 用户态驱动场景，用于观察轮询式软硬件协同引入的额外开销；第 3 类为 PCIe 内核态驱动场景，用于分析中断与调度参与后可能出现的排队长尾问题。上述 3 类场景从纯硬件路径到深度软硬件耦合路径逐级展开，使系统能够在不同复杂度条件下被验证。

4.2 分段时延剖析与应用案例：从测量到优化

1)测量。首先在 QSFP 纯硬件直连场景下验证 SegTracer 的分段测量能力。如图 6 所示，系统将端到端往返路径解构为多个分段区段：链路往返延迟为 771.5 ns，两侧 FPGA 上跨时钟域往返延迟均为 300 ns 左右。系统能够稳定观测到 Role 域与 Shell 域内部的固定流水线开销以及跨时钟域 FIFO 的握手与排队代价，说明原位减法机制能够有效消除局部相位差影响，支撑周期级(纳秒量级)分段测量。这一结果验证了本文核心技术路线的硬件可实现性。

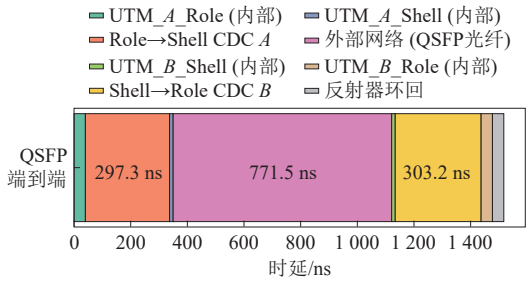


Fig. 6 Fine-grained segment-level latency in the QSFP hardware direct-connect scenario

图 6 QSFP 硬件直连场景下的精细分段时延

2)责任界定。进一步地，在不同物理接口和驱动模式下，SegTracer 将总时延拆解为“FPGA 内部排队与处理时间”和“外部(主机侧软件/驱动)耗时”2 个分段，从而支持责任边界界定。如图 7 所示，QSFP 直连场景下 2 个分段分别为 0.7 μs 和 0.8 μs，时延高度稳定；PCIe 用户态驱动(DPDK 轮询模式)下分别为 11.2 μs 和 11.6 μs，软硬件开销相对均衡；而 PCIe 内核态驱动下，FPGA 侧平均排队与处理时间高达 4 340.7 μs，外部耗时仅 165.5 μs。该对比清晰表明：内核态模式的长尾瓶颈主要积压在 FPGA 内部排队，其根本原因是内核态中断与调度开销导致探测包在

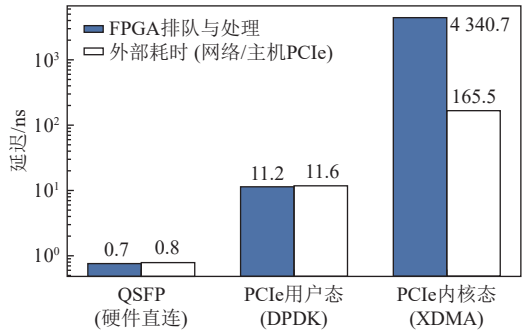


Fig. 7 Latency overhead distribution and bottleneck comparison under different interface/driver modes

图 7 不同接口/驱动模式下的时延开销分布与瓶颈对比

FPGA 发送队列中大量积压, 而非 FPGA 内部流水线或 CDC 边界问题。

3)优化。基于上述归因, 采取 2 项优化措施: 将驱动从内核态切换为 DPDK 用户态轮询模式, 并适当增大 DMA 批处理深度。

4)验证。优化后, SegTracer 再次测量端到端时延分布: 最小延迟从 1 861.2 μs 降至 19 μs, 尾延迟(99.9 分位)从 11 962.2 μs(>11.9 ms)降至 29 μs, 降低幅度超过 99.7%。该案例完整演示了“测量→归因→优化→验证”的闭环过程, 表明 SegTracer 不仅能够精准定位时延瓶颈的责任分段, 还能直接指导系统级优化决策并量化优化效果。

4.3 长尾分布捕获与瓶颈定位

除了平均值与极值外, 时延分布本身也是判断系统健康状况和分析长尾风险的重要依据。围绕这一点, SegTracer 在 PGM 中集成了硬件对数直方图模块, 以较低成本记录宽量程时延分布, 如图 8 所示。实验中, 不同传输方式呈现出明显不同的分布特征: QSFP 直连场景下, 时延高度集中, 反映出纯硬件路径的确定性; PCIe 用户态场景下, 分布开始出现可控拖尾, 表明高负载下存在一定程度的队列积压; PCIe 内核态场景下, 端到端时延分布显著展宽, 并出现毫秒级极端长尾, 说明系统瓶颈已不再是单纯的链路传播, 而是更深层的软件驱动与硬件队列耦合问题。

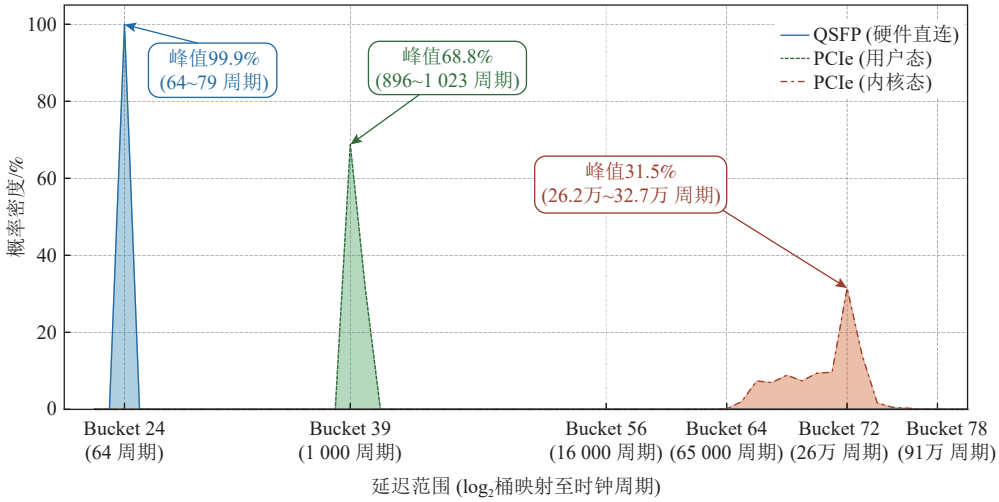


Fig. 8 Log-scale histograms of end-to-end latency probability density distributions in three representative scenarios

图 8 3 种典型场景下的端到端时延概率密度分布对数直方图

与第 3 节的分段剖析能力相结合, 这些分布结果使 SegTracer 不仅能够提供某一时刻或某一组探测包的延迟值, 还能够持续刻画系统在不同运行条件下的尾部分布特征。对于多 FPGA 场景而言, 这一点尤为重要, 因为真实系统体验往往并不由平均值决定, 而更多受到高分位尾事件影响。从这一意义上看, 对数直方图统计机制使 SegTracer 从单纯的分段测量工具进一步扩展为具备分布分析与尾部诊断能力的在线观测工具。

分段时延剖析在复杂 IP 研发中承担着“设计反馈”的关键角色。以 PCIe 场景下的 12 ms 瓶颈诊断为例, 若缺乏细粒度观测, 设计者常误认为瓶颈在于硬件 DMA 处理能力; 而 SegTracer 的剖析数据揭示了长尾主要源于内核态驱动的上下文切换, 直接驱动了团队采用 DPDK 轮询模式的决策, 从而实现了软硬件协同的最优收敛。此外, SegTracer 可作为性能约束验证工具, 在多 FPGA 互连或 Role-Shell 架构中, 通过对关键 CDC 节点的透明监控, 确保每个子模块满足预设的时延预算(latency budget), 显著提升了系统级调试与性能调优的效率。

4.4 轻量化特性与对比评估

为了客观评估 SegTracer 的工程开销与测量能力，本文将其与 Xilinx 集成逻辑分析工具（ILA）、典型带内遥测方案（如 PINT、F3）进行了分析与综合对比，ILA 实例化配置为 4 个探针，每个探针包含 ready、valid、tkeep（64-bit）、tuser、tlast、tdata（512-bit），总探针位宽与 SegTracer 的单个 UTM 处理的探测包位宽（512-bit）及接口数量（4 个 AXI-Stream 通道）基本相当，综合对比如表 4 所示。

Table 4 Comprehensive Evaluation Comparison of Different Latency Measurement and Profiling Schemes

表 4 不同时延测量与剖析方案的综合评估对比

评估维度	SegTracer	ILA	带内遥测
资源消耗（LUT/BRAM）	极低（0.18% LUT，0 BRAM）	较高（0.29% LUT，2.46% BRAM）	中高（需字段解析与重封装逻辑）
报文与带宽开销	极低（少量低优先级独立探测包）	无（纯带外数据采集）	极高（每个业务报文均需附带时间戳）
数据通路时序压力	极小（旁路插入，不改变原始流）	较小（仅引入只读探测扇出）	大（串联于关键路径，恶化时序裕量）
测量盲区	无（支持跨时钟域与罕见长尾感知）	大（触发窗口短，难以持续捕获长尾）	较高（修改原始报文易扰动系统拥塞态）

如表 4 所示，SegTracer 在资源占用、带宽开销、时序压力和测量盲区等方面均优于 ILA 和典型带内遥测方案。

5 讨论：大模型辅助 FPGA/IP 设计的边界、经验与局限

基于第 2 节所复盘的协同过程以及第 3, 4 节所对应的工程实现与验证结果，本节将进一步总结大模型在复杂 FPGA/IP 设计任务中的适用边界、协同经验与主要风险。

5.1 大模型辅助芯片/IP 设计的适用边界

结合本文案例，大模型在芯片/IP 设计中更适合承担的是信息收敛、表达重构与局部实现加速类任务，而不是脱离上下文独立完成完整设计闭环的任务。首先，在需求尚不稳定、术语表达不统一、历史材料较为分散的早期阶段，大模型能够较高效地完成背景扩展、术语统一与问题表述重组，帮助研究者将分散的工程直觉收敛为较明确的研究问题。其次，在核心机制与接口边界已经由人工明确的条件下，大模型对设计文档组织、RTL/脚本框架生成、测试框架搭建以及实验文字整理具有明显效率优势。再次，大模型在自然语言、表格、伪代码、接口说明与实现框架之间的跨表示转换能力，也有助于降低工程材料在不同表示层之间切换的成本。

但与此同时，本文案例也表明，大模型并不适合直接承担那些机制定义、约束确认与正确性责任高度耦合的核心决策任务。例如，CDC 路径中的采样语义、返程阶段时间戳槽位的真实含义，模块内部等待时间是否应纳入测量，统计机制在资源与量程之间如何折中等问题，都不能仅依赖语言模式匹配来解决，而必须依赖设计者对对象系统的持续建模和实验验证。由此，本文案例给出的边界可以概括为：当任务主要是整理、转写、补齐、重构、展开时，大模型通常能够带来明显帮助；当任务进入定义约束、确立机制、承担正确性责任与最终取舍阶段时，人工仍然是不可替代的主体。

为量化大模型辅助带来的效率提升，我们将本次开发与早先一次纯人工 Verilog 原型开发进行对比。该早期原型同样以实现 Role-Shell 分段时延测量为目标，手工编写 Verilog，耗时约 2 周。但仅完成了基本往返测量，未实现 UTM 双打点、原位减法和分布统计等完整功能。并且仍存在多处未修复 Bug，主要瓶颈集中在 AXI-Lite 配置接口实现与 Testbench 编写调试。本次采用大模型辅助 SpinalHDL 实现，从想法确认到稳定通过仿真与上板验证约 3 天时间，核心开发周期缩短约 70%。需要指出，本次开发同时切换到了 SpinalHDL 高级语言，其强类型和面向对象特性本身也提升了开发效率，大模型辅助与高级语言带来的收益难以完全分离。但可以确定的是，相比早先基于 Verilog 的纯人工原型，人机协同与高级语言框架的组合显著加速了设计收敛。主要效率瓶颈从“人工编写与调试复杂 Testbench”转移为“对大模型输出进行语义纠偏”，典型关键机制（如探测包形式、返程减法逻辑）需要 2~5 轮纠偏对话。这一量化对比直观展示了人机协同范式在 FPGA/IP 开发中的实际价值。

5.2 从案例中总结的人机协同经验

本文案例表明，与其将大模型视为“自动设计器”，不如将其理解为一种高吞吐、强交互但不承担最终责任的协作工具。在这一前提下，有 3 点经验较为稳定。

1）应先显式化关键约束，再调用生成能力。本案例中，大模型发挥作用的前提并不是提示词尽可能开放，而恰恰是人工先明确若干不可动摇的前提：测量对象是 Role-Shell 架构中的分段路径，系统不能依赖全局同步时钟，需要支持分段时延而非仅端到

端测量，并最终形成可综合、可上板验证的工程产物。在这些边界被明确后，大模型对文档收敛、接口组织和局部代码生成的质量显著提高。

2)复杂硬件设计更适合多轮澄清，而不是一次性求全。本文案例中，探测包形式、返程减法语义、tsIdx 的作用、UTM 的双打点机制以及统计模块的实现方式，均经历了“提出—生成—发现问题—纠偏—再生成”的反复过程。如果试图在一次提示中同时完成需求理解、架构设定、代码实现和正确性说明，往往更容易在隐含前提上埋下系统性错误。相反，将问题拆解为多个较短链路或在每一步设置显式检查点，更符合真实工程设计的收敛方式。

3)验证应当成为协同流程的中心，而不是末端补救。本文案例中，真正保证设计收敛的并不是生成能力本身，而是贯穿全过程的验证约束：文档是否与实现一致、仿真现象是否符合机制预期、资源与时序是否可接受、板级实验是否支持性能结论。尤其在波形分析、CDC 行为理解、结果归因等环节，大模型只有在人工先抽取关键现象后，才能继续参与推理与排查。因此，对芯片/IP 设计而言，大模型的有效使用方式并不是“先生成、后验证”，而是让验证标准在设计文档、实现与调试阶段持续前置。

5.3 风险、局限与外部效度

尽管本文案例总体呈现出较积极的结果，但也暴露出若干需要警惕的风险。首先是术语表面一致而语义实际漂移的问题。大模型往往能够复用提示词中的关键词并生成结构完整的描述，但这并不意味着其对工程含义的理解完全正确；本文中“带内遥测”“返程减法”“分段节点”“透明打点”等概念，都曾在看似合理的文本中发生语义偏移。其次是默认补全带来的隐性错误。当约束表达不充分时，大模型会自动填补其认为“通常合理”的实现细节，而这些默认答案并不一定适用于当前硬件系统。再次是局部正确掩盖全局不一致的问题，即单段文字、单个模块或局部代码看似成立，但文档、实现、验证与实验解释之间并不一致。最后，还需要警惕对流畅表达的过度信任，即在工程研究中，措辞连贯和逻辑完整不能替代波形、代码、资源报告与实验数据构成的证据链。

除一般性风险外，本文自身也存在若干局限。首先，本文属于单案例研究，虽覆盖了需求重构、架构设计、实现、验证与实验分析等多个环节，但仍主要对应一种具有明确工程背景和实验目标的 FPGA/IP 任务，其结论尚不能直接外推到所有类型的芯片设计问题。其次，本文明显依赖人工先验与领域经验，即研究者并非完全从零开始，而是已有相关问题背景、早期原型与工程判断能力，因此本文更准确地说明的是“在专家主导前提下，大模型可以显著提高工程推进效率”，而非“大模型可以替代设计者独立完成专业设计”。再次，本文观察到的协同效果还受到具体模型能力、上下文长度、Agent 框架与开发环境的影响，不同团队和不同工具条件下的复现效果可能存在差异。最后，本文尚未对效率收益进行严格量化，因此其定位更适合作为一项面向真实工程闭环的案例研究，而非严格意义上的生产率评测工作。

6 总结与展望

本文以云 FPGA 分段时延探测工具——SegTracer 为工程载体，系统复盘了从需求重构到板级验证的人机协同设计闭环。基于“探针极简、原位闭环计算”的设计，SegTracer 在 Xilinx VU37P 平台上实现了免全局同步的周期级分段时延剖析，并精准定位了 PCIe 软硬件协同场景下 12 ms 的长尾排队瓶颈，验证了该架构的工程有效性。

借由该硬件实现过程，本文验证了“人工主导、大模型辅助、证据驱动验证”的协同设计范式。案例表明，大模型在问题重构、文档生成与代码框架搭建方面具备显著效率价值；但在处理 CDC 语义、底层握手时序及系统级实验归因等硬核约束时，仍不可替代人工的架构决断。在复杂 FPGA/IP 设计中，大模型目前更适合作为加速工程迭代的协作工具，而非脱离验证约束的自动设计主体。

未来工作将重点围绕 3 个方向展开：

1)验证协同范式的普适性。在更多类型的复杂数字 IP 任务中开展研究，特别是在高性能计算、异构互连等对时序和并发要求严苛的领域，检验“人工主导、模型辅助”模式在不同工程背景下的迁移能力。

2)完善效能的量化评估体系。引入提示轮数、代码有效采纳率、调试回退深度及 Agent 调用成本等细粒度指标，建立针对 FPGA 研发全周期的效率评估基准，为硬件工程引入大模型后的投入产出比提供客观依据。

3)探索智能驱动的闭环自动修复机制。针对模型在波形分析上的短板，未来可构建验证驱动的调试 Agent，通过脚本调用仿真 API 提取关键节点的紧凑结构化波形，并将其转译为文本语义，使大模型从

“单向生成器”演进为能与底层 EDA 环境双向交互的推理中心，实现从缺陷归因到代码自动回写测试的全链路闭环。

作者贡献声明：张聪武负责方案总体设计与调研，完成实验、论文修订等工作；王亚洲、王攀宇负责实验环境搭建与论文撰写；陈明宇、包云岗、张科对研究提出指导意见并修改论文。

参 考 文 献

[1] Liu Mingjie, Pinckney N, Khailany B, et al. Invited Paper: VerilogEval: Evaluating large language models for Verilog code generation[C]//Proc of the 42nd IEEE/ACM Int Conf on Computer Aided Design. New York: ACM, 2023: 1−8

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

[3] Liu Shang, Lu Yao, Fang Wenji, et al. OpenLLM-RTL: Open dataset and benchmark for LLM-aided design RTL generation[C]//Proc of the 43rd IEEE/ACM Int Conf on Computer Aided Design. New York: ACM, 2024: 60: 1−60: 9

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

[5] Yan Zhiyuan, Fang Wenji, Li Mengming, et al. AssertLLM: Generating hardware verification assertions from design specifications via multi-LLMs[C]//Proc of Asia and South Pacific Design Automation Conf. New York: ACM, 2025: 614−621

[6] Wang Yonghao, Zhou Jiaxin, Lyu Hongqin, et al. DeepAssert: An LLM-aided verification framework with fine-grained assertion generation for modules with extracted module specifications[EB/OL]. (2025-09-18)[2026-03-09]. https://arxiv.org/abs/2509.14668

[7] Fu Weimin, Yang Kaichen, Dutta R G, et al. LLM4SecHW: Leveraging domain-specific large language model for hardware debugging[C]//Proc of IEEE Asian Hardware Oriented Security and Trust Symp. Piscataway, NJ: IEEE, 2023: 1−6

[8] Xu Ke, Sun Jialin, Hu Yuchen, et al. MEIC: Re-thinking RTL debug automation using LLMs[C]//Proc of the 43rd IEEE/ACM Int Conf on Computer Aided Design. New York: ACM, 2024: 100: 1−100: 9

[9] Yu Zhongzhi, Liu Mingjie, Zimmer M, et al. Spec2RTL-Agent: Automated hardware code generation from complex specifications using LLM agent systems[EB/OL]. (2025-06-26)[2026-03-09]. https://arxiv.org/abs/2506.13905

[10] Xiong Chenwei, Liu Cheng, Li Huawei, et al. HLSPilot: LLM-based high-level synthesis[C]//Proc of the 43rd IEEE/ACM Int Conf on Computer Aided Design. New York: ACM, 2024: 226: 1−226: 9

[11] Jiang Zesong, Zhang Qing, Liu Cheng, et al. IICPilot: An intelligent integrated circuit backend design framework using open EDA[EB/OL]. (2024-07-17)[2026-03-09]. https://arxiv.org/abs/2407.12576

[12] Barroso L A, Hölzle U, Ranganathan P. The Datacenter as a Computer: Designing Warehouse-Scale Machines[M]. 3rd ed. Cham: Springer, 2019

[13] Firestone D, Putnam A, Mundkur S, et al. Azure accelerated networking: SmartNICs in the public cloud[C]//Proc of the 15th USENIX Symp on Networked Systems Design and Implementation. Berkeley, CA: USENIX Association, 2018: 51−66

[14] Putnam A, Caulfield A M, Chung E S, et al. A reconfigurable fabric for accelerating large-scale datacenter services[C]//Proc of the 41st Annual Int Symp on Computer Architecture. Piscataway, NJ: IEEE, 2014: 13−24

[15] Caulfield A M, Chung E S, Putnam A, et al. Configurable Clouds[J]. IEEE Micro, 2017, 37(3): 52−61

[16] Ahmed M K, Panoff M K, Mbongue J M, et al. Multi-tenant cloud FPGA: A survey on security, trust, and privacy[J]. ACM Transactions on Reconfigurable Technology and Systems, 2025, 18(2): 23: 1−23: 44

[17] Mbongue J M, Kwadjo D T, Shuping A, et al. Deploying multi-tenant FPGAs within Linux-based cloud infrastructure[J]. ACM Transactions on Reconfigurable Technology and Systems, 2022, 15(2): 19: 1−19: 31

[18] IEEE. IEEE Std 1588−2019: IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems[S]. New York: IEEE, 2019

[19] Patoli A A, Fortino G. FPGA-based system implementation of IEEE 1588 precision time protocol: A review[J]. IEEE Sensors Journal, 2025, 25(11): 18624−18642

[20] Cummings C E. Clock Domain Crossing (CDC) Design & Verification Techniques Using SystemVerilog[C]//Proc of Synopsys User Group Boston Conf (SNUG Boston). Boston: Synopsys, 2008. https://scholar.google.com/scholar?q=C.+E.+Cummings%2C+%22Clock+Domain+Crossing+%28CDC%29+Design+and+Verification+Techniques+using+SystemVerilog%2C%22+SNUG+2008

[21] Ginosar R. Metastability and Synchronizers: A tutorial[J]. IEEE Design & Test of Computers, 2011, 28(5): 23−35

[22] Kleeman L, Cantoni A. Metastable behavior in digital systems[J]. IEEE Design & Test of Computers, 1987, 4(6): 4−19

[23] Arm. AMBA specifications[EB/OL].[2026-03-09]. https://www.arm.com/architecture/system-architectures/amba/amba-specifications

[24] Arm. AMBA AXI-Stream Protocol Specification[S]. Cambridge, UK: Arm, 2021

[25] AMD Xilinx. Vivado Design Suite User Guide: Implementation (UG904)[R]. San Jose, CA: AMD Xilinx, 2021

[26] AMD Xilinx. Vivado Design Suite User Guide: Programming and Debugging (UG908)[R]. San Jose, CA: AMD Xilinx, 2021

[27] Feng Weiqi, Gao Jiaqi, Chen Xiaoqi, et al. F3: Fast and flexible network telemetry with an FPGA coprocessor[J]. Proc ACM on Networking, 2024, 2(CoNEXT4): 25: 1−25: 22

[28] Ben Basat R, Ramanathan S, Li Y, et al. PINT: Probabilistic in-band network telemetry[C]//Proc of ACM SIGCOMM 2020 Conf. New York: ACM, 2020: 662−680

[29] Papadopoulos K, et al. Deterministic and probabilistic P4-enabled lightweight in-band network telemetry[EB/OL]. (2024-04-09)[2026-03-09]. https://arxiv.org/abs/2404.06582

[30] P4 Language Consortium. P4_16 Language Specification[S].

Stanford (CA):P4 Language Consortium, 2017

[31] Wang Han, Soulé R, Dang H T, et al. P4FPGA: A rapid prototyping framework for P4[C]//Proc of Symp on SDN Research. New York: ACM, 2017: 122−135

[32] Barroso L A, Marty M, Patterson D A, et al. Attack of the killer microseconds[J]. Communications of the ACM, 2017, 60(4): 48−54

[33] Dean J, Barroso L A. The tail at scale[J]. Communications of the ACM, 2013, 56(2): 74−80



Zhang Congwu, born in 1995. PhD candidate. His main research interest includes large-scale SoC pre-silicon prototyping validation.

张聪武, 1995 年生。博士研究生。主要研究方向为大规模 SoC 硅前原型验证。



Wang Yazhou, born in 1993. Master. His main research interests include FPGA prototyping verification and accelerated simulation.

王亚洲, 1993 年生。硕士。主要研究方向为 FPGA 原型验证、加速仿真。



Wang Panyu, born in 2003. Master candidate. His main research interests include computer architecture and FPGA prototyping verification.

王攀宇, 2003 年生。硕士研究生。主要研究方向为计算机体系结构、FPGA 原型验证。



Chen Mingyu, born in 1972. PhD, professor. His main research interests include memory architecture, hardware security method of processor, hardware/software optimization on network protocol of data-center.

陈明宇, 1972 年生。博士，研究员。主要研究方向为计算机访存体系结构、处理器硬件安全控制机制、数据中心服务器网络协议栈软硬件优化技术。



Bao Yungang, born in 1980. PhD, professor. Member of CCF. His main research interests include datacenter architecture, agile design methodology of processor chips, and ecosystem of open-source processor chips.

包云岗, 1980 年生。博士、研究员。CCF 会员。主要研究方向为数据中心体系结构、处理器芯片敏捷设计方法论、开源处理器芯片生态。



Zhang Ke, born in 1982. PhD, professor. His main research interests include computer architecture, heterogenous acceleration, and FPGA cloud.

张　科, 1982 年生。博士、正高级工程师。主要研究方向为计算机体系结构、异构加速、FPGA 云。