查看所有版本

Versal 自适应 SoC 系统集成和确认方法指南

UG1388 (v2023.2) 2023 年 11 月 15 日

目录

第 1 章

简介

关于 Versal 自适应 SoC 设计方法论

AMD Versal™ 自适应 SoC 设计方法论是一整套旨在帮助简化当今 Versal 器件设计进程的最佳实践。鉴于这些设计的规模与复杂性，因此必须通过执行特定步骤与设计任务才能确保设计每个阶段都能成功完成。建议您遵循这些步骤和最佳实践进行操作，这将有助于您以尽可能最快且最高效的方式实现期望的设计目标。

按设计进程浏览内容

AMD 自适应计算文档按一组标准设计进程进行组织，以便帮助您查找当前开发任务相关的内容。您可以在设计中心页面上访问 AMD Versal™ 自适应 SoC 设计流程。您还可以使用设计流程助手来更深入了解设计流程，并找到特定于预期设计需求的内容。

- 系统集成与确认：集成和确认系统功能性能，包括时序收敛、资源使用情况和功耗收敛。

如需获取更多方法论相关信息，请参阅以下文档：

- 系统和解决方案规划：确认系统级别的组件、性能、I/O 和数据传输要求。包括解决方案到 PS、PL 和 AI 引擎的应用映射。请参阅《Versal 自适应 SoC 设计指南》(UG1273) 和《Versal 自适应 SoC 系统和解决方案规划方法指南》(UG1504)。
- 嵌入式软件开发：基于硬件平台来创建软件平台，并使用嵌入式 CPU 开发应用代码。还涵盖 XRT 和计算图 API。请访问此链接以参阅《AI 引擎工具和流程用户指南》(UG1076) 中的相应内容。
- AI 引擎开发：创建 AI 引擎计算图及内核、库用法、仿真调试与剖析以及算法开发。还包含 PL 与 AI 引擎内核的集成。请参阅《AI 引擎工具和流程用户指南》(UG1076) 和《AI 引擎内核与计算图编程指南》(UG1079)。
- 硬件、IP 和平台开发：为硬件平台创建 PL IP 块、创建 PL 内核、功能仿真以及评估 AMD Vivado™ 时序收敛、资源使用情况和功耗收敛。还涉及为系统集成开发硬件平台。请参阅《Versal 自适应 SoC 硬件、IP 和平台开发方法指南》(UG1387)。
- 开发板系统设计：通过板级原理图和开发板布局来设计 PCB。还包含功耗、散热以及信号完整性注意事项。请参阅《Versal 自适应 SoC 开发板系统设计方法指南》(UG1506)。

关于本指南

本指南包含对应如下主题的高层次信息、设计指南和设计决策利弊取舍：

- 第 2 章：仿真流程：提供有关 Versal 自适应 SoC 设计所使用的仿真流程的高层次信息。
- 第 3 章：设计收敛：提供有助于满足时序和功耗要求的建议。
- 第 4 章：系统性能收敛：提供有助于分析和提升设计性能的建议。
- 第 5 章：配置与调试：提供设计调试配置及各种调试方法的概述。
- 第 6 章：确认：提供确认各种 Versal 器件资源以及确认整个系统的方法。

第 2 章

仿真流程

为了应对仿真范围、仿真抽象和仿真目的等方面的不同需求，AMD 为 AMD Versal™ 器件设计的各组件提供了专用的流程，包括 AI 引擎、PS 和 PL。此外，AMD 还支持对由 PL、PS 和（可选）AI 引擎组件组成的完整系统进行协同仿真。各设计团队必须先在功能级别确认功能，然后再将其集成到部分系统应用或整个系统中。

下表显示了每个 Versal 器件块可用的仿真模型。

表 1：**Versal 器件块支持的仿真模型**

块	周期精确	性能
PS	QEMU（仅限功能运行）	QEMU（仅限功能运行） CIPS 验证 IP (VIP)
NoC	行为级 SystemVerilog（周期近似）	SystemC
DDR 存储器控制器	行为级 SystemVerilog	SystemC
HBM 控制器	行为级 SystemVerilog	行为级 SystemVerilog
基于 PL 的软核存储器控制器	行为级 SystemVerilog	行为级 SystemVerilog
CPM	行为级 SecureIP	行为级 SecureIP
GT	行为级 SecureIP	文件 I/O（仅适用于 Vitis 软件平台）
基于 GT 的 IP	行为级 SecureIP	AXI Verification IP 文件 I/O（仅适用于 Vitis 软件平台）
基于 HLS 的 IP	RTL	RTL
其他 IP	因 IP 而异	因 IP 而异
PL	行为级 Verilog VHDL SystemVerilog	行为级 Verilog VHDL SystemVerilog
AI 引擎	SystemC（周期近似）	SystemC

以下章节提供了有关每个仿真流程的范围和目的的详细信息。

注释：这些仿真流程中大部分均可用于传统设计流程和基于平台的设计流程。但仅限在基于平台的设计流程中才能对完整系统执行协同仿真。

相关信息

分析仿真中的系统性能

逻辑仿真

逻辑仿真可测试以 PL 互连结构为目标的硬件设计，它属于传统 FPGA 仿真流程。此仿真的范围可调整，从单个硬件块到整个硬件平台都适用。仿真的模型通常为 RTL，从而保证抽象层的周期精确性。仿真速度与测试设计大小成比例，设计越大，仿真耗时越长。要提升仿真性能，可将部分 Versal 自适应 SoC IP 块替换为 SystemC 传输事务级模型，此类模型仿真速度更快，但无法再保障周期精确性。此仿真的目的是先验证并调试详细的硬件功能，然后在器件上实现设计。

逻辑仿真可通过 Vivado Design Suite 获取。如需了解更多信息，请参阅《Vivado Design Suite 用户指南：逻辑仿真》(UG900)。

注释：在传统设计流程和基于平台的设计流程中均可执行逻辑仿真。

使用 SystemC 模型执行逻辑仿真

SystemC 属于 C++ 语言库，支持硬件建模。此库可提供结构元素（例如，模块、端口和接口）以及数据类型。除周期精确的仿真模型外，AMD 还为部分 Versal 自适应 SoC 基础架构提供了能够快速精确完成传输事务的 SystemC 仿真模型，以供其在 Vitis 硬件仿真流程中使用。SystemC 模型仿真速度比 RTL 模型更快，这样有助于缩短总体仿真时间。

总之，SystemC 模型可用于性能分析、架构探索、DMA 同步以及地址追踪生成和性能建模。但是，如果精确性和调试更为重要，那么 AMD 建议使用 RTL 模型，比如用于解决 DMA 传输事务或时序相关问题。

HLS 仿真

HLS 仿真专用于测试 HLS 代码，它是 HLS 开发流程中不可或缺的一环。此仿真的范围是单个 HLS 内核。受支持的抽象层分为以下 2 种：未定时和 RTL（周期精确）。这两个抽象层分别被称为 C 语言仿真 (Csim) 和协同仿真 (Cosim)。在 Csim 流程中，应使用 C 语言仿真通过测试激励文件确认要综合的函数。C 语言测试激励文件包含一个 `main()` 顶层函数，用于调用函数以供 Vitis HLS 工程进行综合。在 Cosim 流程中，HLS 编译器生成的 RTL 代码输出将与 Csim 结果的输出进行自动比对。Cosim 流程的目的是验证 RTL 的功能正确性，以及确认独立环境内的性能，它与其他函数无交互。

有关 HLS 仿真的信息，可通过 Vitis 统一软件平台来获取。欲知详情，请访问此链接以参阅 Vitis HLS 用户指南 (UG1399) 中的相应内容。

注释：在传统设计流程和基于平台的设计流程中均可执行 HLS 仿真。

AI 引擎仿真

您可以使用以下仿真器流程来仿真 AI 引擎计算图和设计。这些流程可在 AI 引擎计算图与内核开发阶段，在仿真速度与准确性之间做出权衡：

- 未定时 (x86simulator)：验证代码功能是否正确，并确认独立环境内的性能，它与其他函数无交互。此仿真流程有助于验证 AI 引擎计算图与内核的功能准确性并提供最快的仿真运行速度。
- 周期近似 (aiesimulator)：更精确地计算核矢量负载和存储器访问，您可以使用它来估算 AI 引擎计算图性能并提高功耗估算的准确性。此外，该仿真流程将 GMIO 和 PLIO 接口建模到 AI 引擎计算图并对 NoC、PS 和 PL 进行建模，以提供周期近似结果。

有两种方法可用于向 AI 引擎仿真提供激励。要执行快速功能验证，可将 `input_plio` 语句添加到计算图文件中，以提供输入文件列表。或者，可使用流量生成器来对动态环境进行更准确的建模。如需了解更多信息，请参阅 AXIS 外部流量生成器功能特性教程。

如需了解有关这些仿真流程的更多信息，请参阅以下资源：

- 请访问此链接以参阅《AI 引擎工具和流程用户指南》(UG1076) 中的相应内容
- Vitis 教程：AI 引擎开发调试演示示例
- Vitis 教程：AI 引擎 Versal 集成

嵌入式软件仿真

嵌入式软件仿真可测试仅以 PS 为目标的软件设计。它以 Quick Emulator (QEMU) 为基础，后者可对 Versal 自适应 SoC 内集成的双核 Arm® Cortex®-A72 的行为进行仿真。此仿真支持对平台操作系统进行快速紧凑的功能确认。此流程包含 SystemC 传输事务级系统模型，支持尽早进行系统浏览和验证。

有关嵌入式软件仿真的信息，可通过 Vitis 统一软件平台来获取。如需了解更多信息，请访问此链接以参阅《Versal 自适应 SoC 系统软件开发者指南》(UG1304) 中的相应内容，并请参阅赛灵思维基：QEMU 用户文档。

注释：在传统设计流程和基于平台的设计流程中均可执行嵌入式软件仿真。

硬件仿真

硬件仿真用于对整个 Versal 自适应 SoC 系统（包含 AI 引擎、PS 和 PL）进行仿真。通过使用 Vitis 软件平台即可集成以全部 3 个计算域为目标的块和功能。Vitis 连接器会自动生成完整的协同仿真设置，包括 RTL、SystemC 和 QEMU 模型：

- PS 上运行的嵌入式软件代码是使用 QEMU 来仿真的。
- AI 引擎上运行的代码是使用 SystemC AI 引擎仿真器来仿真的。
- 用户 PL 内核作为 RTL 代码来进行仿真。
- 根据可用或所选模型的类型，硬件平台中的 IP 块作为 RTL 或 SystemC TLM 来进行仿真。

因此，Vitis 硬件仿真的抽象层非常接近但未完全达成周期精确。Versal 自适应 SoC 平台的部分细节是使用 TLM 模型来抽象化的，目的是为了保证仿真速度。

Vitis 硬件仿真的范围同样定义了其目的。硬件仿真允许您对整个设计进行仿真，并在实现前测试 PL、PS 和 AI 引擎之间的交互。由于硬件仿真可通过调试查看应用的方方面面，因此在此环境中对复杂问题进行调试比在真实硬件中进行调试更简单。

有关硬件仿真的信息，可通过 Vitis 统一软件平台来获取。如需了解更多信息，请参阅《Vitis 统一软件平台文档：应用加速开发》(UG1393) 和《AI 引擎工具和流程用户指南》(UG1076)。

您可使用 AXI Traffic Generator 来代替使用基于文件的输入和输出。AXI Traffic Generator 与静态且受限的基于文件的输入和输出相反，通过动态方式来生产和使用数据。如需了解更多信息，请参阅 AXIS 外部流量生成器功能特性教程。

使用 AXI Traffic Generator 的优势如下：

- 允许您复用 AI 引擎仿真期间所设置的仿真集
- 允许您利用流量生成器抽取系统中的某些块，这样能够更快执行仿真

注释：仅限在基于平台的设计流程中才能执行硬件仿真。

NoC 仿真

NoC 仿真支持随 SystemVerilog 或 SystemC 中的行为模型一起提供。与 SystemVerilog 模型相比，SystemC 模型的仿真速度要快得多，但周期近似且精度较低。

注释：您可使用“IP Project Settings”（IP 工程设置）来选择自己首选的仿真模型。使用“rtl”设置即可选择 SystemVerilog，使用“tlm”设置即可选择 SystemC。这些设置适用于整个工程。

虽然可以同时使用 SystemC 和 SystemVerilog 模型来验证功能，但建议使用 SystemVerilog 模型进行性能分析。使用 SystemVerilog 模型的性能分析误差在硬件的 ±5% 范围内。

您可以使用 Vivado 工具中的仿真器或使用 Vitis 工具提供的硬件仿真流程来仿真 NoC。

重要提示！如需了解有关 NoC 仿真设置和性能调优的更多信息，请访问此链接和此链接，以参阅《Versal Adaptive SoC Programmable Network on Chip and Integrated Memory Controller LogiCORE IP 产品指南》(PG313) 中的相应内容。如需了解有关 AXI Traffic Generator 的更多信息，请参阅 GitHub 仓库中提供的 Versal 片上网络性能 AXI Traffic Generator 教程。

CIPS Verification IP

Control, Interfaces, and Processing System (CIPS) Verification Intellectual Property (VIP) 支持对 Versal 自适应 SoC 应用执行功能仿真。其目标是通过模拟处理器系统 (PS)-PL 接口和 PS 逻辑的 OCM 存储器来支持对可编程逻辑 (PL) 进行功能验证。此 VIP 是作为 SystemVerilog 模块封装包来交付的。VIP 操作通过使用一连串 SystemVerilog 任务来加以控制。如需了解更多信息，请参阅《Versal 自适应 SoC CIPS Verification IP 数据手册》(DS996)。

第 3 章

设计收敛

设计收敛包括满足所有系统性能、时序和功耗要求，并成功确认硬件中的功能。在设计收敛阶段，您可开始通过实现工具运行设计，因此首先需要考量的就是时序和功耗注意事项。

在此设计收敛阶段，估算设计使用率、时序和功耗可以得到准确性更高的结果。这样即可为您提供机会来重新确认时序和功耗目标是可达成的。为确认设计能够满足其要求，AMD 建议制定时序基线和功耗基线。时序基线侧重于在定义准确的时序约束之后，评估时序路径。功耗基线则需要为 AMD Vivado™ 提供正确的翻转信息，以便确定准确的动态功耗信息。

鉴于功耗要求分析与时序要求分析需结合使用，只要其中任一方出现重大偏差，那么为了解决其问题所采取的措施就会对另一方产生重大影响。例如：

- 为了满足诸如按比例缩减功能之类的功耗预算，就可能需要采取非常极端措施。由于器件拥塞减少，因而将显著简化时序收敛。
- 添加逻辑以减少开关时，可能涉及较为极端的措施。这可能增加时序收敛的难度，尤其是在裸片的拥塞区域内。

虽然有许多节省功耗的项目并不会影响时序收敛，但其他项目则可能导致时序收敛难度增大。运用必要的节省功耗技巧有助于您了解时序收敛任务的真正量级。

当您基于基线开始迭代后，应在改善时序时复检功耗数值。这样可以确保您了解哪些更改导致倒退。通常，建议您尽早开启整套功耗节省功能，然后对导致出现时序问题的个别项进行缩减，这样有助于达成适当的平衡，从而满足设计收敛目标。

在设计收敛实现阶段尽早联动开展功耗分析和时序分析将能够节省工程设计时间，实现更准确的工程规划。此外，这样即可留出更多时间用于探索各种工程设计解决方案，不至于在设计周期后期才发现更合适的解决方案。

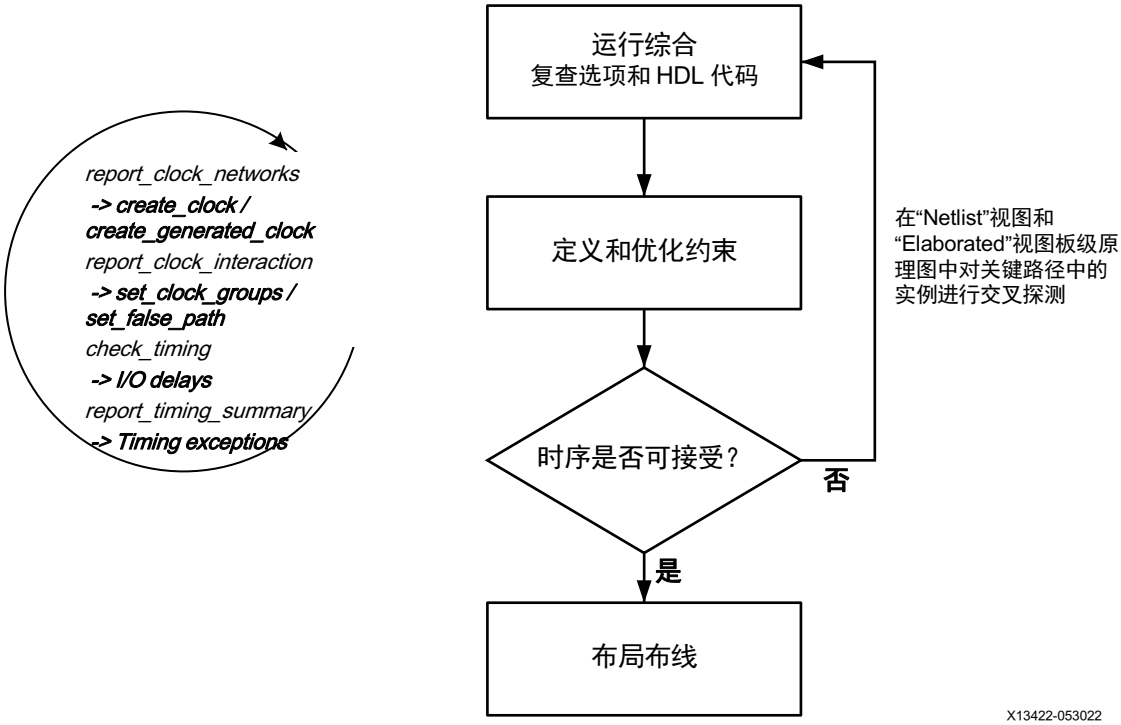
提示：如需了解有关本章中提及的各项报告的更多信息，请参阅《Vivado Design Suite 用户指南：设计分析与收敛技巧》(UG906)。

重要提示！应用性能需求通常是通过下列方式来满足的：在关键的块（基于 PL 的块或硬核 IP 块）之间创建适当的连接架构、为控制块和计算块提供正确的吞吐量和时延预算并为块和存储器之间的数据移动设置正确的服务质量 (QoS) 约束。通常，满足性能目标并不需要尝试为所有 PL 块都实现尽可能最佳的时钟频率，这样可能因增加逻辑区域而导致功耗增加，且无法获得相关的性能增益。

时序收敛

时序收敛是指设计满足所有的时序要求。针对综合采用正确的 HDL 和约束条件就能更易于实现时序收敛。通过选择更合适的 HDL、约束和综合选项，经过多个综合阶段进行迭代同样至关重要，如下图所示。

图 1：**实现快速收敛的设计方法论**



要成功完成时序收敛，请遵循下列常规准则进行操作：

- 最初不能满足时序要求时，请在整个流程中评估时序。
- 集中精力解决每个时钟的最差负时序裕量 (WNS) 是改进总体时序负裕量 (TNS) 的主要途径。
- 复查严重的最差保持时序裕量 (WHS) 违例 (<-1 ns) 以识别遗漏或不恰当的约束。
- 重新评估设计选择、约束和目标架构之间的利弊取舍。
- 了解如何使用工具选项和赛灵思设计约束 (XDC)。
- 请注意，满足时序要求后，工具就不会再尝试进一步改进时序（额外裕度）。

以下章节提供的建议可用于使用方法论设计规则检查 (DRC) 根据基线设定来复查时序约束的完整性和正确性、识别时序违例的根本原因并使用常用技巧来解决违例。

注释：综合后的时序结果使用估算的信号线延迟，而非实际的布线延迟。要获取最终时序结果，请运行实现，然后检查“Report Timing Summary”（时序汇总报告）。

理解时序收敛标准

要实现时序收敛，首先要编写有效的约束，以演示设计在硬件中的运作方式。按照下文各章节所述，检查“Timing Summary”（时序汇总）报告。

检查有效约束

复查“Timing Summary”（时序汇总）报告的“Check Timing”（检查时序）部分以快速评估时序约束覆盖范围，包括：

- 所有有源的时钟管脚都有对应的时钟定义。
- 所有有源的路径端点都具有与已定义的时钟（建立/保持/恢复/移除）相关的要求。
- 所有有源的输入端口都具有输入延迟约束。
- 所有有源的输出端口都具有输出延迟约束。
- 已正确指定时序例外。

> **注意！** 在约束中过度使用通配符可能导致实际约束与期望的约束不同。使用 `report_exceptions` 命令可识别时序例外冲突并复查网表对象以及每个例外所涵盖的时序路径。

除 `check_timing` 外，“Methodology”（方法论）报告（TIMING 和 XDC 检查）会标记可能导致时序分析不准确的时序约束以及可能出现的硬件故障。您必须仔细复查并解决所有报告的问题。

注释： 设定设计基线时，必须使用所有 AMD IP 约束。请勿指定用户 I/O 约束，并忽略因缺少用户 I/O 约束而导致 `check_timing` 和 `report_methodology` 生成的违例。

相关信息

设计基线设定

检查正时序裕量

以下时序指标用于反映设计时序得分。为了满足时序要求，数字必须为正。

- 建立/恢复（最大延迟分析）：WNS > 0 ns 且 TNS = 0 ns
- 保持/移除（最小延迟分析）：WHS > 0 ns 且 THS = 0 ns
- 脉冲宽度：WPWS > 0 ns 且 TPWS = 0 ns

理解时序报告

“Timing Summary”（时序汇总）报告可提供设计的时序特性与所提供的约束对比的高层次信息。验收时请复查“Timing Summary”（时序汇总）报告提供的数据：

- 总体负时序裕量 (TNS)：针对整个设计或针对特定时钟域中的每个端点的建立/恢复违例的总和。最差建立/恢复时序裕量为最差负时序裕量 (WNS)。
- 总体保持时序裕量 (THS)：针对整个设计或针对特定时钟域中的每个端点的保持/移除违例的总和。最差保持/移除时序裕量为最差保持时序裕量 (WHS)。
- 总体脉冲宽度时序裕量 (TPWS)：针对整个设计或针对特定时钟域中的每个时钟管脚执行以下检查时的违例总和：
  - 最小低脉冲宽度
  - 最小高脉冲宽度
  - 最小周期
  - 最大周期
  - 最大偏差（相同叶节点单元的 2 个时钟管脚之间）
- 最差脉冲宽度时序裕量 (WPWS)：对任何给定时钟管脚执行的所有脉冲宽度、周期或偏差检查的最差时序裕量。

总时序裕量（TNS、THS 或者 TPWS）仅可反映设计中的违例情况。当所有时序检查均符合要求时，总时序裕量为零 (null)。

时序路径报告可提供详细计算方法，用于在任意逻辑路径上检查任何时序的时序裕量。在完全约束的设计中，每条路径必须满足 1 个或多个需求，以确保相关逻辑能够可靠运作。

WNS、TNS、WHS 和 THS 涵盖的主要检查源于时序单元的功能需求：

- 建立时间：在到达下一个有效时钟沿之前，新数据必须保持稳定可用状态才可供安全捕获的时间量。
- 保持要求：在有效时钟沿到来后，数据必须保持稳定状态以避免捕获无用值的时间量。
- 恢复时间：异步复位信号切换到无效状态的时间与下一个有效时钟沿之间所需的最短时间。
- 移除时间：在有效时钟沿到来后，异步复位信号可安全切换到其不活动状态之前所需的最短时间。

有 1 个简单示例，即连接到相同时钟信号线的 2 个触发器之间的路径。

在时钟信号线上定义时序时钟之后，时序分析就会在最保守但又合理的工作条件下对目标触发器的数据管脚执行建立时间和保持时间检查。当建立和保持时序裕量均为正值后，即可在源触发器到目标触发器之间执行安全的数据传输。

如需了解有关时序分析的更多信息，请访问此链接以参阅《Vivado Design Suite 用户指南：设计分析与收敛技巧》(UG906) 中的相应内容。

检查设计是否正确约束

在查看时序结果是否存在违例之前，应确保设计中的每个同步端点都已正确约束。

运行 `check_timing` 可识别未约束的路径。此命令可单独运行，但也可随 `report_timing_summary` 一起运行。此外，`report_timing_summary` 还包含“Unconstrained Paths”（未约束的路径）部分，其中已定义的源或目标时序时钟会列出不含时序要求的 N 条逻辑路径。N 由 `-max_path` 选项控制。

对设计实现完全约束后，请运行 `report_methodology` 命令并复查 TIMING 和 XDC 检查，以识别非最优化约束，此类约束可能导致时序分析不完全准确，并导致硬件中时序裕度 (timing margin) 发生变化。要识别并纠正不现实的目标时钟频率或者建立路径要求，请使用 `report_qor_assessment` 命令。

重要提示！要解决缺失的约束或不完整的约束，请使用“Timing Constraints”Wizard（时序约束向导），或者请参阅《Vivado Design Suite 用户指南：使用约束》(UG903)。

修复 check_timing 标记的问题

`check_timing` Tcl 命令报告称时序定义中缺失部分内容或者存在错误。复查并修复由 `check_timing` 标记的问题时，首先请关注最重要的检查。以下检查按重要性从高到低排列。

无时钟和未约束的内部端点

您可据此判定设计中的内部路径是否已受到完整约束。作为“Static Timing Analysis”（静态时序分析）验收质量审查的一部分，必须确保未约束的内部端点数为 0。

未约束的内部端点数为 0 表示所有内部路径均已实现约束，可供进行时序分析。但这并不能保证约束值正确。

生成时钟

生成时钟是设计的正常组成部分。但是，如果衍生出生成时钟的主时钟与生成时钟不属于同一时钟树，就会导致严重问题。时序引擎无法正确计算生成时钟树延迟。这会导致时序裕量计算出错。最糟糕的情况是，根据报告，设计可满足时序要求，但在硬件中无法正常运行。

环路和锁存器环

由于时序引擎会导致时序环路中断，因此良好的设计都不含任何组合环路。在时序分析期间不会报告中断的路径，在实现期间也不会评估此类路径。这可能导致硬件中出现不正确的行为，即使可满足总体时序要求也是如此。

无输入/输出延迟和部分输入/输出延迟

所有 I/O 端口必须正确实现约束。

建议：首先确认基线约束，然后再使用 I/O 时序完成约束。

多个时钟

通常可接受存在多个时钟。AMD 建议您确保这些时钟按期望方式在相同时钟树上传输。您还必须验证这些时钟之间的路径要求，避免由此产生的要求过于苛刻而导致无法在硬件中正常运行设计。

如果发生这种情况，必须在这些时钟之间的这些路径上使用 `set_clock_groups` 或 `set_false_path`。使用时序例外时，必须始终确保这些例外仅影响目标路径。

修复 report_methodology 标记的问题

`report_methodology` 命令可报告其他约束及时序分析问题，在运行布局和布线工具前后您必须仔细审查这些问题。本节将介绍需要检查的主要 XDC 和 TIMING 类别，及其对时序收敛和硬件稳定性的相对影响。首先，您必须专注于解决会影响时序收敛的检查。

如需了解有关其中部分检查的更多信息，请访问此链接以参阅《Vivado Design Suite 用户指南：设计分析与收敛技巧》(UG906) 中的相应内容。另请参阅巧用方法论报告系列博客，以了解有关如何借助 `report_methodology` 来解决问题和节省时间的更多信息。

重要提示！在时序汇总文本报告中也同样包含方法论违例的汇总信息，以便用户查看，因为解决这些问题对于时序正确完成验收至关重要。

方法 DRC 及其对时序收敛的影响

下表中所示 DRC 着重介绍了因增加实现工具压力而导致时序收敛无法实现或出现不一致的设计和时序约束组合。这些 DRC 通常与如下因素有关：缺少时钟域交汇 (CDC) 约束、不适当的时钟树或因逻辑复制导致时序例外覆盖范围不一致。这些问题必须以最高的优先级来处理。

重要提示！请仔细验证严重性为“Critical Warning”（严重警告）的时序检查。

如需了解有关时序方法检查的更多信息，请访问此链接以参阅《Vivado Design Suite 用户指南：设计分析与收敛技巧》(UG906) 中的相应内容。

表 2：**时序收敛方法 DRC**

检查	严重性	描述
TIMING-6	Critical Warning	在相关时钟之间不存在公用时钟
TIMING-7	Critical Warning	在相关时钟之间不存在公共节点
TIMING-8	Critical Warning	在相关时钟之间不存在公共周期
TIMING-14	Critical Warning	时钟树上的 LUT

表 2：**时序收敛方法 DRC** (续)

检查	严重性	描述
TIMING-15	Warning	时钟间路径上的严重保持时间违例
TIMING-16	Warning	严重建立时间违例
TIMING-30	Warning	生成时钟所选主源管脚欠佳
TIMING-31	Critical Warning	相移时钟间的多周期路径不合适
TIMING-32、TIMING-33、TIMING-34、TIMING-37、TIMING-38 和 TIMING-39	Warning	非建议的总线偏差约束
TIMING-36	Critical Warning	针对生成时钟缺少主时钟沿传输
TIMING-42	Warning	时钟传输受阻于路径分段
TIMING-44 TIMING-45	Warning	用户时钟内部和时钟间的不确定性不合理
TIMING-48	Advisory	在锁存器输入上存在“仅最大延迟数据路径”约束
TIMING-49	Critical Warning	来自并行 BUFGCE_DIV 的不安全的使能或复位拓扑结构
TIMING-50	Warning	同级锁存器之间的路径要求不现实
TIMING-56	Warning	缺少按逻辑或物理方式排除的时钟组约束
XDCB-3	Warning	在同一条 `set_clock_groups` 命令中，多个组中提及同一个时钟
XDCH-1	Warning	多周期路径约束中缺少保持时间选项
XDCV-1	Warning	由于缺少复制中使用的原始对象，导致约束覆盖范围不完整
XDCV-2	Warning	由于缺少复制对象，导致约束覆盖范围不完整

方法 DRC 及其对验收质量和硬件稳定性的影响

下表所示 DRC 通常不会标记影响时序收敛难度的问题。而是改为标记因非建议的约束导致的时序分析准确性问题。即使建立和保持时序裕量为正，硬件仍可能无法在所有工作条件下正常工作。大多数检查都涉及：设计边界上未定义的时钟、具有意外波形的时钟、缺少时序要求或不适当的 CDC 电路。对于此处最后一个类别，请使用 `report_cdc` 命令执行更全面的综合信息。

重要提示！请仔细验证严重性为“Critical Warning”（严重警告）的时序检查。

表 3：**验收质量方法 DRC**

检查	严重性	描述
TIMING-1、TIMING-2、TIMING-3、TIMING-4 和 TIMING-27	Critical Warning	非建议的时钟源点定义
TIMING-5、TIMING-25 和 TIMING-19	Critical Warning	意外的时钟波形
TIMING-9 和 TIMING-10	Warning	未知或不完整的 CDC 电路
TIMING-11	Warning	不适当的 `set_max_delay -datapath_only` 命令
TIMING-12	Warning	已禁用“时钟再收敛消极因素移除”
TIMING-13 和 TIMING-23	Warning	由于路径中断导致的不完整时序分析
TIMING-17	Critical Warning	未设置时钟的时序单元

表 3：**验收质量方法 DRC** (续)

检查	严重性	描述
TIMING-18、TIMING-20 和 TIMING-26	Warning	缺少时钟或输入/输出延迟约束
TIMING-21 和 TIMING-22	Warning	MMCM 补偿的问题
TIMING-24	Warning	已改写 `set_max_delay -datapath_only` 命令
TIMING-29	Warning	多周期路径对不一致
TIMING-35	Critical Warning	在具有相同时钟的路径中不存在公共节点
TIMING-40 和 TIMING-43	Warning	时钟拓扑或要求不适当
TIMING-41	Warning	内部管脚上定义的传递时钟无效
TIMING-46	Warning	多周期路径含绑定 CE 管脚
TIMING-47	Warning	同步时钟之间存在伪路径或异步时钟组
TIMING-51	Critical Warning	来自并行 MMCM 或 PLL 的相关时钟之间无公用相位
TIMING-52	Critical Warning	来自扩展频谱 MMCM 的相关时钟之间无公用相位
TIMING-53	Critical Warning	来自 DPLL 的相关时钟之间无公用相位
TIMING-54	Critical Warning	时钟间存在如下约束：限定范围的伪路径、时钟组或仅最大延迟数据路径约束
TIMING-55	Critical Warning	多个时钟到达同一个 CMB 去歪斜管脚
TIMING-56	Warning	缺少按逻辑或物理方式排除的时钟组约束
TIMING-57	Warning	不受支持的配置，其中包含 PHASESHIFT_MODE 和数字去歪斜

其他时序方法 DRC

其他 TIMING 和 XDC 检查可用于识别如下约束：可能导致运行时间增加的约束、覆盖现有约束的约束，或者对网表名称变更非常敏感的约束。相应的信息可用于调试约束冲突。请务必留意 TIMING-28 检查（自动衍生的时钟，供时序约束引用），因为修改设计源代码并重新同步时，自动衍生的时钟名称可能发生更改。在此情况下，先前定义的约束将不再有效，或者将应用到错误的时序路径。

评估设计的最高频率

您可以采用如下方法对给定架构上运行的设计的最大频率 (FMAX) 以及速度等级进行定义和评估：以迭代方式增大目标时钟频率并重新运行综合与实现，直至在完整布线的设计上，时序分析报告显示建立时序裕量违例 (WNS < 0) 较小为止。AMD 建议将 Default 综合指令或 PerformanceOptimized 综合指令与 Explore 实现指令和策略搭配使用，以得到可实现的最佳 FMAX。在某些情况下，根据设计规模以及关键逻辑路径的性质，其他策略显示的 FMAX 可能更高。对于含较小的建立时间违例的实现结果，最大频率计算方式如下：

- FMAX (MHz) = max(1000/(Ti - WNSi))

其中：

- Ti 表示在实现运行“i”期间所使用的目标时钟周期 (ns)
- WNSi 表示在实现运行“i”期间所使用的目标时钟的最差负时序裕量 (ns)

其他重要注意事项：

- 如果使用的时钟周期过紧，可能导致 Vivado 实现工具中自动降低时钟周期以避免因目标不现实和时序违例过大而导致的编译时间过长。请改用合理范围内紧凑的时钟约束。

- 对于含多个时钟的设计，您必须按比例减少所有同步时钟周期，直至在实现后其中某一时钟周期开始发生时序约束失败（最好是最快的时钟或者含时序路径最多的时钟）。

注释：在 `report_timing` 或 `report_timing_summary` 报告中并不会显式提供 FMAX 值。

对于给定的设计实现，硬件上跨目标器件速度等级所支持的温度和电压范围的最大工作频率的定义方式为 1000/(T - WNS)，其中 WNS 为正值或负值。在标称温度和电压条件下工作（通常在实验室环境内）时，通常可以略高一些的频率来运行设计。

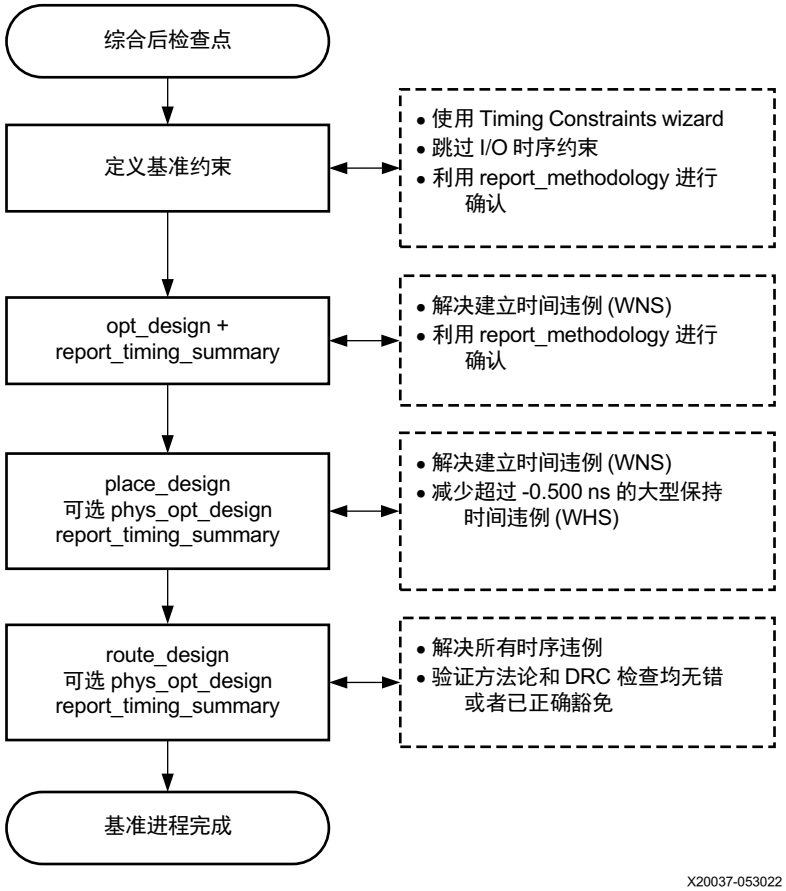
注释：要增大设计的最大频率，可使用本章中所述的技巧或者使用“Intelligent Design Runs”（智能设计运行）。

设计基线设定

设定基线是创建最简单的时序约束的过程，初始情况下忽略 I/O 时序。完全约束所有时钟后，设计中含起点和端点的所有路径都将完成自动约束。由此可提供简单的机制用于识别内部器件时序困难，即使设计不断演变也是如此。由于设计可能存在时钟域交汇，基线约束还必须包含指定时钟（包括生成时钟）之间的关系。

设定设计基线时，每次完成实现步骤后都必须通过分析并解决整个流程中的时序难题来满足时序要求。首先，创建简单且有效的约束以在 AMD Vivado™ 实现工具中展现真实的时序状况。然后，通过不同实现步骤迭代时，即可在执行下一步之前解决时序违例。下图显示了设定基线的过程。

图 2：**设计基线设定**



在基线设定完成后，您可以：

- 消除较小的时序违例
- 实现全面约束覆盖
- 先单独对新模块设定基线，然后将模块添加到顶级设计

建议：AMD 建议在设计进程初就创建基线约束，并根据这些基线约束来规划针对设计 HDL 的任何重大更改。

定义基线约束

要创建最简单的约束集合，请使用不含用户时序约束的有效的综合后 Vivado 检查点。开启检查点后，使用 Timing Constraints Wizard 来定义约束。此向导会引导您逐步完成以结构化方式创建约束的整个流程。

在此阶段中无需定义所有约束。默认情况下，如果没有约束，那么 Vivado 工具会忽略 I/O 时序。因此，此时您无需定义 I/O 时序约束。而可改为在流程后期完成基线设定流程后再定义 I/O 时序约束。

提示：使用 Timing Constraints Wizard 时，请取消选中建议的 I/O 时序约束。

要准确了解器件中的内部时序，请定义如下约束：

- 所有时钟约束
- 时钟域交汇 (CDC) 约束

  默认情况下，同步时钟之间的 CDC 路径可安全完成时序约束，但您必须使用安全的 CDC 电路，并指定异步时钟之间的时序例外。

创建约束后，请识别无法满足时序的路径。重写对应的 RTL 或放宽时钟周期。

重要提示！交付的所有 AMD IP 以及合作伙伴 IP 都具有符合 AMD 约束方法论的特定 XDC 约束。综合和实现期间会自动包含 IP 约束。创建约束并为其设定基线时，必须保持 IP 约束完整。

如果不使用 Timing Constraints Wizard 来定义约束，那么请参阅以下章节，其中涵盖了手动定义基线约束所需执行的步骤。

确定必须创建哪些时钟

首先将综合后的网表或检查点加载到 Vivado IDE 中。在 Tcl 控制台中，使用 `reset_timing` 命令确保移除所有时序约束。

使用 `report_clock_networks` Tcl 命令创建设计中必须定义的所有基准时钟列表。生成的时钟网络列表会显示应创建的时钟约束。使用“Timing Constraints Editor”（时序约束编辑器）为每个时钟指定相应的参数。

确认没有时钟遗漏

在时钟网络报告显示所有时钟网络均已完成约束后，即可开始验证生成时钟的准确性。由于 Vivado 工具会自动通过时钟修改块来传播时钟约束，因此对生成的约束进行复查至关重要。使用 `report_clocks` 可显示使用 `create_clock` 约束创建的时钟以及所生成的时钟。

注释：MMCM、PLL、GT 和时钟缓冲器都属于时钟修改块。

`report_clocks` 的结果显示所有时钟都已完成传输。在属性字段中会显示使用 `create_clock` 创建的基准时钟与生成时钟之间的差异：

- 已传输 (P) 的时钟仅含基准时钟。

- 从其他时钟衍生的时钟显示为已传输 (P) 和已生成 (G)。
- 由时钟修改块生成的时钟显示为已自动衍生 (A)。
- 其他属性表明自动衍生的时钟已重命名 (R)、生成时钟相对于传入的主时钟出现波形反向 (I)，或者基准时钟为虚拟 (V) 时钟。

您还可以使用 `create_generated_clock` 约束来生成时钟。如需了解更多信息，请参阅《Vivado Design Suite 用户指南：使用约束》(UG903)。

图 3：**时钟报告显示从基准时钟生成的时钟**

```
Attributes
  P: Propagated
  G: Generated
  A: Auto-derived
  R: Renamed
  V: Virtual
  I: Inverted

Clock          Period(ns)  Waveform(ns)     Attributes  Sources
sysClk         10.000      {0.000 5.000}    P           {sysClk}
clkfbout       10.000      {0.000 5.000}    P,G,A       {clkgen/mmcm_adv_inst/CLKFBOUT}
cpuClk         20.000      {0.000 10.000}   P,G,A,R     {clkgen/mmcm_adv_inst/CLKOUT0}
wbClk_4        20.000      {0.000 10.000}   P,G,A       {clkgen/mmcm_adv_inst/CLKOUT1}
usbClk_3       10.000      {0.000 5.000}    P,G,A       {clkgen/mmcm_adv_inst/CLKOUT2}
phyClk0_2      10.000      {0.000 5.000}    P,G,A       {clkgen/mmcm_adv_inst/CLKOUT3}
phyClk1_1      10.000      {0.000 5.000}    P,G,A       {clkgen/mmcm_adv_inst/CLKOUT4}
fftClk_0       10.000      {0.000 5.000}    P,G,A       {clkgen/mmcm_adv_inst/CLKOUT5}
```

提示：要确认设计中已不存在未约束的端点，请参阅“检查时序报告”（`no_clock` 类别）。此报告可从“Timing Summary”（时序汇总）报告获取，或者也可使用 `check_timing` Tcl 命令获取。

约束时钟域交汇

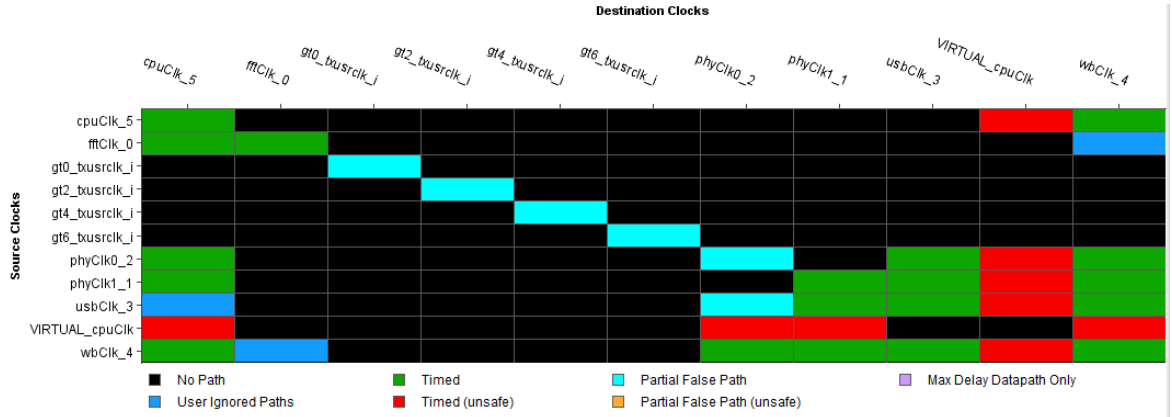
在验证时钟约束时，必须确定异步和过约束的时钟域交汇路径。

注释：本章节并非旨在解释如何正确设置跨时钟区域边界。而是解释如何识别存在哪些交汇以及如何对其加以约束。

检查时钟关系

您可以使用 `report_clock_interaction` Tcl 命令查看时钟之间的关系。该报告显示了源时钟和目标时钟构成的矩阵。每个单元中的颜色都表示时钟之间的交互类型，其中包括时钟间的所有现有约束。下图显示了 1 个时钟交互报告样本。

图 4：**时钟交互报告样本**



Id	Source Clock	Destination Clock	Edges (WNS)	WNS (ns)	TNS (ns)	Failing Endpoints (TNS)	Total Endpoints (TNS)	Path Req (WNS)	Edges (WHS)	WHS (ns)	THS (ns)	Clock Pair Classification	Inter-Clock Constraints
26	phyClk1_1	cpuClk_5	rise - rise	1.901	0.000	0	61	7.000	rise - rise	1.040	0.000	Clean	Timed
31	fftClk_0	cpuClk_5	rise - rise	1.063	0.000	0	61	7.000	rise - rise	1.488	0.000	Clean	Timed
34	VIRTUAL_c...	cpuClk_5	rise - rise	-1.372	-196.991	532	834		rise - rise	1.089	0.000	Virtual Clock	Timed (unsafe)
13	wbClk_4	fftClk_0				0	1635					Ignored	Asynchronous Groups
33	fftClk_0	fftClk_0	rise - rise	0.673	0.000	0	8320	7.000	rise - rise	0.046	0.000	Clean	Timed
1	gt0_txusrclk_i	gt0_txusrclk_i	rise - rise	9.923	0.000	0	222	12.800	rise - rise	0.135	0.000	Clean	Partial False Path
2	gt2_txusrclk_i	gt2_txusrclk_i	rise - rise	9.734	0.000	0	222	12.800	rise - rise	0.095	0.000	Clean	Partial False Path
3	gt4_txusrclk_i	gt4_txusrclk_i	rise - rise	8.856	0.000	0	222	12.800	rise - rise	0.123	0.000	Clean	Partial False Path

下表解释了此报告中每种颜色的含义。

表 4：**report_clock_interaction 颜色**

颜色	标签	含义	备注
黑	No path：无路径	这些时钟域之间无交互。	主要用于参考，除非您希望这些时钟域发生交互。
绿	Timed：已完成时序约束	这些时钟域间有交互，而且路径已完成时序约束。	主要用于参考，除非您不希望这些时钟域之间出现任何交互。
青	Partial False Path：部分伪路径	交互时钟域的某些路径因用户例外而未完成时序约束。	确保确实需要相应的时序例外。
红	Timed (unsafe)：已完成时序约束（不安全）	这些时钟域间有交互，而且路径已完成时序约束。但是，这些时钟似乎为独立（因而异步）时钟。	检查这些时钟是否应声明为异步，或者是否应共享公用基准时钟源。
橙	Partial False Path (unsafe)：部分伪路径（不安全）	这些时钟域间有交互。这些时钟似乎为独立（因而异步）时钟。但是，只有部分路径因例外而未完成时序约束。	检查时序例外未覆盖某些路径的原因。
蓝	User Ignored Paths：用户忽略的路径	在这些时钟域之间存在交互，并且由于时钟组或伪路径时序例外，导致路径未完成时序约束。	确认这些时钟应为异步时钟。另外，检查是否已正确写入对应 HDL 代码，以确保在时钟域之间实现正确同步和可靠的数据传输。
浅蓝	仅最大延迟数据路径	这些时钟域之间存在交互，并且通过 `set_max_delay -datapath_only` 对路径进行时序约束。	确认时钟处于异步状态且指定的延迟正确。

在创建任何伪路径或时钟组约束之前，矩阵中出现的颜色只有黑、红和绿。由于默认情况下所有时钟都已完成时序约束，因此对异步时钟进行去耦的过程至关重要。异步时钟去耦失败通常会导致设计严重过约束。

识别无公用基准时钟的时钟对

时钟交互报告可指示每对交互时钟是否具有公用基准时钟源。不共享公用基准时钟的时钟对通常彼此处于异步状态。因此，通过使用“Common Primary Clock”字段对报告中的列进行排序来识别这些时钟对是很有用的。此报告并非旨在判断时钟域交汇路径设计是否正确。

`report_cdc` Tcl 命令可用于对异步时钟之间的时钟域交汇电路进行全面分析。如需了解有关 `report_cdc` 命令的更多信息，请访问此链接以参阅《Vivado Design Suite 用户指南：设计分析与收敛技巧》(UG906) 中的相应内容。另请参阅《Vivado Design Suite Tcl 命令参考指南》(UG835) 中的 report_cdc。

识别苛刻的时序要求

对于每个时钟对，时钟交互报告还可显示最差路径的建立时间要求。按“Path Req (WNS)”（路径要求 (WNS)）对列进行排序，以查看设计中最苛刻的要求列表。请复查这些时序要求以确保不存在无效的苛刻要求。

Vivado 工具通过如下方式来确定路径要求：将每个时钟扩展至 1000 个时钟周期，然后确定发生非重合边沿对齐的最接近的位置。当 1000 个周期不足以确定最苛刻的要求时，此报告会显示“Not Expanded”，在此情况下，必须将 2 个时钟作为异步来处理。

例如，假设时序路径从 250 MHz 时钟跨越到 200 MHz 时钟：

- 200 MHz 时钟的正沿为 {0、5、10、15、20}。
- 250 MHz 时钟的正沿为 {0、4、8、12、16、20}。

当出现以下情况时，该时钟对的最苛刻的要求就会出现：

- 250 MHz 时钟在 4 ns 处出现上升沿。
- 200 MHz 时钟的下一个上升沿位于 5 ns 处。

这会导致时序从 250 MHz 时钟域约束到 200 MHz 时钟域内的所有路径都在 1 ns 进行时序约束。

注释：位于 20 ns 处的同步时钟沿在本例中并非最苛刻要求，因为捕获沿不能与发送沿相同。

由于这是 1 个相当苛刻的时序要求，您必须采取额外的步骤。根据设计的不同，下列约束之一可能成为处理这些交汇问题的正确方法：

- `set_clock_groups` / `set_false_path` / `set_max_delay -datapath_only`

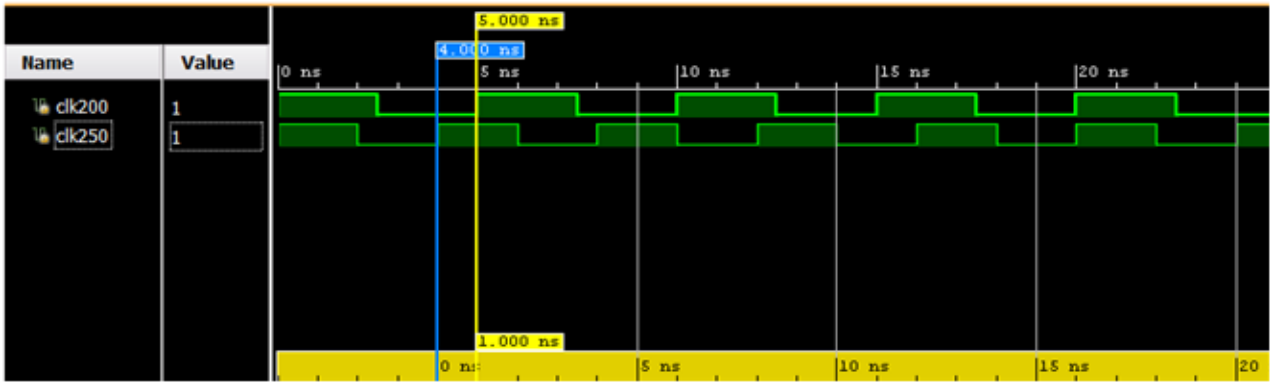
  将时钟对作为异步来处理时，请使用上述约束之一。使用 `report_cdc` Tcl 命令来确认时钟域交汇电路是否安全。

- `set_multicycle_path`

  在放宽时序要求时使用此约束，前提是已采用正确的时钟电路来对发送时钟沿和捕获时钟沿进行相应控制。

如果什么也不做，设计可能会出现跨这 2 个时钟域的时序违例问题。此外，所有最佳最优化、布局和布线都可能变为供这些路径专用而不是供设计中的真正关键路径使用。在执行任何时序驱动的实现步骤之前，请务必明确识别这些类型的路径。

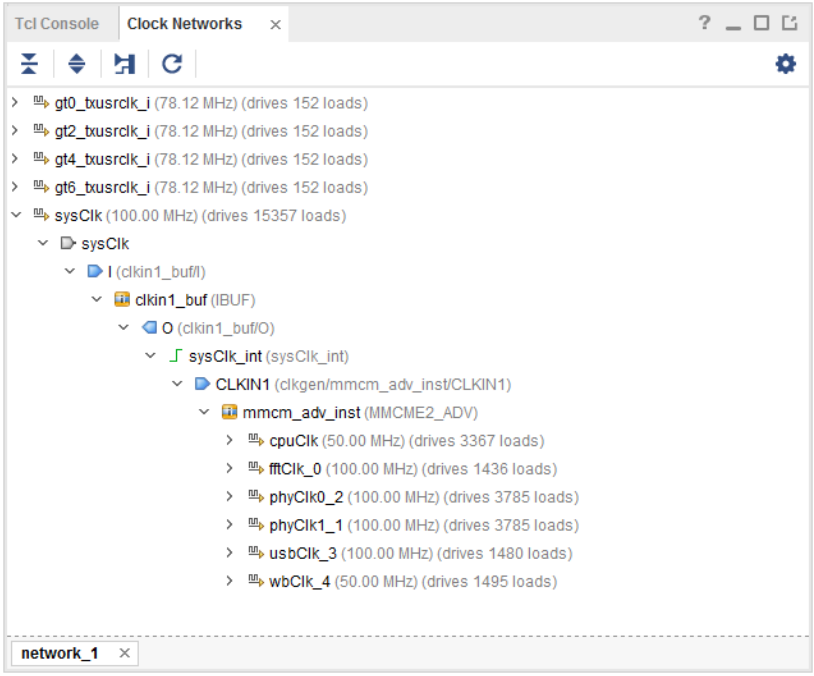
图 5：**从 250 MHz 到 200 MHz 的时钟域交汇**



同时约束基准时钟和生成时钟

创建任何时序例外前，最好返回 `report_clock_networks` 以识别设计中存在的基准时钟。如果所有基准时钟都彼此处于异步状态，那么您可使用单一约束来将基准时钟彼此去耦，并将其生成时钟彼此去耦。如下图所示，使用 `report_clock_networks` 中的基准时钟作为指导，即可将每个时钟组和关联时钟去耦。

图 6：**Report Clock Networks**




```
### Decouple asynchronous clocks
set_clock_groups -asynchronous \
-group [get_clocks sysClk -include_generated_clocks] \
-group [get_clocks gt0_txusrclk_i -include_generated_clocks] \
-group [get_clocks gt2_txusrclk_i -include_generated_clocks] \
-group [get_clocks gt4_txusrclk_i -include_generated_clocks] \
-group [get_clocks gt6_txusrclk_i -include_generated_clocks]
```

限制 I/O 约束和时序例外

大部分时序违例都发生在内部路径上。首次基线设定迭代期间无需 I/O 约束，对于其中发送或捕获寄存器位于 I/O bank 内的 I/O 时序路径尤其如此。当设计和其他约束稳定并且接近满足时序后，即可添加 I/O 时序约束。

提示：您可使用 `config_timing_analysis -ignore_io_paths yes` Tcl 命令在实现期间以及在使用时序信息的所有报告中忽略所有 I/O 路径上的时序。您必须在存储器中打开设计之前或者打开后立即手动输入此命令。

根据 RTL 设计师的建议，时序例外必须加以限制并且不得用于隐藏实际的时序问题。在此之前，必须复查并最终确定时钟之间的伪路径或时钟组。

IP 约束必须完整保留。如果 IP 时序约束缺失，那么已知的伪路径可能报告为时序违例。

完成每个步骤后评估设计 WNS

完成每次综合和每个实现步骤后，必须对设计 WNS 进行评估。如果您使用的是 Tcl 命令行流程，那么完成构建脚本中的每个实现步骤后，只需轻松整合 `report_timing_summary` 即可。如果您使用的是 Vivado IDE，那么完成每个步骤后可使用简单的 `tcl.post` 脚本来运行 `report_timing_summary`。在这两种情况下，发现 WNS 明显下降时，必须分析该步骤前最近的检查点。

除了完成每个实现步骤后对整个设计的时序进行评估外，还可针对各路径采用更有针对性的方法来评估流程中每个步骤对于时序的影响。例如，完成最优化步骤后时序路径的估算信号线延迟可能与布局后该路径的估算信号线延迟存在明显差异。完成每个步骤后对关键路径的时序进行比较是突显关键路径时序偏离收敛的有效方法。

综合后最优化与逻辑后最优化

估算的信号线延迟接近于所有路径可能实现的最佳布局。要修复违例路径，请尝试执行以下操作：

- 更改 RTL。
- 更改使用的综合选项。
- 添加可安全适用于硬件中的功能的时序例外（例如，多周期路径）。

布局前后

布局后，除使用较消极的延迟的长距离和中高扇出信号线外，估算的信号线延迟接近于可能的最佳路径。此外，此时信号线延迟中并未计入拥塞或保持时间修复影响，这导致时序结果较为乐观。

时钟偏差估算结果准确，可用于复查不平衡的时钟树对时序裕量的影响。可通过运行最小延迟分析来估算保持时间修复。slice、块 RAM 或 DSP 之间存在严重的保持时间违例（WHS 不小于 -0.500 ns）时，需要加以修复。可接受较轻微的违例，此类违例可能由其他布线器修复。

注释：往来专用块（如 PCIe® 块）的路径的保持时间估算结果可能大于 -0.500 ns，并由其他布线器自动修复。对于上述状况，布线后请检查 `report_timing_summary` 以验证是否已修复所有对应的保持时间违例。

物理最优化前后

评估是否需要运行物理最优化以修复如下对象相关的时序问题：

- 具有高扇出的信号线（`report_high_fanout_nets` 显示了最高扇出非时钟信号线）
- 所含驱动与负载位置相去较远的信号线
- 流水线寄存器使用率欠佳的数字信号处理器 (DSP) 和块 RAM

布线前后

除未完全完成布线的信号线外，实际已布线的信号线都将报告所含时序裕量。时序裕量可反映保持时间修复对于建立时间的影响以及拥塞产生的影响。

布线后不应残余任何保持时间违例，与最差建立时序裕量 (WNS) 值无关。如果设计未满足保持时间要求，则需进行进一步的分析。常见原因为拥塞过高，在此情况下布线器会放弃对时序进行最优化。此外保持时间违例严重（超过 4 ns）时也同样如此，默认情况下布线器不修复此违例。严重的保持时间违例通常是由于时钟约束不正确、时钟偏差过高或者 I/O 约束不正确所导致的，此类错误在布局后或者甚至在综合后都可能出现。

如果满足保持时间要求 (WHS > 0) 但不满足建立时间要求 (WNS < 0)，请遵循 分析并解决时序违例 中所述步骤进行操作。

基线设定与时序约束确认流程

以下流程有助于跟踪您的时序收敛进展情况并识别潜在瓶颈：

1. 打开已综合的设计。
2. 运行 `report_timing_summary -delay_type min_max`，并记录下表中所示信息。

   表 5：**已综合的设计的时序汇总报告**

	WNS	TNS	故障端点数	WHS	THS	故障端点数
综合 (Synth)						

3. 打开综合后 `report_timing_summary` 文本报告，并记录 `check_timing` 的 `no_clock` 部分。

   设计中未满足时钟要求的时钟数量：_____

4. 运行 `report_clock_networks` 以识别设计中的基准时钟源管脚/端口。

   设计中未约束的时钟数量：_____

5. 运行 `report_clock_interaction -delay_type min_max` 并按 WNS 路径要求对结果进行排序。

   设计中最小 WNS 路径要求：_____

6. 按 WHS 对 `report_clock_interaction` 结果进行排序，以查看综合后是否存在严重保持时间违例 (>500 ps)。

   设计中最大负 WHS：_____

7. 按时钟间约束对 `report_clock_interaction` 结果进行排序，列出显示为不安全的所有时钟对。
8. 打开综合设计时，存在多少严重警告 (Critical Warning)?

   综合设计中的严重警告数：_____

9. 存在哪些类型的严重警告?

   记录每种类型的示例。

10. 运行 `report_high_fanout_nets -timing -load_types -max_nets 25`。

    不受 FF 驱动的高扇出信号线数量：_____

    不受 FF 驱动的最高扇出信号线上的负载数量：_____

    是否有任何高扇出信号线存在负时序裕量？如果有，那么 WNS = _____

11. 实现设计。执行每个步骤时，请运行 `report_timing_summary` 并记录信息，如下表所示。

表 6：**时序汇总报告**

	WNS	TNS	故障端点数	WHS	THS	故障端点数
最优化 (Opt)						
布局 (Place)						
物理最优化 (Physopt)						
布线 (Route)						

12. 运行 `report_exceptions -ignored` 以识别设计中是否存在重叠的约束。记录结果。

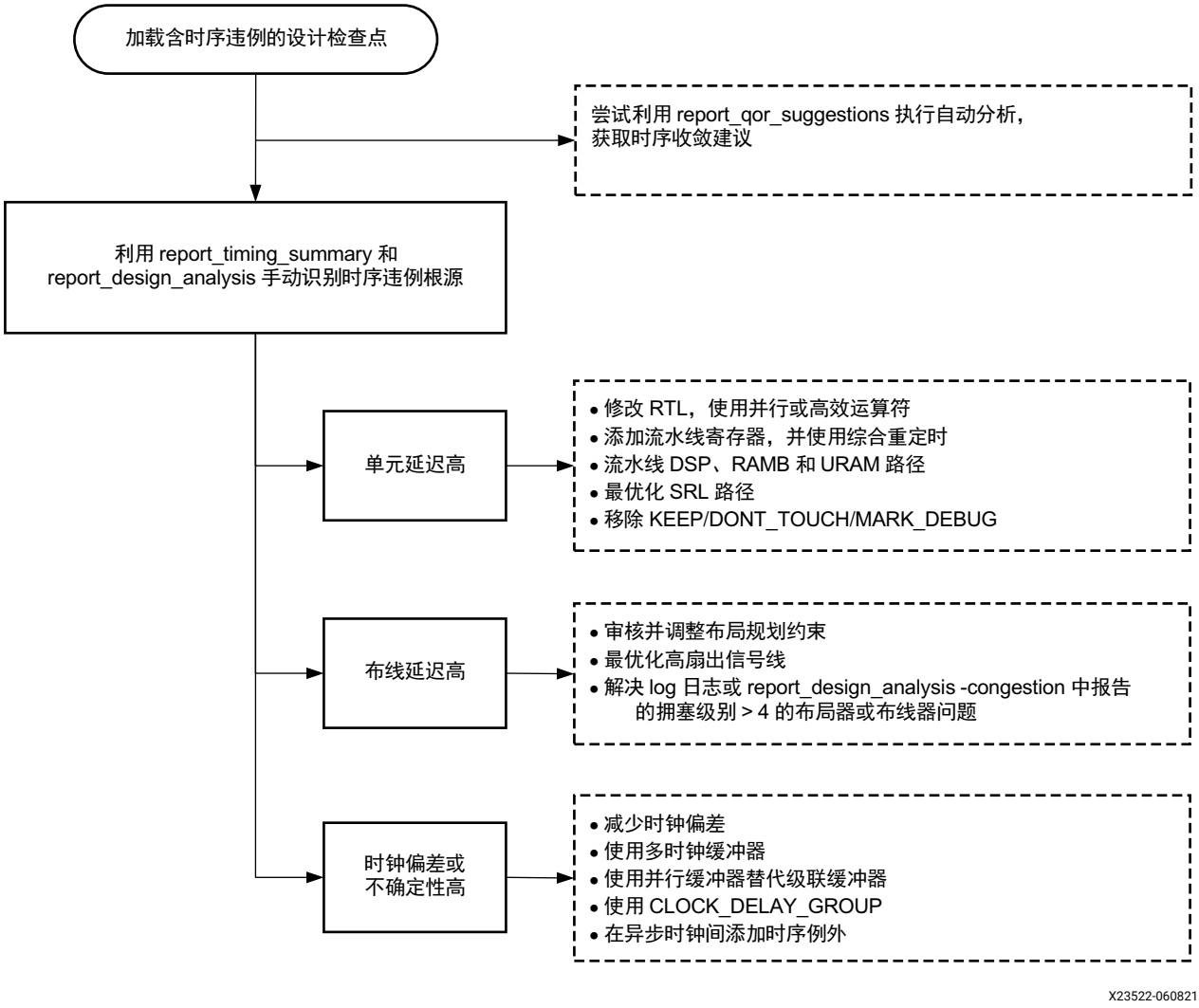
分析并解决时序违例

时序驱动的算法侧重于解决每个时钟域的最严重的违例。修复最严重的违例后，通常重新运行实现工具时，这些工具即可自动解决其他严重性较低的关键路径的问题。您可通过集中解决影响大量路径的问题以加速此修复流程。例如，纠正次优时钟通常会影响大量路径，因此 AMD 建议首先集中解决这些问题，然后再处理特定路径的问题。

“Report QoR Suggestions”（QoR 建议报告）命令可自动识别问题，并根据严重性对建议进行排序。您可通过在应用建议前后分别运行 Report QoR Assessment 命令来判定时序收敛进展情况。QoR 评估得分提高且标记待查的详细表格内容减少即表明结果已得到改善。

下图显示了分析并解决时序违例的基本流程。

图 7：**分析并解决时序违例**



注释：如需了解有关多重输出时钟缓冲器（MBUFG 原语）的使用方式的更多信息，请访问此链接以参阅《Versal 自适应 SoC 硬件、IP 和平台开发方法指南》(UG1387) 中的相应内容。

识别时序违例的根源

对于建立时间，必须首先分析每个时钟组的最差情况违例。时钟组是指由给定时钟捕获到的所有时钟内部路径、时钟间路径和异步路径。

对于保持时间，必须按以下方式复查所有违例：

- 布线前，仅复查超过 0.5 ns 的违例。
- 布线后，从最差情况违例开始。

审查时序裕量

有多个因素会影响建立时间裕量和保持时间裕量。您可通过审查建立时间和保持时间裕量公式来轻松识别其中每个因素，该公式采用如下简化形式：

- 时序裕量（建立/恢复）= 建立路径要求：

  - 数据路径延迟（最大值）

  + 时钟偏差

  - clock uncertainty（时钟不确定性）

  - 建立/恢复时间

- 时序裕量（保持/移除）= 保持路径要求：

  + 数据路径延迟（最小值）

  - 时钟偏差

  - clock uncertainty（时钟不确定性）

  - 保持/移除时间

为进行时序分析，时钟偏差始终按照以下方式计算：

- 时钟偏差 = 目标时钟延迟 - 源时钟延迟（位于任何已有公共节点后）

对违例时序路径进行分析期间，必须审查每个变量的相关影响，以判定哪个变量对违例的影响最大。然后，可开始对主要影响因素进行分析，以了解哪些路径特性对其值影响最大，并尝试识别为降低其影响所需的设计或约束变更。如果无法进行设计或约束变更，那么必须对所有其他影响因素执行同样的分析（从影响最大的因素开始）。以下列表显示了典型的影响因素排序顺序（按影响从高到低排序）。

对于建立/恢复时间：

- 数据路径延迟：从数据路径延迟中减去时序路径要求。如果差值相当于负值时序裕量，则表明路径要求过于苛刻或者数据路径延迟过大。
- 数据路径延迟 + 建立/恢复时间：从数据路径延迟加建立/恢复时间之和减去时序路径要求。如果差值相当于负值时序裕量，则表明路径要求过于苛刻或者建立/恢复时间大于正常值并且对违例存在明显影响。
- 时钟偏差：如果时钟偏差与时序裕量具有相近的负值，并且偏差绝对值达数百 ps，那么偏差是主要的影响因素，并且您必须审查时钟拓扑结构。
- 时钟不确定性：如果时钟不确定性超过 100 ps，那么您必须审查时钟拓扑结构和抖动数值以了解不确定性过高的原因。

对于保持/移除时间：

- 时钟偏差：如果时钟偏差超过 300 ps，那么您必须审查时钟拓扑结构。
- 时钟不确定性：如果时钟不确定性超过 200 ps，那么您必须审查时钟拓扑结构和抖动数值以了解不确定性过高的原因。
- 保持/移除时间：如果保持/移除时间达数百 ps，那么您可审查原语数据手册，以确认这是否符合期望。
- 保持路径要求：此要求通常为 0。如果不为 0，那么必须验证时间约束是否正确。

假定所有时序约束均准确合理，那么最常见的时序违例影响因素通常是数据路径延迟（针对建立/恢复时序路径）和偏差（针对保持/移除时序路径）。在设计周期的早期阶段，可通过分析这 2 个影响因素来修复大部分时序问题。但在改进和优化设计与约束后，剩余的违例是由多种因素组合而造成的，您必须并行审查所有因素以确定需要改进哪些因素。

请访问此链接以获取《Vivado Design Suite 用户指南：设计分析与收敛技巧》(UG906) 中有关时序分析概念的更多信息，并访问此链接以获取该文档中有关时序报告 (`report_timing_summary/report_timing`) 的信息。

使用设计分析报告

当难以实现时序收敛时或者在尝试提升应用的总体性能时，必须在运行综合后以及执行实现流程的任一步骤后审查设计的主要特性。QoR 分析通常要求您同时查看多个全局和局部特性，以确定设计和约束中哪些部分处于次优状态，或者哪些逻辑结构不适合目标器件架构和实现工具。`report_design_analysis` 命令可用于收集逻辑、时序和物理特性，并合并展示在多个表中以便简化 QoR 根源分析。

注释： `report_design_analysis` 不会提供时序约束的完整性和正确性方面的报告。

提示： 在 Vivado IDE 中运行“设计分析”报告可改进可视性、自动筛选和简化交叉探测。

以下部分仅介绍时序路径特性分析。“设计分析”报告还可提供有关拥塞和设计复杂性的实用信息。

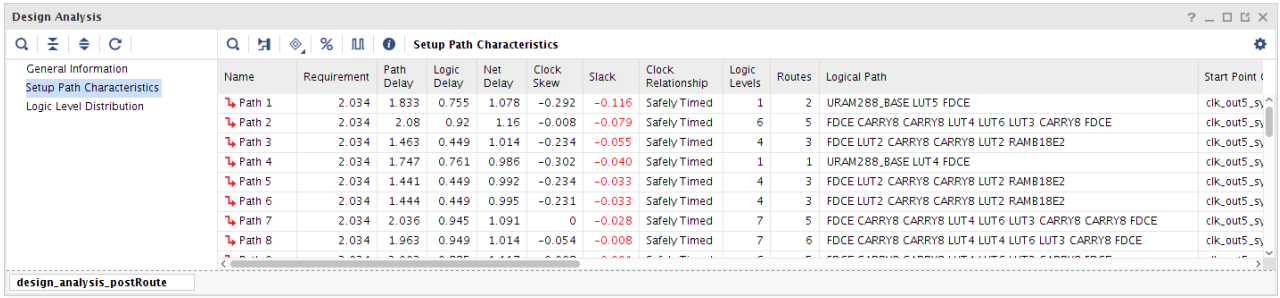
分析路径特性

要报告 50 条最差的建立时序路径，可使用 Vivado IDE 中的“Report Design Analysis”（设计分析报告）对话框，或者也可以使用以下命令：

```
report_design_analysis -max_paths 50 -setup -name design_analysis_postRoute
```

下图显示了由此命令生成的“Setup Path Characteristics”（建立路径特性）表格示例。要在窗口中查看其他列，可进行水平滚动。

图 8：**设计分析报告之布线后时序路径特性**



Name	Requirement	Path Delay	Logic Delay	Net Delay	Clock Skew	Slack	Clock Relationship	Logic Levels	Routes	Logical Path	Start Point
Path 1	2.034	1.833	0.755	1.078	-0.292	-0.116	Safely Timed	1	2	URAM288_BASE LUT5 FDCE	clk_out5_s
Path 2	2.034	2.08	0.92	1.16	-0.008	-0.079	Safely Timed	6	5	FDCE CARRY8 CARRY8 LUT4 LUT6 LUT3 CARRY8 FDCE	clk_out5_s
Path 3	2.034	1.463	0.449	1.014	-0.234	-0.055	Safely Timed	4	3	FDCE LUT2 CARRY8 CARRY8 LUT2 RAMB18E2	clk_out5_s
Path 4	2.034	1.747	0.761	0.986	-0.302	-0.040	Safely Timed	1	1	URAM288_BASE LUT4 FDCE	clk_out5_s
Path 5	2.034	1.441	0.449	0.992	-0.234	-0.033	Safely Timed	4	3	FDCE LUT2 CARRY8 CARRY8 LUT2 RAMB18E2	clk_out5_s
Path 6	2.034	1.444	0.449	0.995	-0.231	-0.033	Safely Timed	4	3	FDCE LUT2 CARRY8 CARRY8 LUT2 RAMB18E2	clk_out5_s
Path 7	2.034	2.036	0.945	1.091	0	-0.028	Safely Timed	7	5	FDCE CARRY8 CARRY8 LUT4 LUT6 LUT3 CARRY8 CARRY8 FDCE	clk_out5_s
Path 8	2.034	1.963	0.949	1.014	-0.054	-0.008	Safely Timed	7	6	FDCE CARRY8 CARRY8 LUT4 LUT4 LUT6 LUT3 CARRY8 FDCE	clk_out5_s

以下是使用此表格的方式：

- 单击“%”（显示百分比）按钮切换显示数字和 %。这对于检查单元延迟和信号线延迟的比例非常实用。
- 默认情况下，仅隐藏含 null 或空值的列。单击“Hide Unused”（隐藏不使用项）按钮以关闭过滤功能并显示所有列，也可右键单击表格标题以选择要显示或隐藏的列。

在该表中，可明确识别导致每条路径产生时序违例的各项特性：

- 逻辑延迟百分比过高（逻辑延迟）
  - 逻辑层次是否过多？(LOGIC_LEVELS)
  - 是否存在阻碍逻辑最优化的任何约束或属性？（DONT_TOUCH、MARK_DEBUG）
  - 路径是否包含具有高逻辑延迟的单元，例如块 RAM 或 DSP 等？（“Logical Path”（逻辑路径）、“Start Point Pin Primitive”（起点管脚原语）、“End Point Pin Primitive”（端点管脚原语））

- 当前路径拓扑结构的路径要求是否过于苛刻？（要求）

- 高信号线延迟百分比（信号线延迟）
  - 在路径中是否有任何高扇出信号线？（高扇出，累积扇出）
  - 分配给多个 Pblock 的单元布局能否拉开距离？(Pblocks)
  - 单元布局能否拉开距离？（边界框大小，时钟区域距离）
  - 在布局看似正确的情况下，是否有 1 个或多个信号线延迟值远高于预期？在“Device”（器件）窗口中选择路径并显示其布局和布线。
  - 在块 RAM 或 DSP 单元中是否缺少流水线寄存器？（Comb DSP、MREG、PREG、DOA_REG、DOB_REG）

- 高偏差（建立 <-0.5 ns，保持 > 0.5 ns）（时钟偏差）
  - 此路径是时钟域交汇路径吗？（起点时钟，端点时钟）
  - 时钟是同步时钟还是异步时钟？（时钟关系）
  - 此路径是否跨多个 I/O 列？（IO 交汇）

提示：要在 Vivado IDE 中直观显示时序路径的详细信息，请选择表格中的路径，然后转至“Properties”（属性）选项卡。

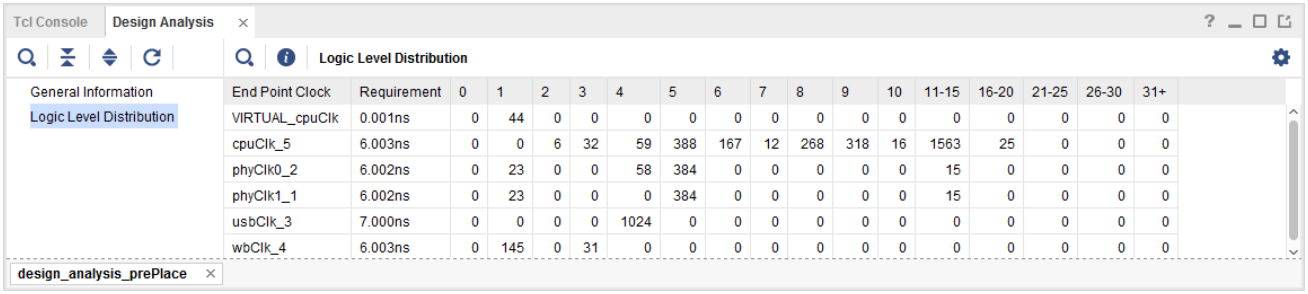
复查逻辑层次分布

`report_design_analysis` 还可为最差的 1000 条路径（默认情况下）生成“Logic Level Distribution”（逻辑层次分布）表，以供您用于识别设计中存在的长路径。通常最长的路径首先由布局器加以最优化以满足时序要求，这可能导致较短的路径的布局质量劣化。您必须不断尝试消除较长的路径，以提高整体时序约束 QoR。因此，AMD 建议在布局之前检查最长路径。

下图显示了如下设计的“逻辑层次分布”示例，在此设计中，最差的 5000 条路径包含具有 17 个逻辑层次的困难路径，而时钟周期为 7.5 ns。运行以下命令，以获取此报告：

```
report_design_analysis -logic_level_distribution -logic_level_dist_paths
5000 -name design_analysis_prePlace
```

图 9：**设计分析报告之布局前时序路径特性**



End Point Clock	Requirement	0	1	2	3	4	5	6	7	8	9	10	11-15	16-20	21-25	26-30	31+
VIRTUAL_cpuClk	0.001ns	0	44	0	0	0	0	0	0	0	0	0	0	0	0	0	0
cpuClk_5	6.003ns	0	0	6	32	59	388	167	12	268	318	16	1563	25	0	0	0
phyClk0_2	6.002ns	0	23	0	0	58	384	0	0	0	0	0	15	0	0	0	0
phyClk1_1	6.002ns	0	23	0	0	0	384	0	0	0	0	0	15	0	0	0	0
usbClk_3	7.000ns	0	0	0	0	1024	0	0	0	0	0	0	0	0	0	0	0
wbClk_4	6.003ns	0	145	0	31	0	0	0	0	0	0	0	0	0	0	0	0

对于高于 10 的逻辑层次，可使用 `-min_level` 和 `-max_level` 选项来提供所识别的最低和最高层次之间的路径的相关分布详情。例如：

```
report_design_analysis -logic_level_distribution -min_level 16 -max_level
20
-logic_level_dist_paths 5000 -name design_analysis_1
```

运行以下命令，以生成最长路径的时序报告：

```
report_timing -name longPaths -of_objects [get_timing_paths -setup -to
[get_clocks
cpuClk_5] -max_paths 5000 -filter {LOGIC_LEVELS>=16 && LOGIC_LEVELS<=20}]
```

根据结果，可通过更改 RTL 或使用不同综合选项来改进网表，或者也可以修改时序约束和物理约束。

数据路径延迟和逻辑层次

通常，路径中 LUT 和其他原语的数量是导致延迟的最重要的因素。因为在不同器件中报告的 LUT 延迟不同，所以必须考量单独限定单元延迟与布线延迟范围。

如果造成路径延迟主要是因为：

- 在 AMD Versal™ 器件中，单元延迟 >50%。

  是否可通过修改路径以将其缩短或者使用更快的逻辑单元？请参阅 减少逻辑延迟。

- 在 Versal 器件中布线延迟 >50%。

  路径是否受到保持时间修复的影响？您可通过运行 `report_design_analysis -show_all` 并检查“Hold Detour”（保持时间绕行）来确定答案。使用对应的分析技巧。

  - 是 - 受影响的信号线是否处于 CDC 路径中？
    - 是 - CDC 路径是否缺少约束？
    - 否 - 经过保持时间修复的路径的起点和端点是否使用平衡的时钟树？查看偏差值。
  - 否 - 请参阅以下拥塞相关信息。

  此路径是否受到拥塞影响？查看每个信号线的延迟和扇出，并观察启用布线详细信息的“Device”视图中的布线（仅限布线后分析）。您还可开启拥塞指标来查看路径位于拥塞区域内部还是附近。使用以下分析步骤进行快速评估或参阅 降低因拥塞导致的信号线延迟，以便开展综合分析。

  - 是 - 对于具有最高延迟值的信号线，扇出是否比较低 (<10)？
    - 是 - 如果布线看似处于最优化状态（直线），但驱动与负载相去较远，那么拥塞原因是布局处于次优状态。请参阅 解决拥塞，以确定最佳解决方法。
    - 否 - 尝试使用物理逻辑最优化来复制信号线驱动。复制后，每个驱动均可自动布局在靠近其负载的位置，这将减少总体数据路径延迟。请参阅 最优化高扇出信号线 以获取更多详细信息，并了解有关替代方法的信息。
  - 否 - 设计散布范围太广。请尝试通过以下方法来改进布局：
    - 减少控制集
    - 调节编译流程
    - 布局规划注意事项

时钟偏差和不确定性

AMD 器件使用各类布线资源支持大部分常用时钟方案与要求，例如高扇出时钟、短传输延迟和极低的偏差。时钟偏差会影响具有组合逻辑或互连的寄存器间路径。

建议：运行设计分析报告 (`report_design_analysis`) 可生成时序报告，其中包含有关时钟偏差数据的信息。验证时钟信号线不包含过大的时钟偏差。

高频率时钟域（超过 300 MHz）中的时钟偏差会影响性能。通常，时钟偏差应不大于 500 ps。例如，500 ps 表示 300 MHz 时钟周期的 15%，等同于 1 级或 2 级逻辑的时序预算。在跨域时钟路径中偏差可能更高，原因是这些时钟使用了不同的资源，并且公共节点位于时钟树的更高层级。基于 SDC 的工具会同时调整所有时钟的时序，除非约束指定禁止此操作（例如，`set_clock_groups`、`set_false_path` 或 `set_max_delay -datapath_only`）。

如果时钟不确定性超过 100 ps，那么您必须审查时钟拓扑结构和抖动数值以了解不确定性过高的原因。

相关信息

减少时钟偏差
减少时钟不确定性

减少逻辑延迟

Vivado 实现首先集中处理最关键的路径，这通常导致布局或布线后，困难程度较低的路径变为关键路径。AMD 建议在综合后或者执行 `opt_design` 后识别并改进最长的路径，因为此类路径对时序和功耗 QoR 的影响最大并且通常可显著减少达成时序收敛所需的布局和布线迭代数量。

在布局之前，时序分析所使用的估算延迟对应于理想布局和典型时钟偏差。通过使用 `report_timing`、`report_timing_summary` 或 `report_design_analysis`，您可快速识别含过多逻辑层级的路径或含高单元延迟的路径，因为这些路径布局前通常无法满足时序要求或者勉强满足时序要求。使用 识别时序违例的根源 中提出的方法论来查找实现设计前需要改进的长路径。

最优化常规互连结构路径

常规互连结构路径是互连结构寄存器或移位寄存器之间遍历各种资源（例如，LUT）的路径。“`report_design_analysis` 时序路径特性”表提供了最佳逻辑路径拓扑结构汇总，其中可识别如下问题：

- 多个小型 LUT 已级联

  到 LUT 的映射受如下因素影响：层级、存在的 KEEP_HIERARCHY、DONT_TOUCH 或 MARK_DEBUG 属性的影响或者含扇出（10 和更高）的中间信号。请运行 `opt_design -remap` 选项或者使用 AddRemap 或 ExploreWithRemap 指令来折叠小型 LUT，并减少逻辑级数。如果由于小型 LUT 之间的信号线扇出大于 1 导致 `opt_design` 无法最优化最长的路径，那么可以通过在 LUT 上设置 LUT_REMAP 属性来强制执行最优化。

- 路径止于移位寄存器 (SRL)

  通过使用 RTL 中的 SRL_STYLE 属性将第 1 个寄存器拉出移位寄存器。欲知详情，请访问此链接以参阅《Vivado Design Suite 用户指南：综合》(UG901) 中的相应内容。或者，您还可以使用在执行 `opt_design` 之前应用的 SRL_STAGES_TO_REG_INPUT 属性来实现相同的最优化。欲知详情，请访问此链接以参阅《Vivado Design Suite 用户指南：实现》(UG904) 中的相应内容。

  注释： `report_qor_suggestions` Tcl 命令可自动应用这种最优化技巧。

- 路径止于互连结构寄存器 (FD) 时钟使能或同步置位/复位

  如果止于数据管脚 (D) 的路径具有更多的裕度和更少的逻辑级数，则使用 EXTRACT_ENABLE 或 EXTRACT_RESET 属性，并在 RTL 中的信号上将其设置为“no”。或者，也可通过在要最优化的寄存器上设置 CONTROL_SET_REMAP 属性来指示 `opt_design` 执行同样的最优化。

  注释： `report_qor_suggestions` Tcl 命令可自动应用这种最优化技巧。

  提示： 如需执行从综合后路径到对应 RTL 视图和源代码的交叉探测，请访问此链接以参阅《Vivado Design Suite 用户指南：设计分析与收敛技巧》(UG906) 中的相应内容。

  注释： 欲知详情，请访问此链接以参阅《Versal 自适应 SoC 硬件、IP 和平台开发方法指南》(UG1387) 中的相应内容。

使用专用块和宏原语对路径进行最优化

专用块与宏原语之间的路径或者双向往来路径（如 DSP、块 RAM、UltraRAM、NoC 主/从单元 (NMU/NSU)、AI 引擎和 XPIO）需特别关注，因为这些原语通常具有如下时序特性：

- 部分管脚的建立时间、保持时间或时钟输出 (clock-to-output) 时序 arc 值较高。例如，块 RAM 的时钟输出延迟分别约为 1.2 ns（无可选输出寄存器）和 0.3 ns（含可选输出寄存器）。请复查目标器件系列的数据手册以了解详情。
- NoC 输出管脚的时钟输出时序 arc 值较高。例如，NoC NSU 的时钟输出延迟约为 0.65 ns。
- 布线延迟比常规 FD/LUT 连接更高。
- 时钟偏差变化比常规 FD-FD 路径更高。
- 互连结构与器件顶层或底层的专用块（例如，XPIO 中的 AI 引擎和专用块，包括 XPHY 逻辑块、I/O 逻辑块和时钟修改块等）之间的布线延迟更高。

此外，相比于 CLB slice，其可用性和站点位置均受到限制，这导致其布局更为困难并产生 QoR 惩罚。

有鉴于此，AMD 建议如下：

- 尽可能采用流水线路径作为往来专用块与宏原语之间的路径。
- 重构连接到这些单元的组合逻辑，以将逻辑级数降低至少 1 或 2 个单元（前提是因流水打拍所产生的时延过大）。
- 布局之前，在这些路径上满足建立时序要求并超出至少 500 ps。
- 复制连接到过多专用块或宏原语的逻辑椎，以便按需将其布局在相隔较远的位置。
- 如果设计对于 DSP 块内部或往来 DSP 块的时序要求较为苛刻，请运行 `opt_design -dsp_register_opt` 以将寄存器移至更接近时序最优化的位置。

  注释：由于在 `opt_design` 期间时序为近似估算，您可能还需要运行 `phys_opt_design -dsp_register_opt` 来更正在预布局阶段未准确呈现时序的移动操作。
- 使用边界逻辑接口 (BLI) 触发器进行流水线触发器的布局，这些触发器可用于与 XPIO 中的 AI 引擎和专用块（如 XPHY 逻辑块、I/O 逻辑块和时钟修改块等）进行交互。部分 IP 可提供使用 BLI 触发器的选项。

相关信息

AI 引擎 PL 接口时序约束方法

降低因物理约束导致的信号线延迟

所有设计都附带一组最少量的物理约束，尤其是对应于 I/O 位置的约束以及（有时）对应于时钟设置和逻辑布局的约束。虽然当设计已准备好进行时序收敛后就无法再修改 I/O 位置，但必须对物理约束（如 Pblock 和 LOC）进行分析。使用“Timing Path Characteristics”（时序路径特性）表 `report_design_analysis` 可识别每个关键路径上存在的多个 Pblock 约束。

在 Vivado IDE 的“Properties”（属性）窗口中，可选择“Timing Path Characteristic”表中的路径以查看哪些 Pblock 正在约束路径中的单元。如果约束强制扩散逻辑，请考虑移除一项或多项 Pblock 约束。

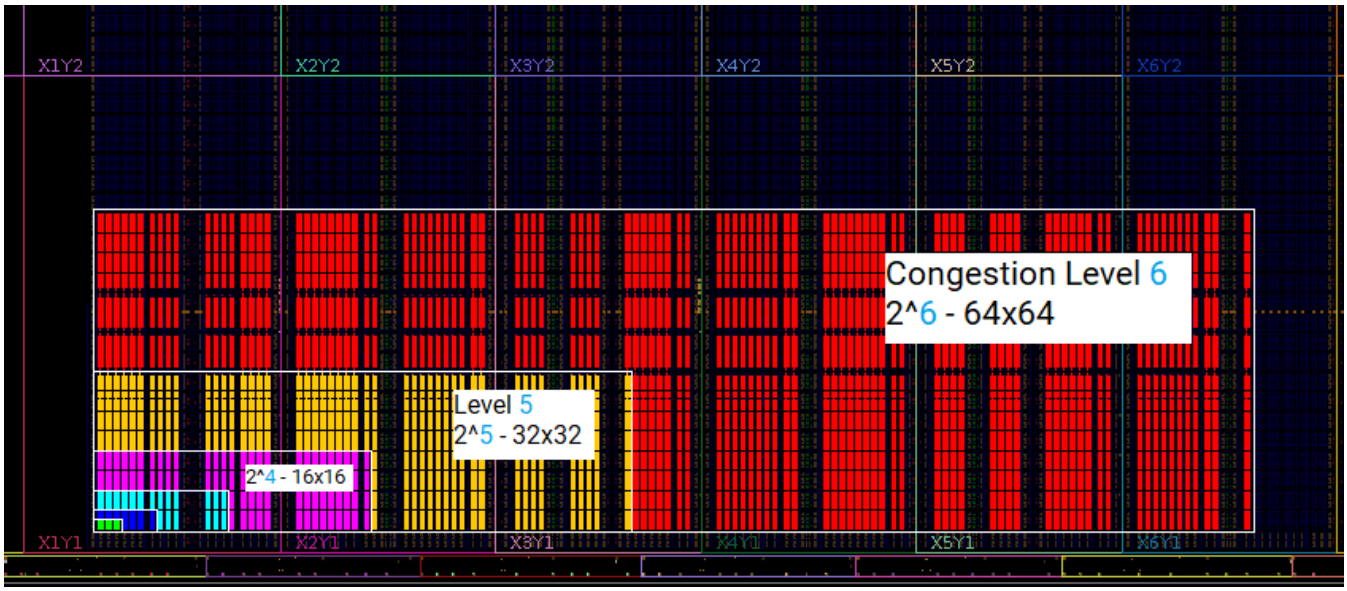
降低因拥塞导致的信号线延迟

如果关键路径布局在拥塞区域内部或附近，或者如果器件使用率过高并且已布局的设计几乎无法布线，那么器件拥塞可能导致难以实现时序收敛。大多数情况下，拥塞将显著增加布线器运行时间。如果路径显示已布线的延迟高于预期，请分析设计拥塞并识别缓解拥塞的最佳方法。

拥塞区域和等级定义

AMD 器件布线架构包括东西南北每个方向上各种长度的互连资源。拥塞区域即相邻互连块 (INT_XnYm) 或 CLB 块 (CLE_M_XnYm) 的最小正方形，其中特定方向的互连资源使用率接近或超过 100%。拥塞等级即对应于此正方形边长的正整数。下图显示了 AMD 器件上的拥塞区域相比于时钟区域的相对大小。

图 10：**“Device”视图中的拥塞等级和拥塞区域**



拥塞等级范围

分析拥塞时，工具报告的等级可按下表所示方式进行分类。

注释：拥塞等级为 5 或更高时，通常会影响 QoR 并且必然会导致布线器运行时间延长。

表 7：**拥塞等级范围**

等级	面积	拥塞	QoR 影响
1 和 2	2x2 和 4x4	无	无
3 和 4	8x8 和 16x16	轻微	可能导致 QoR 劣化
5	32x32	中等	QoR 劣化可能性较高
6	64x64	高	难以布线
7 和 8	128x128 和 256x256	无法操作	可能无法布线

“Device”窗口中“Interconnect Congestion Level”

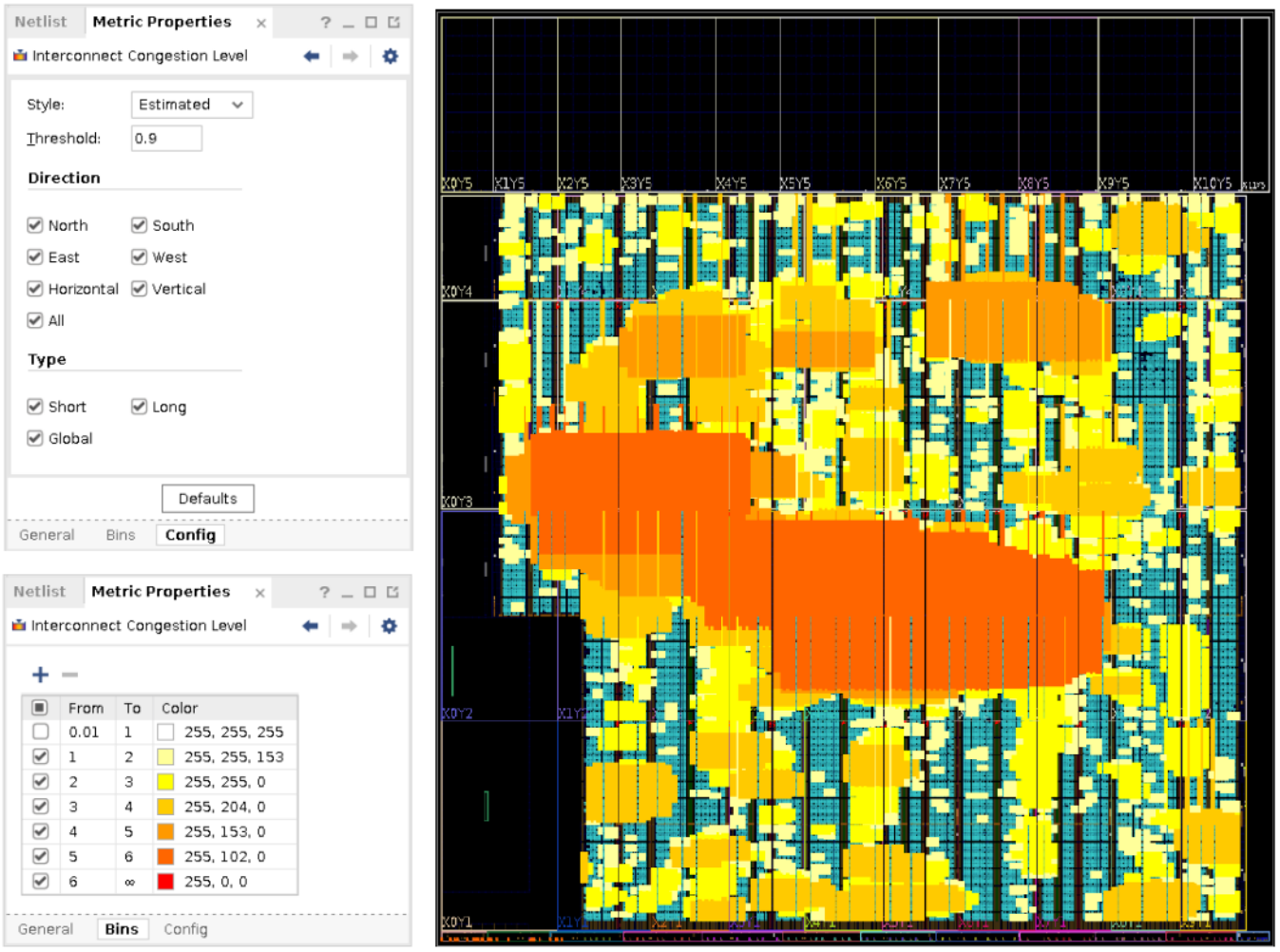
“Interconnect Congestion Level”（互连拥塞等级）指标突出显示了布线资源过度使用的最大连续区域。该指标默认显示估算值，与初始布线后的拥塞等级相似。如果存在布线，也可显示实际布线情况。布局或布线后，可在“Device”（器件）窗口中右键单击并选择“Metric” → “Interconnect Congestion Level”（指标 > 互连拥塞等级）来显示此拥塞指标。

“Interconnect Congestion Level”指标可提供器件中任意拥塞热点的直观概述。下图显示了含多个拥塞区域的布局设计。该指标基于当前互连需求和可用性，阈值为 0.9（即，布线使用率为 90%）。范围为 0.1 至 0.9。

您可以根据以下条件直观显示拥塞：

- 方向 (Direction)：北 (North)、南 (South)、东 (East)、西 (West)、垂直 (Vertical)、水平 (Horizontal)
- 类型 (Type)：短 (Short)、长 (Long)、全局 (Global)
- 方式 (Style)：估算 (Estimated)、布线 (Routed)、混合 (Mixed)

图 11： **“Device”窗口中的“Interconnect Congestion Level”示例**



布局器 Log 日志中的拥塞

布局器会在整个布局阶段中估算拥塞并在拥塞区域中分布此逻辑。这样有助于降低互连使用率以提高可布线性，并提高估算的延迟与布线延迟的相关性。但是，如果由于使用率高或其他原因导致无法降低拥塞，那么布局器就不会打印拥塞详细信息，而是改为发出以下警告：

```
WARNING: [Place 46-14] The placer has determined that this design is highly congested
and may have difficulty routing. Run report_design_analysis -congestion for a
detailed report.
```

在此情况下，QoR 很可能受到影响，最好先解决导致拥塞的问题，然后再继续使用布线器。正如消息中所述，请使用 `report_design_analysis` 命令报告实际拥塞等级，并识别拥塞位置以及布局在相同区域内的逻辑。

布局器 Log 日志中的拥塞

布线器根据拥塞等级和某些资源的布线难度发出附加消息。此外，布线器还可打印数份中间时序汇总。第一份时序汇总是在对所有时钟完成布线后打印的，通常显示 WNS/TNS/WHS/TNS 值，类似于布局后时序分析。下一份布线器中间时序汇总将在完成初始布线后报告。如果时序明显降低，则表明时序约束 QoR 已受到保持修复和/或拥塞的影响。

当拥塞等级为 4 或更高时，布线器会打印初始估算的拥塞表格，给出关于拥塞性质的更多细节：

- Global Congestion 类似于估算布局器拥塞的方式，根据所有互连类型来确定。
- Long Congestion 只考虑给定方向的长距离互连使用率。
- Short Congestion 考虑给定方向的所有其他互连使用率。

只要拥塞面积大于 32x32（等级 5）就可能会影响 QoR 和可布线性（在下表中以黄色高亮显示）。长距离 (Long) 互连上的拥塞增加了短距离 (Short) 互连的使用，从而导致更长的布线延迟。短距离 (Short) 互连上的拥塞通常会导致更长的运行时间，如果其拼块 (tile) % 超过 5%，也可能导致 QoR 劣化（在下表中以红色高亮显示）。

图 12：**初始估算拥塞表**

INFO: [Route 35-449] Initial Estimated Congestion

	Global Congestion		Long Congestion		Short Congestion	
Direction	Size	% Tiles	Size	% Tiles	Size	% Tiles
NORTH	16x16	1.95	32x32	1.68	32x32	11.58
SOUTH	8x8	1.90	16x16	2.00	32x32	9.23
EAST	8x8	0.93	2x2	0.20	32x32	9.14
WEST	8x8	1.37	2x2	0.15	32x32	14.50

在全局迭代 (Global Iterations) 期间，布线器首先尝试找到如下合规解决方案：无重叠、同时满足建立和保持的时序要求，且保持修复的优先级更高。当布线器在全局迭代期间不收敛时，它会停止最优化时序，直至找到有效的布线解决方案为止，如以下示例所示：

```
Phase 4.1 Global Iteration 0
 Number of Nodes with overlaps = 1157522
 Number of Nodes with overlaps = 131697
 Number of Nodes with overlaps = 28118
 Number of Nodes with overlaps = 10971
 Number of Nodes with overlaps = 7324
WARNING: [Route 35-447] Congestion is preventing the router from routing all nets.
The router will prioritize the successful completion of routing all nets over timing
optimizations.
```

找到有效的布线解决方案后，将重新启用时序最优化。

此外，该布线还会标记 CLB 布线拥塞，并提供最拥塞的 CLB 的名称。这样即可发出一条“Info”（参考）消息，并将拥塞的 CLB 和信号线写入消息正文中列出的文本文件。您可检查文本文件，获取 CLB 管脚输送拥塞所涉及的 CLB 拼块和拥塞的信号线列表，并使用“解决拥塞”部分中所列的拥塞缓解方法来解决 CLB 拥塞，然后再进行设计布线。

```
INFO: [Route 35-443] CLB routing congestion detected. Several CLBs have high routing
utilization, which can impact timing closure. Congested CLBs and Nets are dumped in:
iter_200_CongestedCLBsAndNets.txt
```

提示：局部 CLB 布线拥塞可能导致布线失败，即使“全局”拥塞、“长距离”拥塞或“短距离”拥塞所报告的拥塞等级在可接受范围内（小于 5）也是如此。请在生成的文本文件中查找是否存在上述消息以及是否存在局部拥塞热点。

最后，如果布线器无法找到合规的布线解决方案，那么将显示几条“Critical Warning”消息，以表明未完全布线的信号线数量和具有重叠的互连资源数量，如以下示例所示。

```
CRITICAL WARNING: [Route 35-162] 44084 signals failed to route due to routing
congestion. Please run report_route_status to get a full summary of the design's
routing.
...
CRITICAL WARNING: [Route 35-2] Design is not legally routed. There are 91566 node
overlaps.
```

提示：在布线期间，信号线散布在拥塞面积周围，这通常在成功完成设计布线后可降低 log 日志文件中报告的最终拥塞等级。

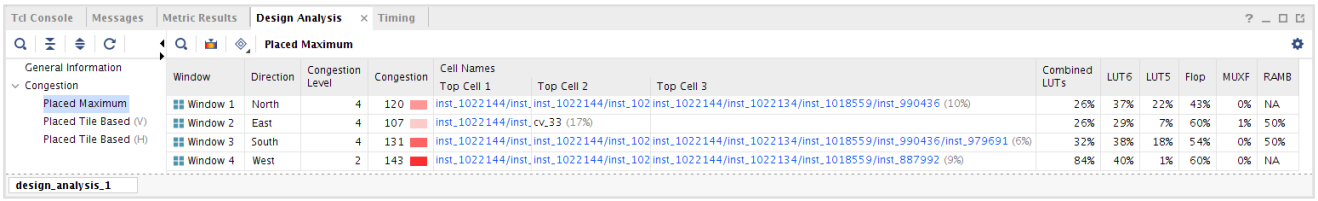
相关信息

解决拥塞

设计分析报告之拥塞报告

为帮助识别拥塞，Report Design Analysis 命令支持您生成拥塞报告以显示器件的拥塞区域，以及这些区域内存在的设计模块的名称。此报告中的拥塞表会显示布局器和布线器算法发现的拥塞区域。下图显示了拥塞表示例。

图 13：**拥塞表**



“Placed Maximum”（已布局的最大拥塞）、“Initial Estimated Router Congestion”（初始估算的布线器拥塞）和“Router Maximum”（布线器最大拥塞）拥塞表可提供有关东西南北四个方向上拥塞最严重的区域的信息。选中该表中的窗口时，在“Device”（器件）窗口中会突出显示对应的拥塞区域。

该表可显示设计流程中不同阶段的拥塞情况：

- Placed Maximum：基于单元位置和布线的模型显示拥塞。
- Initial Estimated Router Congestion：显示布线器快速迭代后的拥塞。这是用于分析拥塞的最实用的阶段，因为它可准确展示因布局导致的拥塞情况。
- Router Maximum：显示广泛使用布线器降低拥塞后的拥塞。

“Congestion Table”（拥塞表）中的“Congestion”（拥塞）百分比显示拥塞窗口中的布线使用率。其中列出了位于拥塞窗口中的前 3 个层级单元，可在“Device”窗口或“Schematic”（板级原理图）窗口中选中并交叉探测这些单元。此外在拥塞窗口中还可显示单元使用率。

确认拥塞区域中存在的层级单元后，即可使用本指南后文中介绍的拥塞缓解技巧来尝试减少总体设计拥塞。

如需了解有关生成和分析 Report Design Analysis 拥塞报告的更多信息，请访问此链接以参阅《Vivado Design Suite 用户指南：设计分析与收敛技巧》(UG906) 中的相应内容。

设计分析报告之复杂性报告

复杂性报告 (Complexity Report) 可按顶层设计和/或层级单元的叶节点单元的类型显示 Rent 指数 (Rent Exponent)、平均扇出 (Average Fanout) 和分布方式。Rent 指数是指在使用最小割 (min-cut) 算法以递归形式对设计进行分区时，网表分区的端口数量和单元数量之间的关系。其计算方法与在全局布局期间布局器所使用的算法类似。因此，它可准确表明布局器所面临的困难，当设计的层级与在全局布局期间所发现的物理分区匹配良好时尤其如此。

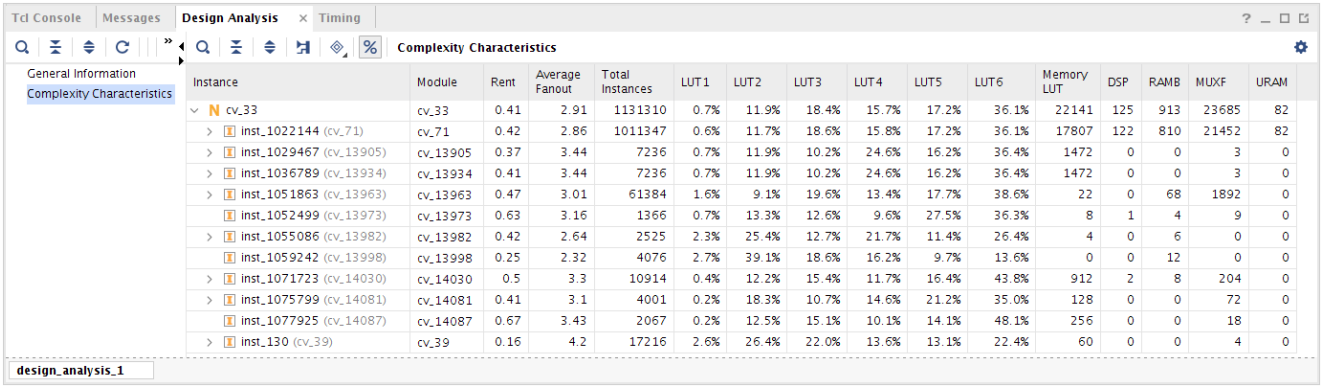
Rent 指数较高的设计表示此类设计中包含逻辑紧密相连的分组，并且这些分组与其他分组同样连接紧密。这通常可理解为全局布线资源使用率较高并且布线复杂性也更高。此报告中提供的 Rent 指数是根据未布局和未布线的网表来计算的。完成布局后，相同设计的 Rent 指数可能改变，因为它基于物理分区而不是逻辑分区。

执行以下任一操作时，将以“Complexity Mode”（复杂性模式）来运行“Report Design Analysis”（设计分析报告）：

- 在“Report Design Analysis”（设计分析报告）对话框的“Options”（选项）选项卡中，选中“Complexity”（复杂性）选项卡。
- 执行含 `-complexity` 选项的 `report_design_analysis` Tcl 命令。

下图显示了“复杂性报告”。

图 14：**复杂性报告**



Instance	Module	Rent	Average Fanout	Total Instances	LUT1	LUT2	LUT3	LUT4	LUT5	LUT6	Memory LUT	DSP	RAMB	MUXF	URAM
cv_33	cv_33	0.41	2.91	1131310	0.7%	11.9%	18.4%	15.7%	17.2%	36.1%	22141	125	913	23685	82
inst_1022144 (cv_71)	cv_71	0.42	2.86	1011347	0.6%	11.7%	18.6%	15.8%	17.2%	36.1%	17807	122	810	21452	82
inst_1029467 (cv_13905)	cv_13905	0.37	3.44	7236	0.7%	11.9%	10.2%	24.6%	16.2%	36.4%	1472	0	0	3	0
inst_1036789 (cv_13934)	cv_13934	0.41	3.44	7236	0.7%	11.9%	10.2%	24.6%	16.2%	36.4%	1472	0	0	3	0
inst_1051863 (cv_13963)	cv_13963	0.47	3.01	61384	1.6%	9.1%	19.6%	13.4%	17.7%	38.6%	22	0	68	1892	0
inst_1052499 (cv_13973)	cv_13973	0.63	3.16	1366	0.7%	13.3%	12.6%	9.6%	27.5%	36.3%	8	1	4	9	0
inst_1055086 (cv_13982)	cv_13982	0.42	2.64	2525	2.3%	25.4%	12.7%	21.7%	11.4%	26.4%	4	0	6	0	0
inst_1059242 (cv_13998)	cv_13998	0.25	2.32	4076	2.7%	39.1%	18.6%	16.2%	9.7%	13.6%	0	0	12	0	0
inst_1071723 (cv_14030)	cv_14030	0.5	3.3	10914	0.4%	12.2%	15.4%	11.7%	16.4%	43.8%	912	2	8	204	0
inst_1075799 (cv_14081)	cv_14081	0.41	3.1	4001	0.2%	18.3%	10.7%	14.6%	21.2%	35.0%	128	0	0	72	0
inst_1077925 (cv_14087)	cv_14087	0.67	3.43	2067	0.2%	12.5%	15.1%	10.1%	14.1%	48.1%	256	0	0	18	0
inst_130 (cv_39)	cv_39	0.16	4.2	17216	2.6%	26.4%	22.0%	13.6%	13.1%	22.4%	60	0	0	4	0

下表显示了 Rent 指数的典型范围。

表 8：**Rent 指数范围**

范围	含义
0.0 - 0.65	此范围较低或正常。
0.65 - 0.85	此范围较高，当实例总数高于 15,000 时尤其如此。
0.85 以上	此范围非常高，如果实例数量也非常高，那么这表明设计可能在实现过程中失败。

下表显示了“平均扇出”的典型范围。

表 9：**平均扇出范围**

范围	含义
低于 4	此范围正常。
4 - 5	此范围较高，表明可能难以实现无拥塞的设计布局。

表 9：**平均扇出范围** (续)

范围	含义
5 以上	此范围非常高，表明设计可能在实现过程中失败。

对于重要性较高的大型模块，必须妥善处理 Rent 指数和 Average Fanout 较高的情况。较小的模块（特别是总计少于 15,000 个实例时）可能 Rent 指数和 Average Fanout 较高，但仍可轻松成功完成布局和布线。因此，必须结合 Rent 指数和“Average Fanout”（平均扇出）复查“Total Instances”（实例总数）列。

提示：即使部分低级模块的 Rent 指数和“Average Fanout”较高，顶层模块的复杂性指标也不一定高。使用 `-hierarchical_depth` 选项优化分析以包含低级模块。

如需了解有关生成和分析“设计分析报告之复杂性报告”的更多信息，请访问此链接以参阅《Vivado Design Suite 用户指南：设计分析与收敛技巧》(UG906) 中的相关内容。

减少时钟偏差

为了满足诸如高扇出时钟、短传输延迟和低时钟偏差等要求，AMD 器件使用专用的布线资源来支持大多数常见的时钟方案。时钟偏差会严重降低高频时钟的时序预算。此外，器件使用率过高时，时钟偏差还会对实现工具同时满足建立时间和保持时间要求施加过多的压力。

对于时钟内时序路径，时钟偏差通常小于 300 ps，而对于平衡的同步时钟之间的时序路径，时钟偏差小于 500 ps。跨资源列时，时钟偏差表现出更多变化，这反映在时序裕量中并由实现工具加以最优化。对于不平衡的时钟树之间的时序路径或没有公共节点的时序路径，时钟偏差可达几纳秒，导致几乎不可能实现时序收敛。

要降低时钟偏差，请执行以下操作：

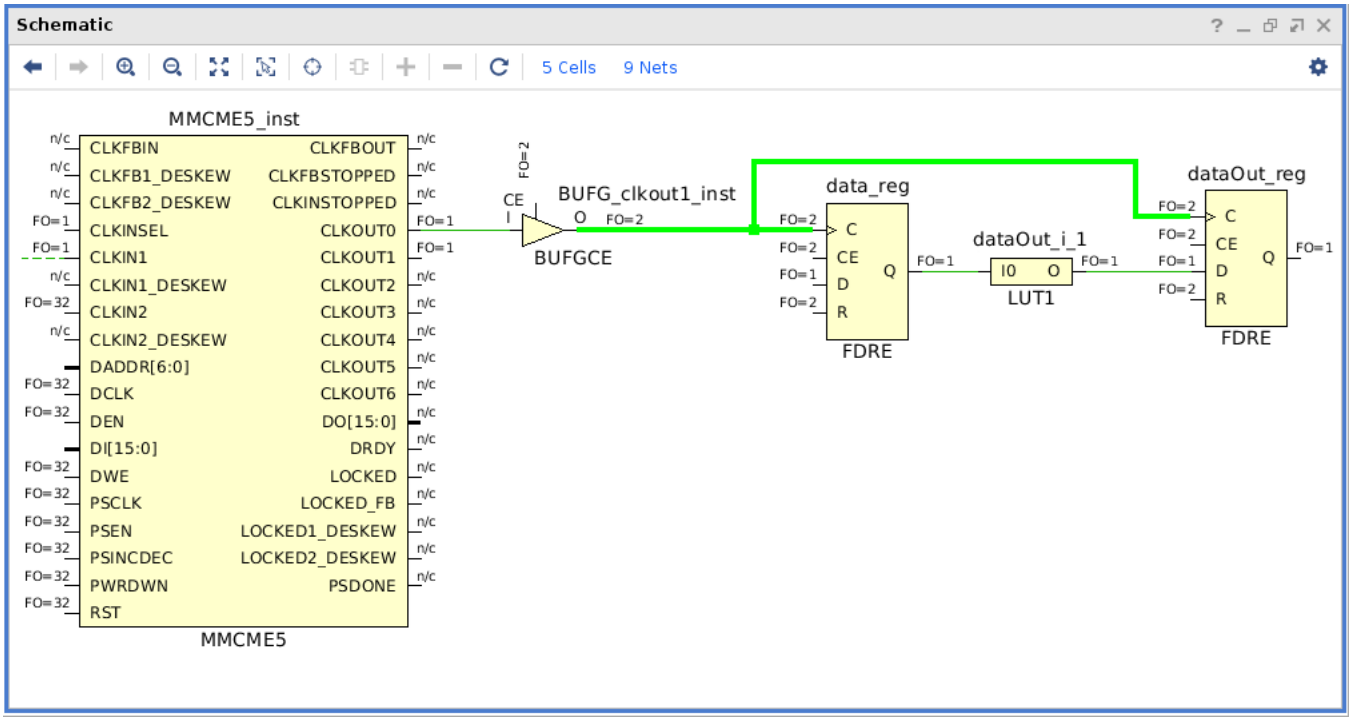
1. 复查所有时钟关系以确保只对同步时钟路径进行时序约束和最优化。

   注释：欲知详情，请访问此链接以参阅《Versal 自适应 SoC 硬件、IP 和平台开发方法指南》(UG1387) 中的相应内容。
2. 复查受到高于预期的时钟偏差影响的时钟树拓扑结构和时序路径的布局，如以下章节中所述。
3. 识别可减少时钟偏差的技巧，如以下章节中所述。

使用时钟内部时序路径

具有由相同时钟缓冲器驱动的相同源时钟和目标时钟的时序路径通常偏差极低。原因在于公共节点位于专用时钟网络上并靠近叶时钟管脚，如下图所示。

图 15：**具有公共节点（位于绿色信号线上）的典型同步时钟拓扑**



分析时序报告中的时钟路径时，不单独提供公共节点前后的延迟，因为公共节点仅存在于设计的物理设计库中，而不存在于逻辑视图中。因此，开启“Routing Resources”（布线资源）时，您可在 Vivado IDE 的“Device”（器件）窗口中看到公共节点，而无法在“Schematic”（板级原理图）窗口中看到此节点。时序报告仅提供偏差计算汇总，其中包括源时钟延迟、目标时钟延迟以及从时钟消极因素移除 (CPR) 直至公共节点的信用值。

在异步时钟之间添加时序例外

如果时序路径中所含源时钟和目标时钟源自不同基准时钟，或者没有公共节点、公共相位或公共周期，那么这些时序路径必须作为异步时钟来处理。在此情况下，偏差可能极其大，导致无法实现时序收敛。

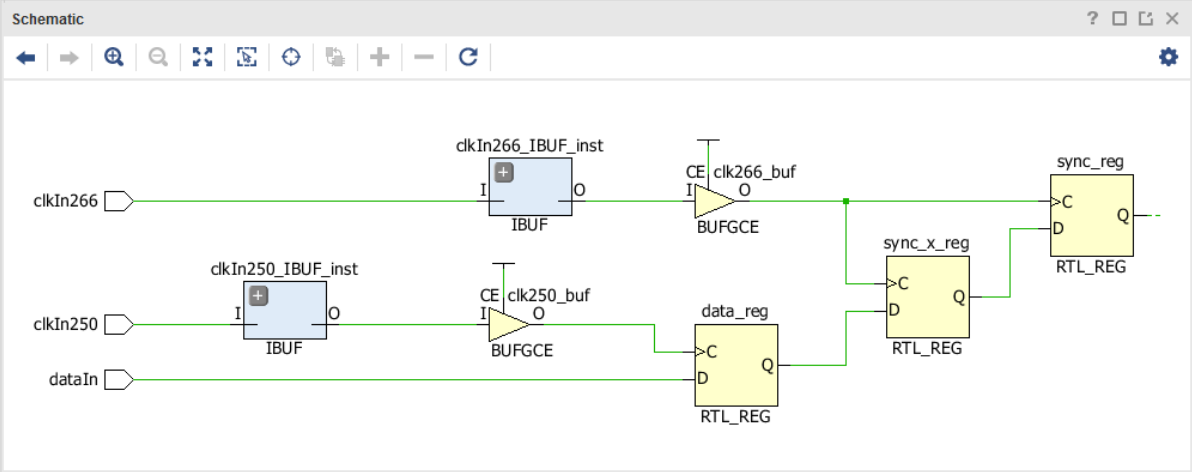
您必须复查异步时钟之间的所有时序路径以确保：

- 异步时钟域交汇电路 (`report_cdc`) 正确
- 忽略时序分析的时序例外定义（`set_clock_groups`，`set_false_path`）或者忽略偏差的时序例外定义 (`set_max_delay -datapath_only`)

您可使用“Clock Interaction Report”（时钟交互报告）(`report_clock_interaction`) 来帮助识别异步时钟和缺少正确的时序例外的时钟。

注释：欲知详情，请访问此链接以参阅《Versal 自适应 SoC 硬件、IP 和平台开发方法指南》(UG1387) 中的相应内容。

图 16：**具有正确 CDC 电路且不含公共节点的异步 CDC 路径**



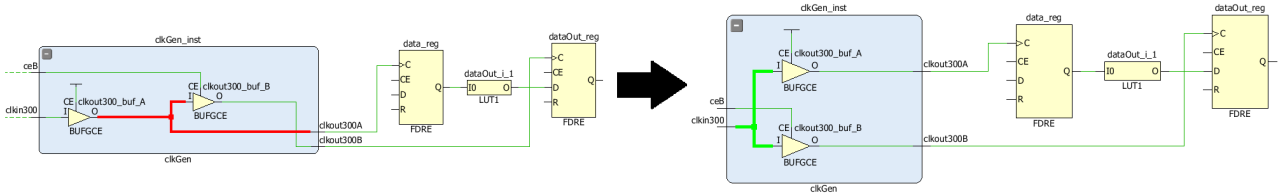
应用可减少时钟偏差的常见技巧

提示：鉴于 Versal 器件时钟架构的灵活性，`report_methodology` 命令包含相应的检查以帮助您创建最优化的时钟设置拓扑结构。

以下技巧适用于大部分常见场景：

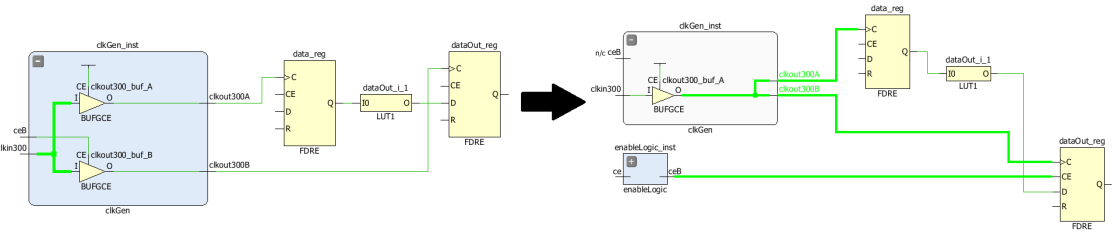
- 通过删除不必要的缓冲器或将其并行连接可以避免在级联时钟缓冲器之间出现时序路径，如下图所示。

图 17：**同步时钟拓扑，含级联 BUFG（已并行重新连接）**



- 将并行时钟缓冲器组合到单一时钟缓冲器中并将任何时钟缓冲器时钟使能逻辑连接到对应的时序单元使能管脚，如下图所示。如果缓冲器内置分频器对部分时钟进行分频，请使用时钟使能逻辑实现等效分频，并根据需要应用多周期路径时序例外。如果下游逻辑同时使用上升和下降时钟沿，或者如果功耗是一个重要因素，那么此技巧可能不适用。

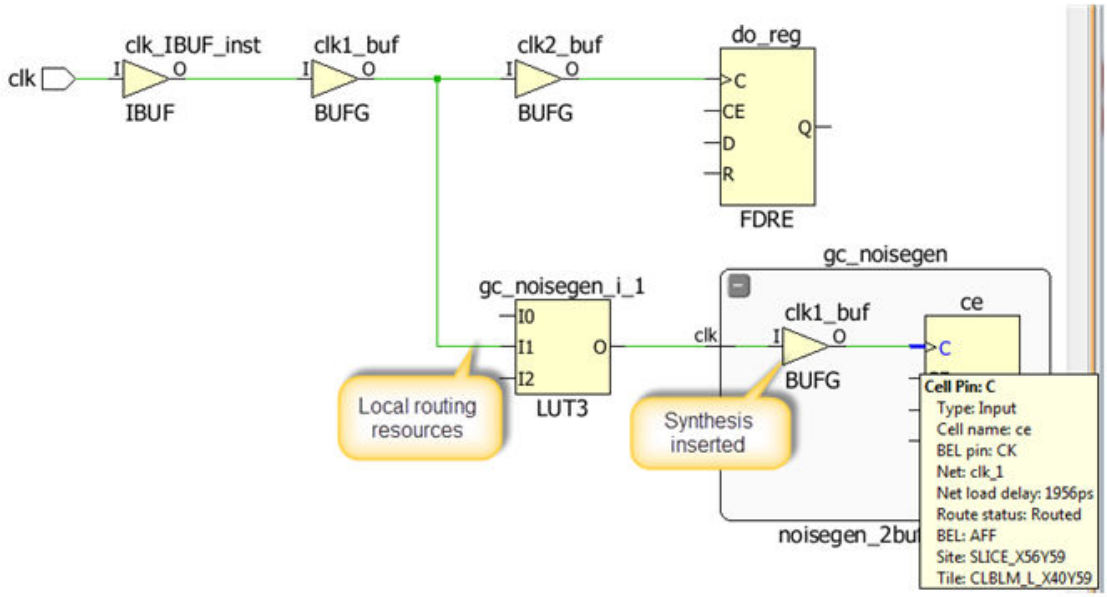
图 18：**同步时钟拓扑，含并行时钟缓冲器（已重组为单个缓冲器）**



- 移除时钟路径中的 LUT 或任何组合逻辑，因为它们会导致布局期间的时钟延迟和时钟偏差不可预测，从而导致结果质量降低。此外，部分时钟路径是使用通用互连资源进行布线的，这些资源相比于全局时钟资源对噪声更为敏感。组合逻辑通常来自于次优时钟门控转换，可迁移至时钟使能逻辑，并连接到时钟缓冲器或时序单元。

  在下图中，在 LUT3 中，第 1 个 BUFG (`clk1_buf`) 用于创建门控时钟条件。

图 19：**因时钟网络上的局部布线而引起的偏差**



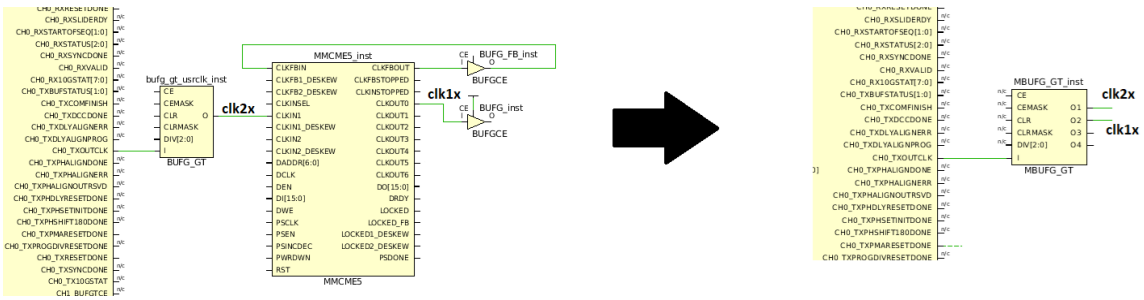
重要提示！要了解最佳实践并验证设计是否符合时钟设置准则的要求，请访问此链接以参阅《Versal 自适应 SoC 硬件、IP 和平台开发方法指南》(UG1387) 中的相应内容。

改进 Versal 器件的偏差

以下是使用 Versal 架构时降低时钟偏差的常规建议。欲知详情，请访问此链接以参阅《Versal 自适应 SoC 硬件、IP 和平台开发方法指南》(UG1387) 中的相应内容。

- 请避免使用 MMCM、XPLL 或 DPLL 来执行 BUFG_GT 时钟的简单分频。BUFG_GT 单元可以对输入时钟进行向下分频。为互连结构提供多次 BUFG_GT 时钟简单分频时，MBUFG_GT 单元可使用叶级分频对时钟进行向下分频，以便最大限度降低资源使用率并改进 QoR。下图显示了如何节省 MMCM 资源并使用 MBUFG_GT 为 2 个源自 GT*_QUAD 单元的时钟实现平衡的时钟树。

图 20：**使用 Versal MBUFG_GT 实现平衡的时钟树**



- 如果 GT 时钟需要进行频率综合，那么在具有 GT*_QUAD 资源的时钟区域中即存在 DPLL。

- 对于同步时钟之间的时序路径，请使用 MBUFGCE、MBUFGCE_DIV、MBUFGCTRL、MBUFG_PS 和 MBUFG_GT 原语来利用叶级分频、最大限度降低资源使用率并改进 QoR。
- 使用并行时钟缓冲器时，在布局和布线期间，针对关键的同步时钟的驱动程序信号线使用 CLOCK_DELAY_GROUP 来强制实现 CLOCK_ROOT 和布线匹配。时钟缓冲器必须由相同的单元驱动才能实现该约束。

  注释： `report_qor_suggestions` Tcl 命令可自动应用这种最优化技巧。
- 如果时序路径难以满足时序要求，并且偏差大于期望值，那么可能存在跨资源列或时钟区域的时序路径。如果情况如此，那么可使用物理约束（例如，Pblock）来强制将源和目标整合到单一时钟区域中或者阻止跨资源列，如片上网络 (NoC)、100G Multirate Ethernet MAC (MRMAC) 或 Integrated Block for PCIe (Gen4 x16)。
- 验证是否使用全局时钟资源对具有 CLOCK_DEDICATED_ROUTE = FALSE 约束的时钟信号线进行布线。使用 ANY_CMT_REGION 代替 FALSE 来确保仅使用专用时钟资源对具有布线豁免的时钟信号线进行布线。如果采用互连结构对时钟信号线与进行布线，请确认解决此情况所需的设计更改或时钟布局约束，并使实现工具改为使用全局时钟资源。采用互连结构布线的时钟路径时钟偏差可能较高，或者可能受开关噪声的影响，从而导致可实现的时钟频率欠佳或硬件中的设计无法正常运作。

减少 Versal 器件中的时钟延迟

在全局时钟布线中，时钟信号线首先通过水平和垂直布线轨道从全局时钟缓冲器布线至称为时钟根的集中位置。时钟信号线从时钟根驱动每个时钟区域内的时钟行穿过 3 级垂直分布轨道，形成平衡的 H 树型以最大程度减少垂直方向的时钟偏差。水平时钟分布在时钟区域边界处发生分段，在时钟区域边界处存在可编程延迟，用于提供水平方向的偏差平衡。

可编程延迟在时钟根处为最大，从该处开始向时钟网络边缘递减。对于某些时钟设置拓扑结构，减少时钟插入延迟可能比最大程度降低时钟偏差更为重要。例如，如果在同步 CDC 时钟设置路径中无法使用 MBUFG，且并行 BUFG_GT 或 BUFGCE_DIV 时钟缓冲器用于驱动同步时钟，那么最重要的是尽可能减小插入延迟，以减少相关时钟之间的最小/最大延迟变化量。在此情况下，应向并行时钟应用 CLOCK_DELAY_GROUP 属性以匹配时钟布线，并且应禁用可编程延迟，通过将时钟信号线的 GCLK_DESKEW 属性设置为 OFF 来尽可能减小插入延迟。通过使用 USER_CLOCK_ROOT 属性将时钟根分配至需最小插入延迟的负载旁可以进一步减小同步 CDC 时钟之间的偏差。

减少时钟不确定性

时钟不确定性表示相对于理想时钟的不确定程度。不确定性可能来自用户指定的外部时钟不确定性 (`set_clock_uncertainty`)、系统抖动或者占空比失真。诸如 MMCM 和 PLL 等时钟修改块还能以离散抖动 (Discrete Jitter) 和相位误差 (Phase Error)（如果使用多个相关时钟）的形式来影响时钟不确定性。

Clocking Wizard 可为指定器件提供准确的不确定性数据，并且可生成各种 MMCM 时钟配置用于比较不同时钟拓扑。为了实现目标架构的最佳结果，AMD 建议使用 Clocking Wizard 重新生成时钟生成逻辑，而不是使用先前架构中的原有时钟生成逻辑。

使用 MBUFGCE 减少时钟不确定性

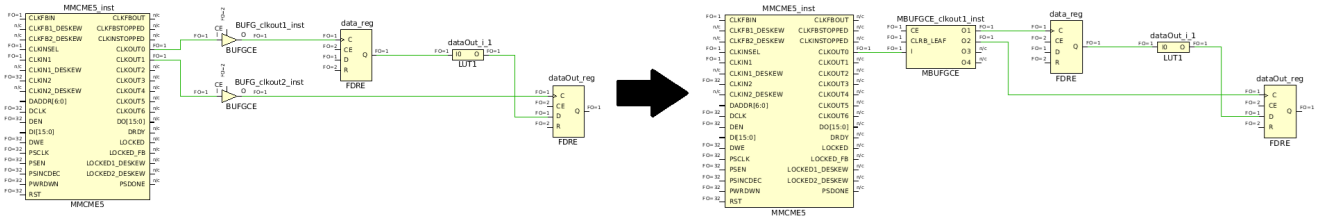
注释： `report_qor_suggestions` Tcl 命令可自动应用这种最优化技巧。

在 Versal 器件中，您可使用 MBUFGCE 单元通过消除 MMCM 相位误差来减少同步时钟域交汇上的时钟不确定性。以 300 MHz 和 150 MHz 时钟域之间路径为例，这 2 个时钟均由相同的 MMCM 生成。

注释： 针对 AMD UltraScale™ 器件推荐使用的并行 BUFGCE_DIV 拓扑结构同样可适用于 Versal 器件。但相比于使用 MBUFGCE 的叶级分频，并行 BUFGCE_DIV 拓扑结构耗用的功耗更多、使用的时钟资源更多，并且发生的偏差也更大。

在此情况下，时钟不确定性针对“建立”和“保持”分析均包含 120 ps 相位误差。此时，无需生成含 MMCM 的 150 MHz 时钟，而可改为将 MBUFGCE 连接到 300 MHz MMCM 输出，并在叶级将该时钟除以 2。

图 21：**为 Versal 器件同步 CDC 时序路径改进时钟拓扑**



以下是利用新拓扑结构实现的改进效果：

- 针对建立分析，时钟不确定性不包含 MMCM 相位误差，并且还减少了 120 ps。
- 针对保持分析，时钟不确定性未增加（仅适用于相同的边缘保持分析）。
- 公共节点移至靠近叶级分频器的位置，从而节省部分时钟消极因素。

下表提供了 Versal 器件同步 CDC 时序路径的建立分析与保持分析的时钟不确定性的比较结果。

表 10：**Versal 器件同步 CDC 时序路径的建立分析的时钟不确定性之比较**

建立分析	MMCM 生成的 150 MHz 时钟		MBUFGCE 150 MHz 时钟
	时钟抖动 (CJ)	0.405 ns	0.403 ns
	相位抖动 (PJ)	0.000 ns	0.000 ns
	相位误差 (PE)	0.120 ns	0.000 ns
	时钟不确定性	0.322 ns	0.202 ns

表 11：**Versal 器件同步 CDC 时序路径的保持分析的时钟不确定性之比较**

保持分析	MMCM 生成的 150 MHz 时钟		MBUFGCE 150 MHz 时钟
	时钟抖动 (CJ)	0.402 ns	0.000 ns
	相位抖动 (PJ)	0.000 ns	0.000 ns
	相位误差 (PE)	0.120 ns	0.000 ns
	时钟不确定性	0.322 ns	0.000 ns

运用常用时序收敛技巧

以下技巧有助于在极富挑战性的设计上完成时序收敛。在尝试使用这些技巧之前，请确保正确完成设计约束，并确定影响主要违例路径的主要问题。

建议：AMD 建议运行 `report_qor_suggestions` Tcl 命令来识别并自动应用其中大部分技巧。

利用块级综合策略来改进网表

虽然大部分设计使用默认 Vivado 综合设置即可满足时序要求，但更复杂的设计通常需要为不同层级混用综合策略才能达成时序收敛。

例如，某 1 个模块可能更适合使用 FF 资源代替 SRL，以在器件中实现流水打拍，但设计其余部分通过实现 SRL 中的逻辑而不是 FF 则可能更有利于减少逻辑扩散。在此情况下，请为需要使用 FF 资源的模块设置 ALTERNATE_ROUTABILITY 策略，并使用默认策略对设计其余部分进行综合。

注释：欲知详情，请访问此链接以参阅《Versal 自适应 SoC 硬件、IP 和平台开发方法指南》(UG1387) 中的相应内容。

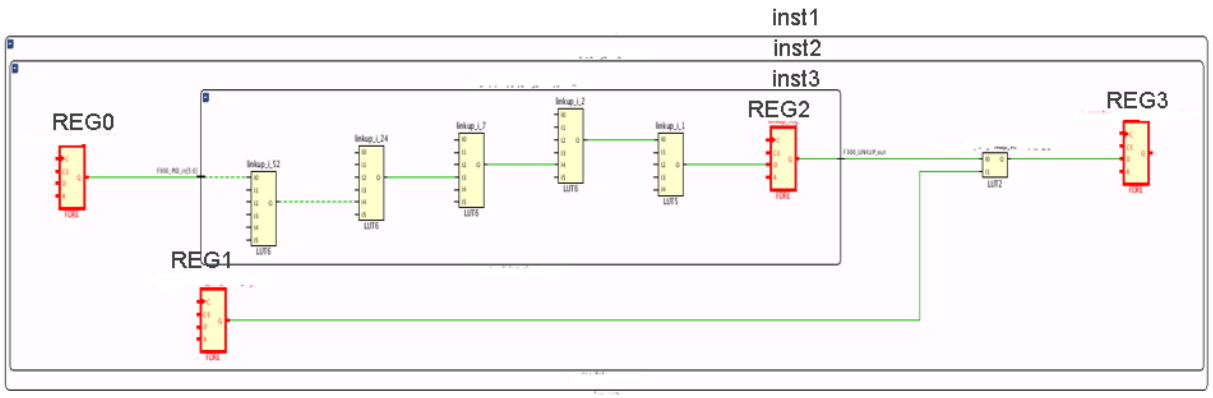
改进逻辑层次

在整个设计周期中，必须验证逻辑层次分布与目标 AMD 器件系列和器件速度等级的时钟频率目标是否相符。虽然限制含大量逻辑层次的路径的数量并不总是会导致时序收敛困难，但您可通过 Vivado 综合重定时选项对设计中最长的路径进行最优化，以便改进时序 QoR。

全局使用重定时选项通常会导致运行时间密度提高从而对功耗产生负面影响。因此，AMD 建议您在综合完成后或者使用最优化布局来识别在哪些具体层级中的含大量逻辑层次的路径上存在违例。如果最长路径的扇入或扇出中的路径所含逻辑层次较少，并且此类路径局限在中小型层级模块中，那么您可使用 BLOCK_SYNTH.RETIMING 块级综合策略。

下图显示了含 5 个 LUT 并受 600 MHz 时钟约束的关键路径。REG2 目标触发器用于驱动含单个 LUT 的时序路径，此 LUT 位于较 REG2 高 1 层的层级中。

图 22：**此板级原理图显示了含 5 个逻辑层级的关键路径**



除了使用 Vivado IDE 中的“Schematic”（板级原理图）窗口外，`report_design_analysis -logic_level_distribution` 命令同样可用于复查特定路径的逻辑层次分布。此命令支持您判断为改进时序 QoR 而需要再平衡的路径数量。

您可使用 Vivado 综合中提供的 `retiming_forward` 和 `retiming_backward` 属性来控制特定寄存器或路径上的最优化。使用这些属性会对一组特定路径（而不是顶层模块或子模块）应用重定时最优化，从而减少面积开销。您可在 RTL 或 XDC 文件中应用这些属性。如需获取更多信息，包括使用情况和限制相关信息，请参阅《Vivado Design Suite 用户指南：综合》(UG901)。

下图显示了 inst1/inst2 层级内 58 条具有 5 个逻辑层次并受 600 MHz 时钟约束的路径以及 32 条仅含 1 个逻辑层次的路径。

图 23：**含默认综合最优化的逻辑层级分布**

```
current_instance inst1/inst2

report_design_analysis -timing -logic_level_distribution -of_timing_paths [get_timing_paths -max_paths 100 -group core_clk_600]

+------------------+-------------+---+----+----+---+---+----+---+---+---+---+----+-------+-------+-------+-------+-----+
|  End Point Clock | Requirement | 0 | 1  | 2  | 3 | 4 | 5  | 6 | 7 | 8 | 9 | 10 | 11-15 | 16-20 | 21-25 | 26-30 | 31+ |
+------------------+-------------+---+----+----+---+---+----+---+---+---+---+----+-------+-------+-------+-------+-----+
| core_clk_600     | 1.667ns     | 0 | 32 | 10 | 0 | 0 | 58 | 0 | 0 | 0 | 0 |  0 |     0 |     0 |     0 |     0 |   0 |
+------------------+-------------+---+----+----+---+---+----+---+---+---+---+----+-------+-------+-------+-------+-----+
```

Vivado 综合可通过将低逻辑层次路径内的寄存器迁移到高逻辑层次路径中来对逻辑层次进行再平衡。在此示例中，您可向综合 XDC 文件添加以下约束，以便在 inst1/inst2 层级上执行重定时：

```
set_property BLOCK_SYNTH.RETIMING 1 [get_cells inst1/inst2]
```

使用相同全局设置和经过更新的 XDC 文件重新运行综合后，可在 inst1/inst2 时序路径上运行定时时序分析，或者重新运行 `report_design_analysis` 命令以便验证是否最长的路径所含逻辑层次较少，如下图所示。关键路径当前为“REG0” → “3 LUTs” → “REG2”（向后重定时），而从 REG2 到 REG4 的路径具有 3 个逻辑层次。

图 24：**针对综合最优化启用重定时的逻辑层次分布**

```
current_instance inst1/inst2

report_design_analysis -timing -logic_level_distribution -of_timing_paths [get_timing_paths -max_paths 100 -group core_clk_600]

+------------------+-------------+---+---+---+-----+---+---+---+---+---+---+----+-------+-------+-------+-------+-----+
|  End Point Clock | Requirement | 0 | 1 | 2 |  3  | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11-15 | 16-20 | 21-25 | 26-30 | 31+ |
+------------------+-------------+---+---+---+-----+---+---+---+---+---+---+----+-------+-------+-------+-------+-----+
| core_clk_600     | 1.667ns     | 0 | 0 | 0 | 100 | 0 | 0 | 0 | 0 | 0 | 0 |  0 |     0 |     0 |     0 |     0 |   0 |
+------------------+-------------+---+---+---+-----+---+---+---+---+---+---+----+-------+-------+-------+-------+-----+
```

减少控制集

注释： `report_qor_suggestions` Tcl 命令可自动应用这种最优化技巧。

复位或时钟使能等控制信号经常会遭到忽视。很多设计人员在进行 HDL 编码时都以“if reset”语句作为开始，而不考虑是否需要复位。虽然所有寄存器都支持复位和时钟使能，但其使用会从最高时钟频率、使用率和功耗方面对最终实现产生显著影响。

首先需要考量的因素即控制集的数量。控制集是时序单元使用的时钟信号、使能信号和置位/复位信号的组合。例如，如果 2 个单元连接到同一个时钟，但只有其中 1 个单元具有复位或者只有其中 1 个单元具有时钟使能，这 2 个单元的控制集就不同。固定不变或不使用的使能和置位/复位寄存器管脚也会构成控制集。

第二个需要考量的因素是目标架构。能封装到一起的控制集数量取决于架构。每个 Versal 器件半 slice 都包含两组寄存器，每组 4 个，并共享同一个时钟和同一个置位/复位。此外，每组 4 个寄存器包含 1 个时钟使能，可忽略置位/复位。从用于馈送 CE 管脚的互连控制多路复用器可提供恒定的逻辑 1 时钟使能。

由控制集导致的 CLB 封装限制会迫使布局器移动部分寄存器，包括其输入 LUT。在某些情况下，寄存器将迁移至非理想位置。由于逻辑扩散（信号线延迟更长）和互连资源使用率提高，距离的增加不仅会对使用率产生不利影响，而且还会对布局 QoR 和功耗产生负面影响。对于含大量低扇出控制信号（例如馈送单寄存器的时钟使能信号）的设计，这是需要考量的主要问题。

虽然 Versal 器件 CLB 所含资源达 UltraScale 器件所含资源的四倍，但通过比较 Versal 器件半 slice 与 UltraScale 器件半 CLB 之间的控制集发现两者所含资源相近。因此，对于这两种架构，AMD 给予相同的建议。

注释：欲知详情，请访问此链接以参阅《Versal 自适应 SoC 硬件、IP 和平台开发方法指南》(UG1387) 中的相应内容。

遵循控制集指南进行操作

下表提供了根据目标器件大小建议的控制集数量指南。

表 12：**控制集指南**

指南	控制集百分比
可接受	低于器件中控制集总数的 7.5%
建议减少数量	介于器件控制集总数的 7.5% 到 15% 之间
必须减少数量	超过器件中控制集总数的 15%

上述指南假定：

- 典型的控制集容量：每 8 个 CLB 寄存器对应 1 个控制集
- 器件中控制集总数：CLB 寄存器数量/8

使用 `report_control_sets -verbose` 确定设计中的控制集数量。

提示：唯一控制集的数量在小部分设计中可能会引发问题，导致对应器件区域中信号线延迟过长或者拥塞。要识别唯一控制集的高局部密度，需要在 Vivado IDE 的“Device”（器件）窗口中进行详细的布局分析，这包括以不同颜色高亮的控制信号。

减少控制集的数量

如果控制集的数量过高，请使用以下任一策略来减少其数量：

- 移除 HDL 源或约束文件中的控制信号上设置的 MAX_FANOUT 属性。复制控制信号会显著增加独立控制集的数量。AMD 建议采用 `place_design` 来执行大颗粒度复制，并在布局器后使用 `phys_opt_design -directive Explore` 来实现更精细的复制。这样即可避免非必要的复制以及等效控制集彼此交汇而导致布线拥塞。
- 增大 Vivado 综合（或其他综合工具）的控制集阈值。复查 `report_control_sets -verbose` 中的控制集扇出分布表，以判定更合适在综合期间使用的控制集阈值。请注意，增大 `contol_set_opt` 可能对功耗产生负面影响，因为这样会消除可积极降低功耗的时钟使能。例如：

```
synth_design -control_set_opt_threshold 16
```

  提示：使用 BLOCK_SYNTH 综合约束来更改受布局扩散或拥塞影响最大的模块上的控制集阈值。

- 在综合后，使用 `opt_design -control_set_merge` 或 `opt_design -merge_equivalent_drivers` 来合并等效控制集。
- 使用 CONTROL_SET_REMAP 属性将驱动同步置位/复位和/或寄存器的 CE 管脚的低扇出控制信号映射到 D 输入。欲知详情，请访问此链接以参阅《Vivado Design Suite 用户指南：实现》(UG904) 中的相应内容。
- 请避免低扇出异步置位/复位（预置/清除），因为它们只能连接到专用异步管脚，而无法通过综合迁移到数据路径。因此，综合控制集阈值选项不适用于异步置位/复位。
- 请避免针对不同时序单元同时使用高电平有效和低电平有效控制信号。
- 仅在必要时使用时钟使能和置位/复位。通常，数据路径包含大量寄存器，这些寄存器会自动刷新未初始化的值，其中仅限第一个阶段和最后一个阶段才需要置位/复位或使能信号。

注释：如需了解其他综合属性以及控制信号相关建议，请访问此链接以参阅《Versal 自适应 SoC 硬件、IP 和平台开发方法指南》(UG1387) 中的相关内容。

最优化高扇出信号线

高扇出信号线常常导致设计实现问题。由于裸片尺寸随每个器件系列不断增加，扇出问题也越来越多。具有数千个端点的信号线上的时序通常难以得到满足，如果在路径上存在附加逻辑，或者如果这些信号线是由非时序单元（例如 LUT 或分布式 RAM）驱动的，则尤其如此。

允许寄存器复制

大部分工具均可复制寄存器来减小关键路径上的高扇出信号线。或者，也可以在特定寄存器或层级上应用属性以指定哪些寄存器可复制或不可复制。例如，复制的信号线上存在 LUT1 即可表明某个属性或常量正在部分阻止最优化。综合期间，可通过在已最优化的信号线所遍历的层级单元上使用 KEEP_HIERARCHY 属性或者不同层级内的信号线段上使用 KEEP 属性来更改复制最优化。综合与实现期间，DONT_TOUCH 约束始终都会阻止有益的复制。

有时，设计师通过在特定信号线上使用 MAX_FANOUT 属性来解决 RTL 或综合中的高扇出信号线问题。这并不总是意味着能够以最优方式来利用布线资源，当 MAX_FANOUT 属性设置过低或者在连接到多个主层级的信号线上设置该属性时尤其如此。此外，如果高扇出信号是寄存器控制信号，并且复制次数多于必要次数，则可能导致控制集数量过多，且因添加时序收敛不需要的额外寄存器而导致增加设计功耗。

通常，要减少扇出，最好对高扇出信号使用平衡树。可考虑根据设计层级手动复制寄存器，因为层级中包含的单元通常布局在一起。

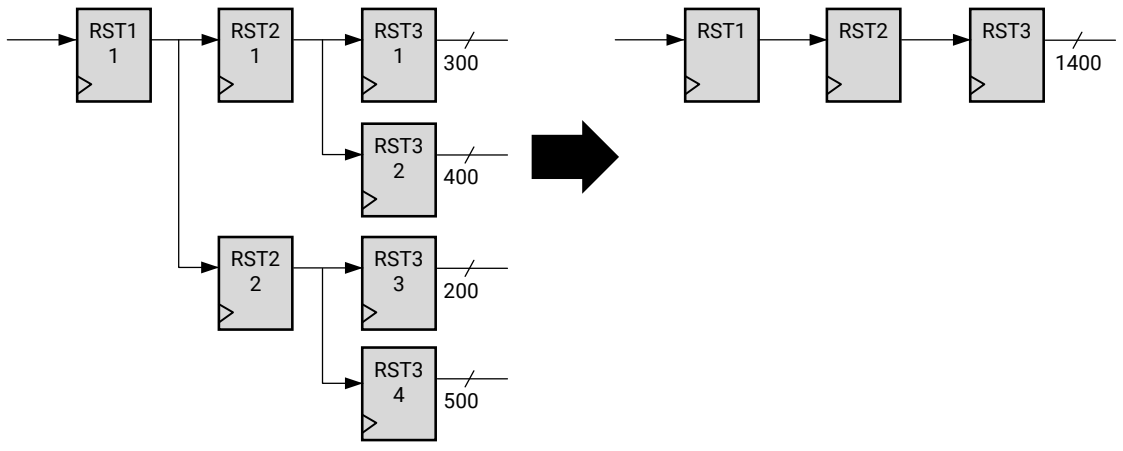
如需对控制集树和高扇出信号线进行重构并减少其数量，可使用 `opt_design` Tcl 命令并搭配以下任一选项：

- `-control_set_merge`：该选项用于将逻辑等效控制信号的驱动程序主动组合为单一驱动程序。
- `-merge_equivalent_drivers`：该选项用于将逻辑等效的信号（包括控制信号）合并为单一驱动程序。

  注释：请首先尝试该选项，因为运行该选项时，工具将检测到主层级和 Pblock 约束。

这些选项是扇出复制的反向操作，可使信号线更适用于基于模块的复制。此合并也适用于多阶段复位树中的各阶段，如下图所示。

图 25：**使用 opt_design -control_set_merge 合并的控制集**

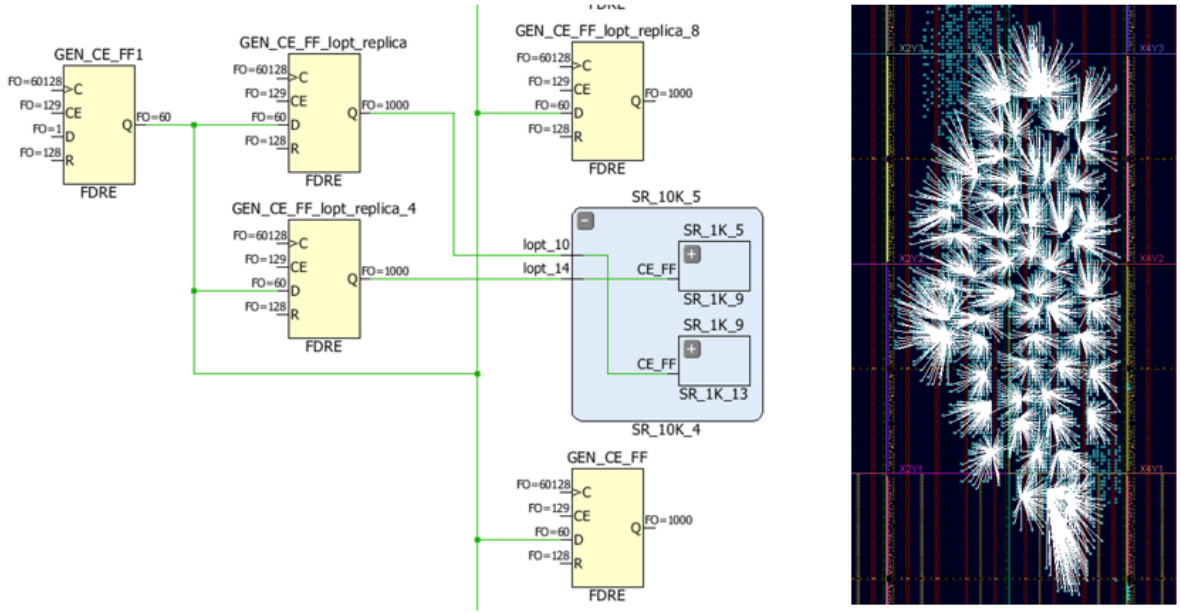


减少复制的对象数量后，即可使用 `opt_design` Tcl 命令基于层级特性执行有限复制并搭配以下选项：

- `-hier_fanout_limit <arg>`：该选项根据层级复制寄存器，其中，`<arg>` 表示复制的扇出限制（取决于逻辑层级）。对于由高扇出信号线驱动的每个层级实例，如果层级中的扇出高于指定限制，那么该层级中的信号线由高扇出信号线的驱动程序副本来驱动。复制的驱动程序与原始驱动程序布局在相同层级内，并且复制并不限于控制集寄存器。

下图显示了使用 `opt_design -hier_fanout_limit 1000` 执行的时钟使能信号线复制，此时钟使能信号线的扇出为 60000。因为每个模块 SR_1K 包含 1000 个负载，驱动程序将复制 59 次。

图 26：**高扇出时钟使能信号线上基于模块的复制**



默认情况下，在 `place_design` 中启用扇出最优化。复制发生在布局器流程早期阶段，并且基于布局信息执行。驱动超过 1000 个负载的寄存器以及驱动 DSP、块 RAM 和 UltraRAM 的寄存器将纳入复制考量范围，如果发生复制，这两种寄存器将与负载放置在一起。您可通过在信号线中添加 FORCE_MAX_FANOUT 属性来强制复制驱动信号线的寄存器或 LUT。FORCE_MAX_FANOUT 的值用于指定完成复制最优化后信号线的最大物理扇出。

您可基于 MAX_FANOUT_MODE 属性的物理器件属性来执行强制复制。受支持的 MAX_FANOUT_MODE 属性为 CLOCK_REGION、SLR 和 MACRO。例如，当 MAX_FANOUT_MODE 属性的值为 CLOCK_REGION 时，即可基于物理时钟区域来复制驱动程序，并将布局到相同时钟区域内的负载聚集在一起。欲知详情，请访问此链接以参阅《Vivado Design Suite 用户指南：实现》(UG904) 中的相应内容。

对于 SSI 技术器件，可针对每个 SLR 复制高扇出驱动程序，并可选择将这些驱动程序与其负载一起分配到符合 SLR 的 Pblock。此技巧有助于降低 SLR 交汇延迟的影响，并支持自由选择将复制的高扇出信号线独立布局在每个 SLR 中。

将高扇出信号线推广到全局布线

注释： `report_qor_suggestions` Tcl 命令可自动应用这种最优化技巧。

通过在驱动程序与负载之间插入时钟缓冲器，可以将较慢的时钟域中的高扇出信号线移动到全局布线上。仅当已使用的时钟缓冲器数量已达上限，并且所含扇出大于 25000 的信号线所驱动的逻辑的时钟周期高于目标器件和速度等级的特定限值时，才会在 `opt_design` 中为该信号线自动执行此最优化。为支持 Versal 器件中的全局布线上的高扇出信号线，BUFG_FABRIC 单元可布局在整个器件上的各 NoC 列中普遍存在的 BUFG_FABRIC 站点 (site) 上。

在 RTL 文件或约束文件 (XDC) 中的信号线上设置 CLOCK_BUFFER_TYPE 属性时，可强制实施 `synth_design` 和 `opt_design` 以插入时钟缓冲器。例如：

```
set_property CLOCK_BUFFER_TYPE BUFG_FABRIC [get_nets netName]
```

使用全局时钟可确保最佳布线，但代价是更高的信号线延迟。为了获得最优布线延迟，时钟缓冲器必须直接驱动时序负载，无中间组合逻辑。在大多数情况下，opt_design 会将非时序负载并行重新连接到时钟缓冲器。如果需要，您可以通过在时钟缓冲器输出信号线上应用 DONT_TOUCH 来防止进行此最优化。此外，如果高扇出信号线是控制信号，您必须确定为何某些负载不属于专用时钟使能或置位/复位管脚。

注释：请访问此链接并参阅《Versal 自适应 SoC 硬件、IP 和平台开发方法指南》(UG1387)，根据其中描述复查如何使用专用综合属性来控制局部时钟使能和置位/复位最优化。

完成时钟布线后，布局器还能在任何可用的全局布线轨道上自动执行高扇出信号线（扇出 > 10000）的布线。在布局器流程接近尾声时执行此最优化，并且仅在时序不发生劣化的情况下执行。可使用 -no_bufg_opt 选项禁用此功能。

使用物理最优化

物理最优化 (phys_opt_design) 可基于裕量和布局信息自动复制高扇出信号线驱动，通常可显著改善时序。AMD 建议您使用互连结构寄存器 (fabric register, FD*) 驱动高扇出信号线，这在物理最优化期间更便于复制和重新定位。

在某些情况下，默认 phys_opt_design 命令不会复制所有关键的高扇出信号线。请使用其他指令来发挥此命令的作用：Explore、AggressiveExplore 或 AggressiveFanoutOpt。此外，布线期间当高扇出信号线发挥关键作用时，您可添加 phys_opt_design 迭代，以在特定信号线上进行强制复制，然后再尝试重新进行设计布线。例如：

```
phys_opt_design -force_replication_on_nets [get_nets [list netA netB netC]]
```

使用 group_path 命令排列关键逻辑的优先顺序

您可使用含 -weight 选项的 group_path 命令来对时钟组中定义的路径端点给予更高的优先级。例如，要向采用特定时钟进行时钟设置的逻辑组分配更高的优先级，请使用以下命令：

```
group_path -name [get_clocks clock] -weight 2
```

在此示例中，实现工具将属于时钟组 clock 的路径的权重设置为 2，从而赋予其较之设计路径中其他路径更高的优先级。

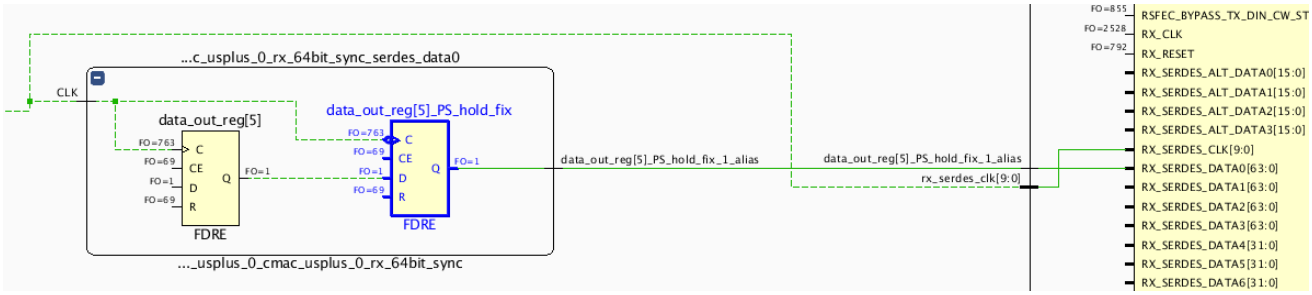
布线前先修复严重保持时间违例

对于存在严重保持时间违例 (> 0.4 ns) 的路径，最好在对设计进行布线前先减少保持时间违例，以便于布线器使用布线绕行来修复其余较轻微的保持时间违例。如果发现保持时间修复导致布线拥塞，那么在布线前减少保持时间违例是很有帮助的。每个 phys_opt_design 保持时间修复选项都使用不同资源，并具有特定目标。根据器件使用率和期望的影响来使用适当的选项至关重要。在运行 phys_opt_design 进行保持时间修复前，重要的是确认设计已包含正确约束的时钟树以便最大程度降低偏差。

在时序元件之间插入下降沿边缘触发寄存器可将时序路径分为 2 个半周期路径，从而显著减少保持违例。您可在 phys_opt_design 实现步骤中使用 -insert_negative_edge_ffs 选项插入下降沿边缘触发寄存器。仅当路径包含触发器驱动，并且时序元件之间最多仅含 1 个 LUT 时，才考虑对此路径执行此项最优化。路径上的建立时序裕量在最优化后必须为正值并且足够大，否则将放弃最优化。

下图显示了在驱动 CMAC 块的触发器后插入的下降沿边缘触发寄存器。执行最优化前，触发器与驱动之间的保持时序裕量为 -0.492 ns。插入下降沿边缘触发寄存器（以蓝色高亮显示）后，建立和保持时序裕量均为正值。

图 27：**通过插入下降沿寄存器修复保持时间违例问题**



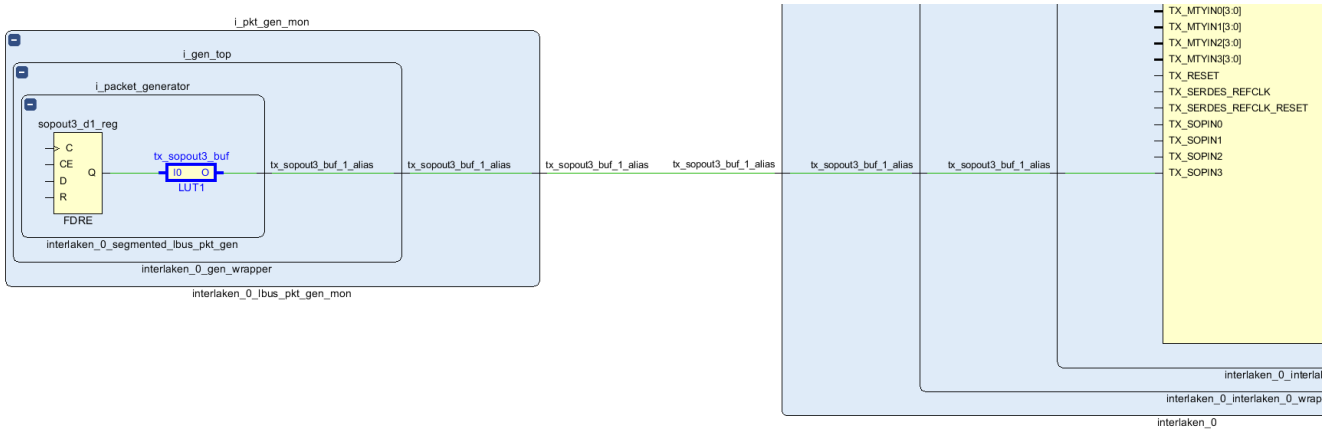
此外，还可将 LUT1 延迟插入数据路径以减少保持时间违例。要插入 LUT1 延迟，请在 `phys_opt_design` 实现步骤中使用以下选项之一：

- `-hold_fix`：执行 LUT1 插入，仅考量具有足够的正建立时序裕量的最大 WHS 违例路径。
- `-aggressive_hold_fix`：执行 LUT1 插入，此方式比标准 `-hold_fix` 选项更激进。`-aggressive_hold_fix` 最优化会考量是否对大量保持时间违例路径执行 LUT1 插入，可用于以牺牲 LUT 使用率为代价显著减少设计 THS。

  注释： `phys_opt_design -directive ExploreWithAggressiveHoldFix` 指令会将 `Explore` 指令与 `-aggressive_hold_fix` 作为单一最优化操作一起运行。

下图显示了在驱动 ILKN 块的触发器之后插入 LUT1 延迟的情况。最优化前，从触发器到 ILKN 的路径是设计中的 WHS 路径，其保持时序裕量为 -0.277 ns。插入 LUT1 延迟（以蓝色高亮显示）之后，保持时序裕量和建立时序裕量均仍为正值。

图 28：**通过插入 LUT1 延迟修复保持时间违例问题**



解决拥塞

有多种因素可能导致拥塞，这个问题较为复杂，并非总能找到直接的解决方案。`report_design_analysis` 拥塞报告可帮助您识别拥塞区域和拥塞窗口中包含的主要模块。有多种技巧可用于对拥塞区域中的模块进行最优化。`report_qor_suggestions` 可以自动解决导致拥塞的诸多问题。

提示：在尝试使用以下章节中所探讨的技巧解决拥塞之前，请确保您的约束简单清晰且符合 AMD 的时钟设置准则建议。保持时间过长（或保持时序裕量为负）和时钟不确定性过大都会导致布线器绕行，从而导致拥塞。重叠 Pblock 同样可能导致拥塞，请务必避免。

降低器件使用率

当多个互连结构资源使用率百分比过高（平均 > 75%）时，如果网表复杂性同样过高（高顶层连接、高 Rent 指数、高平均扇出），布局难度将骤增。高时钟频率设计还附带其他布局难题。在此类情况下，请重新评估设计功能并考虑移除不必要的模块，直至仅有 1 个或 2 个互连结构资源使用率百分比较高为止。如果逻辑缩位不可行，请查阅本章中所提供的其他拥塞缓解技巧。

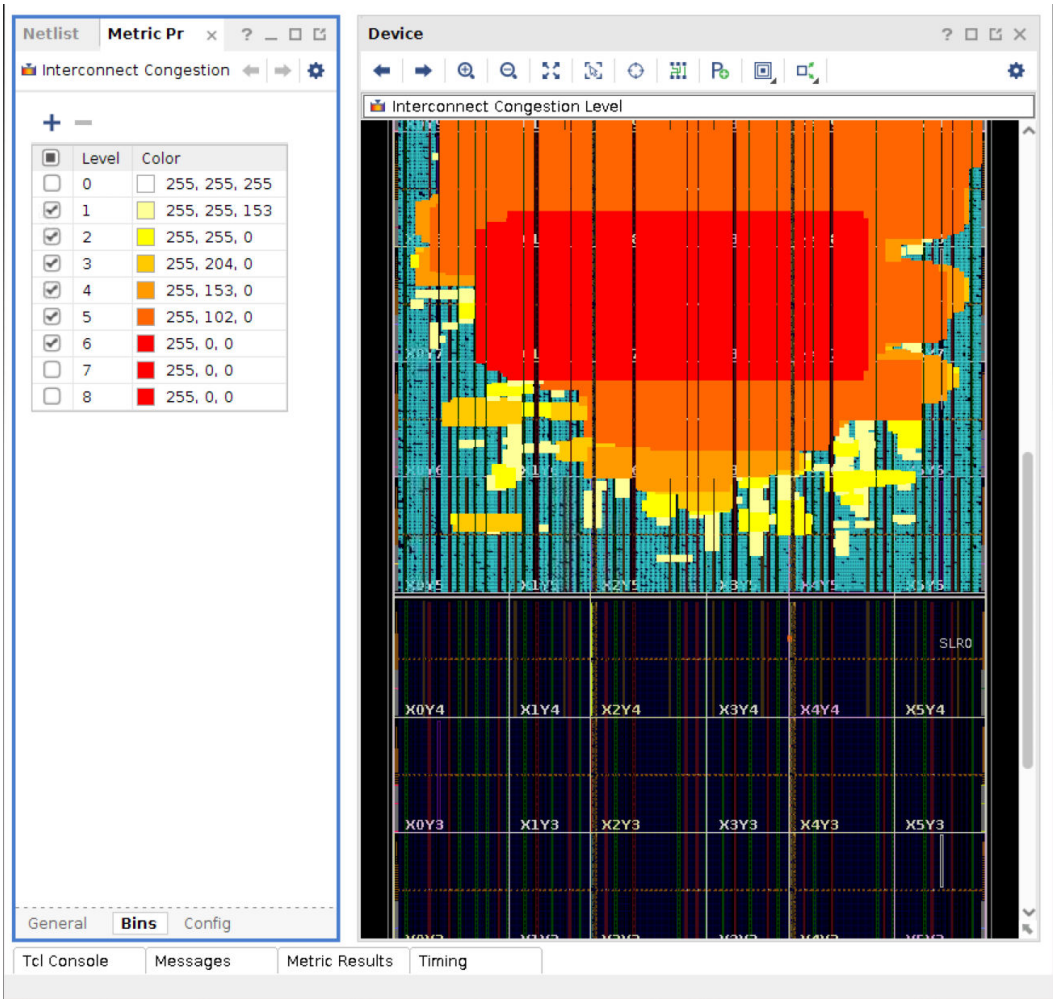
提示：执行 `opt_design` 后请复查资源使用率，以获取删除不使用的逻辑后（而不是综合后）的更准确的数值。

为 SSI 器件平衡 SLR 使用率

处理 SSI 技术器件时，分析每个 SLR 区域的使用率至关重要。总体使用率可能较低，但其中 1 个 SLR 中的使用率过高可能会导致拥塞。

在下图中，设计的整体使用率较低。然而，SLR2 中的使用率较高，并且其中逻辑所需的布线资源比其他 SLR 中的逻辑所需资源更多。该区域中的逻辑是宽总线 MUX，它会使总线资源饱和。

图 29：**每个 SLR 区域的使用率分析**



要平衡使用率，请尝试以下操作：

- 使用不同的布局器指令扩展设计。
- 使用布局规划约束（例如 Pblock）来避免某些模块使用率过高和 SLR 拥塞。

使用替代布局和布线指令

由于布局通常对总体设计的最高时钟频率影响最大，因此应用不同的布局器指令是用于尝试减少拥塞的首选技巧之一。您可考虑运行不含任何现有 Pblock 约束的替代布局器指令，以便布局器根据需要更加灵活地传播逻辑。

有多项布局器指令有助于缓解拥塞，这些指令通过在整个器件上传播逻辑来避免拥塞区域。SpreadLogic 布局器指令包括：

- AltSpreadLogic_high
- AltSpreadLogic_medium
- AltSpreadLogic_low

其他布局器指令或实现策略同样有助于缓解拥塞，使用上述布局器指令后也应一并尝试。

要比较不同布局器指令的拥塞情况，请在运行 `place_design` 后运行“Design Analysis Congestion”（设计分析拥塞）报告，或者检查布线器 log 日志文件中的初始估算拥塞。

相比布局器指令，布线对拥塞的影响较小。但在某些情况下，尝试不同布线指令同样很有用。以下指令可确保布线器尽力增加布线并缓解互连拼块中的拥塞：

- AlternateCLBRouting

欲知详情，请访问此链接以参阅《Vivado Design Suite 用户指南：实现》(UG904) 中的相应内容。

相关信息

拥塞等级范围

关闭跨边界最优化

禁止综合中的跨边界最优化可防止将其他逻辑拉入模块。这样即可降低模块的复杂性，但也会提高总体使用率。可通过 `synth_design` 中的 `-flatten_hierarchy none` 选项来全局执行此操作。同样的技巧也可应用于 RTL 中具有 KEEP_HIERARCHY 属性的特定模块。

禁用 LUT 组合

注释： `report_qor_suggestions` Tcl 命令可自动应用这种最优化技巧。

LUT 组合通过将成对的 LUT 与共享输入组合，形成同时使用 O5 和 O6 输出的单双输出 LUT 来降低逻辑使用率。但 LUT 组合可能会增加拥塞，因为它会增加 slice 的输入/输出连接。如果在拥塞区域内 LUT 组合比例较高 (> 40%)，您可尝试使用综合策略来消除 LUT 组合以帮助缓解拥塞。`Flow_AlternateRoutability` 综合策略和指令可指示综合工具停止生成任何其他 LUT 组合。

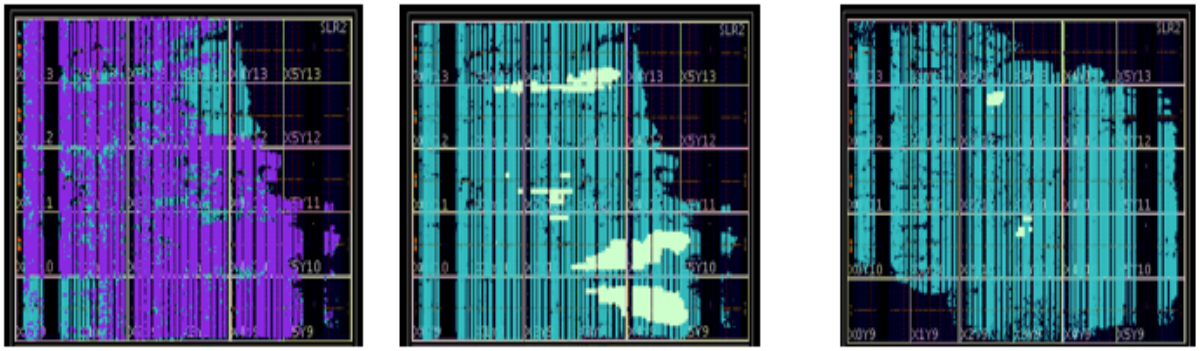
注释： 如果您使用 Synplify Pro 进行综合，那么您可以使用“Device”（器件）选项卡下“Implementation Options”（实现选项）中的“Enable Advanced LUT Combining”（启用高级 LUT 组合）选项。默认情况下，该选项处于已启用状态。如果您修改 Synplify Pro 工程文件 (`*prj`)，请指定以下内容：`set_option -enable_prepacking 1`。

您可以使用以下命令来选择设计中启用 LUT 组合的单元：

```
select_objects [get_cells -hier -filter {SOFT_HLUTNM != "" || HLUTNM != ""}]
```

下图显示了含和不含 LUT 组合的设计的水平拥塞。与 LUT 组合的单元以紫色高亮。

图 30：**LUT 组合对水平拥塞的影响**



含 LUT 组合的水平拥塞

不含 LUT 组合的水平拥塞

X18040-053022

要在与高拥塞区域重叠的模块上禁用 LUT 组合，请使用以下 Tcl 命令：

```
reset_property SOFT_HLUTNM [get_cells -hierarchical -filter {NAME =~
<module name> && SOFT_HLUTNM != ""}]
```

在拥塞区域中限制高扇出信号线

注释： `report_qor_suggestions` Tcl 命令可自动应用这种最优化技巧。

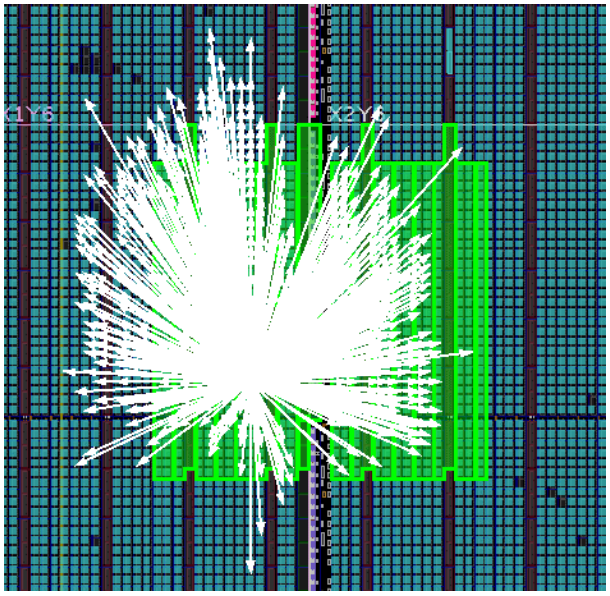
具有严格的时序约束的高扇出信号线要求对布局进行严格分群以满足时序。这可能导致局部拥塞，如下图所示。高扇出信号线还会因占用布线资源，导致拥塞窗口中其他信号线无法使用这些资源，从而促成拥塞。

要分析高扇出非全局信号线对拥塞窗口中可布线性的影响，请执行以下操作：

- 选中拥塞窗口中顶层层级模块中的叶节点单元。
- 使用 find 命令（“Edit” → “Find”）来选中已选单元对象的所有信号线，滤除“Global Clocks”（全局时钟）、“Power”（电源）和“Ground”（接地）信号线。
- 按“Flat Pin Count”（扁平管脚计数）降序对信号线进行排序。
- 选择扇出最高的几个信号线，显示其与拥塞窗口的关系。

这样可帮助您快速识别可能导致拥塞的高扇出信号线。

图 31：**拥塞窗口中的高扇出信号线**



对于拥塞窗口内具有严格时序约束的高扇出信号线，复制驱动将有助于放宽布局约束并缓解拥塞。

具有足够正时序裕量的高扇出信号线（扇出 > 5000）可在全局时钟资源（而不是互连结构资源）上进行布线。布局器会自动在全局布线资源上对扇出 > 1000 的高扇出信号线进行布线，前提是这些资源可用至布局器步骤结束。执行此最优化的前提是它不会造成时序劣化。

您还可在信号线上设置 CLOCK_BUFFER_TYPE=BUFG_FABRIC 属性，并交由综合或逻辑最优化在执行布局器步骤前自动插入缓冲器。完成 `place_design` 后复查新插入的缓冲器布局及其驱动和负载布局，以验证是否达成最优化。如果并未达成最优化，请在时钟缓冲器上使用 CLOCK_REGION 约束来控制其布局。

使用单元膨胀

您可在 `place_design` 步骤中使用单元膨胀来插入空白（增大单元间隔）。这样可以降低裸片给定区域的单元密度，从而通过增加可用布线来减少拥塞。此方法在较高频率的时钟域内狭小拥塞区域中特别有效。

要使用单元膨胀，请将 CELL_BLOAT_FACTOR 属性应用于层级单元，并将值设置为 LOW、MEDIUM 或 HIGH。处理数百个单元的小型模块时，建议将该值设置为 HIGH。

注意！如果器件已使用了大量布线资源，则不建议采用单元膨胀。此外，在大型单元上使用单元膨胀还可能导致已布局单元的彼此间距过大。

调节编译流程

可通过默认编译流程快速获得设计基线 (baseline)，并在未满足时序要求的情况下分析设计。在完成初步实现之后，可能需要调节编译流程以实现时序收敛。

使用策略和指令

您可以使用策略和指令来查找设计的最优解决方案。这些策略将全局应用于工程实现的运行。可针对“工程模式”和“非工程模式”下的实现流程中的每个步骤单独设置指令。

ML 策略

机器学习 (ML) 策略支持您快速获取最优化设计策略。如果您正在运行多项实现策略以生成实现结果，那么您可改为使用 ML 策略来帮助您预测哪些策略生成良好结果的可能性更高。

建议：AMD 建议使用不同策略建议执行 3 轮实现运行，以识别并解决预测中的错误。

您可在已布线的设计上运行 `report_qor_suggestions` 命令以生成策略建议对象。在运行此命令之前，必须按如下方式运行实现流程：

- 在“工程模式”下，使用“Default”或“PerformanceExplore”策略。
- 在“非工程模式”下，使用以下 Tcl 命令：
  - `opt_design`：将 `-directive` 选项设置为 `Default` 或 `Explore`。
  - `place_design`、`phys_opt_design` 和 `route_design`：将 `-directive` 选项设置为 `Default` 或 `Explore`。该选项必须在运行全部 3 条 Tcl 命令的过程中都相匹配。

生成 ML 策略建议后，必须使用 `write_qor_suggestions -strategy_dir <directory>` 写入建议。每项策略都会写入 1 个 RQS 文件。要激活策略对象，必须先使用 `read_qor_suggestions` 读取含策略建议的 RQS 文件，然后再运行 `opt_design`，并且所有命令的指令 (directive) 都必须设置为 RQS（例如，`opt_design -directive RQS`）。

AMD 建议使用 ML 时留意以下几个要点：

- 为了实现最佳结果，请解决所有方法论检查，确保设计的 QoR 评估得分不低于 3 分。请先运行 `synth_design` 或 `opt_design`，然后再运行 `report_qor_assessment` 以进行验证。
- 要进一步增强最高时钟频率，请将 ML 策略建议与相同 RQS 文件内的其他 QoR 建议相结合。

  注释：写入 QoR 建议时，会自动组合 ML 策略建议。要禁用此功能，请使用 `write_qor_suggestions -of_objects [get_qor_suggestions ...]`，并执行筛选，以仅显示所需的建议。

欲知详情，请访问此链接以参阅《Vivado Design Suite 用户指南：设计分析与收敛技巧》(UG906) 中的相应内容。

预定义策略

AMD 提供了一组预定义策略，这些策略经过精心调教，能够为大部分设计需求提供有效的解决方案。

定制策略

如果预定义策略无法满足时序要求，那么您可手动浏览定制指令组合。由于布局通常对设计可实现的时钟频率影响巨大，因此最好在仅含 I/O 位置约束而不含任何其他布局约束的情况下尝试各项布局器指令。通过复查每次布局器运行时的 WNS 和 TNS（这些值可在布局器 log 日志中找到），可选择 2 项或 3 项可提供最佳时序结果的指令作为下游实现流程的基础。

提示：要获取指令列表及其功能简介，请输入实现命令，后接 `-help` 选项（例如，`place_design -help`）。如需了解有关策略的信息，请访问此链接以参阅《Vivado Design Suite 用户指南：实现》(UG904) 中的相应内容。

对于其中每个检查点，可尝试运行多项 `phys_opt_design` 和 `route_design` 指令，并且同样仅保留含有最佳估算或最终 WNS/TNS 的运行结果。在“非工程模式”下，必须以 Tcl 脚本显式描述流程，并保存最佳检查点。在“工程模式”下，您可为每项布局器指令创建单独的实现运行，并启动运行直至布局步骤为止。在运行布局器步骤（由 Tcl-post 脚本判定）后，可针对具有最佳结果的运行轮次继续进行实现。

物理约束（Pblock 约束以及 DSP 和 RAM 宏约束）会妨碍布局器找出最佳解决方案。因此，AMD 建议您在不含任何 Pblock 约束的情况下运行布局器指令。开始使用指令进行布局前，可使用以下 Tcl 命令删除任意 Pblock：

```
delete_pblock [get_pblocks *]
```

运行 `place_design -directive <directive>` 并分析最佳结果的布局还可提供模板用于对设计进行布局规划或者复用块 RAM 宏或 DSP 宏的布局，这样可以稳定各轮次运行之间的流程。

使用最优化迭代

有时将 1 个命令反复循环多次有利于获得最佳结果。例如，最好首先运行含 `force_replication_on_nets` 选项的 `phys_opt_design` 以便对布线期间影响 WNS 的部分关键信号线进行最优化。

随后，运行含任意指令的 `phys_opt_design` 即可改进设计的总体 WNS。

在“非工程模式”下，使用以下命令：

```
phys_opt_design -force_replication_on_nets [get_nets -hier *phy_reset*]
phys_opt_design -directive <directive name>
```

在“工程模式”下，针对 `phys_opt_design` 运行步骤执行 Tcl-pre 脚本时，同时运行第一条 `phys_opt_design` 命令（此命令使用 `-directive` 选项来运行）即可实现相同的结果。

设计过约束

如果布线后设计未满足时序要求但相差不大，通常是因为布局后存在较小的时序裕度。可在布局和物理最优化过程中收紧时序要求，以增大布线器的时序预算。为此，AMD 建议使用 `set_clock_uncertainty` 约束，原因如下：

- 它不修改时钟关系（时钟波形保持不变）。
- 它是对工具计算的时钟不确定性的补充（抖动，相位误差）。
- 它专用于 `-from` 和 `-to` 选项指定的时钟域或时钟交汇。
- 通过对先前时钟不确定性约束应用 null 值，即可轻松将其复位。

在任何情况下，AMD 都建议您：

- 仅对无法满足建立时序的时钟或时钟交汇进行过约束。
- 仅将 `-setup` 选项用于收紧建立时间要求。

  注释：如果您不指定该选项，那么将收紧建立时间和保持时间要求。
- 运行布线器步骤前将附加的不确定性复位。

过约束示例

设计布线前后，在具有 `clk1` 时钟域的路径上与时序相差 -0.2 ns，在从 `clk2` 到 `clk3` 的路径上与时序相差 -0.3 ns。

1. 加载网表设计并应用标准约束。
2. 应用附加时钟不确定性以对特定时钟进行过约束。
   a. 数值应至少达到违例总量。
   b. 约束只能应用于建立路径。

```
set_clock_uncertainty -from clk0 -to clk1 0.3 -setup
set_clock_uncertainty -from clk2 -to clk3 0.4 -setup
```

3. 运行流程直到布线步骤为止。最好能够满足预布线时序要求。
4. 移除附加的不确定性。

```
set_clock_uncertainty -from clk0 -to clk1 0 -setup
set_clock_uncertainty -from clk2 -to clk3 0 -setup
```

5. 运行布线器。

在运行布线器后，您可以复查时序结果以评估过约束的优势。如果在布局后时序得到满足，但在布线后距离满足要求仍有差距，您可以增加不确定性的量并重试。

警告！过约束不得超过 0.5 ns。设计过约束可能因实现工具引入额外的逻辑复制而导致功耗增加，并增加编译时间。

提示：除过约束设计之外，另一种方法是更改每个路径组的相对优先级。默认情况下，在实现期间以相同优先级对每个时钟和用户定义路径组进行独立分析。您可使用 `group_path -weight 2 -name <ClockName>` 选项为任意基于时钟的路径组设置更高的优先级。不得更改用户定义的路径组的优先级。

布局规划注意事项

布局规划支持您引导工具完成高层次层级布局或详细布局。这样即可改进 QoR 并提供可预测性更高的结果。您可修复最严重的问题或者最常见的问题来最大程度改进结果。例如，如果存在离群路径并且这些路径的裕量明显极差或者具有高层次的逻辑，首先请通过 Pblock 将这些路径分组到同一个器件区域内以便对其进行修复。将布局规划局限于设计上需要额外用户干预的部分，而不是对整个设计进行布局规划。

将连接到 I/O 的逻辑布局于此 I/O 周围有时收效显著，具体表现在可通过多次编译提升可预测性。总之，最好将 Pblock 的大小保持在单一时钟区域内。这样可为布局器提供最大限度的灵活性。请避免重叠 Pblock，否则可能导致这些共享区域更加拥塞。如果 2 个 Pblock 之间存在大量连接信号，请考虑将其合并为单个 Pblock。请最大限度减少跨 Pblock 的信号线数量。

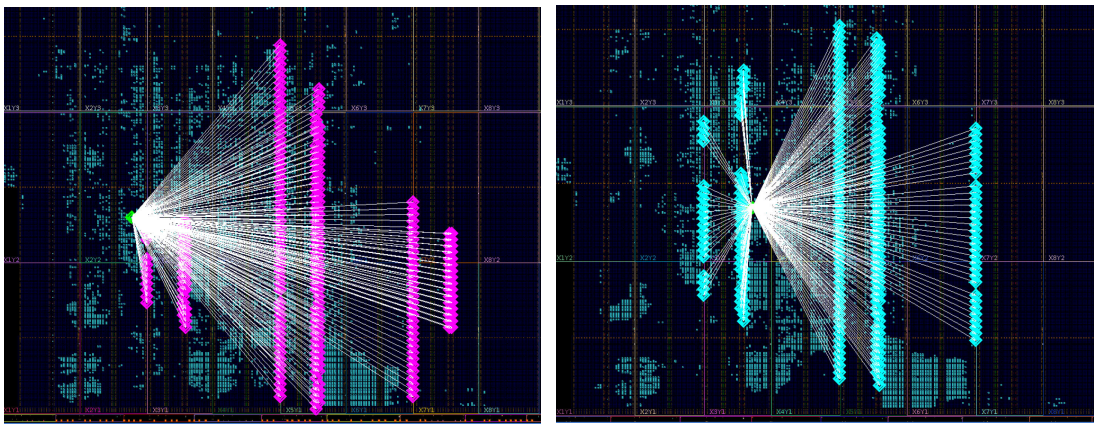
提示：升级到更高版本的 Vivado Design Suite 时，请首先尝试在不含 Pblock 或仅含最少量 Pblock（例如，仅含 SLR 级 Pblock）的情况下进行编译，以查看是否存在任何时序收敛困难。原先有助于提升 QoR 的 Pblock 可能会阻止布局和布线在新版本工具中寻找可能的最佳实现。

对关键逻辑进行分组

对关键逻辑进行分组可避免跨 SLR，这样有助于改进设计的关键路径。下图显示的两个示例是使用 512 个 RAMB36E5 原语实现的大型块 RAM。关键路径为从触发器到组中每个 RAMB36E5 的 ADDRARDADD* 管脚。

- 在左侧，此示例显示的是布局器无法找到路径的最优化布局，因为块 RAM 使用率过高。RAMB36E5 原语以粉色标记。
- 在右侧，此示例显示的是布局器能够满足时序，因为 RAMB36E5 块被组合在一起。RAMB36E5 原语以浅蓝色标记。

图 32：**关键逻辑分组**



复用布局结果

块 RAM 宏和 DSP 宏的布局十分便于复用。复用此布局有助于减少网表版本升级所导致的变化。这些原语通常具有稳定的名称。布局通常易于维护。部分布局指令生成的块 RAM 和 DSP 宏布局比其他布局更好。您可以尝试使用不同布局器指令将任一布局器运行所实现的宏布局改进应用于其他布局器的运行，从而改进 QoR。 以下简单 Tcl 脚本可用于将块 RAM 布局保存到 XDC 文件中。

```
set_property IS_LOC_FIXED 1 \
  [get_cells -hier -filter {PRIMITIVE_TYPE =~ BLOCKRAM.*.*}]
write_xdc bram_loc.xdc -exclude_timing
```

您可编辑 `bram_loc.xdc` 文件以仅保留块 RAM 位置约束，并将其应用于后续连续运行。

重要提示！请勿复用通用 slice 逻辑的布局。请勿复用设计中可能更改的部分的布局。如果对设计进行少量更改并且想要复用先前布局以提升结果可预测性和缩短编译时间，可使用增量编译流程。

使用增量实现

您可以使用增量实现来缩短实现编译时间，并提升结果的可预测性。AMD 建议将增量实现作为您的标准时序收敛策略的组成部分。欲知详情，请访问此链接以参阅《Vivado Design Suite 用户指南：实现》(UG904) 中的相应内容。

本章节涵盖了有关自动增量实现的建议。

选择高质量的参考检查点

由于增量实现流程取决于复用率，因此流程最重要的输入即参考检查点。在“工程模式”下使用自动增量实现时，Vivado 工具会管理参考检查点的更新。这样即可确保高复用率，也可确保接近时序收敛。

在增量实现流程的所有其他用例中，您可控制参考检查点的选择。以下准则有助于改进您选择参考检查点的方式：

- 使用满足时序或接近满足时序的参考检查点。如果参考检查点接近满足时序，那么运行增量实现流程前，它可能有助于改进时序，如下所示。

  注释：对于自动增量实现，除非 WNS 小于 -0.250 ns，否则检查点将被拒绝。

  - 运行 `route_design -tns_cleanup` 以对不属于最差情况路径的路径进行最优化。

  - 布线后运行 `phys_opt_design` 命令以改进不满足时序要求的情况。虽然此命令可能导致运行时间增加，但在增量实现运行中可快速重现这些最优化。
  - 使用 `report_qor_suggestions` 命令可生成设计改进建议。增量实现流程中应用的新建议必须适用增量实现。参考检查点中已应用的建议无需适用增量实现。对于不适用增量实现的建议，请考虑使用默认流程来应用建议和更新检查点。欲知详情，请访问此链接以参阅《Vivado Design Suite 用户指南：设计分析与收敛技巧》(UG906) 中的相应内容。
- 选择拥塞最低的检查点，此类检查点应用更改的速度比拥塞的检查点更快。
- 最大程度实现参考检查点与增量检查点之间的匹配。

  注释：对于自动增量实现，除非单元匹配率达到至少 94% 并且信号线匹配率达到至少 90%，否则检查点将遭拒绝。
  - 使用增量综合来减少因 RTL 更改而对网表进行的更改。在设计收敛周期中应尽早启用增量综合，而不是等到准备使用增量实现时才启用。
  - 请确保 `synth_design` 和 `opt_design` 选项与参考检查点和增量实现运行相匹配。
  - 匹配工具版本。虽然并非强制要求，但阈值更改和添加新的最优化可能导致匹配率降低。
  - 请避免使用 `opt_design AddRemap` 和 `ExploreWithRemap` 指令，除非只能通过这两项指令实现时序收敛。更改代码库时，这些指令导致命名一致性降低。
- 请使用 `report_qor_assessment` 来确定设计是否已准备好运行增量实现流程，以及是否适合从默认流程切换为使用增量实现流程。

提示：要调整增量实现阈值，请运行 `config_implementation -help` 以获取信息。要识别参考检查点和增量检查点之间的差异，请运行 `report_incremental_reuse`。

为高复用模式选择增量实现指令

您可使用指令调整增量实现流程行为。工具遵循这些指令在运行实现时使用增量实现算法。当流程还原到默认算法时，工具遵循使用 `place_design`、`phys_opt_design` 和 `route_design` 命令指定的指令进行操作。

以下是可用于增量实现流程的指令：

- RuntimeOptimized：用于来自参考检查点的 WNS。这有助于保持与参考检查点的一致性，并且至少将布局器和布线器运行时间减半（效率倍增）。如果参考检查点不收敛时序，那么该指令不会尝试收敛时序。该指令为默认指令。
- TimingClosure：用于 WNS = 0.000 ns。如果参考运行非常接近满足时序收敛，并且您愿意牺牲结果一致性和运行时间以换取尽力满足时序要求，则可使用此指令。该模式可在高难度设计上将 WNS 提升到多达 250 ps。将该指令与 QoR 建议配合使用即可最大限度提升收敛时序的可能性。该指令一般存在运行时间命中。
- Quick：该选项用于复用率大于 99% 并可轻松满足时序约束要求的设计。通常情况下，该选项适用于仅有少量不影响时序的更改的 ASIC 仿真和原型设计。

以下是适用于工程模式的命令示例：

```
set_property -name INCREMENTAL_CHECKPOINT.MORE_OPTIONS -value {-directive
TimingClosure} -object [get_runs <runName>]
```

以下是适用于非工程模式的命令示例：

```
read_checkpoint -incremental -directive TimingClosure <reference>.dcp
```

注释：`RuntimeOptimized` 指令可替代 `Default` 映射指令，而 `TimingClosure` 指令可替代先前版本的 Vivado Design Suite 中的 `Explore` 映射指令。

避免布局规划和过约束

使用增量实现流程时，请务必避免：

- 对增量实现运行进行布局规划。

  以参考检查点布局覆盖 Pblock 布局。

- 对布局器进行过约束。

  在增量实现运行中对设计进行过约束可能严重影响复用，因为工具会尝试满足人工修改的目标 WNS。要避免过约束，请选择下列两种方法之一：

  - 修改应用的约束，移除布局器上的任何过约束。
  - 对增量运行应用以下命令。

    注释：此命令不影响非增量运行，但忽略所有用户应用的时钟不确定性约束。

    ```
    config_implementation { {incr.ignore_user_clock_uncertainty 1}}
    ```

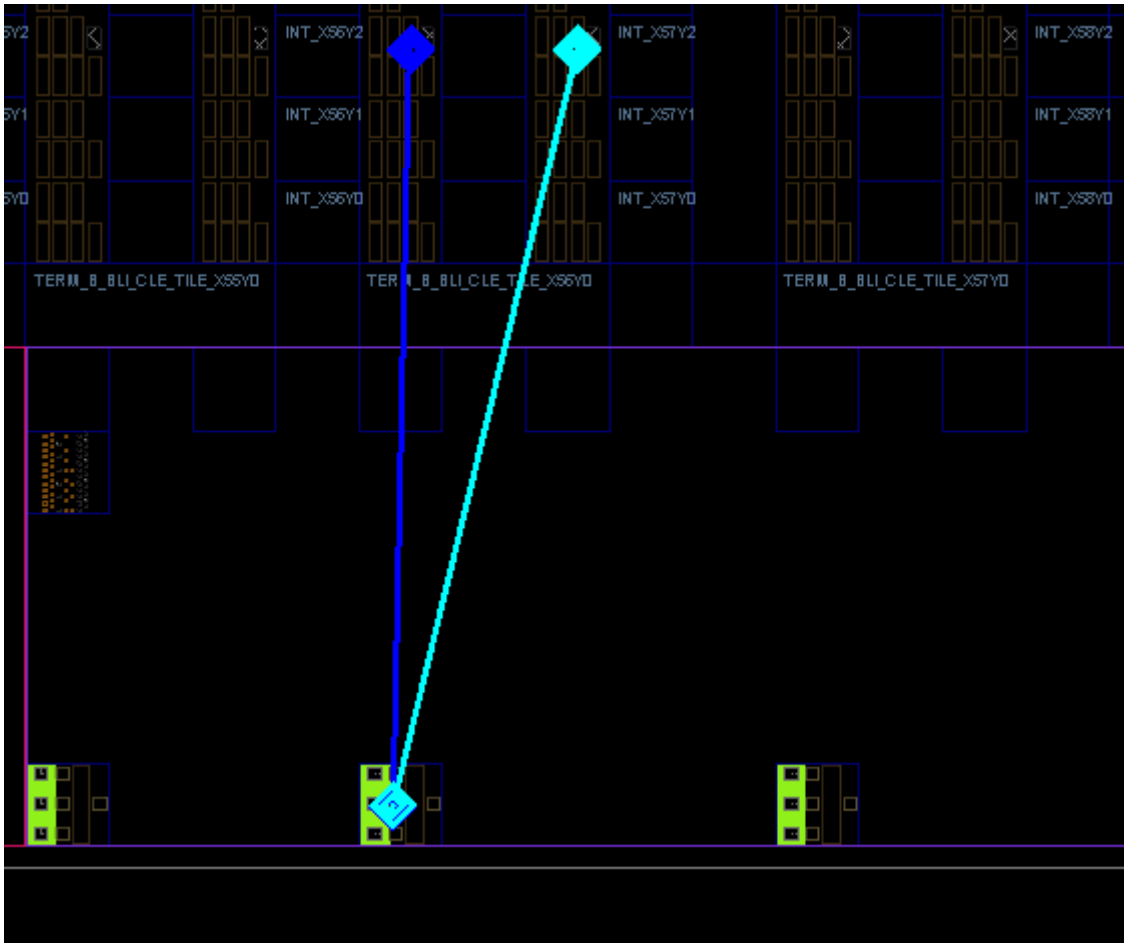
XPIO-PL 接口时序约束方法

边界逻辑接口触发器存在于硬件中的 XPIO 与可编程逻辑 (PL) 接口之间，可供您用于改进时序。XPIO 中的专用块（如 XPHY 逻辑、I/O 逻辑和时钟修改块等）具有边界逻辑接口触发器。您可对设计中的触发器应用边界逻辑接口 (BLI) 约束以便在设计实现期间自动利用此硬件功能。在此示例中，XPIO 中往来 I/O 逻辑单元 ODDRE1 和 IDDRE1 的数据路径使用的正是 BLI 触发器。

```
set_property BLI TRUE [get_cells {oddr_D1_BLI_reg oddr_D2_BLI_reg}]
set_property BLI TRUE [get_cells {iddr_Q1_BLI_reg iddr_Q2_BLI_reg}]]
```

下图显示了通过将 BLI 属性设置为 TRUE 所生成的布局和连接。

图 33：**XPIO-PL 接口中 ODDRE1 和 IDDRE1 的 BLI 触发器布局**



AI 引擎 PL 接口时序约束方法

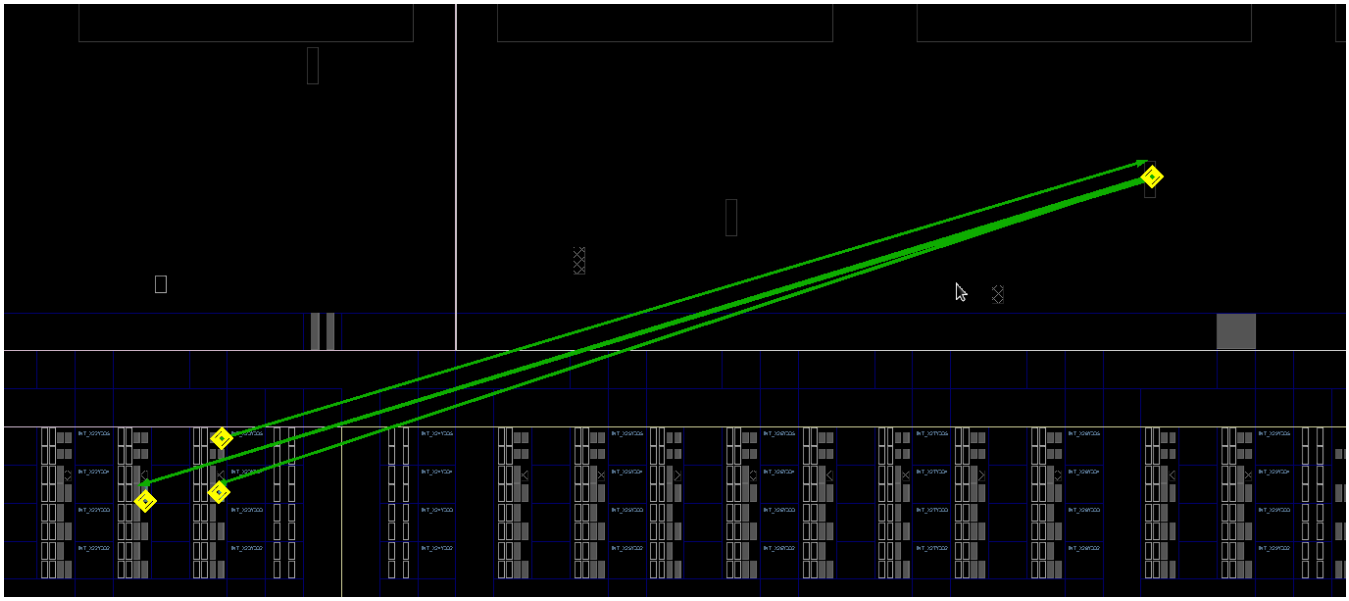
边界逻辑接口触发器存在于硬件中的 AI 引擎可编程逻辑 (PL) 接口中，可供您用于改进时序。您可对设计中的触发器应用边界逻辑接口 (BLI) 约束以便在设计实现期间自动利用此硬件功能。在此示例中，AI 引擎 (AXI_PL_M_AXIS64) 的 AXI4-Stream 接口与触发器驱动的互连结构之间存在双向往来连接。以下是 BLI 约束示例：

```
set_property BLI TRUE [get_cells m_axis_tdata_reg[*]]
```

```
set_property BLI TRUE [get_cells m_axis_*_reg]
```

下图显示了通过将 BLI 属性设置为 TRUE 所生成的布局和连接。

图 34：**AI 引擎 PL 接口 BLI 触发器的布局**



对于时序关键设计，启用 BLI 寄存器有助于实现最高时钟频率。为了控制跨 AI 引擎 PL 通道的 BLI 寄存器的推断，请使用以下 AI 引擎编译器选项：

- `--pl-freq=<number>`

  指定 PL 内核的时钟频率（以 MHz 为单位）。默认值为 AI 引擎核频率的 1/4，并且因速度等级而异。

  示例如下：

  - 所有 AI 引擎 PL 接口的 AI 引擎 PL 频率均为 300 MHz：

    ```
    --pl-freq=300
    ```

  - 不同接口的 AI 引擎 PL 频率不同。每个接口都与不同 AI 引擎计算图 PLIO 关联。约束必须引用 PLIO 名称，而不是 PL 内核名称：

    ```
    --pl-freq=plio_user_port_0:153 -pl-freq= plio_user_port_0:307.2
    ```

- `--pl-register-threshold=<number>`

  为寄存的 AI 引擎 PL 交汇指定频率阈值（以 MHz 为单位）。默认值为 AI 引擎频率的 1/8（基于速度等级）。

  下面给出 1 个示例：

  - ```
    -pl-register-threshold=125
    ```

    所含 AI 引擎 PL 频率高于此设置（在此例中为 125 MHz）的任意 PLIO 均映射到启用 BLI 寄存器的高速通道。否则，可使用任意 AI 引擎 PL 通道。

SSI 技术注意事项

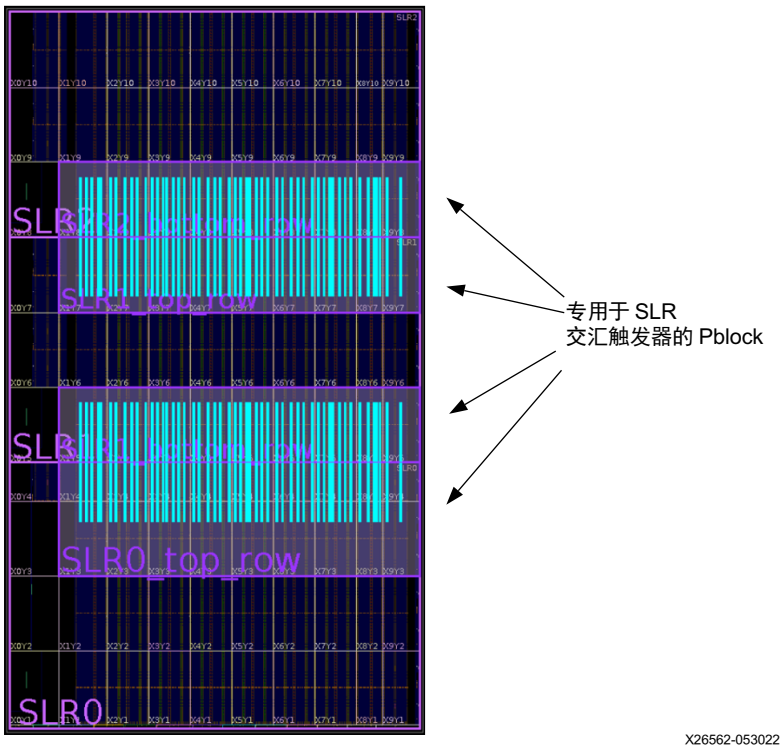
堆叠硅片互联 (SSI) 技术器件包含多个由中介层连接的超级逻辑区域 (SLR)。中介层连接被称为超长线路 (SLL)。当跨 SLR 交汇时会发生延迟惩罚。为了尽量降低 SLL 延迟对设计的影响，请对设计进行布局规划以避免在关键路径内包含 SLR 交汇。通过布局规划来最大限度减少 SLR 交汇，使遇到问题的模块仅保持在单一 SLR 内，这样也可以改进以 SSI 技术器件为目标的设计的时序和可布线性。
    ```

使用硬核 SLR 布局规划约束

对于高目标时钟频率设计，需在主层级之间建立足够的流水打拍以使全局布局和 SLR 分区轻松易行。对于极富挑战的设计，SLR 交汇点可能随运行而发生变化。除了定义 SLR Pblock 外，您还可在 SLR 边界处创建其他与时钟区域对齐的 Pblock 来约束交汇触发器。以下示例显示了含 SLL 拼块 (tile) 的 Versal xcvp1702 SSI 器件，这些 SLL tile 用于驱动 SLR 交汇节点或者由 SLR 交汇节点驱动，这些 SLR 交汇节点以蓝绿色高亮并带有下列 Pblock：

- 3 个 SLR Pblock：SLR0、SLR1 和 SLR2
- 4 个 SLR 交汇 Pblock：SLR0_top_row、SLR1_bottom_row、SLR1_top_row 和 SLR2_bottom_row

图 35：**SLR 交汇 Pblock 示例**



重要提示！AMD 建议使用 SLR 交汇 Pblock 的 CLOCKREGION 范围。

提示：您可以通过指定完整的 SLR 来定义 SLR Pblock。例如，`resize_pblock pblock_SLR0 -add SLR0`。

欲知详情，请访问此链接以参阅《Vivado Design Suite 用户指南：设计分析与收敛技巧》(UG906) 中的相应内容。

视频：如需了解有关使用布局规划技巧来解决性能问题的信息，请观看《Vivado Design Suite QuickTake 视频：设计分析与布局规划》。

使用软核 SLR 布局规划约束

对于大型设计，大部分主要块的逻辑都可按期望方式填充到 1 个 SLR 内，并在数次设计迭代后实现时序收敛。但根据总体设计布局，有少量逻辑（尤其是主块之间的连接和 SLR 之间的连接）会受到 QoR 变化的影响。在此类情况下，布局器和物理最优化算法需要更高的灵活性，才能将部分逻辑复制或迁移到其他 SLR，以解决布局难题并实现时序收敛。

您可使用 USER_SLR_ASSIGNMENT 属性向 SLR 分配大型设计块来完成设计布局规划。将该属性设置为字符串值，该字符串值将应用于层级单元，但不应用于叶节点单元。您为该属性设置的值会对逻辑分区产生如下影响：

- SLR 名称：为层级的单元分配 SLR（SLR0、SLR1、SLR2 等）名称时，布局器会尝试将整个单元布局在指定 SLR 内。
- 字符串值：为层级单元分配任意字符串值时，布局器会选择 SLR。这将阻止将单元分区到多个 SLR 内。

  注释：如有多个单元具有相同的 USER_SLR_ASSIGNMENT 值，那么布局器会尝试将这些单元分组到同一个 SLR 中。

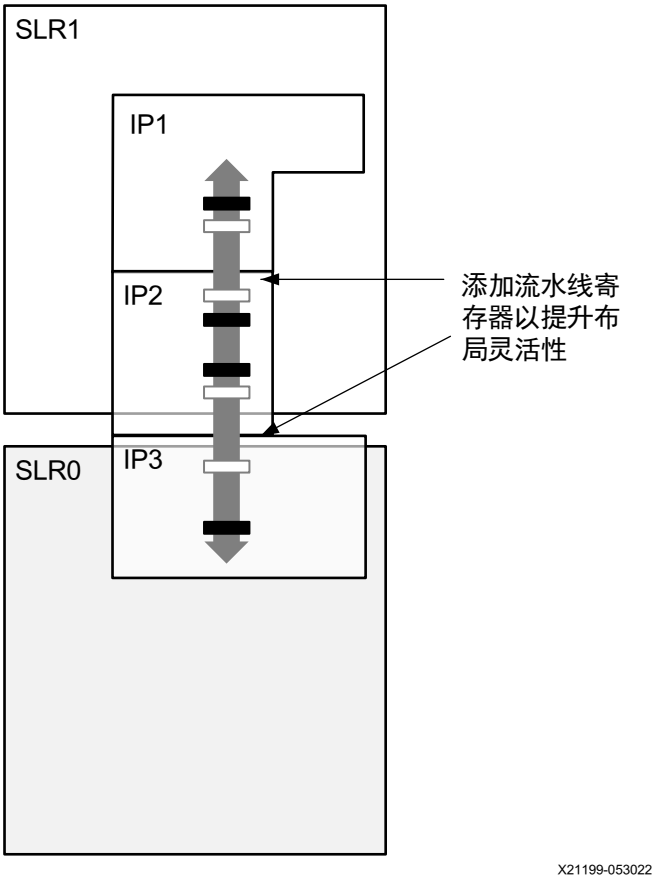
USER_SLR_ASSIGNMENT 属性可作为 SLR 分区期间的软核约束，而 Pblock 则始终作为 SLR 分区与全局布局期间的硬核约束。不同于 Pblock，布局器查找设计的有效 SLR 分区时可忽略 USER_SLR_ASSIGNMENT。
USER_SLR_ASSIGNMENT 和 Pblock 都支持通过详细的布局器和物理最优化来对 SLR 边界附近的叶节点单元布局进行微调以改进时序。这些调整操作包括跨 SLR 边界迁移流水线寄存器，前提是迁移有助于改进时序。不允许跨 Pblock 边界迁移这些寄存器。

在以下示例中包含 3 个时序关键型层级块，单元名称分别为 IP1、IP2 和 IP3，目标为双 SLR 器件。为了拆分这 2 个块以便将 IP1 和 IP2 一起保留在 SLR1 内，而将 IP3 布局在 SLR0 内，请应用以下 XDC 约束：

```
set_property USER_SLR_ASSIGNMENT SLR1 [get_cells {IP1 IP2}]
set_property USER_SLR_ASSIGNMENT SLR0 [get_cells IP3]
```

下图显示了由此生成的布局。为了提高性能，您可整合额外的流水线阶段以遍历器件内的距离。这在期望的 SLR 交汇处（在本例中即 IP2 与 IP3 之间）尤为实用。在执行详细布局和 `phys_opt_design` 期间，来自 IP2 和 IP3 的流水线寄存器可以自动跨 SLR 边界迁移，前提是迁移可改进时序。

图 36：**USER_SLR_ASSIGNMENT 属性的布局示例**



如果无法设置 USER_SLR_ASSIGNMENT 或者如果布局器将布局困难的路径布局在跨 SLR 位置导致出现路径分割，那么可使用 USER_CROSSING_SLR 属性来指示 SLR 交汇应出现的位置和不应出现的位置。通常，该属性可应用于符合以下条件的信号线或叶管脚：其中管脚与信号线驱动程序布局在相同 SLR 内，或者对寄存器链应用 SLR 交汇。请将该属性设置为布尔值，并将该值应用于信号线和管脚以约束各 SLR 交汇：

- TRUE：表示目标信号线对象应跨 SLR 或者目标管脚对象应跨 SLR 连接。TRUE 值只能应用于寄存器间连接，并且寄存器间仅含单扇出。

  注释：TRUE 值不得用于随机逻辑。该选项可用于确保在尝试多种实现策略时，寄存器链始终跨特定寄存器上的 SLR 边界。

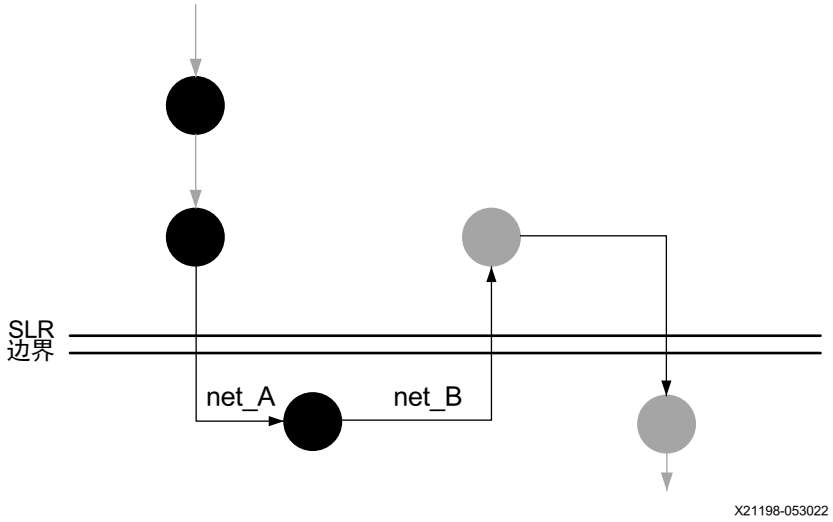
- FALSE：表示目标信号线对象不应跨 SLR 或者目标管脚对象不应跨 SLR 连接。FALSE 值可应用于任意信号线或管脚。

注释：管脚不得位于宏原语内部，因为这些管脚为内部管脚，不得对其加以约束。

在以下示例中，流水线寄存器链两次跨同一个 SLR，导致出现意外的低效率曲折路径。

注释：在接下来的 2 个图中，每个点都表示 1 个寄存器阶段。

图 37：**设置 USER_CROSSING_SLR 属性前的次优 SLR 交汇**

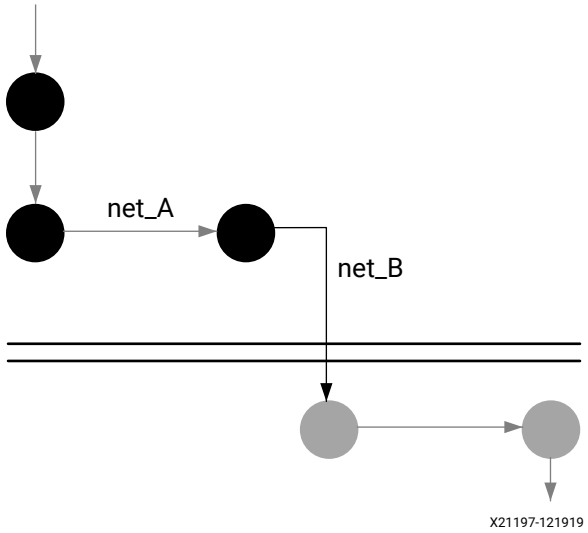


为了实现最优化布局（其中仅有 net_B 跨 SLR），应用以下 XDC 约束：

```
set_property USER_CROSSING_SLR FALSE [get_pins -leaf -of [get_nets net_A]]
set_property USER_CROSSING_SLR TRUE [get_pins -leaf -of [get_nets net_B]]
```

生成的布局仅在 net_B 上包含单个 SLR 交汇，如下图所示。

图 38：**设置 USER_CROSSING_SLR 属性后的最优化 SLR 交汇**



使用 SLR 交汇寄存器

处理 SSI 技术器件时，可将寄存器间 SLR 交汇映射到通过专用 SLL 布线直接驱动 CLB RX_REG 的特定 CLB TX_REG。将 TX_REG 到 RX_REG SLR 交汇拓扑结构用于流水线寄存器交汇可带来如下性能优势：

- SLR 交汇的布局垂直分布，可减少 SLR 边界附近的布线拥塞。

- 将寄存器布局在 SLR 交汇 slice 内可以提高延迟估算准确性，从而提高时序约束 QoR。
- SLR 交汇性能会更快且一致性更高。

您可在预计将布局在 SLR 交汇边界处的寄存器上设置 USER_SLL_REG 属性。如果满足以下条件，`place_design` 会忽略 USER_SLL_REG 约束：

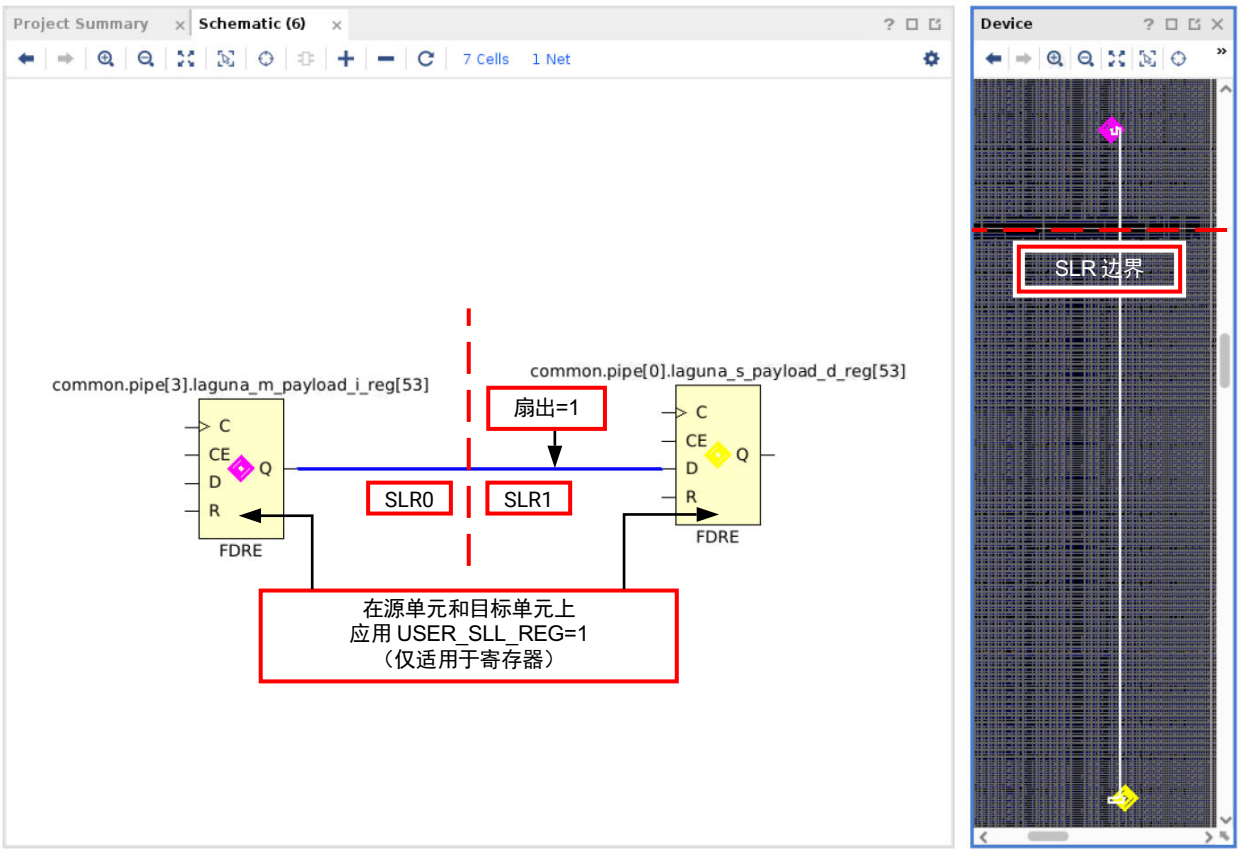
- 寄存器 D 和 Q 管脚连接到不跨越 SLR 边界的信号线。
- 寄存器 D 和 Q 管脚连接到扇出 > 1 的信号线。

以下提供了 USER_SLL_REG 约束示例：

```
# USER_SLL_REG
set_property USER_SLL_REG TRUE [get_cells src_slr_i/G1B.SLL_reg[227]]
set_property USER_SLL_REG TRUE [get_cells dest_slr_i/G1B.SLL_reg[227]]
```

下图显示了 USER_SLL_REG 约束以及生成的最优化布局布线的示例。

图 39：**USER_SLL_REG 约束**



针对 SLR 交汇使用自动流水打拍

无论使用的是软核 SLR 布局规划约束、硬核 SLR 布局规划约束还是不使用任何布局规划约束，满足位于不同 SLR 内的设计主要部分之间的时序要求所需的流水线阶段数量都因如下条件而异：

- 目标频率

- 器件布局规划
- 器件速度等级

您可利用自动流水线功能来允许布局器算法判断所需阶段数量及其最佳位置，从而帮助跨 SLR 边界实现时序收敛。

您可通过在 RTL 中的总线和握手逻辑上设置 AUTOPIPELINING_* 属性来启用自动流水线功能，但请确保额外的时延不会对设计功能产生负面影响。此外，您还可使用 AMD AXI Register Slice Memory Mapped or Streaming IP，此 IP 在 SLR 交汇中进行配置。

集群逻辑

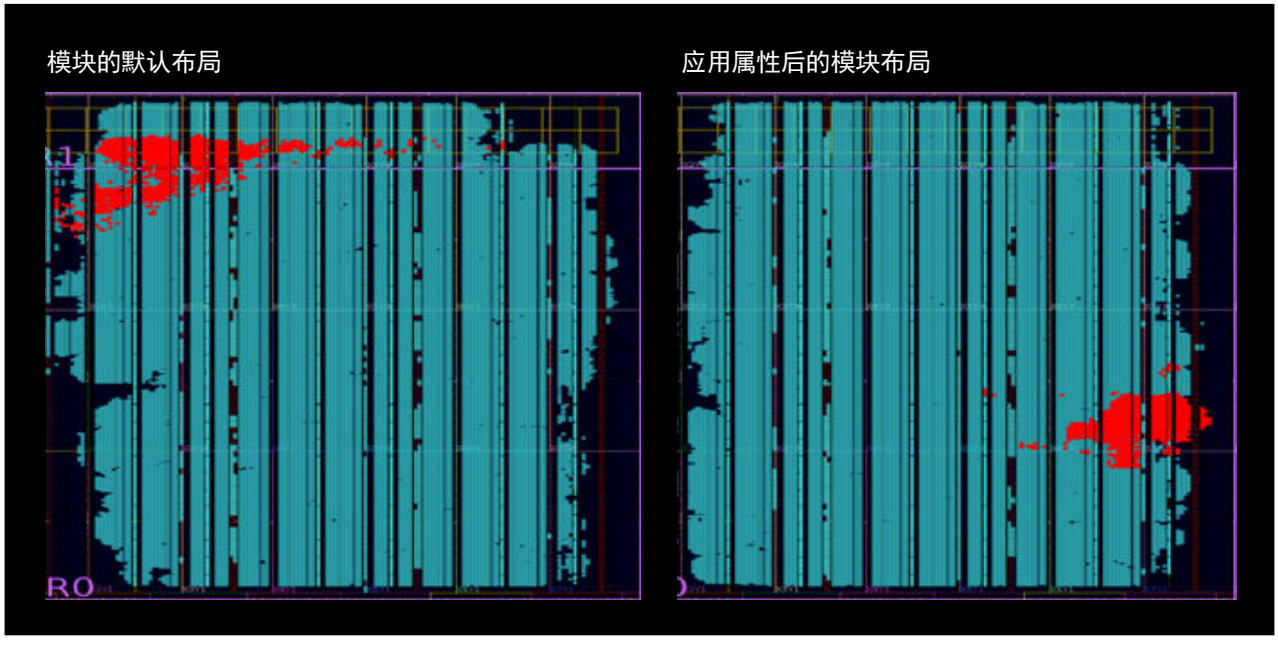
要将某些实例或设计元素尽可能分组并布局在一起，可以使用 USER_CLUSTER 属性指定分组的单元的布局方式。这样您即可管理逻辑分区以及 Vivado Design Suite 布局器的行为。您可在一组分层实例上指定 USER_CLUSTER 属性以形成软集群，并在 SLR 分区和分区驱动的布局器阶段中考量使用这些软集群。如需了解有关该属性的更多信息，请参阅《Vivado Design Suite 属性参考指南》(UG912)。

重要提示！布局器将该属性视为应尽可能尝试遵循的准则。但您可覆盖该属性来达成有效的布局结果。

提示：要跨 SLR 管理布局，请首先使用 USER_SLR_ASSIGNMENT 将逻辑分配给任一 SLR 或分组，然后添加 USER_CLUSTER 来控制 SLR 内的逻辑实例的分组。

分析设计后，找到顶层失败路径中涉及的特定实例，判定失败的实例内的逻辑布局是否已向外散播，您可使用 USER_CLUSTER 属性来改善布局，这样即可提升设计的性能。在下图中，左侧示例显示的是该实例的默认布局。右侧示例显示的是应用 USER_CLUSTER 属性后的实例布局。

图 40：**使用 USER_CLUSTER 属性前后的实例布局**



使用 IMUX 寄存器改善硬核宏的时序

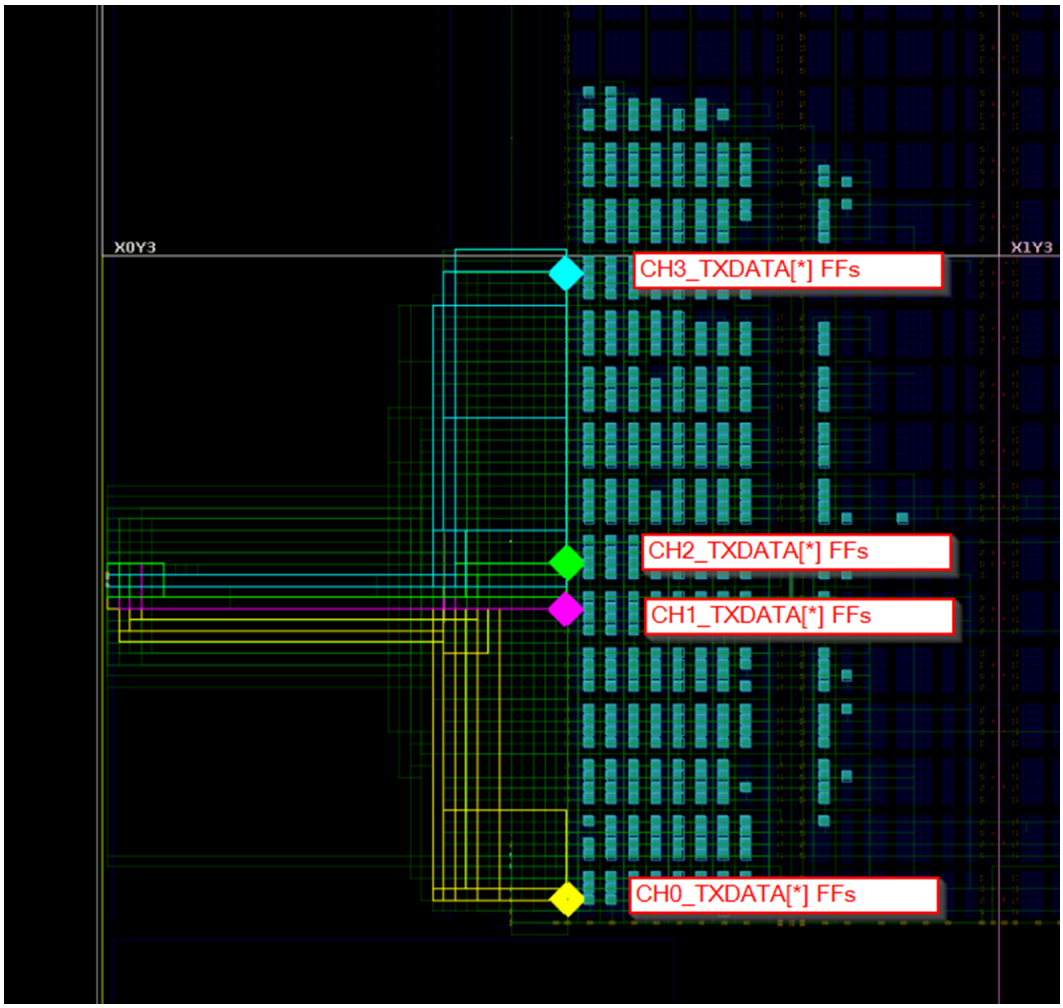
IMUX 寄存器位于驱动 PL 中硬核宏的 IRI_QUAD site 位置，可用于来改善时序。PL 中的所有硬核宏（如 RAMB36、RAMB18、URAM288、DSP58 和 GTYP_QUAD）都有驱动这些硬核宏输入管脚的 IMUX 寄存器触发器。您可对设计中的触发器应用 IMUX 寄存器 (IMR) 约束以便在设计实现期间自动利用此硬件功能。

在以下示例中，GTYP_QUAD 的 CH*_TXDATA[*] 管脚的数据路径利用的正是 IMR 触发器。使用这种方法来约束驱动硬核宏的输入触发器，可以缓解这些路径的时序收敛，由于从 IMUX 寄存器到硬核宏输入的布线是专用的，因此不会受到次优布局或拥塞的影响。

```
set_property IMR TRUE [get_cells -of [get_pins -leaf -of [get_nets -of
[get_pins fmx_pcs_raw_inst/GT/CH*_TXDATA[*]]] -filter DIRECTION==OUT] -
filter PRIMITIVE_GROUP==REGISTER]
```

下图显示了通过将 IMR 属性设置为 TRUE 所生成的布局和连接。

图 41：**IMR 属性设为 TRUE**



使用智能设计运行

要自动解决实现期间的大部分时序收敛难题，您可以使用智能设计运行 (Intelligent Design Run)。智能设计运行是一种特殊类型的实现运行，它可利用 `report_qor_suggestions`、基于 ML 的策略预测以及增量编译。智能设计运行最多能运行 6 次布局布线迭代，因此其典型编译时间达标准运行的 3.5 倍。但由于使用智能设计运行可以减少达成时序收敛所需的知识并节省大量用户分析时间，因此其利益不可小觑。

建议：由于智能设计运行耗时比标准实现运行更长，因此 AMD 建议使用智能设计运行的频率因少于标准运行。例如，在解决所有方法论警告并尝试“Default”（默认）策略和“Explore”（探索）策略等多项常用的实现策略之后，再使用智能设计运行。

提示：要加速迭代，您可从智能设计运行中提取 QoR 建议和 ML 策略，并在标准实现运行中使用这些策略。如果执行了重大设计更改，请重新运行智能设计运行以更新关联文件。

智能设计运行由下列几个阶段组成：

1. 使用 `report_qor_suggestions` 按预定义顺序对设计中的元素应用最优化属性。
2. 使用机器学习 (ML) 策略为 `opt_design`、`place_design`、`phys_opt_design` 和 `route_design` 生成专为设计最优化的工具选项。
3. 使用“Last Mile Timing Closure”（最后一步时序收敛）功能来对难以解决的路径进行广泛尝试，以获得最终结果。

要确保成功使用智能设计运行，请遵循下列要求进行操作：

- 实现必须基于工程来执行。对于非工程用户，最简单的方法是使用 `opt_design` 前的检查点创建基于综合后网表的工程。
- 设计必须具有基线，且基线必须具有准确且可实现的约束。
- 所有设计都必须符合建议的方法论，`report_methodology` Tcl 命令可提供相关报告。
- 对于基于 SSI 技术的器件，可能需要基于 SLR 的布局规划。
- 仅限应用自动实现建议。必须先应用基于文本的建议或 `APPLICABLE_FOR = synth_design` 建议，然后再启动智能设计运行。

欲知详情，请访问此链接以参阅《Vivado Design Suite 用户指南：设计分析与收敛技巧》(UG906) 中的相应内容。

功耗收敛

鉴于尽可能降低功耗的重要性，Vivado 工具支持采用各种方法来获取准确的功耗估算以及各种功耗最优化功能。如需了解其他信息，请参阅《Vivado Design Suite 用户指南：功耗分析与最优化》(UG907)。

估算整个流程的功耗

随着设计流程进入综合与实现阶段，必须定期监控和检查功耗以确保热耗散保持在预算范围内、开发板电压调节器保持在当前工作限制范围内且设计保持在任何系统功耗限制范围内。一旦功耗与预算值过于接近，就可及时采取补救措施。

使用下列 XDC 约束指定功耗预算，以报告功耗裕度：

```
set_operating_conditions -design_power_budget <value in watts>
```

该值供 report_power 命令使用。计算所得片上功耗与功耗预算之差即为功耗裕度，超出功耗预算时，该值在 Vivado IDE 中将以红色显示。这样更便于监控整个流程中的功耗状况。

提示：对于 AMD Versal™ 器件，您可以从包含环境设置的电源设计管理器 (PDM) 工具（从 china.xilinx.com/power 下载）中导出 XDC 文件，包括可用作功耗预算约束的 PDM 工具估算。导出 XDC 文件时，请取消选中“Power Rail Constraints”（电源轨约束）。您可使用 PDM 工具或 XDC 覆盖功耗预算。添加用于功耗裕度报告的 XDC 约束。

功耗估算的精确性因估算时的设计阶段而异。要通过实现来估算综合后功耗，请运行 report_power 命令，或者在 Vivado IDE 中打开“Power Report”。

- 综合后：网表已映射到目标器件中可用的实际资源。
- 布局后：网表组件已布局在实际器件资源中。借助这些封装信息，即可掌握最终逻辑资源计数和配置。这些精确数据可导出至 PDM 工具电子数据表中。这样便可以：
  - 在 PDM 工具中执行假设分析。
  - 为精确填充电子数据表奠定基础，以便将来在具有相同特性的设计中使用。
- 布线后：在布线完成后，将定义所用布线资源的所有相关细节和设计中每个路径的精确时序信息。

建议的功耗约束

对设计应用正确的功耗约束对于设计收敛至关重要。Vivado 工具的 report power 命令可基于应用的预算和其他约束来报告功耗裕度。以下是建议的最低限度约束列表。如需了解更多信息，请参阅《Vivado Design Suite 用户指南：功耗分析与最优化》(UG907)。

建议的最低限度约束

以下约束旨在确保功耗估算会检查功耗预算，并使用最差情况最大值工艺来执行静态功耗分析：

```
set_operating_conditions -design_power_budget <Power in Watts>
set_operating_conditions -process maximum
```

其他建议约束

以下约束用于定义热处理解决方案，并支持使用 report_power 命令来估算结温，从而更准确地估算静态功耗：

```
set_operating_conditions -ambient_temp <max Ambient requested for
application is Celsius>
set_operating_conditions -thetaja <the rise in junction temperature for
every watt dissipated, obtained from thermal simulation, C/W>
```

Vivado 工具的 report_power 命令还支持您使用以下约束基于调节器或电压调节器模块 (VRM) 来报告功耗。

创建新的电源轨

```
create_power_rail <power rail name> -power_sources {supply1, supply2 ,..}
```

为现有电源轨添加电源

```
add_to_power_rail <power rail name> -power_sources {supply1, supply2, ..}
```

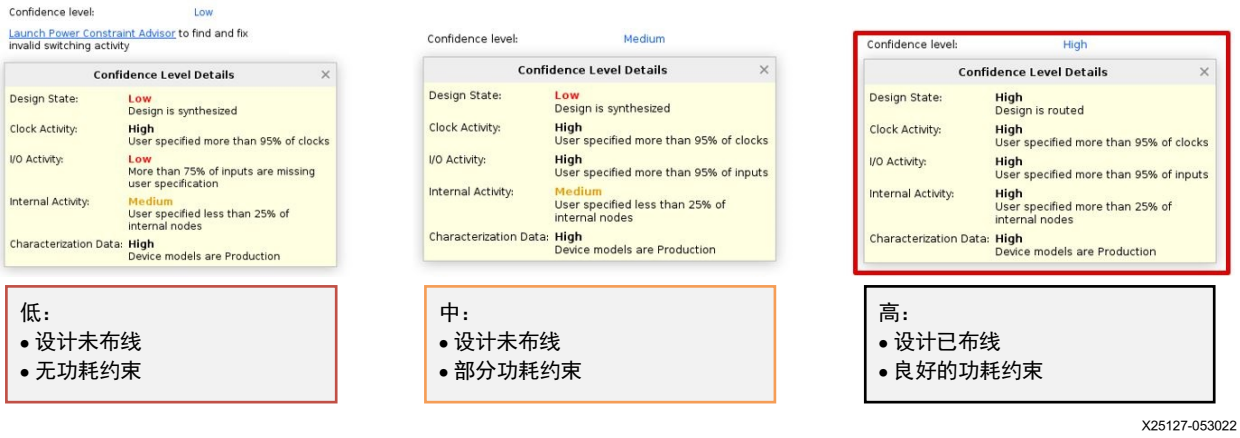
定义电流预算

```
set_operating_conditions -supply_current_budget {<supply rail name>
<current budget in Amp>} -voltage {<supply rail name> <voltage>}
```

有助于精确完成功耗分析的最佳实践

要精确完成功耗分析，请确保时序约束、I/O 约束和开关活动的准确性。`report_power` 命令用于指示可信度，如下图所示。请将可信度目标设为“High”（高）以确保功耗分析准确。欲知详情，请访问此链接以参阅《Vivado Design Suite 用户指南：功耗分析与最优化》(UG907) 中的相应内容。

图 42：**功耗分析可信度**



运行功耗分析后复查设计的配电情况

您可复查片上总功耗和热属性以及资源层面的功耗详情，以确定设计中哪些部分产生的功耗占总功耗比例最大。欲知详情，请访问此链接以参阅《Vivado Design Suite 用户指南：功耗分析与最优化》(UG907) 中的相应内容。

电源/功耗提示：根据当前板级原理图或印刷电路板复查并确认已完成的 Vivado 设计的去耦要求。您可使用如下 Tcl 命令通过 Vivado 工具的 `report_power` 来生成 `.xpe` 文件：

```
set_operating_conditions -process maximum
```

```
set_operating_conditions -ambient_temp <max Ambient requested for
application is Celsius>
```

```
set_operating_conditions -thetaja <the rise in junction temperature for
every watt dissipated, obtained from thermal simulation, C/W>
```

```
report_power -xpe {C:/Design_Runs/Vivado_export.xpe} -name {Any_Name}
```

随后，您可将 `.xpe` 文件导入 电源设计管理器 (PDM) 工具（从 china.xilinx.com/power 下载）。例如，Power Delivery 表根据功耗估算和供电选项显示了去耦要求。

功耗时序裕量

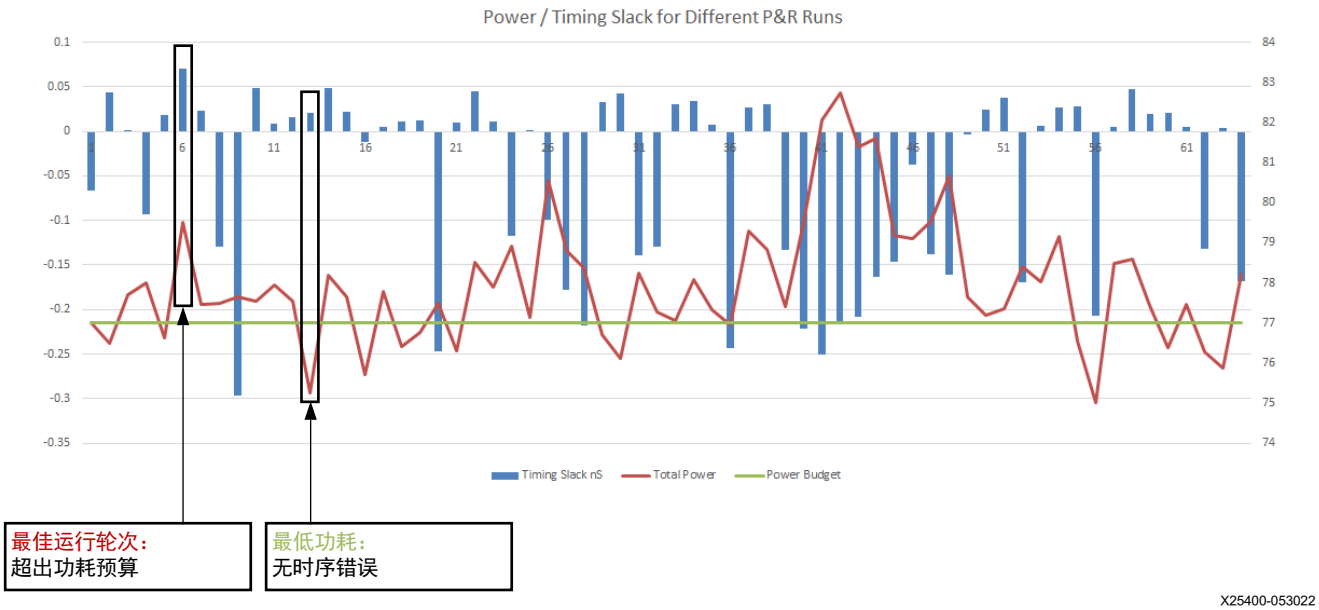
为设计达成时序收敛时，更有效且更高效的做法是，从功耗角度同时达成设计收敛。此方法能够选择使用同时满足这两项条件的最佳运行方式。要同时达成时序收敛和功耗收敛，请在正在运行的脚本中添加 `report_power` 约束。如需了解更多信息并获取脚本示例，请参阅答复记录 76056。

下图显示了此方法的示例。对于全部 64 次时序收敛运行，已同时运行功耗报告，并且所有运行都绘制在一起。在此图中，36 次运行没有时序错误，从功耗角度而言，总功耗预算为 77W。64 次运行在 75 W 到 83 W 范围内，即范围为 8 W 或 ~10%。

如果从时序角度来寻找最佳运行，第 6 次运行的功耗估算为 79.5 W，超出了总功耗预算。但从无时序错误的运行角度来看，第 13 次运行产生的功耗最低，为 75 W，同时没有任何时序错误。通过从时序和功耗角度来理解设计，您即可选择最适合这两者且不影响时序结果的运行。在此示例中，借助此方法能够节省 4W 功耗。

电源/功耗提示：您还可以通过移除 DONT_TOUCH 约束以提前进行逻辑裁剪（包括时钟设置原语），从而降低设计功耗。

图 43：**不同布局布线运行的功耗和时序裕量**



DRC 收敛

Vivado Design Suite 包含一份设计规则检查 (DRC) 列表，您可使用 `report_drc` Tcl 命令来运行其中各项检查。该 DRC 列表拆分为多个规则卡。在实现期间，其中部分规则卡会作为部分命令的前置条件 DRC 来自动执行，例如，`opt_design`、`place_design` 和 `route_design`。

您必须审慎评估来自前置条件 DRC 和来自 `report_drc` 命令的违例。尽早复查这些消息至关重要，因为这样可以避免在实现流程的后期出现时序或逻辑相关的问题。实现期间出现的任何严重警告 DRC 都会在 PDI 生成期间变成错误。您必须先解决严重警告和警告 DRC 的问题，然后才能进入下一个实现阶段。

提示：对于可安全忽略的 DRC 违例，可使用豁免机制来豁免此类违例。欲知详情，请访问此链接以参阅《Vivado Design Suite 用户指南：设计分析与收敛技巧》(UG906) 中的相应内容。

如需了解有关 `report_drc` Tcl 命令的更多信息，请参阅《Vivado Design Suite Tcl 命令参考指南》(UG835)。

第 4 章

系统性能收敛

AMD Versal™ 器件是围绕异构计算引擎来构建的，这些引擎通过 NoC 或 PL 彼此相连并通过高性能收发器和 I/O 连接到外部系统。在系统应用与映射阶段，器件接口和总体计算要求可用于指定器件中实现的每个计算和控制功能的目标性能。每个功能都设计为映射到最合适的硬件资源，此类资源使用对应编程语言和编译软件（例如，对应嵌入式处理器系统使用系统软件，对应 AI 引擎或 PL 内核使用 C/C++ 语言、对应高性能 PL 内核或固件则使用 RTL 等）。

各设计团队必须先在功能级别确认功能和期望的性能，然后再将其集成到部分系统应用或整个系统中。在集成阶段中，功能可能失效，且性能可能降级。由于 AMD Versal™ 器件所支持的系统应用的复杂性和异构性质，因此必须事先明确并规划分析和调试方法论。AMD Vitis™ 和 AMD Vivado™ 工具均为综合性且互补性的设计环境，可提供在硬件中进行功能仿真、设计特性报告以及数据测量或探测所需的所有功能。

重要提示！应用性能需求通常是通过下列方式来满足的：在关键的块（基于 PL 的块或硬核 IP 块）之间创建适当的连接架构、为控制块和计算块提供正确的吞吐量和时延预算并为块和存储器之间的数据移动设置正确的服务质量 (QoS) 约束。通常，满足性能目标并不需要尝试为所有 PL 块都实现尽可能最佳的时钟频率，这样可能因增加逻辑区域而导致功耗增加，且无法获得相关的性能增益。

您可使用 AXI Traffic Generator 来代替使用基于文件的输入和输出。AXI Traffic Generator 与静态且受限的基于文件的输入和输出相反，通过动态方式来生产和使用数据。如需了解更多信息，请参阅 AXIS 外部流量生成器功能特性教程。

以下章节提供了分步骤分析方法建议，用于识别应用瓶颈、识别围绕 1 个或多个功能的性能不匹配问题并基于目标器件资源来解决常见性能问题。

分析基于平台的设计的系统性能

您可按下列章节中所述方式来分析仿真和硬件中的系统性能。

分析仿真中的系统性能

分析仿真中的系统性能时，关键数据路径为 AI 引擎、PL、NoC (DDR) 和高速收发器之间的路径。集成系统前，必须确保每个块都满足其性能估算值。

注释：由于传统设计不含可在 Vitis 工具中使用的平台，因此基于 RTL 仿真和硬件调试功能（例如，ChipScope™ 调试 IP 核及专用的硬核块调试功能等）执行系统性能分析。

分析仿真中的 AI 引擎性能

最终确定应用架构并且 AI 引擎与 PL 之间的设计分区完成后，下一步是开发 AI 引擎应用。您可以使用 `aiesimulator` 通过追踪剖析功能来测量应用性能。您还可使用 `aiesimulator` 输出来测量性能。

以下示例中，性能测量方法为 (结束时间 - 开始时间)/样本数，其中：

- 每一行均表示 1 个 64b 数字 (2 cint16s)。有 51200 个 64b 数字（例如，102400 个 32b 样本）。

- 吞吐量 = 102400/(182452500 ps - 5790 ns) 样本数/s = 579.636 MSps。

```
     1 T 5790 ns
     2 0 0 0 0
     3 T 5792500 ps
     4 0 0 0 0
     5 T 5795 ns
     6 0 0 0 0
     7 T 5797500 ps
     8 0 0 0 0
     9 T 5800 ns
...
...
...
102495 -5107 2007 -32768 -18047
102496 T 182450 ns
102497 -25374 -19023 3957 3067
102498 T 182452500 ps
102499 TLAST
102500 18230 14818 11355 -5427
```

要进一步分析 AI 引擎性能瓶颈，AMD 建议运行 `aiesimulator` 或硬件仿真（搭配 AI 引擎追踪和剖析选项）。您可以打开为仿真运行所生成的运行汇总文件，其中包含 Vitis 分析器中的追踪数据和剖析数据。这样即可生成追踪视图和剖析视图，以帮助您识别性能根源问题。欲知详情，请访问此链接以参阅《AI 引擎工具和流程用户指南》(UG1076) 中的相应内容。

此外，您可使用 AI 引擎运行时事件 API 获取有关 AI 引擎计算图带宽、吞吐量和时延的详细剖析数据。欲知详情，请访问此链接以参阅《AI 引擎工具和流程用户指南》(UG1076) 中的相应内容。

分析仿真中的 PL 内核性能

HLS

由 HLS 开发的所有内核均可使用编译器指令和 HLS 编译指示来加以最优化。Vitis HLS 编译器可生成详细报告，其中包含有关 Fmax、资源使用率和性能的信息。除了汇总报告外，调度查看器还可直观再现设计构建方式以及操作调度方式。您可使用此视图来帮助识别已综合的设计中的次优部分。

您可通过运行 HLS 协同仿真流程来对这些编译时间报告加以补充。运行该流程时，Vitis HLS 会从仿真结果中自动提取性能数据，并报告额外的性能信息，例如， FIFO 高水位标记的最小、最大和平均运行时间。AMD 建议先利用所有这些分析功能，然后在系统中集成 HLS 内核。

注释：在独立环境中无法满足性能要求的内核在完整的系统环境中也无法满足性能要求。

影响 HLS 内核性能的因素多种多样，包括接口属性、循环级别并行度、任务级别并行度等等。理解启动时间间隔 (II) 和数据流的概念对于实现良好的结果尤为重要。启动时间间隔以时钟周期数来衡量，表示特定循环或进程重新启动的频率。例如，如果循环以 II=1 成功完成综合，那么在生成的 RTL 中，每个周期都启动新的循环迭代。II 与吞吐量这一关键性能指标紧密相关。数据流是利用任务级别并行度的性能最优化功能。数据流在设计中尽可能允许不同子进程并行运行而不是顺序运行。利用数据流实现最优结果需要适合的代码结构。如需了解有关启动时间间隔的更多信息以及有关其他 HLS 性能最优化的信息，请参阅《Vitis 高层次综合用户指南》(UG1399)。

RTL

利用诸如 Vivado 仿真器等仿真工具代替 Vitis 环境，通过标准 RTL 仿真方法对 RTL 内核进行仿真。所有标准 RTL 仿真最佳实践都适用于 RTL 内核。请检查 RTL 内核的功能是否正确，并判定性能相关的最佳结果。确保每个 RTL 内核都满足性能目标，然后再将 RTL 内核添加到更大的系统。

您开发的所有 RTL 内核都必须使用定制 RTL 测试激励文件或使用 AMD Vivado™ IP 目录中提供的 AMD LogiCORE™ AXI Verification IP (VIP) 进行块级仿真。如需了解更多信息，请参阅《AXI Verification IP LogiCORE IP 产品指南》(PG267)。

提示：在 RTL 中可写入其他性能计数器来计算 PL 中的周期数，并计算往来 AI 引擎的时延和吞吐量。

使用硬件仿真来分析系统性能

包含 AI 引擎、PS 和 PL 的 Versal 器件系统可使用 Vitis 连接器加以集成，然后使用 Vitis 硬件仿真结合在一起进行仿真。硬件仿真允许您观察并测量 AI 引擎、PS 和 PL 交互作用组合在一起给系统性能所带来的影响。在硬件仿真中，PL 内核作为 RTL 运行、AI 引擎内核在 aiesimulator 中运行，而 PS 代码则在 AMD 快速仿真器 (QEMU) 中运行。部分基础架构块使用传输事务级模型 (TLM) 加以抽象化，以便提升仿真速度。硬件仿真十分接近（但并未完全达成）周期精确，可提供有价值的表征，用于在实现之前对主系统性能要素进行分析、调试和确认。

硬件仿真可自动运行并根据用户设置生成各项性能相关报告，例如，应用剖析汇总以及应用时间线轨迹。您可在 Vitis 分析器中查看这些报告，以获取有关性能的各种实用的洞察，例如，数据传输大小和效率、内核运行时间、停滞信息等。除上述报告外，Vitis 分析器还可提供详细的活动波形，以便您对系统特定部分开展自定义的高精度分析。

如需了解有关硬件仿真和 Vitis 分析器的更多信息，请参阅《Vitis 统一软件平台文档：应用加速开发》(UG1393) 和《AI 引擎工具和流程用户指南》(UG1076)。

其他硬件

如需通过可综合的方式来测量吞吐量，可设计 RTL IP 来计算特定传输事务所耗用的周期数（例如，将含有“B”拍数据的有效负载从源发射至目标所耗用的时间）。这种方法将触发计数器从首个 TVALID 开始对事件进行计数直至 TLAST 为止。或者，您可以利用 Vivado IP 目录中提供的 AXI Performance Monitor (APM) IP 来对事件进行计数。

软件

必须开发轻量级 PS 应用才能运行与应用相关的整个系统（例如，触发器 PL IP、启动流量等）。例如，某些应用可利用 PL 复位断言无效而无需任何 PS 应用。在此情况下，可使用简单的 PS 应用通过打印“Hello World”来触发系统仿真。

裸机应用足以胜任此任务。基于 Linux 的硬件仿真同样可行，但在此阶段无法发挥其他作用。

分析硬件中的系统性能

对于包含 AI 引擎、PL 和 PS 的系统，请确保您的设计遵循以下先决条件以分析性能：

- 硬件
  - 使用 PL IP（例如，APM IP、RTL/HLS 内核等）来计算系统中的吞吐量时延，以测量周期精确性性能。
- 软件
  - 使用 Linux 操作系统 (OS) 以利用软件平台中包含的 aiecompiler 所提供的剖析 API。如需获取有关启用 Linux 和构建附属资料的更多信息以及有关剖析 API 的详细信息，请参阅《AI 引擎工具和流程用户指南》(UG1076)。
  - PS 应用可用于协调 Arm® Cortex®-A72 上运行的整个系统（例如，从 PL/DDR 启动数据流量、启动计数器等）。PS 应用需与 G++/GCC 编译器进行交叉编译，以供 Cortex-A72 使用。PS 应用还可用于读取并输出 RTL/HLS/APM 计数器，以测量吞吐量和时延。
  - 在硅片上实现设计后，可使用 Linux OS 以利用 aiecompiler 所提供的剖析 API。

使用 RTL 或 HLS 监控器来测量性能

您可使用定制 RTL 或 HLS 内核来计算 AXI4-Stream 传输事务开始与结束之间所耗用的周期数。在 PS 应用中，您可读写这些计数器以在运行时间测量性能。

测量 AI 引擎性能

使用 AI 引擎运行时事件 API 测量性能

使用 Vitis 工具或 aiecompiler 编译计算图后，可监控每个 AI 引擎阵列接口（或 shim 接口拼块）以计算特定事件数量。您可使用一些剖析事件来计算 AI 引擎阵列接口内有效的 AXI4-Stream 数据传输事务数量。调用 API 时，PS 会发出一系列 AXI4-MM 命令，以将 AI 引擎阵列接口配置为计算有效事件数。AI 引擎阵列接口中的事件计数提供了一种实用的方法，无需对系统添加任何额外硬件即可对系统进行测量。

注释：每个 AI 引擎阵列接口仅含 2 个性能计数器，但每个 AI 引擎阵列接口中都有 14 个 64b 串流。因此，每次使用这些探测 API 只能监控 2 个 AI 引擎 PL 接口。

以下示例使用 `io_stream_start_to_bytes_transferred_cycles` 事件 API 来测量计算图的吞吐量。此 API 使用 2 个性能计数器来追踪传输的字节数和耗用的周期数。此事件 API 可捕获并计算通过计算图传输指定量的数据的活动、停滞和空闲周期总数之和。此 API 可在输入和输出串流上使用。

```
gr.init();
event::handle handle = event::start_profiling(plio_out,
event::io_stream_start_to_bytes_transferred_cycles, 256*sizeof(int32));
gr.run(8);
gr.wait();
long long cycle_count = event::read_profiling(handle);
event::stop_profiling(handle);
double throughput = (double)256 * sizeof(int32) / (cycle_count * 1e-9); //
byte per second
```

当要传输的字节数未知时，您可使用另一个事件 API。以下示例使用 `io_stream_running_event_count` 事件 API 来测量计算图的吞吐量。串流将按指定时间间隔运行，并捕获串流的活动事件数量。

```
...
...
using namespace adf;
event::handle handle_0;
PLIO duc_plio[2] = {*duc_in0, *duc_out0};
d=0;
while(d < NUM_DUC_SLAVES) {
    long long throughput_out_min = 990000000; // initial value to some high
number
    long long throughput_out_max = 0;
    int iter=0;
    while(iter < 5) {
        long long count_start, count_end;
        long long throughput;
        handle_0 = event::start_profiling(duc_plio[d],
event::io_stream_running_event_count);
        count_start = event::read_profiling(handle_0);
        //precision of usleep is dependent on linux system call
        usleep(1000000); //1s
        count_end = event::read_profiling(handle_0);
        event::stop_profiling(handle_0);
        if (count_end > count_start) throughput = (count_end-count_start);
        else throughput = (count_end-count_start+0x100000000); //roll over
correction for 32b performance counter
```

```
        if (throughput<throughput_out_min) throughput_out_min = throughput;
        if (throughput>throughput_out_max) throughput_out_max = throughput;
        iter++;
    }
    printf("[throughput] %d\tMin:%llu\tMax:%llu\tRange:%llu\n", d,
throughput_out_min, throughput_out_max, throughput_out_max-
throughput_out_min );
    d++;
}
printf("[main] Performance measurements Done ... \n");
...
...
```

欲知详情，请访问此链接以参阅《AI 引擎工具和流程用户指南》(UG1076) 中的相应内容。

利用事件追踪调试测量性能

欲知详情，请访问此链接以参阅《AI 引擎工具和流程用户指南》(UG1076) 中的相应内容。

在硅片上调试 AI 引擎 PL 性能

以下是用于调试 AI 引擎 PL 性能的推荐方法：

- 将 AI 引擎计算图细分为多个较小的计算图，以便分析硅片上的瓶颈。例如：
  - 如果计算图在 AI 引擎和 PL 内包含内核，则可将该计算图划分为多个子计算图，以验证功能和性能。通过这种方法即可找到性能瓶颈。
  - 如果计算内核（位于 AI 引擎内或 PL 内）收到来自多个 AXI4 串流的数据，那么由于不同串流上数据到达时间不同，可能导致内核性能下降。发生此现象的原因可能是由于反压，或者也可能是由于先前计算图中内核的不同计算复杂程度所导致的。计算图可在内核级别加以细分，以验证所有串流是否都已达到最佳性能。

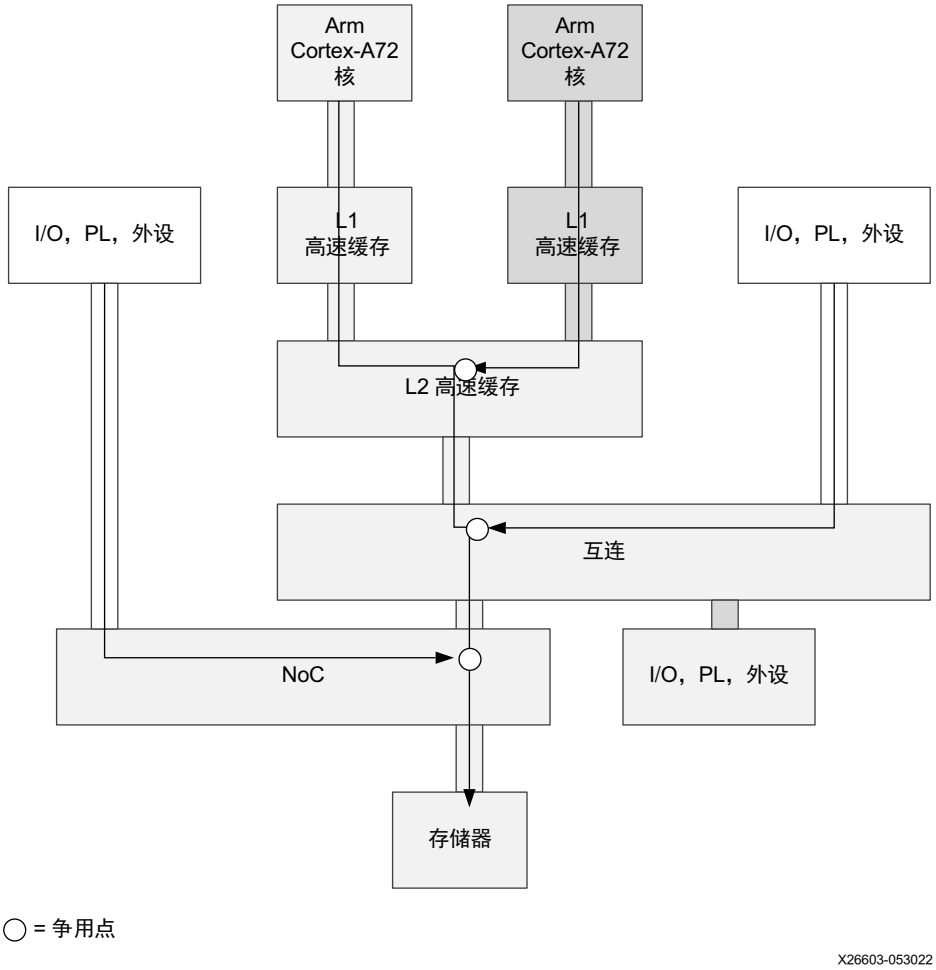
  注释：或者，也可以使用内核级别性能指标和调试来分析瓶颈。
- 将 AI 引擎计算图替换为简单的直通系统。
- 使用事件追踪调试功能来计算不同内核中存储器停滞次数。欲知详情，请访问此链接以参阅《AI 引擎工具和流程用户指南》(UG1076) 中的相应内容。

提升 PS 中的性能

在 Versal 器件处理器系统 (PS) 中，您可使用以下互连流量类型来控制服务质量 (QoS)：低时延数据路径和高吞吐量数据路径。欲知详情，请访问此链接以参阅《Versal 自适应 SoC 技术参考手册》(AM011) 中的相应内容。

下图显示了影响 PS 架构的潜在争用点。

图 44：**PS 架构争用点**



考量处理器系统和存储器子系统性能时，首先要思考的是处理器性能、其时钟频率、存储器层级、高速缓存以及更常用的内部和外部存储器功能。虽然这些考虑因素都很重要，但掌握处理器系统架构也同样重要。

外部共享 DDR 存储器子系统和片上 OCM 是 Versal 器件中存在的 2 个主要存储器子系统。访问存储器子系统的主控制器可位于 PL 或 PS 中。PL 主控制器在遍历 PS 以访问 OCM 或外部 DDR 存储器时可能对 PS 主控制器性能产生显著影响。

PL 主控制器流量布线方式为：

- PL 主控制器（软核 IP）通过 NoC NMU 连接到 NoC
- PL 主控制器连接到 PS AXI 接口
- PL 主控制器连接到 PS ACP

AI 引擎和 CPM 同样可能需要通过 PS 互连来对流量进行布线，请务必关注流量的多种布线方式。

内部 PS 主控制器（APU、RPU、DMA 等）会生成流量，并利用连接到 NoC 的 PS 互连。

- PL 到 PS 接口直连（S_AXI_FPD、S_ACE_LITE_FPD 和 S_AXI_LPD）
  - S_AXI_FPD（位于 FPD 内）采用虚拟连接且无相干关系

- S_ACE_LITE_FPD（位于 FPD 内）采用虚拟连接且有相干关系
- S_AXI_LPD（位于 LPD 内）可配置为物理或虚拟连接且有相干关系

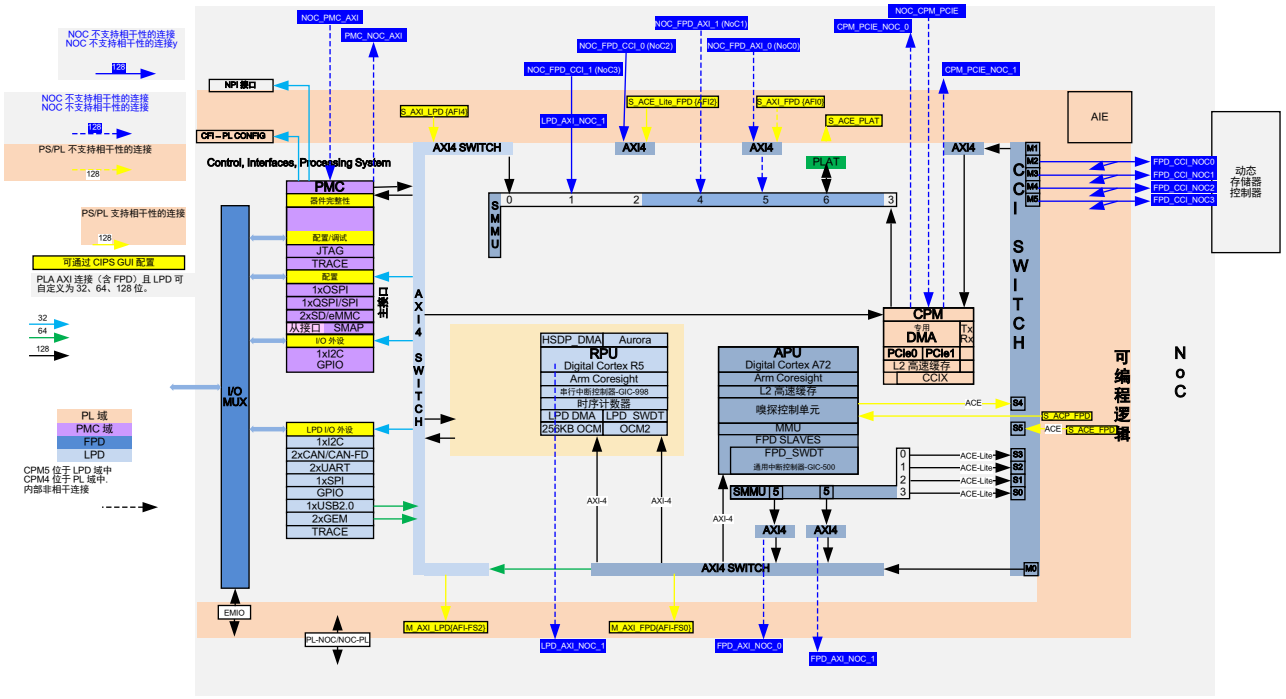
· NoC 到 PS 接口（NoC_FPD_AXI0、NoC_FPD_AXI_1、NoC_FPD_CCI_0、NoC_FPD_CCI_1，全部位于 FPD 内）

- NoC_FPD_AXI0 和 NoC_FPD_AXI1 采用虚拟连接且无相干关系
- NoC_FPD_CCI_0 和 NoC_FPD_CCI_1 采用虚拟连接且有相干关系

了解并利用主控制器到外部 DDR 存储器或 OCM 的多种连接方式对于 PS 互连布线最优化和避免 PS 互连拥塞具有至关重要的意义。

例如，选择使多个主控制器的流量穿过 CCI 的布线方式对于性能不利，因为仅支持将最多 4 个 CCI500 AXI4 主端口连接到 4 个 NoC NMU。在下图中，这些主端口是 M2、M3、M4 和 M5。

图 45：**连接到 NoC NMU 的主端口**



X25052-053022

SMMU 和 CCI 同样可能影响性能。这两者各自都具有共享内部资源，可能导致增加时延从而降低吞吐量。如要通过最优化来提升性能，并且没有虚拟化/隔离和相干性要求，则不建议使用 CCI500 和 SMMU。

流量调节机制可在源（即 NoC 开关中的 NMU）和目标从接口 (MC) 中设置。您可使用这些机制来应用所需的 QoS 方案。

PS 互连不支持虚拟通道，因此必须使用物理分隔。

在入口方向（外部主控制器到 PS 从设备），每个 NSU 都支持单一流量类，因此不同的流量类必须通过不同物理端口进入 PS。同样，在出口方向，内部 PS 互连网络将承载不同流量类通过不同物理通道进入连接到水平 NoC 的 NMU。例如，4 条 CCI 到 NoC 通道中的 2 条可承载 LL（上图中的 M2 和 M3），另 2 条则可承载 BE（上图中的 M4 和 M5）。

要支持不断增长的性能需求，同时使能耗保持在可接受的程度，需要克服重重困难。诸如 Versal 器件之类的复杂器件可通过微调来达成最佳单位功耗性能。

为了满足这些现代化应用的性能要求，请执行下列操作：

- 使用业界标准的基准测试对含有 NoC-DDR 的 PS 进行基准测试
- 使用通用工具执行性能分析时，运行专属算法

PS 性能可基于实时非侵入性措施进行调优，这些措施用于在执行应用时收集性能数据。基于收集的统计数据，可采用频率缩放和任务调度等技巧进行系统调优，在不牺牲性能的前提下降低功耗。

APU、RPU 和 NoC 中的“Performance Monitor Units”（性能监控单元）可提供宝贵数据，用于了解 CPU 使用情况、高速缓存未命中、分支错误预测、查找代码执行热点等。

对于用户交互和设计，在 APU 中设置最多 6 个性能计数器，在 RPU 中设置最多 6 个计数器，用于读取目标事件以获取运行时的处理器性能信息。除了 6 个性能计数器外，每个处理器的 PMU 还可提供专用周期计数器。

对于 APU，可使用 AMD Linux 工具链 PetaLinux 创建 Linux 镜像，通过 Perf（性能）应用提供对 PMU 计数器的访问，用于对工具进行剖析和追踪。

对于 RPU，可实现独立应用来启用其 PMU 计数器。将来计划通过 RPU 独立 BSP 来提供支持。

最后，AMD ChipScoPy 可提供 Python API，通过 JTAG 访问寄存器（例如，性能计数器），并提供了用于显示 NoC 性能的应用示例。

提升 PL 中的性能

以下设计特性旨在定义 PL 性能：

- 由 Vivado 实现工具达成的最大时钟频率
- 通过各 RTL 块的时延，引发此类时延的可能原因为有限状态机 (FSM) 或流水线数
- 给定已连接的块所达到的 QoS 和流量模式情况下的 NoC 效率
- 给定热处理设置、设计翻转率、资源使用率、工作时钟频率和目标速度等级情况下的功耗

要在满足功耗预算的同时提升最大 PL 时钟频率，请执行以下操作：

- 使用传统分析和收敛方法
- 分析 NoC 内部的数据移动
- 确认硬件中测量所得带宽和流量与设计创建期间生成的仿真结果是否相匹配

相关信息

设计收敛
改善 NoC 性能

减少时延

设计时延通常取决于 RTL 中存在的流水线量，这些流水线原先用于改善设计的最大时钟频率。流水线类型如下所述：

- 操作特殊原语（例如，DSP、块 RAM 或 UltraRAM）所需的流水线，其频率为器件数据手册中所列的最大频率。

  虽然这些寄存器对于高时钟频率设计至关重要，但对于较慢的设计并不需要所有这些寄存器，因此可移除不需要的寄存器来缩短时延。

- 用于减少给定目标频率的设计中最长路径上的最大逻辑级数或布线级数所需的流水线。

  这些流水线通常映射到 SLICE 寄存器。当器件级别或 SLR 级别（针对 SSI 器件）的寄存器使用率比例超过 50% 后，逻辑布局可能变得更加难以合规，并且 Fmax 可能降级。此外，如果穿过 RTL 模块或设计的总体时延过高，则必须降低含 0 或 1 层逻辑级数的路径上的寄存器使用率，对于局部布局布线的路径尤其如此。

- 用于平衡部分其他路径的时延所需的流水线。

  Versal 器件采用平均分布的 SRL 单元，默认情况下，您必须尽可能多采用这种方式。Vivado 实现工具支持多种物理最优化，可根据需要从 SRL 中抽出寄存器以帮助满足时序约束。

- 由 Vitis HLS 工具引入的流水线，目的是为了提升逻辑层次并最大程度提升按预测的目标频率完成时序收敛的可能性。

  您可以通过在 C/C++ 语言中设置属性来通过 QoS 约束控制特定函数的最大时延。您还可以为 Vitis HLS 降低目标频率，并执行预布局时序分析来验证设计是否能够根据理想布局和逻辑层次信息来满足时序。如需了解有关 Vitis HLS 工具的更多信息，请参阅《Vitis 高层次综合用户指南》(UG1399)。

除了这些类型的流水线寄存器之外，大部分 AMD 或第三方 IP 均可提供接口寄存器、时延或目标频率选项。您必须遵循各 IP 产品指南中提供的准则来优化所有可用设置，以在时延与 Fmax 之间实现最合适的取舍。此外，您可单独综合并实现每个 IP，以确认是否能够实现时序收敛，最好将 Fmax 裕度范围控制在 5 到 15% 之间。

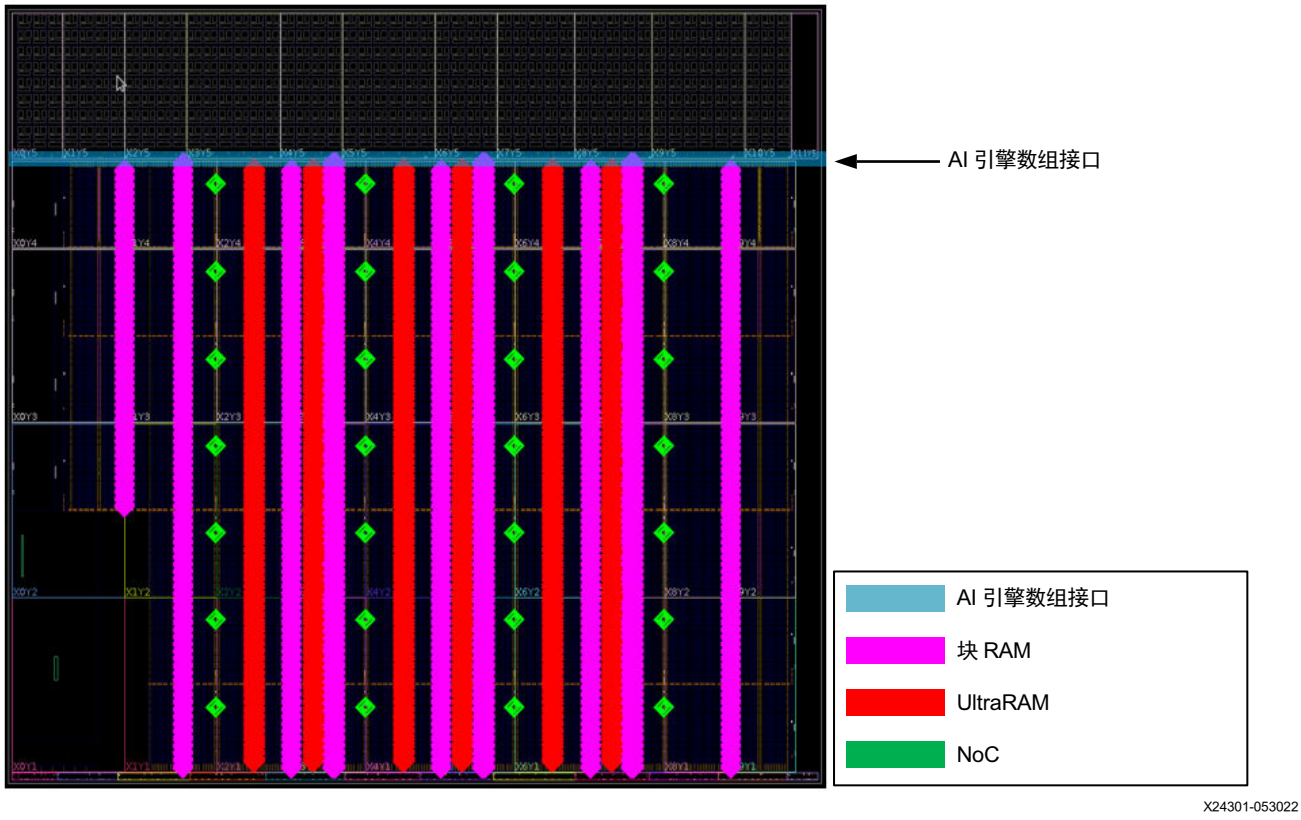
解决布局规划对性能的影响

开始布局规划前，请确保 Versal 器件上的关键资源可供使用。这些资源包括 AI 引擎阵列接口（或 shim 接口拼块）、块 RAM、UltraRAM、DSP、NoC 和 DDRMC。

下图显示了 Vivado IDE 的“Device”（器件）窗口中的 XCVC1902，并高亮显示了关键资源。此图还显示了器件（例如，块 RAM 或 UltraRAM）中的互连结构存储器资源分布情况。根据逻辑的输入/输出串流的连接方式来对齐布局规划对于提升系统性能至关重要。以下提供了逻辑的输入和输出串流的部分示例：

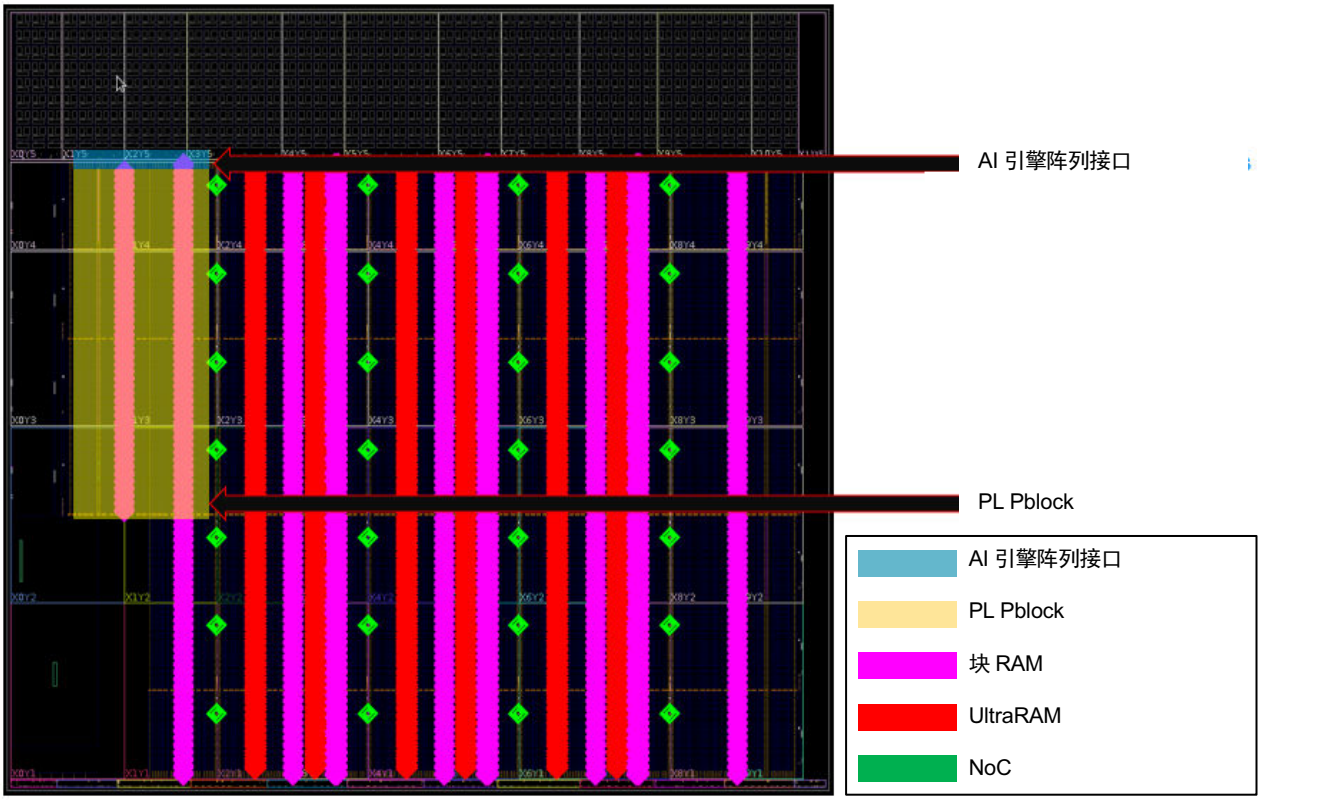
- AI 引擎-PL-NoC-DDRMC
- AI 引擎-PL
- PS-NoC-DDRMC

图 46：**Versal 器件关键资源**



下图显示了 AI 引擎和 PL 的布局规划示例。在此设计中包含 AI 引擎 PL 数据流，并且 PL 不使用 UltraRAM 资源。如果在与 AI 引擎阵列接口交互的数据流中仅使用块 RAM 资源，那么在器件中 PL 所映射到的区域内即使不含 UltraRAM 列，仍可在此区域内执行 AI 引擎串流布局规划。

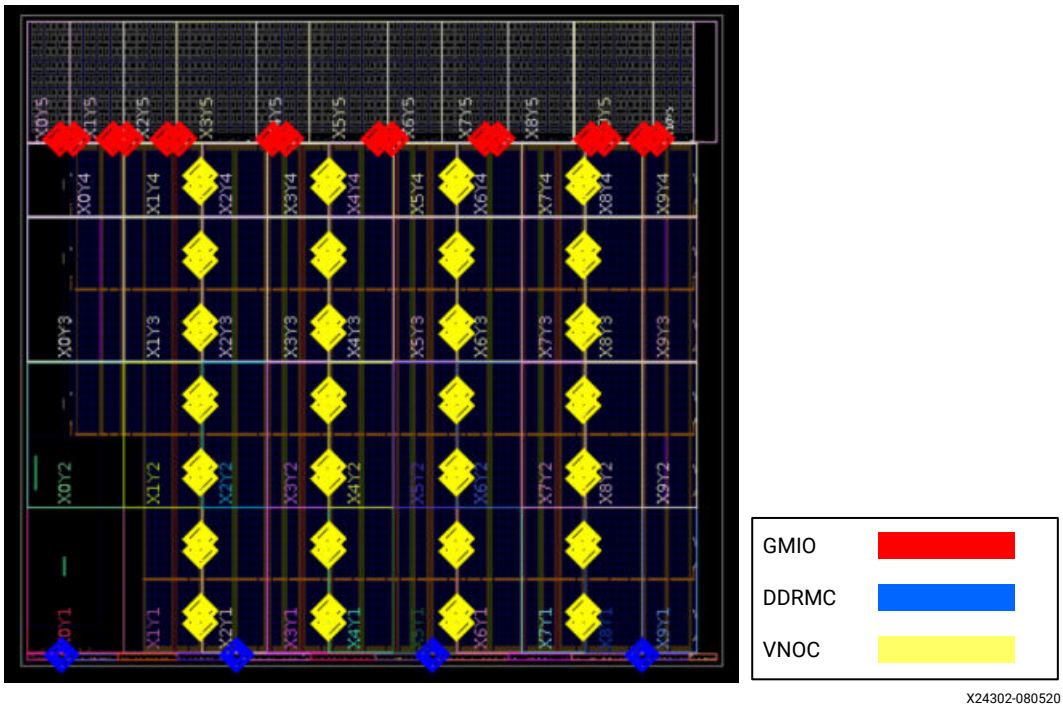
图 47：**AI 引擎和 PL 布局规划**



下图所示设计中显示了通过 NoC 读写 DDRMC 的 AI 引擎中所含的 GMIO。在支持 GMIO 的 AI 引擎中包含一些特定的列。下图显示了“Device”窗口中的 XCVC1902，其中包含 16 个支持 GMIO 的列。以下是访问支持 GMIO 的列时的注意事项：

- 提供适当的 QoS 要求，以便允许 NoC 编译器最大程度提升分配给路径的带宽。
- 允许“aiecompiler”选择 GMIO 列，并允许 NoC 编译器根据 QoS 设置选择相应的 DDRMC 位置。
- 如果需要，请将 GMIO 约束至垂直 NoC (VNoC) 上方相应的列，并约束 VNoc 下方的 DDRMC 以便最大程度缩短穿过 NoC 的时延。

图 48：**支持 GMIO 的列**



以下是对含 AI 引擎和 PL 内核的设计进行布局规划的方法示例：

- AI 引擎阵列接口 Pblock（JSON 文件）

  确定 DDRMC 的宽度，并创建 `<constraints>.json` 文件以供传递到 aiecompiler。

- Pblock（XDC 文件）

  判断 VNoC 的大小，使用基于 VivadoVivado Design Suite XDC 的 Pblock 标准方法。如需了解更多信息，请参阅《Vivado Design Suite 属性参考指南》(UG912) 和《适用于 FPGA 和 SoC 的 UltraFast 设计方法指南》(UG949)。

以下是含指定 Pblock 的 `<constraints>.json` 片段示例。此文件作为实参通过 `--constraints` 选项提供给 AI 引擎编译器。

```
{
     "GlobalContraints":{
          "areaGroup":{
          "name": "unique_area_group_name",
          "exclude": true/false
          "nodeGroup": ["rx.*"],
          "tileGroup": ["(col,row):(col,row)"],
          "shimGroup": ["col: col:]
     }
}
```

在此示例中：

- nodeGroup：列出连接到 Pblock 的所有内核与 PLIO。`rx*` 涵盖了名称以 `rx` 开头的所有实例。
- tileGroup：指定 AI 引擎阵列上用于内核布局的 Pblock 范围。
- shimGroup：指定 AI 引擎阵列上用于阵列接口实例（可以是 PLIO）布局的 Pblock 范围。

以下另提供了其他注意事项：

- 如果从 AI 引擎到 PL 之间存在任何数据流，请查看 DSP、块 RAM 和 UltraRAM 列，并使用 AI 引擎通道，这些通道支持您最大限度利用资源。

  例如，无需 DSP 的 PL 模块则同样不需要距离 DSP 列更近的 AI 引擎通道。

- 请确保根据器件上各流水线阶段的分布方式在宏上插入流水线阶段。

  例如，穿过 AI 引擎 PL 通道的数据流存储在存储器中。这是标准要求，但并没有足够存储器可对齐到 AI 引擎 PL 通道。

- 请根据逻辑的输入和输出串流的连接方式来对齐布局规划。

  例如，对于输出串流，请将布局规划对齐到 PL-NoC-DDR，对于输入串流，请将布局规划对齐到 DDR-NoC-PL-AI 引擎。

改善 NoC 性能

如果根据仿真结果或 NoC 编译器结果质量 (QoR)，的片上网络 (NoC) 性能低于预期，那么请使用本节中的建议来提高 NoC 性能。要执行设计分析并追踪性能问题的根本原因，请按照本节所述执行 NoC 性能测量。

设计注意事项

在尝试设计综合、布局或布线之前，您必须了解本节涵盖的重要设计注意事项。忽略这些设计注意事项很可能导致硬件性能不尽人意。

确保主单元与从单元之间正常通信

为确保正常通信，请检查以下各项：

- 选中“NoC Connectivity”（NoC 连接）选项卡。主单元是否连接到正确的从单元？
- 检查地址编辑器 (Address Editor) 以确保主单元发送到的系统地址范围正确无误。例如，Traffic Generator 设置的地址范围是否正确（与 Address Editor 匹配）？

  对于连接到 NoC 的 LPD，请验证 RPU 是否使用此路径。它只能访问地址范围的前 32 位，不应用于访问高于此范围的任何资源（PL 或 DDR）。AI 引擎是特例，可通过这些工具来处理。

- 检查确认 PMC PLL 参考时钟频率是否正确并与 CIPS Wizard 输入时钟频率相匹配。
- 在 CIPS 的“Clocks”（时钟）选项卡中检查 NoC 的频率。欲知详情，请访问此链接以参阅《Versal Adaptive SoC Programmable Network on Chip and Integrated Memory Controller LogiCORE IP 产品指南》(PG313) 中的相应内容。

提示：实现后，请检查确认存储器控制器是否已通过校准。请访问此链接并参阅《Versal Adaptive SoC Programmable Network on Chip and Integrated Memory Controller LogiCORE IP 产品指南》(PG313) 中的相应内容。

改善 QoS 和带宽

服务质量 (QoS) 能提供一些配置用于确定流量优先级。从高层面来看，流量类别和请求的带宽是 NoC 编译器的约束条件，有助于解决 NoC 资源使用率高的问题。欲知详情，请访问此链接以参阅《Versal Adaptive SoC Programmable Network on Chip and Integrated Memory Controller LogiCORE IP 产品指南》(PG313) 中的相应内容。

注释：虽然流量类别和带宽对于控制 NoC 的行为很有用，但若要尽力达成高优先级类别（低时延、用于读取的常时等量、用于写入的常时等量）的目标，却会对该流量类别产生负作用。也就是说，当在 NoC 内的各种交换元素上存在更高类别的流量时，合作模型就会受到惩罚。

解决次优解决方案

在 AXI-NoC IP 重配置向导的“QoS”选项卡中给带宽要求添加注解时，会将此信息传递给 NoC 编译器。编译器会尝试解决一组带宽要求。然而，并非总能找到解决方案。当无法找到解决方案时，编译器会退回到次优模式，在该模式下，所有带宽都会放宽至可达成的极限。在这种情况下，就会发出以下严重警告，表明解决方案没有反映请求的带宽，无法在硬件上达到设计师的性能目标。

```
CRITICAL WARNING: [Ipconfig 75-4216] A NoC solution that meets the
requested bandwidths could not be found. The following solution relaxes
bandwidth requirements.
```

通常在超额订阅 NoC 中的资源时就会发生这一问题。例如，当垂直 NoC（VNoC）列中的主单元扇出程度较高时，设计可能会在水平 NoC（HNoC）内切换 site 位置。出现这种问题时，您必须复查设计或修改带宽。在实现过程中出现此严重警告时再转而处理硬件，只会适得其反。

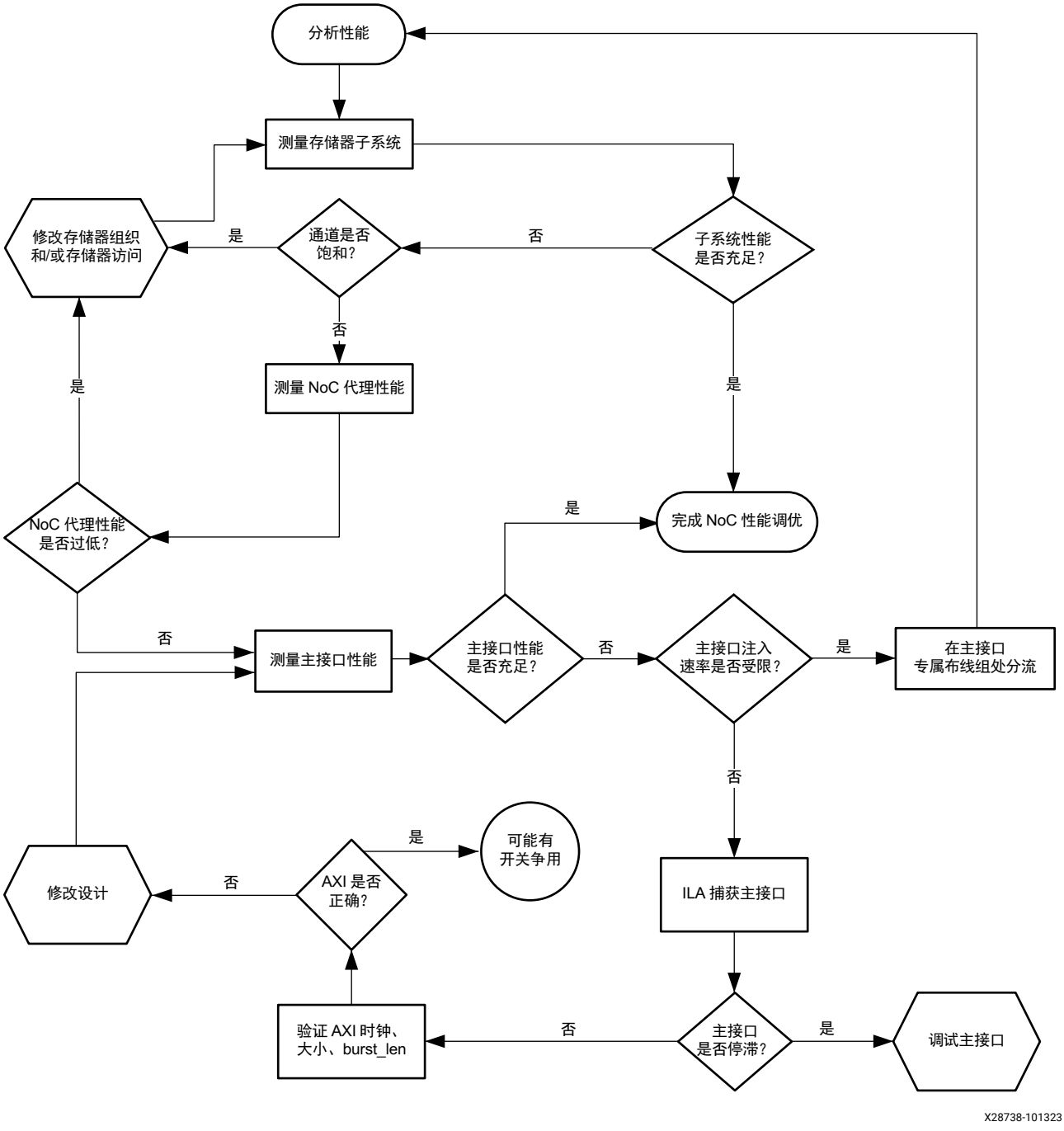
执行 NoC 性能测量

您可使用 ChipScoPy 工具来辅助执行 NoC 性能测量。要安装该工具，请参阅 GitHub 仓库中提供的《ChipScoPy 安装》。安装 ChipScoPy 并将其连接到运行受测设计的硬件后，您可使用该工具来帮助判定 NoC 性能降级的来源。

存储器优先法

NoC 中的许多地方都可能出现性能瓶颈。存储器优先调试方法是判定性能瓶颈原因的推荐方法。在这种方法中，您首先测量存储器子系统，然后返回测量发起流量的主接口。以下流程图显示了诊断和纠正低带宽问题的调试进程。

图 49：**存储器优先法流程**



分析存储器子系统

在 Versal 器件中，有多种类型的存储器子系统。大多数设计都使用高带宽存储器 (HBM)、块 RAM 或 DDR 存储器控制器，或者使用存储器子系统的组合。

分析存储器通道饱和

要判定通道是否饱和，您必须分析通道的性能指标。存储器通道饱和运行状况的关键特征是 ~0% `idle_cycle` 指标计数器，这表示存储器控制器无可用周期。指标计数器小于 10% 也可表示运行状况达到饱和。

造成饱和运行状况的根本原因有几种。例如，存储器子系统布局不当或通用随机存储器访问可能就会导致存储器通道饱和。在某些情况下，饱和运行状况并不表明通道、存储器或 NoC 存在性能问题，而是存储器控制器的突发行为。

下表显示了表明存储器通道饱和的设计特性、根本原因以及应采取的纠正措施。

表 13：**存储器通道饱和分析**

特性	根本原因	纠正措施
bus_turnarounds (%) 过大	读写传输事务高度混用	重新组织访问或者尝试不同的写排序（宽松）
高预充 (%)	bank 缺失数量过多	重新组织访问
高开销 (%)	多种原因	检查突发量和其他 AXI 属性

分析 NoC 代理性能

如果存储器通道性能并非导致 NoC 性能问题的原因，那么下一个检查要点就是 NoC 代理。NoC 代理是 DDR/HBM 存储器控制器的前端，或者对于块 RAM 而言，它则是 PL NSU 的前端。如果 NoC 代理报告称带宽低于预期，原因可能是出现了以下任一情况：

- 上游主接口或主接口遭人为限制。

  您必须测量主接口的性能以判定是否是这个原因。

- 该 site 正出现极高的重合流量。

  解决此问题的方法之一是使用专属布线组。

相关信息

测量主接口性能
使用专属布线组

测量主接口性能

通过检查 Vivado 工具的 NoC 主单元视图中的布线，即可推断出将一个或多个主接口连接至之前分析的存储器端点的布线。如果主接口性能充裕，则很可能是布线上出现了问题。如果判定 NoC 代理低于预期值，则存在汇流效应。出现这种情况意味着在设计的其他地方还必然存在另一个性能低于预期的主接口。

但如果主接口性能的测量结果低于预期比率，则需要进行更多的分析，以正确识别根本原因。

注释：过多交织会给收敛性能带来多种多样的挑战。在这种情况下，提高 NoC 性能的策略是移除交织，减少主接口注入传输事务的数量，以改善存储器绝缘性。

测量主接口注入率

如果主接口无法发出足够的未完成传输事务，就可能人为导致带宽过低。确认 NMU 中的未完成传输事务计数器处于该架构的最大值。

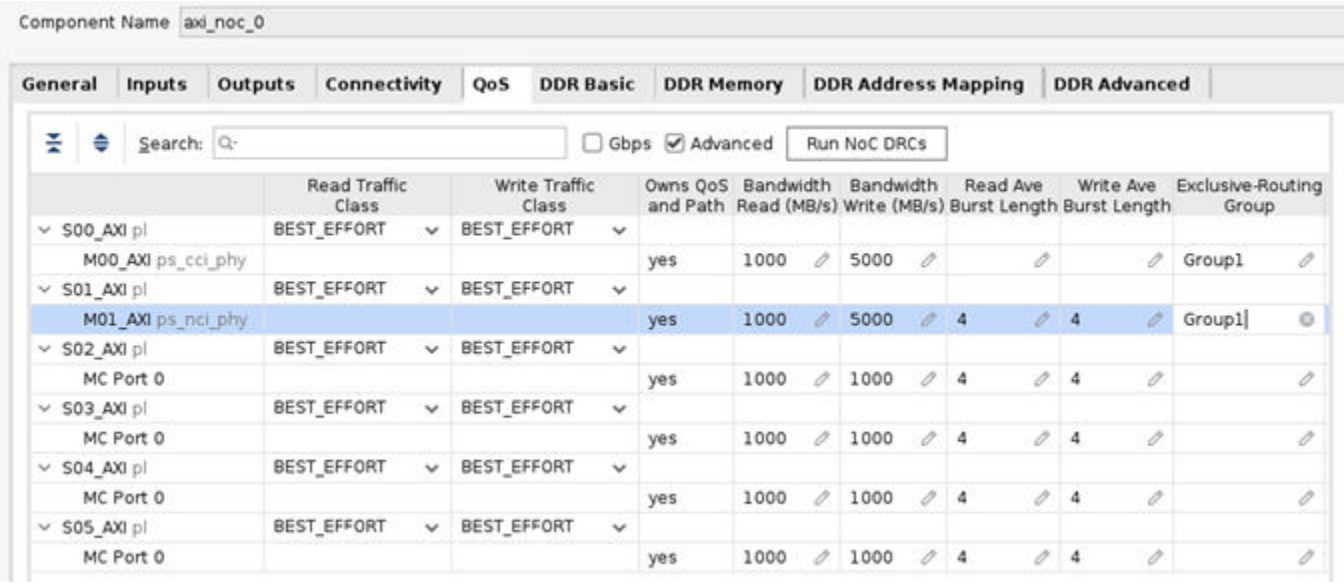
如果注入率低于未完成的传输事务计数器最大值，那么建议重新设计并使用专属布线组。

使用专属布线组

专属布线组会将布线分开，这样 NoC 计算图上的站点就不可供非专属布线组共享，也不可与另一个布线组中的布线共享。这就保证了物理和虚拟的分离。

要创建专属布线组，请打开 AXI-NoC IP 重配置向导的“QoS”选项卡，并根据约束和带宽要求添加分组。

图 50：**AXI-NoC IP 重配置向导中的“QoS”选项卡**



主接口处分流

如果一个主接口负责的目标端点过多，那么注入速率可能会成为限制因素。要解决这个问题，您可以在主接口处拆分流量。通过拆分传输事务并将这些事务转移到不同的主接口，那么注入率就不太可能成为瓶颈。

另一种办法是在系统中添加更多的主接口，并将总工作量更公平地分配给各主接口。使用这种方法重新设计所需工作量可能会很高，但可能是提高 NoC 性能所必需的。

分析主接口传输事务

如果主接口注入率并非造成瓶颈的原因，或者认为重新设计的工作量过高，则应检查主接口行为，确认其是否高效。该分析要求在接口监控模式下对 ILA 进行例化。ILA 必须连接到 M_AXI (S/MM) 主端口。对于 PS 主接口，这可能需要布线到 PL 中。对于其他类型的主接口，这不可行，本节也不适用。有关 ILA 例化、IP 配置和调试技术的信息，请访问此链接以参阅《Vivado Design Suite 用户指南：编程和调试》(UG908) 中的相应内容。

将触发位置设置为窗口的起始位置，并在任务数据通过主接口流向 NoC 时立即发起触发。如果主接口停滞，就表明主接口行为导致部分性能劣化。下一步是直接调试主接口性能。

如果主接口没有停滞，请细查 AXI 传输事务，确认 AXI 突发量、长度（接口的位宽）和寻址模式。通常，突发长度越长、位宽越宽和线性递增的寻址模式都会导致高带宽。同时，在硬件中确认 AXI 时钟正在以指定频率运行。

如需对设计进行任何更改，则需要重新构建和重新测试，以提高 NoC 性能。如果在 AXI 主接口行为中没有识别出问题，那么导致性能问题的原因可能就是 NoC 中的开关特性。

提升 AI 引擎中的性能

有多种技巧可用于剖析和改善 AI 引擎计算图与内核的性能。

您可以使用 Xilinx Runtime (XRT) API 来测量各项性能指标，如平台 I/O 带宽、计算图吞吐量和计算图时延。在主机应用代码中将这些 API 与 AI 引擎计算图对象配合使用。此对象用于初始化、运行、更新和退出计算图。此外，您可使用这些 API 来剖析计算图对象，以测量带宽、吞吐量和时延。欲知详情，请访问此链接以参阅《AI 引擎工具和流程用户指南》(UG1076) 中的相应内容。

AI 引擎性能分析通常涉及各种系统性能问题，如锁定缺失或不匹配、缓冲器溢出以及直接存储器访问 (DMA) 缓冲器编程错误。它还包含存储器/核停滞、死锁以及热点分析。AI 引擎架构可为仿真、硬件仿真或硬件执行期间的事件生成、收集和串流（作为追踪数据）提供直接支持。随后，可对此数据进行分析，以查找内核、存储器停滞、死锁等功能问题和时延问题。欲知详情，请参阅：

- GitHub 仓库中提供了 AI 引擎性能和死锁分析教程
- 请访问此链接并参阅《AI 引擎工具和流程用户指南》(UG1076) 中的相应内容
- 请访问此链接并参阅《AI 引擎工具和流程用户指南》(UG1076) 中的相应内容

AI 引擎 API 对比内部函数

AI 引擎 API 是适用于 AI 引擎加速器的便携式编程接口。它是作为仅限 C++ 头文件的库来实现的，可以提供多种类型和运算，以供转换为高效的低级内部函数。AMD 强烈建议为您的设计使用 AI 引擎 API。仅当设计具有苛刻的性能需求，且需要 AI 引擎 API 中尚未涵盖的功能时，才能考虑使用内部函数。例如，AI 引擎 API 当前不支持 `fft_data_incr` 和 `cyclic_add` 等部分内部函数所提供的功能。虽然 AI 引擎 API 能支持并抽象化主要的置换用例，但并未涵盖所有置换功能。使用内部函数可能使您能够弥补设计所需的性能缺陷。

如需了解有关使用 AI 引擎 API 和内部函数的更多信息，请参阅《AI 引擎内核与计算图编程指南》(UG1079)。

提升通过 CPM 和 PL PCIe 的性能

您可通过检验 I/O 行为和布线、最优化建立时间以最大限度提升性能以及调试性能瓶颈等方式来提升通过 CPM 和 PL PCIe 的性能。

I/O 行为和布线

下列 CPM 端口可供使用：

- AXI4 MM 主端口：仅限在 CPM DMA/AXI4 Bridge 使用模式下才有效：
  - 在 CPM 边界处有 2 个 AXI4 MM 主端口。
    - DMA（XDMA 和 QDMA）：可同时使用 2 个端口。但如果 PCIe 链路聚合吞吐量小于或等于对应此特定器件/硅片速度等级的 NoC NMU 限制，则不要求同时使用这 2 个端口。
    - AXI4 Bridge：仅使用单端口 (AXI-MM0)。如果 AXI4 Bridge 与 DMA 端口搭配使用，那么 AXI4 Bridge 流量将仅使用 AXI-MM0，而 DMA 流量则可使用双端口。
  - 在“Endpoint”（端点）模式下，这 2 个 AXI4 主端口都直接布线至 NoC。

    - 在“Root Port”（根端口）模式（固有 AXI4 Bridge 模式）下，其中一个 AXI4 主端口 (AXI-MM0) 将直接布线至 CCI-500 互连。NoC 可通过 CCI-500 来接入。
- AXI4 MM 从端口：仅限在 CPM DMA/AXI4 Bridge 使用模式下才有效：
    - 在 CPM 边界处仅有 1 个 AXI4 MM 从端口。
        - 全部（包括 XDMA、QDMA 和 AXI Bridge）：可用于访问内部寄存器或者对 PCIe 链路执行总线主控（读写）操作。
    - 在“Endpoint”模式和“Root Port”模式下，此端口均直接连接至 NoC。
- 主从 AXI-ST 端口：仅可在 CPM PCIe 使用模式下使用。
    - 可在“Endpoint”模式和“Root Port”模式下使用。
    - 直接连接至 PL 区域。

下列 PL PCIe 端口可供使用：

- 主从 AXI-ST 端口：仅可在 CPM PCIe 使用模式下使用。
    - 可在“Endpoint”模式和“Root Port”模式下使用。
    - 直接连接至 PL 区域。

性能设置

为了在使用 CPM 时最大程度提升性能，必须考量下列设置：

- AXI4 主端口：由于存在 2 个 AXI4 MM 端口，因此必须适当平衡各数据包并最大程度提升 2 个端口的总线使用率。
    - 用途：
        - 计算 PCIe 链路的聚合吞吐量：计算方法为 PCIe 链路速度 * PCIe 链路宽度。
        - 计算任一端口的 AXI4 MM 端口吞吐量：计算方法为 128 位 * CPMTOPSWCLK 频率（CPM GUI 选项。与速度等级相关，请查询器件/硅片数据手册）。
        - 如果 PCIe 链路吞吐量大于 AXI4 MM 端口吞吐量，则必须同时使用这 2 个端口。

          注释：使用 2 个端口时如果带宽接近相同，则请考量以下因素以及设计复杂性。
            - PCIe 链路具有 TLP 开销，通常为 ~20-25%（取决于数据包大小、最大有效载荷大小以及最大读取请求大小设置）。地址传输未对齐和/或主机存储器分散同样可能因 DMA 传输不足而影响该数值。
            - NoC 在写入侧会因元数据插入导致部分开销，通常为 ~6%，但在读取侧则为接近最优化状态。
            - 如果使用 DDR 存储器，根据流量模式和 DDR 存储体/列/行设置，可能导致额外开销。
    - 用法：
        - 数据包不得拆分到 2 个端口。这些数据包必须尽可能独立运作，以避免因 AXI4 ID 和 PCIe 标签排序而导致的队头阻塞。
            - QDMA：根据队列 ID 拆分流量。将部分队列分配至使用首个 AXI4-MM0，将其余队列分配至使用第二个 AXI4-MM1。
            - XDMA：必须根据 DMA 通道 ID 自动拆分流量。将双数 DMA 通道布线至 AXI4-MM0，将单数 DMA 通道布线至 AXI4-MM1。

- AXI4 Bridge：仅使用单端口 AXI4-MM0。因此，性能最大应不超过 AXI4 MM 端口吞吐量，并且可能无法达到 PCIe 链路吞吐量。

- AXI4 MM 从端口：由于只有 1 个 AXI4 MM 端口，通过此端口的性能最大应不超过 AXI4 MM 端口吞吐量，并且可能无法达到 PCIe 链路吞吐量。
- 主从 AXI4-ST 端口：由于 CPM 在此模式下只能使用 AXI4-ST 端口直接连接到 PL，因此您运行自有设计时的频率和数据总线宽度只需与来自 CPM 或 PCIe PL IP 的 AXI4-ST 接口相同即可。

PL PCIe 只能使用 AXI4-ST 端口，因此您运行自有设计时的频率和数据总线宽度只需与来自 CPM 或 PCIe PL IP 的 AXI4-ST 接口相同即可。

调试步骤

性能瓶颈可能会在各处出现。大部分问题均可分类为 4 个不同调试部分：

- 主控制器：启动传输的器件。
  - 请检查其功能。方法是计算 AXI 数据总线宽度 * 该主控制器的 AXI 频率。
  - 如果 CPM 作为主控：
    - 请检查它是同时使用 2 个 AXI4-MM 端口，还是仅使用单个 AXI4-MM 端口。计算通过这些端口的聚合带宽。
    - 检查 CPMTOPSWCLK 频率。CPM AXI4-MM 端口位宽固定为 128 位。
    - 检查流量模式。确保数据包按传输 ID（可以是队列 ID 或通道 ID）拆分。请勿拆分使用相同 AXI4 ID 的数据包。
- 从设备：接收传输的器件。
  - 请检查其功能。方法是计算 AXI4 数据总线宽度 * 该从设备的 AXI4 频率。
  - 如果 CPM 用于接收传输：
    - CPM 仅有 1 个 AXI4-MM 端口。计算通过此端口的带宽。
    - 检查 CPMTOPSWCLK 频率。CPM AXI4-MM 端口位宽固定为 128 位。
    - 检查流量模式。从端口用于访问内部寄存器（在内部使用 AXI4-Lite 接口且有 1 项未完成的 AXI4 传输事务），且用于对 PCIe 链路执行总线主控（读写）操作。请勿将这些传输事务目标进行交织，以免发生队头阻塞。
- 互连：数据包必须经过的所有互连和开关。
  - 分析传输路径中的所有互连。包括 NoC、CCI-500、SmartConnect 等。请检查其吞吐量能力。方法是计算 AXI4 数据总线宽度 * 这些互连的 AXI4 频率。
  - 检查 AXI4 未完成的传输事务设置。系统时延越高或者 AXI4 数据包越小，则此数值应越高，以免信用值不足。
  - 检查流量模式。确保“慢”数据路径和“快”数据路径不会交织，以免发生队头阻塞。
- 外部/软件：超出 AMD 器件范围的因素。
  - 软件/驱动/应用：软件和驱动通常比硬件慢得多。为了最大程度提升吞吐量，您必须确保传输期间软件具有最低“维护”要求：
    - 最大程度提升可用的描述符队列环大小或描述符链大小。
    - 最大程度提升传输大小（包括主机的最大有效载荷大小和最大读取请求大小设置）。

- 最大程度提升队列和 DMA 通道数量。
- 最大程度减少中断并避免过度使用轮询模式。
- 最大程度减少指针或硬件更新。
- 避免从软件执行总线主控。交由硬件来执行 DMA 或总线主控。
- 避免过度从用户复制到内核层级存储器。指定主机上的某一存储器用于传输。

- 开关/IOMMU/处理器链路：当传输事务进出主机时，有各种常用模块沿这些路径传输，这可能导致时延增加，从而对系统性能产生不利影响：
  - 请选择包含 PCIe 开关最少的路径。分析可用 PCIe 插槽及其总线拓扑结构。确保软件或驱动在连接到该 PCIe 总线的 CPU 上运行。使用直接连接到该 CPU 的存储器器件（磁盘、DDR 存储器等）。
  - 所选 CPU 和 PCIe 开关应具有较多 PCIe 信用值，或者可使用扩展 PCIe 标签。这样会显著提升未完成的 PCIe 数据包数量，因此需要对主机的高时延进行补偿。企业级系统传输的值通常比桌面系统或工作站系统更高。
  - 确保有效的 PCIe 插槽和 PCIe 开关的链路宽度和速度与您的器件相匹配。验证 PCIe 链路是否已训练至最优链路速度和宽度。
  - 在多功能器件中通常需要 IOMMU，但它会增加用于转换这些 PCIe 地址的时延。如非必要，请避免依赖 IOMMU。
  - 在 PCIe 和 CPU 上禁用低功耗状态。虽然这些功能能够节省功耗（在大型数据中心环境内尤其如此），但重复进出这些功耗状态可能导致减缓传输速度并增加时延。

注释：如需了解有关调试的更多信息，请参阅 GitHub 中的 PCIe Debug K-Map。

限制

以下是为 CPM 或 PL PCIe 分析性能时存在的限制：

- 仿真中的性能调试可能无法反映整个系统行为。仅对 CPM 块进行仿真可生成较为准确的性能模型，但您必须注意下列仿真模型限制：
  - 高功耗域 (HPD) 和 PS9 均采用 BFM 来建模。此 BFM 无法以周期精确的方式来呈现硬件。此 BFM 还可能要求您根据自己的 IP 设置来手动设置参数，并且可能无法将相同的值传播至整个硬件中。此 BFM 可能在某些事件中更改所使用的时钟频率以加速仿真。或者为便于操作，它可能以单时钟域进行建模，但在硬件内则不会如此。
  - 将 CPM 或 PL PCIe 作为端点 (Endpoint) 进行仿真时，AMD 会提供根端口 (Root Port) PCIe 模型。此模型并非 BFM，它基于 PL PCIe IP 架构，大多数情况下其响应周转时间比常规主机系统更快。
- CPM、PS 或 NoC 块内部的 ILA 无法用于执行硬件探测，但 NoC NMU 和 NSU 可提供内部统计数据（例如，数据包计数）。但是，您必须谨记，数据流水线中的任一瓶颈最终都会散播到整个数据路径中。即，如果对从端口进行节流，那么互连和主端口都可能呈节流状态。因此，此数据可能需要搭配其他数据进行进一步限定，然后才能作出最终决策。
- 同样，在 DMA 操作中，所有操作都是环环相扣的。如果对软件或主机进行节流，那么硬件最终也将节流，反之亦然。

第 5 章

配置与调试

成功完成设计实现后，下一步就是将设计加载到器件中并在硬件上运行。配置是指将特定应用的数据加载到器件内部存储器中的过程。如果设计在硬件上不满足要求，则需要进行调试。

请参阅以下资源，以获取有关配置和调试软件流程与命令的详细信息：

- 《Vivado Design Suite 用户指南：编程和调试》(UG908)
- 《Vivado Design Suite Tcl 命令参考指南》(UG835)

创建器件镜像

完成综合和实现后，必须创建器件镜像，然后才能在目标器件上运行设计。AMD Versal™ 器件架构使用可编程器件镜像 (.pdi) 文件格式的器件镜像。您可使用 `write_device_image` Tcl 命令或者单击 AMD Vivado™ IDE 中的“Generate Device Image”（生成器件镜像）来创建器件镜像。创建器件镜像前，将运行 DRC，并且早期设计阶段生成的所有“Critical Warnings”（严重警告）都将升级为“Errors”（错误），从而阻止创建器件镜像。如需了解有关创建器件镜像的更多信息，请参阅《Vivado Design Suite 用户指南：编程和调试》(UG908)。

对于 DFX 设计，`write_device_image` 命令支持 `-cell` 开关，此开关接受以实参形式提供的可重配置模块名称。如果不指定 `-cell`，那么 `write_device_image` 会生成完整 PDI 文件和部分 PDI 文件。如无需部分 PDI 文件，则可使用 `-no_partial_pdifile` 开关仅生成完整 PDI 文件。对于 Versal 器件，默认情况下，为部分 PDI 文件启用文件压缩和逐帧循环冗余校验 (CRC)。如需了解有关适用于硬件的部分 PDI 交付机制的详细信息，请参阅《Vivado Design Suite 用户指南：Dynamic Function eXchange》(UG909)。

重要提示！创建设计 PDI 文件前，除了先要解决所有 DRC 错误，还必须确保满足时序要求，并解决重要的方法论检查。`report_timing_summary` 命令可提供时序裕量信息，`report_methodology` 命令可标记设计或约束问题，这些问题可能导致硬件不稳定或硬件故障。

注释：如果设计使用了通过 CIPS PMC IP 启用的高级器件管理功能，那么您必须运行 Bootgen 工具来对 PMC 固件完成 PDI，这与嵌入式软件开发流程类似。欲知详情，请访问此链接以参阅《Vitis 统一软件平台文档：嵌入式软件开发》(UG1400) 中的相应内容。

相关信息

修复 report_methodology 标记的问题

配置

您必须首先成功完成设计综合与实现，然后才能创建可编程器件镜像 (PDI)。生成 PDI 并且对所有 DRC 都完成分析和更正后，即可使用以下任一方法将 PDI 加载到器件上：

- 直接编程：通过线缆、处理器或定制解决方案将 PDI 直接加载到器件。
- 间接编程：将 PDI 加载到外部闪存。闪存再将 PDI 加载到器件。

可使用 Vivado 工具来完成下列操作：

- 创建 PDI (`.pdi`)。
- 选择“Tools” → “Edit Device”（工具 > 编辑器件）以复查 PDI 生成的配置设置。
- 使用以下任一方法对器件进行编程：
  - 直接对器件进行编程。
  - 间接对连接的配置闪存器件进行编程。

    闪存器件是非易失性的器件，编程前必须进行擦除。

> **重要提示！** Vivado Design Suite Device Programmer 可使用 JTAG 读取 AMD 器件上的 JTAG_STATUS 和 ERROR_STATUS 寄存器数据。如发生配置故障，则 JTAG_STATUS 和 ERROR_STATUS 寄存器会捕获具体的错误条件，以帮助查明故障的原因。此外，JTAG_STATUS 寄存器允许您验证模式管脚设置 MODE[3:0]、检查是否可检测到关键电源并确定 SelectMAP 总线宽度。如需了解有关 JTAG_STATUS 和 ERROR_STATUS 寄存器的详细信息，请参阅《Versal 自适应 SoC 技术参考手册》(AM011)。

调试

系统内调试允许您在目标器件上实时调试设计。遇到几乎无法复制仿真器的情况时，就需要执行此步骤。

就调试而言，您需要为设计提供专用调试 IP 以便观察和控制设计。调试完成后，可将仪器或专用 IP 移除，从而提升性能和逻辑缩位。

设计调试是 1 个多步骤的迭代过程。就像大部分复杂难题一样，最好将设计调试过程细分为几个小部分，以便集中精力使设计中的每一小部分能逐一正常运行，而不是尝试一次性让整个设计都能正常运行。

虽然实际调试步骤是在成功实现设计后执行的，但 AMD 建议在设计周期初尽早规划调试方式和对象。您可从 Vivado IDE 的 Flow Navigator 窗口中的“Program and Debug”（编程和调试）部分运行所有必要的命令以执行器件编程和设计的系统内调试。

调试步骤如下所述：

1. 探测：确定设计中要探测的信号和探测的方法。
2. 实现：完成设计实现，包括连接到所探测的信号线上的其他调试 IP。
3. 分析：与设计中包含的调试 IP 进行交互，以便对功能问题进行调试和验证。
4. 修复相位：修复所有缺陷，此步骤可按需重复。

如需了解更多信息，请参阅《Vivado Design Suite 用户指南：编程和调试》(UG908)。

> **提示：** AMD 提供了开源 ChipScoPy API，它支持对硬件中运行的 Versal 器件调试 IP 进行高级别控制，以便您使用 Python API 创建自己的调试应用。通过使用简单的 Python API，您就可以对 ChipScope™ 调试进行控制和通信，例如，Integrated Logic Analyzer (ILA)、Virtual Input/Output IO (VIO)、器件存储器访问等。欲知详情，请访问位于以下地址的 GitHub 仓库：http://www.github.com/Xilinx/chipscopy。

调试配置

如需了解有关调试配置的信息，请参阅 Versal 器件编程/启动调试检查表，其中包含调试 Versal 器件配置和启动时需要考量的各项要求、操作和要点的常规列表。您可使用此检查表在全新的 Versal 开发板上调试基本可编程器件镜像 (PDI) 的首次加载。

以 JTAG 启动模式加载 PDI 成功后，下一步就是对目标启动器件进行编程和启动。如果您当前不使用 JTAG 启动模式，请参阅启动和配置设计中心，获取其他相关资源和检查表，例如，SelectMAP 启动检查表。

调试开发板初始化

要确定开发板初始化问题的潜在根本原因，您必须了解该问题在整个启动流程中的位置。本节简要介绍了 Versal 器件从上电到 Linux 内核部署的启动流程。该流程包括调试检查点，可帮助您确定开发板初始化问题的根本原因。

BootROM 调试

上电时，BootROM 代码处于运行状态。BootROM 代码负责读取启动模式寄存器，然后从静态闪存器件（如 SD 卡或 QSPI）中提取启动镜像。如果您给开发板上电，但串行端口没有响应，那么 AMD 建议读取 Vivado 硬件管理器或 XSCT 中的 ERROR_STATUS 寄存器。典型的错误是找不到闪存镜像或该镜像损坏。BootROM 会将 Platform Loader and Manager (PLM) 加载到平台处理单元 (PPU) RAM 中，并负责将 PPU 解复位和交接给 PLM。

要检查潜在 BootROM 问题，请使用启动调试博文中提供的脚本。为判定特定问题的根本原因，该脚本会将 BootROM 错误与可用的 BootROM 错误编码进行交叉比对，如需获取错误代码，请访问此链接以参阅《Versal 自适应 SoC 技术参考手册》(AM011) 中的相应内容。

Versal 器件编程/启动调试检查表可用于复查在调试 Versal 器件配置属性和启动时需要考量的各项要求、操作和要点。您可使用此检查表在全新的 Versal 自适应 SoC 开发板上调试基本 PDI 的首次加载。

重要提示！ BootROM 代码是 Versal 自适应 SoC 的内部代码，无法更改。

Platform Loader and Manager 调试

Platform Loader and Manager (PLM) 可通过 JTAG 加载，或者也可通过 BootROM 从静态闪存器件加载。PLM 的角色是利用 Vivado Design Suite 中生成的二进制文件配置 Versal 器件（处理器子系统和可编程逻辑）。这些二进制文件会以各种格式交付。例如，CIPS 中使用 Vivado IP integrator 创建的用户配置会作为输入（CDO 文件）传递给 PLM。可编程逻辑二进制文件会作为 RCDO 文件来传递。此 PLM 会在平台管理控制器 (PMC) 中的平台处理单元 (PPU) MicroBlaze™ 处理器上执行。PLM 可执行文件通常是在 Vivado Design Suite 中生成的，并封装在 XSA 文件中。此 PLM 会提取这些输入文件（例如，CDO 文件），在文件中解析并执行各项命令。PLM 和二进制文件会封装进单个二进制文件，称为可编程器件镜像 (PDI) 文件或 BOOT.BIN 文件。

PLM 最常见的错误场景是在执行某个二进制文件的过程中，如，执行含寄存器超时的 CDO 或 RCDO 时。例如，在 RCDO 文件中的 DDR 存储器配置中，会轮询 DDR 存储器校准寄存器询，直到其转为高电平为止。如果校准不转为高电平，PLM 可能会出现问题，导致出现错误消息。如需了解有关这些错误消息的更多信息，请参阅 Versal Platform Loader and Manager 错误代码维基百科页面。

在典型的 Linux 启动流程中，PDI 包含处理器子系统、可编程逻辑二进制文件和辅助启动镜像，例如，U-BOOT ELF 文件。辅助启动镜像包含 Arm® 可信固件 (ATF) 和设备树二进制对象，默认偏移为 0x1000。PLM 交接给 ATF，ATF 再交接给 U-BOOT。

ATF 和 U-Boot 调试

在 Versal 器件中，Arm 可信固件 (ATF) 可提供与其他 SoC 相似的访问和功能，还可提供 Versal 器件专用的访问和功能。为便于操作系统（OS）访问这些底层功能，必须对操作系统进行修改，以支持 ATF 向操作系统导出的安全监控调用。ATF 交接给 U-Boot，ATF 不常出现问题。除非启用调试，否则由片上存储器 (OCM) 执行 ATF。如果启用调试，ATF 源代码库会增大，并改由 DDR 存储器执行。如果 ATF 启用调试，但您在串行端口上看不到预期的 ATF 打印语句，这可能表明存在与 DDR 存储器有关的问题。U-Boot 的主要用途是充当辅助启动加载程序，以加载并交接 Linux 内核。

典型用途是从 SD 卡启动。SD 卡分区包含 `image.ub` 和 `boot.scr`。`image.ub` 是使用名为 mkimage 的 U-Boot 实用工具在 PetaLinux 中创建的平铺镜像树 (FIT) 镜像。FIT 镜像通常包含内核镜像、设备树二进制对象 (dtb) 以及（如果使用）基于 RAM 的文件系统，供内核使用。`image.ub` 具有镜像头文件元数据，U-Boot 会从中读取有关如何加载和执行内部镜像的信息。U-Boot 是从 DDR 存储器执行的，它会读取设备树中的存储器节点，并使用此信息将自身重定位至 DDR 存储器的上半部分。

如果 ATF 执行后什么也没有发生，可能原因是存储器问题，建议执行存储器测试。如果 U-Boot 启动后不久就挂起，那么您必须调查挂起的原因。如需了解有关调试 ATF 和 U-Boot 的信息，请参阅 PetaLinux 镜像调试系列：在 Vitis 中调试 Arm 可信固件和 U-Boot 一文。

Linux 内核调试

内核软件由汇编程序与 C 语言代码混合而成，并从 U-Boot 加载。U-Boot 交接给入口点 head.S，后者会检查架构、处理器和机器类型。`head.S` 还会配置内存管理单元 (MMU)、创建页表和设置虚拟内存，然后调用 `start_kernel`。start_kernel 函数是由约 86 个子系统构成的。

一个关键启动里程碑就是 setup_arch。在 `setup_arch` 中，会发生与架构相关的所有初始化操作，如设置 CPU 和设置机器，这些都是从设备树中提取的。下一步，`setup_arch` 初始化早期控制台和多个子系统，如存储器分配、调度和文件高速缓存等内核服务，直至到达 rest_init。kernel_init 会运行 do_basic_setup 并尝试在文件系统中调用 init，这会将当前内核线程更改为 `init` 进程。如果此时出现问题，很可能无法找到或无法装载文件系统。如果 Linux 工程是在 PetaLinux 中创建的，您可以检查 PetaLinux 配置中的文件系统。

如果在内核初始化中发生问题，您可以在这些子系统的启动 log 日志中对其逐一进行跟踪，以调试内核中的启动问题。如需了解更多信息，请参阅 PetaLinux 镜像调试系列：在 Vitis 中调试 Linux 内核一文。

调试系统

在 Versal 器件上对可编程器件镜像 (PDI) 进行编程时，必须确保器件准备就绪，可随时开始编程和器件初始化。由于 Versal 器件编程使用 PMC，因此编程前确保 PMC 块已上电且系统时钟保持稳定至关重要。确保配置前镜像所使用的功耗域已完全上电同样至关重要。最后，务必确保器件未处于复位状态。

连接到器件时，如果在软件工具中未检测到器件名称，那么对于 vc1902 器件，软件可能显示类似于“XVJTAG40”的名称。通常器件保持处于复位状态时会发生此类情况。在此类情况下，请首先检查复位线和电源。如果这些都处于稳定状态，那么第二种可能的原因是器件锁定。当器件未采用 JTAG 模式启动时，可能导致配置引擎进入锁定状态，从而引发上述状况。要修复此问题，请将器件模式管脚设置为 JTAG，然后重新启动。

当器件达成稳定状态后，对 PDI 进行编程可能会报告错误。源于 PLM 的错误可能会报告从接口启动错误。出现此类消息时，重要的是将 UART 连接到系统以获取错误的详细报告。

JTAG 状态和错误状态

您可使用来自 XSCT 的 `jtag_status` 和 `error_status` 命令捕获整体 PMC 状态和错误状态。

```
xsct% device status jtag_status
```

- 检查 BOOT_MODE [15:12] 位
- 检查 DONE [34] 位 - 这表示配置已完成
- 确保已成功检测到各电压电源 - 位 [8:11] 应断言有效

```
xsct% device status error_status
```

- 当开发板完成掉电并重新上电，并且上电复位 (POR) 断言有效后，期望的 `error_status` 值为 `0x0`

表 14：**JTAG_STATUS 寄存器格式**

位	字段	描述
35	RESERVED	保留
34	DONE	启动和配置状态指示符 DONE 值为 1 表示启动和配置已完成
33	JRDBK ERROR	JTAG 回读状态指示符 JRDBK 值为 1 表示从 SBI 读取数据时出错
32	JCONFIG ERROR	JTAG 数据加载错误指示符 值为 1 表示 SBI 尚未准备好接受数据
31:28	PMC VERSION	PMC 版本
27:24	RESERVED	保留
23	JTAG SEC GATE	安全性门电路状态 值为 1 表示允许 DAP AXI 传输事务
22	RESERVED	保留
21	PMC SCAN CLEAR DONE	指示扫描清除已完成 值为 1 表示扫描清除已完成
20	PMC SCAN CLEAR PASS	指示扫描清除已通过 值为 1 表示扫描清除已通过
19:16	RESERVED	保留
15:12	BOOT MODE [3:0]	释放 POR_B 时，从 MODE 管脚采集的启动模式值
11	VCC_PMC DETECTED	已检测到 VCC_PMC 电源
10	VCC_PSLP DETECTED	已检测到 VCC_PSLP 电源
9	VCCINT DETECTED	已检测到 VCCINT 电源
8	VCC_SOC DETECTED	已检测到 VCC_SOC 电源
7	AES KEY ZEROIZED	AES 密钥补零指示符 值为 1 表示所有密钥均已补零
6	BBRAM KEY ZEROIZED	BBRAM 密钥补零指示符 值为 1 表示 BBRAM 密钥已补零
[5:4]	SELECTMAP BUS WIDTH	已检测到 SelectMAP 启动模式总线宽度 00 = 未检测到总线宽度 01 = SelectMAP 8 位 10 = SelectMAP 16 位 11 = SelectMAP 32 位
3	SBI JTAG ENABLED	SBI JTAG 指示符 值为 1 表示 SBI 已配置为接收来自 JTAG 接口的数据
2	SBI JTAG BUSY	SBI JTAG BUSY 指示符 值为 1 表示 SBI 处于繁忙 (BUSY) 状态，且在 JTAG 模式下无法接受数据
1	RSVD_READS_0	保留，返回 0

表 14：**JTAG_STATUS 寄存器格式** (续)

位	字段	描述
0	RSVD_READS_1	保留，返回 1

表 15：**ERROR_STATUS 寄存器格式**

位	字段	描述
159:155	RSVD_READS_0	保留，返回 0
154:148	RESERVED	保留
147:136	BOOTROM FIRST ERROR	检测到 BootROM 首次出错代码 (FEC)
135:124	BOOTROM LAST ERROR	检测到 BootROM 末次出错代码 (LEC)
123:110	PLM MAJOR ERROR	PLM 主要错误代码
109:94	PLM MINOR ERROR	PLM 次要错误代码
93:64	GSW ERROR	PLM 的常见软件错误
63	RESERVED	保留
62	BOOTROM NCR	BootROM 不可纠正错误 由 RCU BootROM 在启动期间设置
61	PLM CR	Platform Loader and Manager 启动时可纠正错误 由 PLM 在启动期间设置
60	PLM NCR	Platform Loader and Manager 启动时不可纠正错误 由 PLM 在启动期间设置
59	GSW CR	启动后常见软件可纠正错误
58	GSW NCR	启动后常见软件不可纠正错误
57	CFU ERROR	CFU 错误
56	CFRAME ERROR	CFRAME 错误
55	PSM CR	PSM 可纠正错误
54	PSM NCR	PSM 不可纠正错误
53	DDRMC MB CR	DDRMC MicroBlaze 可纠正 ECC 错误
52	DDRMC MB NCR	DDRMC MicroBlaze 不可纠正 ECC 错误
51	NOC CR	NoC 不可纠正错误
50	NOC NCR	NoC 不可纠正错误
49	NOC USER ERROR	NoC 用户错误
48	MMCM LOCK ERROR	MMCM 锁定错误
47	AIE CR	AI 引擎可纠正错误
46	AIE NCR	AI 引擎不可纠正错误
45	DDRMC MC ECC CR	DDRMC MC（存储器控制器）可纠正 ECC 错误
44	DDRMC MC ECC NCR	DDRMC MC（存储器控制器）不可纠正 ECC 错误
43	GT CR	GT 可纠正错误
42	GT NCR	GT 不可纠正错误
41	SYSMON CR	SYSMON 可纠正错误
40	SYSMON NCR	SYSMON 不可纠正错误
39	USER PL0 ERROR	用户定义的 PL 错误

表 15：**ERROR_STATUS 寄存器格式** (续)

位	字段	描述
38	USER PL1 ERROR	用户定义的 PL 错误
37	USER PL2 ERROR	用户定义的 PL 错误
36	USER PL3 ERROR	用户定义的 PL 错误
35	NPI ROOT ERROR	NPI 根错误
34	SSIT ERROR3	SSI 技术 SLR 错误
33	SSIT ERROR4	SSI 技术 SLR 错误
32	SSIT ERROR5	SSI 技术 SLR 错误
31	PMC APB ERROR	PMC APB 错误。包括来自寄存器的错误：PMC_LOCAL、PMC_GLOBAL、CRP、PMC_IOP_SECURE_SLCR、PMC_IOP、BBRAM_CTRL、PMC_ANLG 和 RTC
30	PMC BOOTROM ERROR	PMC BootROM 确认错误
29	RCU HARDWARE ERROR	RCU 硬件错误
28	PPU HARDWARE ERROR	PPU 硬件错误
27	PMC PAR ERROR	PMC 开关和 PMC IOP 奇偶校验错误
26	PMC CR	PMC 可纠正错误
25	PMC NCR	PMC 不可纠正错误
24	PMC SYSMON0 ALARM	PMC 高温关机警报和电源故障检测错误（来自 SYSMON）
23	PMC SYSMON1 ALARM	PMC 高温关机警报和电源故障检测错误（来自 SYSMON）
22	PMC SYSMON2 ALARM	PMC 高温关机警报和电源故障检测错误（来自 SYSMON）
21	PMC SYSMON3 ALARM	PMC 高温关机警报和电源故障检测错误（来自 SYSMON）
20	PMC SYSMON4 ALARM	PMC 高温关机警报和电源故障检测错误（来自 SYSMON）
19	PMC SYSMON5 ALARM	PMC 高温关机警报和电源故障检测错误（来自 SYSMON）
18	PMC SYSMON6 ALARM	PMC 高温关机警报和电源故障检测错误（来自 SYSMON）
17	PMC SYSMON7 ALARM	PMC 高温关机警报和电源故障检测错误（来自 SYSMON）
16	PMC SYSMON8 ALARM	PMC 高温关机警报和电源故障检测错误（来自 SYSMON）
15	PMC SYSMON9 ALARM	PMC 高温关机警报和电源故障检测错误（来自 SYSMON）
14	CFI NCR	CFI 不可纠正错误
13	SEU CRC ERROR	SEU CRC 错误
12	SEU ECC ERROR	SEU ECC 错误
11:10	RSVD_READS_0	保留，返回 0
9	RTC ALARM	RTC 警报错误
8	NPLL ERROR	PMC NPLL 锁定错误
7	PPLL ERROR	PMC PPLL 锁定错误
6	CLOCK MONITOR ERROR	时钟监控器错误
5	PMC TIMEOUT ERROR	PMC 互连超时错误（来自互连任务中断状态寄存器、互连时延状态寄存器和超时中断状态寄存器）
4	PMC XMPU ERROR	PMC XMPU 错误（来自 APB 上的寄存器访问错误）。包括读取权限违例、写入权限违例或安全性违例
3	PMC XPPU ERROR	PMC XPPU 错误（来自 APB 上的寄存器访问错误）。包括：未找到主 ID、读取权限违例、主 ID 访问违例、主 ID 奇偶校验错误和 TrustZone 违例
2	SSIT ERROR0	SSI 技术 SLR 错误
1	SSIT ERROR1	SSI 技术 SLR 错误

表 15：**ERROR_STATUS 寄存器格式** (续)

位	字段	描述
0	SSIT ERROR2	SSI 技术 SLR 错误

电源轨电压状态

确认已成功检测到电源轨。要确保开发板上的电源轨电压值配置正确，请读取 SysMon 寄存器。`0xF1110100` (PWR_STATUS) 和 `0xF111010C` (PWR_SUPPLY_STATUS) 均为给电源轨供电的寄存器。

要读取这些寄存器，请运行以下命令：

```
xsct%mrd -force 0xF1110100
xsct%mrd -force 0xF111010C
```

确认已成功检测到电源轨。要确保开发板上的电源轨电压值配置正确，请运行以下命令以读取 SysMon 寄存器 `0xF111010C` (PWR_SUPPLY_STATUS)：

```
xsct%mrd -force 0xF111010C
```

表 16：**11.1.21 寄存器 (PMC_GLOBAL) PWR_STATUS**

寄存器名称	地址	宽度	类型	复位值	描述
PWR_STATUS	0xF1110100	32	RO	0x00000000	此寄存器可提供“Domain Isolation Status”（域隔离状态）。 1 = 已隔离 0 = 未隔离 此寄存器只能通过上电复位进行复位，在系统复位或内部上电复位期间其状态保持不变。

表 17：**寄存器 PWR_STATUS 位字段详细信息**

字段名称	位	类型	复位值	描述
保留	31:19	RO	0x0	保留
VCCAUX_VCCRAM	18	RO	0x0	VCCAUX-VCCRAM 接口和信号
VCCRAM_SOC	17	RO	0x0	VCCRAM-SoC 接口和信号
VCCAUX_SOC	16	RO	0x0	VCCAUX-SoC 接口和信号
PL_SOC	15	RO	0x0	PL-SoC 接口和信号
PMC_SOC	14	RO	0x0	PMC-SoC 接口和信号
PMC_SOC_NPI	13	RO	0x0	PMC-SoC NPI 接口和信号
PMC_PL	12	RO	0x0	PMC-PL 接口和信号
PMC_PL_CFRAME	10	RO	0x0	PMC-PL 接口和信号
PMC_LPD	9	RO	0x0	PMC-LPD 接口和信号
LPD_PL	6	RO	0x0	LPD-PL 接口和信号
LPD_CPM	4	RO	0x0	LPD-CPM 接口和信号
FPD_SOC	2	RO	0x0	FPD-SoC 接口和信号

表 17：**寄存器 PWR_STATUS 位字段详细信息** (续)

字段名称	位	类型	复位值	描述
FPD_PL	1	RO	0x0	FPD-PL 接口和信号

表 18：**11.1.22 寄存器 (PMC_GLOBAL) PWR_SUPPLY_STATUS**

寄存器名称	地址	宽度	类型	复位值	描述
PWR_SUPPLY_STATUS	0xF111010C	32	RO	X	此寄存器可提供电源状态。

表 19：**寄存器 PWR_SUPPLY_STATUS 位字段详细信息**

字段名称	位	类型	复位值	描述
保留	31:8	RO	0x0	保留
VCC_RAM	7	RO	0x0	RAM 主电源（也称为 VCCINT_RAM）
VCCINT	6	RO	0x0	PL 主电源（也称为 VCCINT_PL）
VCCAUX_SOC	5	RO	0x0	SoC 辅助电源（也称为 VCCAUX）
VCC_SOC	4	RO	0x0	NoC 和 DDR 存储器主电源 (SoC)（也称为 VCCINT_SOC）
VCC_LPD	3	RO	0x0	LPD 主电源（也称为 VCCINT_LPD）
VCC_FPD	2	RO	0x0	FPD 主电源（也称为 VCCINT_FPD）
VCC_PMC	1	RO	0x0	PMC 主电源（也称为 VCCINT_PMC）
VCCAUX_PMC	0	RO	0x0	PMC 辅助电源

功耗调试

功耗调试通常涉及系统级仿真工具和硬件测量。如需了解有关调试功耗问题的信息，请参阅以下资源：

- 功耗分析和功耗最优化或功耗降低

  如需了解有关超出功耗预算时降低功耗的提示和技巧，请访问此链接以参阅《Vivado Design Suite 用户指南：功耗分析与最优化》(UG907) 中的相应内容。

- 配电系统 (PDS) 和配电网络 (PDN)

  如需了解有关确保电源纹波不超出规格的信息，请访问此链接以参阅《Versal 自适应 SoC PCB 设计用户指南》(UG863) 中的相应内容，并请参阅《使用 S 参数模型对 FPGA 功耗完整性进行仿真》(WP411)。

- 上电问题和电源排序问题

  如果您遇到上电问题或电源排序问题，请参阅《简化电源排序应用指南》(XAPP1375) 以获取有关不同排序行为的信息以及有关如何实现正确的电源排序的信息。

- 功耗估算、散热仿真和硬件测量之间的关联问题
  - 硬件测量

    如需了解有关如何达成更准确的功耗估算的信息，请访问此链接以参阅《Vivado Design Suite 用户指南：功耗分析与最优化》(UG907) 中的相应内容。如需了解测量调试指南和有关如何获取更准确的功耗和结温的信息，请访问此链接以参阅《Vivado Design Suite 用户指南：功耗分析与最优化》(UG907) 中的相应内容。

  - 散热设计解决方案和调试问题

如需了解有关散热规格和管理的信息，请参阅《Versal 自适应 SoC 封装和管脚分配架构手册》(AM013)。如需获取散热器解决方案最佳实践，请参阅《为 AMD 器件设计散热器和热处理解决方案》(XAPP1377)。

- 电源管理问题

  如需了解相关信息，请参阅电源管理 - 入门指南维基百科和电源效率网页。

调试 PS/PMC

PS/PMC 外设复位

确认已配置的外设已解复位。

- 在向 PS 或 PMC 中的任意外设发起请求前，请确保外设已解复位。
- 检查 CRx（CRL、CRF 和 CRP）模块中的所有复位寄存器。

要使用 XSCT 读取 QSPI 复位寄存器，请运行以下命令：

```
xsct%mrd -force 0xF1260300
```

当 QSPI 解复位时，上述命令运行后的期望值为 `0x0`。

表 20：**寄存器 RST_QSPI**

寄存器名称	地址	宽度	类型	复位值	描述
RST_QSPI	0xF1260300	1	RW	0x00000001	个别块的复位

表 21：**寄存器 RST_QSPI 位字段详细信息**

字段名称	位	类型	复位值	描述
RESET	0	RW	0x1	当断言有效且值为 1 时，块将被复位

PS/PMC 时钟

确认 PS/PMC PLL 处于稳定且锁定状态、时钟处于有效状态并且已正确配置其除法器、乘法器和电源。

- 检查每个模块的时钟速度等级是否已配置正确。
- 读取 CRx（CRL、CRF 和 CRP）模块寄存器以检查时钟设置。

要使用 XSCT 读取 QSPI 时钟寄存器，请运行以下命令：

```
xsct%mrd -force 0xF1260118
```

表 22：**寄存器 QSPI_REF_CTRL**

寄存器名称	地址	宽度	类型	复位值	描述
QSPI_REF_CTRL	0xF1260118	32	RW	0x01000400	此寄存器用于控制此参考时钟

表 23：**寄存器 QSPI_REF_CTRL 位字段详细信息**

字段名称	位	类型	复位值	描述
保留	31:25	RW	0x0	保留
ClkAct	24	RW	0x1	时钟有效信号。 0 = 禁用时钟。 1 = 启用时钟。
保留	23:18	RW	0x0	保留
Divisor	17:8	RW	0x4	10 位除法器值： 0 = 除以 1 1 = 除以 1 2 = 除以 2 以此类推 1023 = 除以 1023 **注释：**复位值 (0x4) 除以 4。
保留	7:3	RW	0x0	保留
SrcSel	2:0	RW	0x0	000：PPLL 011：NPLL 其他：保留

调试 NoC

在 Versal 器件中，片上网络 (NoC) 用于在可编程逻辑 (PL)、CIPS 和其他集成块中的 IP 端点之间共享数据。NoC 组件由 NoC 主单元 (NMU)、NoC 从单元 (NSU)、NoC 包开关 (NPS) 和 NoC 裸片间桥接 (NIDB) 组成。为便于调试，每个 NoC 组件都设置了一组状态寄存器，这些状态寄存器连接在一起形成中断状态寄存器。来自这些寄存器的中断会被布线到 PMC，供其他进程元素进行监控、错误处理或订阅。每个组件的寄存器通常包含：

- 性能计数器：判定组件观测到的带宽和时延。
- 协议检查器：检查是否存在 AXI 规格或组件限制违例，这在集成定制 PL IP 时非常有用。
- 奇偶校验和纠错码 (ECC) 错误检查器：检查数据传输是否有错误。
- 超时计数器：检查无响应组件，包括组件上游和下游各一个寄存器。超时计数器还可以执行一些基本时延检查性能。

使用这些寄存器进行调试时，通常先接收中断，然后查询每个 NoC 组件，以判定错误源。请参阅 NoC 调试博客，获取有关使用这些寄存器进行调试的教程。《Versal 自适应 SoC NoC 和集成存储器控制器 NPI 寄存器参考资料》(AM019) 中列出了用于报告错误状态和错误管理的 NoC 寄存器。

通过 NoC 的每个连接都具有关联的 QoS 要求。如需了解有关受支持的 QoS 设置及其对于特定流量类型的影响的详细信息，请访问此链接以参阅《Versal Adaptive SoC Programmable Network on Chip and Integrated Memory Controller LogiCORE IP 产品指南》(PG313) 中的相应内容。如需获取有关 NoC/DDRMC 性能调优的指南，请参阅 GitHub 仓库中提供的 Versal 片上网络/DDR 存储器控制器性能调优教程。

调试 DDR 存储器控制器

集成的 DDR 存储器控制器支持 DDR4 和 LPDDR4/4X 存储器接口。DDR 存储器控制器有 4 个可编程 NoC 接口端口，按设计用于处理流量的多重串流。有 5 个服务质量 (QoS) 类别可用于确保对命令进行适当的优先排序。控制器接受突发传输事务并实现命令重新排序，以最大限度提高存储器接口的效率。可选的外部接口可靠性功能包括 ECC 错误检测/纠正和命令地址奇偶校验。如需了解有关 DDR 存储器控制器特性、功能以及与 NoC 交互的更多信息，请参阅《Versal Adaptive SoC Programmable Network on Chip and Integrated Memory Controller LogiCORE IP 产品指南》(PG313)。

调试 DDR 存储器控制器性能

通过 NoC 的每个连接都具有关联的 QoS 要求。如需了解有关受支持的 QoS 设置及其对于特定流量类型的影响的信息，请访问此链接以参阅《Versal Adaptive SoC Programmable Network on Chip and Integrated Memory Controller LogiCORE IP 产品指南》(PG313) 中的相应内容。如需获取有关 NoC 和 DDR 存储器控制器性能调优的指南，请参阅 GitHub 仓库中提供的 Versal 片上网络/DDR 存储器控制器性能调优教程。

Versal 片上网络/DDR 存储器控制器性能调优教程使用的正是《Performance AXI Traffic Generator 产品指南》(PG381) 中记述的 Performance AXI Traffic Generator。这些流量生成器是在仿真和硬件中进行任务模式的流量模式建模的基本构建块。为确保成功完成基于 Versal 器件的设计，必须理解系统级别流量模式和需求，并在设计输入期间通过 Vivado IP integrator 将其转换为 QoS 需求。达成所期望的 QoS 结果后，下一步就是在仿真中利用 Performance AXI Traffic Generator 对此进行建模。Versal 片上网络性能 AXI Traffic Generator 教程演示了性能流量生成器用例。其中一个用例对应不可综合的流量生成器，仅用于仿真。另一个则是可综合的流量生成器，可在仿真和硬件中使用。

调试性能问题时，您必须考虑设计中的流量源以及流量源访问存储器的方式。这包括 AXI 命令的大小、长度和对齐，以及这些命令与 DDR 存储器控制器的交互方式。您还必须了解存储器接口技术、存储器接口拓扑结构、DDR 存储器控制器地址映射以及是否启用通道交织。许多性能问题都归因于流量源的访问模式欠佳，包括流量源访问 DDR 存储器控制器地址映射的方式。Versal 器件 DDR 存储器控制器具有复杂的重新排序功能，但设计还必须考虑底层 DDR 存储器控制器协议的限制。仿真的“Command Decode”（命令解码）模块是一款实用的工具，能用于查看和分析给定访问模式下 DDR 存储器控制器协议执行情况。AMD Versal 自适应 SoC：仿真中的性能分析教程演示了如何将此工具用于 DDR4 场景。

调试 DDR 存储器控制器接口

如需了解有关辅助调试 DDR 存储器控制器接口的信息，请参阅《Versal Adaptive SoC Programmable Network on Chip and Integrated Memory Controller LogiCORE IP 产品指南》(PG313)。该产品指南包含有关遇到错误后该如何操作的信息，欲知详情，请访问此链接。该产品指南还包含有关要采取的各步骤的信息以及发生校准错误时显示的错误消息，欲知详情，请访问此链接。调试硬件相关问题时，请访问此链接以参阅《Versal 自适应 SoC PCB 设计用户指南》(UG863) 中的相应内容，了解如何确保存储器接口满足布局准则的要求。在处理 DDR 存储器控制器校准错误时，请使用尽可能小的硬件设计，通常是 CIPS 实例和您正在调试的单个 NoC DDR 存储器控制器。要消除校准错误，可尝试降低存储器接口频率或启用“DDR Advanced”（DDR 高级）选项卡下的“2T timing for DDR4”（DDR4 的 2T 时序）。

要获取 NoC 和 DDR 存储器控制器的最新已知问题列表，请参阅答复记录 75764。如需了解有关双通道 DDR 存储器控制器拓扑的信息，请参阅答复记录 76830。如果操作系统尝试按此答复记录中所述使用存储空间，那么该操作系统在启动期间或操作期间将遇到意外错误。此答复记录还包含指向重要资源的链接，如 Versal 自适应 SoC DDRMC - DDR4 和 LPDDR4/x PCB 仿真支持文章和 Versal 自适应 SoC DDRMC - DDR4、LPDDR4 和 LPDDR4X 外部参考时钟设计指南文章。这些文章描述了如何为 PCB 级仿真生成 IBIS 模型、解释了 IP 为 DDR 存储器控制器配置生成的默认约束，并提供了有关如何为 DDR 存储器控制器设计外部参考时钟电路的指导信息。

由于 Versal 器件 DDR 存储器控制器对于生成有效的管脚分配有一套不同的限制，您必须遵循获取和验证 Versal 自适应 SoC 存储器管脚分配教程确保生成管脚分配。如果您在确认设计阶段遇到问题，请参阅答复记录 35164。如果您计划将来在使用硬件设计时增加存储器密度，那么为 Versal 器件生成管脚分配时，请访问此链接以参阅《Versal Adaptive SoC Programmable Network on Chip and Integrated Memory Controller LogiCORE IP 产品指南》(PG313) 中的相应内容。

如果 DDR 存储器控制器出现校准后数据错误，请使用 Versal 自适应 SoC DDR 存储器 - 二维眼图扫描教程。该工具能以半字节为单位，为存储器接口绘制二维数据有效窗口图。如果数据有效窗口较小或形状异常，则可能表示 PCB 布局或电源有问题。对于 PCB 布局或电源问题，可尝试以更低的数据速率来操作该接口，查看数据错误率是否会降低或完全消失。如果在原先的硬件设计中未启用 2T 时序选项，那么启用该选项也会有所帮助。您可以运行一次二维眼部扫描以获得基准线，然后在启用 2T 时序并降低数据速率后再次运行二维眼部扫描，以判定数据有效窗口大小是否有所改善。

调试 PCIe

要解决 CPM 和 PCIe 相关问题，您必须使用各种调试技巧和检查方法，找出根本原因并解决问题。常见问题包括链路训练、仿真、驱动程序和一般硬件问题。要获取一整套用于识别此问题的策略，请参阅 GitHub 仓库中提供的《PCIe 调试指南》。本资源提供了编译的检查表和概述文章，其中总结了 Versal 器件中 CPM/PCIe 问题的调试方法。它还包括相关资料，如用户指南、产品指南、版本说明、调试博客、答复记录、QuickTake 视频、白皮书、驱动程序文档和数据表。

调试 AI 引擎

AMD 提供了多种工具和流程用于对 Versal AI 引擎处理器上运行的设计进行调试。这些工具和流程可用于进行功能调试、调试 AI 引擎算法内的性能问题以及对 AI 引擎应用开发的各阶段进行调试。以下总结了可用的各种工具和流程及其适用时机的建议。

AI 引擎内核的功能调试通常需在 Vitis 内运行 x86 仿真器。此类仿真器可广泛用于功能调试，但由于它不记录程序存储器大小，因此请谨慎使用。如需了解有关 x86 仿真器的可用选项和运行方法的更多信息，请访问此链接以参阅《AI 引擎工具和流程用户指南》(UG1076) 中的相应内容。

AI 引擎内核的功能调试还可使用 Vitis 中的 aiesimulator 来执行。该仿真器属于周期近似仿真器，可广泛用于功能调试。它会使用 SystemC 语言对 Versal 器件（包括 NoC、PS 和 PL 组件）进行建模。您可使用内建调试器来单步执行 AI 引擎源代码，也可以将其用于进一步调试设计功能。printf 功能同样可供使用，但它可能影响程序存储器大小（由 aiesimulator 进行追踪）。Vitis IDE 具有调试视图，其中可显示寄存器、变量、可用断点、寄存器/存储器映射变量、内部/外部存储器内容、用于查看指令的分解视图，并且对应每个 AI 引擎内核各有 1 条指令流水线（流水线视图）。

由于 aiesimulator 同样属于周期近似仿真器，因此可以对设计执行广泛的性能调试。该仿真器可提供剖析和事件追踪功能，该功能可用于分析设计中的停滞和性能问题的根本原因。

此外，aiesimulator 还具有可用于检测并报告内核代码中的出界访问违例的选项。如需了解有关 aiesimulator 可用选项及其运行方法的更多信息，请访问此链接以参阅《AI 引擎工具和流程用户指南》(UG1076) 中的相应内容。

使用 Versal 平台通过 V++ 将 AI 引擎内核集成到 Versal 器件其余部分时，可以运行硬件仿真器。该仿真器属于周期精确的仿真器，可使用 RTL 或 SystemC 仿真模型来对 PL 逻辑进行仿真。此仿真步骤应在 v++ 链接步骤后执行，且“hw_emu”目标指定为 v++ 链接命令行。调试器还可用于调试 AI 引擎内核，并单步执行内核代码和 PS 主机代码。此步骤使用 QEMU 和 NoC 的 SystemC 仿真模型来对 PS 进行仿真。如需了解有关硬件仿真器的可用选项及其运行方法的更多信息，请参阅：

- 请访问此链接并参阅《AI 引擎工具和流程用户指南》(UG1076) 中的相应内容
- 请访问此链接并参阅《AI 引擎工具和流程用户指南》(UG1076) 中的相应内容

准备好将设计添加到硬件上后，有多种调试和剖析工具可供您使用。在硬件中运行设计时，您可使用 PS 主机代码中的运行时事件 API 或者使用硬件内建的性能计数器的编译时选项来获取剖析数据。分析此数据有助于您测量内核效率、与每个 AI 引擎关联的停滞和活动时间，并确定性能可能未处于最优状态的 AI 引擎内核。您还可收集有关设计时延、吞吐量和带宽的数据。在 Vitis 分析器工具中可提供剖析的详细热图和直方图。如需了解有关硬件中的设计性能分析可用的选项及其分析方法的更多信息，请访问此链接以参阅《AI 引擎工具和流程用户指南》(UG1076) 中的相应内容。

您也可以对硬件中的 AI 引擎内核源代码进行调试。这样有助于调试硬件中的功能问题。如需了解有关在 Vitis IDE 内运行调试器以及与调试器关联的各项功能的更多信息，请访问此链接以参阅《AI 引擎工具和流程用户指南》(UG1076) 中的相应内容。

使用 SmartLynq+ 执行调试

AMD SmartLynq+ 模块属于高速调试和追踪模块，主要支持 Versal 器件。您可在 Versal 器件设计中使用 SmartLynq+ 模块来充分利用高速调试端口 (HSDP)，从而缩短配置时间并实现高速调试连接。SmartLynq+ 模块在许多情况下还可直接替代远程实验室 PC，因为它支持通过千兆以太网连接来直接连接至局域网。

为了将 HSDP 与 SmartLynq+ 搭配使用，设计通常包含 HSDP 接口（在 Control, Interface, and Processing (CIPS) IP 中启用），并且在开发板级别通过基于 USB 的 JTAG 连接至 USB-C 接口以便与 SmartLynq+ 相连。

欲知详情，请访问此链接以参阅《SmartLynq+ 模块用户指南》(UG1514) 中的相应内容。

调试 PL

如果遇到在 PL 逻辑仿真中难以复现的情况，可能需要调试可编程逻辑 (PL)。本节涵盖了有关允许查看 PL 域的调试工具的信息。

AXI4 Debug Hub 连接

要使用 AMD Vivado™ 调试核，设计必须包含 AXI4 Debug Hub。AXI4 Debug Hub 可用于将 CIPS 的 AXI-MM 接口与 AXI4-Stream 接口相连。此接口可连接到 Vivado 调试核，其中包含以下类型的核：

- AXI4-Stream Integrated Logic Analyzer (AXIS-ILA)
- AXI4-Stream Virtual Input/Output (AXIS-VIO)
- PCI Express® Link Debugger

如果设计包含任何 Vivado 调试核且 CIPS 已启用 1 个或多个 PL 复位，那么 AXI4 Debug Hub 将由 Vivado 在执行 `opt_design` 期间自动插入，并使用 NoC 连接到 CIPS。AXI4 Debug Hub 可手动例化并连接到 CIPS，但这样将不会执行自动插入。

下表显示了在 `opt_design` 期间发生 AXI4 Debug Hub 自动插入的各种不同场景。

表 24：**AXI4 Debug Hub 自动插入**

已综合的网表内容	AXI4 Debug Hub 插入操作
CIPS 已启用一个或多个 PL 复位。无 AXI4 Debug Hub。	在 `opt_design` 期间会插入 AXI4 Debug Hub，并使用 NoC 实例将其连接到 CIPS。
CIPS 含一个 AXI4 Debug Hub（先前已存在）	设计中未手动连接到 AXI4 Debug Hub 的所有调试核都会在执行 `opt_design` 期间连接到先前已存在的 Debug Hub。
CIPS 含多个 AXI4 Debug Hub	仅在发生以下任一操作的情况下才会发生自动整合： · 调试核上的 AXI4-Stream 接口已手动连接到目标 AXI4 Debug Hub。 · 每个调试核与关联的 AXI4 Debug Hub 之间的连接都是使用 `connect_debug_core` 约束指定的。

使用 ILA 核

Integrated Logic Analyzer (ILA) 核支持您在器件上对实现后设计执行系统内调试。如果需要监测设计中的信号，请使用此核。另外，您还可以使用此功能触发硬件事件并以系统级速度捕获数据。

设计探测

Vivado 工具提供了多种方法用于在设计中添加调试探针。下表逐一解释了这些方法，并介绍每种方法各自的优劣。

表 25：**调试流程**

调试流程名称	流程步骤	优势和劣势
HDL 例化探测流程	在 HDL 源或 IP integrator 中将信号显式连接到 ILA 调试核实例。	· 您必须在设计中手动添加/移除调试信号线和 IP，即必须修改 HDL 源文件。 · 此方法支持选择在 HDL 设计层级进行探测。 · 支持在接口层级探测某些协议，例如，AXI 或 AXI4-Stream 协议 · 生成、例化和连接调试核时容易出错。
网表插入探测流程	使用以下两种方法之一来识别要调试的信号： · 使用 MARK_DEBUG 属性来标记源 RTL 代码中要调试的信号。 · 使用 MARK_DEBUG 右键单击菜单选项从已综合的设计网表中选择要调试的信号线。 标记要调试的信号后，使用“Set up Debug”Wizard（设置调试向导）来指导您完成“Netlist Insertion”（网表插入）探测流程。	· 此方法灵活性最高，有良好的预测能力。 · 此方法便于在各个不同设计层级（如 HDL、综合设计、系统设计）进行探测。 · 此方法无需修改 HDL 源文件。
基于 Tcl 的网表插入探测流程	使用 `set_property` Tcl 命令在调试信号线上设置 MARK_DEBUG 属性，然后使用网表插入探测 Tcl 命令来创建调试核并将其连接到调试信号线。	· 此方法可提供全自动网表插入功能。 · 您可通过调整 Tcl 命令来开启或关闭调试功能。 · 此方法无需修改 HDL 源文件。

选择调试信号线

AMD 对于选择调试信号线的建议如下：

- 探测特定层级边界（输入或输出）处的信号线。此方法有助于快速确定问题区域。随后您便可根据需要进一步深入层级进行探测。
- 请勿探测组合逻辑路径间的信号线。如果在位于组合逻辑路径中间的信号线上添加 MARK_DEBUG，那么适用于流程实现阶段的任何最优化都无法应用于此处，从而导致时序 QoR 结果不达标。
- 探测处于同步状态的信号线以获取周期精确的数据捕获。

使用 MARK_DEBUG 保留调试探测信号线名称

您可在 RTL 阶段或综合后添加调试信号标记。信号线上的 MARK_DEBUG 属性可避免对信号线进行复制、重定时、移除或以其他方式进行最优化。您可在顶层端口、信号线、层级模块端口和层级模块内部信号线上应用 MARK_DEBUG 属性。这种方法很有可能能够在综合后保留 HDL 信号名称。带有调试标记的信号线在综合后会显示在“Debug”（调试）窗口中的“Unassigned Debug Nets”（未分配的调试信号线）文件夹下。

将 `mark_debug` 属性添加到 HDL 文件中，如下所示：

VHDL：

```
attribute mark_debug : string;
attribute mark_debug of sine   : signal is "true";
```

Verilog：

```
(* mark_debug = "true" *) wire sine;
```

您还可在综合后网表中添加要调试的信号线。这些方法无需修改 HDL 源文件。但可能因网表最优化涉及吸收或合并设计结构而导致综合后未能保留原始 RTL 信号。完成综合后，可采用以下任一方式添加要调试的信号线：

- 选中“Netlist”（网表）或“Schematic”（板级原理图）窗口等任一设计视图中的信号线，然后选择“Mark Debug”（标记调试）。
- 选中任一设计视图中的信号线，然后将其拖放到“Unassigned Debug Nets”文件夹中。
- 在“Set Up Debug”Wizard（设置调试向导）中使用信号线选择器。
- 使用“Properties”（属性）窗口或 Tcl 控制台来设置 MARK_DEBUG 属性。

  ```
  set_property mark_debug true [get_nets -hier [list {sine[*]}]]
  ```

  这样会对当前打开的网表应用 `mark_debug` 属性。这是一种灵活方法，因为可通过 Tcl 命令开启或关闭 MARK_DEBUG。

ILA 核与时序注意事项

ILA 核的配置会对能否满足整体设计时序目标产生影响。请根据下列建议进行操作，以便最大程度减少对时序的影响：

- 请审慎选择探针宽度。随探针宽度增加，对资源使用率和时序的影响也会增大。
- 请审慎选择 ILA 核数据深度。随数据深度增加，对块 RAM 资源使用率和时序的影响也会增大。
- 请确保为 AXIS-ILA `clk` 端口所选的时钟为自由运行的时钟。否则可能造成在器件上加载设计时无法与调试核通信。
- 请确保连接到 AXI4 Debug Hub 的时钟为自由运行的时钟，并同步到已连接到 S_AXI 端口的 AXI 主接口。否则可能造成在器件上加载设计时无法与调试核通信。
- 请确保输入到 ILA 核的时钟与正在探测的信号同步。否则在设计编程到器件中时会产生时序问题并导致通信失败。
- 在硬件上运行设计之前请确保设计已满足时序要求。否则会导致探测到的波形不可靠。

下表列出了在设计时序和资源时使用特定 ILA 特性的影响。

注释：该表基于含单个 ILA 的设计，不代表所有设计。

表 26：**ILA 特性对设计时序和资源的影响**

ILA 特性	用途	时序	区域
捕获控制/存储条件	捕获相关信息 有效利用数据捕获存储（块 RAM）	影响程度：中到高	· 不增加块 RAM · 少量增加 LUT/FF 数量
高级触发器	当“BASIC”（基础）的触发条件不足以满足需要时 使用复杂的触发来聚焦问题区域	影响程度：高	· 不增加块 RAM · 适度增加 LUT/FF 数量

表 26：**ILA 特性对设计时序和资源的影响** (续)

ILA 特性	用途	时序	区域
每个探针端口的比较器数量 **注释：**最大数量为 4。	在多种条件下使用探针探测： · 1-2 个，对应基础条件 · 1-4 个，对应高级条件 · +1 个，对应捕获控制	影响程度：中到高	· 不增加块 RAM · LUT/FF 数量呈少量增加到适度增加范围内
数据深度	捕获更多数据样本	影响程度：高	· 每个 ILA 核额外增加块 RAM · 少量增加 LUT/FF 数量
ILA 探针端口宽度	按标量来调试大量总线	影响程度：中等	· 每个 ILA 核额外增加块 RAM · 少量增加 LUT/FF 数量
探针端口数量	探测大量信号线	影响程度：低	· 每个 ILA 核额外增加块 RAM · 少量增加 LUT/FF 数量

提示：在设计早期阶段，通常在器件中有大量备用资源可用于调试。

含高速时钟的 ILA 核设计

对于高速时钟设计，请注意如下事项：

- 限制调试的信号数量和宽度。
- 通过设置输入流水线阶段数量，将输入探针通过流水线输送到 AXIS-ILA。此设置可在 AXIS-ILA GUI 的“Advanced”（高级）选项卡下找到，或者也可以在使用 Tcl 插入时通过 `C_INPUT_PIPE_STAGES` 属性来设置。

使用 VIO 核

Virtual Input/Output (AXIS-VIO) 核支持您实时监控和驱动内部器件信号。如果需要启动或监控低速信号（如复位信号或状态信号），请使用此核。AXIS-VIO 调试核必须在设计中例化，并且可在 IP integrator 和 RTL 中使用。在 IP 目录中包含 AXIS-VIO 核，可在基于 RTL 的设计和 IP integrator 中使用。

如需了解有关自定义 AXIS-VIO 核的信息，请参阅《Virtual Input/Output (VIO) with AXI4-Stream Interface LogiCORE IP 产品指南》(PG364)。如需了解有关利用 AXIS-VIO 核进行测量的信息，请访问此链接以参阅《Vivado Design Suite 用户指南：编程和调试》(UG908) 中的相应内容。

VIO 核注意事项

使用 AXIS-VIO 核时，请考量以下注意事项：

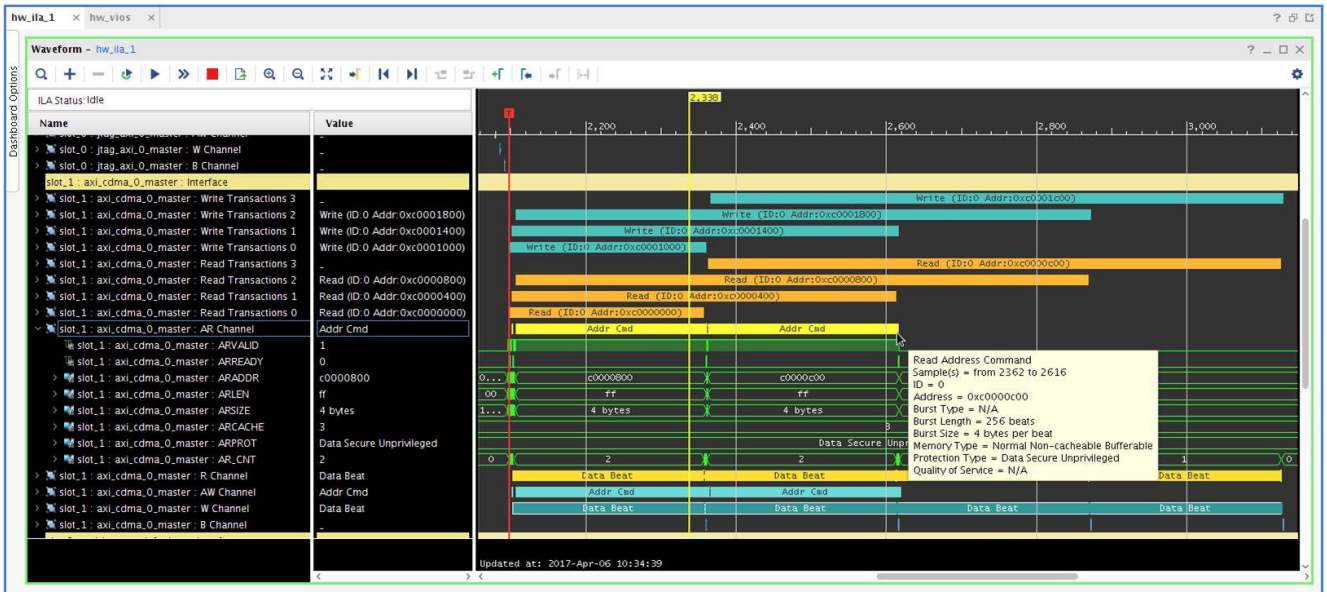
- 连接到 AXIS-VIO 输入探针的信号必须与连接到 AXIS-VIO 核的 AXIS-VIO `clk` 端口的时钟保持同步。连接的信号与 `clk` 端口不同步会导致 AXIS-VIO 输入探针端口上出现时钟域交汇。
- 从 AXIS-VIO 输出探针驱动的信号与连接到 AXIS-VIO 核上的 AXIS-VIO `clk` 端口的时钟将保持同步断言有效和同步断言无效。
- AXIS-VIO 核的刷新率相对较低，因为它旨在替代低速开发板 I/O，例如，按钮或指示灯 (LED)。要捕获高速信号，请考虑使用 ILA 核。

在 Vivado 硬件管理器中调试 AXI 接口

AXIS-ILA 支持您在 AMD 器件上对实现后设计执行系统内调试。如果需要监控设计中的接口和信号，可使用此功能。

如果将 AXIS-ILA 模式更改为“Interface”，即可在“Waveform”窗口中调试和监控 AXI 传输事务和读写事件，如下图所示。“Waveform”窗口可显示对应于 AXIS-ILA 上的接口插槽所探测到的接口的接口插槽、传输事务、事件和信号组。

图 51：**“Waveform”窗口**



如需了解有关 AXIS-ILA 和在 Vivado 硬件管理器中调试 AXI 接口的更多信息，请访问此链接和此链接，以参阅《Vivado Design Suite 用户指南：编程和调试》(UG908) 中的相应内容。

使用 IBERT GTY、GTYP 和 GTM 查看收发器链路特性

Versal 器件 GTY、GTYP 和 GTM 收发器支持 Integrated Bit Error Ratio Tester (IBERT)，它可启用系统内串行 I/O 确认和调试。这样即可支持对高速串行 I/O 链路进行测量和最优化。

在向接收到的信号应用接收器均衡后，如需测量信号质量，即可使用 IBERT。

不同于先前架构，Versal 器件 IBERT 功能已集成到 GTY、GTYP 和 GTM 收发器中，只要求设计使用这些收发器即可。如需了解有关 IBERT 功能的更多信息，请访问此链接以参阅《Versal Adaptive SoC Transceivers Wizard LogiCORE IP 产品指南》(PG331) 中的相应内容。

运行调试相关 DRC

Vivado Design Suite 提供调试相关 DRC，运行 `report_drc` 时将选择这些 DRC 作为默认规则卡的一部分。DRC 会检查是否存在如下问题：

- 由于当前调试核要求，导致超出当前器件可用的块 RAM 资源量。
- 非时钟信号线连接到调试核上的时钟端口。
- 调试核上的端口未连接。
- CIPS 不启用用于 Debug Hub 插入流程的 `pl_resetn0`。

生成 AXI 传输事务

AXI 传输事务可使用调试包控制器 (DPC) 在本地生成，无需其他 IP。

1. 将可编程器件镜像 (PDI) 加载到器件中。
2. 启动赛灵思系统调试器 `xsdb`。
3. 使用 `connect` 连接到硬件服务器。

   注释：远程连接可使用 `-host` 开关来指定远程主机。
4. 输入 `targets` 即可列出可用目标。
5. 将当前目标设置为 DPC：`target <number next to DPC>`。

修改已实现的网表以替换现有调试探针

在已布局布线的设计检查点中可以替换已连接到 ILA 核的调试信号线。您可使用“Engineering Change Order (ECO)”（工程变更单 (ECO)）流程来执行此操作。这是 1 个高级设计流程，用于接近完成的设计，在此流程中您需要交换已连接到现有 ILA 探针端口的信号线。如需了解有关使用 ECO 流程来修改现有 ILA 核上的信号线的相关信息，请访问此链接以参阅《Vivado Design Suite 用户指南：实现》(UG904) 中的相应内容。

在已实现的网表上插入、删除或编辑 ILA 核

如果要添加、删除或修改 ILA 核（例如，调整探针宽度大小、更改数据深度等），AMD 建议您使用“Incremental Compile”（增量编译）流程。适用于调试核的“Incremental Compile”流程在已综合的设计或检查点 (DCP) 上运行并使用已实现的检查点，最好是来自先前实现运行的检查点。相比于完全重新实现设计，此方法可节省时间。

如需了解有关使用“Incremental Compile”流程来插入、删除或编辑 ILA 核的信息，请访问此链接以参阅《Vivado Design Suite 用户指南：编程和调试》(UG908) 中的相应内容。

使用布线后 ECO 将信号线连接到空闲的外部管脚

在某些情况下，您可能需要将信号线引出至空闲的器件管脚，以便使用外部测试设备来进行调试。如果您正在调试的问题需要对设计 QoR 进行少量更改，或者需要执行测量且别无他法，则可使用此方法。您可使用“Engineering Change Order (ECO)”（工程变更单 (ECO)）流程来执行此操作，前提是器件具有未使用的 I/O，可用于此目的。如需了解有关使用 ECO 流程来修改已布线设计的更多信息，请访问此链接以参阅《Vivado Design Suite 用户指南：实现》(UG904) 中的相关内容。

使用远程调试

要对设计进行远程调试或升级，请使用 AMD 硬件服务器产品来连接到实验室内的远程计算机。

如需了解有关远程连接的更多信息，请访问此链接以参阅《Vivado Design Suite 用户指南：编程和调试》(UG908) 中的相应内容。

您也可以使用 SmartLynq+ 模块，该模块在诸多情况下均可直接替代远程实验室 PC。

软件调试

软件调试方法论主要取决于以下要素：

- 应用的目标处理器

- 所使用的操作系统 (OS) 的类型
- 应用软件/内核调试
- 功能/性能调试

目标处理器

应用的目标处理器可以是 Arm® Cortex®-A72 处理器、Cortex-R5F 处理器、PL 中的 MicroBlaze™ 处理器或 AI 引擎核处理器。根据处理器类型，调试器可以连接到任一处理器目标并执行单步步进或断点插入以对代码的特定部分进行调试。如果目标处理器为 Cortex-A72 或 Cortex-R5F，则可使用 HSDP 或 JTAG 连接到 Arm CoreSight™ 接口。Vitis 调试器或第三方调试器（例如，Lauterbach）可连接到 CoreSight 接口并执行单步步进、断点插入、指令反汇编、核寄存器状态等。

对于 AI 引擎处理器，Vitis 调试器可以连接到 AI 引擎调试接口并执行单步步进、断点插入并查看程序/数据存储器。Vitis 调试器可以使用 XSCT 接口连接到目标处理器。XSCT 接口可以提供一整套调试命令，用于读取处理器的内部寄存器和存储器。例如，如果软件代码在某条特定指令处挂起，您可以读取 PC 地址并检查它执行的最后一条指令。如需了解有关 XSCT 命令的更多信息，请参阅《Vitis 嵌入式软件开发流程文档》(UG1400) 中的软件命令行工具。

操作系统

操作系统的类型可确定软件调试步骤。如果应用基于裸机操作系统，则更便于从应用读取特定外设地址。例如，如果应用支持特定内核，但内核数据流程并未发生，则可从应用读取内核的控制寄存器和长度寄存器以检查编程值是否正确。

对于裸机应用，可以基于物理地址转储内核的输入/输出缓冲器，这样有助于从软件角度来执行调试。如果输入缓冲器本身与期望的输入不匹配，那么可以通过从应用对同一 DDR 缓冲器执行写入/回读来进一步排列优先级。如果回读不匹配，那么可集中对 DRAM 接口进行进一步调试。

对于基于 Linux 的应用，要将缓冲器内容转储到文件，则已分配的缓冲器必须是存储器映射的缓冲器并且可返回物理地址。随后，应用即可将输入和输出缓冲器的内容与参考输入进行比对。应用可以使用器件存储器映射 (`/dev/mem`) 来访问存储器映射的内核寄存器空间，以检查编程的内核控制、大小和地址值是否正确。如果这些值不正确，则可对应用到器件驱动接口进行进一步调试。

应用软件/内核调试

基于 Linux 的软件堆栈可能需要在应用层和内核层进行调试。典型 Linux 应用使用 IOCTL 调用来与内核空间驱动程序进行交互。如果从内核驱动程序未返回 I/O 读取/写入响应，那么您可以检查特定硬件寄存器，查看编程的地址、长度、中断相关的掩码/使能信号是否正确。如果寄存器编程正确，那么下一个调试步骤是通过使用 `cat` 命令来观察 `/proc/interrupts`，确定针对硬件器件是否已寄存任何中断。

如果未寄存中断，则需要从硬件侧执行进一步调试。如果已寄存中断，那么您可以在驱动程序中添加具体的内核打印声明，以对中断服务例程和其他函数调用进行调试。应用级别调试可以使用 GDB 调试器来执行。另外，请确保应用代码中没有任何存储器泄漏，通常当动态分配的存储器用完后，如果未能清空，就会发生存储器泄漏。此类问题可使用 Valgrind 工具来调试，该工具有助于检测此类场景。

功能/性能调试

根据调试与功能有关还是与性能有关，可以采用不同的调试策略。对于功能调试，可以根据应用类型、使用的操作系统和应用/内核级别调试来采用以上建议的几种方法。对于性能调试，如果应用位于控制路径中，那么可以使用内置定时器（TTC 或 Cortex-A72 处理器中的全局计数器）执行函数剖析，检查执行加速器回调的函数是否在某个时间点导致性能变慢，并且可从软件角度来执行进一步调试。

如果软件应用位于数据路径内，那么您可执行与函数剖析相似的剖析，以判断缓冲器分配是否导致任何开销，从而导致加速器函数调用发生延迟。通常，如果加速器需要大型相邻存储器，则使用相邻存储器分配器来分配该加速器。如果其他加速器使用此相邻存储器，那么存储器分配可能需要更长时间。在此类情况下，可能可以采用静态分配。要对硬件/软件性能问题优先顺序进行排序，还可以采用其他调试策略，包括计算特定时序窗口内的中断数量。如果中断数量不比期望的数量小，那么可以从硬件角度执行进一步调试。

第 6 章

确认

AMD Versal™ 器件的多种不同计算域给传统 FPGA 确认方法带来了诸多挑战。除了可编程逻辑和处理器子系统外，Versal 器件还包含 AI 引擎，使系统确认任务比传统 FPGA 更复杂。

此确认方法是围绕以下关键概念构建的：

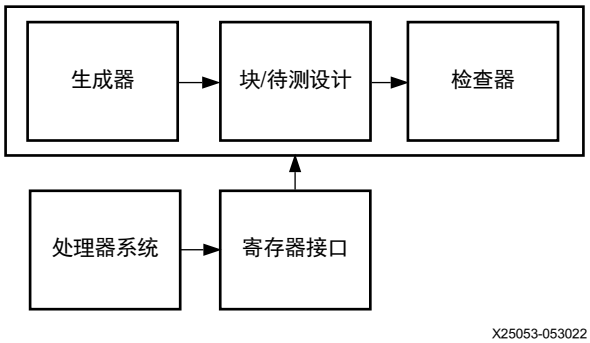
- 块/IP 确认：PL 内的各 RTL 和 HLS IP 可先单独确认，然后再执行系统集成。
- AI 引擎确认：位于接口级别的 AI 引擎可视作为 AXI4 存储器映射或 AXI4-Stream IP。
- 系统确认：完成各块确认后，即可确认整个系统、使用处理器来协调数据流、测试矢量生成、监控等。

块确认和 IP 确认

系统的 PL 可包含不同类型的 IP。如果 IP 与 AMD Vitis™ 环境兼容（即，到 IP 的接口仅含 AXI4 存储器映射接口、AXI4-Stream、时钟和复位），那么您可使用来自安装区域的现有 Vitis 平台来将这些块与生成器和检查器逻辑集成。如果 IP 具有与 Vitis 环境不兼容的独特端口，那么您可围绕此 IP 开发封装文件以使其兼容，或者使用传统 AMD Vivado™ IP integrator 方法来将设计与生成器和检查器集成。

下图显示了通用块确认 (validation) 设计。

图 52：**通用块确认设计**



生成器和检查器

您可以选择使用以下任意生成器和检查器：

- 现成 AXI-DMA 或 MCDMA IP（来自 Vivado IP integrator 目录），可用于以串流方式将数据从 DDR 存储器传输至可编程逻辑。此方法是将块传输到硬件的最简单方法。如需了解更多信息，请参阅《Vivado Design Suite 用户指南：采用 IP integrator 设计 IP 子系统》(UG994)。
- LFSR 生成器和检查器（来自 Vivado 工具语言模板）。或者，您也可以自行设计虚拟随机数据生成器检查器。

- UltraRAM 到块 RAM，可提供特定数据用于验证块的功能。

  注释：您可在 RAM 中嵌入初始存储器内容（MEM 文件）以及其他电路（例如，AXI GPIO），这样即可灵活控制测试工具。

- AXI Traffic Generator (ATG) 可生成预配置的流量类型，也可通过编程来检查所需长度的数据。虽然此方法并非总能提供数据完整性检查，但它可用于快速确认受测设计的用途。
- AXI 或 AXI4-Stream ILA，可对受测设计的输出数据进行监控。借助此方法即可使用 Vivado IDE 来直观分析数据，并以有限的开销将波形可视化。如需了解更多信息，请参阅《Vivado Design Suite 用户指南：编程和调试》(UG908)。

构建测试工具后，即可在硬件上测试设计，而无需任何其他软件。所有寄存器接口均可通过赛灵思系统调试器 (System Debugger) 的 `xsdb` 控制台（JTAG 模式）来获取。

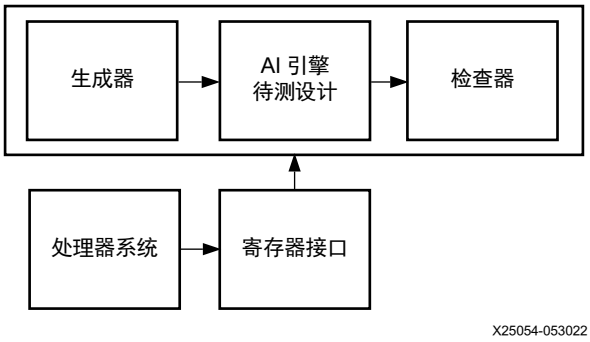
或者，您也可以开发软件应用来根据寄存器控制接口编排生成器和检查器功能。此方法对于包含多个 IP 的设计进程尤其有效，并且可帮助您为设计中不同阶段复用确认基础架构。

AI 引擎设计确认

您可使用与块确认相似的方法来确认 AI 引擎设计。主要差异之一在于 AI 引擎是基于 AXI 接口的 IP，使其更便于与 IP integrator 目录中或 Vitis HLS 中的其他 IP 相连。常用方法之一是使用基于 S2MM 和 MM2S 的 IP 来生成基于 DDR 存储器的流量并传输至 AI 引擎、从 AI 引擎取回数据，并将数据发送回 DDR 存储器。

下图显示了 AI 引擎设计的模块框图及其周围的测试工具。

图 53：**含测试工具的 AI 引擎设计**



不同于先前章节中所述的块 IP 确认方法，AI 引擎确认面临下列难题：

- 从可编程逻辑到 AI 引擎的 AXI 输入和输出串流数量。
- AI 引擎期望的数据带宽。

为便于说明，AI 引擎设计确认可广泛分类为功能确认和性能确认。

功能确认

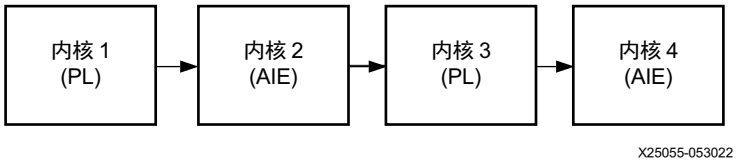
功能确认涉及检查 AI 引擎中 graph 的功能完整性。这可通过使用前述确认方法来完成，例如使用基于 ATG/AXI DMA/MCDMA/S2MM-MM2S HLS 的内核来将测试数据从 DDR 存储器直接传输至 AI 引擎。这是在 AI 引擎内测试 graph 功能的最简单的方法。

要对 AI 引擎进行编程，请启动 graph 并编排数据流，设计必须兼容 Vitis。如需了解更多信息，请参阅《Versal 自适应 SoC 设计指南》(UG1273)。您可使用随 Vitis 安装工具提供的 Vitis 平台，并在此基础上构建确认设计。此外，启用 AI 引擎无需软件主机应用，但需要基于裸机的主机应用才能利用 AI 引擎工具的所有功能，如《AI 引擎工具和流程用户指南》(UG1076) 中所述。

性能确认

性能确认是一项艰巨的任务，但也是确认是否满足 AI 引擎带宽的最重要的步骤。以如下复杂系统设计为例，其中包含多个 AI 引擎和可编程逻辑组件。

图 54：**复杂系统设计**



内核 1、2 和 3 之间的任何反压都将导致内核 4 性能不足，即使可将内核 4 设计为以最优性能运行也是如此。这又增加了一项传统 FPGA 设计中不常见的挑战。

为了使整个系统能够高效正确运行，请先使用本节及先前章节中所述方法对上述每个内核的功能和性能进行验证，然后再将其集成在一起，这样方可达成所需的系统设计。

AI 引擎性能确认要求生成器能够按所需吞吐量生成数据，并要求检查器能够按所需吞吐量接收数据。由于 AI 引擎与可编程逻辑之间的串流接口能够以 4 Gb/s 的速度运行，因此如果不满足性能确认指标，则可修改此接口以增加总线宽度并降低频率。例如，AI 引擎与 PL 间的 64 位/500 MHz 接口可修改为 128b/250 MHz 接口，而不会破坏整个系统的性能。只有在单独完成内核 2 和内核 4 的性能确认后，才能判定是否需要修改接口。欲知详情，请访问此链接以参阅《AI 引擎工具和流程用户指南》(UG1076) 中的相应内容。

AI 引擎工具包含各种 API 用于测量 AI 引擎-PL shim 接口中的每个接口的吞吐量，而无需任何其他硬件。这是了解 AI 引擎与 PL 间的性能数值的最简单方法。欲知详情，请访问此链接以参阅《AI 引擎工具和流程用户指南》(UG1076) 中的相应内容。

有 2 种方法可用于为 AI 引擎生成充足的带宽：

- 对测试工具生成器/检查器（例如，AXI4 DMA/S2MM-MM2S 内核）进行微调，以便按 AI 引擎所需提供充足的发送和接收带宽。
- 设计基于定制 RTL 的内核，以便按互连结构中所需的时钟频率生成基于 LFSR 或 BRAM/URAM 的矢量。

对于高性能设计，后一种方法可以提供一致的带宽，因为在 DDR 存储器与可编程逻辑之间不存在数据流。对于基于 AI 引擎的设计（带宽需求更高），使用后一种方法创建测试工具则更有效。但这需要额外增加工作。

使用 Vitis 工具可执行额外的硬件调试，如《Vitis 统一软件平台文档》 中所述。

系统确认

完成各子系统的确认后，即可在 Vitis 环境中对整个系统进行集成。您可使用本章中所述确认方法来对整个系统进行确认。

设计调试

如果确认 (validation) 与验证 (verification) 结果之间出现功能不匹配，您可按 第 5 章：配置与调试 中所述来调试设计。

附录 A

附加资源与法律声明

查找其他文档

文档门户

AMD 自适应计算文档门户是旨在使用您的网页浏览器提供健全的文档搜索和导航的在线工具。要访问文档门户，请转至 https://docs.xilinx.com。

注释：单击链接将打开英语版本，但您可从下拉列表中选择简体中文版本（如可用）。请注意，简体中文版本可能比英语版本旧。

Documentation Navigator

Documentation Navigator (DocNav) 是预安装的工具，支持访问 AMD 自适应计算文档、视频和支持资源，您可在其中通过筛选和搜索来查找信息。要打开 DocNav，请执行以下操作：

- 在 AMD Vivado™ IDE 中，单击“Help” → “Documentation and Tutorials”。
- 在 Windows 上，单击“Start”（开始）按钮并选中“Xilinx Design Tools” → “DocNav”。
- 在 Linux 命令提示中输入 `docnav`。

注释：如需了解有关 DocNav 的更多信息，请参阅《Documentation Navigator 用户指南》(UG968)。

注释：您无法从 DocNav 访问简体中文版本。请使用设计中心网页。

设计中心 (Design Hub)

AMD 设计中心提供了根据设计任务和其他主题整理的文档链接，可供您用于了解关键概念以及常见问题解答。要访问设计中心，请执行以下操作：

- 在 DocNav 中，单击“Design Hubs View”选项卡。
- 转至设计中心网页。

支持资源

如需获取答复记录、技术文档、下载以及论坛等支持资源，请访问技术支持。

参考资料

以下技术文档是非常实用的补充资料，可配合本指南一起使用：

1. 《Versal 自适应 SoC 技术参考手册》(AM011)
2. 《Versal 自适应 SoC NoC 和集成存储器控制器 NPI 寄存器参考资料》(AM019)
3. 《AXI Verification IP LogiCORE IP 产品指南》(PG267)
4. 《Versal Adaptive SoC Programmable Network on Chip and Integrated Memory Controller LogiCORE IP 产品指南》(PG313)
5. 《Versal Adaptive SoC Transceivers Wizard LogiCORE IP 产品指南》(PG331)
6. 《Virtual Input/Output (VIO) with AXI4-Stream Interface LogiCORE IP 产品指南》(PG364)
7. 《Vivado Design Suite Tcl 命令参考指南》(UG835)
8. 《Vivado Design Suite 用户指南：逻辑仿真》(UG900)
9. 《Vivado Design Suite 用户指南：综合》(UG901)
10. 《Vivado Design Suite 用户指南：使用约束》(UG903)
11. 《Vivado Design Suite 用户指南：实现》(UG904)
12. 《Vivado Design Suite 用户指南：设计分析与收敛技巧》(UG906)
13. 《Vivado Design Suite 用户指南：功耗分析与最优化》(UG907)
14. 《Vivado Design Suite 用户指南：编程和调试》(UG908)
15. 《Vivado Design Suite 属性参考指南》(UG912)
16. 《适用于 FPGA 和 SoC 的 UltraFast 设计方法指南》(UG949)
17. 《Vivado Design Suite 用户指南：采用 IP integrator 设计 IP 子系统》(UG994)
18. 《AI 引擎工具和流程用户指南》(UG1076)
19. 《AI 引擎内核与计算图编程指南》(UG1079)
20. 《Versal 自适应 SoC 设计指南》(UG1273)
21. 《Versal 自适应 SoC 系统软件开发者指南》(UG1304)
22. 《Versal 自适应 SoC 硬件、IP 和平台开发方法指南》(UG1387)
23. 《Vitis 统一软件平台文档：应用加速开发》(UG1393)
24. Vitis HLS 用户指南 (UG1399)
25. 《Vitis 嵌入式软件开发流程文档》(UG1400)
26. 《Versal 自适应 SoC 系统和解决方案规划方法指南》(UG1504)
27. 《Versal 自适应 SoC 开发板系统设计方法指南》(UG1506)
28. 《SmartLynq+ 模块用户指南》(UG1514)

修订历史

下表列出了本文档的修订历史。

章节	修订综述
2023 年 11 月 15 日 2023.2 版	
使用 SLR 交汇寄存器	更新示例。
使用 IMUX 寄存器改善硬核宏的时序	新增章节。
DRC 收敛	新增章节
改善 NoC 性能	修改章节。
调试配置	新增章节。
调试开发板初始化	新增章节。
功耗调试	新增章节。
调试 NoC	添加 NoC 调试博客链接。
调试 DDR 存储器控制器	新增章节。
调试 PCIe	新增章节。
2023 年 5 月 24 日 2023.1 版	
文档标题	标题更改为《Versal 自适应 SoC 系统集成和确认设计方法指南》(UG1388)。
调试系统	新增章节。

请阅读：重要法律声明

版权声明