

 查看所有版本

适用于 FPGA 和 SoC 的 UltraFast 设计方法指南

UG949 (v2025.1) 2025 年 5 月 29 日

本文档为英语文档的翻译版本，若译文与英语原文存在歧义、差异、不一致或冲突，概以英语文档为准。译文可能并未反映最新英语版本的内容，故仅供参考，请参阅最新版本的英语文档获取最新信息。

AMD 自适应计算矢志不渝地为员工、客户与合作伙伴打造有归属感的包容性环境。为此，我们正从产品和相关宣传资料中删除非包容性语言。我们已发起内部倡议，以删除任何排斥性语言或者可能固化历史偏见的语言，包括我们的软件和 IP 中嵌入的术语。虽然在此期间，您仍可能在我们的旧产品中发现非包容性语言，但请确信，我们正致力于践行革新使命以期与不断演进的行业标准保持一致。如需了解更多信息，请参阅此[链接](#)。



目录

第 1 章：简介	4
关于 UltraFast 设计方法	4
了解 UltraFast 设计方法论的概念	6
使用 Vivado Design Suite	10
第 2 章：开发板和器件规划	11
PCB 布局建议	11
器件功耗方面和系统依赖关系	15
时钟资源规划与分配	17
I/O 管脚分配设计流程	18
采用 SSI 器件进行设计	22
采用 HBM 器件进行设计	28
配置	32
第 3 章：使用 RTL 创建设计	34
定义理想的 RTL 设计层级	34
IP 的使用	37
RTL 编码准则	40
时钟设置准则	70
时钟域交汇	112
第 4 章：设计约束	116
对设计约束进行组织以便执行编译	116
定义时序约束	120
定义功耗和散热约束	146
定义物理约束	146
第 5 章：设计实现	147
运行综合	147
综合后的步骤	153
实现设计	157
第 6 章：设计收敛	163
时序收敛	163
功耗收敛	233
DRC 收敛	237
配置与调试	238

附录 A：附加资源与法律声明.....	245
查找其他文档.....	245
支持资源.....	245
参考资料.....	246
培训资料.....	248
修订历史.....	248
请阅读：重要法律声明.....	249

简介

关于 UltraFast 设计方法

AMD UltraFast™ 设计方法论是一整套旨在帮助简化 AMD UltraScale™、AMD UltraScale+™、7 系列和更早的器件系列的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。

- 本指南中描述了各种设计任务、分析与报告功能，以及用于设计创建和收敛的最佳实践。
- 《UltraFast 设计方法快捷参考指南》(UG1231) 中采用易于使用的双面卡格式着重讲解了主要设计方法论步骤。
- 《UltraFast 设计方法时序收敛快捷参考指南》(UG1292) 提供了有关时序收敛的建议，包括运行初始设计检查、设定设计基线和解决时序违例。
- 《UltraFast 设计方法检查表》(XTP301) 可从 AMD Documentation Navigator 获取，也可作为独立电子数据表获取。您可使用此检查表来识别整个设计进程中的常见错误和决策点。
- 《UltraFast 设计方法论系统级设计流程》图展示了 AMD Documentation Navigator 中提供的整个 AMD Vivado™ Design Suite 设计流程。您可单击图中的设计步骤，打开相关文档、附属资料以及常见问题解答以帮助您入门。



建议：除上述资源外，AMD 建议，处理嵌入式设计时请参阅《UltraFast 嵌入式设计方法指南》(UG1046)，通过 Vivado IP integrator 使用基于 C 语言的 IP 开发复杂系统时，则请参阅《Vitis HLS 用户指南》(UG1399) 中的 [HLS 编程指南](#)。

AMD 提供了以下资源以帮助您有效利用 UltraFast 设计方法论的优势：



提示：AMD 还为每个设计阶段提供了方法论相关的设计规则检查 (DRC)，可在 Vivado Design Suite 中使用 `report_methodology` Tcl 命令获取。

如何使用本指南

本指南为最大限度提高系统集成和设计实现的生产力提供了一套最佳实践方法。其中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：开发板和器件规划](#)：其中涵盖了 AMD 建议在创建设计前完成的决策和设计任务。包括 I/O 管脚分配和时钟规划、PCB 布局注意事项、器件容量和吞吐量评估、替代器件定义、功耗估算以及调试。
- [第 3 章：使用 RTL 创建设计](#)：涵盖有关 RTL 定义以及 IP 配置和管理的最佳实践。
- [第 4 章：设计约束](#)：提供相关建议，用于创建适当的时序、功耗和物理约束，以及用于指定综合与实现阶段所使用的其他约束、属性及其他元件。
- [第 5 章：设计实现](#)：涵盖适用于设计的综合与实现的可用选项和最佳实践。

- **第 6 章：设计收敛：**涵盖用于在设计上达成时序收敛或降低功耗估算的各种设计分析和实现方法。还包含为设计添加调试逻辑以便对硬件进行验证的相关注意事项。

本指南包含对其他文档的参考，例如，《Vivado Design Suite 用户指南》、《Vivado Design Suite 教程》和《QuickTake 视频教程》等。本指南不能替代上述文档。AMD 仍建议您参阅这些文档以获取详细信息，包括有关工具使用方式和设计方法的描述。

此信息旨在配合 Vivado Design Suite 一起使用，但其中大部分概念信息同样适用于 ISE® Design Suite。

相关信息

附加资源与法律声明

使用 UltraFast 设计方法检查表

要充分利用 UltraFast 设计方法论，请将本指南与《UltraFast 设计方法检查表》([XTP301](#)) 配合使用。本检查表可从 AMD Documentation Navigator 获取，或者也可以独立电子数据表形式获取。

“UltraFast 设计方法检查表”中的问题着重讲解了设计决策可能产生不利影响的典型领域，并要求读者重点关注经常被忽视或忽略的问题。检查表中的每个选项卡：

- 面向常规设计团队中的特定角色。
- 包含每个设计流程步骤中的常见问题和建议的应对措施，包括工程规划、开发板和器件规划、IP 和子模块设计以及顶层设计收敛。
- 包含“文档与培训”章节，其中列出了设计流程步骤相关的资源。
- 提供指向本指南或其他 AMD 文档中的内容的链接，以提供有关解决因各种问题而引发的设计问题的指导意见。



视频：要获取检查表的演示，请观看《[Vivado Design Suite QuickTake 视频：UltraFast 设计方法检查表简介](#)》。

使用 UltraFast 设计方法论 DRC

Vivado Design Suite 包含一组方法论相关 DRC，可供您使用 `report_methodology` Tcl 命令来运行。此命令针对以下每个设计阶段都具有相应的规则：

- 在综合前，在细化 RTL 设计中用于确认 RTL 结构
- 在综合后，用于确认网表和约束
- 在实现后，用于确认约束和时序相关问题。



建议：为了最大限度发挥作用，请在每个设计阶段运行方法论 DRC，并解决其中的严重警告 (Critical Warnings) 和警告 (Warnings)，然后再继续执行下一个阶段。

如需了解有关设计方法论 DRC 的更多信息，请参阅《[Vivado Design Suite Tcl 命令参考指南](#)》([UG835](#)) 中的 `report_methodology` Tcl 命令。

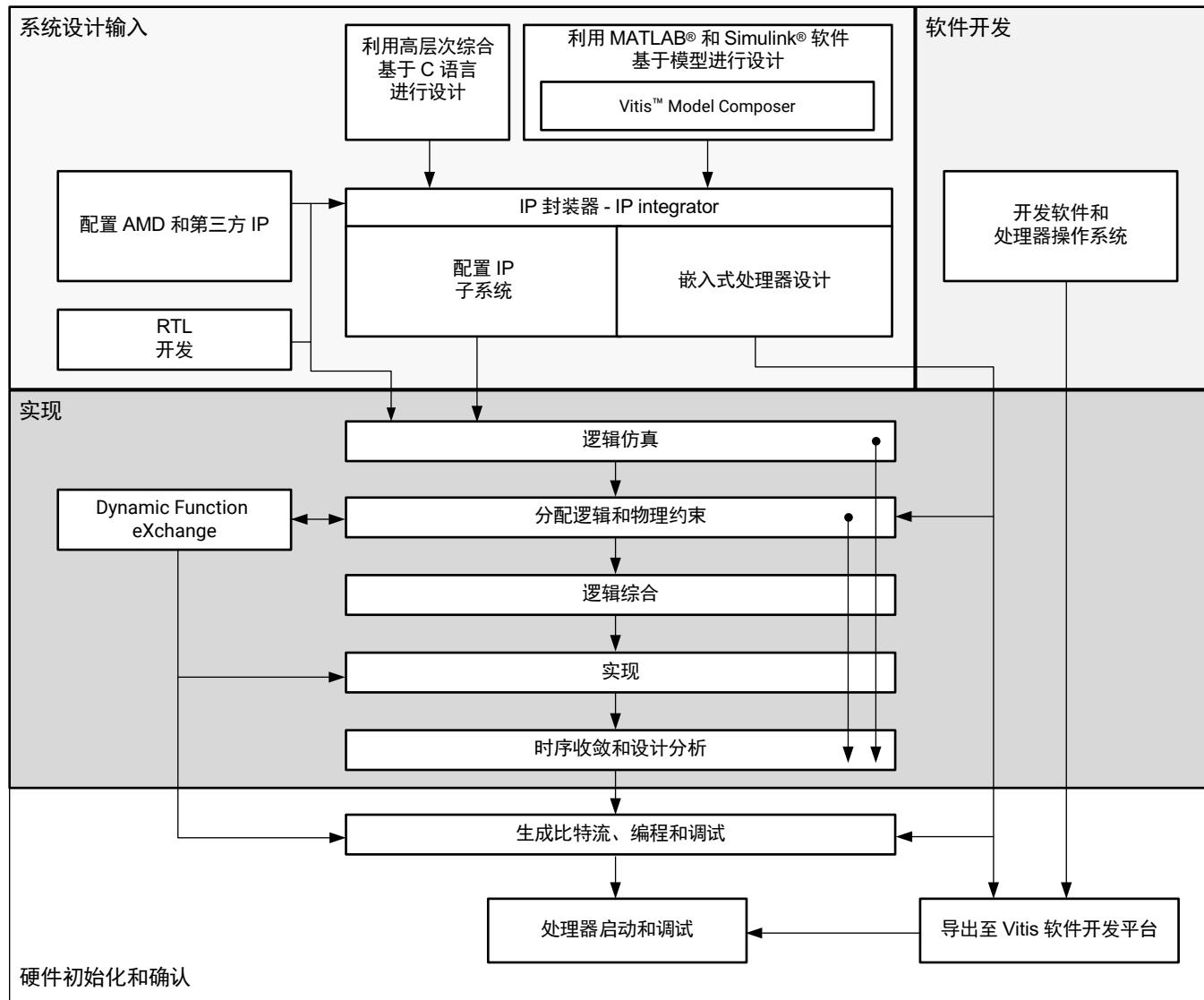
相关信息

运行 Report Methodology

使用 UltraFast 设计方法系统级设计流程图

下图显示了 Vivado Design Suite 中包含的各种设计步骤和特性。您可通过 AMD Documentation Navigator 的“Design Hub View”来访问此图的交互版本，并可在其中单击每个步骤访问相应链接以获取相关资源。

图 1：面向 FPGA 和 SoC 的系统级设计流程



X15150-063021

了解 UltraFast 设计方法论的概念

从设计之初即采用正确方法，从早期阶段开始对设计目标（包括 RTL、时钟、管脚和 PCB 管脚分配）给予足够的重视，这些对于确保设计成功都至关重要。在每个设计阶段中务必正确定义和确认设计，这有助于缓解后续实现阶段的时序收敛、布线收敛和功耗问题。

创建和实现硬件设计

完成器件 I/O 管脚分配、PCB 布局规划并决定使用模型后，即可开始创建设计。设计创建包括：

- 规划设计的层级
- 识别要在设计中使用和定制的 IP 核
- 例化 IP 目录中不可用的特殊互连或功能所需的 RTL 模块
- 创建时序约束、功耗约束和物理约束
- 指定综合与实现阶段所使用的其他约束、属性及其他元件

创建设计时，主要的考虑要素包括：

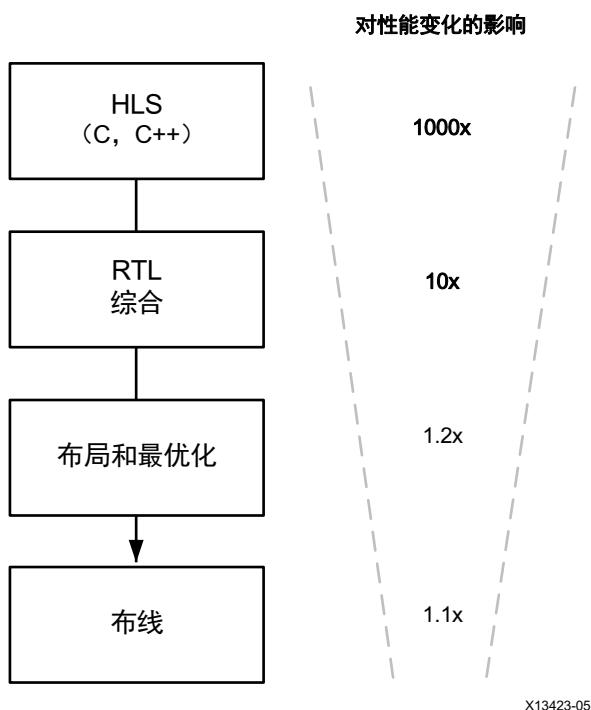
- 实现所需的功能
- 按期望的频率运行
- 按期望的可靠程度运行
- 符合硅片资源和功耗预算要求

在此阶段做出的决策将影响最终产品。在这一阶段的错误决策会导致后续阶段问题层出不穷，进而造成整个设计周期中不断返工。在此过程中尽早花时间详细规划设计有助于达成设计目标并最大限度缩短实验室中的调试时间。

在开发周期早期最大限度扩大影响

如下图所示，设计流程的早期阶段（C、C++ 和 RTL 综合）对于设计性能、密度和功耗的影响远超后期实现阶段的影响。因此，如果设计不满足时序目标，AMD 建议您重新评估综合阶段（包括 HDL 和约束），而不是仅在实现阶段通过迭代来寻找解决方案。

图 2：整个流程中设计变更的影响



在每个设计阶段进行确认

UltraFast 设计方法强调对设计预算（例如，面积、功耗、时延和时序）进行监控以及尽早采取如下措施更正设计的重要性：

- 利用 AMD 模板创建最佳 RTL 结构，并在执行细化后进行综合前采用方法 DRC 来确认 RTL。

由于 Vivado 工具从始至终使用时序驱动的算法，设计必须从设计流程开始就加以正确约束。

- 在综合后开展时序分析。

要指定正确的时序，您必须分析设计中每个主时钟与相关的生成时钟之间的关系。在 Vivado 工具中，每次时钟交互都必须满足时序要求，除非显式声明为异步时钟交互或伪路径 (false path)。

- 在继续执行下一个设计阶段前采用正确的约束满足时序要求。

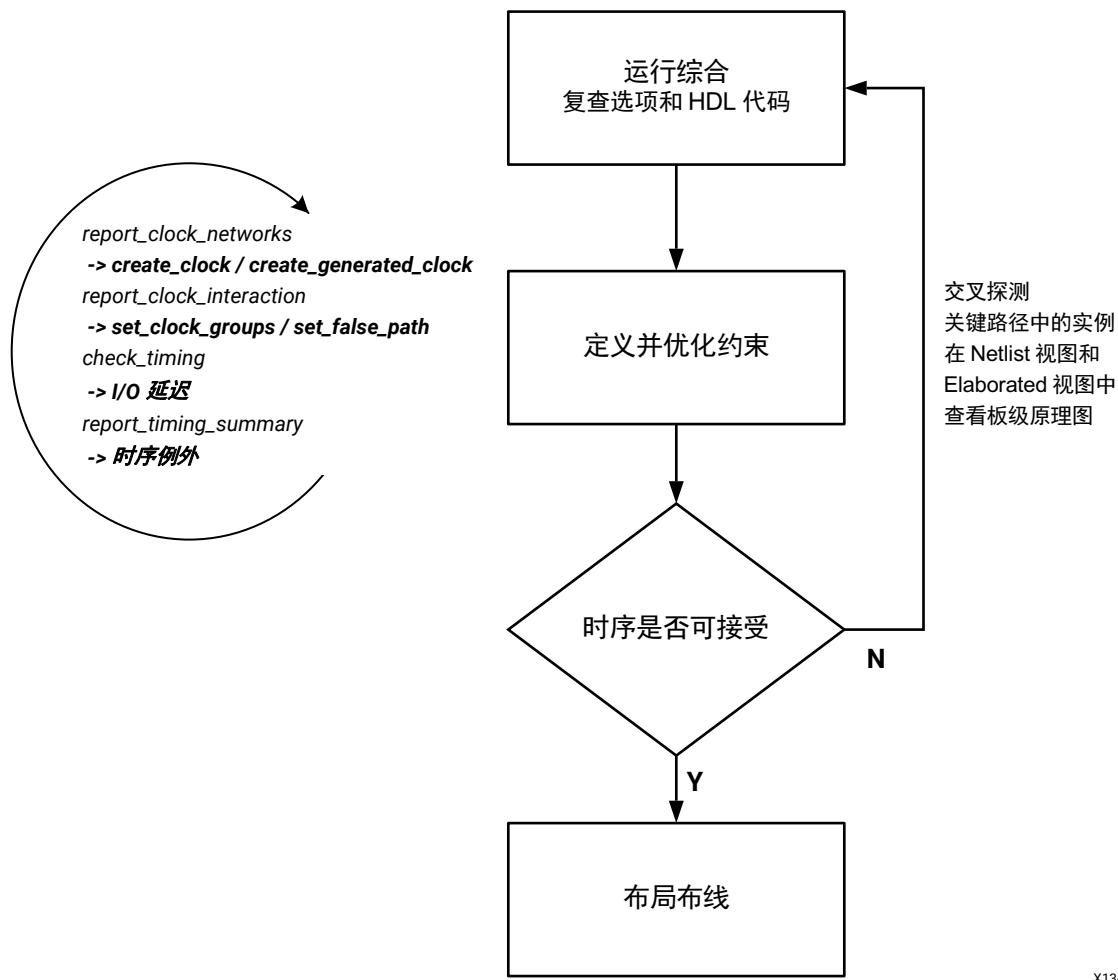
您可遵循如下建议并使用 Vivado Design Suite 的交互式分析环境来加速总体时序与实现收敛。



提示：您还可通过结合上述方法以及本指南中的 HDL 设计指南进一步加速收敛过程。

下图展示了这一推荐的设计方法。

图 3：实现快速收敛的 RTL 设计方法



如能够通过正时序裕度 (positive margin) 或相对较小的负时序裕度 (negative timing margin) 满足设计目标，那么综合即可视为完成。例如，如果综合后未能满足时序要求，那么布局布线结果也不太可能满足时序要求。然而，即便时序得不到满足，您仍然可以继续开展流程其余部分。如果实现工具能为失效的路径分配最佳资源，则可能能够收敛时序。此外，继续执行此流程可以更准确理解负时序裕量的量级，这有助于您确定综合后最差负时序裕量 (WNS) 所需的提升程度。改进 HDL 和约束后返回综合阶段时即可利用此信息。

利用快速确认的优势

本指南还介绍了对系统架构和微架构的具体某些方面进行快速确认的概念，如下所示：

- 在系统设计环境中，I/O 带宽采用系统内确认，此步骤在实现整个设计前完成。确认 I/O 带宽可以着重强调需要先修改系统架构和结构选择，然后才能在 I/O 上实现最终设计。
- 作为设计实现的一部分，基线设定可用于编写一组最简单的约束，以便明确内部器件的时序约束困难。在转入实现阶段前，基线设定用于明确是否需要修改 RTL 微架构选择。

相关信息

[接口带宽确认](#)
[设计基线设定](#)

使用 Vivado Design Suite

Vivado Design Suite 具有灵活的使用模型，可适应各种开发流程和不同类型的设计。如需了解有关如何使用 Vivado Design Suite 中的各项功能的详细信息，请参阅《Vivado Design Suite 用户指南：设计流程概述》([UG892](#)) 和其他 Vivado Design Suite 文档。

采用版本控制系统管理 Vivado Design Suite 源

大部分设计团队都采用商用的版本控制系统来管理自己的设计源与设计结果。Vivado Design Suite 支持通过各种使用模型来管理设计和 IP 数据。如需了解有关将 Vivado 工具与版本控制系统配合使用的更多信息，请访问此[链接](#)以参阅《Vivado Design Suite 用户指南：设计流程概述》([UG892](#)) 中的相应内容。

升级到新发行版的 Vivado Design Suite

新发行版的 Vivado Design Suite 通常包含 AMD IP 更新。请仔细考量您是否要升级自己的 IP，因为升级可能导致设计变更。此外，升级后如需使用通过先前发行版配置的 IP，必须遵循具体规则进行操作。欲知详情，请访问此[链接](#)以参阅《Vivado Design Suite 用户指南：采用 IP 进行设计》([UG896](#)) 中的相应内容。

开发板和器件规划

正确规划开发板上的器件朝向并向特定管脚分配信号可以显著提升总体系统性能、降低功耗、提升散热性能并缩短设计周期。将器件与印刷电路板 (PCB) 进行物理或逻辑交互的方式以直观方式展现出来，即可精简通过器件的数据流。

未正确规划 I/O 配置则可能导致系统性能下降和设计收敛时间延长。AMD 强烈建议在考虑 I/O 管脚分配的同时进行开发板级规划。

如需了解更多信息，请参阅以下资料：

- 《Vivado Design Suite 用户指南：I/O 管脚分配和时钟规划》([UG899](#))
- [Vivado Design Suite QuickTake 视频：I/O 管脚分配概述](#)

PCB 布局建议

开发板上的器件与可交互的组件之间的相对布局方式对 I/O 管脚分配影响巨大。

与 PCB 上的物理组件保持一致

首先应确定器件在 PCB 上的方向。还要考虑固定 PCB 组件的位置，以及内部器件资源。例如，调整器件封装上的 GT 接口，使该接口与 PCB 上与其相连的组件尽可能靠近，从而缩短 PCB 走线长度并减少 PCB 过孔数量。

包含关键接口在内的 PCB 草图对于确定 PCB 上器件的最佳方向以及 PCB 组件布局很有帮助。完成后，即可规划器件 I/O 接口的其余部分。

高速接口（如存储器）与相连的 PCB 组件之间最好采用直接连接，并且连接越短越好。这些 PCB 走线长度通常必须尽可能相匹配并且不使用 PCB 过孔。在此类情况下，最好采用距离器件边缘最近的封装管脚以尽可能缩短连接并避免布线超出 BGA 管脚的大型矩阵范围。

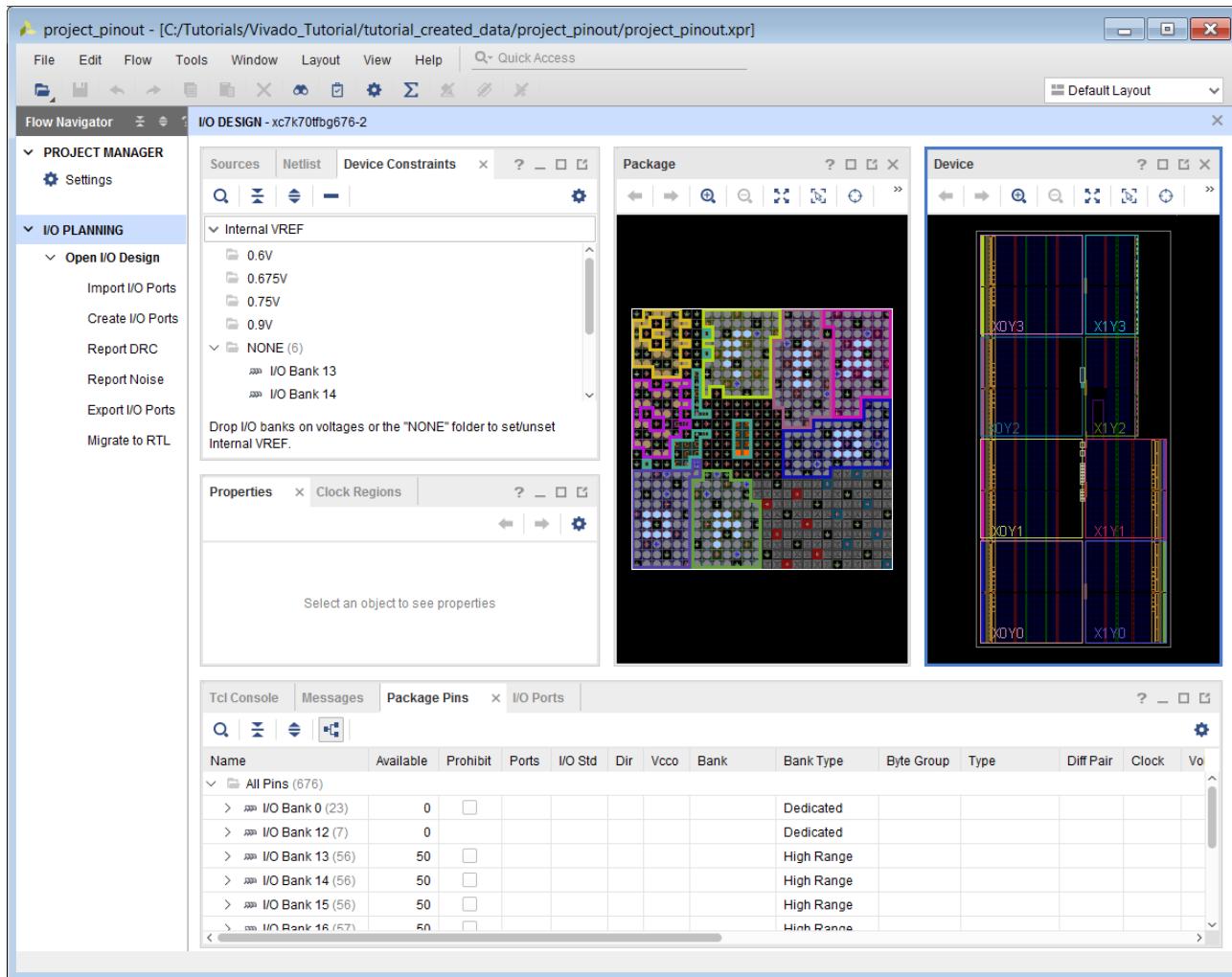
在此阶段，AMD Vivado™ IDE 中的“*I/O Planning*”（I/O 管脚分配）视图布局可用于直观显示与物理器件尺寸相关的 I/O 连接，它可显示顶部视图和底部视图。



散热提示：对于热管理存在难度的设计，请注意与其他高功耗组件相关的器件布局以最大限度减少热耦合并增加空气流动。避免将器件布局在另一个高功耗组件的散热气流内，或是开发板上热量能对器件工作温度产生不利影响的位置。AMD 建议采用热仿真以了解布局和环境条件对于器件结温可能产生的影响。

下图展示了“*I/O Planning*”视图布局。

图 4：“I/O Planning”视图布局



配电系统

开发板设计师在设计 AMD 器件的配电系统 (PDS) 时面临着 1 项特殊的任务。大部分其他大型密集集成电路（例如，大型微处理器）都附带有具体的旁路电容器要求。因为这些器件设计为仅在其硬化的硅片架构内实现特定任务，因此其电源需求是固定的，通常在某一范围内浮动。

AMD 自适应 SoC 并不具有此特性。器件可在多个时钟域内按您确定的频率实现几乎无限数量的应用。

因此，请务必使用 Xilinx Power Estimator (XPE) 电子数据表工具（从 www.amd.com/power 下载）为 7 系列和 AMD UltraScale™ 器件估算功耗，使用电源设计管理器 (PDM) 工具（从 www.amd.com/power 下载）为 AMD UltraScale+™ 器件估算功耗。此外，在进行功耗估算前，请参阅对应您的器件的《PCB 设计指南》以充分了解 PDS 布局和通用去耦要求。

PDS 设计期间需要考量的关键因素包括：

- 根据功耗估算，选择合适的电压调节器以满足噪声和电流要求。

注释：为充分支持并简化电源设计，AMD 与各大电源供应商协作，对可满足所有电源功耗要求的各种参考设计进行设计、构建、记录和测试。如需了解更多信息，请参阅[电源效率](#)网页上的“供电解决方案”选项卡。

- 整合电源。如需了解 UltraScale 器件中支持的整合选项，请参阅《UltraScale 架构 PCB 设计用户指南》(UG583) 中的“UltraScale 器件中的配电系统”。

电源/功耗提示：AMD 建议添加 1 个分流电阻以便监控每条轨道上的电源。或者，您可使用启用 PMBus 的调节器或电流监控集成电路 (IC)。

- 设置 Sysmon 电源 (V_{CCAUX_SMON})。
- 运行配电网路 (PDN) 仿真。对于 UltraScale 器件，请按《UltraScale 架构 PCB 设计用户指南》(UG583) 中所列的建议数量来使用去耦电容，这些建议数量是根据指南中所列假定计算得出的。如果这些假定与您的设计不同，请对您的设计进行仿真以判定是否需要增加或减少去耦数量。运行 PDN 仿真有助于确认为保障电源不超出建议工作范围所需的去耦电容的准确数量。

注释：请参阅《7 系列 FPGA PCB 设计指南》(UG483)、《UltraScale 架构 PCB 设计用户指南》(UG583) 或《Zynq 7000 SoC PCB 设计指南》(UG933)，以获取有关您的器件的详细信息。

如需了解有关 PDN 仿真的更多信息，请参阅《使用 S 参数模型对 FPGA 功耗完整性进行仿真》(WP411)。

电源/功耗提示：AMD 建议使用 SIMetrix/SIMPLIS 中的 SIMPLIS 仿真器对电源设计进行仿真，以确保设计符合 AMD 建议的工作条件范围。大部分电源供应商都提供有限版本的 SIMPLIS 并提供模型以支持您运行此仿真。SIMPLIS 是第三方软件，用于电压调节器的瞬态分析和交流电源 (AC) 分析。如需了解有关供电仿真的更多信息，请联系 SIMPLIS 或您首选的供电供应商。

电源/功耗提示：Vivado 工具的 report_power 命令可基于每个调节器或电压调节器模块 (VRM) 来分析功耗，以确保每条供电轨上的所需电流不超过目标配电系统的承受能力。

相关信息

功耗收敛

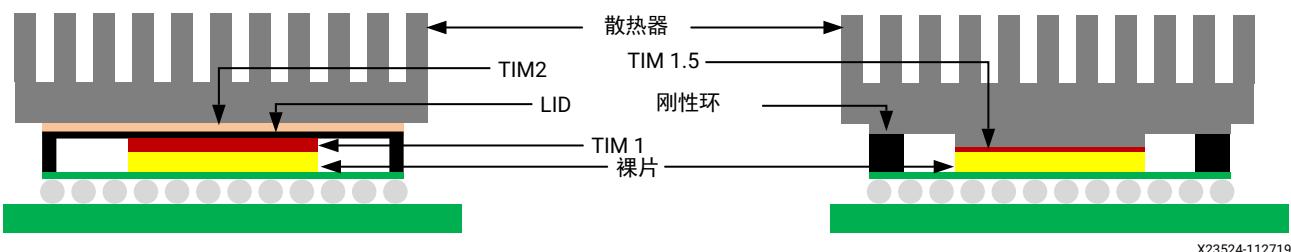
热处理解决方案注意事项

考量设计的功耗估算时，了解热处理解决方案的效率至关重要。结温越低，设计的静态功耗就越低。

AMD 建议，如果设计支持，请使用无盖封装。无盖封装可以提供更高效的热处理解决方案，并且允许直接接触热源，而无需导热介质 (TIM) 层。AMD 有盖部件和无盖部件的处理和制造要求相同。下图对有盖器件和无盖器件的散热器应用进行了比较。

散热提示：AMD 建议散热器每平方英尺 (PSI) 的磅力为 20 到 50 之间，这样可确保最大限度减小粘合层厚度 (BLT)，并且还建议使用 4 孔安装以确保有盖器件和无盖器件均可实现均衡压力。如需了解有关无盖技巧的更多信息，请参阅《无盖倒装芯片封装的机械和散热设计指南》(XAPP1301)。

图 5：散热器示例



AMD 还建议采用热仿真以确保裕度充足且功耗估算准确。在 Xilinx Power Estimator (XPE) 或电源设计管理器 (PDM) 工具（从 www.amd.com/power 下载）中，您可控制如下热处理设置：

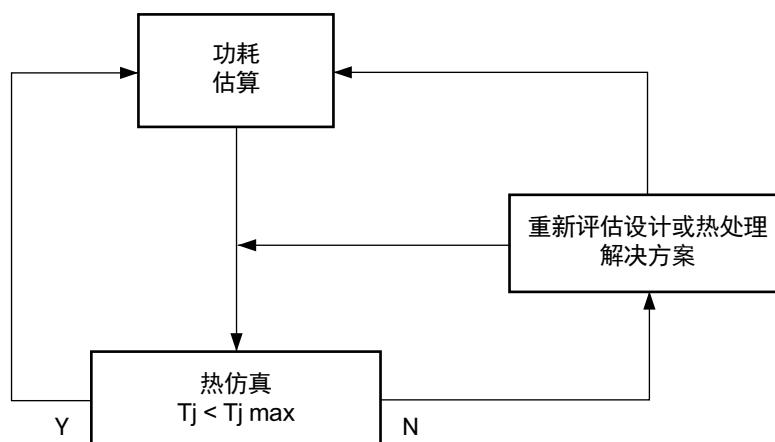
- 结温 T_j : 您可采用期望的结温覆盖此设置，以与热仿真相匹配。如果不运行热仿真，请将结温设置为最差情况。
- 有效 Θ_{JA} : 描述热处理解决方案的热效率，单位以每瓦摄氏度 ($^{\circ}\text{C}/\text{W}$) 来计量。例如， Θ_{JA} 为 $2.1^{\circ}\text{C}/\text{W}$ 表示，器件中每散失 1 瓦，结温就增加 2.1°C 。对于 10 W 设计，这将导致结温比环境温度高 21°C 。

注释：您可使用如下公式通过热仿真来获取 Θ_{JA} :

$$\Theta_{JA} = (T_j - T_a) / \text{PowerDissipated}$$

下图显示了热确认的建议流程。

图 6：建议的热确认流程



X23525-111319

当结温达到规格范围内并且认为裕度足够时，热处理解决方案即被视为有效。



散热提示：将功耗估算和热仿真结果添加到 Vivado 设计约束中。您可使用以下 XDC 约束，这些约束可从 XPE 或 PDM 工具导出，如《Xilinx Power Estimator 用户指南》(UG440) 或《电源设计管理器用户指南》(UG1556) 中所述：

```

# Standard Constraints: set_operating_conditions -process Maximum
set_operating_conditions -design_power_budget <value> #If thermal
simulation completed set_operating_conditions -ambient_temp <value>
set_operating_conditions -theta_ja <value> #Else if no thermal
simulation completed set_operating_conditions -junction_temp <value>
  
```

PCB 设计注意事项

在设计 PCB 时应考量与器件连接速度最快的信号。这些高速信号对于走线几何结构、过孔、损耗和串扰非常敏感。对于多层 PCB，这几个方面显得尤为重要。对于高速接口，请执行信号完整性仿真。必要时可以采用更先进的 PCB 材料或改变走线几何结构来重新设计开发板，以获得所期望的性能。

AMD 建议您在设计 PCB 时执行下列步骤：

1. 查阅以下器件文档：
 - 对应您的器件的《PCB 设计指南》。
 - 对应您的器件的《收发器用户指南》中的“开发板设计指南”。
2. 查阅 IP 产品指南中的存储器 IP 和 PCIe® 设计指南。

3. 使用 Vivado 工具来确认 I/O 管脚分配：
 - 运行同步开关噪声 (SSN) 分析。
 - 运行内置的 DRC。
 - 导出 I/O 缓冲器信息规格 (IBIS) 模型。
4. 按如下所述方式运行信号完整性分析：
 - 针对千兆位收发器 (GT)，使用信道参数运行 Spice 或 IBIS-AMI 仿真。
 - 对于性能更低的接口，运行 IBIS 仿真以检查过冲或下冲问题。
5. 使用 Xilinx Power Estimator (XPE) 或电源设计管理器 (PDM) 工具（从 www.amd.com/power 下载）并将“Process”（工艺）设置为“Maximum”（最大值）以生成设计功耗的初步估算。
6. 请将对应您器件的原理图检查表补充完整，并遵循该检查表进行操作。

注释：请参阅《7 系列板级原理图审查建议》([XMP277](#))、《Kintex UltraScale 和 Virtex UltraScale FPGA 板级原理图审查检查表》([XTP344](#)) 或《UltraScale+ FPGA 和 Zynq UltraScale+ 器件板级原理图审查检查表》([XTP427](#))。
7. 将 XDC 工作条件约束手动添加到 Vivado 工具的 XDC 文件中。使用 XPE 或 PDM 工具来生成赛灵思设计约束 (XDC) 文件，并将此文件导入对应的 Vivado 工程。XPE 和 PDM 工具环境设置会转换为 XDC 约束。估算的片上总功耗将作为 Vivado 功耗分析的设计功耗预算。如需了解更多信息，请参阅《Vivado Design Suite 用户指南：功耗分析与优化》([UG907](#))。

相关信息

[其他 AMD 文档](#)

器件功耗方面和系统依赖关系

规划 PCB 板时，请务必考虑功耗因素：

- 器件以及用户设计提出了系统电源与散热要求。
- 电源必须满足最高功耗要求，而器件在运行期间必须保持在建议的工作电压与温度条件下工作。需进行功耗估算和散热建模以确保器件保持在这些限制范围内。
- 提前规划电源轨的整合及其对功耗域开关的影响。
- 虽然可进行整合，但 AMD 建议使用全功耗管理来尽可能实现最大的灵活性。

因此，您必须了解器件的功耗和散热要求，并将这些要求一并纳入开发板设计过程。



电源/功耗提示：要获取 AMD 合作伙伴列表以及 AMD 核准的供电参考设计，请参阅[电源效率](#)网页。

器件上的供电路径

每个器件都需要多个电源来为其供电，并且这些电源必须按指定顺序提供。请考虑使用电源监控或顺序电路来保障器件以及开发板上的所有其他有源组件的上电顺序正确。在更为复杂的环境中，最好使用微控制器或系统和功耗管理总线（例如，SMBUS 或 PMBUS）来控制上电和复位流程。如需了解有关上电/断电顺序的具体详细信息，请参阅器件数据手册。如需了解有关电源整合和拓扑结构的更多信息，请根据您的器件参阅《7 系列 FPGA PCB 设计指南》([UG483](#))、《UltraScale 架构 PCB 设计用户指南》([UG583](#)) 或《Zynq 7000 SoC PCB 设计指南》([UG933](#))。

不同器件资源均通过单独的电源进行供电。这样可以让不同资源在不同的电压电平下工作，以提高性能或增强信号强度，同时有效避免产生噪声和寄生效应。

功耗类型

从上电到断电，器件要经过多个电源阶段，并伴有不同的功耗需求。

上电

上电功耗是器件首次上电发生的瞬时峰值电流。电压不同时，该电流强度也会发生变化且电流强度取决于器件的结构、电源缓升到标称电压的能力，以及器件的工作条件（比如温度以及不同电源之间的排序）。

在新型器件架构中，不用担心峰值电流的问题，只要遵循正确的上电顺序准则即可。

启动功耗

启动功耗即器件初始化和配置期间所需的功耗。此功耗通常发生于很短的时间内，因此无需考虑热耗散。但仍必须满足电流要求。大多数情况下，正常运行的设计的活动电流将更高，因此无需更改。但对于活动电流较低的低功耗设计，在此期间可能需要采用更高的电流要求。您可使用 Xilinx Power Estimator (XPE) 或电源设计管理器 (PDM) 工具（从 www.amd.com/power 下载）来解读此要求。当“Process”设置为“Maximum”时，每个电压轨的电流要求将指定为工作电流或启动电流中的较高者。如果启动电流更高，那么 XPE 或 PDM 工具会以蓝色显示电流值。

静态功耗

设计静态功耗（又称为“设计待机功耗”）是按设计对器件进行配置后，外部未施加任何活动且内部也未生成任何活动的情况下供电量。静态功耗表示设计运行时电源必须提供的最小连续功耗。

静态功耗是结温的函数。因此，确保正确完成环境和热处理解决方案参数建模对于功耗估算工具准确报告静态功耗至关重要。

相关信息

[建议的功耗约束](#)

动态功耗

动态功耗即器件运行应用并执行开关活动（如时钟和数据路径在逻辑值 High 和 Low 之间进行切换）时所需的功耗。动态功耗是根据一段时间内器件平均开关活动来计算的。总功耗包括静态功耗加上动态功耗。

影响功耗的因素

影响功耗的环境因素

除设计本身外，环境因素同样会影响功耗。这些因素会影响器件电压和结温，从而影响功耗损耗。欲知详情，请访问此[链接](#)以参阅《Vivado Design Suite 用户指南：功耗分析与优化》(UG907) 中的相应内容。



散热提示：-2LI 和 -2LE UltraScale+ 器件支持在指定时间段内发生温度漂移（上限为 110°C），从而降低热处理解决方案成本。如需了解更多信息，请参阅《利用温度漂移扩展热处理解决方案》(WP517)。

电源轨合并对功耗的影响

为了充分利用功耗域的功耗管理开关，您的设计必须保留一些独立的电源轨。这样即可通过电源域开关逻辑来对个别电源轨进行断电操作，但代价是需要额外增加电压调节器或调节器输出。如需了解更多信息，请参阅《UltraScale 架构 PCB 设计用户指南》(UG583) 中的“UltraScale 器件中的配电系统”。



提示：Vivado 工具也支持电源轨约束。欲知详情，请访问此[链接](#)以参阅《Vivado Design Suite 用户指南：功耗分析与优化》(UG907) 中的相应内容。

功耗模型精度

工具中嵌入的特性数据的准确性随时间而发生演变，以反映器件可用性或制造工艺成熟度。欲知详情，请访问此[链接](#)以参阅《Vivado Design Suite 用户指南：功耗分析与优化》(UG907) 中的相应内容。



电源/功耗提示：功耗估算的准确性取决于输入的数据。AMD 建议开展完整的估算，使用估算结果以及散热评估作为设计约束。

器件功耗和总体系统设计进程

从工程构思到完成，设计进程中各方面因素都会影响功耗。欲知详情，请访问此[链接](#)以参阅《Vivado Design Suite 用户指南：功耗分析与优化》(UG907) 中的相应内容。



电源/功耗提示：在设计进程中，可使用 `set_operating_conditions -design_power_budget <Power in Watts>` XDC 约束将设计的总功耗与功耗预算进行比较。如果超出功耗预算，最简便的设计功耗修正方式就是早期干预。

最差情况功耗分析

AMD 建议针对最差情况的功耗设计开发板。欲知详情，请访问此[链接](#)以参阅《Vivado Design Suite 用户指南：功耗分析与优化》(UG907) 中的相应内容。

时钟资源规划与分配

AMD 建议您在设计过程中尽早选择时钟资源，然后再选择管脚分配。您的时钟选择不仅可以决定特定管脚分配(pinout)，还可以指定该逻辑的逻辑布局，对于堆叠硅片互联(SSI)技术器件尤其如此。正确的时钟选择可产生卓越的效果。示例如下：

- 约束创建，尤其是大量使用时钟规划相关资源的大型器件。
- 根据设计收敛的需要手动进行时钟资源布局。
- 特定于器件的功能，此类功能可能需要提前规划以避免出现问题并充分利用器件功能。如需了解有关 7 系列功能的信息，请访问此[链接](#)和此[链接](#)以参阅《7 系列 FPGA 时钟资源用户指南》(UG472) 中的相应内容。如需了解有关 UltraScale 器件功能特性的信息，请参阅《UltraScale 架构时钟资源用户指南》(UG572) 中的“时钟资源”。

相关信息

[时钟设置准则](#)

[自动流水打拍注意事项](#)

[大宽度总线的 SLR 交汇](#)

I/O 管脚分配设计流程

Vivado IDE 支持您以交互方式浏览、直观显示、分配并确认设计中的 I/O 端口和时钟逻辑。此环境不仅可确保实现自动建构校正 (correct-by-construction) 式 I/O 分配。它还支持直观显示与内部裸片焊盘相关的外部封装管脚。

您可通过直观方式查看流经器件的数据流并从外部和内部双视角来正确规划 I/O。通过 Vivado IDE 完成 I/O 分配和配置后，即可为实现工具自动创建约束。

如需了解有关 Vivado Design Suite I/O 管脚分配和时钟规划功能的更多信息，请参阅如下资源：

- 《Vivado Design Suite 用户指南：I/O 管脚分配和时钟规划》([UG899](#))
- [Vivado Design Suite QuickTake 视频：I/O 管脚分配概述](#)

适合 I/O 管脚分配的 Vivado Design Suite 工程类型

您可对以下类型的工程执行 I/O 管脚分配：

- I/O 管脚分配工程：I/O 管脚分配工程作为简单的起点，支持您指定选定 I/O 约束，并从已定义的管脚生成顶层 RTL 文件。
- RTL 工程：RTL 工程允许综合和实现，支持更全面的设计规则检查 (DRC)。RTL 工程还允许生成 IP 核，这对于存储器接口管脚分配规划以及使用 GT 的任意核都至关重要。



提示：您还可先使用 I/O 管脚分配工程，稍后再移植到 RTL 工程。

您可以在综合后网表上运行更全面的 DRC。设计实现与比特流生成后也同样如此。因此，AMD 建议使用包含时钟组件和部分基本逻辑的骨架设计来实践 DRC。这有助于确保开发板的管脚定义后续不会引发任何问题。

推荐的验收流程为：运行 RTL 工程直至比特流生成环节，以实践全部 DRC。但是，并非所有设计周期都有足够时间用于完成此流程。通常必须先定义 I/O 配置，然后再实现可综合的 RTL。虽然 Vivado 工具支持 RTL 前 I/O 管脚分配，但只能执行基本级别的 DRC。或者，您可使用含 I/O 标准和管脚分配的虚拟顶层设计来帮助执行 bank 分配规则相关的 DRC。

RTL 前 I/O 管脚分配

如果设计周期强制要求在获取综合网表前定义 I/O 配置，请务必谨慎处理以确保符合所有相关规则。Vivado 工具包含“Pin Planning Project”（管脚分配工程）环境，支持您使用 CSV 或 XDC 格式文件导入 I/O 定义。您还可仅使用定义的端口方向来创建伪 RTL。提供端口方向可提高 SSN 分析准确性，因为输入信号和输出信号对 SSN 的影响不尽相同。

还可以交互式创建和配置 I/O 端口。基本 I/O bank DRC 规则已提供。

请参阅《7 系列 FPGA PCB 设计指南》([UG483](#))、《UltraScale 架构 PCB 设计用户指南》([UG583](#)) 或《Zynq 7000 SoC PCB 设计指南》([UG933](#))，以确保器件的 I/O 配置正确。欲知详情，请访问此[链接](#)以参阅《Vivado Design Suite 用户指南：I/O 管脚分配和时钟规划》([UG899](#)) 中的相应内容。

基于网表的 I/O 管脚分配

在设计周期中，建议在综合设计之后分配 I/O 和时钟逻辑约束。时钟逻辑路径是在网表中为进行约束分配而建立的。I/O 和时钟逻辑 DRC 同样更加全面。

请参阅《7 系列 FPGA PCB 设计指南》([UG483](#))、《UltraScale 架构 PCB 设计用户指南》([UG583](#)) 或《Zynq 7000 SoC PCB 设计指南》([UG933](#))，以确保器件的 I/O 配置正确。欲知详情，请访问此[链接](#)以参阅《Vivado Design Suite 用户指南：I/O 管脚分配和时钟规划》([UG899](#)) 中的相应内容。

识别与管脚兼容的器件

初始规划期间通常难以预测任意给定设计的最终器件大小。在设计周期的整个过程中可添加或移除逻辑，这可能导致需要更改器件大小。Vivado 工具支持您识别替代器件以确保定义的 I/O 管脚配置在所有选定器件间都兼容，前提是封装保持不变。欲知详情，请访问此[链接](#)以参阅《Vivado Design Suite 用户指南：I/O 管脚分配和时钟规划》([UG899](#)) 中的相应内容。



重要提示！ 器件必须位于同一封装中。



提示：要降低设计移植风险，在设计进程初请谨慎规划以下事项：器件选择、管脚分配 (pinout) 选择和设计标准。移植到相同封装内的更大或更小的器件时请考虑如下要素：管脚分配、时钟和资源管理。

管脚分配

合理的管脚选择可得到合理的设计逻辑布局、更短的布线、更低的功耗以及更高的性能。合理的管脚选择对于大型器件尤为重要，因为分散的管脚可能导致相关信号之间的距离更远。欲知详情，请访问此[链接](#)以参阅《Vivado Design Suite 用户指南：I/O 管脚分配和时钟规划》([UG899](#)) 中的相应内容。

使用 AMD 工具选择管脚分配

AMD 工具可帮助完成交互式设计规划与管脚选择。这些工具的有效性取决于您为其提供的信息。诸如 Vivado I/O Planner 等工具可帮助完成管脚分配相关工作。这些工具能够以图形化方式来显示 I/O 布局、显示时钟与 I/O 组件之间的关系以及提供用于分析管脚选择的 DRC。

如有设计版本可用，您可快速创建顶层布局规划以分析流经器件的数据流。如需了解更多信息，请参阅《Vivado Design Suite 用户指南：设计分析与收敛技巧》([UG906](#))。

所需的信息

为了使工具能够有效工作，您必须尽可能提供有关 I/O 特性和拓扑结构的详细信息。必须指定电气参数，包括 I/O 标准、驱动、斜率和 I/O 方向。

还必须考虑包括时钟拓扑和时序约束在内的所有其他相关信息。尤其是时钟选择，其可能对管脚选择产生重大影响，反之亦然。

对于具有 I/O 要求的 IP（例如，收发器、PCIe 以及存储器接口），必须在完成 I/O 管脚分配前配置 IP。如需了解有关指定 I/O 电气参数的更多信息，请访问此[链接](#)以参阅《Vivado Design Suite 用户指南：I/O 管脚分配和时钟规划》([UG899](#)) 中的相应内容。

相关信息

[时钟设置准则](#)

管脚分配选择

对于如下所述的某些特定信号，AMD 建议谨慎进行管脚选择。

适用于管脚分配选择的通用准则

通用准则如下所述：

- 将相同的接口数据、地址和控制管脚组合到同一个 bank 上。如果无法将这些组件组合到同一个 bank 上，请将其组合到相邻 bank 中。
注释：对于 SSI 技术器件，相邻 bank 也必须处于同一超级逻辑区域 (SLR) 中。
- 将以下接口控制信号布局在所控制的数据总线中间：时钟、使能、复位和选通。
- 将扇出极高且覆盖整个设计的控制信号布局在器件中心。
注释：对于 SSI 技术器件，请将 SLR 中的信号布局在所驱动的 SLR 组件中心位置。

配置管脚

要想设计一套高效的系统，必须选择充分满足系统要求的器件配置模式。需要考虑的因素包括：

- 使用专用或两用配置管脚。
- 每个配置模式会指定特定的专用器件管脚，仅限配置期间才能临时使用其他多功能管脚。配置完成后，这些多功能管脚将被释放以便用作通用管脚。
- 使用配置模式对某些器件 I/O bank 设置电压限制。
 - 为不同的配置管脚选择合适的终端。
 - 对配置管脚，使用建议的上拉或者下拉电阻值。



建议：尽管配置时钟速度较慢，但请在开发板上进行信号完整性分析，以确保信号无干扰。

有多种配置选项可供选择。虽然选项灵活多样，但是每个系统一般都有 1 个最佳的解决方案。在选择最佳配置选项时，请考虑以下方面：

- 设置
- 速度
- 成本
- 复杂性

如需了解有关器件配置选项的更多信息，请参阅《Vivado Design Suite 用户指南：烧录和调试》([UG908](#))。

相关信息

[配置](#)

存储器接口

使用 AMD 存储器 IP 时，需要执行额外的 I/O 管脚分配步骤。自定义 IP 后，即可将顶层 IP 端口分配到 Vivado IDE 中经过细化或综合的设计中的物理封装管脚。与每个存储器 IP 关联的所有端口组合在一起连接到同一个 I/O 端口接口，以便于识别和分配。提供的存储体/字节规划器 (Memory Bank/Byte Planner) 可帮助您将存储器 I/O 管脚分配到物理器件管脚上的字节通道中。欲知详情，请访问此[链接](#)以参阅《Vivado Design Suite 用户指南：I/O 管脚分配和时钟规划》([UG899](#)) 中的相应内容。

分配存储器接口时请谨慎处理，尽可能尝试限制拥塞，对于具有中央 I/O 列的器件尤其如此。密集排布存储器接口可能在器件中造成布线瓶颈。《Zynq 7000 SoC 和 7 系列器件的存储器接口解决方案》(UG586) 和《UltraScale 架构 FPGA 存储器 IP LogiCORE IP 产品指南》(PG150) 包含设计和管脚分配指南。请确保遵循这些指南中的走线长度匹配建议进行操作，验证使用的终端是否准确，并在完成 IP I/O 分配后通过运行 DRC 来确认管脚分配。如需获取有关存储器接口信号终止和布线指南的更多信息，请参阅《UltraScale 架构 PCB 设计用户指南》(UG583)。

千兆位收发器 (GT)

千兆位收发器 (GT) 具有特定的管脚分配 (pinout) 要求，您必须考量如下注意事项：

- 共享参考时钟
- 在四通道中共享 PLL
- PCIe 等硬核块的布局及其与收发器的距离
- 在 SSI 技术器件中发生的 SLR 边界交汇

AMD 建议您使用“GT Wizard”(GT 向导) 来生成核。或者，您也可以使用 AMD IP 核来获取协议。如需了解管脚分配建议，请参阅相关产品指南。

为实现时钟资源平衡，Vivado 布局器会尝试对由 GT 输出时钟 (TXOUTCLK 或 RXOUTCLK) 进行时钟设置的负载进行约束，使这些负载位于为时钟供电的 GT 旁。对于 SSI 技术器件，如果 GT 位于与另一个 SLR 相邻的时钟区域内，进出 SLL 所需的信号将不得不与 GT 输出时钟负载争用所需的布线资源。因此，位于 SLR 交汇附近的时钟区域内的 GT 可能会减少往来这些时钟区域内可用 SLL 交汇的可用布线连接。

高速 I/O

HP (高性能) 和 HR (高量程) bank 的信号发射和接收速度存在差异。请根据所需的 I/O 速度来选择采用 HP bank 或 HR bank。

内部 VREF 和 DCI 级联约束

根据“DCI Cascade”(DCI 级联) 和“Internal VREF”(内部 VREF) 的设置，可释放管脚以便用于常规 I/O。这些设置还可确保运行相关 DRC 检查以确认约束的规范性。欲知详情，请根据您的器件参阅《7 系列 FPGA SelectIO 资源用户指南》(UG471) 或《UltraScale 架构 SelectIO 资源用户指南》(UG571)。

接口带宽确认

创建小型连接设计以确认器件上的每个接口。这些小型设计仅运行特定硬件接口以支持：

- 针对管脚、时钟与时序执行完整 DRC 检查
- 在返还开发板时执行硬件测试设计
- 通过 Vivado 工具进行快速实现，以提供最快速的接口调试途径

有多种选项可用于辅助生成这些接口的测试数据。对于某些接口 IP 核，Vivado 工具可为测试设计提供：

- 针对 SerDes 的 IBERT
- IP 核内的设计示例



提示：如不存在测试设计，请考虑使用 AXI Traffic Generator。

您可能需要创建独立的设计以供在量产环境内进行系统级测试。通常这一独立设计包含经过测试的接口，也可选择在其中包含处理器。您可以使用小型连接设计来构建此设计，以便充分实现设计复用。虽然流程早期无需此设计，但它可支持执行更高效准确的 DRC 检查和早期软件开发，并且您可使用 Vivado IP integrator 快速创建此设计。

采用 SSI 器件进行设计

SSI 管脚分配注意事项

为位于特定 SLR 中的组件规划管脚分配时，请将这些管脚布局到相同 SLR 中。例如，使用器件 DNA 信息作为外部接口的一部分时，请将该接口的管脚布局到 DNA_PORT 所在的主 SLR 中。其他注意事项如下：

- 将特定接口的所有管脚都组合到同一个 SLR 中。
- 对于驱动多个 SLR 中的组件的信号，请将这些信号布局到中间 SLR 中。
- 在各 SLR 之间平衡布局支持时钟功能的 I/O (CCIO) 或时钟管理模块 (CMT) 组件。
- 减少 SLR 交汇。

超级逻辑区域 (SLR)

“超级逻辑区域 (SLR)” 是 SSI 技术器件中包含的单个器件裸片 slice。每个 SLR 都包含少量器件资源（例如，CLB、块 RAM、DSP 拼块 (tile) 和 GT），其结构与非 SSI 器件结构相似。

多个 SLR 组件采用纵向堆叠并通过中介层连接以构成 SSI 技术器件。底部 SLR 为 SLR0，后续 SLR 组件按纵向升序递增方式来命名。

例如，XC7V2000T 器件包含 4 个 SLR 组件。底部 SLR 为 SLR0，SLR0 正上方的 SLR 为 SLR1，SLR1 正上方的 SLR 为 SLR2，顶端 SLR 为 SLR3。

注释：AMD 工具会在图形用户界面 (GUI) 和报告中标明 SLR 组件。

SLR 命名法

了解目标器件的 SLR 命名法对以下方面非常重要：

- 管脚选择
- 布局规划
- 分析时序及其他报告
- 确认逻辑所在的位置以及逻辑的源端和目的端

您可使用 Vivado Tcl 命令 `get_slrs` 来获取有关特定器件的 SLR 的具体信息。例如，使用如下命令：

- `llength [get_slrs]` 可获取器件中的 SLR 数量
- `get_slrs -of_objects [get_cells my_cell]` 可获取 `my_cell` 所在的 SLR

主超级逻辑区域

每个 SSI 技术器件都有 1 个主 SLR。主 SLR 包含主配置逻辑，可初始化器件及其他所有 SLR 元件的配置。主 SLR 包含用于配置的电路、DNA_PORT 和 EFUSE_USER。使用这些组件时，布局布线工具可为合适的 SLR 分配相关管脚和逻辑。总之，无需额外干预。



提示：要查询 Vivado Design Suite 中哪个 SLR 是主 SLR，可输入 `get_slrs -filter IS_MASTER` Tcl 命令。

硅中介层

硅中介层是 SSI 技术器件中的无源层，通过在 SLR 组件间布线来实现：

- 配置
- 全局时钟设置
- 常规互连

超长线路 (SLL) 布线

超长线路 (SLL) 布线将器件内各 SLR 间的信号联通。



提示：为确定 SLR 间的可用 SLL 数量，请使用 SLR 属性。例如：

```
get_property NUM_TOP_SLLS [get_slrs SLR0]
get_property NUM_BOT_SLLS [get_slrs SLR1]
```

传输限制



提示：要实现跨 SLR 高速传输，请务必寄存跨 SLR 边界的信号。

SLL 信号是 SLR 组件之间的唯一数据连接。

下列信号不在 SLR 组件间传输：

- 进位链
- DSP 级联
- 块 RAM 和 UltraRAM 级联
- 其他专用连接，如 DCI 级联

工具通常会考量上述传输限制。为确保设计布线正确，并且符合您的设计目标，在以下情况下，您同样必须将此限制纳入考量范围：

- 构建 1 个超长 DSP、块 RAM 或 UltraRAM 级联，并手动将此逻辑布局在 SLR 边界附近
- 指定设计的管脚分配 (pinout)

SLR 使用率注意事项

Vivado 实现工具使用特殊算法将逻辑分区到多个 SLR 中。对于较困难的设计，可遵循如下准则改进对应 SSI 技术器件的时序收敛。

为改进时序收敛和编译时间，可使用 Pblock 来将逻辑分配到每个 SLR 并确认所有互连结构资源类型间的每个 SLR 都不存在使用率过高的问题。例如，对于块 RAM 使用率达 70% 的设计，如果在 SLR 间未平衡块 RAM 资源并且其中 1 个 SLR 的块 RAM 使用率超过 85%，那么可能导致时序收敛问题。



提示：您可通过指定完整 SLR（例如，`resize_pblock pblock_SLR0 -add SLR0`）来定义 SLR Pblock。

SLR 使用率示例

以下 vu160 使用率报告示例显示总块 RAM 使用率为 56%，其中 SLR0 为 59%，SLR1 为 40%，SLR2 为 58%。块 RAM 使用率平均分布在各 SLR 间，每个 SLR 中均采用合理使用率，以便为 Vivado 实现命令提供更多的灵活性，从而满足时序要求。

图 7：使用率报告中的块 RAM 部分

3. BLOCKRAM					

Site Type	Used	Fixed	Available	Util%	
Block RAM Tile	1843	0	3276	56.26	
RAMB36/FIFO*	1820	2	3276	55.56	
FIFO36E2 only	88				
RAMB36E2 only	1732				
RAMB18	46	8	6552	0.70	
RAMB18E2 only	46				

图 8：使用率报告中的 SLR 部分

14. SLR CLB Logic and Dedicated Block Utilization

Site Type	SLR0	SLR1	SLR2	SLR0 %	SLR1 %	SLR2 %
CLB	22361	40858	42493	61.70	91.28	94.94
CLBL	17061	31936	33311	60.76	90.99	94.90
CLBM	5300	8922	9182	64.95	92.36	95.05
CLB LUTs	104506	196677	236523	36.05	54.93	66.05
LUT as Logic	104482	194364	235584	36.04	54.28	65.79
using O5 output only	268	913	789	0.09	0.25	0.22
using O6 output only	101383	180318	219746	34.97	50.36	61.37
using O5 and O6	2831	13133	15049	0.98	3.67	4.20
LUT as Memory	24	2313	939	0.04	2.99	1.22
LUT as Distributed RAM	24	1316	136	0.04	1.70	0.18
using O5 output only	0	0	0	0.00	0.00	0.00
using O6 output only	8	48	64	0.01	0.06	0.08
using O5 and O6	16	1268	72	0.02	1.64	0.09
LUT as Shift Register	0	997	803	0.00	1.29	1.04
CLB Registers	191575	330568	473777	33.04	46.16	66.16
CARRY8	901	1715	3816	2.49	3.83	8.53
F7 Muxes	1725	10631	4762	1.19	5.94	2.66
F8 Muxes	216	2891	366	0.30	3.23	0.41
F9 Muxes	0	0	0	0.00	0.00	0.00
Block RAM Tile	599	512	732	59.42	40.63	58.10
RAMB36/FIFO	598	501	721	59.33	39.76	57.22
RAMB36E2 only	598	472	662	59.33	37.46	52.54
RAMB18	2	22	22	0.10	0.87	0.87
RAMB18E2 only	2	22	22	0.10	0.87	0.87
URAM	0	0	0	0.00	0.00	0.00
DSPs	1	2	12	0.21	0.33	2.00
PLL	0	0	0	0.00	0.00	0.00
MMCM	0	0	0	0.00	0.00	0.00
Unique Control Sets	1125	3482	8567	1.55	3.89	9.57

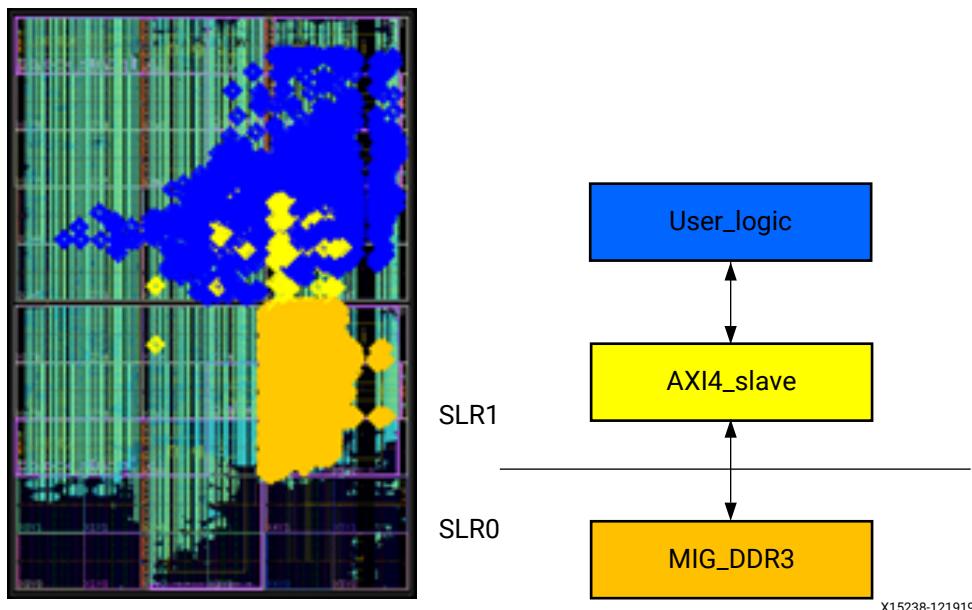
AMD 建议将块 RAM 和 DSP 组分配到各 SLR Pblock，以最大限度减少共享信号跨 SLR 的现象。例如，如果地址总线扇出到遍布于多个 SLR 上的一组块 RAM，则可能导致更难以实现时序收敛，因为 SLR 交汇会导致时序关键信号出现额外延迟。

器件资源位置或用户 I/O 选择会将 IP 锚定到 SLR，例如，GT、ILKN、PCIe 和 CMAC 专用块或存储器接口控制器。
AMD 建议：

- 应特别注意专用块位置与管脚选择，从而避免数据流多次出现跨 SLR 边界现象。
- 使相同 SLR 内的模块与 IP 保持紧密互连。如果无法实现，可以添加流水线寄存器来为布局器提供更大的灵活性，以便在逻辑组间存在 SLR 交汇情况下找到有效的解决方案。
- 确保关键逻辑处于同一 SLR 内。只要确保主模块在其接口处正确完成流水打拍，布局器找到含触发器到触发器 SLR 交汇的 SLR 分区的可能性就更高。

在下图中，约束到 SLR0 的存储器接口需要驱动 SLR1 中的用户逻辑。AXI4-Lite 从接口连接到存储器 IP 后端，而 IP 与 AXI4-Lite 从接口之间精确定义的边界可以提供从 SLR0 到 SLR1 的有效转换。

图 9：SLR0 中的存储器接口驱动 SLR1 中的用户逻辑

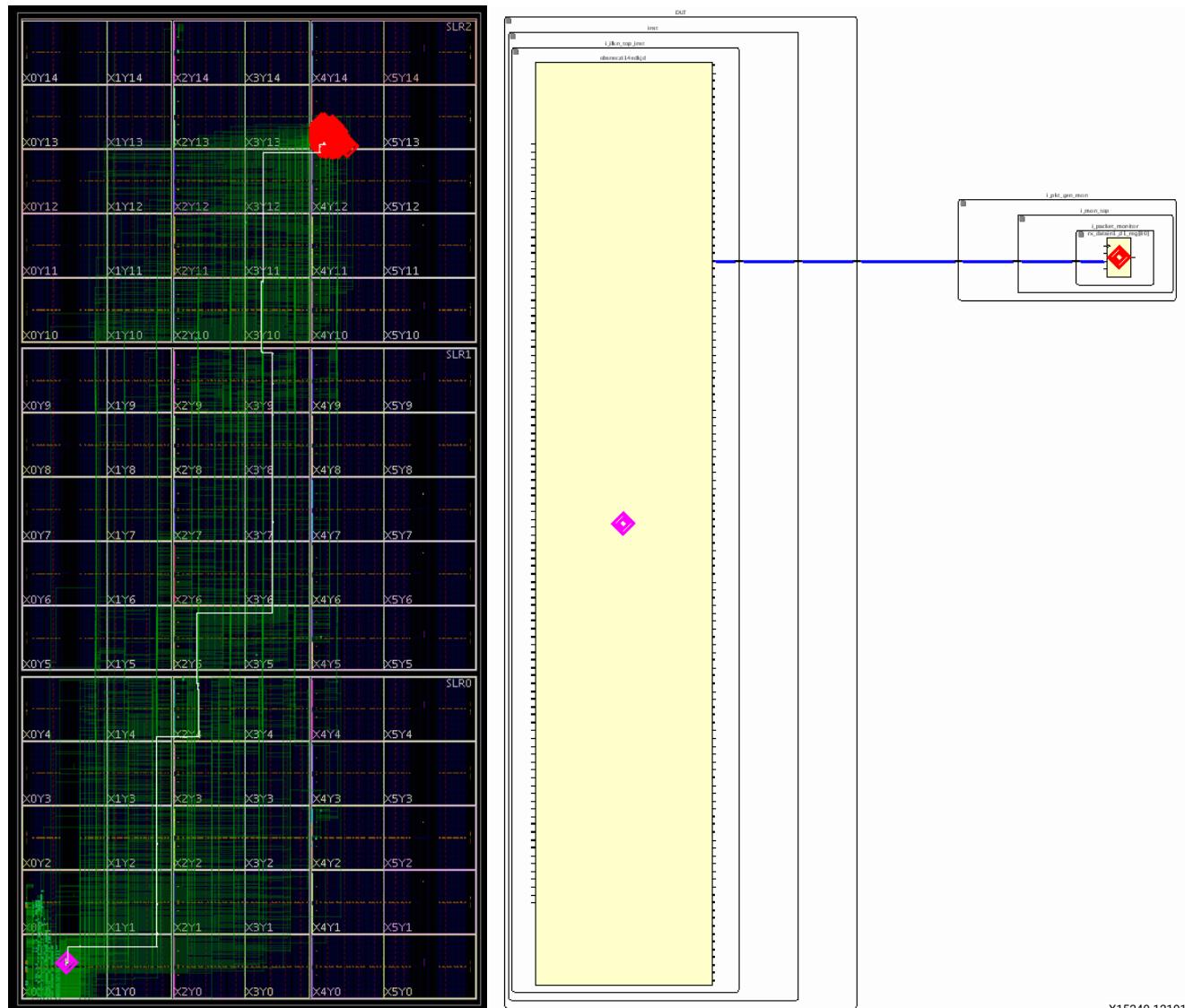


大宽度总线的 SLR 交汇

如果数据流要求中明确要求宽总线必须跨 SLR，请使用流水打拍策略来改进时序收敛并缓解长距离资源的布线拥塞。对于以高于 250 MHz 的频率运行的大宽度总线，AMD 建议使用至少 3 个流水线阶段来跨 1 个 SLR，这 3 个阶段分别位于 SLR 的顶部、底部和中间。对于超高时钟频率的总线，或者在遍历水平距离和垂直距离时，可能需要额外增加流水线阶段。

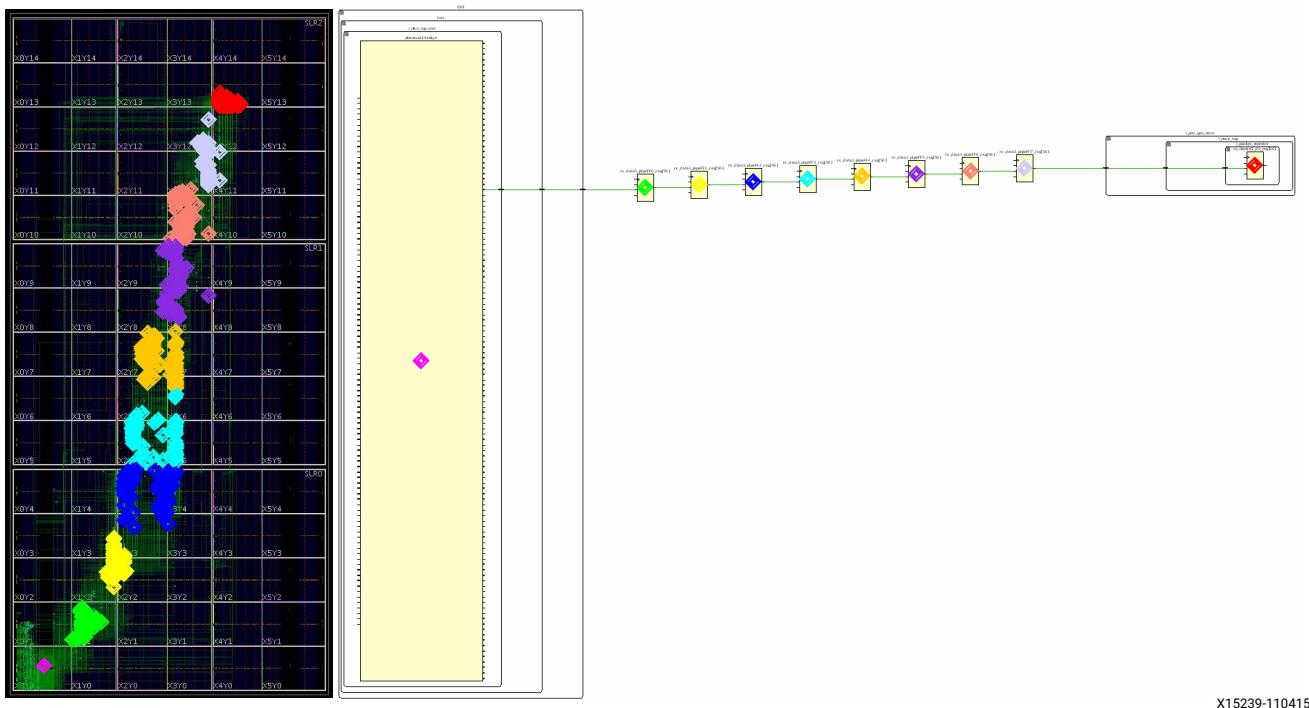
下图显示了 vu190-2 器件的最差情况交汇。本示例从 SLR0 左下角的 Interlaken 专用块开始，直至分配至 SLR2 右上角的包监控块为止。数据总线与包监控之间无往来流水线寄存器的情况下，设计距离 300 MHz 时序要求相去甚远。

图 10：跨 SLR（无流水线触发器）的数据路径



但是，添加 7 个流水线阶段有助于从 SLR0 遍历到 SLR2，从而帮助设计满足时序要求。这样还可减少垂直和水平长距离布线资源的使用，如下图所示。

图 11：已添加跨 SLR（含流水线触发器）的数据路径



提示：使用 AXI Register Slice IP 或定制自动流水打拍 IP 在跨 SLR 的大宽度总线上实现时序收敛。

相关信息

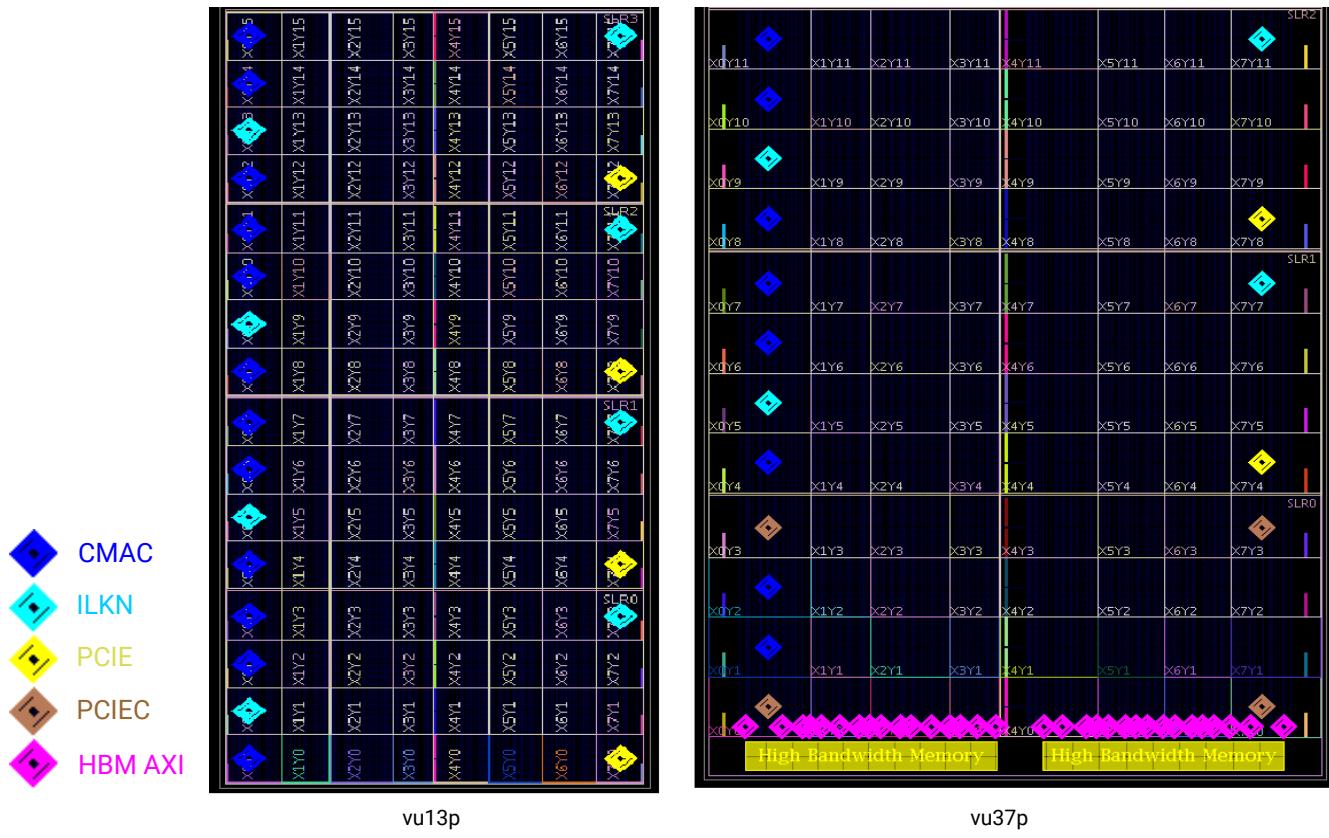
[自动流水打拍注意事项](#)

采用 HBM 器件进行设计

AMD Virtex™ UltraScale+™ HBM 器件在器件裸片附近整合了 4 GB 的高带宽存储器 (HBM) 堆栈。借助 SSI 技术，该器件通过存储器控制器与 HBM 栈进行通信，这些存储器通过器件底部的硅中介层进行连接。每个 Virtex UltraScale+ HBM 器件都包含 1 个或 2 个 4 GB HBM 堆栈，每个器件总计 8 GB 的 HBM。该器件包含 32 个 HBM AXI 接口，用于与 HBM 进行通信。内置开关提供了灵活的寻址功能，支持 32 个 HBM AXI 接口中的任一接口访问 1 个或 2 个 HBM 堆栈上的任何存储器地址。这种器件与 HBM 栈间的灵活连接功能有助于布局规划和时序收敛。

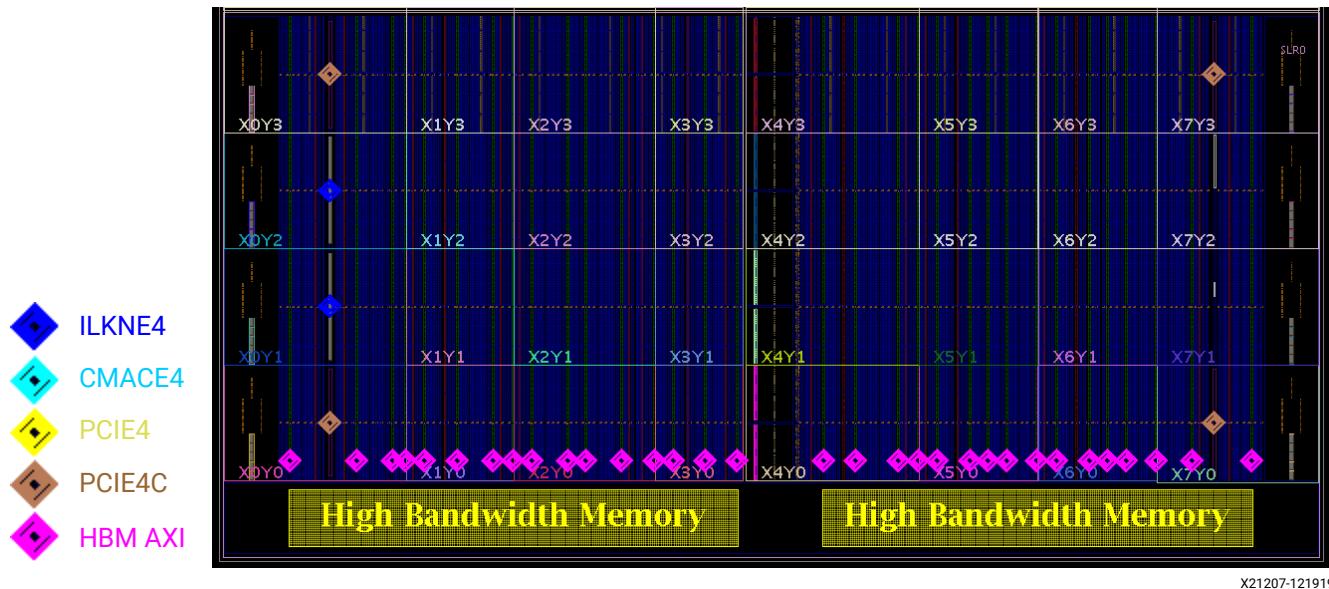
下图显示了与 Virtex UltraScale+ vu13p 器件相邻的 Virtex UltraScale+ HBM vu37p 器件。在 vu37p 器件中，vu13p 器件底部的 2 个 SLR 被替换为 HBM 栈（对应 vu13p 器件中的 SLR0）以及包含 32 个 HBM AXI 接口的 1 个 SLR（对应 vu13p 器件中的 SLR1）。vu13p 和 vu37p 器件的顶部 2 个 SLR 完全相同。

图 12: vu13p 和 vu37p 的“Device”视图



在 vu37p 器件中，SLR0 包含 4 个 PCIE4C 站点、2 个 ILKNE4 站点和 32 个 HBM AXI 接口。Virtex UltraScale+ HBM SLR0 中的 4 个 PCIE4C 的独特之处在于支持加速器缓存一致性互连 (CCIX) 协议，当 V_{CCINT} 为 0.72 V 时，使用 PCIe Gen3 x 16。

图 13: Virtex UltraScale+ HBM vu37p 器件的 SLRO



使用 HBM 器件的布局注意事项

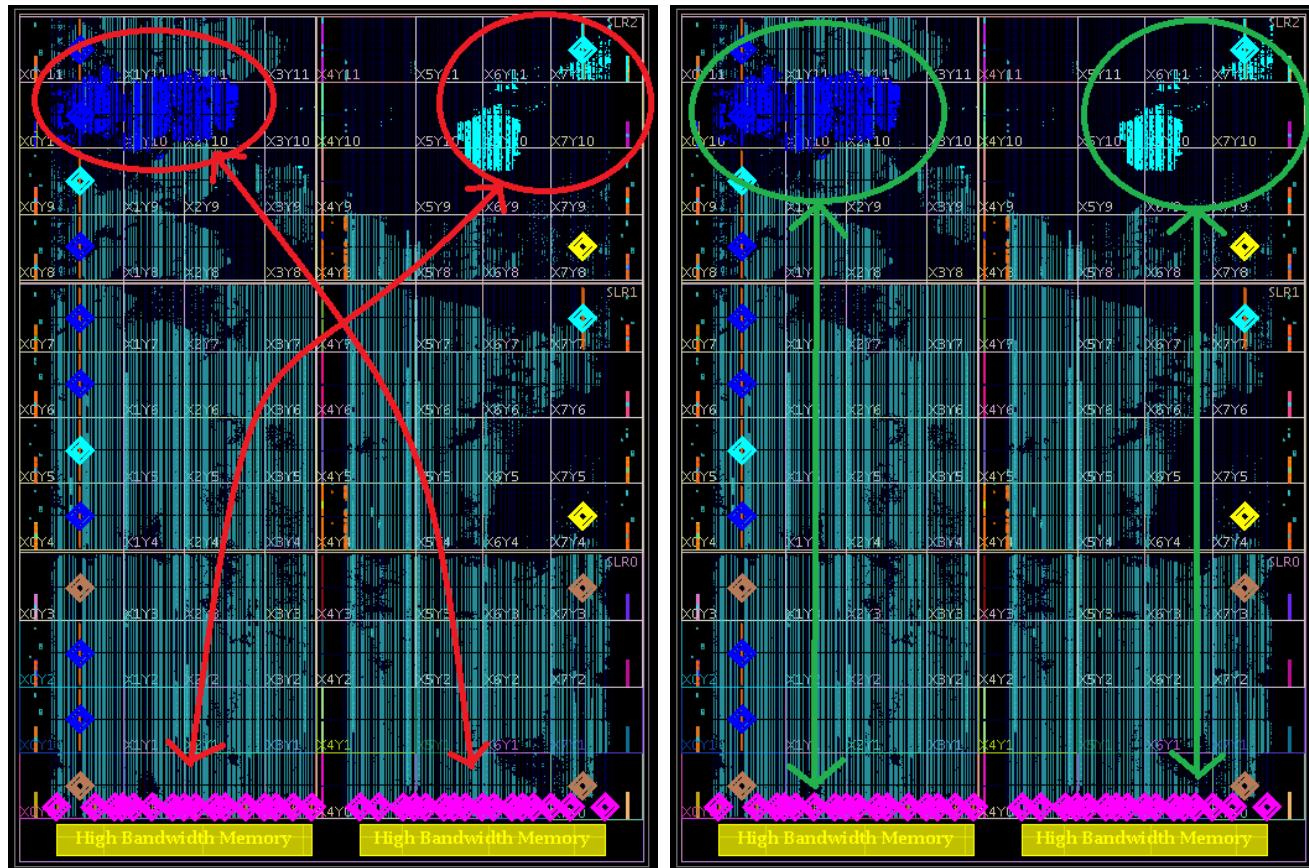
跨 SLR 流水打拍注意事项

Virtex UltraScale+ HBM 器件中跨 SLR 的流水线注意事项与其他 UltraScale 和 Virtex UltraScale+ SSI 技术器件相同。

从 SLR2 中的互连结构逻辑到 SLR0 中的 HBM AXI 接口的路径通常需要经历不少于 5 个流水线阶段才能满足时序。精心完成的 Virtex UltraScale+ HBM 器件设计规划可以消除对于额外增加流水线阶段的需求并减少布线拥塞。下图显示了从 SLR2 到 HBM 接口的跨 SLR 示例。

-
- 建议:** AMD 建议将来自 SLR2 和 SLR1 的路径与其相应的 HBM AXI 接口保持垂直对齐, 以避免对角穿过器件。
-
- 提示:** 使用自动流水打拍 (例如, AXI Register Slice IP) 可确保 HBM 接口与任意 450 MHz 的 SLR 之间实现时序收敛。
-

图 14：HBM 次优设计规划（左）对比最优设计规划（右）



X21196-121919

相关信息

[自动流水打拍注意事项](#)
[大宽度总线的 SLR 交汇](#)

SLR0 内的资源规划

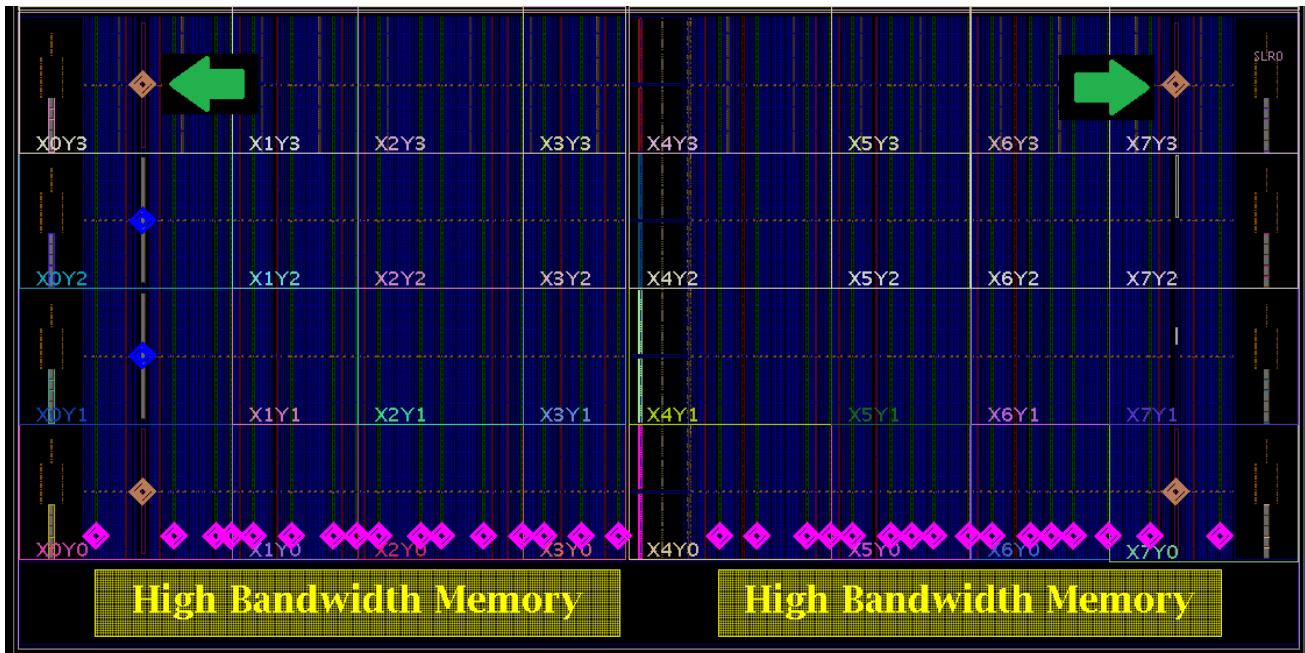
正确管理 HBM AXI 接口和 SLR0 中的其他逻辑可以提供最佳结果质量 (QoR) 并最大程度减少布线拥塞。以下是 HBM 器件中 SLR0 的部分常见设计规划注意事项：

- 对于大量使用 HBM AXI 接口的设计，请规划降低 SLR0 中非 HBM 逻辑的总体互连结构使用率，以适应 HBM AXI 接口的资源需求。
- 在 SLR0 中使用 MIG IP 会导致位于器件 I/O 列旁的 HBM AXI 接口难以达成时序收敛。当使用 MIG IP 时，考虑使用位于 SLR2 或 SLR1 中的 I/O 列。
- 请注意 HBM AXI 接口的地址范围和物理位置，这些可能会影响设计的时延和带宽。为了优化 HBM 性能，请在器件上已寻址的 HBM 堆栈所在的同一侧使用物理 HBM AXI 接口。

SLR0 中的 PCIE4C 到 HBM AXI 路径

采用 HBM 和 PCIE4C 进行设计时，为实现最优化时序 QoR 并最大限度减少布线拥塞，AMD 建议使用距离 SLR0 中的 32 HBM AXI 接口最远的 PCIE4C 站点 (site)。如下图所示，这些站点 (site) 包括 PCIE4CE4_X0Y1 和 PCIE4CE4_X1Y1 (以绿色箭头指示)。

图 15：Virtex UltraScale+ HBM vu37p 器件的 SLR0 中建议的 PCIE4C 站点 (site)



配置

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。由于 AMD 器件配置数据存储在 CMOS 配置锁存器 (CCL) 内，因此配置数据并不稳定，每次器件上电时都必须重新加载。

AMD 器件可通过配置管脚自行加载存放于外部非易失性存储器件的数据。器件还可通过如下外部智能电源进行配置：

- 微处理器
- 微控制器
- DSP 处理器
- 个人计算机 (PC)
- 开发板测试器

执行板级规划时，请提前规划配置以简化配置和调试。每个器件系列都包含《配置用户指南》，它是提供有关每项受支持的配置模式的详细信息及其管脚计数、性能与成本方面权衡取舍信息的主要资源。

相关信息

[其他 AMD 文档](#)

开发板设计技巧

设计开发板时，重要的是考量哪些接口和管脚除执行配置外还有助于调试功能。例如，AMD 建议您确保 JTAG 接口始终访问通畅，即便该接口并未处于主配置模式也是如此。JTAG 接口允许您检查器件 ID、检查器件 DNA 信息并读取关键状态寄存器以供调试。您还可以在原型设计期间使用此接口来启用间接闪存烧录解决方案。

此外，诸如 INIT_B 和 DONE 等信号对于器件配置调试至关重要。INIT_B 信号具有多项功能。它可在上电时指示初始化完成状态，并可指明遇到 CRC 错误的时间。AMD 建议您使用 LED 指示灯驱动程序和上拉电阻将 INIT_B 和 DONE 信号连接到 LED 指示灯。

要获取建议的上拉电阻值，请参阅对应您的器件的配置用户指南：

- 《7 系列 FPGA 配置用户指南》([UG470](#))
- 《UltraScale 架构配置用户指南》([UG570](#))

要识别和检查建议的板级管脚连接，请参阅板级原理图检查表：

- 《7 系列板级原理图审查建议》([XMP277](#))
- 《Kintex UltraScale 和 Virtex UltraScale FPGA 板级原理图审查检查表》([XTP344](#))
- 《UltraScale+ FPGA 和 Zynq UltraScale+ 器件板级原理图审查检查表》([XTP427](#))

使用 RTL 创建设计

完成器件 I/O 管脚分配、PCB 布局规划并决定 AMD Vivado™ Design Suite 的使用模型后，即可开始创建设计。设计创建包括：

- 规划设计的层级
- 识别要在设计中使用和定制的 IP 核
- 对于没有合适的 IP 可用的互连逻辑和功能，请创建定制 RTL
- 创建时序约束、功耗约束和物理约束
- 指定综合与实现阶段所使用的其他约束、属性及其他元件

创建设计时，主要的考虑要素包括：

- 实现所需的功能
- 按期望的频率运行
- 按期望的可靠程度运行
- 符合硅片资源和功耗预算要求

在此阶段做出的决策将影响最终产品。在这一阶段的错误决策会导致后续阶段问题层出不穷，进而造成整个设计周期中不断返工。在此过程中尽早花时间详细规划设计有助于达成设计目标并最大限度缩短实验室中的调试时间。

注释：如需了解有关 I/O 管脚分配进程、在 RTL 前的设计中由 PCB 设计师执行端口对齐以及使用时钟资源的更多信息，请参阅《Vivado Design Suite 用户指南：I/O 管脚分配和时钟规划》(UG899)。

定义理想的 RTL 设计层级

设计创建的第一步是决定如何对设计进行逻辑分区。定义层级时主要考虑的是如何对含特定功能的设计部分进行分区。这样便于特定设计人员单独设计 IP，以及隔离一段代码以供复用。

但仅根据功能来确定层级会导致对时序收敛、运行时间和调试的最优化方法考虑不周。在层级规划过程中考虑如下因素也有助于时序收敛。

在顶层附近添加 I/O 组件

尽可能在顶层附近添加 I/O 组件，以保障设计可读性。推断组件时，请提供要完成功能的描述。然后，综合工具会对 HDL 代码进行解释，以确定使用哪些硬件组件来执行该功能。可推断的组件为简单的单端 I/O (IBUF、OBUF、OBUFT 和 IOBUF) 以及 I/O 中的单倍数据速率寄存器。

使用工具推断 IOBUF 或 OBUFT 组件时，请确保使能逻辑和输入/输出逻辑都位于相同层级内。如果逻辑位于不同层级内并且在各层级之间存在 KEEP_HIERARCHY 或 DONT_TOUCH 属性，那么该工具将无法推断这些缓冲器。

如差分 I/O（IBUFDS 和 OBUFDS）以及双倍数据速率寄存器（IDDR、ODDR、ISERDES 和 OSERDES）等需例化的 I/O 组件也应在顶层附近进行例化。例化组件时，会将组件的实例添加到 HDL 文件中。例化可以让您完全控制组件的使用方式。因此，您将准确掌握逻辑的使用方式。

在顶层附近插入时钟元件

朝顶层方向插入时钟元件便于模块间的时钟共享。时钟共享可以减少时钟资源占用，从而提高资源使用率、提升最高时钟频率，并降低功耗。

除了在其中创建时钟的模块之外，时钟路径只能向下驱动进入模块。任何先自上而下而后又自下而上贯穿的路径都会在 VHDL 仿真中造成增量周期 (delta cycle) 问题，此类问题的调试既艰难又费时。

在逻辑边界处寄存数据路径

对层级边界输出进行寄存可将关键路径包含在单一模块或边界内。输入同样可以寄存在层级边界处。相比于遍布多个模块的路径，模块内部的时序路径始终更便于分析和修复。未在层级边界处寄存的任何路径都应采用层级重构来加以综合或者扁平化以便实现跨层级最优化。在逻辑边界处寄存数据路径有助于保留整个设计进程中的可追溯性（用于调试），因为这样可以最大限度避免跨层级最优化，并且逻辑不会跨模块迁移。

妥善考虑布局规划注意事项

布局规划可确保属于设计网表中特定区域的单元布局在器件的特定位置。您可使用手动布局规划来完成以下操作：

- 使用 SSI 器件时为特定 SLR 提供分区逻辑。
- 使用标准流程无法满足时序时，在设计上收敛时序。

如果有单元未限定在某一层级上，那么所有对象都必须单独包含在布局规划约束中。如果综合后这些对象的名称发生更改，必须更新约束。理想的布局规划应限定在某一层级上，因为这样只需一行约束即可。

布局规划并非必需。仅在必要时才需要执行布局规划。

如需了解有关布局规划的更多信息，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。



建议：虽然 Vivado 工具允许跨层级布局规划，但维护工作也会随之增加。请尽可能避免跨层级布局规划。

针对功能和时序调试最优化层级

如本章前文所述，把关键路径限定在同一层级边界内有助于时序的调试和满足时序要求。同样，出于功能调试（及修改）目的，相关信号应限定在同一层级上。这样即可使相关信号的探测和修改变得相对简单，因为当信号限定在单个层级中时，更易于跟踪通过综合对信号名称进行的最优化。

在模块层次应用属性

在模块层次应用属性可以使代码更整洁且更便于缩放。请勿在信号层次应用属性，而应改为在模块层次应用属性并将属性传输至当前层级中声明的所有信号。在模块层次应用属性还可支持您覆盖全局综合选项。



注意！不同于其他属性，DONT_TOUCH 属性不会从模块传输至该模块中的所有信号。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。

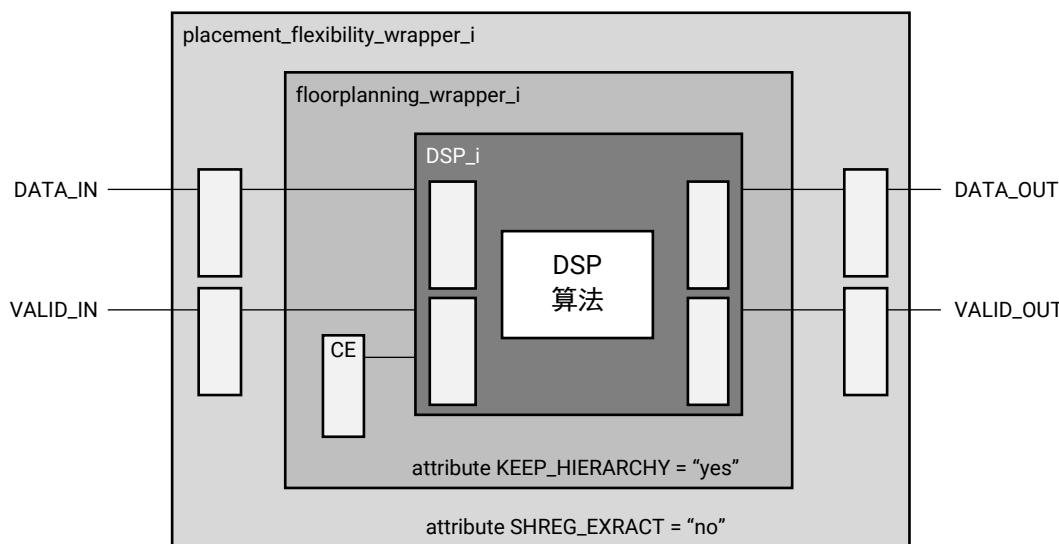
针对高级设计技巧实现层级最优化

高级设计技巧（如自下而上综合、Dynamic Function eXchange (DFX) 和非关联设计）需按层级进行规划。设计师必须根据使用的技巧选择合适的层级。本文不对这些技巧做详细介绍。

高速 DSP 设计的提前层级规划示例

以下示例并不适用于所有设计，但可演示层级的作用。DSP 设计一般允许对设计增加时延。这样可以在设计中添加寄存器，实现时钟频率更高的设计。此外，寄存器还可用于提升布局灵活性。这非常重要，因为时钟频率较高时，信号无法在 1 个时钟周期内遍历裸片。添加寄存器可使难以到达的区域变为可供使用。下图显示了有效的层级规划对于加速时序收敛的作用。

图 16：有效的层级规划示例



X13500-122019

本设计部分分 3 个层级：

- **DSP_i**

在 **DSP_i** 算法块中，输入和输出将同时寄存。由于器件拥有大量寄存器，因此，这是提升时序预算的首选方法。

- **floorplanning_wrapper_i**

在 **floorplanning_wrapper_i** 中有 1 个 **CE** 信号。CE 信号一般为重负载信号，会导致时序问题，应在布局规划时包含这些信号。通过创建布局规划封装，后续即可根据需要手动对此模块进行布局规划。

此外，在模块层次已添加 **KEEP_HIERARCHY** 以确保保留该层级用于布局规划，与任何其他全局综合选项无关。

- **placement_flexibility_wrapper_i**

在 **placement_flexibility_wrapper_i** 中，寄存 **DATA_IN**、**VALID_IN**、**DATA_OUT** 和 **VALID_OUT** 信号。由于并未计划在布局规划中包含这些信号，因此这些信号不包含在 **floorplanning_wrapper_i** 中。如果这些信号包含在布局规划中，将无法满足布局灵活性的要求。

此外，只要将 **DATA_IN + VALID_IN** 或 **DATA_OUT + VALID_OUT** 配对处理，稍后即可添加更多寄存器。如果添加更多寄存器，综合工具可能推断移位寄存器 LUT (SRL)，这将强制所有寄存器集中到 1 个组件中，这对于布局灵活性无益。为避免出现此情况，在模块层次已添加 **SHREG_EXTRACT** 并将其设置为 NO。

IP 的使用

使用预先确认的 IP 核能够大幅减少设计和确认工作量，从而加速产品上市进程。如需了解有关使用 IP 的更多信息，请参阅以下资源：

- 《Vivado Design Suite 用户指南：采用 IP 进行设计》(UG896)
- 《Vivado Design Suite 用户指南：采用 IP integrator 设计 IP 子系统》(UG994)
- [Vivado Design Suite QuickTake 视频：在 Vivado 中配置和管理可复用 IP](#)

规划 IP 要求

对任何新工程而言，规划 IP 要求都是最重要的环节之一：

- 请根据所需功能以及其他设计目标，评估 AMD 或其他第三方合作伙伴提供的 IP 选项。例如：
 - 与可用 IP 核相比，定制逻辑是否更好？
 - 以业界标准格式封装定制设计以便在多个工程中复用是否有意义？
- 考虑需要使用的接口，例如，存储器接口、网络接口和外设接口。



重要提示！ 为确保这些工具正确处理 IP 专用约束，请为工程添加 XCI 或 XCIX IP 源文件。处理 IP 时，请勿将 IP 生成的输出 DCP 文件作为工程源。

AMBA AXI

AMD 已对符合开放式 AMBA® 4 AXI4 互连协议的 IP 接口进行了标准化。这种标准化能够简化 AMD 和第三方提供商的 IP 的集成工作并最大程度提升系统性能。AMD 已与 Arm® 协作定义 AXI4、AXI4-Lite 和 AXI4-Stream 规格，以便将这些规格高效映射到其器件架构中。

AXI4 专为高性能、高时钟频率系统设计制定，适用于高速互连。AXI4-Lite 是 AXI4 的精简版，主要用于访问控制寄存器和状态寄存器。

AXI4-Stream 用于从主接口到从接口的单向数据串流传输。典型应用包括 DSP、视频和通信。

Vivado Design Suite IP 目录

IP 目录是集中包含 AMD 提供的 IP 的场所。在“IP catalog”中，您可查找嵌入式系统、DSP、通信、接口等的 IP 核。

在“IP catalog”中，您可浏览可用 IP 核，并查看任意 IP 的“Product Guide”（产品指南）、“Change Log”（更改日志）、“Product Web page”（产品网页）和“Answer Records”（答复记录）。

您可在“IP catalog”中通过 GUI 或 Tcl shell 来访问核并对其进行自定义。您还可使用 Tcl 脚本来自动执行 IP 核的自定义。

定制 IP

AMD 使用业界标准 IP-XACT 格式来交付 IP，并提供工具（IP 封装器 (IP packager)）来封装定制 IP。因此，您还可将自己的自定义 IP 添加到目录中，并创建 IP 存储库以便在团队内部或整个公司内部共享。来自第三方提供商的 IP 同样可添加到此目录中，前提是此类 IP 是在 IP 封装器中封装的，即使它已采用 IP-XACT 格式也是如此。

从 IP 目录选择 IP

所有 AMD 和第三方厂商的 IP 都按“通信和网络”、“视频和图像处理”、“汽车”以及“工业”等不同应用进行分类。您可根据该编目方法浏览 IP 目录，查看自己感兴趣领域的 IP 核。

IP 目录中的大部分 IP 都是免费提供的。但部分高价值 IP 要收取相应的成本并需要许可证。IP 目录会告知您，此 IP 是否需要购买以及许可证的状态。在从 IP 目录中选择 IP 的时候，应根据设计要求以及特定 IP 的功能考虑下列关键特性：

- 该 IP 所需的芯片资源（见相应的 IP 产品指南）
- 器件是否支持该 IP？是否考虑了速度等级（IP 选择往往决定速度等级选择）？如果支持，最大可实现的吞吐量以及最高频率 (F_{max}) 是多少？
- 设计中所需的与开发板上辅助芯片通信的外部接口标准：
 - 以太网、PCIe® 接口等业界标准接口。
 - 存储器接口：存储器接口的数量、尺寸和性能。
 - AMD 专有接口（如 Aurora）。
- 注释：**也可选择设计自己的定制接口。
- IP 支持的片上总线协议（应用接口）
- 与设计其余部分互动所需的片上总线协议。示例：
 - AXI4
 - AXI4-Lite
 - AXI4-Stream
- 如果涉及多重协议，可能必须使用 IP 目录中的基础架构 IP 来选择桥接 IP 核。例如：
 - AXI-AHB bridge
 - AXI-AXI Interconnect
 - AXI-PCIe bridge
 - AXI-PLB bridge

自定义 IP

可通过 GUI 或 Tcl 脚本来自定义 IP。

相关信息

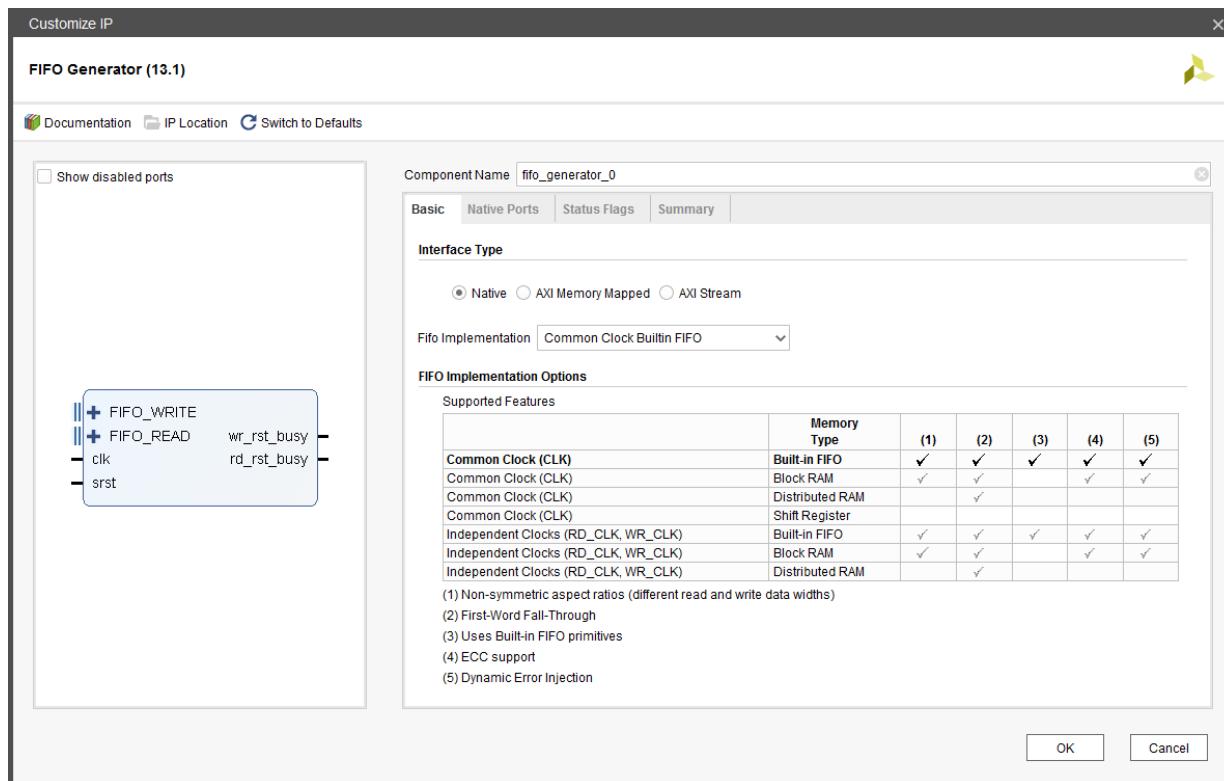
[使用自定义 GUI](#)

[使用 Tcl 脚本](#)

使用自定义 GUI

使用图形界面是查找、研究和自定义 IP 的最简单的途径。每个 IP 都有一组专为其自定义的选项卡或页面。相关配置选项均分组归于一处。定制窗口示例如下图所示。可创建以 XCI 文件表示的独特 IP 定制方案。随后即可以此为基础为任一 IP 创建多个输出文件。

图 17：IP 定制窗口



使用 Tcl 脚本

几乎每项 GUI 操作都会导致发出一条 Tcl 命令。IP 创建过程包括 Tcl 脚本中可执行且无需用户交互的所有定制选项的设置。

您需掌握配置选项的名称以及这些选项可设置为哪些值。通常首先通过 GUI 执行定制，然后在此基础上创建脚本。当您看到生成的 Tcl 脚本后，即可轻松按需修改脚本，例如，更改数据大小。

采用 Tcl 脚本创建 IP 可便于在诸如处理版本控制系统等情况下实现自动化。如需了解有关源代码管理和版本控制的信息，请访问此[链接](#)以参阅《Vivado Design Suite 用户指南：设计流程概述》(UG892) 中的相应内容。

IP 版本和版本控制

自定义 IP 后，该工具会创建 1 个 XCI 文件，其中包含所有选定的参数值。每个 Vivado IDE 版本都仅支持 1 个版本的 IP。AMD 建议您使用此最新 IP 版本。如果您使用较低版本的 IP，那么必须保存所有输出文件以供低版本使用。如需了解有关源代码管理和版本控制的信息，请访问此[链接](#)以参阅《Vivado Design Suite 用户指南：设计流程概述》(UG892) 中的相应内容。



重要提示！对于 7 系列器件中的存储器 IP，除 XCI 文件外还会创建 1 个 PRJ 文件。将版本控制与 7 系列存储器 IP 配合使用时，请将此 PRJ 文件与 XCI 文件保存在相同目录中。

RTL 编码准则

您可创建定制 RTL 来实现互连逻辑功能以及不含适合 IP 的功能。为了获得最佳结果，请遵循本节中的编码准则进行操作。如需获取其他指南，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。

使用 Vivado Design Suite HDL 模板

创建 RTL 或例化 AMD 原语时使用 Vivado Design Suite 语言模板。这些语言模板包含建议的编码结构，用于正确推断 AMD 器件架构。使用语言模板可以简化设计进程，并改进结果。要从 Vivado IDE 打开“Language Templates”（语言模板），请在 Flow Navigator 中选中“Language Templates”选项，然后选择所需模板。

控制信号和控制集

控制集是控制信号（置位/复位信号、时钟使能信号和时钟信号）的组合，用于驱动任意给定 SRL、LUTRAM 或寄存器。对于控制信号的任意独特组合，都会组成 1 个独立控制集。由于 7 系列 slice 中的寄存器全部共享公用控制信号，导致只能将含公用控制集的寄存器封装到相同 slice 中，因此该功能十分重要。例如，如果具有给定控制集的寄存器仅具有 1 个寄存器作为负载，那么其占据的 slice 中的另 7 个寄存器将变为不可用。

如果设计所含独立控制集过多，可能导致资源浪费过多并且布局选项减少，从而导致功耗上升且可实现的时钟频率降低。设计所含控制集越少，则布局选项更多且灵活性更高，并且通常可以得到更好的结果。

在 AMD UltraScale™ 器件中，CLB 中的控制集映射更为灵活。未驱动的复位不包含在控制集中，因为在 slice 内局部生成锁定。但是，最好对独立控制集数量加以限制从而最大限度提升逻辑组布局的灵活性。

复位

复位是需要在设计中考虑并加以限制的较为常见且重要的控制信号之一。复位会对设计的最高时钟频率、面积和功耗产生显著影响。

经推断的同步代码可能产生如下资源：

- LUT
- 寄存器
- SRL
- 块存储器或 LUT 存储器
- DSP48 寄存器

复位的选择和使用会影响这些组件的选择，导致针对给定设计选择的资源欠佳。任何阵列上复位布局错误都可能导致截然不同的结果，比如推断 1 个块 RAM 或推断数千个寄存器。

多路复用器的输入输出中所描述的异步复位可能导致将寄存器放置到 slice 中而不是放置到 DSP 块中。在此类情况下，将使用额外逻辑资源，从而对功耗和设计性能产生负面影响。

复位的使用时机和位置

AMD 器件具有专用的全局置位/复位信号 (GSR)。在器件配置结束时，此信号会设置硬件中所有时序单元的初始值。

如果未指定初始状态，将为时序原语分配默认值。在大多数情况下，默认值为 0。FDSE 和 FDPE 原语是例外，其默认为逻辑 1。每个寄存器在配置结束时都将处于已知状态。因此无需编写仅用于在上电时初始化器件的全局复位代码。

AMD 强烈建议您谨慎判断何时设计需要复位以及何时不需要复位。大多数情况下，在控制路径逻辑上可能需要复位以确保正常运行。然而在数据路径逻辑上通常不需要复位。复位的使用限制如下：

- 限制复位信号线的总体扇出。
- 减少复位布线所需的互连数量。
- 简化复位路径的时序。
- 在大多数情况下，这样即可整体改善时钟频率、面积和功耗。



建议：评估每个同步块，尝试判断是否需要复位以确保正常运行。默认情况下，除非确定确实有需要，否则请勿编写复位代码。

功能仿真可轻松判断是否需要复位。

对于不含复位编码的逻辑，可以灵活选择用于映射逻辑的器件资源。

随后，综合工具即可选择最适合代码的资源，并通过考量如下因素来尽可能实现最佳结果：

- 请求的功能
- 时钟周期要求
- 可用器件资源
- 功耗

同步复位对比异步复位

如需复位，AMD 建议使用同步复位。同步复位相比于异步复位具有如下优势：

- 同步复位可以直接映射至器件架构中的更多资源元件。
- 异步复位会影响通用逻辑结构的最大时钟频率。由于所有 AMD 器件的通用寄存器均可将置位/复位编程为异步或同步，可能看似使用异步复位不会受到任何惩罚。使用全局异步复位并不会增加控制集。但由于需要将此复位信号布局到所有寄存器元件，因此会增加布线复杂性。
- 复位断言有效期间，异步复位导致块 RAM、LUTRAM、以及 SRL 的存储器内容损坏的可能性更高。对于含异步复位（用于驱动块 RAM、LUTRAM 和 SRL 的输入管脚）的寄存器尤其如此。
- 需要更高密度或者微调布局时，同步复位会为控制集重新映射提供更多的灵活性。如果在布局更优化的 slice 中发现不兼容的复位，那么可将同步复位重新映射到寄存器的数据路径。这样即可根据需要减少布线资源使用率并增加布局密度，从而实现正确的适配并改善可实现的时钟频率。
- DSP48 和块 RAM 等部分资源仅包含同步复位以供块内的寄存器元件使用。在与这些元件关联的寄存器元件上使用异步复位时，可能无法在不影响功能的前提下直接将这些寄存器推断到这些块中。

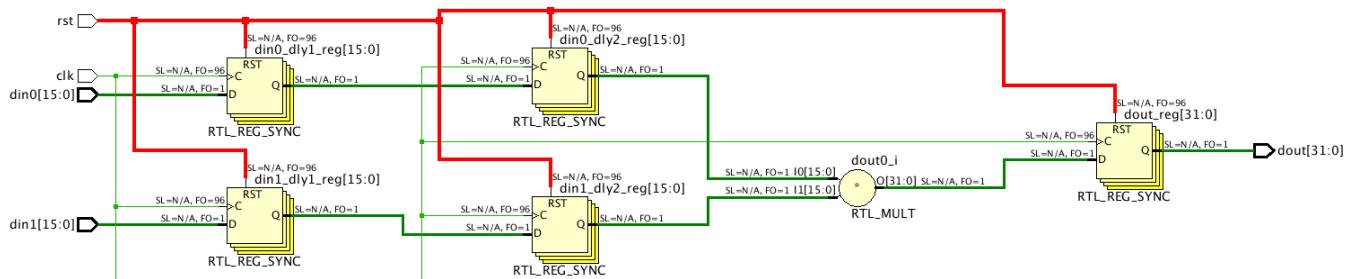
以下另提供了其他注意事项：

- 对于同步复位的复位小毛刺，时钟会充当滤波器的角色。但如果这些毛刺出现在有效时钟沿附近，那么触发器可能会变为亚稳态。
- 同步复位可能需要扩大脉冲宽度，以确保复位信号脉冲宽度足够容纳时钟沿有效期间存在的复位。
- 使用异步复位时，请务必对异步复位断言无效操作进行同步。虽然在复位断言有效期间可以忽略时钟与复位之间的相对时序，但复位释放必须同步到时钟。不执行复位释放时钟沿同步可能导致亚稳态。复位释放期间，与寄存器的时钟管脚相关的复位管脚必须满足建立时间和保持时间的时序条件。异步复位的建立时间和保持时间条件违例（例如，复位的恢复和移除时序）可能导致触发器变为亚稳态，从而因切换至未知状态而导致设计失败。请注意，此状况与触发器数据管脚的建立时间和保持时间条件违例相似。

复位编码示例：含同步复位的乘法器

为了充分利用现有 DSP 原语功能，以下示例显示了含同步复位的乘法器。

图 18：含流水线寄存器的倍频器（同步复位）



在此电路中，DSP48 原语的推断方式为，将所有流水线寄存器都封装在 DSP 原语 (AREG/BREG=1, MREG=1 和 PREG=1) 内。

下图显示了使用同步复位的乘法器流水线寄存器的编码示例。

图 19：同步复位编码示例

```

        always @ (posedge clk) begin
            if (rst) begin
                din0_dly1 <= 16'h0;
                din0_dly2 <= 16'h0;
                din1_dly1 <= 16'h0;
                din1_dly2 <= 16'h0;
                dout      <= 32'h0;
            end else begin
                din0_dly1 <= din0;
                din0_dly2 <= din0_dly1;
                din1_dly1 <= din1;
                din1_dly2 <= din1_dly1;
                dout      <= din0_dly2 * din1_dly2;
            end
        end
    
```

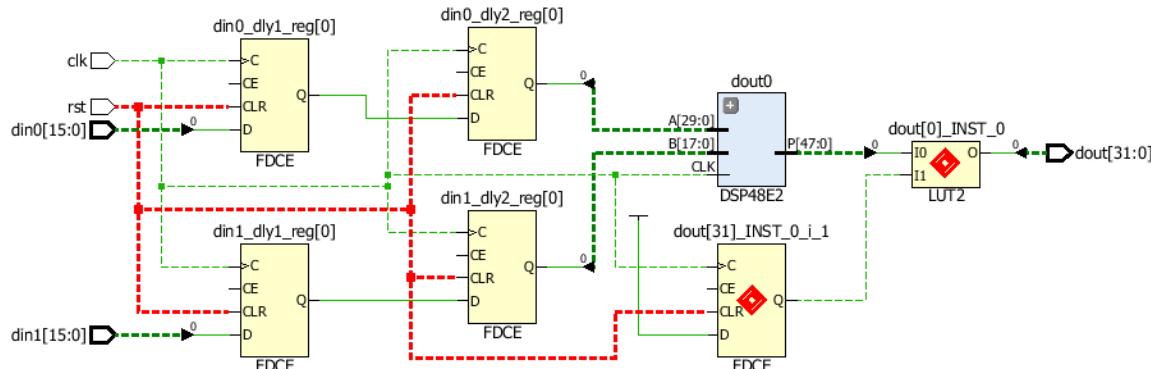
此编码示例具有以下示例：

- 最优资源使用率
- 更好的性能和更低的功耗
- 端点数量较少

复位编码示例：含异步复位的乘法器

以下示例说明了使用具有同步复位的寄存器作为面向专用 DSP 资源的逻辑的重要性。下图展示了使用具有异步复位的流水线寄存器并基于 DSP48 的 16x16 位乘法器。综合必须使用常规互连结构寄存器（针对输入阶段），以及 1 个外部寄存器和 32 个 LUT2（红色标记）来对 DSP 输出上的异步复位进行仿真（DSP48 P 寄存器已启用，但未连接到复位）。这需要额外耗用 65 个寄存器和 32 个 LUT，并且 DSP48 生成的配置为：AREG/BREG =0, MREG=0, PREG=1。

图 20：含使用异步复位的流水线寄存器的乘法器



如下图所示，只需更改复位定义，使乘法器流水线寄存器使用同步复位，综合即可利用 DSP48 内部寄存器：AREG/BREG=1，MREG=1，PREG=1。

图 21：在乘法器上将异步复位更改为同步复位

```

always @ (posedge clk or posedge rst) begin
    if (rst) begin
        din0_dly1 <= 16'h0;
        din0_dly2 <= 16'h0;
        din1_dly1 <= 16'h0;
        din1_dly2 <= 16'h0;
        dout      <= 32'h0;
    end else begin
        din0_dly1 <= din0;
        din0_dly2 <= din0_dly1;
        din1_dly1 <= din1;
        din1_dly2 <= din1_dly1;
        dout      <= din0_dly2 * din1_dly2;
    end
end

```

→

```

always @ (posedge clk) begin
    if (rst) begin
        din0_dly1 <= 16'h0;
        din0_dly2 <= 16'h0;
        din1_dly1 <= 16'h0;
        din1_dly2 <= 16'h0;
        dout      <= 32'h0;
    end else begin
        din0_dly1 <= din0;
        din0_dly2 <= din0_dly1;
        din1_dly1 <= din1;
        din1_dly2 <= din1_dly1;
        dout      <= din0_dly2 * din1_dly2;
    end
end

```

由于节省了互连结构资源并利用所有 DSP48 内部寄存器，设计性能和功耗效率达到最佳。

尝试在 HDL 代码中消除复位时出现问题

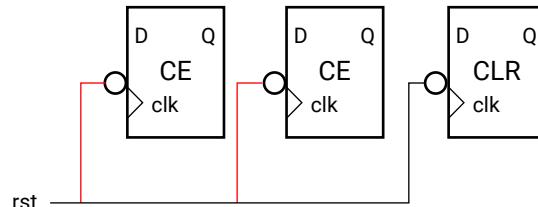
当最优化代码以消除复位时，注释掉复位声明中的条件无法生成期望的结构，反而导致出现问题。例如，下图显示了 3 个分别使用异步复位的流水线阶段。如果尝试通过注释掉含复位条件的代码来消除其中 2 个流水线阶段的复位条件，则将启用异步复位（rst 的逻辑取反）。

图 22：注释掉含复位条件的代码

```

always@(posedge clk or posedge rst)
begin
    if(rst)
    begin
        //din_dly1 <= 16'b0;
        //din_dly2 <= 16'b0;
        dout      <= 16'b0;
    end
    else
    begin
        din_dly1 <= din;
        din_dly2 <= din_dly1;
        dout      <= din_dly2;
    end
end

```



X17086-052016

移除复位的最佳途径是创建独立的顺序逻辑过程，其中一个过程用于复位条件，另一个过程用于非复位条件，如下图所示。

图 23：为含复位和无复位的寄存器创建独立的过程声明

```

always@(posedge clk)
begin
    din_dly1 <= din;
    din_dly2 <= din_dly1;
end

always@(posedge clk or posedge rst)
begin
    if(rst)
        dout <= 16'd0;
    else
        dout <= din_dly2;
end

```



提示：使用复位时，请确保过程语句中的所有寄存器均已复位。

时钟使能

如果正确使用，时钟使能可显著降低设计功耗，同时对面积或最高时钟频率的影响极小。但错误使用时钟使能时，可能会导致：

- 资源使用率增加
- 布局密度降低
- 功耗上升
- 可实现的时钟频率降低

大多数情况下，低扇出时钟使能是造成控制集数量过高的主要因素。

创建时钟使能

在同步块中编写的条件语句不完整时，就会创建时钟使能。通过推断时钟使能即可在先前条件未得到满足时，保留最后一个值。需要使用该功能时，采用此方式进行编码即为有效。在某些情况下，虽然先前条件值未得到满足，但并不影响输出。在此情况下，AMD 建议采用定义的常量（即为信号赋值 1 或 0）来关闭该条件（即，使用 `else` 子句）。

在大多数实现方案中，这不会导致额外增加逻辑，同时可避免使用时钟使能。但对于大型总线而言，推断时钟使能时，如果其中保留的值有助于降低功耗，则属例外情况，不适用此规则。此规则的基本前提是推断少量寄存器数时，由于时钟使能会增加控制集的数量，因此会产生不利影响。但是对较大型的群组而言，其利大于弊，所以建议使用。

复位和时钟使能优先顺序

在 AMD 器件中，所有寄存器构建时均将置位/复位设置为优先于时钟使能，不论所述对象为异步或同步置位/复位都是如此。为了取得最佳结果，AMD 建议您始终在同步时钟内通过 `if/else` 结构进行编码，以将置位/复位置于时钟使能（如果需要）之前。通过编码将时钟使能置于优先位置会导致强制复位进入数据路径，造成逻辑额外增加。

相关信息

时钟设置准则

使用综合属性控制使能/复位提取

您可以通过根据需要应用 `DIRECT_RESET/DIRECT_ENABLE/EXTRACT_RESET/EXTRACT_ENABLE` 属性来强制控制集映射，以处理给定结构的控制集的映射。

当设计包含同步复位/使能时，如果负载等于或高于 `-control_set_opt_threshold` 综合开关所设置的阈值，综合会创建通过 `CE/R/S` 管脚映射的逻辑椎，或者如果负载低于该阈值，那么综合会创建通过 `D` 管脚映射的逻辑椎。默认阈值如下所示：

- 7 系列器件：4
- UltraScale 器件：2

使用 `DIRECT_ENABLE` 和 `DIRECT_RESET`

要使用控制集映射，可向已连接到使能信号/复位信号的信号线应用属性，这将强制综合使用 `CE/R` 管脚。

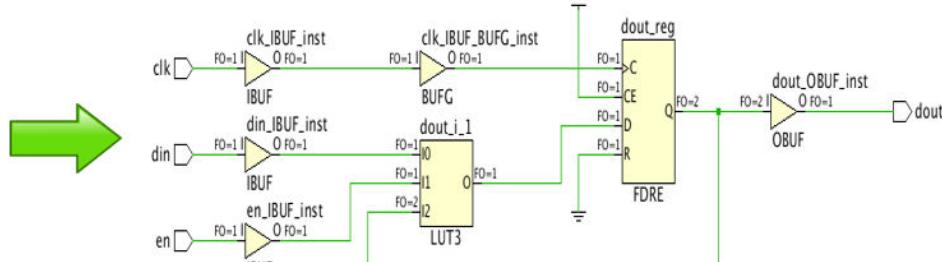
在下图中，使能信号 (`en`) 只能连接到 1 个触发器。因此，综合引擎已将 `en` 信号连接到逻辑的 `FDRE/D` 管脚椎。请注意，`CE` 管脚已绑定到逻辑 1。

图 24：使用数据路径逻辑完成时钟使能实现

```
module test
(
    input clk,
    input en,
    input din,
    output reg dout
);

always@(posedge clk)
begin
    if(en)
        begin
            dout <= din;
        end
    end
end

endmodule
```



要覆盖此默认行为，可使用 `DIRECT_ENABLE` 属性。例如，下图显示了如何通过将 `DIRECT_ENABLE` 属性添加到端口 / 信号来将使能信号 (`en`) 连接到寄存器的 `CE` 管脚。

图25：使用direct_enable完成专用时钟使能实现

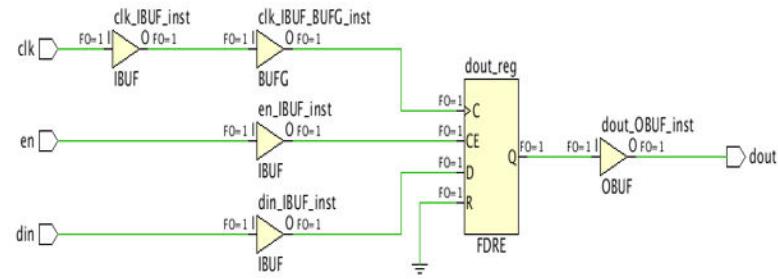
```

module test
(
    input clk,
    (* direct_enable = "true" *) input en,
    input din,
    output reg dout
);

always@(posedge clk)
begin
    if(en)
        begin
            dout <= din;
        end
end

endmodule

```



下图显示了RTL代码，其中global_rst或int_rst可将寄存器复位。默认情况下，两者都映射到逻辑的复位管脚椎。

图26：通过数据路径逻辑映射的多个复位条件

```

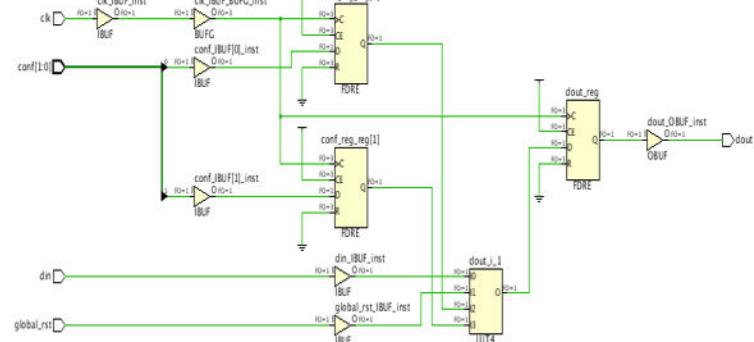
module test (
    input clk,
    input global_rst,
    input [1:0] conf,
    input din,
    output reg dout
);

reg [1:0] conf_reg;
assign int_rst = &conf_reg;

always@(posedge clk)
begin
    conf_reg <= conf;
    if(global_rst || int_rst)
        dout <= 1'b0;
    else
        dout <= din;
end

endmodule

```



可使用DIRECT_RESET属性来指定要连接到寄存器复位管脚的复位信号。例如，下图显示了如何使用DIRECT_RESET属性来仅将global_rst信号连接到寄存器FDRE/R管脚，并将int_rst信号连接到逻辑的FDRE/D椎。

图27：使用DIRECT_RESET属性的专用复位管脚的用法

```

module test (
    input clk,
    (* direct_reset = "true" *) input global_rst,
    input [1:0] conf,
    input din,
    output reg dout
);

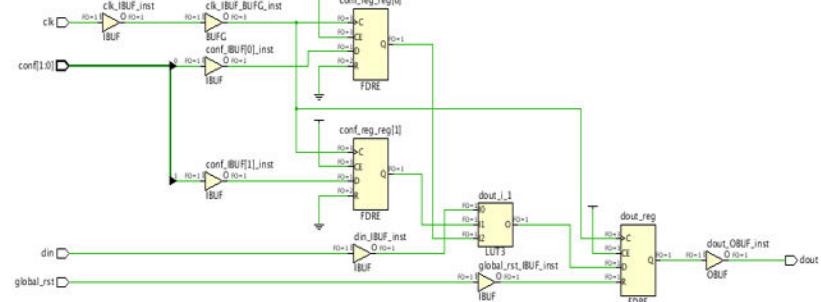
reg [1:0] conf_reg;

assign int_rst = &conf_reg;

always@(posedge clk)
begin
    conf_reg <= conf;
    if(global_rst || int_rst)
        dout <= 1'b0;
    else
        dout <= din;
end

endmodule

```



将逻辑从控制管脚推送到数据管脚

分析关键路径期间，您可能会发现有多条路径止于控制管脚。您必须对这些路径进行分析，以判定是否能够将逻辑推送到数据路径中，同时避免发生惩罚（例如，额外的逻辑层次）。相同逻辑层次的前提下，相比于指向CE/R/S管脚的路径，指向D管脚的路径的延迟较小，因为从最后一个LUT的输出到FF的D输入之间存在直接连接。以下编码示例显示了如何将逻辑从控制管脚推送到寄存器的数据管脚。

在以下示例中，`dout_reg[0]`的使能管脚的逻辑层数次为2，而数据管脚的逻辑层数次则为0。在此情况下，您可通过将使能逻辑移至D管脚来改进时序，方法是在RTL文件中的`dout`寄存器定义上将`EXTRACT_ENABLE`属性设置为“no”。

图28：止于寄存器控制管脚（使能）的关键路径

```

module test
(
    input clk,
    input [9:0] en,
    input [7:0] din,
    output reg [7:0] dout
);

wire en_tmp;
reg [7:0] din_reg;
reg [9:0] en_reg;

assign en_tmp = &en_reg;

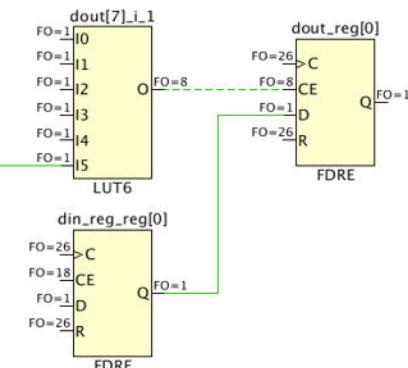
always@(posedge clk)
begin
    en_reg <= en;
    din_reg <= din;
    if(en_tmp)
        dout <= din_reg;
end

endmodule

```



Critical Path



以下示例显示了如何拆分组合逻辑和顺序逻辑以及如何将完整逻辑映射到数据路径中。这将把逻辑推送到仍含 2 个逻辑层次的 D 管脚中。

您可通过将 EXTRACT_ENABLE 属性设置为“no”来实现相同的结构。如需了解有关 EXTRACT_ENABLE 属性的更多信息，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。

图 29：止于寄存器的数据管脚（禁用使能提取）的关键路径

```
module test
(
    input clk,
    input [9:0] en,
    input [7:0] din,
    output reg [7:0] dout
);

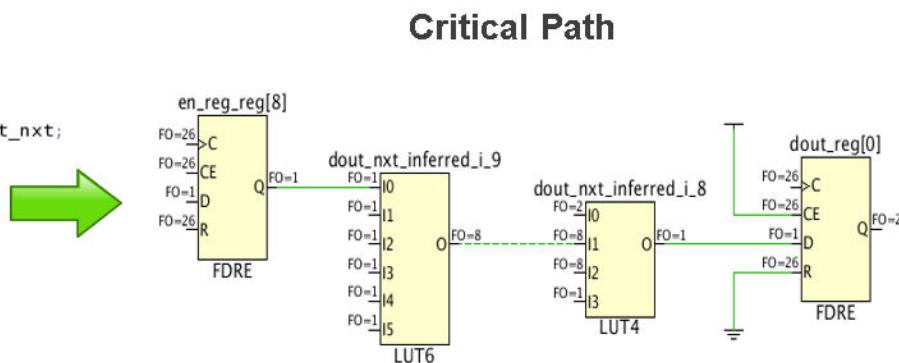
wire en_tmp;
reg [7:0] din_reg;
reg [9:0] en_reg;
(* KEEP = "true" *) reg [7:0] dout_nxt;

assign en_tmp = &en_reg;

always@*
begin
    dout_nxt = dout;
    if(en_tmp)
        dout_nxt = din_reg;
end

always@(posedge clk)
begin
    en_reg <= en;
    din_reg <= din;
    dout <= dout_nxt;
end

endmodule
```



信号控制技巧

- 检查是否真正需要全局复位。
- 避免异步控制信号。
- 保持时钟、使能和复位信号极性一致。
- 勿将置位和复位编码到同一寄存器元件中。
- 如果确实需要异步复位，请务必对其断言无效执行同步。

掌握推断的结果

代码最终必须映射到器件中存在的资源上。请尽力理清所处理的关键架构中的关键算术、存储和逻辑元件。这样在对设计功能进行编码时，即可预测代码将映射到的硬件资源。理解此映射方式将便于您尽早洞悉所有潜在的问题。

以下示例演示了了解硬件资源和映射方式给某些设计决策所带来的帮助：

- 对于超出 4 位的加法、减法和加减法，通常会使用进位链，并且每 2 位加法使用 1 个 LUT（即，8 位加 8 位的加法器使用 8 个 LUT 和关联的进位链）。对于三进制加法或者如果将任一加法器的结果与另一个值相加但中间不使用寄存器，那么每 3 位使用 1 个 LUT（即，8 位加 8 位加 8 位的加法同样使用 8 个 LUT 和关联的进位链）。

如果所需加法不止一次，那么最好在每两级加法后指定寄存器，这样可允许生成三进制实现，从而使器件使用率减半。

- 一般乘法针对的是 DSP 块。宽度小于 18x25（在 UltraScale 器件中为 18x27）的有符号位将映射到单个 DSP 块。乘积更大的乘法可能会映射到多个 DSP 块。DSP 块内部具有流水打拍资源。
针对推断到 DSP 块内的逻辑正确执行流水打拍操作可显著提高性能并降低功耗。描述乘法时，围绕乘法进行三级流水打拍操作可生成最佳的建立时间、时钟输出 (clock-to-out) 和功耗特性。流水打拍操作层级过浅（一级或无）可能导致时序问题并导致这些块的功耗增加，而 DSP 内部的流水打拍寄存器却被闲置。
- 深度不超过 16 位的 2 个 SRL 可映射到单个 LUT，而高达 32 位的单个 SRL 同样可映射到单个 LUT。
- 对于可生成标准 MUX 组件的条件代码：
 - 4 选 1 MUX 可实现 1 个 LUT，从而生成 1 个逻辑层。
 - 8 选 1 MUX 可实现 2 个 LUT 和 1 个 MUXF7 组件，同样可以有效生成 1 个逻辑 (LUT) 层。
 - 16 选 1 MUX 可实现 4 个 LUT 以及 MUXF7 和 MUXF8 资源组合，同样可以有效生成 1 个逻辑 (LUT) 层。

在同一个 CLB/slice 结构中组合 LUT、MUXF7 和 MUXF8 可生成极小的组合延迟。因此，这些组合可视为等同于 1 个逻辑层。充分理解此代码可有助于改善资源管理，从而有助于理解并控制数据路径的逻辑层次。

对于通用逻辑，请考虑给定寄存器的唯一输入的数量。根据此数量，可估算 LUT 数和逻辑级数。一般情况下，输入不超过 6 个时，始终生成 1 个逻辑层。理论上，2 个逻辑层可管理最多 36 个输入。但实际上，应假定 2 个逻辑层最多可管理约 20 个输入。总而言之，随着输入数量以及逻辑公式复杂性的提升，所需 LUT 和逻辑级数也会增加。



重要提示！ 在设计周期中尽早检查硬件资源的可用性以及有效利用这些资源的方式即可简化修改工作。此方法较之设计周期后期时序收敛期间再行检查的效果更好。

推断 RAM 和 ROM

可通过多种方式指定 RAM 和 ROM。每种方法都有各自的优缺点。

- 推断

优点：

- 便于移植
- 便于读取和理解
- 自我文档化
- 快速仿真

不足：

- 不能访问所有可用的 RAM 配置
- 结果可能并非最佳

由于推断结果一般较为理想，因此建议采用推断，除非给定用例不受支持，或者无法可实现的时钟频率、面积或功耗方面产生足够的结果。在此类情况下，请尝试其他方法。

推断 RAM 时，AMD 建议您使用 Vivado 工具中提供的 HDL 模板。如前文所述，使用异步复位会给 RAM 推断造成不利影响，应避免使用。

- 赛灵思可参数化宏 (XPM)

优点：

- 可在各 AMD 器件系列之间进行移植
- 快速仿真

- 支持非对称宽度
- 可预测的结果质量 (QoR)

不足：

- 仅支持 XPM 选项

XPM 是基于使用固定模板的推断构建的，此类模板无法修改。因此，其 QoR 有保证，并且可以支持标准推断所不具备的功能。当标准推断不支持所需功能时，AMD 建议您改为使用 XPM。

注释：使用 `compile_simlib` 编译仿真库时，会自动编译 XPM。如需了解更多信息，请参阅《Vivado Design Suite 用户指南：逻辑仿真》([UG900](#))。

- 直接例化 RAM 原语

优点：

- 对实现方案有最高控制权限
- 能访问块的各项功能

不足：

- 代码可移植性差
- 功能和用途冗长繁琐，难以理解

- IP 目录提供的 IP 核

优点：

- 在使用多个组件时一般能提供更优化的结果
- 易于指定和配置

不足：

- 代码可移植性差
- 需要管理核

相关信息

[使用 Vivado Design Suite HDL 模板](#)

实现 RAM 时的性能注意事项

为了有效推断存储元件，请考虑下列影响性能的因素：

- 使用专用块还是分布式 RAM

RAM 可在专用块 RAM 内实现，或者也可在使用分布式 RAM 的 LUT 内实现。此选择不仅影响资源选择，还会对可实现的时钟频率和功耗产生巨大影响。

一般情况下，RAM 所需深度即第一项标准。深度高达 64 位的存储器阵列通常在 LUTRAM 内实现，其中，深度不超过 32 位时，每个 LUT 映射 2 位，深度高达 64 位时每个 LUT 映射 1 位。根据可用资源和综合工具分配，更深的 RAM 也可在 LUTRAM 内实现。

深度超过 256 位的存储器阵列一般在块存储器中实现。AMD 器件能够按不同宽度和深度组合灵活映射此类结构。您应熟练掌握这些配置以便了解用于代码中较大型的存储器阵列声明的块 RAM 数量和结构。

- 使用输出流水线寄存器

对于以更高时钟频率工作的设计，输出寄存器是必需的，对于所有设计也建议使用输出寄存器来简化时序收敛。这样可改进块 RAM 的时钟输出时序。此外，第二个输出寄存器是很实用的，因为 slice 输出寄存器的时钟输出时序比块 RAM 寄存器更快。同时使用 2 个寄存器的总读取时延为 3。在推断这些寄存器时，这些寄存器与 RAM 阵列应处于相同层级。这样便于工具将块 RAM 输出寄存器合并到原语中。

- 使用输入流水线寄存器

当 RAM 阵列较大并跨多个原语映射时，可覆盖裸片上的大片面积。这可能导致地址线路和控制线路上出现时序收敛问题。请考虑在生成这些信号后，于 RAM 之前添加额外寄存器。为了进一步改进时序，请在流程后期使用 `phys_opt_design` 来复制此寄存器。输入上不含逻辑的寄存器将更便于复制。

阻碍块 RAM 输出寄存器推断的场景

AMD 建议在单一层级上对所有存储器和输出寄存器都进行推断，因为这是确保按期望方式进行推断的最简单的方法。在两种情况下会对块 RAM 输出寄存器进行推断。首先是在输出上存在额外的寄存器，其次是在存储器阵列中对读取地址寄存器重新进行时序约束。只有在使用单端口 RAM 时才会出现此类情况。如下图所示：

图 30：RAM 具有额外的读取寄存器用于进行块 RAM 输出寄存器推断

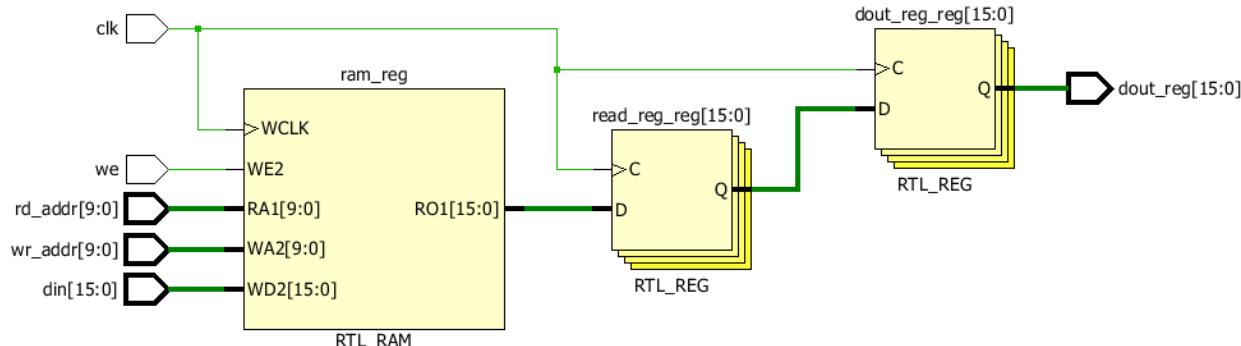
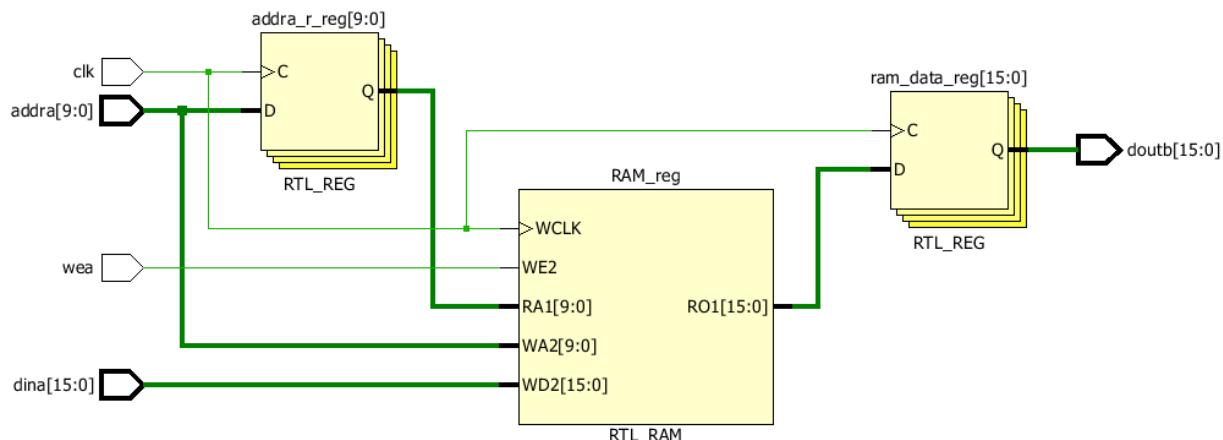


图 31：对地址寄存器重定时前的 RAM 视图

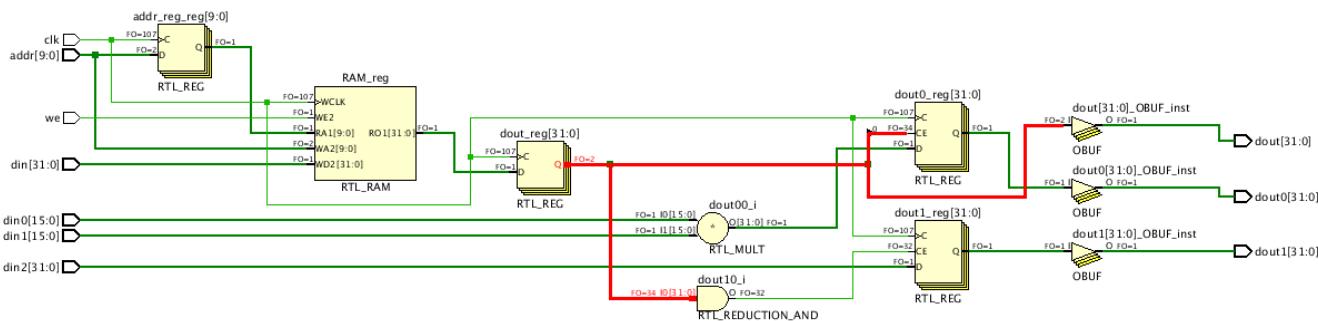


只要与这些示例有些许偏差，就会导致无法推断输出寄存器。

检查读取数据寄存器输出上的多重扇出

为使第二寄存器被 RAM 原语吸收，来自存储器阵列的数据输出位的扇出必须为 1。如下图中所示。

图 32：多重扇出阻止块 RAM 输出寄存器执行推断

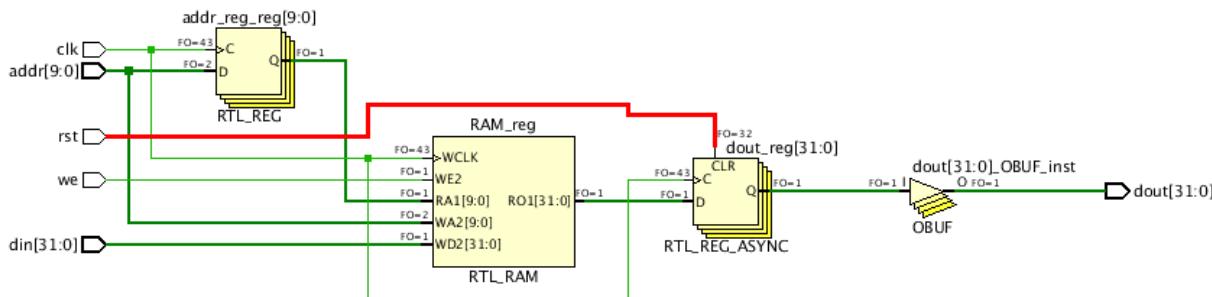


检查地址/读取数据寄存器上的复位信号

存储器阵列不应复位。仅限 RAM 输出才能容许复位。复位必须同步执行才能将输出寄存器推断到 RAM 原语中。异步复位将导致无法将寄存器推断到 RAM 原语中。此外，输出信号只能复位为 0。

下图突出显示了为确保正确推断 RAM 和输出寄存器而需要避免的行为示例。

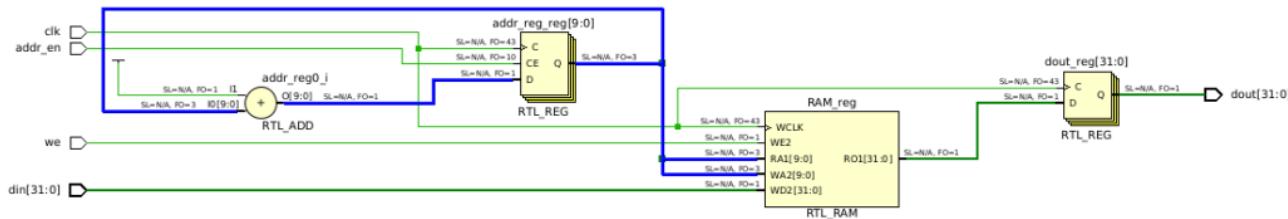
图 33：检查地址/读取数据寄存器上的复位



检查寄存器的反馈结构

请确保寄存器不含反馈逻辑，因为此逻辑会阻碍寄存器最优化。在以下示例中，地址寄存器会驱动 RAM 和加法器，即，无法将此寄存器打包到块 RAM 中。由此生成的电路是块 RAM，其中 dout 寄存器打包到 RAM 内以使 RAM 完全同步。但 RAM 不使用输出寄存器（DOA_REG 和 DOB_REG 将设为“0”），这样效率低下。

图 34：检查 RAM 块周围的寄存器上是否存在反馈



将存储器映射到 UltraRAM 块

UltraRAM 是 1 个 4Kx72 存储器块，具有 2 个使用同一时钟的端口。此原语仅在部分 AMD UltraScale+™ 器件中可用。在这些器件中，除块 RAM 资源外，还包含 UltraRAM。

可采用以下任一方法在设计中运用 UltraRAM：

- 依靠综合在 HDL 中的存储器声明上设置 `ram_style = "ultra"` 属性，以便推断 UltraRAM。
- 例化 AMD XPM_MEMORY 原语。
- 例化 UltraRAM UNISIM 原语。

以下代码示例显示了 XPM 存储器的例化，可在 HDL 语言模型中使用。突出显示的 MEMORY_PRIMITIVE 和 READ_LATENCY 参数均为关键参数，用于推断存储器作为 UltraRAM 用以实现最优资源映射效率。

- `MEMORY_PRIMITIVE = "ultra"` 指定存储器将推断为 UltraRAM。
- `READ_LATENCY` 用于定义存储器输出上存在的流水线寄存器数量。

较大的存储器将映射到 UltraRAM 矩阵，其中包含多个配置为行 x 列结构的 UltraRAM 单元。

可基于深度创建单列或多列矩阵。UltraRAM 列高度的当前默认阈值为 8，可通过 `CASCADE_HEIGHT` 属性来加以控制。

单列和多列 UltraRAM 矩阵的差异如下所述：

- 单列 UltraRAM 矩阵使用内置硬件级联，无互连结构逻辑。
- 多列 UltraRAM 矩阵的每个列中均使用内置硬件级联，并附加一些互连结构逻辑用于连接各列。可能需要增加流水打拍以保持可达成的时钟频率。这是通过增加读取时延来控制的。Vivado 工具会根据需要自动将这些寄存器封装到 UltraRAM 中。

图 35：在 RTL 代码中通过 XPM 指定 UltraRAM

```
xpm_memory_spram # (  
    // Common module parameters  
    .MEMORY_SIZE      (8*(4096*72)),           //positive integer  
    .MEMORY_PRIMITIVE ("ultra"),                //string; "auto", "distributed", "block" or "ultra";  
    .MEMORY_INIT_FILE ("none"),                 //string; "none" or "<filename>.mem"  
    .MEMORY_INIT_PARAM (''),                   //string;  
    .USE_MEM_INIT     (0),                      //integer; 0,1  
    .WAKEUP_TIME     ("disable_sleep"),          //string; "disable_sleep" or "use_sleep_pin"  
    .MESSAGE_CONTROL (0),                      //integer; 0,1  
  
    // Port A module parameters  
    .WRITE_DATA_WIDTH_A (72),                  //positive integer  
    .READ_DATA_WIDTH_A (72),                  //positive integer  
    .BYTE_WRITE_WIDTH_A (72),                //integer; 8, 9, or WRITE_DATA_WIDTH_A value  
    .ADDR_WIDTH_A     (16),                    //positive integer  
    .READ_RESET_VALUE_A ("0"),                //string  
    .READ_LATENCY_A   (9),                    //non-negative integer  
    .WRITE_MODE_A     ("read_first")           //string; "write_first", "read_first", "no_change"  
) xpm_memory_spram_inst (  
    // Common module ports  
    .sleep            (1'b0),  
  
    // Port A module ports  
    .clka             (clka),  
    .rsta             (rsta),  
    .ena              (ena),  
    .regcea           (regcea),  
    .wea              (wea),  
    .adra             (adra),  
    .dina             (dina),  
    .injectsbiterra (1'b0), //do not change  
    .injectdbiterra (1'b0), //do not change  
    .douta            (douta),  
    .sbiterra         (),           //do not change  
    .dbiterra         ()           //do not change  
);
```

上述示例使用 32 K x 72 存储器配置，此配置使用 8 个 UltraRAM。为了增大 UltraRAM 的最高时钟频率，应向级联链添加更多流水打拍寄存器。这是通过增加读取时延参数值来实现的。

如需了解有关 Vivado 综合中推断 UltraRAM 的更多信息，请访问此[链接](#)以参阅《Vivado Design Suite 用户指南：综合》(UG901) 中的相关内容。

通过编码实现最优化的 DSP 和算术推断

AMD 器件中的 DSP 块可以执行许多不同的功能，包括：

- 乘法
- 加法和减法
- 比较器
- 计数器
- 一般逻辑

DSP 块属于高度流水打拍的块，具有多个寄存器阶段，能在降低资源总体功耗的情况下实现高速运行。AMD 建议将准备映射到 DSP48 中的代码完全流水打拍，这样即可利用所有的流水线阶段。为了灵活运用这一额外资源，功能中应避免使用置位条件，这样才能正确映射到该资源。

AMD 器件中的 DSP48 slice 寄存器只包含复位功能，没有置位功能。因此除非必要，应避免围绕乘法器、加法器、计数器或其他可在 DSP48 slice 中实现的逻辑进行置位编码（在施加信号的情况下，逻辑值等于 1）。此外，由于该 DSP slice 只支持同步复位操作，应避免异步复位。如果代码导致置位或异步复位，则会导致面积、性能和功耗等方面的结果不理想。

许多 DSP 设计都非常适合采用 AMD 架构。要充分利用该架构，必须熟悉底层特性和功能，以便设计输入代码能够充分利用这些资源。

DSP48 块使用符号化的算术实现方案。AMD 建议在 HDL 源代码中使用有符号的值进行编码，以便与资源功能实现最佳匹配并实现总体上最有效的映射。如果在代码中使用无符号的总线值，综合工具也许仍然能够使用该资源，但由于需要进行无符号到有符号转换，该组件可能无法实现满位精度。

如果预计目标设计将包含大量加法器，AMD 建议对设计进行评估，以便更好地利用 DSP48 slice 的预加法器和后加法器。例如在使用 FIR 滤波器的情况下，可使用加法器级联来构建脉动型滤波器，而不必使用多重连续加法功能（加法器树）。如果采用对称滤波器，那么可以使用专用预加法器进行评估，以便进一步将该功能整合到数量更少的 LUT、触发器和 DSP slice 中（大多数情况下可减少一半的资源占用）。

如果需要使用加法树，那么 6 输入 LUT 架构可以使用与简单的 2 输入加法同样多的资源来高效地完成三值加法 ($A + B + C = D$)。这样有助于节约和保存进位逻辑资源。不过在大多数情况下无需使用此类技巧。

在了解这些功能的基础上，就可以提前进行适当的权衡取舍，并体现在 RTL 代码当中，从而从一开始就实现更加顺畅和更加高效的实现方案。在大多数情况下，AMD 建议推断 DSP 资源。

如需了解有关 DSP48 slice 的特性和功能的更多信息以及如何最有效地利用该资源来满足您的设计需求，请参阅《7 系列 DSP48E1 slice 用户指南》([UG479](#)) 和《UltraScale 架构 DSP slice 用户指南》([UG579](#))。

对移位寄存器和延迟线进行编码

一般而言，移位寄存器应具备以下部分或全部控制和数据信号特征：

- 时钟
- 串行输入
- 异步置位/复位
- 同步置位/复位
- 同步/异步并行负载
- 时钟使能
- 串行或并行输出

AMD 器件包含专用 SRL16 和 SRL32 资源（集成在 LUT 中）。这样无需使用触发器资源即可高效实现移位寄存器。但是这些元件仅支持左 (LEFT) 移位操作，且 I/O 信号数量有限：

- 时钟
- 时钟使能
- 串行数据输入
- 串行数据输出

此外，SRL 提供用于确定移位寄存器长度的地址输入（针对 SRL16 为 A3、A2、A1、A0 输入）。移位寄存器可采用固定的静态长度，也可以动态调节。在动态模式下，每次向地址管脚应用新地址时，LUT 访问延迟时间过后，就会在 Q 输出上显示新的比特位置值。

在 SRL 原语中不提供同步和异步置位/复位控制信号。但是，如果 RTL 代码包含复位，那么 AMD 综合工具就会围绕 SRL 推断其他逻辑以提供复位功能。

为了在使用 SRL 时得到更高的时钟频率，AMD 建议在专用分片 (slice) 寄存器中实现移位寄存器的最终阶段。slice 寄存器时钟输出 (clock-to-out) 时间比 SRL 更短。这样即可为源自移位寄存器逻辑的路径提供更大的时序裕量。除非因属性或跨层级边界最优化限制而导致例化此资源，或者已阻止综合工具推断此寄存器，否则综合工具会自动推断此寄存器。要推断其他寄存器，请将动态延迟的信号单独寄存在 RTL 内。

AMD 建议您使用 Vivado Design Suite HDL 模板中演示的 HDL 编码样式。

使用寄存器实现芯片内的布局灵活性时，请使用以下属性关闭 SRL 推断：

```
SHREG_EXTRACT = "no"
```

如需了解有关综合属性以及如何在 HDL 代码中指定这些属性的更多信息，请参阅《Vivado Design Suite 用户指南：综合》(UG901)。

所有已推断的寄存器、SRL 和存储器的初始化

GSR 信号线会将所有寄存器初始化为 HDL 代码中指定的初始值。如果未提供初始值，综合工具可自行判断将初始状态赋值为 0 或 1。Vivado 综合通常默认设置为 0，但少数例外除外，如独热 (one-hot) 状态机编码。

对于任何已推断的 SRL、存储器或其他同步元件，同样可为其定义初始状态，并在配置时将其初始状态烧录到关联的元件中。

AMD 强烈建议您对所有同步元件进行相应的初始化。寄存器初始化完全可由所有主要的器件综合工具进行推断。这样可消除单纯为初始化而添加复位的需求，并且使 RTL 代码与功能仿真中实现的设计更紧密匹配，因为所有同步元件配置后都会从器件中的已知值启动。

寄存器和锁存器初始状态 VHDL 编码示例 1：

```
signal reg1 : std_logic := '0'; -- specifying register1 to start as a zero
signal reg2 : std_logic := '1'; -- specifying register2 to start as a one
signal reg3 : std_logic_vector(3 downto 0) := "1011"; -- specifying INIT
value for
4-bit register
```

寄存器和锁存器初始状态 Verilog 编码示例 2：

```
reg register1 = 1, b0; // specifying register1 to start as a zero
reg register2 = 1, b1; // specifying register2 to start as a one
reg [3:0] register3 = 4, b1011; // specifying INIT value for 4-bit register
```

另外，在 Verilog 中还可使用 initial 语句：

```
reg [3:0] register3;
initial begin
    register3= 4, b1011;
end
```

判断例化或推断的时机

AMD 建议您为自己的设计添加 RTL 描述，并使用综合工具将代码映射到器件中的可用资源。除了增强代码的可移植性外，所有推断所得逻辑均可供综合工具查看，从而使其能够执行各项功能之间的最优化。最优化包括逻辑复制、重构与合并，以及通过重定时来平衡寄存器间的逻辑延迟。

综合工具最优化

默认情况下，完成器件库单元例化后，综合工具不会对其进行最优化。即便收到对器件库单元进行最优化的指令，综合工具一般也无法执行与 RTL 相同等级的最优化操作。因此，综合工具通常仅在出入这些单元的路径上执行最优化，而不会对穿过这些单元的路径执行最优化。

例如，如果对某个 SRL 进行例化，并且将其包含在长路径中，那么此路径可能成为瓶颈。此 SRL 相比于常规寄存器的时钟输出 (clock-to-out) 延迟更长。为了保留此 SRL 所达成的面积缩小结果，同时改善其时钟输出时序特性，将创建 1 个比实际期望的延迟少 1 个延迟的 SRL，并在常规触发器中实现最后一个阶段。

判断例化的时机

当综合工具映射无法满足时序、功耗或面积约束时，或者当无法推断器件内的特定功能时，可能需要例化。

借助例化，即可全权掌控综合工具。例如，为了提高时钟频率，可单独使用 LUT 来实现比较器，而不是采用综合工具通常所选择的 LUT 与进位链组合方法，后者通常适用于节省面积。

有时，例化可能是利用器件中可用的复杂资源的唯一途径。原因可能是：

- HDL 语言限制

例如，无法描述 VHDL 中的双倍数据速率 (DDR) 输出，因为它需要 2 个单独的进程驱动同一个信号。

- 硬件复杂性

相比于创建可综合的描述，例化 I/O SerDes 元件可能更简单。

- 综合工具推断限制

例如，综合工具当前无法根据 RTL 描述推断硬核 FIFO。因此，您必须将其例化。

如果您决定例化 AMD 原语，请参阅目标架构的相应《用户指南》和《库指南》，以便充分了解组件功能、配置和连接功能。

对于推断和例化，AMD 建议您使用来自 Vivado Design Suite 语言模板的例化和语言模板。

以下提供了一些实用技巧：

- 尽可能推断功能。
- 当综合的 RTL 代码不满足要求时，请先复查要求，然后再将代码替换为器件库组件例化。
- 编写公用 Verilog 和 VHDL 行为构造时或者需要例化所需原语时，请考虑使用 Vivado Design Suite 语言模板。

改善最高频率的编码样式

对于高性能设计而言，本章节中所讨论的编码方法可以减少潜在的时序危机。

关键路径上的高扇出

高扇出信号线在设计进程早期阶段更便于处理。目标时钟频率要求和路径的结构往往会导致扇出过高。您可以使用以下技巧来解决高扇出信号线的问题。



建议：在综合后使用 `report_high_fanout_nets` Tcl 命令识别高扇出信号线。在实现流程期间，监控这些信号线对设计时序收敛的影响。

在设计各部分中减少不必要的负载

对于高扇出控制信号，请评估是否设计的所有编码部分都需要该信号线。减少负载数量可以大幅度减少时序问题。

复制高扇出信号线驱动

寄存器复制可通过复制寄存器来减少给定信号的扇出，从而提升关键路径的速度。这便于实现工具更加灵活地对各类不同负载和相关逻辑进行布局布线。综合工具广泛采用了这种方法。

大多数综合工具使用扇出阈值限值来自动判定是否需要复制寄存器。降低此全局阈值即可自动复制高扇出信号线。但这样就无法控制需复制的寄存器范围以及这些寄存器的负载分组方式。此外，全局复制机制无法准确评估时序裕量，导致不必要的复制单元、逻辑占用率增加以及潜在功耗增加。

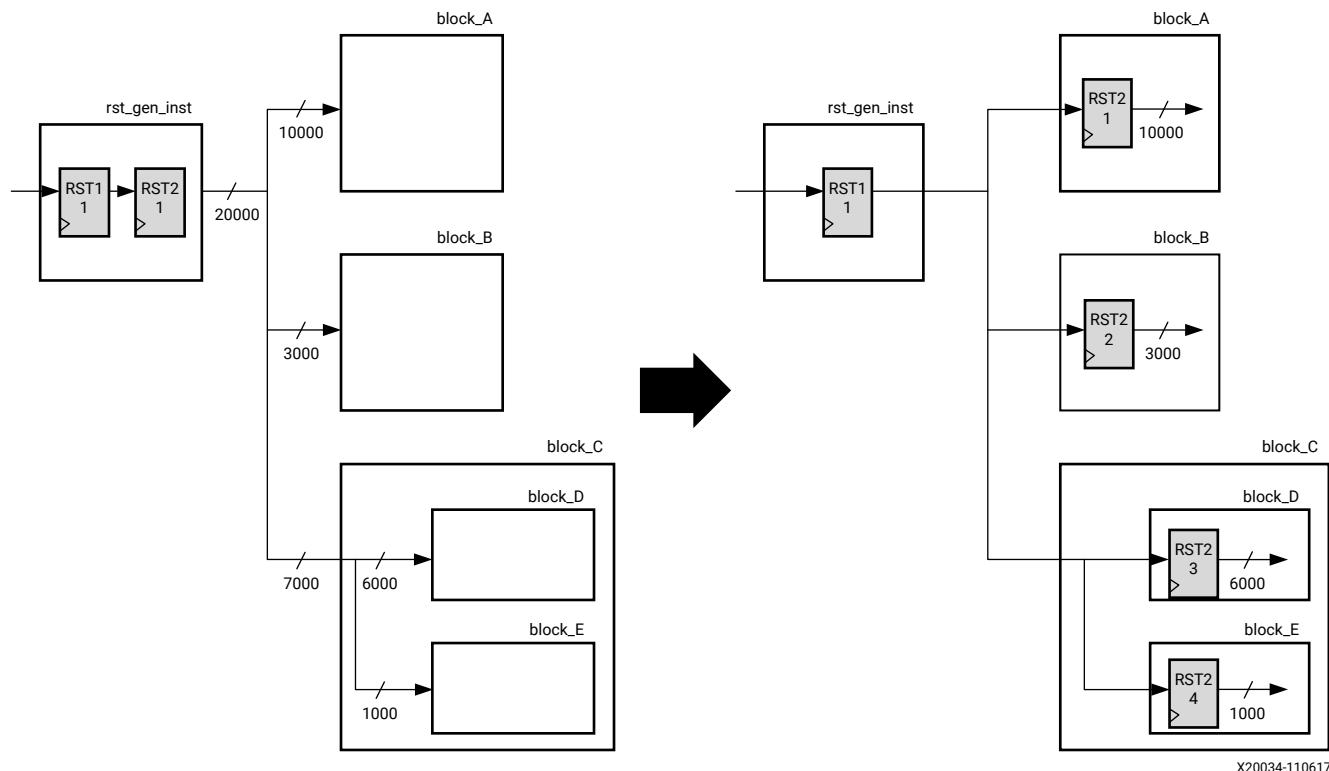
对于高频率设计，要减少扇出，最好对高扇出信号使用平衡树。可考虑根据设计层级手动复制寄存器，因为层级中包含的单元通常布局在一起。例如，在下图所示的平衡复位树中，在 RTL 中复制高扇出复位 FF RST2 以平衡不同模块之间的扇出。如果需要，物理综合可以基于布局信息执行进一步的复制以改进 WNS。



提示：要在综合中保留重复寄存器，请使用 `KEEP` 属性，而不是 `DONT_TOUCH`。在实现流程后期进行物理最优化期间，`DONT_TOUCH` 属性会阻止进行进一步最优化。

注释：如果复制的是 LUT1 而不是寄存器，表明应用的属性或约束错误。

图36：高扇出复位变换为平衡复位树



X20034-110617



建议：在全局高扇出信号上使用 MAX_FANOUT 属性会导致复制结果欠佳，与在综合中降低全局扇出限制时类似。因此，AMD 建议仅在支持中低扇出的局部信号上的层级内使用 MAX_FANOUT。

切勿复制用于同步跨时钟域的信号的寄存器。如果这些寄存器上存在 ASYNC_REG 属性，将导致工具无法复制这些寄存器。如果同步链扇出极高且复制必须满足时序要求，那么请在不含 ASYNC_REG 约束的同步链之后添加额外的寄存器。

下表提供了可能适用于您的设计的扇出数量相关指南。

表1：适用于中等性能的7系列器件的扇出指南

条件	扇出 > 5000	扇出 > 200	扇出 > 100
低频：1 到 125 MHz	同步逻辑之间几乎无逻辑级数，最高频率时逻辑级数小于 13	不适用	不适用
中频：125 到 250 MHz	如果设计不满足时序要求，可能需降低扇出和/或逻辑级数。	最高频率时逻辑级数小于 6。（驱动和负载类型会影响性能。）	不适用
高频 > 250 MHz	对大多数设计不建议使用。	通常速度越高所需逻辑级数越少。	需要先进的流水线方法、精心完成的逻辑复制、紧凑的功能、较少逻辑级数。（驱动和负载类型会影响性能。）



提示：如果时序报告表明高扇出信号正在限制设计性能，请考虑使用实现工具选项（例如，opt_design - hier_fanout_limit、place_design 和 phys_opt_design）来复制信号。



提示：复制寄存器时，请考虑针对寄存器使用命名约定（例如，`<original_name>_a`、`<original_name>_b` 等）来使其更便于了解复制的意图，并且更便于维护 RTL 代码。

流水打拍注意事项

另一种提升性能的方法是对具有多个逻辑层次的长数据路径进行重构并将其分布在多个时钟周期中。此方法可加速时钟周期并增加数据吞吐量，但代价是时延和流水线开销逻辑管理工作增加。

由于器件包含许多寄存器，因此通常额外的寄存器和开销逻辑不足为虑。但是，数据路径将跨多个周期，需特别留意设计其余部分，并考量路径时延增加的问题。

考虑 SSI 器件的流水打拍

为 SLR 边界交汇设计高性能寄存器间连接时，必须在 HDL 代码中描述相应的流水线，并在综合阶段对其进行控制。这样可避免在必须跨 SLR 边界的逻辑路径中发生移位寄存器 LUT (SRL) 推断和其他最优化行为。以此方式修改代码并适当使用 Pblock 即可定义发生 SLR 边界交汇的位置。

提前考量流水打拍

提前而非滞后考量流水线有助于改进时序收敛。在后期对某些路径添加流水线通常会导致在电路中产生传输时延差异。这可能导致看似微小的更改需要对部分代码进行大幅重新设计。

在设计中提前识别是否有机会添加流水线可以显著改进时序收敛、实现运行时间（因为时序问题变得更容易解决）和器件功耗（因为相关逻辑切换次数减少）。

检查推断的逻辑

对设计进行编码时，请注意正在推断的逻辑。请注意如下条件，以了解其他流水线设置注意事项：

- 带有大扇入的逻辑椎

例如，需要大量总线或多个组合信号来计算输出的代码。

- 布局受限、时钟输出缓慢或者具有较高的建立时间要求的块

例如，块 RAM 内不含输出寄存器或者未适当流水打拍的算术代码。

- 导致布线过长的强制布局

例如，如果管脚分配强制跨芯片布线，则可能需要流水打拍才能执行高速操作。

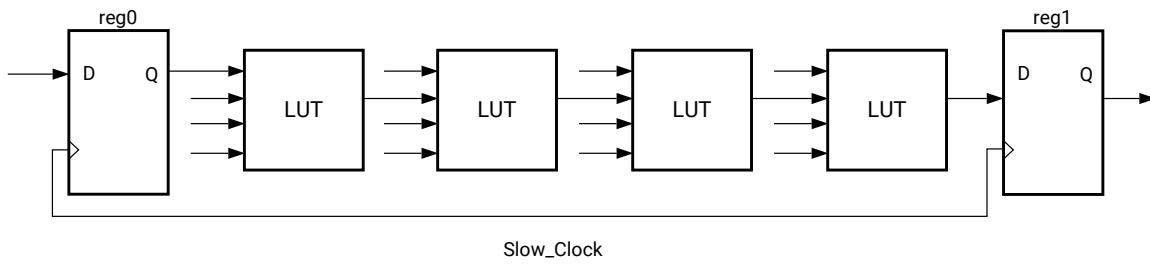
- 包含大量 XOR 函数的逻辑

大量 XOR 函数通常导致转换率高，由此造成大量动态功耗损耗。对这些函数进行流水打拍可降低转换率，从而对所述电路的功耗产生积极影响。

在下图中，时钟速度受下列因素限制：

- 源触发器的时钟输出时间
- 贯穿四个逻辑层次的逻辑延迟
- 四个函数发生器的相关布线
- 目标寄存器的建立时间

图37：流水打拍前

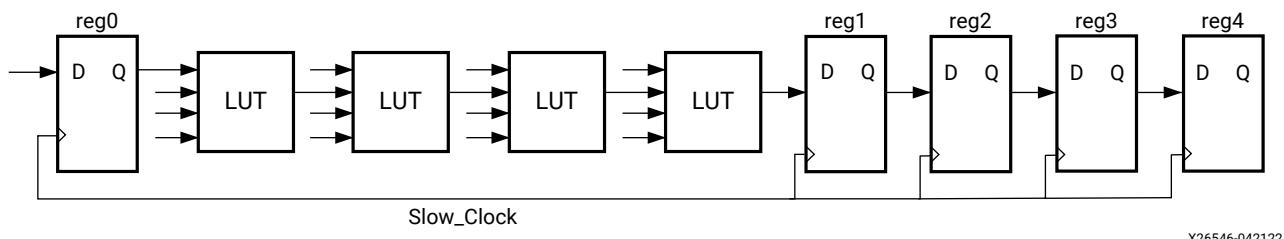


使用下列任一方法确保设计正确使用流水线寄存器：

- 在 RTL 代码中，在要重定时的逻辑之前或之后（最好在层级内）添加寄存器。
- 使用 Vivado 综合全局重定时或 BLOCK_SYNTH.RETIMING 选项，分析路径时序并移动寄存器以改善时序（如可能）。
- 或者为了进一步控制，可使用 `retiming_forward` 和 `retiming_backward` 综合属性。您可在特定寄存器上添加这些属性，强制工具穿过组合逻辑进行定时，忽略逻辑的时序得分。如需了解有关这些属性的更多信息，请参阅《Vivado Design Suite 用户指南：综合》(UG901)。

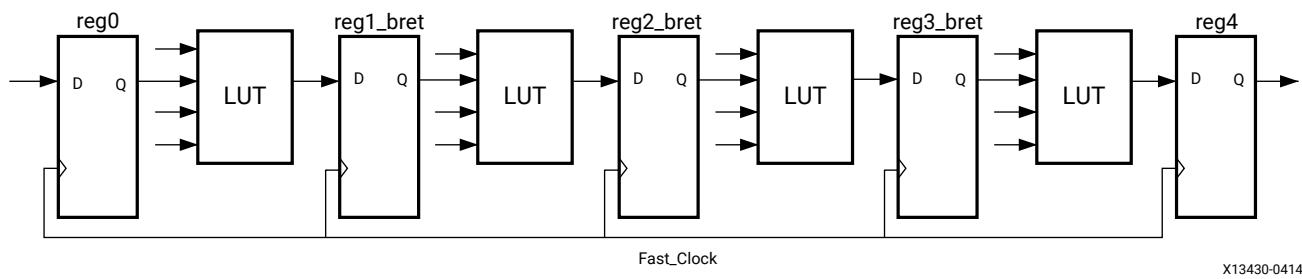
下图显示了添加额外寄存器之后的流水打拍。

图38：添加额外寄存器后的流水打拍



下图是“流水打拍前的图示”中所示的数据路径示例。由于触发器与函数发生器包含在相同 slice 内，时钟速度受到下列要素的限制：源触发器的时钟输出时间、贯穿 1 个逻辑层次的逻辑延迟、布线延迟，以及目标寄存器的建立时间。在此示例中，完成流水打拍和重定时之后，系统时钟运行速度比原始设计中更快。

图39：重定时后流水打拍



以下代码示例演示了如何使用重定时属性来强制执行“重定时后流水打拍”图例中所示的特定重定时。

```
( * retiming_backward = 3 * ) reg reg1;
( * retiming_backward = 2 * ) reg reg2;
( * retiming_backward = 1 * ) reg reg3;
```

确定是否需要流水打拍

常用的流水线方法可以识别大型组合逻辑路径、将其细分为较小的路径，并在这些路径之间引入寄存器阶段，从而实现每个流水线阶段的理想平衡状态。

要确定设计是否需要采用流水线方法，请确定时序的频率和分布在每个时钟组中的逻辑数量。您可以使用含 `-logic_level_distribution` 选项的 `report_design_analysis` Tcl 命令来确定每个时钟组的逻辑级分布。



提示：设计分析报告还可突出显示不含任何逻辑层的路径数，您可以使用这些路径确定代码中需要修改的位置。

平衡时延

要通过添加流水线阶段来平衡时延，请将此阶段添加到控制路径中，而不是数据路径中。数据路径包含更宽的总线，这可增加所使用的触发器和寄存器资源的数量。

例如，如果有一条 128 位数据路径、2 个寄存器阶段并且需要 5 个时延周期，插入 3 个寄存器阶段会导致额外产生 $3 \times 128 = 384$ 个触发器。或者，您可以使用寄存器来控制启用数据路径的逻辑。使用 5 个阶段的单比特寄存器可分别控制数据路径触发器的使能信号和多周期路径时序例外。

注释：此示例仅适用于某些设计。例如，如果在中间数据路径触发器中存在扇出，那么仅采用 2 个阶段是无效的。



建议：器件中的最优 LUT:FF 比率为 1:1。如果设计所含 FF 数量显著增加，就会将更多不相关的逻辑封装成 slice，这将造成布线复杂性增加并且可能导致 QoR 劣化。

平衡流水线深度和 SRL 的使用

如果寄存器流水线较深，应尽可能将更多寄存器映射在 SRL 中，以避免寄存器使用率显著增加。例如，宽度为 32 的数据的流水线深度为 9，这会导致每个比特 9 个寄存器，总计使用 $32 \times 9 = 288$ 个寄存器。要将同样的结构映射到 SRL，需使用 32 个 SRL。每个 SRL 的地址管脚 A4 到 A0 都连接到 5'b01000，从而实现 9 个阶段的深度。

在综合期间可通过多种方法来推断 SRL，包括：

- SRL
- REG -> SRL
- SRL -> REG
- REG -> SRL -> REG

您可在 RTL 代码中使用 `srl_style` 属性创建这些结构，如下所示：

- (* srl_style = "srl" *)
- (* srl_style = "reg_srl" *)
- (* srl_style = "srl_reg" *)
- (* srl_style = "reg_srl_reg" *)

常见的错误是在较深的流水线阶段中使用不同的使能/复位控制信号。以下示例显示的是深度为 9 的流水线阶段中的复位，其中复位连接到第 3、第 5 和第 8 个流水线阶段。对于这种结构，工具只能将流水线阶段映射到寄存器，因为在 SRL 原语中有 1 个复位管脚。

```
FF->FF->FF(reset) -> FF->FF(reset)->FF->FF->FF(reset)->FF
```

要充分利用 SRL 推断，请：

- 确保流水线阶段不含任何复位。
- 分析是否确实需要复位。
- 在 1 个触发器上使用复位（例如，在流水线的第一个阶段或最后一个阶段）。

避免不必要的流水打拍

对于使用率较高的设计，流水打拍过多可能导致次优结果。例如，不必要的流水线级增加了触发器和布线资源的数量，如果使用率高，这可能限制布局和布线算法。

注释：如果有许多具有 0/1 逻辑级别的路径，请检查以确保这是有意为之。

考虑流水打拍宏原语

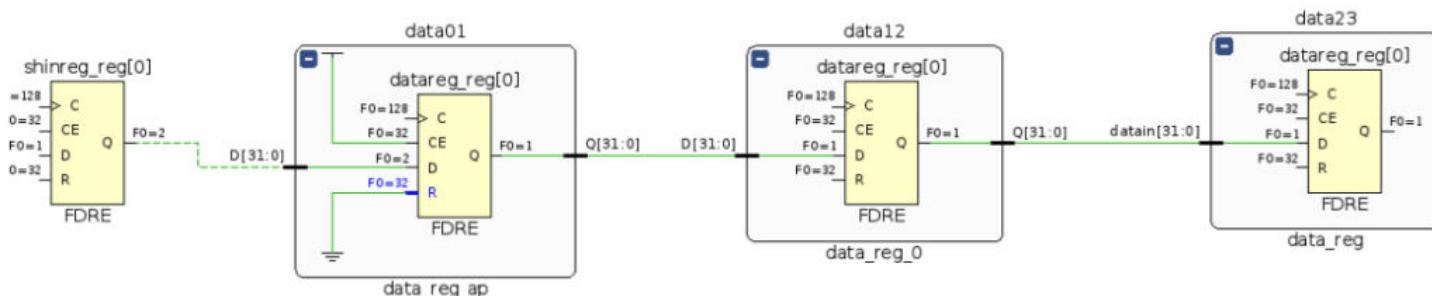
基于目标架构，如果使用足够的流水线，专用原语（如块 RAM 和 DSP）可在 500 MHz 以上的频率范围工作。对于高频设计，AMD 建议在这些块中使用所有流水线。

自动流水打拍注意事项

自动流水打拍功能支持布局器判断所需流水打拍阶段数量及其最佳位置，从而帮助跨接口边界实现时序收敛。您可通过设置 AXI Register Slice 核的自动流水打拍功能或者通过为数据总线应用自动流水打拍 HDL 属性或 XDC 约束来启用该功能。由于插入受时序驱动，因此请务必始终确保在目标路径上应用正确的时序约束。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。

以下示例显示了模块 data01 与 data12 之间的接口上应用的自动流水打拍。data01 的输出由不含控制集的寄存器组成。

图 40：模块间的简单数据流连接



此示例的 RTL 代码如下所示。`autopipeline_module` 属性应用于分层模块 data01 上，而 `autopipeline_group/autopipeline_limit/autopipeline_include` 属性则应用于由寄存器的 Q 管脚直接驱动的信号线上。

```
data_reg_ap #( .C_DATA_WIDTH(C_DATA_WIDTH) ) data01 (
    .clk (clk),
    .datain (shinreg),
    .datareg (d1)
);

data_reg #( .C_DATA_WIDTH(C_DATA_WIDTH) ) data12 (
    .clk (clk),
    .datain (d1),
    .datareg (d2)
```

```
) ;  
  
(* autopipeline_module= "yes" *)  
module data_reg_ap #(  
parameter integer C_DATA_WIDTH = 32  
)  
( input wire clk,  
input wire [C_DATA_WIDTH-1:0] datain,  
(* autopipeline_group= "fwd", autopipeline_limit=24 *)  
output reg [C_DATA_WIDTH-1:0] datareg  
) ;  
  
always @(posedge clk) begin  
datareg <= datain;  
end  
endmodule
```

此示例提供了在 RTL 代码中使用上述属性的另一种替代方式，其 XDC 约束如下所示。

```
# It's suggested to add the USER_SLR_ASSIGNMENT property at the module  
# level to ensure better logic clustering with its driver and load, see  
UG912  
#for more details on this property  
set_property USER_SLR_ASSIGNMENT APSRC [get_cells data01]  
set_property USER_SLR_ASSIGNMENT APDST [get_cells data12]  
  
set_property AUTOPIPELINE_MODULE TRUE [get_cells data01]  
set_property AUTOPIPELINE_GROUP WBUS [get_nets -of [get_pins -filter  
REF_PIN_NAME==Q -of [get_cells data01/*]]]  
set_property AUTOPIPELINE_LIMIT 10 [get_nets -of [get_pins -filter  
REF_PIN_NAME==Q -of [get_cells data01/*]]]
```

改善功耗的编码样式

时钟或数据路径门控

不使用时钟路径或数据路径的结果时，常用的停止转换的方法是对这些路径进行门控。对时钟进行门控可停止所有受驱动的同步负载，并阻止数据路径信号开关和毛刺继续进行传输。

功耗最优化 (`power_opt_design`) 可自动生成信号门控逻辑，以减少开关活动。但您可获得有关应用、数据流和依赖关系的信息，这些信息对于工具不可用并且只有您才能指定。

最大限度增加门控元件数量

最大限度增加受门控信号影响的元件数量。例如，在驱动源位置对时钟域进行门控相比使用时钟使能信号对每个负载进行门控更节省功耗。

使用专用时钟缓冲器的时钟使能管脚

在通过对时钟进行门控或多路复用以最大限度减少活动或降低时钟树使用率时，请使用专用时钟缓冲器的时钟使能端口。无论是插入 LUT 还是使用其他方法通过门控关闭时钟信号，对于满足功耗和时序要求的效果都不理想。

无需优先级编码器时使用 Case 块

不需要进行优先级编码时，请使用 case 块代替 if-then-else 块或三元运算符。

低效率编码示例：

```
if (reg1)
    val = reg_in1;
else if (reg2)
    val = reg_in2;
else if (reg3)
    val = reg_in3;
else val = reg_in4;
```

正确编码示例：

```
(* parallel_case *) casex ({reg1, reg2, reg3})
1xx: val = reg_in1 ;
01x: val = reg_in2 ;
001: val = reg_in3 ;
default: val = reg_in4 ;
endcase
```

块 RAM 的性能/功耗利弊取舍

有多种方法可用于细分存储器配置以满足具体要求。特定器件的要求可包括时钟频率、功耗或两者混合。

以下示例着重演示了为满足您的要求而可生成的各种不同结构。综合可使用 CASCADE_HEIGHT 属性来限制块 RAM 级联以实现时钟频率与功耗之间的利弊取舍。如需了解该属性的用法和实参，请参阅《Vivado Design Suite 用户指南：综合》(UG901)。

下图显示了为实现更高时钟频率（时序）的 4Kx32 存储器配置的示例。

注释：此示例仅适用于 UltraScale 和 UltraScale+ 器件。

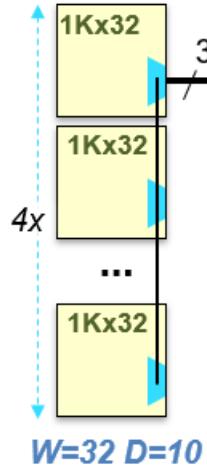
图 41：使用 4Kx8 和 CASCADE_HEIGHT=1 实现 4Kx32 的 RTL 表达形式



在此实现中，所有块 RAM 都始终处于启用状态（针对每次读取或写入）并耗用更多功耗。

下图显示了级联所有块 RAM 以降低功耗的示例。

图 42：使用 1Kx32 和 CASCADE_HEIGHT=4 实现 4Kx32 的 RTL 表达形式



```
module test(
    input clk,
    input we,
    input [31:0] din,
    input [11:0] addr,
    output reg [31:0] dout
);

(* ram_style = "block", cascade_height = 4 *)
reg [31:0] mem [(2**12)-1:0];
reg [11:0] addr_reg;

always @(posedge clk)
begin
    addr_reg <= addr;
    dout <= mem[addr_reg];
    if (we)
        mem[addr_reg] <= din;
end

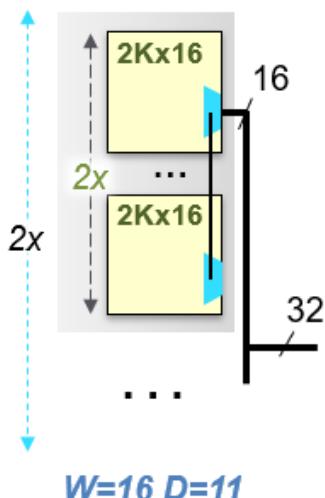
endmodule
```

在此实现中，由于每次仅选中 1 个块 RAM（从每个单元中），因此动态功耗贡献几乎减半。块 RAM 具有专用级联 MUX 和布线结构，支持构建宽而深的存储器，此类存储器需要在功耗效率极高的配置中构建多个块 RAM 原语。

下图显示了如何限制级联并同时保障功耗和时钟频率（通常无需牺牲性能）的示例。

注释：此示例仅适用于 UltraScale 和 UltraScale+ 器件。

图 43：使用 2Kx16 和 CASCADE_HEIGHT=2 实现 4Kx32 的 RTL 表达形式



```
module test(
    input clk,
    input we,
    input [31:0] din,
    input [11:0] addr,
    output reg [31:0] dout
);

(* ram_style = "block", cascade_height = 2 *)
reg [31:0] mem [(2**12)-1:0];
reg [11:0] addr_reg;

always @(posedge clk)
begin
    addr_reg <= addr;
    dout <= mem[addr_reg];
    if (we)
        mem[addr_reg] <= din;
end

endmodule
```

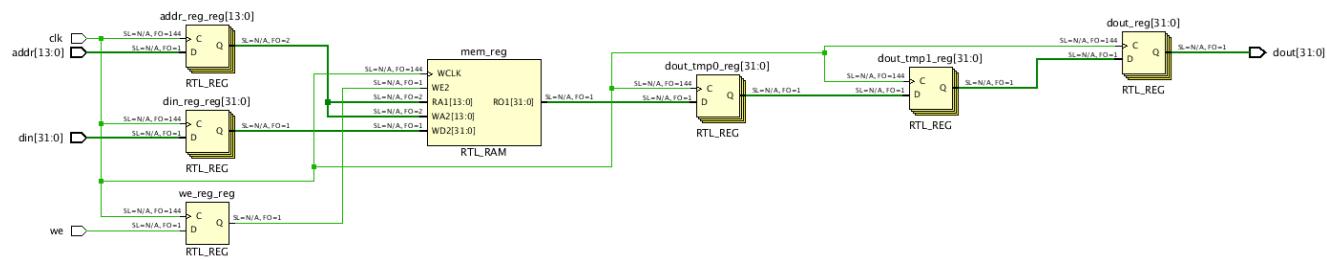
在此实现中，由于每次仅选中2个块RAM，因此动态功耗贡献比高时钟频率结构更好，但不及低功耗结构。此结构相比于低功耗结构的优势在于它在级联路径中仅使用2个块RAM，而相比于低功耗结构的关键路径中的4个块RAM，这将影响目标频率。

分解更深的存储器配置，实现功耗与时钟频率平衡

在使用更深的存储器配置时，可在RTL中使用RAM_DECOMP综合属性，通过改进存储器组合来降低功耗。当RAM_DECOMP属性被应用到存储器阵列上时，存储器逻辑将映射到更宽的块RAM原语阵列上。为平衡功耗与时钟频率，您可以使用CASCADE_HEIGHT属性和RAM_DECOMP属性来控制级联。这种方法需要更多的地址解码逻辑，但有助于减少为每个读取操作启用的块RAM数量，从而帮助降低功耗。

例如，下图显示的是1个32x16K存储器配置。

图44：32x16K存储器配置



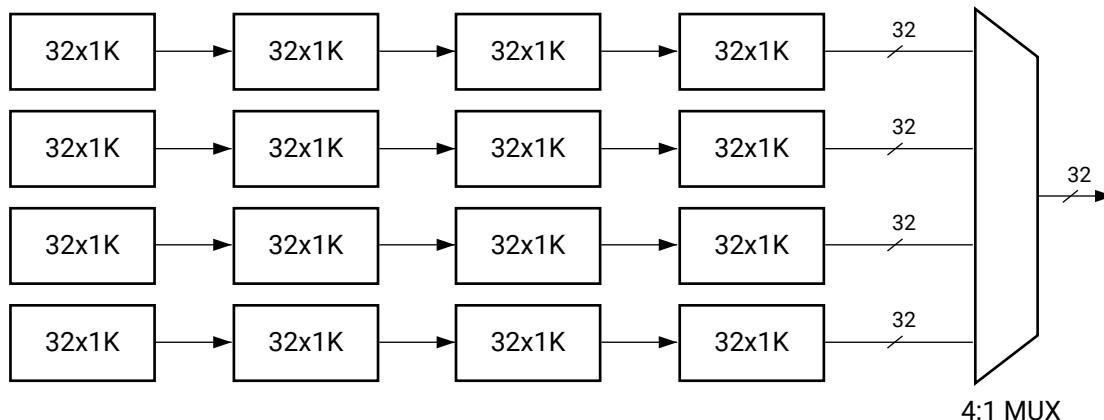
如果您应用下列属性：

```
ram_decomp = "power"
cascade_height = 4
```

将推断得出16个RAMB36E2且存储器分解方式如下：

- 基本原语为32x1K。
- 4个块RAM将通过级联创建32x4K配置。
- 4个并行结构可创建1个16K深的存储器。
- 输出通过多路复用生成输出数据。

图45：使用CASCADE_HEIGHT和RAM_DECOMP属性生成的32x16K存储器配置结构示例



X19283-121919

以下 RTL 代码示例显示了 CASCADE_HEIGHT 和 RAM_DECOMP 属性的用例。

图46：使用CASCADE_HEIGHT和RAM_DECOMP属性的32x16K存储器配置的RTL代码

```

module test
(
    input clk,
    input we,
    input [13:0] addr,
    input [31:0] din,
    output reg [31:0] dout
);

(* ram_style = "block", ram_decomp = "power", cascade_height = 4 *) reg [31:0] mem [(16*1024)-1:0];
reg [13:0] addr_reg;
reg [31:0] dout_tmp0;
reg [31:0] dout_tmp1;
reg [31:0] din_reg;
reg         we_reg;

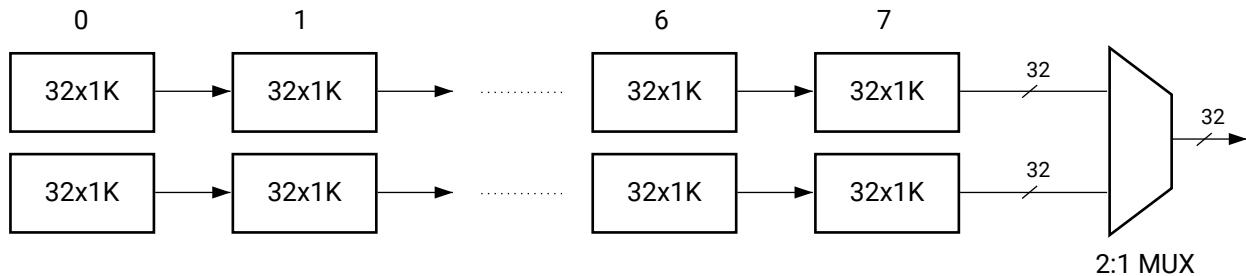
always @(posedge clk)
begin
    addr_reg <= addr;
    din_reg  <= din;
    we_reg   <= we;
    dout_tmp0 <= mem[addr_reg];
    dout_tmp1 <= dout_tmp0;
    dout <= dout_tmp1;
    if (we_reg)
        mem[addr_reg] <= din_reg;
end
endmodule

```

如果仅应用 ram_decomp = "power" 属性，那么将推断得出 16 个 RAMB36E2 并且存储器分解方式如下：

- 基本原语为 32x1K。
- 8 个块 RAM 将通过级联创建 32x8K 配置。
- 2 个并行结构可创建 1 个 16K 深的存储器。
- 输出通过多路复用生成 2:1 MUX，以生成输出数据。

图 47：使用 RAM_DECOMP 属性生成的 32x16K 存储器配置的结构



X19284-050517

以下 RTL 代码示例显示了 RAM_DECOMP 属性的用例。

图 48：使用 RAM_DECOMP 属性的 32x16K 存储器配置的 RTL 代码

```

| module test
|
| input  clk,
| input  we,
| input  [13:0] addr,
| input  [31:0] din,
| output reg [31:0] dout
| );
|
| (* ram_style = "block", ram_decomp = "power")* reg [31:0] mem [(16*1024)-1:0];
| reg [13:0] addr_reg;
| reg [31:0] dout_tmp0;
| reg [31:0] dout_tmp1;
| reg [31:0] din_reg;
| reg [31:0] we_reg;
|
| always @(posedge clk)
| begin
|     addr_reg <= addr;
|     din_reg <= din;
|     we_reg <= we;
|     dout_tmp0 <= mem[addr_reg];
|     dout_tmp1 <= dout_tmp0;
|     dout <= dout_tmp1;
|     if (we_reg)
|         mem[addr_reg] <= din_reg;
| end
|
| endmodule

```

如果仅使用 RAM_DECOMP 属性，那么总体功耗节省与同时使用 RAM_DECOMP 和 CASCADE_HEIGHT 属性相似，因为每次只有 1 个块 RAM 处于活动状态。为了实现最高时钟频率，创建深度为 4 的级联块 RAM 链比深度为 8 的级联块 RAM 链效果更好。

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。

运行 RTL DRC

有一组 RTL DRC 规则用于识别 HDL 潜在的编码问题。您可单击 Flow Navigator 下的“Open Elaborated Design”便可打开细化视图并进行检查。您可在 Flow Navigator 中选中“RTL Analysis”→“Report Methodology”(RTL 分析 > 方法论报告) 或者在 Tcl 命令提示符处执行 report_methodology，以运行这些 DRC 检查。

时钟设置准则

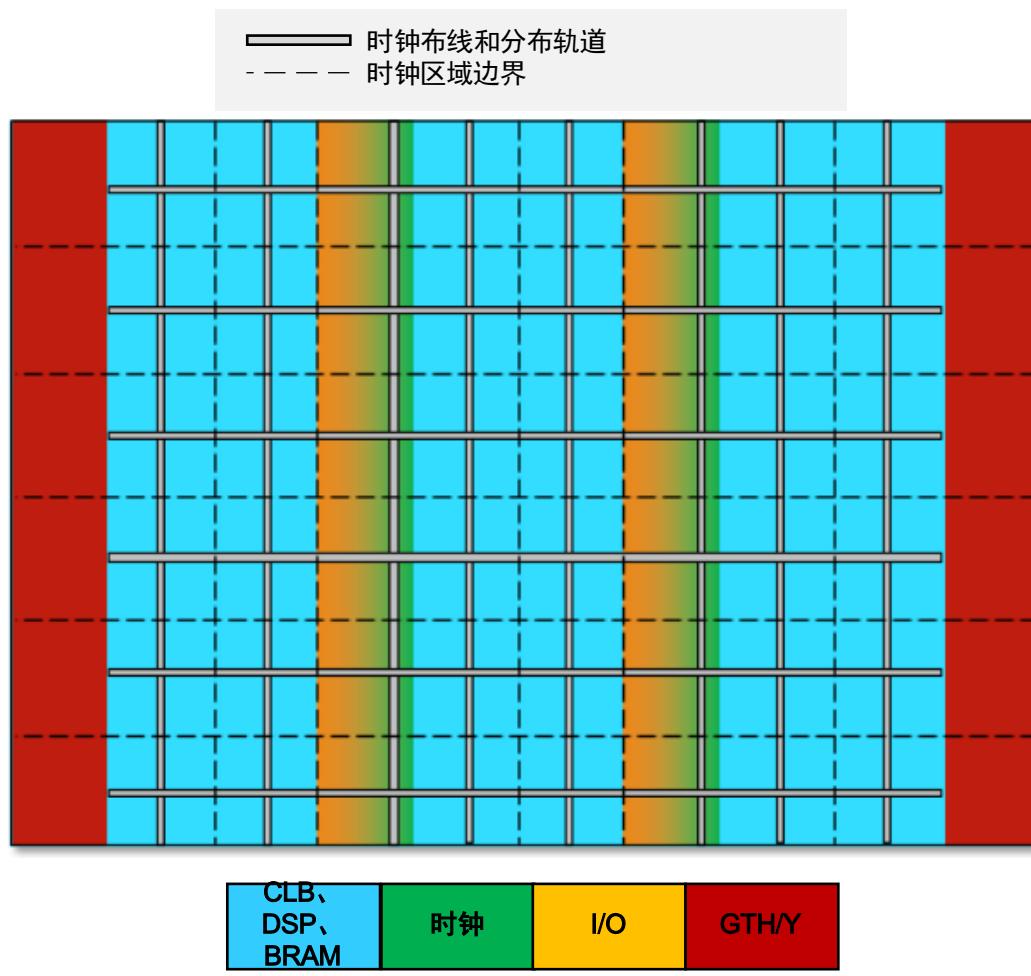
每个器件架构都有用于时钟设置的专用资源。掌握器件架构中的时钟设置资源将使您能够规划好自己的时钟设置，从而实现时钟资源的最佳利用。大多数设计无需您了解这些细节。但如果您能够控制布局，同时熟悉每个时钟域上的扇出，就可以根据以下时钟设置详情，研究出多种备选方案。如果您决定使用任何时钟设置资源，就需要显式例化相应的时钟元件。

UltraScale 器件时钟设置

UltraScale 器件与先前器件架构的时钟结构不同，这模糊了全局时钟与区域时钟之间的界限。UltraScale 器件与 7 系列器件的不同之处在于，前者不具有区域时钟缓冲器，而是改为对局部负载、区域负载和全局负载都使用公用缓冲器和时钟布线结构。

UltraScale 器件在器件之间采用较小的固定大小时钟区域，并且时钟区域不再占据器件水平方向上的一半宽度。每个 UltraScale 器件上每一行的时钟区域数量都不尽相同。每个时钟区域都包含一种时钟网络布线，可分频至 24 条垂直和水平布线轨道以及 24 条垂直和水平分布轨道。下图显示了含 36 个时钟区域（6 列 x 6 行）的器件。同等的 7 系列器件具有 12 个时钟区域（2 列 x 6 行）。

图49：UltraScale器件时钟区域拼块(tile)



X15241-052822

时钟架构设计为仅使用连接给定布局的时钟缓冲器和负载所必需的时钟资源即可，在不含负载的时钟区域内不浪费任何资源。有效利用时钟资源即可支持在架构中添加更多设计时钟，同时提升时钟的性能和功耗特性。以下是时钟类型的主要类别及其关联的时钟结构（按其驱动和用途分组）：

- 高速 I/O 时钟

这些时钟与由 PLL 生成的高速 SelectIO™ 接口位 slice 逻辑关联，并通过专用的低抖动资源来布线到高速 I/O 接口的位 slice 逻辑。通常，此时钟结构由 AMD IP（例如，存储器 IP 或 High Speed SelectIO Wizard）创建并控制，并非由用户指定。

- 通用时钟

这些时钟适用于大部分时钟树结构，可通过 GCIO 封装管脚、MMCM/PLL 或互连结构逻辑单元（通常不建议）提供。通用时钟网络必须由包含 I/O 列的任意时钟区域内提供的 BUFGCE/BUFGCE_DIV/BUFGCTRL 缓冲器来驱动。任意给定时钟区域均可支持最多 24 个独立时钟，而大部分 UltraScale 器件根据其拓扑结构、扇出和负载布局，均可支持 100 余个时钟树。

- 千兆位收发器 (GT) 时钟

千兆位收发器 (GTH 或 GTY) 的发射、接收和参考时钟均使用包含 GT 的时钟区域内的专用时钟。您可以使用 GT 时钟来实现以下功能：

- 使用 BUFG_GT 缓冲器连接到互连结构中的任何负载，以驱动通用时钟网络
- 在相同或不同四通道 (Quad) 中的多个收发器上共享时钟

时钟原语

大部分时钟通过支持全局时钟的 I/O (GCIO) 管脚进入器件。这些时钟通过时钟缓冲器直接驱动时钟网络，或者由位于 I/O 列附近的时钟管理模块 (CMT) 中的 PLL 或 MMCM 加以变换。

CMT 包含以下时钟资源：

- 时钟生成块
 - 2 个 PLL
 - 1 个 MMCM
- 全局时钟缓冲器
 - 24 个 BUFGCE
 - 8 个 BUFGCTRL
 - 4 个 BUFGCE_DIV

注释：不含绑定 I/O 的 I/O 列附近的 CMT 中的时钟资源可供使用。

GT 用户时钟通过 BUFG_GT 缓冲器驱动全局时钟网络。GTH/GTY 列附近的每个时钟区域都包含 24 个 BUFG_GT 缓冲器。

以下是每个 UltraScale 器件时钟缓冲器的汇总信息：

- BUFGCE

最常用的缓冲器是 BUFGCE。它是通用时钟缓冲器，具有等同于 7 系列 BUFHCE 的时钟启用/禁用功能。

- BUFGCE_DIV

可在以下情况下使用 BUFGCE_DIV：需要对时钟进行简单分频时。相比于使用 MMCM 或 PLL 进行简单时钟分频，它更便于使用并且能效更高。只要正确使用，相比于跨时钟域时使用 MMCM 或 PLL，由此产生的时钟域间偏差更小。BUFGCE_DIV 常用于替代 7 系列器件中的 BUFR 功能。但由于 BUFGCE_DIV 可驱动全局时钟网络，因此其功能比 BUFR 组件更强大。

- BUFGCTRL (和 BUFGMUX)

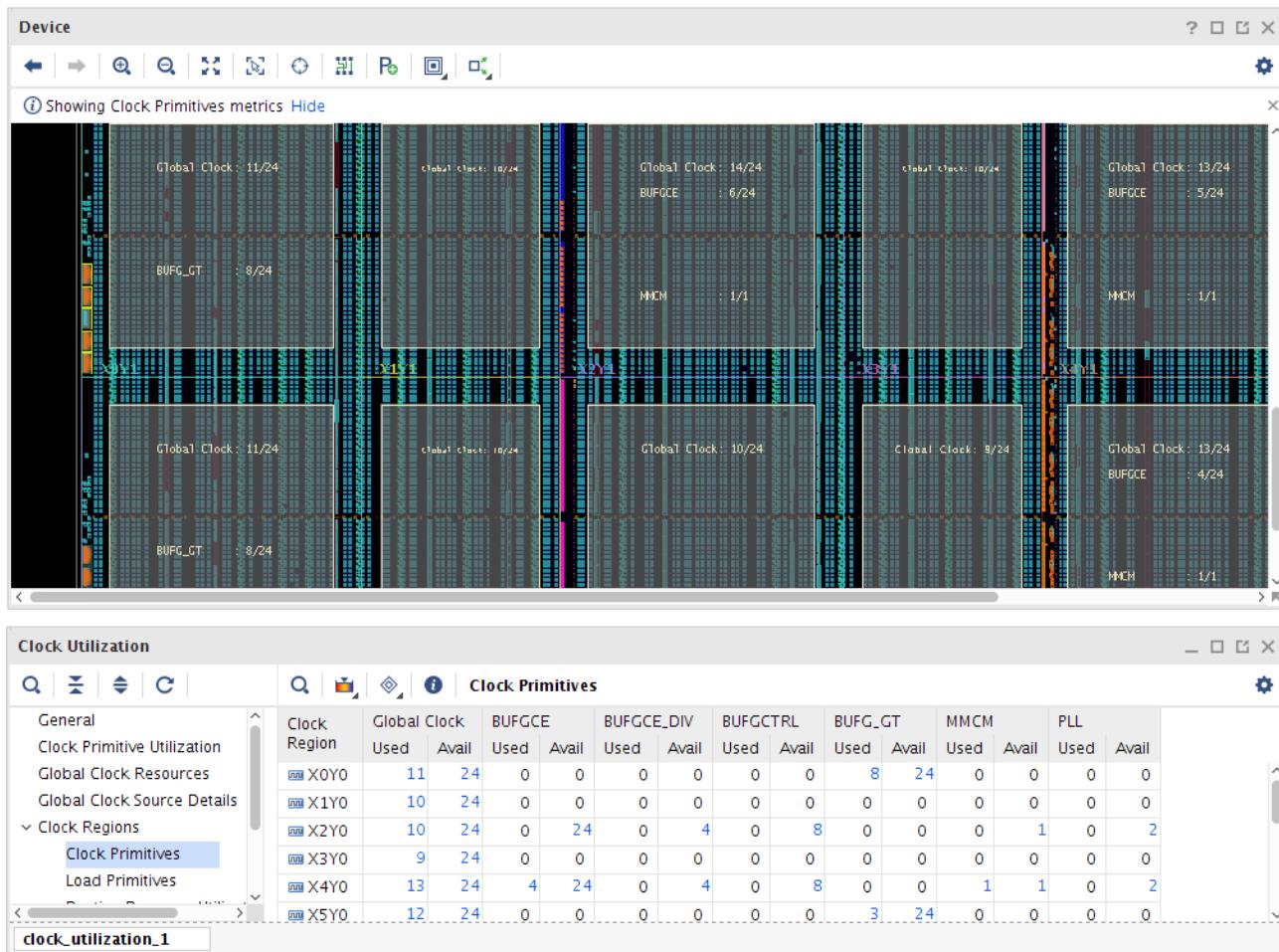
BUFGCTRL 可例化为 BUFGMUX，通常可在以下情况下使用：对 2 个或更多个时钟源进行多路复用以构成单个时钟网络时。就像 BUFGCE 和 BUFGCE_DIV 一样，它可驱动时钟网络用于区域时钟设置或全局时钟设置。

- BUFG_GT

使用由 GT 生成的时钟时，BUFG_GT 时钟缓冲器允许连接至全局时钟网络。大多数情况下，BUFG_GT 用作为区域缓冲器，其负载布局在 1 或 2 个相邻时钟区域内。BUFG_GT 内置动态时钟分频功能，可替代 MMCM 执行时钟速率更改。

您可以使用 Vivado IDE 中的“Clock Utilization”（时钟利用率）报告来以可视化方式分析时钟资源使用率和时钟布线。下图显示了在“Device”（器件）窗口中叠加的时钟资源使用率（按时钟区域）。如需了解有关此报告的更多信息，请参阅《Vivado Design Suite 用户指南：设计分析与收敛技巧》(UG906)。

图50：“Clock Utilization”报告



如需了解有关 BUFGCE、BUFGCE_DIV、和 BUFGCTRL 缓冲器的更多信息，请参阅《UltraScale 架构时钟资源用户指南》([UG572](#))。如需了解有关 BUFG_GT 缓冲器连接和使用详情，请参阅相应的《UltraScale 架构收发器用户指南》：

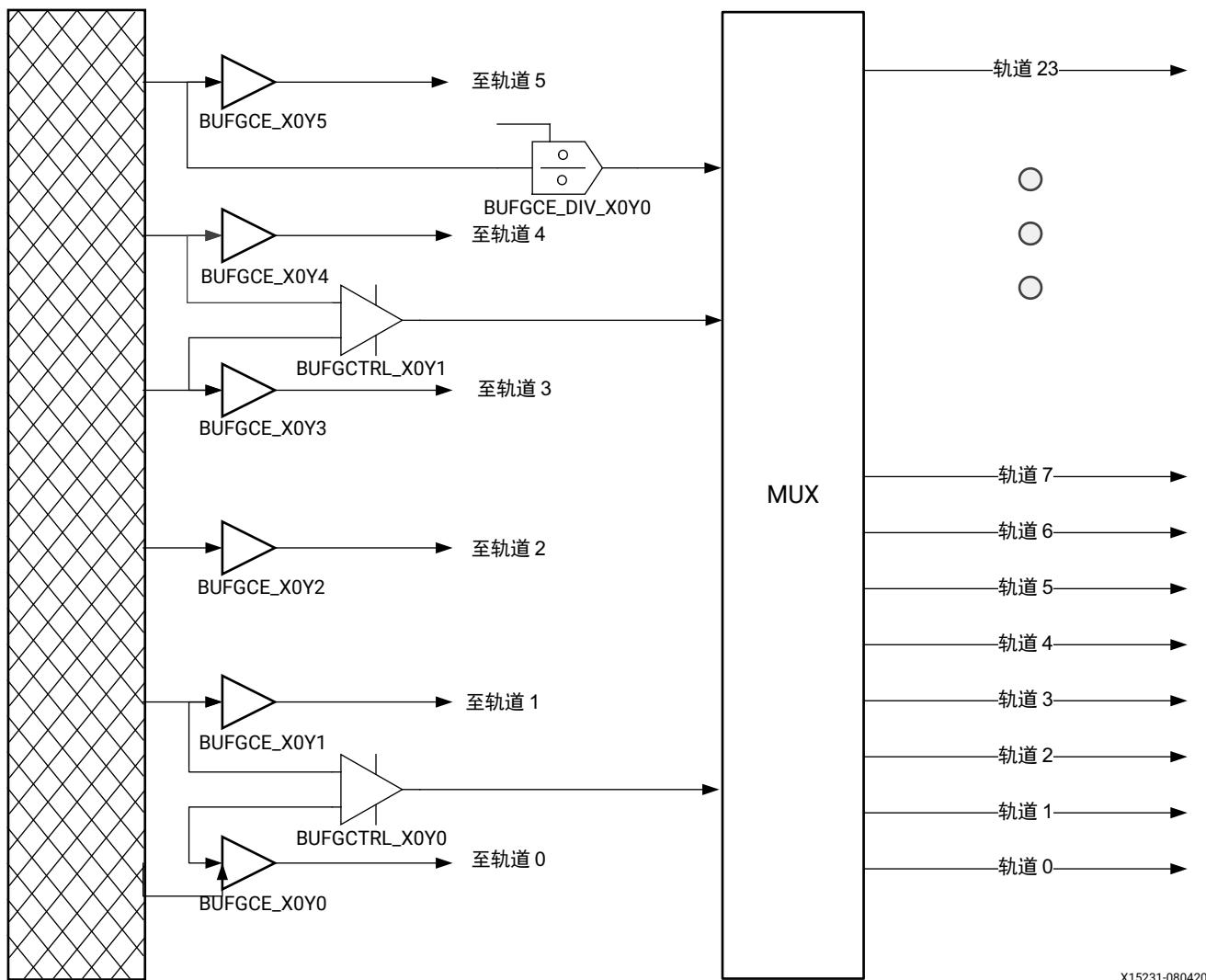
- 《UltraScale 架构 GTH 收发器用户指南》([UG576](#))
- 《UltraScale 架构 GTY 收发器用户指南》([UG578](#))

全局时钟缓冲器连接和布线轨道

每个时钟区域中的全部 24 个 BUFGCE 缓冲器各自都只能驱动特定的时钟布线轨道。然而，BUFGCTRL 和 BUFGCE_DIV 输出可以通过 MUX 结构使用 24 条轨道中的任何一条。每个 BUFGCE_DIV 与特定的 BUFGCE 站点(site)共享输入连接，每个 BUFGCTRL 与 2 个特定的 BUFGCE 站点(site)共享输入连接。因此，当时钟区域中使用 BUFGCE_DIV 或 BUFGCTRL 缓冲器时，BUFGCE 缓冲器的使用会受到限制。下图显示了时钟区域中底部的 6 个 BUFGCE，这 6 个 BUFGCE 在时钟区域内复制 4 次。

注释：针对器件中每个特定轨道 ID 都会分配 1 个全局时钟信号线，以供时钟所使用的所有垂直布线、水平布线和分布资源使用。除非时钟穿过另一个时钟缓冲器，否则无法更改轨道 ID。

图 51：BUFGCE、BUFGCE_DIV 和 BUFGCTRL 共享输入和输出多路复用



时钟布线、时钟根和时钟分布

要正确理解 UltraScale 器件的时钟容量以及设计的时钟使用率，重要的是了解时钟布线使用专用布线资源的方式：

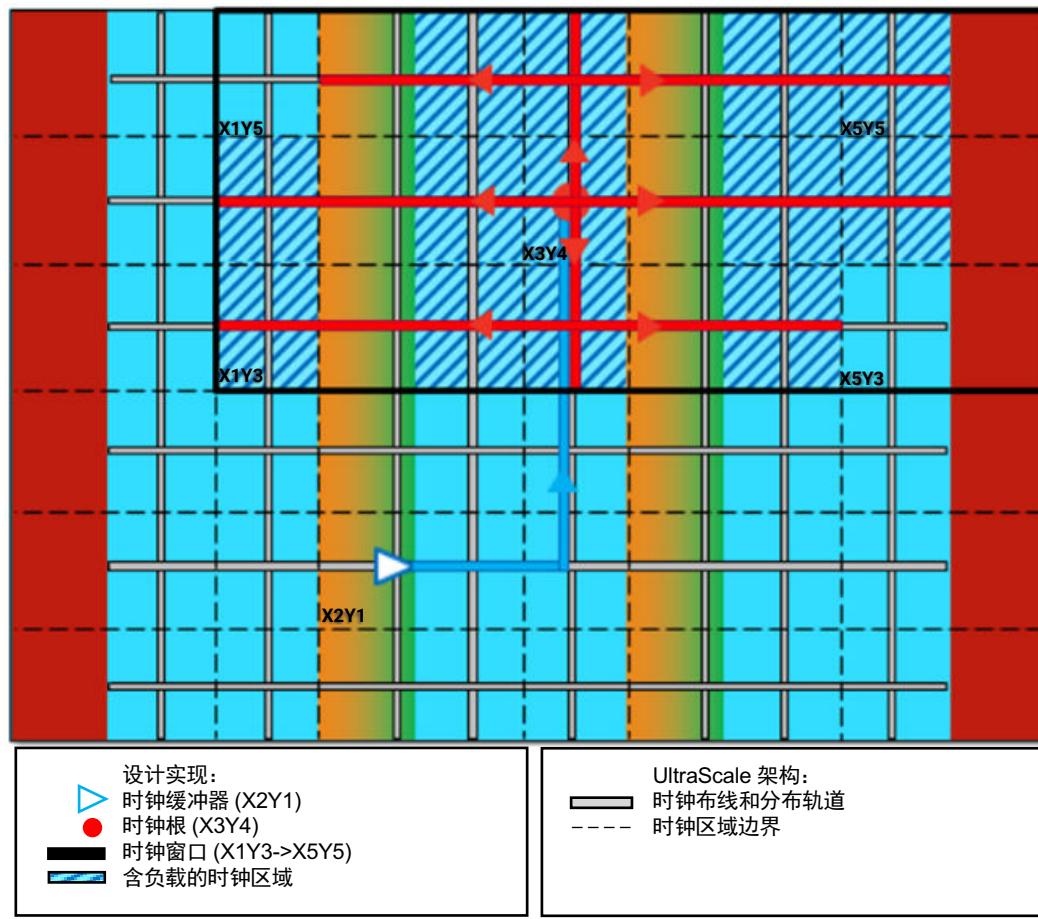
- 从时钟缓冲器到时钟根，时钟信号经过 1 个或多个垂直和水平布线阶段。每个分段都必须使用相同的轨道 ID（介于 0 和 23 之间）。
- 在时钟根处，时钟信号从布线轨道转变为具有相同轨道 ID 的分布轨道。为了减少偏差，时钟根通常在位于时钟窗口中心的时钟区域中。时钟窗口是包含布局时钟信号线负载的所有时钟区域的矩形区域。出于偏差最优化的原因，Vivado IDE 可能将时钟根移动到偏离中心的位置。
- 从时钟根到负载所在的 CLB 列，时钟信号垂直分布（根据需要沿器件向上和向下）方向移动，然后沿水平分布（根据需要向左和向右）移动。
- CLB 列分成两半，分别位于水平分布资源的上方和下方。每一半 CLB 列都包含几个叶时钟布线资源，可以通过任何水平分布轨道来获取。

在某些情况下，时钟缓冲器可以直接驱动到时钟分布轨道上。当时钟根与时钟缓冲器位于相同时钟区域中或者当时钟缓冲器仅驱动非时钟管脚（例如，高扇出信号线）时，通常会发生此情况。

因为时钟布线资源已分段，因此所耗用的资源仅限用于遍历时钟区域或访问时钟区域内的负载的布线和分布段。

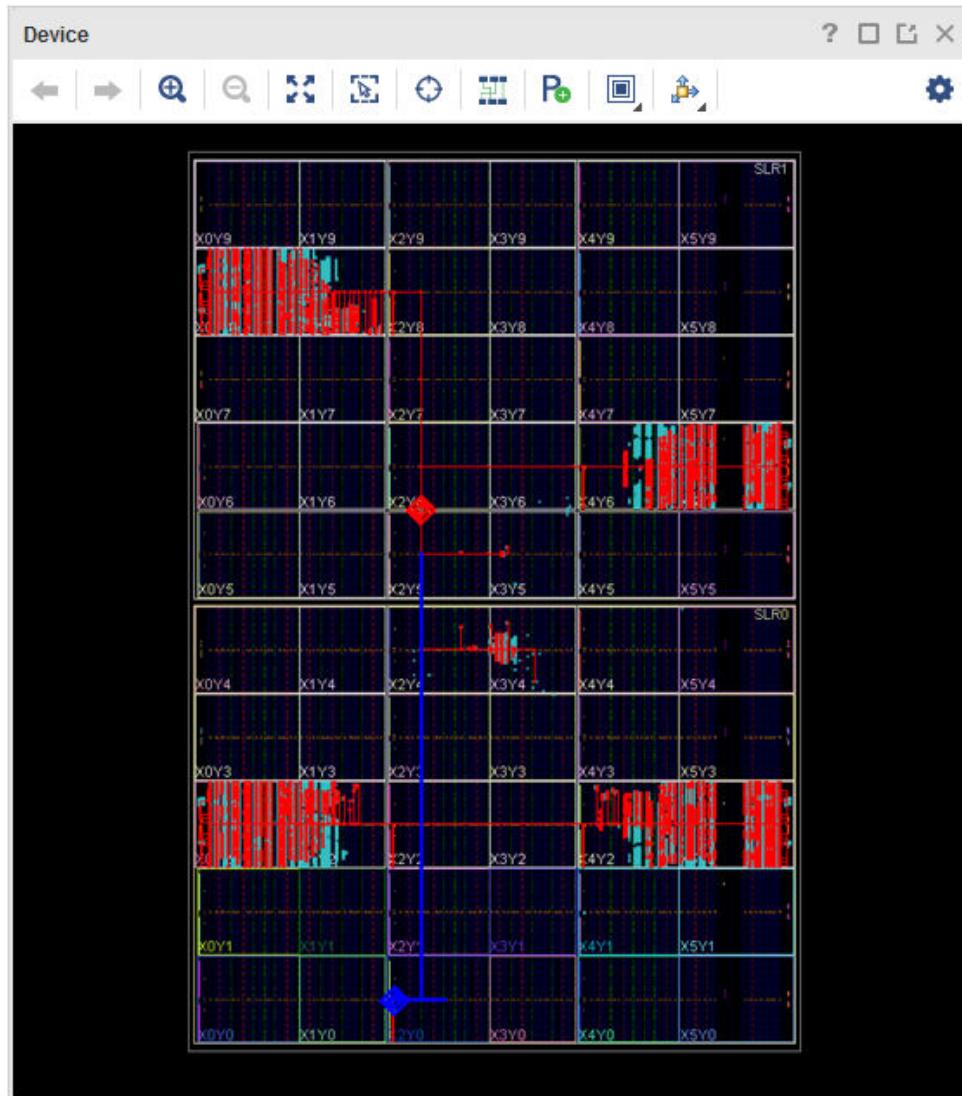
下图显示了位于时钟区域 X2Y1 内的时钟缓冲器访问其布局在时钟窗口内的负载的方式，该时钟窗口是由从 X1Y3 到 X5Y5 的矩形时钟区域构成的。

图 52：UltraScale 器件时钟布线（从驱动到负载）



在下图中，已布线的器件视图显示了遍布器件上大部分区域的全局时钟示例。驱动网络的时钟缓冲器在时钟区域 X2Y0 内标记为蓝色，并驱动到该时钟区域内的水平布线。然后，信号线从水平布线转换到时钟区域 X2Y0 中的垂直布线，并到达时钟区域 X2Y5 内的时钟根。所有时钟布线均标记为蓝色。时钟根在时钟区域 X2Y5 中标记为红色。信号线从 X2Y5 中的时钟根转换到垂直分布，然后通过水平分布到达时钟叶管脚。CLB 列中的分布层和叶时钟布线资源标记为红色。

图 53：已布线的时钟网络的布线器件视图



时钟树布局布线

在以下阶段中，Vivado 布局器会在遵循物理 XDC 约束的同时，判定 MMCM/PLL、全局时钟缓冲器和时钟根的布局：

1. I/O 和时钟布局

布局器根据连接规则和用户约束来布局 I/O 缓冲器和 MMCM/PLL。布局器将时钟缓冲器分配给时钟区域，但不分配给单个站点 (site)，除非使用 LOC 属性进行约束。只有仅驱动非时钟负载的时钟缓冲器才能基于时钟区域的驱动和负载布局在流程中后续迁移到其他时钟区域。

在此阶段的任何布局器错误都是由于连接规则和/或用户约束存在冲突所导致的。log 日志文件会显示有关错误的可能根本原因的详细信息，您必须仔细查看以便执行适当的设计或约束更改。

2. SLR 分区（仅限 SSI 技术器件）和全局布局

布局器根据先前的驱动和负载布局来执行初始时钟树实现。每个时钟信号线都与 1 个时钟窗口相关联。因可预见的时钟布线争用，时钟窗口的过度重叠可能导致布局器错误。

当发生时钟分区错误时，log 日志文件会显示每个时钟信号线的最后一项时钟预算解决方案以及每个时钟区域中存在的唯一时钟信号线的数量。请详细审查 log 日志文件以确定要从过度使用的时钟区域中删除哪些时钟。您可使用以下方法删除时钟：

- 通过组合相同的同步时钟、去除不必要的 MMCM 反馈时钟或者将较低的扇出时钟与高扇出时钟合并来减少设计中的时钟数。
- 将时钟原语迁移到其他时钟区域，特别是不具有基于连接的布局规则的时钟原语。
- 在时钟负载上添加布局规划约束，以使扇出较小的时钟更接近其驱动或远离使用率较高的时钟区域。

布局器会对时钟树实现进行多次优化以帮助提高时序 QoR。例如，在后续布局最优化阶段期间，布局器会分析每个有问题的时钟以确定更合适的时钟根位置。

3. 时钟树预布线

布局器会指导后续实现步骤，并为布局后时序分析提供准确的延迟估算。

布局后，Vivado 工具可按如下所示修改时钟树实现：

- Vivado 物理最优化器可以将单元复制并移动到没有关联时钟的时钟区域。
- Vivado 布线器可通过调整来改进时序 QoR 并使时钟布线合规。

下表总结了主时钟拓扑的布局规则以及约束对这些规则的影响。

表 2：含布局规则和不含布局规则的拓扑结构

约束源	未约束的目标	行为
GCIO	BUFGCE、BUFGCTRL、BUFGCE_DIV、PLL/MMCM	自动布局在相同的时钟区域。
PLL/MMCM	BUFGCE、BUFGCTRL、BUFGCE_DIV	自动布局在相同的时钟区域。
GT*_CHANNEL	BUFG_GT	自动布局在相同的时钟区域。
BUFGCTRL	BUFGCTRL	自动布局在相同的时钟区域。 注释： 您可以使用 CLOCK_REGION 约束覆盖同一时钟区域中的布局。
BUFG*	BUFG*	未约束的目标 BUFG 的布局不可预测。 建议使用 CLOCK_REGION 约束来约束目标 BUFG*。 注释： 这不包括 BUFGCTRL > BUFGCTRL。
BUFG*	MMCM/PLL	未约束的目标 MMCM/PLL 的布局不可预测。 建议使用 LOC 约束来约束 MMCM/PLL。 当布线跨越相邻或多个时钟区域时，建议采用 CLOCK_DEDICATED_ROUTE 约束。

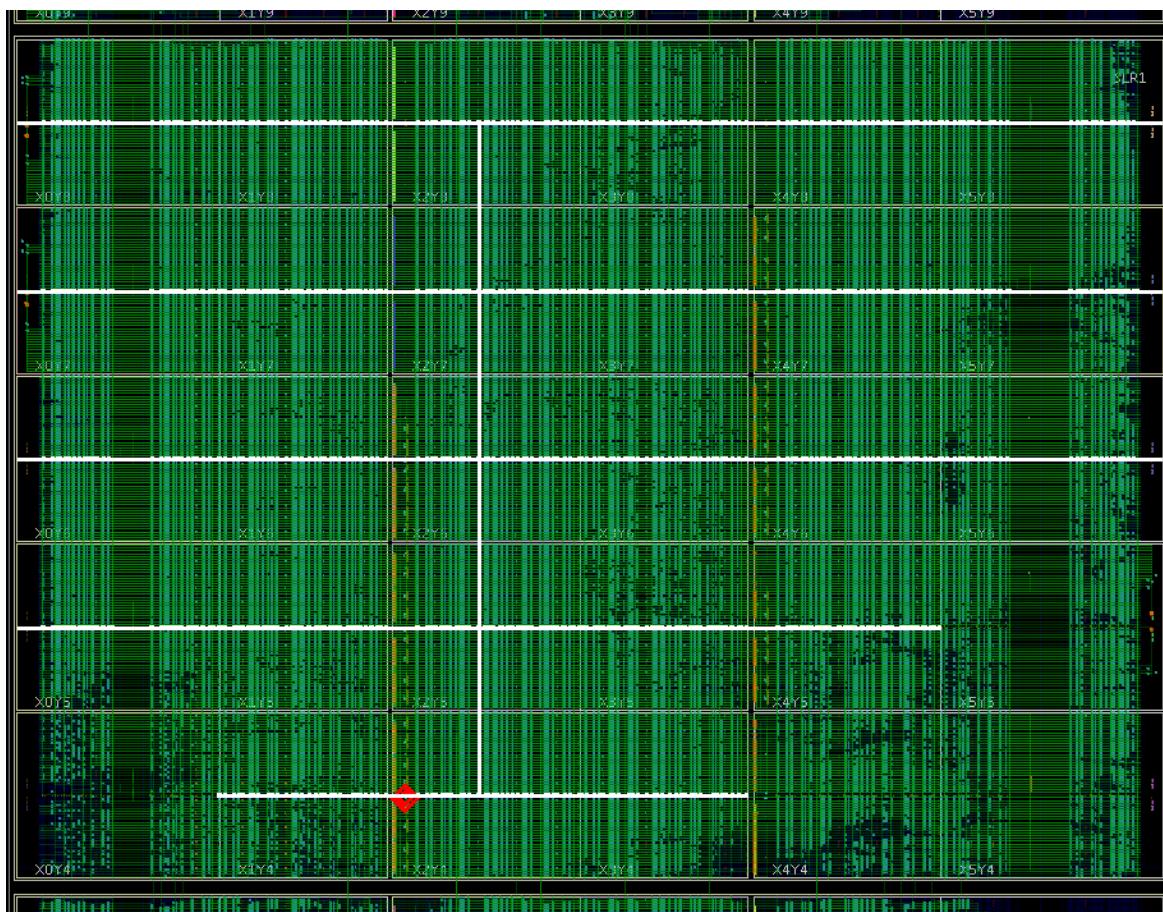
时钟功能

时钟规划必须根据目标器件中的高扇出时钟和低扇出时钟的总数来进行。

高扇出时钟

高扇出时钟覆盖了 SSI 技术器件的几乎整个 SLR 或单片器件的几乎所有时钟区域。下图显示了覆盖了 SSI 技术器件的几乎整个 SLR 的高扇出时钟，其中 BUFGCE 驱动以红色显示。

图 54：覆盖整个 SLR 的高扇出时钟



注释：在单一设计中使用的时钟数量如果超过 24 个，则可能会导致出现需要特殊设计考量或其他提前规划方可解决的问题。

重要提示！ 在 ZHOLD 和 BUF_IN 补偿模式下，MMCM 反馈时钟路径与 CLKOUT0 时钟路径在布线轨道、时钟根位置和分布轨道方面相匹配。因此，当时钟缓冲器与时钟根相距很远时，反馈时钟可以视为高扇出时钟。

相关信息

[含 MMCM ZHOLD/BUF_IN 补偿的 I/O 时序](#)

低扇出时钟

大多数情况下，低扇出时钟是连接到少于 5000 个时钟管脚的时钟信号线，这些时钟管脚布局在不超过 3 个水平相邻的时钟区域内。时钟布线、时钟根和时钟分布全部包含在局部区域内。

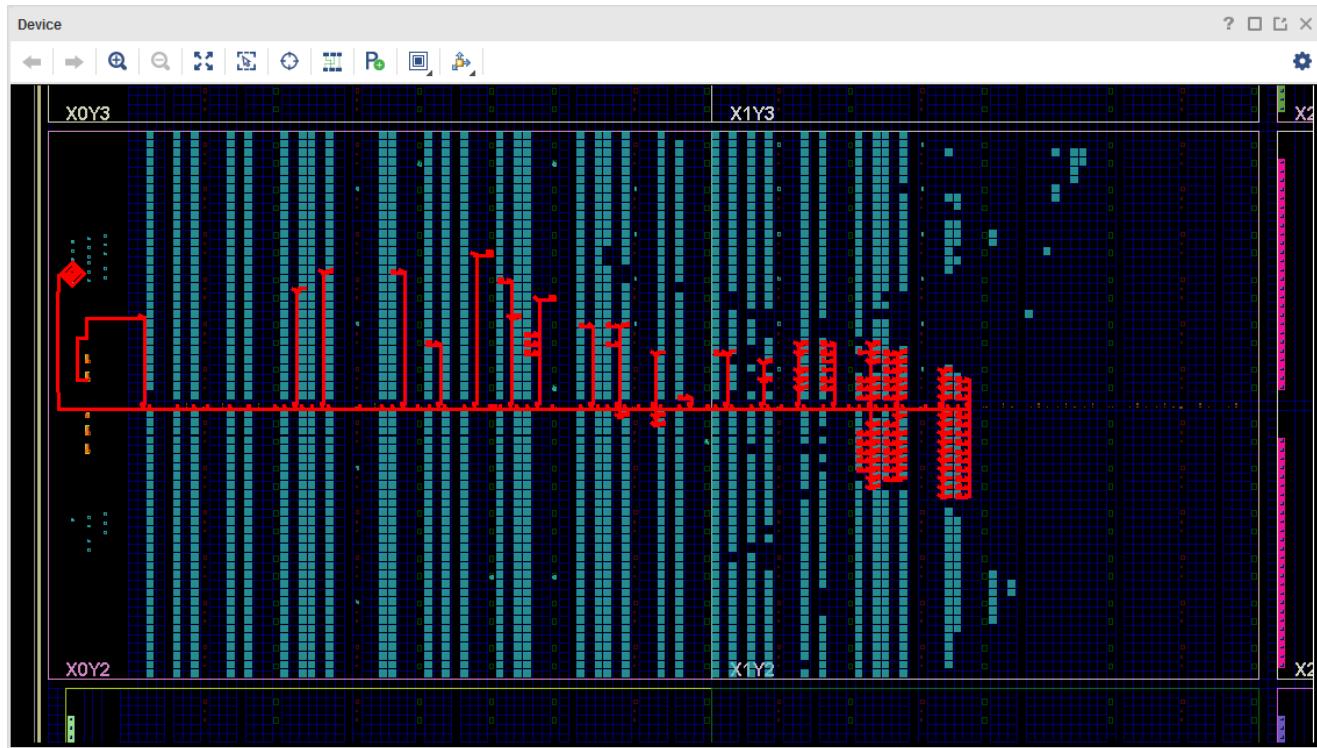
在某些情况下，布局器无法识别出本应可识别的低扇出时钟。这可能是由于设计大小、器件大小或物理 XDC 约束（如，LOC 约束或 Pblock）所导致的，这些因素可能导致布局器无法将负载布局到局部区域内。要解决此问题，您可能需要手动创建 Pblock 或修改现有物理约束以指导该工具完成操作。

由 BUFG_GT 驱动的时钟即为低扇出时钟的示例。Vivado 布局器会自动识别这些时钟信号线，并包含与 GT 接口相邻的时钟区域的负载。下图显示了包含在 2 个时钟区域内的低扇出时钟，其中 BUFG_GT 驱动以红色显示。



提示：您可使用 CLOCK_LOW_FANOUT XDC 约束将低扇出时钟包含在单个时钟区域内。

图 55：2 个时钟区域包含的低扇出时钟



相关信息

使用 CLOCK_LOW_FANOUT 约束

平衡使用高扇出时钟和低扇出时钟

UltraScale 器件支持的时钟比先前 AMD 器件系列支持的时钟数量更多。这样可以支持各种时钟使用场景，例如：

- 多达 24 个时钟

除非用户约束存在冲突，否则所有时钟均可作为高扇出时钟来处理，无需应对布局或布线争用的风险。

- 接近 300 个时钟

如果设计中所处理的目标器件具有 6 个时钟区域行并且仅包含低扇出时钟，每个时钟最多包含在 3 个时钟区域内，那么所需时钟如下所述：6 行 \times 每行 2 个时钟窗口 \times 每个区域 24 个时钟 = 288 个时钟。

低扇出时钟窗口大小不固定，通常在 1 到 3 个时钟区域内。高扇出时钟遍布整个器件或整个 SLR 的场景极为罕见。

以下方法显示了如何实现高扇出时钟与低扇出时钟的平衡，假定少数低扇出时钟来自于 I/O 接口，大部分来自于 GT 接口。您可对每个 SSI 技术器件 SLR 应用相同的方法。

- 高扇出时钟
 - 针对单片器件最多包含 12 个高扇出时钟
 - 针对 SSI 技术器件最多包含 24 个高扇出时钟（假定部分高扇出时钟仅存在于 1 个 SLR 中）

- 低扇出时钟
 - 最多 12 个低扇出时钟，外加每个 GT 使用的四通道 8 个低扇出时钟
 - 或者，最多 12 个低扇出时钟，外加每个 GT 接口 6 个低扇出时钟（共享 RXUSRCLK 和 TXUSRCLK 的 GT 通道组）

时钟约束

物理 XDC 约束驱动时钟树的实现并控制使用高扇出时钟资源。由于 UltraScale 器件时钟比具有先前架构的时钟更灵活，并且包含其他架构约束，因此，了解如何正确约束时钟以完成实现至关重要。

针对 IO/MMCM/PLL/GT 使用 LOC 约束

要约束时钟，可按如下所示分配布局约束：

- 在 I/O 端口的时钟输入处

为 GCIO 上的时钟分配 PACKAGE_PIN 约束或者将 LOC 分配给 IOB 会影响时钟网络。直接连接到输入端口的 MMCM/PLL 和时钟缓冲器必须布局在相同时钟区域内。

- 在 MMCM 或 PLL 上

直接连接到 MMCM 或 PLL 输出的时钟缓冲器和连接到 MMCM 或 PLL 输入的输入时钟端口将自动布局在相同时钟区域内。如果输入时钟端口与 MMCM 或 PLL 直接相连或者约束到不同时钟区域，那么必须手动插入时钟缓冲器，并在连接到 MMCM 或 PLL 的信号线上设置 CLOCK_DEDICATED_ROUTE 约束。

- 在 GT*_CHANNEL 或 IBUFDS_GT* 单元上

由单元驱动的 BUFG_GT 布局在相同时钟区域内。



注意！ AMD 不推荐在时钟缓冲器单元上使用 LOC 约束。此方法将为时钟强制分配特定轨道 ID，这可能导致布局无法进行规范布线。仅当您已明确了解设计的整个时钟树并且设计中的布局保持一致时，才能使用 LOC 约束在 UltraScale 器件中布局高扇出时钟缓冲器。即使在采取这些预防措施后，由于设计或约束变化，在实现期间仍可能发生冲突。

在时钟缓冲器上使用 CLOCK_REGION 属性

您可以使用 CLOCK_REGION 约束为时钟区域分配时钟缓冲器，而不指定站点 (site)。这样即可提升布局器的灵活性，从而对所有时钟树进行优化，并判定相应的缓冲器站点以便成功完成所有时钟的布线。

您还可使用 CLOCK_REGION 约束来提供有关级联时钟缓冲器或由非时钟原语（例如，互连结构逻辑）所驱动的时钟缓冲器的布局指南。

在以下示例中，XDC 约束将向 CLOCK_REGION_X2Y2 分配 clkgen/clkout2_buf 时钟缓冲器。

```
set_property CLOCK_REGION_X2Y2 [get_cells clkgen/clkout2_buf]
```

注释：在大多数情况下，时钟缓冲器由已约束到时钟区域的输入时钟端口、MMCM、PLL 或 GT*_CHANNEL 直接驱动。在此情况下，时钟缓冲器将自动布局在相同时钟区域内，您无需使用 CLOCK_REGION 约束。

使用 Pblock 限制时钟缓冲器布局

如果无需在特定时钟区域中布局时钟缓冲器，那么即可使用 Pblock 来指定期时钟区域范围。例如，如需通过 BUFGCTRL 对位于不同区域内的 2 个时钟进行多路复用设置，可使用 Pblock。您可将 BUFGCTRL 分配给包含这 2 个时钟驱动之间的时钟区域的 Pblock，并让布局器来识别有效布局。

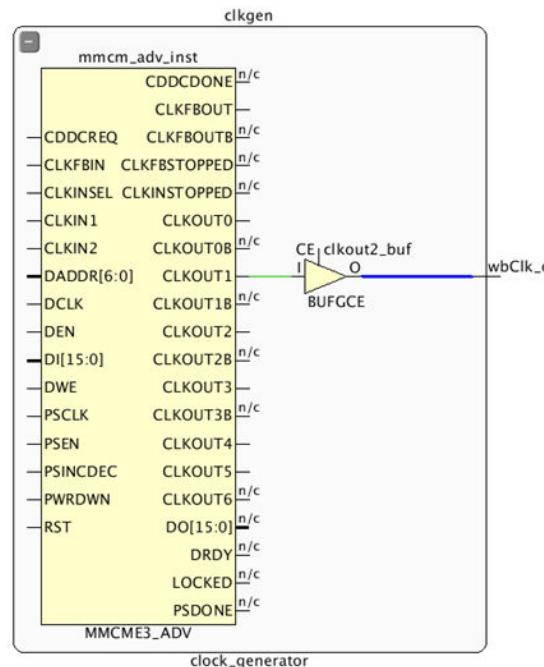
注释：AMD 不建议将 Pblock 用于单一时钟区域。

在时钟信号线上使用 USER_CLOCK_ROOT 属性

USER_CLOCK_ROOT 属性可用于强制限制时钟缓冲器所驱动的时钟的时钟根位置。指定 USER_CLOCK_ROOT 属性会影响设计布局，因为它会修改时钟布线，从而影响插入延迟和偏差。USER_CLOCK_ROOT 值对应时钟区域，并且必须在高扇出时钟缓冲器所驱动的信号线上直接设置该属性。下面给出 1 个示例：

```
set_property USER_CLOCK_ROOT X2Y3 [get_nets clkgen/wbClk_o]
```

图 56：应用于由时钟缓冲器驱动的信号线段上的 USER_CLOCK_ROOT



完成布局后即可使用 CLOCK_ROOT 属性来查询实际时钟根，如以下示例所示。CLOCK_ROOT 会报告由用户分配的根和由 Vivado 工具自动分配的根。

```
get_property CLOCK_ROOT [get_nets clkgen/wbClk_o] => X2Y3
```

复查已实现的设计的时钟根分配的另一种方法是使用 report_clock_utilization Tcl 命令。例如：

```
report_clock_utilization -clock_roots_only
```

下图显示了此报告。

图 57：report_clock_utilization 时钟根分配

Index	Clock Net	Root Clock Region
1	clkgen/clkfbout_buf	X4Y1
2	clkgen/cpuClk_o	X4Y1
3	clkgen/fftClk_o	X3Y2
4	clkgen/phyClk0_o	X3Y3
5	clkgen/phyClk1_o	X3Y2
6	clkgen/usbClk_o	X3Y3

在多个时钟信号线上使用 CLOCK_DELAY_GROUP 约束

您可使用 CLOCK_DELAY_GROUP 约束来匹配由不同时钟缓冲器驱动的多个相关时钟网络的插入延迟。此约束常用于最大限度减少源自相同 MMCM、PLL 或 GT 来源的时钟之间的同步 CDC 时序路径的偏差。您必须在直接连接到时钟缓冲器的信号线上设置 CLOCK_DELAY_GROUP 约束。以下示例显示了由时钟缓冲器直接驱动的 clk1_net 和 clk2_net 时钟信号线：

```
set_property CLOCK_DELAY_GROUP grp12 [get_nets {clk1_net clk2_net}]
```

相关信息

同步 CDC

使用 CLOCK_DEDICATED_ROUTE 约束

通常从时钟区域中某个时钟缓冲器驱动到另一个时钟区域中的 MMCM 或 PLL 时会使用 CLOCK_DEDICATED_ROUTE 约束。默认情况下，CLOCK_DEDICATED_ROUTE 约束设置为 TRUE，并且缓冲器/MMCM 或 PLL 对必须布局在相同时钟区域内。

注释：在 UltraScale 器件上针对 7 系列 CLOCK_DEDICATED_ROUTE 值使用 BACKBONE 会导致产生与 SAME_CMT_COLUMN 相同的行为。

下表汇总了不同 CLOCK_DEDICATED_ROUTE 约束值、使用方式和行为。

注释：仅在全局时钟缓冲器的信号线上应用 ANY_CMT_COLUMN 约束和 SAME_CMT_COLUMN 约束。

表 3：UltraScale 器件 CLOCK_DEDICATED_ROUTE 约束汇总

值	用途	行为
TRUE	时钟信号线上的默认值	全局时钟缓冲器和 MMCM/PLL 必须位于相同的时钟。 该值用于确保仅使用全局时钟资源对信号线进行布线。
SAME_CMT_COLUMN (BACKBONE)	全局时钟缓冲器驱动的信号线 示例： <pre>set_property CLOCK_DEDICATED_ROUTE SAME_CMT_COLUMN \ [get_nets -of [get_pins BUFGCE_inst/O]]</pre>	MMCM/PLL 必须居于相同垂直列中的时钟中。 该值用于确保仅使用全局时钟资源对信号线进行布线。 为实现最佳结果，AMD 建议在 MMCM/PLL 上使用 LOC 约束控制相同垂直列中 MMCM/PLL 的布局。
ANY_CMT_COLUMN	全局时钟缓冲器驱动的信号线 示例： <pre>set_property CLOCK_DEDICATED_ROUTE ANY_CMT_COLUMN \ [get_nets -of [get_pins BUFGCE_inst/O]] set_property CLOCK_DEDICATED_ROUTE ANY_CMT_COLUMN \ [get_nets -of [get_pins BUFGCE_DIV_inst/O]] set_property CLOCK_DEDICATED_ROUTE ANY_CMT_COLUMN \ [get_nets -of [get_pins BUFGCTRL_inst/O]]</pre>	MMCM/PLL 可布局在任何有可用资源的时钟中。 该值用于确保仅使用全局时钟资源对信号线进行布线。 为实现最佳结果，AMD 建议在 MMCM/PLL 上使用 LOC 约束控制器件内 MMCM/PLL 的布局。

表 3：UltraScale 器件 CLOCK_DEDICATED_ROUTE 约束汇总 (续)

值	用途	行为
FALSE	<p>时钟信号线不受全局时钟缓冲器驱动，但受时钟信号线的一部分驱动（例如，受 IBUF 输出驱动的信号线，或直接连接到 MMCM 的输出时钟管脚的信号线）</p> <p>示例：</p> <pre>set_property CLOCK_DEDICATED_ROUTE FALSE \ [get_nets -of [get_pins MMCME4_ADV_inst/ CLKOUT0]] set_property CLOCK_DEDICATED_ROUTE FALSE \ [get_nets -of [get_pins IBUF_inst/O]]</pre>	<p>信号线是使用互连结构和全局时钟资源完成布线的。这会对时钟网络的时序和性能产生负面影响。</p> <p>重要提示！对于 UltraScale 器件，仅当正常情况下使用全局时钟资源布线的时钟因特殊设计原因而需采用互连结构资源进行布线时，才能使用 FALSE 值。</p>

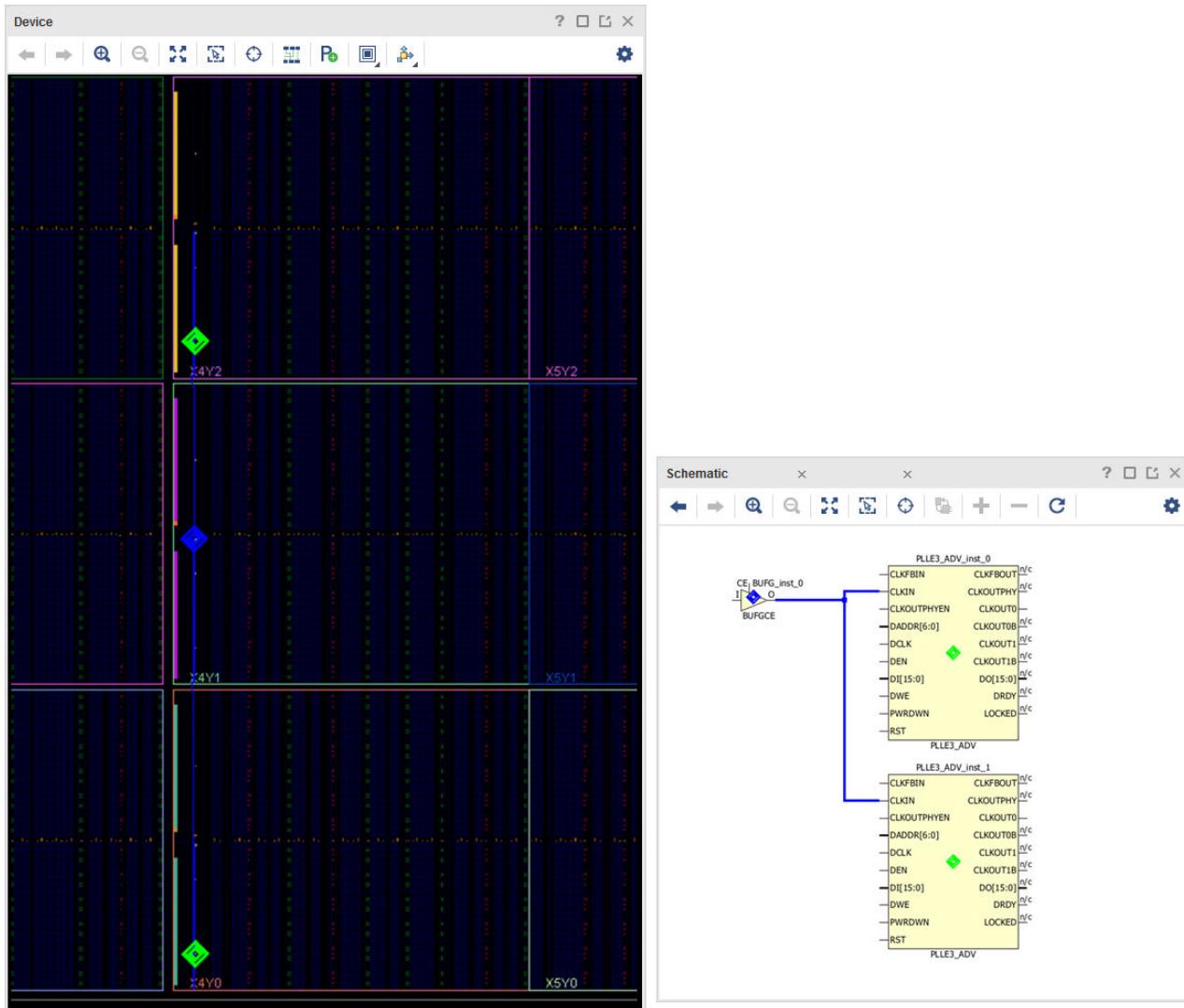
注释：处理 UltraScale 器件时，请勿将 CLOCK_DEDICATED_ROUTE 属性应用于端口直接驱动的信号线。而是改为将 CLOCK_DEDICATED_ROUTE 属性应用于 IBUF 的输出。

垂直相邻的时钟区域的约束示例

从某个时钟区域中的时钟缓冲器驱动到垂直相邻的时钟区域中的 MMCM 或 PLL 时，将 CLOCK_DEDICATED_ROUTE 设置为 BACKBONE（针对 7 系列器件）或设置为 SAME_CMT_COLUMN（针对 UltraScale 器件）即可实现最佳结果。这样可防止出现实现错误，并确保仅使用全局时钟资源对时钟进行布线。在某些情况下，布局器可以在垂直相邻时钟区域内合规布局不超过 2 个 MMCM 或 PLL，而无需设置 CLOCK_DEDICATED_ROUTE 约束。如果布局器能找到合规的解决方案以在垂直相邻时钟区域内布局 MMCM 或 PLL 而无需设置 CLOCK_DEDICATED_ROUTE 约束，那么找到的解决方案对于您的设计而言可能只是次优解决方案。以下示例和图示显示的中央时钟缓冲器在其上方或下方垂直相邻时钟区域内驱动 2 个 PLL。

```
set_property CLOCK_DEDICATED_ROUTE SAME_CMT_COLUMN [get_nets -of [get_pins
BUFG_inst_0/O]]
set_property LOC PLLE3_ADV_X0Y0 [get_cells PLLE3_ADV_inst_0]
set_property LOC PLLE3_ADV_X0Y4 [get_cells PLLE3_ADV_inst_1]
```

图58：CLOCK_DEDICATED_ROUTE 约束设置为 SAME_CMT_COLUMN

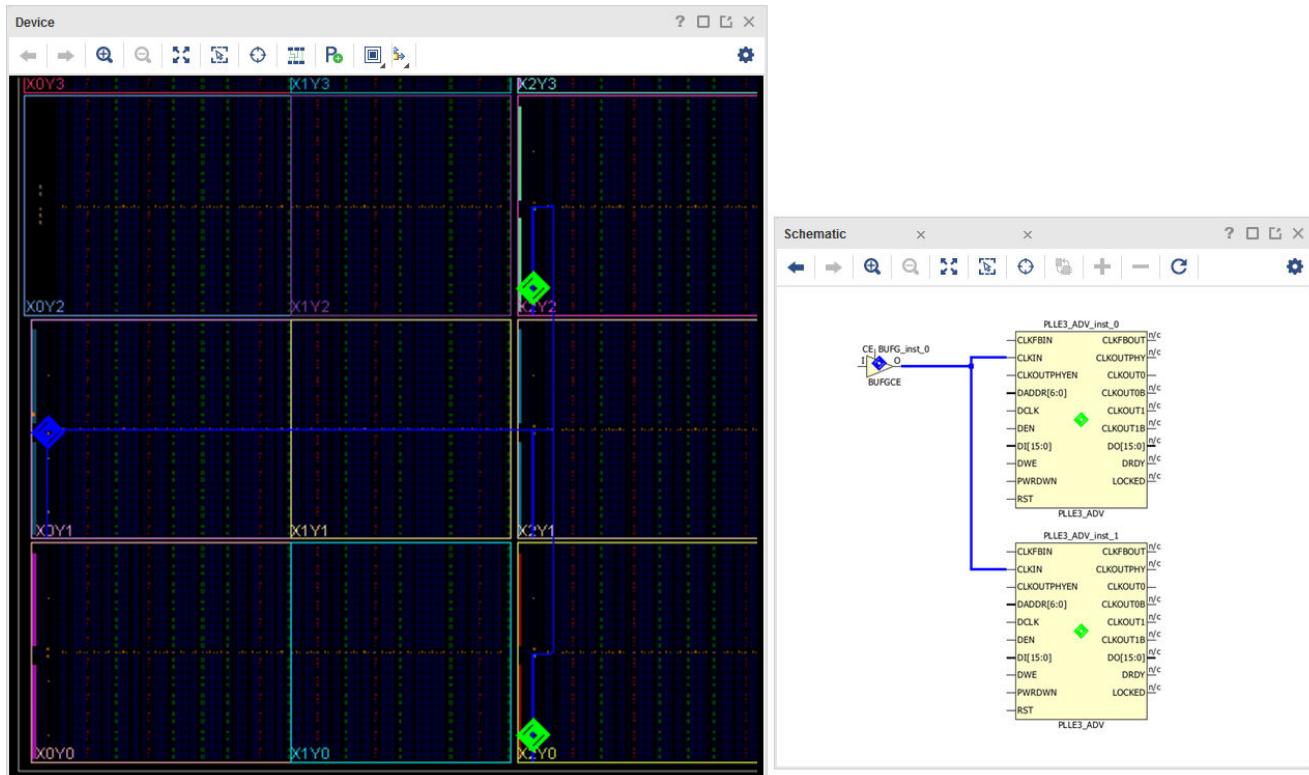


非垂直相邻时钟区域的约束示例

从时钟缓冲器驱动到另一个非垂直相邻的时钟区域时，必须将 CLOCK_DEDICATED_ROUTE 设置为 FALSE（针对 7 系列器件）或设置为 ANY_CMT_COLUMN（针对 UltraScale 器件）。这样可防止出现实现错误，并确保仅使用全局时钟资源对时钟进行布线。以下示例和图示显示了驱动与输入缓冲器不在同一个时钟区域列中的 2 个 PLL 的 BUFGCE。

```
set_property CLOCK_DEDICATED_ROUTE ANY_CMT_COLUMN [get_nets -of [get_pins BUFG_inst_0/O]]
set_property LOC PLLE3_ADV_X1Y0 [get_cells PLLE3_ADV_inst_0]
set_property LOC PLLE3_ADV_X1Y4 [get_cells PLLE3_ADV_inst_1]
```

图59：CLOCK_DEDICATED_ROUTE 设置为 ANY_CMT_COLUMN



使用 CLOCK_LOW_FANOUT 约束

您可以使用 CLOCK_LOW_FANOUT 约束在单个时钟区域中包含时钟缓冲器负载。您可在全局时钟缓冲器直接驱动的时钟信号线上或者在一系列触发器上设置 CLOCK_LOW_FANOUT 约束。

注释：配合其他时钟使用时，CLOCK_LOW_FANOUT 约束优先级较低。如果 CLOCK_LOW_FANOUT 与其他时钟约束（如 USER_CLOCK_ROOT、CLOCK_DELAY_GROUP 或 CLOCK_DEDICATED_ROUTE）存在冲突，那么将不会遵循 CLOCK_LOW_FANOUT。

触发器的约束示例

在全局时钟缓冲器驱动的一系列触发器上设置 CLOCK_LOW_FANOUT 约束将导致 opt_design 创建新的并行全局时钟缓冲器，以隔离这些触发器。在 place_design 期间，由新创建的并行全局时钟缓冲器驱动并已隔离的触发器将包含到单个时钟区域中。

以下示例显示了适用于一系列触发器的 CLOCK_LOW_FANOUT 约束，这些触发器在时钟门控同步电路中用于控制全局时钟缓冲器的时钟使能。

```
set_property CLOCK_LOW_FANOUT TRUE [get_cells safeClockStartup_reg[*]]
```

在设计中，不间断的时钟网络最初驱动 2000 余个负载，包括在时钟门控同步电路中用于对其他逻辑进行时钟门控的触发器。以下板级原理图显示了在 opt_design 创建新的并行全局时钟缓冲器用于隔离时钟门控同步电路前后，时钟门控同步电路的不同状态以及连接到不间断时钟网络的其他逻辑。

图 60：执行 opt_design 变换并对触发器应用 CLOCK_LOW_FANOUT 之前的板级原理图

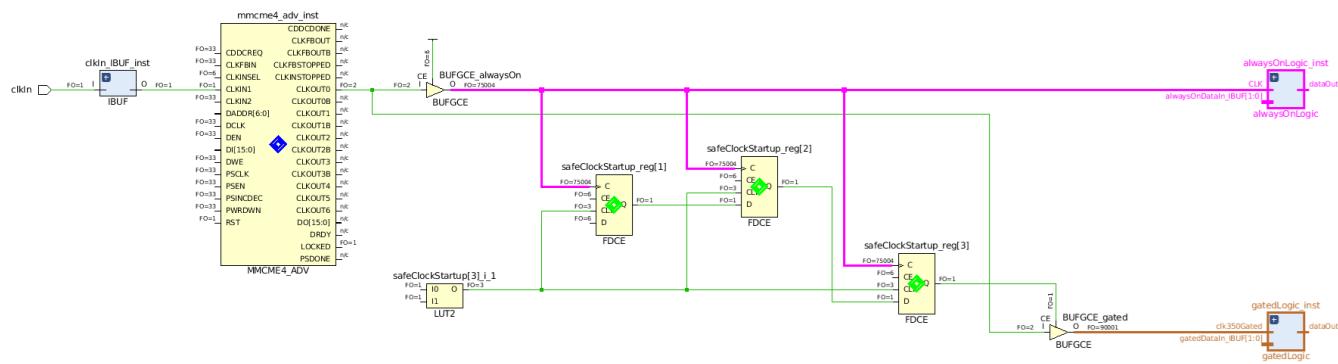
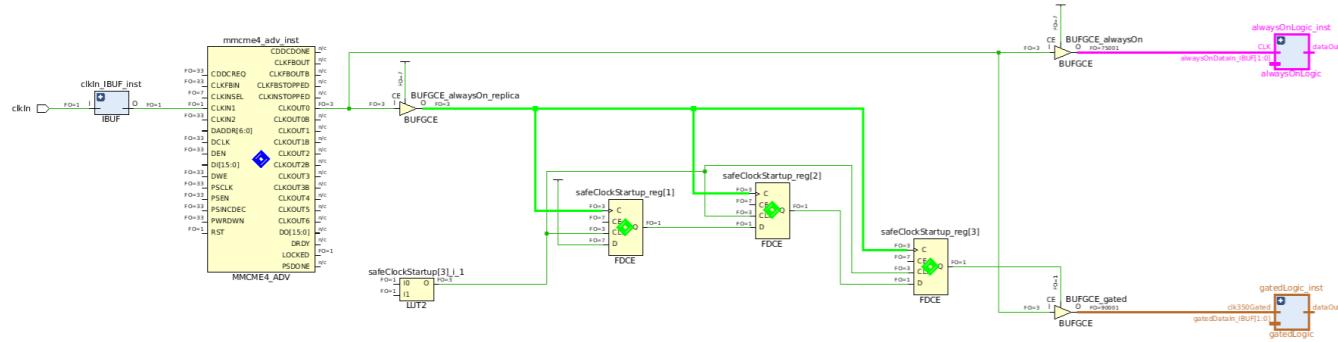
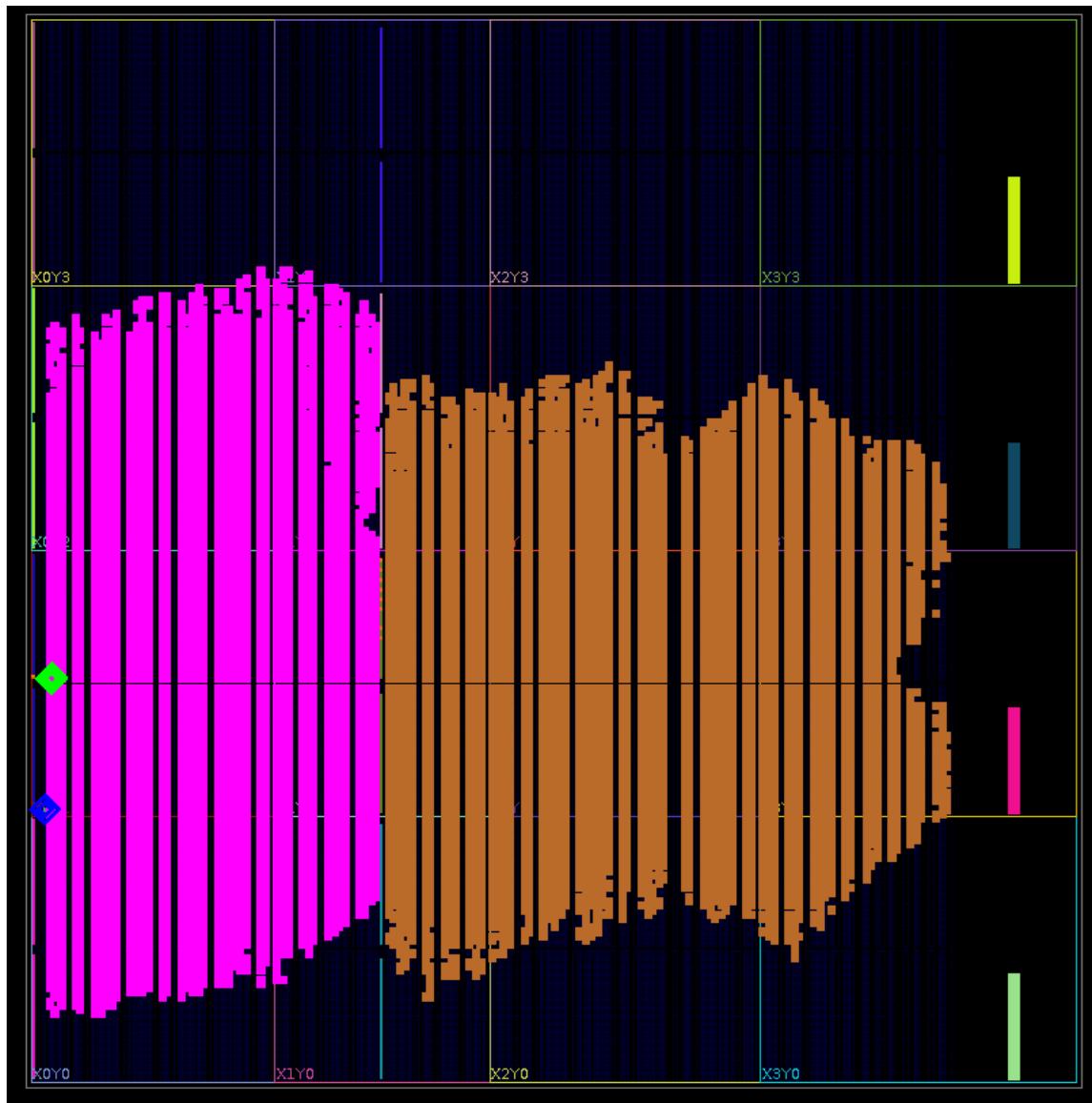


图 61：执行 opt_design 变换并对触发器应用 CLOCK_LOW_FANOUT 之后的板级原理图



完全实现的设计的“Device”（器件）窗口会显示时钟门控同步电路，其中不间断逻辑和时钟门控逻辑带有绿色标记。时钟门控同步电路与 MMCM 布局在相同 CLOCK_REGION 内，位于全局时钟缓冲器旁。

图 62：完全实现的设计（含时钟门控同步电路布局）



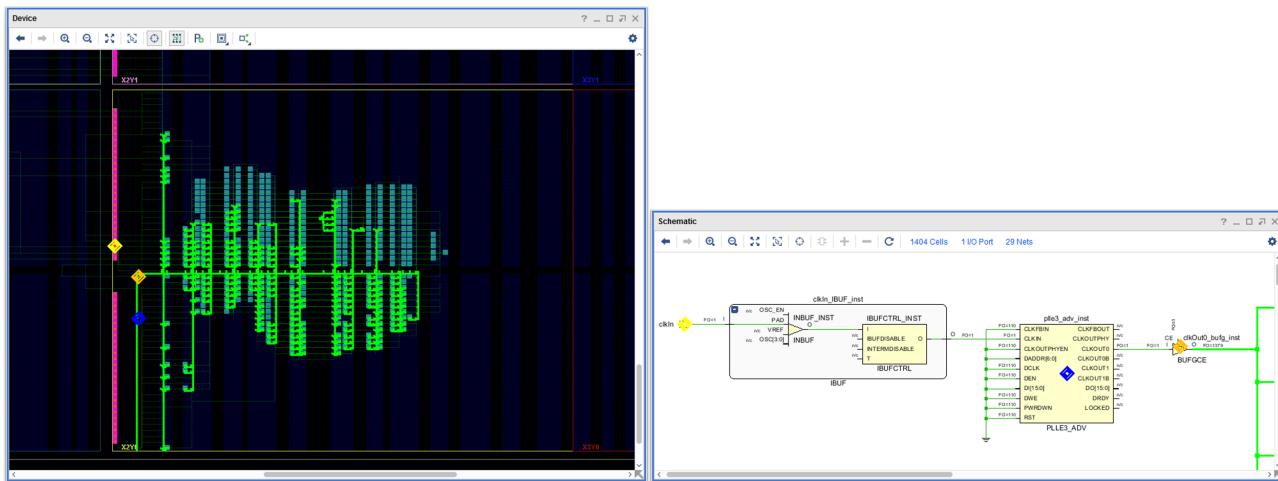
时钟信号线约束示例

如果在全局时钟缓冲器直接驱动的时钟信号线上设置 CLOCK_LOW_FANOUT 属性，并且全局时钟缓冲器的扇出的负载数不足 2000，那么负载布局将包含在单个时钟区域内。

以下示例显示了适用于全局时钟缓冲器直接驱动的时钟信号线段的 CLOCK_LOW_FANOUT 约束。该时钟网络驱动的负载数不足 2000，并包含在单个时钟区域内。输入时钟端口 clkIn 不仅针对位于 CLOCK_REGION X2Y0 的 GCIO 提供 PACKAGE_PIN 分配，而且还可驱动 PLL3_ADV。PLL3_ADV 可驱动全局时钟缓冲器，后者继而驱动含 1379 个负载的时钟网络。全局时钟缓冲器的负载均布局在 CLOCK_REGION X2Y0 内。

```
# PACKAGE_PIN AF9 - IOBank 64 - CLOCK_REGION X2Y0
set_property PACKAGE_PIN AF9 [get_ports clkIn]
set_property IOSTANDARD LVCMS18 [get_ports clkIn]
set_property CLOCK_LOW_FANOUT TRUE [get_nets -of [get_pins
clkOut0_bufg_inst/O]]
```

图63：“Device”窗口与“Schematic”窗口中的CLOCK_LOW_FANOUT示例



时钟拓扑建议

AMD 建议使用简单的时钟树拓扑结构，因其设计所需的时钟缓冲器数量最少。使用额外的时钟缓冲器需要更多的布线轨道，这可能导致时钟区域内因时钟布线要求过高且接近最大容量而出现布局错误或者布线冲突。

以下时钟拓扑结构建议适用于 BUFGCE/BUFGCTRL/BUFGCE_DIV 连接。

并行时钟缓冲器

使用并行时钟缓冲器来实现以下目的：

- 确保跨实现运行的布局可预测。

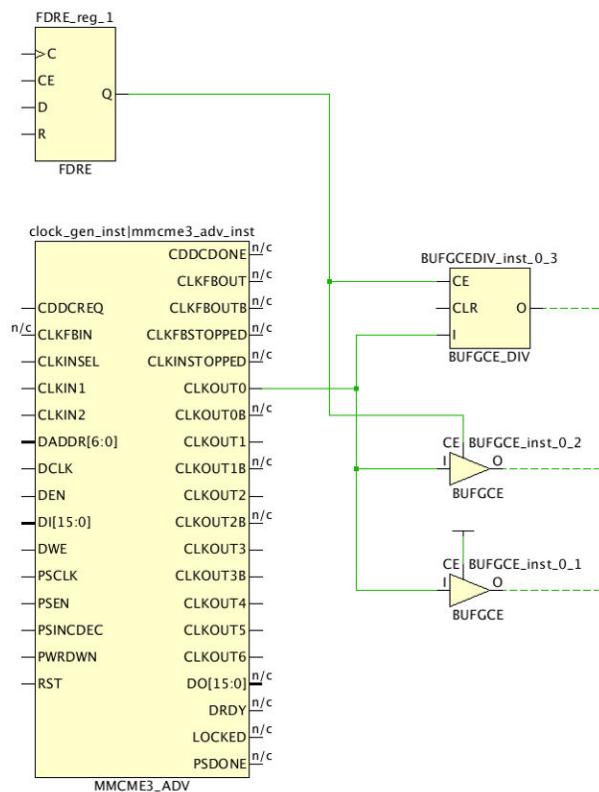
当并行时钟缓冲器由相同输入时钟端口（MMCM、PLL 或 GT*_CHANNEL）直接驱动时，缓冲器始终置于与其驱动相同的时钟区域内，与网表变更或逻辑布局变化无关。

- 匹配时钟树的并行分支之间的插入延迟。

AMD 推荐使用并行缓冲器替代级联时钟缓冲器，在分支之间存在同步路径时尤其如此。当使用级联缓冲器时，时钟树的分支之间的时钟插入延迟将不匹配，即使使用 CLOCK_DELAY_GROUP 或 USER_CLOCK_ROOT 约束也是如此。这可能导致高时钟偏差，从而导致时序收敛难度增加，甚至无法实现。

下图显示了由 MMCM CLKOUT0 端口驱动的 3 个并行 BUFGCE 缓冲器。

图 64：MMCM 输出上的并行 BUFGCE



级联时钟缓冲器

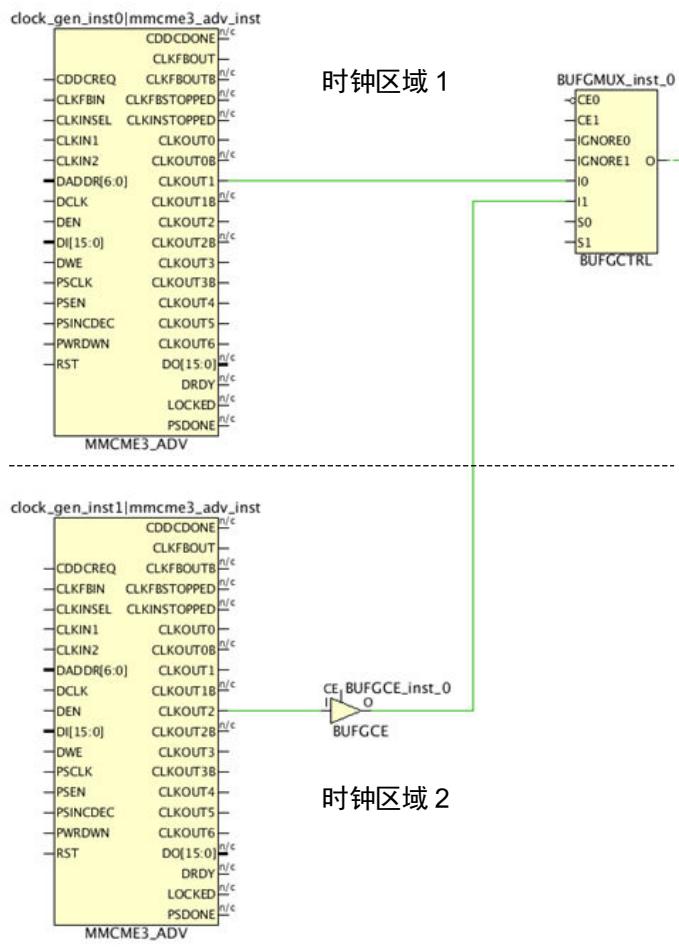
一般 AMD 不建议使用级联缓冲器在不相关的时钟树分支之间人为增加延迟并减少偏差。不同于 BUFGCTRL 之间的连接，其他时钟缓冲器连接在架构中不具有专用路径。因此，时钟缓冲器的相对布局不可预测，并且所有布局规则都优先于未约束的级联缓冲器的布局。

但是，使用级联时钟缓冲器可以实现以下功能：

- 将时钟布线到位于不同时钟区域中的另一个时钟缓冲器。

对由其他时钟区域内的 MMCM 生成的时钟使用时钟多路复用器时，通常采用此方法。虽然其中 1 个 MMCM 可以直接驱动 BUFGCTRL (BUFGMUX)，但是另一个 MMCM 需要中间时钟缓冲器来将时钟信号布线到其他区域。下图显示了 1 个示例。

图 65：将时钟布线到另一个时钟区域



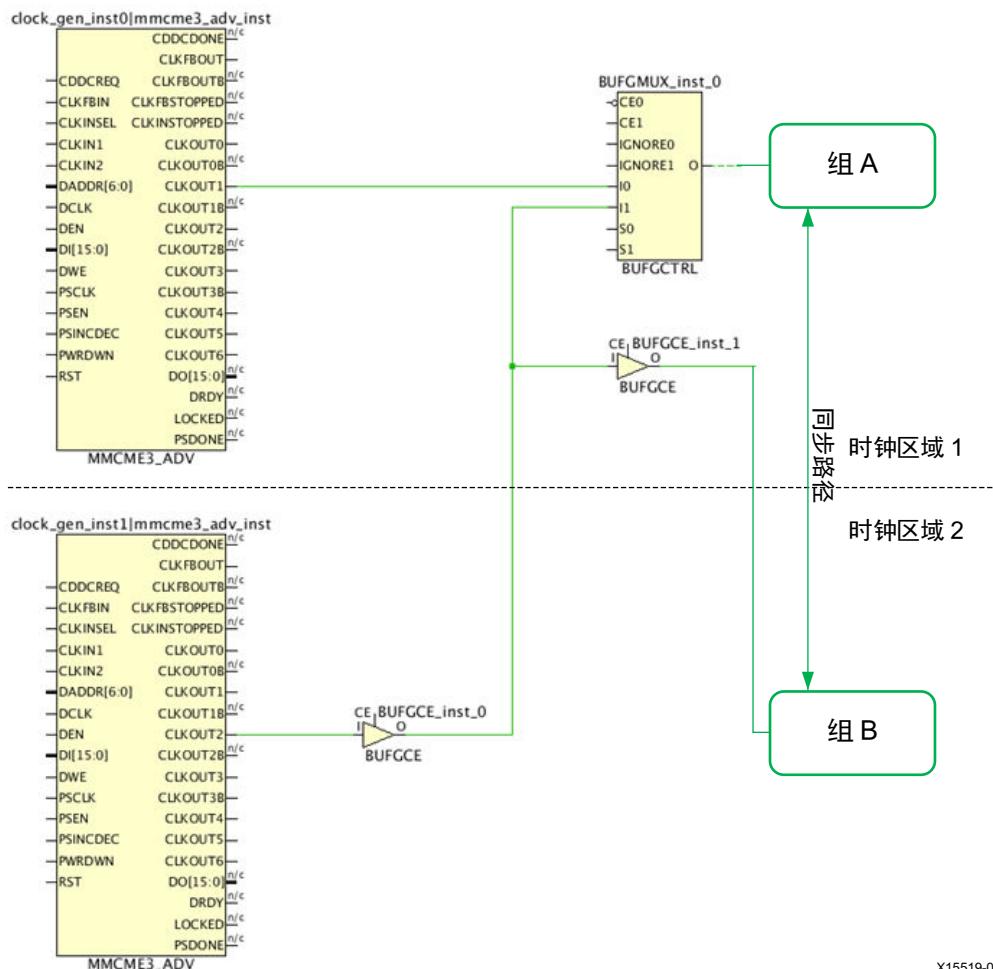
X15518-052822

- 当时钟树分支之间存在同步路径时，平衡这些分支之间的时钟缓冲器等级数。

例如，假设 1 个名为 clk0 的 MMCM 时钟用于驱动组 A（通过位于不同时钟区域内的 BUFGCTRL 驱动的时序单元）和组 B（时序单元）。为了更好地匹配分支之间的延迟，为组 B 插入 1 个 BUFGECE，并将它布局在与 BUFGCTRL 相同的时钟区域中。这样可确保组 A 和组 B 之间的同步路径偏差可控。下图显示了 1 个示例。

注释：Vivado 逻辑最优化命令 opt_design 未发现时序时钟和时钟网络分支之间的时序关系。因此，opt_design 会尽可能删除更多级联时钟缓冲器或冗余时钟缓冲器。在此示例中，opt_design 会移除 BUFGECE_inst_1，除非您在其中设置 DONT_TOUCH= "TRUE" 属性。如果在时钟树分支之间只有异步路径，那么只要接收端时钟域上存在正常的同步电路，就不需要平衡这些分支。

图66：为时钟区域之间的同步路径平衡时钟树



X15519-052822

- 如**时钟多路复用**所述，构建时钟多路复用器。

为减少插入延迟和偏差变动，AMD 建议使用级联时钟缓冲器时遵循以下准则：

- 将级联缓冲器保持在相同或相邻时钟区域内。
- 平衡时钟树分支时，将相同时钟缓冲器都分配到相同时钟区域内。

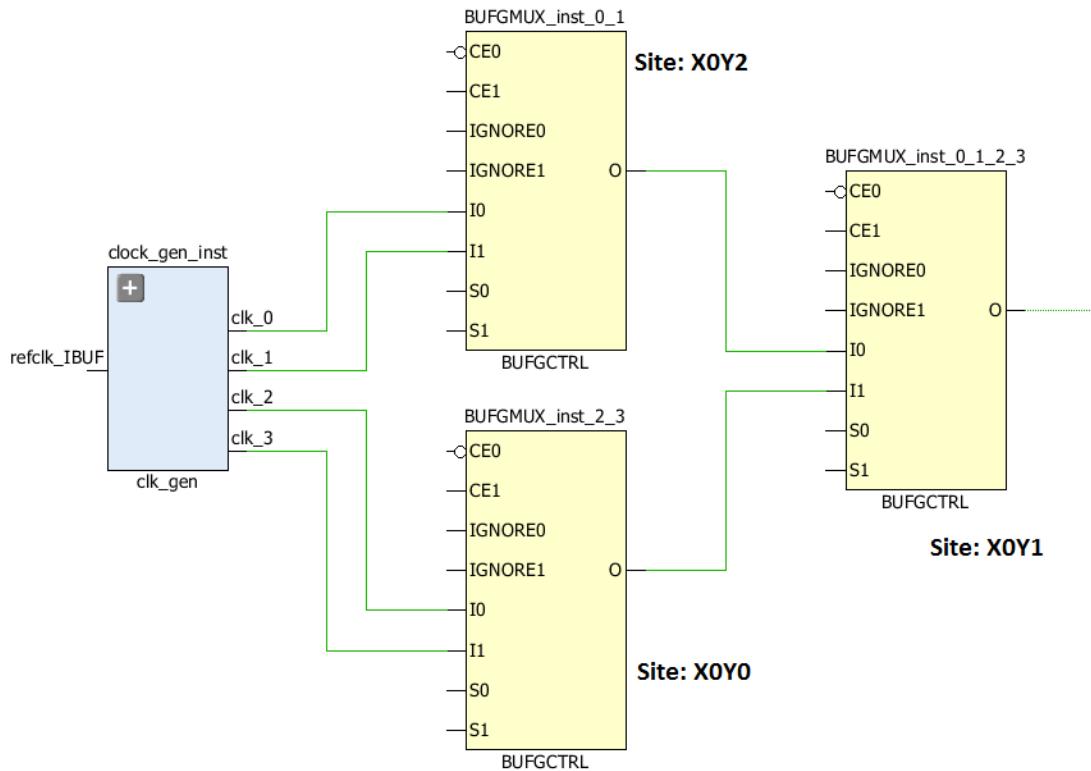
注释：如果确实有必要，AMD 建议使用 2 个级联 BUFGCTRL 代替级联 BUFGECE。使用专用布线即可将 2 个相邻 BUFGCTRL 加以级联，当这 2 个 BUFGCTRL 都布局在相同时钟区域内时可最大程度降低延迟。

时钟多路复用

您可以使用并行和级联 BUFGCTRL 的组合构建时钟多路复用器。布局器基于时钟缓冲器站点 (site) 可用性查找最佳布局。如果可能，布局器将 BUFGCTRL 布局在相邻站点 (site) 中以便充分利用专用级联路径。如无法实现，则布局器将尝试在相邻时钟区域的相同层级中布局 BUFGCTRL。

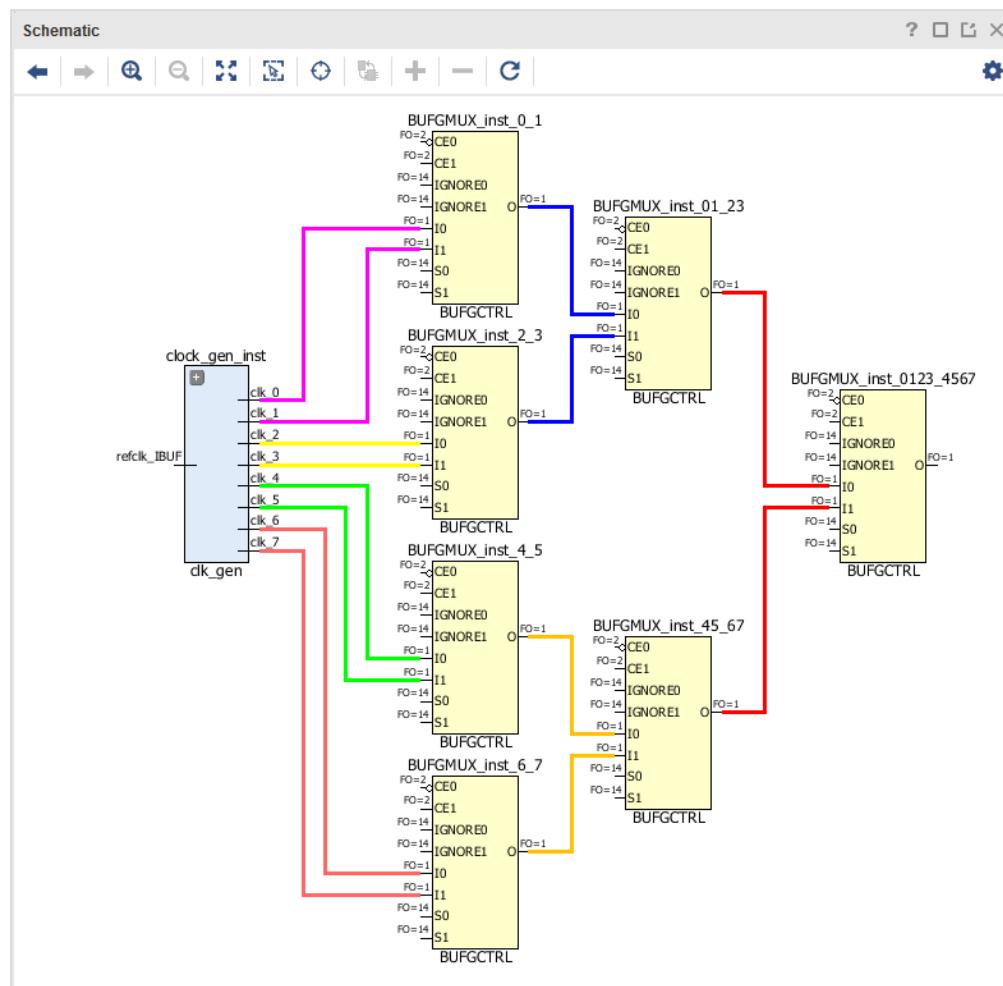
下图显示了具有平衡级联的 4:1 MUX。第 1 级 BUFGCTRL 缓冲器都布局在最后一个 BUFGCTRL (XOY1) 的直接相邻站点 (site) (XOY2, XOY0)。此配置确保到达最后一个 BUFGCTRL 的所有时钟的插入延迟相近。对于 3:1 MUX，可以使用类似结构。

图 67：使用并行 BUFGCTRL 的 4:1 MUX



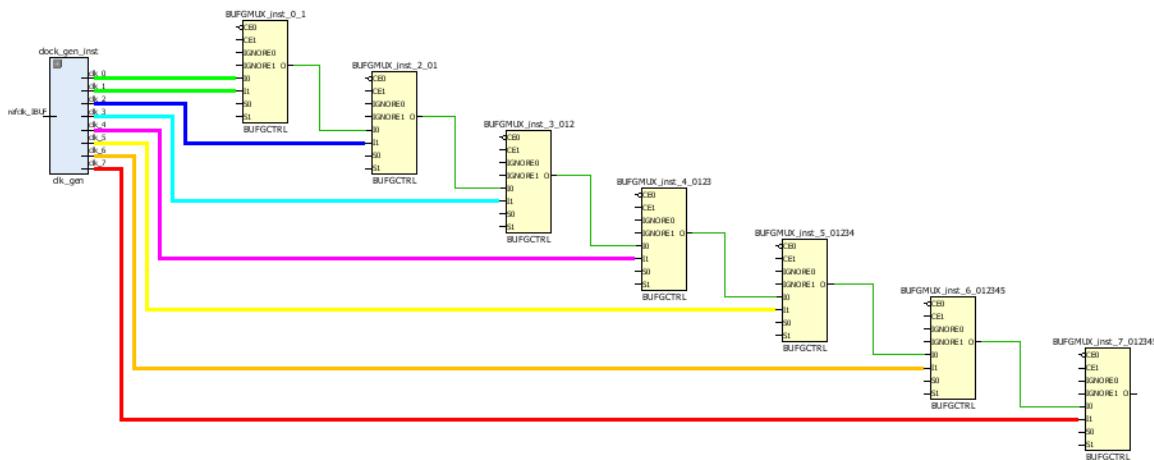
如下图所示，当创建 5:1 或更大的时钟 MUX 结构时，通常会创建 1 个对称的时钟结构。然而这只是次优解决方案，因为每个 BUFGCTRL 只有 1 条到 2 个相邻 BUFGCTRL 的级联路径，这无法为 BUFGCTRL 之间的所有连接提供最小延迟。

图 68：非推荐的 8:1 平衡时钟 MUX 结构



为支持更大的时钟多路复用器（从 5:1 到 8:1 MUX），AMD 建议使用级联 BUFCTRL 缓冲器，如下图所示。此图显示了使用 7 个 BUFCTRL 缓冲器的最优化 8:1 MUX。

图 69：使用级联 BUFGCTRL 的 8:1 MUX



注释：使用基于较宽的 BUFGCTRL 的时钟多路复用器时，无法平衡时钟插入延迟，因为硬件中部分路径比其他路径更长。因此，建议仅对异步时钟多路复用采用此方法。

PLL/MMCM 反馈路径和补偿模式

PLL 不支持延迟补偿，并且时钟在 INTERNAL 补偿模式下工作，即，无需反馈路径。同样，设置为 INTERNAL 补偿模式的 MMCM 也无需反馈路径。在上述这两种情况下，Vivado 工具并不总能自动移除不必要的反馈时钟缓冲器。您必须手动移除时钟缓冲器以降低高扇出时钟资源使用率。这对于因时钟使用率较高而可能导致时钟争用的设计尤为重要。

当 MMCM 补偿设置为 ZHOLD 或 BUF_IN 时，布局器会向反馈缓冲器和直接连接到 CLKOUT0 管脚的所有缓冲器驱动的信号线分配相同的时钟根。这样即可确保插入延迟相匹配，以便使 I/O 端口和连接到 CLKOUT0 的时序单元相位对齐，并且在器件接口上满足保持时间要求。Vivado 工具会考量这些信号线的所有负载，以便按最优化方式定义时钟根。

Vivado 工具不会自动将插入延迟与其他 MMCM 输出相匹配。要为由其他 MMCM 输出缓冲器驱动的信号线匹配插入延迟，请使用以下属性：

- `CLOCK_DELAY_GROUP`

根据需要，针对由反馈时钟缓冲器、CLKOUT0 缓冲器和其他 MMCM 输出缓冲器直接驱动的信号线应用相同的 `CLOCK_DELAY_GROUP` 属性值。这是首选的方法。

- `USER_CLOCK_ROOT`

如需强制指定特定时钟根，请根据需要针对由反馈时钟缓冲器、CLKOUT0 缓冲器和其他 MMCM 输出缓冲器驱动的信号线使用相同的 `USER_CLOCK_ROOT` 属性值。

BUFG_GT 分频器

BUFG_GT 缓冲器可驱动互连结构中的任意负载，并且包含可选分频器，可供您用于对来自 `GT*_CHANNEL` 的时钟进行分频。这样就不再需要使用额外的 MMCM 或 `BUFG_DIV` 来对时钟进行分频。

SelectIO 时钟设置

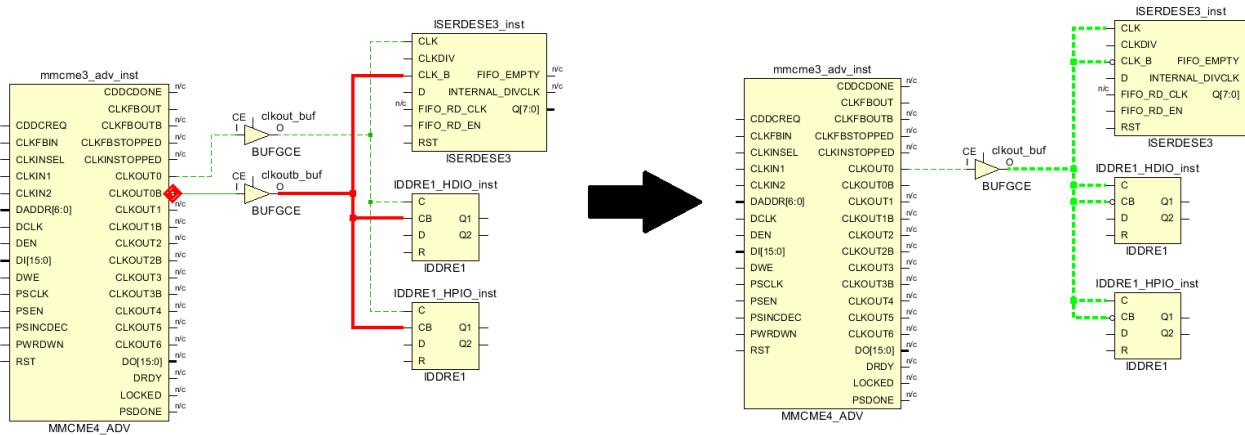
UltraScale 器件 SelectIO 原语对于时钟管脚之间的最大偏差存在要求。对 SelectIO 原语使用最优化时钟设置拓扑可防止出现最大偏差违例、改进 UltraScale 器件与互连结构逻辑之间的接口时序，并减少使用的时钟资源。

ISERDESE3 和 IDDRE1 时钟设置

对于 UltraScale 和 UltraScale+ 器件中的 ISERDESE3 和 IDDRE1 时钟设置，在时钟和反相时钟管脚之间存在最大偏差要求。为满足最大偏差要求，AMD 建议在使用局部反转时，对时钟和反相时钟管脚使用单一信号线。

在下图中，左侧显示的是使用 MMCM 的 CLKOUT0B 输出的次优配置。图右侧显示的是在 ISERDESE3 和 IDDRE1 的 CLK_B 和 CB 管脚上使用局部反转的理想配置。使用最优化配置可以保证使用较少的全局时钟资源时能够满足最大偏差要求。

图 70：针对 ISERDESE3 和 IDDRE1 的次优到最优时钟拓扑



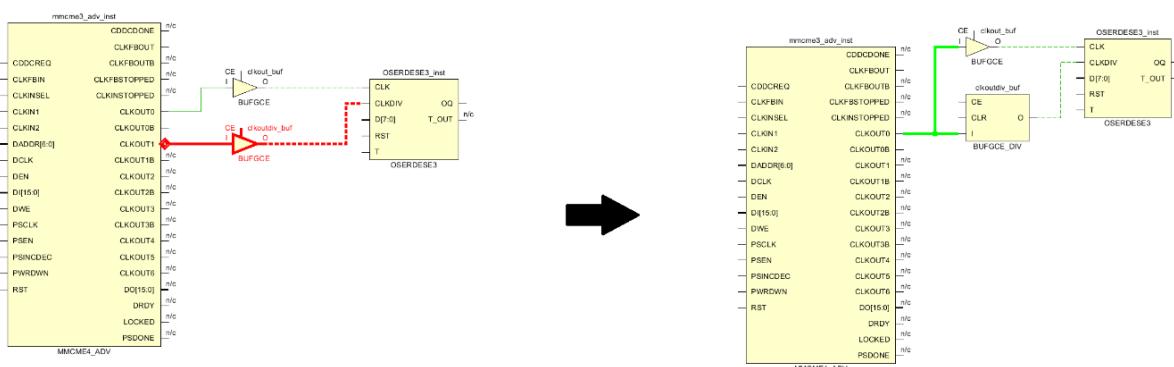
OSERDESE3 时钟设置

对于 UltraScale 和 UltraScale+ 器件中的 OSERDESE3 时钟，在高速时钟与分频时钟管脚之间存在最大偏差要求。为满足最大偏差要求，AMD 建议使用并行全局时钟缓冲器，并且其中一个全局时钟缓冲器为 BUFGCE_DIV。这样可消除 MMCM 的两项输入之间的附加时钟不确定性。

在下图中，左侧显示的是次优配置，此配置使用 2 个独立的 MMCM 输出。图右侧显示的是最优化配置，此配置使用单一 MMCM 输出和 BUFGCE_DIV 单元，并且使用 BUFGCE_DIVIDE 属性提供分频时钟。

注释：高速时钟无需使用 BUFGCE 驱动。您可改为使用 BUFGCE_DIV，其中 BUFGCE_DIVIDE 属性设置为 1。

图 71：针对 OSERDESE3 的次优到最优时钟拓扑



含 MMCM_ZHOLD/BUF_IN 补偿的 I/O 时序

ZHOLD 补偿表示 MMCM 配置为针对整个 I/O 列中的所有 I/O 寄存器提供负保持时间。当支持时钟功能的 I/O (CCIO) 驱动 ZHOLD 补偿模式中配置的单个 MMCM 时，布局器将尝试把 MMCM 与 CCIO 一起布局在相同时钟区域内。在此情况下，CCIO 可直接驱动 MMCM，无需通过 BUFG。这样即可支持 MMCM 的 ZHOLD 补偿保持有效。

但如果 CCIO 同时驱动 ZHOLD 模式下配置的 MMCM 以及另一个 MMCM，那么逻辑最优化将尝试通过在 CCIO 之后插入 BUFG 来使进入 MMCM 的时钟布线合规。由于具有 ZHOLD 补偿的 MMCM 不再由 CCIO 直接驱动，补偿将改为 BUF_IN。为避免出现此情况，请确保 CCIO 直接驱动 ZHOLD 模式下配置的 MMCM，并通过 BUFG 驱动另一个 MMCM。此外，请将 BUFG 驱动的信号线的 CLOCK_DEDICATED_ROUTE 属性设置为 ANY_CMT_COLUMN。

由于时钟插入延迟因时钟根位置而异，而时钟根布局取决于负载的布局，因此不同运行轮次之间可能存在变化。这种变化会影响器件中的时序以及 I/O 时序。

处理高频率 I/O 时，可能需要细化对 I/O 时序的控制并减少不同运行轮次之间的变化。实现此目标的方法之一是强制执行时钟根布局。您可以自动模式运行工具并观察时钟根区域。如果 I/O 时序得到满足，您可在与 I/O 时序关联的缓冲器信号线上强制时钟根布局。要确定时钟根的布局，请使用 `report_clock_utilization [-clock_roots_only]` Tcl 命令。

在以下示例中，I/O 端口位于 X0Y0 区域中。Vivado 布局器根据 I/O 布局以及其他负载的布局来判定 X1Y2 中时钟根的布局。

图 72：含未约束的时钟根的时钟使用情况汇总

Index	Clock Net	Root Clock Region	Clock Root Node
1	mmcm/inst/clk0	X1Y2	RCLK_BRAM_L_X30Y209/CLK_VDISTR_BOT0
2	mmcm/inst/clkfbout_buf_mmcm_zhold	X1Y2	RCLK_CLEL_R_L_X25Y209/CLK_VDISTR_BOT1

以下汇总显示了当时钟根不受约束时的 I/O 约束。

图 73：含未约束的时钟根的时序汇总

Setup	Hold
Worst Negative Slack (WNS): -0.279 ns	Worst Hold Slack (WHS): 0.036 ns
Total Negative Slack (TNS): -5.394 ns	Total Hold Slack (THS): 0.000 ns
Number of Failing Endpoints: 47	Number of Failing Endpoints: 0

在以下示例中，时钟根移至 X0Y0 中的 I/O 寄存器旁，导致降低时钟插入延迟并减少时序消极因素，从而改进 I/O 时序。

图 74：含用户约束的时钟根的时钟使用情况汇总

Index	Clock Net	Root Clock Region	Clock Root Node
1	mmcm/inst/clk0	X0Y0	XIPHY_L_X0Y60/CLK_VDISTR_B0T20
2	mmcm/inst/clkfbout_buf_mmcm_zhold	X0Y0	XIPHY_L_X0Y60/CLK_VDISTR_B0T14

以下汇总显示了迁移时钟根时的 I/O 时序。

图 75：含用户约束的时钟根的时序汇总

Setup	Hold
Worst Negative Slack (WNS): -0.217 ns	Worst Hold Slack (WHS): 0.060 ns
Total Negative Slack (TNS): -1.488 ns	Total Hold Slack (THS): 0.000 ns
Number of Failing Endpoints: 16	Number of Failing Endpoints: 0

同步 CDC

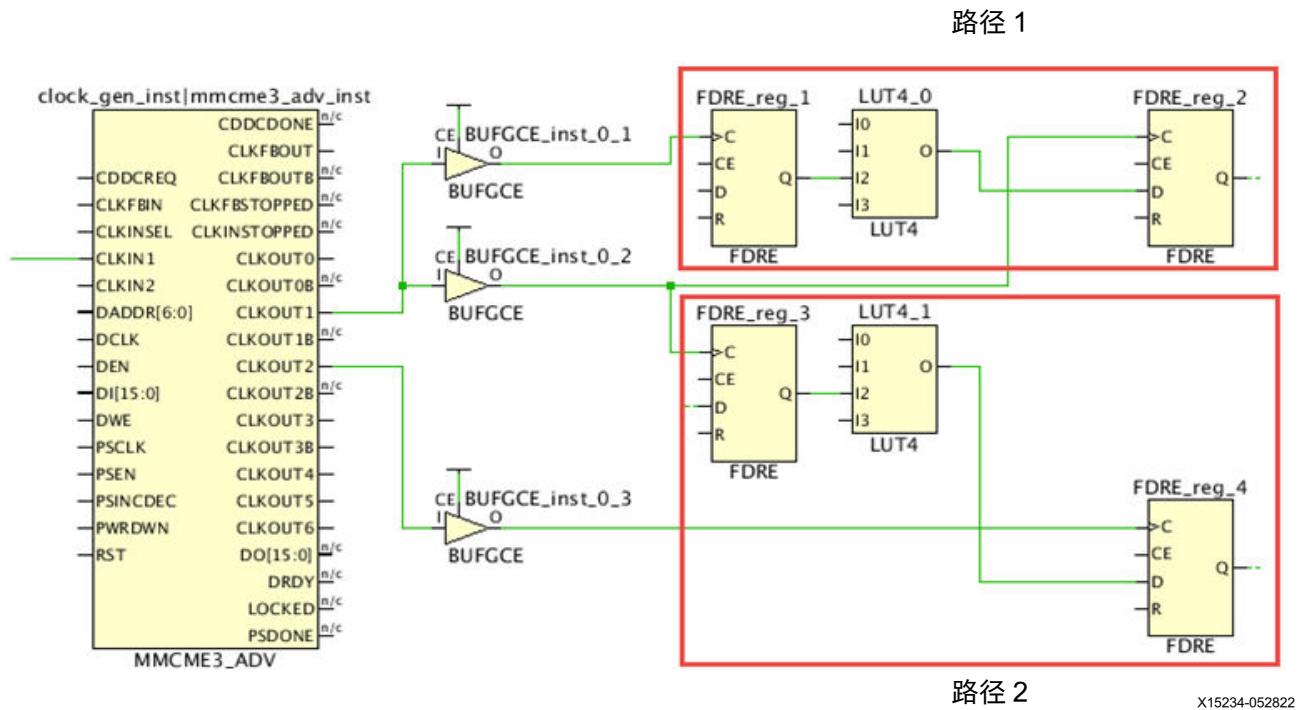
当设计的时钟之间包含同步 CDC 路径，并且这些时钟源自相同 MMCM/PLL 时，您可使用如下技巧来更好地控制时钟插入延迟和偏差，从而控制这些路径上的裕量。

重要提示！ 如果 CDC 路径位于源自不同 MMCM/PLL 的时钟之间，那么跨 MMCM/PLL 的时钟插入延迟更难以控制。在此情况下，AMD 建议您将这些时钟域交汇作为异步时钟来处理并执行相应的设计更改。

如果 2 个时钟源自相同 MMCM/PLL 的不同输出管脚，那么对这 2 个时钟之间的路径进行时序约束时，MMCM/PLL 相位误差会导致路径的时钟不确定性增加。对于使用高时钟频率的设计，相位误差可能导致建立时间和保持时间的时序收敛出现问题。

下图显示了含相位误差和不含相位误差的路径示例。路径 1 为 CDC 路径，此路径由连接到相同 MMCM 输出的 2 个缓冲器进行时钟设置，并且不含相位误差。路径 2 由源自 2 个不同 MMCM 输出的 2 个时钟进行时钟设置，并且包含相位误差。

图 76：MMCM 和相位误差



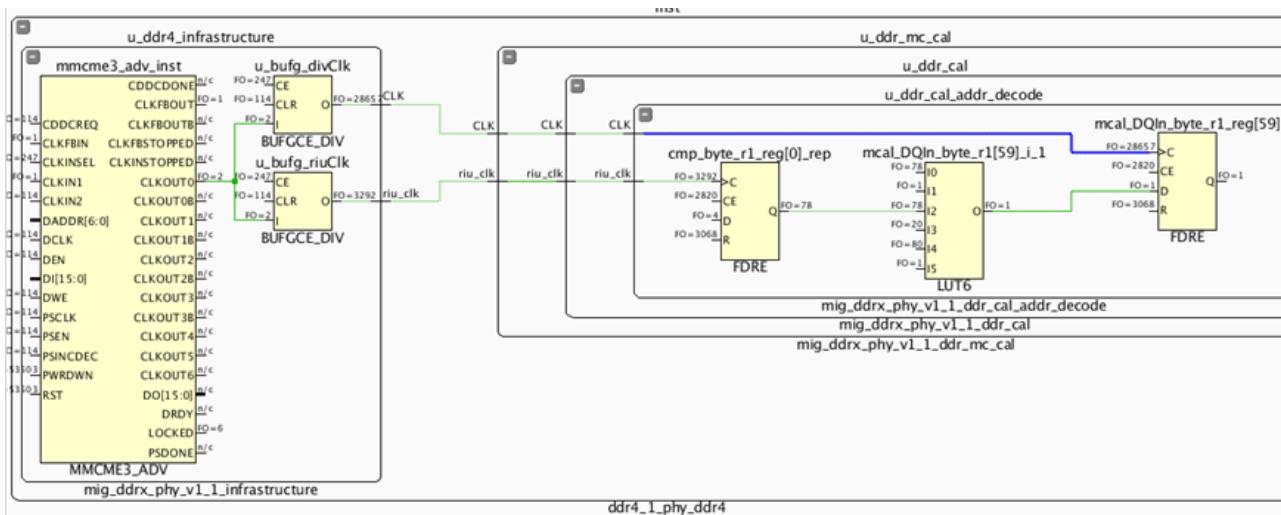
当源自相同 MMCM/PLL 的 2 个同步时钟具有简单的周期比 (/2 /4 /8) 时，可以使用连接到 2 个 BUFGCE_DIV 缓冲器的单个 MMCM/PLL 输出来防止 2 个时钟域之间出现相位误差。BUFGCE_DIV 缓冲器执行时钟分频 (/1 /2 /4 /8)。虽然可采用其他比率 (/3 /5 /6 /7)，但这需要修改时钟占空比，导致混合时钟沿时序路径变得更为困难。

注释：由于 BUFGCE 和 BUFGCE_DIV 没有相同的单元延迟，因此 AMD 建议针对 2 个同步时钟使用相同的时钟缓冲器（2 个均采用 BUFGCE 缓冲器或 2 个均采用 BUFGCE_DIV 缓冲器）。

下图显示了 2 个 BUFGCE_DIV，这 2 个缓存分别将 CLKOUT0 时钟除以 1 和 2。

重要提示！ 如果并行 BUFGCE_DIV 单元中设置的 BUFGCE_DIVIDE 属性值大于 1，那么为确保此类单元之间的时序约束安全，这 2 个缓冲器必须使用相同使能信号 (CE) 和相同复位信号 (RST)。否则，在硬件中分频后的时钟之间可能出现相移，而 Vivado 工具不会报告此现象。

图 77：MMCM 同步 CDC，其中 BUFGCE_DIV 已连接到 1 个 MMCM 输出



要在源自相同 MMCM 或 PLL 的多个时钟之间自动实现平衡，请在需平衡的时钟缓冲器所驱动的信号线上设置相同的 CLOCK_DELAY_GROUP 属性值。以下是提供的附加建议：

- 避免在过多时钟上设置 CLOCK_DELAY_GROUP 约束，因为这会给时钟布局器施压，导致出现次优解决方案或误差。
- 复查“Timing Summary Report”（时序汇总报告）中的关键同步 CDC，以判定哪些时钟必须匹配延迟以满足时序约束要求。
- 对于具有完全相同的时序拓扑结构并且时序要求较为严格的同步时钟组，请限制使用 CLOCK_DELAY_GROUP。



重要提示！ AMD 建议使用 Clocking Wizard 来创建最优化时钟结构，此结构将配合相关时钟分组约束混用 BUFGCE 和 BUFGCE_DIV。

GT 接口时钟

每个 GT 接口都需要数个时钟，包括位于 1 个或多个 GT 四通道内的绑定 GT*_CHANNEL 单元之间共享的部分时钟。UltraScale 器件可提供多达 128 个 GT*_CHANNEL 站点，这可能导致在单一设计中适用数百个时钟。大部分 GT 时钟扇出较低，负载采用局部布局方式布局在时钟区域内关联 GT*_CHANNEL 旁。部分 GT 时钟可驱动整个器件上的负载，并且需要在许多时钟区域内使用时钟布线资源。UltraScale 架构包含以下增强功能，可有效支持所需的大量 GT 时钟。

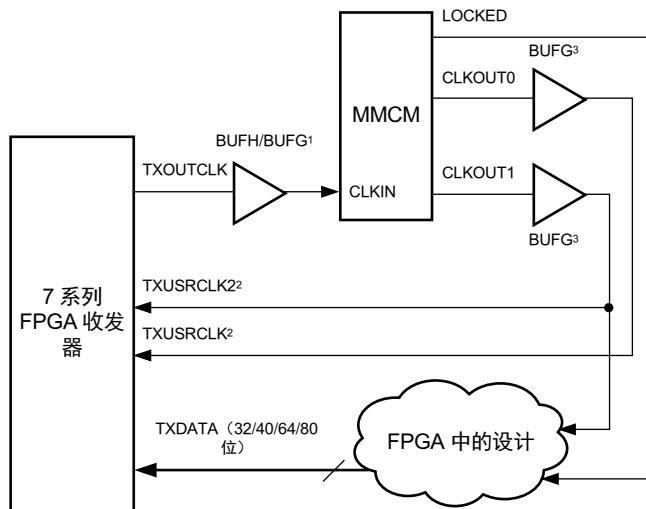
BUFG_GT 与动态分频器

在 UltraScale 器件中，BUFG_GT 缓冲器可简化 GT 时钟设置。由于 BUFG_GT 包括动态分频功能，MMCM 无需在 GT 输出时钟上再执行简单的整数除法。需要已分频的 GT*_CHANNEL 输出时钟和全速率时钟时，这样可节省时钟资源并改进低偏差时钟路径。

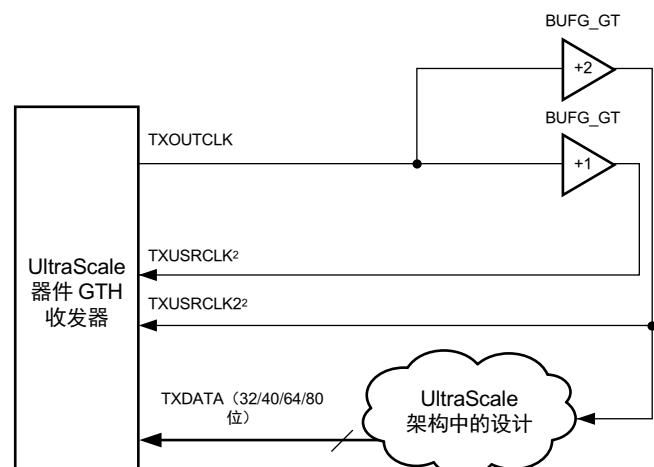
BUFG_GT 全局时钟缓冲器可用于 GT 接口（其中用户逻辑运行的时钟频率为内部 PCS 逻辑的一半）和 PCIe 接口（其中 GT*_CHANNEL 需要为 user_clk、sys_clk 和 pipe_clk 生成多个时钟频率）。下图对 7 系列和 UltraScale 器件之间单通道 GT 接口的时钟要求进行了比较，该接口中 TXUSRCLK2 的频率等于 TXUSRCLK 频率的一半。

图 78：时钟要求比较

7 系列器件（使用 MMCM 执行分频）



UltraScale 器件（使用 BUFG_GT 执行分频）



X15237-052822

您可以使用四通道 (Quad) 中的 GT*_CHANNEL 的任意输出时钟或四通道 (Quad) 中的 IBUFDS_GTE3/ODIV2 管脚生成的任何参考时钟来驱动位于同一时钟区域中的 24 个 BUFG_GT 缓冲器中的任何 1 个。必须采用 BUFG_GT_SYNC 来同步复位和清除由公用时钟源驱动的 BUFG_GT。

注释：如果设计中不含 BUFG_GT_SYNC 原语，Vivado 工具将自动插入该原语。

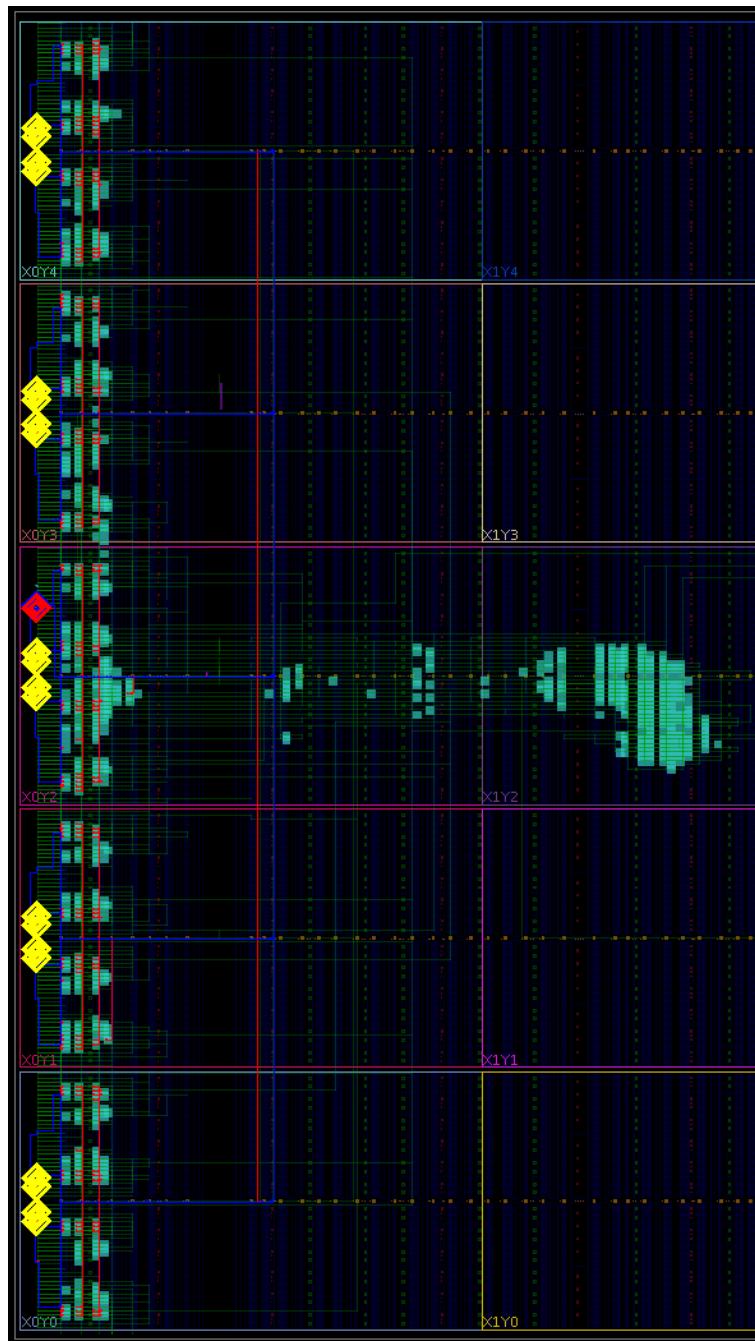
一些应用仍需要使用 MMCM 来生成 GT 输出时钟或 IBUFDS_GTE3/ODIV2 参考时钟的复杂非整数时钟分频。在这些情况下，BUFG_GT 必须直接驱动 MMCM。默认情况下，布局器尝试将 MMCM 布局在与 BUFG_GT 相同的时钟区域行上。如果其他 MMCM 尝试使用同一个 MMCM 站点 (site)，则必须确认自动 MMCM 布局仍尽可能接近 BUFG_GT，以避免长布线而浪费时钟资源。

单四通道 (Single Quad) 接口对比多四通道 (Multi-Quad) 接口

在多通道接口中，主通道可以为接口的所有 GT*CHANNEL 生成 [RT]XUSRCLK[2]。如果某个通道跨多个四通道，那么对于来自参考时钟源的 GT*CHANNEL，允许的最大距离为上下各 2 个时钟区域。

下图显示了 1 个多四通道接口。GT*CHANNEL 以黄色标记，TXUSRCLK 采用蓝色高亮，TXUSRCLK2 以红色高亮显示。驱动 TXUSRCLK 和 TXUSRCLK2 的 BUFG_GT 位于中间四通道内，并分别以蓝色和红色标记。

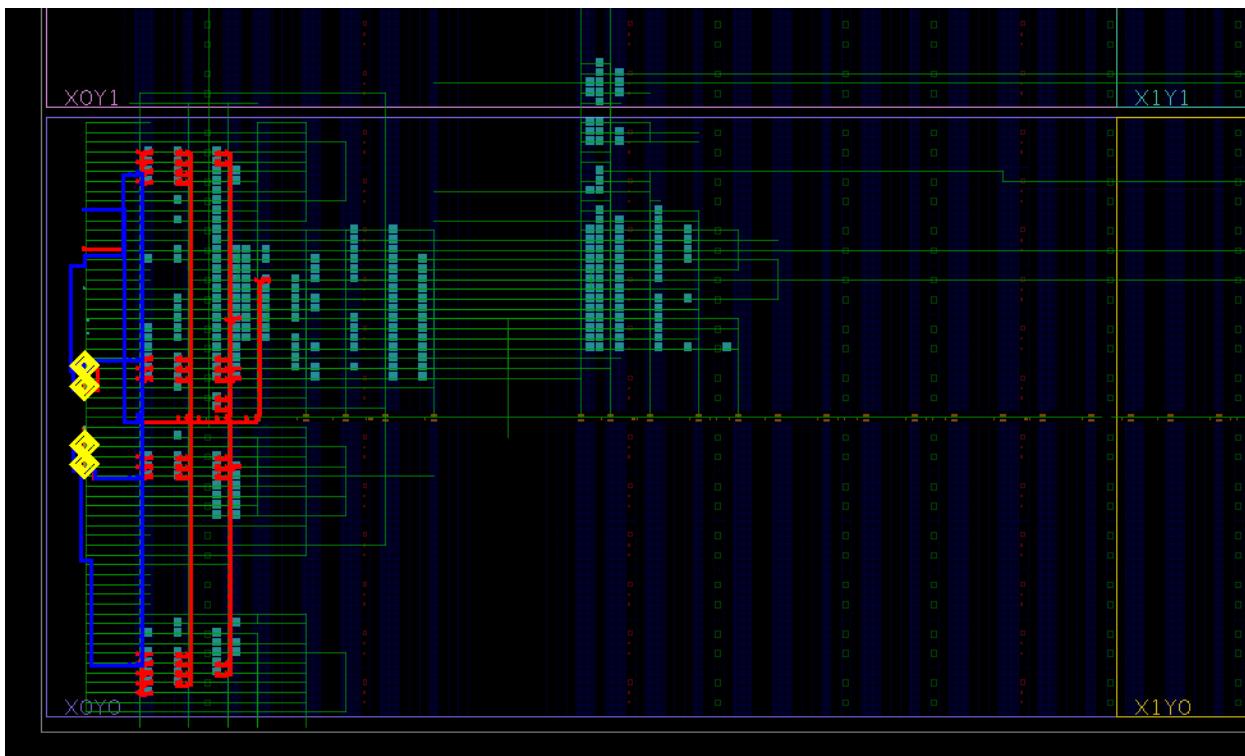
图 79：用于多四通道接口的 TXUSRCLK/TXUSRCLK2 时钟布局



如果 GT 接口包含在单四通道内，那么布局器会将 BUFG_GT 时钟作为局部时钟来处理。在此情况下，布局器会尝试将 BUFG_GT 时钟负载以水平方式布局在时钟区域内的 BUFG_GT 附近，从包含 BUFG_GT 的时钟区域开始，并且可能占用多达器件一半的宽度。

要覆盖布局器区域时钟约束，请将任意 BUFG_GT 时钟负载分配到 Pblock。下图显示了 1 个单四通道接口。GT*CHANNEL 以黄色标记，TXUSRCLK 采用蓝色高亮，TXUSRCLK2 以红色高亮显示。所有 TXUSRCLK2 负载都与 GT*CHANNEL 布局在相同时钟区域内。

图 80：单四通道接口的 TXUSRCLK/TXUSRCLK2 时钟布线



[RT]XUSRCLK/[RT]XUSRCLK2 偏差匹配

当 [RT]XUSRCLK2 的运行频率为 [RT]XUSRCLK 的一半时（即，分别将 BUFG_GT 除以 1 和除以 2），在 GT 接口的每个 GT*CHANNEL 中的 [RT]XUSRCLK/[RT]XUSRCLK2 对之间存在严格的偏差要求。为满足此偏差要求，GT*CHANNEL 最多可比生成 [RT]XUSRCLK/[RT]XUSRCLK2 对的主通道高或低 2 个时钟区域。此外，布局器会严格控制偏差，如下所示：

- 将 BUFG_GT 对分配到四通道中的上方或下方的 12 个 BUFG_GT
- 为靠近含 BUFG_GT 的时钟区域的两个时钟分配时钟根



建议：为避免出现偏差违例，AMD 强烈建议当 [RT]XUSRCLK2 的运行频率为 [RT]XUSRCLK 的一半时采用以下时钟拓扑结构。

Integrated Block for PCI Express CORECLK/PIPECLK/USERCLK 偏差匹配

UltraScale Integrated Block for PCI Express® 需要 3 个时钟：CORECLK、USERCLK 和 PIPECLK。这 3 个时钟源于由物理接口的某个 GT*_CHANNEL 的 TXOUTCLK 管脚驱动的 BUFG_GT。在 CORECLK 与 PIPECLK 管脚之间以及 CORECLK 与 USERCLK 管脚之间存在严格的偏差要求。为满足偏差要求，布局器按如下方式严格控制偏差：

- 将驱动 3 个 PCIe 时钟的 BUFG_GT 组合在一起并分配到四通道中的上方或下方的 12 个 BUFG_GT
- 将全部 3 个时钟的时钟根都分配到同一个时钟区域

注释：如需了解有关 PCIe 时钟要求的更多信息，请参阅《UltraScale 器件 Gen3 Integrated Block for PCI Express LogiCORE IP 产品指南》(PG156)。

7 系列器件时钟设置

注释：本章节使用 AMD Virtex™ 7 时钟资源作为示例。Virtex-6 的时钟资源与此类似。如果使用其他架构，根据器件，请参阅《7 系列 FPGA 时钟资源用户指南》(UG472) 或《UltraScale 架构时钟资源用户指南》(UG572)。

Virtex-6 和 Virtex 7 器件包含 32 个全局时钟缓冲器（称之为 BUFG）。BUFG 可满足设计的大部分时钟需求，且对时钟数量、设计性能及时钟控制要求不高。全局时钟资源包括 BUFG、BUFGCE、BUFGMUX 和 BUFGCTRL 原语，每个都有自己的特性。如需了解有关这些全局时钟组件特性的更多信息，请参阅对应于您的器件的《时钟资源用户指南》（《7 系列 FPGA 时钟资源用户指南》(UG472) 或《UltraScale 架构时钟资源用户指南》(UG572)）以及《库指南》（《Vivado Design Suite 7 系列 FPGA 和 Zynq 7000 SoC 库指南》(UG953) 或《UltraScale 架构库指南》(UG974)）。



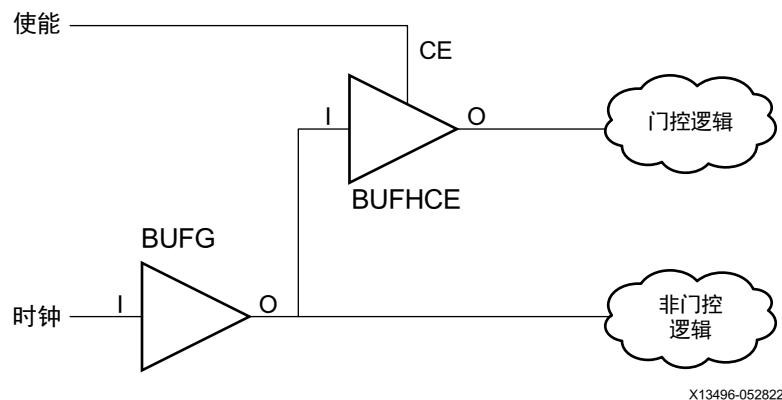
建议：如果时钟需求超过 BUFG 的数量，或是需要更优异的总体时钟特性，应根据可用时钟资源分析时钟需求，并针对任务选择最佳资源。

除全局时钟资源之外，还提供区域时钟资源，其有助于更加严格地控制时钟网络。区域时钟资源包括水平时钟区域缓冲器(BUFH、BUFHCE)、区域时钟缓冲器(BUFR)、I/O 时钟缓冲器(BUFIO) 及多区域时钟缓冲器(BUFMR)。如需了解有关这些区域时钟组件特性的更多信息，请参阅对应于您的器件的《时钟资源用户指南》（《7 系列 FPGA 时钟资源用户指南》(UG472) 或《UltraScale 架构时钟资源用户指南》(UG572)）以及《库资源》（《Vivado Design Suite 7 系列 FPGA 和 Zynq 7000 SoC 库指南》(UG953) 或《UltraScale 架构库指南》(UG974)）。

使用水平时钟区域缓冲器进行时钟门控

水平时钟区域缓冲器(BUFHCE) 可与 BUFG 搭配使用来执行中精度的时钟门控功能。对于时钟域中所含负载范围从数百到数千的各部分，如果要间歇性停止时钟设置，那么可以使用 BUFHCE 作为有效的时钟设置资源。每个 BUFG 均可驱动相同或不同时钟区域内的多个 BUFHCE，从而使您能够单独控制多个低时钟偏差域中的时钟设置。

图 81：水平时钟区域缓冲器



单独使用时，连接到 BUFH 的所有负载都必须驻留在相同时钟区域内。这样即可有效满足超高速、更高精度（负载更少）的时钟设置需求。BUFHCE 可用于在特定时钟区域内实现中等精度的时钟门控。您必须确保由 BUFH 驱动的资源不超过该时钟区域内的可用资源，并确保不存在任何其他冲突。

BUFH 与 BUFG、其他 BUFH 或任何其他资源所驱动的时钟域之间可能存在不同的相位关系。唯一例外是通过驱动 2 个 BUFH 来控制水平相邻的区域。在此情况下，由相同时钟源驱动 2 个 BUFH 时左右时钟区域之间的偏差应具有受严格控制的相位关系，在此关系中数据可安全跨越这 2 个 BUFH 时钟域。BUFH 可用于访问时钟输入或 GT 对面区域内的 MMCM 或 PLL。但必须谨慎使用此方法，以确保 MMCM 或 PLL 可用。

SSI 器件的其他时钟设置注意事项

通常以上提及的所有时钟设置注意事项都同样适用于 SSI 技术器件。但由于其结构原因，处理这些器件时需要考量其他因素。使用 BUFMR 时，它无法驱动跨 SLR 边界的时钟资源。因此，AMD 建议您将驱动 BUFMR 的时钟布局在 SLR 的中心时钟区域内的 bank 或时钟区域中。这样即可访问 SLR 左右两侧的全部 3 个时钟区域。

就全局时钟而言，对所需全局时钟 (BUFG) 数量不超过 16 个的设计，不必考虑其他因素。这些工具会自动分配 BUFG，以避免可能发生的任何争用。对于所需 BUFG 数量超过 16 个（但少于 32 个）的设计，必须根据全局时钟行争用和/或时钟负载的布局来考量有关管脚选择和布局的部分因素，以避免任意资源争用现象。

就像在所有其他 AMD 7 系列器件中一样，支持时钟功能的 I/O (CCIO) 及相关的时钟管理模块 (CMT) 都对它们在给定 SLR 中能够驱动的 BUFG 有限制性要求。SLR 上半部分或下半部分中的 CCIO 只能分别驱动对应 SLR 上半部分或下半部分中的 BUFG。因此，执行管脚和关联 CMT 选择时应注意所有 SLR 的上半部分或下半部分中所需的 BUFG 数量总计不得超过 16 个。为此，工具可以自动分配所有 BUFG 以允许将所有时钟驱动到无争用的所有 SLR。

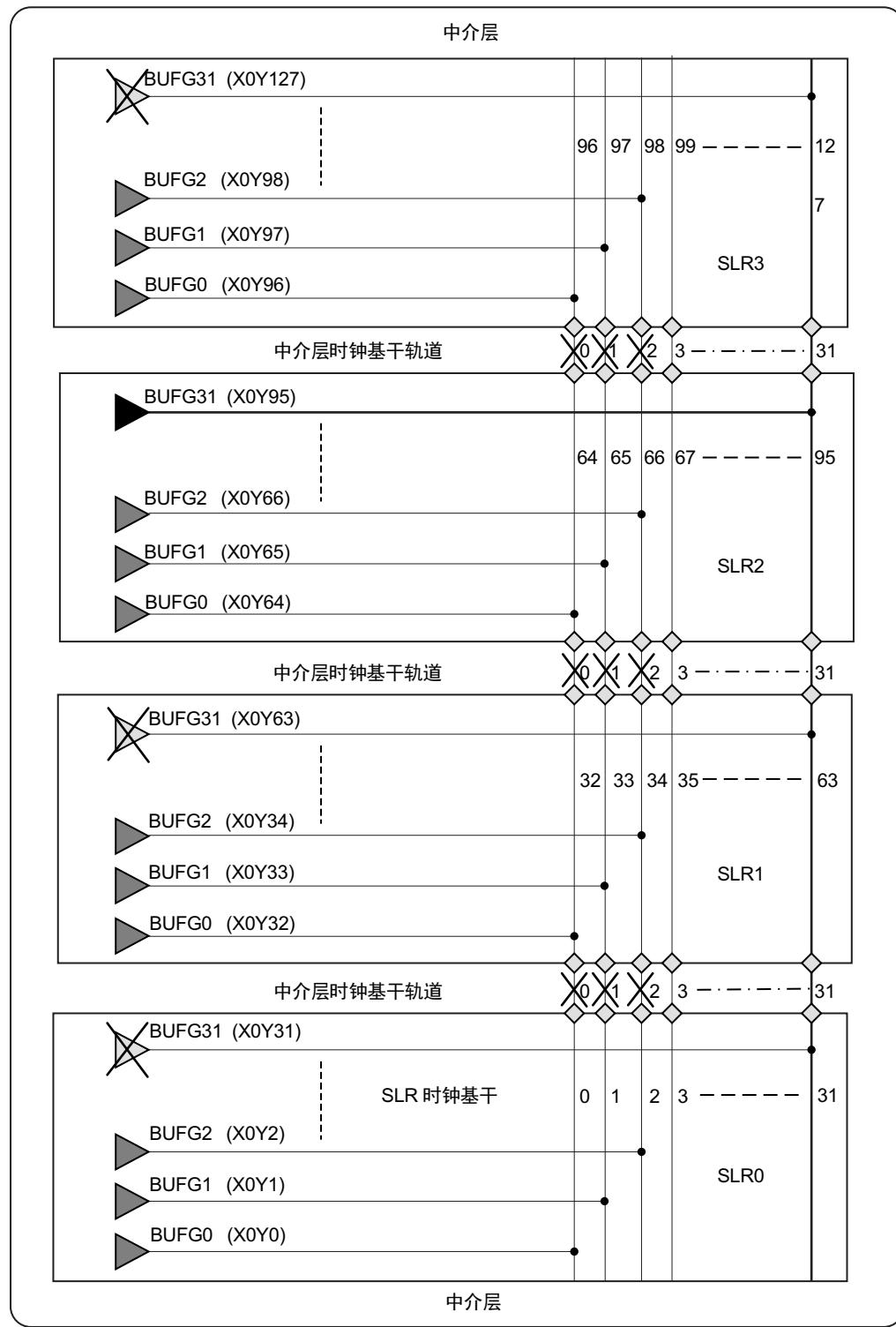
对于所需全局时钟数量超过 32 个的设计，AMD 建议您了解如何针对较小型的时钟域使用 BUFR 和 BUFH 来减少所需的全局时钟域的数量。通过将 BUFR 与 BUFMR 搭配使用可驱动 3 个时钟区域内占一半 SLR (Virtex 7 级 SLR 中约 250,000 个逻辑单元) 的资源。水平相邻的时钟区域可通过低偏差方式来驱动左右两侧的 BUFH 缓冲器，以支持占三分之一 SLR (约 167,000 个逻辑单元) 的时钟域。

尽可能利用这些资源可以减少时钟资源争用，并改进总体布局，从而提升性能和功耗。

如果需要超过 32 个全局时钟来驱动超过 1 个 SLR 中的半数资源或驱动多个 SLR，那么可以对 BUFG 全局时钟轴进行分段。在 SLR 外围的垂直全局时钟线上存在隔离缓冲器，可支持在占用相同垂直全局时钟轨道的不同 SLR 中使用 2 个 BUFG，而不会导致争用。要使用该功能，需要更多的用户控制和干预。在下图中，3 个 SLR 中的 BUFG0 到 BUFG2 已隔离，因此在其各自的 SLR 中具有独立时钟。另一方面，BUFG31 线尚未隔离。因此该 BUFG31 (在图中位于 SLR2 处) 会驱动全部 3 个 SLR 中的时钟线，并且应禁用位于其他 SLR 中的 BUFG31。

对于 BUFG，必须谨慎选择并手动进行布局 (LOC)。此外，每个时钟域的所有负载都必须手动分组并布局在相应的 SLR 内以避免时钟争用。如果所有时钟均已完成布局并且所有负载的管理方式不会造成任何时钟争用，并允许时钟访问所有负载，那么这可能导致全局时钟资源的使用量超过 32。

图82：SSI器件时钟线上的隔离选择



X14051_122019

SSI 技术器件中的全局时钟资源的时钟偏差

任何大型器件中的时钟偏差都可能在给定路径的总体时序预算中占用大部分预算。时钟偏差过大不仅会造成最高时钟速度的问题，还会导致面临苛刻的保持时间要求。在器件中存在多个裸片会导致 PVT 状态公式处理过程更加困难，但可借助 AMD 组装工艺来应对，即仅限将速度相似的裸片封装在一起。

即便借助如此额外操作，AMD 时序工具仍会在时序报告中对这些差异加以说明。在分析路径的过程中，计算建立和保持时间时会对这些方面加以分析，并在根据指定要求报告路径延迟的过程中一并报告结果。对 SSI 技术器件而言，无需用户额外进行计算或考虑，因为时序分析工具已在计算中考虑过这些因素。

如果使用顶部或底部的 SLR，偏差会增大，因为延迟差分随各点彼此间距增加而增大。因此，AMD 建议将必须驱动多个 SLR 的全局时钟布局在中心 SLR 内。这样能够在器件上实现更加均匀的总体时钟网络分布，从而降低总体时钟偏差。

处理 UltraScale 器件时，对时钟布局的影响较小。然而，AMD 仍强烈建议将时钟资源尽可能布局在接近时钟负载的中心点的位置，以降低时钟插入延迟并降低时钟功耗。

时钟结构设计

鉴于您已了解时钟设置判定的主要考量因素，下文将介绍如何实现设计所期望的时钟设置。

推断

Vivado 综合无需用户干预即可为时钟结构自动指定全局缓冲器 (BUFG)，适用的时钟结构数量不超过架构中允许的最大数量（除非另行指定或者由综合工具加以控制）。如前文所述，BUFG 能够提供高可控性低偏差网络，适合满足大部分时钟需求。除非设计时钟超出目标器件中 BUFG 的数量或功能，否则无需额外操作。

但是，对时钟结构应用额外控制可能有助于改善抖动、偏差、布局、功耗、最高时钟频率等方面的特性。

综合约束和属性

控制时钟资源的简单方法是使用 CLOCK_BUFFER_TYPE 综合约束或属性。综合约束可以用于：

- 防止 BUFG 推断。
- 用替代性时钟结构取代 BUFG。
- 指定其他情况下不存在的时钟缓冲器。

通过使用综合约束，无需对代码进行任何修改即可实现此类控制。

属性可置于下列任一位置：

- 直接置于 HDL 代码中，以便使其永久保存于代码中
- 作为 XDC 文件中的约束，这样无需修改源 HDL 代码就能实现此类控制

IP 的使用

某些 IP 对创建时钟结构有帮助。Clocking Wizard 和 Advanced IO Wizard 专用于帮助选择和创建时钟资源和结构，包括：

- BUFG
- BUFGCE

- BUFGCE_DIV (UltraScale 器件)
- BUFGCTRL
- BUFI0 (7 系列器件)
- BUFR (7 系列器件)
- 时钟修改块，如：
 - 混合模式时钟管理器 (MMCM)
 - 锁相环 (PLL) 组件

较复杂的 IP (如 PCIe 或 Transceivers Wizard IP) 还可能在总体 IP 中包含时钟结构。如能妥善考量其作用，即可提供额外的时钟资源。但如忽视其存在，则可能会限制设计其余部分的某些时钟选项。

AMD 强烈建议尽可能准确掌握并妥善利用设计其他部分中的所有例化的 IP 的时钟要求、功能和资源。

相关信息

[IP 的使用](#)

例化

控制时钟结构的最低级且最直接的方法是将所需时钟资源例化到 HDL 设计中。这样即可使用器件的所有可用功能，并全权掌控这些功能。使用 BUFGCE、BUFGMUX、BUFHCE 或需要额外逻辑和控制的其他时钟结构时，例化通常是唯一的选择。但即便是对简单的缓冲器而言，有时候实现期望的结果的最快方法还是直接将其例化到设计中。

将时钟资源包含在独立实体或模块内并在代码顶层或顶层附近将此实体或模块例化是一种有效时钟资源管理方式，在例化时尤其如此。通过将时钟资源置于代码顶层，就可以更方便地将其分配给设计中的多个模块。

请注意可在哪些场合以及应在哪些场合共享时钟资源。创建冗余时钟资源不仅是资源浪费，而且通常会增加功耗，造成更多潜在冲突和布局决策，导致总体实现工具的编译时间延长，使时序约束状况变得更加复杂。这也是把时钟资源置于顶层模块附近的又一重要原因。



提示：您可使用 Vivado HDL 模板来例化特定时钟原语。

相关信息

[使用 Vivado Design Suite HDL 模板](#)

控制时钟的相位、频率、占空比和抖动

本节提供了对时钟特性进行微调的技巧。

使用时钟修改块 (MMCM 和 PLL)

您可使用 MMCM 或 PLL 来更改传入时钟的总体特性。MMCM 最常用于移除时钟的插入延迟（将时钟与传入系统同步数据进行相位对齐）或者用于制约和控制时钟特性，例如：

- 创建更严格的相位控制
- 筛选时钟中的抖动
- 更改时钟频率

- 更正或更改时钟占空比

要使用 MMCM 或 PLL，必须协调多个属性以确保 MMCM 按规格范围内运行，并在其输出端提供所期望的时钟特性。因此，AMD 强烈建议您使用 Clocking Wizard 来正确配置此资源。

您还可直接例化 MMCM 或 PLL，从而进一步强化可控性。但是，请务必使用正确的设置来避免导致以下问题：

- 因抖动增加而导致时钟不确定性增加
- 构建的相位关系不正确
- 时序收敛实现难度增大



重要提示！ 使用 Clocking Wizard 配置 MMCM 或 PLL 时，默认情况下，Clocking Wizard 会尝试配置 MMCM，以使用合理的功耗特性降低输出抖动。

根据目的，您可更改 Clocking Wizard 中的设置以进一步最大程度降低抖动，从而以增加功耗为代价来改进时序。或者，您可降低功耗，但这会增加输出抖动。

使用 MMCM 或 PLL 时，请务必执行以下操作：

- 切勿使任何输入保持浮动。不建议依靠综合或其他最优化工具来锁定浮动值，因为值可能与期望值不同。
- 将 RST 连接到用户逻辑，以便通过由可靠的时钟源所控制的逻辑来对其进行断言。如果时钟中断，RST 接地就会导致问题出现。
- 在实现复位时使用 LOCKED 输出。例如，使 PLL 中完成时钟设置的同步逻辑保持处于复位状态中，直至断言 LOCKED 为止。LOCKED 信号必须达成同步后方可再在设计的同步部分中使用。AMD 建议将 LOCKED 添加到处理器映射中，这样在调试时 LOCKED 即可见。
- 确认 CLKFBIN 与 CLKFBOUT 之间的连接。仅当 PLL/MMCM 输出时钟需与输入参考时钟保持相位对齐时，才需要在反馈路径中包含 BUFG，例如，使用 ZHOLD 补偿模式时。
- 为避免 UltraScale 器件中的同步时钟域交汇路径上遇到 MMCM 或 PLL 相位误差时序惩罚，请使用 BUFGCE_DIV 代替 BUFGCE。



建议： 浏览 Clocking Wizard 中的不同设置，以确保根据总体设计目标来创建最符合期望的配置。

相关信息

[同步 CDC](#)

在时钟上使用 IDELAY 控制相位

对于 7 系列器件，如果只需少量调整相位，可以使用 IDELAY 或 ODELAY（而非 MMCM 或 PLL）来添加额外的延迟。这样可以增大时钟相对于任何相关数据的相位偏移。使用 UltraScale 器件时，在输入时钟源上无法使用 IDELAY。因此，如果需要进行相位操作，AMD 建议使用 MMCM。

使用门控时钟

AMD 器件内置专用时钟网络，可提供高扇出、低偏差的时钟资源。HDL 代码中包含的高精度时钟门控技巧可能会干扰此功能并阻碍专用时钟资源的有效使用。因此，在将 HDL 写入器件时，AMD 不建议将时钟门控结构编码到时钟路径中。而应改为使用编码技巧来控制时钟，通过推断时钟使能来停止设计中的相应部分，以便满足不同功能或功耗需求。

将时钟门控转换为时钟使能

如果代码已包含时钟门控结构，或者仅供需要此类编码样式的其他技术使用，那么 AMD 建议您使用综合工具，此类工具可将布局在时钟路径中的门重新映射到数据路径中的时钟使能。这样即可更有效地映射到时钟资源，并简化进出门控域的数据电路的时序分析。例如，对 Vivado 综合使用 `-gated_clock_conversion auto` 选项来尝试自动转换为寄存器时钟使能逻辑。对于复杂的门控时钟接口，请在 RTL 代码中使用 `GATED_CLOCK` 属性来指导 Vivado 综合。

对时钟缓冲器进行门控

当大部分时钟网络都可分时间段关闭时，您可以使用 BUFGCE 或 BUFGCTRL 启用或禁用时钟网络。此外，在针对性处理 UltraScale 器件时，可以对 BUFGCE_DIV 和 BUFG_GT 进行门控。对于 7 系列器件，还可以使用 BUFHCE、BUFR 和 BUFMRCE 来对时钟进行门控。

当不同时间段内时钟可减慢时，您也可以使用这些缓冲器和附加逻辑来周期性启用时钟信号线。或者，您也可以使用 BUFGMUX 或 BUFGCTRL 将时钟源从速度较快的时钟信号切换为速度较慢的时钟信号。

这些技巧都可以有效降低动态功耗。但是根据要求和时钟拓扑，某一种技巧可能比其他技巧更为行之有效。例如，在 7 系列器件中：

- 如果 BUFR 是外部生成的时钟（低于 450 MHz），并且只需为 3 个时钟区域供电，那么 BUFR 可能最有效。
- 对于 Virtex 7 器件，要将此方法用于多个时钟区域（但仅限最多 3 个垂直相邻区域），可能还需要 BUFMRCE。
- BUFHCE 更适合能够包含在单个时钟区域中的高速时钟。虽然 BUFGCE 可跨器件使用并且是最灵活的方法，但是它或许并不是最节能的选择。

控制和同步器件启动

器件完成配置后，器件将按顺序经历一系列事件并完成配置，随后进入正常工作状态。大部分配置序列中，最终步骤之一是断言全局置位复位 (GSR) 无效，随后断言全局使能 (GWE) 信号无效。在此过程中，设计处于已知的初始状态，随后经释放即可正常工作。

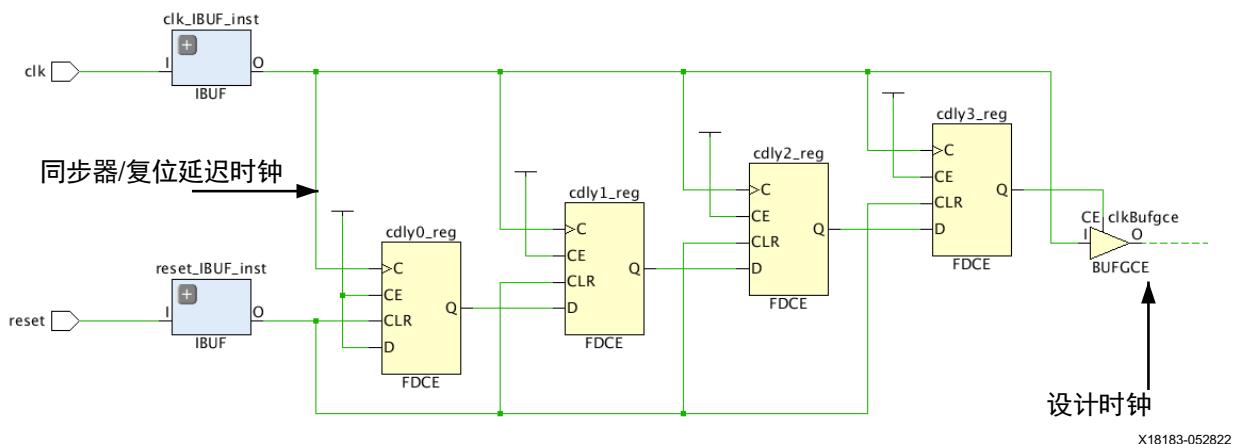
如果此释放点尚未同步到给定时钟域，或者如果时钟运行速度超过安全释放 GWE 的速度，那么部分设计可能会进入未知状态。对于某些设计，这无关紧要。而对于其他设计，这可能导致设计不稳定或者以错误方式处理初始数据集。

如果设计必须以已知状态启动，AMD 建议您采取行动，采用如下任一方法启动同步流程：

- 在设计的关键部分（例如，状态机）上使用时钟使能和/或局部复位（已同步）来确保这些设计部分的启动可控且已知。
- 使用含时钟使能功能的例化时钟缓冲器组件。

启用设计时钟前，延迟执行复位释放操作，且延迟的周期数按需增加。以下示例演示了如何在 UltraScale 器件中释放复位后延迟首个设计时钟沿。通过在同步器寄存器上设置 `ASYNC_REG=TRUE`，所有寄存器都将布局在单个 slice 中，因此无需受全局时钟资源驱动。为防止同步器时钟上发生时钟缓冲器插入，请在输入时钟端口上使用 `CLOCK_BUFFER_TYPE=NONE` 属性。

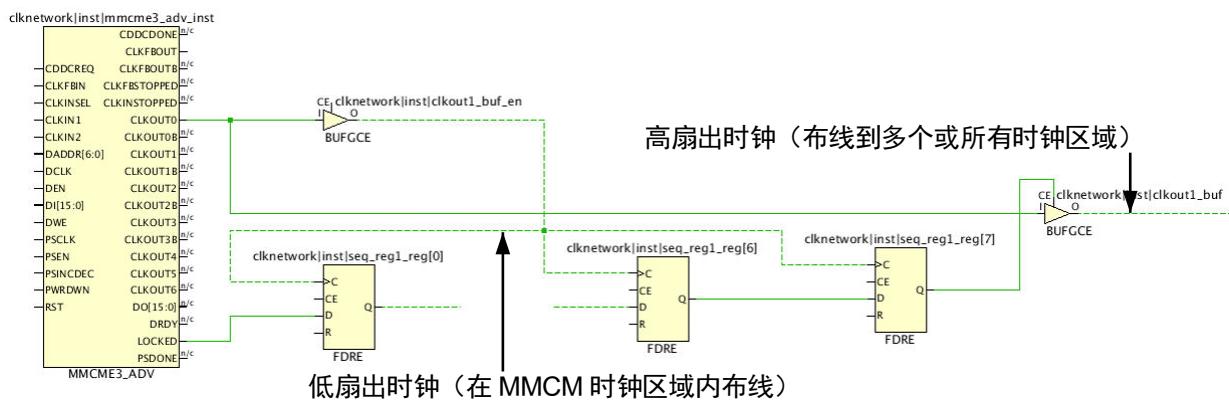
图83：复位安全时钟启动的同步和延迟示例



- 使用 MMCM 时，可以从 Clocking Wizard 中选择“Safe Clock Startup”（安全时钟启动）选项，以确保只有在设计时钟稳定可靠之后才能启用设计时钟。

以下示例显示了 UltraScale 器件 MMCM LOCKED 信号的同步阶段，该信号已连接到 BUFGCE 的 CE 管脚，此管脚用于驱动用户逻辑。第二个 BUFGCE 已并行连接到高扇出 BUFGCE（用户时钟），专用于控制 BUFGCE/CE 管脚的逻辑。此拓扑结构有助于在 UltraScale 器件中的 BUFGCE/CE 上实现时序收敛，因为它可最大限度减小同步器与 BUFGCE 管脚之间的时钟偏差。

图84：MMCM 安全时钟启动示例



提示：如果 MMCM 或 PLL 补偿模式设置为 ZHOLD 或 BUF_IN，那么来自 CLKOUT0 的所有时钟都将与反馈时钟组合，并使用相同的 CLOCK_ROOT。如果由此导致在 BUFGCE/CE 上出现时序违例，请仅在高扇出时钟与反馈时钟之间创建 CLOCK_DELAY_GROUP 约束。或者，您还可在低扇出时钟信号线上设置 USER_CLOCK_ROOT 约束，以将负载限制到与 MMCM 相同的时钟区域。对于 7 系列器件，由于时钟架构不同，通常不需要第 2 个时钟缓冲器来帮助达成时序收敛。

避免使用局部时钟

局部时钟是使用常规互连结构资源而不是专用全局时钟资源进行布线的时钟信号线。大多数情况下，Vivado 综合和 Vivado 逻辑最优化工具会插入时钟缓冲器以满足架构要求，或者用于具有超过 30 个时钟负载的时钟信号线。通常在如下情况下会出现局部时钟：

- 全局时钟由用互连结构逻辑实现的计数器进行分频
- 时钟门控转换不能从时钟路径中删除所有 LUT
- 7 系列器件中使用了太多的时钟缓冲器

注释：UltraScale 器件具有比 7 系列器件更多的时钟缓冲器，并且低扇出时钟缓冲器的高使用率通常不会导致任何问题。

一般情况下请避免使用局部时钟。局部时钟给实现工具带来了几个难题：

- 时钟偏差不可预测，导致难以执行时序收敛
- 增加由布线器谨慎处理的中低扇出信号线会导致潜在的可布线性问题



提示：如果局部时钟引发时序约束 QoR 问题，请尝试使用 Pblock 在一小块面积上对时钟驱动和负载进行布局规划。使用 `report_clock_utilization` 来识别局部时钟的位置，查看时钟布局，并决定如何降低其数量或影响。

创建输出时钟

使用 ODDR 组件即可从器件转发出时钟，以便对该器件外部的器件进行时钟设置。通过使其中一项输入保持高电平，使另一项输入保持低电平，即可轻松创建时钟并妥善控制其相位关系和占空比（例如，使 D1 保持为 0，使 D2 管脚保持为 1，即可实现 180 度相移）。通过使用置位/复位和时钟使能，还能控制时钟停止以及使时钟极性在一段时间内保持不变。

如果需要针对外部时钟进行进一步相位控制，可将 MMCM 或 PLL 与外部反馈补偿和/或低精度或高精度的固定或可变相位补偿配合使用。这样即可更有效地控制相对其他器件的时钟相位和传输时间，简化来自该器件的外部时序要求。

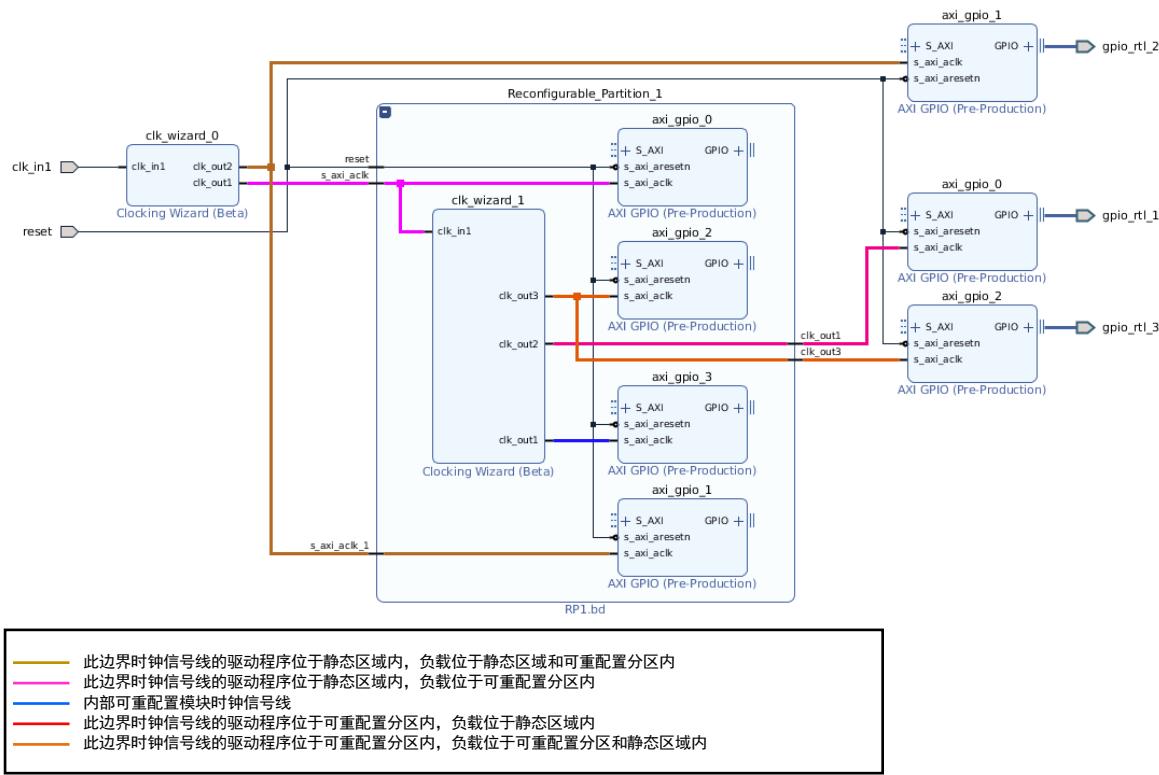
面向平台和 Dynamic Function eXchange 的时钟设置建议

本节涵盖了 Dynamic Function eXchange (DFX) 设计的时钟设置准则。一般，DFX 设计中的时钟分类为内部时钟和边界时钟：

- 可重配置模块内部时钟：含有驱动程序和所有负载的时钟位于可重配置模块 (RM) 内。
- 边界时钟：时钟所含的信号线跨可重配置模块的单元边界，如下所示：
 - 驱动程序位于静态区域内，负载位于 RM 内
 - 驱动程序位于 RM 内，负载位于静态区域内
 - 驱动程序位于静态区域内，负载分布于 RM 与静态区域之间
 - 驱动程序位于 RM 区域内，负载分布于 RM 与静态区域之间

下图显示了不同边界时钟的示例。

图85：DFX时钟拼块共享



如需了解有关 DFX 的更多信息，请参阅《Vivado Design Suite 用户指南：Dynamic Function eXchange》(UG909)。

时钟信号线的 DFX 行为

可重配置模块内部时钟信号线

在可重配置模块 (RM) 内部时钟信号线中，时钟根布局在可重配置分区 (RP) Pblock 内的负载中心位置。此时钟根布局可以在后续实现中为 RM 内部时钟的布局布线提供更多灵活性。AMD 建议尽可能采用此方法以实现更好的偏差和最优时钟根布局。

边界时钟信号线

首次实现后，边界时钟信号线轨道将锁定。边界时钟信号线上的分区管脚位置 (PPLOC) 将分配到可重配置分区 (RP) Pblock 所涵盖的所有时钟区域内。

由于边界时钟信号线可同时驱动静态负载和 RP 负载，因此边界时钟信号线的时钟根可布局在器件中的任意位置。AMD 建议在边界时钟信号线上使用 USER_CLOCK_ROOT 约束来手动约束 CLOCK_ROOT 位置，原因如下：

- 如果边界时钟的负载主要位于静态区域中，那么时钟根可能布局在静态区域内。
- 如果首次实现在 RP Pblock 中使用训练逻辑，那么首次实现后，边界时钟信号线可能锁定，并且时钟根位置偏离中心。
- 由于边界时钟信号线分配到 RP Pblock 所涵盖的所有时钟区域，因此相比于内部 RM 时钟信号线，边界时钟的时钟插入延迟相对较高。

时钟域交汇

设计中存在的时钟域交汇 (CDC) 电路会直接影响设计可靠性。您可自行设计电路，但 Vivado Design Suite 必须能够识别该电路，并且您必须正确应用 ASYNC_REG 属性。AMD 提供了 XPM 以确保电路设计正确，包括：

- 在 place_design 中驱动特定功能，以便缩短同步电路上的平均故障间隔时间 (MTBF)。
- 确保可供 report_synchronizer_mtbf 识别。
- 避免 report_cdc 错误和警告，通常如果迭代较长，那么在设计周期后期会出现此类错误和警告。



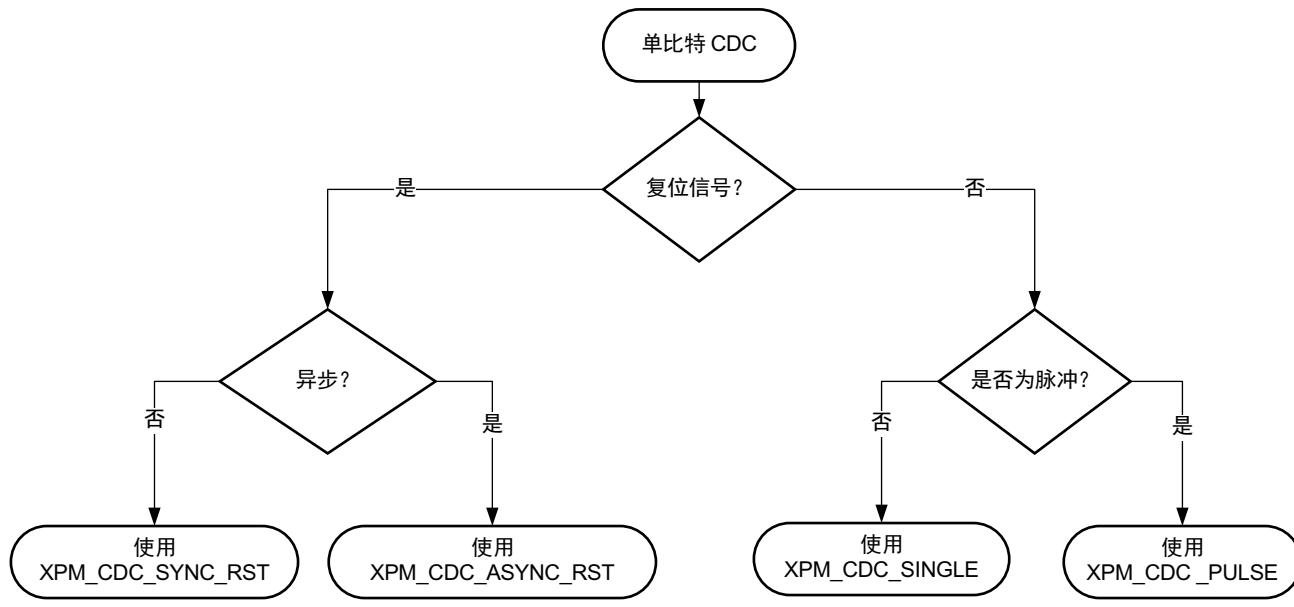
提示：对于可安全忽略的 CDC 违例，您可以使用豁免机制来豁免此类违例。欲知详情，请访问此[链接](#)以参阅《Vivado Design Suite 用户指南：设计分析与收敛技巧》(UG906) 中的相应内容。

跨 2 个异步时钟或者尝试通过添加伪路径约束来放宽 2 个同步时钟之间的时序约束时，需要使用 CDC 电路。使用 XPM 时，可以选择单比特总线或多比特总线来跨 2 个域。

单比特 CDC

下图显示了使用单比特交汇时所需的决策。

图 86：单比特 CDC 决策树



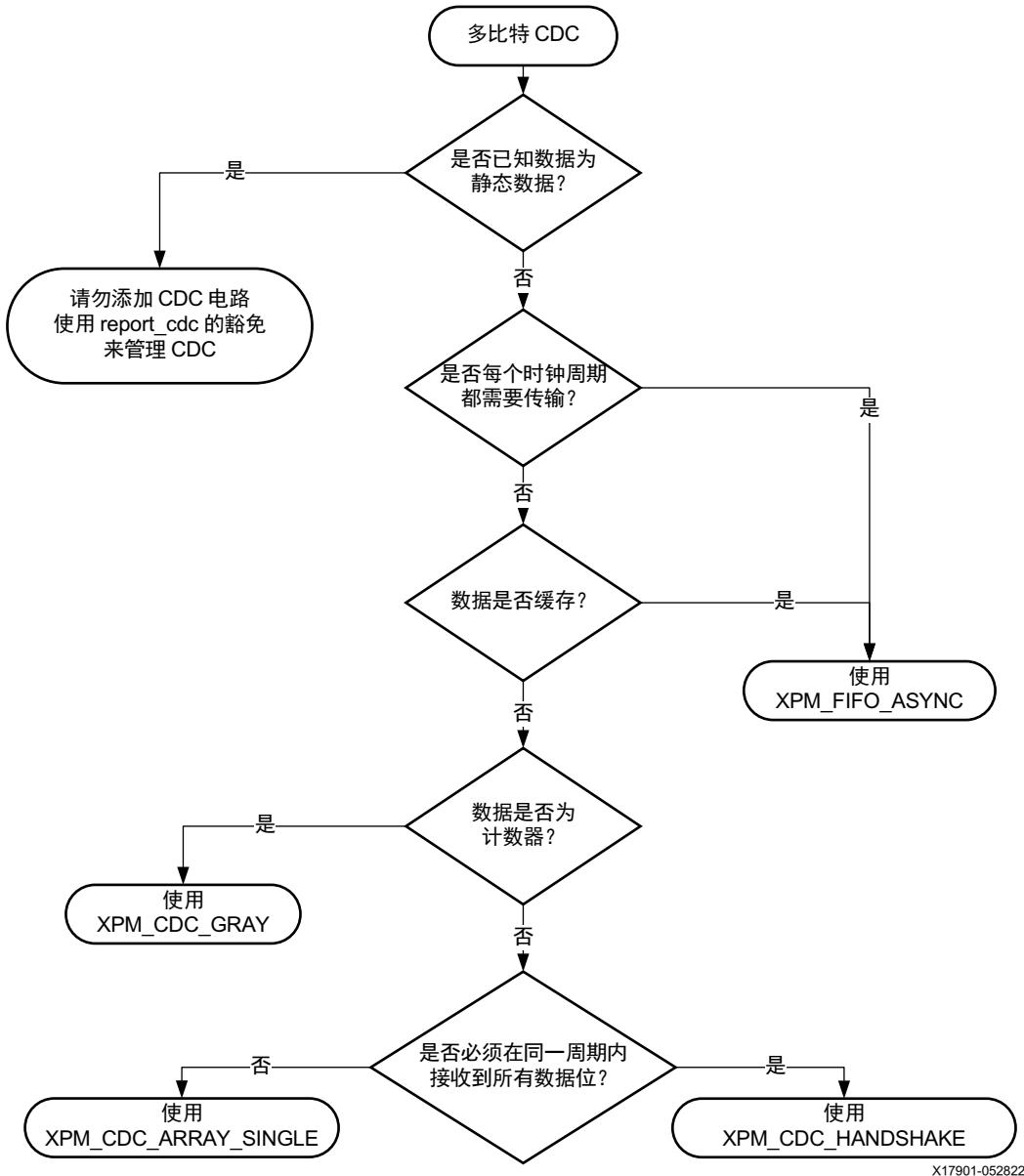
X17900-052822

注释：如需了解有关不同单比特同步器的更多信息，请参阅对应您的器件的《库指南》。

多比特 CDC

下图显示了使用多比特交汇时所需的决策。

图 87：多比特 CDC 决策树



注释：如需了解有关不同多比特同步器的更多信息，请参阅对应您的器件的《库指南》。

最优化 MTBF

设计的总 MTBF 是由如下各项组成的函数：

- 同步器 MTBF
- 由于单事件故障 (SEU) 导致的器件故障时间 (FIT)

注释：由于 SEU 引起的器件 FIT 速率在很大程度上取决于过程和器件尺寸。

同步器 MTBF 与设计关联，并且随以下变化：

- 异步 CDC 点数
- 每个交汇点处的同步器级数
- 目标 FF 的频率
- 源的切换速率

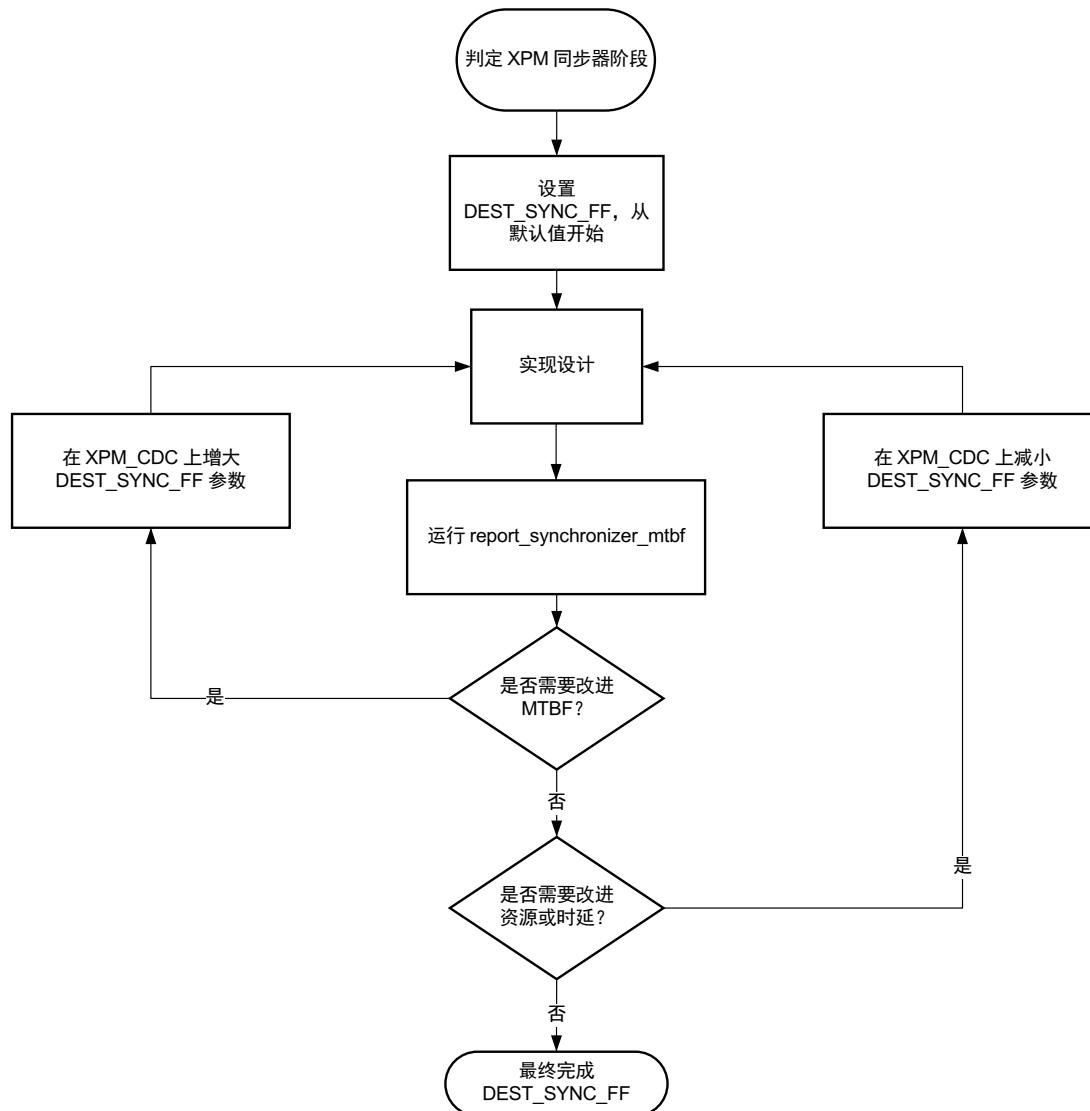
为 DEST_SYNC_FF 参数选择正确的值

使用 XPM CDC 模块时，DEST_SYNC_FF 参数可设置亚稳态保护寄存器的数量。该寄存器值会影响 MTBF、设计大小和交汇点处的时延。为此寄存器选择正确的值是 1 个迭代过程，需要执行如下操作：

1. 通过 Vivado Design Suite 实现流程运行设计。
2. 根据目标器件，执行以下操作之一：
 - 对于 7 系列器件，选择 DEST_SYNC_FF 的默认值。这是一种满足典型可靠性要求的保守方法。对于关键设计，请执行进一步分析。
 - 对于 UltraScale 器件，请运行 `report_synchronizer_mtbf` 命令，以报告整个设计的 MTBF。通过迭代流程（如下图所示），可在 MTBF、时延和资源之间找到合适的取舍。

注释：您还可将此迭代流程用于用户 CDC 电路，其中 ASYNC_REG 属性将可正确应用于所有同步寄存器。

图 88：UltraScale 器件的同步器 MTBF 最优化流程



X17899-052822

正确执行设计约束

XPM CDC 提供了自带的 set_max_delay_datapath_only 约束。XPM CDC 与 set_clock_groups 约束不兼容，后者优先级更高并且将覆盖 XPM 中的约束。

相关信息

[定义时钟组和 CDC 约束](#)

设计约束

设计约束用于定义各项要求，编译流程必须满足这些要求才能在硬件中正常运行设计。对于复杂的设计，约束通常还用于定义工具指南，以帮助实现收敛。并非所有约束都要在编译流程中的所有步骤中使用。例如，物理约束仅在执行实现步骤（最优化、布局和布线）期间使用。

由于综合与实现算法均由时序驱动，因此必须创建正确的时序约束。对设计进行过约束或欠约束都会导致难以实现时序收敛。您必须使用对应于自己的应用要求的合理约束。如需了解有关约束的更多信息，请参阅以下资源：

- 《Vivado Design Suite 用户指南：设计分析与收敛技巧》(UG906)
- [Vivado Design Suite 视频教程](#)网页上提供了“应用设计约束”视频教程

注释：传统设计流程和基于平台的设计流程以相似方式来使用设计约束。但基于平台的设计需要额外关注从设计静态区域到动态区域的跨边界信号。正确约束这些信号可确保平台灵活性并最大程度减少平台修改。

对设计约束进行组织以便执行编译

通常约束按类别和/或按设计模块组织到 1 个或多个文件中。无论采用何种组织方式，您都必须了解其整体依赖关系，并在载入存储器后复查其最终时序。例如，由于时序时钟必须先定义后才可供其他约束使用，因此您必须确保其定义位于约束文件开头和/或位于载入存储器的第一组约束文件中。

建议的约束文件

根据工程大小和复杂性，有多种适用于约束组织的方法可供选择。下面给出了一些建议。

简单设计

对于小型设计团队开发的简单设计：

- 1 个文件存储所有约束
- 1 个文件存储物理约束 + 1 个文件存储时序约束
- 1 个文件存储物理约束 + 1 个文件存储时序（综合）+ 1 个文件存储时序（实现）

复杂设计

对于复杂设计（含多个 IP 核或多个设计团队）：

- 1 个文件存储顶层时序约束 + 1 个文件存储顶层物理约束 + 1 个文件对应 1 个 IP 或主块

确认读取顺序

完成工程约束文件的组织后，必须根据文件内容确认文件读取顺序。在“工程模式”下，可在 AMD Vivado™ IDE 中或者使用 `reorder_files` Tcl 命令来修改约束文件的顺序。在“非工程模式”下，顺序直接由编译流程 Tcl 脚本中的 `read_xdc` 命令（针对 XDC 文件）和 `source` 命令（针对由 Tcl 脚本生成的约束）来定义。

建议的约束顺序

约束语言 (XDC) 基于 Tcl 语法和解读规则。与 Tcl 一样，XDC 属于顺序语言：

- 必须先定义变量，然后才能加以使用。同样，必须先定义时序时钟，然后才能将其用于其他约束中。
- 对于覆盖相同路径并具有相同优先级的等效约束，适用最后一项约束。
- 当多个时序例外覆盖同一条路径时，适用具有更高优先级的约束。

当考虑以上优先规则时，时序约束总体上应遵循以下顺序：

```
## Timing Assertions Section
# Primary clocks
# Virtual clocks
# Generated clocks
# Delay for external MMCM/PLL feedback loop
# Clock Uncertainty and Jitter
# Input and output delay constraints
# Clock Groups and Clock False Paths
## Timing Exceptions Section
# False Paths
# Max Delay / Min Delay
# Multicycle Paths
# Case Analysis
# Disable Timing
```

当使用多个 XDC 文件时，必须特别留意时钟定义，并确认从属关系排序是否正确。

物理约束可能位于任意约束文件中的任意位置。

创建综合约束

综合将提取设计的 RTL 描述，并使用时序驱动的算法将其变换为经最优化的技术所映射的网表。结果质量受 RTL 代码质量和提供的约束的影响。在编译流程的这个阶段，信号线延迟建模采用近似法，无法反映布局约束或复杂影响（例如拥塞）。建模的主要目的是通过真实且简单的约束获取满足时序约束要求或者接近满足要求的网表。

综合引擎接受所有 XDC 命令，但其中只有部分命令真正有效：

- 与建立/恢复分析有关的时序约束会影响 QoR：
 - `create_clock` / `create_generated_clock`
 - `set_input_delay` / `set_output_delay`
 - `set_clock_groups` / `set_false_path` / `set_max_delay` / `set_multicycle_path`
- 与保持和移除分析有关的时序约束在综合步骤中被忽略：
 - `set_min_delay` / `set_false_path -hold` / `set_multicycle_path -hold`

- RTL 属性会强制采纳映射和最优化算法所制定出的决策。以下提供一些示例：

- DONT_TOUCH / KEEP / KEEP_HIERARCHY / MARK_DEBUG
- MAX_FANOUT
- RAM_STYLE / ROM_STYLE / USE_DSP / SHREG_EXTRACT
- FULL_CASE / PARALLEL_CASE (仅限 Verilog RTL)

注释：在 XDC 文件中还可将同样的属性设置为特性。在不改变 RTL 的前提下，仅在某些情况下才能使用基于 XDC 的约束影响综合结果。

- 忽略物理约束 (LOC、BEL、Pblock)

综合约束使用的名称必须来自细化的网表（最好是端口和时序单元）。某些 RTL 信号会在细化过程中消失，并且无法为其赋予 XDC 约束。此外，由于细化后执行的各种最优化，信号线或逻辑单元将合并到各种技术原语（例如，LUT 或 DSP 块）中。要了解详细设计对象的名称，单击 Flow Navigator 中的“Open Elaborated Design”，然后浏览您感兴趣的层级。

部分寄存器被吸收到 RAM 块中，部分层级可能消失，以便允许实现跨边界最优化。

所有经细化的网表对象或层级均可通过使用 DONT_TOUCH、KEEP、KEEP_HIERARCHY 或 MARK_DEBUG 约束来保留，但存在时序或面积 QoR 劣化的风险。

最后，某些约束可能存在冲突而不被综合所认可。例如，如果在跨多个层级的网表上设置 MAX_FANOUT 属性，并且使用 DONT_TOUCH 保留部分层级，那么将限制或完全阻止扇出最优化。



重要提示！ 与实现阶段不同，综合可能会将用于定义时序约束的 RTL 网表对象优化掉以便实现更好的面积 QoR。一般这不会导致问题，前提是约束进行更新和确认以满足实现要求。但如果需要，仍可使用 KEEP 约束来保留任何对象以便在综合和实现期间应用约束。

完成综合后，AMD 建议您复查时序和使用报告，以确认网表质量满足应用要求并且可用于实现。

创建实现约束

实现约束必须准确反映最终应用的要求。物理约束（例如 I/O 位置和 I/O 标准）取决于开发板设计（包括开发板走线延迟）以及源自总体系统要求的设计内部要求。在进入实现步骤之前，AMD 强烈建议您对所有约束的正确性和准确性进行确认。错误约束可能会降低实现的 QoR 以及时序验收质量的可信度。

多数情况下，在综合与实现阶段可以使用相同的约束。但是，由于设计对象在综合阶段可能消失或发生名称变化，因此必须确认所有综合约束都可正确应用于实现网表。如果不是这样，那么您必须创建 1 个附加 XDC 文件，其中包含仅对实现有效的约束。

创建块级约束

开发多团队工程时，为方便起见，可为顶层设计的每个主要块创建独立的约束文件。通常每个主要块都会先独立开发并确认，最后再整合到 1 个或多个顶层设计中。

块级约束必须独立于顶层约束单独开发，并且必须尽可能采用通用设计以便应用于各种环境中。此外，这些约束不得影响块边界外的任何逻辑。

当实现子块时，最好在时序分析中包含全时钟网络，以确保偏差和时钟域交汇分析的准确性。这可能需要 1 个包含时钟组件的 HDL 封装文件和另一个约束文件以便复制顶层时钟约束。它仅用于子模块的时序确认。

如需了解有关约束范围以及将块级约束加载到顶层设计中的规则、准则和机制的更多信息，请访问此[链接](#)以参阅《Vivado Design Suite 用户指南：使用约束》(UG903) 中的相应内容。

为 Vitis 环境指定约束

在 AMD Vitis™ 环境中，您可将硬件内核指定为 C/C++ 内核或 RTL 内核：

- 使用 C/C++ 内核时，必须使用 Vitis HLS 为综合或实现指定附加的用户约束。随后，必须在 IP 封装器内封装 Vitis HLS 输出，此封装 IP 包含用户约束和工具生成的约束。如需了解更多信息，请参阅《Vitis 高层次综合用户指南》(UG1399)。
- 使用 RTL 内核时，必须在 IP 封装期间指定附加的综合与实现约束。如需了解更多信息，请参阅《Vivado Design Suite 用户指南：创建和封装定制 IP》(UG1118)。

在 Vitis 环境内，综合与实现的所有设计约束都必须随 IP 一并封装。如果封装 IP 还需其他约束，则必须重新封装 IP 以包含缺失的约束。

但封装 IP 后，可指定其他仅在实现期间使用的 XDC 约束。虽然 Vitis 环境会将底层 Vivado 工具流程抽象化用于实现可编程逻辑区域，但 Vitis 环境还会提供高级选项用于控制 Vivado 工具流程。借助这些高级控制选项，您即可指定要在每个实现阶段之前或之后执行的特定 Tcl 脚本，包括：init_design、opt_design、place_design、phys_opt_design、route_design 或 write_bitstream。如需了解有关 Tcl 脚本编制的更多信息，请参阅《Vivado Design Suite 用户指南：使用 Tcl 脚本》(UG894)。您可使用这些实现前 (Pre) 和实现后 (Post) Tcl 脚本来执行某些 Vivado 工具命令，例如，通过 read_xdc 或 source Tcl 命令来应用附加的 XDC 约束。

您可通过 Vitis 环境配置文件或者可直接在 v++ 编译器命令行上指定实现前和实现后 Tcl 脚本。

要在 Vitis 环境配置文件内指定实现前和实现后 Tcl 脚本，请在 [vivado] 节内使用 prop=run.impl_1.STEP.<PHASE>.TCL.<PRE|POST> 参数。

其中：

- <PHASE> 用于指定下列实现阶段：INIT_DESIGN、OPT_DESIGN、PLACE_DESIGN、PHYS_OPT_DESIGN、ROUTE_DESIGN 或 WRITE_BITSTREAM。
- PRE 用于在指定的实现阶段之前执行脚本。
- POST 用于在指定的实现阶段之后执行脚本。

例如：

```
[vivado]
prop=run.impl_1.STEPS.OPT_DESIGN.TCL.PRE=<pathToTclScript>
prop=run.impl_1.STEPS.OPT_DESIGN.TCL.POST=<pathToTclScript>
prop=run.impl_1.STEPS.PLACE_DESIGN.TCL.PRE=<pathToTclScript>
prop=run.impl_1.STEPS.PLACE_DESIGN.TCL.POST=<pathToTclScript>
prop=run.impl_1.STEPS.PHYS_OPT_DESIGN.TCL.PRE=<pathToTclScript>
prop=run.impl_1.STEPS.PHYS_OPT_DESIGN.TCL.POST=<pathToTclScript>
prop=run.impl_1.STEPS.ROUTE_DESIGN.TCL.PRE=<pathToTclScript>
prop=run.impl_1.STEPS.ROUTE_DESIGN.TCL.POST=<pathToTclScript>
```

要将 Pre 和 Post Tcl 脚本指定为 v++ 参数，请使用 --vivado.prop run.impl_1.STEP.<PHASE>.TCL.<PRE|POST>=<pathToTclScript> 命令行选项。例如，要指定在 opt_design 之前执行的 Tcl 脚本，请使用如下命令：

```
--vivado.prop run.impl_1.STEP.OPT_DESIGN.TCL.PRE=<pathToTclScript>
```

其中：

- --vivado 是 v++ 命令行选项，用于为 Vivado 工具指定指令。

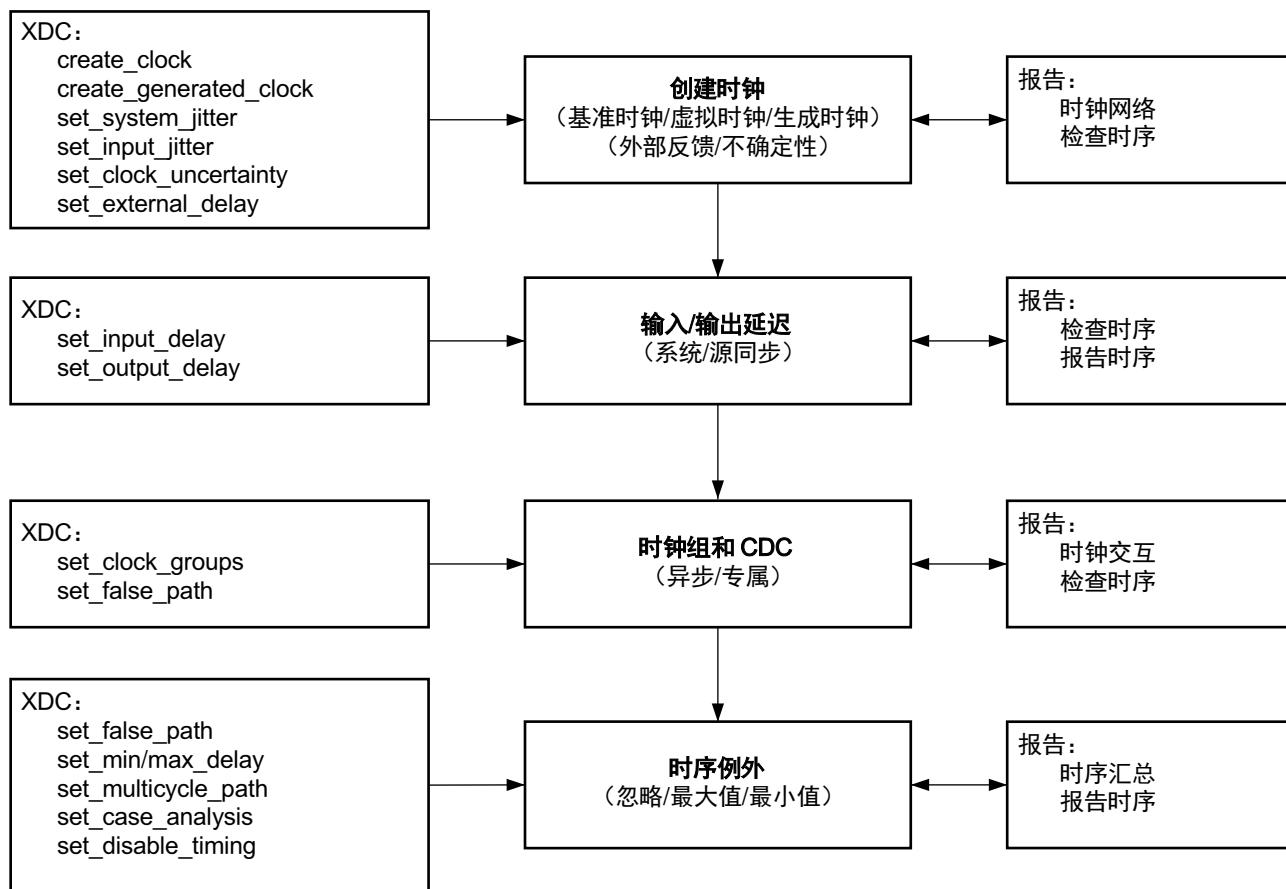
- prop 表示属性设置。
- run. 表示运行属性。
- impl_1. 表示运行轮次名称。
- STEP.OPT_DESIGN.TCL.PRE 表示当前指定的运行属性。
- <pathToTclScript> 表示属性值。

定义时序约束

定义时序约束的四个步骤

合格的约束的定义过程分为四个主要步骤，如下图所示。这些步骤遵循时序约束先后顺序和从属关系规则，并采用符合逻辑的方式来向时序引擎提供信息以执行分析。

图 89：时序约束制定步骤



X13445-052822

- 前 2 个步骤与时序断言有效有关，期间将从时钟波形和 I/O 延迟约束中衍生出默认时序路径要求。

- 在第 3 个步骤中，将对至少共享 1 条逻辑路径的异步或专属时钟域之间的关系进行审核。根据关系的性质，可输入时钟组或伪路径约束以忽略这些路径上的时序分析。
- 最后一个步骤对应于时序例外，设计人员可在此判定如何更改默认时序路径要求，包括利用特定约束来忽略、放宽或收紧时序要求。

约束创建与约束识别和约束确认任务息息相关，这些任务必须通过时序引擎生成的各种报告才能实现。时序引擎只能配合经过完全映射的网表使用，例如综合之后的网表。尽管可以用细化的网表输入约束，但还是建议使用综合后网表创建第一组约束，以便约束的分析和报告可交互执行。

为新设计创建时序约束或者完成现有约束时，AMD 建议使用“Timing Constraints Wizard”（时序约束向导）来快速识别上图中的前 3 个步骤中缺失的约束。“Timing Constraints Wizard”遵循本节中所述方法论来确保设计约束的安全性和可靠性，从而实现正确的时序收敛。如需了解有关“Timing Constraints Wizard”的更多信息，请参阅《Vivado Design Suite 用户指南：使用约束》(UG903)。

下列章节将详细描述以上所述的四个步骤：

- 定义时钟约束
- 约束输入和输出端口
- 定义时钟组和 CDC 约束
- 指定时序例外

在约束创建流程中执行相应的步骤时，请参阅各对应章节以了解详细方法论和用例。

定义时钟约束

时钟必须首先完成定义，方可供其他约束使用。时序约束创建流程的第一步是明确必须定义哪些时钟，以及这些时钟必须定义为“primary clock”（基准时钟）还是“generated clock”（生成时钟）。



重要提示！ 使用特定名称定义时钟 (-name 选项) 时，必须验证该时钟名称未被任何其他时钟约束或现有自动生成时钟占用。如果已在多个时钟约束中使用某个时钟名称，Vivado Design Suite 时序引擎会发出消息，以提醒您第 1 个时钟定义被覆盖。如果同一时钟名称使用了两次，那么第 1 个时钟定义将会丢失，并且 2 个时钟定义之间输入的引用此名称的所有约束也都将丢失。AMD 建议您避免覆盖时钟定义，除非不影响任何其他约束，并且所有时序路径都保持受约束。

识别时钟源

在设计中可通过“Clock Networks”（时钟网络）报告和“Check Timing”（检查时序）报告来识别未约束的时钟源。

时钟网络报告

约束和未约束的时钟源点分别列在 2 个不同类别中。对于每个未约束的时钟源点，必须确定应定义基准时钟还是生成时钟。

```
% report_clock_networks
Unconstrained Clocks
Clock sysClk (endpoints: 15633 clock, 0 nonclock)
Port sysClk
Clock TXOUTCLK (endpoints: 148 clock, 0 nonclock)
GTXE2_CHANNEL/TXOUTCLK
(mgtEngine/ROCKETIO_WRAPPER_TILE_i/gt0_ROCKETIO_WRAPPER_TILE_i/gtxe2_i)
Clock Q (endpoints: 8 clock, 0 nonclock)
FDRE/Q (usbClkDiv2_reg)
```

“检查时序” 报告

`no_clock` 检查用于报告不含时钟定义的有源叶时钟管脚组。每个组均与 1 个时钟源点关联，在其中必须定义时钟方可解决问题。

```
% check_timing -override_defaults no_clock
1. checking no_clock
-----
There are 15633 register/latch pins with no clock driven by root clock
pin: sysClk
(HIGH)
There are 148 register/latch pins with no clock driven by root clock pin:
mgtEngine/ROCKETIO_WRAPPER_TILE_i/gt0_ROCKETIO_WRAPPER_TILE_i/gtxe2_i/
TXOUTCLK
(HIGH)
There are 8 register/latch pins with no clock driven by root clock pin:
usbClkDiv2_reg/C (HIGH)
```

借助 `check_timing`，可使相同的时钟源管脚或端口出现在多个组中，具体取决于整个时钟树的拓扑结构。在此情况下，在建议的源管脚或端口上创建时钟即可解决所有关联组缺失时钟定义的问题。

相关信息

[检查设计是否正确约束](#)

创建基准时钟

基准时钟是指用于为设计定义时序参考的时钟，而时序引擎可利用基准时钟衍生出时序路径要求以及与其他时钟的相位关系。主时钟插入延迟的计算范围是从时钟源点（用于定义时钟的驱动管脚/端口）到时序单元（作为时钟扇出目标）的时钟管脚。

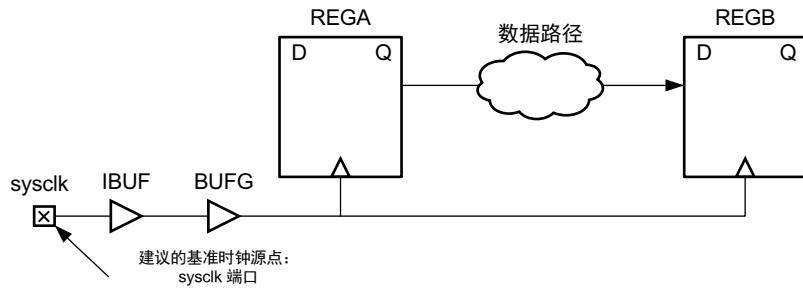
因此，重要的是在对应于设计边界的对象上定义基准时钟，以便准确计算其延迟并间接计算其偏差。

以下部分描述了典型的基准时钟根。

输入端口

您可使用输入端口作为基准时钟根，如下图所示。

图 90：输入端口的 `create_clock`



X13446-052822

约束示例：

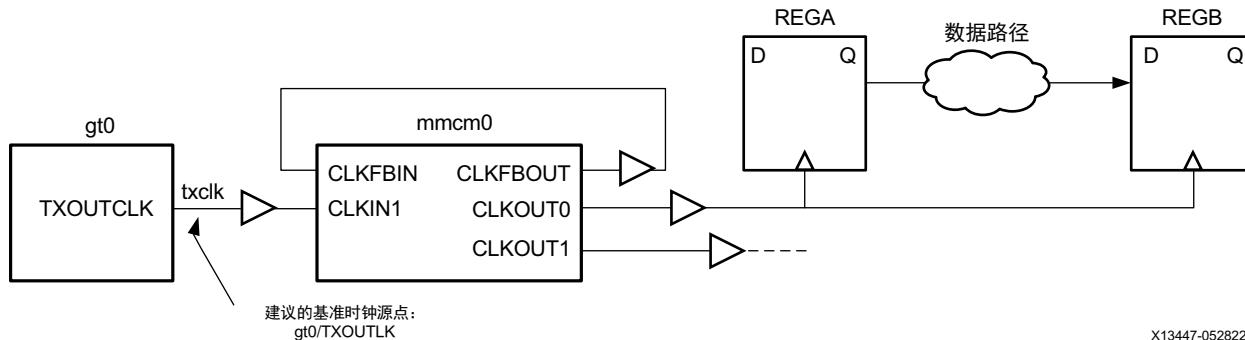
```
create_clock -name SysClk -period 10 -waveform {0 5} [get_ports sysclk]
```

该示例中，波形的占空比定义为 50%。以上显示的 `-waveform` 实参用于展示其使用率，只有在定义占空比非 50% 的时钟时才需要使用。欲知详情，请参阅《Vivado Design Suite Tcl 命令参考指南》(UG835) 中的 `create_clock` Tcl 命令。对于差分时钟输入缓冲器，只需在差分对的 P 侧对基准时钟进行定义即可。

7 系列器件中的千兆位收发器输出管脚

您可使用千兆位收发器输出管脚（例如，已恢复的时钟）作为基准时钟根，如下图所示。

图 91：原语管脚上的 `create_clock`



约束示例：

```
create_clock -name txclk -period 6.667 [get_pins gt0/TXOUTCLK]
```

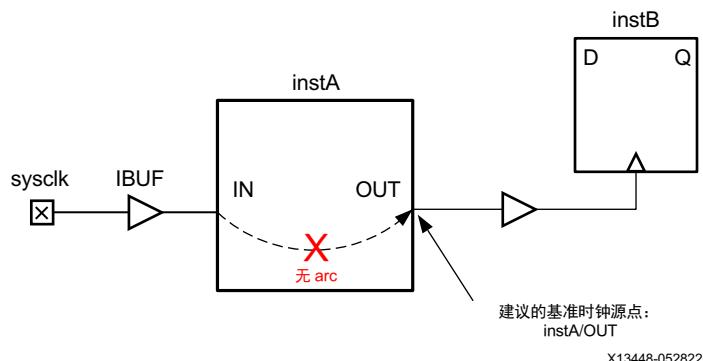
建议：对于面向 7 系列器件的设计，AMD 还建议定义 GT 输入时钟，因为 Vivado 工具会计算 GT 输出管脚上期望的时钟，并将这些时钟与用户创建的时钟进行比较。如果时钟不同或者到 GT 的输入时钟丢失，工具会发出方法论检查警告。

注释：对于面向 AMD UltraScale™ 和 AMD UltraScale+™ 器件的设计，AMD 不建议在 GT 的输出上定义基准时钟，因为在定义 REFCLK 输入时钟时，将自动衍生 GT 时钟。

某些硬件原语输出管脚

您可使用某些硬件原语的输出管脚作为基准时钟根（如下图中所示输出管脚），此类输出管脚不具有来自相同原语的输入管脚的时序 arc。

图 92：时钟路径因缺失时序 arc 而断开

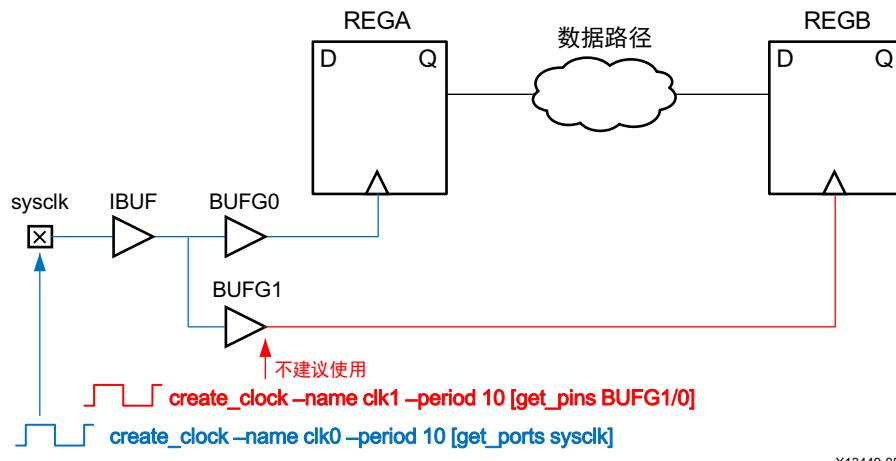




重要提示！ 在基准时钟传递扇出中不应定义另外 1 个基准时钟，因为这种情况不但不符合任何硬件现实，还会妨碍完整的时钟插入延迟计算，从而阻碍正确的时序分析。如果发生任何这种情况，必须重新修改并修正约束。

下图显示的示例中，时钟 `clk1` 是在时钟 `clk0` 的传递扇出中定义的。时钟 `clk1` 会从 BUFG1 输出开始覆盖此输出处所定义的 `clk0`。因此，由于 `clk0` 与 `clk1` 之间歪斜突变无效导致 REGA 与 REGB 之间的时序分析并不准确。

图 93：不建议在另一个时钟的扇出中使用 `create_clock`



创建生成时钟

生成时钟 (generated clock) 是从称为主时钟 (master clock) 的另一个现有时钟衍生的。它通常用于描述逻辑块对主时钟执行的波形变换。由于生成时钟定义取决于主时钟特性，因此必须首先定义主时钟。为显式定义生成时钟，必须使用 `create_generated_clock` 命令。

自动衍生时钟

大部分生成时钟都是由 Vivado 时序引擎自动衍生的，该引擎可识别时钟修改块 (CMB) 及其对主时钟执行的变换。

在 AMD 7 系列器件中，CMB 包括：

- MMCM*/PLL*
- BUFR
- PHASER*

在 AMD UltraScale 系列器件中，CMB 包括：

- MMCM*/PLL*
- BUFG_GT/BUFGCE_DIV
- GT*_COMMON/GT*_CHANNEL/IBUFDS_GTE3
- BITSLICE_CONTROL/RX*_BITSLICE
- ISERDESE3

对于时钟树上的任何其他组合单元而言，时序时钟可通过这些单元进行传输，且无需在输出端重新定义，除非此类单元已进行波形变换。通常应尽可能依靠自动衍生机制，因为就定义可对应于实际硬件行为的生成时钟来说，这是最安全的方法。

如果您认为 Vivado Design Suite 时序引擎所选的自动衍生时钟名称不合适，那么可以使用 `create_generated_clock` 命令（不指定波形变换）强制输入自己选择的名称。该约束应刚好位于约束文件中定义主时钟的约束之后。例如，如果由 MMCM 实例生成的时钟默认名称为 `net0`，那么您可添加以下约束来强制输入自己的名称（在给定示例中，此名称为 `fftClk`）：

```
create_generated_clock -name fftClk [get_pins mmcm_i/CLKOUT0]
```

为避免歧义，约束必须连接到时钟的源管脚。如需了解更多信息，请参阅《Vivado Design Suite 用户指南：使用约束》(UG903)。

用户定义的生成时钟

定义所有基准时钟后，可使用“Clock Networks”（时钟网络）或“Check Timing”（检查时序）(`no_clock`) 报告来识别时钟树中不含时序时钟的部分，并定义相应的生成时钟。

有时要理解逻辑椎对主时钟所执行的变换并不容易。在此情况下，必须采用最保守的约束。例如，源管脚是时序单元输出。主时钟至少除以 2，因此，正确的约束应如下示例所示：

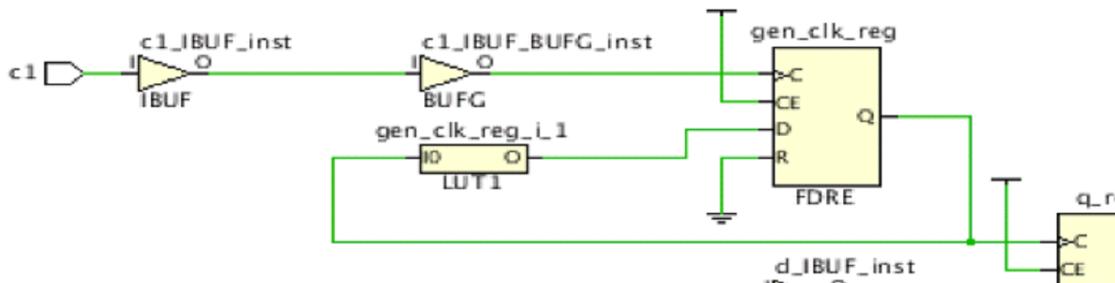
```
create_generated_clock -name clkDiv2 -divide_by 2 \
-source [get_pins fd/C] [get_pins fd/Q]
```

最后，如果设计包含锁存器，那么时序时钟还需要连接到锁存器门控管脚，并且如果缺少约束，则将由“Check Timing”（检查时序）(`no_clock`) 来报告锁存器门控管脚。您可遵循上述示例来定义这些时钟。

主时钟与生成时钟间的路径

与基准时钟不同，生成时钟必须在其主时钟的传递扇出中进行定义，这样时序引擎才能精确计算其插入延迟。不遵守此原则将导致时序分析错误，而且很有可能导致时序裕量计算无效。例如，在下图中，`gen_clk_reg/Q` 用作为下一个触发器(`q_reg`)的时钟，并且它还位于基准时钟 `c1` 的扇出椎中。因此，`gen_clk_reg/Q` 应包含 `create_generated_clock` 而不是 `create_clock`。

图 94：主时钟扇出中的生成时钟



```
create_generated_clock -name GC1 -source [get_pins gen_clk_reg/C] -
divide_by 2
[get_pins gen_clk_reg/Q]
```

验证时钟定义和覆盖范围

在存储器中定义并应用所有设计时钟后，即可使用 `report_clocks` 命令验证每个时钟的波形以及主时钟和生成时钟之间的关系：

```
Clock Period Waveform Attributes Sources
sysClk 10.00000 {0.00000 5.00000} P {sysClk}
clkfbout 10.00000 {0.00000 5.00000} P,G {clkgen/mmcmm_adv_inst/CLKFBOUT}
cpuClk 20.00000 {0.00000 10.00000} P,G {clkgen/mmcmm_adv_inst/CLKOUT0}
...
=====
Generated Clocks
=====
Generated Clock : cpuClk
Master Source : clkgen/mmcmm_adv_inst/CLKIN1
Master Clock : sysClk
Edges : {1 2 3}
Edge Shifts : {0.000 5.000 10.000}
Generated Sources : {clkgen/mmcmm_adv_inst/CLKOUT0}
```

此外，您还可验证所有内部时序路径都被至少 1 个时钟所覆盖。“Check Timing”（检查时序）报告为此提供了两项检查：

- `no_clock`: 报告已定义的时钟无法连接到的任何活动时钟管脚。
- `unconstrained_internal_endpoint`: 如果某些时序单元具有与时钟相关的时序检查但尚未定义时钟，则报告此类时序单元的所有数据输入管脚。

如果两项检查都返回 0，说明时序分析覆盖范围广。

或者，还可运行 XDC 和“Timing Methodology”（时序方法论）检查来验证在建议的网表对象上是否已定义所有时钟，同时避免造成任何约束冲突或不准确的时序分析情境。

请使用以下命令来运行这些检查：

```
report_methodology -checks [get_methodology_checks {TIMING-* XDC*}]
```

相关信息

[运行 Report Methodology](#)

调整时钟特性

定义时钟及其波形后，下一步是输入与噪声或不确定性建模相关的所有信息。XDC 语言用于将抖动和相位误差相关的不确定性与偏差和延迟建模相关的不确定性加以区分。

抖动

对于抖动，最好使用 Vivado Design Suite 所使用的默认值。您可按如下方式修改默认计算：

- 如果基准时钟进入器件时随机抖动大于 0，请使用 `set_input_jitter` 命令指定峰值间抖动值（以纳秒 (ns) 为单位）。
- 如果器件电源有噪声，请使用 `set_system_jitter` 调整全局抖动。AMD 不建议增大默认系统抖动值。

对于生成时钟，抖动是由主时钟和时钟修改块的特性衍生而来的。您无需调整这些数值。

其他不确定性问题

如果需要在某个时钟的时序路径上或 2 个时钟之间的时序路径上添加额外裕度，必须使用 `set_clock_uncertainty` 命令。这也是对部分设计进行过约束而不必修改实际时钟沿和总体时钟关系的最佳且最安全的途径。您定义的时钟不确定性是在 Vivado 工具计算所得抖动的基础上附加的，可为建立时间和保持时间分析单独指定此不确定性。

例如，设计时钟 `clk0` 的所有时钟间路径上的裕度需收紧，幅度为 500 ps，以使设计更稳健，承受建立时间和保持时间噪声的能力更强：

```
set_clock_uncertainty -from clk0 -to clk0 0.500
```

注释：收紧设计上的保持时间裕度可能导致专用站点内部路径和级联路径上出现保持时间违例，并且布线器无法通过绕行站点内部信号线来解决此类违例。

如果您在 2 个时钟之间指定额外的不确定性，那么必须应用双向约束（假定数据流为双向）。以下示例演示了如何在 `clk0` 和 `clk1` 之间仅针对建立时间将不确定性增加 250 ps：

```
set_clock_uncertainty -from clk0 -to clk1 0.250 -setup  
set_clock_uncertainty -from clk1 -to clk0 0.250 -setup
```

时钟源位置的时钟时延

可使用含 `-source` 选项的 `set_clock_latency` 命令在时钟源处对时钟时延进行建模。该方法在两种情况下有用：

- 用于在器件外部指定与输入和输出延迟无关时钟延迟传输。
- 用于在非关联 (OOC) 编译期间，对块所使用的时钟内部传输时延进行建模。在此类编译流程中，不含完整时钟树的描述，因此块外部的最小和最大工作条件之间的差异无法自动进行计算，必须手动建模。

此约束仅限高级用户使用，因为它通常难以提供有效的时延值。

MMCM 或 PLL 外部反馈回路延迟

当连接 MMCM 或 PLL 反馈回路以便补偿开发板延迟（而非内部时钟插入延迟）时，必须使用 `set_external_delay` 命令指定最佳和最差延迟情况下器件外部的延迟。不指定此延迟会导致与 MMCM 或 PLL 关联的 I/O 时序分析变得无关紧要，并且可能导致无法实现时序收敛。此外，使用外部补偿时，必须相应调整输入和输出延迟约束，而不只是考量正常情况下开发板上的时钟走线延迟。

约束输入和输出端口

除了指定设计的每个端口的位置和 I/O 标准外，还必须指定输入和输出延迟约束以描述进出器件接口的外部路径的时序。这些延迟是根据通常同样在开发板上生成并进入器件的时钟来定义的。在某些情况下，如果与 I/O 路径相关的时钟所含波形不同于开发板时钟的波形，那么必须根据虚拟时钟来定义延迟。



重要提示！ 只能为使用 I/O 逻辑的接口（例如，ISERDES/OSERDES/IDDR/ODDR/IOB 寄存器或互连结构）约束 I/O 延迟。如需了解组件模式时序相关准则，请参阅《使用 SelectIO 接口组件原语进行设计》(XAPP1324)。对于使用 UltraScale 器件 SelectIO 本机模式创建的高速 I/O 接口，请参阅答复记录 68618。

系统级透视图

I/O 路径的建模方式与 Vivado Design Suite 时序引擎所执行的寄存器间路径建模方式较为相似，区别在于您必须定义约束才能对位于器件外部的路径延迟部分进行建模。分析内部路径时，建立和保持分析都会考虑最小和最大延迟。对于 I/O 路径而言同样如此。基于这个原因，对最小和最大延迟条件进行描述就显得尤为重要。默认情况下 I/O 时序路径可作为单周期路径进行分析，这意味着：

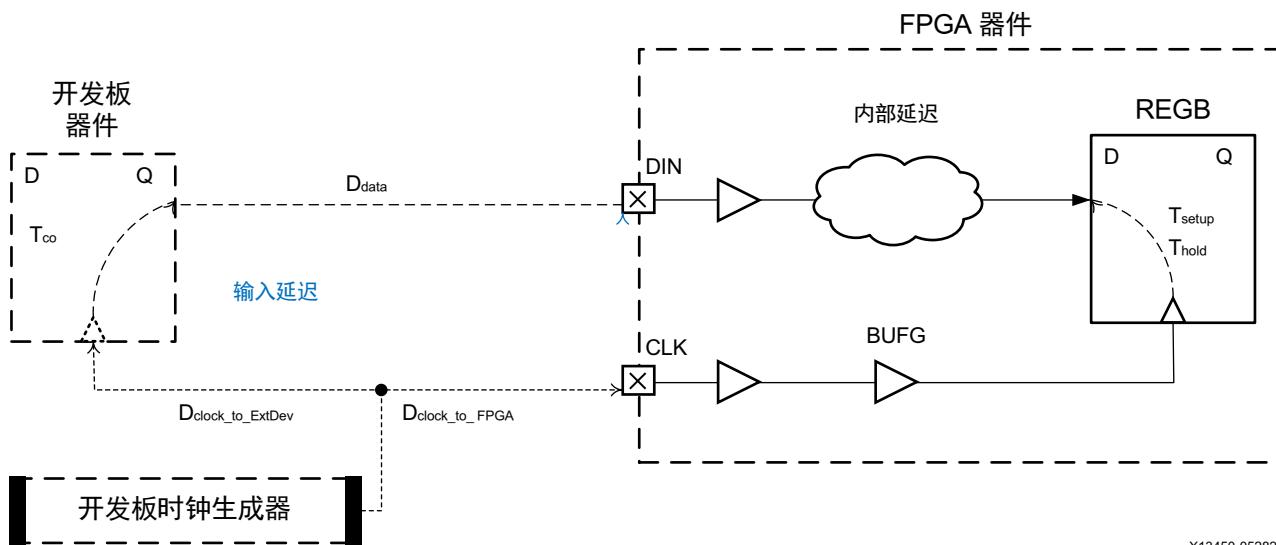
- 为实现最大延迟分析（建立），数据的捕获比单倍数据速率接口的发送沿晚 1 个时钟周期，比双倍数据速率接口的发送沿晚半个时钟周期。
- 为实现最小延迟分析（保持），请在相同时钟沿发送和捕获数据。

如果时钟和 I/O 数据之间的关系必须以不同方式进行时序约束（例如在时钟源同步接口中），那么必须指定不同的 I/O 延迟和附加时序例外。这对应于高级 I/O 时序约束方案。

定义输入延迟

输入延迟定义为与器件接口处的时钟相关的延迟。除非已在参考时钟的源管脚上指定 `set_clock_latency`，否则输入延迟对应于从发送沿到时钟走线、外部器件和数据走线的绝对时间。如果已单独指定期钟时延，即可忽略时钟走线延迟。

图 95：输入延迟计算



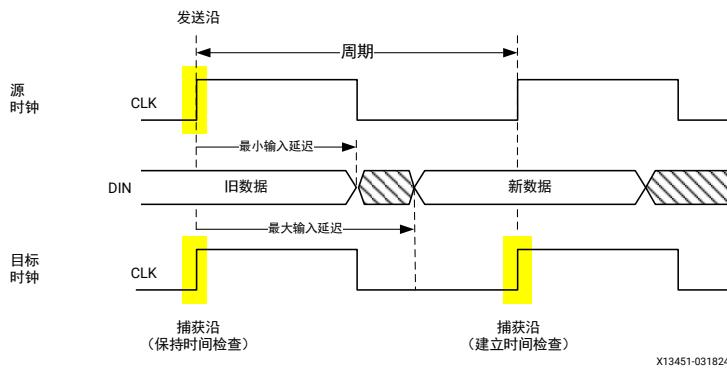
两类分析的输入延迟值为：

```
Input Delay(max) = Tco(max) + Ddata(max) + Dclock_to_ExtDev(max) - Dclock_to_FPGA(min)
Input Delay(min) = Tco(min) + Ddata(min) + Dclock_to_ExtDev(min) - Dclock_to_FPGA(max)
```

下图显示了建立时间（最大值）和保持时间（最小值）分析的输入延迟约束的简单示例，其中假定已在 CLK 端口上定义 sysClk 时钟：

```
set_input_delay -max -clock sysClk 5.4 [get_ports DIN]
set_input_delay -min -clock sysClk 2.1 [get_ports DIN]
```

图 96：解读最小和最大输入延迟

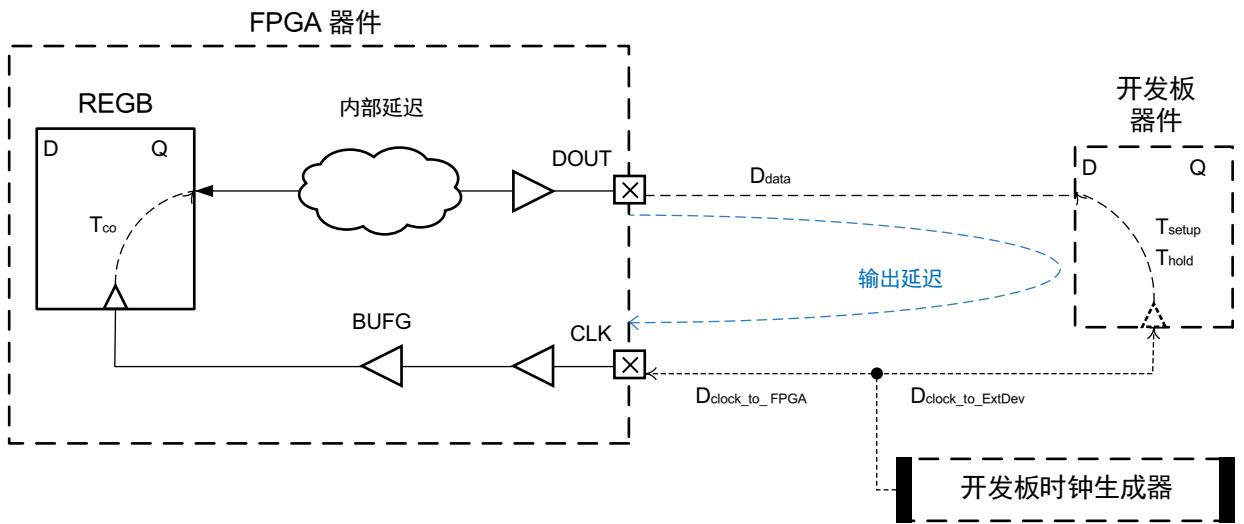


输入延迟为负表示数据到达器件接口的时间早于发送时钟沿。

定义输出延迟

输出延迟与输入延迟类似，区别在于输出延迟表示为了确保在所有情况下均可正常工作，输出路径在器件外部的最短和最长时间。

图 97：输出延迟计算



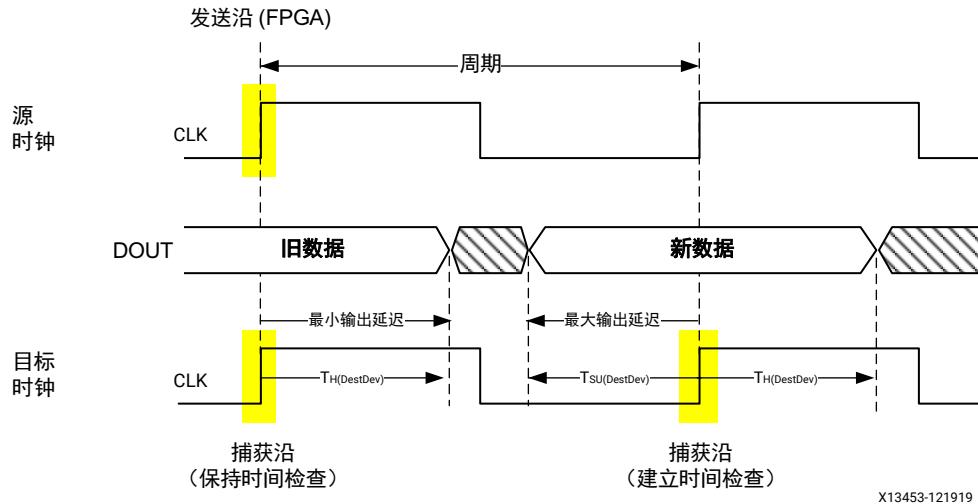
两类分析的输出延迟值为：

```
Output Delay(max) = Tsetup + Ddata(max) + Dclock_to_FPGA(max) -  
Dclock_to_ExtDev(min)  
Output Delay(min) = Ddata(min) - Thold + Dclock_to_FPGA(min) -  
Dclock_to_ExtDev(max)
```

下图显示了建立时间（最大值）和保持时间（最小值）分析的输出延迟约束的简单示例，其中假定在 CLK 端口上已定义 sysClk 时钟：

```
set_output_delay -max -clock sysClk 2.4 [get_ports DOUT]
set_output_delay -min -clock sysClk -1.1 [get_ports DOUT]
```

图 98：解读最小和最大输出延迟



X13453-121919

输出延迟对应于开发板上捕获沿之前的延迟。对于其中时钟和数据开发板走线已实现平衡的常规系统同步接口而言，目标器件的建立时间用于定义输出延迟最大值分析。目标器件保持时间用于定义输出延迟最小值分析。指定的输出延迟最小值表示从设计发出信号开始到在目标器件接口上使用信号进行保持时间分析之前，所发生的最小延迟。因此，块内部的延迟可能小得多。输出延迟最小值为正值表示信号在设计内部可能具有负延迟。因此，输出延迟最小值通常为负值。例如，以下代码示例表示设计内部截至 DOUT 为止的延迟必须至少为 +0.5 ns 才能满足保持时间要求。

```
set_output_delay -min -0.5 -clock CLK [get_ports DOUT]
```

选择参考时钟

根据控制输入或输出端口相关的时序单元的时钟树拓扑结构，必须选择最合适的时钟来定义输入或输出延迟约束。如果 I/O 路径寄存器的时钟是生成时钟，那么通常需要根据基准时钟来定义延迟约束，而基准时钟是在生成时钟上游定义的。这条规则存在部分例外，本节将做出解释。

识别与每个端口相关的时钟

在定义 I/O 延迟约束之前，必须首先识别哪些时钟与每个端口相关。您可使用以下章节中所述方法来识别时钟。

浏览开发板原理图

对于连接到开发板上的另一个器件接口的一组 I/O 端口，可使用同时连接到 AMD 器件和外部器件接口的开发板时钟作为输入或输出延迟约束的参考时钟。要控制相关端口组的时序收敛，必须在外部器件数据手册中验证开发板时钟是否已通过内部变换来实现 I/O 端口的时序收敛，从而确保此设计生成的时钟与 AMD 器件内的时钟相同。

浏览设计板级原理图

对于每个端口，可将路径板级原理图展开至时序单元的第一层，然后沿这些单元的时钟管脚走线回到时钟源。对于连接到高扇出信号线的端口，这种方法不可行。

报告进出端口的时序

无论端口是否已约束，均可使用 `report_timing` 命令识别设计中端口的相关时钟。定义完所有时序时钟后，即可报告进出 I/O 端口的最差路径、创建与报告的时钟相关的 I/O 延迟约束，并为进出设计的其他时钟重新运行相同的时序报告。如果发现端口与多个时钟相关，请创建对应的约束并重复此过程。

例如，`din` 输入端口与设计中的 `clk1` 时钟和 `clk2` 时钟相关：

```
report_timing -from [get_ports din] -sort_by group
```

此报告显示 `din` 端口与 `clk1` 相关。输入延迟约束为（同时适用于该示例中的最小和最大延迟）：

```
set_input_delay -clock clk1 5 [get_ports din]
```

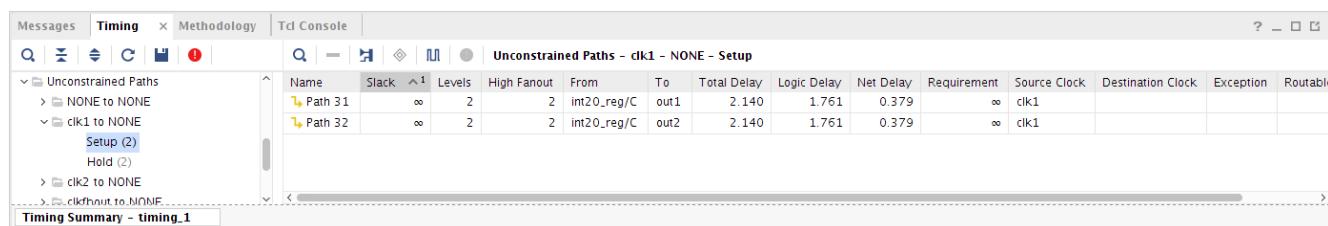
使用先前所用的命令重新运行时序分析，并观察发现 `din` 与 `clk2` 同样有关，原因在于 `-sort_by group` 选项，该选项针对每个端点时钟报告 N 条路径。您可添加对应的延迟约束并重新运行报告以确认 `din` 端口与其他时钟无关。

您还可使用“Timing Summary”（时序汇总）报告并选中 `-report_unconstrained` 选项来运行同样的分析。设计中仅有时钟约束的情况下，“Unconstrained Paths”（未约束的路径）部分显示结果如下：

Unconstrained Path Table		
Path Group	From Clock	To Clock
(none)		
(none)	clk1	
(none)	clk2	
(none)		clk1
(none)		clk2

不含时钟名称（或者在 Vivado IDE 中显示 <NONE>）的字段表示一组路径，这组路径的起点（源时钟）或端点（目标时钟）与时钟无关联。未约束的 I/O 端口归为此类别。您可浏览报告其余部分以检索其名称。例如，在 Vivado IDE 中，通过选择“`clk1` to NONE”类别的“Setup”路径，即可在“To”列中看到由 `clk1` 驱动的端口：

图 99：获取未约束的输出端口的列表



在存储器中添加并应用新约束后，必须重新运行报告以确定哪些端口仍处于未约束状态。对于大多数设计来说，必须增加报告路径的数量，以确保报告中已列出所有 I/O 路径。

使用自动识别的采样时钟

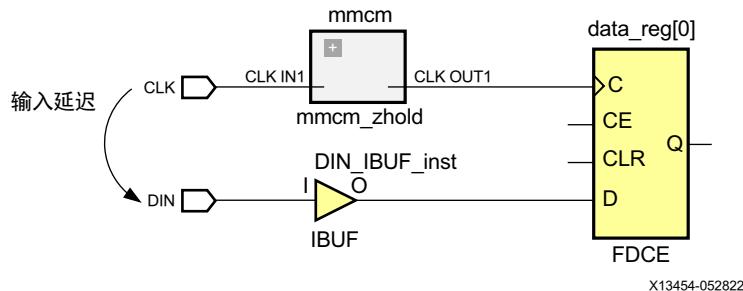
您无需指定相关时钟即可使用 `set_input_delay` 和 `set_output_delay` 约束。Vivado Design Suite 时序引擎将分析设计并自动将每个端口与所有采样时钟关联。随后，通过报告 I/O 路径上的时序，即可查看该工具约束每个 I/O 端口的方式。这样即可便于快速约束设计，但此类通用约束如果过于泛用且无法对硬件实际情况进行准确建模，则可能成为问题。

使用基准时钟

基准时钟（即传入开发板时钟）应在以下情况下使用：当它直接控制 I/O 路径时序单元，而不遍历任何时钟修改块时。I/O 延迟线不能视作为时钟修改块，因为这些延迟线仅影响时钟插入延迟，不影响波形。在 [定义输入延迟](#) 和 [定义输出延迟](#) 中提供的 2 个示例中演示了此情境。大多数情况下，外部器件的接口特性是根据相同开发板时钟定义的。

当以采用零保持时间违例 (ZHOLD) 模式的器件内的 PLL 或 MMCM 补偿基准时钟时，I/O 路径时序单元连接到基准时钟的内部副本（例如，生成时钟）。由于两个时钟的波形完全相同，AMD 建议使用基准时钟作为输入/输出延迟约束的参考时钟。

图 100：时钟路径中存在 ZHOLD MMCM 时的输入延迟



X13454-052822

这些约束与 [定义输入延迟](#) 中提供的示例完全相同，因为 ZHOLD MMCM 充当具有负插入延迟的时钟缓冲器，此插入延迟对应于补偿的量。

使用虚拟时钟

当板载时钟遍历某一时钟修改块以对波形进行变换并补偿整体插入延迟时，建议使用虚拟时钟代替开发板时钟作为输入和输出延迟的参考时钟。虚拟时钟的使用场景主要有以下 3 种：

- 内部时钟与板载时钟具有不同周期：虚拟时钟必须定义为：具有与内部时钟相同的周期和波形。其结果是要求 I/O 路径为常规的单周期路径。
- 对于输入路径来说，内部时钟相比于板载时钟拥有正向移位波形：虚拟时钟的定义类似板载时钟，针对建立时间定义一条从虚拟时钟到内部时钟的多周期路径（双周期）约束。上述约束导致强制执行建立时间时序分析，并且分析要求为 1 个时钟周期加相位移动量。
- 对于输出路径来说，内部时钟相比于板载时钟拥有负向移位波形：虚拟时钟的定义类似板载时钟，针对建立时间定义一条从内部时钟到虚拟时钟的多周期路径（双周期）约束。上述约束导致强制执行建立时间时序分析，并且分析要求为 1 个时钟周期加相位移动量。

综上，使用虚拟时钟需调整默认时序分析，以避免将 I/O 路径作为具有苛刻而又不切实际的要求的时钟域交汇路径来处理。



重要提示！ 当相移导致时钟波形发生修改时，针对具有相移时钟的 I/O 路径只需使用多周期路径即可。将相移添加到时钟修改块的插入延迟并保留时钟波形时，无需使用多周期路径。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。

例如，假设 sysClk 板载时钟频率为 100 MHz，与 MMCM 相乘后生成运行频率为 266 MHz 的 clk266。由 clk266 生成的输出应使用 clk266 作为参考时钟。如果试图使用 sysClk 作为参考时钟（对应 set_output_delay 规格），那么它将显示为异步时钟，并且该路径无法再作为单周期路径进行时序约束。

使用生成时钟

对于输出源同步接口，设计会生成 1 个内部时钟的副本并将其随同数据转发给开发板。如需控制并报告转发时钟与数据之间的相位关系（偏差），即可使用此时钟作为输出数据延迟约束的参考时钟。转发时钟同样能用于系统同步接口的输入与输出延迟约束。

参考时钟上升沿和下降沿

I/O 约束使用的时钟沿必须与连接到器件的外部器件的数据手册一致。默认情况下，`set_input_delay` 和 `set_output_delay` 命令定义 1 个相对于参考时钟上升沿的延迟约束。您必须使用 `clock_fall` 选项来设定 1 个相对于时钟下降沿的延迟。此外，您还可以使用 `add_delay` 选项为相对于时钟上升沿和下降沿的延迟分别指定约束，且在端口上指定第二个约束。

大多数情况下，I/O 参考时钟沿对应于用来锁存或发送器件内 I/O 数据的时钟沿。通过分析 I/O 时序路径，可复查已使用的时钟沿，并核实这些时钟沿与其实际硬件行为是否对应。对于仅与时钟下降沿内部相关的 I/O 路径而言，如果误将时钟上升沿用作为此类路径的参考时钟，那么路径要求为 $\frac{1}{2}$ 周期，这会导致时序收敛更加困难。

验证延迟约束

输入 I/O 时序约束后，复查 I/O 路径上的时序分析方式以及建立时间和保持时间检查的裕量违例数量就显得至关重要。通过使用进出所有端口的时序报告进行建立时间和保持时间分析（即，`delay_type = min_max`）即可验证：

- 是否已将正确的时钟和时钟沿用作延迟约束的参考。
- 是否已使用期望时钟来发送和捕获器件内部的 I/O 数据。
- 是否能通过布局或设置适当的延迟界限配置以合理方式修复违例。如果无法修复，则必须复查约束中输入的 I/O 延迟值，并评估其合理性，以及是否需要修改设计以满足时序要求。

I/O 路径报告命令行示例

```
report_timing -from [all_inputs] -nworst 1000 -sort_by group \
-delay_type min_max
```

```
report_timing -to [all_outputs] -nworst 1000 -sort_by group \
-delay_type min_max
```

I/O 延迟约束错误会导致无法实现时序收敛。实现工具由时序驱动并且致力于对布局布线进行最优化以满足时序要求。如果 I/O 路径要求无法得到满足，并且 I/O 路径在设计中发生最严重的违例，那么总体设计 QoR 将受到影响。

输入至输出馈通路径

有多种等效的方法可用来约束从输入端口到输出端口间的组合路径。

示例 1

使用周期大于或等于馈通路径的目标最大延迟的虚拟时钟，并按如下方式应用最大输入和输出延迟约束：

```
create_clock -name vclk -period 10
set_input_delay -clock vclk <input_delay_val> [get_ports din] -max
set_output_delay -clock vclk <output_delay_val> [get_ports dout] -max
```

其中

```
input_delay_val(max) + feedthrough path delay (max) + output_delay_val(max)
<= vclk period.
```

本例中，仅约束最大延迟。

示例 2

在馈通端口之间使用最小延迟与最大延迟约束组合。示例：

```
set_max_delay -from [get_ports din] -to [get_ports dout] 10
set_min_delay -from [get_ports din] -to [get_ports dout] 2
```

这是同时约束路径上的最小延迟和最大延迟的简单方法。时序分析期间将同时使用相同端口上的所有现有输入和输出延迟约束。因此，这种方法并不常用。

最大延迟通常针对“Slow Timing Corner”（慢速时序角点）进行最优化和报告，而最小延迟则发生在“Fast Timing Corner”（快速时序角点）中。最好对馈通路径延迟约束运行几次迭代，以确认其合理性并确保实现工具可满足这些约束要求，当布局的端口间距离相去较远时尤其如此。

使用 XDC 模板 - 源同步接口

AMD 建议针对源同步接口使用 I/O 约束模板。可采用多种方法来编写源同步约束。Vivado Design Suite 所提供的模板基于默认时序分析路径要求。其语法更简单，但必须调整延迟值来解释执行建立时间分析时为何采用了不同发送沿和捕获沿（1 个周期或 1/2 个周期），而未采用相同的发送沿和捕获沿（0 个周期）。由于时钟沿不直接对应于硬件中的活动时钟沿，因此导致时序报告更难以读取。您可在 Vivado IDE 中通过如下操作导航到这些模板：“Tools” → “Language Templates” → “XDC” → “Timing Constraints” → “Input Delay Constraints” → “Source Synchronous”（工具 > 语言模板 > XDC > 时序约束 > 输入延迟约束 > 源同步）。

定义时钟组和 CDC 约束

默认情况下，Vivado IDE 用于对设计中所有时钟之间的路径进行时序约束。您可使用以下约束来修改此默认行为：

- `set_clock_groups`: 禁用您识别的时钟组之间的时序分析，但不禁用同一个组中的时钟之间的时序分析。
- `set_false_path`: 仅禁用由 `-from` 和 `-to` 选项所指定的方向上的时钟之间的时序分析。

在某些情况下，您可能想要对时钟域交汇 (CDC) 的一条或多条路径使用以下约束来限制时延或总线偏差：

- `set_max_delay -datapath_only`: 对异步 CDC 路径设置最大延迟约束，以限制时延。

注释：如果在时钟组之间或者相同 CDC 路径上已存在时钟组或伪路径约束，那么将忽略最大延迟约束。因此，重要的是完整复查所有时钟对之间的每条路径，然后再逐一选择 CDC 时序约束，以避免约束冲突。



建议：AMD 还建议运行 `report_methodology` 以确认何时 `set_clock_groups` 或 `set_false_path` 约束将覆盖 `set_max_delay -datapath_only` 约束。

- `set_bus_skew`: 使用总线偏差代替时延来约束异步 CDC 路径之间的一组信号。



提示：您还可通过 Vivado IDE 来设置总线偏差约束。在“Timing Constraints”（时序约束）窗口中，展开“Assertions”（断言），然后双击“Set Bus Skew”（设置总线偏差）。

相关信息

[运行 Report Methodology](#)

检查时钟交互

相互间含逻辑路径的时钟需进行时序约束。可能的时钟关系包括：同步、异步和专属。

同步

当 2 个时钟具有固定相位关系时，时钟关系即为同步。共享以下对象的 2 个时钟即可视为具有固定相位关系：

- 公用电路（公共节点）
- 基准时钟（相同初始相位）

异步

当时钟不具有固定相位关系时，时钟关系即为异步。满足以下任一条件的时钟之间即为异步关系：

- 在设计中不共享任何公用电路，并且不具有公用基准时钟。
- 在 1000 个周期（不可扩展）内不具有公共周期，并且时序引擎无法通过正确的时序约束将其组合在一起。
- 具有公用时钟，但不共享公共节点。
- 其所属拓扑结构无法通过时钟自动衍生流程来确保已知的相位关系。

如果 2 个时钟同步，但其公共周期非常短，那么建立路径要求将过于苛刻，导致无法满足时序要求。AMD 建议您将 2 个时钟作为异步来处理，并实现安全的异步 CDC 电路。

专属

如果时钟关系在同一个时钟树上传输并到达相同的时序单元时钟管脚，但无法以物理方式同时激活，那么此时钟关系即为专属关系。

时钟对分类

可使用“Clock Interaction”（时钟交互）和“Check Timing”（检查时序）报告对时钟对进行分类。

“Clock Interaction” 报告

“Clock Interaction”（时钟交互）报告可提供有关如何对 2 个时钟一起定时的高层次综述：

- 2 个时钟是否具有公用基准时钟？正确定义时钟时，设计中相同来源的所有时钟即共享相同的基准时钟。
- 2 个时钟是否具有公共周期？当时序引擎无法判定最消极的建立或保持关系时，这会显示在建立或保持路径要求列中，该列不可展开。
- 时钟组或时序例外约束是否部分或完全覆盖 2 个时钟之间的路径？
- 2 个时钟之间的建立路径要求是否极为苛刻？当 2 个时钟同步，但其周期未指定为精确倍数关系（例如，由于舍入）时，会出现此问题。经过多个时钟周期后，边沿可能出现偏离，导致最差情况时序要求变得极为苛刻。

“Check Timing” 报告

“Check Timing”（检查时序）报告 (multiple_clock) 可识别多个时钟连接到的时钟管脚，而在这些时钟之间尚未定义 set_clock_groups 或 set_false_path 约束。

约束专属时钟组

您可使用常规时序或时钟网络报告来检查时钟路径，并识别如下情况：在相同时钟树上传输 2 个时钟，以及在时序路径中同时使用 2 个时钟（在该时序路径中，起点和端点时钟管脚连接到相同时钟树）。此分析任务相当耗时。您可改为查看“检查时序”报告的 `multiple_clock` 部分。该部分会返回包含时钟管脚及其相关时序时钟的列表。

根据时钟树拓扑结构，您必须应用以下段落中所述的不同约束。

相同时钟源上定义的重叠时钟

在相同网表对象上使用 `create_clock -add` 命令定义两个时钟，并表示单一应用的多种模式时，会出现此情况。在此情况下，可在两个时钟之间安全应用时钟组约束。例如：

```
create_clock -name clk_mode0 -period 10 [get_ports clkin]
create_clock -name clk_mode1 -period 13.334 -add [get_ports clkin]
set_clock_groups -physically_exclusive -group clk_mode0 -group clk_mode1
```

如果 `clk_mode0` 和 `clk_mode1` 时钟生成其他时钟，那么还需对其生成时钟应用相同的约束，操作方式如下所述：

```
set_clock_groups -physically_exclusive \
-group [get_clocks -include_generated_clock clk_mode0] \
-group [get_clocks -include_generated_clock clk_mode1]
```

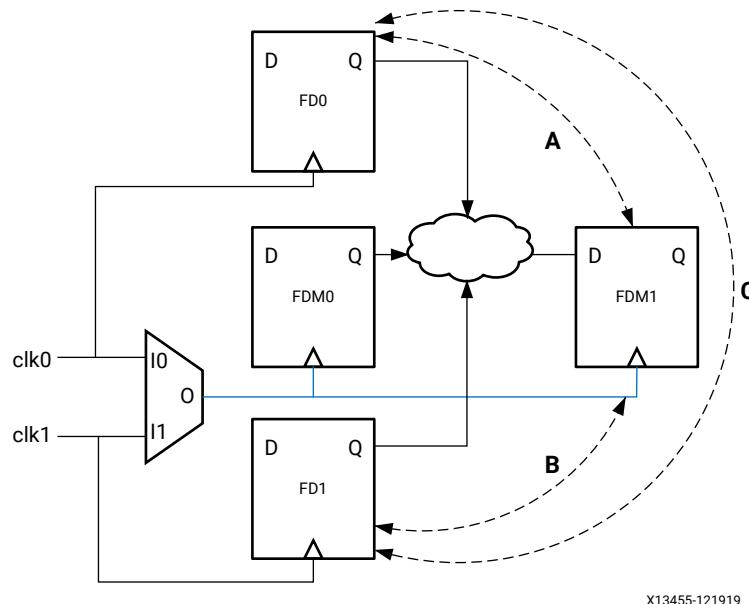
由时钟多路复用器驱动的重叠时钟

如果有 2 个或更多个时钟驱动到某 1 个多路复用器（或者更普遍的情况下，驱动到组合单元内）中，这些时钟全部都能顺利完成传输，并在单元扇出上重叠。实际上，每次只能传输 1 个时钟，但时序分析允许同时报告多种时序模式。

因此，您必须审查 CDC 路径，并添加新约束，以忽略部分时钟关系。正确的约束取决于设计中时钟交互的方式和位置。

下图演示了 2 个时钟驱动到同一个多路复用器中的示例，并演示了在此多路复用器前后这 2 个时钟之间可能发生的交互。

图 101：多路复用时钟



X13455-121919

- 路径 A、B 和 C 都不存在的情况

`clk0` 和 `clk1` 仅在多路复用器 (FDM0 和 FDM1) 的扇出中交互。毋庸置疑，时钟组约束可直接应用到 `clk0` 和 `clk1` 中。

```
set_clock_groups -logically_exclusive -group clk0 -group clk1
```

- 仅存在路径 A、B 或 C 之一的情况

`clk0` 和/或 `clk1` 与多路复用时钟直接交互。为了保留时序路径 A、B 和 C，无法直接向 `clk0` 和 `clk1` 直接应用约束。而是改为必须将其应用于多路复用器的扇出中需要额外的时钟定义的时钟部分。

```
create_generated_clock -name clk0mux -divide_by 1 \
-source [get_pins mux/I0] [get_pins mux/O]

create_generated_clock -name clk1mux -divide_by 1 \
-add -master_clock clk1 \
-source [get_pins mux/I1] [get_pins mux/O]

set_clock_groups -physically_exclusive -group clk0mux -group clk1mux
```

对异步时钟组和时钟域交汇进行约束

在“Clock Interaction”（时钟交互）报告中可快速明确异步关系：无公用基准时钟的时钟对或者无公共周期（未扩展）的时钟对。即使时钟周期相同，从不同时钟源生成的时钟仍为异步关系。必须仔细审查异步“Clock Domain Crossing (CDC)”（时钟域交汇 (CDC)）路径以确保这些路径使用的同步电路正确，此类同步电路不依赖时序正确性，并且可以最大限度降低发生亚稳态的概率。异步 CDC 路径通常具有较高的偏差要求和/或不现实的路径要求。因此不应使用默认时序分析来对其进行时序约束，此分析无法证明其能否在硬件中正常工作。

Report CDC

Report CDC (`report_cdc`) 命令可执行设计中时钟域交汇的结构分析。您可使用此信息来识别潜在不安全的 CDC，此类 CDC 可能导致亚稳态或数据一致性问题。Report CDC 类似于“Clock Interaction”（时钟交互）报告，但 Report CDC 侧重于结构和相关的时序约束。Report CDC 不提供时序信息，因为时序裕量对于跨异步时钟域的路径没有意义。

Report CDC 可识别如下最常见的 CDC 拓扑结构：

- 单位同步装置
- 多位总线同步装置
- 异步复位同步装置
- 由 MUX 和 CE 控制的电路系统
- 同步装置前组合逻辑
- 多时钟扇入到同步装置
- 扇出到目标时钟域

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`。

应采用特定约束以避免异步时钟域交汇上执行默认时序分析。

相关信息

[双向时钟间的全局约束](#)

[对应各 CDC 路径的约束](#)

双向时钟间的全局约束

如果无需限制最大时延，即可使用时钟组。以下是忽略 `clkA` 与 `clkB` 之间的路径的示例：

```
set_clock_groups -asynchronous -group clkA -group clkB
```

当 2 个主时钟及其相应的生成时钟构成 2 个异步域，并且这 2 个异步域之间的所有路径均已正确完成同步时，即可立即对多个时钟应用时钟组约束：

```
set_clock_groups -asynchronous \
-group {clkA clkA_gen0 clkA_gen1} \
-group {clkB clkB_gen0 clkB_gen1}
```

或者直接执行：

```
set_clock_groups -asynchronous \
-group [get_clocks -include_generated_clock clkA] \
-group [get_clocks -include_generated_clock clkB]
```

对应各 CDC 路径的约束

如果 CDC 总线使用格雷编码（例如，FIFO）或者如果需要限制 1 个或多个信号上的 2 个异步时钟之间的时延，则必须使用 `set_max_delay` 约束及 `-datapath_only` 选项来忽略这些路径上的时钟偏差和抖动，并覆盖时延要求的默认路径要求。通常使用源时钟周期作为最大延迟值就足够了，这只是为了确保在任意给定时间，CDC 路径上最多仅存在一项数据。

当时钟周期之间的比率较高时，选择源时钟周期和目标时钟周期的最小值同样足以降低传输时延。简单标准的异步 CDC 路径的源时序单元与目标时序单元之间不应存在任何逻辑，因此实现工具很容易就可以满足“Max Delay Datapath Only”（仅最大延迟数据路径）约束。

某些异步 CDC 路径要求在总线的各个位之间施加偏移控制，而无需对总线时延施加约束。使用总线偏移约束可防止接收时钟域在同一时钟沿上锁存总线的多个状态。您可使用 `set_bus_skew` 命令来对总线设置总线偏移约束。例如，您可将 `set_bus_skew` 应用于使用格雷编码代替“Max Delay Datapath Only”（仅最大延迟数据路径）约束的 CDC 总线。欲知详情，请访问此[链接](#)以参阅《Vivado Design Suite 用户指南：使用约束》(UG903) 中的相应内容。

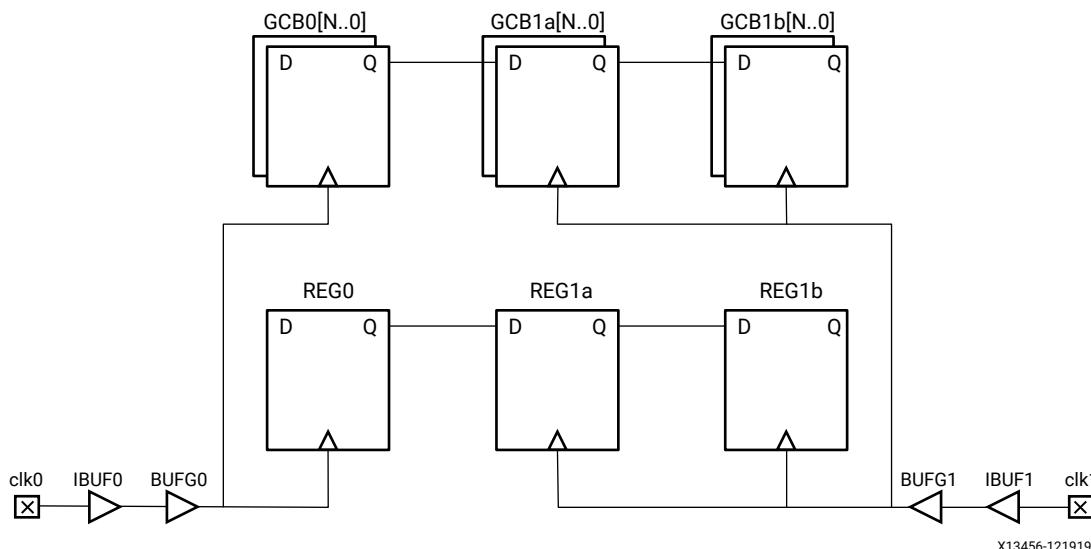
对于不需要时延控制的路径，您可定义 1 个点对点伪路径约束。

时钟例外相对于 `set_max_delay` 的优先顺序

编写 CDC 约束时，请验证是否遵循相应的优先顺序。如果在 2 个时钟之间至少 1 条路径上使用 `set_max_delay - datapath_only`，那么无法在相同时钟之间使用 `set_clock_groups` 约束，并且只能在 2 个时钟之间的其他路径上使用 `set_false_path` 约束。

在下图中，时钟 `clk0` 的周期为 5 ns，并且与 `clk1` 之间处于异步关系。从 `clk0` 域到 `clk1` 域存在两条路径。第 1 条路径为 1 位数据同步。第 2 条路径为多位格雷编码总线传输。

图 102：2 个异步时钟之间的多次交互



设计师判定格雷编码总线传输需要“Max Delay Datapath Only”（仅最大延迟数据路径）来限制比特间延迟变动，因此无法在时钟之间直接使用“Clock Group”（时钟组）或“False Path”（伪路径）约束。而改为必须定义以下 2 个约束：

```
set_max_delay -from [get_cells GCB0[*]] -to [get_cells [GCB1a[*]] \
-datapath_only 5
set_false_path -from [get_cells REG0] -to [get_cells REG1a]
```

无需设置从 `clk1` 到 `clk0` 的伪路径，因为在此示例中不含任何路径。

指定时序例外

时序例外用于修改对特定路径执行时序分析的方式。默认情况下，时序引擎假定所有路径都应通过单一建立时间分析周期要求来完成时序约束，以便覆盖大部分消极的时钟设置场景。对于某些路径，并非如此。以下提供一些示例：

- 由于时钟间缺乏固定的相位关系，导致无法安全完成异步 CDC 路径的时序约束。此类状况应予以忽略（时钟组，伪路径），或者只需设置数据路径延迟约束（仅最大延迟数据路径）即可
- 时序单元发送沿和捕获沿并非在每个时钟周期内都处于活动状态，因此可相应降低路径要求（多周期路径）
- 路径延迟要求需收紧，以增加硬件中的设计裕度（最大延迟）
- 通过组合单元的路径为静态路径，无需时序约束（伪路径，案例分析）
- 应仅限对多路复用器驱动的特定时钟执行分析（案例分析）。

无论在任何情况下，都必须谨慎使用时序例外，并且不得为了隐藏实际时序问题而添加例外。

时序例外准则

请尽量限制使用的时序例外的数量，并使时序例外尽可能保持简单。否则，您将面临下面两大挑战：

- 如果过多使用例外，实现编译时间将显著增加，当这些例外与大量网表对象相关联时尤其如此。
- 当多个例外覆盖相同路径时，约束调试会变得极为复杂。
- 对信号施加约束会阻碍该信号的最优化。因此无论是包含不必要的例外还是在例外命令中包含不必要的点，都会妨碍信号最优化。

以下是可能会对运行时间产生不利影响的时序例外示例：

```
set_false_path -from [get_ports din] -to [all_registers]
```

- 如果 `din` 端口没有输入延迟，那么它将不受约束。因此无需添加伪路径。
- 如果 `din` 端口仅供给时序元件，那么无需对时序单元显式指定伪路径。按如下方法编写此约束更有效：

```
set_false_path -from [get_ports din]
```

- 如果需要伪路径，但从 `din` 端口到设计中的任意时序单元之间仅存在几条路径，那么约束可以更明确（`all_registers` 可能会返回数千个单元，这取决于设计中使用的寄存器数量）：

```
set_false_path -from [get_ports din] -to [get_cells blockA/config_reg[*]]
```

时序例外优先级规则

时序例外需遵循严格的优先级规则。最重要的规则包括：

- 约束越具体，优先级越高。例如：

```
set_max_delay -from [get_clocks clkA] -to [get_pins inst0/D] 12
set_max_delay -from [get_clocks clkA] -to [get_clocks clkB] 10
```

第一项 `set_max_delay` 约束优先级更高，因为 `-to` 选项使用管脚，这比时钟更为具体。

- 例外优先级如下所示：

1. `set_false_path`
2. `set_max_delay` 或 `set_min_delay`

3. set_multicycle_path

`set_clock_groups` 命令不视为时序例外，即使它等同于 2 个时钟之间的 2 条 `set_false_path` 命令也是如此。它的优先级高于时序例外。

`set_case_analysis` 命令和 `set_disable_timing` 命令用于禁用特定设计部分上的时序分析。其优先级高于时序例外。

如需了解有关 XDC 优先级的详细信息，请访问此[链接](#)以参阅《Vivado Design Suite 用户指南：使用约束》(UG903) 中的相应内容。

添加伪路径约束

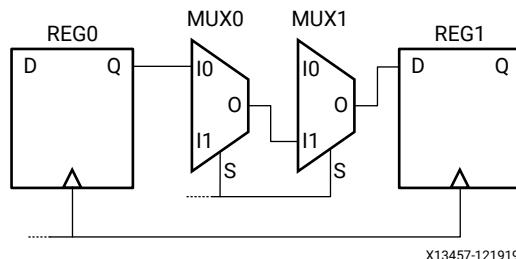
可向时序路径添加伪路径例外以忽略这些路径上的时序裕量计算。通常，要证明路径无需满足时序要求亦可正常运行是极为困难的，即使使用仿真工具也是如此。AMD 通常不建议使用伪路径，除非已正确评估并判断可接受与之关联的风险。

用例

伪路径约束的典型用例为：

- 忽略从不活动的路径上的时序。例如，如果某条路径穿过 2 个多路复用器，这 2 个多路复用器由于所选管脚连接，导致绝不允许在同一时钟周期内传输数据，则忽略该路径。

图 103：不能激活的路径



```
set_false_path -through [get_pins MUX0/I0] -through [get_pins MUX1/I1]
```

- 忽略异步 CDC 路径上的时序。
- 忽略设计中的静态路径。有些寄存器在应用的初始化阶段接收 1 个值后，就不再切换。当这些寄存器出现在设计的关键路径上时，可以忽略其时序，以放宽对实现工具的约束，从而有助于实现时序收敛。仅定义伪路径约束始于静态寄存器即可，无需明确指定路径端点。例如，通过添加如下伪路径约束，便可忽略从 32 位配置寄存器 `config_reg[31..0]` 到设计其余部分的路径：

```
set_false_path -from [get_cells config_reg[*]]
```

对实现的影响

所有实现步骤都对伪路径时序例外比较敏感。

对综合的影响

伪路径约束受综合支持，并且仅影响最大延迟（建立/恢复时间）路径最优化。对于设计中可安全忽略其中时序的部分以及无数据路径延迟要求的异步 CDC 路径，建议采用伪路径时序例外。

对实现的影响

所有实现步骤都对伪路径时序例外比较敏感。

添加最小和最大延迟约束

最小和最大延迟例外用于覆盖分别对应于保持/移除检查和建立/恢复检查的默认路径要求，方法是将发送沿时间和捕获沿时间替换为约束中的延迟数值。

用例

以下描述了使用最小或最大延迟约束的部分常见原因：

- 通过收紧建立/恢复路径要求，对部分设计路径进行过约束。

这有助于强制要求逻辑最优化或布局工具全力以赴处理部分关键路径单元，从而提升布线器的灵活性以便稍后移除最大延迟约束后满足时序要求。

- 替换多周期约束。

对于每 N 个时钟周期存在活动的发送沿和捕获沿的路径，如需在此类路径上设置宽松的建立时间要求，此方法有效，但并不推荐使用。但只有采用这种方法才能在任一时钟周期内对多周期路径进行短时间过约束，从而帮助在布线步骤阶段实现时序收敛。例如，如果某条多周期约束为 3 的路径于布线后成为最差的违例路径并以数百 ps 的差距未能满足时序要求，

那么在最优化和布局期间原多周期路径约束将替换为如下约束，其中，14.5 对应 3 个时钟周期（每个周期 5 ns）减去 500 ps（对应于期望的裕度）：

```
set_max_delay -from [get_pins <startpointCell>/C] \
-to [get_pins <endpointCell>/D] 14.5
```

- 约束异步 CDC 路径上的最大数据路径延迟

如需了解此技巧的详情，请参阅 [定义时钟组和 CDC 约束](#)。

不建议使用 `set_min_delay` 约束对路径强制执行额外的延迟插入，并且此方法并不常用。在裕量为正值时，保持或移除的默认最小延迟要求通常足以确保硬件正常运作。

对综合的影响

综合支持 `set_max_delay` 约束（包括 `-datapath_only` 选项）。`set_min_delay` 约束将被忽略。

对实现的影响

`set_max_delay` 约束会取代建立路径要求并影响整个实现流程。`set_min_delay` 约束会取代保持路径要求，仅当引入该约束以修复保持时间违例时才会影响布线器行为。

避免路径分段

仅当针对 `set_max_delay` 和 `set_min_delay` 命令的 `-from` 或 `-to` 选项指定的起点或端点无效时，才会引入路径分段。当 `set_max_delay` 在路径上引入路径分段后，将不再发生默认保持时间分析。如果还要对保持时间分析进行约束，请使用 `set_min_delay` 来对该路径进行约束。此规则同样适用于建立时间分析相关的 `set_min_delay` 命令。

路径分段只能由专家级用户使用，因为它会改变时序分析的基础：

- 路径分段会对分段路径上的时钟偏差计算进行拆分。

- 路径分段拆分的路径数量比用于分段的 `set_max_delay` 或 `set_min_delay` 命令约束的路径数量更多。

有效的起点和端点

应用约束时，工具在 log 日志文件中报告路径分段。您必须使用有效的起点和端点来避免此类问题：

- 起点：时钟、时钟管脚、时序单元（表示单元的有效起点管脚）、输入或输入输出端口。
- 端点：时钟、时序单元的输入数据管脚、时序单元（表示单元的有效端点管脚）、输出或输入输出端口。

如需了解有关路径分段的详细信息，请访问此[链接](#)以参阅《Vivado Design Suite 用户指南：使用约束》(UG903) 中的相应内容。

添加多周期路径约束

多周期路径例外处理必须反映设计的功能性，并且在源时钟和/或目标时钟上，只要路径并非在每个周期内都包含活动时钟沿，就必须应用此例外处理。根据使用 `-start` 选项时的源时钟或使用 `-end` 选项时的目标时钟，路径倍频器以时钟周期的形式来表示。这样尤其便于独立于时钟周期值来修改起点和端点之间的建立时间和保持时间关系。

保持时间关系始终与建立时间关系相关联。由此导致大多数情况下，修改建立时间关系后还需单独调整保持时间关系。因此才需要使用含 `-hold` 选项的第二个约束。此规则的主要例外是相移时钟之间的同步 CDC 路径：只需修改建立时间即可。

放宽建立时间要求，同时保持时间要求保留不变

当源时序单元和目标时序单元受时钟使能信号控制并且此信号每 N 个周期就会激活时钟时，此状况就会发生。以下示例中，时钟使能每 3 个周期就会激活一次，并且起点和端点时钟相同：

图 104：使用相同时钟信号启用的触发器

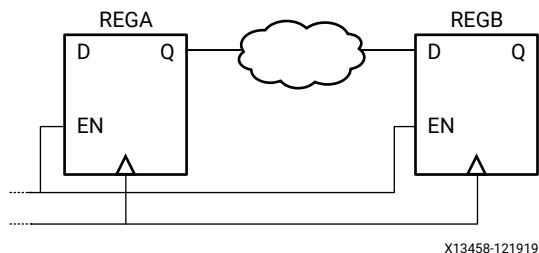
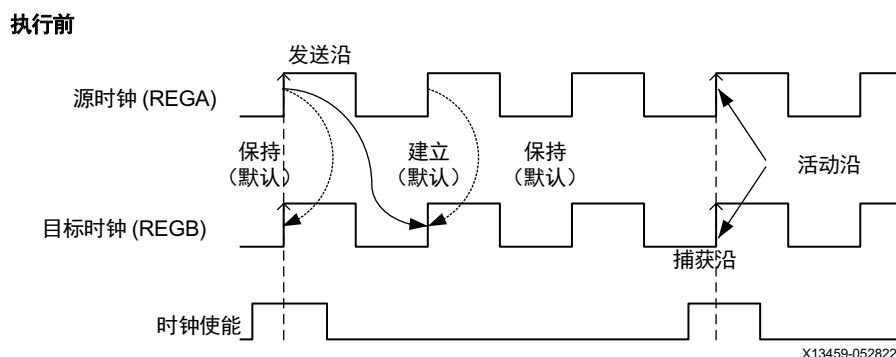


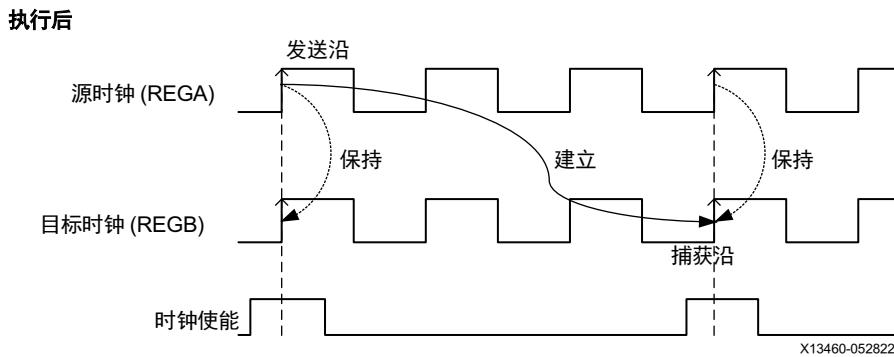
图 105：建立/保持检查的时序图



约束：

```
set_multicycle_path -from [get_pins REGA/C] -to [get_pins REGB/D] -setup 3
set_multicycle_path -from [get_pins REGA/C] -to [get_pins REGB/D] -hold 2
```

图 106：多周期规格应用后修改的建立/保持检查



注释：运行第 1 条命令后，当建立捕获沿移至第 3 沿（即，距离其默认位置达 2 个周期），保持沿同样移动 2 个周期。第 2 条命令用于使保持沿再次移动 2 个周期（反向）从而返回至其原始位置。

如需了解有关其他公用多周期路径场景（例如，同步时钟之间的相移和多周期路径）的更多信息，请访问此[链接](#)以参阅《Vivado Design Suite 用户指南：使用约束》(UG903) 中的相应内容。



重要提示！ 当时钟相移不修改时钟波形，而是改为包含在时钟修改块的插入延迟中时，无需添加 1 个仅限建立的多周期路径用于对往来时钟的路径进行正确的时序约束。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。

对综合与实现的影响

综合步骤支持 `set_multicycle_path` 约束，该约束可将每个时钟周期内功能上处于不活动状态的长路径约束加以放宽，从而大幅提升时序 QoR（仅针对建立时间）。

就综合而言，多周期路径例外可帮助由实现时序所驱动的算法集中处理真正关键的路径。只有在布线过程中，保持时间要求才至关重要。如果通过 `set_multicycle_path` 约束调整建立关系而不调整其对应的保持关系，那么最差保持要求一旦超过 2 或 3 ns，就会变得难以满足。这种情况会对建立裕量产生不利影响，因为在修复保持违例时布线器插入了附加延迟。

常见错误

以下是必须避免的常见错误：

- 多周期路径功能并非在每个时钟周期内都处于活动状态，在此情况下放宽“建立时间”要求，但却不将“保持时间”调整恢复到相同的发送沿和捕获沿。
- 保持要求会变得非常高（大多数情况下为至少 1 个时钟周期），且无法满足。
- 在设计中错误的时间点之间设置多周期路径例外处理。

当假定从起点单元到端点单元只有一条路径时会发生上述情况。在有些情况下也不尽然。端点单元可包含多个数据输入管脚，包括时钟使能和复位管脚，这些管脚在至少 2 个连续时钟沿上处于活动状态。

因此，AMD 建议指定端点管脚而非单元（或时钟）。例如，端点单元 REGB 有 3 个输入管脚：C、EN 和 D。只有 REGB/D 管脚需要由多周期路径例外处理进行约束，EN 管脚不需要，因为在每个时钟周期它都会发生变化。如果将约束连接至单元而不是管脚，那么包括 EN（时钟使能）管脚在内的所有有效的端点管脚都在约束的考虑范围内。

为安全起见，AMD 建议您始终使用如下语法：

```
set_multicycle_path -from [get_pins REGA/C] \ -to [get_pins REGB/D] -  
setup 3 set_multicycle_path -from [get_pins REGA/C] \ -to [get_pins  
REGB/D] -hold 2
```

其他高级时序约束

通过设置一些其他时序约束可忽略和修改默认时序分析，欲知详情，请参阅以下章节。

案例分析

案例分析命令常用于通过在逻辑中设置约束来描述设计中的功能模式，例如，要使用哪些配置寄存器。此命令可应用于输入端口、信号线、层级管脚或叶节点输入管脚。此常量值通过逻辑传输，用于关闭从不活动的任何路径上的分析。其作用与伪路径例外的工作方式相似。

最常见的示例是将多路复用器选择管脚设置为 0 或 1，这样仅允许 1 个或 2 个多路复用器输入通过此管脚传输。以下显示的是在穿过 mux/S 和 mux/I1 管脚的路径上关闭分析的示例：

```
set_case_analysis 0 [get_pins mux/S]
```

禁用时序

禁用时序命令可关闭时序数据库中的时序 arc，从而完全阻止通过此 arc 进行任何分析。禁用的时序 arc 可通过 report_disable_timing 命令来报告。



注意！ 请谨慎使用禁用时序命令。它会造成大量路径中断！

数据检查

set_data_check 命令用于在设计中 2 个管脚之间设置建立时序或保持时序检查的等效检查。例如，此约束可用于报告异步接口上的时序。实现工具会忽略此命令，它仅用于时序报告，通常仅限专家级用户使用。

最大时间借用量

set_max_time_borrow 命令用于设置锁存器可从下一阶段（锁存后逻辑）借用的最大时间量，以及提供给其上一阶段（锁存前逻辑）的最大时间量。通常不建议使用锁存器，因为在硬件中难以对其进行测试和确认。该命令应由专家级用户使用。

定义功耗和散热约束

在 Vivado 工具或 Vitis 环境中开发设计时，您必须确保自己的设计在供电和热处理解决方案的约束范围内，这些约束通常是根据 Xilinx Power Estimator (XPE) 或电源设计管理器 (PDM) 工具（从 www.amd.com/power 下载）工具所确定的早期功耗估算来判定的。由于更改供电和热处理解决方案代价不菲，因此确保设计正确完成约束至关重要。

AMD 建议，至少应用总功耗预算、最大工艺和最差情况结温，使用以下 XDC 约束来创建最差情况功耗分析：

```
set_operating_conditions -design_power_budget <Power in Watts>
set_operating_conditions -process maximum set_operating_conditions -
junction_temp <Max Tj based on Temp Grade>
```



电源/功耗提示：为了执行最差情况功耗估算，截至热处理解决方案的 Theta Ja (Θ_{Ja}) 已知之前，AMD 建议将 Tj 设置为目标温度范围允许的最大值。Theta Ja 可基于热处理仿真结果按如下公式来计算： $\Theta_{Ja} = (T_j - T_a) / P_d$ 。单位为每瓦摄氏度 ($^{\circ}\text{C}/\text{W}$)。

已知热处理设计的 Theta Ja 之后，即可实现最准确的功耗估算。您可使用以下约束将 Theta Ja 和应用支持的最高环境温度 (Ta) 应用于 report_power 以替换结温设置。使用这些约束即可允许 report_power 更准确地估算结温，从而提供更准确的静态功耗估算。

```
set_operating_conditions -design_power_budget <Power in Watts>
set_operating_conditions -process maximum set_operating_conditions -
ambient_temp <Max Supported by Application> set_operating_conditions -
thetaja <Increase in Tj for every W dissipated C/W>
```

此外，您可以使用 XDC 约束来指定供电设计。使用此方法即可允许 report_power 报告总功耗裕度、检查电源轨的功耗估算并基于指定的估算和电源轨整合来报告裕度。如需了解有关这些约束的更多信息，请参阅《Vivado Design Suite 用户指南：功耗分析与优化》([UG907](#))。

```
create_power_rail <power rail name> -power_sources {supply1, supply2, ...}
add_to_power_rail <power rail name> -power_sources {supply1, supply2, ...}
set_operating_conditions -supply_current_budget {<supply rail name>
<current budget in Amp>} -voltage {<supply rail name> <voltage>}
```



电源/功耗提示：请确保使用功耗报告文本获取最详细的电源轨约束报告。

定义物理约束

物理约束用于控制布局规划、特定布局、I/O 分配、布线器以及类似功能。确保已为每个管脚指定 I/O 位置和标准。以下用户指南中涵盖了有关物理约束的信息：

- 如需了解有关锁定布局和布线（包括宏的相关布局）的信息，请参阅《Vivado Design Suite 用户指南：使用约束》([UG903](#))。
- 如需了解布局规划相关信息，请参阅《Vivado Design Suite 用户指南：设计分析与收敛技巧》([UG906](#))。
- 如需了解配置相关信息，请参阅《Vivado Design Suite 用户指南：烧录和调试》([UG908](#))。

设计实现

选定器件、选择并配置 IP 且编写 RTL 和约束条件后，下一步即为实现。实现通过综合和布局布线来编译设计，然后生成用于对器件进行编程的文件。实现过程可能包含一些迭代循环。本章将介绍各个实现步骤，并着重强调需特别注意的事项，同时给出识别和消除具体瓶颈的要诀和技巧。



重要提示！ 您必须定期确认综合与实现均正常完成，不含任何错误，仅含最少量的时序违例，然后才能为 AMD Vitis™ 工具添加新的块或生成平台。

注释：这些实现步骤是在 Vitis 环境流程中自动运行的。您可按本章中所述方法，使用 Vitis 命令行选项和配置文件来改善时序收敛和可实现的最高时钟频率。如需了解更多信息，请参阅《[Vitis 统一软件平台文档](#)》。

运行综合

综合步骤将采用 RTL 和时序约束，并生成功能上等同于 RTL 的优化网表。通常，综合工具可以采用任何合规 RTL，并为其创建逻辑。综合需要现实的时序约束。

如需了解有关综合的更多信息，请参阅以下资源：

- 《Vivado Design Suite 用户指南：综合》([UG901](#))
- [Vivado Design Suite QuickTake 视频：设计流程概述](#)

相关信息

[设计约束](#)

[设计基线设定](#)

综合流程

在 AMD Vivado™ Design Suite 中，可运行以下章节中所述的综合流程，这些流程各有各的利弊取舍。

全局综合

在全局综合流程中，整个设计通过单次运行来完成综合，其优势如下：

- 允许综合工具执行最大程度地最优化。由于综合工具已发现整个设计，该工具可在层级间进行最优化，这是其他流程无法做到的。
- 简化综合运行后的分析操作。

此流程的缺点在于编译时间较长。每次运行综合后，都会重新运行整个设计。但可通过使用增量综合来缓解此缺点的不利影响。

注释：如果设计包含 XDC 约束，那么必须将对象引用到顶层设计。

相关信息

增量综合

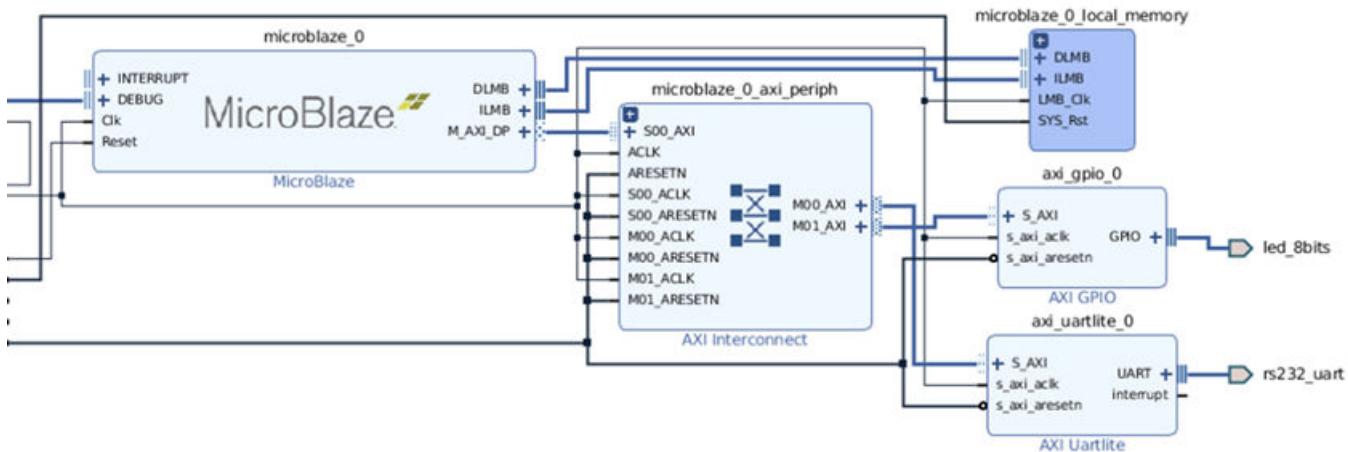
块设计综合

块设计综合流程支持您使用定制 IP 和 AMD IP 创建复杂系统。在此流程中，将使用 Vivado IP integrator 创建块设计 (BD) 文件。AMD IP 或定制 IP 将被添加到此 BD 文件中并作为系统进行连接。此流程具有以下优点：

- 将大量功能封装到小型设计中。
- 支持集中精力处理整个系统，而不是系统的各部分。
- 简化并加速设计的设置和综合。

下图显示了块设计示例。

图 107：块设计示例



创建块设计时，可使用非关联 (OOC) 综合模式或全局综合模式来运行综合。如果使用非关联综合模式，那么块设计将独立于设计其余部分进行单独综合。这样可在修改 BD 文件外部的层级时加速完成重新综合。如果使用全局综合模式，则每次都会编译并综合整个设计。全局综合模式更易于设置，因为它在全局级别设置约束。但使用该模式可能导致重新综合时运行时间更长。您可使用增量综合来缩短运行时间。

非关联综合

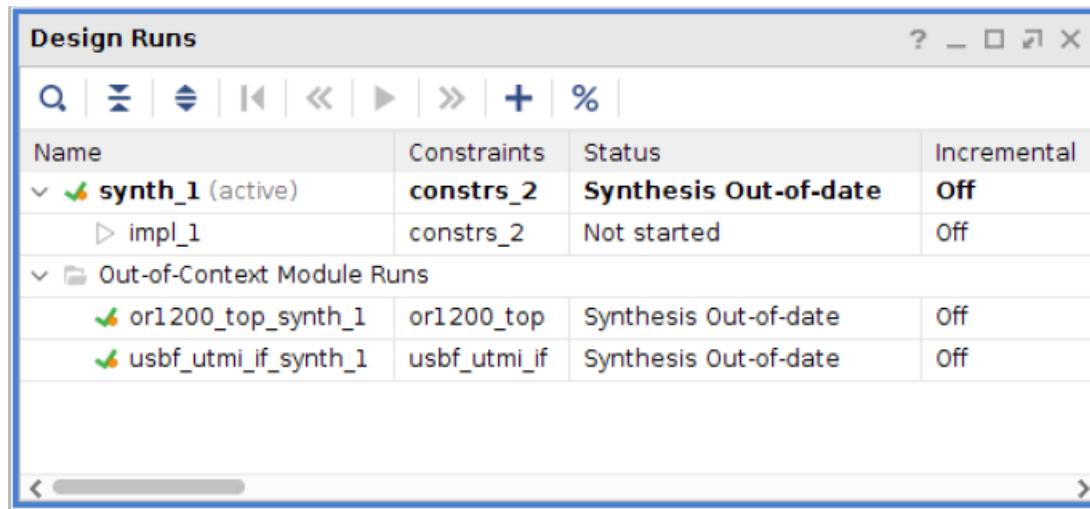
在非关联 (OOC) 综合流程中，某些层级与顶层分离，单独进行综合。非关联层级首先进行综合。然后，运行顶层综合，并且每次非关联运行都作为黑盒来处理。AMD IP 通常采用非关联综合模式来运行。所有非关联综合都运行完成并且顶层综合运行完成后，Vivado 工具会在您打开已综合的顶层设计时从所有综合运行执行设计汇编。此流程具有以下优点：

- 缩短后续综合运行的编译时间。仅对您指定的运行进行重新综合，其他运行保持不变。
- 确保进行设计更改时的稳定性。仅对包含更改的运行进行重新综合。

此流程的缺点在于需要额外进行设置。您必须谨慎选择将哪些模块设置为非关联综合模块。任何额外 XDC 约束都必须单独定义，并且仅限用于非关联综合运行。

下图所示设计包含 1 个顶层综合运行 (synth_1) 和 2 个低层非关联综合运行。

图 108：含 2 个非关联综合运行的顶层综合



如需了解有关设置非关联综合运行的更多信息，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。

增量综合

您可使用增量综合来复用现有综合结果。此方法能够将典型的综合编译时间缩短 50%。此方法配合增量实现流程一起使用时，可缩短总体编译时间并提升时序收敛一致性。

对设计进行综合时，会将其细分为多个 RTL 分区。增量综合会复用来自前一轮综合运行的 RTL 分区。RTL 分区通常随逻辑层级一起创建的。仅当设计足够大、综合创建至少 4 个 RTL 分区并且每个分区包含至少 50000 个实例时，才会运行增量综合。这些实例包含逻辑层级和 RTL 原语。

以下是使用 `synth_design -incremental_mode <value>` 命令时可用的不同模式：

- `off`: 不运行增量综合。
- `quick`: 最快获得结果，但不执行跨边界最优化。此模式会限制逻辑的工作频率。
- `default`: 启用大部分逻辑最优化，包括跨边界最优化。非增量综合，显著缩短编译时间。
- `aggressive`: 启用所有最优化。非增量综合，显著缩短编译时间。

通常建议仅对低性能设计使用 `quick` 模式。由于无跨边界最优化，因此典型设计的时钟频率会降低。但如果精心构造设计且设计包含已寄存的分层边界，那么可能不会影响可实现的时钟频率。由于跨边界最优化受限，因此在给定区域内进行 RTL 更改才会导致在该区域内发生重新综合。在其他综合分区内执行的更改不会触发超出该分区范围的更改。由此导致增加复用，并能更快获得综合结果。对于其他模式，则情况并非如此，RTL 更改可能触发超出单元所在分区范围的其他分区发生再综合。当超出 50% 的分区发生修改时，将触发完全重新综合。

对于高性能设计，建议使用 `default`、`aggressive` 和 `off` 模式。在 `aggressive` 和 `off` 模式下会启用更多最优化，这可能导致进行更多再综合，但 QoR 会更高。

为了更加积极地应对编译时间问题，可将增量综合与 OOC 综合进行比对。OOC 综合通常供 IP 使用，并自动执行设置。含增量综合的全局综合所提供的优势不仅体现在允许跨边界最优化上，还体现在编译时间上。值得考量的领域包括：

- 编译时间

OOC 综合更快，因为它减少了每次细化的代码量。仅当在 OOC 模块内修改 RTL 时，才会细化 RTL。

- Performance

增量综合的最高时钟频率优先于 OOC 综合，因为此模式允许跨 OOC 边界执行最优化。

- 设置

对于非 IP 流程，创建 OOC 综合运行时，必须在从更高的模块传递泛型/参数的同时创建封装文件。此外，还必须创建独立时序约束文件，并以 OOC 级别的端口作为目标。增量综合则不存在这些要求。

如需了解更多信息，请参阅《Vivado Design Suite 用户指南：综合》([UG901](#))。

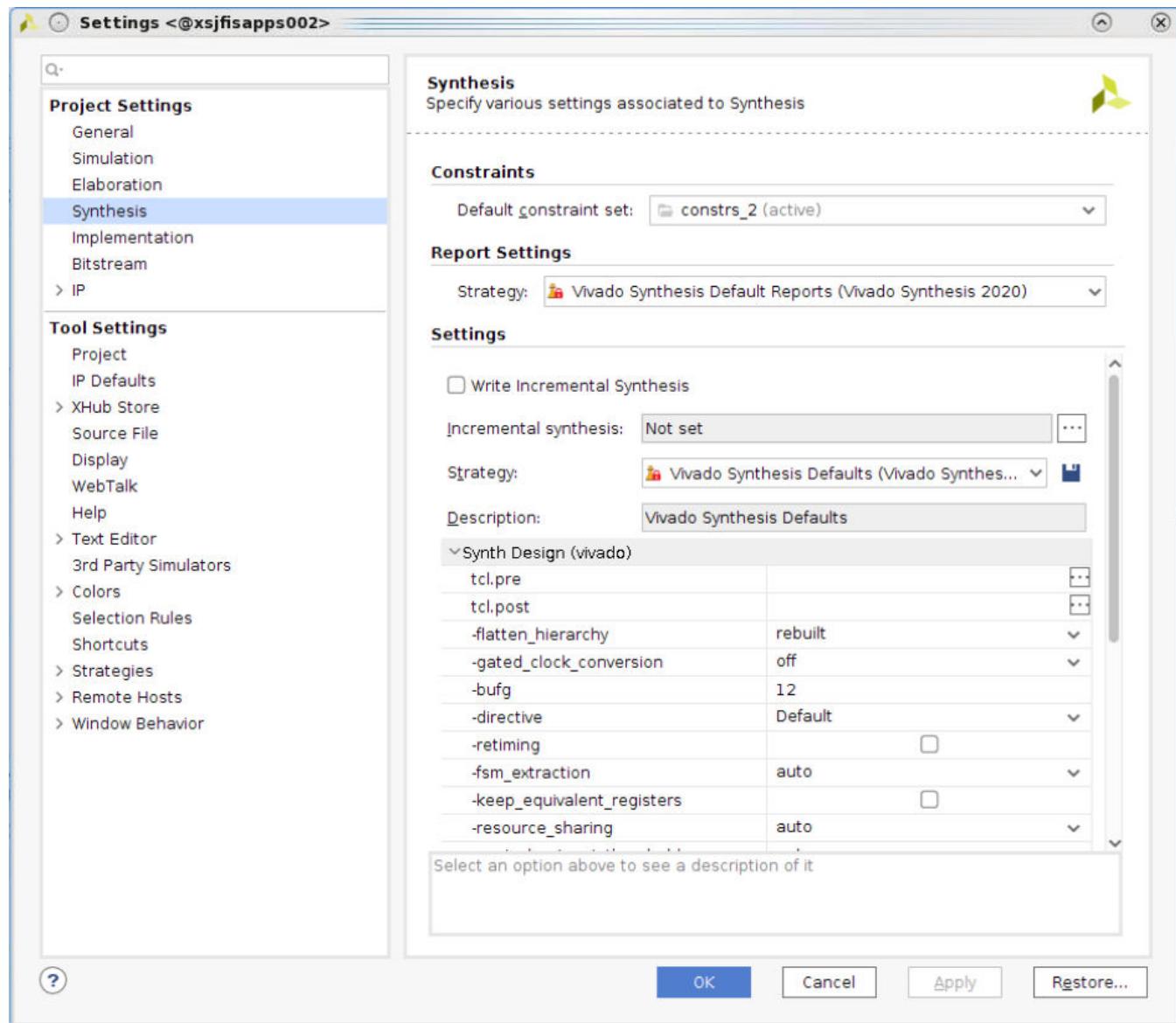
综合最优化

默认情况下，Vivado 综合所应用的最优化措施旨在为尽可能更多的设计产生最佳的结果。但您可以按下列章节中所述方式来调整默认综合最优化操作。

综合设置

您可使用 Vivado Design Suite 综合设置来选择多项影响整个设计的全局设置。这些设置将影响逻辑推断方式以及增量综合的运行方式。AMD 建议开始设计时使用默认选项，随后根据设计的具体需求来更改相应选项。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。

图 109：Vivado 综合设置



综合属性

综合属性支持您以特定方式控制逻辑推断。虽然综合算法致力于为最大数量的设计提供最佳结果，但设计要求常常不尽相同。在此情况下，您可使用属性来更改设计以改进 QoR。如需了解有关综合所支持的属性的信息，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。



电源/功耗提示：请仔细评估综合设置，因为这些设置对于设计功耗影响巨大。例如，设置较低的控制集阈值可以提高寄存器时钟使能的使用率，但会导致封装密度下降。完成综合后，运行 `report_power` 命令以评估综合设置对于功耗的影响。

注释：在为新器件调整设计之前，AMD 建议复查先前针对旧器件运行的设计中的所有综合属性。

使用 KEEP、DONT_TOUCH 和 MAX_FANOUT 属性时，请注意以下章节中所述的特殊注意事项。

KEEP 和 DONT_TOUCH

KEEP 和 DONT_TOUCH 均为非常有价值的设计调试属性。当对象被赋予这两种属性时，工具就不会最优化该对象。

- KEEP 供综合工具使用，并且不作为网表中的属性来进行传递。KEEP 可用于保留特定信号，例如，用于在综合期间关闭信号上的特定最优化。
- DONT_TOUCH 供综合工具使用，并可供传递到布局布线工具以确保对象永不进行最优化。

请谨慎使用这些属性：

- 接收 RAM 输出的寄存器上如果存在 KEEP 属性，就会导致该寄存器无法并入 RAM，从而阻止对块 RAM 进行推断。
- 在驱动更高层级上的三态输出或双向信号的层级上，请勿使用这些属性。如果驱动信号和三态条件都位于当前层级，那么将不会推断 IOBUF，因为工具必须更改层级才能创建 IOBUF。
- 禁用最优化的属性通常会导致电路增大以及功耗增大。AMD 建议有节制地使用这些控制，如不再需要，可将其移除。

此外，请注意将 DONT_TOUCH 放置在信号上与将其放置在层级上的差异：

- 如果将该属性放置在信号上，将保留该信号。
- 如果将该属性放置在层级上，该工具不会触碰该层级的边界，并且该层级中不会发生常数传输。但该层级内部的最优化将予以保留。

MAX_FANOUT

MAX_FANOUT 强制综合通过复制逻辑来满足扇出限制。该工具能够复制逻辑，但不能复制输入或黑盒。因此，如果在设计的直接输入所驱动的信号上放置 MAX_FANOUT 属性，那么该工具将无法处理约束。

请谨慎分析 MAX_FANOUT 所在的信号。如果将 MAX_FANOUT 布局在由含 DONT_TOUCH 的寄存器所驱动的信号上，或者驱动的信号所在层级与 DONT_TOUCH 属性所在层级不同，那么将无法遵循 MAX_FANOUT 属性进行操作。

综合在执行首次复制时会为复制的单元追加 _rep，为后续复制追加 _rep_0、_rep_1，以此类推。通过在单元上选择“Edit”→“Find”（编辑>查找），可以在综合后网表中看到这些单元。



重要提示！ 在综合期间，请谨慎使用 MAX_FANOUT。AMD Vivado™ 工具中的 place_design 和 phys_opt_design 命令可执行基于布局的复制，此操作比在综合内执行逻辑复制更有效。如果需要使用特定扇出，通常有必要花时间和精力来手动编码以生成额外寄存器。

块级综合策略

借助 Vivado 综合，您可使用各种策略和全局设置来自定义设计的综合方式。大多数情况下，这些选项均为全局选项并且影响整个设计。您可使用块级综合策略，通过自上而下的流程采用不同全局选项来对不同层级进行综合。相比于自下而上编译，此流程更快且更容易执行。您可以为整个设计设置约束，而不必先为较低层级设置约束然后再针对顶层进行重新设置。

在 XDC 文件中使用以下语法设置块级综合策略：

```
set_property BLOCK_SYNTH.<option_name> <value> [get_cells <instance_name>]
```

其中：

- <option_name> 为要设置的选项。
- <value> 为要分配给该选项的值。

- <instance_name> 为要在其中设置该选项的层级实例。

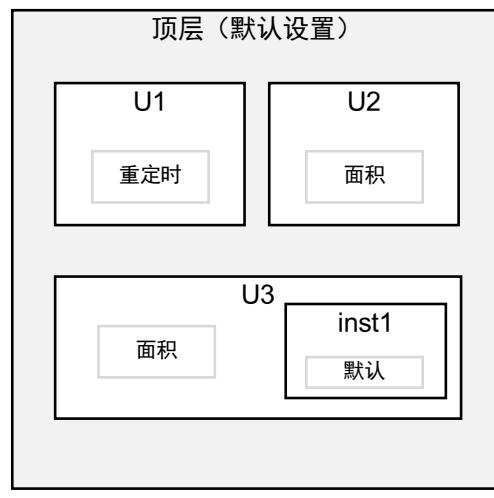
注释：这些属性始终在层级实例上进行设置。这样即可通过不同选项对已多次例化的模块或实体进行综合。

例如，您可以在 XDC 文件中设置以下策略：

```
set_property BLOCK_SYNTH.RETIMING 1 [get_cells U1]
set_property BLOCK_SYNTH.STRATEGY {AREA_OPTIMIZED} [get_cells U2]
set_property BLOCK_SYNTH.STRATEGY {AREA_OPTIMIZED} [get_cells U3]
set_property BLOCK_SYNTH.STRATEGY {DEFAULT} [get_cells U3/inst1]
```

Vivado 综合的执行方式如下图所示。

图 110：块级综合策略示例



X19285-052822

为了尝试不同的选项，您可以在同一实例上设置多种 BLOCK_SYNTH 属性。例如：

```
set_property BLOCK_SYNTH.STRATEGY {ALTERNATE_ROUTABILITY} [get_cells inst]
set_property BLOCK_SYNTH.FSM_EXTRACTION {OFF} [get_cells inst]
```

当使用 IP 时，您可以使用下列块级综合策略：

- 如果 IP 采用全局编译，那么您可在 IP 顶层使用该策略。
- 对于非关联 IP，则不能使用该策略，因为此 IP 是 1 个黑盒。应改为在编译 IP 时使用全局设置。

注释：如需了解有关该功能以及受支持的策略和选项的更多信息，请参阅《Vivado Design Suite 用户指南：综合》(UG901)。

综合后的步骤

确保综合过程中您已获得的网表质量优良，这样它就不会在下游造成问题。在继续执行实现流程的剩余步骤之前，应检查如下重要内容。

检查和清理 DRC

`report_drc` 命令可运行设计规则检查 (DRC) 以寻找常见设计问题和错误。需要进行多重规则检查。默认规则检查如下：

- 综合后网表
- I/O、BUFG 和其他特定的布局需求
- 属性和 MGT、IODELAY、MMCM、PLL 的连线以及其他原语



建议：设计进程中应尽早检查和纠正 DRC 违例，以避免在实现流程的后期出现时序或逻辑相关的问题。



提示：对于可安全忽略的 DRC 违例，您可以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。

运行 Report Methodology

Vivado 工具中提供了“Report Methodology”（方法论报告），专用于检查是否符合方法论指南要求。这些工具根据所处的设计进程阶段运行不同的检查：

- RTL 设计：RTL lint 样式检查
- 综合设计和实现设计：网表、约束和时序检查

在“Project Mode”（工程模式）下，这些工具默认会在实现 (`opt_design` 或 `route_design`) 期间自动运行“Report Methodology”。要手动运行这些检查，请使用以下任一方法：

- 在 Tcl 提示符处，打开要确认的设计，并输入以下 Tcl 命令：`report_methodology`
- 要从 Vivado IDE 运行这些检查，请打开要确认的设计，然后选择“Reports” → “Report Methodology”（报告 > 方法论报告）。



建议：要识别常见的设计问题，请在首次对设计进行综合时运行此报告。添加重要模块、发生重大约束变更或者重大时钟电路变更后，再次运行此报告。

注释：对于 AMD 提供的 IP 核，已经对违例进行过评估与核查。

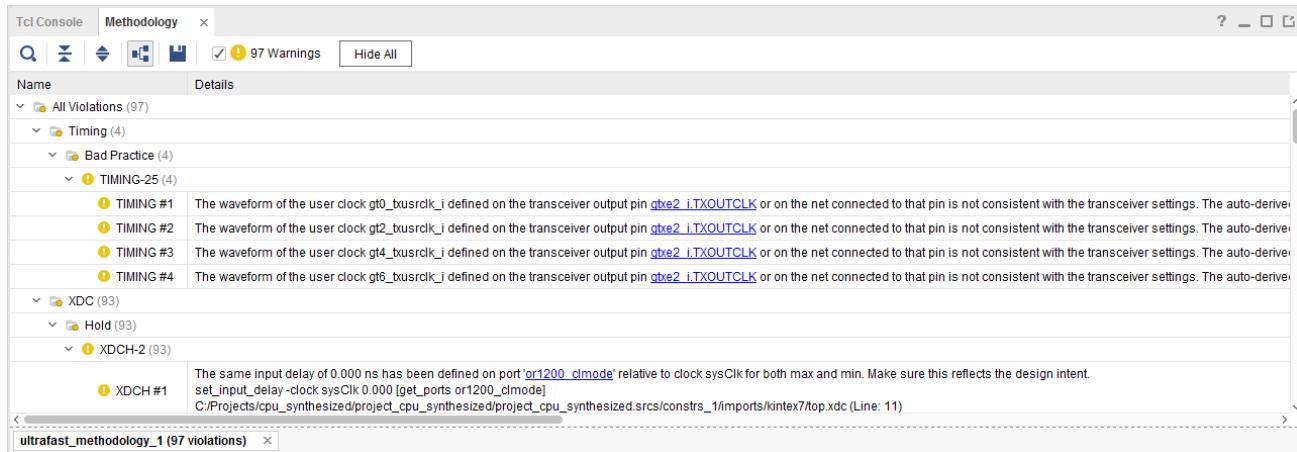
在“Methodology”（方法论）窗口中已列出所有违例情况，如下图所示。如果不需要修复特定方法论违例，请确保您明确了解此违例及其影响，以及此违例不会对您的设计产生负面影响的具体原因。



重要提示！ 您必须解决所有“Critical Warnings”（严重警告）和大部分“Warnings”（警告），以确保 QoR 结果良好、时序分析准确性高，并且满足硬件可靠性和稳定性要求。对于可安全忽略的方法论检查违例，您可以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。

注释：当原语的所有输入或输出路径的建立时序都大于 1 ns 时，将不会报告与 RAMB 和 DSP 原语可选流水打拍 (SYNTH-6、SYNTH-11、SYNTH-12 和 SYNTH-13) 相关的方法论检查。

图 111：“Methodology”窗口



如需了解有关运行 Report Methodology 的更多信息，请参阅《Vivado Design Suite 用户指南：系统级设计输入》([UG895](#))。另请访问此[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。

相关信息

[修复 report_methodology 标记的问题](#)

复查综合 log 日志

您必须复查综合 log 日志文件并确认该工具提供的所有消息与您期望的设计用途相匹配。请特别关注“Critical Warnings”（严重警告）和“Warnings”（警告）。大多数情况下，需修复“Critical Warnings”（严重警告）才能实现可靠的综合结果。



注意！ 如果某条消息出现超过 100 次，那么该工具仅将前 100 次写入综合 log 日志文件。您可通过 Tcl 命令 `set_param messaging.defaultLimit` 来更改 100 次的限值。

检查时序约束

您必须提供简单标准的时序约束以及时序例外（如果适用）。错误的约束将导致编译时间过长、出现最高时钟频率问题以及硬件故障。



建议：复查所有时序约束相关的“Critical Warnings”（严重警告）和“Warnings”（警告），这些警告表示尚未加载或者未能正确应用约束。

相关信息

[对设计约束进行组织以便执行编译](#)

评估综合后的结果质量

“Report QoR Assessment”（QoR 评估报告）可将逻辑层次检查、使用率检查和最常用的时钟拓扑结构检查组合到单一汇报报告中，以便为您提供总体设计评估。此报告可帮助您了解时序收敛问题的严重性。AMD 建议您在含正确时序约束的设计上完成任何重大网表更新后，在综合后的设计上运行此报告。

“Report QoR Assessment” 可提供 1 到 5 之间的评分，用于表示设计实现时序收敛的可能性。下表显示了每个评分的定义。得分为 1 和 2 表示不可能满足时序收敛，得分为 3 表示满足时序收敛可能性较低。因此，低分意味着需要继续努力实现时序收敛。

表 4：QoR 评估报告评分

得分	含义
1	设计将可能无法完成实现。
2	设计将能够完成实现，但无法满足时序。
3	设计将可能无法满足时序。
4	设计将可能满足时序。
5	设计将能够满足时序。

在此报告中，详情表提供了有关评分的基本信息。详情表中的阈值对于器件而言并非绝对限制。这些阈值仅表示何时达成时序收敛的难度可能增加。超出其中任一项的阈值后，达成时序收敛的难度就会显著增大。

请制定相应计划以便纠正 “Report QoR Assessment” 中标记待查的任意项。其中许多项都可使用 “Report QoR Suggestions”（QoR 建议报告）自动解决。

遵循准则解决剩余违例问题



重要提示！ 综合后，分析时序以识别重大设计问题，必须先解决这些问题，才能继续后续流程。

HDL 更改对 QoR 影响最大。因此，最好在实现前先解决问题，以便实现更快速的时序收敛。分析时序路径时，请特别关注以下问题：

- 最常见的错误（即错误最多的时序路径中出现的单元或信号线）
- 源自于未寄存的块 RAM 的路径
- 源自于 SRL 的路径
- 包含未寄存的级联 DSP 块的路径
- 含大量逻辑层次的路径
- 含大扇出的路径

相关信息

[时序收敛](#)

处理高层次逻辑

识别长逻辑路径有助于诊断较难解决的 QoR 挑战。经过评估的综合后信号线延迟接近于最佳布局。要评估存在高层次逻辑延迟的路径是否满足时序要求，可生成无信号线延迟的时序报告。如果路径仍无法满足无信号线延迟的时序要求，那么在这些路径上就无法实现时序收敛。

相关信息

[时序收敛](#)

检查使用率

分别检查 LUT、FF、块 RAM 和 DSP 组件的使用率至关重要。如果块 RAM 使用率较高，而 LUT/FF 使用率较低，那么设计仍可能出现布局问题。`report_utilization` 命令会生成全面的使用率报告，并分章节逐一罗列所有设计对象。

注释：在综合之后，由于设计流程后续执行的优化，使用率可能发生改变。

复查时钟树

本节讨论了如何复查时钟树，包括时钟缓冲器使用率和时钟树拓扑结构。

时钟缓冲器的使用

`report_clock_utilization` 命令用于提供有关时钟原语使用率的详细信息。请观察架构时钟设置规则以避免出现下游布局问题。布局约束无效或者区域时钟缓冲器扇出过高可能导致布局器中出现问题。对于时钟缓冲器使用率过高的设计，可能需要锁定时钟生成器和部分区域时钟缓冲器以帮助完成布局。

对于需要极为严格的时序关系的部分接口，有时最好为需要极为严格的时序关系的信号（例如，源同步接口）锁定特定资源。总而言之，除非存在如上所述的特殊原因，否则作为设计的出发点，只需锁定 I/O 即可。

相关信息

[时序收敛](#)

时钟树拓扑结构

请遵循以下建议来处理时钟树：

- 运行 `report_clock_networks` 命令以在详细的树视图中显示时钟网络。
- 按可最大限度降低偏差的方式来使用时钟树。
- 对于 PLL 和 MMCM 的输出，请使用相同类型的时钟缓冲器来最大限度降低偏差。
- 寻找可能意外引发额外延迟和/或偏差的级联 BUFG 元件。

实现设计

Vivado Design Suite 实现包括在器件资源上对网表进行布局布线，同时满足设计的逻辑、物理和时序约束所需的所有步骤。如需了解有关实现的更多信息，请参阅下列资源：

- 《Vivado Design Suite 用户指南：实现》([UG904](#))
- [Vivado Design Suite QuickTake 视频：设计流程概述](#)

使用工程模式对比使用非工程模式

您可使用工程模式或非工程模式来运行设计实现。工程模式可提供工程基础架构，如运行管理、文件集管理、报告生成和交叉探测。非工程模式可提供简单集成，并且由 Tcl 脚本驱动，此类脚本必须在整个流程中显式调用所需的全部报告。如需了解有关这些模式的更多信息，请访问此[链接](#)以参阅《Vivado Design Suite 用户指南：设计流程概述》([UG892](#)) 中的相应内容。

策略

策略供 Vivado Design Suite 用于控制由“工程模式”下的综合和实现运行所生成的工具选项和报告。您可以使用相关策略对实现目标进行调整并控制所生成的报告。如需了解有关策略的更多信息，请参阅《Vivado Design Suite 用户指南：实现》([UG904](#))。



建议：请首先尝试默认策略 - Vivado 实现默认设置。该策略可在编译时间与设计的最高工作时钟频率之间实现良好的平衡。

注释：策略因工具和版本而异。在某些情况下，策略所需编译时间可能较长。

指令

指令为下列实现命令提供了不同行为模式：

- opt_design
- place_design
- phys_opt_design
- route_design

首先使用默认指令。当设计接近完成时，使用其他指令来浏览设计的解空间。每次只能指定一项指令。如需了解有关指令的更多信息，请参阅《Vivado Design Suite 用户指南：实现》([UG904](#))。

迭代流程

在非工程模式下，您可以使用不同选项通过各种最优化命令进行迭代。例如，可先运行 phys_opt_design -directive AggressiveFanoutOpt，再运行 phys_opt_design -directive AlternateFlowWithRetiming，以便对不满足时序要求的已布局的设计运行不同的物理综合最优化。

以迭代方式运行 phys_opt_design 可以改进时序约束。phys_opt_design 命令会尝试对顶层时序问题路径进行最优化。通过以迭代方式运行 phys_opt_design，可借助最优化来令更多关键路径获益。在布线后阶段中运行 phys_opt_design 会对可能未布线的任何信号线进行重新布线。因此，在布线后运行 phys_opt_design 后，无需再显式运行 route_design。

使用检查点分析不同阶段的设计

Vivado Design Suite 使用物理设计数据库来存储布局布线信息。设计检查点文件 (.dcp) 支持您在设计流程中的关键节点保存 (write_checkpoint 命令) 和复原 (read_checkpoint 命令) 此物理数据库。检查点是处于流程中特定节点的设计快照。在“工程模式”下，Vivado 工具会自动生成设计检查点文件，并将其存储在实现运行目录中。这些文件可在 Vivado 工具的单独实例中打开。

此设计检查点文件包含以下内容：

- 当前网表，包括实现期间执行的任何最优化
- 设计约束
- 实现结果

在设计流程的其余部分中，可使用 Tcl 命令来运行检查点设计。无法通过新设计源来对其进行修改。

以下是常见的检查点使用示例：

- 保存结果，以便您稍后返回并对这部分流程进行进一步分析。
- 尝试运行 `place_design`，使用多项指令并为每项指令保存检查点。这样即可支持您选择具有最佳时序结果的 `place_design` 检查点，以便用于执行后续实现步骤。

如需了解有关检查点的更多信息，请参阅《Vivado Design Suite 用户指南：实现》(UG904)。

使用交互报告文件

打开检查点后，可将生成的报告读取到 Vivado IDE 中，并在其中立即对报告进行分析。要生成报告，请使用以下报告命令并附加 `-rpx <filename.rpx>` 选项：

```
report_timing_summary  
report_timing  
report_power  
report_methodology  
report_drc
```

打开检查点之后，您可以使用“Reports”→“Open Interactive Report”（报告 > 打开交互报告）来打开交互式报告文件。

注释：在“工程模式”下，可以自动生成并打开交互报告。



建议：生成报告时，对 RPX 文件有大小限制。因此，AMD 建议使用 `catch` 命令来防止出现可能导致流程停止的错误。例如：`catch {report_timing_summary -rpx timing_summary.rpx -file timing_summary.rpt}`。

使用增量实现流程

在 Vivado Design Suite 中，您可以使用增量实现来复用现有布局和布线数据，从而缩短实现的编译时间，并提升结果的可预测性。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。



建议：在设计周期的关键阶段中难以对流程脚本执行更改时，增量实现最为有用。确保您的流程脚本在设计周期初即包含增量实现，这样即可在关键时期启用增量实现。

注释：为进一步缩短编译时间和提升 QoR，还可使用增量综合。

相关信息

增量综合

自动增量实现模式

您可使用自动增量实现模式来激活增量实现流程，同时允许 Vivado 工具延迟运行增量实现直至获取有关参考检查点和当前设计的更多信息为止。当发出 `read_checkpoint` 命令时，Vivado 工具会判定使用默认流程算法还是使用增量流程算法来运行实现流程。自动模式提供了易于使用的按钮，因为工具负责管理参考设计数据以供用于增量实现。

注释：自动增量实现模式比运行默认增量实现流程更保守，支持在运行增量实现流程时为 QoR 提供更好的维护。

工程模式

在工程模式下，Vivado 工具可管理检查点的更新以及所使用的算法。要在工程模式下启用自动增量实现模式，请右键单击“Design Runs”（设计运行）窗口，然后选择“Set incremental Compile”→“Automatically use the checkpoint from the previous run”（设置增量编译 > 自动使用上一轮运行的检查点）。

等效的 Tcl 命令为：

```
set_property AUTO_INCREMENTAL_CHECKPOINT 1 [get_runs <runName>]
```

非工程模式

在非工程模式下，Vivado 工具可管理要使用的算法，但您必须判定是否要更新检查点。要在“非工程模式”下启用自动增量实现模式，请使用 `-auto_incremental` 选项。以下是一条示例命令：

```
read_checkpoint -incremental -auto_incremental <reference>.dcp
```

更新检查点时，请在实现流程末尾使用以下命令来避免 WNS 劣化至可接受的下限以下：

```
if {[get_property SLACK [get_timing_path -setup]] > -0.250} {  
    file copy -force <postroute>.dcp <reference>.dcp  
}
```

增量指令和目标 WNS

您可使用增量指令来指定实现流程的目标 WNS。目标 WNS 可判定实现工具是尝试收敛时序还是尝试实现与参考检查点相同等级的时序收敛。当实现流程使用默认算法时，将忽略增量指令并使用 `place_design` 和 `route_design` 指令。

下表显示了每条增量指令以及对应的目标 WNS 行为。

表 5：增量指令与目标 WNS 行为

增量指令	目标 WNS 行为
RuntimeOptimized	与参考检查点相同
TimingClosure	0.000
Quick	流程不受时序驱动，布局受相关逻辑驱动

注释：增量指令将取代先前版本的指令映射。

配置增量流程

您可以使用 `config_implementation` 命令配置增量流程。要查看此命令的默认值和当前值，请使用 `report_config_implementation` 命令。要更新这些值，请使用 `config_implementation` 命令。下面给出 1 个示例：

```
report_config_implementation  
config_implementation { {incr.ignore_user_clock_uncertainty true} }
```

注释：您可采用上述外层括号中所示方法来对键值对进行分组，这样即可同时更新多个元素。

您可配置：

- 最小阈值，用于自动增量流程中的单元匹配、信号线匹配和 WNS。
- 综合与实现的行为，前提是不满足自动增量流程条件。在综合开始运行时以及在为实现执行 `read_checkpoint -incremental` 期间，会执行此项检查。它可设为 `Terminate` 以停止此流程，或者设为 `SwitchToDefaultFlow` 以退出增量流程，而以默认流程设置来继续运行。

- 流程是否忽略用户时钟不确定性约束，此类约束通常用于对布局器进行过约束，并强制执行更紧密的布局。

并行运行

为提升使用默认流程满足时序要求的可能性，常用的方法是实现多轮次的并行运行，每个轮次均采用不同的布局器指令。对于增量流程，该指令会指示需收敛时序还是维持时序。为实现各种不同结果，请为期望的增量指令设置不同的参考点目标。

编译时间注意事项

在自动增量实现模式下或高复用模式下使用 RuntimeOptimized 指令时，如果设计复用率不低于 95%，则编译时间可减半。随复用率下降，编译时间的优势也会一并下降。除非执行影响关键路径的更改，否则通常编译时间可预测。

在自动增量实现模式下或高复用模式下使用 TimingClosure 指令时，需耗费更多时间运行额外算法以达成时序收敛。使用此模式可能导致编译时间增加，当拥塞区域内难以达成时序收敛时或者无法满足时序要求时尤其如此。如果参考检查点可满足时序，那么缩短编译时间的效果与使用先前所述的 RuntimeOptimized 指令的效果类似。

打开综合设计

综合后的第一步是将网表从综合设计读取到存储器中并应用设计约束。您可以根据使用的流程通过各种方式打开综合设计。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。

逻辑最优化 (opt_design)

Vivado Design Suite 逻辑最优化可以对当前存储器中的网表进行最优化。由于这是设计汇编的第 1 个视图 (RTL 和 IP 块)，通常可以进一步对设计进行最优化。默认情况下，`opt_design` 命令会执行逻辑裁剪 (logic trimming)、移除不含负载的单元、传输常量输入和执行块 RAM 功耗最优化。此外，还可选择执行其他最优化操作，例如，重新映射，即：将 LUT 串联组合，减少 LUT 从而降低路径深度。

优化分析

`opt_design` 命令用于生成消息以详述每个优化阶段的结果。完成优化后，可运行 `report_utilization` 来分析使用率的提升情况。为了更好地分析优化结果，请选中 `-verbose` 和 `-debug_log` 选项并重新运行 `opt_design`，以获取有关每次优化对逻辑产生的影响以及用户约束阻止部分优化操作的方式的完整详细信息。欲知详情，请访问此[链接](#)和此[链接](#)以参阅《Vivado Design Suite 用户指南：实现》(UG904) 中的相应内容。

布局 (place_design)

Vivado Design Suite 布局器引擎将来自网表的单元置于目标 AMD 器件中的特定站点 (site) 上。

布局分析

布局后使用“Timing Summary”（时序汇总）报告检查关键路径。

- 含超大建立时间时序负裕量的路径可能需要检查约束以确保完整性和正确性，或者逻辑重组以实现时序收敛。
- 对于含超大保持时间时序负裕量的路径，其主要成因是约束错误或时钟拓扑错误，因此需在进入布线设计之前将其进行修复。
- 具有较小的保持时间时序负裕量的路径有可能通过布线器修复。您还可先运行 `place_design`，然后再运行 `report_clock_utilization` 来查看按时钟区域划分时钟资源和负载计数的报告。

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，并获取有关布局的更多信息。

相关信息

[时序收敛](#)

物理最优化 (phys_opt_design)

物理最优化属于流程中的可选步骤，其用于对设计的负时序裕量路径执行时序驱动的最优化。最优化内容涉及复制、重定时、修复保持时间以及布局提升。由于物理最优化会自动执行所有必要的网表和布局变更，因此在 phys_opt_design 之后不需要 place_design。

判断是否需要物理综合

为了判定物理综合对于设计是否有益，布局后请对时序进行评估。分析不满足要求的路径以确认扇出情况。高扇出的关键路径可受益于扇出优化。此外，执行 route_design 后，如果涉及多个块 RAM 的大型 RAM 块的高扇出数据、地址和控制信号线未能满足时序要求，则同样可能受益于“Forced Net Replication”（强制信号线复制）。如需了解有关物理综合的更多信息，请访问此[链接](#)以参阅《Vivado Design Suite 用户指南：实现》(UG904) 中的相应内容。

布线 (route_design)

Vivado Design Suite 布线器已在布局的设计上执行布线，并在已布线的器件上执行优化，以解决保持时间违例问题。默认情况下，布线器执行优化的方式是在编译时间与设计的工作时钟频率之间实现平衡，同时缓解拥塞。部分布线器指令牺牲编译时间来换取更好的最高设计时钟频率，并积极减少拥塞。如需了解有关布线的更多信息，请访问此[链接](#)以参阅《Vivado Design Suite 用户指南：实现》(UG904) 中的相应内容。

布线分析

时序约束错误通常会导致信号线的布线结果欠佳。在尝试布线器设置前，请确保已确认布线器所看到的约束和时序图。布线前，复查已完成布局的设计的时序报告，以确认时序和约束。

时序约束不良的常见示例包括跨时钟路径和错误的多周期路径，这些错误导致布线延迟插入，并需要修复保持时间。拥塞区域可通过在 RTL 综合中进行针对性的扇出优化或通过物理优化来解决。您可以保留全部或部分设计层级，以防止跨边界优化并降低网表密度。或者，也可以使用布局规划约束来减轻拥塞。

相关信息

[时序收敛](#)

布线编译时间

您可以使用 route_design -ultrathreads 选项，以牺牲可重复性为代价缩短编译时间。该选项给予布线器额外的自由来执行多个线程，从而能够更快地完成布线，但每次的结果又略有不同。相同后续运行之间的裕量相差无几，但可节省大量编译时间。仅当环境不需要严格的可重复结果时，方可考虑采用该选项来缩短布线器编译时间。

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。功耗基线则需要您为 AMD Vivado™ 工具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。由于设计拥塞减少，因而会显著简化时序收敛。
- 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，这样即可留出时间用于探索更多工程设计解决方案，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](#))。

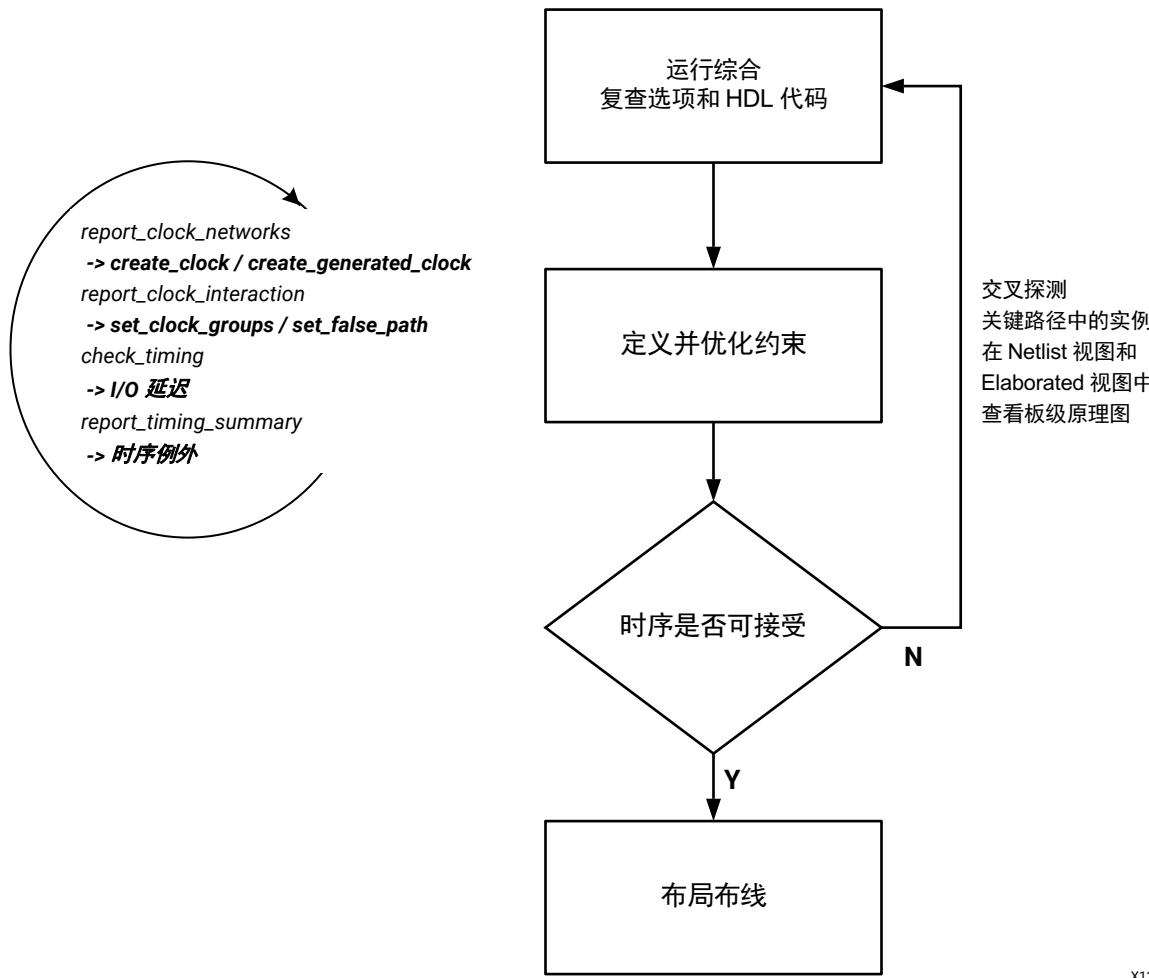


提示：请参阅《UltraFast 设计方法时序收敛快捷参考指南》([UG1292](#))，简略了解本章中所描述技巧，包括运行初始设计检查、设定设计基线以及解决时序违例。

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 112：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



电源/功耗提示：以简单标准的时序来运行设计时，请考虑在 AMD Vivado™ IDE 中使用“Create Runs”（创建运行）命令来运行多项策略。在含准确的开关活动 XDC 约束的每项设计上运行 `report_power` 命令以从时序和功耗角度来查找最佳运行结果。

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看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）： $WNS > 0 \text{ ns}$ 且 $TNS = 0 \text{ ns}$
- 保持/移除（最小延迟分析）： $WHS > 0 \text{ ns}$ 且 $THS = 0 \text{ ns}$
- 脉冲宽度： $WPWS > 0 \text{ ns}$ 且 $TPWS = 0 \text{ ns}$

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

良好的设计不包含任何组合循环。由于时序引擎会导致时序循环中断，因此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

允许不同时钟之间存在时序约束。默认情况下，这些路径均已定时。如果路径有效，请确保时钟真正同步并共享公共资源。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。

如果不同时钟之间的路径并不是有效的时序路径，则必须在这些路径的时钟之间使用 `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。查看时钟交互报告，确保时钟间路径受到适当的约束。

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 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。

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 6：时序收敛方法 DRC

检查	严重性	描述
TIMING-6	严重警告	在相关时钟之间不存在公用时钟
TIMING-7	严重警告	在相关时钟之间不存在公共节点
TIMING-8	严重警告	在相关时钟之间不存在公共周期
TIMING-14	严重警告	时钟树上的 LUT
TIMING-15	警告	时钟间路径上的严重保持时间违例
TIMING-16	警告	严重建立时间违例
TIMING-30	警告	生成时钟所选主源管脚欠佳
TIMING-31	严重警告	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	警告	非建议的总线偏差约束
TIMING-36	严重警告	针对生成时钟缺少主时钟沿传输
TIMING-42	警告	时钟传输受阻于路径分段
TIMING-44 TIMING-45	警告	用户时钟内部和时钟间的不确定性不合理
TIMING-48	咨询	在锁存器输入上存在“仅最大延迟数据路径”约束
TIMING-49	严重警告	来自并行 BUFGCE_DIV 的不安全的使能或复位拓扑结构
TIMING-50	警告	同级锁存器之间的路径要求不现实
TIMING-56	警告	缺少按逻辑或物理方式排除的时钟组约束
XDCB-3	警告	在同一条 set_clock_groups 命令中，多个组中提及同一个时钟
XDCH-1	警告	多周期路径约束中缺少保持时间选项
XDCV-1	警告	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	警告	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 7：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

表 7：验收质量方法 DRC (续)

检查	严重性	描述
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
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-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，其中包括 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) = $\max(1000/(T_i - WNS_i))$

其中：

- T_i 表示在实现运行 “i” 期间所使用的目标时钟周期 (ns)
- WNS_i 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）。

相关信息

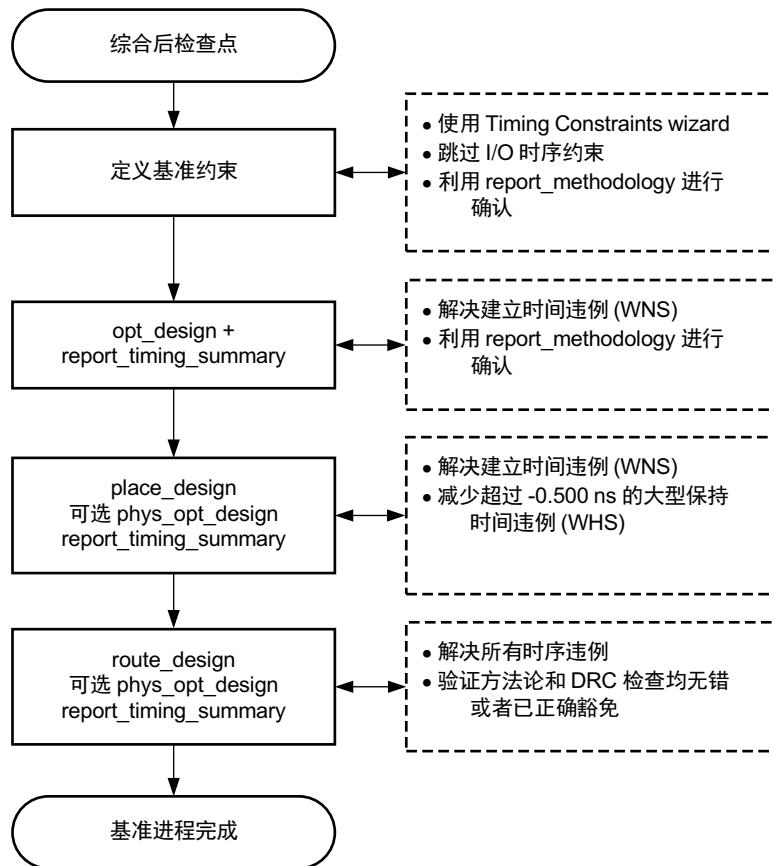
[使用智能设计运行](#)

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 113：设计基线设定



X20037-053022

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 和时钟缓冲器都属于时钟修改块。对于 AMD UltraScale™ 器件，GT 同样属于时钟修改块。

`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 114：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/mmcmb_adv_inst/CLKFBOUT}	
cpuClk	20.000	{0.000 10.000}	P,G,A,R	{clkgen/mmcmb_adv_inst/CLKOUT0}	
wbClk_4	20.000	{0.000 10.000}	P,G,A	{clkgen/mmcmb_adv_inst/CLKOUT1}	
usbClk_3	10.000	{0.000 5.000}	P,G,A	{clkgen/mmcmb_adv_inst/CLKOUT2}	
phyClk0_2	10.000	{0.000 5.000}	P,G,A	{clkgen/mmcmb_adv_inst/CLKOUT3}	
phyClk1_1	10.000	{0.000 5.000}	P,G,A	{clkgen/mmcmb_adv_inst/CLKOUT4}	
fftClk_0	10.000	{0.000 5.000}	P,G,A	{clkgen/mmcmb_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进行约束。

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 115：时钟交互报告样本



下表解释了此报告中每种颜色的含义。

表 8：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`

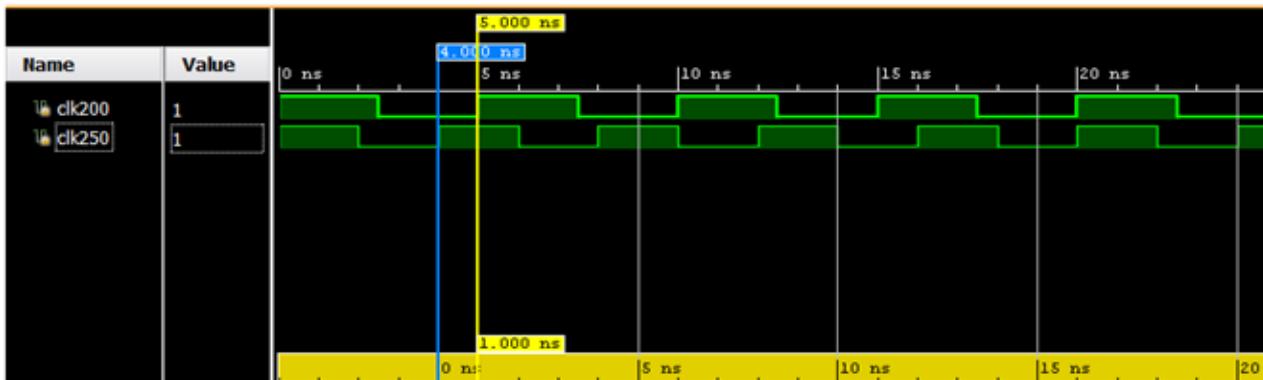
将时钟对作为异步来处理时，请使用上述约束之一。使用 `report_cdc` Tcl 命令来确认时钟域交汇电路是否安全。

- `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。

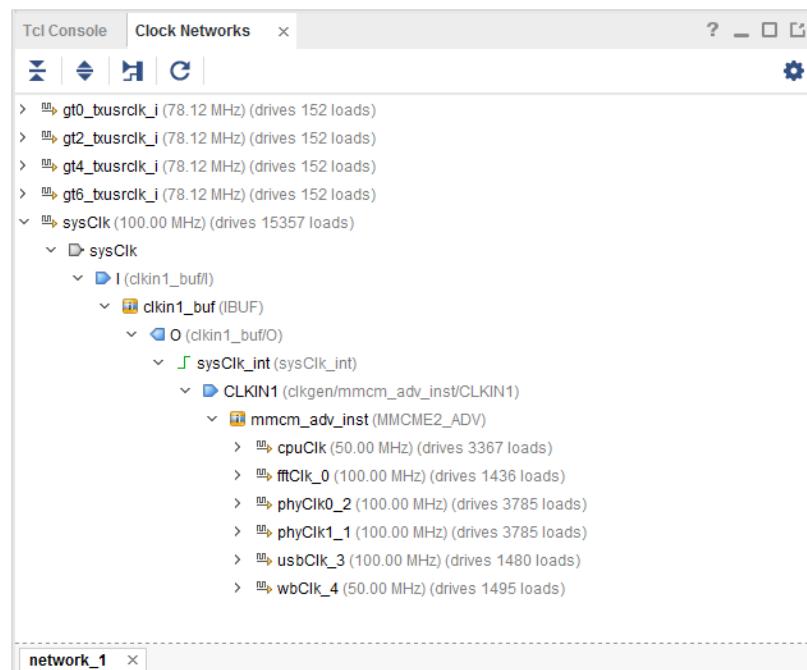
图 116：从 250 MHz 到 200 MHz 的时钟域交汇



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 117：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 9：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。（忽略 QPLLOUTCLK 和 QPLLOUTREFCLK，因为这两项均为仅限脉冲宽度的检查。）
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 10：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。

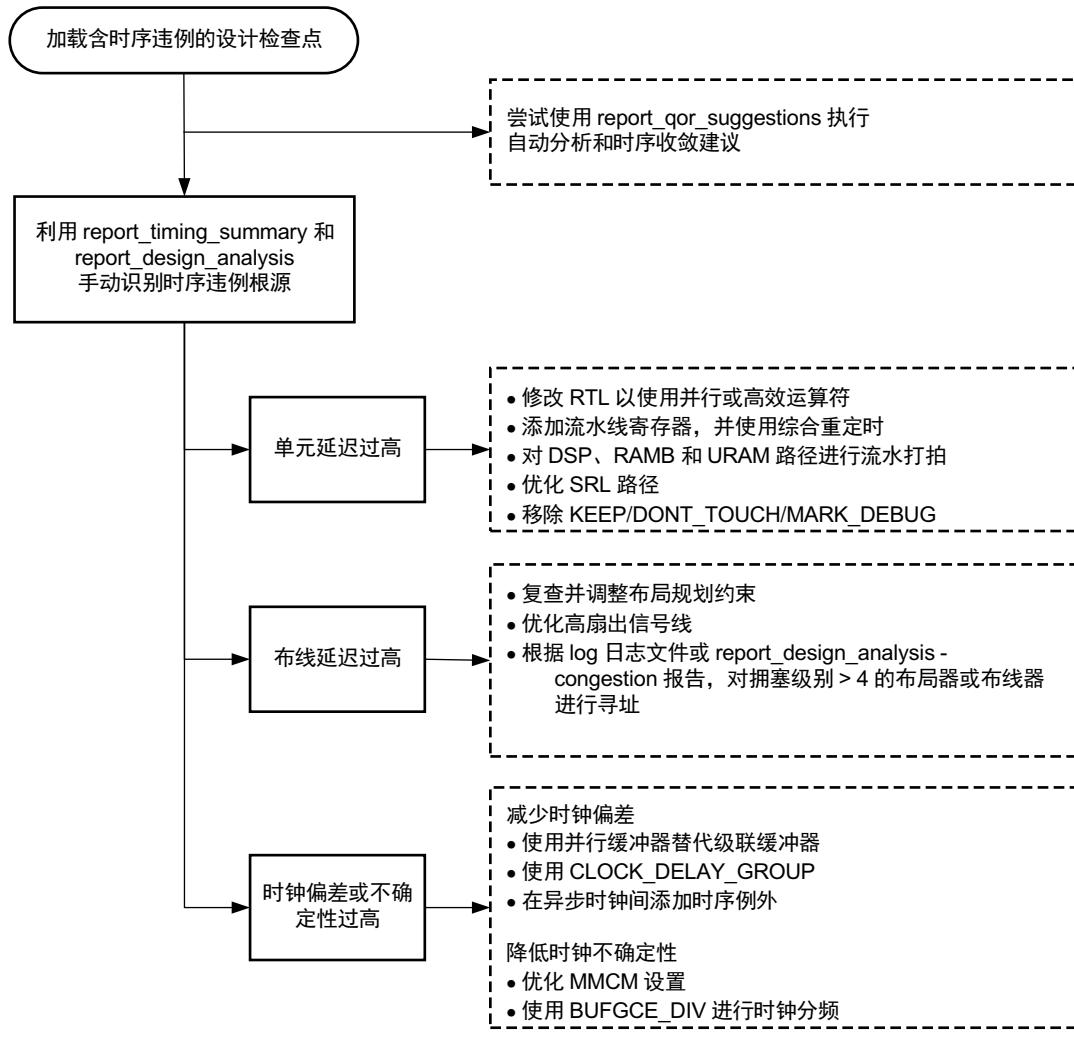
分析并解决时序违例

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 118：分析并解决时序违例



X20036110617

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。

相关信息

检查设计是否正确约束

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 119：设计分析报告之布线后时序路径特性

Setup Path Characteristics													
Name	Requirement	Path Delay	Logic Delay	Net Delay	Clock Skew	Slack	Clock Relationship	Logic Levels	Routes	Logical Path	Start Point (End 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 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 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：

注释：以下括号中显示了表格中相关的列。

- 逻辑延迟百分比过高（逻辑延迟）
 - 逻辑层次是否过多？（逻辑级数）
 - 是否存在阻碍逻辑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）
 - 对于 SSI 技术器件，是否存在跨 SLR 边界的信号线？（SLR 交汇）
 - 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（建立时间 <-0.5 ns，保持时间 > 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。

复查逻辑层次和布线分布表

通常较长的路径首先由布局器加以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。如果在布局之前即已消除较长的路径，则可改善总体时序 QoR。要查看较长的路径，请使用 `report_design_analysis` 命令生成以下表格：

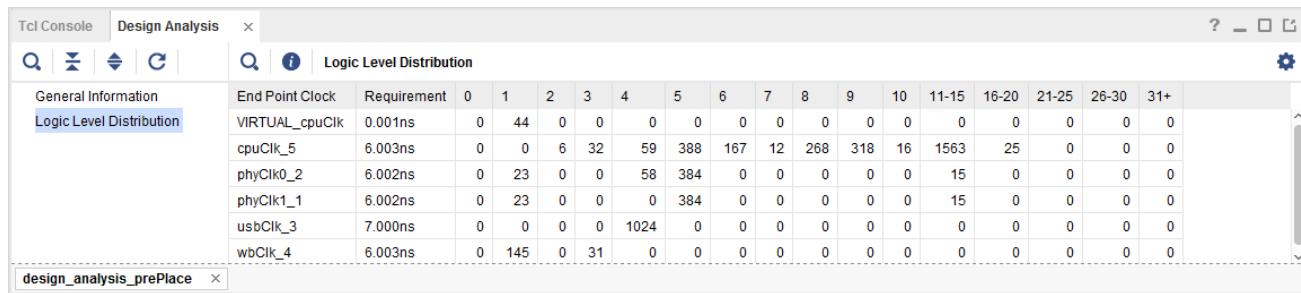
- “Logic Level Distribution” 表：默认显示前 1000 条路径中的逻辑层次分布。
- “Routes Distribution” 表：显示某一时序路径内必须布线穿过正常互连的布线数的分布。例如，该表不包括使用 site 内布线资源和其他专用布线资源的信号线。

注释：当路径显示的逻辑级数与其逻辑延迟不成比例时，这些路径的布线数低于逻辑级数，通常包含 MUXF 或进位链元件。

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 120：设计分析报告之布局前时序路径特性



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 7 系列器件中，单元延迟大于 25%，在 UltraScale 器件中，单元延迟大于 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 7 系列器件中，布线延迟大于 75%，在 UltraScale 器件中，布线延迟大于 50%。
路径是否受到保持时间修复的影响？运行 `report_design_analysis -show_all` 并检查“Hold Detour”（保持时间绕行）列。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。
- o 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`) 可生成包含时序偏差数据的报告。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。在时钟域交汇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 的工具会分析所有时钟组之间的时序，除非时序例外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。

相关信息

[识别时序违例的根源](#)

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 和 LOOKAHEAD）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 属性来强制执行优化。

- 路径中存在单个 CARRY 单元

CARRY 原语级联时对时序约束 QoR 最有利。CARRY 单元比 LUT 更难布局，并且强制综合使用 LUT 而不是单个 CARRY，这在大多数情况下能够实现更好的 LUT 结构和更灵活的布局。尝试 FewerCarryChains 综合指令或 PerfThresholdCarry 策略（仅限“工程模式”），以消除大多数单个 CARRY 单元。或者，使用 CARRY_REMAP 属性指示 `opt_design` 重新将标记的 CARRY 单元映射到 LUT。

注释：要应用此优化技巧，请运行 `report_qor_suggestions` Tcl 命令，然后应用输出文件作为源文件。

- 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：要应用此优化技巧，请运行 `report_qor_suggestions` Tcl 命令，然后应用输出文件作为源文件。

- 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，也可通过在要优化的寄存器上设置 CONTROL_SET_REMAP 属性来指示 `opt_design` 执行同样的优化。

注释：要应用此优化技巧，请运行 `report_qor_suggestions` Tcl 命令，然后应用输出文件作为源文件。



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。

相关信息

[将逻辑从控制管脚推送到数据管脚](#)

使用专用块和宏原语对路径进行优化

专用块与宏原语所涉及的路径（如 DSP、块 RAM 或 UltraRAM）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为约为 1.5 ns（无可选输出寄存器）和 0.4 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。
- 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。

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。

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。

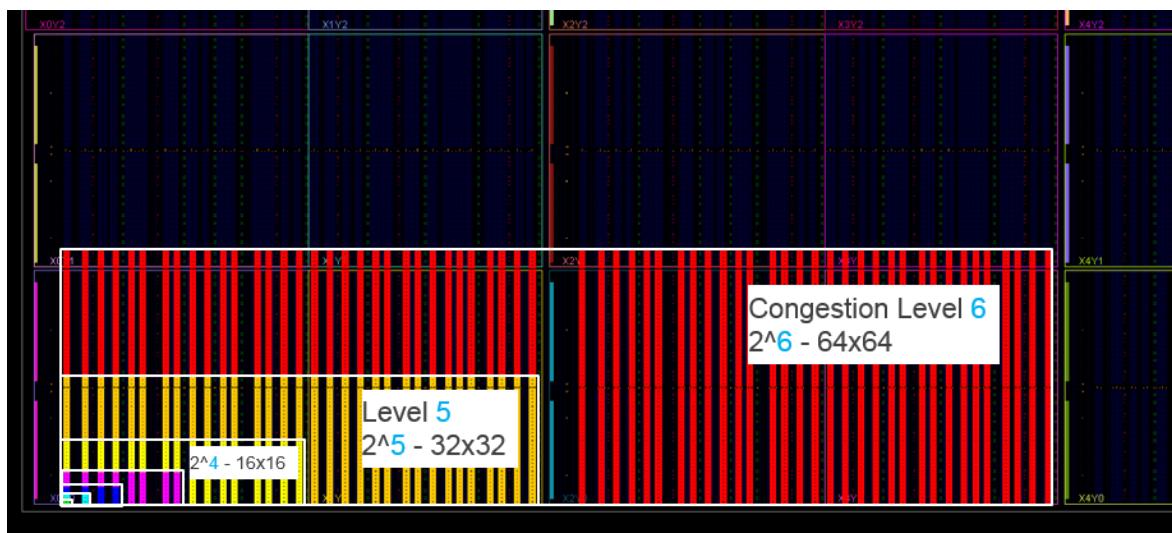
降低因拥塞导致的信号线延迟

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，那么布线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 121：“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 并导致布线器运行时间延长。

表 11：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”

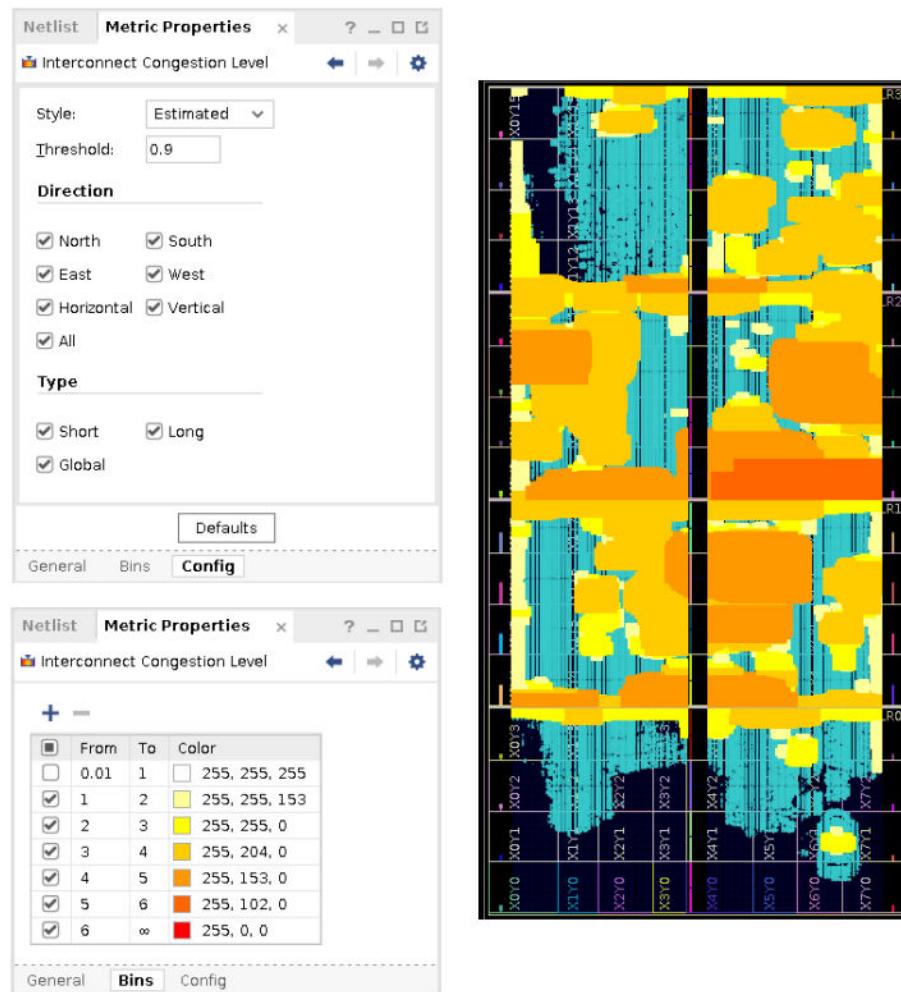
“Interconnect Congestion Level”（互连拥塞等级）指标突出显示了布线资源过度使用的最大连续区域。该指标默认显示估算值，与初始布线后的拥塞等级相似。如果存在布线，也可显示实际布线情况。布局或布线后，可在“Device”（器件）窗口中右键单击并选择“Metric” → “Interconnect Congestion Level”（指标 > 互连拥塞等级）来显示此拥塞指标。

“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)

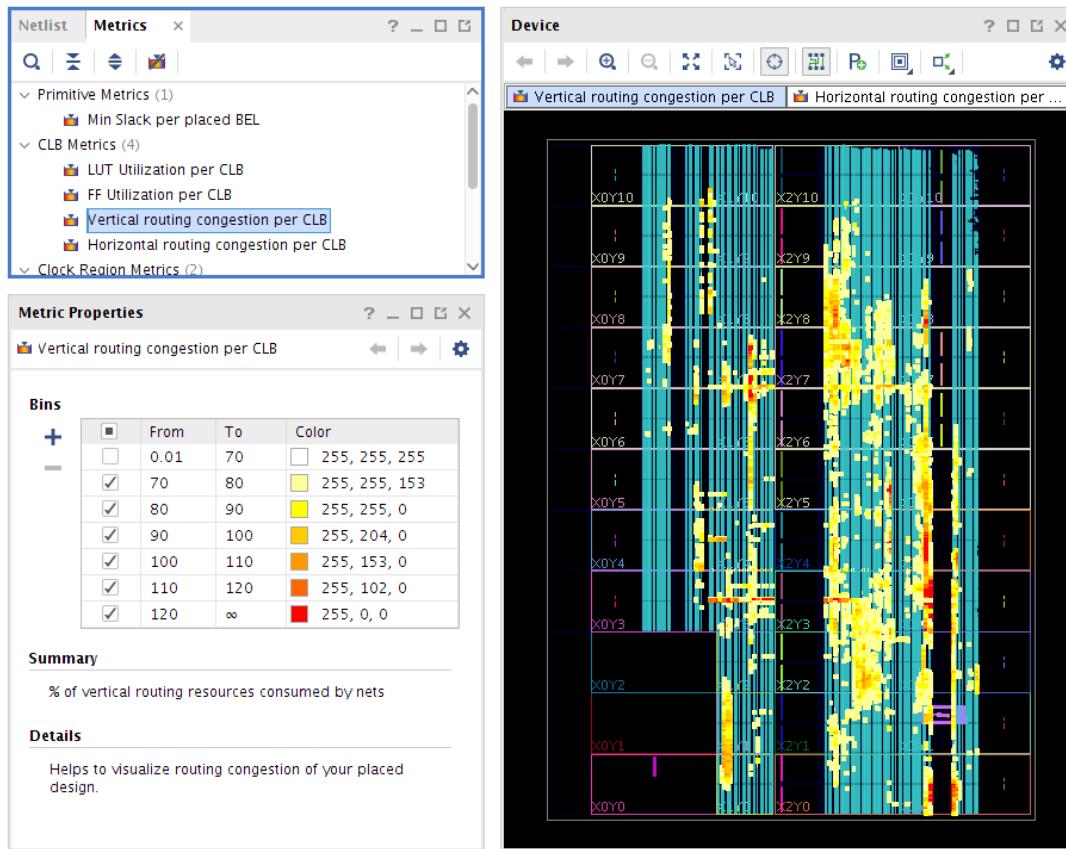
图 122：“Device”窗口中的“Interconnect Congestion Level”示例



根据估算值而非实际布线来使用“Routing Congestion per CLB”（按 CLB 显示布线拥塞）。布局或布线之后，可在“Device”窗口中右键单击并选择“Metric”→“Vertical and Horizontal Routing Congestion per CLB”（指标>按 CLB 显示垂直和水平布线拥塞）。这样即可提供器件中任意拥塞热点的直观概述。下图显示了因高使用率和网表复杂性而导致包含多个拥塞区域的布局设计。

注释：此方法仅适用于 7 系列和 UltraScale 器件。

图 123：“Device”窗口中的“按 CLB 显示拥塞”示例



布局器日志中的拥塞

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。

布线器 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 124：初始估算拥塞表

INFO: [Route 35-449] Initial Estimated Congestion

Direction	Global Congestion		Long Congestion		Short Congestion	
	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。

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 125：拥塞表

General Information	Window	Direction	Congestion Level	Cell Names	Combined LUTs					
					Top Cell 1	Top Cell 2	Top Cell 3	LUT6	LUT5	
Placed Maximum	Window 1	North	4	120 inst_1022144/inst_1022144/inst_102 inst_1022144/inst_1022134/inst_1018559/inst_990436 (10%)	26%	37%	22%	43%	0%	NA
Placed Tile Based (V)	Window 2	East	4	107 inst_1022144/inst_cv_33 (17%)	26%	29%	7%	60%	1%	50%
Placed Tile Based (H)	Window 3	South	4	131 inst_1022144/inst_inst_1022144/inst_102 inst_1022144/inst_1022134/inst_1018559/inst_990436/inst_979691 (5%)	32%	38%	18%	54%	0%	50%
	Window 4	West	2	143 inst_1022144/inst_inst_1022144/inst_102 inst_1022144/inst_1022134/inst_1018559/inst_887992 (9%)	84%	40%	1%	60%	0%	NA

“Placed Maximum”（已布局的最大拥塞）表和“Initial Estimated Router Congestion”（初始估算的布线器拥塞）表提供有关东西南北四个方向上拥塞最严重的区域的信息。分析发生在布线开始时的布局之后，此时可以更好地量化布线要求，不再进行估算。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。

“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”（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 126：复杂性报告

Instance	Module	Rent	Average Fanout	Total Instances	LUT1	LUT2	LUT3	LUT4	LUT5	LUT6	Memory LUT	DSP	RAMB	MUXF	URAM
N 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 12：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 13：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。 当采用 SSI 技术器件时，如果实例总数超过 100,000，则布局器可能难以找到适合 1 个 SLR 或分布在 2 个 SLR 上的解决方案。
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优化。
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。

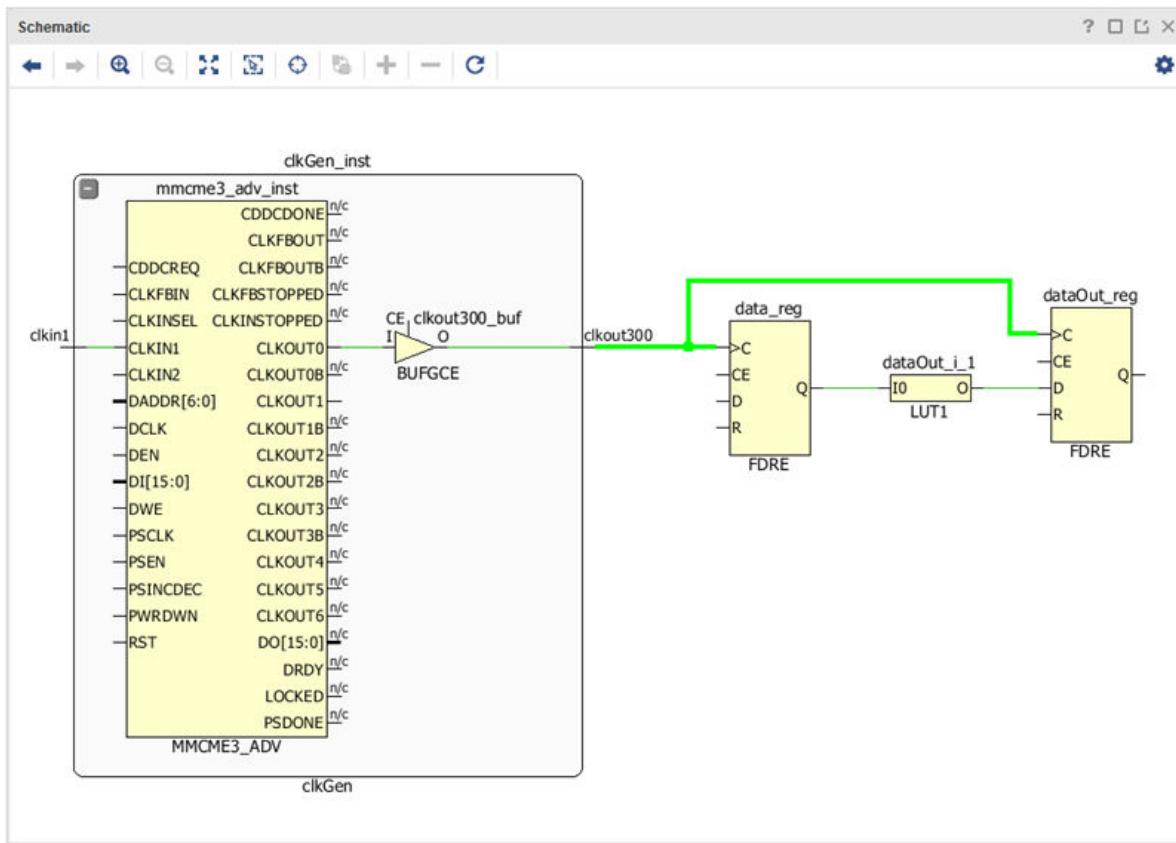
相关信息

[定义时钟组和 CDC 约束](#)

使用同步时钟

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 127：具有公共节点（位于绿色信号线上）的典型同步时钟拓扑



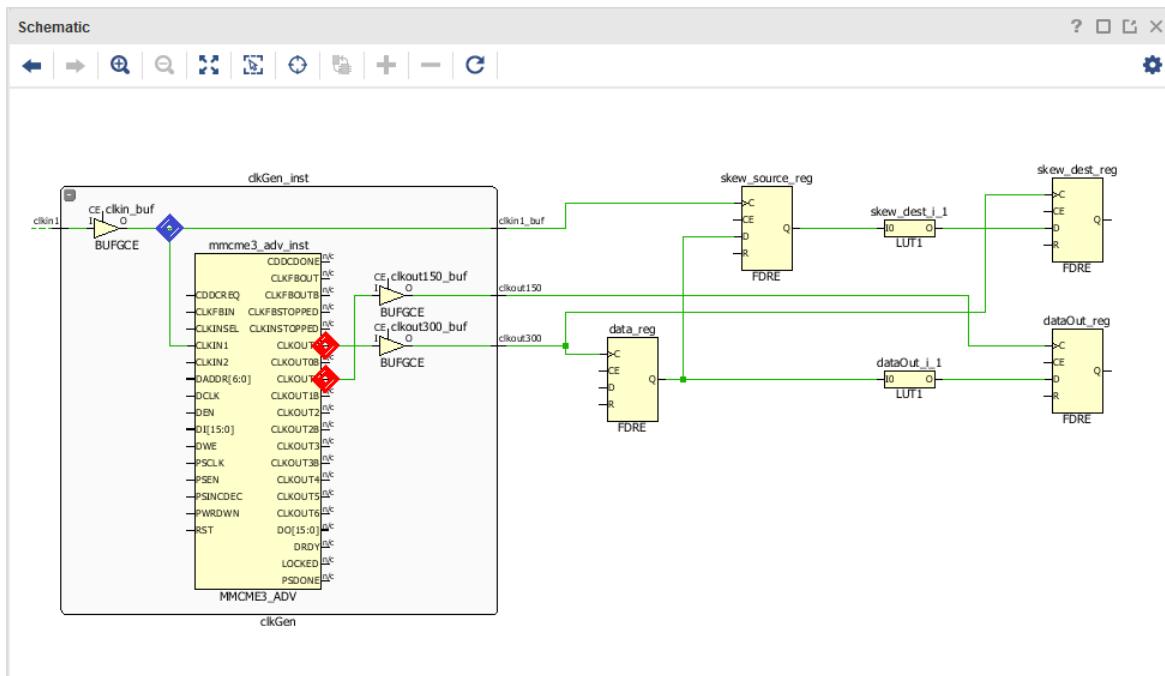
分析时序报告中的时钟路径时，不单独提供公共节点前后的延迟，因为公共节点仅存在于设计的物理设计库中，而不存在于逻辑视图中。因此，开启“Routing Resources”（布线资源）时，您可在 Vivado IDE 的“Device”（器件）窗口中看到公共节点，而无法在“Schematic”（板级原理图）窗口中看到此节点。时序报告仅提供偏差计算汇总，其中包括源时钟延迟、目标时钟延迟以及从时钟消极因素移除 (CPR) 直至公共节点的信用值。

限制同步时钟域交汇路径

由独立时钟缓冲器驱动的同步时钟之间的时序路径会表现出较高的偏差，因为公共节点位于时钟缓冲器之前。即公共节点距离叶时钟管脚更远，导致时序分析的消极因素增加。由于源时钟路径与目标时钟路径之间存在的延迟差异，导致不平衡的时钟树之间的时序路径存在更严重的时钟偏差。虽然正偏差有助于满足建立时间要求，但不利于保持时间收敛，并且反之亦然。

下图中的 3 个时钟具有多条时钟内部和时钟间路径。由 MMCM 驱动的 2 个时钟的公共节点位于 MMCM 的输出处（红色标记）。MMCM 输入时钟与 MMCM 输出时钟之间的路径的公共节点位于 MMCM 之前的信号线上（蓝色标记）。对于 MMCM 输入时钟与 MMCM 输出时钟之间的路径，根据 **clkIn_buf** BUFGCE 位置和 MMCM 补偿模式，时钟偏差值可能极高。

图 128: MMCM 输入和输出上含公共节点的同步 CDC 路径



AMD 建议限制同步时钟域交汇路径的数量，即使时钟偏差处于可接受状态也是如此。并且，如果偏差值异常高且无法降低，AMD 建议将这些路径作为异步路径来处理，即实现异步时钟域交汇电路并添加时序例外。

在异步时钟之间添加时序例外

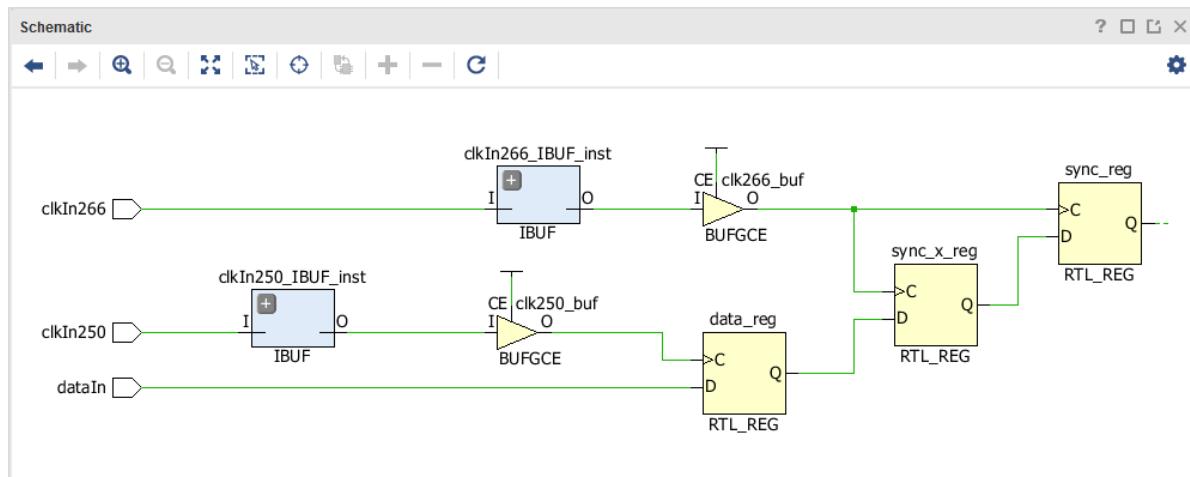
如果时序路径中所含源时钟和目标时钟源自不同基准时钟，或者没有公共节点、公共相位或公共周期，那么这些时序路径必须作为异步时钟来处理。在此情况下，偏差可能极其大，导致无法实现时序收敛。

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。

图 129：具有正确 CDC 电路且不含公共节点的异步 CDC 路径



相关信息

定义时钟组和 CDC 约束

应用可减少时钟偏差的常见技巧

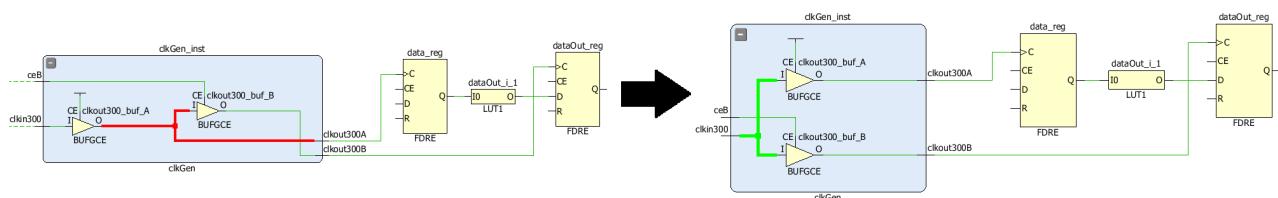


提示：鉴于 UltraScale 器件时钟架构的灵活性，`report_methodology` 命令包含相应的检查以帮助您创建最优的时钟设置拓扑结构。

以下技巧适用于大部分常见场景：

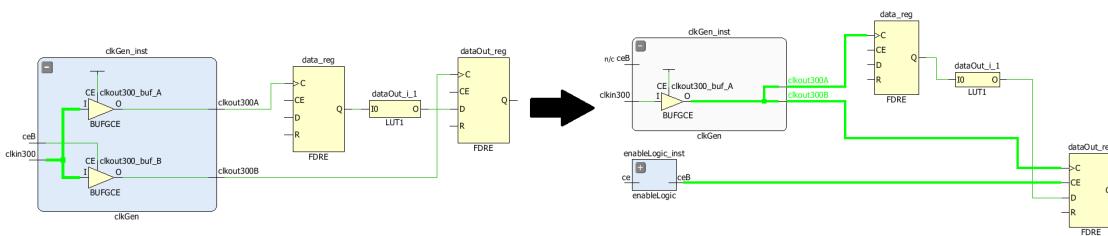
- 通过删除不必要的缓冲器或将其并行连接可以避免在级联时钟缓冲器之间出现时序路径，如下图所示。

图 130：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。

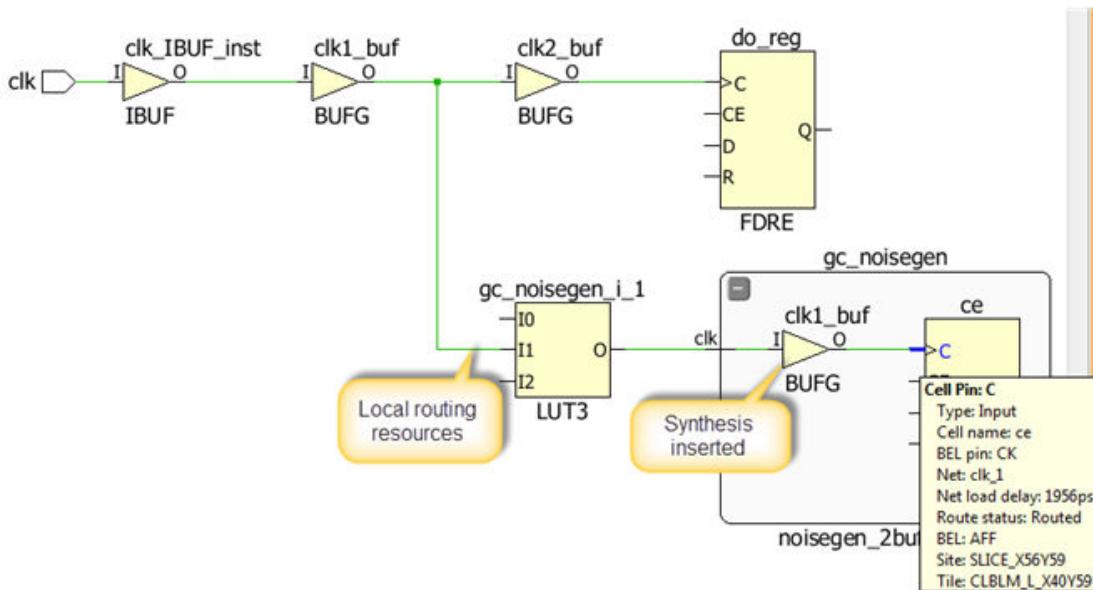
图 131：同步时钟拓扑，合并行时钟缓冲器（已重组为单个缓冲器）



- 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 132：因时钟网络上的局部布线而引起的偏差



重要提示！ 7 系列和 UltraScale 器件时钟架构不尽相同。您必须遵循对应目标架构的时钟设置准则并验证自己的设计是否符合要求。

相关信息

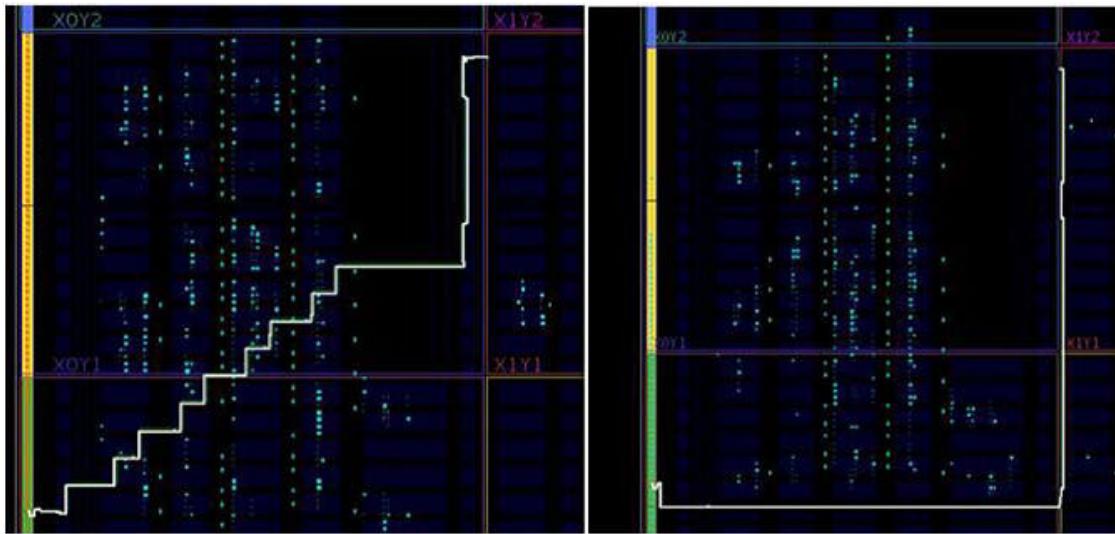
时钟设置准则

应用技巧提升 7 系列器件的偏差

虽然 7 系列和 UltraScale 架构的时钟架构方面存在差异，但有些常规时钟注意事项对这 2 个系列都适用：

- 不得在量产 7 系列设计中使用 CLOCK_DEDICATED_ROUTE=FALSE 约束。CLOCK_DEDICATED_ROUTE=FALSE 只能用作为时钟故障的临时变通方法，并且只能用于生成已实现的设计，以查看要调试的时钟拓扑。采用互连结构布线的时钟路径时钟偏差可能较高，并且可能受开关噪声的影响，从而导致性能欠佳或设计无法正常运作。如下图所示，右侧采用专用时钟布线，而左侧的时钟则禁用专用布线。

图 133：互连结构时钟布线与专用时钟布线的比较

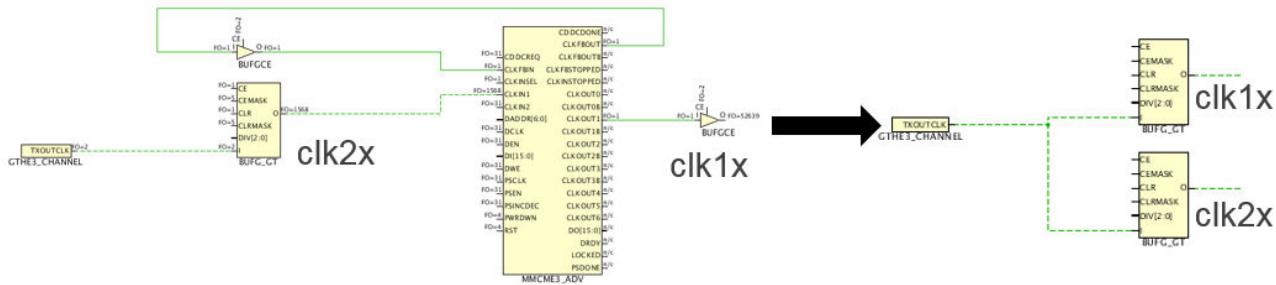


- 不允许区域时钟缓冲器 (BUFR/BUFIO/BUFH) 在多个时钟区域中驱动逻辑，因为这将导致每个区域中的时钟树分支之间的偏差过高。删除不适当的 LOC 或 Pblock 约束即可解决此问题。

降低 UltraScale 和 UltraScale+ 器件的偏差

- 避免使用 MMCM 或 PLL 来执行 BUFG_GT 时钟的简单分频。BUFG_GT 单元能够对输入时钟进行分频。下图显示了如何节省 MMCM 资源，并为源自 GTHE3_CHANNEL 单元的 2 个时钟实现平衡的时钟树。

图 134：使用 UltraScale BUFG_GT 实现平衡的时钟树



- 在关键同步时钟的驱动信号线上使用 CLOCK_DELAY_GROUP 以便在布局和布线期间强制执行 CLOCK_ROOT 及布线匹配。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约束，并且偏差大于预期，则表明时序路径可能跨 SLR 或 I/O 列。发生这种情况时，可将诸如 Pblock 等物理约束用于迫使源和目标进入同一个 SLR 或防止发生跨 I/O 列的现象。
- 处理高速同步时钟域跨时序路径现象时，将时钟修改块（例如，MMCM/PLL）的位置约束到时钟负载中心即可帮助满足时序约束要求。减少时钟网络延迟可减少时钟域交汇路径上的时序消极因素。
- 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_COLUMN 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，从而导致性能欠佳或设计无法正常运作。

相关信息

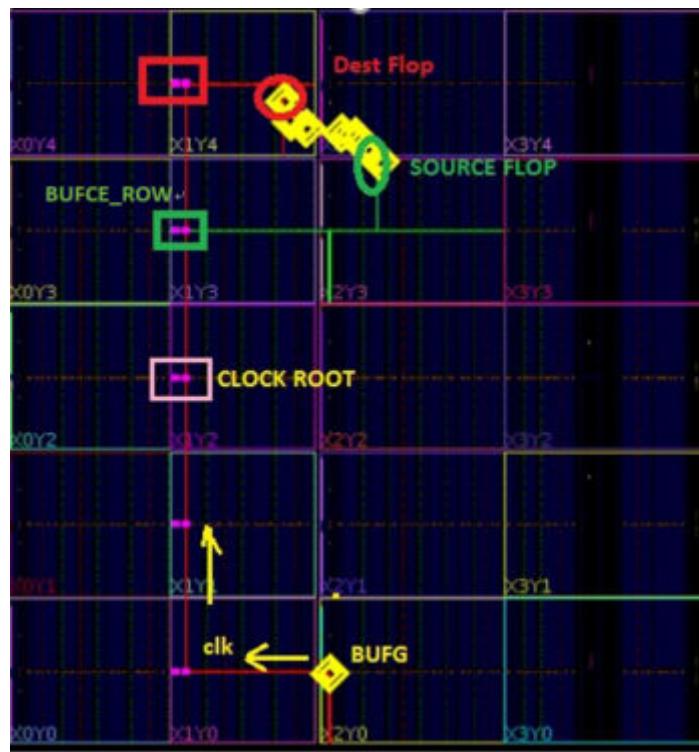
[同步 CDC](#)
[时钟约束](#)

减少 UltraScale 和 UltraScale+ 器件中的时钟延迟

在 UltraScale 和 AMD UltraScale+™ 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向外扩展以通过垂直分布轨道来驱动每个时钟区域中的时钟行。在每一行上，位于 BUFCE_ROW 直通式布线站点上的时钟网络中存在可编程延迟，此类站点可在时钟从时钟根向外扩展的过程中执行低精度的纠偏。

下图显示了从全局时钟缓冲器 (BUFG) 到时钟根的时钟路径。时钟布线从布线切换到垂直分布轨道，穿过驱动水平分布轨道的每个时钟区域行中的 BUFCE_ROW，然后进入叶级。源显示为绿色，目标显示为红色。

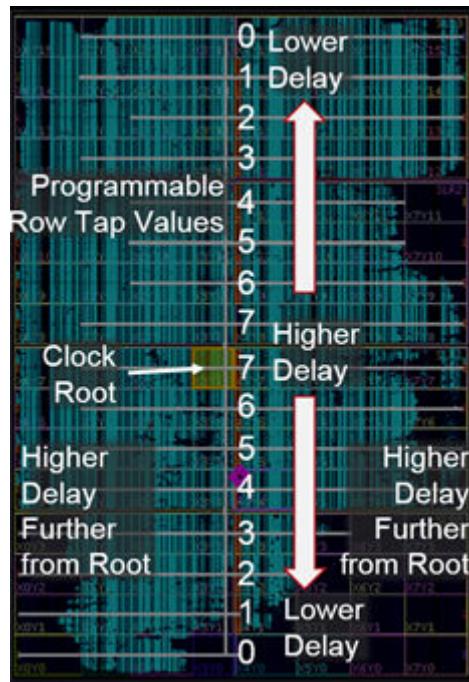
图 135：从 BUFG 穿过 BUFCE_ROW 到叶级的时钟路径



可编程行抽头即时钟根附近最大的延迟。随时钟与根之间垂直方向距离增大，每增加 1 个时钟区域，抽头延迟值即减 1，最终降至 0。

下图显示了从时钟根递减的可编程行抽头的拓扑结构。抽头值越高，表示延迟越高且跨 SLR 时钟偏差越高，原因在于因制造工艺差异导致的最小/最大延迟差值，使抽头值增大，从而使时序不确定性增加。这导致更加难以满足可编程抽头延迟值较高的时钟根附近的时序要求。垂直方向距离时钟根越远，不确定性越低，并且通常更便于修复跨 SLR 总线的保持时间违例。对于水平方向距离时钟根更远的 SLR 交汇总线，时钟行延迟会增加。此额外延迟会导致最小/最大延迟差值增大，从而降低 SLR 交汇的性能。

图 136：跨 UltraScale+ SSI 技术器件的可编程行抽头延迟设置



对于 UltraScale+ SSI 技术器件，您可使用以下任一方法来提高 SLR 交汇速度：

- 将时钟根移至水平方向接近 SLR 交汇处的位置
- 限制可编程行抽头延迟最大值以降低不确定性

注释：垂直方向距离时钟根更远的时序路径可能会因保持时间修复布线绕行而导致变得更慢。但使用这些方法可提升总体性能。

在“Clock Utilization”（时钟利用率）报告中的“Device Cell Placement Summary for Global Clock”（全局时钟的器件单元布局汇总）部分中，可查看 Vivado 工具为设计中每个全局时钟所选的行可编程抽头延迟设置。以下示例显示了 HORIZONTAL PROG DELAY 列中的 g13 全局时钟（黄色高亮部分）的行可编程抽头延迟设置。

图 137：“时钟使用率”报告中的全局时钟行可编程抽头延迟设置

22. Device Cell Placement Summary for Global Clock q13											
Global Id	Driver Type/Pin	Driver Region (D)	Clock	Period (ns)	Waveform (ns)	Root (R)	Slice Loads				
q13	BUFGCE/O	X4Y10	Multiple	4.926	{0.000 2.463}	X3Y8	12511				
* Slice Loads column represents load cell count of all cell types other than IO, GT and clock resources											
** IO Loads column represents load cell count of IO types											
*** Clocking Loads column represents load cell count that are clock resources (global clock buffer, MMCM, PLL, etc)											
**** GT Loads column represents load cell count of GT types											
	X0	X1	X2	X3	X4	X5	X6	X7	HORIZONTAL PROG DELAY		
Y15	2820	4086	308	0	0	0	0	0	0		
Y14	713	392	0	0	3	0	0	0	0		
Y13	0	0	0	0	0	0	0	0	0		
Y12	0	0	0	0	0	0	0	0	0		
Y11	0	0	0	0	0	62	94	3	3		
Y10	0	0	801	3	(D) 45	224	216	0	4		
Y9	0	0	252	91	361	185	10	0	5		
Y8	0	0	66	(R) 261	241	2	0	0	5		
Y7	0	17	365	117	16	0	0	0	4		
Y6	0	165	275	39	2	0	0	0	3		
Y5	0	120	66	0	0	0	0	0	2		
Y4	0	27	7	0	0	0	0	0	1		
Y3	21	21	0	3	0	0	0	0	0		
Y2	11	1	0	0	0	0	0	0	0		
Y1	0	0	0	0	0	0	0	0	0		
Y0	0	0	0	0	0	0	0	0	0		

对于 UltraScale+ SSI 技术器件，布局器会限制行可编程抽头延迟最大值，以减小最小/最大延迟差值，并减小时钟根附近的 SLR 交汇时钟偏差，同时确保通过增大或减小 SLR 交汇任一侧的时钟区域的抽头延迟值来平衡距离时钟根更远的 SLR 交汇路径上的时钟偏差。通过查询时钟信号线的 MAX_PROG_DELAY 属性值，即可找到布局器所用的行可编程抽头延迟最大值。

您还可使用 USER_MAX_PROG_DELAY 属性来限制行可编程抽头值。下面给出了 1 个示例。要设置 USER_MAX_PROG_DELAY 属性，必须直接向全局时钟缓冲器驱动的信号线段应用该值。如果 USER_MAX_PROG_DELAY 属性未设置，布局器可使用可设置的最大抽头值 7。

```
set_property USER_MAX_PROG_DELAY <0-7> [get_nets -of [get_pins BUFG/O]]
```

以下是 USER_MAX_PROG_DELAY 属性的一些使用技巧：

- 对于遍布大部分 UltraScale+ SSI 技术器件的时钟，建议的 USER_MAX_PROG_DELAY 抽头值为 3 或 4。当时钟根靠近 GT、PCIe® 或偏离器件中心的 CMAC 块时，器件上对向的 SLR 交汇性能将受到严重影响，因为发送和捕获时钟的公共节点与 SLR 交汇相距更远。
- 对于使用 CLOCK_DELAY_GROUP 进行时钟网络匹配的时钟组，请确保时钟组内所有时钟都使用相同的 USER_MAX_PROG_DELAY 值。

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 等时钟生成块还能以离散抖动 (Discrete Jitter) 和相位误差 (Phase Error) (如果生成多个相关时钟) 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 MMCM 设置来降低时钟不确定性

注释：`report_qor_suggestions` Tcl 命令会标记此问题。

配置 MMCM 以便执行频率综合时，AMD 建议配置 MMCM 以便在时钟上实现最低输出抖动。将 MMCM 设置最优化为按尽可能最高的压控振荡器 (VCO) 频率运行，前提是此频率满足器件允许的工作范围。以下公式显示了 VCO 频率、M (乘法器)、D (除法器) 与 O (输出除法器) 设置与输入和输出时钟频率之间的关系：

$$F_{VCO} = F_{CLKIN} \times \frac{M}{D}$$
$$F_{OUT} = F_{CLKIN} \times \frac{M}{D \times O}$$



提示：您可通过增大 M 和/或减小 D 来增加 VCO 频率，也可以通过增加 O 来补偿频率变更。增大 VCO 频率会对来自 MMCM 或 PLL 的功耗损耗产生负面影响。从使用 BUFG 的多个 MMCM 时钟输出切换为使用 BUFGCE_DIV 的单一 MMCM 时钟输出时，也可以小幅增加 VCO 频率，这样即可允许更多时钟使用小数分频器。选择 MMCM 或 PLL 时，首选 MMCM，因为 MMCM 能够以更高的 VCO 频率运行、改善选择 M 值和 D 值的粒度并且具有小数分频器 (CLKOUT0)。

不同架构的 VCO 频率最大值也不尽相同。因此，AMD 建议重新生成时钟组件，以便按目标架构获取最优化的时钟组件。AMD 建议使用 Clocking Wizard 来自动计算 M 值和 D 值以及 VCO 频率，以便正确配置目标器件的 MMCM。



提示：通过“IP catalog”(IP 目录) 使用 Clocking Wizard 时，请确保“Jitter Optimization Setting”(抖动最优化设置) 设置为“Minimize Output Jitter”(最大限度减小输出抖动)，这样即可提供更高的 VCO 频率。此外，对所期望的输出时钟频率稍作更改可以进一步提高 VCO 频率，以便进一步降低时钟不确定性。

以下 MMCM 频率综合示例使用输入时钟 62.5 MHz 来生成约 40 MHz 的输出时钟。解决方案有两种，但 VCO 频率更高的 MMCM_2 生成的时钟不确定性更低，因为它降低了抖动和相位误差。

表 14：MMCM 频率综合示例

	MMCM_1	MMCM_2
输入时钟	62.5 MHz	62.5 MHz
输出时钟	40.0 MHz	39.991 MHz
CLKFBOUT_MULT_F(M)	16	22.875
DIVCLK_DIVIDE(D)	1	1
VCO 频率	1000.000 MHz	1429.688
CLKOUT0_DIVIDE_F(O)	25	35.750
抖动 (ps)	167.542	128.632
相位误差 (ps)	384.432	123.641

使用 BUFGCE_DIV 减少时钟不确定性

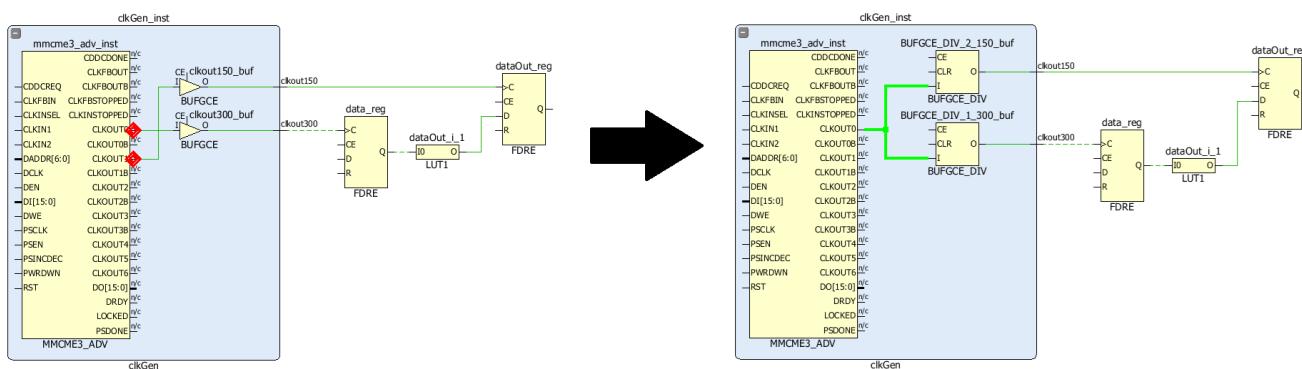


提示：report_qor_suggestions Tcl 命令会标记此问题。

在 UltraScale 器件中，BUFGCE_DIV 单元可用于通过消除 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。

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，并不会生成含 MMCM 的 150 MHz 时钟，而是改为将 BUFGCE_DIV 连接到 300 MHz MMCM 输出，并将该时钟除以 2。为实现最佳结果，300 MHz 时钟还需使用 BUFGCE_DIV 并将 BUFGCE_DIVIDE 设置为 1，以便与 150 MHz 时钟延迟准确匹配，如下图所示。

图 138：为 UltraScale 同步 CDC 时序路径改进时钟拓扑



对于新拓扑结构：

- 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）。
- 公共节点移至更靠近缓冲器位置，从而节省部分时钟消极因素。

通过对 2 个时钟信号线应用 CLOCK_DELAY_GROUP 约束，时钟路径将具有匹配的布线。

注释：report_qor_suggestions Tcl 命令可提供这些约束。

下表提供了 UltraScale 同步 CDC 时序的建立分析与保持分析的时钟不确定性的比较结果。

表 15：UltraScale 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		BUFGCE_DIV 150 MHz 时钟
	总系统抖动 (TSJ)	0.071 ns	0.071 ns
	“Discrete Jitter (DJ)”（离散抖动 (DJ)）	0.115 ns	0.115 ns
	相位误差 (PE)	0.120 ns	0.000 ns
	时钟不确定性	0.188 ns	0.068 ns

表 16：UltraScale 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		BUFGCE_DIV 150 MHz 时钟
总系统抖动 (TSJ) “Discrete Jitter (DJ)” (离散抖动 (DJ)) 相位误差 (PE) 时钟不确定性	总系统抖动 (TSJ)	0.071 ns	0.000 ns
	“Discrete Jitter (DJ)” (离散抖动 (DJ))	0.115 ns	0.000 ns
	相位误差 (PE)	0.120 ns	0.000 ns
	时钟不确定性	0.188 ns	0.000 ns

相关信息

[同步 CDC](#)

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。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。

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 个模块可能需要使用 MUXF* 资源来实现 1 个时序关键函数，但设计其余部分通过实现 LUT 中的逻辑而不是 MUXF* 则可能更有利于减少拥塞。在此情况下，请为时序关键模块设置 PERFORMANCE_OPTIMIZED BLOCK_SYNTH 约束，并使用 Flow_AlternateRoutability 策略来对设计其余部分进行综合以便减少拥塞。

相关信息

[块级综合策略](#)

改进逻辑层次

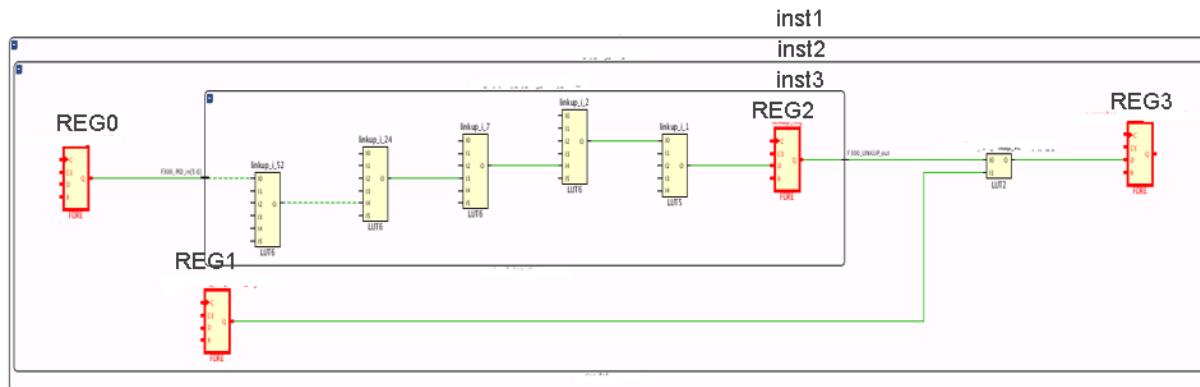
在整个设计周期中，必须验证逻辑层次分布与目标 AMD 器件家族和器件速度等级的时钟频率目标是否相符。

默认情况下启用了全局重定时来提高设计性能，但代价是增加功耗和编译时间。要从某种程度上减少功耗和编译时间，可以考虑禁用全局重定时，仅将重定时用于在逻辑级数相对较高的路径上出现时序违例的特定分层块。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。

注释：您可以将 BLOCK_SYNTH.RETIMING 设置为 1 或 0。如果默认情况下关闭了重定时，但需要对特定面积使用重定时，请将该值设置为 1。如果默认情况下开启了重定时，但不需要对特定面积使用重定时，请将该值设置为 0。

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 139：此原理图显示了含 5 个逻辑层级的关键路径



除了使用 Vivado IDE 中的“Schematic”（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)。

注释：与 `BLOCK_SYNTH.RETIMING` 属性一样，您可以根据是否需要对特定寄存器使用重定时，将 `retiming_forward` 和 `retiming_backward` 同时设置为 0 或 1。

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 140：含默认综合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中的寄存器移入高逻辑层次路径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 个 LUT > REG2（向后重定时），而从 REG2 到 REG4 的路径具有 3 个逻辑层次。

图 141：针对综合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：

- 7 系列器件 slice 或半 CLB 包含 8 个寄存器，这些寄存器全部共享同一个时钟、同一个置位/复位和同一个时钟使能。针对每 8 个一组的寄存器，只能使用 1 个控制集。
- 每个 UltraScale 器件半 CLB 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。固定的置位/复位信号不进行布线，因此可忽略。固定的使能信号作为动态使能信号来处理，需进行布线。在最优状况下，针对每组 8 个寄存器最多可使用 2 个控制集。

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。

尽管 UltraScale 器件 CLB 控制集容量更高，但典型设计所显示的控制集使用率与 7 系列设计相近。因此，对于这两种架构，AMD 给予相同的建议。

遵循控制集指南进行操作

下表提供了根据目标器件大小对应 7 系列器件和 UltraScale 器件建议的控制集数量指南。

表 17：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

要确定设计中的控制集数量：

- 布局前：使用 `report_control_sets -verbose`
- 布局后，使用 `report_utilization`（仅限文本模式）



提示：控制集过多会导致局部拥塞并强制实施布线绕行。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，以判定更适合在综合期间使用的控制集阈值。请注意，增大 `control_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。

- 在综合后使用以下命令之一来合并等效控制集：

```
opt_design -control_set_merge
```

```
opt_design -merge_equivalent_drivers
```

- 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。

相关信息

[控制信号和控制集](#)

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

综合与物理优化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优化。综合期间，可通过在已优化的信号线上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，您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 `-merge_equivalent_drivers` 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 142: 使用 opt_design -control_set_merge 合并的控制集

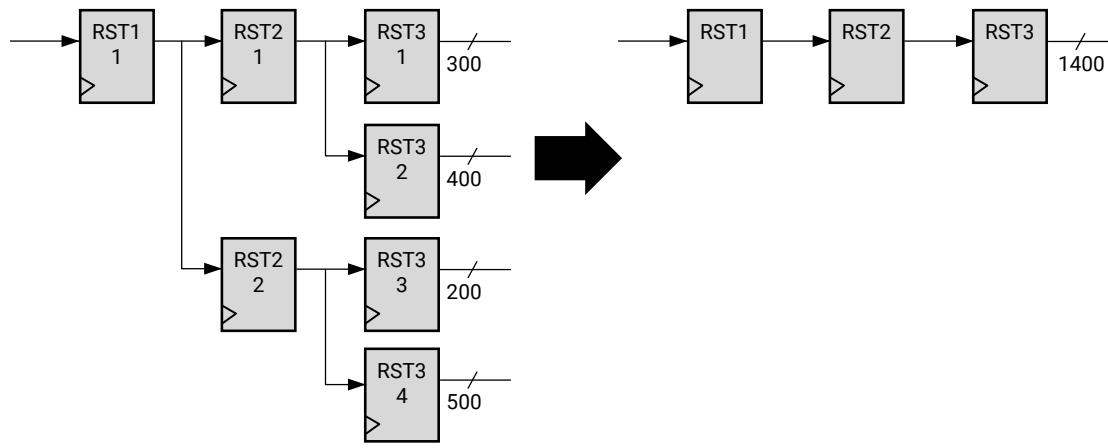
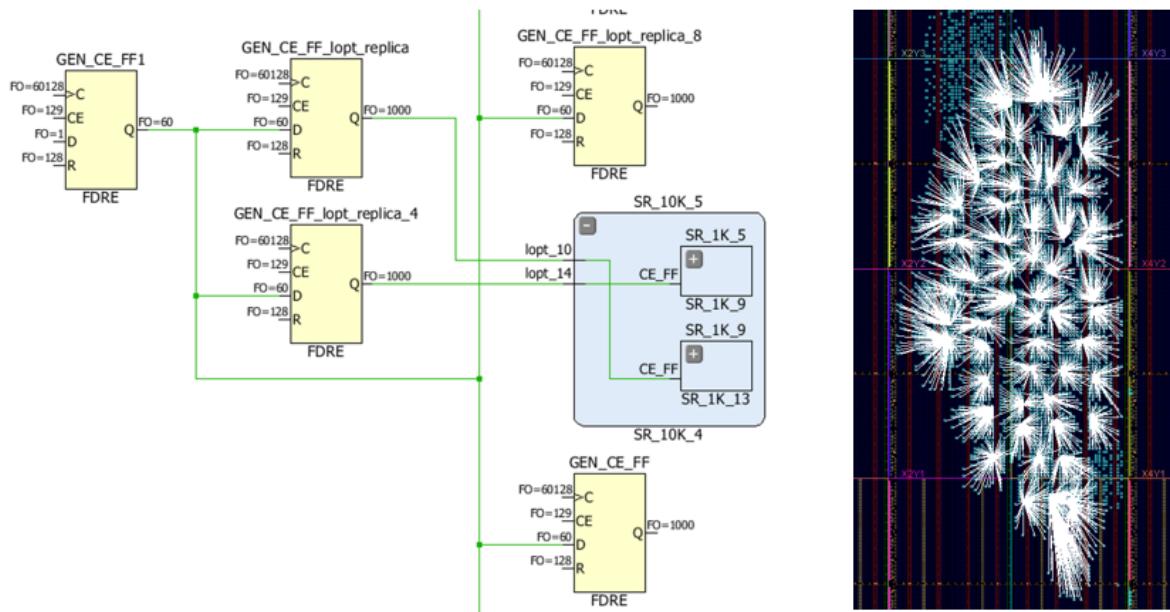


图 143：高扇出时钟使能信号线上基于模块的复制



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。您可通过在信号线上添加 `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。

相关信息

[复制高扇出信号线驱动](#)

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优化。

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 [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。

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。

相关信息

控制信号和控制集

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 2` 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。

注释：只能对整个时钟组设置高优先级，并且只支持一个级别的高优先级。

在时序优化期间，高优先级路径组在布局器和布线器中按最高优先级处理。在 `phys_opt_design` 期间，高优先级路径组会在关键路径优化阶段进行优先级排序。

使用 `-weight` 选项时，请考量以下注意事项：

- 谨慎选择要应用高优先级的路径组，因为这种方法可能会导致其他靠近关键路径的时钟出现劣化。

注释：对较小的时钟域应用高优先级路径组时，其影响更容易控制。AMD 建议将高优先级保留用于高频率时钟。

- 不建议将此选项用于：

- 高度拥塞的设计，因为时序优化可能会进一步加剧拥塞
- 驱动如下模块的时钟：模块所含网表复杂性较高且 Rent 指数较高（高于 0.65）
- 保持时序违例数较高的时钟

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

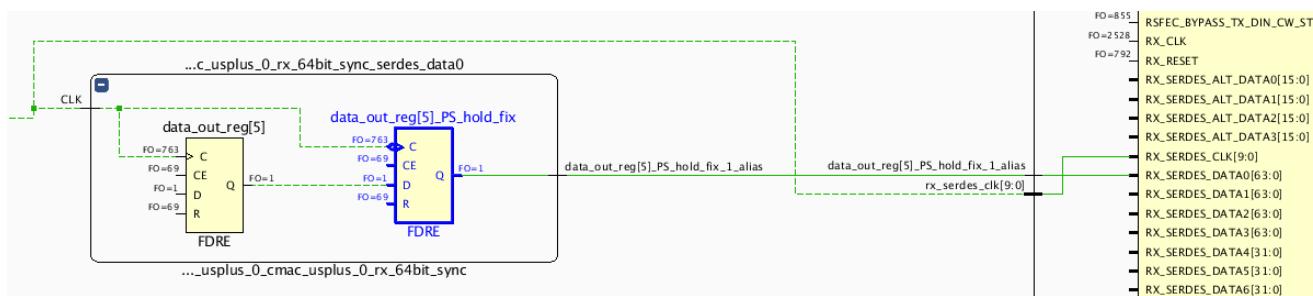
布线前先修复严重保持时间违例

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优化。路径上的建立时序裕量在优化后必须为正值并且足够大，否则将放弃优化。

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 144：通过插入下降沿寄存器修复保持时间违例问题



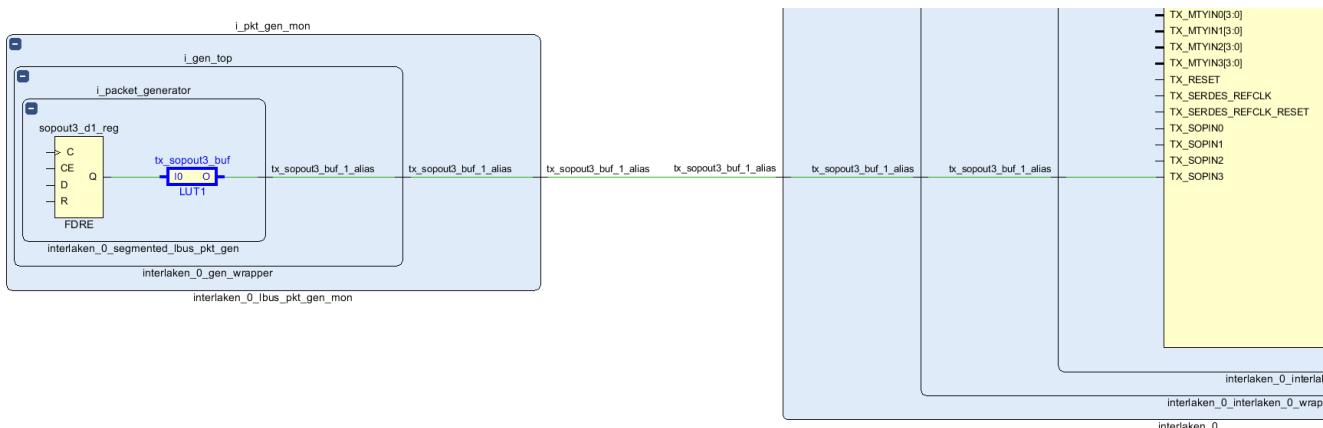
此外，还可将 LUT1 延迟插入数据路径以减少保持时间违例。要插入 LUT1 延迟，请在 `phys_opt_design` 实现步骤中使用以下选项之一：

- `-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` 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。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 145：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_qorSuggestions` 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。



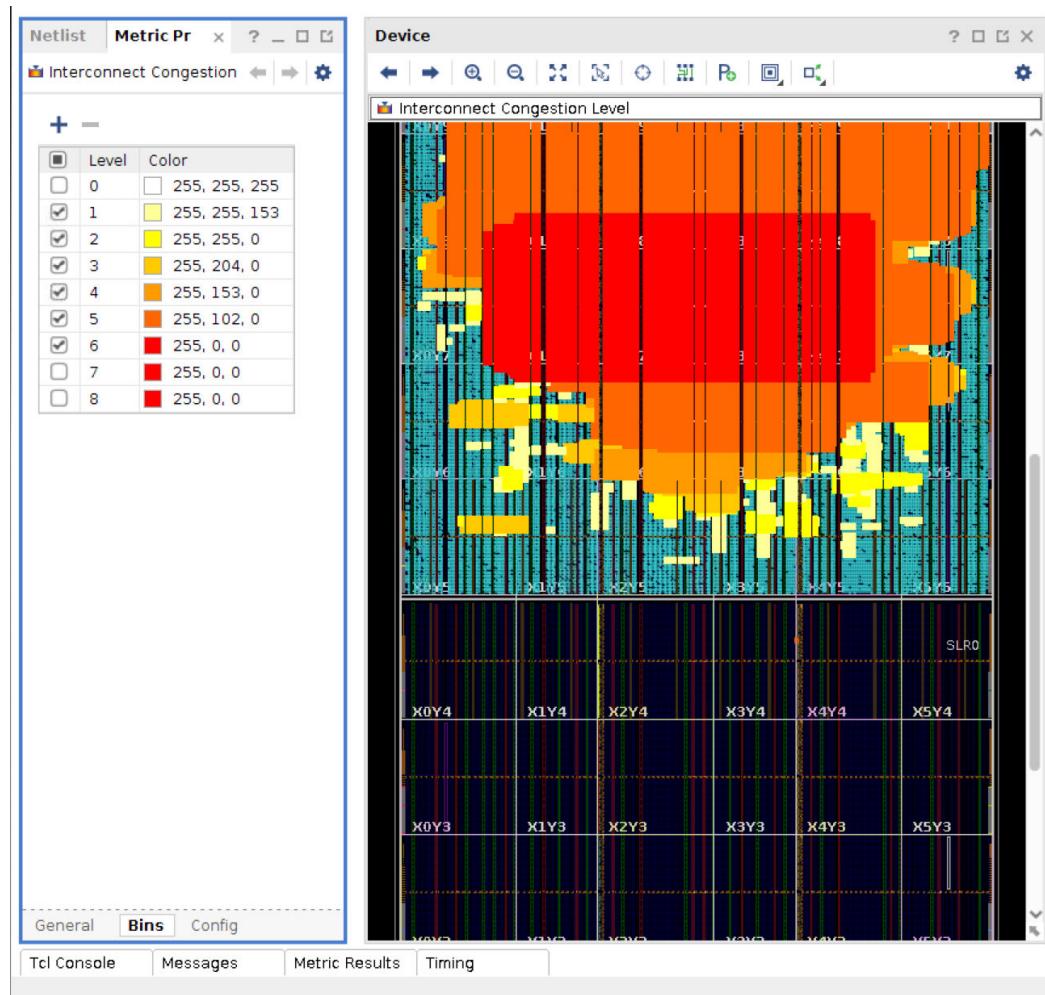
提示：执行 `opt_design` 后请复查资源使用率，以获取删除不使用的逻辑后（而不是综合后）的更准确的数据。

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 146：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
- SSI_SpreadLogic_high

- SSI_SpreadLogic_low

在 SLR 交汇上检测到拥塞时，请考虑使用以下指令：

- SSI_BalanceSLLs 布局器指令，该指令有助于在 SLR 间对设计进行分区，同时尝试平衡 SLR 之间的 SLL。
- SSI_SpreadSLLs 布局器指令，该指令可在 SLR 间进行分区时为区域分配额外面积以提升连通性。

此外，其他布局器指令或实现策略也可能有助于缓解拥塞，使用上述布局器指令后也可一并尝试。

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，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

注释：AlternateCLBRouting 布线指令在以下情况下最有效：仅存在短距离拥塞时或者同时存在短距离拥塞和长距离拥塞时。该指令仅适用于 UltraScale 器件。

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。

减少 MUXF 映射



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

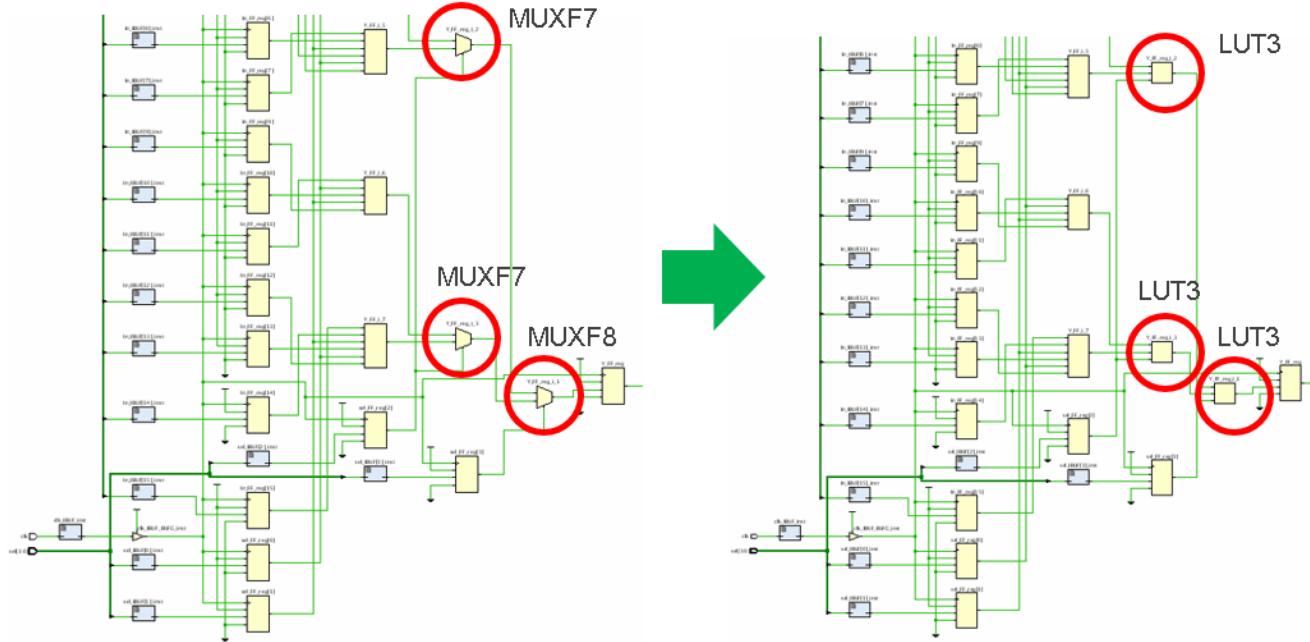
使用 MUXF* 原语能为拥有大量逻辑层或苛刻时钟要求的关键路径提供帮助，同时还可以降低功耗。MUXF* 包括 MUXF7、MUXF8 和 MUXF9，这些均为位于 CLB 中的专用多路复用器资源。这些资源在布局阶段按最多 8 个 LUT 进行分组。这种分组方式会通过提高布线需求来强制提高 CLB 输入使用率，并在网表连接方式较为复杂时限制布局灵活性，从而潜在提高布线拥塞并导致时序劣化。

此外，`opt_design` 命令可提供可选 MUX 最优化阶段以将 MUXF* 结构重新映射至 LUT3 原语，从而提高可布线性。您可使用 `-muxf_remap` 选项以重新映射所有 MUXF* 单元。或者，在已发生拥塞的区域中选定数量的单元上将 `MUXF_REMAP` 属性设置为 TRUE，以限制 MUX 重新映射范围。在 `opt_design` 期间，`MUXF_REMAP` 属性设置为 TRUE 的任意 MUXF* 单元都会自动触发 MUX 最优化阶段并重新映射到 LUT3。

注释：禁用这些资源会导致功耗增加。仅限必要时方可使用此方法来实现时序收敛。

下图显示了 MUXF* 最优化前后的 16-1 MUX。

图 147：MUX 最优化前后的网表



要在执行 MUX 最优化后对网表进行最优化，请使用 `-remap` 选项与 `-muxf_remap` 选项。如果可能，这将把 MUXF* 最优化所生成的 LUT3 原语与已连接的逻辑组合在一起。

布局或布线完成后，您可通过复查 log 日志文件或“Design Analysis”（设计分析）报告 (`report_design_analysis -congestion`) 中的“Router Initial Estimated Congestion”（布线器初始估算拥塞）表来确定时序收敛是否受布线拥塞影响。

在下图中，“Design Analysis”（设计分析）报告显示 7% 的器件受到方向为“South”（南方）的“Short”（短距离）拥塞等级 5 (32x32 CLB) 的影响，并且在对应的拥塞区域中 MUXF 使用率为 26%。

图 148：report_design_analysis 拥塞表格中的南向短距离拥塞

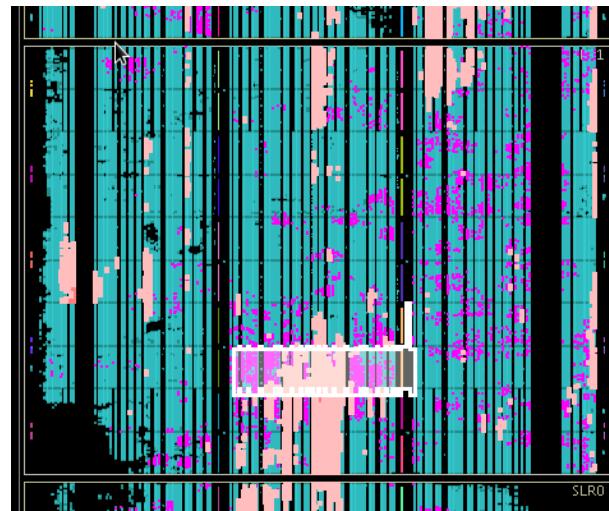
Direction	Type	Congestion Level	Percentage Tiles	Congestion Window	Cell Names	Combined LUTs	LUT6	MUXF	
North	Short	4	6.983%	(CLEL_L_X48Y73,CLEL_R_X63Y88)	inst_name1 (92%)	0%	22%	...	4%
South	Short	5	7.136%	(CLEL_L_X32Y297,CLEL_R_X63Y328)	inst_name2 (99%)	2%	49%	...	26%
East	Short	4	6.005%	(CLEL_L_X48Y109,CLE_M_X63Y125)	inst_name3 (94%)	0%	38%	...	10%
West	Short	4	6.541%	(CLEL_L_X32Y273,CLE_M_X47Y320)	inst_name4 (92%)	1%	48%	...	23%

在 Vivado IDE 中，可选中“Design Analysis”（设计分析）拥塞表中的某一行以在“Device”（器件）窗口中高亮显示对应的拥塞区域。下图显示拥塞与 MUXF 密度较高区域存在重叠。在 Vivado IDE Tcl 控制台中使用以下命令以洋红色高亮显示 MUXF 单元：

```
highlight_objects -color magenta [get_cells -hier -filter REF_NAME=~MUXF*]
```

MUXF* 包含 MUXF7/MUXF8/MUXF9，这些均为位于 CLB 中的专用多路复用器资源。这些资源在布局阶段按最多 8 个 LUT 进行分组，通过提高布线需求强制提高 CLB 输入使用率并限制布局灵活性。使用 Vivado IDE 指标显示每个 CLB 估算的拥塞。

图 149：Vivado IDE 的“Device”窗口中高亮的 MUXF 拥塞



当高 MUXF* 使用率与高拥塞区域重叠时，AMD 建议通过将 MUXF* 的对应功能映射到具有更高的布局和布线灵活性的 LUT 以减少 MUXF* 的数量。您可在 XDC 综合约束中使用以下命令来修改网表：

```
set_property BLOCK_SYNTH.MUXF_MAPPING 0 [get_cells inst_name4]
```

重新运行综合、布局和布线后，“设计分析”报告中经过更新的拥塞表现在会显示“南”向“短”距离拥塞有所降低（等级 4），这通常表示时序结果质量已有所提高。

图 150：降低模块的 MUXF 使用率后的初始布线器拥塞表

Direction	Type	Congestion Level	Percentage Tiles	Congestion Window	Cell Names	Combined LUTs	LUT6	MUXF	
North	Short	4	6.983%	(CLEL_L_X48Y73,CLEL_R_X63Y88)	inst_name1(92%)	0%	22%	...	4%
South	Short	4	9.040%	(CLEL_L_X34Y297,CLEL_R_X49Y312)	inst_name2(99%)	2%	54%	...	4%
East	Short	4	6.005%	(CLEL_L_X48Y109,CLE_M_X63Y125)	inst_name3(94%)	0%	38%	...	10%
West	Short	4	6.541%	(CLEL_L_X32Y273,CLE_M_X47Y320)	inst_name4(92%)	1%	48%	...	23%

禁用 LUT 组合

注释：report_qorSuggestions 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。

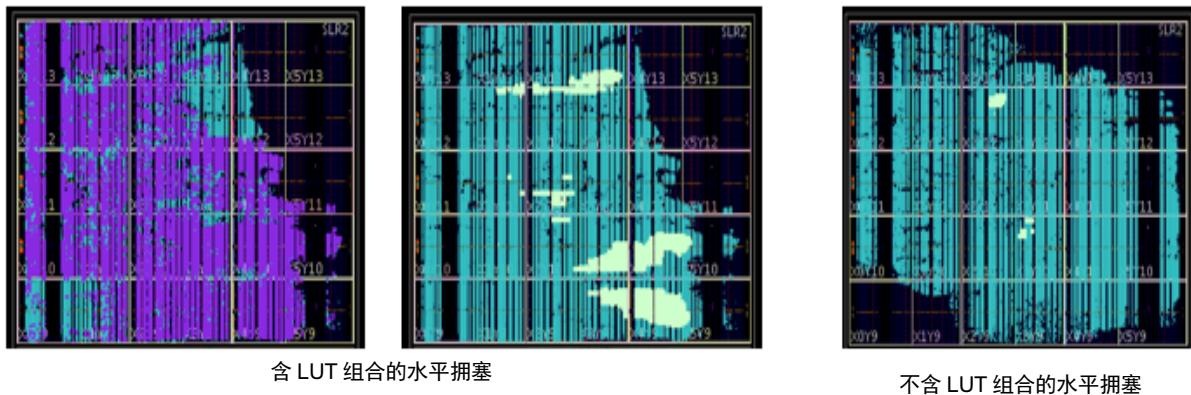
注释：如果您使用 Synplify Pro 进行综合，那么您可以使用“Device”（器件）选项卡下“Implementation Options”（实现选项）中的“Enable Advanced LUT Combining”（启用高级 LUT 组合）选项。默认情况下，该选项处于已启用状态。如果您修改 Synplify Pro 工程文件 (*.prj)，请指定以下内容：set_option -enable_prepacking 1。

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 151：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_qorSuggestions Tcl 命令可自动应用这种最优化技巧。

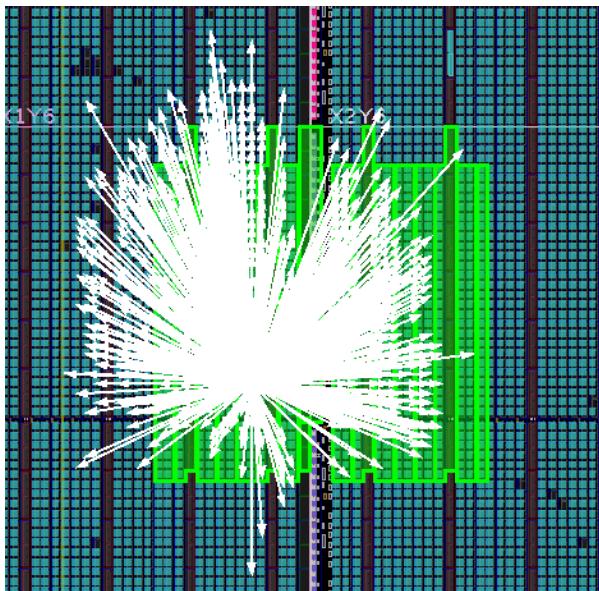
具有严格的时序约束的高扇出信号线要求对布局进行严格分群以满足时序。这可能导致局部拥塞，如下图所示。高扇出信号线还会因占用布线资源，导致拥塞窗口中其他信号线无法使用这些资源，从而促成拥塞。

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 152：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` 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（仅限 UltraScale 器件）或 `LOC` 约束（仅限 7 系列器件）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。在“工程模式”下，每一轮运行都有可用的策略。通过选择运行策略，可以将预选指令应用于运行的各个步骤，并将它们存储为运行属性。您可以通过直接修改这些运行属性或创建自定义运行策略来手动覆盖这些运行属性。在“非工程模式”下，必须手动将指令应用于每条单独的流程命令。

ML 策略

机器学习 (ML) 策略支持您快速获取优化设计策略。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_qorSuggestions ...]`，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。

注释：AMD 不建议为非 SSI 技术器件运行 SSI 技术策略。

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。

对于其中每个布局设计检查点，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 的运行轮次。在“非工程模式”下，必须以 Tcl 脚本显式指定流程，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，从而最大程度改进结果。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。

针对 SSI 技术器件，您还可以考虑使用 SLR Pblock 或软核布局规划约束 (USER_SLR_ASSIGNMENT)。

相关信息

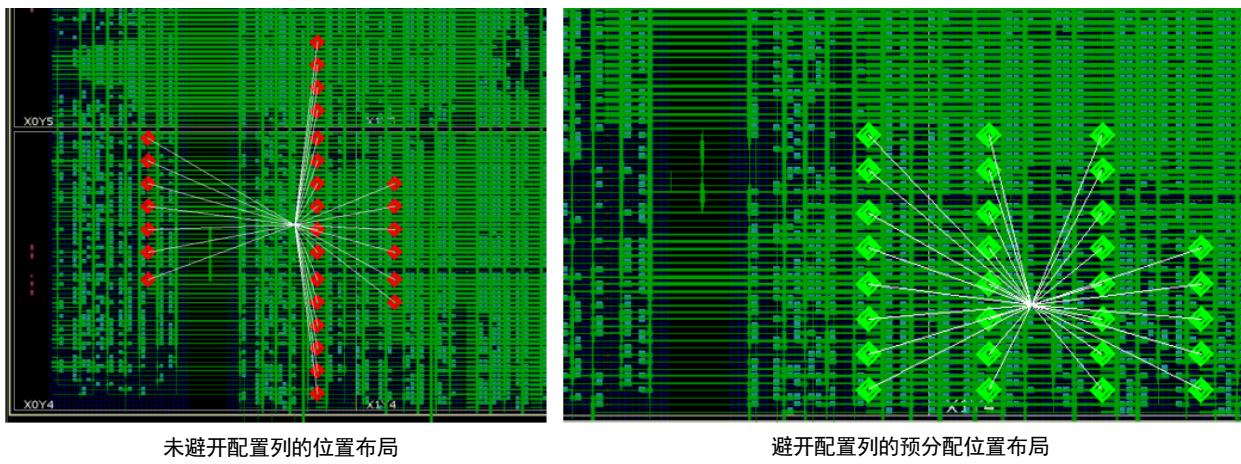
SSI 技术注意事项

对关键逻辑进行分组

对关键逻辑进行分组可避免跨 SLR 或 I/O 列，这样有助于改进设计的关键路径。下图显示了使用 29 个 FIFO36E2 原语实现的 2 个大型 FIFO 示例。其中关键路径为从组中每个 FIFO36E2 的 WRRSTBUSY 管脚穿过 5 个 LUT 到组中每个 FIFO36E2 的 WREN 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。FIFO36E2 原语以红色标记。
- 在右侧，此示例显示的是布局器能够满足时序，因为 FIFO36E2 块采用矩形分组，由此避免了配置列交汇。FIFO36E2 原语以绿色标记。

图 153：避免配置列的位置布局



X18041-053022

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 文件中以供 UltraScale 和 UltraScale+ 器件设计使用。

```
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 $WNS = 0.000\text{ ns}$ 。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}}
```

相关信息

[设计过约束](#)

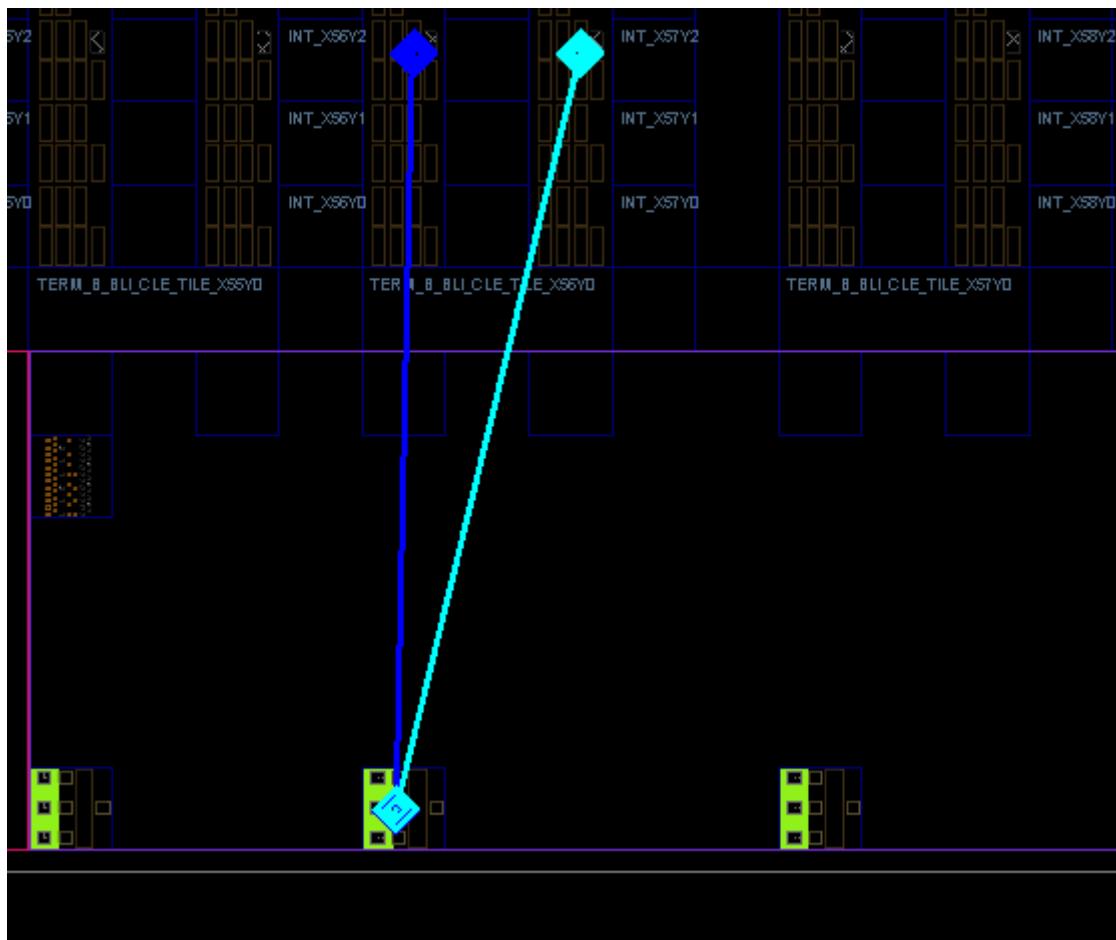
XPIO-PL 接口时序约束方法

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 ODDR1 和 IDDR1 的数据路径使用的正是 BLI 触发器。

```
set_property BLI TRUE [get_cells {oddr_D1_BL1_reg oddr_D2_BL1_reg}]\nset_property BLI TRUE [get_cells {iddr_Q1_BL1_reg iddr_Q2_BL1_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 154：XPIO-PL 接口中 ODDR1 和 IDDR1 的 BLI 触发器布局



SSI 技术注意事项

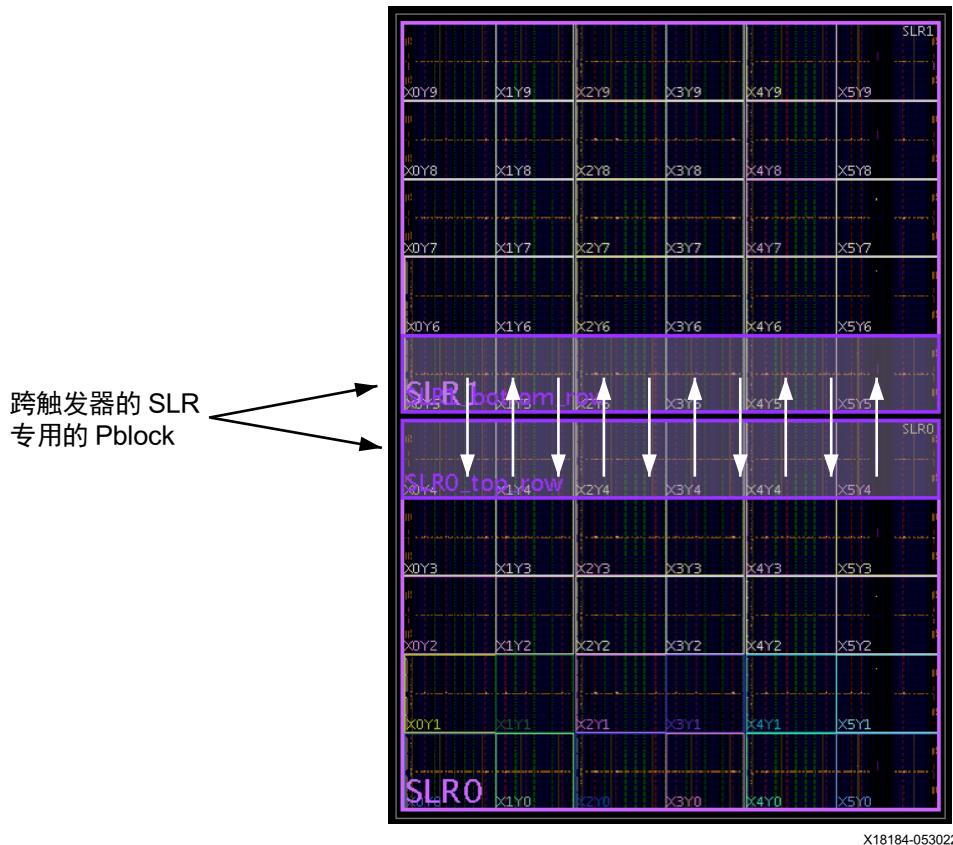
堆叠硅片互联 (SSI) 技术器件包含多个由中介层连接的超级逻辑区域 (SLR)。中介层连接被称为超长线路 (SLL)。当跨 SLR 交汇时会发生延迟惩罚。为了尽量降低 SLL 延迟对设计的影响，请对设计进行布局规划以避免在关键路径内包含 SLR 交汇。通过布局规划来最大限度减少 SLR 交汇，使遇到问题的模块仅保持在单一 SLR 内，这样也可以改进以 SSI 技术器件为目标的设计的时序和可布线性。

使用硬核 SLR 布局规划约束

对于高性能设计，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。以下示例显示了含下列 Pblock 的 UltraScale ku115 SSI 器件：

- 2 个 SLR Pblock：SLR0 和 SLR1
- 2 个 SLR 交汇 Pblock：SLR0_top_row 和 SLR1_bottom_row

图 155：SLR 交汇 Pblock 示例



X18184-053022



重要提示！ AMD 建议针对 SLR 交汇 Pblock 使用 CLOCKREGION 范围代替 LAGUNA 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时序收敛问题的信息，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。

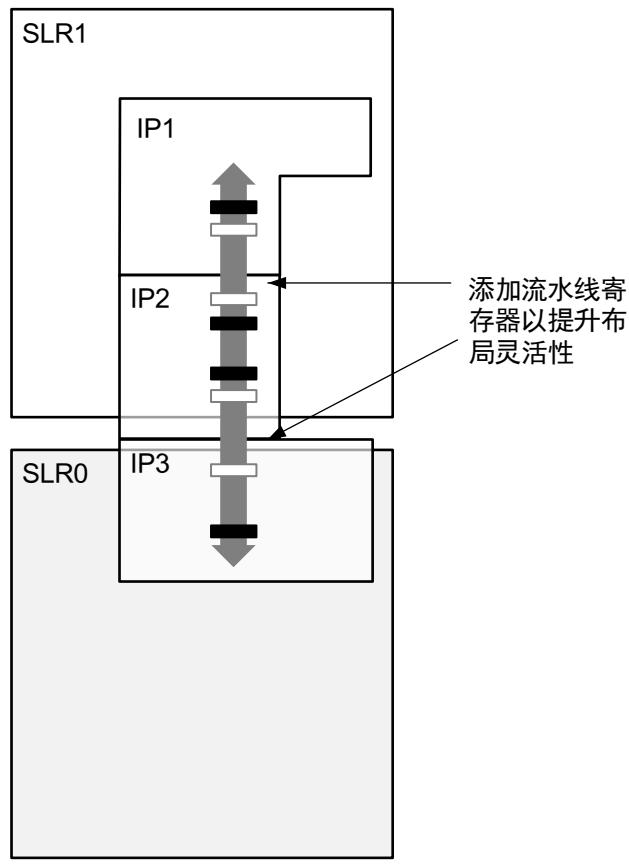
USER_SLR_ASSIGNMENT 和 Pblock 都支持通过详细的布局器和物理最优化来对 SLR 边界附近的叶节点单元布局进行微调以改进时序。这些调整操作包括跨 SLR 边界迁移流水线寄存器，前提是迁移有助于改进时序。不允许跨 Pblock 边界迁移这些寄存器。

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 156：USER_SLR_ASSIGNMENT 属性的布局示例



X21199-053022

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。

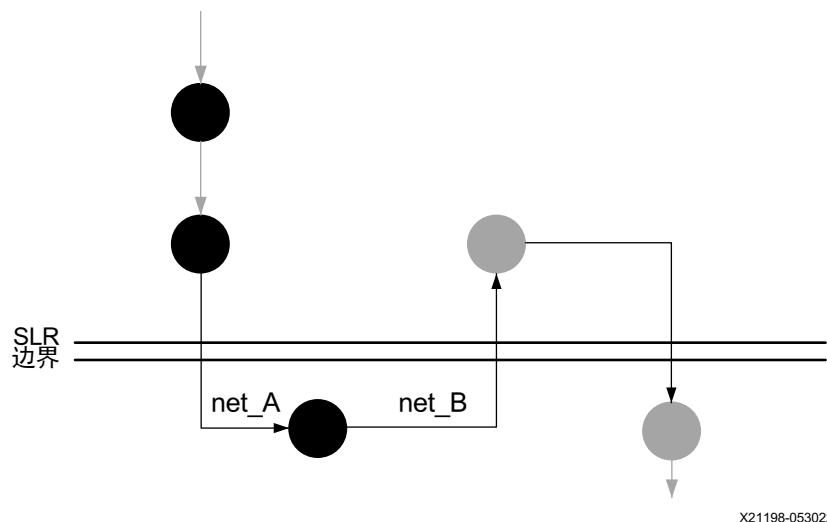
- FALSE：表示目标信号线对象不应跨 SLR 或者目标管脚对象不应跨 SLR 连接。FALSE 值可应用于任意信号线或管脚。

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 157：设置 USER_CROSSING_SLR 属性前的次优 SLR 交汇

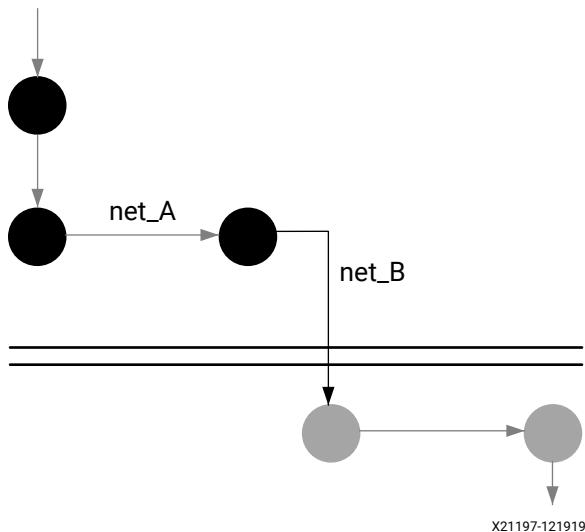


为了实现最优化布局（其中仅有 net_B 跨 SLR），应用以下 XDC 约束：

```
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 158：设置 USER_CROSSING_SLR 属性后的最优化 SLR 交汇



X21197-121919

使用 SLR 交汇寄存器

处理 UltraScale+ SSI 技术器件时，可将寄存器间 SLR 交汇映射到直接驱动 Laguna RX_REG 的 Laguna TX_REG。这种类型的连接仅在 UltraScale+ 器件家族中才能实现，在此类器件中，Vivado 布线器可以通过设置局部可编程时钟延迟来修复保持时间违例。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 Laguna 站点 (site) 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。

注释：处理 UltraScale SSI 技术器件时，只能在 SLR 交汇信号线上使用 Laguna TX_REG 或 RX_REG，并且两者不得同时使用。性能优势与上面列出的类似。

您可在预计将来布局在 Laguna 寄存器站点 (site) 上的 SLR 交汇边界处的寄存器上设置 USER_SLL_REG 属性。如果寄存器 D 和 Q 管脚连接到的信号线并未跨 SLR 边界，或并非用于驱动布局在多个 SLR 中的负载，那么 place_design 会忽略 USER_SLL_REG 约束。例如：

```
set_property USER_SLL_REG TRUE [get_cells {reg_A reg_B}]
```

将寄存的交汇映射到 Laguna 的有效方法之一是对寄存器同时应用 BEL 和 LOC 约束，以将其锁定到位。LOC 值可分配 Laguna 站点 (site)，BEL 值可选择该站点内的特定 Laguna 寄存器，从 6 个 TX_REG 寄存器中选择其中之一，并从 6 个 RX_REG 寄存器中选择其中之一。Laguna 交汇寄存器之间距离固定不变，即每个 TX_REG 寄存器都与 1 个 RX_REG 寄存器配对以实现直接连接。

在以下示例中，在每个 TX_REG 到 RX_REG 的连接上都手动放置 1 个寄存器间连接。流水线寄存器 reg_A 通过寄存器 reg_B 的单一负载来驱动单个扇出。对于 VU9P 目标器件，使用 TX_REG 到 RX_REG 的直接连接来应用以下 XDC 约束，以使 SLR2 中的 reg_A 驱动 SLR1 中 reg_B：

```
set_property BEL TX_REG3 [get_cells reg_A]
set_property BEL RX_REG3 [get_cells reg_B]
set_property LOC LAGUNA_X2Y480 [get_cells reg_A]
set_property LOC LAGUNA_X2Y360 [get_cells reg_B]
```

先应用 BEL 分配，寄存器位置 (0、1、...5) 必须在 TX_REG 和 RX_REG 间匹配，在本例中为 3。最后，配对的 Laguna 站点 (site) 之间的距离是 120 行。寄存器 reg_A 可从 SLR2 Laguna 列的最下一行驱动到 SLR1 Laguna 列的最下一行。在创建 LAGUNA BEL 和 LOC 约束时，请尝试使用相同的时钟、时钟使能及复位信号对寄存器进行分组，以避免出现控制集兼容性问题。

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。使用此功能时，Vivado 布局器会自动使用 Laguna 寄存器，无需额外干预。

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。

相关信息

[自动流水打拍注意事项](#)

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](#))。



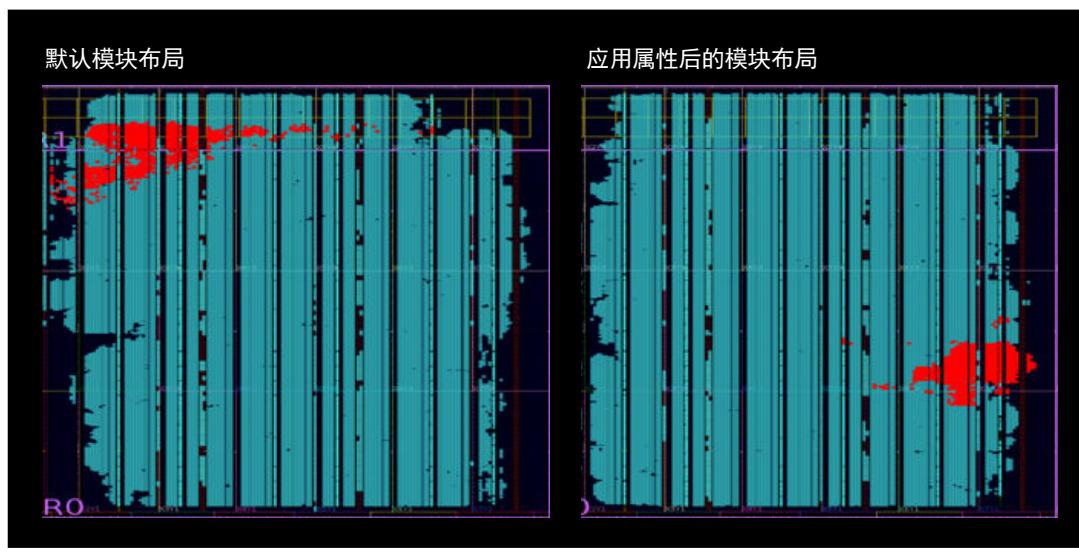
重要提示！ 布局器将该属性视为应尽可能尝试遵循的准则。但您可覆盖该属性来达成有效的布局结果。



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 159：使用 USER_CLUSTER 属性前后的实例布局



X27256-101022

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。
- 器件必须源自基于 UltraScale 器件的家族或基于 AMD UltraScale+™ 器件的家族。
- 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：功耗分析与优化》(UG907)。



建议：在针对性处理 UltraScale 和 AMD UltraScale+™ 器件并使用 Explore 指令或基于 Explore 的策略时，必须在运行 opt_design 之后通过运行 power_opt_design 或者使用 opt_design -bram_power_opt 来手动启用块 RAM 功耗最优化。AMD 建议尽可能减少存储器资源以降低功耗。复查“RAM Utilization Report”(RAM 使用情况报告)中的比特使用情况，查找含低效映射的存储器阵列。对于使用块存储器和 LUTRAM 组合的最高效的阵列映射，还请考虑使用 HDL RAM_STYLE MIXED 属性。

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 UltraScale+ 器件，您可以从包含环境设置的电源设计管理器 (PDM) 工具（从 www.amd.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。您可以使用 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 XPE 或 PDM 工具电子数据表中。这样便可以：
 - 在 XPE 或 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。

除了对最佳和最差逻辑和布线延迟下实现的电路功能进行核实之外，仿真器还可报告内部节点的精确活动（包括毛刺）。在实际测量原型开发板上的功耗之前，此级别的功耗分析可以提供最为精确的功耗估算。

Using the Power Constraints Advisor

Power Constraint Advisor 用于报告设计中所有控制信号上的工具计算的开关活动，从最高扇出开始排序。复查该列表中可信度级别为“低”的项，这些等级表示含开关活动量较高的复位信号以及开关活动量极低或为 0 的使能信号。这两个因素都会导致功耗结果盲目乐观的错误结果。欲知详情，请访问此[链接](#)以参阅《Vivado Design Suite 用户指南：功耗分析与优化》(UG907) 中的相应内容。

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, ...}
```

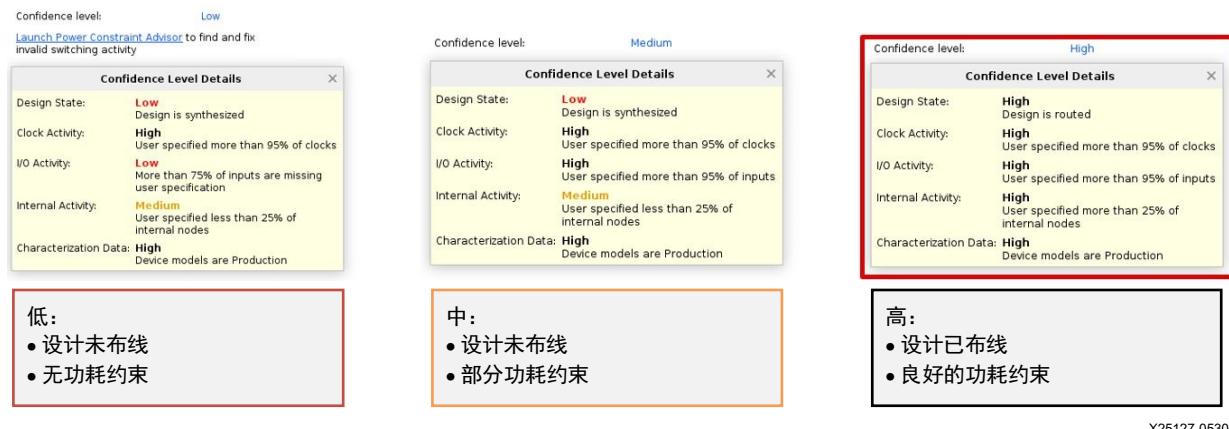
定义电流预算

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

有助于精成功耗分析的最佳实践

要精成功耗分析，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：功耗分析与优化》(UG907) 中的相应内容。

图 160：功耗分析可信度



X25127-053022

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：功耗分析与优化》(UG907) 中的相应内容。

根据当前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 -theta_ja <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 Xilinx Power Estimator (XPE) 或电源设计管理器 (PDM) 工具（从 www.amd.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。

运行功耗分析后进一步优化控制信号活动

尚未使用基于 SAIF 的注解进行准确的功耗分析时，可以在执行第一级分析后对功耗分析进行微调。欲知详情，请访问此[链接](#)以参阅《Vivado Design Suite 用户指南：功耗分析与优化》(UG907) 中的相应内容。

功耗最优化

如果功耗估算超出预算，那么您必须遵循以下章节中所述步骤来降低功耗。

分析功耗估算和最优化结果

使用 report_power 生成功耗估算报告后，AMD 建议执行以下操作：

- 在“Summary”部分中检查总功耗。总功耗和结温是否符合热处理与功耗预算？
- 如果结果严重超出预算，应根据块类型和电源轨检查功耗分布汇总情况。这样便于您了解哪些块功耗最高。
- 复查“Hierarchy”部分。按层级细分后，功耗最高的模块将清晰可见。您可深入查看具体模块，以确定块的功能。也可以在 GUI 中进行交叉探测，以确定模块特定部分的编码方式以及是否可通过能效更高的方法对其进行重新编码。

注释：如果设计具有时序裕度，请运行多次以评估某一次运行的总功耗是否更低。例如，具有 2 ps 裕度的设计的性能与具有 15 ps 的设计性能相近，但 2 ps 设计功耗可能更低。

运行功耗最优化

功耗最优化适用于整体设计或部分设计（使用 `set_power_opt`），用于最大限度降低功耗。

功耗最优化可在设计流程中布局前或布局后运行，但不能在布局前和布局后同时运行。布局前功耗最优化步骤着重于最大限度节省功耗。但这可能导致时序劣化（虽然并不常见）。如果主要目标是保留时序约束，AMD 建议采用布局后功耗最优化步骤。该步骤仅执行用于保持时序约束的功耗最优化步骤。

当出于传统 IP 或时序约束考量而需要保留部分设计时，请使用 `set_power_opt` 命令来排除这些相应部分（例如，特定层级、时钟域或单元类型），然后重新运行功耗最优化。

相关信息

[改善功耗的编码样式](#)

使用功耗优化报告

为确定功耗优化的影响，请在 Tcl 控制台中运行如下命令以生成功耗优化报告：

```
report_power_opt -file myopt.rep
```

使用时序报告来判定功耗最优化的影响

功耗最优化的作用是最大限度降低对时序的影响，同时最大限度节省功耗。但在特定情况下，如果功耗最优化后时序发生劣化，那么可以采用多种方法来抵消此影响。

请使用 `set_power_opt XDC` 命令尽可能仅对非时序关键型时钟域或模块识别并应用功耗最优化。如果最关键的时钟域恰好涵盖设计的绝大部分或者耗用的功耗最多，请复查关键路径，查看其中是否有任何单元包含 `IS_CLOCK_GATED` 属性，且属性值为 `TRUE`，该属性值表示相关路径为功耗最优化所导致的结果。如需以在后续实现中提升功耗为代价来改进时序，请使用 `set_power_opt XDC` 约束来对关键路径中已成功耗最优化的单元禁用功耗最优化。然后，使用 `set_power_opt XDC` 约束或 `Tcl` 命令重新运行实现。

以下 `Tcl` 示例演示了如何对前 100 条失败路径中的单元禁用功耗最优化：

```
set pwr_critical_cells [get_cells -of [get_timing_paths -slack_lesser_than 0 -max_paths 100] -filter {IS_CLOCK_GATED}]  
set_power_opt -exclude_cells $pwr_critical_cells
```

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](#)。

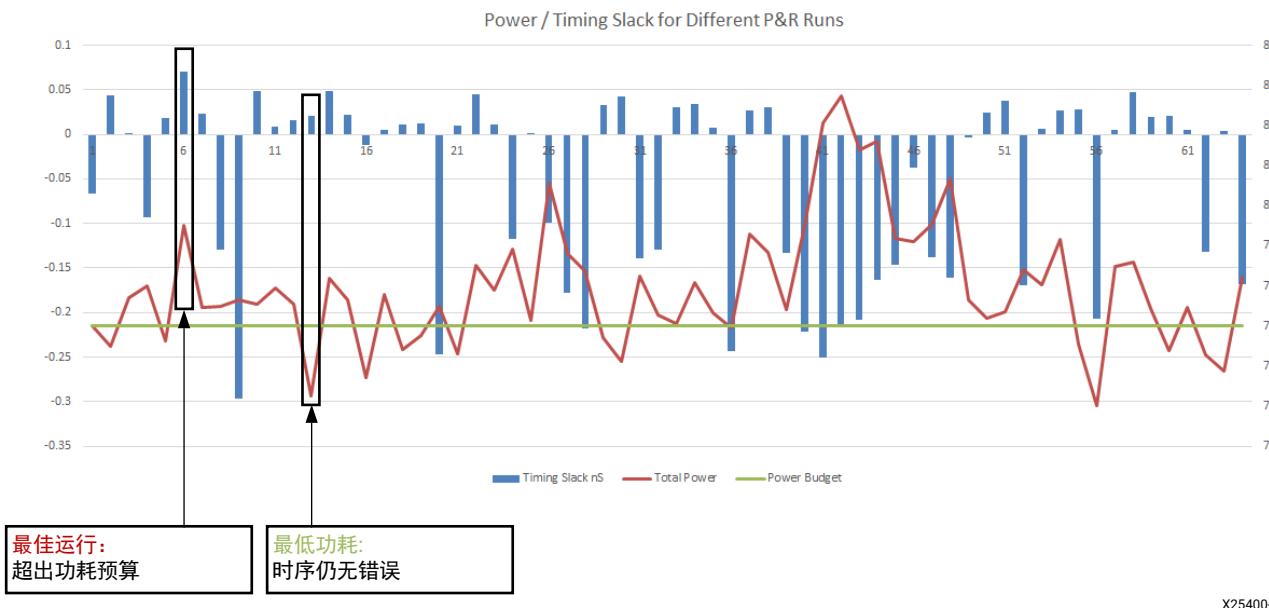
下图显示了此方法的示例。对于全部 64 次时序收敛运行，已同时运行功耗报告，并且所有运行都绘制在一起。在此图中，36 次运行没有时序错误，从功耗角度而言，总功耗预算为 77W。64 次运行在 75 W 到 83 W 范围内，即范围为 8 W 或 ~10%。

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 161：不同布局布线运行的功耗和时序裕量



X25400-060421

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 都会在比特流生成期间变成错误。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)。

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：烧录和调试》([UG908](#))
- 《Vivado Design Suite Tcl 命令参考指南》([UG835](#))
- 《7 系列 FPGA 配置用户指南》([UG470](#))
- 《UltraScale 架构配置用户指南》([UG570](#))
- [Vivado Design Suite QuickTake 视频：在 Vivado 中如何使用“write_bitstream”命令](#)

配置

您必须首先成功完成设计综合与实现，然后才能创建比特流镜像。生成比特流并且对所有 DRC 都完成分析和更正后，即可使用以下任一方法将比特流加载到器件上：

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

可使用 Vivado 工具来完成下列操作：

- 创建比特流 (.bit 或 .rbt)。
- 选择 “Tools” → “Edit Device”（工具 > 编辑器件）以复查比特流生成的配置设置。
- 将比特流格式化为闪存编程文件 (.mcs)。
- 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。除非指定全芯片擦除，否则仅擦除指定 MCS 覆盖的地址范围。



重要提示！ Vivado Design Suite Device Programmer 可使用 JTAG 来读取 AMD 器件上的状态寄存器数据。
如发生配置故障，则该状态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，该状态寄存器还支持您对模式管脚设置 M[2:0] 和总线宽度检测结果进行验证。如需了解有关状态寄存器的详细信息，请参阅对应您的器件的《配置用户指南》。



提示：如果配置不成功，您可对器件使用 JTAG 回读/验证操作来判定是否已将期望的配置数据正确加载到器件中。

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：烧录和调试》([UG908](#))。

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。

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 18：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。	<ul style="list-style-type: none">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： <ul style="list-style-type: none">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。	<ul style="list-style-type: none">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。

表 18：调试流程 (续)

调试流程名称	流程步骤	优势和劣势
基于 Tcl 的网表插入探测流程	使用 <code>set_property</code> Tcl 命令在调试信号线上设置 <code>MARK_DEBUG</code> 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。	<ul style="list-style-type: none"> 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。

相关信息

[修改已实现的网表以替换现有调试探针](#)

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 keep : 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 ILA 核选择的时钟均为自由运行的时钟。否则可能造成在器件上加载设计时无法与调试核通信。
- 请确保提供给 `dbg_hub` 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。可使用 `connect_debug_port` Tcl 命令将 Debug Hub 的 `clk` 管脚连接到自由运行的时钟。
- 在添加调试核之前完成设计上的时序收敛。AMD 不建议使用调试核来调试时序相关问题。
- 如果仍发现因添加 ILA 调试核而导致时序劣化，并且关键路径位于 `dbg_hub` 中，请执行以下步骤：
 1. 打开已综合的设计。
 2. 找到网表中的 `dbg_hub` 单元。
 3. 转至 `dbg_hub` 的“Properties”（属性）窗口。
 4. 找到 `C_CLK_INPUT_FREQ_HZ` 属性。
 5. 将其设置为连接到 `dbg_hub` 的时钟频率 (Hz)。
 6. 找到 `C_ENABLE_CLK_DIVIDER` 属性并将其启用。
 7. 重新执行设计实现。
- 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 19：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	<ul style="list-style-type: none">· 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	<ul style="list-style-type: none">· 不增加块 RAM· 适度增加 LUT/FF 数量
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： <ul style="list-style-type: none">· 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	<ul style="list-style-type: none">· 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	<ul style="list-style-type: none">· 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	<ul style="list-style-type: none">· 每个 ILA 核额外增加块 RAM· 少量增加 LUT/FF 数量

表 19：ILA 特性对设计时序和资源的影响（续）

ILA 特性	用途	时序	区域
探针端口数量	探测大量信号线	影响程度：低	<ul style="list-style-type: none">· 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。
- 将输入探针通过流水线输送到 ILA (C_INPUT_PIPE_STAGES)，可增加流水线阶段的层级。

注释：对于 MMCM/BUFG 可用性受限的设计，请考虑对 Debug Hub 进行时钟设置，并在设计中采用最低的时钟频率来替代时钟分频器。

使用 VIO 核

Virtual Input/Output (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。VIO 调试核必须在设计中例化，并且可在 Vivado IP integrator 块设计和 RTL 中使用。在 IP 目录中包含 VIO 核，可在基于 RTL 的设计和 IP integrator 中使用。

如需了解有关定制 VIO 核的信息，请参阅《Virtual Input/Output LogiCORE IP 产品指南》(PG159)。如需了解有关利用 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：烧录和调试》(UG908) 中的相应内容。

VIO 核注意事项

使用 VIO 核时，请考量以下注意事项：

- 连接到 VIO 输入探针的信号必须与连接到 VIO 核上的 VIO clk 端口的时钟保持同步。连接信号与 clk 端口不同步会导致在 VIO 输入探针端口上出现时钟域交汇。
- 从 VIO 输出探针驱动的信号将与连接到 VIO 核上的 VIO clk 端口的时钟保持同步断言有效和同步断言无效。
- 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 IP integrator 中对设计进行调试

Vivado IP integrator 提供了另一种方法用于对设计进行设置以供调试。您可使用以下流程之一向自己的 IP integrator 设计添加调试核。所选流程取决于您的偏好以及您要调试的信号线和信号类型。

- 在块设计中使用 System ILA 核来对接口和/或信号线进行调试

使用该流程执行如下操作：

- 使用 MicroBlaze™ 器件、AMD Zynq™ 7000 SoC 或 Zynq UltraScale+ MPSoC 的交叉触发功能执行软硬件协同验证。
 - 验证接口级连接功能。
- 网表插入流程

在综合后设计阶段使用该流程来分析 I/O 端口和内部信号线。

注释：您还可组合使用这 2 个流程来对设计进行调试。

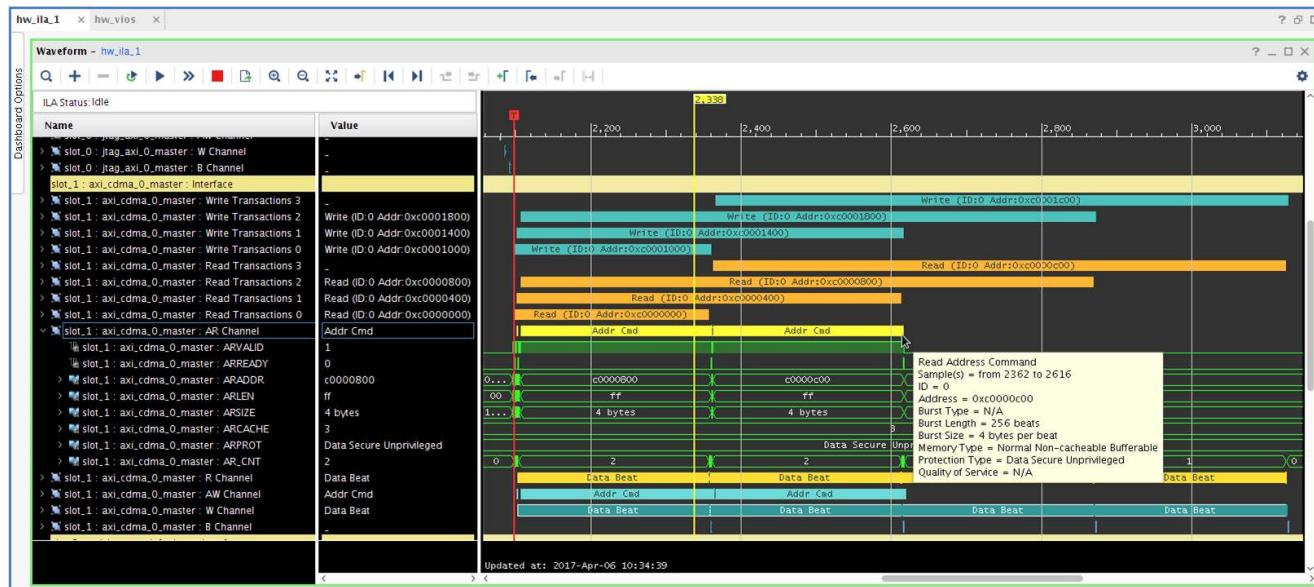
如需了解有关在 IP integrator 设计中使用 System ILA 的更多信息，请参阅《Vivado Design Suite 用户指南：采用 IP integrator 设计 IP 子系统》(UG994)。

在 Vivado Hardware Manager 中调试 AXI 接口

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 ILA 模式更改为“Interface”（接口），即可在“Waveform”（波形）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 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 162：“Waveform”窗口



如需了解有关 Vivado Hardware Manager 中的 System ILA 和 AXI 接口调试的更多信息，请访问此[链接](#)和此[链接](#)以参阅《Vivado Design Suite 用户指南：烧录和调试》(UG908) 中的相应内容。

使用 In-System IBERT

In-System IBERT 核通过 UltraScale 和 UltraScale+ 器件中收发器 RX 数据上的眼图扫描图来提供 RX 裕度分析。该核支持对 GTH/GTY 收发器进行配置和调整，并且可通过与收发器的动态重配置端口 (DRP) 进行通信的逻辑来访问。该核可用于更改属性设置以及可控制 rxrate、rxlpmen、txdiffctrl、txpostcursor 和 txprecursor 端口上的值。

当在器件上对设计进行编程时，硬件管理器 (Hardware Manager) 中的 Vivado Serial I/O Analyzer 可通过 JTAG 与核进行通信。每个设计都只需 1 个 In-System IBERT 实例。In-System IBERT 适用于设计中使用的所有 GT。但是，必须根据不同的 GT 类型（例如，GTH、GTY）生成单独的 In-System IBERT 核。

创建具有内部系统时钟的 In-System IBERT 设计可阻止执行扫描。在创建眼图扫描时，状态从“In Progress”改为“Incomplete”。当将内部系统时钟 (MGTREFCLK) 连接到 In-System IBERT IP 的 clk/drpcclk_i 输入端口时，眼图扫描是不完整的。

注释：如果需要，可考虑采用不会表现出这种行为的外部时钟。或者，单击 Vivado Serial I/O Analyzer 中的任意可用链路。转到“Properties”（属性）窗口，在 LOGIC 字段下找到 MB_RESET 寄存器。将其设为 1，然后切换回 0，最后重新运行眼图扫描或扫描。

如需了解有关此核的更多信息，请参阅《In-System IBERT LogiCORE IP 产品指南》([PG246](#))。

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。

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：烧录和调试》([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 Vivado 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：烧录和调试》([UG908](#)) 中的相应内容。

附加资源与法律声明

查找其他文档

技术信息门户网站

AMD 技术信息门户网站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。要访问该技术信息门户网站，请转至 <https://docs.amd.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

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

以下文档提供了适用于本指南的补充材料。

1. [Vivado Design Suite 文档](#)
2. 《UltraFast 设计方法快捷参考指南》([UG1231](#))
3. 《UltraFast 设计方法时序收敛快捷参考指南》([UG1292](#))
4. 《UltraFast 设计方法检查表》([XTP301](#))

Vivado Design Suite 用户指南及参考资料指南

1. 《Xilinx Power Estimator 用户指南》([UG440](#))
2. 《Vivado Design Suite Tcl 命令参考指南》([UG835](#))
3. 《Vivado Design Suite 用户指南：设计流程概述》([UG892](#))
4. 《Vivado Design Suite 用户指南：使用 Vivado IDE》([UG893](#))
5. 《Vivado Design Suite 用户指南：使用 Tcl 脚本》([UG894](#))
6. 《Vivado Design Suite 用户指南：系统级设计输入》([UG895](#))
7. 《Vivado Design Suite 用户指南：采用 IP 进行设计》([UG896](#))
8. 《Vivado Design Suite 用户指南：I/O 管脚分配和时钟规划》([UG899](#))
9. 《Vivado Design Suite 用户指南：逻辑仿真》([UG900](#))
10. 《Vivado Design Suite 用户指南：综合》([UG901](#))
11. 《Vivado Design Suite 用户指南：使用约束》([UG903](#))
12. 《Vivado Design Suite 用户指南：实现》([UG904](#))
13. 《Vivado Design Suite 用户指南：设计分析与收敛技巧》([UG906](#))
14. 《Vivado Design Suite 用户指南：功耗分析与优化》([UG907](#))
15. 《Vivado Design Suite 用户指南：烧录和调试》([UG908](#))
16. 《Vivado Design Suite 用户指南：Dynamic Function eXchange》([UG909](#))
17. 《Vivado Design Suite 用户指南：入门指南》([UG910](#))
18. 《Vivado Design Suite 属性参考指南》([UG912](#))
19. a. 《Vivado Design Suite 7 系列 FPGA 和 Zynq 7000 SoC 库指南》([UG953](#))
b. 《UltraScale 架构库指南》([UG974](#))
20. 《Vivado Design Suite 用户指南：版本说明、安装和许可》([UG973](#))
21. 《Vivado Design Suite 用户指南：采用 IP integrator 设计 IP 子系统》([UG994](#))
22. 《UltraFast 嵌入式设计方法指南》([UG1046](#))
23. 《Vivado Design Suite 用户指南：创建和封装定制 IP》([UG1118](#))
24. 《电源设计管理器用户指南》([UG1556](#))

Vivado Design Suite 教程

1. 《Vivado Design Suite 教程：设计流程概述》 ([UG888](#))
2. 《Vivado Design Suite 教程：逻辑仿真》 ([UG937](#))
3. 《Vivado Design Suite 教程：Dynamic Function eXchange》 ([UG947](#))

其他 AMD 文档

1. 《UltraScale 架构 FPGA 存储器 IP LogiCORE IP 产品指南》 ([PG150](#))
2. 《UltraScale 器件 Gen3 Integrated Block for PCI Express LogiCORE IP 产品指南》 ([PG156](#))
3. 《Virtual Input/Output LogiCORE IP 产品指南》 ([PG159](#))
4. 《In-System IBERT LogiCORE IP 产品指南》 ([PG246](#))
5. a. 《7 系列 FPGA SelectIO 资源用户指南》 ([UG471](#))
b. 《UltraScale 架构 SelectIO 资源用户指南》 ([UG571](#))
6. a. 《7 系列 FPGA 时钟资源用户指南》 ([UG472](#))
b. 《UltraScale 架构时钟资源用户指南》 ([UG572](#))
7. a. 《7 系列 FPGA PCB 设计指南》 ([UG483](#))
b. 《UltraScale 架构 PCB 设计用户指南》 ([UG583](#))
c. 《Zynq 7000 SoC PCB 设计指南》 ([UG933](#))
8. a. 《UltraScale 架构 GTH 收发器用户指南》 ([UG576](#))
b. 《UltraScale 架构 GTY 收发器用户指南》 ([UG578](#))
9. 《7 系列 FPGA 存储器资源用户指南》 ([UG473](#))
10. 《7 系列 DSP48E1 slice 用户指南》 ([UG479](#))
11. 《7 系列 FPGA 与 Zynq 7000 SoC XADC 双 12 位 1 MSPS 模数转换器用户指南》 ([UG480](#))
12. 《UltraScale 架构 DSP slice 用户指南》 ([UG579](#))
13. 《Zynq 7000 SoC 和 7 系列器件的存储器接口解决方案》 ([UG586](#))
- 14.
15. 《Vitis HLS 用户指南》 ([UG1399](#)) 中的 **HLS 编程指南**
16. 《使用 S 参数模型对 FPGA 功耗完整性进行仿真》 ([WP411](#))
17. 《利用温度漂移扩展热处理解决方案》 ([WP517](#))
18. 《结合使用 SPI 闪存和 7 系列 FPGA》 ([XAPP586](#))
19. 《利用 7 系列 FPGA 实现 BPI 快速配置和 iMPACT 闪存烧录》 ([XAPP587](#))
20. 《参考系统：使用 IP integrator 执行 Kintex 7 MicroBlaze 系统仿真》 ([XAPP1180](#))
21. 《UltraScale FPGA BPI 配置与闪存烧录》 ([XAPP1220](#))
22. 《在 UltraScale FPGA 中进行 SPI 配置与闪存烧录》 ([XAPP1233](#))
23. 《使用加密确保 7 系列 FPGA 比特流的安全》 ([XAPP1239](#))
24. 《无盖倒装芯片封装的机械和散热设计指南》 ([XAPP1301](#))

-
25. 《使用 SelectIO 接口组件原语进行设计》(XAPP1324)
 26. a. 《7 系列板级原理图审查建议》(XMP277)
 - b. 《Kintex UltraScale 和 Virtex UltraScale FPGA 板级原理图审查检查表》(XTP344)
 - c. 《UltraScale+ FPGA 和 Zynq UltraScale+ 器件板级原理图审查检查表》(XTP427)
-

培训资料

1. [UltraFast 设计方法培训课程](#)
 2. [Vivado Design Suite QuickTake 视频：UltraFast Vivado 设计方法](#)
 3. [Vivado Design Suite QuickTake 视频：Vivado 设计流程概述](#)
 4. [Vivado Design Suite QuickTake 视频：使用 Vivado IP integrator 定位 Zynq](#)
 5. [Vivado Design Suite QuickTake 视频：在 Vivado Design Suite 中进行部分重配置](#)
 6. [Vivado Design Suite QuickTake 视频：创建不同类型的工程](#)
 7. [Vivado Design Suite QuickTake 视频：管理工程源文件](#)
 8. [Vivado Design Suite QuickTake 视频：使用 Vivado Design Suite 版本控制](#)
 9. [Vivado Design Suite QuickTake 视频：管理 Vivado IP 版本升级](#)
 10. [Vivado Design Suite QuickTake 视频：I/O 管脚分配概述](#)
 11. [Vivado Design Suite QuickTake 视频：在 Vivado 中配置和管理可复用 IP](#)
 12. [Vivado Design Suite QuickTake 视频：在 Vivado 中如何使用 “write_bitstream” 命令](#)
 13. [Vivado Design Suite QuickTake 视频：设计分析和平面布局](#)
 14. [Vivado Design Suite QuickTake 视频：UltraFast 设计方法检查表简介](#)
 15. [Vivado Design Suite 视频教程](#)
-

修订历史

2025 年 5 月 29 日：与 Vivado Design Suite 2025.1 一起发布，较 2024.2 没有改动。

章节	修订综述
2024 年 12 月 18 日 2024.2 版	
使用 group_path 命令排列关键逻辑的优先顺序	添加有关高优先级路径组的信息。
2024 年 6 月 26 日 2024.1 版	
复查逻辑层次和布线分布表	添加有关布线分布表的信息。

请阅读：重要法律声明

本文档所示信息仅做参考，其中可能包含不准确的技术信息、疏漏和印刷错误。受诸多原因影响，此处所含信息可能发生更改，也可能无法准确呈现，这些原因包括但不限于产品和路线图变更、组件和主板版本更改、新增模型和/或产品发布、不同制造商之间存在的产品差异、软件更改、BIOS 刷新、固件升级等。任何计算机系统均存在安全性漏洞风险，无法彻底阻止也无法缓解这类风险。AMD 没有任何义务来更新或者以任何其他方式纠正或修改这些信息。但 AMD 保留随时修改这些信息和更改文档内容的权利，AMD 没有任何义务将此类修改或更改通知任何人。此处信息

“按原样”提供。AMD 对于本文档内容不作任何陈述或保证，并且对于这些信息中可能出现的任何不准确、错误或疏漏问题不承担任何责任。对于有关任何暗含的非侵权、适销性及适合特定用途的保证，AMD 特此声明不承担任何责任。无论在任何情况下，对于任何人因使用此处包含的任何信息而形成的依赖或者引发的任何直接、间接、特殊或其他后果性损害，AMD 概不负责，即使 AMD 已明确获悉存在发生此类损害的可能性也是如此。

关于与汽车相关用途的免责声明

如将汽车产品（部件编号中含“XA”字样）用于部署安全气囊或用于影响车辆控制的应用（“安全应用”），除非有符合 ISO 26262 汽车安全标准的安全概念或冗余特性（“安全设计”），否则不在质保范围内。客户应在使用或分销任何包含产品的系统之前为了安全的目的全面地测试此类系统。在未采用安全设计的条件下将产品用于安全应用的所有风险，由客户自行承担，并且仅在适用的法律法规对产品责任另有规定的情况下，适用该等法律法规的规定。

版权声明

© Copyright 2013 - 2024 AMD 公司，版权所有。AMD、AMD 箭头标识、Kintex、UltraScale、UltraScale+、Virtex、Vitis、Vivado、Zynq 及其组合均为 Advanced Micro Devices, Inc. 的商标。“AMBA”、“AMBA Designer”、“Arm”、“ARM1176JZ-S”、“CoreSight”、“Cortex”、“PrimeCell”、“Mali”和“MPCore”为 Arm Limited 在美国和/或其他国家或地区的商标。“OpenCL”和“OpenCL”徽标均为 Apple Inc. 的商标，经 Khronos 许可后方能使用。“PCI”、“PCIe”和“PCI Express”均为 PCI-SIG 拥有的商标，且经授权使用。“MATLAB”和“Simulink”均为 The MathWorks, Inc. 拥有的注册商标。此出版物中所使用的其他产品名称仅用于标识目的，可能是其各自所属公司的商标。