UltraFAST 设计方法指南（适用于 Vivado Design Suite）

UG949（v2017.2）2017 年 06 月 07 日

条款中英文版本如有歧义，概以英文文本为准。

修订历史

2017/06/07：与 Vivado® Design Suite 2017.2 一起发布，较 2017.1 没有改动。

日期	版本	修订
2017 年 5 月 26 日	2017.1	更新的内容基于新的 Vivado IDE 外观和感觉。 在第 3 章"设计创建"，"使用寄存器复制"中添加了关于 ASYNC_REG 属性的详细信息，添加了"分解更深的存储器配置，实现功耗与性能平衡"；"使用 CLOCK_DEDICATED_ROUTE 约束"中添加了表和已更新的实例；"规划 IP 要求"中添加了"SelectIO 时钟"，以及 DCP 文件的注解；更新了"报告进出端口的时序"中的实例。 在第 4 章"实现"，将自上而下的综合流程替换为"块级综合策略"和已更新的"使用增量编译流程"。 在第 5 章"设计收敛"，"确认没有时钟遗漏"中更新了属性值；"报告设计分析拥塞报告"中添加了关于拥塞表格的详细信息，添加了"用块级综合策略提升网表"，并更新了"减少控制集"；"使用寄存器复制"中添加了 -merge_equivalent_drivers 和 -fanout_opt 选项的信息；"将高扇出网络推广到全局布线"中添加了 -no_bufg_opt 选项；"使用替代布局和布线指令"中添加了布线器指令的信息；"禁用 LUT 组合和 MUXF 调用"中添加了 MUXF_REMAP 属性的信息；"在拥塞区域中限制高扇出网络"中更新了"使用块级综合策略"，添加了自动优化的信息，并更新了"使用增量编译"。 在附录 A：附加资源与法律提示中，移除了 Vivado Design Suite QuickTake 视频教程：定制和实例化 IP 以及 Vivado Design Suite QuickTake 视频教程：为 UltraFast 设计方法培训课程设计 Vivado IP 集成器并添加参考。

目录

第 1 章

引言

关于 UltraFast 设计方法

赛灵思 UltraFast™ 设计方法是用于为当今 All Programmable 器件优化设计进程的一套最佳实践。这些设计的规模与复杂性需要执行特定的步骤与设计任务，从而确保设计每一个阶段的成功开展。依照这些步骤，并遵循最佳实践，将帮助您以最快的速度和最高的效率实现期望的设计目标。

为帮助您有效利用 UltraFast 设计方法的优势，赛灵思提供了下列资源。

- 本指南中描述了各种设计任务、分析与报表特性，以及用于设计创建和收敛的最佳实践。
- UltraFast 设计方法检查表可通过 Xilinx Documentation Navigator 访问，另外也能够以单独电子数据表的形式查看。您可以借助该检查表认清设计进程中的常见错误与决策点。
- 可以在 Vivado® Design Suite 中使用 Tcl 命令 `report_methodology`，在每个设计阶段做设计方法论相关的设计规则检查 (DRC)。
- UltraFast 设计方法系统级设计流程框图展示了完整的 Vivado Design Suite 设计流程图，这个可以在 Xilinx Documentation Navigator 中找到。您可以通过单击框图中的设计步骤打开相关文档、辅助材料，以及常见问题解答，帮助启动设计。

利用本指南

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。其包含针对下列话题的高层次信息、设计指导方针与设计决策利弊取舍：

- 第 2 章"单板和器件规划"：涵盖在设计创建之前阶段赛灵思推荐的决策与设计任务。包括 I/O 与时钟规划、PCB 布局考虑因素、器件容量与吞吐量评估、替代器件的定义、功耗估算和调试。
- 第 3 章"设计创建"：涵盖用于 RTL 定义、 IP 配置与管理和约束分配的最佳实践。
- 第 4 章"实现"：涵盖用于综合与实现设计的可用选项和最佳实践。
- 第 5 章"设计收敛"：涵盖用于设计时序收敛或降低功耗的各种设计分析与实现技巧。还包括用于为设计硬件验证添加调试逻辑的考虑因素。

本指南包括对其它文档的引用，诸如 Vivado Design Suite 用户指南、 Vivado Design Suite 教程，和 QuickTake 视频教程。本指南不能替代上述技术文档。赛灵思仍然建议您参阅上述技术文档，以了解工具使用和设计方法方面的详细信息。参考文件列表，请参阅附录 A：附加资源与法律提示。

注释： 该信息专为支持 Vivado Design Suite® 使用而编制，大部分概念信息您也可用于 ISE Design Suite。

使用 UltraFAST 设计方法检查表

要全面发挥 UltraFast 设计方法的作用，应将本指南与 UltraFast 设计方法检查表结合使用。本检查表可通过 Xilinx Documentation Navigator 访问，另外也能够以单独电子数据表 (XTP301) 的形式查看。

UltraFast 设计方法检查表中的问题，重点指出了通常难以发现或经常忽略的问题，而这些典型的问题会导致设计决策在后期形成分歧。检查表中的每个视图：

- 面向常规设计团队中的特定角色。
- 包含设计流程每个步骤中常见问题与推荐的应对措施，包括项目规划、主板与器件规划、 IP 与子模块设计，以及顶层设计收敛。
- 还包含文档与培训章节，列出了设计流程步骤的相关资源。
- 提供指向本指南或其他赛灵思文档中内容的链接，为应对相关问题引起的设计顾虑提供了指导。

视频： 如欲了解检查表的演示，请观看 Vivado Design Suite QuickTake 视频：介绍 UltraFAST 设计方法检查表。

使用 UltraFAST 设计方法 DRC

Vivado Design Suite 包含一套方法相关的 DRC，您可以通过 Tcl 命令 `report_methodology` 运行。该命令的规则针对下列各设计阶段：

- 在细化 RTL 中综合前开展 RTL 结构验证
- 综合后验证网表和约束
- 设计实现后验证约束与时序相关问题。

建议： 为达到最佳效果，请在设计的每一个阶段运行方法 DRC，并在开展下一阶段前解决发现的任何问题。

如需了解更多有关设计方法 DRC 的信息，请参阅 《Vivado Design Suite 用户指南：设计分析和收敛技术》(UG906) [参照 21] 中的链接，并请参阅《Vivado Design Suite Tcl 命令参考指南》(UG835) [参照 13] 中的 report_methodology Tcl 命令。

使用 UltraFast 设计方法系统级设计流程图

下图展示了 Vivado Design Suite 中包含的各种设计步骤以及特性。您可以通过 Xilinx Documentation Navigator 访问该图的互动版，单击每个步骤将链接至相关资源。

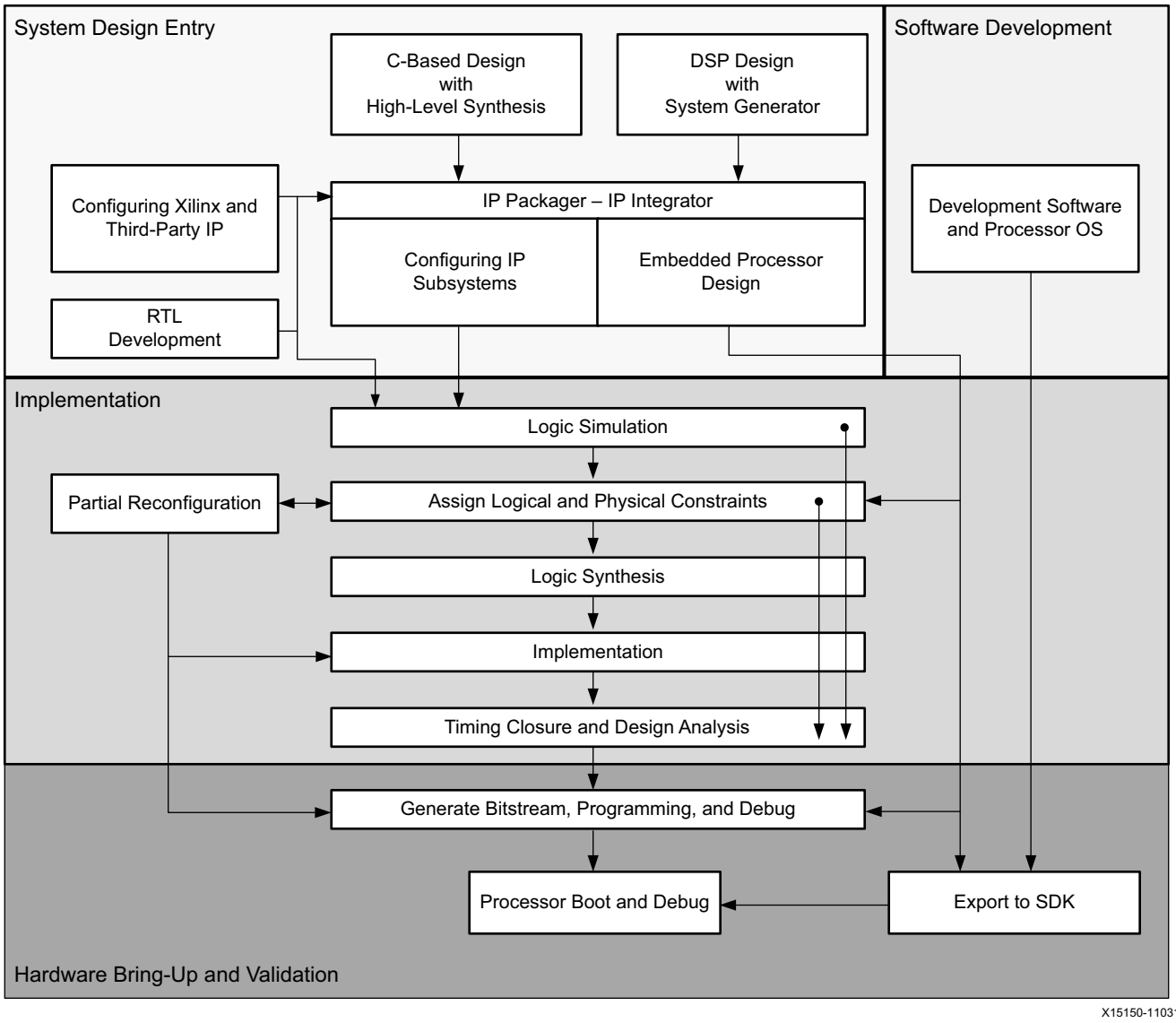


图 1-1：UltraFast 设计方法系统级设计流程

理解 UltraFast 设计方法概念

在设计开始初期就采取正确方法非常重要，此外还应该尽早仔细关注设计目标，包括 RTL、时钟、引脚，以及 PCB 规划。在每个设计阶段对设计进行正确定义并验证，有助于减少后续实现阶段的时序收敛、布线收敛和功耗问题。

在开发周期的早期最大化影响力

如下图所示，设计流程早期阶段 （C、 C++ 和 RTL 综合）对设计性能、密度和功耗的影响要远远高于后续实现阶段。所以，如果设计没有满足时序目标要求，那么赛灵思建议您重新分析综合阶段，包括 HDL 和约束，而不建议您只是在实现阶段通过尝试设计反复来寻找解决方案。

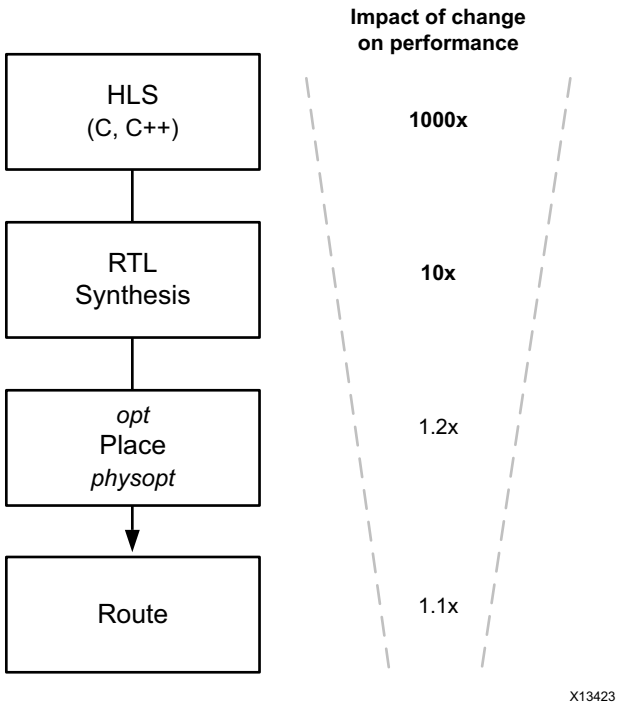


图 1-2：整个流程中设计修改的影响

在每一个设计阶段进行验证

UltraFast 设计方法重点强调监测设计预算的重要性，包括占位区域、功耗、时序，并在早期阶段采取如下措施修正设计：

- 通过赛灵思模板创建最佳 RTL 结构，并在综合前采用 DRC 法验证您的 RTL。

  由于 Vivado 工具从头到尾整个过程使用时序驱动的算法，因此设计从设计流程一开始就要得到正确的约束。

- 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及有关生成时钟之间的关系。在 Vivado 工具中，每对有数据交互的时钟都需要做时序约束，除非明确声明为异步时钟域或伪路径。

- 在开展下一个设计阶段前采用正确的约束满足时序要求。

  采纳以下建议并配合使用 Vivado Design Suite 的交互分析环境可以加速整个时序与实现收敛。

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。

下图展示了该推荐设计方法。

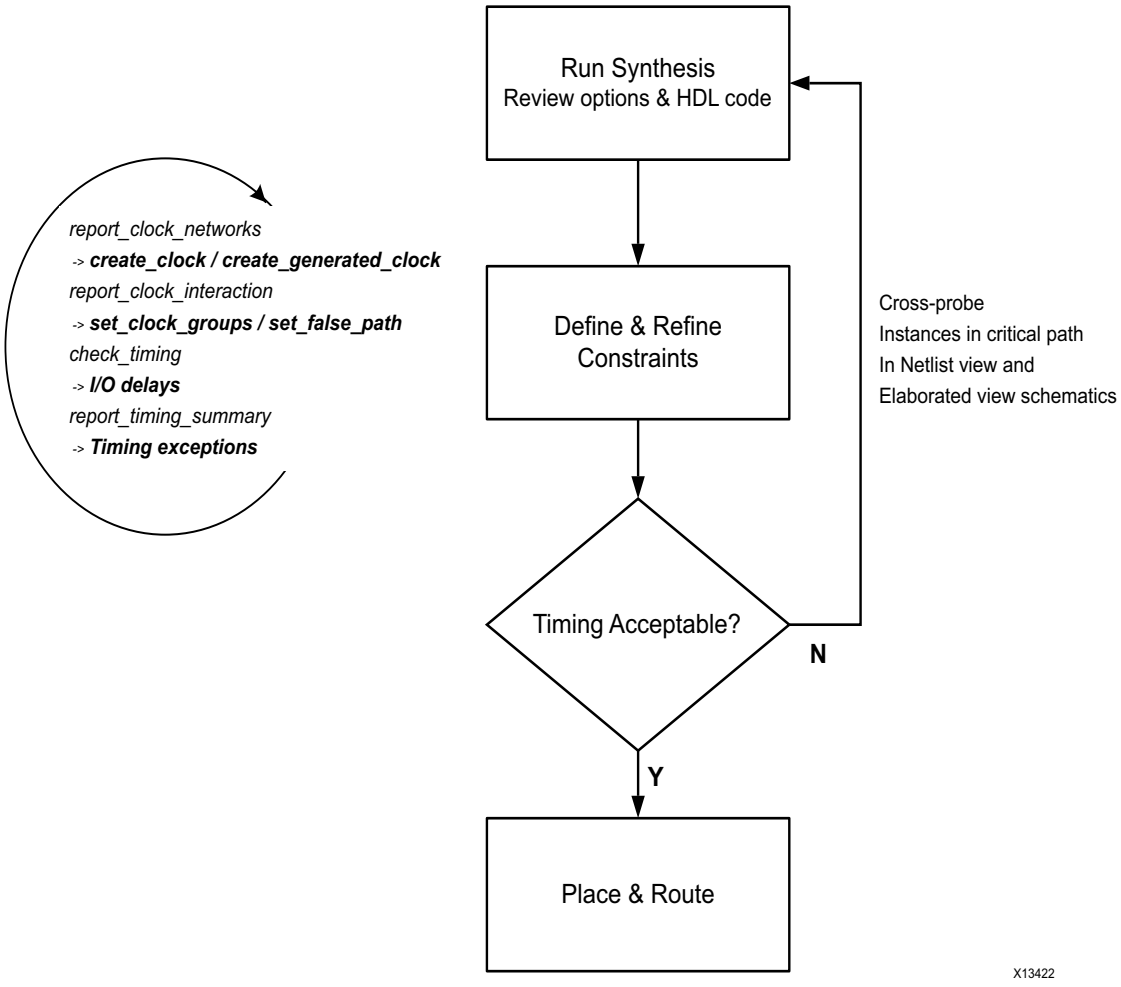


图 1-3：实现快速收敛的设计方法

如能够通过正时序裕度 (positive margin) 或相对较小的负时序裕度 (negative margin) 满足设计目标，那么综合部分可视为完成。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，依照流程进行工作能够提供对负时序裕量 (negative timing slack) 的理解，这有助于您确定用于提升综合后最差负时序裕量 (WNS) 的工作量。您在返回综合阶段改进 HDL 和约束时能够使用该信息。

利用快速验证的优势

本指南还介绍了如下系统架构和微架构各个具体方面的快速验证概念。

- 在系统设计环境下， I/O 带宽进行系统内验证，这一步甚至在实现整个设计之前就要进行。验证 I/O 带宽凸显了在 I/O 最终确定之前可能需要修改系统架构和接口选择。如需了解更多信息，请参阅第 2 章中的“接口带宽验证”。
- 作为设计实现的组成部分，设计基准 (baselining) 用来编写最简单的约束集，从而能够明确内部器件的时序挑战。在进入实现阶段前，基准能够明确是否需要修改 RTL 微架构选择。如需了解更多信息，请参阅第 5 章中的“设计基准 (baseline)”。

使用 Vivado Design Suite

Vivado Design Suite 具有灵活的使用模式，从而可支持各种开发流程和不同的设计类型。如欲了解如何使用 Vivado Design Suite 其中特性的信息，请参阅 《Vivado Design Suite 用户指南：设计流程简介》(UG892) [参照 5] 以及其他 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市场上现有的版本控制系统来管理自身设计源与设计成果。 Vivado Design Suite 能够针对管理设计与 IP 数据提供各种使用模式。如需了解更多解通过版本控制系统使用 Vivado 工具的信息，请参阅 《Vivado Design Suite 用户指南：设计流程简介》(UG892) [参照 5] 中的链接。

升级到新发布的 Vivado Design Suite 版本

新发布的 Vivado Design Suite 版本通常包含对赛灵思 IP 的更新。请仔细考虑您是否希望更新您的 IP，因为更新可能产生设计变更。此外，您在使用由旧版本配置的 IP 开展工作时必须遵循特定的规则。如需了解更多信息，请参阅 《Vivado Design Suite 用户指南：采用 IP 进行设计》(UG896) [参照 9] 中的链接。

访问其他技术文档和培训资料

本指南对 Vivado Design Suite 内的文档信息进行了补充，包括用户指南、参考指南、教程以及 QuickTake 视频。Xilinx Documentation Navigator (DocNav) 提供了访问 Vivado Design Suite 文档和支持资源的渠道，您可以通过筛选或者搜索找到相应的信息。打开 Xilinx Documentation Navigator 的方法：

- 在 Vivado IDE 中，单击“Help > Documentation and Tutorials”。
- 在 Windows 中，单击“Start > All Programs > Xilinx Design Tools > DocNav”。
- 在 Linux 命令提示中输入 `docnav`。

赛灵思设计中心 （Xilinx Design Hubs）提供了根据设计任务和其他话题整理的文档链接，您可以使用链接了解关键概念以及常见问题解答。访问设计中心：

- 在 Xilinx Documentation Navigator 中，单击“Design Hubs View”视图。
- 在赛灵思网站上，查看设计中心页面。

提示： 点击窗口或对话框中的“Quick Help”按钮 ? 就能迅速访问 Vivado IDE 不同部分的相关信息。如需了解有更多关 Tcl 命令的信息，请在 Tcl 控制台中输入命令时加上 `-help` 后缀。

XILINX
ALL PROGRAMMABLE™

第 2 章

单板和器件规划

单板和器件规划简介

正确规划单板上 FPGA 的定向并将信号分配给特定的引脚，这样可以显著改进系统整体性能、功耗和设计周期。可视化 FPGA 器件与印刷电路板 (PCB) 之间的物理和逻辑互动方式，使您可以优化通过器件的数据流。

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。赛灵思强烈建议在考虑 I/O 管脚分配的同时进行板级规划。

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) [参照 4]
- Vivado Design Suite QuickTake 视频教程：I/O 管脚分配简介

PCB 布局建议

单板上 FPGA 器件的布局与其它组件的互动会对 I/O 管脚分配产生巨大影响。

与 PCB 上的物理组件保持一致

首先应确定 FPGA 器件在 PCB 上的定向。还要考虑固定 PCB 组件的位置，以及内部 FPGA 资源。例如，使 FPGA 封装的千兆位收发器接口尽量靠近在 PCB 上与其连接的组件，这样可以缩短 PCB 走线长度，同时减少 PCB 过孔数量。

制作一张包含关键接口的 PCB 草图，通常有助于确定 FPGA 器件在 PCB 上的最佳定向，以及 PCB 组件的布局。完成以后继续规划其余的 FPGA I/O 接口。

像存储器这样的高速接口可以通过非常短的走线直接与 PCB 组件连接。如有可能，这些 PCB 走线经常需要长度匹配，并且不能使用 PCB 过孔。在这种情况下，偏向于选择最接近器件边缘的封装引脚，以保持较短的连接，同时避免布线超出 BGA 引脚的大矩阵。

在该阶段， Vivado IDE 中的“I/O Planning”布局有助于对物理器件尺寸相关的 I/O 连接进行可视化，能够展示顶视图与底视图。

下列数据展示了“I/O Planning”布局。

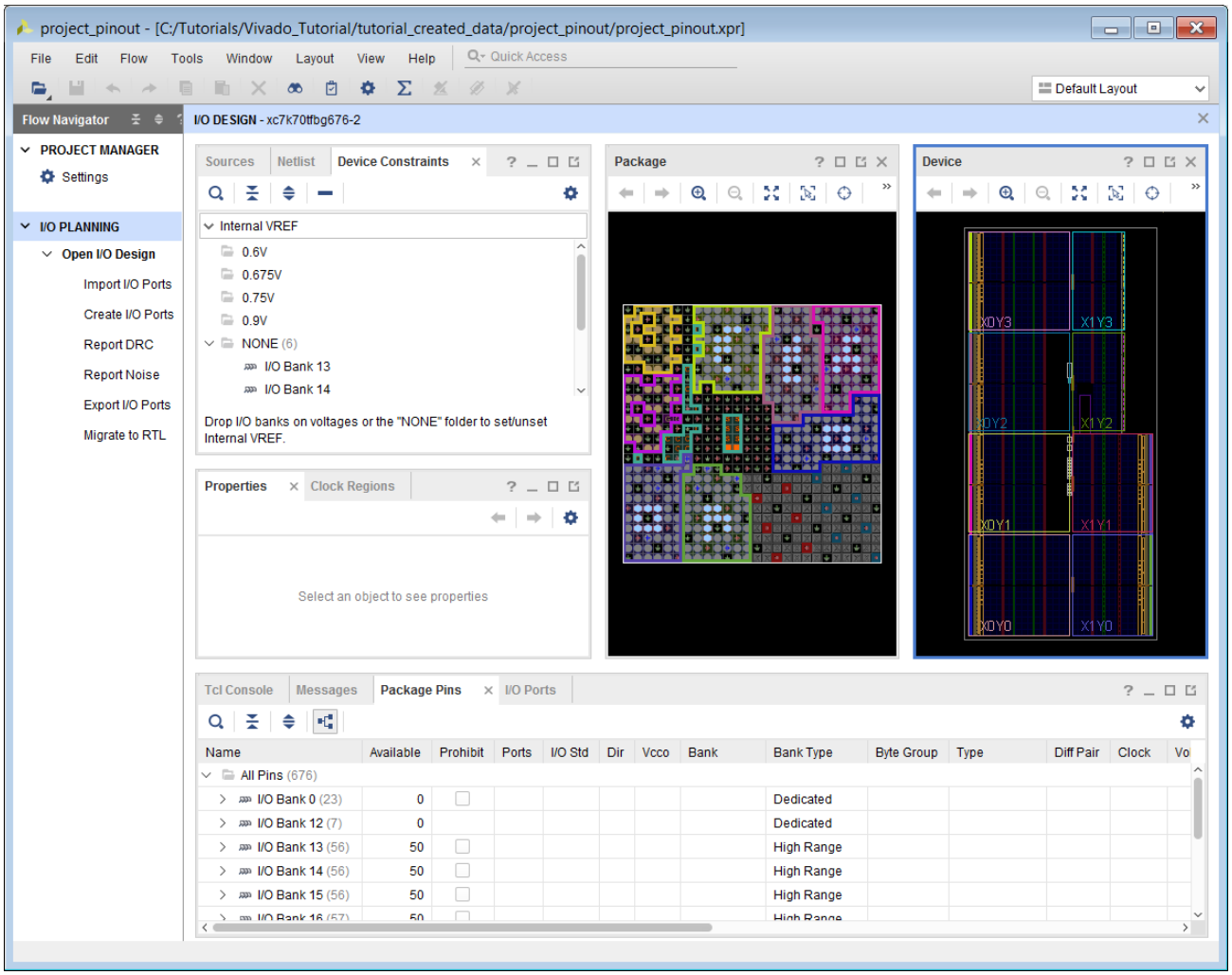


图 2-1：“I/O Planning”布局

配电系统

主板设计师面临着一个特殊的任务，那就是为 FPGA 设计一个配电系统 (PDS)。大多数其它大型密集型集成电路 （如大型微处理器）都要求带有专用的旁路电容器。因为这些器件仅用于执行特定任务，其电源需求固定，而且仅在一定范围内波动。

FPGA 器件不具有这一属性。 FPGA 器件可以在不确定频率和多个时钟域中执行几乎无限多的应用程序。为此，请您务必参考器件 《PCB 设计指南》[参照 36]，全面了解器件 PDS。

PDS 设计期间应考虑的关键因素包括：

- 选择合适的电压，并根据功耗估算，满足噪声和电流要求。如需了解更多信息，请参阅第 5 章中的“功耗分析与优化”。
- 建立 XADC 电源 （Vrefp 和 Vrefn 引脚）。
- 运行 PDN 仿真。《PCB 设计指南》[参照 36] 中建议的去耦电容器数量是以最坏的情况为前提，因为 FPGA 器件可以实现任意功能。运行 PDN 仿真有助于减少确保电源在建议的工作范围内运行所需的去耦电容器数量行。

如需了解更多有关 PDN 仿真的信息，请参阅赛灵思白皮书：《使用 S 参数模型仿真 FPGA 电源完整性》(WP411) [参照 51]。

PCB 设计的具体考虑因素

PCB 设计应考虑到与 FPGA 器件进行最快速度的信号连接。这些高速信号在追踪走线结构、过孔、损耗和串扰时非常敏感。对于多层 PCB，这几个方面会变得尤为突出。为了达到较高的速度，接口会进行信号完整性仿真。采用更先进的 PCB 材料或改变走线几何图形重新设计开发板，这样可以获得所需的性能。

赛灵思建议您在设计 PCB 时遵循下列步骤：

1. 查看以下器件文档：
   - 对应您器件的 《PCB 设计指南》[参照 36]。
   - 对应您器件的 《收发器用户指南》[参照 41] 中的主板设计指导。
2. 查看 IP 产品指南中存储器 IP 和 PCIe® 的设计指导。
3. 采用 Vivado® 工具验证您的 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 缓存信息规格 (IBIS) 模型。
4. 以下列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 仿真检查过冲或下冲问题。
5. 使用 Xilinx Power Estimator (XPE) 为设计的功耗生成早期估算。
6. 遵循对应您器件的原理图检查表完成工作。

时钟资源规划与分配

赛灵思建议您设计时首先选择时钟资源，然后再选择管脚。您的时钟选择不仅可以确定特定的管脚，而且还可以支配逻辑布局，对 SSI 技术的器件尤其如此。正确的时钟选择可以产生非常好的效果。考虑下列事项：

- 约束创建，尤其与时钟规划配合使用的具有高利用率的大型器件。

- 手动布局时钟资源 （设计收敛可能需要）。第 3 章中的“编码指南”详细解释了时钟资源 （如需要手布动局）。
- 针对特定器件的功能或许需要提前开展规划，从而避免问题并有效利用器件特性优势。如需了解有更多关 7 系列特性的信息，请参阅 《7 系列 FPGA 时钟资源用户指南》(UG472) [参照 40] 中的链接与链接。如需了解有更多关 UltraScale™ 器件特性的信息，请参阅 《UltraScale 架构时钟资源用户指南》(UG572) [参照 40] 中的链接。

I/O 管脚分配设计流程

Vivado IDE 支持您在设计中交互式探索、可视化、分配和验证 I/O 端口和时钟逻辑。该环境不仅可确保“生成即保证正确”的 I/O 分配，而且还可对与内部晶片焊盘相关的外部封装引脚提供可视化。

您能够可视化通过器件的数据流，并能够从内外两个方面正确规划 I/O。通过 Vivado IDE 分配和配置 I/O 后会自动创建对实现工具的约束。

如需了解更多有关 Vivado Design Suite I/O 管脚分配和时钟规划性能的信息，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) [参照 4]
- 《Vivado Design Suite 教程：I/O 管脚分配和时钟规划》(UG935) [参照 32]
- Vivado Design Suite QuickTake 视频教程：I/O 管脚分配简介

I/O 管脚分配的相关 Vivado Design Suite 项目种类

您可以采用下列任一种项目开展 I/O 管脚分配：

- I/O 管脚分配项目

  I/O 管脚分配项目为您提供了一个方便的起点，使您能够指定选择 I/O 约束并利用经定义的引脚生成顶层 RTL 文件。

- RTL 项目

  RTL 项目可以开展综合与实现，提供了更加全面的设计规则检查 (DRC)。RTL 项目还有助于生成 IP 核，对存储器接口管脚规划与使用 GT 的核非常重要。

提示： 您可以先由 I/O 管脚分配项目开始工作，并在后期移植至 RTL 项目。

您可以在后综合网表上运行更加全面的 DRC。上述概念同样适用于设计实现与比特流生成。因此，赛灵思建议您使用包含时钟组件和某些基础逻辑的骨架设计实现 DRC。这有助于形成主板引脚定义不会产生后期问题的可信度。

推荐的签发进程为：运行 RTL 项目直至比特流生成环节，实现全部 DRC。但是，并非所有设计进程都有足够时间开展这一进程。通常必须在具备可综合 RTL 之前定义好 I/O 配置。虽然 Vivado 工具支持 Pre-RTL 的 I/O 管脚分配，但是执行的 DRC 检查级别还是相当基础。或者，通过采用 I/O 标准和引脚分配的虚拟顶层设计可以帮助执行与 I/O bank 分配规则相关的 DRC。

Pre-RTL I/O 管脚分配

如果设计周期强制要求在具备综合网表之前定义 I/O 配置，则请务必确保符合所有相关的规则。 Vivado 工具提供一个管脚分配工程环境，允许使用 CSV 或 XDC 格式文件导入 I/O 定义。可以使用定义的端口方向创建一个伪 RTL。提供端口方向可以让同步开关噪声 （SSN）分析更加准确，因为输入和输出信号对 SSN 有不同的影响。

还可以交互式创建和配置 I/O 端口。具有基本 I/O bank DRC 检查规则。

请参阅 《PCB 设计用户指南》[参照 36]，确保器件 I/O 配置正确。如需了解更多信息，请参阅 《Vivado Design Suite 用户指南：I/O 管脚分配和时钟规划》(UG899) [参照 4] 中的链接。

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。在网表中创建时钟逻辑路径旨在实现约束分配。 I/O 和时钟逻辑 DRC 更加全面。

请参阅 《PCB 设计用户指南》[参照 36]，确保器件 I/O 配置正确。如需了解更多信息，请参阅 《Vivado Design Suite 用户指南：I/O 管脚分配和时钟规划》(UG899) [参照 4] 中的链接。

定义替代器件

在初始规划阶段，通常很难预测给定设计的最终器件尺寸。在设计周期中添加或删除逻辑将导致需要更改器件尺寸。

Vivado 工具可帮助您定义替代器件，以确保定义的 I/O 引脚配置与所有选定的器件兼容，但前提是采用相同封装。

重要提示： 器件必须位于同一封装中。

若要低风险地移植设计，请在设计进程开始时仔细规划以下工程：器件选择、管脚选择和设计标准。在移植至同一封装中的较大或较小型器件时，请考虑以下方面：管脚、时钟和资源管理。如需了解更多信息，请参阅 《Vivado Design Suite 用户指南：I/O 管脚分配和时钟规划》(UG899) [参照 4] 中的链接。

引脚分配

合理的管脚选择会得到合理的设计逻辑布局、更短的布线，并减少功耗，提升性能。合理的管脚选择对大型的 FPGA 设备来说至关重要，因为分散的管脚会导致相关信号之间的距离更远。如需了解更多信息，请参阅 《Vivado Design Suite 用户指南：I/O 管脚分配和时钟规划》(UG899) [参照 4] 中的链接。

使用赛灵思工具选择管脚

赛灵思工具可以辅助交互式设计规划和引脚选择。这些工具仅与您所提供的信息一样有效。诸如 Vivado 设计分析工具等工具可以辅助管脚。这些工具能够以图形的方式显示 I/O 布局，并显示时钟和 I/O 组件之间的关系，另外还具有 DRC 功能，用以分析引脚选择。

当出现某个设计版本时，其会快速创建一个顶层布局规划图，用以分析通过器件的数据流。如需了解更多信息，请参阅 《Vivado Design Suite 用户指南：设计分析和收敛技术》(UG906) [参照 21]。

所需的信息

为了使工具能够有效地工作，您必须尽可能多地提供有关 I/O 特性和拓扑结构的信息。您必须规定电气参数，包括 I/O 标准、驱动、斜率和 I/O 方向。

您还必须考虑包括时钟拓扑结构和时序约束在内的所有其它相关信息。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。请参阅第 3 章中的“编码指南”。

针对具有 I/O 要求的 IP，例如收发器、 PCIe，以及存储器接口，您必须在完成 I/O 引脚分配前配置 IP，如“管脚选择”所述。如需了解更多指定 I/O 规定电气参数的信息，请参阅 《Vivado Design Suite 用户指南：I/O 管脚分配和时钟规划》(UG899)[参照 4] 中的链接。

管脚选择

对于如下所述的某些特定信号，赛灵思建议谨慎进行管脚选择。

接口数据、地址和控制引脚

将相同的接口数据、地址和控制引脚集合在同一个 bank 上。如果无法将这些组件集合在同一个 bank 上，则请将其分配在邻近的 bank 上。堆叠硅片互联 (SSI) 技术器件而言，相邻 bank 必须处于同一超级逻辑区域 (SLR) 中。

接口控制信号

将以下接口控制信号放置在所控制的数据总线中间 （时钟、启用、复位和选通）。

极高的扇出、涉及范围大的控制信号

将极高的扇出、涉及范围大的控制信号放置在器件的中心。

对于 SSI 技术器件，请将 SLR 中的信号放在所驱动的 SLR 组件的中间。

配置引脚

要想设计一个高效的系统，则必须选择充分满足系统要求的 FPGA 配置模式。需要考虑的因素包括：

- 使用专用或两用配置引脚。

  每个配置模式专用于特定的 FPGA 引脚，而且只有在配置过程中才可临时使用其它多功能引脚。配置完成后，这些多功能引脚被释放，可用作通用引脚。

- 使用配置模式对一些 FPGA I/O bank 布局电压限制。
- 为不同的配置引脚选择合适的终端。
- 对配置引脚，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，但请在单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。虽然选项灵活多样，但是每个系统一般都有一个最佳的解决方案。在选择最佳配置选项时，请考虑以下方面：

- 建立
- 速度
- 成本
- 复杂性

请参阅“配置”。如需了解更多有关 FPGA 配置选项的信息，请参阅 《Vivado Design Suite 用户指南：编程和调试》 (UG908) [参照 24]。

存储器接口

使用赛灵思存储器 IP 时需要更多的 I/O 管脚分配步骤。自定义 IP 之后，您可采用 Vivado IDE 中的细化或综合设计分配顶层 IP 端口到物理封装引脚。同每一个存储器 IP 关联的所有端口都被纳入一个 I/O 端口接口内，以便识别和分配。一个存储器 bank/字节规划器会帮助您将存储器 I/O 引脚组分配到物理器件引脚上的字节通道中。如需了解更多信息，请参阅 《Vivado Design Suite 用户指南：I/O 管脚分配和时钟规划》 (UG899) [参照 4] 中的链接。

分配存储器接口时请小心处理，应试图最大限度地减少拥塞，这对采用中央 I/O 列的器件尤为重要。将存储器接口密集布置会在器件中产生布线瓶颈。《赛灵思 Zynq-7000 SoC 和 7 系列器件存储器接口用户指南》 (UG586) [参照 48] 和

(PG150) [参照 49] 包含设计和管脚指南。请确保采纳指南中建议的走线长度，核实所使用的终端是否准确无误，并在存储器 IP I/O 分配后运行 DRC 来验证管脚。

千兆位收发器 (GT)

千兆位收发器 (GT) 具有特定的管脚要求，请您务必考虑下列事项：

- 共享参考时钟
- 四通道中共享 PLL
- PCIe 等硬块的布局及其与收发器的距离
- SSI 技术器件中 SLR 边界的交汇

赛灵思建议您使用 GT 向导生成核。或者，您可以为协议使用赛灵思 IP 核有关管脚建议，请参阅相关产品指南。

在时钟资源均衡方面，由 Vivado 布局器尝试约束 GT 输出时钟 （TXOUTCLK 或 RXOUTCLK）控制的负载，旁边的 GT 也会取用这些时钟。堆叠硅片互联 (SSI) 技术器件，如果 GT 所处的时钟域临近另一个 SLR，则进出 SLL 的信号所需布线资源会与 GT 输出时钟负载所需资源产生竞争。因此，位于临近 SLR 交错的时钟域内的 GT 可能会减少那些时钟域内来自和通向 SLL 交错的可用布线连接数。

高速 I/O

HP（高性能）和 HR（大范围） bank 在收发信号的速度上存在差异。根据所需的 I/O 速度，在 HP 或 HR bank 间做出选择。

内部参考电压和 DCI 级联约束

根据 DCI 级联 (DCI Cascade) 和内部参考电压 (VREF) 的设置，您可以释放用于常规 I/O 的引脚。这些设置还可以确保运行相关的 DRC 规则检查，以验证约束的合法性。如需了解更多信息，请参阅 SelectIO 资源用户指南 [参照 39]。

接口带宽验证

创建小型连接设计，用以验证 FPGA 上的每个接口。这些小型设计只能运行特定硬件接口，有助于实现下列目的：

- 管脚、时钟、时钟与时序的完整 DRC 检查
- 主板返回后的硬件测试设计
- 通过 Vivado 工具开展迅速设计实现提供了快速调试接口的方式

有多种选项能够帮助生成这些接口的测试数据。对于某些接口 IP 核， Vivado 工具能够提供测试设计：

- 针对 SerDes 的 IBERT
- IP 核内的示例设计

提示： 如不存在测试设计，请考虑使用 AXI 流量生成器

您或许需要针对生产环境中的系统级测试创建一个单独的设计。通常，这属于包含经测试接口，也可以包含处理器的单一设计。您可以利用小型连接设计带来的设计复用优势来构建该设计。虽然在流程早期并不需要开展该设计，但它能够提供更好的 DRC 检查和早期软件开发，您还可以通过 Vivado IP 集成器迅速创建该设计。

采用 SSI 器件进行设计

SSI 管脚的考虑因素

在为特定 SLR 中的组件规划管脚时，请将引脚放置在同一个 SLR 中。例如，将器件的 DNA 信息作为外部接口的一部分时，请将该接口的引脚放置在 DNA_PORT 所在的主 SLR 中。其它考虑因素包括如下：

- 把特定接口的全部引脚分配到相同 SLR 中。
- 对用于驱动多个 SLR 中的组件的信号，应将这些信号放置在中间的 SLR 中。
- 跨各 SLR 均衡分配 CCIO 或 CMT 组件。
- 减少 SLR 交错。

超级逻辑区域 (SLR)

“超级逻辑区域 (SLR)”是 SSI 技术器件中的单个 FPGA 晶片 slice。每个 SLR 都包含主 FPGA 电路，例如 slice、RAM、DSP slice 和 GT。

多个 SLR 组件纵向堆叠，并通过一个中介层连接，构成一个 SSI 技术器件。底端 SLR 为 SLR0，后续 SLR 组件随着纵向添加而增量命名。例如，XC7V2000T 器件包含四个 SLR 组件。底端 SLR 为 SLR0，SLR0 正上方的 SLR 是 SLR1，SLR1 正上方的 SLR 是 SLR2，顶端 SLR 是 SLR3。

注释： 赛灵思工具可以在图形用户界面 (GUI) 和报告中清楚地识别 SLR 组件。

SLR 术语

了解目标器件的 SLR 术语对以下方面非常重要：

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

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

  - `llength [get_slrs]` 可获得器件中 SLR 的数量
  - `get_slrs -of_objects [get_cells my_cell]` 用以查找 `my_cell` 所在的 SLR

主超级逻辑区域

每个 SSI 技术器件都有一个主 SLR，主 SLR 包含主配置逻辑，可初始化器件及其它所有 LR 元件的配置。主 SLR 内含用于配置的电路 DNA_PORT 和 EFUSE_USER。使用这些组件时，布局布线工具可为合适的 SLR 设定相关引脚与逻辑。总之，无需进行额外干预。

提示： 在 Vivado Design Suite 中如想查询哪一个 SLR 是主 SLR，您可输入下 Tcl 命令：`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，为下列功能在 SLR 组件间布线：

- 配置
- 全局时钟
- 一般互联

超长线路 (SLL) 布线

超长线路 (SLL) 布线将器件内一个 SLR 与另一个 SLR 信号联通。

提示： 为确立两个 SLR 之间的 SLL 数量以及可用资源数量，请在一个包含最低限度逻辑的小型综合后设计中运行 Tcl 命令 `report_utilization -slr`。

传输限制

提示： 对跨越多 SLR 的高速传输而言，要考虑寄存穿过 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 地址级联
- 以及其他专用连接，如 DCI 级联和块 RAM 级联。

这些工具通常会考虑这种传输限制。为确保设计布线正确，并且符合您的设计目标，当您

- 建立一个非常长的 DSP 级联并手动将这种逻辑放置在 SLR 边界附近时，您也必须将该限制纳入考虑范围内，
- 为设计设定管脚时也应如此。

SLR 利用率考虑因素

Vivado 设计实现工具采用了一种特殊算法将逻辑划分为多个 SLR。针对挑战性较高的设计，您可以利用下列指南来提升面向 SSI 技术器件设计的时序收敛。

为提升时序收敛并编译定时，您可以使用 Pblock 将逻辑分配至每个 SLR，并验证个体 SLR 对所有的结构资源类型不存在利用率过度的现象。例如，一个块 RAM 利用率为 70% 的设计如果块 RAM 资源在所有 SLR 上不平衡，其中一个 SLR 使用了 超过 85% 的块 RAM，这或许会造成时序收敛问题。

下面的 vu160 利用率示例报告显示，整体块 RAM 利用率为 56%，而 SLR0 中的块 RAM 利用率为 89% （1008 可用资源中占用了 897）。相对于所有 SLR 中平衡块 RAM 利用率，设计 SLR0 中实现时序收敛或许更加困难。

```
3. BLOCKRAM
-----------
+-------------------+------+-------+-----------+-------+
|     Site Type     | Used | Fixed | Available | Util% |
+-------------------+------+-------+-----------+-------+
| Block RAM Tile    | 1843 |     0 |      3276 | 56.26 |
|   RAMB36/FIFO*    | 1820 |     1 |      3276 | 55.56 |
|     FIFO36E2 only |   78 |       |           |       |
|     RAMB36E2 only | 1742 |       |           |       |
|   RAMB18          |   46 |     0 |      6552 |  0.70 |
|     RAMB18E2 only |   46 |       |           |       |
+-------------------+------+-------+-----------+-------+
```

```
14. SLR CLB Logic and Dedicated Block Utilization
-------------------------------------------------
+-----------+--------+---------+------------+-------------+---------------+-----------+-------+------+
| SLR Index |  CLBs  | (%)CLBs | Total LUTs | Memory LUTs | (%)Total LUTs | Registers | BRAMs | DSPs |
+-----------+--------+---------+------------+-------------+---------------+-----------+-------+------+
| SLR2      |  40109 |   89.61 |     167520 |         156 |         46.78 |    327600 |   512 |    0 |
| SLR1      |  42649 |   95.28 |     205484 |        2297 |         57.38 |    355918 |   434 |    0 |
| SLR0      |  35379 |   97.62 |     163188 |          24 |         56.29 |    313392 |   897 |    0 |
+-----------+--------+---------+------------+-------------+---------------+-----------+-------+------+
| Total     | 118137 |         |     536192 |        2477 |               |    996910 |  1843 |    0 |
+-----------+--------+---------+------------+-------------+---------------+-----------+-------+------+
```

赛灵思建议将块 RAM 和 DSP 组分配至 SLR Pblock，从而最大限度减少 SLR 对共享信号的交汇。例如，一个地址总线向一组块 RAM 扇出，块 RAM 在多个 SLR 上展开，这种情况下实现时序收敛就更加困难，这是因为 SLR 引发了时序关键信号的额外延迟。

器件资源位置或用户 I/O 选择将 IP 与 SLR 绑定，例如 GT、ILKN、PCIe，以及 CMAC 专用块或存储器接口控制器。赛灵思建议：

- 应特别注意专用块位置与管脚选择，从而避免数据流与 SLR 边界多次交汇。
- 确保紧密互联的模块与 IP 处于同一 SLR 内。如无法实现，您可以通过添加流水线寄存器来为布局器提供更大灵活性，从而在 SLR 与逻辑组交汇的情况下找出优秀的解决方案。
- 确保关键逻辑处于同一 SLR 内。通过确保主模块在接口端妥善实现流水线，布局器更有可能找到具有触发器至触发器 SLR 交汇的 SLR 分区。

下图中约束至 SLR0 的一个存储器接口需要驱动 SLR1 中的用户逻辑。与存储器 IP 后端连接的一个 AXI4-Lite 从，和存储器 IP 与 AXI4-Lite 接口间妥善定义的边界，共同提供了自 SLR0 到 SLR1 的转换。

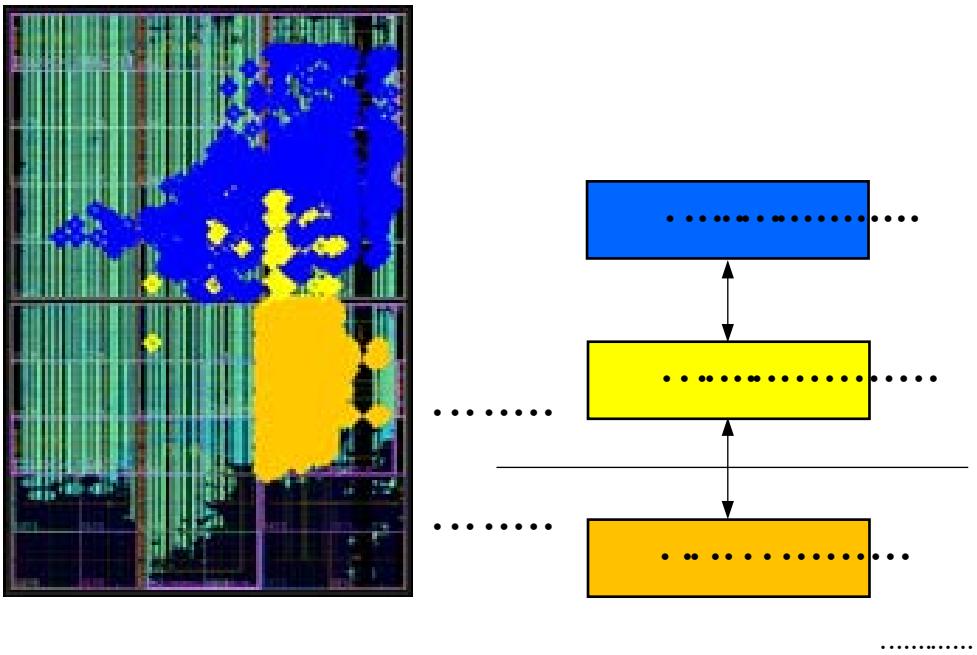


图 2-2：SLR0 中的存储器接口驱动 SLR1 中的用户逻辑

大宽度总线的 SLR 交汇

当数据流程要求采用大宽度总线跨越 SLR 时，请采用流水线策略来提升时序收敛，并缓解长距离资源的布线拥塞。针对工作频率高于 250 MHz 的大宽度总线，赛灵思建议采用至少 3 个流水线阶段跨越一个 SLR，分别在 SLR 顶端、底端和中端各一个。对性能特别高的总线，或同时穿越水平和垂直距离时，或许需要采用额外的流水线阶段。

以下数据展示了 vu190-2 器件交汇的最坏情况。本示例从 SLR0 左下方的一个 Interlaken 专用块开始，到分配至 SLR2 右上方的包监测块为止。由于没有来自和到达包监测模块的数据总线流水线寄存器，该设计离 300 MHz 的时序要求差距较大。

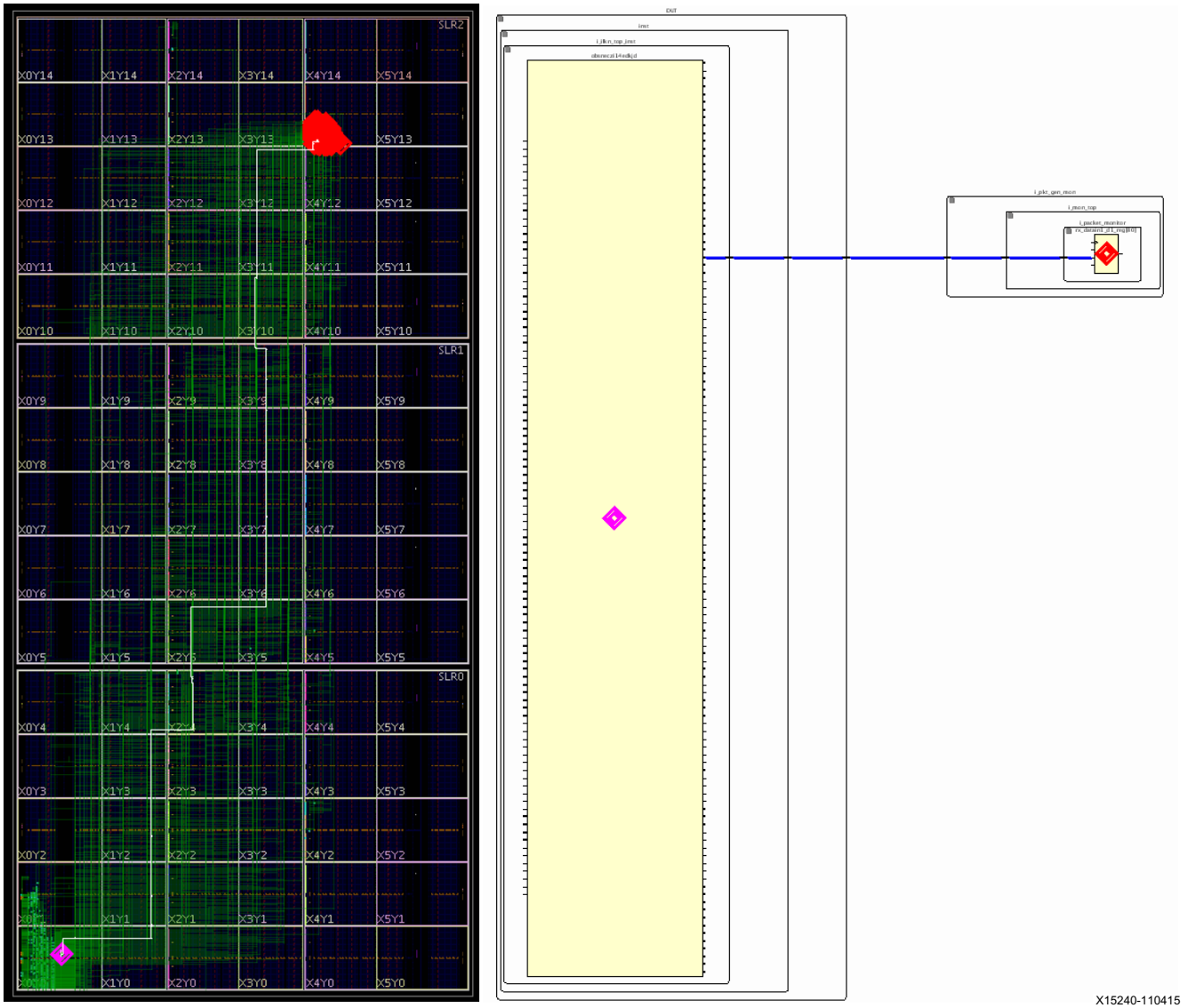


图 2-3：不采用流水线触发器的数据路径 SLR 交汇

然而，添加 7 流水线阶段助力自 SLR0 至 SLR2 的穿越使设计能够满足时序。这样还减少了纵向与横向长距离布线资源的使用，如下图所示。

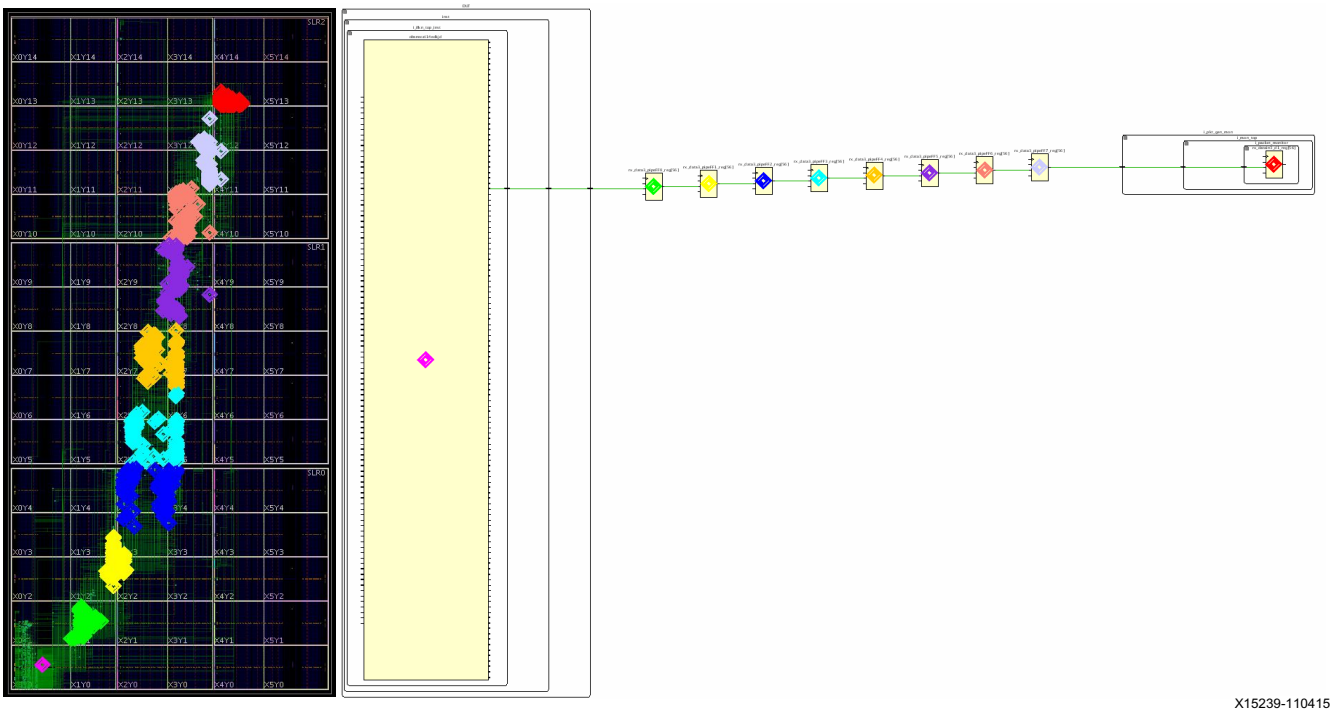


图 2-4：添加流水线触发器的数据路径 SLR 交汇

FPGA 电源因素与系统关联性

规划 PCB 板时，请务必考虑功耗因素：

- FPGA 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确保在建议工作电压与温度条件下工作。为确保器件符合上述限制，或需要开展功耗估算和热建模。

因此，您必须了解 FPGA 器件的用功耗和散热要求，并在单板设计过程中考虑到这些要求。

FPGA 器件的供电源路径

布局 FPGA 器件要求使用多个电源供电。其中部分电源必须按特定顺序排列。可考虑用电源监测或顺序电路为 FPGA 器件和 GT 以及开发板上的其它有源组件提供正确的上电顺序。在较复杂的环境下，可使用微控制器或系统以及 SMBUS 或 PMBUS 等电源管理总线来控制电源和复位进程。关于启动/关闭顺序的详细信息，请参阅器件数据手册。

不同 FPGA 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下工作，以提高性能或增加信号强度，同时避免产生噪声和寄生效应。

功耗模式

从上电到断电， FPGA 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 FPGA 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 FPGA 器件的结构、电源上升到额定电压的能力，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 FPGA 器件架构中，不用担心峰值电流的问题，因为它遵循了适用的上电顺序指南。

启动功耗

启动功耗是指在器件初始化和配置期间所需的功耗。该功耗通常只持续非常短的一段时间，所以无需考虑热耗散。不过，仍然要满足电流要求。在大多数情况下，一个正在运行的设计中的有功电流会更高，所以不需任何改动。但是，对于功耗更低的设计，其有功电流可能会较低，或许有必要在这时采用更高的电流要求。 Xilinx Power Estimator (XPE) 可用于解读此要求。当工艺参数被设为“Maximum”时，对每个电压轨的要求会被定为工作电流或启动电流，按两者中较高的为准。如启动电流更高， XPE 会以蓝色显示电流值。

待机功耗

待机功耗 （又称“设计静态功耗”）是器件按设计配置后未对其施加任何外部活动或者未生成任何内部活动时提供的功耗。

待机功耗是设计运行时电源应提供的最小连续功耗。

有功功耗

有功功耗 （又称“设计动态功耗”）是器件运行应用程序时所需功耗。有功功耗包括待机功耗 （全部静态功耗）以及因设计活动 （设计动态功耗）生成的功耗。有功功耗是瞬时发生的，且根据输入数据模式以及设计内部活动的不同每个时钟周期变化一次。

影响功耗的环境因素

除设计自身之外，环境因素也会影响功耗。这些因素会影响器件的电压和结温，进而影响功率损耗。详情，请参阅《Vivado Design Suite 用户指南：功耗分析和优化》(UG907) [参照 22] 中的链接。

功耗模型精度

嵌入在工具中的特性描述数据的准确度会随着时间发生变化以准确反映器件的可用性以及生产工艺的成熟度。详情，请参阅 《Vivado Design Suite 用户指南：功耗分析和优化》(UG907) [参照 22] 中的链接。

FPGA 器件功耗及整体系统设计进程

从项目构思到完成，设计进程中各方面因素都会影响功耗。详情，请参阅 《Vivado Design Suite 用户指南：功耗分析和优化》(UG907) [参照 22] 中的链接。

利用 Xilinx Power Estimator(XPE) 进行最差情况功耗分析

赛灵思建议针对最坏情况的功耗设计主板。详情，请参阅《Vivado Design Suite 用户指南：功耗分析和优化》(UG907) [参照 22] 中的链接。

配置

配置指的是将特定应用数据加载到 FPGA 器件的内部存储器的进程。

赛灵思 FPGA 配置数据储存在 CMOS 配置锁存 (CCL) 中，因此配置数据很不稳定，且在每次 FPGA 器件上电后都必须重新加载。

赛灵思 FPGA 器件可通过来自外部非易失性存储器件的配置引脚自行加载配置数据。而且还可以用外部智能源配置器件，外部智能源包括：

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

板级规划应首先考虑配置方面，以简化配置和调试工作。

每个器件系列都提供有配置用户指南 [参照 38]，是用户了解各种所支持的配置模式和它们在引脚数、性能与成本上的平衡等详细信息的主要资源。

单板设计技巧

在板设计过程中，很重要的一点在于考虑好哪些接口和引脚将用于在配置外协助调试功能。例如，赛灵思建议您确保 JTAG 接口访问通畅，即便该接口不是主要配置模式时也应如此。JTAG 接口让您能够检查器件 ID 和器件 DNA 信息，您也可使用该接口在原型设计阶段实现间接闪存编程解决方案。

此外，如 INIT_B 和 DONE 等信号对 FPGA 配置调试至关重要。 INIT_B 信号有多个功能。它用于提示上电初始化的完成，并能在遇到 CRC 错误时指出。赛灵思建议您采用 LED 驱动器和上拉电阻来将 INIT_B 与 DONE 信号连接至 LED。请参阅配置用户指南 [参照 38] 获得建议的上拉电阻值。

原理图检查表 [参照 52] 包含以下推荐及其他关键建议。通过这些表来识别和检查受推荐的板级引脚连接。

第 3 章

设计创建

设计创建简介

在规划器件 I/O 管脚，规划如何布局 PCB 以及决定 Vivado® Design Suite 的使用模型后，就可以开始创建设计。设计创建包括：

- 规划设计的层级
- 确定设计中需要使用和自定义的 IP 核
- 对于无法找到合适 IP 的互联逻辑和功能，为其创建定制 RTL；
- 创建时序约束和物理约束
- 指定综合与实现阶段所使用的额外约束、属性及其它元件

创建设计时，主要的考虑因素包括：

- 实现所需的功能
- 在理想的频率上工作
- 按预期可靠地工作
- 符合芯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。在过程的早期花费时间仔细规划您的设计有助于确保您实现您的设计目标，并尽量减少实验室中的调试时间。

定义理想的设计层级

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将含有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重用。

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

靠近顶层添加 I/O 组件

尽可能地靠近顶层添加 I/O 组件，以实现设计可读性。引用组件时，提供要完成功能的描述。然后，使用综合工具解释 HDL 代码以确定使用哪些硬件组件来执行该功能。可引用的组件有简单的单端 I/O （IBUF、 OBUF、 OBUFT 与 IOBUF）以及 I/O 中的单数据速率寄存器。

需要例化的 I/O 组件也应该于靠近顶层处进行例化，如差分 I/O （IBUFDS、 OBUFDS）和双数据速率寄存器 （IDDR、 ODDR、 ISERDES、 OSERDES）。例化组件时，将组件的示例添加到 HDL 文件。例化可以让您完全控制组件的使用方式。因此，你就知道逻辑是如何使用的。

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，从而提高资源利用率，提升性能，并降低功耗。

除了创建有时钟的模块之外，时钟路径只能驱动进入模块。任何自上而下，然后又自下而上贯穿的路径会在 VHDL 仿真中造成 delta cycle 问题，也就是说调试工作既艰难又费时。

在逻辑边界上寄存数据路径

层级边界输出加寄存器是为了将关键路径限定在单个模块或边界之内。也可以在层级边界考虑输入加寄存器。通常而言，分析并修复位于一个模块中的时序路径比跨越多个模块的路径要容易得多。任何在层级边界上的路径为寄存器化，都应在重建层级或扁平化层级的条件下完成综合，以实现跨层级优化。在逻辑边界上寄存数据路径有助于在整个设计流程中保持 （用于调试的）可跟踪性，因为这样可以尽量避免跨层级优化，逻辑也不必跨模块移动。

妥善考虑布局规划事项

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

- 在使用 SSI 器件时，要为特定 SLR 划分逻辑。
- 在使用标准流程无法满足时序要求时，关闭设计的时序；

如果有单元未限定在一个层级上，所有对象必须分别纳入布局规划约束中。如果这些对象的名称在综合后发生改变，必须更新约束。理想的布局规划应限定在一个层级上，因为这样做只需要一行约束。

不是随时都需要布局规划。必要时才进行布局规划。

如需了解更多布局规划的信息，请参阅 《Vivado Design Suite 用户指南：设计分析和收敛技术》(UG906) [参照 21] 中的链接。

建议：虽然 Vivado 工具允许跨层级布局规划，但维护工作也由此增加。应尽量避免跨层级布局规划。

针对功能和时序调试优化层级

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。这允许相对容易地探测和修改相关信号，因为当包含在单个层次结构中时，通过综合优化的信号名称更容易跟踪。

在模块级应用属性

在模块级应用属性可让代码更加简洁和更具可扩展性。不应在信号级应用属性，而应在模块级应用属性，然后把属性传输给这个区域内调用的所有信号。在模块级应用属性还能覆盖全局综合选项。因此，为在 RTL 中应用模块级约束，有时增加一个层级会比较好。

注意！某些属性 （例如 DONT_TOUCH）不会从一个模块传输到该模块内的所有信号。

针对高级设计技术优化层级

自下而上综合、部分重配置和无关联 (OOC) 设计等高级设计技术要求在层级上进行规划。设计必须根据使用的设计技术选择合适的层级。本文不对这些技术做详细介绍。如需了解更多信息，请参阅 《Vivado Design Suite 用户指南：层级设计》(UG905) [参照 20] 中的链接。

高速 DSP 设计的预先层级规划实例

下面这个实例并非适用于所有设计，仅用于说明层级所起的作用。 DSP 设计一般允许增加设计时延。这样可以在设计中添加寄存器，实现性能优化。此外还可以使用寄存器实现灵活布局。这非常重要，因为在高速设计中，用户无法在一个时钟周期内遍历整个晶片。添加寄存器可以让难以到达的区域也能得到利用。下图显示了如何用有效的层级规划加快时序收敛。

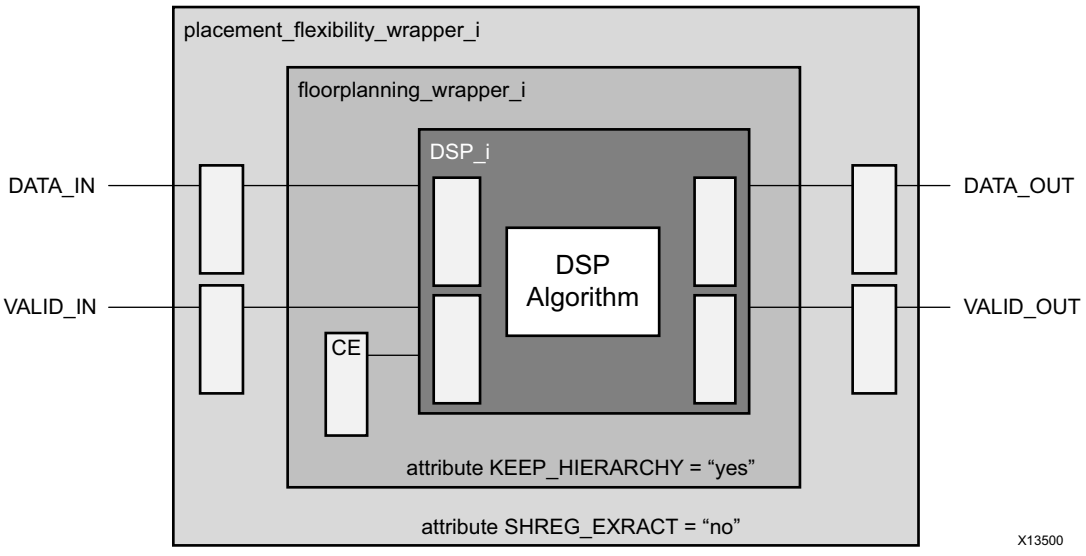


图 3-1：有效的层级规划实例

本设计部分分三个层级：

- `DSP_i`

  在 DSP_i 算法块中将输入和输出同时寄存。由于 FPGA 器件拥有丰富的寄存器，可以使用这种方法来提升时序预算。

- `floorplanning_wrapper_i`

  `Floorplanning_wrapper_i` 中存在 CE 信号。CE 信号一般是重负载信号，会带来时序问题，应在布局规划时加以考虑。需要时，随后可以通过创建布局规划封装程序的方法手动对该模块进行布局规划。

  此外，在模块级已添加 KEEP_HIERARCHY，这样无论是否有其它全局综合选项，仍可确保层级得到保存，用于布局规划。

- `placement_flexibility_wrapper_i`

  `Placement_flexibility_wrapper_i` 用于寄存 DATA_IN、VALID_IN、DATA_OUT 和 VALID_OUT 信号。因为这些信号本来不是布局规划的组成部分，因此它们位于 `floorplanning_wrapper_i` 之外。如果它们在布局规划之内，它们就无法满足灵活布局的要求。

  另外，只要 DATA_IN 和 VALID_IN 或 DATA_OUT 和 VALID_OUT 是成对处理，后续还可以添加更多的寄存器。在添加更多寄存器的时候，综合工具会调用 SRL，将所有寄存器强制集中到一个组件中，这对灵活布局没有帮助。为防止出现这种情况，可以在模块级添加 SHREG_EXTRACT 并将其设置为 NO。

RTL 编码指南

您可以创建定制 RTL 以实现胶合逻辑功能以及没有合适 IP 的功能。为了获得最佳结果，请遵循本节中的编码指南。更详细指导，请参阅 《Vivado Design Suite 用户指南：综合》(UG901) [参照 16] 中的链接。

使用 Vivado Design Suite HDL 模板

在创建 RTL 或实例化赛灵思原语时应使用 Vivado Design Suite 语言模板。这些语言模板包括建议的编码结构，用于正确调用赛灵思器件架构。使用语言模板可以简化设计过程，并改进结果。要从 Vivado IDE 打开语言模板，请在“Flow Navigator”中选择“Language Templates”选项，然后选择所需的模板。

控制信号与控制集

控制集指用于驱动任何给定 SRL、 LUTRAM 或寄存器的控制信号组 （置位/复位、时钟使能和时钟）。对任意控制信号的独特组合，都能构成唯一的控制集。其后的原因是一个重要的概念，即在 7 系列 slice 中的寄存器都共享相同的控制信号，因此只有使用相同控制集的寄存器才能打包到同一个 slice 中。例如，如果具有给定控制器集的寄存器仅具有一个寄存器作为加载，则被占据的片中的其他七个寄存器将不可用。

同时拥有多个唯一控制集的设计会造成大量资源浪费和布局选项数量减少，导致功耗上升，性能下降。从布局的角度而言，拥有较少数量控制集的设计能提供更多选项和更高灵活性，一般也能产生更加理想的结果。

在 UltraScale™ 器件中，在 CLB 内部能灵活地实现控制集的映射。未驱动的复位不形成控制器集的一部分，因为是在片内局部生成的。然而，限制唯一控制集是为一组逻辑布局提供最大灵活性的一个好方法。

复位

复位是需要在设计中考虑和设限的更常见也更重要的控制信号之一。复位会给用户设计的性能、占位面积和功耗产生显著影响。

经引用得到的同步代码会产生下列资源：

- LUT
- 寄存器
- 移位寄存器 LUT(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上述组件的选择，导致给定设计中资源利用率下降。如果在阵列上误置复位，会产生截然不同的结果，可能是调用出一个块 RAM，也可能是调用出数千个寄存器。

在乘法器输入或输出处描述异步复位，可能造成寄存器布置在 slice 上而非 DSP 块上。在此类情况以及在其它情况下，会给资源数量造成影响。总体功耗和性能也会受到显著影响。在大多数情况下，这会影响性能。它还对器件利用率和功耗有负面影响。

使用复位的时间和位置

赛灵思器件提供专用的全局置位/复位信号 (GSR)。在器件配置结束时，该信号将设置硬件中所有顺序单元的初始值。

如果未指定初始状态，则为顺序原语分配默认值。在大多数情况下，默认值为零。 FDSE 和 FDPE 原语是例外，默认为逻辑 1。每个寄存器在配置结束时将处于已知状态。因此没有必要单独为初始化加电器件编写全局复位代码。

赛灵思强烈建议用户仔细判断设计何时需要复位，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。限制复位使用的方法如下：

- 限制复位网络的总体扇出。
- 减少复位路由所需的互联数量。
- 简化复位路径的时序。
- 从而在许多情况下能够从整体上提升性能、缩小占位面积和降低功耗。

建议：评估每一个同步块，尝试判断是否需要复位以确保正常运行。在不确定需要的情况下，切勿将复位编码为默认项。

使用功能仿真应能够轻松地判断是否需要复位。

对没有编码复位功能的逻辑，在选择用于映射逻辑的 FPGA 资源方面具有更高的灵活性。

综合工具随后能为该代码选择最优资源，为了实现可能的最佳结果，应考虑到以下因素，如：

- 要求的功能
- 性能要求
- 可用器件资源
- 功耗

同步复位与异步复位

如果需要复位，赛灵思建议代码同步复位。与异步复位相比，同步复位拥有众多优势。

- 同步复位可以直接映射到 FPGA 器件架构中的更多资源元件。
- 异步复位还会影响通用逻辑结构的性能。由于所有赛灵思 FPGA 通用寄存器都能置位/复位为异步或同步，通常认为使用异步复位不会造成时间延迟。这种假设往往是不对的。使用全局异步复位时，虽然控制集的数量不会增多，但由于需要把这个复位信号路由到所有的寄存器元件，时序复杂性会增大。
- 在使用异步复位时，务必记住同步异步复位的取消断言。

- 在需要较大密度或精细化布局的情况下，同步复位能够更加灵活地实现控制集的重新映射。如果在布局更加理想的 slice 中发现有不兼容的复位，可以把同步复位重新映射到该寄存器的数据路径中。这样可以在需要时缩小走线宽度，提升密度，从而实现良好的适配和更加优异的性能。
- 异步复位可能需要多周期激活，以确保电路正确复位且稳定。如果正确地计时了，同步复位则不包含此要求。
- 在复位激活过程中，若异步复位有更高概率发生块 RAM、 LUTRAM、以及 SRL 存储内容翻转时，可以使用同步复位。
- DSP48 和块 RAM 等部分资源只为块中的寄存器元件提供同步复位。当需要在这些元件相关的寄存器元件上使用异步复位时，不能将这些寄存器直接调用到这些块中，否则会造成功能异常。

复位编码实例 1：乘法器与异步复位

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。图 3-2 展示使用具有异步复位的流水线寄存器基于 DSP48 的 16x16 位乘法器。综合必须使用常规结构寄存器作为输入级，以及一个外部寄存器和 32 个 LUT2 （红色标记）来模拟 DSP 输出上的异步复位 （DSP48 P 寄存器使能，但未连接到复位）。这需要额外的 65 个寄存器和 32 个 LUT，同时 DSP48 配置为 (AREG/BREG =0， MREG=0， PREG=1)。

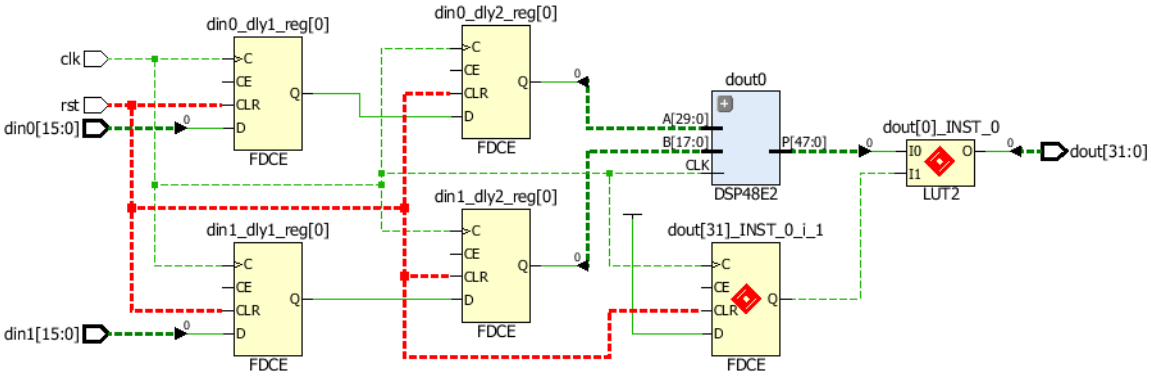


图 3-2：带有异步复位的流水线寄存器乘法器

如下图所示，通过简单地改变复位定义，使得乘法器流水线寄存器使用同步复位，综合时便可以可以利用 DSP48 内部寄存器 (AREG/BREG=1， MREG=1， PREG=1)。

```
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
```

图 3-3：在倍频器上将异步复位更改为同步复位

由于节省了结构资源并利用所有 DSP48 内部寄存器，设计性能和功率效率达到最佳。

复位编码实例 2：同步复位乘法器

如下所述，为了利用现有的 DSP 原语特征，前面的例子可以通过从异步复位到同步复位的改变来重写。

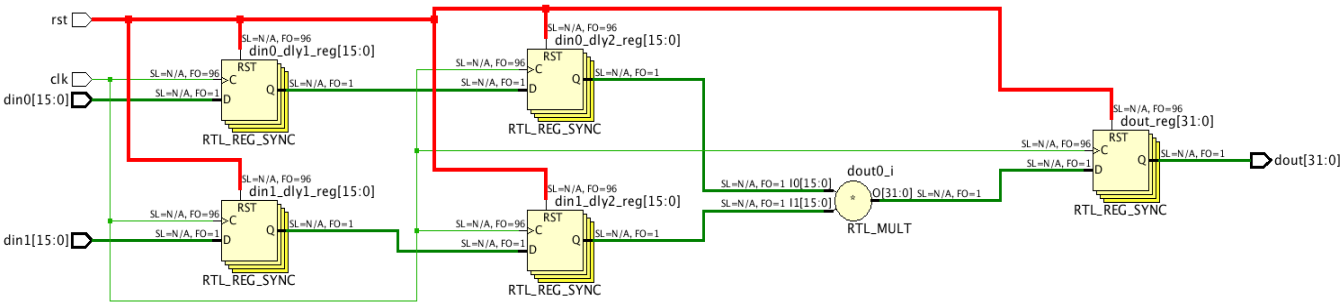


图 3-4：带有流水线寄存器的乘法器 （同步复位）

在这个电路中，DSP48 原语从具有 DSP 原语的所有流水线寄存器（AREG/BREG=1，MREG=1，PREG=1）中进行调用。

第二编码示例的实现具有优于第一编码示例的下列优点：

- 最佳资源使用率
- 更好的性能和更低的功耗
- 较低的端点数

此外，第二个编码实例更简明。

尝试消除 HDL 代码重置时出现问题

当优化代码以消除重置时，注释重置声明中的条件不会创建所需的结构，而是创建问题。例如，下图显示了三个流水线级，每个流水线级都使用了异步复位。如果试图通过用复位条件注释掉代码来消除两个流水线级的复位条件，则异步复位将被使能 （rst 的反逻辑）。

```
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
```

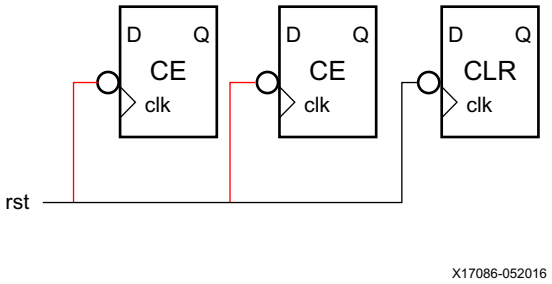


图 3-5：在复位条件注释输出代码

如下图所示，去除复位的最佳方法是创建单独的顺序逻辑过程，一个用于复位条件，另一个用于非复位条件。

```
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
```

图 3-6：有复位和无复位的寄存器单独程序语句

提示：使用复位时，确保程序语句中的所有寄存器都复位。

时钟使能

如果正确使用，时钟使能能够显著地降低系统功耗，同时对面积或性能的影响极小。但是如果不正确地使用时钟使能，可能会造成下列后果：

- 面积增大
- 密度减小
- 功耗上升
- 性能下降

在许多使用大量控制集的设计中，低扇出时钟使能可能是导致控制集数量众多的主要原因。

创建时钟使能

如果在同步块中编写不完整条件语句，就能创建时钟使能。调用时钟使能的目的是当前提条件无法满足时，保留最后一个值。如果这是需要的功能，用这种方式编码就是有效的。但是在有些情况下，虽然前提条件值未得到满足，但输出无所谓。此时赛灵思建议用设定的常数 （即为信号赋值 1 或 0）关闭该条件 （即使用 `else` 子句）。

在大多数实现方案中，这不会造成额外的逻辑，同时可避免使用时钟使能。不过有个情况例外，即对大型总线而言，如果调用时钟使能信号，保持上述的最后一个值，有助于降低功耗。基本前提是如果调用的寄存器数量较少，时钟使能会存在较大弊端，因为它会增加控制集的数量。但是对较大型的群组而言，其利大于弊，所以建议使用。

复位和时钟使能的先后

在赛灵思 FPGA 器件中，所有寄存器的置位/复位功能的优先级均高于时钟使能，不论是异步置位/复位还是同步置位/复位都是如此。为取得最佳结果，赛灵思建议在同步块中的 `if/else` 结构中，应一直在时钟使能 （如有必要使用）之前对置位/复位进行编码。优先对时钟使能进行编码会强制复位进入数据路径，并导致产生更多逻辑。

如需了解更多有关时钟的信息，请参阅“编码指南”。

使用综合属性控制使能/重置提取

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

当设计包括同步复位/使能时，当负载等于或高于由 `-control_set_opt_threshold` 综合开关设置的阈值时，通过 CE/R/S 引脚映射的综合创建逻辑椎，或如果低于阈值，通过 D 引脚映射创建逻辑椎。默认阈值为：

- 7 系列器件：4
- UltraScale 器件：2

使用 DIRECT_ENABLE 和 DIRECT_RESET

要使用控制器集映射，你可以将属性应用于连接启用/复位信号的网络，但这将强制综合使用 CE/R 引脚。

在下图中，使能信号 (en) 只连接到一个触发器上。因此，综合引擎将 en 信号连接到逻辑的 FDRE/D 引脚椎。注意，CE 引脚连接到逻辑 1。

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

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

endmodule
```

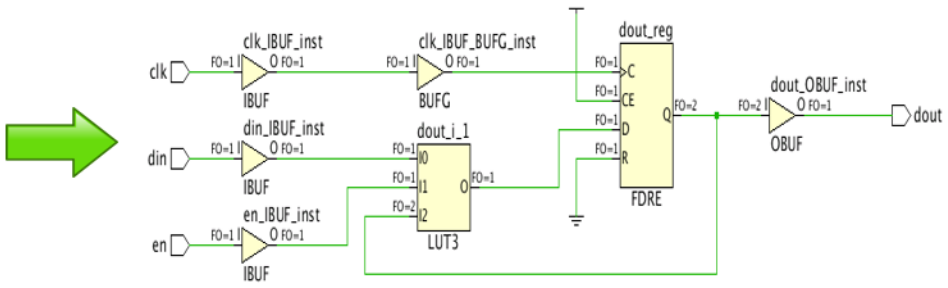


图 3-7：使用数据路径逻辑的时钟使能进行设计实现

要覆盖此默认行为，可以使用 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 引脚。

```
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
```

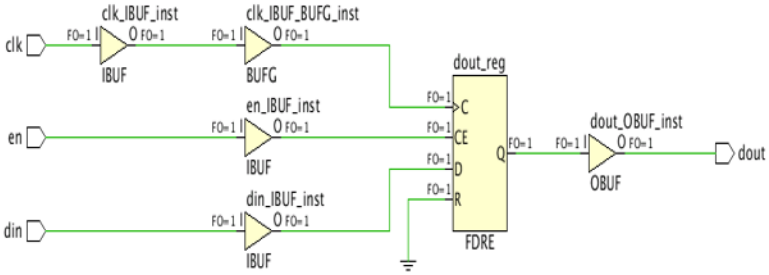


图 3-8：使用 direct_enable 实现专用时钟使能

下图显示了 RTL 代码，其中 global_rst 或 int_rst 可以复位寄存器。默认情况下，两者都映射到逻辑的复位引脚椎。

```
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
```

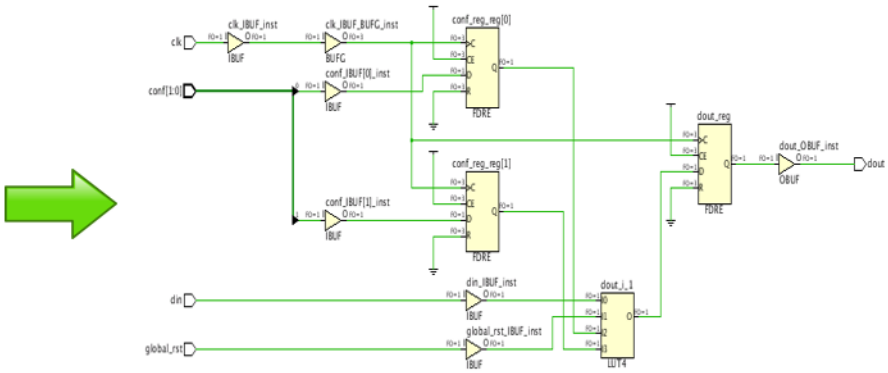


图 3-9：通过数据路径逻辑映射的多个复位条件

您可以使用 DIRECT_RESET 属性来指定连接到寄存器复位引脚的复位信号。例如，下图显示了如何使用 DIRECT_RESET 属性仅将 global_rst 信号连接到寄存器 FDRE/R 引脚，并将 int_rst 信号连接到逻辑的 FDRE/D 椎形。

```
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
```

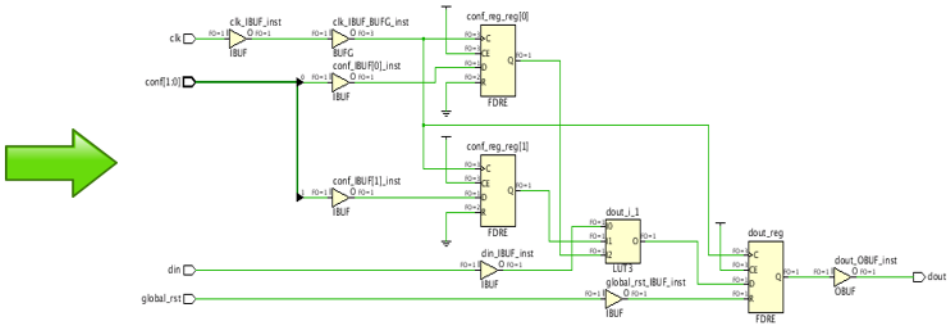


图 3-10：使用 DIRECT_RESET 属性的专用复位引脚的用法

将逻辑从控制引脚推到数据引脚

在分析关键路径时，您可能会发现以控制引脚结束的多个路径。您必须分析这些路径，以确定是否有一种方法将逻辑推入数据路径，而不会产生损失，例如额外的逻辑层。由于存在最后一个 LUT 的输出到 FF 的 D 输入的直接连接，在给

定相同逻辑层的情况下，到 D 引脚路径的延迟比到 CE/R/S 引脚少。以下编码示例说明如何将逻辑从控制引脚推送到寄存器的数据引脚。

在以下示例中， dout_reg [0] 的使能引脚具有 2 个逻辑层，数据引脚具有 0 个逻辑层。在这种情况下，通过将 RTL 文件中 dout 寄存器定义中的 EXTRACT_ENABLE 属性设置为“no”，可以将使能逻辑移至 D 引脚，从而提高时序。

```
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
```

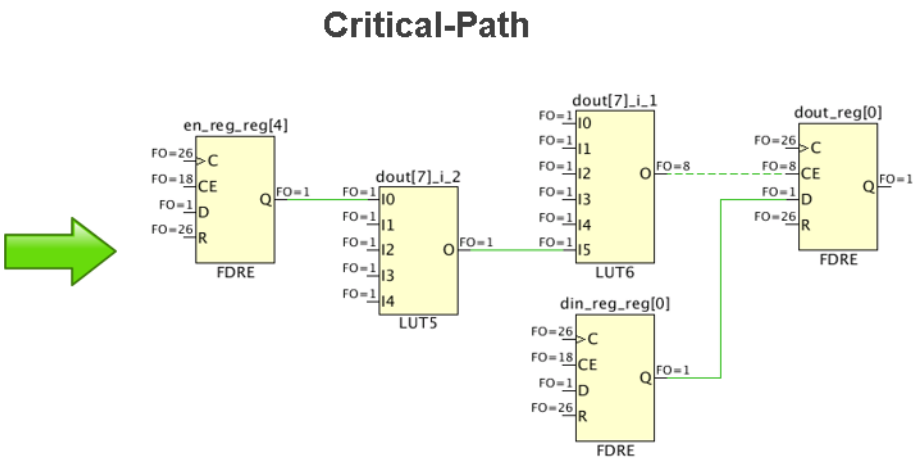


图 3-11：寄存器的控制引脚 （使能）的临界路径结束

以下示例显示如何分离组合和顺序逻辑并将完整逻辑映射到数据路径。这将逻辑推入仍然具有 2 个逻辑层的 D 引脚。

您可以通过将 EXTRACT_ENABLE 属性设置为“no”来实现相同的结构。如需了解更多有关 EXTRACT_ENABLE 属性的信息，请参阅 《Vivado Design Suite 用户指南：综合》(UG901) [参照 16]。

```
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
```

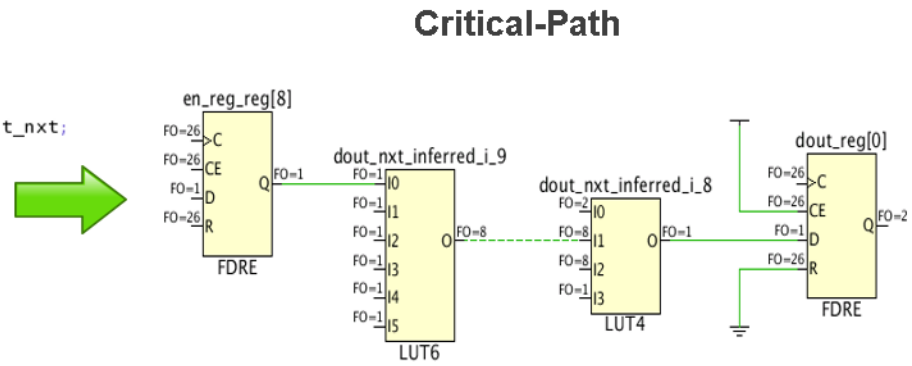


图 3-12：寄存器数据引脚的临界路径结束 （禁止使能提取）

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，应务必与异步复位的解除保持同步。

掌握调用的结果

您的代码最终必须映射到器件资源上。应努力掌握所针对架构中的关键算术、存储和逻辑元件。因此在对设计功能进行编码时，应预计可供代码映射的硬件资源。了解这种映射，可让您尽早洞察出任何潜在的问题。

下面的实例将证明掌握硬件资源和映射如何有助于弄清楚设计决策：

- 对大于四位的加法、减法和加减法，一般会使用进位链，而且每两位相加使用一个 LUT （即 8 位与 8 位相加使用 8 个寄存器和相关的进位链）。在三值相加时或把一个加法器的结果与另一个值且在其间不使用寄存器的情况下相加时，每三位相加使用一个 LUT （即 8 位、 8 位、 8 位相加也使用 8 个寄存器和相关的进位链）。

  如果需超过一次以上的相加，可以在每两级相加后设置寄存器以实现三值相加，从而将器件的占用率减半。

- 一般乘法针对的是 DSP 块。位宽不足 18x25 （在 UltraScale 器件中为 18x27）的符号位映射到单个 DSP 块。会产生更大乘积的乘法可映射到一个以上的 DSP 块。 DSP 块内部有流水线化资源。

  适当地将调用到 DSP 块中的逻辑流水线化，能够显著提升性能和降低功耗。在描述乘法的时候，围绕其进行三级流水线化可实现最佳的建立、输出相对于时钟时延 (clock-to-out) 和功耗特性。太浅的流水线化 （一或零级）可能造成时序问题，增加这些块的功耗，同时 DSP 中的流水线寄存器未得到利用。

- 深度为 16 位或者更浅的两个 SRL 可以映射到一个 LUT，高达 32 位的单个 SRL 也可以映射到一个 LUT 中。
- 对用条件代码得到标准 MUX 组件的情况：
  - 4 选 1 的 MUX (4-to-1 MUX) 可实现在单个 LUT 中，产生一个逻辑层。
  - 8 选 1 的 MUX (8-to-1 MUX) 可实现在两个 LUT 和一个 MUXF7 组件中，实际上仍然只产生一个逻辑 (LUT) 层。
  - 16 选 1 的 MUX (16-to-1 MUX) 可实现在四个 LUT 及一个 MUXF7 和 MUXF8 组件组合中，实际上还是只产生一个逻辑 (LUT) 层。

把 LUT、 MUXF7 和 MUXF8 组合在同一 CLB/slice 结构中，产生的组合延迟极低。因此这些组合可视为等同于仅一个逻辑层。了解这一代码有助于更好地管理资源，也有助于更好地鉴别和控制逻辑层，设置合理的数据路径。

对通用逻辑而言，考虑给定寄存器具有唯一的输入数。根据这个数量，就可以估计出可能实现的 LUT 数量和逻辑层数量。一般来说， 6 个或六个以下输入会产生一个逻辑层。理论上两个逻辑层可以管理多达 36 个输入。但实际上两个逻辑层最多仅能管理 20 个输入。一般来说如果输入数量越多，逻辑等式越复杂，那么需要的 LUT 和逻辑层就越多。

重要提示：检查硬件资源的可用性以及如何在设计周期早期高效率地加以运用，从而简化修改工作。这种方法与在时序收敛过程中等待直到设计周期尾声相比，更能提供优质的结果。

调用 RAM 和 ROM

可以用多种方式设定 RAM 和 ROM。每种方法都有各自的优势和不足。

- **调用**

  优势：

  - 高度可移植
  - 便于阅读和理解
  - 自我文档化
  - 快速仿真

  不足：

  - 不能访问所有可用的 RAM 配置
  - 可能产生不够理想的结果

  由于调用一般能产生良好的结果，因此建议采用此方法，除非不支持指定的用途，或是产生的结果在性能、占位面积或功耗上不令人满意。如果发生这种情况，建议尝试其它方法。

  在调用 RAM 时，赛灵思建议使用 Vivado 工具中提供的 HDL 模板。如前文所述，使用异步复位会给 RAM 调用造成不利影响，应避免使用。请参阅“使用 Vivado Design Suite HDL 模板”。

- **赛灵思可参数化宏 (XPM)**

  优势：

  - 可在赛灵思器件系列之间移植
  - 快速仿真
  - 支持不对称宽度
  - 可预测的 QoR

  不足：

  - 仅支持 XPM 选项

  XPM 是基于不能修改的固定模板的调用构建的。因此，他们可以保证 QoR，并且可以支持标准调用没有的功能。当标准调用不支持所需的功能时，赛灵思建议您改用 XPM。

  注释：当使用 `compile_simlib` 编译仿真库时， XPM 会自动编译。如需了解更多信息，请参阅《Vivado Design Suite 用户指南：逻辑仿真》(UG900) [参照 11]。

- **直接实例化 RAM 原语**

  优势：

  - 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**

  优势：

  - 在使用多个组件时一般能提供更优化的结果
  - 设定和配置简单

  不足：

  - 代码可移植性差
  - 需要管理内核

实现 RAM 时应该考虑的性能因素

要高效地调用存储元件，需要考虑下列影响性能的因素：

- 使用专用块还是分布式 RAM

  RAM 可以在专用块 RAM 或使用分布式 RAM 的 LUT 内实现。不同的选择会影响资源选择，同时还会严重地影响性能和功耗。

  一般来说 RAM 要求的深度是首要选择标准。高达 64 位深度的存储器阵列一般实现在 LUTRAM 中，其中深度不超过 32 位的映射为每个 LUT 的 2 位，深度达到 64 位的映射为每个 LUT 的 1 位。深度更大的 RAM 根据可用资源和综合工具赋值，也可实现在 LUTRAM 中。

  深度超过 256 位的存储器阵列一般实现在块存储器中。赛灵思 FPGA 器件能够灵活地以多种宽度深度组合映射此类阵列。用户需要熟悉这些配置，才能了解代码中更大规模存储器阵列声明所使用的块 RAM 的数量与结构。

- 使用输出流水线寄存器

高性能设计要求使用输出流水线寄存器，同时还建议所有设计都应使用输出寄存器。这可以优化块 RAM 的时钟输出时序。另外增加第二个输出寄存器也有好处，因为与块 RAM 寄存器相比， slice 输出寄存器具有更快的时钟输出时序。使两个寄存器的总读取延迟为 3。在调用这些寄存器时，它们应以 RAM 阵列的方式存在于相同层级上。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 数组很大并映射到许多原语时，它们可以跨越相当大的模具区域。这可能导致地址和控制线上的性能问题。考虑在生成这些信号之后和 RAM 之前添加一个额外的寄存器。为了进一步提高时序，稍后在流程中使用 `phys_opt_design` 来复制此寄存器。无逻辑的寄存器的输入将更容易复制。

场景防止块 RAM 输出寄存器调用

因为这是确保调用的最简单方法，所以赛灵思建议，存储器和输出寄存器都在单个层次结构中调用。有两种情况会调用出块 RAM 输出寄存器。第一个是在输出上存在额外寄存器时，第二个是当读取地址寄存器在存储器阵列中重新时序时。只能使用单端口 RAM。如下图所示：

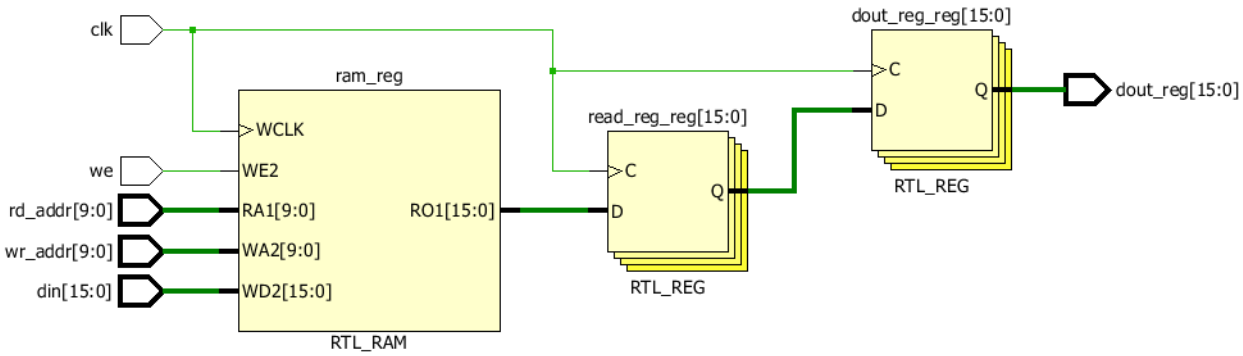


图 3-13：具有额外的读寄存器的 RAM 与用于块 RAM 输出寄存器的调用

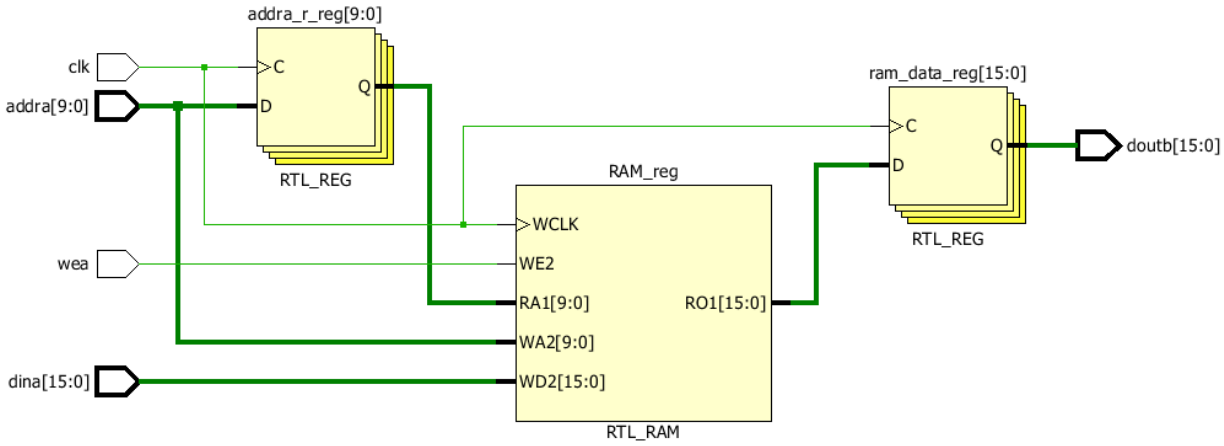


图 3-14：在地址寄存器重定时之前的 RAM 视图

与这些示例的某些偏离可以防止输出寄存器的调用。

检查读取数据寄存器输出的多扇出

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。这在下图中进行了说明。

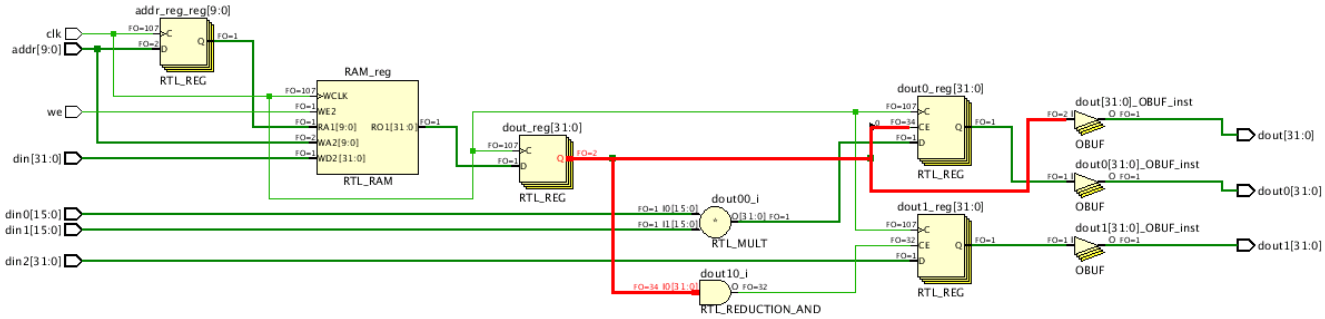


图 3-15：多重扇出防止块 RAM 输出寄存器调用

检查地址/读数据寄存器上的复位信号

不应重置存储器阵列。只有 RAM 的输出可以容许复位。复位必须是同步的，以便将输出寄存器调用到 RAM 基元中。异步复位将使寄存器不被调用为 RAM 基元。此外，输出信号只能复位为 0。

下图为确保 RAM 和输出寄存器的正确调用而应避免的示例。

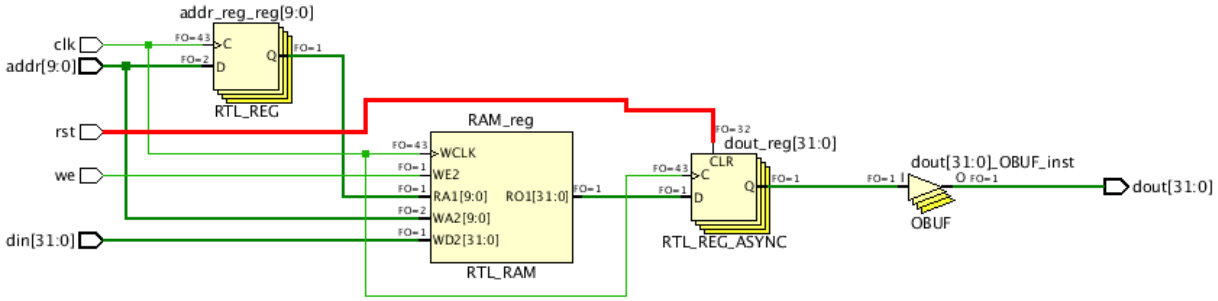


图 3-16：检查地址/读取数据寄存器上的复位

检查寄存器中的反馈结构

请确保寄存器没有反馈逻辑，在下面的示例中，由于加法器需要寄存器的当前值，所以该逻辑不能重新时序并打包到块 RAM 中。结果电路是没有输出寄存器的块 RAM （DOA_REG 和 DOB_REG 设置为‘0’）。

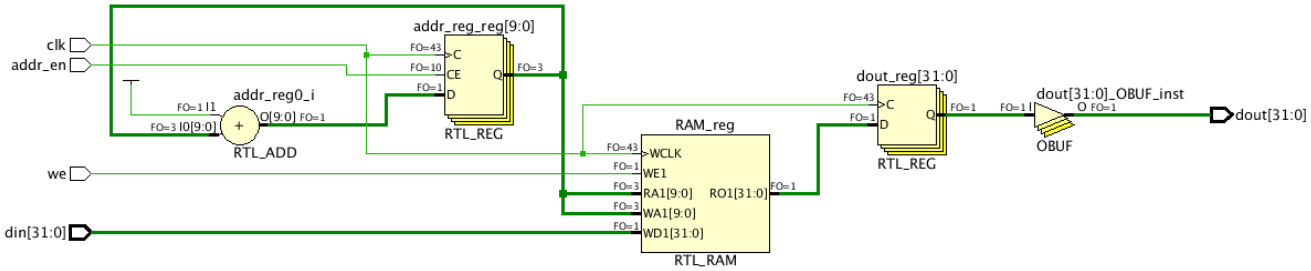


图 3-17：检查 RAM 块周围寄存器的反馈是否存在

将存储器映射到 UltraRAM 块

UltraRAM 是一个 4Kx72 内存块，具有两个端口，使用单个时钟。此原语仅在某些 UltraScale+™ 器件中可用。在这些器件中，除了块 RAM 资源之外，还包括 UltraRAM。

UltraRAM 可以在您的设计中使用以下方法：

- 依靠综合来调用 UltraRAM，通过在 HDL 中的内存声明上设置 `ram_style = "ultra"` 属性。
- 例化赛灵思 XPM_MEMORY 原语。
- 例化 UltraRAM UNISIM 原语。

以下代码示例显示了 XPM 内存的例化，并在 HDL 语言模板中提供。突出显示的参数 `MEMORY_PRIMITIVE` 和 `READ_LATENCY` 是将存储器调用为 UltraRAM 的高性能的关键参数。

- `MEMORY_PRIMITIVE = "ultra"` 指定内存被调用为 UltraRAM。
- `READ_LATENCY` 定义存储器输出上存在的流水线寄存器的数量。

较大的存储器映射到配置为行 × 列结构的多个 UltraRAM 单元组成的 UltraRAM 矩阵。

基于深度，可以使用单列或多列创建矩阵。 URAM 列高度的当前默认阈值为 8，并且可以使用属性 CASCADE_HEIGHT 进行控制。

单列和多列 UltraRAM 矩阵之间的区别如下：

- 单列 UltraRAM 矩阵使用内置的硬件级联，无架构逻辑。
- 多列 UltraRAM 矩阵在每列中使用内置硬件级联，以及连接列的一些结构逻辑。可能需要额外的流水线才能保持性能。这是通过增加读取延迟来调用的。 Vivado 会根据需要将这些寄存器自动封装到 UltraRAM 中。

```
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),
  .addra          (addra),
  .dina           (dina),
  .injectsbiterra (1'b0),  //do not change
  .injectdbiterra (1'b0),  //do not change
  .douta          (douta),
  .sbiterra       (),      //do not change
  .dbiterra       ()       //do not change

);
```

图 3-18：在 RTL 代码中指定 UltraRAM （通过 XPM）

上面的示例使用 32 K x 72 内存配置，并使用八个 URAM。为了提高 UltraRAM 的性能，应该在级联链中添加更多的流水线寄存器。这是通过增加读延迟整数实现的。

如需了解更多在 Vivado 综合中调用 UltraRAM 的信息，请参阅 《Vivado Design Suite 用户指南：综合》(UG901) [参照 16] 中的链接。

为适用 DSP 和算术调用进行编码

赛灵思 7 系列 FPGA 中的 DSP 块能够执行多种不同的功能，包括：

- 乘法
- 加法和减法
- 比较器
- 计数器
- 普通逻辑

DSP 块是一款具有多重寄存器级的高度流水线化块，能在降低资源总体功耗的情况下实现高速运行。赛灵思建议把准备映射到 DSP48 中的代码完全流水线化，这样可以利用所有的流水线级。为了灵活使用这一额外资源，功能中应避免使用置位条件，才能正确地映射到这一资源上。

赛灵思器件中的 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、计数器或其它可在 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。导致置位或异步复位的代码会产生在占位面积、性能和功耗方面欠佳的设计结果。

许多 DSP 设计都非常适合采用赛灵思架构。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利用这些资源的优势。

DSP48 块使用符号算术实现方案。赛灵思建议在 HDL 源代码中使用采用符号值的代码，以便最佳地符合资源功能并实现最有效的映射。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，但由于无符号到有符号的转换，可能无法实现该组件完整的位精度。

如果目标设计预计包含大量加法器，赛灵思建议对设计进行评估，以更好地利用 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，可以用加法器级联来构建脉动滤波器，而不必使用多重顺序加法功能 （加法器树）。如果是对称滤波器，可以评估使用专用预加法器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）。

如果选择使用加法树，选择 6 输入 LUT 架构，使用和简单的 2 输入加法一样多的资源，就可以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，赛灵思建议调用 DSP 资源

如需了解更多有关 DSP48 slice 特性和功能，以及如何根据自己的设计需要充分利用该资源的信息，请参阅 《7 系列 DSP48E1 slice 用户指南》(UG479) [参照 44] 和 《UltraScale 架构 DSP slice 用户指南》(UG579) [参照 45]。

移位寄存器和延迟线编码

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

赛灵思 FPGA 器件提供专用 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。但是这些元件只支持左 (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 （SRL16 的 A3、 A2、 A1、 A0 输入）。移位寄存器的长度可以是固定的静态长度，也可以动态调节。

在动态模式下，每当有新的地址施加到地址引脚，在访问 LUT 造成的延迟之后，就会在 Q 输出上提供新的位置值。同步和异步置位/复位控制信号未在 SRL 原语中提供。但是，如果您的 RTL 代码包含复位，则赛灵思综合工具会调用 SRL 周围的其他逻辑，以提供复位功能。

为在使用 SRL 时获得最佳性能，赛灵思建议把最后一级移位寄存器实现在专用 slice 寄存器中。 slice 寄存器与 SRL 相比，输出相对于时钟时延 (clock-to-out) 更短。这样可以为从移位寄存器逻辑出发的路径提供更大的时序裕量。综合工具将自动调用此寄存器，除非该资源被示例化或由于属性或跨层次边界优化限制而阻止综合工具调用这样的寄存器。

赛灵思建议您使用 Vivado Design Suite HDL 模板中展示的 HDL 编码样式。

如果要使用寄存器来达到灵活布局芯片的目的，使用下列属性关闭 SRL 调用：

```
SHREG_EXTRACT = “no”
```

如需了解更多有关综合属性及如何用 HDL 代码设定这些属性，请参阅 《Vivado Design Suite 用户指南：综合》 (UG901) [参照 16]。

初始化全部调用的寄存器、 SRL 和存储器

GSR 网络用于根据 HDL 代码中规定的初始值完成所有寄存器的初始化。如果没有设定初始值，综合工具会自行将初始状态赋值为 0 或 1。除少数情况，比如 one-hot 状态机编码， Vivado 综合工具一般都设定默认值为 0。

任何调用的 SRL、存储器或其它同步元件也可能都有设定的初始状态，可在配置时编程到相关元件中。

赛灵思强烈建议相应地初始化所有的同步元件。寄存器的初始化完全可使用各种主要的 FPGA 综合工具加以调用。因为经配置后 FPGA 器件中所有的同步元件都会从已知值启动，这样做可避免纯粹为初始化目的添加复位功能，让 RTL 代码在功能仿真中更贴近实现的设计。

寄存器和锁存器初始状态 VHDL 编码实例 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 编码实例 1：

```
reg register1 = 1’b0; // specifying regsiter1 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
```

寄存器和锁存器初始状态 Verilog 编码实例 2：

另外还可以在 Verilog 中使用 `initial` 声明：

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

判断实例化或调用的时机

赛灵思建议用户使用 RTL 来描述设计，然后用综合工具把代码映射到 FPGA 器件中的可用资源上。调用得到的逻辑不仅能够增强代码的可移植性，还便于综合工具查看，以完成各项功能优化。优化内容包括逻辑复制、结构重组与融合，以及重新定时以均衡各寄存器间的逻辑延迟。

综合工具优化

在完成器件库单元实例化之后，在默认条件下综合工具不会对其优化。即便是有指令要求优化器件的库单元，综合工具一般也无法达到和 RTL 相同的优化水平。因此综合工具一般只优化来往于这些单元之间的路径，但优化不会穿越这些单元的路径。

例如，如果某个 SRL 被实例化，构成某个长路径的组成部分，则该路径可能成为瓶颈。与普通的寄存器相比， SRL 的输出相对于 clock-to-out 延迟更长。为了保持 SRL 带来的占位面积缩小，同时提高其时钟输出性能的优势，需要创建一个延迟应小于实际允许延迟的 SRL，并将其在最后一级实现为常规的触发器。

判断实例化的时机

在使用综合工具映射无法满足时序、功耗或占位面积约束时，或是无法调用 FPGA 器件中的特定功能时，就需要使用实例化。

用户使用实例化可以对综合工具进行整体控制。例如：为实现更卓越的性能，可以只使用 LUT 来实现比较器，而不用综合工具一般选择的 LUT 和进位链元件组合。

有时实例化可能是唯一能够利用器件中可用复杂资源的途径。原因可能是：

- HDL 语言限制

  例如在 VHDL 中无法描述双数据速率 (DDR) 输出，因为这需要两个单独的进程来驱动相同的信号。

- 硬件复杂性

  与创建可综合的描述相比，实例化 I/O SerDes 元件更为简单。

- 综合工具调用限制

  例如综合工具不能根据 RTL 描述调用硬 FIFO。因此用户必须通过示例化来实现。

  如果用户决定实例化一条赛灵思原语，请参阅适用于目标架构的用户指南和库指南，充分了解组件的功能、配置和连接功能。

  对调用和实例化，赛灵思都建议用户使用 Vivado Design Suite 语言模板中提供的实例化和语言模板。

提示：
- 尽可能调用功能。

提示： - 当综合 RTL 代码无法满足要求时，如果要用器件库组件实例化来替代代码，应先审核相关要求。

提示： - 在编写通用 Verilog 和 VHDL 行为指令或在有必要实例化所需原语时，应考虑使用 Vivado Design Suite 语言模板。

提升性能的编码方式

对高性能设计而言，本章节中讨论的编码方法 （提升性能的编码方法）能够减少潜在的时序问题。

关键路径上的高扇出

高扇出网络宜在设计过程早期阶段进行处理。性能要求和路径的结构往往会导致高扇出问题。您可以使用以下技术来解决高扇出网络的问题。

建议： 尽早检查有大量负载的网络，评估其对总体设计的影响。

精简负载到不需要高扇出网络的设计部分

对高扇出控制信号，评估设计的所有编码部分是否都需要高扇出网络。降低负载需求可以大幅度减少时序问题。

使用寄存器复制

寄存器复制可以通过复制寄存器来加快关键路径的速度，以减少给定信号的扇出。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。

如果根据时序报告，带有长布线延迟的高扇出网络被报告为关键路径，应考虑复制综合工具上的约束和手动复制寄存器。您往往必须添加额外的综合约束，才能确保手动复制的寄存器不会被综合工具优化掉。

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。对这个全局阈值进行调整，就可以自动复制高扇出网络，但无法提供更精细的用户控制水平，即决定所能够复制的特定寄存器。更好的方法是对特定寄存器或层级级数施加属性，确定哪些寄存器能够或者不能够复制。如果发现有 LUT1 （而不是寄存器）用于复制工作，就说明有属性或约束应用错误。

请勿复制寄存器用于同步跨时钟域的信号。如果在这些寄存器上添加 ASYNC_REG 属性就会造成工具无法复制寄存器。如果同步链有极高扇出且有必须使用复制来满足时序要求，最后一个寄存器可通过移除其上的 ASYNC_REG 属性来完成复制。

但是，由于该寄存器不再是同步链的组成部分，这一更改会修改同步器的 MFTB。如果链中有三个或更多同步寄存器，只有移除 ASYNC_REG 属性才是安全的。建议的替代方法是在同步器后添加流水线寄存器，以实现复制。

下表是您设计中可接受的扇出数量提示性指南。

表 3-1：扇出指南

条件	扇出 < 5000	扇出 < 200	扇出 < 100
低频 1 到 125 MHz	同步逻辑之间基本没有逻辑级数 最高频率时逻辑级数小于 13 个	不適用	不適用
中频 125 到 250 MHz	如果设计不符合时序，则可能需要降低扇出和/或逻辑层。	最高频率时逻辑级数小于 6 个。（驱动器和负载类型会影响性能。）	不適用
高频 大于 250 MHz	对大多数设计不建议使用。	实现更高速度通常需要较少的逻辑级数。	需要先进的流水线方法、精心的逻辑复制、紧凑的功能、较少逻辑级数。（驱动器和负载类型会影响性能。）

提示：如果时序报告提示高扇出信号将限制设计性能，应考虑对其进行复制。 Phys_opt_design 命令可以显著改进寄存器复制工作。如需了解更多信息，请参阅第 4 章中的“MAX_FANOUT”。

提示：当复制寄存器时，应采用惯例对其命名，例如 <original_name>_a、 <original_name>_b 等，以便轻松理解复制的意图，从而更易于将来进行 RTL 代码的维护工作。

流水线考虑因素

另一种提升性能的方法是对拥有多个逻辑级数的长数据路径进行重新组织，并将其分配在多个时钟周期上。这种方法以延迟和流水线开销逻辑管理为代价，来达到加快时钟周期和提高数据吞吐量的目的。

由于 FPGA 器件带有大量的寄存器，额外的寄存器和开销逻辑通常不是问题。但是，数据路径跨多个周期，您必须对设计的其余部分进行特殊注意，以考虑所添加的路径延迟。

考虑 SSI 器件的流水线

在对需要跨越 SLR 边界的高性能寄存器间连接进行设计时，必须在 HDL 代码中描述正确的流水线，同时在综合中加以控制。这样可以确保移位寄存器 LUT (SRL) 调用和其它优化工作不会发生在必须跨越 SLR 边界的逻辑路径上。以这种方式修改代码加上适当使用 Pblock 可以设定 SLR 边界跨越发生的位置。

预先考虑流水线

预先而不是滞后考虑流水线可以降低时序收敛的难度。在较晚阶段对特定路径添加流水线常常会跨越电路传输延迟差异。这样一个看似微小的修改可能需要对部分代码进行大规模的重新设计。

在设计中尽早发现需要使用流水线的地方往往可以大幅度提升时序收敛、实现运行时间 （因时序问题更容易解决）和器件功耗 （因逻辑转换数量下降）。

检查调用的逻辑

在对您的设计进行编码时，应注意正在调用的逻辑。在想要添加流水线时，应注意下列条件：

- 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

- 具有布局限制、输出相对于时钟时延 (clock-to-out) 更短或大量建立要求的块

  例如，没有输出寄存器的块 RAM 或没有正确流水线化的算术代码

- 强制布局导致长布线

  例如一个管脚强制一条跨越芯片的布线，可能需要流水线来实现高速运行

下图中，时钟速度受下列因素限制：

- 源触发器的时钟到输出时间；
- 贯穿四个逻辑级数的逻辑延迟；
- 四个函数发生器的相关布线；
- 目的地寄存器的建立时间。

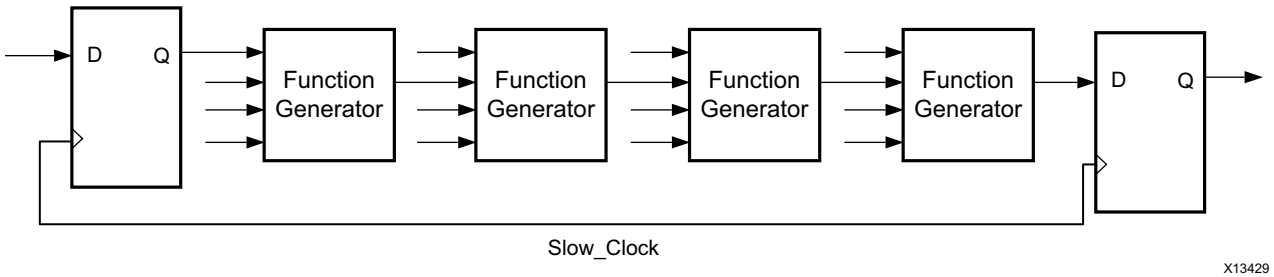


图 3-19：使用流水线前的原理图

下图中的数据路径与图 3-19 中的一样。由于触发器与函数发生器位于相同的 slice 中，时钟速度受源触发器的输出相对于 clock-to-out 延迟、贯穿一个逻辑级数的逻辑延迟、一个布线延迟和目的地寄存器的建立时间限制。在这个例子中，流水线化后的系统时钟的运行速度明显高于流水线化之前。

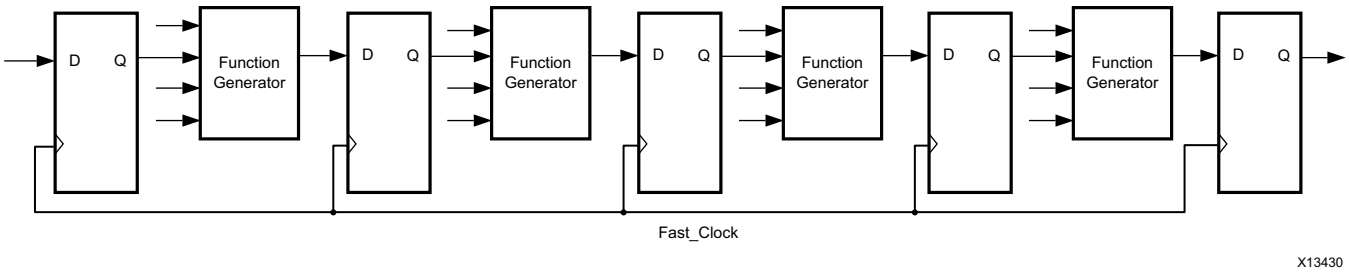


图 3-20：流水线化后的原理图

确定是否需要管道

常用的流水线技术可以识别大的组合逻辑路径，将其分成较小的路径，并在这些路径之间引入寄存器级，理想地平衡每个流水线级。

要确定设计是否需要流水线，请确时序钟的频率和分布在每个时钟组中的逻辑数量。您可以使用 `report_design_analysis` Tcl 命令确定每个时钟组的逻辑级分布。

提示：设计分析报告还突出显示具有零逻辑层的路径数，您可以使用这些路径确定在代码中进行修改的位置。

平衡延迟

要通过添加流水线级来平衡延迟，请将该级添加到控制路径而不是数据路径。数据路径包括更宽的总线，这增加了所使用的触发器和寄存器资源的数量。

例如，如果您有 128 位数据路径， 2 级寄存器，并且需要 5 个周期的时延，则可以添加 3 级寄存器：3 x 128 = 384 触发器。或者，您可以使用控制逻辑使能数据路径。使用单级位流水线的 5 级，并使能数据路径触发器。

注释：此示例仅适用于某些设计。例如，在存在来自中间数据路径触发器的扇出的情况下，仅具有 2 个级不工作。

建议：FPGA 中的最佳 LUT：FF 比为 1：1。具有显着更多 FF 的设计将增加不相关的逻辑打包成片，这将增加布线复杂性并且可以降低 QoR。

平衡管线深度和 SRL 使用

当有更深的流水线阶段时，要尽可能地映射到 SRL 中。这种方法保留了器件上的触发器/寄存器。例如， 9 层深的流水线级 （对于 32 的数据宽度）导致每个比特的 9 个触发器/寄存器，使用 32×9 = 288 个触发器。将相同的结构映射到 SRL 使用 32 个 SRL。每个 SRL 都连接到 5'b01000 的地址引脚 (A4A3A2A1A0)。

在综合期间调用 SRL 有多种方法，包括以下：

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

您可以使用 RTL 代码中的 `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使用不同的使能/复位控制信号。下面是在 9 深度流水线级中使用的复位的示例，其中复位连接到第三，第五和第八流水线级。使用这种结构，工具只将流水线级映射到寄存器，因为 SRL 原语有一个复位引脚。

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

利用 SRL 调用：

- 确保没有用于流水线级的重置。
- 分析是否真的需要重置。
- 在一个触发器上使用复位 （例如，在管道的第一级或最后一级）。

避免不必要的流水线

对于高度利用的设计，太多的流水线可能导致次优的结果。例如，不必要的流水线级增加了触发器和布线资源的数量，如果利用率高，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这么做的。

考虑流水线宏原语

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 以的上的频率范围工作。对于高频设计，赛灵思建议使用这些块中的所有流水线。

提升功耗的编码方式

提升功耗的编码方式包括：

门控时钟或数据路径

对时钟或数据路径实施门控是当不使用路径结果时用来停止跳变的常用技术。门控时钟能停止所有同步负载；并防止数据路径信号开关和毛刺继续传输。

该工具能分析描述内容和网表，以检测到不需要的条件。然而就应用、数据流和相关性而言，有些内容该工具没有提供，而且只能由用户来指定。

使门控元件数量最大化

最大限度地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比用时钟使能信号控制每个负载更能节省功耗。

使用专用时钟缓冲器的时钟使能引脚

当对时钟实施门控或多路复用以最大限度降低活跃度或时钟树使用量时，应采用专用时钟缓存的时钟使能端口。插入 LUT 或使用其它关闭时钟信号的方法在功耗和时序上效率不高。

关注控制集

前面已经讨论过，应该最大限度地减少控制集的数量。赛灵思建议只在门控时钟驱动大量同步元件时才进行时钟门控。否则，有可能造成寄存器的浪费。通过增加门控信号来停止数据或时钟路径的这种方式需要额外的逻辑和布线 （而且，功耗也相应增加）。应最大限度地减少附加结构的数量，以避免违背初衷。

建议：不要使用过度精细的时钟门控。每个门控时钟应能影响大量同步元件。

当不需要优先编码器时使用 Case 块

当不需要优先编码器时，应使用 case 块，而不是如果-则-否则 (if-then-else) 块或三元运算符。

低效率编码实例

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

正确编码实例

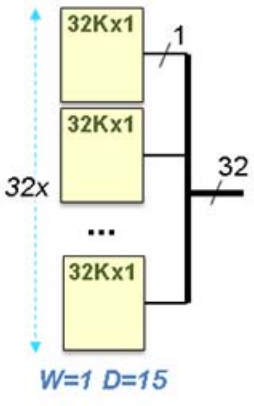
```
(* parallel_case *) casex  ({reg1, reg2, reg3})
1xx: val = reg_in1 ;
01x: val = reg_in2 ;
001: val = reg_in3 ;
default: val = reg_in4 ;
endcase
```

性能/功耗块 RAM 利弊取舍

有多种方式来打破内存配置以满足特定的需求。对特定设计的要求可以是性能，功率或两者的混合。

以下示例强调了可以生成以实现您的需求的不同结构。对于 UltraScale 和更高版本的器件，综合可以限制使用 CASCADE_HEIGHT 属性进行性能/功耗折衷的块 RAM 的级联。属性的用法和参数在 《Vivado Design Suite 用户指南：综合》(UG901) [参照 16] 进行了介绍。

下图显示了用于更高性能 （时序）的 32K x 32 内存配置示例。

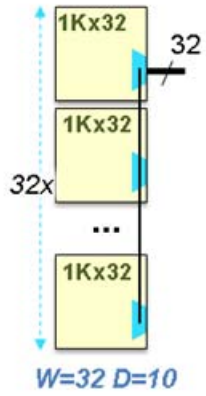



```
module test
(
input  clk,
input  we,
input  [14:0] addr,
input  [31:0] din,
output reg [31:0] dout
);

(* ram_style = "block", cascade_height = 1 *)
reg  [31:0] mem [(2**15)-1:0];
reg  [14:0] addr_reg;

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

endmodule
```

图 3-21：使用 1K x 32 和 CASCADE_HEIGHT=1 实现的 32K x 32 的 RTL 表达形式

该实现方案中，所有块 RAM 一直处于使能状态 （对于每次读或写）而且消耗更多功耗。

下图显示了级联所有块 RAM 的低功耗示例。

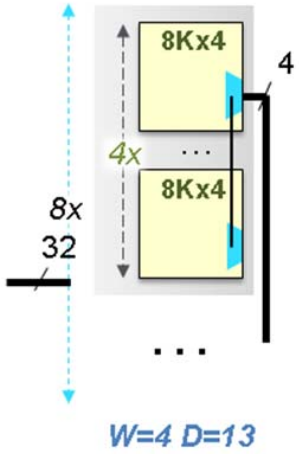



```
module test
(
input  clk,
input  we,
input  [14:0] addr,
input  [31:0] din,
output reg [31:0] dout
);

(* ram_style = "block", cascade_height = 32 *)
reg  [31:0] mem [(2**15)-1:0];
reg  [14:0] addr_reg;

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

endmodule
```

图 3-22：使用 1K x 32 和 CASCADE_HEIGHT=32 实现的 32K x 32 的 RTL 表达形式

该实现方案中，由于每次 （从每个单元中）只选择一个块 RAM，因此动态功耗几乎减半。 UltraScale 器件块 RAM 有专用级联多路复用器和布线结构，支持构建宽而深的存储器，需要一个以上的块 RAM 原语，采用极高能效的配置。

下图显示了如何限制级联并同时获得功耗和性能提高。




```
module test
(
input  clk,
input  we,
input  [14:0] addr,
input  [31:0] din,
output reg [31:0] dout
);

(* ram_style = "block", cascade_height = 4 *)
reg  [31:0] mem [(2**15)-1:0];
reg  [14:0] addr_reg;

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

endmodule
```

图 3-23：使用 8K x 32 和 CASCADE_HEIGHT=4 实现的 32K x 4 的 RTL 表达形式

在该实现中，因为一次选择 8 个块 RAM，所以动态功率贡献比高性能结构好，但不如低功率结构好。与低功率结构相比，该结构的优点在于，在级联路径中仅使用 4 个块 RAM，这与在低功率结构的关键路径中的 32 个块 RAM 相比对目标频率有影响。

分解更深的存储器配置，实现功耗与性能平衡

在使用更深的存储器配置工作时，您可以在 RTL 中使用 RAM_DECOMP 综合属性，通过提升内存组成来降低功耗。当 RAM_DECOMP 属性被应用到存储器阵列上时，存储器逻辑会被映射到更宽的块 RAM 原语阵列上。为平衡功耗与性

能，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。

作为例子，下图显示的是一个 32 x 16K 内存配置。

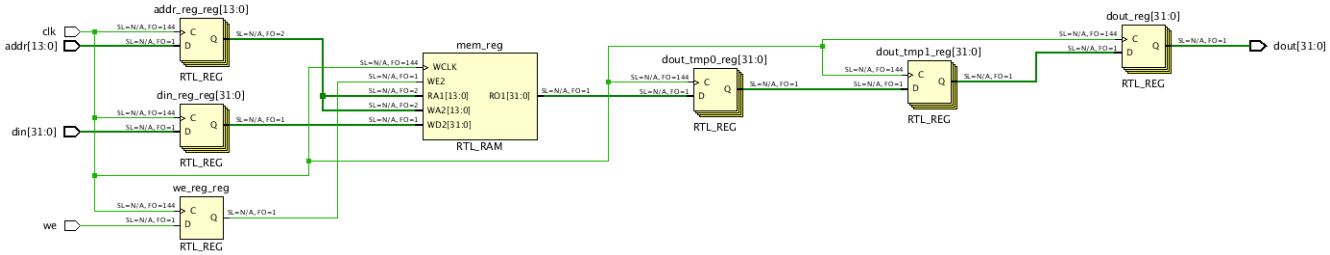


图 3-24：32 x 16K 内存配置

如果您应用下列属性：

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

推导 16 个 RAMB36E2 且内存按下列方式进行分解：

- 基本原语 32 x 1K。
- 级联 4 个块 RAM，以创建 32 x 4K 配置。
- 4 个并行结构可创建一个 16K 深内存。
- 复用输出生成输出数据。

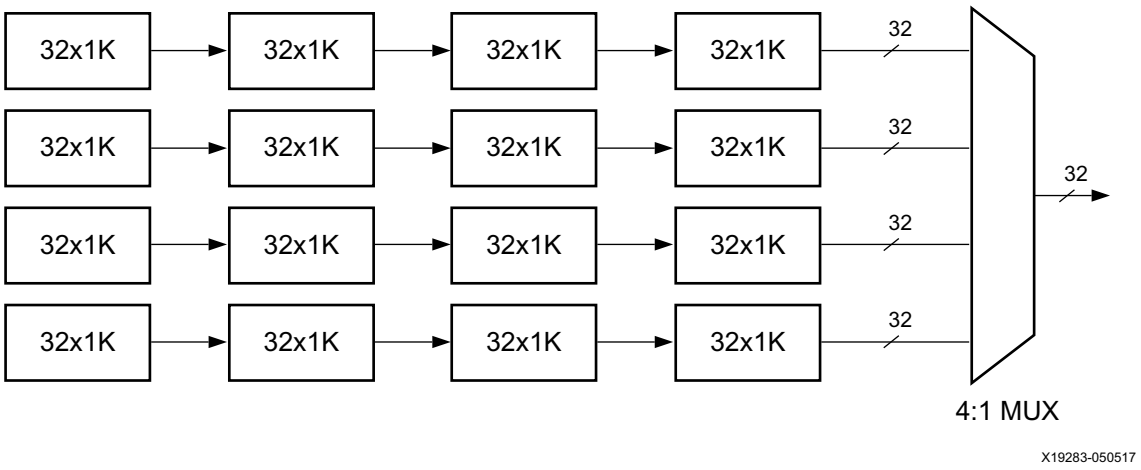


图 3-25：使用 CASCADE_HEIGHT 和 RAM_DECOMP 属性为 32 x 16K 内存配置例生成的结构

下列 RTL 代码例显示的是 CASCADE_HEIGHT 和 RAM_DECOMP 属性的使用。

```
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
```

图 3-26：使用 CASCADE_HEIGHT 和 RAM_DECOMP 属性的 32 x 16K 内存配置 RTL 代码

如果您只应用 ram_decomp = "power" 属性，可推导出 16 个 RAMB36E2 且内存可按下列方式进行分解：

- 基本原语 32 x 1K。
- 级联 8 个块 RAM，以创建 32 x 8K 配置。
- 2 个并行结构可创建一个 16K 深内存。
- 复用输出为 2:1 MUX，以生成输出数据。

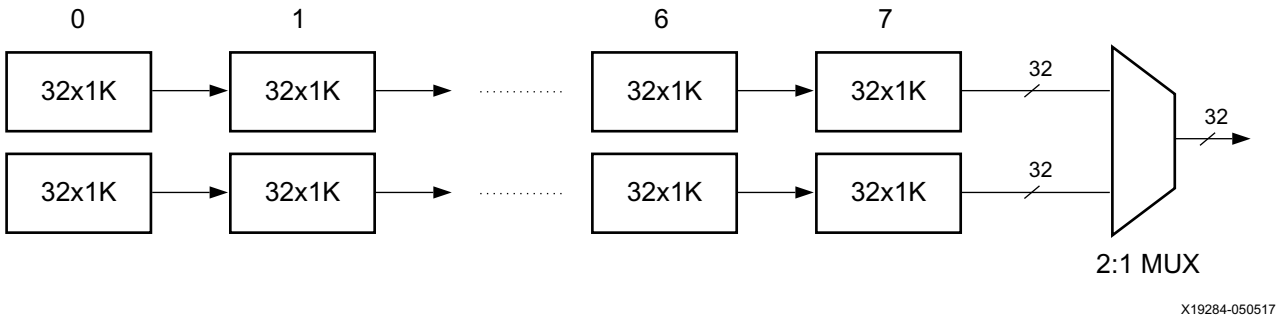


图 3-27：使用 RAM_DECOMP 属性为 32 x 16K 内存配置生成的结构

下列 RTL 代码例体现的是 RAM_DECOMP 属性的使用。

```
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         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
```

图 3-28：使用 RAM_DECOMP 属性的 32 x 16K 内存配置 RTL 代码

如果您只使用 RAM_DECOMP 属性，那么总体功耗下降幅度与同时使用 RAM_DECOMP 和 CASCADE_HEIGHT 属性相似，因为每次只有一个块 RAM 活跃。与深度为 8 的级联块 RAM 链相比，创建深度为 4 的级联块 RAM 链能提供更优异的性能。

如需了解更多信息，请参阅 《Vivado Design Suite 用户指南：综合》(UG901) [参照 16] 中的链接。

运行 RTL DRC

一套 RTL DRC 规则可识别您 HDL 的潜在编码问题。这些 DRC 检查可在 Flow Navigator 中通过“RTL Analysis > Report Methodology”执行，也可根据 Tcl 命令提示符通过执行 `report_methodology` 来实现。您可单击 Flow Navigator 下的“Open Elaborated Design”便可打开细化视图并进行检查。

编码指南

每个 FPGA 架构都为时钟提供有专用资源。掌握 FPGA 架构中的时钟资源，使您能够规划好自己的时钟，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，同时对每个时钟域上的扇出有良好的思路，就可以根据下面的时钟详情，研究出多种备选方案。如果您决定使用任何时钟资源，就需要具体地实例化相应的时钟元件。

UltraScale 器件时钟

与此前的器件架构相比， UltraScale 器件拥有不同的时钟结构；同时使全局时钟与区域时钟之间的界限变得模糊。UltraScale 器件没有像在 7 系列中见到的区域时钟缓冲器，取而代之的是公用缓存与时钟布线结构，无论负载是本地/区域还是全局的。

UltraScale 器件具有跨器件的固定的较小时钟域，并且时钟域在水平方向上不再跨越器件宽度的一半。在每个 UltraScale 器件上每个时钟域的数量不同。每个时钟域包含一个时钟网络布线，它分为 24 个垂直和水平布线轨道和 24 个垂直和水平分布轨道。下图显示了具有 36 个时钟域（6 列 x 6 行）的器件。同等的 7 系列器件有 12 个时钟域（2 列 x 6 行）。

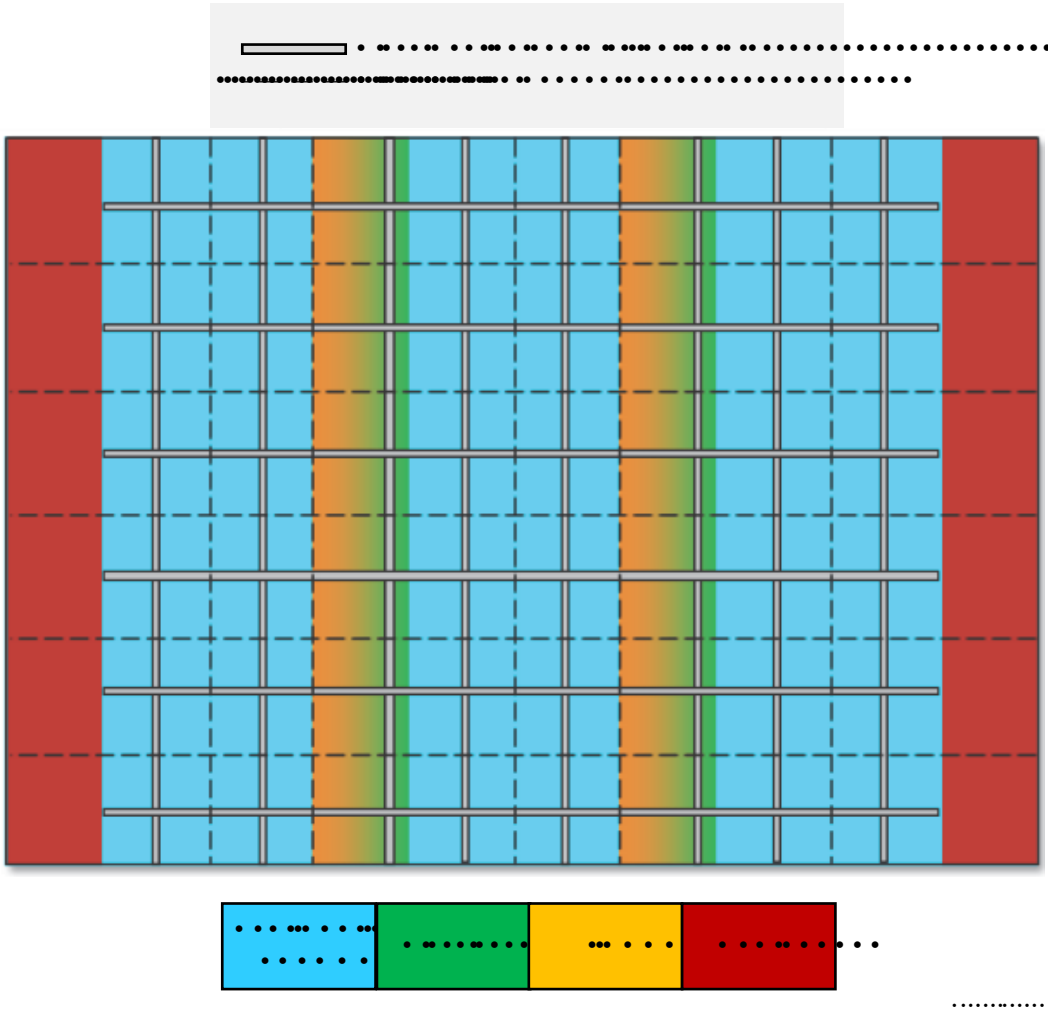


图 3-29：UltraScale 器件时钟域块

时钟架构设计为使得仅使用为给定布局连接时钟缓冲器和负载所需的时钟资源，并且在没有负载的时钟域中没有资源浪费。有效的时钟资源利用率支持架构中的更多设计时钟，同时提升性能和时钟的功耗特性。即根据驱动程序和使用情况，下面是主要的时钟类型和相关的时钟结构：

- 高速 I/O 时钟

  这些时钟与由 PLL 生成的高速 SelectIO™ 接口位片段逻辑相关联，并通过专用的低抖动资源布线到高速 I/O 接口的位片段逻辑。通常，此时钟结构由赛灵思 IP 创建和控制，例如存储器 IP 或高速选择性向导，并且这些不由用户指定。

- 普通时钟

  这些时钟用于大多数时钟树结构，并且可以由 GCIO 封装引脚，MMCM/PLL 或结构逻辑单元（通常不建议）提供。通用时钟网络必须由 BUFGCE/BUFGCE_DIV/BUFGCTRL 缓存驱动，这些缓存可在包含 I/O 列的任何时钟域中使用。任何给时序钟域都能支持多达 24 个独特时钟，大多数 UltraScale 器件能支持 100 多个时钟树，具体取决于拓扑结构扇出和加载布局。

- 千兆位收发器 (GT) 时钟

  千兆位收发器 （GTH 或 GTY）的发送，在包含 GT 的时钟域中接收和参考时钟使用专用时钟。您可以使用 GT 时钟来实现以下功能：

  - 使用 BUFG_GT 缓存驱动通用时钟网络，以连接光纤网中的任何负载
  - 在相同或不同 Quad 中的多个收发器上共享时钟

时钟原语

大多数时钟通过具有全局时钟能力的 I/O (GCIO) 引脚进入器件。这些时钟通过时钟缓冲器直接驱动时钟网络，或者由位于与 I/O 列相邻的时钟管理片 (CMT) 中的 PLL 或 MMCM 转换。

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：CMT 中的时钟资源与具有未绑定 I/O 的相邻的 I/O 列可供使用。

GT 用户时钟通过 BUFG_GT 缓存驱动全局时钟网络。与 GTH/GTY 列相邻的每个时钟域有 24 个 BUFG_GT 缓存。

以下是每个 UltraScale 器件时钟缓冲器的摘要信息：

- BUFGCE

  最常用的缓存是 BUFGCE。这是具有时钟使能/禁用特性的通用时钟缓冲器，相当于 7 系列 BUFGCE。

- BUFGCE_DIV

  在需要简单时钟分频时，该 BUFGCE_DIV 较为有用。这种缓存对简单时钟分频而言，比 MMCM 或 PLL 更简单易用且能效更高。如果使用恰当，相比 MMCM 或 PLL 而言，在跨越多个时钟域时，时钟域间的偏差更低。BUFGCE_DIV 通常用于替代 7 系列器件中的 BUFR 功能。然而，因为 BUFGCE_DIV 可以驱动全局时钟网络，所以它比 BUFR 组件更加强大。

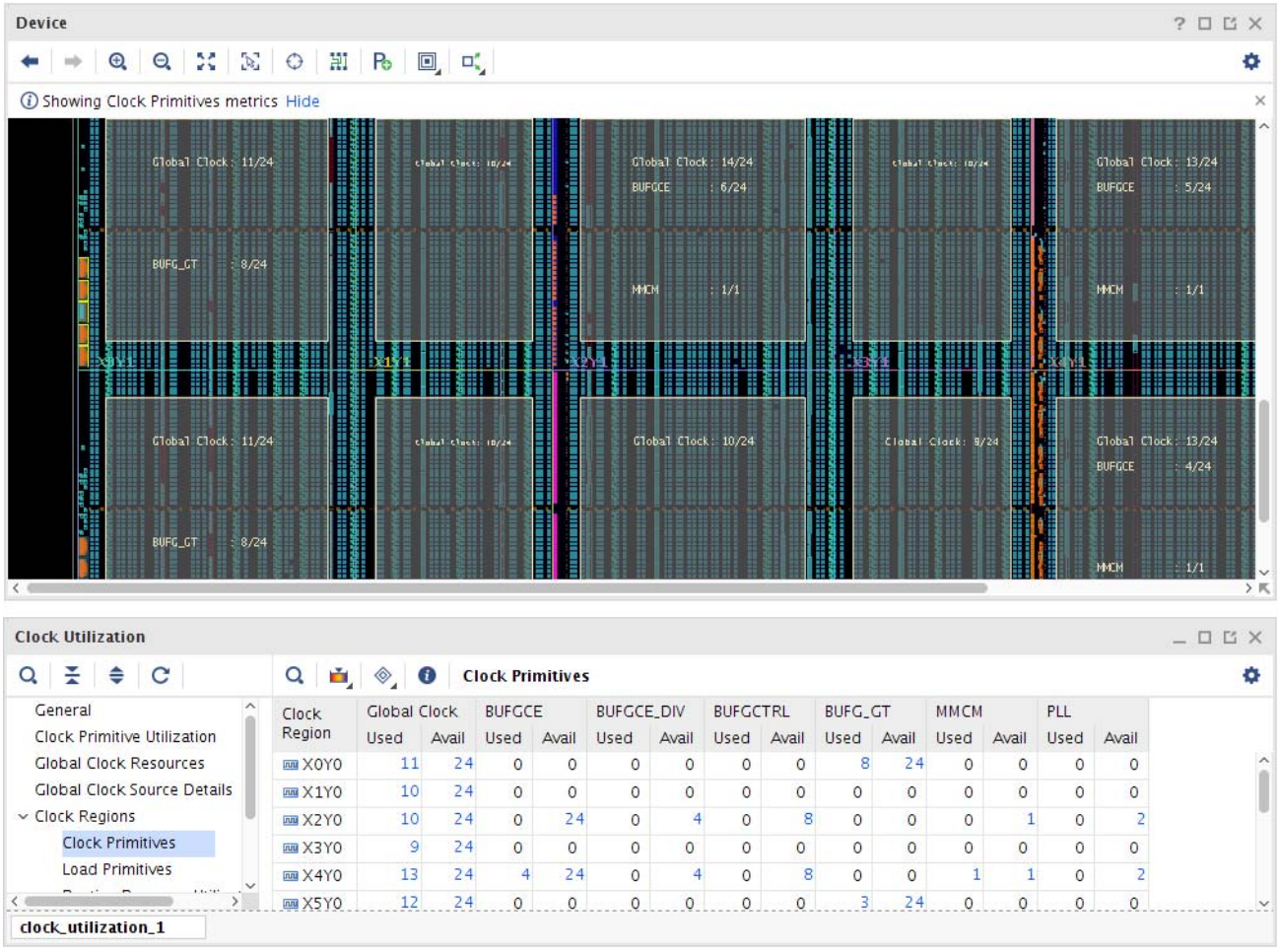
- BUFGCTRL （和 BUFGMUX）

BUFGCTRL 可例化为 BUFGMUX，通常用于将两个或更多时钟源多路复用到单个时钟网络。与 BUFGCE 和 BUFGCE_DIV 一样，它也能驱动时钟网络满足区域或全局时钟。

- BUFG_GT

  使用由 GT 生成的或为 GT 生成的时钟时，使用 BUFG_GT 时钟缓冲器能连接到时钟网络。在大多数情况下，BUFG_GT 用作区域缓存，其负载布局在一个或两个相邻时钟域中。 BUFG_GT 内置动态时钟分频功能，可替代 MMCM 满足时钟速率变化要求。

您可以使用 Vivado IDE 中的“Clock Utilization Report”来进行可视化分析时钟资源利用率和时钟布局。下图显示了在“Device”窗口中叠加的每个时钟域的时钟资源利用率。如需了解更多有关此报告的信息，请参阅《Vivado Design Suite 用户指南：设计分析和收敛技术》(UG906) [参照 21]。



Clock Region	Global Clock		BUFGCE		BUFGCE_DIV		BUFGCTRL		BUFG_GT		MMCM		PLL	
	Used	Avail	Used	Avail	Used	Avail	Used	Avail	Used	Avail	Used	Avail	Used	Avail
X0Y0	11	24	0	0	0	0	0	0	8	24	0	0	0	0
X1Y0	10	24	0	0	0	0	0	0	0	0	0	0	0	0
X2Y0	10	24	0	24	0	4	0	8	0	0	0	1	0	2
X3Y0	9	24	0	0	0	0	0	0	0	0	0	0	0	0
X4Y0	13	24	4	24	0	4	0	8	0	0	1	1	0	2
X5Y0	12	24	0	0	0	0	0	0	3	24	0	0	0	0

图 3-30：时钟利用报告

如需了解更多有关 BUFGCE、 BUFGCE_DIV、和 BUFGCTRL 缓存的信息，请参阅 《UltraScale 架构时钟资源用户指南》(UG572) [参照 40]。如需了解更多有关 BUFG_GT 缓存连接和使用的信息，请参阅 《UltraScale 架构收发器用户指南》[参照 41]。

全局时钟缓冲器连接和布局跟踪

每个时钟域中的 24 个 BUFGCE 缓存只能驱动特定的时钟布线。然而， BUFGCTRL 和 BUFGCE_DIV 输出可以通过 MUX 结构使用 24 个布线中的任何一个。每个 BUFGCE_DIV 与特定的 BUFGCE 位置共享输入连接，每个 BUFGCTRL 与两个特定的 BUFGCE 位置共享输入连接。因此，当在时钟域中使用 BUFGCE_DIV 或 BUFGCTRL 缓存时， BUFGCE 缓存的使用将会受到限制。下图显示在时钟域内复制 4 次的时钟域底部的 6 个 BUFGCE。

注释：对于时钟使用的所有垂直，水平布线和分配资源，全局时钟网络被分配给器件中的特定轨道 ID。时钟无法更改布线 ID，除非时钟通过另一个时钟缓冲器。

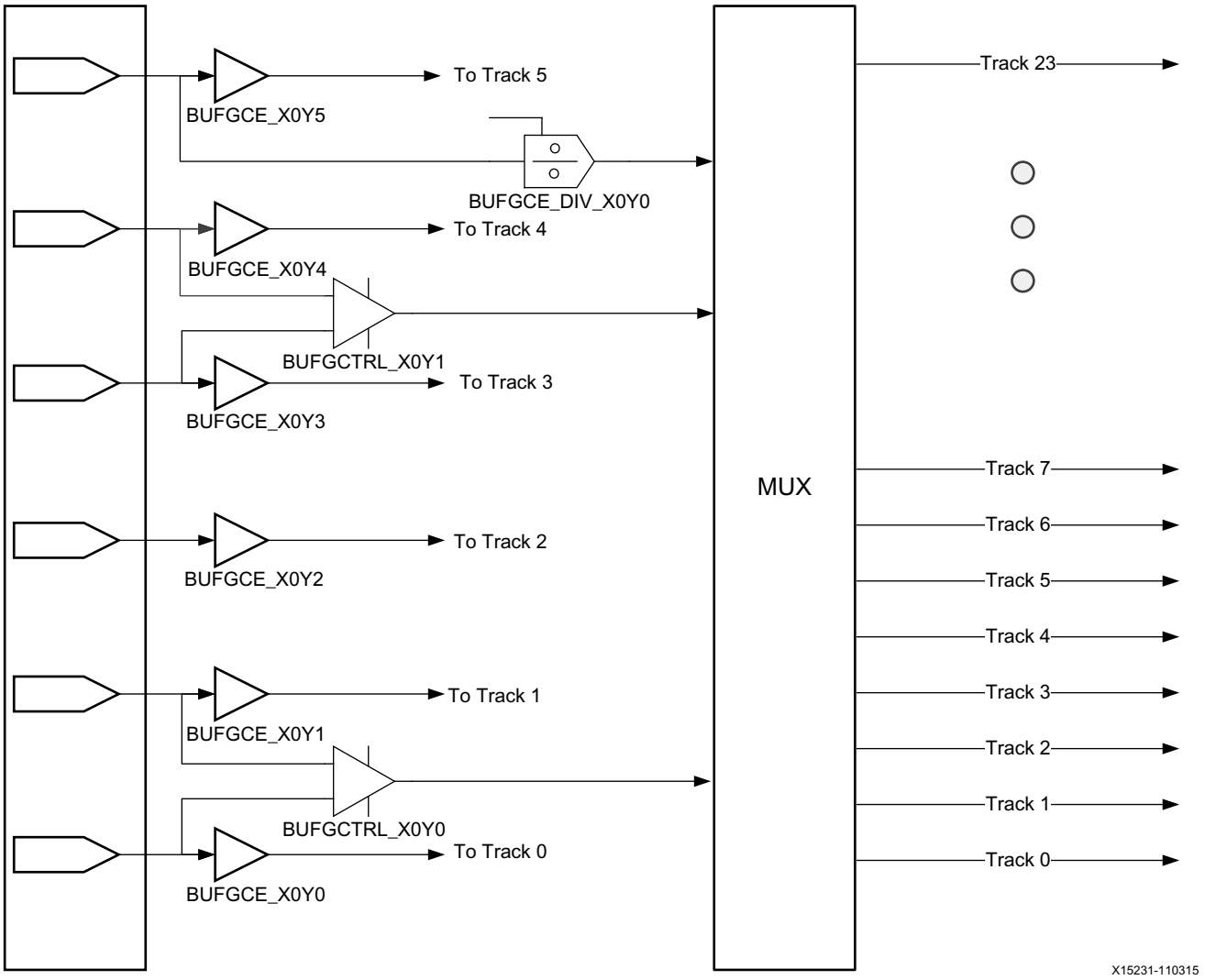


图 3-31：BUFGCE、 BUFGCE_DIV 和 BUFGCTRL 共享输入和复用输出

时钟布线、根和分配

要正确理解 UltraScale 器件的时钟容量和设计的时钟利用率，首先了解时钟布局如何使用专用的布局资源非常关键：

- 从时钟缓冲器到时钟根，时钟信号经过垂直和水平布局的一个或几个阶段。每个细分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到具有相同轨道 ID 的分配轨道。为了减少偏差，时钟根通常在位于时钟窗口中心的时钟域中。时钟窗口是包括布局时钟网络负载的所有时钟域的矩形区域。为了偏差优化， Vivado IDE 可能会将时钟根偏移到偏离中心。
- 从时钟根到负载所在的 CLB 列，时钟信号在垂直分布 （根据需要在器件上下）上行进，然后到水平分布 （根据需要向左和向右）。
- CLB 列分成两半，分别位于水平分配资源的上方和下方。 CLB 列的每一半包含几个叶时钟布线资源，可以通过任何水平分配轨道到达。

在某些情况下，时钟缓冲器可以直接驱动到时钟分配轨道上。这通常发生在时钟根位于与时钟缓冲器相同的时钟域中时，或者当时钟缓冲器仅驱动非时钟引脚 （例如，高扇出网）时。

因为时钟布线资源是分段的，所以仅消耗用于穿过时钟域或到达时钟域中的负载的布线和分配段。

下图显示位于时钟域 X2Y1 中的时钟缓冲器如何到达其位于时钟窗口内的负载，该时钟窗口由 X1Y3 至 X5Y5 的时钟域的矩形形成。

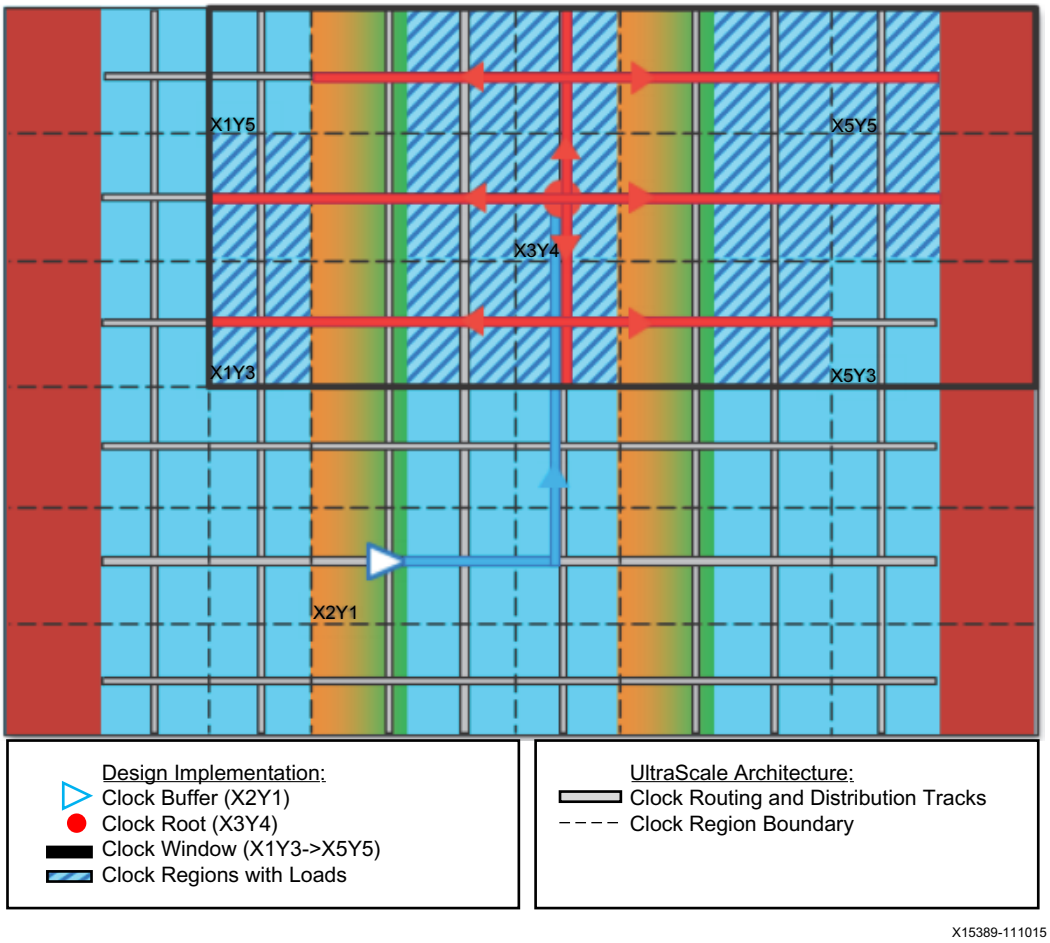


图 3-32：UltraScale 器件时钟从驱动器到负载布线

在下图中，布线器件视图显示了跨越器件大部分的全局时钟。驱动网络的时钟缓冲器在时钟域 X2Y0 中以蓝色突出显示，并且驱动到该时钟域中的水平布线。然后，网络在时钟域 X2Y0 中从水平布线转换到垂直布线，到达时钟域 X2Y5 中的时钟根。所有时钟布线以蓝色突出显示。时钟根在时钟域 X2Y5 中以红色突出显示。从 X2Y5 中的时钟根，网络转换到垂直分布，然后到时钟叶针的水平分布。 CLB 列中的分布层和叶子时钟布线资源以红色突出显示。

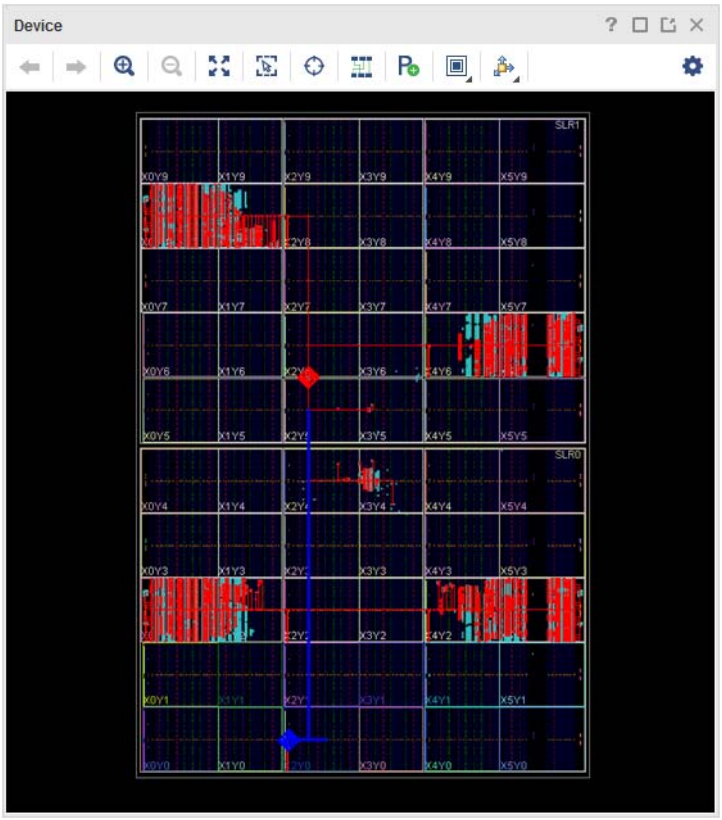


图 3-33：布线器件视图的布线时钟网络

时钟树缓存布局布线

在以下阶段， Vivado 布局器确定 MMCM/PLL，全局时钟缓冲器和时钟根的位置，同时遵守物理 XDC 约束：

1. I/O 和时钟布局

   布局器根据连接规则和用户约束布局 I/O 缓存和 MMCM/PLL。布局器将时钟缓冲器分配给时钟域，但不分配给单个位置，除非使用 LOC 属性进行约束。详情，请参阅表 3-2。只有仅驱动非时钟负载的时钟缓冲器可以基于它们的驱动器和负载的布局移动到该流程中稍后的不同时钟域。

   在此阶段的任何安装器错误是由于冲突的连接规则，用户约束，或两者。日志文件显示有关错误的可能的根本原因的详细信息，您必须仔细查看，以使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

   布局器基于早期驱动器和负载布局执行初始时钟树实现。每个时钟网络与时钟窗口相关联。时钟窗口的过度重叠可能由于预期的时钟布线争用而导致布局器错误。

   当发生时钟分区错误时，日志文件显示每个时钟网络的最后时钟预算解决方案以及每个时钟域中存在的唯一时钟网络的数量。详细查看日志文件以确定从过度使用的时钟域中删除哪些时钟。您可以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。
   - 将时钟基元移动到不同的时钟域，特别是那些没有基于连接的布局规则的时钟基元。
   - 在时钟负载上添加布局规划约束，以使具有较小扇出的时钟更接近其驱动器或远离高利用率时钟域。

   布局器精化时钟树实现几次以帮助提高时序 QoR。例如，在稍后的布局优化阶段期间，布局器分析每个挑战性时钟以确定更好的时钟根位置。

3. 时钟树预布线

   布局器指导后续实现步骤，并为布局后时序分析提供准确的延迟估计。

布局后， Vivado 工具可以按如下所示修改时钟树实现：

- Vivado 物理优化器可以复制和移动单元到没有相关时钟的时钟域。
- 可以进行 Vivado 布线器调整，以提高时序 QoR 和合法化时钟布线。当您使用 `Explore` 布线指令时， Vivado 布线器还可以修改时钟根位置以提高时序 QoR。

下表总结了主时钟拓扑结构的布局规则以及约束如何影响这些规则。

表 3-2：具备和不不被布局规则的拓扑结构

约束源	无约束目的地	行为
GCIO	BUFGCE、 BUFGCTRL、 BUFGCE_DIV、 PLL/MMCM	自动布局在相同的时钟域。
PLL/MMCM	BUFGCE、 BUFGCTRL、 BUFGCE_DIV	自动布局在相同的时钟域。
GT*_CHANNEL	BUFG_GT	自动布局在相同的时钟域。
BUFGCTRL	BUFGCTRL	自动布局在相同的时钟域。 **注释：** 您可以使用 CLOCK_REGION 约束覆盖同一时钟域中的布局。
BUFG*	BUFG*	不可预测的无约束目标 BUFG 的布局。 使用 CLOCK_REGION 约束推荐约束目标 BUFG*。 **注释：** 这不包括 BUFGCTRL - > BUFGCTRL。
BUFG*	MMCM/PLL	不可预测的无约束目的地 MMCM/PLL 的布局。 推荐使用 LOC 约束约束 MMCM/PLL。 当布线跨越相邻或多个时钟域时，推荐 CLOCK_DEDICATED_ROUTE 约束。

时钟功能

时钟规划必须基于目标器件中的高扇出时钟和低扇出时钟的总数。

高扇出时钟

高扇出时钟几乎覆盖了 SSI 技术器件的整个 SLR 或单片器件的几乎所有时钟域。下图显示了一个高扇出时钟，它跨越几乎整个 SLR，而 BUFGCE 驱动器显示为红色。

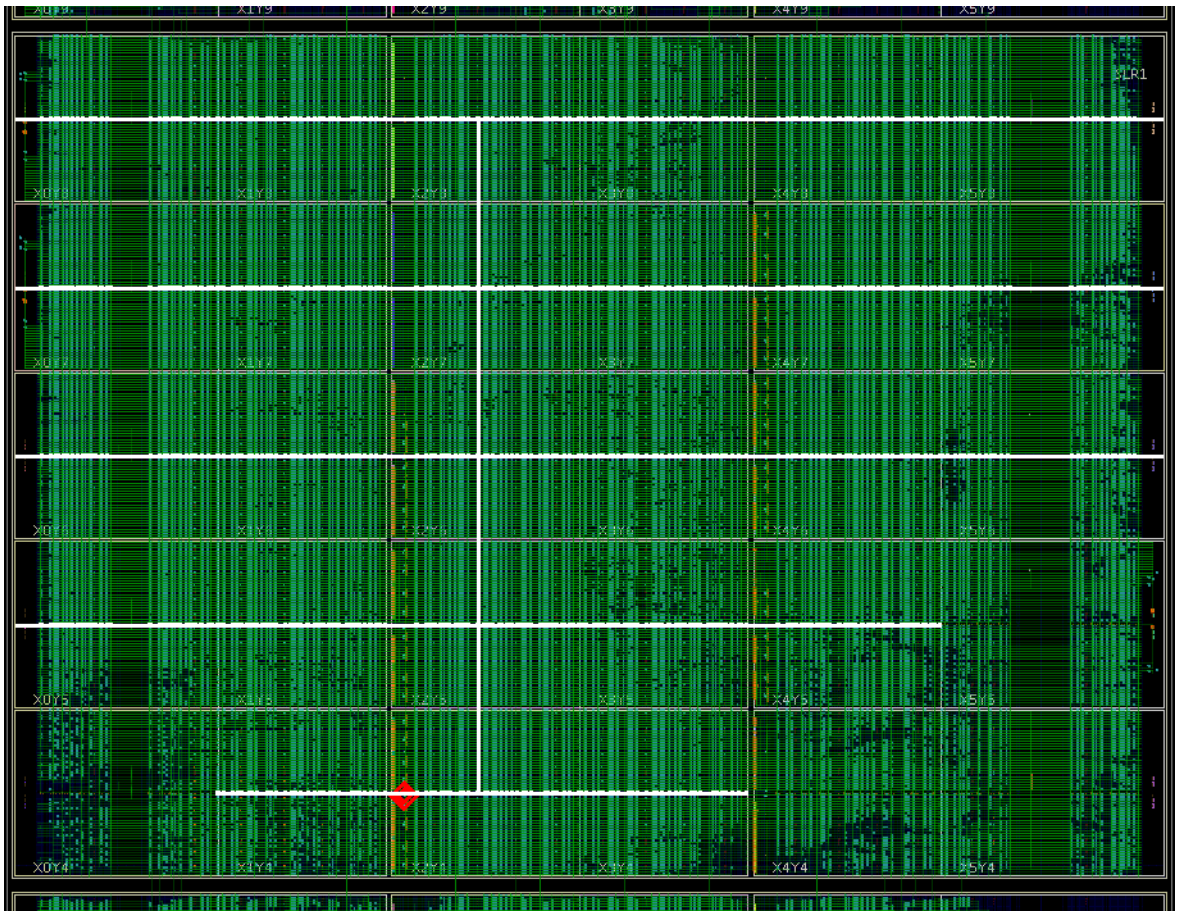


图 3-34：跨越 SLR 的高扇出时钟

注释：在设计中使用超过 24 个时钟可能会导致需要特殊设计考虑或其他前期规划的问题。

重要提示：在 ZHOLD 和 BUF_IN 补偿模式下， MMCM 反馈时钟路径在布线轨道，时钟根位置和分布轨道方面与 CLKOUT0 时钟路径相匹配。因此，当时钟缓冲器和时钟根相距很远时，反馈时钟可以视为高扇出时钟。如需了解更多有关 MMCM 补偿机制的信息，请参阅“使用 MMCM 的 I/O 时序 ZHOLD/BUF_IN 补偿”。

低扇出时钟

在大多数情况下，低扇出时钟是连接少于 5,000 个时钟引脚的时钟网络，它们被布局在 3 个或更少的水平相邻的时钟域中。时钟布线，时钟根和时钟分布都包含在局部区域内。在一些情况下，期望布局器识别低扇出时钟但失败。这可能是由设计尺寸，器件尺寸或物理 XDC 约束引起的，例如 LOC 约束或 Pblock，其阻止布局器将负载布局在局部区域中。要解决此问题，您可能需要通过手动创建 Pblock 或修改现有物理约束来指导工具。

由 BUFG_GT 驱动的时钟是低扇出时钟的示例。 Vivado 布局器自动识别这些时钟网络，并包含与 GT 接口相邻的时钟域的负载。下图显示了两个时钟域中包含的低扇出时钟， BUFG_GT 驱动器显示为红色。

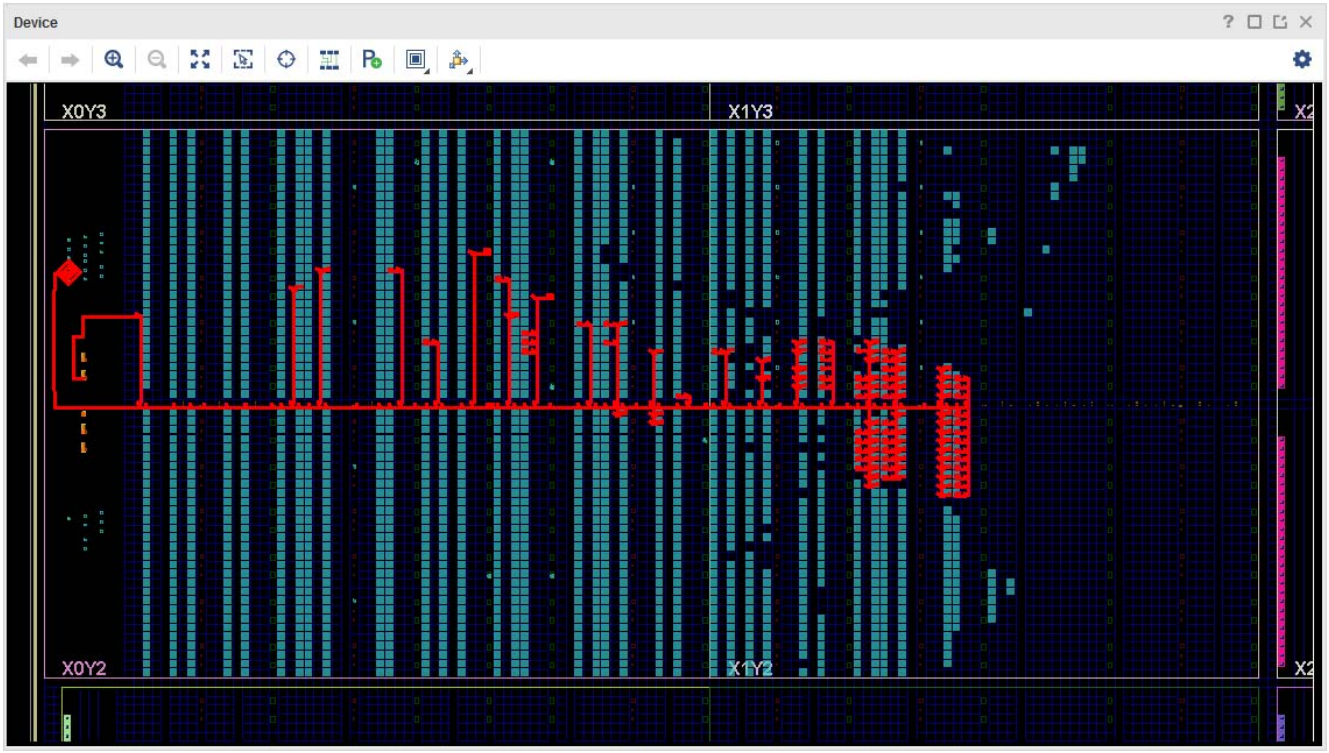


图 3-35：两个时钟域包含的低扇出时钟

平衡利用高和低扇出时钟

UltraScale 器件比以前的赛灵思 FPGA 系列支持更多的时钟。这实现了各种各样的时钟利用场景，例如：

- 24 个时钟或更低

  除非存在冲突的用户约束，否则所有时钟都可以被视为高扇出时钟，而不会存在布局或布线争用的风险。

- 接近 300 个时钟

  对于针对具有 6 个时钟域行的器件并且仅包括低扇出时钟 （每个时钟最多包含在 3 个时钟域）的设计，需要以下时钟：每行 6 行 × 2 个时钟窗口 × 每个区域 24 个时钟 = 288 个时钟。

低扇出时钟窗口不具有固定大小，但通常在 1 到 3 个时钟域之间。高扇出时钟很少跨越整个器件或整个 SLR。

以下方法显示了如何平衡高扇出时钟和低扇出时钟，假设几个低扇出时钟来自 I/O 接口，大多数来自 GT 接口。您可以对每个 SSI 技术器件的 SLR 应用相同的方法。

- 高扇出时钟
  - 最多 12 个单片器件
  - 对于 SSI 技术器件，最多为 24 个 （假设某些高扇出时钟仅存在于 1 个 SLR 中）
- 低扇出时钟
  - 高达 12 加 8 每 GT 使用组
  - 或者，每个 GT 接口多达 12 加 6 （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 器件时钟比采用以前架构的时钟更灵活，并且包括额外的体系结构约束，因此了解如何正确约束时钟以实现是非常重要的。

使用 LOC 约束 IO/MMCM/PLL/GT

要约束时钟，您可以按如下所示分配布局约束：

- 在 I/O 端口的时钟输入

  为 GCIO 上的时钟分配 PACKAGE_PIN 约束或为 IOB 分配 LOC 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的时钟域中。

- 在 MMCM 或 PLL 上

  直接连接到 MMCM 或 PLL 输出的时钟缓冲器，连接到 MMCM 或 PLL 输入的输入时钟端口自动布局在相同的时钟域中。如果输入时钟端口和 MMCM 或 PLL 直接连接并限制在不同的时钟域，则必须手动插入时钟缓冲器并在连接到 MMCM 或 PLL 的网络上设置 CLOCK_DEDICATED_ROUTE 约束。

- 在 GT*_CHANNEL 或 IBUFDS_GTE3 单元上

  由单元驱动的 BUFG_GT 布局在相同的时钟域中。

注意！赛灵思不推荐在时钟缓冲器单元上使用 LOC 约束。此方法将时钟强制到特定轨道 ID，这可能导致无法合法布线的布局。当您明确设计的整个时钟树并且设计中的布局一致时，只能使用 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，在实现期间可能会发生冲突。

在时钟缓冲器上使用 CLOCK_REGION 属性

您可以使用 CLOCK_REGION 约束为时钟域分配时钟缓冲器，而不指定位置。这在优化所有时钟树以及确定适当的缓冲位置以成功布线所有时钟时，将赋予布局器更大的灵活性。

您还可以使用 CLOCK_REGION 约束来提供关于级联时钟缓冲器或由非时钟原语 （例如，构造逻辑）驱动的时钟缓冲器的布局的指导。

在以下示例中， XDC 约束将 `clkgen/clkout2_buf` 时钟缓冲器分配给 `CLOCK_REGION X2Y2`。

```
set_property CLOCK_REGION X2Y2 [get_cells clkgen/clkout2_buf]
```

注释：在大多数情况下，时钟缓冲器由已经约束到时钟域的输入时钟端口，MMCM，PLL 或 GT*_CHANNEL 直接驱动。在这种情况下，则时钟缓冲器将自动布局在同一时钟域中，并且您不需要使用 CLOCK_REGION 约束。

使用 Pblock 限制时钟缓冲器布局

当不需要将时钟缓冲器布局在特时序钟区域中时，可以使用 Pblock 指时序钟区域的范围。例如，当需要 BUFGCTRL 来复用位于不同区域的两个时钟时，使用 Pblock。您可以将 BUFGTRL 分配给包含两个时钟驱动程序之间的时钟域的 Pblock，并让布局器识别有效的布局。

注释：赛灵思不推荐在单个时钟域使用 Pblock。

在时钟网络上使用 USER_CLOCK_ROOT 属性

您可以使用 USER_CLOCK_ROOT 属性，来强制限时序钟缓冲器驱动的时钟根位置。指定 USER_CLOCK_ROOT 属性会影响设计布局，因为它通过修改时钟布线来影响插入延迟和偏移。 USER_CLOCK_ROOT 值对应时钟域，并且必须在由高扇出时钟缓冲器直接驱动的网段上设置属性。下面给出一个实例：

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

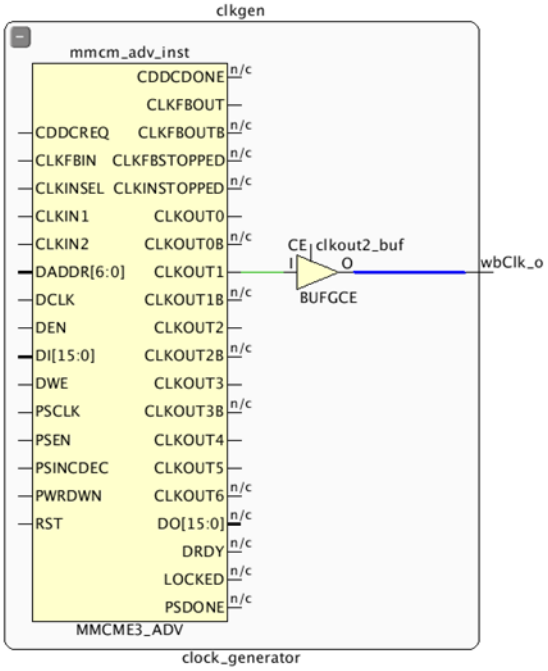


图 3-36：由时钟缓冲器驱动的网络段上应用 USER_CLOCK_ROOT

如下例所示，布局后，您可以使用 LOCK_ROOT 性查询实际的时钟根。 CLOCK_ROOT 将报告分配根，无论是由用户分配还是由 Vivado 工具自动分配。

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

另一种查看已实现设计的时钟根分配的方法是使用 `report_clock_utilization` Tcl 命令。例如：

```
report_clock_utilization [-clock_roots_only]
```

下图显示了此报告。

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

图 3-37：Report_clock_utilization 时钟根分配

在多个时钟网络上使用 CLOCK_DELAY_GROUP 约束

您可以使用 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。此约束通常用于最小化源自同一 MMCM，或 PLL 的时钟之间的同步 CDC 时序路径上的偏移。您必须在与时钟缓冲器直接联通网段上设置 CLOCK_DELAY_GROUP 约束。下面的示例显示了 `clk1_net` 和 `clk2_net` 时钟网络，这些网络由时钟缓冲器直接驱动：

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

有关在时钟之间的路径上使用此约束的详细信息，请参阅“同步 CDC”。

使用 CLOCK_DEDICATED_ROUTE 约束

CLOCK_DEDICATED_ROUTE 约束通常在从一个时钟域中的时钟缓冲器驱动到另一个时钟域中的 MMCM 或 PLL 时使用。默认情况下，CLOCK_DEDICATED_ROUTE 约束设置为 TRUE，并且缓存/MMCM 或 PLL 对必须布局在相同的时钟域中。

下表是对不同 CLOCK_DEDICATED_ROUTE 约束值、使用和行为的总结。

表 3-3：UltraScale 器件 CLOCK_DEDICATED_ROUTE 约束总结

值	使用	行为
TRUE	时钟网络上的默省值	• 全局时钟缓冲器和 MMCM/PLL 必须位于相同的 CLOCK_REGION。 • 该值确保网络布线只使用全局时钟资源。
SAME_CMT_COLUMN	全局时钟缓冲器驱动的网络或 IBUF 的输出 例如： set_property CLOCK_DEDICATED_ROUTE [get_nets -of [get_pins BUFGCE_inst/O]] set_property CLOCK_DEDICATED_ROUTE [get_nets -of [get_pins IBUF_inst/O]]	• MMCM/PLL 必须居于相同垂直列中的 CLOCK_REGION 中。 • 该值可确保网络布线只使用全局时钟资源。 • 为得到理想结果，赛灵思建议在 MMCM/PLL 上使用 LOC 约束，以将 MMCM/PLL 的布局控制在同一垂直列中。
ANY_CMT_COLUMN	全局时钟缓冲器驱动的网络 例如： set_property CLOCK_DEDICATED_ROUTE [get_nets -of [get_pins BUFGCE_inst/O]] set_property CLOCK_DEDICATED_ROUTE [get_nets -of [get_pins BUFGCE_DIV_inst/O]] set_property CLOCK_DEDICATED_ROUTE [get_nets -of [get_pins BUFGCTRL_inst/O]]	• MMCM/PLL 可布局在任何有可用资源的 CLOCK_REGION 中。 • 该值确保网络布线仅使用时钟资源。 • 为得到理想结果，赛灵思建议 在 MMCM/PLL 上使用 LOC 约束，以将 MMCM/PLL 的布局控制在同一垂直列中。
FALSE	除部分时钟网络外，时钟网络不受全局时钟缓冲器的驱动 （例如受 IBUF 输出驱动的网络或与 MMCM 的输出时钟引脚直接连接的网络） 例如： set_property CLOCK_DEDICATED_ROUTE [get_nets -of [get_pins MMCME4_ADV_inst/CLKOUT0]] set_property CLOCK_DEDICATED_ROUTE [get_nets -of [get_pins IBUF_inst/O]]	• 使用架构和全局时钟资源布线的网络。 • 这样会对时钟网络的时序和性能造成负面影响。 重要：于 UltraScale 器件，只有在出于特殊设计原因，使用全局时钟资源正常布线的时钟需要与结构资源进行布线时，才应使用 CLOCK_DEDICATED_ROUTE=FALSE。

注释：在使用 UltraScale 器件工作时，*勿*在端口直接驱动的网络上应用 CLOCK_DEDICATED_ROUTE 属性。相反，将 CLOCK_DEDICATED_ROUTE 属性应用到 IBUF 的输出。

当从一个时钟域中的时钟缓冲器驱动垂直相邻时钟域中的 MMCM 或 PLL 时，必须将 7 系列器件的 CLOCK_DEDICATED_ROUTE 设置为 BACKBONE 或 UltraScale 器件的 CLOCK_DEDICATED_ROUTE 设置为 SAME_CMT_COLUMN。这可以防止实现错误，并确保时钟仅使用全局时钟资源进行布线。以下示例显示了在垂直相邻时钟域中驱动两个 PLL 的时钟缓冲器。

```
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]
```

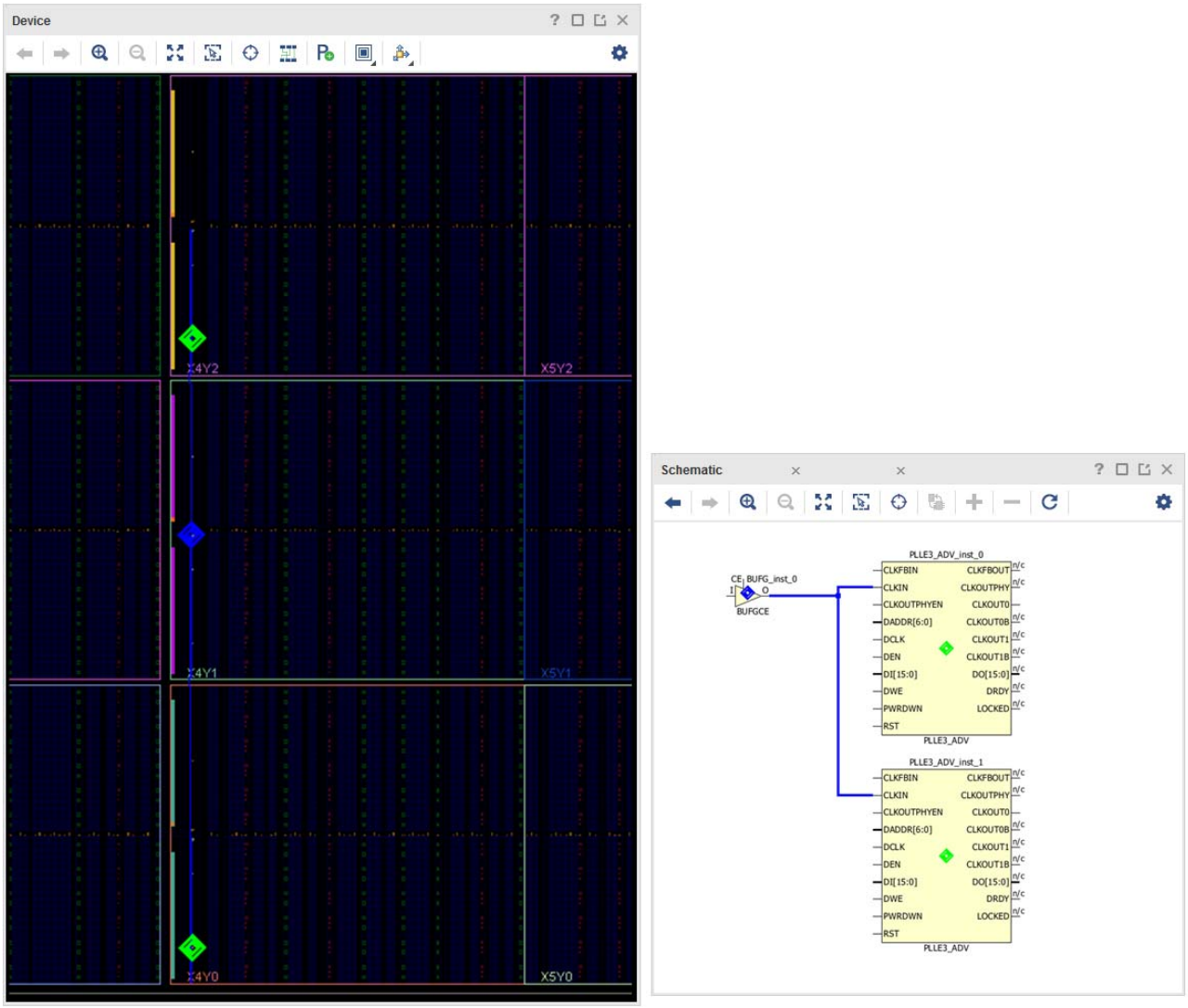


图 3-38：CLOCK_DEDICATED_ROUTE 约束设置为 SAME_CMT_COLUMN

当从时钟缓冲器驱动到不垂直相邻的其他时钟域时，必须将 7 系列器件的 CLOCK_DEDICATED_ROUTE 设置为 FALSE 或 UltraScale 器件的 ANY_CMT_COLUMN 设置为 CLOCK_DEDICATED_ROUTE。这可以防止实现错误，并确保时钟仅使用全局时钟资源进行布线。以下示例和图显示了驱动与输入缓存不在同一时钟域列上的两个 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_X0Y0 [get_cells PLLE3_ADV_inst_0]
set_property LOC PLLE3_ADV_X0Y4 [get_cells PLLE3_ADV_inst_1]
```

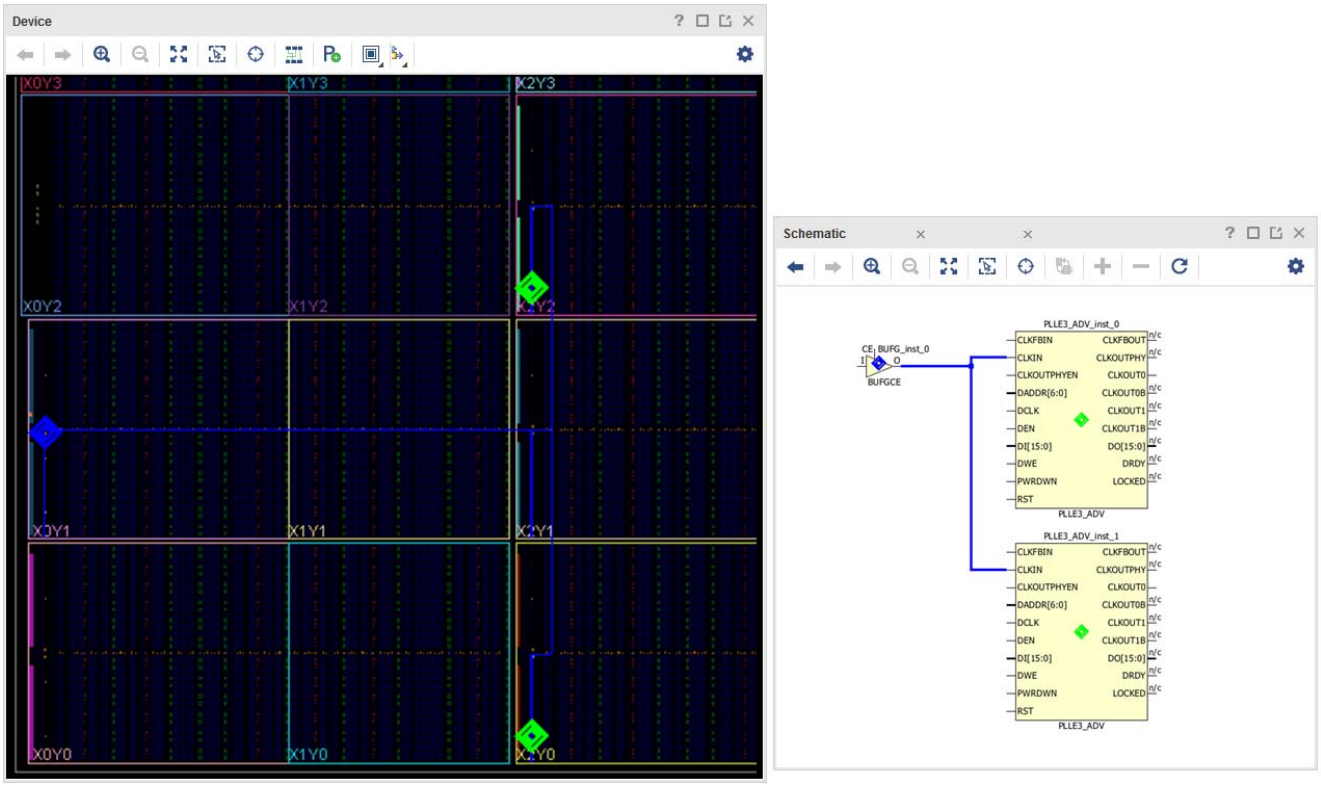


图 3-39：CLOCK_DEDICATED_ROUTE 设置为 ANY_CMT_COLUMN

时钟拓扑结构建议

赛灵思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，这可能导致在时钟布线要求高并且接近最大容量的时钟域中的布局错误或布线冲突。

以下是针对 BUFGCE/BUFGCTRL/BUFGCE_DIV 连接的时钟拓扑结构建议。

并行时钟缓冲器

使用并行时钟缓冲器来实现以下目的：

- 确保跨实现运行的可预测的展示位置

  当并行时钟缓冲器由相同的输入时钟端口 MMCM， PLL 或 GT*_CHANNEL 直接驱动时，无论网表变化或逻辑布局变化如何，缓存始终置于与其驱动器相同的时钟域中。

- 匹配时钟树的并行分支之间的插入延迟

  赛灵思推荐并行缓存通过级联时钟缓冲器，特别是在分支之间存在同步路径时。当使用级联缓存时，即使使用 CLOCK_DELAY_GROUP 或 USER_CLOCK_ROOT 约束，时钟插入延迟在时钟树的分支之间将不匹配。这可能导致高时钟偏移，将导致时序收敛难度增加，甚至无法实现。

下图显示了由 MMCM CLKOUT0 端口驱动的三个并行 BUFGCE 缓存。

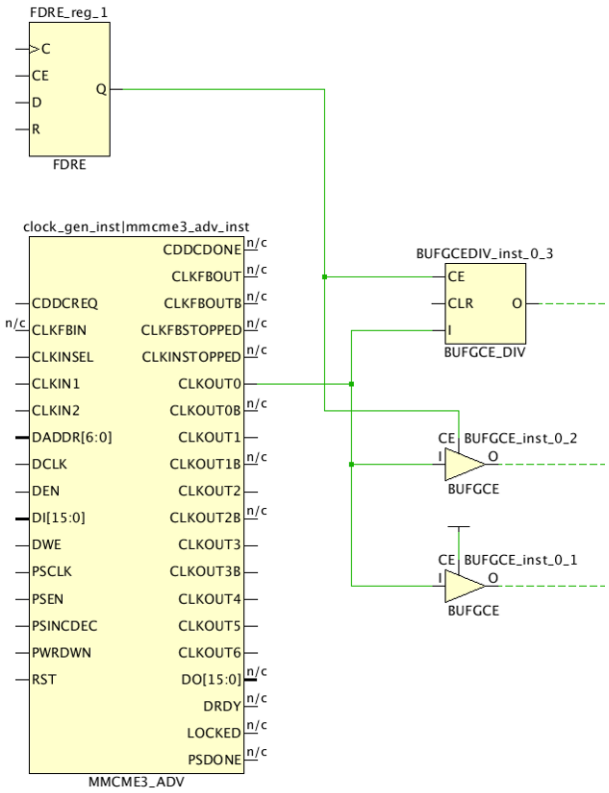


图 3-40：MMCM 输出上的并行 BUFGCE

级联时钟缓冲器

一般来说，赛灵思不建议使用级联缓存人为地增加延迟并减少不相关的时钟树分支之间的偏差。与 BUFGCTRL 之间的连接不同，其他时钟缓冲器连接在架构中没有专用路径。因此，时钟缓冲器的相对布局是不可预测的，并且所有布局规则优先于布局无约束级联缓存。

但是，可以使用级联时钟缓冲器来实现以下功能：

- 将时钟布线到位于不同时钟域中的另一个时钟缓冲器

当将时钟多路复用器用于由位于不同时钟域中的 MMCM 生成的时钟时，该方法较为普遍。虽然一个 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。下图显示了一个示例。

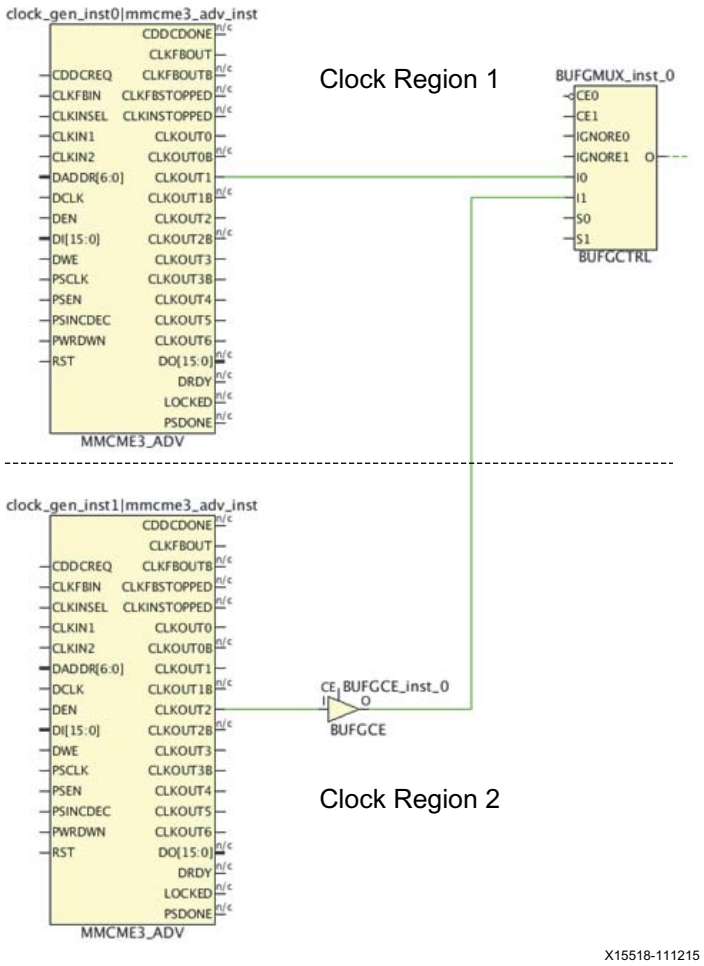


图 3-41：将时钟布线到另一个时钟域

- 在这些分支之间存在同步路径时，平衡时钟树分支上的时钟缓冲器级别数

  例如，一个名为 clk0 的 MMCM 时钟驱动组 A（通过位于不同时钟域中的 BUFGCTRL 驱动的顺序单元）和组 B（顺序单元）。为了更好地匹配分支之间的延迟，为组 B 插入一个 BUFGCE，并将它布局在与 BUFGCTRL 相同的时钟域中。这确保组 A 和组 B 之间的同步路径具有受控的偏差量。下图显示了一个示例。

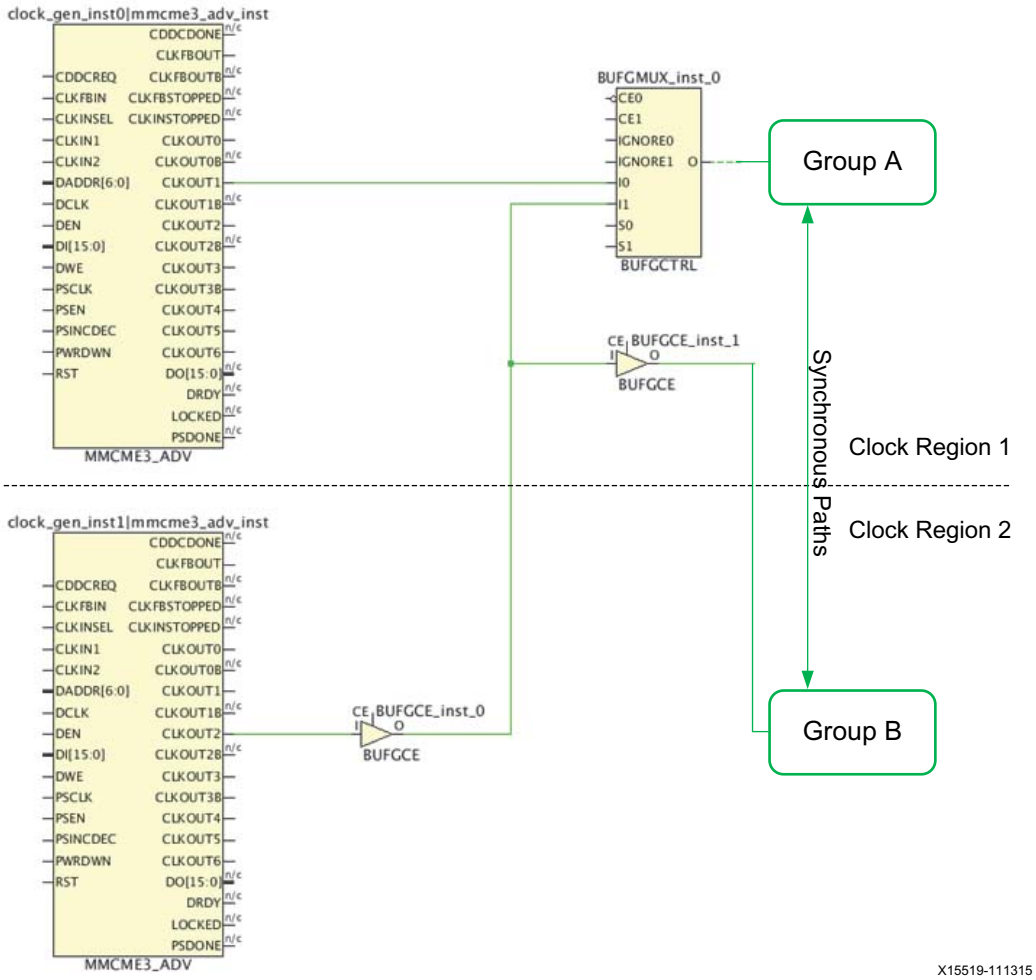


图 3-42：平衡时钟域之间的同步路径的时钟树

注释：如果在时钟树分支之间仅存在异步路径，则只要在接收时钟域上存在适当的同步电路，分支就不需要被平衡。

- 如“时钟多路复用”所述，构建时钟多路复用器。

当使用级联时钟缓冲器时，为了减少插入延迟和偏移的变化，赛灵思建议如下：

- 将级联缓存保持在相同或相邻的时钟域中。
- 当时钟树分支平衡时，将相同级别的所有时钟缓冲器分配给相同的时钟域。

注释：如果绝对需要，赛灵思建议使用两个级联 BUFGCTRL 而不是级联 BUFGCE。使用专用布线，当两个 BUFGCTRL 位于相同的时钟域内时，可以以最小的延迟级联两个相邻的 BUFGCTRL。

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位置可用性查找最佳布局。如果可能，布局器将 BUFGCTRL 布局在相邻位置中以利用专用级联路径。如无法实现，则布局器将尝试将 BUFGCTRL 从相邻时钟域中的相同层布局。

下图显示了具有平衡级联的 4:1 MUX。第一级 BUFGCTRL 缓存都布局在最后一个 BUFGCTRL (X0Y1) 的直接相邻位置 (X0Y2， X0Y0)。此配置确保到达最后一个 BUFGCTRL 的所有时钟的相当的插入延迟。对于 3:1 MUX，可以使用类似结构。

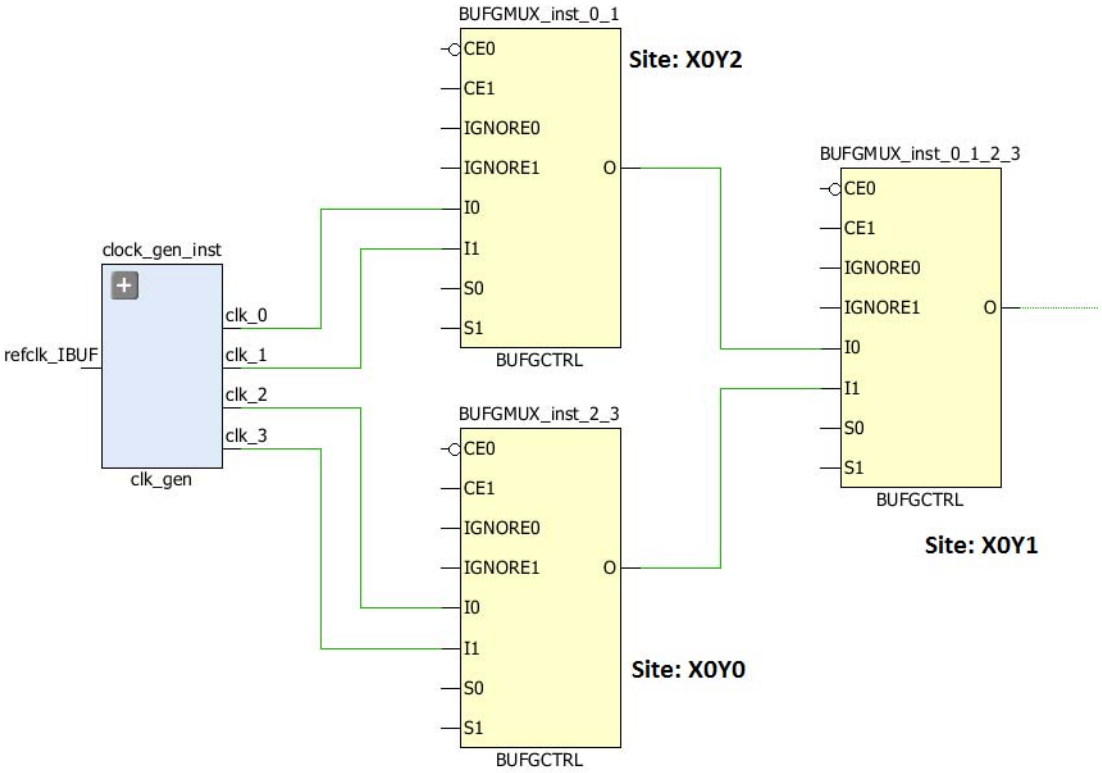


图 3-43：4:1 MUX 使用并行 BUFGCTRL

如下图所示，当创建 5:1 或更大的时钟 MUX 结构时，通常会创建一个对称的时钟结构。然而这并不是最佳解决方案，因为每个 BUFGCTRL 只有一个到两个相邻 BUFGCTRL 的级联路径，这无法为 BUFGCTRL 之间的所有连接提供最小延迟。

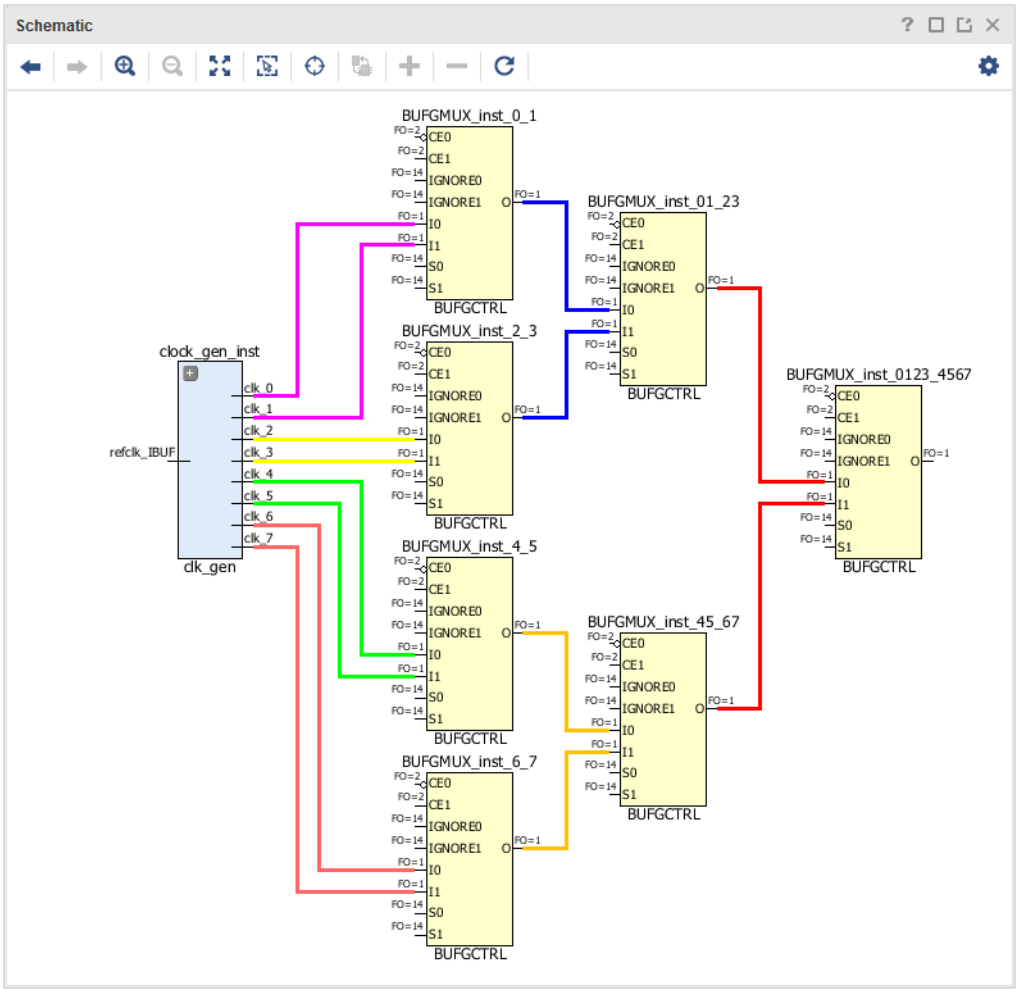


图 3-44：非推荐 8:1 平衡时钟 MUX 结构

如下图所示，为了支持更大的时钟多路复用器（从 5:1 到 8:1 MUX），赛灵思建议使用级联 BUFGCTRL 缓存。此图显示了使用 7 个 BUFGCTRL 缓存的最佳 8:1 MUX。

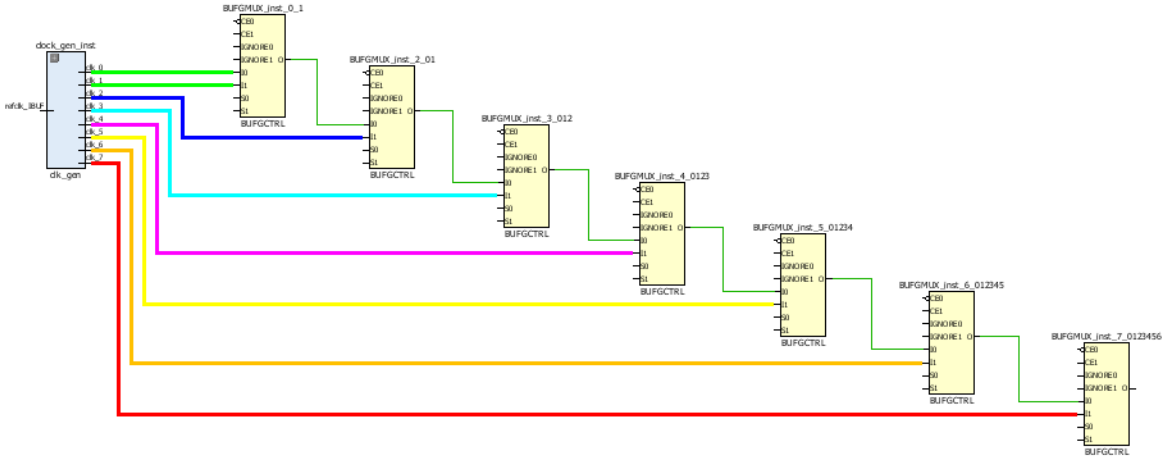


图 3-45：8:1 MUX 使用级联 BUFGCTRL

注释：因为一些路径比硬件中的其它路径长，所以当使用宽的基于 BUFGCTRL 的时钟多路复用器时，时钟插入延迟无法平衡。因此，建议仅对异步时钟进行多路复用。

PLL/MMCM 反馈路径和补偿模式

PLL 不支持延迟补偿，并且始终在内部补偿模式下工作，这意味着它们不需要反馈路径。同样，设置为 INTERNAL 补偿模式的 MMCM 不需要反馈路径。在这两种情况下， Vivado 工具并不总是自动删除不必要的反馈时钟缓冲器。您必须手动删除时钟缓冲器，以减少高扇出时钟资源利用率。这对于可能发生时钟争用的具有高时钟使用的设计尤其重要。

当 MMCM 补偿设置为 ZHOLD 或 BUF_IN 时，布局器为由反馈缓存驱动的网络和直接连接到 CLKOUT0 引脚的所有缓存分配相同的时钟根。这确保插入延迟匹配，使连接到 CLKOUT0 的 I/O 端口和顺序单元相位对齐，并且在器件接口处满足保持时间。 Vivado 工具分析这些网络的所有负载，从而理想化定义时钟根。

Vivado 工具不会自动将插入延迟与其他 MMCM 输出匹配。要匹配由其他 MMCM 输出缓存驱动的网络的插入延迟，请使用以下属性：

- CLOCK_DELAY_GROUP

  将相同的 CLOCK_DELAY_GROUP 属性值应用于由反馈时钟缓冲器，CLKOUT0 缓存和其他 MMCM 输出缓存直接驱动的网络 （如有需要）。这是首选的方法。

- USER_CLOCK_ROOT

  如果需要强制特定的时钟根，请在由反馈时钟缓冲器，CLKOUT0 缓存和其他 MMCM 输出缓存驱动的网络上使用相同的 USER_CLOCK_ROOT 属性值。

BUFG_GT 除法器

BUFG_GT 缓存可以驱动架构中的任何负载，并包括一个可选的除法器，可用于将时钟从 GT*_CHANNEL 分频。这消除了使用额外的 MMCM 或 BUFG_DIV 来划分时钟的需要。

SelectIO 时钟

UltraScale 器件 SelectIO 原语在时钟引脚之间存在最大的偏差要求。将理想时钟拓扑用于 SelectIO 原语，可最大限度地避免突破偏差要求，提升 UltraScale 器件和架构逻辑之间的接口时序，减少时钟资源占用。

ISERDESE3 和 IDDRE1 时钟

关于 UltraScale 器件和 UltraScale+ 器件中的 ISERDESE3 和 IDDRE1 时序，时钟和反转时钟引脚之间存在最大的偏差要求。为满足最大偏差要求，赛灵思建议在使用局部反转时，对时钟和反转时钟引脚使用单一的网络。

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 引脚上使用局部反转的理想配置。使用理想配置可保证在减少全局时钟资源占用的同时最大限度地满足偏差要求。

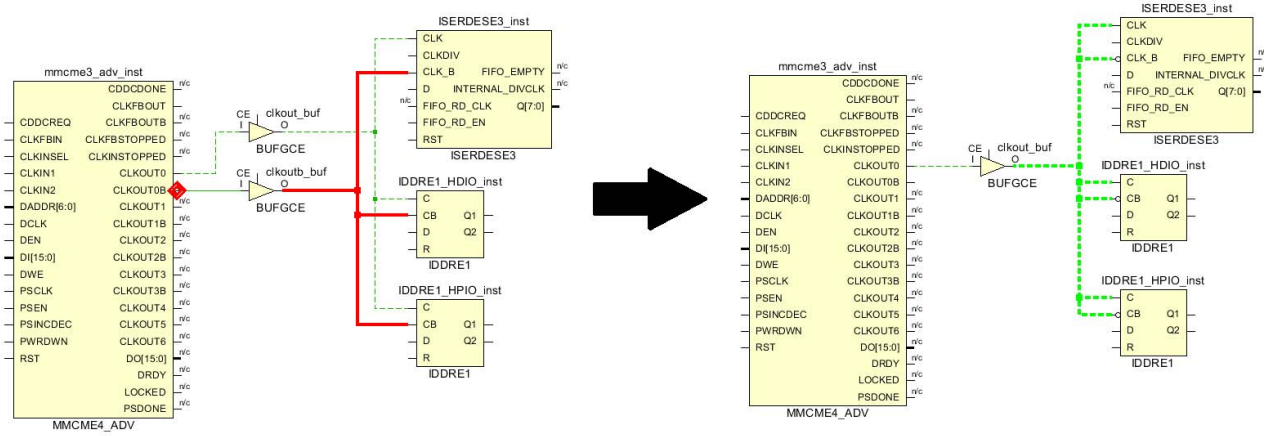


图 3-46：针对 ISERDESE3 和 IDDRE1 的次优和理想时钟拓扑

OSERDESE3 时钟

对于 UltraScale 器件 和 UltraScale+ 器件中的 OSERDESE3 时钟而言，在高速时钟和分频时钟引脚之间存在最大的偏差要求。为满足最大偏差要求，赛灵思建议使用并行全局时钟缓冲器，其中一个全局时钟缓冲器为 BUFGCE_DIV。这样可以消除 MMCM 两个输出间的额外时钟不确定性。

在下图中，左侧是使用 MMCM 的两个单独输出的次优配置。图右侧是使用单个 MMCM 输出和 BUFGCE_DIV 单元的理想配置，该单元能提供使用 BUFGCE_DIVIDE 属性的分频时钟。

注释：高速时钟无需使用 BUFGCE 驱动。作为替代，您可以使用 BUFGCE_DIV，并将 BUFGCE_DIVIDE 属性设置为 1。

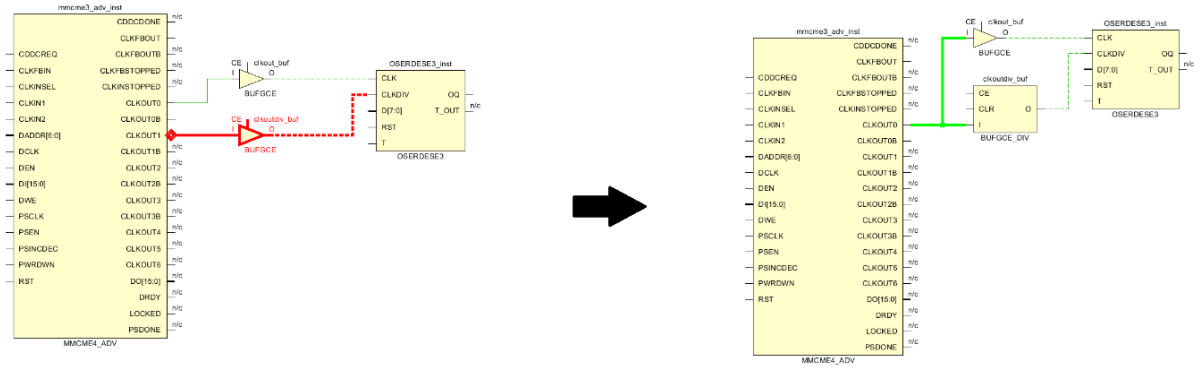


图 3-47：针对 OSERDESE3 的次优和理想时钟拓扑

使用 MMCM 的 I/O 时序 ZHOLD/BUF_IN 补偿

ZHOLD 补偿表示 MMCM 配置，为整个 I/O 列的所有 I/O 寄存器提供负保持。当具有时钟功能的 I/O (CCIO) 驱动配置为 ZHOLD 补偿模式的单个 MMCM 时，布局器尝试将具有 CCIO 的 MMCM 布局在同一时钟域中。在这种情况下，CCIO 可以直接驱动 MMCM，而无需通过 BUFG。这能够使 MMCM 的 ZHOLD 补偿保持有效。

然而，如果 CCIO 驱动在 ZOLD 模式下配置的 MMCM 以及另一个 MMCM，则逻辑优化将尝试通过在 CCIO 之后插入 BUFG 将 MMCM 时钟布线合法化。由于具有 ZHOLD 补偿的 MMCM 不再由 CCIO 直接驱动，因此补偿更改为 BUF_IN。为了避免这种情况，请确保 CCIO 直接驱动在 ZHOLD 模式下配置的 MMCM，并通过 BUFG 驱动附加 MMCM。此外，将由 BUFG 驱动的网络的 CLOCK_DEDICATED_ROUTE 属性设置为 ANY_CMT_COLUMN。

因为时钟插入延迟随时钟根位置而变化，并且时钟根位置取决于负载的位置，所以在运行之间可能存在变化。这种可变性影响 FPGA 内部的时序以及 I/O 时序。

在处理高频 I/O 时，您可能需要更妥善地控制 I/O 时序和运行之间的可变性。实现这一目标的一种方法是强制时钟根位置。您可以在自动模式下运行该工具，并查看时钟根区域。如果 I/O 时序合适，您可以强制在与 I/O 时序相关的缓存网络上设置时钟根。要确定时序钟根的位置，请使用 Tcl 命令 `report_clock_utilization [-clock_roots_only]`。

在以下示例中， I/O 端口位于 X0Y0 区域中。 Vivado 布局器基于 I/O 布局以及其他负载的布局，确定了 X1Y2 中时钟根的布局。

```
| Index | Clock Net                          | Root Clock Region | Clock Root Node                          |
+-------+------------------------------------+-------------------+------------------------------------------+
|     1 | mmcm/inst/clk0                     | X1Y2              | RCLK_BRAM_L_X30Y209/CLK_VDISTR_B0T0      |
|     2 | mmcm/inst/clkfbout_buf_mmcm_zhold  | X1Y2              | RCLK_CLEL_R_L_X25Y209/CLK_VDISTR_B0T     |
+-------+------------------------------------+-------------------+------------------------------------------+
```

图 3-48：具有无约束时钟根的时钟利用率摘要

以下摘要显示了当时钟根不受约束时的 I/O 时序。

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

图 3-49：无约束时钟根的时序摘要

在以下示例中，时钟根移到 X0Y0 中的 I/O 寄存器旁边，这减少了时钟插入延迟和时序不定态，从而提升 I/O 时序。

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

图 3-50：时钟利用率摘要与用户约束时钟根

以下摘要显示移动时钟根时的 I/O 时序。

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

图 3-51：用户约束时钟根的时序摘要

同步 CDC

当设计包括来自同一 MMCM/PLL 的时钟之间的同步 CDC 路径时，您可以使用以下技术来更好地控制时钟插入延迟和时滞，并因此控制这些路径上的松弛。

重要提示：如果 CDC 路径在源自不同 MMCM/PLL 的时钟之间，则跨越 MMCM/PLL 的时钟插入延迟更难以控制。在这种情况下，赛灵思建议您将这些跨时钟域视为异步，并相应地更改设计。

当路径在源自同一 MMCM/PLL 的不同输出引脚的两个时钟之间时序时， MMCM/PLL 相位错误会增加路径的时钟不确定性。对于使用高时钟频率的设计，相位错误可能导致建立和保持时序收敛问题。

下图显示了具有和不具有相位错误的路径的示例。路径 1 是由连接到同一 MMCM 输出的两个缓存计时的 CDC 路径，并且不包括相位错误。路径 2 由来自两个不同 MMCM 输出的两个时钟计时，并且包含相位错误。

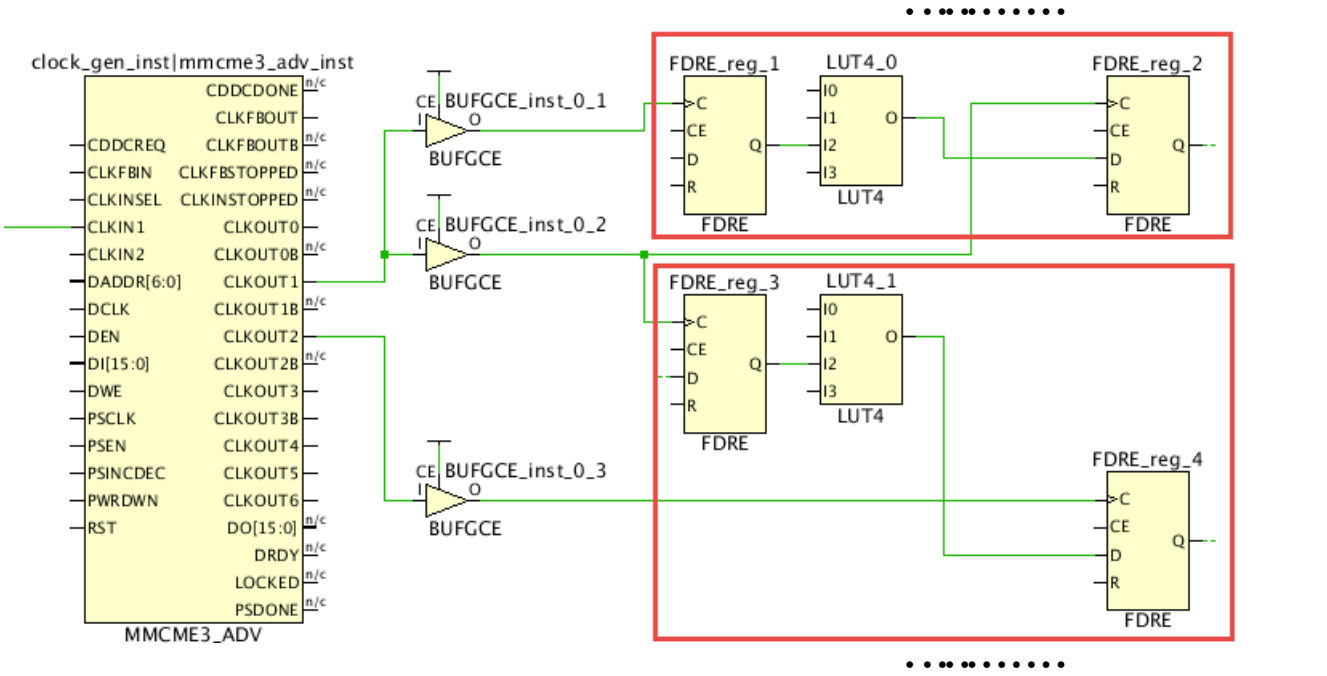


图 3-52：MMCM 和相位错误

当来自同一 MMCM/PLL 的两个同步时钟具有简单的周期比 (/2/4/8) 时，可以使用连接到两个 BUFGCE_DIV 缓存的单个 MMCM/PLL 输出来防范两个时钟域之间的相位错误。 BUFGCE_DIV 缓存执行时钟分频 (/1/2/4/8)。还存在比率的可能 (/3/5/7)，但这需要修改时钟占空比，并使实现混合边沿时序路径更加困难。

注释：由于 BUFGCE 和 BUFGCE_DIV 没有相同的单元延迟，因此赛灵思建议对两个同步时钟 （两个 BUFGCE 或两个 BUFGCE_DIV 缓存）使用相同的时钟缓冲器。

下图显示了两个 BUFGCE_DIV，将 CLKOUT0 时钟分别除以 1 和 2。

注释：尽管下图仅并行使用两个 BUFGCE_DIV，但在时钟域中最多可使用四个 BUFGCE_DIV。

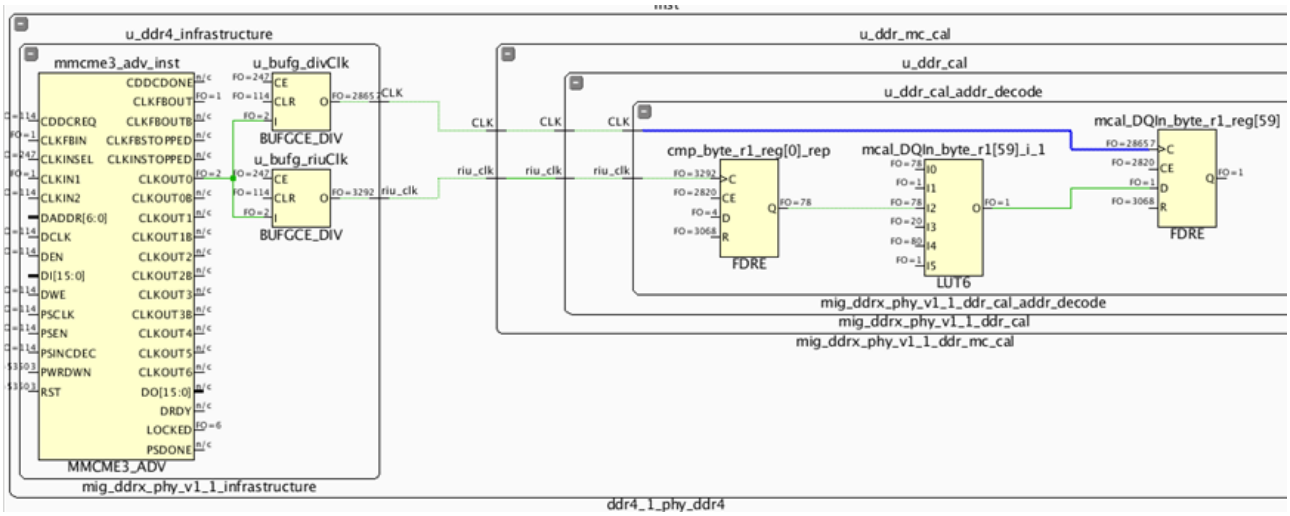


图 3-53：MMCM 同步 CDC，其中 BUFGCE_DIV 连接到一个 MMCM 输出

要自动平衡来自同一 MMCM 或 PLL 的多个时钟，请在需要平衡的时钟缓冲器驱动的网络上设置相同的 CLOCK_DELAY_GROUP 属性值。以下另提供了其他建议：

- 避免在太多时钟上设置 CLOCK_DELAY_GROUP 约束，因为这会过度使用时钟布局器导致次优解或错误。
- 在“Timing Summary Report”中查看关键的同步 CDC 路径，以确定哪些时钟必须进行延迟匹配才能满足时序。
- 在具有严格要求和相同时钟拓扑结构的同步时钟组上限制使用 CLOCK_DELAY_GROUP。

重要提示：赛灵思建议使用时钟向导创建最佳时钟结构，它使用 BUFGCE 和 BUFGCE_DIV 的混合以及相关的时钟分组约束。

GT 接口时钟

每个 GT 接口需要几个时钟，包括一些共享时钟，这些时钟在位于一个或多个 GT quad 中的绑定 GT*_CHANNEL 单元之间共享。 UltraScale 器件提供高达 128 个 GT*_CHANNEL 位置，这可能导致在设计中使用几百个时钟。大多数 GT 时钟具有低扇出，负载被布局在相关 GT*_CHANNEL 旁边的时钟域中。一些 GT 时钟驱动整个器件的负载，并且需要在许多时钟域中利用时钟布线资源。 UltraScale 架构包括以下增强功能，可有效支持所需的大量 GT 时钟。

BUFG_GT 与动态除法器

在 UltraScale 器件中，BUFG_GT 缓存简化了 GT 时钟。由于 BUFG_GT 包括动态分区功能，MMCM 无需在 GT 输出时钟上执行简单的整数除法。这节省了时钟资源，并且在需要分离的 GT*_CHANNEL 输出时钟和全速率时钟时，能改进低偏差时钟路径。

对于 GT 接口，可以使用 BUFG_GT 全局时钟缓冲器，其中用户逻辑以内部 PCS 逻辑时钟频率的半数运行，而对于 PCIe® 接口，GT *_CHANNEL 需要为 user_clk、sys_clk 和 pipe_clk 生成多个时钟频率。下图比较了单通道 GT 接口的 7 系列和 UltraScale 器件之间的时钟要求，其中 TXUSRCLK2 的频率等于 TXUSRCLK 频率的一半。

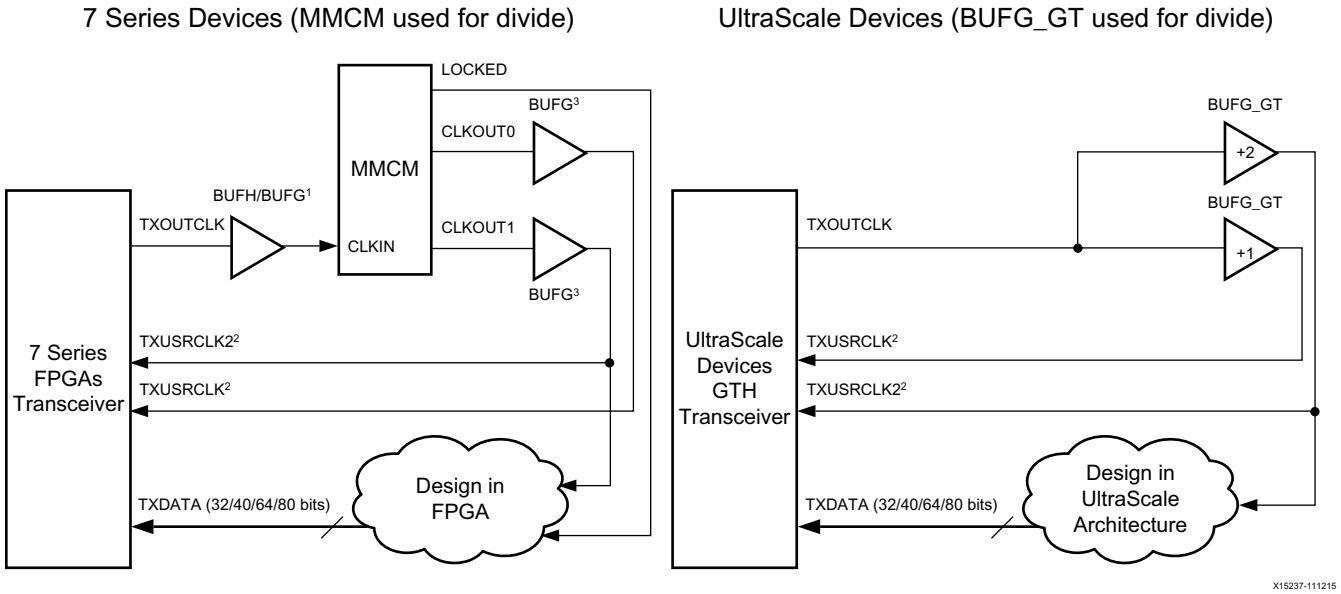

7 Series Devices (MMCM used for divide)
UltraScale Devices (BUFG_GT used for divide)
LOCKED
BUFG3
CLKOUT0
MMCM
BUFH/BUFG1
TXOUTCLK
CLKIN
CLKOUT1
BUFG3
7 Series FPGAs Transceiver
TXUSRCLK22
TXUSRCLK2
TXDATA (32/40/64/80 bits)
Design in FPGA
BUFG_GT
+2
BUFG_GT
+1
TXOUTCLK
UltraScale Devices GTH Transceiver
TXUSRCLK2
TXUSRCLK22
TXDATA (32/40/64/80 bits)
Design in UltraScale Architecture
X15237-111215

图 3-54：时钟要求比较

您可以使用 Quad 中的 GT*_CHANNEL 的任意输出时钟或 Quad 中的 IBUFDS_GTE3/ODIV2 引脚生成的任何参考时钟来驱动位于同一时钟域中的 24 个 BUFG_GT 缓存中的任何一个。必须采用 BUFG_GT_SYNC 来同步复位和清除由公共时钟源驱动的 BUFG_GT。

注释：如果它在设计中不存在， Vivado 工具将自动插入 BUFG_GT_SYNC 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 相同的时钟域行上。如果其他 MMCM 尝试使用同一个 MMCM 位置，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。

单四通道和多四通道接口

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]。如果多通道接口跨越多个 quad，GT*CHANNEL 从参考时钟源起的最大允许距离是上面和下面的 2 个时钟域。

下图显示了一个多四通道接口。 GT*CHANNEL 标记为黄色， TXUSRCLK 以蓝色突出显示， TXUSRCLK2 以红色突出显示。驱动 TXUSRCLK[2] 的 BUFG_GT 位于中心 quad，并标记为蓝色和红色。

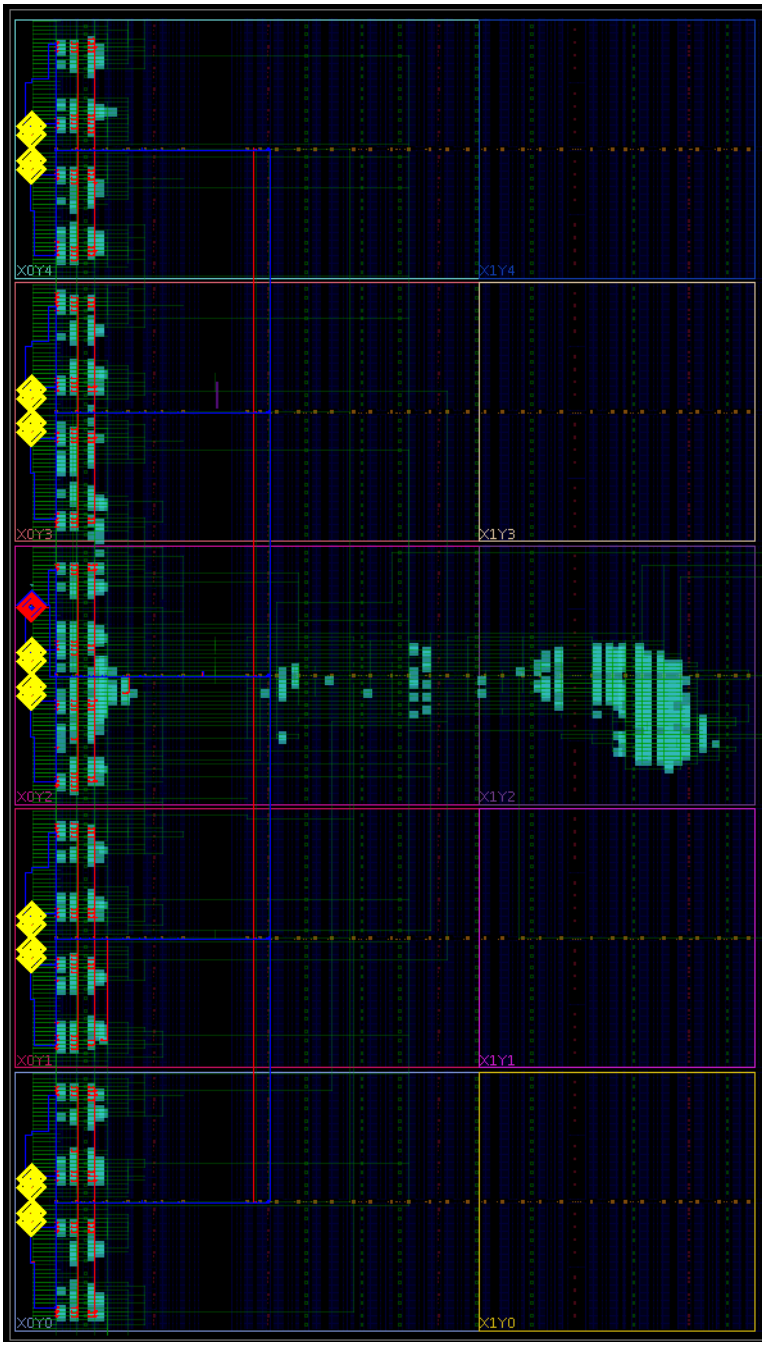


图 3-55：TXUSRCLK/TXUSRCLK2 用于多四通道接口的时钟布线

如果 GT 接口包含在单个 Quad 中，则布局器将 BUFG_GT 时钟视为本地时钟。在这种情况下，布局器尝试将 BUFG_GT 时钟负载布局在与 BUFG_GT 水平相邻的时钟域中，从包含 BUFG_GT 并可能使用高达器件宽度一半的时钟域开始。

要覆盖布局器区域时钟约束，请将任何 BUFG_GT 时钟负载分配给 Pblock。下图显示了单四通道接口。 GT*CHANNEL 标记为黄色， TXUSRCLK 以蓝色突出显示， TXUSRCLK2 以红色突出显示。所有 TXUSRCLK2 负载都布局在与 GT*CHANNEL 相同的时钟域中。

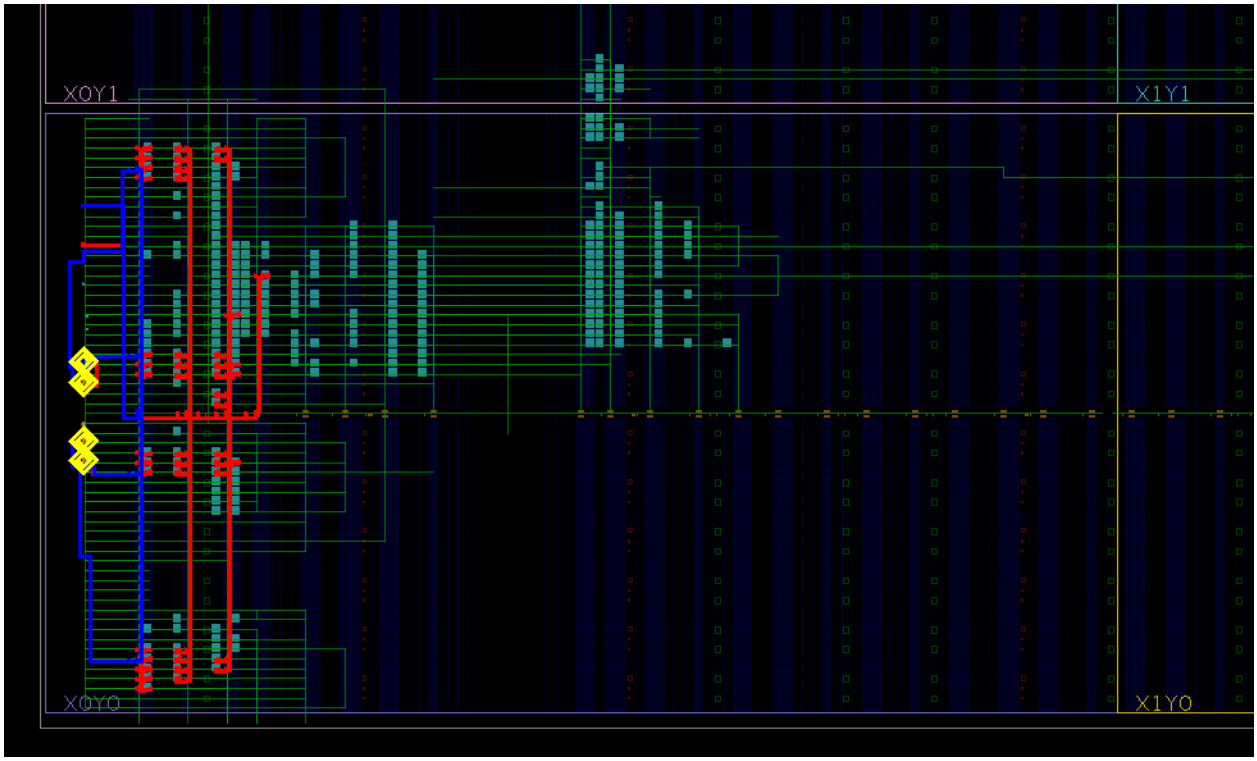


图 3-56：TXUSRCLK/TXUSRCLK2 单四通道接口的时钟布线

[RT]XUSRCLK/[RT]XUSRCLK2 偏差匹配

当 [RT]XUSRCLK2 以 [RT]XUSRCLK 一半的频率运行时 （即，通过除以 1 和除以 2 分离 BUFG_GT），在 GT 接口每个 GT*CHANNEL 的 [RT]XUSRCLK/[RT]XUSRCLK2 对之间存在严格的偏差要求。为了满足时滞要求， GT*CHANNEL 可以是高于或低于生成 [RT]XUSRCLK/[RT]XUSRCLK2 对的主通道的最多 2 个时钟域。此外，如下所示，布局器严格控制偏差：

- 将 BUFG_GT 对分配给四通道中的上部或下部 12 个 BUFG_GT
- 将两个时钟的时钟根分配给包含 BUFG_GT 的时钟域

建议：为避免偏差违规，当 [RT]XUSRCLK2 以 [RT]XUSRCLK 一半的频率运行时，赛灵思强烈建议采用该时钟拓扑结构。

用于 PCI Express 的集成块 CORECLK/PIPECLK/USERCLK 偏差匹配

用于 PCI Express ® 的 UltraScale 集成块需要三个时钟：CORECLK、 USERCLK 和 PIPECLK。这三个时钟由 BUFG_GT 提供，由物理接口的 GT*_CHANNEL 之一的 TXOUTCLK 引脚驱动。在 CORCLK 和 PIPECLK 引脚与 CORECLK 和 USERCLK 引脚之间存在严格的偏差要求。为满足偏差要求，布局器严格控制偏差如下：

- 将驱动组中的三个 PCIe 时钟的 BUFG_GT 分配给四通道中的上或下层 12 个 BUFG_GT
- 将所有三个时钟的时钟根分配给同一时钟域

注释：如需了解更多有关 PCIe 时钟要求的信息，请参阅 《用于 PCI Express LogiCORE IP 的 UltraScale 架构 Gen3 产品指南》(PG156) [参照 42]。

7 系列器件时钟

本章节以 Virtex®-7 时钟源为例。 Virtex-6 的时钟资源与此类似。如果使用不同的架构，请参阅有关器件的 《时钟资源指南》[参照 40]。

Virtex-6 和 Virtex-7 器件内含 32 个称为 BUFG 的全局时钟缓存。 BUFG 可满足设计的大部分时钟需求，且对下列要求不高：

- 时钟数量
- 设计性能
- 功耗需求低
- 其它时钟特性，比如：
  - 时钟门控
  - 多路复用
  - 时钟分频
  - 其它时钟控制

BUFG 可通过综合功能调用得到，同时限制条件极少，适用于大多数普通时钟。

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。

全局时钟资源

这部分将探讨下列全局时钟资源：

- BUFG

  全局时钟缓存 (BUFG) 元件一般供时钟使用。这种全局时钟缓存带有附加功能。这些附加功能可通过手动干预设计代码或综合加以利用。

- BUFGCE

  在使用 BUFGCE 原语的情况下，无需使用任何其它逻辑或资源，就能够访问同步无毛刺时钟使能 （门控）功能。BUFGCE 可让时钟停止一段时间，或用于创建更低偏差、更低功耗的时钟分频信号，比如从更高频率的基时钟创建 ½ 或 ¼ 频时钟，特别适用于在电路工作的不同时间需要不同频率时钟的情况。

- BUFGMUX

  BUFGMUX 可用于安全地修改时钟，实现从一个时钟源到另一个时钟源无毛刺切换，同时不会造成其它时序风险。在需要根据时间条件或工作条件的情况下， BUFGMUX 可使用两个截然不同的时钟频率。

- BUFGCTRL

  BUFGCRTL 能够访问全局时钟网络的全部功能，以便于异步控制时钟来适应更加复杂的时钟条件，比如时钟丢失或停止时钟切换电路等。

  在大多数情况下，该组件必须在代码中实例化，且必须进行正确的连接，才能获得所需的时钟行为。

  在某些情况下，IP 和综合可使用这些更高级的时钟特性。例如：在使用存储器接口生成器 (MIG) 时，就可以把专用时钟缓存用于 I/O 处的高速数据传输和采集。应随时掌握各个 IP 需要的和已用的时钟资源，并在总体时钟架构建和规划过程中加以解释。

如需了解更多使用这些组件的信息，请参阅 《时钟资源用户指南》和介绍特定器件情况的 《库指南》。

区域时钟资源

除全局时钟资源之外，还有区域时钟资源：

- 水平时钟域缓存 （BUFH、 BUFHCE）

  水平时钟域缓存（BUFH、BUFHCE）既可单独使用，也可与 BUFG 配合使用。使用这些缓存可以更加严格地控制与该时钟相连的相关逻辑的时钟和布局，同时为拥有大量时钟域的设计提供更多时钟资源。

  BUFH 和 BUFHCE 资源允许设计使用连接到给定时钟域的全局时钟网络 (BUFG) 的部分。对于位于时钟域内的较小时钟域，允许使用全局时钟网络的未使用部分来访问低偏移资源。 BUFHCE 具有相同的无毛刺时钟使能功能，便于为特定时钟域提供简单安全的时钟门控。

  在由 BUFG 驱动时， BUFHCE 可用作中等粒度时钟门控功能。应对有数百或者数千负载的时钟域，且如果需要在其中的一部分间歇性地关闭时钟，BUFHCE 就是有效的时钟资源。BUFG 可驱动在同一或不同时钟域中的多个 BUFH，从而实现多个时钟可独立控制的低偏差时钟域。

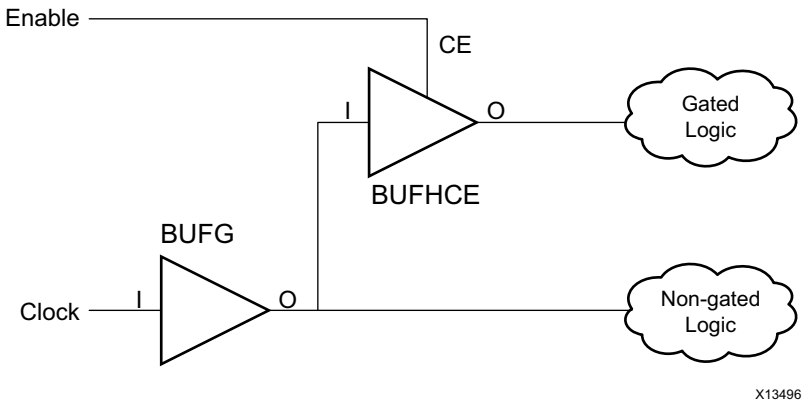


图 3-57：水平时钟域缓存

在独立使用时，所有连接到 BUFH 的负载必须处于相同的时钟域中。这样能良好地满足超高速、更加精细粒度（更少负载数量）的时钟需求。BUFHCH 可用于实现特定时钟域中的中等粒度时钟门控。用户必须确保由 BUFH 驱动的资源不会超出时钟域中的可用资源，且不存在其它冲突。

提示：在这些网络上避免使用重负载，可以避免发生这一问题。

BUFH 和由 BUFG、其他 BUFH 或任何其它时钟域驱动的时钟域之间的相位关系可能不同。唯一的例外是当有两个 BUFH 驱动水平相邻的时钟域时。此时如果两个 BUFH 都由同一时钟源驱动，左右时钟域之间的偏差应具有高度受控的相位关系，这样数据就可以安全地跨越这两个 BUFH 驱动的时钟域。BUFH 还可用于访问对面区域中的 MMCM 或 PLL，用于时钟输入或千兆位收发器。但使用这种方法时必须小心谨慎，以确保有 MMCM 或 PLL 可用。

- 区域缓存 (BUFR)

  区域时钟缓存 (BUFR) 一般用作较低速的 I/O 和架构时钟，供采集和提供更高速的 I/O 数据使用。BUFR 不仅能够使能和禁用 （门控）时钟，还能够完成某些常见的时钟分频功能。在 Virtex-7 器件中， BUFR 只能驱动其所在的时钟域。这使得这种缓存更适用于较小型的时钟网络。

  由于 BUFR 的性能略低于 BUFG 和 BUFH，赛灵思不建议将其用于超高速时钟。但是它也非常适用于许多中低速时钟的需求。附加的内置时钟分频功能也使之适用于由高速 I/O 接口时钟等外部时钟源驱动的分频时钟网络。 BUFR 无需占用全局走线，并能替代 BUFH 的使用。

- I/O 时钟缓存 (BUFIO)

  I/O 时钟缓存 (BUFIO) 只用于采集 I/O 数据到输入逻辑，并在器件上为输出逻辑提供输出时钟。 BUFIO 一般用于：

  - 在 bank 中采集高速源同步数据
  - 在器件中把数据降低到更可控的速度 （结合 BUFR、 ISERDES 或 OSERDES 逻辑使用）。

重要提示：BUFIO 只能驱动位于 ILogic 和 OLogic 结构中的输入和输出组件，比如 IDDR、 ODDR、 ISERDES、OSERDES，或是简单的专用输入或输出寄存器。

  在使用 BUFIO 时，必须考虑将数据从 I/O 逻辑可靠传输到架构的需求，反之亦然。

- 多区域时钟缓存 (BUFMR)

  多区域时钟缓存 (BUFMR) 允许使用单个时钟引脚 (MRCC) 驱动位于自身 bank 中的 BUFIO 和 BUFR，以及其上下的 I/O bank （如果存在）。

  如需了解更多有关赛灵思 7 系列 FPGA 器件的时钟资源的信息，请参阅 《7 系列 FPGA 时钟资源用户指南》(UG472)》[参照 40]。

关于SSI 器件的更多时钟考虑

通常，上述所有时钟考虑因素同样适用于 SSI 技术器件。但由于其结构问题，在针对这些器件时，还有需要考虑更多因素。使用 BUFMR 时，不能驱动跨 SLR 边界的时钟资源。相应地，赛灵思建议用户把负责驱动 BUFMR 到 bank 或时钟域的时钟布局在 SLR 内部的中心时钟域，以便访问 SLR 左右两侧的全部三个时钟域。

就全局时钟而言，对设计需要的全局时钟 (BUFG) 数量不超过 （包括）16 个的情况，没必要考虑更多因素。这些工具会自动分配 BUFG，避免可能发生的冲突。在需要的 BUFG 数量超过 16 个 （但不足 32 个）时，必须在引脚选择和布局方面进一步考虑，才能避免因全局时钟线竞争和/或时钟负载布局引起的资源争用。

和所有其它赛灵思 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。位于 SLR 上半部或下半部的 CCIO 只能驱动对应的上半部或下半部中的 BUFG。因此，在选择引脚和相关的 CMT 的时候，应注意不要让所有 SLR 的上半部或下半部的 BUFG 超过 16 个。为此，该工具可自动分配所有的 BUFG，在避免发生冲突的情况下把全部时钟驱动到所有的 SLR。

对需要的全局时钟数量超过 32 个的情况，赛灵思建议尝试把 BUFR 和 BUFH 用于较小的时钟域，从而减少所需的全局时钟域数量。结合使用 BUFR 和 BUFMR，可以驱动三个时钟域内的资源，覆盖二分之一个 SLR （对于 Virtex-7 级 SLR，约为 25 万个逻辑单元）。水平相邻的时钟域可同时拥有左右两侧由低偏差方式驱动的 BUFH 缓存，实现规模相当于三分之一个 SLR 的时钟域 （约为 16.7 万个逻辑单元）。

尽量利用这些资源不仅可以减少时钟资源争用方面的考虑，而且可以多次完善总体布局，从而提升性能、降低功耗。

如果需要超过 32 个全局时钟来驱动多半个 SLR 或多个 SLR，很可能要细分 BUFG 全局时钟轴。在 SLR 边缘处的垂直全局时钟线上有隔离缓存，便于在不发生冲突的情况下，在占据相同垂直全局时钟线的不同 SLR 中使用两个 BUFG。利用此项功能需要更多用户控制和干预。在下图中，三个 SLR 中从 BUFG0 到 BUFG2 已被隔离，因此可在其各自的 SLR 内拥有独立时钟。另一方面， BUFG31 线未被隔离。因此，同一 BUFG31 （位于图中的 SLR2 中）可用来驱动所有 3 个 SLR 中的时钟线，但应禁用位于其它 SLR 中的 BUFG31。

对 BUFG 而言，必须精心选择和手动布局 (LOC)。此外，每个时钟域的所有负载都必须手动编组、手动布局在适当的 SLR 中，以避免时钟冲突。如果所有全局时钟的布局和所有负载的管理能够在避免发生任何时钟冲突的情况下让时钟达到所有的负载，就能够使用数量多于 32 个的该全局时钟资源。

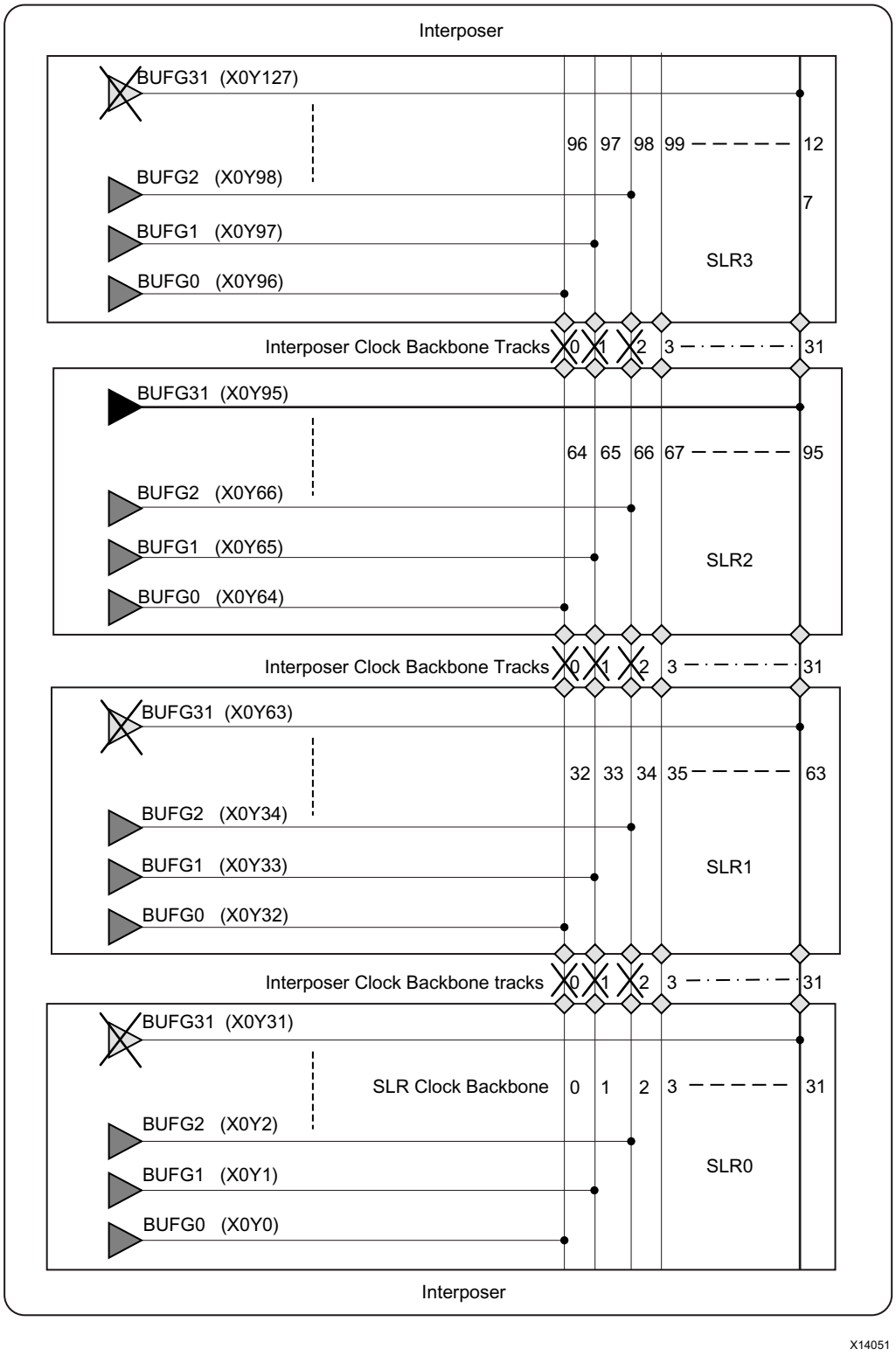


图 3-58：SSI 器件时钟线上的隔离选择

SSI 技术器件中的全局时钟资源的时钟偏差

时钟偏差对任何大型 FPGA 器件而言，都可能占给定路径总体时序预算的主要部分。过大的时钟偏差不仅会给最高时钟速度造成问题，本身还会带来严苛的保持时间要求。在器件中内置多个晶片会带来更为严峻的 PVT 工艺问题，但在赛灵思组装工艺的管理下，只有速度相近的晶片才会封装在一起。

即便有这样的工艺，赛灵思时序工具还是会把这些差异包含在时序报告中。在分析路径的过程中，这些方面会作为建立和保持计算的一部分加以分析，并依据规定的要求，以路径延迟的形式反映在报告中。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 进行延迟微分计算，偏差会增大，而且各点之间距离越远，偏移越大。因此赛灵思建议对全局时钟而言，中心 SLR 中必须布局一个以上的 SLR。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。

针对 UltraScale 器件而言，时钟布局的影响较少。然而，赛灵思仍然强烈建议将时钟资源尽可能接近时钟负载的中心点布置，以降低时钟插入延迟并降低时钟功耗。

时钟结构设计

现在已经清楚地说明时钟决策的主要考虑因素，下面将介绍如何为设计提供需要的时钟。

调用

无需用户干预，Vivado 综合工具就可以自动为所有时钟结构设定全局缓存 (BUFG)，直到架构允许的最大数量（除非用综合工具另行设定或加以控制）。如前文所述， BUFG 能够提供满足大多数时钟需求的、受控良好的低偏差网络。除非器件上的 BUFG 数量或功能无法满足设计的时钟要求，无需另行干预。

但是如果对时钟结构施加额外控制，在抖动、偏差、布局、功耗、性能或其它方面可能会获得更优异的特性。

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 调用。
- 用替代性时钟结构取代 BUFG。
- 设定某种以其它方式无法实现的时钟缓存。

使用综合约束，无需对代码进行任何修改，就可以实现此类控制。

属性可布局在任意下列位置之一：

- 直接布局在 HDL 代码中，这样属性就可以一直存在于代码中
- 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 I/O Wizard 专用于协助时钟资源和结构的选择和创建，包括：

- BUFG
- BUFGCE
- BUFGCE_DIV (UltraScale 器件)
- BUFGCTRL
- BUFIO (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

存储器接口生成器 (MIG)、PCIe 或收发器向导等更复杂的 IP 也可囊括时钟结构，当作总体 IP 的一部分。如果适当加以考虑，这也可以提供额外的时钟资源。但如果不加考虑，可能会限制设计其余部分的某些时钟选项。

赛灵思强烈建议对任何实例化的 IP 均应良好掌握其时钟的要求、功能和资源，并尽量在设计中的其余部分加以运用。

如需了解更多信息，请参阅“充分利用 IP 核”。

实例化

最低级也是最直接的控制时钟结构的方法是将所需的时钟资源实例化到 HDL 设计中。这样就可以使用器件的全部功能并对它们施加绝对的控制。在使用 BUFGCE、 BUFGMUX、 BUFHCE 或其它需要额外逻辑和控制的时钟结构时，实例化通常是不二之选。但是即便是对简单的缓存而言，有时候取得所需结果最迅捷的方法还是直截了当地把它实例化于设计中。

一种有效管理时钟资源的方法 （特别是在实例化时）是将时钟资源限定在单独实体或模块中，在代码顶层或顶层附近实例化。通过将时钟资源置于代码顶层，就可将代码更方便地分配给设计中的多个模块。

应该注意可以共享且应该共享时钟资源的地方。创建冗余时钟资源不仅是资源浪费，而且通常会造成更多能耗，带来更多潜在冲突和布局决策，导致执行工具运行时间延长，以及更加复杂的时序状况。这也是为什么把时钟资源置于顶层模块附近的又一重要原因。

提示：可以使用 Vivado HDL 模版实例化特定时钟原语。请参阅“使用 Vivado Design Suite HDL 模板”。

控制时钟的相位、频率、占空比和抖动

本节提供了微调时钟特性的技术。

使用时钟修改块 （MMCM 和 PLL）

用户可使用 MMCM 或 PLL 修改输入时钟的总体特性。 MMCM 最常用于消除时钟的插入延迟 （将时钟与输入系统同步数据在相位上对齐），例如：

- 创建更加严格的相位控制。
- 滤除时钟中的抖动。
- 修改时钟频率。
- 纠正或更改时钟占空比

为正确使用 MMCM 或 PLL，必须协调多项属性，以确保 MMCM 在规范范围内工作，能够在输出端提供所需的时钟特性。为此，赛灵思强烈建议使用 Clocking Wizard 来正确配置这一资源。

您还可以直接示例化 MMCM 或 PLL，这将提供妥善的控制。但是，请务必使用正确的设置，以避免出现下问题：

- 造成抖动增大，进而降低时钟可靠性。
- 建立不正确的相位关系。
- 提高时序收敛难度。

重要提示：在使用 Clocking Wizard 配置 MMCM 或 PLL 时，在默认条件下 Clocking Wizard 会出于低输出抖动目的，使用合理的功耗特性配置 MMCM。

根据您的目标，您可以更改时钟向导中的设置，以进一步最小化抖动，从而以更高功耗为代价改进时序。或者，您可以降低功耗，但会增加输出抖动。

使用 MMCM 或 PLL 时，请务必执行以下操作：

- 切勿让任何输入浮动。不建议靠综合工具或其它优化工具来锁定浮动值，因为它们锁定的值可能与所需的值有偏差。
- 将 RST 连接到用户逻辑，以便可以由可靠的时钟源控制的逻辑来断言。如果时钟被中断，就会对 RST 接地造成问题。
- 在执行复位时使用 LOCKED 输出。例如，保持从 PLL 复位的同步逻辑时钟，直到 LOCKED 置位。 LOCKED 信号在用于设计同步部分之前需要同步。赛灵思建议将 LOCKED 添加到处理器映射，以便在调试时可见。
- 确认 CLKFBIN 和 CLKBOUT 之间的连接。如果 PLL/MMCM 输出时钟需要与输入参考时钟进行相位对齐，例如使用 ZHOL 补偿模式时， BUFG 只需包含在反馈路径中。
- 要避免 UltraScale 器件中同步跨时钟域路径上的 MMCM 或 PLL 相位错误时序损失，请使用 BUFGCE_DIV 而不是 BUFGCE。详情，请参阅“同步 CDC”。

建议：探索 Clocking Wizard 内的各种不同设置，可确保根据总体设计目标创建最合适的配置。

在时钟上使用 IDELAY 控制相位

对于 7 系列器件来说，如果只需要稍许调整相位，可以使用 IDELAY 或 ODELAY （而非 MMCM 或 PLL）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。在使用 UltraScale 器件时，在输入时钟源上您不可使用 IDELAY。因此，除非需要相位操纵器，赛灵思建议使用 MMCM。

使用门控时钟

赛灵思器件内置专用时钟网络，可提供高扇出低偏差时钟资源。如果在 HDL 代码中包括精细粒度时钟门控技术，则会干扰此功能及防止专用时钟资源的有效使用。因此，在将 HDL 写入器件时，赛灵思不建议把时钟门控结构编码在时钟路径上。与此相反，出于功能或功耗考虑而停止设计的某些部分，应使用通过编码调用时钟使能的方法来控制时钟。

将时钟门控转换为时钟使能

如果编码中已经含有时钟门控结构，或是另有技术要求这样编码，赛灵思建议使用综合工具把已经布局在时钟路径上的门控重新映射到数据路径上的时钟使能。这样做可以更理想地映射到时钟资源，并简化与进/出门控时钟域的数据有关的电路时序分析。例如，利用 `-gated_clock_conversion auto` 选项与 Vivado 综合使用，以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结构，使用 RTL 代码中的 GATED_CLOCK 属性来指导 Vivado 综合。

门控时钟缓冲器

当时钟网络的较大部分可以关闭一段时间时，您可以使用 BUFGCE 或 BUFGCTRL 启用或禁用时钟网络。此外，在定向 UltraScale 器件时，可以门控 BUFGCE_DIV 和 BUFG_GT。对于 7 系列器件，还可以使用 BUFHCE、 BUFR 和 BUFMRCE 来门控时钟。

当时钟可以在一段时间内减慢时，您也可以使用这些缓存和附加逻辑来周期性地使能时钟网络。用户也可以使用 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。

这些技巧中的任何一项都可以有效降低动态功耗。但是根据要求和时钟拓扑结构，某种技巧可能比另一种更加行之有效。例如，在 7 系列器件中：

- 如果 BUFR 是外部生成的时钟 （低于 450 MHz），则只需要提供三个时钟域， BUFR 便可能以最佳状态运行。
- 对于 Virtex-7 器件，也可能需要对多个时钟域 （但最多只能有三个垂直相邻的区域）使用 BUFMRCE 技术。
- BUFHCE 更适合能够包含在单个时钟域中的高速时钟。虽然 BUFGCE 可以跨越器件，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

FPGA 器件完成配置后，器件需要经历一系列事件才能退出配置状态，进入一般工作状态。在大多数配置序列中，最后步骤之一是全局置位复位 (GSR) 解除，然后是全局使能 (GWE) 信号解除。这个步骤完成之后设计进入已知的初始状态，然后释放进入工作状态。

如果上述释放点未同步到给时序钟域，或如果时钟的运行速度快于 GWE 安全释放的速度，部分设计就会进入未知状态。对于一些设计，这没有关系。在其他设计中，这可能导致设计变得不稳定或不正确地处理初始数据集。

如果设计必须在已知状态下启动，赛灵思建议您使用以下任何方法来控制启动同步过程：

- 使用具有时钟使能能力的示例化时钟缓冲器组件。在使能设计时钟之前，将复位释放延迟所需的多个周期。以下示例显示了在 UltraScale 器件中复位后，如何延迟第一个设计时钟边沿。通过在同步寄存器上设置 ASYNC_REG=TRUE，可以讲所有寄存器都放在单个 SLICE 中，因此不需要由全局时钟资源驱动。要防止在同步器时钟上插入时钟缓冲器，请在输入时钟端口上使用 CLOCK_BUFFER_TYPE=NONE 属性。

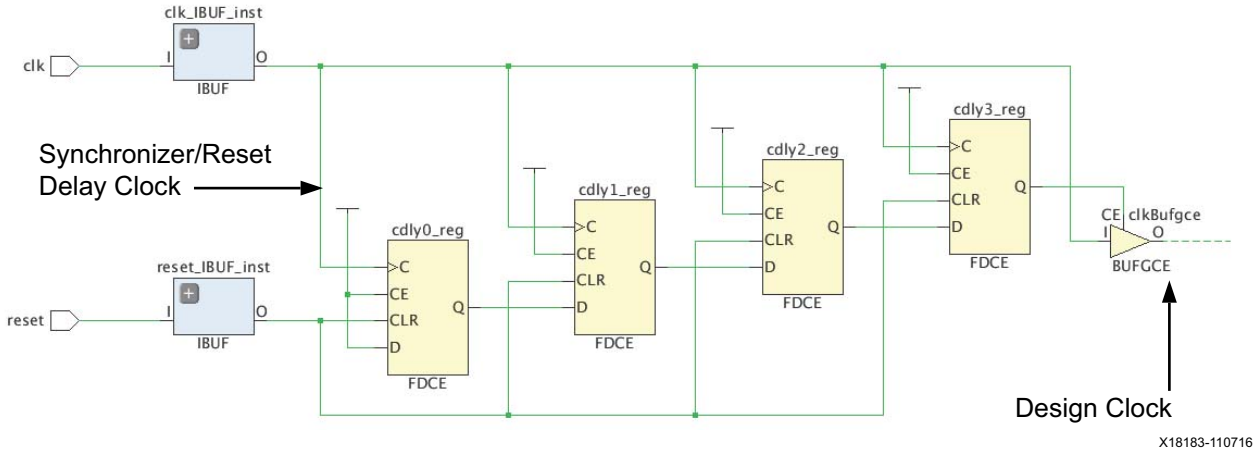


图 3-59：复位安全时钟启动的同步和延迟示例

- 使用 MMCM 时，可以从时钟向导中选择“Safe Clock Startup”选项，以确保只有在设计时钟稳定可靠之后才能使能设计时钟。以下示例显示了连接到 BUFGCE 的 CE 引脚的 UltraScale 器件 MMCM LOCKE D 信号的同步阶段，该引脚驱动用户逻辑。第二个 BUFGCE 并行连接到高扇出 BUFGCE （用户时钟），专用于控制 BUFGCE/CE 引脚的逻辑。此拓扑结构通过最小化同步器和 BUFGCE 引脚之间的时钟偏移，有助于 UltraScale 器件中的 BUFGCE/CE 时序收敛。

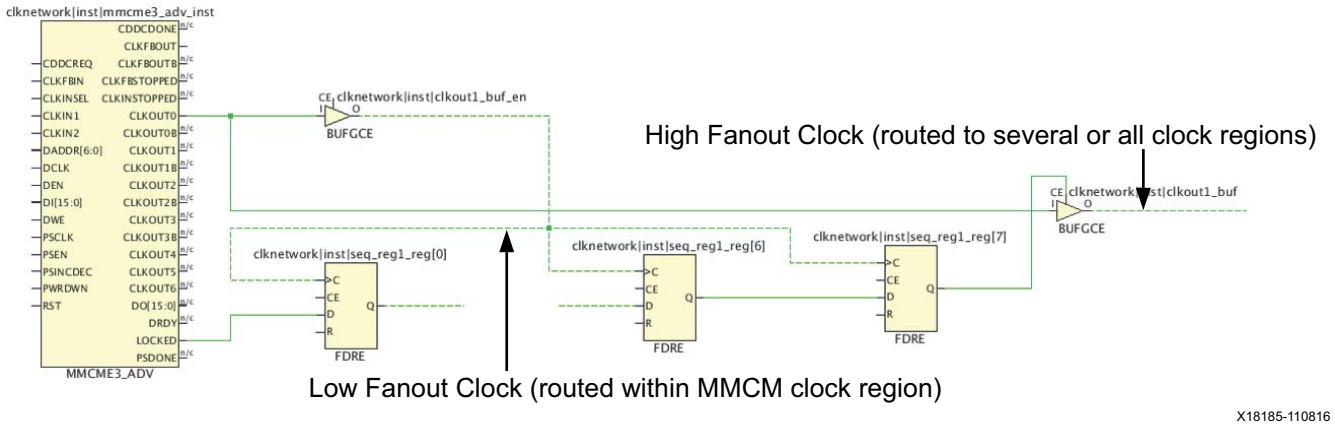


图 3-60：UltraScale 器件 MMCM 安全时钟启动示例

提示：如果 MMCM 或 PLL 补偿模式设置为 ZHOLD 或 BUF_IN，则来自 CLKOUT0 的所有时钟都与反馈时钟分组，并使用相同的 CLOCK_ROOT。如果这在 BUFGCE/CE 上引入了时序违规，则仅在高扇出时钟和反馈时钟之间创建 CLOCK_DELAY_GROUP 约束。或者，您还可以将低扇出时钟网络上的 USER_CLOCK_ROOT 约束设置为与 MMCM 相同的时钟域。对于 7 系列器件，由于不同的时钟架构，通常不需要第二个时钟缓冲器来帮助时序收敛。

- 在设计的关键部分 （例如状态机）上使用时钟使能，本地复位 （同步）或两者，以确保设计的这些部分的启动受控和已知。

避免本地时钟

本地时钟是使用常规结构资源而不是专用全局时钟资源进行布线的时钟网络。在大多数情况下， Vivado 综合和 Vivado 逻辑优化工具在架构要求的时钟缓冲器或具有超过 30 个时钟负载的时钟网络中插入时钟缓冲器。本地时钟通常发生在：

- 全局时钟由用结构逻辑实现的计数器划分
- 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利用率通常不会产生问题。

一般来说，避免使用本地时钟。本地时钟对实现工具带来了几个挑战：

- 不可预测的时钟偏差，导致困难的时序收敛
- 增加低到中等扇出网络，由布线器特别小心处理，导致潜在的可布线性问题

提示：如果本地时钟引入 QoR 问题，尝试布局规划规划时钟驱动器并使用 Pblock 加载到一个小区域。使用 `report_clock_utilization` 标识本地时钟的位置，查看时钟布局，并决定如何减少其数量或影响。

创建输出时钟

把 FPGA 器件的时钟转发给 FPGA 外的时钟器件的最有效方法，就是使用 ODDR 组件。将一个输入保持为高电平，另一个保持为低电平，就可以方便地创建在相位关系和占空比上都受控良好的时钟 （例如：将 D1 保持为 0， D2 引脚保持为 1，就可以实现 180 度的相移）。使用置位/复位和时钟使能，还能控制时钟停止以及保持时钟极性一段时间。

如果需要进一步控制外部时钟的相位，可使用 MMCM 或 PLL 带有外部反馈补偿和/或有粗粒度或精细粒度、固定的或可变的相位补偿。这样就能够更加有力地控制相对于其它器件的时钟相位和传输时间，降低器件提出的外部时序要求。

跨时钟域

跨时钟域 (CDC) 电路在设计中直接影响设计的可靠性。您可以设计自己的电路，但 Vivado Design Suite 必须识别电路，并且必须正确应用 ASYNC_REG 属性。赛灵思提供 XPM 以确保正确的电路设计，包括：

- 在 place_design 中驱动特定功能，减少同步电路上的平均故障间隔时间 (MTBF)。
- 确保通过 report_synchronizer_mtbf 进行识别。
- 避免 report_cdc 错误和警告，这通常在迭代较长时在设计周期中显示。

当在两个异步时钟之间交叉时或者当试图通过添加假路径约束来放松两个同步时钟之间的时序时，需要 CDC 电路。使用 XPM 时，可以选择单个位或多位总线在域之间交错。

单比特 CDC

下图显示了使用单位交叉时所需的决策。

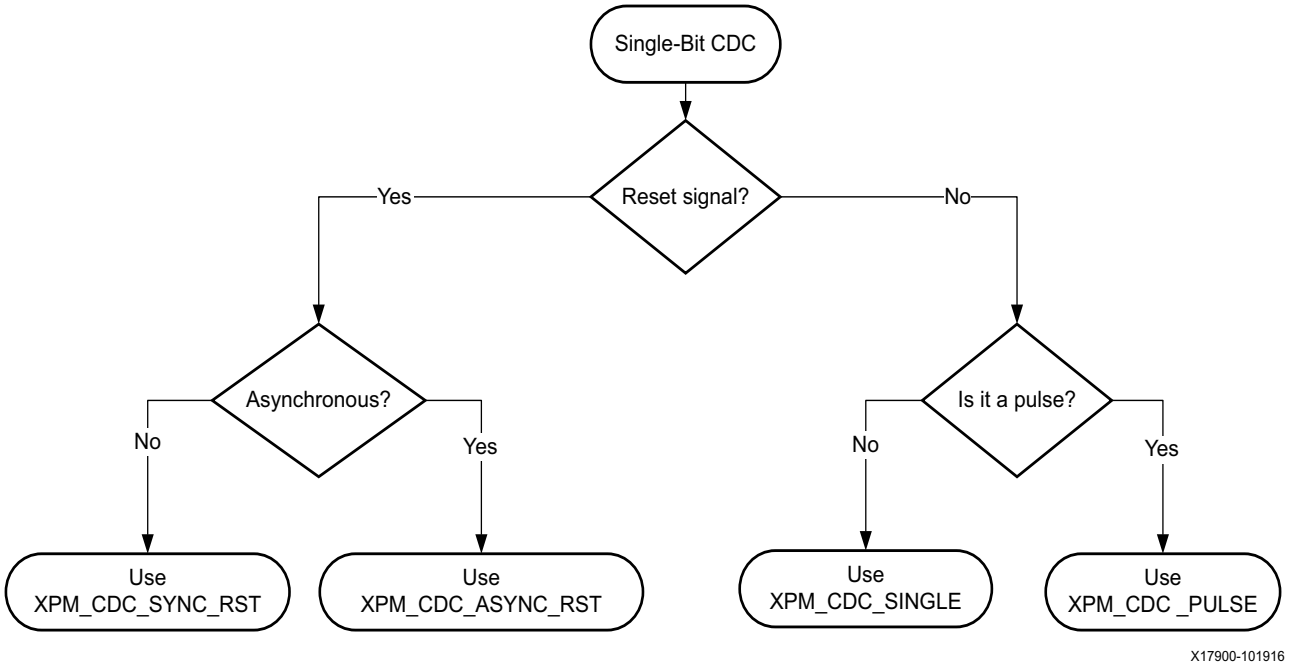


图 3-61：单比特 CDC 决策树

注释：如需了解更多有关不同的单位同步器的信息，请参阅 《Vivado Design Suite 7 系列 FPGA 和 Zynq-7000 All Programmable 库指南》(UG953) [参照 28] 与 《UltraScale 架构库指南》(UG974) [参照 29]。

多位 CDC

下图显示了使用多位交叉时所需的决策。

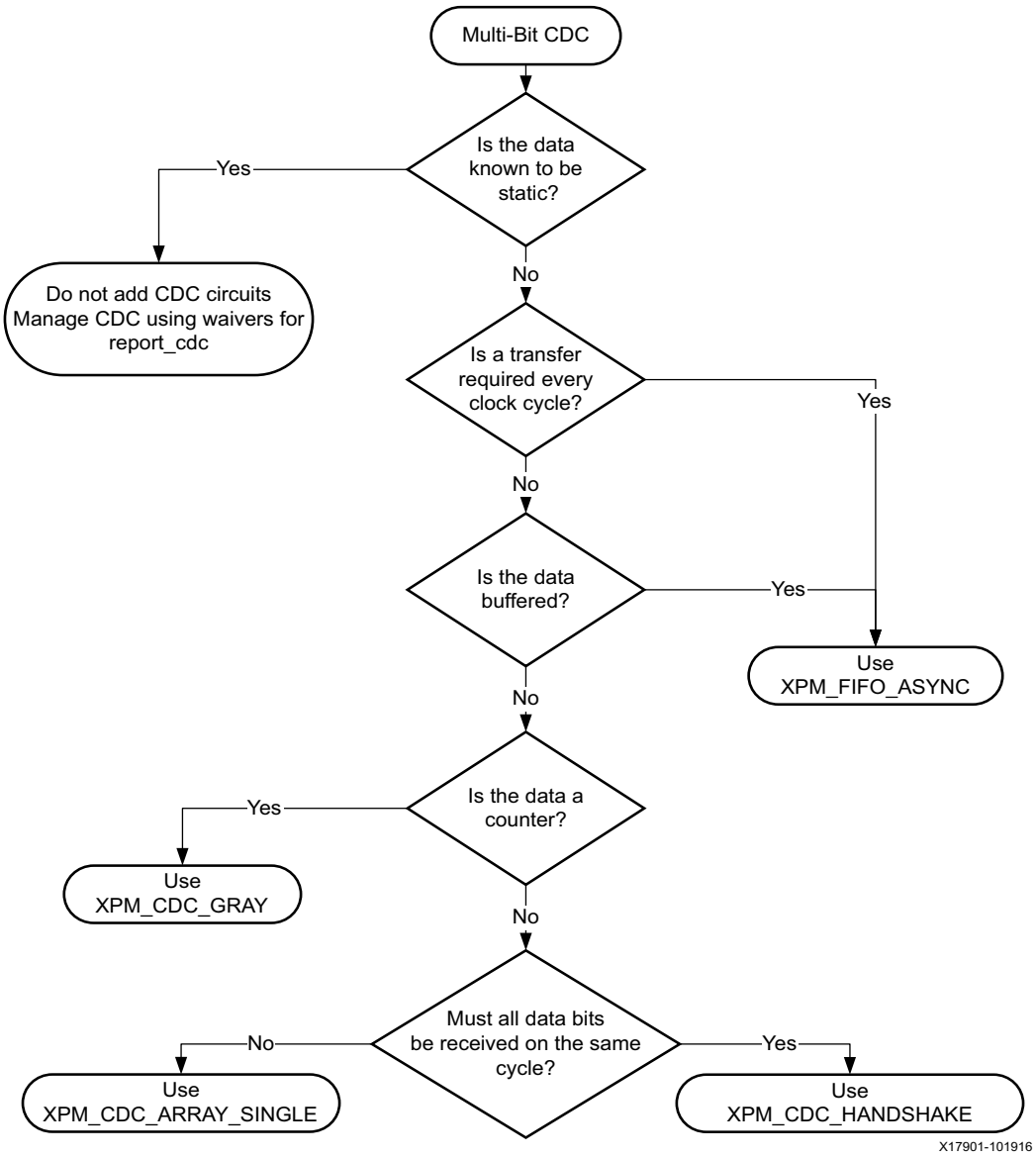


图 3-62： 多比特 CDC 决策树

注释： 如需了解更多有关不同的多位同步器的信息，请参阅 《Vivado Design Suite 7 系列 FPGA 和 Zynq-7000 All Programmable 库指南》(UG953) [参照 28] 与 《UltraScale 架构库指南》(UG974) [参照 29]。

优化 MTBF

设计的总 MTBF 是以下函数：

- 同步器 MTBF
- 由于单事件故障 (SEUs) 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
- 每个交叉点处的同步器级数
- 目标 FF 的频率
- 切换源的速率

为 D EST_SYNC_FF 参数选择正确的值

当使用 XPM CDC 模块时，DEST_SYNC_FF 参数设置亚稳态保护寄存器的数量。该寄存器的值影响 MTBF，设计大小和交叉点处的延迟。选择此寄存器的正确值是一个迭代过程，需要以下内容：

1. 通过 Vivado Design Suite 实现流程运行设计。
2. 根据您的目标器件，执行以下操作之一：
   - 对于 7 系列器件，请选择 DEST_SYNC_FF 的默认值。这是满足典型可靠性要求的保守方法。对于关键设计，进行进一步分析。
   - 对于 UltraScale 器件，请运行 report_synchonizer_mtbf 命令，该命令报告整个设计的 MTBF。通过迭代流程，如下图所示，您可以找到 MTBF，延迟和资源之间的适当权衡。

注释：您还可以将这个迭代过程用于用户 CDC 电路，其中 ASYNC_REG 属性正确应用于所有同步寄存器。

图 3-63：UltraScale 器件的同步器 MTBF 优化流程

正确地约束设计

XPM CDCs 提供自己的 `set_max_delay -datapath_only` 约束。XPM CDC 与 `set_clock_groups` 约束不兼容，set_clock_groups 约束具有更高的优先级，并将覆盖 XPM 中的约束。如需了解更多信息，请参阅“定义时钟组和 CDC 约束”。

充分利用 IP 核

使用预先验证的 IP 核能够大幅减少设计和验证工作量，从而加速产品上市进程。如需了解更多有利用 IP 的信息，请参阅以下资源：

- 《Vivado Design Suite 用户指南：采用 IP 进行设计》(UG896) [参照 9]
- 《Vivado Design Suite 用户指南：采用 IP 集成器设计 IP 子系统》(UG994) [参照 26]
- Vivado Design Suite QuickTake 视频教程：在 Vivado 中配置和管理可重用 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。

- 根据所需功能以及其它设计目的评估赛灵思或其它第三方合作伙伴提供的 IP 选项，例如：
  - 与现可用的 IP 核相比，定制逻辑是否更好？
  - 用业界标准格式封装定制设计，便于在多个工程中重复使用是否有意义？
- 考虑需要使用的接口，比如存储器接口、网络接口和外设接口。

重要提示：为确保这些工具进程正确地处理 IP 专用约束，为项目添加 .xci 或 .xcix IP 源文件。在使用 IP 工作时，勿将 IP 生成的输出 DCP 文件作为项目源。

AMBA AXI

赛灵思已对符合开放式 ABMA® 4 AXI4 互联协议的 IP 接口进行了标准化。这种标准化能够简化赛灵思和第三方提供商提供的 IP 的集成工作并最大化系统性能。为有效地映射到自己的 FPGA 器件架构中，赛灵思与 ARM 共同制定了 AXI4、AXI4-Lite 和 AXI4-Stream 规范。

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

AXI-Stream 用于从主设备到从设备的单向数据流。典型应用包括 DSP、视频和通信。

Vivado Design Suite IP 目录

IP 目录是存放赛灵思提供的 IP 核的唯一地方。您可在 IP 目录中找到用于嵌入式系统、 DSP、通信和接口等的 IP 核。

在 IP 目录中可以查阅所有可用 IP 核，阅读有关任何 IP 的产品指南、变更日志、产品网页和问答记录。

可以通过 GUI 或 Tcl shell 访问和自定义 IP 目录中的 IP 核。 Tcl 脚本能够自动完成 IP 核的自定义工作。

定制 IP

赛灵思使用业界标准的 IP-XACT 格式交付 IP，并提供 IP 封装器，用于封装定制 IP。相应地，您也可以把自己定制 IP 核添加到 IP 目录中，并创建可供团队或整个公司共享的 IP 库。来自第三方提供商的 IP 也可以添加到此目录中，前提是它封装在 IP 封装器中，即使它已经是 IP-XACT 格式。

从 IP 目录选择 IP

所有赛灵思和第三方厂商的 IP 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。根据该编目方法可以浏览 IP 目录，查看自己感兴趣区域的 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 目录会告知用户 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 (Fmax) 是多少？

- 设计中所需的与板上辅助芯片通信的外部接口标准：
  - 以太网、 PCIe 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。
  - Aurora 等赛灵思专有接口。

    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。例如：
  - AXI4
  - AXI4-Lite
  - AXI4-Stream
- 如果涉及多重协议，要使用 IP 目录中的基础架构 IP，可能必须选择桥接 IP 核。例如：
  - AXI-AHB 桥接
  - AXI-AXI 互联
  - AXI-PCIe 桥接
  - AXI-PLB 桥接

自定义 IP

可通过 GUI 或通过 Tcl 脚本自定义 IP。

- “使用定制 GUI”
- “使用 Tcl 脚本”

使用定制 GUI

使用图形界面是查找、搜索和自定义 IP 的最简单的途径。每个 IP 都有为其自定义的一套标签或页面。同时提供相关的配置选项。如下图所示，定制窗口的示例。为 IP 创建唯一的定制方案，生成对应的 XCI 文件。随后用这个 XCI 文件就可以为 IP 生成各种类型的输出结果。

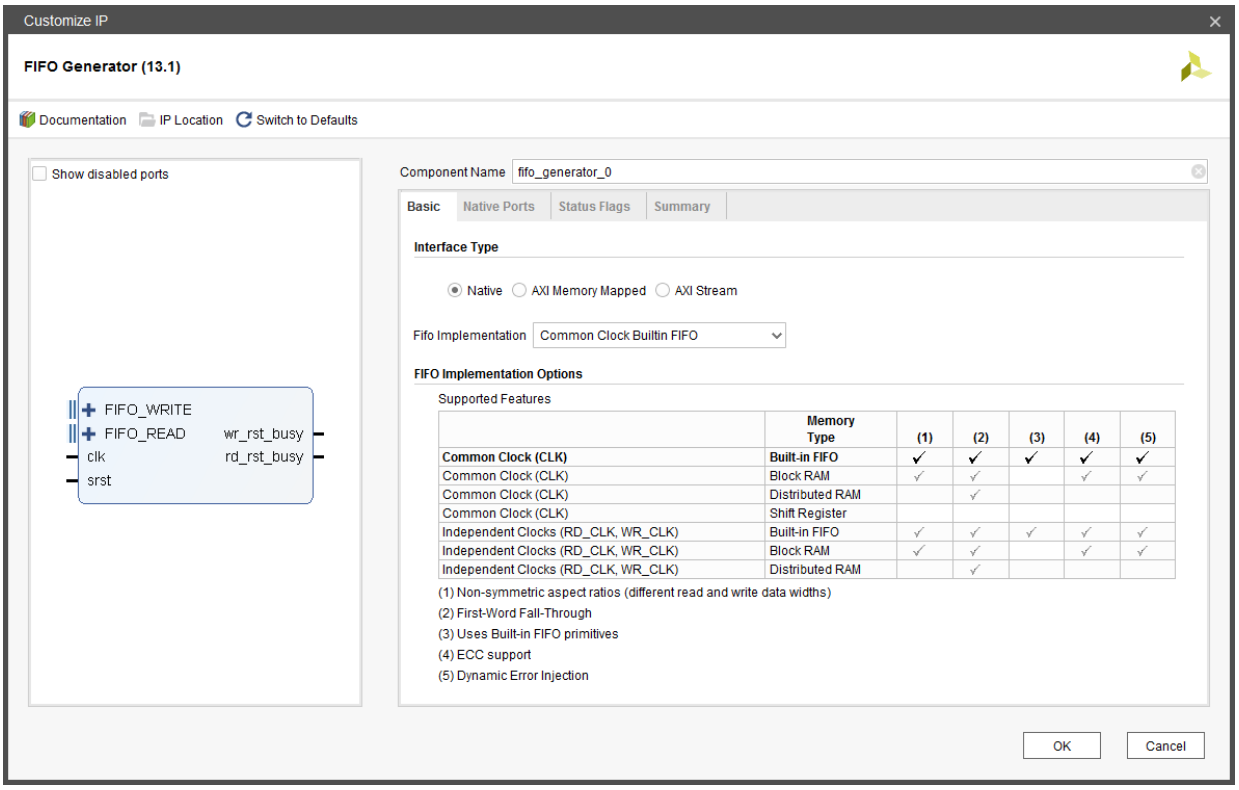


图 3-64：IP 定制窗口

使用 Tcl 脚本

基本上每个 GUI 操作都会发出一条 Tcl 命令。IP 创建包括设置所有定制选项，无需用户干预，即可用 Tcl 脚本自动完成。

用户需要知道配置选项的名称及允许设置的值。一般情况下，用户首先通过 GUI 定制 IP，然后创建脚本。在生成 Tcl 脚本之后，用户就可以方便地根据自己的需要修改脚本，比如修改数据大小。

采用 Tcl 脚本创建 IP 方便实现自动化，比如在使用版本控制系统的时候。如需了解更多有关源代码管理和版本控制的信息，请参阅 《Vivado Design Suite 用户指南：设计流程简介》(UG892) [参照 5] 中的链接。

IP 版本和版本控制

IP 自定义完成后，工具会生成一个包含所有所选参数值的 XCI 文件。每个 Vivado IDE 版本仅支持一个版本的 IP。因此赛灵思建议用户使用最新的 IP 版本。如果用户使用较早的 IP 版本，就应保存较老版本的全部输出结果。如需了解更多有关源代码管理和版本控制的信息，请参阅《Vivado Design Suite 用户指南：设计流程简介》(UG892) [参照 5] 中的链接。

重要提示：对于 7 系列器件中的存储器 IP，除了 XCI 文件之外，还会创建 PRJ 文件。当使用具有 7 系列存储器 IP 的版本控制时，请将 PRJ 文件保存在与 XCI 文件相同的目录中。

利用约束

编制设计约束

设计约束定义了编译流程中必须满足的要求，以使设计方案在硬件中能够正常运行。对于更加复杂的设计，还需要为工具定义便于实现收敛的实施指南。并非所有约束都要在编译流程中的所有步骤中使用。例如，物理约束只在实现阶段使用 （即，由布局器和布线器使用）。

由于综合与实现算法均基于时序，因此建立适当的时序约束至关重要。对于您的设计实施过约束或欠约束都会让时序收敛变得困难。因此您必须使用与应用要求一致的合理约束。如需了解更多有关约束的信息，请参阅以下资料：

- 《Vivado Design Suite 用户指南：设计分析和收敛技术》(UG906) [参照 21]
- 可在赛灵思的网站 Vivado 设计套件视频教程页面上观看“应用设计约束”视频教程

一般按照类别或设计模块 （或二者兼有）将约束编制到一个或多个文件中。无论您如何编制约束，都必须理解它们的整体相关性，并在载入内存时检查它们的最终时序。例如，由于时序时钟必须在被其它约束使用之前进行定义，因此必须确保将时序时钟的定义放在约束文件的开始位置，或位于加载到内存的第一批约束文件中，或二者兼有。

建议的约束文件

有很多编制约束的方法可供选择，这要取决于工程的大小和复杂程度。下面给出一些建议。

简单设计

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 （多个 IP 核或多个设计团队）：

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

验证读取顺序

一旦完成工程约束文件的编制工作，就必须根据文件内容验证文件的读取顺序。在工程模式下，您可以在 Vivado IDE 中或使用 `reorder_files` Tcl 命令修改约束文件顺序。在非工程模式下，该顺序可直接由编译流程 Tcl 脚本中的 `read_xdc` （针对 XDC 文件）和 `source` （针对 Tcl 脚本生成的约束）命令来定义。

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 属于时序语言：

- 必须在使用前，首先定义变量。同样，时序时钟必须在被其它约束使用之前进行定义。
- 对于覆盖相同路径并具有相同优先级的等效约束，应最后一个应用。

当考虑以上优先规则时，时序约束应依照以下顺序：

```
## 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，尤其要注意时钟定义问题，并验证相关性是否正确排序。

物理约束可以位于任意约束文件的任意位置。

创建综合约束

综合步骤收到设计的 RTL 描述，并利用时序驱动的算法将其转变成最佳技术映射网表。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，网络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）。这里的主要目的是获得具有真实和简单约束且满足时序要求或差一点满足时序要求的网表。

综合引擎接受所有 XDC 命令，但只有一部分真正有效：

- 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 属性强制采用映射和优化算法来制定决策。以下提供一些实例：
  - DONT_TOUCH/KEEP/KEEP_HIERARCHY/MARK_DEBUG
  - MAX_FANOUT
  - RAM_STYLE/ROM_STYLE/USE_DSP48/SHREG_EXTRACT
  - FULL_CASE/PARALLEL_CASE(仅限 Verilog RTL)

  注释：同一属性还可被设置为 XDC 文件的特性。使用基于 XDC 的约束便于仅在特定情况且不改变 RTL 的前提下影响综合结果。
- 物理约束被忽略 （LOC、 BEL、 Pblock）

综合约束使用的名称必须来自细化的网表、更好的端口和时序单元。一些 RTL 信号会在细化过程中消失，因此无法为它们赋予 XDC 约束。此外，由于细化后有多种不同优化，因此网络或逻辑单元会并入不同原语，例如 LUT 或 DSP 块。要了解详细设计对象的名称，请单击 Flow Navigator 中的“Open Elaborated Design”，然后浏览您感兴趣的层次结构。

有些寄存器被吸收到 RAM 块中，而且有些层级级数会消失以实现夸边界优化。

任何细化的网表对象或层级级数都可利用 DONT_TOUCH、 KEEP、 KEEP_HIERARCHY 或 MARK_DEBUG 约束进行保存，但存在降低时序或占位面积 QoR 的风险。

最后，有些约束存在冲突或不被综合认可。例如，如果在跨越多个层级级数的网络上对 MAX_FANOUT 属性进行设置，而且有些层级通过 DONT_TOUCH 进行保存，那么扇出优化将受到限制或被完全阻止。

重要提示：与实现阶段不同，综合阶段可对用于定义时序约束的 RTL 网表对象实行优化，以获得更好的 QoR。只要约束针对实现进行了更新和验证，这通常不会有问题。如果需要，还可以使用 DONT_TOUCH 约束保存任意对象，以便在综合与实现阶段均可应用约束。

一旦综合完成后，赛灵思建议您检查时序和使用报告，以验证网表质量满足应用要求并可用于实现阶段。

创建实现约束

实现约束必须准确反应最终应用的要求。物理约束 （例如 I/O 位置和 I/O 标准）决定于单板设计，包括单板走线延迟，以及源自总体系统要求的散热管理要求。在进入实现步骤之前，赛灵思强烈建议您对所有约束的正确性和准确性进行验证。错误约束可能会降低实现的 QoR 以及对于时序验收质量的信心。

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，因此必须核实所有综合约束可在实现网表中正确使用。如果不是这样，您必须创建一个附加 XDC 文件，用以存储仅在实现阶段有效的约束。

将块级约束读入顶层设计

当开发多团队工程时，为方便起见应为顶层设计的每个主要块创建独立的约束文件。这些块在最终整合为一个或多个顶层设计之前通常需要单独开发和验证。

块级约束在开发时必须独立于顶层约束，且必须尽可能多地具备通用性，以便能够用于不同环境。此外，它们必须不影响任何块边界之外的逻辑。

当实现子块时，期望使全时序网络包括在时序分析中以确保精确的偏差和跨时钟域分析。这可能需要一个包含时钟组件和附加约束文件的 HDL 包装器来复制顶级时钟约束。它仅用于子模块的时序验证。

如需了解更多有关约束作用域以及加载块级别的规则，准则和机制的信息，请参阅 《Vivado Design Suite 用户指南：如何使用约束》(UG903) [参照 18] 中的链接。

分四个步骤定义时序约束

好的约束定义过程分为四个主要步骤，如下列图表所示：这些步骤遵循时序约束的优先和从属规则，并以合乎逻辑的方式向时序引擎提供信息以执行分析功能。

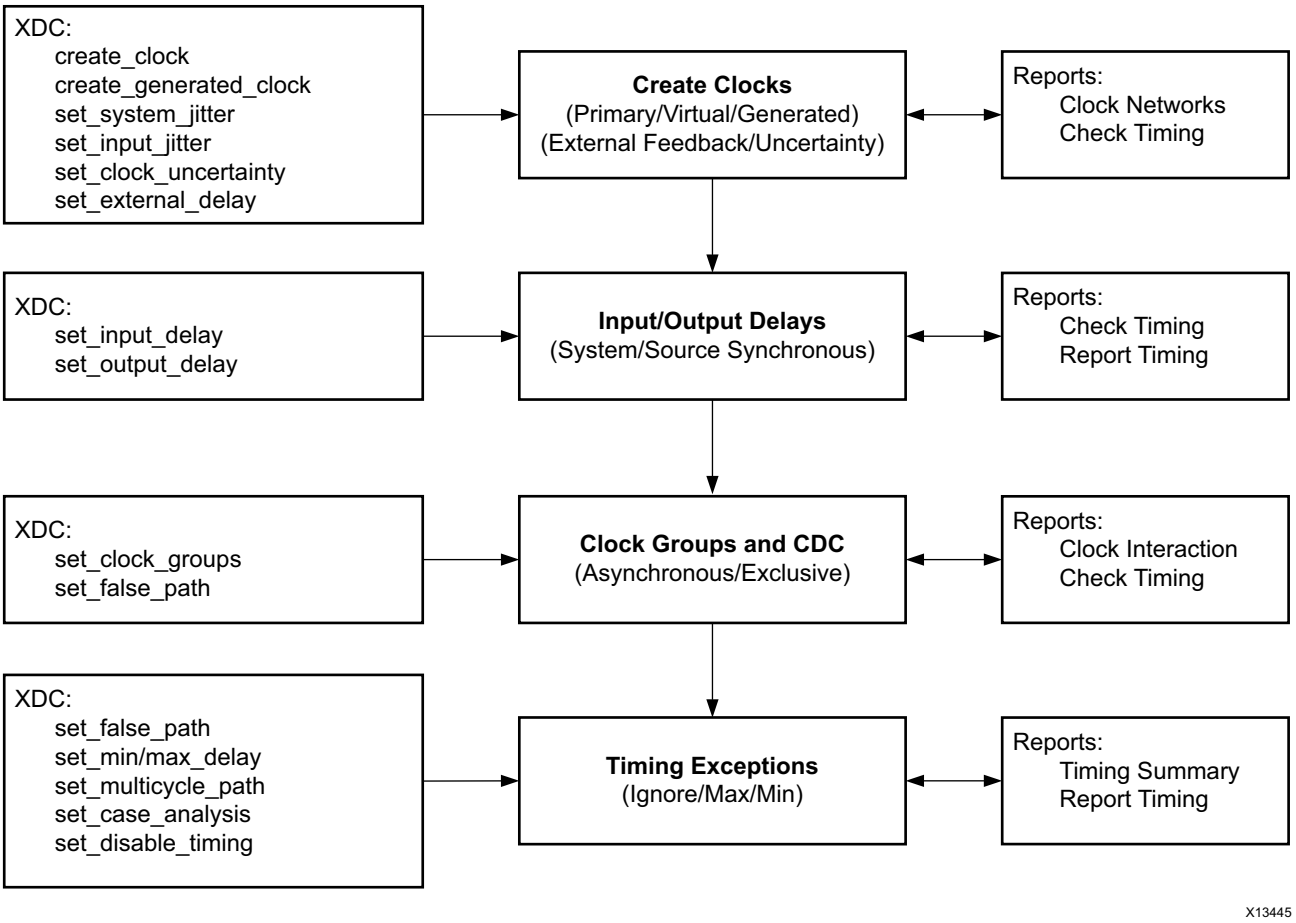


图 3-65：时序约束制定步骤

- 前两个步骤称为时序激活阶段，用来从时钟波形和 I/O 延迟约束中调用出默认时序路径要求。
- 在第三个步骤中，对至少共享一个逻辑路径的异步或专用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。
- 最后一个步骤相当于时序例外，设计人员可以利用特定约束来忽略、减轻或加强默认的时序路径要求。

约束创建与约束识别和约束验证任务息息相关，这些任务必须通过时序引擎生成的各种报告才能实现。时序引擎仅能使用经过完全映射的网表，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个约束集，以便约束的分析和报告可以交互执行。

创建新设计的时序约束或完善现有约束时，赛灵思建议使用 Timing Constraints Wizard 以快速识别缺失的约束。这适用于图 3-65 中的前三步。 Timing Constraints Wizard 按照本节介绍的方法可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。更多关于 Timing Constraints Wizard 的信息请参阅 《Vivado Design Suite 用户指南：如何使用约束》(UG903) [参照 18]。

下面几节详细介绍以上的四个步骤：

- “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”

在约束创建过程中的相应步骤可参考相应章节以获得详细的方法以及使用案例。

定义时钟约束

必须首先定义时钟，以便为其它约束所用。时序约束创建流程的第一步是确定必须在哪里定义时钟以及应该定义为“主时钟”还是“生成时钟”。

重要提示：定义具有特定名字的时钟时 （-name 选项），必须确认时钟名称未被另一个时钟约束所使用或被已有的自动生成时钟使用。当一个时钟名称用于多个时钟约束时， Vivado Design Suite 时序引擎会发出提示，提警告您第一个时钟定义被覆盖。如果同一个时钟名称使用两次，那么第一个时钟定义就会丢失，所有对应于该名称、在两个时钟定义间输入的约束也会丢失。赛灵思建议您避免覆盖时钟定义，除非能确保其他约束均不受影响，而且所有时序路径都保持受约束。

识别时钟源

用下面的两个报告识别设计中未约束的时钟源：

- “时钟网络报告”
- “检查时序报告”

时钟网络报告

约束和未约束的时钟源点在两个不同类别中列出。对于每个未约束的时钟源点，必须确定定义为主时钟还是生成时钟。

```
% 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 检查可报告没有时钟定义的生效叶节点 (leaf) 时钟引脚组。每个组关联一个时钟源点，而且必须在该点定义一个时钟以清除此问题。

```
% 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)
```

利用 check_timing，可以使相同时钟源引脚或端口出现在若干个组中，具体数量取决于整个时钟树的拓扑结构。这种情况下，在选定时钟源引脚或端口建立一个时钟就可以解决所有相关组中遗漏时钟定义的问题。

如需了解更多信息，请参阅第 5 章中的“检查设计是否正确约束”。

创建主时钟

主时钟是指用于为设计定义时序参考的时钟，而时序引擎可利用主时钟获取时序路径要求以及与其它时钟的相位关系。计算主时钟插入延迟时应从时钟源点 （定义主时钟的驱动器引脚/端口位置）开始，一直到时钟扇出所至时序单元的时钟引脚。

基于这个原因，定义主时钟时很重要的一点是要将主时钟定义在与设计边界相对应的对象上，这样主时钟的延迟以及间接条件下的偏差，都可以得到精确计算。

典型的主时钟根包括：

- “输入端口”
- “7 系列器件中千兆位收发器输出引脚”
- “某些硬件原语输出引脚”

输入端口

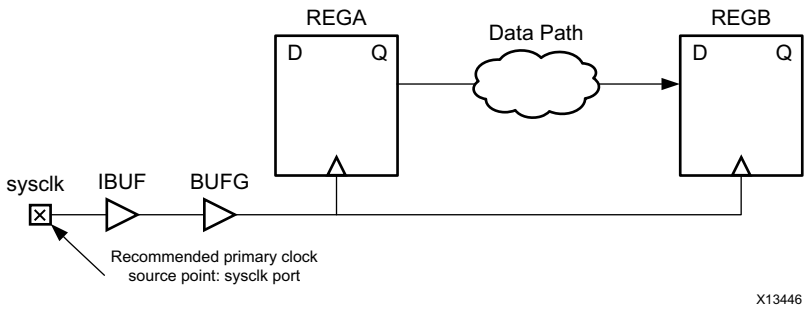


图 3-66：输入端口上的 create_clock

约束实例：

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

该实例中，波形的占空比设定为 50%。上面给出了变量 -waveform 的用法，并且只有在定义占空比不是 50% 的时钟时才有必要使用该变量。对于差分时钟输入缓存，只需在差分对的 P 侧对主时钟进行定义。

7 系列器件中千兆位收发器输出引脚

千兆位收发器输出引脚，例如已恢复的时钟：

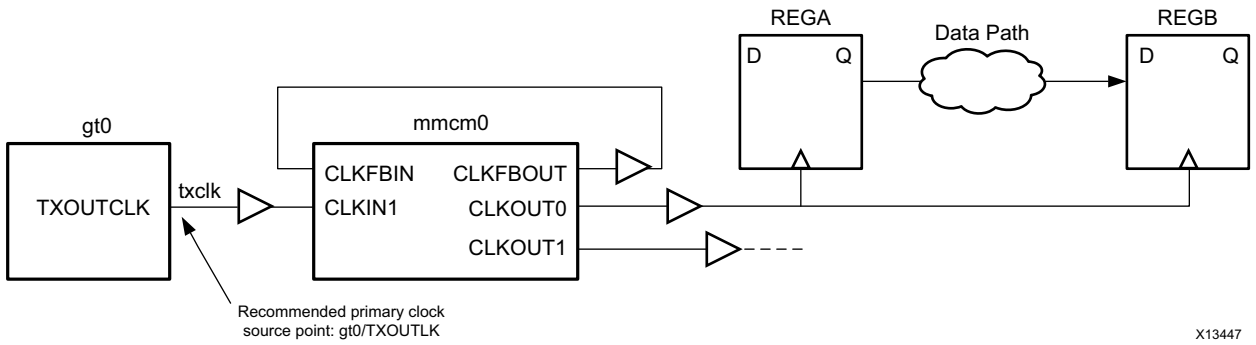


图 3-67：原语引脚上的 create_clock

约束实例：

```
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，赛灵思还建议定义 GT 输入时钟，因为 Vivado 工具能够计算 GT 输出引脚上的预期时钟，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，工具会发出方法检查警告。

注释：对于面向 UltraScale 器件的设计，赛灵思不建议在 GT 的输出上定义主时钟，因为在定义相关电路板输入时钟时，将自动导出 GT 时钟。

某些硬件原语输出引脚

某些硬件原语，例如， BSCANE2 的输出引脚不含连接同一原语输入引脚的时序 arc。

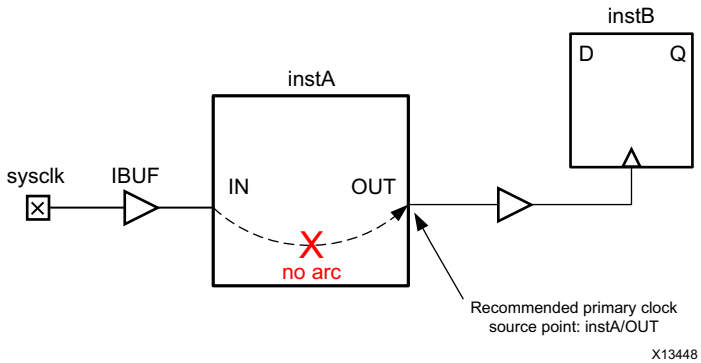


图 3-68：时钟路径因缺失时序 arc 而断开

重要提示：在主时钟传递扇出中不应定义另外一个主时钟，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。

下图给出了一个示例，其中时钟 `clk1` 在时钟 `clk0` 的传递扇出中定义；`clk1` 从其定义位置 BUFG1 的输出开始覆盖 `clk0`。因此， REGA 与 REGB 之间的时序分析就不会准确，因为 `clk0` 和 `clk1` 之间存在无效的偏差计算。

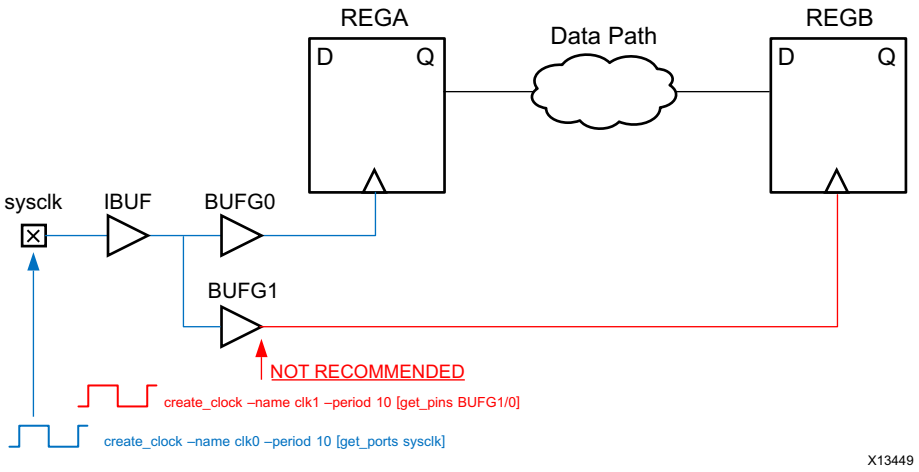


图 3-69：不建议在另一个时钟的扇出中使用 create_clock

创建生成时钟

生成时钟源自另一个现有时钟 （主时钟）。通常用来描述由逻辑块在主时钟上执行的波形变换。由于生成时钟的定义取决于主时钟特性，因此必须首先定义主时钟。要明确定义生成时钟，必须使用 `create_generated_clock` 命令。

自动衍生时钟

大部分生成时钟都由 Vivado Design Suite 时序引擎自动衍生获得，该引擎可识别 Clock Modifying Block (CMB) 及其对主时钟所执行的变换。

赛灵思 7 系列器件中， CMB 为：

- MMCM*/PLL*
- BUFR
- PHASER*

赛灵思 UltraScale 器件系列中， CMB 为：

- 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它组合单元而言，时序时钟可通过它们进行传输，且无需在输出端重新定义，除非波形已被相应单元转换。通常应该尽可能多地依靠自动衍生机制，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 Vivado Design Suite 时序引擎所选择的自动衍生时钟名称并不合适，您可以使用 `create_generated_clock` 命令强行定义自己的名称，此时无需指定波形转换。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，由 MMCM 实例生成的时钟的默认名称是 net0，您可以添加如下约束强制将其设定为自己的名称 （此例中是 `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引脚。如需了解更多信息，请参阅 《Vivado Design Suite 用户指南：如何使用约束》(UG903) [参照 18]。

用户定义的生成时钟

一旦所有主时钟都完成定义，您就可以利用 (no_clock) 报告来识别不含时序时钟的时钟树部分，并相应地定义生成时钟。

有时候很难理解逻辑椎 (cone) 对主时钟执行的变换。这种情况下必须采用最保守的约束。例如，源引脚是时序单元输出。主时钟至少除以 2，因此正确的约束应当为：

```
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，那么锁存器门控引脚也需要连接时序时钟，如果约束缺失， Check Timing (no_clock) 将会报告相应的锁存器问题。您可以按照上面的实例来定义这些时钟。

主时钟与生成时钟间的路径

与主时钟不同，生成时钟必须在主时钟的传递扇出中进行定义，这样时序引擎就能精确计算它们的插入延迟。若不遵守这个原则会导致错误的时序分析，而且很有可能导致无效的时序裕量。例如，在下图中 gen_clk_reg/Q 作为下一个寄存器 (q_reg) 的时钟，而且还位于主时钟 c1 的扇出椎中。因此， gen_clk_reg/Q 上应具有 create_generated_clock，而不是 create_clock。

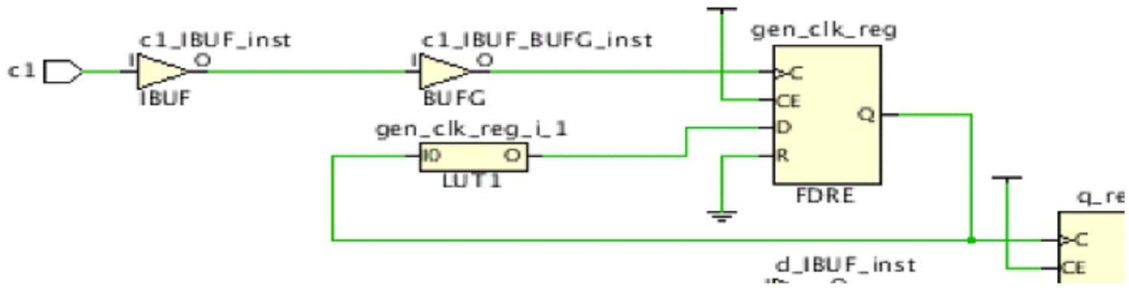


图 3-70：主时钟扇出中的生成时钟

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

核实时钟定义与覆盖范围

一旦所有设计时钟都定义完毕并且应用于内存之后，您就可以核实每个时钟的波形，并使用 report_clocks 命令查看主时钟与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/mmcm_adv_inst/CLKFBOUT}
cpuClk       20.00000  {0.00000 10.00000}   P,G         {clkgen/mmcm_adv_inst/CLKOUT0}
…
==================================================
Generated Clocks
==================================================

Generated Clock    : cpuClk
Master Source      : clkgen/mmcm_adv_inst/CLKIN1
Master Clock       : sysClk
Edges              : {1 2 3}
Edge Shifts        : {0.000 5.000 10.000}
Generated Sources : {clkgen/mmcm_adv_inst/CLKOUT0}
```

此外，您还可以核实所有内部时序路径是否至少被一个时钟覆盖。 check_timing 报告可以为此提供两项检查内容：

- **no_clock**

  报告定义时钟没有达到的任何活动时钟引脚。

- **unconstrained_internal_endpoint**

  若时序单元具有与时钟有关的时序检查且这个时钟尚未定义，则会报告此类时序单元的所有数据输入引脚。

如果两项检查都返回零，说明时序分析覆盖范围广。

您也能运行 XDC 和 Timing Methodology 检查来验证是否所有时钟定义在建议网表对象上，同时不会造成任何约束冲突或不准确的时序分析情境。

如果您在使用 Vivado Design Suite2016.3 之前版本，请使用以下命令运行以下检查：

```
report_drc -checks [get_drc_checks {XDC* TIMING-*}]
```

如果您使用的是 Vivado Design Suite 版本 2016.3 或更高版本，请使用以下命令运行以下检查：

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

如需了解更多信息，请参阅第 4 章中的“运行报告方法”。

调整时钟特性

在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有关的信息。 XDC 语言可将与抖动和相位误差有关的不确定性从与偏差和延迟建模有关的不确定性中区分开来。

- “抖动”
- “更多不确定性”
- “时钟源位置的时钟延迟”
- “MMCM 或 PLL 外部反馈回路延迟”

抖动

对于抖动，最好使用 Vivado Design Suite 的默认值。按下列方法修改默认计算：

- 如果随机抖动大于 0 的主时钟输入器件，请使用 `set_input_jitter` 命令以设定抖动值。
- 如果器件电源有噪声，想要调整全局抖动，应使用 `set_system_jitter`。赛灵思不建议增加默认的系统抖动值。

对于生成时钟，抖动源自主时钟和时钟修改块的特性。用户不需要调整这些数字。

更多不确定性

当您需要在时钟的时序路径上或两个时钟之间添加额外裕量时，必须使用 `set_clock_uncertainty` 命令。这也是对设计某个部分进行过约束且不改变实际时钟边缘和整体时钟关系的最佳、最安全的方法。用户定义的时钟不确定性会增加 Vivado 工具计算的抖动，而且可针对建立和保持分析单独进行指定。

例如，设计时钟 clk0 全部时钟间路径的裕量需严格地设置在 500 ps，以使设计的建立和保持抗噪声能力更强：

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

如果您在两个时钟之间指定更多不确定性，那么约束必须同时应用于两个方向 （假设数据向两个方向流动）。下面的例子展示了如何仅针对设置而将 `clk0` 和 `clk1` 之间的不确定性增大 250 ps：

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

时钟源位置的时钟延迟

可使用带 `-source` 选项的 `set_clock_latency` 命令对时钟源位置的时钟延迟进行建模。该方法在两种情况下有用：

- 指定器件外部与输入和输出延迟约束无关的时钟延迟传输。
- 对块在无关联 (OOC) 编译过程中使用的时钟的内部传输延迟进行建模。在这样的编译流程中，未对完整时钟树进行描述，因此无法自动计算块外部最小和最大运行条件之间的差异，必须进行手动建模。

此约束只能由高级用户使用，因为通常很难提供合法的延迟数值。

MMCM 或 PLL 外部反馈回路延迟

当连接 MMCM 或 PLL 反馈回路用以补偿单板延迟 （而非内部时钟插入延迟）时，必须使用 `set_external_delay` 命令指定最好和最差情况下 FPGA 器件外部延迟。未指定此延迟将使 MMCM 或 PLL 有关的 I/O 时序分析变得无关紧要，并可能导致时序收敛无法实现。此外，当使用外部补偿时，必须相应地调整输入和输出延迟增益值，而不是仅仅考虑正常情况下单板上的时钟走线延迟。

约束输入和输出端口

除了指定设计中每个端口的位置和 I/O 标准以外，还必须指定输入和输出延迟约束，以描述进/出 FPGA 器件接口的外部路径时序。定义这些延迟所对应的时钟通常也在单板上生成并进入 FPGA 器件。在某些情况下，即当 I/O 路径和具有与单板时钟不同波形的时钟有关时，这些延迟必须定义与虚拟时钟有关。

系统级视角

I/O 路径与任何其它 `reg-to-reg` 路径一样由 Vivado Design Suite 时序引擎进行建模，除非这部分路径位于 FPGA 器件外部并需要由用户来进行描述。当分析内部路径时，应在建立和保持分析时考虑最小和最大延迟。这对于 I/O 路径

来说也是如此。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，也就是：

- 对于最大延迟分析 （建立），单数据速率接口在发送沿后的一个时钟周期捕获数据；而双数据速率接口在发送沿后的半个时钟周期捕获数据。
- 对于最小延迟分析 （保持），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计时 （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。这相当于高级 I/O 时序约束方案。

定义输入延迟

输入延迟相对于器件接口处的时钟进行定义。除非已经在参考时钟的源引脚上指定了 `set_clock_latency`，否则输入延迟相当于从发送沿到时钟走线、外部器件和数据走线的绝对时间。如果已单独指定时钟延迟，那么就可以忽略时钟走线延迟。

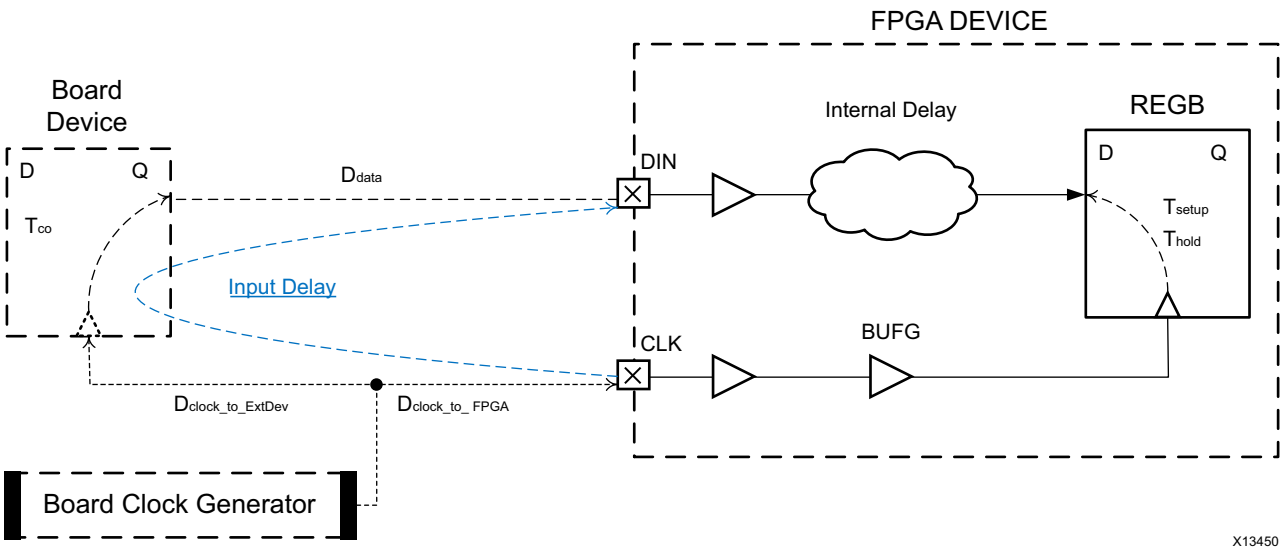


图 3-71：输入延迟计算

两类分析的输入延迟数值：

```
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)
```

下图是一个解读最小和最大输入延迟给出了建立 （最大）和保持 （最小）分析中输入延迟约束的简单示例，假设已在 CLK 端口上对 sysClk 时钟进行定义：

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

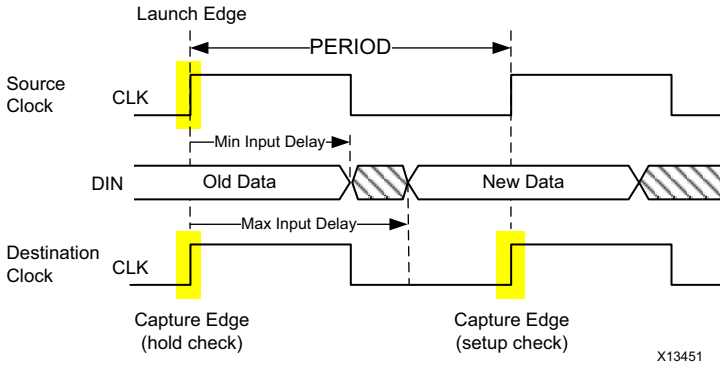


图 3-72：解读最小和最大输入延迟

负输入延迟意味着数据在发送时钟沿之前到达器件接口。

定义输出延迟

输出延迟与输入延迟类似，除非指的是为了在所有条件下起作用的 FPGA 器件外的输出路径最小和最大时间。

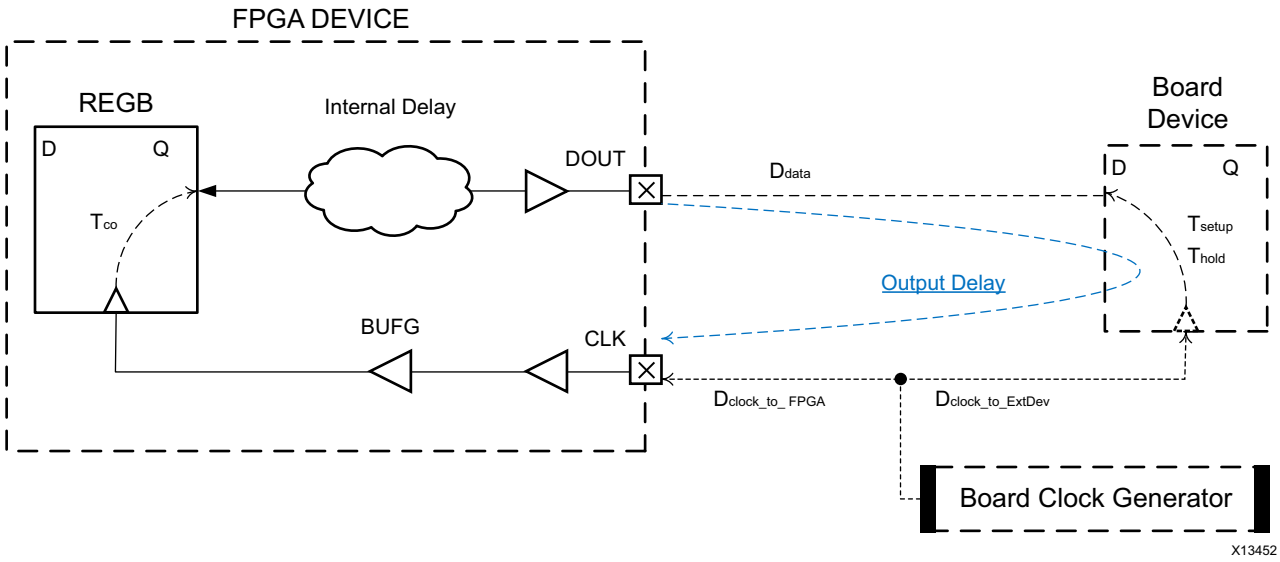


图 3-73：输出延迟计算

两类分析的输出延迟数值：

```
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)
```

解读最小和最大输出延迟给出了建立 （最大）和保持 （最小）分析中输入延迟约束的简单示例，假设已在 CLK 端口上对 sysClk 时钟进行定义：

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

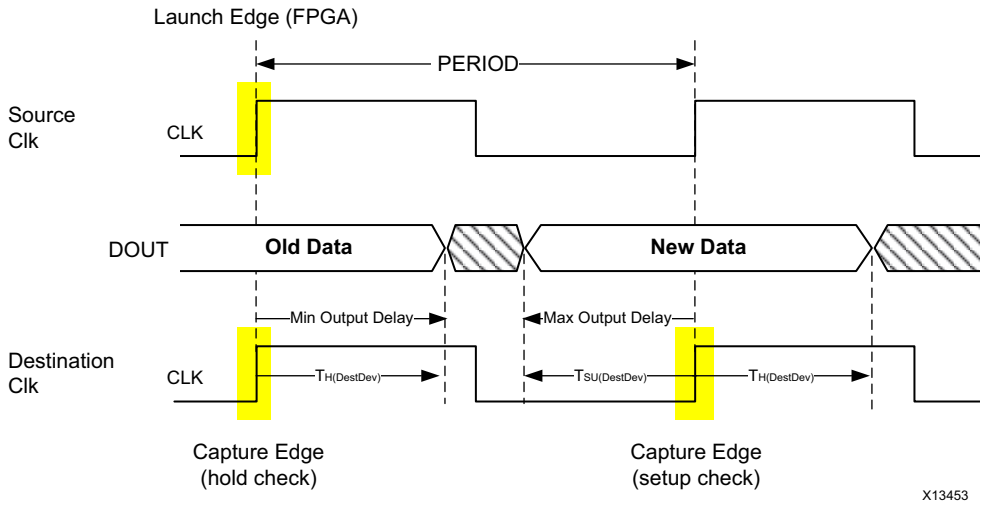


图 3-74：解读最小和最大输出延迟

输出延迟相当于捕获沿之前单板上的延迟。对于时钟和数据单板走线比较平衡的一般系统同步接口而言，目标器件的建立时间可定义最大分析的输出延迟值。目标器件保持时间可定义最小分析的输出延迟。指定最小输出延迟表示信号从设计中出来以后在用于目标器件接口的保持分析之前所引发的最小延迟。因此，块内部的延迟可以小很多。最小输出延迟正值表示信号在设计内部可具有负延迟。这就是为什么最小输出延迟经常是负值的原因。例如：

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

表示设计内部直到 DOUT 的延迟必须至少超过 0.5 ns，才能满足保持时间的需要。

选择参考时钟

应根据用于控制时序单元 （这些单元与输入或输出端口有关）的时钟树拓扑结构情况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，那么延迟约束通常需要根据主时钟设置，也就是生成时钟的上游定义。这条规则存在部分例外，本节将做出解释。

识别与每个端口有关的时钟

在定义 I/O 延迟约束之前，必须首先识别哪些时钟与端口相关联。有几种方法可用来识别这些时钟：

- “浏览单板原理图”
- “浏览设计原理图”
- “报告进出端口的时序”
- “使用自动识别的采样时钟”

浏览单板原理图

对连接到单板上特定器件的一组端口来说，您可用能够连接器件和 FPGA 的同一个单板时钟作为输入或输出延迟参照时钟。您需要在器件数据手册中确认单板时钟是否为 I/O 端口计时进行了内部转换，以确保 FPGA 设计生成相同的时钟来控制相关端口组的时序。

浏览设计原理图

对于每个端口，可将路径原理图扩展至时序单元的第一层，然后将这些单元的时钟引脚追溯至时钟源。对于连接高扇出网络的端口而言，这种方法无法实现。

报告进出端口的时序

无论端口是否经过约束，都可以使用 report_timing 命令识别其在设计中的相关时钟。一旦所有时序时钟都定义完毕，就可以报告进出 I/O 端口的最差路径，创建与报告时钟有关的 I/O 延迟约束，并重新运行设计中其它时钟的相同时序报告。如果端口似乎与多个时钟关联，应建立相应的约束并重复此过程。

例如， din 输入端口关联设计中的 clk1 和 clk2 时钟：

```
report_timing -from [get_ports din] -sort_by group
```

报告显示 din 端口与 clk1 关联。输入延迟约束为 （同时适用于该实例中的最小和最大延迟）：

```
set_input_delay -clock clk1 5 [get_ports din]
```

采用与之前相同的命令重新运行时序分析，并观察到 din 也关联于 clk2，这是因为 -sort_by group 选项可报告每个端点时钟的 N 条路径。您可以添加相应的延迟约束，并重新运行报告以验证 din 端口与其它时钟没有关联。

您还可以使用 -report_unconstrained 选项，用“时序总结报告”功能运行相同的分析。由于设计中只有时钟约束，则该部分内容如下所示：

```
------------------------------------------------
| Unconstrained Path Table
------------------------------------------------

Path Group      From Clock      To Clock
----------      ----------      --------
(none)
(none)          clk1
(none)          clk2
(none)                          clk1
(none)                          clk2
```

没有时钟名 （或 Vivado IDE 中的<NONE>）的字段是指起点 (From Clock) 或端点 (To Clock) 未关联时钟的一组路径。未约束的 I/O 端口属于此类情况。可通过浏览报告的剩余部分来对它们的名称进行检索。例如，在 Vivado IDE 中，通过选择 clk1 到 NONE 类别的 Setup 路径，就可以在“To”列中看到由 clk1 驱动的端口：

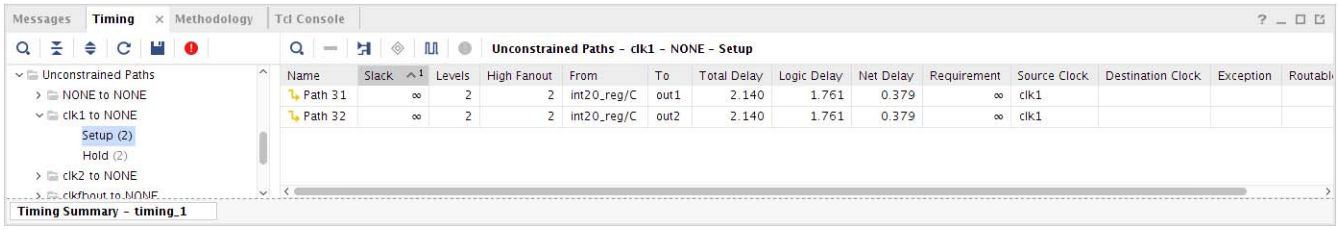


图 3-75：获得未约束的输出端口列表

在添加新约束并应用于存储器后，必须重新运行报告以确定哪些端口仍然未被约束。对于大多数设计来说，必须增加报告路径的数量，以确保所有 I/O 路径都已在报告中列出。

使用自动识别的采样时钟

无需指定关联时钟，您就能使用 set_input_delay 和 set_output_delay 约束。Vivado Design Suite 时序引擎对设计进行分析，并自动为每个端口关联所有采样时钟。然后，通过报告 I/O 路径上的时序，就可看到该工具如何约束

每个 I/O 端口。这样便于快速对设计进行约束，但如果这种通用约束过于普遍且无法准确模拟硬件的实际情况，那么这种通用约束就会引发问题。

使用主时钟

当时钟直接控制 I/O 路径时序单元，且未穿过任何时钟修改块时，应使用主时钟 （即输入单板时钟）。不能将 I/O 延迟线路视为时钟修改块，因为它们只影响时钟插入延迟，不影响波形。之前在“定义输入延迟”和“定义输出延迟”中定义的两部分中提供的两个示例已对这种情况加以阐述。大部分时间里，外部器件的接口特性也针对相同的单板时钟进行定义。

当主时钟由零保持违规 (ZHOLD) 模式下 FPGA 内的 PLL 或 MMCM 进行补偿时，I/O 路径时序单元会被连接到主时钟的一个内部副本中 （例如生成时钟）。由于两个时钟的波形相同，因此赛灵思建议将主时钟作为输入/输出延迟约束的参考时钟来使用。

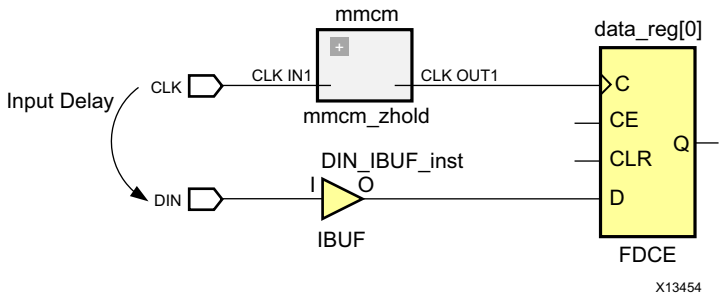


图 3-76：时钟路径中 ZHOLD MMCM 点的输入延迟

这些约束与“定义输入延迟”中提供的示例相同，因为 ZHOLD MMCM 的作用类似于时钟缓冲器，并具有一个相当于补偿量的负插入延迟。

使用虚拟时钟

若单板时钟穿过时钟修改块，而该块除了补偿整体插入延迟外还对波形进行转换，那么建议使用虚拟时钟代替单板时钟作为输入输出延迟的参考时钟。有三种情况需要使用虚拟时钟：

- 内部时钟与单板时钟具有不同周期：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，内部时钟相对于单板时钟而言拥有正向移位波形，虚拟时钟的定义类似单板时钟，用于建立的两个周期中的一条多周期路径约束定义为从虚拟时钟到内部时钟。上述约束迫使建立时序分析时要满足一个时钟周期加上相位移动量的要求。
- 对于输出路径来说，内部时钟相对于单板时钟而言有负向移位波形，虚拟时钟的定义类似板时钟，而用于建立的两个周期中的一条多周期路径限制定义为从内部时钟到虚拟时钟。上述约束迫使建立时序分析时要满足一个时钟周期加上相位移动量的要求。

综上，虚拟时钟使用要调整默认时序分析，以避免将 I/O 路径作为时钟域交叉路径并带来非常严格而又不切实际的要求。

重要提示：当相移导致时钟波形修改时，只需要使用具有相移时钟的 I/O 路径的多周期路径。当相移被添加到时钟修改块的插入延迟并且时钟波形被保留时，您不需要使用多周期路径。如需了解更多信息，请参阅 《Vivado Design Suite 用户指南：设计分析和收敛技术》(UG906) [参照 21] 中的链接。

例如，假设 sysClk 单板时钟频率为 100 MHz，与 MMCM 相乘后生成频率为 266 MHz 的 clk266。由 clk266 生成的输出应使用 clk266 作为参考时钟。如果试图将 sysClk 作为参考时钟 （针对 `set_output_delay` 规范），它将表现为异步时钟，且该路径不再作为单周期路径进行计时。

使用生成时钟

对于输出源同步接口，设计会生成一个内部时钟的副本并将其随同数据转发送给单板。无论何时试图控制和报告转发时钟与数据间的相位关系 (偏差)，该时钟都将作为输出数据延迟约束的参考时钟来使用。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 FPGA 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` 命令定义一个相对于参考时钟上升沿的延迟约束。您必须使用 `clock_fall` 选项来设定一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，且第二个约束在端口上。

大多数情况下， I/O 参考时钟沿对应于用来锁存或发送 FPGA 内 I/O 数据的时钟沿。通过分析 I/O 时序路径，您可以检查哪些时钟沿已经使用，并核实它们对应于实际的硬件行为。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作为该路径的参考时钟，那么路径要求是 ½ 周期，就会使时序收敛更加困难。

核实延迟约束

一旦输入 I/O 时序约束，请务必检查 I/O 路径上的时序是如何进行分析的，以及建立和保持检查的时序裕量违规量。使用建立和保持分析 （`delay type = min_max`）中进出所有端口的时序报告，可以核实以下内容：

- 将正确的时钟和时钟沿用作延迟约束的参考。
- 使用预期时钟来发送和捕获 FPGA 器件内部的 I/O 数据。
- 通过布局或设置适当的延迟线 tap 配置可以适当修复违规问题。如果不行，那么必须检查 I/O 约束中输入的延迟数值，并评估它们是否符合实际，以及是否需要修改设计以满足时序要求。

I/O 路径报告命令行实例

```
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
```

不正确的 I/O 延迟约束会导致时序无法收敛。实现工具均基于时序，并用于优化布局和布线以达到时序要求。如果 I/O 路径要求无法得到满足，加上设计中 I/O 路径违规问题最为严重，那么整体设计的 QoR 将会受到影响。

输入至输出馈通路径

有多种用来约束输入端口到输出端口间组合路径的等效方法。

实例 1

针对馈通路径使用周期大于或等于最大目标延迟的虚拟时钟，并按如下方法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。

实例 2

在馈通端口之间使用最小与最大延迟约束组合。实例：

```
set_max_delay -from [get_ports din] -to [get_ports dout] 10
set_min_delay -from [get_ports din] -to [get_ports dout] 2
```

该方法便于同时约束路径上的最小与最大延迟。时序分析过程中相同端口上任何现有的输入和输出延迟约束也都被使用。因此，这种方法不是很受欢迎。

最大延迟通常应针对“Slow Timing Corner”进行优化和报告，而最小延迟则在“Fast Timing Corner”中。最好对馈通路径延迟约束运行几次迭代，以验证其合理性并可满足于实现工具，尤其对于端口间距离比较远的情况更应如此。

使用 XDC 模板——源同步接口

尽管大部分用户都可以为系统同步接口正确编写时序约束，但是赛灵思仍然建议针对源同步接口使用 I/O 约束模板。源同步约束可用多种方法进行编写。Vivado Design Suite 提供的模板以默认时序分析路径要求为基础。语法更为简单，但延迟值必须进行调整以充分说明为何用捕获沿和发送沿 （1 个周期或 1/2 个周期）而不是相同沿 （0 个周期）来执行建立分析。由于时钟沿不直接对应于硬件中的活动时钟沿，因此时序报告读起来会更加困难。您可以通过 Vivado GUI 的“Tools> Language Templates > XDC > TimingConstraints > Input Delay Constraints > Source Synchronous”来找到模板。

定义时钟组和 CDC 约束

Vivado IDE 在默认情况下可为设计中所有时钟之间的路径设定时序。您可以使用以下约束修改此默认行为：

- `set_clock_groups`：关闭所识别出的时钟组之间的时序分析，而非某个相同组中时钟间的时序分析。
- `set_false_path`：仅在 `-from` 和 `-to` 选项指定的方向上禁用时钟之间的时序分析。

在某些情况下，您可能希望在跨时钟域 (CDC) 的一个或多个路径上使用以下约束来限制延迟或总线偏移：

- `set_max_delay -datapath_only`：设置异步 CDC 路径的最大延迟约束以限制延迟。

  注释：如果时钟组或伪路径约束已存在于时钟之间或相同 CDC 路径上，那么最大延迟约束将被忽略。因此，在选择 CDC 时序约束之前，为避免约束冲突，一定要彻底检查所有时钟对之间的每条路径。

建议：当 `set_max_delay -datapath_only` 约束被 `set_clock_groups` 或 `set_false_path` 约束覆盖，赛灵思还推荐运行 `report_methodology` 来进行确认。详情，请参阅第 4 章中的“运行报告方法”。

- `set_bus_skew`：通过总线偏移而不是时延来限制异步 CDC 路径之间的一组信号。

提示：您还可以从 Vivado IDE 设置总线偏移约束。在“Timing Constraints”窗口中，展开“Assertions”，然后双击“Set Bus Skew”。

检查时钟交互

若两个时钟之间有一个逻辑路径，应为两时钟设定时序。时钟关系可以是：

- “同步”
- “异步”
- “专属”

同步

当两个时钟具有固定相位关系时，时钟关系为同步。当条件符合时，两个时钟共享下列内容：

- 共用电路 （共用节点）
- 主时钟 （相同初始相位）

异步

当两个时钟不具备固定相位关系时，时钟关系为异步。当满足下列条件之一时成立：

- 它们不共享设计中的任何共用电路，也没有共用主时钟。
- 它们在 1000 个时钟周期 (unexpandable) 内没有共同周期，而且时序引擎无法正确将它们的时序安排在一起。

如果两个时钟同步，但共同周期很短，那么建立路径要求会太严格，导致难以满足时序要求。赛灵思建议您将两个时钟作为异步处理，并实现安全的异步 CDC 电路系统。

专属

当在相同时钟树上传输并到达相同时序单元时钟引脚时，时钟为专属关系，无法同时处于活动状态。

时钟对分类

可使用下列报告对时钟对进行分类：

- “时钟交互报告”
- “检查时序报告”

时钟交互报告

时钟交互报告高度总结了如何将两个时钟进行时序同步：

- 两个时钟是否有共用主时钟？当正确定义时钟时，源自设计中相同时钟源的所有时钟共享相同的主时钟。
- 两个时钟是否有共同周期？当时序引擎无法确定最悲观的建立或保持关系时，该内容显示在建立或保持路径要求列中 (“unexpandable”)。
- 两个时钟之间的路径是否部分或完全被时钟组或时序例外约束覆盖？
- 两个时钟之间的建立路径要求是不是非常严格？当两个时钟为同步，但它们的周期未被指定为精确倍数时 （例如因为四舍五入），会出现这种情况。经过多个时钟周期后，边沿会出现偏离，从而导致最差条件的时序要求非常严格。

检查时序报告

检查时序报告 (`multiple_clock`) 可确定哪些时钟引脚能连接一个以上时钟，同时， `set_clock_groups` 或 `set_false_path` 约束尚未在这些时钟之间定义。

约束专属时钟组

可以使用常规时序或时钟网络报告来检查时钟路径，并鉴别哪些情况下两个时钟在相同时钟树上传输并在起点和端点时钟引脚连接到相同时钟树的时序路径中被同时使用。这种分析相当耗时。另一种方法是您可以检查时序报告的 `multiple_clock` 部分。该部分返回一个时钟引脚及其相关时序时钟的列表。

根据时钟树拓扑结构的类型，您应使用不同约束：

- “在相同时钟源上定义的重叠时钟”
- “由时钟多路复用器驱动的重叠时钟”

在相同时钟源上定义的重叠时钟

当两个时钟通过 create_clock -add 命令定义在相同的网表对象上并代表应用的多个模式时，会出现这种情况。此时，为安全起见应在时钟之间使用一个时钟组约束。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 时钟生成其它时钟，还需要它们生成的时钟应用相同约束，可按如下方式实现：

```
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

当两个或更多时钟进入多路复用器 （或更为常见的组合单元）时，这些时钟都可传输出去，并在单元的扇出上重叠。实际上，一次只能传输一个时钟，但时序分析允许同时报告多个时序模式。

因此，您必须检查 CDC 路径并添加新的约束，以忽略一些时钟关系。正确的约束取决于设计中的时钟如何以及在哪里交互。

下图中的示例是两个进入多路复用器的时钟以及进入多路复用器前后时钟间可能的交互情况。

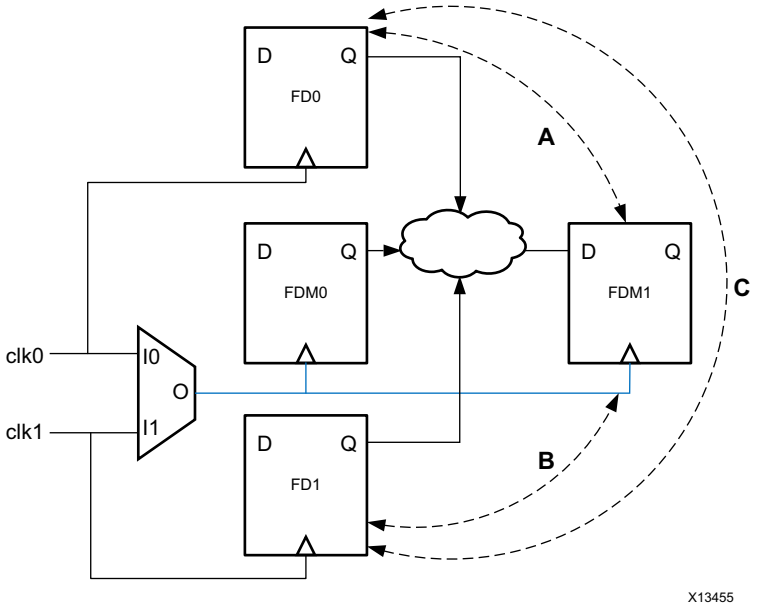


图 3-77：多路复用时钟

- **当路径 A、 B 和 C 不存在时的情况**

  clk0 和 clk1 只在多路复用器（FDM0 和 FDM1）的扇出中交互。直接向 clk0 和 clk1 应用时钟组约束是安全的。

  ```
  set_clock_groups -logically_exclusive -group clk0 -group clk1
  ```

- **当只有路径 A 或 B 或 C 存在时的情况**

  clk0 和/或 clk1 直接与多路复用时钟交互为保持路径 A、 B 和 C 的时序，约束不能直接应用于 clk0 和 clk1。而是应用于位于多路复用器扇出中的那部分时钟，这就需要附加的时钟定义。

  ```
  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
  ```

约束异步时钟组和时钟域交叉

可在时钟交互报告中快速识别异步关系：无共用主时钟或无共同周期 (unexpanded) 的时钟对。如果时钟从不同的时钟源生成，即使时钟周期相同，它们仍是异步时钟。必须仔细检查异步 CDC 路径以确保它们使用正确的同步电路，且同步电路不依赖于时序的正确性并可以最大程度降低发生亚稳态的几率。异步 CDC 路径通常具有较高的偏差和/或不实际的路径要求，因此不能用默认时序分析进行计时，因为这样无法证明它们能够在硬件中发挥功能。

Report CDC

Report CDC (`report_cdc`) 命令能够对设计中的时钟域交叉进行结构分析。您可用该信息识别潜在不安全的 CDC，这可能造成亚稳态或数据一致性问题。 Report CDC 类似于时钟交互报告 (Clock Interaction Report)，不过 Report CDC 侧重于结构及相关时序约束。 Report CDC 不提供时序信息，因为时序裕量不了解交叉异步时钟域的路径。

Report CDC 能识别下列最常见的 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

如需了解更多有关 `report_cdc` 命令的信息，请参阅 《Vivado Design Suite 用户指南：设计分析和收敛技术》 (UG906) [参照 21] 中的连接和 《Vivado Design Suite Tcl 命令参考指南》 (UG835) [参照 13] 中的 report cdc。

应采用具体约束，以避免异步 CDC 的默认时序分析：

- “双向时钟间的全局约束”
- “单独路径上的约束”

双向时钟间的全局约束

当无需限制最大延迟时，可以使用时钟组。下面是忽略 `clkA` 和 `clkB` 间路径的实例：

```
set_clock_groups -asynchronous -group clkA -group clkB
```

当两个主时钟以及它们各自的生成时钟构成两个异步域，且两域之间所有路径都被正确同步时，可立即对多个时钟同时应用时钟组约束：

```
set_clock_groups -asynchronous \
-group {clkA clkA_gen0 clkA_gen1 …} \
-group {clkB clkB_gen0 clkB_gen1 …}
```

或简单地：

```
set_clock_groups -asynchronous \
-group [get_clocks -include_generated_clock clkA] \
-group [get_clocks -include_generated_clock clkB]
```

单独路径上的约束

如果 CDC 总线使用格雷码编码 （如 FIFO）或者延迟需要限制在一个或更多信号上的两个异步时钟之间，就必须使用带有 `-datapath_only` 选项的 `set_max_delay` 约束来忽略这些路径上的时钟偏差与抖动，还要用延迟要求覆盖默认路径要求。通常源时钟周期作为最大延迟数值就足够了，只为确保任何时间 CDC 路径上都不会出现一个以上的数据。

当时钟周期间的比率较高时，选择源时钟周期与目的地时钟周期中的最小值也是减少传递延迟的好方法。干净的异步 CDC 路径在源时序单元与目的地时序单元之间不应存在任何逻辑，因此 Max Delay Datapath Only 约束对实现工具来说通常很容易满足。

一些异步 CDC 路径需要在总线的位之间的偏移控制，而不是对总线等待时间的约束。使用总线偏移约束可以防止接收时钟域在同一时钟边沿锁存总线的多个状态。您可以使用 `set_bus_skew` 命令在总线上设置总线偏移约束。例如，您可以将 `set_bus_skew` 应用于使用灰色编码的 CDC 总线，而不是使用 Max Delay Datapath Only 约束。如需了解更多信息，请参阅 《Vivado Design Suite 用户指南：如何使用约束》(UG903) [参照 18] 中的链接。

对于不需要延迟控制的路径而言，您可以定义一个点对点伪路径约束。

时钟例外优先于 set_max_delay

当编写 CDC 约束时，应确认优先级是否得到了满足。如果您在两个时钟间的至少一个路径上使用了 `set_max_delay -datapath_only`，那么就不能在相同的时钟之间使用 `set_clock_groups` 约束，而且 `set_false_path` 约束仅用于两个时钟之间的其它路径上。

如下图所示，时钟 `clk0` 的周期为 5 ns，而且与 `clk1` 异步。从 `clk0` 域到 `clk1` 域之间有两条路径。第一条路径为 1 位数据同步。第二条路径为多位格雷码编码总线传送。

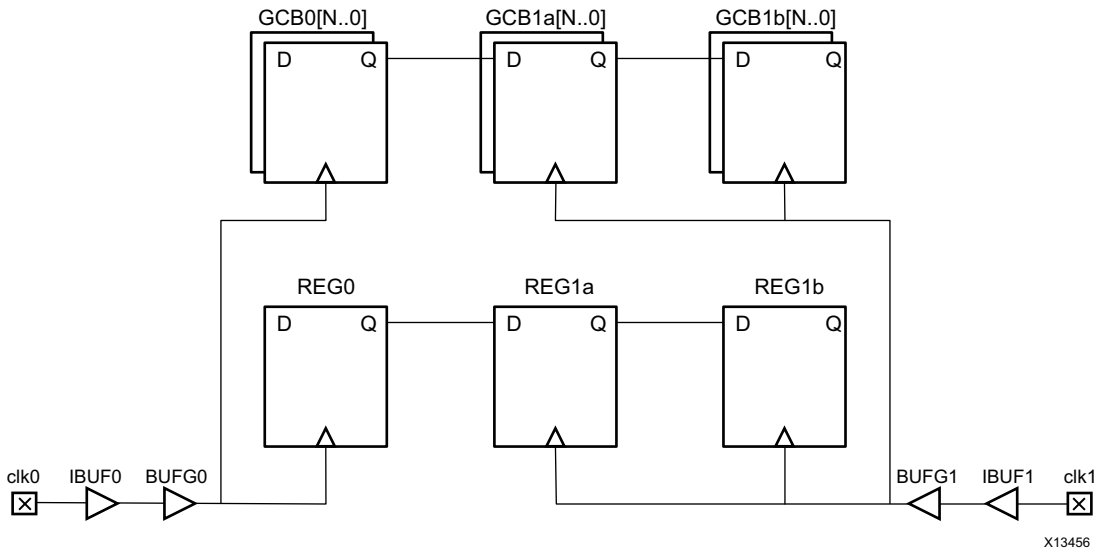


图 3-78：两个异步时钟间的多重交互

设计人员认为格雷码编码总线传送需要 Max Delay Datapath Only 来限制这些位之间的延迟变化，因此不可能在时钟之间直接使用 Clock Groups 或 False Path 约束。而是必须定义以下两个约束：

```
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]
```

没有必要设置一个从 `clk1` 到 `clk0` 的伪路径，因为本例中它们之间不存在路径。

指定时序例外

时序例外用来修改时序分析在特定路径上的执行方式。默认情况下，时序引擎假设所有路径都应依照设置分析的单周期要求进行定时，以覆盖最悲观的时钟情况。对于特定路径，并非如此。以下提供一些实例：

- 异步 CDC 路径由于缺乏时钟之间的固定相位关系而无法安全定时。它们应该被忽略 （时钟组、伪路径），或简单采用数据路径延迟约束 （仅最大延迟数据路径 Max Delay Datapath Only）。
- 时序单元发送和捕获沿并非在每个时钟周期都是活动的，因此可以相应降低路径要求 （多周期路径）
- 路径延迟要求需要增强，以增大硬件中的设计余量 （最大延迟）
- 通过组合单元的路径是静态的，而且不需要定时 （伪路径， Case 分析）
- 只能利用多路复用器驱动的一个特定时钟来执行分析 （Case 分析）。

任何情况下都必须小心使用时序例外，不能因添加时序例外而隐藏真实的时序问题。

时序例外使用指南

使用有限数量的时序例外，且尽可能简单。否则，您将面临下面两大挑战：

- 当使用多个例外时，编译流程的运行时间将显著增加，尤其是当它们附加到大量网表对象中时。
- 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对信号的优化。因此包含非必要的例外，或是在例外命令中包含非必要的点，会妨碍信号优化。

以下是可能会对运行时间产生不利影响的时序例外实例：

```
set_false_path -from [get_ports din] -to [all_registers]
```

- 如果 din 端口没有输入延迟，就不被约束。因此也不需要添加伪路径。
- 如果 din 端口只提供给时序元件，那么无需明确向时序单元设定伪路径。可更有效地编写约束：

  ```
  set_false_path -from [get_ports din]
  ```

- 如果需要伪路径，但从 din 端口到设计中的任何时序单元之间仅存在少量路径，那么约束可以更明确（all_registers 可以返回数千个单元，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

时序例外依照严格的优先级规则。最重要的规则是：

- 约束越明确，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
  ```

第一个 set_max_delay 约束具有更高的优先级，因为 -to 选项使用的引脚比时钟更明确。

- 例外优先级如下：

1. set_false_path
2. set_max_delay 或 set_min_delay
3. set_multicycle_path

尽管 set_clock_groups 命令相当于两个时钟之间的两个 set_false_path 命令，但不可视为时序例外。它的优先级高于时序例外。

set_case_analysis 和 set_disable_timing 命令禁用设计特定部分的时序分析。它们的优先级高于时序例外。

如需了解更多有关 XDC 优先级和优先级的详细信息，请参阅 《Vivado Design Suite 用户指南：如何使用约束》 (UG903) [参照 18] 中的链接。

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。即便使用仿真工具通常也很难证明路径不需要时序就能确保功能。赛灵思通常不建议使用伪路径，除非相关风险得到正确评估并且可以接受。

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，穿过两个多路复用器的路径由于选择引脚连接的原因绝不会让数据在同一时钟周期内传输。

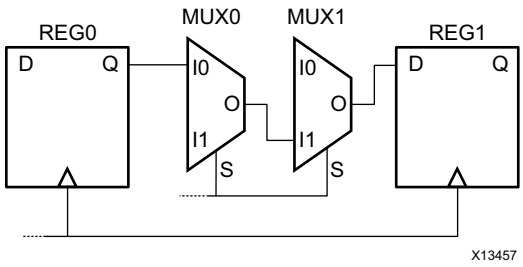


图 3-79：不能激活的路径

```
set_false_path -through [get_pins MUX0/I0] -through [get_pins MUX1/I1]
```

- 忽略异步 CDC 路径上的时序。这种情况已经在“定义时钟组和 CDC 约束”。
- 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一个值后，就不再切换。当这些寄存器似乎在设计的关键路径上时，可以忽略其时序，以缓解对实现工具的约束并有助于时序收敛。仅从静态寄存器定义伪路径约束就足够了，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[*]]
```

对综合的影响

伪路径约束由综合支持，而且只影响最大延迟 （建立/恢复）路径优化。除非忽略 CDC 路径，否则在综合过程中通常不需要使用伪路径例外。

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来覆盖对保持/移除和建立/恢复检查的默认路径要求，方法是用约束中的延迟数值替代发射和捕捉沿时间。

用例

使用最小或最大延迟约束的常见原因是：

- 通过收紧建立/恢复路径要求对设计中的一些路径进行过度约束。

  这有助于迫使逻辑优化或布局工具更有效地作用于一些关键路径单元，这样可以为布线器提供更大灵活性，以满足后续的时序要求 （在移除最大延迟约束后）。

- 替代多周期约束。

  对于每 N 个时钟周期即获得活动的发送和捕获沿的路径而言，可以采用这种方法降低该路径的建立要求，但不推荐使用这种方法。但是，这种方法是唯一能够以单个时钟周期几分之一的幅度对多周期路径进行过度约束以促进布线阶段时序收敛的方法。例如，多周期约束为 3 的路径似乎是布线后的最差违规路径，并且时序误差为几百皮秒。

  在优化和布局阶段，可使用以下约束替代最初的多周期路径约束：

  ```
  set_max_delay -from [get_pins <startpointCell>/C] \
  -to [get_pins <endpointCell>/D] 14.5
  ```

  其中

  14.5 等于 3 个时钟周期 （每个为 5 ns）减去 500 ps，也就是所需的额外余量。

- 约束异步 CDC 路径上的最大数据路径延迟。

  该方法已在“定义时钟组和 CDC 约束”部分介绍过。

不建议使用 set_min_delay 约束在路径上强制插入额外延迟。保持或移除的默认最小延迟要求通常足以在正裕量时确保正确的硬件功能。

对综合的影响

综合步骤支持 set_max_delay 约束，包括 -datapath_only 选项。可忽略 set_min_delay 约束。

对实现的影响

set_max_delay 约束替代建立路径要求，并影响整个实现流程。 set_min_delay 约束替代保持路径要求，在需要修复保持时仅影响布线器行为。

避免路径分段

仅当为 set_max_delay 和 set_min_delay 命令的 -from 或 -to 选项指定无效的起点或端点时才引入路径分段。当 set_max_delay 为一条路径引入路径分段时，默认的保持分析才不会发生。如果希望对保持分析进行约束，那么必须使用 set_min_delay 约束该路径。该原则适用于 set_min_delay 命令以及建立分析。

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：

- 路径分段打破分段路径上的时钟偏差计算。
- 路径分段可以打破 set_max_delay 或 set_min_delay 分段命令约束的路径以外的更多路径。

当使用约束时，工具会在日志文件中报告路径分段。必须使用有效的起点和端点加以避免：

- **起点**

  时钟、时钟引脚、时序单元 （使用单元的有效起点引脚）、输入或双向端口

- **端点**

  时钟、时序单元的输入数据引脚、时序单元 （使用单元的有效端点引脚）、输出或双向端口

  有关路径分割的详细信息，请参阅《Vivado Design Suite 用户指南：如何使用约束》(UG903) [参照 18] 中的链接。

添加多周期路径约束

多周期路径例外处理必须反应设计的功能性，而且使用该例外处理的路径在源时钟和 （或）目的地时钟的每个周期中不能有活动的时钟沿。路径乘法器以时钟周期为表达形式；当使用 -start 选项时基于源时钟，当使用 -end 选项时则基于目的地时钟。这样尤其便于独立于时钟周期值以外修改起点和端点之间的建立和保持关系。

保持关系始终与建立关系相关联。因此大多数情况下，保持关系需要在建立关系修改后进行单独调整。这就是为什么需要一个带 -hold 选项的第二个约束。该原则的例外情况是相移时钟之间的同步 CDC 路径：只有建立需要修改。

降低“建立”要求，同时让“保持”不变

当源时序单元和目标时序单元被时钟使能信号 （每 N 个周期激活时钟）控制时会出现这种情况。以下示例，在每 3 个周期具有有效时钟使能，在起始点和端点具有相同的时钟：

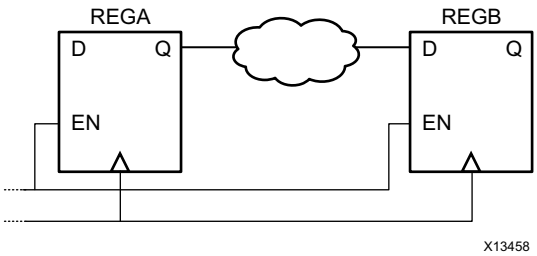


图 3-80：激活了具有相同时钟信号的寄存器

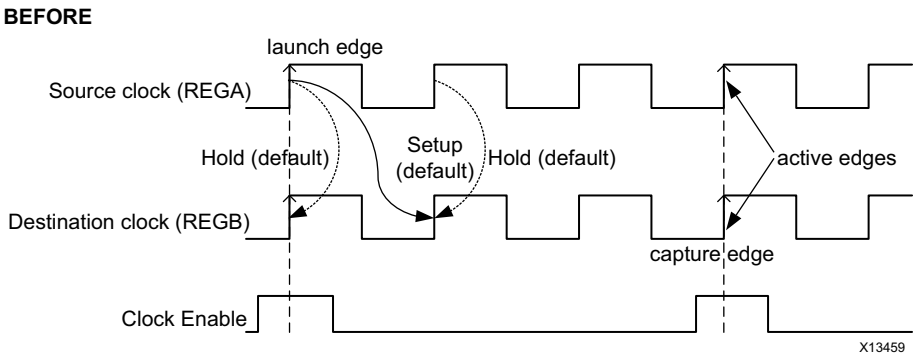


图 3-81：建立/保持检查的时序图

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
```

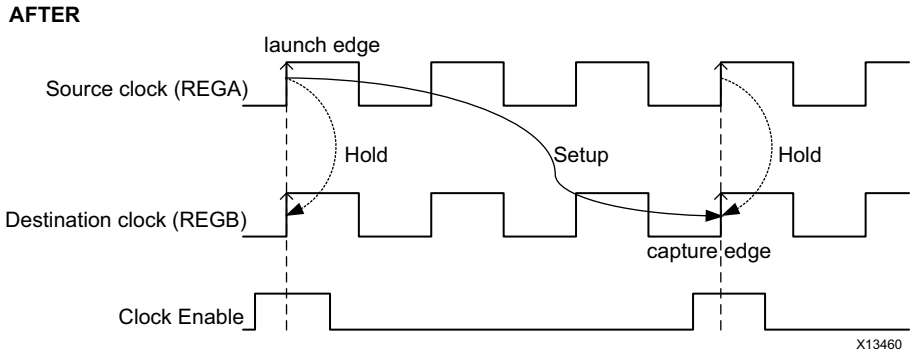


图 3-82：多周期规格描述之后修改建立/保持检查

注释：使用第一个命令后，建立捕获沿移到第三个时钟沿 （即从默认位置移动 2 个周期），保持沿也移动两个 2 周期。第二个命令用于将保持沿移回到初始位置，即反方向在移动 2 个周期。

如需了解更多有关其他常见多周期路径情况的信息，如同步时钟之间的相移和多周期路径，请参阅 《Vivado Design Suite 用户指南：如何使用约束》(UG903) [参照 18] 中的链接。

重要提示：当时钟相移不修改时钟波形，而是包括在时钟修改块的插入延迟中时，您不需要添加一个仅设置多周期路径来正确计时从时钟到时钟的路径。如需了解更多信息，请参阅 《Vivado Design Suite 用户指南：设计分析和收敛技术》(UG906) [参照 21] 中的链接。

对综合与设计实现的影响

综合步骤支持 `set_multicycle_path` 约束。该约束可缓解每个时钟周期内功能上不处于活动状态的长路径，从而大幅提升时序 QoR （仅针对建立）。

就综合而言，多周期路径异常处理有助于设计实现时序驱动的算法集中于真正的关键路径。只有在布线过程中保持要求才非常重要。如果建立关系通过 `set_multicycle_path` 约束 （而非相应的保持关系）进行调整，那么最差保持要求如果超过 2 ns 或 3 ns 就变得难以满足。这种情况会对建立裕量产生不利影响，因为在修复保持违规时布线器插入了附加延迟。

常见错误

以下是两个绝对需要避免的典型错误：

- 在多周期路径无法在每个时钟周期实现功能激活的情况下放松“建立”但未将“保持”调整到之前的发送和捕获沿。保持要求会变得非常高 （至少 1 个时钟周期），且无法满足。
- 在设计中错误点之间建立多周期路径异常处理。

  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。端点单元可以有多个数据输入引脚，包括时钟使能和复位引脚，它们至少在两个连续时钟沿处于活动状态。

  因此，赛灵思建议指定端点引脚而非单元 （或时钟）。例如，端点单元 REGB 有三个输入引脚：C、 EN 和 D。只有 REGB/D 引脚需要由多周期路径例外处理进行约束（EN 引脚不用），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引脚，那么所有有效的端点引脚，包括 EN （时钟使能）引脚，都在约束的考虑范围内。

  为安全起见，赛灵思建议您始终使用如下语句：

```
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
```

其它高级时序约束

可设置其它一些时序约束以忽略和修改默认时序分析：

- “Case 分析”
- “不分析时序”
- “数据检查”
- “最长时间借用”

Case 分析

Case 分析命令可像配置寄存器那样在逻辑中设定常数，用以描述设计中的功能模式。它可被应用于输入端口、网络、层级引脚或叶节点单元输入引脚。该常数值在逻辑中传输，并关闭永不活跃路径上的分析。其效果类似于伪路径异常处理的工作方式。

最常见的范例是将多路复用器选择引脚设定为 0 或 1，以便只让两个多路复用器输入中的一个传输通过。下面的实例可关闭通过 `mux/S` 和 `mux/I1` 引脚的路径上的分析：

```
set_case_analysis 0 [get_pins mux/S]
```

不分析时序

不分析时序命令可关闭时序数据库中的时序 arc，能完全阻止任何通过该 arc 的分析。可用 `report_disable_timing` 命令报告关闭的时序 arc。

注意！使用不分析时序命令时要小心。它会打断比预想中更多的路径！

数据检查

`set_data_check` 命令可设置设计中两个引脚之间的建立或保持时序等效检查，常用来约束和报告异步接口。该命令应由专家级用户使用。

最长时间借用

`set_max_time_borrow` 命令能设置锁存器可从下一级 （锁存器后面的逻辑）借取的最长时间，并将时间送给前级（锁存器前面的逻辑）。通常不建议使用锁存器，因为它们很难在硬件中测试和验证。该命令应由专家级用户使用。

定义物理约束

物理约束可用于控制布局规划、特定布局、 I/O 分配、布线器以及类似功能。应确保每个引脚都具有特定 I/O 接口和标准。以下用户指南包括物理约束：

- 《Vivado Design Suite 用户指南：如何使用约束》(UG903) [参照 18]：锁定布局与布线，包括与宏命令相关的布局
- 《Vivado Design Suite 用户指南：设计分析和收敛技术》(UG906) [参照 21]：关于了解布局规划，
- 《Vivado Design Suite 用户指南：编程和调试》(UG908) [参照 24]：关于配置，

实现

综合和设计实现简介

选定器件之后，选择和配置了 IP，且编写了 RTL 和约束条件，那么下一步进入实现阶段。设计实现通过综合和布局布线来编译设计，然后生成用于配置器件的比特流。如第 1 章"引言"所述，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特定瓶颈的要诀和技巧。

运行综合

综合步骤将采用 RTL 和时序约束，并生成在功能上等同于 RTL 的优化网表。通常情况下，综合工具能接受任何合法的 RTL 并为其生成逻辑。如第 3 章中的"利用约束"和第 5 章中的"设计基准 (baseline)"所描诉的一样，综合需要现实的时序约束。

如需了解更多有关综合的信息，请参阅下列资源：

- 《Vivado Design Suite 用户指南：综合》(UG901) [参照 16]
- Vivado Design Suite QuickTake 视频教程：设计流程

综合属性

综合属性允许您以特定方式控制逻辑推理。尽管综合算法在设定上是为最大数量的设计提供最佳结果，但经常会存在一些具有不同要求的设计。这种情况下，您可以利用属性更改设计以提高 QoR。如需了解综合所支持的属性信息，请参阅 《Vivado Design Suite 用户指南：综合》(UG901) [参照 16]。

注释：在新器件运行该设计之前，赛灵思建议先查看之前运行该设计的旧器件的综合的所有属性。

当使用 KEEP、 DONT_TOUCH 和 MAX_FANOUT 等属性时，请注意以下的各节中所描述的特殊考虑因素。

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，工具就不会优化该对象。

- KEEP 由综合工具使用，并且不作为网表中传递的属性。 KEEP 可用于保留特定信号，例如，关闭综合期间信号的特定优化。
- DONT_TOUCH 由综合工具使用，然后再传递给布局布线工具，使对象永远不会被优化。

使用这些属性时需要倍加小心：

- 接收 RAM 输出的寄存器上的 KEEP 属性，能阻止该寄存器并入 RAM，从而阻止块 RAM 的调用。
- 不要在以上水平中的驱动三态输出或双向信号的层级上使用这些属性。如果驱动信号和三态条件处于这个层级中，那么 IOBUF 将无法被调用，工具为了创建 IOBUF 必须改变层级。

此外，请注意，在信号上和在层次结构级别上布局 DONT_TOUCH 之间是存在差异的：

- 如果将该属性放置在信号上，则保持该信号；
- 如果将该属性布局在某个层级上，那么工具不会接触该层级的边界，而且层级中不会发生常数传输。但是，该层次结构级别内的优化被保留下来。

MAX_FANOUT

MAX_FANOUT 强制综合复制逻辑，以满足扇出限值。该工具能复制逻辑，但不能复制输入或黑盒。因此，如果将 MAX_FANOUT 属性放置在由设计方案的直接输入进行驱动的信号，那么工具将无法处理该约束。

要注意分析放置了 MAX_FANOUT 的信号。如果 MAX_FANOUT 所在信号是由具有 DONT_TOUCH 的寄存器驱动，或者当该层级上含有 DONT_TOUCH 属性时，MAX_FANOUT 驱动位于不同层级的信号，那么 MAX_FANOUT 属性将无法执行。

综合利用 _rep 添加复制的单元，第一次复制，随后的复制分别为 _rep__0、 _rep__1 等。通过选择“Edit > Find”在单元上查找，可以在后综合网表中看到这些单元。

重要提示：在综合时有所保留地使用 MAX_FANOUT。 Vivado® 工具中的 phys_opt_design 命令对设计的布局有更好的理解，且复制工作也比综合更加优秀。如果需要特定的扇出，那么花费时间和精力对额外寄存器进行手动编码往往都是值得的。

块级综合策略

借助 Vivado 综合，您可以使用各种策略和全局设置来定制设计综合的过程。在大多数情况下，这些选项都是全局性的，会影响整个设计。在一个自上而下的流程中，您可以使用块级综合策略来综合拥有不同全局选项的不同层级。与自下而上的编译相比，该流程能更快速、更方便地执行。您可以为整个设计设置约束，而不是为较低的级别设置约束，然后重新设置最高级别。

使用块级综合策略

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

- 设置 option_name 选项。
- value 是分配给该选项的值。
- instance_name 是设置该选项的分级实例名。

注释：这些属性通常在分级实例上设置。这能通过不同的选项对实例化的模块或实体进行多次综合。

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 的综合表现如下图所示。

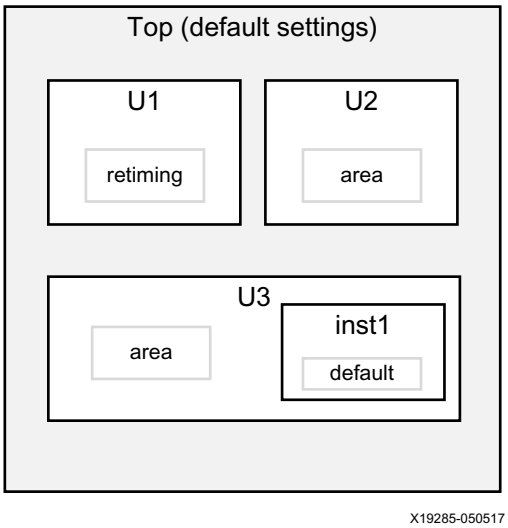


图 4-1：块级综合策略示例

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 是全局编译的，那么您可在 IP 顶层使用该策略。
- 如果 IP 脱离了上下文，则不能使用该策略，因为 IP 是一个黑盒。相反，在编译 IP 时应使用全局设置。

注释：如需了解更多有关此功能以及所支持的策略和选项的信息，请参阅 《Vivado Design Suite 用户指南：综合》 (UG901) [参照 16]。

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：

- 后综合网表
- 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：设计过程中应尽早检查和纠正 DRC 违规，以避免在实现流程的后期出现时序或与逻辑相关的问题。

运行报告方法

鉴于 UltraFast 设计方法的重要性， Vivado 工具提供了一个专门检查是否符合方法论的方法报告。工具根据所处的设计流程阶段运行不同的检查。

- RTL 设计：RTL lint 风格检查
- 综合和实现设计：网表，约束和时序检查。

如果用 Tcl 提示符运行上述检查，打开待验证设计，然后输入下列 Tcl 命令：

```
report_methodology
```

如需在 Vivado IDE 中运行上述检查，打开待验证设计，并选择“Tools > Report > Report Methodology”。如下图所示，任何违例情况都列在“Methodology”窗口中。在权衡是否不必清除设计中某项特定方法违规时，请确切了解该项违例的内容及其含义，并弄清该项违例不会对设计造成消极影响的原因。

注释：对于赛灵思提供的 IP 核，已经对违例进行过评估和核查。

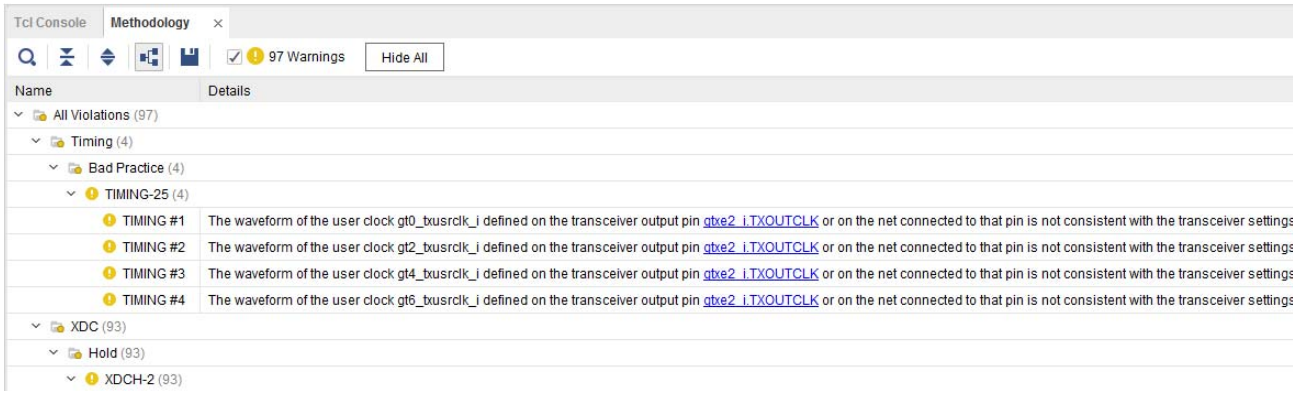


图 4-2：“Methodology”窗口

建议：要确定常见的设计问题，请在第一次综合设计时运行“Report Methodology”。当约束，时钟拓扑结构或大型逻辑变更发生变化时，再次运行此报告。

提示：在布线之后，您也可以将这些检查设置为默认运行。在“Implementation Run Properties”窗口的“Properties”视图中，选择“Run report ultrafast methodology after routing”。

如需了解更多运行“Report Methodology”的信息，请参阅《Vivado Design Suite 用户指南：系统级设计输入》(UG895) [参照 8] 和 《Vivado Design Suite 用户指南：设计分析和收敛技术》(UG906) [参照 21]。

检查综合日志

必须检查综合日志文件，并确认工具给出的所有信息在设计内容上与预期一致。应特别注意严重警告 (Critical Warning) 和警告 (Warning)。在大多数情况下，需要清除 Critical Warning 以获得可靠的综合结果。

注意！如果一条消息的显示次数超过 100 次，该工具只会将显示的前 100 次写入综合日志文件。可以使用 Tcl 命令 `set_param messaging.defaultLimit` 来更改数值为 100 的限值。

检查时序约束

必须提供干净的时序约束，并在适当情况下提供时序例外处理。不好的约束会导致较长的运行时间、性能问题和硬件故障。

建议：检查所有与时序约束有关、告知时序未被加载或正确应用的 Critical Warning 和 Warning。

如需了解更多信息，请参阅第 3 章中的“编制设计约束”。

满足综合后的时序要求

下面几节将介绍如何满足综合后的时序要求。

遵循处理剩余违规行为的指南

重要提示：在继续流程之前分析综合后的时序，以识别必须解决的主要设计问题。

HDL 变化对 QoR 的影响最大。因此，最好在实现之前解决这些问题，以获得更快的时序收敛。在分析时序路径时应特别注意以下几点：

- 最频繁的问题点，即在最差故障时序路径中出现次数最多的单元或网络
- 由未寄存的块 RAM 提供的路径
- 由 SRL 提供的路径
- 包含未寄存的级联 DSP 块的路径
- 具有逻辑级大数路径
- 大扇出路径

如需了解更多信息，请参阅第 5 章中的“时序收敛”。

处理高逻辑级数

识别长逻辑路径有助于诊断较难解决的 QOR 挑战。综合后预计的网络延迟接近于最佳布局。要想评估具有高逻辑级数延迟的路径是否满足时序要求，您可以生成无网络延迟的时序报告。如果路径仍违反无网络延迟的时序，那么在这些路径上就无法实现时序收敛。如需了解更多信息，请参阅第 5 章中的“时序收敛”。

检查利用率

分别检查 LUT、 FF、 RAMB 和 DSP 组件的利用率十分重要。如果块 RAM 利用率比较高，那么 LUT/FF 利用率较低的设计仍可能出现布局难题。 `report_utilization` 命令针对所有设计对象生成分章节的综合利用率报告。

注释：在综合之后，由于之后在设计流程中的优化，利用率可能发生改变。

检查时钟树

本节讨论了查看时钟树，包括时钟缓冲区利用率和时钟树拓扑结构。

时钟缓存的使用

`report_clock_utilization` 命令提供关于时钟原语使用的详细信息。应观察架构时钟规则以避免下游出现布局问题。例如，在 7 系列器件中， BUFH 只能扇出到其时钟域内的负载中。区域时钟缓存的无效布局约束或非常高的扇出会

导致布局器出现问题。对于时钟缓存利用率很高的设计，有必要锁定时钟生成器以及一些区域时钟缓存，以辅助完成布局任务。

有些接口需要非常严格的时序关系，因此有时可能要为需要严格时序关系的信号锁定特定资源，例如源同步接口。通常，作为设计的出发点，只需锁定 I/O 即可，除非存在以上引用的特殊原因。

如需了解更多有关建议布局增益的信息，请参阅第 5 章中的“时序收敛”。

时钟树拓扑结构

使用时钟树时，请遵循以下建议：

- 运行 `report_clock_networks` 命令，以便在详细树形视图中显示时钟网络。
- 通过某种方式使用时钟树以最大限度地降低偏差。
- 对于 PLL 和 MMCM 的输出，使用相同的时钟缓存类型以最大限度降低偏差。
- 寻找可引起额外延迟或偏差 （或二者皆有）的非预定的级联 BUFG 元素。

实现设计

Vivado Design Suite 实现过程包括将网表布局布线到 FPGA 器件资源所需的全部步骤，同时满足设计的逻辑、物理和时序约束。如需了解更多有关实现的信息，请参阅下列资源：

- 《Vivado Design Suite 用户指南：实现》(UG904) [参照 19]
- Vivado Design Suite QuickTake 视频教程：设计流程

使用工程模式与非工程模式选项

您可在工程模式或非工程模式中运行设计实现。工程模式可提供运行管理、文件集管理、报告生成和交叉探测等工程基础架构。非工程模式则提供简便的集成，而且由 Tcl 脚本驱动；Tcl 脚本必须在整个流程中显式调用所需的全部报告。如需了解更多有关此些模式的信息，请参阅 《Vivado Design Suite 用户指南：设计流程简介》(UG892) [参照 5] 中的链路。

策略

策略是一组自定义的用于控制工程模式中运行行为的 Vivado Design Suite 实现选项。策略行为则由施加于单个实现命令上的指令来控制。

建议：首先尝试使用 Vivado Design Suite 实现的默认策略。这样可在运行时间和设计性能之间实现很好的折中。

注释：策略针对特定的工具和版本。

指令

指令为下列的实现命令提供不同行为模式：

- `opt_design`
- `place_design`
- `phys_opt_design`
- `route_design`

首先使用默认指令。在设计快完成时再使用其他指令，以探索设计的解决方案空间。一次只能执行一个指令。

如需了解更多有关策略和指令的信息，请参阅 《Vivado Design Suite 用户指南：实现》(UG904) [参照 19]。

反复循环的流程

在非工程模式下，您可以用不同选项在各种优化命令之间设计反复。例如，您可以运行 `phys_opt_design -directive AggressiveFanoutOpt`，然后运行 `phys_opt_design -directive AlternateFlowWithRetiming`，以便在不符合时序的已布局设计上运行不同的物理综合优化。

设计反复运行 `phys_opt_design` 可改进时序。`Phys_opt_design` 命令能够优化顶层时序问题路径。通过反复运行 `phys_opt_design`，较低层的时序问题也会因优化而有所改善。在布局后期运行 `phys_opt_design` 将会对任何可能未布线的网络重新布线。因此，在布局后期运行 `phys_opt_design` 之后，您不需要运行 `route_design`。

使用检查点分析不同阶段的设计

Vivado Design Suite 采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) 可用来存储 （`write_checkpoint` 命令）和回读 （`read_checkpoint` 命令）设计流程关键点的物理数据库。检查点是流程中特定点的设计快照。在工程模式下， Vivado 工具自动生成设计检查点文件，并将它们存储在设计实现运行目录中。这些可以在单独的 Vivado 实例中打开。

设计检查点文件包含以下内容：

- 当前的网表，包括实现过程中所做的任何优化
- 设计约束
- 实现结果

可利用 Tcl 命令在设计流程剩余部分运行检查点设计。但无法通过新设计源对其进行修改。

检查点使用方面的一些常见实例如下：

- 保存结果，以便返回上一级并在该流程部分执行进一步分析。
- 使用多个指令尝试运行 `place_design`，并为每个指令保存检查点。这样可以选出具有最佳时序结果的 place_design 检查点，以便用于后续的实现步骤。

如需了解更多有关检查点的信息，请参阅 《Vivado Design Suite 用户指南：实现》(UG904) [参照 19]。

使用增量编译流程

在 Vivado Design Suite 中，您可以使用增量编译来重用现有的布局和布线数据，减少实现的运行时间，并产生更多可预测的结果。当使用具有 95% 或更高重用性的设计时，增量的布局和布线时间通常至少比正常布局和布线的运行时间缩短两倍，同时可维护参考运行的 WNS。如需了解更多信息，请参阅 《Vivado Design Suite 用户指南：实现》(UG904) [参照 19] 中的链接。

建议：增量编译在设计周期的关键阶段最有用，因为在那个阶段很难对流程的脚本进行更改。确保您的流程脚本在设计周期的早期阶段包括增量编译，这样您就可以在关键时期启用增量编译。

增量编译流程模式

增量编译支持高重用模式和低重用模式，这些模式支持不同的指令并可改变流程的行为。如需了解更多关于包括支持指令和目标 WNS 在内的重用模式信息，请参阅《Vivado Design Suite 用户指南：实现》(UG904) [参照 19] 中的链接。

运行时间考虑因素

在高重用模式中， 95% 以上的设计将被重用，运行时间可以减少一半。随着重用的减少，收益也会下降。

在低重用模式下，运行时间是不可预测的。当布局和布线运行得更接近于两者汇合时间时， Vivado 工具可能会增加运行时间以满足时间安排。在其他情况下，如果现有的布局和布线数据被有效重用，那么 Vivado 工具可能会降低运行时间。

以下因素也会影响运行时间:

- 如果不能重用关键路径的布局和布线，我们则需要做更多的工作来保存时间。
- 小的设计更改可能会引入参考设计中不存在的新时序问题。

并行流程

将标准流程运行与增量编译运行并行运行，以获得最大的灵活性。这允许您利用增量编译的好处，同时又可保留时序收敛。此外，它还允许您使用增量编译算法，该算法与标准流程不同，可以产生不同的结果。

提示：您可以定期更新参考检查点，以帮助维护流程的有效性。在资源有限的情况下，在启动标准流程运行之前，应先启动增量编译运行，以更好地利用运行时间。

打开综合设计。

综合后的第一步是将网表从综合设计中读入内存并应用设计约束。您可以根据使用的流程通过各种方式打开综合设计。如需了解更多信息，请参阅 《Vivado Design Suite 用户指南：实现》(UG904) [参照 19] 中的链接。

逻辑优化 (opt_design)

Vivado Design Suite 逻辑优化功能可优化当前存储器内的网表。由于这是组合设计的第一个视图 （RTL 和 IP 块），因此通常可进一步优化。默认情况下， `opt_design` 命令执行逻辑整理，移除无负载的单元，传输常数输入，以及进行块 RAM 功耗优化。此外，它还可执行其他优化内容，例如重新映射，即将 LUT 串联组合为更少的 LUT 以缩短路径深度。

优化分析

`opt_design` 命令生成的消息可详细介绍每个优化阶段的结果。优化完成后您可运行 `report_utilization` 来分析利用率的提升情况。为了更好地分析优化结果，应通过 `-verbose` 选项来查看受到 `opt_design` 优化影响的逻辑的更多信息。

如需了解更多有关逻辑优化的信息，请参阅 《Vivado Design Suite 用户指南：实现》(UG904) [参照 19] 中的链接。

实现过程中的功耗注意事项

如需了解相关设计的功耗优化，请参阅第 5 章中的“功耗优化”。

布局 (place_design)

Vivado Design Suite 布局器引擎可将来自网表的单元放到目标赛灵思器件的特定位置。

布局分析

布局后使用时序总结报告检查关键路径。

- 具有超大型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。

- 具有超大型保持时间时序负裕量的路径最有可能因为不正确的约束或不好的拓扑结构造成，因此需在进入布线设计之前对其进行修复。
- 具有较小保持时间时序负裕量的路径有可能被布线器修复。您也可以在 `place_design` 之后运行 `report_clock_utilization`，以便查看按照时钟域详细分解时钟资源和负载数量的报告。

如需了解更多信息，请参阅第 5 章中的“时序收敛”。如需了解更多有关布局的信息，请参阅 《Vivado Design Suite 用户指南：实现》(UG904) [参照 19] 中的链接。

物理优化 (phys_opt_design)

物理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优化。优化内容涉及复制、重定时、修复保持时间以及布局提升。由于物理优化自动执行所有必要的网表和布局变更，因此在 `phys_opt_design` 之后不需要进行 `place_design`。

物理综合需求

为了确定设计是否能受益于物理综合，需要在完成布局后对时序进行评估。应针对扇出特点分析故障路径。高扇出关键路径可受益于扇出优化。此外，如果大型 RAM 块涉及到多个在 `route_design` 后产生时序失败的块 RAM，那么其高扇出数据、地址和控制网络可受益于强制网络复制 （Forced Net Replication）。如需了解更多有关物理综合的信息，请参阅 《Vivado Design Suite 用户指南：实现》(UG904) [参照 19] 中的链接。

布线 (route_design)

Vivado Design Suite 布线器对已布局的设计进行布线，并优化已布线的设计，以解决保持时间违规问题。该功能默认情况下采用时序驱动，但也可关闭。

布线分析

网络没有达到最佳布线通常是由不正确的时序约束造成。在试验布线器的设置之前，一定要验证布线器所见的约束和时序图。在布线之前检查已布局设计的时序报告，以验证时序和约束。

时序约束不良的常用范例包括跨时钟路径与错误的多循环路径会引起由于保持时间固定而造成路由插入延迟。拥塞区域可以通过 RTL 综合中的目标扇出优化或通过物理优化来解决。您可以保留所有或部分的设计层次结构，以防止跨边界优化并且能够减少网表密度。还可以使用布局规划约束来缓解拥塞。

如需了解更多信息，请参阅第 5 章中的“时序收敛”。如需了解更多布线的信息，请参阅 《Vivado Design Suite 用户指南：实现》(UG904) [参照 19] 中的链接。

XILINX
ALL PROGRAMMABLE™

第 5 章

设计收敛

设计收敛简介

设计收敛包括同时满足时序和功耗要求，并成功编写可验证硬件功能性的配置比特流。设计收敛通常需要在结果分析、设计修改和约束修改之间运行几次迭代。

时序收敛

时序收敛包括可满足所有时序要求的设计。如果我们针对综合具有正确的 HDL 和约束条件，那么实现时序收敛会更容易。此外，如下图所示，一定要通过改进的 HDL、约束和综合选项在综合阶段运行迭代。

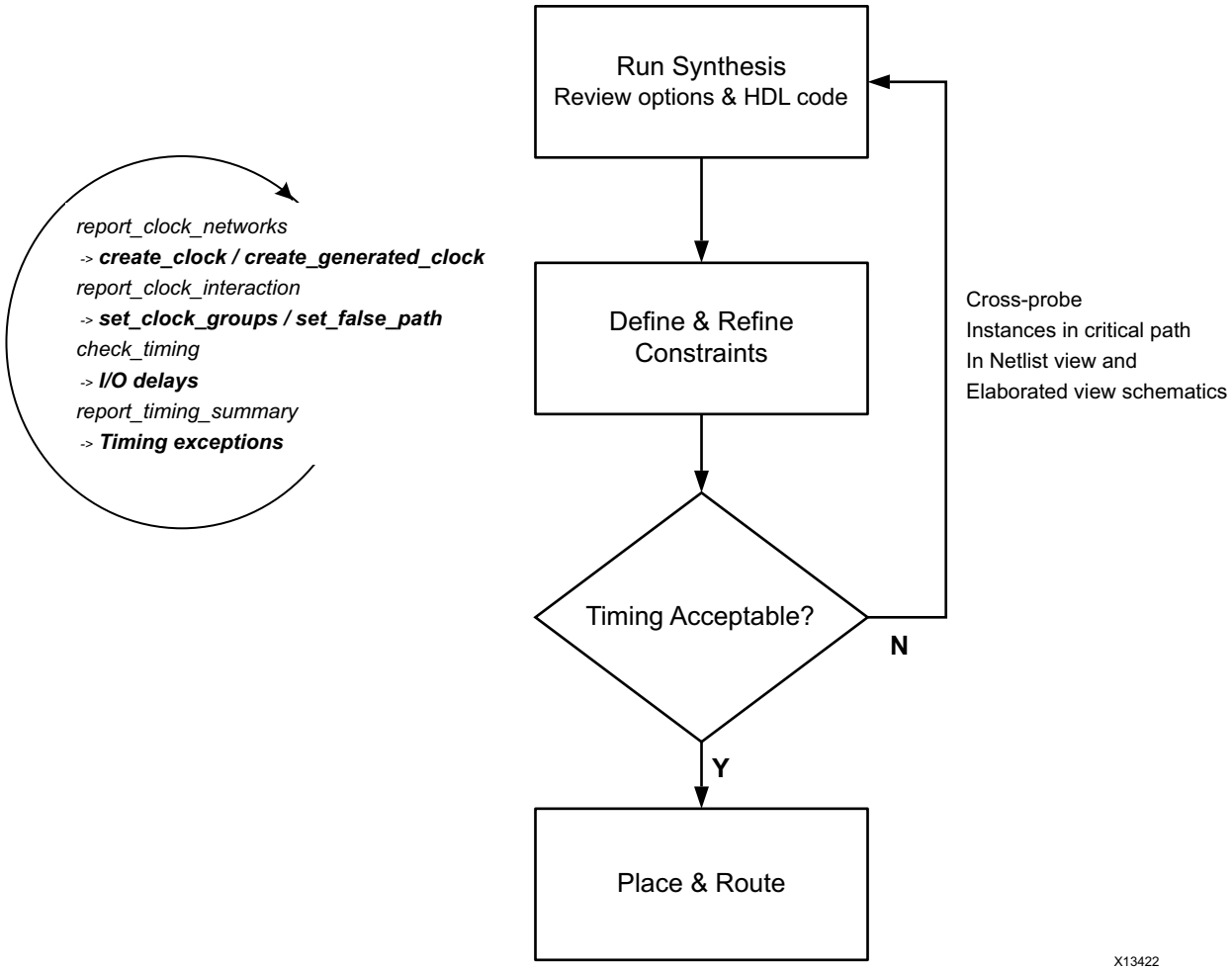


图 5-1：实现快速收敛的设计方法

要成功收敛时序，需要遵循下列一般性准则：

- 最初不能满足时序要求时，则在整个流程中评估时序。
- 关注每个时钟的 WNS，以此作为提升 TNS 的主要方法。
- 检查严重 WHS 违规 (< -1 ns)，以便确定遗漏的或者不恰当的约束。
- 重新利弊取舍设计选择、约束和目标架构之间的权衡。
- 知道如何使用工具选项和 XDC。
- 要知道一旦时序得到满足工具就不会再进一步提升时序 （更多裕量）。

以下部分提供的建议可用于通过 DRC 方法和基线来查看时序约束的完整性和正确性，找出时序违规的根本原因以及使用常用技术解决违规等。

理解时序收敛指标

实现时序收敛首先要编写用来反映设计如何在硬件中运行的有效约束。按照下文各章节所述，检查 Timing Summary 报告。

检查有效约束

查看 Timing Summary 报告的 Check Timing 部分，以便快速评估时序约束覆盖范围，其中包括以下内容：

- 时钟定义能抵达所有活动时钟引脚。
- 所有活动路径端点具有与所定义时钟有关的要求 （建立/保持/恢复/移除）。
- 所有活动输入端口具有一个输入延迟约束。
- 所有活动输出端口具有一个输出延迟约束。
- 正确规定时序例外。

注意 ！ 在约束中过度使用通配符可能导致实际约束与预想的不一致。使用 `report_exceptions` 命令可确定时序例外冲突，并查看每个异常所覆盖的网表对象、时序时钟以及时序路径等。

除 `check_timing` 以外， Methodology 报告 （TIMING 与 XDC 检查）还会标记时序约束，这可能会导致不准确的时序分析与潜在的您必须仔细查看并解决所有报告的问题。

注释： 为设计配置基线标准时，您必须使用所有的赛灵思 IP 约束。不要指定用户 I/O 约束，并且因用户 I/O 约束缺失而忽略由 `check_timing` 和 `report_methodology` 生成的违规。

检查正时序裕量

以下是指示存在时序违规的时序指标。为了满足时序要求，数字必须为正。如需了解更多信息，请参阅"了解时序报告"。

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

了解时序报告

与约束相比，时序摘要报告可提供设计时序特性的高层次信息。在结束时检查时序总结数字：

- TNS （总体时序负裕量）：整个设计或针对特定时钟域中每个端点建立/恢复违规的总和。最差建立/恢复时序裕量为 WNS （最差负时序裕量）。
- THS （总体保持时序裕量）：在整个设计或针对特定的时钟域中，每个端点的保持/移除违规的总和。最差保持/移除时序裕量为 WHS （最差保持时序裕量）。
- THS （总体保持时序裕量）：在整个设计或针对特定的时钟域中，每个时钟引脚都进行以下检查时的违规总和：
  - 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 （相同单元的两个时钟引脚之间）
- WPWS （最差脉冲宽度时序裕量）：适用于所有脉冲宽度、周期，或用于对任何给定时钟引脚进行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规情况。当所有的时序检查均符合要求时，总时序裕量为零。

此外，时序路径报告还提供在任意逻辑路径上针对任何时序检查其时序裕量的详细计算方法。在被完全约束的设计中，每条路径都有一个或几个必须满足的需求，以确保相关逻辑能够可靠地运转。

涵盖 WNS、 TNS、 WHS 和 THS 的主要检查工程源于连续单元的功能需求：

- 建立时间：为安全采集数据，在下个有效时钟沿到来前必须提供最新稳定数据就绪所需的时间。
- 保持要求：是指有效时钟沿到来后为避免捕获无用值使数据须保持稳定的总时间。
- 恢复时间：将异步复位信号切换到无效状态以及下一个有效时钟沿所需时间之间的最短时间。
- 移除时间：有效时钟沿之后、将异步复位信号安全切换到其非活动状态之前所需的最少时间。

一个简单实例是两个连接到相同时钟网络的触发器之间的路径。

在时钟网络上定义时序时钟之后，时序分析就会对目标触发器的数据引脚在最保守但又合理的工作条件下执行建立和保持检查。当建立和保持时序裕量都通过后，从源触发器到目的触发器之间的数据传输就能安全进行。

如需了解更多时序分析的信息，请参阅 《Vivado Design Suite 用户指南：设计分析和收敛技术》(UG906) [参照 21] 中的链接。

检查设计是否正确约束

在查看时序结果是否有违规问题之前，应确保设计中的每个同步端点都被正确约束。

运行 `check_timing` 以识别未约束的路径。您可将该命令作为独立命令运行，但其也是 `report_timing_summary` 的一部分。此外， `report_timing_summary` 还包括 Unconstrained Paths 部分，其中已定义的源或目的地时序时钟会列出没有时序要求的 N 个逻辑路径。 N 由 `-max_path` 选项控制。

对设计进行全面约束后，运行 `report_methodology` 命令并查看 TIMING 和 XDC 检查，以确定其是否为非最佳约束，这可能使时序分析不完全准确，并会导致硬件的时序裕度发生变化。

注释： 对于 Vivado Design Suite 2015.4 及更早的版本，请改用 `report_drc -ruledeck methodology_checks` Tcl 命令。

修复由 check_timing 标记的问题

`check_timing` Tcl 命令会报告在时序定义中缺失或错误的东西。当检查和修复由 `check_timing` 标记的问题时，首先应关注最重要的检查。下列进行检查的顺序重要性从高到低排列。

无时钟和未约束的内部端点

这将帮助您确定是否对设计中的内部路径进行了完整约束。作为静态时序分析最终质量检查的一部分，必须确保未约束的内部端点数为零。

无约束内部端点为 0 代表针对时序分析对所有内部路径进行了约束。这只能但不能保证约束值是正确的。

生成时钟

生成时钟是设计的正常组成部分。但是，如果生成时钟的主时钟不是同一个时钟树的一部分，就会带来严重问题。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错误。最糟糕的情况是，从报告上看设计满足时序要求，但是在硬件中无法工作。

环路和锁存器环

因为时序环路被时序引擎打断，所以良好的设计都不存在任何组合环路。被打断的路径无法在时序分析过程中进行报告，或无法在实现中评估。这样，即使整体时序要求得到满足，但也会导致硬件中出现错误行为。

无输入/输出延迟和部分输入/输出延迟

所有 I/O 端口必须得到正确的约束。

建议： 首先验证基线约束，然后再使用 I/O 时序完成约束。

多个时钟

通常可接受多个时钟。赛灵思建议确保这些时钟在相同的时钟树上传输。还需核实这些时钟之间的路径不会引入超过实际需要的更严格要求，以便让设计在硬件正常工作。

如果是这种情况，必须在这些路径上的时钟之间使用 `set_clock_groups` 或 `set_false_path`。您在使用时序例外时，必须确保它们只影响目标路径。

重要提示： 因为 XDC 遵循 Tcl 句法和语义规则，所以约束的顺序至关重要。

修复被 report_methodology 标记的问题

`report_methodology` 命令可报告其他的约束及时序分析问题，您必须在运行布局和布线工具的前后进行仔细检查。本章节将介绍需要检查的三大 XDC 和 TIMING 类别，以及它们对时序收敛和硬件稳定性的相对影响。首先，您必须关注解决会影响时序收敛的检查。

方法 DRC 对时序收敛的影响

下表中显示的 DRC 可标志设计和时序约束组合，这会增加实现工具的压力，从而导致不可能或不一致的时序收敛。这些 DRC 通常指向缺少跨时钟域 (CDC) 约束、不适当的时钟树或由于逻辑复制而导致不一致的时序例外覆盖。我们必须以最高的优先级处理。

表 5-1：时序收敛方法 DRC

检查	描述
TIMING-6	在相关时钟之间不存在通用主时钟
TIMING-7	在相关时钟之间不存在通用节点
TIMING-8	在相关时钟之间不存在共同周期
TIMING-14	时钟树上的 LUT
TIMING-15	时钟间路径上的大量保持违规
TIMING-16	大量建立违规
TIMING-30	所选用于生成的时钟的主引脚源不理想
XDCB-3	在同一个的 `set_clock_groups` 命令中，多个组中提到同一时钟
XDCH-1	多周期路径约束中缺少保持选项
XDCV-1	由于缺少在复制中使用的初始对象而导致的不完整约束覆盖
XDCV-2	由于缺少复制对象而导致的不完整约束覆盖

方法 DRC 对验收质量的影响

下表中显示的 DRC 通常不会标记为影响时序收敛难度的问题。相反，由于非建议的约束，这些 DRC 标记了时序分析精度的问题。即使建立并保持裕量为正，硬件可能无法在所有工作条件下正常工作。大多数检查是指未在设计边界上定义的时钟，具有意外波形的时钟，缺少时序要求或不适当的 CDC 电路。对于最后一类，使用 `report_cdc` 命令执行更全面综合的分析。

表 5-2：验收质量方法 DRC

检查	描述
TIMING-1、 TIMING-2、 TIMING-3、 TIMING-4 和 TIMING-27	非建议的时钟源点定义
TIMING-5、 TIMING-25 和 TIMING-19	意外的时钟波形
TIMING-9、 TIMING-10	未知或不完整的 CDC 电路
TIMING-11	不适当的 `set_max_delay -datapath_only` 命令
TIMING-12	时钟重新收敛保守消除禁用
TIMING-13、 TIMING-23	由于路径断开导致的不完整时序分析
TIMING-17、 TIMING-18、 TIMING-20 和 TIMING-27	缺少时钟或输入/输出延迟约束
TIMING-21、 TIMING-22	MMCM 补偿的问题
TIMING-24	改写 `set_max_delay -datapath_only` 命令
TIMING-29	不一致的多周期路径对
TIMING-35	在具有相同时钟的路径中不存在公共节点

其他时序方法 DRC

其他 TIMING 和 XDC 检查确定的约束可能导致更多运行时间，覆盖现有约束或对网表名称更改变得高度敏感。相应的信息对于调试约束冲突非常有用。您必须特别注意 TIMING-28 检查 （由时序约束引用的自动衍生时钟），因为在修改设计源代码和进行重新综合时，自动衍生的时钟名称可能会改变。在这种情况下，先前定义的约束将不再工作或将应用于错误的时序路径。

设计基准 (baseline)

创建基准约束意味着要生成最简单的时序约束集。在完整约束所有时钟之后，会自动约束设计中所有具有起点和端点的路径。这样就提供了一种轻松易行的机制，即使设计在不断演进变化也能确定内部器件存在的时序难题。此外，由于设计还可能存在跨时钟域，因此基线 (baseline) 约束还应包括指定时钟 （包含生成时钟）之间的关系。

基线 (baseline) 设定的主要目标是创建一个正确的极简约束集，该约束集可覆盖大部分时序路径，而不是一直等待，直到所有约束都能被完全指定被完全指定。所以，基线 (baseline) 约束不包括 I/O 时序约束。相反， I/O 时序定义和收敛要等到过后才调用，即设计已取得显著的演进而且对 I/O 时序有更好的了解。

建议： 赛灵思建议您在设计过程的早期即创建基线约束，并根据这些基线约束来计划设计 HDL 的任何重大更改。

提示： 如果您遇到基准 (baseline) 约束的时序问题，请参阅“分析并解决时序违规”。

所有赛灵思及其合作伙伴的 IP 核配套提供符合赛灵思约束方法的 XDC 特定约束。这些 IP 约束会自动加入到综合和实现过程中。创建设计基准约束时，必须保留它们。

提示： 在按设计流程逐步操作和优化设计约束时，请完成“基线 （baselining）与时序约束验证流程”。

定义基准 (baseline) 约束

如果您对时钟约束不确定，可利用 Vivado IDE 在综合后的网表上创建完整的时钟约束集。如下面的章节所描述， IDE 的图形界面以及 Vivado Design Suite 的报告功能可准确显示应该约束的内容。

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`，以确保删除所有时序约束。

使用 `report_clock_networks` Tcl 命令创建一个完整的主时钟列表。其中主时钟必须在设计中定义好。这个时钟网络的结果列表会显示应该创建哪些时钟约束。使用 Timing Constraints Editor 为每个时钟指定合适的参数。

确认没有时钟遗漏

在时钟网络报告显示所有时钟网络都受到约束后，您就能开始验证所生成时钟的精度。由于 Vivado 工具会自动在时钟修改块中传输时钟约束，因此一定要检查所生成的约束。使用 `report_clocks` 来显示哪些时钟是用 `create_clock` 约束创建的，哪些时钟是生成的。

注释： MMCM、 PLL 和时钟缓冲器均是时钟修改块。此外，对于 UltraScale™ 器件， GT 也属于时钟修改块。

`report_clocks` 结果显示所有时钟都被传输。主时钟 （用 `create_clock` 创建）与生成时钟之间的区别在属性字段中显示。

- 被传输的仅显示为 (P) 的时钟为主时钟。
- 从其他时钟衍生的时钟既可以显示为传输 (P) 也可以显示为生成 (G)。
- 时钟修正块生成的时钟显示为自动衍生 （A）。
- 其他属性提示自动衍生时钟曾被重命名为 （R），生成时钟相对于输入主时钟波形反向 （I）或主时钟为虚拟时钟 (V)。

您还可以使用 `create_generated_clock` 约束来创建生成时钟。如需了解更多信息，请参阅《Vivado Design Suite 用户指南：如何使用约束》(UG903) [参照 18]。

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

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

图 5-2：时钟报告显示从主时钟生成的时钟

提示： 要验证设计中没有无约束的端点，请参阅检查时序报告 （no_clock 类别）。该报告可从 Report Timing Summary 中访问或使用 `check_timing` Tcl 命令获取。

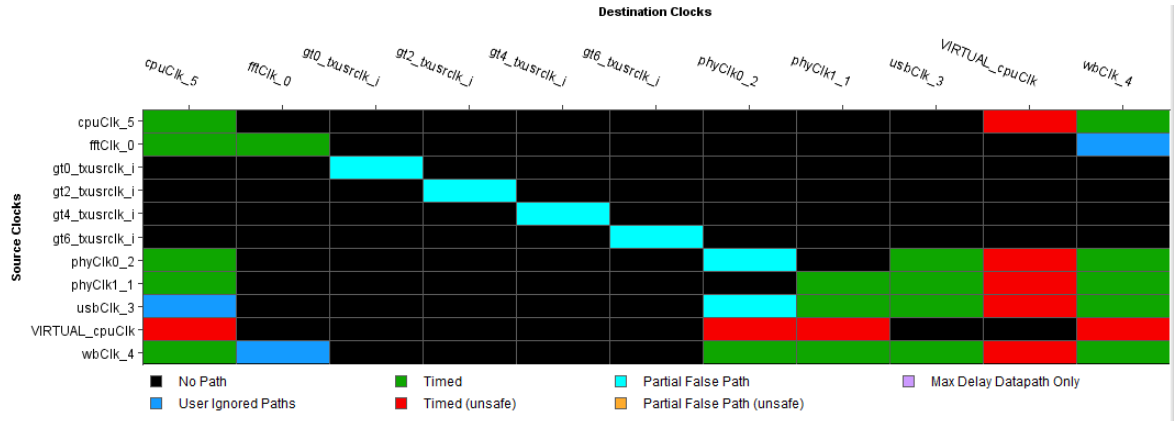
跨时钟域约束

在验证时钟约束时，您必须确定异步和超结束的跨时钟域路径。

注释： 本章节不对如何正确地跨时钟域边界做阐述。而是阐述如何找出跨时钟的边界以及如何对其进行约束。

检查时钟关系

您能使用 `report_clock_interaction` Tcl 命令查看时钟之间的关系。该报告给出一个源时钟和目标时钟的表格。每个单元中的颜色都表示时钟之间的交互类型，其中包括它们之间的任何现有约束。下图显示了一个时钟交互报告实例。



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

图 5-3：时钟交互报告实例

下表介绍了报告中每种颜色的含义。

表 5-3：report_clock_interaction 颜色

颜色	标签	含义	备注
黑	无路径	这些时钟域中无交互。	主要用于参考，除非您希望这些时钟域交互。
绿	已运行时序约束	这些时钟域间有交互，而且路径已被定时。	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	部分伪路径	交互时钟域的某些路径因用户自定义例外没有被定时。	确定真的需要设置时序例外。
红	已运行时序约束（不安全）	时钟域之间有交互，而且路径已被定时。但是，时钟似乎是独立的 （因此是异步）	检查这些时钟是否应该标记为异步，或者它们是否应为共享共用的主时钟源。
橙	部分伪路径（不安全）	这些时钟域间有交互。时钟似乎是独立的 （因此是异步）。但是，只有部分路径因异常而未被定时。	检查为什么时序例外没有覆盖某些路径。
蓝	用户忽略的路径	在这些时钟域之间存在交互，并且由于时钟群或伪路径时序例外，没有对路径进行时序约束。	确认这些时钟应该是异步的。另外，检查相应的 HDL 代码是否正确编写，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	这些时钟域互动且路径通过如下方式获得时序：`set_max_delay -datapath only`.	确认时钟处于异步状态且设定的延迟调整正确。

在创建任何伪路径或时钟组约束之前，表格中出现的颜色只有黑、红和绿。由于所有时钟在默认情况下都被定时，因此去耦异步时钟的过程非常重要。异步时钟去耦失败通常会导致高度过约束的设计。

识别没有主时钟的时钟对

时钟交互报告指出每个交互时钟对是否具有共用的主时钟源。不共享共用主时钟的时钟对经常是彼此异步的。因此，使用共用 Common Primary Clock 字段整理报告中的各列，这对识别这些时钟对很有帮助。报告没法确定跨路径的时钟域在设计上是否合理。

使用 `report_cdc` Tcl 命令对异步时钟之间的跨时钟域电路进行全面分析。如需了解更多有关 `report_cdc` 命令的信息，请参阅 《Vivado Design Suite 用户指南：设计分析和收敛技术》(UG906) [参照 21] 中的连接和 《Vivado Design Suite Tcl 命令参考指南》(UG835) [参照 13] 中的 report cdc。

明确严格的时序要求

此外，对于每个时钟对，时钟交互报告还可显示最差路径的建立要求。分类“Path Req (WNS)”对列排序，可查看设计中最严格的要求。图 5-3 显示了按 WNS 列排序的时序报告。应检查这些时序要求以确保不存在无效要求。

Vivado 工具将每个时钟扩展至 1000 个时钟周期，然后确定哪里出现最近的不一致时钟沿对齐情况，以此明确路径要求。当 1000 个周期不足以确定最严格的要求时，报告显示 Not Expanded，在这种情况下，必须将两个时钟视为异步。

例如，考虑一条从 250 MHz 时钟到 200 MHz 时钟的时序路径：

- 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，会对时钟对施加最严格的要求：

- 250 MHz 时钟在 4 ns 具有一个上升沿。
- 200 MHz 时钟的下一个上升沿在 5 ns。

这会导致从 250 MHz 时钟域到 200 MHz 时钟域的所有路径被定时在 1 ns。

注释： 在 20 ns 的同步时钟沿在本例中不是最严格要求，因为捕获沿不能与发送沿相同。

因为这是一个相当紧迫的时序要求，所以您必须采取额外的步骤。根据设计的不同，下面的约束之一可能成为处理这些跨越问题的正确方法：

- `set_clock_groups/set_false_path/set_max_delay -datapath_only`

  将时钟对视为异步时，则使用这些约束的其中之一。使用 `report_cdc` Tcl 命令来验证跨时钟域电路是否安全。

- `set_multicycle_path`

  在放宽时序要求时使用此约束，从而假定适当的时钟电路可相应控制发射和捕获时钟边沿。

如果什么也不做，设计可能会出现跨越这两个时钟域的时序违规问题。此外，所有最佳优化、布局、布线等最终可能被专门用于这些路径而不是设计中的关键路径。在执行任何时序驱动的实现步骤之前，一定要鉴别出这类路径，这一点很重要。

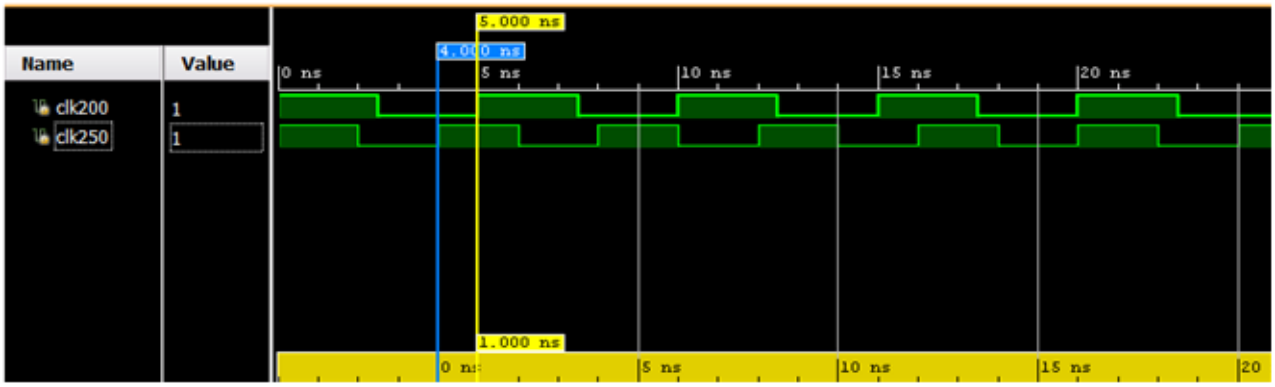


图 5-4：从 250 MHz 到 200 MHz 的时钟域

同时约束主时钟和生成时钟

在创建时序例外之前，最好返回 report_clock_networks，确定设计中存在哪些主时钟。通常所有主时钟都是彼此异步的，利用单个约束即可将主时钟去耦，并去耦它们的生成时钟。如下图所示，使用 report_clock_networks 中的主时钟作为指导，您就能将每个时钟组和关联的时钟进行去耦。

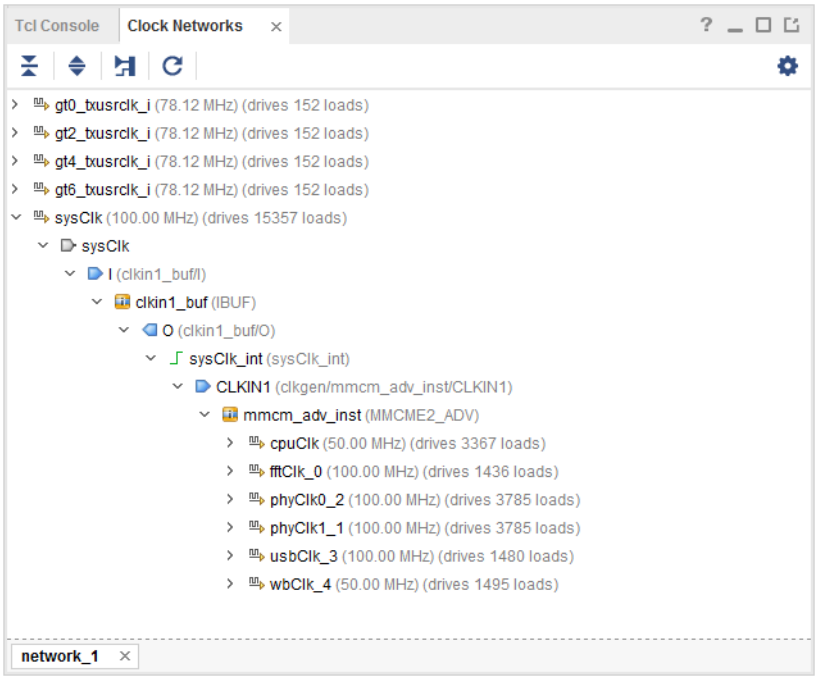


图 5-5：报告时钟网络

```
### 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

大部分时序违规都存在于内部路径上。在首次基准 (baseline) 设计迭代过程中无需 I/O 约束，尤其对于发送和捕获寄存器位于 I/O bank 内部的 I/O 时序路径来说更不需要 I/O 约束。在设计及其它约束已稳定并且时序接近收敛之后，您就能添加 I/O 时序约束。

提示： 您最开始可以使用 Vivado Design Suite 2015.3 版本，之后再用 `config_timing_analysis -ignore_io_paths yes` Tcl 命令在设计实现期间和使用时序信息的报告中忽略所有 I/O 路径上的时序。您必须在打开存储器中的设计之前或之后手动输入此命令。

以 RTL 设计师的建议为基础，时序例外必须受到限制，而且确保不至于掩盖真正的时序问题。在这个时候，必须检查和最终确定时钟之间的伪路径或时钟组。

必须完全保留 IP 约束。若 IP 时序约束丢失，已知的伪路径会被报告为时序违规。

在每个步骤前后评估设计 WNS

必须在每个实现步骤之后评估设计的 WNS。如果您正在使用 Tcl 命令行流程，那么您就能在每个实现步骤之后轻松将 `report_timing_summary` 整合到您的构建脚本中。如果您在使用 Vivado IDE，则能在每个步骤后使用简单的 `tcl.post` 脚本来运行 `report_timing_summary`。在两种情况下，当发现 WNS 明显变差时，都必须在实施本步骤之前立即对检查点进行分析。

除了在每个实现步骤前后评估整个设计的时序之外，还可采取面向单个路径的更有针对性的方案，以评估流程中每个步骤对时序的影响。例如，某个时序路径在优化后的估计网络延迟可能与该路径在布局后的估计网络延迟存在很大差异。在每个步骤后比较关键路径的时序是找出关键路径时序在哪里出现收敛偏离的有效方法。

综合后与逻辑优化后

估计网络延迟接近于所有路径的最佳布局。请尝试通过下列方式解决违规路径问题：

- 修改 RTL。
- 使用不同的综合选项。
- 如果对于硬件中的功能来说是安全适用的，就可以添加时序例外，例如多周期路径等。

布局前后

在布局后，除了采用更悲观延迟的长距离和中高扇出网络以外，估计网络延迟接近于最佳布线。此外，拥塞或保持修复的影响未计算在网络延迟中，这让时序结果比较乐观。

精确估计时钟偏差并用于检查不均衡时钟树对时序裕量的影响。

您能通过运行最小延迟分析估算保持固定。slice、块 RAM 或 DSP 之间的 WHS 为 -0.500 ns 或更大的高保持违规将需要修复。如果违规程度较低，可以接受而且还有可能被布线器修复。

注释： 往/返于诸如 PCIe® 块等专用块的路径能具有大于 -0.500 ns 的保持时间估计，其由布线器进行自动固定。对于这些情况，请在布线之后检查 `report_timing_summary` 以验证所有相应的保持违规是否都已修复。

物理优化前后

修复以下相关时序问题之前先评估执行物理优化的必要性：

- 具有高扇出的网络 （`report_high_fanout_nets` 显示最高扇出非时钟网络）
- 载荷和驱动距离较远的网络
- 流水线寄存器的使用为次优化的 DSP 和 RAMB

预布线和后布线

报告实际布线的网路延迟 （除了还没有被完全布线的网络）的时序裕量。时序裕量反映出保持修复对建立以及拥塞的影响。

无论最差建立裕量 (WNS) 值是多少，布线后也不应保持违规。如果设计保持失败，就需要进一步分析。这一般由非常严重的布线拥塞引起，此时布线器放弃对时序进行优化。此外，这也会发生于高保持违规 （超过 4 ns）的情况，这种情况下布线器不进行默认修复。高保持违规通常是由不正确的时钟约束、高时钟偏差，或不正确的 I/O 约束造成，本应在布局甚至综合后即可得到解决。

如果保持得到满足 (WHS>0)，但建立失败 (WNS<0)，应遵循在 “分析并解决时序违规” 中所述的分析步骤。

基线 （baselining）与时序约束验证流程

下面流程有助于用户跟踪实现时序收敛的进展情况及确定潜在的瓶颈。

1. 打开综合设计。

2. 运行 report_timing_summary -delay_type min_max，然后在下表中记录显示的信息。

	WNS	TNS	失效端点数	WHS	THS	失效端点数
综合						

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

   设计中丢失的时钟要求数量：_____

4. 运行 report_clock_networks，确定设计中的主时钟源引脚/端口数。（忽略 QPLLOUTCLK、QPLLOUTREFCLK，因为它们属于纯脉冲宽度检查。）

   设计中无约束时钟数量：_____

5. 运行 report_clock_interaction -delay_type min_max，然后按 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，检查是否综合后是否存在较大数值的保持时间违规（大于 500 ps）。

   设计中最大负 WHS：_____

7. 按时钟间约束 （Inter-Clock Constraints）对 report_clock_interaction 结果进行排序，列出所有显示为不安全的时钟对。
8. 在打开综合设计时，出现多少个 CRITICAL_WARNINGS？

   综合设计中 CRITICAL_WARNINGS 的数量：_____

9. 存在哪些类型的 CRITICAL_WARNINGS？

   记录每种类型的示例。

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

    并非由 FF 驱动的高扇出网络数量：_____

    并非由 FF 驱动的最高扇出网络上的负载数量：_____

    有没有高扇出网络有负时序裕量？如果有， WNS= _____

11. 实现设计。完成每一步后都应运行 report_timing_summary，并把显示的息记录在下表中。

	WNS	TNS	失效端点数	WHS	THS	失效端点数
Opt						
布局						
Physopt						
走线						

运行 report_exceptions -ignored，查看设计中是否存在约束重叠。记录结果。

分析并解决时序违规

时序驱动的算法侧重于最严重的违规。了解并修复与最严重违规相关的问题，可能会解决绝大部分在重新运行实现流程中出现的较小的违规问题。您必须识别引起每个违规的主要时序的特性，然后应用本章所描述的相应解决技术进行解决。

确定时序违规的根源

为进行建立，必须首先分析每个时钟组的最严重违规。时钟组是指由特定时钟捕获到的所有内部路径、时序域间路径和异步路径。

对于保持，所有违规都必须按以下方式进行审核：

- 在布线之前，仅检查超过 0.5 ns 的违规。
- 布线后，应从最恶劣的违规开始。

检查时序裕量

有几个因素会影响建立与保持时间时序裕量。当时间建立与保持时序裕量计算公式被填入下列精简表格时，检查这些公式，就可以轻易地确定每个影响因素。

时序裕量 （建立/恢复）	=	建立路径要求 – 数据路径延迟 （最大量） + 时钟偏差 – 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 （最小量） – 时钟偏差 – 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 （位于公共节点后，如果有）

在分析违规时钟路径的过程中，必须检查与每一个变量有关的影响，以确定哪个变量最有可能导致违规。然后可以开始分析导致违规的主要原因，以便了解路径的哪种特性对它的值影响最大，并且尝试确定一种设计或者约束修改，以减少这个特性的影响。如果一种设计或约束修改不能解决实际问题，必须从导致最严重违规的那个原因开始，用所有其他原因做相同的分析。以下清单显示了导致违规的典型原因，顺序从影响最大到最小排列。

有关建立/恢复时间：

- 数据路径延迟：将时序路径要求从数据路径延迟中去掉。如果这个差异相当于 （负的）时序裕量值，那么不仅路径要求太严格，数据路径延迟也太大。
- 数据路径延迟 + 建立/恢复时间：将时序路径要求从数据路径延迟加建立/恢复时间去掉。如果这个差异相当于（负值）序裕量值，那么不仅路径要求过于严格，而且建立/恢复时间也大于通常值，并且会明显导致违规。
- 时钟偏差：如果时钟偏差和时序裕量有类似的负值，并且偏差绝对值超过数百 ps，那么这个偏差是个引起违规的较大原因，必须检查拓扑结构。
- 时钟不确定性：如果时钟不确定性超过数百 ps，那么必须检查拓扑结构与抖动数量，以便了解不确定性如此高的原因。

有关保持/移除时间有关：

- 时钟偏差：如果时钟偏差超过 300 ps，必须检查时钟结构。
- 时钟不确定性：时钟不确定性：如果时钟不确定性超过数百 ps，那么必须检查拓扑结构与抖动数量，以便了解不确定性如此高的原因。
- 保持/移除时间：保持/移除时间超过数 100 ps，可查看原始数据手册，以验证这个值在预期范围内。
- 要求通常为零。如果要求不为零，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，则造成时序违规最常见的原因通常是建立/恢复路径引起的数据路径延迟，以及用于保持/移除时序路径造成的偏差。在设计周期的早期阶段，通过分析这两个原因，可修复多数时序问题。然而，在改进和优化设计和约束之后，剩余的违规是由多种因素的组合造成的，您必须并行检查所有因素，才能确定需要改进的因素。

如需了解更多有关时序分析概念的信息，请参阅链接；如需了解更多有关 《Vivado Design Suite 用户指南：设计分析和收敛技术》(UG906) [参照 21] 中时序报告 (`report_timing_summary/report_timing`) 的信息，请参阅链接。

使用设计分析报告

当时序收敛难以实现时，或者当您尝试提高应用程序的整体性能时，您必须在运行综合之后与执行流程的任何步骤之后查看设计的主要特性。 QoR 分析通常需要您同时查看数个全局和局部特性，以确定在设计和约束中什么是欠佳，或者哪种逻辑结构不适合目标器件架构和实现工具。 `report_design_analysis` 命令可在几个表中收集逻辑、时序和物理特性，从而简化 QoR 根本原因分析。

注释： `report_design_analysis` 命令不会报告时序约束的完整性和正确性。如需了解更多有查看和修复时序约束的信息，请参阅“检查设计是否正确约束”。

提示： 在 Vivado IDE 中运行设计分析报告，以改进可视化、自动过滤功能和便利的交叉探测。

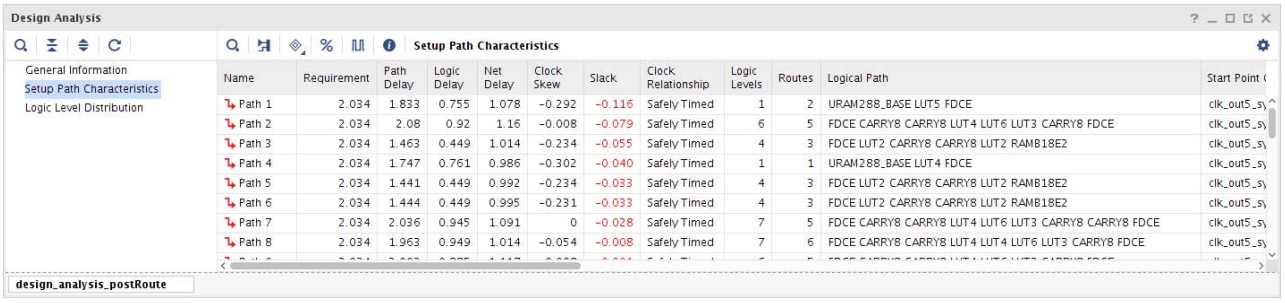
以下部分仅介绍时序路径特性分析。如“识别拥塞”所述， “Design Analysis”报告还提供有关拥塞和设计复杂性的实用信息。

分析路径特性

您能够使用以下命令来报告 50 个最差的建立时序路径：

```
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 表格的范例。要想在窗口中查看额外的列，可进行水平滚动。



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

图 5-6：报告设计分析时序路径特性后布线

以下是使用此表格的方式：

- 通过单击“%” （显示百分比）按钮在数字和 % 之间切换。这对于检查单元延迟和网络延迟的比例特别有帮助。
- 默认情况下，仅隐藏 null 或空值的列。单击“Hide Unused”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。

从此表中，您能够隔离哪些特性正在为每个路径引入时序违规：

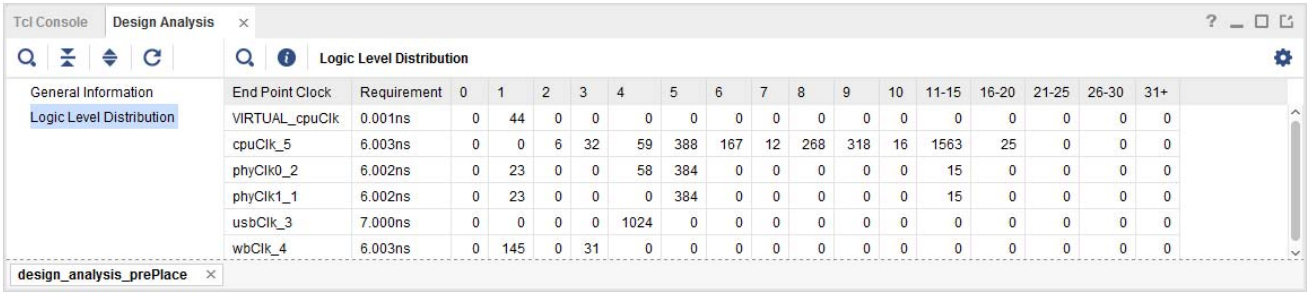
- 高逻辑延迟百分比 (Logic Delay)
  - 有很多逻辑层次吗？ (LOGIC_LEVELS)
  - 是否有阻碍逻辑优化的任何约束或属性？ (DONT_TOUCH、 MARK_DEBUG)
  - 路径是否包括具有高逻辑延迟的单元，例如块 RAM 或 DSP 等？ （Logical Path、 Start Point Pin Primitive、 End Point Pin Primitive）
  - 当前路径拓扑结构的路径要求是否过于严格？ （要求）
- 高网络延迟百分比 (Net Delay)
  - 在路径中是否有任何高扇出网络？ （High Fanout、 Cumulative Fanout）
  - 分配给几个 Pblock 的单元是否能远距离布局？ (Pblock)
  - 是否可将单元的距离放置较远分开？ （Bounding Box Size、 Clock Region Distance）
  - 对于 SSI 技术设备，是否存在跨越 SLR 边界的网络？ （SLR Crossings）
  - 在布局似乎正确的情况下，是否一个或几个网络延迟值远远高于预期？在“Device”窗口中选择路径并显示其布局和布线。
  - 在块 RAM 或 DSP 单元中是否缺少流水线寄存器？ （Comb DSP、 MREG、 PREG、 DOA_REG、 DOB_REG）
- 高偏差 （建立<-0.5 ns，保持> 0.5 ns）（时钟偏差）
  - 是跨时钟域路径吗？ （Start Point Clock、 End Point Clock）
  - 时钟是同步还是异步？ (Clock Relationship)
  - 是路径穿越 I/O 列吗？ (IO Crossings)

提示： 要在赛灵思 Vivado IDE 中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”标签。

查看逻辑电平分布

此外， report_design_analysis 命令还会为最差的 1000 条路径 （默认值）生成 Logic Level Distribution 表格，您可以使用它们来确定设计中存在的较长路径。通常由布局器对最长路径进行优化以便满足时序，这将潜在地降低较短路径的布局质量。您必须始终尝试消除更长的路径，以提高整体 QoR。因此，赛灵思建议在布局之前检查最长路径。下图显示了设计的 Logic Level Distribution 示例，其中在最差的 5000 个路径中包括具有 17 个逻辑电平的困难路径，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
```



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

图 5-7：报告设计分析时序路径特性预布局

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}]
```

根据您发现的内容，您可以通过更改 RTL 或使用不同的综合选项来完善网表，也能修改时序和物理约束。

数据路径延迟和逻辑级数

通常，路径中 LUT 与其他原语的数量是引起延迟的极其重要的因素。如下所述，由于在 7 系列和 UltraScale 器件中 LUT 延迟的报告方式有所不同，因此必须考虑单独的单元延迟和布线延迟范围。

如果造成路径延迟主要是因为：

- **7 系列器件的单元延迟 >25%， UltraScale 器件的延迟 >50%**

  路径是否可被改短或者使用更快的逻辑单元？请参阅“减少逻辑延迟”。

- **7 系列器件的布线延迟 >75%， UltraScale 器件的延迟 >50%**

  路径是否受到保持修复影响？您能够通过运行 report_design_analysis 并检查“Hold Detour”列来确定这一点。使用相应的分析技术。

  - 是–受影响的网络是否处于 CDC 路径中？
    - 是–CDC 路径是否丢失一个约束？
    - 否–保持修复后路径的起点和端点是否使用平衡时钟树？查看偏差值。
  - 否 - 请参阅有关拥塞的以下信息。

  路径是否受到拥塞影响？查看每个网络延迟和扇出，并观察“Device”视图（打开布线细节）中的布线（仅布线后分析）。还可打开拥塞指标以查看路径是否位于或靠近拥塞区域。使用以下分析步骤进行快速评估或查看综合分析“识别拥塞”。

  - 是–对于具有最高延迟值的网络，扇出是否比较低 (<10)？

- 是–如果布线看起来很理想 （直线），但驱动器与负载离得较远，那么次优化布局与拥塞有关。检查“解决拥塞”以确定最佳解决技术。
- 否–尝试使用物理逻辑优化来复制网络的驱动器。复制后，每个驱动器可自动放在离负载更近的位置，从而缩短整体数据路径延迟。如需了解更多详细信息和学习相关替代技术，查看“优化高扇出网”。

。 否–设计展开过度。请尝试使用以下技巧之一来完善布局：

- “减少控制集”
- “调节编译流程”
- “布局规划考虑因素”

时钟偏差与不确定性

赛灵思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。时钟偏差会影响任何具有组合逻辑或互联的寄存器至寄存器路径。

建议： 运行设计分析报告 (report_design_analysis) 以生成时序报告，其中包括有关时钟偏差数据的信息。验证时钟网络不包含过度时钟偏差。

高性能时钟域 （超过 300 MHz）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 或 2 个逻辑电平的时序预算。在跨域时钟路径中偏差可能更高，原因是这些时钟使用了不同的资源，并且公共节点位于时钟树向上更深一层。基于 SDC 的工具为所有时钟一起定时，除非约束设定它们不应该被一起定时，例如：

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

如果您质疑高时钟偏差，请检查 “减少时钟偏差” 和 “减少时钟不确定性”。

减少逻辑延迟

Vivado 实现首先关注最关键的路径，这通常使不太困难的路径在布局之后或布线之后变得至关重要。赛灵思建议在综合之后或 opt_design 之后识别和改进最长路径，因为它将对 QoR 产生最大的影响，并且通常会显著减少布局和布线迭代的次数以实现时序收敛。

在布局之前，时序分析可使用对应于理想布局和典型时钟偏差的估计延迟。通过使用 report_timing、report_timing_summary 或 report_design_analysis，您能够快速识别具有过多逻辑电平或具有高单元延迟的路径，因为它们通常在布局之前计时失败或几乎不能达到计时要求。使用 “确定时序违规的根源” 提出的方法来查找在实现设计之前需要改进的长路径。

优化常规结构路径

常规结构路径是结构寄存器或移位寄存器之间的路径，并且穿过 LUT、 MUXF 和 CARRY 的混合。
report_design_analysis Timing Path Characteristics 表格可提供能识别以下问题的最佳逻辑路径拓扑结构摘要：

- 几个小 LUT 均为级联：映射至 LUT 受层级、存在 KEEP_HIERARCHY、 DONT_TOUCH 或 MARK_DEBUG 属性的影响，具有一些扇出 （10 和更高）的中间信号。尝试运行 opt_design -remap 选项或 AddRemap 指令以尝试折叠较小的 LUT 并减少逻辑电平的数量。如果 opt_design 无法优化最长路径，请在“Elaborated”视图中分析路径，并重新构建 RTL 描述。
- 单个 CARRY 单元存在于路径中。CARRY 原语在处于级联时对 QoR 最有利。CARRY 单元比 LUT 更难布局，并且迫使综合使用 LUT 而不是单个 CARRY，这使得在许多情况下能够实现更好的 LUT 结构化和更灵活的布局。尝试 FewerCarryChains 综合指令或 PerfThresholdCarry 策略 （仅项目模式），以消除大多数单个 CARRY 单元。
- 路径在移位寄存器 (SRL) 处结束：通过使用 RTL 中的 SRL_STYLE 属性将第一个寄存器拉出移位寄存器。如需获取更多信息，请查看 《Vivado Design Suite 用户指南：综合》(UG901)[参照 16] 中的链接。

- 结构寄存器 (FD) 时钟使能或同步建立/复位时结束的路径：如果结束于数据引脚 (D) 的路径具有更多的裕量和更少的逻辑电平，则使用 EXTRACT_ENABLE 或 EXTRACT_RESET 属性，并基于在 RTL 中的信号之上将其布局为“No”。如需获取更多信息，请参阅 第 3 章中的“将逻辑从控制引脚推到数据引脚”。

提示： 如需从后综合路径到相应的“RTL”视图和源代码进行交叉探测，请在 《Vivado Design Suite 用户指南：设计分析和收敛技术》(UG906) [参照 21] 中查看链接。

使用专用块和宏原语优化路径

从专用块和宏原语 （例如 DSP、块 RAM、 FIFO 或 GT_CHANNEL） /到/在它们之间的路径需要特别注意，因为这些原语通常具有以下时序特性：

- 某些引脚的更高建立/保持/时钟至输出时序 arc 值。例如，块 RAM 具有大约 1.5 ns 的时钟到输出延迟，没有可选的输出寄存器和 0.4 ns 的可选输出寄存器。查看您目标器件架构的数据手册，了解完整的详细信息。
- 比常规 FD/LUT 连接更高的布线延迟
- 比常规 FD-FD 路径更高的时钟偏差变化

此外，与 CLB slice 相比，它们的可用性和网站位置受到限制，这通常使得它们的布局更具挑战性并经常引起一些 QoR 惩罚。

鉴于这些原因，赛灵思建议：

- 尽可能妥善管理往返于 Dedicated Blocks 和 Macro Primitives 的路径
- 重新调整连接到这些单元的组合逻辑以便如果流水线引起的延迟问题较大，则将逻辑电平降低至少 1 或 2 个单元
- 布局之前，在这些路径上达到至少 500 ps 的建立时间
- 如果它们需要布局得相隔很远，复制连接到太多 Dedicated Blocks 或 Macro Primitives 的逻辑锥

降低网络延迟

欠佳的网络延迟是对具有挑战性的设计或对布局器和路由器算法的不适当约束的结果。最常见的根本原因是：

- 存在物理约束迫使逻辑被布局得很远
- 由于非常高的设备利用率，降低了布局质量 （扩展逻辑）
- 难以布线高扇出网络
- 由于 I/O 列和设备边界等网表复杂性和设备障碍，存在拥塞区域

以下部分介绍几种分析和解决技术。

检查物理约束

所有设计都具有一组最小的物理约束，特别是对于 I/O 位置，有时用于时钟和逻辑布局。虽然在设计时序收敛时准备好不能修改 I/O 位置，但是必须分析诸如 Pblocks 和 LOC 的物理约束。使用 `report_design_analysis` Timing Path Characteristics 表格来识别每个关键路径上存在的若干 Pblocks 约束。

在 Vivado IDE “Properties”窗口中，可以在“Timing Path Characteristic”表格中选择路径，以查看哪些 Pblocks 在路径中约束单元。如果约束会强制逻辑扩展，则可考虑删除一个或多个 Pblock 约束。

识别拥塞

如果将关键路径布局在拥塞区域内部或旁边，或者如果设备利用率高并且布局的设计几乎不可布线，则设备拥塞能潜在地导致时序收敛困难。在许多情况下，拥塞将显著增加布线器运行时间。如果路径显示具有比预期更长的布线延迟，赛灵思建议分析设计的拥塞并确定最佳的拥塞缓解技术。

拥塞区和水平定义

赛灵思 FPGA 布线架构包括每个方向上各种长度的互连资源：北、南、东和西。拥塞区域被报告为覆盖相邻互连块（INT_XnYm）或 CLB 块 （CLE_M_XnYm）的最小正方形，其中在特定方向上的互连资源利用率接近或超过 100%。拥塞级别对应于正方形边长的正整数。下图显示了 UltraScale 设备上的拥塞区域与时钟域的相对大小。

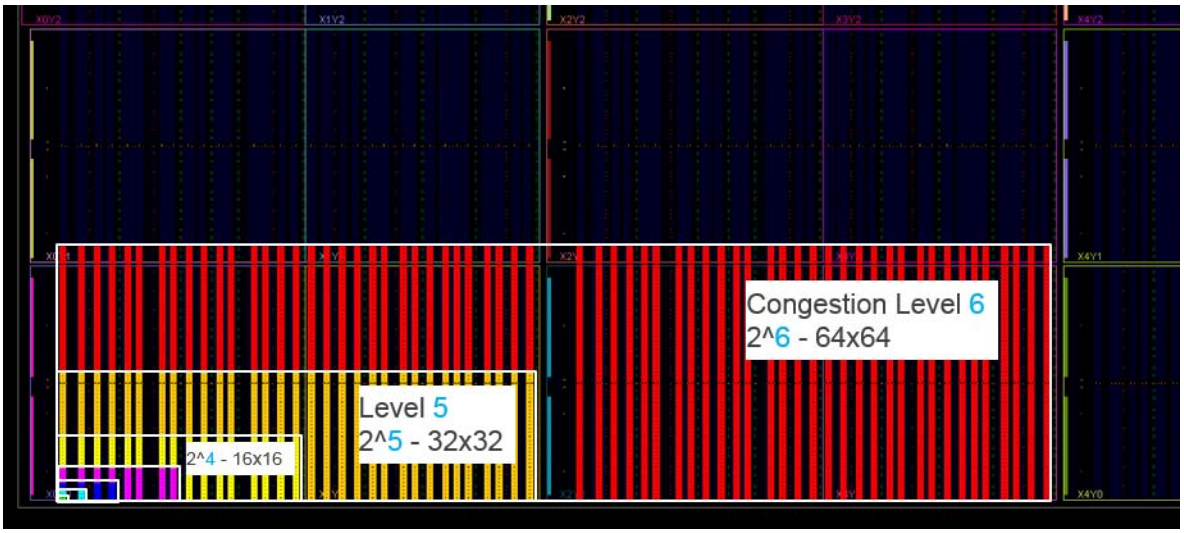


图 5-8：UltraScale 设备视图中的拥塞级别和区域

拥塞级别范围

在分析拥塞时，工具报告的级别可以按照下表所示进行分类。

注释： 拥塞水平 5 或更高通常影响 QoR，并始终导致更长的布线器运行时间。

表 5-4：拥塞级别范围

水平	占位面积	拥塞	QoR Impact
1、 2	2x2、 4x4	无	无
3、 4	8x8、 16x16	温和	可能的 QoR 降级
5	32x32	中等	可能的 QoR 降解
6	64x64	高	难以布线
7、 8	128x128、 256x256	不可能	可能不可布线

设备窗口中每个 CLB 的布线拥塞

每个 CLB 的布线拥塞是基于估计而非实际布线。布局之后或布线之后，您可以通过右键单击设备窗口，选择 Metric，然后选择 Vertical and Horizontal Routing Congestion per CLB 来显示此拥塞指标。这可提供设备中的任何拥塞热点的快速可视概览。下图显示了由于高利用率和网表复杂性导致的带有多个拥塞区域的布局设计。

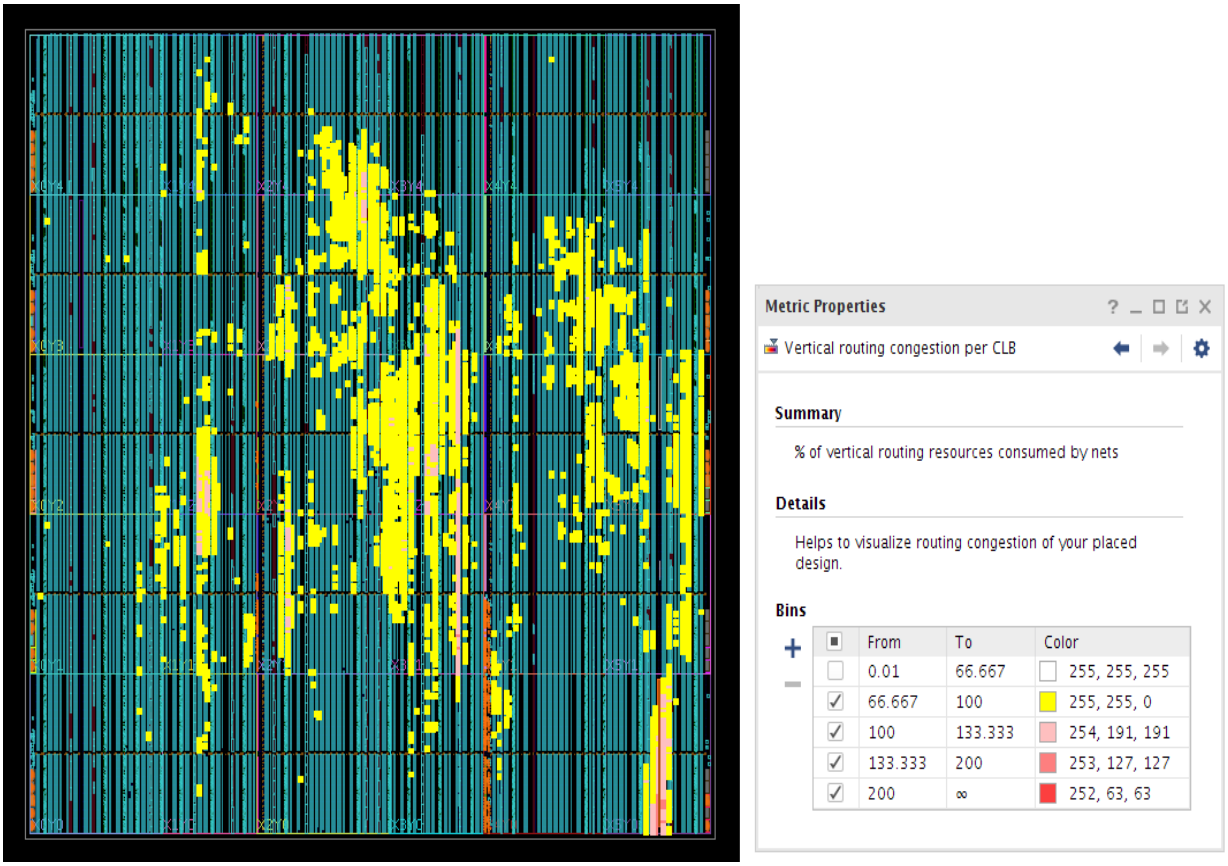


图 5-9：设备窗口中的拥塞示例

在 Placer 日志中的拥塞

布局器估计在整个布局阶段的拥塞并在拥塞区域中扩展逻辑。这有助于减少互连利用率以提高可布线性以及估计与布线延迟的相关性。但是，当由于高利用率或其他原因而无法减少拥塞时，布局器不打印拥塞详细信息，但发出以下警告：

```
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.
```

在这种情况下， QoR 很可能受到影响，谨慎地解决导致拥塞的问题，然后在布线器上继续。如消息中所述，使用 repor t_design_analysis 命令来报告实际拥塞级别并确定它们的位置和布局在同一区域的逻辑。

布线器日志中的拥塞

布线器根据拥塞级别和布局某些资源的难度发出附加消息。此外，布线器还可打印几个中间时序摘要。第一个是布线所有时钟后，通常显示 WNS/TNS/WHS/TNS 号码类似于后布局时序分析。在初始布线之后，报告下一个布线器中间时序摘要。如果时序明显降低，则 QoR 已受保持修复和/或拥塞的影响。

当拥塞级别为 4 或更高时，布线器打印初始估计拥塞表格，给出关于拥塞的性质的更多细节：

- Global Congestion 类似于如何估计布局器拥塞，并且基于所有类型的互连之上。
- Long Congestion 只考虑给定方向的长互连利用率。
- Short Congestion 考虑给定方向的所有其他互连使用率。

任何大于 32x32（级别 5）的拥塞区域都可能会影响 QoR 和可布线性（在下表中以黄色高亮显示）。Long 互连上的拥塞增加了 Short 互连的使用，这会导致更长的布线延迟。 Short 互连上的拥塞通常会导致更长的运行时间，如果它们的瓦数 % 超过 5%，其也可能导致 QoR 退化 （在下表中以红色高亮显示）。

INFO: [Route 35-449] Initial Estimated Congestion

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

图 5-10：初始估计拥塞表

在 Global Iterations，布线器首先尝试找到没有重叠的合法解决方案，并且满足建立和保持的时序，针对保持修复具有更高的优先级。当布线器在全局迭代期间不收敛时，它停止优化时序，直至找到有效的布线解决方案，如下面的示例所示：

```
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，重新启用时序优化。

此外，该布线还标记 CLB 布线拥塞，并提供最拥塞 CLB 的名称。请参阅下面的 INFO 消息。

```
INFO: [Route 35-443] CLB routing congestion detected.Several CLBs have high routing
utilization, which can impact timing closure.Top ten most congested CLBs are:
CLEL_L_X29Y384 CLEL_R_X29Y384 CLE_M_X43Y107 CLEL_R_X43Y107 CLEL_L_X31Y389
CLEL_R_X31Y389
```

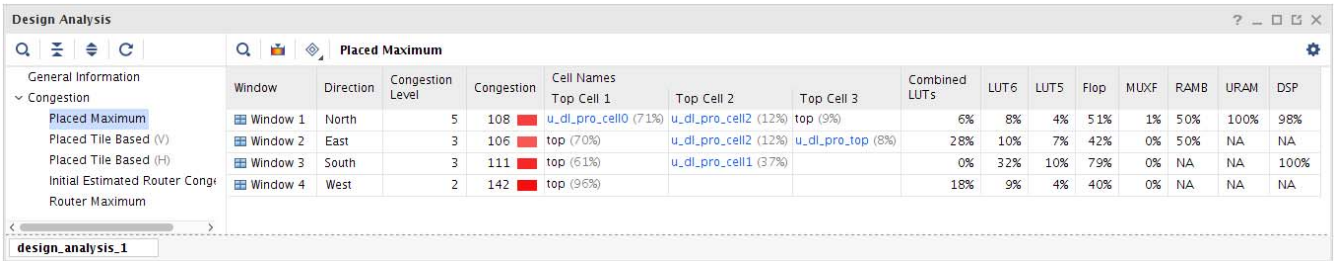
最后，如下所示，当布线器找不到合法布线解决方案时，几个 Critical Warning 消息指示未完全布线的网络数量和具有重叠的互连资源数量。

```
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，网络散布在拥塞区域周围，这通常在对设计进行成功布线时减少在日志文件中报告的最终拥塞级别。

报告设计分析拥塞报告

为了有助于识别拥塞， Report Design Analysis 命令可允许您生成拥塞报告，以显示设备的拥塞区域和这些区域中存在的设计块的名称。报告中的拥塞表格会显示布局器和布线器算法展示的拥塞区域。下图显示了一个示例。



Window	Direction	Congestion Level	Congestion	Cell Names Top Cell 1	Top Cell 2	Top Cell 3	Combined LUTs	LUT6	LUT5	Flop	MUXF	RAMB	URAM	DSP
Window 1	North	5	108	u_dl_pro_cell0 (71%)	u_dl_pro_cell2 (12%)	top (9%)	6%	8%	4%	51%	1%	50%	100%	98%
Window 2	East	3	106	top (70%)	u_dl_pro_cell2 (12%)	u_dl_pro_top (8%)	28%	10%	7%	42%	0%	50%	NA	NA
Window 3	South	3	111	top (61%)	u_dl_pro_cell1 (37%)		0%	32%	10%	79%	0%	NA	NA	100%
Window 4	West	2	142	top (96%)			18%	9%	4%	40%	0%	NA	NA	NA

图 5-11：拥塞表格

“Placed Maximum”、 “Initial Estimated Router Congestion”及“Router Maximum Congestion”报告可提供有关北部、南部、东部和西部方向最拥挤区域的信息。当您在表中选择窗口时，相应的拥塞区域将在“Device”窗口中高亮显示。有关拥塞和 QoR 影响的信息，请参阅表 5-4。

该表格显示的是设计流程不同阶段上的拥塞：

- “Placed Maximum”：显示基于单元位置和布线模式的拥塞。
- “Initial Estimated Router Congestion”：显示快速布线器迭代后的拥塞。这是分析拥塞最有用的阶段，因为它能准确地描绘布局引起的拥塞问题。
- “Router Maximum”：显示广泛使用布线器降低拥塞后的拥塞情况。

Congestion Table 中的“Congestion”百分比显示拥塞窗口中的布线利用率。列出位于拥塞窗口中顶部的三个层级单元，并且能够选择和交叉探测到“Device”窗口或“Schematic”窗口。还显示出了拥塞窗口中的小区利用率。

通过识别拥塞区域中存在的层级单元，您能够使用本指南稍后讨论的拥塞减轻技术来尝试减少总体设计拥塞。

如需了解更多生成和分析“Report Design Analysis Congestion”报表的信息，请参阅 《Vivado Design Suite 用户指南：设计分析和收敛技术》(UG906) [参照 21] 中的 链接。

报告设计分析复杂性报告

对于顶层设计和/或层级单元，Complexity Report 会显示每个叶节点单元类型的 Rent Exponent、Average Fanout 和分配。 Rent 指数是指在使用 min-cut 算法以递归形式分割设计时，网表分区的端口数量和单元数量之间的关系。它的计算方法与在全局布局期间由布局器所用的算法类似。因此，特别是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，它才能提供布局器所见难度的良好指示。

具有较高 Rent 指数的设计对应的设计 （其中具有高度连接逻辑的组）也与其他组具有极强的连接与。这通常可转换为更高的全局布线资源利用率和增加的布线复杂性。根据未布局和未布线的网表对本报告中提供的 Rent 指数进行计算。在布局之后，相同设计的 Rent 指数可以不同，因为它基于物理分区而不是逻辑分区。

“Report Design Analysis”在执行以下任一操作时以“Complexity Mode”运行：

- 在“Report Design Analysis”对话框的“Options”标签卡中检查“Complexity”选项。
- 使用 `report_design_analysis` 选项执行 `-complexity` Tcl 命令。

下图显示了 Complexity Report。

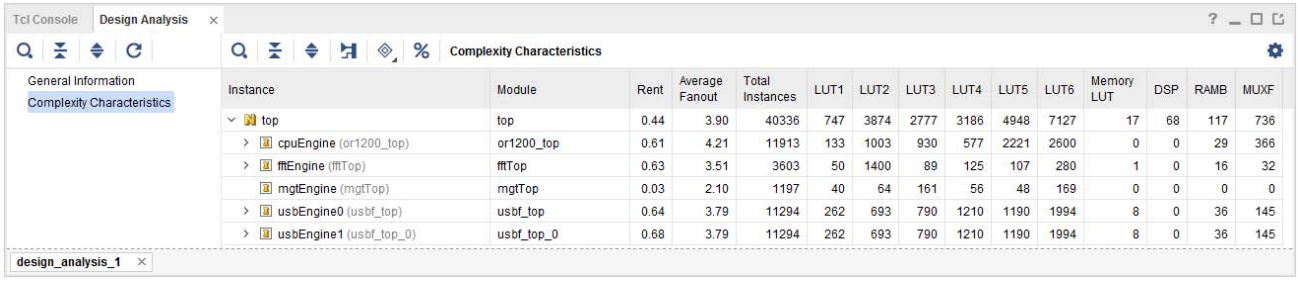


图 5-12：复杂性报告

下表显示了 Rent Exponent 的典型范围。

表 5-5：Rent 指数范围

范围	含义
0.0 到 0.65	这个范围低至正常。
0.65 到 0.85	此范围很广，特别是当实例总数超过 15,000 时。
0.85 以上	这个范围非常广，表明如果实例数量也非常高，设计可能在实现过程中失败。

下表显示了 Average Fanout 的典型范围。

表 5-6：平均扇出范围

范围	含义
低于 4	这个范围是正常的。
4 到 5	这个范围很广，表明布局设计没有拥塞可能很困难。 **注記** ：当采用 SSI 技术器件时，如果实例的总数超过 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	这个范围非常广，表明设计可能在实施过程中失败。

对于具有较高重要性的较大模块，您必须处理高 Rent 指数和 Average Fanouts。较小的模块，特别是低于 15,000 个总体实例，可以具有高 Rent 指数和高 Average Fanout，并且仍然易于成功布局和布线。因此，必须查看“Total Instances”列以及 Rent 指数和 Average Fanout。

提示： 即使一些较低层模块也具有高 Rent 指数和高 Average Fanouts，因而顶层模块不一定具有高复杂性指标。使用 `-hierarchical_depth` 选项可优化分析以包括较低级别的模块。

如需了解更多有生成和分析“Report Design Analysis Complexity”报表的信息，请参阅 《Vivado Design Suite 用户指南：设计分析和收敛技术》(UG906) [参照 21]中的链接。

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，赛灵思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，时钟偏差还会对实现工具增加过多的压力，以便在设备利用率非常高的时候满足建立和保持要求。

对于时钟内时序路径，时钟偏差通常小于 300 ps，而对于平衡同步时钟之间的时序路径，时钟偏差小于 500 ps。当跨越 I/O 列或 SLR 边界时，时钟偏差表现出更多变化，这反映在时序裕量中并且由实现工具进行优化。对于不平衡时钟树之间或没有公共节点的时序路径，时钟偏差可以是几纳秒，这使得时序收敛几乎不可能。

降低时钟偏差：

1. 如第 3 章中的“定义时钟组和 CDC 约束”中所述，需要查看所有时钟关系以确保只对同步时钟路径执行时序和优化。
2. 以下各章节内容包含：如何查看时钟树拓扑结构和时序路径的位置是否受到比预期更高的时钟偏差影响。
3. 以下各章节所述的是，如何识别可能的时钟偏差减少技术。

使用内部时钟时序路径

具有相同源和目的地时钟 （由相同时钟缓冲器驱动）的时序路径通常表现出非常低的偏差。如下图所示，因为公共节点位于专用时钟网络上，因而靠近叶时钟引脚。

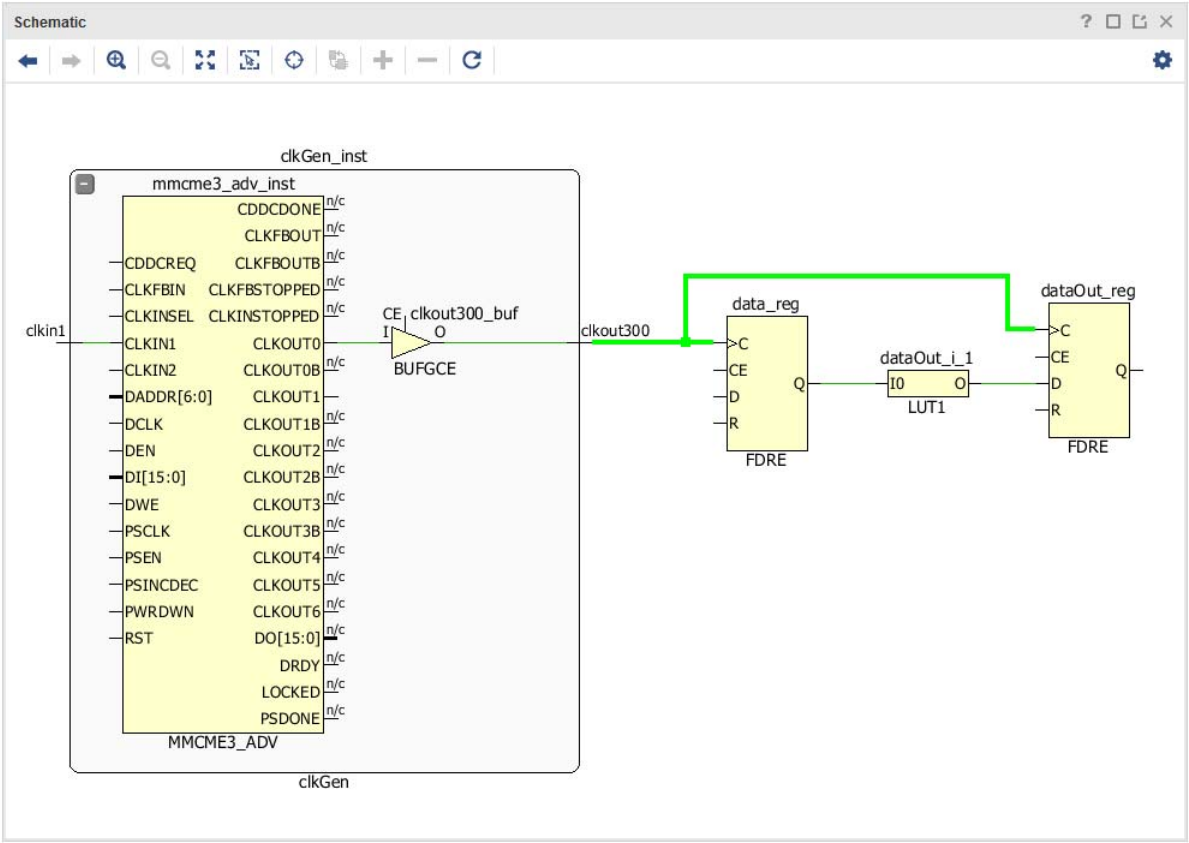


图 5-13：具有位于绿色网络上公共节点的典型同步时钟拓扑结构

因为公共节点仅存在于设计的物理数据库而非在逻辑视图中，所以当分析时序报告中的时钟路径时，不单独提供公共节点之前和之后的延迟。因此，当 Routing Resources 打开但不在“Schematic”窗口中时，您可以在 Vivado IDE 的“Device”窗口中看到公共节点。时序报告仅可为偏差计算的摘要提供源时钟延迟、目标时钟延迟和从时钟保守消除 (CPR) 到公共节点的信用。

限制同步跨时钟域路径

由单独的时钟缓冲器驱动的同步时钟之间的时序路径表现出较高的偏差，因为公共节点位于时钟缓冲器之前。也就是说，公共节点离叶片时钟引脚越远，导致时序分析中的保守度越高。由于源和目的地时钟路径之间存在延迟差，因而时钟偏差在不平衡时钟树之间的时序路径甚至更糟。虽然正偏差有助于满足建立时间要求，但它会伤害保持时间收敛，反之亦然。

在下图中，三个时钟具有几个内部和时序域间的时钟路径。由 MMCM 驱动的两个时钟的公共节点位于 MMCM （红色标识）的输出端。 MMCM 输入时钟和 MMCM 输出时钟之间的路径的公共节点位于 MMCM （蓝色标识）之前的网络上。对于 MMCM 输入时钟和 MMCM 输出时钟之间的路径，时钟偏差可能因 clkin_buf BUFGCE 位置和 MMCM 补偿模式的不同而变得特别高。

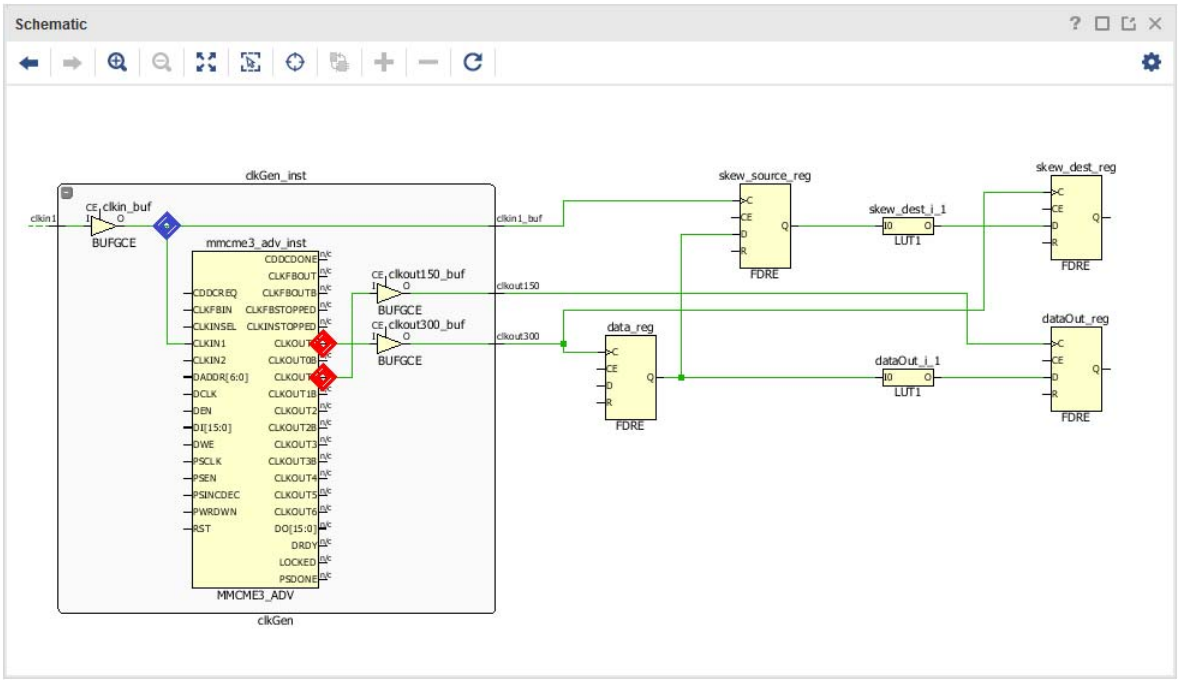


图 5-14：在 MMCM 的输入和输出上具有公共节点的同步 CDC 路径

即使在时钟偏差可接受的情况下，赛灵思也建议限制同步跨时钟域路径的数量。此外，当偏差异常高且无法降低时，赛灵思建议通过实现异步跨时钟域电路和添加时序例外来将这些路径作为异步处理。

在异步时钟之间添加时序例外

必须将时序路径 （其中源和目标时钟源自不同主时钟或没有公共节点）作为异步时钟处理。在这种情况下，偏差可能极其大，从而使得无法关闭时序。

您必须查看异步时钟之间的所有时序路径，以确保满足以下条件：

- 正确的异步跨时钟域电路 (`report_cdc`)
- 忽略时序分析 (`set_clock_groupsset_false_path`) 或忽略偏差的时序例外定义 (`set_max_delay -datapath_only`)

您可以使用 Clock Interaction Report (`report_clock_interaction`) 来帮助识别异步并缺少正确时序例外的时钟。如需了解更多有关 CDC 路径约束的信息，请参阅第 3 章中的“定义时钟组和 CDC 约束”。

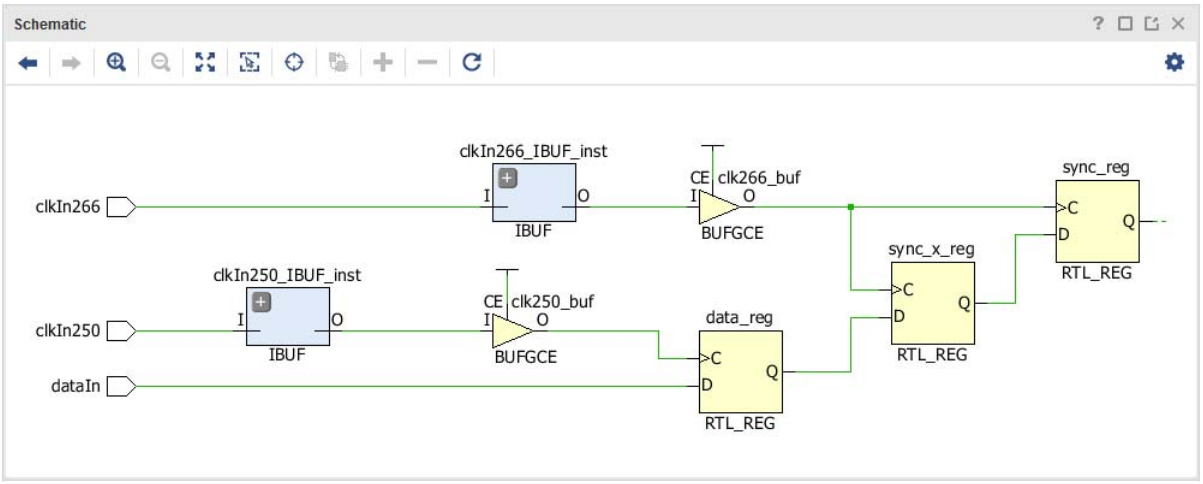


图 5-15：异步 CDC 路径与正确的 CDC 电路和无公共节点

应用可减少时钟偏差的常见技术

由于 7 系列和 UltraScale 器件时钟架构不同，因此需要查阅第 3 章中的“编码指南”以了解每个架构的最佳实践，并验证您的设计是否符合规范。

提示： 考虑到 UltraScale 器件时钟架构的灵活性， `report_methodology` 命令包含了有助于您创建最佳时钟拓扑结构的检查。

以下技术可以处理最常见的情况：

- 如下图所示，通过消除不必要的缓冲器或并行连接它们可以避开级联时钟缓冲器之间的时序路径。

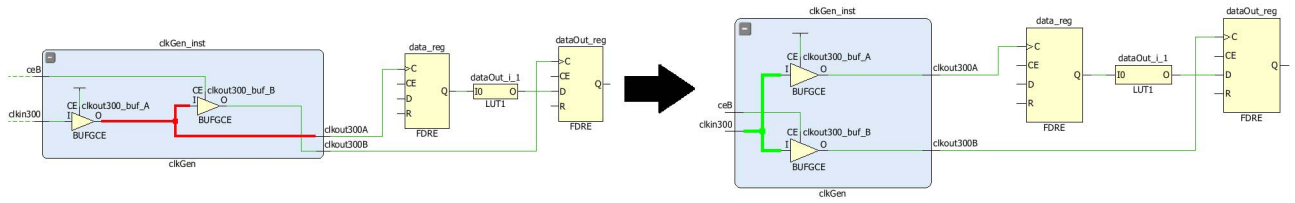


图 5-16：带重新并行连接的级联 BUFG 的同步时钟拓扑结构

- 如下图所示，将并行时钟缓冲器组合进单个时钟缓冲器，并将任何时钟缓冲器时钟使能逻辑连接到相应的顺序单元使能引脚。如果一些时钟由缓冲器内置分频器进行分频，则实现具有时钟使能逻辑的等效除法，并根据需要应用多周期路径时序例外。当下行逻辑使用上升和下降时钟边沿时或当功耗是一个重要因素时，此技术可能不适用。

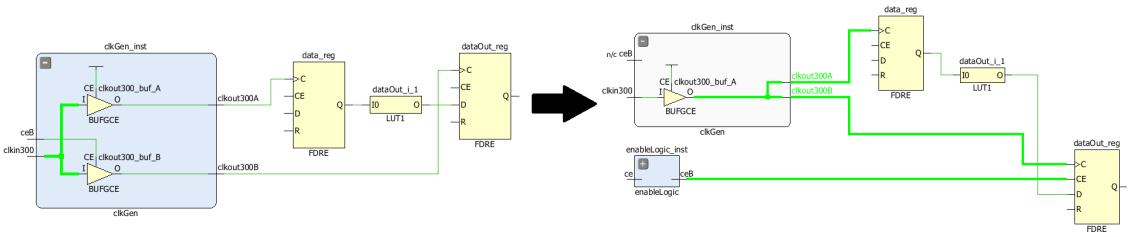


图 5-17：将并行时钟缓冲器重组为单缓冲区的同步时钟拓扑结构

- 在时钟路径中移除 LUT 或任何组合逻辑，因为它们可使布局期间的时钟延迟和时钟偏差不可预测，从而导致较低的结果质量。此外，还可采用通用互连资源 （与全局时钟资源相比对噪声较为敏感）对时钟路径的一部分进行布线。组合逻辑通常来自欠佳的时钟门控转换，且通常能够移至时钟使能逻辑，无论连接到时钟缓冲器还是连续单元均可。

  在下图中，第一个 BUFG (clk1_buf) 在 LUT3 中用于创建门控时钟条件。

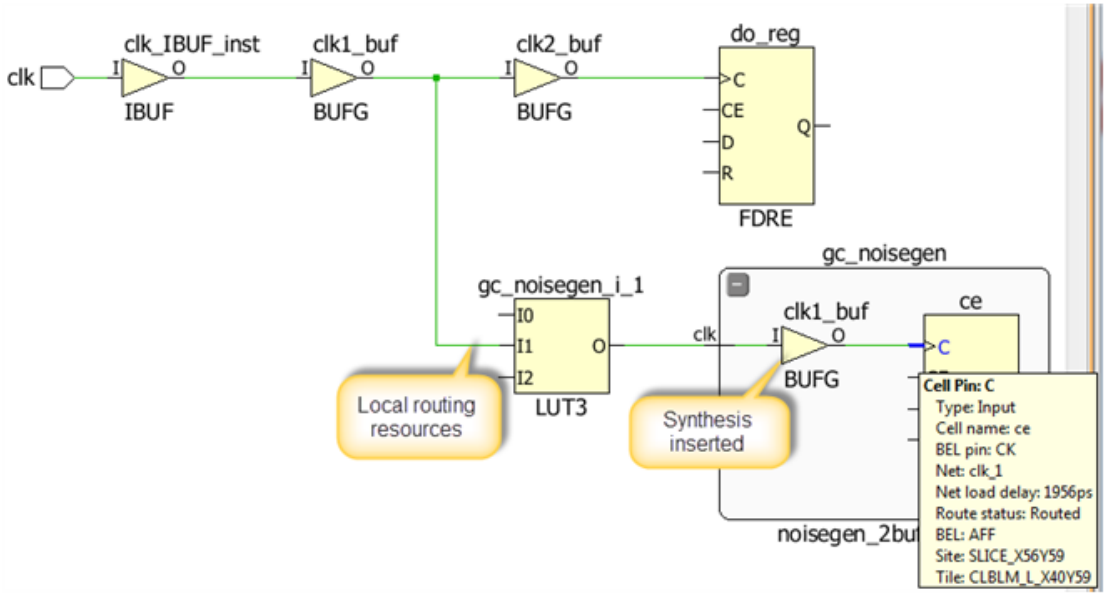


图 5-18：时钟网络局部布线引起的偏差

应用技巧提升 7 系列器件的偏差

虽然 7 系列和 UltraScale 架构在时钟架构方面不同，但是一些时钟考虑因素仍适用于两个系列：

- 不要在已经量产的 7 系列设计中使用 CLOCK_DEDICATED_ROUTE=FALSE 约束。使用 CLOCK_DEDICATED_ROUTE=FALSE 仅作为时钟故障的临时解决方法，仅用于生成实现的设计，以便查看用于调试的时钟拓扑结构。采用结构互连布线的时钟路径可能具有高时钟偏差，并受开关噪声的影响，从而导致性能较差或无功能设计。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。

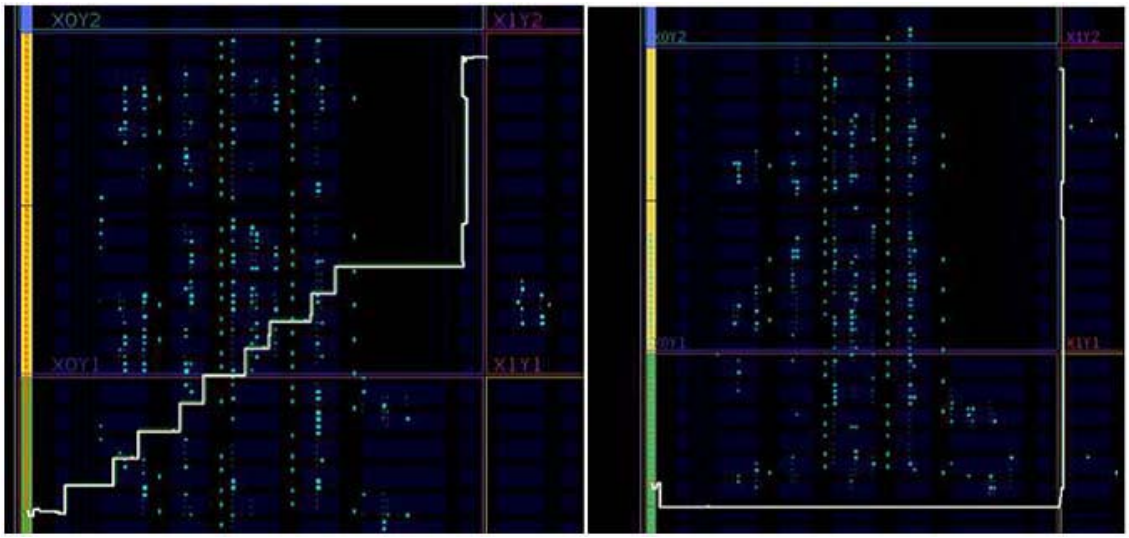


图 5-19：结构时钟布线与专用时钟布线的比较

- 随着每个区域中的时钟树分支之间的偏差将非常高，所以不允许区域时钟缓冲器 (BUFR/BUFIO/BUFH) 在几个时钟域中驱动逻辑。删除不适当的 LOC 或 Pblock 约束以解决此问题。

提升 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 单元具有分频输入时钟的能力。下图显示了如何保存 MMCM 资源，并为两个源自 GTHE3_CHANNEL 单元的时钟实现平衡的时钟树。

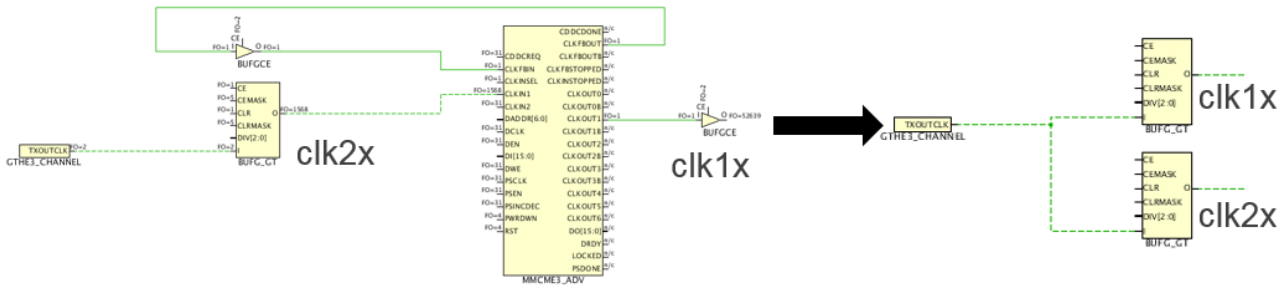


图 5-20：使用 UltraScale BUFG_GTs 实现平衡的时钟树

- 在临界同步时钟的驱动器网络上使用 CLOCK_DELAY_GROUP 以便在布局和布线期间强制执行 CLOCK_ROOT 及布线匹配。时钟的缓冲器必须由相同的单元驱动以实现该约束。详情，请参阅第 3 章中的“同步 CDC”。
- 如果时序路径难以满足时序，并且偏差大于预期，则时序路径跨过 SLR 或 I/O 列可行。如果是这种情况，则可将诸如 Pblock 的物理约束用于迫使源和目的地进入单个 SLR 或防止 I/O 列的交叉。
- 当使用高速同步跨时钟域时序路径时，将诸如 MMCM/PLL 等时钟修改块定位到时钟负载的中心能有助于满足时序。时钟网络上减少的延迟将导致在跨时钟域路径上较少的时序保守。
- 验证具有 CLOCK_DEDICATED_ROUTE = FALSE 约束的时钟网络是否使用全局时钟资源进行布线。使用 ANY_CMT_COLUMN 而不是 FALSE 来确保具有布线放弃器的时钟网络仅使用专用时钟资源进行布线。如需获取有关 CLOCK_DEDICATED_ROUTE 支持的更多信息，请参阅第 3 章中的“时钟约束”。如果采用结构互连对时钟网络与进行布线，请识别解决此情况所需的设计更改或时钟布局约束，并使实现工具使用全局时钟资源。采用结构互连布线的时钟路径可能具有非常高的时钟偏差，或受开关噪声的影响，这会导致性能较差或无功能设计。

减少时钟不确定性

时钟不确定性是一个相对于理想时钟的不确定性量。不确定性可能来自用户指定的外部时钟不确定性 (`set_clock_uncertainty`)、系统抖动或占空比失真。诸如 MMCM 和 PLL 等 Clock Modifying Blocks 还能以 Discrete Jitter 的形式和 Phase Error （如果使用多个相关时钟）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，并且可生成用于比较不同时钟拓扑结构的各种 MMCM 时钟配置。为了实现目标架构的最佳结果，赛灵思建议使用 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 建立来减少时钟不确定性

当配置用于频率综合的 MMCM 时，目标频率可以具有几个可能的 M （乘法器）和 D （除法器）值以实现相同的目标。导致最高 VCO 频率 （不超过器件的最大 VCO 频率）的 M 和 D 值将最小化时钟不确定性。

下面的 MMCM 频率综合实例使用 62.5 MHz 的输入时钟来生成 40 MHz 左右的输出时钟。这里有两种解决方案，但具有更高 VCO 频率的 MMCM_2 由于减少了抖动和相位误差而生成较少的时钟不确定性。

表 5-7：MMCM 频率综合实例

	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	16	22.875
CLKOUT0_DIVIDE_F	25	35.750
VCO 频率	1000.000 MHz	1429.688
抖动 (ps)	167.542	128.632
相位误差 (ps)	384.432	123.641

提示： 当使用 IP 目录中的 Clocking Wizard 时，确保将 Jitte Optimization Setting 设定为 Minimum Output Jitter，这样将提供更高的 VCO 频率。

使用 BUFGCE_DIV 减少时钟不确定性

在 UltraScale 器件中，通过消除 MMCM Phase Error， BUFGCE_DIV 单元可用于减少同步跨时钟域的时钟不确定性。例如，需要考虑到在 300 MHz 和 150 MHz 时钟域之间存在路径，其中两个时钟由相同的 MMCM 生成。

在这种情况下，时钟不确定性包括建立与保持分析的 120 ps 相位错误。与使用 MMCM 生成 150 MHz 时钟不同，可将 BUFGCE_DIV 连接到 300 MHz MMCM 输出，并将时钟除以 2。如下图所示，为了获得最佳结果， 300 MHz 时钟还需要使用 BUFGCE_DIV，而不是有效除法，才能准确匹配 150 MHz 时钟延迟。

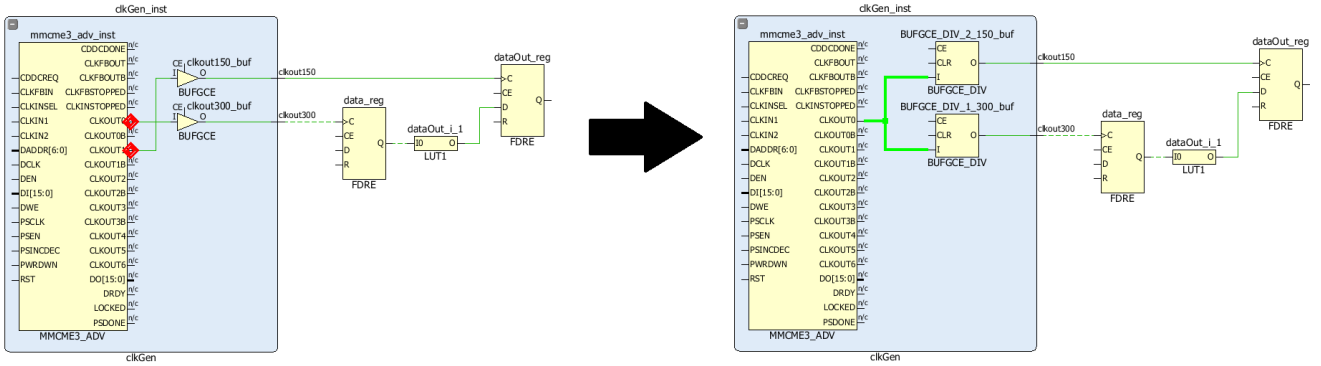


图 5-21：改进适用于 UltraScale 同步 CDC 时序路径的时钟拓扑结构

使用新拓扑结构：

- 对于建立分析，时钟不确定性不但不包括 MMCM 相位误差，而且还减少了 120 ps。
- 对于保持分析，没有更多的时钟不确定性 （仅用于相同的边缘保持分析）。
- 公共节点移动更靠近缓冲区，这节省了一些时钟保守。

通过对两个时钟网络应用 CLOCK_DELAY_GROUP 约束，时钟路径将具有匹配的布线。如需了解更多信息，请参阅第 3 章中的“同步 CDC”。

对于 UltraScale 同步 CDC 时序路径，下表对建立和保持分析的时钟不确定性进行了比较。

表 5-8：对于 UltraScale 同步 CDC 时序路径，对建立和保持分析的时钟不确定性进行比较

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
	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
保持分析	**MMCM 生成的 150 MHz 时钟**		**BUFGCE_DIV 150 MHz 时钟**
	总系统抖动 (TSJ)	0.071 ns	0.000 ns
	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

应用通用时序收敛技术

以下技术可帮助在极富挑战性的设计上关闭设计。在尝试使用这些技术之前，您需要确保设计受到适当约束，并确定影响顶部违规路径的主要问题。

提示： 您可以使用 `report_qor_suggestions` Tcl 命令来自动确定并应用这些技术。如需了解更多信息，请参阅《Vivado Design Suite 用户指南：设计分析和收敛技术》(UG906) [参照 21] 中的链接。

用块级综合策略提升网表

虽然大部分设计使用默省的 Vivado 综合设置就能满足时序要求，但是较大较复杂的设计通常要求对不同层级使用混合综合策略来实现时序收敛。

减少 MUXF 映射以降低拥塞

使用 MUXF 单元能为拥有大量逻辑层或严格时钟要求的关键路径提供帮助。但是在网表连接较复杂时，这种方法会制约布局的灵活性，导致更严重的潜在布线拥塞和时序劣化问题。为减轻拥塞模块中的 MUXF 占用，您可以结合 BLOCK_SYNTH.MUXF_MAPPING 约束来使用块级综合策略。如需了解更多使用此策略的信息，请参阅第 4 章中的“块级综合策略”。

正如在“识别拥塞”中介绍的，在布局或布线完成后，您可以通过审核日志文件或设计分析报告（report_design_analysis -congestion）中的“布线器初始估计拥塞表”来判断时序收敛是否受布线拥塞的影响。

在下图中，设计分析报告显示在南向上， 7% 的器件受 5 级短拥塞 （32x32 CLB）影响，而在对应的拥塞区域中，MUXF 的占用率为 26%。

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%

图 5-22：report_design_analysis 拥塞表格中的南向短拥塞

在 Vivado IDE 中，您可以选择设计分析拥塞报告表格中的一列，在器件窗口中高亮显示对应的拥塞区域。下图显示的是与较高 MUXF 密度区域叠加的拥塞。在 Vivado IDE Tcl 控制台上使用下列命令即可用洋红高亮显示 MUXF 单元：

```
highlight_objects -color magenta [get_cells -hier -filter REF_NAME=~MUXF*]
```

MUXF* 含 MUXF7/MUXF8/MUXF9，这些是位于 CLB 内的专用复用器资源。这些资源在布局过程中按多达 8 个的 LUT 分组，强制高 CLB 输入占用率，提高布线要求，限制布局的灵活性。每个 CLB 的估计拥塞可使用 Vivado IDE 指标显示。

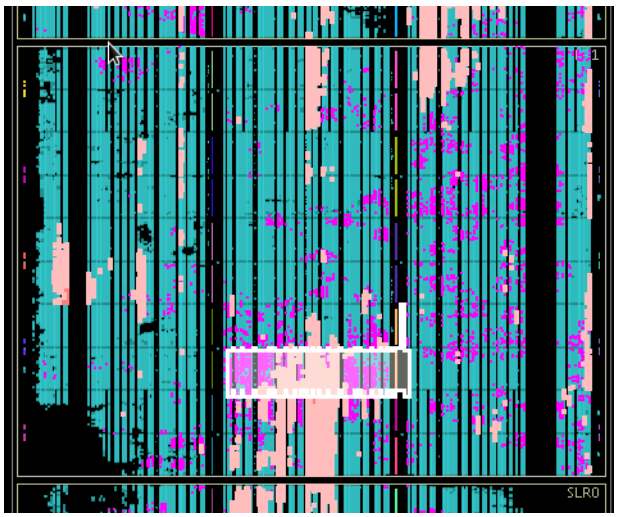


图 5-23：Vivado IDE 器件窗口中高亮显示的 MUXF 拥塞

在这种情况下，赛灵思建议通过将它们对应的功能映射到有更高布局和布线灵活性的 IUT，以减少 MUXF* 的数量。您可以在 XDC 综合约束中使用下列命令来修改网表：

```
set_property BLOCK_SYNTH.MUXF_MAPPING 0 [get_cells inst_name4]
```

在重新运行综合、布局和布线后，设计分析报告中更新后的拥塞表格现在显示南向短拥塞降低 （4 级），时序结果质量普遍得到提升。

```
+-----------+--------+-----------------+------------------+---------------------------------+-----------------+---------------+------+...+------+
| 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% |
+-----------+--------+-----------------+------------------+---------------------------------+-----------------+---------------+------+...+------+
```

图 5-24：在模块上减少 MUXF 使用后的初始布线器拥塞表格

提升逻辑层次

在整个设计周期中，您必须为目标赛灵思 FPGA 系列和器件速度等级验证逻辑层次分布与时钟频率目标相符合。尽管有限数量的路径加上大数量的逻辑层次并不总会带来时序收敛难题，但您可以使用 Vivado 综合时钟重生选项来优化最长路径，达到提升时序 QoR 的目的。

在全局使用时钟重生选项一般会造成高密度的运行时间。因此，赛灵思建议您为综合后或优化布局后存在大量逻辑层次的路径指定具体的违规层级。如果最长路径的扇入或扇出部分中的路径逻辑层次较少且被约束在中小层级模块内，您可以使用 BLOCK_SYNTH.RETIMING 块级综合策略。

下图显示的是用 600 MHz 时钟约束、有 5 个 LUT 的关键路径。REG2 目的地触发器可包含在 REG2 之上的一个层级中的单 LUT 驱动时序路径。

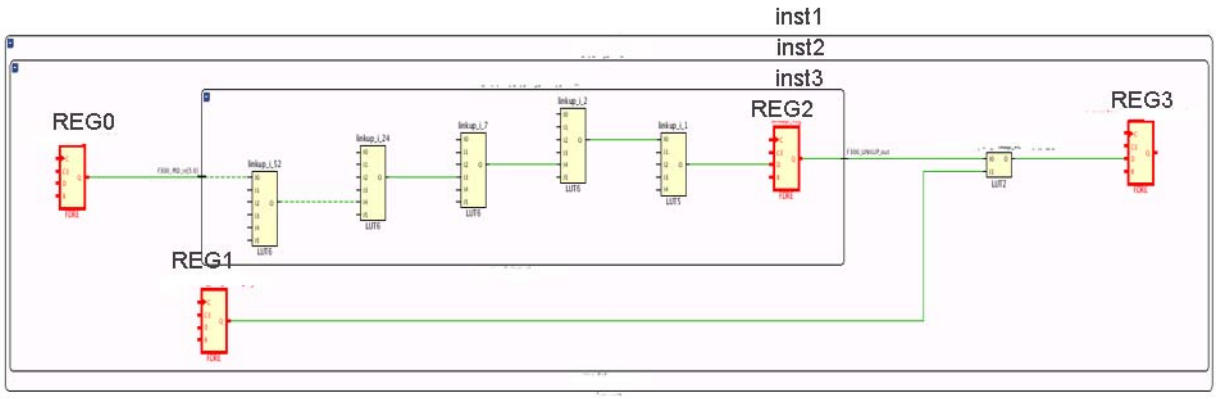


图 5-25：显示有 5 个逻辑层次关键路径的原理图

除了在 Vivado IDE 中使用原理图窗口，您还可以使用 `report_design_analysis -logic_level_distribution` 命令来审核特定路径的逻辑层次分布。这样便于您确定需要重新均衡路径的数量，以提升时序 QoR。

下图所示的是有 5 个逻辑层次的 58 个路径，这些路径位于受 600 MHz 时钟约束的 inst1/inst2 层级内，同时下图也展示了另外 32 个只有 1 个逻辑层次的路径。

```
instance inst1/inst2

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

-----------+-------------+---+----+----+---+---+----+---+---+---+---+----+-------+-------+-------+-------+-----+
oint Clock | Requirement | 0 | 1  | 2  | 3 | 4 | 5  | 6 | 7 | 8 | 9 | 10 | 11-15 | 16-20 | 21-25 | 26-30 | 31+ |
-----------+-------------+---+----+----+---+---+----+---+---+---+---+----+-------+-------+-------+-------+-----+
lk_600     | 1.667ns     | 0 | 32 | 10 | 0 | 0 | 58 | 0 | 0 | 0 | 0 |  0 |     0 |     0 |     0 |     0 |   0 |
-----------+-------------+---+----+----+---+---+----+---+---+---+---+----+-------+-------+-------+-------+-----+
```

图 5-26：使用默省综合优化得到的逻辑层次分布

通过将低逻辑层次路径中的寄存器移动到高逻辑层次路径中， Vivado 综合能重新均衡逻辑层次。在本例中，您可以将下列约束添加到综合 XDC 文件，在 inst1/inst2 层级上进行时序重排：

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

在使用相同的全局设置和更新后的 XDC 文件重新运行综合后，您可以在 inst1/inst2 时序路径上定期运行时序分析，或是重新运行 `report_design_analysis` 命令，验证最长路径的逻辑层次数量是否下降，如下图所示。关键路径现在为 REG0 > 3 个 LUT > REG2 （后向时序重排），同时 REG2 到 REG4 的路径有 3 个逻辑层次。

```
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 |
+-----------------+-------------+---+---+---+-----+---+---+---+---+---+---+----+-------+-------+-------+-------+-----+
```

图 5-27：为综合优化启用时序重排的逻辑层次分布

减少控制集

复位或时钟使能等控制信号一直没有得到足够的重视。很多设计人员在进行 HDL 编码时都以“if reset”说明作为开始，并没有考虑是否需要复位。所有寄存器都支持复位和时钟使能，因此其使用会严重影响最终实现方案的性能、利用率和功耗。

第一个要考虑的因素是控制集的数量。控制集是指时序单元使用的一组时钟、使能和置位/复位信号。例如，由相同时钟连接的两个单元，如果只有一个单元有复位信号，或者只有一个单元有时钟使能信号，那么这两个单元具有不同的控制集。恒定或未使用的使能和置位/复位寄存器引脚也有助于形成控制集。

第二个要考虑的因素是目标架构。能封装到一起的控制集数量取决于架构：

- 在 7 系列器件 slice 中：8 个寄存器共享 1 个时钟， 1个建立/复位和 1 个时钟使能。每个 slice 只能使用一个控制集。
- 在 UltraScale 器件一半的 CLB 中：8 个寄存器共享相同的时钟， 1 个可选的置/复位和 2 个时钟使能。虽然常量置位/复位不被布线并且可以被忽略，但是恒定使能信号可被视为动态使能信号。在理想条件下，最多可将 4 个控制集封装到半个 CLB 内。

由于逻辑扩展 （更长的净延迟）和更高的互连资源利用率， CLB 封装限制不仅可以不利地影响利用率，而且可以负面影响布局 QoR 和功耗。对于具有很多很低扇出控制信号 （例如到单个寄存器的时钟使能信号）的设计来说需要考虑这个问题。

下表提供了关于采用赛灵思 7 系列 FPGA 的设计中数量可接受的控制集的指南。控制集信息可用：

- 在展示位置后的使用情况报告的文本版本中：`report_utilization`

- 在控制集报告中 （仅限文本）：`report_control_sets -verbose`

表 5-9：7 系列 FPGA 的控制集指南

条件	一般可接受	需要分析	建议调整设计
唯一控制集数量[a]	不足 slice 总数的 7.5%[b]	超过 slice[a, b] 总数的 15%	超过 slice[b] 总数的 25% 减少控制集的数量能够提高利用率和性能。
控制集限制的寄存器损失数量 [a]	slice 利用率 < 75%，寄存器损失 < 4%[c]	slice 利用率 > 85%，寄存器损失 > 2%[c]	slice 利用率 > 90%，寄存器损失 > 1%[c]

a. 运行 `report_utilization` 和 `report_control_sets -verbose`。
b. 在器件产品列表可找到 slice。例如：XC7VX690T 包含 108,300 个 slice。
可接受：108,300 个 slice x 7.5% = 8122 个控制集
需要分析：108,300 个 slice x 15% = 16,245 个控制集
c. 可用寄存器总数。

建议： UltraScale 器件能更加灵活地映射与使用控制集。因此，不用非常严格地遵循控制集使用指南。然而，赛灵思依然建议遵循 7 系列指南以实现最佳的设计优化、布局以及可移植性。

提示： 唯一控制集的数量在设计的小部分中可能是问题，可能导致相应设备区域中的更长的网络延迟或拥塞。识别唯一控制集的高本地密度需要在“Vivado IDE Device”窗口中进行详细的布局分析，突出显示的控制信号以不同的颜色显示。

如果控制集数量较多，请使用以下策略减少其数量：

- 删除在 HDL 源或约束文件中的控制信号上设置的 MAX_FANOUT 属性。复制控制信号将大大增加唯一控制集的数量。赛灵思建议使用 `place_design -fanout_opt` 在布局器中复制。这样可以防止不必要的重复和等效控制集彼此交错，进而造成布线拥塞问题。
- 增加 Vivado 综合 （或其他 FPGA 综合工具）的控制集阈值。例如：

```
synth_design -control_set_opt_threshold 16
```

- 在综合后使用 `opt_design -control_set_merge` 合并等效控制集。
- 避免低扇出异步设置/复位 （预置/清零），因为它们只能连接到专用异步引脚，并且不能通过综合移动到数据路径。因此，综合控制集阈值选项不适用于异步设置/复位。
- 避免对不同的顺序单元同时使用高电平和低电平有效的控制信号。
- 仅在必要时使用时钟使能和布局/复位。通常，数据路径包含许多自动刷新未初始化值的寄存器，并且在第一级和最后一级仅需要布局/复位或使能信号。

参阅第 3 章中的“控制信号与控制集”获取关于控制信号的附加综合属性和建议。

优化高扇出网

高扇出网络常常导致设计实现问题。随着每个 FPGA 系列的芯片尺寸增加，扇出问题也增加。经常难以满足具有数千个端点的网络上的时序，特别是如果在路径上存在附加逻辑，或者如果它们由非顺序单元 （例如 LUT 或分布式 RAM）驱动的话。

使用寄存器复制

大多数综合工具提供扇出阈值限制来强制高扇出驱动程序复制。 Vivado 综合选项是 `synth_design -fanout_limit 5000`。调整全局阈值不能让您控制哪些寄存器可以复制。更好的方法是对特定寄存器或层级级数施加属性，确定哪些寄存器能够或者不能够复制。例如，如果是 LUT1 而不是寄存器用于复制，则它指示属性或约束正在阻止优化，例如分层单元上的 DONT_TOUCH 属性或不同层次结构中的网段。

有时，设计人员使用全局扇出限制或特定网络上的 MAX_FANOUT 属性来处理 RTL 或综合中的高扇出网络。这并不总会产生最佳的布线资源使用情况，特别是如果 MAX_FANOUT 属性布局得太低的时候。此外，如果高扇出信号是寄存器控制信号并且被复制超过必要，则这可以导致更多数量的控制集合。

通常，减少扇出的更好方法是对高扇出信号使用平衡树。考虑基于设计层次手动复制寄存器，因为包括在层次结构中的单元通常布局在一起。例如，在下图所示的平衡复位树中，高扇出复位 FF RST2 在 RTL 中复制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。 DONT_TOUCH 属性阻止在实现流程中稍后的物理优化期间的进一步优化。

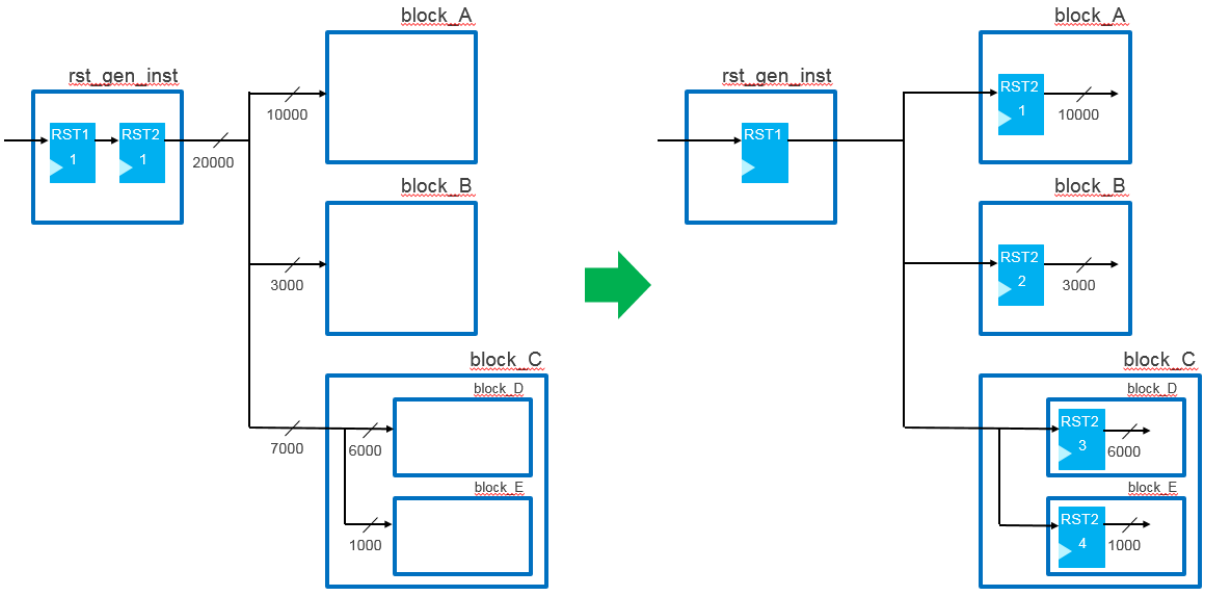


图 5-28：高扇出复位变换为平衡复位树

要按照前面的示例所述构建控制集树，可以使用带有以下选项之一的 opt_design Tcl 命令：

- -control_set_merge：该选项将逻辑等效控制信号的驱动器减少到单个驱动器。
- -merge_equivalent_drivers：该选项将逻辑等效控制信号的驱动器减少到单个驱动器。

这与扇出复制相反，并且会导致网络更适合于基于模块的复制。如下图所示，此合并也适用于多级复位树。

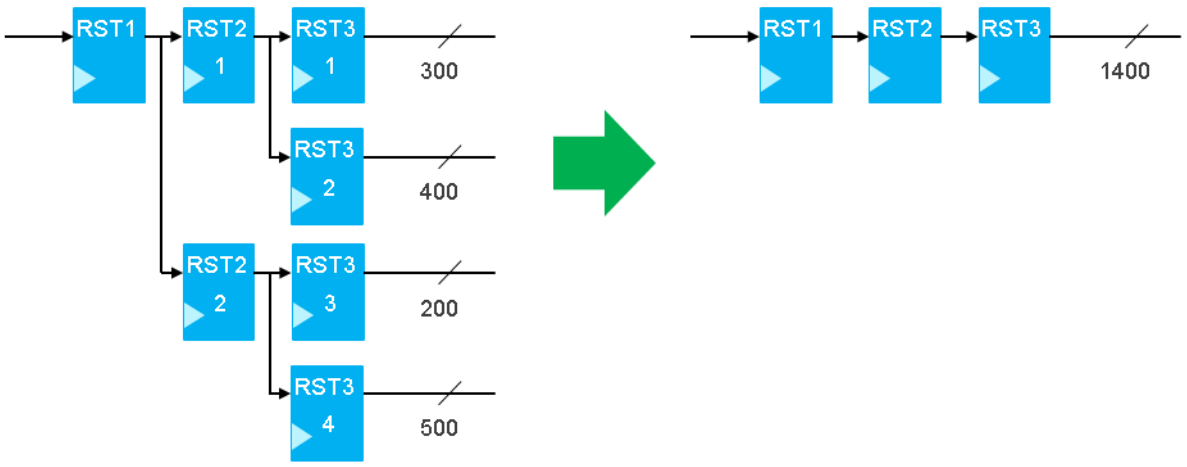


图 5-29：使用 opt_design -control_set_merge 控制集合并

- `-hier_fanout_limit <arg>`：该选项根据层次结构复制寄存器，其中 `<arg>` 表示根据逻辑层次结构的复制的扇出限制。对于由高扇出网驱动的每个分层实例，如果分层结构内的扇出大于指定的限制，则分层结构内的网由高扇出网的驱动器的副本驱动。复制的驱动程序布局在与原始驱动程序相同的层次结构中，复制不限于控制集合寄存器。下图显示了使用 `opt_design -hier_fanout_limit 1000` 在扇出为 60,000 的时钟使能网络上进行复制。因为每个模块 SR_1K 包含 1000 个负载，所以驱动程序被复制 59 次。

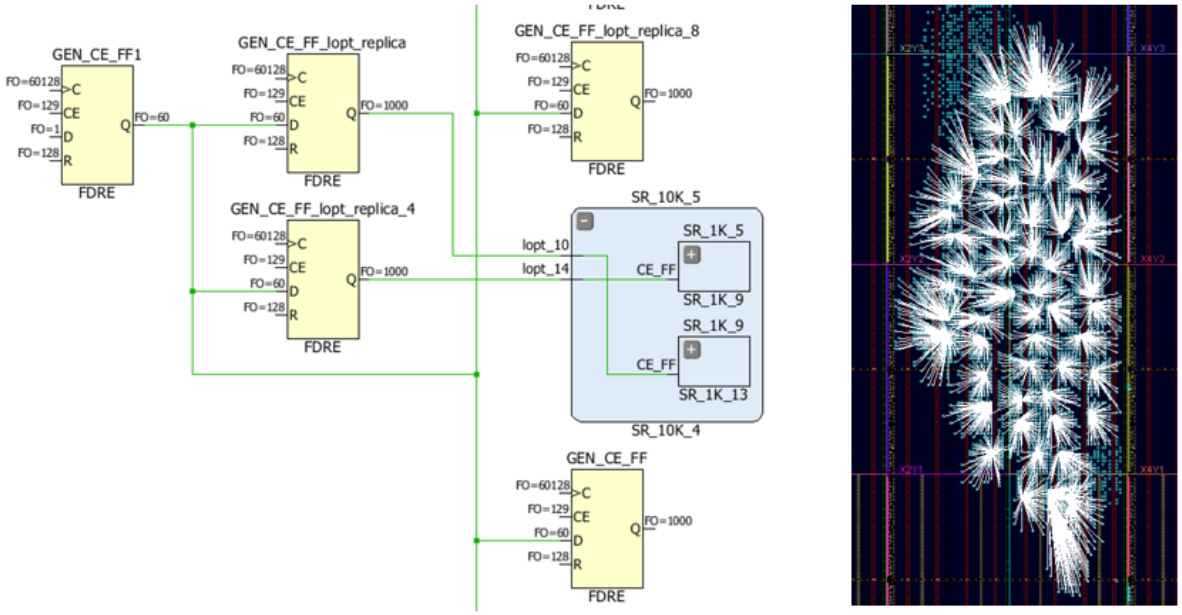


图 5-30：高风扇时钟使能网络上基于模块的复制

您还可以使用 -fanout_opt 选项在 place_design 中启用扇出优化。复制发生在布局器流程的初期阶段并基于布局信息。对驱动超过 1000 个负载的寄存器和驱动 DSP、块 RAM、 FIFO 和 URAM 的寄存器，可考虑复制。如果进行复制，可与负载位于一处。

对于 SSI 技术设备，可以为每个 SLR 复制高扇出驱动器，并可选地将其分配给 SLR 对齐的 Pblock 及其负载。该技术有助于减少 SLR 交叉延迟的影响，并且给予在每个 SLR 中独立地布局复制的高扇出网络的更多自由度。

将高扇出网络推广到全局布线

通过在驱动器和负载之间插入时钟缓冲器，可以将较低性能的高扇出网络移动到全局布线上。只有当已经使用有限数量的时钟缓冲器时，对于扇出大于 25000 的网络，此优化在 opt_design 中自动执行。在 RTL 文件或约束文件 (XDC) 中的网络上布局 CLOCK_BUFFER_TYPE 属性时，可以强制 synth_design 和 opt_design 插入时钟缓冲器。例如：

```
set_property CLOCK_BUFFER_TYPE BUFG [get_nets netName]
```

使用全局时钟确保了最佳布线，但代价是更高的网络延迟。为了获得最佳性能，时钟缓冲器必须直接驱动顺序负载，无需中间组合逻辑。在大多数情况下， opt_design 会将非顺序加载与时钟缓冲器并行重新连接。如果需要，您可以通过在时钟缓冲器输出网络上应用 DONT_TOUCH 来防止进行此优化。此外，如果高扇出网络是控制信号，您必须确定为什么某些负载不是专用时钟使能或布局/复位引脚。在 第 3 章中的“控制信号与控制集”中查看专用综合属性的使用，以控制本地时钟启用和布局/复位优化。

布局器还能在任何完成时钟布线后可用的全局布线路径上自动布线高扇出网络 （扇出 > 1000)。该优化发生在接近布局器流程的尾声，只在时序不发生劣化的情况下进行。您可以使用 -no_bufg_opt 选项禁用该功能。

物理优化

物理优化 (phys_opt_design) 基于裕量和布局信息自动复制高扇出网络驱动器，并通常显著改进时序。赛灵思建议您使用光纤寄存器 (FD*) 驱动高扇出网络，这在物理优化期间更容易复制和重新定位。

在某些情况下，默认的 phys_opt_design 命令不会复制所有关键的高扇出网络。使用不同的指令增加命令的努力：探索，进取或上进的扇出选项。此外，当高扇出网络在布线期间变得至关重要时，您可以添加 phys_opt_design 的迭代在特定网络上进行强制复制，然后再次尝试布线设计。例如：

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

解决拥塞

拥塞可以由各种因素引起，是一个复杂的问题，并非总是存在直观的解决方案。复杂性和拥塞具有相同的解决技术。检查复杂的模块是否布局在设备的拥塞区域。 `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顶层模块。存在各种技术来优化拥塞区域中的模块。

提示： 在尝试通过下面讨论的技巧解决拥塞之前，请确保您具有干净的约束。重叠 Pblock 可能导致拥塞，而且应避免。过多的保持时间故障或负保持裕量要求布线器绕行，这可能导致拥塞。

降低设备利用率

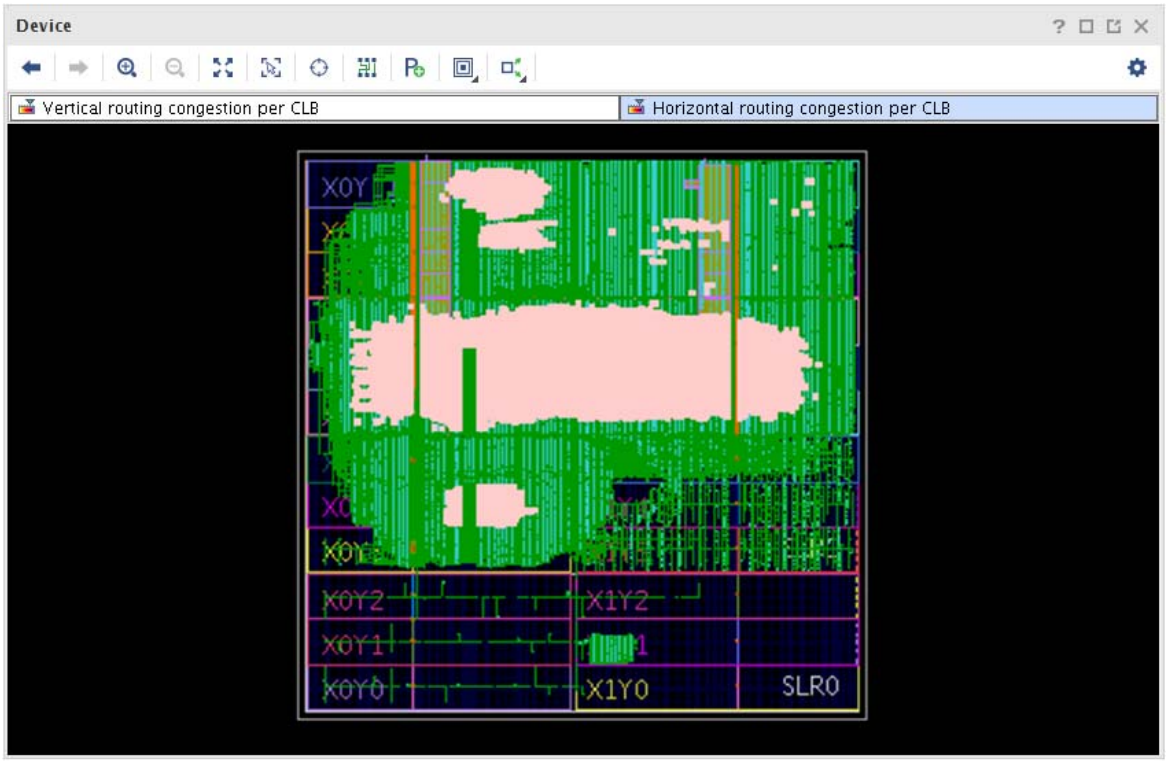
当多个结构资源利用率高 （平均 > 75%）时，如果网表复杂度也很高 （高顶级连接，高 rent 指数，高平均扇出），布局就变得更具挑战性。高性能设计还带来了额外的布局挑战。在这种情况下，重新访问设计功能并考虑删除非必要模块，直到只有一个或两个结构资源利用率百分出现比较高的情况。如果不可能减少逻辑，请查看本章中介绍的其他拥塞减轻技术。

提示： 在 `opt_design` 之后查看资源利用率，以便获得更准确的数字，一旦未使用的逻辑已修剪，而不是综合后。

平衡 SSI 设备的 SLR 利用率

当针对 SSI 技术设备时，分析每个 SLR 区域的利用率很重要。总体利用率可能较低，但是在一个 SLR 中的高利用率可能导致拥塞。

在下图中，设计的整体利用率较低。然而， SLR2 中的利用率高，并且逻辑比其他 SLR 中的逻辑需要更多的布线资源。该区域中的逻辑是使总线资源饱和的宽总线 MUX。



Type	Row	Col	Sites	Cells	Vert route cong (%)	Hori route cong (%)
CLEM	309	577	1	13	186.24	156.97
CLEL_R	309	574	1	8	120.52	142.57

图 5-31：利用率分析

要平衡使用率，请尝试以下操作：

- 使用不同的布局指令扩展设计。
- 使用布局规划约束，例如 Pblocks，以保持一些模块不受高度利用和拥塞的单反。

使用替代布局和布线指令

由于布局通常对整体设计性能有最大的影响，因此应用不同的布局器指令是应该尝试减少拥塞的第一种技术之一。考虑运行替代布局器指令而没有任何现有的 Pblock 约束，以便给予布局器更多的自由来根据需要传输逻辑。

存在几个布局器指令，其可以通过在整个设备中扩展逻辑来避免拥塞，从而避免拥塞区域。 SpreadLogic 布局器指令包含：

- AltSpreadLogic_high
- AltSpreadLogic_medium
- AltSpreadLogic_low
- SSI_SpreadLogic_high
- SSI_SpreadLogic_low

当在 SLR 交叉口上检测到拥塞时，请考虑使用：

- SSI_BalanceSLLs 布局指令，有助于跨 SLR 划分设计，同时尝试平衡 SLR 之间的 SLL
- 在跨 SLR 分区时， SSI_SpreadSLLs 布局器指令为更高的连接分配额外区域。

其他安置指令或实施策略也可能有助于缓解拥塞，也应该在上面提到的布局指令后尝试。

要比较不同布局器指令的拥塞，可以在 `place_design` 之后运行设计分析拥塞报告，也可以检查布线器日志文件中的初始估计拥塞。在 表 5-4中根据显示的拥塞级别范围查看结果。

布线对拥塞的影响比对布局器指令的影响小。但是，在某些情况下尝试不同的布线指令也是有用的。下列指令可确保布线器能更努力地覆盖更多的布线，减轻互联块中的拥塞：

- AlternateCLBRouting

注释： 在只有短拥塞或短拥塞和长拥塞同时存在时， AlternateCLBRouting 的布线指令效果最好。该指令只适用于 UltraScale 器件。

如需了解更多信息，请参阅 《Vivado Design Suite 用户指南：实现》(UG904) [参照 19] 中的链接。

关闭跨边界优化

禁止综合中的跨边界优化防止附加逻辑被拉入模块。这降低了模块的复杂性，但也可以提高总体利用率。可以使用 `synth_design` 中的 `-flatten_hierarchy none` 选项全局完成。同样的技巧可以应用于 RTL 中具有 KEEP_HIERARCHY 属性的特定模块。

禁用 LUT 组合和 MUXF 调用

如果在拥塞区域 (> 40%) 中 MUXF* 原语或 LUT 组合的利用率较高，则可以尝试使用消除 MUXF* 使用和 LUT 组合的综合策略，以帮助缓解拥塞。 MUXF* 使用和 LUT 组合可能增加拥塞，因为它们倾向于增加 slice 的输入连接。该 Flow_AlternateRoutability 综合策略和指令指示综合工具避免 MUXF* 结构，并且不生成任何额外的 LUT 组合。

此外， `opt_design` 命令提供了 MUX 优化阶段选项，将 MUXF* 结构重映射到 LUT3 基元以提高可布线性。您可以在拥塞区域使用 `-muxf_remap` 选项，重新映射所有的 MUXF* 单元或在选定数量单元上将 MUXF_REMAP 属性设置为 TRUE，以限制 MUX 重映射的范围。

下图显示了 MUXF* 优化之前和之后的 16-1 MUX。

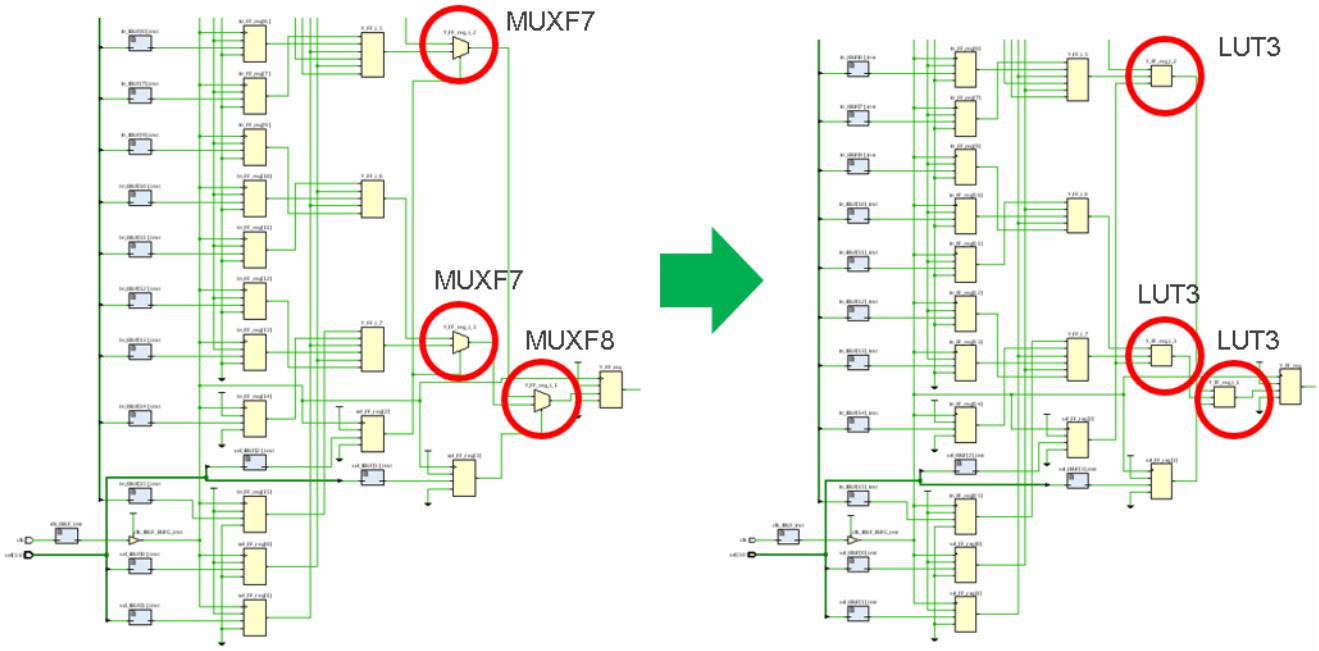


图 5-32：网表之前和之后 MUX 优化

要在执行 MUX 优化后进一步优化网表，请使用 `-remap` 和 `-muxf_remap` 选项。如果可能，这将由 MUXF* 优化生成的 LUT3 基元与连接的逻辑组合。

注释： 如果您使用 Synplify Pro 进行综合，那么您可以使用“Device”标签下“Implementation Options”中的“Enable Advanced LUT Combining”选项。默认情况下，此选项为 ON。如果要修改 Synplify Pro 项目文件 (*prj)，则指定以下内容：`set_option -enable_prepacking 0|1`（默认值为“1”）。

您可以使用以下命令在您的设计中突出显示 LUT 组合：

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

下图显示了这些地区的布局器估计拥堵 （紫色）和 MUXF *使用 （黄色）。在这种情况下，减少 MUXF* 的使用可以帮助减轻拥塞。

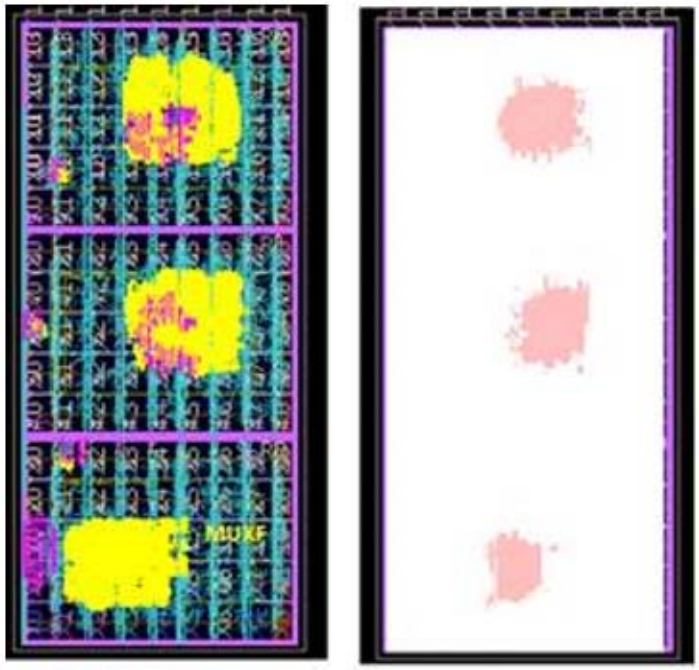


图 5-33：布局器估计拥塞和 MUXF* 使用

下图显示了使用和不使用 LUT 组合的设计的水平拥塞。使用 LUT 组合的单元以紫色突出显示。

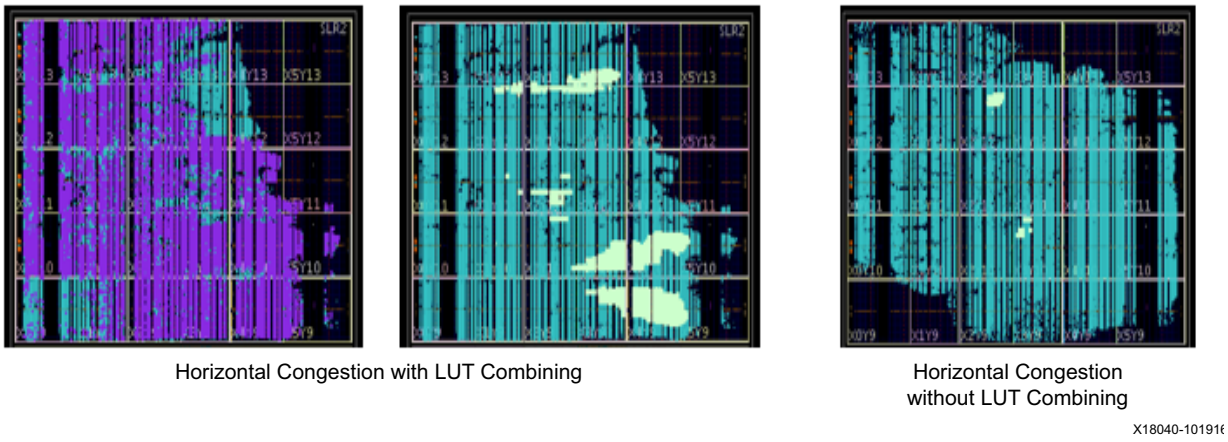


图 5-34：LUT 组合对水平拥塞的影响

使用块级综合策略

在一些情况下，尝试设计中不同模块的不同综合选项或策略或更具优势。例如，一个模块可能需要使用 MUXF* 资源来实现时序关键功能，而设计的其余部分可能受益于实现 LUT 中的逻辑而不是 MUXF* 以减少拥塞。在这种情况下，可用时序关键模块设置 PERFORMANCE_OPTIMIZED 策略，而使用 Flow_AlternateRoutability 策略综合设计的其余部分以减少拥塞。如需了解更多信息，请参阅第 4 章中的“块级综合策略”。

在拥塞区域中限制高扇出网络

具有严格时序约束的高扇出网络需要紧密集群布局以满足时序。如下图所示，这可能导致局部拥塞。高扇出网络还可以通过消耗不再可用于拥塞窗口中的其他网络的布线资源来促成拥塞。

要分析高扇出非全局网络对拥塞窗口中的可布线性的影响，您可以：

- 在拥塞窗口中选择顶层分层模块的叶节点单元格
- 使用 find 命令 (“Edit > Find”) 选择所选单元格对象的所有网络 （过滤掉全局时钟，电源和接地网络）
- 按 Flat Pin Count 降序对网络进行排序
- 选择顶部扇出网络以显示它们与拥塞窗口的关系

这可以快速帮助您识别可能有助于拥塞的高扇出网络。

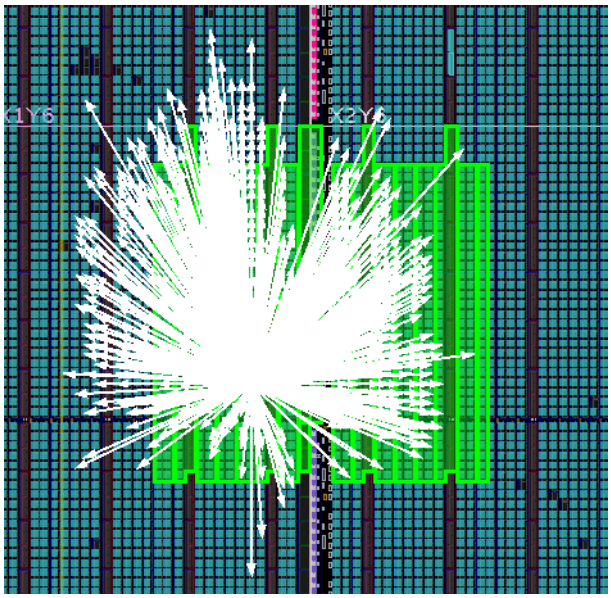


图 5-35：拥堵窗口中的高风扇网

对于在拥塞窗口中具有严格时序约束的高扇出网络，复制驱动器将有助于放松布局约束并减轻拥塞。

具有足够正时序裕量的高扇出网络 （扇出> 5000）可以在全局时钟资源而不是结构资源上布线。对于扇出 > 1000 的高扇出网络，布局器会使用全局布线资源自动布线，只要这些资源在布局器的步骤尾声可用即可。该优化只在不劣化时序的情况下进行。

您还可以在网络上设置属性 CLOCK_BUFFER_TYPE=BUFG，在执行布局器步骤之前让综合或逻辑优化自动插入缓冲器。在执行 `place_design` 之后请检查新插入的缓冲器布局及其驱动程序，并加载布局以验证其是否最佳。如果不理想，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）来控制其位置。

调节编译流程

可通过默认编译流程快速获得设计基准 (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。

策略与指令

策略与指令可用于增大实现解决方案空间，从而为您的设计找到最佳解决方案。策略被全局地应用于工程实现运行，而指令则被分别设置到工程模式和非工程模式下实现流程的每个步骤。在用指令对流程实现自定义化之前，应该首先执行预定义的策略。赛灵思不建议针对非 SSI 技术器件采用 SSI 技术策略。

如果用预定义的策略无法满足时序要求，则您能手动探索定制指令组合。由于布局通常对总体设计性能有很大影响，最好仅利用 I/O 位置约束 （无其它布局约束）来尝试各种布局器指令。通过检查每种布局器运行后的 WNS 和 TNS （可在布局器日志中找到这些数值），您可以选择出提供最佳时序结果的两个或三个指令作为后续实现流程的基础。

针对每个检查点尝试运行多个针对 `phys_opt_design` 和 `route_design` 的指令，同样只保留具有最佳估计 WNS/TNS 或最终 WNS/TNS 的运行方案。在非工程模式下，您必须用 Tcl 脚本详细地描述流程，并保存最佳检查点。在工程模式下，可以为每个布局器指令创建单独的实现运行方案，并将运行方案应用于布局步骤。在布局步骤之后，您可以继续对具有最佳结果的运行方案执行实现操作 （依照 Tcl-post 脚本确定）。

物理约束 （Pblock 和 DSP 和 RAM 宏约束）会妨碍布局器找出最佳解决方案。因此，赛灵思建议您可在没有任何 Pblock 约束的情况下运行布局器指令。在采用指令开始布局之前，可使用下面的 Tcl 命令删除 Pblock：

```
delete_pblock [get_pblocks *]
```

运行 `place_design -directive <directive>` 并分析最佳结果的布局，这样也可提供一个布局规划设计模板，用以稳定每一次运行。

优化反复循环

有时将一个命令反复循环多次有利于获得最佳结果。例如，首先用 `-force_replication_on_nets` 选项运行 `phys_opt_design`，以优化那些可能在布线过程中会对 WNS 产生影响的关键网络。

接下来可用任何指令运行 `phys_opt_design`，以提升设计的整体 WNS。

在非工程模式下，使用下面的命令：

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

在工程模式下，可通过运行 `phys_opt_design` 运行步骤（采用 `-directive` 选项运行）下 Tcl-pre 脚本中的第一个 `phys_opt_design` 命令得到相同结果。

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。可在布局和物理优化过程中加强时序要求，以增大布线器的时序预算。为此，赛灵思建议使用 `set_clock_uncertainty` 约束，原因如下：

- 没有改变时钟关系 （时钟波形保持不变）。
- 是工具计算时钟不确定性的附加 （抖动，相位误差）。
- 专门针对由 `-from` 和 `-to` 选项规定的时钟域或时钟交叉。
- 通过利用空值来覆盖之前的时钟不确定性约束这种方法，可以轻松将其复位。

在任何情况下，赛灵思都建议您：

- 只对无法满足建立时序的时钟或时钟交叉进行过约束。
- 在执行布线步骤之前复位额外的不确定性。

参见下面的实例：

设计在布线前后，在具有 clk1 时钟域的路径上时序相差 -0.2 ns，在 clk2 到 clk3 的路径上时序相差 -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度约束。
   a. 数值应至少是违规总量。
   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。最好能够满足预布线时序。
4. 删除附加不确定性。

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

5. 运行布线器。

在布线完成后，您可以查看时序结果以评估过度约束的优势。如果在布局后时序得到满足，但在布线后仍差了一点未满足，您可以增加不确定性的量再试一次。

建议： 过度约束不要超过 0.5 ns。

使用增量编译

您可以使用增量编译来缩短实现运行时间，并产出更多可预测的结果。赛灵思建议将增量编译作为您的标准时序收敛策略的组成部分。如需了解更多信息，请参阅 《Vivado Design Suite 用户指南：实现》(UG904) [参照 19] 中的链接。

本部分是对高重用模式和低重用模式的建议。在单元重复使用率大于 75% 时，启用高重用模式。在单元重复使用率低于 75% 时，启用低重用模式。

选择高质量的参考检查点

选择如下高质量的参考检查点：

- 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，赛灵思建议运行 route_design -tns_cleanup 命令来清除不属于最劣情况路径的路径。

- 确保用于 synth_design 和 opt_design 的参考检查点工具选项与增量运行匹配。
- 如果有多个参考检查点满足时序且有匹配的 synth_design 和 opt_design 选项，请选择拥塞程度最小的检查点。

  检查 Vivado 布线器日志文件，寻找最初的估计拥塞。详情，请参阅“识别拥塞”。

限制高重用模式下的差异

在高重用模式下，重用的数量取决于参考与增量运行之间的差异，具体包括下列差异：

- 源代码和 IP

- 用于 synth_design 和 opt_design 的工具选项
- 工具版本

工具版本

- 若要取得最佳效果，请按如下方式最大化重用：与标准时序收敛流程并行运行增量流程。
- 如果您有多个不同的 synth_design 和 opt_design 选项运行，请对每个运行使用不同的参考检查点。
- 尽量在各个运行之间使用相同的工具版本。较新的工具版本可能包含对现有选项的算法和阈值的修改。
- 使用 report_incremental_reuse -hierarchical 显示每个层级区域的匹配百分比。如果本应匹配的层级区域为显示高匹配百分比，请比较参考设计和增量设计的工具选项和工具版本。

避免可能重用的下列综合差异：

- 启动寄存器的重定时功能
- 保存或取消逻辑层级
- 改变约束

注释： 在标准运行中使用 place_design、 route_design 和 phys_opt_design 选项不会影响重用。

为高重用模式选择增量编译指令

在高重用模式下，参考检查点结果的可重复性优先。您可以使用下列受支持的指令细调工具行为：

- **默省**：对默省指令，布线走线工具指向来自参考运行的 WNS。这有助于保持与参考检查点和运行时间的一致性。
- **探索**：使用探索指令时，工具指向 WNS = 0.0 ns 的情况。在参考运行极为贴近满足时序时使用该指令，同时您愿意在结果一致性和运行时间折衷，不再努力试图满足时序。该模式可在高难度设计上将 WNS 提升到多达 100 ps。该选项一般存在运行时间命中。

提升高重用模式的时序 QoR

在高重用模式下，用下列方式提升时序 QoR：

- 确保参考检查点已有收敛的时序。

  在参考检查点中，如果 WNS > 0.100 ns，可增大参考运行的力度来提升 WNS。当 WNS 小于 0.100 ns 时，您可以使用探索指令来尝试收敛时序。

- 使用不同的参考检查点运行多个增量运行。

  这一般意味着可使用不同的布局器指令和所有的收敛时序生成参考检查点。

- 不要对增量运行进行布局规划。

  用参考检查点布局覆 Pblock 布局。

- 请遵循 “设计过约束” 中的说明，勿过度约束布局器。

  在增量运行中过度约束设计会严重影响重用，因为工具会努力满足被人为修改后的目标 WNS。

降低低重用模式的 QoR 可变性

在低重用模式里，您可以重用特定单元 （例如设计中的层级单元）或单元类型 （例如 DSP 或块 RAM）。在下列两种情况都成立时这是有效的做法：

- 部分设计运行显示设计能满足时序，但多次运行则不能满足。
- 处于设计早期阶段，正在生成流或正在做重大改动。

在布局特定单元显著影响 WNS 的情况下，重用层级单元是有效的。对 DSP、块 RAM 等块相对密度较高的设计，重用 DSP、块 RAM 或两者都是有用的。

要重用特定的单元或单元类型：

- 分析参考运行，例如检查未能满足的检查点，找出需要指向的区域。
- 在确定需要指向的区域后，将运行集与基线运行集做对比，评估其有效性。
- 使用不同的 `place_design` 指令。

  注释： 支持所有来自标准流程的指令，同时目标 WNS 总是 0.00 ns。

若要重用块内存布局，请使用下列 Tcl 脚本：

```
read_checkpoint -incremental routed.dcp \
    -only_reuse [get_cells -hier -filter {PRIMITIVE_TYPE =~ BMEM.*.*}] -fix_reuse
```

若要重用纯 DSP 布局，请使用下列 Tcl 脚本：

```
read_checkpoint -incremental routed.dcp \
    -only_reuse [get_cells -hier -filter {PRIMITIVE_TYPE =~ MULT.dsp.*}] -fix_reuse
```

若要同时重用块存储器和 DSP 布局，请使用下列 Tcl 脚本：

```
read_checkpoint -incremental routed.dcp \
    -only_reuse [get_cells -hier -filter {PRIMITIVE_TYPE =~ BMEM.*.*}] \
    -only_reuse [get_cells -hier -filter {PRIMITIVE_TYPE =~ MULT.dsp.*}] -fix_reuse
```

若要在特定的层级单元和该单元下的所有层级中重用层级，请使用下列 Tcl 脚本：

```
read_checkpoint -incremental routed.dcp \
    -only_reuse [get_cells <cell_name>] -fix_reuse
```

布局规划考虑因素

布局规划使您可以通过高层次层级布局或详细布局来引导工具。以提供更佳的 QoR 和更具预测性的结果。通过修复最严重的问题或最常见的问题实现最大提升。例如，如存在具有很差时序裕量或高层次逻辑的孤立路径，应利用 Pblock 将这些路径分组到器件的相同区域内，以便修复这些路径。限制布局规划只适用于设计中需要额外用户干预的部分，而不适用于对整个设计进行布局规划。

连接 I/O 及其附近区域的布局规划逻辑，有时会产生在从一个编译到下一个编译具有较好预测性的结果。总之，最好保持 Pblock 的大小在时钟域内。这样可为布局器提供最大的灵活性。应避免叠加 Pblock，因为共享区域有可能变得更加拥塞。在两个 Pblock 之间存在大量连接信号的情况下，考虑将它们合并成单个 Pblock。应最大程度减少穿过 Pblock 的网络的数量。

提示： 当升级到较新版本的 Vivado Design Suite 时，首先尝试在没有 Pblocks 或在具有最小 Pblock （即只有 SLR 级 Pblock）时进行编译，以查看是否存在任何时序收敛问题。以前帮助提高 QoR 的 Pblock 可能阻止在较新版本的工具中找到最佳实现的布局与布线。

群关键逻辑

将关键逻辑分组以避免交叉 SLR 或 I/O 列有助于提升设计的关键路径。下图显示的两个示例是使用 29 个 FIFO36E2 原语实现的大型 FIFO。关键路径是从组中每个 FIFO36E2 的 WRRSTBUSY 引脚通过 5 个 LUT 到组中每个 FIFO36E2 的 WREN 引脚。

- 在左边，示例显示布局器无法找到路径的最佳布局，因为块 RAM 利用率非常高。 FIFO36E2 原语以红色标记。
- 在右边，示例显示了布局器能够满足时序，因为将 FIFO36E2 块分组在避免配置列交叉的矩形中。 FIFO36E2 原语以绿色标记。

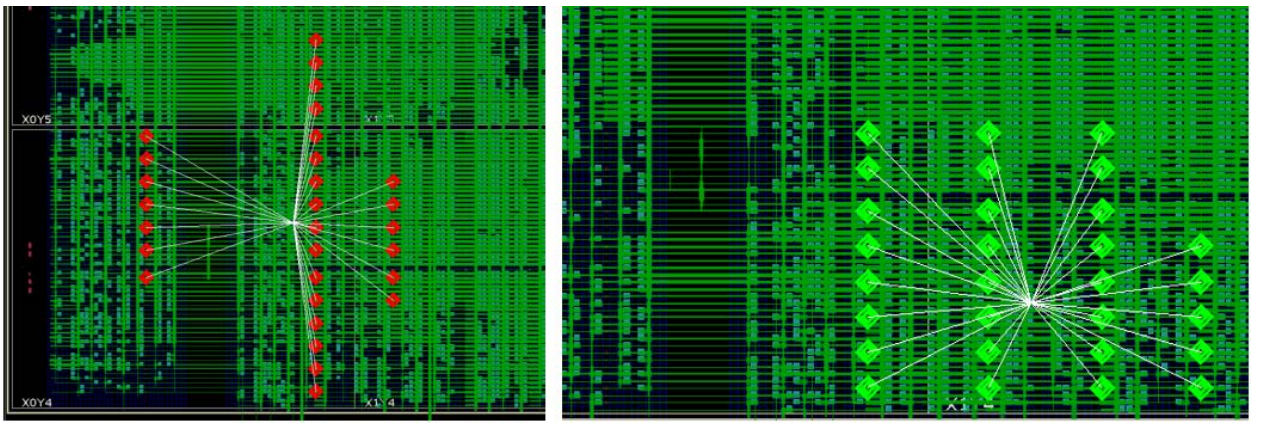


图 5-36：FIFO 位置避免配置列

SSI 技术考虑因素

堆叠硅片互联 (SSI) 技术器件由多个超级逻辑区域 (SLR) 和一个中介层制成。中介层连接被称为超长线路 (SLL)。当从一个 SLR 经过另一 SLR 时会有一些延迟。为了尽量降低 SLL 延迟对设计的影响，应对设计进行布局规划以使 SLR 交叉不包含在关键路径之内。应通过布局规划使 Pblock 仅保持在一个 SLR 之内，以便最大程度减少 SLR 交错，这样可改进使用 SSI 技术器件的设计的时序和布线。

对于高性能设计，需要在主要层级之间有足够的流水线，以使全局布局和 SLR 分区轻松易行。当设计极富挑战性时，SLR 相交点能随着运行而发生变化。除了定义 SLR Pblock 外，您还可以创建与时钟域对齐并位于 SLR 边界附近的其他 Pblock，以限制交叉触发器。以下示例显示了具有以下 Pblock 的 UltraScale ku115 SSI 器件：

- 2 个 SLR Pblock：SLR0 和 SLR1
- 2 个 SLR 交叉 Pblock：SLR0_top_row 和 SLR1_bottom_row

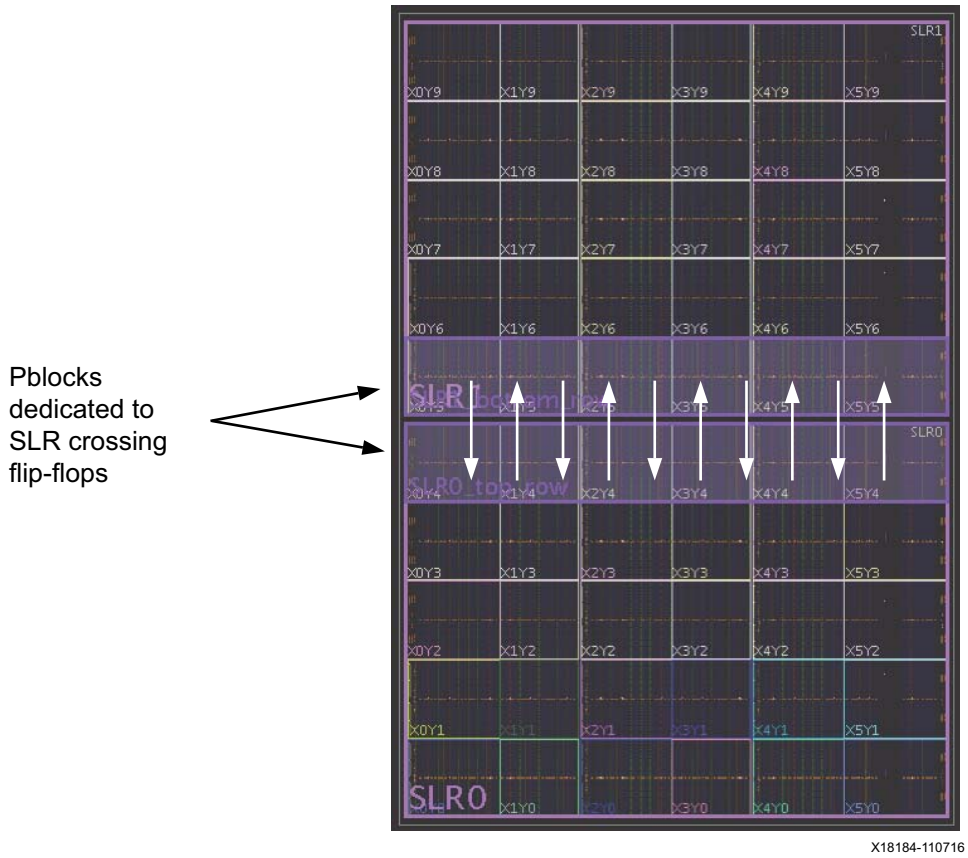


图 5-37：SLR 交叉 Pblock 示例

重要提示： 赛灵思建议对 SLR 交叉 Pblocks 使用 CLOCKREGION 范围，而不使用 LAGUNA 范围。

如需了解更多信息，请参阅 《Vivado Design Suite 用户指南：设计分析和收敛技术》(UG906)[参照 21] 中的链接。

视频： 如需了解更多有关使用布局规划技术以解决设计性能问题的信息，请观看 Vivado Design Suite QuickTake 视频：设计分析和平面布局。

重复使用布局

重新使用块 RAM 宏和 DSP 宏的位置是相当容易的。重复使用此布局有助于减少从一个网表修订到下一个网表修定的结果变化。这些原语通常具有稳定的名称。布局通常易于维护。一些布局指令会导致比其他布局更好的块 RAM 和 DSP 宏布局。您可以尝试将此改进的宏布局从一个布局器运行应用到使用不同布局器指令的其他运行，以提高 QoR。可将块 RAM 布局保存到 XDC 文件中的简单 Tcl 脚本如下所示。

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

您可以编辑 bram_loc.xdc 文件以便仅保留块 RAM 位置约束并将其应用于连续运行。

重要提示： 不要重复使用通用 slice 逻辑的布局。不要为可能更改的设计部分重复使用布局。

功耗分析与优化

鉴于功耗的重要性， Vivado 工具可提供精确的功耗估计方法以及一些功耗优化功能。如需了解更多信息，请参阅《Vivado Design Suite 用户指南：功耗分析和优化》(UG907) [参照 22]。

估计整个流程的功耗

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。一旦功耗与预算值过于接近就可及时采取补救措施。

功耗估计的精确性因设计阶段的不同而变化。要估计综合后到实现阶段的功耗，应运行 report_power 命令，或打开 Vivado IDE 中的“Power Report”。

- 综合后

  网表被映射到目标器件中可用的实际资源。

- 布局后

  将网表组件放到实际的器件资源中。有了这些包装信息，最终的逻辑资源数和配置将变得可用。

  这些精确数据可被导出到 Xilinx Power Estimator (XPE) 电子数据表中。这样便可以：

  - 在 XPE 中执行假设分析。
  - 提供可精确填充电子数据表的基础内容，便于以后在具有相同特性的设计中使用。

- 布线后

  在布线完成后，所用布线资源的所有相关细节和设计中每个路径的具体时序信息均可被定义。

除了对最佳和最差逻辑和布线延迟下的实现电路功能进行核实之外，仿真器还能显示内部节点的具体行为，包括毛刺。在实际测量原型设计电路板的功耗之前，此阶段的功耗分析可以提供最为精确的功耗估计。

使用电源约束顾问

电源约束顾问 (Power Constraint Advisor) 可报告设计中所有控制信号上工具计算的开关活动。如需了解更多信息，请参阅 《Vivado Design Suite 用户指南：功耗分析和优化》(UG907) [参照 22] 中的链接。

精确功耗分析的最佳做法

要进行精确的功耗分析，请确保具有精确的时钟约束、 I/O 约束和信号。如需了解更多信息，请参阅 《Vivado Design Suite 用户指南：功耗分析和优化》(UG907) [参照 22] 中的链接。

在运行 Vivado Design Suite 功耗分析后检查设计的配电

为了确定您设计中哪些部分对总功耗影响最大，您能查看总片上功耗和热属性以及资源级别的功耗详细信息。如需了解更多信息，请参阅 《Vivado Design Suite 用户指南：功耗分析和优化》(UG907) [参照 22] 中的链接。

在运行 Vivado Design Suite 功耗分析后进一步优化控制信号活动

当使用基于 SAIF 的注释进行精确功耗分析时，可在完成第一级分析后精确调整功耗分析。如需了解更多信息，请参阅《Vivado Design Suite 用户指南：功耗分析和优化》(UG907) [参照 22] 中的链接。

功耗优化

如果功耗估计值超出预算，您必须采取以下章节所述的步骤来降低功耗。

分析功耗估计和优化结果

使用 `report_power` 生成功耗估计报告之后，赛灵思建议执行如下步骤：

- 在“Summary”部分检查总功耗。总功耗和结温是否符合热性能与功耗预算？
- 如果结果严重超出预算，应根据块类型和电源轨检查功耗总分配情况。这样可找到功耗最高的块。
- 检查“Hierarchy”部分。按照层级划分后，很容易找到功耗最高的模块。可以深入研究特定块以确定各块的功能。您也可以在 GUI 中进行交叉探测，以确定模块中特定部分的编码方式，并确定是否存在可降低功耗的其它编码方式。

运行功耗优化

提示： 要实现功耗优化的最大效用，请参阅第 3 章中的“提升功耗的编码方式”。

功耗优化功能能够使整个设计或部分设计 （当使用 `set_power_opt`）的功耗降至最低。

功耗优化可以在设计流程布局前或布局后使用，但不能在两个阶段同时使用该功能。布局前功耗优化步骤着重于最大限度节省功耗。但这会降低时序性能 （较少情况下）。如果保持时序是首要目标，那么赛灵思建议使用布局后功耗优化步骤。该步骤只执行可保持时序不变的功耗优化。

些情况下，出于传统 IP 或时序的考虑，应保存部分设计，使用 `set_power_opt` 命令来排除这些设计部分 （诸如特定层级、时钟域或单元类型）并重新运行功耗优化。

使用功耗优化报告

为确定功耗优化的影响，应在 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
```

使用时序报告确定功耗优化的影响

功耗优化的作用是在最大限度节省功耗的同时将其对时序的影响最小化。但在特定情况下，如果时序在功耗优化后变差，您可以采用一些技术来抵消这一影响。

尽可能只在的非时序关键时钟域或模块中使用 `set_power_opt` XDC 命令来确定和应用功耗优化。如果最关键的时钟域恰好涵盖设计的绝大部分或者消耗的功耗最多，那么应检查关键路径中的所有单元是否都进行了功耗优化。

进行过功耗优化的对象都具有 `IS_CLOCK_GATED` 属性。应在功耗优化中排除这些单元。

要定位时钟门控单元，运行如下 Tcl 命令：

```
get_cells -hier -filter {IS_CLOCK_GATED==1}
```

配置与调试

成功完成设计实现后，下一步就是将设计载入 FPGA 并在硬件上运行。配置是将特定应用数据（比特流）加载到 FPGA 器件内部存储器的过程。如果设计不能满足硬件需求，就需要进行调试。

请参阅下列资源，以便详细了解配置及调试软件的流程和命令：

- 《Vivado Design Suite 用户指南：编程和调试》(UG908) [参照 24]
- 《Vivado Design Suite Tcl 命令参考指南》(UG835) [参照 13]
- 《7 系列 FPGA 配置用户指南》(UG470) [参照 38]
- (UG570) [参照 38]
- Vivado Design Suite QuickTake 视频教程：如何使用 Vivado 中的 write_bitstream 命令

配置

首先，您必须成功地综合和实现您的设计，以创建比特流图像。一旦生成了比特流并所有 DRC 进行分析和校正，您就能使用以下方法之一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。

可以使用 Vivado 工具来完成以下操作：

- 创建 FPGA 比特流 （.bit 或 .rbt）。
- 选择“Tools > Edit Device Properties”，以查看比特流生成的配置建立。
- 将比特流格式转换成闪存编程文件 （.mcs）。
- 使用以下方法之一对设备进行编程：
  - 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。除非指定擦除全部芯片，否则只擦除指定 MCS 覆盖的地址范围。

重要提示： Vivado Design Suite Device Programmer 可使用 JTAG 读取赛灵思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，该状态寄存器还便于您对模式引脚设置 M[2:0] 和总线宽度检测结果进行验证。如需了解更多有关 Status 寄存器的信息，请参阅您器件的配置用户指南 [参照 38]。

提示： 如配置不成功，您就能使用 FPGA 器件上的 JTAG 回读/验证操作来明确所需的配置数据是否已正确载入器件。

调试

系统内调试允许您在目标器件上实时地调试设计。如果您发现很难复制到仿真器上，这个步骤就是必要的。

就调试而言，您需要为设计提供专门的调试硬件，以便观察和控制设计。调试完成后，可以将仪器或专用硬件移除，从而提升性能，减少逻辑。

FPGA 设计的调试是一个多步骤、反复循环的过程。和大多数复杂问题的处理方式一样，最好把 FPGA 调试流程分成多个较小的步骤，集中精力逐一处理，而不是试图让整个设计一次性投入工作。

虽然实际调试步骤在您设计成功实现之后进行，但赛灵思 建议在设计周期中尽早规划调试的方案和位置。用户可从 Vivado IDE 中“Flow Navigator”窗口的“Program and Debug”菜单运行 FPGA 器件编程和设计系统内调试的一切必要命令。

以下是调试步骤：

1. 探测：确定设计中需要探针探测的信号和探针探测的方法。
2. 实现：实现的设计包含连接在探针网络上的额外调试 IP。
3. 分析：与设计中包含的调试 IP 进行交互，进而调试和验证功能问题。
4. 修正相位：修正任何缺陷并根据需要反复进行。

如需了解更多信息，请参阅 《Vivado Design Suite 用户指南：编程和调试》(UG908) [参照 24]。

设计探测

Vivado 工具提供多种在设计中添加调试探针的方法。下表逐一列明这些方法并介绍每种方法各自的优劣。

表 5-10：调试流程

调试流程名称	流程步骤	优/劣
HDL 实例化探针探测流程	在 HDL 源中明确地将信号连接到 ILA 调试核实例。	• 您必须在设计中手动添加/移除调试网络和 IP，即必须修改 HDL 源文件 • 利用这种方法可以在 HDL 设计层进行探针探测。 • 在生成、实例化和连接调试核时容易出错

表 5-10：调试流程 （续）

调试流程名称	流程步骤	优/劣
网表插入探针探测流程（建议）	使用以下两种方法之一明确调试信号： • 用 MARK_DEBUG 属性在 RTL 源代码中标记调试信号。 • 用 MARK_DEBUG 右键点击菜单选项在综合设计网表中选择调试网络。 一旦信号做上调试标记，则用 Set up Debug 向导来指导分步完成网表插入探针探测流程。	• 这种方法灵活性最高，有良好的预测能力。 • 这种方法便于在各个不同设计层级进行探针探测 （如 HDL、综合设计、系统设计）。 • 这种方法无需修改 HDL 源文件。
基于 Tcl 的网表插入探针探测流程	用 set_property Tcl 命令在调试网络上设置 MARK_DEBUG 属性，然后用 Netlist insertion probing Tcl 命令创建调试核并连接其到调试网络。 请参阅“修改已实现的网表以替换现有的调试探针”了解后综合 ILA 核情况。	• 这种方法能全自动插入网表 • 您可通过调整 Tcl 命令开启/禁用调试功能。 • 这种方法无需修改 HDL 源文件。

选择调试网络

赛灵思对选择调试网络提出如下建议：

- 将探针探测网络布置在特定层级的边界上 （输入或输出）。这种方法有助于迅速隔离问题区域。这样便于用户在需要的时候深入层级探针探测。
- 勿用探针探测组合逻辑路径之间的网络。如果在组合逻辑路径中间的网络上添加 MARK_DEBUG，就无法在流程的实现阶段应用可适用的优化功能，导致出现不理想的 QOR 结果。
- 为了获得周期精确数据采集，探针网络可同步。

使用 MARK_DEBUG 保留调试探针探测网络名称

用户可以在 RTL 阶段或综合后阶段将信号做上调试标记。 MARK_DEBUG 属性在网络上可避免网络被复制、重新定时、移除或以其它方式进行优化。用户可以在高层次端口、网络、层级模块端口以及层级模块内置网络上应用 MARK_DEBUG 属性。这种方法最有可能保留综合后 HDL 信号名称。做有调试标记的网络显示在“Debug”窗口综合后的“Unassigned Debug Net”文件夹下。

按如下所示增加 mark_debug 属性到 HDL 文件:

VHDL

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

Verilog：

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

您同样也能在综合后网表中增加调试网络。这些方法无需修改 HDL 源文件。然而，也可能存在由于网表优化涉及了采用或合并设计结构，而导致在综合的地方没有保留原始 RTL 信号的情况。完成综合后，可以用下列任何方式添加用于调试的网络：

- 在任何设计视图中选择一个网络 （比如“Netlist”或“Schematic”窗口），然后点击右键点击“Mark Debug”。
- 在任何设计视图中点击选择一个网络，然后拖放该网络到“Unassigned Debug Net”文件夹。
- 在 Set Up Debug 向导中使用网络选择器。

- 使用属性窗口或 Tcl 控制台设置 Mark_DEBUG 属性。

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

在当前开放式网表上应用 mark_debug 属性。这个方法是灵活的，因为您能通过 Tcl 命令打开或关闭 MARK_DEBUG。

使用 ILA 核

使用 Integrated Logic Analyzer (ILA) 核，可以在 FPGA 器件上进行实现后设计的系统内调试。如果需要监测设计中的信号，就可以使用此核。另外，您还可以使用这个功能触发硬件事件并以系统级速度采集数据。

ILA 核和时序考虑因素

ILA 核的配置对满足整体设计时序目标有影响。请根据下列建议尽量减少对时序的影响：

- 审慎选择探头宽度。探头宽度越大，对资源利用和时序的影响越大。
- 审慎选择 ILA 核数据深度。数据深度越大，对块 RAM 资源利用和时序的影响越大。
- 确保为 ILA 核选择的时钟都是自由运行时钟。如果无法满足，可能造成当设计加载到器件上以后无法与调试核通信。
- 确保提供给 dbg_hub 的时钟是自由运行时钟。如果无法满足，可能造成当设计加载到器件上以后无法与调试核通信。可以使用 connect_debug_port Tcl 命令把调试集线器的 clk 引脚连接到自由运行时钟上。
- 在添加调试核之前收敛设计上的时序。赛灵思不建议使用调试核调试相关时序问题。
- 如果您仍然观察到因添加 ILA 调试核造成时序劣化以及关键路径位于 dbg_hub 中时，请执行下列步骤：
  a. 打开综合设计。
  b. 找到网表中的 dbg_hub 单元；
  c. 转至 dbg_hub 的属性；
  d. 找到 C_CLK_INPUT_FREQ_HZ 属性。
  e. 将其设置为连接到 dbg_hub 的时钟的频率 （单位：Hz）；
  f. 找到 C_ENABLE_CLK_DIVIDER 属性并启用该属性；
  g. 重新实现设计。
- 确保输入到 ILA 核的时钟与正在探针探测的信号同步。如不能满足，在设计编程到器件中时会产生时序问题和通信失败。
- 确保在硬件上运行设计之前满足时序要求。如不能满足，会造成不可靠的结果。

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： 本列表是基于一个设计有一个 ILA，不代表所有的设计。

表 5-11：ILA 特性对设计时序和资源的影响

ILA 特性	何时使用	时序	占位面积
采集控制/存储条件	• 采集相关信息 • 有效利用数据采集存储 (块 RAM)	中影响力到高影响力	• 不新增块 RAM • 轻微增加 LUT/FF 数量
高级触发器	• 当 BASIC 的触发条件不充分时 • 使用复杂的触发来专注在问题区域	高影响力	• 不新增块 RAM • 稳健增加 LUT/FF 数量

表 5-11：ILA 特性对设计时序和资源的影响 （续）

ILA 特性	何时使用	时序	占位面积
每个探针探测端口的比较器数量 注記 ：最大数值是 4。	在多种条件下使用探针探测： • 1-2 为基础 • 1-4 为高级 • +1 为采集控制	中影响力到高影响力	• 不新增块 RAM • UT/FF 从轻微到稳健的增长数量
数据深度	采集更多数据样本	高影响力	• 每个 ILA 核的额外块 RAM • 轻微增加 LUT/FF 数量
ILA 探针探测端口宽度	大量的总线与标量进行调试比较	中影响力	• 每个 ILA 核的额外块 RAM • 轻微增加 LUT/FF 数量
探针探测端口数量	探测一些网络	低影响力	• 每个 ILA 核的额外块 RAM • 轻微增加 LUT/FF 数量

提示： 在 FPGA 设计的早期阶段， FPGA 中通常有大量的备用资源可用于调试。

对于高速时钟设计，可进行以下考虑：

- 限制被调试的信号数量和宽度
- 将输入探针探测用流水线输送到 ILA (C_INPUT_PIPE_STAGES)，可以形成额外级别的管道阶段 (pipe stage)。

对于有 MMCM/BUFG 可用性限制的设计，可以考虑在设计中为最低时钟频率的调试集线器定时来代替在调试集线器里面使用时钟分频器。

使用 Vivado IP 集成器进行调试设计

Vivado IP 集成器提供不同的方法来建立调试设计。可以使用以下流程之一将核添加到 IP 集成器设计中。选择的流程取决于您的偏好以及您想调试的网络和信号类型。

- 使用 System ILA 核在块设计中调试接口和网络。

  使用该流程来：

  - 使用 MicroBlaze 器件或 Zynq-7000 All Programmable (AP) SoC 的交叉触发器功能来进行软硬件的协同验证。
  - 验证接口级连接功能。
- 网表插入流程

  在综合后设计中使用该流程来分析 I/O 端口和内部网络。

注释： 通过使用两个流程的结合来调试设计。

如需了解更多在 IP 集成器设计中如何使用 System ILA 的信息，请参阅 《Vivado Design Suite 用户指南：采用 IP 集成器设计 IP 子系统》(UG994) [参照 26]。

在 Vivado 硬件管理器中调试 AXI 接口

使用 IP 集成器中的 System ILA IP 可以在赛灵思器件上执行实现后设计的系统内调试。如果需要监控设计中的接口和信号，就可以使用这个特性。

如果在 IP 集成器块设计中插入了 System ILA 调试核，则能够在下图所示的“Waveform”窗口中调试和监控 AXI 处理和读取/写入事件。 “Waveform”窗口可显示与 System ILA IP 探测的接口相对应的接口插槽、处理、事件和信号组。

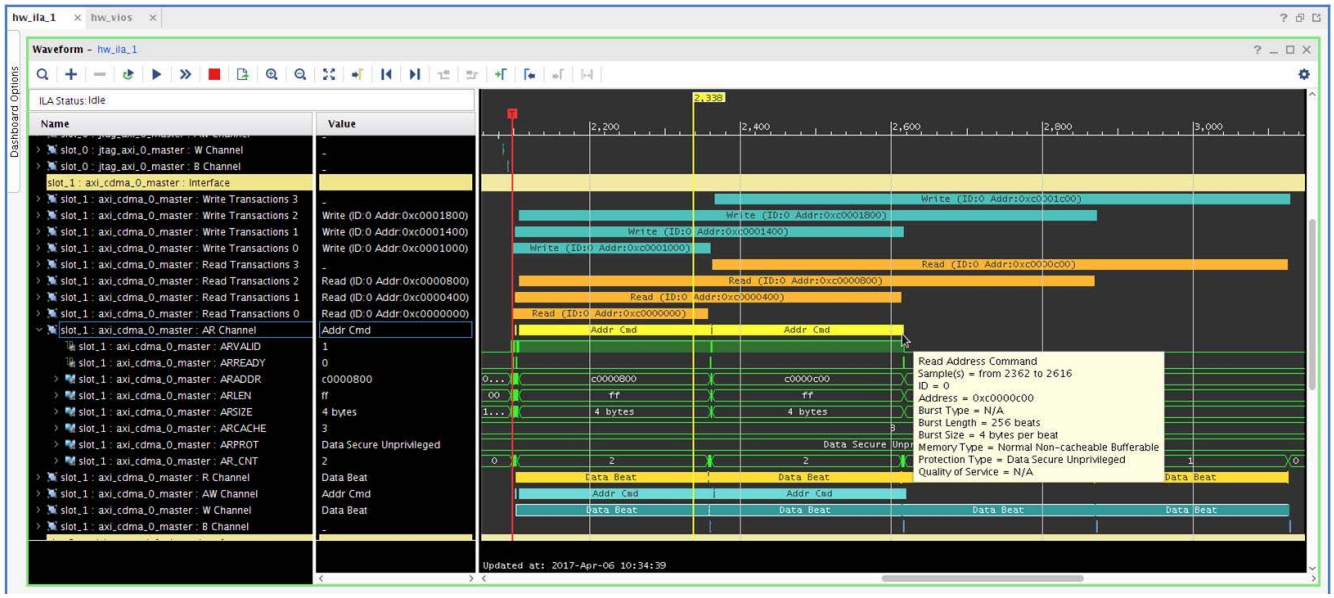


图 5-38：“Waveform”窗口

如需了解更多有关 Vivado 硬件管理器中 System ILA 和调试 AXI 接口的信息，请参阅《Vivado Design Suite 用户指南：编程和调试》(UG908) [参照 24] 中的链接和链接。

使用系统内 IBERT

系统内 IBERT 核通过对 UltraScale 和 UltraScale+™ 器件中收发器 RX 数据上的眼图扫描图提供 RX 裕量分析。该核使能 GTH/GTY 收发器的配置和调整，并通过与收发器的动态重新配置端口 (DRP) 通信的逻辑进行访问。您可以使用核更改属性建立及寄存器，以控制 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。每个设计都只需一个系统内 IBERT 的实例。系统内 IBERT 可与设计中使用的所有 GT 一起使用。但是，必须根据不同的 GT 类型 （例如， GTH、 GTY）生成单独的系统内 IBERT 核。

创建具有内部系统时钟的系统内 IBERT 设计可以防止执行扫描。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) 连接到系统内 IBERT IP 的 clk/drpclk_i 输入端口时，眼图扫描不完整。

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，也可单击 Vivado Serial I/O Analyzer 中的任何可用链接。转到“Properties”窗口，并在 LOGIC 字段下找到 MB_RESET reg。将其设为 1，然后切换回 0。重新运行眼图扫描或扫描。

如需了解更多有关此核的信息，请参阅 《系统内 IBERT LogiCORE IP 产品指南》(PG246) [参照 50]。

运行有关调试的 DRC

Vivado Design Suite 提供有关调试的 DRC，即是在运行 `report_drc` 时作为默认规则卡 (Rule Deck) 的一部分。 DRC 检查如下所示:

- 器件的块 RAM 资源是超出的因为调试核的现今需求。
- 无时钟网络连接到调试核的时钟端口。
- 调试核上的端口未连接。

生成 AXI 事务

使用 JTAG-to-AXI 调试核可以生成与在硬件上运行的系统中的各种 AXI Full 和 AXI Lite 辅核交互的 AXI 事务处理。从 IP 目录将此核实例化到设计中，就可以在运行时间生成 AXI 事务处理和 FPGA 内部的调试/驱动 AXI 信号。您可以在设计中使用此核，且无需处理器。

修改已实现的网表以替换现有的调试探针

可以在布局和布线的设计检查点中替换连接到 ILA 核的调试网络。您可以使用工程变更单 (ECO) 流程执行此操作。这是一种高级设计流程，用于接近完成的设计，在其中您需要交换连接到现有 ILA 探针端口的网络。如需了解更多有关如何使用 ECO 流程来修改现有 ILA 核上的网络的信息，请参阅 《Vivado Design Suite 用户指南：实现》(UG904) [参照 19] 中的链接。

在已实现的网表上插入、删除或编辑 ILA 核

如果您希望添加、删除或修改 ILA 核 （例如，调整探针宽度，更改数据深度等），赛灵思建议您使用增量编译流程。调试核的增量编译流程在综合设计或检查点 (DCP) 上运行，并在理想情况下使用来自之前执行运行的引用实现检查点。与完全重新实现设计相比，这种方法可以节省您的时间。如需了解更多有关使用增量编译流程插入、删除或编辑 ILA 核的信息，请参阅 《Vivado Design Suite 用户指南：编程和调试》(UG908) [参照 24] 中的链接。

附录 A

附加资源与法律提示

赛灵思资源

如需了解问答、技术文档、下载以及论坛等支持性资源，请参阅赛灵思技术支持。

解决方案中心

如需了解设计周期各阶段有关器件、工具和 IP 等的技术支持，请参阅赛灵思解决方案中心。相关专题包括设计辅助、建议和故障排除提示等。

Documentation Navigator 与设计中心

Xilinx Documentation Navigator 提供了访问赛灵思文档、视频和支持资源的渠道，您可以在其中筛选搜索信息。打开 Xilinx Documentation Navigator (DocNav) 的方法：

- 在 Vivado IDE 中，单击“Help > Documentation and Tutorials”。
- 在 Windows 中，单击“Start > All Programs > Xilinx Design Tools > DocNav”。
- 在 Linux 命令提示中输入 `docnav`。

赛灵思设计中心提供了根据设计任务和其他话题整理的文档链接，您可以使用链接了解关键概念以及常见问题解答。访问设计中心：

- 在 Xilinx Documentation Navigator 中，单击“Design Hubs View”视图。
- 在赛灵思网站上，请参阅设计中心页面。

注释：如需了解更多有关 Documentation Navigator 的信息，请参阅赛灵思网站上的 Documentation Navigotor。

参考资料

以下技术文档是本指南非常实用的补充材料。

1. Vivado® Design Suite 技术文档
2. UltraFast™ 设计方法检查表

Vivado Design Suite 用户指南及参考资料指南

3. 《Vivado Design Suite 用户指南：版本说明、安装和许可》(UG973)
4. 《Vivado Design Suite 用户指南：I/O 管脚分配和时钟规划》(UG899)
5. 《Vivado Design Suite 用户指南：设计流程简介》(UG892)
6. 《Vivado Design Suite 用户指南：如何使用 Vivado IDE》(UG893)
7. 《Vivado Design Suite 用户指南：如何使用 Tcl 脚本》(UG894)
8. 《Vivado Design Suite 用户指南：系统级设计输入》(UG895)
9. 《Vivado Design Suite 用户指南：采用 IP 进行设计》(UG896)
10. 《Vivado Design Suite 用户指南：嵌入式处理器硬件设计》(UG898)
11. 《Vivado Design Suite 用户指南：逻辑仿真》(UG900)
12. 《Vivado Design Suite 用户指南：着手设计》(UG910)
13. 《Vivado Design Suite Tcl 命令参考指南》(UG835)
14. 《Vivado Design Suite 属性参考指南》(UG912)
15. 《AXI BFM 内核 LogiCORE IP 产品指南》(PG129)
16. 《Vivado Design Suite 用户指南：综合》(UG901)
17. 《Vivado Design Suite 用户指南：高层次综合》(UG902)
18. 《Vivado Design Suite 用户指南：如何使用约束》(UG903)
19. 《Vivado Design Suite 用户指南：实现》(UG904)
20. 《Vivado Design Suite 用户指南：层级设计》(UG905)
21. 《Vivado Design Suite 用户指南：设计分析和收敛技术》(UG906)
22. 《Vivado Design Suite 用户指南：功耗分析和优化》(UG907)
23. 《Xilinx Power Estimator 用户指南》(UG440)
24. 《Vivado Design Suite 用户指南：编程和调试》(UG908)
25. 《Vivado Design Suite 用户指南：部分重配置》(UG909)
26. 《Vivado Design Suite 用户指南：采用 IP 集成器设计 IP 子系统》(UG994)
27. 《Vivado Design Suite 用户指南：创建和封装定制 IP》(UG1118)
28. 《Vivado Design Suite 7 系列 FPGA 和 Zynq-7000 All Programmable 库指南》(UG953)
29. 《UltraScale 架构库指南》(UG974)

Vivado Design Suite 教程

30. 《Vivado Design Suite 教程：高层次综合》(UG871)
31. 《Vivado Design Suite 教程：设计流程简介》(UG888)

32. 《Vivado Design Suite 教程：I/O 管脚分配和时钟规划》(UG935)
33. 《Vivado Design Suite 教程：逻辑仿真》(UG937)
34. 《Vivado Design Suite 教程：嵌入式处理器硬件设计》(UG940)
35. 《Vivado Design Suite 教程：部分重配置》(UG947)

其它赛灵思技术文档

36. 《7 系列 FPGA PCB 设计用户指南》(UG483)

    《UltraScale 架构 PCB 设计用户指南》(UG583)

    《Zynq-7000 All Programmable SoC PCB 设计指南》(UG933)
37. 《UltraFast 嵌入式设计方法指南》(UG1046)
38. 《7 系列 FPGA 配置用户指南》(UG470)

    《UltraScale 架构配置用户指南》(UG570)
39. 《7 系列 FPGA SelectIO 资源用户指南》(UG471)

    《UltraScale 架构 SelectIO 资源用户指南》(UG571)
40. 《7 系列时钟资源指南》(UG472)

    《UltraScale 架构时钟资源用户指南》(UG572)
41. 《UltraScale 架构 GTH 收发器用户指南》(UG576)

    《UltraScale GTY 收发器高级规范用户指南》(UG578)
42. 《UltraScale 架构 Gen3 面向 PCI Express® LogiCORE IP 的集成块产品指南》(PG156)
43. 《7 系列 FPGA 存储器资源用户指南》(UG473)
44. 《7 系列 FPGA DSP48E1 slice 用户指南》(UG479)
45. 《UltraScale 架构用户指南》(UG579)
46. 《7 系列 FPGA 与 Zynq-7000 All Programmable SoC XADC 双 12 位 1 MSPS 数模转换器用户指南》(UG480)
47. 《参考系统：使用 IP 集成器的 Kintex-7 MicroBlaze 系统仿真》(XAPP1180)
48. 《Zynq-7000 SoC 与 7 系列 FPGA 器件存储器接口用户指南》(UG586)
49. UltraScale 架构 FPGA 存储器 IP LogiCORE IP 产品指南 (PG150)
50. 《系统内 IBERT LogiCORE IP 产品指南》(PG246)
51. 赛灵思白皮书：《使用 S 参数模型仿真 FPGA 电源完整性》(WP411)
52. 7 系列原理图查看建议 (XMP277)

    UltraScale 架构原理图查看检查表 (XTP344)

    UltraScale+ FPGA and Zynq UltraScale+ MPSoC 原理图查看检查表 (XTP427)
53. 《UltraScale FPGA BPI 配置与闪存编程》(XAPP1220)
54. 《利用 7 系列实现 BPI 快速配置与 iMPACT 闪存编程》(XAPP587)
55. 《结合使用 SPI 闪存和 7 系列 FPGA》(XAPP586)
56. 《利用 UltraScale FPGA 实现 SPI 配置与闪存编程》(XAPP1233)
57. 《使用加密确保 7 系列 FPGA 比特流的安全》(XAPP1239)

提示：使用文档导航器访问全套赛灵思技术文档。如需了解更多信息，请参阅“访问其他技术文档和培训资料”。

培训资料

1. UltraFast 设计方法 培训课程
2. Vivado Design Suite QuickTake 视频教程：UltraFast Vivado 设计方法
3. Vivado Design Suite QuickTake 视频教程：Vivado 设计流程
4. Vivado Design Suite QuickTake 视频教程：使用 Vivado IP 集成器定位 Zynq
5. Vivado Design Suite QuickTake 视频教程：Vivado Design Suite 中的部分重配置
6. Vivado Design Suite QuickTake 视频教程：创建不同类型的项目
7. Vivado Design Suite QuickTake 视频教程：管理工程设计源文件
8. Vivado Design Suite QuickTake 视频教程：使用版本控制系统
9. Vivado Design Suite QuickTake 视频教程：管理 Vivado IP 版本升级
10. Vivado Design Suite QuickTake 视频教程：I/O 管脚分配
11. Vivado Design Suite QuickTake 视频教程：在 Vivado 中配置和管理可重用 IP
12. Vivado Design Suite QuickTake 视频教程：如何使用 Vivado 中的 write_bitstream 命令
13. Vivado Design Suite QuickTake 视频教程：设计分析和平面布局
14. Vivado Design Suite QuickTake 视频教程：介绍 UltraFAST 设计方法检查表
15. Vivado 视频辅导资料

请阅读：重要法律提示