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

UG949 (v2015.1) 2015 年 6 月 1 日

条款中英文版本如有歧异，概以英文本□准。

修订历史

下表列出了本文档的修订历史。

日期	版本	修订
2015 年 06 月 01 日	2015.1	重新编制并更新第 2 章：使用 Vivado Design Suite，包括新增的源文件管理及版本控制建议部分。 更新第 3 章：单板和器件规划中的I/O 管脚分配设计流程。 重新编制并更新第 4 章：设计创建。 更新第 5 章：实现中的时序收敛，包括将常见设计瓶颈说明移到 《Vivado Design Suite 用户指南：设计分析和收敛技术》(UG906)中。 重新编制和更新第 6 章：配置与调试，包括增加到各种附加资源的链接。
2014 年 10 月 14 日	2014.3	修订 IP 流程相关章节。根据特定反馈/建议进行了较小的修正/说明。
2014 年 04 月 02 日	2014.1	精减“功耗”部分重点修订“Vivado Design Suite 流程”和“配置与调试”两章。修改特定编排错误、标题名称/字体及其它小的改动。
2013 年 12 月 18 日	2013.4	删除检查表附录。用 Documentation Navigator 中提供的检查表版本替换这些链接。
2013 年 11 月 25 日	2013.3	修正目录中的错误内容。
2013 年 10 月 27 日	2013.3	修改错误超级链接。
2013 年 10 月 23 日	2013.3	赛灵思初始版本

目录

第 1 章：引言

关于本指南 5
指南内容 5
指南适用性与参考资料 5
对设计方法的需求 6
设计方法检查表 6
设计流程 7
快速验证 9
访问技术文档和培训资料 10

第 2 章：使用 Vivado Design Suite

Vivado Design Suite 简介 13
Vivado Design Suite 使用模型 16
配置 IP 18
使用 IP 集成器创建 IP 子系统 21
封装定制 IP 和 IP 子系统 24
创建定制外设 24
逻辑仿真 25
综合、实现与设计分析 31
源文件管理及版本控制建议 31
将设计和 IP 升级到最新的 Vivado Design Suite 版本 41

第 3 章：单板和器件规划

单板和器件规划简介 43
PCB 布局建议 43
时钟资源规划与分配 44
I/O 管脚分配设计流程 46
FPGA 电源因素与系统关联性 58
利用 Xilinx Power Estimator (XPE) 进行最差情况功耗分析 65
配置 68

第 4 章：设计创建

设计创建简介 69
定义理想的设计层级 69
RTL 编码指南 72
充分利用 IP 核 116
利用约束 121

第 5 章：实现

实现简介 153
综合 153
综合属性 156

自下而上流程 157
综合后的步骤 158
实现设计 160
时序收敛 170
功耗. 201

第 6 章：配置与调试
配置与调试简介 209
配置. 209
调试. 213

附录 A: 基线 (baselining) 与时序约束验证流程
引言. 224
流程. 224

附录 B: 附加资源与法律提示
赛灵思资源 226
解决方案中心 226
参考资料 226
培训资料 228
请阅读：重要法律提示. 228

第 1 章

引言

关于本指南

赛灵思可编程器件含有数百万个逻辑单元 (LC)，集成了越来越多的当前复杂电子系统，其中包括：

- 嵌入式子系统
- 模拟与数字处理
- 高速连接功能
- 网络处理

为了在很短的设计时间内创建出如此复杂的系统，设计人员需要综合 RTL 级的众多大型逻辑模块，并重复利用赛灵思或第三方提供的 IP模块。

鉴于该流程的复杂性，采用统称为 UltraFast™ 设计方法的一系列最佳实践至关重要，这一系列最佳实践可最大限度地提高系统集成和设计实现方面的生产力。

指南内容

本指南介绍了为高效、快速完成设计实现应遵循的设计方法流程，从而充分发挥赛灵思器件和工具的最大价值。

在大多数情况下，本指南会介绍有关建议背后的推理过程。理解有关推理，更利于您了解建议方法能实现什么样的潜在效果，也有助于采取适当的预防措施。

指南适用性与参考资料

尽管本指南主要适用于赛灵思 Vivado® Design Suite，但本指南中的大多数概念性信息也可用于赛灵思 ISE® Design Suite。本指南提供了高级信息、设计指南和设计决策权衡信息。

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。如欲了解最新详细信息，包括工具使用和设计方法的说明，仍应参考上述技术文档。完整参考文件列表,敬请参见：附录 B：附加资源与法律提示。

本指南在不同地方会就特定任务给出 Vivado 工具命令。运行带有 `-help` 选项的命令，可了解详细信息（包括实例使用）。

对设计方法的需求

在当今日益复杂的电子产品中使用的高级算法正在挑战密度、性能和功耗的极限，同时也使设计团队面临诸多挑战，要求他们必须在限定的预算内按时完成设计目标，获得机会窗口。 UltraFast 设计方法可帮助工程管理人员：

- 加速产品上市进程，从而提高产品营收和市场份额。
- 制定一个准确的工程进度与成本估算表，以降低风险。

本指南汇集了有关板级规划、设计创建、 IP 集成、设计实现与收敛技术、编程以及硬件调试等各个方面的最佳实践。这些最佳实践和建议是过去多年的广泛专家级用户的经验总结。本指南中的有关建议将帮助您像许多其他赛灵思客户一样取得成功。

Vivado Design Suite 通过提供下列特性，还可让 UltraFast 设计方法实现部分自动化：

- DRC 规则检查——提供有关 HDL 编码和 XDC 约束方面的指南，从而可帮助工程师在设计流程早期阶段提高设计质量，避免问题影响到后续各个阶段导致设计反复带来更高的成本。
- 经过验证的模板——针对特定 HDL 编码和 XDC 约束，实现“生成即保证最佳”的代码。

这些 DRC 规则检查和 HDL/XDC 模板是整个 UltraFast 设计方法的一部分，从某种意义上来说，有一个明确的DRC和准则，遵循Vivado Design Suite 提供的模板进行操作，有利于用户按时完成设计工作。

设计方法检查表

要全面发挥 UltraFast 设计方法的作用，应将本指南与设计方法检查表结合使用。该检查表包含从规划到所有后续设计阶段等整个设计流程中要考虑的常见问题及建议采取的行为。检查表中重点列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。

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

检查表中列出的大多数问题都可以链接到本指南或其它赛灵思技术文档中的相关内容。这些参考资料可指导您如何解决因这些问题而产生的设计问题。

Vivado Design Suite 配套提供 Documentation Navigator （参见：使用 Documentation Navigator）。您可用文档导航器 Documentation Navigator v2013.4 或其更高版本访问检查表功能。在 Documentation Navigator 中可通过以下步骤开始使用设计方法检查表：

1. 点击“Design Hub View”标签。
2. 在左侧菜单顶部点击“Create Design Checklist”。
3. 在新设计检查表对话框中输入信息并点击“OK”。
4. 打开新的检查表。检查表顶部上的各个标签 （见：图1-1）提供了导航功能。 “Title Page”标签提供了使用检查表的基本信息。您也可点击其它标签查看检查表相关问题及指南。

图 1-1 : Documentation Navigator 中的“Design Methodology Checklist”标签

此外， UltraFast 设计方法检查表还提供了电子数据表版本。

设计流程

图1-2 显示了设计流程的各个步骤。这些步骤在时间上往往是重叠的。有时进程可能返回到前一步，从而形成设计反复。

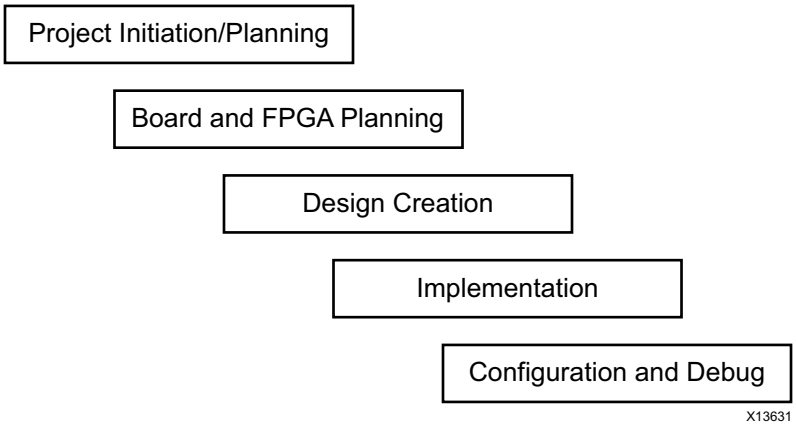


图 1-2 : 设计流程中的各个步骤

该检查表和本方法指南按设计阶段进行编写。当进入每一个设计阶段时，赛灵思建议审查检查表中的对应标签和本方法指南中的章节。

本指南介绍了从设计初期阶段适当监控设计预算 （如占位面积、功耗、时序等）以及纠正设计的重要性。在进入实现阶段之前，需高度重视为设计创建正确的时序约束。由于 Vivado 工具从头到尾整个过程使用基于时序的算法，因此设计从设计流程一开始就要得到正确的约束。

指定正确的时序，要求分析设计中每个主时钟及其有关生成时钟之间的关系。与 ISE (UCF) 不同的是，在 Vivado 工具 (XDC) 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。时序分析应在综合之后进行，且必须在进入下一设计阶段之前确保在每个实现阶段中在正确的约束下满足时序要求。

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。另外，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。图1-3 给出了这种高级方法的一些细节。

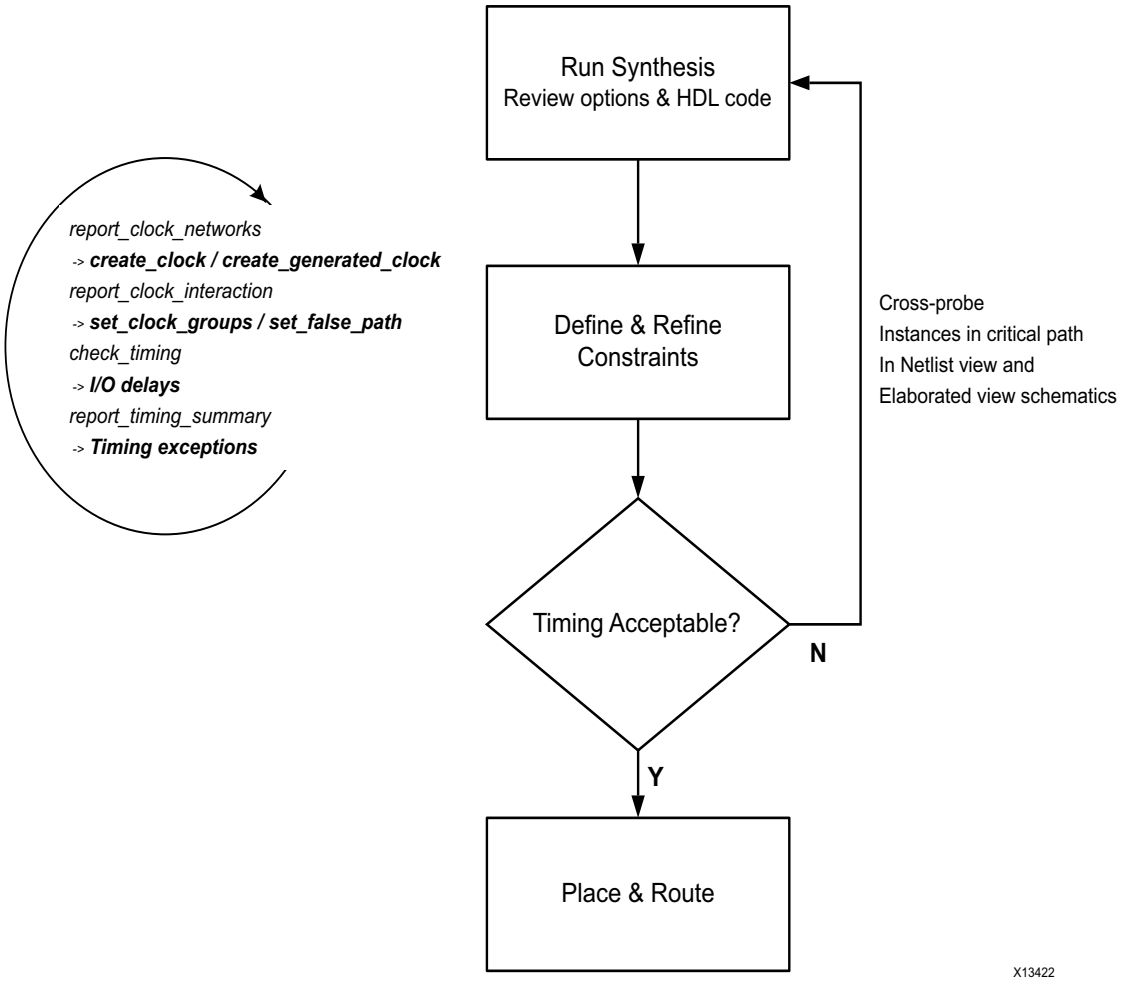


图 1-3 : 实现快速收敛的设计方法

如能够通过正时序裕量 (positive slack) （或相对较小的负时序裕量 (negative slack)）满足设计目标，那么设计流程的综合部分可视为完成。例如，如果综合后未能满足 （或未接近满足）时序要求，那么布局布线结果也不太可能满足时序要求。不过，我们仍可以往前继续执行其它流程。如果实现工具能为伪路径分配最佳资源，那么它们有时也有可能收敛时序。即便时序未得到满足，您仍能更准确地掌握负时序裕量。更准确地掌握实现后负时序裕量，有助于判断使用改进后的 HDL 和约束重新综合时，综合后最差负时序裕量 (WNS) 应改善的程度。

从概念上讲，设计流程早期阶段 （C、 C++ 和 HDL 综合）对设计性能、密度和功耗的影响要远远高于后续阶段 （如图 1-4 所示）。

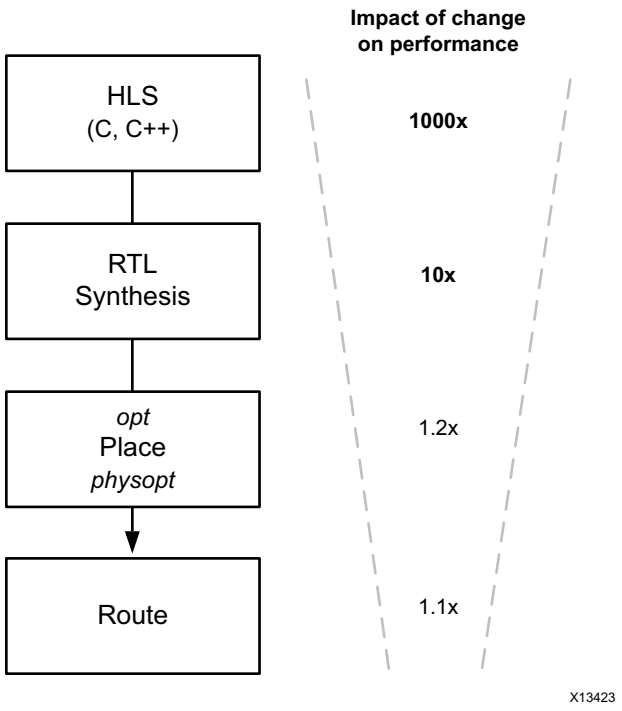


图 1-4 : 整个流程中设计修改的影响

相应的，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及其输入 （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。

因为确保从一开始就正确而且从早期阶段就关注设计目标非常重要，本指南还提供了有关RTL、时钟、引脚、 PCB 规划方面的指南。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。

快速验证

本指南介绍了系统架构和微架构选择各个具体方面的快速验证概念。这个概念可应用于两种不同环境。

在系统设计环境下， I/O 带宽进行系统内验证，这一步甚至在实现核心设计之前就要进行。如需了解更多信息，敬请参阅：第 3 章中的单板和器件规划。这个步骤凸显了在 I/O 最终确定之前可能需要修改系统架构和接口选择。

在设计实现的环境中，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。如需了解更多信息，敬请参阅：第 5 章中的设计基准 (baseline)。在进入实现阶段前，明确这一过程是否需要修改 RTL 微架构选择。

作为建立良好设计方法的一部分，到底如何与Vivado Design Suite 进行交互非常重要。 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。第 2 章：使用 Vivado Design Suite 探讨 Vivado 工具支持的不同使用模型。这将有助于您确定使用什么样的模型。后续各章将帮助您了解以下相关方法和技术各方面的更多详情：

- 时序约束定义与验证
- 器件中的 I/O 和时钟规划
- 选择和配置 IP
- 创建 IP 子系统

- 封装定制 IP
- 逻辑仿真
- 设计规则检查 (DRC)
- 功耗分析与优化
- 时序收敛流程
- 硬件验证 （调试核心插入与配置）

建议： 遵循本指南所提供的设计方法建议，可最大限度地发挥赛灵思器件优势，同时消耗最少的的时间和精力。

访问技术文档和培训资料

在适当的时间获得正确的信息,对于及时完成设计并确保整体设计成功而言十分重要。参考手册、用户指南、教程和视频能够帮助您尽快掌握 Vivado Design Suite。本节为您列出了部分技术文档和培训资料的来源。

使用 Documentation Navigator

Vivado Design Suite 配套提供“Xilinx Documentation Navigator” （图1-5），用于访问和管理全套赛灵思软/硬件文档、培训资料和辅助材料。借助 Documentation Navigator,您可查看赛灵思最新及过去的技术文档。通过版本、文档类型或设计任务来过滤技术文档显示内容。结合搜索功能可帮助您快速找到正确的信息。 Methodology Guide 是技术文档类型下的过滤器之一，借助该过滤器,您几乎可以在瞬间找到任何的 Methodology Guide。

Documentation Navigator 会扫描赛灵思网站，以检测并提供技术文档更新。 “Update Catalog”功能可提醒您有可用的更新内容，并提供有关文档的具体信息。赛灵思建议您在出现提醒时要更新目录，以使其保持最新。您可以为指定的文档建立本地技术文档目录并对其进行管理。

Documentation Navigator 中有一个“Design Hub View”标签。 “Design Hub”是指与设计活动 （如应用设计约束、综合与实现，以及编程和调试等）相关的文档集。文档和视频被纳入每个设计中心内，以简化相关领域的学习过程。每个设计中心均包含“Getting Started” （快速入门）部分、含有相关流程常见问题 (FAQ) 的“Support Resources” （辅助性资料）部分，以及“Additional Learning Material （更多学习资料） ”。 “Getting Started”部分可为新用户提供清晰的入门指导。对已经熟悉该流程的用户来说， “Key Concept”和“FAQ”部分可能是他们比较感兴趣的内容，有助于他们获得专业知识。

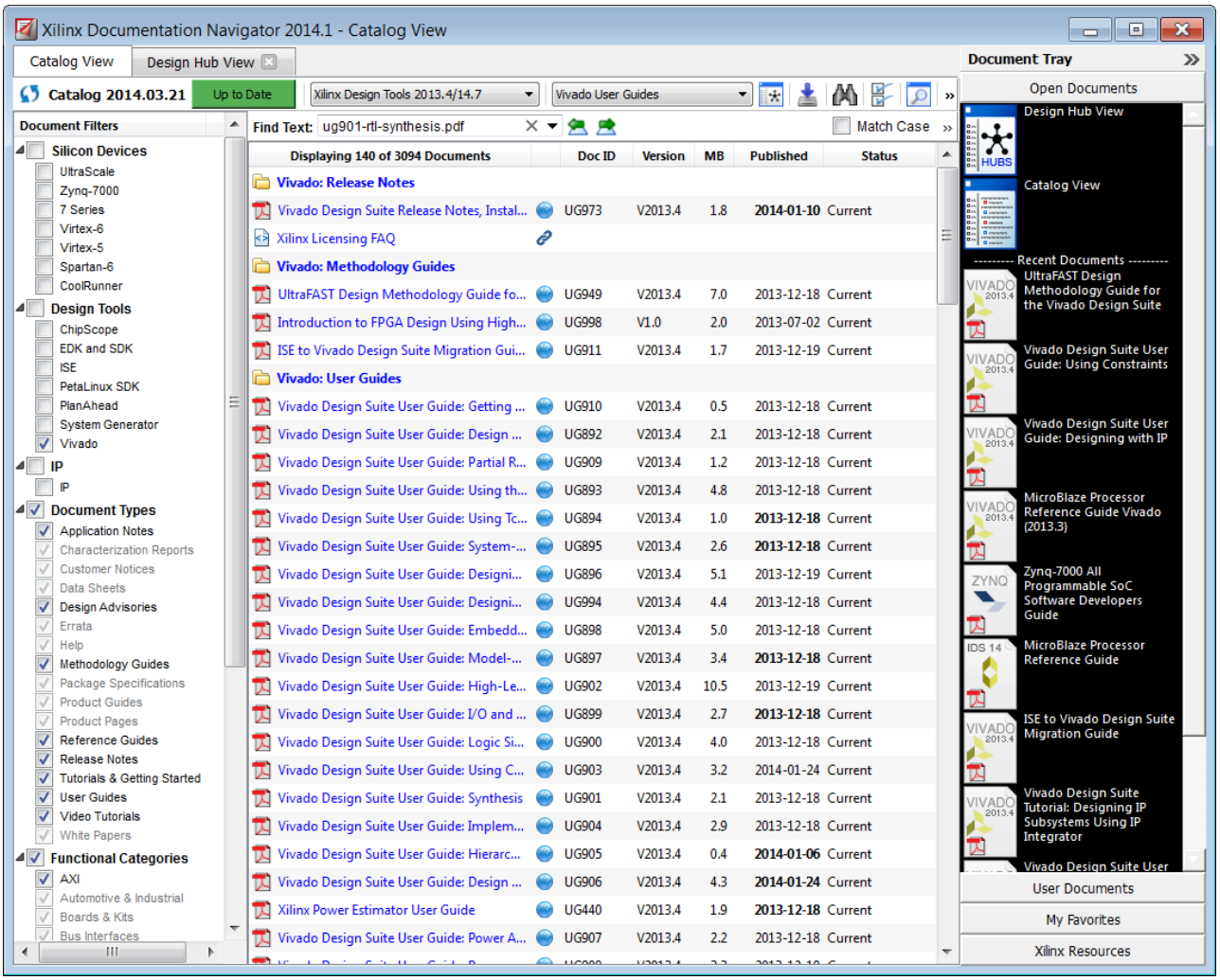


图 1-5 : Xilinx Documentation Navigator “Catalog” 视图

访问QuickTake 视频教程

赛灵思 QuickTake 视频教程为使用Vivado Design Suite 功能提供了指南。这些教程简明扼要，可在赛灵思网站 Vivado Design Suite 视频教程页面上观看，也可在赛灵思赛灵思优酷频道上观看，还可从本地下载。

提示： 如果连接速度干扰观看质量，可将视频下载到本地观看。 QuickTake 视频教程也可通过 Documentation Navigator 获得，参见：图1-6。

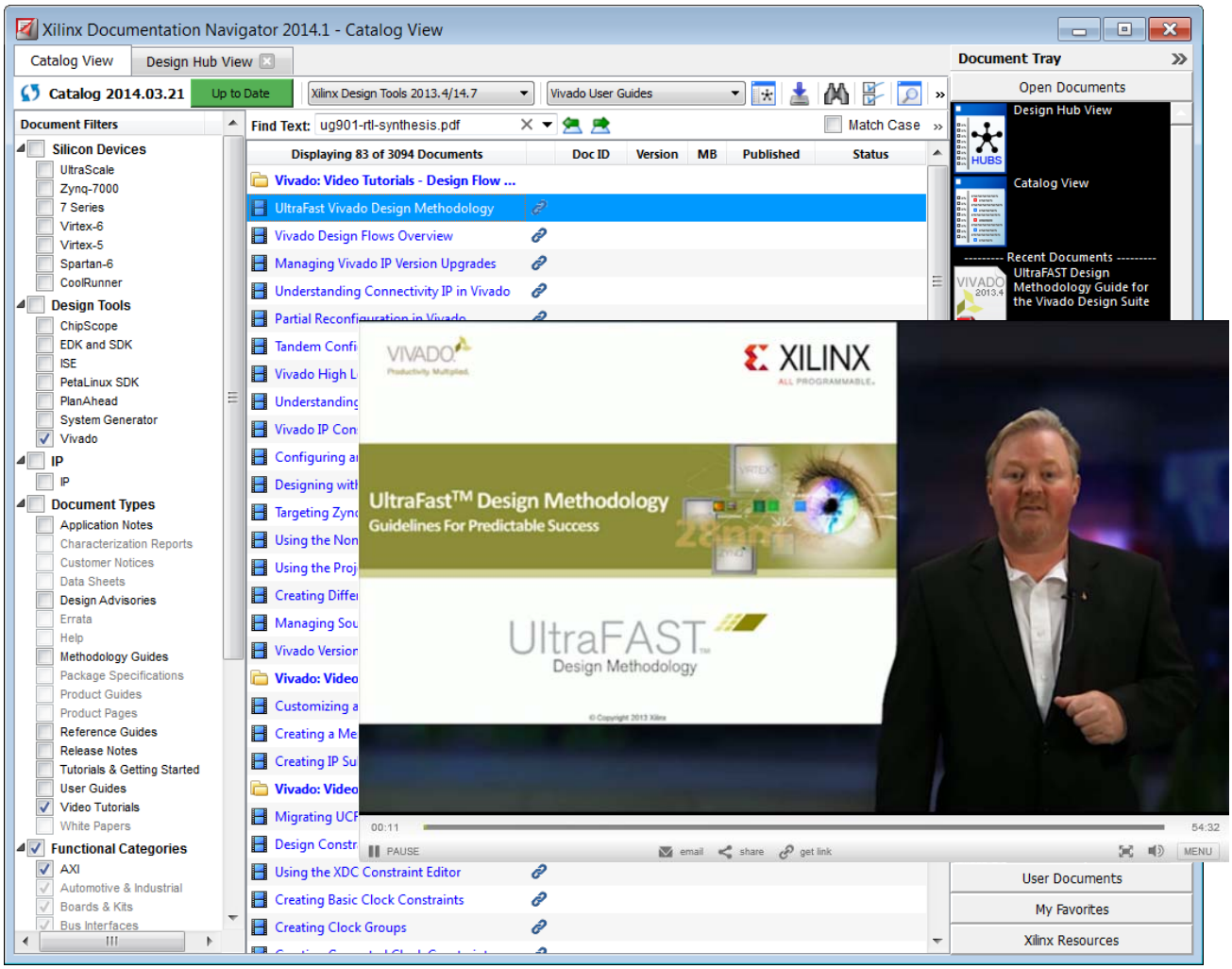


图 1-6 : 使用 Documentation Navigator 访问 QuickTake 视频教程

除了 QuickTake 视频以外，赛灵思还提供了带有实例的书面教程。这些教程中包含实例设计和执行具体设计任务的分步指导。Vivado Design Suite 教程还可通过赛灵思网站或技术 Documentation Navigator 获得。此外，您还可以注册由赛灵思或其合作伙伴提供的培训课程。

第 2 章

使用 Vivado Design Suite

Vivado Design Suite 简介

本章节将详细介绍使用 Vivado® Design Suite 时的各种使用模型、特性和选项。这包括准备和管理设计源文件和 IP。关于配置和运行实现工具以及开展设计分析的详细介绍，敬请参阅：第 4 章：设计创建和第 5 章：实现。

您可将 Vivado Design Suite 用于各种不同类型的设计，例如层级RTL设计、基于网表的设计或 I/O 规划设计。根据不同设计类型，所涉及的工具流程和特性会有所不同。本文档详细介绍基于 HDL 的 FPGA 设计流程。该流程中 RTL 源文件、 IP 核或第三方综合网表通过实现步骤编译而成，其结果随后可用于编程和调试 FPGA 器件。

以下文档和视频教程提供关于 Vivado Design Suite 流程的更多信息：

- Vivado Design Suite QuickTake 视频：Vivado 设计流程简介
- 《Vivado Design Suite 用户指南：设计流程简介》(UG892) [参照 5]
- 《Vivado Design Suite 教程：设计流程简介》(UG898) [参照 29]
- 赛灵思培训视频：UltraFast™ Vivado 设计方法

下图显示了Vivado Design Suite 设计流程。

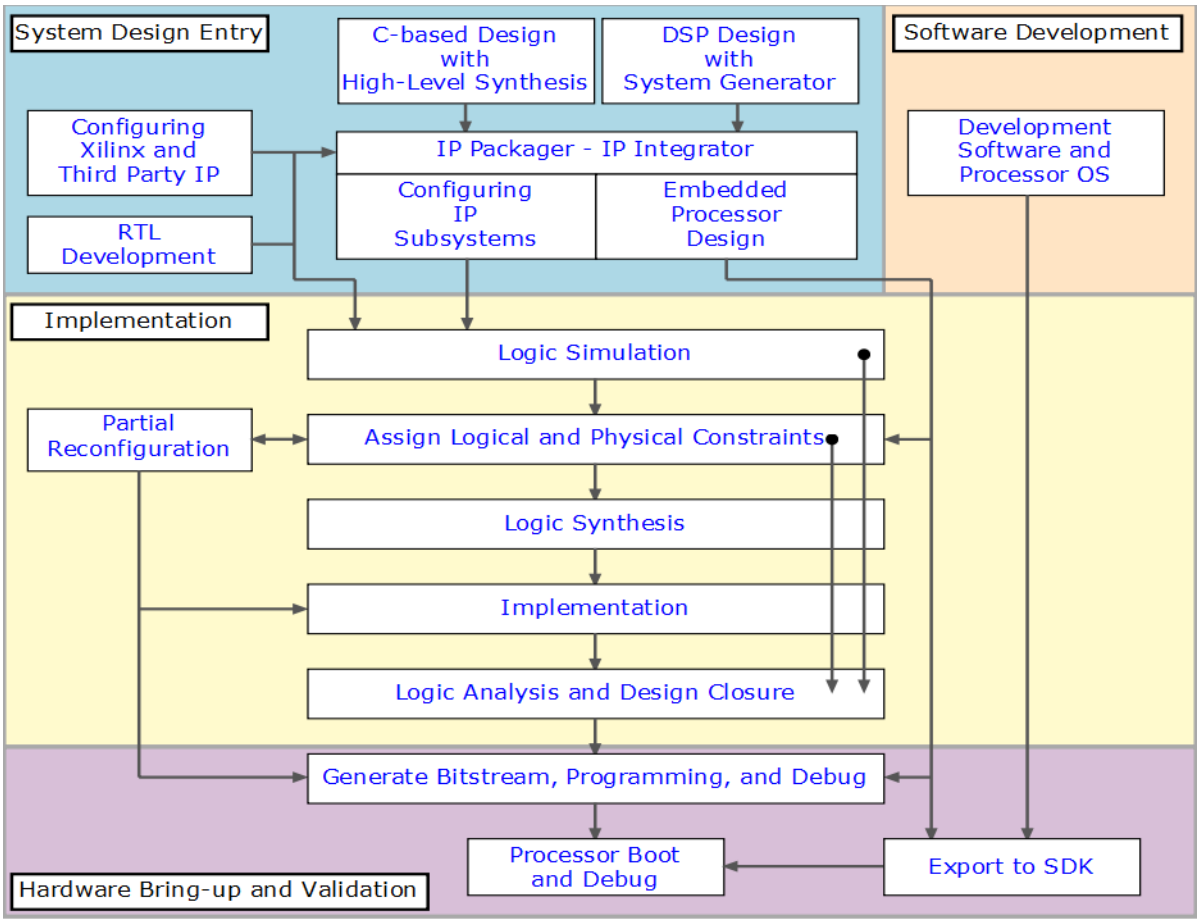


图 2-1 : Vivado Design Suite 流程

RTL 到比特流设计流程

您可指定供创建工程使用的 RTL 源文件，然后使用这些源文件进行 RTL 代码开发、分析、综合和实现。 Vivado 的综合与实现功能支持多种源文件类型，包括 Verilog、VHDL、SystemVerilog 和 XDC。本文重点介绍用于定义层级 RTL 源文件和赛灵思 设计约束 (XDC) 的正确编码与设计技巧，以及介绍如何使用 Vivado Design Suite 的特定特性和提升编程设计性能的技术。关于创建并处理 RTL 工程，敬请参阅：《Vivado Design Suite 用户指南：系统级设计输入》(UG895) [参照 8] 中的链接 。

此外， Vivado Design Suite 也支持使用包括 EDIF 或结构化 Verilog 在内的第三方综合网表。但是，来自 Vivado IP 目录的 IP 核必须使用 Vivado 综合功能加以综合，不支持使用第三方综合工具进行综合。不过也有例外情况，比如用于7系列器件的存储器接口生成器(MIG)核则可以使用第三方综合工具进行综合。如需了解更多信息，敬请参阅具体 IP 数据手册。

默认情况下 Vivado Design Suite 使用无关联 (OOC) 设计流程来综合来自赛灵思 IP 目录的 IP 核，以及来自 Vivado IP 集成器的模块设计。此外,您也可选择将层级 RTL 设计的特定模块综合为无关联 (OOC) 模块。这种无关联 (OOC) 流程便于您在与顶层设计无关联或独立于顶层设计的情况下综合、实现和分析层级设计、 IP 核或模块设计的设计模块。对支持层级团队设计、综合并实现 IP 与 IP 子系统 以及管理大型复杂设计的模块而言，这种无关联流程 (OOC) 是一种高效的方法。有关无关联设计流程 (OOC) 的更多信息，敬请参阅：《Vivado Design Suite 用户指南：设计流程简介》(UG892) [参照 5]。

Vivado Design Suite 能够在设计流程的各个阶段进行设计的行为、功能和时序仿真。您也可以使用能够集成到 Vivado IDE 中并能从 Vivado IDE 启动的第三方仿真器。如需了解更多信息,敬请参阅：《Vivado Design Suite 用户指南：逻辑仿真》(UG900) [参照 11] 。

在综合网表可用时， Vivado 实现方案可提供优化、布局布线网表到目标器件可用器件资源上所需的全部必要特性。Vivado 实现方案旨在满足设计的逻辑、物理及时序约束要求。在设计实现之后，您可使用各种时序和功耗分析功能来分析实现结果，并通过在 Vivado 硬件管理器中编程赛灵思器件和调试设计，使之在硬件中运行。

本 Vivado Design Suite 还支持下列章节介绍的其他设计流程。这些流程中的每一个都是由 RTL 到比特流设计流程演变而来，因此本文介绍的实现与分析技术均适用于其他设计流程和设计类型。

嵌入式处理器设计流程

创建嵌入式处理器设计时所用的工具流程略有不同。由于处理器需要用软件来高效启动和运行，因此软件设计流程必须与硬件设计流程协调一致。硬件流程和软件流程之间的数据切换以及跨越这两个域的验证工作是确保设计成功的关键。

创建嵌入式处理器硬件设计会用到 Vivado Design Suite 的Vivado IP 集成器功能。您可在 IP 集成器环境中对处理器内核及其接口进行实例化、配置和组装。该 IP 集成器能执行基于规则的连接功能，并提供设计辅助。在硬件设计通过实现完成编译后,会被导出到赛灵思软件开发套件 (SDK) 中，用于软件开发与验证。仿真与调试功能使您可以跨两个域对设计进行仿真和验证。

以下资料介绍了嵌入式处理器设计流程：

- 《Vivado Design Suite 用户指南：嵌入式处理器硬件设计》(UG898) [参照 10]
- 《Vivado Design Suite 教程：嵌入式处理器硬件设计》(UG940) [参照 32]
- 《UltraFast 嵌入式设计方法指南》(UG1046) [参照 34]
- Vivado Design Suite QuickTake 视频：使用 Vivado IP 集成器进行设计
- Vivado Design Suite QuickTake 视频：针对 Zynq 使用 Vivado IP 集成器

基于 C 语言的高层次综合流程

Vivado Design Suite 中的基于 C 语言的高层次综合 (HLS) 工具允许您使用 C、C++、System C 和 OpenCL™ API 语言描述设计中的各种 DSP 功能。语言描述设计中的各种 DSP 功能。可以使用较高级的语言对算法描述、数据类型、规格描述等内容进行抽象。您可以使用不同参数创建"假设"场景，以优化设计性能和器件占位面积。

使用 HLS，您能够借助基于 C 语言的测试平台仿真直接从 RTL 设计环境生成的 RTL。HLS 会自动使用多个优化的代码库,并通过 math.h 支持浮点算法。C 到 RTL (C-to-RTL) 综合能将基于 C 语言的设计转换成 RTL 模块，打包和实现为更大型 RTL 设计的组成部分，或者实例化成 IP 集成器的模块设计。

以下资料介绍了 HLS 工具流程与特性：

- 《Vivado Design Suite 用户指南：高层次综合》(UG902) [参照 17]
- 《Vivado Design Suite 教程：高层次综合》(UG871) [参照 28]
- 有关 Vivado 高层次综合的视频资料，敬请访问赛灵思网站 Vivado Design Suite 视频教程页面。

部分重配置设计流程

部分重配置可以让运行中的赛灵思器件的组成部分实时进行重配置，修改正在运行的设计的特性与功能。必须合理规划可重配置模块，确保其按预期运行，实现最高性能。部分重配置流程对设计流程要求相当严格，确保设计出合理的可重配置模块，以便在部分比特流升级过程中实现无故障运行。这包括减少可重复配置模块的接口信号数量、器件资源布局规划和引脚布局，以及符合特定的部分重配置 DRC。这种器件编程方法必须合理规划，以确保配置 I/O 引脚得到合理分配。

以下资料介绍了部分重配置工具流程与特性：

- 《Vivado Design Suite 用户指南：部分重配置》(UG909) [参照 25]
- 《Vivado Design Suite 教程：部分重配置》(UG947) [参照 33]

- Vivado Design Suite QuickTake 视频：Vivado Design Suite 中的部分重配置功能

Vivado Design Suite 使用模型

正如 Vivado Design Suite 能支持大量不同的设计流程一样，这些工具也能够根据您期望的设计管理方式和与 Vivado 工具交互的方式支持多种不同的使用模型。本章节将协助并指导您为实现与 Vivado 工具交互而选用使用模型时必须做出的一些决策。

部分此类决策包括：

- 您主要使用脚本或命令，还是倾向于使用图形用户界面 (GUI) 工作？
- 您希望把 IP 核配置用于单个设计工程，还是建立远程资源库以便在多个工程中重复利用配置好的 IP 核？
- 您希望通过使用工程结构让 Vivado Design Suite 管理设计的源文件、状态和结果，还是要自己来快速创建并管理设计？
- 您是否在版本控制系统内部管理自己的源文件？
- 您是否使用第三方工具进行综合或仿真？

建议： 在使用 Vivado 工具开始您的首个 FPGA 设计之前，敬请参阅：《Vivado Design Suite 用户指南：着手设计》(UG910) [参照 12] 以及 《Vivado Design Suite 用户指南：设计流程简介》(UG892) [参照 5]。

理解工程与非工程软件使用模式

Vivado Design Suite 有两种主要的使用模型：工程模式和非工程模式。两种使用模型都能通过 Vivado IDE 运行，但 Vivado IDE 能为工程模式提供众多优势。两种使用模型都能通过 Tcl 命令或脚本运行，但 Tcl 命令是运行非工程模式的最简单方式。

注释： 您可使用基于 Tcl 脚本的流程，也可在需要时使用 Vivado IDE 来创建物理约束和时序约束，或是在设计分析过程中查看结果。

使用工程模式

在工程模式下， Vivado 工具能自动管理自己的设计流和设计数据。使用工程模式的主要优势在于利用按钮实现方案完成设计流程的自动化。

提示： 工程模式的主要优势在于Vivado Design Suite 可管理整个设计流程，包括工程文件管理、报告生成、数据存储等。

工程可跟踪并报告设计状态、源文件依赖性和实现结果等。在工程模式下工作时， Vivado Design Suite 会在磁盘上创建一个目录结构，用于在本地或远程管理设计源文件以及管理源文件的修改与更新。工程基础架构也用于管理自动综合与实现运行、跟踪运行状态、存储综合与实现结果及报告。例如：

- 如果您在综合后修改 HDL 源文件， Vivado Design Suite 会确认当前结果为过期结果，提示您重新综合。
- 如果您要修改设计约束， Vivado 工具会提示您重新综合或重新实现，或者两者兼有之。
- 在布线完成后， Vivado 工具将自动生成时序报告和功耗报告。
- Vivado IDE 中可一键式运行整个设计流程。

重要提示： 部分操作系统 （例如 Microsoft Windows）限制了可用于文件路径和文件名称的字符数量 （比如最多 256 个）。如果您的操作系统有这样的限制，赛灵思建议您在离驱动器根目录较近的位置创建工程，以量缩短文件路径和名称。

使用非工程模式

在非工程模式下，您自己使用 Tcl 脚本管理设计源文件和设计流程。主要优势在于您能完全掌控流程的每个步骤。您可随意生成设计检查点和报告。每个实现步骤可进行定制，以满足具体的设计挑战。此外，您还可在每个设计步骤后分析结果。设计源文件一般通过他们的当前位置访问，比如版本控制系统，而不是将它们拷贝到本地工程目录结构下。随着设计流程的推进，设计结果保存在已分配给 Vivado Design Suite 的内存中。换言之，所有设计都存储在内存中并在设计流程中实时更新。

提示： 在非工程模式下，会在内存中创建工程结构以对设计进行处理，但工程未被写入磁盘。

在非工程模式下，每个设计步骤都使用 Tcl 命令控制，例如：

- 如果您在综合后修订 HDL 文件，您必须记得要再次运行综合，以更新内存中的网表。
- 如果您需要布线后的时序报告，那么您必须在布线完成时明确要求生成时序报告。
- 使用 Tcl 命令来设置“Design Parameters”和“Implementation”选项。
- 您可使用 Tcl 在设计过程的任何阶段保存设计检查点 (DCP) 并生成报告。

随着设计推进，您必须创建报告或设计检查点，以保存内存设计。设计检查点 (DCP) 是指一个准确表达内存设计结果的文件。您可于设计流程每个步骤后保存设计检查点，比如综合后、优化后、布局后。 DCP 文件可读回到 Vivado Design Suite 中，以将设计恢复到检查点文件中所捕获的状态。

您还可以在 Vivado IDE 中打开一个 DCP，用于开展交互约束分配和设计分析。由于您查看的是存储器中的动态设计，因此任何设计修改都会在设计流程中自动向前传递。您也可以将更新内容保存到新的约束文件或设计检查点中供以后运行。

虽然工程模式提供大多数非工程模式特性，但非工程模式并不提供部分工程模式特性。这些特性包括源文件和运行结果管理，保存设计和工具配置、设计状态以及 IP 集成。另一方面，您可以使用非工程模式跳过某些流程，从而减少设计的内存空间占用，节省与工程有关的磁盘空间。

使用Vivado 集成设计环境 (IDE)

在工程模式和非工程模式下均可使用 Vivado 集成设计环境 (IDE)。 IDE 中显示的功能因调用 IDE 的方式和时间而异。如需了解有关 Vivado IDE 的更多信息，敬请参阅：《Vivado Design Suite 用户指南：如何使用 Vivado IDE》(UG893) [参照 6]。

Vivado IDE 通过打开内存中的设计，可在整个设计流程中开展分析和分配约束。打开设计时，允许在设计流程的特定阶段加载设计网表，向设计分配约束，并将设计应用于目标器件。这个流程便于您在每一个设计阶段查看设计情况并与设计进行交互，如图2-2 所示。

在使用工程模式时， Vivado IDE 提供名为 Flow Navigator 的界面，用于汇编、实现并验证您的设计和 IP。此外，Vivado IDE 还支持可管理所有设计源文件、配置和结果的按钮式设计流程。

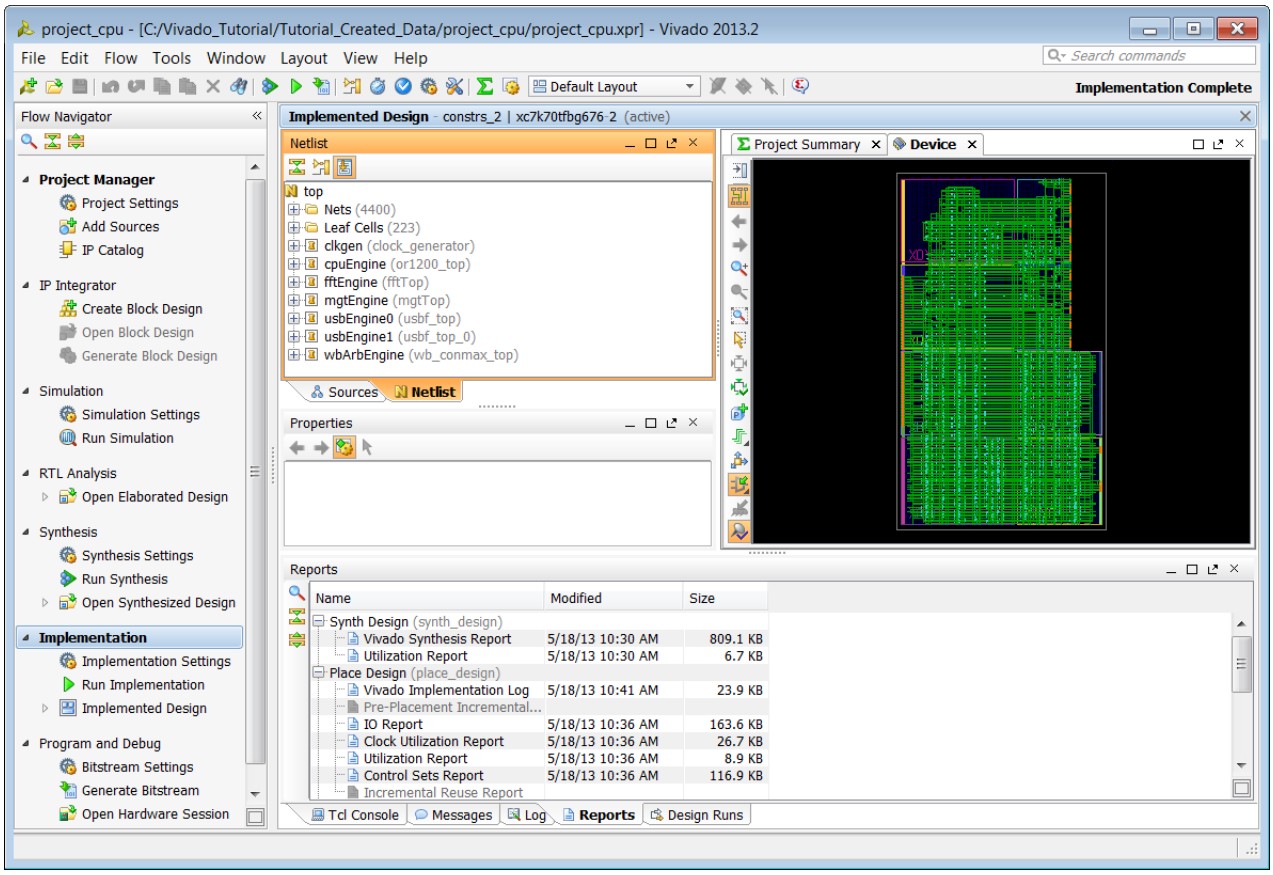


图 2-2 : 在 Vivado IDE 中打开实现后的设计

如需了解有关 Vivado IDE 的更多信息，敬请参阅：《Vivado Design Suite 用户指南：如何使用 Vivado IDE》(UG893) [参照 6]。

使用 Tcl

所有支持的设计流程和使用模型都可用 Tcl 命令来运行。您既可使用单独的 Tcl 命令，也可使用保存的 Tcl 脚本命令。您可使用 Tcl 脚本来运行包含设计分析及设计报告在内的整个设计流程，或者设计流程的一部分 （比如设计创建和设计综合）。

如果您倾向于直接使用 Tcl 命令，那么您可通过 Vivado Design Suite 的Tcl Shell 并借助 Vivado IDE 提供的 Tcl 控制台来与自己的设计进行交互。如需了解有关 Tcl 和 Tcl 脚本使用方面的更多信息，敬请参阅：《Vivado Design Suite 用户指南：如何使用 Tcl 脚本》(UG894) [参照 7]。对于介绍如何在 Vivado 工具中使用 Tcl 的入门级教程，敬请参阅：《Vivado Design Suite 教程：设计流程简介》(UG898) [参照 29]。

当使用 Tcl 时，您仍可利用 Vivado IDE 的交互式 GUI 分析和约束定义功能。使用 Tcl 脚本，您可以在综合后随时保存设计检查点，供以后在 Vivado IDE 中打开检查点。

配置 IP

Vivado IDE 提供以 IP 为中心的设计流程，便于您从各种设计源文件为自己的设计添加 IP 模块。Vivado IDE 还提供包含由赛灵思交付的即插即用型 IP 的可扩展 IP 目录。 IP 目录可通过添加下列内容进行扩展：

- System Generator for DSP 的设计模块 （Simulink® 算法中的 MATLAB®）。
- Vivado 高层次综合(HLS)设计 （C/C++ 算法）
- 第三方 IP
- 使用 Vivado IP 打包器封装为 IP 的设计

IP 目录为访问赛灵思IP 提供了统一便捷的途径，这些 IP 包括构建块、向导、连接、 DSP、嵌入式、 AXI 基础架构和视频 IP。不论正在开发的最终应用是什么，这些 IP 都通过统一的通用库提供。 IP 目录还列出了由赛灵思联盟合作伙伴提供的 IP 核，并提供许可信息的链接。

大部分通过 IP 目录提供的 IP 既可用于 RTL 工程，也可用于 IP 集成器模块设计。有少量 IP 只能在 IP 集成器或 RTL 设计中使用。当您试图把 IP 用于不支持的流程时，会向您发出警示消息。

Vivado IP 目录会在 IP 目录的许可栏下显示 IP 属于“Included”还是“Purchase”。下列定义适用于赛灵思提供的 IP：

- 已包含：赛灵思最终用户许可协议包含在赛灵思 Vivado Design Suite 软件工具范围内免费许可使用的赛灵思 LogiCORE™ IP 核。
- 需购买：内核许可协议适用于需付费的赛灵思 LogiCORE IP 核，内核评估许可协议则适用于需付费的赛灵思 LogiCORE IP 核的评估。

通过购买获得许可的 IP 可能拥有不同的功能。如需了解特定 IP 的许可信息，请在 IP 目录中选中该 IP，然后点击右键，便可单击“License Status”来查看。下列表格是需购买的 IP 的可能状态：

表 2-1 : 需购买 IP 的状态

状态	权利政策	描述
购买	完整	许可客户完全使用内核的功能
Hardware_evaluation	评估	许可客户使用内核的有限功能 （例如：硬件超时电路）
Design_Linking	仿真	许可客户仅将内核用于仿真目的

对工程中使用的 IP 核的许可状态信息，可单击“Tools > Report > Report IP Status”，报告 IP 状态。关于如何取得 IP 许可的更多信息，敬请参阅赛灵思网站上的赛灵思 IP 许可页面。

赛灵思及其合作伙伴还提供未作为默认 Vivado IP 目录组成部分的更多 IP 核。如需了解有关可用 IP 的更多信息，敬请参阅赛灵思网站上的 IP。

下列章节将探讨如何配置、使用和管理 IP 核。

使用 Vivado IP 目录

Vivado IDE 的 IP 目录功能是配置 IP 的最佳方法，其便于浏览、配置和生成输出结果以及验证 IP。利用 Vivado IDE 的 IP 目录和配置向导可简化整个配置过程。您也可以从 IP 目录中直接访问产品指南、修改日志和答复记录 （如适用）等 IP 文档。

这里有类似于 Tcl 的脚本命令可启用 IP 定制脚本编写，但并非所有用于 IP 配置的 Tcl 参数都记录在内。如果需要编写脚本，在用 IDE 配置 IP 和生成输出结果后，可使用 Vivado 日志文件创建脚本。

使用配置向导对 IP 进行定义后，会创建一个赛灵思内核实例 (`.xci`) 文件。该文件包含了面向 IP 的所有定制选项。工具可从这个文件生成面向 IP 的所有输出结果。这些输出结果包含用于综合和仿真的 HDL、约束，也有可能是测试平台、 C 模块以及实例设计等。该工具可根据所用的定制选项创建这些文件。

由于 IP 是针对特定的逻辑器件生成的，因此用描述性的名称来命名 IP 便于以后识别。

如需了解更多信息，敬请参阅：

- 升级 IP
- 管理 IP （介绍用于存储和使用 IP 配置的可用选项）

- 《Vivado Design Suite 用户指南：采用 IP 进行设计》(UG896) [参照 9]

生成 IP 输出结果

创建 IP 输出结果，这样综合、仿真和实现工具就可以使用特定 IP 配置。在生成输出结果时，建立目录结构用以存储各种 IP 输出结果。文件夹和文件要一目了然，并保持完整。

网表选项

您在生成输出结果时，可选择为 IP 创建的数据层级。选项的选择会响应设计的实现方式。

- 使用默认设置“Out-of-Context”(OOC)，在生成输出结果的过程中将只在特定的 IP 定制版本上执行逻辑综合，以创建综合设计检查点文件 (`.dcp`)。之后此网表与 IP 提供的约束一同用于实现过程。 IP 自身的综合被称为无关联 (OOC) 综合。

  在将第三方综合工具用于设计其余部分时，也要用到 OOC 方法，同时会为它们创建一个模块存根文件，以便为 IP 调用一个黑盒。还会同时生成 Verilog 版本和 VHDL 版本的功能仿真模型。如果您没有混合语言仿真工具，则可使用该功能网表。在使用 Vivado Design Suite 启动第三方仿真器时，应根据情况适当使用 HDL 或功能网表。

- 单击“Global”将为该 IP 生成 RTL 和 XDC IP 源文件。这些源文件将随同其余的设计源文件在设计的顶层综合与实现过程中使用。

提示： 您还可以将 OOC 综合结果用于实现步骤 （进行模块分析或保存时序）。实现 OOC 模块需要更多约束 （例如：HD.CLK_SRC）来确保精确的时序结果。如需了解更多信息，敬请参阅：《Vivado Design Suite 用户指南：层级设计》(UG905) [参照 20]。

赛灵思建议针对用户定义的每个 IP，您应当生成所有可用的输出结果，包括综合设计检查点。这样做可以提供完整的 IP 展示，并可将其进行存档或存放在版本控制系统中。如果未来 Vivado Design Suite 版本不包含该 IP，或者 IP 发生了不应有的变化 （例如接口变化），这时您会获得仿真所需的所有输出结果，并可利用未来的 Vivado 软件版本进行综合与实现。

IP 约束

许多 IP 核都包含综合与实现过程中所使用的 XDC 约束。如果通过定制过程中创建的 XCI 来使用 IP，那么不论是工程模式还是非工程模式都会自动使用这些约束。手动修改 IP 约束使其可用于顶层，这项工作非常繁琐且容易出错。

很多 IP 核都参考约束中的输入时钟。这些时钟可以来自顶层用户，甚至可来自设计中的其它 IP 核。默认情况下，Vivado 工具会在早期阶段处理任何 IP 时钟创建以及任何自定义顶层时钟创建工作。该过程会将时钟提供给需要它们的 IP 核。

验证 IP

Vivado IP 目录中许多赛灵思 IP 均提供实例设计。如下图所示，您可从 Manage IP 或者 RTL 工程的 IP 源文件区域选择 IP，并查看“Open IP Example Design”是否可选，以判断该 IP 是否提供实例设计。这一过程同样可通过 Tcl 实现。

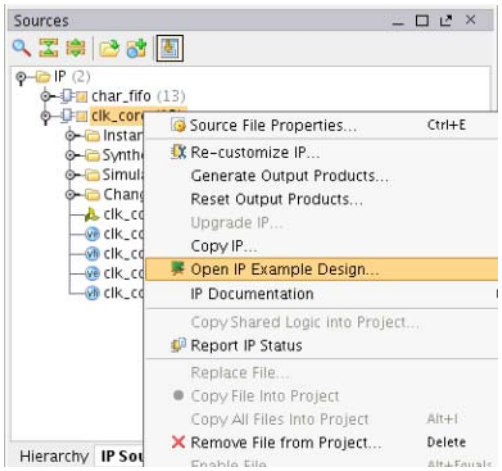


图 2-3 : 打开实例设计

在打开实例设计时，会使用自定义 IP 创建一个新工程。您可以参考实例设计，了解 IP 自定义的正确使用与连接方法。

某些 IP 随实例设计提供测试平台，您可使用测试平台验证定制 IP 的功能。您可以运行行为仿真、综合后仿真或实现后仿真。您可以运行功能仿真或时序仿真。如果要执行时序/功能仿真，您需先综合/实现实例设计。

如需了解有关仿真 IP 的具体信息，敬请参考 IP 的产品指南。如需了解有关仿真的更多详情，敬请参阅：《Vivado Design Suite 用户指南：逻辑仿真》(UG900) [参照 11]。如需了解有关使用实例设计和 IP 输出结果的更多详情，敬请参阅：《Vivado Design Suite 用户指南：采用 IP 进行设计》(UG896) [参照 9]。

如果您的设计中有存储器接口生成器(MIG) IP，敬请参阅如下资料：

- 如需了解有关仿真的详情，敬请参阅：《基于 LogiCORE IP UltraScale 架构的 FPGA 存储器接口解决方案产品指南》 (PG150) [参照 45]
- 如需了解 MicroBlaze™ 设计的 MIG 仿真实例，敬请参阅：《参考系统：使用 IP 集成器的 Kintex-7 MicroBlaze 系统仿真》 (XAPP1180) [参照 43]。

使用存储器接口生成器 (MIG) IP

使用赛灵思存储器 IP 时需要更多的 I/O 引脚规划步骤。在完成 IP 定制后，您可将顶层 I/O 端口分配给 Vivado IDE 中的细化设计或综合设计中的物理封装引脚。

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。另外还为您提供 Memory Bank/Byte Planner，协助您将 Memory I/O 引脚组分配给物理器件引脚上的字节通道。

如需了解更多信息，敬请访问：《Vivado Design Suite 用户指南：I/O 和时钟规划 (UG899) [参照 4] 中的链接。

使用 IP 集成器创建 IP 子系统

本章节将探讨如何创建和管理 IP 子系统。

Vivado IP 子系统

Vivado Design Suite 的 IP 集成器功能可用来创建模块设计 (.bd)。这些模块设计本质上是包含任意数量用户配置 IP 及 IP 之间连接的 IP 子系统。如需了解有关打包和使用自定义 IP 以及 AXI 接口的信息，敬请参阅：封装定制 IP 和 IP 子系

统和创建定制外设。 IP 集成器是用于连接 IP 核的接口，以创建特定领域子系统和设计，包括采用 Zynq®-7000 All Programmable (AP) SoC 和 MicroBlaze 处理器的嵌入式处理器设计。它能够实例化来自 Vivado HLS 的高层次综合模块、来自系统生成器的 DSP 模块以及使用 Package IP 功能提供的用户自定义 IP。

如需了解更多信息，敬请参阅以下资料：

- 《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：使用 Vivado IP 集成器进行设计

构建 IP 子系统

IP 子系统 的最佳配置方法是使用 Vivado IDE 的 IP 集成器。 IP 集成器的互动模块设计功能可简化 IP 的配置和组装工作。

如果需要编写脚本，您可以在使用 Vivado IDE 创建子系统后，使用 Vivado Design Suite 的日志文件。IP 集成器可以创建 Tcl 脚本，以便在内存中重新创建当前的模块设计。您还可以综合利用 GUI、 Tcl shell 命令和脚本来创建 IP 子系统。

IP 子系统 可进行配置，这样导入设计工程中时，可作为一组 HDL 源文件和约束文件 （默认），或作为包含综合网表和整个子系统约束的设计检查点文件 (.dcp)。

设计人员辅助

如需加快子系统或设计的创建，可使用 IP 集成器的模块自动化和连接自动化功能。模块自动化功可用来配置基于处理器的基本设计以及一些复杂的 IP 子系统，其中要用连接自动化功能为设计中的不同引脚或端口建立重复连接。 IP 集成器具有单板感知能力，目前可支持所有赛灵思评估板。这意味着如果您使用评估板作为目标硬件， IP 集成器可以识别出特定评估板上的所有组件，这样您可以使用连接自动化功能将设计的 I/O 端口连接到目标板上的已有组件。设计人员辅助还有助于建立时钟和复位连接。 “Signals”视图和“Board”视图等多种视图可协助您在模块设计中建立连接。使用设计人员辅助不仅有助于加快建立互连，而且可消除设计错误。

模块自动化

对于一些基于 IP 和处理器的复杂设计而言， IP 集成器可为其提供一种名为模块自动化的功能。这项功能使您能够利用常用的组件比较快速地组合基本处理器/IP 子系统。将用于嵌入式设计的基础构建块组合到一起之后，您就可以从 IP 目录中添加其它 IP 以对这一基础系统进行扩展。

连接自动化

在执行模块自动化功能构建出基础系统后，您需要把设计连接到外部 I/O 引脚。 IP 集成器的连接自动化功能不仅有助于建立与 I/O 引脚的连接，还有助于建立与设计本身的不同源文件的连接。结合使用单板感知功能，连接自动化功能有助于您将模块设计的端口连接到目标板上的外部组件，并为这些端口创建物理约束。

基于规则的连接检查

在设计的组装过程中， IP 集成器可实时运行基本的设计规则检查。但是，在设计创建中有些内容可能会出错。例如，时钟引脚频率可能被错误设置。该工具可通过运行设计验证来找出这类错误。设计验证可在模块设计方案上执行设计规则检查，并报告适用于该设计的警告和 （或）错误。然后，您可以从消息视图到方块图追踪警告和 （或）错误。赛灵思建议验证图形化设计以找出设计错误，否则只能到设计流程后期才能发现。

设计验证还可在模块设计方案上运行参数传递。参数传播功能可让 IP 根据自身与设计的连接情况自动更新参数设定。您可按照特定传播规则封装 IP，然后随着方块图的生成， IP 集成器会运行这些规则。

您可以单击“Tools” > “Validate Design”或者使用 Tcl 命令 `validate_bd_design` 来运行设计验证。 IP 集成器菜单还包含一个“Validate Design”图标。

使用平台单板流程

Vivado Design Suite 具有单板感知能力，能自动从内置的单板文件中导出 I/O 约束和 IP 配置数据。通过这些单板文件， Vivado Design Suite 得以知晓目标板上的各种组件并能够自定义并配置准备与特定板载组件相连的 IP。目前可支持部分 7 系列、 Zynq-7000 AP SoC 以及 UltraScale™ 器件单板。您可从合作伙伴网站上下载合作伙伴开发的单板的支持文件。

IP 集成器在“Board”视图这一独立的视图中显示了目标板上的所有组件接口。您可使用该视图选择所需的组件和 IP 集成器提供的“Designer Assistance”功能，从而轻松地将您的设计与您所选的板载组件连接在一起。接口可以单独拖放使用，也可以成组使用以便利用默认 IP 配置。当接口在 IP 集成器图示上提示可用时，双击该接口，使用互动方法即可添加带有 IP 选项和可配置项的接口。在使用该功能的过程中，会自动生成所有的 I/O 约束。

您也可为定制单板生成单板文件，然后把包含单板文件的库添加到工程中。如需了解有关生成定制单板文件的更多信息，敬请访问：《Vivado Design Suite 用户指南：系统级设计输入》(UG895) [参照 8] 中的链接。

创建分层 IP 子系统

IP 集成器可用来创建层级 IP 子系统。具有大量模块的设计适合使用该功能，否则很难在 GUI 菜单中进行管理。该工具支持层级架构的多个层次，方便您按设计功能将模块进行分组，从而保持设计在 IP 集成器菜单上的模块化和整洁性。

您还可以改变设计中不同对象的视觉效果。例如，可将“时钟”和“复位”标记为不同颜色。您还可以选择显示不同类型的连接，同时隐藏其他连接。

生成模块设计

在模块设计或 IP 子系统创建完成后，您就可以生成设计。生成的内容包括所有的源代码、 IP 核的必要约束、模块设计的结构网表。您可以右键点击“Sources”窗口中的模块设计并在弹出菜单中单击“Generate Output Products”来生成模块设计。使用 Vivado Design Suite Flow Navigator，您也可单击“IP Integrator > Generate Block Design”。等效的 Tcl 命令为：

```
generate_target all [get_files <path to the block design>]
```

这一步完成之后，设计就已经准备好集成到更高层次的 HDL 设计中，或进行综合与实现步骤。

使用针对模块设计的无关联 (OOC) 综合

分层设计流程能够将设计划分为可单独处理的更小、更好管理的模块。在 Vivado Design Suite 中，这些流程的基础是能够综合与设计其余部分无关联 (OOC) 的分区模块。

Vivao Design Suite 支持模块设计使用两种无关联 (OOC) 模式。顶层设计采用第三方综合流程时，单击“Out of context per Block design”。这是 IP 集成器附带的一个通用流程，用于创建设计设计检查点 （DCP）文件。如果作为较大型设计的组成部分使用，该模块设计无需在您每次修改设计其他部分的时候 （在 IP 集成器之外）重新再综合。

如果您想为实例化在模块设计上的每一个 IP 都创建一个 OOC 运行，可单击 “Out of context per IP”。该选项能为模块设计中的每一个 IP 均创建一个设计检查点文件。在该流程中，只为其配置因重新定制或参数传播造成的影响而发生改变的 IP 生成输出结果。该流程可缩短运行时间，适用于需要考虑运行时间的情况，尤其是在设计探索的早期阶段。可将 IP 高速缓存与本选项配套使用，防止 IP 未发生变化的情况下重新生成输出结果。

创建远程模块设计 （建议）

IP 集成器的重用特性允许您在 Vivado Design Suite 工程外创建可供多个团队重复使用的模块设计，从而方便开展团队设计工作。当您创建完成设计并将其置于版本控制下时，多个团队可以重复使用同一模块设计来创建多个工程。

要在远程位置建立模块设计，可在“Create Block Design”对话框下拉目录列表中指定所需位置。

验证 IP 子系统

在设计完成后，可从 IP 集成器菜单中的工具栏上单击 “Validate Design”，然后对模块设计进行全面的设计规则检查。这一操作将进行参数传播，并检查设计不同端点间任何参数失配。

将模块设计集成至顶层设计中

IP 集成器模块设计可集成到更高层次的设计中，也可成为设计层级结构的最高层次。为将 IP 集成器设计集成到更高层次的设计中，应在更高层次的 HDL 文件中实例化该设计。

您可在 Vivado IDE 源文件窗口中选择模块设计并单击 “Create HDL Wrapper”，即可在更高层次上将该模块设计进行实例化。这可为 IP 集成器子系统生成顶层 HDL 文件。

封装定制 IP 和 IP 子系统

您可利用 Vivado IP 打包器来创建定制 IP，以便将其交付于 Vivado IP 目录中。采用行业标准 IP-XACT 格式打包 IP。将打包 IP 的位置添加到 Vivado Design Suite Project Settings 的“Repository Manager”部分。在添加完含有一个或多个 IP 的资源库之后， IP 核将显示在 IP 目录中。您现在就可选择和定制 Vivado IP 目录中出现的 IP。下面给出 Vivado IP Packager的使用流程简介：

1. 使用 Create and Package IP wizard 打包 HDL 和定制 IP 的相关数据文件。
2. 为 IP 客户提供封装定制 IP。
3. 最终用户向 Vivado Design Suite Project Settings 的“Repository”部分添加 IP 位置。
4. IP 目前出现在 IP 目录中，最终用户可以选择和定制与赛灵思 IP 相似的 IP。

Create and Package IP Wizard 将引导您逐步通过 IP 封装流程，让您从工程、模块设计或指定的目录封装 IP，或是创建并封装新的模板 AXI4 外设。

重要提示： 需建立定制 IP 的目录结构，以便使组成 IP 定义的所有 HDL 文件均位于正被封装的 IP 的目录下。工具可通过绝对路径在更高目录级参考 HDL 文件，但这会使打包的 IP 不能在网络间进行移植。

IP 打包器的使用让最终用户无论是使用赛灵思 IP、第三方 IP 还是定制 IP 时都能获得一致的体验。如需了解有关创建和打包 IP 的更多信息，敬请参阅：《Vivado Design Suite 用户指南：创建和封装定制 IP》(UG1118) [参照 27]。

重要提示： 当创建定制 IP 定义文件时，应确保正确设定所需支持的器件系列列表。当您的 IP 需与多个器件系列配合使用时，这一点尤其重要。

提示： 在打包 IP HDL 之前，通过仿真和综合来验证设计以确保其正确性。

在打包定制 IP 的创建阶段， Vivado IDE 的另外一个实例可与`edit_ip`工程一起打开。该工程属于临时缓存，工具会在封装 IP 完成后立即将其清除。

创建定制外设

Vivado Design Suite 要求所有存储器映射接口均采用 AXI 接口。 Vivado Design Suite 提供了Create and Package IP Wizard 的选项，有助于创建定制 IP 以符合 AXI 接口标准。 Create and Package IP Wizard 可生成三类 AXI 接口：

- AXI4：用于单个地址相位中最多允许 256 个数据传送周期的内存映射接口
- AXI4-Lite：轻量、单事务存储器映射接口
- AXI4-Stream：无需地址阶段而且数据突发包的大小无限制的 AXI 接口

如果您已经拥有 IP 的核功能，可以用 Create and Package IP Wizard 为您的定制 IP 生成 AXI 接口逻辑。 Create and Package IP Wizard 可创建一个包含 HDL、驱动、测试应用、总线功能模型 (BFM) 和实例模板的模板 AXI4 外设。在外设创建完毕后，添加用户设计文件即可完成定制 IP。如需了解更多详情，敬请参阅：《Vivado Design Suite 用户指南：创建和封装定制 IP》(UG1118) [参照 27] 以及 《Vivado Design Suite 用户指南：采用 IP 集成器设计 IP 子系统》(UG994) [参照 26] 。

赛灵思 LogiCORE™ IP AXI Bus Functional Model (BFM) 内核由 Cadence Design Systems 专为赛灵思开发的，用于支持由客户设计的基于 AXI 的 IP。 AXI BFM 内核支持各个 AXI 版本（AXI3、AXI4、AXI4-Lite 和 AXI4-Stream）。BFM 是一种加密的 Verilog 模块。BFM 的运行使用 Verilog 语法文本文件中包含的 Verilog 任务序列加以控制。该内核可在模块设计上进行实例化，并连接到定制 IP，以便检查 AXI 功能。如需了解有关 BFM 内核的更多信息，敬请参阅：《AXI BFM 内核 LogiCORE IP 产品指南》 (PG129) [参照 15]。

逻辑仿真

图2-4 显示了所有应通过 Vivado 仿真来实现功能和时序仿真的位置。

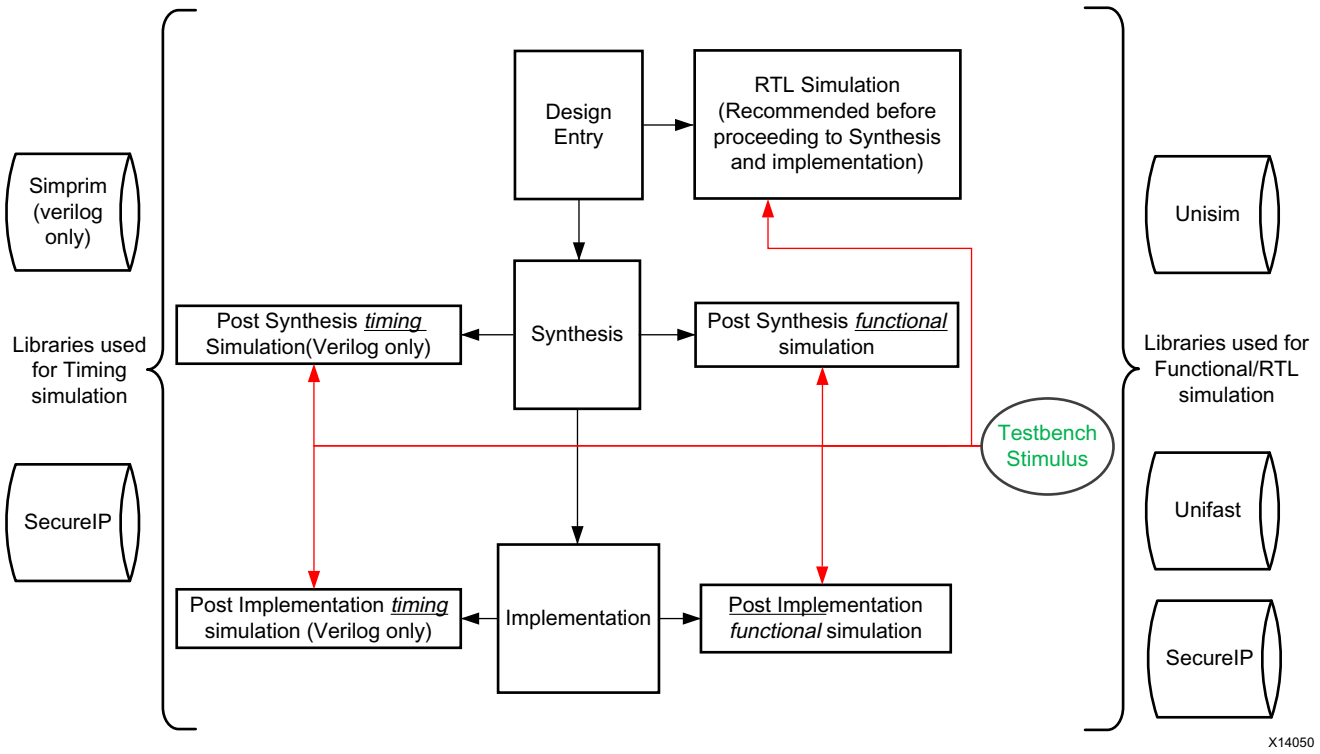


图 2-4 : 设计流程中不同点的仿真

设计流程中早期阶段的功能仿真

使用功能或寄存器传输级 (RTL) 仿真来验证语法和功能性。对于较大型的层级硬件描述语言 (HDL) 设计，可在测试用户整个设计之前对每个模块单独进行仿真。这个过程可让代码调试更轻松易行。行首次通过仿真通常用来验证 RTL （行

为）代码并确认设计能否按预期运行。这一步不会提供时序信息，您应在单位延迟模式下执行仿真以避免可能产生竞态条件。

初始设计的创建应使用可综合的 HDL 架构。除非有必要，否则严禁实例化特定组件。这样允许：

- 更多可读代码
- 更快速、更简单的仿真
- 代码可移植性 （能移植到不同器件系列）
- 代码重用 （能在以后设计中使用相同代码）

如果该组件不可调用，您可能有必要实例化该组件。组件的实例化可能会使代码具备架构特定性。实例化的组件可以包括：

- 实例化 UNISIM 库组件
- 实例化 UniMacro 组件
- SecureIP

当每个模块按预期发挥作用后，即可创建设计层面的测试平台，以验证整个设计是否能按计划发挥作用。在最后时序仿真中再次使用该测试平台，以确定设计是否可在最差延迟条件下按预期发挥功能。

仿真时使用结构网表

在您完成设计的综合/实现后，即可在功能模式或时序模式下执行网表仿真。网表仿真能帮助您处理下列事项：

- 发现下列因素导致的综合后和实现后功能变化：
  - 会导致不匹配的综合属性或约束 （例如 full_case 和 parallel_case）
  - 在赛灵思设计约束 (XDC) 文件中应用的 UNISIM 属性
  - 综合与仿真对语言的任何解读差异
  - 双端口 RAM 冲突
  - 遗漏或未正确应用的时序约束
  - 操作异步路径
  - 优化技术引起的功能问题
- 对在 STA 过程中声明为虚假或多周期的时序路径的敏感化
- 生成评估功耗的网表转换活动
- 不定态

对于网表仿真，您需要使用表 2-2 给出的一个或多个库。

表 2-2 : 仿真库的使用

库名称	描述	VHDL 库名称	Verilog 库名称
UNISIM	赛灵思原语的功能仿真	UNISIM	UNISIMS_VER
UNIMACRO	赛灵思宏命令的功能仿真	UNIMACRO	UNIMACRO_VER
UNIFAST	快速仿真库	UNNIFAST	UNIFAST_VER

表 2-3 提供了这些库的位置。

UNIFAST 库是可选库，可在仿真期间用于加快仿真运行速度。

重要提示： 您不能将 UNIFAST 模型用于时序仿真。

建议： 赛灵思建议将 UNIFAST 库用于设计的初步验证。完整验证应使用 UNISIM 库。

如需了解有关赛灵思仿真库的更多详情，敬请参阅：《Vivado Design Suite 用户指南：逻辑仿真》(UG900) [参照 11] 中的链接。

表 2-3 : 仿真库的位置

库	HDL 类型	位置
UNISIM	Verilog	<Vivado_Install_Area>/data/verilog/xsim/unisims
	VHDL	<Vivado_Install_Area>/data/vhdl/xsim/unisims
UNIFAST	Verilog	<Vivado_Install_Area>/data/verilog/src/unifast
	VHDL	<Vivado_Install_Area>/data/vhdl/src/unifast
UNIMACRO	Verilog	<Vivado_Install_Area>/data/verilog/src/unimacro
	VHDL	<Vivado_Install_Area>/data/vhdl/src/unimacro
SECUREIP	Verilog	<Vivado_Install_Area>/data/secureip/<simulator>/<simulator>_secureip_cell.list.f

注释： Vivado 工具允许您在综合与实现阶段运行功能和/或时序仿真。如需了解有关网表生成的更多信息，敬请参阅：《Vivado Design Suite Tcl 命令参考指南》(UG835) [参照 13] 中的 `write_verilog` 和 `write_sdf` 命令。

除时钟元件外， UNISIM 库的原语或元件没有任何时序信息。为防止功能仿真过程中出现竞态条件，时钟元件具有 100 ps 的时钟相对于输出的延迟。组合信号的波形视图可能因 UNISIM 元件中缺少延迟而出现尖峰和毛刺。

时序仿真

许多用户因运行时间长，不运行时序仿真。如果您决定跳过时序仿真，应确保如下内容：

- 确保您的 STA 约束绝对正确。需要特别注意异常情况。
- 确保您的网表完全等同于通过 RTL 实现的结果。应特别注意综合工具提供的任何与调用有关的信息。

您应该考虑时序仿真，因为完整时序下的仿真是最接近硬件行为的模仿方式。如果您的设计不在硬件上运行，那么在仿真中调试错误会容易得多，只要执行能重新生成故障情况的时序仿真即可。

在 Vivado Design Suite 中运行时序仿真

赛灵思仅支持 Verilog 中的时序仿真。您可使用 `write_verilog` Tcl 命令生成时序仿真网表。仿真网表中的 Verilog 系统任务 `$sdf_annotate` 可指定要读取的标准延迟格式 (SDF) 文件的名称。当仿真器编译 Verilog 仿真网表时，工具自动读取 SDF 文件。

提示： Vivado 仿真器支持混合语言仿真，即如果您是 VHDL 用户，您可以生成 Verilog 仿真网表，然后从 VHDL 测试平台将其实例化。

仿真时间分辨率

赛灵思建议您使用 1 ps 的分辨率运行仿真。有些赛灵思原语组件 （例如 DCM）在功能或时序仿真中都需要 1 ps 的分辨率才能正常工作。

提示： 由于大部分仿真时间都花在 Delta 周期内，因此对赛灵思仿真模型使用较粗糙的分辨率不会带来显著的仿真器性能提升。

建议： 赛灵思建议您不要使用更精细的分辨率，例如 fs。有些仿真器可能舍入数字，而其它仿真器则可能截断数字。

仿真全局置位/复位 (GSR)

赛灵思器件具有连接器件中每个寄存器的专用布线和电路。当您生效专用的全局设置/复位 (GSR) 网络时，该网络会在器件配置后立即释放。所有触发器和锁存器都收到此复位信号，并达成取决于寄存器的定义值。

在网表仿真中，在第一个 100 ns 内自动生效 GSR 信号，用以仿真配置后的复位信号。您也可以为综合前的功能仿真提供一个 GSR 脉冲，但如果设计中具有可复位所有寄存器的本地复位信号，那么就没必要这样做。

如需了解有关 GSR 的更多信息，敬请参阅：《Vivado Design Suite 用户指南：逻辑仿真》(UG900) [参照 11] 中的链接。

禁用不定态传播

当在时序仿真过程中发生时序违规时，锁存器、寄存器、 RAM 或其它同步元件的默认行为是在仿真过程中在相应元件上输出一个不定态。之所以出现不定态是因为实际输出值未知。在该硬件上，寄存器的输出可能显示如下任意行为：

- 保留之前的值
- 更新为新的值
- 进入亚稳态，即在同步元件计时片刻之后才建立确定值

由于该值无法确定，且无法保证准确的仿真结果，因此该元件输出一个不定态 (X) 来代表未知值。不定态输出一直保持到下一个时钟周期，此时如果没有发生其他违规，下一个计时值将更新该输出。

不定态输出的出现会严重影响仿真。例如，由某寄存器生成的不定态会在后续的时钟周期中传播到其它寄存器。这会导致处于测试状态中的设计的很大部分变成未知。

要纠正设计中的大范围不定态传播，应：

- 在同步路径上，分析路径并修复与该路径或其它路径相关联的任何时序问题，以确保电路正确工作。
- 在异步路径上，如果不能以其他方式避免时序违规，应通过使用 ASYNC_REG 属性在时序违规期间禁用同步元件上的不定态传播。如需了解有关使用 ASYNC_REG 约束的更多信息，敬请参阅： 《Vivado Design Suite 用户指南：逻辑仿真》(UG900) [参照 11] 中的链接。

当不定态传播被禁用时，仿真器在寄存器的输出端保留之前的值。在实际的芯片中，寄存器可能会有不同的行为。禁用不定态传播可能会产生与芯片行为不匹配的仿真结果。

注意 ！ 使用这个选项时需非常谨慎。只有在您了解该时序违规且该设计能够从该违规中恢复整个电路的情况下使用。

编译仿真库

赛灵思建议您在开始仿真之前运行库编译，尤其是在您的设计实例化 VHDL 原语或赛灵思 IP 核的情况下。赛灵思 IP 的大部分采用 VHDL，因此如果您没有编译仿真库，就会看到与库绑定有关的错误信息。您可以运行 `compile_simlib` Tcl 命令为目标仿真器编译赛灵思仿真库。另外，您也可以通过单击“Tools > Compile Simulation Libraries”，使用 Vivado IDE 发布该命令。

以系统管理员身份编译库

使用 `compile_simlib` Tcl 命令编译库的系统管理员应在所有用户都能够访问的默认位置编译该库。

以下实例说明的是如何针对所有器件、库和语言为 QuestSim 编译库：

1. 启动 Vivado 工具。
2. 建立 QuestSim。请查阅：《Vivado Design Suite 用户指南：逻辑仿真》(UG900) [参照 11]。
3. 转到 Tcl 控制台，键入下列命令：

```
compile_simlib -simulator questa
```

以用户身份编译库

当您以用户身份运行 compile_simlib 时，应为自己的每个工程都编译库。如果您的工程以单个器件为目标，则只针对该特定器件编译库。

下列命令所示的是如何针对使用 Kintex® UltraScale 器件的设计为 Cadence IES 编译库：

```
compile_silib -simulator ies -directory.-family kintexu
```

仿真流程

Vivado Design Suite 同时支持交互式用户流程模式和批处理用户流程模式。

交互式仿真

Vivado IDE 可与所有支持的仿真器全面集成。在交互模式下，您通过一键点击即可轻松编译并仿真设计。关于建立仿真流程设计的步骤,敬请参阅： 《Vivado Design Suite 用户指南：逻辑仿真》(UG900) [参照 11]。

注释： 关于支持的仿真器的信息，敬请参阅： 《Vivado Design Suite 用户指南：版本说明、安装和许可》(UG973) [参照 3]。

关于该交互式流程，赛灵思提出如下建议：

1. 确保在下列文件中正确设置编译库的位置选项：
   - Mentor 仿真器：`modelsim.ini`
   - IES 仿真器：`cds.lib`
   - VCS 仿真器：`synopsys_sim.setup`
   - Aldec 工具：`library.cfg`

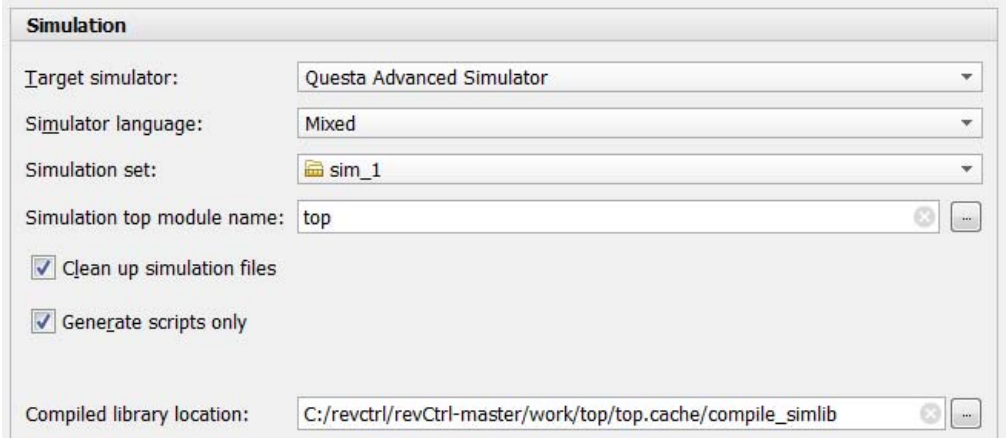


图 2-5 : 设置仿真选项

C:/revctrl/revCtrl-master/work/top/top.cache/compile_simlib 在本例中是一个工程层面的库。您可以使用同一选项指向系统管理员编译的库。

2. 如果您持有的是混合语言仿真器许可证，可始终选择 “Mixed” 作为仿真语言选项。根据仿真器语言选择，您在 IP 仿真时既可使用 RTL 模型，也可使用综合后网表。如果您没有选择“Mixed”，而是在 VHDL 或 Verilog 中二选一，可能会造成仿真速度严重滞后。如需了解该选项的更多信息，敬请参阅：《Vivado Design Suite 用户指南：逻辑仿真》(UG900) [参照 11]。
3. 除非检测到数据损坏，否则请勿单击“Clean up simulation files”。该选项会删除整个仿真目录，这样会导致增量编译选项因仿真器不能访问原始数据库去检查源文件之前是否已被编译过而停止工作。
4. 如果您的验证环境有自检测试平台，请在批处理模式运行仿真。如果从交互模式下调用仿真器波形会显著延长运行时间。

批处理仿真

Vivado Design Suite 可提供名为 launch_simulation 的 Tcl 命令，用于为批处理模式用户生成仿真脚本。建议您使用 launch_simulation 生成的脚本，如果该脚本不适用于您的仿真环境，则建议使用该脚本作为参考，构建自定义仿真脚本。仿真的每一个阶段 （编译、细化和仿真）使用的脚本是独立的，因此您可以方便地把生成的脚本整合到自己的验证环境中。

为 IP 生成仿真脚本

赛灵思IP为仿真提供符合 IEEE P1735 规范的加密 RTL，且这些 RTL 也同样能用于实现，这样您所仿真的设计与将要实现的设计便是完全一样的。但是 RTL 文件加密让您无法直观地检查 VHDL 仿真源文件的库关联性。您可通过在 Vivado Design Suite 中查询文件属性，随时掌握文件的关联性，但赛灵思建议您使用仿真 launch_simulation Tcl 命令来创建仿真脚本。

您可通过发出下列命令为 IP 生成仿真脚本：

```
launch_simulation -scripts_only -of_objects [get_files <ip_name.xci>]
```

例如，下列命令可为模块存储器生成器 IP 生成仿真脚本：

```
launch_simulation -scripts_only -of_objects [get_files blk_mem_gen_0.xci]
```

注释： 该脚本是在为工程选择的 target_simulator 属性的基础上生成。

为设计生成仿真脚本

使用 launch_simulation Tcl 命令您可为整个设计生成仿真脚本。在设计层面生成仿真脚本与按 IP 生成仿真脚本相比能提供明显的优势。在设计层面生成脚本时，在 launch_simulation 中有一个滤波算法，能够清除仿真文件清单中所有 IP 文件副本。较小的文件清单能显著缩短编译时间。

```
launch_simulation -scripts_only
```

仿真流程总结

1. 在进行实现之前运行行为仿真。及早发现问题有利于节省时间与成本。
2. 始终针对支持的第三方仿真器版本。如需了解更多信息，敬请参阅：《Vivado Design Suite 用户指南：版本说明、安装和许可》(UG973) [参照 3]。
3. 使用赛灵思 Tcl 命令 launch_simulation 为选定的仿真器生成脚本。
4. 尽量在工程层面生成仿真脚本。
5. 随时将 target_language 设置为“Mixed“，除非您的仿真器没有混合模式许可证。
6. 如果针对的是 7 系列器件，则使用 UNIFAST 库以提升仿真性能。

   注释： UNIFAST 库不支持 UltraScale 器件原语。

7. 实例化原语会显著延长仿真运行时间。尽量调用逻辑。
8. 在使用第三方仿真器时，确保启用增量编译功能。
9. 尽量禁用波形功能。在 Vivado 仿真器中，在 `xelab` 期间应禁用调试功能从而提升性能。
10. 在 Vivado 仿真器中，启用多线程以加快编译时间。

综合、实现与设计分析

有关这些课题的介绍，敬请参阅：第 5 章：实现。如需了解更多信息，敬请参阅以下资料：

- 《Vivado Design Suite 用户指南：综合》(UG901)[参照 16] 或 《综合设计中心》
- 《Vivado Design Suite 用户指南：实现》(UG904) [参照 19] 或 《实现设计中心》
- 《Vivado Design Suite 用户指南：设计分析和收敛技术》(UG906) [参照 21] 或 《时序收敛和设计分析设计中心》

源文件管理及版本控制建议

版本控制方法

简介

源文件管理和版本控制方法随用户和企业的偏好以及版本控制管理软件而变化。本章将介绍设计团队在管理进行中的设计工程时需做出的部分基本方法选择。本章还将介绍将 Vivado Design Suite 与版本控制系统组合使用的具体建议。在本章范围内， “管理”一词意指使用版本控制系统检入检出源文件版本的过程。

许多设计团队都使用源文件管理系统来管理各种设计配置和版本。有许多现成的系统可供选用，如：Git、 RCS、 CVS、 Subversion、 ClearCase、 Perforce、 Bitkeeper 以及许多其他此类系统。但没有一个系统独占优势。 Vivado Design Suite 能与所有此类系统互动，与其中的每一种系统都能够良好协作，但未考虑与任何一种单独的工具直接集成。

大多数用户采用的方法是把源文件检出到本地版本库，用作本地修改的暂存区。这被称为主版本库的 “沙盒” 或其本地工作版本。这个暂存区的目的是便于用户先对本地修改进行分组和测试，然后再将它们传回主版本库，让其他用户看到最新的修改。为完成设计，可以修改该本地库中的源文件。这些修改后的源文件可随时作为新版本返回到源文件控制系统中。设计结果也可选择性地检入到系统中用于版本存储。许多用户都使用目录结构来存储并管理源文件及结果。

版本控制系统使用不同机制仅对上次检出后有所修改的源文件进行更新。

在 Vivado Design Suite 文件中，有一个明确的子集应该使用版本控制加以管理。还有 Vivado Design Suite 生成的一系列中间文件，管理起来效率低下，而且也没有必要管理。 Vivado Design Suite 源类型中的清单突出显示了关键文件。

单用户访问与多用户访问的对比

设计源文件和 IP 一般存储在可供任意多名正从事某一工程的设计人员引用的数据库或资源库中。但使用公用资源库提供的设计源文件或 IP 会受到任何在其设计中使用这些文件的人所做的修改的影响。对共享源文件或 IP 所做的任何修改都会被所有其他引用同一源文件的设计人员所采用。如果设计的特定修改影响到多个设计，这样会造成意外的后果。

实现多用户访问的最理想方式是让单个设计人员管理设计源文件和 IP，然后按原状共享给其他所有设计。这些源文件可能/应该设置为只读模式，以防任何设计人员无意间更新公用源文件。

此外， Vivado Design Suite IP 和 IP 子系统针对的是特定的器件。因此在考虑设计间共享源文件时，还应该注意确保器件兼容性。

直接使用只读托管的源文件

Vivado Design Suite 允许使用只读源文件和 IP 来汇编设计。如果源文件在设计过程中预计不会发生变化，就可以方便地从它们的只读托管位置引用。如果需要修改源文件，可以把它们检出到本地工作区。

这个调用 Vivado Design Suite 的工作区应该有写入特权，以便写入日志与报告文件、特定缓存和设计结果。

使用本地工作区

用于设计源文件管理的通用方法是将托管的源文件拷贝到进行中工程的本地工作区。设计人员可在这个工作区中随意进行修改，以完成自己的设计。遇到具有里程碑意义的事件时，可将源文件检回版本控制。需要注意的是所有做出的修改都是针对位于工程目录结构深处的文件的本地副本。因此，请注意确保检回版本控制系统的是这个文件。

这个方法只适用于预计会发生修改的设计源文件。静态源文件可从原始的只读托管位置引用。

管理最小源文件集

部分设计团队想要管理用于重新生成设计所需的最小设计文件集。详情，敬请参阅：需管理的最小源文件集。

Vivado Design Suite 实现该功能的具体说明如下：

- IP 及 IP 集成器子系统的输出结果需要重新生成。只有重新生成使用的软件与原本创建该 IP 时使用的软件版本相同，才能获得完全相同的副本。如果全部 IP 输出结果都处于托管状态，该 IP 则可供将来的软件版本引用。
- 在设计编译过程中生成输出结果会显著延长编译时间。

建议： 为取得最佳结果，赛灵思建议管理下一章节建议管理的源文件中列出的所有源文件。

建议管理的源文件

您可以使用一组源文件重新创建 Vivado Design Suite 设计及相关的 IP 和 BD 文件。为规避使用未来版本的 Vivado Design Suite 重新创建的风险以及缩短编译时间，赛灵思建议管理下列源文件。检入所有这些文件，即可使用当前源文件和工具配置设置来重新创建设计。

- 脚本
  - 曾用于编译设计的运行脚本，包括启动运行时使用的任何预加载和后加载脚本。
  - 如曾对 Vivado `init.tcl` 文件做过修改，该文件必须检入。
  - 使用 write_project_tcl 命令创建工程重建脚本。
  - 还可使用工程 .xpr 文件重建设计工程。
- RTL 和仿真测试平台。
  - 所有基于 RTL 的源文件，包括 .include 文件。
  - 仿真测试平台和激励文件。
- 约束
  - 所有 XDC 约束文件及用作约束的 Tcl 命令
- IP
  - 所有 IP 目录和输出结果的文件名称不变
  - 任何使用的 .coe、 .csv、 .bmm 和 .elf 文件
- IP 集成器
  - 整体模块设计位置及所有 IP 子目录、文件和名称均不变
- HLS

  - 所有 C 语言源文件
  - 供 Vivado 综合使用的经封装 HLS IP
  - 脚本
- System Generator for DSP
  - 整个 System Generator 创建的模块目录及所有子目录、文件和名称均不变
- Vivado Hardware Manager
  - 器件编程所需的所有 .bit 文件
  - 包含工具默认设置和自定义设置的 .ltx 文件
- 软/硬件接口
  - 交接 (Handoff) 设计文件 （.hdf 扩展名）含有赛灵思 SDK 为您的设计创建相应的软件工程所需的全部信息。当您使用 `write_hwdef` 或 `write_sysdef` Tcl 命令，或通过“File > Export > Export Hardware”命令导出设计时，就会创建 HDL 文件。
- 技术文档
  - 所有设计相关的技术文档、规格描述、报告等。

需管理的最小源文件集

Vivado Design Suite 设计及其相关的 IP 和 BD 可使用最小源文件集重新创建。但这种方法存在某些局限，具体见管理最小源文件集。

下面是用于重新创建设计的最小文件集：

- 脚本
  - 曾用于编译设计的运行脚本，包括启动运行时使用的任何预加载和后加载脚本。
  - 使用 write_project_tcl 命令创建工程重建脚本。
  - 还可使用工程 .xpr 文件重建设计工程。
- RTL 和测试平台
  - 所有基于 RTL 的源文件，包括 .include 文件
  - 仿真测试平台和激励文件。
- 约束
  - 所有 XDC 约束文件及用作约束的 Tcl 命令
- IP
  - 与每个 IP 关联的 .xci 文件
- IP 集成器
  - 使用 `write_bd_tcl` 命令创建 BD 重建 Tcl 命令
  - 用于每个 BD 的 .bd 文件
  - 模块与注释的图形位置的 UI 目录

Vivado Design Suite 源类型

含有设计描述的 Vivado Design Suite 环境参考源文件。 Vivado Design Suite 中的选项控制如何创建、使用和管理源文件类型。这些源文件包括：

- HDL 和网表文件：Verilog (.v)、 SystemVerilog (.sv)、 VHDL (.vhd) 和 EDIF (.edf)
- 基于 C 语言的源文件 (.c)

- Tcl 文件、运行脚本和 init.tcl(.tcl)
- 逻辑和物理约束(.xdc)
- IP 核文件 (.xci)
- IP 集成器模块设计文件 (.bd)
- Design Checkpoint 文件 (.dcp)
- System Generator 子系统 (.sgp)
- 供相关工具使用的次要文件 （例如仿真器使用的 “do” 文件）
- 模块存储器映射文件 (.bmm、 .mmi)
- 可执行和可链接格式文件 (.elf)
- 系数文件 (.coe)
- 存档工程 (.zip)

Vivado Design Suite 如何识别源文件修改

Vivado Design Suite 会给大部分源文件加盖时间/日期戳，用于提示源文件是否被更新。对各种源文件进行的任何操作都会触发新的时间/日期戳，提示设计已经过期，需更新。

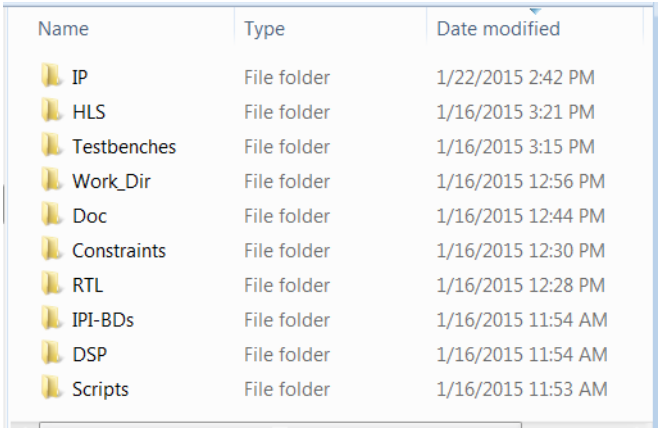
重要提示： 应确保通用版本控制动作不会修改文件的时间/日期戳。

部分版本控制系统即便是在未对文件内容做修改的情况下也会修改时间戳，或是根据检入/检出时间修改时间戳。为保留源文件和生成结果的相对时间修改顺序，应将文件按时间戳倒序排列，然后再进行检入/检出。下面是在保留时间戳顺序的情况下将文件添加到 Git 的指示命令 （Unix 命令）：

```
find .-type f -print0 | xargs -0 ls-rt | xargs git add
```

定义设计源文件目录结构

为有效地区分和管理各类设计源文件，赛灵思建议为每一个进行中的设计都创建一个目录结构。各类目录可用于存储特定类型的源文件，如下图中所示。



Name	Type	Date modified
IP	File folder	1/22/2015 2:42 PM
HLS	File folder	1/16/2015 3:21 PM
Testbenches	File folder	1/16/2015 3:15 PM
Work_Dir	File folder	1/16/2015 12:56 PM
Doc	File folder	1/16/2015 12:44 PM
Constraints	File folder	1/16/2015 12:30 PM
RTL	File folder	1/16/2015 12:28 PM
IPI-BDs	File folder	1/16/2015 11:54 AM
DSP	File folder	1/16/2015 11:54 AM
Scripts	File folder	1/16/2015 11:53 AM

图 2-6 : 设计源文件目录结构实例

这个构思旨在将各类 Vivado 设计源进行分类和组织。为自己的设计和设计团队定义最贴切的命名规则和子目录结构。这个区域本身可由版本控制系统管理，同时也可以是特定设计的检出副本。

Vivado IP 和 IP 子系统 均为器件专用，因此在考虑存储目录名称时应予以注意。这样有助于识别它们，以便后续重复用于其他设计中。

使用活动工作区

赛灵思建议创建一个活动工作目录，以提供一个可写入的区域供创建工程、编译设计、写入结果和进行实验完成设计使用。如需要，这个区域也可以包含设计源文件及 IP 的可写入版本。

适当思考一下如何命名和组织设计专用源文件、脚本和结果，以便于识别，这是一个良好的设计实践。制定子目录和源文件命名方案，以助于后续了解其内容。清除陈旧的或不成功的设计尝试和陈旧的源文件。

注意 ！ Windows 操作系统对路径长度有 260 个字符的长度限制，这会给 Vivado 工具造成影响。为避免这一问题，在创建工程、定义 IP 或托管的 IP 工程以及创建模块设计时，应使用最简洁的名称和目录位置。

管理 Vivado Design Suite 工程

使用 Vivado Design Suite 工程会让与源文件控制系统的交互复杂化。在使用有版本控制的 Vivado Design Suite 工程时，应遵循一定的步骤。工程可维持自己的源文件副本或引用远程源文件并提供自己的源文件管理。但是使用带版本控制的工程有特定的方法。建议使用下面的方法。

使用远程或本地源文件

当通过 Vivado 工具创建和管理工程时，既可在工程的原始位置也可在其本地副本上参考源文件。使用 Vivado IDE 时，本地源文件和远程源文件都可以交互式操作。文本编辑器可用来编辑源文件，同时在开放设计中分析并修改结果。

设计源文件受只读保护，并存储在能从设计目录进行网络访问的任何位置。只读源文件不允许修改，从而会限制 Vivado IDE 交互式设计特性的优势。

为在您使用工程模式时以最快速度与版本控制系统进行连接，您应使用远程设计源文件来创建工程，而不是把源文件拷贝到工程目录中。远程管理源文件便于您在工程中使用交互式 Vivado IDE 功能的同时轻松维护设计源文件。当对源文件做出修改时，您可检入新版本到版本控制系统中。

或者您也可选择把源文件拷贝到在本地工程目录中，让整个设计工程更便于携带且更加独立完善，但这样也会让设计源文件深藏于工程子目录中。

提示： 大多数源文件修改可保存到新的文件名中。

仅使用 .xpr 文件重建工程

仅用单个文件 (`<project_name>.xpr`)，您就可以重建和管理 Vivado Design Suite 工程。工程文件和各类工程源文件是您唯一需要在版本控制下管理的文件。只要源文件在其初始位置并以远程源文件的方式添加到工程中，打开该工程文件和与之关联的源文件即可重建整个工程。如果源文件位于工程内部，使用单独的 XPR 文件将无法正确地进行工程重建。

使用 write_project_tcl 命令重建工程

在使用带源文件控制系统的工程时，赛灵思建议使用 Tcl 脚本式方法重新构建工程。 Vivado `write_project_tcl` 命令能够创建 Tcl 脚本，您可利用该脚本和相同的源文件和设置重新创建现有设计。

```
write_project_tcl <script_file_name>.tcl
```

生成的脚本文件应使用版本控制加以管理并可按下列方式用于重建工程：

```
source <script_file_name>.tcl
```

使用Vivado Design Suite 脚本流程

与源文件控制系统最简单的互动方式就是使用非工程脚本流程。设计人员可将所需的源文件检出到本地目录结构中，也可直接从其托管的位置读取。可能还需要创建新的源文件。当文件就绪后，使用 `read_<source_type> Tcl` 命令将文件传递给 Vivado 综合与实现命令。源文件保持在初始位置。利用合适的代码编辑器，通过交互方式或在设计阶段通过 Tcl 命令修改检出的源文件。比较常见的修改实例就是时序约束修改。

注释： 尽管源文件是只读形式，但这可禁止其被修改。

源文件随后依照设计人员的裁定检入到源文件控制系统。设计检查点、报告和比特文件等设计结果也可检入，以进行版本管理。

使用 IP 和 IP 子系统

IP 和 IP 集成器子系统最好在 Vivado IDE 内通过交互方式进行配置。 IP 自定义向导和交互式 IP 集成器菜单让这项工作变得极为简便。当 IP 或 IP 子系统 配置完成且已生成输出结果后，就可以使用 Vivado Tcl 命令读取源文件。对于 IP 而言，应使用 .xci 文件作为源文件。对 IP 集成器而言，应使用 .bd 文件作为源文件。

注释： 在基于脚本的流程中，还可以使用设计检查点文件 (.dcp) 作为 IP 或 IP 集成器的源文件。赛灵思建议使用 .xci 和 .bd 文件，这样做可以保证它们始终是最新版本。

管理脚本与报告

设计人员还应该管理重建设计所需的所有 Tcl 脚本和修改后的 .ini 启动文件。管理各种输出报告文件同样也有助于识别设计状态，便于将来参考。

管理 IP

赛灵思建议您为设计工程中所使用的 IP 核创建工程专用存储位置。由于 IP 可从任何使用 IP 的设计工程中进行重新配置，因此创建设计专用 IP 可以防止其他设计人员进行不必要的更新。

Vivado Design Suite IP 和 IP 子系统 是器件专用，因此如果想要在设计之间将其共享以及在设置管理区域时，需要予以注意。

Vivado Design Suite IP 可使用下列方法进行配置、存储和管理。

- 创建“管理 IP”工程 （建议）
- 工程本地 IP
- 工程远程 IP

创建“管理 IP”工程 （建议）

在使用集中式 IP 管理功能时，自定义 IP 及其输出结果将存储在集中式资源库中，位于当前设计或工程目录结构之外。每个经过自定义的 IP 都将存储在自己的独立目录下.该IP既可在工程模式也可在非工程模式下从该资源库引用，既可以通过 Tcl 脚本，也可以作为远程源文件添加到工程。

“管理 IP”工程通过创建 IP 存储位置来配置、验证及存储多个 IP 核。您可利用 Vivado IDE 并使用 IP 目录、源文件管理和分析进行 IP 配置，同时，运行基础架构可验证并存储每个 IP 核的结果。这一功能便于 IP 设计人员从 IP 目录选择 IP 核，进行配置和自定义，然后在所需的设计配置下生成输出结果。该环境还允许您通过执行综合与实现来验证 IP。另外，应在管理中让包含 managed_ip_project 目录的整个 IP 位置目录结构保持不变，因为它维持着 IP 运行的状态和输出结果。

工程本地 IP

IP 可在 Vivado Design Suite 工程内部进行配置和存储。该 IP 的输出结果可存放在工程目录结构的内部。将 IP 输出结果存储到本地设计工程中可为整个设计创建一个独立实体，便于封装与共享。如果设计采用独特的 IP 配置，并且您不希望将配置存放在共享 IP 资源库中，那么该方法也同样适合使用。

在工程中利用 IP 目录自定义和添加 IP 很简单。工程的内容完备，便于在一个位置进行管理。在 IP 未被多个工程或多人使用的情况下，这也是一个简便易行的方法。 IP 仅作为工程的一部分，与所有其它设计数据 （如 RTL 源文件和运行结果）一同管理。

整个工程目录结构应在管理中保持不变。

工程远程 IP

此外， Vivado Design Suite 还能让单个 IP 进行单独地远程配置，同时供工程模式和非工程模式使用。在配置任一 Vivado Design Suite IP 的时候，您可以指定 IP 的位置，它可以是工程目录之外的一个位置。参见：图2-7。

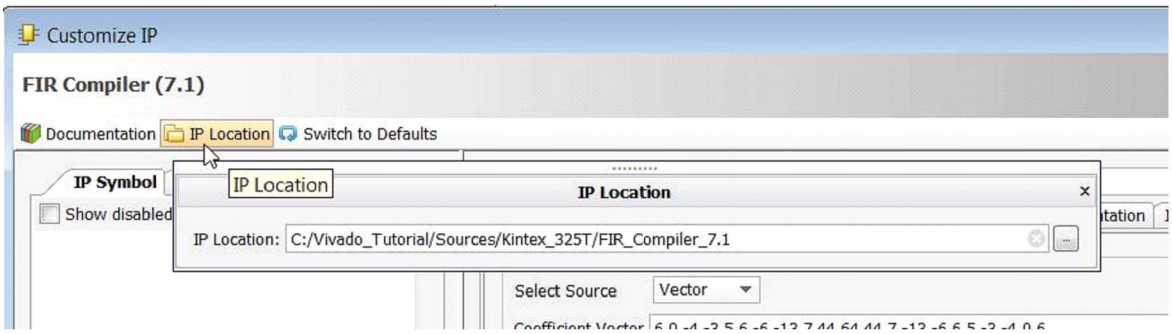


图 2-7 : 为单独 IP 定义远程位置

将 IP 位置设置为 Vivado Design Suite 工程之外的目录下，这可将 IP 配置文件 (`.xci`) 与包括 RTL 源文件和约束在内的各类 IP 输出结果写入到一个独立的目录中，并用与该 IP 自定义名称相匹配的名称命名。

远程 IP 可供任意数量的设计工程使用。但是，由于 Vivado IDE 允许对远程 IP 进行修改和版本更新，因此在修改或更新共享远程IP的时候您必须格外注意，因为这可能会影响其他用户及设计。建立设计专用远程 IP 位置就能避免其他设计人员不必要的修改。

在使用远程 IP 位置时，应遵循下列指导原则：

- 设计一个能区分器件类型、 IP 类型及其它元素的 IP 存储目录结构。
- 在 IP 目录外面存储实例设计，确保不受 IP 更新影响。

管理整个远程 IP 目录结构，使之保持不变。虽然您可以指定 IP 位置，但 IP 本身一直放置在根据 IP 命名的自身目录下，这正和工程内部和 “管理 IP” 工程的情况一样。

管理 IP 源文件

使用 Vivado Design Suite IP 的时候，每个 IP 核都存储在单独的子目录下。这个子目录存放着 .xci IP 主配置文件以及综合和实现该 IP 所需的 RTL、 XDC 和其它相关输出结果文件。赛灵思建议在管理所有这些文件的时候，应让它们的目录结构保持不变，以降低在未来版本的软件中无法复制结果的风险。

注释： 包括 .xci 文件在内的所有输出结果源文件均为只读文件。

重要提示： 在未托管的位置创建 IP 实例设计可便于编译和实验。

某些 IP 还有其它关联文件，如 COE、CSV、ELF 和 BMM 文件。这些文件需要借助版本控制系统来进行管理。在工程流程中，这些文件都会注册进工程中，并可能在设计源文件区出现。虽然这些文件尚未导入到工程中，但它们会停留在它们被引用的位置上。

例如，当您为 FIR 编译器指定一个 .coe 文件时，该文件就会添加到“Design Sources”目录下的“Coefficient Files”文件夹中。该 .coe 文件的位置在 IP XCI 文件中被存储为一个相对路径。当把该 IP 和该 COE 文件放置于版本控制下时，就需要保持这个相对路径。如果需要修改这个相对路径，这个路径就成为了 IP 的属性。您可以使用下列 Tcl 命令来更新这个路径：

```
set_property CONFIG.Coefficient_File {/location/of/coe/file} [get_ips my_FIR_compiler]
```

提示： 您还可以通过在 GUI 中重新自定义该 IP 和指定 COE 文件位置来更新这个路径属性。

注释： 存档工程可将这些文件拷贝到工程目录结构中并更新该 IP 的路径引用，从而正确地管理这些文件。在使用 `write_project_tcl` 命令时，这些文件既可从它们的当前位置引用，也可根据使用的选项导入。

管理全部 IP 输出结果 （建议）

建议您使用版本控制管理含有全部 IP 输出结果的整个 IP 目录。在未来某个时间点，这样可以灵活决定何时以及是否升级 IP。另外，由于无需在每次使用 IP 时都要重新生成 IP，这还缩短了运行时间。

Vivado IDE 只支持 IP 目录中 IP的一个版本。如果您升级了 Vivado IDE 且 IP 不再是当前版本，不过 IP 仍然可以使用。如果 IP 被锁定，您就不能重新自定义 IP 或生成输出结果。但是如果所有的输出结果都存在，那么该 IP 仍可使用。

在默认设置下，所有 IP 都是独立于设计其余部分而进行综合的。产生的综合设计检查点是该 IP 的输出结果。如果在创建 IP 时使用的是默认的无关联流程，那么为确保该 IP 在 Vivado IDE 的未来版本中仍可使用，必须提供这些文件。和其它输出结果一样，该 IP 若不是当前版本，就无法创建这些文件。

仅管理 IP 的 .xci 文件

为恢复设计中使用的 IP 自定义内容，您必须至少保留用于 IP 特定配置的 `.xci` 文件。有了 `.xci`， IP 就可以通过使用创建时使用的相同 Vivado IDE 版本重新生成。如果您想要一直使用该软件版本，或者想要始终更新 IP 至最新的版本，IP `.xci` 可以满足需求。但是，如果将 IP 核 （包括其当前自定义内容和 RTL、 XDC 等）用于 Vivado IDE 的未来版本，您应将整个 IP 输出结果目录放置于版本控制之下，并维持目录结构。

维持 IP 目录结构的目的是为确保各个位置都能够生成输出结果。切勿将多个 .xci 文件存储在单个目录下，因为输出结果在生成的过程中会发生冲突。

管理 IP 集成器子系统

IP 集成器子系统包含多个 IP 以及它们的自定义参数和互联。这些 IP 子系统是用 IP 集成器创建的，并被调用为 Vivado IDE 中的模块设计(BD)。

创建远程模块设计(建议)

当您在创建 Vivado IP 集成器模块设计时 （.bd扩展名），您可指定一个远程位置。这是管理和存储模块设计版本的最简单的方法。 Vivado 工具会把模块设计中使用的所有文件和 IP 存储在远离工程目录的一个目录中，类似于工程远程 IP。

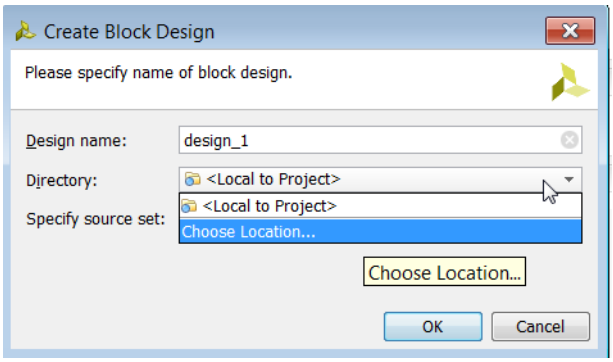


图 2-8 : 为模块设计定义远程位置

如欲了解更多详情，敬请参阅：《Vivado Design Suite 用户指南：采用 IP 集成器设计 IP 子系统》(UG994) [参照 26] 中关于创建模块设计的链接。

创建基于工程的模块设计

在默认条件下， Vivado IP 集成器会在本地工程目录结构内部创建并存储模块设计数据。由于 Vivado IDE 支持交互式修改和源文件管理，这使得与版本控制系统的交互变得棘手。将 IP 输出结果存储在设计工程中能为整个设计建立独立的实体，便于打包和共享。如果设计采用独特的 IP 配置，并且您不希望将配置存放在共享 IP 资源库中，那么该方法也同样适用。

管理 IP 集成器源文件

在使用 Vivado IP 集成器时， BD 中使用的每一个 IP 核都存储在 BD 下的一个单独子目录中。这个子目录存放 .xci IP 主配置文件以及 RTL、 XDC 和其它用于实现每个 IP 的相关输出结果文件。赛灵思建议使用保持不变的目录结构和文件名称来管理全部 BD 文件，以降低使用未来版本的软件无法复制结果的风险。

包括 .bd 文件在内的所有输出结果文件均为只读文件，然而这对 Vivado IDE 的模块设计功能造成了限制。

在使用 IP 实例设计时，您应注意确保在无托管的位置创建它们，以便进行编译和实验。在默认设置下它们创建在 IP 子目录内部。

仅使用 .bd 文件重建模块设计

与 BD 关联的 .bd 源文件至少应该使用版本控制进行管理。把 .bd 作为工程源文件进行添加，会触发模块设计和相关 IP 输出结果的重新生成。这种方法相当耗时，因为所有的 IP 和模块设计输出结果都需要重新生成并进行编译。这也会造成限制，只能用其原来创建时使用的软件版本来重新创建模块设计。赛灵思建议检入整个 bd 目录。确保 BD 已成功生成，包含所有相关的 IP 输出结果。

使用 write_bd_tcl 命令重新创建模块设计

Vivado `write_bd_tcl`命令能够创建 Tcl 脚本，您可利用 Tcl 脚本、相同的 IP、连接功能以及设置重新创建当前的 IP 集成器模块设计 (`.bd`)。确保 BD 已成功生成，包含所有相关的 IP 输出结果。

```
write_bd_tcl <script_file_name>.tcl
```

只要源文件仍然位于相同位置，您在工程流程和非工程流程下使用该命令就能重新创建模块设计。

```
source <script_file_name>.tcl
```

您应使用版本控制来管理生成的脚本文件。

管理仿真源文件

仿真源文件结合测试平台和仿真脚本，通常是构成设计源文件的主体部分。这些文件全部都应该使用版本控制进行管理。它们可以从 Vivado Design Suite 导出到单个目录中。

如欲了解有关用于逻辑仿真的导出文件的更多信息，敬请参阅：为 IP 生成仿真脚本和为设计生成仿真脚本。

管理 System Generator 源文件

赛灵思 System Generator 创建的整个工程目录都应在版本控制的管理下保持不变。

管理 HLS 源文件

所有基于 C 语言的设计源文件、运行脚本以及可能的输出封装 RTL IP 均应通过版本控制进行管理。

管理设计结果

根据您设计团队的具体需求，您可能还需要让更多文件接受管理。这些可能包括设计比特流文件、实现结果、设计检查点、日志文件与报告、自定义 Tcl 命令和运行脚本。正确管理这些文件。

使用设计检查点

Vivado Design Suite 使用设计检查点文件 (`.dcp`) 存储设计流程中当前阶段的当前状态。设计检查点包括检查点写入阶段的网表、约束以及设计结果。

在每个设计阶段完成后都应写入设计检查点。在使用工程进行综合与实现运行时，会自动创建设计检查点。

可在 Vivado IDE 中打开检查点，以进行设计分析和约束修改。约束更改可被写入到新的约束文件中，以在下次流程运行中使用。检查点是流程中给定状态下的设计图像。检查点不包含整个工程或源文件。检查点一般会被传递到后续的设计步骤，比如生成比特流。

您的设计团队也可选择存储里程碑设计检查点。

设计存档

`archive_design` 命令可将整个工程打包成 zip 压缩文件。该命令有几个用于存储源文件和运行结果的选项。本质上说，整个工程被复制到本地内存，然后压缩并存储到磁盘上，同时保持初始工程的完整性。该命令还能将任何远程源文件复制并存档。该功能便于把设计描述发送给其他人或是存储为独立实体。如果您使用该文件设置会影响设计的具体参数或变量，可能还需要发送您自己的 `init.tcl` 版本的文件。

如需了解更多信息，敬请参阅以下资料：

- 《Vivado Design Suite 用户指南：系统级设计输入》(UG895) [参照 8]
- Vivado Design Suite QuickTake 视频：创建不同类型的工程
- Vivado Design Suite QuickTake 视频：用工程管理源文件

管理硬件管理器工程和源文件

工程 .bit 文件和 .ltx 文件是在 Vivado 硬件管理器中使用 Vivado Design Suite 调试与编程功能所需的主要输出文件。赛灵思建议如果您想要在 Vivado 实验室版本中使用该工程，您可使用版本控制来管理这些文件。

在使用 Vivado Design Suite 工程时

对于在 Vivado 硬件管理器中使用调试与编程功能创建的自定义控制面板、触发器、采集条件、波形配置文件等信息，其会被存储在您的 Vivado Design Suite 工程中的 `project_name.hw` 目录下。赛灵思建议您使用版本控制来管理您 Vivado Design Suite 工程中的 `project_name.hw` 目录。如果您想把该工程切换至 Vivado 实验室版本中使用，这也能起到帮助作用。

管理 Vivado 实验室版本源文件

赛灵思建议您使用版本控制来管理为 Vivado 实验室版本中的工程创建的工程目录。对于在 Vivado 硬件管理器中使用调试与编程功能创建的自定义控制面板、触发器、采集条件、波形配置文件等信息，其会被存储在实验室版本工程目录中的 `hw_*` 目录下。实验室版本工程目录中的 .lpr 文件是您需要使用版本控制进行管理的工程文件。只要 `hw_*` 目录处于它们的原始位置，整个工程可以通过打开这个工程文件而进行重建。

将设计和 IP 升级到最新的 Vivado Design Suite 版本

在设计过程中很可能会推出新版本的 Vivado Design Suite 。您可以升级到新版本，或者保持当前版本。赛灵思推荐使用最新版本，并非强制要求。但是，您应该知道赛灵思不支持两个主要版本之前的版本。新版本可能包含：

- 软件错误修复
- 新 IP 版本
- 更新的器件文件 （包括各种器件的速度文件）
- 赛灵思的新器件
- 新的软件功能
- 性能改进

当升级到新软件版本时， Vivado Design Suite 工程可能也会相应升级。工程和器件文件的升级一般自动执行，无需用户参与。

您可选择升级到最新的 IP 版本，或锁定在配置 IP 时的版本。因此，如果要使用 IP 的锁定版本，应先为该 IP 生成输出结果，然后在后续软件更新过程中使用它们。这样确实能阻止在最新版本上重新配置 IP，除非您使用最初配置 IP 时的软件版本。赛灵思建议您使用最新 IP 版本，但不是强制性的。

IP 子系统 （用 IP 集成器创建）中包含的 IP 也要进行管理。您既可以维持锁定版本的 IP 输出结果，也可以升级到最新版本。包含在特定 IP 子系统中的所有 IP 必须同时升级。

如需了解更多信息，敬请参阅以下资料：

- 充分利用 IP 核
- 《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：管理 Vivado IP 版本升级

如需了解有关将设计移植到最新软件版本的更多信息，敬请参阅：《Vivado Design Suite 用户指南：版本说明、安装和许可》(UG973) [参照 3]。

升级工程

打开工程时，如果工程上次是使用较早版本的 Vivado Design Suite 保存的，会提示您进行升级。该功能允许您以只读方式打开工程。工程更新只与新工程格式有关。该功能不能自动更新 IP 或 IP 子系统。建议将每一个版本都更新到最新的工程格式。

升级 IP

如果 IP 需在以后的 Vivado Design Suite 版本中更新，赛灵思建议您对其进行升级。当使用 Vivado IDE 时，您可经常升级到最新 IP。变更记录会描述升级详情。如果您不希望升级，那么必须生成和存档 IP 的所有输出结果。如需了解更多信息，敬请参阅：管理 IP。

这些保存的目标文档可以使用，但是无法利用新版本的 Vivado Design Suite 进行重新定制化，也无法在最新的 Vivado IDE 版本中创建更多输出结果。

如需了解有关 IP 子系统升级的更多信息，敬请参阅：升级 IP 子系统。

升级 IP 子系统

如果子系统中使用的任一 IP 已经在未来版本的 Vivado Design Suite 中完成更新，那么赛灵思建议您升级整个模块设计。如果您不希望升级，那么必须生成和存档 IP 的所有输出结果。编辑模块设计功能被禁用。如需了解更多信息，敬请参阅：第 36 页的管理 IP。

这些保存的目标文档可以使用，但是无法利用新版本的 Vivado Design Suite 进行重新定制化，也无法在最新的 Vivado IDE 版本中创建更多输出结果。

如需了解更多信息，敬请参阅：《Vivado Design Suite 用户指南：采用 IP 集成器设计 IP 子系统》(UG994) [参照 26]。

第 3 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，敬请参阅：以下资料：

- 《Vivado Design Suite 用户指南：I/O 和时钟规划》(UG899) [参照 4]
- Vivado Design Suite QuickTake 视频：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。完成以后继续规划余下的 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。

配电系统

FPGA 设计师面临着一个特殊的任务，那就是设计一个配电系统 (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 设计和引脚规划指南，全面了解器件 PDS。

PDS 设计期间应考虑的关键因素包括：

- 选择合适的稳压器，并根据功耗估算，满足噪声和电流要求。如需了解更多信息，敬请参阅：第 5 章中的功耗。
- 设置 XADC 电源 （Vrefp 和 Vrefn 引脚）。
- 运行 PDN 仿真。 PCB 设计和引脚规划用户指南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。

如需了解有关 PDN 仿真的更多信息，敬请参阅：赛灵思白皮书：《使用 S 参数模型仿真 FPGA 电源完整性》 (WP411) [参照 46]。

PCB 设计的具体考虑事项

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。

赛灵思建议在设计 PCB 时参考以下条目列表：

- 查看千兆位收发器 (GT) 的 PCB 设计检查清单。
  - 如需了解更多信息，敬请参阅：相关器件的收发器用户指南。
  - 使用信道参数运行 Spice 或 IBIS-AMI 仿真
- 查看 MIG 和 PCIe 设计指南

  如需了解更多信息，敬请参阅：各自的产品指南。
- 使用合适的 PCB 去耦电容器。

  如需了解更多信息，敬请参阅：相关器件的 PCB 设计与引脚规划指南。
- 运行噪声分析。

  Vivado™ Design Suite I/O 管脚分配器可对给定管脚进行 SSN 分析。
- 运行信号完整性分析。

  Vivado 工具可以为设计编写 IBIS 文件。
- 检查端接不当是否引起过冲或下冲问题。
- 运行 I/O 引脚规划中的内置 Vivado DRC。
- 运行设计的功耗估计。
  - 确保掌握总功耗。
  - Vivado Design Suite 具有功耗估计工具 (XPE)，其有助于分析给定设计的功耗。
- 确定开发板是否具有合适的配电系统 (PDS)。
- 查看原理图建议。

  如需了解更多信息，敬请参阅：相关器件的 PCB 设计与引脚规划指南。

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。正确的时钟选择可以产生非常好的效果。需要考虑：

- 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 4 章中的时钟详细介绍了时钟资源 （如需要手布动局）。

选择时钟资源

正确的选择时钟需要对目标架构资源有一定了解，后者根据不同代次的器件或有差异。

7 系列器件

赛灵思 7 系列器件包含 32 个全局时钟缓冲器 (BUFG)。其中 16 个全局时钟缓冲器位于 FPGA 器件水平方向中心的上半部分，而另外 16 个则位于水平方向中心的下半部分。

芯片上半部的 I/O、 PLL 和 MMCM 只能连接到水平方向中心以上的 16 个 BUFG 上。而芯片下半部的 I/O、 PLL 和 MMCM 只能连接到水平方向中心以下的 16 个 BUFG 上。

所以，在为时钟资源选择 I/O 引脚时，应平衡器件上下两部分的时钟输入。这一点在为 BUFGCTRL 选择将要多路复用的多项时钟时非常重要。所有相关的 BUFGCTRL 时钟输入应被放在器件中相同的那一半使他们和同一个 BUFGCTRL 连接。针对 SSI 器件时有一条类似的规则。但是，它只应用于 SLR 的上半部或下半部，而不是应用于一个器件的上下两部分。选择 PLL 或 MMCM 时，请尽量使用 PLL，因为其具有更严格的抖动控制。在如下情况下也可以使用 MMCM：(1) PLL 已用尽；或 (2) 所需的高级功能 MMCM 能提供，而 PLL 不具备。

BUFG 组件可以满足设计的大多数时钟要求，而仅需要较少的：

- 时钟数量
- 设计性能

BUFG 组件易于通过综合调用，并且限制较少，支持大多数普通时钟。

但是，如果时钟要求超出 BUFG 组件的容量或数量，或者用户需要更好的时钟特性，那么赛灵思建议您：

1. 根据可用的时钟资源分析时钟需求。
2. 选择并管理任务所需的最佳资源。

如需了解有关其它时钟组件的信息，敬请参阅：第 4 章中的时钟。

单个或多个区域时钟引脚选择

根据接口尺寸，您可以决定使用支持单个时钟域 (SRCC) 的引脚或支持多个时钟域 (MRCC) 的引脚。如果接口涉及多个 Bank，则必须使用 MRCC 引脚，因为这会增加通过时钟网络的延迟。

单端口时钟必须连接至时钟差分对的 P 端。

UltraScale 器件

对 UltraScale™ 器件来说， FPGA 的时钟输入通常被分为以下几类：

- 全局时钟 (GC)- 指既能驱动 PLL （每组两个）或 MMCM （每组一个）也具备对全局时钟网络专用访问能力的大部分通用时钟输入。通常建议将此时钟布置在阵列内靠近 I/O 或逻辑布局的位置来最大限度降低插入延迟和功耗；不过由于时钟结构的关系布局也有一定灵活性。
- 字节通道时钟 (DBC QBC)- 是指通常与存储器接口及其它高速时钟结合使用的专用 I/O 位片(bit slice)时钟引脚。一般由创建高速接口的同一个 IP 来分配，以确保能达到所需连接功能的正确布局。
- MGT 参考时钟 (MGTREFCLK)- 指 MGT(GTH或GTY) 接口专用的差分参考时钟输入。必须为该时钟输入提供通过一个 IBUFGDS_GTE3 组件的引脚。
- 配置时钟 (EMCCLK、 TCK、 I2C_SCLK、 CCLK)- 这些是有关配置、 JTAG 或 SYSMON 接口的专用时钟。请参阅 UltraScale 配置用户指南了解详情。

如需了解有关 UltraScale 器件时钟的更多信息，敬请参阅：第 4 章中的时钟。

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。

如需了解有关Vivado Design Suite I/O 和时钟规划性能的更多信息，敬请参阅：以下资料：

- 《Vivado Design Suite 用户指南：I/O 和时钟规划》(UG899) [参照 4]
- 《Vivado Design Suite 教程：I/O 和时钟规划》(UG935) [参照 30]
- Vivado Design Suite QuickTake 视频：I/O 管脚分配简介

确定何时需要最终 I/O 配置

PCB 板制造进度安排一般会指出何时需要最终 FPGA I/O 配置。尽可能在创建和综合最初 RTL 设计之后执行 I/O 管脚分配。按照此顺序是因为综合网表具有时钟感知，并且已在结构层面定义了逻辑。这种序列支持许多与时钟相关的 DRC，可确保 I/O Bank 和时钟逻辑分配得当。

设计还可以通过实现以确保：(1) 遵守所有 I/O 和时钟规则；以及 (2) 设计成功生成比特流。赛灵思建议采用该最终 I/O 配置验证流程。

但是，并不是所有的设计周期都允许占用这么长时间。通常必须在具备可综合 RTL 之前定义好 I/O 配置。虽然 Vivado 工具支持基于 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。如需了解更多信息，敬请参阅：PCB 设计指南中的选定器件及相关 I/O 硬件文档。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 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 工具提供一个引脚规划工程环境，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"同时切换噪声"(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。

参阅 《PCB 设计和引脚规划用户指南》，确保器件 I/O 配置正确。如需了解更多信息，敬请参阅：《Vivado Design Suite 用户指南：I/O 和时钟规划》(UG899) [参照 4] 中的"Pre-RTL I/O 管脚分配"章节。

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 非常全面。

参阅 《PCB 设计和引脚规划用户指南》，确保器件 I/O 配置正确。如需了解更多信息，敬请参阅： 《Vivado Design Suite 用户指南：I/O 和时钟规划》(UG899) [参照 4] 中的"网表 I/O 管脚分配"章节。

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，请在设计流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。在迁移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。如需了解更多详情，敬请访问 《Vivado Design Suite 用户指南：I/O 和时钟规划》(UG899) 中的链接。

引脚分配

合理的引脚选择会得到合理的设计逻辑布局。不合理的布局也会导致布线较长、功耗增加，以及性能降低。管脚选择是否合理对于大型 FPGA 器件来说尤为重要。因为有些大型 FPGA 器件会涉及多个晶片，而展开的管脚会导致相关信号传输距离过长。如需了解更多详情，敬请访问 《Vivado Design Suite 用户指南：I/O 和时钟规划》(UG899) 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“设计规则检查 (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，敬请参阅： 《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 特性和拓扑的信息。您必须规定电气特性，包括 I/O 标准、驱动和斜率。

您还必须考虑包括时钟拓扑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。详见选择时钟资源。

如需了解有关为 I/O 规定电气特性的更多信息，敬请参阅：《Vivado Design Suite 用户指南：I/O 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。对于堆叠硅片互联 (SSI) 器件，请将特定接口的所有引脚集合在同一个 SLR 上。

接口控制信号

将以下接口控制信号放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 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 进行电压限制。
- 为不同的配置引脚选择合适的终端。
- 对配置引脚，使用建议的上拉或下拉电阻。

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。

有多种不同的配置选项。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：

- 设置
- 速度
- 成本
- 复杂性

参见：配置。如需了解有关 FPGA 配置选项的更多信息，敬请参阅：《Vivado Design Suite 用户指南：编程与调试》(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 引脚规划步骤。定制了 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。

如需了解更多详情，敬请访问：《Vivado Design Suite 用户指南：I/O 和时钟规划》(UG899) [参照 4] 中的链接。

《赛灵思 Zynq-7000 SoC 和 7 系列器件存储器接口用户指南》(UG586) [参照 44] 和 基于 LogiCORE IP UltraScale 架构的 FPGA 存储器接口解决方案产品指南》(PG150) [参照 45] 包含设计和管脚指南。请确保采纳指南中建议的走线长度，核实所使用的端接是否准确无误，并在 MIG 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。您可在多个 GT 之间共享参考时钟。针对7系列器件，一个 Quad 的参考时钟可取自临近的另一 Quad。而针对 UltraScale 器件，一个 Quad 的参考时钟可取自其上下的最多两个 Quad。对于支持堆叠硅片互联 (SSI) 技术的器件，参考时钟被限于其自身的超级逻辑域 (SLR) 内赛灵思建议您使用 GT Wizard 生成内核。有关管脚建议，敬请参阅：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。对于堆叠硅片互联 (SSI) 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，敬请参阅：《7 系列 FPGA SelectIO 资源用户指南》 (UG471) [参照 36] （或者与您的器件相关的 SelectI/O™ 资源用户指南）。

CCIO 和 CMT 的使用

平衡器件的上下两部分之间的 CCIO 和 CMT 使用，从而平衡对上下 BUFG 组件的访问。对于 SSI 器件，在 SLR 中根据其它 SLR 组件平衡上下 CCIO 组件或 CMT 组件。

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。还无需创建内部的设计。每个硬件接口都应创建单独的设计，并且应该用于在全带宽及所需的速度下运行硬件。可以使用 FPGA 内部环回或简易检查器验证数据是否以所需的速度成功传输。由于单板上的 FPGA 接口正在设计中，所以这些设计可用于验证接口和单板是否能够以所需的速度运行。

这些小型测试设计可以通过 Vivado 快速实现。该设计流程还可根据布局合规性和接口时序要求严格验证所选的 I/O。由于引脚位置已经最终确定下来，所以可以检验出任何潜在的 DRC 或时序问题。

对于某些接口 IP 核， Vivado 套件可以提供测试设计，例如针对 SerDes 的 IBET，或 PCIe 的实例设计。

随后，这些相同的设计还可用于系统性地验证每个硬件组件，然后再验证整个设计的比特流。

采用 SSI 器件进行设计

SSI 管脚考虑事项

在为特定 SLR 中的组件规划管脚时，请将引脚放在同一个 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在 DNA_PORT 所在的主 SLR 中。其它考虑包括如下：

- 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，应将这些信号布局在中间的 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 器件中的单个 FPGA 晶片 Slice。

有源电路

每个 SLR 都包含大多数赛灵思 FPGA 器件常见的有源电路。该电路包含大量的：

- 六输入 LUT
- 寄存器
- I/O 组件
- 千兆位收发器 (GT)
- 模块存储器
- DSP 模块
- 其它模块

SLR 组件

多个 SLR 组件组装成一个 SSI 器件。

SLR 组件被垂直堆叠在硅中介层。

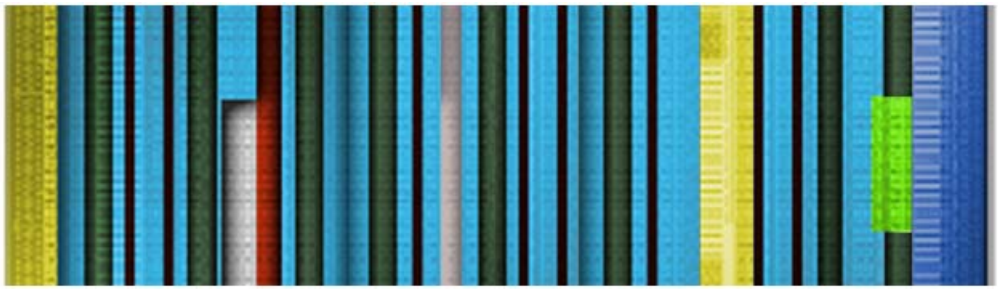


图 3-1 : 单个 SSI SLR

垂直堆叠多个 SLR 组件，用以创建 SSI 器件。

- 底部 SLR 是 SLR0。
- SLR组件序号随着位置垂直上升而递增。

例如，XC7V2000T 器件中有四个 SLR 组件。底部 SLR 为 SLR0。SLR0 的上一层为 SLR1。SLR1 的上一层为 SLR2。顶部 SLR 为 SLR 3。

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

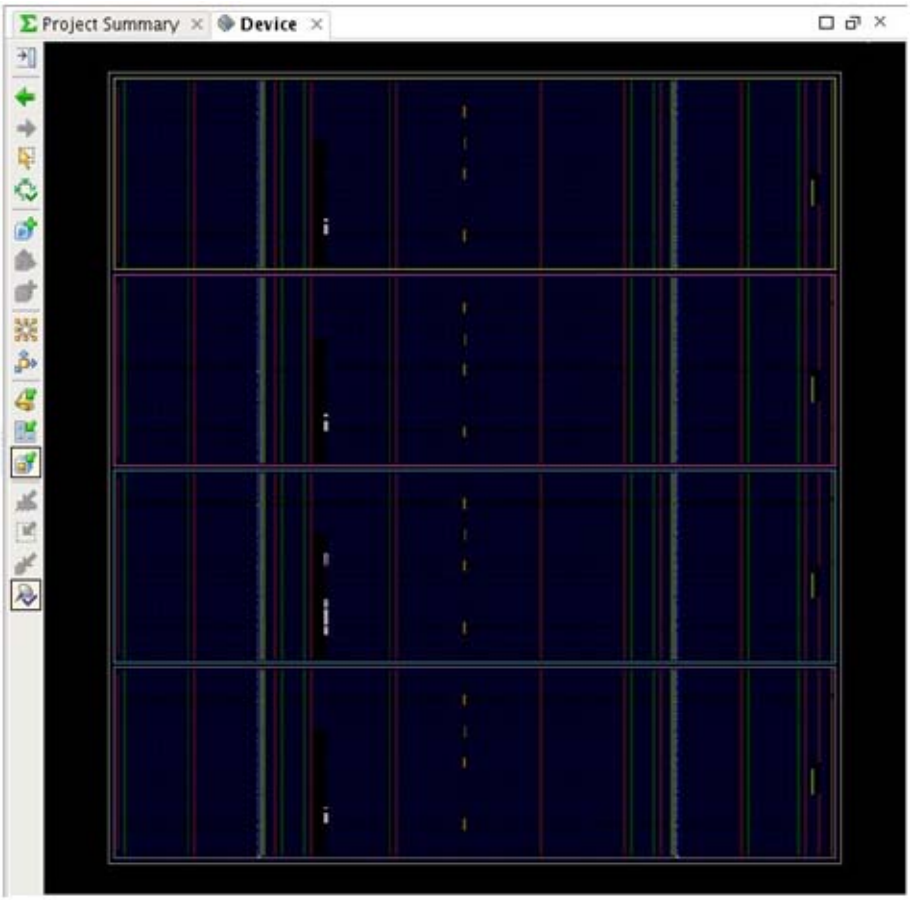


图 3-2 : Vivado 工具清楚呈现 XC7V2000T 器件

主超级逻辑区域

每个 SSI 器件都有一个主 SLR，见下表。

表 3-1 : 主 SLR 参考基准

器件	器件查看器	JTAG 指令
XC7V2000T	SLR1	SLR0
XC7VX1140T	SLR1	SLR0
XC7VH580T	SLR1	SLR0
XC7H870T	SLR2	SLR0
XCKU085	SLR0	SLR0
XCKU115	SLR0	SLR0
XCVU125	SLR0	SLR0
XCVU160	SLR1	SLR1
XCVU190	SLR1	SLR1
XCVU440	SLR1	SLR1

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 是唯一包含专用电路的 SLR，专用电路包括：

- DNA_PORT
- 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`

图3-3 显示了 XC7V2000T 器件中的主 SLR 。

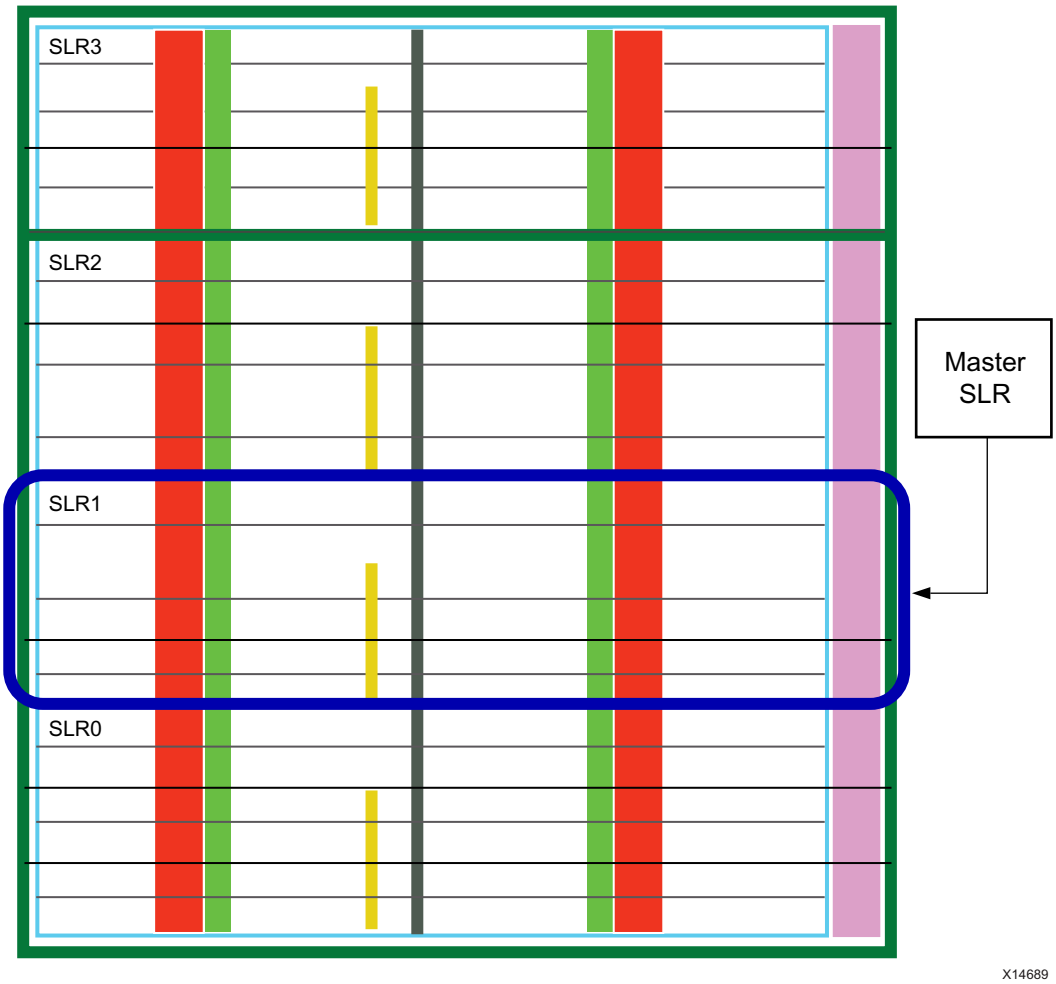


图 3-3 : XC7V2000T 器件中的主 SLR

Virtex-7 器件系列中的 SLR 组件

使用两个不同的 SLR 组件创建 Virtex®-7 器件系列：

- XC7V2000T 器件

- XC7VX1140T 和 Virtex-7 HT 器件系列

XC7V2000T 器件

XC7V2000T 器件共享同一类型的 SLR，其中包括：

- 近 500,000 个逻辑单元
- 下列组件的组合：
  - I/O
  - 块状 RAM
  - DSP 模块
  - GTX 收发器
  - 其它模块

XC7VX1140T 和 Virtex-7 HT 器件系列

XC7VX1140T 器件和 Virtex-7 HT 器件系列采用的 SLR 组件包括：

- 近 290,000 个逻辑单元
- GTH 收发器
- 大量块状 RAM 和 DSP 组件，比 XC7V2000T SLR 组件更多

表 3-2 : 每类 Virtex-7 SLR 中的关键资源

	Virtex-7 T SLR	Virtex-7 XT/HT SLR
逻辑单元	488,640 版本	284,800 版本
Slice	76,350 版本	44,500 版本
块状 RAM	323 版本	470 版本
DSP Slice	540 版本	840 版本
时钟区域/MMCM	6 版本	6 版本
I/O	300 版本	300 版本
收发器	12 版本	24 版本
SLR 之间互联	13,440 版本	10,560 版本

注释： 外合 I/O 与收发器的实际数量或会根据所选器件和封装而有所不同。

UltraScale 器件系列中的 SLR 组件

使用三个不同的 SLR 组件创建 UltraScale 器件系列：

- XCKU085 与 XCKU115
- XCVU125、 XCVU160、 XCVU190
- XCVU440

XCKU085 与 XCVU115

XCKU085 与 XCKU115 共享同一类型的 SLR，其中包括：

- 近 580,000 个逻辑单元
- GTH 收发器

- 大量的 DSP

XCVU 125、 XCVU 160、 XCVU 190

XCVU 125、 XCVU 160与 XCVU 190 共享同一类型的 SLR，其中包括：

- 近 625,000 个逻辑单元
- 多个 GTH 与 GTY 收发器
- 多个 PCIe、 Interlaken 与以太网硬IP块

XCVU440

XCVU440 其中包括：

- 近 1,500,000 个逻辑单元
- 多个选择 I/O
- GTH 收发器

表 3-3： 每类 UltraScale 器件 SLR 中的关键资源

	Kintex? SLR	XCVU190	XCVU440
逻辑单元	580,440 版本	626,640 版本	1,477,560 版本
CLB	41,460 版本	44,760 版本	105,540 版本
块状 RAM	1,080 版本	1260 版本	840 版本
DSP Slice	2,760 版本	600 版本	960 版本
时钟区域/MMCM	30 版本	30 版本	45 版本
I/O	624 版本	520 版本	520 版本
GTH 收发器	32 版本	20 版本	30 版本
GTY 收发器	0 版本	20 版本	0 版本
SLR 之间互联	17,280 版本	17,280 版本	25,920 版本

注释： 外合 I/O 与收发器的实际数量或会根据所选器件和封装而有所不同。

硅中介层

硅中介层是 SSI 器件中的无源层。

该层在 SLR 组件之间为以下工程布线：

- 配置
- 全局时钟
- 一般互联

硅中介层具有：

- 电源和接地连接
- 配置
- 晶片内连接
- 其它所需连接

该有源电路位于 SLR 中。硅中介层通过硅通孔 (TSV) 组件与封装基片相结合。这些组件将 FPGA 器件的电路连接到封装球栅上。

硅中介层是 SLR 组件和封装基片之间的管道。可以将以下工程连接至器件封装上：

- 电源和接地连接
- I/O 组件
- 千兆位收发器 (GT)

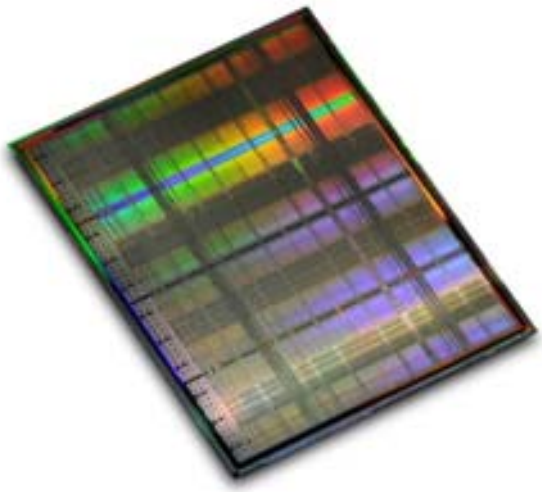


图 3-4 : 硅中介层

超长线路 (SLL) 布线

- 超长线路 (SLL) 布线可以为从一个 SLR 发送至另一个 SLR 的信号提供一般连接。
- SLL 布线位于硅中介层中。
- SLL 布线通过直接与 SLR 互联模块相连的微凸块结点连接至 SLR 组件。
- SLL 布线与 SLR 中的 Vertical 12 布线的中心相连。

SLL 连接

在 7 系列器件中，每个 SLL 组件的垂直长度为 50 个互联模块 （相当于 50 个 Slice 组件）。这与赛灵思 7 系列 FPGA 器件中一个时钟区域的高度完全相等。

在 UltraScale 器件中，每个 SLL 组件的垂直长度为 60 个互联模块 （相当于 60 个 Slice 组件）。这与赛灵思 UltraScale 器件中一个时钟区域的高度完全相等。

在 7 系列 SLR 相邻的时钟区域中，在时钟区域的每个互联模块上都有个与相邻 SLR 相连的互联点。UltraScale 器件和 7 系列的区别在于 SLL 是连接在一个被称为 Laguna 的专用接口上的。 Laguna 接口拥有可用或可绕开的专用寄存器，从而实现高速流水线接口或较低速的飞流水线接口穿过 SLR。

表 3-4 : 每个 SLR 交错的 SLL 组件

器件	SLL 组件
7V2000T	13,440 版本
7VX1140T	10,560 版本
KU085	17,280 版本

表 3-4 : 每个 SLR 交错的 SLL 组件

器件	SLL 组件
KU115	17,280 版本
VU125	17,280 版本
VU160	17,280 版本
VU190	17,280 版本
VU440	25,920 版本

同 xc7v2000t 相比，7VX1140T 器件的 SLL 组件数较少，因为它拥有更多 DSP 和块存储器列。这些列可在同一给定区域内取代更多互联模块。 UltraScale 器件中邻近 SLR 的时钟域内 SLL 数量是固定的。每个时钟区有 2880 个 SLL。

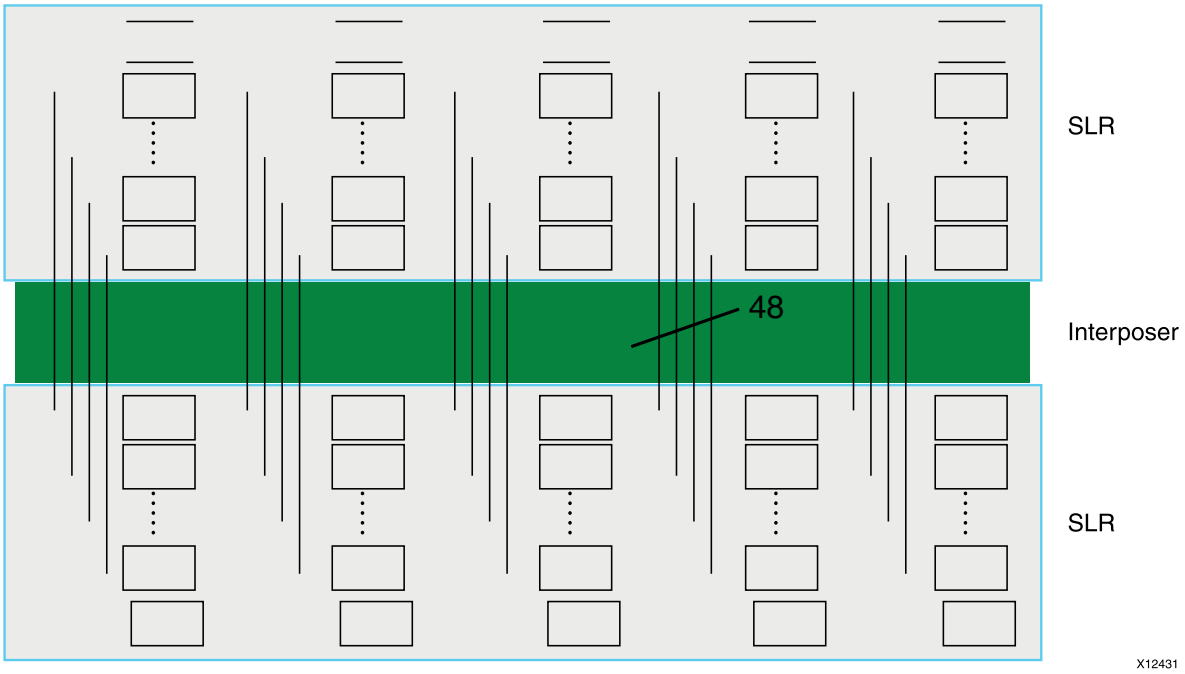


图 3-5 : 一个 7 系列 SSI 器件中的交错式 SLL 连接

SLR 组件之间的比例以及间隙大小仅供说明用。实际间隙相对小得多。

在 7 系列器件中，SLL 组件连接至位于 Vertical 12 Long Line 中心点的 SLR，该垂直长线跨越 SLR 中的 12 个互联模块。UltraScale 器件使用专用的 Laguna 接口。

这种连接方式提供三个最佳位置，供 SLL 连接至 SLL 或将 SLL 从 SLR 连接至相邻的 SLR，这种连接方式让布局更灵活，从而减少对性能或功耗的影响。

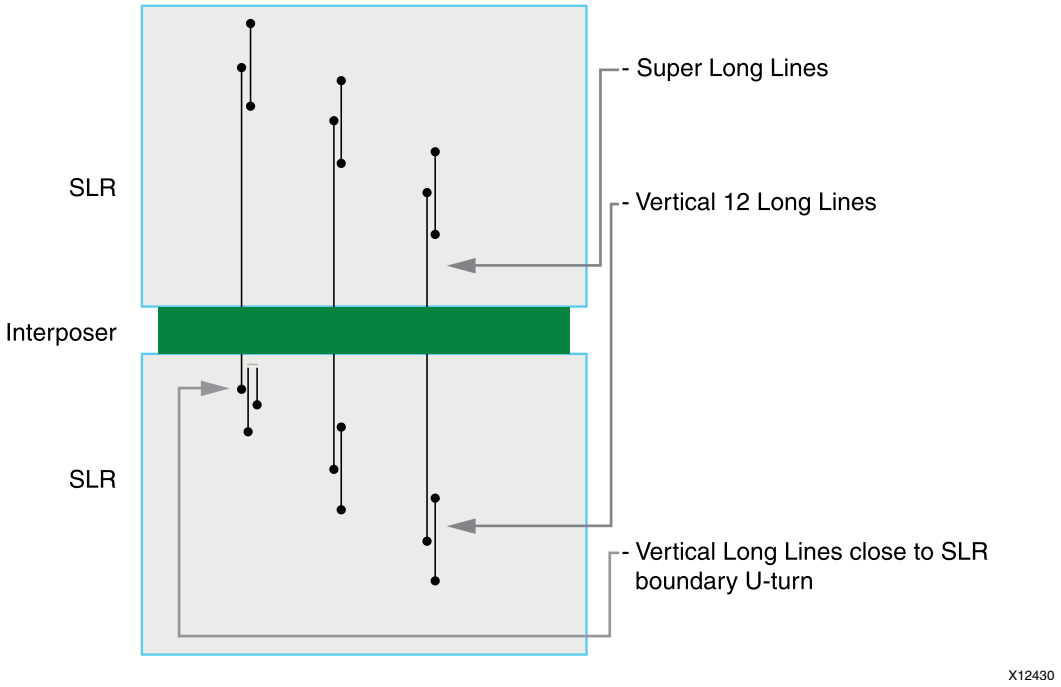


图 3-6 : SLR 中 SLL 的连接图

传输限制

提示： 对跨越多 SLR 的高速运行而言，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
- 块状 RAM 地址级联
- 以及其他专用连接，如 DCI 级联和块状 RAM 级联。

这些工具通常会考虑这种传输限制。为确保设计思路正确且能达到您的设计目标，在建立超长 DSP 级联时也务必考虑传输限制，并手动将该逻辑电路布置到 SLR 边界附近；为设计设定管脚时也应如此。

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。还无需创建内部的设计。每个硬件接口都应创建单独的设计，并且应该用于在全带宽及所需的速度下运行硬件。可以使用 FPGA 内部环回或简易检查器验证数据是否以所需的速度成功传输。由于单板上的 FPGA 接口正在设计中，所以这些设计可用于验证接口和单板是否能够以所需的速度运行。

这些小型测试设计可以通过 Vivado 快速实现。该设计流程还可根据布局合规性和接口时序要求严格验证所选的 I/O。由于引脚位置已经最终确定下来，所以可以检验出任何潜在的 DRC 或时序问题。

对于某些接口 IP 核， Vivado 套件可以提供测试设计，例如针对 SerDes 的 IBET，或 PCIe 的实例设计。

随后，这些相同的设计还可用于系统性地验证每个硬件组件，然后再验证整个设计的比特流。

FPGA 电源因素与系统关联性

规划 PCB 板时，请务必考虑功耗：

- FPGA 器件以及用户设计提出了系统电源与散热要求。
- 电气及物理因素可影响电源以及 FPGA 器件的散热，从而大大影响器件性能。

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

FPGA 器件的供电路径

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加电顺序。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，敬请参阅：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。

FPGA 器件的功耗组成

每个电源的总供电量均由三部分组成。

- 器件静态 （泄漏）功耗

  器件运行及编程所消耗的功率。其中很大部分功耗是由于维持器件配置所用晶体管的泄漏所致。

- 设计静态功耗

  器件配置好但未处于活动状态下时消耗的其它连续功率包括来自 I/O 终端、时钟管理器以及无论设计活动运行与否，只要使用便会耗电的其它电路等所消耗的静态电流。

- 设计动态功耗

  由设计活动消耗的其它功率。这一功耗会随着设计活动的进行发生变化。其功耗大小还取决于电压以及使用的逻辑电路和布线资源。

功耗路径

供给器件的总电力会通过多种路径输入或者输出到 FPGA，包括热能和片外功耗。

热能

热功耗是 FPGA 内部消耗的功率，表现为热的产生，产生的热量会使器件的结温升高。然后这部分热量会传递到环境中。因此单板上必须设计散热路径，以确保将结温保持在器件工作范围内。

片外功耗

片外功耗是从电源流经 FPGA 电源引脚，然后流出 I/O，最后被外部单板组件消耗的电流。 FPGA 器件提供的电流一般会被 I/O 终端、 LED 或其它芯片的 I/O 缓冲器等片外组件消耗。这些功耗不会使 FPGA 器件自身的结温上升。然而，必须设计电源线和接地线来传输这部分电力。

功率模式

从加电到断电， FPGA 器件要经过多个电源阶段，并伴有不同的功率需求：

- 加电
- 启动功率
- 待机功率
- 有功功率

加电

加电功率是 FPGA 器件首次加电时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。

启动功率

启动功率是指在器件初启和配置期间所需的功率。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。 XPE可用于解读此要求。当工艺参数被设为 Maximum 时，对每个电压的电流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。

待机功率

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

待机功率是设计运行时电源应提供的最小连续功率。

有功功率

有功功率 （又称“设计动态功率”）是器件运行应用程序时所需功率。有功功率包括待机功率 （全部静态功率）以及因设计活动 （设计动态功率）产生的功率。有功功率是瞬时发生的，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

除自身设计外，功耗还取决于诸多因素。以下因素会影响器件的电压和结温，进而影响功耗。影响功耗的环境因素包括：

- 供电策略
- 冷却策略

供电策略

供电策略包括：

- 稳压器技术
- 去耦网络性能
- FPGA 器件选择

稳压器技术

现有多种不同稳压器技术来平衡输入输出压差、响应时间、最大电流以及输出电压精度限制。

去耦网络性能

除了在短暂的大功率需求时段为 FPGA 器件供电外，还用一个高效去耦电路减少来自稳压器的电流浪涌请求，并改善稳压器的整体功耗。

FPGA 器件选择

不同 FPGA 器件需要不同的电源数目和电压水平。所有赛灵思 FPGA 器件都在资源、性能与功耗方面进行了权衡。选择最能满足您要求的器件。

过度关注一个特性 （比如性能）会对另一个特性 （比如功耗）形成负面影响。选择能够支持较低内核电压或较低电压 I/O 接口的器件可降低功耗。

冷却策略

冷却策略包括：

- 系统环境
- 散热器
- 封装选择
- 组件布局

系统环境

系统机箱的形状和尺寸 （以及环境温度）是影响产生的热量向环境扩散的主要因素。

散热器

散热器的尺寸、形状、导热胶和安装以及最后的相关强制气流系统决定了从 FPGA 器件提取热量的多少。

封装选择

除成本和信号完整性外，封装的尺寸、材料以及与单板的连接都会影响产生的热量从顶部和底部向环境传递的方式。散热器和开发板之间的接触面越大，热阻就越小。

组件布局

相对于系统机箱及其它开发板材料、装配和组件的组件布局会影响热量向环境传递的方式。比如障碍物可能会减少 FPGA 器件附近的气流或使气流转向。紧挨 FPGA 器件如有其它可产生热量的组件，则该组件可能会使 FPGA 器件附近的气流温度上升，并降低散热器的散热效果，或通过开发板材料将热传导至 FPGA 器件内。

功率模型精度

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。这一准确度认定会显示在“特性描述 (Characterization)”字段。器件系列的特性描述数据会按以下顺序发生变化：

- 早期器件认定

初步器件认定生产器件认定

建议： 用最新版“Xilinx Power Estimator (XPE)”反映最新可用数据。

早期器件认定

带早期器件认定的器件拥有数据模型，其数据模型主要以对早期批量生产器件的仿真结果或测量值为基础。预先数据一般会在产品推出一年内提供。尽管可能出现漏报或多报，但认为预先数据相对稳定和保守。一般认为预先数据的精度低于初步数据和生产数据。赛灵思建议您与现场应用工程师 (FAE)探讨最新数据。

初步器件认定

初步器件认定是以完整的早期生产芯片为基础。几乎描述了器件结构中所有模块的特性。与预先数据相比，它对功耗报告的准确性得到了提高。

生产器件认定

在对某个器件列的成员的足够量产芯片进行特性描述以提供多个生产批量间的全面功耗关系，之后发布生产器件认定。带有生产特性描述数据的器件模型预计未来不会发生变化。

FPGA 器件功耗及整体系统设计流程

从工程构思到完成，需要考虑诸多不同功耗影响因素。暂时忽略所有其它问题 （包括功能、性能、成本和上市时间），功耗相关任务可分为以下几类：

- 物理域，包括机箱、开发板外形、电力输送系统以及热功耗系统
- 功能域，包括面积、性能，以及 I/O 接口信号完整性

一般来说，在设计流程初期会先选择硬件并确定硬件尺寸以便对单板进行原型设计。在设计流程中还可以早早估算出 FPGA 器件功能对功耗的影响，然后随着设计逻辑的推进对其进行优化。图3-7 说明了一种典型的系统设计流程，并强调了与功耗相关的决策点。

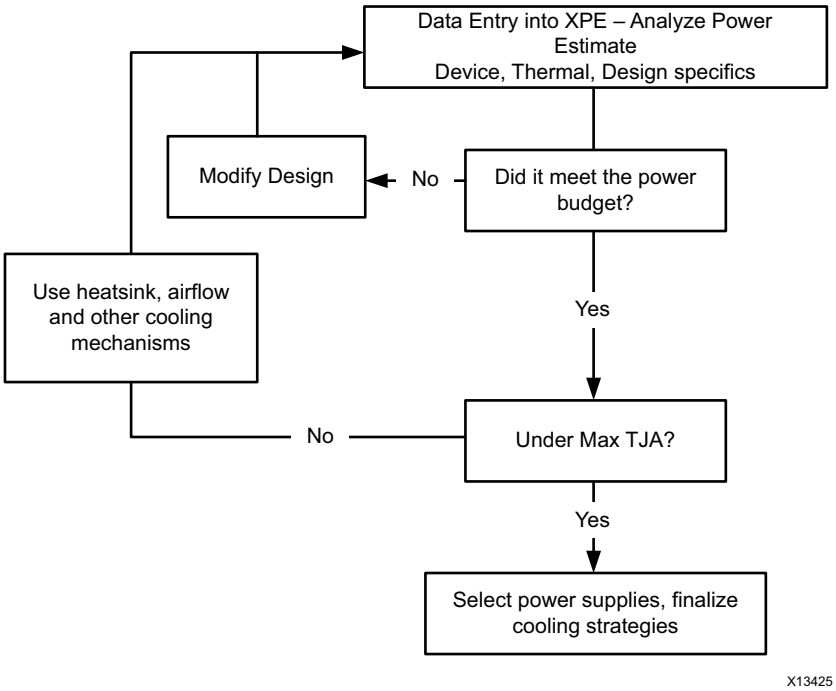


图 3-7 : PCB 规划流程中的功耗管理

当您选择器件和相关冷却部件时， FPGA 逻辑尚未到位。需要通过某种方法谨慎地估算 FPGA 的逻辑功耗要求。赛灵思建议使用可与目前设计相当早期设计的逻辑数据进行估算。最佳估计值输入设计数据，之后再修改该数据。关于将早期设计的数据导入 XPE 的信息，敬请参阅：第 5 章中的实现设计。

赛灵思强烈建议进行热模拟。可向赛灵思索要热模型。这样可以提高准确度，同时提供 ThetaJA，可在 XPE 中设定该 ThetaJA。

- 如果无法进行热模拟，请猜想最有可能的环境条件，比如散热器和气流。
- 如果 XPE 估计的功耗超出了功耗预算，则可能需要对设计进行优化以符合功耗要求。
- 如果功耗估计值在预算内,但 TJA 超过允许的最高值，请考虑其他冷却技术 （如散热器或气流）和功耗优化技术以提高散热量并且降低期间内热扩散。根据这些新的工况参数重新用 XPE 进行估算，因为这些数值还会影响功耗。

系统级冷却策略

冷却策略可确保让器件产生的热量被环境提取和吸收。通常在设计初期会制定如下冷却策略，但在设计后期，这些策略的可行性会有所降低。这些策略会明显影响器件的静态功耗。如下冷却策略，如增加气流、降低环境温度，以及使用散热器 （或更大的散热器），或选择其它稳压器。

系统级供电策略

电压对静态功耗和动态功耗均有较大影响。主动控制电压水平可确保向器件施加想要的电压。

- 使用开关式稳压器。

  开关式稳压器比线性稳压器具有更高的电源效率，但需要使用更多的元件。

- 使用可调式稳压器。

  如用同一电源给多个 FPGA 器件供电，则让感应电压尽可能接近 FPGA 器件以及功耗最大的器件。

- 选择具有严格公差的稳压器。

  具有严格公差的稳压器可确保向器件提供稳定的电压。

测量功耗与温度

本节将简要介绍何测量 FPGA 器件的功耗和散热。其中一些方法会占用内部 FPGA 资源。而其它方法会使用单板或外部组件。有些应用需要在部署后积极监测和调整功耗与温度。而其它应用会在原型设计和验证阶段在实验室中运用这些测量方法。

功耗测量方法

功耗测量方法包括：

- 使用电流检测电阻
- 使用高级稳压器和数字电源控制器
- 执行机载监控
- 使用单独的电压

使用电流检测电阻

将一个电流检测电阻串联到稳压器输出端和 FPGA 器件之间，形成小幅电压降，根据欧姆定律，电压降与流经的电流成比例关系。通过 XADA 测量此处电压，便能计算出供给 FPGA 器件的电流。如需了解如何连接以获得所需测量精度，敬请参阅：《7 系列 FPGA 与 Zynq-7000 AllProgrammable SoC XADC 双 12 位 1 MSPS 模数转换器用户指南》(UG480) [参照 41] （又称 《XADC 用户指南》）。

使用高级稳压器和数字电源控制器

最新评估套件包括高级稳压器和数字电源控制器，您可以用数字电源控制器捕捉稳压器的输出电流和电压，然后将捕捉到的信息通过 USB 接口发送至监控计算机。这是最简便的电源监控方式。

大多数赛灵思开发板都集成有 TI UCD92xx 控制器，可用 PMBus(I2C)-USB 接口模块在 PC 上通过融合数字电源设计(Fusions Digital Power Designer)软件访问该控制器。

执行机载监控

赛灵思 7 系列器件具有内部传感器以及至少一个模数转换器，可测量供电电压和器件温度。通过 Vivado 硬件管理器可实时访问 JTAG，从而测量器件配置前后不同的供电压或器件结温 （见：图3-8)。您还可以将用您的代码对系统监控器或 XADC 组件进行实例化，以便在您的 FPGA 应用中访问这些测量值。

使用单独的电压

如有可能，给每个供电电压设置单独的电压。如将多个电压捆绑在一起，则在这些电压间测量功耗时请记录并做出备注。

热测量方法

热测量方法包括：

- 执行外部监控
- 执行机载监控

执行外部监控

由于器件封装会阻止接入芯片，因此无法直接测量结温。可通过测量封装、散热器和热电耦的其它位置的温度来估算结温。

还可以用热成像相机来可视化器件温度及其与邻近元件和大环境之间的散热作用。

执行机载监控

还可以用测量功率的方法来测量热量。在器件配置之前和之后您都可以使用 Vivado 硬件管理器 （图3-8）

您还可以在设计中用系统监控器/XADC 原语来读取器件结温。

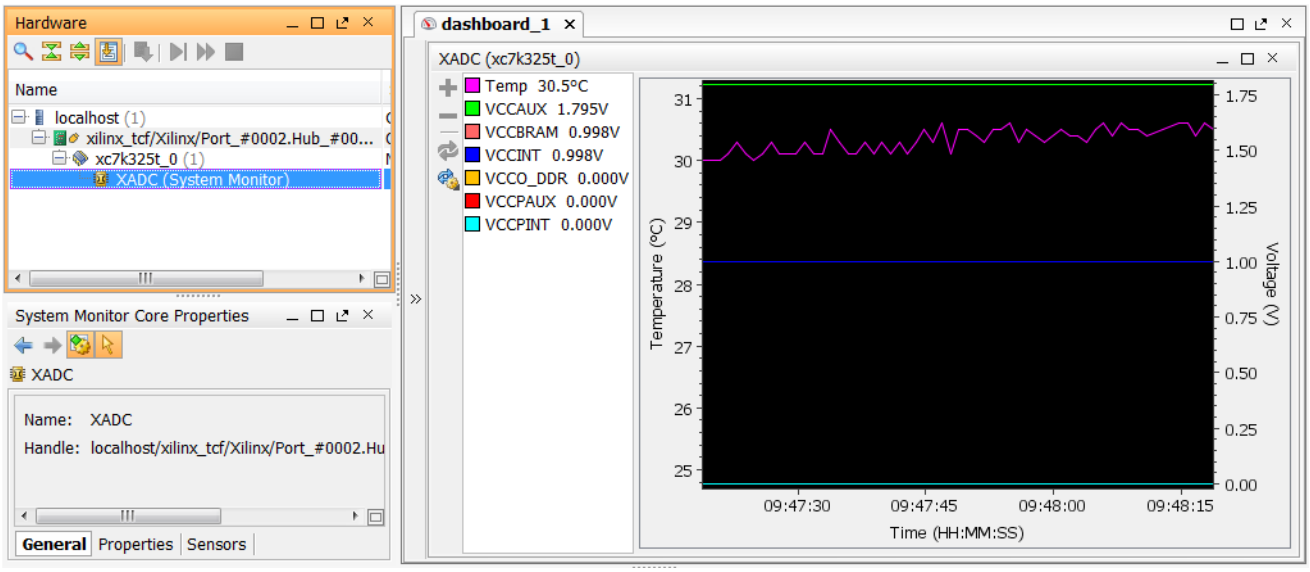


图 3-8 : 监控电压和结温

功耗与温度测量方法

为了估算影响设计总功耗的三大要素，您必须控制器件结温，并在测量前使其保持稳定。由于器件和设计静态功耗受器件结温影响极大，所以必须控制并稳定结温。

影响设计总功耗的三大要素：

- 器件静态
- 设计静态

- 设计动态

器件静态

下载一个 Blank 设计，确保：(1) 未捕捉到任何输入噪声；以及 (2) 所有内部逻辑与配置电路都处于已知状态。

注释： Blank 设计指的是带有单个门或永远无法触发的触发器的设计，其中所有输出均为三态配置。

等到结温稳定后测量 VCCINT、 VCCAUX 和您感兴趣的任何其它电源。通过特种设备——一支简单的热风枪或冷冻喷剂，您可以迫使温度变化，以评估环境对器件静态功耗的影响。

设计静态

将设计下载到 FPGA 器件上，不要启动任何输入或内部活动 （输入数据以及内外部时钟生成）。等到器件温度稳定后，测量您感兴趣的所有电源的功率。

从这些数值中减去器件静态功耗测量值，您便可以得出设计中使用的特定逻辑资源与配置的额外静态功耗 （设计静态功耗）。

设计动态

将设计下载到 FPGA 器件上，并提供设计的时钟和典型输入图案资料集群。等到结温稳定，然后测量您感兴趣的所有电源。

这一功耗代表的是设计的瞬时总功耗。它会随着每个时钟周期的活动变化而变化。

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

单板的设计应符合最坏情况的功耗要求。关于如何使用 Xilinx Power Estimator(XPE) 对最差情况下的功率进行分析，敬请参阅：《Xilinx Power Estimator (XPE) 用户指南》(UG440) [参照 23]。

设置期望值

了解总功耗要求有助于您定义电力传输和冷却系统的规范。您要确定以下各项数值：

- 供电电压
- 每个器件的功耗
- 被吸收的可生成热量的能量

XPE 可以回答上述这些问题。它可以帮您并行发 FPGA 逻辑以及焊接在器件上的印刷电路板。此外，还能让您了解预期裕量，从而确信建成之后系统将在功耗预算内运行。图3-9 显示的功耗信息，展示了 Xilinx Power Estimator 界面示例。

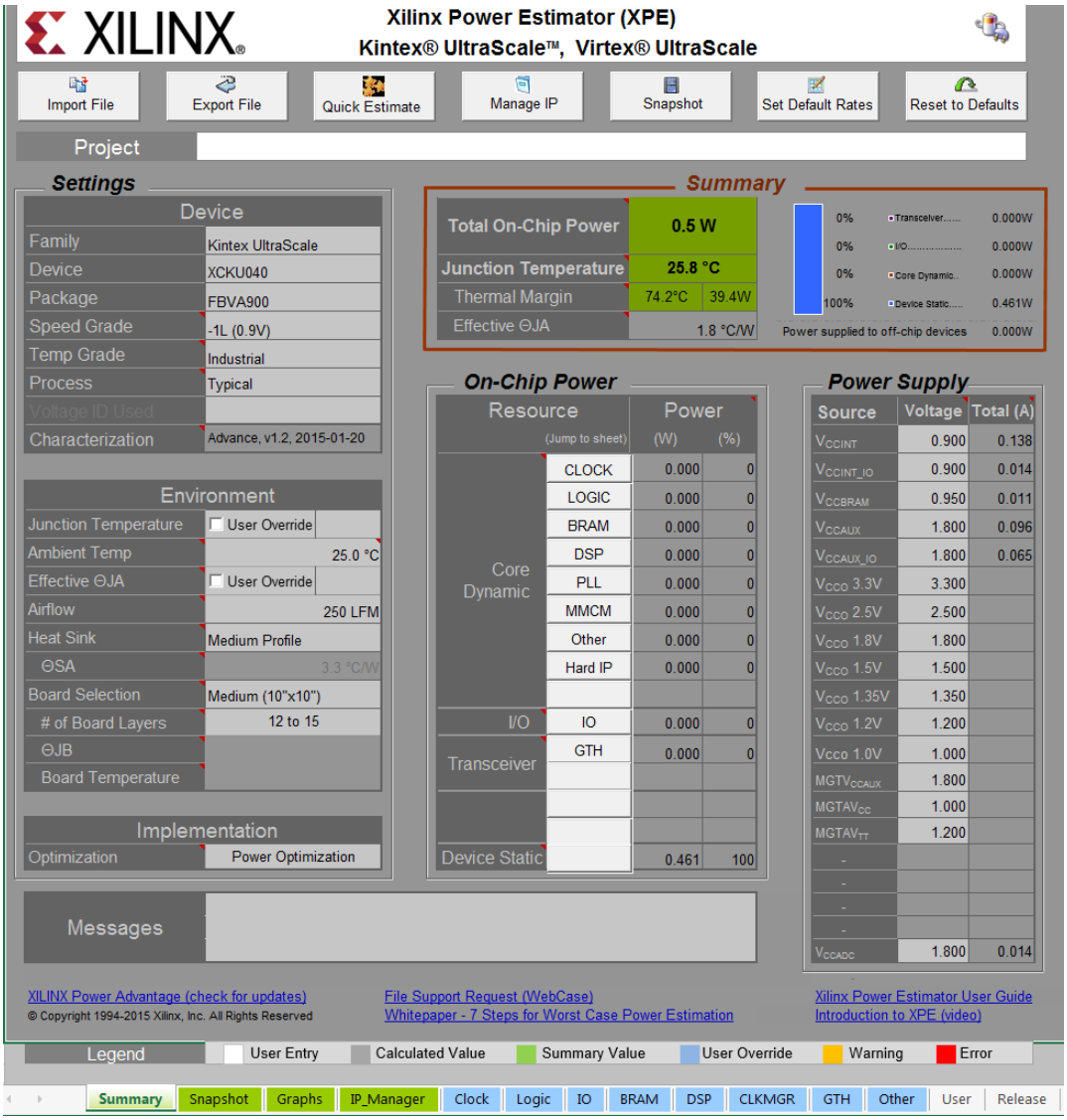


图 3-9 : Xilinx Power Estimator(XPE) 所展示的功耗信息

用 XPE 估算功耗

正确设置供电和冷却技术参数，确保构建能够可靠运行的系统。大多数情况下，应在 PCB 设计之前设置这些功耗和热性能参数。由于 FPGA 器件高度灵活， FPGA 设计通常在系统设计或 PCB 装配后才完成，有时甚至才刚启动。

这种顺序给 FPGA 设计师们带来了巨大战，因为功耗和热性能特性会随着芯片的比特流 （设计）、时钟和数据的输入发生显著变化。

如果功耗或热系统设计欠安全，可能会造成 FPGA 器件超出规范运行。这会造成器件未能按预期性能运行，还可能导致其它更严重的后果。

如果功耗系统超安全标准设计一般不会带来严重后果，但这种做法仍不可取，因为这样会增加不必要的成本，并让整个系统变得更加复杂。

在分析过程中，我们可以探索并运用多种功耗优化技术，这些技术可以大大节省功耗。我们将在 第 4 章：设计创建和 第 5 章：实现探讨这些技术。

使用 Xilinx Power Estimator (XPE) 可以在单板设计阶段进行功耗分析/估计。访问：《Vivado Design Suite 用户指南：功耗分析和优化》(UG907) 中的链接，了解有关如何使用 XPE 获得功率估算的更多信息。关于 XPE 方法的部分重点内容包括：

确保您采用的是最新版 Xilinx Power Estimator(XPE) 工具。功耗信息会定期更新以反映最新的功耗建模以及特性描述数据。您可在赛灵思网站能效页面获取最新版本的 XPE。

提供特定器件选择的准确信息。器件设置会给静态功耗和时钟功耗计算造成严重影响。

设置器件准备运行的环境条件。这些条件会影响散热性能，进而影响器件温度。而温度转而又影响功耗。

根据电源或调节器的容限，将电压提升到 FPGA 器件允许的最高值。

如果设计已经在 Vivado 工具中运行，从设计中导入 XPower 导出文件(.xpe)到 XPE 中，有助于填写资源信息。或如果设计之前的修订版本已经运行过，可将该修订版本作为分析的理想起点。

提示： 在导入Vivado Design Suite XPE 文件后，应检查数据是否正确并关联。把该信息作为理想起点，但并非完整解决方案。

如果具有可比性的 .xpe 文件不存在，按资源类型检查并 （如有必要）填写准备在设计中使用的资源，包括：

- 时钟树功耗
- 逻辑功耗
- I/O 功耗
- 块状 RAM 功耗
- DSP 功耗
- 时钟管理器 (CLKMGR)
- GT

审核每个含有翻转率、平均扇出或使能率的视图的设置值，并根据需要调整。

例如：

- 如果存储接口的训练模式程序在这个接口上使用可持续高翻转率，提高翻转率以反映这种额外活动。
- 如果电路某部分的时钟使能方式会降低电路的总体活动，降低翻转率。

如需了解有关翻转率判断方法的更多信息，敬请参阅：《Xilinx Power Estimator (XPE) 用户指南》(UG440) [参照 23]。

对逻辑扇出，应仔细考虑数据路径和控制路径的性质，例如：

- 在存在结构良好的串行数据路径的设计中 （比如 DSP 设计），扇出一般比设置的默省值低；
- 在存在大量数据执行路径的设计中 （比如某些嵌入式设计），可能会看到更高的扇出。

在审核结果之前，如有必要应按上述次序逐步进行。在完成这些步骤之后，再分析结果。

应确保不超过结温，且耗用的功耗不超过项目要求的预算。如果散热或功耗特性超出预算：

- 调整环境特性 （比如增大气流或添加散热器），或
- 调整设计的资源和功耗特性，直至得到可接受的结果。

许多时候需要做权衡取舍来求得更严格功耗预算下所需的功能。权衡取舍各种选择的时间最好是在设计流程早期阶段。当数据完全加载且该器件工作在所选等级允许的温度范围内，就可以使用 XPE 报告的功耗来设定设计的电压。

如果用户有信心加载的数据量不会太大，可以适当放大数值以避免把器件的电源系统设计地过于局促。

当设计成熟时，应不断审核和更新电子表格中的数据，以体现最新要求和实现详情。这样可以反映设计功耗的最新情况，还能根据设计当前的功耗趋势，及早发现是否有上调或下调功耗预算的需要。

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过程。

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 器件加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 FPGA 器件系列都提供有配置用户指南，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。例如：

- 《7 系列 FPGA 配置用户指南》(UG470) [参照 35]
- 《UltraScale 架构配置高级规范用户指南》 (UG570) [参照 42]

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，塞林思建议您确保 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。

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。它用于提示加电时初始化的完成，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。参见：FPGA 系列配置用户指南获得建议的上拉电阻值。

原理图检查清单包含以下推荐及其他关键建议。通过这些清单来识别和检查受推荐的板级引脚连接：

- 《7 系列原理图检查推荐》(XMP277) [参照 47]
- 《UltraScale 架构原理图检查清单》(XTP344) [参照 48]

第 4 章

设计创建

设计创建简介

您已经完成了器件 I/O 规划、 PCB 布局规划，并选择了 Vivado® Design Suite的使用模型，现在就可以开始创建设计。

需要考虑的重点包括：

- 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。

为满足上述目标，设计创建应：

- 规划设计的层级；
- 确定设计中需要使用和定制的 IP 核；
- 对于不能找到合适 IP 的互联逻辑和功能，为其创建定制的 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。

这个阶段的任何决策都会给最终产品带来广泛而深远的影响。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。在设计流程早期花一定时间创建精心规划的设计是值得的。这有助于达到预期的设计目标，并最大限度地减少实验室调试时间。

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。

靠近顶层调用I/O 组件

尽可能地靠近顶层调用 I/O 组件，以实现设计可读性。可调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。需要实例化的 I/O 组件也应该于靠近顶层处进行实例化，如差分 I/O （IBUFDS、 OBUFDS）和双数据速率寄存器 （IDDR、 ODDR、 ISERDES、 OSERDES）。

朝顶层方向布置时钟元件

朝顶层方向布置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 差异问题，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

寄存层级边界的输出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寄存输入通常而言，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。任何未寄存在层级边界上的路径，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。您可手动完成布局规划：

- 在使用 SSI 器件时，将逻辑分给特定的 SLR。以限定源寄存器、目的寄存器，或者两者。
- 在使用标准流程无法满足时序要求时，用于收敛设计的时序；
- 在使用部分配置等分层设计流程时。

如果元语或模块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。

如需了解有关布局规划的更多信息，敬请访问 《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修复。同样，出于功能调试 （及修改）目的，相关信号应限定在同一层级上。这样可以为探针探测和修改相关信号提供便利。

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，然后把属性传播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）不会从一个模块传播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，敬请参阅 《Vivado Design Suite 用户指南：层级设计》(UG905) [参照 20] 中的“设计约束”。

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。图4-1 显示了如何用有效的层级规划加快时序收敛。

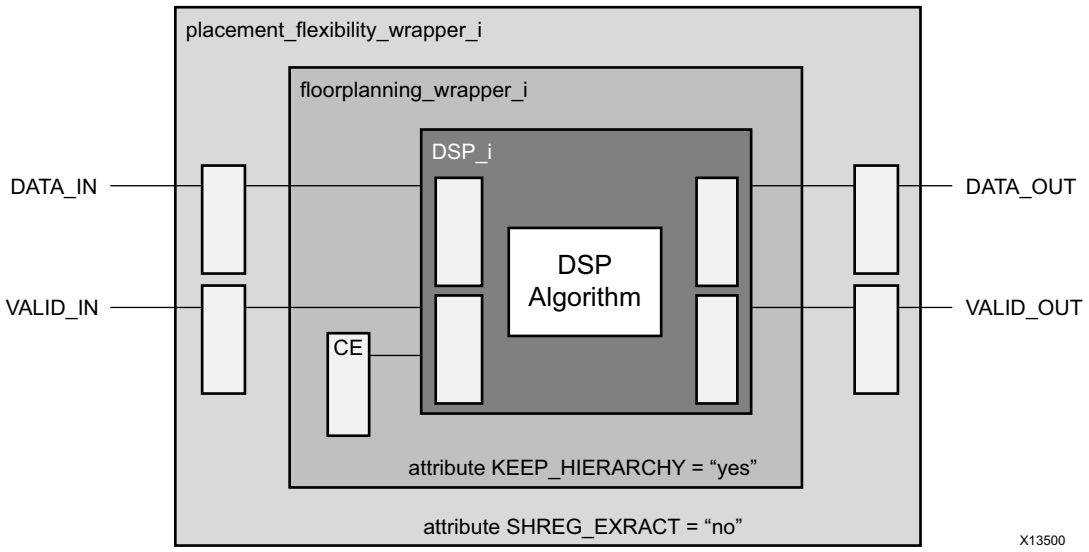


图 4-1 : 有效的层级规划实例

本设计部分分三个层级：

- `DSP_i`

  输入和输出同时寄存在 DSP_i 算法模块中。由于 FPGA 器件拥有丰富的寄存器，可以使用这种方法来改善时序预算。

- `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

您可能需要编写自己的定制 RTL，用于实现胶合逻辑功能或是某些未能找到对应的合适 IP 的功能。

基本功能

编写 RTL 的方法应保证能够可靠地实现功能。否则设计的功能会与 RTL 仿真结果大相径庭。RTL 应避免使用竞态条件，避免常见的编码漏洞，以免仿真结果和综合结果不匹配。

在编写可综合的 RTL 代码时，应遵循一些基本指南。有大量现成的有关可综合 RTL 代码的文献对这些指南都有论及。下文介绍的一些常见的指南，但并非全部。您也可运行一组 RTL DRC，如第 5 章中的检查 HDL 代码中介绍的。

阻塞赋值语句与非阻塞赋值语句

Verilog 阻塞赋值语句和非阻塞赋值语句使用不当，可能诱发竞态条件，从而导致 RTL 仿真与网表仿真不匹配。作为一般性规则，赛灵思建议对所有时序元件使用非阻塞赋值语句，对所有组合元件使用阻塞赋值语句。这样可以让仿真事件的序列更加符合预期，不易导致竞态条件。

敏感列表不完整

采用进程语句 (VHDL) 或 always 语句块 (Verilog) 的敏感列表是一个进程语句 (VHDL) 或 always 语句块 (Verilog) 所敏感的信号的一个列表。当列表所列信号的值发生变化，进程语句 (VHDL) 或 always 语句块 (Verilog) 就会触发并执行其语句。

在敏感列表不完整的情况下，虽然您可以得到想要的硬件，但RTL仿真和综合后仿真的结果会不同。此时某些综合工具可能会发出提示信息，警告有不完整敏感列表存在。在这种情况下应检查综合日志文件，在必要时应修复RTL代码。

下面的例子介绍的是使用一个进程和一个 always 语句块的简单 AND 函数。敏感列表完整，并生成一个 LUT。

VHDL 进程编码实例一：

```
process (a,b) begin
c <= a and b;
end process;
```

Verilog Always 语句块编码实例一

```
always @(a or b)
c <= a & b;
```

但是，如果从敏感列表中删除信号 b，虽然综合工具还是会生成组合逻辑 （AND 函数），但 RTL 仿真可能无法触发根据 b 的变化对 c 的评估。结果， RTL 仿真行为与实际电路行为不一。下面是警示信息的范例：

```
WARNING:[Synth 8-567] referenced signal <signal name> should be on the sensitivity
list [<file name>:<line number>]
```

在 Verilog 中，如果要定义一个组合性 always 语句块，应使用带星号的敏感列表：

```
always @(*)
```

这样会自动使用整个敏感列表所含内容。

注释 ：或者，在使用 SystemVerilog 时，结合使用 always_comb，这样无需指定敏感列表以及记录模块用途。

RTL 代码中的延迟

应避免在 RTL 代码中使用任何类型的延迟，不论是使用 `Wait` 或 `AFTER` (VHDL)，还是使用 `#delay` (Verilog)。延迟无法综合到组件中。在包含明确的延迟赋值的设计中，仿真设计中体现的功能与*综合后的*设计的功能往往无法吻合。

锁存器调用

综合器根据组合非时序逻辑中的不完整条件表达来对锁存器进行调用，比如：

- 不带 `else` 子句的 `if` 语句
- 不带上升沿或下降沿结构的寄存器

不带 else 子句的 if 语句的 VHDL 编码实例

```
process (G, D) begin
if (G='1') then
Q <= D;
end if;
end process;
```

不带 else 子句的 If 语句的 Verilog 编码实例

```
always @(G or D)
if (G)
Q = D;
```

经常会发生丢失分支条件或边沿的错误。应检查自己的综合日志，查看是否有正在被调用的锁存器。确定所有锁存器都是应要求调用的，而不是因疏忽调用的。

赛灵思建议您在 FPGA 设计中避免使用锁存器，因为使用锁存器会加大时序分析难度，即便能够通过仿真。

遵循综合工具技术文档中建议的编码方式，可避免发生调用锁存器的情况。

注释 ：如果在使用 SystemVerilog 时需要一个锁存器，赛灵思建议采用 always_latch 结构，以记录锁存器是否是预期结果。这有助于调试和代码审核。

复位规范不完整

在下面的代码段实例中，只有 `reg1` 在复位分支中有赋值， `reg2` 则没有。综合操作将假设在激活复位时， `reg2` 保持其值不变。因此复位信号会连接到 CE 引脚，从而创建另一套独有的控制集。参见控制信号与控制集。

```
always @(posedge clk)
  if (rst)
  reg1<= 1'b0;
else
begin
  reg1 <= din1;
 reg2 <= din2;
end
```

提示： 在使用复位的时候，应注意确保复位分支中不会发生漏掉寄存器的错误。

使用Vivado Design Suite HDL 模板

建议 ： 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 FPGA 器件架构。在许多情况下，使用这些模板既能简化设计，又能提升结果质量。

要从 Vivado Design Suite GUI 访问这些模板，应：

1. 转到“Windows > Language Template”。
2. 单击所需的模板。

高效率 HDL 编码

在代码中使用循环

为尽可能减少编码工作量， HDL 中常常使用循环。在调用硬件时，循环展开可能会导致结构效率低下，进而降低占位面积 和时序方面的性能。将未展开的逻辑映射到可用资源上，可能会导致实现方案欠佳。赛灵思建议使用该工具易于解读的结构来表达同一功能。

下面是使用 `for` 循环的优先级 MUX 代码实例：

```
always @(posedge clk) begin
  for(i=0;i<=3;i=i+1) begin
    if(en[i]) dout[i] <= i;
  end
end
```

同一功能也可以用 `case/if-else` 语句来编码，而且该工具更容易解读并生成有效的硬件。

不过在某些时候用 `for` 循环就可以在不影响结果质量的情况下达到要求的简明性 （例如在总线反转码中）。

在代码中使用循环的实例

```
reg [3:0] dout;
integer i;

always@(posedge clk)
  for(i=0;i<=3;i=i+1)
    dout[3-i] <= din[i];
```

提示： 用调用循环的方法实现基本连接功能是可以接受的。但在代码调用硬件资源 （非走线/互联）的时候，最好避免使用循环。

状态机指南

可以使用多种方法为状态机编码。遵守一定的编码方式，可确保综合工具有限状态机 (FSM) 提取算法能够正确识别和优化状态机，并提升电路的仿真、时序和调试性能。状态机的选择取决于目标架构及状态机具体的尺寸和行为。下面介绍的是几种不同实现方案的一些基本权衡。

- Mealy 型与 Moore 型

  众所周知，状态机有 Mealy 和 Moore 两种实现方式。Mealy 和 Moore 之间的主要区别在于 Mealy 状态机同时根据当前状态和状态机的输入来确定输出值，而 Moore 状态机则仅根据当前状态来确定输出值。

一般来说，用 FPGA 器件能够实现最佳的 Moore 状态机，因为大多数情况下 one-hot 状态机是首选编码方法，输出值几乎无需解码逻辑。

就二进制编码而言，有时使用 Mealy 状态机能实现更加紧凑或速度更快的状态机。但在不知道状态机具体状况的条件下，难以做出判断。

- **One-hot 编码与二进制编码**

  状态机设计有多种编码方法可供选择。FPGA 设计中最常见的两种是二进制法和 one-hot 法。大多数现代综合工具都包含有 FSM 提取算法，能够识别状态机代码，并选择最佳编码方法。有时手动编码方法可能更有优势，因为这样可以更好地控制设计，而且还有可能减轻设计实现后的调试工作。如需了解有关状态机提取功能的详情，敬请参阅综合工具技术文档。

- **安全与速度**

  必须了解，在对状态机编码时，要实现安全和速度这两个通常相互冲突的目标。安全的状态机实现方案指如果状态机收到未知输入或是进入未知状态，该状态机可能在下一周期恢复到已知状态，并从该恢复状态中复原。另一方面，如果不要这项要求（无需恢复状态），许多时候可以用较少逻辑更快地实现状态机。设计安全状态既包括在状态机的 Case 语句中编写一个默认状态，也可以是规定综合工具用“安全”模式实现状态机编码。如果需要安全状态功能，一般来说二进制编码更合适，因为二进制编码使用的未赋值状态较少。在实现安全状态机时，敬请阅读综合工具技术文档了解详情。

- **枚举类型**

  SystemVerilog 加入了一个新数据类型 enum （枚举的缩写），在许多时候有助于创建状态机。使用 enum 数据类型，可以让命名状态避免隐含地映射到寄存器编码。这样综合过程可以灵活地选用状态机编码方法，灵活地进行仿真，可以根据名称显示和查询具体状态，从而在总体上改善调试工作。出于这些原因，赛灵思建议如果选择 SystemVerilog 或 VHDL （一般都有此项功能）作为设计语言的话，应使用 enum 类型。

  请参阅： 《Vivado Design Suite 用户指南：综合》(UG901) [参照 16]。

避免保存层级边界

保存层级边界可形成硬化边界，这会对跨界优化造成负面影响。

请考虑下列代码段实例：

```
assign ored_signal = din[3]|din[2];
sub sub_inst (.clk (clk),
.din0 (ored_signal),
.din1 (din[1:0]),
.dout (dout));
endmodule

module sub …..
assign din_tmp = |din1 || din0;
endmodule
```

这里有两个“或” （OR）：

- 顶层的ored_signal
- 分支中的din_tmp

如果两者之间的层级边界使用综合属性或约束加以保存，这两个 OR 就无法合并，从而给设计的占位面积 和时序造成不利影响。如需了解更多信息，敬请参阅：定义理想的设计层级。

避免触发器发生沿混合

如果您使用时钟的正负时钟沿触发时序元件，则由这两种不同的极性触发的元件之间的路径只利用了半个时钟周期。这样会让时序更加紧张。

提示： 如果需要同时使用正负时钟沿采集或者提供外部 DDR 类型数据，请使用赛灵思 IDDR/ODDR 原语。

使用调试逻辑

编码效率决定着设计实现的效率。不必要的结构往往导致不必要的逻辑。在设计对设计功能没有必要，但对设计分析有用的调试信号或逻辑的时候，应牢记这点。许多时候这种调试代码在设计阶段有意义，但随着设计的成熟就变成无用的累赘。您在设计这种逻辑的时候，应注意既能让它起到调试作用，又不会保留在最终设计中。

要达成这个目的可以使用几种方法：

- 使用 `` `ifdef `` 参数保护逻辑，或使用设置可禁用或启用这些代码段的类属。
- 在编写逻辑的时候，注意采用便于将来将其注释掉的方法。
- 用模块或实体的单独调试版本，以备将来加以替换。

不管采用何种方法，思路不变：不仅有好的方法来调试设计代码和实现的硬件，还应在不需要逻辑的时候能够有好的办法加以去除，这非常重要。

如需了解有关调试方法的详细信息，敬请参阅：第 6 章：配置与调试。

阵列型端口声明

虽然 VHDL 允许把端口声明为阵列型，但出于下列原因，赛灵思建议您应避免采取这种做法：

- 与 Verilog 不兼容

  在 Verilog 中不存在把端口声明为阵列型的等效方法。这不仅会限制跨编程语言的可移植性，同时还会限制把代码用于多语言工程的功能。

- 无法存储和重新创建原始阵列声明

  如果在 VHDL 中把端口声明为阵列型，原始的阵列声明无法进行存储和重新创建。无论是电子数据交换格式 (EDIF) 网表还是赛灵思数据库，都无法为该阵列存储原始类型声明。因此当生成仿真网表时，没有关于说明该端口原本是如何声明的信息。

  生成的网表一般情况下有不匹配的端口声明以及产生的信号名称。这一点同样适用于顶层端口声明和层级设计的较低层端口声明，因为可以使用 KEEP_HIERARCHY 来试图保存那些网表名。

- 软件引脚名称的关联不当

  阵列型端口声明会导致软件引脚名称与原始源代码之间的关联不当。由于该工具必须把每一组 I/O 当作单独的标签对待，这样单列端口的名称就可能与您的期待不同。这会造成更加难以分析设计约束通过、设计分析及设计报告。

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。

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。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。

表 4-1 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。

表 4-1 : 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

提示： 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。

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。

经调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)
- 模块或 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 模块 RAM，也可能是调用出数千个寄存器。如不必要地在流水线上描述复位，结果可能是几个 SRL LUT，也可能是数百个寄存器，差异同样巨大。

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 模块上。在此类情况以及在其它情况下，会给资源数量造成显著影响。而且总体功耗和性能也会受到显著影响。

使用复位的时间和位置

FPGA 器件提供专用的全局置位/复位信号 (GSR)。这些信号能够在器件配置完成时，把所有寄存器初始化到 HDL 代码中初始值设定的状态。

如果没有设定初始状态，就将其默认为逻辑零。由此不管 HDL 代码中对复位拓扑做何种规定，在配置结束时每个寄存器都会处于一个已知的状态。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改善性能、缩小占位面积 和降低功耗。

建议 ： 评估每一个异步时钟，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。例如，如果简单的延迟线（移位寄存器）编写有复位功能，工具很有可能把延迟线映射到带有通用复位功能的寄存器集中。

如果省略复位功能，同一逻辑可能生成：

- SRL
- SRL 和寄存器组合
- 全部寄存器
- LUT 或 Block 存储器

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。
- DSP48 和块状 RAM 等部分资源只为模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模块中，否则会造成功能异常。
- 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。如需了解更多信息，敬请参阅：使用无时序约束复位。
- 在使用异步复位时，务必记住同步异步复位的失效。如需了解更多信息，敬请参阅：控制和同步器件启动。

- 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 BRAM、 LUTRAM、以及 SRL 存储内容翻转时，可以使用同步复位。

控制信号极性 （高电平有效与低电平有效）

对于时钟使能或复位等高扇出控制信号而言，最好是在整个设计中使用高电平有效信号。如果某个模块使用低电平有效复位信号或时钟使能信号工作，就会在设计中加入反相器，造成与之相关的时序延迟。在这种情况下，会限制选择综合选项或是要求重新构建综合选项以优化反相器，或要求实现定制解决方案。

赛灵思 FPGA 时钟使能信号和复位信号的 Slice 和内部逻辑为固有的高电平有效。描述低电平有效复位信号或时钟使能信号，会将更多LUT当作简单的反相器用于这些路由。

对于 UltraScale 器件来说，可在复位过程中使用可编程反相逻辑。因此，复位极性更加灵活。然而，赛灵思仍然建议保持复位极性编码一致 （全部高电平有效或全部低电平有效），以实现封装逻辑的最大灵活性。这一实现并没有反相逻辑，因此赛灵思建议始终描述为高电平有效。

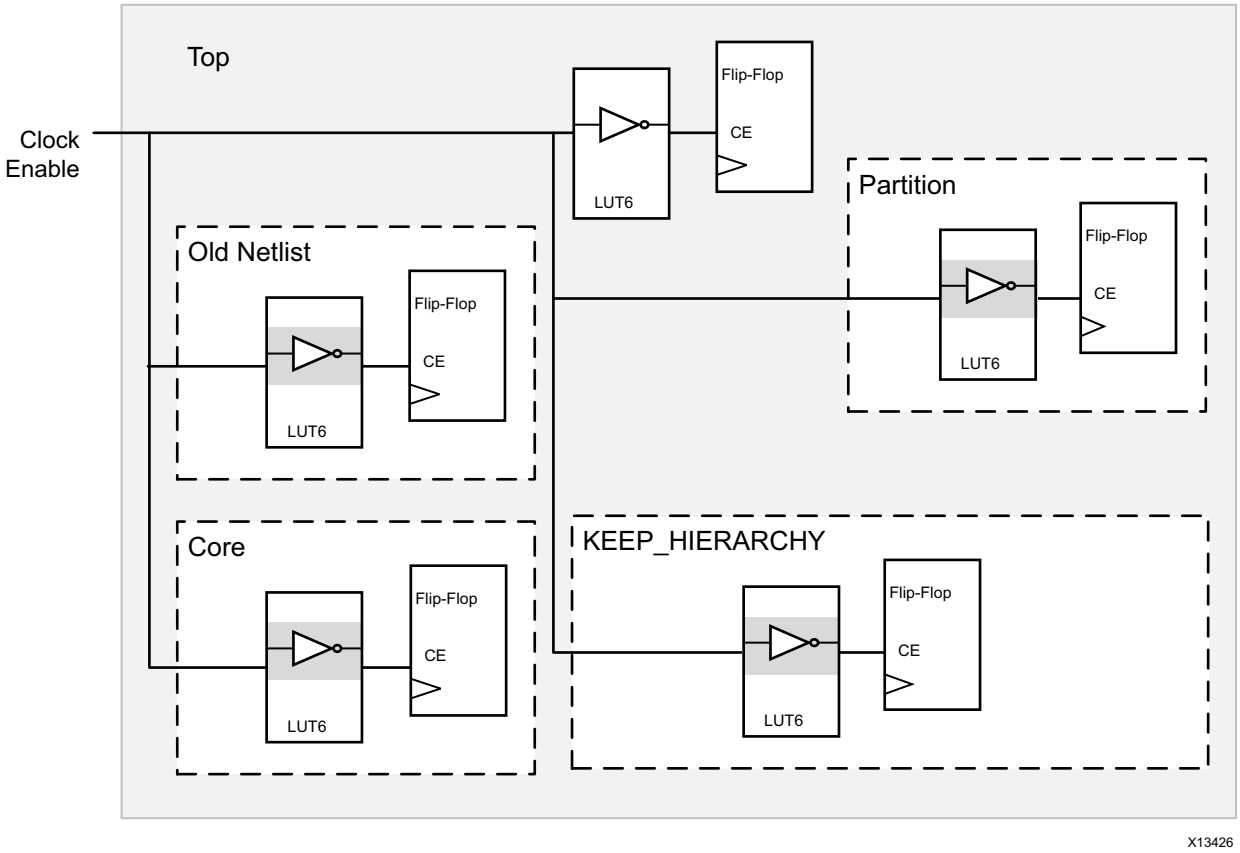


图 4-2 : 低电平有效控制信号造成的额外反相器

复位编码实例一

下面的实例编码构成了一个高度流水线化的乘法和极性生成函数：

```
// Reset synchronization
always @(posedge CLK) begin
  reset_sync <= SYS_RST;
  reset_reg <= reset_sync;
end
// Uses active-Low, async reset
// Also using an active-Low CE
always @(posedge CLK, negedge reset_reg)
if (!reset_reg) begin
  data1_reg <= 16'h0000;
  data2_reg <= 16'h0000;
  DATA_VALID <= 1'b0;
end else if (!NEW_DATA) begin
  data1_reg <= DATA1;
  data2_reg <= DATA2;
  DATA_VALID <= data_valid_delay[3];
end
// Uses an async reset when a reset is not necessary
always @(posedge CLK, negedge reset_reg)
if (!reset_reg) begin
  parity <= 4'h0;
  data1_pipe <= 32'h00000000;
  data2_pipe <= 32'h00000000;
  mult_data_reg <= 32'h00000000;
  mult_pipe <= 32'h00000000;
  mult_pipe2 <= 32'h00000000;
  mult_par_reg <= 36'h000000000;
  mult_par_pipe <= 36'h000000000;
  data_valid_delay <= 4'h0;
  DATA_OUT <= 36'h000000000;
end else begin
  data1_pipe <= data1_reg;
  data2_pipe <= data2_reg;
  mult_data_reg <= data1_pipe * data2_pipe;
  mult_pipe <= mult_data_reg;
  parity <= {^mult_pipe[31:24], ^mult_pipe[23:16],
  ^mult_pipe[15:8], ^mult_pipe[7:0]};
  mult_pipe2 <= mult_pipe;
  mult_par_reg <= {parity[3], mult_pipe2[31:24],
  parity[2], mult_pipe2[23:16],
  parity[1], mult_pipe2[15:8],
  parity[0], mult_pipe2[7:0]};
  data_valid_delay <= {data_valid_delay[2:0], NEW_DATA};
  mult_par_pipe <= mult_par_reg;
  DATA_OUT <= mult_par_pipe;
end
```

复位编码实例二

上述代码也可以重写成：

- 去除不必要的复位
- 修改异步复位为同步复位

- 修改低电平有效复位为高电平有效复位

```
// Reset synchronization, inversion moved here
always @(posedge CLK) begin
  reset_sync <= SYS_RST;
  reset_reg <= ~reset_sync;
end
// Notice the inversion above
// sync reset has become active High, though:
// from the top level port (SYS_RST) perspective,
// it is still active Low.
// Also changed to active-High CE
always @(posedge CLK)
if (reset_reg) begin
  data1_reg <= 16'h0000;
  data2_reg <= 16'h0000;
  DATA_VALID <= 1'b0;
end else if (NEW_DATA) begin
  data1_reg <= DATA1;
  data2_reg <= DATA2;
  DATA_VALID <= data_valid_delay[3];
end

// Removed unnecessary reset on datapath
always @(posedge CLK) begin
  data1_pipe <= data1_reg;
  data2_pipe <= data2_reg;
  mult_data_reg <= data1_pipe * data2_pipe;
  mult_pipe <= mult_data_reg;
  parity <= {^mult_pipe[31:24], ^mult_pipe[23:16],
  ^mult_pipe[15:8], ^mult_pipe[7:0]};
  mult_pipe2 <= mult_pipe;
  mult_par_reg <= {parity[3], mult_pipe2[31:24],
  parity[2], mult_pipe2[23:16],
  parity[1], mult_pipe2[15:8],
  parity[0], mult_pipe2[7:0]};
  data_valid_delay <= {data_valid_delay[2:0], NEW_DATA};
  mult_par_pipe <= mult_par_reg;
  DATA_OUT <= mult_par_pipe;
end
```

表 4-2 中显示了将第二个编码实例的实现方案与第一个编码实例的实现方案进行了对比。

表 4-2 : 针对 7 系列器件的编码实例对比

参数	结果
资源	对特定类型资源的依赖性降低 33%-75%
性能	提高 36%
时序端点数量	降低 40%
220 MHz 下的动态功耗	降低 40%

此外，第二个编码实例更简明。

复位编码实例三

有时设计可能需要进行低电平有效复位 （例如 AXI 标准就要求复位信号必须是低电平有效）。由于异步复位有同步电路来确保解除有时控，可以对同步电路稍作修改，这样就可以减少部分 LUT 数量。如需了解异步复位的同步电路的原理图实例，敬请参阅：控制和同步器件启动。

原始 HDL 代码:

```
always @ (posedge clk or negedge rst_n) //async. negedge reset
begin
  if (!rst_n)
    synchronizer_ckt <= 4'b0; // 4 stage reset syncornization
  else
    synchronizer_ckt <= {synchornizer_ckt[2:0], 1'b1};
end

assign synchronized_rst_n = synchronizer_ckt[3]; // the final reset signal which is
used to reset the actual flops in the design
```

复位路径上有一个 LUT，如图4-3 中的黑圈中所示。由于复位信号被馈送给许多寄存器 (flop)，忽略该 LUT 上的延迟会影响到许多路径。

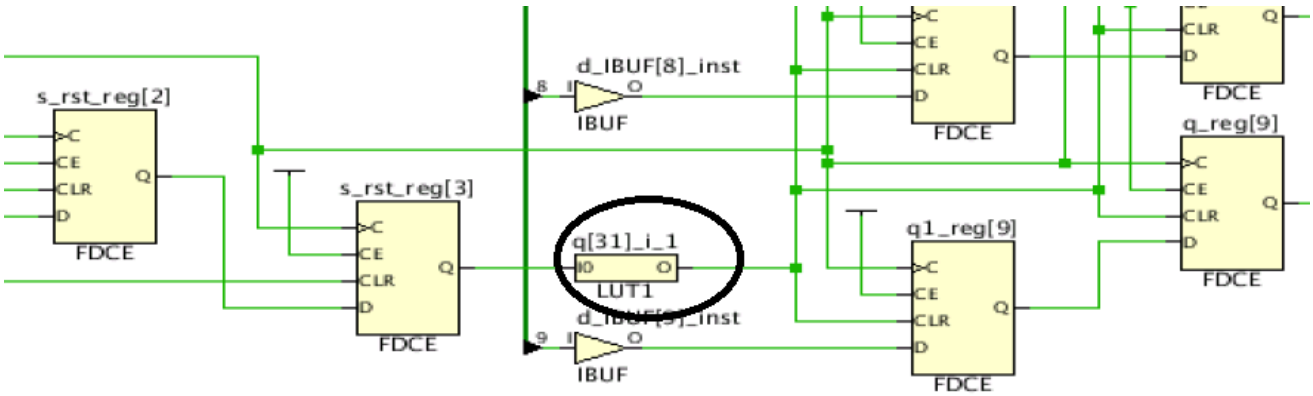


图 4-3 : 复位路径上的 LUT

修改后的 HDL 代码:

```
always @ (posedge clk or negedge rst_n) //async. negedge reset
begin
  if (!rst_n)
    synchronizer_ckt <= 4'hf // 4 stage reset syncornization
  else
    synchronizer_ckt <= {synchornizer_ckt[2:0], 1'b0};
end

assign synchronized_rst_n = ~synchronizer_ckt[3]; // the final reset signal which is
used to reset the actual flops in the design
```

Sychronizer_ckt 已赋予一个反相逻辑，最终的 synchronized_rst_n 也添加了另一个反相逻辑，以恢复极性。这种对同步电路的小幅修改可以去除同步电路和 （进入设计的寄存器中的）实际信号之间存在的 LUT，如图图4-4 所示。

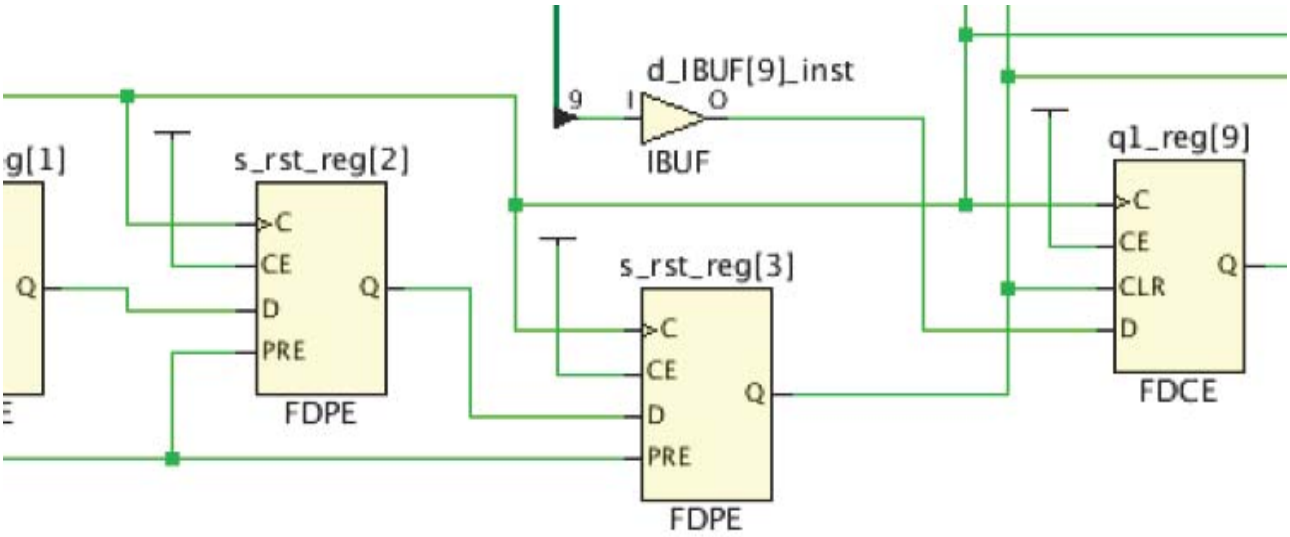


图 4-4 : 修改后的复位电路

切勿在同一进程或 Always 语句块中同时对置位和复位功能进行编码

赛灵思 FPGA 器件寄存器包含有置位或复位功能，但是不支持同时使用。因此在同时使用同步置位和复位功能时，会给数据路径添加额外的信号。这样可能会影响占位面积 和时序，但影响程度因布局、扇出和时序而异。出于这个原因，赛灵思建议只有在明确要求的情况下，才能把置位和复位功能编写在同一个时序模块中。

鉴于异步置位和复位给资源利用率和时序造成的影响更为显著，应避免使用。内含异步复位信号、异步置位信号和/或动态值异步控制信号的寄存器会导致电路无法正确计时。因此,赛灵思要求在需要异步置位和复位时至少有一个置位和复位条件应转换为同步活动。

时钟使能

如果使用得当，时钟使能能够显著地降低系统功耗，同时对占位面积 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：

- 占位面积 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

如果在同步模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，赛灵思建议在同步模块中的 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。不论是在同步条件下还是在异步条件下，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。

如需了解有关时钟的信息，敬请参阅：时钟。

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

  适当地将调用到 DSP 模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，增加这些模块的功耗，同时 DSP 中的流水线寄存器未得到利用。

- 不需要复位或多抽头点的移位寄存器或延迟线一般映射到移位寄存器 LUT 或 SRL 中。16 位或深度更浅的两个 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。

  要想充分利用 SRL，应避免在这些模块上使用复位。在无需使用复位的时候，器件利用率、性能和功耗都有望得到提升。

- 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，经验法则是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，六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 ：在设计早期阶段确定硬件资源的可用性以及高效利用硬件资源的方法，比在设计后期的时序收敛阶段才确定，更便于设计修改，从而实现更好的设计结果。

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，赛灵思强烈建议使用 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。敬请参阅：使用Vivado Design Suite HDL 模板。

- 直接实例化 RAM 原语

  优势：

  - 对实现方案有最高控制权限
  - 能访问模块的各项功能

  不足：

  - 代码可移植性差
  - 功能和用途冗长繁琐，难以理解
- 使用 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 Block RAM 的时钟输出时序。另外增加第二个输出寄存器也有好处，因为与块状 RAM 寄存器相比，Slice 输出寄存器具有更快的时钟输出时序。同时使用两个寄存器的总读取时延为 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。这样便于工具将块状 RAM 输出寄存器合并到原语中。

建议 ：应尽早判断在读取过程中是否容许发生额外一个时钟周期的时延。如果可以，为存储器阵列输出编码额外一级寄存器，专用于改善这些路径的总体时序。

- 使用专用模块还是分布式RAM

  RAM 可通过下列两种方式实现：（1）专用块状 RAM；或者 （2）在 LUT 中使用分布式 RAM。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 的两位，深度达到 64 位的映射为每个 LUT 的一位。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 Block 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 Block RAM 的数量与结构。

重要提示 ：这些模块的编码方法如稍有偏差，就会导致资源利用率欠佳。例如存储器异步读取会调用 LUTRAM 而非块状 RAM。但是添加存储器复位功能会导致实现在一个寄存器阵列中，而不是 LUTRAM 中。

选择合适的块状 RAM 写入模式

赛灵思块状 RAM 能够改变写入模式。这会对功能、行为和功耗产生影响。赛灵思建议根据下列指南为特定操作选择最佳写入模式：

- 首先考虑功能

  在选择写入模式时，应首先考虑功能。在写入到块状 RAM 的特定端口时，是否需要输出端读取特定值的数据？如果在写入过程中必须看到块状 RAM 中之前的值，请选择 READ_FIRST。如果需要读取正在写入块状 RAM 中的新数据，请选择 WRITE_FIRST。如果不关心写入过程中的数据读取，那么接下来的选择标准就是处理内存争用。

- 使用 READ_FIRST 模式

  如果实现的是双端口存储器，且将同一时钟连接到该块状 RAM，但无法保证不会发生内存争用，请选择 READ_FIRST。 READ_FIRST 模式可确保在同一时钟同时连接到块状 RAM 的两个端口时，不会发生内存争用。如需了解更多信息，敬请参阅： 《7 系列 FPGA 器件存储器资源用户指南》(UG473) [参照 38]。

- 使用 WRITE_FIRST 模式

  在位宽 SDP 模式下（RAMB36 在 72 位宽或 RAMB18 在 36 位宽模式）使用 7 系列 Block RAM，如使用 WRITE_FIRST 模式，则能避免不同的时钟在两个端口上或发生读/写冲突。在位宽 SDP 模式下，该模式是功耗最低的运行模式，因而我们建议使用这一模式。

- 使用 NO_CHANGE 模式

  在所有的其它情况下，赛灵思建议使用 NO_CHANGE 模式。 NO_CHANGE 模式具有最佳功耗特性。如果在功能写入过程中读取 （如果不考虑冲突问题），那么在 NO_CHANGE 模式下块状 RAM 及相关互联电路的功耗较低。

FIFO创建

先进先出 (FIFO) 缓冲器是 FPGA 设计中最常用的存储器之一。

注释 ：异步先进先出 (FIFO) 缓冲器也称为*异步 FIFO* 或*多速率 FIFO*。

FIFO 缓冲器常用于将数据从一个时钟域传输到另一个时钟域。在创建和使用 FIFO 时，有多种方法可供选择，我们可以权衡利弊，综合评估。

为 FIFO 选择合适的输入方式和资源

赛灵思 FPGA 器件内置的块状 RAM 具备专用的 FIFO 电路。一般情况下赛灵思建议使用该专用 FIFO 电路，以实现最佳的占位面积 、功耗、性能和 MTBF 特性。

使用硬 FIFO，无需考虑额外的时序约束或内存争用，从而可简化设计。但如果该电路不能满足您的需求，可以通过创建软 FIFO 来实现各种行为和特性。

如果需要使用软 FIFO，建议最好通过IP 目录来创建。这样不仅可以为最常用的 FIFO 实现方案创建合适的逻辑，达到简化设计的目的，还可以建立适当的时序约束和属性，确保正确实现和分析。如果需要终极定制，还可以采用调用。

用软实现方案实现异步 FIFO 缓冲器面临的设计挑战

如果使用软 FIFO，需要考虑下列因素。为判断 FIFO 状态并安全地传输数据，设计必须监控和响应状态标志 （空和满信号）。

由于这些标志基于没有相位或周期关联的两个时钟域，难以随时确定标志的时序和可预测性。为此，在使用异步 FIFO 时，必须采取特殊的预防措施。

对大多数异步 FIFO 实现方案而言，标志有效和失效在本质上不具备周期确定性。虽然功能或时序仿真可以显示状态标志在一个时钟周期内的变化情况，但在 FPGA 器件本身上，状态标志可能会在前一周期或后一周期发生改变。发生这种情况的原因可能是仿真器中事件的时序和顺序与 FPGA 器件中事件的时序和顺序不一致。

FPGA 器件的最终时序是由工艺、电压和温度 (PVT) 决定的。因此在不同芯片上，或是同一芯片在不同环境条件下可能发生周期差异。您在设计电路时必须考虑到这些差异。

如果想要数据在一定数量的时钟周期中或之后后有效，而您又不直接监控空和满标志，就会遇到一系列问题。在大多数 FIFO 实现方案中，即使有内存空间，从仍含有效空标志的 FIFO 读取或是向带有有效满标志的 FIFO 写入，都会造成无效的读取或者写入条件。这样会导致结果出乎意料并造成严重的调试问题。赛灵思强烈建议始终监控状态标志，无论异步 FIFO 的实现是否通过仿真。

在大多数异步 FIFO 实现方案中，如果同时或近乎同时在状态标志边界上执行读写操作，空标志和满标志会默认为安全条件。即便 FIFO 实际并未满，也会激活满标志。即使 FIFO 实际并未空，也会激活空标志。这样可以提供少许安全，避免发生标志不被激活的风险。

在测试异步条件时，可借助各种综合和仿真命令，让异步 FIFO 按已知方式执行。

在设计 FIFO 标志逻辑时，许多情况下无法避免时序违规。如果在时序仿真过程中发生时序违规，仿真器会产生已知的 (X) 输出来说明未知状态。出于这个原因，如果逻辑是用已知异步源驱动的，则应采取适当的设计预防措施，以确保即便发生时序违规也能正常运行，赛灵思建议为 FIFO 标志逻辑中的相关同步装置增添 ASYNC_REG=TRUE 属性。这个属性用于提示寄存器能够安全地接受异步输入。此时寄存器上的时序违规不会产生 X 输出，而是保持其先前的值。这样也可以避免工具复制寄存器或是进行其它可能对寄存器的运行造成不利影响的优化。

如果在写入某个存储位置的同时读取这个位置，就会发生内存争用。通过有效地使用标志 （空和满），可以尽量避免发生内存争用。否则读取的数据可能出错。如果在设计中已经采取有效措施防止读取因冲突产生的错误数据，就可以使用 RAM 模型上的 SIM_COLLISION_CHECK 属性禁用冲突检查功能。这样做能够加速仿真模型，但是赛灵思建议仅在不会发生内存争用时使用此方法。

提示：
- 使用 Vivado 工具中提供的 HDL 模板。
- 判断 Block 存储器和分布式存储器哪一个更适合存储器功能。
- 尽量使用输出寄存器。
- 避免在存储器结构周边使用异步复位。
- 根据电路要求考虑最佳写入模式。
- 对 FIFO 实现，首先考虑专用硬 FIFO。

为适用 DSP 和算术调用进行编码

赛灵思 7 系列 FPGA 中的 DSP 模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 模块是一款具有多重寄存器级的高度流水线化模块，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 模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。

赛灵思 7 系列 DSP48E1 Slice 中的乘法器的输入位精度对有符号数据是 18 位 x 25 位。因此，对无符号数据的位精度是 17 位 x 24 位。对于 UltraScale 器件来说， DSP48E2 Slice 中的乘法器的输入位精度对有符号数据是 18 位 x 27 位。因此，对无符号数据的位精度是 17 位 x 26 位。对 Verilog 代码而言，除非在代码中声明，数据被当作无符号处理。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，应对 DSP 资源进行调用。

有关 DSP48 Slice 特性和功能的更多信息，以及了解如何根据自己的设计需要充分利用该资源，敬请参阅： 《7 系列 DSP48E1 Slice 用户指南》(UG479) [参照 39] 及 《UltraScale DSP Slice 用户指南》(UG579) [参照 40]。

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 LUT 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 造成的时延之后，就会在 Q 输出上提供新的位位置值。同步和异步置位/复位控制信号未在 SRL 原语中提供。

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。由于综合工具一般能为正确编码的移位寄存器调用编码自动调用该寄存器，除非该资源被实例化或综合工具被阻止调用这样的寄存器，所以没必要处理额外的工作。

为了调用 SRL，您不应编码置位/复位功能。赛灵思建议采用 Vivado Design Suite HDL 模板中的 HDL 编码方式进行编码。

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”
```

如需了解有关综合属性以及如何用 HDL 代码设定这些属性，敬请参阅： 《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 编码实例一：

```
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 编码实例一

```
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 编码实例二

另外还可以在 Verilog 中使用初始声明：

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

为确保所有时序元件同时脱离复位，敬请参阅：控制和同步器件启动。

时钟

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。

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。如果您使用的是其它某种架构，敬请阅读时钟资源技术文档，了解相关架构。

Virtex-6 和 Virtex-7 架构内含 32 个称为 BUFG 的全局时钟缓冲器。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

  全局时钟缓冲器 (BUFG) 元件一般供时钟使用。这种全局时钟缓冲器带有附加功能。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冲器用于 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，并在总体时钟架构构建和规划过程中加以解释。

  如需了解如何使用这些组件的更多信息，敬请参阅： 《时钟资源用户指南》和 库指南》，以了解特定器件的情况。

区域时钟资源

除全局时钟资源之外，还有区域时钟资源：

- 水平时钟域缓冲器 （BUFH、 BUFHCE）

  水平时钟域缓冲器（BUFH、BUFHCE）既可单独使用，也可与 BUFG 配合使用。使用这些缓冲器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。这样就可以访问全局时钟网络未使用部分的低偏差资源，供位于时钟域内部的较小时钟域使用。 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。

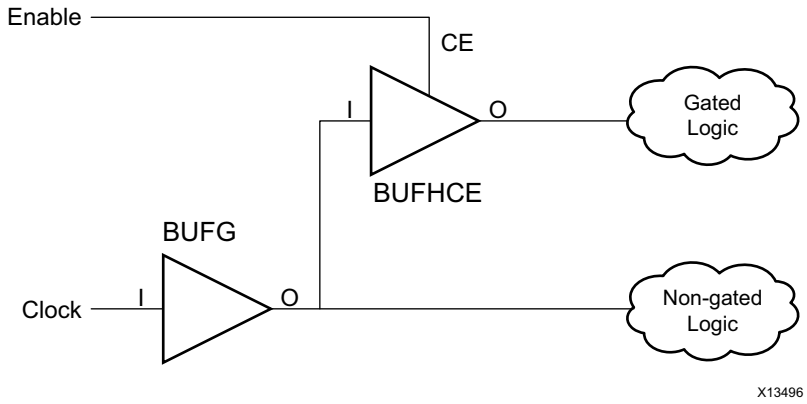


图 4-5 : 水平时钟域缓冲器

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。

- 区域缓冲器 (BUFR)

  区域时钟缓冲器 (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。这使得这种缓冲器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 时钟缓冲器 (BUFIO)

  I/O 时钟缓冲器 (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。

- 多区域时钟缓冲器 (BUFMR)

  多区域时钟缓冲器 (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）。

  关于赛灵思 7 系列 FPGA 器件的时钟资源的更多信息，敬请参阅： 《7 系列时钟资源指南》(UG472) [参照 37]。

关于SSI 器件的更多时钟考虑

通常，上述所有时钟考虑事项同样适用于 SSI 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。如前一节提到的，除使用 BUFMR 的情况,区域时钟均可等同视之。此时区域时钟不能跨越 SLR 边界驱动时钟资源。相应地，赛灵思建议用户把负责驱动 BUFMR 到单元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缓冲器，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冲器，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。

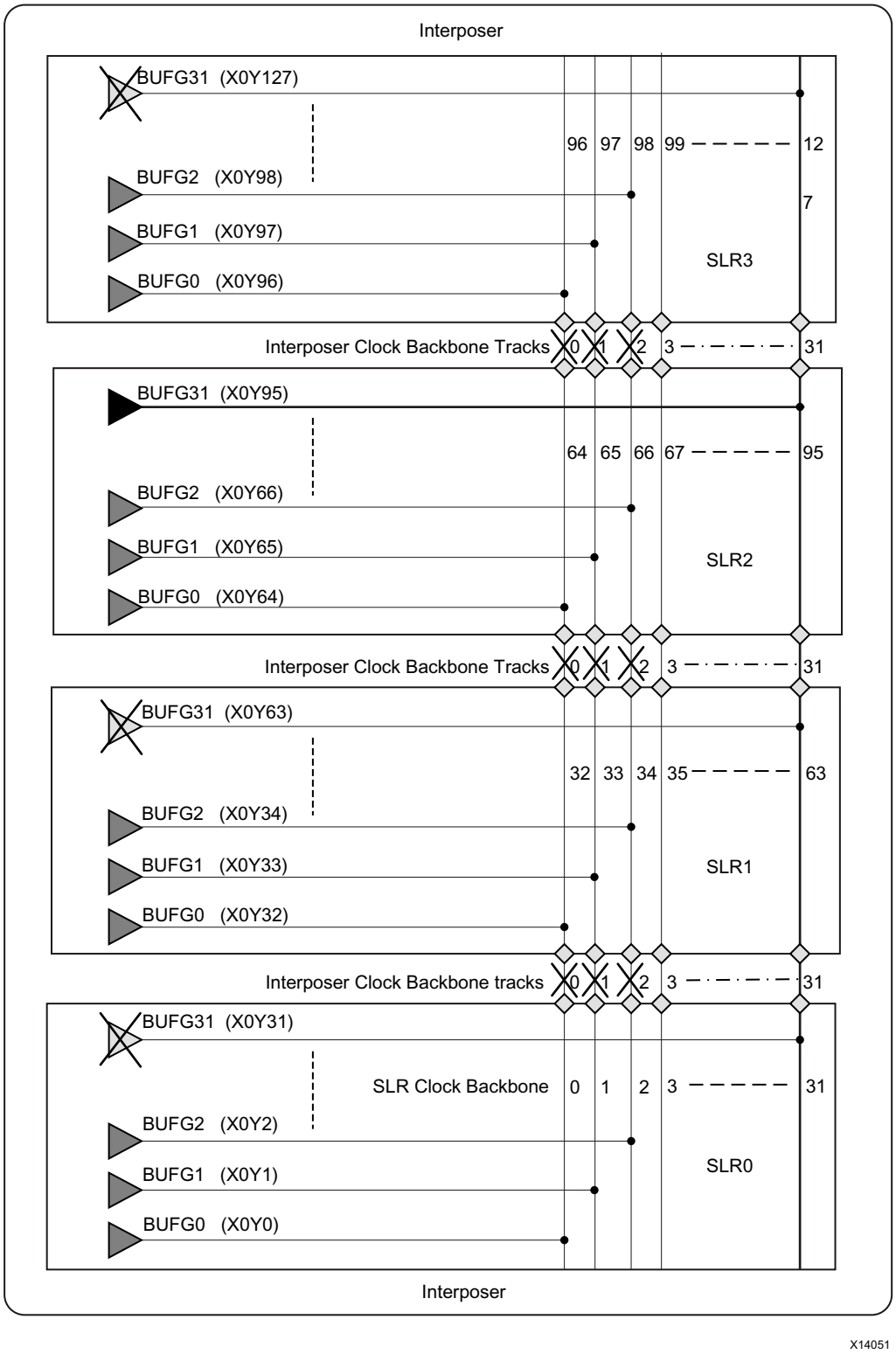


图 4-6 : SSI 器件时钟线上的隔离选择

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概念的界限变得模糊。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冲区与时钟布线结构，无论负载是本地/区域还是全局的。时钟架构根据设计能自动使用仅连接电路所必需的时钟资源。利用这种结构，可以支持架构中更多的时钟，同时优化性能及功耗等时钟特性。即根据驱动程序和使用情况，共有三种时钟类型和相关的时钟结构：

- 高速 I/O 时钟

  这些时钟与高速 Select-I/O 位片(bit slice) 逻辑相关联，由PLL生成，并驱动到位片逻辑的专用低抖动布线，旨在进行高速数据采集和传输。整体而言，此时钟架构采用诸如 MIG 或高速 I/O 向导等赛灵思 IP 来创建和控制，通常不由用户设定。

- 普通时钟

  该结构用于大多数时钟应用，可由 I/O （建议 GCIO）、 MMCM/PLL、 BUFGCE/BUFGCTRL 或通用逻辑 （一般不建议）驱动。时钟网络进入点可通过 I/O 存在的任何时钟域的 BUFGCE、 BUFGCE_DIV 或 BUFGCTRL 进行定位。任何给定时钟域都能支持多达 24 个独特时钟，大多数 UltraScale 器件能支持 100 多个时钟，具体取决于时钟使用和布局。

- 千兆位收发器 (GT) 时钟

  传输和/或接收用的时钟以及用于千兆位收发器 （GTH 或 GTY）的参考时钟，使用与 GT 相关联的时钟域中的专用时钟。这种时钟不仅能为收发器提供连接，而且能驱动到使用 BUFG_GT 组件的通用时钟网络，并用于器件上任何位置的其他逻辑的时钟。

UltraScale 器件时钟缓冲器

UltraScale 器件有以下类型的时钟缓冲器。

- BUFGCE

  最常用的缓冲器是 BUFGCE。这是具有时钟使能/禁用特性的通用时钟缓冲器，完全等效于 7 系列 BUFGCE。

- BUFGCE_DIV

  在需要简单时钟分频时，该缓冲器比较有用。这种缓冲器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。它通常用来替代 7 系列器件的 BUFR 功能，不过由于它能驱动全局时钟网络，因此其功能比 BUFR 组件要强得多。

- BUFGCTRL （和 BUFGMUX）

  BUFGCTRL 可实例化为 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需求。

- 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。此外，它也能连接使用以满足区域或全局时钟需求。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。如需了解有关缓冲器连接和使用详情，敬请参阅可用的 《UltraScale 架构收发器用户指南》。

UltraScale 器件时钟布线、根和分配

要全面了解 UltraScale 器件时钟网络，我们必须首先介绍时钟布线、时钟根和时钟分配等概念。时钟布线是专门的布线结构，从时钟缓冲器发送时钟到时钟域中心点，并分配给负载。每个时钟区域有 24 个垂直和 24 个水平时钟布线，能根据布线需求，与任何相邻区域连接或隔离。对 SSI 器件而言，这包括相邻但在不同 SLR 上的时钟区域。

时钟网络的中心点就是时钟根。时钟根可以是器件中的任何时钟区域，通常根据负载选择，从而得到时钟域的最低偏差。时钟根是时钟布线的末端，并且连接到时钟分配。时钟分配是与时钟布线相分离的布线层，能将时钟分配给所有的负载。与时钟布线类似，每个时钟域中有 24 个垂直和 24 个水平时钟分配连接，可与相邻时钟域连接或隔离，包括 SSI 器件中相邻 SLR 中的区域。

图4-7 显示的是时钟如何从时钟源进行分配。

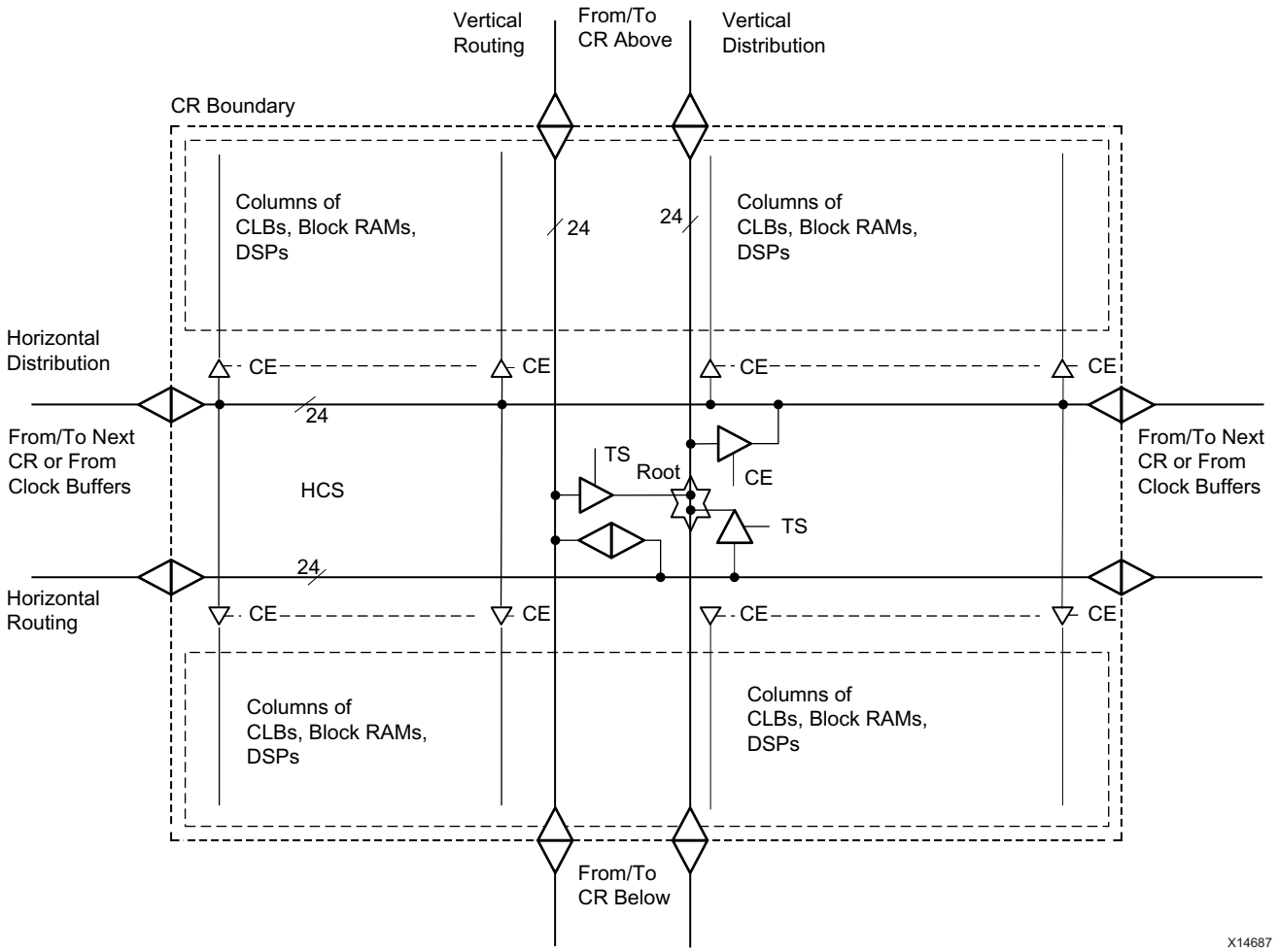


图 4-7 : 时钟区域计时

控制时钟缓冲器布局布线

我们通常建议让 Vivado place_design 自动管理时钟缓冲器布局和时钟根分配。这能提高时钟优化，让布局器灵活地避免时钟争用。总体而言，我们不建议在时钟缓冲器上布局 LOC。这样做会迫使时钟采用某个时钟布线和/或分配路径，如果管理不当，就会导致时钟无法合规布线的布局方案。 Vivado place_design 需要灵活进行缓冲器布局，从而减少时钟争用。如果需要，我们可在时钟缓冲器上设置 CLOCK_REGION 属性来替代 LOC，以指定某个时钟区域作为缓冲器布局，但并不是该区域的特定位置。这就能提供足够的灵活性，确保 Vivado 工具能有效管理时钟合规性问题。有关本属性的信息请见：Vivado Design Suite Properties Reference Guide (UG912) [参照 14]。

同样，我们建议让 Vivado 去选择时钟根，从而实现最佳性能选择，同时保持合规的时钟布线。不过，如果需要的话，也可通过时钟网络上的 CLOCK_ROOT 属性控制时钟根。指定 CLOCK_ROOT 会迫使时钟从时钟布线层转到指定时钟区域的分配层，这就会影响时钟的布线和性能 (偏差) 特性。

SSI 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 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事项，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冲器 (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冲器。

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。

如需了解更多支持和命令信息，敬请查阅：参数、属性和约束。

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)
- 时钟修改模块，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，敬请参阅：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冲器而言，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。敬请参阅：使用Vivado Design Suite HDL 模板。

控制时钟的相位、频率、占空比和抖动

本节将介绍对时钟特性的一些精细粒度调试：

- 使用时钟修改模块 （MMCM 和 PLL）
- 在时钟上使用 IDELAY 控制相位
- 使用门控时钟

使用时钟修改模块 （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）。

此外， MMCM 还可用于：

- 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。
- 校正或修改时钟占空比，从而对设计的重要方面进行严格控制。

对调整和控制时钟特性而言，使用 MMCM 或 PLL 是常见的做法。

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。

另外， MMCM 或 PLL 也可以直接实例化，以便更加严格地控制。但应注意使用正确的设置。如果 MMCM 或 PLL 设置不当，可能会：

- 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。

不过根据用户的目标， Clocking Wizard 中的设置可修改为：

- 进一步降低抖动、改善时序，但会增加功耗，或
- 进一步降低功耗，但会增大输出抖动。

在使用 MMCM 或 PLL 时，应注意下列问题：

- 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。
- RST 应与用户逻辑相连，这样就可以按照 7 Series FPGAs Clocking Resources User Guide (UG472) [参照 37] 中介绍的方法进行激活。如果时钟被中断，就会对 RST 接地造成问题。
- “LOCKED”输出应用于实现复位。例如：由 PLL 的输出时钟进行时钟控制的同步逻辑应保持在复位状态，直至激活 “LOCKED”。 “LOCKED”信号在用于设计同步部分之前需要同步。
- 只有在 PLL/MMCM 输出时钟需要与输入参考时钟相位对齐时，才需要在反馈路径中使用 BUFG。
- 确认 CLKFBIN 和 CLKBOUT 之间的连接。

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 FPGA 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 代码中隐含精细粒度时钟门控技术，则会干扰此功能及到该专用时钟网络的映射。因此在直接针对 FPGA 器件编码时，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。

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。

如果需要把时钟网络的较大部分关闭一段时间，可以使用 BUFGCE 或 BUFGCTRL 来启用或禁用时钟网络。或者，对于 UltraScale 器件而言，还可对 BUFGCE_DIV 与 BUFG_GT 进行门控。对于 7 系列器件来说， BUFHCE、 BUFR 以及 BUFMRCE 也能用于门控时钟。如果在一定时间段内需要降低时钟速度，也可将这些缓冲器与额外逻辑配合使用，以定期启用时钟网络。用户也可以使用 BUFGMUX 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，某种技巧可能比另一种更加行之有效。例如：

- BUFR 的最佳使用方法是将其用作外部生成时钟 （低于 450MHz），只给不超过三个时钟域提供时钟。
- 对 Virtex-7 器件，如果要将门控时钟用于一个以上的时钟域 （但不超过三个垂直相邻的时钟域），还需要使用一个 BUFMRCE。
- BUFHCE 更适用可限定在单个时钟域中的更高速时钟。虽然 BUFGCE 可能会横穿整个器件 （也是最灵活的），但对最大程度地降低功耗而言并不是最佳选择。

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递时间，降低器件提出的外部时序要求。

时钟资源选择总结

BUFG

- 必须在将高扇出时钟提供给贯穿器件的多个时钟域时使用。如果看到级联 BUFG，应确保是否需要使用，或是否是无意使用 BUFG。
- 在不适合实例化时钟或不适合手动控制时钟的时候使用。
- 用于超高扇出非时钟网络，比如不存在混合极性的多个中低速时钟全局复位。赛灵思建议此用途在任何设计中不超过两处。
- 对时钟必须跨越一个以上 SLR 的 SSI 器件，将 BUFG 布置在其中一个中心 SLR 中。这样能够更加均匀地在整个器件中分配时钟网络，从而最大程度地降低偏差。

BUFGCE

- 用于停止高扇出、多区域的时钟域。

BUFGMUX/BUFGCTRL

- 用于在设计运行中修改时钟频率或时钟源。

BUFGCE_DIV （仅针对 UltraScale 器件）

- 用于生成简单的时钟分频

BUFG_GT （仅针对 UltraScale 器件）

- 当时钟由 GT 或 GT 专用参考时钟提供时使用。

BUFH （仅针对 7 系列器件）

- 用于可限定在单个时钟域中的较小时钟域逻辑。
- 用于超高速时钟域
- 用于与 BUFG 争用时钟资源可能性较小的时钟域
- 对 SSI 器件，赛灵思建议一般使用位于上部或下部的 SLR，以减少与布局在中心 SLR 中 BUFG 争用资源的几率

BUFHCE （仅针对 7 系列器件）

- 用于可布局在单个时钟域中时钟网络的中等粒度部分的时钟门控。该 BUFHCE 可通过 BUFG 驱动。
- 用于可限定在单个时钟域中的高扇出非时钟信号，如复位。

BUFR （仅针对 7 系列器件）

- 用于性能要求不高于 450MHz 的中小规模时钟网络。
- 用于可限定在最多三个垂直相邻时钟域内且要求时钟分频的外部时钟。
- 对 SSI 器件，赛灵思建议一般使用位于上部或下部的 SLR，以减少与布局在中心 SLR 中 BUFG 争用资源的几率。

BUFIO （仅针对 7 系列器件）

- 用于一般源同步数据采集使用的外部高速 I/O 时钟。

BUFMR （仅针对 7 系列器件）

- 用于在使用单个时钟源时，将 BUFR 或 BUFIO 用于一个以上的垂直相邻时钟域的情况。
- 对 SSI 器件，赛灵思建议将 BUFMR 和相关引脚布置在 SLR 内的中心时钟域。如果需要，可以从该 BUFMR 访问全部三个时钟域。

BUFMRCE （仅针对 7 系列器件）

- 用于在单个时钟源中需要将 BUFR 或 BUFIO 用于一个以上的垂直相邻时钟域，且该时钟需要定期停止的情况。
- 用于使用一个以上的 BUFR 且需要时钟分频的情况。 BUFMRCE 可用于确保所有连接的 BUFR 的正确相位启动

PLL 和 MMCM

- 用于为系统同步输入和输出消除时钟插入延迟 （将时钟与输入数据在相位上对齐）。
- 用于时钟相位控制，将源同步数据与时钟对齐，以便正确进行时钟采集。
- 用于修改输入时钟的时钟频率或占空比而不是简单的分频。
- 用于滤除时钟抖动。

PLL 能提供更理想的抖动控制，而 MMCM 能提供更大范围的输出频率。对时序要求更严格的情况， PLL 可能最适合，因为 PLL 能够准确提供所需的频率。

IDELAY/IODELAY

- 对 7 系列器件而言，用于在输入时钟上添加少量额外相位偏移 （延迟）。
- 用于在输入数据上添加额外延迟，以有效降低时钟相位相对于数据的偏移。

ODDR

- 用于创建来自器件的外部转发时钟。

SSI 器件的特殊时钟考虑因素

除了本节前文提及的所有考虑因素，在设计用于 SSI 的时钟结构时，还应该考虑下列因素：

- 如果时钟跨度超过一个 SLR，则把 BUFG 布置在其中一个中心 SLR 中，以最大程度降低偏差。
- 对 7 系列器件而言，一般使用靠上方或靠下方 SLR 中的 BUFH 和 BUFR，才能减少与布局在中心 SLR 中的 BUFG 发生资源争用的几率。
- 对于 7 系列器件而言，把 BUFMR 和相关引脚布置在 SLR 内部的中心时钟域内。这种布局便于在需要的时候从 BUFMR 访问全部三个时钟区域。

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 的输出相对于时钟时延 (clock-to-out) 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 或 DSP48 对称舍入和饱和。因此用户必须通过实例化来实现。

  如果用户决定实例化一条赛灵思原语，敬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。

提高可靠性的编码方式

对具体的设计情况而言，需要具体的考量来提高可靠性。

跨时钟域

每当有数据或控制信号从一个时钟域传输到另一个时钟域，必须了解跨时钟的性质。跨时钟可分类为：

- 同步跨越

  同步跨越指两个时钟域之间存在已知且可预测相位关系时的跨越。

- 异步跨越

  异步跨越指相位关系不确定时的跨越。

在异步跨越情况下，有时候时钟偏差会非常严重。超高时钟偏差的情况会更难以满足时序要求。

重要提示 ：如果偏差指标有利于满足建立要求，就会难以满足保持要求。与此相反，如果偏差指标有利于满足**保持**要求，就会难以满足**建立**要求。

同步时钟域跨越

- 由同一 MMCM、 PLL 或器件引脚从一个 BUFG 网络驱动到另一个 BUFG 网络，且两个 BUFG 均位于芯片的相同位置 （上半部或下半部）。
- 对于 7 系列器件而言，从一个 BUFH 网络驱动水平相邻的另一个 BUFH 网络 （假定这两个 BUFH 本身由同一时钟源驱动）。
- 对于 7 系列器件而言，从一个 BUFR 驱动到另一个由同一 BUFMR 驱动且配置类似的 BUFR。

  注释 ：如果 BUFR 未处于 BYPASS 模式下，必须通过同步所有相关 BUFR 上的复位来对齐相位。
- 对 7 系列器件而言，由相同时钟源驱动在位于相同时钟域的 BUFIO 和 BUFR 之间的往返。

可能存在极高偏差的同步时钟域跨越

- 使用 MMCM 或 PLL 往返某时钟网络，或是即便有相同源时钟生成的 MMCM 或 PLL，但未使用 MMCM 或 PLL 往返某时钟网络的情况。
- 在 BUFH 和任何其它网络之间往返，但该 BUFH 网络与其它网络水平相邻的情况除外。
- 前往或来自并非由专用时钟资源 （比如外部时钟引脚、 MMCM 或 PLL）直接驱动的时钟网络。
- 在位于器件上半部的 BUFG 和位于器件下半部的 BUFG 之间往返。

异步时钟域跨越

- 从一个时钟网络前往另一时钟网络，且两种间不存在相位关系。
- 在由同一 MMCM 或 PLL 生成的多个域之间往返，且彼此间的频程无规律。
- 在任何赛灵思 FPGA 器件的任何 GT TXCLKOUT 或 RXCLKOUT 输出端与另一时钟域之间往返。这还包括 TXCLKOUT 与 RXCLKOUT 之间的路径。

简言之，同步域指时钟之间存在已知且可预测相位关系的域。这一般发生在所涉时钟域符合下列情况时：（1）互为衍生物；或 （2）由同一内部源或外部源提供。在这种情况下，时序可以分析，且 （如果需要）可安全地从一个域传递到另一个域。

根据在公用节点后为到达时钟源端和时钟目标端所穿越的时钟资源的距离和性质，时钟偏差可能会非常明显，不可忽略。对寄存器到寄存器路径等小数据路径，这种时钟偏差的长度可能会超过导致保持违规的数据延迟。如果存在多个逻辑层次，这种额外的偏差会导致极难以满足时序要求。赛灵思强烈建议用户严密监测这种跨时钟域的逻辑层次，同时考虑逻辑层次过少或过多所带来的影响。

对于异步时钟域跨越，必须采取特殊措施来避免在这些路径上可能给数据完整性造成不利影响的不当总线捕获、亚稳定性及其它问题。

一般来说有两种常规方法可以让数据安全地跨越异步时钟域。如果只需要传输单比特数据，或是使用格雷码等方法来传输超过 1 比特的相关数据，可插入寄存器同步器来降低电路的平均无故障时间 (MTBF)。对多比特数据 （即总线），一般建议的方法是使用独立时钟 （异步） FIFO 把数据安全地从一个域传输到另一个域。如果用软逻辑构建，这个 FIFO 可通过调用实现。但是如果要使用专用的硬 FIFO （或是预先特性化和预先定义的 FIFO 逻辑能够简化工作），也可以直接实例化这个 FIFO。还可以使用 FIFO 原语或 FIFO 生成器来构建这个 FIFO。

在 HDL 代码中使用 ASYNC_REG 属性可以识别所有的同步寄存器。通过采用这种方法， Vivado Design Suite 设计工具能够更好地了解并使用专用算法来改进综合、仿真、布局布线，减少亚稳态发生，达到改善 MTBF 的目的。

ASYNC_REG 实例

```
module synchronizer #(
 parameter SYNC_STAGES = 2
) (
    input ASYNC_IN,
    input CLK,
    output SYNC_OUT
);
 (* ASYNC_REG = "TRUE" *) reg [SYNC_STAGES-1:0] sync_regs = {SYNC_STAGES{1'b1}};




  always @(posedge CLK)
    sync_regs <= {sync_regs[SYNC_STAGES-2:0], ASYNC_IN};

  assign SYNC_OUT = sync_regs[SYNC_STAGES-1];

endmodule
```

提示： 可以考虑运行静态检查器，以识别跨时钟域并确认适当的同步。

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定时钟域，或如果时钟的运行速度快于 GWE 安全释放的步伐，部分设计就会进入未知状态。对某些设计而言，这无关紧要。但对另一些设计而言，这种情况会导致设计失去稳定性或不能正确处理初始数据集。如果设计必须进入已知状态，赛灵思建议采取措施控制启动同步过程。实现方法有几种。

一种方法是延迟所有设计时钟，直到 GWE 有效后一段时间。为此赛灵思建议：

- 使用实例化的 BUFGCE、 BUFHCE 或 BUFR 组件。
- 使用这些组件的使能信号在配置后延迟时钟一定数量时钟周期。
- 在使用 MMCM 时，延迟输出时钟，但勿延迟反馈时钟。从 Clocking Wizard 中选择**"Safe Clock Startup"**选项即可。

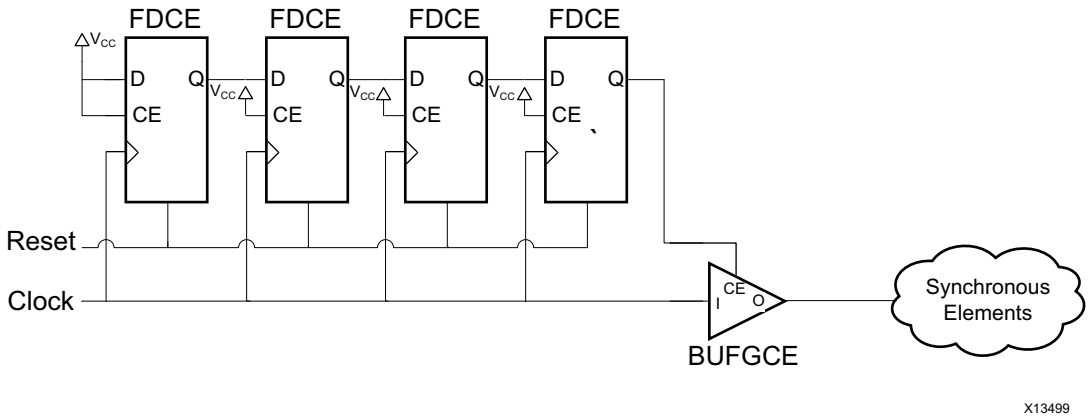


图 4-8 : 时钟启动

此外，您还可以使用时钟使能、局部复位 （同步的）或设计关键部分 （如状态机）上的两者来确保设计这些部分的启动是受控、已知的。

使用无时序约束复位

复位 （尤其是全局复位）可能会在大部分 FPGA 阵列上产生较高扇出。在这些情况下，不论时钟频率或时序要求如何，复位时序都难以满足。这可能会给高速设计带来极大难度。

在复位路径上缺失时序可能导致无法确定的行为，例如初始或间歇性数据损坏、整个设计锁死或在极端情况下灾难性功能失常。

对大多数设计而言，问题发生在解除复位而非复位的过程中 （虽然在某些情况下使能也可能造成问题）。如果在时钟在某个特定的寄存器上激活的同时解除复位，该寄存器的输出可能无法确定 （恢复/解除违规）。如果复位信号因为偏差，造成设计的一些部分从一个时钟周期复位中被释放，而其它部分在另一个时钟周期复位中被释放，会导致无法知晓的电路行为。在控制和同步器件启动中介绍了一种同步复位解除的实例方法。

赛灵思建议除添加同步电路之外，您还应在复位路径上布置适当的时序约束。同时在设计输入阶段，应尽量缩小时序收敛对复位的影响。方法之一是移除复位或是复制驱动器，以限制复位信号驱动的负载的数量。在寄存器上使用 ASYNC_REG 属性以同步复位，可以防止复制驱动器。因此，如欲进行复制 （比如复位正在驱动高扇出），可在同步装置末端添置一个额外的寄存器，不过最后一个寄存器不包括 ASYNC_REG 属性。最后的寄存器可用于复制，并且它不属于同步链的组成部分。

如果还不足以满足时序要求，您可在复位过程中使用 BUFGCE 功能在内部或从外部停止时钟，从而为复位信号提供多周期解除时间。

重要提示 ： 复位解除必须进行时序约束，这样整个设计才能在相同的周期中同时复位。

UltraScale 器件 IDDRE1 复位考虑事项

UltraScale 器件中 IDDRE1 的 RST 引脚恢复时间与 CLK 引脚的下降沿有着严格的时序关系。严格的时序要求可能导致时序违规。如果无法满足恢复时序要求，赛灵思有以下设计建议：

在 IDDRE1 RST 解除后等待几个周期，并对目标 IDDRE1 RST 引脚应用 set_multicycle_path 约束。例如：

- 保持 D 引脚上输入数据的稳定，在 RST 解除后等待 3 个时钟周期，然后再使用 IDDRE1 的 Q 引脚上的数据
- 忽略 RST 解除后的 3 个时钟周期中来自 IDDRE1 的 Q 引脚上的数据 （例如可禁用下游寄存器进行数据采样）
- 由于 IDDRE1 处于复位状态，因此其输出为“0”。保持 IDDRE1 的 D 输入也为“0”持续 3 个周期。

如果设计要求 IDDRE1 的 Q 引脚的数据必须在 RST 解除后立即采样，那么赛灵思建议复制寄存器驱动 IDDRE1 的 RST 引脚。在某些情况下，每个 IDDRE1 都可能需要一个寄存器。让驱动 IDDRE1 的 RST 引脚的寄存器位置尽可能靠近 IDDRE1，方法是采用 LOC 约束或 PBlocks，从而在最短的时间内获得 RST 信号。

避免使用组合环路

在 FPGA 设计中应避免使用组合反馈路径。在时序上造成的后果难以仿真、分析和根据各种工作条件全面考虑。结果会难以预测。

如需了解关于如何使用 `check_timing` 命令检查无意间造成的组合环路，敬请参阅：第 5 章：实现。

改善性能的编码方式

违背上一章节中介绍的编码方法 （提高可靠性的编码方式) 一般会给性能造成不利影响。对高性能设计而言，本章节中讨论的编码方法 （改善性能的编码方法）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。

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。

如果您发现高扇出网络，可采用下列方法缓解这个问题：

- 精简负载到不需要高扇出网络的设计部分
- 使用寄存器复制

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。对数据路径，判断是否可以通过限制逻辑来减少扇出。

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要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。不过这个寄存器也将不再构成同步链的组成部分。

表 4-3 是您设计中可接受的扇出数量提示性指南。

表 4-3 : 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		
中频 125 到 250 MHz	取决于结果。可能需要减少扇出和/或减少逻辑级数来实现。	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。）	
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，敬请参阅：第 5 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。

流水线考虑事项

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。采用这种方法可以让数据路径跨越多个时钟周期。相应地，必须对设计的其余部分进行特殊考虑，容纳这种增加的路径延迟。

SSI 器件的流水线考虑事项

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播延迟差异。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改善时序收敛、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）。

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，应注意下列三个条件：

- 带有大扇入的逻辑锥

  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模块

  例如，没有输出寄存器的块状 RAM 或没有正确流水线化的算术代码。

- 当强制布局导致长布线时

  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。

图4-9 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。

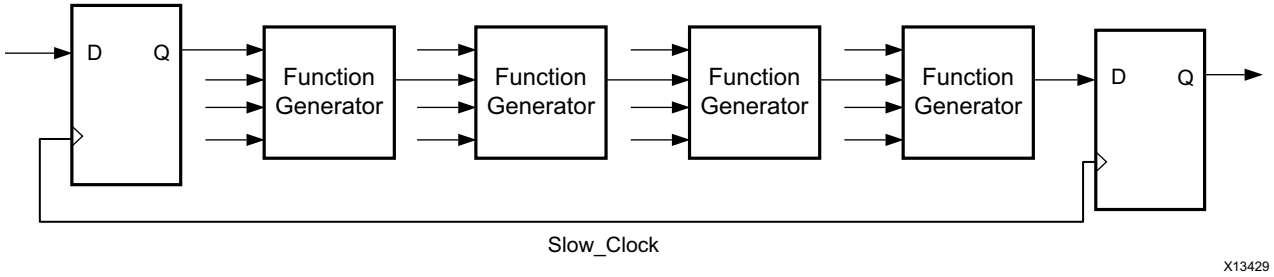


图 4-9 : 使用流水线前的原理图

图4-10 中的数据路径与图4-9 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时钟时延 (clock-to-out)、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。

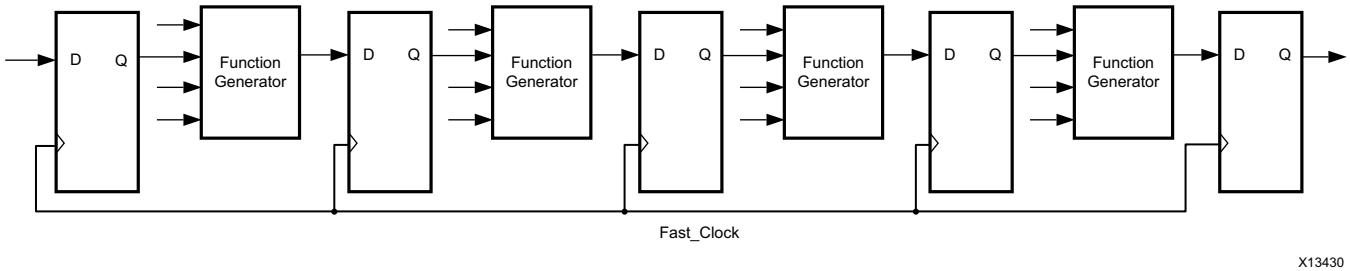


图 4-10 : 流水线化后的原理图

提示： 通过均衡寄存器之间的逻辑级数数量来提升设计性能。

管理宽总线

对高吞吐量的需求催生对高频率下更大宽度总线功能的需求。例如，要实现每秒 200GB 的吞吐量数据传输，需 1024 位宽总线以 200 MHz 频率传输数据。

更大宽度的功能和更大宽度的存储器已经成为新一代 FPGA 设计的主流。赛灵思在芯片上提供各种布线资源以及先进的布局布线算法来满足这些需求。

您应充分考虑可用资源，用它们来实现更加理想的性能。可以从下列方面入手，使用合适的设计方法来改善设计：

- 存储器组织结构

  - 寄存器地址和存储器数据输出。
  - 在RTL编码时应按宽度区分存储器，以实现高性能。
- 大宽度功能
  - 在实现大宽度算术功能和缩减运算符时，使用充足的流水线级数。
  - 在使用 SSI 目标和进行手动设计分区时，确保最小程度的 SLR 交错。
  - 使用赛灵思 IP 来实现大宽度总线复杂算术运算。用 IP 核处理高性能所需的时钟再生和流水线要求。
- I/O
  - 使用赛灵思串行 IP 核实现大宽度总线、高吞吐量、高可靠芯片间数据传输。
  - 适当分配同一组或相邻组的主 I/O，最大限度降低传入或输出总线接口各数位之间的偏差效应。

改善功耗的编码方式

改善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播。

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端口

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，应采用专用时钟缓冲器的时钟使能端口。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 模块

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

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
```

设计中块状 RAM 的最佳实践

块状 RAM 的总功耗与其使能时间成正比。为了节省功耗，应在不使用块状 RAM 的设计中将块状 RAM 使能端驱动至低电平。块状 RAM 使能速率和时钟速率都是功耗优化时必须考虑的重要参数。

块状 RAM （例如 NO_CHANGE、 READ_FIRST 和 WRITE_FIRST）的模式设置在之前的“选择正确的 BRAM 写入模式”一节中已经介绍过。

实现更深和更宽的存储器设计必须遵循深度划分机制，以节省动态功耗。在 IP 定制过程中，如果您选择节能方案，Vivado IDE 会创建深度划分。

下列图表给出了针对以上内容的实例，采用的存储器配置为 4Kx1024位，按宽度划分。

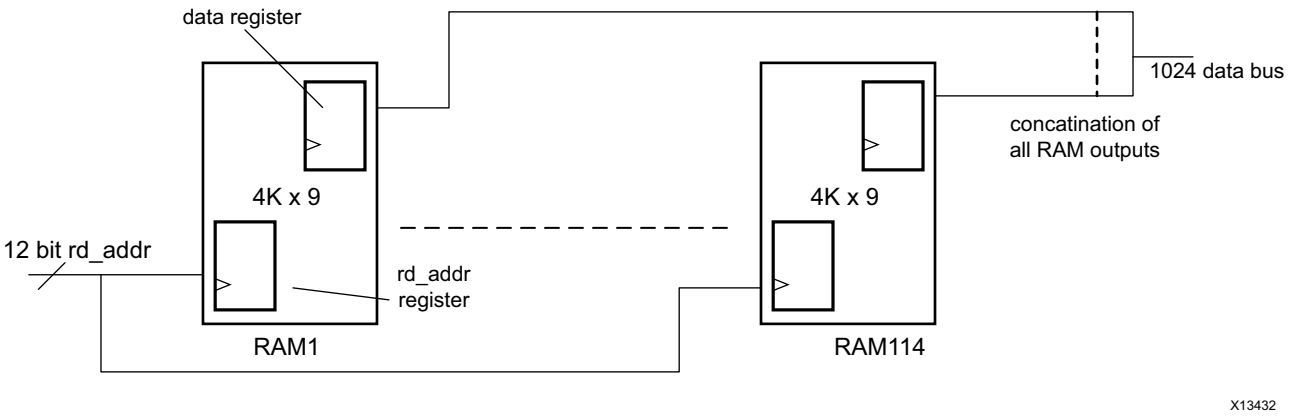


图 4-11 : 使用 4K x 9 实现的 4K x 1024 的 RTL 表达形式

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

下列图表为按深度划分的实例。

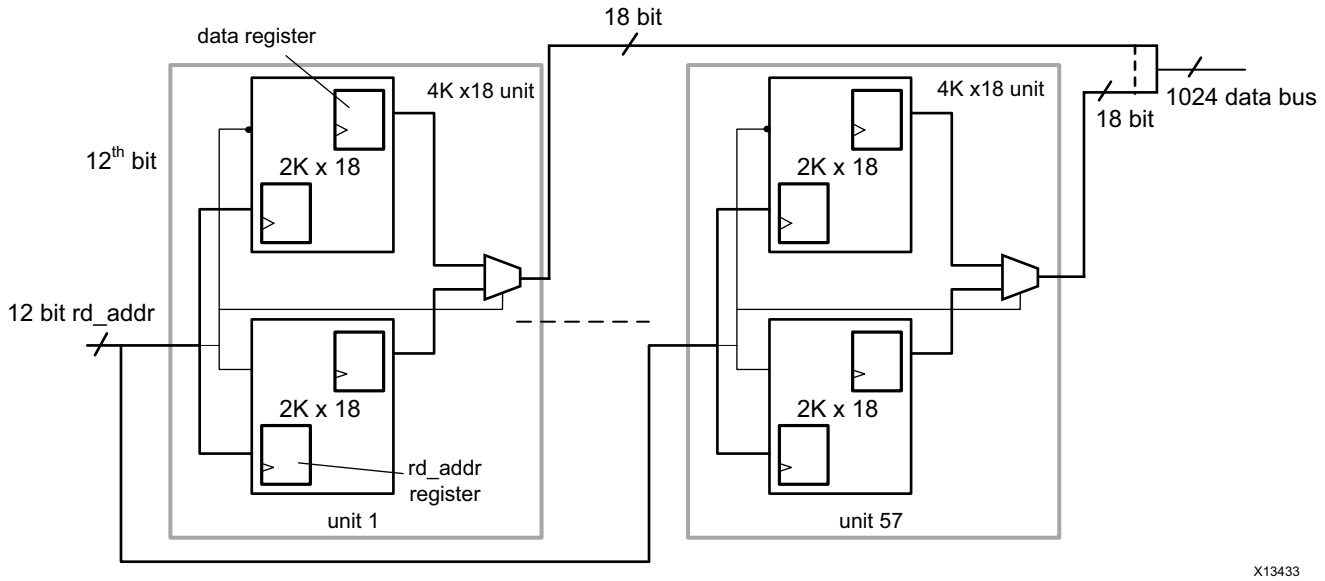


图 4-12 : 使用 2K x 18 实现的 4K x 1024 的 RTL 表达形式

该实现方案中，由于每次（从每个单元中）只选择一个块状 RAM，因此动态功耗几乎减半。 UltraScale 器件块状 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，需要一个以上的块状 RAM 原语，采用极高能效的配置，同时对性能影响很有限。综合调用和 Block Memory Generator 都能自动根据指定的 RAM 配置利用电路系统。

下图给出了四个级联块状 RAM 的高级概念视图。

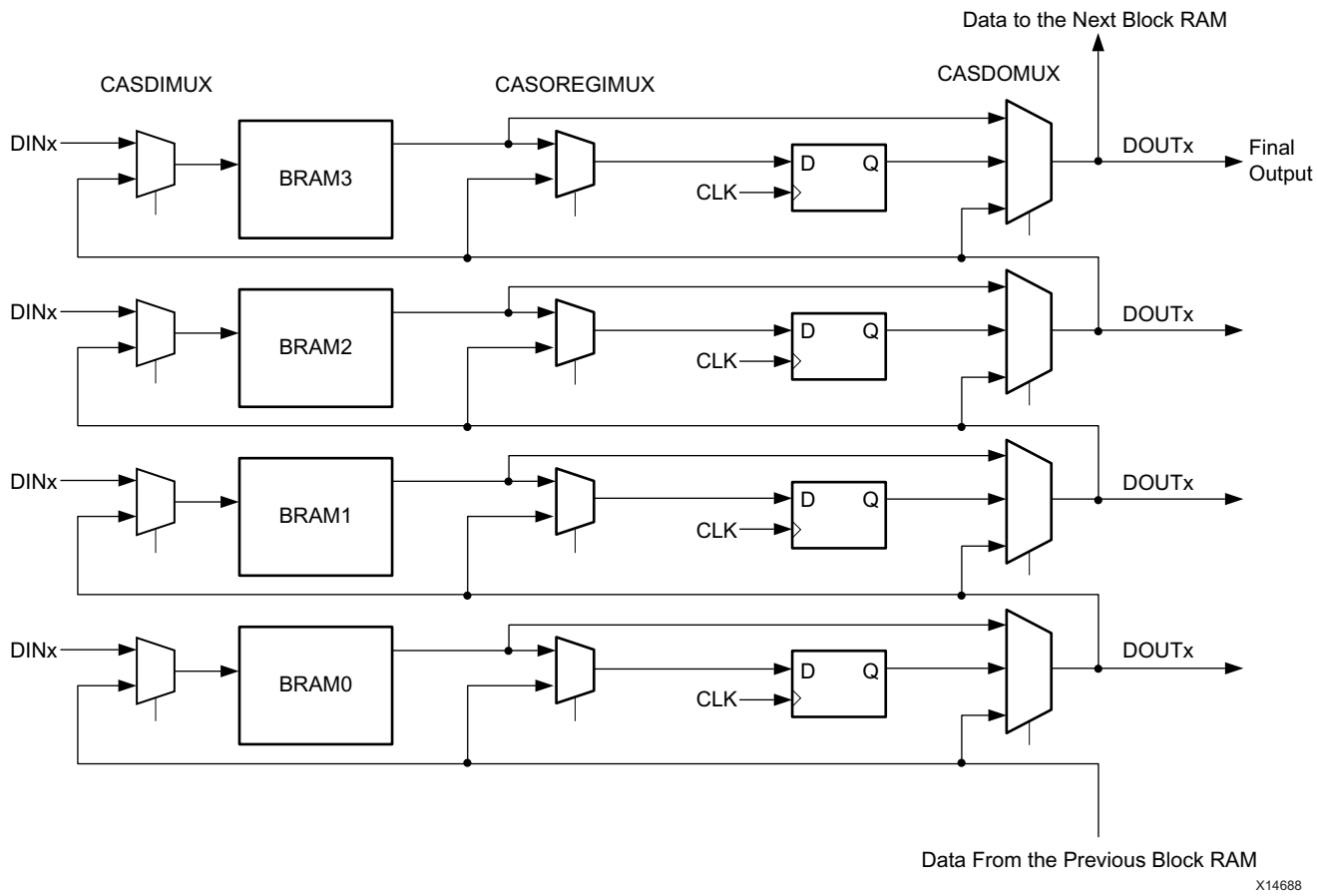


图 4-13 : 块状 RAM 级联架构的高级视图

参数、属性和约束

根据上下文，应在参数、属性或约束之间使用可互换的命名。本部分内容将介绍这些概念。用户应了解这些概念，这样即便是另有文献资料使用某种其它命名，用户仍能解读文献深层次表达的含义。

- 参数
- 约束和属性

参数

参数 （在 VHDL 中为“类属 (Generic)”）指与器件架构原语组件有关，会对实例化后的组件的功能或设计造成影响的特性。这些特性的传递方式对 VHDL 而言是“类属”映射，或者说对 Verlilog 而言是内联参数传递。这些特性在 VHDL 和 Verilog 中都称之为“类属”或参数。

注释 ：虽然可以使用定义参数声明语句来修改参数，赛灵思不建议采用这种操作方式。

参数实例包括：

- LUT6 组件上的 INIT 特性
- MMCM 上的 DIVCLK_DIVIDE 特性

所有参数在 《赛灵思库指南》中的原语组件介绍部分均有介绍。用户可使用参数定制赛灵思原语的具体行为。

就*原语*而言，在说明实例化过程中可修改的特性时，部分文献使用"*属性*"一词来代替本文中定义的"*参数*"或"*类属*"。

VHDL 原语参数 （类属）编码实例：

在下面的 VHDL 编码实例中，通过设置一个实例化的 RAM32x1S 原语的 INIT 类属，将该 RAM 符号的初始内容确定为十六进制值 A1B2C3D4。

```
small_ram_inst :RAM32X1S
generic map (
INIT => X"A1B2C3D4")
port map (
O => ram_out, -- RAM output
A0 => addr(0), -- RAM address[0] input
A1 => addr(1), -- RAM address[1] input
A2 => addr(2), -- RAM address[2] input
A3 => addr(3), -- RAM address[3] input
A4 => addr(4), -- RAM address[4] input
D => data_in, -- RAM data input
WCLK => clock, -- Write clock input
WE => we -- Write enable input
);
```

Verilog 原语参数编码实例：

在下面的 Verilog 编码实例中，可将一个实例化 IBUFDS 符号中的 DIFF_TERM 和 IOSTANDARD 分别设定为 FALSE 和 LVDS_25。

```
IBUFDS #(
.DIFF_TERM("FALSE"), // Differential Termination
.IOSTANDARD("DEFAULT") // Specify the input I/O standard
) IBUFDS_inst (
.O(O), // Buffer output
.I(I), // Diff_p buffer input (connect directly to top-level port)
.IB(IB) // Diff_n buffer input (connect directly to top-level port)
);
```

约束和属性

约束和属性往往在使用中可以互换。不过严格来说，*属性*是指在 HDL 代码本身中提供的指令，而*约束*则是以约束文件 (XDC) 的方式提供。属性和约束都针对特定工具给出了如何解读和实现特定信号或实例的方法指南。

有几项特性既可作为 HDL 中的"*属性*"，也可作为 XDC 中的"*约束*"提供。因此这种特定的特性既可视为属性，也可视为约束。相应地就这几项特性而言，属性和约束在使用中可以互换。

约束可分为三类：

- 综合约束
- 时序约束
- 物理约束

综合约束和属性

综合约束的作用是针对特定设计或 HDL 代码段引导综合工具的优化技巧。综合约束或是嵌入在 VHDL 或 Verilog 代码中（也称为属性），或是存在于单独的综合约束文件中。综合属性的例子有 USE_DSP48 和 RAM_STYLE。

赛灵思建议：

- 将影响功能的指令以属性的形式嵌入在 HDL 代码中。 HDL 代码应始终随同相关属性。

- 将临时约束 （比如调试所需的属性）布置在单独的约束文件中。这样无需修改实际的 HDL 文件即可以方便地删除或添加约束。

综合属性、约束和指令常嵌入在早期实现方案或架构的代码或综合约束文件中。赛灵思建议将这些内容注释掉或删除这些元素。它们可能会导致结果质量下降，同时也可能并非是将来实现方案的最佳选择。

提示： 在将现有设计重新用于新设计或器件时，删除嵌入在现有设计代码或网表中的所有 LOC、 RLOC 或 BEL 约束，或其它物理约束。

对旧架构而言，理想的布局未必对新设计或架构也是理想的布局。在某些情况下，一定的约束 （例如与位置相关的约束）对新架构甚至可能无效。

下面的实例说明如何只使用 HDL 代码传递属性。

属性声明实例

```
attribute attribute_name : attribute_type;
```

在端口或信号上使用属性的示例

```
attribute attribute_name of object_name : signal is attribute_value
```

参见下面的实例：

```
library IEEE;
use IEEE.std_logic_1164.all;
entity d_reg is
port (
CLK, DATA: in STD_LOGIC;
Q: out STD_LOGIC
);
attribute KEEP_HIERARCHY : string;
attribute KEEP_HIERARCHY of d_reg : entity is “true”;
end d_reg;
```

在实例上使用属性的示例：

```
attribute attribute_name of object_name : label is attribute_value
```

参见下面的实例：

```
architecture struct of spblkrams is
attribute LOC: string;
attribute LOC of SDRAM_CLK_IBUFG: label is “AA27”;
begin
-- IBUFG:Single-ended global clock input buffer
-- All FPGA
-- Xilinx HDL Language Template
SDRAM_CLK_IBUFG :IBUFG
generic map (
IOSTANDARD => “DEFAULT”)
port map (
O => SDRAM_CLK_o, -- Clock buffer output
I => SDRAM_CLK_i -- Clock buffer input
);
-- End of IBUFG_inst instantiation
```

在组件上使用属性的示例

```
attribute attribute_name of object_name : component
is attribute_value
```

参见下面的实例：

```
architecture xilinx of tenths_ex is
attribute black_box : boolean;
component tenths
port (
CLOCK : in STD_LOGIC;
CLK_EN : in STD_LOGIC;
Q_OUT : out STD_LOGIC_VECTOR(9 downto 0)
);
end component;
attribute black_box of tenths : component is true;
begin
```

以往 Verilog 并没有与 VHDL"*属性*"相似的概念。因此大多数工具针对 Verilog 都有自己的*编译提示*。对传递与 VHDL 相似的属性， Verilog 2001 提供统一的语法。由于属性声明紧跟对象声明，在属性声明中未提及对象名称。

```
(* (attribute_name = "attribute_value" *)
Verilog_object;
```

参见下面的实例：

```
(* (RLOC = "R1C0.S0" *) FDCE #(
.INIT(1'b0) // Initial value of register (1'b0 or 1'b1)
) U2 (
.Q(q1), // Data output
.C(clk), // Clock input
.CE(ce), // Clock enable input
.CLR(rst), // Asynchronous clear input
.D(q0) // Data input
);
```

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"Elaborated Design > Report DRC"执行，也可根据 Tcl 命令提示符通过执行 `report_drc -ruledeck methodology_checks` 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。

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。查看如下资源，了解有关利用 IP 的更多信息：

- 《Vivado Design Suite 用户指南：采用 IP 进行设计》(UG896) [参照 9]
- Vivado Design Suite QuickTake 视频：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，回答下列问题：

- 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。

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 访问和定制 IP 目录中的 IP 核。 Tcl 脚本能够自动完成 IP 核的定制工作。

定制 IP

赛灵思使用业界标准的 IP-XACT 格式交付 IP，并提供 IP 打包器，用于封装定制 IP。相应地，您也可以把自己定制的 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。第三方提供商提供的 IP 也可以添加到本 IP 目录中。

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，查看自己感兴趣领域的 IP 核。

视频： 有关如何使用 IP 目录定制、添加和实例化 IP 到工程中，敬请观看：Vivado Design Suite QuickTake 视频：定制和实例化 IP。

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

IP 和 I/O

与外界互动的 IP 必须与 I/O 引脚建立关联。为此，赛灵思建议用户在选择 IP 时应考虑 I/O 分配问题。具体包括：

- 并行接口
- 串行接口
- I/O 电压和 I/O 标准

并行接口

根据 I/O 单元 (IOB) 中可用 I/O 的数量决定选择哪个 I/O 单元。

串行接口

- 低速串行接口：可以使用属于通用 IOB 组成部分的 ISERDES/OSERDES；
- 高速串行接口：可以使用低功耗千兆位收发器 (GT)。

I/O 电压和 I/O 标准

- 如果 I/O 电压为 1.8V，应选择支持 1.8V Vccio 的I/O 单元；
- 在低数据速率下，应使用单端 I/O 标准，如 LVCMOS；
- 在高数据速率下，应使用差分 I/O 标准，如：
  - LVDS
  - DIFF-SSTL
  - DIFF-HSTL

IP 选择和定制的决策流程实例

通信和网络系统设计时应考虑以下要求：

- 10 端口 10G 以太网 MAC 聚合
- 用于系统配置的 PCIe 接口
- 外部存储器存储

根据要求和可用的 IP，您现在必须检查每个 IP 的主要功能特性，确定其是否适用于自己的设计，是否需要定制。经过这个过程，您就可以根据自己的设计目标选择合适的 IP。

10 端口 10G 以太网 MAC 聚合

IP 支持可选的 XGMII 接口选项。如果系统需要使用 XAUI 或 10G PCS/PMA 作为其外部接口，必须选择 XGMII 选项。赛灵思提供的 XAUI 或 10G PCS/PMA IP 支持 XGMII 接口。

由于 MAC 的数据是通过 AXI4-Stream 接口来传输的，因此系统必须能够接收来自 MAC 的数据。随后必须与 AXI 互联相连接，实现与其它 IP 核通信，或通过专有协议采用封装程序端接。

可通过可选的 AXI-Ltie 接口或简单的读/写接口来配置该内核。如果选择 AXI-Lite 接口，系统内部应能够支持 AXI-Lite。

用于系统配置的 PCIe 接口

除了上面提及的考虑事项，用户还必须清楚接口的数据速率要求。为便于用户做出选择，可参阅：表 4-4。

表 4-4 : 器件的数据速率要求

	Artix®-7	Kintex®-7	Virtex®-7T	Virtex-7 XT	Virtex-7
GEN （集成模块）	Gen2	Gen2	Gen2	Gen3	Gen3
宽度	X4	X8	X8	X8	X8
模块数量	1	1	3-4	2-4	1-3
串行数据速率 (Gb/s)	5	5	8	8	8

外部存储器存储

您需要清楚系统中需要支持的 DDR3 存储器的数量。在该设计中考虑存储器控制器 (MC) 效率时，要求总的存储数据速率约为 80Gb/s 有效带宽，或 100Gb/s 原始带宽。具体实现方法有如下数种：

- 使用单个 1600Mbps 速率的 64 位 DDR3 控制器
- 使用四个 1600Mbps 速率的 16 位 DDR3 控制器

由于存储器控制器的数据是通过 AXI4 接口传输的，那么该系统应能够接收来自 MC 的数据。用户可以使用 ½ 或者 ¼ 速率的接口。在使用 ¼ 速率的接口时，存储器控制器的应用数据宽度为 8xDDR3。例如， 16 位 DDR3，那么 AXI-Stream 的数据宽度应为 128 位宽。可以使用 AXI Interconnect IP 将从存储器控制器 IP 和负责访问存储器的主外设相连。在这种情况下，来自 IP 核的数据会通过 AXI4-Stream 接口，此时需要添加一个 DMA，供控制器把 AXI4-Stream 转换为 AXI4。

赛灵思建议使用存储器接口生成器 (MIG) 来生成存储器控制器，其还可指导如何选择 I/O Bank。

定制IP

可通过 GUI 或通过 Tcl 脚本定制 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、搜索和定制 IP 的最简单的途径。每个 IP 都有为其定制的一套标签或页面。同时提供相关的配置选项。下面就是这种定制窗口的实例。首先为 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。

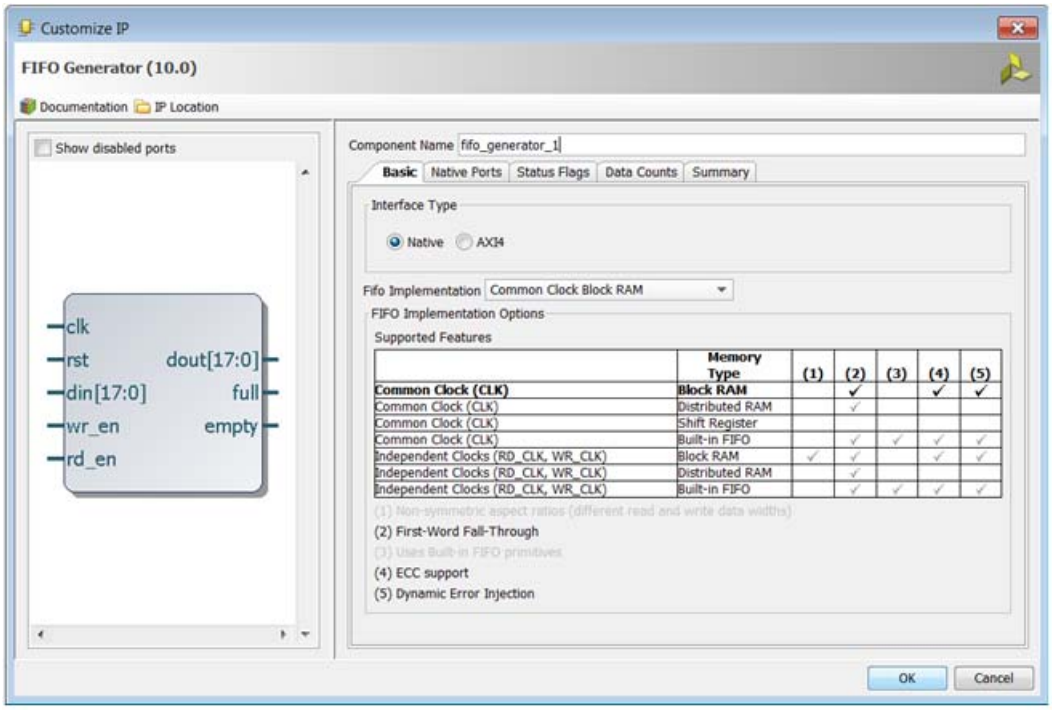


图 4-14 : IP 定制窗口

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。参见第 2 章中的源文件管理及版本控制建议。

IP 输出结果

IP 定制完成后，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。如需了解更多信息，敬请参阅：第 2 章中的管理 IP。

提示： 为 MIG 创建的是 `.prj` 文件而非 XCI 文件。在涉及 IP 的情况下，后续所有的 XCI 对 MIG 所指的都是 `.prj` 格式。

生成输出结果

根据 Vivado 工具的设置，默认情况下， IP 定制过程中会自动创建部分输出结果。用户可以在“Project Setting”中加以修改。使用定制过程中生成的 XCI 文件，用户可以创建 IP 提供的任何相关输出结果，包括：

- 实例化模板

  用于说明如何实例化 IP

- 综合

为在特定的 IP 定制视图上运行综合生成脚本。

- DCP （无关联 (OOC) 综合）

  生成综合列表，这可直接用于较高一级的设计中，无需重新综合设计中的 IP 部分。

- 测试平台

  生成用作仿真环境的测试平台，供验证 IP 功能使用。

- 仿真

  为使用上述测试平台在特定的 IP 定制视图上运行仿真生成脚本。

XCI 文件和这些输出结果包含着在设计中正确仿真和综合 IP 所需的全部要素 （例如 HDL 文件和约束文件）。

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。如需了解有关约束的更多信息，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 或主模块

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：

```
## 时序使用阶段
# 主时钟
# 虚拟时钟
# 衍生时钟
# 外部MMCM/PLL反馈回路延迟
# 时钟不确定性和抖动
# 输入和输出延迟约束
# 时钟组与时钟伪路径

## 时序例外阶段
# 伪路径
# 最大延迟/最小延迟
# 多周期路径
# Case 分析
# 不分析时序
```

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，并利用基于时序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` 和 `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_false_path -hold`

  - `set_min_delay`
  - `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)

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 模块。如需了解设计对象细化的名称，请参见第 5 章中的使用细化设计。

有些寄存器被吸收到 RAM 模块中，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。

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。

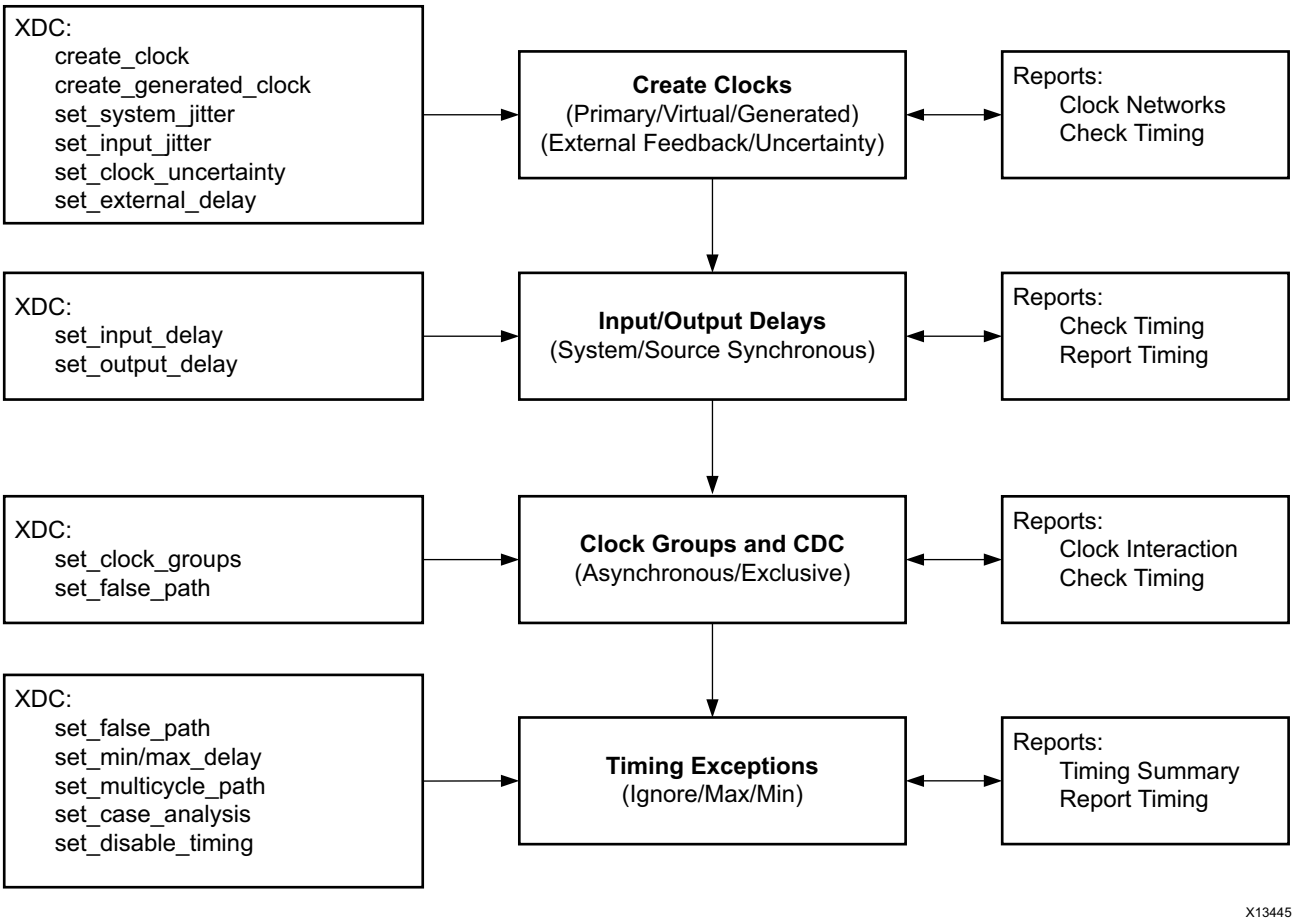


图 4-15 : 时序约束制定步骤

- 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4-15 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 的信息请见：Vivado Design Suite User Guide: Using Constraints (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根：

- 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，敬请参阅：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

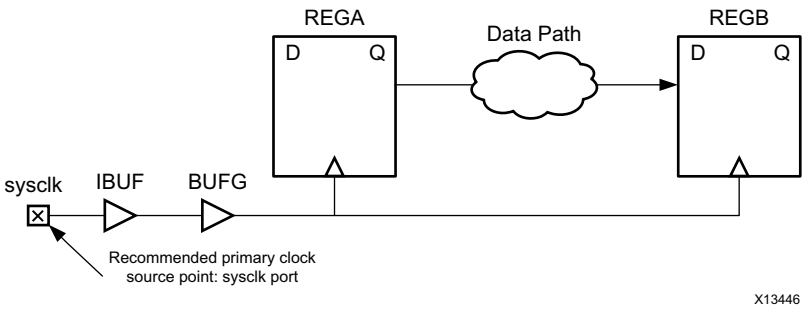


图 4-16 : 输入端口上的 create_clock

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冲器，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：

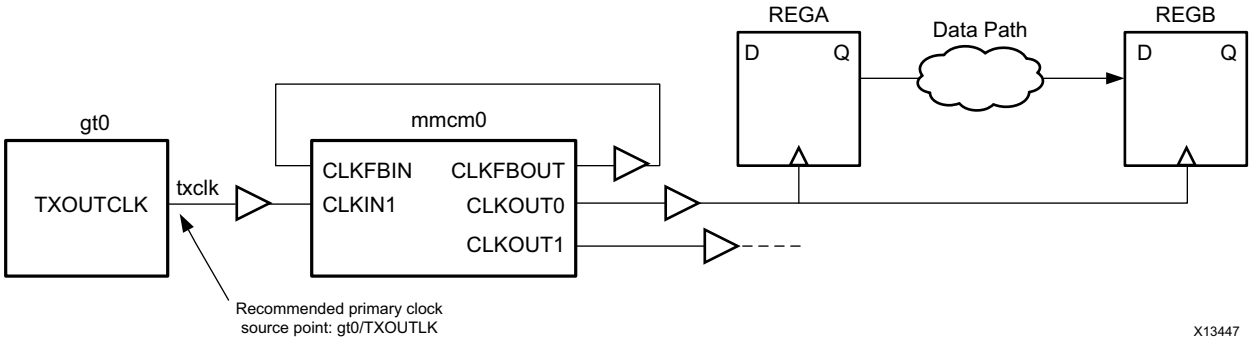


图 4-17 : 原语引脚上的 create_clock

约束实例：

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

注释 ：在 UltraScale 器件中，只要有关单板输入时钟被定义，就会自动衍生出千兆位收发器时钟。为此，对于 UltraScale 器件，在设计中赛灵思不建议在千兆位收发器的输出端定义主时钟。

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。

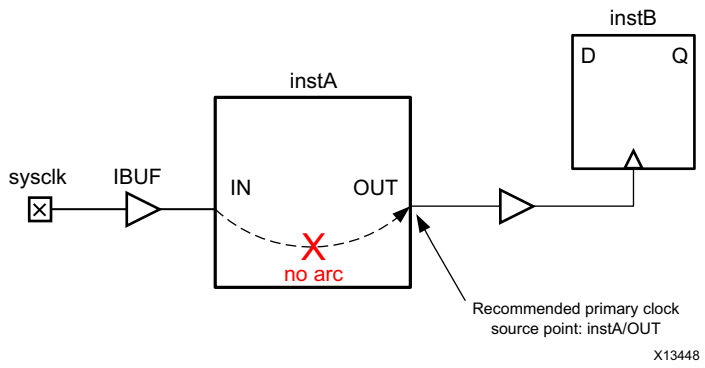


图 4-18 : 时钟路径因缺失时序 arc 而断开

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。

图4-19 给出了一个实例，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。

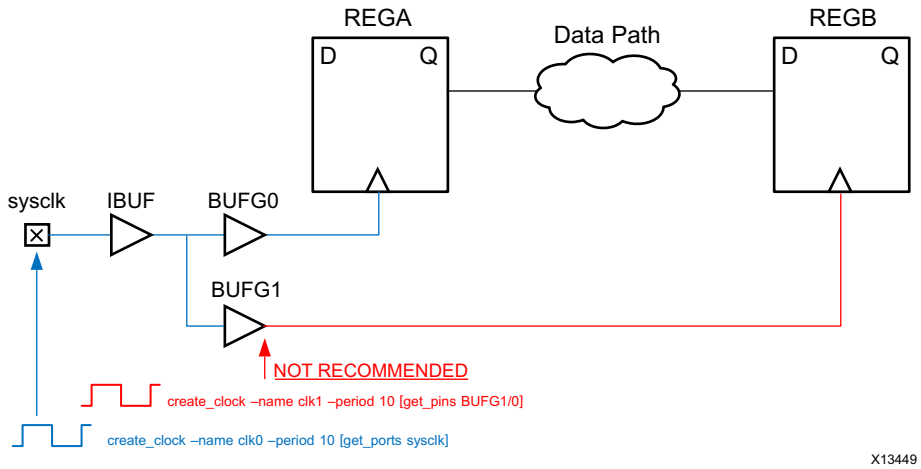


图 4-19 : 不建议在另一个时钟的扇出中使用 create_clock

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播，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，敬请参阅 Vivado Design Suite User Guide: Using Constraints (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，在图4-20 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`。

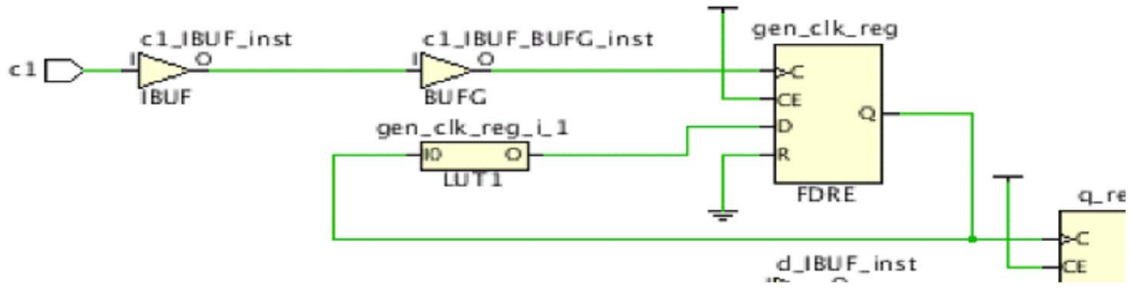


图 4-20 : 主时钟扇出中的生成时钟

```
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}
…
=====================================================
生成时钟
=====================================================

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。

您也能运行与时钟约束相关的 Methodology XDC 和 Timing DRC 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。要运行所有 Methodology DRC 或只运行 XDC 和 Timing DRC，请使用以下命令：

```
report_drc  -ruledecks methodology_checks
```

或

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

如需了解更多信息，敬请参阅：第 5 章中的运行设计方法中的 DRC 功能。

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模块的特性。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 500ps，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250ps：

```
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播。
- 对模块在无关联 (OOC) 编译过程中使用的时钟的内部传播延迟进行建模。在这样的编译流程中，未对完整时钟树进行描述，因此无法自动计算模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接收数据。

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。

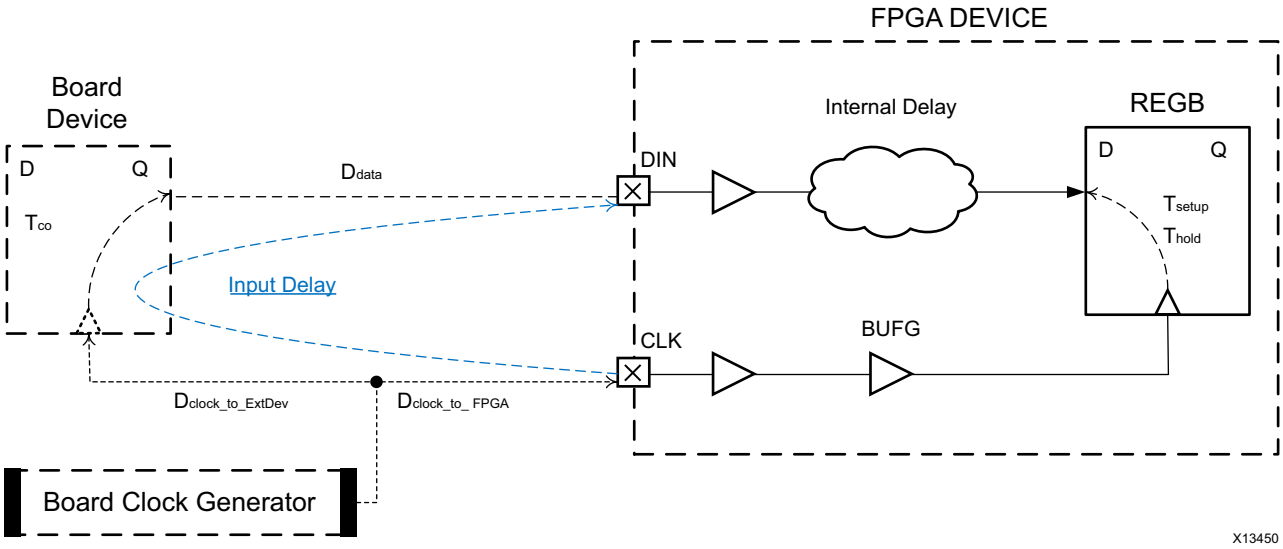


图 4-21 : 输入延迟计算

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)
```

图4-22 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实例，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]
```

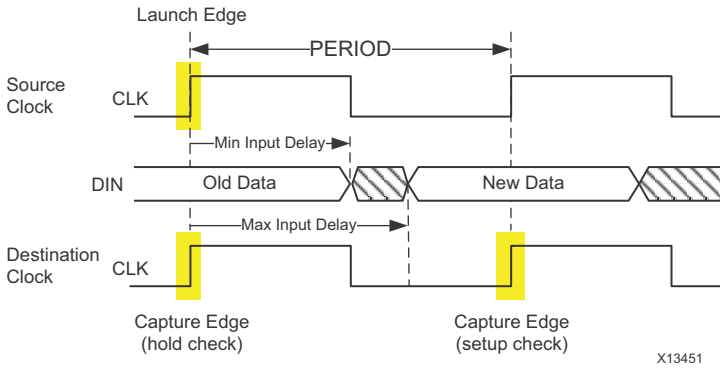


图 4-22 : 解读最小和最大输入延迟

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。

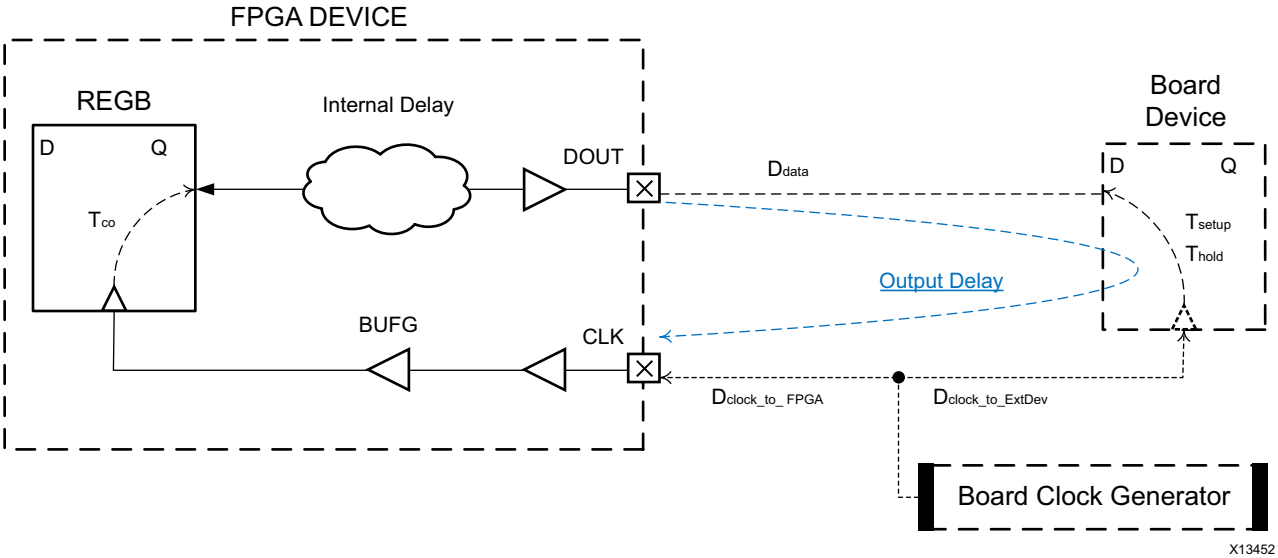


图 4-23 : 输出延迟计算

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

图4-24 解读最小和最大输出延迟给出了建立（最大）和保持（最小）分析中输出延迟约束的简单实例，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]
```

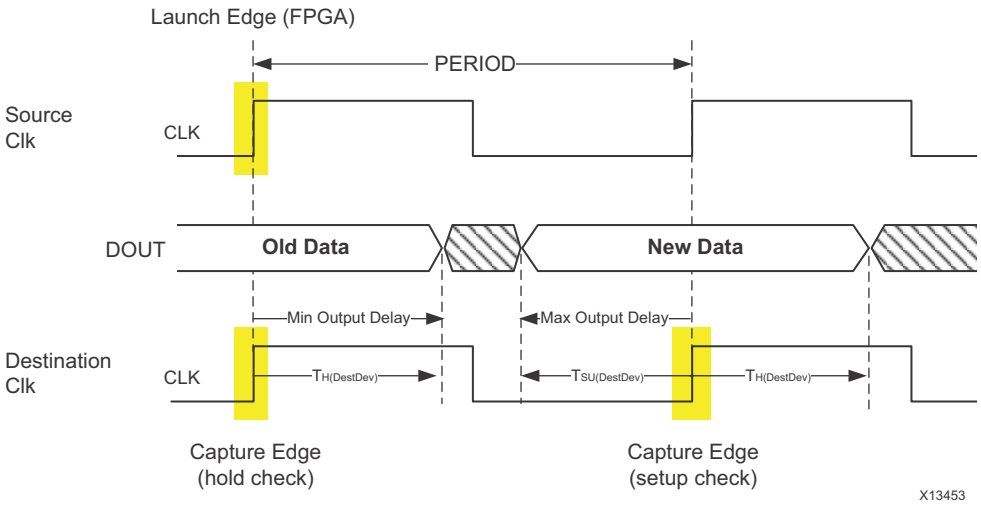


图 4-24 : 解读最小和最大输出延迟

输出延迟相当于捕获边缘之前单板上的延迟。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，模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 clk0 和 clk1 时钟：

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

报告显示 din 端口与 clk0 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 clk0 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 clk1，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。

通过查看未约束路径段，可利用 Timing Summary 报告完成相同的分析。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)           clk0
(none)                            clk0
(none)                            clk1
```

没有时钟名 （或 Vivado IDE 中的<NONE>）的字段是指起点 (From Clock) 或终点 (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 clk0 到 NONE 类别的 Setup 路径，就可以在“To”列中看到由 clk0 驱动的端口：

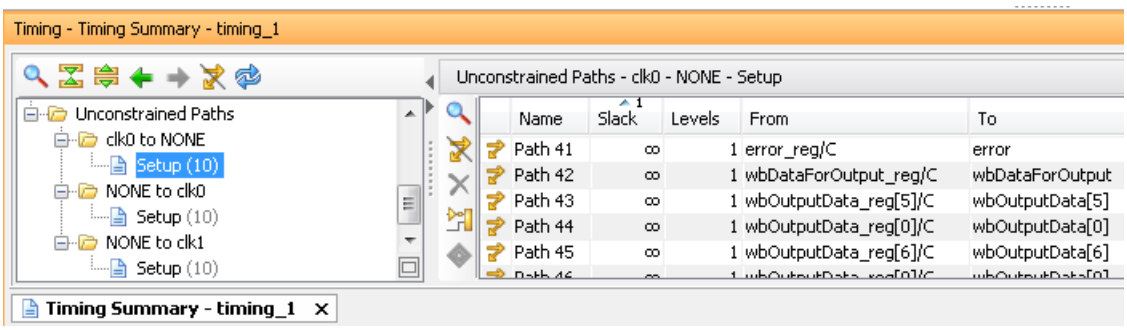


图 4-25 : 获得未约束的输出端口列表

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模块时，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模块，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两部分中提供的两个实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。

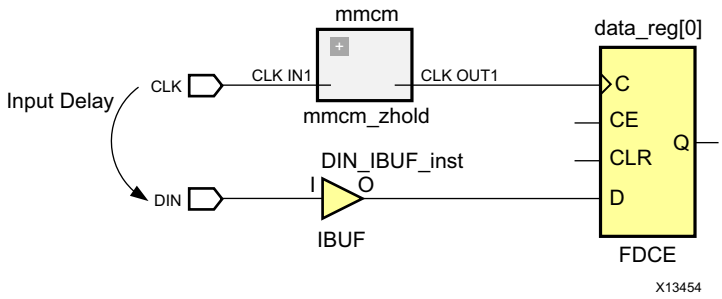


图 4-26 : 时钟路径中 ZHOLD MMCM 点的输入延迟

这些约束与定义输入延迟中提供的实例相同，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模块，而该模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。

例如，假设 sysClk 单板时钟频率为 100MHz，与 MMCM 相乘后生成频率为 266MHz 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 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接收 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发送和接收沿 （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 “Window > 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。`set_clock_groups` 命令关闭的是您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。与 `set_clock_groups` 不同， `set_false_path` 约束只忽略由 `from` 和 `to` 选项设定了方向的时钟之间的时序。在某些特定情况下，可在时钟域交叉 (CDC) 路径上设定最大延迟约束，以限制这些路径上一个或多个信号的延迟。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，您还应运行 methodology_check DRC 规则平台来进行验证。参见 第 5 章中的运行设计方法中的 DRC 功能。

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播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播并在起点和终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，这些时钟都可传播出去，并在单元的扇出上重叠。实际上，一次只能传播一个时钟，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。

图4-27 中的实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。

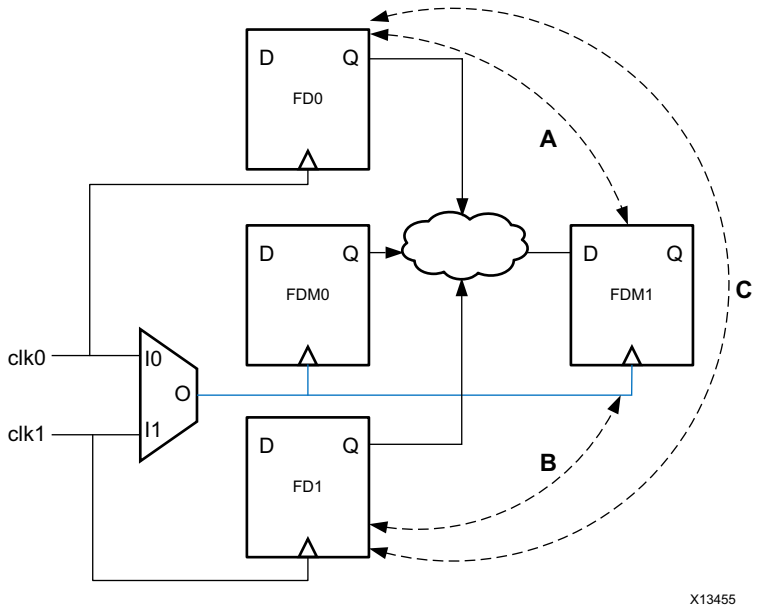


图 4-27 : 多路复用时钟

- 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：

- • 单位同步装置
- • 多位总线同步装置
- • 异步复位同步装置
- • 由 MUX 和 CE 控制的电路系统
- • 同步装置前组合逻辑
- • 多时钟扇入到同步装置
- • 扇出到目标时钟域

如需了解有关 report_cdc 命令的更多信息，敬请参阅： 《Vivado Design Suite 用户指南：设计分析和收敛技术》(UG906) [参照 21] 以及 report_cdc，位于 《Vivado Design Suite Tcl 命令参考指南》(UG835) [参照 13]。

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 与 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。

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。

在图4-28中，时钟 clk0 的周期为 5ns，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。

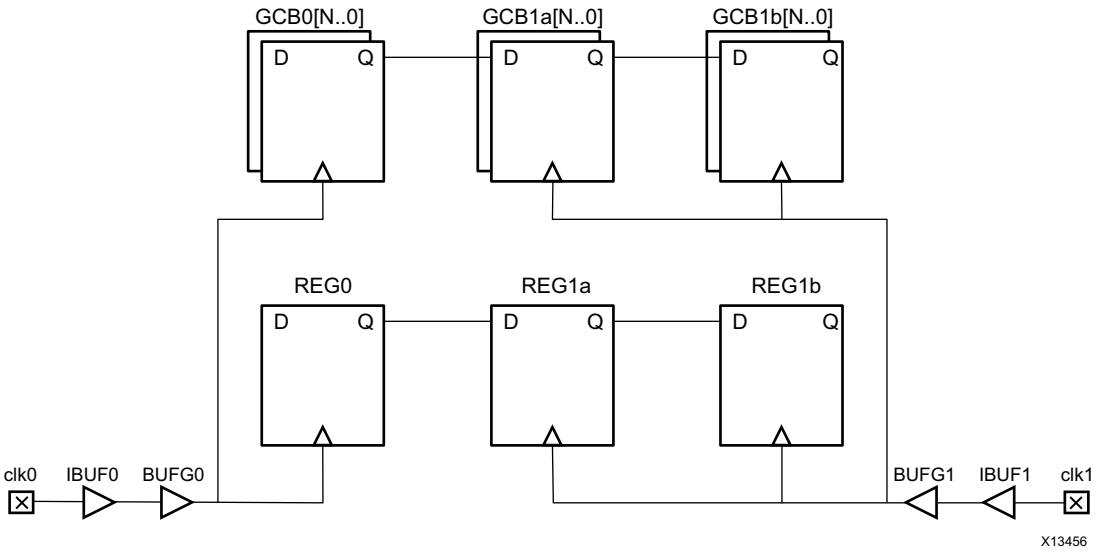


图 4-28 : 两个异步时钟间的多重交互

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）。
- 时序单元发送和接收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。

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播。

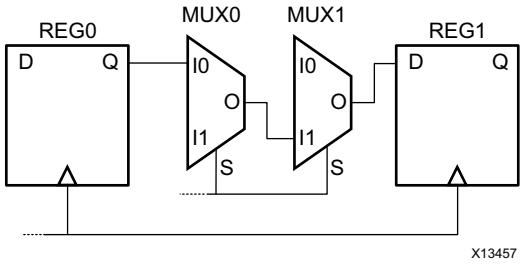


图 4-29 : 不能激活的路径

```
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”章节讨论过。
- 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接收沿的路径而言，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终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终点加以避免：

- 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

- 终点

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

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与终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。在以下的用例部分提供了一个实例。

多周期路径异常处理用例

多周期路径异常处理用例主要有两类：

- 降低“建立”要求，同时让“保持”不变
- 调整移位时钟之间路径的建立沿分析。

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。下面是三个实例：

- 实例 1：起点和终点使用相同时钟，时钟使能每 3 个周期激活一次
- 实例 2：从慢时钟到快时钟的路径
- 实例 3：从慢时钟到快时钟的路径

实例 1：起点和终点使用相同时钟，时钟使能每 3 个周期激活一次

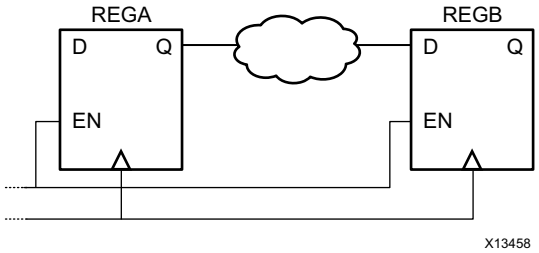


图 4-30 : 激活了具有相同时钟信号的寄存器

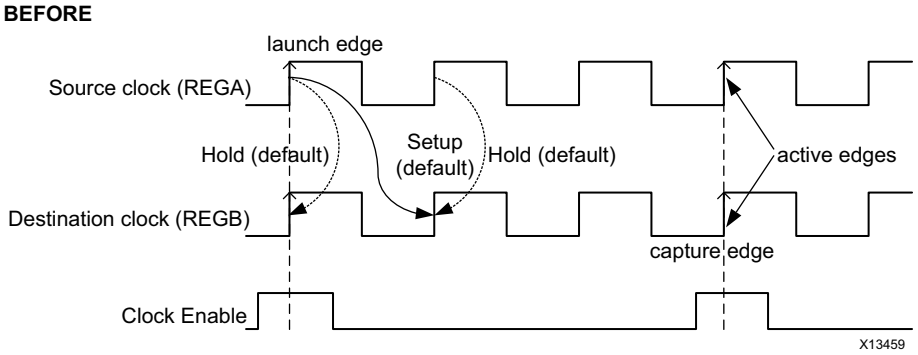


图 4-31 : 建立/保持检查的时序图

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
```

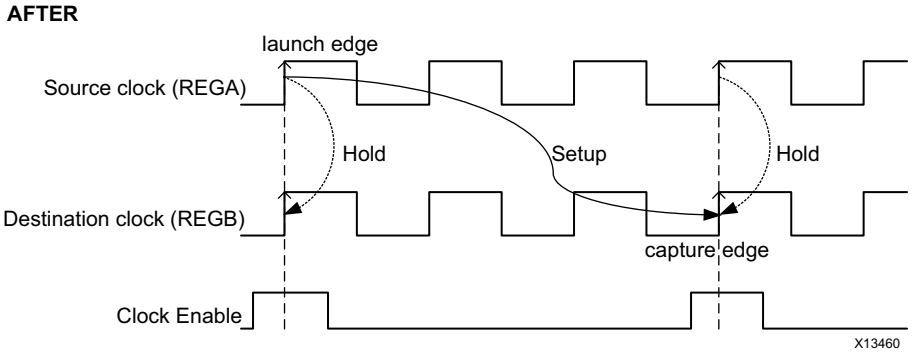


图 4-32 : 多周期规格描述之后修改建立/保持检查

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。

实例 2：从慢时钟到快时钟的路径

这种情况下，假设只有目标触发器被时钟使能信号控制，而且时钟使能信号在慢时钟的上升沿始终处于活动状态。建立路径乘数是 3。

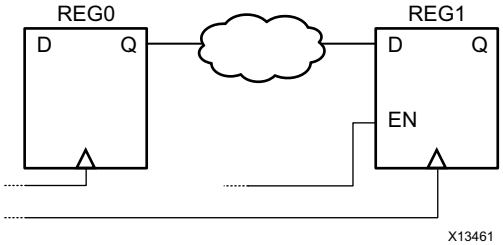


图 4-33 : 从慢时钟到快时钟交叉

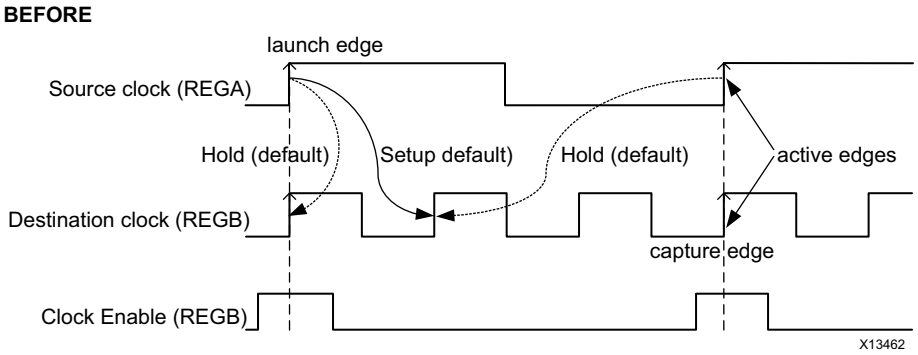


图 4-34 : 建立/保持检查的时序原理图–慢时钟到快时钟

约束：

```
set_multicycle_path -from [get_pins REG0/C] -to [get_pins REG1/D] -setup 3 -end
set_multicycle_path -from [get_pins REG0/C] -to [get_pins REG1/D] -hold 2 -end
```

`-end` 选项只用来针对目标时钟 （或终点时钟）修改建立和保持分析的时钟沿。已使用了正确的源时钟沿。

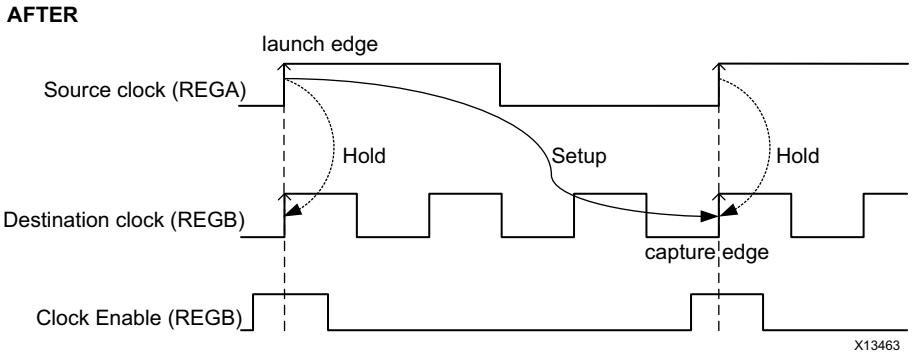


图 4-35 : 建立/保持检查的时序原理图–慢时钟到快时钟

实例 3：从慢时钟到快时钟的路径

这种情况与上一种情况类似 （从慢时钟到快时钟的路径），区别在于仅必须修改源时钟沿。约束实例：

```
set_multicycle_path -from [get_pins REGA/C] -to [get_pins REGB/D] -setup 3 -start
set_multicycle_path -from [get_pins REGA/C] -to [get_pins REGB/D] -hold 2 -start
```

调整移位时钟之间路径的建立沿分析

对两个时钟进行移位的主要原因是：

- 放松从一个时钟到后一时钟的建立路径，但要以收紧另一方向的路径为代价。这在 I/O 接口上比较普遍，用以调整器件接口处的时序。
- 在转发时钟与源同步接口的数据之间创建 90 度的相位移动。

默认情况下，时序引擎使用构成最悲观建立关系的源时钟和目标时钟的活动沿。当在目标时钟定义中插入一个正相位移时，建立关系对应于相移而非周期加相移，因为这是最严格的正向路径要求。下面给出一个实例：

```
Source clock waveform: rise @ 0ns, fall @ 5ns, rise @ 10ns
Destination clock waveform: rise @ 2.5ns, fall @ 7.5ns, rise @ 12.5ns
```

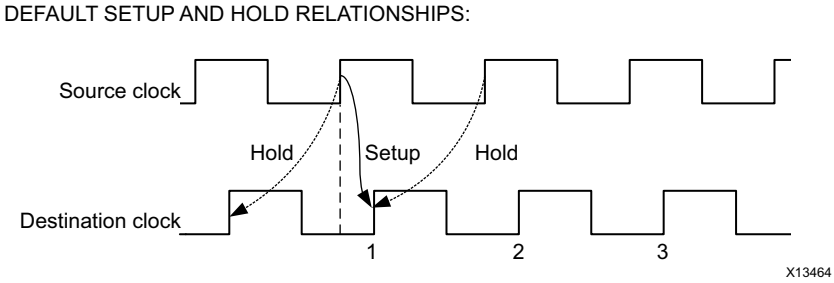


图 4-36 : 相移时钟的建立/保持沿

如果决定捕捉沿 #2 作为建立分析的有效捕捉沿，那么必须定义一个多周期路径约束。如果两个相移时钟之间的所有路径都必须进行修改，可直接在时钟上指定约束：

```
set_multicycle_path -from [get_clocks clk] -to [get_clocks clkshift] -setup 2
```

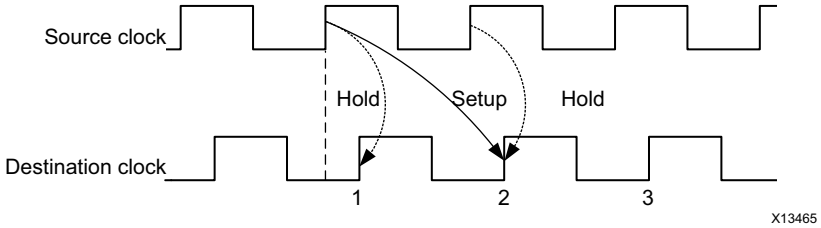


图 4-37 : 相移时钟的建立/保持沿–多周期技术参数后

重要提示 ： 这种情况下，没必要采用额外的 set_multicycle_path 约束来修改保持关系，因为已经依照建立关系对其进行了正确建立，而且所有上升时钟沿都处于活动状态。

对综合的影响

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，从而大幅改善时序 QoR （仅针对建立）。

对实现的影响

就综合而言，多周期路径异常处理有助于使基于时序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接收沿。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终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。终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，赛灵思建议指定终点引脚而非单元 （或时钟）。例如，终点单元 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，那么所有有效的终点引脚，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] -setup 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、层级引脚或叶单元输入引脚。该常数值在逻辑中传播，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播通过。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。

创建模块级约束

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模块创建独立的约束文件。这些模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。

模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模块边界之外的逻辑。

模块级约束规则

模块级约束必须符合如下规则：

1. 如果时钟需要在设计的顶层创建，则无需在模块级约束中定义这些时钟。

   但可在模块内使用 `get_clocks -of_objects` 命令查询这些时钟。该命令可返回穿过设计中特定对象的所有时钟。实例：

   ```
   set blockClock [get_clocks -of_objects [get_ports clkIn]]
   ```

   如果某个时钟需要在模块内部定义，那么它必须位于驱动实例化输入/双向缓冲器的输入/双向端口上，也可位于用来创建/变换时钟 （除了自动由时序工具处理的 MMCM/PLL 或特殊缓冲器）的单元的输出端。例如：

   - 带输入缓冲器的输入时钟
   - 时钟分频器
   - 千兆位收发器恢复时钟
2. 只有当端口直接连接到顶层端口并且 I/O 缓冲器在 IP 内部实例化时，才指定输入和输出延迟。实例：
   - 带输入缓冲器的输入数据端口
   - 带输出缓冲器的输出数据端口
3. 切勿在没有绑定到 IP 的两个时钟之间定义时序例外。
4. 切勿用名称来指代时钟，因为名称会根据顶层时钟名称而相应变化，或者如果模块被多次实例化也会导致名称变化。
5. 如果模块可在相同的顶层设计中多次实例化，则不添加布局约束。

将模块级约束读入顶层设计

Vivado Design Suite 可提供一种作用范围机制，用于将模块级约束读入顶层设计。该机制基于 current_instance 命令的行为，即所有基于名称的查询只能返回包含在当前实例中的对象。

当读入模块级约束时，则将当前实例设定为模块实例，这样只有属于该模块的对象才可以被约束。但存在几个例外情况：

- 时序时钟是全局的，而且可从设计的任意位置查询，包括从模块内部。必须小心谨慎地使用 get_clocks 命令，因为它可从模块外查询时钟。
- 可用 get_ports 命令查询模块定义的端口。根据模块实例在顶层设计中连接方式的不同，返回的对象类型会有所差别：
  - 如果模块端口直接连接到顶层端口，那么 get_ports 命令返回的则是顶层端口。

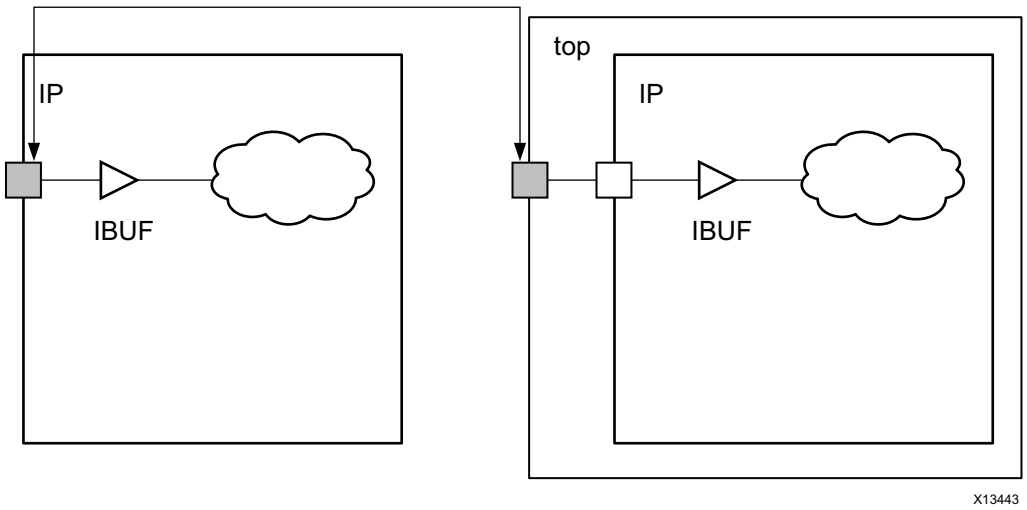


图 4-38 : 针对直接连接顶层端口的模块端口使用 get_ports 命令

  - 如果模块端口没有直接连接到顶层端口，那么 get_ports 命令返回的是模块接口的相应层级引脚。

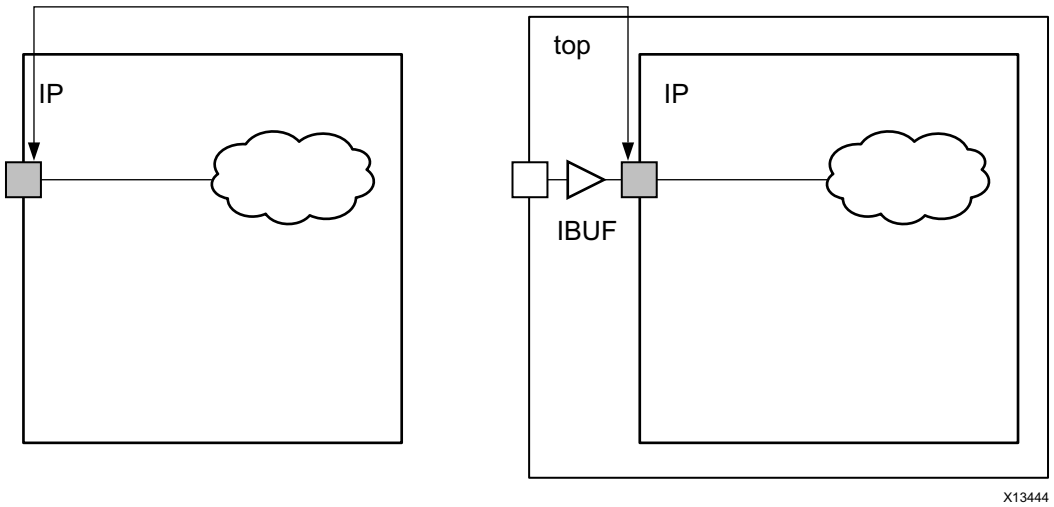


图 4-39 : 针对未直接连接顶层端口的模块端口使用 get_ports 命令

所有提供约束的 Vivado Design Suite IP 核都可采用这种范围机制。如需了解更多信息，敬请参阅：Vivado Design Suite User Guide: Using Constraints (UG903) [参照 18]。

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 User Guide: Using Constraints (UG903) [参照 18] 锁定布局与布线，包括与宏命令相关的布局
- 关于了解布局规划，敬请参见：《Vivado Design Suite 用户指南：设计分析和收敛技术》(UG906) [参照 21]
- 关于配置，敬请参见：《Vivado Design Suite 用户指南：编程与调试》(UG908) [参照 24]。

第 5 章

实现

实现简介

既然已经选定器件，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。实现过程将生成用于编辑器件的比特流。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。

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。下面是在进行综合设计时需要考虑的几点注意事项。如需了解有关综合的更多信息，敬请参阅下列资源：

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

设计移植过程中的综合属性

DONT_TOUCH 或 MAX_FANOUT 等综合属性会显著影响结果质量 (QoR)。设置这些属性的时候一定要谨慎。如果您的工程最初就是采用不同的综合工具运行，而现在要将该工程移植到 Vivado® Design Suite 中进行运行，那么任何已有属性的需求情况您都需要注意。应移除为控制此前的优化行为而添加的所有属性。

在最初创建 RTL 时一般不用 KEEP、 DONT_TOUCH和MAX_FANOUT 等属性。它们通常用来调整综合工具以使其达到最佳性能。由于所有的综合工具在优化上多少有些不同，因而在向新工具迁移时使用这类属性会对 QoR 造成不利影响。

建议： 从新的 RTL 开始，然后再专门针对新工具或新的要求应用属性。

准确的时序约束

由于 Vivado Design Suite 综合工具是基于时序的，因此要确保时序约束准确无误。参见：设计基准 (baseline)。如果设计需要时序例外约束，也必须提供。工具自动将约束从 XDC 文件发送到综合，以执行基于时序的运行过程。如果发送到综合与发送到布局布线的约束不相同，那么综合与布局布线就会采用不同路径。若出现这种情况，就很难实现时序收敛。

检查 HDL 代码

如果在综合后不能达到理想的 QoR，则应针对以下内容检查您的 HDL 代码和调用的逻辑：

- 评估信号 （例如置位与复位）以确定信号是否有必要。如果确实需要，应优先使用同步信号 （高电平有效）。
- DSP 和块状 RAM 具有内部寄存器。可使用这些寄存器。

要理解 HDL 代码如何影响调用和 QoR，敬请参见：第 4 章：设计创建。

调试综合后的设计

如果综合后的网表没有表现出所期望的行为，我们需要探究问题的根源。

使用细化设计

细化设计是分析或调试设计的第一步，是 RTL 代码本身的直接结果。用户使用细化设计视图，就可在运行综合之前对设计进行调试，从而在设计流程早期发现 RTL 代码问题。例如，可考虑图5-1所示的细化设计视图。

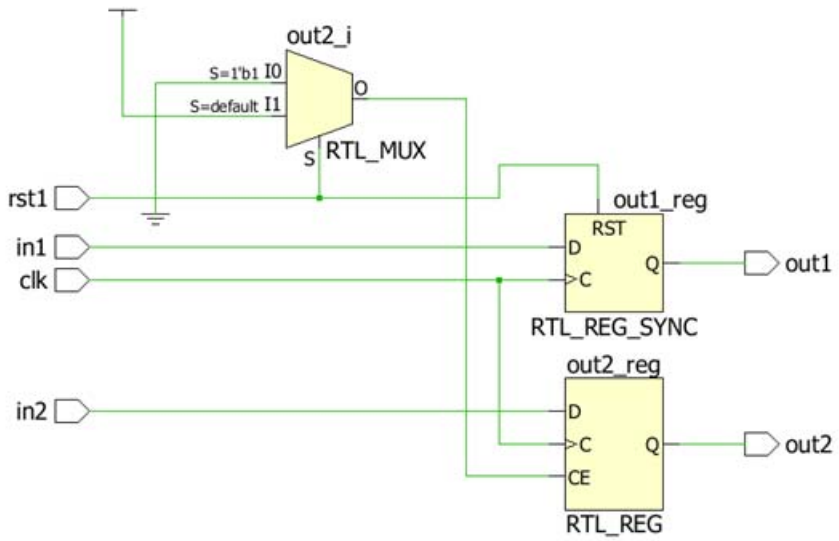
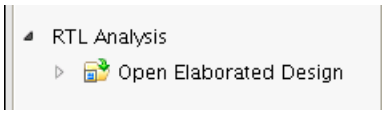


图 5-1 : Elaborated Design 视图实例

显然， `out2_reg` 由 `rst1` 信号使能。这最有可能存在编码错误。该视图的交叉探测功能可直接为您引出生成这个逻辑的 RTL。

此外，还可将“Elaborated”视图用于改善无法满足时序要求的设计。在对设计进行综合并找到关键路径后，在细化视图中可搜索相同的路径。这有助于发现能改进设计时序的 RTL 修改。在视图中很容易查看到大型的 MUX 结构或并非用流水线实现的 DSP 或 Block RAM 结构。

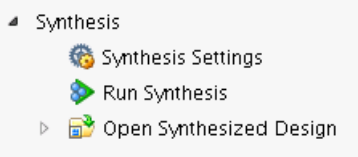
要打开“Elaborated”视图，应单击 Flow Navigator 中的“Open Elaborated Design”。



如需了解更多详情，敬请访问 《Vivado Design Suite 用户指南：系统级设计输入》(UG895) 中的链接。

使用 Synthesized Design 视图

此外，与“Elaborated”视图一样， “Synthesized Design”视图在调试综合的设计方面也非常实用。要想在设计完成综合后打开“Synthesized Design”视图，应单击Flow Navigator中的 “Open Synthesized Design”。



该视图基于用来创建网表的赛灵思原语之上。 “Synthesized Design”视图可用来查看 RTL 如何被转换为原语。视图将列出原语的所有属性。

考虑使用下面的实例：

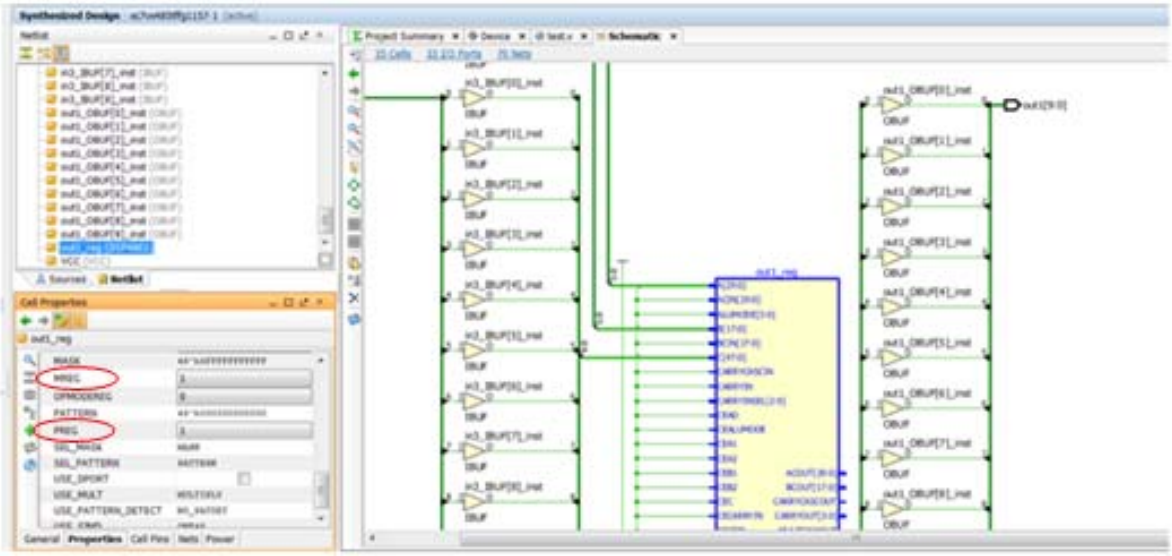


图 5-2 : Synthesized Design 视图实例

通过查看 `out1_reg` （图的左下侧）的属性，可以看到 DSP48E1 正在使用名为 MREG 和 PREG 的嵌入式流水线寄存器。这有助于确定如何用流水线实现您的设计。

此外，综合后的设计也是用户在分析设计的时序时应首先检查的地方。理解综合后时序非常重要，只有这样才能在运行实现前了解任何潜在的时序瓶颈。打开设计后，即可运行“Tools > Timing > Report Timing Summary”来查看时序信息。该报告可提供很多有用的信息，包括，例如：

- 各时钟和时钟之间路径的汇总
- 未约束的路径或 I/O
- 未给定时序约束的时钟

您还可在 Tcl 控制面板中检查时序约束，以确保已接受约束。赛灵思强烈建议您始终这样操作。如果约束未被接受，就无法保证 Vivado 工具在正确的路径上工作。

可使用 `report_timing` 命令检查所需路径，查看用户的约束是否已应用于该路径。例如，如果用户想要应用：

```
set_false_path -from [get_pins inst1/pin1] -through [get_cells inst2]
```

在应用上述约束后，应运行下列命令以确认路径时序裕量显示为“无限”。

```
report_timing -from [get_pins inst1/pin1] -through [get_cells inst2]
```

在综合的设计中，层级查看器可显示不同“Hierarchy”视图。

这在调试有可能存在占位面积问题的设计的过程中非常有用。不同模块可根据内部所包含的原语数量设定尺寸。点击“Netlist”对话框中的每一个“Hierarchy”视图，然后查看“Netlist Properties”对话框中的“Statistics”视图，您可以看到设计中原语的具体数量和类型。这些数字代表该层级和下面所有层级的原语数量。通过使用该视图可轻松看到何处出现了占位面积不足 (area blow up) 的问题。如需了解更多详情，敬请访问 《Vivado Design Suite 用户指南：设计分析和收敛技术》(UG906) 中的链接。

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。然后，通常可以通过属性对工具进行微调，这样就可以对设计做少量修改，进而实现不同的 QoR。例如：

- MAX_FANOUT 属性能增强特定网络上的最大扇出。
- RAM_STYLE 属性能强制 RAM 以特定方式来实现。

赛灵思建议在不使用任何属性的情况下运行工具，实现初步运行。然后再根据具体的设计和给出的结果添加综合属性，以获得所需的结果。

当把多个属性施加在单个信号上、或将不同属性施加于相互关联的信号上时需要非常谨慎。而工具会尝试逐一使用每个属性，但若这些属性是用来实现相互冲突的行为，（例如， KEEP_HIERARCHY 和 MAX_FANOU）那么在某些情况下，将无法如愿。

通常，当属性能被工具识别时 （例如 KEEP），该属性就会被工具使用，同时可在网表中看到其效果。然而，属性本身也不会再出现在输出网表中。当某个属性未被工具识别时，可以假定在随后的流程中将会使用到该属性。这种情况下，会将属性和值传递给输出网表。

如需了解综合所支持的属性信息，敬请查看： 《Vivado Design Suite 用户指南：综合》(UG901) [参照 16]。

在此需要特别提到几个属性，因为它们有时会导致一些需要引起注意的问题：

- KEEP 和DONT_TOUCH
- MAX_FANOUT

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。可将 KEEP 用于对综合工具的行为进行微调，以保留特定信号，也就是说，在综合期间关闭针对特定信号的特定优化功能。
- DONT_TOUCH 由综合工具使用，然后再被传递给布局布线工具，以便永远不会被优化。

另外，将 DONT_TOUCH 放置在层级上与放在信号上也存在差别。如果将该属性放置在信号上，则保持该信号；如果将该属性放置在某个层级上，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播，但该层级中的优化仍然继续。

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 Block 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。这一点非常重要！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，因为为了实现这一点，工具为了创建 IOBUF 必须改变层级。

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。

建议： 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，且复制工作做得比综合要好很多。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。

自下而上流程

通常需要将预先编译的较低层层级导入到 Vivado 工具中作为自下而上流程。自下而上流程可实现更快的运行速度，因为综合不需要每次都对这些模块进行编译和映射。另一方面，如果不支持综合进行跨边界优化， QoR 会降低。

创建更低层的网表

要建立自下而上流程，首先需要创建更低层的网表。为此，首先正常建立工程，再把较低层的层级指定为设计的顶层，同时根据层级部分相应指定约束文件。在运行综合之前：

1. 打开“Synthesis Settings”。
2. 在“More Options”行中输入：

   ```
   -mode out_of_context
   ```

这会告诉综合不要插入任何 I/O 缓冲器。

这一步很有必要，因为流程后期当 I/O 缓冲器插到设计剩余部分时，如果存在没有接触设计垫片的 IBUF 或 OBUF 组件，工具将出现错误。

注意 ！ 检查设计中是否存在输入或输出三态。由于 IOBUF 或 OBUFT 是赛灵思库中唯一能处理三态的组件，因此关闭 I/O 插入会导致错误发生。

如果较低层网表中需要输入或三态，应在 RTL 中对 IOBUF 或 OBUFT 组件实例化。即使打开 `out_of_context` 模式，综合工具也不会移除实例化的 I/O 缓冲器。

提示： `out_of_context` 是 Vivado Design Suite 的模式选项设置。其他综合工具也都支持此流程，但采用不同方式。如需了解其他工具如何执行该功能，敬请参阅第三方综合工具文档。

运行综合后创建 `.edf`文件，用作该部分设计的网表：

1. 打开综合设计。
2. 在 Tcl 控制台中输入：

   ```
   write_edif <name>.edf
   ```

当使用较低层网表时运行顶层设计

另一方面，当运行顶层设计并实例化较低层网表 （通常指的是黑盒）时，您必须执行如下操作。首先，网表必须实例化。实例化的方法对于 VHDL 和 Verilog 有所不同。两种情况下，都必须向综合工具描述较低层端口。对于 VHDL，采用组件声明来描述黑盒。

```
component <name>
port (in1, in2 : in std_logic;
out1 : out std_logic);
```

由于 Verilog 不具备 VHDL 所具有的等效元件，因此要使用封装程序文件用于向工具描述端口。该封装程序文件看起来像普通的 Verilog，但其只含有端口列表。

```
module <name> (in1, in2, out1);
input in1, in2;
output out1;
endmodule
```

两种情况下都要确保端口定义正确。如果不匹配，当在综合后插入低层网表时候就会出现错误。

组装设计

既然低层网表已创建，并且顶层设计在正确实例化网表，就可以像任何其他源文件一样将低层网表添加到 Vivado Design Suite 工程中。综合在顶层设计运行完成后，工具就会将低层网表插入到流程中。然后，布局布线正常执行。

如果网表中有低层 IOBUF，必须告诉综合工具不要向有问题的黑盒引脚插入任何 IOUBF。您需要指定 BUFER_TYPE 属性以防在综合中向特定端口插入 IOBUF。

其他综合工具也支持自下而上流程。如需了解如何利用第三方综合工具执行该功能，敬请参阅相应工具的技术文档。

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
- 检查综合日志
- 检查时序约束
- 满足综合后的时序要求

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。该命令的默认规则检查如下：

- 检查综合后与网表有关的 DRC。
- 检查 I/O、 BUFG 和其他特定的布局需求。
- 对属性和 MGT、 IODELAY、 MMCM、 PLL 的连线以及其他原语进行基本检查。

除了运行默认规则检查外，还需要运行 `methodology_checks` 和 `timing_checks` 规则检查。

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。确保运行方法规则检查 （参见：第 161 页的运行设计方法中的 DRC 功能)。

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，敬请参阅：第 4 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：

- 违规的相关指南
- 处理高逻辑级数
- 检查利用率
- 检查时钟树

违规的相关指南

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 Block RAM 提供的路径
- 由 SRL 提供的路径
- 包含未寄存的级联 DSP 模块的路径
- 具有逻辑级大数路径
- 大扇出路径

如需了解更多信息，敬请参阅：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，敬请参阅：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 RAMB 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。

检查时钟树

本节将讨论如何查看时钟树，包括内容如下：

- 时钟缓冲器的使用
- 时钟树拓扑

时钟缓冲器的使用

`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， BUFH 只能扇出到其时钟区域内的负载中。区域时钟缓冲器的无效布局约束或非常高的扇出会导致布局器出现问题。对于时钟缓冲器利用率很高的设计，有必要锁定时钟生成器以及一些区域时钟缓冲器，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。

如需了解有关建议布局增益的更多信息，请参阅：时序收敛。

时钟树拓扑

- 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 Skew。
- 对于 PLL 和 MMCM 的输出，使用相同的时钟缓冲器类型以最大限度降低 Skew。
- 寻找可引起额外延迟或 Skew （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。如需了解有关实现的更多信息，敬请参考下列资源：

- 《Vivado Design Suite 用户指南：实现》(UG904) [参照 19]
- Vivado Design Suite QuickTake 视频：设计流程简介

工程模式与非工程模式选项的对比

实现过程可在工程模式或非工程模式中完成。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。如需了解有关布局规划的更多信息，敬请访问： 《Vivado Design Suite 用户指南：设计流程简介》(UG892) 中的链路。

工程模式

工程模式的基础是运行。您可以创建和推行采用不同综合结果和设计约束 （或两者同时使用）的最新实现运行方式，以增大实现解决方案的空间，并找到最佳结果。在工程模式下， Vivado IDE 可支持您在单个设计上运行多种策略；定制化实现策略以根据您的设计要求调整算法；并可保存定制实现策略，以便用于其它工程。一旦为您的设计找到最佳策略后，就可以在具有类似特性的未来设计中使用该策略。

非工程模式

在非工程模式下，使用定义设计流程的 Tcl 脚本运行实现过程。

建议流程

下面给出了最少的命令列表，这些命令在读入设计后必须执行，用以生成有效的比特流：

- `link_design`
- `opt_design`
- `place_design`
- `route_design`
- `report_drc`
- `report_timing_summary`
- `write_bitstream`

时序约束应该完整并且正确。应以正时序裕量满足这些约束，以确保设计可在硬件中工作。

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，您可在 `route_design` 之后运行 `place_design -post_place_opt`，以便对几个关键路径不满足时序要求的布线后的设计进行布局后优化。布局器使用实际时序延迟执行布局后优化。用户需按此步骤再次运行`route_design`。

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。在布线后期阶段调用`phys_opt_design`将会对任何可能未布线的网络重新布线。所以，布线后的`phys_opt_design`无需进行另一个`route_design`的显式运行。

运行设计方法中的 DRC 功能

鉴于方法的重要性，Vivado 工具可提供一组“设计规则检查 (DRC)”功能，专门用于检查方法合规性。根据设计流程所处的阶段应使用不同类型的 DRC。 RTL 的 Lint 风格检查运行在细化的 RTL 设计上；基于网表的逻辑和约束检查运行在综合后的设计上；实现和时序检查运行在实现后的设计上。

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_drc  -ruledeck methodology_checks
```

如需从 IDE 中运行上述检查，打开待验证设计，并运行 ReportDRC 命令。当出现对话框后，选择方法检查规则选项，如图5-3所示。

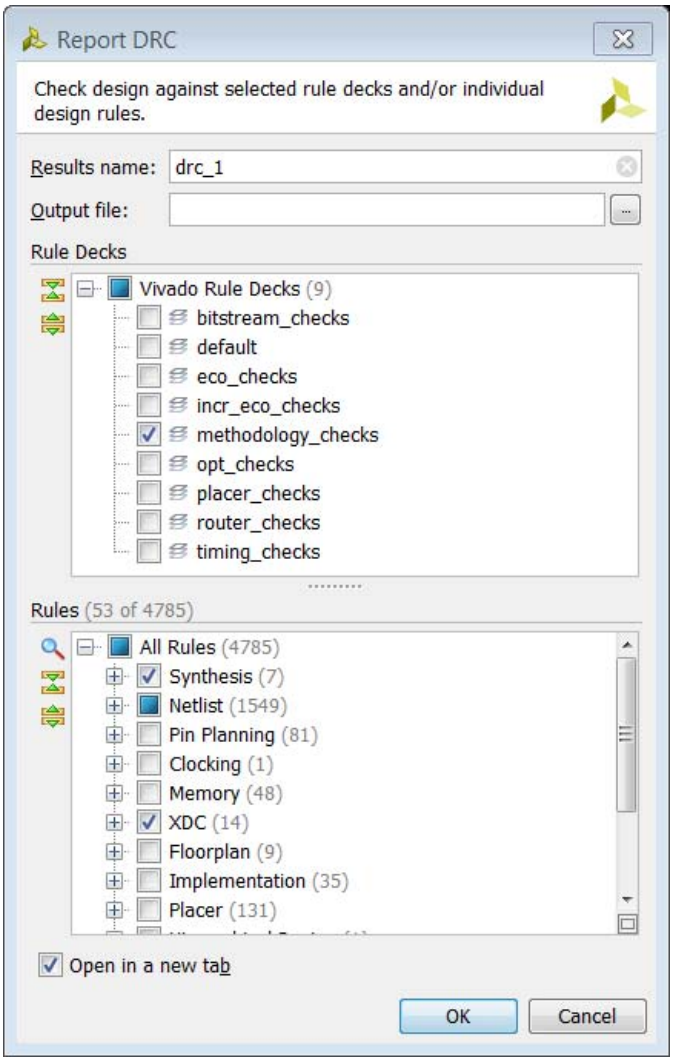


图 5-3 : Report DRC 对话框

违规情况 （如有）会在 DRC 窗口列出，如图5-4 所示。

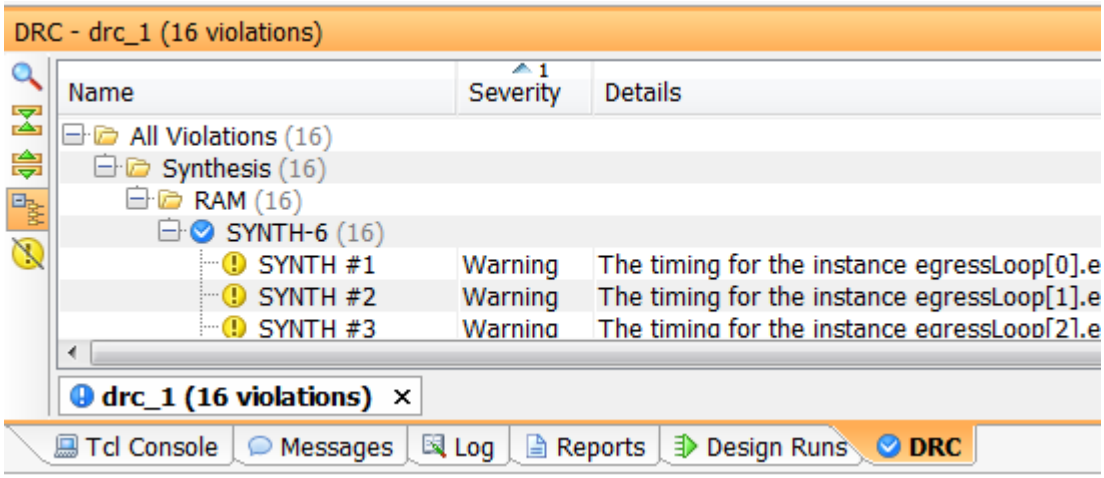


图 5-4 : DRC 违规情况

如需了解有关运行设计方法 DRC 的更多信息，敬请参阅： 《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]。

在权衡是否不必清除设计中某项特定 DRC 违规时，请确切了解该项违规的内容及其含义，并弄清该项违规不会对特定设计造成消极影响的原因。对于赛灵思提供的 IP 核，已经对 DRC 违规进行过评估和核查。

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。如需了解更多信息，敬请参阅：指令。

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。

策略针对特定的工具和版本。每个 Vivado Design Suite 的主要版本都包含针对特定版本的策略。

策略根据其用途分为多个不同类别，并以类别名称作为前缀。参见：表 5-1。

性能策略的目的是提高设计性能，而代价则是运行时间加长。例如， Performance_Explore 策略有助于改进多种设计方案结果，但运行时间会显著延长。

表 5-1 : 策略类别

类别	目的
性能	改善设计性能。
占位面积	减少 LUT 数量。
功耗	添加完整的功耗优化。
流程	修改流程步骤。
拥塞	减少拥塞和相关问题。

重要提示： 包含 SLL 或 SLR 的策略能提供针对 SSI 器件的附加控制。

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。一次只能指定一条指令。指令选项与其他选项不相容。

如需了解有关策略和指令的更多信息，敬请参阅： 《Vivado Design Suite 用户指南：实现》(UG904) [参照 19]。

中间步骤和检查点

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。

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。

增量流程

当此前已实现的设计与当前设计存在一定相似度时， Vivado Design Suite 的增量布局布线功能就会重新利用已有的布局布线数据来缩短实现运行时间并生成更多可预测的结果。当设计有 95% 以上的相似单元、网络和端口时，增量式布局布线的运行时间将比一般布局布线平均缩短 2 倍。

平均运行时间的提升程度会随参考设计与当前设计之间相似度的降低而有所下降。

若相似度低于 80%，则使用增量布局布线功能只有很微小的优势或者基本没有优势。

适用于增量流程的良好的使用模型包括：

- 修复已实现的设计方案，它们已接近满足时序要求，并只需要较小的局部修复。
- 为已实现的设计添加调试内核。
- 重新设计对有限数量逻辑产生影响的关键局部路径。
- 创建全新的设计修订版本。

  注释： 这倾向于具有较低的相似度。

除了缩短运行时间外，增量编译对没有发生变化的设计部分造成的破坏也很小，因此能减少时序变化。

能否有效重新利用参考设计的布局布线取决于两个不同方案之间的设计差异。有时，源设计中细微的变化会对最终结果产生很大的影响，从而导致重新利用变得困难重重或效果较差。

如需了解更多信息，敬请参阅： 《Vivado Design Suite 用户指南：实现》(UG904) [参照 19] 中的“利用增量编译器节省布局器布线器运行时间”。

RTL 微小变化的影响

尽管综合试图最大限度地减少网表名称的变化，但如下所列的 RTL 微小变化有时会导致较大的设计变更：

- 加大调用存储器的尺寸
- 加宽内部总线
- 将数据类型从无符号变为有符号

改变约束和综合选项的影响

类似地，改变约束和综合选项也会对增量布局布线造成很大的影响，例如：

- 改变时序约束和重新综合
- 保存或取消逻辑层级
- 启动寄存器的重定时功能 (re-timing)

如需了解更多详情，敬请访问 《Vivado Design Suite 用户指南：实现》(UG904) 中的链接。

验证网表质量

为了确保获得最佳实现结果，检查启动网表的质量十分重要。如果综合阶段没有完成检查，或者您无法确定网表质量，请参见：综合后的步骤了解有关检查网表质量的说明。

根据包含设计描述内容的源文件的特点以及设计状态，采用以下 Tcl 命令可将已综合的设计读取到存储器中：

- `synth_design/launch_runs synth_1`
- `open_checkpoint`
- `open_run`
- `link_design`

表 5-2 : 可使用 Tcl 命令中的模式

命令	工程模式	非工程模式
synth_design	X (launch_runs synth_1)	X (synth_design)
open_checkpoint		X
open_run	X	
link_design		X

如需了解更多信息，敬请参阅： 《Vivado Design Suite 用户指南：实现》(UG904) [参照 19]。

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 模块），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，传播常数输入，以及进行块状 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。

影响逻辑优化的约束和属性

Vivado Design Suite 在逻辑优化过程中要顾及 DONT_TOUCH 和 MARK_DEBUG 属性，并不会将具有这些属性的网络优化掉。如需了解更多信息，敬请参阅： 《Vivado Design Suite 用户指南：综合》(UG901) [参照 16]。

- 将 MARK_DEBUG 放置在采用 Vivado Logic Analyzer 工具进行探索的候选网络中，并将含有 MARK_DEBUG 的网络连接到 Slice 边界上以确保该网络被探针探测到。
- 通常将 DONT_TOUCH 属性放置在叶单元上以防这些单元被优化。位于分级单元上的 DONT_TOUCH 保留了单元边界，但单元内部仍会发生优化。
- DONT_TOUCH 属性可能被用于具有范围 （scoped）约束的设计段和 IP 核，以确保被施加约束的对象不会被优化掉。

逻辑优化指令

有些指令用来改变 `opt_design` 命令的行为，以运行强调或不强调占位面积削减效果的多次优化过程，并将 LUT 重新映射到默认流程中。

如需了解有关逻辑优化指令的更多信息，敬请参阅： 《Vivado Design Suite 用户指南：综合》(UG901) [参照 19]。

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详情。

功耗优化

如需了解相关设计的功耗优化，敬请参阅：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。与其他实现命令一样， Vivado Design Suite 布局器可处理和更新内存里的设计。

影响布局的约束

以下约束将影响 Vivado Design Suite 布局器中设计对象的布局：

- I/O 约束 （实例：IOB、 IOSTANDARD）
- 位置约束）（实例：LOC、 PBLOCK、 PROHIBIT)
- 时序约束 （实例：create_clock）
- 网表约束 （实例：LOCK_PINS、 CLOCK_DEDICATED_ROUTE)
- RPM 和 XDC 宏 （实例：create_macro）

如需了解有关布局约束的更多信息，敬请参阅：Vivado Design Suite User Guide: Using Constraints (UG903) [参照 18]。

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时钟拓扑结构造成，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`，以便查看按照时钟区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，敬请参阅：时序收敛。

如果 Vivado Design Suite 布局器没能找到用于时钟和 I/O 布置的解决方案，布局器就会报告设计方案违反了哪些布局规则，并简要描述受影响的单元。

布局会因以下几个原因而失败，包括：

- 由约束冲突引起的时钟树问题
- 布局器无法解决的过于复杂的时钟树问题
- RAM 和 DSP 模块布局与其他约束 （例如 Pblocks）相冲突
- 资源的过度使用
- I/O 单元要求和规则

为了解决布局问题，应仔细分析所生成的错误信息。在多数情况下，这些信息指的是无法实现有效解决方案的中间布局。尝试消除可能导致布局问题的布局约束，或者对复杂时钟树问题来说，约束时钟缓冲器可能会实现成功布局。

SSI 布局

布局策略

利用内置的布局算法，工具试图：

1. 采取不超出 SLL 资源的方式对设计进行布局。
2. 限制必须穿过 SLR 元件的时序关键路径的数量。
3. 对资源进行平衡，但不会使用给定资源过度填充 SLR。
4. 将 SLL 交叉数量限制到最低。

通过遵循这些策略，工具会尝试在打破平衡布局的同时满足性能要求。

其他影响 SLR 选择的因素

其他可能也影响 SLR 选择的设计与实现因素，包括：

1. 引脚布局
2. 时钟选择
3. 资源类型
4. 物理约束，例如布局规划 (PBlocks) 和 LOC 约束
5. 时序约束
6. I/O 标准和其他约束时序约束

赛灵思建议您允许工具分配 SLR 元件，并执行智能引脚布局、时钟选择和其他设计选择。

如需了解更多相关信息，敬请参阅本章中的下列部分：

- 布局器指令介绍专门用于基于 SSI 设计的指令类型。
- 策略介绍专门用于基于 SSI 设计的策略。

手动 SLR 分配

当工具无法找到满足设计要求的解决方案时，或者当运行间的重复性比较重要时，或许有必要进行手动 SLR 分配。

执行手动 SLR 分配

执行手动 SLR 分配：

1. 创建大型 PBlock （区域群组）。

2. 将设计的某些部分分配到那些区域。

将设计的较大部分分配到单个 SLR：

1. 创建一个包含单个 SLR 的 PBlock。
2. 将逻辑的相关层级分配至该 PBlock。

尽管可将逻辑分配到多个相邻的 SLR 元件中，但仍须确保 PBlock 包含整个 SLR。

不要创建穿过 SLR 边界但无法约束整个 SLR 的 PBlocks。这样做会使自动 SLR 布局算法难以将布局合法化。

手动SLR 分配指南

当您手动向 SLR 元件分配逻辑时，赛灵思建议您:

1. 采取不超出 SLL 资源的方式对设计进行布局。
2. 限制必须穿过 SLR 元件的时序关键路径的数量。
3. 对资源进行平衡，但不会使用给定资源过度填充 SLR。
4. 将 SLL 交叉数量限制到最低。

布局器指令

因为布局对整体设计性能通常起主要影响作用，因此为您提供了多个布局器指令，可针对不同情况探索解决方案空间。

提示： 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。

表 5-3 显示了哪种指令有利于哪类设计。

表 5-3 : 常见情景

指令类型	适用设计
模块布局	RAM、 DSP 模块或两者兼有的设计
NetDelay	设计中预计会有很多长距离网络连接，以及设计中有很多扇出到多个不同模块的网络。
SpreadLogic	设计中具有非常高速的连接，可能造成拥塞。
ExtraPostPlacement 选项	所有设计类型
SSI	可受益于不同类型分区以缓解拥塞或提升时序的 SSI 设计。

如需了解有关布局器指令的更多信息，敬请参阅： 《Vivado Design Suite 用户指南：实现》(UG904) [参照 19]。

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、重新 Re-timing、保持固定以及布局提升。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 模块涉及到多个在 `route_design` 后产生时序失败的 Block 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）。如需了解有关物理综合的更多信息，敬请参阅： 《Vivado Design Suite 用户指南：实现》(UG904) [参照 19]。

影响物理优化的约束

时序约束会影响物理优化。大部分物理优化都在 WNS 百分率之内具有负裕量 （在时序路径上执行）。对网表进行修改，并逐步添加变更。只有在评估时序裕量、占位面积和功耗后才能执行变更。

Vivado Design Suite 在物理优化过程中也会顾及 DONT_TOUCH 和 MARK_DEBUG 属性，其与在逻辑优化过程中需要顾及这两个属性的原因相同。

物理优化指令

几个物理优化指令可以让您针对不同情况探索解决方案空间。

提示： 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。

如需了解有关物理优化指令的更多信息，敬请参阅： 《Vivado Design Suite 用户指南：实现》(UG904) [参照 19]。

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。

影响布线的约束

以下约束将会影响 Vivado Design Suite 布线器中的布线：

- 固定布线 （例如：FIXED_ROUTE)
- 引脚锁定约束 （例如：LOCK_PINS）
- 时序约束 （例如：create_clock）

相互冲突的约束会在布线器中造成错误。

如需了解有关布线约束的更多信息，敬请查阅：Vivado Design Suite User Guide: Using Constraints (UG903) [参照 18]。

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。使用 RTL 综合中的定向扇出优化，或通过物理优化可以记录下拥塞区域的地址。可以保存全部或者部分设计层，以避免跨境优化，并可降低网表密度。还可以使用布局规划约束来缓解拥塞。

如需了解更多信息，敬请参阅：时序收敛。

中间布线结果

当布线失败时， Vivado Design Suite 布线器仍继续提供尽可能完整的设计方案，以协助调试。如果布线不完整，还需要进行手动参与。在下列提示的帮助下确定后续步骤：

- 运行 `report_route_status` 并检查“Nets with Routing Errors”部分。找到网络然后创建原理图，然后查找类似高扇出网络或时钟规则违规这样的区域。运行 DRC 检查器有时能发现时钟规则违规。
- 如果物理布局约束 (Pblock) 导致该问题，将全部 Pblock 约束清除后重新生成一个版本。

- 审核 vivado.log 文件的布线部分，找到“Phase 3.2 Budgeting“项。总体拥塞程度记录在“级别 (level)“中，其中“7”表示最为严重。 7 级拥塞指覆盖 2^7 (128) 个模块区域的布线利用率接近 100%。同时报告布线方向 （东、西、南、北）。 “INT_XXX”数值是器件布线资源视图中可见的互联布线模块的坐标。
- 打开布线后设计检查点文件(.dcp)然后再打开“器件垂直与水平指标 (Device Metric Vertical and Horizontal)”拥塞叠加功能。这个视图给出了对每一个逻辑块 （比如：一个 CLB）布线利用率的预估。如果预估值超过 100%，布线器就必须绕道，以便为进入拥塞区域的所有网络成功布线。查找日志文件拥塞报告中所报告模块区域中的热点区。在器件视图中选择热点中的全部单元并生成原理图。寻找可以使用全局时钟缓冲器进行缓冲的出现了大量扇出信号的网络，以释放局部布线资源。也可以确定主要扇出区域，以创建布局规划约束，或者用不同的选项或属性进行综合。
- 使用重点针对拥塞的布局指令重新运行设计。

如需了解有关仅对特定网络重新布线的更多信息，敬请参阅：使用重入布线模式。

布线器指令

几个布线器指令可以让您针对不同情况探索解决方空间。

提示： 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。

如需了解有关布线器指令的更多信息，敬请参阅 《Vivado Design Suite 用户指南：实现》(UG904) [参照 19]。

使用重入布线模式

route_design命令本质上是可重入。对部分布线的设计而言， Vivado Design Suite 布线器会利用现有布线作为起点，而非从头开始布线。重入模式通常是交互式运行以满足具体的布线问题，例如预先布线关键网络并在完全布线之前锁定资源；以及手动解除非关键网络的布线以释放布线资源用于更关键的网络。如需了解有关重入模式的更多信息，敬请参阅： 《Vivado Design Suite 用户指南：实现》(UG904) [参照 19]。

时序收敛

时序收敛是指设计能够满足所有时序要求。本节将介绍如何让您的设计实现时序收敛。用户经常试图在实现阶段收敛时序。但正如第 1 章：引言所述，如果我们进入综合阶段时具有正确的 HDL 和约束，时序收敛会更容易。我们再次给出图1-3 以概括总结在综合阶段利用提升的 HDL、约束和综合选项进行设计反复循环的重要性。如需了解更多详情，敬请访问 《Vivado Design Suite 用户指南：设计分析和收敛技术》(UG906) 中的链接。

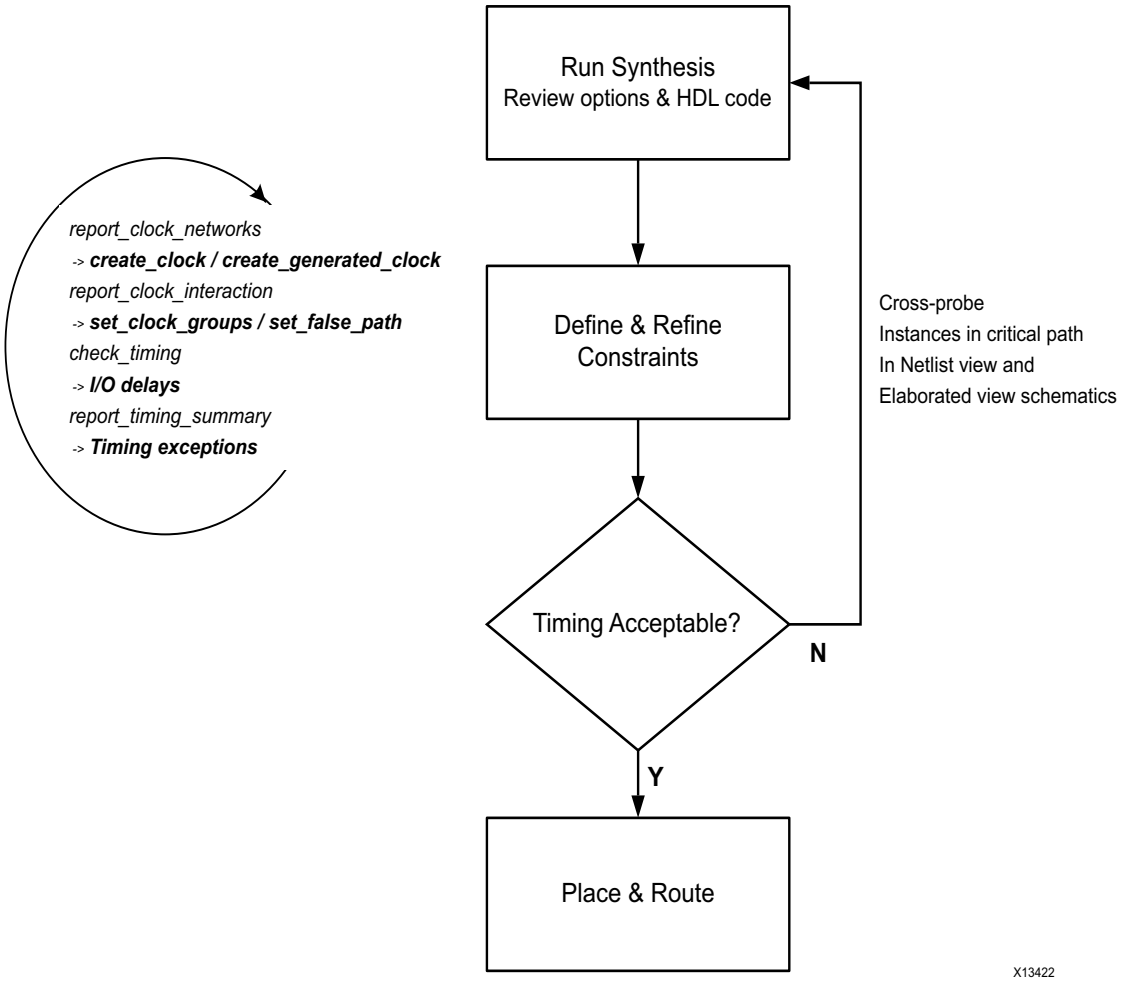


图 5-5 :

图 5-5 : 实现快速收敛的设计方法

遵循如下指南：

- 若最初不满足时序要求，需评估整个流程的时序。
- 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 (<-1ns),以便确定遗漏的或者不恰当的约束。
- 重新平衡设计选择、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）。

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。一旦时钟 （包括生成的时钟）被完全约束，所有在设计中包含起点和终点的路径 （所有寄存器至寄存器路径）都会被自动约束。这样就建立一种简单机制，即使设计在不断变化也能识别内部器件的时序挑战。由于设计还可能存在时钟域交叉，因此基准 (baseline) 约束还应包括指定时钟 （和生成时钟）之间的关系。

基准 (baseline) 设定的主要理念是创建一个正确的极简约束集，该约束集要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。因此， I/O 时序定义和收敛要等到过后才调用，即设计已有了很大进展而且对 I/O 时序有更好的了解。相应地，基准 (baseline) 约束不包括 I/O 时序约束。

赛灵思建议您在设计过程中尽早创建基准 (baseline) 约束。设计若发生任何较大变化，应依照这些基准 (baseline) 约束来调整 HDL 的时序。定期调整设计更新的时序能确保任何时序瓶颈在刚出现时就能被发现。

提示： 敬请参阅：定义基准 (baseline) 束以创建基准约束。敬请参阅：了解时序报告以了解和解读时序报告。如果您遇到基准 (baseline) 约束的时序问题，敬请参阅：调试和修复时序问题。

当知道和确定 I/O 时序需求时，可能会添加 I/O 约束。

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。

在按设计流程逐步操作和优化设计约束时，请填写附录 A：基线(baselining)与时序约束验证流程提供的调查表。该流程有助于用户跟踪实现时序约束的进展情况和确定潜在的瓶颈。

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 （总体负时序裕量）是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。最差设置建立/恢复时序裕量为 WNS （最差负时序裕量）。
- THS （总体保持时序裕量）是整个设计或针对特定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）。
- TPWS （总体脉冲宽度时序裕量）是整个设计或针对特定时钟域中每个时钟引脚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）是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*是指为安全采集数据在下个有效时钟沿到来前可提供新稳定数据之前所需的时间。
- *保持需求*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。
- 恢复时间是指无效状态下的异步复位信号切换到下一个有效时钟沿所需的最短时间。
- *移除时间*是指有效时钟沿之后、异步复位信号安全切换到非活动状态之前的最小时间。

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。

一旦在时钟网络上定义时序时钟，时序分析就会对目标触发器的数据引脚在最悲观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。

如需了解有关时序分析的更多信息，敬请访问 《Vivado Design Suite 用户指南：设计分析和收敛技术》(UG906) 中的链接。

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。应满足以下指标：

- 明确的约束
- 无时序违规

明确的约束

- 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。

除了传输时携带赛灵思IP协议的约束之外，输入与输出延迟约束，以及某些路径特定时序例外均不适用于基准约束。

无时序违规

- 建立/恢复 （最大分析）：WNS > 0 ns 且 TNS = 0 ns
- 保持/移除 （最小分析）：WHS > 0 ns 且 THS = 0 ns
- 脉宽：WPWS > 0 ns 且 TPWS = 0 ns

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。该命令可作为独立命令运行，但也是 `report_timing_summary` 的一部分。

`Check_timing` 命令报告如下情况。这些情况指出时序定义中的缺失或错误，或正确满足时序要求的含义：

- no_clock 或 constant_clock
- unconstrained_internal_endpoints
- no_input_delay
- no_output_delay
- multiple_clock
- generated_clocks
- loops
- partial_input_delay
- partial_output_delay
- latch_loops

no_clock 或 constant_clock

定义时序时钟未接入的时钟引脚数。报告常数时钟引脚。检查是否有时钟约束丢失。或者是否有时钟引脚被误接到常数时钟。

unconstrained_internal_endpoints

无时序要求的路径端点的数量 （不包括输出端口）。该数值直接与`no_clock`检查报告的丢失时钟定义有关。

no_input_delay

一个输入延迟约束都没有的 non-clock 输入端口数量。检查 `set_input_delay` 是否丢失。

no_output_delay

一个输出延迟约束都没有的 non-clock 输出端口数量。

multiple_clock

接入一个以上时序时钟的时钟引脚数。如果有一个时钟树中存在时钟多路复用器，就会出现这种情况。默认情况下共享相同时钟树的时钟会一起定时，这不能反映实际的时序条件。在给定时间内只有一个时钟能出现在时钟树上。

如果您不相信时钟树具有 MUX，应检查时钟树以便了解多个时钟如何以及为什么连接到达特定时钟引脚。

generated_clocks

所生成时钟的数量，且生成时钟均参考位于另外的时钟树扇入锥形区域内的某个主时钟源。

loops

设计中组合环路的数量。这些环路自动被 Vivado Design Suite 时序引擎打破，以报告时序。

partial_input_delay

只有一个最小输入延迟或最大输入延迟约束的输入端口数。不会既针对建立分析又针对保持分析来分析这些端口。

partial_output_delay

只有一个最小输出延迟或最大输出延迟约束的输出端口数。不会既针对建立分析又针对保持分析来分析这些端口。

latch_loops

检查和提醒通过设计中锁存器的环路。这些环路不作为组合环路的一部分来报告，并会借用相同路径上的计算资源，从而影响锁存时间。

修复被 check_timing 标记的问题

并不是所有检查都同等重要。当检查和修复由 `check_timing` 标记的问题时，对下面的检查项按照重要性进行了分类（最重要到最不重要）。

无时钟和未约束的内部端点

这是最重要的检查项，用来确定设计中的内部路径是否被完全约束。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。

如果内部未约束的端点数为零，不应掉以轻心。代表所有内部路径都针对时序分析进行了约束，这只能但不能保证约束值是正确的。

生成时钟

`Generated_clocks`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

一个好的设计没有任何组合环路。时序环路将被时序引擎打断。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： 首先使用基准 (baseline) 约束。一旦经过验证，再用 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播。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： 由于 XDC 是 Tcl 程序，因此约束有顺序关系。

调试和修复时序问题

下面的表格给出了关于调试和修复时序报告中时序错误 （如有）系统方案的快速指南。

表 5-4 : 调试和修复时序问题的步骤

步骤	具体内容章节
检查所有时钟及其关系都被正确定义	定义基准 (baseline) 束
检查时钟 Skew 和不确定性不是太高	时钟偏差与不确定性
检查路径中的逻辑级数不是太多	数据路径延迟和逻辑级数
检查路径使用的是最优资源 （单元/引脚）	MMCM 频率综合
检查没有过多不必要的控制集	控制集

在确定上述各项情况良好后，检查剩下的违规，以便确定下一个行动步骤，此步骤涉及到带后台的不同选项，包括在所有其他方法都不能奏效的情况下，如何手动创建布局规划。

如欲了解更多有关其他设计分析技巧的详情，敬请访问 《Vivado Design Suite 用户指南：设计分析和收敛技术》 (UG906) 中的链接。

定义基准 (baseline) 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。 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什么内容。

- 步骤 1：确定必须创建哪些时钟
- 步骤 2：确认没有时钟遗漏
- 步骤 3：识别异步时钟域

步骤 1：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 控制台中重设时序，以确保删除所有时序约束。这样就能确定控制台 (slate) 为干净的。

然后生成一个时钟网络报告，以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。使用时钟创建向导为每个时钟指定合适的参数。

步骤 2：确认没有时钟遗漏

一旦时钟网络报告显示所有时钟网络都已设置了约束，就可以开始验证生成时钟的精确性。由于 Vivado 工具会自动在时钟修改模块 （例如 MMCM、 PLL 和 BUFGCTRL）中传播时钟约束，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。

`Report_timing` 结果显示所有时钟都被传播。主时钟 （用 `create_clock` 创建）与生成时钟 （由时钟修改模块创建）之间的区别在属性字段中显示。

- 被传播的仅显示为 (P) 的时钟为主时钟。
- 被生成的时钟既可以显示为传播 (P) 也可以显示为生成 (G)。

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，敬请参阅：Vivado Design Suite User Guide: Using Constraints (UG903) [参照 18]。

```
Attributes
  P: Propagated
  G: Generated
  V: Virtual
  I: Inverted

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}
```

图 5-6 : Report_Clocks 显示了从母时钟生成的时钟

步骤 3：识别异步时钟域

一旦完成时钟约束验证，必须识别出跨路径的异步时钟域。

注释： 本节不介绍如何正确跨越时钟域边界，只介绍如何识别哪里有边界跨越以及如何约束。

查看时钟域交互情况的最佳方法是使用 `report_clock_interaction`。该报告给出一个源时钟和目标时钟的表格。每个单元的颜色指明相应行列代表的时钟之间的交互特性。图5-7 显示了一个时钟交互报告实例。

图 5-7 : 时钟交互报告实例

表 5-5 介绍了报告中每种颜色的含义。

表 5-5 : report_clock_interaction 颜色

颜色	含义	备注
黑	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	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	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	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被声明为异步，或者它们是否应共享共用的主时钟源。
橙	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。	检查为什么只有少数路径得到"用户自定义异常"处理？是否所有路径都应该得到"异常"处理？
蓝	时钟域之间有交互，而且路径没有被定时。	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	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过约束设计。

识别未共享主时钟的时钟对

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，使用共用主时钟 (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。如需了解正确设计夸路径时钟域的更多信息，敬请参阅：第 4 章：设计创建。

明确严格的时序要求

对于每个时钟对，时钟交互报告还会显示源时钟到目标时钟之间所有路径的路径要求。按路径要求 (WNS) 对列排序，可查看设计中最严格的要求。图5-7 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。

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 ...}。

以下情况会对时钟对提供最严格的要求：

- 250 MHz 时钟在 4 ns 具有一个上升沿，并且
- 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，因为接收沿不能与发送沿相同。

由于这是一个相当严格的时序要求，因此必须采取其它步骤。

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`
- `false_path`
- `max_delay_path`
- `multicycle path`

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，这一点很关键。

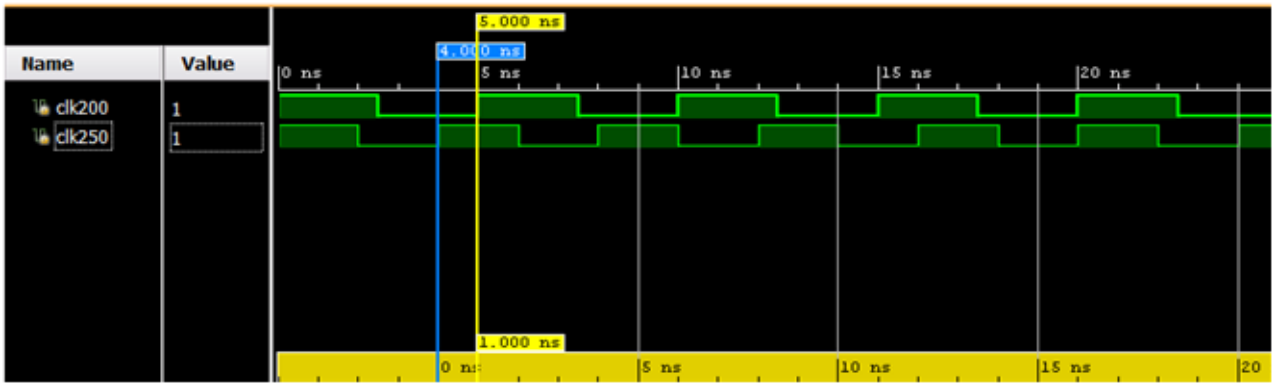


图 5-8 : 从 250 MHz 到 200 MHz 的时钟域

使用 Report_Clock_Networks 去耦主时钟和生成时钟

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，如果是这样，利用单个约束即可将主时钟去耦，并去耦它们的生成时钟。在 report_clock_networks 中使用主时钟作为指南，对每个时钟组及其相关时钟进行去耦，如图5-9 所示。

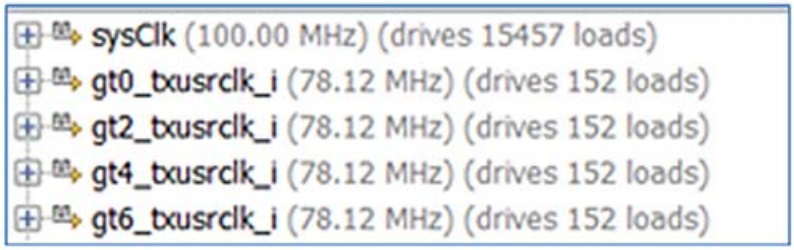


图 5-9 : 报告时钟网络

```
### 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) 设计反复过程中无需 I/O 约束，尤其对于发送和接收寄存器位于 I/O Bank 内部的 I/O 时序路径来说更不需要 I/O 约束。一旦设计及其它约束已稳定并且时序接近收敛，I/O 时序约束就可以被添加回来。

根据RTL设计师的建议，必须对时序例外加以限制，而且不能用来隐藏真正的时序问题。这个时候，时钟之间的伪路径或时钟组必须已经过检查并最终确定。

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。使用 Tcl 命令行流程的 Tcl 用户可以在每个实现步骤之后轻松地将 `report_timing_summary` 整合到他们的构建脚本中。 IDE 用户可利用简单的 tcl.post 脚本在每个步骤后运行 `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，还可采取面向各条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。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
- 添加时序例外，例如多周期路径 （如果对于硬件中的功能来说是安全适用的）。

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。

通过最小延迟分析来估计保持修复 (hold fixing)。如果违规程度较大需要对时钟树进行修改。如果违规程度较低，可以接受而且还有可能被布线器修复。

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）
- 目标距离较远的网络
- 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。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)，应遵循以下所述的分析步骤。

确定时序违规的根源

基于时序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。

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 = 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

为进行保持，必须从最严重的开始检查所有违规。

有几个因素会影响设置与保持时间时序裕量。当时间设置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 （设置/恢复）	=	设置路径要求 – 数据路径延迟 （最大量） + 时钟偏差 – 时钟不确定性 – 设置/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。

有关设置/恢复时间：

- 数据路径延迟：将设置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 + 设置/恢复时间：这种情况与前面描述过的数据路径延迟相同，但是这可能意味着设置/恢复时间比那些寻常明显的原因更容易引起违规。
- 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皮秒，那么这个偏差是个引起违规的较大原因，必须检查时钟拓扑。
- 时钟不确定性：如果时钟不确定性超过数百皮秒，那么必须检查时钟拓扑与抖动数量，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 500 ps，必须检查时钟拓扑。
- 时钟不确定性：如果时钟不确定性超过数百皮秒，那么必须检查时钟拓扑与抖动数量，以便了解不确定性如此高的原因。
- 保持/移除时间：如果保持/移除时间超过数百皮秒，可查看原始数据手册，以验证这个值在预期范围内。
- 保持路径要求：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设置/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。但是当设计和约束已经过了提升与优化，遗留的未解决违规 （如果有）通常由多种因素共同造成。在这种情况下，必须同时检查所有原因，并确定可提升项。

有多种方式可执行设计分析。如需了解更多详情，敬请访问： 《Vivado Design Suite 用户指南：设计分析和收敛技术》(UG906) 中的链接。

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。

如果造成路径延迟主要是因为：

- 单元延迟在 50% 到 100% 之间

  路径是否可被改短或者使用更快的逻辑单元？参见：检查技术选择。

  如需了解更多详情，敬请访问 《Vivado Design Suite 用户指南：设计分析和收敛技术》(UG906) 中的链接。

- **布线延迟在 50% 到 100% 之间**

  路径是否受到保持修复影响？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终点是否使用平衡时钟树？查看偏差值。
  - 否–见下面的拥塞内容

  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。

- 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。尝试手动移动驱动器或负载，并对相同路径重新运行时序分析，以查看时序裕量是否得到提升而且没有对其它路径造成不良影响。在针对多个网络实施相同操作之后，应建立布局规划约束，以确保下次运行此实现工具时可使用类似的布局解决方案。
  - 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。
- 否–设计展开过度。研究布局规划方案，以便根据设计特定部分与 I/O 组件 （如果有）或其它特定定位点的连接情况来识别设计中哪些部分必须保持在特定区域。详见布局规划部分。

如需了解更多详情，敬请访问 《Vivado Design Suite 用户指南：设计分析和收敛技术》(UG906) 中的链接。

时钟偏差与不确定性

赛灵思 FPGA 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、短传播延迟和极低的 Skew。时钟 Skew 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： 使用 Tcl 控制台 (report_clock_utilization) 运行时钟资源利用率报告功能以生成时序报告。时钟部分具体包含全局、区域和局部时钟资源的时钟 Skew。验证时钟网络不包含过度时钟 Skew。

```
Details of Global Clocks
---------------------------------------------------------------------------
Num Loads
Index BUFG cell Net Name BELs Sites Locked MaxDelay (ns) Skew (ns)
---------------------------------------------------------------------------
1 clkgen/clkf_buf clkgen/clkfbout_buf 1 1 no 1.51 0.156
```

高性能时钟域 （超过 300 MHz）的时钟偏差会影响性能。时钟偏差不应超过周期的 15%。在 300 MHz 的情况下，单个时钟域中的最大偏差为 500 ps。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)
```

如果您怀疑高时钟偏差，则可在 Vivado IDE 中对该路径执行时序分析，并创建原理图来研究时钟拓扑。

调试具有高时钟偏差的时序报告

首先必须了解源时钟和目标时钟以及它们的关系：

- 当源时钟与目标时钟同步
- 当源时钟与目标时钟同步

当源时钟与目标时钟同步

当源时钟和目标时钟相同，或者源于同一个主时钟 （同步时钟），该工具使用时钟路径上的共用节点来确定时钟偏差。所有同步路径都包含一个共用节点。当根据时序报告分析时钟路径时，由于报告结果是偏差计算的总和，因此可能难以确定共用节点在器件视图或者在原理图中的准确位置。

有些同步时钟，公共节点位于一个时钟修改模块之前，比如在一个 MMCM 或一个 PPL 之前，因此偏差也有可能很高。赛灵思建议避免这样的同步时序路径，因为它们可能会在满足时序要求方面造成困难。相反，您必须将其视为同步路径，添加时序例外并实现跨越整个电路的同步时钟域。

当源时钟与目标时钟同步

当源时钟和目标时钟不同，并且它们源自于不同的主时钟时，在时钟路径上没有公共节点。偏差是与从主时钟源 （通常是输入端口）到路径的连续单元之间的整个时钟树的延迟差异相对应的。根据时钟拓扑和布局，偏差可能非常大，大到无法满足时序要求。您必须检查同步时钟之间的所有时序路径，并且添加时序例外。您可以通过添加

set_clock_groups 或者 set_false_path constraints，将时序例外添加到“完全忽略时序分析”，也可以通过添加 set_max_delay -datapath_only 约束，将时序例外添加到“仅忽略时钟偏差与时钟不确定性”。

提示： 分析时钟路径的最佳方法是使用 Vivado IDE 中的原理图查看器，并用时序报告进行交叉探测。

导致高时钟偏差的原因

高时钟偏差可能由如下原因导致：

- 时钟信号来自门控逻辑源
- 串联的 BUFG 组件驱动同步元件
- BUFG 驱动同步元件
- IBUFG 驱动多个 MMCM （相关时钟）
- BUFG 驱动寄存器元件和 MMCM （相关时钟）
- BUFR/BUFIO/BUFH 驱动多个时钟域中的寄存器元件
- 使用 CLOCK_DEDICATED_ROUTE=FALSE 约束

时钟信号来自门控逻辑源

不建议使用这种方法，因为会导致过度的时钟偏差。由于门控逻辑驱动器缓冲器不直接接入全局时钟线路，因此它使用的是局部结构布线资源。尽管有些情况下门控逻辑可连接至 BUFG，但还是会导致严重的布线延迟。查看 report_clock_utilization 结果，了解过度的时钟 Skew。

当检验时钟报告时，可能存在由布局布线算法自动插入的额外高 Skew 时钟。这是防止未使用的时钟生成器中芯片长期处于亚稳态的常用做法。时钟在低频 (Hz) 范围内运行，占用资源极少，而且不影响设计性能。

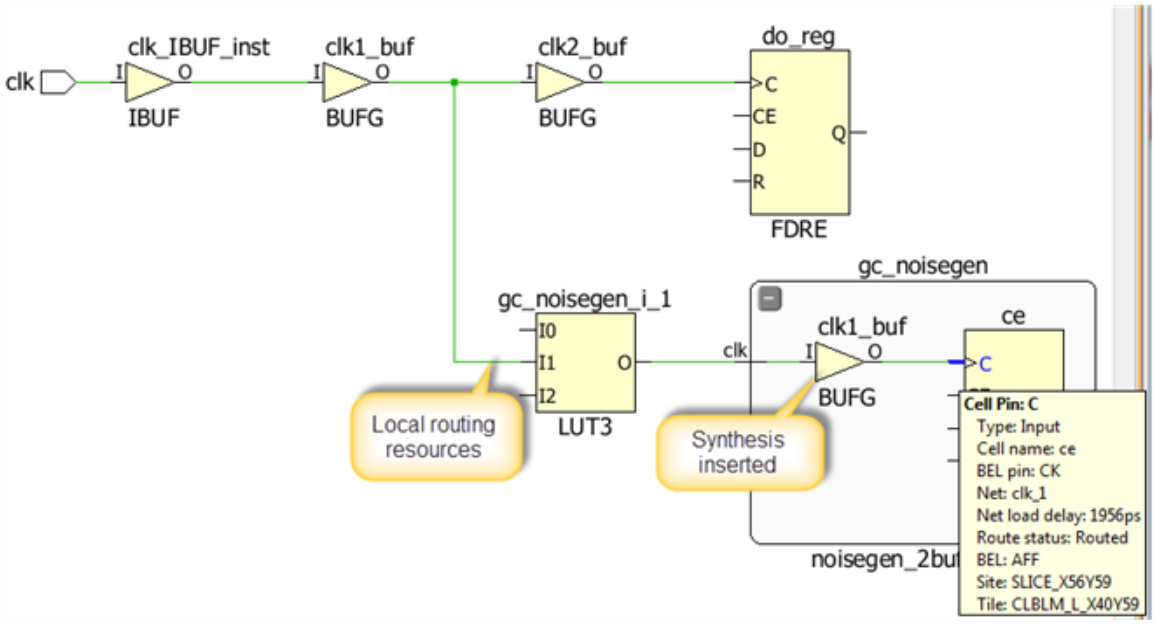


图 5-10 : 时钟网络局部布线引起的歪斜

在 图5-10 中，将第一个 BUFG (clk1_buf)用在 LUT3 中，以便创建门控时钟条件。不建议采用这种做法。为满足这一要求，需使用较慢的局部布线连接。模块 gc_noisegen 中的第二个 BUFG 由综合算法自动插入。

串联的 BUFG 组件驱动同步元件

当添加综合 IP 网表时，应验证综合工具插入的全局时钟缓冲器数是否正确。导入黑盒 IP 时的一个常见的错误是当它检测到连接某个单元 CLK 端口的信号时，综合器会自动插入一个 BUFG。如果下游黑盒 IP 包含一个全局时钟缓冲器，那么这两个 BUFG 组件会加大时钟偏差程度。

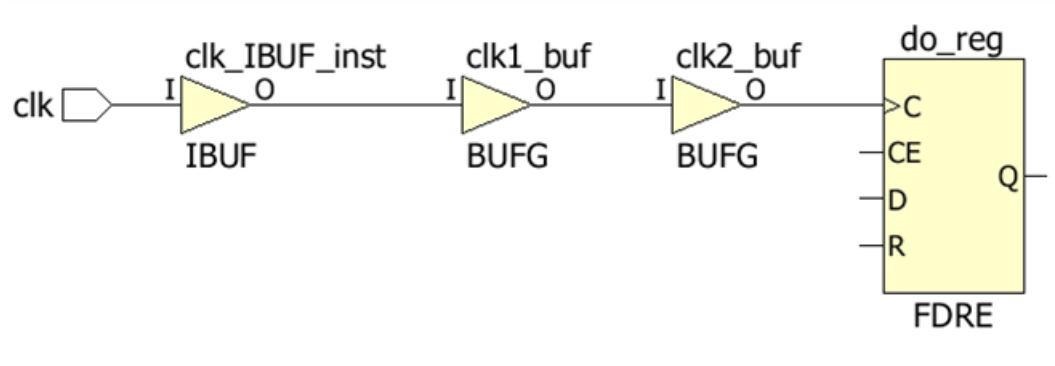


图 5-11 : 级联 BUFG 产生的偏差

在以上实例中，寄存器时钟引脚处的时钟网络延迟为 2.362 ns。如果 BUFG 不由 MMCM 驱动，则没有需要补偿的 PVT 和架构 Skew。

提示： 如果有额外的 MMCM 可用，可以用来减少时钟偏差。

BUFG 驱动同步元件

每个时钟域均包含相同的时钟布线。源时钟引脚和目标时钟引脚在时钟树上的相对位置决定了时钟偏差的差异。如果来自共同节点的源时钟和目标时钟延迟相同，那么时钟偏差将会达到最小。

如果源时钟和目标时钟位于不同时钟域或不同 SLR 中，那么时钟偏差经常会高于正常值。使源时钟和目标时钟保持在一个时钟域内有助于最大限度地减少时钟偏差。 AREA GROUPS 或 PBLOCKS 可强制将源时钟和目标时钟元件放在相同时钟域内。

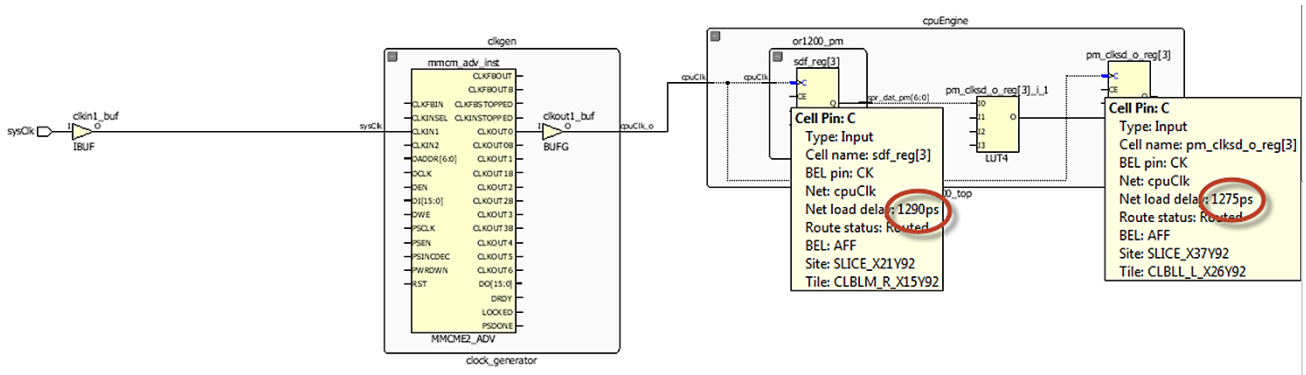


图 5-12 : 源时钟和目标时钟位于相同时钟域内实现的低偏差

在 图5-12中，源寄存器时钟引脚处的时钟网络延迟为 1.290 ns，目标寄存器时钟引脚处的时钟网络延迟则为 1.275 ns。这样造成的时钟偏差超过工艺误差 (PRV) 仅 15 ps。该时钟网络所有目标位置的最大偏差为 0.287 ps。

```
Num Loads
Index BUFG cell Net Name BELs Sites Locked MaxDelay (ns) Skew (ns)
```

```
-----------------------------------------------------------------------------
10 clkgen/clkout1_buf clkgen/cpuClk_o 3297 1298 no 1.37 0.287
```

IBUFG 驱动具有多个输出的单个 MMCM （相关时钟）

将周期约束定义在驱动器引脚或树根端口上。如果时钟信号驱动一个 MMCM 生成多个通用输出频率，那么每个相关联时钟的偏差对输出 BUFG 是相同的。源时钟和目标时钟可能位于不同时钟域。如果这影响您的时序性能，赛灵思建议您使用 AREA GROUPS 或 PBLOCKS。在上面的实例中，两个时钟域之间的时钟网络偏差为 36 ps。

提示： Clocking Wizard 能根据您的时钟要求提供性能指南 （抖动和相位误差）。

IBUFG 驱动多个 MMCM （相关时钟）

赛灵思建议尽可能使用简化的时钟拓扑。压缩时钟域数量有助于改善性能、资源利用和时序收敛。

提示： 在使用驱动其他 MMCM 和其他寄存器的的 MMCM 输出 BUFG 时一定要小心。额外的 BUFG 会导致更高偏差。

BUFG 驱动寄存器元件和 MMCM （相关时钟）

如果可能，应确保 MMCM CLKIN BUFG 只用于驱动 MMCM。可使用 MMCM 的 CLK0 输出驱动寄存的元件。MMCM 可克服工艺误差 (PVT) 实现时钟稳定性，在这种情况下,BUFG 则做不到。考虑到当今设计的复杂性，时序引擎有可能在下游某个位置检测到设计人员无法检测到的跨域时钟路径。

在 图5-13 中,用输入时钟监控 MMCM 的时钟信号。在时钟源被中断时，通常会这么做。信号 `clockRef` 上的资源数量为最小值。

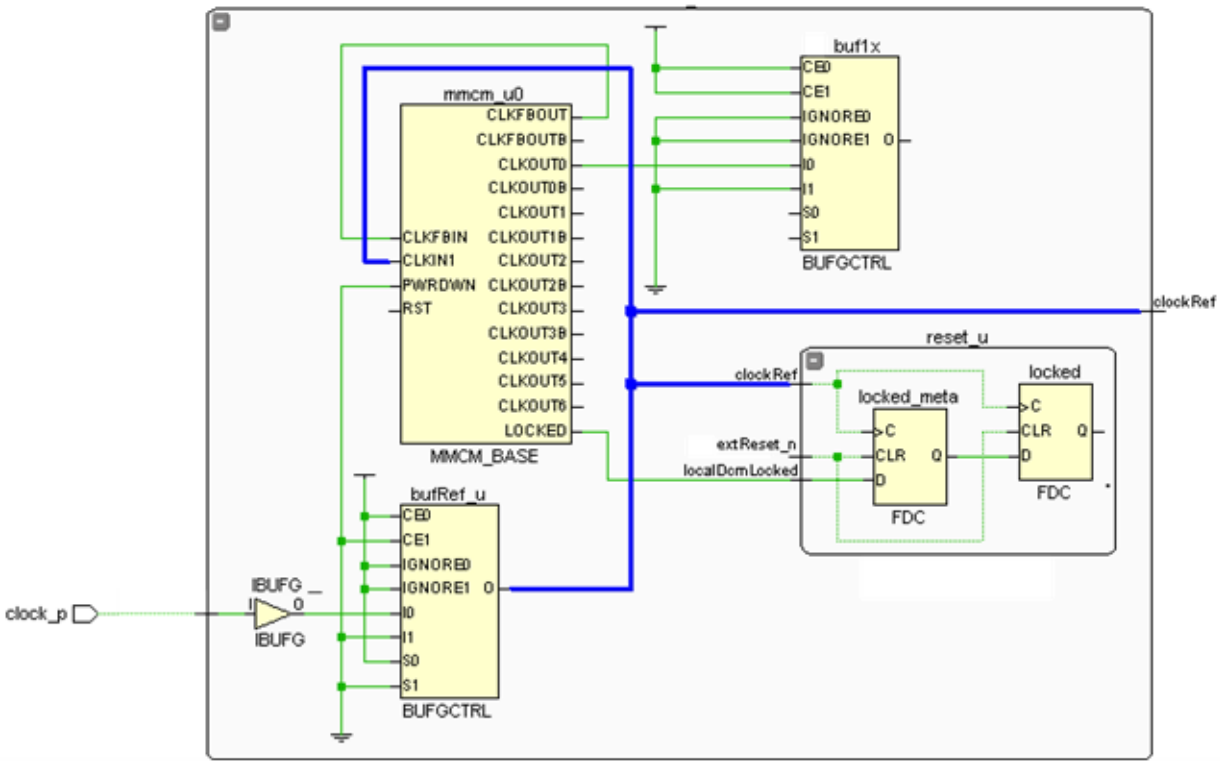


图 5-13 : 由 MMCM 驱动的时钟

BUFR/BUFIO/BUFH 驱动多个时钟域中的寄存器元件

Clock_report_utilization 会报告所有类型的区域时钟缓冲器。核实每个区域时钟的时钟偏差是合理的(<< 1 ns)。在下面的实例中， BUFR 时钟偏差非常高，并显示目标元件违反了时钟规则。（BUFR 只能驱动它所在区域的资源）。

```
Details of Regional Clocks
-------------------------------------------------------------------------------
Num Loads
Index BUFR cell Net Name BELs Sites Locked MaxDelay (ns) Skew (ns)
-------------------------------------------------------------------------------
1 u0_pcie/txoutclk_i u0_pcie/refclk 1 2 no 0.594 0.055
2 u0_pcie/usrclk1_i1 u0_pcie/pipe_userclk1_in 11 25 no 5.93 5.36
3 u0_pcie/usrclk2_i1 u0_pcie/pipe_userclk2_in 463 160 no 0.728 0.202
4 u0_pcie/pclk_i1 u0_pcie/pipe_bclk_in 557 248 no 0.952 0.396
```

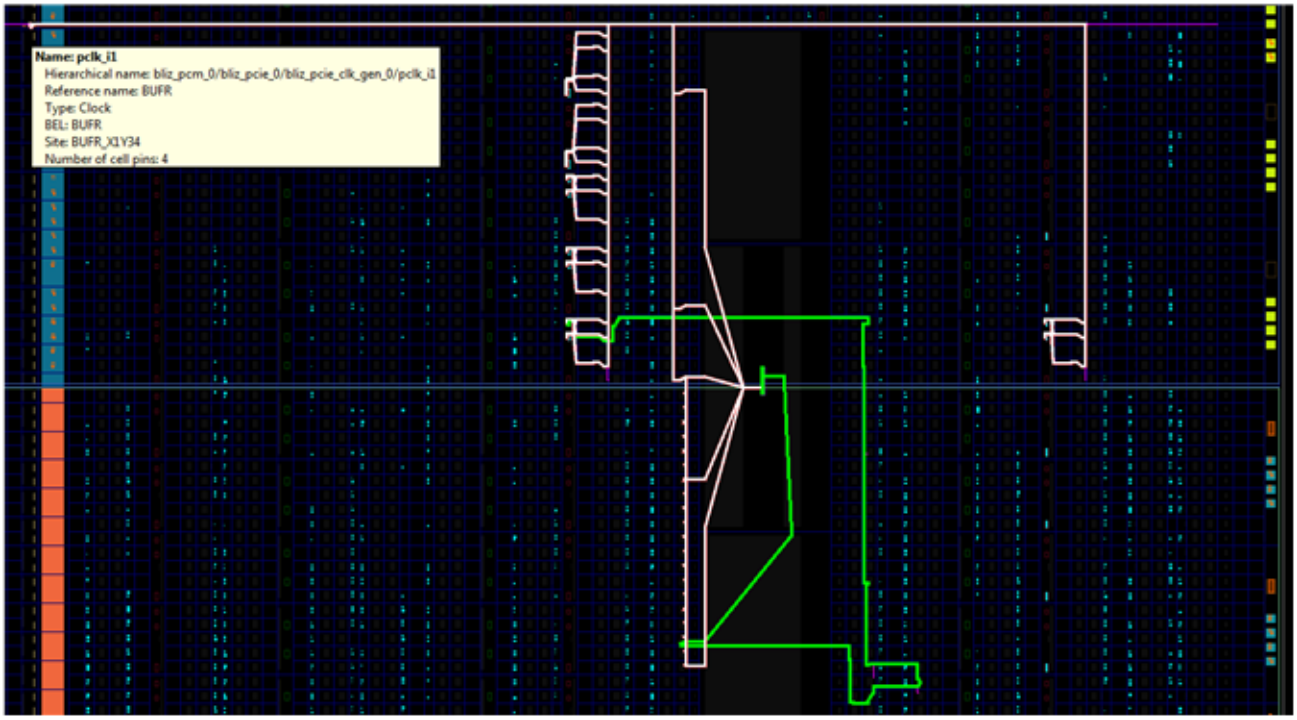


图 5-14 : BUFR 驱动两个区域内的触发器

使用 CLOCK_DEDICATED_ROUTE=FALSE 约束

不要在生产设计中使用 CLOCK_DEDICATED_ROUTE=FALSE 约束。

CLOCK_DEDICATED_ROUTE=FALSE 仅作为处理时钟故障的临时方案，仅仅用在通过布局布线获得设计，以便调试时在器件和原理图查看器中查看时钟拓扑的情况。此类路径具有高时钟偏差，可导致较差的性能或设计功能无法实现。在图5-15 中，右侧采用专用时钟布线，而左侧的时钟则禁用专用布线。

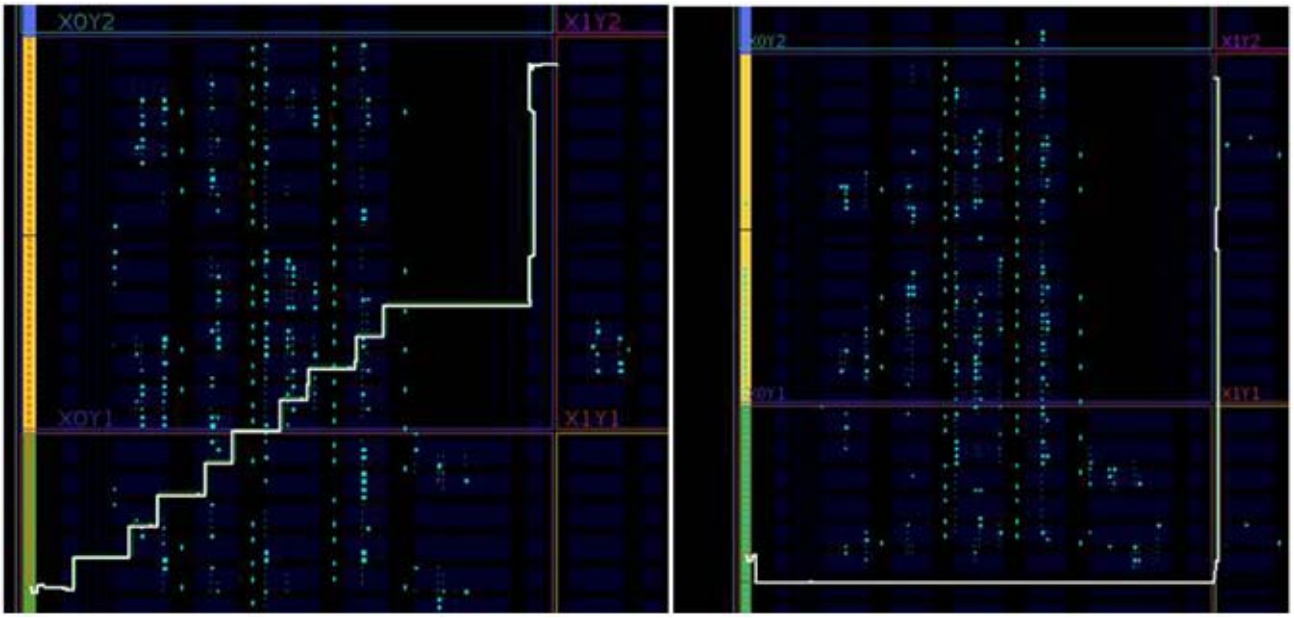


图 5-15 : 时钟专用布线的使用

造成高度不确定的原因

不确定性是指因特定用户外部时钟不确定性、抖动或占空比失真引起的不确定性的总量 （相对于理想时钟而言）。MMCM 和 PLL 等时钟模块会导致时钟不确定性。

Clocking Wizard 可为特定器件提供准确的不确定性数据。 Clocking Wizard 还能生成用于比较不同拓扑结构的各种 MMCM 时钟配置。

使用尚未移植到新器件架构的原有代码为旧的 FPGA 架构创建时钟拓扑，是常见的事。赛灵思建议使用目标器件重新创建时钟环节，以计算并验证系统性能参数和 DRC 规则。

MMCM

MMCM 在重新生成所需时钟时可过滤输入时钟不确定性。如果使用多个相关时钟， MMCM 会产生系统抖动、离散抖动和相位误差等一些时钟不确定性。

设计中的相位对齐不会影响系统性能，如图5-16 所示，源于某个 MMCM 的同一输出为逻辑计时，而设计不会受到影响。

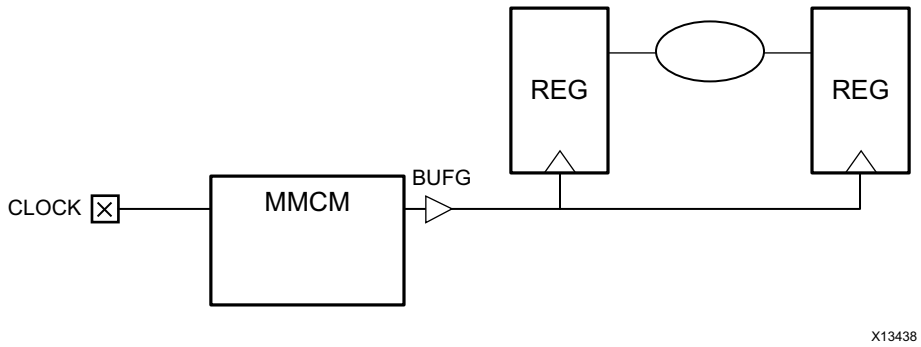


图 5-16 : MMCM 不受相位误差的影响

MMCM 和 I/O 时序

如果设计中 MMCM 的输入与输出之间的相位对齐比较重要，那么对此类设计进行检查以确保所有时序约束仍然满足（即set_input_delay 和 set_output_delay)。参见：图5-17。

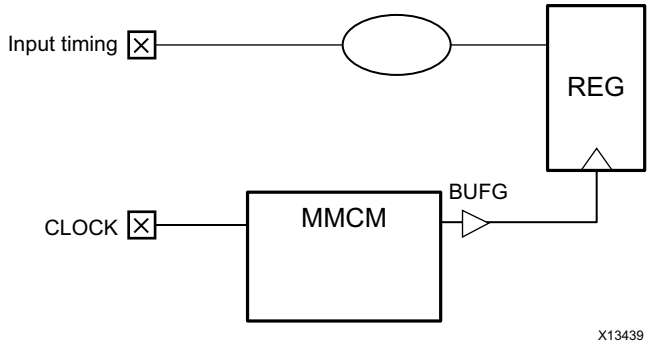


图 5-17 : MMCM 的相位对齐可能影响时序

MMCM 频率综合

当针对频率综合配置 MMCM 时，目标频率可能具有若干 M （乘法器）和 D （除法器）值。为最大限度降低时钟不确定性，应使用能生成更高 VCO 频率的值，记住不要超出器件的最大 MMCMVCO 频率开关特性。如果您正在将设计从较老的技术中移植出来，一定要修改 M 和 D 值，以便为当前技术提供最高 VCO 频率。

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。有两种方案，但只有一种方案 (MMCM_2) 产生的抖动较小，时钟不确定性也很低。

表 5-6 : 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

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。

检查技术选择

必须要知道设计与综合的选择如何对设计的整体时序、资源利用和功耗产生影响。通常可采用众多不同类型的资源来实现同一逻辑功能，而且资源的选择会对结果产生明显影响。例如，采用分布式 RAM 实现的 RAM 与采用块状 RAM 实现的 RAM 在性能方面截然不同。若在设计中注重技术细节，就可以做出很好的权衡以提升结果质量。

逻辑架构由可配置的模块构建而成，每个模块共享相同的控制信号。最小的模块单元为 Slice 或 CLB （可配置逻辑块），具体视架构而定。下面将对 CLB 结构逻辑进行探讨，但考虑到特定技术时除外，例如采用 Slice 的赛灵思系列 FPGA 器件。

除了时钟外，时序原语还需要复位、置位和时钟使能等控制信号。由于很多资源共享相同的控制信号，因此会限制控制信号的使用。控制信号使用效率低下会导致器件资源利用率低下和逻辑封装使用效率低下，进而导致其它问题，例如布线拥塞以及 Slice 和 CLB 的过度使用。

本节概括介绍了组合逻辑与时序逻辑资源的选择以及控制信号实现的影响。尽管给出的实例和数据只是基于赛灵思 7 系列 FPGA 器件技术的一般特性，但类似的分析也可以运用于 UltraScale™ 架构器件和未来技术。如需了解有关时序参数的详情，敬请参见器件数据手册中的 AC 开关特性。

组合逻辑：LUT 引脚延迟

并不是所有通过 LUT 的路径都具有相同的延迟。在时序报告中，每个输入引脚延迟看上去都相同，但通向每个输出的网络延迟中都整合了一些额外的线路延迟。这是由相关互联的物理实现方案引起的。尽管这些工具只是尝试去实现逻辑，以便关键信号能使用最快的输入，但了解这方面情况还是有助于减少复杂时序故障，并有助于分析次优化 LOCK_PINS 约束。

LUT 可被描述为逻辑引脚或物理引脚。逻辑引脚被命名为 I0、I1、I2、I3、I4 和 I5，并反映网表连接。这些引脚在时序报告中也有命名。物理引脚的命名为 A1、A2、A3、A4、A5 和 A6，也可能用不同字母 B、C 或 D，这要取决于所用的 BEL。物理引脚代表实际器件引脚，而且一般只有当分析器件级的物理实现时才能看到。

逻辑引脚通过由布局器、布线器或根据 LOCK_PINS 属性选择的映射方案被映射到物理引脚。通常，物理 A6 LUT 输入是最快的路径，其次是 A5、 A4 等，最慢的是 A1。 A6 路径一般比通过 A1 的最慢路径快几百皮秒。

对于每个 LUT，可在布局后看到逻辑引脚到物理引脚之间的映射。例如， LUT6 具有如下引脚映射（如单元属性的 Cell Pin 视图中所示的默认引脚映射），或者使用 `get_site_pins` 命令实现的引脚映射。

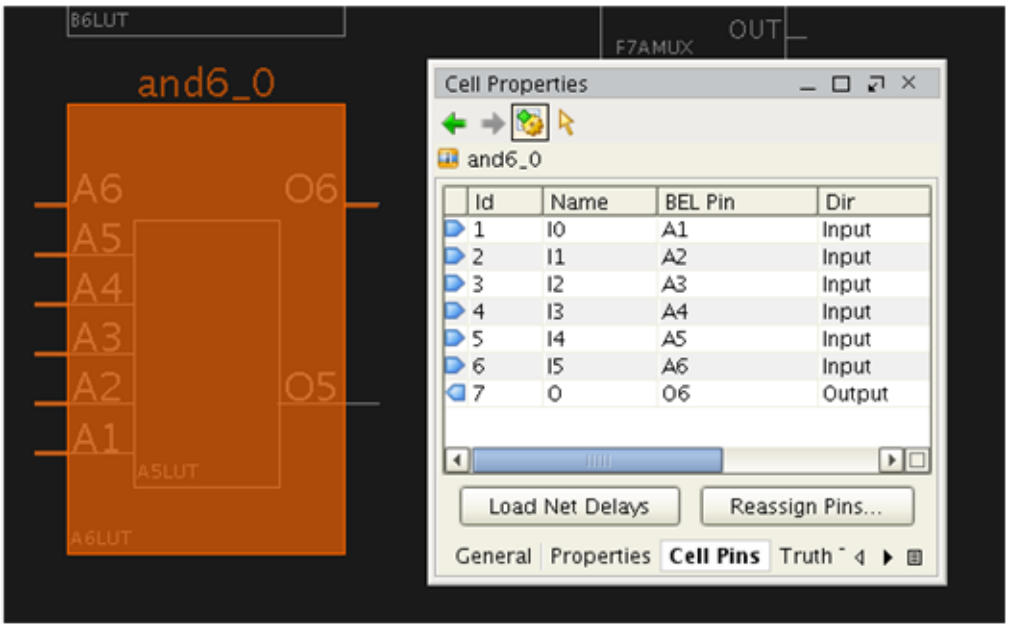


图 5-18 : LUT 逻辑与物理引脚

BEL Pin 列给出了从逻辑引脚映射的物理引脚，如下所示：

```
I0 maps to A1
I1 maps to A2
.
.
.
I5 maps to A6
```

在实现阶段，布局、物理优化和布线过程可交换LUT输入引脚以优化关键路径时序。将关键时序逻辑引脚移动到更快的物理引脚上，例如 A6，而将较慢的逻辑引脚移动到较慢的物理引脚上。对于穿越多个 LUT 的关键路径来说，使用最快物理引脚和使用最慢物理引脚的差别会非常明显。引脚交换可通过设置单元上的 LOCK_PINS 属性并定义其显式映射来覆盖。

组合逻辑：组合LUT

赛灵思 7 系列 FPGA 器件的逻辑 LUT 在设计上比较灵活，可支持一个以上的 6 输入函数。它有 O6 和 O5 两个输出，允许将两个逻辑 LUT 函数进行组合以适应单个资源。内部逻辑表达式中包含两个共享共用输入的 5 输入 LUT。一个 LUT 用来生成 O5 输出，同时 O6 将 LUT5 函数与第六个输入 A6 组合。

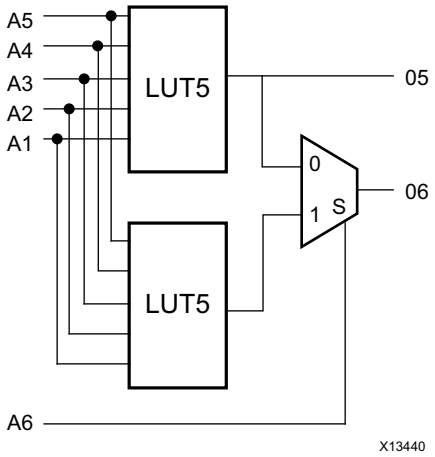


图 5-19 : 来自相同 LUT6 的多个输出

以下是 LUT 组合的实例：

- 完全不相关的 LUT2 和 LUT3
- 两个 LUT3，至少有一个共用输入
- 两个 LUT4，至少有两个共用输入
- 两个 LUT5，全部共用输入
- 一个 LUT5 和 LUT6，其中 LUT6 与 LUT5 组合，生成 O5 输出和 A6 输入。

由于最快的 A6 输入专门用于 O6 多路复用，因此一定要知道当 LUT 被组合时其使用会受到限制。 A6 引脚或者保持未使用状态，或者，如果组合 LUT 包含一个 6 输入函数， A6 引脚必须用于非共用输入。

时序逻辑：寄存器

寄存器可被映射到器件数种资源中的一种资源：

- CLB 寄存器
- 作为 SRL 的 CLB LUTRAM
- ILOGI
- OLOGIC
- DSP 和块状 RAM （如果寄存器接近于算法或存储器功能）

并非所有的时序逻辑都能灵活映射到以上的任何资源中。但是，特定时序逻辑有时可以映射到一种以上的资源。如果有选择的余地，您可能会选择实现特定时序逻辑的最快资源。

由 LUT 驱动的 CLB 寄存器的建立时间很短，因此这部分路径延迟通常可以忽略不计。对于位于 ILOGIC 中的寄存器也是如此。当数据通过绕开 LUT 的 X 输入进入 CLB 时，建立时间会稍微延长。如果数据通过 FMUX 或进位逻辑，建立时间会突然增加至几百皮秒。对于映射到 OLOGIC 或 LUTRAM 的寄存器来说也如此。在此类情况下，建立时间会成为关键路径的重要影响因素。

表 5-7 : 位于不同资源内的寄存器的相对建立要求

FF 位置	
ILOGI	速度更快
LUT 驱动的 CLB FF	
X 输入驱动的 CLB FF	
LUTRAM(用作 SRL)	
MUXFX 或 CARRY 驱动的 CLB FF	
OLOGIC	速度较慢

CLB 寄存器的时钟到输出延迟通常极短，对赛灵思 7 系列速度等级最高的 FPGA 器件而言，通常为 250 ps 左右。 CLB 寄存器有两个 BEL 位置。较低的 Q 位置要比 MUX 位置 （需在输出前经过多路复用器）稍微快一点。

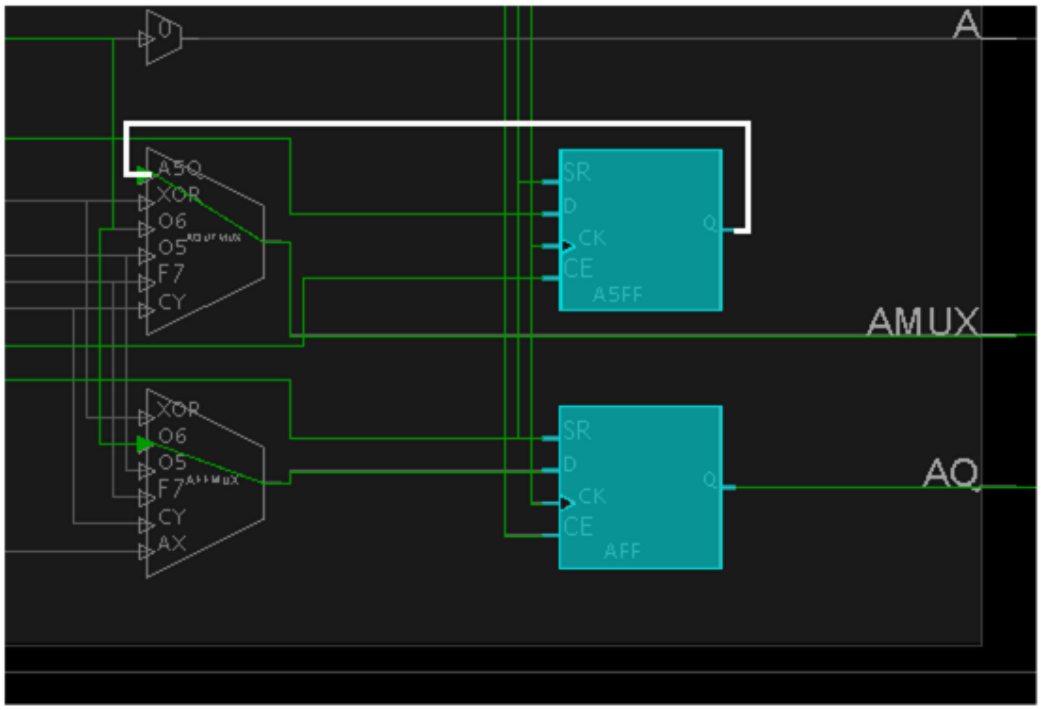


图 5-20 : 较低的寄存器具有更低的延迟

对于映射到 LUTRAM （用作 SRL）的寄存器，其时钟到输出延迟明显较低，处于纳秒级。如果 SRL 输出驱动关键路径，则有必要把最后的寄存器级从 LUTRAM 移到其 CLB 对寄存器，以减少时钟到输出的延迟。

ILOGIC 和 OLOGIC 的时钟到输出延迟要比 CLB 寄存器时钟到输出的延迟低。

表 5-8 : 不同资源中的时钟到输出延迟

FF 位置	
Q BEL 中的 CLB FF	速度更快
MUX BEL 中的 CLB FF	
ILOGI	
OLOGIC	
LUTRAM(用作 SRL)	
DSP	
BRAM	速度较慢

存储器

赛灵思器件中的存储器可用块状 RAM 或分布式 RAM 来实现。这两种 RAM 都能通过综合来调用；可使用 IP 目录生成；或者都能被例化为 UNISIM。使用这两种存储器的任意一种均可实现大部分单端口与双端口 RAM 与 ROM 功能。功能要求通常决定所需的 RAM 类型。例如，异步读取路径需要分布式 RAM。有时需求会从一种类型转向另一种。容量很大的 RAM 通常需要多个块状 RAM。窄数据宽度一般在分布式 RAM 中更高效。不论选择哪一种，您都必须了解每种类型的设计影响。

块状 RAM

块状 RAM 是用于 RAM、ROM 与 FIFO 的专用硬件资源。这些块状 RAM 被划分为跨越器件高度的列，各个列在 CLB 之间的分布相当均匀。

由于块状 RAM 是专用模块，因此其更加适合高容量。此外，与相同容量的分布式 RAM 相比， Block RAM 的功耗更低。但是，进出块状 RAM 列的延迟通常比较高。图5-21 给出了两种布线路径。

图 5-21 : 实例：进出块状 RAM 的布线

块状 RAM 的读取访问相对较慢：时钟到输出延迟大约为 1.5 至 2 ns；还要再花 400-500 ps 的布线延迟来获取 CLB 逻辑。

块状 RAM 具有一个可选的数据输出寄存器，可将时钟到输出延迟减少一半以上。建立保持时间也会严重影响高速路径。二者范围均在 500 至 700 ps 之间，当使用 READ_FIRST 模式时，两者均可减少一半以上。

提示： 当手动放置块状 RAM 时，应使 RAM 共享各列中相同的地址线。各列均可访问快速、专用的地址线布线，以便级联块状 RAM，从而创建更大容量的 RAM。

分布式 RAM

分布式 RAM 可用 CLB 逻辑 （LUTRAM、寄存器、 LUT、 MUX）来实现。由于分布式 RAM 是用 CLB 逻辑来实现的，因此更适合较小的容量。与块状 RAM 相比，更大尺寸的分布式 RAM 会消耗更多 CLB 资源和功耗。但是，较小的尺寸可实现非常好的性能，因为进出 RAM 的布线延迟会减少很多。

LUTRAM 是用于分布式 RAM 存储的 LUT。 LUTRAM 具有与块状 RAM 类似的建立时间，但保持时间大约是后者的一半。读取访问时间也大约是后者的一半；但退出 CLB 需要更多延迟，布线的额定时间为 200-400 ps，而对于需要经过多路复用逻辑的传播来说则可能上升到纳秒级。图5-22 给出了从分布式 RAM 读取的典型最快路径。

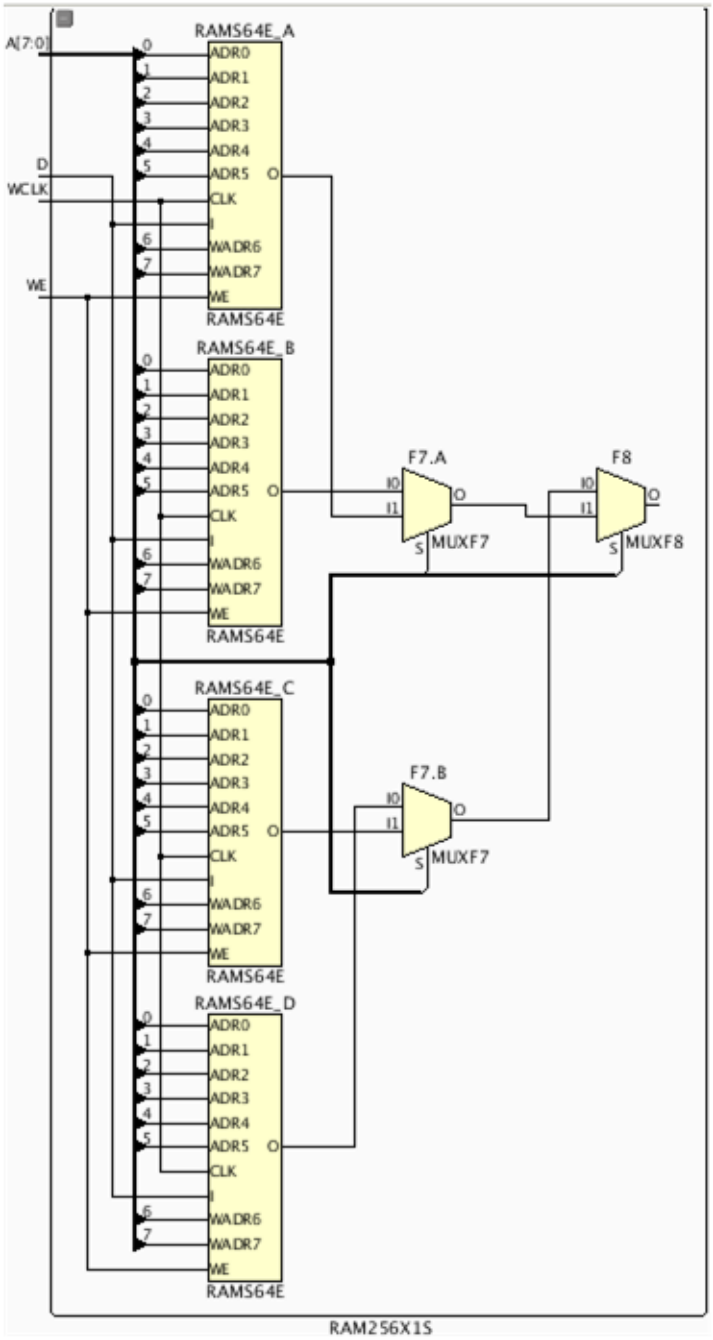


图 5-22 : 分布式 RAM 的读取延迟

Block RAM 与分布式 RAM 之间的对比

下面的实例主要介绍块状 RAM 与分布式 RAM 之间的区别。采用具有 Virtex®-7 -2 速度等级 IP 默认设置的 IP 目录来创建两种不同尺寸的单端口 RAM。布线后的结果反映的是理想条件。实际性能可能因包含设计的周围逻辑变化而有所不同。

表 5-9 : 8kx32 RAM 的实现对比

	块状 RAM	分布式 RAM
最大频率	500 MHz 以上	250 MHz
占位面积	8 RAMB36、 18 CLB	2043 个 CLB
功耗	370 mW	440 mW

结果表明对于这种高度和宽度来说，块状 RAM 实现方案的总体效果更好。分布式 RAM 关键路径如图5-23 所示。

注释： 增加流水线级数以平衡整个解码逻辑的延迟，这样可增大最大频率，但最大频率会受限于通过 RAM 单元的延迟。

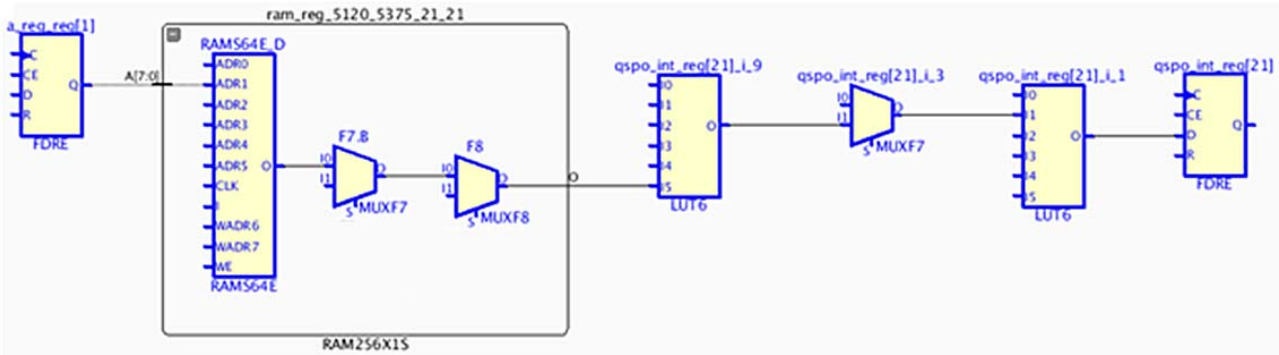


图 5-23 : 贯穿分布式 RAM 的关键路径

表 5-10 : 128x4 RAM 的实现对比

	Block RAM	分布式 RAM
最大频率	约 400 MHz	500 MHz 以上
占位面积	1 个 RAMB18， 3 个 Slice	4 个 Slice
功耗	260 mW	260 mW

对于这种相对较小的尺寸来说，采用分布式 RAM 实现方案具有快速、简捷以及占用更少布线资源的优势。

DSP48E1 模块

在大多数情况下，综合与 IP 目录可决定算法功能的最佳实现。大部分高级功能 （尤其那些依靠宽位、高速乘法的功能）最适合在 DSP48E1 模块中实现，该模块具有用来分担 CLB 任务的专用硬件乘法器和 ALU。DSP48E1 不仅内部高度优化，而且在沿布置该模块的 DSP 列的位置有专用的高速布线。这使得多个 DSP48E1 能够实现更宽的乘法器和级联电路，例如流水线 FIR 滤波器，而且运行频率均在 500MHz 以上。

CLB 进位逻辑通常更适合于特定电路，例如通过常数乘法器和小宽度乘法器来实现的乘法运算。当特定类型的资源被过度使用或高度使用时，功能可从一种类型转移到另一种。当 DSP 模块的运行不足时，可从基于 DSP48E1 的功能转为 CLB 逻辑。

同样地，当 CLB 被过度使用时，众多基于 CLB 逻辑的功能也可被转移到 DSP48E1。后者对解决拥塞区域有帮助。DSP48E1 不仅可实现乘法器和乘法累加功能，还可实现加减器、计数器乃至宽泛的并行逻辑门。

DSP48E1 模块以流水线方式实现，包含输入和输出寄存器以及位于乘法器和 ALU 之间的中间寄存器，如图5-24 所示。

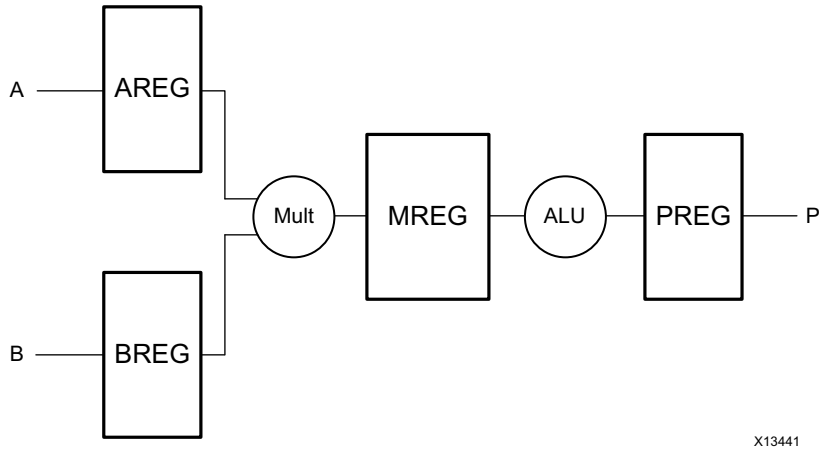


图 5-24 : DSP48 中可用的流水线寄存器

必须使用所有寄存器级数，以实现对应于三个周期延迟的最高性能。下面的实例说明了不同数量的寄存器级数如何影响时序。使用中档的 -2 速度等级，在 Virtex-7 器件上实现具有 32 位输出的 16x16 有符号乘法器。

表 5-11 : DSP48 寄存器对最大频率 (FMax) 的影响

时延	AREG/BREG	MREG	PREG	设置路径	时钟到输出路径	最大频率
0	无	无	无	不可用	不可用	250 MHz
1	无	无	有	2.65 设置 + 400ps 布线	350 ps clk->out + 770 ps 布线	300 MHz
2	有	无	有	260 设置 + 760ps 布线	350 ps clk->out + 700 ps 布线	360 MHz
3	有	有	有	260 设置 + 760ps 布线	350 ps clk->out + 700 ps 布线	500 MHz 以上

有一些出入 DSP 模块的布线延迟。

- 使用一个寄存器级数 （PREG 级）时，到达寄存器输入的延迟比较大，如果包括从 CLB 逻辑到 DSP 列的布线，超过 3 ns。
- 使用*两个*寄存器级数 （输入寄存器和输出寄存器）时，最大频率受限于内部的寄存器到寄存器路径。
- 使用*三个*寄存器级数时，DSP 模块内部运行频率在 500 MHz 以上。如果连接 DSP 模块的逻辑位置摆放较好，包含系统可达到 500 MHz 的最大频率。

如果必须将三级乘法器移至 CLB 逻辑，那么等效实现方案可达到大约 440 MHz，并需要约 143 个 Slice，包括用来累加部分乘积的 15 个进位链。进位链的高度为有 5 至 6 个 Slice，而且必须放在垂直相邻的 CLB 中。布局器必须能够将这些高的宏指令整合到已有的 CLB 逻辑中。参见：图5-25。

图 5-25 : 在 CLB 中实现的 DSP48

需要一个附加的流水线级数来实现相似性能。

控制信号与控制集

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。以下各节定义了控制信号与控制集。

控制信号

表 5-12 列出了位于 CLB 资源上的库原语控制信号。

表 5-12 : 控制信号

时钟	使能	复位
• 时钟与门电路 （用于锁存器）	• 时钟使能 • 写使能 • 门使能 （用于锁存器）	• 逻辑 0 。 复位 （同步） 。 清除 （异步）
		• 逻辑 1 。 置位 （同步） 。 预设 （异步）

控制集

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。这包括缺少使能和设置/复位的情况。例如，由相同时钟计时的两个单元，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，那么这两个单元实际上具有不同的控制集。

控制集的数量将影响可以放入 Slice 中的寄存器数量，因为所有八个寄存器共享相同的时钟、复位和时钟使能信号。这意味着如果有两个寄存器具有不同的时钟、复位或使能信号 （包括不含某一时钟），那么它们将无法共享相同的 Slice。这对使用和布局都会造成负面影响，反过来，这也会给性能和功耗带来不利影响。

在赛灵思 7 系列 FPGA Slice 中 （每个 Slice 有八个寄存器，是 CLB 的一半），所有寄存器共享相同控制信号，因此共享相同的控制集。如果控制集中的寄存器数量无法被 8 整除，有些寄存器就必须不用。对于具有若干很低扇出控制信号（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。对于具有很多控制集的设计而言，因为寄存器必须处于不可用状态，因此寄存器的利用率通常比较低。

如需了解有关控制信号的更多信息，敬请参阅：第 4 章中的控制信号与控制集。

提示： 如果您遇到利用率或拥塞等问题，可以使用 `report_control_sets` Tcl 命令查看您的设计中是否在过多控制信号。

如果控制集的数量过多，可使用以下方法减少控制集的数量，尤其着重于一些具有较低扇出的控制集。

- 避免使用那些同时具有异步和同步复位的时序元件。
- 避免对非恒定值进行异步分配。这会导致一些问题：
  - 形成规模更大的电路，两个寄存器、一个锁存器和一个 LUT。
  - 每个序列都有不同控制集，最少占用 3 个 CLB。
  - 如果不进行正确分析，若干个异步时序路径可能会导致更多影响整体设计稳定性的时序威胁。
- 在可能的情况下，使用高电平有效控制信号。
- 避免异步置位/复位。每个异步复位都是都是不可被移动到数据路径的控制信号。与使用同步复位或置位的情况不同，无法通过将异步复位或置位分解到数据路径逻辑这种方式来缓和控制集的增加。这能够为包装和布局带来更大的灵活性。
- 只在需要时使用置位/复位：
  - 数据路径包含很多可自动清理未初始化值的寄存器。
  - 必须复位到已知数值的 I/O、状态机和关键控制信号的寄存器应使用置位和复位。
  - 使用同步复位可以针对信号的有效和无效来评估时序。通常，同步信号的预测性更强，除非是绝对需要异步复位的情况，否则建议使用同步信号。
- 时钟使能有时实际是冗余逻辑，无法通过综合或逻辑优化来减少这些冗余逻辑。
- 使用具有扇出控制的警告进行综合以及逻辑和物理优化。低扇出限值会引入太多不必要的控制集。

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
- 优化反复循环
- 增量编译
- 设计过约束

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定制化之前，应该首先执行预定义的策略。赛灵思不建议针对非 SSI 器件采用 SSI 策略。

如果用默认策略无法满足时序要求，需要手动设定自定义的指令组合。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 TNS 产生影响的关键网络。

```
phys_opt_design -force_replication_on_nets
```

接下来可用任何指令运行 phys_opt_design，以改善设计的整体 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。

增量编译

当参考设计的关键路径不受当前设计变化影响时，增量编译可在为设计保留 QOR 时产生最佳结果。如需了解有关使用增量编译的更多详情，敬请参阅：增量流程。

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量。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。实现这一点的建议方法是针对如下原因采用 set_clock_uncertainty 约束：

- 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重置。

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重置额外的不确定性。

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 clk0 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 clk0 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。

布局规划考虑事项

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来实现最大改善。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人工布局的部分，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。应最大程度减少穿过 Pblock 的网络的数量。

应特别考虑堆叠硅片互联 (SSI) 器件。SSI 器件由多个超级逻辑域 (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 器件的设计的时序和布线。如需了解更多信息，敬请参阅：《Vivado Design Suite 用户指南：设计分析和收敛技术》(UG906) 中的链接。

视频： 如需了解有关使用布局规划技术以解决设计性能问题的更多信息，敬请观看：Vivado Design Suite QuickTake 视频：设计分析与布局规划。

保存布局布线

一旦结果满足时序约束，您可能希望锁定设计的布局及其关键部分的布线方案。这有助于保存电路的性能，为后续工作提供更具预测性的结果。

很容易重用以下内容的布局方案：

- I/O
- 全局时钟资源
- 块状 RAM 宏命令
- DSP 宏命令

重用布局有助于减少网表修订版本与另一个版本之间的结果变化。这些原语通常具有固定的名称，而且布局通常易于维护。

有时候希望保存关键网络或部分关键网络的布线方案，以确保不同运行具有相同时序。可将网络的 `is_route_fixed` 属性设为 1，以保存关键网络的布线方案。使用 Vivado IDE 或 Tcl 命令设置该属性。要在 Vivado Design Suite 中修复网络的布线，应在器件视图中选择该网络，点击右键并从上下文菜单中选择“Fix Routing”。

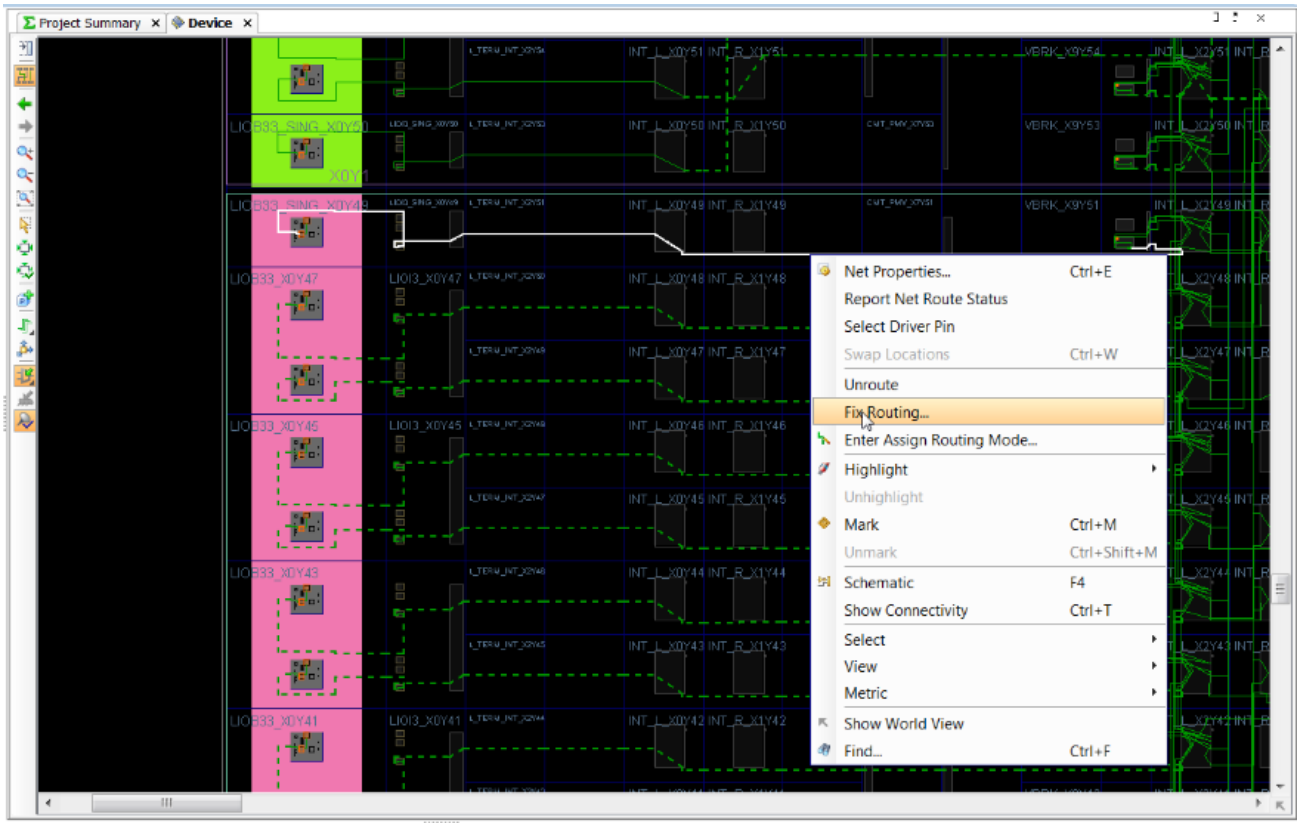


图 5-26 : 保存特定布线

下面的 Tcl 命令实例修复 `$net` 中所有网络的布线：

```
set_property IS_ROUTE_FIXED 1 $net
```

这样可将布线标记为修复，并向内存设计添加一个约束。已修复布线后的网络在“Device”中用虚线显示。

有时候，当一个设计的利用率很高，或有很严格的时序要求，即便设计或者约束中的一个很小的修改都有可能导致布局、布线，或两者均发生巨大变化。保留设计中的相关部分可能在遇到这样的情况时会特别有帮助。

功耗

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，敬请参阅：《Vivado Design Suite 用户指南：实现》(UG904) [参照 22]。

估计 Vivado Design Suite 流程中各阶段的功耗

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。

  如需了解更多信息，敬请参阅：第 3 章：单板和器件规划。

- 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。

除了对最佳和最差门控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，包括干扰。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。

report_power 支持的功耗估计类型

report_power 命令支持两种功耗分析模式。可根据精度要求选择合适的类型。

基于矢量的估计

可在设计开发的各个阶段同步执行仿真，以确认设计能够按预期工作。提供不同的验证技术，用以根据设计开发状态、设计复杂度以及公司政策来选择。

下面几段将重点介绍可获得哪些有用数据，以及与使用这些数据执行功耗分析有关的常见陷阱。实现精确功耗估计的一个重要因素是设计行为必须真实。其应代表数据进入仿真模块的最佳或最差案例情景。在执行确认或验证功能时不一定提供此类信息。

可输入无效的数据以验证系统能否处理这些数据以及在有无效数据或命令输入时系统能否保持稳定。使用这种测试案例执行功耗分析会导致功耗估计不精确，因为设计逻辑没有按照典型系统工作条件下的激励方式进行激励。

系统处理等级

在设计周期的早期，您可能已经在 PCB 上的器件之间或者在 FPGA 应用的不同功能之间建立了处理描述文件。您可从中提取每个功能模块针对特定 I/O 端口和大部分时钟域的预期行为。该信息有助于您填写 XPE 电子数据表。

FPGA 描述等级

当为应用定义 RTL 时，您可以通过行为仿真来验证该功能。这样有助于验证数据流以及针对时钟周期的计算正确性。下面的内容尚无法实现：

- 所使用的准确 FPGA 资源
- 计数

- 配置

可以手动外推资源使用信息，并提取针对 I/O 端口或内部控制信号 （置位、复位、时钟使能）的活动。该信息可用于改善 XPE 电子数据表信息。

您的仿真器应该可以提取节点活动，并以 SAIF 文件的形式将其导出。如果不打算运行实现后仿真，可保存该文件用以在 Vivado Design Suite 设计流程 （例如布局布线之后）中实现更加精确的功耗分析。

FPGA 实现等级

可用能够提取到具有不同功耗信息的结果，在实现流程的不同阶段执行仿真。这个额外信息也可用来改善 XPE 电子数据表以及 Vivado Design Suite 功耗分析。还有可能保存 I/O 端口和具体的模块行为，以便在 Vivado Design Suite 功耗分析中重新使用。

仿真文件

Vivado Design Suite 的 Report Power 将设计数据库中的网与仿真结果网表中的名称进行匹配。功耗的仿真数据被存储在 Switching Activity Interchange Format (SAIF)文件中。

由于 `report_power` 执行网络名称匹配，因此最好从正在执行功耗分析的同一个设计视图 （例如综合后和布线后）中获取仿真结果。

重要提示： 在 Vivado IDE 中“Report Power”对话框的“Input Files”选项卡中设定 SAIF 文件名，以便读取 SAIF 仿真输出文件。或者，使用 `read_saif` Tcl 命令读取 SAIF 仿真输出文件。如需了解如何从 Mentor Graphics ModelSim 仿真器生成用于在 Vivado Design Suite 中进行功耗分析的 SAIF 文件，敬请参阅：答复记录 53544。

无矢量估计

当未提供设计节点活动时 （自己的或仿真结果），可利用无矢量功耗估计算法来预测该活动。

无矢量引擎向所有未定义节点分配初始种子 （默认信号速率和静态概率）。从设计主输入开始，经过各种电路元件将活动传播到内部节点；然后重复这个操作直到达到主输出为止。

该算法可支持设计连接和资源功能性及配置。其试探法甚至可接近网表中任意节点的毛刺速率。在稳定至最终值之前，设计元素会在时钟有效边沿之间多次变换状态，这时就会出现毛刺。

然而无矢量传递引擎没有布线后仿真那么精确，因为布线后仿真具有较长但比较合理的持续时间和真实的激励，因此无矢量传递引擎是精确度与计算效率之间优秀的折衷方案。

精确功耗分析的最佳做法

使用如下内容实现精确功耗分析：

- 精确时钟约束
- 精确 I/O 约束
- 顶层控制信号上的精确信号速率和高电平占空百分比 （%）

精确时钟约束

由于功耗很大程度取决于工作频率，因此必须精确设定时钟频率。

无论进行何种设计，您通常都能够知道一些特定节点的活动，因为它们是由系统规范或FPGA器件的通信接口指定。将该信息提供给工具，这样有助于引导功耗估计算法。

对于驱动多个 FPGA 器件单元的节点来说，该信息尤其有用。

- 置位
- 复位
- 时钟使能
- 时钟信号

无论 FPGA 时钟域由外部提供 （输入端口），还是内部生成，抑或是从外部提供给 PCB （输出端口），通常都能知道所有 FPGA 时钟域的精确频率。

精确 I/O 约束

在知道 FPGA 器件内部和外部数据流的精确协议和格式后，通常就可以针对一些 I/O 组件在工具中设定信号转换速率和/或高电平占空百分比。

例如，有些协议具有 DC 平衡要求 （高电平占空百分比 = 50%），或者您可能知道从存储器接口读写数据的频度。这样您就可以设定选通脉冲和数据信号的数据速率。

由输出端口驱动的电路板和其它外部电容通常是已知的。

在 Tcl 提示信息中输入下面的内容，用以设置所有输出端口上的负载：

```
set_load <value in pF> [all_outputs]
```

顶层控制信号上的精确信号速率和高电平占空百分比 （%）

了解系统和预期功能之后，就可以预测控制信号 （例如置位、复位和时钟使能）的活动。由于这些信号通常可打开和关闭大块设计逻辑，因此提供该活动信息可显著提高功耗估计精度。

如果知道 I/O 接口的数据模式，应设定该活动。除非您在单独的工具 （例如电子数据表）中计算每个电源的总功率，否则应指定输出的终端技术，以便让“Report Power”将 FPGA 器件提供给外部元件的功耗包含在内。

可通过 Tcl 提示信息中的 `set_switching_activity`，或 Vivado IDE “Power Properties”窗口中的高信号速率和静态概率 （%）获得以上所有信号类型的准确信号特性。

工程器件设置

检查“Report Power”对话框“Environment”和“Power Supply”选项卡中不同的用户可编辑选项。确保流程、电压和环境数据预期环境匹配良好。这些设置对总体估计功耗有显著影响。

这些选项卡中的用户可编辑选项为：

- 器件设置
- 设计热设置
- 供电电压设置

器件设置

- 温度等级

  选择合适的器件等级（一般是“Commercial”或“ Industrial”）。根据这个设置，有些器件可可以获得不同的器件静态功耗技术参数。合理设置可以正确显示所选器件的结温限值。

- 流程

  对于最差情况分析，建议的流程设置为 Maximum。尽管默认设置“Typical”的统计测量结果更精确，但将设置改为“Maximum”可将功耗参数改为最差情况值。

在 Tcl 提示信息中，使用以下内容设置温度等级和流程选项：

```
set_operating_conditions -process maximum
set_operating_conditions -grade industrial
```

设计热设置

检查“Report Power”对话框“Environment”选项卡中不同的用户可编辑选项。

- 环境温度(°C)

  对采用 FPGA 设计的机壳的内部最大预期温度进行设定。该部分以及气流和其它热耗散路径（例如散热片）可实现对结温的精确计算，从而实现更加精确的器件静态功耗计算。

  在 Tcl 提示信息中，输入如下内容以设置环境温度选项：

  ```
  set_operating_conditions -ambient_temp 75
  ```

- 气流 (LFM)

  经过芯片的气流以每分钟直线英尺 (LFM) 为单位进行测量。 LFM 的计算方法是用每分钟立方英尺 (CFM) 的风扇出风量除以空气通过的横截面积。

  FPGA 器件或风扇的特定摆放位置可能会影响通过器件的有效空气流动，从而影响热耗散。这一参数的默认数值为 250 LFM。如果计划让 FPGA 器件在无气流运动下工作 （静止空气运行），应将 250 LFM 默认值改为 0 LFM。

  在 Tcl 提示信息中，输入以下内容以设置气流选项：

  ```
  set_operating_conditions - airflow 250
  ```

- 散热片 （如果有）

  如果使用散热片，而且没有比较具体的散热信息，应针对散热片种类选择合适的外形。这一要求（以及其它输入参数）有助于计算有效的 ThetaJB（PCB 热阻），从而实现更精确的结温和静态功耗计算。根据插座的设计以及结构，有些插座可起到散热片的作用。

  在 Tcl 提示信息中，输入以下内容以设置散热片选项：

  ```
  set_operating_conditions - heatsink low
  ```

- 电路板选择和板层数量 （如果有）

  选择合适的电路板尺寸和层数，并考虑电路板本身的导热性，有助于计算有效的 ThetaJB。

- ThetaJB （印刷电路板 PCB 热阻）

  如果有更加精确的电路板和系统的热模型，应使用该模型规定来自 FPGA 器件的散热量。

  自定义 ThetaJB 设定的越精确，结温的估计值就越精确，从而影响器件静态功耗计算。

重要提示： 要设定自定义 ThetaJB，必须将“Board Selection”设置为“Custom”。如果设定了自定义 ThetaJB，还必须设定“Board Temperature”用以精确计算功耗。

在 Tcl 提示信息中，输入以下内容以设定单板选项：

```
set_operating_conditions - board jedec
```

在 Tcl 提示信息中，输入以下内容以设置散热片选项：

```
set_operating_conditions - thetajb 3
```

供电电压设置

查看“Report Power”对话框“Power Supply”选项卡中不同的用户可编辑选项。

- 电源

如果电源信息已知，应确保在“Power Supply”选项卡中针对不同电源正确设定所有电压级。电压是产生静态和动态功耗的主要因素。

在 Tcl 提示信息中输入以下内容，以设定“VccAux”电压轨的电压：

```
set_operating_conditions -voltage {Vccaux <value>}
```

在运行 Vivado Design Suite 功耗分析后检查设计的功耗分配

打开“Summary”视图，查看“Total On-Chip Power”和热属性。 “On-ChipPower”图表显示了每种器件资源的功耗。有了这个高级视图，您就可以确定设计中的哪些部分对总功耗的贡献最多。 “Power Supply”选项卡显示每个电源的电流消耗，并将总值分解为静态和动态功耗。

如需在“Utilization Detail”选项卡中查看器件资源功耗的更多详情，可点击图表中的不同资源类型。不同资源视图被组织成树形表。可拖拽列标题改变列顺序。点击列标题可改变排列顺序。

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。

“Report Power”提取并列出“Signal”视图中所有不同的控制信号。您可能从应用的预期行为方式中了解到有些“Set/Reset”信号在正常设计操作时处于非活动状态。这种情况下，可能需要调整这些信号的活动状态。类似地，应用中的有些信号在某些模块未使用时会禁用设计中的所有模块。应根据预期功能调整它们的活动状态。

综合工具和布局布线算法可推理或重新映射控制信号以优化您的 RTL 描述，因此视图中所列出的很多信号可能比较陌生。如果您不确定这些是什么信号，可以用工具确定其活动状态。

写出功耗报告文本文件

从 Vivado IDE 中的 Flow Navigator 窗口打开“Report Power”对话框。使用该对话框检查功耗设置，并调整您的设计中已知元素的活动状态。

为进行工程存档，您可能希望将功耗估计结果保存在输出文本文件中。

在其它情况下，您可能要试验不同的映射、布局布线选项，才能满足性能或区域约束。当多个试验结果满足要求时，保存每次试验的功耗结果有助于您选择功耗最低的解决方案。

在 Tcl 提示信息中输入如下内容：

```
report_power -file report.pwr
```

功耗引擎在当前工作目录下写出一个 `report.pwr` 文件。该文件包含功耗估计结果。

将 Vivado 工具的功耗估计结果导出到 XPE

从 Vivado IDE 中打开“Report Power”对话框。您可在该对话框中检查功耗设置，并调整您的设计中已知元素的活动状态。

XPE 输出文件将所有环境信息、器件使用和设计活动保存在一个(`.xpe`)文件中，随后您可将该文件导入到 XPE 电子数据表。当超出功耗预算，而且仅凭软件优化功能无法满足您的预算要求时，该文件可派上用场。

在这种情况下，您可以：

- 将当前的实现结果导入到 XPE。
- 探索不同的映射、门控、折叠 (folding) 及其它策略。

- 在修改 RTL 代码或重新运行实现之前评估它们对功耗的影响。

将您 XPE 电子数据表中的假设与这些最终结果进行对比。这样有助于在以后的设计中为 XPE 提供更精确的输入。

在 Tcl 提示信息中输入如下内容：

```
report_power -xpe report.xpe
```

功耗引擎在当前工作目录下写出一个 report.xpe 文件。该文件现可导入到 XPE 电子数据表中。

功耗优化

如果功耗估计值超出预算，您必须采取如下措施以降低功耗。

- 分析功耗估计和优化结果
- 运行功耗优化
- 使用功耗优化报告
- 使用时序报告确定功耗优化的影响

分析功耗估计和优化结果

功耗估计结果生成之后，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，应根据模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模块。
- 检查“Hierarchy”部分。按照层级划分后，很容易找到功耗最高的模块。可以深入研究特定模块以确定各模块的功能。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，敬请参阅：第 4 章中的改善功耗的编码方式。

功耗优化功能能够使整个设计或部分设计 （当使用`set_power_opt`时）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}
```

第 6 章

配置与调试

配置与调试简介

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。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。

请参阅下面两个用户指南，以详细了解配置及调试软件的流程和命令：

- 《Vivado Design Suite 用户指南：编程与调试》(UG908) [参照 24]
- 《Vivado Design Suite Tcl 命令参考指南》(UG835) [参照 13]

配置

本章节介绍在您选择好配置模式后如何成功实现目标配置解决方案，以及将设计加载到 FPGA 器件中的方法。如欲了解初始规划阶段的常见配置模式和建议，敬请参阅以下资料：

- 《Vivado Design Suite 用户指南：编程与调试》(UG908) [参照 24]
- 《UltraScale FPGA BPI 配置与闪存编程》(XAPP1220) [参照 49]
- 《利用 7 系列的 BPI Fast 配置与 iMPACT 闪存编程》(XAPP587) [参照 50]
- 《结合 SPI 闪存使用 7 系列 FPGA》 (XAPP586) [参照 51]
- 《UltraScale FPGA 的 SPI 配置与闪存编程》(XAPP1233) [参照 52]
- 《使用加密来确保 7 系列的 FPGA 比特流安全》(XAPP1239) [参照 53]
- 编程及调试视频教程可从 Vivado Design Suite 视频教程的赛灵思网站的页面上获得

必须在成功完成您设计的综合及实现后，才能创建比特流 (`.bit`) 镜像。在创建好比特流之后，通过下述两种方法其中的任何一种即可将其加载到 FPGA 器件：

- **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 FPGA 器件。

- **间接编程**

  可先将比特流加载到外部闪存存储器。闪存存储器再将比特流加载到 FPGA 器件。

赛灵思提供的软件工具可用于：

- 创建 FPGA 比特流 （`.bit` 或 `.rbt`）
- 将比特流格式转换成闪存编程文件 （`.mcs`）
- 直接对 FPGA 器件进行编程

- 间接对连接的配置闪存器件进行编程

比特流生成

比特流 (.bit) 是代表用户设计的二进制文件。比特流包含的配置数据可载入 FPGA 器件。一些比特流文件格式选项和多种比特流输入选项可通过对 FPGA 内部配置寄存器进行初始化来支持相关特性。在生成比特流文件之前，应先分析在 XDC 文件或 Tcl shell 的比特流的生成设置 `set_property`，确保其针对所选的目标配置模式进行了正确设置，这一点非常重要。

例如，在主配置模式下(Master SPI 或 Master BPI 等），您可以使用内部配置时钟 (CCLK) 或外部主配置时钟 (EMCCLK)。使用适当的 `set_property` (例如, `BITSTREAM_CONFIG_CONFIGRATE` 或 `BITSTREAM.CONFIG.EXTMASTERCCLK_EN`)来使时钟源定向。

如需详细了解比特流生成属性，敬请参阅以下资源：

- 《Vivado Design Suite 用户指南：编程与调试》(UG908) [参照 24]
- 《Vivado Design Suite Tcl 命令参考指南》(UG835) [参照 13]
- 如需了解有关如何使用 `write_bitstream` 命令的更多信息，敬请观看：Vivado Design Suite QuickTake 视频：如何在 Vivado Design Suite 中使用“write_bitstream”命令，或参考 Vivado® Design Suite Tcl shell 中有关该命令的帮助信息。

使用 `list_property_value` 可查看属性可使用的值，也可参见 《Vivado Design Suite 用户指南：编程与调试》(UG908)中的这个链接。

闪存文件生成

Vivado Design Suite Tcl 命令 `write_cfgmem` 支持 FPGA 比特流 (`.bit`) 转换成为一种标准的闪存编程文件格式 (`.mcs`)。

在产生 .mcs 编程文件时，必须正确点击 `write_cfgmem` 接口选项，以确保按总线宽度规定的数据顺序操作，并确保通过闪存成功配置 FPGA。如需可用选项，敬请参阅： 《Vivado Design Suite 用户指南：编程与调试》(UG908) [参照 24]。

Vivado Design Suite 器件编程器：系统内 JTAG 编程

Vivado Design Suite 器件编程器具有多项功能。此编程器最常见的用途就是通过 JTAG 和赛灵思配套提供的下载线缆为 FPGA 器件编程，间接编程外部 SPI 闪存和并行 NOR 闪存，或对 FPGA 器件 eFUSE AES 密钥或用户代码进行编程。

如需了解更多支持和命令信息，敬请查阅：

- 《Vivado Design Suite Tcl 命令参考指南》(UG835) [参照 13]
- 《Vivado Design Suite 用户指南：编程与调试》(UG908) [参照 24]

基于 JTAG 的编程模式需要 JTAG 线缆接口。在使用 Vivado Design Suite 器件编程器对 FPGA 器件进行编程时，请记住 JTAG 最大频率受限于 JTAG 链中速度最慢的器件。

间接编程 SPI NOR 或并行 NOR 闪存

对于采用外部闪存的基本配置解决方案而言， FPGA 器件自动在加电时从闪存存储器检索比特流。由于可将 FPGA 器件连接到闪存存储器进行配置，这就使 FPGA 器件能够通过连接接口对闪存进行编程。

系统内间接编程闪存是个常用选项。 Vivado Design Suite v2014.1 和更高版本）支持间接编程某些“SPI NOR”选项和“并行 NOR 闪存”。通过加载的间接编程比特流对闪存进行间接编程。获取支持的闪存存储器器件列表，敬请参阅 《Vivado Design Suite 用户指南：编程与调试》(UG908) [参照 24]。

Vivado Design Suite 比特流镜像控制或只访问 FPGA 配置闪存接口引脚。所有其它的 FPGA SelectIO™ 技术引脚未被使用。您可对未使用的 SelectIO 引脚进行编程来获得所有的内部上拉电阻，所有的内部下拉电阻，或所有的既不上拉也不下拉的电阻。在间接闪存编程期间，对于出乎意料的单板信号活动，可在“Vivado Design Suite”选项中检查这些未使用的 SelectIO 技术引脚。

表 6-1 : 闪存间接编程建议

操作	建议
擦除	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。
BlankCheck	确认擦除操作。
读回	将闪存内容读回文件，与原始闪存编程文件进行比较。该操作将回读全部闪存内容，而不仅是指定 MCS 覆盖的地址范围。

如需了解间接编程 SPI 闪存或并行 NOR 闪存以及相关命令的更多信息，敬请参阅以下资料:

- 《UltraScale FPGA BPI 配置与闪存编程》(XAPP1220) [参照 49]
- 《利用 7 系列的 BPI Fast 配置与 iMPACT 闪存编程》(XAPP587) [参照 50]
- 《结合 SPI 闪存使用 7 系列 FPGA》(XAPP586) [参照 51]

基本配置调试

当您在实现配置解决方案时，一旦遇到问题，本章节探讨的最佳实践可助您一臂之力，提供调试和解决办法。

在开始配置解决方案的全面调试前，您应采用比特流默认值创建并测试简单设计 （例如，计数器或 LED 输出模式）。这种简单设计测试有助于排除高级比特流设置或单板接口的潜在问题。

文件生成审核

如果未成功完成配置工作，应审核是否正确选择了比特流属性和闪存编程文件选项。要验证镜像使用的比特流生成选项，请运行以下 Tcl 命令：

```
report_property [current_design]
```

该命令显示应用到设计中的所有属性，且都已经过修改。如没有显示值，则说明应用了默认值。

此外，应审核 `write_cfgmem` 闪存编程文件生成选项。

状态寄存器

如果未正确完成配置，状态寄存器会提供重要信息，列出哪些错误可能导致此类故障。如需了解更多信息，敬请参阅：

- FPGA 系列配置用户指南
- 赛灵思配置解决方案中心问答记录 34909

赛灵思 FPGA 器件的 FPGA 状态寄存器数据可由 Vivado Design Suite 器件编程器通过 JTAG 读取。如发生配置故障，该寄存器会捕获具体错误条件，以助查明故障原因。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。有关该状态寄存器的详细介绍，敬请参阅： 《7 系列 FPGA 配置用户指南》(UG470) [参照 35] 和《UltraScale 架构配置高级规范用户指南》(UG570) [参照 42]。

验证和回读

如配置不成功， FPGA 器件上的 JTAG 回读/验证操作可明确所需的配置数据是否已正确载入器件。如有偏差，可调查这一偏差情况。为了让 Vivado Design Suite 器件编程器执行 JTAG 验证操作，需要掩膜 (`.msk`) 文件。该文件可在比特流生成的过程中进行创建。

重要提示： 对于 UltraScale™ 器件,如果您尝试在将密钥编程到 BBR 寄存器之前加载加密比特流(使用 BBR 作为关键资源),FPGA 器件将会锁定,并且您将不能加载 BBR 密钥。您始终都能加载未加密的比特流,但您不能加载已加密的比特流,因为 FPGA 器件将会阻止您把密钥下载到 BBR 中。您必须对单板进行重启 (power-cycle) 来取消 UltraScale 器件的锁定,然后再重新加载 BBR 密钥。

配置顺序

配置期间，一些基本的检查有助于将问题隔离出来。赛灵思 FPGA 比特流有一个唯一的报头，报头包含特殊的同步字，也可能包含自动检测、配置时钟类型和速率设置等。大多数赛灵思 FPGA 器件的“同步(Sync)”字为 32 个字节(bit [31:0] = AA995566)，并且是重要的调试参数。

只有在 FPGA 引脚上的“Sync”字得到确认后 FPGA 配置状态机才开始启动。如配置未启动，您可观察配置数据引脚，确保正在正确接收同步字。此外，如果正确设置比特流报头，那么配置或 EMCCLK 选项设置造成的任何提升都应该能在配置时钟里看到，否则报头将不会被识别。

配置启动

根据常见配置顺序给 FPGA 器件加电。 FPGA 器件配置用户指南对此有详细介绍。下列 FPGA 特性能延长配置启动顺序：

- Wait for PLL
- MMCM lock
- DCI match

如果使用以上任何选项，请确保配置源的文件为 MultiBoot 镜像提供了适当的间隔。如需了解有关 MultiBoot 镜像处理的更多信息，敬请参阅：《FPGA 系列配置用户指南》。此外，在使用从串模式时，请确保提供足够的时钟周期，从而完成启动顺序。

如果启动时钟 （JTAGCLK、 CCLK、 EMCCLK）在启动过程中未达到启动顺序末端，则会出现以下症状，可能意为启动不正确或未完成：

- I/O 保持为三态。
- 双模引脚工作在 LVCMOS 中而不是在指定的 I/O 标准中。
- ICAP 接口不能从 FPGA 器件架构中进行访问，因为配置逻辑被锁定。

启动成功完成的表现为 EOS 信号驱动变为高电平。用 STARTUP 原语可在 STATUS 寄存器中观察或在 FPGA 器件架构中检测到这一变化。

对访问 ICAP 的设计而言，赛灵思建议您将 STARTUP 原语实例化，该原语有一个 EOS 引脚，表明：(1)配置程序已经完成；以及(2)ICAP 可用于读写访问。

远程更新

赛灵思 FPGA 器件支持 MultiBoot 和回读特性，使现场系统更新更加稳健可靠。比特流镜像可进行现场动态更新。MultiBoot 和回读特性可与所有主配置模式配合使用。

MultiBoot 特性支持用户应用在镜像间进行切换。如果在 MultiBoot 配置进程中检测到错误，那么器件可触发回读机制，从不同的闪存地址中检索已知的比特流。

稳健系统内升级解决方案的实施涉及一系列有关初始配置方法、升级方法和回读机制的决策。

MultiBoot 解决方案需要闪存足够大，可容纳下所需的所有比特流。由于压缩结果各不相同，因此赛灵思建议在规划闪存存储器映射时考虑最大未经压缩的比特流。

在以下几种特殊情况下，回读需要使用高级选项。

- 在 7 系列 FPGA 中， Master SPI 配置模式回读为 x1 模式。
- 在 7 系列 FPGA 中， BPI 配置模式同步读取回读为异步读取模式。这意味着如采用回读，那么同步读取的更高时钟速度可能会读取失败。所用的时钟频率必须可同时支持这两种模式。

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。

调试涉及的步骤包括：

1. **探针探测**：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，敬请参阅： 《Vivado Design Suite 用户指南：编程与调试》(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 6-2 : 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 调试内核实例。	• 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、实例化和连接调试内核时容易出错

表 6-2 : 调试流程

调试流程名称	流程步骤	优/劣
网表插入探针探测流程（建议） 这种方法只对 ILA 2.1 或更高版本有效。	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 Wizard 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 命令创建调试内核并连接其到调试网络。 参见 修正网表 了解后综合 ILA 内核情况。	• 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。

两步调试插入法

在对象网表上使用 MARK_DEBUG 和在设计中插入调试内核可分为两个步骤：

1. 确定需要调试的网表，方法包括：右键点击网络，设置 MARK_DEBUG 属性；或使用属性窗口和/或 Tcl 命令设置该属性。
2. 在综合完成后使用 Set up Debug wizard。使用 wizard 的方式包括从“Flow Navigator Synthesis”视图打开，或当您正在进行设计时，点击“Tools” > “Set up Debug”菜单项。

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 Wizard 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 内核

使用 Integrated Logic Analyzer (ILA) 内核可以在 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，就可以使用这个内核。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。

赛灵思建议在综合后插入 ILA 内核，这样就不必修改 HDL 源文件，也无需再重新验证设计。

向调试内核添加网络，请打开综合设计并从“Flow Navigator”窗口中选择“Set up Debug”，或选择“Tools > Set up Debug”菜单项。

如需了解更多信息，敬请参阅： 《Vivado Design Suite 用户指南：编程与调试》(UG908) [参照 24]。

ILA 内核和时序考虑事项

ILA 内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 内核数据深度。数据深度越大，对块状 RAM 资源利用和时序的影响越大。
- 确保为 ILA 内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内核通信。
- 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内核通信。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。
- 在添加调试内核之前收敛设计上的时序。赛灵思不建议使用调试内核调试相关时序问题。
- 如果您仍然观察到因添加 ILA 调试内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 内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 6-3 : 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 (BRAM)	中影响力到高影响力	• 无需额外的 BRAM • 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	• 无需额外的 BRAM • 稳健增加 LUT/FF 数量
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	• 无需额外的 BRAM • 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 内核的额外 BRAM • 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 内核的额外 BRAM • 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 内核的额外 BRAM • 轻微增加 LUT/FF 数量

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。

支持 ILA Cross Trigger 模式

ILA Cross Triggering 功能可在 ILA 之间以及 ILA 与处理器(例如，Zynq®-7000 AP SoC 或 MicroBlaze™ 器件)之间实现交叉触发。

对于使用 Cross-Trigger 模式，在内核产生的时候，您应该设置 ILA 内核来获得专用的触发输入端口 (TRIG_IN和 TRIG_IN_ACK) 和专用的触发输出端口 (TRIG_OUT和TRIG_OUT_ACK)。如果您想使用 ILA 触发器输入或输出信号，可以考虑使用 HDL 实例化方法，将 ILA 内核添加到您的设计中。如果您将 ILA 插入您的设计后综合之后，请注意需要使用网表修改命令来将这些端口连接到您的设计网络中。参见修正网表了解后综合 ILA 内核情况。

如需获得介绍有关在 FPGA 架构和 Zynq-7000 AP SoC 处理器之间的使用交叉触发器功能的相详细教程，敬请参阅：《Vivado Design Suite 教程：嵌入式处理器硬件设计》(UG940) [参照 32]

在硬件中调试

调试内核实现在设计中在设计中后，可使用运行时间逻辑分析器功能在硬件中调试设计。要使用 Vivado Design Suite 的逻辑分析器与实例化在设计中的 ILA 调试内核进行交互，请点击："Flow Navigator > Program and Debug > Open Hardware Session"。

在本步骤中，应完成下列工作：

- 连接您的目标硬件
- 把比特流编程到器件中
- 设置ILA调试内核触发器和探测条件
- 加载 ILA 调试内核触发器
- 在波形窗口中分析从 ILA 调试内核采集的数据。

ILA Trigger 模式设置

触发器模式控制着实时硬件事件的检测，体现为采集窗口中的触发器标识。

- BASIC_ONLY：当调试探头比较结果的基本 AND/OR 功能得到满足时，使用 ILA Basic Trigger 模式触发 ILA 内核；
- ADVANCED_ONLY:使用 ILA Advanced Trigger 模式触发设定为用户定义状态机的 ILA 内核；
- TRIG_IN_ONLY:当 ILA 内核的 TRIG_IN 引脚从低电平变为高电平时，使用 ILA TRIG_IN Trigger 模式触发 ILA 内核。
- BASIC_OR_TRIG_IN:如果需要 ILA 内核的 TRIG_IN 引脚和 BASIC_ONLY Trigger 模式处于 logical ORing 状态，使用 ILA BASIC_OR_TRIG_IN Trigger 模式触发 ILA 内核。
- ADVANCED_OR_TRIG_IN:如果需要ILA内核的 TRIG_IN 引脚和 ADVANCED_ONLY Trigger 模式处于 logical ORing 状态，使用 ILA ADVANCED_OR_TRIG_IN Trigger 模式触发 ILA 内核。

可将触发器位置设定为数据采集缓存中的特定位置。例如对采样深度为 1024 的数据采集缓存：

- 采样号 0 对应数据采集缓存中第一个 （最左边）的采样；
- 采样号 1023 对应数据采集缓存中最后一个 （最右边）的采样；
- 采样号 511 和 512 对应数据采集缓存中的中间两个采样。

ILA 内核采集模式

采集模式负责控制 ILA 内核采集数据的方式。

- ALWAYS:在每一个时钟周期上采集探测数据。
- BASIC:在调试探头比较结果的基本 AND/OR 功能得到满足时采集探测数据。

共享通过 ILA 内核所采集数据的波形

用户可使用 `write_hw_ila_data` Tcl 命令把 ILA 采集的数据保存到一个存档文件中。如果需要从这个存档文件中恢复采集的数据并显示在波形观察器中，应使用 Tcl 命令 `display_hw_ila_data`。

使用这两个命令就可以查看 ILA 内核所采集数据的波形。在运行上述命令之前，应先运行 `open_hw` 命令。

ILA 数据波形窗口的波形配置设置 （分频器、标识、色彩、探头基数等）也保存在 ILA 采集数存档文件中。可以用这些存储的波形配置设计恢复和显示任何之前保存的ILA数据。

多次采集窗口

可以通过多次采集功能采集多次触发得到的信号。使用这项功能应首先在运行时间选择采集窗口的量，如图6-1 所示，然后和正常一样进行触发。

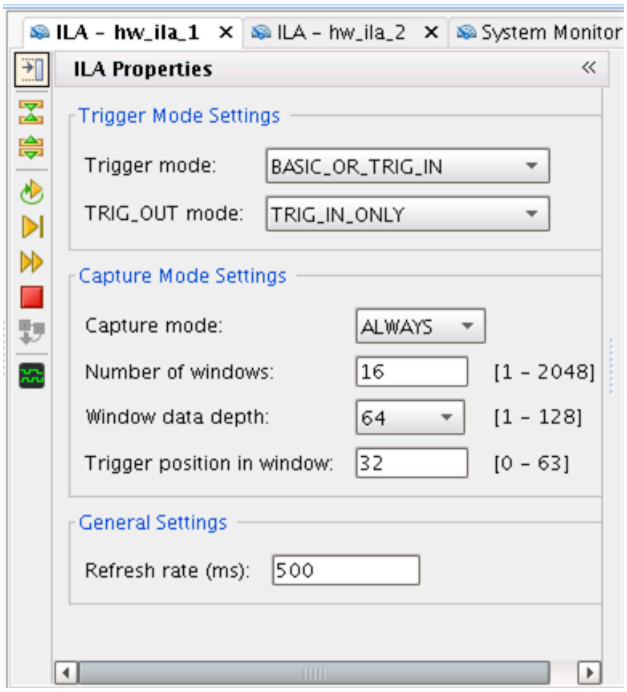


图 6-1 : 针对多次采集窗口设置 ILA

图6-2 为多次采集窗口示例，在信号 `fast_cnt_reset_1` 的每次下降沿进行触发。观察带有触发器标识的多个窗口和交替的“checkerboard”窗口背景。

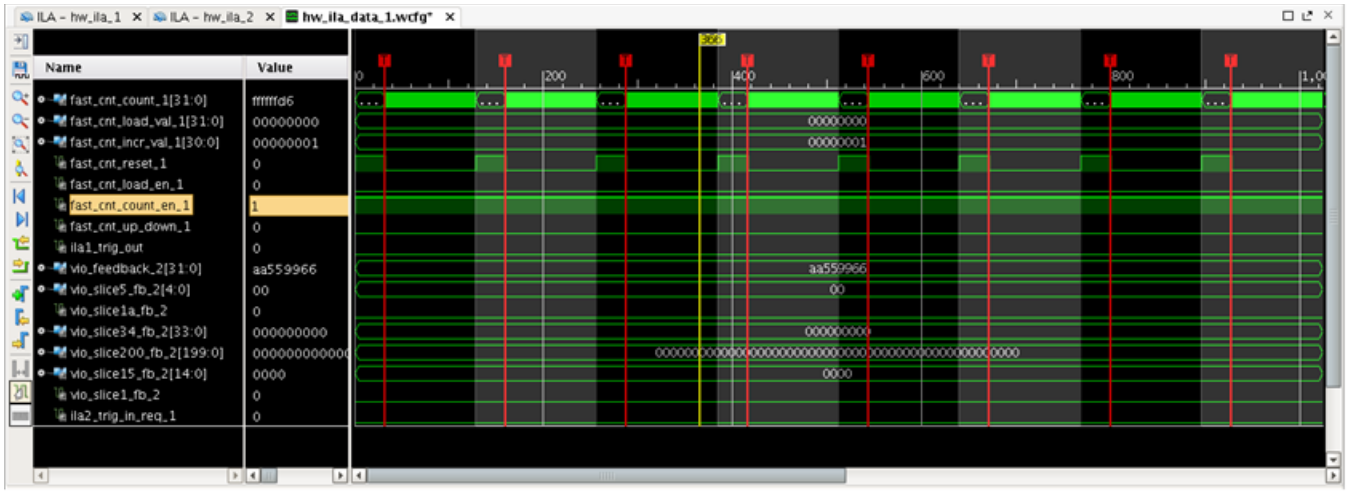


图 6-2 : 多次采集窗口示例

虚拟 I/O （VIO）

Virtual Input/Output (VIO) 内核是一种可定制内核，能够实时监测和驱动内部 FPGA 信号。在无法物理访问目标硬件的时候，可以使用这个 IP 驱动和监测出现在真实硬件上的信号。有两种不同的输入内核和两种不同的输出内核，大小均可定制，以便与设计接口。如果需要 VIO 内核，可从IP 目录中选择，然后实例化在设计中。

VIO 内核输出探头用于把值写入运行在 FPGA 器件上的设计。VIO 输出探头一般用作受测设计的小带宽控制信号。与此类似， VIO 输入引脚用于从运行在 FPGA 器件上的设计读取值。

VIO 内核可完成下列工作：

- 通过输入端口提供虚拟 LED 及其它状态指示器；
- 在输入端口提供可选活动检测器，用于检测采样之间的上升和下降转换；
- 通过输出端口提供虚拟按钮及其它控制；
- 提供定制输出初始化功能，以便在器件配置和驱动完成后立即设定 VIO 内核输出值；
- 运行 VIO 内核的时间重置功能，恢复其初始值。

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内核添加到 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。

- HDL 实例化流程

  使用该流程来：

  - 使用 MicroBlaze 器件或 Zynq-7000 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

  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 ILA 的信息，敬请参阅： 《Vivado Design Suite 用户指南：采用 IP 集成器设计 IP 子系统》(UG994) [参照 26]

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` 时作为默认规则检查的一部分。DRC 检查如下所示:

- 器件的块状 RAM 资源是超出的因为调试内核的现今需求。
- 无时钟网络连接到调试内核的时钟端口。
- 调试内核上的端口未连接。

生成 AXI 事务

使用 JTAG-to-AXI 调试内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 目录将此内核实例化到设计中，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该内核，且无需处理器。

使用探针探测文件

探针探测文件是一个对应于与器件关联的 .bit 文件的 .ltx 文件。Vivado 工具会在实现过程中自动生成该探测文件。如果 Vivado 工具处于工程模式 （Project Mode）下，且如果探针探测文件 （debug_nets.ltx）所在的目录和与该器件相关的比特流编程 （.bit）文件相同， Vivado 工具还能自动将调试探针探测文件与硬件器件关联。

如果怀疑编程到器件中的比特文件和与 .bit 文件相关的探针探测文件不匹配，应确保 .bit 文件和探针探测文件更新到最新版本。

如要把调试探头探测信息写入文件，应在综合设计中使用 write_debug_probes Tcl 命令。使用下列流程即可设定探针探测文件的位置：

1. 在硬件窗口中选择 FPGA 器件；
2. 在硬件器件属性窗口中设置探针探测文件位置；
3. 点击“Apply”。

您也可以使用 set_property Tcl 命令设置位置：

```
set_property PROBES.FILE {location} [lindex [get_hw_devices] 0]
```

降低 JTAG 时钟速度

JTAG 链的速度由该链条中速度最慢的器件决定。因此如果要降低 JTAG 时钟频率，应连接到 JTAG 时钟频率低于默认 JTAG 时钟频率的目标器件。

对 Diligent 线缆连接，应尽量用默认的 15MHz JTAG 时钟频率；对 USB 线缆连接，应使用 6MHz 的 JTAG 时钟频率。如果无法以这些速度连接，赛灵思建议根据下面的介绍，进一步降低默认的 JTAG 时钟频率。

如果要修改 JTAG 时钟频率，请使用 Vivado IDE 中的 Open New Hardware Target Wizard，如图6-3 所示。

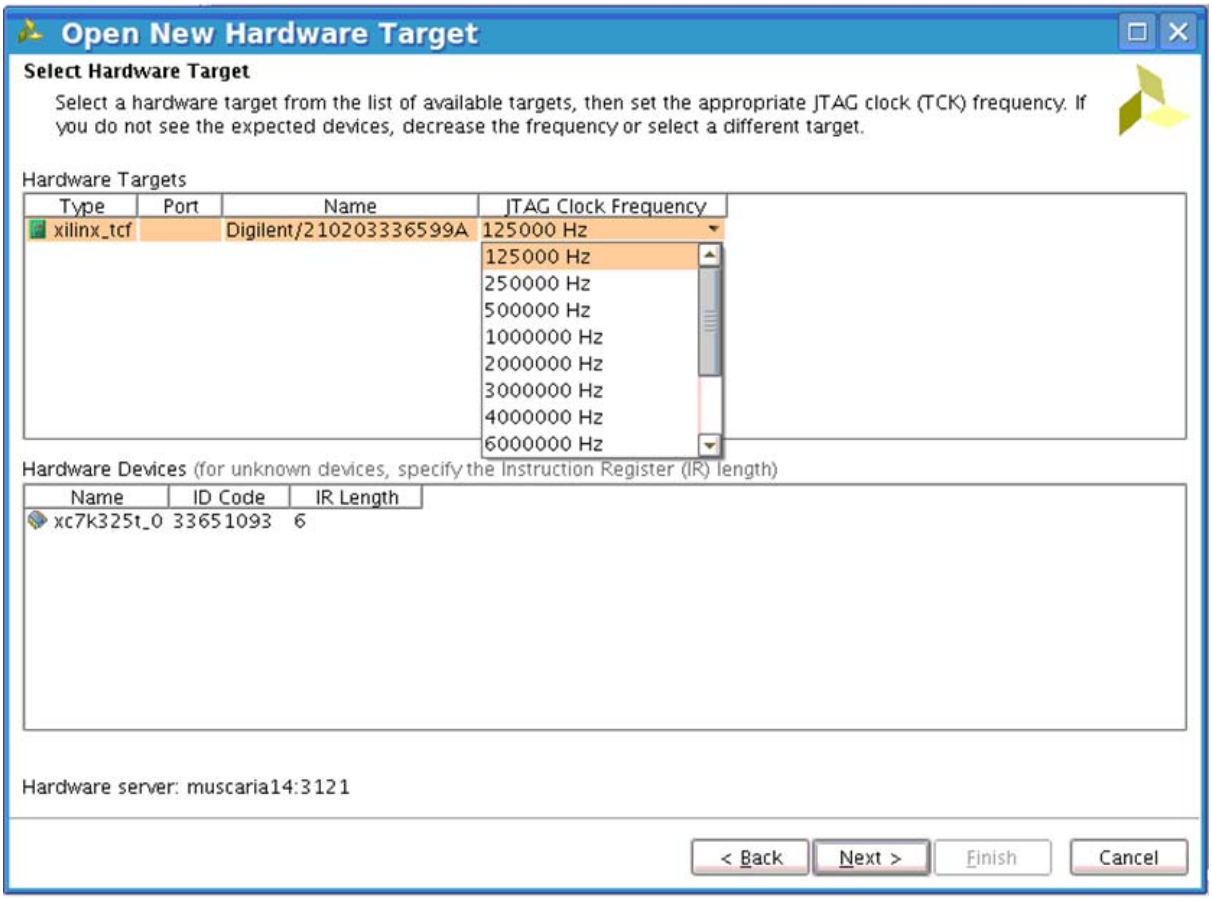


图 6-3 : 设置 JTAG 频率

另外，还可以使用下列 Tcl 命令序列：

```
open_hw
connect_hw_server -host localhost -port 60001 -url machinename:3121
current_hw_target [get_hw_targets */xilinx_tcf/Digilent/210203327962A]
set_property PARAM.FREQUENCY 250000 [get_hw_targets
*/xilinx_tcf/Digilent/210203327962A]
open_hw_target
```

Vivado 调试布局

当设计加载到器件上时， Vivado IDE 提供与调试进程有关的具体布局。将 Vivado IDE 设置为正确布局，以确保在调试环境中显示正确的窗口。在使用 ILA 和 VIO 内核调试设计时，应使用 Logic Analyzer 布局。在使用 IBERT 内核调试设计时，应使用串行 I/O Analyzer 布局。

Tcl 对象与命令

表 6-4 : Tcl 对象与命令

Tcl 对象	代表
hw_server	硬件服务器
hw_target	硬件目标（线缆）连接。A hw_target 是实时物理地链接到服务器。

表 6-4 : Tcl 对象与命令

Tcl 对象	代表
hw_device	硬件中的实际器件。可通过连接至硬件的实时服务器自动检测该器件，也可通过 hw_part 查询进行手动创建。
hw_ila	ILA 内核。
hw_ila_data	采集的 ILA 数据以及探询信息。
hw_probe	调试探针探测。可将探针探测与您设计中的网表进行关联。
hw_probeset	一系列 hw_probe 对象。
hw_propset	硬件内核中的属性集。

本章描述的许多行为都可通过使用 Tcl 命令实现。如需了解 Tcl 的调试命令介绍，敬请参阅：《Vivado Design Suite Tcl 命令参考指南》(UG835) [参照 13]。

建议使用的设计实践

1. 在设计阶段要尽早提前做好调试计划，并应遵循以下指南：
   a. 预留逻辑 Slice 和用于系统内调试的块状 RAM。
   b. 确保可通畅访问至 FPGA 器件的 JTAG 接口，并能用于调试。
   c. 明确设计调试所需的相关调试流程。该流程可为：
      - HDL 实例化
      - 网表插入 （建议）
      - 两个流程的组合使用
   d. 考虑探询如触发器和块状 RAM 等同步单元输出，而不是整合逻辑。这能最小化对设计优化的影响，而且还能增加满足时序要求的几率。
   e. 为您的设计添加定制调试逻辑。
2. 在执行调试时，应遵循以下指南：
   a. 不要同时测试整个设计。
   b. 以增量方式修改设计，每次逐一测试特性。
   c. 考虑使用增量编译实现设计。
   d. 将控制和高速数据信号布线到引脚，以用于逻辑分析器或示波器 (scope) 上的分析。
   e. 为了基于事件来采集数据，可将调试内核添加到您的设计中。
3. 考虑在 RTL 仿真中重建问题，并验证补丁在仿真中是否也能正常运行。
4. 当您成功进行设计调试后，应考虑在投产之前移除调试内核，以便：
   a. 减少使用 JTAG 进行非授权访问设计的可能性。
   b. 降低功耗。
   c. 考虑将调试内核添加到您的设计中可能对设计时序约束产生的影响。

如需了解更多信息，敬请参阅：

- 《Vivado Design Suite 用户指南：编程与调试》(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 35]
- 《UltraScale 架构配置高级规范用户指南》(UG570) [参照 42]

修正网表

有时需要变更网表以固定功能逻辑缺陷，从而满足时序约束或插入调试逻辑。这通常被称作“工程变更规则 (ECO)”。您可使用综合后、布局后和布线后的 Tcl 命令来修正已存在的网表。如需了解更多修正网表命令，敬请参阅： 《Vivado Design Suite 用户指南：实现》(UG904) [参照 19]。

附录 A

基线(baselining)与时序约束验证流程

引言

在设计按实现步骤向前推进，约束条件不断得到优化的同时，应填写下列问卷。该问卷有助于跟踪时序收敛的流程/偏差、任何潜在瓶颈和需要修改的约束。

流程

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						

12. 运行 `report_exceptions -ignored`，查看设计中是否存在约束重叠。记录结果。

附录 B

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，敬请访问：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，敬请参见：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。

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 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：编程与调试》(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)

Vivado Design Suite 教程

28. 《Vivado Design Suite 教程：高层次综合》(UG871)
29. 《Vivado Design Suite 教程：设计流程简介》(UG888)
30. 《Vivado Design Suite 教程：I/O 和时钟规划》(UG935)
31. 《Vivado Design Suite 教程：逻辑仿真》(UG937)
32. 《Vivado Design Suite 教程：嵌入式处理器硬件设计》(UG940)
33. 《Vivado Design Suite 教程：部分重配置》(UG947)

其它赛灵思技术文档

34. 《UltraFast 嵌入式设计方法指南》(UG1046)
35. 《7 系列 FPGA 配置用户指南》(UG470)
36. 《7 系列 FPGA SelectIO 资源用户指南》(UG471)
37. 《7 系列时钟资源指南》(UG472)
38. 《7 系列 FPGA 存储器资源用户指南》(UG473)
39. 《7 系列 FPGA DSP48E1 Slice 用户指南》(UG479)
40. 《UltraScale DSP Slice 用户指南》(UG579)
41. 《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)
42. 《UltraScale 架构配置高级规范用户指南》(UG570)
43. 《参考系统：使用 IP 集成器的 Kintex-7 MicroBlaze 系统仿真》(XAPP1180)
44. 《Zynq-7000 SoC 与 7 系列 FPGA 器件存储器接口用户指南》(UG586)
45. 《基于 LogiCORE IP UltraScale 架构的 FPGA 存储器接口解决方案产品指南》(PG150)
46. 赛灵思白皮书：《使用 S 参数模型仿真 FPGA 电源完整性》(WP411)
47. 7 系列原理图查看建议(XMP277)
48. UltraScale 架构的原理图查看检查表(XTP344)
49. 《UltraScale FPGA BPI 配置与闪存编程》(XAPP1220)
50. 《利用 7 系列实现 BPI 快速配置与 iMPACT 闪存编程》(XAPP587)
51. 《结合使用 SPI 闪存和 7 系列 FPGA》(XAPP586)

52. 《利用 UltraScale FPGA 实现 SPI 配置与闪存编程》(XAPP1233)
53. 《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，敬请参阅：使用 Documentation Navigator。

培训资料

赛灵思提供多种多样的培训课程和 QuickTake 视频，可帮助用户进一步了解有关本文档中提出的概念。使用以下链接获取相关培训资料：

1. FPGA 设计基础
2. 赛灵思培训视频：UltraFast Vivado 设计方法
3. Vivado Design Suite QuickTake 视频：Vivado 设计流程简介
4. Vivado Design Suite QuickTake 视频：使用 Vivado IP Integrator 进行设计
5. Vivado Design Suite QuickTake 视频：针对 Zynq 使用 Vivado IP Integrator
6. Vivado Design Suite QuickTake 视频：Vivado Design Suite 中的部分重配置
7. Vivado Design Suite QuickTake 视频：创建不同类型的工程
8. Vivado Design Suite QuickTake 视频：管理工程设计源文件
9. Vivado Design Suite QuickTake 视频：如何使用带版本控制的 Vivado Design Suit
10. Vivado Design Suite QuickTake 视频：管理 Vivado IP 版本升级
11. Vivado Design Suite QuickTake 视频：I/O 规划简介
12. Vivado Design Suite QuickTake 视频：在 Vivado 中配置和管理可重用 IP
13. Vivado Design Suite QuickTake 视频：如何在 Vivado Design Suite 中使用“write_bitstream”命令。
14. Vivado Design Suite QuickTake 视频：定制和实例化 IP
15. Vivado Design Suite QuickTake 视频：使用 Vivado 进行设计分析和平面布局
16. Vivado Design Suite QuickTake 视频：介绍 UltraFAST 设计方法检查表
17. Vivado 视频辅导资料

请阅读： 重要法律提示

本文向贵司/您所提供的信息（下称“资料”）仅在对赛灵思产品进行选择和使用时参考。在适用法律允许的最大范围内：（1）资料均按“现状”提供，且不保证不存在任何瑕疵，赛灵思在此声明对资料及其状况不作任何保证或担保，无论是明示、暗示还是法定的保证，包括但不限于对适销性、非侵权性或任何特定用途的适用性的保证；且（2）赛灵思对任何因资料发生的或与资料有关的（含对资料的使用）任何损失或赔偿（包括任何直接、间接、特殊、附带或连带损失或赔偿，如数据、利润、商誉的损失或任何因第三方行为造成的任何类型的损失或赔偿），均不承担责任，不论该等损失或者赔偿是何种类或性质，也不论是基于合同、侵权、过失或是其他责任认定原理，即便该损失或赔偿可以合理预见或赛灵思事前被告知有发生该损失或赔偿的可能。赛灵思无义务纠正资料中包含的任何错误，也无义务对资料或产品说明书发生的更新进行通知。未经赛灵思公司的事先书面许可，贵司/您不得复制、修改、分发或公开展示本资料。部分产品受赛灵思有限保证条款的约束，请参阅赛灵思销售条款：http://china.xilinx.com/legal.htm#tos；IP 核可能受赛灵思向贵司/您签发的许可证中所包含的保证与支持条款的约束。赛灵思产品并非为故障安全保护目的而设计，也不具备此故障安全保护功能，不能用于任何需要专门故障安全保护性能的用途。如果把赛灵思产品应用于此类特殊用途，贵司/您将自行承担风险和责任。请参阅赛灵思销售条款：http://china.xilinx.com/legal.htm#tos。

© 2013–2015 年赛灵思公司版权所有。Xilinx、赛灵思标识、Artix、ISE、Kintex、Spartan、Virtex、Vivado、Zynq 及本文提到的其它指定品牌均为赛灵思在美国及其它国家的商标。 “MATLAB”和“Simulink”均为 MathWorks, Inc. 拥有的注册商标。“OpenCL”和“OpenCL”

标识均为 Apple Inc. 的商标，经 Khronos 许可后方能使用。“PCI”、“PCIe”和“PCI Express”均为 PCI-SIG 拥有的商标，且经授权使用。所有其它商标均为各自所有方所属财产。