

UltraScale+ 器件 Integrated Block for PCI Express v1.3

产品指南

Vivado Design Suite

PG213 (v1.3) 2022 年 11 月 16 日

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

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





目录

第 1 章：引言	4
功能特性	4
IP 相关信息	5
第 2 章：概述	6
功能特性总结	8
应用	9
不支持的功能	10
许可和订购	10
第 3 章：产品规格	11
标准合规性	11
资源使用情况	11
器件最低要求	11
可供 PCI Express 使用的集成块	12
GT 位置	16
端口描述	16
配置空间	67
第 4 章：用核设计	73
串联配置	73
时钟	98
复位	101
AXI4-Stream 接口描述	101
功耗管理	212
生成中断请求	215
接收报文接口	219
配置管理接口	222
在根端口上启用环回主控制器	224
链路训练：2 通道、4 通道、8 通道和 16 通道组件	224
通道翻转	225
第 5 章：设计流程步骤	227
自定义和生成核	227
核约束	249
仿真	250
综合与实现	252

第 6 章：设计示例	253
设计示例概述	253
生成核	263
打开设计示例	265
设计示例仿真	266
设计示例的综合和实现	267
第 7 章：测试激励文件	268
对应端点的根端口模型测试激励文件	268
对应根端口的端点模型测试激励文件	280
附录 A：升级	282
从 UltraScale 移植到 UltraScale+ 器件	282
在 Vivado Design Suite 中执行升级	285
附录 B：管理传入完成包的接收缓冲器空间	287
常见注意事项和概念	287
完成空间的管理方法	289
附录 C：GT 位置	292
Artix UltraScale+ 器件可用的 GT 四通道	295
Kintex UltraScale+ 器件可用的 GT 四通道	295
Virtex UltraScale+ 器件可用的 GT 四通道	299
Zynq UltraScale+ 器件可用的 GT 四通道	314
附录 D：调试	325
在 Xilinx.com 上寻求帮助	325
硬件调试	326
附录 E：使用赛灵思虚拟线缆进行调试	329
概述	329
主机 PC XVC-Server 应用	330
主机 PC XVC-over-PCIe 驱动程序	330
启用 XVC-over-PCIe 的 FPGA 设计	330
使用 PCIe-XVC-VSEC 设计示例	335
附录 F：附加资源与法律声明	343
赛灵思资源	343
Documentation Navigator 与设计中心	343
参考资料	343
修订历史	344
请阅读：重要法律声明	348

引言

UltraScale+™ 器件 Integrated Block for PCI Express® (PCIe®) 解决方案 IP 核是具备高带宽、高可缩放性和高可靠性的串行互连构建块解决方案，适用于 UltraScale+ 器件。赛灵思在 UltraScale+ 架构内提供了 2 个 PCIe 集成块：PCIE4 集成块和 PCIE4C 集成块。UltraScale+ 器件内所包含的 PCIE4 块可支持 PCIe IP。含高带宽存储器 (HBM) 的 Virtex® UltraScale+™ 器件同时包含 PCIE4 块和增强型 PCIE4C 块。PCIE4 块支持 1 通道、2 通道、4 通道、8 通道和 16 通道配置，包括 Gen1 (2.5 GT/s)、Gen2 (5.0 GT/s) 和 Gen3 (8 GT/s) 速度。它符合《PCI Express 基本规范第 3.1 版》的要求。PCIE4C 块功能上与 PCIE4 块相同，并且对于 1 通道、2 通道、4 通道和 8 通道配置还额外支持 Gen4 (16 GT/s) 速度，符合《PCI Express 基本规范第 4.0 版》的要求。该解决方案支持使用 AXI4-Stream 接口作为客户使用接口。

注释：符合《PCI Express 基本规范第 4.0 版本》要求的 PCIE4C 块与符合 Gen4 标准的器件（按 Gen4 速度）之间能够实现互操作性，但其中部分功能不受支持。PCIE4C 块彼此之间将能够以 Gen4 速度实现互操作性。为判定 PCIE4C 块是否适用于特定应用，赛灵思建议在评估 PCIE4C 块能否按 Gen4 速度来使用时，设计师应复查 [不受支持的 PCI Express 基本规范第 4.0 版功能 \(PCIE4C\)](#) 中所述的不受支持的功能。

功能特性

- 其设计符合《PCI Express 基本规范第 3.1 版》的要求。
- PCIe® 端点、传统端点或根端口模式。
- 针对 PCIE4 和 PCIE4C 块支持 x1、x2、x4、x8 或 x16 链路宽度以及 Gen1、Gen2 和 Gen3 链路速度。
- 针对 PCIE4C 块支持 x1、x2、x4 和 x8 链路宽度以及 Gen4 链路速度。
- AXI4-Stream 接口，可连接到客户逻辑。
- 高级错误报告 (AER) 和端到端 CRC (ECRC)。
- 用于传输事务缓冲的块 RAM。
- 1 个 PCIe 虚拟通道和 8 个流量类。
- 最多支持 4 项物理功能和 252 项虚拟功能。
- 完全可配置的 3 × 64 位或 6 × 32 位基址寄存器 (BAR)。

如需获取完整特性列表，请参阅 [功能特性总结](#)。

IP 相关信息

LogiCORE™ IP 相关信息表	
核规格	
支持的器件系列 ¹	UltraScale+
支持的用户接口	AXI4-Stream
资源	性能和资源使用情况网页 ²
随核提供	
设计文件	Verilog
设计示例	Verilog
测试激励文件	Verilog
约束文件	XDC
仿真模型	Verilog
支持的软硬件驱动程序 ²	不适用
经过测试的设计流程 ³	
设计输入	Vivado Design Suite
仿真	如需了解受支持的仿真器的相关信息，请参阅 赛灵思设计工具：版本说明指南 。
综合	Vivado 综合
支持	
版本说明和已知问题	主答复记录：65751
所有 Vivado IP 变更日志	Vivado IP 主更改日志：72775
赛灵思技术支持网页	

注释：

- 如需获取受支持的器件的完整列表，请参阅 Vivado IP 目录。
- 资源使用情况数据适用于 PCIE4 块和 PCIE4C 块。
- 如需了解受支持的工具版本的相关信息，请参阅[赛灵思设计工具：版本说明指南](#)。

概述

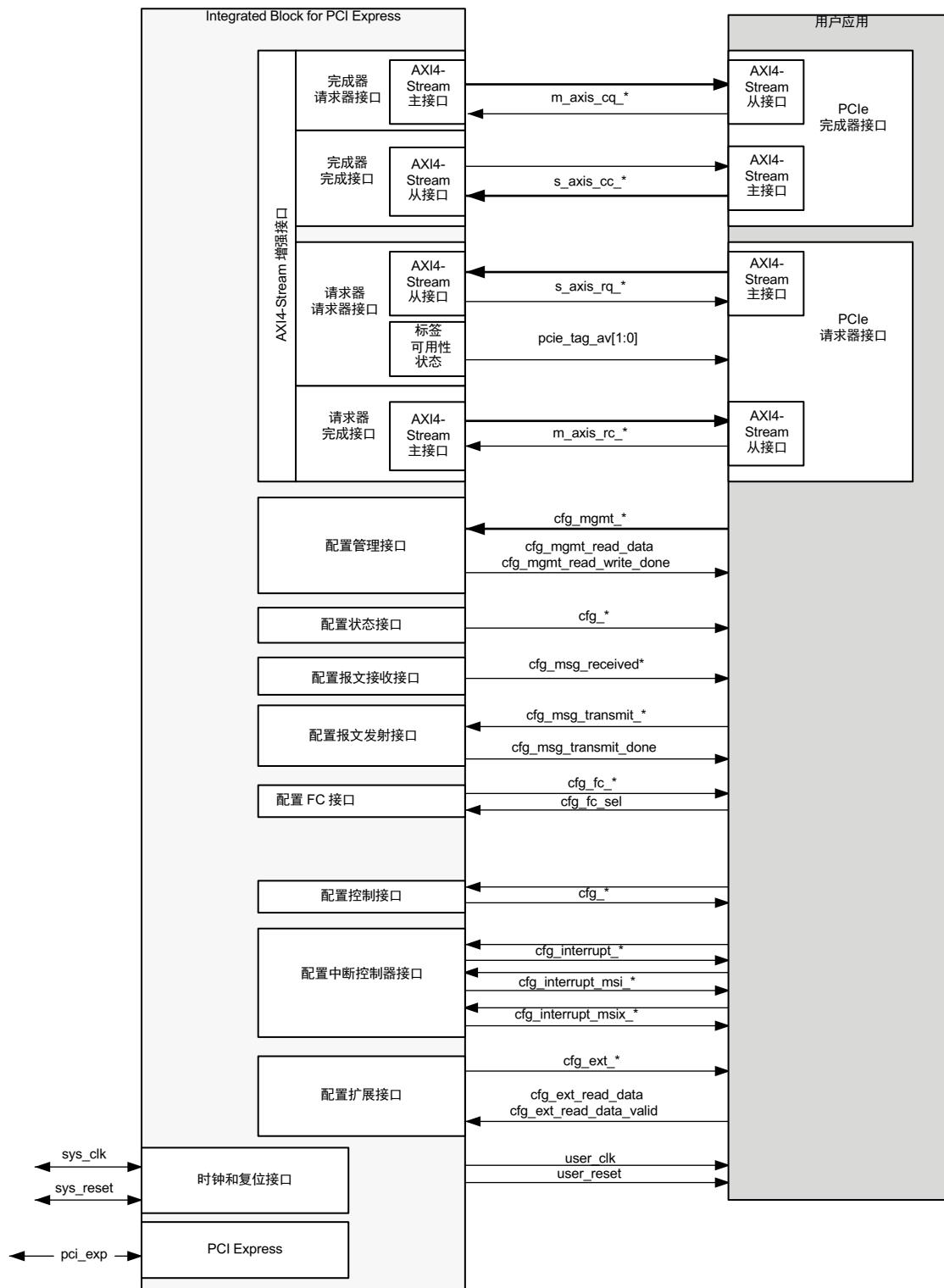
Integrated Block for PCIe® 核是高可靠性、高带宽、高可缩放性的串行互连构建块，适合与 UltraScale+™ 器件配合使用。该核可对 UltraScale+ 器件中提供的集成块进行例化。



重要提示！ 如需获取有关为 UltraScale™ 器件实现设计的详细信息，请参阅《UltraScale 器件 Gen3 Integrated Block for PCI Express LogiCORE IP 产品指南》(PG156)。

下图显示了核的接口。

图 1：核接口



X16318-091522

功能特性总结

PCIE4 和 PCIE4C 集成块解决方案中的 GTH 和 GTY 收发器支持 1 通道、2 通道、4 通道、8 通道和 16 通道操作，并支持以 2.5 GT/s (Gen1)、5.0 GT/s (Gen2) 和 8.0 GT/s (Gen3) 的线速运行。此外，PCIE4C 集成块支持以 16.0 GT/s (Gen4) 线速运行的 1 通道、2 通道、4 通道和 8 通道操作。端点配置和根端口配置也都受支持。

客户用户接口与 AMBA® AXI4-Stream 接口兼容。此接口支持独立的请求器接口、完成接口和报文接口。它支持灵活的数据对齐和奇偶校验检查。在接收方向和发射方向均支持数据流量控制。此外，发射方向还支持中断进行中的传输事务。可选连续传输事务可使用跨接来提供更大的链路带宽。

该核的详细特性包括：

- PCIE4 和 PCIE4C 块符合 [《PCI Express 基本规范第 3.1 版》](#) 的要求。
- PCI Express 端点、传统端点或根端口模式
- PCIE4 和 PCIE4C 块都支持：x1、x2、x4、x8 或 x16 链路宽度，并支持 Gen1、Gen2 和 Gen3 链路速度
- PCIE4C 块支持：x1、x2、x4 和 x8 链路宽度，并支持 Gen4 链路速度
- AXI4 串流接口，可连接到客户逻辑
 - 可配置的 64 位/128 位/256 位/512 位数据路径宽度
 - 4 个独立请求/完成串流
- 在内部逻辑数据路径和数据接口上提供极性保护
- 高级错误报告 (AER) 和端到端 CRC (ECRC)
- 1 个 PCI Express 虚拟通道，8 个流量类
- 支持多种功能和单根 I/O 虚拟化
 - 最多支持 4 项物理功能
 - 最多支持 252 项虚拟功能
- 内置通道翻转和接收器通道间去歪斜
- 完全可配置的 3 × 64 位或 6 × 32 位基址寄存器 (BAR)
 - 支持扩展 ROM BAR
- 最大有效载荷大小：128、256、512 和 1024 字节
- 支持所有中断类型：
 - INTx
 - 32 多矢量 MSI 功能
 - 含最多 2048 个矢量的 MSI-X 功能，具有可选内置 MSI-X 矢量表
- 内置发起方读取请求/完成标签管理器
 - 支持最多 256 项未完成的发起方读取请求传输事务
- 支持动态重配置端口 (DRP)

- 支持高性能应用的特性包括：
 - 请求器完成接口上支持 AXI4 串流事务层包 (TLP) 跨接
 - RX 完成报头信用值最高为 256，RX 完成有效载荷空间最高为 32 KB
 - 接收数据路径中的宽松传输事务排序
 - 地址转换服务 (ATS) 消息传递
 - 支持原子操作传输事务
 - TLP 处理提示 (TPH)
- 支持多项易用性和可配置性功能：
 - 基于 BAR 和 ID 的接收传输事务筛选
 - 可选 ASPM
 - 配置扩展接口
 - AXI4 串流接口地址对齐模式
 - 基于 PCI Express 媒体配置访问端口 (MCAP) 的配置，100 ms 配置上电（IP 核计划将于未来版本中提供此项支持）
 - 调试和诊断接口

相关信息

[不支持的功能](#)

应用

核架构支持各种计算和通信目标应用，并强调性能、成本、可缩放性、功能可缩放性以及对于任务至关重要的可靠性等因素。典型应用包括：

- 数据通信网络
- 远程通信网络
- 宽带有线和无线应用
- 网络接口卡
- 芯片到芯片 (Chip-to-Chip) 和背板接口卡
- 适用于各种应用的服务器插卡

不支持的功能

不受支持的 PCI Express 基本规范第 3.1 版功能

《PCI Express 基本规范第 3.1 版》具有诸多可选功能。以下列出了部分不受支持的功能：

- 无法实现地址转换服务，但允许在外部软核逻辑内实现此服务。
- 切换端口。
- 大小可调的 BAR 扩展功能。
- 基于 ID 的 TLP 排序。
- PCI Express 安全 IP 模型不支持对 DRP 接口进行仿真。
- 不支持串联配置（串联 PROM、串联 PCIe 和 DFX over PCIe）仿真；集成块的仿真模型不支持此类配置。
- 在根端口模式下，此 IP 并不会在内部基于“Bus Master Enable”（总线主控制器使能）寄存器位来实现 TLP 筛选。如果设计中需要此功能特性，那么您必须构建外部包筛选逻辑。

不受支持的 PCI Express 基本规范第 4.0 版功能 (PCIE4C)

PCIE4C (UltraScale+ HBM) 不支持以下 PCI Express 基本规范 4.0 的 1.0 版功能：

- 标签缩放 (10b 标签)
- DLLP 功能，流量控制缩放
- 重定时器存在位
- 通道裕度设置
- 轮询合规性状态以及合规性模式更改
- 链路扩展器件 (重定时器)



重要提示！ 虽然由于上述限制导致符合 4.0 标准的能力受到了限制，但仍能以 16.0 GT/s 速度进行互操作，并且对于大部分 4.0 器件都可行。请与已连接的 PCIe 器件的供应商协作，确保这些限制不会导致任何问题。

许可和订购

根据[赛灵思最终用户许可条款](#)，此 UltraScale+ 器件 Integrated Block for PCI Express 核随附于 Vivado Design Suite 免费提供。

如需获取有关此模块及其它赛灵思 LogiCORE™ IP 模块的信息，请访问[赛灵思知识产权](#)页面。如需了解有关其它赛灵思 LogiCORE IP 模块和工具的定价和可用性信息，请联系您当地的[赛灵思销售代表](#)。

产品规格

标准合规性

UltraScale+ 器件 Integrated Block for PCI Express 解决方案符合业界标准应用外形尺寸（如《PCI Express® 卡机电 (CEM) 规范》(v3.0) 和《PCIe® 工业计算机制造商集团 (PICMG)》3.4 版规范）的要求。

资源使用情况

如需获取有关性能与资源使用情况的完整详情，请访问[性能与资源使用情况网页](#)。

注释：上述链接中所提供的资源使用情况数据适用于 PCIE4 块和 PCIE4C 块。

器件最低要求

赛灵思在 UltraScale+™ 架构内提供了 2 个 PCIe® 集成块。在所有 UltraScale+ 器件内均包含 PCIE4 块，它可支持 PCIe IP。含高带宽存储器 (HBM) 的 Virtex UltraScale+ 器件同时包含 PCIE4 块和增强型 PCIE4C 块。PCIE4C 集成块的功能与 PCIE4 完全相同，同样支持 -2LV 速度等级的 PCIe Gen3 x16。

核的最低器件要求如下：

表 1：PCIE4 的器件最低要求

功能链路速度	链路宽度	受支持的速度等级
Gen1/Gen2	x16	-1、-1L、-1LV、-2、-2L、-2LV 和 -3
Gen3	x16	-1、-1L、-2、-2L 和 -3
	x8 NL ¹	-1、-1L、-1LV、-2、-2L、-2LV 和 -3
	x8 LL ²	-1、-1L、-2、-2L 和 -3

注释：

1. x8 NL = Gen3 x8 正常时延 (core_clock = 250 MHz)。
2. x8 LL = Gen3 x8 低时延 (core_clock = 500 MHz)。

表 2：PCIE4C 的器件最低要求

功能链路速度	链路宽度	受支持的速度等级
Gen1/Gen2	x16	-1、-2、-2L、-2LV 和 -3
Gen3	x16	-1、-2、-2L、-2LV 和 -3
	x8 NL	-1、-2、-2L、-2LV 和 -3
	x8 LL	-1、-2、-2L 和 -3
Gen4	x8	-2、-2L 和 -3

可供 PCI Express 使用的集成块

下表列出了可在包含多个集成块的器件中使用的 PCI Express® 集成块。在某些情况下，由于某些集成块旁缺少相邻的绑定 GTH 和 GTY 收发器 site，并非所有集成块都可供使用。

表 3：可供 PCI Express 使用的集成块 - Artix® UltraScale+™

器件选择		PCI Express 块位置		支持的链路	
器件	封装	PCIe 块	块类型	速度	宽度
XCAU25P	SFVB784	X0Y0	PCIE4	Gen1/2/3	最高 x8
	FFVB676	X0Y0	PCIE4	Gen1/2/3	最高 x8
XCAU20P	SFVB784	X0Y0	PCIE4	Gen1/2/3	最高 x8
	FFVB676	X0Y0	PCIE4	Gen1/2/3	最高 x8
XCAU15P	SBVB484	X0Y0	PCIE4C	Gen1/2/3	最高 x8
	UBVA368	X0Y0	PCIE4C	Gen1/2/3	最高 x8
	FFVB676	X0Y0	PCIE4C	Gen1/2/3/4	对于 Gen4，最高为 x4，其它则最高为 x8
XCAU10P	SBVB484	X0Y0	PCIE4C	Gen1/2/3	最高 x8
	UBVA368	X0Y0	PCIE4C	Gen1/2/3	最高 x8
	FFVB676	X0Y0	PCIE4C	Gen1/2/3/4	对于 Gen4，最高为 x4，其它则最高为 x8

表 4：可供 PCI Express 使用的集成块 - Kintex® UltraScale+™

器件选择		PCI Express 块位置						
器件	封装	X0Y0	X0Y1	X0Y2	X0Y3	X1Y0	X1Y1	X1Y2
XCKU11P	FFVE1517			可用	可用	可用	可用	
	FFVA1156			可用	可用	可用	可用	
	FFVD900					可用	可用	
XCKU15P	FFVE1517			可用	可用	可用	可用	可用
	FFVA1156			可用	可用	可用	可用	可用
	FFVA1760			可用	可用	可用	可用	可用
	FFVE1760			可用	可用	可用	可用	可用
XCKU19P	FFVB2104	可用	可用	可用				

表 4：可供 PCI Express 使用的集成块 - Kintex® UltraScale+™ (续)

器件选择		PCI Express 块位置							
器件	封装	X0Y0	X0Y1	X0Y2	X0Y3	X1Y0	X1Y1	X1Y2	
	FFVJ1760	可用	可用	可用					
XCKU3P	FFVA676	可用							
	FFVB676	可用							
	FFVD900	可用							
	SFVB784	可用							
XCKU5P	FFVA676	可用							
	FFVB676	可用							
	FFVD900	可用							
	SFVB784	可用							

表 5：可供 PCI Express 使用的集成块 - Virtex® UltraScale+™

器件选择		PCI Express 块位置									
器件	封装	X0Y0	X0Y1	X0Y2	X0Y3	X0Y4	X0Y5	X1Y0	X1Y1	X1Y2	X1Y4
XCVU3P	FFVC1517		可用					可用			
XCVU5P	FLVA2104		可用		可用			可用		可用	
	FLVB2104		可用		可用			可用		可用	
	FLVC2104		可用		可用			可用		可用	
XCVU7P	FLVA2104		可用		可用			可用		可用	
	FLVB2104		可用		可用			可用		可用	
	FLVC2104		可用		可用			可用		可用	
XCVU9P	FLGA2104		可用		可用					可用	可用
	FLGB2104		可用		可用					可用	可用
	FLGC2104		可用		可用		可用	可用		可用	可用
	FLGA2577		可用		可用		可用	可用		可用	可用
XCVU11P	FLGA2577	可用	可用	可用							
	FLGB2104	可用	可用	可用							
	FLGC2104	可用	可用	可用							
	FLGF1924	可用	可用	可用							
XCVU13P	FHGA2104		可用	可用							
	FHGB2014		可用	可用	可用						
	FHGC2104	可用	可用	可用	可用						
	FLGA2577	可用	可用	可用	可用						
	FIGD2104		可用	可用	可用						
	FSGA2577	可用	可用	可用	可用						
XCVU23P	FSVJ1760	可用	可用	可用	可用						
	VSVA1365	可用	可用	可用	可用						

表 5：可供 PCI Express 使用的集成块 - Virtex® UltraScale+™ (续)

器件选择		PCI Express 块位置									
器件	封装	X0Y0	X0Y1	X0Y2	X0Y3	X0Y4	X0Y5	X1Y0	X1Y1	X1Y2	X1Y4
XCVU29P	FIGD2104	可用									
	FSGA2577	可用									
XCVU31P ¹	FSVH1924	可用 ³	可用 ³					可用 ³	可用 ³		
XCVU33P ¹	FSVH2104	可用 ³	可用 ³					可用 ³	可用 ³		
XCVU35P ²	FSVH2892	可用 ^{3,4}	可用 ³					可用 ³	可用 ³		
	FSVH2104	可用 ^{3,4}	可用 ³					可用 ³	可用 ³		
XCVU37P ²	FSVH2892	可用 ^{3,4}	可用 ^{3,4}					可用 ³	可用 ³		
XCVU57P	FSVK2892	可用 ³	可用 ³					可用 ³	可用 ⁴		
XQVU7P	FLRA2104		可用		可用			可用		可用	
	FLRB2104		可用		可用			可用		可用	
XQVU11P	FLRC2104	可用	可用	可用							

注释：

1. 此类器件仅包含 PCIE4C 块。
2. 此类器件包含 PCIE4 块和 PCIE4C 块。
3. PCIE4C 块位置。
4. PCIE4 块位置。

表 6：可供 PCI Express 使用的集成块 - Zynq® UltraScale+™

器件选择		PCI Express 块位置						
器件	封装	X0Y0	X0Y1	X0Y2	X0Y3	X1Y0	X1Y1	X1Y2
XCZU11EG	FFVC1760			可用	可用	可用	可用	
	FFVB1517					可用	可用	
	FFVC1156					可用	可用	
	FFVF1517					可用	可用	
XCZU17EG	FFVC1760			可用	可用	可用	可用	可用
	FFVE1924					可用	可用	可用
	FFVB1517					可用	可用	
	FFVD1760			可用	可用	可用	可用	可用
XCZU19EG	FFVC1760			可用	可用	可用	可用	可用
	FFVE1924					可用	可用	可用
	FFVB1517					可用	可用	可用
	FFVD1760			可用	可用	可用	可用	可用
XQZU19EG	FFRB1517					可用	可用	可用
	FFRC1760			可用	可用	可用	可用	可用
XCZU4EV	FBVB900	可用	可用					
	SFVC784	可用	可用					
XCZU5EV	FBVB900	可用	可用					
	SFVC784	可用	可用					

表 6：可供 PCI Express 使用的集成块 - Zynq® UltraScale+™ (续)

器件选择		PCI Express 块位置						
器件	封装	X0Y0	X0Y1	X0Y2	X0Y3	X1Y0	X1Y1	X1Y2
XCZU7EV	FBVB900	可用	可用					
	FFVC1156	可用	可用					
	FFVF1517	可用	可用					
XCZU4CG	FBVB900	可用	可用					
	SFVC784	可用	可用					
XCZU5CG	FBVB900	可用	可用					
	SFVC784	可用	可用					
XCZU7CG	FBVB900	可用	可用					
	FFVC1156	可用	可用					
	FFVF1517	可用	可用					
XCZU4EG	FBVB900	可用	可用					
	SFVC784	可用	可用					
XCZU5EG	FBVB900	可用	可用					
	SFVC784	可用	可用					
XCZU7EG	FBVB900	可用	可用					
	FFVC1156	可用	可用					
	FFVF1517	可用	可用					
XCZU21DR	FFVD1156	可用	可用					
XCZU25DR	FFVE1156	可用						
	FFVG1517	可用						
	FSVE1156	可用						
	FSVG1517	可用						
XCZU28DR	FFVE1156	可用	可用					
	FFVG1517	可用	可用					
	FSVE1156	可用	可用					
	FSVG1517	可用	可用					
XCZU29DR	FFVF1760	可用	可用					
	FSVF1760	可用	可用					
XCZU27DR	FFVE1156	可用	可用					
	FFVG1517	可用	可用					
	FSVE1156	可用	可用					
	FSVG1517	可用	可用					
XAZU4EV	SFVC784	可用	可用					
XAZU5EV	SFVC784	可用	可用					
XQZU5EV	FFRB900	可用	可用					
	SFRC784	可用	可用					
XQZU7EV	FFRB900	可用	可用					
	FFRC1156	可用	可用					

表 6：可供 PCI Express 使用的集成块 - Zynq® UltraScale+™ (续)

器件选择		PCI Express 块位置						
器件	封装	X0Y0	X0Y1	X0Y2	X0Y3	X1Y0	X1Y1	X1Y2
XQZU21DR	FFRD1156	可用	可用					
XQZU28DR	FFRE1156	可用	可用					
	FFRG1517	可用	可用					

GT 位置

附录 C: GT 位置 提供了可供此 IP 核使用的 GT 位置列表，并列出了选择 GT 位置时应予以考量的部分关键建议。

端口描述

本节提供了下列接口的详细端口描述：

- AXI4-Stream 核接口
- 其它核接口

AXI4-Stream 核接口

64/128/256 位接口

除了状态和控制接口，核还有 4 个必需的 AXI4-Stream 接口，用于发射和接收传输事务，本节将对此作详细描述。

相关信息

512 位接口

完成器请求接口

通过“Completer Request (CQ)”（完成器请求）接口所含端口即可将从链路接收到的所有请求交付到用户应用。下表定义了核的 CQ 接口中的端口。在“宽度”列中，DW 表示已配置的数据总线宽度（64、128 或 256 位）。

表 7：完成器请求接口端口描述

端口	I/O	宽度	描述
m_axis_cq_tdata	输出	DW	从 CQ 接口发射数据。 当接口宽度为 128 位时，仅使用下 128 位，当接口宽度为 64 位时，则仅使用下 64 位。 当接口宽度配置为 128 位时，核会将位 [255:128] 永久设置为 0，而当接口宽度配置为 64 位时，则会将位 [255:64] 永久设置为 0。

表 7：完成器请求接口端口描述 (续)

端口	I/O	宽度	描述
m_axis_cq_tuser	输出	88	CQ 用户数据。 这组信号中包含所传输的传输事务层包 (TLP) 的边带信息。当 m_axis_cq_tvalid 为高电平时，这些信号有效。 表 8: m_axis_cq_tuser 中的边带信号描述 描述了这组信号中的每个信号。
m_axis_cq_tlast	输出	1	CQ 数据的 TLAST 指示。 核在包的最后一个节拍内断言此信号有效以指示包结束。在单一节拍内完成 TLP 传输时，核会在传输的首个节拍内设置该信号。
m_axis_cq_tkeep	输出	DW/32	CQ 数据的 TKEEP 指示。 传输期间断言此总线的位 i 有效对于用户应用而言，表示 m_axis_cq_tdata 总线的 Dword i 包含的数据有效。核会针对从描述符的首个 Dword 开始到有效载荷的最后一个 Dword 为止的所有 Dword 将该位连续设置为 1。因此，在包的所有节拍中 m_axis_cq_tdata 必须全部设置为 1，但在包的总大小并非数据总线宽度的整数倍（在 2 个 Dword 内都是如此）的情况下，最后一个节拍除外。对于有效载荷传输的 Dword 对齐模式和地址对齐模式都是如此。 当接口宽度配置为 128 位时，核会将该总线的位 [7:4] 永久设置为 0，而当接口宽度配置为 64 位时，则会将位 [7:2] 永久设置为 0。
m_axis_cq_tvalid	输出	1	CQ 数据有效。 核在 m_axis_cq_tdata 总线上驱动有效数据时始终断言此输出有效。核会在包传输期间使有效信号保持处于断言有效状态。用户应用可以使用 m_axis_cq_tready 信号调整数据传输节奏。
m_axis_cq_tready	输入	1	CQ 数据就绪。 用户逻辑激活此信号对于核而言，表明用户应用已准备好接受数据。在同一周期内 m_axis_cq_tvalid 和 m_axis_cq_tready 均断言有效时，就会通过该接口传输数据。 当 m_axis_cq_tvalid 为高电平时，如果用户应用断言就绪信号无效，那么核会在总线上保留数据，并使有效信号保持断言有效状态，直至用户应用断言就绪信号有效为止。
pcie_cq_np_req	输入	2	此输入供用户应用用于请求交付非转发请求。核通过实现基于信用值的流量控制机制来控制通过该接口交付非转发请求的过程，并避免阻塞转发 TLP。 此核输入可控制内部信用值。在每个时钟周期内基于如下 pcie_cq_np_req[1:0] 设置来更新信用值： <ul style="list-style-type: none">· 00：无更改· 01：递增 1· 10 或 11：保留（位 [1] 仅适用于 512 位接口） 在通过接口交付每个非转发请求后，信用值会递减。当信用值达到 0 时，核会暂时停止向用户逻辑交付非转发请求。即使已暂停交付非转发请求，核仍会继续交付从链路接收到的任何转发 TLP。 用户应用可以在每个周期内根据其非转发请求接收缓冲器的状态来设置 pcie_cq_np_req[1:0]，或者如果不需对非转发请求运用选择性反压，则可将其永久设置为 11。 pcie_cq_np_req[1:0] 的设置无需与完成器请求接口上的包传输对齐。

表 7：完成器请求接口端口描述 (续)

端口	I/O	宽度	描述
pcie_cq_np_req_count	输出	6	<p>此输出所提供的值是核为了向用户逻辑交付非转发请求而保留的信用值的当前值。仅当此信用值为非 0 值时，核才能通过完成器请求接口交付非转发请求。该信用值饱和上限为 32。</p> <p>由于存在内部流水线延迟，当用户应用在 pcie_cq_np_req[1:0] 输入上提供信用值后，可能需要经过多个延迟周期后，PCIe 核才会更新 pcie_cq_np_req_count 输出响应。</p> <p>此计数将于执行 user_reset 以及断言 user_lnk_up 无效时复位。</p>

表 8：m_axis_cq_tuser 中的边带信号描述

位索引	名称	宽度	描述
3:0	first_be[3:0]	4	<p>有效载荷的首个 Dword 的字节使能。</p> <p>该字段可反映 TLP 的传输事务层报头中的 First_BE 位的设置。对于存储器读取和 I/O 读取，这 4 个位用于表示首个 Dword 中将读取的有效字节。对于存储器写入和 I/O 写入，这些位表示有效载荷的首个 Dword 中的有效字节。对于原子操作和含有效载荷的报文，这些位全部设置为 1。</p> <p>该字段在包的第一拍中有效。即当 sop 和 m_axis_cq_tvalid 均为高电平时有效。</p>
7:4	last_be[3:0]	4	<p>最后一个 Dword 的字节使能。</p> <p>该字段可反映 TLP 的传输事务层报头中的 Last_BE 位的设置。对于存储器读取，这 4 个位表示在数据块的最后一个 Dword 中要读取的有效字节。对于存储器写入，这些位表示有效载荷的最后一个 Dword 中的有效字节。对于原子操作和含有效载荷的报文，这些位全部设置为 1。对于仅含单次 DW 传输且传输长度为 0 的存储器读取和写入，这些位应为 0。</p> <p>该字段在包的第一拍中有效。即当 sop 和 m_axis_cq_tvalid 均为高电平时有效。</p>
39:8	byte_en[31:0]	32	<p>用户逻辑可以选择使用这些字节使能位来判定所传输的包的有效载荷中的有效字节。传输期间断言此总线的位 i 有效表明 m_axis_cq_tdata 总线的字节 i 包含的有效载荷字节是有效的。针对描述符字节，不会断言该位有效。</p> <p>虽然字节使能可由用户逻辑根据请求描述符中的信息（地址和长度）来生成，也可以根据 first_be 信号和 last_be 信号的设置来生成，但您还可选择直接使用这些信号，而不必从其它接口信号生成这些信号。</p> <p>当有效载荷的大小超过 2 个 Dword (8 字节) 时，该总线上对应有效载荷的值为 1 的位始终保持连续。当有效载荷大小不超过 2 个 Dword 时，值为 1 的位可能不连续。</p> <p>对于 PCI Express 规范所定义的存储器写入传输事务长度为 0 的特殊情况，传输关联的 1 DW 有效载荷时，byte_en 位全部为 0。</p> <p>当接口宽度配置为 128 位时，核会将该总线的位 [31:16] 永久设置为 0，而当接口宽度配置为 64 位时，则会将位 [31:8] 永久设置为 0。</p>
40	sop	1	<p>包起始。</p> <p>核在包的第一拍中断言此信号有效以指示包开始。此信号的使用为可选。</p>

表 8：m_axis_cq_tuser 中的边带信号描述 (续)

位索引	名称	宽度	描述
41	discontinue	1	如果核在从其内部 FIFO 存储器读取 TLP 有效载荷时检测到不可纠正的错误，那么它会在 TLP 的最后一个节拍中断言此信号有效。当核发出此类错误信号时，用户应用必须丢弃整个 TLP。 当 TLP 不含有效载荷时，永远不会断言此信号有效。仅在 m_axis_cq_tlast 为高电平的周期内才断言此信号有效。 当核配置为端点时，核也会使用高级错误报告 (AER) 向所连接到的根联合体报告此错误。
42	tph_present	1	此位表示在通过接口交付的请求 TLP 中存在“Transaction Processing Hint (TPH)”（传输事务处理提示）。当 sop 和 m_axis_cq_tvalid 均为高电平时，该位有效。
44:43	tph_type[1:0]	2	当请求 TLP 中存在 TPH 时，这 2 个位可提供与提示关联的 PH[1:0] 字段的值。当 sop 和 m_axis_cq_tvalid 均为高电平时，这些位才有效。
52:45	tph_st_tag[7:0]	8	当请求 TLP 中存在 TPH 时，此输出可提供与提示关联的 8 位导向标签。当 sop 和 m_axis_cq_tvalid 均为高电平时，这些位才有效。
84:53	parity	32	位 i 可提供针对 m_axis_cq_tdata 的字节 i 计算所得奇校验。当接口宽度为 128 位时，仅使用下 16 位，当接口宽度为 64 位时，则仅使用下 8 位。当接口宽度配置为 128 位时，核会将位 [31:16] 永久设置为 0，而当接口宽度配置为 64 位时，则会将位 [31:8] 永久设置为 0。
87:85	保留	3	保留以供 512 位接口使用。

注释：当 axis ready 和 axis valid 均处于高位时，节拍采用总线传输。

完成器完成接口

用户应用生成的完成包通过完成器完成 (CC) 接口端口来响应发射的完成器请求。您可将所有非转发传输事务作为独立拆分的传输事务来进行处理。即，CC 接口可持续接受请求器完成接口上的新请求，同时针对请求发送完成 (completion) 包。下表定义了核的 CC 接口中的端口。在“宽度”列中，DW 表示已配置的数据总线宽度 (64、128 或 256 位)。

表 9：完成器完成接口端口描述

端口	I/O	宽度	描述
s_axis_cc_tdata	输入	DW	完成器完成数据总线。 从用户应用到核的完成数据。当接口宽度为 128 位时，仅使用下 128 位，当接口宽度为 64 位时，则仅使用下 64 位。
s_axis_cc_tuser	输入	33	完成器完成用户数据。 这组信号中包含所传输的 TLP 的边带信息。当 s_axis_cc_tvalid 为高电平时，这些信号有效。 下表描述了这组信号中的每个信号。
s_axis_cc_tlast	输入	1	对应完成器完成数据的 TLAST 指示。 用户应用必须在包的最后一个周期内断言此信号有效以指示包结束。在单一节拍 (beat) 内完成 TLP 传输时，用户应用必须在传输的首个周期内设置该位。

表 9：完成器完成接口端口描述 (续)

端口	I/O	宽度	描述
s_axis_cc_tkeep	输入	DW/32	<p>对应完成器完成数据的 TKEEP 指示。</p> <p>传输期间断言此总线的位 i 有效对于核而言，表示 s_axis_cc_tdata 总线的 Dword i 包含的数据有效。针对从描述符的首个 Dword 开始到有效载荷的最后一个 Dword 为止的所有 Dword 将该位连续设置为 1。因此，在包的所有节拍中，s_axis_cc_tdata 必须全部设置为 1，但在包的总大小并非数据总线宽度的整数倍（在 2 个 Dword 内都是如此）的情况下，最后一个节拍除外。对于有效载荷传输的 Dword 对齐模式和地址对齐模式都是如此。</p> <p>当接口宽度配置为 128 位时，核不使用该总线的位 [7:4]；当接口宽度配置为 64 位时，不使用位 [7:2]。</p>
s_axis_cc_tvalid	输入	1	<p>完成器完成数据有效。</p> <p>用户应用在 s_axis_cc_tdata 总线上驱动有效数据时必须始终断言此输出有效。用户应用必须在包传输期间使有效信号保持处于断言有效状态。该核会使用 s_axis_cc_tready 信号来调整数据传输节奏。</p>
s_axis_cc_tready	输出	4	<p>完成器完成数据就绪。</p> <p>核激活此信号表明它已准备好接受数据。在同一周期内 s_axis_cc_tvalid 和 s_axis_cc_tready 均断言有效时，就会通过该接口传输数据。</p> <p>当有效信号为高电平时，如果核断言就绪信号无效，那么用户应用必须在总线上保留数据，并使有效信号保持处于断言有效状态，直至该核断言就绪信号有效为止。</p> <p>对于此输出端口，每个位均指示相同的值，因此用户逻辑可以使用任何位。</p>

表 10：s_axis_cc_tuser 中的边带信号描述

位索引	名称	宽度	描述
0	discontinue	1	<p>如果传输期间用户应用在所传输的数据中检测到错误（例如，读取来自存储器的有效载荷时检测到不可纠正 ECC 错误）并且需要中止该数据包，即可断言此信号有效。核会将链路上对应 TLP 置空，以避免数据损坏。</p> <p>在传输期间，用户应用可在任意周期内断言此信号有效。它可以选择在周期内发出错误信号处提前终止该数据包，或者也可以继续处理，直至将有效载荷的所有字节都交付到核为止。针对后者，该核会针对数据包的后续节拍将此错误作为粘滞错误来处理，即使用户应用包结束前断言 discontinue 信号无效也是如此。</p> <p>仅当 s_axis_cc_tvalid 为高电平时，才能断言 discontinue 信号有效。仅当 s_axis_cc_tready 为高电平时，核才会对此信号进行采样。因此，一旦断言有效后，在 s_axis_cc_tready 变为高电平之前不应将其断言无效。</p> <p>当核配置为端点时，核也会使用 AER 向所连接到的根联合体报告此错误。</p>
32:1	Parity	32	<p>256 位数据的奇校验。</p> <p>在核上启用奇偶校验检查时，用户逻辑必须将该逻辑的位 i 设置为针对 s_axis_cc_tdata 的字节 i 计算所得的奇校验。当接口宽度为 128 位时，仅使用下 16 位，当接口宽度为 64 位时，则仅使用下 8 位。</p> <p>当检测到接口奇偶校验错误时，会将其记录为不可纠正的内部错误，并丢弃该数据包。根据《基本规范 6.2.9》，不可纠正的内部错误即组件内发生的导致组件无法正常工作的错误。从不可纠正的内部错误恢复的唯一方法是复位或更换硬件。</p> <p>如果在核中未启用奇偶校验检查，那么奇偶校验位可永久绑定到 0。</p>

请求器请求接口

用户应用通过“Requester Request (RQ)”（请求器请求）接口所包含的接口生成请求并发送给远程 PCIe® 器件。下表定义了核的 RQ 接口中的端口。在“宽度”列中，DW 表示已配置的数据总线宽度（64、128 或 256 位）。

表 11：请求器请求接口端口描述

端口	I/O	宽度	描述
s_axis_rq_tdata	输入	DW	请求器请求数据总线。 此输入包含从用户应用到核的请求器侧请求数据。当接口宽度为 128 位时，仅使用下 128 位，当接口宽度为 64 位时，则仅使用下 64 位。
s_axis_rq_tuser	输入	62	请求器请求用户数据。 这组信号中包含所传输的 TLP 的边带信息。当 s_axis_rq_tvalid 为高电平时，这些信号有效。下表描述了这组信号中的每个信号。
s_axis_rq_tlast	输入	1	对应请求器请求数据的 TLAST 指示。 用户应用必须在 TLP 的最后一个周期内断言此信号有效以指示包结束。在单一节拍 (beat) 内完成 TLP 传输时，用户应用必须在传输的首个周期内设置该位。
s_axis_rq_tkeep	输入	DW/32	对应请求器请求数据的 TKEEP 指示。 传输期间断言此总线的位 i 有效对于核而言，表示 s_axis_rq_tdata 总线的 Dword i 包含的数据有效。用户应用必须针对从描述符的首个 Dword 开始到有效载荷的最后一个 Dword 为止的所有 Dword 将该位连续设置为 1。因此，在包的所有节拍中，s_axis_rq_tkeep 必须全部设置为 1，但在包的总大小并非数据总线宽度的整数倍（在 2 个 Dword 内都是如此）的情况下，最后一个节拍除外。对于有效载荷传输的 Dword 对齐模式和地址对齐模式都是如此。 当接口宽度配置为 128 位时，核不使用该总线的位 [7:4]；当接口宽度配置为 64 位时，不使用位 [7:2]。
s_axis_rq_tvalid	输入	1	请求器请求数据有效。 用户应用在 s_axis_rq_tdata 总线上驱动有效数据时必须始终断言此输出有效。用户应用必须在包传输期间使有效信号保持处于断言有效状态。该核会使用 s_axis_rq_tready 信号来调整数据传输节奏。
s_axis_rq_tready	输出	4	请求器请求数据就绪。 核激活此信号表明它已准备好接受数据。在同一周期内 s_axis_rq_tvalid 和 s_axis_rq_tready 均断言有效时，就会通过该接口传输数据。 当有效信号为高电平时，如果核断言就绪信号无效，那么用户应用必须在总线上保留数据，并使有效信号保持处于断言有效状态，直至该核断言就绪信号有效为止。 对于此输出端口，每个位均指示相同的值，因此用户逻辑可以使用任何位。
pcie_rq_seq_num0	输出	6	请求器请求 TLP 发射序列号。 您可选择使用此输出在核的发射流水线中保留请求的进展情况记录。要使用此功能，请为 seq_num[3:0] 总线上每个请求提供 1 个序列号。当流水线中的请求 TLP 进展至特定状态（即，来自用户应用的完成 TLP 都无法继续传递该包）时，核会在 pcie_rq_seq_num0[3:0] 输出上输出此序列号。此机制支持您在发送到核的 CC 接口的完成包与发送到请求器请求接口的转发请求之间保留顺序不变。当 pcie_rq_seq_num0[3:0] 为高电平时，pcie_rq_seq_num0[3:0] 输出上的数据有效。

表 11：请求器请求接口端口描述 (续)

端口	I/O	宽度	描述
pcie_rq_seq_num_vld0	输出	1	请求器请求 TLP 发射序列号有效。 当此输出将有效数据置于 pcie_rq_seq_num0[3:0] 上时，核会断言此输出有效，并保持 1 个周期。
pcie_rq_tag0 pcie_rq_tag1	输出	10	请求器请求非转发标签。 当由核执行非转发请求的标签管理操作时 (AXISTEN_IF_ENABLE_CLIENT_TAG 为 0)，此输出供该核用于传达为接收到的每个非转发请求所分配的标签。 当 pcie_rq_tag_vld0 为高电平时，该总线上的标签值在 1 个周期内有效。您必须复制此标签并将其用于将完成数据与暂挂请求相关联。 在 s_axis_rq_tdata 总线上传输请求后，可能要经过数个周期的延迟之后，核才会断言 pcie_rq_tag_vld0 有效以便为请求提供已分配的标签。与此同时，用户应用可以继续发送新请求。在此总线上，请求的标签按 FIFO 顺序来传递，这样用户应用即可轻松将标签值与其传输的请求相关联。
pcie_rq_tag_vld0 pcie_rq_tag_vld1	输出	1	请求器请求非转发标签有效。 当该核将标签分配给从请求器请求接口传入的非转发请求，并将此标签置于 pcie_rq_tag0 输出上时，就会断言此输出有效，并保持 1 个周期。

表 12：s_axis_rq_tuser 中的边带信号描述

位索引	名称	宽度	描述
3:0	first_be[3:0]	4	首个 Dword 的字节使能。 该字段必须基于请求 TLP 的传输事务层报头中的 First_BE 位的期望值来设置。对于存储器读取、I/O 读取和配置读取，这 4 个位表示首个 Dword 中将读取的有效字节。对于存储器写入、I/O 写入和配置写入，这些位表示有效载荷的首个 Dword 中的有效字节。 当 s_axis_rq_tvalid 和 s_axis_rq_tready 均为高电平时，核会在包的第一拍中对该字段进行采样。
7:4	last_be[3:0]	4	最后一个 Dword 的字节使能。 该字段必须基于 TLP 的传输事务层报头中的 Last_BE 位的期望值来设置。对于不少于 2 个 Dword 的存储器读取，这 4 个位表示在数据块的最后一个 Dword 中要读取的有效字节。对于仅含单次 DW 传输且传输长度为 0 的存储器读取和写入，这些位应为 0。对于不少于 2 个 Dword 的存储器写入，这些位表示有效载荷的最后一个 Dword 中的有效字节。 当 s_axis_rq_tvalid 和 s_axis_rq_tready 均为高电平时，核会在包的第一拍中对该字段进行采样。
10:8	addr_offset[2:0]	3	当在此接口中使用地址对齐模式时，用户应用必须提供字节通道编号，其中有效载荷数据在此边带总线的数据总线上开始（取模数 4）。这样，核即可判定所传输的数据块的对齐方式。 当 s_axis_rq_tvalid 和 s_axis_rq_tready 均为高电平时，核会在包的第一拍中对该字段进行采样。 当请求器请求接口采用 Dword 对齐模式配置时，该字段必须始终设置为 0。 在根端口配置下，配置包必须始终对齐到 DW0，因此，对于此类型的包，该字段在两种对齐方式下都必须设置为 0。

表 12：s_axis_rq_tuser 中的边带信号描述 (续)

位索引	名称	宽度	描述
11	discontinue	1	<p>如果传输期间用户应用在所传输的数据中检测到错误并且需要中止该数据包，即可断言此信号有效。核会将链路上对应 TLP 置空，以避免数据损坏。</p> <p>在传输期间，您可在任意周期内断言此信号有效。您可以选择在周期内发出错误信号处提前终止该数据包，或者也可以继续处理，直至将有效载荷的所有字节都交付到核为止。针对后者，该核会针对数据包的后续节拍将此错误作为粘滞错误来处理，即使用户应用包结束前断言 discontinue 信号无效也是如此。</p> <p>仅当 s_axis_rq_tvalid 为高电平时，才能断言 discontinue 信号有效。仅当 s_axis_rq_tready 为高电平时，核才会对此信号进行采样。因此，一旦断言有效后，在 s_axis_rq_tready 变为高电平之前不应将其断言无效。针对非转发 TLP 不支持 discontinue 信号。在传输期间，用户逻辑可在除首个周期外的任意周期内断言此信号有效。</p> <p>当核配置为端点时，核也会使用高级错误报告 (AER) 向所连接到的根联合体报告此错误。</p>
12	tph_present	1	<p>此位表示在通过接口交付的请求 TLP 中存在“Transaction Processing Hint (TPH)”（传输事务处理提示）。当 s_axis_rq_tvalid 和 s_axis_rq_tready 均为高电平时，核会在包的第一拍中对该字段进行采样。</p> <p>如果不使用 TPH 功能，那么该位必须永久绑定到 0。</p>
14:13	tph_type[1:0]	2	<p>当请求 TLP 中存在 TPH 时，这 2 个位可提供与提示关联的 PH[1:0] 字段的值。当 s_axis_rq_tvalid 和 s_axis_rq_tready 均为高电平时，核会在包的第一拍中对该字段进行采样。</p> <p>如果 tph_present 设置为 0，那么这些位可设置为任意值。</p>
15	tph_indirect_tag_en	1	<p>设置该位后，核会使用 tph_st_tag[7:0] 的较低的位，作为其“Steering Tag Table”（导向标签表）中的索引，并将来自该位置的标签插入发射的请求 TLP 中。</p> <p>当该位设置为 0 时，核直接使用 tph_st_tag[7:0] 上的值作为导向标签。</p> <p>当 s_axis_rq_tvalid 和 s_axis_rq_tready 均为高电平时，核会在包的第一拍中对该位进行采样。</p> <p>如果 tph_present 设置为 0，那么该位可设置为任意值。</p>
23:16	tph_st_tag[7:0]	8	<p>当请求 TLP 中存在 TPH 时，此输出可提供与提示关联的 8 位导向标签。当 s_axis_rq_tvalid 和 s_axis_rq_tready 均为高电平时，核会在包的第一拍中对该字段进行采样。</p> <p>如果 tph_present 设置为 0，那么这些位可设置为任意值。</p>
27:24	seq_num[3:0]	4	<p>您可以选择在此字段中提供 4 位序列号，以便在核的发射流水线中保留请求的进展情况记录。当流水线中的请求 TLP 进展至特定状态（即，完成 TLP 无法继续传递该包）时，该核会在其 pcie_rq_seq_num[3:0] 输出上输出此序列号。</p> <p>当 s_axis_rq_tvalid 和 s_axis_rq_tready 均为高电平时，核会在包的第一拍中对该字段进行采样。</p> <p>当用户应用未监控核的 pcie_rq_seq_num[3:0] 输出时，此输入可硬连线到 0。</p>

表 12：s_axis_rq_tuser 中的边带信号描述 (续)

位索引	名称	宽度	描述
59:28	parity	32	<p>256 位数据的奇校验。</p> <p>在核上启用奇偶校验检查时，用户逻辑必须将该逻辑的位 i 设置为针对 s_axis_rq_tdata 的字节 i 计算所得的奇校验。当接口宽度为 128 位时，仅使用下 16 位，当接口宽度为 64 位时，则仅使用下 8 位。</p> <p>当检测到接口奇偶校验错误时，会将其记录为不可纠正的内部错误，并丢弃该数据包。根据《基本规范 6.2.9》（《PCI-SIG 规范》(https://www.pcisig.com/specifications)），不可纠正的内部错误即组件内发生的导致组件无法正常工作的错误。从不可纠正的内部错误恢复的唯一方法是复位或更换硬件。</p> <p>如果在核中未启用奇偶校验检查，那么奇偶校验位可永久绑定到 0。</p>
61:60	seq_num[5:4]	2	seq_num 的扩展，与 [27:24] 相同。

请求器完成接口

通过“Requester Completion (RC)”（请求器完成）接口所含端口可将从链路接收到的完成包（作为对您的请求的响应）提供给用户应用。下表定义了核的 RC 接口中的端口。在“宽度”列中，DW 表示已配置的数据总线宽度（64、128 或 256 位）。

表 13：请求器完成接口端口描述

端口	I/O	宽度	描述
m_axis_rc_tdata	输出	DW	<p>请求器完成数据总线。</p> <p>从核请求器完成接口向用户应用发射数据。当接口宽度为 128 位时，仅使用下 128 位，当接口宽度为 64 位时，则仅使用下 64 位。</p> <p>当接口宽度配置为 128 位时，核会将位 [255:128] 永久设置为 0，而当接口宽度配置为 64 位时，则会将位 [255:64] 永久设置为 0。</p>
m_axis_rc_tuser	输出	75	<p>请求器完成用户数据。</p> <p>这组信号中包含所传输的 TLP 的边带信息。当 m_axis_rc_tvalid 为高电平时，这些信号有效。</p> <p>下表描述了这组信号中的每个信号。</p>
m_axis_rc_tlast	输出	1	<p>对应请求器完成数据的 TLAST 指示。</p> <p>核在包的最后一个节拍内断言此信号有效以指示包结束。在单一节拍内完成 TLP 传输时，核会在传输的首个节拍内设置该位。仅当禁用跨接选项时，才使用此输出。针对 256 位接口启用跨接选项时，核会将该输出永久设置为 0。</p>
m_axis_rc_tkeep	输出	DW/32	<p>对应请求器完成数据的 TKEEP 指示。</p> <p>传输期间断言此总线的位 i 有效即表示 m_axis_rc_tdata 总线的 Dword i 包含的数据有效。核会针对从描述符的首个 Dword 开始到有效载荷的最后一个 Dword 为止的所有 Dword 将该位连续设置为 1。因此，在包的所有节拍中 m_axis_rc_tkeep 必须设置为 1，但在包的总大小并非数据总线宽度的整数倍（在 2 个 Dword 内都是如此）的情况下，最后一个节拍除外。对于有效载荷传输的 Dword 对齐模式和地址对齐模式都是如此。</p> <p>当接口宽度配置为 128 位时，核会将该总线的位 [7:4] 永久设置为 0，而当接口宽度配置为 64 位时，则会将位 [7:2] 永久设置为 0。</p> <p>当接口宽度为 256 位并启用跨接选项时，这些输出全部永久设置为 1。在此情况下，用户逻辑必须使用 m_axis_rc_tuser 中的信号，以判定通过该接口传输的完成 TLP 的起始和结束。</p>

表 13：请求器完成接口端口描述 (续)

端口	I/O	宽度	描述
m_axis_rc_tvalid	输出	1	请求器完成数据有效。 核在 m_axis_rc_tdata 总线上驱动有效数据时始终断言此输出有效。核会在包传输期间使有效信号保持处于断言有效状态。核会使用 m_axis_rc_tready 信号来调整数据传输节奏。
m_axis_rc_tready	输入	1	请求器完成数据就绪。 用户逻辑激活此信号对于核而言，表明用户应用已准备好接受数据。在同一周期内 m_axis_rc_tvalid 和 m_axis_rc_tready 均断言有效时，就会通过该接口传输数据。 当有效信号为高电平时，如果用户应用断言就绪信号无效，那么核会在总线上保留数据，并使有效信号保持断言有效状态，直至用户应用断言就绪信号有效为止。

表 14：m_axis_rc_tuser 中的边带信号描述

位索引	名称	宽度	描述
31:0	byte_en	32	用户逻辑可以选择使用这些字节使能位来判定所传输的包的有效载荷中的有效字节。传输期间断言此总线的位 i 有效表明 m_axis_rc_tdata 总线的字节 i 包含的有效载荷字节是有效的。针对描述符字节，不会断言该位有效。 虽然字节使能位可由用户逻辑根据请求描述符中的信息（地址和长度）来生成，但用户逻辑还可以选择直接使用这些信号，而不必从其它接口信号生成这些信号。该总线中对应 TLP 有效载荷的值为 1 的位始终连续。 当接口宽度配置为 128 位时，核会将该总线的位 [31:16] 永久设置为 0，而当接口宽度配置为 64 位时，则会将位 [31:8] 永久设置为 0。字节使能位同样在收到完成包（作为对长度为 0 的存储器读取请求的响应）时设置。
32	is_sof_0	1	首个完成 TLP 的起始位置。 对于 64 位接口和 128 位接口，以及无跨接的 256 位接口，核会在包的第一拍中断言 is_sof_0 有效以指示 TLP 起始。在这些接口上，每个数据节拍中只能启动单个 TLP，并且 is_sof_1 永久设置为 0。不启用跨接选项时，此信号的使用为可选。 当接口宽度为 256 位并启用跨接选项时，核可在同一拍内跨接 2 个完成 TLP。在此情况下，完成 TLP 不会被格式化为 AXI4-Stream 包。断言 is_sof_0 有效表明完成 TLP 在此节拍中开始。如果前一个 TLP 在此节拍前结束，那么此完成 TLP 的第 1 个字节位于字节通道 0 中，或者如果前一个 TLP 在此节拍中继续，那么该字节位于字节通道 16 中。
33	is_sof_1	1	当接口宽度为 256 位并启用跨接选项时，将使用此信号，在此情况下，核可在同一拍内跨接 2 个完成 TLP。在所有其它情况下，输出永久设置为 0。 断言 is_sof_1 有效表明第 2 个完成 TLP 在此节拍中开始，其第 1 个字节位于字节通道 16 中。仅当前一个 TLP 在同一拍内在字节位置 0-15 中的任一位置结束时，核才会在字节位置 16 开始第 2 个 TLP；即仅当 is_eof_0[0] 也在同一拍内设置时才会如此。
37:34	is_eof_0[3:0]	4	第 1 个完成 TLP 的结束位置，及其最后一个 Dword 的偏移。 仅当接口宽度为 256 位并启用跨接选项时，才会使用这些输出。 断言 bit is_eof_0[0] 有效表明第 1 个完成 TLP 在当前节拍内结束。设置此位时，位 is_eof_0[3:1] 可提供该 TLP 的最后一个 Dword 的偏移。

表 14: m_axis_rc_tuser 中的边带信号描述 (续)

位索引	名称	宽度	描述
41:38	is_eof_1[3:0]	4	<p>第 2 个完成 TLP 的结束位置，及其最后一个 Dword 的偏移。仅当接口宽度为 256 位并启用跨接选项时，才会使用这些输出。随后，核将在同一节拍中跨接 2 个完成 TLP。在所有其它情况下，这些输出均保留。</p> <p>断言 is_eof_1[0] 有效表明第 2 个 TLP 在同一拍中结束。设置 is_eof_1 的位 0 时，位 [3:1] 可提供在此节拍中结束的 TLP 的最后一个 Dword 的偏移。由于第 2 个 TLP 只能在范围为 27-31 的字节位置中结束，is_eof_1[3:1] 只能采用以下 2 个值中的任一值：6 或 7。</p> <p>第 2 个 TLP 的最后一个字节的偏移可根据 TLP 的起始地址和长度来确定，或者根据字节使能信号 byte_en[31:0] 来确定。</p> <p>如果 is_eof_1[0] 为高电平，那么 is_eof_0[0] 信号和 is_sof_1 信号在同一拍内同样为高电平。</p>
42	discontinue	1	<p>如果核在从其内部 FIFO 存储器读取 TLP 有效载荷时检测到不可纠正的错误，那么它会在 TLP 的最后一个节拍中断言此信号有效。当核发出此类错误信号时，用户应用必须丢弃整个 TLP。</p> <p>当 TLP 不含有效载荷时，永远不会断言此信号有效。仅在有效载荷传输的最后一个节拍内（即 is_eof_0[0] 为高电平时）才会断言此信号有效。</p> <p>启用跨接选项时，核如果在节拍中已断言 discontinue 信号，则不会启动第 2 个 TLP。</p> <p>当核配置为端点时，核也会使用高级错误报告 (AER) 向所连接到的根联合体报告此错误。</p>
74:43	parity	32	<p>256 位发射数据的奇校验。</p> <p>位 i 可提供针对 m_axis_rc_tdata 的字节 i 计算所得奇校验。当接口宽度为 128 位时，仅使用下 16 位，当接口宽度为 64 位时，则仅使用下 8 位。当接口宽度配置为 128 位时，核会将位 [31:16] 永久设置为 0，而当接口宽度配置为 64 位时，则会将位 [31:8] 永久设置为 0。</p>

512 位接口

本章节提供了与核的用户接口相关联的端口的描述。选中 512 位接口时，请复查设计示例中的赛灵思顶层 XDC 文件中的 Pblock 约束。这些约束是将软核 512 位 AXI4-Stream 逻辑保留在 PCIe 集成块附近以便改善时序所必需的。

相关信息

[64/128/256 位接口](#)

完成器请求接口

表 15: 完成器请求接口端口描述 (512 位接口)

名称	I/O	宽度	描述
m_axis_cq_tdata	输出	512	从 PCIe 完成器请求接口向用户应用发射数据。
m_axis_cq_tuser	输出	183	这组信号中包含所传输的 TLP 的边带信息。当 m_axis_cq_tvalid 为高电平时，这些信号有效。下表描述了这组信号中的每个信号。
m_axis_cq_tlast	输出	1	核在包的最后一个节拍内断言此信号有效以指示包结束。在单一节拍内完成 TLP 传输时，核会在传输的首个节拍内设置该位。仅当禁用跨接选项时，才使用此输出。启用跨接选项时，核会将该输出永久设置为 0。

表 15：完成器请求接口端口描述（512 位接口）（续）

名称	I/O	宽度	描述
m_axis_cq_tkeep	输出	16	<p>传输期间断言此总线的位 i 有效对于用户逻辑而言，表示 m_axis_cq_tdata 总线的 Dword i 包含的数据有效。核会针对从描述符的首个 Dword 开始到有效载荷的最后一个 Dword 为止的所有 Dword 将该位连续设置为 1。因此，在包的所有节拍中 m_axis_cq_tdata 必须全部设置为 1，但在包的总大小并非数据总线宽度的整数倍（在 2 个 Dword 内都是如此）的情况下，最后一个节拍除外。对于有效载荷传输的 Dword 对齐模式和 128b 地址对齐模式都是如此。</p> <p>仅当在 CQ 接口上未启用跨接选项时，tkeep 位才有效。启用跨接时，在所有节拍中，tkeep 位全部永久设置为 1。在此情况下，用户逻辑必须在 m_axis_cq_tuser 总线上使用 is_sop/is_eop 信号，以判定通过该接口传输的 TLP 的起始和结束。</p>
m_axis_cq_tvalid	输出	1	核在 m_axis_cq_tdata 总线上驱动有效数据时始终断言此输出有效。核会在包传输期间使有效信号保持处于断言有效状态。核会使用 m_axis_cq_tready 信号来调整数据传输节奏。
m_axis_cq_tready	输入	1	<p>用户逻辑激活此信号对于 PCIe 核而言，表明用户逻辑已准备好接受数据。在同一周期内 m_axis_cq_tvalid 和 m_axis_cq_tready 均断言有效时，就会通过该接口传输数据。</p> <p>当 m_axis_cq_tvalid 为高电平时，如果用户逻辑断言就绪信号无效，那么核会在总线上保留数据，并使有效信号保持断言有效状态，直至用户逻辑断言就绪信号有效为止。</p>
pcie_cq_np_req	输入	2	<p>此输入供用户应用用于请求交付非转发请求。核通过实现基于信用值的流量控制机制来控制通过该接口交付非转发请求的过程，并避免阻塞转发 TLP。</p> <p>此核输入可控制内部信用值。在每个时钟周期内基于如下 pcie_cq_np_req[1:0] 设置来更新信用值：</p> <ul style="list-style-type: none">· 00：无更改· 01：递增 1· 10 或 11：递增 2 <p>在通过接口交付每个非转发请求后，信用值会递减。当信用值达到 0 时，核会暂时停止向用户逻辑交付非转发请求。即使已暂停交付非转发请求，核仍会继续交付从链路接收到的任何转发 TLP。</p> <p>用户应用可以在每个周期内根据其非转发请求接收缓冲器的状态来设置 pcie_cq_np_req[1:0]，或者如果不需对非转发请求运用选择性反压，则可将其永久设置为 11。</p> <p>pcie_cq_np_req[1:0] 的设置无需与完成器请求接口上的包传输对齐。</p>
pcie_cq_np_req_count	输出	6	<p>此输出所提供的值是核为了向用户逻辑交付非转发请求而保留的信用值的当前值。仅当此信用值为非 0 值时，核才能通过完成器请求接口交付非转发请求。该信用值饱和上限为 32。</p> <p>由于存在内部流水线延迟，当用户应用在 pcie_cq_np_req[1:0] 输入上提供信用值后，可能需要经过多个延迟周期后，PCIe 核才会更新 pcie_cq_np_req_count 输出响应。</p> <p>此计数将于执行 user_reset 以及断言 user_lnk_up 无效时复位。</p>

表 16: m_axis_cq_tuser 中的边带信号 (512 位接口)

位索引	名称	宽度	描述
7:0	first_be[7:0]	8	<p>针对有效载荷的第一个 Dword 的字节使能。first_be[3:0] 可反映此节拍中第 1 个 TLP 中的传输事务层报头中的“First Byte Enable”（首字节使能）位的设置，而 first_be[7:4] 则可反映此节拍中第 2 个 TLP 中的传输事务层报头中的 First Byte Enable 位的设置。对于存储器读取和 I/O 读取，这 4 个位用于表示首个 Dword 中将读取的有效字节。对于存储器写入和 I/O 写入，这些位表示有效载荷的首个 Dword 中的有效字节。对于原子操作和含有效载荷的报文，这些位全部设置为 1。</p> <p>仅当在 CQ 接口上启用跨接时，first_be 的位 [7:4] 才有效。禁用跨接时，这些位将永久设置为 0。</p> <p>该字段在包的第一拍中有效。当 m_axis_cq_tvalid 和 is_sop[0] 都断言为高电平有效时，first_be[3:0] 有效。当 m_axis_cq_tvalid 和 is_sop[1] 都断言为高电平有效时，first_be[7:4] 有效。</p>
15:8	last_be[7:0]	8	<p>针对有效载荷的最后一个 Dword 的字节使能。last_be[3:0] 可反映此节拍中第 1 个 TLP 中的传输事务层报头中的“Last Byte Enable”（末字节使能）位的设置，而 last_be[7:4] 则可反映此节拍中第 2 个 TLP 中的传输事务层报头中的 Last Byte Enable 位的设置。对于存储器读取，这 4 个位表示在数据块的最后一个 Dword 中要读取的有效字节。对于存储器写入，这些位表示有效载荷的最后一个 Dword 中的有效字节。对于仅含单次 DW 传输且传输长度为 0 的存储器读取和写入，这些位应为 0。对于原子操作和含有效载荷的报文，这些位全部设置为 1。</p> <p>仅当在 CQ 接口上启用跨接时，last_be 的位 [7:4] 才有效。禁用跨接时，这些位将永久设置为 0。</p> <p>该字段在包的第一拍中有效。当 m_axis_cq_tvalid 和 is_eop[0] 都断言为高电平有效时，last_be[3:0] 有效。当 m_axis_cq_tvalid 和 is_eop[1] 都断言为高电平有效时，last_be[7:4] 有效。</p>
79:16	byte_en[63:0]	64	<p>用户逻辑可以选择使用这些字节使能位来判定所传输的包的有效载荷中的有效字节。传输期间断言该总线的位 i 有效对于用户逻辑而言，表明 m_axis_cq_tdata 总线的字节 i 包含的有效载荷字节是有效的。针对描述符字节，不会断言该位有效。</p> <p>虽然字节使能可由用户逻辑根据请求描述符中的信息（地址和长度）来生成，也可以根据 first_be 信号和 last_be 信号的设置来生成，但用户逻辑还可以选择直接使用这些信号，而不必从其它接口信号生成这些信号。</p> <p>当有效载荷的大小超过 2 个 Dword (8 字节) 时，该总线上对应有效载荷的 first 位始终保持连续。当有效载荷大小不超过 2 个 Dword 时，first 位可能不连续。</p> <p>对于 PCI Express 规范所定义的存储器写入传输事务长度为 0 的特殊情况，传输关联的单 Dword 有效载荷时，byte_en 位全部为 0。</p>
81:80	is_sop[1:0]	2	<p>用作为此节拍中新 TLP 的起始信号。这些输出在 TLP 的第一拍中设置。禁用跨接时，仅限 is_sop[0] 有效，is_sop[1] 则永久设置为 0。启用跨接时，设置如下：</p> <ul style="list-style-type: none">· 00：没有新 TLP 在此节拍中起始。· 01：有单一新 TLP 在此节拍中起始。其起始位置由 is_sop0_ptr[1:0] 来表示。· 11：有 2 个新 TLP 在此节拍中起始。is_sop0_ptr[1:0] 提供第 1 个 TLP 的起始位置，is_sop1_ptr[1:0] 则提供第 2 个 TLP 的起始位置。· 10：保留。 <p>当禁用跨接选项时，是否使用该信号由用户逻辑来判断，因为新 TLP 始终在 tlast 断言有效后的节拍中开始。</p>

表 16: m_axis_cq_tuser 中的边带信号 (512 位接口) (续)

位索引	名称	宽度	描述
83:82	is_sop0_ptr[1:0]	2	用于指示在此节拍中开始的第 1 个 TLP 的第 1 个字节的位置： <ul style="list-style-type: none">· 00：字节通道 0· 10：字节通道 32· 01 和 11：保留 仅当在 CQ 接口上启用跨接选项时，此字段才有效。否则，它永久设置为 0，因为 TLP 只能从字节通道 0 开始。
85:84	is_sop1_ptr[1:0]	2	用于指示在此节拍中开始的第 2 个 TLP 的第 1 个字节的位置： <ul style="list-style-type: none">· 10：字节通道 32· 00、01 和 11：保留。 仅当在 CQ 接口上启用跨接选项时，才使用此输出。随后，核将在同一节拍中跨接 2 个 TLP。禁用跨接时，输出永久设置为 0。
87:86	is_eop[1:0]	2	表示 TLP 在此节拍中结束。这些输出在 TLP 的最后一个节拍中设置。禁用跨接时，仅限 is_eop[0] 有效，is_eop[1] 则永久设置为 0。启用跨接时，设置如下： <ul style="list-style-type: none">· 00：没有任何 TLP 在此节拍中结束。· 01：有单一 TLP 在此节拍中结束。is_eop0_ptr[3:0] 提供此 TLP 的最后一个 Dword 的偏移。· 11：有 2 个 TLP 在此节拍中结束。is_eop0_ptr[3:0] 提供第 1 个 TLP 的最后一个 Dword 的偏移，is_eop1_ptr[3:0] 则提供第 2 个 TLP 的最后一个 Dword 的偏移。· 10：保留。 当不启用跨接选项时，是否使用该信号由用户逻辑来判断，因为在 TLP 的最后一个节拍中断言 tlast 有效。
91:88	is_eop0_ptr[3:0]	4	用于指示在此节拍中结束的第 1 个 TLP 的最后一个 Dword 的偏移。当 is_eop[0] 断言有效时，此输出有效。
95:92	is_eop1_ptr[3:0]	4	用于指示在此节拍中结束的第 2 个 TLP 的最后一个 Dword 的偏移。当 is_eop[1] 断言有效时，此输出有效。 禁用跨接时，输出永久设置为 0。
96	discontinue	1	如果核在从其内部 FIFO 存储器读取 TLP 有效载荷时检测到不可纠正的错误，那么它会在 TLP 的最后一个节拍中断言此信号有效。当核发出此类错误信号时，用户应用必须丢弃整个 TLP。 当 TLP 不含有效载荷时，永远不会断言此信号有效。仅在有效载荷传输的最后一个节拍内（即 is_eop[0] 为高电平时）才会断言此信号有效。 启用跨接选项时，核如果在节拍中已断言 discontinue 信号，则不会启动第 2 个 TLP。 当核配置为端点时，核也会使用高级错误报告 (AER) 向所连接到的根联合体报告此错误。
98:97	tph_present[1:0]	2	这些位表示在通过接口交付的请求 TLP 中存在传输事务处理提示 (TPH)。 <ul style="list-style-type: none">· tph_present[0] 表示在此节拍的第 1 个 TLP 中存在提示。当 m_axis_cq_tvalid 和 is_sop[0] 均为高电平时，tph_present[0] 有效。· tph_present[1] 表示在此节拍的第 2 个 TLP 中存在提示。当 m_axis_cq_tvalid 和 is_sop[1] 均为高电平时，tph_present[1] 有效。

表 16: m_axis_cq_tuser 中的边带信号 (512 位接口) (续)

位索引	名称	宽度	描述
102:99	tph_type[3:0]	4	<p>当请求 TLP 中存在 TPH 时，这 2 个位可提供与提示关联的 PH[1:0] 字段的值。</p> <ul style="list-style-type: none"> · tph_type[1:0] 可提供与此节拍的第 1 个 TLP 关联的 TPH 类型。当 m_axis_cq_tvalid 和 is_sop[0] 均为高电平时，tph_type[1:0] 有效。 · tph_type[3:2] 可提供与此节拍的第 2 个 TLP 关联的 TPH 类型。当 m_axis_cq_tvalid 和 is_sop[1] 均为高电平时，tph_type[3:2] 有效。
118:103	tph_st_tag[15:0]	8	<p>当请求 TLP 中存在 TPH 时，此输出可提供与提示关联的 8 位导向标签。</p> <ul style="list-style-type: none"> · tph_st_tag[7:0] 可提供与此节拍的第 1 个 TLP 关联的导向标签 (Steering Tag)。当 m_axis_cq_tvalid 和 is_sop[0] 均为高电平时，tph_st_tag[7:0] 有效。 · tph_st_tag[15:8] 可提供与此节拍关联的第 2 个 TLP 的 Steering Tag，当 m_axis_cq_tvalid 和 is_sop[1] 均为高电平时，tph_st_atg[15:8] 有效。
182:119	parity	64	512 位发射数据的奇校验。位 i 可提供针对 m_axis_cq_tdata 的字节 i 计算所得奇校验。

完成器完成接口

表 17: 完成器完成接口端口描述 (512 位接口)

名称	I/O	宽度	描述
s_axis_cc_tdata	输入	512	从用户应用到 PCIe 核的完成数据。
s_axis_cc_tuser	输入	81	这组信号中包含所传输的 TLP 的边带信息。当 s_axis_cc_tvalid 为高电平时，这些信号有效。 下表描述了这组信号中的每个信号。
s_axis_cc_tlast	输入	1	用户应用必须在包的最后一个周期内断言此信号有效以指示包结束。在单一节拍 (beat) 内完成 TLP 传输时，用户应用必须在传输的首个周期内设置该位。 仅当禁用“straddle”（跨接）选项时，此输入才可供核使用。启用跨接选项时，该核会忽略此输入的设置，改为使用 s_axis_cc_tuser 总线中的 is_sop/is_eop 信号来判定 TLP 的起始和结束位置。
s_axis_cc_tkeep	输入	16	传输期间断言此总线的位 i 有效对于核而言，表示 s_axis_cc_tdata 总线的 Dword i 包含的数据有效。用户逻辑必须针对从描述符的首个 Dword 开始到有效载荷的最后一个 Dword 为止的所有 Dword 将该位连续设置为 1。因此，在包的所有节拍中，s_axis_cc_tdata 必须全部设置为 1，但在包的总大小并非数据总线宽度的整数倍（在 2 个 Dword 内都是如此）的情况下，最后一个节拍除外。对于有效载荷传输的 Dword 对齐模式和 128b 地址对齐模式都是如此。 仅当在 CC 接口上未启用跨接选项时，tkeep 位才有效。如启用跨接，那么核在整个接口中接收数据时，会忽略这些位的设置。在此情况下，用户逻辑必须在 s_axis_cc_tuser 总线上设置 is_sop/is_eop 信号，以作为通过该接口传输的 TLP 的起始和结束信号。
s_axis_cc_tvalid	输入	1	用户应用在 s_axis_cc_tdata 总线上驱动有效数据时必须始终断言此输出有效。用户应用必须在包传输期间使有效信号保持处于断言有效状态。该核会使用 s_axis_cc_tready 信号来调整数据传输节奏。

表 17：完成器完成接口端口描述（512 位接口）（续）

名称	I/O	宽度	描述
s_axis_cc_tready	输出	4	<p>PCIe 核激活此信号表明它已准备好接受数据。在同一周期内 s_axis_cc_tvalid 和 s_axis_cc_tready 均断言有效时，就会通过该接口传输数据。</p> <p>当有效信号为高电平时，如果核断言就绪信号无效，那么用户逻辑必须在总线上保留数据，并使有效信号保持处于断言有效状态，直至该核断言就绪信号有效为止。</p> <p>对于此输出端口，每个位均指示相同的值，因此用户逻辑可以使用任何位。</p>

表 18：s_axis_cc_tuser 中的边带信号

位索引	名称	宽度	描述
1:0	is_sop[1:0]	2	<p>用作为此节拍中新 TLP 的起始信号。这些输出在 TLP 的第一拍中设置。禁用跨接时，仅限 is_sop[0] 有效。启用跨接时，设置如下：</p> <ul style="list-style-type: none"> · 00：没有新 TLP 在此节拍中起始。 · 01：有单一新 TLP 在此节拍中起始。其起始位置由 is_sop0_ptr[1:0] 来表示。 · 11：有 2 个新 TLP 在此节拍中起始。is_sop0_ptr[1:0] 提供第 1 个 TLP 的起始位置，is_sop1_ptr[1:0] 则提供第 2 个 TLP 的起始位置。 · 10：保留。 <p>仅当启用“straddle”（跨接）选项时，此字段才可供核使用。禁用跨接时，核使用 tlast 来判定传入 TLP 的第一拍。</p>
3:2	is_sop0_ptr[1:0]	2	<p>用于指示在此节拍中开始的第 1 个 TLP 的第 1 个字节的位置：</p> <ul style="list-style-type: none"> · 00：字节通道 0 · 10：字节通道 32 · 01 和 11：保留 <p>仅当启用“straddle”（跨接）选项时，此字段才可供核使用。禁用跨接时，用户逻辑必须始终在字节通道 0 中启动 TLP。</p>
5:4	is_sop1_ptr[1:0]	2	<p>用于指示在此节拍中开始的第 2 个 TLP 的第 1 个字节的位置：</p> <ul style="list-style-type: none"> · 10：字节通道 32 · 00、01 和 11：保留。 <p>仅当在 CC 接口上启用跨接选项时，才使用此输出。随后，用户将在同一节拍中跨接 2 个 TLP。</p>
7:6	is_eop[1:0]	2	<p>用作为 TLP 在此节拍中结束的信号。这些输出在 TLP 的最后一个节拍中设置。禁用跨接时，仅限 is_eop[0] 有效。启用跨接时，设置如下：</p> <ul style="list-style-type: none"> · 00：没有任何 TLP 在此节拍中结束。 · 01：有单一 TLP 在此节拍中结束。is_eop0_ptr[3:0] 提供此 TLP 的最后一个 Dword 的偏移。 · 11：有 2 个 TLP 在此节拍中结束。is_eop0_ptr[3:0] 提供第 1 个 TLP 的最后一个 Dword 的偏移，is_eop1_ptr[3:0] 则提供第 2 个 TLP 的最后一个 Dword 的偏移。 · 10：保留。 <p>仅当启用“straddle”（跨接）选项时，此字段才可供核使用。禁用跨接时，核使用 tlast 和 tkeep 来判定最后一拍和 EOP 的位置。</p>
11:8	is_eop0_ptr[3:0]	4	<p>用于指示在此节拍中结束的第 1 个 TLP 的最后一个 Dword 的偏移。当 is_eop[0] 断言有效时，此输出有效。</p> <p>仅当启用“straddle”（跨接）选项时，此字段才可供核使用。</p>

表 18：s_axis_cc_tuser 中的边带信号 (续)

位索引	名称	宽度	描述
15:12	is_eop1_ptr[3:0]	4	用于指示在此节拍中结束的第 2 个 TLP 的最后一个 Dword 的偏移。当 is_eop[1] 断言有效时，此输出有效。 仅当启用“straddle”（跨接）选项时，此字段才可供核使用。
16	discontinue	1	如果传输期间用户应用在所传输的数据中检测到错误（例如，读取来自存储器的有效载荷时检测到不可纠正 ECC 错误）并且需要中止该数据包，即可断言此信号有效。核会将链路上对应 TLP 置空，以避免数据损坏。 在传输期间，用户逻辑可在除 TLP 第一拍之外的任何节拍中断言此信号有效。它可以选择在周期内发出错误信号处提前终止该数据包，或者也可以继续处理，直至将有效载荷的所有字节都交付到核为止。针对后者，该核会针对数据包的后续节拍将此错误作为粘滞错误来处理，即使用户逻辑在包结束前断言 discontinue 信号无效也是如此。 仅当 s_axis_cc_tvalid 为高电平时，才能断言 discontinue 信号有效。仅当 s_axis_cc_tready 为高电平时，核才会对此信号进行采样。因此，一旦断言有效后，在 s_axis_cc_tready 变为高电平之前不应将其断言无效。 在 CC 接口上启用跨接选项时，如果 TLP 结束时已断言 discontinue 信号有效，那么在同一节拍内，用户不应启动新 TLP。 当核配置为端点时，核也会使用高级错误报告 (AER) 向所连接到的根联合体报告此错误。
80:17	parity	64	256 位数据的奇校验。在核上启用奇偶校验检查时，用户逻辑必须将该逻辑的位 i 设置为针对 s_axis_cc_tdata 的字节 i 计算所得的奇校验。 检测到奇偶校验错误时，核会将链路上的对应 TLP 置空，并将其报告为“Uncorrectable Internal Error”（不可纠正的内部错误）。 如果在核中未启用奇偶校验检查，那么奇偶校验位可永久绑定到 0。

请求器请求接口

表 19：请求器请求接口端口描述 (512 位接口)

名称	I/O	宽度	描述
s_axis_rq_tdata	输入	512	从用户应用到 PCIe 核的请求器侧请求数据。
s_axis_rq_tuser	输入	137	这组信号中包含所传输的 TLP 的边带信息。当 s_axis_rq_tvalid 为高电平时，这些信号有效。下表描述了这组信号中的每个信号。
s_axis_rq_tlast	输入	1	用户应用必须在 TLP 的最后一个周期内断言此信号有效以指示包结束。在单一节拍 (beat) 内完成 TLP 传输时，用户逻辑必须在传输的首个周期内设置该位。 仅当禁用“straddle”（跨接）选项时，此输入才可供核使用。启用跨接选项时，该核会忽略此输入的设置，改为使用 s_axis_rq_tuser 总线中的 is_sop/is_eop 信号来判定 TLP 的起始和结束位置。

表 19：请求器请求接口端口描述（512 位接口）（续）

名称	I/O	宽度	描述
s_axis_rq_tkeep	输入	16	<p>传输期间断言此总线的位 i 有效对于核而言，表示 s_axis_rq_tdata 总线的 Dword i 包含的数据有效。用户逻辑必须针对从描述符的首个 Dword 开始到有效载荷的最后一个 Dword 为止的所有 Dword 将该位连续设置为 1。因此，在包的所有节拍中，s_axis_rq_tdata 必须全部设置为 1，但在包的总大小并非数据总线宽度的整数倍（在 2 个 Dword 内都是如此）的情况下，最后一个节拍除外。对于有效载荷传输的 Dword 对齐模式和 128b 地址对齐模式都是如此。</p> <p>仅当在 RQ 接口上未启用跨接选项时，tkeep 位才有效。如启用跨接，那么核在整个接口中接收数据时，会忽略这些位的设置。在此情况下，用户逻辑必须在 s_axis_rq_tuser 总线上设置 is_sop/is_eop 信号，以作为通过该接口传输的 TLP 的起始和结束信号。</p>
s_axis_rq_tvalid	输入	1	用户应用在 s_axis_rq_tdata 总线上驱动有效数据时必须始终断言此输出有效。用户应用必须在包传输期间使有效信号保持处于断言有效状态。该核会使用 s_axis_rq_tready 信号来调整数据传输节奏。
s_axis_rq_tready	输出	4	<p>PCIe 核激活此信号表明它已准备好接受数据。在同一周期内 s_axis_rq_tvalid 和 s_axis_rq_tready 均断言有效时，就会通过该接口传输数据。</p> <p>当有效信号为高电平时，如果核断言就绪信号无效，那么用户逻辑必须在总线上保留数据，并使有效信号保持处于断言有效状态，直至该核断言就绪信号有效为止。</p> <p>对于此输出端口，每个位均指示相同的值，因此用户逻辑可以使用任何位。</p>
pcie_rq_tag_vld0	输出	1	当该核将标签分配给从请求器请求接口传入的非转发请求，并将此标签置于 pcie_rq_tag0 输出上时，就会断言此输出有效，并保持 1 个周期。此位编码方式如下：
			<ul style="list-style-type: none">· 0：在此周期内不提供任何标签。· 1：在 pcie_rq_tag0 上存在 1 个标签。
pcie_rq_tag_vld1	输出	1	当该核将标签分配给从请求器请求接口传入的非转发请求，并将此标签置于 pcie_rq_tag1 输出上时，就会断言此输出有效，并保持 1 个周期。此位编码方式如下：
			<ul style="list-style-type: none">· 0：在此周期内，在 pcie_rq_tag1 上不提供任何标签。· 1：在 pcie_rq_tag1 上存在 1 个标签。
pcie_rq_tag0	输出	8	<p>在 IP customization GUI 中不勾选“Enable Client Tag”（启用客户标签）时，由核执行非转发请求的标签管理操作，在此情况下，此输出供该核用于传达从客户接收到的每个非转发请求所分配的标签。当 pcie_rq_tag_vld0 为高电平时，pcie_rq_tag0 上的标签值在 1 个周期内有效。客户必须复制此标签并将其用于将完成数据与暂挂请求相关联。</p> <p>在 s_axis_rq_tdata 总线上上传输请求后，可能要经过数个周期的延迟之后，核才会断言 pcie_rq_tag_vld0 有效以便为请求提供已分配的标签。与此同时，客户可以继续发送新请求。在此总线上，请求的标签按 FIFO 顺序来传递。因此，用户应用必须将已分配的标签与请求相关联，且关联的顺序必须与通过接口传输请求的顺序相同。</p> <p>在同一周期内 pcie_rq_tag0 和 pcie_rq_tag1 均有效时，pcie_rq_tag0 上的值对应于通过接口传输的 2 个请求中优先传输的请求。</p>
pcie_rq_tag1	输出	8	此信号的描述与 pcie_rq_tag0 相同，唯一区别在于当 pcie_rq_tag_vld1 断言有效时，pcie_rq_tag1 上的标签值有效期限为 1 个周期。

表 19：请求器请求接口端口描述（512 位接口）（续）

名称	I/O	宽度	描述
pcie_rq_seq_num0	输出	6	用户可以选择使用此输出在核的发射流水线中保留请求的进展情况记录。要使用此功能，用户应用必须为 s_axis_rq_seq_num0[5:0] 总线上每个请求提供 1 个序列号。当流水线中的请求 TLP 进展至特定状态（即，来自客户的完成 TLP 无法继续传递该包）时，该核会在 pcie_rq_seq_num0[5:0] 输出上输出此序列号。此机制支持客户在发送到核的完成器完成接口的完成包与发送到请求器请求接口的转发请求之间保留顺序不变。 当 pcie_rq_seq_num_vld0 为高电平时，pcie_rq_seq_num0[5:0] 输出上的数据有效。
pcie_rq_seq_num1	输出	6	此输出的功能与 pcie_rq_seq_num0 相同。当在 pcie_rq_seq_num0 上出现第 1 个序列号时，它用于在同一周期内提供第 2 个序列号。 当 pcie_rq_seq_num_vld1 为高电平时，pcie_rq_seq_num1[5:0] 输出上的数据有效。
pcie_rq_seq_num_vld0	输出	1	当此输出将有效数据置于 pcie_rq_seq_num0[5:0] 上时，核会断言此输出有效，并保持 1 个周期。
pcie_rq_seq_num_vld1	输出	1	当此输出将有效数据置于 pcie_rq_seq_num1[5:0] 上时，核会断言此输出有效，并保持 1 个周期。

表 20：s_axis_rq_tuser 中的边带信号（512 位接口）

位索引	名称	宽度	描述
7:0	first_be[7:0]	8	首个 Dword 的字节使能。该字段必须基于请求 TLP 的传输事务层报头中的 First_BE 位的期望值来设置。first_be[3:0] 对应于此节拍中开始的首个 TLP 的字节使能，而 first_be[7:4] 则对应于此节拍中开始的第 2 个 TLP（如果存在）的字节使能。 对于存储器读取、I/O 读取和配置读取，这 4 位表示首个 Dword 中将读取的有效字节。对于存储器写入、I/O 写入和配置写入，这些位表示有效载荷的首个 Dword 中的有效字节。 当 s_axis_rq_tvalid 和 s_axis_rq_tready 均为高电平时，核会在包的第一拍中对该字段进行采样。
15:8	last_be[7:0]	8	最后一个 Dword 的字节使能。 该字段必须基于 TLP 的传输事务层报头中的 Last_BE 位的期望值来设置。last_be[3:0] 对应于此节拍中开始的首个 TLP 的字节使能，而 last_be[7:4] 则对应于此节拍中开始的第 2 个 TLP（如果存在）的字节使能。 对于仅含单次 DW 传输且传输长度为 0 的存储器读取和写入，这些位应为 0。 对于不少于 2 个 Dword 的存储器读取，这 4 位表示在数据块的最后一个 Dword 中要读取的有效字节。对于不少于 2 个 Dword 的存储器写入，这些位表示有效载荷的最后一个 Dword 中的有效字节。 当 s_axis_rq_tvalid 和 s_axis_rq_tready 均为高电平时，核会在包的第一拍中对该字段进行采样。
19:16	addr_offset[3:0]	4	当在此接口中使用 128b 地址对齐模式时，用户应用必须提供偏移，其中有效载荷数据在此边带总线的数据总线上开始（按 4 字节的倍数）。这样，核即可判定所传输的数据块的对齐方式。 addr_offset[1:0] 对应于在此节拍中开始的首个 TLP 的偏移，而 addr_offset[3:2] 则保留以供将来使用。 当 s_axis_rq_tvalid 和 s_axis_rq_tready 均为高电平时，核会在包的第一拍中对该字段进行采样。 当请求器请求接口采用 Dword 对齐模式配置时，这些位必须始终设置为 0。

表 20：s_axis_rq_tuser 中的边带信号（512 位接口）（续）

位索引	名称	宽度	描述
21:20	is_sop[1:0]	2	<p>用作为此节拍中新 TLP 的起始信号。这些输出在 TLP 的第一拍中设置。禁用跨接时，仅限 is_sop[0] 有效。启用跨接时，设置如下：</p> <ul style="list-style-type: none">· 00：没有新 TLP 在此节拍中起始。· 01：有单一新 TLP 在此节拍中起始。其起始位置由 is_sop0_ptr[1:0] 来表示。· 11：有 2 个新 TLP 在此节拍中起始。is_sop0_ptr[1:0] 提供第 1 个 TLP 的起始位置，is_sop1_ptr[1:0] 则提供第 2 个 TLP 的起始位置。· 10：保留。 <p>当不启用跨接选项时，是否使用该信号由用户逻辑来判断，因为新 TLP 始终在 tlast 断言有效后的节拍中开始。</p>
23:22	is_sop0_ptr[1:0]	2	<p>用于指示在此节拍中开始的第 1 个 TLP 的第 1 个字节的位置：</p> <ul style="list-style-type: none">· 00：字节通道 0· 10：字节通道 32· 01 和 11：保留
25:24	is_sop1_ptr[1:0]	2	<p>用于指示在此节拍中开始的第 2 个 TLP 的第 1 个字节的位置：</p> <ul style="list-style-type: none">· 10：字节通道 32· 00、01 和 11：保留。 <p>仅当在接口上启用跨接选项时，才使用此输出。</p>
27:26	is_eop[1:0]	2	<p>用作为 TLP 在此节拍中结束的信号。这些输出在 TLP 的最后一个节拍中设置。禁用跨接时，仅限 is_eop[0] 有效。启用跨接时，设置如下：</p> <ul style="list-style-type: none">· 00：没有任何 TLP 在此节拍中结束。· 01：有单一 TLP 在此节拍中结束。is_eop0_ptr[3:0] 提供此 TLP 的最后一个 Dword 的偏移。· 11：有 2 个 TLP 在此节拍中结束。is_eop0_ptr[3:0] 提供第 1 个 TLP 的最后一个 Dword 的偏移，is_eop1_ptr[3:0] 则提供第 2 个 TLP 的最后一个 Dword 的偏移。· 10：保留。 <p>当不启用跨接选项时，是否使用该信号由用户逻辑来判断，因为在 TLP 的最后一个节拍中断言 tlast 有效。</p>
31:28	is_eop0_ptr[3:0]	4	<p>用于指示在此节拍中结束的第 1 个 TLP 的最后一个 Dword 的偏移。当 is_eop[0] 断言有效时，此输出有效。</p>
35:32	is_eop1_ptr[3:0]	4	<p>用于指示在此节拍中结束的第 2 个 TLP 的最后一个 Dword 的偏移。当 is_eop[1] 断言有效时，此输出有效。</p>

表 20：s_axis_rq_tuser 中的边带信号（512 位接口）（续）

位索引	名称	宽度	描述
36	discontinue	1	<p>如果传输期间用户应用在所传输的数据中检测到错误并且需要中止该数据包，即可断言此信号有效。核会将链路上对应 TLP 置空，以避免数据损坏。</p> <p>在传输期间，用户逻辑可在除第一拍以外的任一 TLP 节拍内断言此信号有效。它可以选择在周期内发出错误信号处提前终止该数据包，或者也可以继续处理，直至将有效载荷的所有字节都交付到核为止。针对后者，该核会针对数据包的后续节拍将此错误作为粘滞错误来处理，即使用户逻辑在包结束前断言 discontinue 信号无效也是如此。</p> <p>仅当 s_axis_rq_tvalid 为高电平时，才能断言 discontinue 信号有效。仅当 s_axis_rq_tready 为高电平时，核才会对此信号进行采样。因此，一旦断言有效后，在 s_axis_rq_tready 变为高电平之前不应将其断言。</p> <p>在 RQ 接口上启用跨接选项时，如果 TLP 结束时已断言 discontinue 信号有效，那么在同一节拍内，用户不应启动新 TLP。</p> <p>当核配置为端点时，核也会使用高级错误报告 (AER) 向所连接到的根联合体报告此错误。</p>
38:37	tph_present[1:0]	2	<p>此位表示在通过接口交付的请求 TLP 中存在“Transaction Processing Hint (TPH)”（传输事务处理提示）。当 s_axis_rq_tvalid 和 s_axis_rq_tready 均为高电平时，核会在包的第一拍中对该字段进行采样。</p> <ul style="list-style-type: none">· tph_present[0] 对应于此节拍中开始的第 1 个 TLP。· tph_present[1] 对应于此节拍中开始的第 2 个 TLP（如果存在）。 <p>如果不使用 TPH 功能，那么这些输入必须永久绑定到 0。</p>
42:39	tph_type[3:0]	4	<p>当请求 TLP 中存在 TPH 时，这 2 个位可提供与提示关联的 PH[1:0] 字段的值。当 s_axis_rq_tvalid 和 s_axis_rq_tready 均为高电平时，核会在包的第一拍中对该字段进行采样。</p> <ul style="list-style-type: none">· tph_type[1:0] 对应于此节拍中开始的第 1 个 TLP。· tph_type[3:2] 对应于此节拍中开始的第 2 个 TLP（如果存在）。 <p>如果对应 tph_present 位设置为 0，那么这些位可设置为任意值。</p>
44:43	tph_indirect_tag_en[1:0]	2	<p>设置该位后，核会使用 tph_st_tag 上存在的标签的较低的位，作为其“Steering Tag Table”（导向标签表）中的索引，并将来自该位置的标签插入发射的请求 TLP 中。当该位设置为 0 时，核直接使用 tph_st_tag 上的值作为导向标签。</p> <ul style="list-style-type: none">· tph_indirect_tag_en[0] 对应于此节拍中开始的第 1 个 TLP。· tph_indirect_tag_en[1] 对应于此节拍中开始的第 2 个 TLP（如果存在）。 <p>当 s_axis_rq_tvalid 和 s_axis_rq_tready 均为高电平时，核会在包的第一拍中对该位进行采样。</p> <p>如果对应 tph_present 位设置为 0，那么这些输入可设置为任意值。</p>
60:45	tph_st_tag[15:0]	16	<p>当请求 TLP 中存在 TPH 时，此输出可提供与提示关联的 8 位导向标签。当 s_axis_rq_tvalid 和 s_axis_rq_tready 均为高电平时，核会在包的第一拍中对该字段进行采样。</p> <ul style="list-style-type: none">· tph_st_tag[7:0] 对应于此节拍中开始的第 1 个 TLP。· tph_st_tag[15:8] 对应于此节拍中开始的第 2 个 TLP（如果存在）。 <p>如果对应 tph_present 位设置为 0，那么这些输入可设置为任意值。</p>

表 20：s_axis_rq_tuser 中的边带信号（512 位接口）（续）

位索引	名称	宽度	描述
66:61	seq_num0[5:0]	6	<p>用户逻辑可以选择在此字段中提供 6 位序列号，以便在核的发射流水线中保留请求的进展情况记录。当流水线中的请求 TLP 进展至特定状态（即，来自用户逻辑的完成 TLP 无法继续传递该包）时，该核会在其 pcie_rq_seq_num0 或 pcie_rq_seq_num1 输出上输出此序列号。</p> <p>当 s_axis_rq_tvalid 和 s_axis_rq_tready 均为高电平时，核会在包的第一拍中对该字段进行采样。</p> <p>当用户逻辑未监控核的 pcie_rq_seq_num* 输出时，此输入可硬连线到 0。</p>
72:67	seq_num1[5:0]	6	<p>如果有第 2 个 TLP 在同一个节拍中开始，那么用户逻辑可以选择在此输入上为此 TLP 提供 6 位序列号。此序列号使用方式与 seq_num0 相同。</p> <p>当 s_axis_rq_tvalid 和 s_axis_rq_tready 均为高电平时，核会在包的第一拍中对该字段进行采样。</p> <p>当用户逻辑未监控核的 pcie_rq_seq_num* 输出时，此输入可硬连线到 0。</p>
136:73	parity	64	<p>512 位数据的奇校验。在核上启用奇偶校验检查时，用户逻辑必须将该逻辑的位 i 设置为针对 s_axis_rq_tdata 的字节 i 计算所得的奇校验。</p> <p>检测到奇偶校验错误时，核会将链路上的对应 TLP 置空，并将其报告为“Uncorrectable Internal Error”（不可纠正的内部错误）。</p> <p>如果在核中禁用奇偶校验检查，那么这些位可设置为 0。</p>

请求器完成接口

表 21：请求器完成接口端口描述（512 位接口）

名称	I/O	宽度	描述
m_axis_rc_tdata	输出	512	从 PCIe 请求器完成接口向用户应用发射数据。
m_axis_rc_tuser	输出	161	这组信号中包含所传输的 TLP 的边带信息。当 m_axis_rc_tvalid 为高电平时，这些信号有效。下表描述了这组信号中的每个信号。
m_axis_rc_tlast	输出	1	核在包的最后一个节拍内断言此信号有效以指示包结束。在单一节拍内完成 TLP 传输时，核会在传输的首个节拍内设置该位。仅当禁用跨接选项时，才使用此输出。启用跨接选项时，核会将该输出永久设置为 0。
m_axis_rc_tkeep	输出	16	<p>传输期间断言此总线的位 i 有效对于用户逻辑而言，表示 m_axis_rc_tdata 总线的 Dword i 包含的数据有效。核会针对从描述符的首个 Dword 开始到有效载荷的最后一个 Dword 止的所有 Dword 将该位连续设置为 1。因此，在包的所有节拍中 m_axis_rc_tkeep 必须全部设置为 1，但在包的总大小并非数据总线宽度的整数倍（在 2 个 Dword 内都是如此）的情况下，最后一个节拍除外。对于有效载荷传输的 Dword 对齐模式和地址对齐模式都是如此。</p> <p>启用跨接选项时，这些输出全部永久设置为 1。在此情况下，用户逻辑必须使用 m_axis_rc_tuser 中的信号，以判定通过该接口传输的完成 TLP 的起始和结束。</p>
m_axis_rc_tvalid	输出	1	核在 m_axis_rc_tdata 总线上驱动有效数据时始终断言此输出有效。核会在包传输期间使有效信号保持处于断言有效状态。核会使用 m_axis_rc_tready 信号来调整数据传输节奏。

表 21：请求器完成接口端口描述（512 位接口）（续）

名称	I/O	宽度	描述
m_axis_rc_tready	输入	1	<p>用户逻辑激活此信号对于 PCIe 核而言，表明用户逻辑已准备好接受数据。在同一周期内 m_axis_rc_tvalid 和 m_axis_rc_tready 均断言有效时，就会通过该接口传输数据。</p> <p>当有效信号为高电平时，如果用户逻辑断言就绪信号无效，那么核会在总线上保留数据，并使有效信号保持断言有效状态，直至用户逻辑断言就绪信号有效为止。</p>

表 22：m_axis_rc_tuser 中的边带信号（512 位接口）

位索引	名称	宽度	描述
63:0	byte_en	64	<p>客户逻辑可以选择使用这些字节使能位来判定所传输的包的有效载荷中的有效字节。传输期间断言此总线的位 i 有效对于客户而言，表明 m_axis_cq_tdata 总线的字节 i 包含的有效载荷字节是有效的。针对描述符字节，不会断言该位有效。</p> <p>虽然字节使能可由客户逻辑根据请求描述符中的信息（地址和长度）来生成，但客户还可以选择直接使用这些信号，而不必从其它接口信号生成这些信号。该总线中对应 TLP 有效载荷的值为 1 的位始终连续。</p>
67:64	is_sop[3:0]	4	<p>用作为此节拍中新 TLP 的起始信号。这些输出在 TLP 的第一拍中设置。禁用跨接时，仅限 is_sop[0] 有效，is_sop[3:1] 则永久设置为 0。启用跨接时，设置如下：</p> <ul style="list-style-type: none"> · 0000：没有新 TLP 在此节拍中起始。 · 01：有单一新 TLP 在此节拍中起始。ts 起始位置以 is_sop0_ptr[1:0] 来表示。 · 0011：有 2 个新 TLP 在此节拍中起始。is_sop0_ptr[1:0] 提供第 1 个 TLP 的起始位置，is_sop1_ptr[1:0] 则提供第 2 个 TLP 的起始位置。 · 0111：有 3 个新 TLP 在此节拍中开始。is_sop0_ptr[1:0] 提供第 1 个 TLP 的起始位置，is_sop1_ptr[1:0] 提供第 2 个 TLP 的起始位置，is_sop2_ptr[1:0] 提供第 3 个 TLP 的起始位置。 · 1111：有 4 个新 TLP 在此节拍中开始。is_sop0_ptr[1:0] 提供第 1 个 TLP 的起始位置，is_sop1_ptr[1:0] 提供第 2 个 TLP 的起始位置，is_sop2_ptr[1:0] 提供第 3 个 TLP 的起始位置，is_sop3_ptr[1:0] 提供第 4 个 TLP 的起始位置。 · 所有其它设置均保留。 <p>当不启用跨接选项时，是否使用该信号由客户来判断，因为新 TLP 始终在 m_axis_rc_tlast 断言有效后的节拍中开始。</p>
69:68	is_sop0_ptr[1:0]	2	<p>用于指示在此节拍中开始的第 1 个 TLP 的第 1 个字节的位置：</p> <ul style="list-style-type: none"> · 00：字节通道 0 · 01：字节通道 16 · 10：字节通道 32 · 11：字节通道 48 <p>仅当在 RC 接口上启用跨接选项时，此字段才有效。否则，它永久设置为 0，因为 TLP 只能从字节通道 0 开始。</p>
71:70	is_sop1_ptr[1:0]	2	<p>用于指示在此节拍中开始的第 2 个 TLP 的第 1 个字节的位置：</p> <ul style="list-style-type: none"> · 00：保留 · 01：字节通道 16 · 10：字节通道 32 · 11：字节通道 48 <p>仅当在 RC 接口上启用跨接选项时，才使用此输出。禁用跨接时，输出永久设置为 0。</p>

表 22：m_axis_rc_tuser 中的边带信号（512 位接口）（续）

位索引	名称	宽度	描述
73:72	is_sop2_ptr[1:0]	2	<p>用于指示在此节拍中开始的第 3 个 TLP 的第 1 个字节的位置：</p> <ul style="list-style-type: none">· 00：保留· 01：保留· 10：字节通道 32· 11：字节通道 48 <p>仅当在 RC 接口上启用跨接选项时，才使用此输出。禁用跨接时，输出永久设置为 0。</p>
75:74	is_sop3_ptr[1:0]	2	<p>用于指示在此节拍中开始的第 4 个 TLP 的第 1 个字节的位置：</p> <ul style="list-style-type: none">· 00、01 和 10：保留· 11：字节通道 48 <p>仅当在 RC 接口上启用跨接选项时，才使用此输出。禁用跨接时，输出永久设置为 0。</p>
79:76	is_eop[3:0]	4	<p>此信号表示，仅当启用跨接时，1 个或多个 TLP 在此节拍中结束。这些输出在 TLP 的最后一个节拍中设置。设置如下：</p> <ul style="list-style-type: none">· 0000：没有任何 TLP 在此节拍中结束。· 0001：有单一 TLP 在此节拍中结束。is_eop0_ptr[3:0] 提供此 TLP 的最后一个 Dword 的偏移。· 0011：有 2 个 TLP 在此节拍中结束。is_eop0_ptr[3:0] 提供第 1 个 TLP 的最后一个 Dword 的偏移，is_eop1_ptr[3:0] 则提供第 2 个 TLP 的最后一个 Dword 的偏移。· 0111：有 3 个 TLP 在此节拍中结束。is_eop0_ptr[3:0] 提供第 1 个 TLP 的最后一个 Dword 的偏移，is_eop1_ptr[3:0] 提供第 2 个 TLP 的最后一个 Dword 的偏移，is_eop2_ptr[3:0] 提供第 3 个 TLP 的最后一个 Dword 的偏移。· 1111：有 4 个 TLP 在此节拍中结束。is_eop0_ptr[3:0] 提供第 1 个 TLP 的最后一个 Dword 的偏移，is_eop1_ptr[3:0] 提供第 2 个 TLP 的最后一个 Dword 的偏移，is_eop2_ptr[3:0] 提供第 3 个 TLP 的最后一个 Dword 的偏移，is_eop3_ptr[3:0] 提供第 4 个 TLP 的最后一个 Dword 的偏移。· 所有其它设置均保留。 <p>当禁用跨接选项时，m_axis_rc_tlast 指示 TLP 的最后一个节拍。</p>
83:80	is_eop0_ptr[3:0]	4	<p>用于指示在此节拍中结束的第 1 个 TLP 的最后一个 Dword 的偏移。当 is_eop[0] 断言有效时，此输出有效。</p> <p>仅当在 RC 接口上启用跨接选项时，才使用此输出。禁用跨接时，输出永久设置为 0。</p>
87:84	is_eop1_ptr[3:0]	4	<p>用于指示在此节拍中结束的第 2 个 TLP 的最后一个 Dword 的偏移。当 is_eop[1] 断言有效时，此输出有效。</p> <p>仅当在 RC 接口上启用跨接选项时，才使用此输出。禁用跨接时，输出永久设置为 0。</p>
91:88	is_eop2_ptr[3:0]	4	<p>用于指示在此节拍中结束的第 3 个 TLP 的最后一个 Dword 的偏移。当 is_eop[2] 断言有效时，此输出有效。</p> <p>仅当在 RC 接口上启用跨接选项时，才使用此输出。禁用跨接时，输出永久设置为 0。</p>
95:92	is_eop3_ptr[3:0]	4	<p>用于指示在此节拍中结束的第 4 个 TLP 的最后一个 Dword 的偏移。当 is_eop[3] 断言有效时，此输出有效。</p> <p>仅当在 RC 接口上启用跨接选项时，才使用此输出。禁用跨接时，输出永久设置为 0。</p>

表 22：m_axis_rc_tuser 中的边带信号（512 位接口）（续）

位索引	名称	宽度	描述
96	discontinue	1	如果核在其内部 FIFO 存储器读取 TLP 有效载荷时检测到不可纠正的错误，那么它会在 TLP 的最后一个节拍中断言此信号有效。当核发出此类错误信号时，客户应用必须丢弃整个 TLP。 当 TLP 不含有效载荷时，永远不会断言此信号有效。仅在有效载荷传输的最后一个节拍内（即 is_eop[0] 为高电平时）才会断言此信号有效。 启用跨接选项时，核如果在节拍中已断言 discontinue 信号，则不会启动新的 TLP。 当核配置为端点时，核也会使用高级错误报告 (AER) 向所连接到的根联合体报告此错误。
160:97	parity	64	512 位发射数据的奇校验。位 i 可提供针对 m_axis_cq_tdata 的字节 i 计算所得奇校验。

其它核接口

本章描述了核所提供的接口。

功耗管理接口

下表定义了核的“Power Management”（功耗管理）接口中的端口。

表 23：功耗管理接口端口

端口	I/O	宽度	描述
cfg_pm_aspm_l1_entry_reject	输入	1	配置功耗管理 ASPM L1 输入拒绝：驱动到 1b 时，下游端口拒绝转换为 L1 状态的请求。
cfg_pm_aspm_tx_l0s_entry_disable	输入	1	配置功耗管理 ASPM L0s 输入禁用：驱动到 1b 时，阻止端口进入 TX L0s。

配置管理接口

“Configuration Management”（配置管理）接口用于在配置空间寄存器上执行读取和写入。下表定义了核的“Configuration Management”接口中的端口。

表 24：配置管理接口端口描述

端口	I/O	宽度	描述
cfg_mgmt_addr	输入	10	读取/写入地址 配置空间 Dword 对齐地址。
cfg_mgmt_function_number	输入	8	PCI 功能编号 选择配置寄存器读取/写入的 PCI 功能编号。
cfg_mgmt_write	输入	1	写入使能 针对写入操作断言有效。高电平有效。
cfg_mgmt_write_data	输入	32	写入数据 写入数据用于对配置和管理寄存器进行配置。

表 24：配置管理接口端口描述 (续)

端口	I/O	宽度	描述
cfg_mgmt_byte_enable	输入	4	字节使能 针对写入数据的字节使能，其中 cfg_mgmt_byte_enable[0] 对应于 cfg_mgmt_write_data[7:0]，以此类推。
cfg_mgmt_read	输入	1	读取使能 针对读取操作断言有效。高电平有效。
cfg_mgmt_read_data	输出	32	读取数据输出 读取数据用于提供配置和管理寄存器的配置。
cfg_mgmt_read_write_done	输出	1	读取/写入操作完成 当操作完成时，断言有效并保持 1 个周期。高电平有效。
cfg_mgmt_debug_access	输入	1	类型 1 RO，写入 当核配置为采用“Root Port”（根端口）模式时，如果在写入类型 1 配置空间寄存器期间断言此输入有效，则将导致强制写入寄存器的某些只读字段（请参阅 RC 模式配置寄存器的描述）。当核处于“Endpoint”（端点）模式下时，或者写入除类型 1 配置空间寄存器以外的任何其它寄存器时，此输入无效。

配置状态接口

配置状态接口可提供有关核配置方式的信息，例如，协商的链路宽度和速度、核的功耗状态以及配置错误等。下表定义了核的“Configuration Status”（配置状态）接口中的端口。

表 25：配置状态接口端口描述

端口	I/O	宽度	描述
cfg_phy_link_down	输出	1	<p>配置链路中断 PCI Express 链路状态，基于物理层 LTSSM。</p> <ul style="list-style-type: none"> · 1b：链路中断（LinkUp 状态变量为 0b） · 0b：链路为上行（LinkUp 状态变量为 1b） <p>注释：根据 PCI Express 基本规范 3.0 版，LinkUp 在“Recovery”（恢复）、L0、L0s、L1 和 L2 cfg_ltssm 状态下为 1b。在“Configuration”（配置）状态下，LinkUp 可为 0b 或 1b。当使用“Detect > Polling > Configuration”（检测 > 轮询 > 配置）达到“Configuration”状态时，则始终为 0b。如果通过任何其它状态转换达到 Configuration 状态，则 LinkUp 为 1b。</p> <p>注释：断言复位有效时，此信号的输出为 0b，直至释放复位为止。</p>
cfg_phy_link_status	输出	2	<p>配置链路状态 PCI Express 链路的状态。</p> <ul style="list-style-type: none"> · 00b：未检测到接收器 · 01b：正在进行链路训练 · 10b：链路上行，正在进行 DL 初始 · 11b：链路上行，DL 初始已完成

表 25：配置状态接口端口描述 (续)

端口	I/O	宽度	描述
cfg_negotiated_width	输出	3	<p>协商的链路宽度 此输出指示给定 PCI Express 链路的协商宽度，当 cfg_phy_link_status[1:0] == 11b (DL 初始化完成) 时有效。</p> <p>协商的链路宽度值：</p> <ul style="list-style-type: none">· 000b = x1· 001b = x2· 010b = x4· 011b = x8· 100b = x16· 其它值均为保留值。
cfg_current_speed	输出	3	<p>当前链路速度 此信号用于输出给定 PCI Express 链路的当前链路速度。</p> <ul style="list-style-type: none">· 00b: 2.5 GT/s PCI Express 链路速度· 01b: 5.0 GT/s PCI Express 链路速度· 10b: 8.0 GT/s PCI Express 链路速度· 11b: 保留
cfg_max_payload	输出	2	<p>Max_Payload_Size 此信号用于输出最大有效载荷大小，从器件控制寄存器位 7 向下直至位 5。该字段用于设置最大 TLP 有效载荷大小。作为接收器，逻辑处理的 TLP 数量不得超过此处设定的值。作为发射器，逻辑生成的 TLP 数量不得超过此处设定的值。</p> <ul style="list-style-type: none">· 00b: 128 字节最大有效载荷大小· 01b: 256 字节最大有效载荷大小· 10b: 512 字节最大有效载荷大小· 11b: 1024 字节最大有效载荷大小
cfg_max_read_req	输出	3	<p>Max_Read_Request_Size 此信号用于输出最大读取请求大小，从器件控制寄存器位 14 向下直至 12。该字段作为“Requester”(请求器)，用于为逻辑设置最大读取请求大小。此逻辑生成的读取请求大小不得超过此处设定的值。</p> <ul style="list-style-type: none">· 000b: 128 字节最大读取请求大小· 001b: 256 字节最大读取请求大小· 010b: 512 字节最大读取请求大小· 011b: 1024 字节最大读取请求大小· 100b: 2048 字节最大读取请求大小· 101b: 4096 字节最大读取请求大小· 其它值均为保留值

表 25：配置状态接口端口描述 (续)

端口	I/O	宽度	描述
cfg_function_status	输出	16	<p>配置功能状态 这些输出用于指示每个功能的 PCI 配置空间中的命令寄存器位的状态。这些输出用于启用来自主机逻辑的请求和完成包。位的分配如下：</p> <ul style="list-style-type: none">· 位 0：功能 0 I/O 空间使能· 位 1：功能 0 存储器空间使能· 位 2：功能 0 总线主控制器使能· 位 3：功能 0 INTx 禁用· 位 4：功能 1 I/O 空间使能· 位 5：功能 1 存储器空间使能· 位 6：功能 1 总线主控制器使能· 位 7：功能 1 INTx 禁用· 位 8：功能 2 I/O 空间使能· 位 9：功能 2 存储器空间使能· 位 10：功能 2 总线主控制器使能· 位 11：功能 2 INTx 禁用· 位 12：功能 3 I/O 空间使能· 位 13：功能 3 存储器空间使能· 位 14：功能 3 总线主控制器使能· 位 15：功能 3 INTx 禁用
cfg_vf_status	输出	504	<p>配置虚拟功能状态 · 位 0：虚拟功能 0：已由软件配置/启用。 · 位 1：虚拟功能 0：PCI 命令寄存器，总线主控制器使能。 · 位 2：虚拟功能 1：已由软件配置/启用。 · 位 3：虚拟功能 1：PCI 命令寄存器，总线主控制器使能。</p>
cfg_function_power_state	输出	12	<p>配置功能功耗状态 这些输出用于指示物理功能的功耗状态。位 [2:0] 用于捕获功能 0 的功耗状态，位 [5:3] 则用于捕获功能 1 的功耗状态，以此类推。可能的功耗状态包括：</p> <ul style="list-style-type: none">· 000: D0_uninitialized· 001: D0_active· 010: D1· 100: D3_hot· 其它值均为保留值。
cfg_vf_power_state	输出	756	<p>配置虚拟功能功耗状态 这些输出用于指示虚拟功能的功耗状态。位 [2:0] 用于捕获虚拟功能 0 的功耗状态，位 [5:3] 则用于捕获虚拟功能 1 的功耗状态，以此类推。可能的功耗状态包括：</p> <ul style="list-style-type: none">· 000: D0_uninitialized· 001: D0_active· 010: D1· 100: D3_hot· 其它值均为保留值。

表 25：配置状态接口端口描述 (续)

端口	I/O	宽度	描述
cfg_link_power_state	输出	2	<p>PCI Express 链路的当前功耗状态，当 cfg_phy_link_status[1:0] == 11b (DL 初始化完成) 时有效。</p> <ul style="list-style-type: none">· 00: L0· 01: TX L0s· 10: L1· 11: L2/3 就绪
cfg_local_error_out	输出	5	<p>局部错误状况：已记录错误优先级，优先级 0 为最高优先级。</p> <ul style="list-style-type: none">· 00000b - 保留· 00001b - 检测到物理层错误（优先级 21）· 00010b - 链路回放超时（优先级 17）· 00011b - 链路回放滚动（优先级 18）· 00100b - 接收到链路错误 TLP（优先级 15）· 00101b - 接收到链路错误 DLLP（优先级 16）· 00110b - 链路协议错误（优先级 10）· 00111b - 回放缓冲器 RAM 可纠正 ECC 错误（优先级 27）· 01000b - 回放缓冲器 RAM 不可纠正 ECC 错误（优先级 4）· 01001b - 接收转发请求 RAM 可纠正 ECC 错误（优先级 25）· 01010b - 接收转发请求 RAM 不可纠正 ECC 错误（优先级 2）· 01011b - 接收完成 RAM 可纠正 ECC 错误（优先级 26）· 01100b - 接收完成 RAM 不可纠正 ECC 错误（优先级 3）· 01101b - 接收转发缓冲器上溢错误（优先级 6）· 01110b - 接收非转发缓冲器上溢错误（优先级 7）· 01111b - 接收完成缓冲器上溢错误（优先级 8）· 10000b - 流量控制协议错误（优先级 9）· 10001b - 检测到发射奇偶性错误（优先级 5）· 10010b - 接收到意外完成（优先级 20）· 10011b - 检测到完成超时（优先级 19）· 10100b - AXI4ST RQ INTFC 包丢弃（优先级 22）· 10101b - AXI4ST CC INTFC 包丢弃（优先级 23）· 10110b - AXI4ST CQ 毒化丢弃（优先级 24）· 10111b - axi2cfg_rq_parity_error_detected_i (优先级 1)· 11000b - axi2cfg_cc_parity_error_detected_i (优先级 0)· 11001b - 11111b - 保留 <p>注释：此信号可能并非对于所有 PCIe 链路宽度/速度配置都有效。请勿单独依赖此信号来判定错误。还可以通过解码 AER 寄存器来准确检测错误。</p>
cfg_local_error_valid	输出	1	<p>局部错误状况有效：当遇到 cfg_local_error_out[4:0] 中的任何错误时，块会激活此输出并保持 1 个周期。驱动到 1b 时，cfg_local_error_out[4:0] 会指示局部错误类型。此时将记录错误报告的优先级（用于并发错误）。</p> <p>注释：此信号可能并非对于所有 PCIe 链路宽度/速度配置都有效。请勿单独依赖此信号来判定错误。还可以通过解码 AER 寄存器来准确检测错误。</p>

表25：配置状态接口端口描述(续)

端口	I/O	宽度	描述
cfg_rx_pm_state	输出	2	当前RX有效状态功耗管理L0s状态：以下列出了编码，且当cfg_ltssm_state指示L0时有效： <ul style="list-style-type: none">· RX_NOT_IN_L0s = 0· RX_L0s_ENTRY = 1· RX_L0s_IDLE = 2· RX_L0s_FTS = 3
cfg_tx_pm_state	输出	2	当前TX有效状态功耗管理L0s状态：以下列出了编码，且当cfg_ltssm_state指示L0时有效： <ul style="list-style-type: none">· TX_NOT_IN_L0s = 0· TX_L0s_ENTRY = 1· TX_L0s_IDLE = 2· TX_L0s_FTS = 3
cfg_ltssm_state	输出	6	LTSSM状态。显示当前LTSSM状态： <ul style="list-style-type: none">· 00: Detect.Quiet· 01: Detect.Active· 02: Polling.Active· 03: Polling.Compliance· 04: Polling.Configuration· 05: Configuration.Linkwidth.Start· 06: Configuration.Linkwidth.Accept· 07: Configuration.Lanenum.Accept· 08: Configuration.Lanenum.Wait· 09: Configuration.Complete· 0A: Configuration.Idle· 0B: Recovery.RcvrLock· 0C: Recovery.Speed· 0D: Recovery.RcvrCfg· 0E: Recovery.Idle· 10: L0· 11-16: 保留· 17: L1.Entry· 18: L1.Idle· 19-1A: 保留· 20: 禁用· 21: Loopback_Entry_Master· 22: Loopback_Active_Master· 23: Loopback_Exit_Master· 24: Loopback_Entry_Slave· 25: Loopback_Active_Slave· 26: Loopback_Exit_Slave· 27: Hot_Reset· 28: Recovery_Equalization_Phase0· 29: Recovery_Equalization_Phase1· 2a: Recovery_Equalization_Phase2· 2b: Recovery_Equalization_Phase3

表25：配置状态接口端口描述(续)

端口	I/O	宽度	描述
cfg_rcb_status	输出	4	<p>RCB 状态。</p> <p>用于提供每个物理功能的链路控制寄存器中的读取完成边界(RCB)位的设置。在“Endpoint”模式下，位0指示物理功能0(PF0)的RCB，位1指示PF1的RCB，以此类推。在RC模式下，位0指示RP的链路控制寄存器的RCB设置，位1保留。</p> <p>对于每个位，值为0表示RCB为64字节，值为1表示128字节。</p>
cfg_dpa_substate_change	输出	4	<p>动态功耗分配子状态更改。</p> <p>在“Endpoint”模式下，当“Configuration Write”(配置写入)传输事务写入“Dynamic Power Allocation Control”(动态功耗分配控制)寄存器以修改器件的DPA功耗状态时，核会在其中一个输出上生成1个周期的脉冲。位0上发生脉冲表示针对PFO发生此类DPA事件，位1上发生脉冲则表示针对PF1发生同样的DPA事件。另2个位为保留位。这些输出在Root Port模式下无效。</p>
cfg_obff_enable	输出	2	<p>优化缓冲器刷新填充启用。</p> <p>此输出用于反映器件控制2寄存器中的“OBFF Enable”(OBFF启用)字段的设置。</p> <ul style="list-style-type: none">· 00: OBFF已禁用。· 01: OBFF已启用，使用报文信号Variation A。· 10: OBFF已启用，使用报文信号Variation B。· 11: OBFF已启用，使用WAKE#信号。
cfg_pl_status_change	输出	1	<p>此输出供核在Root Port模式下用于发出以下某一链路训练相关事件的信号：</p> <ul style="list-style-type: none">· 由于链路宽度或运行速度更改导致链路带宽更改，此更改仅在本地发起(并非由链路伙伴发起)，且链路未中断。此中断由链路控制寄存器中的“Link Bandwidth Management Interrupt Enable”(链路带宽管理中断启用)位启用。此中断的状态可从链路状态寄存器的“Link Bandwidth Management Status”(链路带宽管理状态)位读取；或者· 由于链路宽度或运行速度更改导致链路带宽自主更改，此更改由远程节点发起。此中断由链路控制寄存器中的“Link Autonomous Bandwidth Interrupt Enable”(链路自主带宽中断启用)位启用。此中断的状态可从链路状态寄存器的“Link Autonomous Bandwidth Status”(链路自主带宽状态)位读取；或者· 由于硬件接收到来自远程节点的链路均衡请求，因此它设置了链路状态2寄存器中的“Link Equalization Request”(链路均衡请求)位。此中断由链路控制3寄存器中的“Link Equalization Interrupt Enable”(链路均衡中断启用)位启用。此中断的状态可从链路状态2寄存器的“Link Equalization Request”(链路均衡请求)位读取。 <p>当核配置为Endpoint时，pl_interrupt输出无效。</p>
cfg_tph_requester_enable	输出	4	此输出的位0用于反映物理功能0的TPH请求器功能结构中的TPH请求器控制寄存器的TPH请求器使能位[8]的设置。位1对应于物理功能1。
cfg_tph_st_mode	输出	12	此输出的位[2:0]用于反映物理功能0的TPH请求器控制寄存器中的ST模式选择位的设置。位[5:3]用于反映PF1的相同寄存器字段的设置。
cfg_vf_tph_requester_enable	输出	252	此输出的每个位用于反映对应虚拟功能的TPH请求器功能结构中的TPH请求器控制寄存器的TPH请求器使能位8的设置。
cfg_vf_tph_st_mode	输出	756	此输出的位[2:0]用于反映物理功能0的TPH请求器控制寄存器中的ST模式选择位的设置。位[5:3]用于反映VF1的相同寄存器字段的设置，以此类推。

表 25：配置状态接口端口描述 (续)

端口	I/O	宽度	描述
pcie_tfc_nph_av	输出	4	<p>此输出用于指示核的发射端的非转发 TLP 的当前可用报头信用值。用户逻辑可先检查此输出，然后在请求器请求接口上发射非转发请求，以避免在没有信用值可用时阻塞此接口。编码包括：</p> <ul style="list-style-type: none">· 0000：无信用值可用· 0001：可用信用值为 1· 0010：可用信用值为 2· ...· 1110：可用信用值为 14· 1111：可用信用值不少于 15 <p>由于存在流水线延迟，此输出上的值不包含最近最多 8 个周期内的非转发请求所耗用的信用值。用户逻辑必须根据前几个时钟周期内非转发请求所耗用的信用值（如果有）来调整此输出上的值。</p>
pcie_tfc_npd_av	输出	4	<p>此输出用于指示核的发射端的非转发 TLP 的当前可用有效载荷信用值。用户逻辑会先检查此输出，然后在请求器请求接口上发射非转发请求，以避免在没有信用值可用时阻塞此接口。编码包括：</p> <ul style="list-style-type: none">· 0000：无信用值可用· 0001：可用信用值为 1· 0010：可用信用值为 2· ...· 1110：可用信用值不少于 14· 1111：可用信用值不少于 15 <p>由于存在流水线延迟，此输出上的值不包含最近最多 8 个时钟周期内用户逻辑发送的非转发请求所耗用的信用值。用户逻辑必须根据前几个时钟周期内非转发请求所耗用的信用值（如果有）来调整此输出上的值。</p>
pcie_rq_tag_av	输出	4	<p>此输出用于指示可供分配到核的 PCIe 主控制器端的非转发请求的可用标签数量。用户逻辑会先检查此输出，然后在请求器请求接口上发射非转发请求，以避免在没有标签可用时阻塞此接口。编码包括：</p> <ul style="list-style-type: none">· 0000：无标签可用· 0001：可用标签数为 1· 0010：可用标签数为 2· ...· 1110：可用标签数为 14· 1111：可用标签数不少于 15 <p>由于存在流水线延迟，此输出上的值不包含最近最多 8 个时钟周期内用户逻辑发送的非转发请求所耗用的标签。用户逻辑必须根据前几个时钟周期内所发送的非转发请求的数量（如果有）来调整此输出上的值。</p>

配置接收报文接口

“Configuration Received Message”（配置接收报文）接口用于向逻辑指示已接收到来自链路的可解码报文、与数据关联的参数以及报文的类型。下表定义了核的“配置接收报文”接口中的端口。

表 26：配置接收报文接口

端口	I/O	宽度	描述
cfg_msg_received	输出	1	<p>配置已接收可解码报文。</p> <p>当核接收到来自链路的可解码报文时，就会断言此输出有效，并保持 1 个或多个连续的时钟周期。其断言有效的持续时间是由报文类型来确定的。当 cfg_msg_received 为高电平时，在 1 个或多个周期内，核会在 cfg_msg_data[7:0] 输出上传输与报文关联的任意参数。下表列出了 cfg_msg_received 断言有效的周期数以及在每个周期内针对每一种报文类型在 cfg_msg_data[7:0] 上传输的参数。</p> <p>当启用 cfg_msg_received 接口时，核会在此接口上交付的连续 2 条报文之间插入至少 1 个周期的间隙。</p> <p>在 Vivado IDE 中进行核配置期间，“配置接收报文”接口必须保持启用。</p>
cfg_msg_received_data	输出	8	该总线用于传输与“接收报文”关联的任意参数。下表中列出了针对不同报文类型，该总线在每个周期内所承载的信息。
cfg_msg_received_type	输出	5	<p>接收报文类型。</p> <p>当 cfg_msg_received 为高电平时，这 5 个位用于指示核所发出的报文类型。上表中列出了各报文类型。</p>

表 27：接收报文接口上的报文类型编码

cfg_msg_received_type[4:0]	报文类型
0	ERR_COR
1	ERR_NONFATAL
2	ERR_FATAL
3	Assert_INTA
4	Deassert_INTA
5	Assert_INTB
6	Deassert_INTB
7	Assert_INTC
8	Deassert_INTC
9	Assert_INTD
10	Deassert_INTD
11	PM_PME
12	PM_E_TO_Ack
13	PM_E_Turn_Off
14	PM_Active_State_Nak
15	Set_Slot_Power_Limit
16	时延容限报告 (LTR)
17	保留
18	解锁
19	Vendor Defined 类型 0
20	Vendor Defined 类型 1
25 - 31	保留

表 28：接收报文接口上的报文参数

报文类型	cfg_msg_received 断言有效的周期数	cfg_msg_received_data[7:0] 上传输的参数
ERR_COR、ERR_NONFATAL、ERR_FATAL	2	周期 1：请求器 ID、总线编号 周期 2：请求器 ID、器件/功能编号
Assert_INTx、Deassert_INTx	2	周期 1：请求器 ID、总线编号 周期 2：请求器 ID、器件/功能编号
PM_PME、PME_TO_Ack、PME_Turn_off、PM_Active_State_Nak	2	周期 1：请求器 ID、总线编号 周期 2：请求器 ID、器件/功能编号
Set_Slot_Power_Limit	6	周期 1：请求器 ID、总线编号 周期 2：请求器 ID、器件/功能编号 周期 3：有效载荷的位 [7:0] 周期 4：有效载荷的位 [15:8] 周期 5：有效载荷的位 [23:16] 周期 6：有效载荷的位 [31:24]
时延容限报告 (LTR)	6	周期 1：请求器 ID、总线编号 周期 2：请求器 ID、器件/功能编号 周期 3：嗅探时延的位 [7:0] 周期 4：嗅探时延的位 [15:8] 周期 5：无嗅探时延的位 [7:0] 周期 6：无嗅探时延的位 [15:8]
解锁	2	周期 1：请求器 ID、总线编号 周期 2：请求器 ID、器件/功能编号
Vendor_Defined 类型 0	不存在数据时为 4 个周期，存在数据时则为 8 个周期。	
Vendor_Defined 类型 1	不存在数据时为 4 个周期，存在数据时则为 8 个周期。	
		周期 1：请求器 ID、总线编号 周期 2：请求器 ID、器件/功能编号 周期 3：供应商 ID[7:0] 周期 4：供应商 ID[15:8] 周期 5：有效载荷的位 [7:0] 周期 6：有效载荷的位 [15:8] 周期 7：有效载荷的位 [23:16] 周期 8：有效载荷的位 [31:24]
		周期 1：请求器 ID、总线编号 周期 2：请求器 ID、器件/功能编号 周期 3：供应商 ID[7:0] 周期 4：供应商 ID[15:8] 周期 5：有效载荷的位 [7:0] 周期 6：有效载荷的位 [15:8] 周期 7：有效载荷的位 [23:16] 周期 8：有效载荷的位 [31:24]

配置发射报文接口

“Configuration Transmit Message”（配置发射报文）接口供用户应用用于向核发射报文。用户应用将向核提供发射报文类型和数据信息，核则以 `done` 信号作为响应。下表定义了核的“配置发射报文”接口中的端口。

表 29：配置发射报文接口

端口	I/O	宽度	描述
cfg_msg_transmit	输入	1	配置发射编码报文。 此信号与提供编码报文类型的 cfg_msg_transmit_type 和提供与报文关联的可选数据的 cfg_msg_transmit_data 一起断言有效，直至在响应中断断言 cfg_msg_transmit_done 有效为止。
cfg_msg_transmit_type	输入	3	配置发射编码报文类型。 指示要发射的 PCI Express 报文类型。受支持的编码包括： <ul style="list-style-type: none">· 000b：时延容限报告 (LTR)· 001b：最优化闪存刷新/填充 (OBFF)· 010b：设置插槽功耗限制 (SSPL)· 011b：功耗管理 (PM PME)· 100b - 111b：保留
cfg_msg_transmit_data	输入	32	配置发射编码报文数据。 指示与特定报文类型关联的报文数据。 000b: LTR - <ul style="list-style-type: none">· cfg_msg_transmit_data[31] < 噪探时延要求· cfg_msg_transmit_data[28:26] < 噪探时延缩放比例· cfg_msg_transmit_data[25:16] < 噪探时延值· cfg_msg_transmit_data[15] < 无噪探时延要求· cfg_msg_transmit_data[12:10] < 无噪探时延缩放比例· cfg_msg_transmit_data[9:0] < 无噪探时延值 001b: OBFF - <ul style="list-style-type: none">· cfg_msg_transmit_data[3:0] < OBFF 代码 010b: SSPL - <ul style="list-style-type: none">· cfg_msg_transmit_data[9:0] < {插槽功耗限制缩放比例, 插槽功耗限制值} 011b: PM_PME - <ul style="list-style-type: none">· cfg_msg_transmit_data[1:0] < PF1, PF0· cfg_msg_transmit_data[9:4] < VF5、VF4、VF3、VF2、VF1、VF0, 其中 1 个或多个 PF 或 VF 可同时发出 PM_PME 信号。 100b - 111b: 保留
cfg_msg_transmit_done	输出	1	配置发射编码报文已完成。 请求完成后，此端口断言有效以作为对于 cfg_msg_transmit 断言有效的响应，并保持 1 个周期。

配置流量控制接口

下表定义了核的“配置流量控制”接口中的端口。

表 30：配置流量控制接口

端口	I/O	宽度	描述
cfg_fc_ph	输出	8	转发报头流量控制信用值。 此输出可提供转发报头流量控制信用值的数值。此多路复用输出可用于显示与核所维护的转发报头信用值相关的各种流量控制参数和变量。通过 cfg_fc_sel[2:0] 输入可选择要在该核上显示的流量控制信息。

表 30：配置流量控制接口（续）

端口	I/O	宽度	描述
cfg_fc_pd	输出	12	转发数据流量控制信用值。 此输出可提供转发数据流量控制信用值的数值。此多路复用输出可用于显示与核所维护的转发数据信用值相关的各种流量控制参数和变量。通过 cfg_fc_sel[2:0] 输入可选择要在该核上显示的流量控制信息。
cfg_fc_nph	输出	8	非转发报头流量控制信用值。 此输出可提供非转发报头流量控制信用值的数值。此多路复用输出可用于显示与核所维护的非转发报头信用值相关的各种流量控制参数和变量。通过 cfg_fc_sel[2:0] 输入可选择要在该核上显示的流量控制信息。
cfg_fc_npd	输出	12	非转发数据流量控制信用值。 此输出可提供非转发数据流量控制信用值的数值。此多路复用输出可用于显示与核所维护的非转发数据信用值相关的各种流量控制参数和变量。通过 cfg_fc_sel[2:0] 输入可选择要在该核上显示的流量控制信息。
cfg_fc_cplh	输出	8	完成报头流量控制信用值。 此输出可提供完成报头流量控制信用值的数值。此多路复用输出可用于显示与核所维护的完成报头信用值相关的各种流量控制参数和变量。通过 cfg_fc_sel[2:0] 输入可选择要在该核上显示的流量控制信息。
cfg_fc_cpld	输出	12	完成数据流量控制信用值。 此输出可提供完成数据流量控制信用值的数值。此多路复用输出可用于显示与核所维护的完成数据信用值相关的各种流量控制参数和变量。通过 cfg_fc_sel[2:0] 可选择要在该核上显示的流量控制信息。
cfg_fc_sel	输入	3	流量控制参考信息选择。 这些输入用于选择要在核的 cfg_fc_* 输出上显示的流量控制的类型。针对这些输入的不同设置可访问的各种流量控制参数和变量包括： <ul style="list-style-type: none">· 000：链路伙伴可用的接收信用值001：保留010：链路伙伴耗用的接收信用值011：保留100：可用发射用户信用值101：发射信用值上限110：已用发射信用值111 == 保留 该值表示接收器 FIFO 中的实际未使用的信用值，建议是仅将该值用作为接收器 FIFO 充盈度的近似指示，与颁发的初始信用限值相对，例如， $\frac{1}{4}$ 满、 $\frac{1}{2}$ 满、 $\frac{3}{4}$ 满、全满。 可用发射信用值无限 (cfg_fc_sel == 3'b100) 针对报头信用值和数据信用值分别表示为 8'h80 和 12'h800。对于所有其它 cfg_fc_sel 选择，无限信用值针对报头类别和数据类别分别表示为 8'h00 和 12'h000。

配置接口

此接口为受限使用，针对 IP 的通用用例或操作不予支持。

配置控制接口

“Configuration Control”（配置控制）接口信号允许在用户应用与核之间交换各种信息。用户应用使用此信息来执行以下操作：

- 设置配置空间。
- 指示是否发生可纠正的错误或不可纠正的错误。
- 设置器件序列号。
- 设置下游总线、器件和功能编号。
- 接收各项功能配置信息。

当发生功耗状态变更或功能级别复位时，此接口还可提供用户应用与核之间的握手。

表 31：配置控制接口端口描述

端口	I/O	宽度	描述
cfg_hot_reset_in	输入	1	配置热复位输入 在 RP 模式下，断言有效会将 LTSSM 转换为热复位状态，即高电平有效。
cfg_hot_reset_out	输出	1	配置热复位输出 在 EP 模式下，断言有效表示 EP 已转换到热复位状态，即高电平有效。
cfg_config_space_enable	输入	1	配置配置空间使能 当此输入在端点模式下设置为 0 时，核会生成 CRS 完成以响应配置请求。当由于属性更改导致从 DRP 加载核配置寄存器时，此端口应保持断言无效状态。这样可避免核在所有寄存器都完成加载前就响应配置请求。当配置空间枚举之前无需修改配置寄存器的上电默认值时，此输入可为高电平。此输入不适用于根端口模式。
cfg_dsn	输入	64	配置器件序列号 表示应传输到 PF0 上的“Device Serial Number Capability”（器件序列号功能）的值。位 [31:0] 传输到首个（下位）Dword（对应功能的字节偏移 0x4h），字节 [63:32] 则传输到第二个（上位）Dword（对应功能的字节偏移 0x8h）。用户逻辑更新 cfg_dsn 后，新的 cfg_dsn 应显示在“Extended Configuration Space”（扩展配置空间）上。无需其它限定控制信号。
cfg_ds_bus_number	输入	8	配置下游总线编号 下游端口：提供下游端口的请求器 ID (RID) 的总线编号部分。它可在核内部生成的 TLP 中使用，例如，UR 完成和功耗管理报文；它不影响 AXI 接口上存在的 TLP。 上游端口：无角色。
cfg_ds_device_number	输入	5	配置下游器件编号 下游端口：提供下游端口的 RID 的器件编号部分。它可在核内部生成的 TLP 中使用，例如，UR 完成和功耗管理报文；它不影响 TRN 接口上存在的 TLP。 上游端口：无角色。
cfg_ds_function_number	输入	3	配置下游功能编号 下游端口：提供下游端口的 RID 的功能编号部分。它可在核内部生成的 TLP 中使用，例如，UR 完成和功耗管理报文；它不影响 TRN 接口上存在的 TLP。 上游端口：无角色。

表 31：配置控制接口端口描述 (续)

端口	I/O	宽度	描述
cfg_power_state_change_ack	输入	1	<p>配置功耗状态确认</p> <p>当准备好转换到配置写入请求所请求的低功耗状态时，必须将此核输入断言有效并保持 1 个周期，以响应 cfg_power_state_change_interrupt 的断言有效。如果用户应用无需针对配置写入传输事务延迟返回完成包（导致功耗状态更改），则可将此输入永久保持为高电平。</p>
cfg_power_state_change_interrupt	输出	1	<p>功耗状态更改中断</p> <p>当物理功能或虚拟功能的功耗状态通过写入其功耗管理控制寄存器变为 D1 或 D3 状态时，核可断言此输出有效。核会将此输出保持高电平，直至用户应用断言核的 cfg_power_state_change_ack 输入有效为止。当 cfg_power_state_change_interrupt 保持高电平时，针对所接收到的任何暂挂配置读取或写入传输事务，核不会返回完成包。这样即可使导致状态更改的配置写入传输事务延迟完成，直至用户应用准备好转变为低功耗状态为止。当 cfg_power_state_change_interrupt 断言有效时，在 cfg_ext_function_number[7:0] 输出上会提供与配置写入传输事务关联的功能编号。当用户应用断言 cfg_power_state_change_ack 有效时，发生状态更改的功能的新状态会反映在核的 cfg_function_power_state (对应 PF) 或 cfg_vf_power_state (对应 VF) 输出上。</p>
cfg_ds_port_number	输入	8	<p>配置下游端口编号</p> <p>在链路功能寄存器中提供端口编号字段。</p>
cfg_err_cor_in	输入	1	<p>检测到可纠正的错误</p> <p>用户应用激活此输入并保持 1 个周期，表明在用户逻辑中检测到可纠正的错误，并且需通过 PCI Express 高级错误报告 (AER) 机制将此错误报告为内部错误。在响应中，核会在所有已启用的功能的 AER 可纠正错误状态寄存器中设置已纠正的内部错误状态位，还会发送错误报文（如果已启用此操作）。此错误并非特定于功能的错误。</p> <p>注释：此信号可能并非对于所有 PCIe 链路宽度/速度配置都有效。请勿单独依赖此信号来判定错误。还可以通过解码 AER 寄存器来准确检测错误。</p>
cfg_err_cor_out	输出	1	<p>检测到可纠正的错误</p> <p>在端点模式下，当块检测到可纠正的错误并且其报告未被屏蔽时，就会激活此输出并保持 1 个周期。当启用多个功能时，这是所有功能的器件状态寄存器中的可纠正错误状态位的逻辑 OR。</p> <p>注释：此信号可能并非对于所有 PCIe 链路宽度/速度配置都有效。请勿单独依赖此信号来判定错误。还可以通过解码 AER 寄存器来准确检测错误。</p>
cfg_err_fatal_out	输出	1	<p>检测到致命错误</p> <p>在端点模式下，当块检测到致命错误并且其报告未被屏蔽时，就会激活此输出并保持 1 个周期。当启用多个功能时，此输出是所有功能的器件状态寄存器中的致命错误状态位的逻辑 OR。</p> <p>在根端口模式下，当检测到局部致命错误并且其报告未被屏蔽时，就会激活此输出。此输出并不会响应远程器件使用 PCI Express 错误报文发出的任何错误信号。这些错误报文将通过报文接口交付给用户。</p> <p>注释：此信号可能并非对于所有 PCIe 链路宽度/速度配置都有效。请勿单独依赖此信号来判定错误。还可以通过解码 AER 寄存器来准确检测错误。</p>

表 31：配置控制接口端口描述 (续)

端口	I/O	宽度	描述
cfg_err_nonfatal_out	输出	1	<p>检测到非致命错误 在端点模式下，当块检测到非致命错误并且其报告未被屏蔽时，就会激活此输出并保持 1 个周期。当启用多个功能时，此输出是所有功能的器件状态寄存器中的非致命错误状态位的逻辑 ORs。</p> <p>在根端口模式下，当检测到局部非致命错误并且其报告未被屏蔽时，就会激活此输出。此输出并不会响应远程器件使用 PCI Express 错误报文发出的任何错误信号。这些错误报文将通过报文接口进行交付。</p> <p>注释：此信号可能并非对于所有 PCIe 链路宽度/速度配置都有效。请勿单独依赖此信号来判定错误。还可以通过解码 AER 寄存器来准确检测错误。</p>
cfg_err_uncor_in	输入	1	<p>检测到不可纠正的错误 用户应用激活此输入并保持 1 个周期，表明在用户逻辑中检测到不可纠正的错误，并且需通过 PCI Express 高级错误报告机制将此错误报告为内部错误。在响应中，核会在所有已启用的功能的 AER 不可纠正错误状态寄存器中设置未纠正的内部错误状态位，还会发送错误报文（如果已启用此操作）。此错误并非特定于功能的错误。</p> <p>注释：此信号可能并非对于所有 PCIe 链路宽度/速度配置都有效。请勿单独依赖此信号来判定错误。还可以通过解码 AER 寄存器来准确检测错误。</p>
cfg_flr_done	输入	4	<p>功能级别复位完成 用户应用已完成虚拟功能的复位操作后必须断言此输入有效。这样会导致核针对物理功能 i 断言 cfg_flr_in_process 无效，并重新启用对该物理功能的配置访问。如果特定物理功能的 cfg_flr_in_process = 1，那么该核将向此物理功能发出针对配置请求的 CRS，直至 cfg_flr_done 不再断言有效为止。</p>
cfg_vf_flr_done	输入	1	<p>针对虚拟功能的功能级别复位已完成 用户应用已完成虚拟功能的复位操作后必须断言此输入有效。这样会导致核针对功能 i 断言 cfg_vf_flr_in_process 无效，并重新启用对该虚拟功能的配置访问。如果特定虚拟函数的 cfg_vf_flr_in_process = 1，那么该核将向此虚拟功能发出针对配置请求的 CRS，直至 cfg_vf_flr_done 不再断言有效为止。</p>
cfg_vf_flr_func_num	输入	8	<p>针对虚拟功能 i 的功能级别复位已完成。 当虚拟功能 i 的复位操作完成时，此用户应用会驱动此输入上的有效虚拟功能数，同时断言 cfg_vf_flr_done 有效。 有效输入为 8'h04-8'hFF（针对 VF0-VF251）。值 8'h00-8'h03 为保留值。</p>
cfg_flr_in_process	输出	4	<p>正在执行功能级别复位 当主机通过其配置空间中的 FLR 位启动物理功能 i 的复位时，核会断言该总线的位 i 有效。该核会继续使输出保持高电平，直至用户为对应物理功能设置对应 cfg_flr_done 输入以指示复位操作完成为止。</p>
cfg_vf_flr_in_process	输出	252	<p>针对虚拟功能正在执行功能级别复位 当主机通过其配置空间中的 FLR 位启动虚拟功能 i 的复位时，核会断言该总线的位 i 有效。该核会继续使输出保持高电平，直至用户为对应功能设置 cfg_vf_flr_done 输入并驱动 cfg_vf_flr_func_num 以指示复位操作完成为止。</p>

表 31：配置控制接口端口描述 (续)

端口	I/O	宽度	描述
cfg_req_pm_transition_l2_3_ready	输入	1	当核配置为端点时，用户应用会断言此输入有效以将核的功耗管理状态转变为 L23_READY (请参阅《PCI Express 规范》第 5 章)，另请参阅《PCI-SIG 规范》(https://www.pcisig.com/specifications) 以获取功耗管理的详细描述。此操作是在核中的 PCI 功能置于 D3 状态并且用户应用确认来自根联合体的 PME_Turn_Off 报文之后执行的。断言此输入有效会导致链路转变为 L3 状态，并且需要硬复位才能恢复操作。如果链路无需转换到 L3，此输入可硬连线到 0。在根联合体模式下不使用此输入。
cfg_link_training_enable	输入	1	此输入必须设置为 1 才能支持链路训练状态机 (LTSSM) 初始化链路。将其设置为 0 会导致强制 LTSSM 保持处于 Detect.Quiet 状态。
cfg_bus_number	输出	8	显示从接收到的 CfgWr Type0 捕获的总线数量。仅在端点配置中激活。
cfg_vend_id	输入	16	配置供应商 ID： 表示应传输到所有 PF 上的 “PCI Capability Structure Vendor ID” (PCI 功能结构供应商 ID) 字段的值。
cfg_subsys_vend_id	输入	16	配置子系统供应商 ID： 表示应传输到所有 PF 上的 “Type 0 PCI Capability Structure Subsystem Vendor ID” (类型 0 PCI 功能结构子系统供应商 ID) 字段的值。
cfg_dev_id_pf0	输入	16	配置器件 ID PF0： 表示应传输到 PF0 上的 “PCI Capability Structure Device ID” (PCI 功能结构器件 ID) 字段的值。
cfg_dev_id_pf1	输入	16	配置器件 ID PF1： 表示应传输到 PF1 上的 “PCI Capability Structure Device ID” 字段的值。
cfg_dev_id_pf2	输入	16	配置器件 ID PF2： 表示应传输到 PF2 上的 “PCI Capability Structure Device ID” 字段的值。
cfg_dev_id_pf3	输入	16	配置器件 ID PF3： 表示应传输到 PF3 上的 “PCI Capability Structure Device ID” 字段的值。
cfg_rev_id_pf0	输入	8	配置版本 ID PF0： 表示应传输到 PF0 上的 “PCI Capability Structure Revision ID” (PCI 功能结构版本 ID) 字段的值。
cfg_rev_id_pf1	输入	8	配置版本 ID PF1： 表示应传输到 PF1 上的 “PCI Capability Structure Revision ID” 字段的值。
cfg_rev_id_pf2	输入	8	配置版本 ID PF2： 表示应传输到 PF2 上的 “PCI Capability Structure Revision ID” 字段的值。
cfg_rev_id_pf3	输入	8	配置版本 ID PF3： 表示应传输到 PF3 上的 “PCI Capability Structure Revision ID” 字段的值。
cfg_subsys_id_pf0	输入	16	配置子系统 ID PF0： 表示应传输到 PF0 上的 “Type 0 PCI Capability Structure Subsystem ID” (类型 0 PCI 功能结构子系统 ID) 字段的值。
cfg_subsys_id_pf1	输入	16	配置子系统 ID PF1： 表示应传输到 PF1 上的 “Type 0 PCI Capability Structure Subsystem ID” 字段的值。

表 31：配置控制接口端口描述 (续)

端口	I/O	宽度	描述
cfg_subsys_id_pf2	输入	16	配置子系统 ID PF2： 表示应传输到 PF2 上的 “Type 0 PCI Capability Structure Subsystem ID” 字段的值。
cfg_subsys_id_pf3	输入	16	配置子系统 ID PF3： 表示应传输到 PF3 上的 “Type 0 PCI Capability Structure Subsystem ID” 字段的值。

配置中断控制器接口

配置中断控制器接口允许用户应用设置传统 PCIe 中断、MSI 中断或 MSI-X 中断。核可通过配置中断发送信号和失败信号来提供中断状态。下表定义了与核的配置中断控制器接口关联的接口端口。

遗留中断接口

表 32：遗留中断接口端口描述

名称	I/O	宽度	描述
cfg_interrupt_int	输入	4	配置 INTx 矢量：当核配置为端点 (EP) 时，这 4 个输入供用户应用用于将中断信号从其任意 PCI 功能发送到 RC，此操作使用的是 PCI Express 的传统 PCI Express 中断交付机制。这 4 项输入分别对应于 PCI 总线的 INTA、INTB、INTC 和 INTD。其中任一信号断言有效将导致核向外发送 Assert_INTx 报文，此信号断言无效则会导致核发射 Deassert_INTx 报文。
cfg_interrupt_sent	输出	1	配置 INTx 已发送：此输出上出现脉冲表示核已发送 INTx Assert 或 Deassert 报文，以作为对任一 cfg_interrupt_int 输入状态改变的响应。
cfg_interrupt_pending	输入	4	配置 INTx 中断暂挂：（按功能）表示来自用户的暂挂中断。cfg_interrupt_pending[0] 对应于 Function #0。其中每项输入均连接至对应功能的 PCI 状态寄存器的中断暂挂位。

MSI 中断接口

表 33：MSI 中断接口端口描述

名称	I/O	宽度	描述
cfg_interrupt_msi_enable	输出	4	配置中断 MSI 功能启用 表示报文信号中断 (MSI) 报文信号已启用（按功能）。这些输出可反映物理功能 0 - 3 的 MSI 控制寄存器的“MSI Enable”（MSI 启用）位的设置。
cfg_interrupt_msi_int	输入	32	配置中断 MSI 矢量 在端点模式下配置为支持 MSI 中断时，这些输入用于表示与从用户逻辑到核的 PCI 功能（物理或虚拟）关联的 32 个不同中断状况对应的信号。必须在 cfg_interrupt_msi_function_number 输入上指定功能编号。在 cfg_interrupt_msi_function_number 输入上设置功能编号后，用户逻辑必须激活其中任一信号并保持 1 个周期，以供发射中断。用户逻辑在同一周期内最多只能激活这 32 个中断输入中的 1 个输入。核会在内部寄存中断状况，寄存位置为 cfg_interrupt_msi_int 中的任意位的 0 到 1 转换。断言中断有效后，用户逻辑必须等待至收到来自核 cfg_interrupt_msi_sent 或 cfg_interrupt_msi_fail 指示之后才能断言新中断有效。

表 33：MSI 中断接口端口描述（续）

名称	I/O	宽度	描述
cfg_interrupt_msi_function_number	输入	8	配置 MSI 启动功能 指示用于启动 MSI 中断的端点功能编号。 <ul style="list-style-type: none">· 8'h00 - 8'h03: PF 0 - PF 3· 8'h04 - 8'hFF: VF 0 - VF 252· 其它编码均为保留值。
cfg_interrupt_msi_sent	输出	1	配置中断 MSI 中断已发送 核会在此输入上生成 1 个周期的脉冲，以表示在链路上已发射 MSI 或 MSI-X 中断报文的对应信号。用户逻辑必须等待收到此脉冲后才能向核发出另一个中断状况信号。
cfg_interrupt_msi_fail	输出	1	配置中断 MSI 中断操作失败 此输入上出现 1 个周期的脉冲表示 MSI 中断报文在链路上发射前已异常中止。在此情况下，用户逻辑必须重新发射 MSI 中断。
cfg_interrupt_msi_mmnable	输出	12	配置中断 MSI 功能多报文使能 当核在端点模式下配置为支持 MSI 中断时，这些输出由与物理功能关联的 MSI 控制寄存器的“Multiple Message Enable”（多报文使能）位驱动。这些位会通过编码来表示为对应功能分配的 MSI 中断矢量的数量。位 [2:0] 对应于物理功能 0，位 [5:3] 对应于 PF 1，以此类推。这 3 个位的有效编码为： <ul style="list-style-type: none">· 000b: 1 个矢量· 001b: 2 个矢量· 010b: 4 个矢量· 011b: 8 个矢量· 100b: 16 个矢量· 101b: 32 个矢量
cfg_interrupt_msi_pending_status	输入	32	配置 MSI 中断暂挂状态 为用户提供这些输入表示与物理功能关联的 MSI 中断的中断暂挂状态。当与 PF 关联的 MSI 中断的状态发生更改时，用户必须在这些输入上设置新中断状态，并在 cfg_interrupt_msi_pending_status_function_num 输入上设置对应的功能编号，激活 cfg_interrupt_msi_pending_status_data_enable 输入并保持 1 个周期。随后，核会在其对应物理功能的 MSI 暂挂位寄存器中锁存新状态。
cfg_interrupt_msi_pending_status_function_num	输入	2	配置中断 MSI 暂挂目标功能编号 <ul style="list-style-type: none">· 00 = PF 0· 01 = PF 1· 10 = PF 2· 11 = PF 3 当用户在 cfg_interrupt_msi_pending_status 输入上设置中断状态时，此输入用于识别功能编号。
cfg_interrupt_msi_pending_status_data_enable	输入	1	配置中断 MSI 暂挂数据有效 用户应用随 cfg_interrupt_msi_pending_status 值和 cfg_interrupt_msi_pending_status_function_num 值一起断言此信号有效，以更新对应功能中的 MSI 暂挂位。
cfg_interrupt_msi_mask_update	输出	1	配置中断 MSI 功能掩码已更新 当任何已启用的 PF 的 MSI 掩码寄存器值发生更改时，SR-IOV 核会断言此信号有效并保持 1 个周期。随后，用户可以从 cfg_interrupt_msi_data 输出读取新的掩码设置。

表 33：MSI 中断接口端口描述（续）

名称	I/O	宽度	描述
cfg_interrupt_msi_select	输入	2	配置中断 MSI 选择 这些输入用于选择功能编号，以便从核中读取 MSI 掩码寄存器设置。值 0 - 3 分别对应于物理功能 0 - 3。在 1 个周期后，所选 PF 的 MSI 掩码寄存器内容会显示在 cfg_interrupt_msi_data 输出上。
cfg_interrupt_msi_data	输出	32	配置中断 MSI 数据 这些输出可反映 cfg_interrupt_msi_select 输入所选的物理功能的 MSI 掩码寄存器设置。
cfg_interrupt_msi_attr	输入	3	配置中断 MSI TLP 属性 这些位支持您设置属性位，用于 MSI 和 MSI-X 中断请求。 <ul style="list-style-type: none">· 位 0 为 “No Snoop”（无嗅探）位。· 位 1 为 “Relaxed Ordering”（宽松排序）位。· 位 2 为 “ID-Based Ordering”（基于 ID 排序）位。 核会在 cfg_interrupt_msi_int 位（使用 MSI 时）或 cfg_interrupt_msix_int（使用 MSI-X 时）上执行 0 到 1 转换过程中对这些位进行采样。
cfg_interrupt_msi_tph_present	输入	1	配置中断 MSI/MSI-X TPH 已预发送 表示在 MSI/MSI-X 中断请求中存在可选传输事务处理提示（TPH）。用户应用断言 cfg_interrupt_msi_int 位（使用 MSI 时）或 cfg_interrupt_msix_int（使用 MSI-X 时）有效时必须设置该位，前提是在 MSI 或 MSI-X 传输事务中包含 TPH。
cfg_interrupt_msi_tph_type	输入	2	配置中断 MSI/MSI-X TPH 类型 当 cfg_interrupt_msi_tph_present 为 1'b1 时，这 2 个位用于提供与提示关联的 2 位类型。核会根据使用的是 MSI 中断还是 MSI-X 中断，在 cfg_iminterrupt_msi_int 或 cfg_iminterrupt_msix_int 的任意位上执行 0 到 1 的转换过程中对这些位进行采样。
cfg_interrupt_msi_tph_st_tag	输入	8	配置中断 MSI/MSI-X TPH 导向标签 断言 cfg_interrupt_msi_tph_present 有效时，必须在 cfg_interrupt_msi_tph_st_tag[7:0] 上设置与提示关联的“Steering Tag”（导向标签）。核会根据使用的是 MSI 中断还是 MSI-X 中断，在 cfg_iminterrupt_msi_int 或 cfg_iminterrupt_msix_int 的任意位上执行 0 到 1 的转换过程中对这些位进行采样。

MSI-X 中断外部接口

表 34：MSI-X 中断外部接口端口描述

名称	I/O	宽度	描述
cfg_interrupt_msix_enable	输出	4	配置中断 MSI-X 功能启用 这些输出可反映物理功能 0 - 3 的 MSI-X 控制寄存器的 MSI-X 使能位的设置。
cfg_interrupt_msix_mask	输出	4	配置中断 MSI-X 功能掩码 这些输出可反映物理功能 0 - 3 的 MSI-X 控制寄存器的 MSI-X 功能掩码位的设置。
cfg_interrupt_msix_vf_enable	输出	252	来自 VF 的配置中断 MSI-X 启用 这些输出可反映虚拟功能 0 - 251 的 MSI-X 控制寄存器的 MSI-X 使能位的设置。
cfg_interrupt_msix_vf_mask	输出	252	配置中断 MSI-X VF 掩码 这些输出可反映虚拟功能 0 - 251 的 MSI-X 控制寄存器的 MSI-X 功能掩码位的设置。

表 34：MSI-X 中断外部接口端口描述 (续)

名称	I/O	宽度	描述
cfg_interrupt_msix_address	输入	64	配置中断 MSI-X 地址 当核配置为支持 MSI-X 中断并且在用户存储器中实现 MSI-X 表时，该总线可供用户逻辑用于传达将用于生成 MSI-X 中断的地址。
cfg_interrupt_msix_data	输入	32	配置中断 MSI-X 数据 当核配置为支持 MSI-X 中断并且在用户存储器中实现 MSI-X 表时，该总线可供用户逻辑用于传达将用于生成 MSI-X 中断的数据。
cfg_interrupt_msix_int	输入	1	配置中断 MSI-X 数据有效 用户断言此信号有效表示用户请求发送 MSI-X 中断。用户必须将识别信息置于指定输入上之后才能断言此中断有效。 当 MSI-X 表和暂挂位阵列均在用户存储器中实现时，识别信息由中断的存储器地址、数据和源功能编号组成。 这些信息必须分别置于 cfg_interrupt_msix_address[63:0]、cfg_interrupt_msix_data[31:0] 和 cfg_interrupt_msi_function_number[7:0] 上。核会在内部寄存这些参数，寄存位置为 cfg_interrupt_msix_int 的 0 到 1 转换。 当 MSI-X 表和暂挂位阵列均由核来实现时，识别信息由中断的源功能编号与中断矢量组成。 这些信息必须分别置于 cfg_interrupt_msi_function_number[7:0] 和 cfg_interrupt_msi_int[31:0] 上。 cfg_interrupt_msi_int[31:0] 的位 i 表示中断矢量 i，当 cfg_interrupt_msix_int 断言有效时，只能将该总线中的 1 个位设为 1。 断言中断有效后，用户逻辑必须等待至收到来自核 cfg_interrupt_msi_sent 或 cfg_interrupt_msi_fail 指示之后才能断言新中断有效。

表 34：MSI-X 中断外部接口端口描述（续）

名称	I/O	宽度	描述
cfg_interrupt_msix_vec_pending	输入	2	<p>配置中断 MSI-X 暂挂位查询/清除 仅当核配置为包含 MSI-X 表和暂挂位阵列时，才使用这些模式位。当 cfg_interrupt_msix_int 断言有效以发送 MSI-X 中断时，通过设置这 2 个位即可在与选定功能和中断矢量关联的 MSI-X 暂挂位上执行某些操作。适用模式包括：</p> <ul style="list-style-type: none"> 00b：正常中断生成。如果断言 cfg_interrupt_msix_int 有效时与矢量关联的掩码位为 0，那么核会在链路上发射 MSI-X 请求 TLP。如果掩码位为 1，那么核不会立即发送中断，而是改为在其 MSI-X 暂挂位阵列中设置与中断矢量关联的暂挂位（后续当清除掩码后再发射 MSI-X 请求 TLP）。在上述 2 种情况下，核会断言 cfg_interrupt_msi_sent 有效，并保持 1 个周期，以指示已接受中断请求。用户可通过对 cfg_interrupt_msix_vec_pending_status 输出进行采样来区分这 2 种情况，此输出会反映对应于中断矢量的 MSI-X 暂挂位的当前设置。 01b：暂挂位查询。在此模式下，当 cfg_interrupt_msix_int 的任一位断言有效时，核会将此视为查询其暂挂位状态的请求来进行处理。用户还必须将要查询的暂挂位的功能编号置于 cfg_interrupt_msi_function_number 输入上。核不会在响应中发射 MSI-X 请求，而是断言 cfg_interrupt_msi_sent 有效并保持 1 个周期，同时在 cfg_interrupt_msix_vec_pending_status 输出上显示 MSI-X 暂挂位的状态。 10b：暂挂位清除。在此模式下，当 cfg_interrupt_msix_int 的任一位断言有效时，核会将此视为清除其暂挂位的请求来进行处理。用户还必须将要查询的暂挂位的功能编号置于 cfg_interrupt_msi_function_number 输入上。核不会在响应中发射 MSI-X 请求，而是改为清除矢量的 MSI-X 暂挂位（如果已设置），并激活 cfg_interrupt_msi_sent 并保持 1 个周期以作为确认。核还会在 cfg_interrupt_msix_vec_pending_status 输出上提供 MSI-X 暂挂位的先前状态，用户可通过对其进行采样来判定核是否已清除暂挂位，然后再发出用户请求（因为暂挂中断实际已发射）。此模式可用于实现 MSI-X 中断的“轮询模式”，即永久屏蔽中断，软件会轮询暂挂位以对中断进行检测并维护。处理完成每个中断后，可通过此接口清除暂挂位。
cfg_interrupt_msix_vec_pending_status	输出	1	<p>配置中断 MSI-X 暂挂位状态 此输出可提供与 MSI-X 中断关联的暂挂位的状态，以作为使用 cfg_interrupt_msix_vec_pending 输入进行查询的响应。 仅当核配置为包含 MSI-X 表和暂挂位阵列时，此位才会处于活动状态。</p>

MSI-X 中断内部接口

表 35：MSI-X 中断内部接口端口描述

名称	I/O	宽度	描述
cfg_interrupt_msi_int	输入	8	该核支持每个功能 8 个矢量，它属于独热编码 (one-hot encoding)，因此每个位对应于 1 个矢量。请参阅 MSI 中断接口 中的描述。
cfg_interrupt_msi_function_number	输入	8	请参阅 MSI 中断接口 中的描述。
cfg_interrupt_msi_attr	输入	3	请参阅 MSI 中断接口 中的描述。
cfg_interrupt_msi_tph_present	输入	1	请参阅 MSI 中断接口 。
cfg_interrupt_msi_tph_type	输入	2	请参阅 MSI 中断接口 。

表 35：MSI-X 中断内部接口端口描述（续）

名称	I/O	宽度	描述
cfg_interrupt_msi_tph_st_tag	输入	8	请参阅 MSI 中断接口 。
cfg_interrupt_msi_sent	输出	1	请参阅 MSI 中断接口 中的描述。
cfg_interrupt_msi_fail	输出	1	请参阅 MSI 中断接口 中的描述。
cfg_interrupt_msix_int	输入	1	请参阅 MSI 中断接口 中的描述。
cfg_interrupt_msix_vec_pending	输入	2	请参阅 MSI 中断接口 中的描述。
cfg_interrupt_msix_vec_pending_status	输出	1	请参阅 MSI 中断接口 中的描述。
cfg_interrupt_msix_enable	输出	4	请参阅 MSI 中断接口 中的描述。
cfg_interrupt_msix_mask	输出	4	请参阅 MSI 中断接口 中的描述。
cfg_interrupt_msix_vf_enable	输出	252	请参阅 MSI 中断接口 中的描述。
cfg_interrupt_msix_vf_mask	输出	252	请参阅 MSI 中断接口 中的描述。

配置扩展接口

在实现外部实现的配置寄存器时，“Configuration Extend”（配置扩展）接口允许核随用户应用一起传输配置信息。下表定义了核的配置扩展接口中的端口。

表 36：配置扩展接口端口描述

端口	I/O	宽度	描述
cfg_ext_read_received	输出	1	<p>已接收配置扩展读取。 块接收到来自链路的配置读取请求时，会断言此输出有效。 在 Vivado IDE 的核配置下的“User Defined Configuration Capabilities”（用户定义的配置功能）中选中“PCI Express Extended Configuration Space Enable”（PCI Express 扩展配置空间使能）时，即可设置此端口。</p> <p>所含 cfg_ext_register_number 范围如下的所有接收配置读取都被视为“PCIe Extended Configuration Space”（PCIe 扩展配置空间）。</p> <ul style="list-style-type: none">· UltraScale+™ PCIe4 核: 0x480-0x4FF· UltraScale+ HBM PCIe4C 核: 0xE80-0xFFFF <p>通过断言 cfg_ext_read_received 有效并保持 1 个周期来表示所有接收的配置读取（与其地址无关），并且在 cfg_ext_register_number 和 cfg_ext_function_number 上驱动有效数据。</p> <p>超出 IP 范围的用户应用只需响应接收到的上述范围内的配置读取。</p>

表36：配置扩展接口端口描述(续)

端口	I/O	宽度	描述
cfg_ext_write_received	输出	1	<p>已接收配置扩展写入。块接收到来自链路的配置写入请求时，会断言此输出有效。在 Vivado IDE 的核配置的“User Defined Configuration Capabilities”中选中“PCI Express Extended Configuration Space Enable”时，即可设置此端口。</p> <p>在 cfg_ext_register_number、cfg_ext_function_number、cfg_ext_write_data 和 cfg_ext_write_byte_enable 上会显示对应于所有接收配置写入（所含 cfg_ext_register_number 在 0xb0-0xbf 范围内）的数据。</p> <p>在 cfg_ext_register_number、cfg_ext_function_number、cfg_ext_wrte_data 和 cfg_ext_write_byte_enable 上会显示所含 cfg_ext_register_number 范围如下的所有接收配置写入。</p> <ul style="list-style-type: none"> UltraScale+ PCIe4 核: 0x480-0x4FF UltraScale+ HBM PCIe4C 核: 0xE80-0xFFFF
cfg_ext_register_number	输出	10	<p>配置扩展寄存器编号</p> <p>读取或写入的配置寄存器的 10 位 DWORD 地址。例如，要访问 0x480 地址，应在 cfg_ext_register_number 上布局 $0x480/4 = 0x120$，因为这是 DWORD 地址。当 cfg_ext_read_received 或 cfg_ext_write_received 为高电平时，数据有效。</p>
cfg_ext_function_number	输出	8	<p>配置扩展功能编号</p> <p>对应于配置读取或写入请求的 8 位功能编号。当 cfg_ext_read_received 或 cfg_ext_write_received 为高电平时，数据有效。</p>
cfg_ext_write_data	输出	32	<p>配置扩展写入数据</p> <p>写入配置寄存器的数据。当 cfg_ext_write_received 为高电平时，此输出有效。</p>
cfg_ext_write_byte_enable	输出	4	<p>配置扩展写入字节使能</p> <p>针对配置写入传输事务的字节使能。</p>
cfg_ext_read_data	输入	32	<p>配置扩展读取数据</p> <p>您可通过此总线将数据从外部实现的配置寄存器提供给核。如果您已设置 cfg_ext_read_data_valid，那么核会在将 cfg_ext_read_received 设置为高电平后，在时钟的下一个上升沿对此数据进行采样。</p>
cfg_ext_read_data_valid	输入	1	<p>配置扩展读取数据有效</p> <p>用户应用通过向核断言此输入有效，以提供来自外部实现的配置寄存器的数据。核会在将 cfg_ext_read_received 设置为高电平后，在时钟的下一个上升沿对此输入进行采样。在 cfg_ext_read_received 信号上接收到读取请求后，核期望在用户时钟的 262144 ('h4_0000) 个时钟周期内断言此信号有效。如果在此时间内未接收到任何响应，那么核将发送含'h0 有效载荷的自动响应，用户应用必须丢弃响应，并立即终止该特定请求。</p>

时钟和复位接口

核正常运作的根本前提是时钟和复位接口向核提供系统级别时钟和复位，并提供用户应用时钟和复位信号。下表定义了核的时钟和复位接口中的端口。

user_clk 信号是源自 TXOUTCLK 管脚的衍生时钟，此管脚由 GT Wizard IP 输出。TXOUTCLK 依赖于 pmareset、progdivreset 和 txpisopd 信号以及连接到 GT Wizard IP 的 sys_clk 或 refclk。因此，不要求 user_clk 连续运行。如需了解有关 TXOUTCLK 的详细信息，请参阅对应的 GT Wizard 文档。

表 37：时钟和复位接口端口描述

端口	I/O	宽度	描述
user_clk	输出	1	用户时钟输出（62.5、125 或 250 MHz） 此时钟采用固定频率，在 Vivado® 集成设计环境（IDE）内进行配置。
user_reset	输出	1	此信号会与 user_clk 同步断言无效。它随 sys_reset 断言有效进行异步断言无效和异步断言有效。
sys_clk	输入	1	参考时钟 此时钟采用可选频率 100 MHz。
sys_clk_gt	输入	1	用于 GT 的 PCIe 参考时钟。此时钟必须直接从 IBUFDS_GTE 驱动（与 sys_clk 的定义和频率相同）。此时钟采用可选频率 100 MHz，与 sys_clk 相同。
sys_reset	输入	1	核的基本复位输入（异步） 默认情况下，此输入处于低电平有效状态，以与 PCIe 边缘连接器复位极性相匹配。
phy_rdy_out	输出	1	phy ready 信号表示 GT Wizard 已就绪。此信号由 phy_rst FSM 在接收到来自 GT Wizard 核的 phy 状态时驱动。

UltraScale+ 器件集成块没有专用的复位管脚布线。从 UltraScale 器件升级至 UltraScale+ 器件时，请谨慎更新设计。

PCI Express 接口

PCI Express (PCI_EXP) 接口由分为多条通道的不同发射和接收对组成。每个 PCI Express 通道都包含一对发射差分信号 (pci_exp_txp 和 pci_exp_txn) 和一对接收差分信号 (pci_exp_rxp 和 pci_exp_rxn)。1 通道核仅支持通道 0，2 通道核支持通道 0-1，4 通道核支持通道 0-3，8 通道核支持通道 0-7，16 通道核支持通道 0-15。下表中定义了 PCI_EXP 接口的发射和接收信号。

表 38：对应 1 通道核、2 通道核、4 通道核、8 通道核和 16 通道核的 PCI Express 接口信号

通道编号	名称	I/O	描述
1 通道核			
0	pci_exp_txp0	输出	PCI Express 发射（正）：串行差分输出 0 (+)
	pci_exp_txn0	输出	PCI Express 发射（负）：串行差分输出 0 (-)
	pci_exp_rxp0	输入	PCI Express 接收（正）：串行差分输入 0 (+)
	pci_exp_rxn0	输入	PCI Express 接收（负）：串行差分输入 0 (-)
2 通道核			
0	pci_exp_txp0	输出	PCI Express 发射（正）：串行差分输出 0 (+)
	pci_exp_txn0	输出	PCI Express 发射（负）：串行差分输出 0 (-)
	pci_exp_rxp0	输入	PCI Express 接收（正）：串行差分输入 0 (+)
	pci_exp_rxn0	输入	PCI Express 接收（负）：串行差分输入 0 (-)
1	pci_exp_txp1	输出	PCI Express 发射（正）：串行差分输出 1 (+)
	pci_exp_txn1	输出	PCI Express 发射（负）：串行差分输出 1 (-)
	pci_exp_rxp1	输入	PCI Express 接收（正）：串行差分输入 1 (+)
	pci_exp_rxn1	输入	PCI Express 接收（负）：串行差分输入 1 (-)

表38：对应1通道核、2通道核、4通道核、8通道核和16通道核的PCI Express 接口信号(续)

通道编号	名称	I/O	描述
4通道核			
0	pci_exp_txp0	输出	PCI Express 发射 (正) : 串行差分输出 0 (+)
	pci_exp_tnx0	输出	PCI Express 发射 (负) : 串行差分输出 0 (-)
	pci_exp_rxp0	输入	PCI Express 接收 (正) : 串行差分输入 0 (+)
	pci_exp_rnx0	输入	PCI Express 接收 (负) : 串行差分输入 0 (-)
1	pci_exp_txp1	输出	PCI Express 发射 (正) : 串行差分输出 1 (+)
	pci_exp_tnx1	输出	PCI Express 发射 (负) : 串行差分输出 1 (-)
	pci_exp_rxp1	输入	PCI Express 接收 (正) : 串行差分输入 1 (+)
	pci_exp_rnx1	输入	PCI Express 接收 (负) : 串行差分输入 1 (-)
2	pci_exp_txp2	输出	PCI Express 发射 (正) : 串行差分输出 2 (+)
	pci_exp_tnx2	输出	PCI Express 发射 (负) : 串行差分输出 2 (-)
	pci_exp_rxp2	输入	PCI Express 接收 (正) : 串行差分输入 2 (+)
	pci_exp_rnx2	输入	PCI Express 接收 (负) : 串行差分输入 2 (-)
3	pci_exp_txp3	输出	PCI Express 发射 (正) : 串行差分输出 3 (+)
	pci_exp_tnx3	输出	PCI Express 发射 (负) : 串行差分输出 3 (-)
	pci_exp_rxp3	输入	PCI Express 接收 (正) : 串行差分输入 3 (+)
	pci_exp_rnx3	输入	PCI Express 接收 (负) : 串行差分输入 3 (-)
8通道核			
0	pci_exp_txp0	输出	PCI Express 发射 (正) : 串行差分输出 0 (+)
	pci_exp_tnx0	输出	PCI Express 发射 (负) : 串行差分输出 0 (-)
	pci_exp_rxp0	输入	PCI Express 接收 (正) : 串行差分输入 0 (+)
	pci_exp_rnx0	输入	PCI Express 接收 (负) : 串行差分输入 0 (-)
1	pci_exp_txp1	输出	PCI Express 发射 (正) : 串行差分输出 1 (+)
	pci_exp_tnx1	输出	PCI Express 发射 (负) : 串行差分输出 1 (-)
	pci_exp_rxp1	输入	PCI Express 接收 (正) : 串行差分输入 1 (+)
	pci_exp_rnx1	输入	PCI Express 接收 (负) : 串行差分输入 1 (-)
2	pci_exp_txp2	输出	PCI Express 发射 (正) : 串行差分输出 2 (+)
	pci_exp_tnx2	输出	PCI Express 发射 (负) : 串行差分输出 2 (-)
	pci_exp_rxp2	输入	PCI Express 接收 (正) : 串行差分输入 2 (+)
	pci_exp_rnx2	输入	PCI Express 接收 (负) : 串行差分输入 2 (-)
3	pci_exp_txp3	输出	PCI Express 发射 (正) : 串行差分输出 3 (+)
	pci_exp_tnx3	输出	PCI Express 发射 (负) : 串行差分输出 3 (-)
	pci_exp_rxp3	输入	PCI Express 接收 (正) : 串行差分输入 3 (+)
	pci_exp_rnx3	输入	PCI Express 接收 (负) : 串行差分输入 3 (-)
4	pci_exp_txp4	输出	PCI Express 发射 (正) : 串行差分输出 4 (+)
	pci_exp_tnx4	输出	PCI Express 发射 (负) : 串行差分输出 4 (-)
	pci_exp_rxp4	输入	PCI Express 接收 (正) : 串行差分输入 4 (+)
	pci_exp_rnx4	输入	PCI Express 接收 (负) : 串行差分输入 4 (-)

表38：对应1通道核、2通道核、4通道核、8通道核和16通道核的PCI Express 接口信号(续)

通道编号	名称	I/O	描述
5	pci_exp_txp5	输出	PCI Express 发射 (正) : 串行差分输出 5 (+)
	pci_exp_txn5	输出	PCI Express 发射 (负) : 串行差分输出 5 (-)
	pci_exp_rxp5	输入	PCI Express 接收 (正) : 串行差分输入 5 (+)
	pci_exp_rxn5	输入	PCI Express 接收 (负) : 串行差分输入 5 (-)
6	pci_exp_txp6	输出	PCI Express 发射 (正) : 串行差分输出 6 (+)
	pci_exp_txn6	输出	PCI Express 发射 (负) : 串行差分输出 6 (-)
	pci_exp_rxp6	输入	PCI Express 接收 (正) : 串行差分输入 6 (+)
	pci_exp_rxn6	输入	PCI Express 接收 (负) : 串行差分输入 6 (-)
7	pci_exp_txp7	输出	PCI Express 发射 (正) : 串行差分输出 7 (+)
	pci_exp_txn7	输出	PCI Express 发射 (负) : 串行差分输出 7 (-)
	pci_exp_rxp7	输入	PCI Express 接收 (正) : 串行差分输入 7 (+)
	pci_exp_rxn7	输入	PCI Express 接收 (负) : 串行差分输入 7 (-)
16通道核			
0	pci_exp_txp0	输出	PCI Express 发射 (正) : 串行差分输出 0 (+)
	pci_exp_txn0	输出	PCI Express 发射 (负) : 串行差分输出 0 (-)
	pci_exp_rxp0	输入	PCI Express 接收 (正) : 串行差分输入 0 (+)
	pci_exp_rxn0	输入	PCI Express 接收 (负) : 串行差分输入 0 (-)
1	pci_exp_txp1	输出	PCI Express 发射 (正) : 串行差分输出 1 (+)
	pci_exp_txn1	输出	PCI Express 发射 (负) : 串行差分输出 1 (-)
	pci_exp_rxp1	输入	PCI Express 接收 (正) : 串行差分输入 1 (+)
	pci_exp_rxn1	输入	PCI Express 接收 (负) : 串行差分输入 1 (-)
2	pci_exp_txp2	输出	PCI Express 发射 (正) : 串行差分输出 2 (+)
	pci_exp_txn2	输出	PCI Express 发射 (负) : 串行差分输出 2 (-)
	pci_exp_rxp2	输入	PCI Express 接收 (正) : 串行差分输入 2 (+)
	pci_exp_rxn2	输入	PCI Express 接收 (负) : 串行差分输入 2 (-)
3	pci_exp_txp3	输出	PCI Express 发射 (正) : 串行差分输出 3 (+)
	pci_exp_txn3	输出	PCI Express 发射 (负) : 串行差分输出 3 (-)
	pci_exp_rxp3	输入	PCI Express 接收 (正) : 串行差分输入 3 (+)
	pci_exp_rxn3	输入	PCI Express 接收 (负) : 串行差分输入 3 (-)
4	pci_exp_txp4	输出	PCI Express 发射 (正) : 串行差分输出 4 (+)
	pci_exp_txn4	输出	PCI Express 发射 (负) : 串行差分输出 4 (-)
	pci_exp_rxp4	输入	PCI Express 接收 (正) : 串行差分输入 4 (+)
	pci_exp_rxn4	输入	PCI Express 接收 (负) : 串行差分输入 4 (-)
5	pci_exp_txp5	输出	PCI Express 发射 (正) : 串行差分输出 5 (+)
	pci_exp_txn5	输出	PCI Express 发射 (负) : 串行差分输出 5 (-)
	pci_exp_rxp5	输入	PCI Express 接收 (正) : 串行差分输入 5 (+)
	pci_exp_rxn5	输入	PCI Express 接收 (负) : 串行差分输入 5 (-)

表38：对应1通道核、2通道核、4通道核、8通道核和16通道核的PCI Express 接口信号(续)

通道编号	名称	I/O	描述
6	pci_exp_txp6	输出	PCI Express 发射 (正) : 串行差分输出 6 (+)
	pci_exp_tnx6	输出	PCI Express 发射 (负) : 串行差分输出 6 (-)
	pci_exp_rxp6	输入	PCI Express 接收 (正) : 串行差分输入 6 (+)
	pci_exp_rxn6	输入	PCI Express 接收 (负) : 串行差分输入 6 (-)
7	pci_exp_txp7	输出	PCI Express 发射 (正) : 串行差分输出 7 (+)
	pci_exp_tnx7	输出	PCI Express 发射 (负) : 串行差分输出 7 (-)
	pci_exp_rxp7	输入	PCI Express 接收 (正) : 串行差分输入 7 (+)
	pci_exp_rxn7	输入	PCI Express 接收 (负) : 串行差分输入 7 (-)
8	pci_exp_txp8	输出	PCI Express 发射 (正) : 串行差分输出 8 (+)
	pci_exp_tnx8	输出	PCI Express 发射 (负) : 串行差分输出 8 (-)
	pci_exp_rxp8	输入	PCI Express 接收 (正) : 串行差分输入 8 (+)
	pci_exp_rxn8	输入	PCI Express 接收 (负) : 串行差分输入 8 (-)
9	pci_exp_txp9	输出	PCI Express 发射 (正) : 串行差分输出 9 (+)
	pci_exp_tnx9	输出	PCI Express 发射 (负) : 串行差分输出 9 (-)
	pci_exp_rxp9	输入	PCI Express 接收 (正) : 串行差分输入 9 (+)
	pci_exp_rxn9	输入	PCI Express 接收 (负) : 串行差分输入 9 (-)
10	pci_exp_txp10	输出	PCI Express 发射 (正) : 串行差分输出 10 (+)
	pci_exp_tnx10	输出	PCI Express 发射 (负) : 串行差分输出 10 (-)
	pci_exp_rxp10	输入	PCI Express 接收 (正) : 串行差分输入 10 (+)
	pci_exp_rxn10	输入	PCI Express 接收 (负) : 串行差分输入 10 (-)
11	pci_exp_txp11	输出	PCI Express 发射 (正) : 串行差分输出 11 (+)
	pci_exp_tnx11	输出	PCI Express 发射 (负) : 串行差分输出 11 (-)
	pci_exp_rxp11	输入	PCI Express 接收 (正) : 串行差分输入 11 (+)
	pci_exp_rxn11	输入	PCI Express 接收 (负) : 串行差分输入 11 (-)
12	pci_exp_txp12	输出	PCI Express 发射 (正) : 串行差分输出 12 (+)
	pci_exp_tnx12	输出	PCI Express 发射 (负) : 串行差分输出 12 (-)
	pci_exp_rxp12	输入	PCI Express 接收 (正) : 串行差分输入 12 (+)
	pci_exp_rxn12	输入	PCI Express 接收 (负) : 串行差分输入 12 (-)
13	pci_exp_txp13	输出	PCI Express 发射 (正) : 串行差分输出 13 (+)
	pci_exp_tnx13	输出	PCI Express 发射 (负) : 串行差分输出 13 (-)
	pci_exp_rxp13	输入	PCI Express 接收 (正) : 串行差分输入 13 (+)
	pci_exp_rxn13	输入	PCI Express 接收 (负) : 串行差分输入 13 (-)
14	pci_exp_txp14	输出	PCI Express 发射 (正) : 串行差分输出 14 (+)
	pci_exp_tnx14	输出	PCI Express 发射 (负) : 串行差分输出 14 (-)
	pci_exp_rxp14	输入	PCI Express 接收 (正) : 串行差分输入 14 (+)
	pci_exp_rxn14	输入	PCI Express 接收 (负) : 串行差分输入 14 (-)

表 38：对应 1 通道核、2 通道核、4 通道核、8 通道核和 16 通道核的 PCI Express 接口信号 (续)

通道编号	名称	I/O	描述
15	pci_exp_txp15	输出	PCI Express 发射 (正)：串行差分输出 15 (+)
	pci_exp_tnx15	输出	PCI Express 发射 (负)：串行差分输出 15 (-)
	pci_exp_rxp15	输入	PCI Express 接收 (正)：串行差分输入 15 (+)
	pci_exp_rnx15	输入	PCI Express 接收 (负)：串行差分输入 15 (-)

配置空间

PCI 配置空间包含以下几个主要部分，如下表中所述。其中包括：

- 传统 PCI v3.0 类型 0/1 配置空间报头：
 - 类型 0 配置空间报头，供端点应用使用；请参阅 [表 39：PCI 配置空间报头（类型 0 和类型 1）](#)
 - 类型 1 配置空间报头，供根端口应用使用；请参阅 [表 39：PCI 配置空间报头（类型 0 和类型 1）](#)
- 遗留扩展功能项：
 - PCIe 功能项
 - 功耗管理功能项
 - 报文信号中断 (MSI) 功能项
 - MSI-X 功能项（可选）
- PCIe 功能：
 - 高级错误报告 (AER) 扩展功能结构
 - 备用请求器 ID (ARI)（可选）
 - 器件序列号 (DSN) 扩展功能结构（可选）
 - 单根 I/O 虚拟化 (SR-IOV)（可选）
 - 虚拟通道 (VC) 扩展功能结构（可选）
- PCIe 扩展功能：
 - 器件序列号扩展功能结构（可选）
 - 虚拟通道扩展功能结构（可选）
 - 高级错误报告扩展功能结构（可选）
 - 媒体配置访问端口 (MCAP) 扩展功能结构（可选）

该核最多可实现 4 个遗留扩展功能项。

如需了解启用该功能的更多信息，请参阅 [自定义和生成核](#)。

该核最多可实现 10 项 PCI Express 扩展功能。剩余 PCI Express 扩展功能空间可供用户用于实现。可供用户使用的空间的起始地址始于 3DCh。如果选择在此空间内实现寄存器，则可选择此空间的起始位置，此空间必须在用户应用内实现。

表 39：PCI 配置空间报头（类型 0 和类型 1）

字节偏移	寄存器（类型 0：端点）				寄存器（类型 1：根/DS 端口）								
00h	器件 ID		供应商 ID		与端点相同								
04h	状态		命令										
08h	类代码			版本 ID									
0Ch	BIST	Header	Lat Tim	CacheL									
10h	BAR0												
14h	BAR1												
18h	BAR2				SecLTim	SubBus#	SecBus#	PrimBus#					
1Ch	BAR3				辅助状态		I/O Lim	I/O Base					
20h	BAR4				存储器限制		存储器基址						
24h	BAR5				PrefetchMemLimit	PrefetchMemBase							
28h	Cardbus CIS Pointer				可预取基址的上 32 位								
2Ch	子系统 ID	子系统供应商 ID			可预取限制的上 32 位								
30h	扩展 ROM BAR				I/O 限制的上 16 位	I/O 基址的上 16 位							
34h	保留		CapPtr		保留								
38h	保留				扩展 ROM BAR								
3Ch	Max_Lat	Min_Gnt	IntrPin	IntrLine	桥接控制		IntrPin	IntrLine					

表 40：PCI Express 配置空间

字节偏移 (DW 偏移)	寄存器（端点）				寄存器（根/DS 端口）									
40h (10h)	PM 功能		下一项功能		PM 功能 ID	与端点相同								
44h (11h)	数据	BSE	PMCSR											
48h (12h)	MSI 控制		下一项功能		MSI 功能 ID									
4Ch (13h)	报文地址（下位）													
50h (14h)	报文地址（上位）													
54h (15h)	保留		报文数据											
58h (16h)	掩码位													
5Ch (17h)	暂挂位													
60h (18h)	MSIX 控制		下一项功能		MSIX 功能 ID	保留								
64h (19h)	表偏移				表 BIR	保留								
68h (1Ah)	PBA Offset				PBA BIR	保留								
6Ch (1Bh)	保留				保留									
70h (1Ch)	PCIE 功能		下一项功能		PCIE 功能 ID	与端点相同								
74h (1Dh)	器件功能													
78h (1Eh)	器件状态		器件控制											
7Ch (1Fh)	链路功能													
80h (20h)	链路状态		链路控制											
84h (21h)	保留				插槽功能									
88h (22h)	保留				插槽状态	插槽控制								

表 40：PCI Express 配置空间 (续)

字节偏移 (DW 偏移)	寄存器 (端点)		寄存器 (根/DS 端口)			
8Ch (23h)	保留		根功能 ¹	根控制 ¹		
90h (24h)	保留		根状态 ¹			
94h (25h)	器件功能 2		与端点相同			
98h (26h)	器件状态 2	器件控制 2				
9Ch (27h)	链路功能 2					
A0h (28h)	链路状态 2	链路控制 2				
A4-FCh	未实现的配置空间 (返回 00000000h)					

注释：

- 仅限根端口；在开关 DS 端口中则保留。

表 41：PCIe 功能列表

PF0	PF1-3	VF	起始地址
传统 PCI CSH	传统 PCI CSH	传统 PCI CSH	0x00
PM	PM	-	0x40
MSI	MSI	-	0x48
MSI-X	MSI-X	MSI-X	0x60
PCIE	PCIE	PCIE	0x70
扩展	扩展		0xB0

表 42：PCI Express 扩展配置空间

字节偏移 (DW 偏移)	寄存器 (端点)			寄存器 (根端口)	
100h (40h)	下一项功能	功能版本	AER 扩展功能		
104h (41h)	不可纠正的错误状态寄存器				
108h (42h)	不可纠正的错误掩码寄存器				
10Ch (43h)	不可纠正的错误严重性寄存器				
110h (44h)	可纠正的错误状态寄存器				
114h (45h)	可纠正的错误掩码寄存器			与端点相同	
118h (46h)	高级错误功能和控制寄存器				
11Ch (47h)	报头 Log 日志寄存器 1				
120h (48h)	报头 Log 日志寄存器 2				
124h (49h)	报头 Log 日志寄存器 3				
128h (4Ah)	报头 Log 日志寄存器 4				
12Ch (4Bh)	保留			根错误命令寄存器	
130h (4Ch)	保留			根错误状态寄存器	
134h (4Dh)	保留			错误源 ID 寄存器	

表 42：PCI Express 扩展配置空间（续）

字节偏移 (DW 偏移)	寄存器 (端点)			寄存器 (根端口)
140h (50h)	下一项功能	功能版本	SR-IOV 扩展功能	
144h (51h)	功能寄存器			
148h (52h)	SR-IOV 状态		控制	
14Ch (53h)	VF 总数		VF 初始数量	
150h (54h)	功能：深度链路		VF 数量	
154h (55h)	VF 步幅		首个 VF 偏移	
158h (56h)	VF 器件 ID		保留	
15Ch (57h)	支持的页面大小			
160h (58h)	系统页面大小			
164h (59h)	VF 基址寄存器 0			
168h (5Ah)	VF 基址寄存器 1			
16Ch (5Bh)	VF 基址寄存器 2			保留
170h (5Ch)	VF 基址寄存器 3			
174h (5Dh)	VF 基址寄存器 4			
178h (5Eh)	VF 基址寄存器 5			
180h (60h)	下一项功能	功能版本	ARI 扩展功能	
184h (61h)	控制		下一项功能	功能组
188h - 19Ch	保留			
1A0h (68h)	下一项功能	功能版本	DSN 扩展功能	
1A4h (69h)	器件序列号 (第 1 项)			
1A8h (6Ah)	器件序列号 (第 1 项)			
1ACh - 1BCh	保留			
1C0h (70h)	下一项功能	功能版本	第 2 项 PCIE 扩展功能	
1C4h (71h)	通道控制			
1C8h (72h)	保留		通道错误状态	
1CCh (73h)	通道 1 均衡控制寄存器	通道 0 均衡控制寄存器		
1D0h (74h)	通道 3 均衡控制寄存器	通道 2 均衡控制寄存器		
1D4h (75h)	通道 5 均衡控制寄存器	通道 4 均衡控制寄存器		
1D8h (76h)	通道 7 均衡控制寄存器	通道 6 均衡控制寄存器		
1DCh (77h)	通道 9 均衡控制寄存器	通道 8 均衡控制寄存器		与端点相同
1E0h (78h)	通道 11 均衡控制寄存器	通道 10 均衡控制寄存器		
1E4h (79h)	通道 13 均衡控制寄存器	通道 12 均衡控制寄存器		
1E8h (7Ah)	通道 15 均衡控制寄存器	通道 14 均衡控制寄存器		
1ECh (7Bh)	通道 1 均衡控制寄存器 2	通道 0 均衡控制寄存器 2		
1F0h (7Ch)	通道 3 均衡控制寄存器 2	通道 2 均衡控制寄存器 2		
1F4h (7Dh)	通道 5 均衡控制寄存器 2	通道 4 均衡控制寄存器 2		
1F8h (7Eh)	通道 7 均衡控制寄存器 2	通道 6 均衡控制寄存器 2		

表 42：PCI Express 扩展配置空间（续）

字节偏移 (DW 偏移)	寄存器 (端点)			寄存器 (根端口)
1FCh (7Fh)	保留			
200h(80h)	下一项功能	功能版本	VC 扩展功能	
204h(81h)			端口 VC 功能寄存器 1	
208h(82h)			端口 VC 功能寄存器 2	
20Ch(83h)			端口 VC 状态	
210h (84h)			VC 资源功能寄存器 0	
214h (85h)			VC 资源控制寄存器 0	
218h (86h)			VC 资源状态 0	
21Ch (87h)	保留			
220h (88h)	下一项功能	功能版本	TPH 扩展功能	
224h (89h)			TPH 请求器功能寄存器	
228h (8Ah)			TPH 请求器控制寄存器	
22Ch (8Bh) - 32Ch	TPH ST 表			
330h (CCh)	保留		下一项功能	环回 VSEC
334h (CDh)				环回报头
338h (CEh)				环回控制
33Ch (CFh)				环回状态
340h (D0h)				错误计数 1
344h (D1h)				错误计数 2
348h (D2h)				错误计数 3
34Ch (D3h)				错误计数 4
350h (D4h)	下一项功能	功能版本	MCAP VSEC	
354h (D5h)	MCAP 报头			
358h (D6h)			JTAG ID	
35Ch (D7h)			比特流版本	
360h (D8h)			状态寄存器	
364h (D9h)			控制寄存器	
368h (DAh)			数据寄存器	
36Ch (DBh)			寄存器读取数据 0	
370h (DCh)			寄存器读取数据 1	
374h (DDh)			寄存器读取数据 2	
378h (DEh)			寄存器读取数据 3	
37Ch - FFCh	保留			

表 43：用户设计扩展配置列表

PF0	PF1-3	VF	起始地址	PF0 下一项功能指针
PCI Express 扩展配置空间使能	PCI Express 扩展配置空间使能	PCI Express 扩展配置空间使能	0x0	PCIE4: 0x480
				PCIE4C: 0xE80

用核设计

本节包含有关利用该核来协助简化设计的指导信息和其它信息。

串联配置

PCI Express® 属于即插即用协议，即上电时，PCIe® 主机将枚举系统。在此流程中，主机将从每个器件读取请求的地址大小，然后向器件分配基址。因此，当主机查询 PCIe 接口时，这些接口必须处于就绪状态，否则不会为其分配基址。PCI Express 规范声明，当系统电源正常的 100 ms 后，PERST# 必须断言无效，并且在 PERST# 断言无效后的 20 ms 内 PCIe 端口必须准备好进行链路训练。这通常被称为 100 ms 启动时间要求。

“Tandem Configuration”（串联配置）使用 2 阶方法以支持 IP 满足 PCI Express 规范中所述的配置时间要求。此技术支持多种用例，如下所述：

- “Tandem PROM”（串联 PROM）：从闪存加载单一 2 阶比特流。
- “Tandem PCIe”（串联 PCIe）：从闪存加载第一阶段比特流，并通过 PCIe 链路将第二阶段比特流交付至 MCAP。
- “Tandem PCIe with Field Updates”（含现场更新的串联 PCIe）：完成串联 PCIe 初始配置后，在 PCIe 链路保持有效时更新整个用户设计。其中，更新区域（布局规划）和设计结构均已预定义，并且已提供 Tcl 脚本。

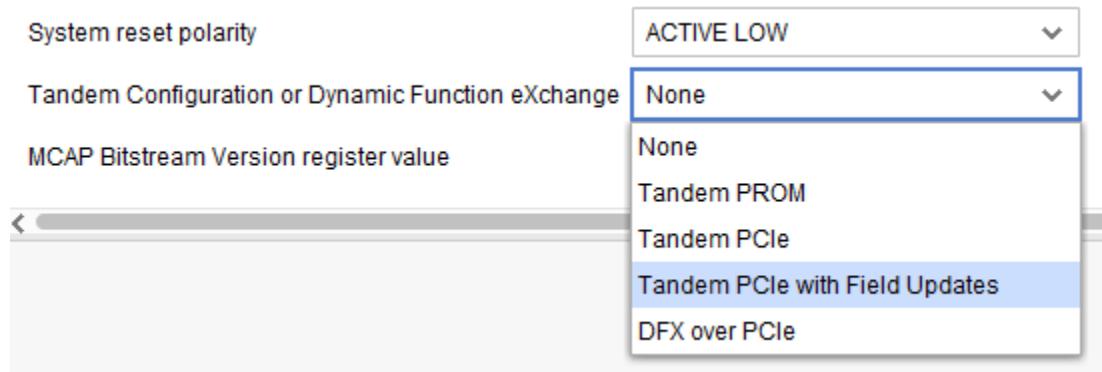
注释：在 UltraScale+™ 器件中，针对“Field Updates”（现场更新），必须使用串联 PCIe。串联 PROM 不支持现场更新。

- 串联 PCIe + Dynamic Function eXchange：这是较常用的用例，针对任意大小或任意数量的动态区域采用串联配置后接 Dynamic Function eXchange (DFX)。
- “DFX over PCIe”（基于 PCIe 的 DFX）：这是标准配置，后接 DFX，使用 PCIe / MCAP 作为部分比特流的交付路径。

为启用上述任意功能，自定义核时请选择相应的选项。在“Basic”（基本）选项卡中：

1. 将“Mode”（模式）切换为“Advanced”（高级）。
2. 根据特定用例，更改“Tandem Configuration or Dynamic Function eXchange”（串联配置或 Dynamic Function eXchange）选项：
 - “Tandem PROM”适用于串联 PROM 用例。
 - “Tandem PCIe”适用于串联 PCIe 或串联 PCIe + Dynamic Function eXchange 用例。
 - “Tandem PCIe with Field Updates”则仅适用于预定义的“Field Updates”用例。
 - “DFX over PCIe”用于为 DFX 启用 MCAP 链路，而无需启用“Tandem Configuration”。

图 2：“Tandem Configuration or Dynamic Function eXchange” 选项



AXI DMA/Bridge Subsystem for PCI Express 针对 UltraScale+ 器件支持“Tandem Configuration 和 Dynamic Function eXchange”功能，包括“Tandem PCIe with Field Updates”。在《DMA/Bridge Subsystem for PCI Express 产品指南》(PG195) 中记录了器件支持详细信息，但仅在本文档中此处提供了完整的串联实现详细信息。

支持的器件

Integrated Block for PCIe 核以及 Vivado® 工具流程支持以赛灵思参考开发板与特定器件/封装组合为目标来完成实现。

“Tandem Configuration”（串联配置）可用作为大部分 UltraScale+™ 器件的量产解决方案。默认情况下，针对所有 ES 硅片禁用比特流生成。串联配置支持下表中所示配置。

表 44：串联 PROM/PCIe 支持的配置

器件	描述			
HDL	仅限 Verilog			
PCIe 配置	所有配置（最大：X16Gen3 或 X8Gen4）			
赛灵思参考开发板支持	适用于 Kintex® UltraScale+™ FPGA 的 KCU116 评估板 适用于 Virtex® UltraScale+™ FPGA 的 VCU118 评估板			
器件支持	器件 ¹	PCIe 块位置	串联配置	含现场更新的串联 PCIe
Artix UltraScale+	AU10P	PCIE40E4_X0Y0	量产	量产
	AU15P	PCIE40E4_X0Y0	量产	量产
	AU20P	PCIE40E4_X0Y0	量产	量产
	AU25P	PCIE40E4_X0Y0	量产	量产
Kintex UltraScale+	KU3P	PCIE40E4_X0Y0	量产	量产
	KU5P	PCIE40E4_X0Y0	量产	量产
	KU11P	PCIE40E4_X1Y0	量产	量产
	KU15P	PCIE40E4_X1Y0	量产	量产
	KU19P ²	PCIE4CE4_X0Y0	仅限串联 PROM	不支持

表 44：串联 PROM/PCIe 支持的配置 (续)

器件	描述		
Virtex UltraScale+	VU3P	PCIE40E4_X1Y0	量产
	VU5P	PCIE40E4_X1Y0	量产
	VU7P	PCIE40E4_X1Y0	量产
	VU9P	PCIE40E4_X1Y2	量产
	VU11P	PCIE40E4_X0Y0	量产
	VU13P	PCIE40E4_X0Y1	量产
	VU19P	PCIE4CE4_X0Y2	量产
	VU23P	PCIE4CE4_X0Y0	量产
	VU27P	PCIE40E4_X0Y0	量产
	VU29P	PCIE40E4_X0Y0	量产
	VU31P	PCIE4CE4_X1Y0	量产
	VU33P	PCIE4CE4_X1Y0	量产
	VU35P	PCIE4CE4_X1Y0	量产
	VU37P	PCIE4CE4_X1Y0	量产
	VU45P	PCIE4CE4_X1Y0	量产
	VU47P	PCIE4CE4_X1Y0	量产
	VU57P	PCIE4CE4_X1Y0	量产
Zynq® UltraScale+™ MPSoC	ZU4CG/EG/EV	PCIE40E4_X0Y1	量产
	ZU5CG/EG/EV	PCIE40E4_X0Y1	量产
	ZU7CG/EG/EV	PCIE40E4_X0Y1	量产
	ZU11EG	PCIE40E4_X1Y0	量产
	ZU17EG	PCIE40E4_X1Y0	量产
	ZU19EG	PCIE40E4_X1Y0	量产
Zynq® UltraScale+™ RFSoC ²	ZU21DR	PCIE40E4_X0Y0	仅限串联 PROM
	ZU25DR	PCIE40E4_X0Y0	仅限串联 PROM
	ZU27DR	PCIE40E4_X0Y0	仅限串联 PROM
	ZU28DR	PCIE40E4_X0Y0	仅限串联 PROM
	ZU29DR	PCIE40E4_X0Y0	仅限串联 PROM
	ZU39DR	PCIE40E4_X0Y1	仅限串联 PROM
	ZU43DR	PCIE4CE4_X0Y0	仅限串联 PROM
	ZU46DR	PCIE4CE4_X0Y0	仅限串联 PROM
	ZU47DR	PCIE4CE4_X0Y0	仅限串联 PROM
	ZU48DR	PCIE4CE4_X0Y0	仅限串联 PROM
	ZU49DR	PCIE4CE4_X0Y0	仅限串联 PROM

注释：

- 仅针对量产硅片提供正式支持。针对所有工程样品芯片（ES1 和 ES2）器件禁用比特流生成。
- 并非所有 Zynq RFSoC 器件和 Kintex UltraScale+ KU19P 都具有启用 MCAP 的 PCIe 块位置。现已添加显式支持，以满足仅适用于这些器件的“Tandem PROM”（串联 PROM）配置需求。

串联工具流程概述

串联 PROM 和串联 PCIe 解决方案仅在 Vivado® Design Suite 中受支持。这 2 种解决方案的工具流程如下所述：

1. 对核进行自定义：从上表中选择受支持的器件，选中“Advanced configuration Mode”（高级配置模式）选项，然后针对“Tandem Configuration or Dynamic Function eXchange”（串联配置或 Dynamic Function eXchange）选择“Tandem PROM”（串联 PROM）或“Tandem PCIe”（串联 PCIe）。
2. 生成核。
3. 打开工程示例，并实现设计示例。
4. 使用来自工程中的工程示例的 IP 和 XDC 并对核进行例化。
5. 对设计进行综合和实现。
6. 生成 bit 文件，然后生成 prom 文件。

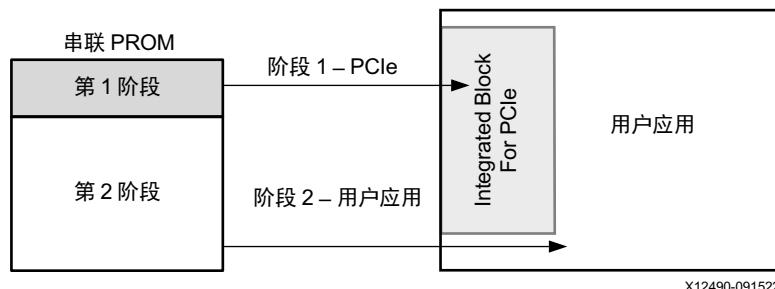
在串联流程中，位于 PCIe 核逻辑外部的某些元素也必须在阶段 1 比特流中启动。Vivado 设计规则检查 (DRC) 会识别此类情况并提供有关如何解决问题的指示信息。此信息通常由修改设计约束或添加额外设计约束组成。

当创建设计示例时，会生成 XDC 文件示例，其中包含的某些约束需复制到对应您的特定工程的 XDC 文件内。这些特定约束记录在设计示例的 XDC 文件中。此外，此设计示例 XDC 文件包含如何为闪存器件（如 BPI 和 SPI）设置选项的示例。

串联 PROM

Tandem PROM 解决方案可将任一比特流拆分为 2 个部分，这 2 个部分都从板载本地配置存储器（通常为任意 PROM 或闪存器件）加载。比特流的第一部分用于配置设计的 PCI Express® 部分，第二部分则用于配置 FPGA 的其余部分。虽然设计显示为含 2 个独立阶段（如下图所示），但生成的 BIT 文件则为包含阶段 1 和阶段 2 的整体式文件。

图 3：串联 PROM 比特流加载步骤

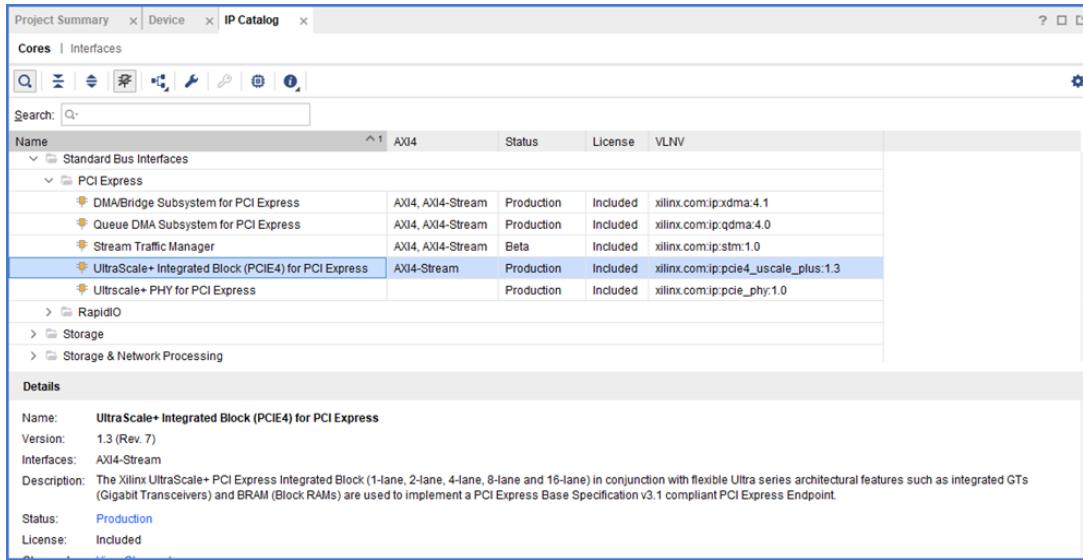


串联 PROM UltraScale+ 工具流程示例

本章节演示了以 UltraScale+ 器件为目标的从始至终的整个 Vivado 工具流程。此流程描述中的路径和指针假定使用默认组件名称 pcie4_ultrascale_plus_0。

1. 创建新的 Vivado 工程，并选择先前表格中所示的受支持的器件/封装。
2. 在 Vivado IP 目录中，展开“Standard Bus Interfaces” → “PCI Express”（标准总线接口 > PCI Express），然后双击“UltraScale+ PCI Express Integrated Block”即可打开“Customize IP”（自定义 IP）对话框。

图 4：Vivado IP 目录



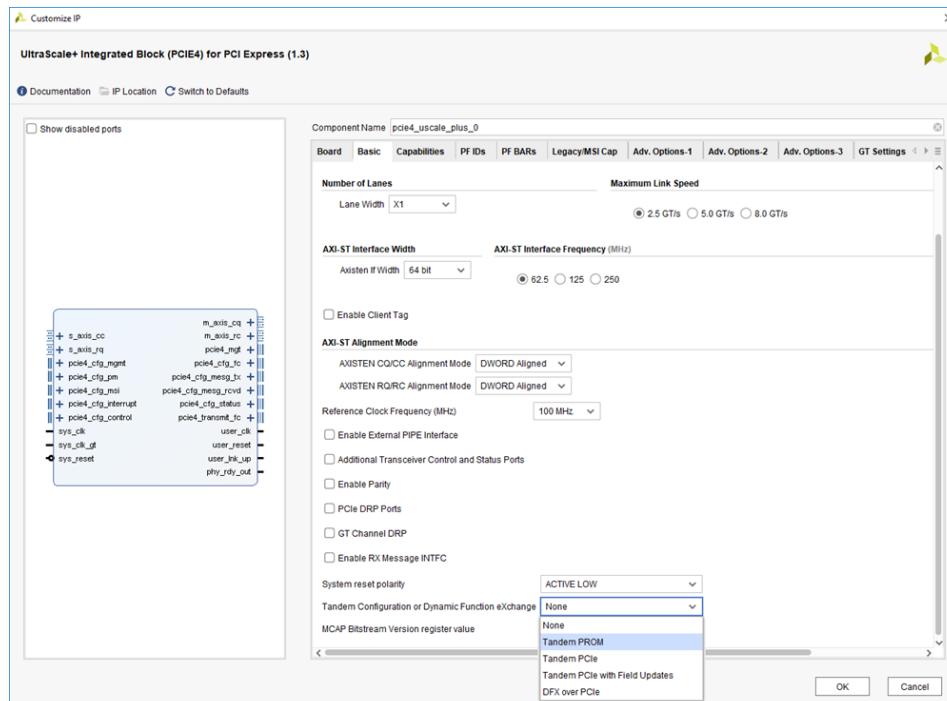
3. 在“Customize IP”对话框的“Basic”（基本）选项卡中，请确保选中下列选项：

- “Mode”（模式）：“Advanced”（高级）
- “PCIe Block Location”（PCIe 块位置）：“X1Y2”

注释：根据先前表格，针对目标器件使用所需的 PCIe 块位置。此设计示例目标为 VU9P。

- “Tandem Configuration or Dynamic Function eXchange”（串联配置或 Dynamic Function eXchange）：“Tandem PROM”（串联 PROM）

图 5：Tandem PROM



4. 执行其它 PCIe 自定义操作，然后单击“OK”以生成核。
5. 当该工具提示您选择要生成的输出文件时，请单击“Generate”。
6. 在“Sources”选项卡中右键单击该核，然后选择“Open IP Example Design”（打开 IP 设计示例）。这样即可创建 Vivado 的新实例，并在 Vivado IDE 中自动加载设计示例。
7. 运行综合与实现。

单击 Flow Navigator 中的“Run Implementation”（运行实现）。单击“OK”首先运行综合直至完成。这样设计会运行完整的工具流程，并生成支持串联 PROM 的已完全布线的设计。

8. 设置 PROM 或闪存设置。

设置相应的设置以便为 PROM 或闪存器件正确生成比特流。在 PCIe 核约束文件（例如，`xilinx_pcnie4_uscale_plus_x1y2.xdc`）中：

- 取消注释并自定义用于定义配置设置的任何约束。
 - 必需的约束为 CONFIG_MODE。例如：`set_property CONFIG_MODE BPI16 [current_design]`
- 如需了解更多信息，请参阅 [器件编程](#)。

9. 生成比特流。

完成综合和实现后，请单击 Flow Navigator 中的“Generate Bitstream”（生成比特流）。这样会在 runs 目录中生成支持串联配置的比特流，例如：`./pcie_ultrascale_plus_0_example.runs/impl/xilinx_pcnie4_uscale_plus_ep.bit`。

注释：您可以选择单独创建阶段 1 和阶段 2 比特流。此流程允许您通过 JTAG 接口控制每个阶段的加载过程，以便于测试。这些比特流与使用 JTAG 加载时用于串联 PCIe 解决方案的比特流相同。由于用于串联 PCIe 设计的 HD.OVERRIDE_PERSIST 设置中存在差异，在硬件中尝试仅从闪存加载阶段 1 比特流是无效的。

```
set_property HD.TANDEM_BITSTREAMS SEPARATE [current_design]
```

由此创建生成的 .bit 文件名为 `xilinx_pcnie4_uscale_plus_ep_tandem1.bit` 和 `xilinx_pcnie4_uscale_plus_ep_tandem2.bit`。

10. 生成 PROM 文件。

在 Vivado Tcl Console 中运行以下命令以创建赛灵思开发板上支持的 PROM 文件。

```
write_cfm -format mcs -interface BPIx16 -size 256 -loadbit "up 0x0  
xilinx_pcnie4_uscale_plus_ep.bit" xilinx_pcnie3_uscale_ep.mcs
```

串联 PROM 总结

通过使用串联 PROM，即可显著缩短配置 UltraScale+ 器件设计的 PCIe 部分所需的时间量。UltraScale+ 器件 Integrated Block for PCIe 核可管理大量设计详细信息，使您能够集中精力处理用户应用。

串联 PCIe

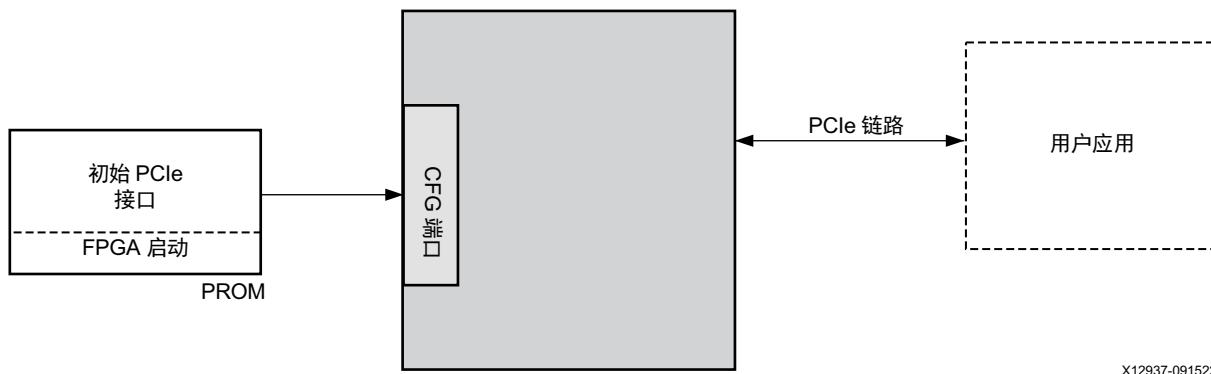
串联 PCIe 与串联 PROM 相似。在第 1 阶段比特流中，仅从 PROM 加载 PCI Express 正常运行所必需的配置存储器单元。阶段 1 比特流加载完成后，PCI Express 端口即可响应枚举流量。因此，将通过 PCI Express 链路来发射阶段 2 比特流。



[视频：创建一个面向 KCU105 的 Tandem PCIe 设计](#) 中解释了如何创建以 KCU105 评估套件为目标的串联设计。

下图显示了比特流加载流程。

图 6：串联 PCIe 比特流加载步骤



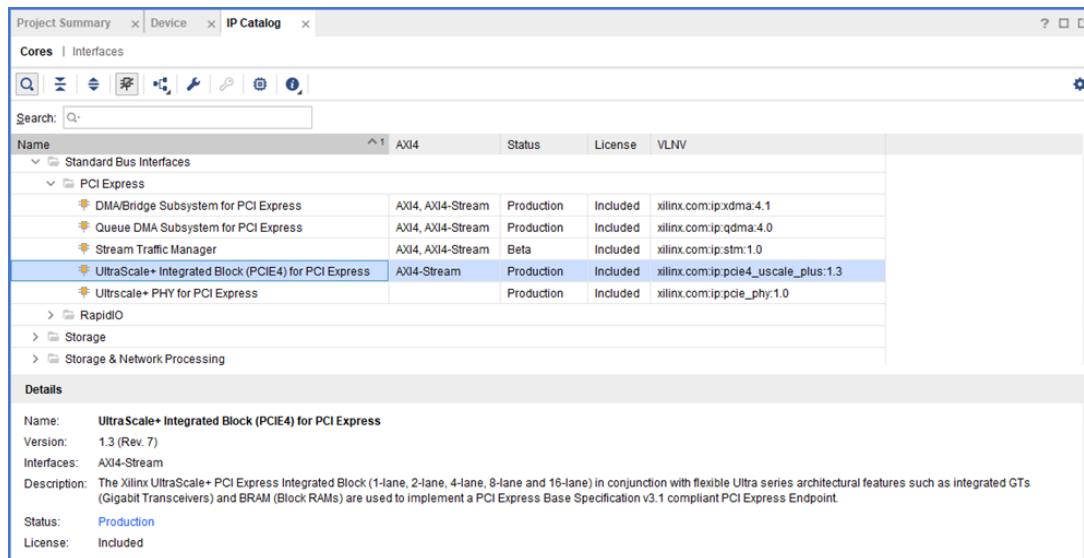
串联 PCIe 在工具流程和比特流生成方面与当前所使用的标准模型相似。运行比特流生成时，会生成 2 个比特流。其中 1 个 BIT 文件表示阶段 1 比特流，并下载到 PROM 中，而另一个 BIT 文件则表示用户应用（阶段 2），并通过使用媒体配置访问端口 (MCAP) 来配置 FPGA 的其余部分。

串联 PCIe UltraScale+ 工具流程示例

本章节演示了以 UltraScale+ 参考板为目标的从始至终的整个 Vivado® 工具流程。此流程描述中的路径和指针假定使用默认组件名称 pcie4_ultrascale_plus_0。

1. 创建新的 Vivado 工程时，请选择先前表格中所示的受支持的器件/封装。
2. 在 Vivado IP 目录中，展开“Standard Bus Interfaces” → “PCI Express”（标准总线接口 > PCI Express），然后双击“UltraScale+ PCI Express Integrated Block”即可打开“Customize IP”（自定义 IP）对话框。

图 7：Vivado IP 目录

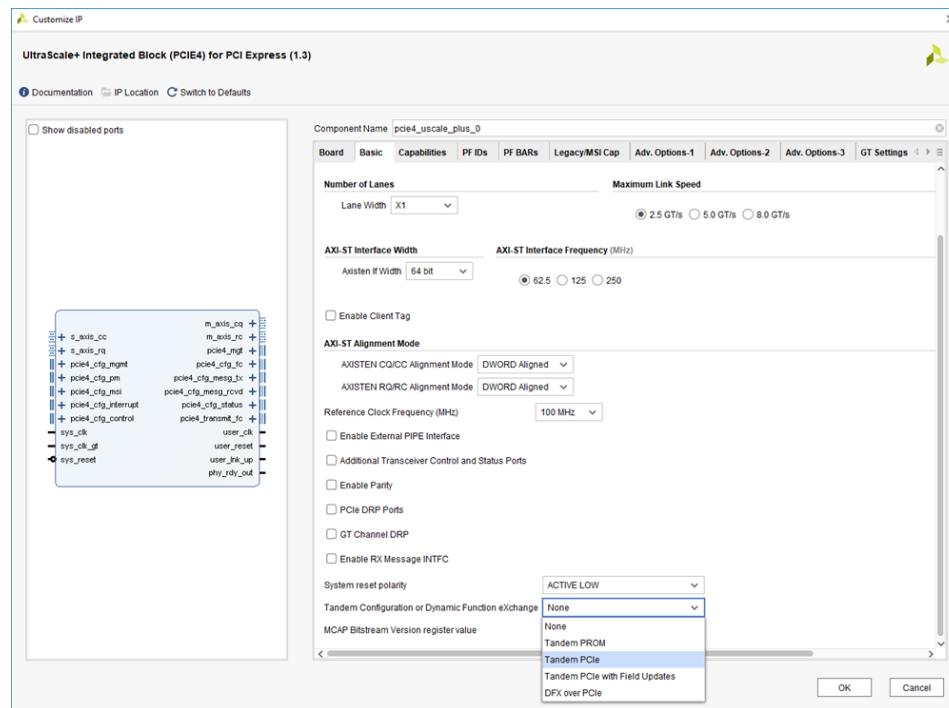


3. 在“Customize IP”对话框的“Basic”（基本）选项卡中，请确保选中下列选项：
 - “Mode”（模式）： “Advanced”（高级）
 - “PCIe Block Location”（PCIe 块位置）： “X1Y2”

注释：根据先前表格，针对目标器件使用所需的 PCIe 块位置。此设计示例目标为 VU9P。

- “Tandem Configuration or Dynamic Function eXchange”（串联配置或 Dynamic Function eXchange）：
“Tandem PCIe”（串联 PCIe）。

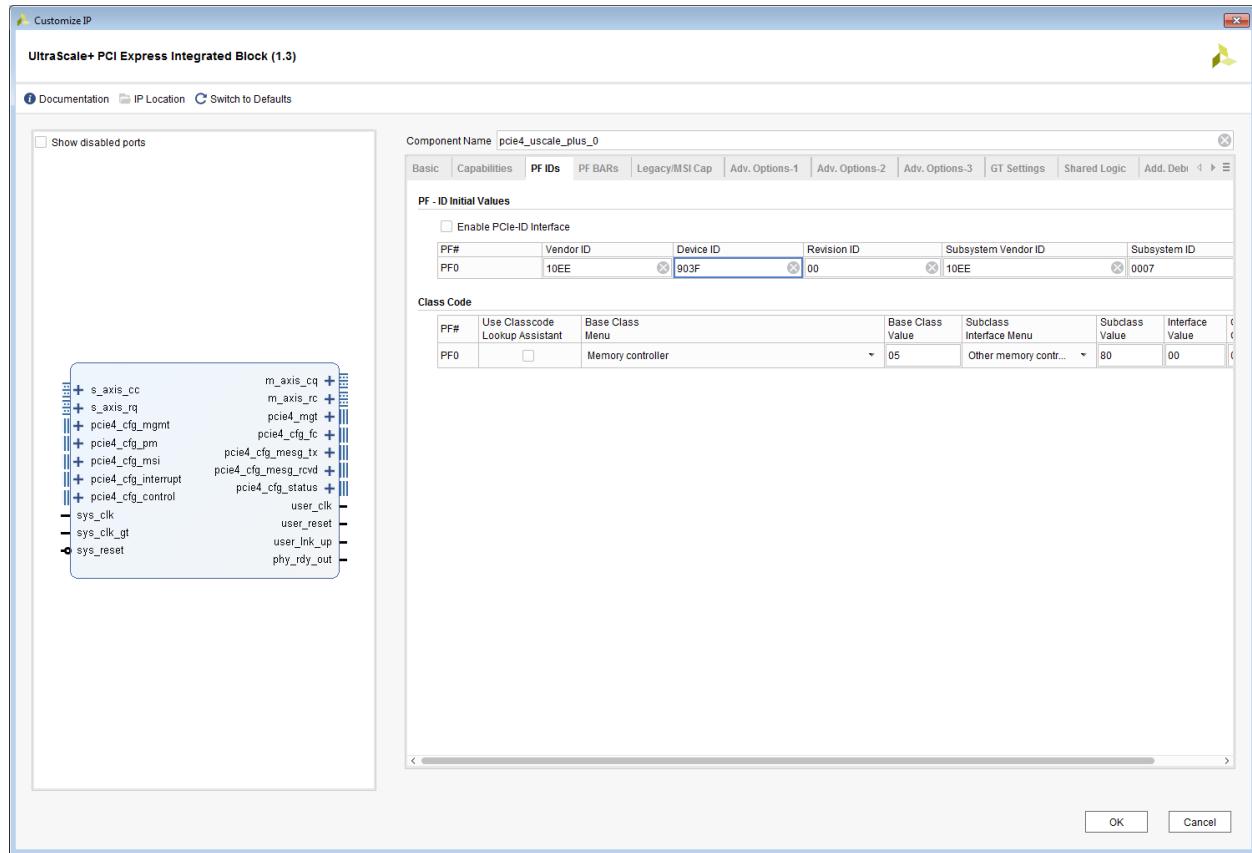
图 8：Tandem PCIe



- 设计示例软件通过“Vendor ID”（供应商 ID）和“Device ID”（器件 ID）连接至器件。供应商 ID 必须为 16'h10EE，器件 ID 必须为 16'h903F。在“PF0 IDs”选项卡中，请设置：
 - “Vendor ID”（供应商 ID）：“10EE”
 - “Device ID”（器件 ID）：“903F”

注释：另一种解决方案是 Vendor ID 和 Device ID 均可更改，并通过更新驱动和主机 PC 软件来匹配新的值。

图 9：ID



5. 执行其它 PCIe 自定义操作，然后选择“OK”（确定）以生成该核。

生成核后，在 Vivado IDE 的“Sources”（源）选项卡中会显示核的层级。

6. 在“Sources”选项卡中右键单击该核，然后选择“Open IP Example Design”（打开 IP 设计示例）。这样即可创建 Vivado 的新实例，并在 Vivado IDE 中自动加载设计示例工程。
7. 运行综合与实现。

单击 Flow Navigator 中的“Run Implementation”（运行实现）。单击“OK”首先运行综合直至完成。这样设计会运行完整的工具流程，最终结果是生成支持串联 PCIe 的已完全布线的设计。

8. 设置 PROM 或闪存设置，并请求 2 个显式 bit 文件。

通过以下步骤设置相应的设置以便为 PROM 或闪存器件正确生成比特流：

- 修改 PCIe IP 约束文件中的约束（例如，pcie4_ultrascale_plus_0_tandem）。
- 通过设置下列属性（如设计示例约束文件中所示）请求 2 个显式比特流：

```
set_property HD.OVERRIDE_PERSIST FALSE [current_design]
set_property HD.TANDEM_BITSTREAMS Separate [current_design]
```

HD.TANDEM_BITSTREAMS 的其它值包括用于 Tandem PROM 解决方案的 Combined（默认值）和为整个器件生成标准单阶比特流的 None。如需了解更多信息，请参阅 [器件编程](#)。

9. 生成比特流。

完成综合和实现后，请单击 Flow Navigator 中的“Generate Bitstream”（生成比特流）。这样会在 runs 目录中创建并放置以下 2 个文件：

```
xilinx_pcnie4_uscale_plus_ep_tandem1.bit  
xilinx_pcnie4_uscale_plus_ep_tandem2.bit
```

10. 为阶段 1 生成 PROM 文件。

在 Vivado Tcl Console (Tcl 控制台) 中运行以下命令以创建 UltraScale+ 开发板上支持的 PROM 文件。

```
write_cfgmem -format mcs -interface BPI -size 256 -loadbit "up 0x0  
xilinx_pcnie4_uscale_plus_ep_tandem1.bit"  
xilinx_pcnie4_uscale_plus_ep_tandem1.mcs
```

通过 PCI Express 加载阶段 2

随 IP 提供了内核模式驱动示例和用户空间应用。如需获取有关检索软件和文档的信息，请参阅 [AR 64761](#)。

请谨记，串联 PCIe 阶段必须保持处于已链接状态。阶段 1 和阶段 2 比特流是从单个已布线的设计镜像创建的，对此设计进行任何更改都意味着必须更新这两个阶段。串联 PCIe 的阶段 2 比特流只能交付一次，以完成整个器件配置。如果需要另一个或者新的阶段 2 镜像或者需要重新加载阶段 2 的能力，则必须使用现场更新方法论。此解决方案会添加 DFX，这意味着对于任一给定的固定阶段 1 比特流，通过锁定的接口端口和多轮设计运行来确保其多个阶段 2 比特流之间的兼容性。

在 Zynq MPSoC 器件上使用串联 PCIe

虽然处理器系统 (PS) 和处理器配置访问端口 (PCAP) 可控制配置逻辑，但可编程逻辑 (PL) 和媒体配置访问端口 (MCAP) 以及内部配置访问端口 (ICAP) 则均被锁定，无法控制配置逻辑。默认情况下，PCAP 处于启用状态，MCAP 无权加载阶段 2，直至切换 PCAP_PR 位为止。

PCAP 与 MCAP/ICAP 接口为互斥关系，无法同时使用。虽然可在 PCAP 与 MCAP/ICAP 之间进行切换，但您必须确保更改接口前没有任何命令或数据处于正在发射或正在接收状态。否则可能导致出现意外行为。控制寄存器 (PCAP_CTRL) 的位 0 (PCAP_PR) 可用于选择 MCAP/ICAP 或 PCAP 来执行 PL 配置。默认选项为 PCAP (1)，但可更改位 MCAP/ICAP (0) 以启用此配置端口。要为 Zynq® UltraScale+™ MPSoC 器件启用 MCAP/ICAP，请设置 `pcap_ctrl` (CSU) 寄存器的 `PCAP_PR` 字段。该位可选择使用 MCAP、ICAP 或 PCAP 来执行 PL 配置。有 2 种方法可用于清除 PCAP_PR 位并授予 ICAP/MCAP 加载阶段 2 镜像的权限。

1. 通过 JTAG 写入 PCAP_CTRL 寄存器以将 MCAP/ICAP 设置为配置端口：
 - 使用 Vitis™ 赛灵思软件命令行工具 (XSCT) 连接到 JTAG 链。
 - 选择 PSU 作为目标。
 - 使用存储器读取 (MRD) 和存储器写入 (MWR) 来更改默认行为。
2. 通过 FSBL 写入 PCAP_CTRL 寄存器以将 MCAP/ICAP 设置为启用的配置端口：
 - 将逻辑置于 `XFsbl_HookBeforeHandoff()` 下。

如需获取 FSBL 挂钩列表，请访问 <https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842019/FSBL>。

```
//Command to SET ICAP as Configuration port//  
Xil_Out32(0xFFCA3008, 0x0); // write PCAP_CTRL
```

串联 PCIe 总结

通过使用串联 PCIe，即可显著缩短配置 UltraScale 器件的 PCIe 部分所需的时间量，并且可降低比特流闪存存储要求。UltraScale+™ 器件 Integrated Block for PCIe 核可管理大量设计详细信息，使您能够集中精力处理用户应用。

含现场更新的串联 PCIe

含现场更新的串联 PCIe 是面向 UltraScale+ 器件的解决方案，它能支持设计师应对快速变化的配置需求，并可动态更改用户应用，因为它可通过 PCIe 链路加载新比特流，而无需 PCIe 链路中断。上电时，该解决方案使用串联 PCIe 配置对器件进行初始配置，随后对 FPGA 中的预定义区域执行 Dynamic Function eXchange (DFX)。这样即可支持将阶段 2 比特流锁定在闪存内，仅当 UltraScale+ 器件 Integrated Block for PCIe 核的特性或者配置 bank (bank 65) 中的 I/O 或时钟管理块必须进行更改时，才需要对镜像进行更新。当需要新增特性或功能时，现场更新允许用户应用（基本上包括设计中的其它所有一切）进行动态重新加载。

含现场更新的串联 PCIe 方法使用 DFX，无需特殊许可。含现场更新的串联 PCIe 是通用串联 PCIe + DFX 解决方案的特殊预定义用例。这 2 个解决方案（串联 PCIe 和 DFX）在相同设计中所受的支持大体相同，使您能够对用户应用中多个更小的区域进行部分重配置。

生成 PCIe IP 后，可创建设计样本以便为现场更新结构提供模板。此设计可显示所需的结构、布局规划、属性和脚本，并可随您的设计需求而变。请遵循此示例将您的设计排序为 2 个部分，并将其映射到 2 个比特流阶段。设计样本在脚本化非工程模式下交付，但也可使用工程模式交付。

可重配置阶段 2

UltraScale+ 器件支持一种全新的现场更新方法，称为“Reconfigurable Stage Twos”（可重配置阶段 2）。该解决方案能够更加灵活地调整比特流交付方式，以便对用户应用进行配置和重新配置，并可减少必须受管理的比特流的数量。

注释：该功能对于 UltraScale 不可用，因为它所依靠的硅片功能在旧架构中不可用。

本质上对于可重配置阶段 2，阶段 2 比特流可充当部分比特流，因此与固定阶段 1 比特流配对时可互换。您可选择任一兼容的阶段 2 比特流以完成器件的初始配置，然后返回至同样的阶段 2 比特流，以便对器件进行部分重配置，并即时加载新用户应用。PCIe 端点保持启动状态，通过这些转换来链接，并用作为比特流交付的途径。

在此情况下，所谓兼容即表示这些阶段 2 已使用 Dynamic Function eXchange 设计流程完成创建，其中 PCIe IP 为静态设计，其它所有一切均布局在可重配置分区。只要 PCIe IP 设计的实现处于锁定状态，PR Verify 就会确认所有阶段 2 设计的兼容性，从而确保这些比特流在安全的环境内交付至目标器件。



注意！ 虽然仍可在 UltraScale+ 器件中为“Tandem PCIe with Field Updates”（含现场更新的串联 PCIe）创建传统部分比特流，但这些比特流不得用作为阶段 2 比特流。此类比特流只能用于对用户应用进行动态重配置，不得用于完成器件的初始启动。



注意！ 在 Vivado 2019.2 中，阶段 2 比特流缺少部分配置帧，这可能导致器件在动态重配置后出现行为错误。此问题在 Vivado 2020.1 中已得到解决，但如果必须使用旧版本，则请继续使用针对阶段 2 和部分重配置采用不同比特流的解决方案。如需获取有关变通方法的更多信息和详情，请参阅 AR 71877。

与串联配置的差异

在标准串联配置与含现场更新的串联 PCIe 之间存在 2 个主要差异，分别与设计布局和结构有关。

设计布局

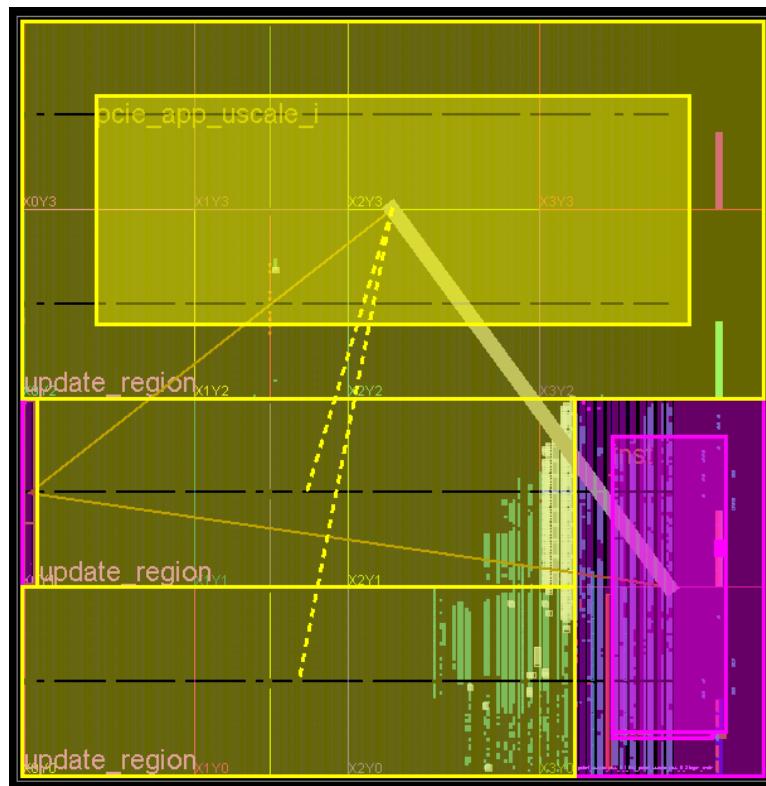
首先，对应这两种变化，在 IP 创建过程中会建立布局规划，此布局规划不应修改，但含现场更新的串联 PCIe 会创建 2 组（而不是 1 组）Pblock。除了带有 HD.TANDEM 标记的 Pblock（对应阶段 1 逻辑）外，还会为用户应用推断第 2 组更大的 Pblock（带有 HD.RECONFIGURABLE 标签）。前者所应用的阶段 1 创建规则与标准串联配置解决方案相同。后者则会强制执行 Dynamic Function eXchange 的所有规则，最值得注意的是布线阻隔，用于确保部分比特流包含用户应用的完整实现。

下图显示了对应含现场更新的串联 PCIe，专为 KU5P 设计样本生成的布局规划。粉色区域为供 PCIe IP 使用的保留区域。此区域包含 PCIe 硬核块、CLB、块 RAM 和收发器 site（用于实现 IP）和 1 个 I/O bank（用于启用物理复位管脚）。黄色区域与粉色区域相反，表示用户应用的可重配置分区 (RP)。它涵盖了 PCIe IP 未涵盖的所有其它资源，包括所有时钟、收发器、I/O 和逻辑。

注释：在 RP 右下角（名为 update_region），您可以找到 PIO 设计示例逻辑以及用于连接设计 2 个部分的分区管脚集合。

如需获取有关分区管脚或 DFX 解决方案的其它方面的更多信息，请参阅《Vivado Design Suite 用户指南：Dynamic Function eXchange》([UG909](#))。

图 10：含现场更新的串联 PCIe 的 KU5P 布局规划



设计结构

第 2 项差异在于设计结构。要将任一用户应用从某一版本切换为下一个版本，必须完全将其封入其自己的层级内。此例化的接口无法更改；否则，顶层静态设计需进行重新编译。除 UltraScale+ 器件 Integrated Block for PCIe 核（位于顶层以下其自身层级内）以及置于 bank 65 内的所有 I/O 逻辑（缓冲器等）之外的所有一切都位于该层级及下层层级内。这意味着，所有其它 bank 的所有 I/O 逻辑都必须置于此处，且不得在顶层进行推断，因此需对 I/O 缓冲器进行例化。

注释：要查看此设计结构的简单示例，请以 KCU116 演示板为目标生成设计示例。

另一个设计结构要求是要置于配置框架内的所有元素都必须包含在此顶层设计内（或者包含在另一个层级内并与 PCIe IP 和用户应用分离）。这些元素包括 BSCAN、ICAP、STARTUP 和相关组件。请参阅《Vivado Design Suite 用户指南：Dynamic Function eXchange》(UG909)，以获取完整列表。这些元素必须按层级隔离，因为不允许对其进行动态重配置。这暗示对于需要这些元素的 IP 核（例如，Vivado Debug Hub 和 Memory Interface Generator (MIG)，两者都使用 BSCAN），必须谨慎处理并确保以安全方式来实现。欲知详情，请参阅[对含现场更新的串联 PCIe 设计进行调试](#)。

串联配置的所有其它注意事项同样适用于含现场更新的串联 PCIe。

含现场更新的串联配置软件流程

请遵循以下步骤以构建串联 IP 并编译设计样本。Vivado® Design Suite 可以针对 2 种设计配置来处理源于 IP 自定义的设计，直至完成比特流生成。

1. 启动“Customize IP”（自定义 IP）对话框，以便自定义 UltraScale+ 器件 Integrated Block for PCIe 核。
注释：仅限 IP 1.3 版或更高版本才支持此解决方案。
2. 选中“Tandem PCIe with Field Updates”（含现场更新的串联 PCIe），并自定义 PCIe IP 核心。必须选中“Advanced Mode”（高级模式）选项才能看到该选项。
3. 使用默认“Out of context per IP”（非关联按 IP）综合选项生成输出文件。这样即可通过综合 IP 来创建可插入完整设计的检查点。
4. 在“Design Sources”（设计源）选项卡中右键单击 IP 并单击“Open IP Example Design”（打开 IP 设计示例）。



重要提示！这样所交付的设计必须按脚本化非工程模式来处理，因为针对 DFX 尚未启用设计工程样本。

设计示例随附有一套脚本以供非工程 Tcl 流程使用。这些脚本样本位于 `field_update_scripts` 文件夹中，但这些脚本全部供设计示例文件夹中的主脚本引用。您可根据需要将此设计示例转换为工程模式。

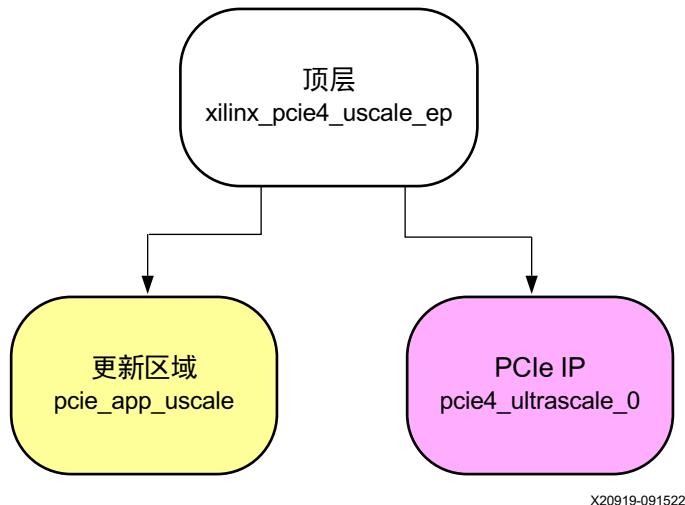
5. 在 Vivado Tcl shell 中，使用 `source` 命令找到 `design_field_updates.tcl`，此文件位于 project 目录下。此文件可将设计示例编译为 2 个版本：
 - 使用默认设置的 Ver1 是初始设计，因此可使用来自该版本的静态设计（本质上只是 PCIe IP）。
 - Ver2 逻辑上与 Ver1 相同，但显示为独立配置，以提供处理第 2 个设计版本时的结果示例。在实际用户设计中，这些逻辑模块将被不同功能所替代。

有关设计示例的详细信息

随 IP 生成的简单 PIO 设计可显示“Tandem PCIe with Field Updates”（含现场更新的串联 PCIe）所需的设计结构。顶层设计文件 `xilinx_pcie_uscale_ep.v` 旨在声明顶层管脚列表，并例化 `pcie4_ultrascale_0` (IP) 和 `pcie_app_uscale` (所有其它对象)。这 2 个子模块分别带有 `HD.TANDEM` 和 `HD.RECONFIGURABLE` 属性标签，用于指令实现工具在布局布线期间遵循上述两组规则。

下图显示了含现场更新的串联 PCIe 设计的基本层级。阶段 1 比特流仅对应粉色模块，阶段 2 由黄色和白色模块组成。部分比特流仅由黄色模块构成。来自样本设计的层级实例的名称如图所示。

图 11：含现场更新的串联 PCIe 的必需设计层级



由于设计的绝大部分都布局在“可重配置分区”中，这绝大部分必须驻留在 `pcie_app_uscale` 层级树中。这包括除 bank 65 中 PCIe IP PERSTN 管脚旁的任意 I/O 和 I/O 逻辑外的所有一切。所有用户 I/O 缓冲器和逻辑、时钟、GT 和其它所有一切都必须位于该层级内，因此当有新版本准备就绪后即可替换。此要求对于设计而言，意味着只要其中的 IP 包含嵌入式 I/O 并且这些 I/O 必须与其它部分布局在同一个 bank 内，那么就不应将此类设计视作为现场更新。由于可能需要手动抽取这些 I/O，导致管理难度增加。

设计中的 3 个分区（顶层、PCIe IP 和用户应用）全都单独进行综合，且其中 2 个子模块标记为非关联。这样可确保每个部分都不会跨界进行最优化，从而避免通过交换这些块来执行实现。任何综合工具均可供使用，前提是禁用自动 I/O 插入。对于 Vivado 综合，可通过选中 `-mode out_of_context` 选项来禁用此功能。每个版本的实现都是基于整个关联设计完成的。从第 2 个版本起，PCIe IP 的布局布线结果和最小顶层逻辑将被锁定，因此无法对其进行更改。

设计脚本相关详情

`design_field_updates.tcl` 脚本为主脚本，用于调用 `field_update_scripts` 文件夹中的其它脚本。设置此主脚本中的变量即可仅对第 1 个版本 (ver1) 或第 2 个版本 (ver2) 执行综合与实现，如需其它版本，则可使用该版本作为模板。基本流程采用的是 Dynamic Function eXchange 流程。如需了解有关 DFX 的更多详情，请参阅《Vivado Design Suite 用户指南：Dynamic Function eXchange》(UG909)。在《Vivado Design Suite 教程：Dynamic Function eXchange》(UG947) 中提供了脚本样本。

以下是有关脚本的几点注意事项。

- 在脚本内定义了特定目录结构，因此遵循示例所创建和使用的结构是最简便的方法。
- 可在行 99 到 109 上设置标志来判定哪些配置必须进行编译。
 - 如果只有初始设计版本可用，那么所有版本 2 标志都应设为 0，对于 PR_verify 同样如此，因为没有可供比较的对象。
 - 同样，如果版本 1 已完成并且您准备编译版本 2，请完全触发这些标志以运行版本 2 配置和 PR_verify 步骤。
- 此脚本是针对初始配置 (ver1) 和一次现场更新 (ver2) 来设置的。要创建更多现场更新，请复制 ver2 的所有参考以创建 ver3 等其它版本。另一种方法是将 ver2 全局替换为 ver3、设置标志并按上述第 2 项用例进行操作。如果 ver1 和 ver2 均已完成，则无需重新访问。
- 用户应用模块的非关联综合在 `update_verX_synth.tcl` 脚本中进行管理（其中 X 表示版本号）。源、约束和选项的声明均在这些文件中完成。

- 每个版本的关联实现则由 `update_verX_impl.tcl` 脚本来管理。通过修改这些脚本即可更改选项、添加约束文件以及生成任一版本的报告。

比特流生成

所提供的脚本可为请求的任何版本创建比特流。`set runUpdateVerXBitstreams 1` 标志（其中 X 表示版本号）可调用 bitstream generation routine later 脚本。如 `design_field_updates.tcl` 脚本中的“Running Bitstream Generation”部分中所示，可通过提供多个值来按需生成不同比特流类型。在大多数情况下，仅限使用 TandemPCIe，当然对于串流核也是如此。以下值可生成下列对应的比特流（显示 ver1 作为示例）：

- TandemPCIe：生成以下比特流。
 - `ver1_tpcie_tandem1` - 对应串联 PCIe 的阶段 1 比特流，将存储在闪存中。
 - `ver1_tpcie_tandem2` - 对应串联 PCIe 的阶段 2 比特流，将通过 PCIe 链路交付；此比特流可实时重新加载。
- PR：仅生成此部分比特流。
 - `ver1_tpcie_update_region_partial` - 将在 ver1 功能中加载部分比特流。

由于阶段 2 比特流可在 UltraScale+ 现场更新解决方案中用作为部分比特流，因此针对 AXI4 串流核无需使用该选项。对于核的 DMA 版本，部分 bit 文件的大小与阶段 2 的 bit 文件大小不同。部分 bit 文件不包含设计的 DMA 部分（因此重新配置期间将不会进行复位），但该文件可使用扩展布线，在某些情况下可覆盖整个器件。

请注意，默认情况下可生成多种格式（.bit、.bin、.mcs、.prm），所用格式在 `generate_bitstreams.tcl` 中进行请求。要调整所创建的文件或更改比特流生成设置，请编辑此 Tcl 文件。

Ver1 并不强制采用从闪存启动的版本。此版本应为可用的最具挑战性的设计版本。此第一版的布局布线结果可用于判定分区管脚位置，以便锁定 PCIe 核与用户应用之间的接口上的布线。

硬件操作详情

器件的初始配置与正常串联配置并无不同。器件的初始配置分 2 个阶段，即，分为使用串联 PCIe 方法的 2 条独立比特流，其中 1 条比特流使用闪存加载，另一条则通过 PCIe 加载。加载阶段 1 后，只有 PCIe IP 正常工作，且功能受限，因为其背后的设计其余部分尚不存在。它可执行链路训练，并且可供系统中的根端口识别。加载阶段 2 比特流后，此器件即进入正常工作模式。

后续动态更新遵循 UltraScale+ 器件 Dynamic Function eXchange 的基本规则，但使用不同的 verX 阶段 2 比特流作为部分比特流。通过链路加载时，逻辑去耦由 PCIe 驱动自动管理。

使用串联 PCIe 设计的比特流名称和提供的脚本时，顺序如下：

1. 给 FPGA 上电。`ver1_tpcie_tandem1.mcs` 从本地闪存发送。
 - 此时，PCIe 端点可正常操作，并且可发生枚举。
 - PCIe 核与未配置的器件其余部分相隔离。
2. 通过 PCIe 链路交付 `ver1_tpcie_tandem2.bin`。
 - 完整器件现已完成编程，设计将通过在 PCIe 与设计其余部分之间建立通信来自动切换至完整使用模式。
 - 或者，还可通过 PCIe 链路发送 `ver2_tpcie_update_region_partial.bin` 来完成器件的初始编程。
3. 请尽情操作 FPGA。
4. 收到更新请求时，请通过 PCIe 链路发送 `ver2_tpcie_update_region_partial.bin` 以交付新用户应用。

- 完整器件再次恢复正常工作，并且现在支持新功能。
 - 自动完成从 PCIe 隔离切换到完整操作模式的过程。
5. 重复步骤 4 以继续更新到新版本（或者返回到 ver1）。

对含现场更新的串联 PCIe 设计进行调试

含现场更新的串联 PCIe 设计使用的是 Dynamic Function eXchange 流程。“Update Region”（更新区域）为“Reconfigurable Partition”（可重配置分区），插入该分区的每个“User Application”（用户应用）均为 1 个“Reconfigurable Module”（可重配置模块）。现已支持在可重配置模块内执行常规调试，对于为任意含现场更新的串联 PCIe IP 实例所生成的设计示例，其中均含有调试电路。可重配置分区顶层(`pci_app_uscale`)包含自动插入 Debug Hub 所需的边界扫描端口，允许在每个可重配置模块内对调试核进行例化。请查找 S_BSCAN 端口，但请勿更改管脚名称。如需获取完整详情，请参阅《Vivado Design Suite 用户指南：Dynamic Function eXchange》(UG909) 的第 8 章。

重要注意事项

下列注意事项对于确保目标器件的操作安全性与可靠性至关重要。

- 请始终确保阶段 2 比特流与 FPGA 中的当前静态设计兼容，然后再加载这些比特流。`PR_Verify` 是 Dynamic Function eXchange 解决方案的基本部分，因此必须配合“Tandem PCIe with Field Updates”（含现场更新的串联 PCIe）使用。`PR_Verify` 可确认多种设计配置（即版本）彼此兼容，因此可在硬件内安全重叠。
- 器件的初始串联配置中必须设置比特流，并将其编译为含现场更新的串联 PCIe 流程中的版本。如果初始比特流加载中已设置标准串联比特流，那么它将与后续现场更新部分比特流不兼容。由此可能发生争用并造成器件损坏。
- 使用可重配置阶段 2 并含现场更新的串联 PCIe 仅适用于 UltraScale+ 器件。现场更新不支持 7 系列器件，可重配置阶段 2 增强功能不支持 UltraScale 器件。对此类方法，应考虑使用通用 Dynamic Function eXchange 解决方案。
- 通用串联 PCIe + DFX 的方法同样受支持。此用例允许您在使用串联初始启动的设计中使用更小的可重配置分区和/或添加更多可重配置分区。此流程与同样在静态设计中包含 PCIe IP 的标准 DFX 流程更类似。要启用此流程，只需在生成 PCIe IP 时使用标准串联 PCIe 选项，然后将其添加到 DFX 设计即可。此操作可在工程模式或非工程模式下进行编译，唯一要求是相应解决方案的 Pblock 不得重叠。

已知问题与限制

- 针对串联配置解决方案默认启用的比特流压缩功能与 Dynamic Function eXchange 逐帧 CRC 功能不兼容。如果任意“更新”部分比特流需执行逐帧 CRC 检查，请启用该功能并禁用比特流压缩，然后重新运行比特流生成。`write_bitstream -celf` 选项仅限用于为每个器件镜像创建部分比特流和清除比特流。

将串联用于用户硬件设计

有 2 种方法可用于将串联流程应用于用户设计。第 1 种方法是使用核随附的设计示例。第 2 种方法是将 PCIe IP 导入现有设计，并根据需要更改设计层级结构。

无论您采用何种方法，都应创建 PCIe 设计示例以获取串联解决方案所需的时钟结构、时序约束和物理块 (Pblock) 约束示例。

方法 1 - 使用现有 PCI Express 设计示例

这是 PCI Express 核必要操作方面最简单的方法，但可能并非对所有用户都可行。如果此方法满足您的设计结构需求，则请遵循下列步骤进行操作。

1. 创建设计示例。

按 [串联 PROM UltraScale+ 工具流程示例](#) 和 [串联 PCIe UltraScale+ 工具流程示例](#) 中所述方式生成设计示例。

2. 插入用户应用。
将 PIO 设计示例替换为用户设计。建议在顶层设计中插入全局元素和顶层元素（例如，I/O 和全局时钟设置）。
3. 根据开发板设计需要，取消注释并修改 SPI 或 BPI 闪存编程设置。
4. 正常实现该设计。

方法 2 - 将 PCIe 设计移植到新 Vivado 工程中

如果无法使用上述方法 1，则应遵循下列步骤，在新工程中使用 PCIe 核以及所期望的串联流程（PROM 或 PCIe）。工程示例具有众多必需的 RTL 和脚本，这些 RTL 和脚本必须移植到用户设计中。

1. 创建设计示例。

按 [串联 PROM UltraScale+ 工具流程示例](#) 和 [串联 PCIe UltraScale+ 工具流程示例](#) 中所述方式生成设计示例。

2. 移植外部 GT Wizard。
如果在核生成期间，在“Shared Logic”（共享逻辑）选项卡中设置了“Include GT Wizard in example design”（在设计示例中包含 GT Wizard）选项，那么将在设计示例顶层例化 GT Wizard IP。此 GT Wizard IP 应移植到用户设计，以提供必要的 GT 连接。

3. 移植到顶层约束。
赛灵思设计约束 (XDC) 文件示例包含适用于 PCIe 核的时序约束、位置约束和 Pblock 约束。所有这些约束 (I/O 位置约束和 I/O 标准约束除外) 都需要移植到用户设计中。其中部分约束包含分层参考，当设计层级与设计示例不同时，需要更新此分层参考。

4. 移植到顶层 Pblock 约束。
以下约束很容易被忽视，因此在此步骤中被挑出来单独讲解。Pblock 约束应指向 PCIe 核的顶层。

```
add_cells_to_pblock [get_pblocks main_pblock_boot] [get_cells -quiet [<path>]]
```



重要提示！ 请勿对 XDC 文件中定义的物理约束进行任何更改，因为约束与器件之间存在相互依赖关系。

5. 将串联 PCIe IP 添加到 Vivado 工程中。
单击 Flow Navigator 中的“Add Sources”（添加源文件）。在“Add Source” Wizard（添加源文件向导）中，选中“Add Existing IP”（添加现有 IP），然后浏览原先用于创建串联 PCIe 设计示例的 XCI 文件。

6. 将相应的 SPI 或 BPI 闪存设置从设计示例 XCD 文件复制粘贴到您的设计 XDC 文件中。
7. 正常实现该设计。

串联配置 RTL 设计

串联配置需对非串联 PCI Express® 产品进行些许修改。本章节解释了核内集成的附加逻辑以及用户应用实现串联 PROM 解决方案的其它职责。

多路复用关键输入

核的某些输入端口采用多路复用，因此在阶段 2 配置过程中禁用这些端口。这些多路复用由 `mcap_design_switch` 信号控制。

加载阶段 2 比特流时，这些输入保持处于断言无效状态。这样可屏蔽由于缺少阶段 2 逻辑而引发的所有意外毛刺，并使 PCIe 核保持处于有效状态。当 `mcap_design_switch` 断言有效时，将切换多路复用，并且所有接口信号的行为都符合本文档中所述方式。

TLP 请求

除了接收配置请求包外，PCI Express 端点还可能接收 PCI Express 硬核块中不处理的 TLP 请求。接收的典型 TLP 请求包括供应商定义的报文和读取请求。在加载阶段 2 之前，TLP 请求会返回请求不受支持 (UR)。阶段 2 加载完成后，`mcap_design_switch` 输出将断言有效，且 TLP 请求按用户设计定义的方式来运行。

串联配置逻辑

核与设计示例包含“Tandem Configuration”（串联配置）专用的端口（信号）。这些信号可在阶段 1（核）与阶段 2（用户逻辑）之间提供握手。握手是核与用户逻辑之间进行交互所必需的。下表定义了核上的握手端口。

表 45：握手端口

名称	方向	极性	描述
<code>mcap_design_switch</code>	输出	高电平有效	用于识别切换至阶段 2 用户逻辑的操作何时完成。 0：尚未加载阶段 2。 1：已加载阶段 2。
<code>cap_req</code>	输出	高电平有效	配置访问端口仲裁请求信号。此信号应用于对用户实现的多个配置接口之间的 FPGA 配置逻辑的使用进行仲裁。如果仅使用媒体配置访问端口 (MCAP) 作为用户实现的配置接口，那么此信号应保持处于未连接状态。
<code>cap_rel</code>	输入	高电平有效	配置访问端口仲裁释放请求信号。此信号应用于对用户实现的多个配置接口之间的 FPGA 配置逻辑的使用进行仲裁。如果仅使用 MCAP 作为用户实现的配置接口，那么此信号应绑定到低电平 (1'b0)。这样即可根据需要允许通过 MCAP 访问 FPGA 配置逻辑。
<code>cap_gnt</code>	输入	高电平有效	配置访问端口仲裁授权信号。此信号应用于对用户实现的多个配置接口之间的 FPGA 配置逻辑的使用进行仲裁。如果仅使用 MCAP 作为用户实现的配置接口，那么此信号绑定到高电平 (1'b1)。这样即可根据请求授权通过 MCAP 访问 FPGA 配置逻辑。
<code>user_reset</code>	输出	高电平有效	当 PCIe 复位时，此信号可用于复位 PCIe 接口逻辑。与 <code>user_clock</code> 同步。
<code>user_clk</code>	输出	不适用	供 PCIe 接口逻辑使用的时钟。
<code>user_link_up</code>	输出	高电平有效	用于识别 PCIe Express 核是否正常链接至主机器件。

这些信号可在用户应用内协调事件，例如，释放输出三态缓冲器（如“串联配置详细信息”中所述）。以下是有关这些信号的部分附加信息：

- 加载阶段 2 之后，即断言 `mcap_design_switch` 有效。加载阶段 2 之后，此输出由根端口系统控制。只要此信号断言无效，PCIe 解决方案 IP 就会与用户设计其余部分隔离，并且 TLP BAR 访问会返回“Unsupported Requests (URs)”（请求不受支持）。

- `cap_req` 信号、`cap_rel` 信号和 `cap_gnt` 信号应用于对多个配置接口（例如，内部配置访问端口 (ICAP)）之间的 FPGA 配置逻辑的使用进行仲裁。ICAP 可在其它 IP 核中使用，或者也可在用户设计中直接例化。要在 MCAP 与 ICAP 之间进行仲裁，必须创建逻辑，并使用 `cap_*` 信号按用户设计期望的方式允许访问每个接口。对于 MCAP，应始终为其授予对配置逻辑的专享访问权限，直至阶段 2 完全完成加载为止。这可通过断言 `mcap_design_switch` 输出有效来识别。初始阶段 2 设计完成加载后，MCAP 接口即可按系统级设计所期望的方式来使用。当根端口连接请求访问配置逻辑时，`cap_req` 断言有效。用户设计可通过在响应中断言 `cap_gnt` 有效来授权访问。随后，用户设计可通过断言 `cap_rel` 有效来请求 MCAP 释放配置逻辑的控制权。根端口连接可通过 `deasserting cap_req` (断言无效) 来释放控制权。如果用户逻辑未断言 `cap_gnt` 有效，则不应访问 MCAP。同样，已向 MCAP 接口授予访问权时，其它配置接口不应尝试访问配置逻辑。
- 核本身复位时，`user_reset` 同样可用于复位与该核通信的任何逻辑。
- `user_clk` 仅作为 PCIe IP 核的内部主时钟。此时钟用于对与该核直接通信的任何用户逻辑进行同步。
- `user_lnk_up` 正如其名，表示 PCIe 核当前正在运行已建立的链路。

用户应用握手

在 FPGA 内必须存在内部完成事件，这样串联解决方案才能在 PCI Express 块的核控制与用户应用之间执行握手。[多路复用关键输入](#) 解释了需要此握手机制的原因。发生此切换时，`mcap_design_switch` 将断言有效。

串联配置详细信息

I/O 行为

对于串联配置设计的阶段 1 所需的每个 I/O，此 I/O 所在的整个 bank 都必须在阶段 1 比特流内进行配置。除此 bank 外，还会启用配置 bank (65)，因此以下详细信息适用于这 2 个 bank (或者如果复位管脚位于配置 bank 内，则适用于其中 1 个 bank)。对于 PCI Express，阶段 1 设计中所需的唯一信号为 `sys_reset` 输入端口。因此，与 `sys_reset` 端口位于相同 I/O bank 内的任何阶段 2 I/O 都随阶段 1 一起进行配置。与 `sys_reset` 位于相同 I/O bank 内的任何管脚在内部都处于未连接状态，因此输出管脚会表现出未知行为，直至阶段 2 配置完成其内部连接为止。并且，针对阶段 2 功能需要初始化的组件除非在完成阶段 2 编程后由设计进行复位，否则这些组件不应布局在这些 I/O bank 中。

如果输出管脚必须与 `sys_reset` 管脚驻留在相同 bank 内，并且其值在阶段 2 完成前不得浮动，那么可采用以下方法。在阶段 1 完成（当输出变为有效时）与阶段 2 完成（当驱动逻辑变为有效时）之间使用保持处于三态的 OBUFT。`mcap_design_switch` 信号可用于控制启用管脚，当握手事件完成时释放此输出。



提示：在顶层设计中，推断或例化 OBUFT。使用 `mcap_design_switch` 控制启用（名为 T 的端口） - 注意极性！

```
OBUFT test_outobuf (.O(test_out), .I(test_internal), .T(!mcap_design_switch));
```

使用以下语法作为示例，创建 Pblock 以包含复位管脚位置。此 Pblock 应包含整个 I/O bank 和关联的 I/O 资源与时钟设置原语。FPGA slice 资源的第一列同样应包含在 Pblock 内，以便与部分配置边界对齐。应置于此区域中的所有逻辑都应添加到 Pblock 内，并使用 HD.TANDEM 属性将这些逻辑标识为阶段 1。请谨记，此逻辑在加载阶段 1 后即变为有效，而驱动逻辑可能要在加载阶段 2 之后才会变为有效。创建系统设计时应牢记此考虑因素。建议将其组合在一起，组成其独立的 Pblock。以下是名为 `test_outobuf` 的输出端口的示例。

```
# Create a new Pblock
create_pblock IO_pblock
set_property HD.TANDEM 1 [get_cells <my_cell>]
# Range the Pblock to include the entire IO Bank and the associate XiPhy
and clocking
primitives.
resize_pblock [get_pblocks IO_pblock] -add { \
```

```
IOB_X1Y52:IOB_X1Y103 \
SLICE_X86Y60:SLICE_X86Y119 \
MMCME3_ADV_X1Y1 \
PLLE3_ADV_X1Y2:PLLE3_ADV_X1Y3 \
PLL_SELECT_SITE_X1Y8:PLL_SELECT_SITE_X1Y15 \
BITSLICE_CONTROL_X1Y8:BITSLICE_CONTROL_X1Y15 \
BITSLICE_TX_X1Y8:BITSLICE_TX_X1Y15 \
BITSLICE_RX_TX_X1Y52:BITSLICE_RX_TX_X1Y103 \
XIPHY_FEEDTHROUGH_X4Y1:XIPHY_FEEDTHROUGH_X7Y1 \
RIU_OR_X1Y4:RIU_OR_X1Y7 \
}
# Add components and routes to stage 1 external Pblock
# This constraint should be repeated for each primitive within this pblock
region
add_cells_to_pblock [get_pblocks IO_pblock] [get_cells test_outobuf]
# Identify the logic within this pblock as stage1 logic by applying the
HD.TANDEM
property.
# This constraint should be repeated for each primitive within this pblock
region
set_property HD.TANDEM 1 [get_cells test_outobuf]
```

默认情况下在阶段 2 配置期间，内部上拉电阻器会将设计中的剩余用户 I/O 强制拉高。当 PUDC_B 管脚保持高电平有效状态时，使用此管脚会强制使超出上述 3 个 bank 范围外的所有 bank 中的 I/O 都变为三态模式。在阶段 1 和阶段 2 之间（对于 PCIe 可能需大量时间），每个 I/O 的内部弱下拉将把这些管脚拉低，因为在此期间这些管脚未配置。仅限在这两个配置阶段内才会考虑使用 PUDC_B，在这两个阶段之间则不予使用，因此此管脚的值可能会根据外部电阻器值进行切换。在阶段 2 完成后，所有 I/O 在其已配置状态下都会被释放。

配置管脚行为

DONE 管脚表示已采用标准方法完成配置。DONE 同样适用于串联配置，但用法略有不同。按启动顺序运行时，DONE 会在阶段 1 脉冲至高电平。当阶段 2 加载开始时，它会返回低电平。对于串联 PROM，此变化立即发生，因为阶段 2 处于相同 bit 文件内。对于串联 PCIe，当第二条比特流交付到 PCIe MCAP 接口时，才会发生此变化。在阶段 2 配置结束时，它将拉高并保持高电平。

配置保留（仅限串联 PROM）

串联 PROM 配置不支持用于阶段 1 或阶段 2 配置的双用途配置 I/O。这表示阻止对串联 PROM 设计的外部配置闪存进行访问。因此，当串联 PROM 配置完成后，用户设计无法访问配置闪存。如需从 FPGA 用户设计执行配置闪存更新，则应使用串联 PCIe。“Configuration Persist”（配置保留）在 write_bitstream 期间自动设置，并且其设置必须使 ICAP（必须保留处于阶段 1 外部）能够在用户设计中正常运行。

如果针对所需配置模式正确设置 CONFIG_MODE 选项，但必要的双模式 I/O 管脚仍被用户 I/O 占用，那么 write_bitstream 期间针对每个实例都会发出以下错误：

```
ERROR: [Designutils 12-1767] Cannot add persist programming for site
IOB_X0Y151.
ERROR: [Designutils 12-1767] Cannot add persist programming for site
IOB_X0Y152.
```

占用这些 site 位置的用户 I/O 位置必须进行调整才能使用串联 PROM。

避免配置 bank

Bank 65 包含专用和双模式配置管脚。串联配置可使用其中大部分管脚，包括 UltraScale+ 器件中的专用 PCIe 复位管脚 (PERSTN) 和许多配置管脚（例如，EMCCLK、CSI_B）以及地址管脚和数据管脚来增大接口宽度。



重要提示！ 赛灵思建议串联配置用户避免将 bank 65 用于设计应用（尤其是在使用串联 PROM 时），以避免复杂操作，因为编程比特流分为 2 个阶段。需明确说明的是，Memory Interface Generator (MIG) 所构建的 IP 核不得使用 bank 65 I/O。这样可以确保 IP 完全保留在阶段 2 中，并避免因其嵌入式 I/O 和严苛的时序约束而导致产生复杂结果。

要查看您要使用的配置模式所需的管脚，请参阅《UltraScale 架构配置用户指南》([UG570](#)) 中的配置图。

PROM 选择

配置 PROM 不含针对串联配置的特殊要求。但为了满足 100 ms 规格，您所选 PROM 必须满足下列 3 项条件：

1. 受赛灵思配置支持。
2. 其大小适合阶段 1 和阶段 2；即，PROM 必须能够包含整个比特流。
 - 对于串联 PROM，阶段 1 和阶段 2 都存储在此处；此比特流比标准比特流稍大 (4-5%)。
 - 对于串联 PCIe，比特流大小通常约为 1 MB，但可因设计实现结果、器件选择和压缩效率而略有差异。
3. 满足 PCI Express 的配置时间要求（基于第一阶段比特流大小和比特流加载时间计算结果）。请参阅 [计算串联的比特流加载时间](#)。

请参阅《UltraScale 架构配置用户指南》([UG570](#))。

器件编程

将串联比特流编程为 PROM 与将标准比特流编程为 PROM 并无差异。您可使用所有标准闪存编程方法（如 JTAG、主从 SelectMAP、SPI 和 BPI）来对串联比特流进行编程。当加载完成第一阶段并开始操作后，会断言 DONE 管脚有效，这与所使用的编程方法无关。

注释：针对“JTAG Only”（仅限 JTAG）模式，请勿将模式管脚设置为 101。这样会限制此 ICAP 和 MCAP 功能（对于 SSI 器件尤为明显，因为禁用对从 SLR 的访问），从而阻碍阶段 2 的正常加载过程。如果使用 JTAG 对阶段 1 进行编程，后续使用 MCAP 时，模式管脚应设置为 000、001 或 111。

为了准备进行 SPI 或 BPI 闪存编程，必须在比特流生成前先启用相应的设置。方法是在设计 XDC 文件中添加特定闪存器件设置，如此处所示。在使用 PCI Express 设计示例生成的约束中提供了相应的示例。复制现有（含注释）选项以满足您的开发板要求和闪存编程要求。

以下是对应串联 PROM 的示例：

```
# BPI Flash Programming
set_property CONFIG_MODE BPI16 [current_design]
set_property BITSTREAM.CONFIG.BPI_SYNC_MODE Type1 [current_design]
set_property BITSTREAM.CONFIG.CONFIGRATE 33 [current_design]
set_property CONFIG_VOLTAGE 1.8 [current_design]
set_property CFGBVS GND [current_design]
```

针对 SPI 和 BPI 编程，支持内部生成的 CCLK 和外部提供的 EMCCLK。EMCCLK 可用于提供更快的配置速率，因为配置时钟上的容限更严格。请参阅《UltraScale 架构配置用户指南》([UG570](#))。

如需获取有关 Vivado® Design Suite 中的配置的更多信息，请参阅《Vivado Design Suite 用户指南：编程和调试》([UG908](#))。

比特流加密

针对串联配置（包括串联 PROM 和串联 PCIe 方法）支持比特流加密。对于串联 PCIe，阶段 2 比特流必须使用与阶段 1 比特流相同的密钥来保持加密状态，因为 MCAP（不同于 ICAP）在接收已加密的初始负载后无法接收未加密的比特流。仅限针对 Zynq® 器件才支持比特流身份验证，针对 FPGA 则不予支持。

多重启动和回退

“Tandem Configuration”（串联配置）仅针对阶段 1 支持多重启动和回退功能。对于“Tandem PROM”（串联 PROM），即使在单个编程镜像内交付这两个阶段，仅在阶段 1 失败就可能触发回退事件，因为当器件变为活动状态时，禁用看门狗定时器。对于“Tandem PCIe”（串联 PCIe），它仅适用于阶段 1，因为阶段 2 是通过 PCIe 链路交付的。鉴于“Tandem Configuration”的最终目标是在电源良好状态下的 120 ms 内进行端点枚举，使用多重启动和回退策略所需的额外事件可能导致配置解决方案无法达成此目标。

串联 PROM/PCIe 资源限制

PCIe 必须与全局芯片复位 (GSR) 隔离，此复位在阶段 2 比特流完成加载到 FPGA 后立即发生。因此，阶段 1 和阶段 2 逻辑不得位于同一配置帧内。PCIe IP 所使用的配置帧由纵向遍布于同一个时钟区域内的串行收发器、I/O、FPGA 逻辑、块 RAM 或时钟设置所组成。资源限制如下所述：

- 每个 GT 四通道均包含 4 个串行收发器。在 X1 或 X2 设计中，将使用整个 GT 四通道，未使用的串行收发器不可供用户应用使用。耗用的 GT 四通道数量取决于在 Vivado® IDE 中对核进行自定义时所做的 GT 四通道选择。
- 在阶段 1 I/O bank 与阶段 2 I/O bank 之间不支持 DCI 级联。
- 将 DCI Match_Cycle 选项设置为 NoWait 即可最大程度缩短阶段 1 配置时间：

```
set_property BITSTREAM.STARTUP.MATCH_CYCLE NoWait [current_design]
```

移动 PCIe 复位管脚

总之，为了达成最佳（最小）第一阶段比特流大小，应将 PCIe 复位包管脚与其它配置管脚一起布局在 bank 65 内。如果需要为复位管脚提供新的位置，应考虑将在阶段 1 中配置的任意 I/O 的位置。如果 I/O 的物理位置距离核较远，则会导致在第一阶段内包含额外的配置帧。原因是需要额外的布线资源用于将这些 I/O 包含在第一阶段内。

无论复位管脚位于何处，在阶段 1 内都应保留 bank 65。即使使用 QSPI 等配置模式，也仍然需要 EMCCLK 用于提供尽可能最快的配置，并且该双模式管脚位于 bank 65 中。

非工程流程

在非工程环境中，使用的基本方法与工程环境相同。首先，使用 IP 目录创建 IP，如 [串联 PCIe UltraScale+ 工具流程示例](#) 中所示。核生成的结果之一是 .xci 文件，其中罗列了所有核的详细信息。此文件用于重新生成所有必需的设计源。

1. 读入设计源（可以使用设计示例或您自己的设计）。

```
read_verilog <verilog_sources>
read_vhdl <vhdl_sources>
read_xdc <xdc_sources>
```

2. 定义目标器件。

```
set_property PART <part> [current_project]
```

注释：虽然这是非工程流程，但幕后仍有 1 个隐藏的工程。必须完成此操作以建立显式器件，然后才能读入 IP。

3. 读入 PCIe IP。

```
read_ip pcie_ip_0.xci
```

4. 对设计执行综合。此步骤可从 .xci 输入生成 IP 源。

注释：使用非关联综合时，您可能需要使用仅在实现期间才能应用的约束文件来应用 Pblock 约束。原因在于，部分约束依赖于组合整个设计才能应用约束。

5. 请确保在设计 XDC 文件中完成对设计进行的任何自定义操作，例如，识别配置模式以设置保留的管脚。
6. 实现设计。

```
opt_design  
place_design  
route_design
```

7. 生成 bit 文件。-bin_file 选项应用于串联 PCIe。BIN 文件对齐到 32 位边界，可加速软件通过 PCIe 加载阶段 2 比特流的过程。

```
write_bitstream -bin_file <file>.bit
```

串联设计仿真

为了在启用“Tandem Configuration”（串联配置）时对 PCIe IP 进行仿真，必须设置具体参数来定义核的行为。用户必须设置属性来禁用 MCAP stage1/stage2 设计开关。这样即使加载阶段 2 也仍可允许运行仿真，但并不会对来自 MCAP 寄存器的设计开关行为进行仿真。如果不想强制使用内部信号，那么作为替代选项，也可以在编译器内设置 +define+SIMULATION 指令。

要禁用的属性如下：

```
defparam board.EP.pcie4_uscale_plus_0_i.inst.MCAP_INPUT_GATE_DESIGN_SWITCH = "FALSE";  
defparam board.EP.pcie4_uscale_plus_0_i.inst.MCAP_GATE_MEM_ENABLE_DESIGN_SWITCH = "FALSE";  
defparam board.EP.pcie4_uscale_plus_0_i.inst.MCAP_GATE_IO_ENABLE_DESIGN_SWITCH = "FALSE";  
defparam board.EP.pcie4_uscale_plus_0_i.inst.MCAP_ENABLEMENT = "NONE";
```

在设计示例的测试激励文件中可以找到这些属性。要访问该测试激励文件，请执行以下操作：

- 从生成的核中打开 IP 设计示例。
- 找到并打开顶层测试激励文件 board.v。

实例名称将与生成的核相匹配。

计算串联的比特流加载时间

配置加载时间是配置时钟频率和精度、配置接口的数据宽度以及比特流大小的函数。计算方式可细分为 3 个步骤：

1. 基于标称时钟频率减去标称值的任意变化量来计算最小时钟频率。

最小时钟频率 = 标称时钟 - 时钟变化量

2. 计算最小 PROM 带宽，它是数据总线宽度、时钟频率和 PROM 类型的函数。PROM 带宽计算方式为最小时钟频率乘以总线宽度。

PROM 带宽 = 最小时钟频率 × 总线宽度

3. 计算第一阶段比特流加载时间，即将 write_bitstream 报告的第一阶段比特流大小除以步骤 2 中的最小 PROM 带宽。

阶段 1 加载时间 = (阶段 1 比特流大小) / (PROM 带宽)

write_bitstream 报告的阶段 1 比特流大小可从终端或者从 log 日志文件直接读取。

以下是来自 write_bitstream log 日志的片段，其中显示了 VU9P 器件（采用默认设置，包括压缩）中阶段 1 的比特流大小：

```
Creating bitstream...
Tandem stage1 bitstream contains 11822112 bits.
Tandem stage2 bitstream contains 110742368 bits.
Writing bitstream ./xilinx_pcip.bit...
```

这些值表示单 bit 文件或双 bit 文件中比特流阶段的显式值。比特流压缩的影响会体现在这些值中。

示例 1

示例 1 的配置为：

- QSPI 闪存 (x4)，工作频率为 66 MHz ± 200 ppm
- 阶段 1 大小 = 11822112 比特 = 11.27 Mb

配置加载时间计算步骤如下：

1. 计算最低时钟频率：

$$66 \text{ MHz} \times (1 - 0.0002) = 65.98 \text{ MHz}$$

2. 计算最低 PROM 带宽：

$$4 \text{ 比特} \times 65.98 \text{ MHz} = 263.92 \text{ Mb/s}$$

3. 计算第一阶段比特流加载时间：

$$11.27 \text{ Mb} / 263.92 \text{ Mb/s} = \sim 0.0427 \text{ 或 } 42.7 \text{ ms}$$

示例 2

示例 2 的配置为：

- BPI (x16) 同步模式，工作频率为 50 MHz ± 100 ppm
- 阶段 1 大小 = 11822112 比特 = 11.27 Mb

配置加载时间计算步骤如下：

1. 计算最低时钟频率：

$$50 \text{ MHz} \times (1 - 0.0001) = 49.995 \text{ MHz}$$

2. 计算最低 PROM 带宽：

$$16 \text{ 比特} \times 49.995 \text{ MHz} = 799.92 \text{ Mb/s}$$

3. 计算第一阶段比特流加载时间：

$$11.27 \text{ Mb} / 799.92 \text{ Mb/s} = \sim 0.0141 \text{ s 或 } 14.1 \text{ ms}$$

使用比特流压缩

最大程度缩小阶段 1 比特流大小是串联配置的最终目标，而使用比特流压缩则有助于达成此目标。该选项使用多帧写入技巧来减小比特流的大小，从而缩短所需的配置时间。压缩量因设计而异。选择串联时，在 IP 级别约束中压缩即处于开启状态。在用户设计约束中可按需覆盖此设置。以下命令可用于启用或禁用比特流压缩。

```
set_property BITSTREAM.GENERAL.COMPRESS <TRUE|FALSE> [current_design]
```

其它比特流加载时间注意事项

比特流配置时间还会受到如下因素的影响：

- 电源缓升时间，包括因调节器引发的任何延迟
- T_{POR} （上电复位）

电源缓升时间与设计息息相关。请注意在设计中避免缓升时间过长或延迟过大。在《UltraScale 架构配置用户指南》(UG570) 中列出了开始 FPGA 配置所必需的 FPGA 电源。

在许多情况下，FPGA 电源可同时缓升，或者甚至可稍早于系统电源开始缓升。在此类情况下，设计能获得时序裕度，因为在系统电源稳定前，100 ms 不会开始计数。重申，这完全取决于设计。系统特性应设计为判定 FPGA 电源与系统电源之间的关系。

T_{POR} 针对标准电源缓升速率为 57 ms，针对 UltraScale+ 器件的快速缓升速率则为 15 ms。请参阅《Kintex UltraScale+ FPGA 数据手册：DC 和 AC 开关特性》(DS922) 和《Virtex UltraScale+ FPGA 数据手册：DC 和 AC 开关特性》(DS923)。

针对示例 1 (QSPI 闪存 [x4] 运行速率为 66 MHz \pm 200 ppm) 计算串联的比特流加载时间，需考量下列 2 种情况：

- 情况 1：无 ATX 电源
- 情况 2：有 ATX 电源

假定在 3.3V 和 12V 系统电源稳定之后，FPGA 电源缓升至稳定电平 (2 ms)。此时间差称为 T_{FPGA_PWR} 。在此情况下，由于 FPGA 电源缓升晚于系统电源，因此电源缓升时间会丧失 100 ms 裕度。

测试公式为：

$T_{POR} + \text{比特流加载时间} + T_{FPGA_PWR} < 100 \text{ ms}$ (针对无 ATX 电源的情况)

$T_{POR} + \text{比特流加载时间} + T_{FPGA_PWR} - 100 \text{ ms} < 100 \text{ ms}$ (针对含 ATX 电源的情况)

- 情况 1：无 ATX 电源：由于没有 ATX 电源，因此当 3.3V 和 12 V 系统电源分别达到其标称电压的 9% 和 8% 范围内之后，100 ms 才会开始计数 (请参阅《PCI-SIG 规范》(<https://www.pcisig.com/specifications>)下的“PCI Express 卡机电规范”)。

$$50 \text{ ms} (T_{POR}) + 42.7 \text{ ms} (\text{比特流时间}) + 2 \text{ ms} (\text{缓升时间}) = 94.7 \text{ ms}$$

$$94.7 \text{ ms} < 100 \text{ ms} \text{ PCIe 标准 (正常)}$$

在此情况下，裕度为 5.3 ms。

- 情况 2：有 ATX 电源：ATX 电源提供 PWR_OK 信号，用于指示系统电源何时达到稳定。至少在实际电源达到稳定的 100 ms 后，才会断言此信号有效。因此，可在时序裕度中添加此额外的 100 ms。

$$50 \text{ ms} (T_{POR}) + 42.7 \text{ ms} (\text{比特流时间}) + 2 \text{ ms} (\text{缓升时间}) - 100 \text{ ms} = -5.3 \text{ ms}$$

-5.3 ms < 100 ms PCIe 标准（正常）

在此情况下，裕度为 105.3 ms。

比特流大小样本

阶段 1 比特流的最终大小因诸多因素而异，包括：

- IP：第一阶段 Pblock 的大小和形状用于确定阶段 1 所需的帧数。x8 和 x16 配置在阶段 1 布局规划中需要更多 GT 四通道，这将导致阶段 1 比特流变得更大。
- 器件：器件宽度越大，将 IP 连接到时钟资源所需的布线帧数越多。
- 设计：复位管脚的位置是因添加用户应用而引入的诸多因素之一。
- GT 位置：所用 GT 四通道的选择会影响阶段 1 比特流的大小。为了最有效利用资源，应使用 PCI Express 硬核块相邻的 GT 四通道。
- 压缩：随着器件使用率增加，压缩的有效性会降低。

作为基线，以下提供了部分比特流大小和配置时间样本，对应于随 PCIe IP 生成的 PIO 设计示例。

表 46：比特流大小和配置时间示例¹

器件	完整比特流	完整：BPI16 频率为 50 MHz	串联阶段 1 ²	串联：BPI16 频率为 50 MHz
AU25P	117.7 Mb	147.2 ms	14.4 Mb	18.1 ms
KU15P	277.3 Mb	346.6 ms	17.6 Mb	22.0 ms
VU9P	611.6 Mb	764.5 ms	17.5 Mb	21.8 ms

注释：

1. 此处显示的配置时间不包含 T_{POR}。
2. 由于 PIO 设计非常小，压缩对于降低比特流大小很有效。这些数值是在无压缩前提下获取的，这样可以更准确估算完整设计可能出现的结果。这些数值是使用 PCIe Gen3x16 配置生成的。

使用串联 PCIe 方法加载阶段 2 比特流所需的时间量取决于另外 3 个因素：

- PCI Express 链路的宽度和速度。
- 用于对 MCAP 进行编程的时钟的频率。
- 根端口主机向端点 FPGA 设计交付比特流的效率。对于大部分设计而言，这属于限制因素。

这 3 个因素的下限带宽决定了加载阶段 2 比特流的速度快慢。

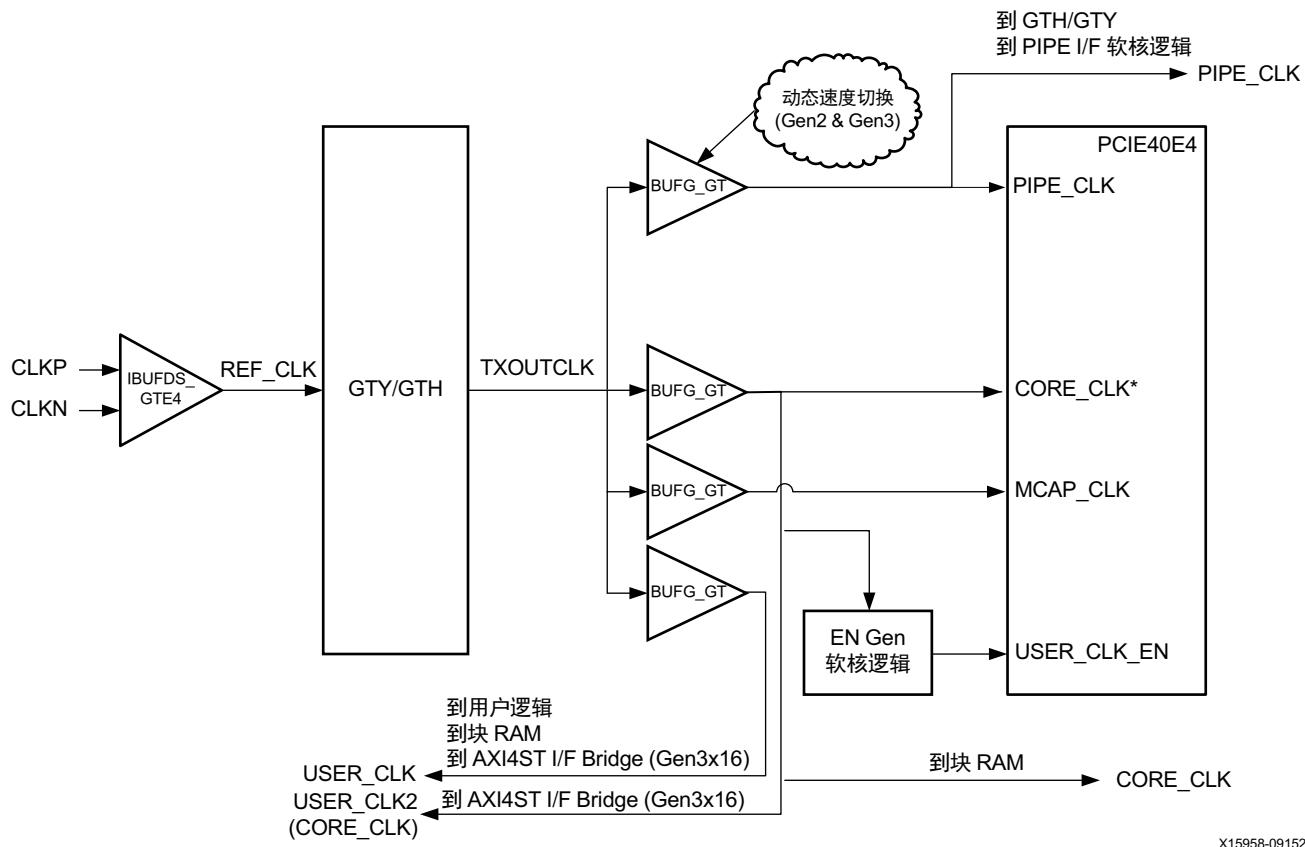
时钟

核需要 100 MHz、125 MHz 或 250 MHz 的参考时钟输入。如需了解更多信息，请参阅[赛灵思 PCIe 解决方案中心](#)内的答复记录。

适用规则如下：

- 参考时钟可采用同步或异步时钟，最高 ± 300 PPM 或 600 PPM（最差情况）。（如果启用扩展频谱时钟 (SSC)，那么必须采用同步链路。）
- PCLK 是 PIPE 接口的基准时钟 (primary clock)。
- 除 PCLK 之外，还需要另 2 个时钟 (CORECLK 和 USERCLK) 以支持该核。
- BUFG_GT 用于生成核时钟。这些时钟全部从 TXOUTCLK 管脚驱动，此管脚为通过 CPLL 衍生的时钟（源自 GTREFCLK0）。在使用 QPLL 的应用中，仅当继续从 CPLL 衍生 TXOUTCLK 时，才会将 QPLL 提供给 GT PCS/PMA 块。
- UltraScale+™ GTH 参考时钟的时钟源必须直接来自 IBUFDS_GTE4。
- 要将参考时钟用于 FPGA 常规互连，必须使用另一个 BUFG_GT。

图 12：时钟架构



所有 PCIe 时钟 (pipe_clk、core_clk、user_clk 和 mcap_clk) 均由 BUFG_GT (源自 TXOUTCLK 管脚) 驱动。这些时钟是通过 CPLL 衍生的时钟 (源自 GTREFCLK0)。在使用 QPLL 的应用中，仅当继续从 CPLL 衍生 TXOUTCLK 时，才会将 QPLL 提供给 GT PCS/PMA 块。核的所有用户接口信号的时序约束都与相同时钟 (user_clk) 有关，根据配置的链路速度和宽度，该时钟频率可以是 62.5 MHz、125 MHz 或 250 MHz（请参阅上图）。

在典型 PCI Express® 解决方案中，PCI Express 参考时钟为扩展频谱时钟 (SSC)，提供的频率为 100 MHz。在大部分商用 PCI Express 系统中，都无法禁用 SSC。如需了解有关 SSC 和 PCI Express 的更多信息，请参阅《PCI Express 基本规范第 3.0 版》中的章节 4.3.7.1.1。



重要提示！ 由于大部分主机系统的参考时钟上都有 SSC，所有插卡设计都必须使用同步时钟设置。对于使用时隙时钟的器件，在 Vivado® IP 目录中必须启用“Link Status”（链路状态）寄存器中的“Slot Clock Configuration”（时隙时钟配置）设置。

每个链路伙伴器件都共享相同时钟源。下图显示了使用 100 MHz 参考时钟的系统。即使器件包含在嵌入式系统中，只要系统使用商用 PCI Express 根联合体或开关并配合采用典型主板时钟设置方案，就仍应使用同步时钟设置。

注释：时钟设置图显示了开发板布局的高层表示法。执行开发板布局时请确保耦合、端接和详细信息都正确无误。请参阅《UltraScale 架构 GTH 收发器用户指南》(UG576)

图 13：使用 100 MHz 参考时钟的嵌入式系统

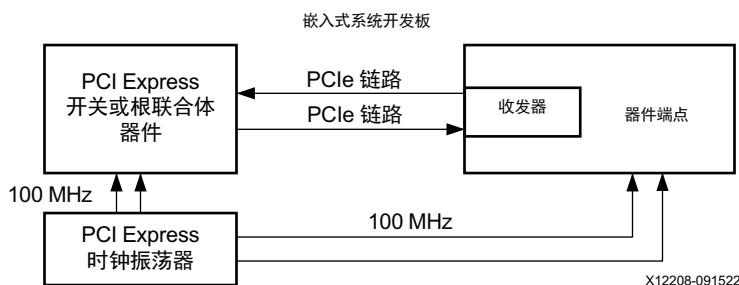
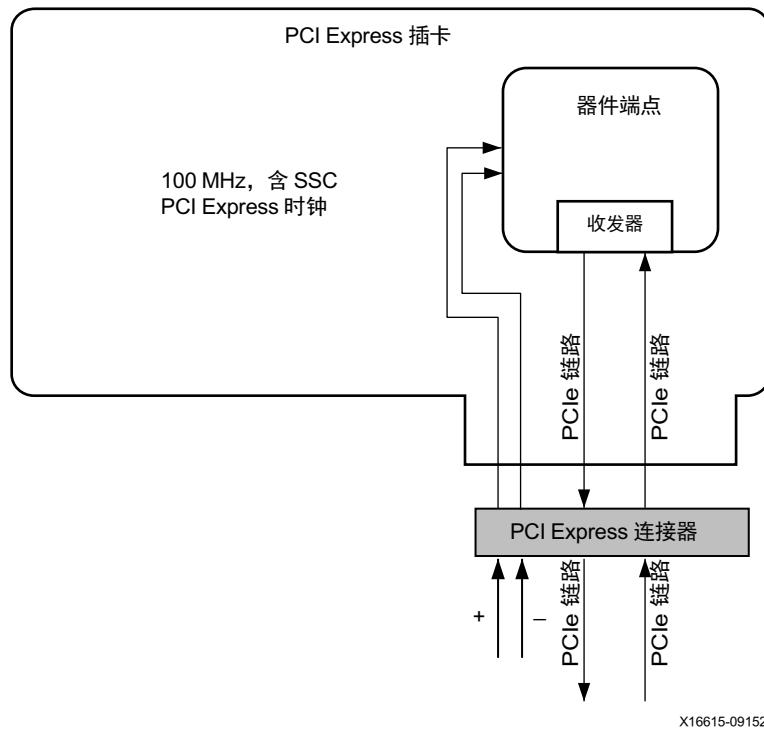


图 14：使用 100 MHz 参考时钟的开放式系统插卡



PCIe 核会检查确认 GT 电源稳定，然后再启用时钟。

- 这导致针对由 I_BUFDS_GTE4 (PCIe 参考时钟) 驱动的 BUFG_GT，将产生逻辑驱动的 CE (而不是 VCC)。
- 在 CE 中执行此更改之前，如有另一个 (并行) BUFG_GT 已连接到 I_BUFDS_GTE4 且 CE 由 VCC 驱动，那么由 opt_design/MLO 插入的 BUFG_GT_SYNC 可驱动这 2 个 BUFG_GT。

- 如有并行 BUFG_GT 与 PCIe BUFG_GT 时钟不共享相同的 CE，那么 2 个 BUFG_GT_SYNC 由 opt_design/MLO 插入。
- 由于针对 IBUFDS_GTE4 驱动的 BUFG_GT 只能有 1 个 BUFG_GT_SYNC，布线器不知晓如何处理第 2 个 BUFG_GT_SYNC 并且不会对 IBUFDS_GTE4/ODIV2 驱动的时钟信号线进行布线。
- 您必须确保由 IBUFDS_GTE4 驱动的 BUFG_GT 具有相同的 CE/CLR 管脚。

复位

核使用 PCI Express® 基础复位期间断言有效的低电平有效复位信号 sys_reset 来对系统进行复位。断言此信号有效会导致包括收发器在内的整个核发生硬复位。复位必须保持断言有效，直至参考时钟按《PCI Express 基本规范》中的定义达成稳定状态为止。复位释放后，核会尝试进行链路训练并恢复正常操作。在典型的端点应用（例如，插卡）中，通常存在边带复位信号并且此信号应连接至 sys_reset。对于无边带复位信号的端点应用，初始硬件复位应在本地生成。

在 PCI Express 中可能发生 4 种复位事件：

- “Cold Reset”（冷复位）：在上电时发生的基础复位。断言 sys_reset 信号有效就会导致核发生冷复位。
- “Warm Reset”（暖复位）：由硬件触发的基础复位，无需断电并重新上电。断言 sys_reset 信号有效就会导致核发生暖复位。
- “Hot Reset”（热复位）：在整个 PCI Express 链路上通过协议来进行复位的频带内传输就会导致整个端点器件发生复位。在此情况下，不使用 sys_reset。对于热复位，断言 cfg_hot_reset_out 信号有效即可标示复位源。
- “Function-Level Reset”（功能级别复位）：在整个 PCI Express 链路上通过协议来进行复位的频带内传输就会导致仅限特定功能发生复位。在此情况下，核会断言 cfg_flr_in_process 和/或 cfg_vf_flr_in_process（对应于要复位的功能）的位有效。与要复位的功能关联的逻辑必须断言 cfg_flr_done 或 cfg_vf_flr_done 的对应位有效才能标示它已完成复位流程。

当向“Initiate Function Level Reset”（启动功能级别复位）位写入 1b 以启动 FLR 之后，该功能必须在 100 ms 内完成 FLR 和特定于该功能的所有初始化操作。

核的“User Application”（用户应用）接口具有输出信号 user_reset。此信号会与 user_clk 同步断言无效。在以下任何情况下，都会断言 user_reset 信号有效：

- “Fundamental Reset”（基础复位）：由于断言 sys_reset 有效而发生冷复位或暖复位。
- “PLL within the Core Wrapper”（核封装文件中的 PLL）：锁定丢失，表示时钟输入稳定性存在问题。
- “Loss of Transceiver PLL Lock”（收发器 PLL 锁定丢失）：只要任意收发器发生锁定丢失，均表示 PCI Express 链路存在问题。

解决列出的所有情况后，user_reset 信号与 user_clk 将同步断言无效，从而允许核尝试进行链路训练并恢复正常操作。

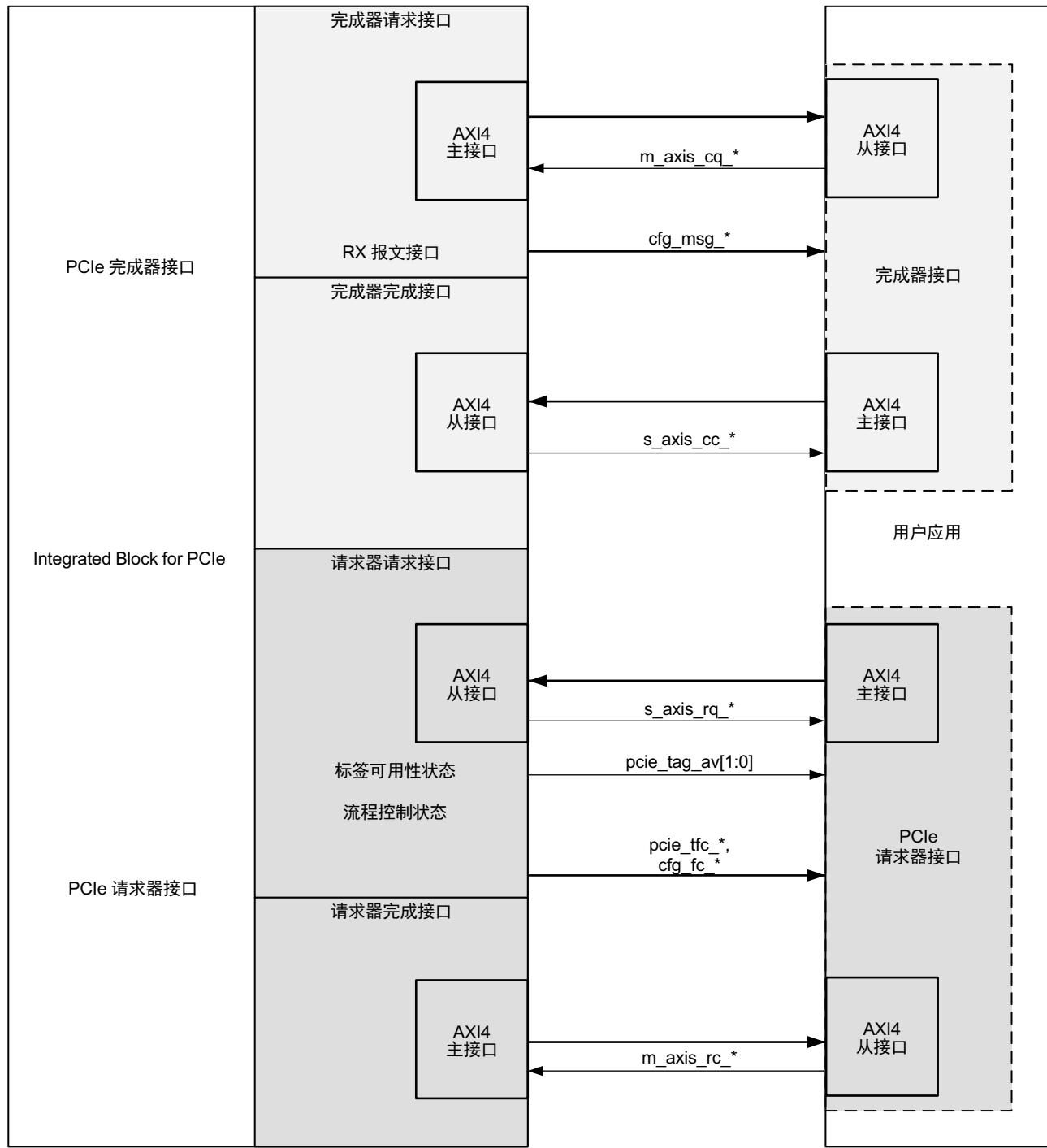
AXI4-Stream 接口描述

本章节提供了与核的用户接口相关联的特性、参数和信号的详细描述。

功能综述

下图演示了该核的用户接口。

图 15：集成块用户接口模块框图



X12207-091522

用户接口分为 4 个独立接口，数据通过这些接口在 PCIe 链路与用户应用之间进行传输：

- PCIe Completer Request (CQ)（完成器请求）接口，源自链路的请求通过该接口交付到用户应用。
- PCIe Completer Completion (CC)（完成器完成）接口，用户应用通过该接口将响应发送回完成器请求接口。用户应用可将所有非转发传输事务作为独立拆分的传输事务来进行处理。即，它可持续接受完成器请求接口上的新请求，同时针对请求发送完成 (completion) 包。
- PCIe Requester Request (RQ)（请求器请求）接口，用户应用可通过该接口生成请求并发送到链路随附的远程 PCIe 器件。
- PCIe Requester Completion (RC)（请求器完成）接口，用户应用可通过该接口接收来自链路的完成包（以作为 PCIe 请求器来响应用户应用请求）。

这 4 个接口全都基于 AMBA® AXI4-Stream 协议规范。根据所选的通道数量和数据速率，这些接口的宽度可配置为 64、128、256 或 512 位，可选用户时钟频率包括 62.5 MHz、125 MHz 或 250 MHz。仅限 Gen3 x16 接口才可包含 512 位 (64 字节) 数据并以 250 MHz 的时钟频率运行，前提是每个方向上的峰值传输速率为 16 GB/s，此速率足以支持 Gen3 x16 PCI Express® 链路。

下表列出了对应集成块所支持的不同链路宽度和链路速度有效的接口宽度与用户时钟频率组合。无论在任何情况下，全部 4 个 AXI4-Stream 接口均配置为相同宽度。

此外，集成块包含以下接口，状态信息可通过这些接口传递到用户应用的 PCIe 主接口侧：

- 连接到请求器请求 (RQ) 接口的流量控制状态接口，用于提供当前可用发射信用值的信息。它支持用户应用基于可用信用值来调度请求，避免由于其链路伙伴缺少信用值而导致控制器内部流水线发生阻塞。
- 连接到请求器请求 (RQ) 接口的标签可用性状态接口，用于提供有关可供分配给非转发请求的标签数量信息。当 PCIe IP 中的标签管理单元已将所有可用标签都用于传出的非转发请求时，它支持客户调度请求并避免发生阻塞。
- 连接到完成器请求 (CQ) 接口的接收报文接口，用于交付从链路接收到的报文 TLP。当从链路接收到报文（而不是将整条报文通过 AXI4 接口传输到用户应用）时，它可选择是否向用户逻辑提供指示信息。

受支持的时钟频率和接口宽度

表 47：针对各种配置支持的时钟频率和接口宽度

PCIe 链路速度	PCIe 链路宽度	PIPE 接口数据宽度 (位)	AXI4 串流接口数据宽度 (位)	pipe_clk 频率 (MHz)	core_clk 频率 (MHz)	user_clk 频率 (MHz) (axi4st)	user_clk 频率 (MHz) (cfg, axi4st)	mcap_clk 频率 (MHz)	GT TxOutClk (MHz)
Gen1	X1	16	64	125	250	62.5	62.5	62.5/125	250
		16	64	125	250	125	125	125	250
		16	64	125	250	250	250	125	250
	X2	16	64	125	250	62.5	62.5	62.5/125	250
		16	64	125	250	125	125	125	250
		16	64	125	250	250	250	125	250
	X4	16	64	125	250	125	125	125	250
		16	64	125	250	250	250	125	250
	X8	16	64	125	250	250	250	125	250
		16	128	125	250	125	125	125	250
	X16	16	128	125	250	250	250	125	250

表 47：针对各种配置支持的时钟频率和接口宽度（续）

PCIe 链路速度	PCIe 链路宽度	PIPE 接口数据宽度（位）	AXI4 串流接口数据宽度（位）	pipe_clk 频率 (MHz)	core_clk 频率 (MHz)	user_clk2 频率 (MHz) (axi4st)	user_clk 频率 (MHz) (cfg, axi4st)	mcap_clk 频率 (MHz)	GT TxOutClk (MHz)
Gen2	X1	16	64	125/250	250	62.5	62.5	62.5/125	250
		16	64	125/250	250	125	125	125	250
		16	64	125/250	250	250	250	125	250
	X2	16	64	125/250	250	125	125	125	250
		16	64	125/250	250	250	250	125	250
	X4	16	64	125/250	250	250	250	125	250
		16	128	125/250	250	125	125	125	250
	X8	16	128	125/250	250	250	250	125	250
		16	256	125/250	250	125	125	125	250
	X16	16	256	125/250	250	250	250	125	250
Gen3	X1	16/32	64	125/250	250	125	125	125	250
		16/32	64	125/250	250	250	250	125	250
	X2	16/32	64	125/250	250	250	250	125	250
		16/32	128	125/250	250	125	125	125	250
	X4	16/32	128	125/250	250	250	250	125	250
		16/32	256	125/250	250	125	125	125	250
	X8	16/32	256	125/250	250/500 ²	250	250	125	500
	X16	16/32	512	125/250	500	500	250	125	500
	X1	16/32/64	64	125/250	250	250	250	125	500
		16/32/64	128	125/250	250	125	125	125	500
Gen4	X2	16/32/64	128	125/250	250	250	250	125	500
		16/32/64	256	125/250	250	125	125	125	500
	X4	16/32/64	256	125/250	250	250	250	125	500
	X8	16/32/64	512	125/250	500	500	250	125	500

注释：

- 对于 Gen1/Gen2 速度，管道数据宽度为 16。对于 Gen3 速度，管道数据宽度为 32，对于 Gen4 速度，则为 64。在 Gen3 配置中处于 Gen1 速度下时，管道数据宽度切换为 16。同样，在 Gen4 中会基于链路训练进程中的速度进行自动切换。用户无法配置管道数据宽度。
- 对于 Gen3 x8 配置，core_clk 频率为 250MHz，适用于 -1L、-1LV 和 -2LV 速度等级的器件。

数据对齐选项

传输事务层包 (TLP) 在每个 AXI4-Stream 接口上都作为描述符后接有效载荷数据（当 TLP 包含有效载荷时）的形式来传输。描述符大小固定，在请求接口上为 16 字节，在完成接口上则为 12 字节。集成块在其发射侧（目标为链路）根据用户应用在描述符中提供的参数来汇编 TLP 报头。集成块在其接收侧（目标为用户接口）从接收到的 TLP 的报头中提取参数，并构成描述符以交付到用户应用。根据 AXI4-Stream 接口协议中的定义，每个 TLP 均作为数据包来传输。

64/128/256 位接口：

当存在有效载荷时，有 2 个选项可用于对齐与数据路径有关的有效载荷的第一个字节。

1. Dword 对齐模式：在此模式下，只要存在有效载荷，描述符字节后紧接的就是下一个 Dword 位置中的有效载荷字节。
2. 地址对齐模式：在此模式下，有效载荷可从数据路径上的任意字节位置开始。对于从集成块传输到用户应用的数据，第 1 个字节的位置判定方式为

```
n = A mod w
```

其中 A 表示描述符中指定的存储器或 I/O 地址（针对报文和配置请求，采用的地址为 0），而 w 则表示已配置的数据总线宽度（以字节数为单位）。描述符的结束位置与有效载荷的第 1 个字节的起始位置之间的间隔全部以空字节来填充。

对于从集成块传输到用户应用的数据，数据对齐方式根据用户存储器中数据块的目标起始位置来判定。对于从用户应用传输到集成块的数据，使用地址对齐模式时，用户应用必须使用 tuser 边带信号向集成块明确标示第 1 个字节的位置。

在地址对齐模式下，不允许有效载荷与描述符重叠。即，当发射器已发射描述符之后，会开始 1 个新节拍以启动有效载荷的传输。发射器会以空字节填充描述符的最后 1 个字节与有效载荷的第 1 个字节之间的所有间隔。

512 位接口：

当存在有效载荷时，有 2 个选项可用于对齐与数据路径有关的有效载荷的第 1 个字节。

1. Dword 对齐模式：在此模式下，只要存在有效载荷，描述符字节后紧接的就是下一个 Dword 位置中的有效载荷字节。如果 D 表示描述符大小（以字节为单位），那么对应于有效载荷的第 1 个字节的通道编号的判定方式为：

```
n = (S + D + (A mod 4)) mod 64
```

其中 S 表示描述符的第 1 个字节出现的通道编号（有效值包括 0、16、32 或 48），D 表示描述符宽度（可以是 12 字节或 16 字节），A 表示用户存储器中数据块的第 1 个字节的地址（对于报文和配置请求，采用的地址为 0）。

2. 128b 地址对齐模式：在此模式下，512 位总线上有效载荷的起始位置与 128 位边界对齐。对应于有效载荷的第一个字节的通道编号的判定方式为：

```
n = (S + 16 + (A mod 16)) mod 64
```

其中 S 表示描述符的第 1 个字节出现的通道编号（有效值包括 0、16、32 或 48），A 表示对应于有效载荷的第一个字节的存储器或 I/O 地址（对于报文和配置请求，采用的地址为 0）。描述符的结束位置与有效载荷的第一个字节的起始位置之间的间隔全部以空字节来填充。

在 4 个用户接口之间，用于数据对齐的地址 A 的源各不相同，如下所述：

- CQ 接口：对于通过 CQ 接口从核传输到用户应用的数据，用于对齐的地址位为描述符中指定的下位地址，即用户存储器中数据块的起始地址。
- CC 接口：对于通过 CC 接口从用户应用传输到核的完成数据，基于用户在描述符中提供的地址位来进行对齐。
- RQ 接口：对于通过 RQ 接口从用户应用传输到核的存储器请求，根据用户随使用边带信号的请求一起提供的地址位来进行对齐。用户可以为 A 指定任意值，与描述符中地址字段的设置无关。
- RC 接口：对于通过 RC 接口从核传输到用户应用的完成数据，根据用户随使用边带信号（在 RQ 接口上发出该信号时）一起提供的地址位来进行对齐。核会保存来自请求的对齐信息，并在通过 RC 接口交付完成包有效载荷时，将此信息用于对齐对应完成包的有效载荷。

128b 地址对齐模式将 512 位 AXI 节拍分割为 4 个子节拍，每拍 128 位。有效载荷只能在紧接在描述符之后的子节拍中开始。不允许有效载荷与描述符在同一子节拍中重叠。发射器会以空字节填充描述符的最后 1 个字节与有效载荷的第一个字节之间的所有间隔。

通过设置 IP 自定义 GUI 即可为请求器接口（RQ 或 RC）和完成器接口（CQ 或 CC）单独选择对齐模式。

注释：如果性能是设计中的关键要素，那么应使用 Dword 对齐模式代替地址对齐模式。

Vivado® IP 目录会将数据对齐选项全局应用于全部 4 个接口。但高级用户可以为这 4 个 AXI4-Stream 接口分别选择对齐模式。此操作可通过设置对应的对齐模式参数来完成。请参阅 [64/128/256 位完成器接口](#) 和 [512 位完成器接口](#) 以获取有关地址对齐和示例图的更多详细信息。

CQ、CC 和 RQ 接口上的跨接选项

CQ、CC 和 RQ 接口具有跨接选项，允许在同一拍内通过该接口传输最多 2 个 TLP。这样可以为小型 TLP 提升吞吐量，并且当 TLP 在上半拍内结束时同样很有用。在 Vivado® IDE 中进行核配置期间可以为每个接口单独启用跨接选项。跨接选项只能配合 Dword 对齐模式一起使用。

RC 接口上的跨接选项

RC 接口支持跨接选项，允许在同一拍内通过该接口传输最多 4 个 TLP。在 Vivado® IDE 中进行核配置期间可启用该选项。启用该选项后，核可在字节通道 0、16、32 或 48 上启动新的完成 TLP。因此，启用该选项后，核即可在同一拍内在 AXI 总线上发送完整的 4 个完成 TLP，前提是每个完成 TLP 的有效载荷大小都不超过 1 个 Dword。仅当 RC 接口配置为采用 Dword 对齐模式时，才可使用跨接选项。

当请求器完成 (RC) 接口配置为 256 位或 512 位的宽度时，根据 TLP 的类型以及有效载荷大小，如果允许在接口每一拍内最多开始或结束 1 个 TLP（针对 256 位接口）或 2 个 TLP（针对 512 位接口），那么接口使用效率可能明显下降。当颁发的接收器信用值无限时，这种 RC 接口使用效率过低现象可能导致完成 FIFO 发生上溢。您必须完成以下任一操作：

- 限制未完成的非转发请求数量，以便使接收到的完成包总数保持在 64 以下并且不超过所选完成 FIFO 大小。如果您有 Gen3X16 -2LV 速度等级的器件，FIFO 大小设为 32 KB，对于所有其它器件或配置，大小设为 64 KB；或者
- 使用 RC 接口跨接选项。请参阅对应 256 位（[图 68：在启用跨接选项的请求器完成接口上执行完成 TLP 的传输](#)）和 512 位（[图 100：在启用跨接选项的请求器完成接口上执行完成 TLP 的传输](#)）的波形图，其中显示了该选项。

跨接选项仅限在 256 位或 512 位宽的 RC 接口上才可供使用，可通过 Vivado IP 目录启用。请参阅 [第 5 章：设计流程步骤](#)，以获取有关在 IP 目录 (IP catalog) 中启用该选项的指示信息。启用该选项时，当前一个 TLP 在同一拍内终止于字节通道 15/31/47 或者在这些通道之前终止时，集成块可在字节通道 16/32/48 上启动新的完成 TLP。因此，启用该选项后，集成块即可在同一拍内在 RC 接口上发送完整的多个完成 TLP，前提是每个完成 TLP 的有效载荷都不超过 1 个 Dword。

仅当接口宽度设置为 256 位或 512 位并且 RC 接口设置为 Dword 对齐模式时，跨接设置才可用。

下表列出了接口宽度、寻址模式和跨接选项的有效组合。

表 48：接口宽度、对齐模式和跨接的有效组合

接口宽度	对齐模式	跨接选项	描述
64 位	Dword 对齐	不适用	64 位，Dword 对齐
64 位	地址对齐	不适用	64 位，地址对齐
128 位	Dword 对齐	不适用	128 位，Dword 对齐
128 位	地址对齐	不适用	128 位，地址对齐
256 位	Dword 对齐	禁用	256 位，Dword 对齐，禁用跨接
256 位	Dword 对齐	启用	256 位，Dword 对齐，启用跨接（仅限请求器完成接口）
256 位	地址对齐	不适用	256 位，地址对齐
512 位	Dword 对齐	禁用	512 位，Dword 对齐，禁用跨接

表 48：接口宽度、对齐模式和跨接的有效组合 (续)

接口宽度	对齐模式	跨接选项	描述
512 位	Dword 对齐	启用	512 位, Dword 对齐, 启用跨接 (针对所有接口允许 2-TLP 跨接, 仅限针对请求器完成接口才允许 4-TLP 跨接)
512 位	地址对齐	不适用	512 位, 128 位地址对齐

接收传输事务排序

核在其接收侧包含相应的逻辑，用于确保从链路接收到的 TLP 以及交付至其完成器请求接口和请求器完成接口的 TLP 不会违反 PCI Express® 传输事务排序约束。集成块基于下列关键规则来执行排序操作：

- 转发请求必须能够在完成器请求 (CQ) 接口上传递非转发请求。为支持此功能，集成块会在 CQ 接口上实现流量控制机制，用户逻辑可通过此机制来控制非转发请求的流量，而不会影响转发请求。用户逻辑可通过断言 `pcie_cq_np_req[0]` 信号有效来标示接收非转发请求的缓冲器的可用情况。
- 仅当可用信用值为非 0 值时，集成块才会向用户应用交付非转发请求。当非转发请求交付因缺少信用值而暂停时，集成块会持续交付转发请求。当信用值机制对非转发请求的交付不施加任何反压时，集成块会按从链路接收到转发请求和非转发请求的顺序来交付这些请求。如需获取有关控制非转发请求流程的更多信息，请参阅 [针对非转发请求的选择性流量控制](#)。
- PCIe 排序要求不允许完成 TLP 传递转发请求，但存在下列例外情况：
 - 已设置“Relaxed Ordering”（宽松排序）属性位的完成包可传递转发请求。
 - 已设置基于 ID 的排序位的完成包可传递转发请求，前提是完成器 ID 与转发请求器 ID 不同。

只有在先前到达的所有转发 TLP 全部完成传输之后，集成块才会在请求器完成 (RC) 接口上开始传输从链路接收到的完成 TLP，除非适用下列 2 项规则之一。

在将 TLP 完全传输到用户接口之后，将由用户应用负责根据需要强制执行排序约束。

表 49：接收排序规则

行传递	转发	非转发	完成
转发	否	是	是
非转发	否	否	是
完成	a) 否 b) 是 (宽松排序)	是	否

发射传输事务排序

在发射侧，集成块在 2 个不同接口上接收 TLP：请求器请求 (RQ) 接口和完成器完成 (CC) 接口。集成块不会对从每个接口接收到的传输事务进行重新排序。在集成块的发射流水线中，当请求器侧的请求和完成器侧的完成包已多路复用到单一数据流后，其排序方式将变得非常难以预测。如果完成 TLP 必须保持其相对于请求的顺序不变，那么对于需要严格保留排序（相对于从 CC 接口发射的完成包）不变的任意请求，用户逻辑可以在 `s_axis_rq_tuser` 总线中的 `seq_num[3:0]` 输入上为其提供 4 位序列号。集成块会将此序列号置于其 `pcie_rq_seq_num[3:0]` 输出上，当发射流水线中的请求 TLP 进展至特定状态（即，来自用户应用的所有新完成 TLP 都无法继续传递该包）时，集成块会断言 `pcie_rq_seq_num_vld` 有效。在以下情况下，此机制可用于保留 TLP 排序不变：

- 用户逻辑要求保留请求 TLP 与紧随其后的完成 TLP 之间的排序不变。在此情况下，用户逻辑必须等待请求器请求的序列号显示在 `pcie_rq_seq_num[3:0]` 输出上之后，方可在目标完成接口上开始传输完成 TLP。

- 用户逻辑要求保留请求 TLP 与 MSI/MSI-X TLP（通过 MSI 报文接口发送信号）之间的排序不变。在此情况下，用户逻辑必须等待请求器请求的序列号显示在 `pcie_rq_seq_num[3:0]` 输出上之后，才能在 MSI 报文接口上发出 MSI 或 MSI-X 信号。

64/128/256 位完成器接口

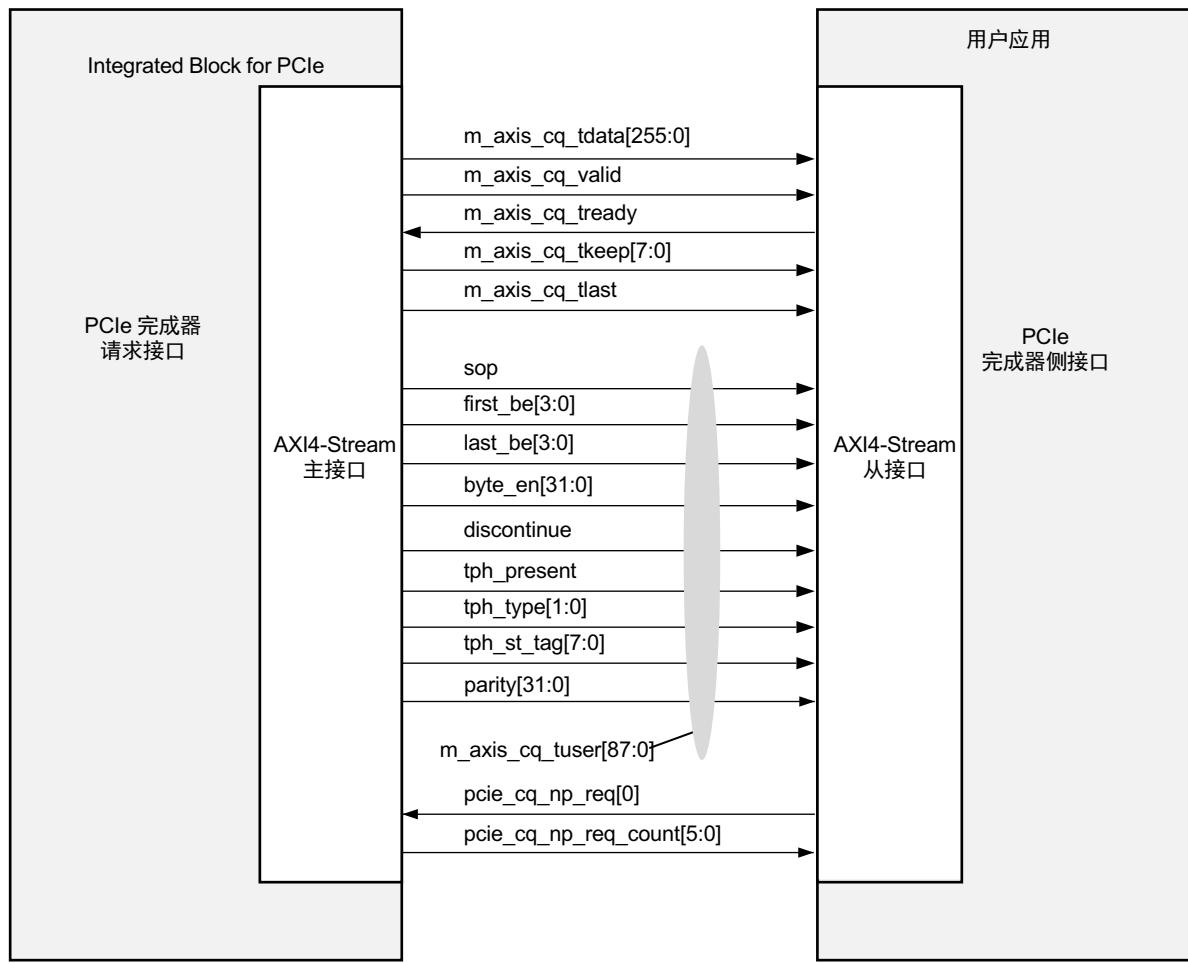
本章节描述了对应 64/128/256 位接口的用户接口的操作。

此接口可基于 AXI4-Stream 协议，将从 PCIe 链路接收到的传输事务（存储器、I/O 读写、报文、原子操作）映射到完成器请求 (CQ) 接口上的传输事务。完成器接口包含 2 个独立接口，对应每个方向的数据传输使用 1 个接口。每个接口都基于 AXI4-Stream 协议，其宽度可配置为 64、128 或 256 位。CQ 接口用于将请求（含任意关联的有效载荷数据）传输到用户应用，完成器完成 (CC) 接口则用于从用户应用接收完成包数据（针对非转发请求）以供通过链路进行转发。这 2 个接口均单独运行。即，集成块可通过 CQ 接口传输新请求，同时接收前一个请求的完成包。

完成器请求接口操作

下图显示了与核的完成器请求接口关联的信号。核将此接口上的每个 TLP 均作为 1 个 AXI4-Stream 数据包来交付。对于含有效载荷的 TLP，此数据包以 128 位描述符开头并后接数据。

图 16：完成器请求接口信号



X19416-091522

完成器请求接口支持 2 种不同的数据对齐模式。在 Dword 对齐模式下，有效数据的第一个字节显示在通道 $n = (16 + A \bmod 4) \bmod w$ 中，其中，A 是要传输的数据的字节级别起始地址，而 w 则是接口宽度（以字节为单位）。

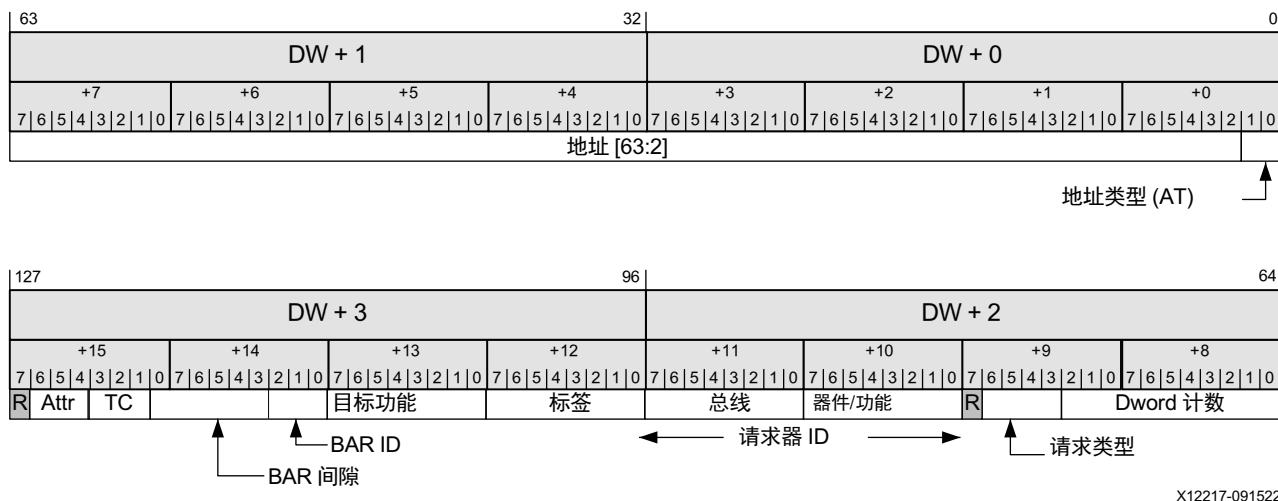
在地址对齐模式下，数据始终在描述符终止后的新的节拍中开始，其首个有效字节位于通道 $n = A \bmod w$ 上，其中 w 是接口宽度（以字节为单位）。对于存储器、I/O 和原子操作请求，地址 A 为请求中包含的地址。对于报文，地址始终为 0，以便判定其有效载荷的对齐位置。

完成器请求描述符格式

集成块将从链路接收到的每个请求 TLP 作为独立 AXI4-Stream 数据包通过 CQ 接口进行传输。每个数据包均以 1 个描述符开头，在描述符后可包含有效载荷数据。描述符长度始终为 16 字节，并在请求包的前 16 字节内发送。描述符在 64 位接口上的前 2 个节拍内进行传输，在 128 位或 256 位接口上的第 1 个节拍内进行传输。下图演示了不同类型的请求的描述符格式。

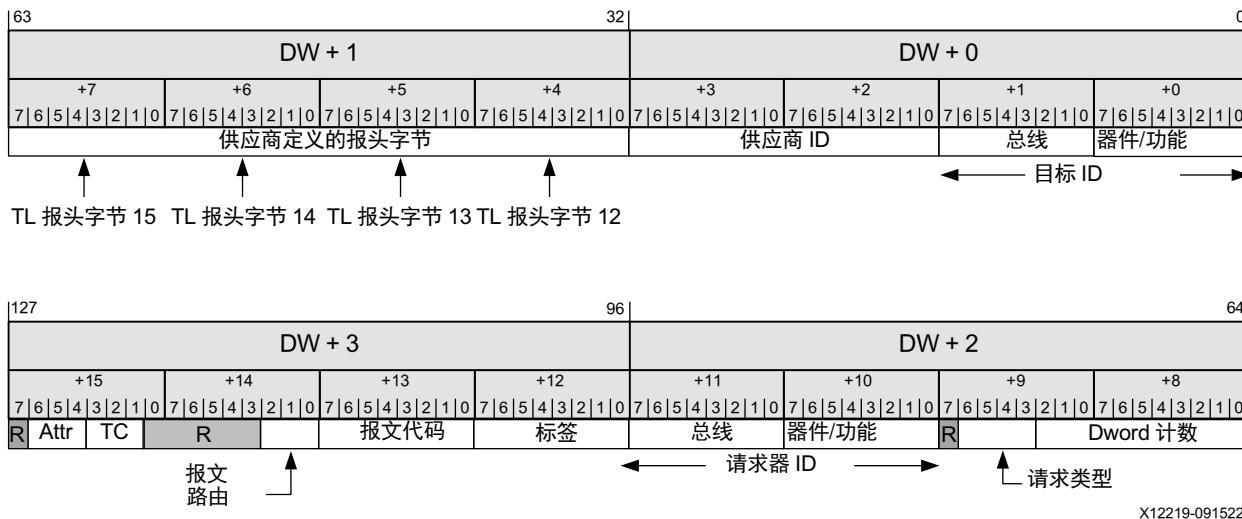
当所传输的请求 TLP 为存储器读取/写入请求、I/O 读取/写入请求或原子操作 (Atomic Operation) 请求时，即适用下图所示格式。

图 17：对应存储器、I/O 和原子操作请求的完成器请求描述符格式



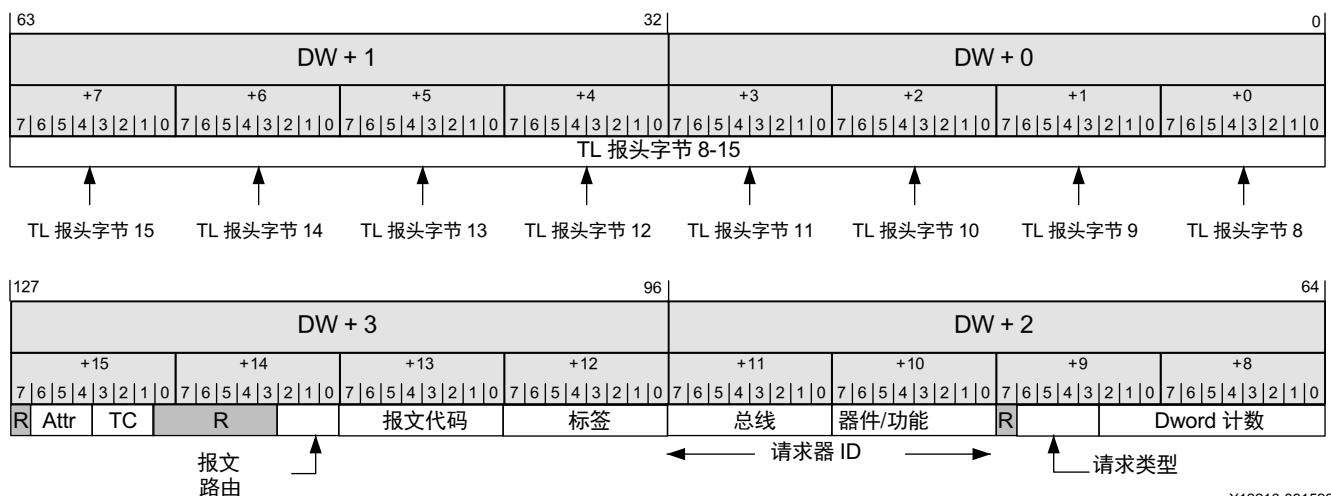
下图格式仅适用于类型 0 或类型 1 的“Vendor-Defined Messages”（供应商定义的报文）。

图 18：对应于供应商定义的报文的完成器请求描述符



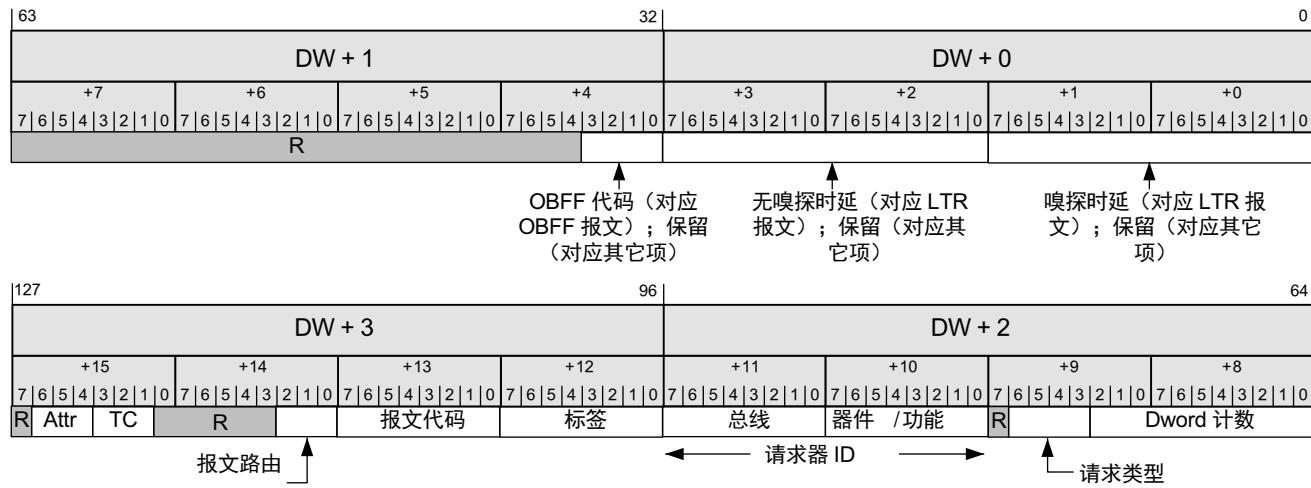
下图所示格式适用于所有 ATS 报文，包括“Invalid Request”（无效请求）、“Invalid Completion”（无效完成）、“Page Request”（页面请求），“PRG Response”（PRG 响应）。

图 19：对应于 ATS 报文的完成器请求描述符



对于所有其它报文，描述符采用下图所示格式。

图 20：对应于所有其它报文的完成器请求描述符格式



下表描述了完成器请求描述符的每个字段。

表 50：完成器请求描述符字段

位索引	字段名称	描述
1:0	“Address Type” (地址类型)	该字段定义仅适用于存储器传输事务和原子操作。其中包含从请求的 TL 报头抽取的 AT 位。 <ul style="list-style-type: none"> · 00: 请求中的地址未经转换 · 01: 传输事务为转换请求 · 10: 请求中的地址为已转换的地址 · 11: 保留
63:2	“Address” (地址)	该字段适用于存储器、I/O 和原子操作请求。它可提供来自 TLP 报头的地址。这是请求所引用的首个 Dword 的地址。必须使用来自 m_axis_cq_tuser 的 First_BE 位来判定字节级别的地址。 当此传输事务指定 32 位地址时，此字段的位 [63:32] 为 0。
74:64	“Dword Count” (Dword 计数)	这 11 个位用于指示要读取或写入的块大小（以 Dword 为单位），对于报文，这些位指示报文有效载荷的大小。其范围为 0 - 256 个 Dword。对于 I/O 访问，Dword 计数始终为 1。 对于长度为 0 的存储器读取/写入请求，Dword 计数为 1，且 First_BE 位全部设置为 0。
78:75	“Request Type” (请求类型)	用于识别传输事务类型。 表 51：传输事务类型 中列出了传输事务类型及其编码。
95:80	“Requester ID” (请求器 ID)	与请求关联的 PCI 请求器 ID。在 RID 的传统解读中，这 16 个位分割为 8 位总线编号 [95:88]、5 位器件编号 [87:83] 和 3 位功能编号 [82:80]。启用 ARI 时，位 [95:88] 包含 8 位总线编号，而位 [87:80] 则提供功能编号。 如果请求属于非转发传输事务，那么用户完成器应用必须存储该字段，并将其与完成数据一起重新提供给集成块。
103:96	“Tag” (标签)	与请求关联的 PCIe 标签。如果请求属于非转发传输事务，那么用户逻辑必须存储该字段，并将其与完成数据一起重新提供给集成块。针对存储器写入和报文，可忽略该字段。

表 50：完成器请求描述符字段 (续)

位索引	字段名称	描述
111:104	“Target Function” (目标功能)	<p>该字段定义仅适用于存储器、I/O 和原子操作请求。它可提供请求的目标功能编号（由 BAR 检查来判定）。使用 ARI 时，该字段的全部 8 位都有效。否则，仅限位 [106:104] 有效。</p> <p>以下是目标功能值到 PF/VF 映射的映射：</p> <ul style="list-style-type: none">· 0: PF0· 1: PF1· 2: PF2· 3: PF3· 4: VF0· 5: VF1· 6: VF2· 7: VF3
114:112	BAR ID	<p>该字段定义仅适用于存储器、I/O 和原子操作请求。它可为请求中的地址提供匹配的 BAR 编号。</p> <p>在 RP 模式下，BAR ID 始终为 000。</p> <ul style="list-style-type: none">· 001: BAR 1 (针对 VF 为 VF-BAR 1)· 010: BAR 2 (针对 VF 为 VF-BAR 2)· 011: BAR 3 (针对 VF 为 VF-BAR 3)· 100: BAR 4 (针对 VF 为 VF-BAR 4)· 101: BAR 5 (针对 VF 为 VF-BAR 5)· 110: 扩展 ROM 访问 <p>对于 64 位传输事务，所提供的 BAR 编号为匹配的 BAR 对的下位地址（即，0、2 或 4）。</p>
120:115	“BAR Aperture” (BAR 间隙)	<p>该 6 位字段定义仅适用于存储器、I/O 和原子操作请求。它可提供与请求匹配的 BAR 的间隙设置。此信息适用于判定在存储器或 I/O 空间寻址过程中将使用的位。例如，值为 12 表示匹配的 BAR 的间隙为 4K，因此，用户应用可以忽略地址的位 [63:12]。</p> <p>对于 VF BAR，此输出上提供的值基于 BAR 所覆盖的单一 VF 耗用的存储器空间。</p>
123:121	“Transaction Class (TC)” (传输事务类)	<p>与请求关联的 PCIe 传输事务类 (TC)。如果请求属于非转发传输事务，那么用户完成器应用必须存储该字段，并将其与完成数据一起重新提供给集成块。</p>
126:124	“Attributes” (属性)	<p>这些位可提供与请求关联的 Attribute 位的设置。位 124 为 “No Snoop” (无嗅探) 位，位 125 则为 “Relaxed Ordering” (宽松排序) 位。位 126 为 “ID-Based Ordering” (基于 ID 排序) 位，只能针对存储器请求和报文进行设置。</p> <p>如果请求属于非转发传输事务，那么用户完成器应用必须存储该字段，并将其与完成数据一起重新提供给集成块。</p>
15:0	“Snoop Latency” (嗅探时延)	<p>该字段定义仅适用于 LTR 报文。它可提供报文的 TLP 报头中的 16 位 “Snoop Latency” (嗅探时延) 字段的值。</p>
31:16	“No-Snoop Latency” (无嗅探时延)	<p>该字段定义仅适用于 LTR 报文。它可提供报文的 TLP 报头中的 16 位 “No-Snoop Latency” (无嗅探时延) 字段的值。</p>
35:32	“OBFF Code” (OBFF 代码)	<p>该字段定义仅适用于 OBFF 报文。“OBFF Code” 字段用于区分各种 OBFF 用例：</p> <ul style="list-style-type: none">· 1111b: CPU Active - 系统完全处于活动状态，可执行包括总线主控和中断在内的所有器件操作。· 0001b: OBFF - 可用于器件存储器读取/写入总线主控活动的系统存储器路径。· 0000b: Idle - 系统处于空闲低功耗状态。· 所有其它代码均为保留值。

表 50：完成器请求描述符字段 (续)

位索引	字段名称	描述
111:104	“Message Code” (报文代码)	该字段定义适用于所有报文。其中包含从 TLP 报头提取的 8 位报文代码。《PCI Express 基本规范第 3.0 版》的附录 F 提供了受支持的报文代码的完整列表。 用户应将含不受支持的报文代码的描述符作为 UR 来处理，切换 cfg_err_uncor_in 信号以指示检测到非致命 (Non-fatal) 错误。
114:112	“Message Routing” (报文路由)	该字段定义适用于所有报文。这些位可提供来自 TLP 报头的 3 位路由 (Routing) 字段 r[2:0]。
15:0	“Destination ID” (目标 ID)	该字段仅适用于供应商定义的报文。当报文按 ID 进行路由（即，当“Message Routing”(报文路由) 字段为 010 二进制值）时，该字段可提供报文的“Destination ID”(目标 ID)。
63:32	“Vendor-Defined Header” (供应商定义的报头)	该字段仅适用于供应商定义的报文。它包含从 TLP 报头的 Dword 3 抽取的字节。
63:0	“ATS Header” (ATS 报头)	该字段仅适用于 ATS 报文。它包含从 TLP 报头的 Dword 2 和 3 抽取的字节。

表 51：传输事务类型

请求类型 (二进制)	描述
0000	存储器读取请求
0001	存储器写入请求
0010	I/O 读取请求
0011	I/O 写入请求
0100	存储器提取和添加请求
0101	存储器无条件交换请求
0110	存储器比较和交换请求
0111	锁定读取请求 (仅在传统器件中支持)
1000	类型 0 配置读取请求 (仅限在请求器侧)
1001	类型 1 配置读取请求 (仅限在请求器侧)
1010	类型 0 配置写入请求 (仅限在请求器侧)
1011	类型 1 配置写入请求 (仅限在请求器侧)
1100	任意报文 (ATS 报文和供应商定义的报文除外)
1101	供应商定义的报文
1110	ATS 报文
1111	保留

完成器存储器写入操作

以下时序图显示了当接口宽度分别配置为 64 位、128 位和 256 位时，从链路接收到存储器写入 TLP (采用 Dword 对齐模式) 后，通过完成器请求 (CQ) 接口来进行传输的过程。为便于演示，写入存储器的数据块的起始 Dword 地址假定为 $(m \times 32 + 1)$ ，其中 $m > 0$ 且为整数。其大小假定为 n 个 Dword，其中 $n = k \times 32 + 29$ 且 $k > 0$ 。

在 Dword 对齐模式和地址对齐模式下，传输均从 16 个描述符字节开始，紧随其后即为有效载荷字节。在数据包持续时间段内，`m_axis_cq_tvalid` 信号保持处于断言有效状态。您可随时通过使用 deasserting `m_axis_cq_tready` 断言此信号无效来延长任一节拍。AXI4-Stream 接口可发出 `m_axis_cq_tkeep` 信号（每个信号对应 1 个 Dword 位置）以指示数据包中的有效 Dword，包括描述符和描述符与有效载荷之间插入的任意空字节。即，从描述符的第一个 Dword 开始到有效载荷的最后一个 Dword 为止的所有 `tkeep` 位均连续设置为 1。在数据包传输期间，当数据包无法填满接口的完整宽度时，`tkeep` 位仅限在数据包的最后一拍内才能设为 0。在数据包的最后一拍内，`m_axis_cq_tlast` 信号始终断言有效。

CQ 接口在 `m_axis_cq_tuser` 总线上还包含“First Byte Enable”（首字节使能）位和“Last Byte Enable”（末字节使能）位。这些位在数据包的第一拍内有效，并用于指定有效载荷的第一个和最后一个 Dword 的有效字节。

`m_axis_cq_tuser` 总线还可提供多个参考信号，这些参考信号可用于简化与用户接口关联的逻辑，或者用于支持其它功能。在每个数据包的第一拍中，`sop` 信号均断言有效（前提是其描述符位于总线上）。字节使能输出 `byte_en[31:0]`（每个输出对应 1 个字节通道）可用于指示有效载荷中的有效字节。仅当对应通道内包含的有效载荷字节确实有效时，`byte_en` 位才会断言有效，即针对描述符或描述符与有效载荷之间的填充字节，则不会断言该信号有效。断言有效的字节使能位从有效载荷开始始终连续，除非有效载荷大小不超过 2 个 Dword。对于 1 个 Dword 写入或 2 个 Dword 写入，字节使能可以不连续。另一种特殊情况是对于长度为 0 的存储器写入，集成块会传输含单 Dword 的有效载荷，其中所有 `byte_en` 位均设为 0。因此，无论在任何情况下，用户逻辑均可直接使用 `byte_en` 信号来启用将关联字节写入存储器的操作。

在 Dword 对齐模式下，根据有效载荷的第一个有效字节的地址，在描述符的结束位置与有效载荷的第一个字节之间可能存在 0、1、2 或 3 个字节位置的间隔。有效载荷中第一个有效字节的实际位置可通过 `m_axis_cq_tuser` 总线中的 `first_be[3:0]` 或 `byte_en[31:0]` 来判定。

当接收到的 TLP 中存在“Transaction Processing Hint”（传输事务处理提示）时，集成块所传输的参数均为与 `m_axis_cq_tuser` 总线内的信号上的提示相关联的参数，提示分为：“TPH Steering Tag”（TPH 导向标签）和“Steering Tag Type”（导向标签类型）。

图 21：完成器请求接口上的存储器写入传输事务（Dword 对齐模式、64 位接口）

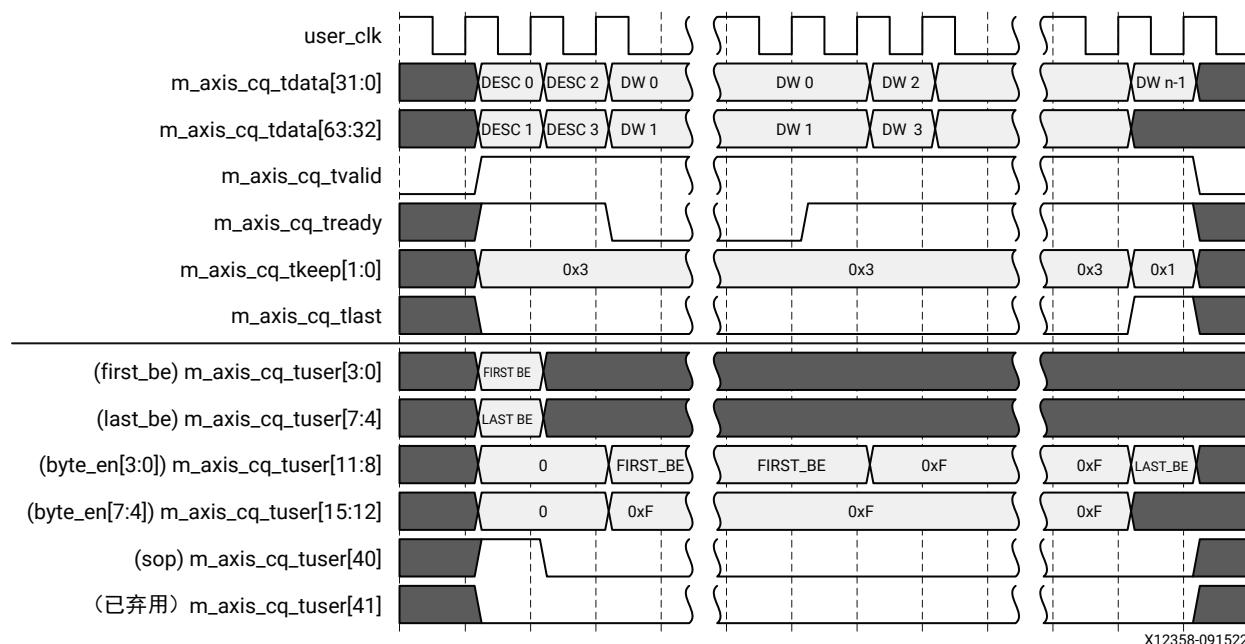
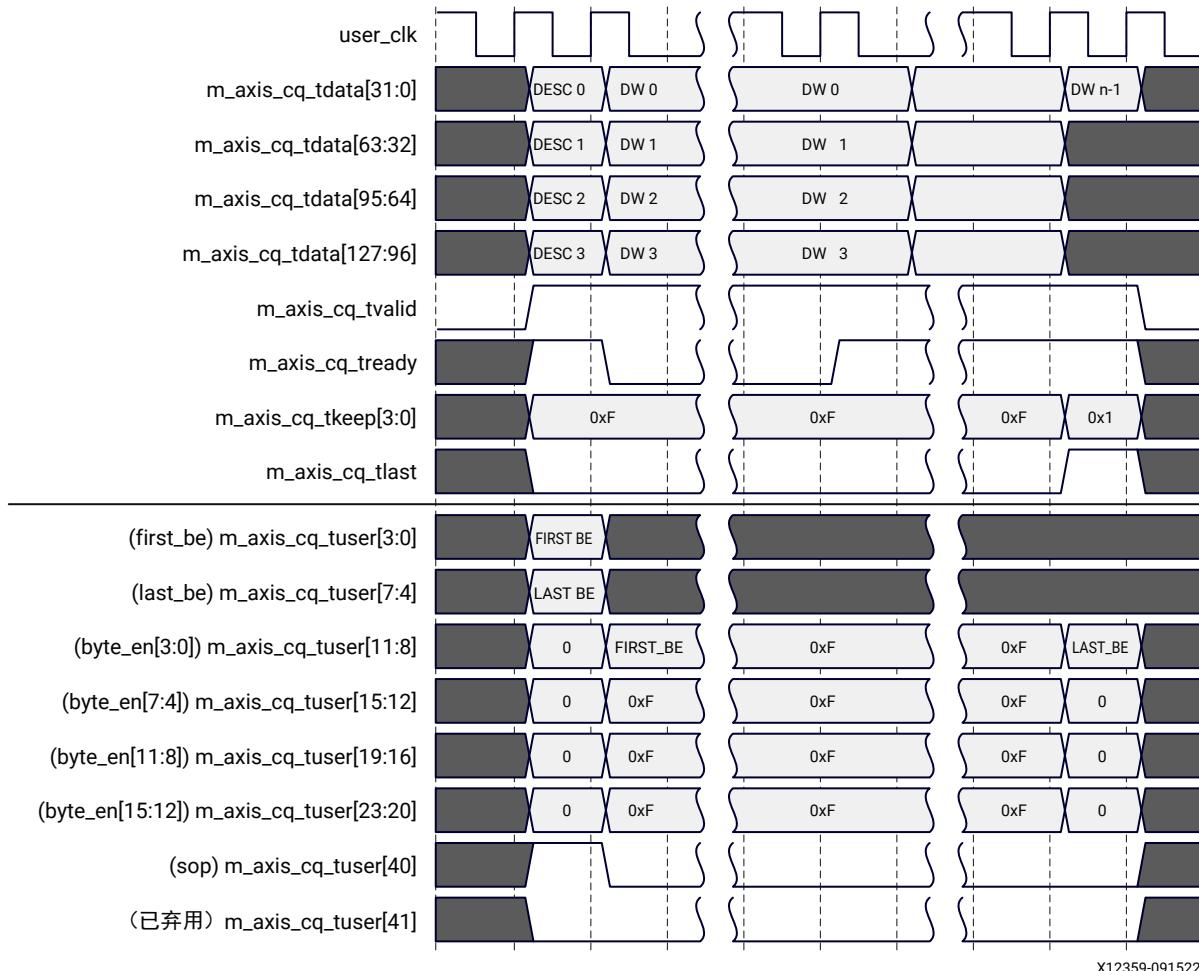
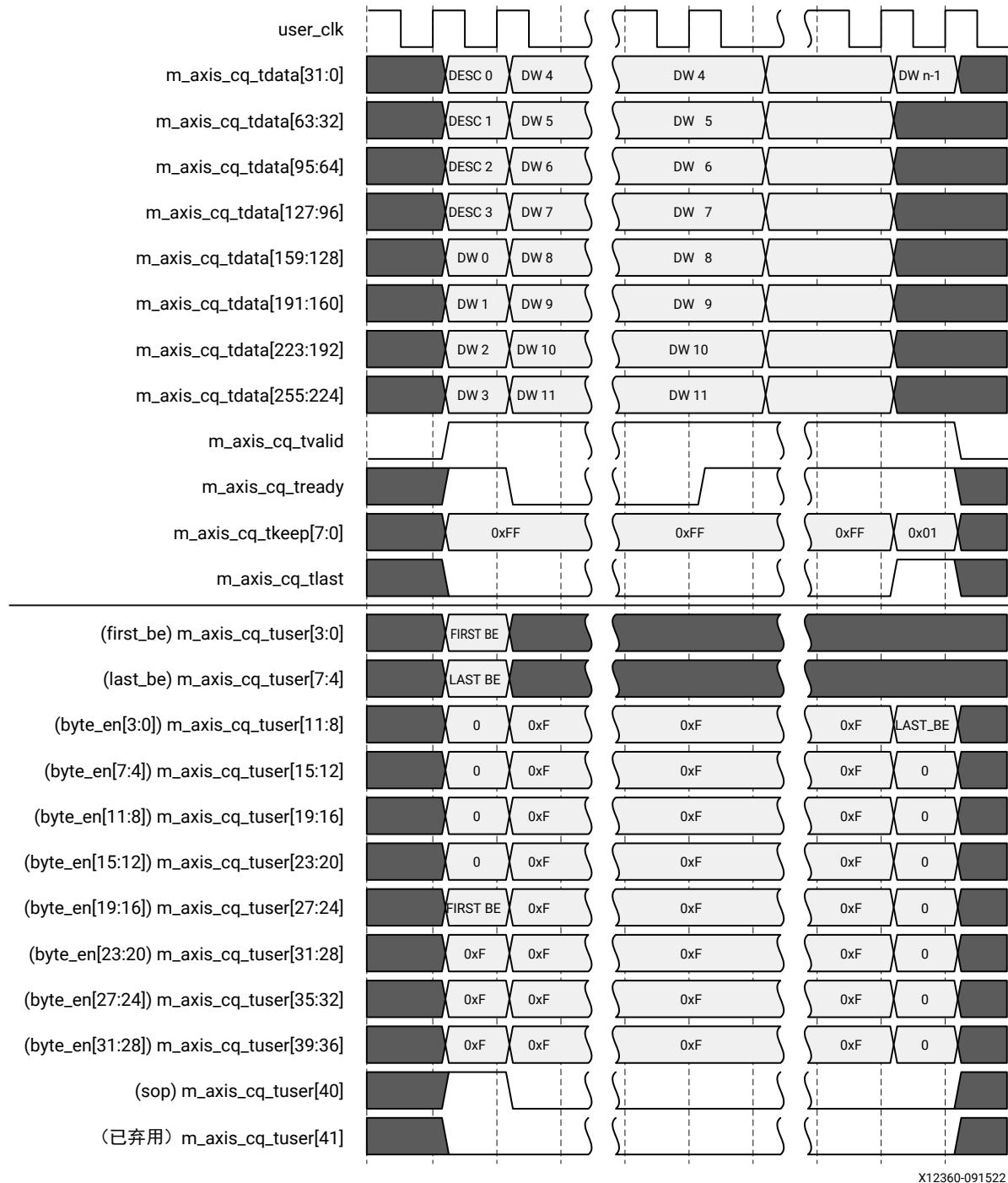


图 22：完成器请求接口上的存储器写入传输事务（Dword 对齐模式、128 位接口）



X12359-091522

图 23：完成器请求接口上的存储器写入传输事务（Dword 对齐模式、256 位接口）



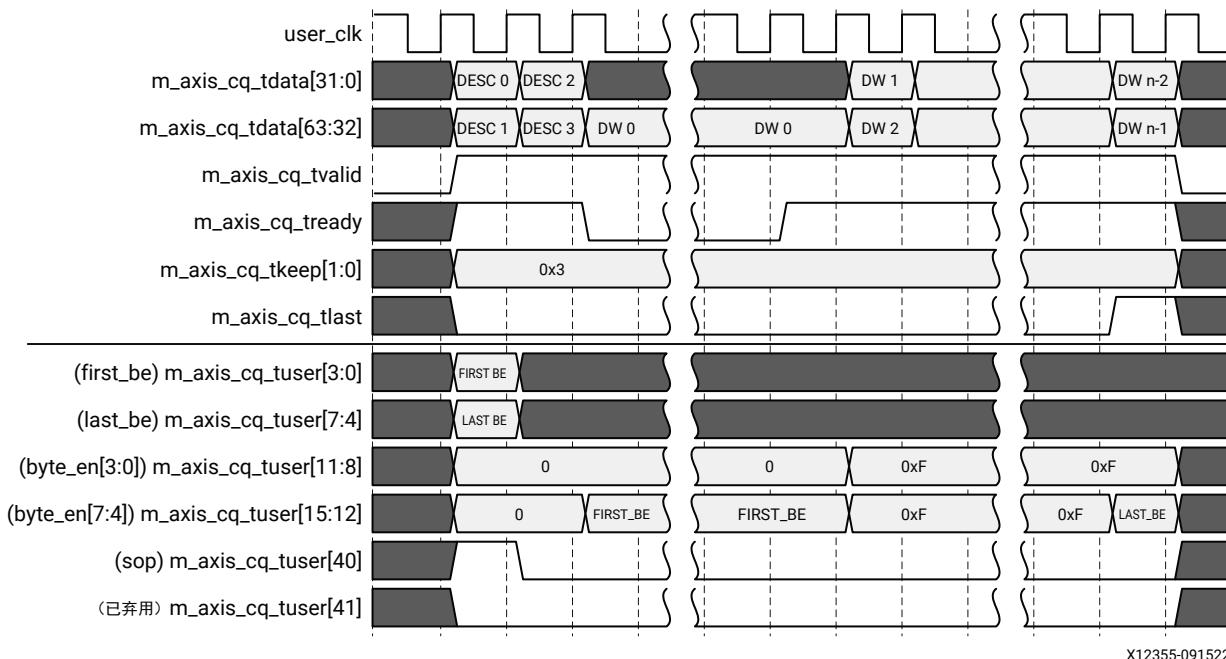
X12360-091522

以下时序图显示了当接口宽度分别配置为 64 位、128 位和 256 位时，从链路接收到存储器写入 TLP（采用地址对齐模式）后，通过 CQ 接口来进行传输的过程。为便于演示，写入存储器的数据块的起始 Dword 地址假定为 $(m \times 32 + 1)$ ，其中 $m > 0$ 且为整数。其大小假定为 n 个 Dword，其中 $n = k \times 32 + 29$ 且 $k > 0$ 。

在地址对齐模式下，有效载荷的交付始终从描述符的最后一个字节后的节拍中开始。根据有效载荷的第一个有效字节的地址，有效载荷的第一个字节可显示在任意字节通道上。此 keep 输出的 `m_axis_cq_tkeep` 在描述符与有效载荷之间的间隙内保持处于高电平有效状态。有效载荷中的第一个有效字节的实际位置可根据描述符中地址的最低有效位或者根据 `m_axis_cq_tuser` 总线中的字节使能位 `byte_en[31:0]` 来判定。

如果写入不超过 2 个 Dword，那么从有效载荷开始，`byte_en` 上值为 1 的位不能连续。对于长度为 0 的存储器写入，集成块会传输含单 Dword 的有效载荷，其中有效载荷字节的所有 `byte_en` 位均设为 0。

图 24：完成器请求接口上的存储器写入传输事务（地址对齐模式、64 位接口）



X12355-091522

图 25：完成器请求接口上的存储器写入传输事务（地址对齐模式、128 位接口）

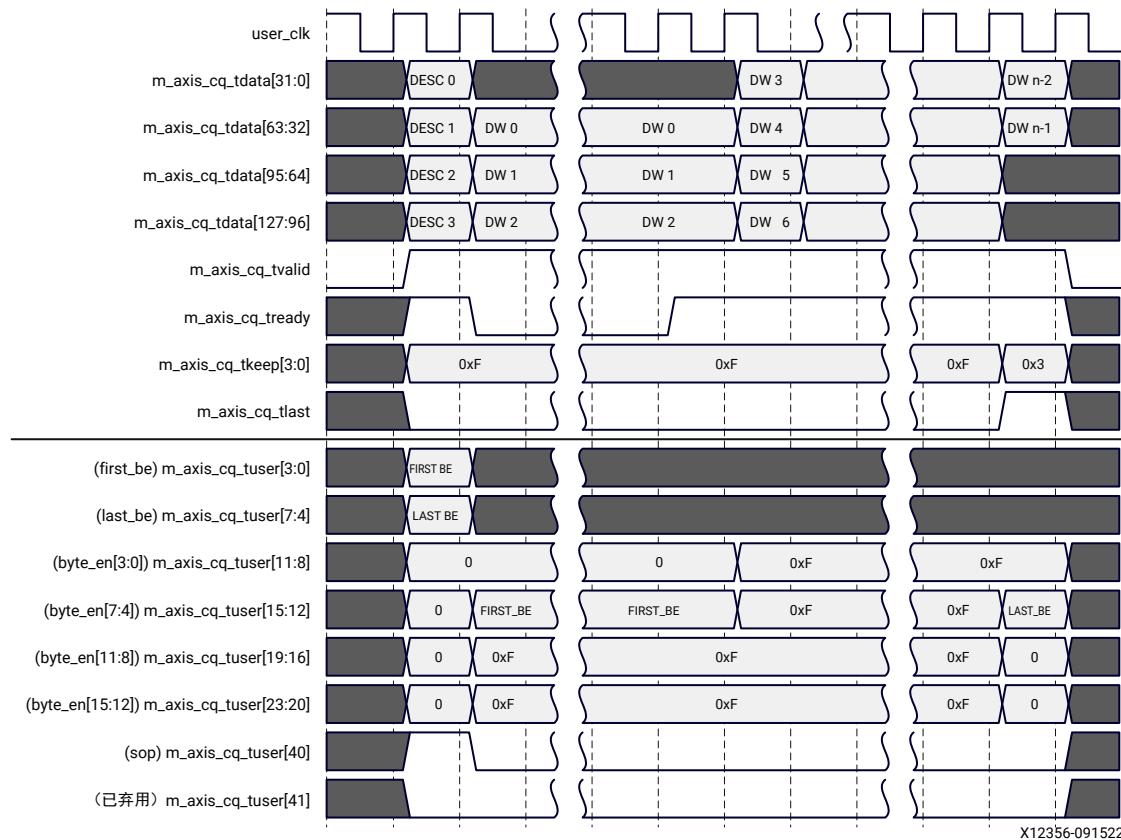
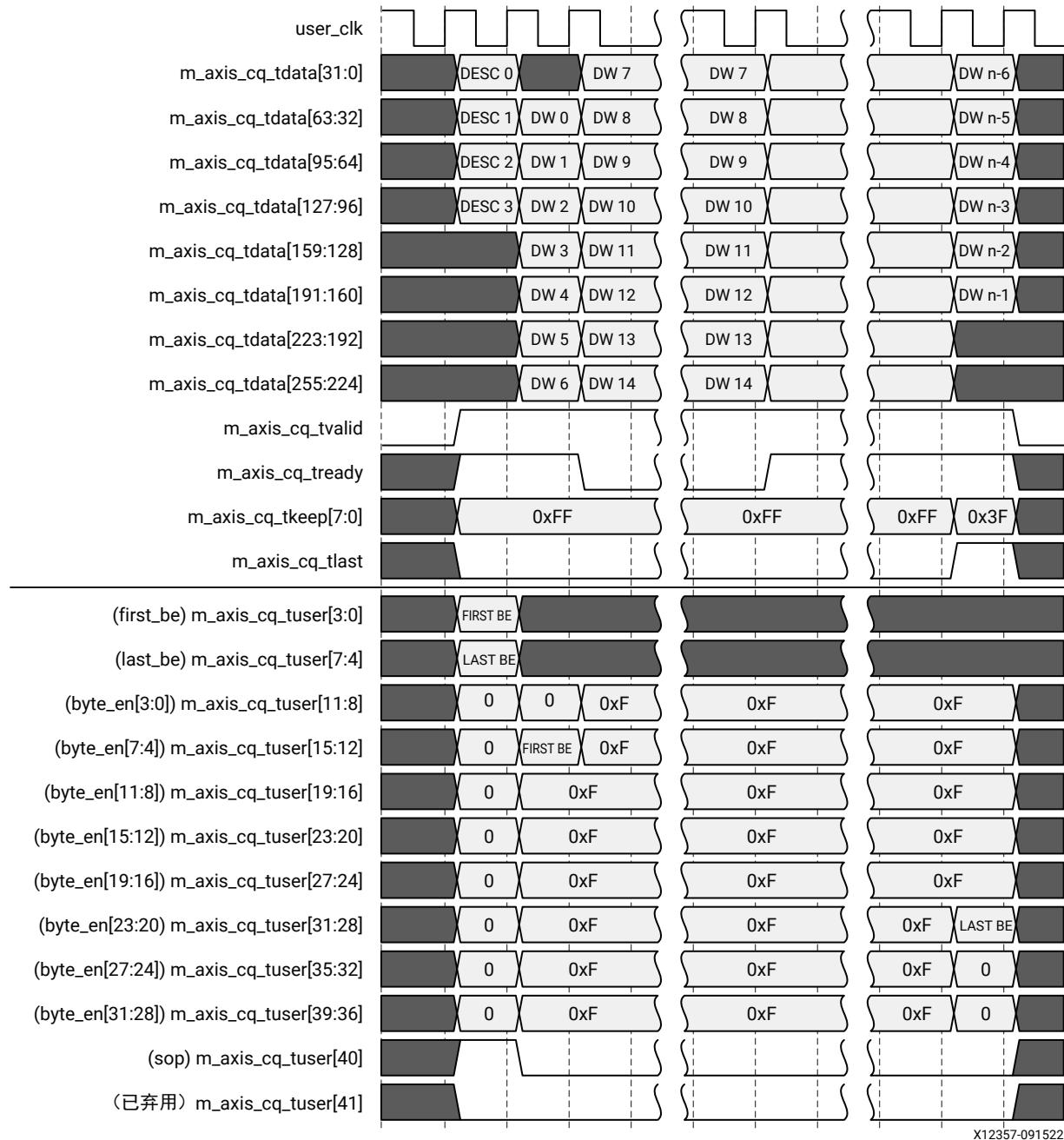


图 26：完成器请求接口上的存储器写入传输事务（地址对齐模式、256 位接口）



完成器存储器读取操作

存储器读取请求跨完成器请求接口进行传输的顺序与存储器写入请求相同，除非 AXI4-Stream 数据包仅包含 16 字节描述符。以下时序图显示了当接口宽度分别配置为 64 位、128 位和 256 位时，从链路接收到存储器读取 TLP 后，通过完成器请求接口来进行传输的过程。此数据包在 64 位接口上占据连续 2 个节拍，但在 128 位和 256 位接口上，其传输只需一拍即可。在数据包持续时间段内，**m_axis_cq_tvalid** 信号保持处于断言有效状态。您可随时通过断言 **m_axis_cq_tready** 无效来延长任一节拍。当第一个描述符字节位于总线上时，**m_axis_cq_tuser** 总线中的 **sop** 信号断言有效。

图 27：完成器请求接口上的存储器读取传输事务（64 位接口）

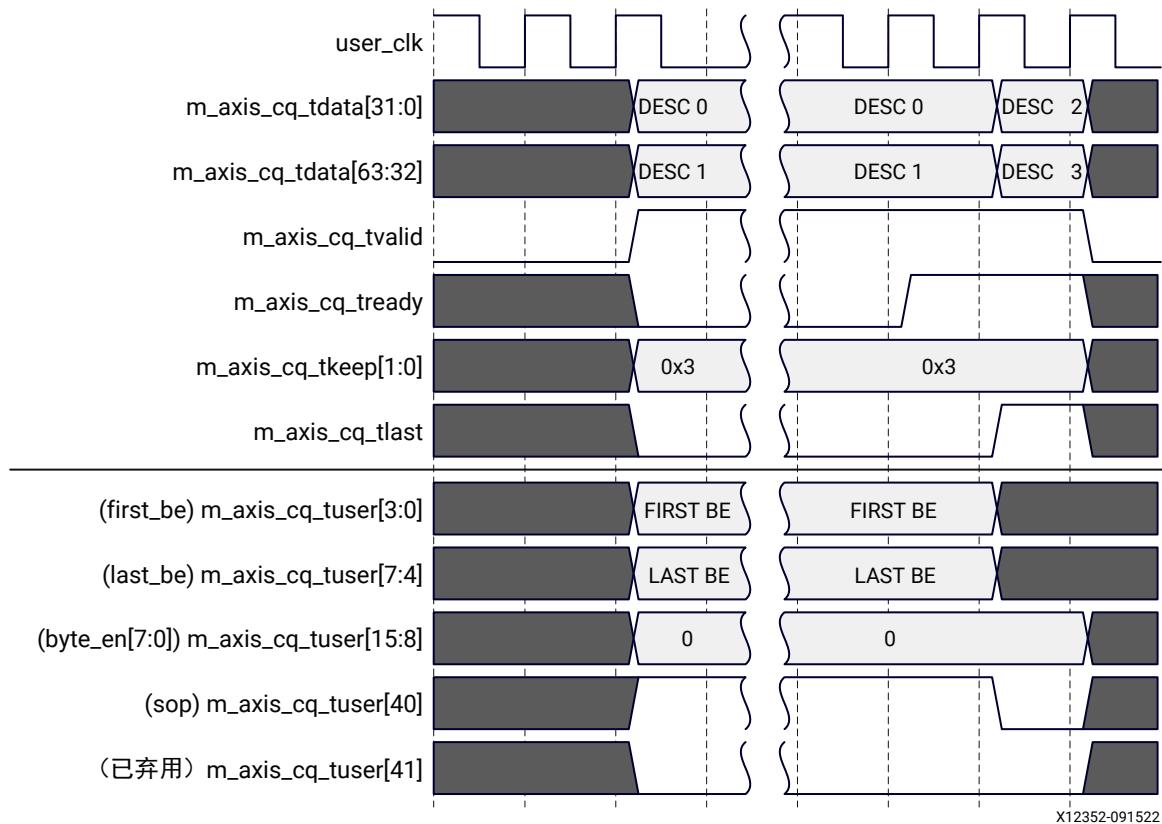


图 28：完成器请求接口上的存储器读取传输事务（128 位接口）

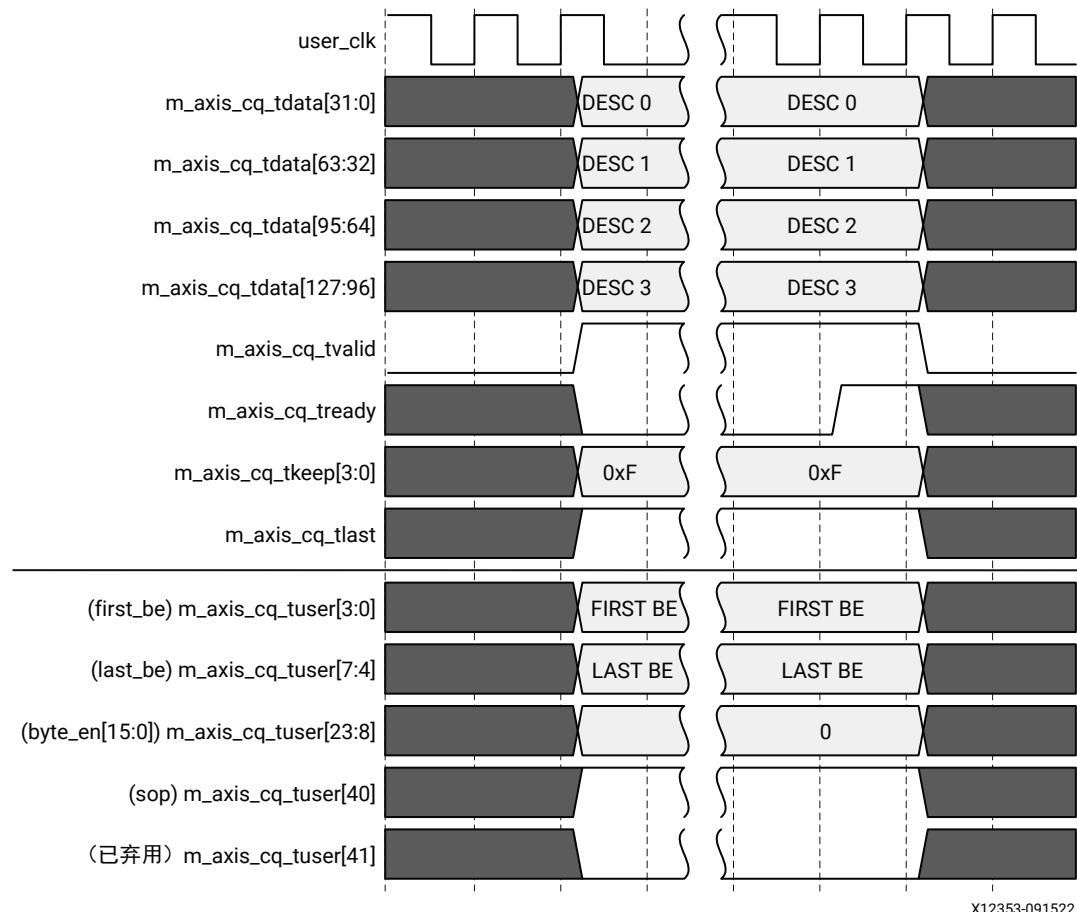
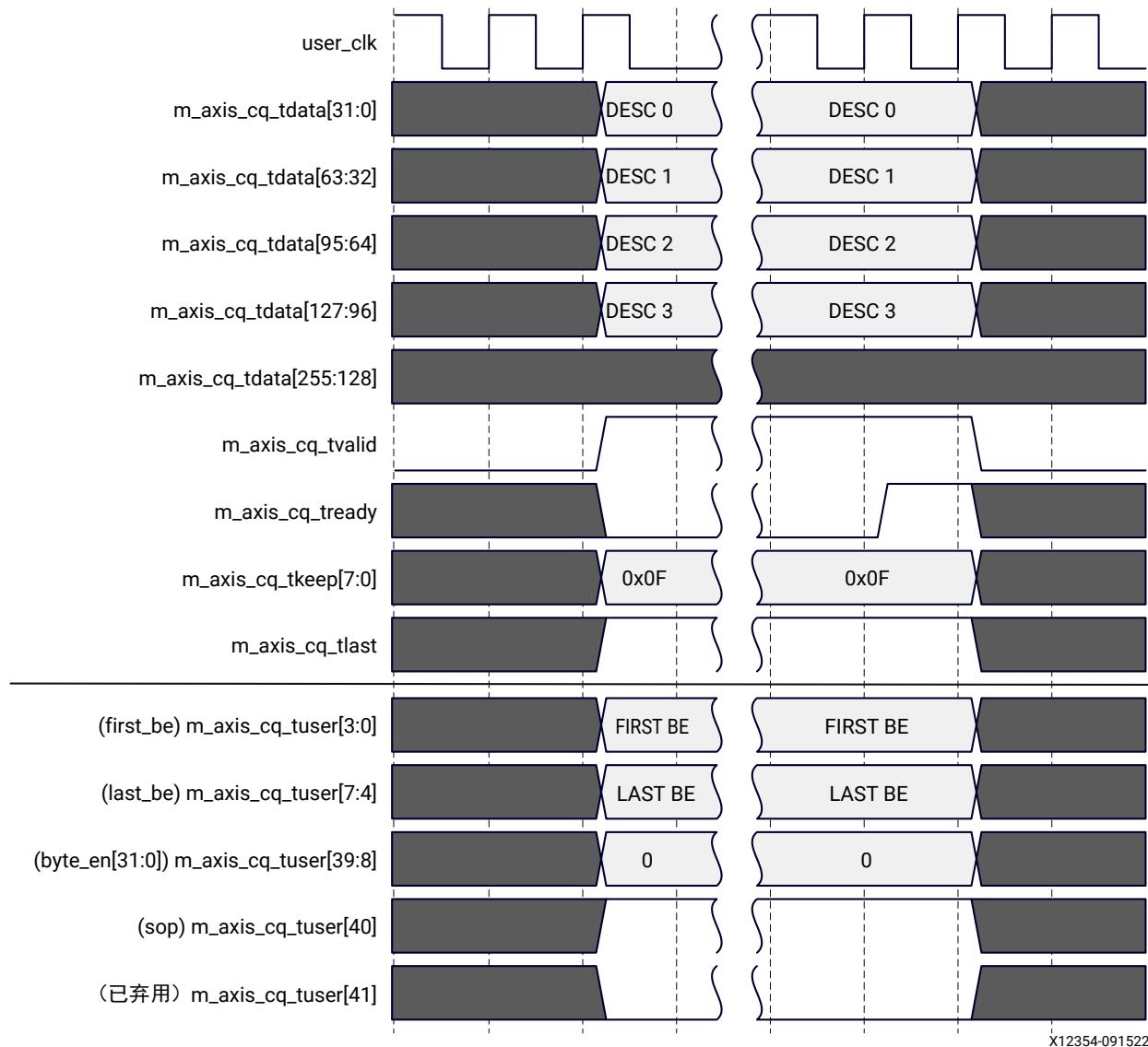


图 29：完成器请求接口上的存储器读取传输事务（256 位接口）



与第一个 Dword 和最后一个 Dword 的读取请求关联的字节使能位由 `m_axis_cq_tuser` 边带总线上的集成块提供。当传输第一个描述符字节时，这些位即为有效，并且必须使用这些位来判定字节级别的起始地址和与该请求关联的字节计数。对于 1 个 Dword 读取和 2 个 Dword 读取的特殊情况，字节使能可以不连续。字节使能在所有其它情况下均连续。长度为 0 的存储器读取在 CQ 接口上发送，其描述符中的“Dword count”（Dword 计数）字段设置为 1，且首字节使能和末字节使能均设为 0。

用户应用必须以完成包来响应每个存储器读取请求。由读取所请求的数据可作为单一完成包来发送，或者也可作为多个拆分完成包来发送。这些完成包必须通过集成块的完成器完成 (CC) 接口来发送。2 个不同请求的完成包可按任意顺序发送，但同一个请求的拆分完成包则必须按顺序发送。如需获取有关 CC 接口操作的说明，请参阅 [完成器完成接口操作](#)。

I/O 写入操作

在 CQ 接口上传输 I/O 写入请求的过程与含单 Dword 的有效载荷的存储器写入请求的传输过程相似。传输以 128 位描述符开始，紧随其后即为含单 Dword 的有效载荷。当使用 Dword 对齐模式时，有效载荷 Dword 紧接在描述符之后。当使用地址对齐模式时，有效载荷 Dword 在描述符后的新的一拍中提供，它在数据路径中的对齐方式则基于描述符中的地址来判定。`m_axis_cq_tuser` 中的首字节使能位用于指示有效载荷中的有效字节。字节使能位 `byte_en` 同样可提供此信息。

因为 I/O 写入为非转发传输事务，用户逻辑必须使用不含数据有效载荷的完成包来作为其响应。I/O 请求的完成包可按任意顺序发送。通过将完成器描述符中的“Completion Status”（完成状态）字段设置为相应的 CA（完成器异常中止）或 UR（请求不受支持）即可向请求器发出信号，表示发生与 I/O 写入传输事务关联的错误。如需获取有关完成器完成接口操作的说明，请参阅[完成器完成接口操作](#)。

I/O 读取操作

在 CQ 接口上传输 I/O 读取请求的过程与存储器读取请求的传输过程相似，且仅含描述符。请求的数据长度始终为 1 个 Dword，`m_axis_cq_tuser` 中的首字节使能位用于指示要读取的有效字节。

用户逻辑必须以单 Dword 完成包来响应 I/O 读取请求，或者如果发生错误，则以不含数据的完成包来响应。对应 2 个不同 I/O 读取请求的完成包可按任意顺序发送。通过将完成器描述符中的“Completion Status”（完成状态）字段设置为相应的 CA（完成器异常中止）或 UR（请求不受支持）即可向请求器发出信号，表示发生与 I/O 读取传输事务关联的错误。如需获取有关完成器完成接口操作的说明，请参阅[完成器完成接口操作](#)。

完成器请求接口上的原子操作

在完成器请求接口上传输原子操作请求的过程与存储器写入请求的传输过程相似。原子操作的有效载荷范围为 1 到 8 个 Dword，其起始地址始终与 Dword 边界对齐。传输以 128 位描述符开始，紧随其后即为有效载荷。当使用 Dword 对齐模式时，第一个有效载荷 Dword 紧接在描述符之后。当使用地址对齐模式时，有效载荷在描述符后的新的一拍中开始，其对齐方式则基于描述符中的地址来判定。`m_axis_cq_tkeep` 输出可指示有效载荷的结束位置。`m_axis_cq_tuser` 中的 `byte_en` 信号还可指示有效载荷中的有效字节。`m_axis_cq_tuser` 中的“首字节使能”和“末字节使能”不应用于原子操作。

因为原子操作为非转发传输事务，用户逻辑必须使用含操作结果的完成包来作为其响应。通过将完成器描述符中的“Completion Status”（完成状态）字段设置为相应的“Completer Abort (CA)”（完成器异常中止）或“Unsupported Request (UR)”（请求不受支持）即可向请求器发出信号，表示发生与此操作关联的错误。如需获取有关完成器完成接口操作的说明，请参阅[完成器完成接口操作](#)。

完成器请求接口上的报文请求

在 CQ 接口上传输报文的过程与存储器写入请求的传输过程类似，但有时其中可能不存在有效载荷。传输以 128 位描述符开始，其后即为有效载荷（如果存在）。当使用 Dword 对齐模式时，有效载荷紧接在描述符之后。当使用地址对齐模式时，有效载荷的第一个 Dword 在描述符后的新节拍内提供，并且始终从字节通道 0 开始。您可根据 `m_axis_cq_tlast` 信号和 `m_axis_cq_tkeep` 信号状态来判定有效载荷的结束位置。`m_axis_cq_tuser` 中的 `byte_en` 信号还可指示有效载荷中的有效字节。`m_axis_cq_tuser` 中的首字节使能位和末字节使能位不应用于报文传输事务。

传输异常中止

对于包含关联有效载荷的任何请求，集成块可以在传输的有效载荷中发出错误信号，方法是在数据包的最后一拍中将 `m_axis_cq_tuser` 总线中的 `discontinue` 信号随 `m_axis_cq_tlast` 一起断言有效。当集成块在读取其内部存储器中的数据时，如果检测到不可纠正的错误，则会发生此操作。当用户应用在数据包的最后一拍中检测到 `discontinue` 信号已断言有效时，必须丢弃整个数据包。此状况在集成块中也被视作为致命错误。

针对非转发请求的选择性流量控制

PCI Express® 基本规范要求完成器请求接口持续不断交付转发传输事务，即使用户应用无法接受非转发传输事务也是如此。为启用此功能，集成块会在 CQ 接口上实现基于信用值的流量控制机制，用户逻辑可通过此机制来控制非转发请求的流量，而不会影响转发请求。用户逻辑可使用 `pcie_cq_np_req[0]` 信号来标示用于接收非转发请求的缓冲器的可用情况。仅当可用信用值为非 0 值时，核才会交付非转发请求。当非转发请求交付因缺少信用值而暂停时，集成块会持续交付转发请求。当信用值机制对非转发请求的交付不施加任何反压时，集成块会按从链路接收到转发请求和非转发请求的顺序来交付这些请求。

集成块会保留内部信用值计数器，以跟踪完成器请求接口上可用于非转发请求的信用值。以下算法用于保持记录可用信用值：

- 在复位时，计数器设为 0。
- 当集成块解复位后，在每个时钟周期内：
 - 如果 `pcie_cq_np_req[0]` 为高电平有效状态并且在此周期内未交付任何非转发请求，那么信用值会递增 1，除非它已达到其饱和限值 12。
 - 如果 `pcie_cq_np_req[0]` 为低电平且在此周期内仅交付 1 个非转发请求，那么信用值递减 1，除非它已为 0。
 - 在所有其它情况下，信用值均保持不变。
- 仅当信用值大于 0 时，集成块才会开始交付非转发 TLP。

用户应用每次准备好接收非转发请求时，均可在 `pcie_cq_np_req[0]` 上提供 1 个周期的脉冲，或者如果不需要对非转发请求应用选择性反压，那么也可以将其永久保留为断言有效状态。如果信用值始终为非 0 值，那么集成块会按从链路接收到转发请求和非转发请求的顺序来交付这些请求。如果有时信用值保持为 0，那么非转发请求可在该集成块的 FIFO 中累积。当信用值后续变为非 0 值时，集成块会首先交付先于已交付的转发请求到达并累积的非转发请求，然后才会还原为按从链路接收到请求的顺序来交付这些请求。

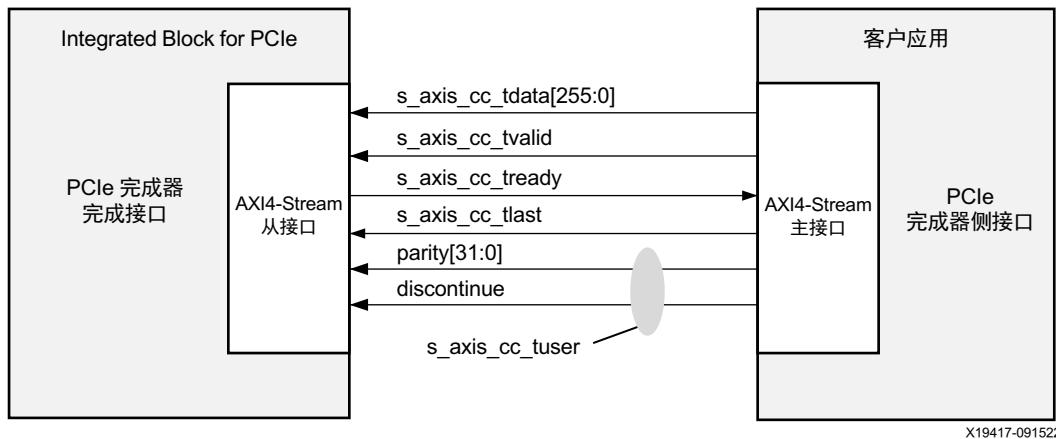
`pcie_cq_np_req[0]` 信号的断言有效和断言无效无需与完成器请求接口上的包传输对齐。

您可监控 `pcie_cq_np_req_count[5:0]` 输出上的当前信用值。计数器饱和上限为 32。由于存在内部流水线延迟，当集成块在 `pcie_cq_np_req[0]` 输入上接收到脉冲之后，可能需要经过多个延迟周期后，才会在响应中更新 `pcie_cq_np_req_count[5:0]` 输出。因此，当用户应用具有足够的缓冲器空间可用时，它应提前提供信用值，以便避免核应缺少信用值而延误交付非转发请求。

完成器完成接口操作

下图显示了与核的完成器完成接口关联的信号。核将此接口上的每个 TLP 均作为 1 个 AXI4-Stream 数据包来交付。

图 30：完成器完成接口信号



CC 接口支持 2 种不同的数据对齐模式。在 Dword 对齐模式下，有效数据的第 1 个字节必须存在于通道 $n = (12 + A \bmod 4) \bmod w$ 中，其中 A 是要传输的数据块的字节级别起始地址，如描述符的“Lower Address”（下位地址）字段中所述，而 w 则是接口宽度（以字节为单位，值为 8、16 或 32）。在地址对齐模式下，数据始终在描述符终止后的新的节拍中开始。为存储器或 I/O 读取请求传输完成包有效载荷时，其第 1 个有效字节位于通道 $n = A \bmod w$ 上。对于所有其它完成包，有效载荷与字节通道 0 对齐。

完成器完成描述符格式

用户应用将完成器请求的完成数据作为独立 AXI4-Stream 包发送至集成块的 CC 接口。每个数据包均以 1 个描述符开头，在描述符后可包含有效载荷数据。描述符长度始终为 12 字节，并在完成包的前 12 字节内发送。描述符在 64 位接口上的前 2 个节拍内进行传输，在 128 位或 256 位接口上的第 1 个节拍内进行传输。当用户应用将请求的完成数据拆分为多个拆分完成 (Split Completion) 包后，它必须将每个拆分完成包作为独立 AXI4-Stream 包随其描述符一起发送。

下图演示了完成器完成描述符的格式。下表描述了完成器请求描述符的每个字段。

图 31：完成器完成描述符格式

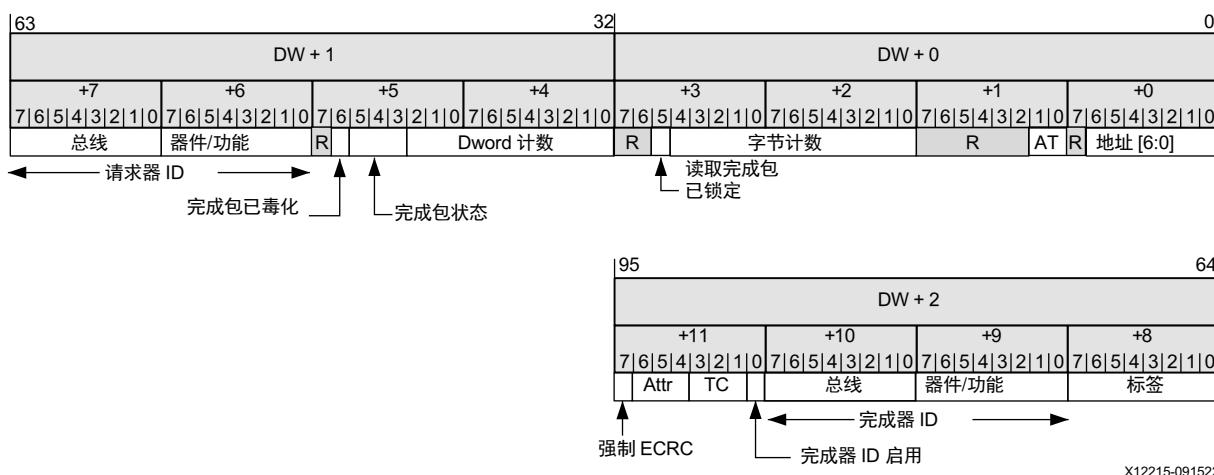


表 52：完成器完成描述符字段

位索引	字段名称	描述	
6:0	“Lower Address”（低位地址）	对于存储器读取完成包，该字段必须设置为所传输的存储器块的起始字节级地址的 7 个最低有效位。对于首个完成包或唯一完成包，完成器生成该字段的方式为：将请求地址的 5 个最低有效位与字节使能针对请求的首个 Dword 所形成的 2 位字节级地址串联来生成该字段。	
		first_be[3:0]	Lower Address[1:0]
		4'b0000	2'b00
		4'bxxx1	2'b00
		4'bxx10	2'b01
		4'bx100	2'b10
		4'b1000	2'b11
9:8	“Address Type”（地址类型）	对于任何后续“Completion”（完成）包，“Lower Address”（下位地址）字段始终为 0，但针对所含读取完成边界（RCB）值为 64 字节的根联合体所生成的“Completion”包除外。在此情况下，“Lower Address”字段的 6 个最低有效位始终为 0，且“Lower Address”字段的最高有效位将根据 64 位数据有效载荷的对齐方式进行切换。	
		对于所有其它完成包，下位地址必须全部设置为 0。	
28:16	“Byte Count”（字节计数）	<p>该 13 位可包含范围在 0 - 4,096 个字节内的值。如果存储器读取请求已通过使用单一完成包完成，那么“Byte Count”值将以字节为单位来表示“Payload”（有效载荷）大小。针对 I/O 读取完成包和 I/O 写入完成包，该字段必须设置为 4。针对长度为 0 的存储器读取发送完成包时，字节计数必须设置为 1，并且在描述符之后必须附加含单 Dword 的虚拟有效载荷。</p> <p>对于每个“Memory Read Completion”（存储器读取完成），“Byte Count”字段必须表明完成请求所需的剩余字节数，包括随完成包返回的字节数。</p> <p>如果存储器读取请求已使用多个完成包完成，那么后续每个完成包的字节计数值均表示为前一个完成包减去随前一个完成包返回的字节数。下表显示了完成存储器读取请求所需的总字节数。</p> <p>“Byte Count”字段的 MSB 为保留值。</p>	
29	“Locked Read Completion”（锁定读取完成）	当“Locked Read”（锁定读取）请求的响应中包含完成包时，必须设置该位。针对所有其它完成包，该位必须设置为 0。	
42:32	“Dword Count”（Dword 计数）	这 11 位表示当前包的有效载荷大小（以 Dword 数为单位）。其范围是 0 - 1000 个 Dword。针对 I/O 读取完成包，该字段必须设置为 1，针对 I/O 写入完成包，该字段必须设置为 0。针对长度为 0 的存储器读取发送完成包时，Dword 计数必须设置为 1。发送 UR 或 CA 完成时，Dword 计数必须设置为 0。在所有其它情况下，Dword 计数必须对应于当前包的有效载荷中 Dword 的实际数量。	
45:43	“Completion Status”（完成状态）	这些位必须基于所发送的完成（Completion）包的类型来设置。有效设置仅包括： <ul style="list-style-type: none"> · 000：成功完成 · 001：请求不受支持（UR） · 100：完成器异常中止（CA） 	
46	“Completion Status”（完成状态）	此位可用于对所发送的完成 TLP 进行毒化。针对所有完成包，此位必须设置为 0，除非用户应用在位于描述符后的数据块中检测到错误，并且想使用 PCI Express 的“Data Poisoning”（数据毒化）功能来传递此信息。	
63:48	“Requester ID”（请求器 ID）	与请求关联的 PCI 请求器 ID（从请求复制所得）。	
71:64	“Tag”（标签）	与请求关联的 PCI Express 标签（从请求复制所得）。	

表 52：完成器完成描述符字段 (续)

位索引	字段名称	描述
79:72	“Target Function/ Device Number” (目标功能/ 器件编号)	<p>完成器功能的器件和/或功能编号。</p> <p>端点模式：</p> <p>ARI 已启用：</p> <ul style="list-style-type: none">· 位 [79:72] 必须设置为完成器功能编号。 <p>ARI 已禁用：</p> <ul style="list-style-type: none">· 位 [74:72] 必须设置为完成器功能编号。· 不使用位 [79:75] <p>开关的上游端口用例 (在 IP 中选中端点模式) :</p> <p>ARI 已启用：</p> <ul style="list-style-type: none">· 位 [79:72] 必须设置为完成器功能编号。 <p>ARI 已禁用：</p> <ul style="list-style-type: none">· 位 [74:72] 必须设置为完成器功能编号。· 如果完成包源自开关本身，则不使用位 [79:75]。如果开关正在中继完成包（完成器位于开关外部），那么这些位必须设置为完成器器件编号，此编号对应于该完成包的来源位置。该位可与描述符中的“Completer ID Enable”位配合使用。 <p>根端口模式 (下游端口) :</p> <p>ARI 已启用：</p> <ul style="list-style-type: none">· 位 [79:72] 必须设置为完成器功能编号。 <p>ARI 已禁用：</p> <ul style="list-style-type: none">· 位 [74:72] 必须设置为完成器功能编号。· 位 [79:75] 必须设置为完成器器件编号。该位可与描述符中的“Completer ID Enable”位配合使用。
87:80	“Completer Bus Number” (完成器总线编号)	<p>与完成器功能关联的总线编号。</p> <p>端点模式：</p> <ul style="list-style-type: none">· 不使用 <p>开关的上游端口用例 (在 IP 中选中端点模式) :</p> <ul style="list-style-type: none">· 如果完成包源自开关本身，则不使用这些位。如果开关正在中继完成包（完成器位于开关外部），那么这些位必须设置为完成器总线编号，此编号对应于该完成包的来源位置。该位可与描述符中的“Completer ID Enable”位配合使用。 <p>根端口模式 (下游端口) :</p> <ul style="list-style-type: none">· 必须设置为完成器总线编号。该位可与描述符中的“Completer ID Enable”位配合使用。

表 52：完成器完成描述符字段 (续)

位索引	字段名称	描述
88	“Completer ID Enable” (完成器 ID 使能)	<p>值包括：</p> <ul style="list-style-type: none"> · 1'b1：客户在描述符中提供总线编号、器件编号和功能编号，以供填充到 TLP 报头中的“Completer ID”（完成器 ID）字段中。 · 1'b0：IP 使用从接收到的配置请求中捕获的总线和器件编号，客户在描述符中提供功能编号，以供填充到 TLP 报头中的“完成器 ID”字段中。 <p>端点模式：</p> <ul style="list-style-type: none"> · 必须设置为 1'b0。 <p>开关的上游端口用例（在 IP 中选中端点模式）：</p> <ul style="list-style-type: none"> · 如果完成包源自开关本身，则设置为 1'b0。 · 当开关正在中继完成（完成器位于开关外部）时，则设置为 1'b1。不启用 ARI 时，该位可与完成器总线编号位 [95:88] 和完成器功能/器件编号位 [87:83] 配合使用。 <p>根端口模式：</p> <ul style="list-style-type: none"> · 必须设置为 1'b1。不启用 ARI 时，该位可与完成器总线编号位 [95:88] 和完成器功能/器件编号位 [87:83] 配合使用。
91:89	“Transaction Class (TC)” (传输事务类)	与请求关联的 PCIe 传输事务类 (TC)。用户应用必须从关联的请求描述符的 TC 字段复制该值。
94:92	属性	与请求关联的 PCIe 属性（从请求复制所得）。位 92 为“No Snoop”（无嗅探）位，位 93 为“Relaxed Ordering”（宽松排序）位，位 94 为“ID-Based Ordering”（基于 ID 排序）位。
95	“Force ECRC” (强制 ECRC)	强制执行 ECRC 插入。将该位设置为 1 会强制集成块将包含 ECRC 的 TLP 摘要追加到完成 TLP，即使针对发送完成包的功能未启用 ECRC 也是如此。

表 53：根据完成器请求 first_be[3:0]、last_be[3:0]、Dword Count[10:0] 计算字节计数

first_be[3:0]	last_be[3:0]	字节计数总数
1xx1	0000	4
01x1	0000	3
1x10	0000	3
0011	0000	2
0110	0000	2
1100	0000	2
0001	0000	1
0010	0000	1
0100	0000	1
1000	0000	1
0000	0000	1
xxx1	1xxx	Dword_count × 4
xxx1	01xx	(Dword_count × 4)-1
xxx1	001x	(Dword_count × 4)-2
xxx1	0001	(Dword_count × 4)-3
xx10	1xxx	(Dword_count × 4)-1
xx10	01xx	(Dword_count × 4)-2

表 53：根据完成器请求 first_be[3:0]、last_be[3:0]、Dword Count[10:0] 计算字节计数 (续)

first_be[3:0]	last_be[3:0]	字节计数总数
xx10	001x	(Dword_count × 4)-3
xx10	0001	(Dword_count × 4)-4
x100	1xxx	(Dword_count × 4)-2
x100	01xx	(Dword_count × 4)-3
x100	001x	(Dword_count × 4)-4
x100	0001	(Dword_count × 4)-5
1000	1xxx	(Dword_count × 4)-3
1000	01xx	(Dword_count × 4)-4
1000	001x	(Dword_count × 4)-5
1000	0001	(Dword_count × 4)-6

含成功完成状态的完成包

每次从完成器请求接口接收到 1 个非转发请求时，用户应用都必须向核的 CC 接口返回 1 个完成 (Completion) 包。当请求完成且不含任何错误时，用户应用必须返回含成功完成 (SC) 状态的完成包。根据请求类型，此完成包可能包含有效载荷，也可能不含有效载荷。此外，当数据块大小超出配置的有效载荷最大大小时，与请求关联的数据可拆分为多个拆分完成 (Split Completion) 包。用户逻辑负责根据需要将数据块拆分为多个拆分完成包。用户应用必须将每个拆分完成包作为独立 AXI4-Stream 数据包（包含其自己的 12 字节描述符）通过完成器完成接口进行传输。

在本章的时序图示例中，要传输的数据块的起始 Dword 地址（如描述符的下位地址字段的位 [6:2] 中所述）假定为 $(m \times 8 + 1)$ ，其中 m 为整数。数据块的大小假定为 n 个 Dword，其中， $n = k \times 32 + 28$ 且 $k > 0$ 。

CC 接口支持 2 种数据对齐模式：Dword 对齐和地址对齐。以下时序图显示了当接口宽度分别配置为 64 位、128 位和 256 位时，从用户应用接收到完成包（采用 Dword 对齐模式）后，通过 CC 接口来进行传输的过程。在此情况下，有效载荷的第一个 Dword 紧接在描述符之后开始。当数据块并非 4 字节的倍数时，或者当有效载荷的开始位置与 Dword 地址边界未对齐时，用户应用必须添加空字节以使有效载荷对齐到 Dword 边界，并使有效载荷成为 Dword 数量的整数倍。例如，当数据块从字节地址 7 开始且大小为 3 字节时，用户应用必须在第 1 个字节前添加 3 个空字节，并在数据块末尾添加 2 个空字节，以使其长度达到 2 个 Dword。并且，对于非连续读取，数据块中返回的字节并非全都有效。在此情况下，用户应用必须在适当位置返回有效字节，并根据需要在有效字节间的间隙内填充空字节。此接口不提供任何信号用于指示有效载荷中的有效字节。此类信号并非必需，原因在于请求器会负责保留请求中字节使能的记录，并丢弃来自完成包的无效字节。

在 Dword 对齐模式下，传输从 12 个描述符字节开始，紧随其后即为有效载荷字节。用户应用必须在数据包的整个持续时间段保持 s_axis_cc_tvalid 信号处于断言有效状态。集成块会将数据包传输期间断言无效的 s_axis_cc_tvalid 作为错误来处理，并将链路上发射的对应完成 TLP 置空，以避免数据损坏。

用户应用还必须在数据包的最后一拍中断言 s_axis_cc_tlast 信号有效。如果集成块未准备好接受数据，那么可在任意周期内断言 s_axis_cc_tready 无效。在集成块含 deasserted s_axis_cc_tready 的时钟周期内，用户应用不得更改 CC 接口上的值。

图 32：完成器完成接口上的正常完成包的传输（Dword 对齐模式、64 位接口）

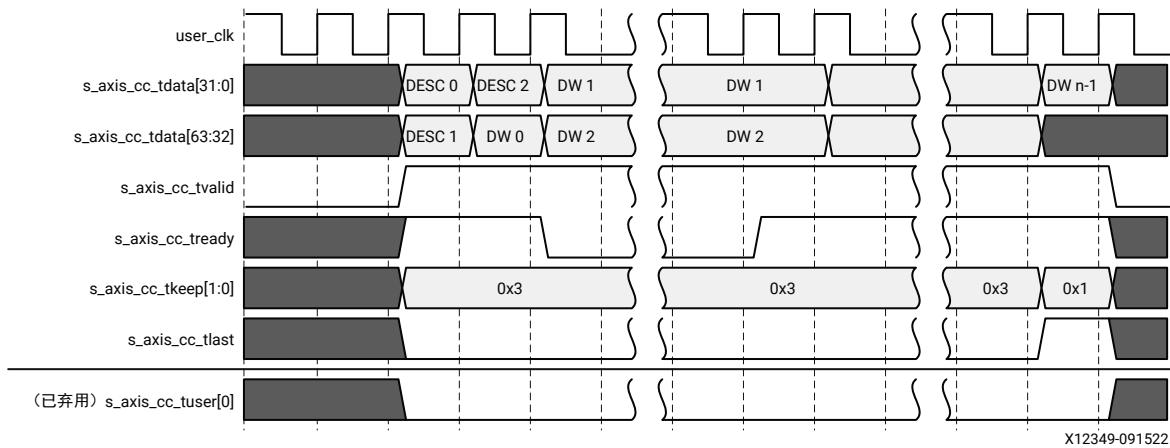


图 33：完成器完成接口上的正常完成包的传输（Dword 对齐模式、128 位接口）

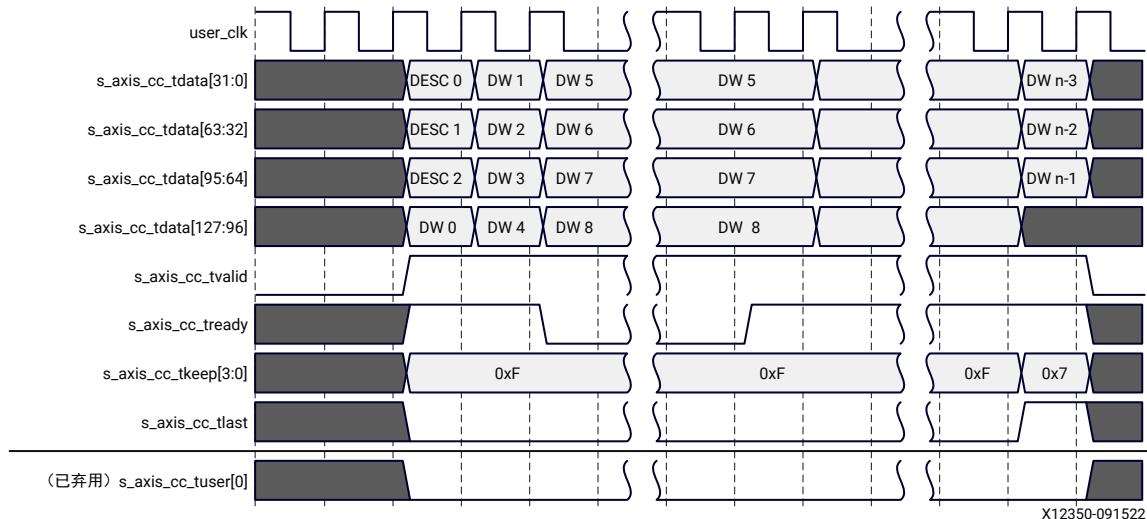
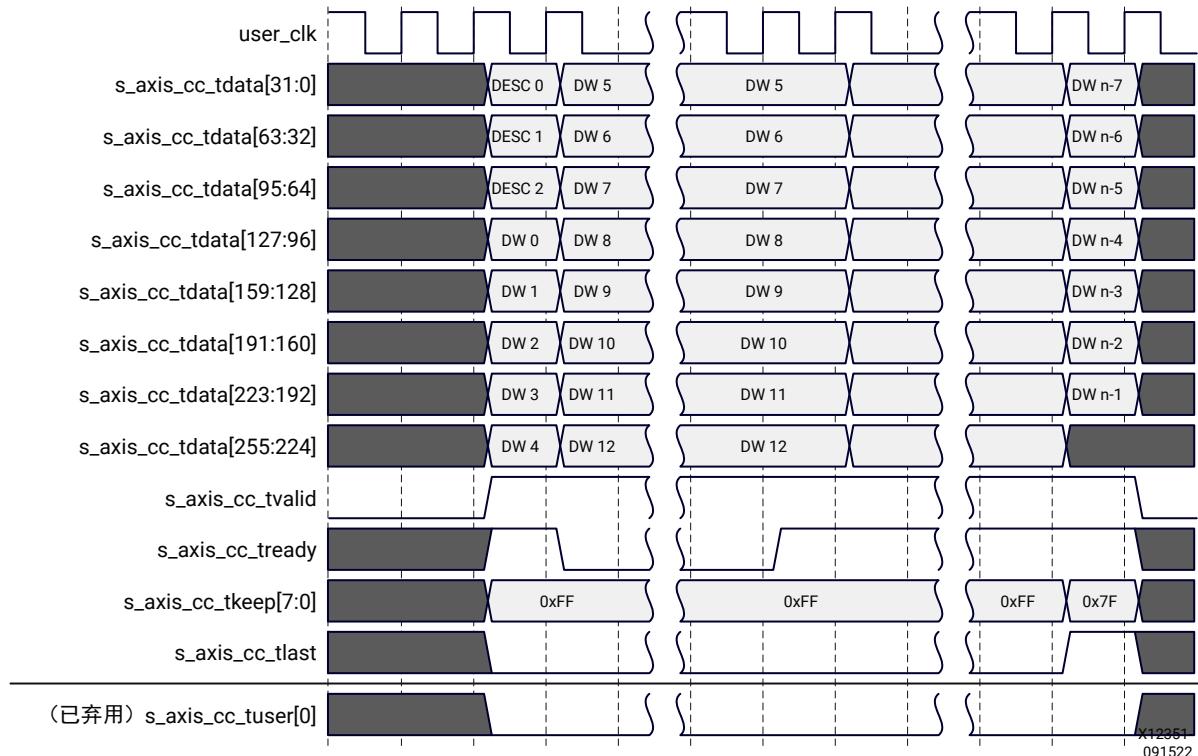


图 34：完成器完成接口上的正常完成包的传输（Dword 对齐模式、256 位接口）



在地址对齐模式下，有效载荷的交付始终从描述符的最后一个字节后的节拍中开始。对于存储器读取完成包，根据有效载荷的第 1 个有效字节的地址，有效载荷的第 1 个字节可显示在任意字节通道上。对于所有其它完成包，有效载荷必须从字节通道 0 开始。

以下时序图显示了当接口宽度分别配置为 64 位、128 位和 256 位时，通过完成器完成接口来传输存储器读取完成包（采用地址对齐模式）的过程。为便于演示，要传输的数据块的起始 Dword 地址（如描述符的下位地址字段的位 [6:2] 中所述）假定为 $(m \times 8 + 1)$ ，其中 m 为整数。数据块大小假定为 n 个 Dword，其中， $n = k \times 32 + 28$ 且 $k > 0$ 。

图 35：完成器完成接口上的正常完成包的传输（地址对齐模式、64 位接口）

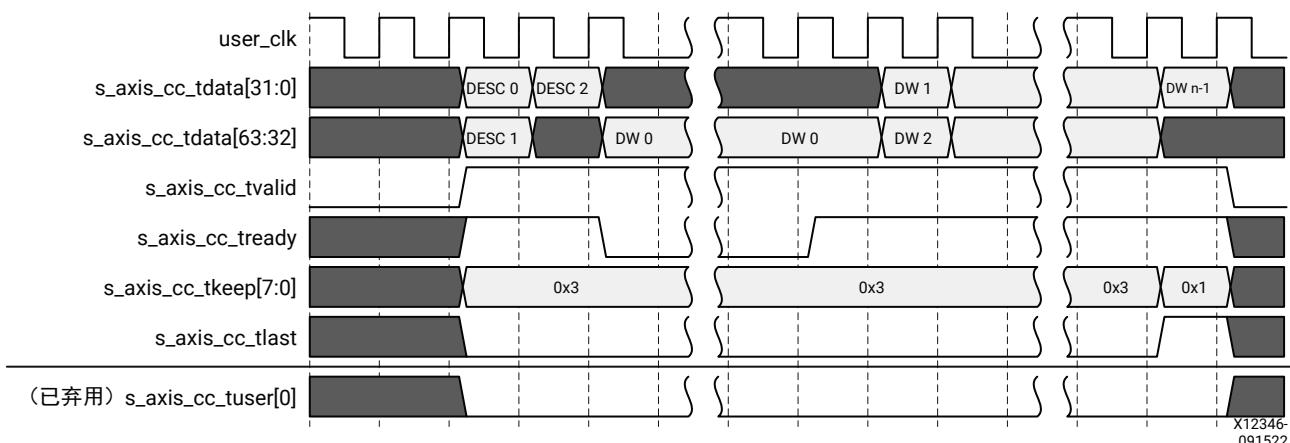


图 36：完成器完成接口上的正常完成包的传输（地址对齐模式、128 位接口）

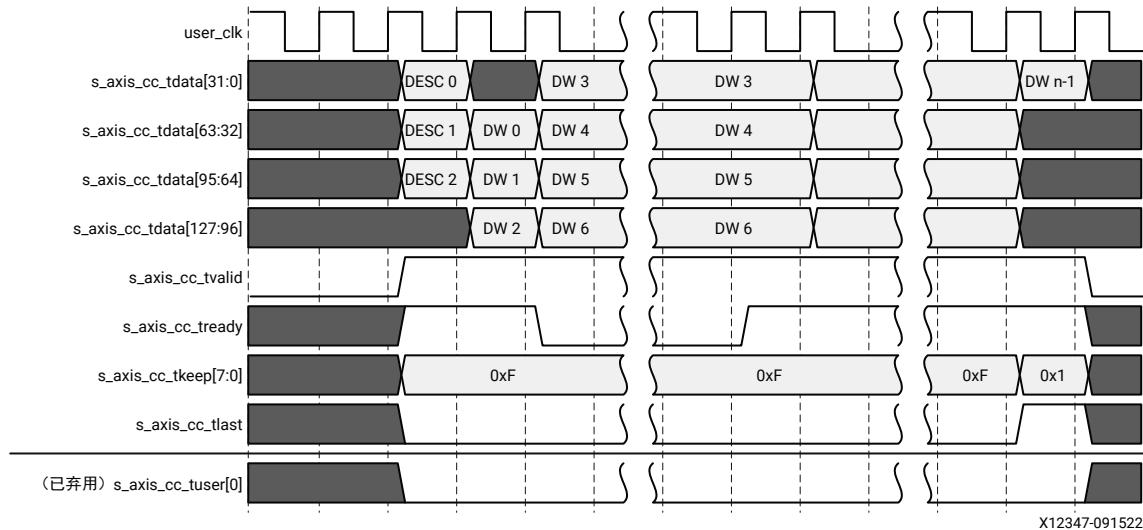
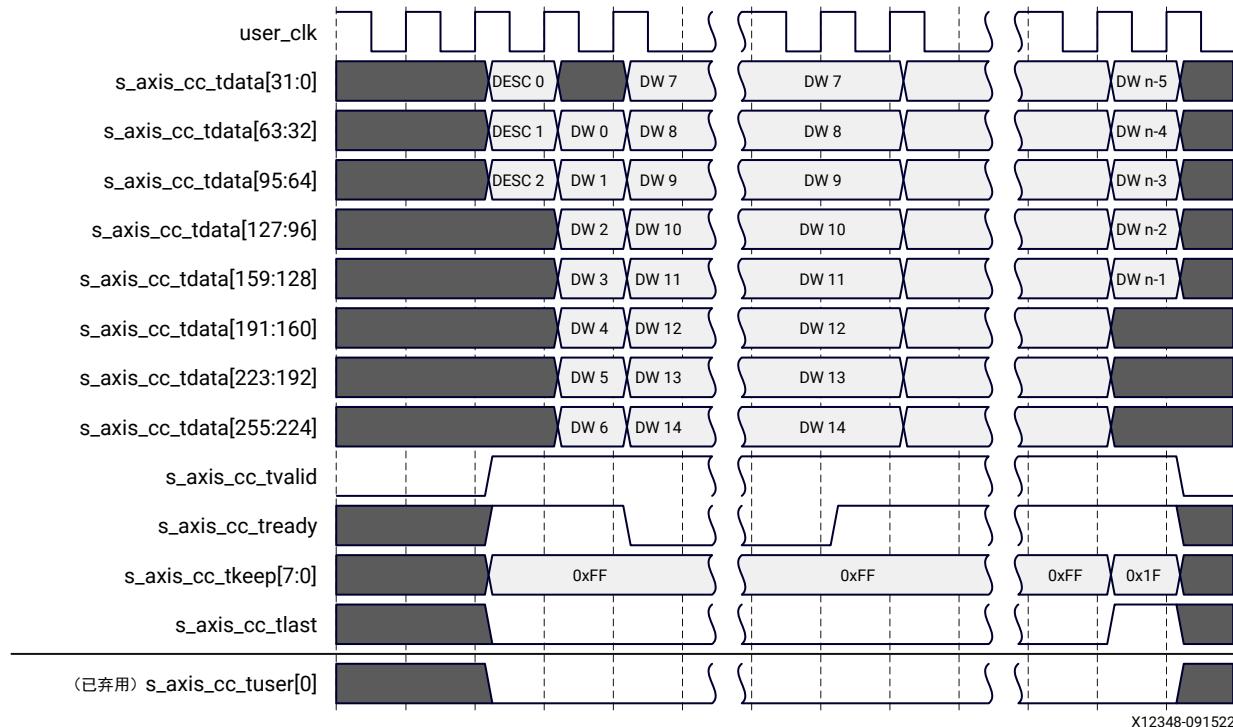


图 37：完成器完成接口上的正常完成包的传输（地址对齐模式、256 位接口）



完成包传输异常中止

用户应用可在传输有效载荷期间随时通过断言 **s_axis_cc_tuser** 总线中的 **discontinue** 信号有效来异常中止完成器完成接口上的完成传输事务的传输。集成块会将链路上对应 TLP 置空，以避免数据损坏。

在传输期间，当传输的完成包具有关联有效载荷时，用户应用可以在任意周期内断言此信号有效。用户应用可以选择在周期内发出错误信号处提前终止该数据包（通过断言 `s_axis_cc_tlast` 有效），或者也可以继续处理，直至将有效载荷的所有字节都交付到集成块为止。针对后者，该集成块会针对数据包的后续节拍将此错误作为粘滞错误来处理，即使用户应用在达到包结束前断言 `discontinue` 信号无效也是如此。

仅当 `s_axis_cc_tvalid` 为高电平有效时，才能断言 `discontinue` 信号有效。当 `s_axis_cc_tvalid` 和 `s_axis_cc_tready` 均断言有效时，集成块会对此信号进行采样。因此，断言有效后，在断言 `s_axis_cc_tready` 有效前不应断言 `discontinue` 信号无效。

当集成块配置为端点时，集成块会使用高级错误报告 (AER) 机制向所连接到的根联合体报告此错误（作为不可纠正的内部错误）。

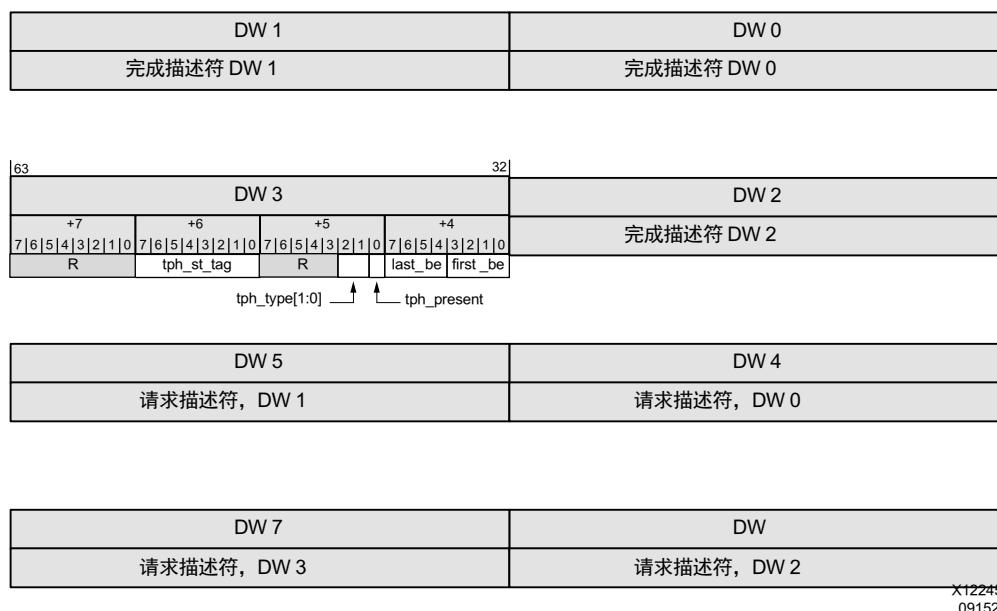
含错误状态的完成包 (UR 和 CA)

如果完成器请求接口上接收到的请求的状态为请求不受支持 (UR) 或完成器异常中止 (CA)，那么响应此类请求时，用户应用必须发送含 3 个 Dword 的完成描述符（格式如 [完成器完成描述符格式](#) 中的完成器完成描述符格式图中所示），后接另 5 个 Dword（其中包含有关生成完成包的请求的信息）。集成块需使用这 5 个 Dword 在其 AER 报头 Log 日志寄存器中记录有关该请求的信息。

下图显示了发送含 UR 或 CA 状态的完成包时的信息传输顺序。这些信息格式化为 AXI4-Stream 数据包，其中包含总计 8 个 Dword，组织方式如下：

- 前 3 个 Dword 包含完成描述符，格式如 [完成器完成描述符格式](#) 中的完成器完成描述符格式图中所示。
- 第 4 个 Dword 包含 `m_axis_cq_tuser` 中的下列信号的状态（复制自请求）：
 - `m_axis_cq_tuser` 中的首字节使能位 `first_be[3:0]`。
 - `m_axis_cq_tuser` 中的末字节使能位 `last_be[3:0]`。
 - 承载传输事务处理提示相关信息的信号：`m_axis_cq_tuser` 中的 `tph_present`、`tph_type[1:0]` 和 `tph_st_tag[7:0]`。

图 38：含 UR 和 CA 完成包的 AXI4-Stream 数据包的组成方式



整个数据包在 64 位接口上含 4 个节拍，在 128 位接口上含 2 个节拍，在 256 位接口上则含 1 个节拍。此数据包在 Dword 对齐模式和地址对齐模式下按相同顺序进行传输，其中所有 Dword 封装在一起。用户应用必须在数据包的整个持续时间段保持 `s_axis_cc_tvalid` 信号处于断言有效状态。它还必须在数据包的最后一拍中断言 `s_axis_cc_tlast` 信号有效。如果集成块未准备好接受数据，那么可在任意周期内断言 `s_axis_cc_tready` 无效。在集成块已断言 `s_axis_cc_tready` 无效的任意周期内，用户应用都不得更改 CC 接口上的值。

64/128/256 位请求器接口

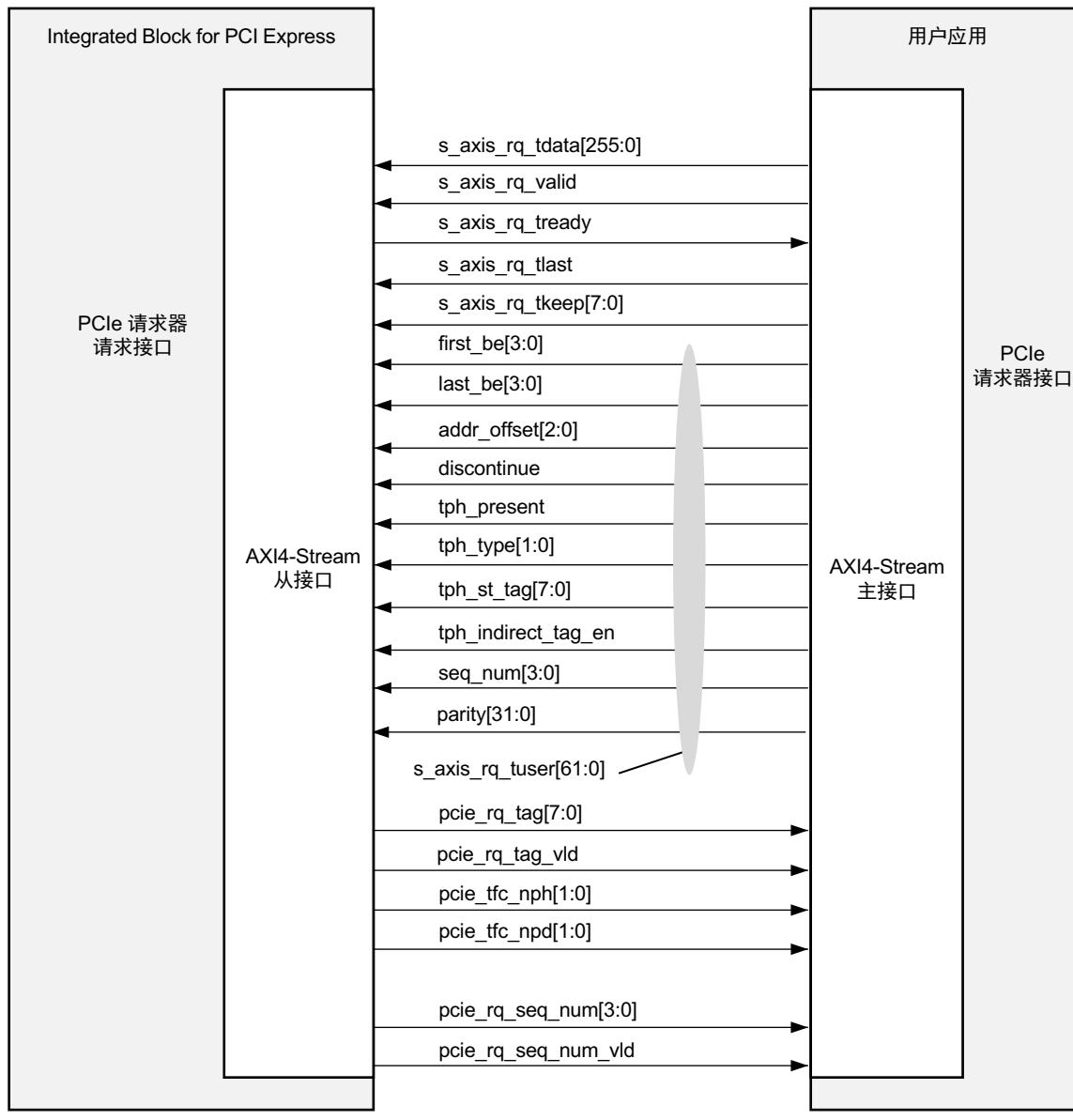
请求器接口支持用户端点应用发起 PCIe 传输事务，并将其作为总线主控制器以跨 PCIe 链路传输至主机存储器。对于根联合体，此接口还用于发起 I/O 和配置请求。此接口还可供端点和根联合体用于在 PCIe® 链路上发送报文。此接口上的传输事务类似于完成器接口上的传输事务，但核及用户应用的角色发生了对换。转发传输事务作为单一不可分割操作来执行，非转发传输事务则作为拆分传输事务来执行。

请求器接口包含 2 个独立接口，对应每个方向的数据传输使用 1 个接口。每个接口都基于 AXI4-Stream 协议，其宽度可配置为 64、128 或 256 位。请求器请求 (RQ) 接口用于将请求（含任何关联的有效载荷数据）从用户应用传输到集成块，而请求器完成 (RC) 接口则供集成块用于将从链路接收到的完成包（针对非转发请求）交付到用户应用。这 2 个接口均单独运行。即，用户应用可通过 RQ 接口传输新请求，同时接收前一个请求的完成包。

请求器请求接口操作

在 RQ 接口上，用户应用将每个 TLP 都作为 1 个 AXI4-Stream 数据包来进行交付。对于含有效载荷的 TLP，此数据包以 128 位描述符开头并后接数据。下图显示了与请求器请求接口关联的信号。

图 39：请求器请求接口



X22926-091522

RQ 接口支持通过 2 种不同数据对齐模式来传输有效载荷。在 Dword 对齐模式下，用户逻辑必须紧接在描述符的最后一个 Dword 之后提供有效载荷的第一个 Dword。并且，还必须在 `first_be[3:0]` 中设置相应的位以指示第一个 Dword 中的有效字节，并在 `last_be[3:0]` 中设置相应的位（两者均为 `s_axis_rq_tuser` 总线的一部分）以指示有效载荷的最后一个 Dword 中的有效字节。在地址对齐模式下，用户应用必须在描述符的最后一个 Dword 之后的节拍中开始有效载荷的传输，其第一个 Dword 可位于数据路径上的任意可能的 Dword 位置内。用户应用使用 `s_axis_rq_tuser` 中的 `addr_offset[2:0]` 信号来传达数据路径上的第一个 Dword 的偏移。采用 Dword 对齐模式的情况下，用户应用还必须在 `first_be[3:0]` 中设置相应的位以指示第一个 Dword 中的有效字节，并在 `last_be[3:0]` 中设置相应的位以指示有效载荷的最后一个 Dword 中的有效字节。

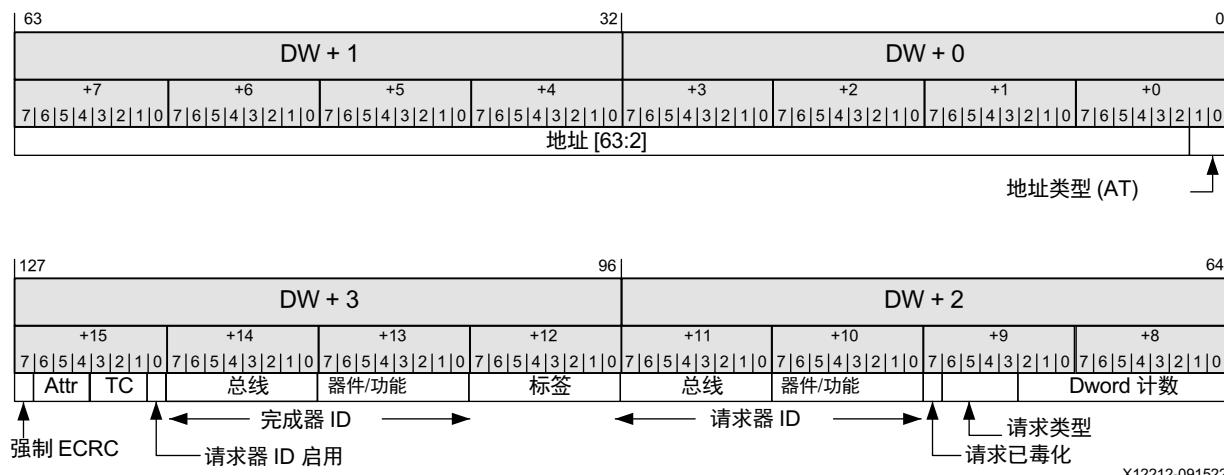
在核中启用“Transaction Processing Hint Capability”（传输事务处理提示功能）的情况下，用户应用可以通过使用 s_axis_rq_tuser 总线中包含的 `tph_*` 信号来为任何存储器传输事务提供可选提示。要随请求提供提示，用户逻辑必须在数据包的第一拍内断言 `tph_present` 有效，并分别在 `tph_st_tag[7:0]` 和 `tph_st_type[1:0]` 上提供“TPH Steering Tag”（TPH 导向标签）和“Steering Tag Type”（导向标签类型）。除了提供要使用的导向标签值外，用户应用还具有提供间接导向标签的选项。具体方法是在断言 `tph_present` 有效时，将 `tph_indirect_tag_en` 信号设置为 1，并在 `tph_st_tag[7:0]` 上放置索引以代替标签值。随后，集成块会读取其“Steering Tag Table”（导向标签表）中存储的标签，并将其插入请求 TLP，此标签与位于索引中指定的偏移处的请求器功能相关联。

请求器请求描述符格式

用户应用必须将链路上要发射的每个请求都作为独立 AXI4-Stream 包传输到集成块的 RQ 接口上。每个数据包都必须以 1 个描述符开头，在描述符后可包含有效载荷数据。描述符长度始终为 16 字节，并必须在请求包的前 16 字节内发送。描述符在 64 位接口上的前 2 个节拍内进行传输，在 128 位或 256 位接口上的第 1 个节拍内进行传输。下图演示了不同类型的请求的描述符格式。

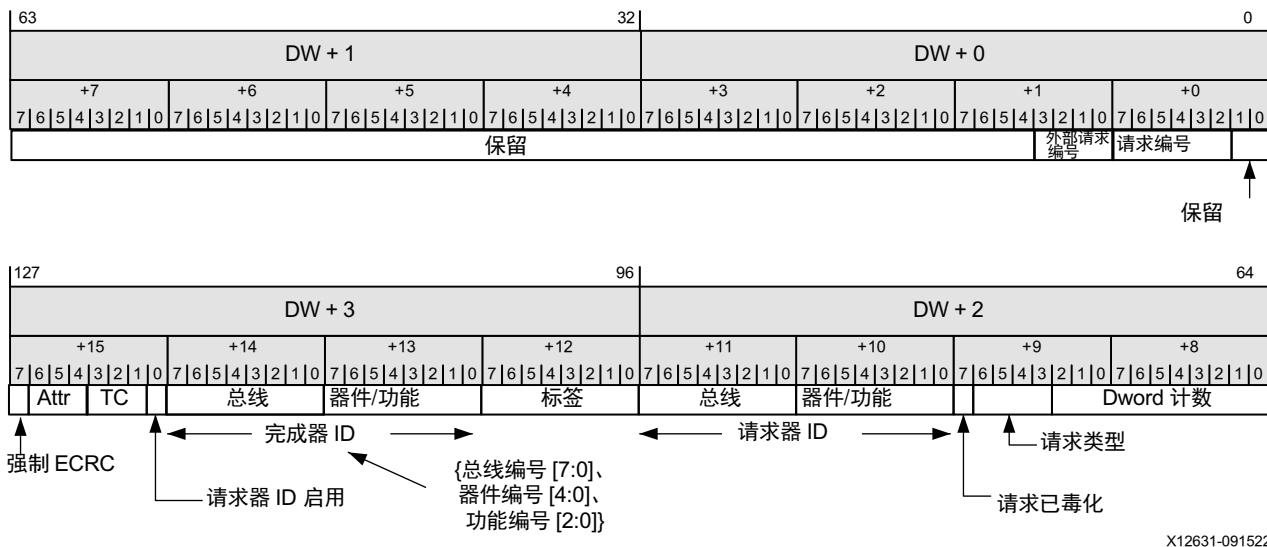
当所传输的请求 TLP 为存储器读取/写入请求、I/O 读取/写入请求或原子操作 (Atomic Operation) 请求时，即适用下图所示格式。

图 40：对应存储器、I/O 和原子操作请求的请求器请求描述符格式



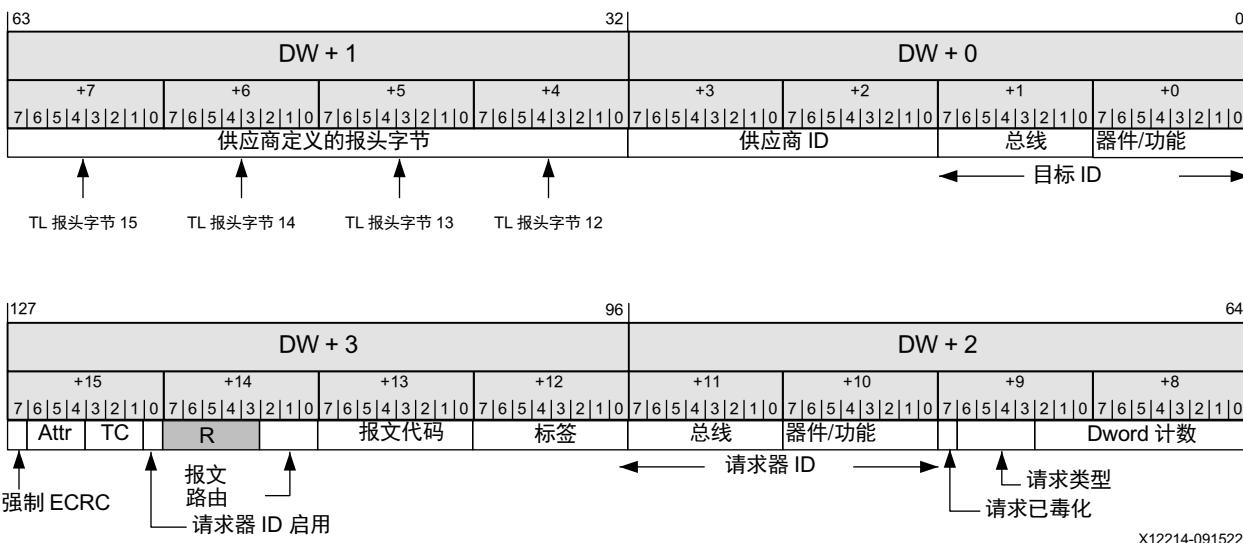
下图所示格式用于“Configuration Request”（配置请求）。

图 41：对应配置请求的请求器请求描述符格式



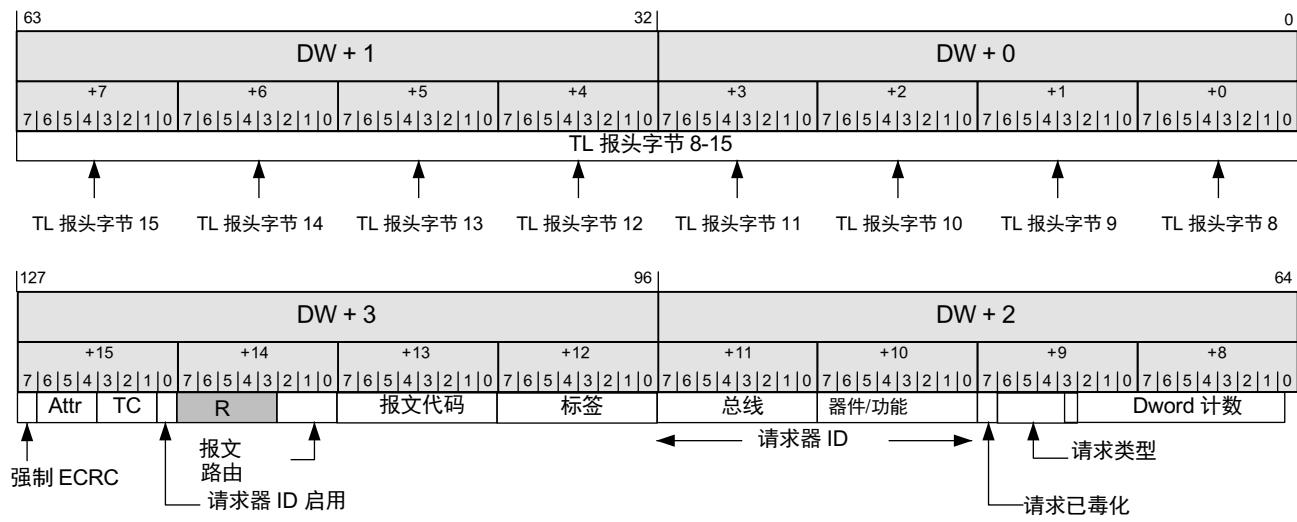
下图所示格式仅用于类型 0 或类型 1 的“Vendor-Defined Messages”（供应商定义的报文）。

图 42：对应供应商定义的报文的请求器请求描述符格式



下图所示格式适用于所有 ATS 报文，包括“Invalid Request”（无效请求）、“Invalid Completion”（无效完成）、“Page Request”（页面请求）、“PRG Response”（PRG 响应）。

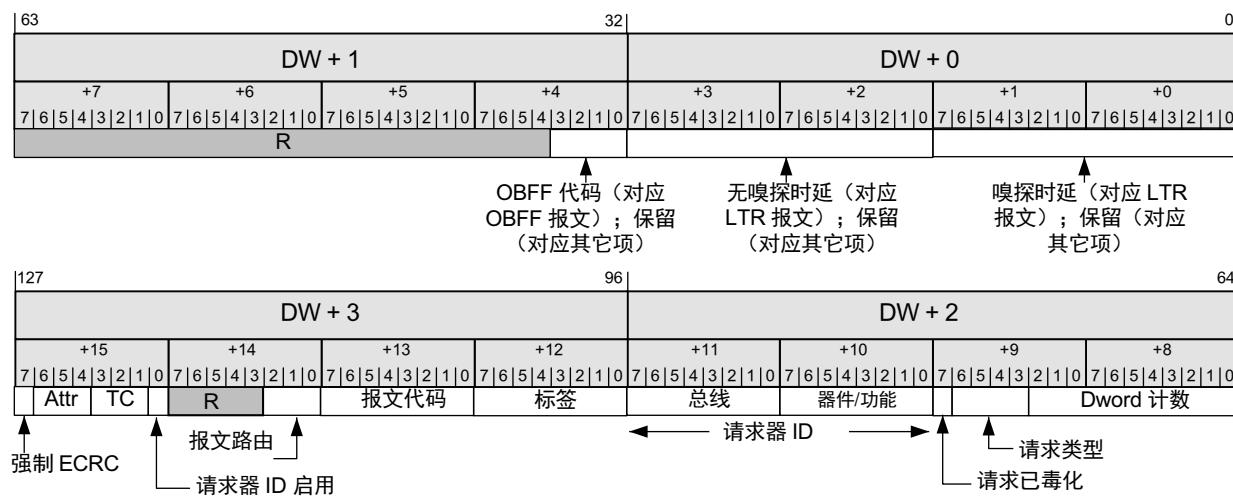
图 43：对应于 ATS 报文的请求器请求描述符



X12211-091522

对于所有其它报文，描述符采用下图所示格式。

图 44：对应于所有其它报文的请求器请求描述符格式



X12213-091522

表 54：请求器请求描述符字段

位索引	字段名称	描述
1:0	“Address Type” (地址类型)	<p>该字段定义仅适用于存储器传输事务和原子操作。集成块会将此字段复制到请求 TLP 的 TL 报头的 AT 中。</p> <ul style="list-style-type: none"> · 00: 请求中的地址未经转换 · 01: 传输事务为转换请求 · 10: 请求中的地址为已转换的地址 · 11: 保留

表 54：请求器请求描述符字段 (续)

位索引	字段名称	描述
63:2	“Address” (地址)	该字段适用于存储器、I/O 和原子操作请求。这是请求所引用的首个 Dword 的地址。用户应用还必须在 s_axis_rq_tuser 中设置 First_BE 位和 Last_BE 位，这两个位分别用于指示首个 Dword 和最后一个 Dword 中的有效字节。 当传输事务指定 32 位地址时，该字段的位 [63:32] 必须设置为 0。
74:64	“Dword Count” (Dword 计数)	这 11 个位用于指示要读取或写入的块大小（以 Dword 为单位），对于报文，这些位指示报文有效载荷的大小。存储器写入请求的有效范围为 0-256 个 Dword。存储器读取请求的有效范围为 1-1024 个 Dword。对于 I/O 访问，Dword 计数始终为 1。 对于长度为 0 的存储器读取/写入请求，Dword 计数必须为 1，且 First_BE 位全部设置为 0。 集成块不会对照提供的有效载荷（针对含有效载荷的请求）的实际长度检查该字段的设置，也不会对照集成块的最大有效载荷大小或读取请求大小设置检查此项设置。
78:75	“Request Type” (请求类型)	用于识别传输事务类型。 表 51：传输事务类型 中列出了传输事务类型及其编码。
79	“Poisoned Request” (毒化请求)	此位可用于对所发送的请求 TLP 进行毒化。在除类型 0 和类型 1 配置写入请求外的所有请求类型上都支持此功能。针对所有请求，此位必须设置为 0，除非用户应用在位于描述符后的数据块中检测到错误，并且想使用 PCI Express 的“Data Poisoning”（数据毒化）功能来传递此信息。 在除类型 0 和类型 1 配置写入请求外的所有请求类型上都支持此功能。
87:80	“Requester Function/Device Number” (请求器功能/器件编号)	请求器功能的器件和/或功能编号。 端点模式： <ul style="list-style-type: none">· ARI 已启用：<ul style="list-style-type: none">◦ 位 [87:80] 必须设置为请求器功能编号。· ARI 已禁用：<ul style="list-style-type: none">◦ 位 [82:80] 必须设置为请求器功能编号。◦ 不使用位 [87:83] 开关的上游端口用例（在 IP 中选中端点模式）： <ul style="list-style-type: none">· ARI 已启用：<ul style="list-style-type: none">◦ 位 [87:80] 必须设置为请求器功能编号。· ARI 已禁用：<ul style="list-style-type: none">◦ 位 [82:80] 必须设置为请求器功能编号。◦ 如果请求源自开关本身，则不使用位 [87:83]。如果开关正在中继请求（请求器位于开关外部），那么这些位必须设置为请求器器件编号，此编号对应于该请求的来源位置。该位可与描述符中的“Requester ID Enable”位配合使用。 根端口模式（下游端口）： <ul style="list-style-type: none">· ARI 已启用：<ul style="list-style-type: none">◦ 位 [87:80] 必须设置为请求器功能编号。· ARI 已禁用：<ul style="list-style-type: none">◦ 位 [87:80] 必须设置为请求器功能编号。◦ 位 [87:83] 必须设置为请求器器件编号。该位可与描述符中的“Requester ID Enable”位配合使用。

表 54：请求器请求描述符字段 (续)

位索引	字段名称	描述
95:88	“Requester Bus Number” (请求器总线编号)	<p>与请求器功能关联的总线编号。</p> <p>端点模式：</p> <ul style="list-style-type: none">不使用 <p>开关的上游端口用例（在 IP 中选中端点模式）：</p> <ul style="list-style-type: none">如果请求源自开关本身，则不使用这些位。如果开关正在中继请求（请求器位于开关外部），那么这些位必须设置为请求器总线编号，此编号对应于该请求的来源位置。该位可与描述符中的“Requester ID Enable”位配合使用。 <p>根端口模式（下游端口）：</p> <ul style="list-style-type: none">必须设置为请求器总线编号。该位可与描述符中的“Requester ID Enable”位配合使用。
103:96	“Tag” (标签)	<p>与请求关联的 PCIe 标签。</p> <p>对于非转发传输事务，如果已设置 AXISTEN_IF_ENABLE_CLIENT_TAG 参数（即，当标签管理由用户应用执行时），集成块会使用来自该字段的值。位 [101:96] 用作为标签。位 [103:102] 为保留位。如果未设置此参数，那么集成块中的标签管理逻辑会生成要使用的标签，并且不使用描述符的标签字段中的值。</p>
119:104	“Completer ID” (完成器 ID)	<p>该字段仅适用于按 ID 进行路由的配置请求和报文。对于这些请求，该字段会指定与请求关联的 PCI 完成器 ID（这 16 个位将在传统解读模式下分割为 8 位总线编号、5 位器件编号和 3 位功能编号。而在 ARI 模式下，这 16 个位则作为 8 位总线编号 + 8 位功能编号来处理）。</p>
120	“Requester ID Enable” (请求器 ID 使能)	<p>1'b1：客户在描述符中提供总线编号、器件编号和功能编号，以供填充到 TLP 报头中的“Requester ID”（请求器 ID）字段中。</p> <p>1'b0：IP 使用从接收到的配置请求中捕获的总线和器件编号，客户在描述符中提供功能编号，以供填充到 TLP 报头中的“Requester ID”字段中。当“Requester ID Enable”（请求器 ID 使能）为 0 时，描述符中的器件编号字段也应为 0。</p> <p>端点模式：</p> <ul style="list-style-type: none">必须设置为 1'b0。 <p>开关的上游端口用例（在 IP 中选中端点模式）：</p> <ul style="list-style-type: none">如果请求源自开关本身，则设置为 1'b0。当开关正在中继请求（请求器位于开关外部）时，则设置为 1'b1。不启用 ARI 时，该位可与请求器总线编号位 [95:88] 和请求器功能/器件编号位 [87:83] 配合使用。 <p>根端口模式：</p> <ul style="list-style-type: none">必须设置为 1'b1。不启用 ARI 时，该位可与请求器总线编号位 [95:88] 和请求器功能/器件编号位 [87:83] 配合使用。
123:121	“Transaction Class (TC)” (传输事务类)	与请求关联的 PCIe 传输事务类 (TC)。
126:124	属性	这些位可提供与请求关联的 Attribute 位的设置。位 124 为“No Snoop”（无嗅探）位，位 125 则为“Relaxed Ordering”（宽松排序）位。位 126 为“ID-Based Ordering”（基于 ID 排序）位，只能针对存储器请求和报文进行设置。如果在功能的 PCI Express 器件控制寄存器中未启用对应属性，那么集成块在链路上发送的请求中会将属性位强制设置为 0。
127	“Force ECRC” (强制 ECRC)	强制执行 ECRC 插入。将该位设置为 1 会强制集成块将包含 ECRC 的 TLP 摘要追加到请求 TLP，即使针对发送请求的功能未启用 ECRC 也是如此。
15:0	“Snoop Latency” (嗅探时延)	该字段定义仅适用于 LTR 报文。它可提供报文的 TLP 报头中的 16 位“Snoop Latency”（嗅探时延）字段的值。
31:16	“No-Snoop Latency” (无嗅探时延)	该字段定义仅适用于 LTR 报文。它可提供报文的 TLP 报头中的 16 位“No-Snoop Latency”（无嗅探时延）字段的值。

表 54：请求器请求描述符字段 (续)

位索引	字段名称	描述
35:32	OBFF 代码	<p>“OBFF Code” 字段用于区分各种 OBFF 用例：</p> <ul style="list-style-type: none">· 1111b: CPU Active - 系统完全处于活动状态，可执行包括总线主控和中断在内的所有器件操作。· 0001b: OBFF - 可用于器件存储器读取/写入总线主控活动的系统存储器路径。· 0000b: Idle - 系统处于空闲低功耗状态。· 所有其它代码均为保留值。
111:104	“Message Code” (报文代码)	该字段定义适用于所有报文。其中包含要在 TL 报头中设置的 8 位报文代码。《PCI Express® 基本规范第 3.0 版》(《PCI-SIG 规范》(https://www.pcisig.com/specifications)) 的附录 F 提供了受支持的报文代码的完整列表。
114:112	“Message Routing” (报文路由)	该字段定义适用于所有报文。集成块会将这些位复制到请求 TLP 的 TLP 报头的 3 位路由字段 r[2:0] 中。
15:0	Destination ID	该字段仅适用于供应商定义的报文。当报文按 ID 进行路由 (即, 当 “Message Routing” (报文路由) 字段为 010 二进制值) 时, 该字段必须设置为报文的 “Destination ID” (目标 ID)。
63:32	“Vendor-Defined Header” (供应商定义的报头)	该字段仅适用于供应商定义的报文。它将被复制到 TLP 报头的 Dword 3 中。
63:0	ATS 报头	该字段仅适用于 ATS 报文。集成块会将其中包含的字节复制到 TLP 报头的 Dword 2 和 Dword 3 中。

请求器存储器写入操作

在 Dword 对齐模式下, 传输从 16 个描述符字节开始, 紧随其后即为有效载荷字节。用户应用必须在数据包的整个持续时间段保持 s_axis_rq_tvalid 信号处于断言有效状态。集成块会将数据包传输期间断言无效的 s_axis_rq_tvalid 作为错误来处理, 并将链路上发射的对应请求 TLP 置空, 以避免数据损坏。

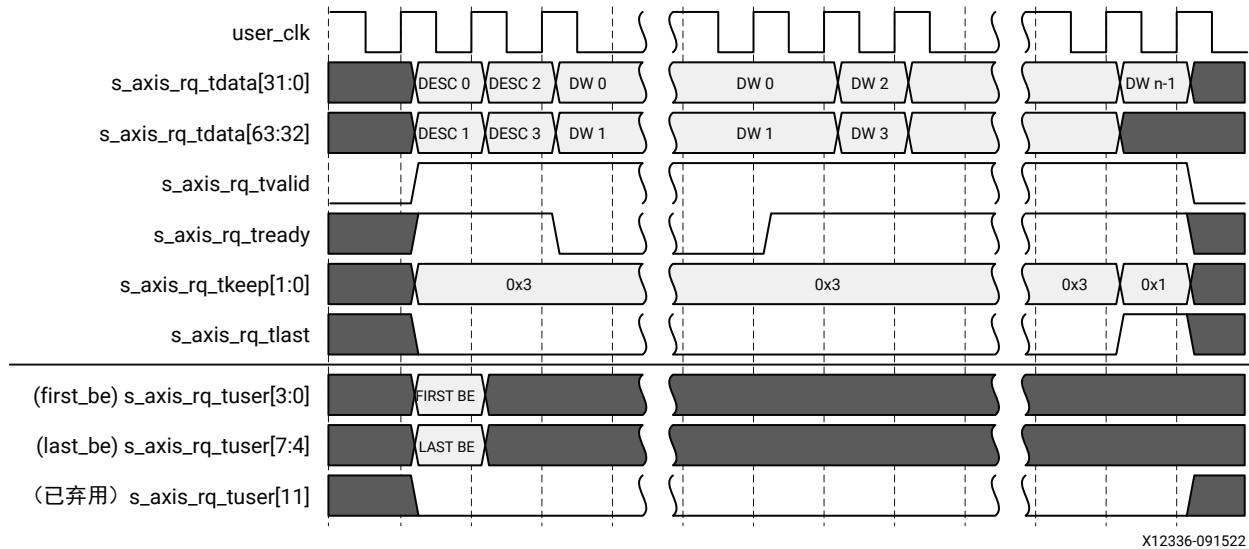
用户应用还必须在数据包的最后一拍中断言 s_axis_rq_tlast 信号有效。如果集成块未准备好接受数据, 那么可在任意周期内断言 s_axis_rq_tready 无效。在集成块已断言 s_axis_rq_tready 无效的周期内, 用户应用不得更改 RQ 接口上的值。必须设置 AXI4-Stream 接口信号 m_axis_rq_tkeep (每个信号对应 1 个 Dword 位置) 以指示数据包中的有效 Dword, 包括描述符和描述符与有效载荷之间插入的任意空字节。即, 从描述符的第一个 Dword 开始到有效载荷的最后一个 Dword 为止的所有 tkeep 位都必须连续设置为 1。在数据包传输期间, 当数据包无法填满接口的完整宽度时, tkeep 位仅限在数据包的最后一拍内才能设为 0。

请求器请求接口在 s_axis_rq_tuser 总线中还包含 “First Byte Enable” (首字节使能) 位和 “Last Byte Enable” (末字节使能) 位。在数据包的第一拍内必须设置这些位, 并提供有关有效载荷的第一个和最后一个 Dword 中的有效字节的信息。

用户应用必须根据集成块中配置的最大有效载荷大小来限制单一请求内传输的有效载荷大小, 并且必须确保有效载荷不超过 4 KB 的界限。对于不超过 2 个 Dword 的存储器写入, first_be 和 last_be 上值为 1 的位可能不连续。另一种特殊情况是对于长度为 0 的存储器写入请求, 用户应用必须提供虚拟有效载荷, 其中包含 1 个 Dword 且 first_be 和 last_be 都全部设置为 0。对于仅含 1 个 DW 的传输, last_be[3:0] 应为 0, 且 first_be[3:0] 可指示有效字节。在所有其它情况下, first_be 和 last_be 中值为 1 的位必须连续。

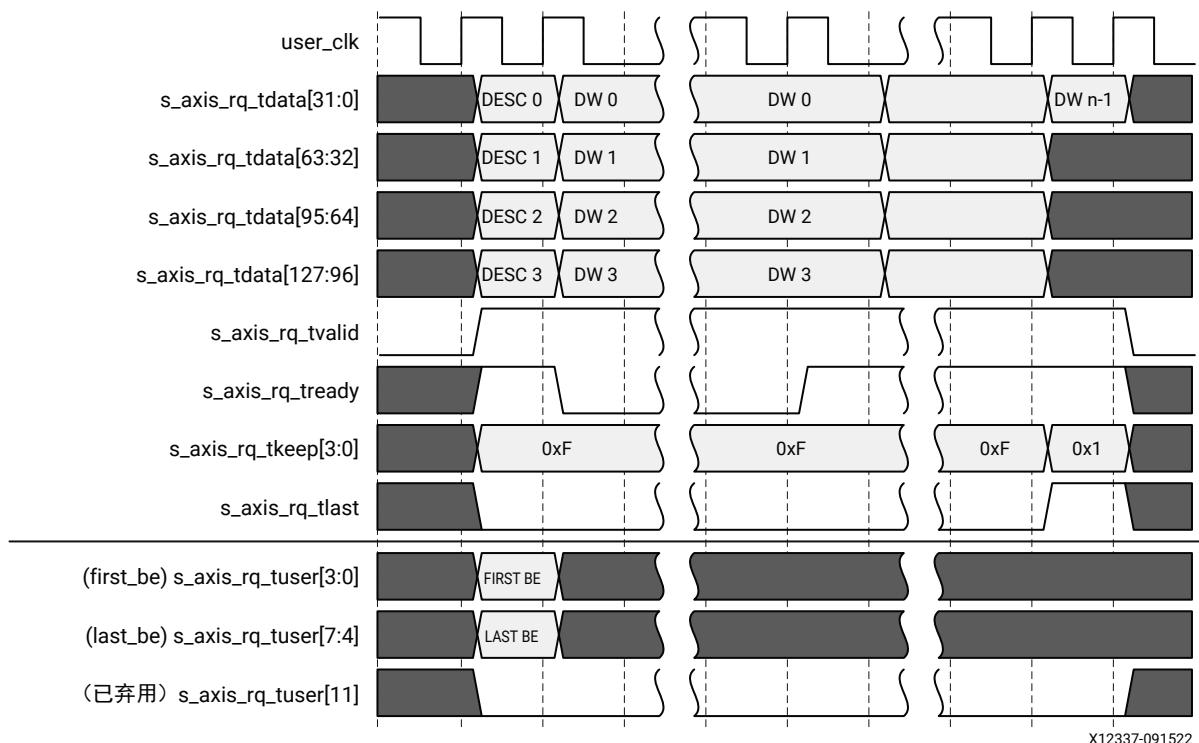
以下时序图显示了当接口宽度分别配置为 64 位、128 位和 256 位时, 通过请求器请求接口来传输源于用户应用的存储器写入请求 (采用 Dword 对齐模式) 的过程。为便于演示, 写入用户应用存储器的数据块大小假定为 n 个 Dword, 其中, $n = k \times 32 + 29$ 且 $k > 0$ 。

图 45：请求器请求接口上的存储器写入传输事务（Dword 对齐模式、64 位接口）



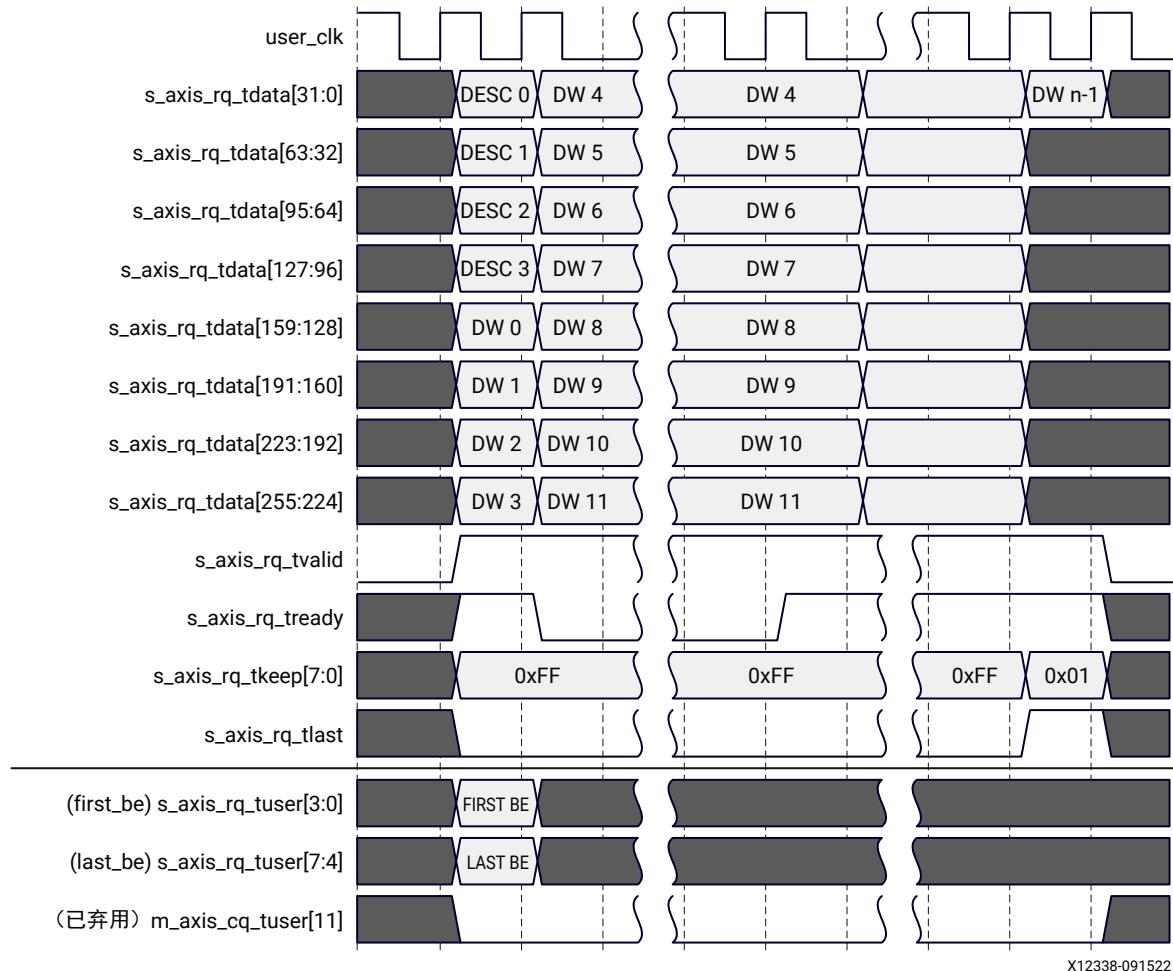
X12336-091522

图 46：请求器请求接口上的存储器写入传输事务（Dword 对齐模式、128 位接口）



X12337-091522

图 47：请求器请求接口上的存储器写入传输事务（Dword 对齐模式、256 位接口）



X12338-091522

以下时序图显示了当接口宽度分别配置为 64 位、128 位和 256 位时，通过 RQ 接口来传输源于用户应用的存储器写入请求（采用地址对齐模式）的过程。为便于演示，写入用户应用存储器的数据块的起始 Dword 偏移假定为 $(m \times 32 + 1)$ ，其中 $m > 0$ 且为整数。其大小假定为 n 个 Dword，其中 $n = k \times 32 + 29$ 且 $k > 0$ 。

在地址对齐模式下，有效载荷的交付始终从描述符的最后一个字节后的节拍中开始。有效载荷的第一个 Dword 可显示在任一 Dword 位置。用户应用必须在数据路径上使用 **s_axis_rq_tuser** 中的 **addr_offset[2:0]** 信号来传达有效载荷的第一个 Dword 的偏移。用户应用还必须在 **first_be[3:0]** 中设置相应的位以指示第一个 Dword 中的有效字节，并在 **last_be[3:0]** 中设置相应的位以指示有效载荷的最后一个 Dword 中的有效字节。

图 48：请求器请求接口上的存储器写入传输事务（地址对齐模式、64 位接口）

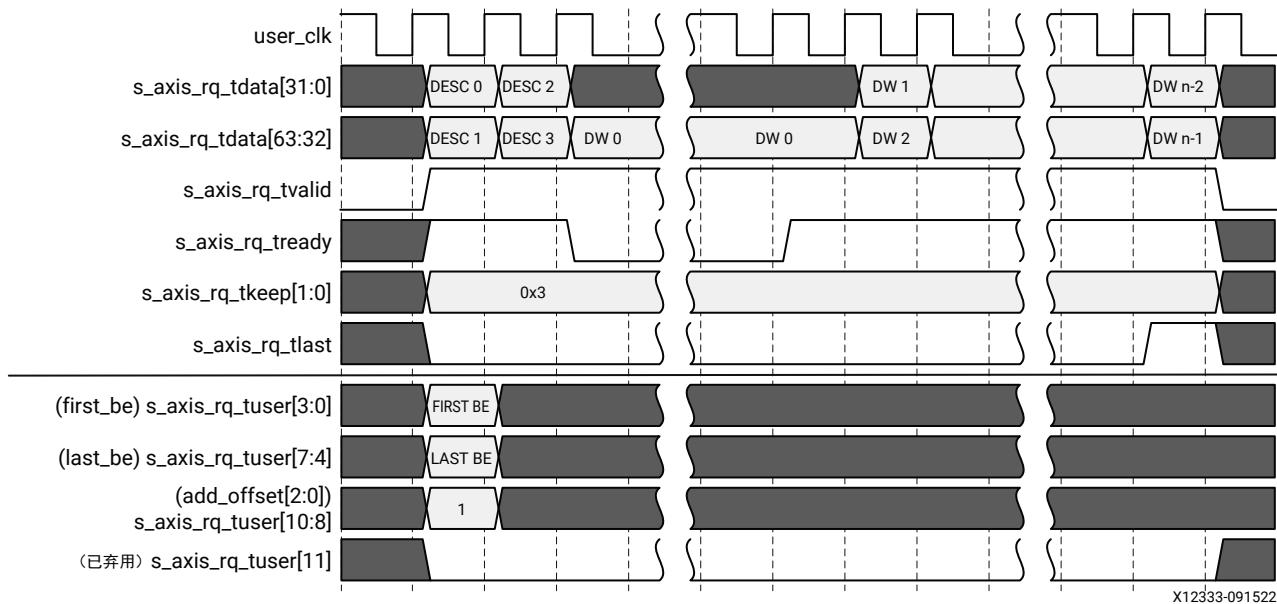


图 49：请求器请求接口上的存储器写入传输事务（地址对齐模式、128 位接口）

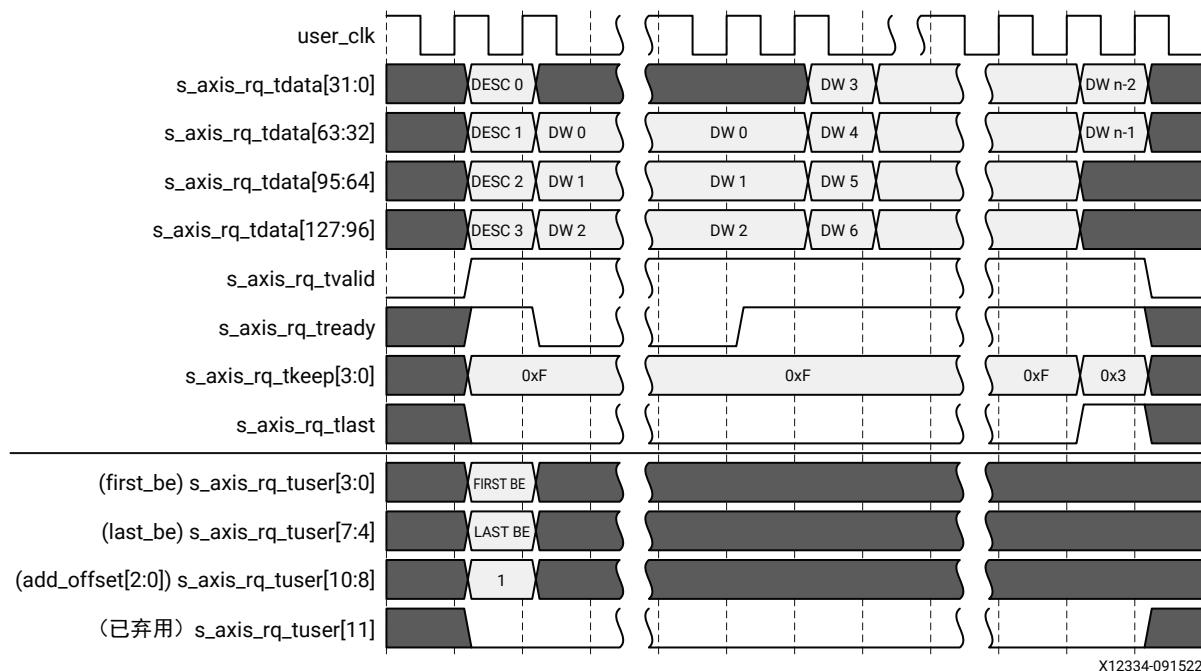
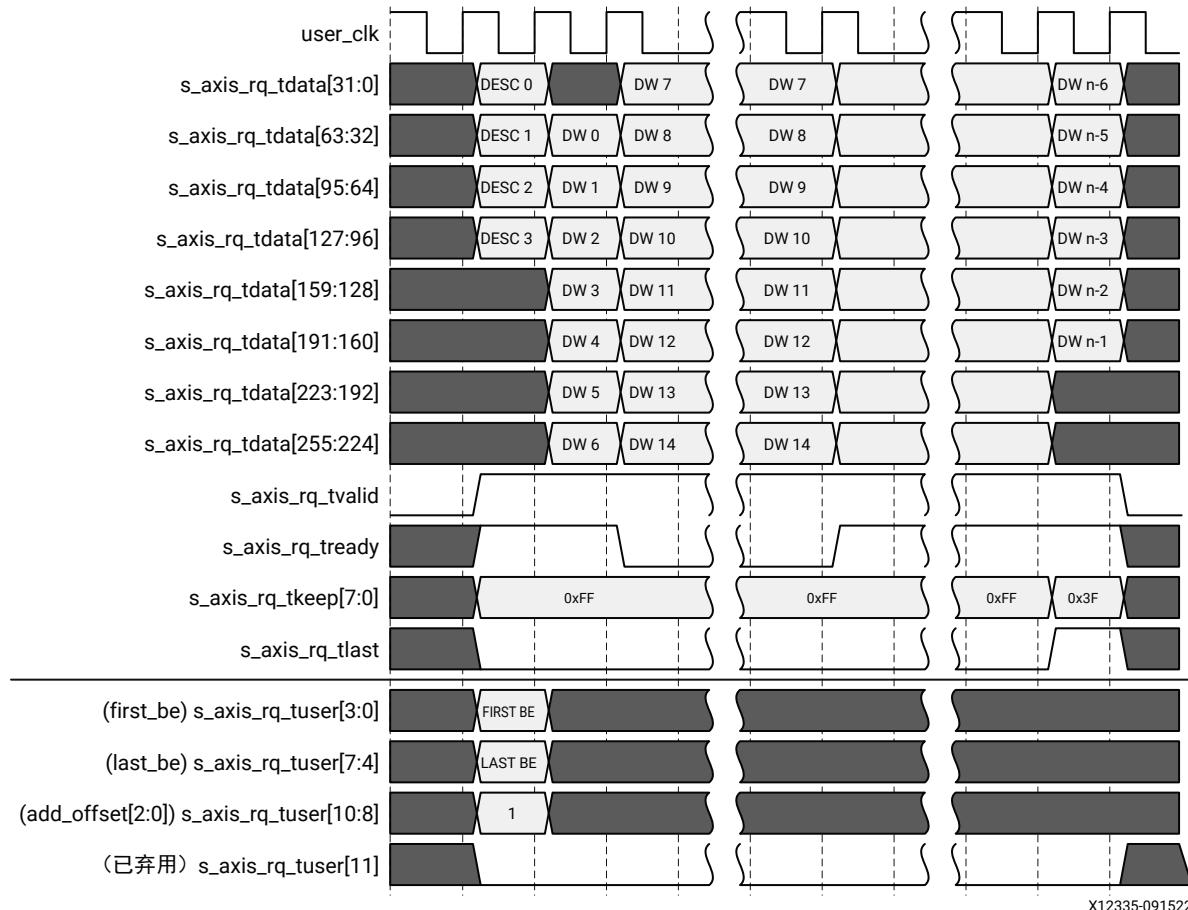


图 50：请求器请求接口上的存储器写入传输事务（地址对齐模式、256 位接口）



不含有效载荷的非转发传输事务

不含有效载荷（存储器读取请求、I/O 读取请求、配置读取请求）的非转发传输事务将按存储器写入请求相同的方式跨 RQ 接口进行传输，但区别在于 AXI4-Stream 数据包仅包含 16 字节描述符。以下时序图显示了当接口宽度分别配置为 64 位、128 位和 256 位时，通过 RQ 接口来传输存储器读取请求的过程。此数据包在 64 位接口上占据连续 2 个节拍，但在 128 位和 256 位接口上，其传输只需一拍即可。在数据包持续时间段内，**s_axis_rq_tvalid** 信号必须保持处于断言有效状态。集成块可断言 **s_axis_rq_tready** 无效以延长节拍。**s_axis_rq_tlast** 信号必须在数据包的最后一拍内设置，**s_axis_rq_tkeep[7:0]** 中的位则必须在存在描述符的所有 Dword 位置内进行设置。

要读取的数据块的第一个和最后一个 Dword 中的有效字节必须分别使用 **first_be[3:0]** 和 **last_be[3:0]** 来表示。对于长度为 0 的存储器读取的特殊情况，请求长度必须设置为 1 个 Dword，且 **first_be[3:0]** 和 **last_be[3:0]** 全部设置为 0。此外在地址对齐模式下，**s_axis_rq_tuser** 中的 **addr_offset[2:0]** 可用于指定请求器完成接口上返回的数据的期望的起始对齐位置。此对齐无需与请求地址相关联。

图 51：请求器请求接口上的存储器读取传输事务（64 位接口）

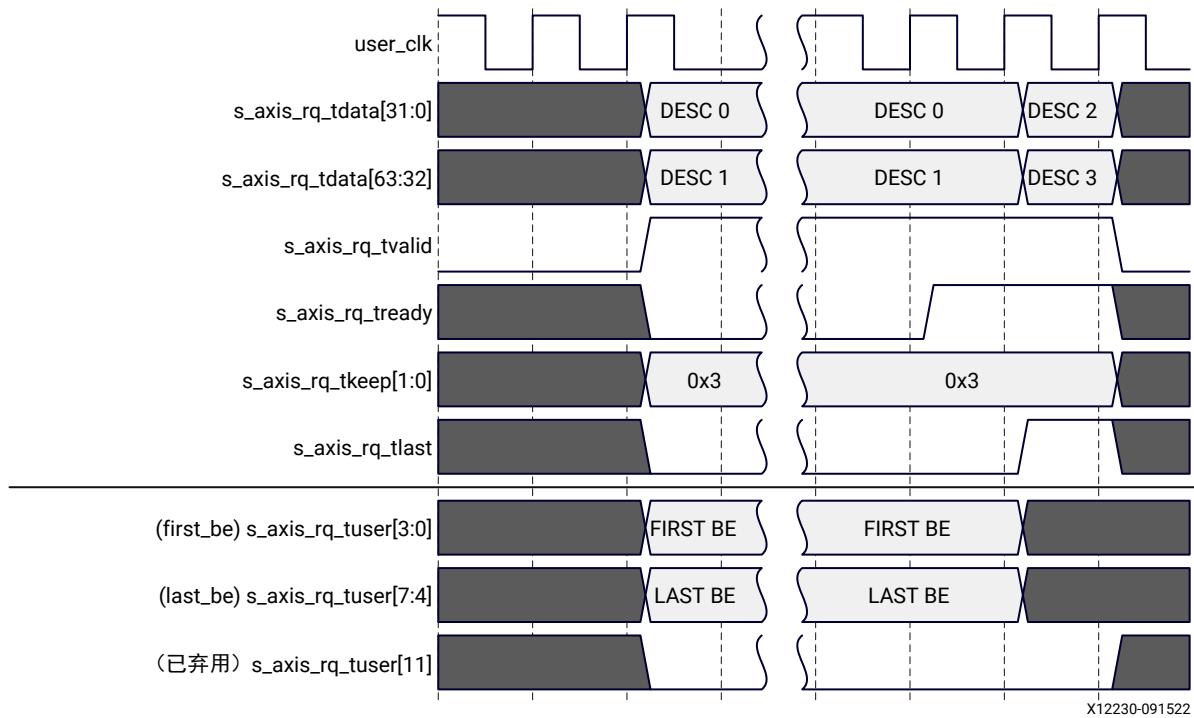


图 52：请求器请求接口上的存储器读取传输事务（128 位接口）

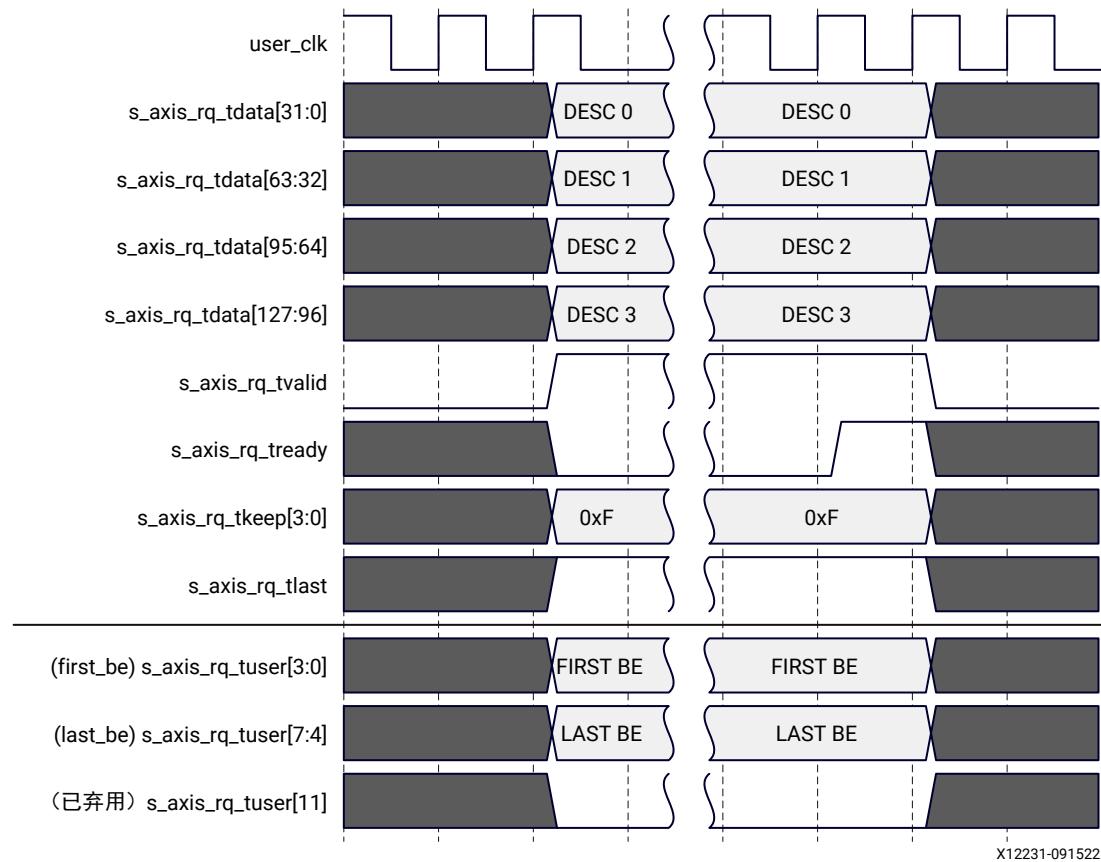
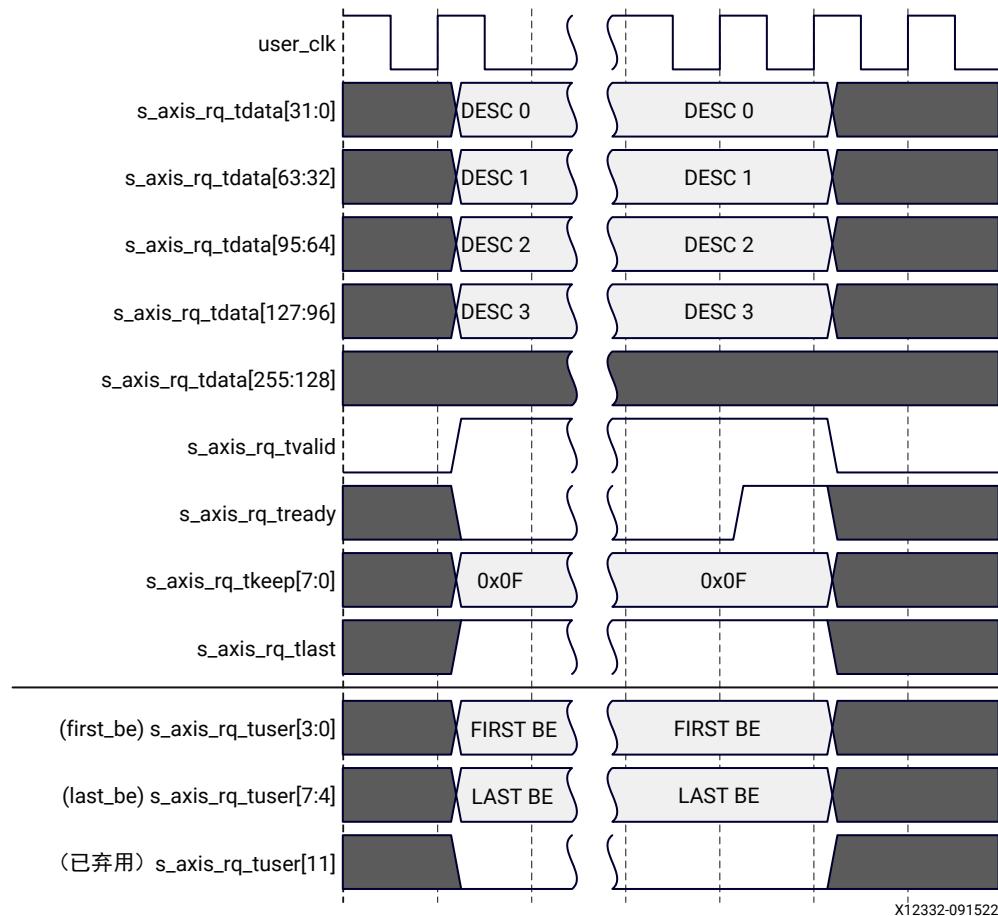


图 53：请求器请求接口上的存储器读取传输事务（256 位接口）



含有效载荷的非转发传输事务

含有效载荷的非转发请求（I/O 写入请求、配置写入请求或原子操作请求）的传输与存储器请求类似，但数据路径上有效载荷的对齐方式存在如下改变：

- 在 Dword 对齐模式下，有效载荷的第一个 Dword 位于描述符的最后一个 Dword 之后，两者间没有间隙。
 - 在地址对齐模式下，有效载荷必须始终从描述符的最后一个 Dword 后的节拍中开始。有效载荷可在数据路径上的任意 Dword 位置开始。其第一个 Dword 的偏移必须使用 `addr_offset[2:0]` 信号来指定。

对于 I/O 和配置写入请求，必须使用 `first_be[3:0]` 来明示单 Dword 有效载荷中的有效字节。对于原子操作请求，第一个和最后一个 Dword 中的所有字节均假定有效。

请求器接口上的报文请求

在 RQ 接口上传输报文的过程与存储器写入请求的传输过程类似，但有时其中可能不存在有效载荷。传输以 128 位描述符开始，其后即为有效载荷（如果存在）。当使用 Dword 对齐模式时，有效载荷的第一个 Dword 必须紧接在描述符之后。当使用地址对齐模式时，有效载荷必须在描述符之后的节拍中开始，并且必须对齐到字节通道 0。当使用地址对齐模式时，集成块的 `addr_offset` 输入针对报文必须设置为 0。集成块可根据 `s_axis_rq_tlast` 信号和 `s_axis_rq_tkeep` 信号来判定有效载荷的结束位置。首字节使能位和末字节使能位 (`first_be` 和 `last_be`) 都不用于报文请求。

传输异常中止

对于包含关联有效载荷的任意请求，用户应用可以在传输有效载荷期间随时通过断言 `s_axis_rq_tuser` 总线中的 `discontinue` 信号有效来异常中止该请求。集成块会将链路上对应 TLP 置空，以避免数据损坏。

在传输期间，当传输的请求具有关联有效载荷时，用户应用可以在任意周期内断言此信号有效。用户应用可以选择在周期内发出错误信号处提前终止该数据包（通过断言 `s_axis_rq_tlast` 有效），或者也可以继续处理，直至将有效载荷的所有字节都交付到集成块为止。针对后者，该集成块会针对数据包的后续节拍将此错误作为粘滞错误来处理，即使用户应用在达到包结束前断言 `discontinue` 信号无效也是如此。

仅当 `s_axis_rq_tvalid` 为高电平有效时，才能断言 `discontinue` 信号有效。当 `s_axis_rq_tvalid` 和 `s_axis_rq_tready` 均为高电平有效时，集成块会对此信号进行采样。因此，断言有效后，在 `s_axis_rq_tready` 处于高电平有效状态之前不应断言 `discontinue` 信号无效。

当集成块配置为端点时，集成块会使用高级错误报告 (AER) 机制向所连接到的根联合体报告此错误（作为不可纠正的内部错误）。

非转发传输事务的标签管理

集成块的请求器侧可保留由用户应用发起的所有暂挂非转发传输事务（存储器读取、I/O 读取和写入、配置读取和写入以及原子操作）的状态，因此目标所返回的完成包可与对应请求相匹配。每个未完成的传输事务的状态都保存在接口的请求器侧的拆分完成包表中，其容量为不超过 256 个非转发传输事务。返回的完成包将使用 8 位标签与暂挂请求相匹配。有 2 个选项可用于这些标签的管理。

- “Internal Tag Management”（内部标签管理）：如果在 Vivado® IDE 中未设置“Enable Client Tag”（启用客户标签）选项（该核的默认设置），则启用“Internal Tag Management”。在此模式下，集成块中的逻辑负责为从请求器侧发起的每个非转发请求分配标签。当用户应用发起非转发传输事务时，集成块会保留可用标签列表并向每个请求分配 1 个标签，并通过输出 `pcie_rq_tag0[7:0]` 将分配的标签值传达给用户应用。当集成块断言 `pcie_rq_tag_vld0` 有效时，该总线上的值即有效。用户逻辑必须复制此标签以便使集成块在请求的响应中所交付的任意完成包都与请求相匹配。

在此模式下，集成块中的逻辑会检查“拆分完成包表”的完整条件，如果当前未完成的非转发请求总数已达到上限，则会对来自用户应用的非转发请求执行反压（使用 `s_axis_rq_tready`）。

- “External Tag Management”（外部标签管理）：在此模式下，用户逻辑负责为从请求器侧发起的每个非转发请求分配标签。用户逻辑所选择的标签值与当时未完成的所有其它非转发传输事务的标签之间不得存在任何冲突，并且必须将此所选标签值通过请求描述符传达给集成块。集成块仍在其“拆分完成包表”中保留未完成的请求，并将传入完成包与请求相匹配，但不会对标签的唯一性进行任何检查，也不会检查“拆分完成包表”的完整条件。

使用“Internal Tag Management”时，集成块会断言 `pcie_rq_tag_vld0` 有效并为每个非转发请求保持 1 个周期，然后它会将其分配的标签置于 `pcie_rq_tag[7:0]` 上。在 RQ 总线上传输请求后，可能要经过数个周期的延迟之后，集成块才会断言 `pcie_rq_tag_vld0` 有效以便为请求提供已分配的标签。与此同时，用户应用可以继续发送新请求。在 `pcie_rq_tag0` 总线上，请求的标签按 FIFO 顺序来传递，这样即可轻松将标签值与其传输的请求相关联。当用户应用接受拆分完成包的最后一个完成包的帧结束 (EOF) 时，即可复用标签。

避免转发请求发生队头阻塞

集成块会因缺少发射信用值或者缺少可用标签而将其 RQ 接口上接收到的非转发请求置于保留状态。这可能导致转发传输事务发生队头阻塞 (head-of-line (HOL) blocking)。集成块为用户逻辑提供了相应的机制，可通过以下信号避免出现此状况：

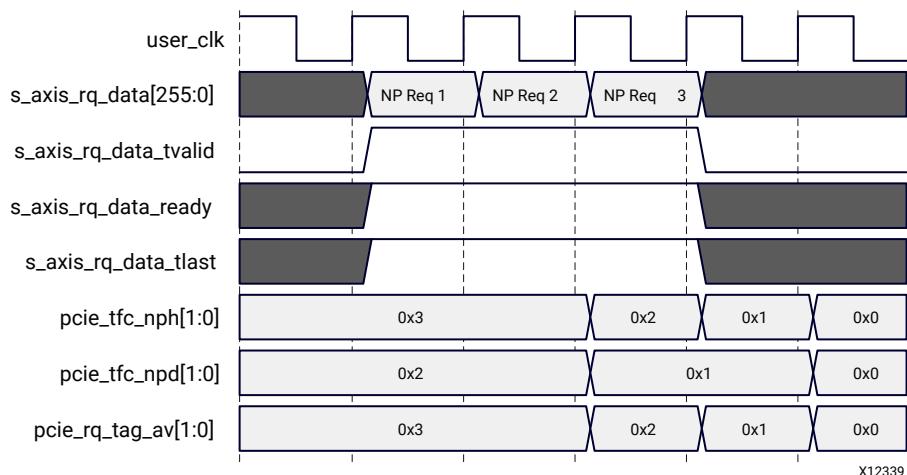
- `pcie_tfc_nph_av[1:0]`：这些输出表示非转发请求的当前可用头信用值，其中：
 - 00 = 无信用值可用
 - 01 = 信用值为 1
 - 10 = 信用值为 2

11 = 信用值不少于 3

- pcie_tfc_npd_av[1:0]：这些输出表示非转发请求的当前可用数据信用值，其中：
00 = 无信用值可用
01 = 信用值为 1
10 = 信用值为 2
11 = 信用值不少于 3

用户逻辑可选择先检查这些输出，然后再发射非转发请求。由于存在内部流水线延迟，这些输出上的信息相比于 RQ 接口上传输的描述符的最后一个字节所在周期晚 2 个用户时钟周期。因此用户逻辑必须调整这些值，将前 2 个时钟周期内发射的所有非转发请求一并纳入考量。下图演示了针对 256 位接口，这些信号的操作。在此示例中，集成块最初的非转发头信用值为 3，非转发数据信用值为 2，并且可供分配的可用标签数为 3。来自用户应用的请求 1 具有 1 个有效载荷（含 1 个 Dword），因此耗用的头信用值和数据信用值各为 1，使用的标签数同样为 1。下一个时钟周期内的请求 2 耗用的头信用值为 1，但不耗用数据信用值。当用户应用在下一个时钟周期内提供请求 3 时，它必须调整可用信用值和可用标签计数，将请求 1 和 2 一并纳入考量。如果请求 3 耗用的头信用值和数据信用值各为 1，那么 2 个周期后这 2 种可用信用值均为 0，可用标签数量同样如此。

图 54：请求器请求接口（256 位接口）上信用值和标签可用性信号



下图显示了相同示例分别对应接口宽度为 128 位和 64 位情况下的信用值和标签可用性信号的时序。

图 55：请求器请求接口（128 位接口）上信用值和标签可用性信号

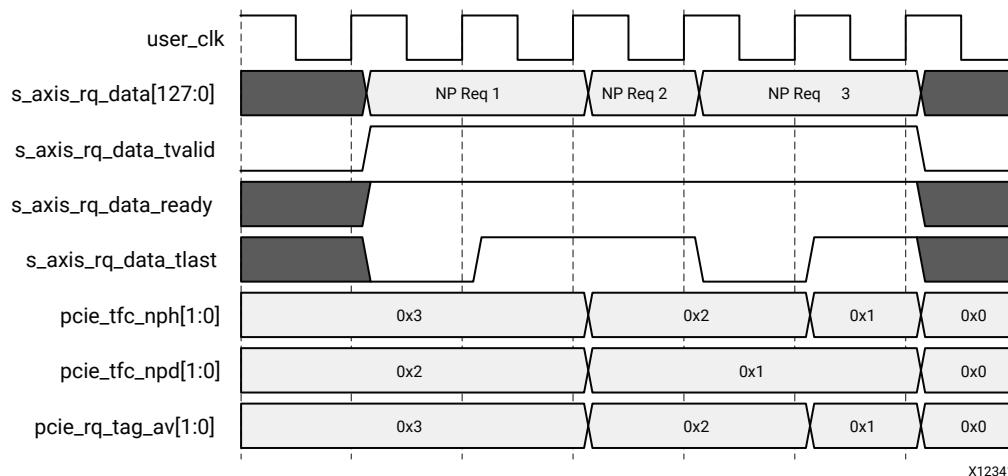
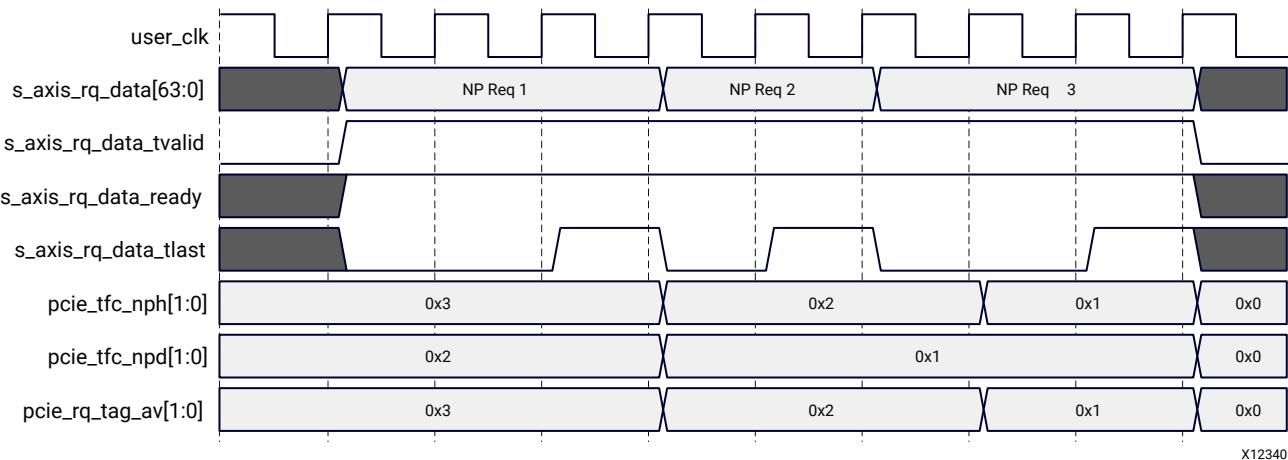


图 56：请求器请求接口（64 位接口）上信用值和标签可用性信号



注释：如果用户逻辑选择使用 `pcie_tfc_*` 接口来监控可用的发射信用值，请确保在 `pcie_tfc_npd_av` 或 `pcie_tfc_nph_av` 达到 0 之后，没有任何其它非转发数据包进入 RQ 接口。集成块并不会丢失在此之后发出的非转发数据包；但 `pcie_tfc_*` 接口将无法再提供准确的信用值审计。

当 `cfg_fc_sel` 设置为可用发射信用值模式后，在 `cfg_fc_npd` 和 `cfg_fc_nph` 接口中同样可提供类似的发射信用值信息。

保持传输事务排序不变

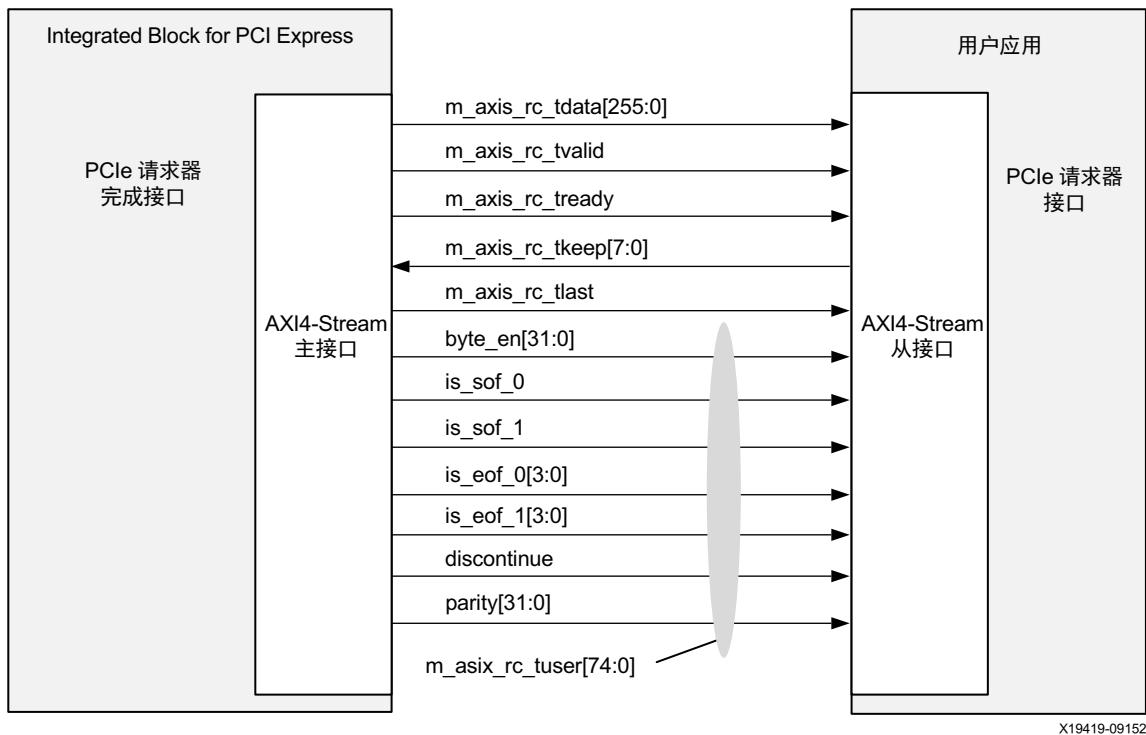
集成块在链路上发射用户应用请求的顺序与其在请求器接口上接收这些请求的顺序相同。如果用户应用想要精确控制 RQ 接口和 CC 接口上发送的传输事务的顺序（通常目的是在使用严格排序时，禁止完成包传递转发请求），那么集成块会为用户应用提供相应的机制，以便监控转发传输事务通过其流水线的整个过程，从而判定何时在完成器完成接口上调度完成包，同时可以避免传递从请求器请求接口发射的特定转发请求。

在跨请求器请求接口传输转发请求（存储器写入传输事务或报文）时，用户应用可以在其 `s_axis_rq_tuser` 内的 `seq_num[3:0]` 输入上向集成块提供可选 4 位序列号。此序列号在数据包的第一拍内必须有效。随后，用户应用即可监控该核的 `pcie_rq_seq_num[3:0]` 输出，以确认何时出现这些序列号。当传输事务达到集成块的内部发射流水线中的相应阶段，且在此阶段中完成包无法传递该传输事务时，集成块会断言 `pcie_rq_seq_num_valid` 有效并保持 1 个周期，并在 `pcie_rq_seq_num[3:0]` 输出上提供转发请求的序列号。在 `pcie_rq_seq_num[3:0]` 上出现序列号之后，由集成块发射的任何完成包均无法在内部发射流水线中传递转发请求。

请求器完成接口操作

对应由用户逻辑生成的请求的完成包显示在集成块请求完成 (RC) 接口上。请参阅下图，其中显示了与请求器完成接口关联的信号。未启用跨接时，该集成块将此接口上的每个 TLP 均作为 1 个 AXI4-Stream 数据包来交付。对于含有效载荷的完成包，此数据包以 96 位描述符开头并后接数据。

图 57：请求器完成接口



RC 接口支持通过 2 种不同数据对齐模式来传输有效载荷。在 Dword 对齐模式下，集成块会紧接在描述符的最后一个 Dword 之后传输完成包有效载荷的第一个 Dword。在地址对齐模式下，集成块会在描述符的最后一个 Dword 之后的节拍中开始有效载荷的传输，其第一个 Dword 可位于数据路径上的任意可能的 Dword 位置内。有效载荷的第一个 Dword 的对齐方式将根据用户应用在向集成块发送请求时提供的地址偏移（即 RQ 接口的 `addr_offset[2:0]` 输入设置）来判定。因此，仅当 RQ 接口同样配置为使用地址对齐模式时，才能在 RC 接口上使用地址对齐模式。

请求器完成描述符格式

集成块的 RC 接口可将从链路接收到的完成数据作为 AXI4-Stream 数据包发送到用户应用。每个数据包均以 1 个描述符开头，在描述符后可包含有效载荷数据。描述符长度始终为 12 字节，并在完成包的前 12 字节内发送。描述符在 64 位接口上的前 2 个节拍内进行传输，在 128 位或 256 位接口上的第 1 个节拍内进行传输。当完成数据被拆分为多个拆分完成包后，集成块会将每个拆分完成包作为含专有描述符的独立 AXI4-Stream 数据包进行发送。

下图演示了请求器完成描述符的格式。下表描述了 RC 描述符的每个字段。

图 58：请求器完成描述符格式

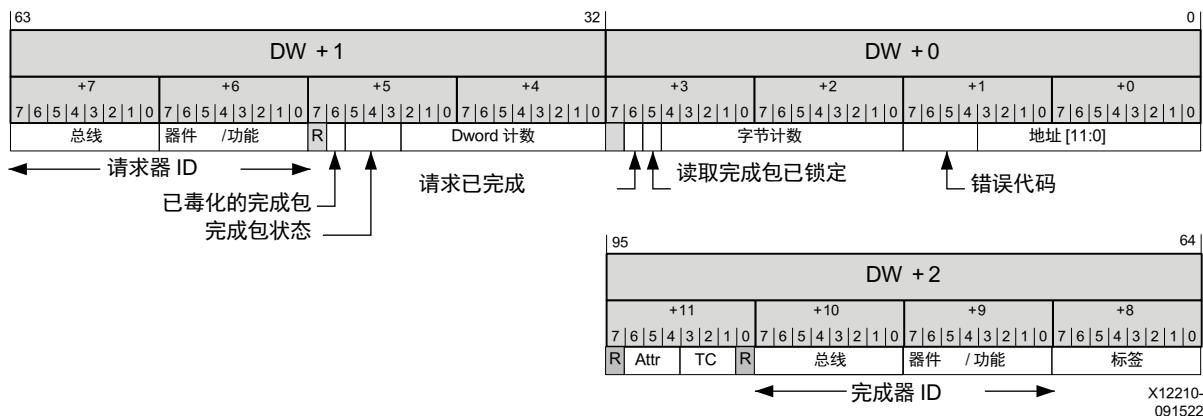


表 55：请求器完成描述符字段

位索引	字段名称	描述
11:0	“Lower Address”（低位地址）	<p>该字段可提供请求引用的第 1 个字节的 12 个最低有效位。集成块从其“Split Completion Table”（拆分完成包表）返回该地址，在该表中存储有请求器侧的所有暂挂非转发请求的地址和其它参数。</p> <p>当完成包交付错误时，应仅将该地址的位 [6:0] 视为有效。</p> <p>该地址为字节级别地址。</p> <p>对于 ATS 转换请求，该字段为保留字段，暗示其值应为 0。</p>
15:12	“Error Code”（错误代码）	<p>完成包错误代码。</p> <p>这 3 位代码表示集成块对接收到的完成包所执行的错误检查检测到存在错误状况。其编码为：</p> <ul style="list-style-type: none"> 0000：正常终止（已接收到所有数据）。 0001：完成 TLP 已被毒化。 0010：因出现状态为 UR、CA 或 CRS 的完成包，请求已被终止。 0011：因出现不含任何数据的完成包，或者完成包中的字节计数高于请求期望的字节总数，请求已被终止。 0100：当前交付的完成包所含标签与未完成请求的标签相同，但其 Requester ID、TC 或 Attr 字段与未完成请求的参数不匹配。 0101：起始地址中存在错误。完成 TLP 报头中的下位地址位与请求的下一个期望字节的起始位置不匹配。 0110：标签无效。此完成包与任意未完成请求的标签都不匹配。 1001：请求已终止，完成包超时。描述符中除位 [30]、请求器功能 [55:48] 和标签字段 [71:64] 以外的其它字段在此情况下都无效，因为该描述符不对应于完成 TLP。 1000：因出现以生成请求的功能为目标的功能级别复位 (FLR)，此请求已被终止。描述符中除位 [30]、请求器功能 [55:48] 和标签字段 [71:64] 以外的其它字段在此情况下都无效，因为该描述符不对应于完成 TLP。

表 55：请求器完成描述符字段 (续)

位索引	字段名称	描述
28:16	“Byte Count” (字节计数)	<p>这 13 个位可包含范围在 0 - 4,096 个字节内的值。如果存储器读取请求已通过使用单一完成包完成，那么“Byte Count”值将以字节为单位来表示“Payload”（有效载荷）大小。针对 I/O 读取完成包和 I/O 写入完成包，该字段必须设置为 4。针对长度为 0 的存储器读取发送完成包时，字节计数必须设置为 1，并且在描述符之后必须附加含单 Dword 的虚拟有效载荷。</p> <p>对于每个“Memory Read Completion”（存储器读取完成），“Byte Count”字段必须表明完成请求所需的剩余字节数，包括随完成包返回的字节数。</p> <p>如果存储器读取请求已使用多个完成包完成，那么后续每个完成包的字节计数值均表示为前一个完成包减去随前一个完成包返回的字节数。</p>
29	“Locked Read Completion” (锁定读取完成)	当“Locked Read”（锁定读取）请求的响应中包含“Completion”（完成包）时，该位设置为 1。针对所有其它完成包，该位均设置为 0。
30	“Request Completed” (请求完成)	<p>集成块在请求的最后一个完成包的描述符中断言该位有效。断言该位有效可表示请求正常终止（因为已接收到所有数据）或者异常终止（因为存在错误状况）。用户逻辑可使用该指示来清除其未完成的请求状态。</p> <p>分配标签时，用户逻辑应在接收到来自集成块的完成描述符并且其中包含匹配的标签字段且“Request Completed”（请求完成）位已设置为 1 之后，才能将标签重新分配到请求。</p>
42:32	“Dword Count” (Dword 计数)	这 11 个位表示当前包的有效载荷大小（以 Dword 数为单位）。其范围是 0 - 1000 个 Dword。针对 I/O 读取完成包，该字段设置为 1，针对 I/O 写入完成包，该字段设置为 0。针对长度为 0 的存储器读取传输完成包时，Dword 计数同样设置为 1。在所有其它情况下，Dword 计数都对应于当前包的有效载荷中 Dword 的实际数量。
45:43	“Completion Status” (完成状态)	<p>这些位可反映接收到的完成 TLP 的“Completion Status”（完成状态）字段的设置。有效设置包括：</p> <ul style="list-style-type: none"> · 000：成功完成 · 001：请求不受支持 (UR) · 010：配置请求重试状态 (CRS) · 100：完成器异常中止 (CA)
46	“Poisoned Completion” (毒化完成)	设置该位可指示完成 TLP 中已设置“Poison”（毒化）位。在此情况下，应将数据包中的数据视为已损坏。
63:48	“Requester ID” (请求器 ID)	与完成包关联的 PCI 请求器 ID。
71:64	“Tag” (标签)	与完成包关联的 PCIe 标签。
87:72	“Completer ID” (完成器 ID)	在完成 TLP 中接收到的完成器 ID。（这 16 个位在传统解读模式下分割为 8 位总线编号、5 位器件编号和 3 位功能编号。而在 ARI 模式下，这 16 个位则必须作为 8 位总线编号 + 8 位功能编号来处理。）。
91:89	“Transaction Class (TC)” (传输事务类)	与完成包关联的 PCIe 传输事务类 (TC)。
94:92	“Attributes” (属性)	与完成包关联的 PCIe 属性。位 92 为“No Snoop”（无嗅探）位，位 93 为“Relaxed Ordering”（宽松排序）位，位 94 为“ID-Based Ordering”（基于 ID 排序）位。

不含数据的完成包的传输

以下时序图显示了当接口宽度分别配置为 64 位、128 位和 256 位时，从不含关联有效载荷的链路接收到完成 TLP 后，通过 RC 接口来进行传输的过程。本章节中的时序图假定在 256 位接口上，完成 (Completion) 包不存在跨接。在 [256 位接口的跨接选项](#) 中对跨接功能进行了描述。

图 59：请求器完成接口上不含数据的完成包的传输（64 位接口）

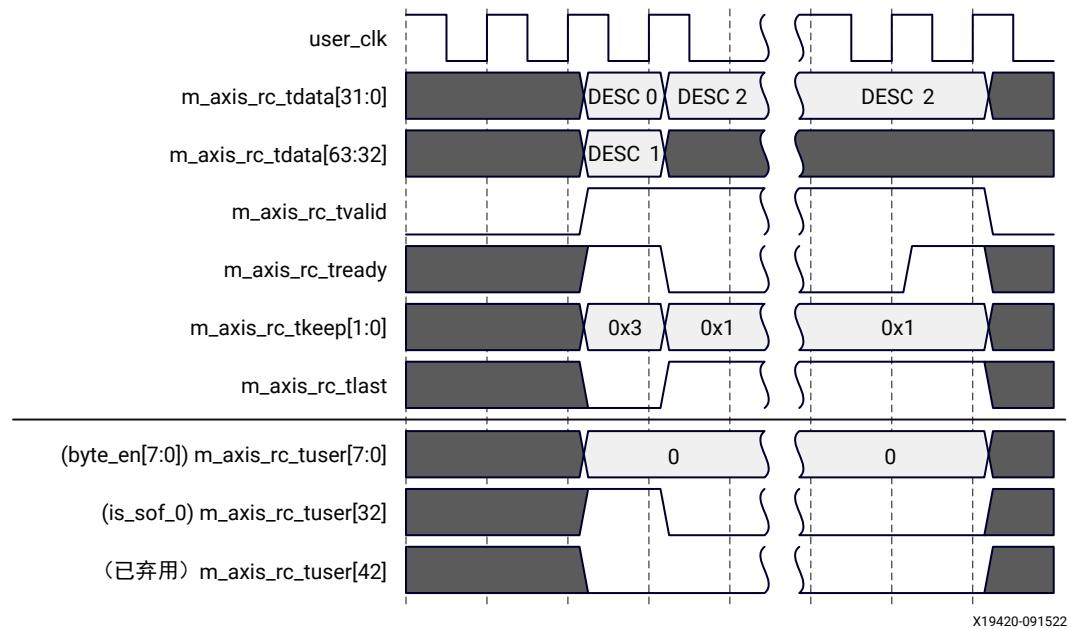


图 60：请求器完成接口上不含数据的完成包的传输（128 位接口）

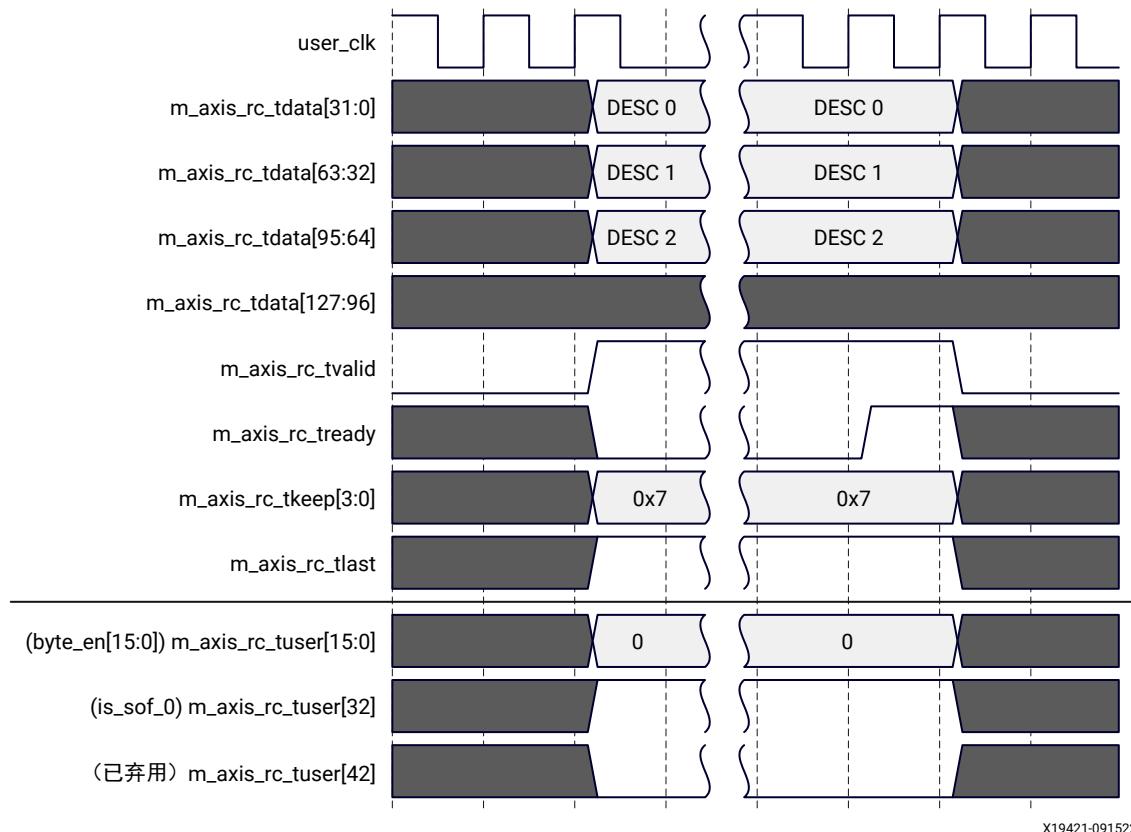
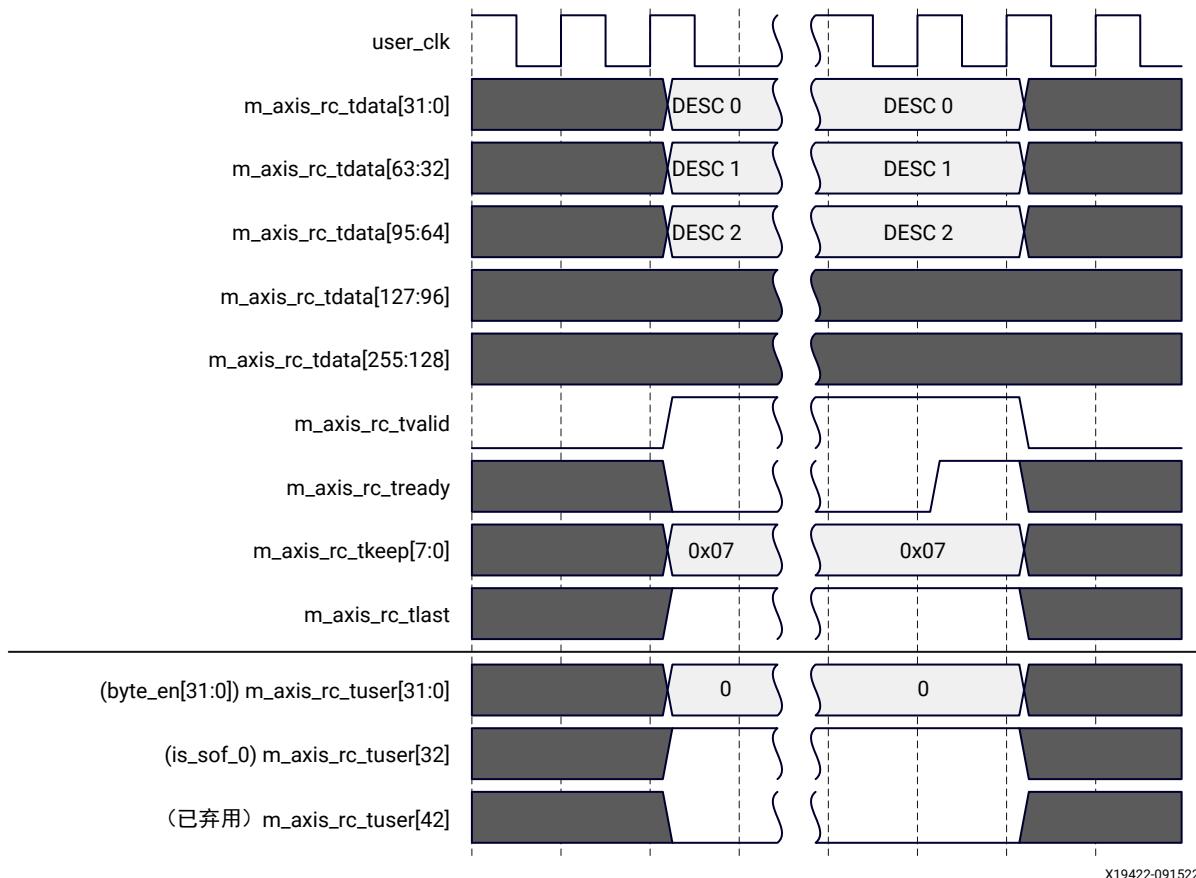


图 61：请求器完成接口上不含数据的完成包的传输（256 位接口）



完成 TLP 的整个传输过程在 256 位和 128 位接口上仅需一拍，在 64 位接口上则需两拍。集成块会在数据包的整个持续时间段保持 `m_axis_rc_tvalid` 信号处于断言有效状态。用户应用可以随时通过断言 `m_axis_rc_tready` 无效来延长任一节拍。AXI4-Stream 接口可发出 `m_axis_rc_tkeep` 信号（每个信号对应 1 个 Dword 位置）以指示数据包中的有效描述符 Dword。即，从描述符的第一个 Dword 开始到最后一个 Dword 为止的所有 `tkeep` 位均连续设置为 1。在数据包传输期间，`tkeep` 位仅限在数据包的最后一拍内才能设为 0。在数据包的最后一拍内，`m_axis_rc_tlast` 信号始终断言有效。

`m_axis_rc_tuser` 总线还包含 `is_sof_0` 信号，在每个数据包的第一拍内，此信号均断言有效。用户应用可以选择使用此信号来限定接口上描述符的起始位置。不使用跨接选项时，`m_axis_rc_tuser` 中的所有其它信号与不含数据的完成包的传输都不相关。

含数据完成包的传输

以下时序图显示了当接口宽度分别配置为 64 位、128 位和 256 位时，从含关联有效载荷的链路接收到完成 TLP（采用 Dword 对齐模式）后，通过 RC 接口来进行传输的过程。为便于演示，写入用户应用存储器的数据块大小假定为 n 个 Dword，其中， $n = k \times 32 + 28$ 且 $k > 0$ 。本章节中的时序图假定在 256 位接口上，完成 (Completion) 包不存在跨接。在 [256 位接口的跨接选项](#) 中对跨接功能进行了描述。

在 Dword 对齐模式下，传输从 3 个描述符 Dword 开始，紧随其后即为有效载荷 Dword。包括描述符和有效载荷在内的完整 TLP 作为单一 AXI4-Stream 数据包进行传输。当有效载荷长度超过 2 个 Dword 时，有效载荷内的数据始终为连续字节数据流。这样即可根据请求完成描述符的“Lower Address”（下位地址）字段和“Byte Count”（字节计数）字段来判定有效载荷的第一个 Dword 内的第一个有效字节的位置以及最后一个 Dword 的最后 1 个有效字节的位置。当有效载荷大小不超过 2 个 Dword 时，有效载荷中的有效字节不得连续。在此类情况下，用户应用必须存储与通过 RQ 接口向外发送的每个请求关联的“First Byte Enable”（首字节使能）字段和“Last Byte Enable”（末字节使能）字段，并将其用于判定完成有效载荷内的有效字节。用户应用可以选择使用 `m_axis_rc_tuser` 总线中的字节使能输出 `byte_en[31:0]` 来判定有效载荷中的有效字节，在连续有效载荷和非连续有效载荷情况下都是如此。

集成块会在数据包的整个持续时间段保持 `m_axis_rc_tvalid` 信号处于断言有效状态。用户应用可以随时通过断言 `m_axis_rc_tready` 无效来延长任一节拍。AXI4-Stream 接口可发出 `m_axis_rc_tkeep` 信号（每个信号对应 1 个 Dword 位置）以指示数据包中的有效 Dword，包括描述符和描述符与有效载荷之间插入的任意空字节。即，从描述符的第一个 Dword 开始到有效载荷的最后一个 Dword 为止的所有 `tkeep` 位均连续设置为 1。在数据包传输期间，当数据包无法填满接口的完整宽度时，`tkeep` 位仅限在数据包的最后一拍内才能设为 0。在数据包的最后一拍内，`m_axis_rc_tlast` 信号始终断言有效。

`m_axis_rc_tuser` 总线可提供多个参考信号，这些参考信号可用于简化与接口的用户应用侧关联的逻辑，或者用于支持其它功能。在每个数据包的第一拍中，`is_sof_0` 信号均断言有效（前提是其描述符位于总线上）。字节使能输出 `byte_en[31:0]`（每个输出对应 1 个字节通道）可用于指示有效载荷中的有效字节。仅当对应通道内包含的有效载荷字节确实有效时，这些信号才会断言有效，而针对描述符或空字节，则不会断言该信号有效。断言有效的字节使能位从有效载荷开始始终连续，除非有效载荷大小不超过 2 个 Dword。当完成包的有效载荷不超过 2 个 Dword 时，`byte_en` 上值为 1 的位可能不连续。另一种特殊情况是对于长度为 0 的存储器读取，集成块会传输含单 Dword 的有效载荷，其中所有 `byte_en` 位均设为 0。因此，无论在任何情况下，用户逻辑均可直接使用 `byte_en` 信号来启用将关联字节写入存储器的操作。

`m_axis_rc_tuser` 总线内的 `is_sof_1` 信号、`is_eof_0[3:0]` 信号和 `is_eof_1[3:0]` 信号不得用于 64 位接口和 128 位接口，当未启用跨接选项时，这些信号也不得用于 256 位接口。

图 62：请求器完成接口上含数据完成包的传输（Dword 对齐模式、64 位接口）

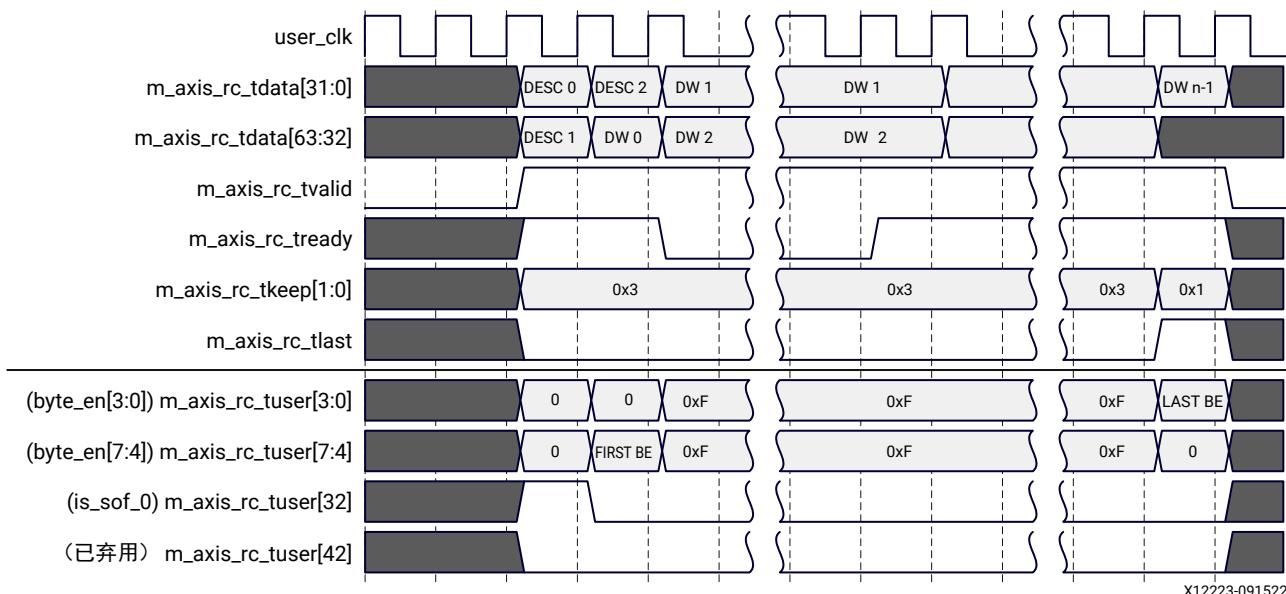
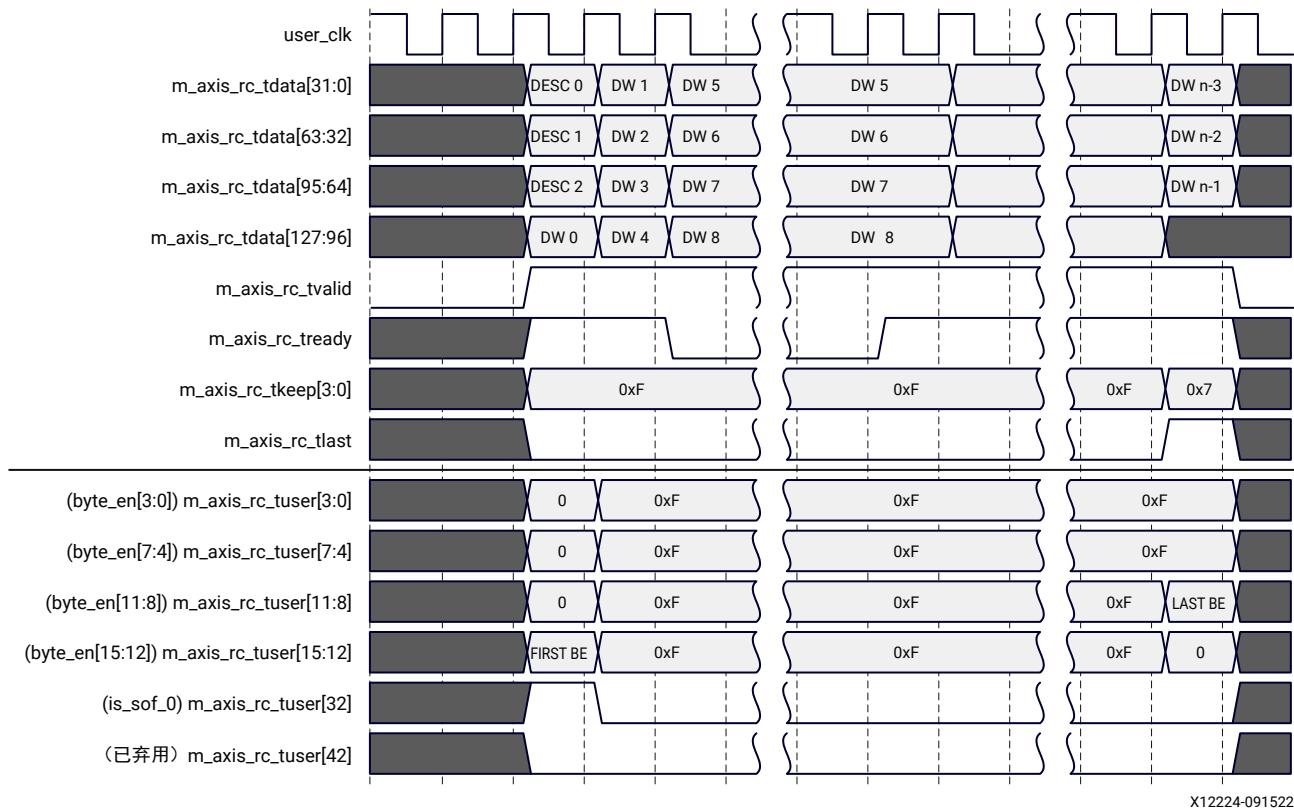
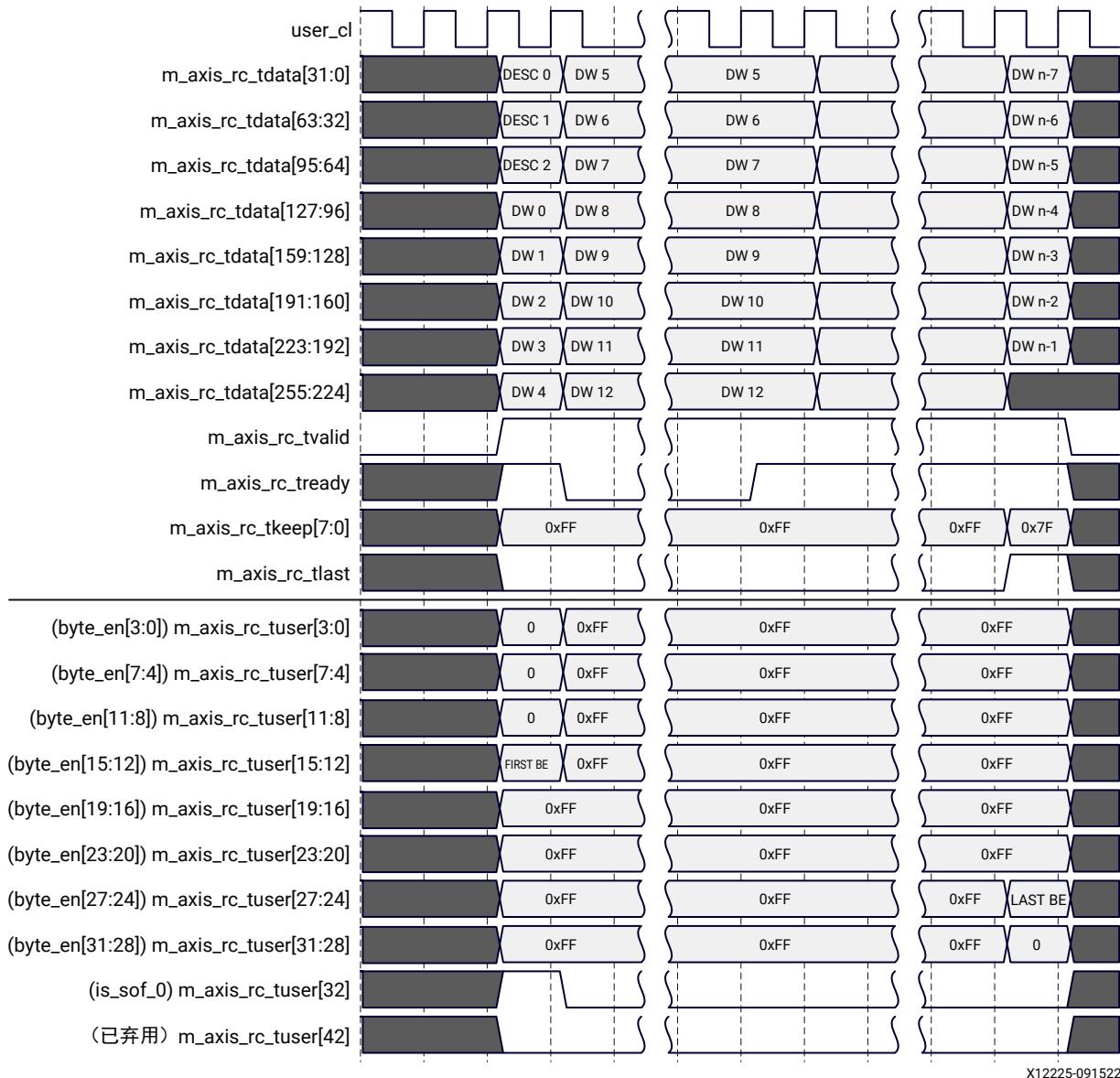


图 63：请求器完成接口上含数据完成包的传输（Dword 对齐模式、128 位接口）



X12224-091522

图 64：请求器完成接口上含数据完成包的传输（Dword 对齐模式、256 位接口）



以下时序图显示了当接口宽度分别配置为 64 位、128 位和 256 位时，从含关联有效载荷的链路接收到完成 TLP（采用地址对齐模式）后，通过 RC 接口来进行传输的过程。在时序图示例中，要传输的数据块的起始 Dword 地址（如描述符的下位地址字段的位 [6:2] 中所述）假定为 $(m \times 8 + 1)$ ，其中 m 为整数。数据块的大小假定为 n 个 Dword，其中， $n = k \times 32 + 28$ 且 $k > 0$ 。针对地址对齐传输，跨接选项无效，因此时序图假定在 256 位接口上完成包不存在跨接。

在地址对齐模式下，有效载荷的交付始终从描述符的最后一个字节后的节拍中开始。根据有效载荷的第 1 个有效字节的地址，有效载荷的第 1 个字节可显示在任意字节通道上。从描述符的第一个 Dword 开始到有效载荷的最后一个 Dword 为止的所有 tkeep 位均连续设置为 1。数据总线上第一个 Dword 的对齐方式是根据当用户应用向集成块发送请求时，请求器请求接口的 addr_offset[2:0] 输入的设置来确定的。用户应用可以选择使用字节使能输出 byte_en[31:0] 来判定有效载荷中的有效字节。

图 65：请求器完成接口上含数据完成包的传输（地址对齐模式、64 位接口）

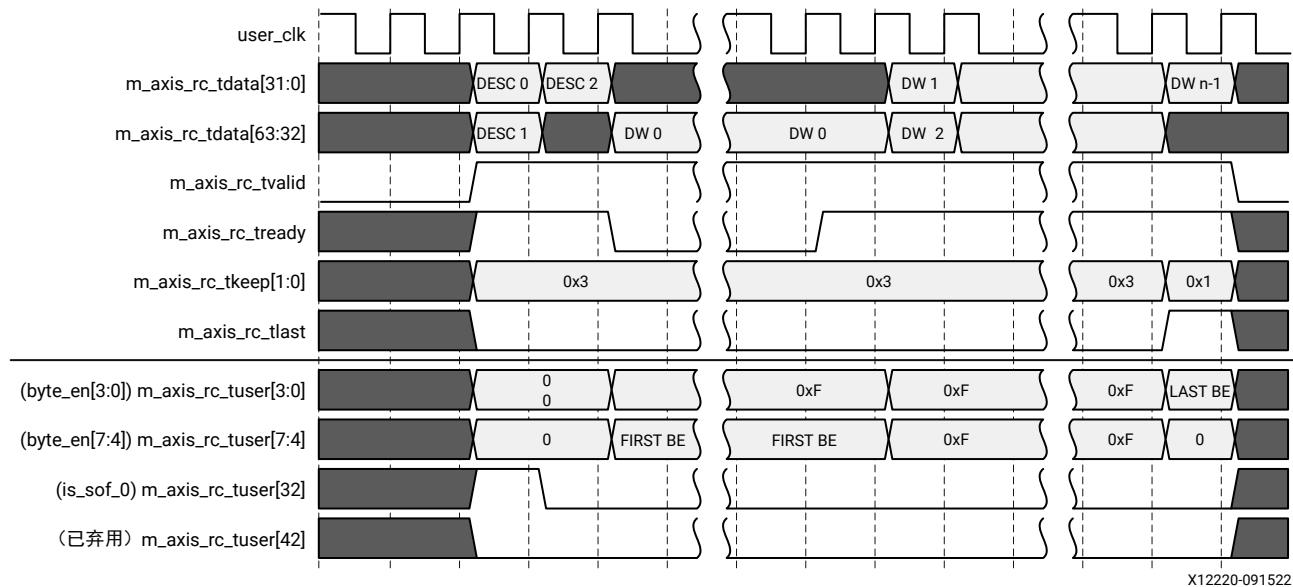


图 66：请求器完成接口上含数据完成包的传输（地址对齐模式、128 位接口）

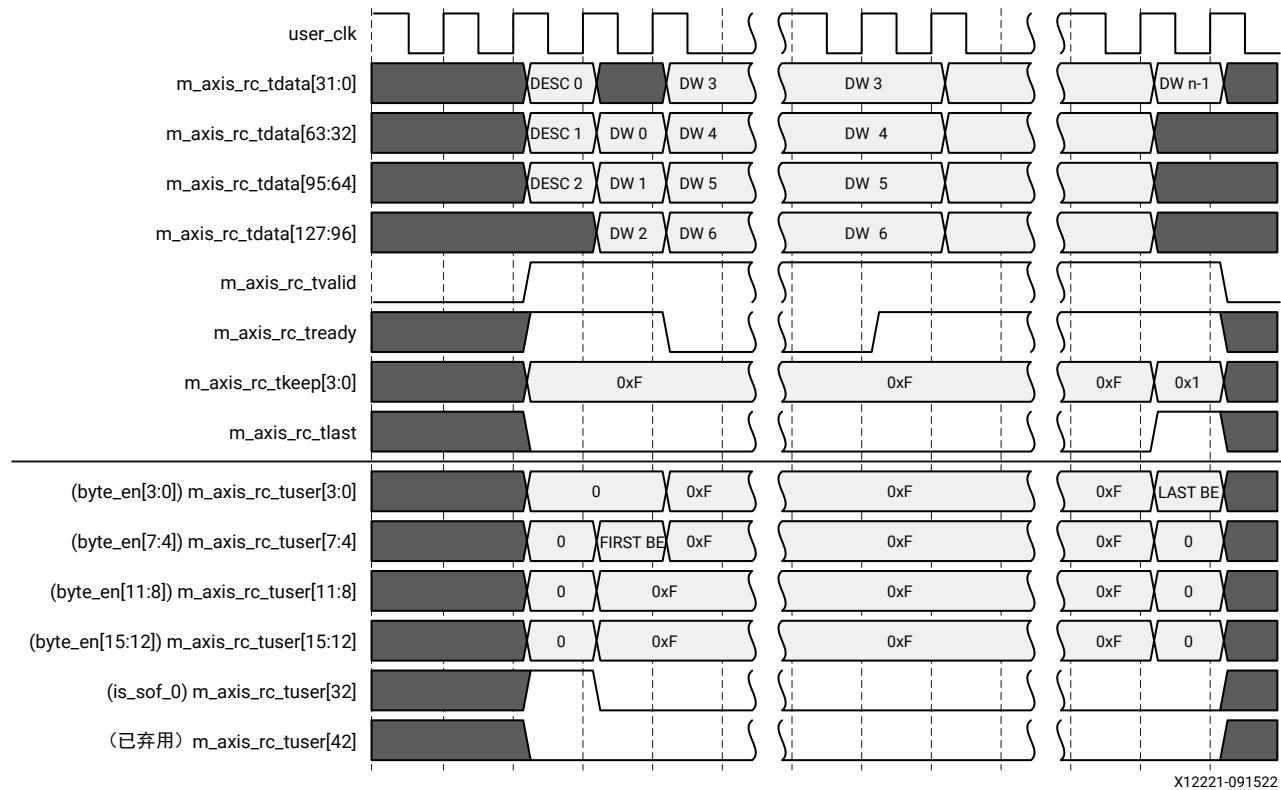
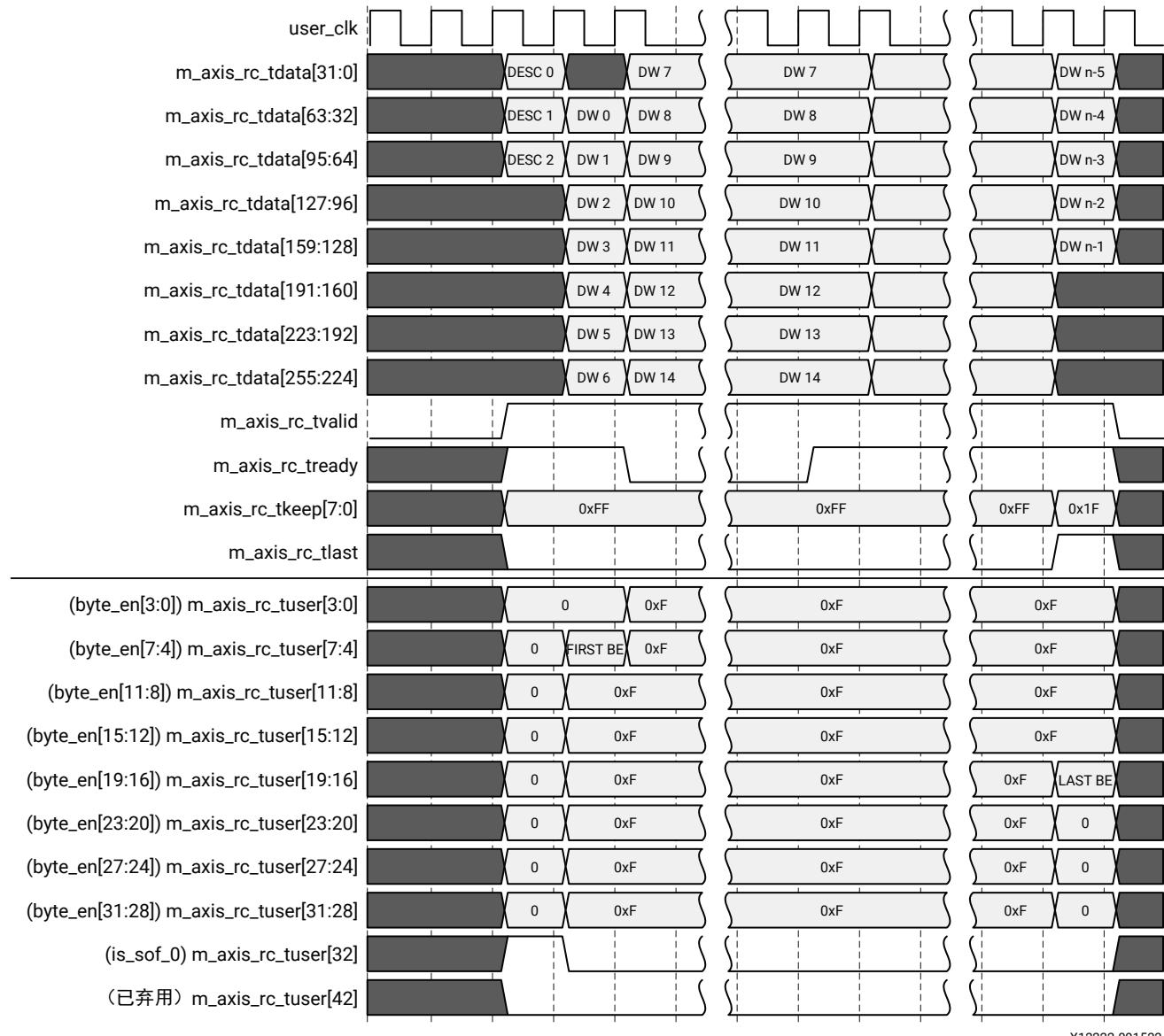


图 67：请求器完成接口上含数据完成包的传输（地址对齐模式、256 位接口）



X12222-091522

256 位接口的跨接选项

接口宽度配置为 256 位的情况下，当上一个完成包终止于数据总线上的 Dword 位置 3 或者在此位置前终止时，集成块可在同一拍内在 RC 接口上启动新的完成 (Completion) 包传输。跨接选项只能配合 Dword 对齐模式一起使用。

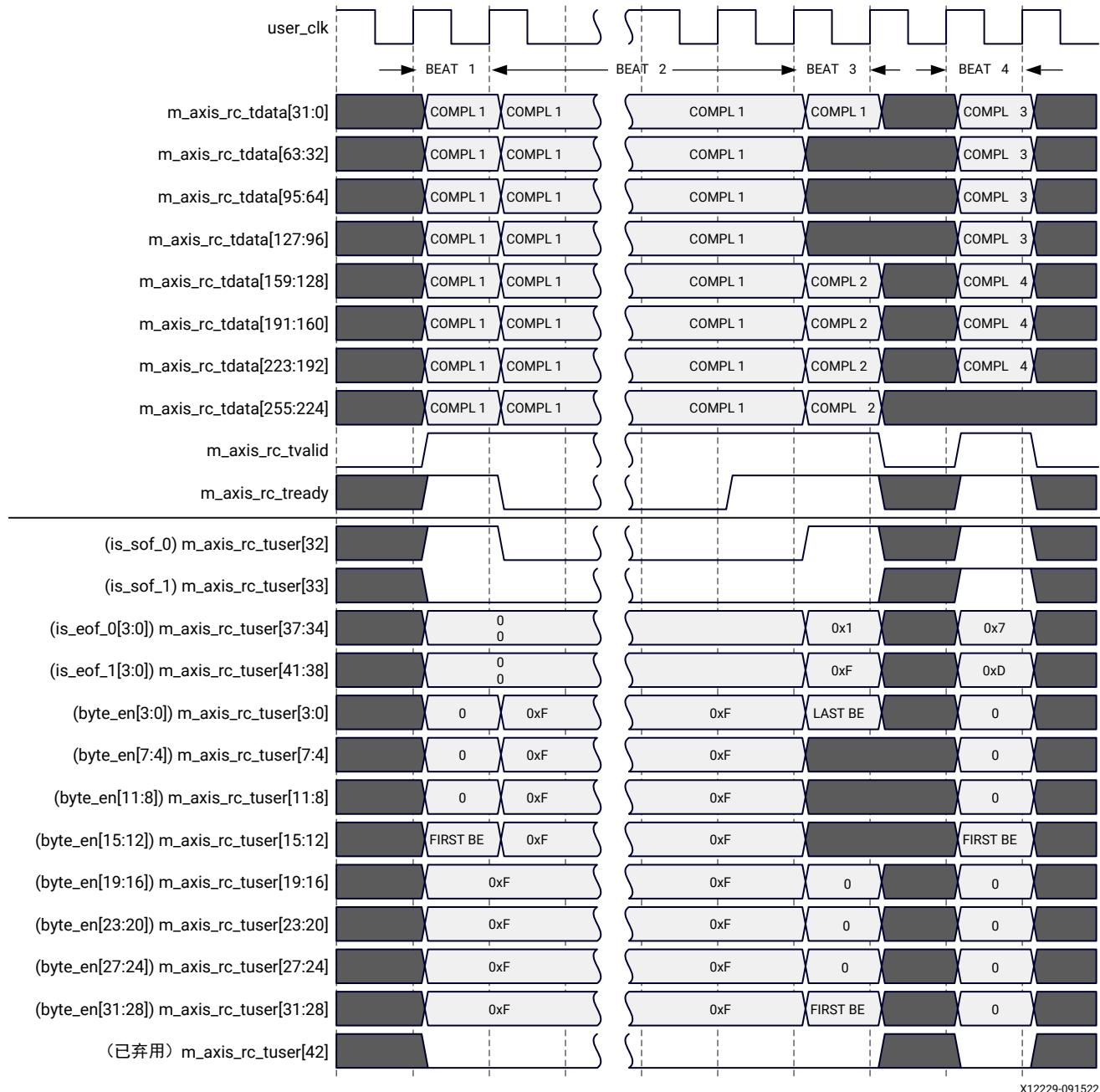
当启用跨接选项时，在 RC 接口上，完成 TLP 将作为无数据包边界（从 AXI4-Stream 角度而言）的连续数据流进行传输。因此，在判定接口上交付的完成 TLP 的边界过程中，不使用 `m_axis_rc_tkeep` 信号和 `m_axis_rc_tlast` 信号（使用跨接选项时，集成块会永久性将 `m_axis_rc_tkeep` 的位全部设置为 1，将 `m_axis_rc_tlast` 全部设置为 0）。并改为使用 `m_axis_rc_tuser` 总线中所提供的以下信号来执行 TLP 的界定：

- `is_sof_0`: 当有至少 1 个完成 TLP 从节拍中开始时，在该节拍中，集成块会将此输出驱动至高电平有效。此完成 TLP 的第 1 个字节的位置判定方式为：
 - 如果前一个完成 TLP 在此节拍前结束，那么此完成 TLP 的第 1 个字节位于字节通道 0 中。

- 如果前一个 TLP 在此节拍继续，那么此完成 TLP 的第 1 个字节位于字节通道 16 中。仅当前一个 TLP 在当前节拍中结束时（即同时设置 `is_eof_0[0]` 时），才有可能出现此情况。
- `is_sof_1`: 当有至少 2 个完成 TLP 从节拍中开始时，在该节拍中，集成块会断言此输出有效。第 1 个 TLP 始终在字节位置 0 开始，第 2 个 TLP 则在字节位置 16 开始。仅当前一个 TLP 在同一拍内在字节位置 16 之前结束时，集成块才能在字节位置 16 启动第 2 个 TLP；即，仅当 `is_eof_0[0]` 也在同一拍内设置时才会如此。
- `is_eof_0[3:0]`: 这些输出用于指示完成 TLP 结束及其最后一个 Dword 在数据总线上的位置。断言位 `is_eof_0[0]` 有效表明至少有 1 个完成 TLP 在此节拍中结束。设置 `is_eof_0` 的位 0 时，位 [3:1] 可提供在此节拍中结束的 TLP 的最后一个 Dword 的偏移。最后一个字节的偏移可根据 TLP 的起始地址和长度来确定，或者根据字节使能信号 `byte_en[31:0]` 来确定。当有 2 个完成 TLP 在一拍内结束时，`is_eof_0[3:1]` 即设置为第 1 个完成 TLP 的最后一个 Dword 的偏移（在此情况下，其范围为 0 - 3）。
- `is_eof_1[3:0]`: 断言 `is_eof_1[0]` 有效表明第 2 个 TLP 在同一拍中结束。设置 `is_eof_1` 的位 0 时，位 [3:1] 可提供在此节拍中结束的第 2 个 TLP 的最后一个 Dword 的偏移。由于第 2 个 TLP 只能在字节通道 16 上起始，因此只能在 27 - 31 范围内的字节通道上结束。因此，偏移 `is_eof_1[3:1]` 只能采用以下 2 个值中的任一值：6 或 7。如果 `is_sof_1[0]` 为高电平有效，那么 `is_eof_0[0]` 信号和 `is_eof_1` 信号在同一拍内始终为高电平有效。如果 `is_sof_1` 为高电平有效，那么 `is_eof_0` 同样为高电平有效。如果 `is_eof_1` 为高电平有效，那么 `is_eof_0` 同样为高电平有效。

下图显示了启用跨接选项的情况下，256 位 RC 接口上 4 个完成 TLP 的传输过程。第 1 个完成 TLP (COMPL 1) 从节拍 1 的 Dword 位置 0 开始，并在节拍 3 的 Dword 位置 0 结束。第 2 个 TLP (COMPL 2) 从同一拍内的 Dword 位置 4 开始。此第 2 个 TLP 仅含 1 个有效载荷（含 1 个 Dword），因此同样在同一拍内结束。第 3 和第 4 个完成 TLP 完全在节拍 4 中传输，因为完成包 3 仅含 1 个有效载荷（含 1 个 Dword），而完成包 4 则不含任何有效载荷。

图 68：在启用跨接选项的请求器完成接口上执行完成 TLP 的传输



X12229-091522

完成包传输异常中止

对于包含关联有效载荷的任何完成包，集成块可以在传输的有效载荷中发出错误信号，方法是在数据包的最后一拍中将 `m_axis_rc_tuser` 总线中的 `discontinue` 信号断言有效。当集成块在读取其内部存储器中的数据时，如果检测到不可纠正的错误，则会发生此操作。当用户应用在数据包的最后一拍中检测到 `discontinue` 信号已断言有效时，必须丢弃整个数据包。这在集成块中也被视作为致命错误。

使用跨接选项时，集成块如果已断言 `discontinue` 有效并导致此节拍内结束的完成 TLP 异常中止，则不会在同一拍内启动第 2 个完成 TLP。

完成错误的处理

当从链路接收到完成 TLP 时，集成块会将其与拆分完成包表 (Split Completion Table) 中未完成的请求进行比对，以判定对应的请求，并将其报头中的字段与期望的值进行比对以检测是否存在任何错误状况。随后，集成块会发出信号，以 4 位错误代码形式将错误状况包含在完成描述符中，一并发送给用户应用。集成块还会通过在描述符中设置“Request Completed”（请求完成）位（位 30）来指示请求的最后完成状态。下表提供了各错误代码所表示的错误状况的定义。

表 56：错误代码的编码

错误代码	描述
0000	未检测到任何错误。
0001	从链路接收到的完成 TLP 已被毒化。用户应用应丢弃位于描述符之后的所有数据。此外，如果在描述符中未设置请求完成位，那么用户应用应继续丢弃于此标签完成之后接收的数据，直至它接收到已设置请求完成位的完成描述符为止。接收到已设置请求完成位的完成描述符后，用户应用即可移除对应请求的所有状态。
0010	因出现状态为 UR、CA 或 CRS 的完成 TLP，请求已被终止。在此情况下，不存在与完成包关联的数据，并且完成描述符中已设置请求完成位。从集成块接收到此类完成后，用户应用即可丢弃对应请求。
0011	因完成 TLP 字节计数不正确，此读取请求已被终止。当接收到的完成 TLP 的字节计数与期望的计数不匹配时，就会发生此状况。完成描述符中已设置请求完成位。从集成块接收到此类完成后，用户应用即可丢弃对应请求。
0100	此代码对应状况如下：当前交付的完成包所含标签与未完成请求的标签相同，但其 Requester ID、TC 或 Attr 字段与未完成请求的参数不匹配。用户应用应丢弃位于描述符之后的所有数据。此外，如果在描述符中未设置请求完成位，那么用户应用应继续丢弃于此标签完成之后接收的数据，直至它接收到已设置请求完成位的完成描述符为止。接收到已设置请求完成位的完成描述符后，用户应用即可移除与该请求关联的所有状态。
0101	起始地址中存在错误。完成 TLP 报头中的下位地址位与请求的下一个期望字节的起始位置不匹配。用户应用应丢弃位于描述符之后的所有数据。此外，如果在描述符中未设置请求完成位，那么用户应用应继续丢弃于此标签完成之后接收的数据，直至它接收到已设置请求完成位的完成描述符为止。接收到已设置请求完成位的完成描述符后，用户应用即可丢弃对应请求。
0110	标签无效。此错误代码指示完成 TLP 中的标签与任意未完成请求的标签都不匹配。用户应用应丢弃位于描述符之后的所有数据。
1001	请求已终止，完成包超时。当未完成的请求超时且未从链路接收到完成包时，会使用此错误代码。集成块会为每个未完成的请求保留完成定时器，并通过在请求器完成接口上向用户应用发射虚拟完成描述符来响应完成超时，因此，用户应用可以终止暂挂的请求或者重试该请求。由于此描述符不对应于从链路接收到的完成 TLP，因此，在此描述符中只有请求完成位（位 30）、标签字段（位 [71:64]）和请求器功能字段（位 [55:48]）有效。
1000	因出现以生成请求的功能为目标的功能级别复位 (FLR)，此请求已被终止。在此情况下，集成块会在请求器完成接口上向用户应用发射虚拟完成描述符，因此用户应用可终止暂挂请求。由于此描述符不对应于从链路接收到的完成 TLP，因此，在此描述符中只有请求完成位（位 30）、标签字段（位 [71:64]）和请求器功能字段（位 [55:48]）有效。

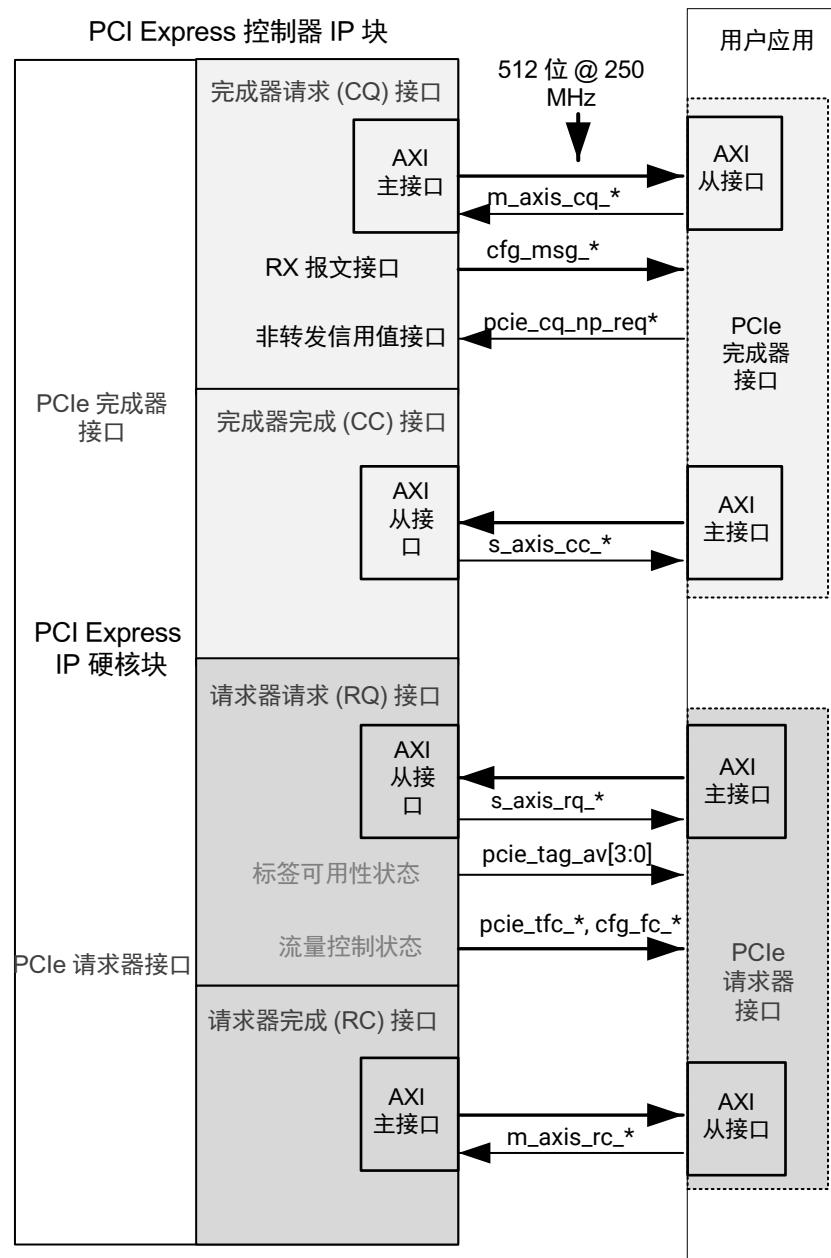
当标签由集成块进行内部管理时，集成块中的逻辑可确保在接收到对应请求的所有完成包或者当请求超时之后，才能复用分配到暂挂请求的标签。

但当标签由用户应用管理时，用户应用必须确保先由集成块通过在完成描述符中设置请求完成位的方式来发送请求终止信号，然后才能复用分配给请求的标签。用户应用可在接收到含非 0 错误代码的完成时结束暂挂请求，但如果在完成描述符中未设置请求完成位，则不应释放关联标签以作他用。当请求接收到多个拆分完成包，并且其中任一信号包含错误时，会发生此状况。在此情况下，集成块可以继续接收暂挂请求的完成 TLP，即使检测到错误也是如此，并且如果过早重新分配该请求的标签，则会导致这些完成包与其它请求出现匹配错误。在某些情况下，集成块可能必须等待请求超时后才能允许复用标签，即使已接收到含错误的拆分完成包也是如此。

512 位完成器接口

本章节旨在描述与 512 位 AXI4-Stream 接口关联的用户侧接口中的完成器接口的操作。下图显示了软核桥接、PCIe® 核和用户应用之间的连接。软核桥接将 500 MHz 的 256 位数据包转换为 250 MHz 的 512 位数据包。

图 69：含软核桥接的 PCIe IP 核的原理框图



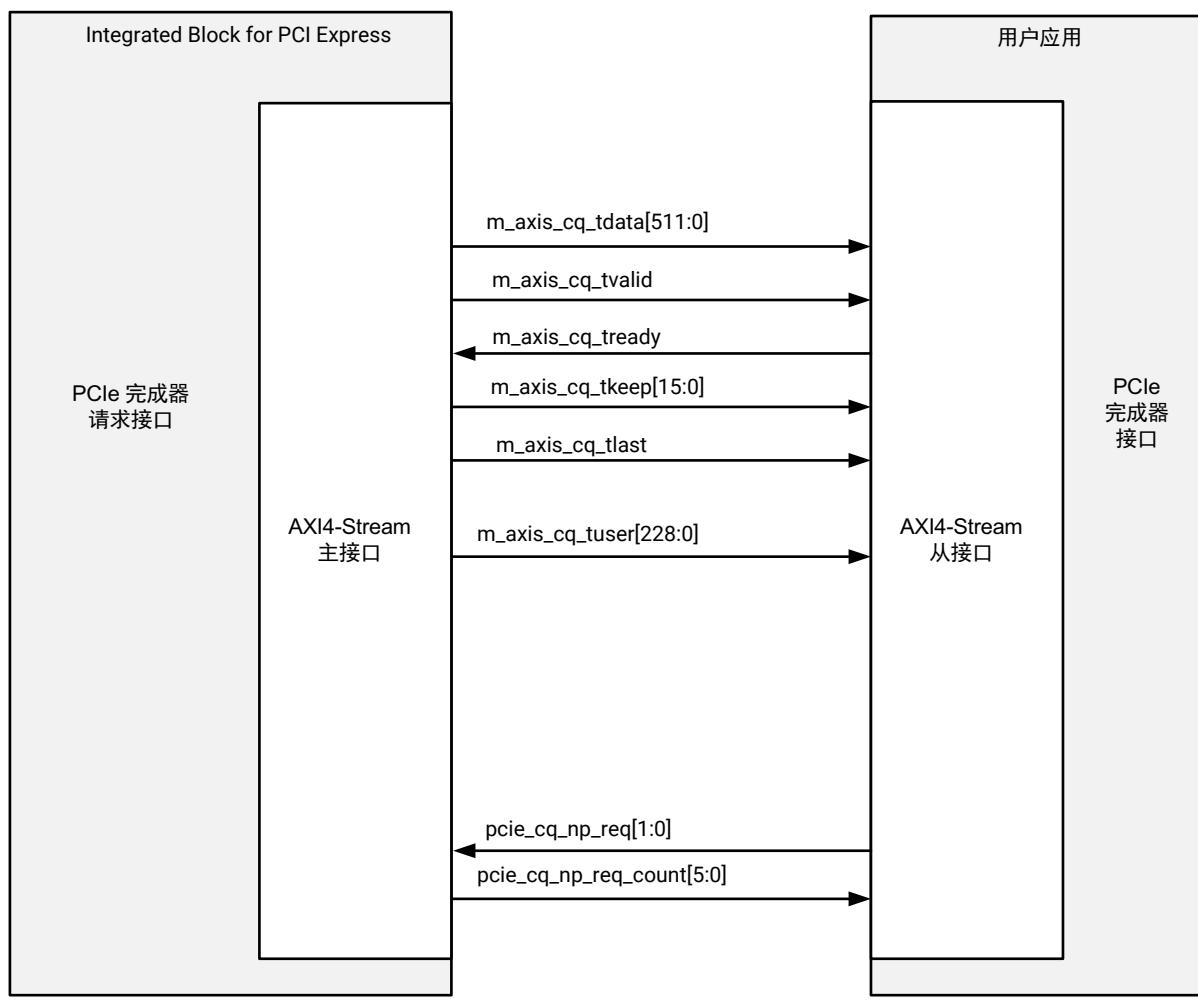
X16756-030223

完成器接口可基于 AXI4-Stream 协议，将从 PCIe 链路接收到的传输事务（存储器、I/O 读写、报文、原子操作）映射到完成器请求接口上的传输事务。完成器接口需连接到所有 PCIe 端点实现中的用户应用，但此接口对于根联合体则为可选。完成器接口包含 2 个独立接口，对应每个方向的数据传输使用 1 个接口。每个接口都基于 AXI4-Stream 协议，数据宽度为 512 位。完成器请求接口用于将请求（含任意关联的有效载荷数据）传输到用户应用，完成器完成接口则用于从用户应用接收完成包数据（针对非转发请求）以供通过链路进行转发。这 2 个接口均单独运行。即，核可通过完成器请求接口传输新请求，同时接收前一个请求的完成包。

完成器请求接口操作（512 位）

下图显示了与核的完成器请求接口关联的信号。核将此接口上的每个 TLP 均作为 1 个 AXI4-Stream 数据包来交付。对于含有效载荷的 TLP，此数据包以 128 位描述符开头并后接数据。

图 70：完成器请求接口信号



完成器请求接口支持 2 种不同的数据对齐模式，在 Vivado® IDE 中执行核自定义期间可选择模式。在 Dword 对齐模式下，有效数据的第一个字节显示在通道 $n = S + 16 + (A \bmod 4) \bmod 64$ 中，其中 A 是要传输的数据块的字节级别起始位置，S 是显示描述符的第一个字节的通道编号。对于报文和配置请求，地址 A 取 0。不使用跨接选项时，起始通道编号 S 始终为 0，但启用跨接选项时编号可为 0 或 32。

在 128 位地址对齐模式下，512 位总线上的有效载荷的起始位置始终在 128 位边界上对齐。但 512 位总线上的描述符的起始位置始终仅对齐到字节 0 或字节 32。对应于有效载荷的第 1 个字节的字节偏移判定方式为 $n = (S + 16 + (A \bmod 16)) \bmod 64$ ，其中， S 是显示描述符的第 1 个字节的字节偏移（可为 0 或 32），而 A 则是对应于有效载荷的第 1 个字节的存储器或 I/O 地址。这意味着，如果描述符从字节 0 开始，那么有效载荷可从以下 4 字节通道中的任一通道开始：16、20、24 和 28；或者如果描述符从字节 32 开始，那么有效载荷可从以下 4 字节通道中的任一通道开始：48、52、56 和 60。

描述符的结束位置与有效载荷的第 1 个字节的起始位置之间的间隔全部以空字节来填充。

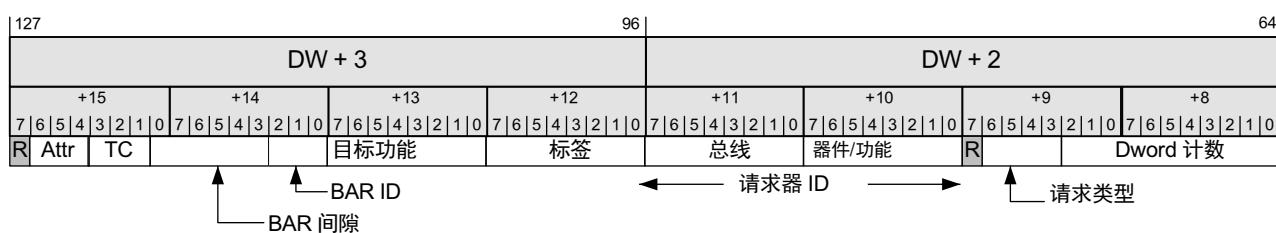
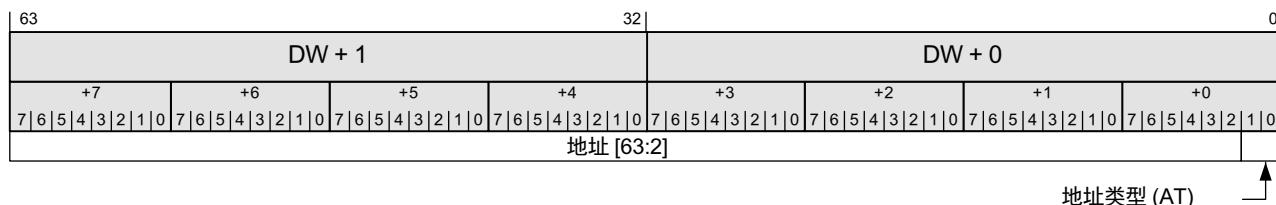
该接口还支持跨接选项，此选项允许在同一拍内跨接口传输最多 2 个 TLP。跨接选项只能配合 Dword 对齐模式一起使用，使用 128 位地址对齐模式时，不支持跨接选项。后续章节中的描述假定每拍 1 个 TLP。[CQ 接口上的跨接选项](#) 中描述了启用跨接选项的接口的操作。

完成器请求描述符格式

核将从链路接收到的每个请求 TLP 作为独立 AXI4-Stream 数据包通过完成器请求接口进行传输。每个数据包均以 1 个描述符开头，在描述符后可包含有效载荷数据。描述符长度始终为 16 字节，并在请求包的前 16 字节内发送。描述符始终在 512 位接口的第一拍内进行传输。下图演示了不同类型的请求的描述符格式。

当所传输的请求 TLP 为存储器读取/写入请求、I/O 读取/写入请求或原子操作 (Atomic Operation) 请求时，即适用下图所示格式。

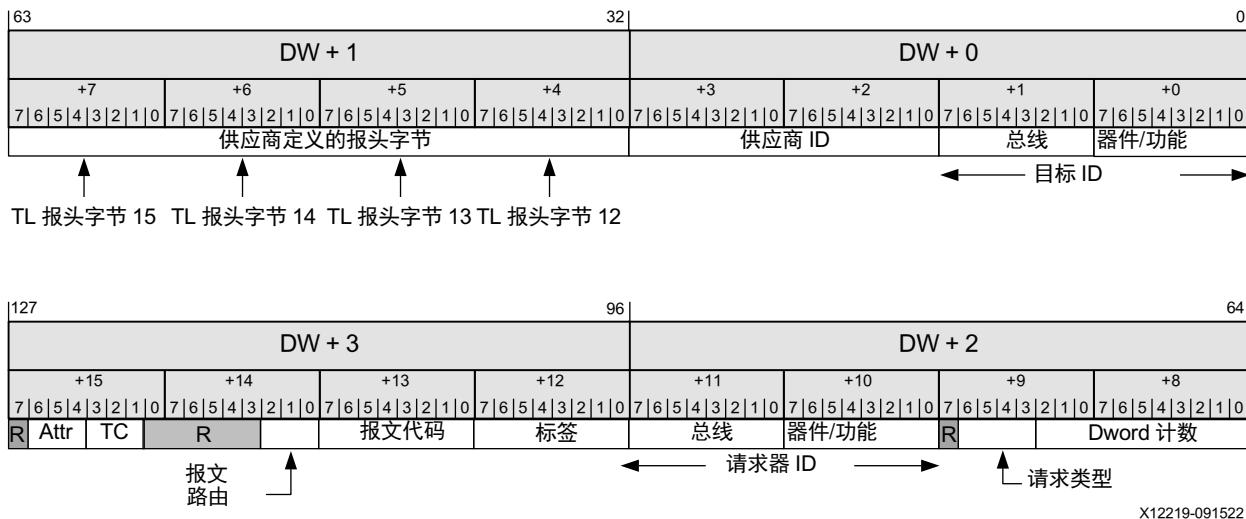
图 71：对应存储器、I/O 和原子操作请求的完成器请求描述符格式



X12217-091522

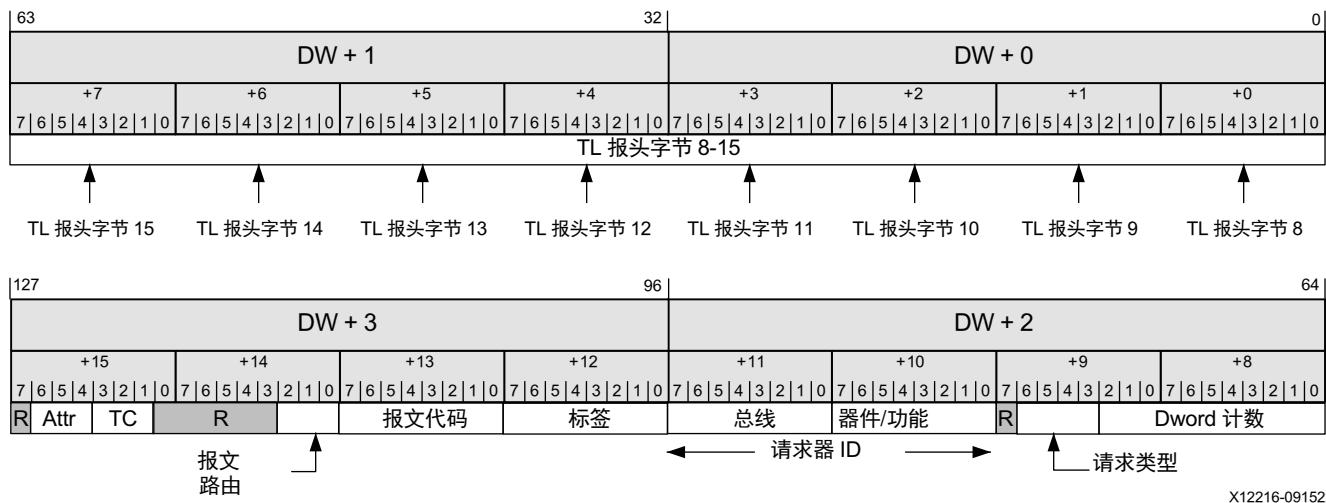
下图格式仅适用于类型 0 或类型 1 的“Vendor-Defined Messages”（供应商定义的报文）。

图 72：对应于供应商定义的报文的完成器请求描述符



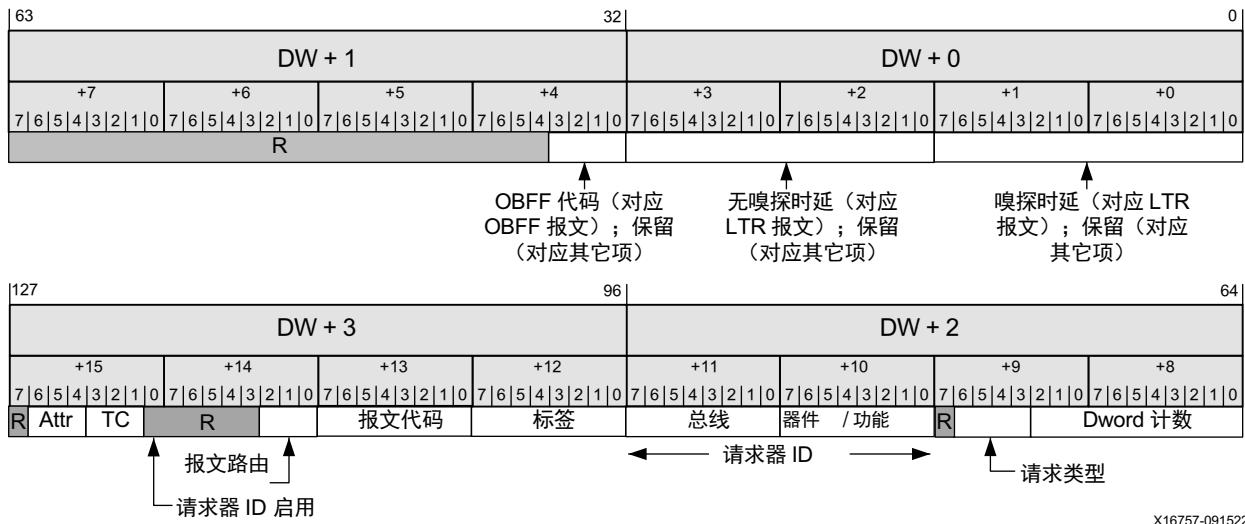
下图所示格式适用于所有 ATS 报文，包括“Invalid Request”（无效请求）、“Invalid Completion”（无效完成）、“Page Request”（页面请求）、“PRG Response”（PRG 响应）。

图 73：对应于 ATS 报文的完成器请求描述符



对于所有其它报文，描述符采用下图所示格式。

图 74：对应于所有其它报文的完成器请求描述符格式



X16757-091522

表 57：完成器请求描述符字段

位索引	字段名称	描述
1:0	“Address Type” (地址类型)	该字段定义仅适用于存储器传输事务和原子操作。其中包含从请求的 TL 报头抽取的 AT 位。 <ul style="list-style-type: none"> · 00: 请求中的地址未经转换 · 01: 传输事务为转换请求 · 10: 请求中的地址为已转换的地址 · 11: 保留
63:2	“Address” (地址)	该字段适用于存储器、I/O 和原子操作请求。它可提供来自 TL 报头的地址。这是请求所引用的首个 Dword 的地址。必须使用来自 m_axis_cq_tuser 的 First_BE 位来判定字节级别的地址。当此传输事务指定 32 位地址时，此字段的位 [63:32] 为 0。
74:64	“Dword Count” (Dword 计数)	这 11 个位用于指示要读取或写入的块大小（以 Dword 为单位），对于报文，这些位指示报文有效载荷的大小。其范围为 0 - 256 个 Dword。对于 I/O 访问，Dword 计数始终为 1。 对于长度为 0 的存储器读取/写入请求，Dword 计数为 1，且 First_BE 位全部设置为 0。
78:75	“Request Type” (请求类型)	用于识别传输事务类型。 表 58：传输事务类型 中列出了传输事务类型及其编码。
95:80	“Requester ID” (请求器 ID)	与请求关联的 PCI 请求器 ID。在 RID 的传统解读中，这 16 个位分割为 8 位总线编号 [95:88]、5 位器件编号 [87:83] 和 3 位功能编号 [82:80]。启用 ARI 时，位 [95:88] 包含 8 位总线编号，而位 [87:80] 则提供功能编号。 当请求为非转发传输事务时，用户完成器应用必须存储该字段，并将其与完成数据一起重新提供给核。
103:96	“Tag” (标签)	与请求关联的 PCIe 标签。当请求为非转发传输事务时，用户完成器应用必须存储该字段，并将其与完成数据一起重新提供给核。针对存储器写入和报文，可忽略该字段。
111:104	“Target Function” (目标功能)	该字段定义仅适用于存储器、I/O 和原子操作请求。它可提供请求的目标功能编号（由 BAR 检查来判定）。使用 ARI 时，该字段的全部 8 位都有效。否则，仅限位 [106:104] 有效。

表 57：完成器请求描述符字段 (续)

位索引	字段名称	描述
114:112	BAR ID	<p>该字段定义仅适用于存储器、I/O 和原子操作请求。它可为请求中的地址提供匹配的 BAR 编号。</p> <ul style="list-style-type: none"> · 000 = BAR 0 (针对 VF 为 VF-BAR 0) · 001 = BAR 1 (针对 VF 为 VF-BAR 1) · 010 = BAR 2 (针对 VF 为 VF-BAR 2) · 011 = BAR 3 (针对 VF 为 VF-BAR 3) · 100 = BAR 4 (针对 VF 为 VF-BAR 4) · 101 = BAR 5 (针对 VF 为 VF-BAR 5) · 110 = 扩展 ROM 访问 <p>注释：在根端口 (RP) 模式下，BAR ID 始终为 000。</p> <p>对于 64 位传输事务，所提供的 BAR 编号为匹配的 BAR 对的下位地址 (即，0、2 或 4)。</p>
120:115	“BAR Aperture” (BAR 间隙)	<p>该 6 位字段定义仅适用于存储器、I/O 和原子操作请求。它可提供与请求匹配的 BAR 的间隙设置。此信息适用于判定在存储器或 I/O 空间寻址过程中用户将使用的位。例如，值为 12 表示匹配的 BAR 的间隙为 4K，因此，用户可以忽略地址的位 [63:12]。</p> <p>对于 VF BAR，此输出上提供的值基于 BAR 所覆盖的单一 VF 耗用的存储器空间。</p>
123:121	“Transaction Class (TC)” (传输事务类)	与请求关联的 PCIe 传输事务类 (TC)。当请求为非转发传输事务时，用户完成器应用必须存储该字段，并将其与完成数据一起重新提供给核。
126:124	“Attributes” (属性)	<p>这些位可提供与请求关联的 Attribute 位的设置。位 124 为 “No Snoop” (无嗅探) 位，位 125 则为 “Relaxed Ordering” (宽松排序) 位。位 126 为 “ID-Based Ordering” (基于 ID 排序) 位，只能针对存储器请求和报文进行设置。</p> <p>当请求为非转发传输事务时，用户完成器应用必须存储该字段，并将其与完成数据一起重新提供给核。</p>
114:112	“Message Routing” (报文路由)	该字段定义适用于所有报文。这些位可提供来自 TL 报头的 3 位路由 (Routing) 字段 r[2:0]。
15:0	“Destination ID” (目标 ID)	该字段仅适用于供应商定义的报文。当报文按 ID 进行路由 (即，当 “Message Routing” (报文路由) 字段为 010 二进制值) 时，该字段可提供报文的 “Destination ID” (目标 ID)。
63:32	“Vendor-Defined Header” (供应商定义的报头)	该字段仅适用于供应商定义的报文。它包含从 TL 报头的 Dword 3 抽取的字节。
63:0	“ATS Header” (ATS 报头)	该字段仅适用于 ATS 报文。它包含从 TL 报头的 Dword 2 和 3 抽取的字节。

表 58：传输事务类型

请求类型 (二进制)	描述
0000	存储器读取请求
0001	存储器写入请求
0010	I/O 读取请求
0011	I/O 写入请求
0100	存储器提取和添加请求
0101	存储器无条件交换请求
0110	存储器比较和交换请求
0111	锁定读取请求 (仅在传统器件中支持)

表 58：传输事务类型 (续)

请求类型 (二进制)	描述
1000	类型 0 配置读取请求 (仅限在请求器侧)
1001	类型 1 配置读取请求 (仅限在请求器侧)
1010	类型 0 配置写入请求 (仅限在请求器侧)
1011	类型 1 配置写入请求 (仅限在请求器侧)
1100	任意报文 (ATS 报文和供应商定义的报文除外)
1101	供应商定义的报文
1110	ATS 报文
1111	保留

完成器存储器写入操作

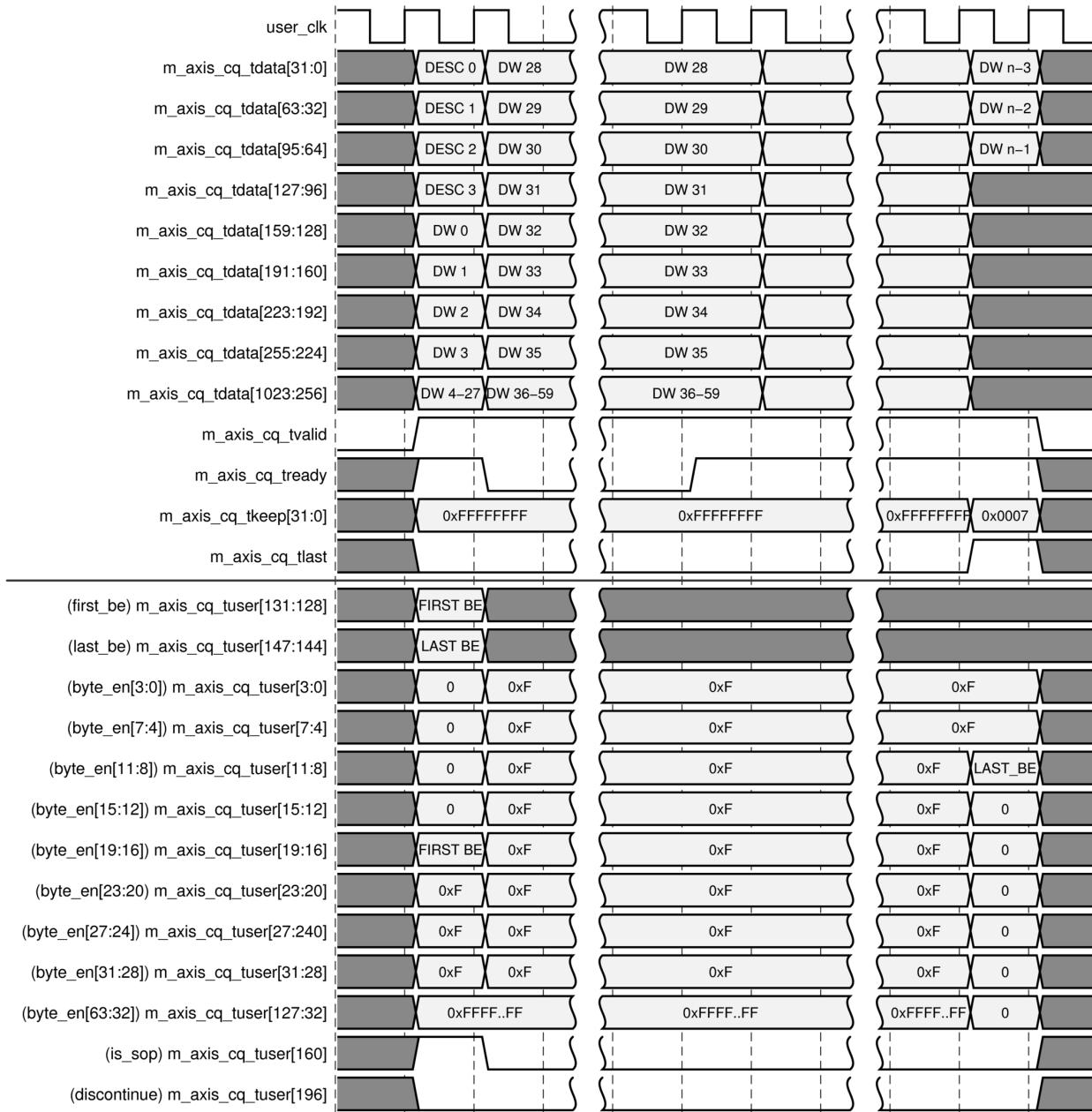
下图显示了从链路接收到存储器写入 TLP (采用 Dword 对齐模式) 后，通过完成器请求接口来进行传输的过程。为便于演示，写入用户存储器的数据块的起始 Dword 地址假定为 $(m * 16 + 3)$ ，其中 $m > 0$ 且为整数。其大小假定为 n 个 Dword，其中 $n = k * 16 - 1$ 且 $k > 1$ 。

传输从 16 个描述符字节开始，紧随其后即为有效载荷字节。在数据包持续时间段内，`m_axis_cq_tvalid` 信号保持断言有效。用户逻辑可以随时通过下拉 `m_axis_cq_tready` 来延长任一节拍。AXI4-Stream 接口可发出 `m_axis_cq_tkeep` 信号 (每个信号对应 1 个 Dword 位置) 以指示数据包中的有效 Dword，包括描述符和描述符与有效载荷之间插入的任意空字节。即，从描述符的第一个 Dword 开始到有效载荷的最后一个 Dword 为止的所有 `m_axis_cq_tkeep` 位均连续设置为 1。在数据包传输期间，当数据包无法填满接口的完整宽度时，`tkeep` 位仅限在数据包的最后一拍内才能设为 0。在数据包的最后一拍内，`m_axis_cq_tlast` 信号始终断言有效。

完成器请求接口在 `m_axis_cq_tuser` 总线中还包含“First Byte Enable”(首字节使能)位和“Last Byte Enable”(末字节使能)位。这些位在数据包的第一拍内激活，并提供有关有效载荷的第一个和最后一个 Dword 中的有效字节的信息。

`m_axis_cq_tuser` 总线还可提供多个可选信号，这些可选信号可用于简化与接口的用户侧关联的逻辑，或者用于支持其它功能。在每个数据包的第一拍内，`is_sop` 信号都断言有效 (前提是其描述符位于总线上)；当不使用跨接选项时，`m_axis_cq_tuser` 中的所有其它 sop 和 eop 都与请求信号传输无关。字节使能输出 `byte_en[63:0]` (每个输出对应 1 个字节通道) 可指示有效载荷中的有效字节。仅当对应通道内包含的有效载荷字节确实有效时，这些信号才会断言有效，而针对描述符或空字节，则不会断言该信号有效。断言有效的字节使能位从有效载荷开始始终连续，除非有效载荷大小不超过 2 个 Dword。当写入不超过 2 个 Dword 时，`byte_en` 上值为 1 的位不连续。

图 75：完成器请求接口上的存储器写入传输事务（Dword 对齐模式）



另一种特殊情况是对于长度为 0 的存储器写入，核会传输含单 Dword 的有效载荷，其中所有 byte_en 位均设为 0。因此，无论在任何情况下，用户逻辑均可直接使用 byte_en 信号来启用将关联字节写入存储器的操作。

在 Dword 对齐模式下，根据有效载荷的第一个有效字节的地址，在描述符的结束位置与有效载荷的第一个字节之间可能存在 0、1、2 或 3 个字节位置的间隔。有效载荷中第一个有效字节的实际位置可通过 m_axis_cq_tuser 总线中的 first_be[3:0] 或 byte_en[63:0] 来判定。

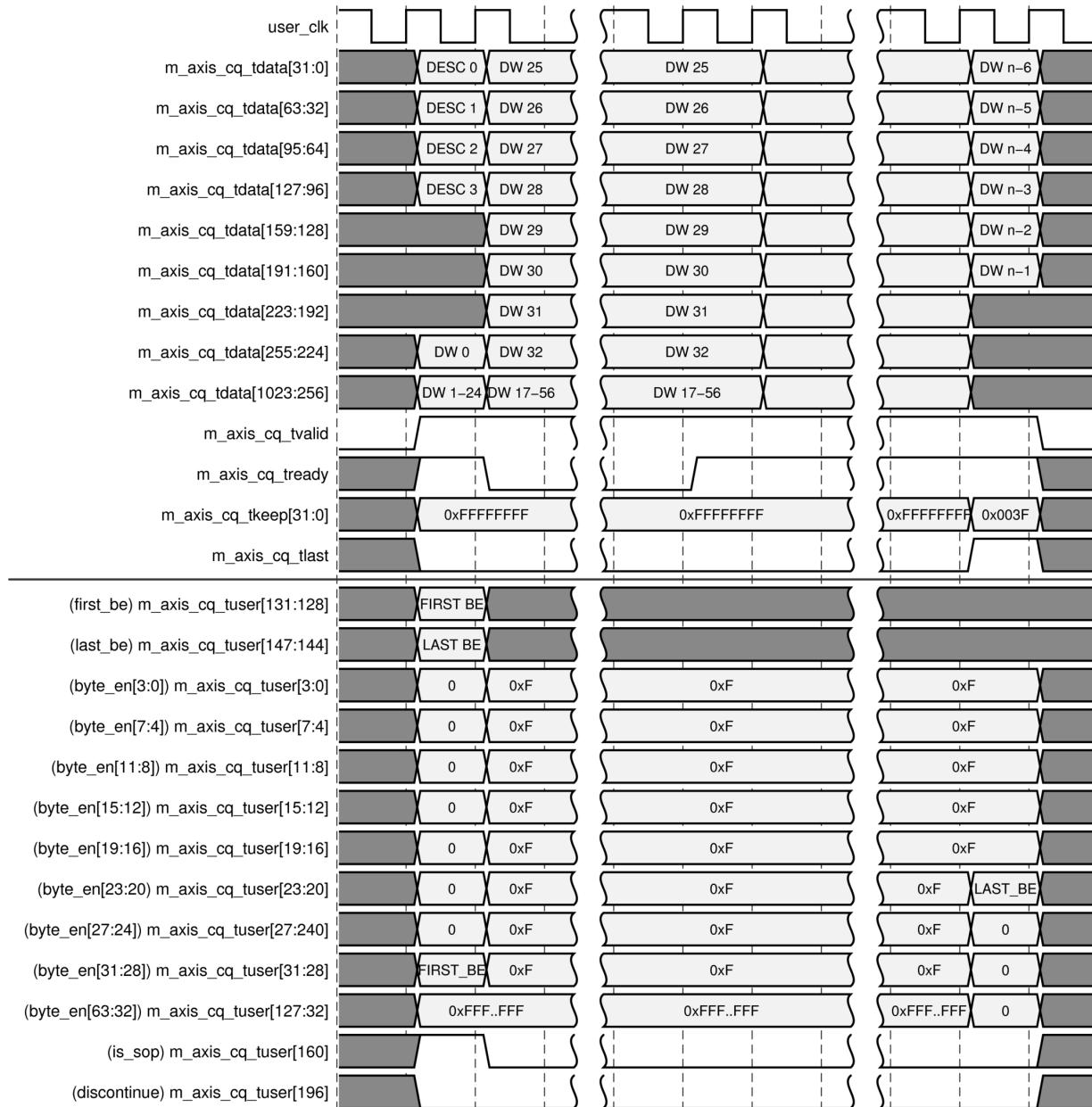
当接收到的 TLP 中存在“Transaction Processing Hint”（传输事务处理提示）时，核所传输的参数均为与 m_axis_cq_tuser 总线内的信号上的提示相关联的参数，提示分为：“TPH Steering Tag”（TPH 导向标签）和“Steering Tag Type”（导向标签类型）。

下图中的时序图显示了从链路接收到存储器写入 TLP（采用 128 位地址对齐模式）后，通过完成器请求接口来进行传输的过程。为便于演示，写入用户存储器的数据块的起始 Dword 地址假定为 $(m \cdot 16 + 3)$ ，其中 $m > 0$ 且为整数。其大小假定为 n 个 Dword，其中 $n = k \cdot 16 - 1$ 且 $k > 1$ 。

在地址对齐模式下，有效载荷的交付始终从第一拍的第 2 个四分之一拍（位 255:128）开始，紧随第 1 个四分之一拍的描述符之后。根据有效载荷的第一个有效 Dword 的地址，有效载荷的第一个 Dword 可显示在第 2 个四分之一拍的 4 个 Dword 位置中的任一位置。此 keep 输出的 `m_axis_cq_tkeep` 在描述符与有效载荷之间的间隙内保持处于高电平状态。有效载荷中的第 1 个有效字节的实际位置可根据描述符中地址的最低有效位或者根据 `m_axis_cq_tuser` 总线中的字节使能位 `byte_en[63:0]` 来判定。

如果写入不超过 2 个 Dword，那么从有效载荷开始，`byte_en` 上值为 1 的位不连续。对于长度为 0 的存储器写入，核会传输含单 Dword 的有效载荷，其中有效载荷字节的所有 `byte_en` 位均设为 0。

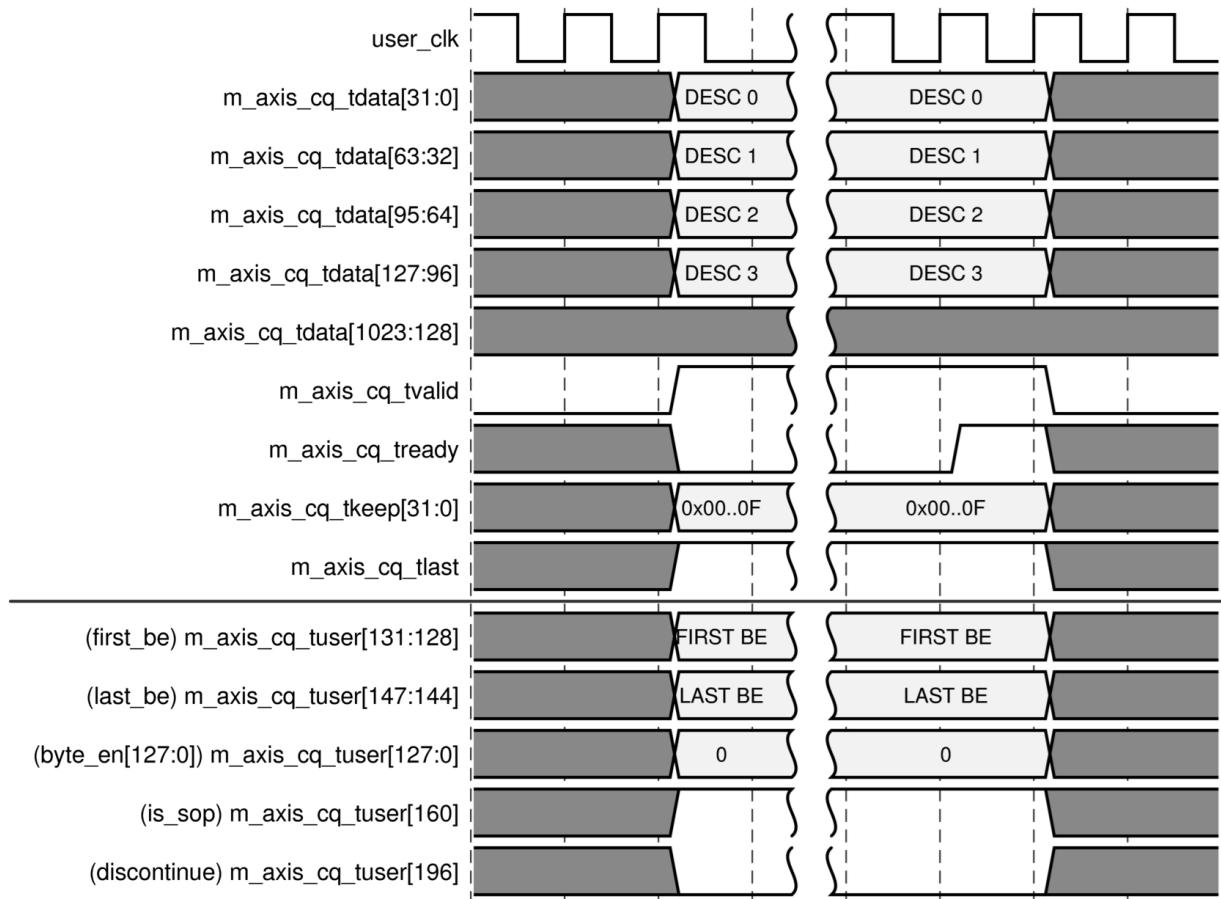
图 76：完成器请求接口上的存储器写入传输事务（128 位地址对齐模式）



完成器存储器读取操作

存储器读取请求跨完成器请求接口进行传输的顺序与存储器写入请求相同，除非 AXI4-Stream 数据包仅包含 16 字节描述符。下图显示了从链路收到存储器读取 TLP 后，通过完成器请求接口来进行传输的过程。此数据包在接口上传输只需一拍即可。在数据包持续时间段内，`m_axis_cq_tvalid` 信号保持断言有效。用户逻辑可以通过下拉 `m_axis_cq_tready` 来延长任一节拍。当第 1 个描述符字节位于总线上时，`m_axis_cq_tuser` 总线中的 `is_sop` 信号断言有效。

图 77：完成器请求接口上的存储器读取传输事务



与第一个和最后一个 Dword 的读取请求关联的字节使能位由核在 `m_axis_cq_tuser` 边带总线上提供。传输描述符时，这些位有效，并且必须供用户逻辑用于判定与该请求关联的字节级别起始地址和字节计数。对于 1 个 Dword 读取和 2 个 Dword 读取的特殊情况，字节使能可以不连续。字节使能在所有其它情况下均连续。长度为 0 的存储器读取在完成器请求接口上发送，其描述符中的“Dword count”（Dword 计数）字段设置为 1，且首字节使能和末字节使能均设为 0。

用户逻辑必须以完成包来响应每个存储器读取请求。由读取所请求的数据将作为单一完成包来发送，或者也可作为多个拆分完成包来发送。这些完成包必须发送到核的完成器完成接口。2 个不同请求的完成包可按任意顺序发送，但同一个请求的拆分完成包则必须按顺序发送。如需获取有关完成器完成接口操作的说明，请参阅 [64/128/256 位完成器接口](#) 和 [512 位完成器接口](#)。

I/O 写入操作

在完成器请求接口上传输 I/O 写入请求的过程与含单 Dword 的有效载荷的存储器写入请求的传输过程相似。传输以 128 位描述符开始，紧随其后即为含单 Dword 的有效载荷。当使用 Dword 对齐模式时，有效载荷 Dword 紧接在描述符之后。当使用 128 位地址对齐模式时，有效载荷 Dword 在位 255:128 中提供，其对齐方式则基于描述符中的地址来判定。`m_axis_cq_tuser` 中的首字节使能用于指示有效载荷中的有效字节。字节使能位 `byte_en` 同样可提供此信息。

因为 I/O 写入为非转发传输事务，用户逻辑必须使用不含数据有效载荷的完成包来作为其响应。I/O 请求的完成包可按任意顺序发送。通过将完成器描述符中的“Completion Status”（完成状态）字段设置为相应的 CA（完成器异常中止）或 UR（请求不受支持）即可向请求器发出信号，表示发生与 I/O 写入传输事务关联的错误。如需获取有关完成器完成接口操作的说明，请参阅 [64/128/256 位完成器接口](#) 和 [512 位完成器接口](#)。

I/O 读取操作

在完成器请求接口上传输 I/O 读取请求的过程与存储器读取请求的传输过程相似，且仅含描述符。请求的数据长度始终为 1 个 Dword，`m_axis_cq_tuser` 中的首字节使能位用于指示要读取的有效字节。

用户逻辑必须以单 Dword 完成包来响应 I/O 读取请求，或者如果发生错误，则以不含数据的完成包来响应。对应 2 个不同 I/O 读取请求的完成包可按任意顺序发送。通过将完成器描述符中的“Completion Status”（完成状态）字段设置为相应的 CA（完成器异常中止）或 UR（请求不受支持）即可向请求器发出信号，表示发生与 I/O 读取传输事务关联的错误。如需获取有关完成器完成接口操作的说明，请参阅 [64/128/256 位完成器接口](#) 和 [512 位完成器接口](#)。

完成器请求接口上的原子操作

在完成器请求接口上传输原子操作请求的过程与存储器写入请求的传输过程相似。原子操作的有效载荷范围为 1 到 8 个 Dword，其起始地址始终与 Dword 边界对齐。传输以 128 位描述符开始，紧随其后即为有效载荷。当使用 Dword 对齐模式时，第一个有效载荷 Dword 紧接在描述符之后。当使用 128 位地址对齐模式时，有效载荷从位 255:128 开始，其对齐方式则基于描述符中的地址来判定。`keep` 输出 `m_axis_cq_tkeep` 和 `m_axis_cq_tuser` 同样可指示有效载荷中的有效字节。首字节使能位和末字节使能位用于指示有效载荷的结束位置。不应使用 `m_axis_cq_tuser` 中的 `byte_en` 信号。

因为原子操作为非转发传输事务，用户逻辑必须使用含操作结果的完成包来作为其响应。通过将完成器描述符中的“Completion Status”（完成状态）字段设置为相应的 CA（完成器异常中止）或 UR（请求不受支持）即可向请求器发出信号，表示发生与此操作关联的错误。如需获取有关完成器完成接口操作的说明，请参阅 [64/128/256 位完成器接口](#) 和 [512 位完成器接口](#)。

完成器请求接口上的报文请求

在完成器请求接口上传输报文的过程与存储器写入请求的传输过程类似，但有时其中不存在有效载荷。传输以 128 位描述符开始，紧随其后即为有效载荷（如果存在）。有效载荷始终在字节通道 16 中开始，与使用的寻址模式无关。用户逻辑可以根据 `m_axis_cq_tlast` 信号和 `m_axis_cq_tkeep` 信号的状态来判定有效载荷的结束位置。`m_axis_cq_tuser` 中的 `byte_en` 信号还可指示有效载荷中的有效字节。`m_axis_cq_tuser` 中的“首字节使能”和“末字节使能”不应使用。

`ATTR_AXISTEN_IF_ENABLE_RX_MSG_INTFC` 属性必须设为 0 才能启用通过完成器请求接口交付报文的功能。当该属性设为 0 时，`ATTR_AXISTEN_IF_ENABLE_MSG_ROUTE` 属性可用于选择用户希望通过完成器请求接口交付的具体报文类型。将属性位设为 1 即可启用在该接口上交付对应报文类型的功能，将其设为 0 则会导致核将此报文过滤掉。

表 59：`AXISTEN_IF_ENABLE_MSG_ROUTE` 属性位描述

位索引	报文类型
0	<code>ERR_COR</code>
1	<code>ERR_NONFATAL</code>
2	<code>ERR_FATAL</code>
3	<code>Assert_INTA</code> 和 <code>Deassert_INTA</code>
4	<code>Assert_INTB</code> 和 <code>Deassert_INTB</code>
5	<code>Assert_INTC</code> 和 <code>Deassert_INTC</code>
6	<code>Assert_INTD</code> 和 <code>Deassert_INTD</code>
7	<code>PM_PME</code>

表 59：AXISTEN_IF_ENABLE_MSG_ROUTE 属性位描述 (续)

位索引	报文类型
8	PME_TO_Ack
9	PME_Turn_Off
10	PM_Active_State_Nak
11	Set_Slot_Power_Limit
12	时延容限报告 (LTR)
13	保留
14	解锁
15	Vendor_Defined 类型 0
16	Vendor_Defined 类型 1
17	无效请求、无效完成包、页面请求、PRG 响应

当 ATTR_AXISTEN_IF_ENABLE_RX_MSG_INTFC 设为 1 时，在完成器请求接口上不交付任何报文。改为通过专用接收报文接口发送已接收到报文的指示信息（请参阅 [接收报文接口](#)）。

传输异常中止

对于包含关联有效载荷的任何请求，可以在传输的有效载荷中向接口发出错误信号，方法是在数据包的最后一拍中将 m_axis_cq_tuser 总线中的 discontinue 信号随 m_axis_cq_tlast 一起断言有效。当核在读取其内部存储器中的数据时，如果检测到不可纠正的错误，则会发生此操作。当用户应用在数据包的最后一拍中检测到 discontinue 信号已断言有效时，必须丢弃整个数据包。在 discontinue 断言有效的节拍中，接口不会启动新数据包传输，即使启用跨接选项也是如此。

针对非转发请求的选择性流量控制

PCI Express® 规范要求完成器请求接口持续不断交付转发传输事务，即使用户逻辑无法在接口上接受非转发传输事务也是如此。为启用此功能，核会在完成器接口上实现基于信用值的流量控制机制，用户逻辑可通过此机制来控制通过此接口的非转发请求的流量，而不会影响转发请求。用户逻辑可使用 pcie_cq_np_req[1:0] 信号来向核示用于接收非转发请求的缓冲器的可用情况。仅当可用信用值为非 0 值时，核才会向用户逻辑交付非转发请求。当非转发请求交付因缺少信用值而暂停时，核仍会持续交付转发请求。当信用值机制对非转发请求的交付不施加任何反压时，核会按从链路接收到转发请求和非转发请求的顺序来交付这些请求。

核会保留内部信用值计数器，以跟踪完成器请求接口上可用于非转发请求的信用值。以下算法用于保持记录可用信用值：

- 在复位时，计数器设为 0。
- 当接口解复位后，在每个时钟周期内：
 - 如果 pcie_cq_np_req 为非 0 值并且在此周期内未交付任何非转发请求，那么信用值会递增 1，除非它已达到其饱和限值 12。当 pcie_cq_np_req = 2'b01 时，递增量为 1，当 pcie_cq_np_req = 2'b10 或 2'b11 时，递增量为 2。
 - 如果 pcie_cq_np_req = 2'b00 且在此周期内仅交付单个非转发请求，那么信用值递减 1，除非它已为 0。
 - 如果 pcie_cq_np_req = 2'b00 且在此周期内交付 2 个非转发请求，那么信用值递减 2，除非它已达到 0。
 - 在所有其它情况下，信用值均保持不变。
- 仅当信用值大于 0 时，核才会开始向用户逻辑交付非转发 TLP。

用户应用每次准备好接收非转发请求时，均可在 `pcie_cq_np_req` 上提供 1 或 2 个信用值，或者如果不需要对非转发请求应用选择性反压，那么也可以将其保留永久设置为 `2'b11`。如果信用值始终为非 0 值，那么核会按从链路接收到转发请求和非转发请求的顺序来交付这些请求。如果有时信用值保持为 0，那么非转发请求可在该核的 FIFO 中累积。当信用值后续变为非 0 值时，核会首先交付先于已交付到用户应用的转发请求到达并累积的非转发请求，然后核才会还原为按从链路接收到请求的顺序来交付这些请求。

`pcie_cq_np_req` 的设置无需与完成器请求接口上的包传输对齐。

用户应用可以监控 `pcie_cq_np_req_count[5:0]` 输出上的当前信用值。计数器饱和上限为 32。由于存在内部流水线延迟，当核在 `pcie_cq_np_req` 输入上接收到脉冲之后，可能需要经过多个延迟周期后，核才会在响应中更新 `pcie_cq_np_req_count` 输出。因此，当用户逻辑具有足够的缓冲器空间可用时，它应提前提供信用值，以便避免核应缺少信用值而延误交付非转发请求。

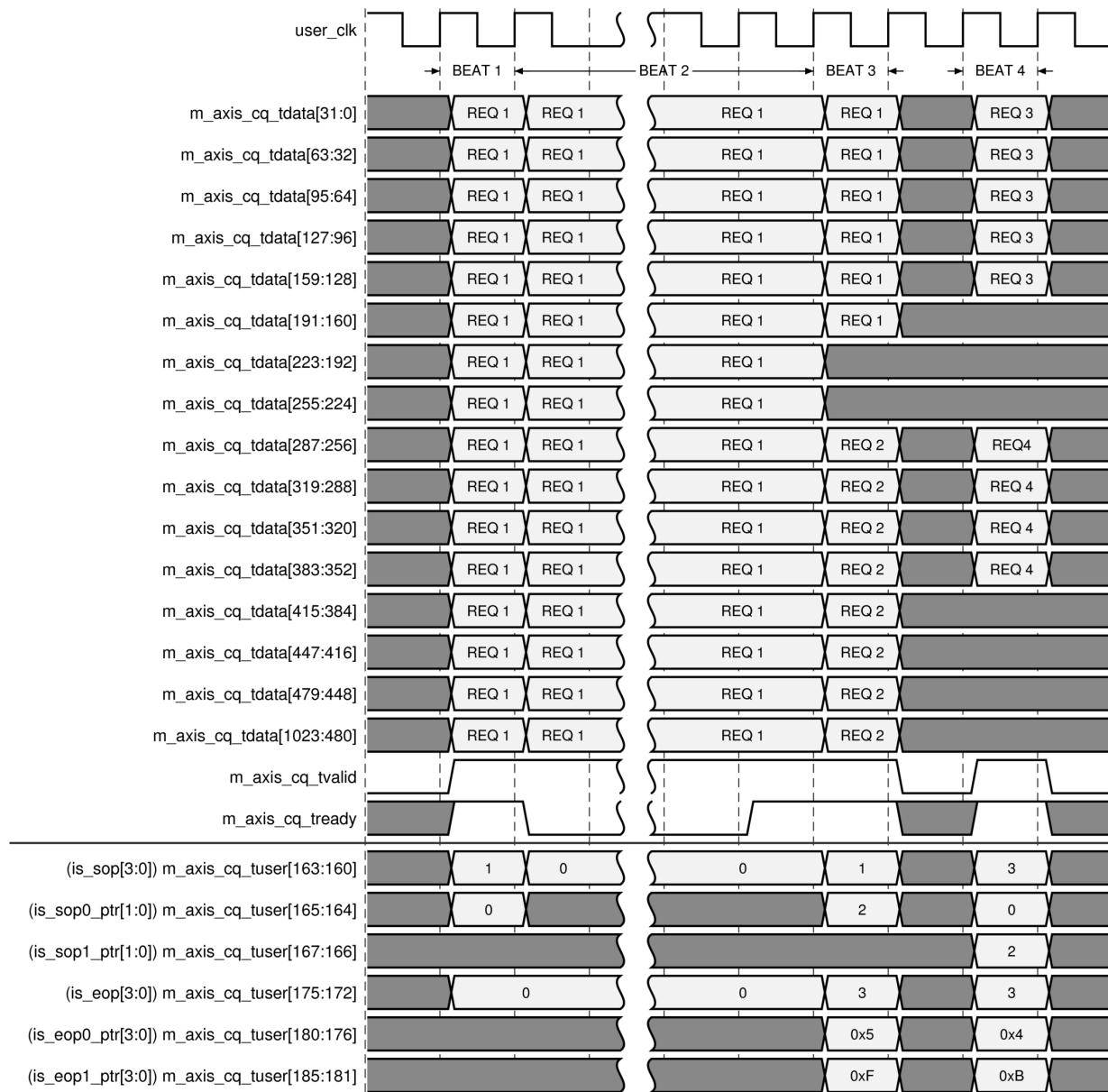
CQ 接口上的跨接选项

当上一个请求已终止于数据总线上的 Dword 位置 7 或者在此位置前终止时，核具备在同一拍内在请求器完成接口上传输新请求的功能。在 Vivado® IDE 中进行核自定义期间可启用此跨接选项。跨接选项只能配合 Dword 对齐模式一起使用。

当启用跨接选项时，在 AXI4-Stream 接口上，请求 TLP 将作为无数据包边界的连续数据流进行传输。因此，在判定接口上交付的 TLP 的边界过程中不使用 `m_axis_rc_tkeep` 信号和 `m_axis_rc_tlast` 信号（使用跨接选项时，核会永久性将 `m_axis_rc_tkeep` 的位全部设置为 1，将 `m_axis_rc_tlast` 全部设置为 0）。并改为使用 `m_axis_rc_tuser` 总线中所提供的以下信号来执行 TLP 的界定。

- `is_sop[0]`: 当有至少 1 个请求 TLP 从节拍中开始时，在该节拍中，核就会将此输出设置为高电平有效。此 TLP 的描述符的第 1 个字节的位置判定方式：
 - 如果前一个 TLP 在此节拍前结束，那么描述符的第 1 个字节位于字节通道 0 中。
 - 如果前一个 TLP 在此节拍继续，那么此描述符的第 1 个字节位于字节通道 32 中。仅当前一个 TLP 在当前节拍中结束时（即同时设置 `is_eop[0]` 时），才有可能出现此情况。
- `is_sop[1]`: 当有至少 2 个请求 TLP 从同一拍中开始时，在该节拍中，核会断言此输出有效。第 1 个 TLP 始终在字节位置 0 开始，第 2 个 TLP 则在字节位置 32 开始。仅当前一个 TLP 在同一拍内在字节位置 32 之前结束时，核才能在字节位置 32 启动第 2 个 TLP；即，仅当 `is_eop[0]` 也在同一拍内设置时才会如此。
- `is_eop[0]`: 此输出用于指示请求 TLP 结束。断言此输入有效即表示有至少 1 个 TLP 在此节拍中结束。
- `is_eop0_ptr[3:0]`: 断言 `is_eop[0]` 有效时，`is_eop0_ptr[3:0]` 提供在此节拍中结束的对应 TLP 的最后一个 Dword 的偏移。对于含有效载荷的 TLP，最后一个字节的偏移同样可根据 TLP 的起始地址和长度来确定，或者根据字节使能信号 `byte_en[63:0]` 来确定。
- `is_eop1_ptr[3:0]`: 断言 `is_eop[1]` 有效时，`is_eop1_ptr[3:0]` 提供在此节拍中结束的第 2 个 TLP 的最后一个 Dword 的偏移。对于含有效载荷的 TLP，最后一个字节的偏移同样可根据 TLP 的起始地址和长度来确定，或者根据字节使能信号 `byte_en[63:0]` 来确定。由于第 2 个 TLP 只能在字节通道 32 上起始，因此只能在 47-63 范围内的字节通道上结束。因此，偏移 `is_eop1_ptr[3:0]` 只能取 11-15 范围内的值。

图 78：在启用跨接选项的完成器请求接口上执行请求 TLP 的传输

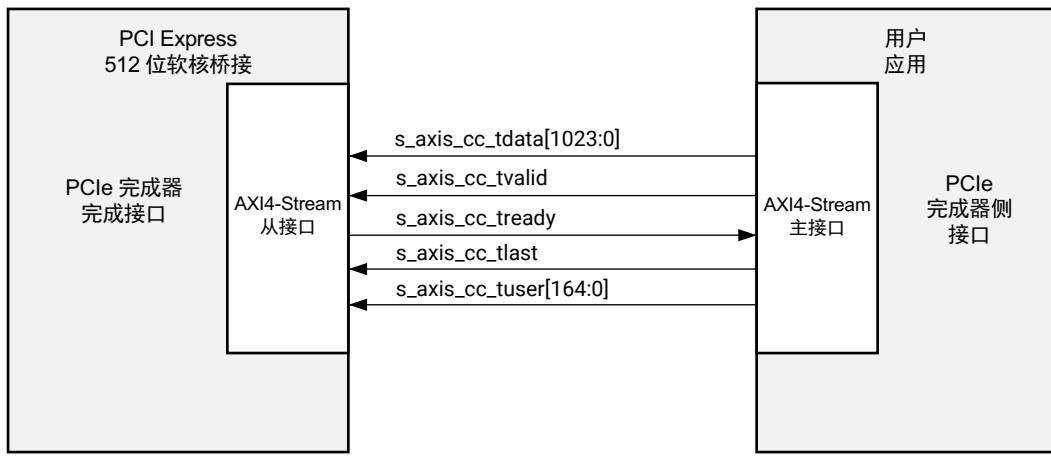


上图显示了启用跨接选项的情况下，在完成器请求接口上传输 4 个请求 TLP 的过程。对于所有 TLP，有效载荷的第 1 个 Dword 始终位于描述符之后，且没有任何间隙。第 1 个请求 TLP (REQ 1) 从节拍 1 的 Dword 位置 0 开始，并在节拍 3 的 Dword 位置 5 结束。第 2 个 TLP (REQ 2) 从同一拍内的 Dword 位置 8 开始。此第 2 个 TLP 仅含 1 个有效载荷（其中含 4 个 Dword），因此同样在同一拍内结束。第 3 和第 4 个请求 TLP 完全在节拍 4 中传输，因为 REQ 3 仅含 1 个有效载荷（含 1 个 Dword），而 REQ 4 则不含任何有效载荷。

完成器完成接口操作 (512 位)

下图显示了与核的完成器完成接口关联的信号。核将此接口上的每个 TLP 均作为 1 个 AXI4-Stream 数据包来交付。对于含有效载荷的完成包，此数据包以 96 位描述符开头并后接数据。

图 79：完成器完成接口信号



完成器请求接口支持 2 种不同的数据对齐模式，在 Vivado® IDE 中执行核自定义期间可选择模式。在 Dword 对齐模式下，有效数据的第一个字节必须显示在通道 $n = (S + 12 + (A \bmod 4)) \bmod 64$ 中，其中 A 是要传输的数据块的字节级别起始位置，S 是显示描述符的第一个字节的通道编号。地址 A 将取作为描述符的“Lower Address”（下位地址）字段中的值。不使用跨接选项时，起始通道编号 S 始终为 0，但启用跨接选项时编号可为 0 或 32。

在 128 位地址对齐模式下，对于有效载荷的第一个字节的通道编号判定方式为 $n = (S + 16 + (A \bmod 16)) \bmod 64$ ，其中 S 为出现描述符的第一个字节的通道编号（0 或 32），A 则是对应于有效载荷的第一个字节的地址。描述符的结束位置与有效载荷的第一个字节的起始位置之间的间隔全部以空字节来填充。

该接口还支持跨接选项，此选项允许在同一拍内跨接口传输最多 2 个 TLP。跨接选项只能配合 Dword 对齐模式一起使用，使用 128 位地址对齐模式时，不支持跨接选项。以下章节中的描述假定每拍 1 个 TLP。[CC 接口上的跨接选项](#) 中描述了启用跨接选项的接口的操作。

完成器完成描述符格式

用户应用将完成器请求的完成数据作为独立 AXI4-Stream 包发送至核的完成器完成接口。每个数据包均以 1 个描述符开头，在描述符后可包含有效载荷数据。描述符长度始终为 12 字节，并在完成包的前 12 字节内发送。描述符始终在完成 TLP 的第一拍内进行传输。当用户应用将请求的完成数据拆分为多个拆分完成 (Split Completion) 包后，它必须将每个拆分完成包作为独立 AXI4-Stream 包随其描述符一起发送。

下图演示了完成器完成描述符的格式。下表描述了完成器请求描述符的每个字段。

图 80：完成器完成描述符格式

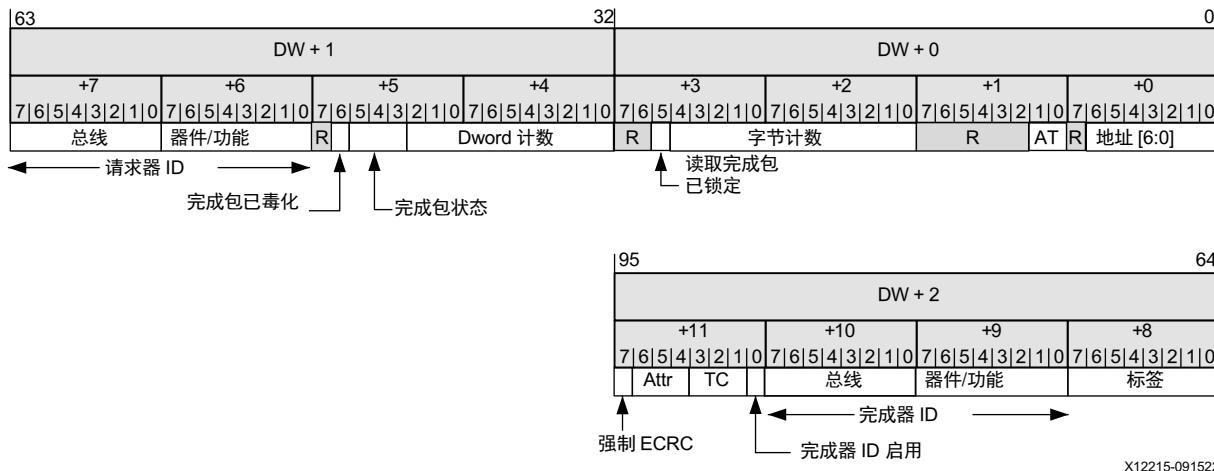


表 60：完成器完成描述符字段

位索引	字段名称	描述
6:0	“Lower Address” (低位地址)	对于存储器读取完成包，该字段必须设置为所传输的存储器块的起始字节级地址的 7 个最低有效位。对于所有其它完成包，下位地址必须全部设置为 0。
9:8	“Address Type” (地址类型)	该字段定义仅适用于存储器传输事务和原子操作的完成包。对于这些完成包，用户逻辑必须将 AT 位从对应的请求描述符复制到该字段中。针对所有其它完成包，该字段必须设置为 0。
28:16	“Byte Count” (字节计数)	这 13 位可包含范围在 0 - 4,096 个字节内的值。如果存储器读取请求已通过使用单一完成包完成，那么 “Byte Count” 值将以字节为单位来表示 “Payload” (有效载荷) 大小。针对 I/O 读取完成包和 I/O 写入完成包，该字段必须设置为 4。针对长度为 0 的存储器读取发送完成包时，字节计数必须设置为 1，并且在描述符之后必须附加含单 Dword 的虚拟有效载荷。 对于每个 “Memory Read Completion” (存储器读取完成)， “Byte Count” 字段必须表明完成请求所需的剩余字节数，包括随完成包返回的字节数。如果存储器读取请求已使用多个完成包完成，那么后续每个完成包的字节计数值均表示为前一个完成包减去随前一个完成包返回的字节数。
29	“Locked Read Completion” (锁定读取完成)	当 “Locked Read” (锁定读取) 请求的响应中包含完成包时，必须设置该位。针对所有其它完成包，该位必须设置为 0。
42:32	“Dword Count” (Dword 计数)	这 11 位表示当前包的有效载荷大小 (以 Dword 数为单位)。其范围是 0 - 1000 个 Dword。针对 I/O 读取完成包，该字段必须设置为 1，针对 I/O 写入完成包，该字段必须设置为 0。针对长度为 0 的存储器读取发送完成包时，Dword 计数必须设置为 1。发送 UR 或 CA 完成时，Dword 计数必须设置为 0。在所有其它情况下，Dword 计数必须对应于当前包的有效载荷中 Dword 的实际数量。
45:43	“Completion Status” (完成状态)	这些位必须基于所发送的完成 (Completion) 包的类型来设置。有效设置仅包括： <ul style="list-style-type: none">· 000: 成功完成· 001: 请求不受支持 (UR)· 100: 完成器异常中止 (CA)
46	“Poisoned Completion” (毒化完成)	此位可供用户逻辑用于对所发送的完成 TLP 进行毒化。针对所有完成包，此位必须设置为 0，除非用户逻辑在位于描述符后的数据块中检测到错误，并且想使用 PCI Express 的 “Data Poisoning” (数据毒化) 功能来传递此信息。
63:48	“Requester ID” (请求器 ID)	与请求关联的 PCI 请求器 ID (由用户逻辑从请求复制所得)。

表 60：完成器完成描述符字段 (续)

位索引	字段名称	描述
71:64	“Tag” (标签)	与请求关联的 PCIe 标签（由用户逻辑从请求复制所得）。
79:72	“Target Function/Device Number” (目标功能/器件编号)	<p>完成器功能的器件和/或功能编号。</p> <p>端点模式：</p> <p>ARI 已启用：</p> <ul style="list-style-type: none">· 位 [79:72] 必须设置为完成器功能编号。 <p>ARI 已禁用：</p> <ul style="list-style-type: none">· 位 [74:72] 必须设置为完成器功能编号。· 不使用位 [79:75] <p>开关的上游端口用例（在 IP 中选中端点模式）：</p> <p>ARI 已启用：</p> <ul style="list-style-type: none">· 位 [79:72] 必须设置为完成器功能编号。 <p>ARI 已禁用：</p> <ul style="list-style-type: none">· 位 [74:72] 必须设置为完成器功能编号。· 如果完成包源自开关本身，则不使用位 [79:75]。如果开关正在中继完成包（完成器位于开关外部），那么这些位必须设置为完成器器件编号，此编号对应于该完成包的来源位置。该位可与描述符中的“Completer ID Enable”位配合使用。 <p>根端口模式（下游端口）：</p> <p>ARI 已启用：</p> <ul style="list-style-type: none">位 [79:72] 必须设置为完成器功能编号。 <p>ARI 已禁用：</p> <ul style="list-style-type: none">位 [74:72] 必须设置为完成器功能编号。位 [79:75] 必须设置为完成器器件编号。该位可与描述符中的“Completer ID Enable”位配合使用。
87:80	“Completer Bus Number” (完成器总线编号)	<p>与完成器功能关联的总线编号。</p> <p>端点模式：</p> <ul style="list-style-type: none">· 如果完成包源自开关本身，则不使用这些位。如果开关正在中继完成包（完成器位于开关外部），那么这些位必须设置为完成器总线编号，此编号对应于该完成包的来源位置。该位可与描述符中的“Completer ID Enable”位配合使用。 <p>根端口模式（下游端口）：</p> <ul style="list-style-type: none">· 必须设置为完成器总线编号。该位可与描述符中的“Completer ID Enable”位配合使用。

表 60：完成器完成描述符字段 (续)

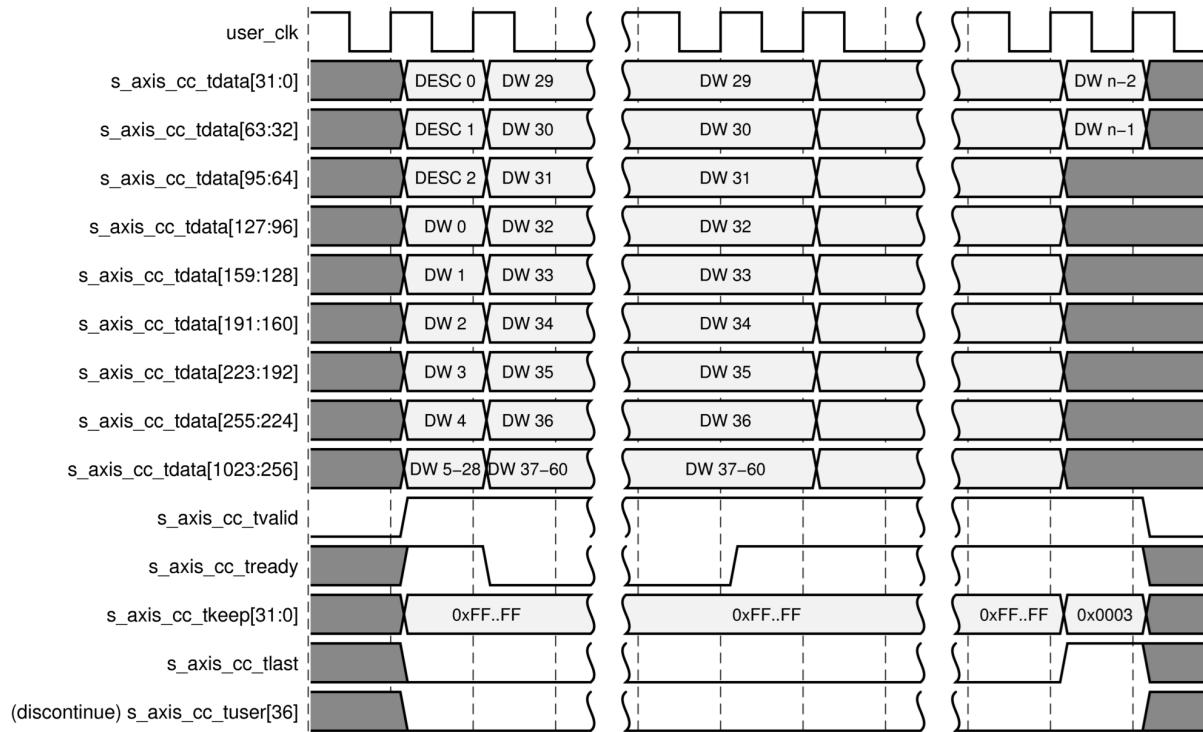
位索引	字段名称	描述
88	“Completer ID Enable” (完成器 ID 使能)	1'b1：客户在描述符中提供总线编号、器件编号和功能编号，以供填充到 TLP 报头中的“Completer ID”(完成器 ID) 字段中。 1'b0：IP 使用从接收到的配置请求中捕获的总线和器件编号，客户在描述符中提供功能编号，以供填充到 TLP 报头中的“完成器 ID”字段中。 端点模式： <ul style="list-style-type: none">必须设置为 1'b0。 开关的上游端口用例（在 IP 中选中端点模式）： <ul style="list-style-type: none">如果完成包源自开关本身，则设置为 1'b0。当开关正在中继完成（完成器位于开关外部）时，则设置为 1'b1。不启用 ARI 时，该位可与完成器总线编号位 [95:88] 和完成器功能/器件编号位 [87:83] 配合使用。 根端口模式： <ul style="list-style-type: none">必须设置为 1'b1。不启用 ARI 时，该位可与完成器总线编号位 [95:88] 和完成器功能/器件编号位 [87:83] 配合使用。
91:89	“Transaction Class (TC)” (传输事务类)	与请求关联的 PCIe 传输事务类 (TC)。用户逻辑必须从关联的请求描述符的 TC 字段复制该值。
94:92	属性	与请求关联的 PCIe 属性（从请求复制所得）。位 92 为“No Snoop”（无嗅探）位，位 93 为“Relaxed Ordering”（宽松排序）位，位 94 为“ID-Based Ordering”（基于 ID 排序）位。
95	保留	保留以供将来使用。

含成功完成 (SC) 状态的完成包

每次从完成器请求接口接收到 1 个非转发请求时，用户逻辑都必须向核的完成器完成接口返回 1 个完成 (Completion) 包。当请求完成且不含任何错误时，用户逻辑必须返回含成功完成 (SC) 状态的完成包。根据请求类型，此完成包可能包含有效载荷。此外，当数据块大小超出配置的有效载荷最大大小时，与请求关联的数据可拆分为多个拆分完成 (Split Completion) 包。用户逻辑负责根据需要将数据块拆分为多个拆分完成包。用户逻辑必须将每个拆分完成包作为独立 AXI4-Stream 数据包（包含其自己的 12 字节描述符）通过完成器完成接口进行传输。

在以下时序图示例中，要传输的数据块的起始 Dword 地址（如描述符的下位地址字段的位 [6:2] 中所述）假定为 (m^*8+1) ，其中 m 为整数。数据块的大小假定为 n 个 Dword，其中 $n = k^*32+28$ 且 $k > 0$ 。

图 81：完成器完成接口上正常完成包的传输（512 位接口、Dword 对齐模式）



上图显示了通过完成器完成接口来传输源于用户逻辑的完成包（采用 Dword 对齐模式）的过程。在此情况下，有效载荷的第一个 Dword 紧接在描述符之后开始。当数据块并非 4 字节的倍数时，或者当有效载荷的开始位置与 Dword 地址边界未对齐时，用户应用必须添加空字节以使有效载荷对齐到 Dword 边界，并使有效载荷成为 Dword 数量的整数倍。例如，当数据块从字节地址 7 开始且大小为 3 字节时，用户逻辑必须在第 1 个字节前添加 3 个空字节，并在数据块末尾添加 2 个空字节，以使其长度达到 2 个 Dword。并且，对于非连续读取，数据块中返回的字节并非全都有效。在此情况下，用户应用必须在适当位置返回有效字节，并根据需要在有效字节间的间隙内填充空字节。此接口不提供任何信号用于指示有效载荷中的有效字节。此类信号并非必需，原因在于请求器会负责保留请求中字节使能的记录，并丢弃来自完成包的无效字节。

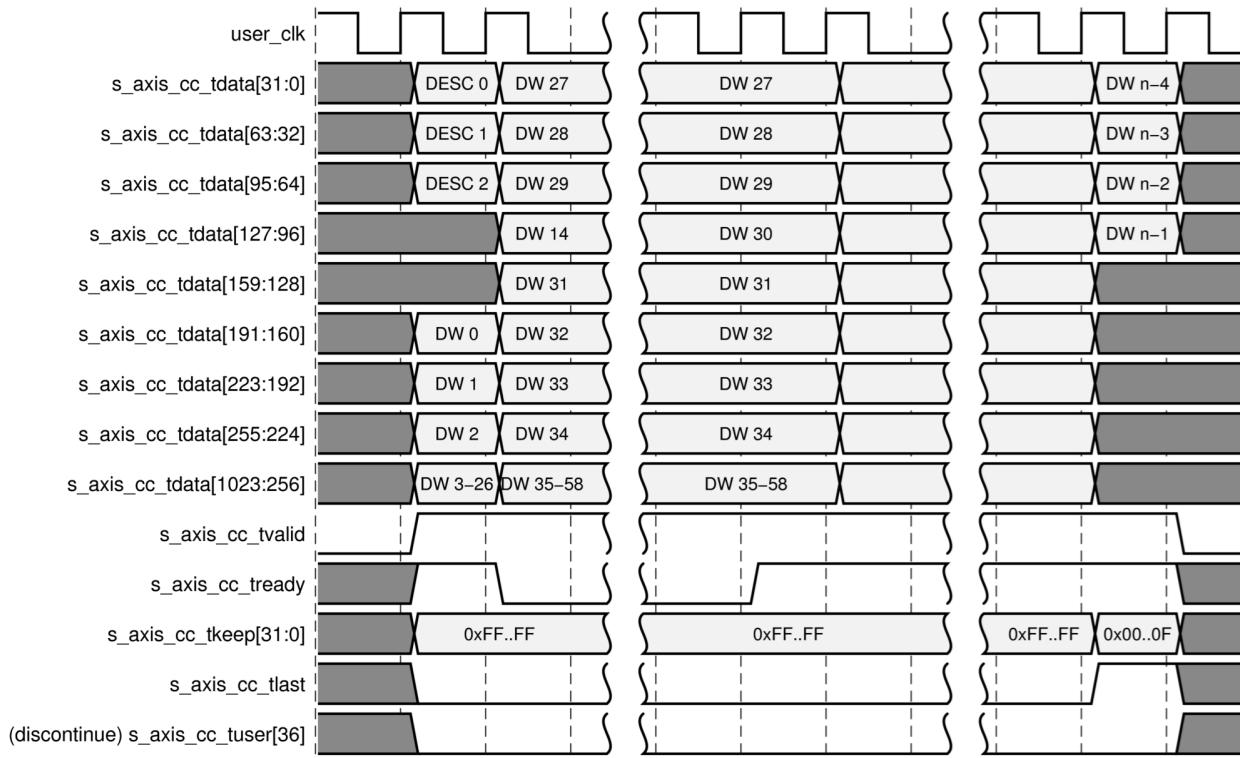
在 Dword 对齐模式下，传输从 12 个描述符字节开始，紧随其后即为有效载荷字节。用户应用必须在数据包的整个持续时间段保持 `s_axis_cc_tvalid` 信号处于断言有效状态。核会将数据包传输期间断言无效的 `s_axis_cc_tvalid` 作为错误来处理，并将链路上发射的对应完成 TLP 置空，以避免数据损坏。

用户应用还必须在数据包的最后一拍中断言 `s_axis_cc_tlast` 信号有效。如果核未准备好接受数据，可下拉 `s_axis_cc_tready`。传输期间，当核已断言 `s_axis_cc_tready` 无效时，用户应用不得更改 `s_axis_cc_tdata` 和 `s_axis_cc_tlast` 上的值。

在 128 位地址对齐模式下，有效载荷的交付必须始终从 512 位字的第 2 个 128 位四分之一拍开始，紧随第 1 个四分之一拍的描述符之后。即，如果描述符的第 1 个字节位于字节通道 0 上，那么有效载荷必须从字节通道 16 - 31 范围内的某一字节通道开始。在其 128 位四分之一拍内，有效载荷的第 1 个字节的偏移必须对应于相应的描述符中“下位地址”字段的最低有效位。

以下时序图显示了通过完成器完成接口来传输存储器读取完成包（采用 128 位地址对齐模式）的过程。为便于演示，要传输的数据块的起始 Dword 地址（如描述符的下位地址字段的位 [6:2] 中所述）假定为 $(m * 16 + 1)$ ，其中 m 为整数。数据块的大小假定为 n 个 Dword，其中 $n = k * 16 - 1$ 且 $k > 1$ 。

图 82：完成器完成接口上正常完成包的传输（128 位地址对齐模式）



完成包传输异常中止

用户逻辑可以在传输有效载荷期间随时通过断言 `s_axis_cc_tuser` 总线中的 `discontinue` 信号有效来异常中止完成器完成接口上的完成包传输。核会将链路上对应 TLP 置空，以避免数据损坏。

在传输期间，当传输的完成包具有关联有效载荷时，用户逻辑可以在任意周期内断言此信号有效。用户逻辑可以选择在周期内发出错误信号处提前终止该数据包（通过断言 `s_axis_cc_tlast` 有效），或者也可以继续处理，直至将有效载荷的所有字节都交付到核为止。针对后者，该核会针对数据包的后续节拍将此错误作为粘滞错误来处理，即使用户逻辑在达到包结束前断言 `discontinue` 信号无效也是如此。

仅当 `s_axis_cc_tvalid` 为高电平有效时，才能断言 `discontinue` 信号有效。当 `s_axis_cc_tvalid` 和 `s_axis_cc_tready` 均为高电平有效状态时，核会对此信号进行采样。因此，一旦断言有效后，在 `s_axis_cc_tready` 处于高电平有效状态之前不应将其断言无效。

当核配置为端点时，核会使用高级错误报告 (AER) 机制向所连接到的根联合体报告此错误（作为不可纠正的内部错误）。

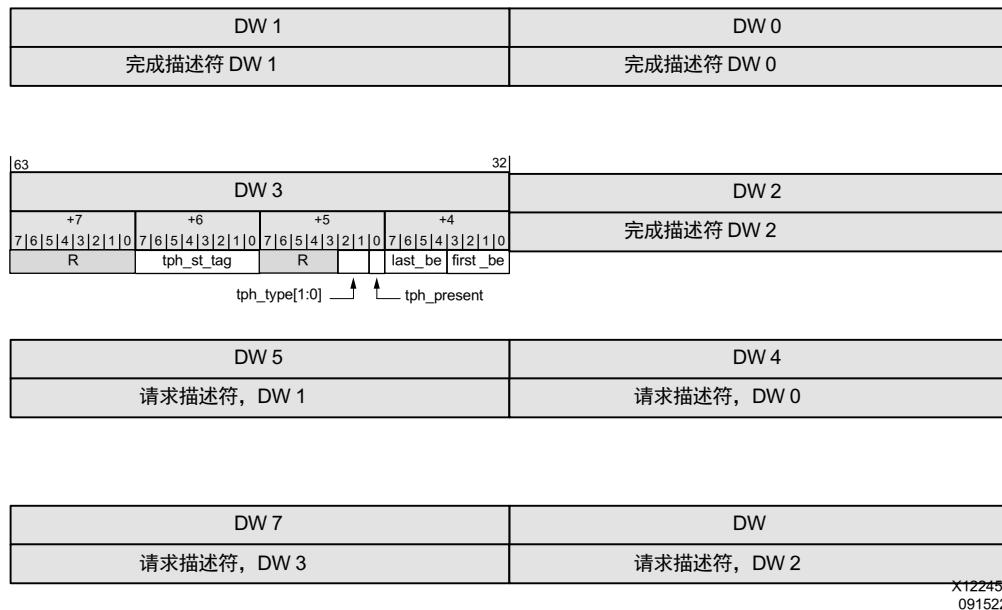
含错误状态的完成包（UR 和 CA）

如果完成器请求接口上接收到的状态为请求不受支持 (UR) 或完成器异常中止 (CA)，那么响应此类请求时，用户逻辑必须发送含 3 个 Dword 的完成描述符（格式如 [完成器完成描述符格式](#) 中的完成器完成描述符格式图中所示），后接另 5 个 Dword（其中包含有关生成完成包的请求的信息）。核需使用这 5 个 Dword 在其 AER 报头 log 日志寄存器中记录有关该请求的信息。

下图显示了发送含 UR 或 SC 状态的完成包时的信息传输顺序。这些信息格式化为 AXI4-Stream 数据包，其中包含总计 8 个 Dword，组织方式如下：

- 前 3 个 Dword 包含完成描述符，格式如 [完成器完成描述符格式](#) 中的完成器完成描述符格式图中所示。
- 第 4 个 Dword 包含 `m_axis_cq_tuser` 中的下列信号的状态（复制自请求）：
 - `m_axis_cq_tuser` 中的首字节使能位 `first_be[3:0]`。
 - `m_axis_cq_tuser` 中的末字节使能位 `last_be[3:0]`。
 - 承载传输事务处理提示相关信息的信号：`m_axis_cq_tuser` 中的 `tph_present`、`tph_type[1:0]` 和 `tph_st_tag[7:0]`。
 - 随请求一起从核接收到的请求描述符的 4 个 Dword。

图 83：含 UR 和 CA 完成包的 AXI4-Stream 数据包的组成方式



CC 接口上的跨接选项

当上一个请求已终止于数据总线上的 Dword 位置 7 或者在此位置前终止时，核具备在同一拍内在完成器完成接口上传输新完成数据包的能力。在 Vivado® IDE 中进行核自定义期间可启用此跨接选项。跨接选项只能配合 Dword 对齐模式一起使用。

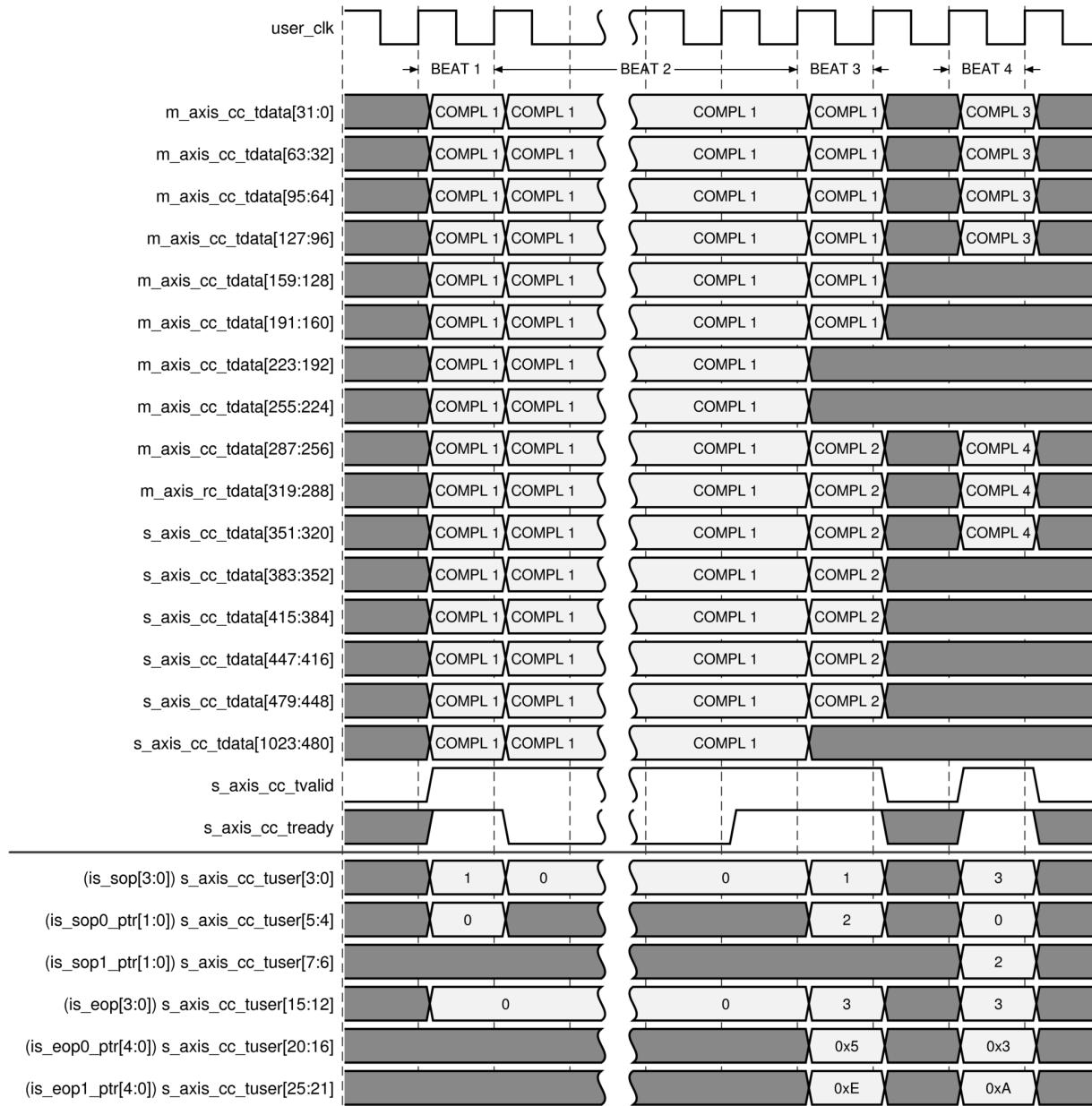
当启用跨接选项时，在 AXI4-Stream 接口上，完成 TLP 将作为无数据包边界的连续数据流进行传输。因此，在判定接口上交付的 TLP 的边界过程中不使用 `m_axis_cc_tkeep` 信号和 `m_axis_cc_tlast` 信号。并改为使用 `m_axis_cc_tuser` 总线中所提供的以下信号来执行 TLP 的界定。

- `is_sop[0]`: 当有至少 1 个完成 TLP 从节拍中开始时，在该节拍中此输入必须设为高电平。此 TLP 的描述符的第一个字节的位置判定方式：
 - 如果前一个 TLP 在此节拍前结束，那么描述符的第一个字节位于字节通道 0 中。
 - 如果前一个 TLP 在此节拍继续，那么此描述符的第一个字节位于字节通道 32 中。仅当前一个 TLP 在当前节拍中结束时（即同时设置 `is_eop[0]` 时），才有可能出现此情况。
- `is_sop0_ptr[1:0]`: 设置 `is_sop[0]` 时，该字段必须指示当前节拍中起始的第一个完成 TLP 的偏移。有效设置包括 `2'b00` (TLP 起始位置为 Dword 0) 和 `2'b10` (TLP 起始位置为 Dword 8)。

- `is_sop[1]`: 当有 2 个完成 TLP 在同一拍内起始时，在该节拍中此输入必须设置为高电平。第 1 个 TLP 必须始终在字节位置 0 开始，第 2 个 TLP 则在字节位置 32 开始。仅当前一个 TLP 在同一拍内在字节位置 32 之前结束时，用户应用才能在字节位置 32 启动第 2 个 TLP；即，仅当 `is_eop[0]` 也在同一拍内设置时才会如此。
- `is_sop1_ptr[1:0]`: 设置 `is_sop[1]` 时，该字段必须提供当前节拍中起始的第 2 个请求 TLP 的偏移。其唯一有效设置为 `2'b10` (TLP 起始位置为 Dword 8)。
- `is_eop[0]`: 此输入用于指示完成 TLP 结束。断言此输入有效即表示有至少 1 个 TLP 在此节拍中结束。
- `is_eop0_ptr[3:0]`: 断言 `is_eop[0]` 有效时，`is_eop0_ptr[3:0]` 必须提供在此节拍中结束的对应 TLP 的最后一个 Dword 的偏移。
- `is_eop[1]`: 当有 2 个 TLP 在当前节拍中结束时，此输入设置为高电平。仅当 `is_eop[0]` 信号和 `is_sop[0]` 信号在同一拍内同样为高电平时，才能设置 `is_eop[1]`。
- `is_eop1_ptr[3:0]`: 断言 `is_eop[1]` 有效时，`is_eop1_ptr[3:0]` 必须提供在此节拍中结束的第 2 个 TLP 的最后一个 Dword 的偏移。由于第 2 个 TLP 只能在字节通道 32 上起始，因此只能在 43-63 范围内的字节通道上结束。因此，偏移 `is_eop1_ptr[3:0]` 只能取 10-15 范围内的值。

下图显示了启用跨接选项的情况下，在完成器完成接口上传输 4 个完成 TLP 的过程。对于所有 TLP，有效载荷的第 1 个 Dword 始终位于描述符之后，且没有任何间隙。第 1 个完成 TLP (COMPL 1) 从节拍 1 的 Dword 位置 0 开始，并在节拍 3 的 Dword 位置 0 结束。第 2 个 TLP (COMPL 2) 从同一拍内的 Dword 位置 8 开始。此第 2 个 TLP 仅含 1 个有效载荷（其中含 4 个 Dword），因此同样在同一拍内结束。第 3 和第 4 个完成 TLP 完全在节拍 4 中传输，因为 COMPL 3 仅含 1 个有效载荷（含 1 个 Dword），而 COMPL 4 则不含任何有效载荷。

图 84：在启用跨接选项的完成器完成接口上执行完成 TLP 的传输（512 位接口）



512 位请求器接口

本章节旨在描述与 512 位 AXI4-Stream 接口关联的用户侧请求器接口的操作。[512 位完成器接口](#) 中的模块框图显示了软核桥接、PCIe® 核和用户应用之间的连接。软核桥接将 500 MHz 的 256 位数据包转换为 250 MHz 的 512 位数据包。

请求器接口支持用户端点应用发起 PCI 传输事务，并将其作为总线主控制器以跨 PCIe 链路传输至主机存储器。对于根联合体，此接口还用于发起 I/O 和配置请求。此接口还可供端点和根联合体用于在 PCIe 链路上发送报文。此接口上的传输事务类似于完成器接口上的传输事务，但核及用户应用的角色发生了对换。转发传输事务作为单一不可分割操作来执行，非转发传输事务则作为拆分传输事务来执行。

请求器接口包含 2 个独立接口，对应每个方向的数据传输使用 1 个接口。每个接口都基于 AXI4-Stream 协议，其宽度可配置为 64、128 或 256 位。请求器请求接口用于将请求（含任何关联的有效载荷数据）从用户应用传输到核，而请求器完成接口则供核用于将从链路接收到的完成包（针对非转发请求）交付到用户应用。2 个接口各自独立运行，即，用户应用可以通过请求器请求接口传输新请求，同时接收前一个请求的完成包。

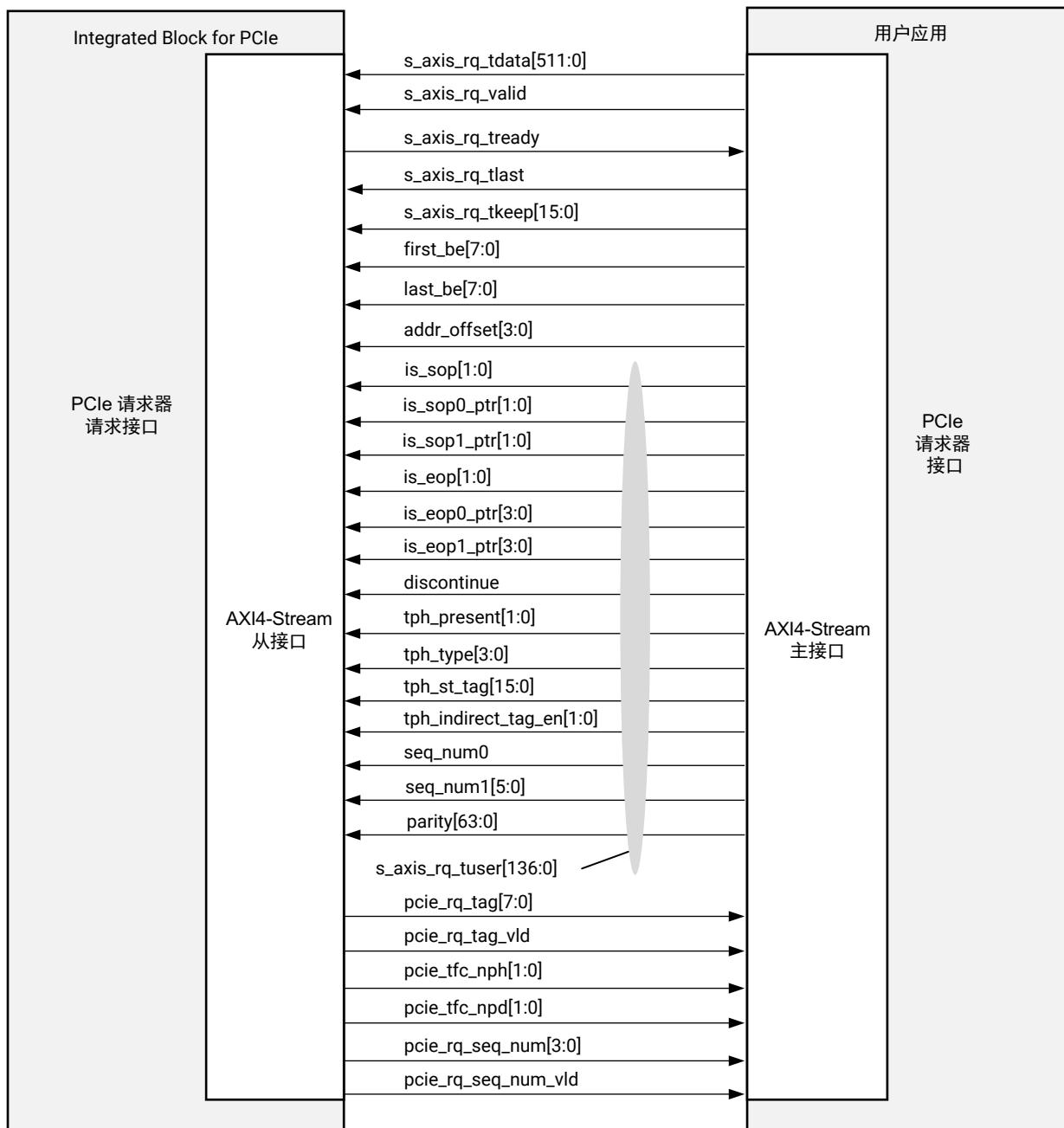
请求器请求接口操作（512 位）

下图显示了与核的请求器请求接口关联的信号。核将此接口上的每个 TLP 均作为 1 个 AXI4-Stream 数据包来交付。对于含有效载荷的 TLP，此数据包以 128 位描述符开头并后接数据。

请求器请求接口支持通过 2 种不同的数据对齐模式来进行有效载荷传输，此模式在 Vivado® IDE 中执行核自定义期间进行设置。在 Dword 对齐模式下，用户逻辑必须紧跟在描述符的最后一个 Dword 之后提供有效载荷的第一个 Dword。并且，还必须在 `first_be[7:0]` 中设置相应的位以指示第一个 Dword 中的有效字节，并在 `last_be[7:0]` 中设置相应的位（两者均为 `s_axis_rq_tuser` 总线的一部分）以指示有效载荷的最后一个 Dword 中的有效字节。在地址对齐模式下，用户逻辑必须在描述符的最后一个 Dword 之后的节拍中开始有效载荷的传输，其第一个 Dword 可位于数据路径上的任意可能的 Dword 位置内。用户应用使用 `s_axis_rq_tuser` 中的 `addr_offset[3:0]` 信号来传达数据路径上的第一个 Dword 的偏移。采用 Dword 对齐模式的情况下，用户应用还必须在 `first_be[7:0]` 中设置相应的位以指示第一个 Dword 中的有效字节，并在 `last_be[7:0]` 中设置相应的位以指示有效载荷的最后一个 Dword 中的有效字节。在已启用跨接的情况下，`addr_offset[3:2]`、`first_be[7:4]` 和 `last_be[7:4]` 用于指示第 2 个 TLP 的信息，而 `addr_offset[1:0]`、`first_be[3:0]` 和 `last_be[3:0]` 则用于指示该数据节拍上第 1 个 TLP 的信息。

在核中启用“Transaction Processing Hint Capability”（传输事务处理提示功能）的情况下，用户逻辑可以通过使用 `s_axis_rq_tuser` 总线中包含的 `tph_*` 信号来为任何存储器传输事务提供可选提示。要随请求提供提示，用户逻辑必须在数据包的第一拍内断言 `tph_present` 有效，并分别在 `tph_st_tag[7:0]` 和 `tph_st_type[1:0]` 上提供“TPH Steering Tag”（TPH 导向标签）和“Steering Tag Type”（导向标签类型）。

图 85：请求器请求接口信号



X22927-030223

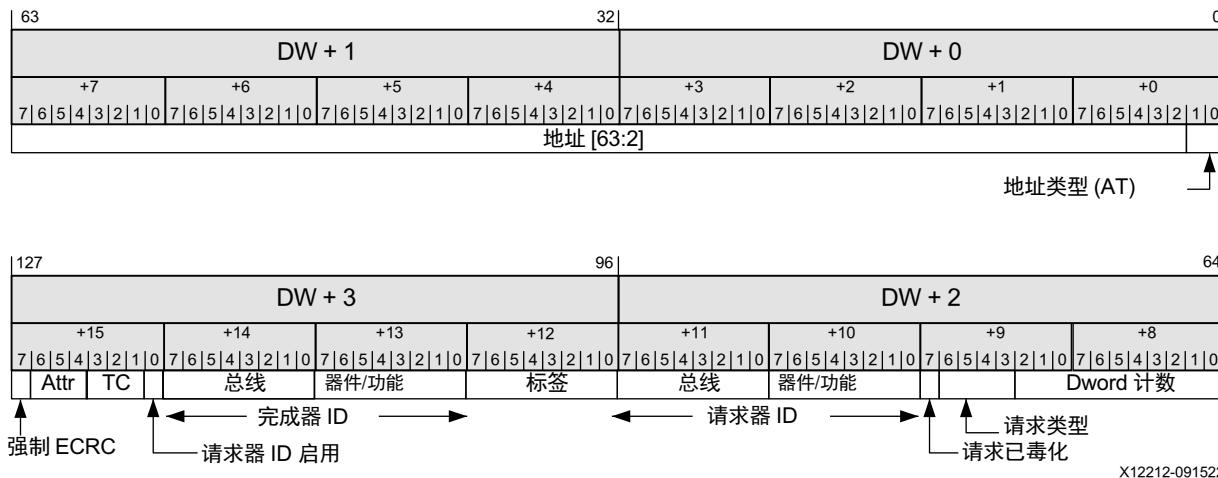
该接口还支持跨接选项，此选项允许在同一拍内跨接口传输最多 2 个 TLP。跨接选项只能配合 Dword 对齐模式一起使用，使用 128 位地址对齐模式时，不支持跨接选项。以下章节中的描述假定每拍 1 个 TLP。[RQ 接口上的跨接选项](#) 中描述了启用跨接选项的接口的操作。

请求器请求描述符格式

用户应用必须将链路上要发射的每个请求都作为独立 AXI4-Stream 包传输到核的请求器请求接口上。每个数据包都必须以 1 个描述符开头，在描述符后可包含有效载荷数据。描述符长度始终为 16 字节，并必须在请求包的前 16 字节内发送。描述符在 64 位接口上的前 2 个节拍内进行传输，在 128 位或 256 位接口上的第 1 个节拍内进行传输。下图演示了不同类型的请求的描述符格式。

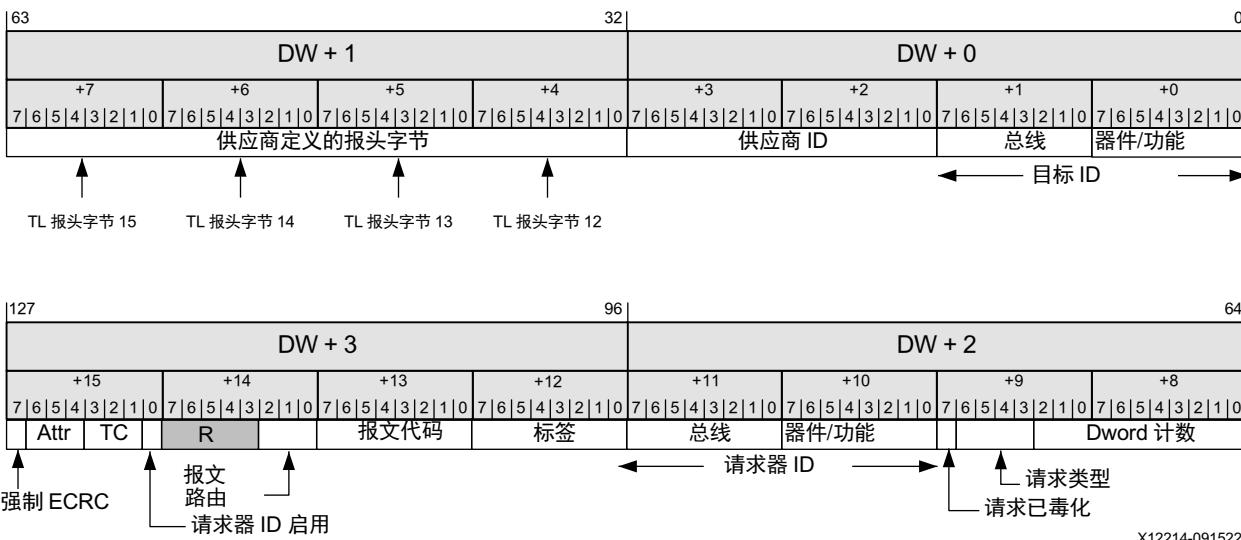
当所传输的请求 TLP 为存储器读取/写入请求、I/O 读取/写入请求或原子操作 (Atomic Operation) 请求时，即适用下图所示格式。

图 86：对应存储器、I/O 和原子操作请求的请求器请求描述符格式



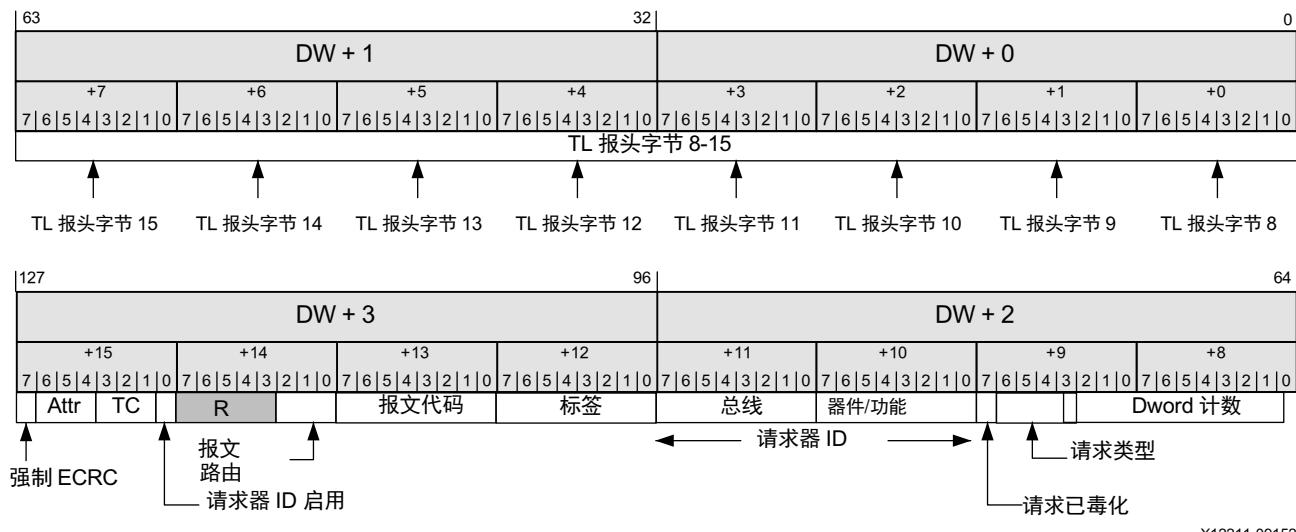
下图格式仅适用于类型 0 或类型 1 的“Vendor-Defined Messages”（供应商定义的报文）。

图 87：对应供应商定义的报文的请求器请求描述符格式



下图所示格式适用于所有 ATS 报文，包括“Invalid Request”（无效请求）、“Invalid Completion”（无效完成）、“Page Request”（页面请求）、“PRG Response”（PRG 响应）。

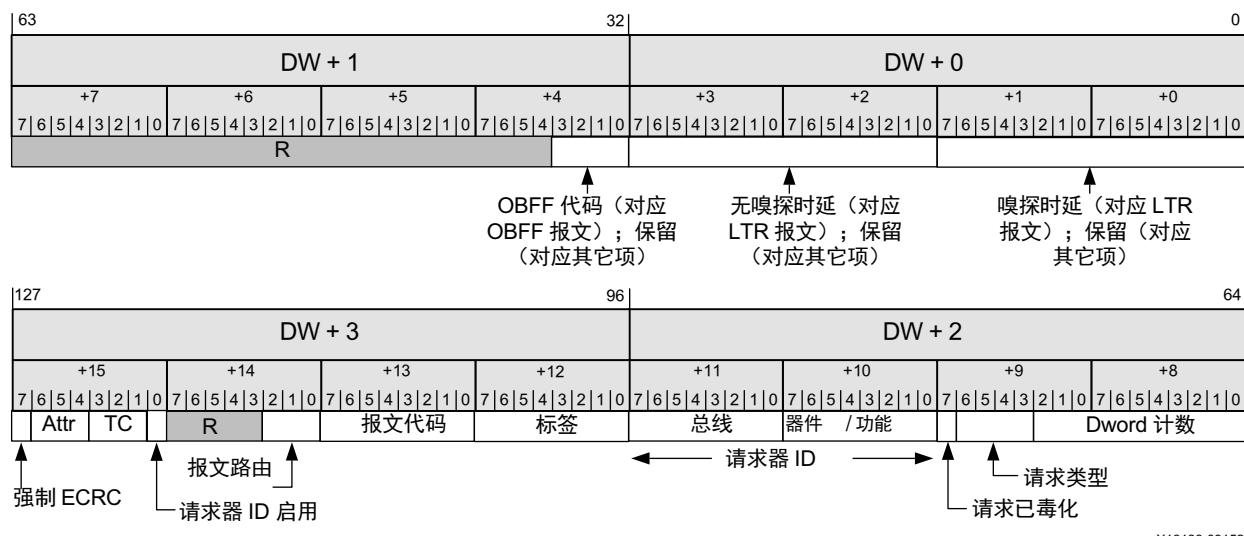
图 88：对应于 ATS 报文的请求器请求描述符



X12211-091522

对于所有其它报文，描述符采用下图所示格式。

图 89：对应于所有其它报文的请求器请求描述符格式



X16186-091522

表 61：请求器请求描述符字段

位索引	字段名称	描述
1:0	“Address Type” (地址类型)	该字段定义仅适用于存储器传输事务和原子操作。核会将此字段复制到请求 TLP 的 TL 报头的 AT 中。 <ul style="list-style-type: none"> 00: 请求中的地址未经转换 01: 传输事务为转换请求 10: 请求中的地址为已转换的地址 11: 保留

表 61：请求器请求描述符字段 (续)

位索引	字段名称	描述
63:2	“Address” (地址)	该字段适用于存储器、I/O 和原子操作请求。这是请求所引用的首个 Dword 的地址。用户逻辑还必须在 <code>s_axis_rq_tuser</code> 中设置 First_BE 位和 Last_BE 位，这两个位分别用于指示首个 Dword 和最后一个 Dword 中的有效字节。 当传输事务指定 32 位地址时，该字段的位 [63:32] 必须设置为 0。
74:64	“Dword Count” (Dword 计数)	这 11 个位用于指示要读取或写入的块大小（以 Dword 为单位），对于报文，这些位指示报文有效载荷的大小。其范围为 0 - 256 个 Dword。对于 I/O 访问，Dword 计数始终为 1。 对于长度为 0 的存储器读取/写入请求，Dword 计数必须为 1，且 First_BE 位全部设置为 0。 核不会对照提供的有效载荷（针对含有效载荷的请求）的实际长度检查该字段的设置，也不会对照核的最大有效载荷大小或读取请求大小设置检查此项设置。
78:75	“Request Type” (请求类型)	用于识别传输事务类型。下表中列出了传输事务类型及其编码。
79	“Poisoned Request” (毒化请求)	此位可供用户逻辑用于对所发送的请求 TLP 进行毒化。针对所有请求，此位必须设置为 0，除非用户逻辑在位于描述符后的数据块中检测到错误，并且想使用 PCI Express 的 “Data Poisoning” (数据毒化) 功能来传递此信息。
87:80	“Requester Function/Device Number” (请求器功能/器件编号)	请求器功能的器件和/或功能编号。 端点模式： <ul style="list-style-type: none">· ARI 已启用：<ul style="list-style-type: none">◦ 位 [87:80] 必须设置为请求器功能编号。· ARI 已禁用：<ul style="list-style-type: none">◦ 位 [82:80] 必须设置为请求器功能编号。◦ 不使用位 [87:83] 开关的上游端口用例（在 IP 中选中端点模式）： <ul style="list-style-type: none">· ARI 已启用：<ul style="list-style-type: none">◦ 位 [87:80] 必须设置为请求器功能编号。· ARI 已禁用：<ul style="list-style-type: none">◦ 位 [82:80] 必须设置为请求器功能编号。◦ 如果请求源自开关本身，则不使用位 [87:83]。如果开关正在中继请求（请求器位于开关外部），那么这些位必须设置为请求器器件编号，此编号对应于该请求的来源位置。该位可与描述符中的 “Requester ID Enable” 位配合使用。 根端口模式（下游端口）： <ul style="list-style-type: none">· ARI 已启用：<ul style="list-style-type: none">◦ 位 [87:80] 必须设置为请求器功能编号。· ARI 已禁用：<ul style="list-style-type: none">◦ 位 [87:80] 必须设置为请求器功能编号。◦ 位 [87:83] 必须设置为请求器器件编号。该位可与描述符中的 “Requester ID Enable” 位配合使用。

表 61：请求器请求描述符字段 (续)

位索引	字段名称	描述
95:88	“Requester Bus Number”（请求器总线编号）	<p>与请求器功能关联的总线编号。</p> <p>端点模式：</p> <ul style="list-style-type: none">不使用 <p>开关的上游端口用例（在 IP 中选中端点模式）：</p> <ul style="list-style-type: none">如果请求源自开关本身，则不使用这些位。如果开关正在中继请求（请求器位于开关外部），那么这些位必须设置为请求器总线编号，此编号对应于该请求的来源位置。该位可与描述符中的“Requester ID Enable”位配合使用。 <p>根端口模式（下游端口）：</p> <ul style="list-style-type: none">必须设置为请求器总线编号。该位可与描述符中的“Requester ID Enable”位配合使用。
103:96	“Tag”（标签）	<p>与请求关联的 PCIe 标签。对于转发事务，核始终使用来自该字段的值作为请求的标签。</p> <p>对于非转发传输事务，如果在核配置期间，在 Vivado IDE 中已设置“Enable Client Tag”（启用客户标签）（即，当标签管理由用户逻辑执行时），那么该核会使用来自该字段的值。如果未设置该属性，那么核中的标签管理逻辑负责生成要使用的标签，并且不使用描述符的标签字段中的值。</p>
119:104	“Completer ID”（完成器 ID）	<p>该字段仅适用于按 ID 进行路由的配置请求和报文。对于这些请求，该字段会指定与请求关联的 PCI 完成器 ID（这 16 个位将在传统解读模式下分割为 8 位总线编号、5 位器件编号和 3 位功能编号。而在 ARI 模式下，这 16 个位则作为 8 位总线编号 + 8 位功能编号来处理）。</p>
120	“Requester ID Enable / T8”（请求器 ID 使能 / T8）	<p>1'b1：客户在描述符中提供总线编号、器件编号和功能编号，以供填充到 TLP 报头中的“Requester ID”（请求器 ID）字段中。</p> <p>1'b0：IP 使用从接收到的配置请求中捕获的总线和器件编号，客户在描述符中提供功能编号，以供填充到 TLP 报头中的“Requester ID”字段中。当“Requester ID Enable”（请求器 ID 使能）为 0 时，描述符中的器件编号字段也应为 0。</p> <p>端点模式：</p> <ul style="list-style-type: none">必须设置为 1'b0。 <p>开关的上游端口用例（在 IP 中选中端点模式）：</p> <ul style="list-style-type: none">如果请求源自开关本身，则设置为 1'b0。当开关正在中继请求（请求器位于开关外部）时，则设置为 1'b1。不启用 ARI 时，该位可与请求器总线编号位 [95:88] 和请求器功能/器件编号位 [87:83] 配合使用。 <p>根端口模式：</p> <ul style="list-style-type: none">必须设置为 1'b1。不启用 ARI 时，该位可与请求器总线编号位 [95:88] 和请求器功能/器件编号位 [87:83] 配合使用。
123:121	“Transaction Class (TC)”（传输事务类）	与请求关联的 PCIe 传输事务类 (TC)。
126:124	属性	<p>这些位可提供与请求关联的 Attribute 位的设置。位 124 为“No Snoop”（无嗅探）位，位 125 则为“Relaxed Ordering”（宽松排序）位。位 126 为“ID-Based Ordering”（基于 ID 排序）位，只能针对存储器请求和报文进行设置。</p> <p>如果在功能的 PCI Express 器件控制寄存器中未启用对应属性，那么核在链路上发送的请求中会将属性位强制设置为 0。</p>
111:104	“Message Code”（报文代码）	<p>该字段定义适用于所有报文。其中包含要在 TL 报头中设置的 8 位报文代码。</p> <p>《PCI Express 3.0 规范》(http://www.pcisig.com/specifications) 的附录 F 提供了受支持的报文代码的完整列表。</p>
114:112	“Message Routing”（报文路由）	该字段定义适用于所有报文。核会将这些位复制到请求 TLP 的 TL 报头的 3 位路由字段 r[2:0] 中。

表 61：请求器请求描述符字段 (续)

位索引	字段名称	描述
15:0	Destination ID	该字段仅适用于供应商定义的报文。当报文按 ID 进行路由（即，当“Message Routing”（报文路由）字段为 010 二进制值）时，该字段必须设置为报文的“Destination ID”（目标 ID）。
63:32	“Vendor-Defined Header”（供应商定义的报头）	该字段仅适用于供应商定义的报文。它将被复制到 TL 报头的 Dword 3 中。
63:0	ATS 报头	该字段仅适用于 ATS 报文。核会将其中包含的字节复制到 TL 报头的 Dword 2 和 Dword 3 中。

请求器存储器写入操作

在 Dword 对齐模式和 128 位地址对齐模式下，传输均从 16 个描述符字节开始，后接有效载荷字节。用户应用必须在数据包的整个持续时间段保持 `s_axis_rq_tvalid` 信号处于断言有效状态。核会将数据包传输期间断言无效的 `s_axis_rq_tvalid` 作为错误来处理，并将链路上发射的对应请求 TLP 置空，以避免数据损坏。

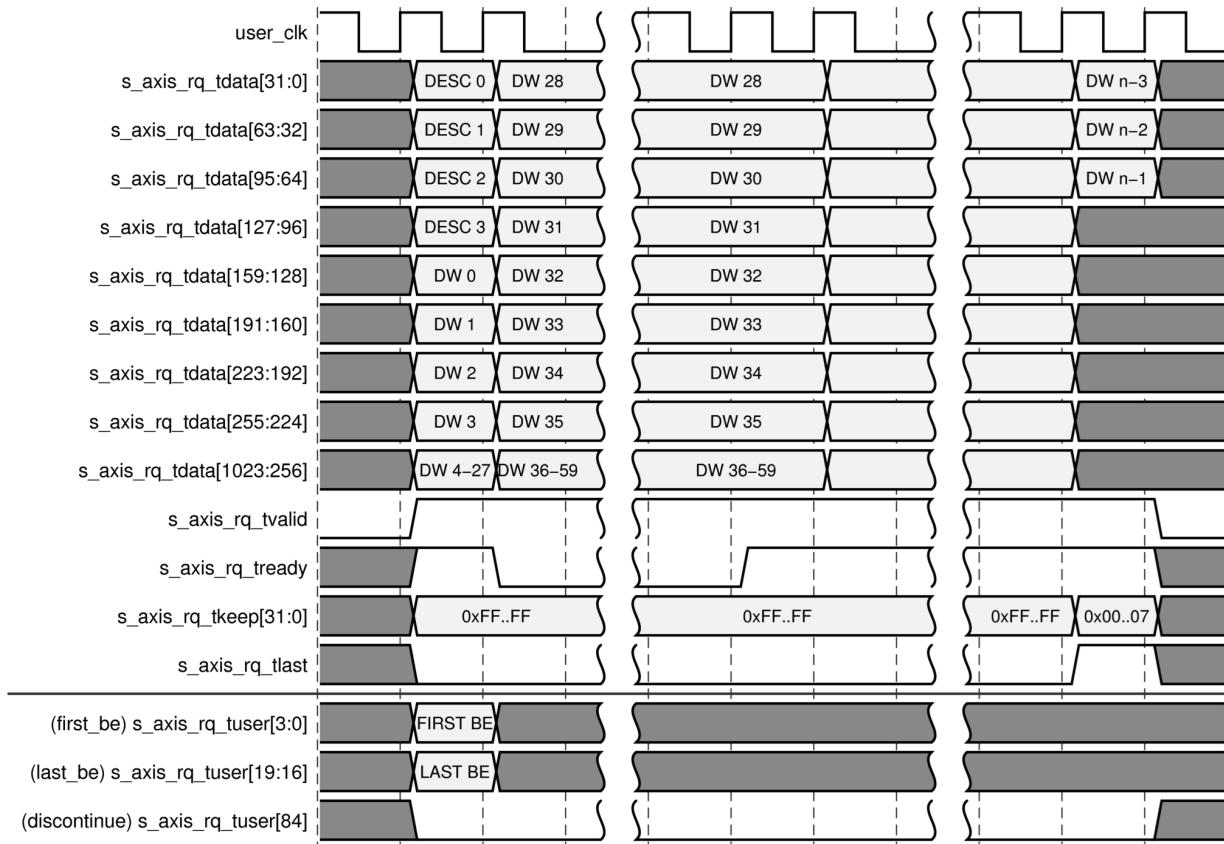
用户应用还必须在数据包的最后一拍中断言 `s_axis_rq_tlast` 信号有效。如果核未准备好接受数据，可下拉 `s_axis_rq_tready`。传输期间，当核已断言 `s_axis_rq_tready` 无效时，用户应用不得更改 `s_axis_rq_tdata` 和 `s_axis_rq_tlast` 上的值。必须设置 AXI4-Stream 接口信号 `m_axis_rq_tkeep`（每个信号对应 1 个 Dword 位置）以指示数据包中的有效 Dword，包括描述符和描述符与有效载荷之间插入的任意空字节。即，从描述符的第一个 Dword 开始到有效载荷的最后一个 Dword 为止的所有 `m_axis_rq_tkeep` 位都必须连续设置为 1。在数据包传输期间，当数据包无法填满接口的完整宽度时，`m_axis_rq_tkeep` 位仅限在数据包的最后一拍内才能设为 0。

请求器请求接口在 `s_axis_rq_tuser` 总线中还包含“First Byte Enable”（首字节使能）位和“Last Byte Enable”（末字节使能）位。在数据包的第一拍内必须设置这些位，并提供有关有效载荷的第一个和最后一个 Dword 中的有效字节的信息。

用户应用必须根据核中配置的最大有效载荷大小来限制单一请求内传输的有效载荷大小，并且必须确保有效载荷不超过 4 KB 的界限。当存储器写入不超过 2 个 Dword 时，`first_be[7:0]` 和 `last_be[7:0]` 上值为 1 的位不连续。另一种特殊情况是对于长度为 0 的存储器写入请求，用户应用必须提供虚拟 `one_dword` 有效载荷，其中 `first_be[7:0]` 和 `last_be[7:0]` 都全部设置为 0。在所有其它情况下，`first_be[7:0]` 和 `last_be[7:0]` 中值为 1 的位必须连续。在已启用跨接的情况下，`addr_offset[3:2]`、`first_be[7:4]` 和 `last_be[7:4]` 用于指示第 2 个 TLP 的信息，而 `addr_offset[1:0]`、`first_be[3:0]` 和 `last_be[3:0]` 则用于指示该数据节拍上第一个 TLP 的信息。

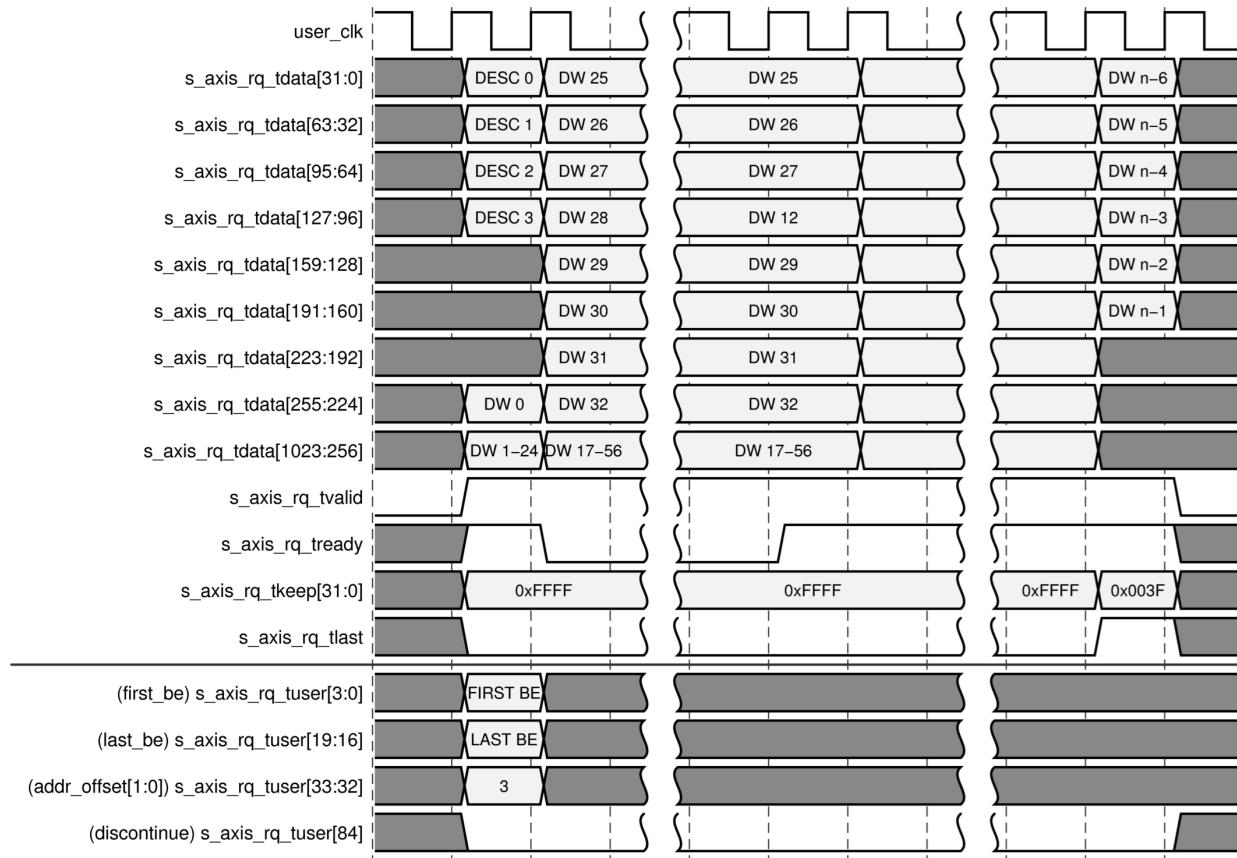
下图显示了通过请求器请求接口来传输源于用户逻辑的存储器写入请求（采用 Dword 对齐模式）的过程。为便于演示，写入用户存储器的数据块的大小假定为 n 个 Dword，其中， $n = k * 16 - 1$ 且 $k > 1$ 。

图 90：请求器请求接口上的存储器写入传输事务（Dword 对齐模式）



下图显示了通过请求器请求接口来传输源于用户应用的存储器写入请求（采用 128 位对齐模式）的过程。为便于演示，数据块的起始 Dword 偏移假定为 $(m*16 + 3)$ ，其中 $m > 0$ 且为整数。其大小假定为 n 个 Dword，其中 $n = k*16 - 1$ 且 $k > 1$ 。在 128 位地址对齐模式下，有效载荷的交付必须始终从 512 位字的第 2 个 128 位四分之一拍开始，紧随第 1 个四分之一拍的描述符之后。用户应用还必须在 **s_axis_rq_tuser** 总线的 **addr_offset[3:0]** 字段中告知有效载荷的第一个 Dword 的偏移。用户应用还必须在 **first_be[7:0]** 中设置相应的位以指示第一个 Dword 中的有效字节，并在 **last_be[7:0]** 中设置相应的位以指示有效载荷的最后一个 Dword 中的有效字节。

图 91：请求器请求接口上的存储器写入传输事务（128 位地址对齐模式）

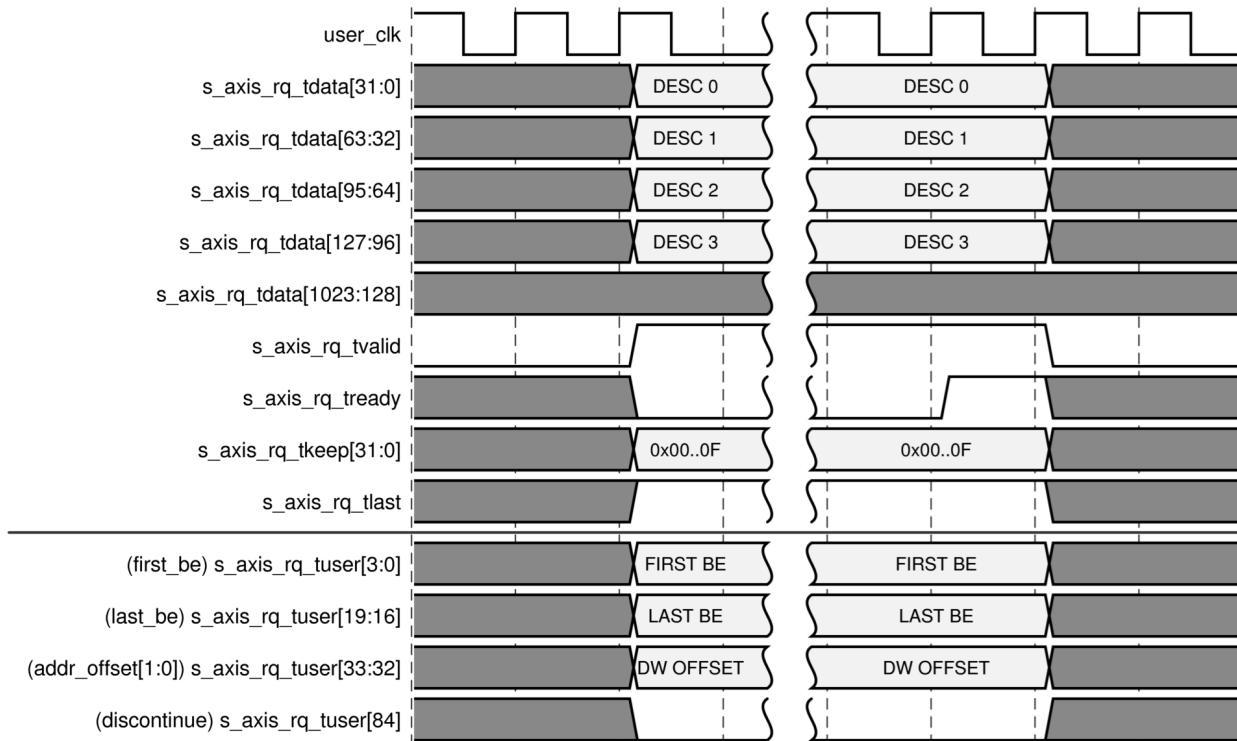


不含有效载荷的非转发传输事务

不含有效载荷（存储器读取请求、I/O 读取请求、配置读取请求）的非转发传输事务将按存储器写入请求相同的方式跨请求器请求接口进行传输，但区别在于 AXI4-Stream 数据包仅包含 16 字节描述符。下图显示了通过请求器请求接口来传输存储器读取请求的过程。在数据包持续时间段内，**s_axis_rq_tvalid** 信号必须保持处于断言有效状态。核可通过下拉 **s_axis_rq_tready** 来延长节拍。**s_axis_rq_tlast** 信号必须在数据包的最后一拍内设置，**s_axis_rq_tkeep[15:0]** 中的位则必须在存在描述符的所有 Dword 位置内进行设置。

用户应用必须分别使用 **s_axis_rq_tuser** 总线中的 **first_be[7:0]** 和 **last_be[7:0]** 来指示数据块的第一个和最后一个 Dword 的有效字节。对于长度为 0 的存储器读取的特殊情况，请求长度必须设置为 1 个 Dword，且 **first_be[7:0]** 和 **last_be[7:0]** 全部设置为 0。用户应用还必须在 **s_axis_rq_tuser** 总线的 **addr_offset[3:0]** 字段中告知生成的完成包（通过请求器请求接口交付）的有效载荷的第一个 Dword 的偏移。在已启用跨接的情况下，**addr_offset[3:2]**、**first_be[7:4]** 和 **last_be[7:4]** 用于指示第 2 个 TLP 的信息，**addr_offset[1:0]**、**first_be[3:0]** 和 **last_be[3:0]** 则用于指示该数据节拍上第 1 个 TLP 的信息。

图 92：请求器请求接口上的存储器读取传输事务



含有效载荷的非转发传输事务

含有效载荷的非转发请求（I/O 写入请求、配置写入请求或原子操作请求）的传输与存储器写入请求类似，但数据路径上有效载荷的对齐方式存在如下改变：

- 在 Dword 对齐模式下，有效载荷的第一个 Dword 位于描述符的最后一个 Dword 之后，两者间没有间隙。
- 在 128 位地址对齐模式下，有效载荷必须从紧随描述符之后的第一拍的第 2 个 128 位四分之一拍开始。有效载荷在此四分之一拍内的 4 个 Dword 位置中的任一位置开始。其第一个 Dword 的偏移必须在 s_axis_rq_tuser bus 的 addr_offset[3:0] 字段中指定。

对于 I/O 和配置写入请求，必须使用 first_be[7:0] 来明示单 Dword 有效载荷中的有效字节。对于原子操作请求，第一个和最后一个 Dword 中的所有字节均假定有效。

请求器接口上的报文请求

在请求器请求接口上传输报文的过程与存储器写入请求的传输过程类似，但有时其中不存在有效载荷。传输以 128 位描述符开始，其后即为有效载荷（如果存在）。有效载荷的第一个 Dword 必须紧接在描述符之后，与是否使用地址对齐模式无关。当使用地址对齐模式时，s_axis_rq_tuser 总线中的 addr_offset[3:0] 字段针对报文必须设置为 0。核可根据 s_axis_rq_tlast 信号和 s_axis_rq_tkeep 信号来判定有效载荷的结束位置。首字节使能位和末字节使能位 (first_be[7:0] 和 last_be[7:0]) 都不用于报文请求。

传输异常中止

对于包含关联有效载荷的任意请求，用户应用可以在传输有效载荷期间随时通过断言 s_axis_rq_tuser 总线中的 discontinue 信号有效来异常中止该请求。核会将链路上对应 TLP 置空，以避免数据损坏。

在传输期间，当传输的请求具有关联有效载荷时，用户应用可在任意周期内断言此信号有效。用户应用可以选择在周期内发出错误信号处提前终止该数据包（通过断言 `s_axis_rq_tlast` 有效），或者也可以继续处理，直至将有效载荷的所有字节都交付到核为止。针对后者，该核会针对数据包的后续节拍将此错误作为粘滞错误来处理，即使用户逻辑在达到包结束前断言 `discontinue` 信号无效也是如此。

仅当 `s_axis_rq_tvalid` 为高电平时，才能断言 `discontinue` 信号有效。当 `s_axis_rq_tvalid` 和 `s_axis_rq_tready` 均为高电平时，核会对此信号进行采样。因此，一旦断言有效后，在 `s_axis_rq_tready` 变为高电平之前不应将其断言无效。当通过断言 `discontinue` 输入有效来异常中止前一个数据包时，用户应用不得在同一拍内启动新的数据包。

当核配置为端点时，核会使用高级错误报告 (AER) 机制向所连接到的根联合体报告此错误（作为不可纠正的内部错误）。

RQ 接口上的跨接选项

当上一个请求已终止于数据总线上的 Dword 位置 7 或者在此位置前终止时，PCIe® 核具备在同一拍内在请求器请求接口上启动传输新请求包的能力。在 Vivado® IDE 中进行核自定义期间可启用此跨接选项。跨接选项只能配合 Dword 对齐模式一起使用。

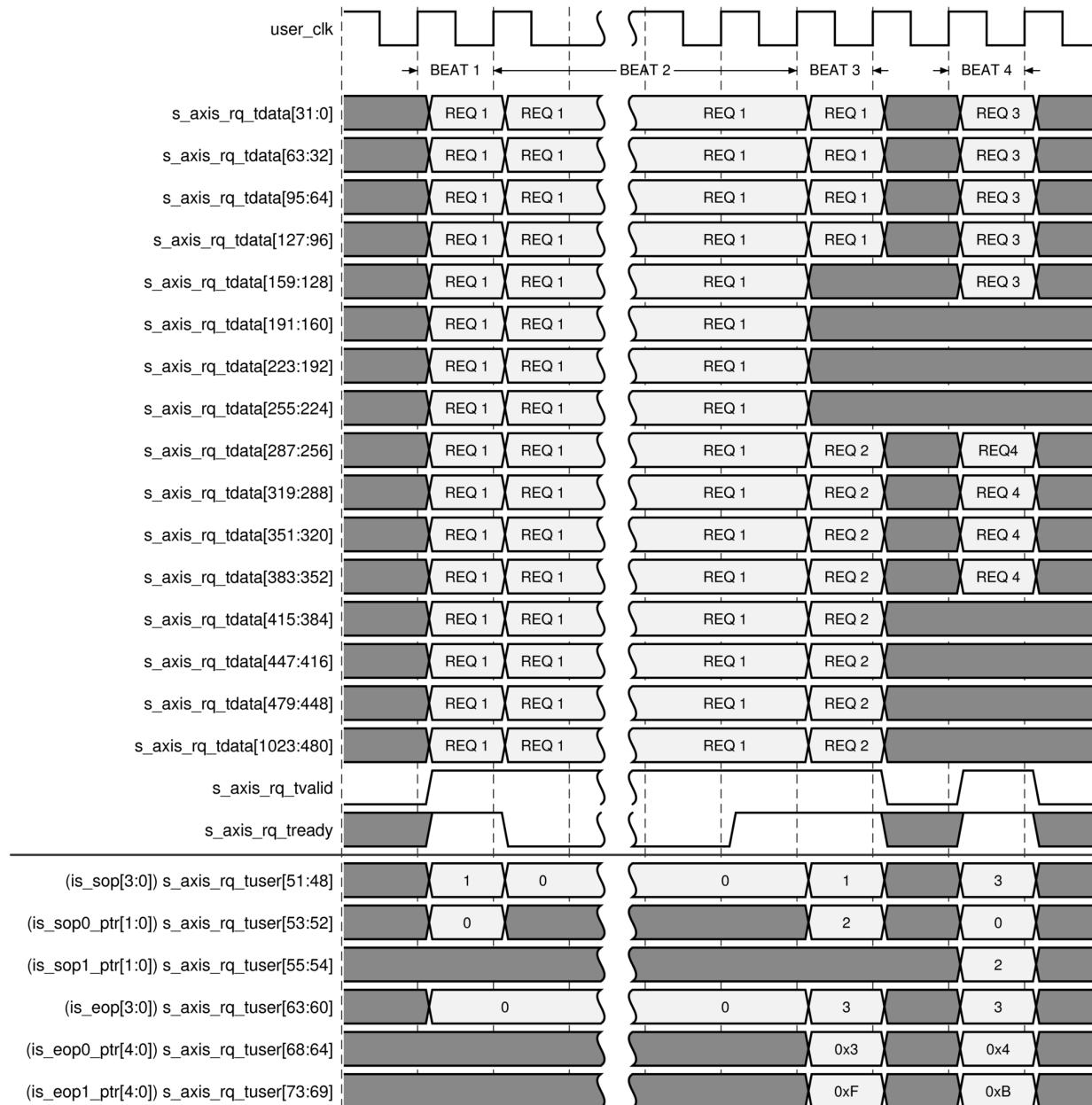
当启用跨接选项时，在 AXI4-Stream 接口上，请求 TLP 将作为无数据包边界的连续数据流进行传输。因此，在判定接口上交付的 TLP 的边界过程中不使用 `m_axis_rq_tkeep` 信号和 `m_axis_rq_tlast` 信号。并改为使用 `m_axis_rq_tuser` 总线中所提供的以下信号来执行 TLP 的界定。

- `is_sop[0]`: 当有至少 1 个请求 TLP 从节拍中开始时，在该节拍中此输入必须设为高电平。此 TLP 的描述符的第一个字节的位置判定方式：
 - 如果前一个 TLP 在此节拍前结束，那么描述符的第一个字节位于字节通道 0 中。
 - 如果前一个 TLP 在此节拍继续，那么此描述符的第一个字节位于字节通道 32 中。仅当前一个 TLP 在当前节拍中结束时（即同时设置 `is_eop[0]` 时），才有可能出现此情况。
- `is_sop0_ptr[1:0]`: 设置 `is_sop[0]` 时，该字段必须指示当前节拍中起始的第一个请求 TLP 的偏移。有效设置包括 2'b00 (TLP 起始位置为 Dword 0) 和 2'b10 (TLP 起始位置为 Dword 8)。
- `is_sop[1]`: 当有 2 个请求 TLP 在同一拍内起始时，在该节拍中此输入必须设置为高电平。第 1 个 TLP 必须始终在字节位置 0 开始，第 2 个 TLP 则在字节位置 32 开始。仅当前一个 TLP 在同一拍内在字节位置 32 之前结束时，用户应用才能在字节位置 32 启动第 2 个 TLP；即，仅当 `is_eop[0]` 也在同一拍内设置时才会如此。
- `is_sop1_ptr[1:0]`: 设置 `is_sop[1]` 时，该字段必须提供当前节拍中起始的第 2 个请求 TLP 的偏移。其唯一有效设置为 2'b10 (TLP 起始位置为 Dword 8)。
- `is_eop[0]`: 此输入用于指示请求 TLP 结束。断言此输入有效即表示有至少 1 个 TLP 在此节拍中结束。
- `is_eop0_ptr[3:0]`: 断言 `is_eop[0]` 有效时，`is_eop0_ptr[3:0]` 必须提供在此节拍中结束的对应 TLP 的最后一个 Dword 的偏移。
- `is_eop[1]`: 当有 2 个 TLP 在当前节拍中结束时，此输入设置为高电平。仅当 `is_eop[0]` 信号和 `is_sop[0]` 信号在同一拍内同样为高电平时，才能设置 `is_eop[1]`。
- `is_eop1_ptr[3:0]`: 断言 `is_eop[1]` 有效时，`is_eop1_ptr[3:0]` 必须提供在此节拍中结束的第 2 个 TLP 的最后一个 Dword 的偏移。由于第 2 个 TLP 只能在字节通道 32 上起始，因此只能在 43-63 范围内的字节通道上结束。因此，偏移 `is_eop1_ptr[3:0]` 只能取 10-15 范围内的值。

当第 2 个 TLP 在同一拍中起始时，第 2 个 TLP 的“First Byte Enable”（首字节使能）位和“Last Byte Enable”（末字节使能）位分别由 `tuser` 总线中的位字段 `first_be[7:4]` 和 `last_be[7:4]` 来指定。

下图显示了启用跨接选项的情况下，在请求器请求接口上传输 4 个请求 TLP 的过程。对于所有 TLP，有效载荷的第 1 个 Dword 始终位于描述符之后，且没有任何间隙。第 1 个请求 TLP (REQ 1) 从节拍 1 的 Dword 位置 0 开始，并在节拍 3 的 Dword 位置 3 结束。第 2 个 TLP (REQ 2) 从同一拍内的 Dword 位置 8 开始。此第 2 个 TLP 仅含 1 个有效载荷（其中含 4 个 Dword），因此同样在同一拍内结束。第 3 和第 4 个完成 TLP 完全在节拍 4 中传输，因为 REQ 3 仅含 1 个有效载荷（含 1 个 Dword），而 REQ 4 则不含任何有效载荷。

图 93：在启用跨接选项的请求器请求接口上执行请求 TLP 的传输



非转发传输事务的标签管理

核的请求器侧可保留由用户应用发起的所有暂挂非转发传输事务（存储器读取、I/O 读取和写入、配置读取和写入以及原子操作）的状态，因此目标所返回的完成包可与对应请求相匹配。每个未完成的传输事务的状态都保存在接口的请求器侧的“Split Completion Table”（拆分完成包表）中，其容量为不超过 256 个非转发传输事务。返回的完成包将使用 8 位标签与暂挂请求相匹配。有 2 个选项可用于这些标签的管理：

- “Internal Tag Management”（内部标签管理）：如果在 Vivado® IDE 中未设置“Enable Client Tag”（启用客户标签）选项（该核的默认设置），则启用“Internal Tag Management”。在此模式下，核中的逻辑负责为从请求器侧发起的每个非转发请求分配标签。该核会保留可用标签列表，当用户逻辑发起非转发传输事务时，该核会向每个请求分配 1 个标签，并通过输出 `pcie_rq_tag0[7:0]` 和 `pcie_rq_tag1[7:0]` 将分配的标签值传达给用户逻辑。当核断言 `pcie_rq_tag_vld0` 和 `pcie_rq_tag_vld1` 均有效时，该总线上的值即有效。
`pcie_rq_tag_vld0` 和 `pcie_rq_tag_vld1` 的使用与是否启用跨接选项无关。集成块可使用 `pcie_rq_tag_vld0` 或 `pcie_rq_tag_vld1` 端口来展示有效标签。用户逻辑必须复制此标签以便使核在请求的响应中所交付的任意完成包都与请求相匹配。

在此模式下，核中的逻辑会检查“Split Completion Table”的完整条件，如果当前未完成的非转发请求总数已达到上限，则会对来自用户逻辑的非转发请求进行反压（使用 `s_axis_rq_tready`）。

- “External Tag Management”（外部标签管理）：在 Vivado® IDE 中设置“Enable Client Tag”选项时，即启用“External Tag Management”。在此模式下，用户逻辑负责为从请求器侧发起的每个非转发请求分配标签。用户逻辑所选择的标签值与当时未完成的所有其它非转发传输事务的标签之间不得存在任何冲突，并且必须将此所选标签值传达给请求描述符中的核。核仍在其“拆分完成包表”中保留未完成的请求，并将传入完成包与请求相匹配，但不会对标签的唯一性进行任何检查，也不会检查“拆分完成包表”的完整条件。

使用“Internal Tag Management”（内部标签管理）时，核会断言 `pcie_rq_tag_vld` 有效并为每个非转发请求保持 1 个周期，然后它会将其分配的标签置于 `pcie_rq_tag` 上。启用跨接选项时，在同一周期内，核会在此接口上提供最多 2 个已分配的标签。`pcie_rq_tag_vld` 信号和 `pcie_rq_tag` 信号的状态必须按如下方式进行解读：

- 在任一周期内断言 `pcie_rq_tag_vld0` 有效即表示该核已将已分配的标签置于 `pcie_rq_tag0[7:0]` 上。
- 在同一周期内将 `pcie_rq_tag_vld0` 和 `pcie_rq_tag_vld1` 同步断言有效，表明该核已放置 2 个已分配的标签，其中第 1 个置于 `pcie_rq_tag0[7:0]` 上，第 2 个则置于 `pcie_rq_tag1[7:0]` 上。
`pcie_rq_tag0[7:0]` 上的标签对应于用户逻辑先前发送的请求，而 `pcie_rq_tag1[7:0]` 上的标签则对应于后续请求。
- 只要 `pcie_rq_tag_vld0` 不断言有效，`pcie_rq_tag_vld1` 就从不断言有效。即，在任一周期内如果只有 1 个标签要传达，则始终在 `pcie_rq_tag0[7:0]` 上传达此标签。
- 不使用跨接时，在任一周期内只能传达 1 个标签，并且 `pcie_rq_tag_vld1` 从不断言有效。

在 `s_axis_rq_tdata` 总线上上传输请求后，可能要经过数个周期的延迟之后，核才会断言 `pcie_rq_tag_vld` 有效以便为请求提供已分配的标签。与此同时，用户逻辑可以继续发送新请求。在 `pcie_rq_tag` 总线上，请求的标签按 FIFO 顺序来传递，这样用户逻辑即可轻松将标签值与其传输的请求相关联。

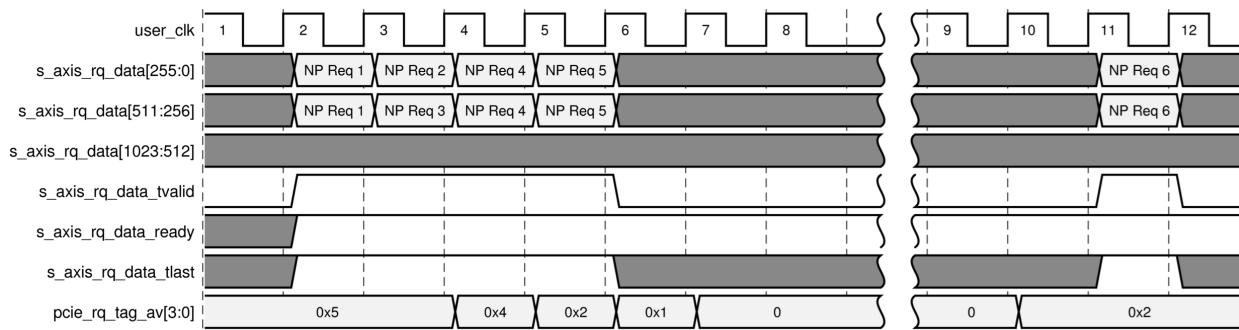
避免转发请求发生队头阻塞

核会因缺少发射信用值或者缺少可用标签而将其请求器请求接口上接收到的非转发请求置于保留状态。这可能导致转发传输事务发生队头阻塞 (HOL blocking)。如果用户逻辑能够检查非转发传输事务的发射信用值和标签可用性，则可避免发生此类状况。核为此提供了以下信号：

- `pcie_tfc_nph_av[3:0]`：这些输出表示非转发请求的当前可用头信用值（0000 = 无信用值可用、0001 = 可用信用值为 1、0010 = 信用值为 2、...、1111 = 可用信用值为 15 或更高）。
- `pcie_tfc_npd_av[3:0]`：这些输出表示非转发请求的当前可用数据信用值（0000 = 无信用值可用、0001 = 可用信用值为 1、0010 = 信用值为 2、...、1111 = 可用信用值为 15 或更高）。
- `pcie_rq_tag_av[3:0]`：这些输出表示当前可供分配给非转发请求的可用标签数量（0000 = 无标签可用、0001 = 可用标签数为 1、0010 = 可用标签数为 2、...、1111 = 可用标签数为 15 或更高）。

用户逻辑可选择先检查这些输出，然后再发射非转发请求。由于存在内部流水线延迟，这些输出上的信息相比于请求器请求接口上传输的最后一个字节所在周期晚 2 个用户时钟周期，因此用户逻辑必须调整这些值，将前 2 个时钟周期内发射的所有非转发请求一并纳入考量。下图演示了这些信号的操作。在此示例中，核最初的非转发头信用值为 7，非转发数据信用值为 3，并且可供分配的可用标签数为 5。来自用户逻辑的请求 1 具有 1 个有效载荷（含 1 个 Dword），因此耗用的头信用值和数据信用值各为 1，使用的标签数同样为 1。下一个时钟周期 3 内的请求 2 和 3（跨接）各自耗用的头信用值均为 1，但不耗用数据信用值。当用户逻辑在时钟周期 4 内提供请求 4 时，它必须调整可用信用值和可用标签计数，将前 2 个周期内的请求 1、2 和 3 一并纳入考量。请求 4 耗用的头信用值和数据信用值各为 1。当用户逻辑在时钟周期 5 内提供请求 5 时，它必须调整可用信用值和可用标签计数，将请求 2、3 和 4 一并纳入考量。如果请求 5 耗用的头信用值和数据信用值各为 1，那么 2 个周期后可用数据信用值为 0，可用标签数量同样如此。因此，请求 6 必须等待有新的信用值可用。

图 94：请求器请求接口上的信用值和标签可用性信号的操作



注释：如果用户逻辑选择使用 `pcie_tfc_*` 接口来监控可用的发射信用值，请确保在 `pcie_tfc_npd_av` 或 `pcie_tfc_nph_av` 达到 0 之后，没有任何其它非转发数据包进入 RQ 接口。集成块并不会丢失在此之后发出的非转发数据包；但 `pcie_tfc_*` 接口将无法再提供准确的信用值审计。

当 `cfg_fc_sel` 设置为可用发射信用值模式后，在 `cfg_fc_npd` 和 `cfg_fc_nph` 接口中同样可提供类似的发射信用值信息。

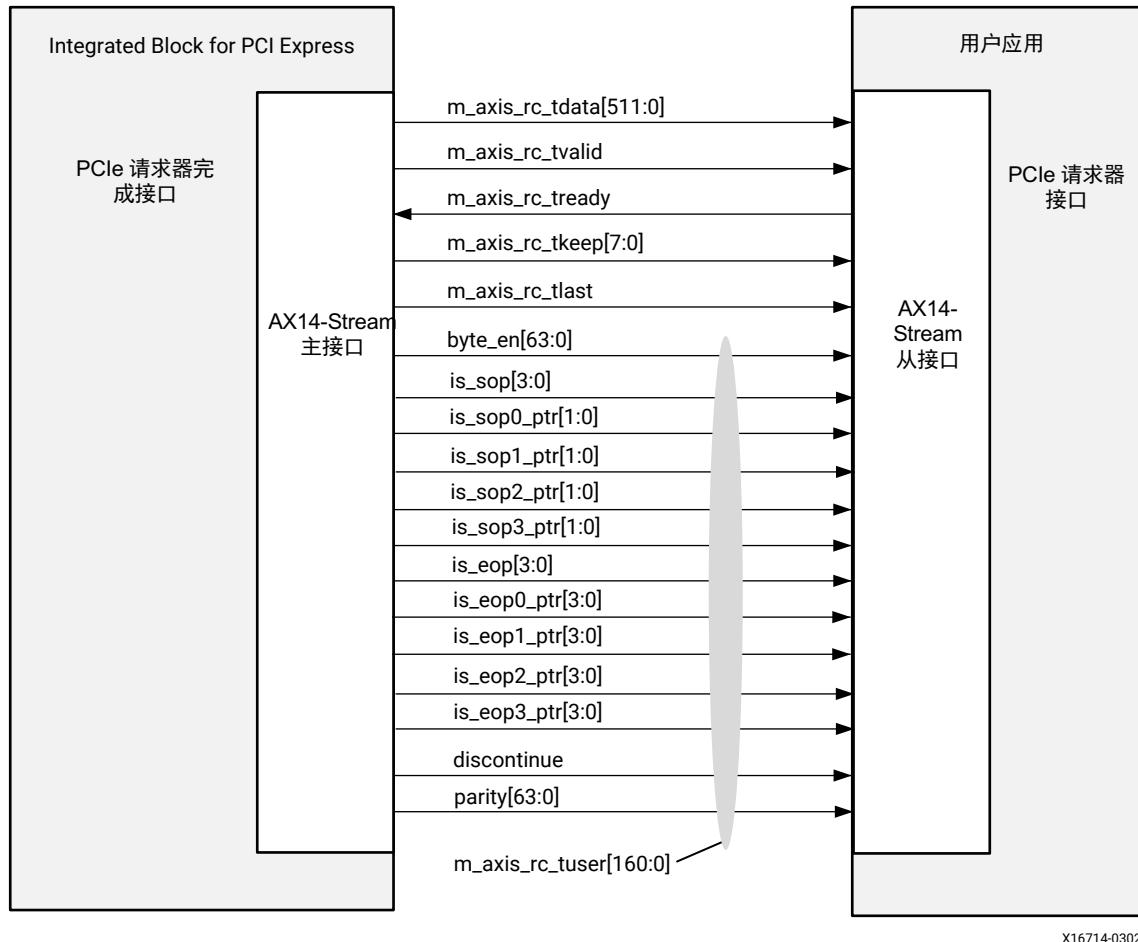
保持传输事务排序不变

核在链路上发射用户请求的顺序与其请求器接口上接收这些请求的顺序相同。如果用户逻辑想要精确控制请求器请求接口和完成器完成接口上的传输事务的发送顺序（通常目的是在使用严格排序时，禁止完成包传递转发请求），那么核会为用户逻辑提供相应的机制，以便监控转发传输事务通过其流水线的整个过程，从而判定何时需在完成器完成接口上调度完成包，同时可以避免传递从请求器请求接口发射的特定转发请求。

在跨请求器请求接口传输转发请求（存储器写入传输事务或报文）时，用户逻辑会在其第一拍内向 PCIe® 核提供可选 6 位序列号。`s_axis_rq_tuser` 中的序列号字段 `seq_num0[5:0]` 用于为此拍中开始的第 1 个 TLP 发送序列号，`field seq_num1[5:0]` 则用于为此拍中开始的第 2 个 TLP（如果存在）发送序列号。随后，用户逻辑即可监控核的 `pcie_rq_seq_num0[5:0]` 和 `pcie_rq_seq_num1[5:0]` 输出，以确认何时出现这些序列号。当传输事务到达核的内部发射流水线中的相应阶段，且在此阶段中完成包无法传递该传输事务时，核会断言 `pcie_rq_seq_num_vld0` 有效并保持 1 个周期，并在 `pcie_rq_seq_num0[5:0]` 输出上提供转发请求的序列号。如果同一周期内该流水线中存在第二个转发请求，那么核还会在同一周期内断言 `pcie_rq_seq_num_vld1` 有效，并在 `pcie_rq_seq_num1[5:0]` 输出上提供第二个转发请求的序列号。因此，用户逻辑必须监控这两组序列号输出，以检查特定 TLP 是否已到达相应的流水线阶段。在 `pcie_rq_seq_num0[5:0]` 或 `pcie_rq_seq_num1[5:0]` 上出现序列号之后，由该核发射的任何完成包均可确保在核的内部发射流水线中无法传递对应的转发请求。

请求器完成接口操作（512 位）

图 95：请求器完成接口信号



X16714-030223

下图显示了与核的请求器完成接口关联的信号。未启用跨接时，该核将此接口上的每个 TLP 均作为 1 个 AXI4-Stream 数据包来交付。对于含有效载荷的完成包，此数据包以 96 位描述符开头并后接数据。

请求器完成接口支持通过 2 种不同的数据对齐模式来进行有效载荷传输，此模式在 Vivado® IDE 中执行核自定义期间进行设置。在 Dword 对齐模式下，核会紧跟在描述符的最后一个 Dword 之后传输完成包有效载荷的第一个 Dword。在 128 位地址对齐模式下，核会在 512 位字的第 2 个 128 位四分之一拍开始有效载荷的传输，紧随第 1 个四分之一拍的描述符之后。有效载荷的第一个 Dword 可位于第 2 个四分之一拍中的 4 个可能的 Dword 位置中的任一位置，其偏移根据用户逻辑在向核发送请求（即设置请求器请求接口的 `addr_offset` 输入）时所提供的地址偏移来确定。因此，仅当请求器请求接口同样配置为使用 128 位地址对齐模式时，才能在请求器完成接口上使用 128 位地址对齐模式。

请求器完成描述符格式

核的请求器完成接口可将从链路接收到的数据作为 AXI4-Stream 数据包发送到用户应用。每个数据包均以 1 个描述符开头，在描述符后可包含有效载荷数据。描述符长度始终为 12 字节，并在完成包的前 12 字节内发送。当完成数据被拆分为多个拆分完成包后，核会将每个拆分完成包作为含专有描述符的独立 AXI4-Stream 数据包进行发送。

下图演示了请求器完成描述符的格式。下表描述了请求器完成描述符的每个字段。

图 96：请求器完成描述符格式

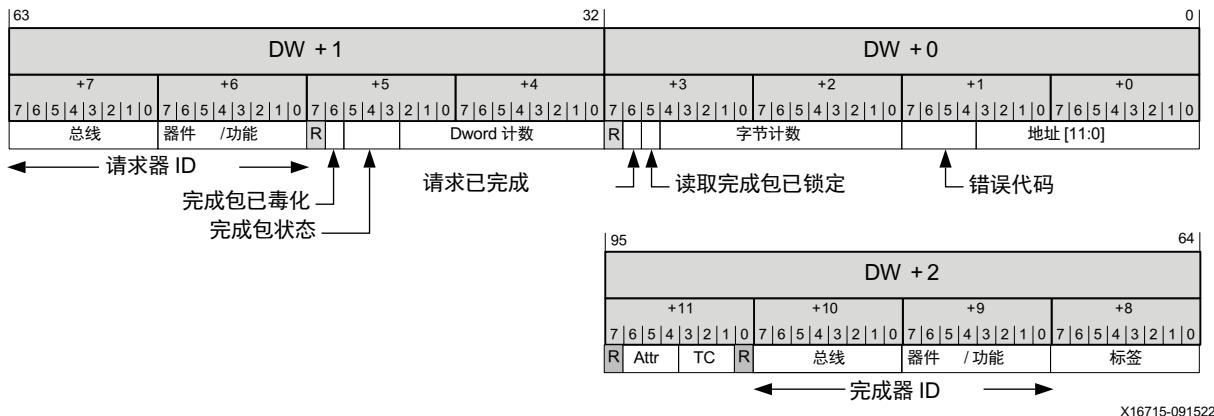


表 62：请求器完成描述符字段

位索引	字段名称	描述
11:0	“Lower Address” (低位地址)	<p>该字段可提供请求引用的第 1 个字节的 12 个最低有效位。核从其“Split Completion Table”（拆分完成包表）返回该地址，在该表中存储有请求器侧的所有暂挂非转发请求的地址和其它参数。</p> <p>当完成包交付错误时，应仅将该地址的位 [6:0] 视为有效。</p> <p>该地址为字节级别地址。</p> <p>对于 ATS 转换请求，该字段为保留字段，暗示其值应为 0。</p>
15:12	错误代码	<p>完成包错误代码。这 3 位代码表示核对接收到的完成包所执行的错误检查检测到存在错误状况。其编码为：</p> <ul style="list-style-type: none"> 0000：正常终止（已接收到所有数据）。 0001：完成 TLP 已被毒化。 0010：因出现状态为 UR、CA 或 CRS 的完成包，请求已被终止。 0011：因出现不含任何数据的完成包，或者完成包中的字节计数高于请求期望的字节总数，请求已被终止。 0100：当前交付的完成包所含标签与未完成请求的标签相同，但其 Requester ID、TC 或 Attr 字段与未完成请求的参数不匹配。 0101：起始地址中存在错误。完成 TLP 报头中的下位地址位与请求的下一个期望字节的起始位置不匹配。 0110：标签无效。此完成包与任意未完成请求的标签都不匹配。 1000：因出现以生成请求的功能为目标的功能级别复位 (FLR)，此请求已被终止。描述符中除位 [30]、请求器功能 [55:48] 和标签字段 [71:64] 以外的其它字段在此情况下都无效，因为该描述符不对应于完成 TLP。 1001：请求已终止，完成包超时。描述符中除位 [30]、请求器功能 [55:48] 和标签字段 [71:64] 以外的其它字段在此情况下都无效，因为该描述符不对应于完成 TLP。
28:16	“Byte Count” (字节计数)	<p>这 13 位可包含范围在 0 - 4,096 个字节内的值。如果存储器读取请求已通过使用单一完成包完成，那么“Byte Count”值将以字节为单位来表示“Payload”（有效载荷）大小。针对 I/O 读取完成包和 I/O 写入完成包，该字段必须设置为 4。针对长度为 0 的存储器读取发送完成包时，字节计数必须设置为 1，并且在描述符之后必须附加含单 Dword 的虚拟有效载荷。</p> <p>对于每个“Memory Read Completion”（存储器读取完成），“Byte Count”字段必须表明完成请求所需的剩余字节数，包括随完成包返回的字节数。</p> <p>如果存储器读取请求已使用多个完成包完成，那么后续每个完成包的字节计数值均表示为前一个完成包减去随前一个完成包返回的字节数。</p>

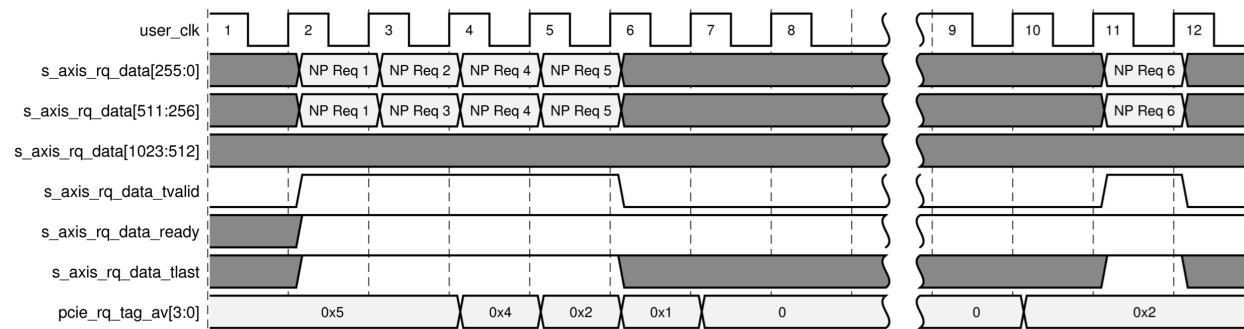
表 62：请求器完成描述符字段 (续)

位索引	字段名称	描述
29	“Locked Read Completion” (锁定读取完成)	当 “Locked Read” (锁定读取) 请求的响应中包含 “Completion” (完成包) 时，该位设置为 1。针对所有其它完成包，该位均设置为 0。
30	“Request Completed” (请求完成)	核在请求的最后一个完成包的描述符中断言该位有效。断言该位有效表示请求正常终止 (因为已接收到所有数据) 或者异常终止 (因为存在错误状况)。用户逻辑可使用该指示来清除其未完成的请求。 用户逻辑分配标签时，用户逻辑应在接收到来自核的完成描述符并且其中包含匹配的标签字段且 “Request Completed” 位已设置为 1 之后，才能将标签重新分配到请求。
42:32	“Dword Count” (Dword 计数)	这 11 位表示当前包的有效载荷大小 (以 Dword 数为单位)。其范围是 0 - 1000 个 Dword。针对 I/O 读取完成包，该字段设置为 1，针对 I/O 写入完成包，该字段设置为 0。针对长度为 0 的存储器读取传输完成包时，Dword 计数同样设置为 1。在所有其它情况下，Dword 计数都对应于当前包的有效载荷中 Dword 的实际数量。
45:43	“Completion Status” (完成状态)	这些位可反映接收到的完成 TLP 的 “Completion Status” (完成状态) 字段的设置。有效设置包括： <ul style="list-style-type: none">· 000: 成功完成。· 001: 请求不受支持 (UR)。· 010: 配置请求重试状态 (CRS)。· 100: 完成器异常中止 (CA)。
46	“Poisoned Completion” (毒化完成)	设置该位可指示完成 TLP 中已设置 “Poison” (毒化) 位。在此情况下，应将数据包中的数据视为已损坏。
63:48	“Requester ID” (请求器 ID)	与完成包关联的 PCI 请求器 ID。
71:64	“Tag” (标签)	与完成包关联的 PCIe 标签。
87:72	“Completer ID” (完成器 ID)	在完成 TLP 中接收到的完成器 ID。(这 16 个位在传统解读模式下分割为 8 位总线编号、5 位器件编号和 3 位功能编号。而在 ARI 模式下，这 16 个位则必须作为 8 位总线编号 + 8 位功能编号来处理。)
91:89	“Transaction Class (TC)” (传输事务类)	与完成包关联的 PCIe 传输事务类 (TC)。
94:92	属性	与完成包关联的 PCIe 属性。位 92 为 “No Snoop” (无嗅探) 位，位 93 为 “Relaxed Ordering” (宽松排序) 位，位 94 则为保留位。

不含数据的完成包的传输

下图显示了从不含关联有效载荷的链路接收到完成 TLP 后，通过请求器完成接口来进行传输的过程。本章节中的时序图假定在接口上，完成 (Completion) 包不存在跨接。在 [RC 接口的跨接选项](#) 中对跨接功能进行了描述。

图 97：请求器完成接口上不含数据的完成包的传输



完成 TLP 的整个传输过程在该接口上只需一拍。核会在数据包的持续时间段内保持 `m_axis_rc_tvalid` 信号处于断言有效状态。用户逻辑可以随时通过下拉 `m_axis_rc_tready` 来延长任一节拍。AXI4-Stream 接口可发出 `m_axis_rc_tkeep` 信号（每个信号对应 1 个 Dword 位置）以指示数据包中的有效描述符 Dword。即，从描述符的第一个 Dword 开始到最后一个 Dword 为止的所有 `m_axis_rc_tkeep` 位均连续设置为 1。`m_axis_rc_tlast` 信号始终处于断言有效状态，以表示此数据包将在其当前节拍内终止。

`m_axi_rc_tuser` 总线还包含 `is_sop[0]` 信号，在每个数据包的第一拍内，此信号均断言有效。用户逻辑可以选择使用此信号来限定接口上描述符的起始位置。当不使用跨接选项时，`m_axi_rc_tuser` 中的所有其它 sop 和 eop 指示都与完成包的传输无关。

含数据完成包的传输

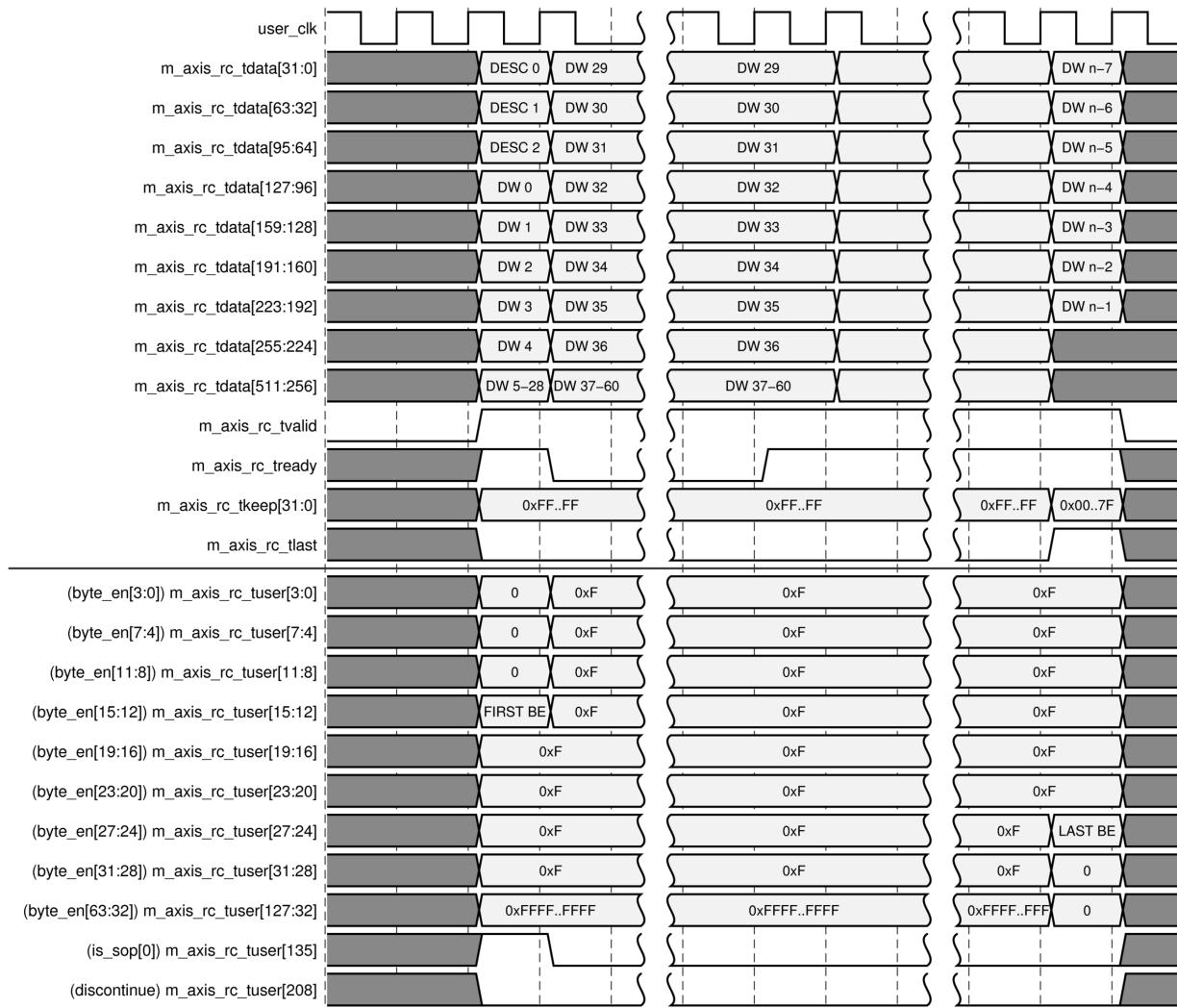
在 Dword 对齐模式下，传输从 3 个描述符 Dword 开始，紧随其后即为有效载荷 Dword。包括描述符和有效载荷在内的完整 TLP 作为单一 AXI4-Stream 数据包进行传输。当有效载荷长度超过 2 个 Dword 时，有效载荷内的数据始终为连续字节数据流。这样即可根据请求完成描述符的“Lower Address”（下位地址）字段和“Byte Count”（字节计数）字段来判定有效载荷的第一个 Dword 内的第一个有效字节的位置以及最后一个 Dword 的最后 1 个有效字节的位置。当有效载荷大小不超过 2 个 Dword 时，有效载荷中的有效字节不得连续。在此类情况下，用户逻辑必须存储与通过请求器请求接口向外发送的每个请求关联的“First Byte Enable”（首字节使能）字段和“Last Byte Enable”（末字节使能）字段，并将其用于判定完成有效载荷内的有效字节。用户逻辑可选择使用 `m_axi_rc_tuser` 总线中的字节使能输出 `byte_en[63:0]` 来判定有效载荷中的有效字节，在连续有效载荷和非连续有效载荷情况下都是如此。

核会在数据包的整个持续时间段内保持 `m_axis_rc_tvalid` 信号处于断言有效状态。用户逻辑可以随时通过下拉 `m_axis_rc_tready` 来延长任一节拍。AXI4-Stream 接口可发出 `m_axis_rc_tkeep` 信号（每个信号对应 1 个 Dword 位置）以指示数据包中的有效 Dword，包括描述符和描述符与有效载荷之间插入的任意空字节。即，从描述符的第一个 Dword 开始到有效载荷的最后一个 Dword 为止的所有 `tkeep` 位均连续设置为 1。在数据包传输期间，当数据包无法填满接口的完整宽度时，`m_axis_rc_tkeep` 位仅限在数据包的最后一拍内才能设为 0。在数据包的最后一拍内，`m_axis_rc_tlast` 信号始终断言有效。

`m_axi_rc_tuser` 总线可提供多个可选信号，这些可选信号可用于简化与接口的用户侧关联的逻辑，或者用于支持其它功能。在每个数据包的第一拍内，`is_sop[0]` 信号都断言有效（前提是其描述符位于总线上）；当不使用跨接选项时，`m_axi_rc_tuser` 中的所有其它 sop 和 eop 都与完成包传输无关。字节使能输出 `byte_en[63:0]`（每个输出对应 1 个字节通道）可指示有效载荷中的有效字节。仅当对应通道内包含的有效载荷字节确实有效时，这些信号才会断言有效，而针对描述符或空字节，则不会断言该信号有效。断言有效的字节使能位从有效载荷开始始终连续，除非有效载荷大小不超过 2 个 Dword。当完成包的有效载荷不超过 2 个 Dword 时，`byte_en` 上值为 1 的位不连续。另一种特殊情况是对于长度为 0 的存储器读取，核会传输含单 Dword 的有效载荷，其中所有 `byte_en` 位均设为 0。因此，无论在任何情况下，用户逻辑均可直接使用 `byte_en` 信号来启用将关联字节写入存储器的操作。

下图显示了从含关联有效载荷的链路接收到完成 TLP（采用 Dword 对齐模式）后，通过请求器完成接口来进行传输的过程。为便于演示，写入用户存储器的数据块的大小假定为 n 个 Dword，其中， $n = k * 16 + 4$ 且 $k > 1$ 。本章节中的时序图假定在接口上，完成 (Completion) 包不存在跨接。在 [RC 接口的跨接选项](#) 中对跨接功能进行了描述。

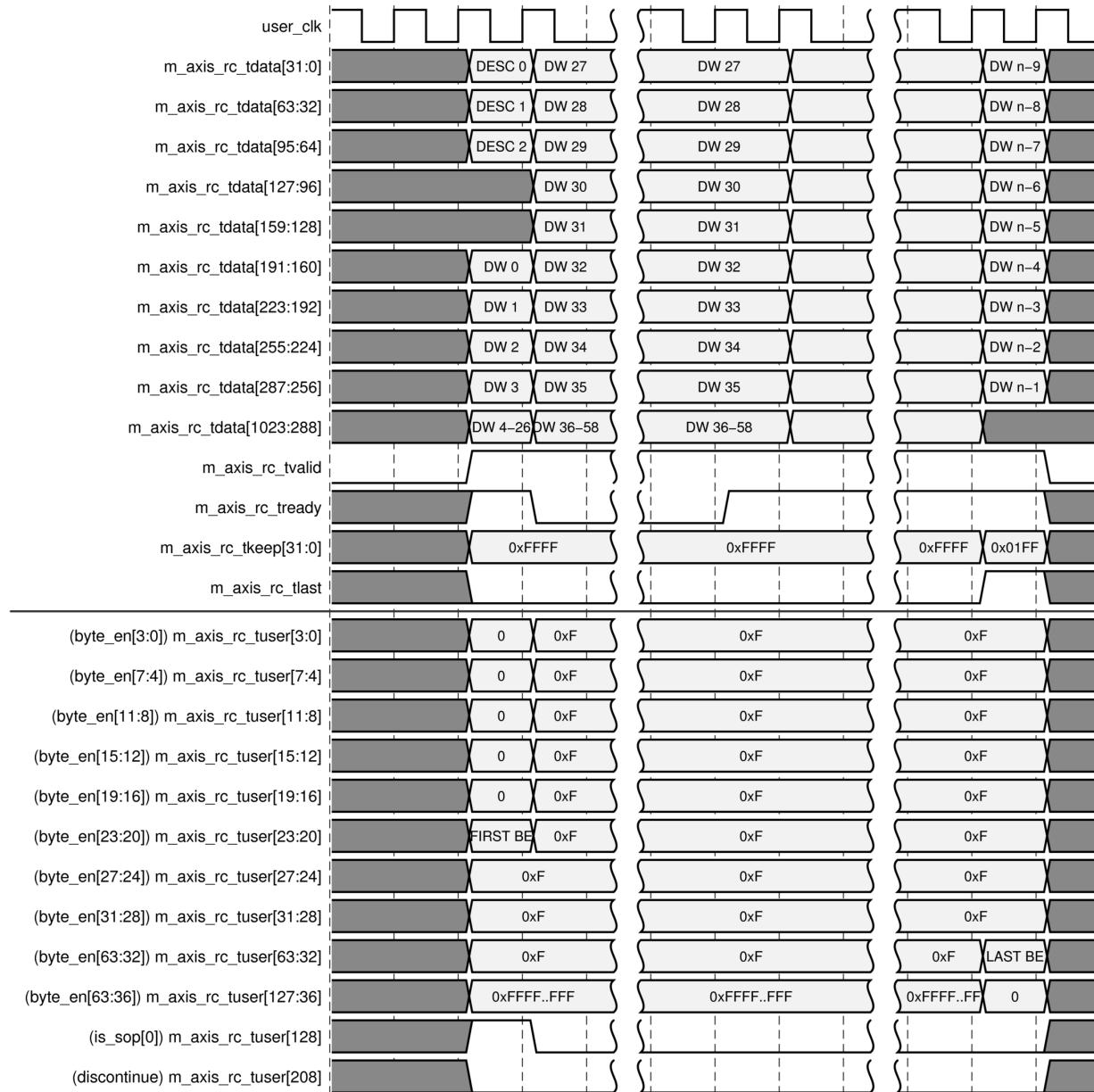
图 98：请求器完成接口上含数据完成包的传输（Dword 对齐模式）



下图显示了从含关联有效载荷的链路接收到完成 TLP（采用地址对齐模式）后，通过请求器完成接口来进行传输的过程。在时序图示例中，要传输的数据块的起始 Dword 地址（如描述符中的下位地址字段中所述）假定为 $(m * 16 + 1)$ ，其中 m 为整数。数据块的大小假定为 n 个 Dword，其中 $n = k * 16 + 4$ 且 $k > 0$ 。针对 128 位地址对齐传输，跨接选项无效，因此时序图假定接口上完成包不存在跨接。

在 128 位对齐模式下，有效载荷的交付始终从描述符的最后一个字节后的节拍中开始。根据有效载荷的第一个有效字节的地址，有效载荷的第一个字节可显示在字节通道 16 - 32 范围内的任意通道上。从描述符的第一个 Dword 开始到有效载荷的最后一个 Dword 为止的所有 m_axis_rc_tkeep 位均连续设置为 1。数据总线上 128 位字段范围内的第一个 Dword 的对齐方式是根据当用户应用向核发送请求时，请求器请求接口的 addr_offset[1:0] 输入的设置来确定的。用户应用可选择使用字节使能输出 byte_en[63:0] 来判定有效载荷中的有效字节。

图 99：请求器完成接口上含数据完成包的传输（128 位接口、地址对齐模式）



RC 接口的跨接选项

PCIe® 核的 RC 接口具备在请求器完成接口上在同一拍内启动 4 个完成 (Completion) 包的能力。在 Vivado® IDE 中进行核自定义期间可启用此跨接选项。跨接选项只能配合 Dword 对齐模式一起使用。

当启用跨接选项时，在 AXI4-Stream 接口上，完成 TLP 将作为无数据包边界的连续数据流进行传输。因此，在判定接口上交付的完成 TLP 的边界过程中不使用 `m_axis_rc_tkeep` 信号和 `m_axis_rc_tlast` 信号（使用跨接选项时，核会永久性将 `m_axis_rc_tkeep` 的位全部设置为 1，将 `m_axis_rc_tlast` 全部设置为 0）。并改为使用 `m_axis_rc_tuser` 总线中所提供的以下信号来执行 TLP 的界定。

- `is_sop[3:0]`: 当有至少 1 个完成 TLP 从节拍中开始时，在该节拍中，核就会将此输出设置为非 0 值。禁用跨接时，仅限 `is_sop[0]` 有效，`is_sop[3:1]` 则永久设置为 0。启用跨接时，设置如下：
 - 0000: 没有新 TLP 在此节拍中起始
 - 0001: 有单一新 TLP 在此节拍中起始。其起始位置由 `is_sop0_ptr[1:0]` 来表示。
 - 0011: 有 2 个新 TLP 在此节拍中起始。`is_sop0_ptr[1:0]` 提供第 1 个 TLP 的起始位置，`is_sop1_ptr[1:0]` 则提供第 2 个 TLP 的起始位置。
 - 0111: 有 3 个新 TLP 在此节拍中起始。`is_sop0_ptr[1:0]` 提供第 1 个 TLP 的起始位置，`is_sop1_ptr[1:0]` 提供第 2 个 TLP 的起始位置，`is_sop2_ptr[1:0]` 则提供第 3 个 TLP 的起始位置。
 - 1111: 有 4 个新 TLP 在此节拍中起始。`is_sop0_ptr[1:0]` 提供第 1 个 TLP 的起始位置，`is_sop1_ptr[1:0]` 提供第 2 个 TLP 的起始位置，`is_sop2_ptr[1:0]` 提供第 3 个 TLP 的起始位置，`is_sop3_ptr[1:0]` 则提供第 4 个 TLP 的起始位置。
 - 所有其它设置均保留。
- `is_sop0_ptr[1:0]`: 设置 `is_sop[0]` 时，该字段表示当前节拍中起始的第 1 个完成 TLP 的偏移。有效设置包括 2'b00 (TLP 起始位置为 Dword 0)、2'b01 (TLP 起始位置为 Dword 4)、2'b10 (TLP 起始位置为 Dword 8) 和 2'b11 (TLP 起始位置为 Dword 12)。
- `is_sop1_ptr[1:0]`: 设置 `is_sop[1]` 时，该字段表示当前节拍中起始的第 2 个完成 TLP 的偏移。有效设置包括 2'b01 (TLP 起始位置为 Dword 4)、2'b10 (TLP 起始位置为 Dword 8) 和 2'b11 (TLP 起始位置为 Dword 12)。
- `is_sop2_ptr[1:0]`: 设置 `is_sop[2]` 时，该字段表示当前节拍中起始的第 3 个完成 TLP 的偏移。有效设置包括 2'b10 (TLP 起始位置为 Dword 8) 和 2'b11 (TLP 起始位置为 Dword 12)。
- `is_sop3_ptr[1:0]`: 设置 `is_sop[3]` 时，该字段表示当前节拍中起始的第 4 个完成 TLP 的偏移。其唯一有效设置为 2'b11 (TLP 起始位置为 Dword 12)。
- `is_eop[3:0]`: 这些输出表示有 1 个或多个 TLP 在此节拍中结束。这些输出在 TLP 的最后一个节拍中设置。禁用跨接时，仅限 `is_eop[0]` 有效，`is_eop[3:1]` 则永久设置为 0。启用跨接时，设置如下：
 - 0000: 没有任何 TLP 在此节拍中结束。
 - 0001: 只有 1 个 TLP 在此节拍中结束。`is_eop0_ptr[3:0]` 的设置可提供此 TLP 的最后一个 Dword 的偏移。
 - 0011: 有 2 个 TLP 在此节拍中结束。`is_eop0_ptr[3:0]` 提供第 1 个 TLP 的最后一个 Dword 的偏移，`is_eop1_ptr[3:0]` 则提供第 2 个 TLP 的最后一个 Dword 的偏移。
 - 0111: 有 3 个 TLP 在此节拍中结束。`is_eop0_ptr[3:0]` 提供第 1 个 TLP 的最后一个 Dword 的偏移，`is_eop1_ptr[3:0]` 提供第 2 个 TLP 的最后一个 Dword 的偏移，`is_eop2_ptr[3:0]` 则提供第 3 个 TLP 的最后一个 Dword 的偏移。
 - 1111: 有 4 个 TLP 在此节拍中结束。`is_eop0_ptr[3:0]` 提供第 1 个 TLP 的最后一个 Dword 的偏移，`is_eop1_ptr[3:0]` 提供第 2 个 TLP 的最后一个 Dword 的偏移，`is_eop2_ptr[3:0]` 提供第 3 个 TLP 的最后一个 Dword 的偏移，`is_eop3_ptr[3:0]` 则提供第 4 个 TLP 的最后一个 Dword 的偏移。
 - 所有其它设置均保留。
- `is_eop0_ptr[3:0]`: 设置 `is_eop[0]` 时，该字段提供在此节拍中结束的第 1 个 TLP 的最后一个 Dword 的偏移。有效值为 0 到 15 (含) 范围内的任意值。最后一个字节的偏移可根据 TLP 的起始地址和长度来确定，或者根据字节使能信号 `byte_en[63:0]` 来确定。
- `is_eop1_ptr[3:0]`: 设置 `is_eop[1]` 时，该字段提供在此节拍中结束的第 2 个 TLP 的最后一个 Dword 的偏移。有效值为 6 到 15 (含) 范围内的任意值。
- `is_eop2_ptr[3:0]`: 设置 `is_eop[2]` 时，该字段提供在此节拍中结束的第 3 个 TLP 的最后一个 Dword 的偏移。有效值为 10 到 15 (含) 范围内的任意值。

- `is_eop3_ptr[3:0]`: 设置 `is_eop[3]` 时，该字段提供在此节拍中结束的第 4 个 TLP 的最后一个 Dword 的偏移。有效值为 14 或 15。

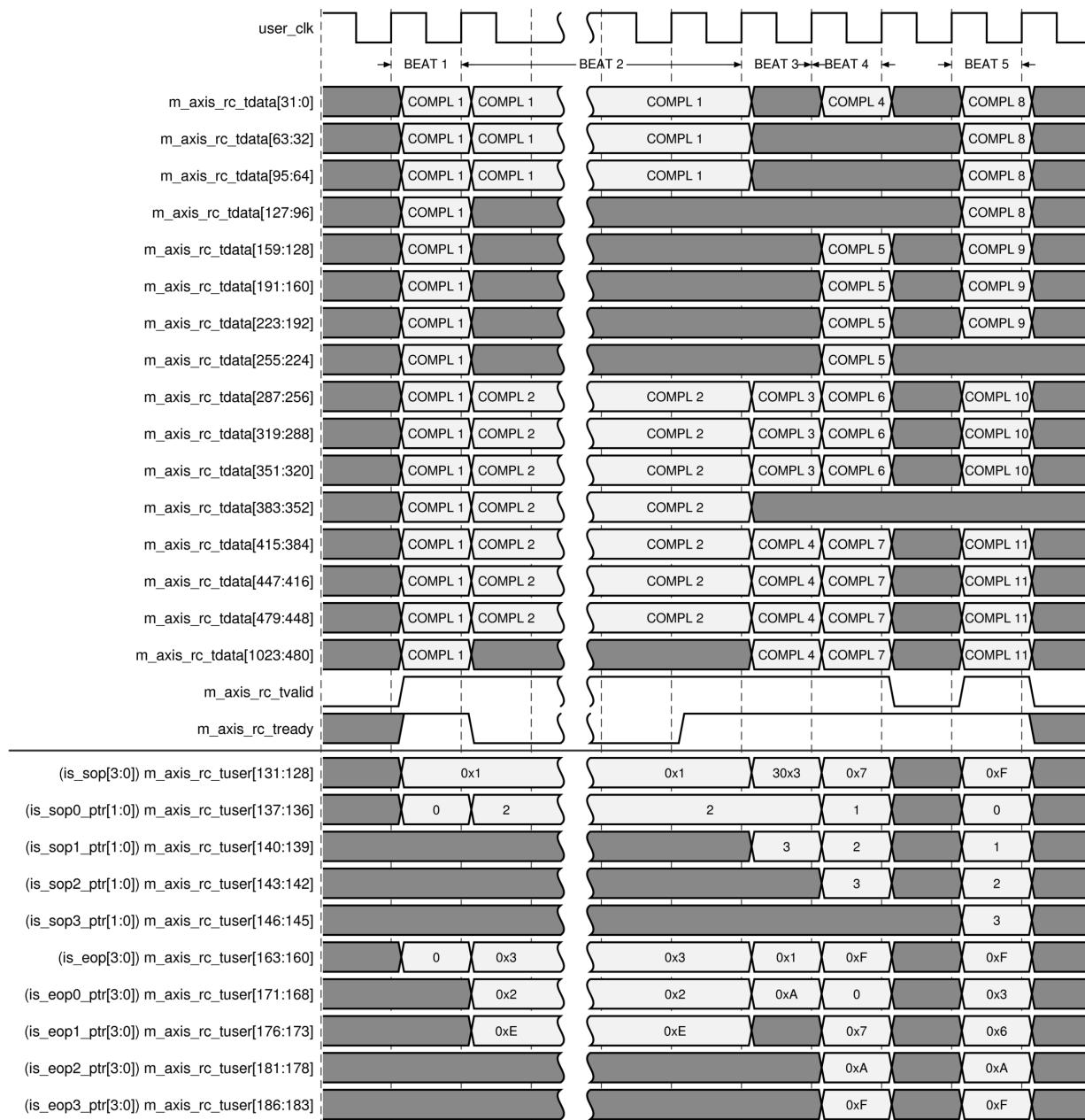
下图显示了启用跨接选项的情况下，在请求器完成接口上传输 11 个完成 TLP 的过程。第 1 个完成 TLP (COMPL 1) 从节拍 1 的 Dword 位置 0 开始，并在节拍 2 的 Dword 位置 2 结束。第 2 个 TLP (COMPL 2) 从同一拍内的 Dword 位置 8 开始，并在 Dword 位置 14 结束。因此，在节拍 1 中，有 1 个 TLP 起始，其起始位置由 `is_sop0_ptr` 来指示，并有 2 个 TLP 结束，其结束 Dword 位置分别由 `is_eop0_ptr` 和 `is_eop1_ptr` 来指示。

节拍 3 包含 COMPL 3，起始位置为 Dword 偏移 8 且结束位置为 Dword 偏移 10。在同一拍中还有第 2 个 TLP (CMPL 4)，其起始位置为 Dword 偏移 12 并持续直至下一拍。在此节拍中，`is_sop0_ptr` 指向 COMPL 3 的起始 Dword 偏移位置，`is_sop1_ptr` 则指向 COMPL 4 的起始 Dword 偏移位置。`is_eop0_ptr` 指向 COMPL 4 的最后一个 Dword 偏移的位置。

节拍 4 包含 COMPL 4，其结束位置为 Dword 偏移 0，其中还包含 3 个新的完成 TLP（分别为 COMPL 5、6 和 7）。新的完成 5、6 和 7 的起始 Dword 偏移位置分别由 `is_sop0_ptr`、`is_sop1_ptr` 和 `is_sop2_ptr` 提供。完成 4、5、6 和 7 的结束偏移位置则分别由 `is_eop0_ptr`、`is_eop1_ptr`、`is_eop2_ptr` 和 `is_eop3_ptr` 来指示。

最后，节拍 5 包含 4 个完成 TLP (COMPL 8 - 11)。其起始 Dword 偏移位置分别由 `is_sop0_ptr`、`is_sop1_ptr`、`is_sop2_ptr` 和 `is_sop3_ptr` 来指示。结束偏移位置则分别由 `is_eop0_ptr`、`is_eop1_ptr`、`is_eop2_ptr` 和 `is_eop3_ptr` 来指示。因此，全部 4 个 SOP 和 EOP 指针均可在此节拍内提供有效信息。

图 100：在启用跨接选项的请求器完成接口上执行完成 TLP 的传输



完成包传输异常中止

对于包含关联有效载荷的任何完成包，核会在传输的有效载荷中发出错误信号，方法是在数据包的最后一拍中将 `m_axis_rc_tuser` 总线中的 `discontinue` 信号断言有效。当核在读取其内部存储器中的数据时，如果检测到不可纠正的错误，则会发生此操作。当用户应用在数据包的最后一拍中检测到 `discontinue` 信号已断言有效时，必须丢弃整个数据包。

使用跨接选项时，核如果已断言 `discontinue` 有效并导致此节拍内结束的完成 TLP 异常中止，则不会在同一拍内启动新的完成 TLP。

完成错误的处理

当从链路接收到完成 TLP 时，核会将其与拆分完成包表 (Split Completion Table) 中未完成的请求进行比对以判定对应的请求，并将其报头中的字段与期望的值进行比对以检测是否存在任何错误状况。随后，核会发出信号，以 4 位错误代码形式将错误状况包含在完成描述符中，一并发送给用户逻辑。核还会通过在描述符中设置“Request Completed”(请求完成) 位 (位 30) 来指示请求的最后完成状态。错误状态以各种错误代码来表示，如下所述：

- 0010：因出现状态为 UR、CA 或 CRS 的完成 TLP，请求已被终止。在此情况下，不存在与完成包关联的数据，并且完成描述符中已设置请求完成位。从核接收到此类完成后，用户逻辑即可丢弃对应请求。
- 0011：因完成 TLP 字节计数不正确，此读取请求已被终止。当接收到的完成 TLP 的字节计数与期望的计数不匹配时，就会发生此状况。完成描述符中已设置请求完成位。从核接收到此类完成后，用户逻辑即可丢弃对应请求。
- 0100：此代码对应状况如下：当前交付的完成包所含标签与未完成请求的标签相同，但其 Requester ID、TC 或 Attr 字段与未完成请求的参数不匹配。用户逻辑应丢弃位于描述符之后的所有数据。此外，如果在描述符中未设置请求完成位，那么用户逻辑应继续丢弃于此标签完成之后接收的数据，直至它接收到已设置请求完成位的完成描述符为止。接收到已设置请求完成位的完成描述符后，用户逻辑即可移除与该请求关联的所有状态。
- 0101：起始地址中存在错误。完成 TLP 报头中的下位地址位与请求的下一个期望字节的起始位置不匹配。用户逻辑应丢弃位于描述符之后的所有数据。此外，如果在描述符中未设置请求完成位，那么用户逻辑应继续丢弃于此标签完成之后接收的数据，直至它接收到已设置请求完成位的完成描述符为止。接收到已设置请求完成位的完成描述符后，用户逻辑即可丢弃对应请求。
- 0110：标签无效。此错误代码指示完成 TLP 中的标签与任意未完成请求的标签都不匹配。用户逻辑应丢弃位于描述符之后的所有数据。
- 1000：请求已终止，完成包超时。当未完成的请求超时且未从链路接收到完成包时，会使用此错误代码。核会为每个未完成的请求保留完成定时器，并通过在请求器完成接口上向用户逻辑发射虚拟完成描述符来响应完成超时，因此，用户逻辑可以终止暂挂的请求或者重试该请求。由于此描述符不对应于从链路接收到的完成 TLP，因此，在此描述符中只有请求完成位 (位 30)、标签字段 (位 [71:64]) 和请求器功能字段 (位 [55:48]) 有效。
- 1000：因出现以生成请求的功能为目标的功能级别复位 (FLR)，此请求已被终止。在此情况下，核会在请求器完成接口上向用户逻辑发射虚拟完成描述符，因此用户逻辑可终止暂挂请求。由于此描述符不对应于从链路接收到的完成 TLP，因此，在此描述符中只有请求完成位 (位 30)、标签字段 (位 [71:64]) 和请求器功能字段 (位 [55:48]) 有效。

当标签由核进行内部管理时，核中的逻辑可确保在接收到对应请求的所有完成包或者当请求超时之后，才能复用分配到暂挂请求的标签。但当标签由用户逻辑管理时，用户逻辑必须确保先由核通过在完成描述符中设置请求完成位的方式来发送请求终止信号，然后才能复用分配给请求的标签。用户逻辑可在接收到含非 0 错误代码的完成时结束暂挂请求，但如果在完成描述符中未设置请求完成位，则不应释放关联标签以作他用。当请求接收到多个拆分完成包，并且其中任一信号包含错误时，会发生此状况。在此情况下，核可以继续接收暂挂请求的完成 TLP，即使检测到错误也是如此，并且如果过早重新分配该请求的标签，则会导致这些完成包与其它请求出现匹配错误。请注意，在某些情况下，核可能需要等待请求超时后才能允许复用标签，即使已接收到含错误的拆分完成包也是如此。

注释：每个奇偶校验位都对应于 AXIS_tdata 中的 1 个字节的奇偶校验。有多个 64 位奇偶校验位对应于 512 位 AXI_tdata (另有多个 32 位奇偶校验位对应于 256 位 AXI_tdata)。m_axis_cq_tuser 信号和 m_axis_rc_tuser 信号上接收到的奇偶校验位对于以下字节有效：

- AXIS_tdata 中的描述符字节。
- AXIS_tdata 中有效的有效载荷字节 (由 AXIS_tuser 中的 byte_en 字段来表示)。例如，如果 byte_en[63:0]=0x0000_0000_0000_FFFF，那么仅限下 16 位奇偶校验位有效。如果 byte_en[63:0] = 0xFFFF_FFFF_FFFF_FFFF，则启用全部 64 个奇偶校验位。

功耗管理

核支持下列功耗管理模式：

- 活动状态功耗管理 (ASPM)
- 编程功耗管理 (PPM)

在 PCI Express 设计中实现这些功耗管理功能可使 PCI Express® 层级无缝交换功耗管理报文，从而节省系统功耗。所有功耗管理报文标识功能均已实现。本章中分各小节来描述支持上述功耗管理模式的用户逻辑定义。

如需获取有关 ASPM 和 PPM 实现的更多信息，请参阅《PCI Express 基本规范》。

活动状态功耗管理

核会通过播发 N_FTS 值 255 来确保在退出 L0s 时正确对齐。如果修改 N_FTS 值，那么必须确保接收到足够的 FTS 序号，以确保正确对齐并避免转换为“Recovery”（恢复）状态。

活动状态功耗管理 (ASPM) 功能可自主运行，从用户逻辑功能的角度来看，它处于透明状态。核支持 ASPM 所要求的条件。集成块支持 ASPM L0s 和 ASPM L1。L0s 和 L1 不应并行启用。

- ASPM 在非同步时钟模式下不受支持。
- L0s 仅在为 Gen1 和 Gen2 生成的设计上并且仅限端点模式下才受支持。
- 启用 ASPM L0s/ASPM L1 可显示两个链路伙伴在链路上报告可纠正错误（例如，重放定时器超时、重放定时器滚动或接收器错误）。赛灵思建议，应用禁用可纠正错误报告或者忽略发起 ASPM L0s/ASPM L1 转换时报告的可纠正错误。

编程功耗管理

为了在 PCI Express® 层级树上大幅节省功耗，核支持下列编程功耗管理 (PPM) 链路状态：

- L0：活动状态（数据交换状态）
- L1：在此状态下，时延更高，待机功耗更低
- L3：链路中断状态

编程功耗管理协议由下游组件/上游端口启动。

PPM L0 状态

L0 状态表示正常运行，对于用户逻辑，此状态是透明的。当核根据协议成功完成 PCI Express® 链路初始化和训练后，核即达到 L0（活动状态）。

PPM L1 状态

下列步骤用于描述核转换为 PPM L1 状态的过程：

1. 转换为低功耗 PPM L1 状态的操作始终由上游器件发起，方法是将 PCI Express® 器件功耗状态编程为 D3-hot（或者如果受支持，可编程为 D1 或 D2 状态）。
2. 器件功耗状态将通过 `cfg_function_power_state` 输出告知用户逻辑。
3. 随后，核会对用户逻辑进行节流或暂停其在用户接口上发起任何新的传输事务，方法是断言 `s_axis_rq_tready` 无效。但用户接口上任何暂挂的传输事务都会被完全接受，并且可稍后完成。
 - 核配置为端点，且启用用户配置空间。在此情况下，如果 `cfg_function_power_state` 指示非 D0 结果，那么用户应用必须停止发送新的请求 TLP，但是用户应用可以向目标为用户配置空间的配置传输事务返回完成包。

- 核配置为根端口。为了在此情况下仍保持合规，如果 `cfg_function_power_state` 指示非 D0 结果，那么用户应用必须停止发送新请求。
4. 该核会与其链路伙伴交换相应的功耗管理 DLLP，以成功完成将链路转换为低功耗 PPM L1 状态的操作。此操作对于用户逻辑而言，属于透明操作。
 5. 在器件功耗状态不为 D0 的时间段内暂停所有用户传输事务，但步骤 3 中所述例外情况除外。

PPM L3 状态

下列步骤用于描述 PCI Express® 端点转换为 PPM L3 状态的过程：

1. 在接收到来自上游链路伙伴的 PME_Turn_Off 报文时，核会通过协商来转换为 L23 就绪链路状态。
2. 接收到 PME_Turn_Off 报文时，核会通过 `cfg_power_state_change_interrupt` 与用户逻辑发起握手（如下表所示），并期望用户逻辑返回 `cfg_power_state_change_ack`。
3. 握手成功将导致核向其上游链路伙伴发射功耗管理关闭确认 (PME-turnoff_ack) 报文。
4. 核会关闭其所有接口、禁用物理层/数据链路层/传输事务层，并准备好对核进行断电。

此规则存在下列 2 个例外：

- 核配置为端点，且启用用户配置空间。在此情况下，如果 `cfg_function_power_state` 指示非 D0 结果，那么用户应用必须停止发送新的请求 TLP，但是用户应用可以向目标为用户配置空间的配置传输事务返回完成包。
- 核配置为根端口。为了在此情况下仍保持合规，如果 `cfg_function_power_state` 指示非 D0 结果，那么用户应用必须停止发送新请求。

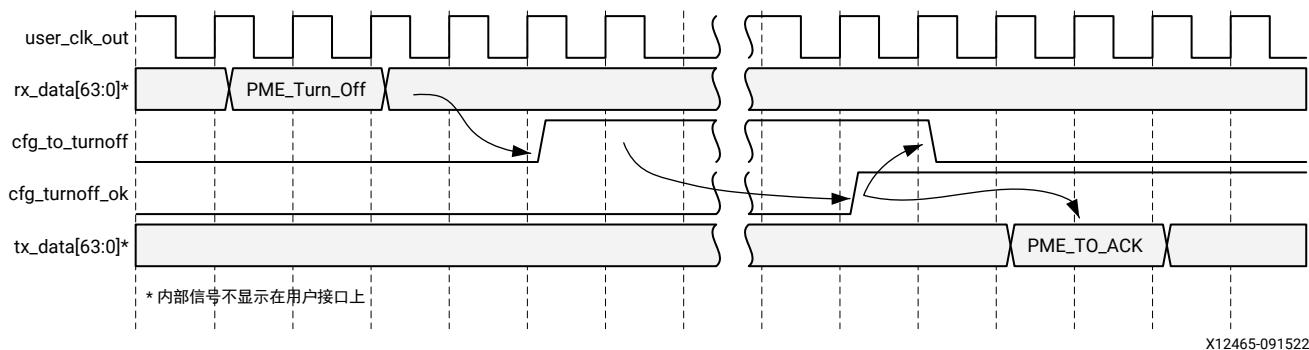
表 63：功耗管理握手信号

端口名称	方向	描述
<code>cfg_power_state_change_interrupt</code>	输出	如果接收到来自上游器件的掉电请求 TLP，则断言此端口有效。断言有效后， <code>cfg_power_state_change_interrupt</code> 将保持断言有效状态，直至用户应用断言 <code>cfg_power_state_change_ack</code> 有效为止。
<code>cfg_power_state_change_ack</code>	输入	当可安全掉电时，由用户应用断言此端口有效。

掉电协商遵循如下步骤：

1. 在电源和时钟关闭前，下游开关中的根联合体或热插拔控制器会发出 PME_Turn_Off 广播报文。
2. 当核接收到此 TLP 时，它会向用户应用断言 `cfg_power_state_change_interrupt` 有效，并开始轮询 `cfg_power_state_change_ack` 输入。
3. 当用户应用检测到 `cfg_to_turnoff` 断言有效时，它必须完成正在进行中的所有数据包，并停止生成任何新的数据包。当用户应用准备好关闭时，它会向核断言 `cfg_power_state_change_ack` 有效。断言 `cfg_power_state_change_ack` 有效后，用户应用将落实关闭。
4. 核检测到 `cfg_power_state_change_ack` 断言有效后，就会发送 PME_TO_Ack 报文。

图 101：功耗管理握手：64 位



生成中断请求

请参阅 [配置中断控制器接口](#) 的表格中的 `cfg_interrupt_msi_*` 和 `cfg_interrupt_msix_*` 描述。

注释：本章节仅适用于 Integrated Block for PCIe 核的端点配置。

该核支持将中断请求作为遗留中断、报文 MSI 中断或 MSI-X 中断来进行发送。此模式是使用“MSI Capability Structure”（MSI 功能结构）的“Message Control register”（报文控制寄存器）中的“MSI Enable”（MSI 启用）位和“MSI-X Capability Structure”（MSI-X 功能结构）的“MSI-X Message Control register”（MSI-X 报文控制寄存器）中的“MSI-X Enable”（MSI-X 启用）位来进行编程的。

“MSI Enable”位和“MSI-X Enabled”位分别由 `cfg_interrupt_msi_enable` 输出和 `cfg_interrupt_msix_enable` 输出来反映。下表描述了根据核的 `cfg_interrupt_msi_enable` 输出和 `cfg_interrupt_msix_enable` 输出，对器件进行编程后的对应“Interrupt Mode”（中断模式）。

表 64：中断模式

	<code>cfg_interrupt_msixenable=0</code>	<code>cfg_interrupt_msixenable=1</code>
<code>cfg_interrupt_msi_enable = 0</code>	遗留中断 (INTx) 模式。 <code>cfg_interrupt</code> 接口仅发送 INTx 报文。	MSI-X 模式。 可使用 <code>cfg_interrupt</code> 接口生成 MSI-X 中断。
<code>cfg_interrupt_msi_enable = 1</code>	MSI 模式。 <code>cfg_interrupt</code> 接口仅发送 MSI 中断 (MWrr TLP)。	未定义。 系统软件不允许此模式。但如果选择此模式，那么 <code>cfg_interrupt</code> 接口处于有效状态并发送 MSI 中断 (MWrr TLP)。

MSI 控制寄存器中的“MSI Enable”位、MSI-X 控制寄存器中的“MSI-X Enable”位以及 PCI 命令寄存器中的“Interrupt Disable”（中断禁用）位均由根联合体进行编程。用户应用对于这些位不具有直接控制权。

核中的“Internal Interrupt Controller”（内部中断控制器）仅生成遗留中断和 MSI 中断。MSI-X 中断需由用户应用生成，并显示在 AXI4-Stream 发射接口上。`cfg_interrupt_msi_enable` 的状态可用于判定内部中断控制器生成的中断类型：

如果“MSI Enable”位设置为 1，那么核会通过发送存储器写入 TLP 来生成 MSI 请求。如果“MSI Enable”位设置为 0，那么核会生成遗留中断报文，前提是 PCI 命令寄存器中的中断禁用位设置为 0。

- `cfg_interrupt_msi_enable = 0`: 遗留中断

- `cfg_interrupt_msi_enable = 1: MSI`
- 命令寄存器位 $10 = 0$: 启用 INTx 中断
- 命令寄存器位 $10 = 1$: 禁用 INTx 中断 (核会阻止请求)

用户应用可监控 `cfg_function_status` 以检查 INTx 中断处于启用还是禁用状态。如需了解更多信息，请参阅 [配置状态接口](#)。

该核可配置为播发多种中断模式支持，但在运行时，每次所有功能间仅限启用一种中断模式。赛灵思不建议一次性启用多种中断模式，但如果无法避免同时启用 MSI 和 MSI-X 中断，那么必须在该核外部实现 MSI-X 中断，并通过请求器请求接口端口 (`s_axis_rq`) 来构成和发送中断包。

用户应用可通过两种方式之一来请求中断服务，下一章中对这两种方式分别进行了描述。

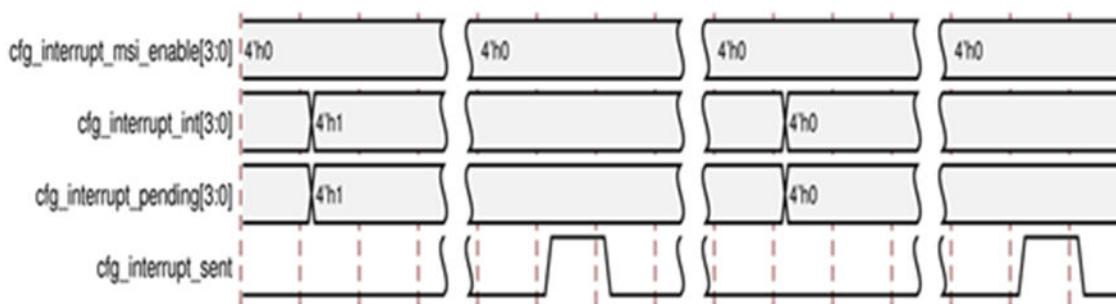
遗留中断模式

- 用户应用首先断言 `cfg_interrupt_int` 和 `cfg_interrupt_pending` 有效，以断言中断有效。
- 然后，核会断言 `cfg_interrupt_sent` 有效，以指示已接受中断。如果 PCI 命令寄存器中的中断禁用位设为 0，那么核会发送断言中断有效报文 (Assert_INTA)。中断处理完成后，用户应用会发出 deasserts `cfg_interrupt_int` 以将其断言无效。
- 当用户应用发出 deasserts `cfg_interrupt_int` 以将其断言无效后，核会发出断言中断无效报文 (Deassert_INTA)。这通过第 2 次断言 `cfg_interrupt_sent` 有效来表示。

`cfg_interrupt_int` 必须保持断言有效，直至用户应用接收到断言中断有效报文 (Assert_INTA) 的确认（通过断言 `cfg_interrupt_sent` 有效来表示），并且中断已由根的中断服务例程 (ISR) 来完成处理/清除为止。断言 `cfg_interrupt_int` 无效会导致核发出断言中断无效报文 (Deassert_INTA)。`cfg_interrupt_pending` 必须随 `cfg_interrupt_int` 一起保持断言有效，直至中断处理完成为止，否则状态寄存器中的中断状态位将无法正确更新。当首次断言 `cfg_interrupt_sent` 有效后，`cfg_interrupt_pending` 可随 `cfg_interrupt_int` 一起断言无效。当软件/根的 ISR 收到断言中断有效报文后，它会读取此中断状态位以判定针对此功能是否存在中断暂停。

注释：对于 PCIE4C 块，在命令寄存器中如果中断禁用位已置位（即，命令寄存器位 $10 = 1$ ），那么该核不会阻塞 INTx 中断。用户应用必须监控 `cfg_function_status` 以检查 INTx 中断处于启用还是禁用状态，仅当在命令寄存器中已启用中断时，`cfg_interrupt_int` 才能断言有效。

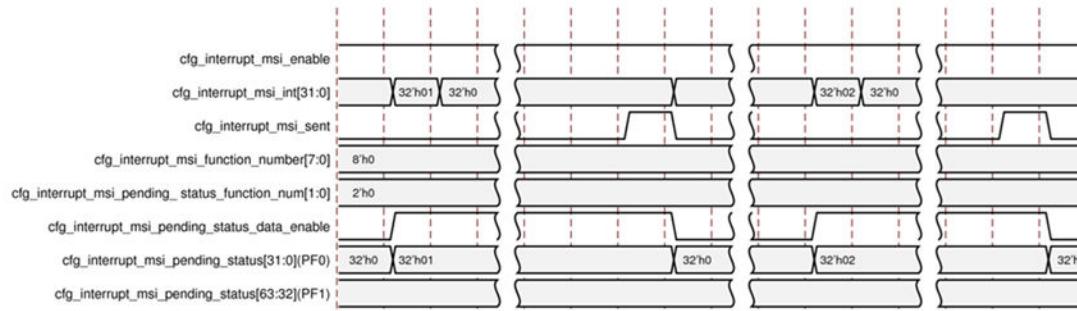
图 102：遗留中断信号



MSI 模式

用户应用首先断言 `cfg_interrupt_msi_int` 上的值有效，如前图所示。核会断言 `cfg_interrupt_msi_sent` 有效，以指示已接受中断，并且核会发送 MSI 存储器写入 TLP。

图 103：MSI 模式



MSI 请求是 32 位可寻址存储器写入 TLP 或 64 位可寻址存储器写入 TLP。此地址取自“MSI Capability Structure”（MSI 功能结构）的“Message Address”（报文地址）字段和“Message Upper Address”（报文上位地址）字段，而有效载荷则取自“Message Data”（报文数据）字段。这些值由系统软件通过将配置写入 MSI 功能结构来进行编程。当核配置为多矢量 MSI 时，系统软件可通过在“Multiple Message Enable”（多报文使能）字段中编程非 0 值来允许多矢量 MSI 报文。

发送的 MSI TLP 类型（32 位可寻址或 64 位可寻址）取决于 MSI 功能结构中的“Upper Address”（上位地址）字段的值。默认情况下，MSI 报文作为 32 位可寻址存储器写入 TLP 来发送。仅当系统软件在上位地址寄存器中编程非 0 值时，MSI 报文才会使用 64 位可寻址存储器写入 TLP。

当启用多矢量 MSI 报文时，用户应用可以覆盖每个已发射的 MSI TLP 的“报文数据”字段中的 1 个或多个低阶位以便区分上游发送的多条 MSI 报文。“Message Data”（报文数据）字段中可供用户应用使用的低阶位数量判定方式为：取 IP 目录中所设置的“Multiple Message Capable”（多报文支持）字段与系统软件所设置的“Multiple Message Enable”（多报文使能）字段之间较小的值，并可作为 `cfg_interrupt_mmenable[2:0]` 核输出以供使用。该核会屏蔽 `cfg_interrupt_msi_select` 中系统软件未通过“Multiple Message Enable”配置的所有位。

以下伪代码可显示所需的处理：

```
// Value MSI_Vector_Num must be in range: 0 ≤ MSI_Vector_Num ≤
(2^cfg_interrupt_mmenable)-1

if (cfg_interrupt_msienable) {           // MSI Enabled
    if (cfg_interrupt_mmenable > 0) {    // Multi-Vector MSI Enabled
        cfg_interrupt_msi_int[MSI_Vector_Num] = 1;
    } else {                           // Single-Vector MSI Enabled
        cfg_interrupt_msi_int[MSI_Vector_Num] = 0;
    }
} else {                                // Legacy Interrupts Enabled
}
```

例如：

- 如果 `cfg_interrupt_mmenable[2:0] == 000b`，即启用 1 个 MSI 矢量，那么 `cfg_interrupt_msi_int = 01h;`
- 如果 `cfg_interrupt_mmenable[2:0] == 101b`，即启用 32 个 MSI 矢量，那么 `cfg_interrupt_msi_int = {32'b1 << [MSI_Vector#]};`

其中，`MSI_Vector#` 为 5 位值，并且允许的范围为 `00000b ≤ MSI_Vector# ≤ 11111b`。

如果启用“Per-Vector Masking”（按矢量屏蔽），那么请首先验证“Mask register”（掩码寄存器）中未屏蔽所示矢量。具体操作方法是在配置接口上读取该寄存器（核不查看掩码寄存器）。

MSI-X 模式

核支持 MSI-X 中断及其信号，如下图所示。MSI-X 矢量表和 MSI-X 暂挂位阵列需在用户逻辑中实现，方法是在不使用内置 MSI-X 矢量表的情况下声明 BAR 间隙。

图 104：MSI-X 模式

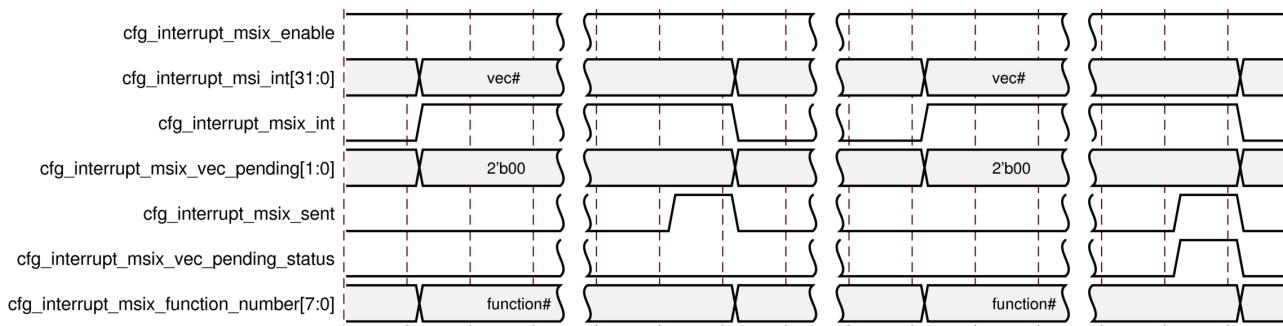


含内置 MSI-X 矢量表的 MSI-X 模式

核可选择支持内置 MSI-X 矢量表，包括暂挂位阵列。

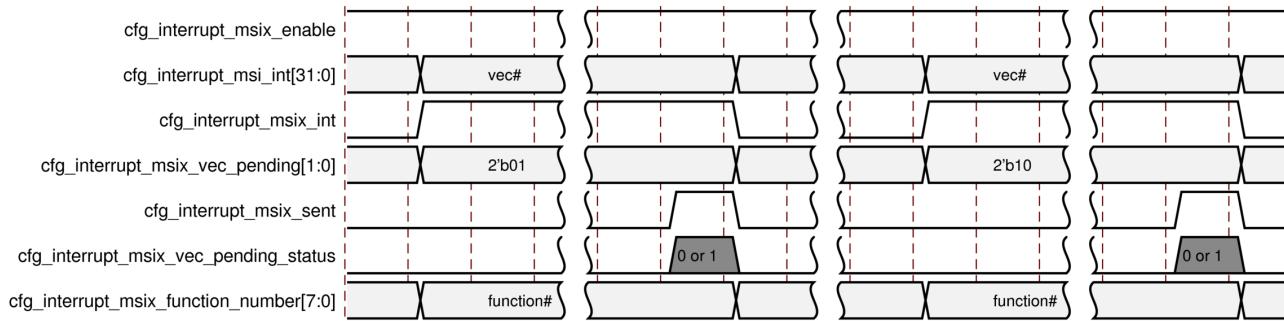
- 如下图所示，用户应用首先以 `cfg_interrupt_msi_int` 中所设矢量数来断言 `cfg_interrupt_msix_int` 有效。
- 核通过断言 `cfg_interrupt_msi_sent` 有效来发出信号，表明已接受中断。如果 `cfg_interrupt_msix_vec_pending_status` 已清除，则核会发送 MSI-X 存储器写入 TLP。否则，核会等待至清除功能掩码后再发送 MSI-X 存储器写入 TLP。

图 105：含内置 MSI-X 矢量表的 MSI-X 信号



- 除了生成中断外，用户应用也可以改为通过额外将 `cfg_interrupt_msix_vec_pending` 分别设置为 `2'b01` 或 `2'b10` 来查询或清除暂挂位阵列，如下图所示。
- 查询和清除时，`cfg_interrupt_msix_vec_pending_status` 会在查询或清除前反映暂挂状态。
- `cfg_interrupt_msi_int[31:0]` 是 MSI [31:0] 与 MSI-X [7:0] 之间的共享信号。

图 106：MSI-X 暂挂位阵列查询和清除



注释：如果应用需生成 MSI/MSI-X 中断，且流量类位不等于 0 或者地址转换位不等于 0，则此类应用必须使用 RQ 接口来生成中断（存储器写入描述符）。

接收报文接口

核可提供独立接收报文接口，以供用户应用用于接收表示已接收到来自链路的报文的指示信息。当启用接收报文接口时，集成块会发出表示已收到来自链路的报文的信号，包括设置 `cfg_msg_received_type[4:0]` 输出以指示报文类型（请参阅下表）以及脉冲 `cfg_msg_received` 信号并保持 1 个或多个周期。`cfg_msg_received` 的断言有效持续时间是由接收到的报文类型来确定的（请参阅 [表 65：接收报文接口上的报文类型编码](#)）。当 `cfg_msg_received` 处于高电平有效状态时，集成块会在 `cfg_msg_received_data` 总线上传输与总线上的报文关联的任意参数（每次 8 位）。在 [表 66：接收报文接口上的报文参数](#) 表中列出了在 `cfg_msg_received` 断言有效的每个周期内，针对各报文类型在此总线上传输的参数。对于供应商定义的报文，集成块仅传输通过此接口的任意关联有效载荷的第一个 Dword。当使用的有效载荷较大时，应使用完成器请求接口来交付报文。

表 65：接收报文接口上的报文类型编码

<code>cfg_msg_received_type[4:0]</code>	报文类型
0	ERR_COR
1	ERR_NONFATAL
2	ERR_FATAL
3	Assert_INTA
4	Deassert_INTA
5	Assert_INTB
6	Deassert_INTB
7	Assert_INTC
8	Deassert_INTC
9	Assert_INTD
10	Deassert_INTD
11	PM_PME
12	PME_TO_Ack
13	PME_Turn_Off
14	PM_Active_State_Nak

表 65：接收报文接口上的报文类型编码 (续)

cfg_msg_received_type[4:0]	报文类型
15	Set_Slot_Power_Limit
16	时延容限报告 (LTR)
17	最优化闪存刷新/填充 (OBFF)
18	解锁
19	Vendor_Defined 类型 0
20	Vendor_Defined 类型 1
21	ATS 无效请求
22	ATS 无效完成
23	ATS 页面请求
24	ATS PRG 响应
25 - 31	保留

表 66：接收报文接口上的报文参数

报文类型	cfg_msg_received 断言有效的周期数	cfg_msg_received_data[7:0] 上传输的参数
ERR_COR、ERR_NONFATAL、ERR_FATAL	2	周期 1：请求器 ID、总线编号 周期 2：请求器 ID、器件/功能编号
Assert_INTx、Deassert_INTx	2	周期 1：请求器 ID、总线编号 周期 2：请求器 ID、器件/功能编号
PM_PME、PME_TO_Ack、PME_Turn_off、PM_Active_State_Nak	2	周期 1：请求器 ID、总线编号 周期 2：请求器 ID、器件/功能编号
Set_Slot_Power_Limit	6	周期 1：请求器 ID、总线编号 周期 2：请求器 ID、器件/功能编号 周期 3：有效载荷的位 [7:0] 周期 4：有效载荷的位 [15:8] 周期 5：有效载荷的位 [23:16] 周期 6：有效载荷的位 [31:24]
时延容限报告 (LTR)	6	周期 1：请求器 ID、总线编号 周期 2：请求器 ID、器件/功能编号 周期 3：嗅探时延的位 [7:0] 周期 4：嗅探时延的位 [15:8] 周期 5：无嗅探时延的位 [7:0] 周期 6：无嗅探时延的位 [15:8]
最优化闪存刷新/填充 (OBFF)	3	周期 1：请求器 ID、总线编号 周期 2：请求器 ID、器件/功能编号 周期 3：OBFF 代码
解锁	2	周期 1：请求器 ID、总线编号 周期 2：请求器 ID、器件/功能编号

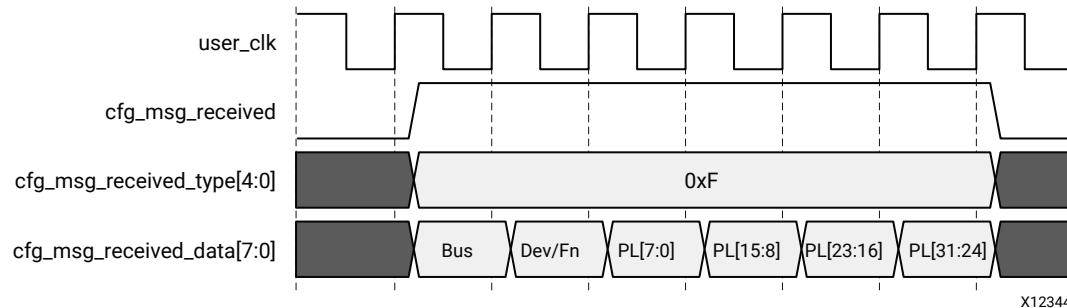
表 66：接收报文接口上的报文参数（续）

报文类型	cfg_msg_received 断言有效的周期数	cfg_msg_received_data[7:0] 上传输的参数
Vendor_Defined 类型 0	不存在数据时为 4 个周期，存在数据时则为 8 个周期。	周期 1：请求器 ID、总线编号 周期 2：请求器 ID、器件/功能编号 周期 3：供应商 ID[7:0] 周期 4：供应商 ID[15:8] 周期 5：有效载荷的位 [7:0] 周期 6：有效载荷的位 [15:8] 周期 7：有效载荷的位 [23:16] 周期 8：有效载荷的位 [31:24]
Vendor_Defined 类型 1	不存在数据时为 4 个周期，存在数据时则为 8 个周期。	周期 1：请求器 ID、总线编号 周期 2：请求器 ID、器件/功能编号 周期 3：供应商 ID[7:0] 周期 4：供应商 ID[15:8] 周期 5：有效载荷的位 [7:0] 周期 6：有效载荷的位 [15:8] 周期 7：有效载荷的位 [23:16] 周期 8：有效载荷的位 [31:24]
ATS 无效请求	2	周期 1：请求器 ID、总线编号 周期 2：请求器 ID、器件/功能编号
ATS 无效完成	2	周期 1：请求器 ID、总线编号 周期 2：请求器 ID、器件/功能编号
ATS 页面请求	2	周期 1：请求器 ID、总线编号 周期 2：请求器 ID、器件/功能编号
ATS PRG 响应	2	周期 1：请求器 ID、总线编号 周期 2：请求器 ID、器件/功能编号

以下时序图显示了接收报文接口上的 Set_Slot_Power_Limit 报文的示例。此报文含有关联的有效载荷（含 1 个 Dword）。对于此报文，参数通过 6 个连续周期进行传输。在每个周期内，cfg_msg_received_data 总线上都会显示以下信息：

- 周期 1：请求器 ID 的总线编号
- 周期 2：请求器 ID 的器件/功能编号
- 周期 3：有效载荷 Dword 的位 [7:0]
- 周期 4：有效载荷 Dword 的位 [15:8]
- 周期 5：有效载荷 Dword 的位 [23:16]
- 周期 6：有效载荷 Dword 的位 [31:24]

图 107：接收报文接口



集成块会在 `cfg_msg_received` 输出上的连续脉冲之间插入 1 个间隙，其长度至少为 1 个时钟周期。不存在对通过接收报文接口交付的报文指示信息应用反压的机制。使用此接口时，用户逻辑必须始终随时准备好接收报文指示信息。

配置管理接口

如需了解有关配置管理接口所使用的端口的信息，请参阅 [配置管理接口](#)。根端口必须使用配置管理接口来设置配置空间。端点还可以使用配置管理接口来读取和写入，但必须谨慎操作以避免对系统产生负面影响。

用户应用必须以 Dword 地址而不是字节地址的形式来提供地址。



提示：要计算寄存器的 Dword 地址，请将字节地址除以 4。

例如：

对于 PCI 配置空间报头中的命令/状态寄存器：

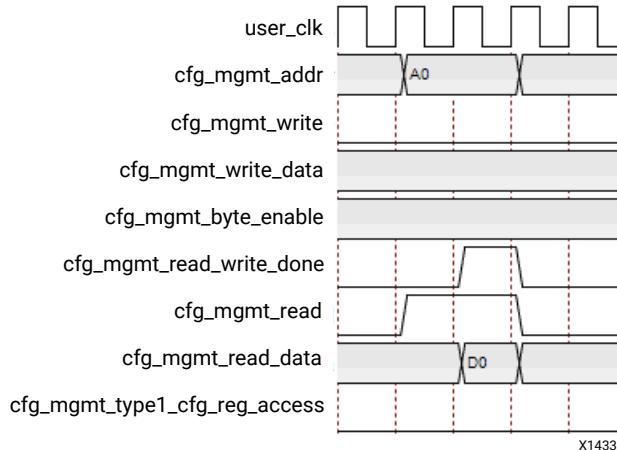
- Dword 地址为 01h。
注释：字节地址为 04h。

对于 BAR0：

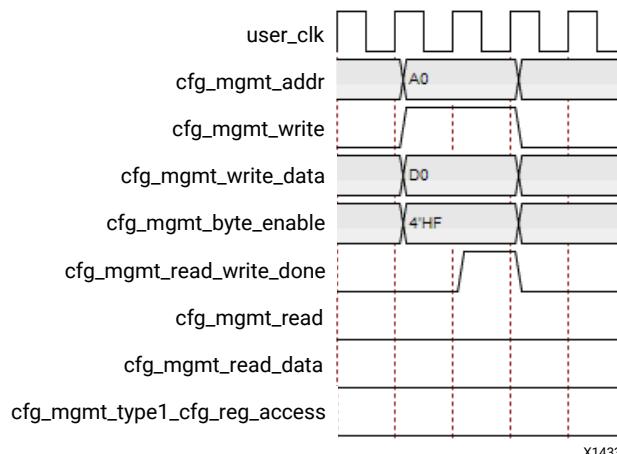
- Dword 地址为 04h。
注释：字节地址为 10h。

要读取配置空间内的任意寄存器，用户应用会将寄存器 Dword 地址驱动到 `cfg_mgmt_addr[9:0]`。

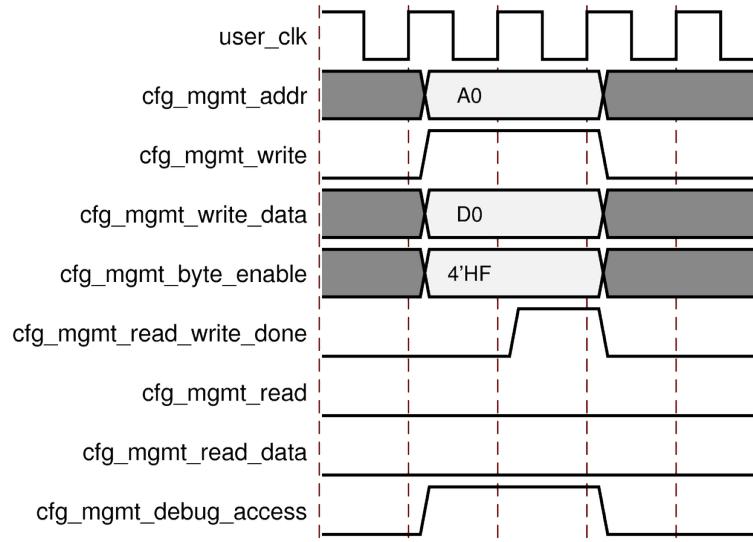
`cfg_mgmt_function_number[7:0]` 用于选择与配置寄存器关联的 PCI 功能。核会将已寻址的寄存器的内容驱动到 `cfg_mgmt_read_data[31:0]` 上。`cfg_mgmt_read_data[31:0]` 上的值由 `cfg_mgmt_read_write_done` 上的信号断言有效来限定。下图显示了从配置空间读取的示例。

图 108: **cfg_mgmt_read_type0_type1**

要在配置空间内写入任何寄存器，用户逻辑需将地址置于 `cfg_mgmt_addr[9:0]` 上、将功能编号置于 `cfg_mgmt_function_number[7:0]` 上、在 `cfg_mgmt_write_data` 上写入数据、使 `cfg_mgmt_byte_enable [3:0]` 上的字节有效，并断言 `cfg_mgmt_write` 信号有效。在响应中，当写入完成时（可能需耗时多个周期），核会断言 `cfg_mgmt_read_write_done` 信号有效。用户逻辑必须使 `cfg_mgmt_addr`、`cfg_mgmt_function_number`、`cfg_mgmt_write_data`、`cfg_mgmt_byte_enable` 和 `cfg_mgmt_write` 保持稳定，直至断言 `cfg_mgmt_read_write_done` 有效。用户逻辑还必须在核断言 `cfg_mgmt_read_write_done` 有效之后的下一个周期内断言 `cfg_mgmt_write` 无效。

图 109: **cfg_mgmt_write_type0**

当核配置为采用“Root Port”（根端口）模式时，写入类型 1 PCIe® 配置寄存器期间如果断言 `cfg_mgmt_debug_access` 有效，则会强制写入寄存器的某些只读字段。

图 110: **cfg_mgmt_debug_access**

在根端口上启用环回主控制器

“Loopback Master”（环回主控制器）功能是在 VSEC 空间内实现的（请参阅 [PCI Express 扩展配置空间](#)）。根端口的环回主控制器功能从字节地址 'h330 开始。要实践 “Loopback Start”（环回启动），请使用 `cfg_mgmt` 接口将 `1'b1` 写入字节地址 'h338（环回控制）中的位 0。

由于 `cfg_mgmt` 使用 DW 地址，请使用 `cfg_mgmt_write_data = 32'h1` 来执行如下寻址操作：

- `cfg_mgmt_address = 10'hCE`
- `cfg_mgmt_write = 1'b1`
- `cfg_mgmt_byte_enable = 4'b1`

完成环回启动后，请通过写入链路控制寄存器来对链路进行重新训练（字节地址：'h80、DW 地址：'h20）。使用 `cfg_mgmt_write_data = 'h20` 来执行如下寻址操作：

- `cfg_mgmt_address = 10'h20`
- `cfg_mgmt_write = 1'b1`
- `cfg_mgmt_byte_enable = 4'b1`

链路训练：2 通道、4 通道、8 通道和 16 通道组件

2 通道、4 通道和 8 通道核可以小于《PCI Express® 基本规范》所要求的最大通道宽度的通道宽度来运行。存在 2 种情况会导致核以小于其指定的最大通道宽度的通道宽度来运行，如下列小节中所述。

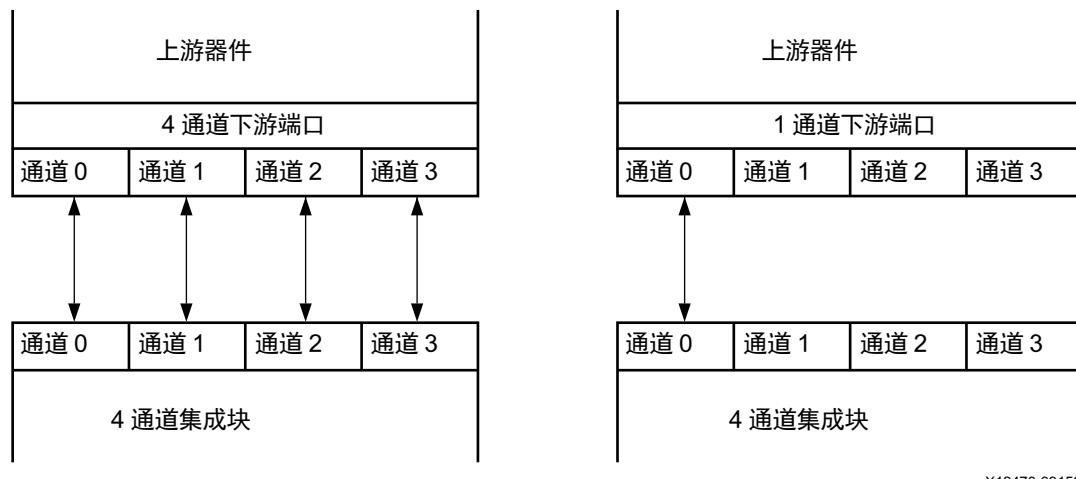
链路伙伴支持的通道数量减少

当 2 通道核连接到的器件仅实现 1 通道时，2 通道核会使用通道 0 作为 1 通道器件来训练并运行。

当 4 通道核连接到的器件实现 1 通道时，4 通道核会使用通道 0 作为 1 通道器件来训练并运行，如下图所示。同样，如果 4 通道核连接到 2 通道器件，那么该核会使用通道 0 和 1 作为 2 通道器件来训练并运行。

当 8 通道核连接到的器件仅实现 4 通道时，它会使用通道 0-3 作为 4 通道器件来训练并运行。此外，如果所连接的器件仅实现 1 或 2 条通道，那么 8 通道核会分别作为 1 通道器件和 2 通道器件来训练并运行。

图 111：将 4 通道端点块从 4 通道缩小到 1 通道运行



通道故障

如果链路经训练达到核与链路伙伴器件所支持的最大通道宽度之后发生故障，核会尝试恢复并训练至较低的通道宽度（如可用）。如果通道 0 发生故障，则链路会发生不可恢复性中断。如果通道 1-7 中的任意或全部通道发生故障，链路会进入恢复状态，并尝试对仍正常运行的任何通道恢复可行的最大链路。

例如，使用 8 通道核时，通道 1 中断会导致恢复为在通道 0 上执行单通道操作，而通道 6 中断则会导致恢复为在通道 0-3 上执行 4 通道操作。发生恢复后，如果故障通道恢复正常，则核不会尝试恢复至更大的链路宽度。仅当链路确实中断并尝试从头开始重新训练时，才可能增大链路宽度。

`user_clk` 时钟输出采用固定频率，在 IP 目录中配置。`user_clk` 在发生链路恢复或训练中断时，不会变换频率。

通道翻转

该集成块支持有限通道翻转功能，因此可提升链路伙伴开发板设计过程中的灵活性。链路伙伴可选择按翻转的通道编号设计开发板布局，集成块仍可成功完成链路训练，并正常运行。提供通道翻转支持的配置包括 16x、x8 和 x4（不包括下移模式）。下移表示链路伙伴播发的通道宽度功能不同时，所发生的链路宽度协商过程。通道宽度协商的结果是，链路伙伴将通过协商采用较小的播发通道宽度。下表描述了包括下移模式以及通道翻转支持可用性在内的几种可行的组合。

表 67：通道翻转支持

集成块颁发通道宽度	协商的通道宽度	通道编号映射（端点链路伙伴）		通道翻转支持
		端点	链路伙伴	
x16	x16	通道 0 至通道 15	通道 15 至通道 0	支持
x16	x8	通道 0 至通道 7	通道 7 至通道 0	不支持
x16	x4	通道 0 至通道 3	通道 3 至通道 0	不支持
x16	x2	通道 0 至通道 1	通道 1 至通道 0	不支持
x8	x8	通道 0 至通道 7	通道 7 至通道 0	支持
x8	x4	通道 0 至通道 3	通道 7 至通道 4	不支持 ¹
x8	x2	通道 0 至通道 3	通道 7 至通道 6	不支持 ¹
x4	x4	通道 0 至通道 3	通道 3 至通道 0	支持
x4	x2	通道 0 至通道 1	通道 3 至通道 2	不支持 ¹
x2	x2	通道 0 至通道 1	通道 1 至通道 0	支持
x2	x1	通道 0 至通道 1	通道 1	不支持 ¹

注释：

1. 当在开发板布局中采用翻转通道并且在端点与链路伙伴之间插入下移适配卡时，链路伙伴的通道 0 将保持处于未连接状态（如表中的通道映射所示），因此不执行链路训练。



重要提示！ 默认情况下，端点配置通过 DISABLE_LANE_REVERSAL 属性禁用通道翻转支持。如果链路伙伴具有通道翻转功能，则不得启用端点配置下的通道翻转。

设计流程步骤

本章节描述了核的自定义和生成方式、核的约束方式以及此 IP 核的仿真、综合与实现的具体步骤。如需获取有关标准 Vivado® 设计流程以及有关 IP integrator 的详细信息，请参阅以下 Vivado Design Suite 用户指南：

- 《Vivado Design Suite 用户指南：采用 IP integrator 设计 IP 子系统》([UG994](#))
- 《Vivado Design Suite 用户指南：采用 IP 进行设计》([UG896](#))
- 《Vivado Design Suite 用户指南：入门指南》([UG910](#))
- 《Vivado Design Suite 用户指南：逻辑仿真》([UG900](#))

自定义和生成核

本节包含有关如何使用赛灵思工具在 Vivado® Design Suite 中自定义和生成核的信息。

如果要在 Vivado IP integrator 中自定义和生成核，请参阅《Vivado Design Suite 用户指南：采用 IP integrator 设计 IP 子系统》([UG994](#)) 了解详情。确认或生成设计时，IP integrator 可能会自动计算某些配置值。要查看配置值是否会更改，请参阅本章中的参数说明。要查看参数值，请在 Tcl 控制台中运行 `validate_bd_design` 命令。

您可以遵循以下步骤通过指定与 IP 核关联的各种参数值来自定义设计中使用的 IP：

1. 从 IP 目录选择 IP。
2. 双击所选 IP，或者从工具栏或右键单击菜单中选择“Customize IP”（自定义 IP）命令。

欲知详情，请参阅《Vivado Design Suite 用户指南：采用 IP 进行设计》([UG896](#)) 和《Vivado Design Suite 用户指南：入门指南》([UG910](#))。

本章中的附图是 Vivado IDE 的插图。此处展示的布局可能与当前版本中的布局有所不同。

UltraScale+ 器件 Integrated Block for PCI Express 核的“Customize IP”（自定义 IP）对话框包含 2 种模式：[基本模式参数](#) 和 [高级模式参数](#)。要选择模式，请使用“Customize IP”对话框的第一页上的“Mode”（模式）下拉列表。以下章节解释了每个模式中可用的参数。

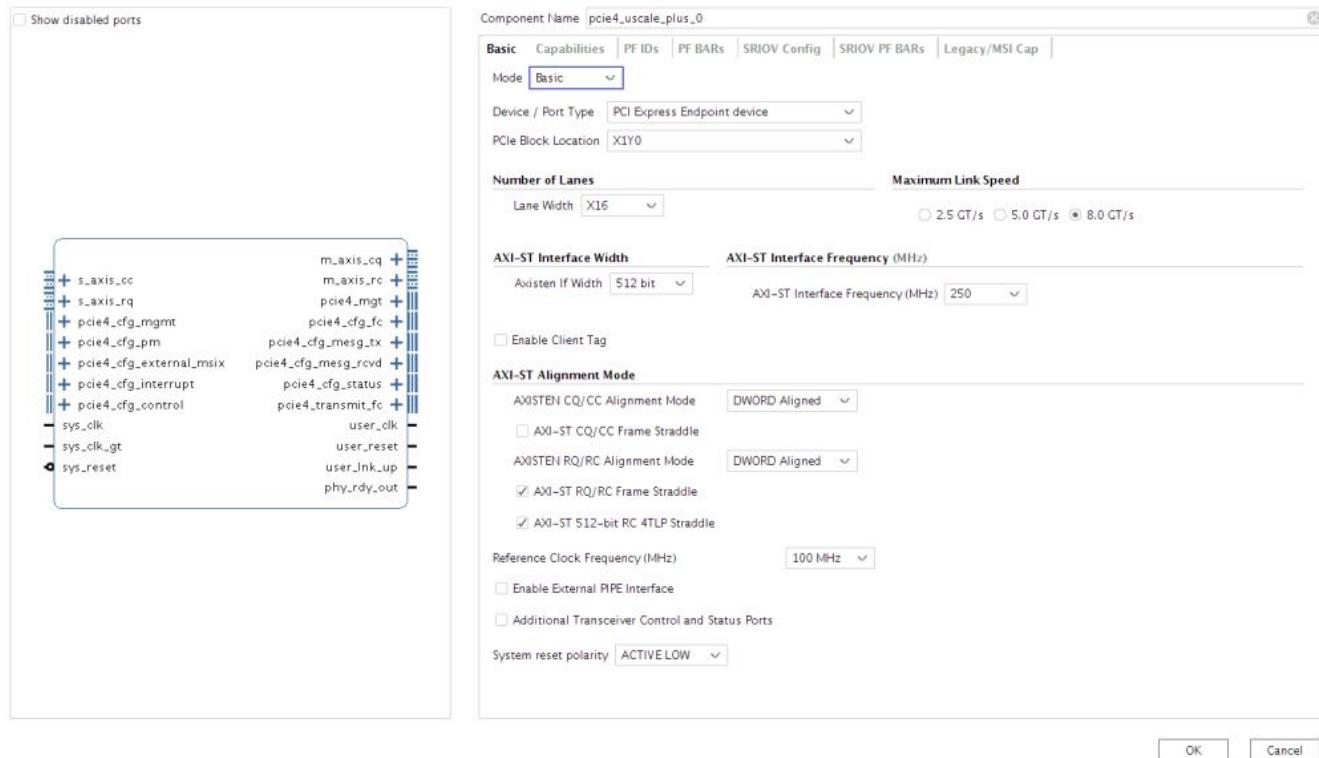
基本模式参数

本章对基本模式参数进行了解释。

“Basic” 选项卡

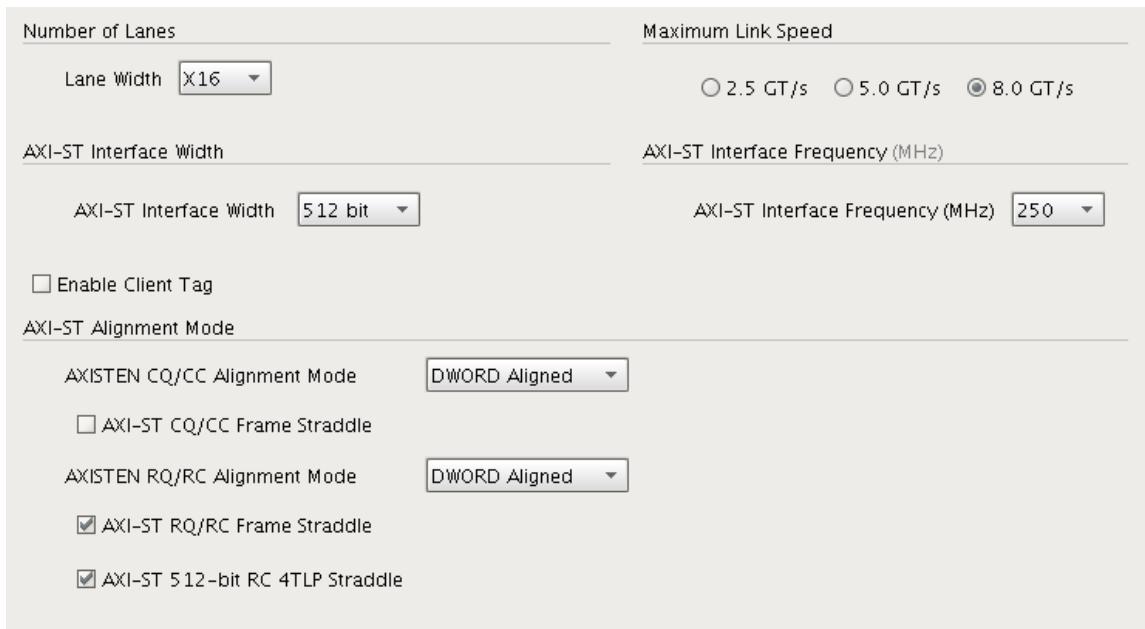
下图显示了初始自定义页面，此页面用于设置“Basic”（基本）模式参数。

图 112：“Basic” 选项卡



- “Component Name”（组件名称）：针对核生成的输出文件的基本名称。此名称必须以字母开头，可由下列字符组成：a 到 z、0 到 9 以及 “_”。
- “Mode”（模式）：允许您为核的配置选择“Basic”（基本）模式或“Advanced”（高级）模式。
- “Device/Port Type”（器件/端口类型）：用于指示 PCI Express 逻辑器件类型。
- “PCIe Block Location”（PCIe 块位置）：用于从可用集成块中进行选择，以支持生成特定位置的约束文件和管脚分配。该选项可在默认设计示例脚本中使用。
- “Number of Lanes”（通道数）：核需要选择初始通道宽度。核设置的通道宽度越宽，则连接到通道宽度更小的器件时，就可以向下训练至更小的通道宽度。
- “Maximum Link Speed”（最大链路速度）：核允许您选择器件支持的最大链路速度。核设置的链路速度越快，那么连接到支持更低链路速度的器件时，就可以训练至更低的链路速度。
- “AXI-ST Interface Frequency”（AXI-ST 接口频率）：允许您指定 AXI-ST 接口频率。
- “AXI-ST Interface Width”（AXI-ST 接口宽度）：核允许您选择接口宽度。“Customize IP”（自定义 IP）对话框中设置的默认接口宽度是支持的最低接口宽度。
- “AXI-ST Alignment Mode”（AXI-ST 对齐模式）：当存在有效载荷时，有 2 个选项可用于对齐与数据路径有关的有效载荷的第 1 个字节。其中提供了选择 CQ/CC 接口和 RQ/RC 接口的选项。
- “Enable AXI-ST Frame Straddle”（启用 AXI-ST 跨帧）：当接口宽度为 256 位时，核提供了在请求器完成接口上跨接数据包的选项。

图 113：“Basic”选项卡：“Straddle”选项



- “AXI-ST CQ/CC Frame Straddle”（AXI-ST CQ/CC 跨帧）和“AXI-ST RQ/RC Frame Straddle”（AXI-ST RQ/RC 跨帧）：选择 512 位 AXI-ST 接口宽度时，针对 CQ、CC、RQ 和 RC AXI-ST 接口支持 AXI-ST 跨帧。其中提供了同时选中 CQ 和 CC AXI-ST 跨帧的选项，以及选择 RQ 和 RC 接口跨帧的选项。
- “AXI-ST 512-bit RC 4TLP Straddle”（AXI-ST 512 位 RC 4TLP 跨接）：当 AXI-ST 512 位 RC 4TLP 跨接设置为 TRUE 时，即表示它采用 4 TLP 跨接模式，否则采用 2 TLP 跨接模式，在后一种情况下，当前一个 TLP 终止于字节通道 31 或者在此通道前终止时，可在字节通道 32 上启动新的 TLP。
- “Enable Client Tag”（启用客户标签）：该选项支持您使用客户标签。
- “Reference Clock Frequency”（参考时钟频率）：选择 sys_clk。
- “Enable External PIPE Interface”（启用外部 PIPE 接口）：选中该选项时，可启用外部第三方总线功能模型（BFM）以连接至 Integrated Block for PCIe 的 PIPE 接口。欲知详情，请参阅《使用集成端点 PCIe Express 块采用 Gen3 x8 和 Gen2 x8 配置进行 PIPE 模式仿真》（XAPP1184）。请参阅以下设计，以了解如何将 UltraScale™ 器件核的外部 PIPE 接口端口连接至第三方 BFM。
- “Additional Transceiver Control and Status Ports”（其它收发器控制和状态端口）：这些端口可用于调试收发器相关问题和 PCIe 相关信号。您必须根据相应的 GT 用户指南进行驱动。



重要提示！ “Enable In System IBERT”（启用 In-System IBERT）选项和“Enable JTAG Debugger”（启用 JTAG Debugger）选项应仅用于硬件调试目的。针对使用这些选项生成的核，不支持执行仿真。

- “System reset polarity”（系统复位极性）：此参数用于将 sys_rst 的极性设置为 ACTIVE_HIGH 或 ACTIVE_LOW。

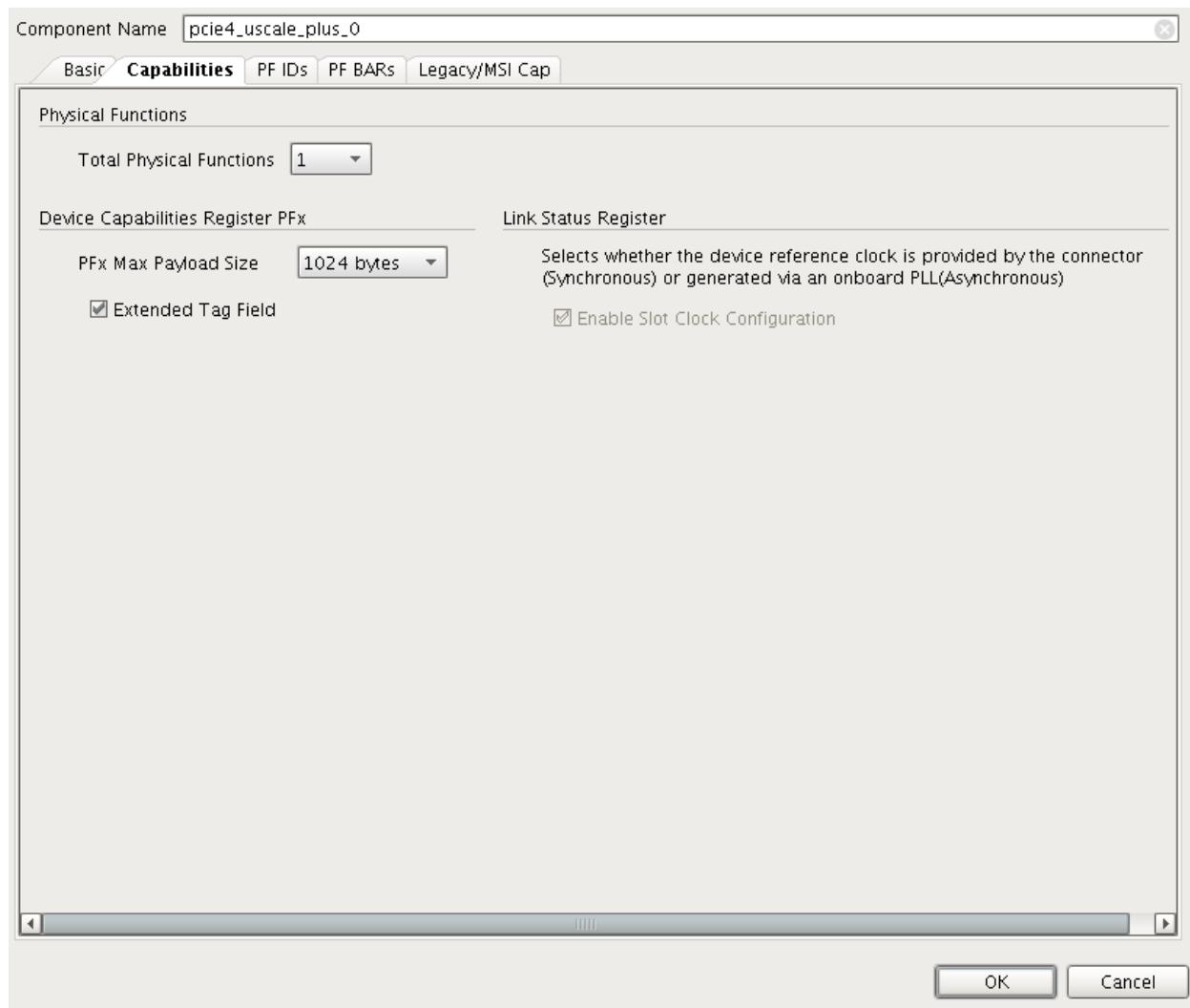
相关信息

收发器控制端口和状态端口
时钟
受支持的时钟频率和接口宽度
链路训练：2 通道、4 通道、8 通道和 16 通道组件
数据对齐选项
256 位接口的跨接选项
512 位完成器接口

“Capabilities” 选项卡

本章节中描述了“Capabilities”（功能）设置，如下图所示。

图 114：“Capabilities” 选项卡



- “Total Physical Functions”（物理功能总数）：支持您选择物理功能的数量。受支持的物理功能数量为 4。

- “PFx Max Payload Size”（PFx 最大有效载荷大小）：该字段表示器件或功能针对 TLP 所支持的最大有效载荷大小。该值将通过“Device Capabilities Register”（器件功能寄存器）广播到系统。
- “Extended Tag Field”（扩展标签字段）：该字段表示针对“Tag”（标签）字段受支持的最大大小。选项为：
 - 选中时，支持 8 位标签字段（256 个标签）
 - 不选中时，支持 5 位标签字段（32 个标签）
- “Enable Slot Clock Configuration”（启用时隙时钟配置）：在链路状态寄存器中启用时隙时钟配置位。
 - 选中此选项时，将同步执行链路时钟设置。
 - 不选中此选项时，支持 SRNS 模式下的异步时钟。SRNS 表示不含 SSC 的独立参考时钟（不含 SRIS 支持的异步时钟）。

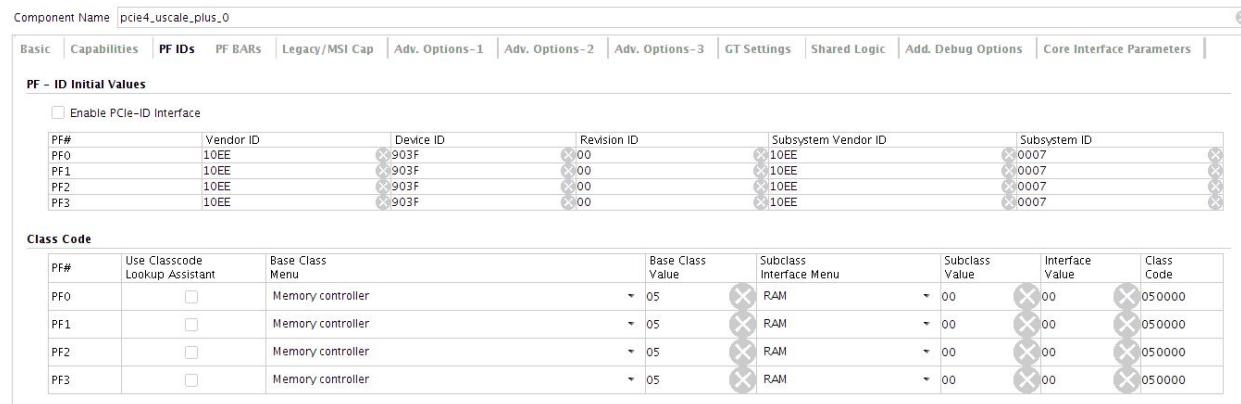
相关信息

时钟

“PF IDs” 选项卡

下图显示了“Identity Settings”（身份设置）参数。

图 115：“PF IDs” 选项卡



- “Enable PCIe-ID Interface”（启用 PCIe-ID 接口）：如果选中该参数，那么根据选中的 PFx 数量，在核顶层边界处会显示 PCIe ID 端口 `cfg_vend_id`、`cfg_subsys_vend_id`、`cfg_dev_id_pf*`、`cfg_rev_id_pf*` 和 `cfg_subsys_id_pf*`，并可供用户逻辑驱动。如未选中该参数，则不会在顶层显示这些端口，并根据自定义时设置的值来驱动这些端口。
- “PFO ID Initial Values”（PFO ID 初始值）：
 - “Vendor ID”（供应商 ID）：用于识别器件或应用的制造商。有效标识由 PCI Special Interest Group 指定，以保证每个标识都唯一。默认值 10EEh 为赛灵思的供应商标识。请在此处输入供应商标识号。此处 FFFFh 为保留值。
 - “Device ID”（器件 ID）：表示应用的唯一标识；默认值取决于所选配置。默认值为 70<link speed><link width>h。该字段可采用任何值；请针对应用更改该值。默认器件 ID 参数取决于：
 - 器件系列：9 表示 UltraScale+，8 表示 UltraScale，7 表示 7 系列器件。
 - EP 或 RP 模式。

- 链路宽度：1 表示 x1，2 表示 x2，4 表示 x4，8 表示 x8，F 表示 x16。
- 链路速度：1 表示 Gen1，2 表示 Gen2，3 表示 Gen3，4 表示 Gen4。

如果以上任意值发生更改，则将重新计算“Device ID”值，以替换先前设置的值。



建议：始终建议首先更改链路宽度、链路速度和器件端口类型，然后再更改器件 ID 值。请确保器件 ID 值设置正确，然后再生成 IP。

- “Revision ID”（版本 ID）：表示器件或应用的版本；作为器件 ID 的扩展。默认值为 00h；请针对应用输入相应的值。
- “Subsystem Vendor ID”（子系统供应商 ID）：用于进一步限定器件或应用的制造商。请在此处输入子系统供应商 ID；默认值为 10EEh。通常，该值与供应商 ID 相同。将该值设为 0000h 可能导致合规性测试出现问题。
- “Subsystem ID”（子系统 ID）：用于进一步限定器件或应用的制造商。该值通常与器件 ID 相同；默认值取决于所选通道宽度和链路速度。将该值设为 0000h 可能导致合规性测试出现问题。
- “Class Code”（类代码）：类代码用于识别器件的常规功能，分为以下 3 个字节大小的字段：
 - “Base Class”（基本类）：用于广泛识别器件执行的功能类型。
 - “Sub-Class”（子类）：用于进一步具体识别器件功能。
 - “Interface”（接口）：用于定义特定寄存器级别编程接口（如果有），允许不从属于器件的软件与器件进行连接。

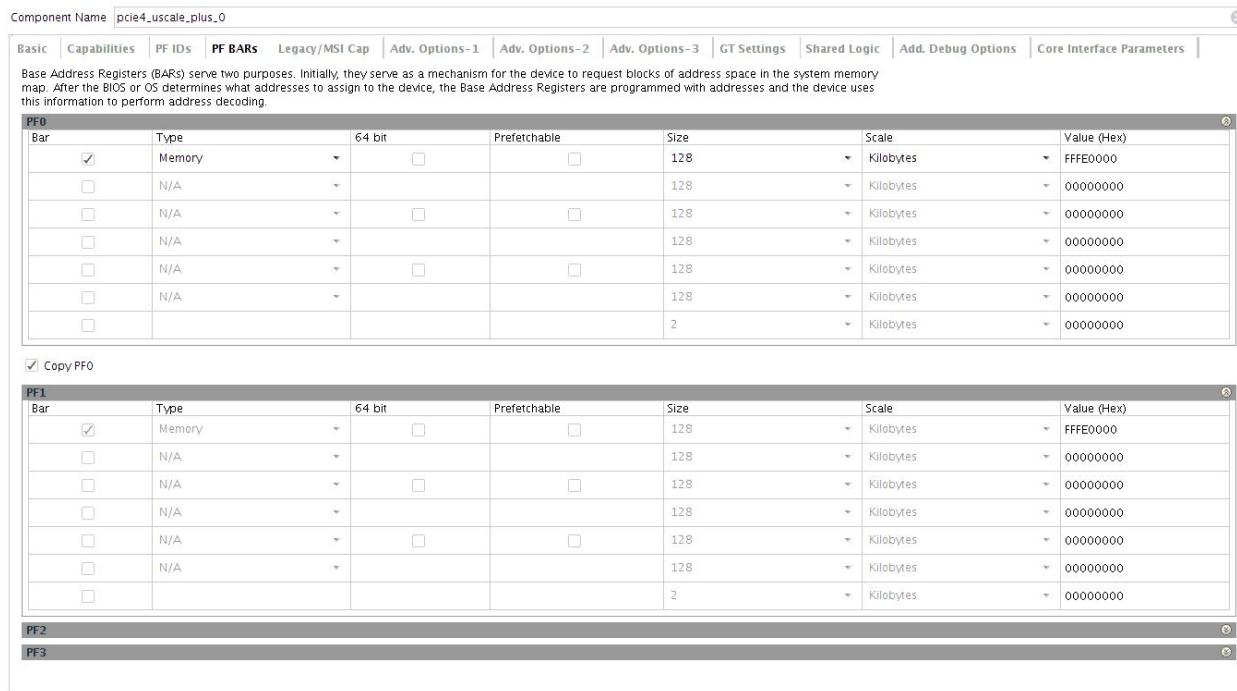
在 [PCI-SIG 网站](#) 上提供了类代码编码。

- “Class Code Look-up Assistant”（类代码查找助手）：类代码查找助手可针对选定的器件常规功能提供对应的基本类、子类和接口值。此查找助手工具仅针对选定功能显示这 3 个值。您必须在类代码中输入这 3 个值，才能将这些值转换为器件设置。

“PF BARs” 选项卡

“PF BARs” 选项卡（如下图所示）可用于为端点配置设置基址寄存器空间。每个 BAR（0 到 5）均可用于配置物理功能的“BAR Aperture Size”（BAR 间隙大小）和“Control”（控制）属性。

图 116：仅显示 PF0 和 PF1 的“PF BARs”选项卡



- “Base Address Register Overview”（基址寄存器概述）：在端点配置中，核支持最多 6 个 32 位 BAR 或 3 个 64 位 BAR 以及扩展只读存储器 (ROM) BAR。在根端口配置中，核支持最多 2 个 32 位 BAR 或 1 个 64 位 BAR 以及扩展 ROM BAR。BAR 分 2 种大小：
 - 32 位 BAR：地址空间最小可达 128 字节或者最大可达 2 千兆字节 (GB)。用于存储器或 I/O。
 - 64 位 BAR：地址空间最小可达 128 字节或者最大可达 8 艾字节 (EB)。仅用于存储器。

所有 BAR 寄存器共享下列选项：

- “Checkbox”（复选框）：单击复选框可启用 BAR；取消选择复选框可禁用 BAR。
- “Type”（类型）：BAR 可分为 I/O 和存储器。
 - I/O：I/O BAR 只能采用 32 位；“Prefetchable”（可预取）选项不适用于 I/O BAR。仅限针对传统 PCI Express 端点才能启用 I/O BAR。
 - “Memory”（存储器）：存储器 BAR 可采用 64 位或 32 位，并且可预取。当 BAR 设为 64 位时，它使用下一个 BAR 作为扩展地址空间，并使下一个 BAR 不可访问。
- “Size”（大小）：可用大小范围取决于所选 PCIe 器件/端口类型和 BAR 类型。下表列出了可用 BAR 大小范围。

表 68：器件配置的 BAR 大小范围

PCIe 器件/端口类型	BAR 类型	BAR 大小范围
PCI Express 端点	32 位存储器	128 字节 (B) - 2 GB
	64 位存储器	128 B - 8 EB

表 68：器件配置的 BAR 大小范围 (续)

PCIe 器件/端口类型	BAR 类型	BAR 大小范围
传统 PCI Express 端点	32 位存储器	128 B - 2 GB
	64 位存储器	128 B - 8 EB
	I/O	16 B - 2 GB

- “Prefetchable”（可预取）：识别存储器空间预取功能。
- “Value”（值）：表示基于当前选择分配给 BAR 的值。
- “Expansion ROM Base Address Register”（扩展 ROM 基址寄存器）：如果选中此项，则可激活扩展 ROM 并将其大小从 2 KB 调整为 4 GB。根据 [PCI-SIG 网站](#)上的《PCI 局部总线规范第 3.0 版》，扩展 ROM BAR 的最大大小应不超过 16 MB。选中大小超过 16 MB 的地址空间可能导致合规性测试出现问题。
- “Managing Base Address Register Settings”（管理基址寄存器设置）：存储器、I/O、类型和可预取设置均可通过为目标基址寄存器设置相应的设置来进行处理。

存储器或 I/O 设置用于指示地址空间定义为存储器还是 I/O。基址寄存器仅响应访问指定地址空间的命令。通常，应避免使用大小小于 4 KB 的存储器空间。允许的最小 I/O 空间为 16 字节；在所有新设计中应避免使用 I/O 空间。

可预取功能是对存储器空间进行预取的功能。只要对于读取没有副作用（即，读取数据不会像从 RAM 读取一样导致数据销毁），即表示存储器空间可预取。在适当情况下，多个字节写入操作可合并为单一双字写入操作。

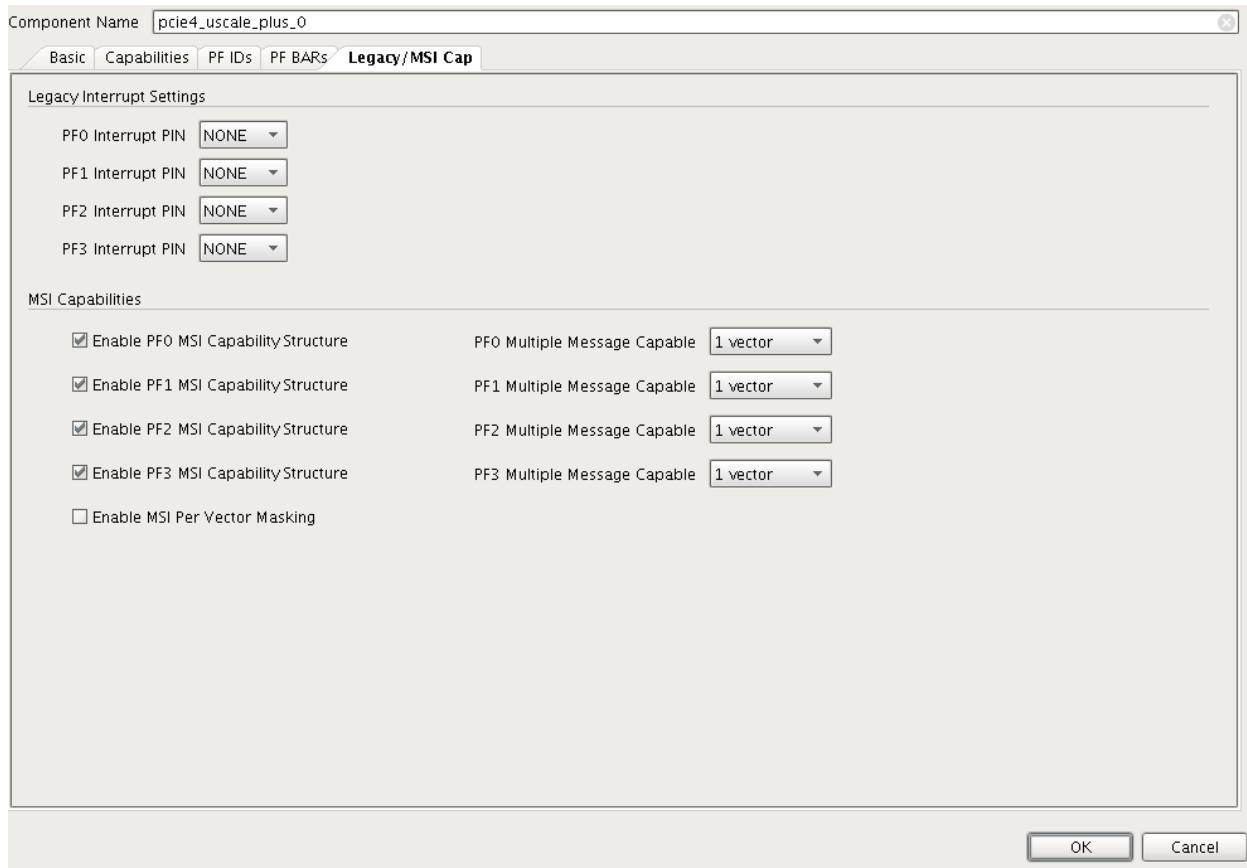
针对非传统 PCIe 将核配置为端点时，已设置可预取位的所有 BAR（BAR5 除外）都必须支持 64 位寻址。针对未设置可预取位的所有 BAR，则允许 32 位寻址。可预取位相关的要求不适用于传统端点。BAR 支持的最小存储器地址范围为 128 字节（针对 PCI Express 端点）和 16 字节（针对传统 PCI Express 端点）。

- “Disabling Unused Resources”（禁用未使用的资源）：为实现最佳结果，请禁用未使用的基址，以节省系统资源。通过在“Customize IP”（自定义 IP）对话框中取消选中未使用的 BAR 即可禁用基址寄存器。
- “Copy PFO”（复制 PFO）：设置“Copy PFO”选项后，即可允许您将剩余 PF 的所有 BAR 设置都设为与 PFO 相同的值。如有多个物理功能 (PF)，则适用该选项。

“Legacy/MSI Cap” 选项卡

在此页面上，您可为所有使用的物理功能和虚拟功能设置“Legacy Interrupt Settings”（遗留中断设置）和“MSI Capabilities”（MSI 功能）。在“Capabilities”页面上选中“SRIOV Capability”（SRIOV 功能）时，此页面不显示。选中“Advanced”（高级）模式时即可显示“Capabilities”页面。在“Basic”模式和“Advanced”模式下不选中“SRIOV Capability”参数时，均可显示此页面。

图 117：“Legacy/MSI Cap” 选项卡



- “Legacy Interrupt Settings”（遗留中断设置）：
 - PF0/PF1/PF2/PF3 中断 PIN (PF0/PF1/PF2/PF3 Interrupt PIN)：表示遗留中断报文的映射。设置为“None”（无）即表示不使用遗留中断。
注释：启用 PASID 时，无法使用（并禁用）遗留中断。
- “MSI Capabilities”（MSI 功能）：
 - “PF0/PF1/PF2/PF3 Enable MSI Capability Structure”（PF0/PF1/PF2/PF3 启用 MSI 功能结构）：表示存在 MSI 功能结构。
注释：虽然可以不启用 MSI 或 MSI-X，但这样会导致核不合规。《PCI Express 基本规范》要求启用 MSI 和/或 MSI-X。在“Advanced”模式下的“[MSI-X Capabilities](#)”选项卡中启用“MSI-X Internal”（MSI-X 内部）时，不支持任何 MSI 功能，因为 MSI-X 内部会使用部分 MSI 接口信号。
 - “PF0/PF1/PF2/PF3 Multiple Message Capable”（PF0/PF1/PF2/PF3 多报文支持）：选择要从根联合体请求的 MSI 矢量数量。
 - “Enable MSI Per Vector Masking”（按矢量屏蔽启用 MSI）：为已启用的所有物理功能启用“MSI Per Vector Masking Capability”。
注释：不支持单独为个别物理功能启用该选项。

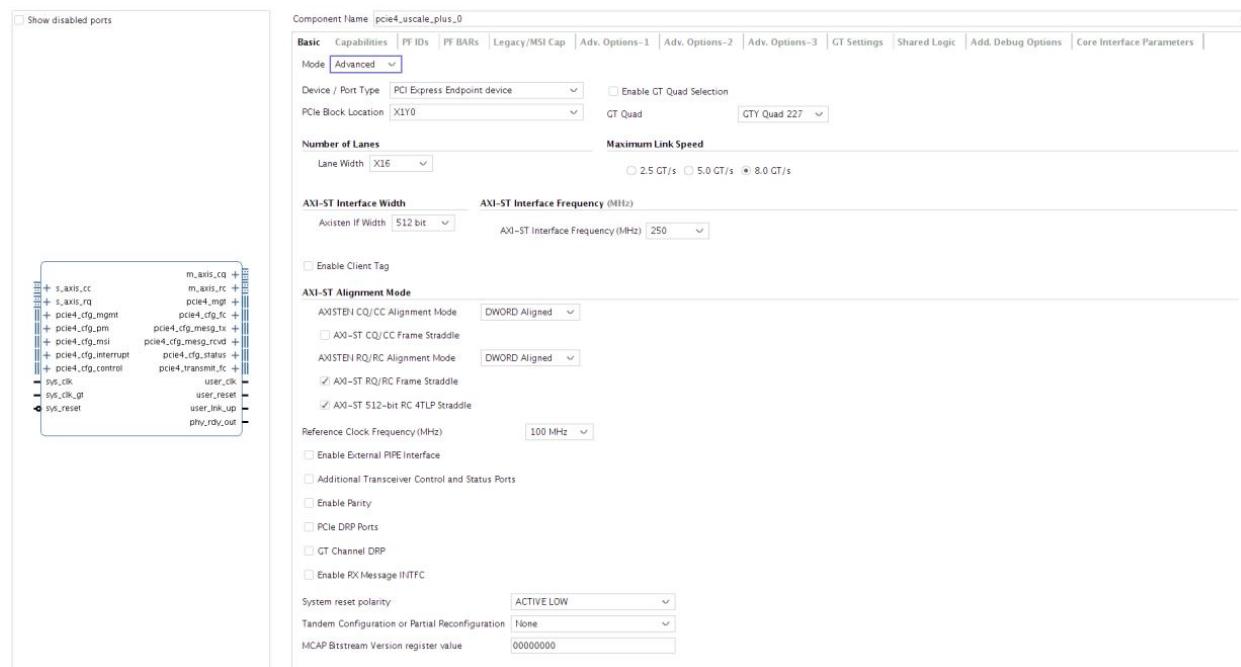
高级模式参数

在“Basic”（基本）页面上针对“Mode”（模式）选中“Advance”（高级）模式时，在IP目录的不同页面上会显示以下参数。

“Basic” 选项卡

在“Basic”（基本）页面中选中“Advanced”（高级）模式（如下图所示）后，会出现更多设置。选中“Advanced”模式时，会在“Basic”页面上显示下列参数。

图 118：“Basic” 选项卡的“Advanced” 模式



- “Enable GT Quad Selection”（启用 GT 四通道选择）：选中此参数时，即可选择 PCIe 块可用的不同 GT 四通道。
- “GT Quad”（GT 四通道）：此下拉菜单会列出对于给定 PCIe 块可用的所有 GT 四通道。请注意，根据所选 GT 四通道，针对此特定四通道可配置的最大链路宽度将发生改变。
- “Core Clock Frequency”（核时钟频率）：此参数允许您选择核时钟频率。

对于 Gen3 链路速度：

- 对于 -1、-2 和 -3 速度等级和 x8 链路宽度，值 250 MHz 和 500 MHz 可供选择
- 对于 -2L、-2LV、-1L 和 -1LV 速度等级和 x8 链路宽度，此参数默认设置为 250 MHz 且不可供选择。
- 对于 x16 链路宽度，此参数值默认设置为 500 MHz 且不可供选择。
- 对于 x1、x2 和 x4 链路宽度，此参数默认设置为 250 MHz 且不可供选择。

对于 Gen1 和 Gen2 链路速度：

- 对于除 x8 以外的链路宽度，此参数默认设置为 250 MHz 且不可供选择。

- 对于 x8 链路宽度，此参数默认设置为 500 MHz 且不可供选择。
- “Enable Parity”（启用奇偶校验）：在 TX/RX 接口上启用奇偶校验（包括 MSI-X）。
- “PCIe DRP Ports”（PCIe DRP 端口）：选中此项时，启用 PCIe DRP 接口。
- “GT Channel DRP”（GT 通道 DRP）：选中此项时，启用 GT 通道 DRP 接口。

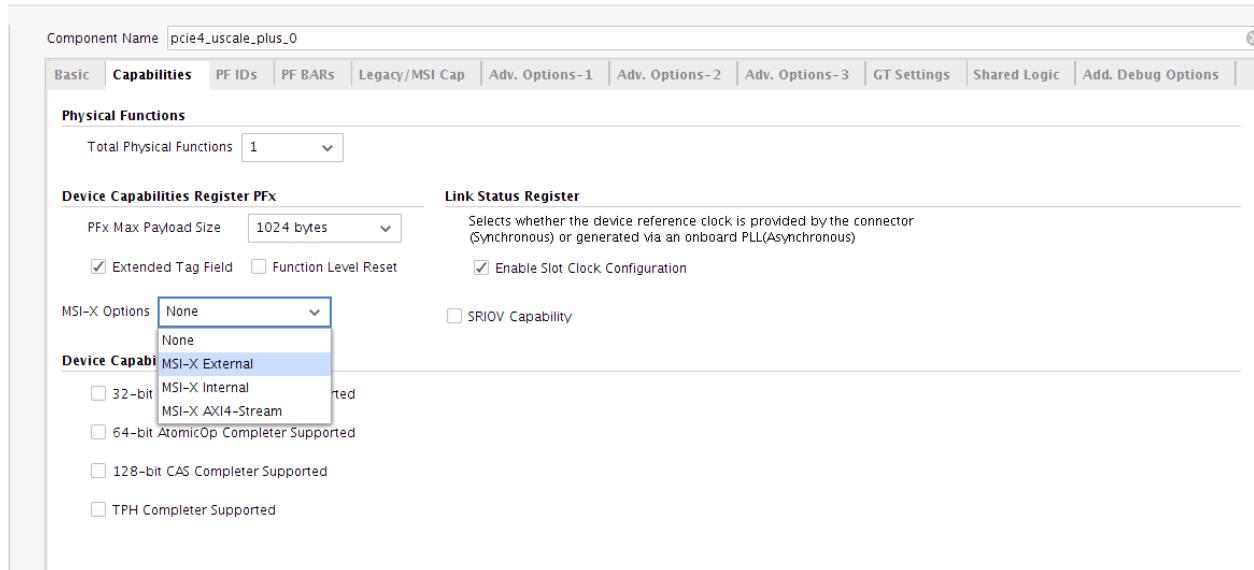
相关信息

GT 位置

“Capabilities” 选项卡

“Advanced”（高级）模式的“Capabilities”（功能）设置（如下图所示）比“Basic”（基本）模式多 2 个参数，如下所述。

图 119：“Capabilities” 选项卡的“Advanced” 模式



- “SRIOV Capabilities”（SRIOV 功能）：启用单根端口 I/O 虚拟化 (SR-IOV) 功能。集成块实现扩展单根端口 I/O 虚拟化 PCIe。启用时，SR-IOV 即可在所有选定物理功能上实现。启用 SR-IOV 功能时，禁用 MSI 支持，您可使用 MSI-X 支持，如上图所示。

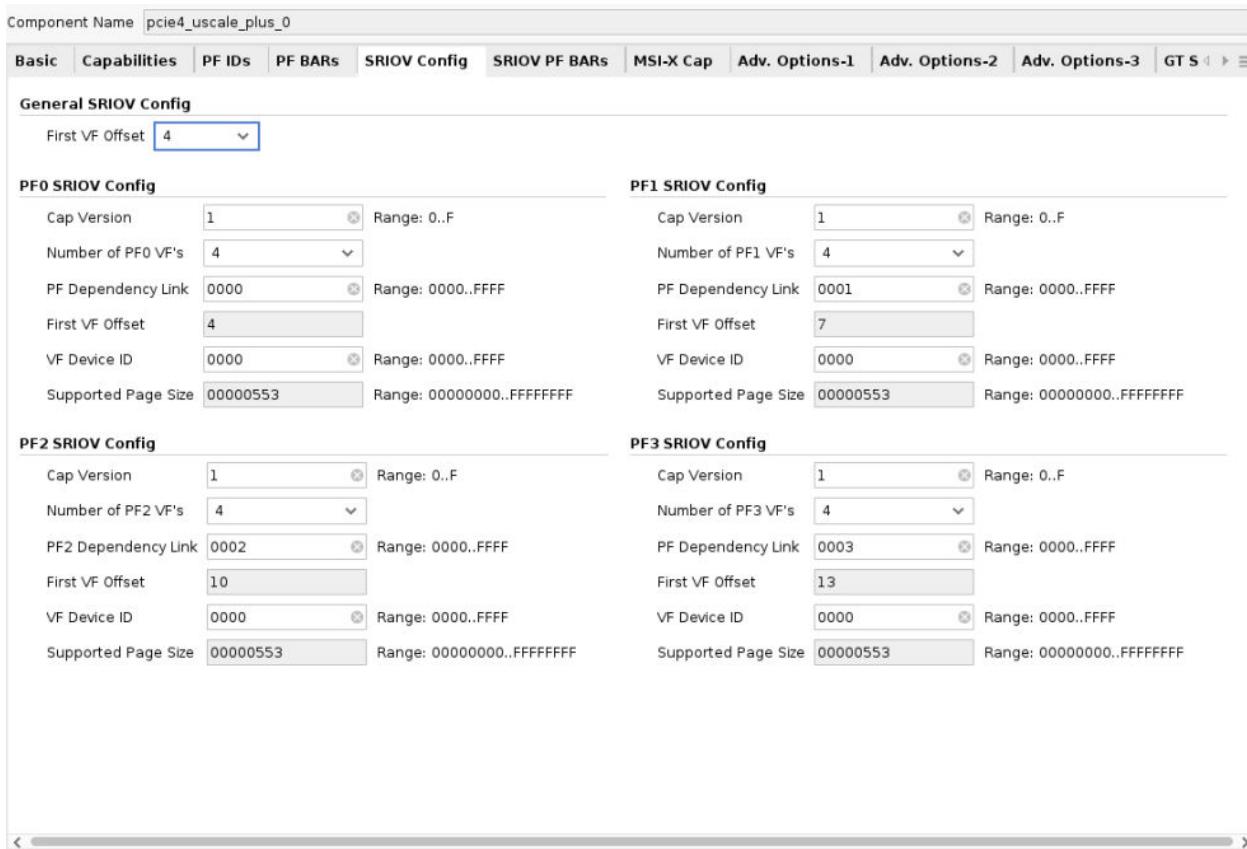
注释：启用 SR-IOV 功能时，禁用 MSI 支持，您可使用 MSI-X 支持。

- “Device Capabilities Registers 2”（器件功能寄存器 2）：指定对应 AtomicOps 和 TPH 完成器支持的选项。请参阅《PCI Express 基本规范》中的器件功能寄存器 2 描述以获取更多信息。这些设置适用于所有选定的物理功能。

“SRIOV Config” 选项卡

本章节中描述了 SRIOV 配置高级参数，如下图所示。

图 120：SRIOV 配置选项卡



- “General SRIOV Config”（通用 SRIOV 配置）：该值用于指定第 1 项 PF 的偏移，其中含至少 1 项已启用的 VF。启用 ARI 时，允许的值为 'd4' 或 'd64'，所有 PF 中的 VF 总数加上该字段的值不得超过 256。禁用 ARI 时，该字段将设为 1，这样仅支持 1PF 加上 7VF 的非 ARI SR-IOV 配置。
- “Cap Version”（功能版本）：表示对应物理功能的 4 位 SR-IOV 功能版本。
- “Number of PFx VF's”（PFx VF 数量）：表示与物理功能关联的虚拟功能的数量。共有 252 项虚拟功能可供在 4 项物理功能之间灵活使用。任何已启用的 PF 的 VF 数量均不得为 0。
- “PFx Dependency Link”（PFx 依赖关系链路）：表示对应物理功能的 SR-IOV 功能依赖关系链路。器件的编程模型的一组功能之间可能存在特定于供应商的依赖关系。“Function Dependency Link”（功能依赖关系链路）字段用于描述这些依赖关系。
- 首个 VF 偏移：表示对应物理功能 (PF) 的首个虚拟功能 (VF) 的偏移。PFx 偏移始终固定不变。PF0 驻留在偏移 0 处、PF1 驻留在偏移 1 处、PF2 驻留在偏移 2 处，而 PF3 则驻留在偏移 3 处。

共有 252 项虚拟功能可供使用。这些虚拟功能的功能编号范围为 4 到 255。

您可在自定义 GUI 中针对首个 VF 偏移选择 4 或 64，前提是最后一个 VF 偏移不超过 255。

- 示例：

- 当已启用的 VF 总数小于 192 时，请针对首个 VF 偏移选择 4 或 64。最后一个 VF 偏移将小于 255。
- 当已启用的 VF 总数超过 192 时，请针对首个 VF 偏移选择 4。在此情况下，针对首个 VF 偏移不允许使用 64，因为最后一个 VF 偏移将超过 255。

虚拟功能按顺序映射到 VF，但 PF 优先。例如，如果 PF0 具有 2 项虚拟功能，而 PF1 具有 3 项虚拟功能，则会发生以下映射：

PF_x_FIRST_VF_OFFSET 的计算方式为将物理功能的偏移减去虚拟功能的首个偏移。

```
PFx_FIRST_VF_OFFSET = (PFx first VF offset - PFx offset)
```

在以上示例中，使用的偏移如下：

```
PF0_FIRST_VF_OFFSET = (4 - 0) = 4  
PF1_FIRST_VF_OFFSET = (6 - 1) = 5
```

PF1 的初始偏移是连接到 PF0 的 VF 数量的函数，定义方式如以下伪代码所示：

```
PF1_FIRST_VF_OFFSET = FIRST_VF_OFFSET + NUM_PF0_VFs - 1
```

同样，对于其它 PF：

```
PF2_FIRST_VF_OFFSET = FIRST_VF_OFFSET + NUM_PF0_VFs + NUM_PF1_VFs - 2  
PF3_FIRST_VF_OFFSET =  
    FIRST_VF_OFFSET + NUM_PF0_VFs + NUM_PF1_VFs + NUM_PF2_VFs - 3
```

- VF 器件 ID：表示与物理功能关联的所有虚拟功能的 16 位器件 ID。
- “SRIOV Supported Page Size”（SRIOV 支持的页面大小）：表示物理功能支持的页面大小。此物理功能支持的页面大小为 $2^{(n+12)}$ ，前提是设置的位 n 为 32 位寄存器。

“SRIOV BARs” 选项卡

“SRIOV Base Address Registers (BARs)”（SRIOV 基址寄存器）可用于为端点配置设置基址寄存器空间。每个 BAR（0 到 5）均可用于配置 SR-IOV BAR 间隙大小和 SR-IOV 控制属性。

图 121：“SRIOV BARs” 选项卡：“Advanced” 模式

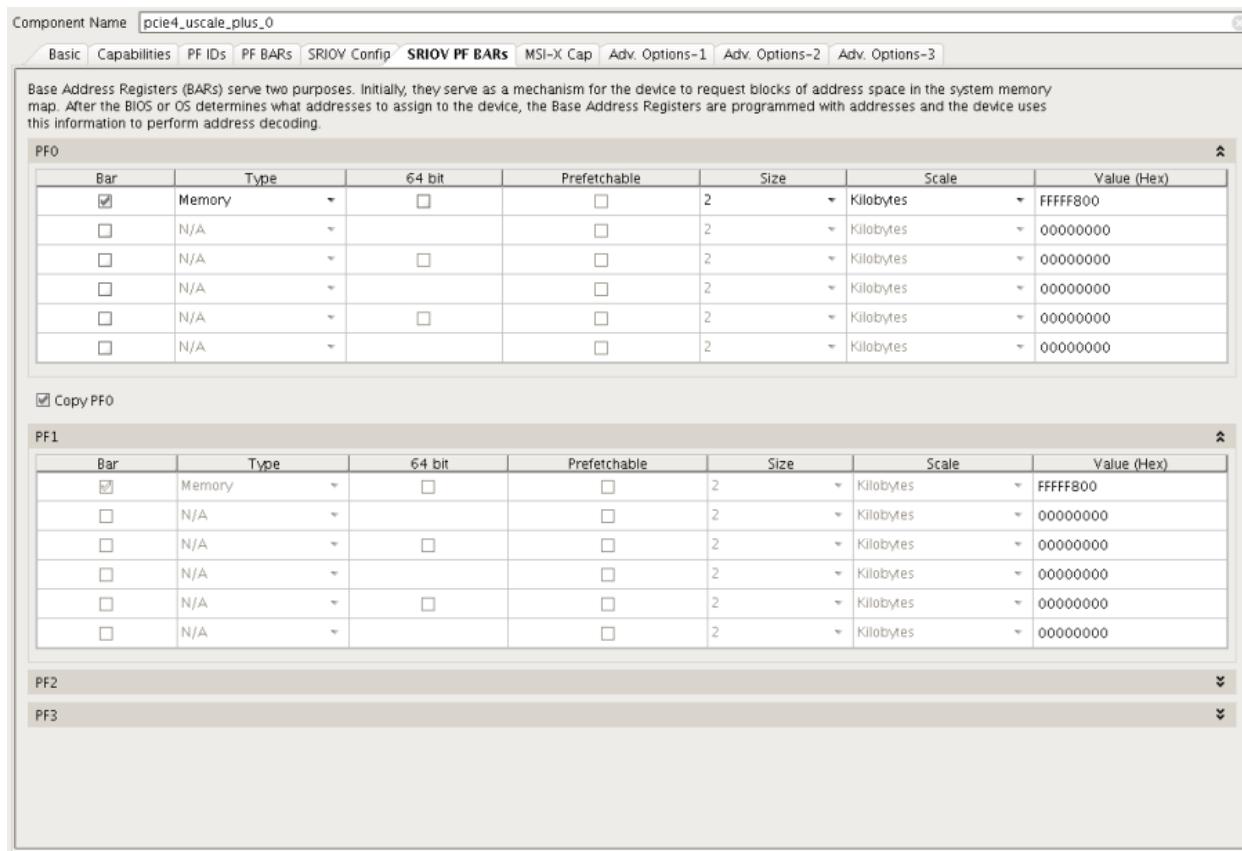


图 122：“SRIOV BARs” 选项卡：“Advanced” 模式

表 69：虚拟功能映射示例

物理功能	虚拟功能	功能编号范围
PF0	VF0	64
PF0	VF1	65
PF1	VF0	68
PF1	VF1	69
PF1	VF1	70

- SRIOV 基址寄存器概述：在端点配置中，核支持最多 6 个 32 位 BAR 或 3 个 64 位 BAR。在根端口配置中，核支持最多 2 个 32 位 BAR 或 1 个 64 位 BAR。SR-IOV BAR 分 2 种大小：
 - 32 位 BAR：地址空间最小可达 16 字节或者最大可达 2 GB。用于存储器到 I/O。
 - 64 位 BAR：地址空间最小可达 128 字节或者最大可达 256 千兆字节 (GB)。仅用于存储器。

所有 SR-IOV BAR 寄存器都具有下列选项：

- “Checkbox”（复选框）：单击复选框可启用 BAR；取消选择复选框可禁用 BAR。
- “Type”（类型）：SR-IOV BAR 可为 I/O 或存储器。

- I/O: I/O BAR 只能采用 32 位; “Prefetchable” (可预取) 选项不适用于 I/O BAR。仅限针对传统 PCI Express 端点才能启用 I/O BAR。
- 存储器: 存储器 BAR 可采用 64 位或 32 位, 并且可预取。当 BAR 设为 64 位时, 它使用下一个 BAR 作为扩展地址空间, 并使下一个 BAR 不可访问。
- “Size” (大小) : 可用大小范围取决于 PCIe 器件/端口类型和所选 BAR 类型。下表列出了可用 BAR 大小范围。

表 70: 器件配置的 SRIOV BAR 大小范围

PCIe 器件/端口类型	BAR 类型	BAR 大小范围
PCI Express 端点	32 位存储器	128 字节 - 2 GB
	64 位存储器	128 字节 - 8 EB
传统 PCI Express 端点	32 位存储器	16 字节 - 2 GB
	64 位存储器	16 字节 - 8 EB
	I/O	16 字节 - 2 GB
根端口模式	32 位存储器	16 字节 - 2 GB
	64 位存储器	4 字节 - 8 EB
	I/O	16 字节 - 2 GB

- “Prefetchable” (可预取) : 识别存储器空间预取功能。
- “Value” (值) : 表示基于当前选择分配给 BAR 的值。
- “Managing SRIOV Base Address Register Settings” (管理 SRIOV 基址寄存器设置) : 存储器、I/O、类型和可预取设置均可通过在 “Customize IP” (自定义 IP) 对话框中为目标基址寄存器设置相应的设置来进行处理。

存储器或 I/O 设置用于指示地址空间定义为存储器还是 I/O。基址寄存器仅响应访问指定地址空间的命令。通常, 应避免使用大小小于 4 KB 的存储器空间。允许的最小 I/O 空间为 16 字节。在所有新设计中应避免使用 I/O 空间。

只要对于读取没有副作用 (即, 读取数据不会像从 RAM 读取一样导致数据销毁), 即表示存储器空间可预取。在适当情况下, 多个字节写入操作可合并为单一双字写入操作。

针对非传统 PCIe 将核配置为端点时, 已设置可预取位的所有 SR-IOV BAR (BAR5 除外) 都必须支持 64 位寻址。针对未设置可预取位的所有 SR-IOV BAR, 则允许 32 位寻址。可预取位相关的要求不适用于传统端点。BAR 支持的最小存储器地址范围为 128 字节 (针对 PCI Express 端点) 和 16 字节 (针对传统 PCI Express 端点)。

- “Disabling Unused Resources” (禁用未使用的资源) : 为实现最佳结果, 请禁用未使用的基址, 以节省系统资源。通过在 “Customize IP” 对话框中取消选中未使用的 BAR 即可禁用基址寄存器。
- “Copy PFO” (复制 PFO) : 设置 “Copy PFO” 选项后, 即可允许您将剩余 PF 组的所有 BAR 设置都设为与 PFO 组相同的值。如有多个物理功能 (PF), 则适用该选项。

“MSI-X Capabilities” 选项卡

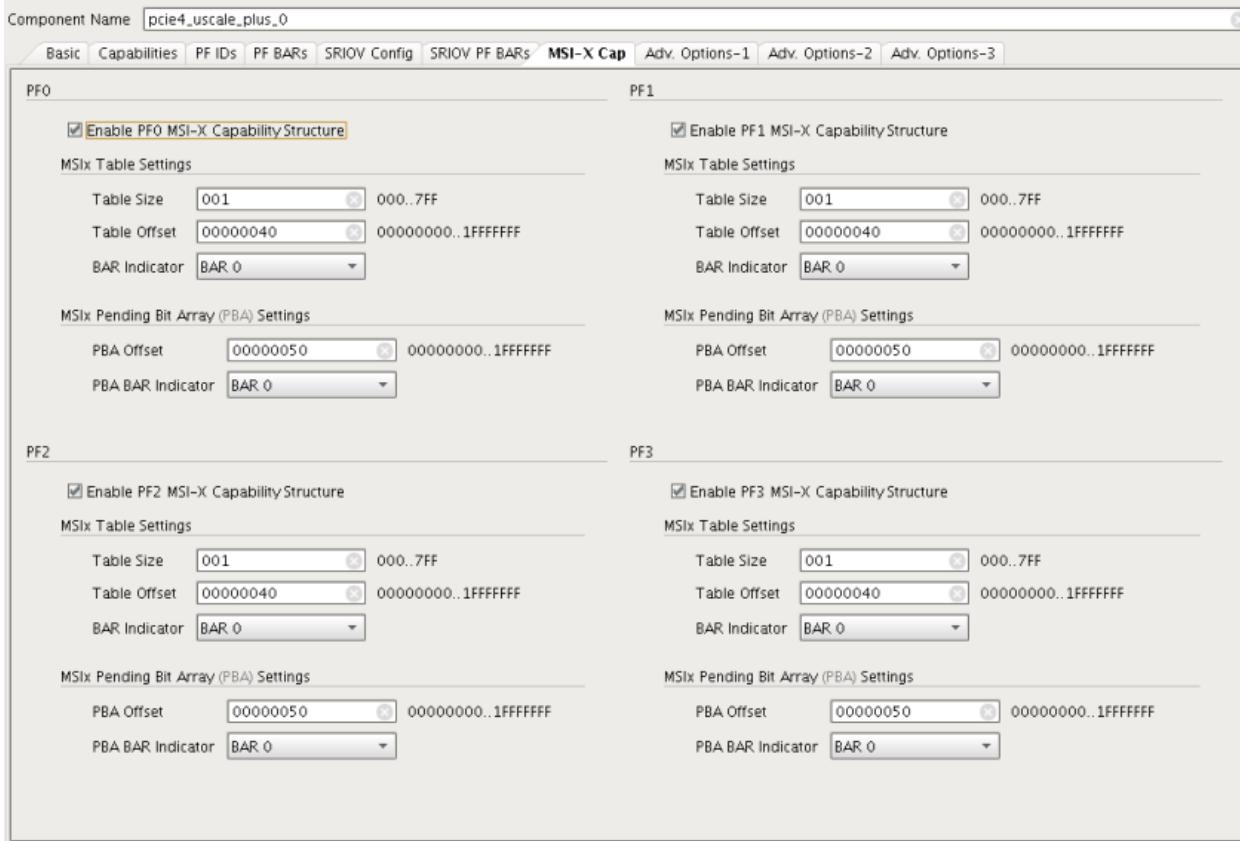
下图所示 “MSI-X Capabilities” (MSI-X 功能) 参数仅在 “Advanced” (高级) 模式下可用。要启用 MSI-X 功能, 请选中 “Advance” 模式, 然后在 “Capabilities” (功能) 页面上选择所需的选项。有 4 个选项可供选择。

- “MSI-X External” (MSI-X 外部) : 在此模式下, 您需要在核外部实现 MSI-X 外部接口驱动逻辑、MSI-X 表和 PBA 缓冲器。您可配置 MSI-X BAR。
- “MSI-X Internal” (MSI-X 内部) : 在此模式下, 您仅需实现 MSI-X 内部接口驱动逻辑。MSI-X 表和 PBA 缓冲器已构建到核内部。您可配置 MSI-X BAR。

- MSI-X AXI4-Stream：在此模式下，用户应在 AXI4-Stream 接口上驱动 MSI-X 中断。您可配置 MSI-X BAR。
- “None”（无）：不支持 MSI-X。

选中 SRIOV 功能时，也适用相同的 MSI-X 选项。

图 123：“MSI-X Cap”选项卡：“Advanced”模式



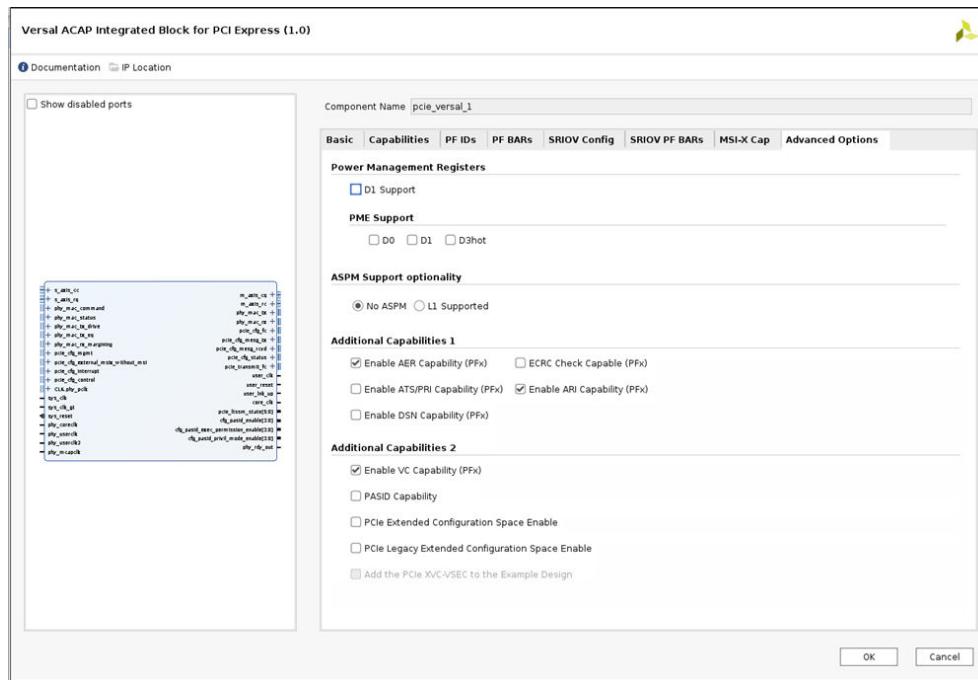
- “Enable MSI-X Capability Structure”（启用 MSI-X 功能结构）：表示存在 MSI-X 功能结构。

注释：功能结构需要配置至少 1 个存储器 BAR。您必须在应用中保留 MSI-X 表和暂挂位阵列。

- “MSI-X Table Settings”（MSI-X 表设置）：定义 MSI-X 表结构。
 - “Table Size”（表大小）：用于指定 MSI-X 表的大小。“Table Size”字段需 N-1 次中断（0x0F 将配置计数为 16 的表）。
 - “Table Offset”（表偏移）：指定对应指向 MSI-X 表的基址的基址寄存器的偏移量。
 - “BAR Indicator”（BAR 指示符）：用于指示配置空间内的基址寄存器，此基址寄存器用于将 MSI-X 表中的功能映射到存储器空间。对于 64 位基址寄存器，这表示下位 DWORD。
- “MSIx Pending Bit Array (PBA) Settings”（MSIx 暂挂位阵列 (PBA) 设置）：用于定义 MSI-X 暂挂位阵列 (PBA) 结构。
 - “PBA Offset”（PBA 偏移）：指定对应指向 MSI-X PBA 的基址的基址寄存器的偏移量。
 - “PBA BAR Indicator”（PBA BAR 指示符）：用于指示配置空间内的基址寄存器，此基址寄存器用于将 MSI-X PBA 中的功能映射到存储器空间。

“Advanced Options” 选项卡

图 124：“Advanced Options” 选项卡



- “Power Management and ASPM Support”（功耗管理和 ASPM 支持）：这部分允许您启用“Power Management Registers”（功耗管理寄存器）和 ASPM。仅当链路速度为 2.5 Gb/s 和 5.0 Gb/s 时，才支持 L0s。在根端口配置中不支持 ASPM L1/L0s。
- “Additional Capabilities 1”（额外功能 2）：这部分支持您为核选择 AER、ECRC、ATS、PRI、ARI 和 DSN 功能。
- “Additional Capabilities 2”（额外功能 2）：这部分支持您为核选择 VC、PASID 和用户定义的功能（PCIe 扩展配置空间和 PCIe 遗留扩展配置空间）。

Adv.Options-1

“Advanced Options”（高级选项）选项卡支持功耗管理寄存器和 ASPM。当链路速度并非 2.5 GT/s 或 5.0 GT/s 时，不提供 L0s 支持。

图 125：“Adv. Options-1”选项卡

Component Name: pcie4_uscale_plus_0

Basic | Capabilities | PF IDs | PF BARs | Legacy/MSI Cap | **Adv. Options-1** | Adv. Options-2 | Adv. Options-3 | GT Settings

Power Management Registers

D1 Support

PME Support

D0 D1 D3hot

ASPM Support optionality

No ASPM
 L0s Supported
 L1 Supported

Adv. Options-2

“Advanced Options”（高级选项）选项卡支持您为核选择 AER、ARI 和 DSN 功能。

图 126：“Adv. Options-2”选项卡

Component Name: pcie4_uscale_plus_0

Basic | Capabilities | PF IDs | PF BARs | Legacy/MSI Cap | Adv. Options-1 | **Adv. Options-2** | Adv. Options-3 | GT Settings | Shared Log

AER Capabilities

The Advanced Error Reporting(AER) Capability is an optional PCIe Extended Capability, which when enabled, allows advanced error control and reporting.

Enable AER Capability (PFx) ECRC Check Capable (PFx)

ARI Capability

The Alternative Routing ID-Interpretation (ARI) Capability is an optional PCIe Extended Capability, which when enabled, allows a device to support up to 256 functions by reducing the ID from 3 field vector (Bus Number, Device Number, Function Number) to a 2 field vector (Bus Number, Function Number).

Enable ARI Capability (PFx)

Device Serial Number Capability

The Device Serial Number (DSN) Capability is an optional PCIe Extended Capability, that contains a unique Device Serial Number. This identifier must be presented on the Device Serial Number Input pin of port.

Enable DSN Capability (PFx)

Adv.Options-3

“Advanced Options”（高级选项）选项卡支持您为核选择 TPH、VC 和用户定义的功能。

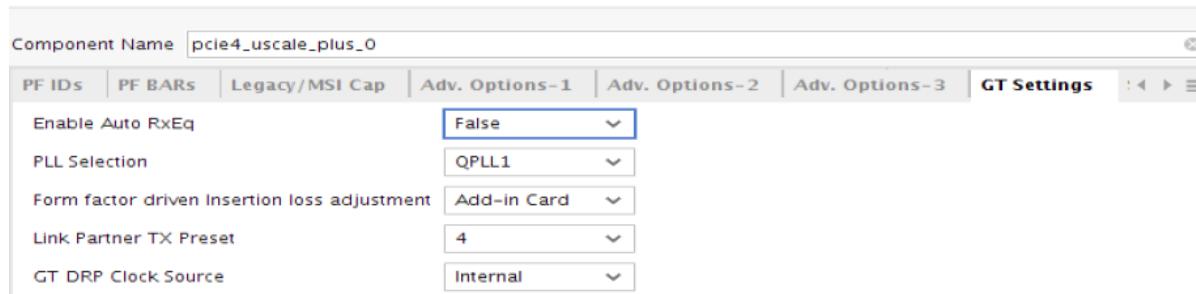
图 127：“Adv. Options-3”选项卡



“GT Settings”选项卡

“GT Settings”（GT 设置）选项卡中的设置允许您自定义通常不可访问的特定收发器设置。

图 128：“GT Settings”选项卡



- “PLL Selection”（PLL 选择）：（仅限选中 Gen2 链路时才可用）允许选择 QPLL1 或 CPLL 作为时钟源。当以 Gen2 链路速度运行时，如果希望附加协议位于相同 GT 四通道内，则可使用此功能。Gen3 速度需要 QPLL1，而 Gen1 速度则始终使用 CPLL。



重要提示！除非赛灵思有明确说明，否则其余设置不应修改。

下表显示了每种链路速度的选项和默认设置。

表 71：PLL 类型

链路速度	PLL 类型	注释
2.5_Gt/s	CPLL	默认值为 CPLL，不可供选择。
5.0_Gt/s	QPLL1 或 CPLL	默认值为 QPLL1，可供选择。
8.0_Gt/s	QPLL1	默认值为 QPLL1，不可供选择。

- “Enable Auto RxEq”（启用自动接收器均衡）：当此参数设置为 True 时，它会自动选择“Receiver Equalization”（接收器均衡）模式（LPM 或 DFE）。
 - True：默认值为 DFE，但将根据通道特性更改为 LPM。
 - False：默认值为 DFE 且可通过设置“Form Factor Driven Insertion Loss Adjustment”来进行更改。
- “Form Factor Driven Insertion Loss Adjustment”（外形尺寸驱动的插入损失调整）：表示根据外形尺寸选择，以奈奎斯特频率运行时发射器到接收器的插入损失。其中提供了 3 个选项：
 - “Chip-to-Chip”（芯片到芯片）：值为 5 dB。
 - “Add-in Card”（插卡）：这是默认选项，值为 15 dB。
 - “Backplane”（背板）：值为 20 dB。

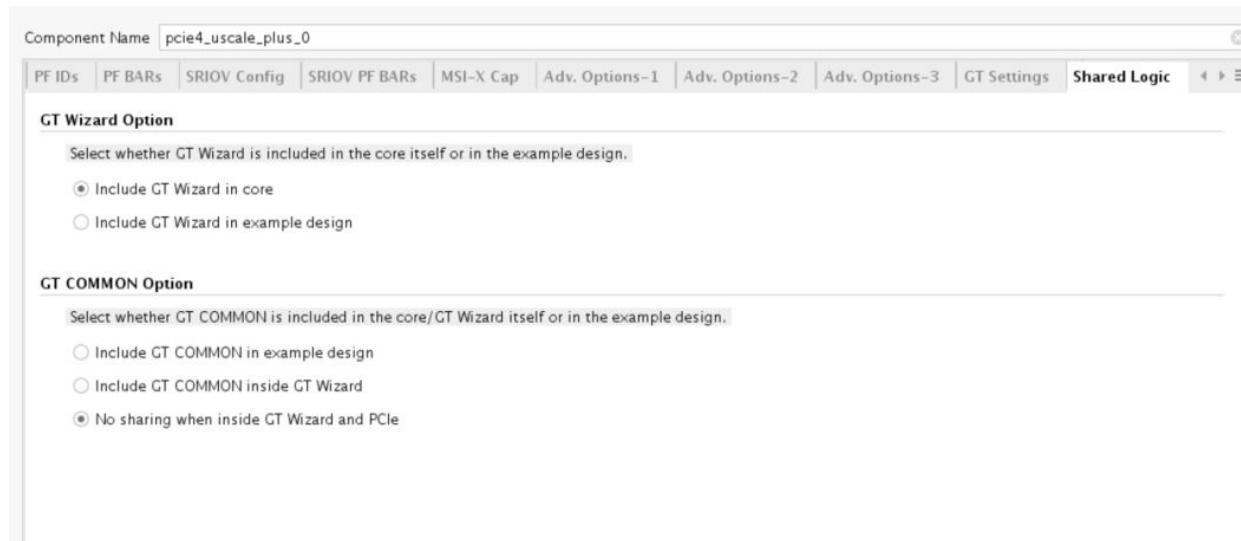
此插入损失值适用于 GT Wizard 子核。

- “Link Partner TX Preset”（链路伙伴 TX 预置）：默认值为 4，不建议更改。对于部分系统，预置值 5 可能更适合。在“GT Settings”（GT 设置）选项卡上提供了此参数以供使用。
- “GT DRP Clock Source”（GT DRP 时钟源）：添加该选项可供选择 GT 时钟源为外部时钟源还是内部时钟源。当选中“External”（外部）时，将从 300 MHz 的外部时钟源提供 DRP 时钟，并且在赛灵思顶层模块中，该时钟将被分割为 100/125 MHz。默认 GT DRP 时钟源为“Internal”（内部），但在“Add. Debug Options”（附加调试选项）页面中启用“Enable In System IBERT”（启用 In-System IBERT）选项时，默认时钟源为“External”。

“Shared Logic” 选项卡

下图显示了“Shared Logic”（共享逻辑）选项卡。

图 129: Shared Logic



- “GT Wizard Option”（GT Wizard 选项）：您可选择在设计示例中包含 GT Wizard，然后 GT Wizard IP 将被添加到设计示例区域中。您可重新配置 IP 以便进一步进行测试。默认情况下，在 PCIe IP 核中，GT Wizard IP 将作为层级 IP 交付，您无需对其进行重新自定义。如需获取信号描述及其它详细信息，请参阅《UltraScale 架构 GTY 收发器用户指南》(UG578) 或《UltraScale 架构 GTH 收发器用户指南》(UG576)。

- “GT COMMON Option”（GT COMMON 选项）：该选项用于共享设计中使用的 GT COMMON 块，前提是选中 Gen2（对应 PLL 选择为“QPLL1”）和 Gen3 链路速度。
- 当选中“Include GT COMMON in example design”（在设计示例中包含 GT COMMON）时，将在支持封装文件中提供 GT COMMON 块实例，此支持封装文件位于赛灵思顶层文件内，并且可供核或外部逻辑使用。
- 使用“Include GT COMMON inside GT Wizard”（在 GT Wizard 中包含 GT COMMON）时，GT COMMON 可供外部逻辑共享。
- 选中“No Sharing when inside GT Wizard and PCIe”（在 GT Wizard 和 PCIe 内部无共享）时，不允许共享 GT COMMON 块。
- 同时选中“Include GT COMMON in example design”和“Include GT Wizard in example design”时，必须使用来自相同配置的 GT Wizard IP 的设计示例工程的最新 GT COMMON 设置。此特定选项用于交付静态 GT COMMON 封装，其中包含仅适用于 2017.3 时间范围的最新设置。

附加调试选项

您可选择附加调试部分用于进行调试。相关参数如下所述。

- “Enable In System IBERT”（启用 In-System IBERT）：此调试选项用于检查串行链路的眼图，确认该链路是否按期望的链路速度运行。如需了解有关 In-System IBERT 的更多信息，请参阅《In-System IBERT LogiCORE IP 产品指南》(PG246)。



重要提示！ 该选项主要用于硬件调试。使用该选项时，不支持仿真。

眼图检查步骤如下：

1. 选择适合的赛灵思参考板。
2. 通过以下选项对核进行配置：
 - 选择“Gen3”、“Gen2”或“Gen1”链路速度（任意链路宽度）。
 - 在“Add. Debug Options”（附加调试选项）页面中选中的“Enable In System IBERT”（启用 In-System IBERT）。
3. 打开设计示例。
4. 生成 .bit 文件和 .ltx 文件。
5. 打开 Hardware Manger (HM) 并使用生成的 .bit 文件和 .ltx 文件配置器件。
6. 重启机器以重新扫描并重新运行枚举过程。
7. 选择位于 HM 底部的“Serial I/O links”（串行 I/O 链路）选项卡，并为扫描窗口创建链路。
8. 在“Serial I/O links”选项卡中选中任一链路，右键单击并选择“scan link”（扫描链路）选项。
9. 为了实现更好的结果，请尝试将“Horizontal”（水平）和“Vertical”（垂直）递增值从默认值调整为 2
10. 选中眼图扫描后，即可绘制出眼图。



重要提示！ “Enable In System IBERT”不应配合“GT Settings”（GT 设置）选项卡中的“Falling Edge Receiver Detect”（下降沿接收器检测）选项来使用。“Add. Debug Options”选项卡中的“Enable In System IBERT”选项会同时将“GT Settings”选项卡中的“GT DRP Clock Source”（GT DRP 时钟源）设置为“External”（外部）。

- “Enable Descrambler for Gen3 Mode”（为 Gen3 模式启用解扰器）：此调试选项在 PCIe 核内集成解扰器模块的加密版本，用于对往来采用 Gen3 链路速度模式的 PCIe 集成块的 PIPE 数据进行解扰。
- “Enable JTAG Debugger”（启用 JTAG 调试器）：该功能提供了易于使用的调试功能，用于执行：

- LTSSM 状态转换：显示从链路建立开始后执行的所有 LTSSM 状态转换。
- PHY 复位 FSM 转换：显示 PHY 复位 FSM（供 PCIe 解决方案 IP 使用的内部状态机）。
- 接收器检测：显示已成功完成“Receiver Detect”（接收器检测）的所有通道

步骤如下：

1. 打开新的 Vivado 工程，并连接至开发板。现在，您应可看到 hw_axi_1。



2. 在 Vivado Tcl Console 中，输入 source test_rd.tcl。

3. 对于后处理，请双击：

- draw_ltssm.tcl (Windows) 或 wish draw_ltssm.tcl
- draw_reset.tcl (Windows) 或 wish draw_reset.tcl
- draw_rxdet.tcl (Windows) 或 wish draw_rxdet.tcl

这样会以图形化方式显示 LTSSM 状态转换。

相关信息

硬件调试

“Core Interface Parameters” 选项卡

“Core Interface Parameters”（核接口参数）选项卡支持您将自己的应用不需要的接口禁用。

要支持设计示例仿真，请启用：

- “Config Management Interface”（配置管理接口），以支持端点设计示例仿真。
- “Config Control Interface”（配置控制接口），以支持根端口设计示例仿真。

图 130：“Core Interface Parameters” 选项卡



输出生成

如需了解更多详情，请参阅《Vivado Design Suite 用户指南：采用 IP 进行设计》(UG896)。

核约束

所需约束

UltraScale+ 器件 Integrated Block for PCI Express 解决方案需满足时序约束和其它物理实现约束的规格，方可满足指定的 PCI Express® 性能要求。在赛灵思设计约束 (XDC) 文件中，这些约束随端点和根端口解决方案一起提供。生成的 XDC 中的管脚分配和层级名称对应于所提供的设计示例。



重要提示！如果不使用设计示例顶层文件，请将参考时钟的 IBUFDS_GTE 实例、sys_rst 的 IBUF 实例以及与这 2 个实例关联的位置和时序约束一起复制到您的本地设计顶层。

为了达成一致的实现结果，通过赛灵思工具运行设计时，必须使用包含这些未经修改的原始约束的 XDC。如需获取有关 XDC 或特定约束的定义及其使用方式的更多详细信息，请参阅《Vivado Design Suite 用户指南：使用约束》(UG903)。

随集成块解决方案提供的约束已通过硬件测试，可提供一致结果。约束可修改，但前提是充分了解每个约束的影响。此外，如果设计背离所提供的约束，则对此类设计不予支持。

相关信息

[GT 位置](#)

器件、封装和速度等级选择

XDC 的器件选择部分可将有关设计的目标器件、封装和速度等级的信息告知实现工具。



重要提示！由于 XDC 是专为特定器件和封装组合所生成的，因此这部分不应修改。

器件选择部分始终包含器件选择行，但也包含特定于器件或封装的选项。以下显示了器件选择行示例：

```
CONFIG PART = XCKU040-ffva1156-3-e-es1
```

时钟频率、时钟管理和时钟布局

请参阅 [第 4 章：用核设计](#) 以获取有关时钟要求的详细信息。

bank 分配

本节不适用于此 IP 核。

收发器布局

本节不适用于此 IP 核。

I/O 标准与布局

本节不适用于此 IP 核。

软核逻辑布局

选中 512 位接口时，请复查设计示例中的赛灵思顶层 XDC 文件中的 Pblock 约束。这些约束是将软核 512 位 AXI4-Stream 逻辑保留在 PCIe 集成块附近以便改善时序所必需的。

调整集成块核的位置

默认情况下，IP 核级约束可将块 RAM、UltraRAM、收发器和集成块锁定到建议的位置。要调整这些块的位置，必须在 XDC 约束文件中覆盖这些块的约束。为此，请执行以下操作：

1. 从核级 XDC 约束文件复制需要覆盖的块的约束。
2. 将这些约束置于用户 XDC 约束文件中。
3. 将约束更新到新位置。

用户 XDC 约束通常限定为设计顶层；因此，请确保这些约束所引用的单元在复制粘贴后仍有效。通常，您需要以完整层级名称来更新模块路径。

注释：如果某些位置需要进行交换位置（即，新位置当前被另一个模块占据），有 2 种方式来执行此操作。

- 如有临时位置可用，请首先将第 1 个模块移至新的临时位置。然后，将第 2 个模块移至原先被第 1 个模块占据的位置。下一步，将第 1 个模块移至第 2 个模块的位置。这些步骤可在 XDC 约束文件中完成。
- 如果没有其它位置可用作为临时位置，请在 Tcl 命令窗口中对第 1 个模块使用 `reset_property` 命令，然后将第 2 个模块移至此位置。`reset_property` 命令无法在 XDC 约束文件中执行，必须从 Tcl 命令文件调用，或者直接输入 Tcl 控制台 (Tcl Console)。

仿真

有关 Vivado® 仿真组件的全面信息，以及与如何使用支持的第三方工具相关的信息，请参阅《Vivado Design Suite 用户指南：逻辑仿真》([UG900](#))。

如需获取有关设计示例仿真的更多信息，请参阅 [设计示例仿真](#)。

PIPE 模式仿真

Integrated Block for PCIe 支持 PIPE 模式仿真，在此模式下，核的 PIPE 接口连接到链路伙伴的 PIPE 接口。此模式可提升仿真速度。

在“Endpoint”（端点）模式和“Root Port”（根端口）模式下，均可使用“Customize IP”（自定义 IP）对话框的“Basic”（基本）选项卡上的“Enable External PIPE Interface”（启用外部 PIPE 接口）在当前 Vivado Design Suite 解决方案设计示例中启用 PIPE 模式仿真。在核边界处生成的外部 PIPE 接口信号可用于访问外部器件。启用该功能还可提供必要的挂钩，以便使用第三方 PCI Express® VIP/BFM 替代随设计示例提供的 Root Port 模型。



提示：PIPE 模式仅用于仿真。不支持实现。

欲知详情，请参阅“[Basic 选项卡](#)”中的“启用外部 PIPE 接口”选项。

下表描述了核顶层可用的 PIPE 总线信号及其在 EP 核 (pcie_top) PIPE 信号内的对应映射。



重要提示！在仿真目录内提供了一个新文件 `xil_sig2pipe.v`，用于取代 `phy_sig_gen.v`。BFM/VIP 应与 `board.v` 中的 `xil_sig2pipe` 实例相连。

表 72：常用输入/输出命令和端点 PIPE 信号映射

输入命令	端点 PIPE 信号映射	输出命令	端点 PIPE 信号映射
common_commands_in[25:0]	不使用	common_commands_out[0]	pipe_clk ¹
		common_commands_out[2:1]	pipe_tx_rate_gt ²
		common_commands_out[3]	pipe_tx_rcvr_det_gt
		common_commands_out[6:4]	pipe_tx_margin_gt
		common_commands_out[7]	pipe_tx_swing_gt
		common_commands_out[8]	pipe_tx_reset_gt
		common_commands_out[9]	pipe_tx_deemph_gt
		common_commands_out[16:10]	不使用 ³

注释：

1. `pipe_clk` 是基于核配置的输出时钟。对于 Gen1 速率，`pipe_clk` 为 125 MHz。对于 Gen2 和 Gen3，`pipe_clk` 为 250 MHz。
2. `pipe_tx_rate_gt` 表示流水线速率 (2'b00-Gen1、2'b01-Gen2 和 2'b10-Gen3)。
3. 此端口的功能已被弃用，可将其保留并保持未连接状态。

表 73：输入/输出总线与端点 PIPE 信号映射

输入总线	端点 PIPE 信号映射	输出总线	端点 PIPE 信号映射
pipe_rx_0_sigs[31:0]	pipe_rx0_data_gt	pipe_tx_0_sigs[31: 0]	pipe_tx0_data_gt
pipe_rx_0_sigs[33:32]	pipe_rx0_char_is_k_gt	pipe_tx_0_sigs[33:32]	pipe_tx0_char_is_k_gt
pipe_rx_0_sigs[34]	pipe_rx0_elec_idle_gt	pipe_tx_0_sigs[34]	pipe_tx0_elec_idle_gt
pipe_rx_0_sigs[35]	pipe_rx0_data_valid_gt	pipe_tx_0_sigs[35]	pipe_tx0_data_valid_gt
pipe_rx_0_sigs[36]	pipe_rx0_start_block_gt	pipe_tx_0_sigs[36]	pipe_tx0_start_block_gt
pipe_rx_0_sigs[38:37]	pipe_rx0_synheader_gt	pipe_tx_0_sigs[38:37]	pipe_tx0_synheader_gt
pipe_rx_0_sigs[83:39]	不使用	pipe_tx_0_sigs[39]	pipe_tx0_polarity_gt
		pipe_tx_0_sigs[41:40]	pipe_tx0_powerdown_gt
		pipe_tx_0_sigs[69:42]	不使用 ¹

注释：

1. 此端口的功能已被弃用，可将其保留并保持未连接状态。

综合与实现

如需了解有关综合和实现方面的详情，请参阅《Vivado Design Suite 用户指南：采用 IP 进行设计》([UG896](#))。

如需获取有关设计示例的综合和实现的信息，请参阅 [设计示例的综合和实现](#)。

设计示例

本章包含有关 Vivado® Design Suite 中提供的设计示例的信息。

设计示例概述

本章提供了 UltraScale+ 器件 Integrated Block for PCI Express 核设计示例。

集成块端点配置概述

集成块的端点配置的仿真设计示例由 2 个不同部分组成：

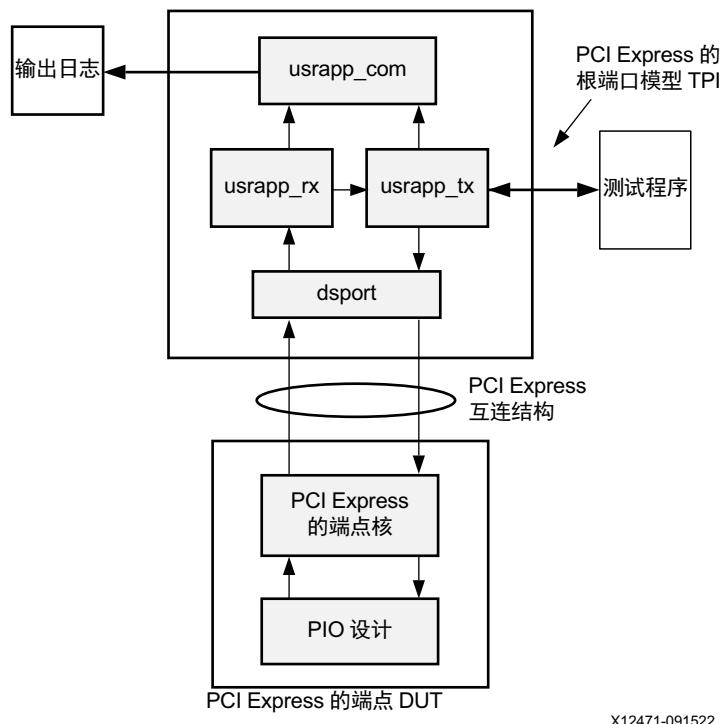
- 测试激励文件生成的根端口模型 (Root Port Model) 可生成、耗用并检查 PCI Express® 总线流量。
- 编程输入/输出 (PIO) 设计示例是 PCI Express 的完成器应用。PIO 设计示例会响应其存储器空间内的读取和写入请求，可对其加以综合以便在硬件中进行测试。

注释：并非所有模式都支持设计示例，例如，跨接、地址对齐模式、SRIOV、MSI-X、FLR 和 MSI。用于虚拟功能的 FLR 在设计示例中无法完全实现；要将其完全实现，则需要修改用户逻辑。

仿真设计概述

对于仿真设计，传输事务从根端口模型发送到核（配置为端点）并由 PIO 设计示例来处理。下图演示了随核提供的仿真设计。如需获取有关根端口模型的更多信息，请参阅 [对应端点的根端口模型测试激励文件](#)。

图 131：仿真设计示例模块框图

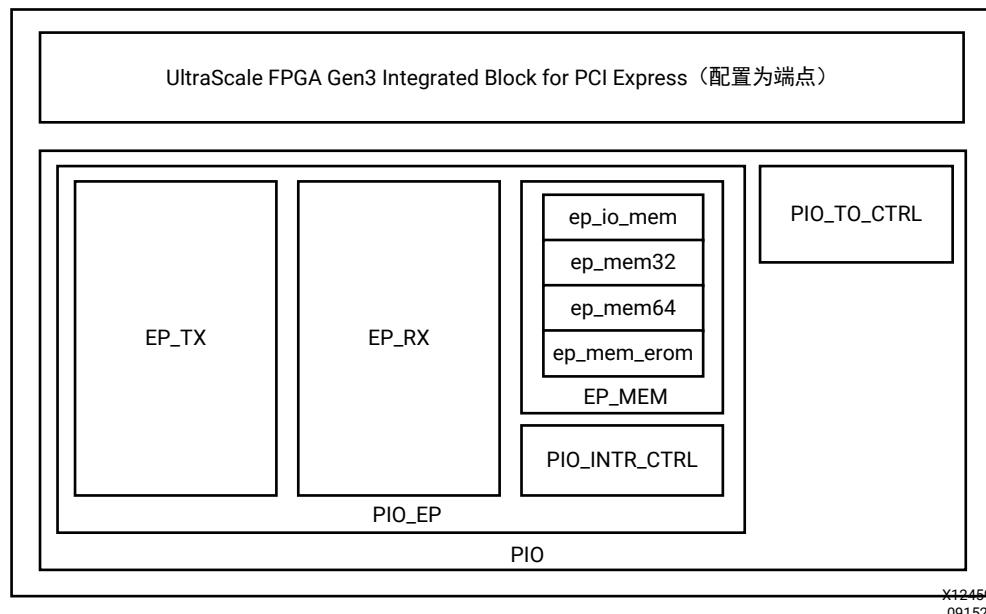


X12471-091522

实现设计概述

实现设计包含简单的 PIO 示例，可接受读取和写入传输事务并响应请求，如下图所示。随核提供了示例的源代码。如需了解有关 PIO 设计示例的更多信息，请参阅 [编程输入/输出：端点设计示例](#)。

图 132：实现设计示例模块框图



设计示例元素

PIO 设计示例元素包括：

- 核封装文件
- 用于例化核以及设计示例的 Verilog HDL 封装文件示例
- 可定制的演示测试激励文件，用于对设计示例进行仿真

设计示例已通过 Vivado Design Suite 以及下列仿真器的测试和验证：

- Vivado 仿真器
- Questa Advanced Simulator
- Synopsys Verilog Compiler Simulator (VCS)

如需了解受支持的工具版本的相关信息，请参阅[赛灵思设计工具：版本说明指南](#)。

编程输入/输出：端点设计示例

编程输入/输出 (PIO) 传输事务通常可供 PCI Express® 系统主机 CPU 用于访问 PCI Express 逻辑中的存储器映射输入/输出 (MMIO) 和配置映射输入/输出 (CMIO) 位置。PCI Express 端点可接受存储器请求和 I/O 写入传输事务，并以含数据的完成包传输事务来响应存储器请求和 I/O 读取传输事务。

在 Vivado® IP 目录所生成的核（端点配置）下包含 PIO 设计示例（PIO 设计），以便您使用已知确认有效的设计来初始化设计，以验证开发板的链路核功能。

核、Endpoint Block Plus for PCI Express 和 Endpoint PIPE for PCI Express 解决方案共享 PIO 设计端口模型。本章节一般情况下可代表名为 Endpoint for PCI Express（或 Endpoint for PCIe®）的所有解决方案。

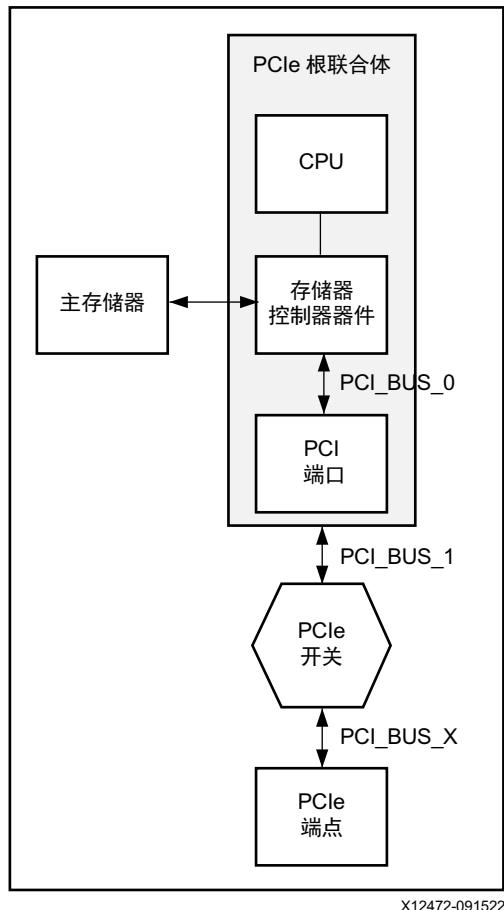
系统概述

PIO 设计是以目标为唯一导向的简单应用，与 PCIe 核传输事务 (AXI4-Stream) 接口的端点相连，可供您用作为起点来构建自己的设计。其功能包括：

- 在“Address Align Mode”（地址对齐模式）下，有 4 个传输事务专用的 2 KB 目标区域通过使用内部 FPGA 块 RAM 来提供总计 8,192 字节的目标空间。
- 在地址对齐模式下，支持单 Dword 有效载荷读写 PCI Express 传输事务，可传输至 32 位或 64 位地址存储器空间，并支持完成 TLP。
- 对于 Dword 对齐模式，PIO 设计支持多 Dword 有效载荷（最多 256 DW）读写 PCI Express 传输事务，可传输至 32 位地址存储器空间，支持完成 TLP。使用核的 BAR ID[2:0] 和完成器请求描述符 [114:112] 来区分 TLP 目标基址寄存器。
- 为 64 位、128 位、256 位和 512 位 AXI4-Stream 接口分别提供经过最优化的实现。

下图演示了 PCI Express 系统架构组件，其中包括根联合体、PCI Express 开关器件和 PCIe 端点。PIO 操作可将来自下游根联合体 (CPU 寄存器) 的数据移至端点，和/或将来自上游端点的数据移至根联合体 (CPU 寄存器)。在任一情况下，迁移数据的 PCI Express 协议请求均由主机 CPU 发起。

图 133：系统概述



X12472-091522

当 CPU 发出将寄存器存储至 MMIO 地址命令时，就会将数据向下游迁移。根联合体通常会生成存储器写入 TLP，其中包含相应的 MMIO 位置地址、字节使能和寄存器内容。当端点接收到存储器写入 TLP 并更新对应的本地寄存器后，此传输事务即告终止。

当 CPU 发出从 MMIO 地址加载寄存器命令时，就会将数据向上游迁移。根联合体通常会生成存储器读取 TLP，其中包含相应的 MMIO 位置地址和字节使能。端点会在接收到存储器读取 TLP 后生成含数据完成 TLP。此完成包会被导向到根联合体，有效载荷将加载到目标寄存器中，并完成传输事务。

PIO 硬件

对于地址对齐模式，PIO 设计会在 FPGA 块 RAM 中实现 1 个 8,192 字节的目标空间，此空间位于 PCIe 端点之后。此 32 位目标空间可通过单 Dword I/O 读取、I/O 写入、存储器读取 64、存储器写入 64、存储器读取 32 和存储器写入 32 TLP 来访问。

对于核向 PIO 设计提供的有效的存储器读取 32 TLP、存储器读取 64 TLP 或 I/O 读取 TLP 请求，PIO 设计会生成 1 个完成包（其有效载荷含 1 个 Dword）以作为响应。此外，此 PIO 设计还会为 I/O 写入 TLP 请求返回无数据完成包（状态为成功）。对于 Dword 对齐模式，PIO 设计会在 FPGA 块 RAM 中实现 2048 字节的目标空间。此目标空间和数据宽度因 AXI4-Stream 接口而异，且等于 AXI4-Stream 接口的宽度。此目标空间可通过存储器写入 32 TLP 和存储器读取 32 TLP 来访问。

对于来自核的有效的存储器读取 32 TLP 请求，PIO 会生成完成包（大小即有效载荷的大小）以作为响应。



PIO 设计可启动：

- 存储器读取传输事务，前提是接收的写入地址为 `11'hEA8`、写入数据为 `32'hAAAA_BBBB`，且目标为 BAR0。
- 遗留中断，前提是接收的写入地址为 `11'hEEC`、写入数据为 `32'hCCCC_DDDD` 且目标为 BAR0。
- MSI，前提是接收的写入地址为 `11'hEEC`、写入数据为 `32'hEEEE_FFFF` 且目标为 BAR0。
- MSIX，前提是接收的写入地址为 `11'hEEC`、写入数据为 `32'hDEAD_BEEF` 且目标为 BAR0。

PIO 设计在地址对齐模式下可处理含单 Dword 有效载荷的存储器或 I/O 写入 TLP，在 Dword 对齐模式下则可处理多 Dword 有效载荷，方法是将有效载荷更新到 FPGA 块 RAM 空间的目标地址中。

基址寄存器支持

在地址对齐模式下，PIO 设计支持 4 个离散目标空间，每个空间均包含 2 KB 的存储器块，由单独的基址寄存器 (BAR) 来表示。通过使用默认参数，Vivado® IP 目录可生成核，并将其配置为配合本章节中定义的 PIO 设计来运行，其中包括：

- 1 个 64 位可寻址存储器空间 BAR
- 1 个 32 位可寻址存储器空间 BAR

您可更改 PIO 设计所使用的默认参数；但在某些情况下，您可能需要根据自己的系统来更改用户应用。请参阅 [更改 IP 目录工具默认 BAR 设置](#) 以获取有关更改默认 Vivado Design Suite IP 参数及其对于 PIO 设计的影响的相关信息。

由 BAR 所表示的 4 个 2 KB 地址空间各自对应于 PIO 设计中 4 个 2 KB 地址区域中的 1 个地址区域。每个 2 KB 区域都是使用 1 个 2 KB 双端口块 RAM 来实现的。由于核可接收到传输事务，因此核会对地址进行解码，并判定 4 个区域中哪个区域才是目标。核会向 PIO 设计展示 TLP，并断言 (BAR ID[2:0]) 完成器请求描述符 [114:112] 的相应的位，如下表所述。

表 74：TLP 流量类型

块 RAM	TLP 传输事务类型	默认 BAR	BAR ID[2:0]
ep_io_mem	I/O TLP 传输事务	禁用	禁用
ep_mem32	32 位地址存储器 TLP 传输事务	2	000b
ep_mem64	64 位地址存储器 TLP 传输事务	0-1	001b
ep_mem_erm	32 位地址存储器 TLP 传输事务（目标为 EROM）	扩展 ROM	110b

对于 Dword 对齐模式，PIO 设计支持 1 个目标空间，其中包含 2048 字节的存储器。此存储器是使用 SDRAM 实现的。由于核可接收到传输事务，核会将 TLP 提供给 PIO 设计，并将 BAR ID[2:0] 和完成器请求描述符 [114:112] 的有效位断言为 `001b`。

更改 IP 目录工具默认 BAR 设置

您可更改 Vivado® IP 目录参数并继续使用 PIO 设计来创建自定义 Verilog 源代码以匹配所选 BAR 设置。但由于 PIO 设计参数的局限性比核参数更大，因此在更改默认 IP 目录参数时请注意以下设计示例限制：

- 设计示例支持 1 个 I/O 空间 BAR、1 个 32 位存储器空间（不能使用扩展 ROM 空间）和 1 个 64 位存储器空间。如果超出这些限制，那么仅限给定类型的首个空间保持活动 - 访问其它空间不会导致生成完成包。
- 每个空间均以 2 KB 存储器来实现。如果对应 BAR 配置为更宽的间隙，那么访问超出 2 KB 限制时会导致卷绕并与 2 KB 存储器空间重叠。
- PIO 设计支持 1 个 I/O 空间 BAR，此空间默认处于禁用状态，但可根据需要加以更改。

虽然 PIO 设计存在限制，但您可使用提供的 Verilog 源代码根据自己的特定需求对设计示例进行微调。

TLP 数据流

本章节定义了 PIO 设计成功处理 TLP 的数据流程。

PIO 能够成功处理单 Dword 有效载荷存储器读取和写入（对应地址对齐模式）、多 Dword 有效载荷（对应 Dword 对齐模式）以及 I/O 读取和写入 TLP（仅针对地址对齐模式受支持）。PIO 设计无法正确处理长度超过 1 个 Dword 的存储器读取或存储器写入 TLP。但对于地址对齐模式，核则接受这些 TLP，并将其传递给 PIO 设计。如果 PIO 设计收到长度超过 1 个 Dword 的 TLP，那么可从核完全接收此 TLP 并将其丢弃。不会生成任何对应的完成包。对于 Dword 对齐模式，如果存储器读取和存储器写入 TLP 的有效载荷包含多个 Dword，则 PIO 设计可正确予以支持和处理。PIO 设计可从核完全接收此 TLP，然后生成对应的完成包。

存储器和 I/O 写入 TLP 处理

当 PCIe® 的端点接收到存储器或 I/O 写入 TLP 时，TLP 目标地址和传输事务类型将与核 BAR 中的值进行比对。如果 TLP 通过此比对检查，那么核会将此 TLP 传递到 PIO 设计的 AXI4-Stream 接收接口。PIO 设计采用不同方式来处理存储器写入和 I/O TLP 写入：PIO 设计通过生成无数据完成 (cpl) 包来响应 I/O 写入，这符合 PCI Express 规范的要求。

AXI4-Stream 完成器请求接口会随包起始、包结束和就绪握手信号一起，断言相应的 (BAR ID[2:0]) 完成器请求描述符 [114:112] 信号有效，以向 PIO 设计标示与传入 TLP 匹配的特定目标 BAR。在接收时，PIO 设计 RX 状态机会处理传入的写入 TLP 并提取 TLP 数据和相关地址字段，以便将这些信息一起传递到 PIO 设计内部块 RAM 写入请求控制器。

在地址对齐模式下，根据断言有效的特定 BAR ID[2:0] 信号，RX 状态机会先向内部写入控制器标示要使用的相应 2 KB 块 RAM，然后再断言写入使能请求有效。例如，如果以 BAR0 为目标的核接收到 I/O 写入请求，那么该核会将此 TLP 传递到 PIO 设计，并将 BAR ID[2:0] 设置为 000b。RX 状态机会从 I/O 写入 TLP 提取下位地址位和数据字段，并指令内部的存储器写入控制器开始写入此块 RAM。

而对于 Dword 对齐模式，当 BAR ID[2:0] = 01b 时，RX 状态机会断言写入使能请求。RX 状态机会从存储器 32 写入 TLP 提取下位地址位和数据字段，并指令内部的存储器写入控制器开始写入此块 RAM。

在此示例中，将 BAR ID[2:0] 设置为 000b 并断言此设置有效导致指令 PIO 存储器写入控制器访问 ep_memo (默认情况下，这表示 2 KB 的 I/O 空间)。对 FPGA 块 RAM 执行写入时，PIO 设计 RX 状态机会断言 m_axis_cq_tready 无效，导致 AXI4-Stream 接收接口暂停接收其它 TLP，直至内部的存储器写入控制器写入块 RAM 为止。对于使用该核的设计而言，并非所有设计都需要以此方式断言 m_axis_cq_tready 无效；PIO 设计使用此方法可简化 RX 状态机的控制逻辑。

存储器和 I/O 读取 TLP 处理

当 PCIe® 的端点接收到存储器或 I/O 读取 TLP 时，TLP 目标地址和传输事务类型将与核 BAR 中编程的值进行比对。如果 TLP 通过此比对检查，那么核会将此 TLP 传递到 PIO 设计的 AXI4-Stream 接收接口。

AXI4-Stream 完成器请求接口会随包起始、包结束和就绪握手信号一起，断言相应的 BAR ID[2:0] 信号有效，以向 PIO 设计标示与传入 TLP 匹配的特定目标 BAR。在接收时，PIO 设计状态机会处理传入的读取 TLP，提取相关 TLP 信息并将其传递给 PIO 设计的内部块 RAM 读取请求控制器。

在地址对齐模式下，根据断言有效的特定 BAR ID[2:0] 信号，RX 状态机会先向内部读取请求控制器标示要使用的相应 2 KB 块 RAM，然后再断言读取使能请求有效。而对于 Dword 对齐模式，RX 状态机会基于 BAR ID [2:0] 检查请求是否对应于存储器读取 32 TLP，以启用读取请求并丢弃所有其它请求。例如，如果以默认 Mem32 BAR2 为目标的核接收到存储器读取 32 请求 TLP，那么该核会将此 TLP 传递到 PIO 设计，并将 BAR ID[2:0] 设置为 010b。RX 状态机会从存储器 32 读取 TLP 提取下位地址位，并指令内部的存储器读取请求控制器开始读取操作。

在此示例中，将 BAR ID[2:0] 设置为 010b 会指令 PIO 存储器读取控制器访问 Mem32 空间，默认情况下，这表示 2 KB 的存储器空间。存储器写入和读取 TLP 的处理方法之间的显著差异在于，对于存储器或 I/O 读取请求，要求接收方器件返回含数据完成 TLP。

处理读取时，PIO 设计 RX 状态机会断言 `m_axis_cq_tready` 无效，这将导致 AXI4-Stream 接收接口暂停接收任何其它 TLP，直至内部的存储器读取控制器完成从块 RAM 进行的读取访问并生成完成包为止。对于使用该核的设计而言，并非所有设计都需要以此方式断言 `m_axis_cq_tready` 无效。PIO 设计使用此方法可简化 RX 状态机的控制逻辑。

PIO 文件结构

下表定义了 PIO 设计文件结构。根据具体的目标核，Vivado® IP 目录中并非所有文件都需要交付，部分文件可能无需交付。主要差别在于部分 PCIe® 端点解决方案使用 32 位用户数据路径，其它解决方案则使用 64 位数据路径，而 PIO 设计则对于这两种情况都适用。数据路径宽度取决于具体的目标核。

表 75：PIO 设计文件结构

文件	描述
PIO.v	顶层设计封装
PIO_INTR_CTRL.v	PIO 中断控制器
PIO_EP.v	PIO 应用模块
PIO_TO_CTRL.v	PIO 关闭控制器模块
PIO_RX_ENGINE.v	32 位接收引擎
PIO_TX_ENGINE.v	32 位发射引擎
PIO_EP_MEM_ACCESS.v	端点存储器访问模块
PIO_EP_MEM.v	端点存储器
PIO_EP_XPM_SDRAM_WRAP.v	采用 Dword 对齐模式的端点存储器

已提供 4 种 PIO 设计配置：PIO_64、PIO_128 以及分别含 64 位、128 位、256 位和 512 位 AXI4-Stream 接口的 PIO_256。所生成的 PIO 配置取决于所选端点类型、PCI Express 通道数量以及所选接口宽度。下表识别了根据您的选择所生成的 PIO 配置。

表 76：PIO 配置

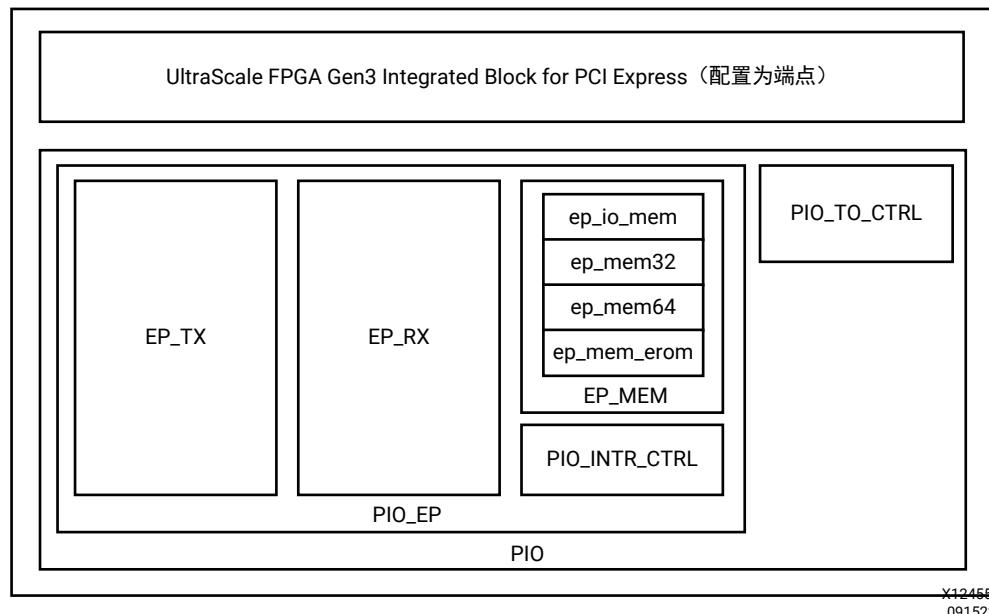
核	x1	x2	x4	x8
Integrated Block for PCIe	PIO_64	PIO_64 和 PIO_128	PIO_64、PIO_128 和 PIO_256	PIO_64、PIO_128 ¹ 和 PIO_256

注释：

- 该核不支持 128 位 x8 8.0 Gb/s 配置和 500 MHz 用户时钟频率。

下图显示了 PIO 设计的各组件，可分为 4 个主要部分：发射引擎、接收引擎、存储器访问控制器和功耗管理关闭控制器。

图 134：PIO 设计组件

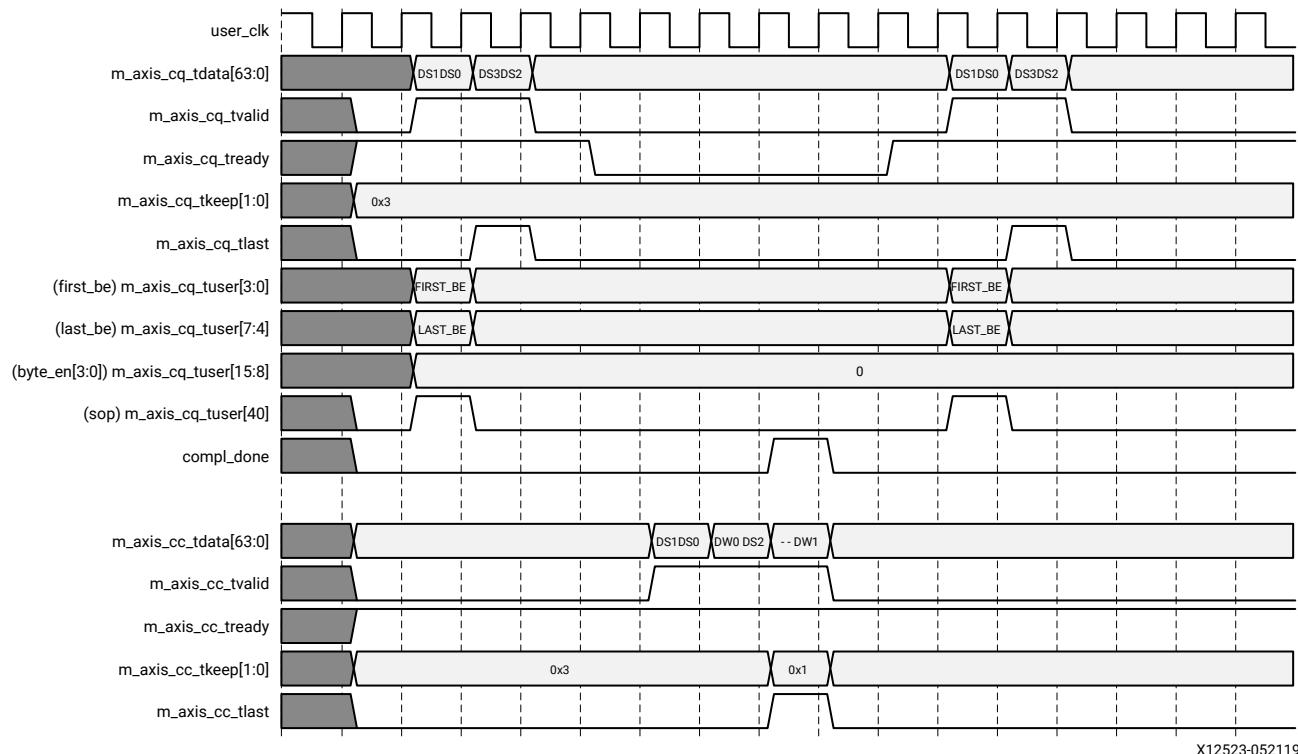


PIO 操作

PIO 读取传输事务

下图显示了对 PIO 设计发出的连续存储器读取请求。完全接收到第一个 TLP 后，接收引擎会立即断言 `m_axis_rx_tready` 无效。仅当发射引擎断言 `compl_done_o` 有效以表明已成功发射第一个请求的完成包之后，才会接受下一项读取传输事务。

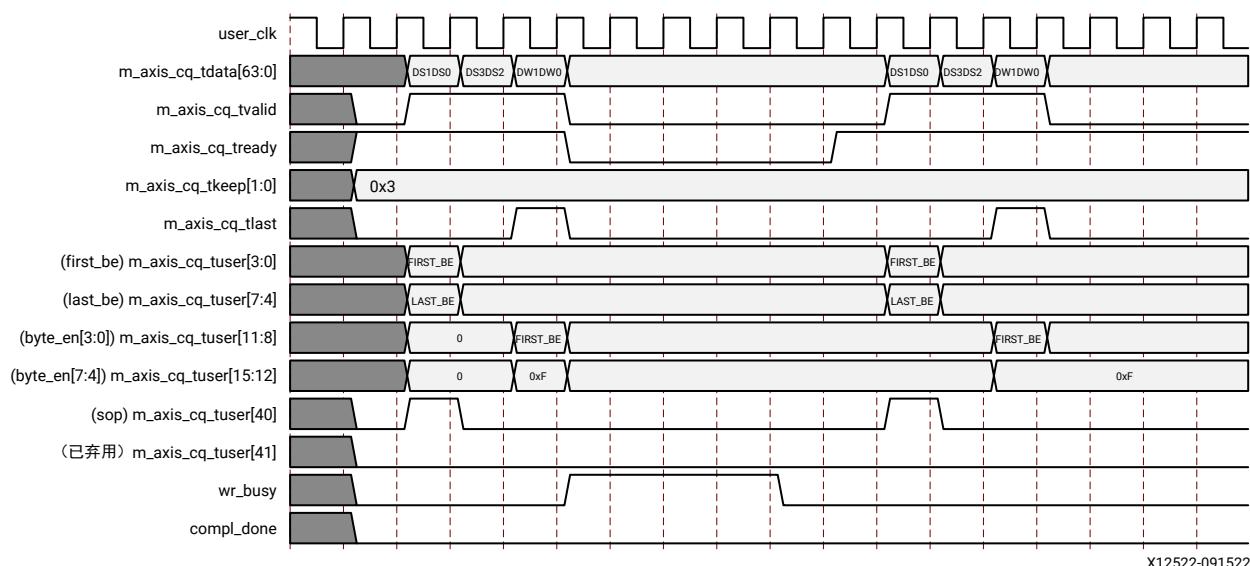
图 135：连续读取传输事务



PIO 写入传输事务

下图显示了对 PIO 设计发出的连续存储器写入请求。仅当存储器访问单元断言 wr_busy_o 无效以表示与第一个请求关联的数据已成功写入存储器间隙之后，才会接受下一个写入传输事务。

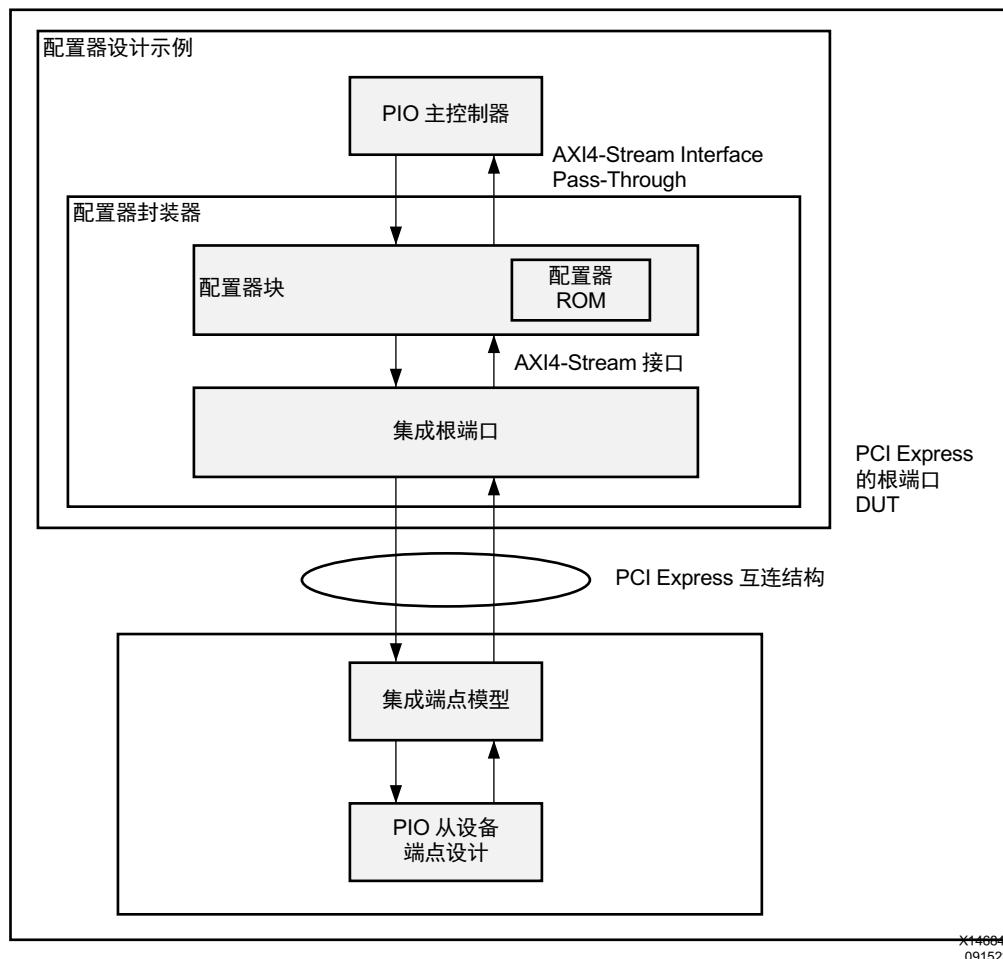
图 136：连续写入传输事务



配置器：根端口设计示例

下图显示了总体系统视图中块的连接方式。

图 137：配置器设计示例



配置器文件结构

下表定义了配置器设计示例的文件结构。

表 77：设计示例文件结构

文件	描述
xilinx_pcie4_uscale_rp.v	对应配置器设计示例的顶层封装文件
cgator_wrapper.v	对应配置器和根端口的封装文件
cgator.v	对应配置器子块的封装文件
cgator_cpl_decoder.v	完成包解码器
cgator_pkt_generator.v	配置 TLP 生成器
cgator_tx_mux.v	发射 AXI4-Stream 多路复用逻辑

表 77：设计示例文件结构 (续)

文件	描述
cgator_gen2_enabler.v	5.0 Gb/s 导向速度变更模块
cgator_controller.v	配置器发射引擎
cgator_cfg_rom.data	配置器 ROM 文件
pio_master.v	PIO 主控制器封装文件
pio_master_controller.v	PIO 主控制器 TX 和 RX 引擎
pio_master_checker.v	检查传入的用户应用完成 TLP
pio_master_pkt_generator.v	生成用户应用 TLP

配置器设计示例层级为：

```
xilinx_pcnie_uscale_rp.v

· cgator_wrapper
    o pcie_uscale_core_top (位于 source 目录中) : 此目录包含根端口配置中核的所有源文件。
    o cgator
        - cgator_cpl_decoder
        - cgator_pkt_generator
        - cgator_tx_mux
        - cgator_gen2_enabler
        - cgator_controller: 此目录包含 <cgator_cfg_rom.data> (由 ROM_FILE 指定)。
    · pio_master
        o pio_master_controller
        o pio_master_checker
        o pio_master_pkt_generator
```

注释：cgator_cfg_rom.data 为 ROM 数据文件的默认名称。您可通过更改 ROM_FILE 参数的值来覆盖此设置。

生成核

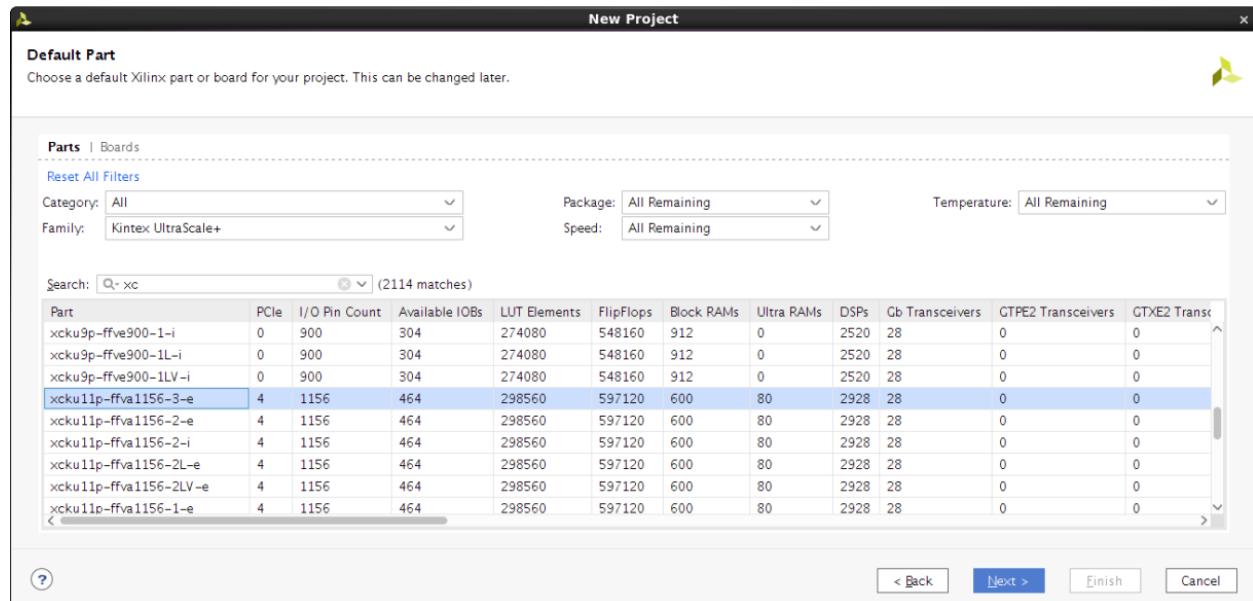
要在 Vivado® IDE 中使用默认值生成核，请遵循以下步骤进行操作：

1. 启动 Vivado IP 目录。
2. 选中“File” → “Project” → “New”（文件 > 工程 > 新建）。
3. 输入工程名称和位置，然后单击“Next”。此示例使用的是 project_name.xpr 和 project_dir。
4. 在“New Project Wizard”（新建工程向导）页面中，请勿添加源代码、现有 IP 或约束。
5. 请在下方的“Part”（器件）选项卡中选择下列筛选选项：
 - Family: Kintex® UltraScale+™

- Device: xcku11p
- Package: ffva1156
- Speed Grade: -3

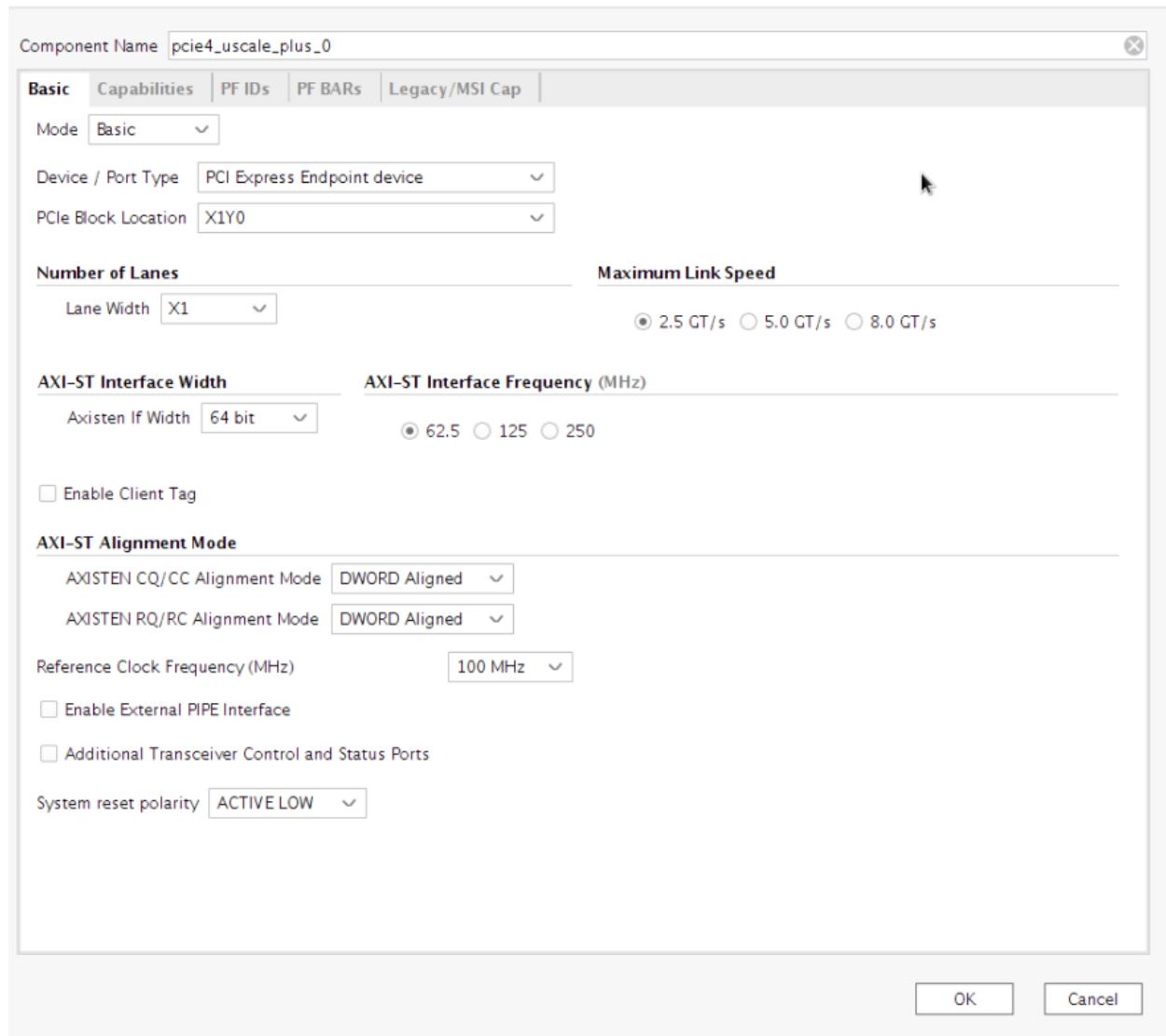
注释：如果选择了不受支持的硅片器件，那么在核列表中，该核将灰显（不可用）。

6. 从列表中选择 xcku11p-ffva1156-3-e。



7. 在最终工程汇总信息页面中，单击“OK”（确定）。
8. 在 Vivado IP 目录中，展开“Standard Bus Interfaces” → “PCI Express”（标准总线接口 > PCI Express），然后双击“UltraScale+ Devices Integrated Block for PCIe”即可显示“Customize IP”（自定义 IP）对话框。
9. 在“Component Name”（组件名称）字段中，输入核的名称。

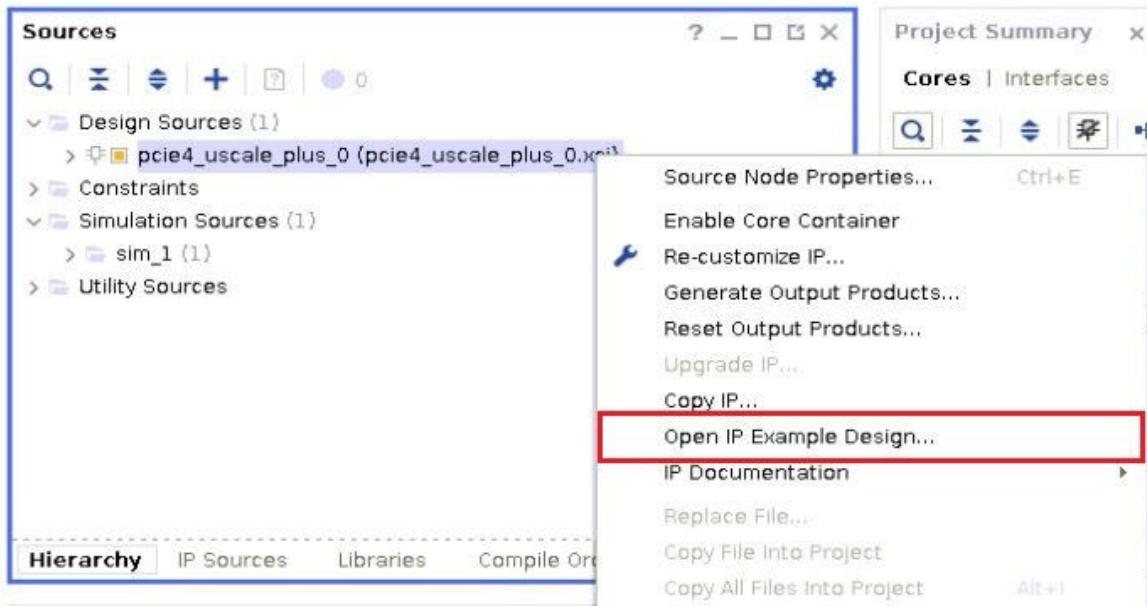
注释：在此示例中使用的名称为 <component_name>。



10. 在“Device/Port Type”（器件/端口类型）下拉菜单中，选择核的相应器件/端口类型：“Endpoint”（端点）或“Root Port”（根端口）。
11. 单击“OK”以使用默认参数生成核。

打开设计示例

1. 要打开 IP 设计示例，请右键单击生成的 IP 核，然后选择“Open IP Example Design”（打开 IP 设计示例）。



2. 在打开的窗口中，单击“OK”（确定）。Vivado 会在“Example project directory”（工程示例目录）中创建名为<core_name>.ex 的目录。
3. 按需调整路径，然后单击“OK”。

相关信息

[生成核](#)

设计示例仿真

设计示例可提供一种简便的方法，用于对使用 Vivado Design Suite 生成的 PCI Express® 端点和根端口设计示例工程的核的行为进行仿真和观察。

当前受支持的仿真器包括：

- Vivado 仿真器（默认）
- Questa Advanced Simulator
- Cadence Incisive Enterprise Simulator (IES)
- Synopsys Verilog Compiler Simulator (VCS)

您可在工程示例上生成设计示例工程并运行仿真。仿真器会使用设计示例测试激励文件以及随两种设计配置的设计示例一起提供的测试用例。

使用默认 Vivado 仿真器的仿真运行步骤如下：

1. 在“Sources”窗口中右键单击工程示例文件 (.xci) 并选中“Open IP Example Design”（打开 IP 设计示例）。这样即可创建工程示例。

- 在左侧 Flow Navigator 窗格中的“Simulation”（仿真）下，右键单击“Run Simulation”（运行仿真）并选中“Run Behavioral Simulation”（运行行为仿真）。



重要提示！ 针对 PCI Express 块不支持综合后和实现后仿真选项。

运行“Run Behavioral Simulation Option”后，即可通过“Tcl Console”以及“Log”窗口的“Simulation”选项卡中的活动来观察编译和细化阶段。

- 在“Tcl Console”（Tcl 控制台）中，输入 `run all` 命令并按“Enter”键。这样即可根据设计示例测试激励文件中提供的测试用例运行完整仿真。

完成仿真后，可在“Tcl Console”中查看结果。

端点配置

端点配置下，随 UltraScale+ 器件 Integrated Block for PCI Express 核提供的仿真环境可对 PIO 设计示例执行简单的存储器访问测试。传输事务由根端口模型生成，并由 PIO 设计示例响应。

- PCI Express 传输事务层数据包 (TLP) 由测试激励文件发射用户应用 (`pci_exp_usrapp_tx`) 生成。在发射 TLP 时，它还会生成 log 日志文件 `tx.dat`。
- PCI Express TLP 由测试激励文件接收用户应用 (`pci_exp_usrapp_rx`) 来接收。当用户应用接收 TLP 时，它会生成 log 日志文件 `rx.dat`。

如需获取有关测试激励文件的更多信息，请参阅下一章中的 [对应端点的根端口模型测试激励文件](#)。

设计示例的综合和实现

要在 Vivado Design Suite 环境中对设计示例运行综合和实现，请执行以下步骤：

- 转至 XCI 文件，右键单击并选择“Open IP Example Design”。

这样会打开新的 Vivado 工具窗口并在 project 目录中显示名为“example_project”的工程。

- 在 Flow Navigator 中，单击“Run Synthesis”（运行综合）和“Run Implementation”（运行实现）。



提示：首先单击“Run Implementation”即可同时运行综合和实现。单击“Generate Bitstream”（生成比特流）即可按顺序运行综合、实现，然后再运行比特流。

测试激励文件

对应端点的根端口模型测试激励文件

PCI Express® 根端口模型是稳健的测试激励文件环境，可提供测试程序接口，此接口可配合提供的编程输入/输出 (PIO) 设计或您的设计一起使用。根端口模型的用途是提供源机制用于生成下游 PCI™ Express TLP 流量以对客户设计进行仿真，并提供目标机制用于接收来自仿真环境内的客户设计的上游 PCI™ Express TLP 流量。

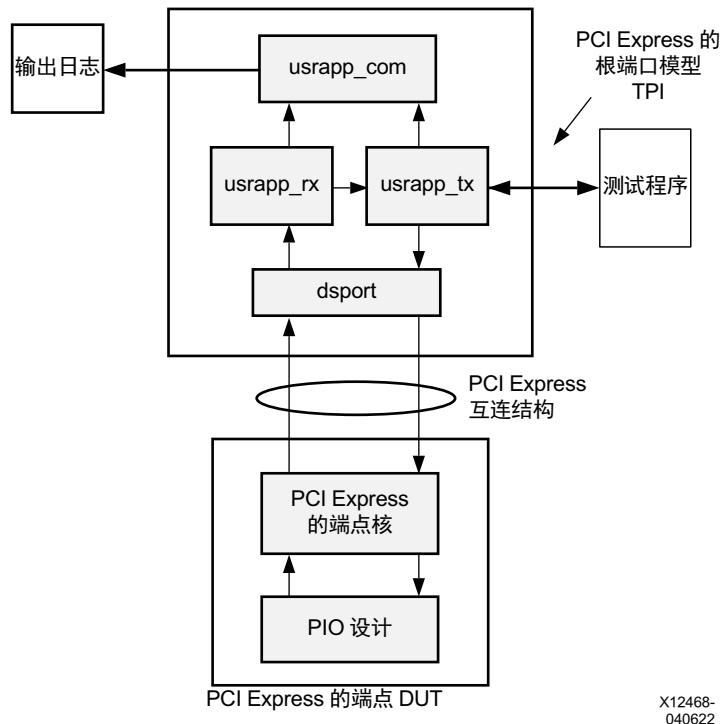
其中包含的根端口模型的源代码可提供模型，作为您的测试激励文件的起点。此外，包括初始化核配置空间、创建 TLP 传输事务、生成 TLP 日志和提供接口用于创建和验证测试在内的所有主要工作都已全部完成。这样，您即可集中精力来验证设计功能，而无需花时间来开发端点核测试激励文件基础架构。

根端口模型包括：

- 测试编程接口 (TPI)，支持您对 PCI Express 的端点器件进行仿真。
- 测试示例，用于演示如何使用测试程序的 TPI。
- Verilog 源代码，用于所有根端口模型组件，以支持您对测试激励文件进行自定义。

下图显示了配合 PIO 设计使用的根端口模型。

图 138：根端口模型和顶层端点

X12468-
040622

架构

根端口模型包括以下块：

- dsport (根端口)
- usrapp_tx
- usrapp_rx
- usrapp_com (仅限 Verilog)

usrapp_tx 和 usrapp_rx 块与 dsport 块相连，以通过端点受测设计 (DUT) 发射和接收 TLP。端点 DUT 由 PCIe 端点和 PIO 设计（如图所示）或客户设计组成。

usrapp_tx 块将 TLP 发送至 dsport 块以供通过 PCI Express 链路发送至端点 DUT。而端点 DUT 器件则通过 PCI Express 链路将 TLP 发射至 dsport，随后此块将被传递到 usrapp_rx 块。通过 PCI Express 逻辑进行通信时，dsport 与核共同负责数据链路层和物理链路层处理。usrapp_tx 和 usrapp_rx 均使用 usrapp_com 来执行共享功能，例如，TLP 处理和日志文件输出。传输事务顺序或测试程序由 usrapp_tx 块发起，以对端点器件互连结构接口进行仿真。usrapp_rx 块将会接收到来自端点器件的 TLP 响应。usrapp_tx 块与 usrapp_rx 块之间的通信使 usrapp_tx 块能够在 usrapp_rx 块接收到来自端点器件的 TLP 时，验证行为是否正确并执行相应的操作。

缩放式仿真超时

核的仿真模型在链路训练期间使用按比例缩短的时间来支持链路在仿真期间以合理的时间量来完成训练。根据《PCI Express 规范第 3.0 版》(<http://www.pcisig.com/specifications>)，存在多种超时情况与链路训练状态机 (LTSSM) 所处的状态相关联。核可在仿真期间按 256 的倍数来缩放这些超时值，但在 Recovery Speed_1 LTSSM 状态下除外，在此状态下超时值无法进行缩放。

测试选择

可用测试

下表描述了随根端口模型提供的测试。

表 78：根端口模型所提供的测试

测试名称	语言	描述
sample_smoke_test0	Verilog	发出 PCI 类型 0 配置读取 TLP 并等待完成 TLP；然后将返回的值与期望的器件/供应商 ID 值进行比对。
sample_smoke_test1	Verilog	执行与 sample_smoke_test0 相同的操作，但改为使用目标任务。此测试使用 2 个独立的测试程序线程：第 1 个线程发出 PCI 类型 0 配置读取 TLP，第 2 个线程则发出含数据的完成 TLP 目标任务。此测试演示了使用目标任务的并行测试的形式。此测试形式允许确认来自设计的任何 TLP 的接收状态。此外，如无需考虑排序，则此方法还可用于确认接收 TLP 的接收状态。

Verilog 测试选项

用于根端口模型的 Verilog 测试模型支持您指定要运行的测试的名称，此测试将作为命令行参数在仿真器上运行。

如需更改要运行的测试，请更改 TESTNAME 值，该值在测试文件 sample_tests1.v 和 pio_tests.v 中定义。此机制适用于 Mentor Graphics Advanced Simulator。Vivado 仿真器使用 -testplusarg 选项来指定 TESTNAME。例如：

```
demo_tb.exe-gui -view wave.wcfg -wdb wave_isim -tclbatch isim_cmd.tcl -testplusarg  
TESTNAME=sample_smoke_test0
```

波形转储

如需了解有关仿真器波形转储的信息，请参阅《Vivado Design Suite 用户指南：逻辑仿真》(UG900)。

Verilog 流程

根端口模型可提供相应的机制，以使用 +dump_all 命令行参数将仿真波形输出至文件。

输出日志记录

当设计示例或客户设计中出现测试失败时，测试编程人员可对违规测试用例进行调试。通常，测试编程人员会检测仿真的波形文件并将其与标准输出上显示的报文进行交叉引用。由于此方法极为耗时，根端口模型提供了输出 log 日志记录机制，以帮助测试人员对失败的测试用例进行调试，以便加速完成整个流程。

根端口模型会在每个仿真运行轮次内创建 3 个输出 log 日志文件。这些文件为 tx.dat、rx.dat 和 error.dat。rx.dat 和 tx.dat 文件都各自包含由根端口模型分别接收和发射的每个 TLP 的详细记录。



提示：了解特定测试用例期间期望的 TLP 发射行为后，即可确定故障。

error.dat 文件应与目标任务配合使用。使用目标任务的测试程序会向标准输出生成常规错误报文。在 error.dat 中可找到有关因目标错误而发生的特定测试失败比较结果的详细信息。

并行测试程序

“Root Port Model”（根端口模型）支持 2 类测试。

- 顺序测试：此类测试存在于单一进程中，其行为与顺序程序相似。测试程序：pio_writeReadBack_test0（详见本章后文）中所示测试即为顺序测试的示例。顺序测试适用于验证事件行为是否符合已知顺序。
- 并行测试：涉及多个进程线程的测试。sample_smoke_test1 测试即为含 2 个进程线程的并行测试示例。并行测试适用于验证一组特定事件是否已发生，但这些事件的顺序未知。

典型的并行测试采用 1 个命令线程加 1 个或多个预期线程的格式。这些线程协同工作以验证器件功能。命令线程的角色是创建必要的 TLP 传输事务，这些传输事务会导致器件接收并生成 TLP。预期线程的角色是验证是否接收到预期的 TLP。Root Port Model TPI 具有一整套预期任务，可配合并行测试来使用。

由于设计示例是以目标为唯一导向的器件，并行测试程序使用 PIO 设计时只能预期收到完成 TLP。但预期任务的整个库在配合客户设计（可包含总线主控功能）使用时即可用于预期接收任何类型的 TLP。

完成器模型

在完成核配置后，通过在 Vivado Tcl 控制台中执行以下命令来启用完成器模型：

```
set_property -dict [list CONFIG.completer_model {true}] [get_ips <PCIe IP Core Name>]
```

当为核配置 512 位 AXI 接口后，您可选择使用此完成器模型测试激励文件，此文件可配合您的设计用于实践总线总控功能（从端点 DUT 到根端口模型的上游方向流量）。

完成器模型可提供根端口侧存储器阵列 (DATA_STORE_2)，此阵列可通过存储器写入传输事务来写入，并可通过从端点 DUT 执行存储器读取传输事务来读取。此存储器可通过根端口模型模块 (*xilinx_pcie_uscale_rp.v*) 顶层提供的 2 个不同参数来进行配置。

- RP_BAR[63:0]：提供 DATA_STORE_2 阵列的第一个字节的地址。
- RP_BAR_SIZE[5:0]：提供 DATA_STORE_2 阵列的字节地址位 -1 的数值。例如，值为 11 可提供 $2^{(11+1)}$ 字节或 4 KB 可用存储器。

将根据根端口模型中设置的上述 2 个参数、字节使能、4K 边界、最大有效载荷大小和最大读取请求大小规则来检查每个存储器传输事务与对应的存储器阵列位置。返回的每个存储器读取完成包都将根据最大有效载荷大小和读取完成边界规则来进行拆分。完成器模型还支持长度为 0 的写入数据包（拦截数据包但不会存储其有效载荷数据）和长度为 0 的读取数据包（返回含 1 个 DW 的有效载荷数据）。

测试描述

根端口模型可提供测试程序接口 (TPI)。TPI 提供了通过调用一系列 Verilog 任务来创建测试的方法。所有根端口模型测试都应遵循以下 6 步：

1. 对唯一测试名称执行条件比较。
2. 设置主控制器超时，以防止仿真挂起。
3. 等待复位和链路建立完成。
4. 初始化端点的配置空间。
5. 在根端口模型与端点 DUT 之间执行 TLP 的发射和接收。
6. 验证测试是否成功。

测试程序：pio_writeReadBack_test0

```
1.     else if(testname == "pio_writeReadBack_test1"
2.     begin
3.         // This test performs a 32 bit write to a 32 bit Memory space and
4.         // performs a read back
5.         TSK_SIMULATION_TIMEOUT(10050);
6.         TSK_SYSTEM_INITIALIZATION;
7.         TSK_BAR_INIT;
8.         for (ii = 0; ii <= 6; ii = ii + 1) begin
9.             if (BAR_INIT_P_BAR_ENABLED[ii] > 2'b00) // bar is enabled
10.                 case(BAR_INIT_P_BAR_ENABLED[ii])
11.                     2'b01 : // IO SPACE
12.                     begin
13.                         $display("[%t] : NOTHING: to IO 32 Space BAR %x", $realtime, ii);
14.                     end
15.                     2'b10 : // MEM 32 SPACE
16.                     begin
17.                         $display("[%t] : Transmitting TLPs to Memory 32 Space BAR %x",
18.                         $realtime, ii);
19.                     //-----
20.                     //-----
21.                     DATA_STORE[0] = 8'h04;
22.                     DATA_STORE[1] = 8'h03;
23.                     DATA_STORE[2] = 8'h02;
24.                     DATA_STORE[3] = 8'h01;
25.                     P_READ_DATA = 32'hffff_ffff; // make sure P_READ_DATA has known
initial value
26.                     TSK_TX_MEMORY_WRITE_32(DEFAULT_TAG, DEFAULT_TC, 10'd1,
BAR_INIT_P_BAR[ii][31:0], 4'hF, 4'hF, 1'b0);
27.                     TSK_TX_CLK_EAT(10);
28.                     DEFAULT_TAG = DEFAULT_TAG + 1;
29.                     //-----
30.                     // Event : Memory Read 32 bit TLP
31.                     //-----
32.                     TSK_TX_MEMORY_READ_32(DEFAULT_TAG, DEFAULT_TC, 10'd1,
BAR_INIT_P_BAR[ii][31:0], 4'hF, 4'hF);
33.                     TSK_WAIT_FOR_READ_DATA;
34.                     if (P_READ_DATA != {DATA_STORE[3], DATA_STORE[2], DATA_STORE[1],
DATA_STORE[0] })
35.                     begin
36.                         $display("[%t] : Test FAILED --- Data Error Mismatch, Write Data %x !
= Read Data %x",
$realtime,{DATA_STORE[3], DATA_STORE[2], DATA_STORE[1], DATA_STORE[0]}, P_READ_DATA);
37.                     end
38.                     else
39.                     begin
40.                         $display("[%t] : Test PASSED --- Write Data: %x successfully
received", $realtime, P_READ_DATA);
41.                     end
```

扩展根端口模型

创建的根端口模型旨在配合 PIO 设计一起使用，因此可根据 PIO 设计的限制通过相应调整来执行特定检查并发出警告。默认情况下，当 Vivado IP 目录生成根端口模型时，这些检查和警告处于启用状态。但可将这些限制禁用，以避免对客户设计造成影响。

由于创建的 PIO 设计旨在支持最多 1 个 I/O BAR、1 个 Mem64 BAR 和 2 个 Mem32 BAR（其中之一必须为 EROM 空间），因此默认情况下，根端口模型会在器件配置期间执行检查，以验证核是否已正确配置以满足此要求。如果此检查发现违例，则会导致显示警告报文，并导致在测试激励文件中禁用违例的 BAR。可通过在 `pci_exp_usrapp_tx.v` 文件中将 `pio_check_design` 变量设置为 0 来禁用此检查。

根端口模型 TPI 任务列表

根端口模型 TPI 任务包含以下任务。

测试设置任务

表 79：测试设置任务

名称	输入		描述
TSK_SYSTEM_INITIALIZATION	无		等待事务传输接口复位以及根端口模型与端点 DUT 之间的链路建立完成。 此任务必须在端点核初始化之前调用。
TSK_USR_DATA_SETUP_SEQ	无		初始化全局 4096 字节 DATA_STORE 阵列和大小可调的 DATA_STORE_2 阵列条目，将其设置为范围在 0 到 4095 之间的顺序值。
TSK_TX_CLK_EAT	时钟计数	31:30	等待 clock_count 传输事务接口时钟。
TSK_SIMULATION_TIMEOUT	超时	31:0	设置主仿真超时值，以传输事务接口时钟数为单位。此任务应用于确保所有 DUT 测试全部完成。

TLP 任务

表 80：TLP 任务

名称	输入			描述
TSK_TX_TYPE0_CONFIGURATION_READ	tag_ reg_addr_ first_dw_be_	7:0 11:0 3:0		将“Type 0 PCI Express Config Read TLP”（类型 0 PCI Express 配置读取 TLP）从“Root Port Model”（根端口模型）发送到“Endpoint DUT”（端点 DUT）的 reg_addr（含 tag_ 和 first_dw_be_ 输入）。 从 Endpoint DUT 返回的 Cpld 使用全局 EP_BUS_DEV_FNS 的内容作为完成器 ID。

表 80：TLP 任务 (续)

名称	输入			描述
TSK_TX_TYPE1_CONFIGURATION_READ	tag_ reg_addr_ first_dw_be_	7:0 11:0 3:0		将“Type 1 PCI Express Config Read TLP”(类型 1 PCI Express 配置读取 TLP)从“Root Port Model”发送到“Endpoint DUT”的 reg_addr_(含 tag_ 输入和 first_dw_be_ 输入)。从 Endpoint DUT 返回的 CplID 使用全局 EP_BUS_DEV_FNS 的内容作为完成器 ID。
TSK_TX_TYPE0_CONFIGURATION_WRITE	tag_ reg_addr_ reg_data_ first_dw_be_	7:0 11:0 31:0 3:0		将“Type 0 PCI Express Config Write TLP”(类型 0 PCI Express 配置写入 TLP)从“Root Port Model”发送到“Endpoint DUT”的 reg_addr_(含 tag_ 输入和 first_dw_be_ 输入)。从 Endpoint DUT 返回的 Cpl 使用全局 EP_BUS_DEV_FNS 的内容作为完成器 ID。
TSK_TX_TYPE1_CONFIGURATION_WRITE	tag_ reg_addr_ reg_data_ first_dw_be_	7:0 11:0 31:0 3:0		将“Type 1 PCI Express Config Write TLP”(类型 1 PCI Express 配置写入 TLP)从“Root Port Model”发送到“Endpoint DUT”的 reg_addr_(含 tag_ 输入和 first_dw_be_ 输入)。从 Endpoint DUT 返回的 Cpl 使用全局 EP_BUS_DEV_FNS 的内容作为完成器 ID。
TSK_TX_MEMORY_READ_32	tag_ tc_ len_ addr_ last_dw_be_ first_dw_be_	7:0 2:0 10:0 31:0 3:0 3:0		将“PCI Express Memory Read TLP”(PCI Express 存储器读取 TLP)从“Root Port”(根端口)发送到“Endpoint DUT”的 32 位存储器地址 addr_。此请求使用全局 RP_BUS_DEV_FNS 的内容作为请求器 ID。
TSK_TX_MEMORY_READ_64	tag_ tc_ len_ addr_ last_dw_be_ first_dw_be_	7:0 2:0 10:0 63:0 3:0 3:0		将“PCI Express Memory Read TLP”从 Root Port Model 发送到 Endpoint DUT 的 64 位存储器地址 addr_。此请求使用全局 RP_BUS_DEV_FNS 的内容作为请求器 ID。
TSK_TX_MEMORY_WRITE_32	tag_ tc_ len_ addr_ last_dw_be_ first_dw_be_ ep_	7:0 2:0 10:0 31:0 3:0 3:0 -		将“PCI Express Memory Write TLP”(PCI Express 存储器写入 TLP)从“Root Port Model”发送模型到“Endpoint DUT”的 32 位存储器地址 addr_。此请求使用全局 RP_BUS_DEV_FNS 的内容作为请求器 ID。使用全局 DATA_STORE 字节阵列将写入数据传递到任务。
TSK_TX_MEMORY_WRITE_64	tag_ tc_ len_ addr_ last_dw_be_ first_dw_be_ ep_	7:0 2:0 10:0 63:0 3:0 3:0 -		将“PCI Express Memory Write TLP”(PCI Express 存储器写入 TLP)从“Root Port Model”发送模型到“Endpoint DUT”的 64 位存储器地址 addr_。此请求使用全局 RP_BUS_DEV_FNS 的内容作为请求器 ID。使用全局 DATA_STORE 字节阵列将写入数据传递到任务。

表 80：TLP 任务 (续)

名称	输入		描述
TSK_TX_COMPLETION	req_id_ tag_ tc_ len_ byte_count_ lower_addr_ comp_status_ ep_	15:0 7:0 2:0 10:0 2:0 11:0 6:0 -	将“PCI Express Completion TLP”(PCI Express 完成 TLP)从“Root Port Model”发送到“Endpoint DUT”，使用全局 RP_BUS_DEV_FNS 作为完成器 ID 并使用 req_id_ 输入作为请求器 ID。 comp_status_ 输入可设置为以下值之一： 3'b000 = 成功完成 3'b001 = 请求不受支持 3'b010 = 配置请求重试状态 3'b100 = 完成器异常中止
TSK_TX_COMPLETION_DATA	req_id_ tag_ tc_ len_ byte_count_ lower_addr_ ram_ptr comp_status_ ep_	15:0 7:0 2:0 10:0 11:0 6:0 RP_BAR_SIZE:0 2:0 -	将“PCI Express Completion with Data TLP”(PCI Express 含数据完成 TLP)从“Root Port Model”发送到“Endpoint DUT”，使用全局 RP_BUS_DEV_FNS 作为完成器 ID 并使用 req_id_ 输入作为请求器 ID。 使用全局 DATA_STORE_2 字节阵列来将完成数据传递到任务，使用 ram_ptr 输入偏移此阵列中的起始字节。
TSK_TX_MESSAGE	tag_ tc_ len_ data_ message_rtg_ message_code_	7:0 2:0 10:0 63:0 2:0 7:0	将“PCI Express Message TLP”(PCI Express 报文 TLP)从“Root Port Model”发送到“Endpoint DUT”。 此请求使用全局 RP_BUS_DEV_FNS 的内容作为请求器 ID。
TSK_TX_MESSAGE_DATA	tag_ tc_ len_ data_ message_rtg_ message_code_	7:0 2:0 10:0 63:0 2:0 7:0	将“PCI Express Message with Data TLP”(PCI Express 报文含数据 TLP)从“Root Port Model”发送到“Endpoint DUT”。 使用全局 DATA_STORE 字节阵列将报文数据传递到任务。 此请求使用全局 RP_BUS_DEV_FNS 的内容作为请求器 ID。
TSK_TX_IO_READ	tag_ addr_ first_dw_be_	7:0 31:0 3:0	将“PCI Express I/O Read TLP”(PCI Express I/O 读取 TLP)从“Root Port Model”发送到“Endpoint DUT”的 I/O 地址 addr_[31:2]。 此请求使用全局 RP_BUS_DEV_FNS 的内容作为请求器 ID。
TSK_TX_IO_WRITE	tag_ addr_ first_dw_be_ data	7:0 31:0 3:0 31:0	将“PCI Express I/O Write TLP”(PCI Express I/O 写入 TLP)从“Root Port Model”发送到“Endpoint DUT”的 I/O 地址 addr_[31:2]。 此请求使用全局 RP_BUS_DEV_FNS 的内容作为请求器 ID。

表 80：TLP 任务 (续)

名称	输入			描述
TSK_TX_BAR_READ	bar_index byte_offset tag_ tc_	2:0 31:0 7:0 2:0		将 PCI Express one Dword Memory 32、Memory 64 或 I/O Read TLP 从 Root Port Model 发送到目标地址，此地址对应于来自 Endpoint DUT 的 BAR bar_index 的偏移 byte_offset。此任务会基于初始化期间 BAR bar_index 的配置方式来发送相应的 Read TLP。只有在成功完成 TSK_BAR_INIT 后才能调用此任务。 此请求使用全局 RP_BUS_DEV_FNS 的内容作为请求器 ID。
TSK_TX_BAR_WRITE	bar_index byte_offset tag_ tc_ data_	2:0 31:0 7:0 2:0 31:0		将 PCI Express one Dword Memory 32、Memory 64 或 I/O Write TLP 从 Root Port 发送到目标地址，此地址对应于来自 Endpoint DUT 的 BAR bar_index 的偏移 byte_offset。 此任务会基于初始化期间 BAR bar_index 的配置方式来发送相应的 Write TLP。只有在成功完成 TSK_BAR_INIT 后才能调用此任务。
TSK_WAIT_FOR_READ_DATA	无			等待端口 DUT 发送的下一个含数据完成 TLP。成功完成后，来自 CplD 的数据的第一个 Dword 将存储在全局 P_READ_DATA 中。在 TPI 中请求含数据完成 TLP 的任何读取任务完成后应立即调用此任务，以避免出现任何争用状况。 默认情况下，此任务会局部超时，并在经过 1000 个传输事务接口时钟后终止仿真。全局 cpld_to_finish 可设置为 0，以便局部超时向调用测试返回执行指令，并避免导致仿真超时。在本例中，测试程序应检查全局 cpld_to，后者设置为 1，表明此任务已超时并且 P_READ_DATA 的内容无效。
TSK_TX_SYNCHRONIZE	first_ active_ last_call_ tready_sw_	- - - -		等待 AXI4-Stream Requester Request (请求器请求) 信号或 Completer Completion Interface Ready (完成器完成接口就绪) 信号断言有效，并将 log 日志文件中的输出同步到当前有效的每个传输事务。 first_ 输入表示包起始。 active_ 输入表示当前正在执行传输事务。 last_call_ 输入表示包结束。 tready_sw_ 输入用于选择 Requester Request 信号或 Completer Completion Interface Ready 信号。
TSK_BUILD_RC_TO_PCIE_PKT	rc_data_QW0 rc_data_QW1 m_axis_rc_tkeep m_axis_rc_tlast	63:0 63:0 KEEP_WIDTH-1:0 -		将位于请求器完成接口的 AXI4-Stream 包从描述符包格式转换为 PCIe TLP 包格式，以便于记录到 log 日志中。
TSK_BUILD_CQ_TO_PCIE_PKT	cq_data cq_be m_axis_cq_tdata	63:0 7:0 63:0		将位于完成器请求接口的 AXI4-Stream 包从描述符包格式转换为 PCIe TLP 包格式，以便于记录到 log 日志中。

表 80：TLP 任务 (续)

名称	输入		描述
TSK_BUILD_CPLD_PKT	cq_addr cq_be m_axis_cq_tdata	63:0 15:0 63:0	针对从端口 DUT 接收到的存储器读取返回完成包或含数据完成包。使用完成器模型时，生成的完成包将根据“Root Port Model”中的“Max Payload Size”（最大有效载荷大小）和“Read Completion Boundary”（读取完成边界）规则进行拆分。含数据完成包使用全局 DATA_STORE_2 阵列中存储的数据。

BAR 初始化任务

表 81：BAR 初始化任务

名称	输入	描述
TSK_BAR_INIT	无	使用 PCI Express 互连结构按标准顺序执行以端点器件为目标的基址寄存器初始化任务。根据端点 PCI BAR 范围要求执行扫描、执行必要的存储器和 I/O 空间映射计算，最后对端点进行编程，使其做好准备以供访问。 完成后，用户测试程序即可开始执行以器件为目标的存储器和 I/O 传输事务。该功能标准输出可显示为存储器和 I/O 表，其中详列端点的初始化方式。此任务还可在根端口模型中初始化可供测试程序使用的全局变量。仅限在 TSK_SYSTEM_INITIALIZATION 之后调用此任务。
TSK_BAR_SCAN	无	使用 PCI Express 逻辑按顺序执行一系列 PCI 类型 0 配置写入和配置读取，以判定端点的存储器和 I/O 要求。 该任务会将此信息存储在全局阵列 BAR_INIT_P_BAR_RANGE[] 中。仅限在 TSK_SYSTEM_INITIALIZATION 之后调用此任务。
TSK_BUILD_PCIE_MAP	无	执行存储器和 I/O 映射算法，并基于端点要求来分配存储器 32、存储器 64 和 I/O 空间。 此任务已自定义为按 PIO 设计限制来执行，仅限在 TSK_BAR_SCAN 完成之后才能调用。
TSK_DISPLAY_PCIE_MAP	无	显示端点核 PCI 基址寄存器的存储器映射信息。对于每个 BAR，BAR 值、BAR 范围和 BAR 类型均已给定。仅限在完成 TSK_BUILD_PCIE_MAP 之后调用此任务。

PIO 设计任务示例

表 82：PIO 设计任务示例

名称	输入		描述
TSK_TX_READBACK_CONFIG	无		使用 PCI Express 逻辑执行一系列 PCI 类型 0 配置读取，读取目标为端点器件基址寄存器、PCI 命令寄存器和 PCIe 器件控制寄存器。 仅限在 TSK_SYSTEM_INITIALIZATION 之后调用此任务。
TSK_MEM_TEST_DATA_BUS	bar_index	2:0	通过对 bar_index 输入指向的 I/O 或存储器地址执行 32 位遍历 1 (walking 1s) 数据测试，来测试 PIO 设计 FPGA 块 RAM 数据总线接口是否已正确连接。 如需进行详尽测试，应对该任务进行 4 次调用，针对 PIO 设计中所使用的每个块 RAM 调用 1 次。

表 82：PIO 设计任务示例 (续)

名称	输入		描述
TSK_MEM_TEST_ADDR_BUS	bar_index nBytes	2:0 31:0	通过执行遍历 1 (walking 1s) 地址测试（从 bar_index 输入指向的 I/O 或存储器地址开始），来测试 PIO 设计 FPGA 块 RAM 地址总线接口是否已准确连接。 如需进行详尽测试，应对该任务进行 4 次调用，针对 PIO 设计中所使用的每个块 RAM 调用 1 次。此外，nBytes 输入应指定各块 RAM 的完整大小。
TSK_MEM_TEST_DEVICE	bar_index nBytes	2:0 31:0	通过对所有位（从 bar_index 输入指向的块 RAM 开始，范围由 nBytes 输入指定）执行递增/递减测试，来测试 PIO 设计 FPGA 块 RAM 的每个位的完整性。 如需进行详尽测试，应对该任务进行 4 次调用，针对 PIO 设计中所使用的每个块 RAM 调用 1 次。此外，nBytes 输入应指定各块 RAM 的完整大小。
TSK_RESET	复位	0	启动 board.v 文件中的 sys_rst_n 信号。 强制发射 sys_rst_n 信号以断言复位有效。使用 TSK_RESET (1'b1) 来断言复位有效，并使用 TSK_RESET (1'b0) 来释放复位信号。
TSK_MALFORMED	malformed_bits	7:0	用于创建格式错误的 TLP 的控制位： 0001：为执行此任务后立即调用的 I/O 请求和配置请求生成格式错误的 TLP 0010：为根端口处接收到的存储器读取请求生成格式错误的完成 TLP

目标任务

表 83：目标任务

名称	输入		输出	描述
TSK_EXPECT_CPLD	traffic_class td ep attr length completer_id completer_status bcm byte_count requester_id tag address_low	2:0 - - 1:0 10:0 15:0 2:0 - 11:0 15:0 7:0 6:0	目标状态	等待含数据完成 TLP，其中含匹配的 traffic_class、td、ep、attr、length 和 payload。 成功完成时返回 1；否则返回 0。

表 83：目标任务 (续)

名称	输入	输出	描述
TSK_EXPECT_CPL	traffic_class td ep attr completer_id completer_status bcm byte_count requester_id tag address_low	2:0 - - 1:0 15:0 2:0 - 11:0 15:0 7:0 6:0	目标状态 等待无数据完成 TLP，其中含匹配的 traffic_class、td、ep、attr 和 length。 成功完成时返回 1；否则返回 0。
TSK_EXPECT_MEMRD	traffic_class td ep attr length requester_id tag last_dw_be first_dw_be address	2:0 - - 1:0 10:0 15:0 7:0 3:0 3:0 29:0	目标状态 等待含匹配的报头字段的 32 位地址存储器读取 TLP。 成功完成时返回 1；否则返回 0。此任务只能配合总线主控设计一起使用。
TSK_EXPECT_MEMRD64	traffic_class td ep attr length requester_id tag last_dw_be first_dw_be address	2:0 - - 1:0 10:0 15:0 7:0 3:0 3:0 61:0	目标状态 等待含匹配的报头字段的 64 位地址存储器读取 TLP。成功完成时返回 1；否则返回 0。 此任务只能配合总线主控设计一起使用。
TSK_EXPECT_MEMWR	traffic_class td ep attr length requester_id tag last_dw_be first_dw_be address	2:0 - - 1:0 10:0 15:0 7:0 3:0 3:0 29:0	目标状态 等待含匹配的报头字段的 32 位地址存储器写入 TLP。成功完成时返回 1；否则返回 0。 此任务只能配合总线主控设计一起使用。

表 83：目标任务 (续)

名称	输入	输出	描述
TSK_EXPECT_MEMWR64	traffic_class td ep attr length requester_id tag last_dw_be first_dw_be address	2:0 - - 1:0 10:0 15:0 7:0 3:0 3:0 61:0	目标状态 等待含匹配的报头字段的 64 位地址存储器写入 TLP。成功完成时返回 1；否则返回 0。 此任务只能配合总线主控设计一起使用。
TSK_EXPECT_IOWR	td ep requester_id tag first_dw_be address data	- - 15:0 7:0 3:0 31:0 31:0	目标状态 等待含匹配的报头字段的 I/O 写入 TLP。成功完成时返回 1；否则返回 0。 此任务只能配合总线主控设计一起使用。

对应根端口的端点模型测试激励文件

对应根端口配置下的核的端点模型测试激励文件是一个简单的测试激励文件示例，通过将配置器设计示例与 PCI Express 端点模型相连来允许这两者在同一物理系统内以 2 个器件的方式来运行。由于配置器设计示例包含通过自我初始化并生成和耗用总线流量的逻辑，因此测试激励文件示例仅通过实现此逻辑来监控系统运行和终止仿真。

端点模型测试激励文件包括：

- Verilog 或 VHDL 源代码（适用于所有端点模型组件）。
- PIO 从接口设计。

本章节中的前图显示了配合配置器设计示例使用的端点模型。

架构

端点模型包含下列块：

- PCI Express 端点（端点配置中的核）模型。
- PIO 从接口设计，包括：
 - PIO_RX_ENGINE
 - PIO_TX_ENGINE
 - PIO_EP_MEM
 - PIO_TO_CTRL

PIO_RX_ENGINE 和 PIO_TX_ENGINE 块与端点块相连，以便对往来根端口受测设计 (DUT) 的 TLP 执行接收与发射。根端口 DUT 由配置为根端口的核与配置器设计示例（包括配置器块和 PIO 主控制器设计或客户设计）组成。

如需了解有关 PIO 从接口设计的更多详细信息，请参阅 [编程输入/输出：端点设计示例](#)。

设计仿真

随模型提供的 simulate_mti.do 仿真脚本文件可用于简化使用 Mentor Graphics Advanced Simulator 执行的仿真。

仿真示例脚本文件位于以下目录：

```
<project_dir>/<component_name>/simulation/functional
```

在“设计流程步骤”章节的“仿真”部分中提供了有关采用端点模型对配置器设计示例执行仿真的指示信息。

注释：对于 Cadence IES 用户，必须将此工作构造手动插入 cds.lib 文件：

```
DEFINE WORK WORK
```

缩放式仿真超时

核的仿真模型在链路训练期间使用按比例缩短的时间来支持链路在仿真期间以合理的时间量来完成训练。根据《PCI Express 规范第 3.0 版》(<http://www.pcisig.com/specifications>)，存在多种超时情况与链路训练状态机 (LTSSM) 所处的状态相关联。核可在仿真期间按 256 的倍数来缩放这些超时值，但在 Recovery Speed_1 LTSSM 状态下除外，在此状态下超时值无法进行缩放。

波形转储

如需了解有关仿真器波形转储的信息，请参阅《Vivado Design Suite 用户指南：逻辑仿真》([UG900](#))。

输出日志记录

测试激励文件输出的报文（在仿真日志内捕获）用于指示发生下列操作的时间：

- user_reset 断言无效
- user_lnk_up 断言有效
- 配置器断言 cfg_done 有效
- PIO 主控制器断言 pio_test_finished 有效
- 仿真超时（前提是不断言 pio_test_finished 或 pio_test_failed 有效）

升级

本附录包含有关升级至更新版本的 IP 核的信息。

从 UltraScale 移植到 UltraScale+ 器件

本章节为用户提供了从 UltraScale+™ 器件 Integrated Block for PCIe 核移植到 UltraScale™ 器件 Integrated Block for PCIe 核的相关信息。

新增端口

下表列出了从 UltraScale+ 器件核移植到 UltraScale 器件核中涉及的新端口。

表 84：UltraScale+ 器件核中的新端口

名称	I/O	注释
pcie_rq_seq_num0[5:0]	输出	UltraScale 中的 pcie_rq_seq_num
pcie_rq_seq_num_vld0	输出	UltraScale 中的 pcie_rq_seq_num_vld
pcie_rq_tag0[7:0]	输出	UltraScale 中的 pcie_rq_tag
pcie_rq_tag_vld0	输出	UltraScale 中的 pcie_rq_tag_vld
pcie_rq_seq_num1[5:0]	输出	
pcie_rq_seq_num_vld1	输出	
pcie_rq_tag1[7:0]	输出	
pcie_rq_tag_vld1	输出	
cfg_mgmt_function_number[7:0]	输入	
cfg_mgmt_debug_access	输入	
cfg_local_error_valid	输入	
cfg_local_error_out[4:0]	输入	
cfg_rx_pm_state[1:0]	输出	
cfg_tx_pm_state[1:0]	输出	
cfg_bus_number[7:0]	输出	
cfg_dev_id_pf0[15:0]	输入	用户可通过 I/O 访问 ID
cfg_dev_id_pf1[15:0]	输入	
cfg_dev_id_pf2[15:0]	输入	
cfg_dev_id_pf3[15:0]	输入	
cfg_vend_id[15:0]	输入	
cfg_rev_id_pf0[7:0]	输入	

表 84: UltraScale+ 器件核中的新端口 (续)

名称	I/O	注释
cfg_rev_id_pf1[7:0]	输入	
cfg_rev_id_pf2[7:0]	输入	
cfg_rev_id_pf3[7:0]	输入	
cfg_subsys_id_pf0[15:0]	输入	
cfg_subsys_id_pf1[15:0]	输入	
cfg_subsys_id_pf2[15:0]	输入	
cfg_subsys_id_pf3[15:0]	输入	
cfg_vf_flr_func_num[7:0]	输入	
cfg_interrupt_msi_pending_status_function_num[1:0]	输入	
cfg_interrupt_msi_select[1:0]	输入	
cfg_pm_aspm_l1_entry_reject	输入	
cfg_pm_aspm_tx_l0s_entry_disable	输入	
cfg_interrupt_msix_vec_pending[1:0]	输入	
cfg_interrupt_msix_vec_pending_status	输出	
pl_redo_eq	输入	
pl_redo_eq_speed	输入	
pl_eq_mismatch	输出	
pl_redo_eq_pending	输出	

端口宽度更改

下表列出了 UltraScale 器件核与 UltraScale+ 器件核之间宽度发生更改的端口。

表 85: 端口宽度更改

名称	I/O
pcie_rq_tag_av[3:0]	输出
pcie_tfc_nph_av[3:0]	输出
pcie_tfc_npd_av[3:0]	输出
pcie_cq_np_req[1:0]	输入
pcie_cq_np_req_count[5:0]	输出
cfg_mgmt_addr[9:0]	输入
cfg_negotiated_width[2:0]	输出
cfg_current_speed[1:0]	输出
cfg_max_payload[1:0]	输出
cfg_vf_status[503:0]	输出
cfg_vf_power_state[755:0]	输出
cfg_vf_tph_requester_enable[251:0]	输出
cfg_vf_tph_st_mode[755:0]	输出
cfg_vf_flr_in_process[251:0]	输出

表 85: 端口宽度更改 (续)

名称	I/O
cfg_vf_flr_runc_num[7:0]	输入
cfg_interrupt_msix_vf_enable[251:0]	输出
cfg_interrupt_msix_vf_mask[251:0]	输出
cfg_interrupt_msi_tph_st_tag[7:0]	输入
cfg_interrupt_msi_function_number[7:0]	输入

已弃用的端口

下表列出了 UltraScale+ 器件核中已弃用的与 UltraScale 器件核相关的端口。

表 86: UltraScale+ 器件核中不可用的端口

名称	I/O
cfg_mgmt_type1_cfg_reg_access	输入
cfg_local_error	输出
cfg_ltr_enable	输出
cfg_dpa_substate_change[3:0]	输出
cfg_per_func_status_control[2:0]	输入
cfg_per_func_status_data[15:0]	输出
cfg_per_function_number[3:0]	输入
cfg_per_function_output_request	输入
cfg_per_function_update_done	输出
cfg_ds_function_number[2:0]	输入
cfg_interrupt_msi_vf_enable[7:0]	输出
cfg_interrupt_msix_sent	输出
cfg_interrupt_msix_fail	输出
user_tph_stt_address[4:0]	输入
user_tph_function_num[3:0]	输入
user_tph_stt_read_data[31:0]	输出
user_tph_stt_read_data_valid	输出
user_tph_stt_read_enable	输入
pl_eq_reset_eieos_count	输入

专用 PERST 布线

在 UltraScale+ 核中不使用专用复位布线。

在 Vivado Design Suite 中执行升级

本节提供了在 Vivado Design Suite 中升级到此 IP 核的更新版本时，有关对用户逻辑或端口指定所作的任何更改的信息。

参数更改

下表显示了当前版本的核中所发生的参数更改。

表 87：新参数

用户参数名称	显示名称	新增/更改/移除	详细信息	默认值
axisten_if_enable_rx_msg_intfc	Enable RX Message INTFC	新增	选中此项时，报文将被路由至位于“Receive Message Interface”（接收报文接口）的 cfg_msg_received 信号。否则，报文将路由至 CQ Interface。	False (不选中)
enable_auto_rxeq	“Enable Auto RxEq” (启用自动接收器均衡)	新增	在 Auto Mode 中选择 Receiver Equalization	False (不选中)
mcap_fpga_bitstream_version	MCAP Bitstream Version register value	新增	指定 MCAP 寄存器空间中的 MCAP 比特流版本寄存器的值。	00000000

端口更改

选中“Shared logic option GT common in core”并且 PLL 类型未设为 CPLL 时，就会显示下表中的端口。

表 88：新增端口

名称	I/O	宽度
ext_qpllrxcalenb	输出	1 位

选中“Shared logic option GT-Wizard in core”时，就会显示下表中的端口。

表 89：新增端口

名称	I/O	宽度
ext_phy_clk_bufg_gt_ce	输出	1 位
ext_phy_clk_bufg_gt_reset	输出	1 位
ext_phy_clk_RST_IDLE	输出	1 位
ext_phy_clk_txoutclk	输出	1 位
ext_phy_clk_bufgtcemask	输出	1 位
ext_phy_clk_gt_bufgrstmask	输出	1 位
ext_phy_clk_bufgtdiv	输出	8 位
ext_phy_clk_pclk2_gt	输入	1 位
ext_phy_clk_int_clock	输入	1 位
ext_phy_clk_pclk	输入	1 位
ext_phy_clk_phy_pclk2	输入	1 位

表 89: 新增端口 (续)

名称	I/O	宽度
ext_phy_clk_phy_coreclk	输入	1 位
ext_phy_clk_phy_userclk	输入	1 位
ext_phy_clk_phy_mcapclk	输入	1 位
ext_qpllxxrcalenb	输出	1 位

管理传入完成包的接收缓冲器空间

《PCI™ Express 基本规范》要求所有端点将接收到的完成包的无限流量控制信用值播发至其链路合作伙伴。这意味着端点发射非转发请求的必要前提是，它具有相应空间可用于为这些非转发请求接受完成响应。本附录描述了用户应用如何在 UltraScale+™ 器件 Gen3 Integrated Block for PCIe 核中管理接收缓冲器空间以满足此项要求。

常见注意事项和概念

完成空间

下表定义了该核在接收缓冲器内保留的完成空间。值因该核的不同“Capability Max Payload Size”（功能最大有效载荷大小）设置以及您所选的性能级别而异。值为以十进制表示的信用值。

表 90：接收器缓冲器完成空间

功能最大有效载荷大小（字节）	性能级别：极限	
	CPH	CPD
128	128	32,768B
256	128	32,768B
512	128	32,768B
1024	128	32,768B

最大请求大小

“Memory Read”（存储器读取）中请求的数量不得超过 Max_Request_Size 中声明的值，该值是由“Configuration”（配置）位 cfg_max_read_req[2:0] 按下表中的定义给定的。如果用户应用不读取 Max_Request_Size 值，那么它必须使用默认值，即 128 字节。

表 91：Max_Request_Size 设置

cfg_max_read_req[2:0] 】	Max_Request_Size			
	字节数	DW	QW	信用值
000b	128	32	16	8
001b	256	64	32	16
010b	512	128	64	32
011b	1024	256	128	64
100b	2048	512	256	128

表 91: Max_Request_Size 设置 (续)

cfg_max_read_req[2:0]] 1	Max_Request_Size			
	字节数	DW	QW	信用值
101b	4096	1024	512	256
110b-111b	保留			

读取完成边界

可通过多个完成包来答复单个存储器请求，将这些完成包放置在一起即可返回请求的所有数据。为了预留资源用于数据包报头开销，用户应用必须分配足够的空间，以便容纳可能返回的最大数量的完成包。

为了简化此进程，《PCI Express 基本规范》量化了所有完成包的长度，使每个完成包都必须始于和止于自然对齐的读取完成边界 (RCB)，除非完成包所服务的对象为原始请求的起始地址或结束地址。如果请求跨地址边界，且边界为 RCB 字节的整数倍，那么可使用多个完成包来完成此类请求，但返回的数据不得分段，位于以下地址边界处的数据除外：

- 第一个完成包必须以请求中指定的地址开始，并且止于以下地址之一：
 - 请求中指定的地址加上请求指定的长度（例如，整个请求）。
 - 地址边界位于请求的起始和结束之间，并且为 RCB 字节的整数倍。
- 最后一个完成包必须止于请求中指定的地址加上请求指定的长度。
- 第一个完成包和最后一个完成包（不含这两个包本身）之间的所有完成包的长度必须为 RCB 字节的整数倍。

在 cfg_rcb_status[3:0] 上会提供 RCB 的编程值。位索引与物理功能编号相关联；例如，cfg_rcb_status[0] 和 cfg_rcb_status[1] 分别与物理功能 0 和 1 相关联（根据功能链路控制寄存器 [3]）。如果用户应用不读取 RCB 值，那么它必须使用默认值，即 64 字节。

表 92：读取完成边界设置

cfg_rcb_status[0]、 cfg_rcb_status[1]、 cfg_rcb_status[2] 或 cfg_rcb_status[3]	读取完成边界			
	字节数	DW	QW	信用值
0	64	16	8	4
1	128	32	16	8

计算非转发请求所需的完成信用值时，必须确定完成响应可能需要的 RCB 绑定块的数量，这与所需的完成报头信用值相同。

有关高性能应用的重要注释

虽然用户应用可以使用经过编程的 RCB 值来计算针对任一请求返回的完成包的最大数量，但大部分高性能存储器控制器都具有可选功能特性可用于将 RCB 大小的完成包组合在一起，构成达到或者接近针对该链路所编程的 Max_Payload_Size 值的完成包，从而响应大型读取请求（读取长度是 RCB 值的倍数）。如果主机 CPU 上的存储器控制器支持该功能特性，那么建议您对其加以妥善利用。对于大小为 RCB 值的整数倍 (>1) 的完成包，基于此类完成包进行数据交换会显著提升 PCI Express 接口利用率和有效载荷效率，并且能够更有效地利用“端点”接收器内的完成空间。

完成空间的管理方法

有 5 种方法可选（如表 C-4 中所示），用户应用可选择其中任一方法来管理接收缓冲器完成空间。为方便起见，此处探讨的内容将这些方法称为 LIMIT_FC、PACKET_FC、RCB_FC 和 DATA_FC。每一种方法都有各自的优缺点，您需要在开发用户应用时妥善考量取舍。

表 93：管理接收完成空间方法

方法	描述	优点	缺点
LIMIT_FC	限制未完成的非转发请求总数	此方法最便于在用户逻辑中实现	有大量完成容量闲置
PACKET_FC	跟踪未完成的 CplH 和 CplD 信用值数量；可基于数据包来进行分配和取消分配	相对简单的用户逻辑；分配粒度更细，这意味着浪费的容量少于 LIMIT_FC	与 LIMIT_FC 相同，非转发的信用值会被占用，直至完全满足请求为止
RCB_FC	跟踪未完成的 CplH 和 CplD 信用值数量；可基于 RCB 来进行分配和取消分配	占用信用值的时间少于 PACKET_FC	用户逻辑比 LIMIT_FC 或 PACKET_FC 更复杂
DATA_FC	跟踪未完成的 CplH 和 CplD 信用值数量；可基于 RCB 来进行分配和取消分配	浪费的容量最少	用户逻辑比 LIMIT_FC、PACKET_FC 和 RCB_FC 更少

LIMIT_FC 方法

LIMIT_FC 方法最便于实现。用户应用会对每次允许的未完成的非转发请求的最大数量（即 MAX_NP）进行评估。要计算该值，请执行以下步骤：

- 判定 Max_Request_Size 包所需的 CplH 信用值数量：

$$\text{Max_Header_Count} = \text{ceiling}(\text{Max_Request_Size} / \text{RCB})$$

- 假定完成包均为最大大小，则判定 CplD 信用值池所支持的此类完成包的最大数量：

$$\text{Max_Packet_Count_CplD} = \text{floor}(\text{CplD} / \text{Max_Request_Size})$$

- 假定完成包均为最大大小，则判定 CplH 信用值池所支持的此类完成包的最大数量：

$$\text{Max_Packet_Count_CplH} = \text{floor}(\text{CplH} / \text{Max_Header_Count})$$

- 使用步骤 2 和步骤 3 中两种数量中较小的值来获取未完成的非转发请求的最大数量：

$$\text{MAX_NP} = \min(\text{Max_Packet_Count_CplH}, \text{Max_Packet_Count_CplD})$$

已知 MAX_NP 后，用户应用可在复位时加载含零位的寄存器 NP_PENDING，并确保它始终处于 0 到 MAX_NP 的范围内。发射非转发请求后，NP_PENDING 减小 1。接收到未完成的非转发请求的所有完成包之后，NP_PENDING 增加 1。例如：

- Max_Request_Size = 128B
- RCB = 64B
- CplH = 64
- CplD = 15,872B
- Max_Header_Count = 2
- Max_Packet_Count_CplD = 124

- Max_Packet_Count_CplH = 32
- MAX_NP = 32

虽然此方法最易于实现，但它浪费的接收器空间最大，因为对于每个非转发请求，都会分配完成信用值的整个 Max_Request_Size 块，与实际请求大小无关。如果用户应用发出的简短的存储器读取（约为单个 DWORD）、I/O 读取和 I/O 写入所占比例增加，那么浪费的量会进一步增加。

PACKET_FC 方法

PACKET_FC 方法用于比 LIMIT_FC 更精细的粒度来分配信用值块，它能以更有效的方式来使用接收完成空间，但所需用户逻辑少量增加。从 CPLH_PENDING 和 CPLD_PENDING（复位时加载，含零位）这两个寄存器开始，然后执行以下步骤：

1. 当用户应用需发送非转发请求时，判定可能需要的 CplH 和 CplID 信用值的潜在数值：

- NP_CplH = ceiling[((Start_Address mod RCB) + Request_Size) / RCB]
- NP_CplID = ceiling[((Start_Address mod 16 bytes) + Request_Size) / 16 字节] (I/O 写入除外，它会返回零数据) [(req_size + 15)/16]

modulo 和 ceiling 函数用于确保将所有细小的 RCB 或信用值块都向上舍入。例如，如果存储器从地址 7Ch 读取 8 字节数据，返回的数据可能分为两个完成包 (7Ch-7Fh, 后接 80h-83h) 来返回。这就需要 2 个 RCB 块和 2 个数据信用值。

2. 检查以下项：

- CPLH_PENDING + NP_CplH < Total_CplH
- CPLD_PENDING + NP_CplID < Total_CplID

3. 如果这两个不等式均成立，则发射非转发请求，将 CPLH_PENDING 增大 NP_CplH，并将 CPLD_PENDING 增大 NP_CplID。对于发射的每个非转发请求，请保留 NP_CplH 和 NP_CplID 以供稍后使用。

4. 当任一非转发请求的所有完成数据都返回后，请相应减小 CPLH_PENDING 和 CPLD_PENDING。

此方法的资源浪费少于 LIMIT_FC，但仍会占用任一非转发请求的全部完成空间直至满足整个请求为止。RCB_FC 和 DATA_FC 可提供更细粒度的解分配，但需要更多逻辑。

RCB_FC 方法

RCB_FC 方法可用于按 RCB 粒度对信用值的块进行分配和解分配。信用值基于 RCB 来进行清除。就像使用 PACKET_FC 时一样，从 CPLH_PENDING 和 CPLD_PENDING（复位时加载，含零位）这两个寄存器开始。

1. 按 RCB 计算数据信用值：

- CplID_PER_RCB = RCB / 16 字节

2. 当用户应用需发送非转发请求时，判定可能需要的 CplH 信用值的潜在数值。使用以下公式来按 RCB 粒度分配 CplID 信用值：

- NP_CplH = ceiling[((Start_Address mod RCB) + Request_Size) / RCB]
- NP_CplID = NP_CplH × CplID_PER_RCB

3. 检查以下项：

- CPLH_PENDING + NP_CplH < Total_CplH
- CPLD_PENDING + NP_CplID < Total_CplID

4. 如果这两个不等式均成立，则发射非转发请求，将 CPLH_PENDING 增大 NP_CplH，并将 CPLD_PENDING 增大 NP_CplD。
5. 每个传入完成包开始时，或者如果该完成包始于或跨某个 RCB 但并未止于此 RCB，则 CPLH_PENDING 减小 1，且 CPLD_PENDING 减小 CplD_PER_RCB。任意完成包均可跨多个 RCB。跨 RCB 数量可按如下方式计算：
 - $RCB_CROSSED = \text{ceiling}[(Lower_Address \bmod RCB) + Length] / RCB]$

“Lower_Address”和“Length”字段均可从完成报头进行解析。或者，也可以在每个传入完成包起始位置加载含 Lower_Address 的寄存器 CUR_ADDR，按相应的 DW 或 QW 递增，然后一旦 CUR_ADDR 发生翻滚，即可对 RCB 进行计数。

此方法的资源浪费少于 PACKET_FC，但仍提供 RCB 粒度。如果用户应用发射 I/O 请求，那么用户应用可采用如下策略，即为每个 I/O 读取仅分配 1 个 CplID 信用值，为每个 I/O 写入分配 0 个 CplID 信用值。用户应用必须按原始非转发请求的类型（存储器写入、I/O 读取、I/O 写入）来为传入完成包匹配每个标签。

DATA_FC 方法

DATA_FC 方法以牺牲逻辑为代价，提供了最精细的分配粒度。就像使用 PACKET_FC 和 RCB_FC 时一样，从 CPLH_PENDING 和 CPLD_PENDING（复位时加载，含零位）这两个寄存器开始。

1. 当用户应用需发送非转发请求时，判定可能需要的 CplH 和 CplD 信用值的潜在数值：
 - $NP_CplH = \text{ceiling}[(Start_Address \bmod RCB) + Request_Size] / RCB]$
 - $NP_CplD = \text{ceiling}[(Start_Address \bmod 16 \text{ bytes}) + Request_Size] / 16 \text{ 字节}$ (I/O 写入除外，它会返回零数据)
2. 检查以下项：
 - $CPLH_PENDING + NP_CplH < Total_CplH$
 - $CPLD_PENDING + NP_CplD < Total_CplD$
3. 如果这两个不等式均成立，则发射非转发请求，将 CPLH_PENDING 增大 NP_CplH，并将 CPLD_PENDING 增大 NP_CplD。
4. 每个传入完成包开始时，或者如果该完成包始于或跨某个 RCB 但并未止于此 RCB，则 CPLH_PENDING 减小 1。跨 RCB 数量可按如下方式计算：
 - $RCB_CROSSED = \text{ceiling}[(Lower_Address \bmod RCB) + Length] / RCB]$

“Lower_Address”和“Length”字段均可从完成报头进行解析。或者，也可以在每个传入完成包起始位置加载含 Lower_Address 的寄存器 CUR_ADDR，按相应的 DW 或 QW 递增，然后一旦 CUR_ADDR 发生翻滚，即可对 RCB 进行计数。

5. 每个传输完成包开始时，或者如果该完成包始于或跨自然对齐的信用值边界，则 CPLD_PENDING 减小 1。跨信用值边界次数计算方式如下：
 - $DATA_CROSSED = \text{ceiling}[(Lower_Address \bmod 16 \text{ B}) + Length] / 16 \text{ B}$

或者，也可以在每个传入完成包起始位置加载含 Lower_Address 的寄存器 CUR_ADDR，按相应的 DW 或 QW 递增，然后一旦 CUR_ADDR 滚过每个 16 字节地址边界，即可对 RCB 进行计数。此方法的资源浪费最少，但所需用户逻辑最多。如果需要更精细的粒度，可以将 Total_CplD 值按 2 或 4 比例进行缩放，以分别得到完成 QWORD 或 DWORD 数量，并对数据计算方式进行相应调整。

GT 位置

本附录提供了可供此 IP 核使用的 GT 位置列表，并列出了选择 GT 位置时应予以考量的部分关键建议。以下各章节所含表格根据 IP 自定义期间所选的 PCIe 块位置，列示了可供选择的 GT bank。

- Artix® UltraScale+™ 器件可用的 GT 四通道
- Kintex® UltraScale+™ 器件可用的 GT 四通道
- Virtex® UltraScale+™ 器件可用的 GT 四通道
- Zynq® UltraScale+™ 器件可用的 GT 四通道

每个 GT 四通道 (GT Quad) 均由 4 条 GT 通道组成。为 PCIe IP 选择 GT 四通道时，赛灵思建议使用距离 PCIe 硬核块最近的 GT 四通道。虽然这并非强制要求，但可以改进设计的布局布线和时序收敛。

- 链路宽度 x1、x2 和 x4 需要 1 个绑定的 GT 四通道，并且在 2 个 GT 四通道之间不应分割各通道。
- 链路宽度 x8 需要 2 个相邻的 GT 四通道，这 2 个 GT 四通道需绑定在一起并位于相同 SLR 内。
- 链路宽度 x16 需要 4 个相邻的 GT 四通道，这 4 个 GT 四通道需绑定在一起并位于相同 SLR 内。

默认情况下，PCIe 通道 0 布局在最上层 GT 四通道的最上方 GT 内（如 Vivado® 集成设计环境 (IDE) 的“Device”视图中所示）。后续通道使用沿器件垂直方向向下数（按通道编号递增）的下一个可用 GT。这表示默认情况下，编号最高的 PCIe 通道使用最下层 GT 四通道的最下方的 GT（用于 PCIe）。IP 自定义期间，您可从下拉选项列表中选择所期望的 GT 四通道用于 PCIe 通道 0。

默认情况下，PCIe 参考时钟 (`sys_clk_p/sys_clk_n`) 使用 PCIe 通道 0 GT 四通道中的 `GTREFCLK0` 作为 x1、x2、x4 和 x8 配置。对于 x16 配置，PCIe 参考时钟应在与通道 4-7 或通道 8-11 关联的 GT 四通道上使用 `GTREFCLK0`。这样即可将时钟前向传送至全部 16 个 PCIe 通道。您可通过给设计添加管脚位置约束来修改参考时钟默认位置。

下图显示了与代表性器件的 PCIe 块位置相关的各种 PCIe 链路配置的 GT 四通道和参考时钟的理想选择。

图 139：对应 x1、x2、x4 PCIe 链路宽度最近的 GT 四通道位置

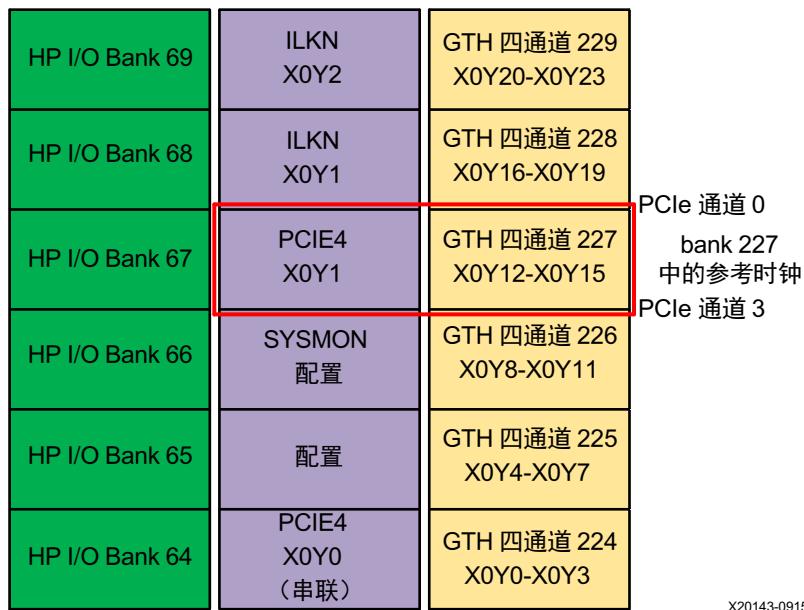


图 140：对应 x8 PCIe 链路宽度最近的 GT 四通道

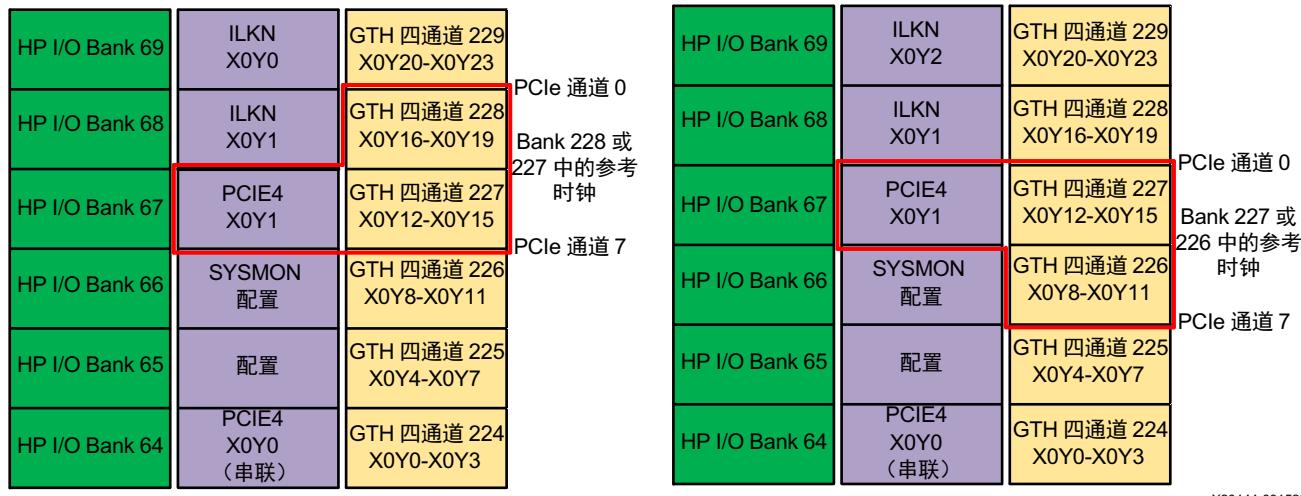
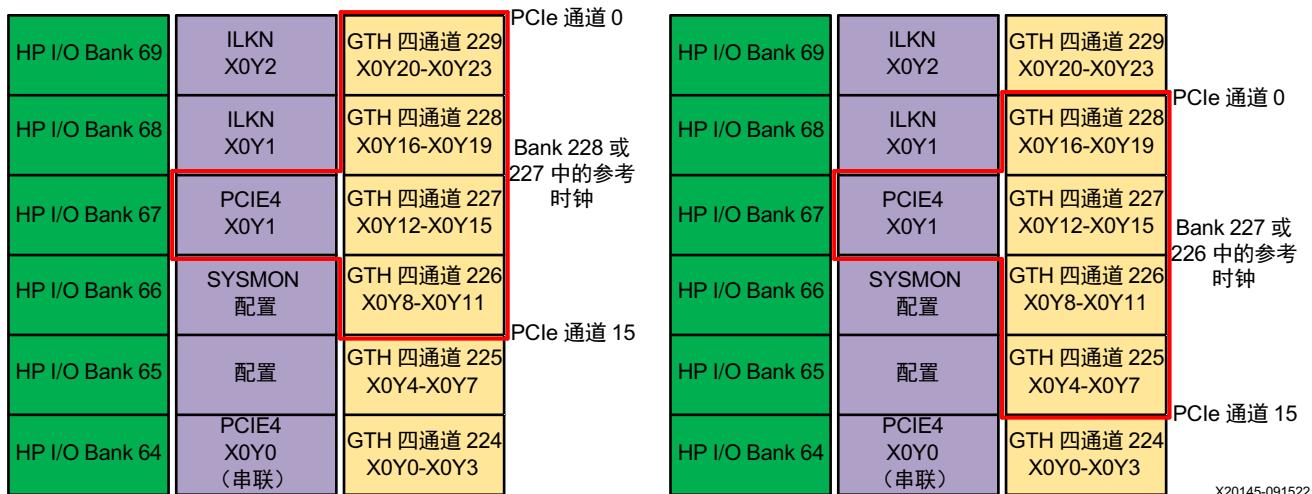
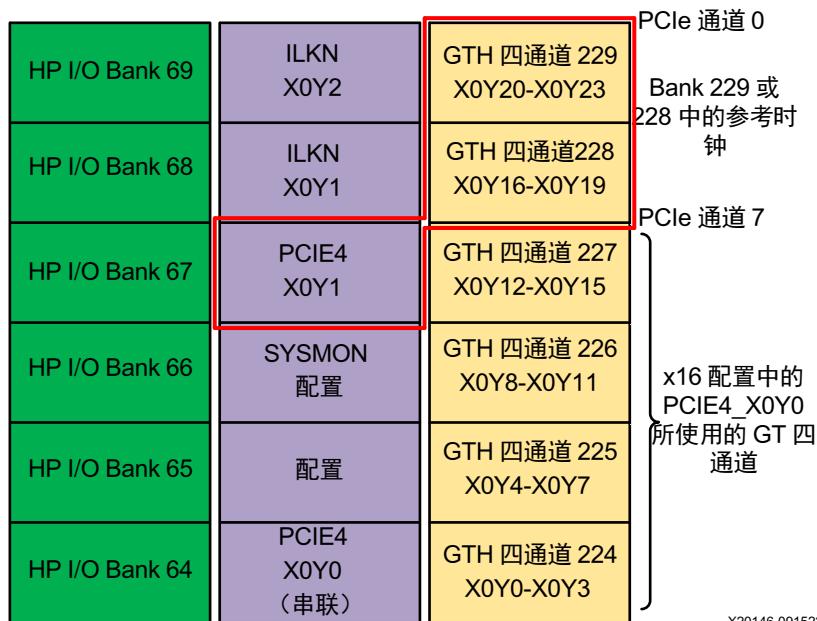


图 141: 对应 x16 PCIe 链路宽度最近的 GT 四通道



部分 PCIe 位置由于紧邻器件边缘、SLR 边界或其它 PCIe 块的边缘而导致无法选择最理想的 GT 四通道。在此类情况下，最近的 GT 可能并非布局布线的最佳选择，但仍可根据期望方式工作。下图显示了 1 个常见示例。

图 142: PCIe GT 位置替代选择



后续章节提供的器件列表中包含针对每一种链路宽度配置支持的 GT 四通道选项。例如，选择 XCVU27P-FSGA2577 器件时，针对 x8 链路宽度支持两种 GTY 四通道组合。

- 当 GUI 中所选选项为 GTY_Quad_225 时，第一个选项是 GTY_Quad_225 搭配 GTY_Quad_224。
- 当 GUI 中所选选项为 GTY_Quad_226 时，第二个选项是 GTY_Quad_226 搭配 GTY_Quad_225。

Artix UltraScale+ 器件可用的 GT 四通道

下表显示了可用于不同 Artix® UltraScale+™ 器件的 PCIe® 通道 0 GT 四通道选项。所示 GT 四通道位置是使用 GT 四通道 bank 编号而不是 GT XY 坐标来表示的。《UltraScale 与 UltraScale+ FPGA 封装和管脚分配产品规格》(UG575) 提供了相应的图示，用于描述已启用的 GT 四通道的相关 PCIe 块的位置，包括 GT bank 编号以及（如果需要）XY 坐标。该表按可用器件、封装和 PCIe 块列出了可供选择的 GT 四通道的编号。IP 自定义期间，在“Basic”（基本）选项卡中，选择“Advance”（高级）模式、选择“Enable GT Quad Selection”（启用 GT 四通道选择），这样即可选择所需 GT 四通道。

注释：粗体选项为每个器件的默认选择，通常为针对具有 x16 PCIe 连接器的开发板经过最优化的选择。如果未指示默认选项，请选择距离 PCIe 集成块最近的 GT 四通道。

表 94：Artix UltraScale+ 器件可用的 GT 四通道

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XCAU20P	SFVB784	PCIE4_X0Y0	不适用	GTY_Quad_225	GTY_Quad_224
	FFVB676	PCIE4_X0Y0	不适用	GTY_Quad_225	GTY_Quad_224
XCAU25P	FFVB676	PCIE4_X0Y0	不适用	GTY_Quad_225	GTY_Quad_224
	SFVB784	PCIE4_X0Y0	不适用	GTY_Quad_225	GTY_Quad_224
XCAU10P	FFVB676	PCIE4C_X0Y0	不适用	GTH_Quad_225	GTH_Quad_224
	UBVA368	PCIE4C_X0Y0	不适用	GTH_Quad_225	GTH_Quad_224
	SBVB484	PCIE4C_X0Y0	不适用	GTH_Quad_225	GTH_Quad_224
XCAU15P	FFVB676	PCIE4C_X0Y0	不适用	GTH_Quad_225	GTH_Quad_224
	UBVA368	PCIE4C_X0Y0	不适用	GTH_Quad_225	GTH_Quad_224
	SBVB484	PCIE4C_X0Y0	不适用	GTH_Quad_225	GTH_Quad_224

Kintex UltraScale+ 器件可用的 GT 四通道

下表显示了可用于不同 Kintex® UltraScale+™ 器件的 PCIe® 通道 0 GT 四通道选项。所示 GT 四通道位置是使用 GT 四通道 bank 编号而不是 GT XY 坐标来表示的。《UltraScale 与 UltraScale+ FPGA 封装和管脚分配产品规格》(UG575) 提供了相应的图示，用于描述已启用的 GT 四通道的相关 PCIe 块的位置，包括 GT bank 编号以及（如果需要）XY 坐标。

该表按可用器件、封装和 PCIe 块列出了可供选择的 GT 四通道的编号。IP 自定义期间，在“Basic”（基本）选项卡中，选择“Advance”（高级）模式、选择“Enable GT Quad Selection”（启用 GT 四通道选择），这样即可选择所需 GT 四通道。

注释：粗体选项为每个器件的默认选择，通常为针对具有 x16 PCIe 连接器的开发板经过最优化的选择。如果未指示默认选项，请选择距离 PCIe 集成块最近的 GT 四通道。

表 95: Kintex UltraScale+ 器件可用的 GT 四通道 (XCKU11P)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XCKU11P	FFVE1517	X0Y3	GTY_Quad_131	GTY_Qaud_130, GTY_Qaud_129	GTY_Qaud_128
		X0Y2	GTY_Quad_130	GTY_Qaud_129, GTY_Qaud_128	GTY_Qaud_127
		X1Y0	GTH_Quad_227	GTH_Qaud_226, GTH_Qaud_225	GTH_Qaud_224
		X1Y1	GTH_Quad_229	GTH_Qaud_228, GTH_Qaud_227	GTH_Qaud_226
	FFVA1156	X0Y3	不支持 X16	GTY_Quad_130	GTY_Qaud_129
		X0Y2	不支持 X16	GTY_Quad_130	GTY_Qaud_129
		X1Y0	GTH_Quad_227	GTH_Qaud_226, GTH_Qaud_225	GTH_Qaud_224
		X1Y1	不支持 X16	GTH_Quad_228 , GTH_Qaud_227	GTH_Qaud_226
	FFVD900	X1Y0	GTH_Quad_227	GTH_Qaud_226, GTH_Qaud_225	GTH_Qaud_224
		X1Y1	不支持 X16	GTH_Quad_227	GTH_Qaud_226

表 96: Kintex® UltraScale+™ 器件可用的 GT 四通道 (XCKU15P)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XCKU15P	FFVE1517	X0Y3	GTY_Qaud_132	GTY_Qaud_131, GTY_Qaud_130	GTY_Qaud_129
		X0Y2	GTY_Qaud_130	GTY_Qaud_129, GTY_Qaud_128	GTY_Qaud_127
		X1Y0	GTH_Qaud_227	GTH_Qaud_226, GTH_Qaud_225	GTH_Qaud_224
		X1Y1	GTH_Qaud_229	GTH_Qaud_228, GTH_Qaud_227	GTH_Qaud_226
		X1Y2	GTH_Qaud_231	GTH_Qaud_230, GTH_Qaud_229	GTH_Qaud_228
	FFVA1156	X0Y3	不支持 X16	GTY_Qaud_130	GTY_Qaud_129
		X0Y2	不支持 X16	GTY_Qaud_130	GTY_Qaud_129
		X1Y0	GTH_Qaud_227	GTH_Qaud_226, GTH_Qaud_225	GTH_Qaud_224
		X1Y1	不支持 X16	GTH_Qaud_228 , GTH_Qaud_227	GTH_Qaud_226
		X1Y2	不支持 X16	GTH_Qaud_228 , GTH_Qaud_227	GTH_Qaud_226
	FFVA1760	X0Y3	GTY_Qaud_132	GTY_Qaud_131, GTY_Qaud_130	GTY_Qaud_129
		X0Y2	GTY_Qaud_130	GTY_Qaud_129, GTY_Qaud_128	GTY_Qaud_127
		X1Y0	GTH_Qaud_227	GTH_Qaud_226, GTH_Qaud_225	GTH_Qaud_224
		X1Y1	GTH_Qaud_229	GTH_Qaud_228, GTH_Qaud_227	GTH_Qaud_226
		X1Y2	GTH_Qaud_231	GTH_Qaud_230, GTH_Qaud_229	GTH_Qaud_228
	FFVE1760	X0Y3	GTY_Qaud_132	GTY_Qaud_131, GTY_Qaud_130	GTY_Qaud_129
		X0Y2	GTY_Qaud_130	GTY_Qaud_129, GTY_Qaud_128	GTY_Qaud_127
		X1Y0	GTH_Qaud_227	GTH_Qaud_226, GTH_Qaud_225	GTH_Qaud_224
		X1Y1	GTH_Qaud_229	GTH_Qaud_228, GTH_Qaud_227	GTH_Qaud_226
		X1Y2	GTH_Qaud_231	GTH_Qaud_230, GTH_Qaud_229	GTH_Qaud_228

表 97: Kintex® UltraScale+™ 器件可用的 GT 四通道 (XCKU3P)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XCKU3P	FFVA676	X0Y0	GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224
	FFVB676	X0Y0	GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224
	FFVD900	X0Y0	GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224
	SFVB784	X0Y0	GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224

表 98: Kintex® UltraScale+™ 器件可用的 GT 四通道 (XCKU5P)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XCKU5P	FFVA676	X0Y0	GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224
	FFVB676	X0Y0	GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224
	FFVD900	X0Y0	GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224
	SFVB784	X0Y0	GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224

表 99: Kintex® UltraScale+™ 器件可用的 GT 四通道 (XCKU19P)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XCKU19P	FFVB2104	PCIE4_X0Y0	GTY_Quad_228	GTY_Quad_227 和 GTY_Quad_226	GTY_Quad_225
		PCIE4_X0Y1	GTY_Quad_230	GTY_Quad_229 和 GTY_Quad_228	GTY_Quad_227
		PCIE4_X0Y2	GTY_Quad_232	GTY_Quad_231 和 GTY_Quad_230	GTY_Quad_229
	FFVJ1760	PCIE4_X0Y0	GTY_Quad_228	GTY_Quad_227 和 GTY_Quad_226	GTY_Quad_225
		PCIE4_X0Y1	GTY_Quad_230	GTY_Quad_229 和 GTY_Quad_228	GTY_Quad_227
		PCIE4_X0Y2	GTY_Quad_232	GTY_Quad_231 和 GTY_Quad_230	GTY_Quad_229

Virtex UltraScale+ 器件可用的 GT 四通道

下表显示了可用于不同 Virtex® UltraScale+™ 器件的 PCIe® 通道 0 GT 四通道选项。所示 GT 四通道位置是使用 GT 四通道 bank 编号而不是 GT XY 坐标来表示的。《UltraScale 与 UltraScale+ FPGA 封装和管脚分配产品规格》(UG575) 提供了相应的图示，用于描述已启用的 GT 四通道的相关 PCIe 块的位置，包括 GT bank 编号以及（如果需要）XY 坐标。

该表按可用器件、封装和 PCIe 块列出了可供选择的 GT 四通道的编号。IP 自定义期间，在“Basic”（基本）选项卡中，选择“Advance”（高级）模式、选择“Enable GT Quad Selection”（启用 GT 四通道选择），这样即可选择所需 GT 四通道。

注释：粗体选项为每个器件的默认选择，通常为针对具有 x16 PCIe 连接器的开发板经过最优化的选择。如果未指示默认选项，请选择距离 PCIe 集成块最近的 GT 四通道。

表 100: Virtex UltraScale+ 器件可用的 GT 四通道 (XCVU27P)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XCVU27P	FSGA2577	PCIE4_X0Y0	GTY_Quad_227	GTY_Quad_225, GTY_Quad_226	GTY_Quad_224
	FIGD2104	PCIE4_X0Y0	GTY_Quad_227	GTY_Quad_225, GTY_Quad_226	GTY_Quad_224

表 101: Virtex UltraScale+ 器件可用的 GT 四通道 (XCVU3P)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XCVU3P	FFVC1517	X0Y1	GTY_Quad_128	GTY_Quad_127, GTY_Quad_126	GTY_Quad_125
		X1Y0	GTY_Quad_228, GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224
		PCIE4_X0Y1	GTY_Quad_128	GTY_Quad_127, GTY_Quad_126	GTY_Quad_125
		PCIE4_X1Y0	GTY_Quad_228, GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_124

表 102: Virtex UltraScale+ 器件可用的 GT 四通道 (XCVU5P)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XCVU5P	FLVA2104	X0Y1	不支持 X16	GTY_Quad_127 , GTY_Quad_126	GTY_Quad_125
		X0Y3	不支持 X16	GTY_Quad_132 , GTY_Quad_131	GTY_Quad_130
		X1Y0	GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224
		X1Y2	不支持 X16	GTY_Quad_233 , GTY_Quad_232	GTY_Quad_231
	FLVB2104	X0Y1	GTY_Quad_128	GTY_Quad_127, GTY_Quad_126	GTY_Quad_125
		X0Y3	GTY_Quad_133	GTY_Quad_132, GTY_Quad_131	GTY_Quad_130
		X1Y0	GTY_Quad_228 , GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224
		X1Y2	GTY_Quad_233, GTY_Quad_232	GTY_Quad_231, GTY_Quad_230	GTY_Quad_229
XCVU5P	FLVC2104	X0Y1	GTY_Quad_128	GTY_Quad_127, GTY_Quad_126	GTY_Quad_125
		X0Y3	GTY_Quad_133	GTY_Quad_132, GTY_Quad_131	GTY_Quad_130
		X1Y0	GTY_Quad_228, GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224
		X1Y2	GTY_Quad_233, GTY_Quad_232	GTY_Quad_231, GTY_Quad_230	GTY_Quad_229

表 103: Virtex UltraScale+ 器件可用的 GT 四通道 (XCVU7P)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XCVU7P	FLVA2104	X0Y1	不支持 X16	GTY_Quad_127, GTY_Quad_126	GTY_Quad_125
		X0Y3	不支持 X16	GTY_Quad_132, GTY_Quad_131	GTY_Quad_130
		X1Y0	GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224
		X1Y2	不支持 X16	GTY_Quad_233, GTY_Quad_232	GTY_Quad_231
	FLVB2104	X0Y1	GTY_Quad_128	GTY_Quad_127, GTY_Quad_126	GTY_Quad_125
		X0Y3	GTY_Quad_133	GTY_Quad_132, GTY_Quad_131	GTY_Quad_130
		X1Y0	GTY_Quad_228, GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224
		X1Y2	GTY_Quad_233, GTY_Quad_232	GTY_Quad_231, GTY_Quad_230	GTY_Quad_229
	FLVC2104	X0Y1	GTY_Quad_128	GTY_Quad_127, GTY_Quad_126	GTY_Quad_125
		X0Y3	GTY_Quad_133	GTY_Quad_132, GTY_Quad_131	GTY_Quad_130
		X1Y0	GTY_Quad_228 , GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224
		X1Y2	GTY_Quad_233, GTY_Quad_232	GTY_Quad_231, GTY_Quad_230	GTY_Quad_229

表 104: Virtex UltraScale+ 器件可用的 GT 四通道 (XCVU9P)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XCVU9P	FLGA2104	X0Y1	不支持 X16	GTY_Quad_122 , GTY_Quad_121	GTY_Quad_120
		X0Y3	不支持 X16	GTY_Quad_127 , GTY_Quad_126	GTY_Quad_125
		X1Y2	GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224
		X1Y4	不支持 X16	GTY_Quad_233 , GTY_Quad_232	GTY_Quad_231
	FLGB2104	X0Y1	GTY_Quad_123	GTY_Quad_122, GTY_Quad_121	GTY_Quad_120
		X0Y3	GTY_Quad_128	GTY_Quad_127, GTY_Quad_126	GTY_Quad_125
		X1Y2	GTY_Quad_228, GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224
		X1Y4	GTY_Quad_233, GTY_Quad_232	GTY_Quad_231, GTY_Quad_230	GTY_Quad_229
	FLGC2104	X0Y1	不支持 X16	GTY_Quad_122 , GTY_Quad_121	GTY_Quad_120
		X0Y3	GTY_Quad_128	GTY_Quad_127, GTY_Quad_126	GTY_Quad_125
		X0Y5	GTY_Quad_133	GTY_Quad_132, GTY_Quad_131	GTY_Quad_130
		X1Y0	不支持 X16	GTY_Quad_222 , GTY_Quad_221	GTY_Quad_220
		X1Y2	GTY_Quad_228, GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224
		X1Y4	GTY_Quad_233, GTY_Quad_232	GTY_Quad_231, GTY_Quad_230	GTY_Quad_229

表 104: Virtex UltraScale+ 器件可用的 GT 四通道 (XCVU9P) (续)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XCVU9P	FLGA2577	X0Y1	GTY_Quad_123	GTY_Quad_122, GTY_Quad_121	GTY_Quad_120
		X0Y3	GTY_Quad_128	GTY_Quad_127, GTY_Quad_126	GTY_Quad_125
		X0Y5	GTY_Quad_133	GTY_Quad_132, GTY_Quad_131	GTY_Quad_130
		X1Y0	GTY_Quad_223, GTY_Quad_222	GTY_Quad_221, GTY_Quad_220	GTY_Quad_219
		X1Y2	GTY_Quad_228, GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224
		X1Y4	GTY_Quad_233, GTY_Quad_232	GTY_Quad_231, GTY_Quad_230	GTY_Quad_229
	FSGD2104	X0Y1	GTY_Quad_123	GTY_Quad_122, GTY_Quad_121	GTY_Quad_120
		X0Y3	GTY_Quad_127	GTY_Quad_126, GTY_Quad_125	GTY_Quad_124
		X0Y5	不支持 X16	不支持 X8	GTY_Quad_131
		X1Y2	GTY_Quad_228, GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224
		X1Y4	GTY_Quad_233, GTY_Quad_232	GTY_Quad_231, GTY_Quad_230	GTY_Quad_229

表 105: Virtex UltraScale+ 器件可用的 GT 四通道 (XCVU11P)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XCVU11P	FLGA2577	X0Y0	GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224
		X0Y1	GTY_Quad_231	GTY_Quad_230, GTY_Quad_229	GTY_Quad_228
		X0Y2	GTY_Quad_235	GTY_Quad_234, GTY_Quad_233	GTY_Quad_232
	FLGB2104	X0Y0	GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224
		X0Y1	GTY_Quad_231	GTY_Quad_230, GTY_Quad_229	GTY_Quad_228
		X0Y2	不支持 X16	GTY_Quad_233	GTY_Quad_232
	FLGC2104	X0Y0	GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224
		X0Y1	GTY_Quad_231	GTY_Quad_230, GTY_Quad_229	GTY_Quad_228
		X0Y2	GTY_Quad_235	GTY_Quad_234, GTY_Quad_233	GTY_Quad_232
	FLGF1924	X0Y0	GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224
		X0Y1	GTY_Quad_231	GTY_Quad_230, GTY_Quad_229	GTY_Quad_228
		X0Y2	不支持 X16	GTY_Quad_233	GTY_Quad_232
	FSGD2104	X0Y0	GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224
		X0Y1	GTY_Quad_231	GTY_Quad_230, GTY_Quad_229	GTY_Quad_228
		X0Y2	不支持 X16	GTY_Quad_233	GTY_Quad_232

表 106: Virtex UltraScale+ 器件可用的 GT 四通道 (XCVU13P)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XCVU13P	FHGA2104	X0Y1	GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224
		X0Y2	不支持 X16	GTY_Quad_231 , GTY_Quad_230	GTY_Quad_229
	FHGB2104	X0Y1	GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224
		X0Y2	GTY_Quad_231	GTY_Quad_230, GTY_Quad_229	GTY_Quad_228
		X0Y3	不支持 X16	GTY_Quad_233	GTY_Quad_232
	FHGC2014	X0Y0	不支持 X16	GTY_Quad_223 , GTY_Quad_222	GTY_Quad_221
		X0Y1	GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224
		X0Y2	GTY_Quad_231	GTY_Quad_230, GTY_Quad_229	GTY_Quad_228
		X0Y3	不支持 X16	GTY_Quad_233	GTY_Quad_232
	FLGA2577	X0Y0	GTY_Quad_223	GTY_Quad_222, GTY_Quad_221	GTY_Quad_220
		X0Y1	GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224
		X0Y2	GTY_Quad_231	GTY_Quad_230, GTY_Quad_229	GTY_Quad_228
		X0Y3	GTY_Quad_235	GTY_Quad_234, GTY_Quad_233	GTY_Quad_232
	FIGD2104	X0Y1	GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224
		X0Y2	GTY_Quad_231	GTY_Quad_230, GTY_Quad_229	GTY_Quad_228
		X0Y3	不支持 X16	GTY_Quad_233	GTY_Quad_232
	FSGA2577	X0Y0	GTY_Quad_223	GTY_Quad_222, GTY_Quad_221	GTY_Quad_220
		X0Y1	GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224
		X0Y2	GTY_Quad_231	GTY_Quad_230, GTY_Quad_229	GTY_Quad_228
		X0Y3	GTY_Quad_235	GTY_Quad_234, GTY_Quad_233	GTY_Quad_232

表 107: Virtex UltraScale+ 器件可用的 GT 四通道 (XCVU35P)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XCVU35P	FSVH2892	PCIE4C_X0Y0	GTY_Quad_127	GTY_Quad_126 和 GTY_Quad_125	GTY_Quad_124
		PCIE4C_X0Y1	GTY_Quad_127	GTY_Quad_126 和 GTY_Quad_125	GTY_Quad_124
		PCIE4C_X1Y1	GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224
		PCIE4C_X1Y0	GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224
		PCIE4_X0Y0	GTY_Quad_231	GTY_Quad_230, GTY_Quad_229	GTY_Quad_228
	FSVH2104	PCIE4C_X0Y0	GTY_Quad_127	GTY_Quad_126 和 GTY_Quad_125	GTY_Quad_124
		PCIE4C_X0Y1	GTY_Quad_127	GTY_Quad_126 和 GTY_Quad_125	GTY_Quad_124
		PCIE4C_X1Y1	GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224
		PCIE4C_X1Y0	GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224
		PCIE4_X0Y0	GTY_Quad_231	GTY_Quad_230, GTY_Quad_229	GTY_Quad_228

表 108: Virtex UltraScale+ 器件可用的 GT 四通道 (XCVU37P)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XCVU37P	FSVH2892	PCIE4C_X0Y0	GTY_Quad_127	GTY_Quad_126 和 GTY_Quad_125	GTY_Quad_124
		PCIE4C_X0Y1	GTY_Quad_127	GTY_Quad_126 和 GTY_Quad_125	GTY_Quad_124
		PCIE4C_X1Y1	GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224
		PCIE4C_X1Y0	GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224
		PCIE4_X0Y0	GTY_Quad_231	GTY_Quad_230, GTY_Quad_229	GTY_Quad_228
		PCIE4_X0Y1	GTY_Quad_235	GTY_Quad_234, GTY_Quad_233	GTY_Quad_232

表 109: Virtex UltraScale+ 器件可用的 GT 四通道 (XCVU31P)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XCVU31P	FSVH1924	PCIE4C_X0Y0	GTY_Quad_127	GTY_Quad_126, GTY_Quad_125	GTY_Quad_124
		PCIE4C_X0Y1	GTY_Quad_127	GTY_Quad_126 和 GTY_Quad_125	GTY_Quad_124
		PCIE4C_X1Y0	GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224
		PCIE4C_X1Y1	GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224

表 110: Virtex UltraScale+ 器件可用的 GT 四通道 (XCVU33P)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XCVU33P	FSVH2104	PCIE4C_X0Y0	GTY_Quad_127	GTY_Quad_126, GTY_Quad_125	GTY_Quad_124
		PCIE4C_X0Y1	GTY_Quad_127	GTY_Quad_126, GTY_Quad_125	GTY_Quad_124
		PCIE4C_X1Y0	GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224
		PCIE4C_X1Y1	GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224

表 111: Virtex UltraScale+ 器件可用的 GT 四通道 (XQVU7P)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XQVU7P	FLRA2104	X0Y1	不支持 X16	GTY_Quad_127, GTY_Quad_126	GTY_Quad_125
		X0Y3	不支持 X16	GTY_Quad_132, GTY_Quad_131	GTY_Quad_130
		X1Y0	GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224
		X1Y2	不支持 X16	GTY_Quad_233, GTY_Quad_232	GTY_Quad_231
	FLRB2104	X0Y1	GTY_Quad_128	GTY_Quad_127, GTY_Quad_126	GTY_Quad_125
		X0Y3	GTY_Quad_133	GTY_Quad_132, GTY_Quad_131	GTY_Quad_130
		X1Y0	GTY_Quad_228, GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224
		X1Y2	GTY_Quad_233, GTY_Quad_232	GTY_Quad_231, GTY_Quad_230	GTY_Quad_229

表 112: Virtex UltraScale+ 器件可用的 GT 四通道 (XQVU11P)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XQVU11P	FLRC2104	X0Y0	GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224
		X0Y1	GTY_Quad_231	GTY_Quad_230, GTY_Quad_229	GTY_Quad_228
		X0Y2	GTY_Quad_235	GTY_Quad_234, GTY_Quad_233	GTY_Quad_232

表 113: Virtex UltraScale+ 器件可用的 GT 四通道 (XCVU29P)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XCVU29P	figd2104	X0Y0	GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224
	fsga2577	X0Y0	GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224

表 114: Virtex UltraScale+ 器件可用的 GT 四通道 (XQVU3P)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XQVU3P	FFRC1517	PCIE4_X0Y1	GTY_Quad_128	GTY_Quad_127, GTY_Quad_126	GTY_Quad_125
		PCIE4_X1Y0	GTY_Quad_228, GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_124

表 115: Virtex UltraScale+ 器件可用的 GT 四通道 (XCVU45P)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XCVU45P	FSVH2892	PCIE4C_X1Y0	GTY_Qaud_127	GTY_Qaud_225, GTY_Qaud_226	GTY_Qaud_224
		PCIE4C_X1Y1	GTY_Qaud_127	GTY_Qaud_225, GTY_Qaud_226	GTY_Qaud_224
		PCIE4C_X0Y1	GTY_Qaud_127	GTY_Qaud_125, GTY_Qaud_126	GTY_Qaud_124
		PCIE4C_X0Y0	GTY_Qaud_127	GTY_Qaud_125, GTY_Qaud_126	GTY_Qaud_124
		PCIE4_X0Y0	GTY_Qaud_231	GTY_Qaud_230, GTY_Qaud_229	GTY_Qaud_228
	FSVH2104	PCIE4C_X1Y0	GTY_Qaud_227	GTY_Qaud_225, GTY_Qaud_226	GTY_Qaud_224
		PCIE4C_X1Y1	GTY_Qaud_227	GTY_Qaud_225, GTY_Qaud_226	GTY_Qaud_224
		PCIE4C_X0Y1	GTY_Qaud_127	GTY_Qaud_125, GTY_Qaud_126	GTY_Qaud_124
		PCIE4C_X0Y0	GTY_Qaud_127	GTY_Qaud_125, GTY_Qaud_126	GTY_Qaud_124
		PCIE4_X0Y0	GTY_Qaud_231	GTY_Qaud_230, GTY_Qaud_229	GTY_Qaud_228

表 116: Virtex UltraScale+ 器件可用的 GT 四通道 (XCVU47P)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XCVU47P	FSVH2892	PCIE4C_X1Y0	GTY_Qaud_227	GTY_Qaud_225、 GTY_Qaud_226 和	GTY_Qaud_224
		PCIE4C_X1Y1	GTY_Qaud_227	GTY_Qaud_225、 GTY_Qaud_226 和	GTY_Qaud_224
		PCIE4C_X0Y1	GTY_Qaud_127	GTY_Qaud_125、 GTY_Qaud_126 和	GTY_Qaud_124、
		PCIE4C_X0Y0	GTY_Qaud_127	GTY_Qaud_125、 GTY_Qaud_126 和	GTY_Qaud_124、
		PCIE4_X0Y1	GTY_Qaud_235	GTY_Qaud_234、 GTY_Qaud_233	GTY_Qaud_232
		PCIE4_X0Y0	GTY_Qaud_231	GTY_Qaud_230、 GTY_Qaud_229	GTY_Qaud_228

表 117: Virtex UltraScale+ 器件可用的 GT 四通道 (XCVU7P_C)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XCVU7P_C	FLVA2104	PCIE4_X0Y1	不支持 X16	GTY_Quad_127 , GTY_Quad_126	GTY_Quad_125
		PCIE4_X0Y3	不支持 X16	GTY_Quad_132 , GTY_Quad_131	GTY_Quad_130
		PCIE4_X1Y0	GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224
		PCIE4_X1Y2	不支持 X16	GTY_Quad_233 , GTY_Quad_232	GTY_Quad_231
	FLVB2104	PCIE4_X0Y1	GTY_Quad_128	GTY_Quad_127, GTY_Quad_126	GTY_Quad_125
		PCIE4_X0Y3	GTY_Quad_133	GTY_Quad_132, GTY_Quad_131	GTY_Quad_130
		PCIE4_X1Y0	GTY_Quad_228, GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224
		PCIE4_X1Y2	GTY_Quad_233, GTY_Quad_232	GTY_Quad_233, GTY_Quad_232	GTY_Quad_229
	FLVC2104	PCIE4_X0Y1	GTY_Quad_128	GTY_Quad_127, GTY_Quad_126	GTY_Quad_125
		PCIE4_X0Y3	GTY_Quad_133	GTY_Quad_132, GTY_Quad_131	GTY_Quad_130
		PCIE4_X1Y0	GTY_Quad_228 , GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224
		PCIE4_X1Y2	GTY_Quad_233, GTY_Quad_232	GTY_Quad_233, GTY_Quad_232	GTY_Quad_229

表 118: Virtex UltraScale+ 器件可用的 GT 四通道 (XQVU9P)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XQVU9P	FLQA2104	PCIE4_X0Y1	不支持 X16	GTY_Quad_122, GTY_Quad_121	GTY_Quad_120
		PCIE4_X0Y3	不支持 X16	GTY_Quad_127, GTY_Quad_126	GTY_Quad_125
		PCIE4_X1Y2	GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224
		PCIE4_X1Y4	不支持 X16	GTY_Quad_233, GTY_Quad_232	GTY_Quad_231
	FLQB2104	PCIE4_X0Y1	GTY_Quad_123	GTY_Quad_122, GTY_Quad_121	GTY_Quad_120
		PCIE4_X0Y3	GTY_Quad_128	GTY_Quad_127, GTY_Quad_126	GTY_Quad_125
		PCIE4_X1Y2	GTY_Quad_228, GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224
		PCIE4_X1Y4	GTY_Quad_233 , GTY_Quad_232	GTY_Quad_231, GTY_Quad_230	GTY_Quad_229
	FSQD2104	PCIE4_X0Y1	GTY_Quad_123	GTY_Quad_122, GTY_Quad_121	GTY_Quad_120
		PCIE4_X0Y3	GTY_Quad_127	GTY_Quad_126, GTY_Quad_125	GTY_Quad_124
		PCIE4_X0Y5	不支持 X16	不支持 X16	GTY_Quad_131
		PCIE4_X1Y2	GTY_Quad_228, GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224
		PCIE4_X1Y4	GTY_Quad_233, GTY_Quad_232	GTY_Quad_231, GTY_Quad_230	GTY_Quad_229

表 119: Virtex UltraScale+ 器件可用的 GT 四通道 (XQVU13P)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XQVU13P	FHQA2104	PCIE4_X0Y1	GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224
		PCIE4_X0Y2	不支持 X16	GTY_Quad_231 , GTY_Quad_230	GTY_Quad_229
	FHQB2104	PCIE4_X0Y1	GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224
		PCIE4_X0Y2	GTY_Quad_231	GTY_Quad_230, GTY_Quad_229	GTY_Quad_228
		PCIE4_X0Y3	不支持 X16	GTY_Quad_233	GTY_Quad_232
	FIQD2104	PCIE4_X0Y1	GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224
		PCIE4_X0Y2	GTY_Quad_231	GTY_Quad_230, GTY_Quad_229	GTY_Quad_228
		PCIE4_X0Y3	不支持 X16	GTY_Quad_233	GTY_Quad_232

表 120: Virtex UltraScale+ 器件可用的 GT 四通道 (XQVU37P)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XQVU37P	FSQH2892	PCIE4C_X0Y0	GTY_Quad_127	GTY_Quad_126, GTY_Quad_125	GTY_Quad_124
		PCIE4C_X0Y1	GTY_Quad_127	GTY_Quad_126, GTY_Quad_125	GTY_Quad_124
		PCIE4C_X1Y1	GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224
		PCIE4C_X1Y0	GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224

表 121: Virtex UltraScale+ 器件可用的 GT 四通道 (XCVU19P)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XCVU19P	FSVA3824	PCIE4C_X0Y0	不支持 X16	GTY_Quad_222 , GTY_Quad_221	GTY_Quad_220
		PCIE4C_X0Y1	不支持 X16	GTY_Quad_222 , GTY_Quad_221	GTY_Quad_220
		PCIE4C_X0Y2	不支持 X16	GTY_Quad_227 , GTY_Quad_226	GTY_Quad_225
		PCIE4C_X0Y3	不支持 X16	GTY_Quad_227 , GTY_Quad_226	GTY_Quad_225
		PCIE4C_X0Y4	不支持 X16	GTY_Quad_232 , GTY_Quad_231	GTY_Quad_230
		PCIE4C_X0Y5	不支持 X16	GTY_Quad_232 , GTY_Quad_231	GTY_Quad_230
		PCIE4C_X0Y6	不支持 X16	GTY_Quad_237 , GTY_Quad_236	GTY_Quad_235
		PCIE4C_X0Y7	不支持 X16	GTY_Quad_237 , GTY_Quad_236	GTY_Quad_235
	FSVB3824	PCIE4C_X0Y0	GTY_Quad_222	GTY_Quad_221, GTY_Quad_220	GTY_Quad_219
		PCIE4C_X0Y1	GTY_Quad_223	GTY_Quad_222 , GTY_Quad_221	GTY_Quad_220
		PCIE4C_X0Y2	GTY_Quad_227	GTY_Quad_226, GTY_Quad_225	GTY_Quad_224
		PCIE4C_X0Y3	GTY_Quad_228	GTY_Quad_227 , GTY_Quad_226	GTY_Quad_225
		PCIE4C_X0Y4	GTY_Quad_233	GTY_Quad_232, GTY_Quad_231	GTY_Quad_230
		PCIE4C_X0Y5	GTY_Quad_234	GTY_Quad_233, GTY_Quad_232	GTY_Quad_231
		PCIE4C_X0Y6	GTY_Quad_237	GTY_Quad_236, GTY_Quad_235	GTY_Quad_234
		PCIE4C_X0Y7	GTY_Quad_238	GTY_Quad_237, GTY_Quad_236	GTY_Quad_235

表 122: Virtex UltraScale+ 器件可用的 GT 四通道 (XCVU23P)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XCVU23P	FSVJ1760	PCIE4C_X0Y3	GTY_Qaud_231	GTY_Qaud_230 和 GTY_Qaud_229	GTY_Qaud_228
		PCIE4C_X0Y2	GTY_Qaud_231	GTY_Qaud_230 和 GTY_Qaud_229	GTY_Qaud_228
		PCIE4C_X0Y1	GTY_Qaud_228	GTY_Qaud_227 和 GTY_Qaud_226	GTY_Qaud_225
		PCIE4C_X0Y0	GTY_Qaud_227	GTY_Qaud_226 和 GTY_Qaud_225	GTY_Qaud_224
	VSVA1365	PCIE4C_X0Y3	GTY_Qaud_231	GTY_Qaud_230 和 GTY_Qaud_229	GTY_Qaud_228
		PCIE4C_X0Y2	GTY_Qaud_231	GTY_Qaud_230 和 GTY_Qaud_229	GTY_Qaud_228
		PCIE4C_X0Y1	GTY_Qaud_228	GTY_Qaud_227 和 GTY_Qaud_226	GTY_Qaud_225
		PCIE4C_X0Y0	GTY_Qaud_227	GTY_Qaud_226 和 GTY_Qaud_225	GTY_Qaud_224
	CIV-FSVJ1760	PCIE4C_X0Y3	GTY_Qaud_231	GTY_Qaud_230 和 GTY_Qaud_229	GTY_Qaud_228
		PCIE4C_X0Y2	GTY_Qaud_231	GTY_Qaud_230 和 GTY_Qaud_229	GTY_Qaud_228
		PCIE4C_X0Y1	GTY_Qaud_228	GTY_Qaud_227 和 GTY_Qaud_226	GTY_Qaud_225
		PCIE4C_X0Y0	GTY_Qaud_227	GTY_Qaud_226 和 GTY_Qaud_225	GTY_Qaud_224
	CIV-VSVA1365	PCIE4C_X0Y3	GTY_Qaud_231	GTY_Qaud_230 和 GTY_Qaud_229	GTY_Qaud_228
		PCIE4C_X0Y2	GTY_Qaud_231	GTY_Qaud_230 和 GTY_Qaud_229	GTY_Qaud_228
		PCIE4C_X0Y1	GTY_Qaud_228	GTY_Qaud_227 和 GTY_Qaud_226	GTY_Qaud_225
		PCIE4C_X0Y0	GTY_Qaud_227	GTY_Qaud_226 和 GTY_Qaud_225	GTY_Qaud_224

表 123: Virtex UltraScale+ 器件可用的 GT 四通道 (XCVU57P)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XCVU57P	FSVK2892	PCIE4C_X0Y0	GTY_Qaud_127	GTY_Qaud_126 和 GTY_Qaud_125	GTY_Qaud_124
		PCIE4C_X0Y1	GTY_Qaud_127	GTY_Qaud_126 和 GTY_Qaud_125	GTY_Qaud_124
		PCIE4C_X1Y0	GTY_Qaud_227	GTY_Qaud_226 和 GTY_Qaud_225	GTY_Qaud_224
		PCIE4_X1Y1	GTY_Qaud_227	GTY_Qaud_226 和 GTY_Qaud_225	GTY_Qaud_224

Zynq UltraScale+ 器件可用的 GT 四通道

下表显示了可用于不同 Zynq® UltraScale+™ 器件的 PCIe 通道 0 GT 四通道选项。所示 GT 四通道位置是使用 GT 四通道 bank 编号而不是 GT XY 坐标来表示的。如需了解更多信息，请参阅《Zynq UltraScale+ 器件封装和管脚分配产品规格用户指南》(UG1075)。

该表按可用器件、封装和 PCIe 块列出了可供选择的 GT 四通道的编号。IP 自定义期间，在“Basic”（基本）选项卡中，选择“Advance”（高级）模式、选择“Enable GT Quad Selection”（启用 GT 四通道选择），这样即可选择所需 GT 四通道。

注释：粗体选项为每个器件的默认选择，通常为针对具有 x16 PCIe 连接器的开发板经过最优化的选择。如果未指示默认选项，请选择距离 PCIe 集成块最近的 GT 四通道。

表 124: Zynq UltraScale+ 器件可用的 GT 四通道 (XCZU11EG)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XCZU11EG	FFVC1760	X0Y3	GTY_Quad_131	GTY_Quad_130, GTY_Quad_129	GTY_Quad_128
		X0Y2	不支持 X16	GTY_Quad_130 , GTY_Quad_129	GTY_Quad_128
		X1Y0	GTH_Quad_227	GTH_Quad_226, GTH_Quad_225	GTH_Quad_224
		X1Y1	GTH_Quad_230, GTH_Quad_229	GTH_Quad_228, GTH_Quad_227	GTH_Quad_226
	FFVB1517	X1Y0	GTH_Quad_227	GTH_Quad_226, GTH_Quad_225	GTH_Quad_224
		X1Y1	不支持 X16	GTH_Quad_227	GTH_Quad_226
	FFVC1156	X1Y0	GTH_Quad_227	GTH_Quad_226, GTH_Quad_225	GTH_Quad_224
		X1Y1	不支持 X16	GTH_Quad_228 , GTH_Quad_227	GTH_Quad_226
	FFVF1517	X1Y0	GTH_Quad_227	GTH_Quad_226, GTH_Quad_225	GTH_Quad_224
		X1Y1	GTH_Quad_229	GTH_Quad_228, GTH_Quad_227	GTH_Quad_226

表 125: Zynq UltraScale+ 器件可用的 GT 四通道 (XCZU17EG)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XCZU17EG	FFVC1760	X0Y3	GTY_Quad_131	GTY_Quad_130, GTY_Quad_129	GTY_Quad_128
		X0Y2	不支持 X16	GTY_Quad_130 , GTY_Quad_129	GTY_Quad_128
		X1Y0	GTH_Quad_227	GTH_Quad_226, GTH_Quad_225	GTH_Quad_224
		X1Y1	GTH_Quad_230, GTH_Quad_229	GTH_Quad_228, GTH_Quad_227	GTH_Quad_226
		X1Y2	GTH_Quad_231, GTH_Quad_230	GTH_Quad_229, GTH_Quad_228	GTH_Quad_227
	FFVE1924	X1Y0	GTH_Quad_227	GTH_Quad_226, GTH_Quad_225	GTH_Quad_224
		X1Y1	GTH_Quad_229	GTH_Quad_228, GTH_Quad_227	GTH_Quad_226
		X1Y2	GTH_Quad_230	GTH_Quad_229, GTH_Quad_228	GTH_Quad_227
	FFVB1517	X1Y0	GTH_Quad_227	GTH_Quad_226, GTH_Quad_225	GTH_Quad_224
		X1Y1	不支持 X16	GTH_Quad_227	GTH_Quad_226
	FFVD1760	X0Y3	GTY_Quad_132	GTY_Quad_131, GTY_Quad_130	GTY_Quad_129
		X0Y2	不支持 X16	GTY_Quad_130 , GTY_Quad_129	GTY_Quad_128
		X1Y0	GTH_Quad_227	GTH_Quad_226, GTH_Quad_225	GTH_Quad_224
		X1Y1	GTH_Quad_229	GTH_Quad_228, GTH_Quad_227	GTH_Quad_226
		X1Y2	GTH_Quad_230	GTH_Quad_229, GTH_Quad_228	GTH_Quad_227

表 126: Zynq UltraScale+ 器件可用的 GT 四通道 (XCZU19EG)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XCZU19EG	FFVC1760	X0Y3	GTY_Quad_131	GTY_Quad_130, GTY_Quad_129	GTY_Quad_128
		X0Y2	不支持 X16	GTY_Quad_130 , GTY_Quad_129	GTY_Quad_128
		X1Y0	GTH_Quad_227	GTH_Quad_226, GTH_Quad_225	GTH_Quad_224
		X1Y1	GTH_Quad_230, GTH_Quad_229	GTH_Quad_228, GTH_Quad_227	GTH_Quad_226
		X1Y2	GTH_Quad_231, GTH_Quad_230	GTH_Quad_229, GTH_Quad_228	GTH_Quad_227
	FFVE1924	X1Y0	GTH_Quad_227	GTH_Quad_226, GTH_Quad_225	GTH_Quad_224
		X1Y1	GTH_Quad_229	GTH_Quad_228, GTH_Quad_227	GTH_Quad_226
		X1Y2	GTH_Quad_230	GTH_Quad_229, GTH_Quad_228	GTH_Quad_227
	FFVB1517	X1Y0	GTH_Quad_227	GTH_Quad_226, GTH_Quad_225	GTH_Quad_224
		X1Y1	不支持 X16	GTH_Quad_227	GTH_Quad_226
		X1Y2	不支持 X16	不支持 X8	GTH_Quad_227
	FFVD1760	X0Y3	GTY_Quad_132	GTY_Quad_131, GTY_Quad_130	GTY_Quad_129
		X0Y2	不支持 X16	GTY_Quad_130 , GTY_Quad_129	GTY_Quad_128
		X1Y0	GTH_Quad_227	GTH_Quad_226, GTH_Quad_225	GTH_Quad_224
		X1Y1	GTH_Quad_229	GTH_Quad_228, GTH_Quad_227	GTH_Quad_226
		X1Y2	GTH_Quad_230	GTH_Quad_229, GTH_Quad_228	GTH_Quad_227

表 127: Zynq UltraScale+ 器件可用的 GT 四通道 (XCZU4EV)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XCZU4EV	FBVB900	X0Y0	GTH_Quad_226	GTH_Quad_225, GTH_Quad_224	GTH_Quad_223
		X0Y1	GTH_Quad_226	GTH_Quad_225, GTH_Quad_224	GTH_Quad_223
	SFVC784	X0Y0	不支持 X16	不支持 X8	GTH_Quad_224
		X0Y1	不支持 X16	不支持 X8	GTH_Quad_224

表 128: Zynq UltraScale+ 器件可用的 GT 四通道 (XCZU5EV)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XCZU5EV	FBVB900	X0Y0	GTH_Quad_226	GTH_Quad_225, GTH_Quad_224	GTH_Quad_223
		X0Y1	GTH_Quad_226	GTH_Quad_225, GTH_Quad_224	GTH_Quad_223
	SFVC784	X0Y0	不支持 X16	不支持 X8	GTH_Quad_224
		X0Y1	不支持 X16	不支持 X8	GTH_Quad_224

表 129: Zynq UltraScale+ 器件可用的 GT 四通道 (XCZU7EV)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XCZU7EV	FBVB900	X0Y0	GTH_Quad_227	GTH_Quad_226, GTH_Quad_225	GTH_Quad_224
		X0Y1	GTH_Quad_227	GTH_Quad_226, GTH_Quad_225	GTH_Quad_224
	FFVC1156	X0Y0	GTH_Quad_227, GTH_Quad_226	GTH_Quad_225, GTH_Quad_224	GTH_Quad_223
		X0Y1	GTH_Quad_227, GTH_Quad_226	GTH_Quad_225, GTH_Quad_224	GTH_Quad_223
	FFVF1517	X0Y0	GTH_Quad_227, GTH_Quad_226	GTH_Quad_225, GTH_Quad_224	GTH_Quad_223
		X0Y1	GTH_Quad_227, GTH_Quad_226	GTH_Quad_225, GTH_Quad_224	GTH_Quad_223

表 130: Zynq UltraScale+ 器件可用的 GT 四通道 (XCZU21DR)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XCZU21DR	FFVD1156	X0Y0	GTY_Quad_131	GTY_Quad_130, GTY_Quad_129	GTY_Quad_128
		X0Y1	GTY_Quad_131	GTY_Quad_130, GTY_Quad_129	GTY_Quad_128

表 131: Zynq UltraScale+ 器件可用的 GT 四通道 (XCZU25DR)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XCZU25DR	FFVE1156	X0Y0	不支持 X16	GTY_Quad_129	GTY_Quad_128
	FFVG1517	X0Y0	不支持 X16	GTY_Quad_129	GTY_Quad_128
	FSVE1156	X0Y0	不支持 X16	GTY_Quad_129	GTY_Quad_128
	FSVG1517	X0Y0	不支持 X16	GTY_Quad_129	GTY_Quad_128

表 132: Zynq UltraScale+ 器件可用的 GT 四通道 (XCZU27DR)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XCZU27DR	FFVG1517	X0Y0	GTY_Quad_131	GTY_Quad_130, GTY_Quad_129	GTY_Quad_128
		X0Y1	GTY_Quad_131	GTY_Quad_130, GTY_Quad_129	GTY_Quad_128
	FFVE1156	X0Y0	不支持 X16	GTY_Quad_129	GTY_Quad_128
		X0Y1	不支持 X16	GTY_Quad_129	GTY_Quad_128
	FSVG1517	X0Y0	GTY_Quad_131	GTY_Quad_130, GTY_Quad_129	GTY_Quad_128
		X0Y1	GTY_Quad_131	GTY_Quad_130, GTY_Quad_129	GTY_Quad_128
	FSVE1156	X0Y0	不支持 X16	GTY_Quad_129	GTY_Quad_128
		X0Y1	不支持 X16	GTY_Quad_129	GTY_Quad_128

表 133: Zynq UltraScale+ 器件可用的 GT 四通道 (XCZU29DR)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XCZU29DR	FFVF1760	X0Y1	GTY_Quad_131	GTY_Quad_130, GTY_Quad_129	GTY_Quad_128
		X0Y0	GTY_Quad_131	GTY_Quad_130, GTY_Quad_129	GTY_Quad_128
	FSVF1760	X0Y1	GTY_Quad_131	GTY_Quad_130, GTY_Quad_129	GTY_Quad_128
		X0Y0	GTY_Quad_131	GTY_Quad_130, GTY_Quad_129	GTY_Quad_128

表 134: Zynq UltraScale+ 器件可用的 GT 四通道 (XCZU28DR)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XCZU28DR	FFVG1517	X0Y0	GTY_Quad_131	GTY_Quad_130, GTY_Quad_129	GTY_Quad_128
		X0Y1	GTY_Quad_131	GTY_Quad_130, GTY_Quad_129	GTY_Quad_128
	FFVE1156	X0Y0	不支持 X16	GTY_Quad_129	GTY_Quad_128
		X0Y1	不支持 X16	GTY_Quad_129	GTY_Quad_128
	FSVG1517	X0Y0	GTY_Quad_131	GTY_Quad_130, GTY_Quad_129	GTY_Quad_128
		X0Y1	GTY_Quad_131	GTY_Quad_130, GTY_Quad_129	GTY_Quad_128
	FSVE1156	X0Y0	不支持 X16	GTY_Quad_129	GTY_Quad_128
		X0Y1	不支持 X16	GTY_Quad_129	GTY_Quad_128

表 135: Zynq UltraScale+ 器件可用的 GT 四通道 (XCZU4CG)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XCZU4CG	FBVB900	X0Y0	GTH_Quad_226	GTH_Quad_225, GTH_Quad_224	GTH_Quad_223
		X0Y1	GTH_Quad_226	GTH_Quad_225, GTH_Quad_224	GTH_Quad_223
	SFVC784	X0Y0	不支持 X16	不支持 X8	GTH_Quad_224
		X0Y1	不支持 X16	不支持 X8	GTH_Quad_224

表 136: Zynq UltraScale+ 器件可用的 GT 四通道 (XCZU5CG)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XCZU5CG	FBVB900	X0Y0	GTH_Quad_226	GTH_Quad_225, GTH_Quad_224	GTH_Quad_223
		X0Y1	GTH_Quad_226	GTH_Quad_225, GTH_Quad_224	GTH_Quad_223
	SFVC784	X0Y0	不支持 X16	不支持 X8	GTH_Quad_224
		X0Y1	不支持 X16	不支持 X8	GTH_Quad_224

表 137: Zynq UltraScale+ 器件可用的 GT 四通道 (XCZU7CG)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XCZU7CG	FBVB900	X0Y0	GTH_Quad_227	GTH_Quad_226, GTH_Quad_225	GTH_Quad_224
		X0Y1	GTH_Quad_227	GTH_Quad_226, GTH_Quad_225	GTH_Quad_224
	FFVC1156	X0Y0	GTH_Quad_227, GTH_Quad_226	GTH_Quad_225, GTH_Quad_224	GTH_Quad_223
		X0Y1	GTH_Quad_227, GTH_Quad_226	GTH_Quad_225, GTH_Quad_224	GTH_Quad_223
	FFVF1517	X0Y0	GTH_Quad_227, GTH_Quad_226	GTH_Quad_225, GTH_Quad_224	GTH_Quad_223
		X0Y1	GTH_Quad_227, GTH_Quad_226	GTH_Quad_225, GTH_Quad_224	GTH_Quad_223

表 138: Zynq UltraScale+ 器件可用的 GT 四通道 (XCZU4EG)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XCZU4EG	FBVB900	X0Y0	GTH_Quad_226	GTH_Quad_225, GTH_Quad_224	GTH_Quad_223
		X0Y1	GTH_Quad_226	GTH_Quad_225, GTH_Quad_224	GTH_Quad_223
	SFVC784	X0Y0	不支持 X16	不支持 X8	GTH_Quad_224
		X0Y1	不支持 X16	不支持 X8	GTH_Quad_224

表 139: Zynq UltraScale+ 器件可用的 GT 四通道 (XCZU5EG)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XCZU5EG	FBVB900	X0Y0	GTH_Quad_226	GTH_Quad_225, GTH_Quad_224	GTH_Quad_223
		X0Y1	GTH_Quad_226	GTH_Quad_225, GTH_Quad_224	GTH_Quad_223
	SFVC784	X0Y0	不支持 X16	不支持 X8	GTH_Quad_224
		X0Y1	不支持 X16	不支持 X8	GTH_Quad_224

表 140: Zynq UltraScale+ 器件可用的 GT 四通道 (XCZU7EG)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XCZU7EG	FBVB900	X0Y0	GTH_Quad_227	GTH_Quad_226, GTH_Quad_225	GTH_Quad_224
		X0Y1	GTH_Quad_227	GTH_Quad_226, GTH_Quad_225	GTH_Quad_224
	FFVC1156	X0Y0	GTH_Quad_227, GTH_Quad_226	GTH_Quad_225, GTH_Quad_224	GTH_Quad_223
		X0Y1	GTH_Quad_227, GTH_Quad_226	GTH_Quad_225 GTH_Quad_224	GTH_Quad_223
	FFVF1517	X0Y0	GTH_Quad_227, GTH_Quad_226	GTH_Quad_225, GTH_Quad_224	GTH_Quad_223
		X0Y1	GTH_Quad_227, GTH_Quad_226	GTH_Quad_225, GTH_Quad_224	GTH_Quad_223

表 141: Zynq UltraScale+ 器件可用的 GT 四通道 (XAZU4EV)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XAZU4EV	SFVC784	X0Y0	不支持 X16	不支持 X8	GTH_Quad_224
		X0Y1	不支持 X16	不支持 X8	GTH_Quad_224

表 142: Zynq UltraScale+ 器件可用的 GT 四通道 (XAZU5EV)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XAZU5EV	SFVC784	X0Y0	不支持 X16	不支持 X8	GTH_Quad_224
		X0Y1	不支持 X16	不支持 X8	GTH_Quad_224

表 143: Zynq UltraScale+ 器件可用的 GT 四通道 (XQZU19EG)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XQZU19EG	FFRB1517	X1Y0	GTH_Quad_227	GTH_Quad_226, GTH_Quad_225	GTH_Quad_224
		X1Y1	不支持 X16	GTH_Quad_227	GTH_Quad_226
		X1Y2	不支持 X16	不支持 X8	GTH_Quad_227
	FFRC1760	X0Y3	GTY_Quad_131	GTY_Quad_130, GTY_Quad_129	GTY_Quad_128
		X0Y2	不支持 X16	GTY_Quad_130, GTY_Quad_129	GTY_Quad_128
		X1Y0	GTH_Quad_227	GTH_Quad_226, GTH_Quad_225	GTH_Quad_224
		X1Y1	GTH_Quad_230, GTH_Quad_229	GTH_Quad_228, GTH_Quad_227	GTH_Quad_226
		X1Y2	GTH_Quad_231, GTH_Quad_230	GTH_Quad_229, GTH_Quad_228	GTH_Quad_227

表 144: Zynq UltraScale+ 器件可用的 GT 四通道 (XQZU5EV)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XQZU5EV	FFRB900	X0Y0	GTH_Quad_226	GTH_Quad_225, GTH_Quad_224	GTH_Quad_223
		X0Y1	GTH_Quad_226	GTH_Quad_225, GTH_Quad_224	GTH_Quad_223
	SFRC784	X0Y0	不支持 X16	不支持 X8	GTH_Quad_224
		X0Y1	不支持 X16	不支持 X8	GTH_Quad_224

表 145: Zynq UltraScale+ 器件可用的 GT 四通道 (XQZU7EV)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XQZU7EV	FFRB900	X0Y0	GTH_Quad_227	GTH_Quad_226, GTH_Quad_225	GTH_Quad_224
		X0Y1	GTH_Quad_227	GTH_Quad_226, GTH_Quad_225	GTH_Quad_224
	FFRC1156	X0Y0	GTH_Quad_227, GTH_Quad_226	GTH_Quad_225, GTH_Quad_224	GTH_Quad_223
		X0Y1	GTH_Quad_227, GTH_Quad_226	GTH_Quad_225, GTH_Quad_224	GTH_Quad_223

表 146: Zynq UltraScale+ 器件可用的 GT 四通道 (XQZU21DR)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XQZU21DR	FFRD1156	X0Y0	GTY_Quad_131	GTY_Quad_130, GTY_Quad_129	GTY_Quad_128
		X0Y1	GTY_Quad_131	GTY_Quad_130, GTY_Quad_129	GTY_Quad_128

表 147: Zynq UltraScale+ 器件可用的 GT 四通道 (XQZU28DR)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XQZU28DR	FFRE1156	X0Y0	不支持 X16	GTY_Quad_129	GTY_Quad_128
		X0Y1	不支持 X16	GTY_Quad_129	GTY_Quad_128
	FFRG1517	X0Y0			
		X0Y1			

表 148: Zynq UltraScale+ 器件可用的 GT 四通道 (XCZU39DR)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XCZU39DR	FFVF1760	PCIE4_X0Y0	GTY_Quad_131	GTY_Quad_129, GTY_Quad_130	GTY_Quad_128
		PCIE4_X0Y1	GTY_Quad_131	GTY_Quad_129, GTY_Quad_130	GTY_Quad_128
	FSVF1760	PCIE4_X0Y0	GTY_Quad_131	GTY_Quad_129, GTY_Quad_130	GTY_Quad_128
		PCIE4_X0Y1	GTY_Quad_131	GTY_Quad_129, GTY_Quad_130	GTY_Quad_128

表 149: Zynq UltraScale+ 器件可用的 GT 四通道 (XCZU46DR)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XCZU46DR	FFVH1760	PCIE4C_X0Y0	GTY_Quad_131	GTY_Quad_129, GTY_Quad_130	GTY_Quad_128
		PCIE4C_X0Y1	GTY_Quad_131	GTY_Quad_129, GTY_Quad_130	GTY_Quad_128
	FSVH1760	PCIE4C_X0Y0	GTY_Quad_131	GTY_Quad_129, GTY_Quad_130	GTY_Quad_128
		PCIE4C_X0Y1	GTY_Quad_131	GTY_Quad_129, GTY_Quad_130	GTY_Quad_128

表 150: Zynq UltraScale+ 器件可用的 GT 四通道 (XCZU47DR)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XCZU47DR	FFVE1156	PCIE4C_X0Y0	不支持 X16	GTY_Quad_129	GTY_Quad_128
		PCIE4C_X0Y1	不支持 X16	GTY_Quad_129	GTY_Quad_128
	FSVE1156	PCIE4C_X0Y0	不支持 X16	GTY_Quad_129	GTY_Quad_128
		PCIE4C_X0Y1	不支持 X16	GTY_Quad_129	GTY_Quad_128
	FFVG1517	PCIE4C_X0Y0	GTY_Quad_131	GTY_Quad_129, GTY_Quad_130	GTY_Quad_128
		PCIE4C_X0Y1	GTY_Quad_131	GTY_Quad_129, GTY_Quad_130	GTY_Quad_128
	FSVG1517	PCIE4C_X0Y0	GTY_Quad_131	GTY_Quad_129, GTY_Quad_130	GTY_Quad_128
		PCIE4C_X0Y1	GTY_Quad_131	GTY_Quad_129, GTY_Quad_130	GTY_Quad_128

表 151: Zynq UltraScale+ 器件可用的 GT 四通道 (XCZU48DR)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XCZU48DR	FFVE1156	PCIE4C_X0Y0	不支持 X16	GTY_Quad_129	GTY_Quad_128
		PCIE4C_X0Y1	不支持 X16	GTY_Quad_129	GTY_Quad_128
	FSVE1156	PCIE4C_X0Y0	不支持 X16	GTY_Quad_129	GTY_Quad_128
		PCIE4C_X0Y1	不支持 X16	GTY_Quad_129	GTY_Quad_128
	FFVG1517	PCIE4C_X0Y0	GTY_Quad_131	GTY_Quad_129, GTY_Quad_130	GTY_Quad_128
		PCIE4C_X0Y1	GTY_Quad_131	GTY_Quad_129, GTY_Quad_130	GTY_Quad_128
	FSVG1517	PCIE4C_X0Y0	GTY_Quad_131	GTY_Quad_129, GTY_Quad_130	GTY_Quad_128
		PCIE4C_X0Y1	GTY_Quad_131	GTY_Quad_129, GTY_Quad_130	GTY_Quad_128

表 152: Zynq UltraScale+ 器件可用的 GT 四通道 (XCZU49DR)

器件	封装	PCIE 块	支持最大链路宽度 X16 的四通道	支持最大链路宽度 X8 的四通道	支持最大链路宽度 X4 的四通道
XCZU49DR	FFVF1760	PCIE4C_X0Y0	GTY_Qaud_131	GTY_Qaud_129, GTY_Qaud_130	GTY_Qaud_128
		PCIE4C_X0Y1	GTY_Qaud_131	GTY_Qaud_129, GTY_Qaud_130	GTY_Qaud_128
	FSVF1760	PCIE4C_X0Y0	GTY_Qaud_131	GTY_Qaud_129, GTY_Qaud_130	GTY_Qaud_128
		PCIE4C_X0Y1	GTY_Qaud_131	GTY_Qaud_129, GTY_Qaud_130	GTY_Qaud_128

调试

本附录包含有关赛灵思支持网站和调试工具上可用资源的详细信息。

在 Xilinx.com 上寻求帮助

为了帮助您在使用核时完成设计和调试进程，[赛灵思技术支持网页](#)上提供了大量关键资源，如产品文档、版本说明、答复记录、已知问题相关信息以及如何获取进一步产品支持的链接。[赛灵思社区论坛](#)还可供会员学习、参与、分享和提出与赛灵思解决方案相关的问题。

文档

本产品指南是与该核相关的主要文档。本指南以及有助于设计进程的所有产品相关文档都可以在[赛灵思技术支持网页](#)上找到，也可以通过赛灵思 Documentation Navigator 来获取。要下载赛灵思 Documentation Navigator，请访问[下载页面](#)。如需了解此工具和可用功能的详细信息，请在安装后打开联机帮助。

调试指南

如需了解有关 PCIe 调试的更多信息，请参阅 [PCIe 调试卡诺图](#)。

答复记录

答复记录包括有关常见问题的信息、有关如何解决这些问题的实用信息以及有关赛灵思产品的所有已知问题。我们每天都会创建和维护答复记录，确保用户可以获取最准确的信息。

您可以通过[赛灵思技术支持网页](#)（主页）上的“搜索支持”框找到对应该核的答复记录。要最大程度扩展搜索结果范围，请使用关键字，例如：

- 产品名称
- 工具消息
- 所遇到问题的摘要

返回结果后，可以使用过滤器搜索来进一步定位结果。

Integrated Block for PCIe 的主答复记录

答复记录 [65751](#)。

技术支持

赛灵思在[赛灵思社区论坛](#)上为此 LogiCORE™ IP 产品提供技术支持，前提是用户按产品文档中所述方式使用该产品。如果您执行以下任何操作，则赛灵思无法保证产品时序和功能的正常运行，也无法保证提供相应支持：

- 在文档中未定义的器件中实现解决方案。
- 超出产品文档中允许的范围自定义解决方案。
- 更改设计中任何标记有“DO NOT MODIFY”的部分。

如需提问，请导航至[赛灵思社区论坛](#)。

硬件调试

集成调试选项

UltraScale+ 器件 Integrated Block PCIe 核随附有 3 个调试选项，集成在核内：

- Viz
- JTAG 调试器
- In-System IBERT 和解扰器模式（适用于 Gen3 链路速度）

这些选项位于 Vivado IP 自定义页面内。请参阅赛灵思答复记录 [68134](#)，以获取有关这些调试选项的详细描述。

收发器控制端口和状态端口

下表描述了用于调试收发器相关问题的端口。



重要提示！ 收发器控制和状态接口中的端口必须根据相应的 GT 用户指南来驱动。使用下表中所列的输入信号可能导致 IP 核出现不可预测的行为。

表 153：用于收发器调试的端口

端口	I/O	宽度	描述
gt_pcieuerratedone	输入	1	连接到收发器通道原语上的 PCIEUSERRATEDONE
gt_loopback	输入	3	连接到收发器通道原语上的 LOOPBACK
gt_txprbsforceerr	输入	1	连接到收发器通道原语上的 TXPRBSFORCEERR
gt_txinhibit	输入	1	连接到收发器通道原语上的 TXINHIBIT
gt_txprbssel	输入	4	PRBS 输入
gt_rxprbssel	输入	4	PRBS 输入
gt_rxprbscntreset	输入	1	连接到收发器通道原语上的 RXPRBSCNTRESET
gt_txelecidle	输出	1	连接到收发器通道原语上的 TXELECIDLE
gt_txresetdone	输出	1	连接到收发器通道原语上的 TXRESETDONE
gt_rxresetdone	输出	1	连接到收发器通道原语上的 RXRECLKOUT
gt_rxpmaresetdone	输出	1	连接到收发器通道原语上的 TXPMARESETDONE

表 153: 用于收发器调试的端口 (续)

端口	I/O	宽度	描述
gt_txphaligndone	输出	1	连接到收发器通道原语的 TXPHALIGNDONE
gt_txphinitdone	输出	1	连接到收发器通道原语的 TXPHINITDONE
gt_txdlysresetdone	输出	1	连接到收发器通道原语的 TXDLYSRESETDONE
gt_rxphaligndone	输出	1	连接到收发器通道原语的 RXPHALIGNDONE
gt_rxdlysresetdone	输出	1	连接到收发器通道原语的 RXDLYSRESETDONE
gt_rxsyncdone	输出	1	连接到收发器通道原语的 RXSYNCDONE
gt_eyescandataerror	输出	1	连接到收发器通道原语上的 EYESCANDATAERROR
gt_rxprbserr	输出	1	连接到收发器通道原语上的 RXPRBSERR
gt_dmonitorout	输出	16	连接到收发器通道原语上的 DMONITOROUT
gt_rxcommadet	输出	1	连接到收发器通道原语上的 RXCOMMADETEN
gt_phystatus	输出	1	连接到收发器通道原语上的 PHYSTATUS
gt_rxvalid	输出	1	连接到收发器通道原语上的 RXVALID
gt_rxcdrlock	输出	1	连接到收发器通道原语上的 RXCDRLOCK
gt_pcierateidle	输出	1	连接到收发器通道原语上的 PCIERATEIDLE
gt_pcieuerratestart	输出	1	连接到收发器通道原语上的 PCIEUSERRATESTART
gt_gtpowergood	输出	1	连接到收发器通道原语上的 GTPOWERGOOD
gt_cplllock	输出	1	连接到收发器通道原语上的 CPLLLOCK
gt_rxoutclk	输出	1	连接到收发器通道原语上的 RXOUTCLK
gt_rxrecclkout	输出	1	连接到收发器通道原语上的 RXRECCLKOUT
gt_qpll1lock	输出	1	连接到收发器公用原语上的 QPLL1LOCK
gt_rxstatus	输出	3	连接到收发器通道原语上的 RXSTATUS
gt_rxbufstatus	输出	3	连接到收发器通道原语上的 RXBUFSTATUS
gt_bufgtdiv	输出	9	连接到收发器通道原语上的 BUFGTDIV
phy_txeq_ctrl	输出	2	PHY TX 均衡控制位
phy_txeq_preset	输出	4	PHY TX 均衡预置位
phy_RST_fsm	输出	4	PHY RST FSM 状态位
phy_txeq_fsm	输出	3	PHY RX 均衡 FSM 状态位 (Gen3)
phy_rxreq_fsm	输出	3	PHY TX 均衡 FSM 状态位 (Gen3)
phy_RST_IDLE	输出	1	PHY 处于空闲 (IDLE) 状态
phy_rrst_n	输出	1	同步复位 (由 sys_clk 生成)
phy_prst_n	输出	1	同步复位 (由 pipe_clk 生成)
gt_gen34_eios_det	输出	1	连接到 gtwizard IP 的 rxctrl0_out 端口
gt_txoutclk	输出	1	连接到收发器通道的 TXOUTCLK
gt_txoutclkfabric	输出	1	连接到收发器通道的 TXOUTCLKFABRIC
gt_rxoutclkfabric	输出	1	连接到收发器通道的 RXOUTCLKFABRIC
gt_txoutclkpcs	输出	1	连接到收发器通道的 TXOUTCLKPCS
gt_rxoutclkpcs	输出	1	连接到收发器通道的 RXOUTCLKPCS
gt_txpmareset	输入	1	连接到收发器通道的 TXPMARESET
gt_rxpmareset	输入	1	连接到收发器通道的 RXPMARESET

表 153: 用于收发器调试的端口 (续)

端口	I/O	宽度	描述
gt_txpcreset	输入	1	连接到收发器通道的 TXPCSRESET
gt_rxpcreset	输入	1	连接到收发器通道的 RXPCSRESET
gt_rxbufreset	输入	1	连接到收发器通道的 RXBUFRESET
gt_rxcdrreset	输入	1	连接到收发器通道的 RXDRRESET
gt_rxdfelpmreset	输入	1	连接到收发器通道的 RXDFELPMRESET
gt_txprogdivresetdone	输出	1	连接到收发器通道的 TXPROGDIVRESETDONE
gt_txpmaresetdone	输出	1	连接到收发器通道的 TXPMARESETDONE
gt_txsyncdone	输出	1	连接到收发器通道的 TXSYNCDONE
gt_rxprbslocked	输出	1	连接到收发器通道的 RXPRBSLOCKED
gt_dmonfiforeset	输入	1	连接到收发器通道的 DMONFIFORESET
gt_dmonitorclk	输入	1	连接到收发器通道的 DMONITORCLK
gt_qpll0lock	输出	1	连接到收发器通道的 QPLL0LOCK

PCIe DRP 端口

下表列出了选中“PCIe DRP Ports”选项时可用的信号。

表 154: PCIe DRP 端口

名称	I/O	宽度	描述
drp_addr	输入	10 位	PCIe DRP 地址
drp_en	输入	1 位	PCIe DRP 使能
drp_di	输入	16 位	PCIe DRP 数据输入
drp_do	输出	16 位	PCIe DRP 数据输出
drp_rdy	输出	1 位	PCIe DRP 就绪
drp_we	输入	1 位	PCIe DRP 写入/读取

GT DRP 端口

下表列出了启用 GT 通道 DRP 参数时可用的信号。

表 155: GT DRP 端口

名称	I/O	宽度	描述
ext_ch_gt_drpaddr	输入	通道数 x 10	GT Wizard DRP 地址
ext_ch_gt_drpen	输入	通道数 x 1	GT Wizard DRP 使能
ext_ch_gt_drpdi	输入	通道数 x 16	GT Wizard DRP 数据输入
ext_ch_gt_drpdo	输出	通道数 x 16	GT Wizard DRP 数据输出
ext_ch_gt_drprd	输出	通道数 x 1	GT Wizard DRP 就绪
ext_ch_gt_drpwe	输入	通道数 x 1	GT Wizard DRP 写入/读取

使用赛灵思虚拟线缆进行调试

赛灵思虚拟线缆 (XVC) 允许 Vivado® Design Suite 通过非 JTAG 接口连接到 FPGA 调试核。标准 Vivado® Design Suite 调试功能使用 JTAG 连接到物理硬件 FPGA 资源并通过 Vivado 执行调试。本章节主要聚焦如何使用 XVC 通过 PCIe® 链路而不是标准 JTAG 调试接口来执行调试。此过程称为 XVC-over-PCIe，支持 Vivado ILA 波形捕获、VIO 调试控制以及使用 PCIe 链路作为信道与其它赛灵思调试核进行交互。

当 JTAG 调试不可用时，应使用 Vivado Design Suite 调试功能通过 XVC-over-PCIe 来远程执行 FPGA 调试。此方法常用于数据中心应用，在此类应用中，FPGA 连接到 PCIe 主机系统，且不连接到任何其它硬件器件。

使用通过 XVC 进行调试的方法需要软件、驱动程序和 FPGA 硬件设计组件。由于 XVC-over-PCIe 调试涉及 FPGA 硬件设计组件，因此要执行调试，需满足以下条件：FPGA 已完成加载、其中具备 FPGA 硬件设计用于实现 XVC-over-PCIe，并且已建立到主机 PC 的 PCIe 链路。通常要完成这些操作，需先将启用 XVC-over-PCIe 的设计加载到板上的配置闪存中，然后再将卡插入数据中心位置。由于使用 XVC-over-PCIe 进行调试与 PCIe 信道有关，因此，此方法不应用于调试 PCIe 链路相关问题。



重要提示！ XVC 仅提供到 FPGA 内部调试核的连接。它不提供器件编程或访问器件 JTAG 和配置寄存器的功能。这些操作可通过其它标准赛灵思接口或外设（如 PCIe MCAP VSEC 和 HWICAP IP）来执行。

概述

支持 XVC-over-PCIe 调试的主要组件包括：

- 主机 PC XVC-Server 应用
- 主机 PC PCIe-XVC 驱动程序
- 启用 XVC-over-PCIe 的 FPGA 设计

这些组件可作为参考，以供演示如何为赛灵思 FPGA 设计创建 XVC 连接。这 3 个组件如下图所示，并通过 TCP/IP 套接字连接到 Vivado Design Suite 调试功能。

图 143：XVC-over-PCIe 软件和硬件组件



X18837-090522

主机 PC XVC-Server 应用

使用调试功能时，`hw_server` 应用由 Vivado Design Suite 启动。您可通过 Vivado IDE 将 `hw_server` 连接到本地或远程 FPGA 目标。此接口同样可用于连接到本地或远程 PCIe-XVC 目标。主机 PCIe XVC-Server 应用使用 TCP/IP 套接字连接到赛灵思 `hw_server`。这样即可允许 Vivado（使用 `hw_server`）和 XVC-Server 应用在同一台 PC 上运行或者在通过以太网连接的不同 PC 上运行。XVC-Server 应用需在直接连接到 FPGA 硬件资源的 PC 上运行。在此情况下，FPGA 硬件通过 PCIe® 连接到主机 PC。XVC-Server 应用通过同样在主机 PC 上运行的 PCIe-XVC 驱动程序来连接到 FPGA 硬件器件。

主机 PC XVC-over-PCIe 驱动程序

XVC-over-PCIe 驱动程序可提供与连接到主机 PC 并启用 PCIe 的 FPGA 硬件资源的连接。因此，此驱动程序作为 Linux 内核模式驱动程序提供，用于访问位于以下位置的 PCIe 硬件器件：`<Vivado_Installation_Path>/data/xicom/driver/pcie/xvc_pcie.zip`。此驱动程序的必要组件必须添加到为特定 FPGA 平台创建的驱动程序中。此驱动程序用于实现 XVC-Server 应用通过 PCIe 与 FPGA 进行通信所需的基本功能。

启用 XVC-over-PCIe 的 FPGA 设计

传统上，Vivado® 调试是通过 JTAG 来执行的。默认情况下，Vivado 工具自动化可将赛灵思调试核连接到 FPGA 中的 JTAG BSCAN 资源以执行调试。为执行 XVC-over-PCIe 调试，此信息必须通过 PCIe 链路而不是 JTAG 线接口来发射。赛灵思 Debug Bridge IP 支持您将调试网络通过 PCIe 扩展配置接口 (PCIe-XVC-VSEC) 或通过 PCIe BAR 的 AXI4-Lite 存储器映射接口 (AXI-XVC) 连接到 PCIe。

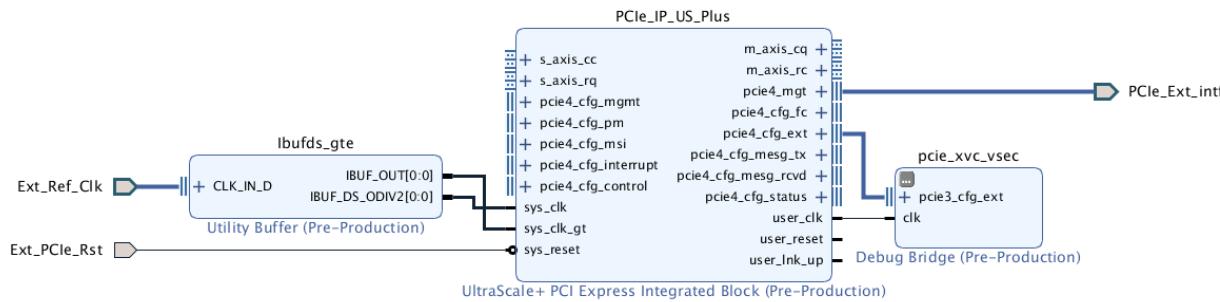
Debug Bridge IP 配置为“From PCIe to BSCAN”或“From AXI to BSCAN”之后即可为分别源于 PCIe 扩展功能或 AXI4-Lite 接口的赛灵思调试网络提供连接点。Vivado 工具自动化可将此 Debug Bridge 实例连接到设计中找到的赛灵思调试核，而不是将其连接到 JTAG BSCAN 接口。对于设计而言，是将 Debug Bridge 连接到 PCIe 扩展配置空间还是将其连接到 AXI4-Lite，这其中各有利弊。以下章节描述了这 2 种实现的实现注意事项和寄存器映射。

通过 PCIe 扩展配置空间的 XVC-over-PCIe (PCIe-XVC-VSEC)

通过使用 PCIe-XVC-VSEC 方法，Debug Bridge IP 即可使用 PCIe 供应商指定扩展功能 (VSEC) 来实现从 PCIe 到 Debug Bridge IP 的连接。PCIe 扩展配置空间设置为可从主机 PC 交付的扩展功能的链接列表。这对于部分平台尤为实用，在此类平台中仅限通过某一版本的设计来实现 PCIe-XVC-VSEC，其它设计实现则无效。此链接列表可用于检测 PCIe-XVC-VSEC 是否存在并予以相应的响应。

PCIe 扩展配置接口使用 PCIe 配置传输事务，而不是 PCIe 存储器 BAR 传输事务。虽然 PCIe 配置传输事务慢得多，但这些事务不会在 PCIe IP 边界处与 PCIe 存储器 BAR 传输事务相互干涉。这样即可在 FPGA 中建立独立的数据和调试通信路径。如果您预计将对数据路径进行调试，那么这是理想的方法。即使数据路径受损或中止，PCIe 扩展配置接口仍可保持运行以执行调试。下图描述了 PCIe IP 与 Debug Bridge IP 之间用于实现 PCIe-XVC-VSEC 的连接。

图 144: 含 PCIe 扩展功能接口的 XVC-over-PCIe

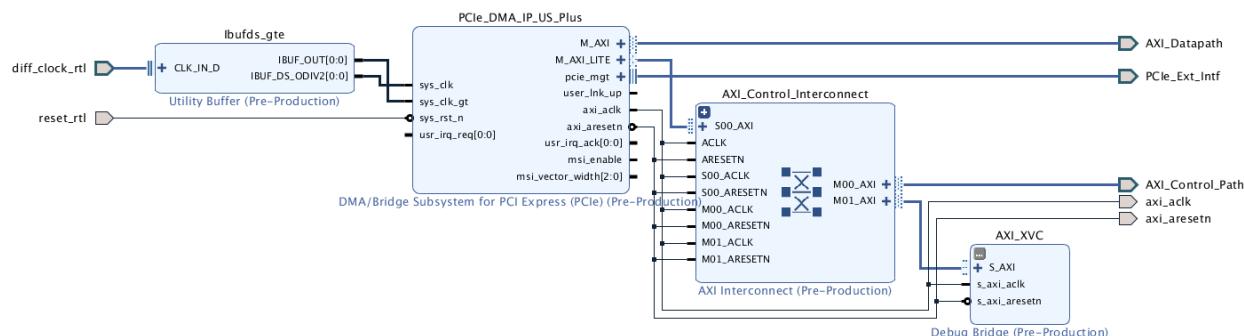


注释: 虽然上图仅显示 UltraScale+™ 器件 Integrated Block for PCIe IP，但其它 PCIe IP (即，UltraScale™ 器件 Integrated Block for PCIe、AXI Bridge for PCIe 或 PCIe DMA IP) 均可在此图中互换使用。

通过 AXI 的 XVC-over-PCIe (AXI-XVC)

通过使用 AXI-XVC 方法，Debug Bridge IP 即可通过 AXI Interconnect IP 连接至 PCIe IP。Debug Bridge IP 采用与其它 AXI4-Lite 从接口 IP 相似的方法连接至 AXI Interconnect，并且同样要求为其分配特定的地址范围。原先此配置中的 `debug_bridge` IP 连接到控制路径网络，而不是系统数据路径网络。下图描述了 DMA Subsystem for PCIe IP 与 Debug Bridge IP 之间用于此实现的连接。

图 145: 含 AXI4-Lite 接口的 XVC over PCIe



注释: 虽然上图显示了 PCIe DMA IP，但在此图中，任何支持 AXI 的 PCIe IP 均可互换使用。

AXI-XVC 实现支持更高速的传输事务。但 XVC 调试流量与其它 PCIe 控制路径流量会通过相同的 PCIe 端口进行传递和互连，导致在此路径上执行传输事务调试的难度增加。因此，应使用 AXI-XVC 调试来调试特定的外设或差分 AXI 网络，而不是尝试对与 AXI-XVC 调试通信路径重叠的数据路径进行调试。

XVC-over-PCIe 寄存器映射

PCIe-XVC-VSEC 和 AXI-XVC 包含的寄存器映射稍有不同，在设计 XVC 驱动程序和软件时必须考量其差异。下表中的寄存器映射显示了对应基址的字节偏移。

- PCIe-XVC-VSEC 基址必须位于 PCIe 扩展配置空间的有效范围内。此范围在 Debug Bridge IP 配置内指定。
- AXI-XVC Debug Bridge 的基址是 Debug Bridge IP 的偏移，此偏移在 Vivado 地址编辑器中指定。

下表描述了针对“From PCIe-Ext to BSCAN”模式或“From AXI to BSCAN”模式进行配置时的 Debug Bridge IP 的寄存器映射（作为对应基址的偏移）。

表 156: Debug Bridge 对应 XVC-PCIe-VSEC 寄存器映射

寄存器偏移	寄存器名称	描述	寄存器类型
0x00	PCIe Ext Capability Header	PCIe 定义的供 VSEC 使用的字段。	只读
0x04	PCIe VSEC Header	PCIe 定义的供 VSEC 使用的字段。	只读
0x08	XVC Version Register	IP 版本和功能信息。	只读
0x0C	XVC Shift Length Register	移位长度。	读写
0x10	XVC TMS Register	TMS 数据。	读写
0x14	XVC TDIO Register	TDO/TDI 数据。	读写
0x18	XVC Control Register	通用控制寄存器。	读写
0x1C	XVC Status Register	通用状态寄存器。	只读

表 157: Debug Bridge 对应 AXI-XVC 寄存器映射

寄存器偏移	寄存器名称	描述	寄存器类型
0x00	XVC Shift Length Register	移位长度。	读写
0x04	XVC TMS Register	TMS 数据。	读写
0x08	XVC TDI Register	TDI 数据。	读写
0x0C	XVC TDO Register	TDO 数据。	只读
0x10	XVC Control Register	通用控制寄存器。	读写
0x14	XVC Status Register	通用状态寄存器。	只读
0x18	XVC Version Register	IP 版本和功能信息。	只读

PCIe Ext Capability Header

此寄存器用于识别添加到 PCIe 设计的 PCIe-XVC-VSEC。PCIe Ext Capability Header 中的字段和值由 PCI-SIG 定义，用于识别扩展功能的格式，并提供指向下一项扩展功能（如果适用）的指针。当用作为 PCIe-XVC-VSEC 时，应先对相应的 PCIe ID 字段求值，然后再进行解读。这些字段包括 PCIe 供应商 ID、PCIe 器件 ID、PCIe 版本 ID、子系统供应商 ID 和子系统 ID。提供的驱动程序会专门检查 PCIe 供应商 ID 与赛灵思 ID (0x10EE) 是否匹配，然后再解读此寄存器。下表描述了此寄存器中的字段。

表 158: PCIe Ext Capability Header 寄存器描述

位的位置	字段	描述	初始值	类型
15:0	PCIe Extended Capability ID	该字段为 PCI-SIG 定义的 ID 号，表示扩展功能 (Extended Capability) 的性质和格式。VSEC 的 Extended Capability ID 为 0x000B	0x000B	只读
19:16	Capability Version	该字段为 PCI-SIG 定义的版本号，表示存在的功能结构的版本。针对此版本的规格，该编号必须为 0x1。	0x1	只读
31:20	Next Capability Offset	该字段从用户传入，包含距离下一个 PCI Express 功能结构的偏移，或者如果已链接的功能列表中没有任何其它项，则该字段为 0x000。对于 PCIe 扩展配置空间内实现的扩展功能，该值必须始终保持在 PCIe 扩展配置空间的有效范围内。	0x000	只读

PCIe VSEC Header (仅限 PCIe-XVC-VSEC)

当 Debug Bridge IP 位于此模式下时，此寄存器用于识别 PCIe-XVC-VSEC。这些字段由 PCI-SIG 定义，但其值则取决于供应商 ID（赛灵思 ID 为 0x10EE）。应先限定 PCIe Ext Capability Header 寄存器值，然后再解读此寄存器。

表 159: PCIe XVC VSEC Header 寄存器描述

位的位置	字段	描述	初始值	类型
15:0	VSEC ID	该字段所表示的 ID 值可用于识别 PCIe-XVC-VSEC，其值取决于供应商 ID（赛灵思 ID 为 0x10EE）。	0x0008	只读
19:16	VSEC Rev	该字段所表示的版本 ID 值可用于识别 PCIe-XVC-VSEC 版本。	0x0	只读
31:20	VSEC Length	该字段表示整个 PCIe-XVC-VSEC 结构中的字节数，包括 PCIe Ext Capability Header 寄存器和 PCIe VSEC Header 寄存器。	0x020	只读

XVC Version Register (仅限 PCIe-XVC-VSEC)

此寄存器是由赛灵思工具填充的，供 Vivado Design Suite 用于识别硬件设计中实现的 Debug Bridge IP 的特定功能。

XVC Shift Length Register

此寄存器用于在调试扫描链内设置扫描链移位长度。

XVC TMS Register

此寄存器用于在调试扫描链中设置 TMS 数据。

XVC TDO/TDI 数据寄存器

此寄存器用于 TDO/TDI 数据访问。使用 PCIePCI-XVC-VSEC 时，这 2 个寄存器组合到单一字段中。使用 AXI-XVC 时，则作为 2 个独立寄存器来实现。

XVC Control Register

此寄存器用于 XVC 控制数据。

XVC Status Register

此寄存器用于 XVC 状态信息。

XVC 驱动程序和软件

在 Vivado Design Suite 安装过程中提供了 XVC 驱动程序和软件示例，位置如下：

<Vivado_Installation_Path>/data/xicom/driver/pcie/xvc_PCIE.zip。在将 XVC 功能集成到赛灵思 FPGA 平台设计驱动程序和软件时，应使用此示例作为参考。所提供的 Linux 内核模式驱动程序和软件可用于为 PCIe-XVC-VSEC 实现和 AXI-XVC Debug Bridge 实现执行 XVC-over-PCIe 调试。

在 PCIe-XVC-VSEC 模式下工作时，此驱动程序将发起 PCIe 配置传输事务，以便与 FPGA 调试网络相连。在 AXI-XVC 模式下工作时，此驱动程序将发起 32 位 PCIe 存储器 BAR 传输事务，以便与 FPGA 调试网络相连。默认情况下，此驱动程序将尝试发现 PCIe-XVC-VSEC，如果在已链接的 PCIe 配置扩展功能列表中未找到 PCIe-XVC-VSEC，则使用 AXI-XVC。

所提供的驱动程序位于 Vivado 安装数据目录中，采用 .zip 文件格式。此 .zip 文件应复制到通过 PCIe 连接至赛灵思 FPGA 的主机 PC 上，并解压以供使用。其中包含 README.txt 文件；请复查这些文件，以获取有关安装和运行 XVC 驱动程序和软件的说明。

有关串联配置设计或 Dynamic Function eXchange 设计的特殊注意事项

串联配置和 Dynamic Function eXchange (DFX) 设计可能具有额外的注意事项需要考量，因为这些流程会将物理资源分区到多个独立区域内。在将调试 IP 添加到设计中（例如，VIO、ILA、MDM 和 MIG-IP）时，应考量使用这些物理分区。专为“From PCIe-ext to BSCAN”或“From AXI to BSCAN”配置的 Debug Bridge IP 只能放置在设计的静态分区。在 DFX 或串联现场更新区域内使用调试 IP 时，应将另一个调试 BSCAN 接口添加到动态区域模块定义中，并在动态区域模块例化过程中使其保持未连接状态。

要将 BSCAN 接口添加到可重配置分区定义中，应将相应的端口和端口属性添加到可重配置分区定义中。以下提供的 Verilog 可用作为将 BSCAN 接口添加到端口声明的模板。

```
...
// BSCAN interface definition and attributes.
// This interface should be added to the DFX module definition
// and left unconnected in the DFX module instantiation.
(* X_INTERFACE_INFO = "xilinx.com:interface:bscan:1.0 S_BSCAN_drck" *)
(* DEBUG="true" *)
input S_BSCAN_drck,
(* X_INTERFACE_INFO = "xilinx.com:interface:bscan:1.0 S_BSCAN_shift" *)
(* DEBUG="true" *)
input S_BSCAN_shift,
(* X_INTERFACE_INFO = "xilinx.com:interface:bscan:1.0 S_BSCAN_tdi" *)
(* DEBUG="true" *)
input S_BSCAN_tdi,
(* X_INTERFACE_INFO = "xilinx.com:interface:bscan:1.0 S_BSCAN_update" *)
(* DEBUG="true" *)
input S_BSCAN_update,
(* X_INTERFACE_INFO = "xilinx.com:interface:bscan:1.0 S_BSCAN_sel" *)
(* DEBUG="true" *)
input S_BSCAN_sel,
(* X_INTERFACE_INFO = "xilinx.com:interface:bscan:1.0 S_BSCAN_tdo" *)
(* DEBUG="true" *)
output S_BSCAN_tdo,
(* X_INTERFACE_INFO = "xilinx.com:interface:bscan:1.0 S_BSCAN_tms" *)
(* DEBUG="true" *)
input S_BSCAN_tms,
(* X_INTERFACE_INFO = "xilinx.com:interface:bscan:1.0 S_BSCAN_tck" *)
(* DEBUG="true" *)
input S_BSCAN_tck,
(* X_INTERFACE_INFO = "xilinx.com:interface:bscan:1.0 S_BSCAN_runtst" *)
(* DEBUG="true" *)
input S_BSCAN_runtst,
(* X_INTERFACE_INFO = "xilinx.com:interface:bscan:1.0 S_BSCAN_reset" *)
(* DEBUG="true" *)
input S_BSCAN_reset,
(* X_INTERFACE_INFO = "xilinx.com:interface:bscan:1.0 S_BSCAN_capture" *)
(* DEBUG="true" *)
```

```
input S_BSCAN_capture,  
(* X_INTERFACE_INFO = "xilinx.com:interface:bscan:1.0 S_BSCAN bscanid_en" *)  
(* DEBUG="true" *)  
input S_BSCAN_bscanid_en,  
....
```

运行 `link_design` 时，公开的端口将通过工具自动化连接到调试网络的静态部分。ILA 也会根据设计需要连接到调试网络。在设计顶层可能还会添加另一个 `dbg_hub` 单元。对于含现场更新的串联 PCIe 设计，`dbg_hub` 和工具插入的时钟缓冲器必须添加到相应的设计分区。以下是完成 `opt_design` 后可运行的 Tcl 命令示例，这些命令用于将 `dbg_hub` 原语与相应的设计分区加以关联。

```
# Add the inserted dbg_hub cell to the appropriate design partition.  
set_property HD.TANDEM_IP_PBLOCK Stage1_Main [get_cells dbg_hub]  
# Add the clock buffer to the appropriate design partition.  
set_property HD.TANDEM_IP_PBLOCK Stage1_Config_IO [get_cells  
dma_pcie_0_support_i/  
pcie_ext_cap_i/vsec_xvc_inst/vsec_xvc_dbg_bridge_inst/inst/bsip/ins  
t/USE_SOFTBSCAN.U_TAP_TCKBUFG]
```

使用 PCIe-XVC-VSEC 设计示例

PCIe-XVC-VSEC 已集成到 PCIe 设计示例中，包含在 UltraScale+™ Integrated Block for PCIe IP 的“Advanced”（高级）设置内。本章节旨在提供相关指示信息，以指导您如何使用 PCIe-XVC-VSEC 生成 PCIe 设计示例，然后使用所提供的 XVC 驱动程序和软件通过 PCIe 对 FPGA 进行调试。此示例是在客户应用中使用 XVC 的示例。FPGA 设计、驱动程序和软件元素都需要集成到客户设计中。

生成 PCIe-XVC-VSEC 设计示例

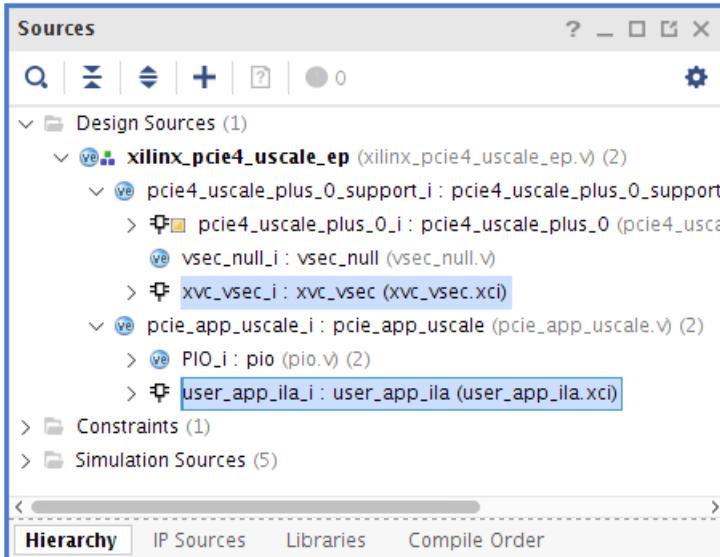
通过选择以下选项，可将 PCIe-XVC-VSEC 添加到 UltraScale+™ PCIe 设计示例。

1. 将核配置为期望的配置。
2. 在“Basic”（基本）选项卡上，选中“Advanced”（高级）模式。
3. 在“Adv. Options-3”（高级选项 3）选项卡上：
 - a. 选中“PCI Express Extended Configuration Space Enable”（PCI Express 扩展配置空间启用）复选框，以启用 PCI Express 扩展配置接口。这样即可将附加扩展功能添加到 PCI Express 核中。
 - b. 选中“Add the PCIe-XVC-VSEC to the Example Design”（将 PCIe-XVC-VSEC 添加到设计示例）复选框，以在设计示例生成过程中启用 PCIe-XVC-VSEC。
4. 验证 PCIe IP 的其它配置选项。以下选项是配置驱动程序用于硬件实现所必需的选项：
 - PCIe Vendor ID（0x10EE 对应赛灵思）
 - PCIe Device ID（取决于用户选择）
5. 单击“OK”（确定）以最终确认选择，并生成 IP。
6. 为应用所需的 IP 生成输出文件。
7. 在“Sources”（源）窗口中，右键单击此 IP 并选中“Open IP Example Design”（打开 IP 设计示例）。
8. 选择用于生成设计示例的目录，然后单击“OK”。

生成后的设计示例会显示：

- PCIe IP 已连接到支持封装文件中的 `xvc_vsec`，并且
- 已向设计的用户应用部分添加了 ILA IP。

此处显示了 FPGA 设计的硬件部分所需的连接。可根据您的应用的实际需求来添加其它调试核。



注释：虽然上图所示仅适用于 UltraScale+ 器件 Integrated Block for PCIe IP，但对于其它 PCIe IP 而言，设计示例层级是相同的。

9. 双击识别为 `xvc_vsec` 的 Debug Bridge IP 即可查看此 IP 的配置选项。记录以下配置参数，因为这些参数将用于配置驱动程序。

- PCIe XVC VSEC ID（默认值为 `0x0008`）
- PCIe XVC VSEC Rev ID（默认值为 `0x0`）



重要提示！ 使用赛灵思供应商 ID 或提供的 XVC 驱动程序和软件时，请勿修改这些参数值。这些值用于检测 XVC 扩展功能。（请参阅 PCIe 规范，以获取其它详细信息。）

10. 在 Flow Navigator 中，单击“Generate Bitstream”（生成比特流）即可为设计示例工程生成比特流。随后，此比特流将被加载到 FPGA 板上，以启用基于 PCIe 的 XVC 调试。

完成基于 PCIe 的 XVC (XVC-over-PCIe) 硬件设计后，可使用已启用 XVC 的相应 PCIe 驱动程序和关联的 XVC-Server 软件应用来将 Vivado Design Suite 连接到 PCIe 连接的 FPGA。Vivado 可连接到本地机器上运行的 XVC-Server 应用，或者也可以使用 TCP/IP 套接字远程连接到其它机器上的应用。

系统初始化

第一步是对系统上的 FPGA 和功耗进行编程，使主机系统可检测到 PCIe 链路。这可通过下列任一方式完成：

- 将设计文件编程到 FPGA 板上存在的闪存中，或者
- 直接通过 JTAG 对器件进行编程。

如果卡由主机 PC 供电，则需要将其上电后才能使用 JTAG 来执行编程，然后重新启动以允许 PCIe 链路执行枚举。系统启动并正常运行后，您可使用 Linux `lspci` 实用工具来列出基于 FPGA 的 PCIe 器件的详细信息。

编译和加载驱动程序

所提供的 PCIe 驱动程序和软件应根据特定平台进行自定义。要完成此操作，所开发的驱动程序和软件通常会先验证供应商 ID、器件 ID、版本 ID、子系统供应商 ID 和子系统 ID，然后再尝试访问器件扩展功能或外设（如 PCIe-XVC-VSEC 或 AXI-XVC）。由于提供的驱动程序为通用型驱动程序，因此它仅验证供应商 ID 和器件 ID 以确认兼容性，然后再尝试验证 PCIe-XVC-VSEC 或 AXI-XVC 外设。

XVC 驱动程序和软件均作为 ZIP 文件提供，包含在 Vivado Design Suite 安装内。

1. 请将此 ZIP 文件从 Vivado 安装目录复制到 FPGA 连接的主机 PC 并解压其内容。此文件位于 Vivado 安装目录内的以下路径中。

```
XVC Driver and SW Path: .../data/xicom/driver/pcie/xvc_pcnie.zip
```

driver_* 和 xvcserver 目录内的 README.txt 文件用于识别如何编译、安装和运行 XVC 驱动程序和软件，详细步骤汇总如下。将驱动程序和软件文件复制到主机 PC 并以具有 root 用户权限的用户身份登录后，请遵循以下步骤进行操作。

2. 修改 driver_*/xvc_pcnie_user_config.h 文件中的变量以匹配您的硬件设计和 IP 设置。请考虑修改以下变量：

- PCIE_VENDOR_ID：PCIe® IP 自定义中定义的 PCIe 供应商 ID。
- PCIE_DEVICE_ID：PCIe® IP 自定义中定义的 PCIe 器件 ID。
- Config_space：允许选择使用 PCIe-XVC-VSEC 或 AXI-XVC 外设。默认值 AUTO 会首先尝试发现 PCIe-XVC-VSEC，然后，如果未找到 PCIe-XVC-VSEC，则会尝试连接至 AXI-XVC 外设。值 CONFIG 或 BAR 可用于在 PCIe®-XVC-VSEC 实现和 AXI-XVC 实现之间明确选择所期望的实现。
- config_vsec_id：当“Bridge Type”（桥接类型）配置为“From PCIE to BSCAN”（从 PCIE 到 BSCAN）时，该值为在“Debug Bridge IP”中定义的 PCIe XVC VSEC ID（默认值 0x0008）。该值仅用于检测 PCIe®-XVC-VSEC。
- config_vsec_rev：当“Bridge Type”配置为“From PCIE to BSCAN”（从 PCIE 到 BSCAN）时，该值为在“Debug Bridge IP”中定义的 PCIe XVC VSEC Rev ID（默认值 0x0）。该值仅用于检测 PCIe-XVC-VSEC。
- bar_index：PCIe BAR 索引，当“Bridge Type”配置为“From AXI to BSCAN”（从 AXI 到 BSCAN）时，应使用此索引值来访问“Debug Bridge IP”。此 BAR 索引指定为 PCIe IP 自定义与系统设计中可寻址的 AXI 外设的组合。该值仅用于检测 AXI-XVC 外设。
- bar_offset：PCIe BAR 偏移，当“Bridge Type”配置为“From AXI to BSCAN”时，应使用此偏移值来访问“Debug Bridge IP”。此 BAR 偏移指定为 PCIe IP 自定义与系统设计中可寻址的 AXI 外设的组合。该值仅用于检测 AXI-XVC 外设。

3. 请将源文件移至您所选的目录内。例如，请使用：

```
/home/username/xil_xvc or /usr/local/src/xil_xvc
```

4. 请确保您具有 root 用户权限，并切换至包含驱动程序文件的目录。

```
# cd /driver_*/
```

5. 编译驱动程序模块：

```
# make install
```

内核模块对象文件将安装为：

```
/lib/modules/[KERNEL_VERSION]/kernel/drivers/pci/pcie/Xilinx/xil_xvc_driver.ko
```

6. 运行 depmod 命令以选择新安装的内核模块：

```
# depmod -a
```

7. 请确保未加载任何旧版本的驱动程序:

```
# modprobe -r xil_xvc_driver
```

8. 加载模块:

```
# modprobe xil_xvc_driver
```

如果运行 dmesg 命令, 您将看到以下消息:

```
kernel: xil_xvc_driver: Starting...
```

注释: 您还可在内核对象文件上使用 insmod 来加载模块:

```
# insmod xil_xvc_driver.ko
```

但由于与旧内核存在兼容性问题, 因此不推荐此方法, 如非必要请勿使用。

9. 生成的字符文件 /dev/xil_xvc/cfg_ioc0 归用户根和组根所有, 并且它需要具备 660 的权限。如果此文件不允许应用与驱动程序进行交互, 请更改其中的权限。

```
# chmod 660 /dev/xil_xvc/cfg_ioc0
```

10. 为驱动程序构建简单的测试程序:

```
# make test
```

11. 运行测试程序:

```
# ./driver-test/verify_xil_xvc_driver
```

您应可看到各项不同长度的测试成功消息, 后接以下消息:

```
"XVC PCIE Driver Verified Successfully!"
```

编译和启动 XVC-Server 应用

XVC-Server 应用可提供 Vivado 硬件服务器与启用 XVC 的 PCIe 器件驱动程序之间的连接。Vivado Design Suite 使用 TCP/IP 连接至 XVC-Server。所需端口号将需要通过您的网络防火墙正常公开。以下步骤可用于使用默认端口号 10200 来编译和启动 XVC 软件应用。

1. 请确保系统上的防火墙设置已公开将用于连接到 Vivado Design Suite 的端口。对于此示例, 使用的是端口 10200。
2. 记录主机名或 IP 地址。需要该主机名和端口号才能将 Vivado 连接到 xvcserver 应用。请参阅操作系统帮助页面以获取有关对应您的操作系统的防火墙端口设置的信息。
3. 请将源文件移至您所选的目录内。例如, 请使用:

```
/home/username/xil_xvc 或 /usr/local/src/xil_xvc
```

4. 切换到包含应用源文件的目录:

```
# cd ./xvcserver/
```

5. 编译该应用:

```
# make
```

6. 启动 XVC-Server 应用：

```
# ./bin/xvc_pcie -s TCP:::10200
```

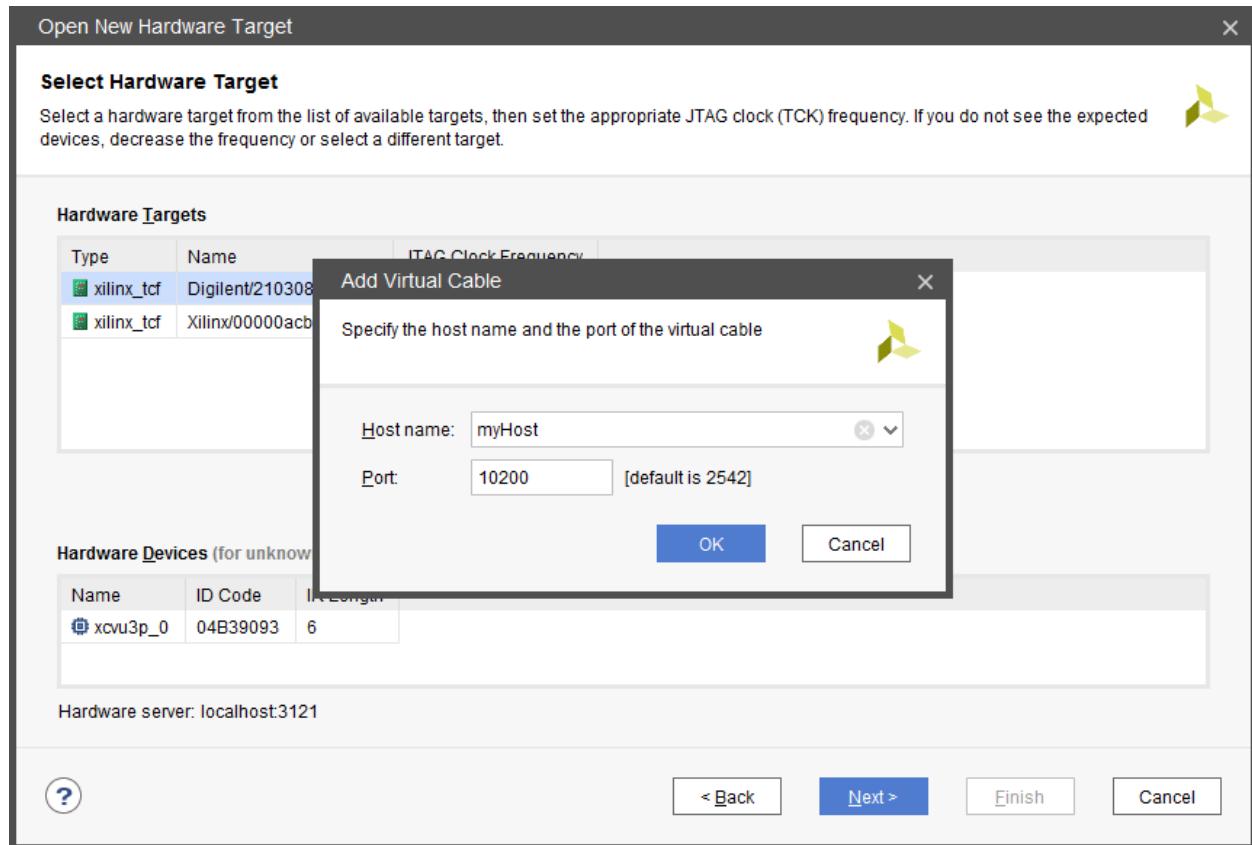
当 Vivado Design Suite 已连接到 XVC-Server 应用后，在 XVC-Server 中应显示以下消息。

```
Enable verbose by setting VERBOSE env var.  
Opening /dev/xil_xvc/cfg_ioc0
```

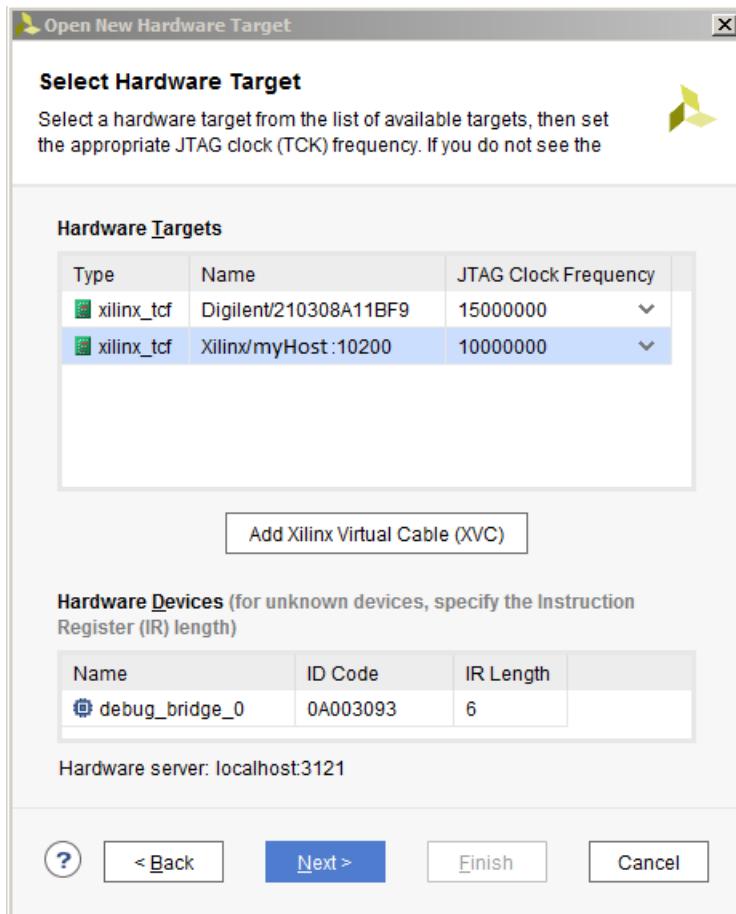
将 Vivado Design Suite 连接到 XVC-Server 应用

Vivado Design Suite 可在运行 XVC-Server 应用的计算机上运行，或者也可以在通过以太网网络连接的另一台计算机上远程运行。但端口必须可供运行 Vivado 的机器访问。要将 Vivado 连接到 XVC-Server 应用，请遵循以下步骤进行操作，这些步骤使用默认端口号。

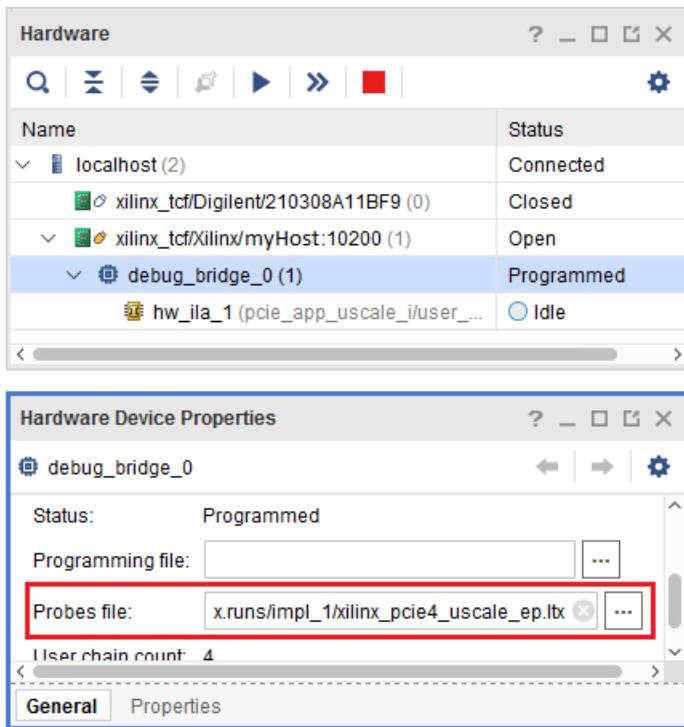
1. 启动 Vivado Design Suite。
 2. 选择“Open HW Manager”（打开硬件管理器）。
 3. 在“Hardware Manager”（硬件管理器）中，选中“Open target”→“Open New Target”（打开目标 > 打开新目标）。
 4. 单击“Next”。
 5. 选择“Local server”（本地服务器），然后单击“Next”（下一步）。
- 这样即可在本地机器上启动 `hw_server`，随后它会连接到 `xvcserver` 应用。
6. 选择“Add Xilinx Virtual Cable (XVC)”（添加赛灵思虚拟线缆 (XVC)）。
 7. 在“Add Virtual Cable”（添加虚拟线缆）对话框中，输入相应的“Host name”（主机名）或“IP address”（IP 地址）和“Port”（端口）以连接到 `xvcserver` 应用。单击“OK”（确定）。



8. 从“Hardware Targets”（硬件目标）表中选择新添加的 XVC 目标，然后单击“Next”（下一步）。



9. 单击“Finish”（完成）。
10. 在“Hardware Device Properties”（硬件器件属性）面板中，选择 Debug Bridge 目标，并指定相应的探测 .ltx 文件。



这样 Vivado 即可识别您的调试核和调试信号，并且您可通过 Vivado 硬件工具接口使用标准调试方法来调试自己的设计。

由此，您即可通过 PCIe 连接而不是通过使用赛灵思虚拟线缆技术的 JTAG 来调试赛灵思 FPGA 设计。您可在 Vivado 中使用右键单击菜单来关闭硬件服务器，以终止连接。如果 PCIe 连接中断或者 XVC-Server 应用停止运行，则与 FPGA 和关联的调试核的连接也将中断。

运行时间注意事项

对器件进行编程时，不应运行 Vivado 到 XVC-Server 应用的连接。XVC-Server 及其与 Vivado 的连接应仅限在器件完成编程且硬件 PCIe 接口处于活动状态时才能启动。

对于 DFX 设计，在 DFX 操作期间终止连接至关重要。在 DFX 操作期间，如果动态区域内部存在调试核，那么应对部分调试树进行重新编程。DFX 操作期间，Vivado 调试工具不应通过 XVC 与 FPGA 主动进行通信。

附加资源与法律声明

赛灵思资源

如需获取答复记录、技术文档、下载以及论坛等支持性资源，请参阅[赛灵思技术支持](#)。

Documentation Navigator 与设计中心

赛灵思 Documentation Navigator (DocNav) 提供了访问赛灵思文档、视频和支持资源的渠道，您可以在其中筛选搜索信息。要打开 DocNav，请执行以下操作：

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

赛灵思设计中心提供了根据设计任务和其它主题整理的文档链接，可供您用于了解关键概念以及常见问题解答。要访问设计中心，请执行以下操作：

- 在 DocNav 中，单击“Design Hubs View”选项卡。
- 在赛灵思网站上，查看[设计中心](#)页面。

注释：如需了解有关 DocNav 的更多信息，请参阅赛灵思网站上的 [Documentation Navigator](#) 页面。

参考资料

以下技术文档是非常实用的补充资料，可配合本指南一起使用：

1. PCI-SIG 文档 (www.pcisig.com/specifications)
2. 《UltraScale 器件 Gen3 Integrated Block for PCI Express LogiCORE IP 产品指南》([PG156](#))
3. 《DMA/Bridge Subsystem for PCI Express 产品指南》([PG195](#))
4. 《Virtex-7 FPGA Integrated Block for PCI Express LogiCORE IP 产品指南》([PG023](#))
5. 《UltraScale 架构配置用户指南》([UG570](#))

6. 《Kintex UltraScale FPGA 数据手册：DC 和 AC 开关特性》(DS892)
7. 《Virtex UltraScale FPGA 数据手册：DC 和 AC 开关特性》(DS893)
8. 《UltraScale 架构 PCB 设计用户指南》(UG583)
9. 《UltraScale 与 UltraScale+ FPGA 封装和管脚分配产品规格》(UG575)
10. 《UltraScale 架构 GTH 收发器用户指南》(UG576)
11. 《UltraScale 架构 GTY 收发器用户指南》(UG578)
12. 《Zynq UltraScale+ 器件封装和管脚分配产品规格用户指南》(UG1075)
13. 《Kintex UltraScale+ FPGA 数据手册：DC 和 AC 开关特性》(DS922)
14. 《Virtex UltraScale+ FPGA 数据手册：DC 和 AC 开关特性》(DS923)
15. 《Versal ACAP Integrated Block for PCI Express LogiCORE IP 产品指南》(PG343)
16. 《Vivado Design Suite 用户指南：采用 IP 进行设计》(UG896)
17. 《Vivado Design Suite 用户指南：采用 IP integrator 设计 IP 子系统》(UG994)
18. 《Vivado Design Suite 用户指南：入门指南》(UG910)
19. 《Vivado Design Suite 用户指南：使用约束》(UG903)
20. 《Vivado Design Suite 用户指南：逻辑仿真》(UG900)
21. 《Vivado Design Suite 用户指南：编程和调试》(UG908)
22. 《Vivado Design Suite 教程：Dynamic Function eXchange》(UG947)
23. 《In-System IBERT LogiCORE IP 产品指南》(PG246)
24. [ATX 电源设计指南](#)
25. 《使用集成端点 PCI Express 块采用 Gen2 x8 和 Gen3 x8 配置进行 PIPE 模式仿真》(XAPP1184)
26. 《AMBA AXI4-Stream 协议规范》(ARM IHI 0051A)

修订历史

下表列出了本文档的修订历史。

章节	修订综述
2022 年 11 月 16 日 1.3 版	
常规更新	整个文档。
配置接口	新增章节。
2022 年 6 月 10 日 1.3 版	
附录 B: 管理传入完成包的接收缓冲器空间	新增附录。
2021 年 12 月 7 日 1.3 版	
可供 PCI Express 使用的集成块、附录 C: GT 位置 和串联回路 支持的器件	更新受支持的器件，包含新的 Artix UltraScale+ 器件支持。
不受支持的 PCI Express 基本规范第 3.1 版功能	对根端口的总线主控制器使能位添加限制。

章节	修订综述
配置流量控制接口	更新 cfg_fc_sel 的值。
活动状态功耗管理	添加有关启用 ASPM L0s/ASPM L1 的注释。
“PF BARs” 选项卡 “SRIOV BARs” 选项卡	添加 “Copy PF0” 选项的描述。
复位	新增澄清。
配置流量控制接口	移除错误的 cfg_*_scale 信号。
时钟和复位接口 所需约束	将 IBUFDs 重命名为 IBFUDS_GTE。
2021 年 4 月 29 日 1.3 版	
串联配置	更新 支持的器件 中的 Zynq® UltraScale+™ RFSoC 器件支持。
	在 多重启动和回退 中添加澄清。
2021 年 2 月 11 日 1.3 版	
串联配置	更新 支持的器件 ，并添加 多重启动和回退 章节。
时钟	更新时钟架构图示。
生成中断请求 和 遗留中断模式	添加澄清信息。
附加调试选项	更新眼图检查步骤。
核约束	添加 “软核逻辑布局” 章节。
2020 年 9 月 22 日 1.3 版	
Virtex UltraScale+ 器件可用的 GT 四通道	更新 Virtex UltraScale+ 器件可用的 GT 四通道。
2020 年 7 月 22 日 1.3 版	
不受支持的 PCI Express 基本规范第 4.0 版功能 (PCIE4C)	将不受支持的内容从表格移至列表中。
完成器请求接口操作（512 位）、避免转发请求发生队头阻塞 和 避免转发请求发生队头阻塞	澄清编辑更新。
串联配置	在全文中将部分重配置更新为 Dynamic Function eXchange。 验证并更新受支持的器件。
Virtex UltraScale+ 器件可用的 GT 四通道 和 Zynq UltraScale+ 器件可用的 GT 四通道	更新受支持的器件和可用四通道。
2019 年 11 月 18 日 1.3 版	
配置空间	新增用户设计扩展配置列表表格。
串联配置	更新受支持的器件表。
	添加 在 Zynq MPSoC 器件上使用串联 PCIe 部分。
遗留中断模式	更新遗留中断信号表。
Virtex UltraScale+ 器件可用的 GT 四通道	更新 XCVU35P 器件和 XCVU37P 器件的 GT 四通道表。
2019 年 6 月 24 日 1.3 版	
不支持的功能	新增对应 PCIe Secure IP 模型（不支持 DRP 接口仿真）的项和 串联 PROM 仿真（不受支持）的项。
串联配置	新增并更新受支持的器件。
64/128/256 位请求器接口 和 512 位请求器接口	将 pcie_rq_tag[5:0] 更改为 pcie_rq_tag[7:0]。
在根端口上启用环回主控制器	新增章节。
集成块端点配置概述	新增有关用于虚拟功能的 FLR 的澄清信息，此类功能在设计示例 中未完全实现。
2018 年 12 月 5 日 1.3 版	
IP 相关信息	新增 PCIE4C 集成块支持详细信息。

章节	修订综述
第 2 章: 概述	新增 PCIE4C 集成块支持详细信息。 新增有关 PCI Express 基本规范 4.0 合规性限制的详细信息。
第 3 章: 产品规格	在对应 PCIE4C 的器件最低要求中新增 Gen4 链路速度。 在“可供 PCI Express 使用的集成块 - Virtex UltraScale+”表中新增 PCIEC 和 PCIE4 块位置。 新增 pcie_rq_tag0[7:0] 和 pcie_rq_tag1[7:0] 用于替代 pcie_rq_tag。 更新 cfg_ext_read_received 描述详细信息。 在“配置接收报文接口”部分中，从“接收报文接口上的报文类型编码”表移除 ATS 报文类型，并移除“接收报文接口上的报文参数”表。
第 5 章: 设计流程步骤	更新“First VF Offset”参数详细信息（在“SRIOV 配置”表中）。
附录 C: GT 位置	为含高带宽存储器 (HBM) 的受支持的 Virtex UltraScale+ 器件新增 PCIE4 和 PCIE4C 块位置。
2018 年 6 月 6 日 1.3 版	
第 2 章: 概述	在“可供 PCI Express 使用的集成块 - Virtex UltraScale+”表中新增器件。
第 3 章: 产品规格	在“配置状态接口端口描述”表中将 error_out 更新为 cfg_local_error_out。
第 4 章: 用核设计	更新“串联 PROM/PCIe 支持的配置”表和“含现场更新的串联 PCIe”部分。 在“时钟设置”部分中新增更多详细信息。 在“通道翻转”部分中新增重要注释。
附录 C: GT 位置	在“Virtex Ultrascale+ 器件可用的 GT 四通道”表中新增器件和封装，并新增“Zynq UltraScale+ 器件可用的 GT 四通道”表。
2018 年 4 月 4 日 1.3 版	
常规更新	新增 PCIE4C 器件最低要求信息。 在“用核设计”章节的“串联 PROM/PCIe 支持的配置”表中更新器件支持。 在“调试”附录中新增“集成调试选项”部分。
2017 年 12 月 20 日 1.3 版	
第 3 章: 产品规格	移除“受支持的器件”表。 更新“完成器请求描述符字段”表中的“目标功能”字段的描述。 更新“请求器完成描述符字段”表中“下位地址”字段的描述。 在“时钟和复位接口”部分中新增有关 user_clk 信号的详细信息。
第 4 章: 用核设计	在“串联配置”部分中新增有关配置 bank 65 的注释。 更新“遗留中断信号”图示和描述。
常规更新	在附录 A “升级”中新增 enable_auto_rreq 参数。 在附录 B “GT 位置”中，内容均为全新内容。
2017 年 10 月 4 日 1.3 版	
第 3 章: 产品规格	更新“可供 PCI Express 使用的集成块 - Virtex UltraScale+”表。 在 m_axis_cq_tuser 名称/描述中更新位索引 87:85 的边带信号描述。 新增 cfg_err_cor_out、cfg_err_nonfatal_out、cfg_err_fatal_out 和 cfg_ds_port_number。 更新 sys_clk_gt 的描述。

章节	修订综述
第 4 章：用核设计	<p>更新“串联配置”部分。</p> <p>更新“时钟设置架构”图示。</p> <p>对应完成器完成描述符字段（位索引 7972、87:80 和 88）以及请求器请求描述符字段（位索引 87:80、95:88 和 120）的描述进行了重大更新。</p> <p>更新“完成错误的处理”部分中的注释。</p>
第 5 章：设计流程步骤	<p>更新以下 Vivado IP 目录选项：</p> <ul style="list-style-type: none">“Extended Tag”字段（位于“Capabilities”选项卡中）。位于“Identity Setting (PF0 IDs and PF1 IDs)”（身份设置（PF0 ID 和 PF1 ID））选项卡中的“PF0 ID Initial Values > Device ID value”（PF0 ID 初始值 > 器件 ID 值）。位于“MSI-X Capabilities”选项卡中的“MSIx Table Settings > Table Size”。位于“SRIOV Config”（SRIOV 配置）选项卡中的“General SRIOV Config”（通用 SRIOV 配置） <p>新增 GT DRP 时钟源（位于“GT Settings”选项卡上）和 GT COMMON 选项（位于“Shared Logic”选项卡上）</p>
常规更新	<p>在附录 A “升级”中新增端口和参数详细信息。</p> <p>在附录 B “GT 位置”中，更新下列表格：Virtex UltraScale+ 器件 GT 位置、Zynq UltraScale+ 器件 GT 位置、Kintex UltraScale+ 器件可用的 GT 四通道和 Zynq Ultrascale+ 器件可用的 GT 四通道</p> <p>在附录 C “调试”中，更新记录的 gt_dmonitorout 端口宽度。</p> <p>在附录 D “使用赛灵思虚拟线缆进行调试”：</p> <ul style="list-style-type: none">新增 XVC 状态寄存器更新“对应 XVC-PCIe-VSEC 寄存器映射的 Debug Bridge”表和“对应 AXI-XVC 寄存器映射的 Debug Bridge”表。
2017 年 6 月 7 日 1.2 版	
常规更新	<p>更新“可供 PCI Express 使用的集成块 - Zynq UltraScale+”表。</p> <p>更新 cfg_interrupt_msi_function_number 的端口描述。</p> <p>更新所有 AXI4-Stream 接口图例中的 TUSER 信号。</p> <p>更新含跨接操作的 512 位 AXI 接口的 TUSER 信号定义。</p> <p>针对“串联配置”部分的次要更新。</p> <p>将 pcie_cq_np_req 更新为 pcie_cq_np_req[0]。</p> <p>新增缺失的“AXISTEN_IF_ENABLE_MSG_ROUTE 属性位描述”表。</p> <p>更新“Zynq UltraScale+ 器件 GT 位置”表。</p> <p>澄清了“使用赛灵思虚拟线缆进行调试”附录中的“XCV 驱动和软件位置示例”。</p>
2017 年 4 月 5 日 1.2 版	
常规更新	<p>新增“使用赛灵思虚拟线缆进行调试”附录。</p> <p>更新“串联配置”部分。</p> <p>更新“升级”附录中的新端口和参数信息。</p>

章节	修订综述
2016 年 11 月 30 日 1.1 版	
第 5 章：设计流程步骤	更新“对应不同器件配置的 SRIOV BAR 大小范围”表。 在“GT 设置”选项卡中新增“PLL 选择”选项和“链路伙伴 TX 预置”选项。 澄清“启用 In-System IBERT”选项和“启用 JTAG Debugger”选项应仅用于硬件调试。针对使用这些选项生成的核，不支持执行仿真。
附录 C: GT 位置	新增“可用 GT 四通道”部分。
2016 年 10 月 5 日 1.1 版	
常规更新	将性能和资源使用情况数据移至网页。 更新“器件最低要求”表。 新增串联配置支持。 新增用于收发器调试的 IBERT 端口、GT DRP 端口和 PCIe 端口，并新增支持这些端口的 Vivado Design Suite 核自定义选项。 在“移植和升级”附录中新增“端口更改”表。 为 Virtex UltraScale+ 器件更新“GT 位置”表和“可供 PCI Express 使用的集成块”表。
2016 年 6 月 8 日 1.1 版	
常规更新	针对根端口模型测试激励文件新增“完成器模型”选项。 新增 MXI-X 中断内部（内置）支持。 新增用于插入损失调整的 GT 设置。 在 Gen2 (5Gb/s) 模式下新增 QPLL1 支持。
2016 年 4 月 6 日 1.1 版	
初始版本。	不适用

请阅读：重要法律声明

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任何形式的损失或赔偿），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、过失或是其它责任认定原理，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：<https://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：<https://china.xilinx.com/legal.htm#tos>。

关于与汽车相关用途的免责声明

如将汽车产品（部件编号中含“XA”字样）用于部署安全气囊或用于影响车辆控制的应用（“安全应用”），除非有符合 ISO 26262 汽车安全标准的安全概念或冗余特性（“安全设计”），否则不在质保范围内。客户应在使用或分销任何包含产品的系统之前为了安全的目的全面地测试此类系统。在未采用安全设计的条件下将产品用于安全应用的所有风险，由客户自行承担，并且仅在适用的法律法规对产品责任另有规定的情况下，适用该等法律法规的规定。

版权声明

© 2016 - 2022 年 Advanced Micro Devices, Inc. 版权所有。Xilinx、赛灵思徽标、Alveo、Artix、Kintex、Kria、Spartan、Versal、Vitis、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或地区的商标。“PCI”、“PCIe”和“PCI Express”均为 PCI-SIG 拥有的商标，且经授权使用。“AMBA”、“AMBA Designer”、“Arm”、“ARM1176JZ-S”、“CoreSight”、“Cortex”、“PrimeCell”、“Mali”和“MPCore”为 Arm Limited 在欧盟及其它国家或地区的注册商标。所有其它商标均为各自所有方所属财产。