QDMA Subsystem for PCI Express

产品指南

Vivado Design Suite

PG302 (v5.0) 2022 年 10 月 27 日

AMD
XILINX

目录

AMD XILINX

第 1 章

简介

赛灵思 QDMA Subsystem for PCI Express (PCIe®) 旨在利用多队列的概念实现高性能 DMA，以搭配 PCI Express® 3.x Integrated Block 一起使用，它与 DMA/Bridge Subsystem for PCI Express 的不同之处在于，后者使用多个赛灵思卡到主机 (C2H) 和主机到卡 (H2C) 通道。

本指南适用于面向 UltraScale+™ 器件的 QDMA Subsystem for PCIe。如需了解有关 Versal® ACAP 子系统的详细信息，请参阅《Versal ACAP DMA and Bridge Subsystem for PCI Express 产品指南》(PG344)。

功能特性

- UltraScale+™ 器件中支持 PCIe Integrated Block（PCIe 集成块）。
- 支持 128、256 和 512 位数据路径。

  注释：不再支持 64 位数据宽度。要启用 64 位数据宽度，请应用答复记录 000034501 中提供的补丁。
- 支持 x1、x2、x4、x8 或 x16 链路宽度。
- 支持 Gen1、Gen2 和 Gen3 链路速度。Gen4 适用于 PCIE4C 块。
- 支持 AXI4 存储器映射接口和 AXI4-Stream 接口（按队列）。
- 2048 个队列集合
  - 2048 个 H2C 描述符环。
  - 2048 个 C2H 描述符环。
  - 2048 个 C2H 完成 (CMPT) 环。
- 支持轮询模式（状态描述符写回）和中断模式。
- 中断
  - 2048 个 MSI-X 矢量。
  - 每个功能最多 8 个 MSI-X。
  - 中断聚合。
- C2H 串流中断调制。
- C2H 串流完成队列条目合并。
- 通过用户逻辑进行描述符和 DMA 自定义
  - 允许定制描述符格式。
  - 流量管理。

- 支持含最多 4 个物理功能 (PF) 和 252 个虚拟功能 (VF) 的 SR-IOV
  - 精简虚拟机管理器型号。
  - QID 虚拟化。
  - 仅允许有特权的功能/物理功能对上下文和寄存器进行编程。
  - 支持功能级别复位 (FLR)。
  - 邮箱。
- 基于队列的丰富可编程性，例如，AXI4 存储器映射接口对比 AXI4-Stream 接口。

IP 相关信息

LogiCORE IP 相关信息表	
子系统规格	
支持的器件系列 [1]	UltraScale+™
支持的用户接口	AXI4 存储器映射、AXI4-Stream、AXI4-Lite
资源	资源使用情况网页。
子系统	
设计文件	加密 System Verilog
设计示例	Verilog
测试激励文件	Verilog
约束文件	赛灵思约束文件 (XDC)
仿真模型	Verilog
支持的软件驱动程序	Linux、DPDK 和 Windows 驱动程序 [2]
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：70927
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：

1. 如需获取受支持的器件的完整列表，请参阅 Vivado IP 目录。
2. 如需获取驱动程序详情，请参阅赛灵思 DMA IP 驱动程序。
3. 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。
4. 对于 Versal ACAP，请参阅《Versal ACAP DMA and Bridge Subsystem for PCI Express 产品指南》(PG344)。

AMD XILINX

第 2 章

概述

Queue-based Direct Memory Access (QDMA) 基于队列的直接存储器访问子系统是基于 PCI Express® (PCIe®) 的 DMA 引擎，该引擎专为满足高带宽和高包计数数据传输需求而优化。QDMA 由 UltraScale+™ Integrated Block for PCI Express 以及广泛的 DMA 和 Bridge 基础架构组成，可实现卓越的性能和灵活性。

QDMA Subsystem for PCIe 提供了广泛的设置和使用选项，大部分设置和选项均可按队列来选择，例如存储器映射 DMA 或串流 DMA、中断模式和轮询。该子系统提供了诸多选项，用于通过用户逻辑来自定义描述符和 DMA，以提供复杂的流量管理功能。

QDMA 引擎可凭借使用 QDMA 进行数据传输的主要机制来对主机操作系统所提供的指令（描述符）进行操作。QDMA 可使用描述符在主机到卡 (H2C) 方向或卡到主机 (C2H) 方向进行数据移动。您可基于逐个队列选择 DMA 流量是进入 AXI4 存储器映射 (MM) 接口还是进入 AXI4-Stream 接口。此外，QDMA 可以选择实现 AXI4 MM 主端口或实现 AXI4 MM 从端口，从而允许 PCIe 流量完全绕过 DMA 引擎。

QDMA 与其它 DMA 产品的主要区别在于队列的概念。队列的概念源自高性能计算 (HPC) 互连的远程直接存储器访问 (RDMA) 的“队列集”概念。这些队列可以通过接口类型进行单独配置，并且能够在许多不同的模式下工作。基于单个队列加载 DMA 描述符的方式，每个队列均可提供极低开销的建立时间选项和连续更新功能。通过将队列作为资源分配给多个 PCIe 物理功能 (PF) 和虚拟功能 (VF)，即可在各种多功能和虚拟化应用空间中使用同一个 QDMA 核与 PCI Express 接口。

QDMA Subsystem for PCIe 搭配赛灵思提供的 QDMA 参考驱动程序一起使用和实践，随后构建并输出以满足各种应用空间的需求。

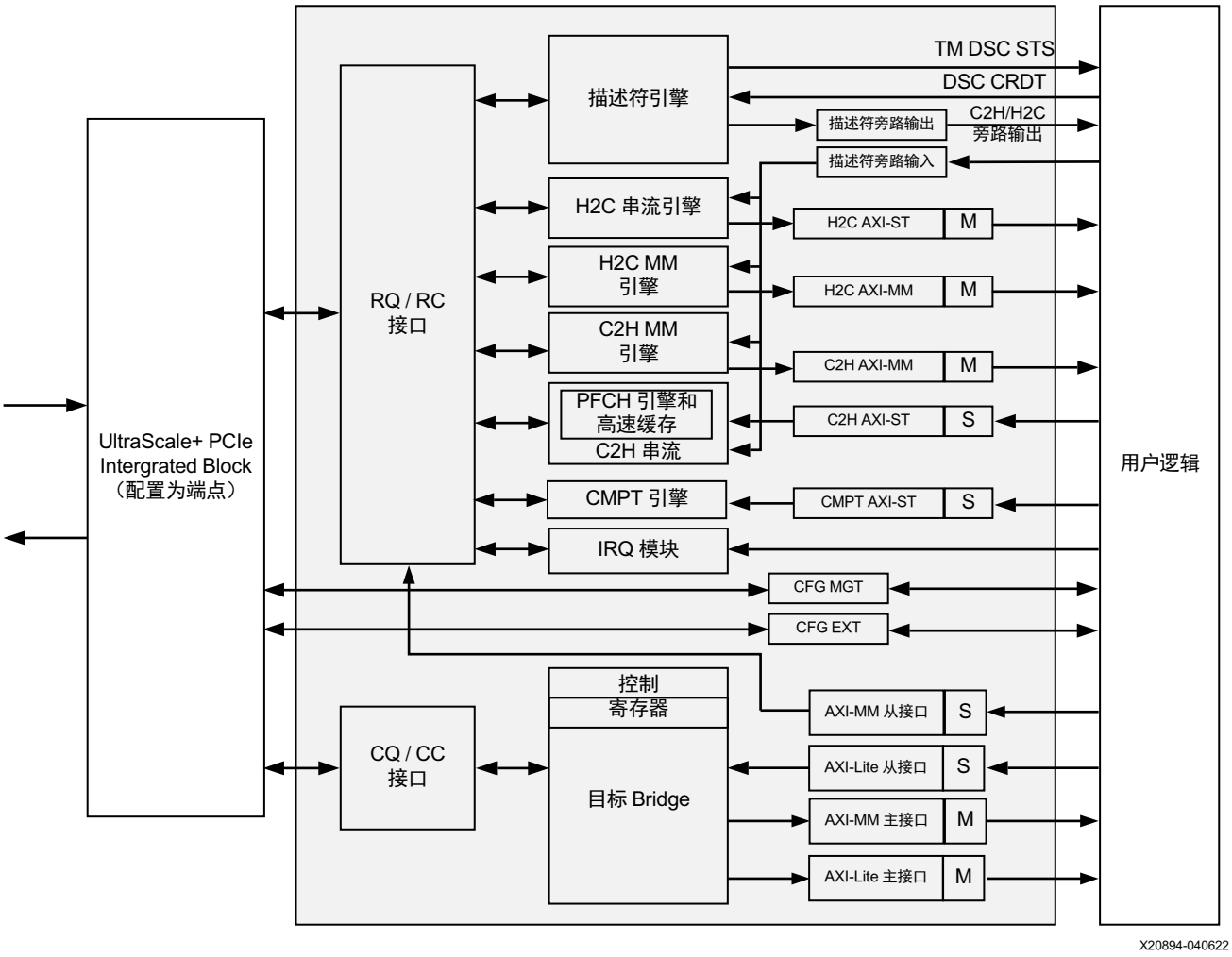
相关信息

端口描述

QDMA 架构

下图显示了 QDMA Subsystem for PCIe 的模块框图。

图 1：**QDMA 架构**



DMA 引擎

描述符引擎

描述符引擎可选择下列两种模式之一来提取主机到卡 (H2C) 和卡到主机 (C2H) 描述符：内部模式和描述符旁路模式。描述符引擎可维护每个队列的上下文，包括跟踪每个队列的软件 (SW) 生产者索引指针 (PIDX)、使用者索引指针 (CIDX)、队列基址 (BADDR) 和队列配置。描述符引擎使用循环算法来提取描述符。描述符引擎具有各自独立的缓冲器，分别用于 H2C 和 C2H 队列，并确保提取的描述符量永不超过可用空间。任一时间每个队列内，描述符引擎有且仅有一个未完成的 DMA 读取操作，描述符引擎可读取的描述符数量仅受 MRRS 中可包含的描述符数量限制。描述符引擎负责对乱序完成进行重新排序，并确保队列的描述符始终有序。

描述符旁路可按队列启用，并且缓冲后提取的描述符将被发送到相应的旁路输出接口，而不是直接发送到 H2C 或 C2H 引擎。在内部模式中，描述符将根据上下文设置来发送，并按 H2C 存储器映射 (MM)、C2H MM、H2C 串流或 C2H 串流引擎分别执行删除。

描述符引擎还负责生成对应 DMA 操作完成的状态描述符。除了 C2H 串流模式外，所有模式都使用这种机制来传达每个 DMA 操作的完成，以便软件回收描述符并清空任何关联的缓冲器。状态描述符的 CIDX 字段可指示此操作。

建议：如果队列与中断聚合相关联，赛灵思建议关闭状态描述符，改为从中断聚合环接收 DMA 状态。

要给提取的描述符数量施加限制（例如，限制存储描述符所需的缓冲量），可以逐队列开启信用值并进行节流。在此模式下，描述符引擎会提取描述符（上限不超过可用信用值），并且每个队列提取的描述符总数上限不超过提供的信用值。用户逻辑可以通过 `dsc_crdt` 接口返回信用值。信用值按描述符大小粒度来设置。

为了帮助用户开发的流量管理器排列工作负载优先顺序，PIDX 更新的待提取的可用描述符（增量 PIDX 值）将会被发送到 `tm_dsc_sts` 接口上的用户逻辑。通过使用此接口，即可实现相应设计来排列描述符存储的优先顺序并加以最优化。

相关信息

中断聚合环

H2C MM 引擎

H2C MM 通过 H2C AXI-MM 接口将数据从主机存储器移至卡存储器。该引擎会在 PCIe 上生成读取，并基于 MRRS 以及 PCIe 读取不得超过 4 KB 边界的要求，将描述符拆分为多个读取请求。收到读取请求的完成数据后，就会在 H2C AXI-MM 接口上生成 AXI 写入。对于未对齐的源和目标地址，硬件将对数据进行移位，并在 AXI-MM 上拆分写入，以免跨 4 KB 边界。检查每个已完成的描述符，判定是否需要写回和/或中断。

对于内部模式，描述符引擎会将存储器映射的描述符直接交付至 H2C MM 引擎。用户逻辑也可以将描述符注入 H2C 描述符旁路接口，以将数据从主机移至卡存储器。这样即可执行一些有趣的操作，如在同一个队列中混用控制和 DMA 命令。控制信息可发送至控制处理器，以指示 DMA 操作完成。

C2H MM 引擎

C2H MM 引擎通过 C2H AXI-MM 接口将数据从卡存储器移入主机存储器。该引擎会在 C2H AXI-MM 总线上生成 AXI 读取，基于 4 KB 边界将描述符拆分为多个请求。一旦在 AXI4 接口上接收到读取请求的完成数据，就会使用来自 AXI 读取的数据作为写入内容来生成 PCIe 写入。对于未对齐的源和目标地址，硬件将对数据进行移位，并在 PCIe 上拆分写入，从而满足最大有效载荷大小 (MPS) 要求并避免跨 4 KB 边界。检查每个已完成的描述符，判定是否需要写回和/或中断。

对于内部模式，描述符引擎会将存储器映射的描述符直接交付至 C2H MM 引擎。与 H2C MM 引擎一样，用户逻辑也可以将描述符注入 C2H 描述符旁路接口，以将数据从卡移至主机存储器。

对于多功能配置支持，在 AXI-MM 接口总线的 `aruser` 位中将提供 PCIe 功能编号信息，以便用户逻辑对卡存储器进行虚拟化。除了数据和用户总线外，还会提供奇偶校验总线用于端到端奇偶校验支持。

H2C 串流引擎

H2C 串流引擎用于将数据从主机移至 H2C 串流接口。对于内部模式，描述符直接交付至 H2C 串流引擎；对于旁路模式下的队列，描述符可重新格式化并馈送到旁路输入接口。引擎负责将 DMA 读取分解为 MRRS 大小，保证有足够空间可用于完成，并确保对完成进行重新排序以确保将 H2C 串流数据按顺序交付到用户逻辑。

引擎有足够缓冲可用于进行最多 256 次描述符读取和处理最多 32 KB 的数据。DMA 会提取数据并与 AXI4 接口侧要传输的第一个字节对齐。这样每个描述符就都有了随机偏移和随机长度。所有描述符的总长度必须小于 64 KB。

对于内部模式队列，每个描述符定义一个要传输到 H2C AXI-ST 接口的 AXI4-Stream 包。由于缺少逐队列存储空间，因此不允许跨接多个描述符的包。但是，跨接多个描述符的包可使用描述符旁路模式来实现。在此模式下，当用户逻辑有足够描述符可形成一个包时，即可启动 H2C DMA 引擎。DMA 引擎的启动方式是将跨接多个描述符的包随其它 H2C ST 包描述符一起通过旁路接口来进行交付，这样可确保各描述符之间不发生交织。另外，通过旁路接口，用户逻辑可以控制状态描述符的生成。

C2H 串流引擎

C2H 串流引擎负责接收来自用户逻辑的数据，并将其写入由 C2H 描述符提供的主机存储器地址，以供给定队列使用。

C2H 引擎有两个主要模块用于完成 C2H 串流 DMA：描述符预取高速缓存 (PFCH) 和 C2H-ST DMA 写入引擎。PFCH 具有逐队列上下文，旨在增强其功能的性能以及增强用于对其进行编程的软件的性能。

PFCH 高速缓存基于每个队列具有三种主要模式：称为 Simple Bypass Mode（简单旁路模式）、Internal Cache Mode（内部高速缓存模式）和 Cached Bypass Mode（高速缓存旁路模式）。

- 在简单旁路模式下，引擎不会对队列进行任何跟踪，用户逻辑可自行定义描述符的接收方法。随后，用户逻辑负责通过简单旁路接口来交付包和关联的描述符。在旁路模式下，所有队列在旁路接口和 C2H 串流接口中提取描述符的顺序都必须保持不变。
- 在内部高速缓存模式和高速缓存旁路模式下，PFCH 模块可以为最多 512 个描述符提供存储空间，这些描述符可供最多 64 个不同队列使用。在此模式下，为了控制要提取的描述符，引擎采用的方法是根据流水线中收到的包，按需对 C2H 描述符队列信用值加以控制。预取模式可按队列启用，启用此模式会导致适时预取描述符，使描述符在数据包可用之前即可用。其状态可在预取上下文中找到。这样即可以近乎即时方式将包数据传输至 PCIe 集成块，而不必等待相关描述符完成获取，因而显著缩短时延。对于队列（PFCH 上下文）来说，数据缓冲器的大小是固定的，引擎可将数据包分散在多达 7 个描述符上。在高速缓存旁路模式下，描述符通过旁路连接到用户逻辑以进行进一步处理（如地址转换），并通过接口中的旁路发回。这种模式不假设任何排序描述符和 C2H 串流包接口，并且预取引擎可以与包和描述符匹配。启用预取模式时，请勿向 IP 提供信用值。预取引擎负责信用值管理。

完成引擎

完成 (CMPT) 引擎用于写入到完成队列。尽管完成引擎可搭配 AXI-MM 接口和串流 DMA 引擎一起使用，但 C2H 串流 DMA 引擎旨在与完成引擎紧密合作。完成引擎也可用于将即时数据传递给完成环。完成引擎可用于在完成环中写入最多 64B 的完成。当完成与 DMA 引擎搭配一起使用时，驱动程序会使用完成来判定随每个包传输的数据字节数。这样驱动程序即可回收描述符。

完成引擎会维护完成上下文。此上下文由驱动程序进行编程，并按队列进行维护。完成上下文可存储各种信息，如，完成环的基址、PIDX、CIDX 和完成引擎的多方面信息，这些信息可通过设置完成上下文的各字段来加以控制。

引擎还可以根据软件需要，按每个队列进行配置，以生成中断和/或完成状态更新。如果多个队列的中断聚合到中断聚合环中，那么在中断聚合环中还会提供状态描述符信息。

CMPT 引擎具有多达 64 个条目的高速缓存，用于将多个较小的 CMPT 写入合并为 64B 写入，以提升 PCIe 效率。无论何时，完成均可同时合并，上限不超过 64 个队列。除此之外，只要有任何额外队列需要写入 CMPT 条目，就会导致从高速缓存中逐出最近使用频率最低的队列。为此，所用高速缓存的深度可配置为下列值：8、16、32 和 64。

Bridge 接口

AXI 存储器映射 Bridge 主接口

AXI MM Bridge 主接口用于从主机对 AXI 存储器映射空间进行高带宽访问。此接口支持最多 32 个未完成的 AXI 读取和写入。可将任意物理功能 (PF) 或虚拟功能 (VF) 的一个或多个 PCIe BAR 映射到 AXI-MM Bridge 主接口。此选择必须在设计编译之前完成。在 AXI-MM 接口的 `aruser` 和 `awuser` 中都将提供功能 ID、BAR ID、VF 组和 VF 组偏移，以便允许用户逻辑识别每次存储器访问的来源。在 AXI Bridge 主端口中会列出 `m_axib_awuser/m_axib_aruser[54:0]` 用户位的映射。

虚拟功能组 (VFG) 表示 VF 组编号。它等同于对应 VF 关联的 PF 编号。VFG_OFFSET 表示与特定 PF 相关的 VF 编号。请注意，它并非每个 PF 的 FIRST_VF_OFFSET。

例如，如果 PF0 和 PF1 均有 8 个 VF，则 PF0 和 PF1 的 FIRST_VF_OFFSET 分别为 4 和 11。以下是 VFG 和 VFG_OFFSET 的映射。

表 1：**AXI-MM 接口虚拟功能组**

功能编号	PF 编号	VFG	VFG_OFFSET
0	0	0	0
1	1	0	0
4	0	0	0（因为 PF0 的 FIRST_VF_OFFSET 为 4，所以 PF0 的第一个 VF 从 FN_NUM=4 开始，VFG_OFFSET=0 表示这是 PF0 的第一个 VF）
5	0	0	1（VFG_OFFSET=1 表示这是 PF0 的第二个 VF）
…	…	…	…
12	1	1	0（VFG=1 表示此 VF 与 PF1 关联）
13	1	1	1

每个主机发起的访问均可通过 PCIe 到 AXI BAR 转换单独映射到 64 位 AXI 地址空间。

由于所有功能都共享相同的 AXI 主接口地址空间，因此需要相应机制来将请求从不同功能映射到 AXI 主接口侧的不同地址空间。以下提供的示例演示了如何使用 PCIe 到 AXI 转换矢量。请注意，属于相同 PF 的所有 VF 都共享相同的 PCIe 到 AXI 转换矢量。因此，每个 VF 的 AXI 地址空间都连接在一起。VFG_OFFSET 可用于计算特定 VF 的 AXI 实际起始地址。

总结，AXI 主写入地址或主读取地址判定方式如下：

- 对于 PF，`address` = `pcie2axi_vec` + `axib_offset`。
- 对于 VF，`address` = `pcie2axi_vec` + (VFG_OFFSET + 1)*`vf_bar_size` + `axib_offset`。

其中，`pcie2axi_vec` 是 PCIe 到 AXI BAR 转换（从 Vivado® IP 目录配置 IP 核时即可设置此转换）。

`axib_offset` 是请求的目标空间中的地址偏移。

AXI4-Lite Bridge 主接口

可将任意物理功能 (PF) 或虚拟功能 (VF) 的一个或多个 PCIe BAR 映射到 AXI4-Lite 主接口。此操作必须在配置 IP 时完成。在 AXI4-Lite 接口的 `aruser` 和 `awuser` 中将提供功能 ID、BAR ID（BAR 命中）、VF 组和 VF 组偏移，帮助用户识别存储器访问的来源。

在 AXI4-Lite 主端口中会列出 m_axil_awuser/m_axil_aruser[54:0] 用户位的映射。

虚拟功能组 (VFG) 表示 VF 组编号。它等同于对应 VF 关联的 PF 编号。VFG_OFFSET 表示与特定 PF 相关的 VF 编号。请注意，它并非每个 PF 的 FIRST_VF_OFFSET。

例如，如果 PF0 和 PF1 均含 8 个 VF，PF0 和 PF1 的 FIRST_VF_OFFSET 分别为 4 和 11，以下是 VFG 和 VFG_OFFSET 的映射。

表 2：**AXI4-Lite 接口 VFG**

功能编号	PF 编号	VFG	VFG_OFFSET
0	0	0	0
1	1	0	0
4	0	0	0（因为 PF0 的 FIRST_VF_OFFSET 是 4，所以 PF0 的第一个 VF 从 FN_NUM=4 开始，VFG_OSSET=0 表示这是 PF0 的第一个 VF）
5	0	0	1（VFG_OSSET=1 表示这是 PF0 的第二个 VF）
…	…	…	…
12	1	1	0（VFG=1 表示此 VF 与 PF1 关联）
13	1	1	1

每个主机发起的访问均可通过 PCIe 到 AXI BAR 转换单独映射到 64 位 AXI 地址空间。

由于所有功能都共享相同的 AXI4 主接口地址空间，因此需要相应机制来将请求从不同功能映射到 AXI 主接口侧的不同地址空间。以下显示了如何使用 PCIe 到 AXI 转换矢量。请注意，属于相同 PF 的所有 VF 都共享相同的 PCIe 到 AXI 转换矢量。因此，每个 VF 的 AXI 地址空间都连接在一起。VFG_OFFSET 可用于计算特定 VF 的 AXI 实际起始地址。

综上所述，m_axil_awaddr 的判定方式为：

- 对于 PF，m_axil_awaddr = pcie2axi_vec + axil_offset。
- 对于 VF，m_axil_awaddr = pcie2axi_vec + (VFG_OFFSET + 1)*vf_bar_size + axil_offset

其中，pcie2axi_vec 是 PCIe 到 AXI BAR 转换（可在 IP 配置期间设置）。

axib_offset 是请求的目标空间中的地址偏移。

每个主机发起的访问均可单独映射到 64 位 AXI 地址空间。在此接口上支持一个未完成的读取操作和一个未完成的写入操作。

在 IP 配置时，还可将扩展 ROM BAR 映射到 AXI4-Lite 接口。

PCIe 到 AXI BAR

对于每个物理功能，PCIe 配置空间由一组 6 个 32 位存储器 BAR 和一个 32 位扩展 ROM BAR 组成。启用 SR-IOV 后，将为每个虚拟功能启用额外的 6 个 32 位 BAR。这些 BAR 会为 AXI4 存储器映射空间功能、接口布线和 AXI4 请求属性配置提供地址转换。任意成对 BAR 均可配置为单个 64 位 BAR。 在《AXI Bridge for PCI Express Gen3 Subsystem 产品指南》(PG194) 的“地址转换”部分中可找到编程示例（示例 3）。

请求存储器类型

可通过 attr_dma_pciebar2axibar_*_cache_pf* 属性来为每个 PCIe BAR 设置存储器类型。

- AxCache[0] 设为 1 表示可修改，设为 0 则表示不可修改。
- AxCache[1] 设为 1 表示可缓存，设为 0 则表示不可缓存。

AXI 存储器映射 Bridge 从接口

AXI-MM Bridge 从接口用于用户逻辑与主机之间的高带宽存储器传输。支持通过 AXI 到 PCIe BAR 进行 AXI 到 PCIe 的转换。此接口将按需拆分请求，以遵循 PCIe MPS 和跨 4 KB 边界要求。最多支持 32 个未完成的读取和写入请求。

AXI4-Lite Bridge CSR 从接口

AXI4-Lite 从接口用于访问 AXI Bridge 和 QDMA 内部寄存器。地址位 [15] 指示访问的是 QDMA 寄存器还是 AXI Bridge 寄存器。

- 当 `s_axil_csr_awaddr[15] = 1'b1` 时，写入访问对应的是 QDMA CSR 寄存器。
- 当 `s_axil_csr_awaddr[15] = 1'b0` 时，写入访问对应的是 Bridge 寄存器（访问 Bridge 寄存器时，从地址 `0x000` 到 `0xDFF` 的访问将重定向至 PCIe 核配置空间访问，源自地址 `0xE00` 的访问则将定向至 Bridge 寄存器）。
- 当 `s_axil_csr_araddr[15] = 1'b1` 时，读取访问对应的是 QDMA CSR 寄存器。
- 当 `s_axil_csr_araddr[15] = 1'b0` 时，读取访问对应的是 Bridge 寄存器。访问 Bridge 寄存器时，从地址 `0x000` 到 `0xDFF` 的访问将重定向至 PCIe 核配置空间访问，源自地址 `0xE00` 的访问则将定向至 Bridge 寄存器。

QDMA 寄存器专为 VF 和 PF 经过虚拟化。例如，VF 和 PF 可以访问地址空间的不同部分，并且各自均可访问自己的队列。为满足访问特定功能的需求，用户逻辑可以在写入访问 `s_axil_awuser[7:0]` 上和读取访问 `s_axil_aruser[7:0]` 上提供功能 ID，从而为 QDMA 提供适当的内部寄存器访问。在 AXI4-Lite 从接口上支持单一未完成读取请求和单一未完成写入请求。

AXI4-Lite 从接口还可使用 Bridge 寄存器生成供应商定义报文 (VDM)。

相关信息

VDM

AXI 到 PCIe BAR

在 Bridge 从接口中，有 1 个 BAR 可配置为 32 位或 64 位。此 BAR 可提供从 AXI 地址空间到 PCIe 地址空间的地址转换。地址转换是通过 BDF 表编程来配置的。如需了解有关 BDF 编程的信息，请参阅“Slave Bridge”章节。

相关信息

Slave Bridge

中断模块

IRQ 模块用于聚合各种来源的中断。中断源是基于队列的中断、用户中断和错误中断。

在 PF 和 VF 上允许基于队列的中断和用户中断，但仅在 PF 上才允许错误中断。如果未启用 SR-IOV，则每个 PF 都可以选择 MSI-X 或 Legacy Interrupts（遗留中断）。启用 SR-IOV 后，所有功能都仅支持 MSI-X 中断。

MSI-X 中断默认启用。主机系统（根联合体）将启用硬件中支持的单一或全部中断类型。如果启用 MSI-X，则 MSI-X 优先。

每个功能最多有 8 个中断可用。为了允许任一给定功能均可使用大量队列并且每个队列都有中断，QDMA Subsystem for PCIe 提供了全新的方法，可用于将来自多个队列的中断聚合到单个中断矢量中。这样原则上全部 2048 个队列都可以映射到单个中断矢量。QDMA 可提供 256 个中断聚合环，可在 256 个可用功能之间灵活分配。

PCIe 块接口

PCIe CQ/CC

PCIe 完成器请求 (CQ)/完成器完成 (CC) 模块用于从远程 PCIe 代理接收 TLP 请求并加以处理。此接口与 UltraScale+ Integrated Block for PCIe IP 对接，并在地址对齐模式下工作。此模块使用来自 Integrated Block for PCIe IP 的 BAR 信息以判定应转发的请求。这些请求的可能目标包括：

- DMA 配置模块
- AXI4 MM Bridge 主接口
- AXI4-Lite Bridge 主接口

非转发请求应接收来自目标的完成，并将其转发至远程 PCIe 代理。如需了解更多详细信息，请参阅《UltraScale+ Integrated Block for PCI Express LogiCORE IP 产品指南》(PG213)。

PCIe RQ/RC

PCIe 请求器请求 (RQ)/请求器完成 (RC) 接口会在 RQ 总线上生成 PCIe TLP，并处理来自 RC 总线的 PCIe 完成 TLP。此接口与 UltraScale+ Integrated Block for PCIe® 核对接，在 DWord 对齐模式下工作。对于 512 位接口，将启用跨接。支持跨接时，可能并不会实现 RQ 跨接传输事务的所有组合。如需了解更多详细信息，请参阅《UltraScale+ Integrated Block for PCI Express LogiCORE IP 产品指南》(PG213)。

PCIe 配置

有多个因素会导致对传出非转发传输事务进行节流。传出非转发传输事务是根据来自 PCIe® Integrated Block 的流量控制信息来加以节流的，以防止转发请求发生队头阻塞。DMA 将根据 PCIe 接收 FIFO 空间来测量非转发传输事务。

通用队列设计

QDMA Subsystem for PCIe 的多队列 DMA 引擎使用成对的 RDMA 模型队列支持用户逻辑中的 RNIC 实现。每个队列组均由主机到卡 (H2C)、卡到主机 (C2H) 和 C2H 串流完成 (CMPT) 组成。每个队列的元素均为描述符。

H2C 和 C2H 始终由驱动程序/软件写入；硬件始终从这些队列中读取。H2C 承载的描述符用于执行来自主机的 DMA 读取操作。C2H 承载的描述符则用于对主机执行 DMA 写入操作。

在内部模式下，H2C 描述符承载地址和长度信息，被称为聚集描述符。这些描述符支持 32 位元数据，这些元数据可随每个描述符一起从软件传递到硬件。根据上下文设置，描述符可作为存储器映射（承载主机地址、卡地址和 DMA 传输长度）描述符，或者也可以作为串流（仅承载主机和 DMA 传输长度）描述符。通过描述符旁路可定义任意描述符格式，以便软件将即时数据和/或附加元数据随包一起传递。

C2H 队列存储器映射描述符包含卡地址、主机地址和长度。在串流内部缓存模式下，描述符仅承载主机地址。描述符的缓冲器大小由驱动程序编程，对于整个队列，其大小应固定不变。实际传输中，与每个描述符关联的数据并不一定必须是缓冲器大小的完整长度。

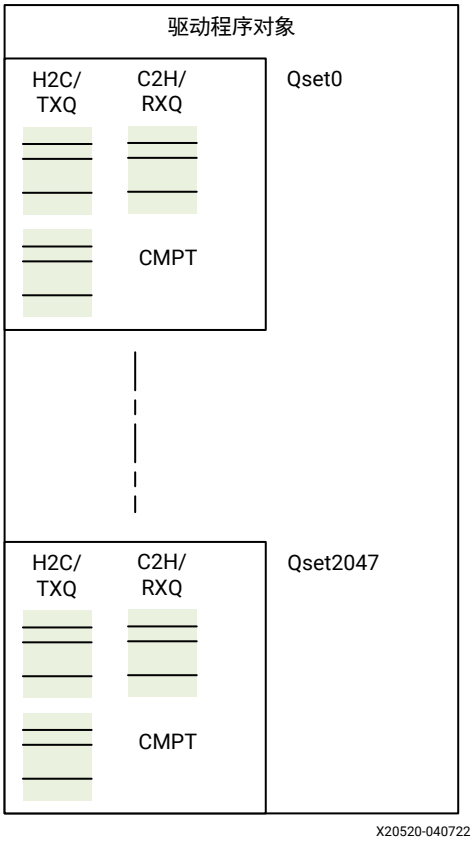
软件通过向硬件写入其生产者索引 (PIDX) 来为 H2C 和 C2H 队列播发有效描述符。状态描述符是描述符环（C2H 串流环除外）的最后一个条目。状态描述符承载硬件的使用者索引 (CIDX)，以便驱动程序知晓何时回收描述符并在主机中取消分配缓冲器。

对于 C2H 串流模式，C2H 描述符将根据 CMPT 队列条目进行回收。通常情况下，每个 C2H 包承载一个条目，表示耗用一个或多个 C2H 描述符。CMPT 队列条目承载有足够的信息，可供软件申领所耗用的所有描述符。通过外部逻辑可将其扩展到承载其它种类的完成或信息，以提供给主机。

驱动程序可使用描述符中的颜色位或 CMPT 环末端的状态描述符来检测由硬件写入环中的 CMPT 条目。每个 CMPT 条目均可承载 C2H 串流包的元数据，并可充当用户应用的定制完成或即时通知。

所有环形缓冲器（H2C、C2H 和 CMPT）的基址都应对齐到 4 KB 地址。

图 2：**队列环结构**



该软件可按 16 种不同的环大小进行编程。可从上下文编程中选择每个队列的环大小。最后一个队列条目是描述符状态，允许的条目数为（队列大小 - 1）。

例如，如果队列大小为 8，其中包含条目索引 0 到 7，那么最后一个条目（索引 7）保留用于表示状态。无论如何，此索引都不应该用于 PIDX 更新，PIDX 更新也不应该等于 CIDX。在此情况下，如果 CIDX 为 0，最大 PIDX 更新将为 6。

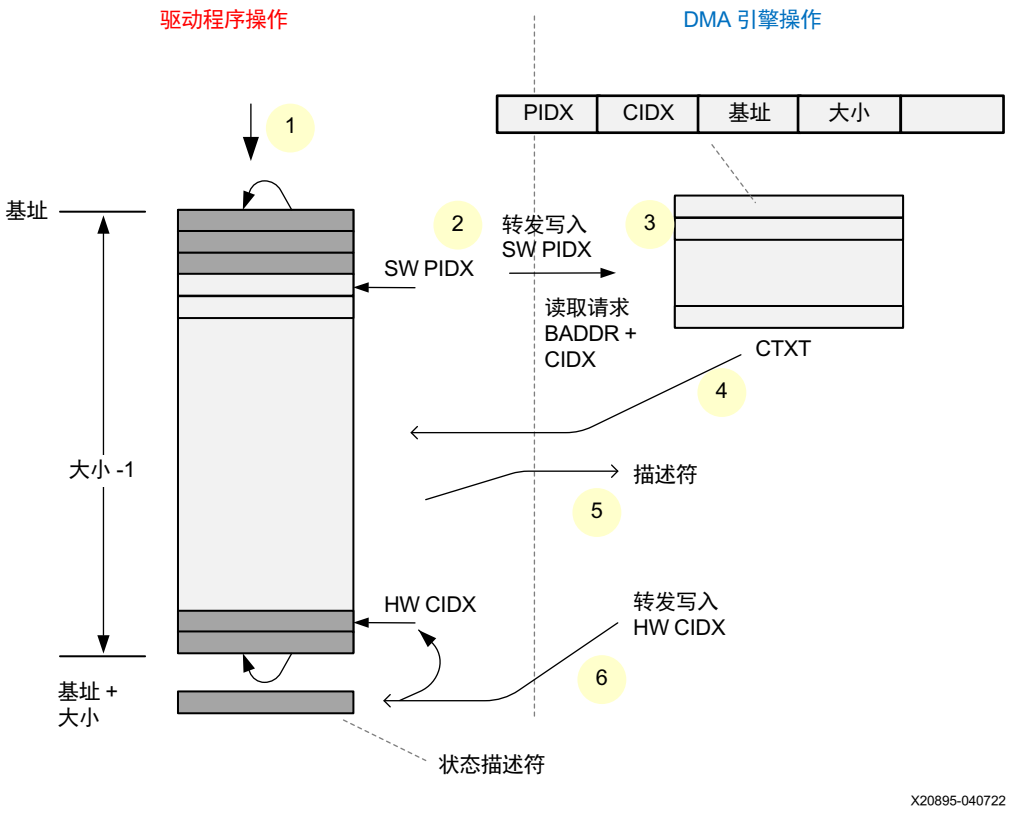
在以上示例中，如果流量已启动，并且 CIDX 为 4，那么最大 PIDX 更新为 3。

H2C 和 C2H 队列

H2C/C2H 队列是位于主机存储器中的环。对于这两种类型的队列，生产者是软件，使用者则是描述符引擎。软件负责维护生产者索引 (PIDX) 和硬件使用者索引 (HW CIDX) 的副本，以避免覆盖未读取的描述符。描述符引擎还负责维护使用者索引 (CIDX) 和 SW PIDX 的副本，以确保引擎不会读取未写入的描述符。队列中的最后一个条目专用于状态描述符，引擎会在该状态描述符中写入 HW CIDX 和其它状态。

引擎在本地存储器中维护总计 2048 个 H2C 上下文和 2048 个 C2H 上下文。上下文负责存储队列的属性，例如，队列的基址 (BADDR)、SW PIDX、CIDX 和深度。

图 3：**简单的 H2C 和 C2H 队列**



上图显示了 H2C 和 C2H 的提取操作。

1. 对于 H2C，驱动程序会将有效载荷写入主机缓冲器，与有效载荷缓冲器信息一起构成 H2C 描述符，并将其放入 PIDX 位置处的 H2C 队列中。对于 C2H，驱动程序以保留的可用缓冲器空间来构成描述符，以便接收来自 DMA 的包写入。
2. 驱动程序会将转发的写入发送到描述符引擎中的 PIDX 寄存器中，以获取关联的队列 ID (QID) 及其当前 PIDX 值。
3. 在接收到 PIDX 更新后，引擎会根据地址偏移和功能 ID 来计算指针更新的绝对 QID。通过使用 QID，引擎将从与 QDMA Subsystem for PCIe 关联的存储器提取绝对 QID 的上下文。
4. 引擎会根据上下文判定允许提取的描述符数量。引擎会使用基址 (BADDR)、CIDX 和描述符大小来计算描述符地址，然后引擎会发出 DMA 读取请求。
5. 描述符引擎从主机存储器接收到读取完成后，就会将其交付至内部模式下的 H2C 引擎或 C2H 引擎。对于旁路，描述符将传出至关联描述符旁路输出接口。

6. 对于编程为内部模式的存储器映射或 H2C 串流队列，在完全处理完提取的描述符后，引擎会将 CIDX 值写入状态描述符。对于编程为旁路模式的队列，用户逻辑会通过接口中的旁路来控制写回。状态描述符可根据上下文设置进行调节。对于完成，C2H 串流队列始终使用 CMPT 环。

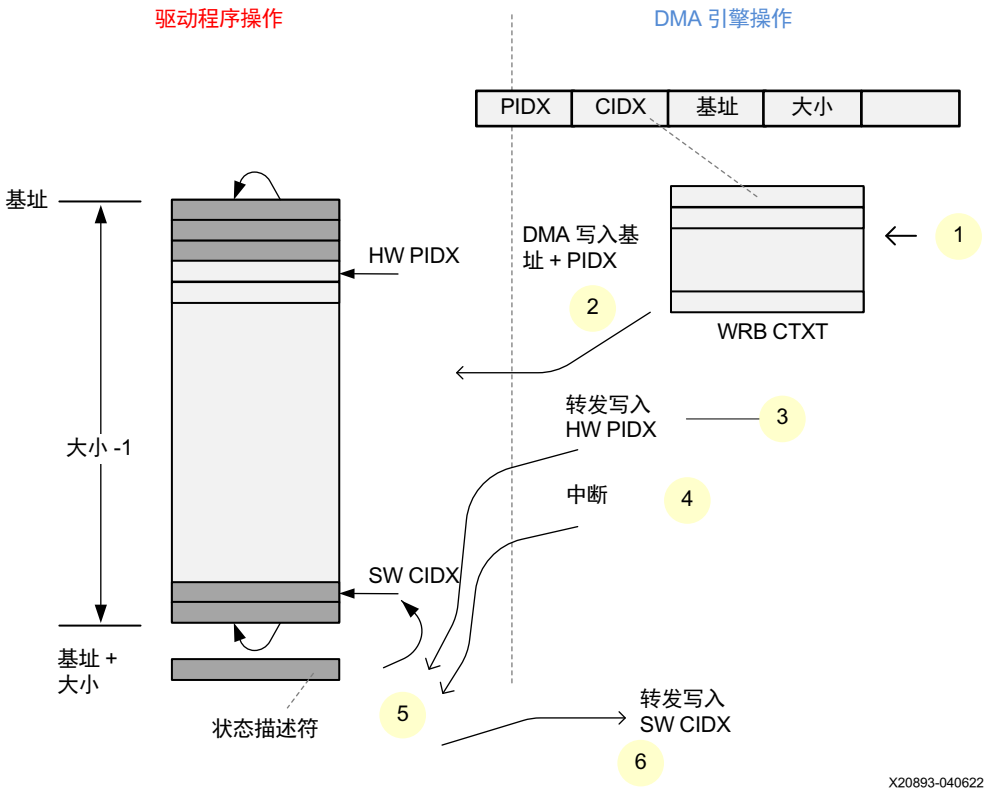
对于 C2H，通过 CMPT 环来隐式完成提取操作。

完成队列

完成 (CMPT) 队列是位于主机存储器中的环。使用者是软件，生产者则是 CMPT 引擎。软件负责维护使用者索引 (CIDX) 和硬件生产者索引 (HW PIDX) 的副本，以避免读取未写入的完成。CMPT 引擎还负责维护 PIDX 和软件使用者索引 (SW CIDX) 副本，以确保引擎不会覆盖未读取的完成。队列中的最后一个条目专用于状态描述符，引擎会在该状态描述符中写入硬件生产者索引 (HW PIDX) 和其它状态。

该引擎在本地存储器中总计维护 2048 个 CMPT 上下文。上下文可存储队列的属性，如基址、SW CIDX、PIDX 和队列深度。

图 4：**简单的完成队列流程**



C2H 串流应使用 CMPT 队列将完成传输至主机，但它也可以用于其它类型的完成操作或用于向驱动程序发送报文。通过 CMPT 的报文保证不会绕过对应的 C2H 串流包 DMA。

符合上述编号方式的 DMA CMPT 队列操作的简单流程如下所示：

1. CMPT 引擎通过 CMPT 接口接收完成报文，但完成报文的 QID 来自 C2H 串流接口。该引擎会读取 CMPT 上下文 RAM 的 QID 索引。
2. DMA 则会将 CMPT 条目写入地址 BASE+PIDX。
3. 如果所有条件都满足，则可选择将 PIDX 随颜色位一起写入 CMPT 队列的状态描述符中。

4. 如果启用中断模式，CMPT 引擎会生成中断事件报文并发送给中断模块。
5. 驱动程序可采用轮询模式或中断模式。无论采用何种方式，驱动程序都会识别新的 CMPT 条目，方法是匹配颜色位，或者将状态描述符中的 PIDX 值与其当前的软件 CIDX 值进行比较。
6. 驱动程序会为该队列更新 CIDX。这样硬件即可复用描述符。软件完成对 CMPT 的处理后，即在停止轮询或离开中断处理程序之前，驱动程序会对关联队列的 CIDX 更新寄存器发出写入。

SR-IOV 支持

QDMA Subsystem for PCIe 提供了可选功能特性，以支持单根 I/O 虚拟化 (SR-IOV)。PCI-SIG® 单根 I/O 虚拟化与共享 (SR-IOV) 规范（详见《PCI-SIG 规范》(www.pcisig.com/specifications)）旨在对数据路径传输事务内所涉及的 VMM 的旁路方法加以规范化，并允许单一端点显示为多个独立端点。SR-IOV 将各功能分类为：

- **物理功能 (PF)**：全功能特性的 PCIe® 功能，包含 SR-IOV 等功能。
- **虚拟功能 (VF)**：PCIe 功能，特供含基址寄存器 (BAR) 的配置空间，但缺少完整配置资源，由 PF 配置进行控制。VF 的主要作用是数据传输。

除了 PCIe 定义的配置空间，QDMA Subsystem for PCI Express 还会将数据路径操作（如队列的指针更新和中断）加以虚拟化。其余管理和配置功能都将推迟到物理功能驱动程序。没有足够特权的驱动程序必须通过邮箱接口与有特权的驱动程序进行通信，邮箱接口在 QDMA Subsystem for PCI Express 内提供。

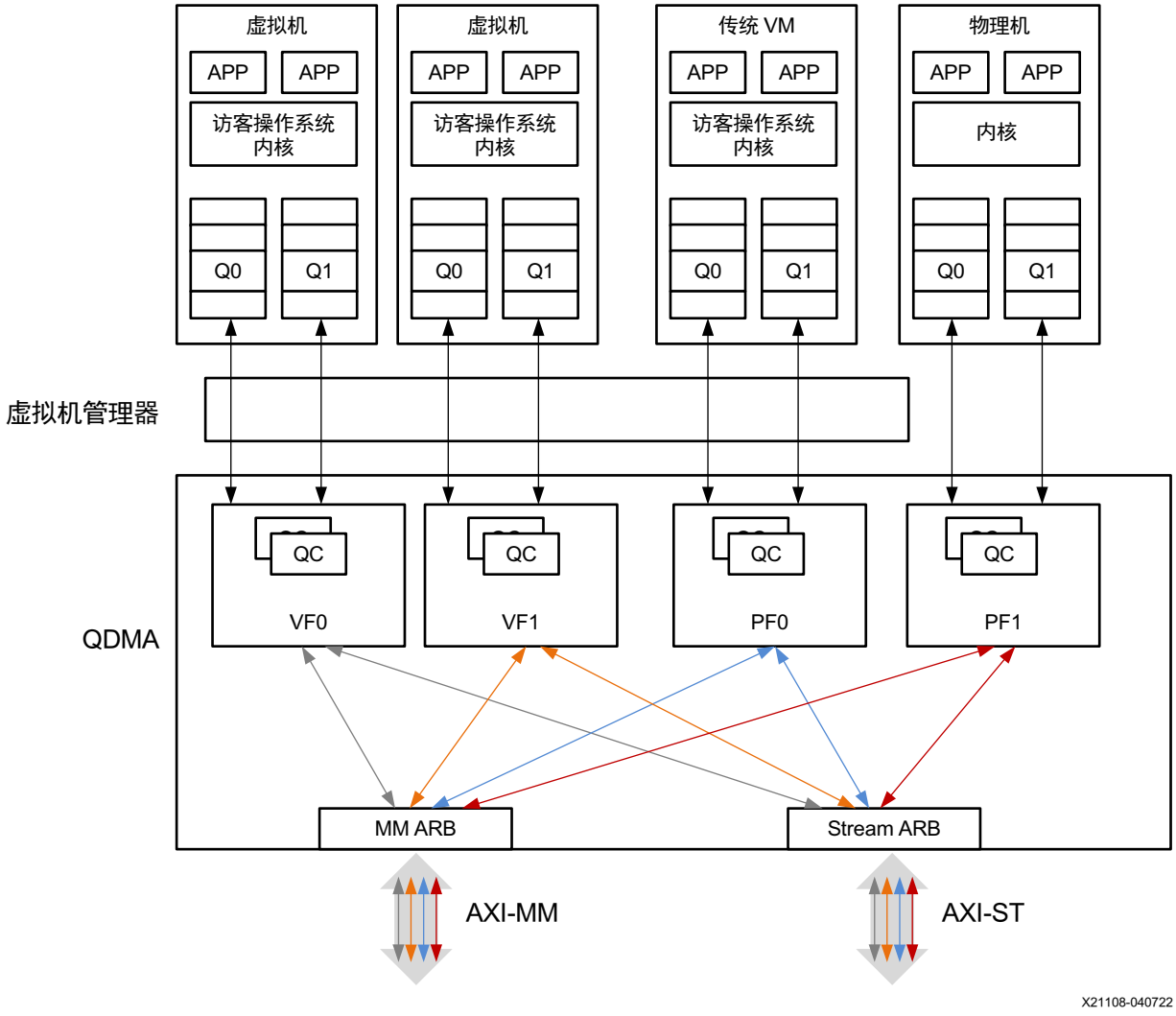
安全性是虚拟化的一个重要方面。QDMA Subsystem for PCI Express 可提供以下安全性功能：

- QDMA 只允许有特权的 PF 对每个队列上下文和寄存器进行配置。VF 会将任何队列上下文编程告知对应的 PF。
- 驱动程序仅限于为分配到的队列执行指针更新。
- 通过开启系统 IOMMU 即可检查 PF 或 VF 当前请求的 DMA 访问。ARID 来自于有特权的功能所编程的队列上下文。

任何 PF 或 VF 均可通过邮箱与除本身以外的其它 PF 进行通信。每个功能均可实现一个 128B 收件箱和 128B 发件箱。这些邮箱对自身功能的 DMA BAR（通常是 BAR0）中的驱动程序均可见。无论何时，任一功能都能有一条传出邮箱报文和一条传入邮箱报文处于未完成状态。

下图显示了典型系统如何使用 QDMA 的不同功能和操作系统。不同的队列可分配到不同的功能，每个功能都可彼此独立传输 DMA 包。

图 5：**系统中的 QDMA**



限制

QDMA Subsystem for PCIe 的限制如下：

- DMA 支持任意 VF 功能上最多 256 个队列。
- Slave Bridge AXI 不支持窄突发传输。

建议：AXI SmartConnect 可用于支持窄突发。

- SRIOV 在 Bridge 模式下不受支持。
- QDMA5.0 不支持 64 位 AXI 宽度组合（例如，Gen1x1）。

应用

QDMA Subsystem for PCIe 广泛应用于联网、计算和数据存储应用。QDMA Subsystem for PCIe 的常用示例是用于实现数据中心和远程通信应用，例如，计算加速、Smart NIC、NVMe、启用 RDMA 的 NIC (RNIC)、服务器虚拟化和用户逻辑中的 NFV。通过为每个应用分配不同的队列集和 PCIe 功能，即可实现多个共享 QDMA 的应用。随后，即可在用户逻辑中通过调整这些队列的规模来实现限速、流量优先级和定制工作队列条目 (WQE)。

许可和订购

根据赛灵思最终用户许可条款，此赛灵思 LogiCORE™ IP 模块随附于赛灵思 Vivado® Design Suite 免费提供。

如需了解有关该子系统的更多信息，请访问 QDMA Subsystem for PCIe 产品页面网页。

AMD XILINX

第 3 章

产品规格

标准

QDMA Subsystem for PCI Express 遵循以下标准：

- 《AMBA AXI4-Stream 协议规范》(ARM IHI 0051A)
- PCI Express 基本规范 v3.1
- PCI 局部总线规范
- PCI-SIG® 单根 I/O 虚拟化和共享 (SR-IOV) 规范

欲知详情，请参阅《PCI-SIG 规范》(https://www.pcisig.com/specifications)。

性能和资源使用情况

性能

如需获取 QDMA 性能和详细分析，请参阅答复记录 71453。

赛灵思提供了两个设计示例供您实验。标准设计示例仅用于功能测试。要生成用于性能分析的设计示例，使用以下 Tcl 命令来生成性能设计示例：

```
set_property CONFIG.performance_exdes{true} [get_ips qdma_0]
```

以下是赛灵思推荐的 QDMA 寄存器设置，此设置旨在提升性能。性能数值将根据系统和使用的操作系统而异。

- QDMA_C2H_INT_TIMER_TICK (0xB0C) 设为 25。对应于 100 ns（对于 250 MHz 用户时钟，1 拍 = 4 ns）
- C2H 触发模式设为 Counter（计数器）+ Timer（定时器），计数器设为 64，定时器则与往返时延相匹配。定时器的全局寄存器值应为 30，表示 3 µs。
- QDMA_GLBL_DSC_CFG (0x250)，`max_desc_fetch` = 6，`wb_int` = 5
- QDMA_H2C_REQ_THROT (0xE24)，`req_throt_en_data` = 1，`data_thresh` = 0x4000
- QDMA_C2H_PFCH_CFG (0xB08/0xA80/0xA84)
  - `evt_qcnt_th` = (QDMA_C2H_PFCH_CACHE_DEPTH/2) - 2
  - `pfch_qcnt` = QDMA_C2H_PFCH_CACHE_DEPTH/2
  - `num_pfch` = 8。建议最小值为 8。如果环境内的活动队列数量较少，那么采用更高的值有助于提升性能。
  - `pfch_fl_th` = 256

- QDMA_C2H_WRB_COAL_CFG (0xB50)，
  - max_buf_sz = QDMA_C2H_CMPT_COAL_BUF_DEPTH (0xBE4)
  - tick_val = 25
  - tick_cnt = 5
- TX/RX API 突发大小 = 64，环深度 = 2048。驱动程序应以 64 为单位成批更新 TX/RX PIDX。
- PCIe MPS = 256 字节，MRRS >= 512 字节，启用扩展标签，启用宽松排序
- 驱动程序将以 64 为单位成批更新完成 CIDX，从而在更新 C2H PIDX 前减少 MMIO 写入数量
- 驱动程序应以 64 为单位成批更新 H2C PIDX，并通过更新来获取分散聚集列表的最后一个描述符。
- C2H 上下文：
  - bypass = 0（内部模式）
  - frcd_en = 1
  - qen = 1
  - wbk_en = 1
  - irq_en = irq_arm = int_aggr = 0
- C2H 预取上下文：
  - pfch = 1
  - bypass = 0
  - valid = 1
- C2H CMPT 上下文：
  - en_stat_desc = 1
  - en_int = 0 (Poll_mode)
  - int_aggr = 0（轮询模式）
  - trig_mode = 5
  - counter_idx = 对应于 64
  - timer_idx = 对应于 3 µs
  - valid = 1
- H2C 上下文：
  - bypass = 0（内部模式）
  - frcd_en = 0
  - fetch_max = 0
  - qen = 1
  - wbk_en = 1
  - wbi_chk = 1
  - wbi_intvl_en = 1
  - irq_en = 0（轮询模式）
  - irq_arm = 0（轮询模式）
  - int_aggr = 0（轮询模式）

为了获取最优 QDMA 串流性能，描述符环的包缓冲器应至少对齐到 256 字节。

建议：赛灵思建议您在 PCIe 上将描述符提取的未完成总数限制为小于 8 KB。例如，对于 16B 描述符，将所有队列的未完成信用值限制为 512。

描述符旁路模式中的性能

在描述符旁路模式下配置设计时，以上所有设置都适用。以下信息提供了在旁路模式下提升性能的建议。

1. 在 `h2c_byp_in_st_sdi` 端口中设置旁路时，QDMA IP 会为每个包生成状态写回。赛灵思建议在 32 位包或 64 位包中，对此端口执行一次断言有效。如无更多描述符需处理，则在最后一个描述符处将 `h2c_byp_in_st_sdi` 断言有效。此要求基于每个队列，并且适用于 AXI4-MM（H2C 和 C2H）旁路传输和 AXI4-Stream H2C 传输。
2. 对于 AXI-Stream C2H 简单旁路模式，出于性能原因，`dsc_crdt_in_fence` 端口应设为 1。此建议假定用户设计已为每个队列合并信用值并且已将其发送至 IP。在内部模式下，请在 QDMA_C2H_PFCH_CFG_2 (0xA84) 寄存器中设置 `fence` 位。

资源使用情况

如需了解有关 QDMA 资源使用情况的信息，请参阅资源使用情况网页。

器件最低要求

Gen3x16 功能至少需要 -2 速度等级。

表 3：**器件最低要求**

功能链路速度	功能链路宽度	受支持的速度等级
含 PCIE4 块的 UltraScale+™ 器件		
Gen1/Gen2	x1、x2、x4、x8 和 x16	-1、-1L、-1LV、-2、-2L、-2LV 和 -3
Gen3	x1、x2 和 x4	-1、-1L、-1LV、-2、-2L、-2LV 和 -3
	x8	-1、-2、-2L 和 -3
	x16	-2、-2L 和 -3
含 PCIE4C 块的 UltraScale+ 器件		
Gen1/Gen2	x1、x2、x4、x8 和 x16	-1、-2、-2L、-2LV 和 -3
Gen3	x1、x2 和 x4	-1、-2、-2L、-2LV 和 -3
	x8	-1、-2、-2L 和 -3
	x16	-2、-2L 和 -3
Gen4[1]	x1、x2、x4 和 x8	-2、-2L 和 -3

注释：
1. 如需了解 Gen4 模式限制，请参阅《UltraScale+ Integrated Block for PCI Express LogiCORE IP 产品指南》(PG213)。

注释：此 IP 支持含 PCIe 块的所有 UltraScale+™ 器件，以下器件除外：

- Zynq UltraScale+ MPSoC 器件 ZU5 和更小的器件
- Kintex UltraScale+ FPGA 器件 KU3 和更小的器件
- Artix UltraScale+ FPGA 器件 AU25P 和更小的器件

QDMA 操作

描述符引擎

描述符引擎负责管理每个队列的主机到卡 (H2C) 和卡到主机 (C2H) 描述符环形缓冲器的使用者侧的操作。每个队列的上下文用于判定描述符引擎处理每个队列的不同方式。当描述符可用且满足其它条件时，描述符引擎将向 PCIe 发出读取请求以提取描述符。收到的描述符将卸载到描述符旁路输出接口（旁路模式）或直接交付给 DMA 引擎（内部模式）。当 H2C 串流或存储器映射 DMA 引擎完成描述符时，可将状态写回状态描述符，并且可生成中断和/或标记响应，以将当前 DMA 进度告知软件和用户逻辑。描述符引擎还会提供流量管理器接口，将每个队列的某些状态通知用户逻辑。这样在需要对 DMA 行为进行自定义和最优化时，即可允许用户逻辑做出明智的决策。

描述符上下文

描述符引擎在描述符上下文中存储每个队列的配置、状态和控制信息，这些信息可以存储在块 RAM 或 UltraRAM 中，上下文则由 H2C 或 C2H QID 来索引。启用队列之前，硬件和信用值上下文必须首先清零。完成这些操作后，即可对软件上下文进行编程，并且可通过对 `qen` 位进行置位来启用该队列。启用队列后，除非要禁用该队列，否则只能通过直接映射的地址空间来更新软件上下文以便更新生产者索引和中断 Arm 位。硬件上下文和信用值上下文仅包含状态。仅限在队列初始化的过程中，才需要与硬件和信用值上下文进行交互，以便将其全部清零。一旦启用队列，即可由硬件动态更新上下文。启用队列时，通过间接总线对上下文的任何修改都会导致意外的行为。不建议在队列启用时读取上下文，因为这可能导致性能下降。

相关信息

QDMA_DMAP_SEL_H2C_DSC_PIDX[2048] (0x18004)
QDMA_DMAP_SEL_C2H_DSC_PIDX[2048] (0x18008)

软件描述符上下文结构（0x0 C2H 和 0x1 H2C）

描述符上下文供描述符引擎适用。

表 4：**软件描述符上下文结构定义**

位	位宽	字段名称	描述
[255:140]	116	reserved	保留。设为 0。
[139]	1	int_aggr	如果该位完成置位，则将在中断环中聚合中断。
[138:128]	[10:0]	vec	此 MSI-X 矢量用于对应直接中断的中断或者对应聚合中断的中断聚合条目。
[127:64]	64	dsc_base	描述符环的 4K 对齐基址。
[63]	1	is_mm	该字段可判定队列是否是存储器映射队列。如果该字段置位，描述符将交付至关联的 H2C 或 C2H MM 引擎。 1：存储器映射 0：串流
[62]	1	mrkr_dis	如果该字段置位，则在内部模式中禁用标记响应。 不适用于 C2H ST。
[61]	1	irq_req	由于等待发送时出错（等待 irq_arm）而中断。初始化队列上下文时，该位应清零。 不适用于 C2H ST。

表 4：**软件描述符上下文结构定义** (续)

位	位宽	字段名称	描述
[60]	1	err_wb_sent	针对错误已发送写回/中断。该位完成置位后，则针对此队列将不再发送写回或中断。初始化队列上下文时，该位应清零。 不适用于 C2H ST。
[59:58]	2	err	错误状态。 Bit[1] dma – DMA 操作期间发生错误。检查引擎状态寄存器。 Bit[0] dsc – 描述符提取或更新期间发生错误。检查描述符引擎状态寄存器。初始化队列上下文时，该字段应设为 0。
[57]	1	irq_no_last	未发送中断，并且在内部模式下，生产者索引 (PIDX) 或使用者索引 (CIDX) 已处于空闲状态。当 irq_arm 位进行置位时，将发送中断。发送中断时或者队列的 PIDX 更新后，该位应自动清零。 初始化队列上下文时，该位应初始化为 0。 不适用于 C2H ST。
[56:54]	3	port_id	Port_id 在用户接口上将针对与此队列关联的事件发送此端口 ID。
[53]	1	irq_en	中断使能。 在主机状态更新时，将向主机发送中断。 针对 C2H ST 设为 0。
[52]	1	wbk_en	写回使能。 在主机状态更新时，将发送针对状态描述符的存储器写入。
[51]	1	mm_chn	设为 0 且不可修改。
[50]	1	bypass	如果该位完成置位，则队列将在旁路模式下运行，否则将在内部模式下运行。
[49:48]	2	dsc_sz	描述符提取大小。0：8B，1：16B；2：32B；3：64B。 如果不启用旁路模式，则存储器映射 DMA 需 32B，H2C 串流 DMA 需 16B，C2H 串流 DMA 需 8B。 如果针对旁路模式配置队列，则可选择任意描述符大小。描述符将在旁路输出接口上交付。由用户逻辑负责处理描述符，然后将其回馈给描述符旁路输入。
[47:44]	4	rng_sz	描述符环大小索引。该索引会从 16 个寄存器（偏移 0x204:0x240）中选出大小与其它环不同的那一个环。
[43:41]	3	reserved	保留
[40:37]	4	fetch_max	针对此队列未完成的描述符提取的最大数量。最大未完成为 fetch_max + 1。更高的值可提升单一队列性能，
[36]	1	at	基址的地址类型。 0：未转换 1：已转换 此类型将作为地址类型 (AT) 在 PCIe 上用于描述符提取和状态描述符写回。
[35]	1	wbi_intvl_en	写回/中断时间间隔。 基于处理的描述符数量启用定期状态更新。 适用于内部模式。 不适用于 C2H ST。写回时间间隔由寄存器 QDMA_GLBL_DSC_CFG (0x250) bits[2:0] 来判定。

表 4：**软件描述符上下文结构定义** (续)

位	位宽	字段名称	描述
[34]	1	wbi_chk	暂挂检查后的写回/中断。 当队列已完成所有可用描述符时启用状态更新。 适用于内部模式。
[33]	1	fcrd_en	启用提取信用值。 提供给该队列的信用值数量将用于限定提取的描述符数量。 针对 C2H ST 设为 1。
[32]	1	qen	表示队列已启用。
[31:25]	7	reserved	保留
[24:17]	8	fnc_id	功能 ID
[16]	1	irq_arm	中断装备。当该位进行置位时，允许队列生成中断。
[15:0]	16	pidx	生产者索引。

硬件描述符上下文结构（0x2 C2H 和 0x3 H2C）

表 5：**硬件描述符结构定义**

位	位宽	字段名称	描述
[47]	1	reserved	保留
[46:43]	4	fetch_pnd	描述符提取待处理
[42]	1	evt_pnd	事件待处理
[41]	1	idl_stp_b	队列无效，且无待处理描述符。 启用队列时，该位即置位。禁用队列（软件上下文 qen 位）且没有其它待处理描述符时，该位即清零。 [0] 禁用队列（软件上下文 qen 位）并且没有其他待处理描述符。 [1] 启用队列。
[40]	1	dsc_pnd	描述符待处理。如果上一个完成的 CIDX 与当前的 PIDX 不匹配，描述符就会被定义为待处理。
[39:32]	8		保留
[31:16]	16	crd_use	耗用的信用值。如果在软件上下文中启用提取信用值，则适用。
[15:0]	16	cidx	上一个提取的描述符的使用者索引。

信用值描述符上下文结构

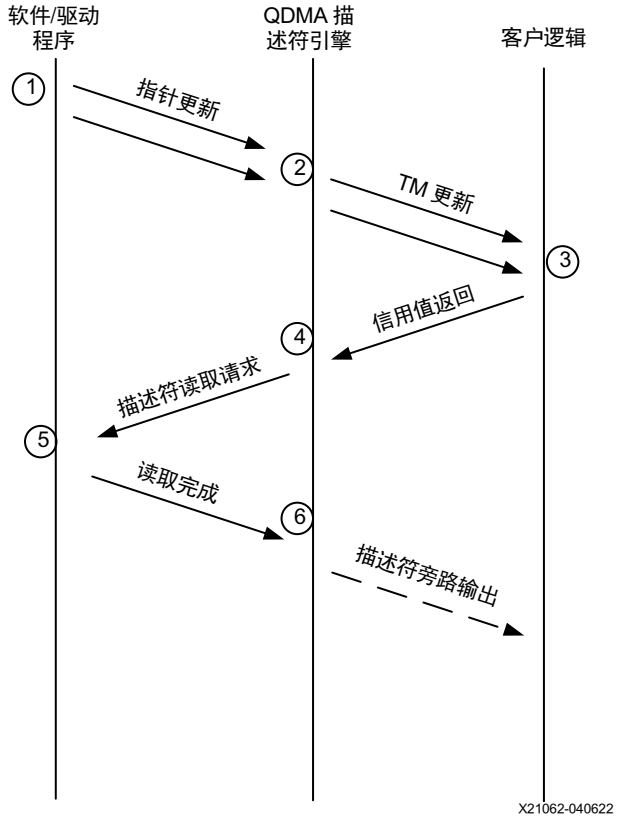
表 6：**信用值描述符上下文结构定义**

位	位宽	字段名称	描述
[31:16]	16	reserved	保留
[15:0]	16	credt	提取接收的信用值。 如果在软件上下文中启用提取信用值，则适用。

信用值描述符上下文仅供内部 DMA 使用，并且可从间接总线读取以供调试。此上下文会为使用 CREDIT_ADD 操作通过描述符信用值接口所接收到的每个队列存储信用值。如果信用值操作的 `dsc_crdt_in_fence` 位设为 1，那么仅当描述符读取请求生成时才会添加信用值。

描述符提取

图 6：**描述符提取流程**



1. 通过更新队列的描述符 PIDX，即可将描述符的可用性告知描述符引擎。这部分上下文将直接映射到 QDMA_DMAP_SEL_H2C_DSC_PIDX 和 QDMA_DMAP_SEL_C2H_DSC_PIDX 地址空间。
2. 更新 PIDX 时，描述符引擎会根据上次提取的使用者索引 (CIDX) 来评估可用描述符的数量。通过流量管理器状态接口即可将新描述符的可用性传达给用户逻辑。
3. 如果提取信用值已启用，那么需要用户逻辑来为应提取的每个描述符提供信用值。
4. 如果描述符可用并且提取信用值已禁用或为非零值，则描述符引擎将向 PCIe 生成描述符提取。提取的描述符的数量由 PCIe 最大读取请求大小 (MRRS) 和描述符提取信用值（如果启用）加以进一步限定。由于完成空间不足，也可能会导致描述符提取停滞。在每个方向上，均为 C2H 和 H2C 各分配了 256 个条目用于描述符提取完成。每个条目宽度都与数据路径相同。如有足够空间可用，则允许继续进行提取。无论何时，任一给定队列仅限在 PCIe 上有一个待处理的描述符提取。
5. 主机会接收读取请求并将描述符读取完成提供给描述符引擎。
6. 描述符存储在缓冲器中，直到可将其卸载为止。如果在旁路模式下配置队列，则描述符将发送到描述符旁路输出端口。否则，将直接交付至 DMA 引擎。交付后，描述符提取完成缓冲器空间就将取消分配。

注释：可用的描述符大小始终是 <ring size> - 2。无论何时，软件都不应将 PIDX 更新至超过 <ring size> - 2。

例如，如果队列大小为 8，其中包含条目索引 0 到 7，那么最后一个条目（索引 7）保留用于表示状态。该索引不应用于 PIDX 更新，并且 PIDX 更新应与 CIDX 永不相等。在此情况下，如果 CIDX 为 0，最大 PIDX 更新将为 6。

内部模式

通过设置软件上下文旁路字段，即可将队列配置为以描述符旁路模式或内部模式运行。在内部模式下，该队列不需要外部用户逻辑来处理描述符。由描述符引擎提取的描述符会被直接交付至相应的 DMA 引擎并进行处理。内部模式允许对用户逻辑进行信用值提取和状态更新，以便在运行时对描述符提取行为进行自定义。

内部模式写回和中断（AXI MM 和 H2C ST）

状态写回和/或中断是由硬件根据队列上下文自动生成的。当 `wbi_intvl_en` 置位时，写回/中断将根据寄存器 QDMA_GLBL_DSC_CFG (0x250) bits[2:0] 中选定的时间间隔来发送。由于中断缓慢，在间隔模式下，中断可能会迟到或跳过时间间隔。如果 `wbi_chk` 上下文位已置位，那么当描述符引擎检测到当前 PIDX 的最后一个描述符已经完成时，将发送写回/中断。建议针对所有内部模式操作，都应对 `wbi_chk` 位进行置位，包括启用间隔模式时。在软件对 `irq_arm` 位完成置位之后，才会生成中断。发送中断后，硬件会将 `irq_arm` 位清零。如果在 `irq_arm` 位未置位时需要中断，则中断将置于待处理状态，直至 `irq_arm` 位完成置位为止。

描述符完成定义如下：如描述符数据传输已完成并且其写入数据在 AXI（对应 AXI MM 为 H2C bresp，对应 ST 则为 Valid/Ready）上已得到确认，或者已被 PCIe 控制器的传输事务层接受并用于发射 (C2H MM)，即表示描述符完成。

描述符旁路模式

描述符旁路模式还支持对用户逻辑进行信用值和状态更新。此外，描述符旁路模式允许用户逻辑自定义描述符和状态更新的处理方式。由描述符引擎提取的描述符通过描述符旁路输出接口交付至用户逻辑。这样即可允许用户逻辑按需对描述符进行预处理或存储。在描述符旁路输出接口上，描述符可采用定制格式（遵循描述符大小要求）。为执行 DMA 操作，用户逻辑会将描述符（必须采用 QDMA 格式）驱动到描述符旁路输入接口中。

如果用户逻辑已有描述符（必须是 QDMA 格式），则可通过描述符旁路端口将其直接提供给 DMA。如果描述符已在用户逻辑中，那么用户逻辑就不需要从主机中提取描述符。用户逻辑不应通过描述符信用值接口来发送信用值。

描述符旁路模式写回/中断

在旁路模式下，用户逻辑对主机更新拥有显式控制权，并通过标记向用户逻辑返回响应。将每个描述符提交给存储器映射引擎（H2C 和 C2H）或 H2C 串流 DMA 引擎的描述符旁路输入端口后，会出现 CIDX 和 `sdi` 字段。此 CIDX 用于识别在描述符完成时生成的任何状态更新（主机写回、标记响应或合并中断）中，有哪个描述符已完成。如果描述符的 `sdi` 字段为输入，那么当 `wbk_en` 位完成置位后将对主机生成写回。如果 `sdi` 位已置位，并且上下文 `irq_en` 和 `irq_arm` 位也已置位，那么还可发送中断。

如果已启用中断，则用户逻辑必须监控 `irq_arm` 的流量管理器输出。为该队列观测 `irq_arm` 位之后，将向 DMA 发送含 `sdi` 位的描述符。发送含 `sdi` 位的描述符后，将先再次执行 `irq_arm` 断言有效，然后才会发送另一个含 `sdi` 位的描述符。如果在尚未能正确观测到 arm 位时，用户就对 `sdi` 位进行置位，那么可能无法发送中断，并且软件可能会无限期挂起以等待中断。未启用中断时，对于 `sdi` 位的置位则不存在任何限制。但写回事件过多可能严重降低描述符引擎性能，并耗用针对主机的写入带宽。

描述符完成定义如下：如描述符数据传输已完成并且其写入数据在 AXI4（对应 AXI MM 为 H2C `bresp`，对应 ST 则为 Valid/Ready）上已得到确认，或者已被 PCIe 控制器的传输事务层接受并用于发射 (C2H MM)，即表示描述符完成。

标记响应

要为任何描述符生成标记响应，只需对 `mrkr_req` 位进行置位即可。标记响应是在描述符完成后生成的。与主机写回类似，标记响应请求过多也会降低描述符引擎性能。如果在上下文中已配置 `sbi` 位，那么标记响应也可以随该位一起发送到用户逻辑。标记响应在队列状态端口上发送，这些端口可通过队列 ID 来识别。

描述符完成定义如下：如描述符数据传输已完成并且其写入数据在 AXI（对应 AXI MM 为 H2C `bresp`，对应 ST 则为 Valid/Ready）上已得到确认，或者已被 PCIe 控制器的传输事务层接受并用于发射 (C2H MM)，即表示描述符完成。

流量管理器输出接口

流量管理器接口可向用户逻辑提供队列状态详细信息，以便用户逻辑管理描述符的提取和执行。在正常操作中对于已启用的队列，每次 `irq_arm` 位断言有效或队列 PIDX 执行更新时，描述符引擎都会断言 `tm_dsc_sts_valid` 有效。`tm_dsc_sts_avl` 信号会指明自上次更新以来可用的新描述符数量。通过这种机制，用户逻辑可以跟踪每个队列可用的工作量。其作用主要体现在，可通过描述符引擎的提取信用值机制或其它用户最优化来排列提取的优先顺序。在有效周期内，`tm_dsc_sts_irq_arm` 表明 `irq_arm` 位先前为 0 并已置位。在旁路模式下，这实质上是此队列中断的信用值。当软件指定或者由于错误导致队列失效时，`tm_dsc_sts_qinv` 信号将置位。观测该位可以发现，描述符引擎已中止该队列的新描述符提取。在此情况下，`tm_dsc_sts_avl` 上的内容会指明描述符引擎所持有的可用提取信用值的数量。此信息可用于帮助用户逻辑协调给予描述符引擎的信用值，以及它应收到的描述符的数量。即使在 `tm_dsc_sts_qin` 断言有效后，提取流水线中已存在的有效描述符仍将继续交付至 DMA 引擎（内部模式）或者交付至描述符旁路输出端口（旁路模式）。

`tm_dsc_sts` 接口的其它字段会识别队列 ID、DMA 方向（H2C 或 C2H）、内部或旁路模式、串流或存储器映射模式、队列启用状态、队列错误状态和端口 ID。

虽然 `tm_dsc_sts` 接口是 valid/ready（有效/就绪）状态的接口，但为了获得最佳性能，不应对其施加反压。由于有多起事件会触发单个 `tm_dsc_sts` 周期，如果填满内部缓冲器，那么将中止描述符提取以免生成新事件。

相关信息

QDMA 流量管理器信用值输出端口

描述符信用值输入接口

当队列的 `fcrd_en` 上下文位已置位时，信用值接口即为相关。它允许用户逻辑对为每个队列提取的描述符进行优先级排序和计量。您可以指定 DMA 方向、qid 和信用值。对于典型用例，描述符引擎使用信用值输入来提取描述符。而在内部，将为每个队列跟踪已接收和已消耗的信用值。如果在队列未启用时添加信用值，则将通过流量管理器输出接口返回信用值，同时 `tm_dsc_sts_qinv` 断言有效，而 `tm_dsc_sts_avl` 中的信用值则无效。监控 `tm_dsc_sts` 接口以保留每个队列耗用的信用值记录。

相关信息

QDMA 描述符信用值输入端口

错误

在描述符提取和描述符执行期间都可能发生错误。在这两种情况下，一旦检测到队列错误，就会导致该队列失效、在上下文中记录错误位、停止为遇到错误的队列提取新描述符，并且还可在状态寄存器中记录错误。如果为写回、中断或标记响应启用相应的功能，DMA 还将对这些接口生成状态更新。完成此操作后，将不会为队列发送额外的写回、中断或标记响应（内部模式），直到队列上下文被清除为止。因错误引发队列失效后，还将生成流量管理器输出周期，指示该错误及队列失效。队列失效后，如有错误，您可通过读取该队列的错误寄存器和上下文来确定原因。您必须清除并移除该队列，然后，在后续有需要时重新添加该队列。

尽管将中止执行描述符提取，但流水线中已有的提取将继续处理，并且描述符将照常交付至 DMA 引擎或描述符旁路输出接口。如果描述符提取本身遇到错误，则将为该描述符添加错误位标记。如果错误位已置位，则描述符内容应视为无效。同一队列的后续描述符提取可能不会遇到错误，并且错误位不会置位。

存储器映射 DMA

在存储器映射 DMA 操作中，DMA 的源和目标均为存储器映射空间。在 H2C 传输中，源地址属于 PCIe 地址空间，而目标地址则属于 AXI MM 地址空间。在 C2H 传输中，源地址属于 AXI MM 地址空间，而目标地址则属于 PCIe 地址空间。不支持 PCIe 到 PCIe DMA 和 AXI MM 到 AXI MM DMA。除 DMA 方向之外，传输 H2C 和 C2H DMA 的行为是相似的，并且共享相同的描述符格式。

操作

存储器映射 DMA 引擎（H2C 和 C2H）的启用方式是在存储器映射引擎控制寄存器中对 `run` 位进行置位。当 `run` 断言无效时，描述符可被丢弃。在源缓冲器提取时已启动的任何描述符都将继续处理。`run` 位重新断言有效将导致对内部引擎状态进行复位，仅当引擎静止时才能执行此操作。描述符是从描述符引擎直接接收到的，或者是从描述符旁路输入接口接收到的。对于内部模式下的任何队列，都不应通过描述符旁路输入接口为其提供描述符。发送到未运行的 MM 引擎的所有描述符都将被丢弃。对于启用混用内部模式队列和旁路模式队列的配置，将执行循环仲裁以确立顺序。

DMA 存储器映射引擎首先向源接口生成读取请求，并在特定于接口的对齐边界处拆分描述符。PCIe 和 AXI 读取接口均可配置为按不同对齐进行拆分。读取数据的完成空间是在发出读取时预先分配的。同样，对于写入请求，DMA 引擎将按适当对齐方式进行拆分。在 AXI 接口上，每个引擎都将使用单一 AXI ID。DMA 引擎会按发出读取的顺序，对读取完成/写入数据进行重新排序。收到足够的读取完成数据后，就会向目标接口发出写入请求，其顺序与请求读取数据的顺序相同。在停用该请求前，目标接口必须接受所有写入数据并提供完成响应。对于 PCIe，当传输事务层接受写入请求时，将在下一个链路上发出写入完成。对于 AXI 存储器映射接口，则 `bresp` 是完成标准。一旦满足完成标准，就会为描述符生成相应的主机写回、中断和/或标记响应。

DMA 存储器映射引擎还支持描述符旁路输入的 `no_dma` 字段以及零长度 DMA。在该引擎中，这两种情况的处理方式相同。此处描述符与所有其它描述符一样通过 DMA 引擎进行传输，因此仍可在队列中观测到描述符排序。但不会生成 DMA 读取或写入请求。当所有先前的描述符都已完成其状态更新检查后，就会处理零长度描述符/`no_dma` 描述符的状态更新（写回、中断和/或标记响应）。

相关信息

描述符旁路模式写回/中断

错误

DMA 存储器映射引擎主要有两种类别的错误。第一类是随传入描述符置位的错误位。在此情况下，不处理描述符的 DMA 操作，但描述符将穿过该引擎直至状态更新阶段，并显示错误指示。根据上下文和配置，这将导致写回、中断和/或标记响应。这也将导致队列失效。DMA 存储器映射引擎的第二类错误是在 DMA 本身执行期间遇到的错误。其中包括 PCIe 读取完成错误和 AXI `bresp` 错误 (H2C)，或 AXI `bresp` 错误和 PCIe 写入错误，这些错误都是由于总线主接口使能或功能级别复位 (FLR) 以及 RAM ECC 错误所导致的。第一个启用的错误会记录到 DMA 引擎中。请参阅存储器映射引擎错误日志。读取时如发生错误，则 DMA 写入将异常中止（如可行）。从 RAM 中提取写入数据时如检测到错误，则无法中止请求。而是改为生成无效的数据奇偶校验，以确保目标能发现问题。在遇到错误的描述符通过 DMA 引擎后，它将继续生成含错误指示的状态更新。与描述符错误一样，它也会导致队列失效。

H2C 和 C2H 的 AXI 存储器映射描述符 (32B)

表 7：**H2C 和 C2H 的 AXI 存储器映射描述符结构**

位	位宽	字段名称	描述
[255:192]	64	reserved	保留
[191:128]	64	dst_addr	目标地址

表 7：**H2C 和 C2H 的 AXI 存储器映射描述符结构** (续)

位	位宽	字段名称	描述
[127:92]	36	reserved	保留
[91:64]	28	lengthInByte	读取长度（以字节为单位）
[63:0]	64	src_addr	源地址

内部模式下的存储器映射 DMA 必须将描述符队列配置为 32B，并遵循上述描述符格式。在旁路模式下，描述符格式由用户逻辑定义，用户逻辑必须驱动 H2C 或 C2H MM 旁路输入端口。

H2C 和 C2H 的 AXI 存储器映射写回状态结构

MM 写回状态寄存器位于（H2C 或 C2H）描述符的最后一个条目之后。

表 8：**H2C 和 C2H 的 AXI 存储器映射写回状态结构**

位	位宽	字段名称	描述
[63:48]	16	reserved	保留
[47:32]	16	pidx	写回时的生产者索引
[31:16]	16	cidx	使用者索引
[15:2]	14	reserved	保留
[1:0]	2	err	错误 位 1：描述符提取错误 位 0：DMA 错误

串流模式 DMA

H2C 串流引擎

H2C Stream Engine（串流引擎）负责将串流数据从主机传输到用户逻辑。H2C 串流引擎在 H2C 串流描述符上运行。每个描述符均可指定要传输给用户逻辑的数据的起始地址和长度。H2C 串流引擎会解析描述符，通过 PCIe 向主机发出读取请求，并在 MRRS 边界处拆分读取请求。H2C 串流引擎中最多可以有 256 个未完成的请求用于隐藏主机读取时延。H2C 串流引擎能够实现 32 KB 的重新排序缓冲器，以便在 TLP 返回时对其进行重新排序。数据发送到用户逻辑的顺序与向 PCIe 发送请求的顺序相同。

如果在关联的 H2C 上下文中启用了状态描述符，则该引擎在完成向用户逻辑发送数据后，还可额外向主机发送状态写回。

内部和旁路模式

QDMA Subsystem for PCIe 中的每个队列均可在两种 H2C 串流模式下进行编程：内部模式和旁路模式。在队列上下文中即可指定模式。H2C 串流引擎已知当前正在处理的描述符对应的是内部模式下的队列还是旁路模式下的队列。

下图显示了内部模式和旁路模式的流程。

图 7：**H2C 内部模式流程**

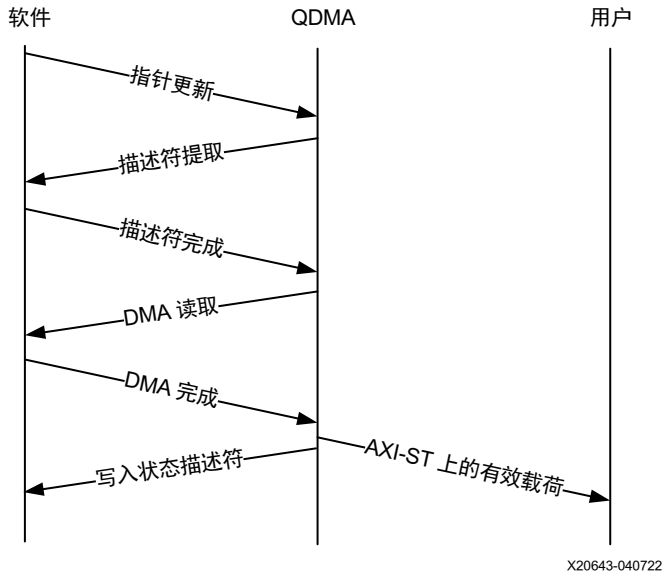
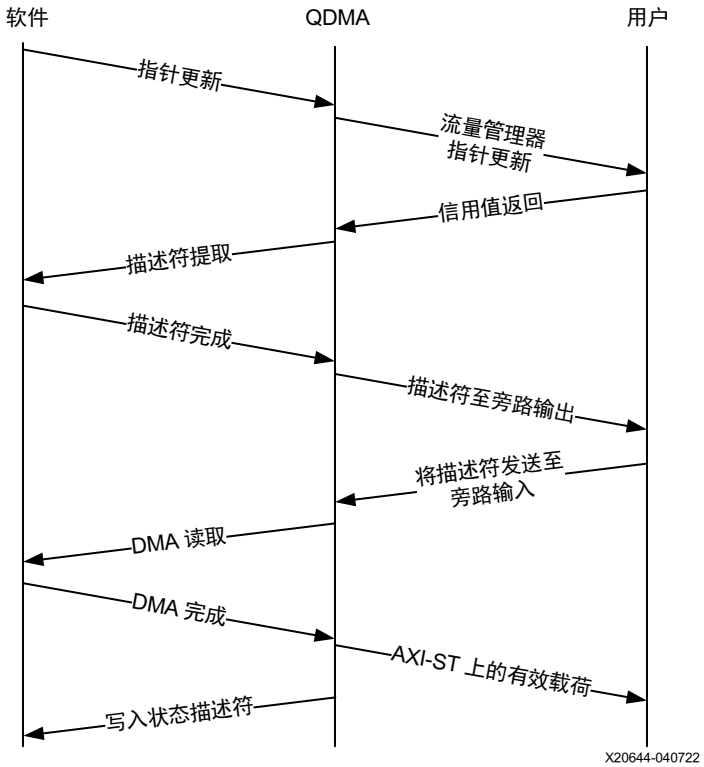


图 8：**H2C 旁路模式流程**



对于“Internal mode”（内部模式）下的队列，从主机提取描述符后，会将其直接馈送给 H2C 串流引擎以供处理。在此情况下，数据包不得跨多个描述符。因此，对于内部模式下的队列，每个描述符在 QDMA H2C AXI4-Stream 输出上都仅生成一个 AXI4-Stream 包。如果此包存在于主机存储器中的非连续空间内，那么必须以多个描述符来对其进行定义，这就要求在旁路模式下对此队列进行编程。

在“Bypass mode”（旁路模式）下，从主机提取描述符后，会使用 QDMA 旁路输出端口将其直接发送给用户逻辑。QDMA 不会对这些描述符进行任何解析。用户逻辑可以存储这些描述符，然后使用 QDMA H2C 串流描述符旁路输入接口将所需信息从这些描述符发送回 QDMA。QDMA 会使用此信息来构建描述符，然后将其馈送给 H2C 串流引擎以供处理。

在软件上下文中启用 `fcrd_en` 时，DMA 将等待用户应用提供信用值，即上图中的“信用值返回”。如果 `fcrd_en` 未置位，那么 DMA 会使用指针更新、提取描述符并发出描述符。用户应用不应传入信用值。在此情况下，上图中的“信用值返回”并不适用。

以下是使用旁路模式的优点：

- 用户逻辑可采用定制描述符格式。原因是 QDMA Subsystem for PCIe 不会对旁路模式下队列的描述符进行解析。在 H2C 串流旁路输入接口上，用户逻辑会对这些描述符进行解析，并提供 QDMA 所需的信息。
- 即时数据可从软件传递至用户逻辑，无需 DMA 操作。
- 用户逻辑准备好灌入所有数据时，即可通过将描述符发送至 QDMA 来执行流量管理。描述符可缓存在本地 RAM 中。
- 执行地址转换。

使用旁路模式时，对用户逻辑会施加一些要求。由于旁路模式允许包跨多个描述符，因此用户逻辑需向 QDMA 指明哪个描述符用于标记包起始 (SOP)，哪个描述符用于标记包结束 (EOP)。在 QDMA H2C 串流旁路输入接口中，用户逻辑需要提供的信息包括但不限于：地址、长度、SOP 和 EOP。当用户逻辑将 SOP 描述符信息馈入 QDMA 后，它最终还必须馈入 EOP 描述符信息。这些多描述符数据包的描述符必须按顺序馈入。不属于该包的其它描述符不得在多描述符包中进行交织。用户逻辑必须将描述符累积到 EOP 描述符，然后再将其反馈给 QDMA。否则会导致挂起。一旦 QDMA 为 EOP 描述符发出最后一拍后，QDMA 就将在 QDMA H2C AXI Stream 数据输出生成 TLAST。这是必然的，因为用户需按顺序提交给定包的描述符。

H2C 串流接口由所有队列共享，如果用户逻辑没有预留空间用于灌入该包，则可能会出现队头阻塞问题。如果包尺寸很大，则服务质量可能会受到严重影响。串流引擎旨在为大小低至 128B 的包实现 PCIe 饱和，因此赛灵思建议您将包大小限制为主机页面大小，或用户应用所需的最大传输单元。

H2C 串流引擎中提供的性能控制能力是指，当 H2C 串流引擎发现 PCIe 侧未完成的数据达到一定量之后，即停止向 PCIe RQ/RC 发出请求。要使用此功能特性，软件必须在 H2C_REQ_THROT(0xE24) 寄存器中对阈值进行编程。H2C 串流引擎所含待交付至用户逻辑的未完成数据量超过此阈值后，它就会停止向 PCIe RQ/RC 发送更多读取请求。默认情况下禁用此功能特性，可使用 H2C_REQ_THROT (0xE24) 寄存器将其启用。此功能特性有助于提升 C2H 串流性能，因为 H2C 串流引擎提交请求的速度比 C2H 串流引擎快得多。这可能会耗尽 PCIe 侧的 H2C 流量资源，导致对 C2H 流量产生不利影响。H2C_REQ_THROT (0xE24) 寄存器还允许软件单独启用并编程 H2C 串流引擎中可包含的未完成读取请求的最大数量阈值。因此，该寄存器可用于单独启用和编程 H2C 串流引擎中未完成的请求和数据的阈值。

H2C 串流描述符 (16B)

表 9：**H2C 描述符结构**

位	位宽	字段名称	描述
[127:96]	32	addr_h	地址高位。主机中源地址的高 32 位
[95:64]	32	addr_l	地址低位。主机中源地址的低 32 位
[63:48]	16	reserved	保留
[47:32]	16	len	包长度。要为此描述符提取的数据的长度。 这也是包的长度，因为在内部模式下，任一包都不能跨多个描述符。 包的最大长度可以是 64K-1 字节。

表 9：**H2C 描述符结构** (续)

位	位宽	字段名称	描述
[31:0]	32	metadata	元数据。QDMA 在 H2C-ST TUSER 上将此字段随每一拍上的数据一起传递。对于内部模式下的队列，它可用于将报文随数据一起从软件传递到用户逻辑。

此 H2C 描述符格式仅适用于内部模式。对于旁路模式，用户逻辑可以根据用户应用所需来定义自己的格式。

描述符元数据

与旁路模式类似，内部模式也提供相应机制用于将信息从软件直接传递到用户逻辑。除了地址和长度外，H2C 串流描述符还具有 32b 元数据字段。QDMA Subsystem for PCIe 并不会将该字段用于 DMA 操作。而是改为在包的每一拍，将其传递到 H2C AXI4-Stream `tuser` 上。对于旁路模式下的队列，不支持在 `tuser` 上传递元数据，因此没有输入可供在 QDMA H2C 串流旁路输入接口上提供元数据。

零长度描述符

描述符的长度字段可以为零。在此情况下，H2C 串流引擎将在 PCIe 上发出一个零字节的读取请求。QDMA 接收到请求完成后，H2C 串流引擎将在 QDMA H2C AXI4-Stream 接口上传出一拍数据，其中包含 `tlast`。在该接口上将以如下方式来指示此零字节包：对 `tuser` 中的 `zero_b_dma` 位进行置位。用户逻辑必须为零字节描述符同时设置 SOP 和 EOP。否则，H2C 串流引擎将标记错误。

H2C 串流状态描述符写回

在旁路输入接口上提供描述符信息时，用户逻辑可以请求 QDMA Subsystem for PCIe 在完成从主机提取数据时，向主机发送状态写回。用户逻辑也可请求在 DMA 完成后向其发送状态。这些行为可使用旁路输入接口中的 `sdi` 和 `mrkr_req` 输入来控制。

H2C 写回状态寄存器位于 H2C 描述符列表的最后一个条目之后。

注释：写入描述符环的 H2C-ST 状态描述符格式与写入中断合并条目的描述符格式不同。

表 10：**AXI4-Stream H2C 写回状态描述符结构**

位	位宽	字段名称	描述
[63:32]	32	reserved	保留
[47:32]	16	pidx	生产者索引
[31:16]	16	cidx	使用者索引
[15:2]	14	reserved	保留（生产者索引）
[1:0]	2	error	错误 0x0：无错误 0x1：在此队列上遇到描述符错误或数据错误 0x2 和 0x3：保留

相关信息

QDMA 描述符旁路输入端口

H2C 串流数据对齐器

H2C 引擎有一个数据对齐器，用于将数据对齐到零字节 (0B) 边界，然后再将其发送到用户逻辑。这样即可以任意对齐方式来对齐描述符的起始地址，同时仍能接收 H2C AXI4-Stream 数据总线上的数据，而不会在数据开始处出现任何孔隙。用户逻辑可以将描述符成批从 SOP 发送到 EOP，且每个描述符均可采用任意地址和长度对齐方式。对齐器将对来自不同描述符的数据进行对齐和打包，并将在 H2C AXI4-Stream 数据总线上发出连续数据串流。发出 EOP 描述符的最后一拍后，该接口上的 `tlast` 将断言有效。

处理含错误的描述符

如果提取描述符时遇到错误，QDMA 描述符引擎会将此描述符标记为含错误。对于内部模式下的队列，H2C 串流引擎处理错误描述符的方式为不执行任何 PCIe 或 DMA 活动。改为等待错误描述符穿过流水线并在完成此操作后强制写回。对于旁路模式下的队列，由用户逻辑负责禁止发出含错误描述符的批量描述符。用户逻辑必须改为仅发送单个含错误输入的描述符，并且此错误输入在 H2C 串流旁路输入接口上断言有效，同时将 SOP、EOP、`no_dma` 信号和 `sdi` 或 `mrkr-req` 信号置位，使 H2C 串流引擎向主机发送写回。

处理来自 PCIe 的数据中的错误

如果 H2C 串流引擎在来自 PCIe 的数据上遇到错误，它会在整个包中保持此错误粘滞。通过 H2C 串流数据输出的 `err` 位来向用户指示出现此错误。当 H2C 串流传出发现 PCIe 数据错误的包的最后一拍后，它还会向软件发送写回，以将该错误告知软件。

C2H 串流引擎

C2H 串流引擎 DMA 通过 C2H 描述符队列将串流包写入主机存储器内主机驱动程序提供的描述符中。

预取引擎负责计算写入包的 DMA 所需的描述符数量。缓冲器缓冲大小基于队列来固定。对于内部模式和高速缓存旁路模式，预取模块可在任意给定时间，为最多 64 个不同队列提取多达 512 个描述符。

预取引擎还可提供低时延功能特性 `pfch_en = 1`，其中引擎可在接收到包时预取描述符（数量不超过 `qdma_c2h_pfch_cfg.num_pfch`），以便后续包可避免 PCIe 时延。

QDMA 要求软件转发完整的环大小，因此 C2H 串流引擎可为接收到的所有包提取所需数量的描述符。如果描述符环中没有足够的描述符，QDMA 将停止包传输。由于性能的原因，软件需尽快转发 PIDX，以确保环中始终有足够的描述符。

C2H 串流包数据长度限制为 31 * C2H 缓冲器尺寸。可从 0xAB0 到 0xAEC 地址范围内对 C2H 缓冲器大小进行编程，欲知详情，请参阅 `qdma_v5_0_pf_registers.csv` 文件。

在旧版本（例如，2018.3）中，C2H 串流包数据长度限制为 7 * C2H 缓冲器大小。

C2H 串流描述符 (8B)

表 11：**AXI4-Stream C2H 描述符结构**

位	位宽	字段名称	描述
[63:0]	64	addr	目标地址

C2H 预取引擎

预取引擎在描述符提取引擎与 C2H DMA 写入引擎之间进行交互，使描述符与其有效载荷进行配对。

表 12：**C2H 预取上下文结构**

位	位宽	字段名称	描述
[45]	1	valid	上下文有效
[44:29]	16	sw_crdt	软件信用值 该字段是由硬件写入的，仅供内部使用。软件必须将其初始化为 0，然后将其作为只读来处理。
[28]	1	pfch	队列处于预取状态 该字段是由硬件写入的，仅供内部使用。软件必须将其初始化为 0，然后将其作为只读来处理。
[27]	1	pfch_en	启用预取
[26]	1	err	在此队列中检测到错误 在提取描述符进程期间，检测到的任何错误都将记录在此处。此操作将按队列来执行。
[25:8]	18	reserved	保留
[7:5]	3	port_id	端口 ID
[4:1]	4	buf_size_idx	缓冲器大小索引
[0]	1	bypass	C2H 旁路模式，如需使用简单旁路模式，请将该位置位

C2H 串流模式

C2H 描述符可来自描述符提取引擎或 C2H 旁路输入接口。来自描述符提取引擎的描述符始终处于高速缓存模式。预取引擎会保留描述符的顺序，以便与来自用户的 C2H 数据包配对。来自 C2H 旁路输入接口的描述符针对简单模式和高速缓存模式使用同一个接口（请注意，简单旁路和高速缓存旁路使用同一接口）。对于简单模式，用户应用会保留描述符的顺序，以便与 C2H 数据包配对。对于高速缓存模式，预取引擎会保留描述符的顺序，以便与来自用户的 C2H 数据包配对。

预取上下文有一个旁路位。如果它是 `1'b1`，用户应用会发送描述符的信用值。如果它是 `1'b0`，预取引擎会处理描述符的信用值。

描述符上下文有一个旁路位。如果它是 `1'b1`，描述符提取引擎会传出 C2H 旁路输出接口上的描述符。用户应用可对其进行转换，稍后将其环回到 C2H 旁路输入接口上的 QDMA Subsystem for PCIe。如果旁路上下文位是 `1'b0`，描述符提取引擎会将描述符直接发送到预取引擎。

有一个 1K 描述符输入缓冲器可用于存储来自旁路输入端口的描述符。深度为 1K 的缓冲器与所有 Q 共享。

每个队列支持三种模式。

- 高速缓存内部模式
- 高速缓存旁路模式
- 简单旁路模式

通过在“软件描述符上下文”和“C2H 预取上下文”中对旁路位进行置位，即可在简单旁路模式和高速缓存旁路模式之间进行选择，如下表所示。

注释：对于 C2H 串流应用，如果已在器件上高速缓存了描述符，则无需从主机提取描述符，应采用简单旁路模式。采用简单旁路模式时，请勿提供信用值来提取描述符，而需改为在描述符旁路接口上传入描述符。

注释：AXI4-Stream C2H **简单旁路模式**和**高速缓存旁路模式**都在端口（`c2h_byp_in_st_csh_*` 端口）中使用相同的旁路。

表 13：**C2H 串流模式**

	c2h_byp_in 端口	desc_byp 软件描述符上下文	C2H 预取上下文
简单旁路模式	c2h_byp_in_st_csh_*	1	1
高速缓存旁路模式	c2h_byp_in_st_csh_*	1	0
高速缓存内部模式	不适用	0	0

简单旁路模式

对于简单旁路模式，描述符提取引擎在 C2H 旁路输出接口上传出描述符。用户应用会转换描述符，并将其环回到简单模式 C2H 旁路输入接口上的 QDMA。用户应用会发送描述符的信用值，并保留描述符的顺序。

为了使简单旁路传输有效进行，需要一个预取标签（可从 QDMA IP 提取）。

在通过 C2H ST 引擎发送简单旁路队列的任何流量之前，用户应用必须请求预取标签。无效队列或非旁路队列不应使用此方法请求任何标签，因为这样会冻结未使用过的标签，从而影响性能。

在开始传输任何流量之前，必须预先保留预取标签。每个目标主机需要一个预取标签。在大多数应用中，每个主机都需要一个预取标签。采用简单旁路模式时，标签不会绑定到任何描述符环。对于共享相同预取标签的队列，数据和描述符需按相同顺序传入。对于“Simple Bypass”（简单旁路），数据和描述符均由用户控制，因此需保证顺序保持不变。

例如，在描述符输入时，如果数据串流所含数据包顺序为 Q0、Q1 和 Q2，则无法发送 Q1、Q2、Q0 等。顺序需保持相同。

用户应用使用简单旁路队列的 qid 写入 MDMA_C2H_PFCH_BYP_QID (0x1408) 寄存器，然后从 MDMA_C2H_PFCH_BYP_TAG (0x140C) 寄存器读取以检索对应的预取标签。只要当前 qid 有效，就必须使用所有 bypass_in 描述符驱动此标签。如果当前 qid 无效，则必须使用有效 qid 请求新的预取标签。

对于所有传输，必须将预取标签分配给输入端口 `c2h_byp_in_st_csh_pfch_tag[6:0]`。预取标签指向用于在预取引擎中存储活动队列的 CAM。此外，用于预取标签的 qid 必须用作所有简单旁路包的 qid。将该 qid 分配给 `s_axis_c2h_ctrl_qid`。

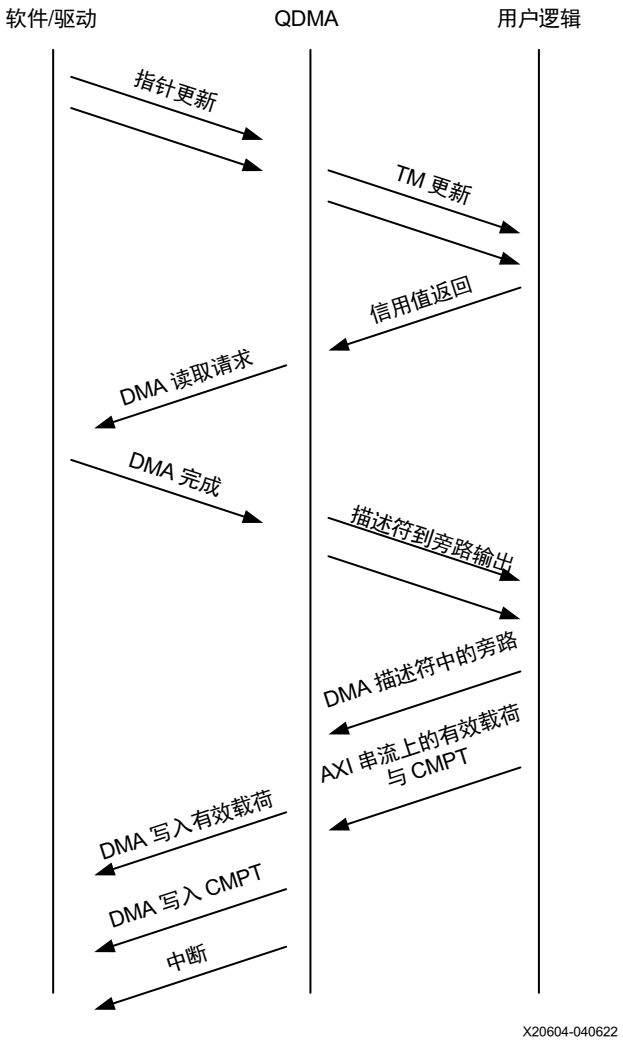
提取预取标签的步骤如下所示：

1. 软件指示信息：
   a. 初始化队列 (`qid`)。
   b. 使用有效的 `qid` 写入 MDMA_C2H_PFCH_BYP_QID 0x1408。
   c. 读取 MDMA_C2H_PFCH_BYP_TAG 0x140C 以获取预取标签。
   d. 预取标签和用于提取标签的 `qid` 应用于所有简单旁路包。此信息必须传达给用户端。
2. 用户端：
   a. 将用于提取标签的 `qid` 分配给 `s_axis_c2h_ctrl_qid`。
   b. 将包传输的实际 `qid` 分配给 `s_axis_c2h_cmpt_ctrl_qid`。
   c. 向 `c2h_byp_in_st_csh_pfch_tag` 分配预取标签值。
   d. 将包传输的实际 qid 分配给 `c2h_byp_in_st_csh_qid`。

   注释：只要原始 qid 有效，`c2h_byp_in_st_csh_pfch_tag[6:0]` 端口就可以具有相同的 prefetch_tag。

下面显示的简单旁路流程不包括“prefetch_tag”的提取。

图 9：**C2H 简单旁路模式流程**



注释：描述符旁路输入、数据有效载荷与完成包之间不需要特定顺序。

如果已有描述符，则无需更新指针或提供信用值。而是在描述符旁路接口中传入描述符，然后发送数据和完成 (CMPT) 包。

高速缓存旁路模式

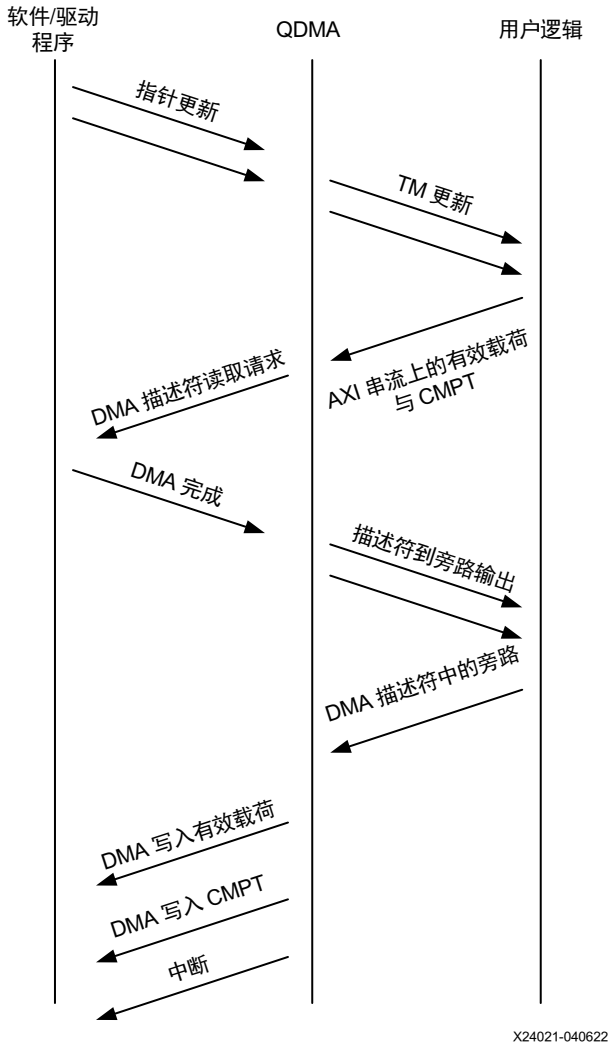
对于高速缓存旁路模式，描述符提取引擎会在 C2H 旁路输出接口上传出描述符。用户应用会转换描述符，并将其环回到高速缓存模式 C2H 旁路输入接口上的 QDMA。预取引擎会发送描述符的信用值，并保留描述符的顺序。

对于高速缓存内部模式，描述符提取引擎会将描述符发送至预取引擎。预取引擎会传出描述符的信用值并保留描述符的顺序。在这种情况下，描述符不会在 C2H 旁路输出上发出，也不会在 C2H 旁路输入接口上返回。采用高速缓存内部模式时，预取标签由 IP 在内部维护。

采用高速缓存旁路模式或高速缓存内部模式时，可以开启预取模式，这样会预取描述符，从而显著降低传输时延。启用预取模式后，用户应用无法在 QDMA 描述符信用值输入端口中将信用值作为输入发送。所有队列的信用值都将由预取引擎进行维护。

采用高速缓存旁路模式时，预取标签由 IP 在内部维护。信号 `c2h_byp_out_pfch_tag[6:0]` 应作为输入 `c2h_byp_in_st_csh_pfch_tag[6:0]` 进行环回。预取标签指向用于在预取引擎中存储活动队列的 CAM。

图 10：**C2H 高速缓存旁路模式流程**



注释：有效载荷与完成包之间不需要特定顺序。

相关信息

QDMA 描述符旁路输入端口
QDMA 描述符旁路输出端口
QDMA 描述符信用值输入端口
队列状态端口

C2H 串流包类型

以下描述了多种不同的 C2H 串流包。

常规包

常规 C2H 包既包含数据包又包含完成 (CMPT) 包。这两种包为一对一匹配。

常规 C2H 数据包可包含多个节拍。

- `s_axis_c2h_ctrl_qid` = C2H 描述符队列 ID。
- `s_axis_c2h_ctrl_len` = 包的长度。
- `s_axis_c2h_mty` = 在最后一个节拍中应设置空字节。
- `s_axis_c2h_ctrl_has_cmpt` = `1'b1`。该数据包具有一个对应的 CMPT 包。

常规 C2H CMPT 包为一个节拍。

- `s_axis_c2h_cmpt_ctrl_qid` = 包的完成队列 ID。这可能与 C2H 描述符 QID 不同。
- `s_axis_c2h_cmpt_ctrl_cmpt_type` = `HAS_PLD`。该完成包具有一个对应的数据包。
- `s_axis_c2h_cmpt_ctrl_wait_pld_pkt_id` = 此完成包必须等待具有此 ID 的数据包发送后，才能发送 CMPT 包。

当用户应用发送数据包时，必须计算每个包的包 ID 数量。第一个数据包的包 ID 为 1，此后每个数据包的包 ID 都会逐一递增。

对于常规 C2H 包，数据包和完成包一对一匹配。因此，`s_axis_c2h_ctrl_has_cmpt` 为 `1'b1` 的数据包的数量应该与 `s_axis_c2h_cmpt_ctrl_cmpt_type` 为 `HAS_PLD` 的 CMPT 包的数量相等。

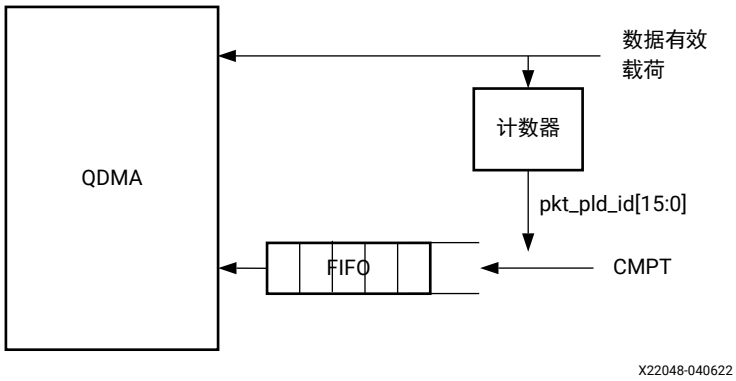
QDMA Subsystem for PCIe 的完成输入 FIFO 深度较浅，仅为 2。为了提升性能，可为完成输入添加 FIFO，如下图所示。FIFO 的深度和宽度因用例而异。宽度取决于应用的最大 CMPT 大小，深度取决于性能需求。对于 64 字节 CMPT，为了达成最佳性能，建议使用深度 512。

当用户应用发送数据有效载荷时，它会对每个包进行计数。第一个包以 `pkt_pld_id` 1 开始。第二个包的 `pkt_pld_id` 为 2，以此类推。它是一个 16 位计数器，一旦计数达到 16'hffff，它就会复位为 0，重新开始计数。

用户应用会定义 CMPT 类型。

- 如果 `s_axis_c2h_cmpt_ctrl_cmpt_type` 为 HAS_PLD，则表示 CMPT 具有对应的数据有效载荷。用户应用必须将该包的 `pkt_pld_id` 填入 `s_axis_c2h_cmpt_ctrl_wait_pld_pkt_id` 字段中。DMA 仅在传出对应的数据有效载荷包后才会传出此 CMPT。
- 如果 `s_axis_c2h_cmpt_ctrl_cmpt_type` 为 NO_PLD_NO_WAIT，则表示 CMPT 没有任何数据有效载荷，因此无需等待有效载荷。然后 DMA 将传出此 CMPT。
- 如果 `s_axis_c2h_cmpt_ctrl_cmpt_type` 为 NO_PLD_BUT_WAIT，则表示 CMPT 没有对应的数据有效载荷包。CMPT 必须等待收到特定的数据有效载荷包后才会将其传出。因此，用户应用必须将该特定数据有效载荷的 `pld_pkt_id` 填入 `s_axis_c2h_cmpt_ctrl_wait_pld_pkt_id` 字段中。在传出具有该 `pld_pkt_id` 的数据有效载荷之后，DMA 才会传出 CMPT。

图 11：**CMPT 输入 FIFO**



即时数据包

用户应用可能具有如下类型的包：只写入完成环，而没有要传输到主机的对应数据包。此类包称为即时数据包。对于即时数据包，QDMA 不会发送数据有效载荷，但会写入 CMPT 队列。即时包不耗用描述符。

对于即时数据包，用户应用仅向 DMA 发送 CMPT 包，而不会发送数据包。

以下是即时完成包的设置。没有对应的数据包。

在某些应用中，即时完成包无需等待任何数据包。但在某些应用中，它可能仍需要等待数据有效载荷包。如果完成类型为 `NO_PLD_NO_WAIT`，则无需等待任何数据包即可传出完成包。如果完成类型为 `NO_PLD_BUT_WAIT`，则完成包必须指定需要等待的数据包 ID。

- `s_axis_c2h_cmpt_user_cmpt_type` = `NO_PLD_NO_WAIT` 或 `NO_PLD_BUT_WAIT`。
- `s_axis_c2h_cmpt_ctrl_wait_pld_pkt_id` = 不会递增包计数。

标记包

QDMA 的 C2H 串流引擎为用户应用提供了一种将标记与 C2H 包一起插入 QDMA 的方法。然后，此标记通过 C2H 引擎流水线传播，并在队列状态端口接口上输出。通过在 C2H 串流包中对标记位进行置位，即可插入标记。通过在队列状态端口上对 `qsts_out_op[7:0] = 0x0`（CMPT 标记响应）位进行置位，QDMA 即可向用户应用指示标记响应。对于标记包，QDMA 不会传出有效载荷包，但仍会写入完成环。由于存在对应的标记请求，因此并不会生成所有标记响应。QDMA 有时会在遇到异常事件时生成标记响应。有关 QDMA 何时在内部生成标记响应的详细信息，请参阅以下部分。

允许用户应用将标记传入 QDMA，主要目的是为了确定将标记之前的所有流量全部清空的时机。在用户应用的关闭序列中，可使用此操作。尽管并非强制要求，但在将标记传入 QDMA 时，用户应用可以使用已置位的 `user_trig` 位来发送标记。这样，QDMA 就能够生成中断，真正确保将标记之前的所有流量全部清空。QDMA 完成引擎在收到来自用户应用的标记时会执行以下操作：

DMA 还可以选择在标记包期间不向完成环发送完成信息。在标记包期间可设置端口 `s_axis_c2h_cmpt_ctrl_no_wrb_marker`。此选项将在生成该标记包时禁用针对完成环的任何写操作。如果针对标记包将此信号置位，则 DMA 不会进行数据传输或完成传输，但会使用 `qsts_out_op[7:0]` 上的标记响应进行响应。

- 将伴随标记的完成数据发送到 C2H 串流完成环。
- 将完成数据的低 24 位发送到队列状态数据端口 `qsts_out_data[26:3]`。
- 如果已启用状态描述符，则生成状态描述符（如果在插入标记时 `user_trig` 已置位）。

- 如果已启用中断但尚未完成，则生成中断。
- 发送标记响应。如果由于正在启用中断，但尚未完成而未发送中断，则标记响应中的 `retry marker_req` 位将置位，以通知用户无法为该标记请求发送中断。如需了解有关这些字段的详细信息，请参阅队列状态端口接口描述。

标记包既包含数据包又包含 CMPT 包。这两种包一对一匹配。

以下是具有标记的数据包的设置：

- 1 拍数据
- `s_axis_c2h_ctrl_marker` = `1'b1`
- `s_axis_c2h_ctrl_len` = 数据宽度（例如，如果数据宽度是 512 位，那么该值为 64）
- `s_axis_c2h_mty` = 0
- `s_axis_c2h_ctrl_has_cmpt` = `1'b1`

以下是具有标记的 CMPT 包的设置：

- 1 拍 CMPT 包
- `s_axis_c2h_cmpt_ctrl_marker` = `1'b1`
- `s_axis_c2h_cmpt_ctrl_cmpt_type` = `HAS_PLD`
- `s_axis_c2h_cmpt_ctrl_wait_pld_pkt_id` = 此完成包必须等到发送具有此 ID 的数据有效载荷包后，才会发送 CMPT 包。
- `s_axis_c2h_cmpt_ctrl_no_wrb_marker` = `1'b0`。如果将其设置为 `1'b1`，那么就没有 CMPT 包。

即时数据包和标记包不耗用描述符；它们将写入 C2H 完成环。软件需要调整 C2H 完成环设置的大小，使其足以容纳未完成的即时包和标记包。

当 DMA 接收到标记请求而完成环已满时，将传出标记响应。但是，由于完成环已满，完成条目将被丢弃，队列将失效。

零长度包

数据包的长度可以为零。在输入时，用户需要发送 1 拍数据。零长度包会耗用描述符。QDMA 将传出 1DW 有效载荷数据。

以下是零长度包的设置：

- 1 拍数据
- `s_axis_c2h_ctrl_len` = 0
- `s_axis_c2h_mty` = 0

注释：内部模式和高速缓存旁路模式不支持零字节包。如果由于没有可用的描述符而丢弃了零字节包，QDMA 可能会挂起。简单旁路模式支持零字节包。

禁用完成包

用户应用可禁用特定包的完成。QDMA 可提供对有效载荷的直接存储器访问 (DMA)，但不会写入 C2H 完成环。用户应用只向 DMA 发送数据包，而不会发送 CMPT 包。

以下是禁用完成包的设置：

- `s_axis_c2h_ctrl_has_cmpt = 1'b0`

相关信息

QDMA 描述符旁路输出端口

处理含错误的描述符

如果在提取描述符（在预取或常规模式中）时遇到错误，QDMA 描述符引擎将描述符标记为错误。对于内部模式下的队列，C2H 串流引擎处理错误描述符的方式为不执行任何 PCIe 或 DMA 活动。改为等待错误描述符穿过流水线并在完成此操作后强制写回。对于旁路模式下的队列，由用户逻辑负责禁止发出含错误描述符的批量描述符。用户逻辑必须改为仅发送单个含错误输入的描述符，并且此错误输入在 C2H 串流旁路输入接口上断言有效，同时将 SOP、EOP、`no_dma` 信号和 `sdi` 或 `mrkr_req` 信号置位，使 C2H 串流引擎向主机发送写回。

完成引擎

完成引擎会写入 CMPT 队列中的 C2H AXI4-Stream 完成 (CMPT)。用户应用会将 CMPT 包和其它信息（包括但不限于 CMPT QID 和 CMPT_TYPE）发送至 QDMA Subsystem for PCIe。QDMA 会使用此信息来处理 CMPT 包。您可指示 QDMA 将 CMPT 包保持不变，直接写入 CMPT 队列。或者，用户应用可以指示 QDMA 在 CMPT 包中先插入某些字段（如错误和颜色），然后再将其写入 CMPT 队列。此外，用户应用可使用 CMPT 接口信号指示 QDMA 按特定方式（相对于 C2H 数据输入上的流量）对 CMPT 包的写入进行排序。尽管并非强制要求，但 CMPT 通常搭配 C2H 队列一起使用。在此情况下，CMPT 用于告知软件，C2H 的 DMA 所使用的 C2H 描述符已达到一定数量。这样软件即可回收 C2H 描述符。CMPT 也可以在没有对应 C2H DMA 操作的情况下使用，在此情况下，它被称为即时数据。

CMPT 包的用户定义部分通常需要指定传输的数据包的长度，以及数据包传输是否会耗用描述符。即时和标记类型的数据包不耗用任何描述符。用户定义的数据的具体内容由用户来判定。

完成上下文结构

完成上下文供“Completion Engine”（完成引擎）使用。

表 14：**完成上下文结构定义**

位	位宽	字段名称	描述
[256:183]	17		保留。初始化为 0。
[182:180]	3	port_id	端口 ID。完成引擎会根据此处配置的 port_id 检查其输入处接收到的事件的 port_id。如果检查失败，则丢弃输入，并在 C2H_ERR_STAT 寄存器中记录一个错误。以下是对 port_id 执行的检查： 1.s_axis_c2h_cmpt 接口上的所有事件。这包括 CMPT、即时数据、标记和 VirtIO 控制报文。 2.CMPT CIDX 指针更新（仅当更新来自 AXI 侧时才进行检查）。
[179]	1		保留。初始化为 0
[178:175]	4	baddr4_low	由于在此情况下支持的最小对齐是 64B，所以该字段必须为 0
[174:147]	28		保留。初始化为 0
[146]	1	dir_c2h	DMA 方向是 C2H。CMPT 引擎可用于管理 C2H 的完成/已用环以及 H2C 队列。 0x0：DMA 方向是 H2C 0x1：DMA 方向是 C2H
[145]	1		保留。初始化为 0

表 14：**完成上下文结构定义** (续)

位	位宽	字段名称	描述
[144]	1	dis_int_on_vf	随 VF 禁用中断
[143]	1	int_aggr	中断聚合 设置此项即可在中断聚合模式下配置 QID
[142:132]	11	vec	中断矢量编号
131	1	at	地址转换 该位用于判定队列地址是否已转换。此信息将在 CMPT 和状态写入时发送到 PCIe。 0：地址未转换 1：地址已转换
130	1	ovf_chk_dis	完成环上溢检查禁用 如果此项置位，那么 CMPT 引擎不检查在完成环中写入完成条目是否会造成环溢出。由此导致 QDMA 始终传出完成，而不会首先检查是否会造成完成环上溢，并且也不会采取通常在遇到完成环上溢场景时所采取的任何操作。转而交由软件和用户逻辑通过协商来确保不会导致完成环上溢
[129]	1	full_upd	全面更新 如果此项复位，则忽略所有字段（Completion-CIDX-update 的 CIDX 除外）。仅将 CIDX 字段从更新复制到上下文中。 如果此项置位，那么 Completion CIDX update 即可在此上下文中更新下列字段： · timer_ix · counter_ix · trig_mode · en_int · en_stat_desc
[128]	1	timer_running	如果此项置位，表明该队列上有一个定时器正在运行。这个定时器用于对 CMPT 中断进行调节。理想情况下，软件必须确保在此 QID 上没有任何定时器正在运行，然后才能关闭队列。该字段供硬件在内部使用。软件必须将其初始化为 0，然后将其作为只读来处理。
[127]	1	user_trig_pend	如果此项置位，它表示用户逻辑发起的中断正在等待生成。用户逻辑可以通过 s_axis_c2h_cmpt_ctrl_user_trig 信号来请求中断。当用户逻辑请求中断，而在此 QID 上已有另一个中断等待处理时，该位即置位。当 QDMA 收到下一个 Completion CIDX update 时，此待处理的位是否生成中断取决于完成环中是否有任何条目等待读取。该字段供硬件在内部使用。软件必须将其初始化为 0，然后将其作为只读来处理。
[126:125]	2	err	表示完成上下文存在错误。这是由硬件写入的字段。软件必须将其初始化为 0，然后将其作为只读来处理。此处指出了下列错误： 0：无错误。 1：检测到一个来自软件的错误 CIDX 更新。 2：检测到描述符错误。 3：用户逻辑在完成环已满时发送了一个完成包。
[124]	1	valid	上下文有效。
[123:108]	16	cidx	“Completion Ring Consumer Index”（完成环使用者索引）的硬件拷贝的当前值。
[107:92]	16	pidx	完成环生产者索引。这是由硬件写入的字段。软件必须将其初始化为 0，然后将其作为只读来处理。

表 14：**完成上下文结构定义** (续)

位	位宽	字段名称	描述
[91:90]	2	desc_size	完成条目大小： 0：8B 1：16B 2：32B 3：64B
[89:32]	58	baddr	完成环的 64B 对齐基址：位 [63:6]。
[31:28]	4	qsize_idx	完成环大小索引。该索引会从 16 个寄存器（偏移 0x204 :0x240）中选出大小与其它环不同的那一个环。
[27]	1	color	完成时使用的颜色位。
[26:25]	2	int_st	中断状态： 0：ISR 1：TRIG 该字段供硬件在内部使用。软件必须将其初始化为 0，然后将其作为只读来处理。 脱离复位时，硬件初始化为 ISR 状态，对触发事件不敏感。如果软件需要中断或状态写入，它必须发送一个初始完成 CIDX 更新。这导致硬件进入 TRIG 状态，因此它变为对任何触发条件都敏感。
[24:21]	4	timer_idx	定时器寄存器索引，适用于基于 TIMER 的触发模式。
[20:17]	4	counter_idx	计数器寄存器索引，适用于基于 COUNT 的触发模式。
[16:13]	4		保留。初始化为 0
[12:5]	8	fnc_id	功能 ID
[4:2]	3	trig_mode	中断和完成状态写入触发模式： 0x0：Disabled 0x1：Every 0x2：reserved 0x3：User 0x4：User_Timer 0x5：reserved
[1]	1	en_int	启用完成中断。
[0]	1	en_stat_desc	启用完成状态写入。

完成状态结构

Completion Status（完成状态）位于完成环的最后一个位置，即，完成环基址 + (完成长度大小 (8,16,32) * (完成环大小 – 1))。

为了使 QDMA Subsystem for PCIe 将完成状态写入完成环，必须在完成上下文中启用完成状态。除了影响中断外，在完成上下文中定义的触发模式也会对完成状态的写入进行调制。根据中断/状态调制，发生以下任一状况时，即可写入完成状态：

1. CMPT 包写入完成环。
2. 收到来自软件的 CMPT-CIDX 更新，表明有更多完成条目等待读取。
3. 与各 CMPT QID 关联的定时器到期，并在基于定时器的触发模式下进行编程。

表 15：**AXI4-Stream 完成状态结构**

位	位宽	字段名称	描述
[63:37]	27		保留
[36:35]	2	error	错误。 0x0：无错误 0x1：收到错误的 CIDX 更新 0x2：描述符错误 0x3：CMPT 环上溢错误
[34:33]	2	int_state	中断状态。 0：ISR 1：TRIG
[32]	1	color	颜色状态位
[31:16]	16	cidx	使用者索引 (RO)
[15:0]	16	pidx	生产者索引

完成条目结构

完成 (CMPT) 环条目的大小为 512 位。这包括用户定义的数据、可选错误位和可选颜色位。用户定义的数据有 4 种大小选择：8B、16B、32B 和 64B。CMPT 条目中可选错误位和颜色位的位置均可单独配置。这是通过在编译 QDMA Subsystem for PCIe 时，使用 Vivado® IDE IP 自定义选项指定这些字段的位置来完成的。有七个颜色位位置选项和八个错误位位置选项可选。位置是按距离完成条目的 LSB 位的偏移量的形式来指定的。

用户应用将完成包驱动到 QDMA Subsystem for PCIe 中时，它会在接口处提供一个 `s_axis_cmpt_ctrl_col_idx[2:0]` 值和一个 `s_axis_cmpt_ctrl_err_idx[2:0]` 值。这些索引供 QDMA Subsystem for PCIe 用于使用颜色位和错误位的正确位置。例如，如果 `s_axis_cmpt_ctrl_col_idx[2:0] = 0` 且 `s_axis_cmpt_ctrl_err_idx[2:0] = 1`，那么 QDMA Subsystem for PCIe 使用“C2H Stream Completion Color bits”（C2H 串流完成颜色位）位置选项 0 表示颜色位置，并使用“C2H Stream Completion Error bits”（C2H 串流完成错误位）位置选项 1 表示错误位置。颜色或错误信号的索引为 7 表示更新完成条目时，DMA 不会更新对应的颜色位或错误位（忽略这些字段）。在 Vivado® IDE 的“PCIe DMA”选项卡中设置 C2H 串流完成位选项。

在编译时使用的错误位和颜色位的位置值可供软件用于读取 MMIO 寄存器。有 7 个寄存器可供读取：QDMA_C2H_CMPT_FORMAT (0xBC4) 到 QDMA_GLBL_ERR_MASK (0x24C)。其中每个寄存器中都各有一个颜色位位置和一个错误位位置。

- C2H 串流完成位的对应颜色位位置的选项 0 和对应错误位位置的选项 0 均可通过 QDMA_C2H_CMPT_FORMAT_0 寄存器来使用。
- C2H 串流完成位的对应颜色位位置的选项 1 和对应错误位位置的选项 1 均可通过 QDMA_C2H_CMPT_FORMAT_1 寄存器来使用。
- 以此类推。

表 16：**完成条目结构**

名称	大小（位）	索引
针对 64 字节的用户定义的位设置	510-512	取决于是否存在颜色位和错误位。
针对 32 字节的用户定义的位设置	254-256	取决于是否存在颜色位和错误位。
针对 16 字节的用户定义的位设置	126-128	取决于是否存在颜色位和错误位。
针对 8 字节的用户定义的位设置	62-64	取决于是否存在颜色位和错误位。

表 16：**完成条目结构** (续)

名称	大小（位）	索引
错误		错误位的位置由寄存器 QDMA_C2H_CMPT_FORMAT_0 (0xBC4) 到 QDMA_C2H_CMPT_FORMAT_6 (0xBDC) 定义。这些寄存器可显示在 IP 生成期间用户定义的颜色位的位置。您可根据输入 CMPT 端口 `s_axis_c2h_cmpt_ctrl_err_idx[2:0]` 索引到该寄存器内。您可选择不包含错误位（索引值 7）。在此情况下，用户定义的数据会占用该空间
颜色		颜色位的位置由寄存器 QDMA_C2H_CMPT_FORMAT_0 (0xBC4) 到 QDMA_C2H_CMPT_FORMAT_6 (0xBDC) 来定义。这些寄存器可显示在 IP 生成期间用户定义的颜色位的位置。您可根据输入 CMPT 端口 `s_axis_c2h_cmpt_ctrl_col_idx[2:0]` 索引到该寄存器内。如果不包含颜色位（索引值 7），则用户定义的数据会占用该空间。

相关信息

QDMA_CSR (0x0000)
“PCIe DMA”选项卡

完成输入包

用户应用会向 QDMA 发送 CMPT 包。

CMPT 包和数据包不要求一对一的匹配。例如，即时数据包只有 CMPT 数据包，而没有数据包。禁用完成包只有数据包，没有 CMPT 包。

每个 CMPT 包都有一个 CMPT ID。此 ID 即关联 CMPT 队列的 ID。每个 CMPT 队列都有一个 CMPT 上下文。驱动程序会设置 C2H 描述符队列到 CMPT 队列的映射。也可能有与 C2H 队列无关联的 CMPT 队列。

以下是来自用户应用的 CMPT 包。

表 17：**CMPT 输入包**

名称	大小	索引
数据	512 位	[511:0]

CMPT 包有 4 种类型：8B、16B、32B 或 64B。它只有一次大小为 512 位的数据输送。

完成状态/中断调节

QDMA Subsystem for PCIe 提供了一种按队列调节完成中断和完成状态写操作的方法。该软件可以为每个队列选择五种模式之一。为队列选择的模式存储在该队列的完成环上下文的 QDMA Subsystem for PCIe 中。为队列选择一种模式后，驱动程序在向 QDMA 发送完成环 CIDX 更新时始终可以选择另一种模式。

完成中断调节由完成引擎处理。完成引擎会存储所有队列的完成环上下文。每个队列的中断和完成状态的发送都可以单独启用或禁用，此信息存储在完成环上下文中。请注意，此处描述的模式不仅可以调节中断，还可以调节完成状态写操作。此外，由于可以为每个队列单独启用/禁用中断和完成状态写操作，因此仅当在该队列的完成上下文中启用中断/完成状态时，这些模式才有效。

针对每个队列，QDMA Subsystem for PCIe 只保留一个未完成的中断。即使该模式已满足发送中断的所有其它条件，QDMA 也会强制执行此策略。QDMA Subsystem for PCIe 根据从驱动程序接收该队列的 CIDX 更新来判定中断已处理完成。

所有中断调节模式遵循的基本策略是：当队列没有未完成的中断时，QDMA Subsystem for PCIe 会持续监控该模式要满足的触发条件。一旦满足条件，就会传出中断。当 QDMA 子系统等待要处理的中断时，会时刻警惕要满足的中断条件并记住它们。当收到 CIDX 更新时，QDMA 子系统会评估是否仍然满足条件。如果仍然满足，则传出另一个中断。如果不满足，则不会传出中断，并且 QDMA 会重新开始监控要满足的条件。

请注意，QDMA 子系统提供的中断调节模式不一定精确。因此，如果用户应用发送两个带有发送中断指示的 CMPT 包，并不表示必然会生成两个中断。此行为的主要原因是当驱动程序被中断以读取完成环时，它没有义务准确读取到生成中断的完成。因此，驱动程序可能不会读取到中断完成，或者如果有要读取的有效描述符，它甚至可能会读取至中断完成描述符范围之外。此行为要求 QDMA Subsystem for PCIe 每次从驱动程序接收 CIDX 更新时都重新评估触发条件。

每种模式的详细描述如下：

- TRIGGER_EVERY：这种模式在中断频率方面是最高的。这种模式的策略是每当完成引擎确定完成环中存在未读的完成描述符时就发送中断。
- TRIGGER_USER：QDMA Subsystem for PCIe 提供了一种方法，可向子系统发送 CMPT 包，并在其中包含如下指示，当该子系统完成向主机发送包后，就传出中断。这使用户应用能够在 TRIGGER_USER 模式置位的情况下执行中断调节。
- TRIGGER_USER_COUNT：这种模式使 QDMA Subsystem for PCIe 对两个触发器中的任何一个都保持敏感。用户会将其中任一触发器随 CMPT 包一起发送。另一个触发器的条件是硬件发现完成环中未读完成条目数超出编程阈值。该阈值可由驱动程序按队列进行编程。检测到上述任一触发器时，QDMA 会评估是否发送中断。如前所述，除了满足触发器条件外，还必须满足其它条件才能发送中断。
- TRIGGER_USER_TIMER：采用此模式时，QDMA Subsystem for PCIe 对两个触发器中的任何一个保持敏感。用户会将其中任一触发器随 CMPT 包一起发送。另一个触发器的条件是与 CMPT 队列关联的定时器到期。定时器的周期可由驱动程序按队列进行编程。检测到上述任一触发器时，QDMA 会评估是否发送中断。如前所述，除了满足触发器条件外，还必须满足其它条件才能发送中断。如需了解更多信息，请参阅 完成定时器。
- TRIGGER_USER_TIMER_COUNT：这种模式使 QDMA Subsystem for PCIe 对三个触发器中的任何一个都保持敏感。第一个触发器随 CMPT 包一起由用户发送。第二个触发器的条件是与 CMPT 队列关联的定时器到期。定时器的周期可由驱动程序按队列进行编程。第三个触发器的条件是硬件发现完成环中未读完成条目数超出编程阈值。该阈值可由驱动程序按队列进行编程。检测到上述任一触发器时，QDMA 会评估是否发送中断。如前所述，除了满足触发器条件外，还必须满足其它条件才能发送中断。
- TRIGGER_DIS：采用此模式时，QDMA Subsystem for PCIe 不会发送完成中断，无论是否针对给定队列启用中断都是如此。在这种情况下，驱动程序读取完成环的唯一方法是定期轮询该环。如果设置了这种模式，驱动程序必须使用完成环中提供的颜色位功能，因为此模式还会禁用将任何完成状态描述符发送到完成环。

在 TRIGGER_USER_TIMER_COUNT 模式下对队列进行编程时，软件可以选择不读取完成环中由中断（或完成状态写操作）所指示的所有可用完成条目。在这种情况下，软件可以提供完成 CIDX 更新以执行部分读取。这是可行的，因为 QDMA 将在接收到 CIDX 更新后重新启动定时器，一旦定时器到期，将生成另一个中断。此过程将重复直至读取完所有完成条目为止。

但是，在 TRIGGER_EVERY、TRIGGER_USER 和 TRIGGER_USER_COUNT 模式下，仅当 QDMA 接收到来自用户逻辑的完成包时，才会发送中断。对于用户逻辑生成的每个中断发送请求，QDMA 都会发送一个仅且一个中断。因此，在这种情况下，如果软件并未读取可供读取的所有完成条目，并且用户逻辑不再发送任何完成请求中断，则 QDMA 不会再生成任何中断。这将导致剩余的完成无限期地留在完成环中。为避免发生这种情况，在 TRIGGER_EVERY、TRIGGER_USER 和 TRIGGER_USER_COUNT 模式下，软件必须读取完成环中由中断（或完成状态写操作）所指示的所有完成条目。

以下是不同模式的流程图。这些流程图反映的是完成引擎的视角。完成包从用户逻辑传入，并写入完成环。软件 (SW) 更新是指从软件发送到硬件的完成环 CIDX 更新。

图 12：**EVERY 模式的流程图**

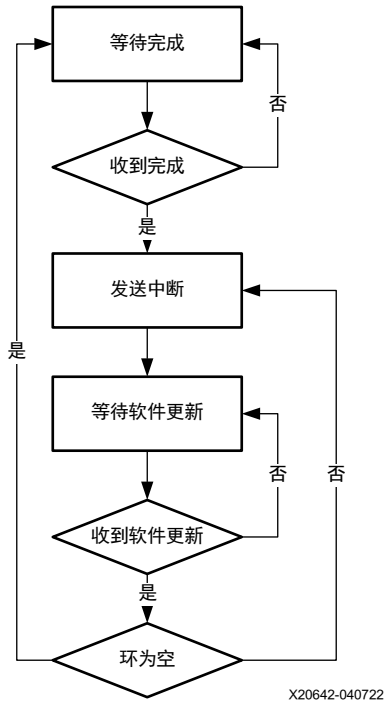


图 13：**USER 模式的流程图**

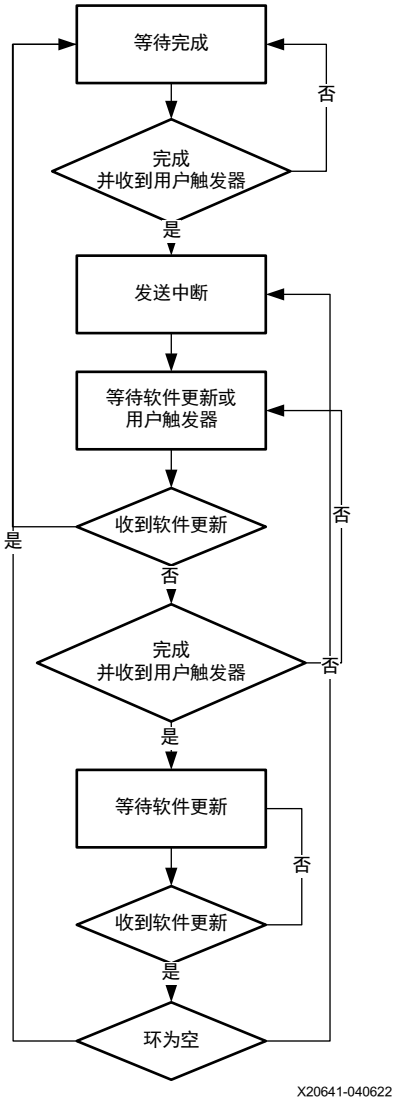


图 14：**USER_COUNT 模式的流程图**

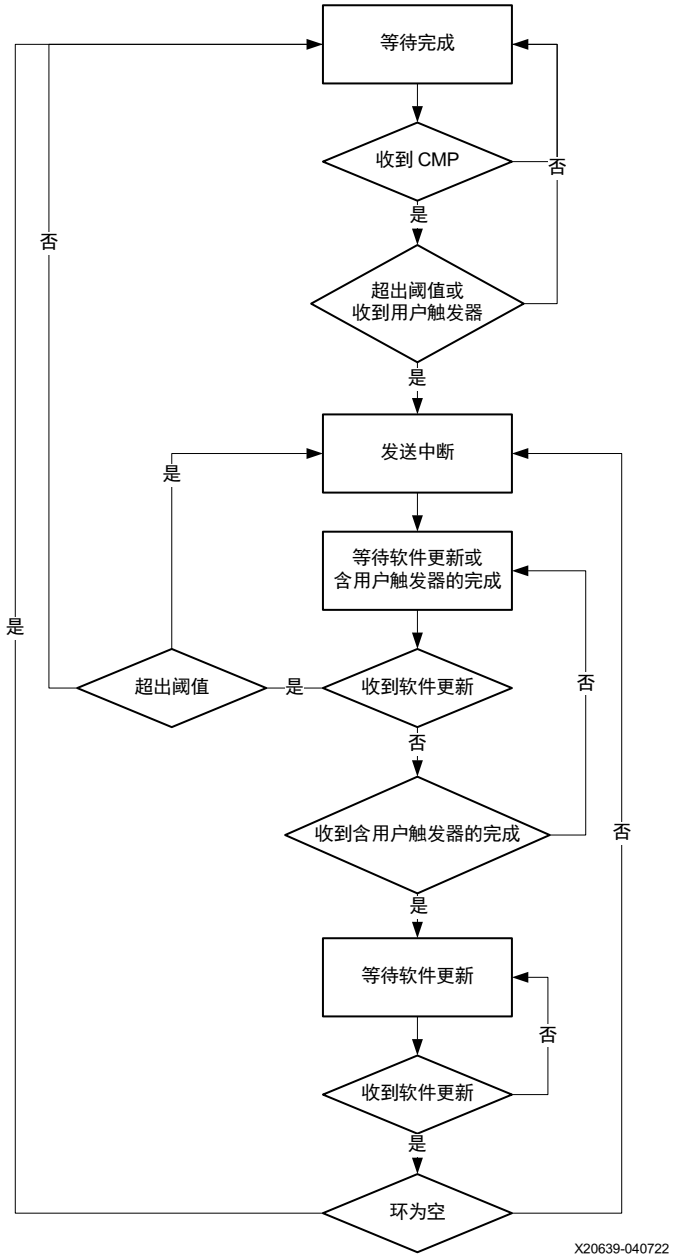


图 15：**USER_TIMER 模式的流程图**

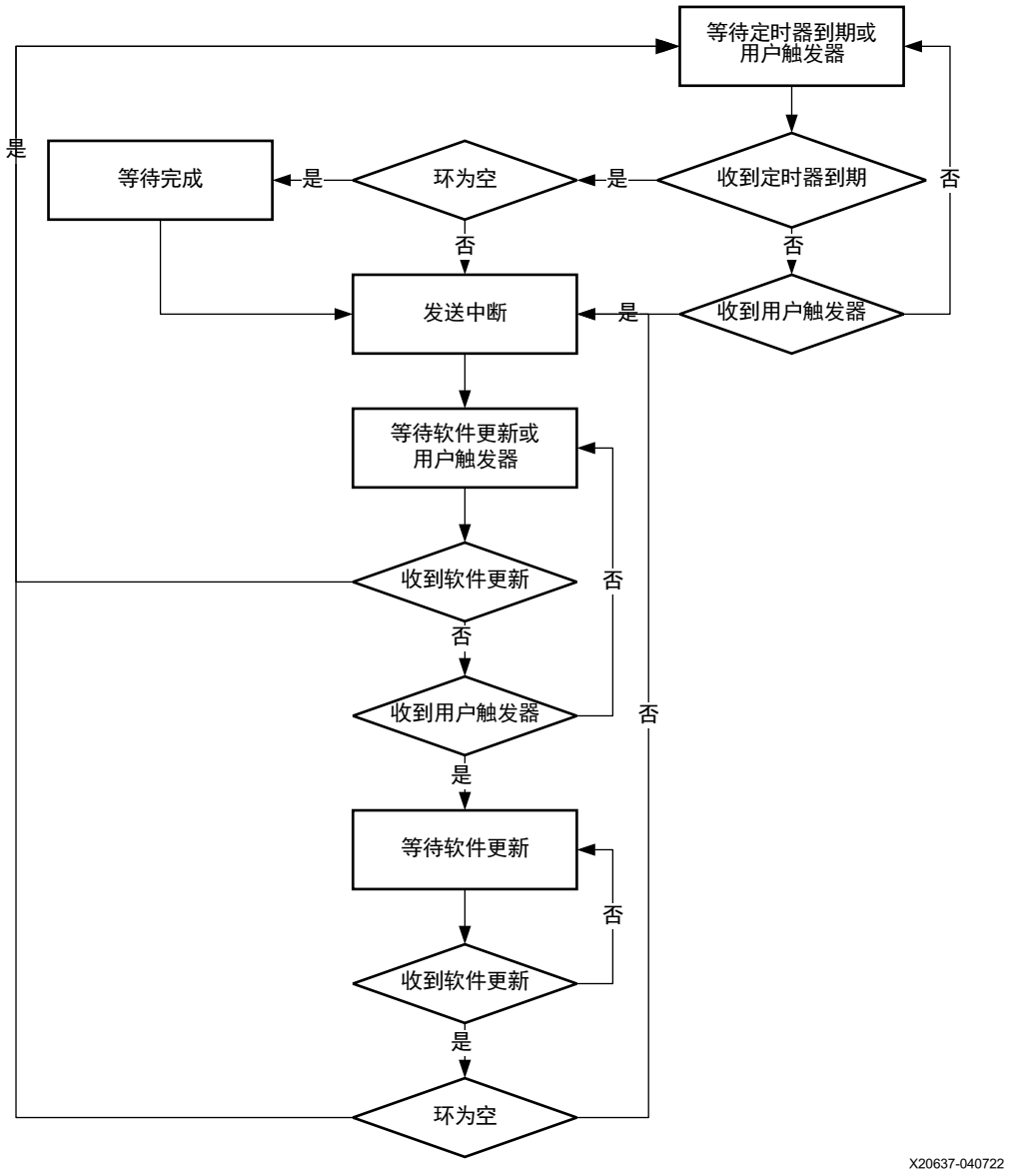
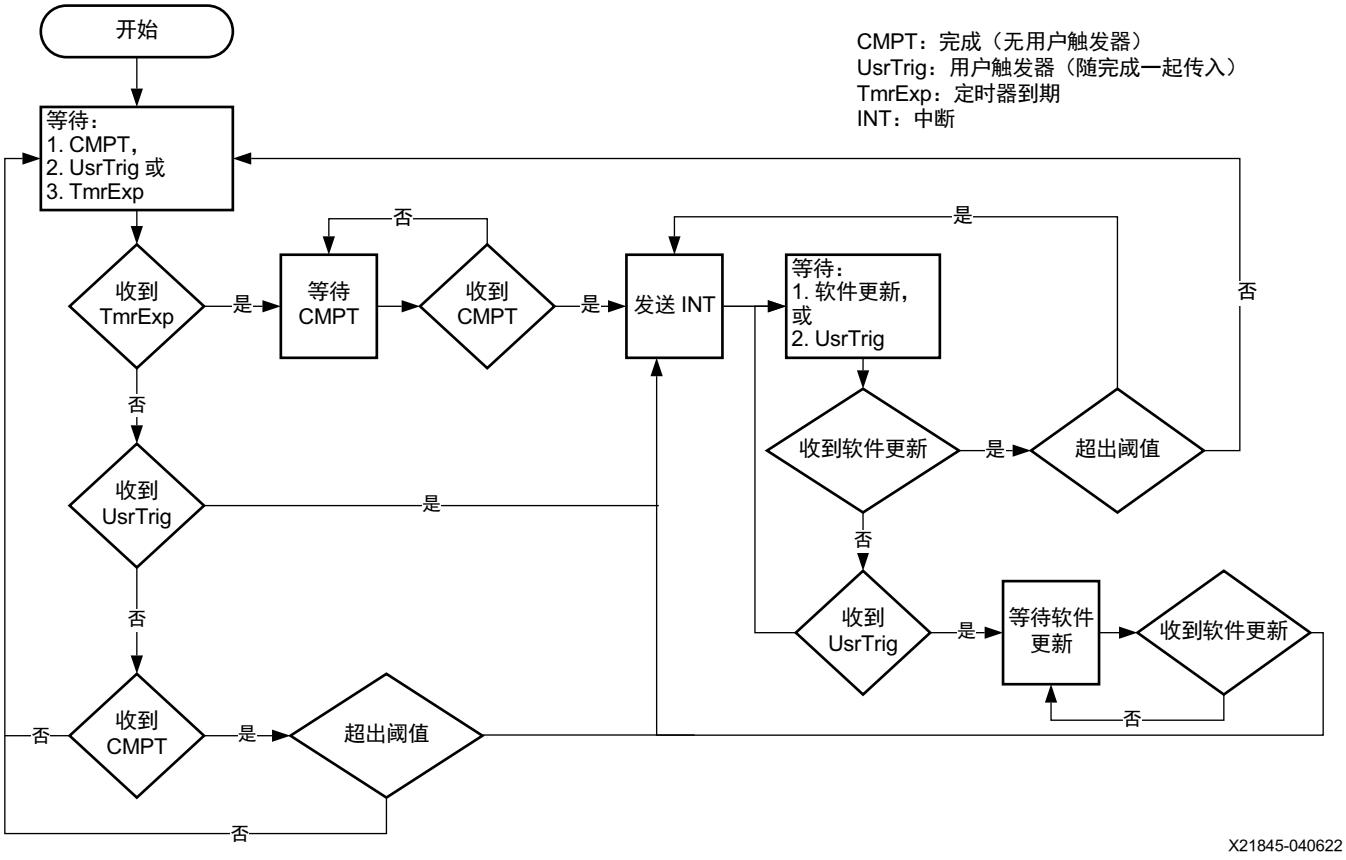


图 16：**USER_TIMER_COUNT 模式的流程图**



完成定时器

“Completion Timer”（完成定时器）引擎在完成上下文中支持定时器触发模式。它支持 2048 个队列，每个队列都有自己的定时器。定时器到期时，就会向完成模块发送定时器到期信号。如有多个定时器同时到期，则以循环方式发送信号。

参考定时器

参考定时器基于定时器节拍。寄存器 QDMA_C2H_INT (0xB0C) 用于定义定时器节拍的值。16 个寄存器 QDMA_C2H_TIMER_CNT (0xA00-0xA3c) 的定时器计数均基于定时器节拍。完成上下文中的 `timer_idx` 是 16 个 QDMA_C2H_TIMER_CNT 寄存器的索引。每个队列均可选择自己的 `timer_idx`。

处理异常事件

无效队列上的 C2H 完成

当 QDMA 在队列上接收到完成，并且此队列中具有无效上下文（以 C2H CMPT 上下文中的 Valid 位表示），那么此完成将被静默丢弃。

完整环上的 C2H 完成

完成环中的完成条目的最大数量比完成环中条目总数少 2。C2H 完成上下文中含有 PIDX 和 CIDX。这样即可支持 QDMA 计算完成环中的完成数。当 QDMA 在已满的队列上接收到完成时，QDMA 可采取以下操作：

- 使该队列的 C2H 完成上下文失效。
- 标记含错误的 C2H 完成上下文。
- 丢弃此完成。
- 发送标记为错误的“Status Descriptor”（状态描述符），前提是启用此操作。
- 如果启用且并非未完成，则发送中断。
- 发送含错误的标记响应。
- 将错误记录到 C2H 错误状态寄存器中。

含描述符错误的 C2H 完成

当 QDMA C2H 引擎遇到描述符错误时，在 C2H 完成引擎的上下文中会采取以下操作：

- 使该队列的 C2H 完成上下文失效。
- 标记含错误的 C2H 完成上下文。
- 将完成传出至完成环。它会被标记为错误。
- 如果启用且未完成，则发送含错误标记的状态描述符。
- 如果启用且并非未完成，则发送中断。请注意，完成引擎只能发送未完成的中断和/或状态描述符。这样做的意义是，如果遇到描述符错误时，中断恰好未完成，那么就不会向软件发送队列中断。尽管如此，如果软件未对错误加以掩码，那么仍会记录错误，且仍会发送错误中断
- 发送含错误的标记响应。

含无效 CIDX 的 C2H 完成

C2H 完成引擎具有相应逻辑用于检测 CIDX 更新中的 CIDX 值是否指向完成环中的空位置。检测到此类错误时，C2H 完成引擎会执行以下操作：

- 使完成上下文失效。
- 标记含错误的完成上下文。
- 将错误记录到 C2H 错误状态寄存器中。

端口 ID 不匹配

CMPT 上下文指定相应端口，基于此端口的 CMPT 应适用于该 CMPT 队列。如果传入 CMPT 中的 `port_id` 与 CMPT 上下文中的 `port_id` 不同，那么 CMPT 引擎会将传入 CMPT 作为错误定向的 CMPT 来处理并将其丢弃。它还会记录错误。请注意，发生 `port_id` 错误匹配时，CMPT 队列不会失效。

Bridge

Bridge 核是 AXI4 与 PCI Express 集成块之间的接口。它包含存储器映射 AXI4 到 AXI4-Stream Bridge 以及 AXI4-Stream Enhanced Interface Block for PCIe。存储器映射 AXI4 到 AXI4-Stream Bridge 包含一个寄存器块和两个功能半桥，称为 Slave Bridge 和 Master Bridge。

- Slave Bridge 作为从器件连接到 AXI4 Interconnect，以处理任何发出的 AXI4 主接口读取或写入请求。

- Master Bridge 作为主器件连接到 AXI4 Interconnect，以处理 PCIe 生成的读取或写入 TLP。
- 寄存器块包含在 Bridge 核中使用的寄存器，用于将使用 AXIBAR 参数提供的 AXI4 存储器映射 (MM) 地址范围动态映射到 PCIe 范围的地址。

该核使用一组中断来检测和标记错误情况。

相关信息

Bridge 寄存器空间

Slave Bridge

Slave Bridge 可终止来自 AXI4 主器件（如处理器）的存储器映射 AXI4 传输事务。Slave Bridge 提供一种方式，可将 AXI4 存储器映射地址域中映射的地址转换为 PCIe 的域地址。根据配置的“最大有效载荷大小”设置，将针对 Slave Bridge 的写入传输事务转换为一个或多个 `MemWr` TLP，随后将这些 TLP 传递给 Integrated Block for PCI Express。当远程 AXI 主接口向 Slave Bridge 发起读取传输事务时，会捕获读取地址和限定符，将 `MemRd` 请求 TLP 传递给核，并且启动完成超时定时器。通过核接收的完成将与暂挂的读取请求相关联，读取数据则将返回到 AXI4 主接口。Slave Bridge 最多可支持 32 个 AXI4 写入请求和 32 个 AXI4 读取请求。

注释：如果从接口读取和写入有效，那么 IP 会优先读取而后写入。建议您执行适当的仲裁，在各读取之间保留间隔以便写入能穿越其中。

BDF 表

AXI 地址的地址转换是基于 BDF 表编程（0x2420 到 0x2434）来完成的。这些 BDF 表格条目均可通过 AXI CSR 从接口 `s_axil_csr_*` 进行编程。其中提供了 8 个窗口，类似于 PCIe 总线上的 8 个 BAR。BDF 表格编程中的每个条目均表示 1 个窗口。如果用户需 2 个窗口，则需对 2 个条目进行编程，以此类推。

BDF 表格编程存在一些限制。

1. 在对 BDF 进行编程前，所有 PCIe Slave Bridge 数据传输都必须保持禁止。
2. 每个 BDF 表格条目都有 6 个寄存器。全部 6 个寄存器都必须完成编程才能构成 1 个有效的条目。即使部分寄存器包含 0 值，您仍需对该寄存器中的 0 值进行编程。
3. 全部 6 个寄存器都需按顺序进行编程才能使对应条目有效。顺序如下所示。
    a. 0x2420
    b. 0x2424
    c. 0x2428
    d. 0x242C
    e. 0x2430
    f. 0x2434

BDF 表格条目起始地址 = 0x2420 + (0x20 * i)，其中 i = 表格条目编号。

保护

通过 TrustZone 使用 AXI4 `prot` 字段有助于为 BAR 内的不同窗口指定保护级别。从 PMC 执行的任何访问操作都包含 `a*prot[1]=0`，因此将享有完全访问权。

对于 BDF 空间，保护域 ID 本身存储在 BDF 表中。如果传入请求包含 `a*rpot[1]=0`，那么将为其授予完整访问权。含 `a*prot[1]=1` 的请求仅有权访问保护级别较低的 BDF 条目。

下表对此行为进行了描述：

表 18：**AXI BAR 保护级别**

访问类型	BDF 表值 (prot[2:0])	a*prot[2:0] 中的值（AXI 接口）	操作
安全访问	3'hXXX	3'hX0X (bit 1=0)	允许
对安全条目执行非安全访问	3'hX0X	3'hX1X (bit 1=1)	不允许
对安全级别较低的条目执行非安全访问	3'hX1X	3'hX1X (bit 1=1)	如果在 a*prot 与 BDF 条目之间 bit [2] 与 bit [0] 相匹配，则允许

地址转换

在 IP 配置期间可通过选中“No Address Translation”（无地址转换）选项来关闭地址转换。选中该选项时，会为从数据传输提供 1 个完整的 64 位 BAR 空间。您必须按需设置所有地址转换。如果未选中“No Address Translation”（无地址转换），那么 DMA 将执行地址转换。

每个 64 位 BAR 空间均拆分为 8 个窗口 (window)，因此有 8 个窗口空间可用于地址转换。Slave Bridge 传输的地址转换分 2 步完成。

1. 上半位元的地址转换在 GUI 配置中进行编程。在 IP integrator 画布中，在“地址编辑器”选项卡中编辑上半位元的地址转换。
2. 下半位元的地址转换在 BDF 表中进行编程。

从地址转换示例

示例 1：BAR 大小为 64 KB，含 1 个 4 KB 大小的窗口

窗口 0：大小为 4 KB，含 0xF 的地址转换（对应位 [15:12]）。

1. Vivado IP 配置的“AXI BAR”选项卡中的选项如下：
   - AXI BAR 大小 64K：“0xFFFF bits [15:0]”
   - 位 [63:16] 的地址转换可在 GUI 中设置。在此示例中 [63:16] = 0x0
   - 设置间隙基址：“0x0000_0000_0000_0000”
   - 设置间隙高位地址：“0x0000_0000_0000_FFFF”
2. BDF 表编程：
   - 针对 1 个窗口编程 1 个条目
   - 窗口大小 = AXI BAR 大小/8 = 64K / 8 = 0x1FFF = 8 KB（13 个位）。每个窗口的最大大小为 8 KB。
   - 在此示例中，窗口大小为 4K，因此 0x1 将于 0x2430 位 [25:0] 处进行编程。
   - 位 [15:13] 的地址转换将于 0x2420 和 0x2424 处进行编程。
   - 在此示例中，位 [15:13] 的地址转换设为 0x7。

表 19：**BDF 表编程**

编程值	寄存器
0x0000_E000	地址转换值低位

表 19：**BDF 表编程** (续)

编程值	寄存器
0x0	地址转换值高位
0x0	PASID/保留
0x0	[11:0]：功能编号
0xC0000001	[31:30] 读/写访问权限 [29]：R0 访问错误 [28:26] 保护 ID [25:0] 窗口大小 （[25:0]*4K = 窗口的实际大小）
0x0	保留

对于此示例，从地址 0x0000_0000_0000_0100 将转换为 0x0000_0000_0000_E100。

示例 2：BAR 大小为 64 KB，含 1 个 8 KB 大小的窗口

窗口 0：大小为 8 KB，含 0x6 ('b110) 的地址转换（对应位 [15:13]）。

1. Vivado IP 配置的“AXI BAR”选项卡中的选项如下：
   - AXI BAR 大小 64K：“0xFFFF bits [15:0]”
   - 位 [63:16] 的地址转换可在 GUI 中完成。在此示例中 [63:16] = 0x0
   - 设置间隙基址：“0x0000_0000_0000_0000”
   - 设置间隙高位地址：“0x0000_0000_0000_FFFF”
2. BDF 表编程：
   - 针对 1 个窗口编程 1 个条目。
   - 窗口大小 = AXI BAR 大小/8 = 64K / 8 = 0x1FFF = 8 KB（13 个位）。每个窗口的最大大小为 8 KB。
   - 在此示例中，窗口大小为 8K，因此 0x2 将于 0x2430 位 [25:0] 处进行编程。
   - 位 [15:13] 的地址转换将于 0x2420 和 0x2424 处进行编程。
   - 在此示例中，位 [15:13] 的地址转换设为 0x6 ('b110)。

表 20：**BDF 表编程**

偏移	编程值	寄存器
0x2420	0x0000_C000	地址转换值低位
0x2424	0x0	地址转换值高位
0x2428	0x0	PASID/保留
0x242C	0x0	[11:0]：功能编号
0x2430	0xC0000002	[31:30] 读/写访问权限 [29]：R0 访问错误 [28:26] 保护 ID [25:0] 窗口大小 （[25:0]*4K = 窗口的实际大小）
0x2434	0x0	保留

对于此示例，从地址 0x0000_0000_0000_0100 将转换为 0x0000_0000_0000_C100。

示例 3：BAR 大小为 32 GB，含 4 个不同大小的窗口

- 窗口 0：大小为 4 KB，含 0xAAAAA 的地址转换（对应位 [31:12]）。
- 窗口 1：大小为 4 GB，窗口上不含地址转换。
- 窗口 2：大小为 64 KB，含 0xBBBB 的地址转换（对应位 [31:16]）。
- 窗口 3：大小为 1 GB，含 11111 的地址转换（对应位 [34:30]）。

1. Vivado IP 配置的“AXI BAR”选项卡中的选项如下：
   - AXI BAR 大小 32G：“0x7_FFFF_FFFF bits [34:0]”。
   - 位 [63:35] 的地址转换可在 GUI 中进行编程。在此示例中 [63:36] = 0xAB。
   - 设置间隙基址：“0x0000_0AB0_0000_0000”。
   - 设置间隙高位地址：“0x0000_0AB7_FFFF_FFFF”。
2. BDF 表编程：
   - 窗口大小 = AXI BAR 大小/8 = 32 GB / 8 = 0xFFFF_FFFF = 4 GB（32 个位）。每个窗口的最大大小为 4 GB。
   - 针对 4 个窗口编程 4 个条目：
     - BDF 表格条目 0 起始地址为 0x2420。
     - BDF 表格条目 1 起始地址为 0x2440。
     - BDF 表格条目 2 起始地址为 0x2460。
     - BDF 表格条目 3 起始地址为 0x2480。
   - 窗口 0 大小为 4 KB。
     - 将 0x1 编程为 0x2430 位 [25:0]。
     - 位 [34:32] 的地址转换将于 0x2420 和 0x2424 处进行编程。
     - 将 0x0000_0000 编程为 0x2420。
     - 将 0x0000_0007 编程为 0x2424
   - 窗口 1 大小为 4 GB。
     - 将 0x10_0000 编程为 0x2450 位 [25:0]。
     - 无地址转换，因为此窗口将使用所有地址位。
     - 将 0x0000_0000 编程为 0x2440。
     - 将 0x0000_0000 编程为 0x2444
   - 窗口 2 大小为 64 KB。
     - 将 0x10 编程为 0x2470 位 [25:0]。
     - 位 [34:32] 的地址转换将于 0x2460 和 0x2464 处进行编程。
     - 将 0x0000_0000 编程为 0x2460
     - 将 0x0000_00005 编程为 0x2464

- 窗口 3 大小为 1 GB。
  - 将 0x4_0000 编程为 0x2490 位 [25:0]。
  - 位 [34:30] 的地址转换将于 0x2480 和 0x2484 处进行编程。
  - 将 0x0000_0000 编程为 0x2480。
  - 将 0x0000_0003 编程为 0x2484

表 21：**BDF 表格编程条目 0**

偏移	编程值	寄存器
0x2420	0x0000_0000	地址转换值低位
0x2424	0x7	地址转换值高位
0x2428	0x0	PASID/保留
0x242C	0x0	[11:0]：功能编号
0x2430	0xC0000001	[31:30] 读/写访问权限 [29]：R0 访问错误 [28:26] 保护 ID [25:0] 窗口大小 （[25:0]*4K = 窗口的实际大小）
0x2434	0x0	保留

表 22：**BDF 表格编程条目 1**

偏移	编程值	寄存器
0x2440	0x0000	地址转换值低位
0x2444	0x0	地址转换值高位
0x2448	0x0	PASID/保留
0x244C	0x0	[11:0]：功能编号
0x2450	0xC010_0000	[31:30] 读/写访问权限 [29]：R0 访问错误 [28:26] 保护 ID [25:0] 窗口大小 （[25:0]*4K = 窗口的实际大小）
0x2454	0x0	保留

表 23：**BDF 表格编程条目 2**

偏移	编程值	寄存器
0x2460	0x_0000	地址转换值低位
0x2464	0x05	地址转换值高位
0x2468	0x0	PASID/保留
0x246C	0x0	[11:0]：功能编号

表 23：**BDF 表格编程条目 2** (续)

偏移	编程值	寄存器
0x2470	0xC000_00010	[31:30] 读/写访问权限 [29]：R0 访问错误 [28:26] 保护 ID [25:0] 窗口大小 （[25:0]*4K = 窗口的实际大小）
0x2474	0x0	保留

表 24：**BDF 表格编程条目 3**

偏移	编程值	寄存器
0x2480	0x0000_0000	地址转换值低位
0x2484	0x3	地址转换值高位
0x2488	0x0	PASID/保留
0x248C	0x0	[11:0]：功能编号
0x2490	0xC004_0000	[31:30] 读/写访问权限 [29]：R0 访问错误 [28:26] 保护 ID [25:0] 窗口大小 （[25:0]*4K = 窗口的实际大小）
0x2494	0x0	保留

对于此示例

从地址 0x0000_0000_0000_0100 将转换为 0x0000_0AB7_0000_0100。

从地址 0x0000_0001_0000_0100 将转换为 0x0000_0AB0_0000_0100。

从地址 0x0000_0002_0000_0100 将转换为 0x0000_0AB5_0000_0100。

从地址 0x0000_0003_0000_0100 将转换为 0x0000_0AB3_0000_0100。

Slave Bridge 不支持窄突发 AXI 传输事务。为避免窄突发传输，请连接 AXI 智能连接模块，此模块将把窄突发传输转换为完整突发的 AXI 传输。

Master Bridge

Master Bridge 负责处理从 Integrated Block for PCI Express 接收的 PCIe `MemWr` 和 `MemRd` 请求 TLP；并提供一种方法，将在 PCIe 域地址中映射的地址转换为存储器映射 AXI4 地址域。每个 PCIe `MemWr` 请求 TLP 报头都用于为存储器映射 AXI4 总线创建地址和限定符，并将关联的写入数据传递到寻址的存储器映射 AXI4 从接口。Master Bridge 最多可支持 32 个活动 PCIe `MemWr` 请求 TLP。PCIe `MemWr` 请求 TLP 支持如下所示：

- 4 用于 64 位 AXI4 数据宽度
- 8 用于 128 位 AXI4 数据宽度
- 16 用于 256 位 AXI4 数据宽度
- 32 用于 512 位 AXI4 数据宽度

每个 PCIe MemRd 请求 TLP 报头都用于为存储器映射 AXI4 总线创建地址和限定符。从寻址存储器映射 AXI4 Slave Bridge 中收集读取数据，用于生成完成 TLP，然后将这些 TLP 传递到 Integrated Block for PCI Express。处于 AXI Bridge 模式的 Master Bridge 最多可支持 32 个活动 PCIe MemRd 请求 TLP（含暂挂完成），以提高 AXI4 流水打拍的性能。

中断

QDMA Subsystem for PCIe 支持最多总计 2K 个 MSI-X 矢量。单个 MSI-X 矢量可用于支持多个队列。每个功能可支持最多 8 个矢量（8 * 256 个功能= 2K 个矢量）。未启用 SR-IOV 功能时，每个 PF 功能可支持最多 32 个矢量。如果每个功能所需矢量数量超过 8 个，请联系赛灵思技术支持团队。

QDMA 支持中断聚合。每个矢量都有一个关联的中断聚合环。需维护的队列的 QID 和状态都会写入中断聚合环。当主机接收到 PCIe® MSI-X 中断时，软件会读取中断聚合环，以判定需维护的队列。队列到矢量的映射是队列上下文中提供的可编程矢量编号。它支持 SR-IOV 和非 SR-IOV 的 MSI-X 中断模式。

异步中断和基于队列的中断

QDMA 支持异步中断和基于队列的中断。

异步中断用于捕获与任何 DMA 操作都不同步的事件，即错误、状态和调试条件。

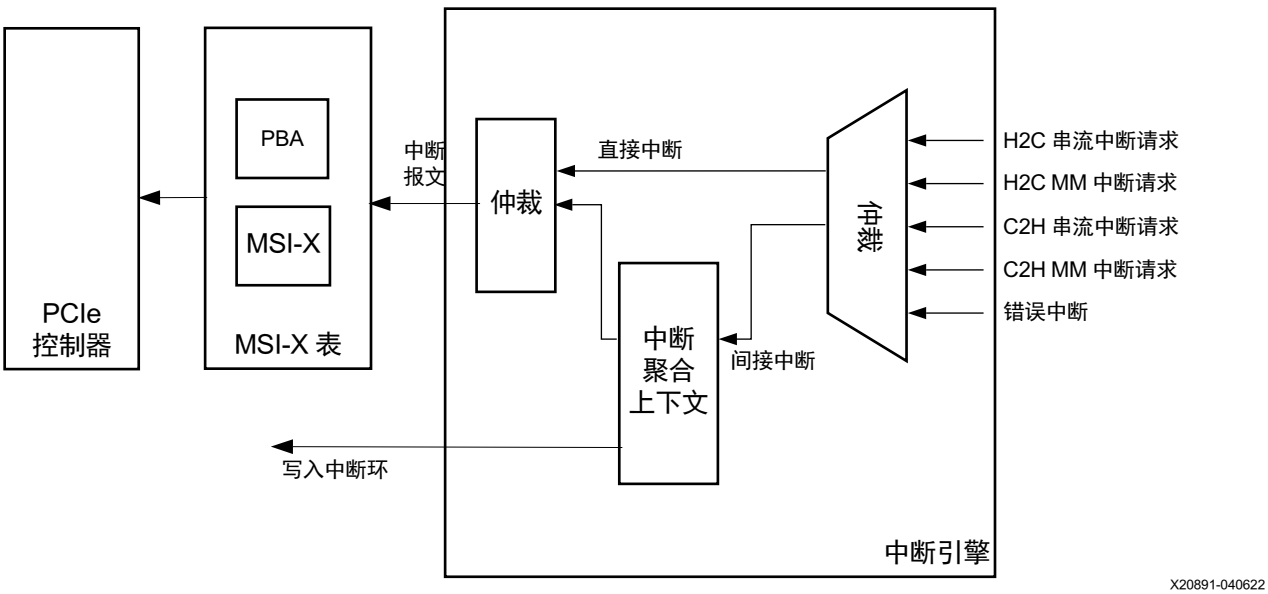
中断会被广播到所有 PF，对于基于队列的方案中的每个 PF，其状态保持不变。基于队列的中断包含来自 H2C MM、H2C 串流、C2H MM 和 C2H 串流的中断。

中断引擎

中断引擎用于处理基于队列的中断和错误中断。

下图显示了中断引擎模块框图。

图 17：**中断引擎模块框图**



中断引擎从 H2C MM、H2C 串流、C2H MM、C2H 串流或错误中断获取中断。

它以两种方式处理中断：直接中断或间接中断。中断来源包含相应信息用于显示它是直接中断还是间接中断。其中也有矢量信息。对于直接中断，矢量即为中断矢量，用于生成 PCIe MSI-X 报文（MSIX 表的中断矢量 `indix`）。对于间接中断，矢量则是中断聚合环的环索引。中断源从描述符软件上下文、完成上下文或错误中断寄存器中获取中断类型和矢量的信息。

直接中断

对于直接中断，中断引擎会从源获取中断矢量，随后直接传出 PCIe MSI-X 报文。

中断聚合环

对于间接中断，它会进行中断聚合。以下是中断聚合的一些限制。

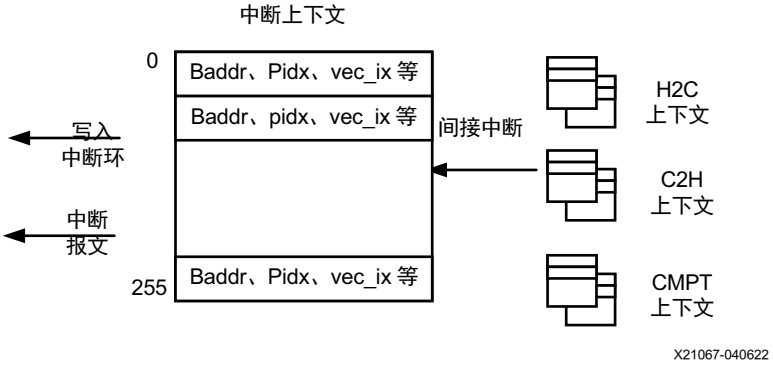
- 每个中断聚合环只能与一个功能关联。但是多个环可与同一功能关联。
- 中断引擎最多支持来自同一源的三个中断，直至软件对中断进行处理。

中断引擎采用以下步骤处理间接中断。

- 中断源会提供它所属的中断环的索引。
- 读取该队列的中断上下文。
- 写入中断聚合环。
- 传出 PCIe MSI-X 报文。

下图显示了间接中断模块框图。

图 18：**间接中断**



中断上下文包含中断聚合环的信息。它有 256 个条目，最多支持 256 个中断聚合环。

由于添加了颜色位，因此软件仅读取它应读取的条目。当软件为中断聚合环分配存储器空间时，`coal_color` 以 `1'b0` 开头。软件需要将中断上下文的颜色位初始化为 `1'b1`。当硬件完成整个环并在下一轮中翻转到第一个条目时，它还会将颜色值翻转为 0 并开始在颜色位空间中写入 0。软件在完成最后一个颜色值为 1 的条目后也同样执行此操作，在第二轮中转到第一个条目，预期的颜色值为 0。如果软件没有找到指示旧条目的颜色值 0，则会等待颜色值为 0 的新条目出现。

软件读取中断聚合环以获取 `Qid` 和 `int_type`（H2C 或 C2H）。通过 `Qid`，软件可以确定队列是串流还是 MM。

中断聚合环中的 `stat_desc` 是来自中断源的状态描述符。如果状态描述符已禁用，软件可从中断聚合环获取状态描述符信息。

这其中可能有两种情况：

- 中断源为 C2H 串流。然后，它是 C2H 完成环的状态描述符。软件可读取 C2H 完成环的 pidx。
- 中断源为其它（H2C 串流、H2C MM、C2H MM）。然后，它是该源的状态描述符。软件可读取 cidx。

最后，中断引擎使用来自中断上下文的中断矢量传出 PCIe MSI-X 报文。当存在来自任何源的中断时，中断引擎会更新 PIDX 并检查该中断上下文的 int_st。如果 int_st 为 0 (WAITING_TRIGGER)，那么中断引擎将发送中断。如果 int_st 为 1 (ISR_RUNNING)，那么中断引擎将不发送中断。如果中断引擎发送中断，它会将 int_sts 更新为 1，一旦软件更新 CIDX，并且 CIDX 与 PIDX 匹配，就会清除 int_sts。此进程说明如下。

当主机接收到 PCIe MSI-X 中断时，软件将读取中断聚合环以确定需要处理哪个队列。软件读取环后，会对软件 CIDX 执行动态指针更新，以指示软件读取的累积指针。软件使用寄存器 QDMA_DMAP_SEL_INT_CIDX[2048] (0x18000) 进行动态指针更新。如果软件 CIDX 等于 PIDX，这将触发写入中断上下文，以针对该队列的中断状态清除 int_st。这是为了向 QDMA 表明，软件已读取中断聚合环中的所有条目。如果软件 CIDX 不等于 PIDX，中断引擎将传出另一条 PCIe MSI-X 报文。因此，软件可再次读取中断聚合环。之后，软件可对中断源环进行指针更新。例如，如果是 C2H 串流中断，那么软件将更新中断源环（即 C2H 完成环）的指针。

以下是针对软件的步骤：

1. 软件收到 PCIe MSI-X 报文后，会读取中断聚合环条目。
2. 软件使用 coal_color 位来识别写入的条目。每个条目都有 Qid 和 Int_type（H2C 或 C2H）。通过 Qid 和 Int_type，软件可以检查它是串流还是 MM。这样会指向对应的源环。例如，如果是 C2H 串流，则源环是 C2H 完成环。然后，软件可读取源环以获取信息，之后对源环进行动态指针更新。
3. 软件读取完中断聚合环中所有写入的条目后，会使用寄存器 QDMA_DMAP_SEL_INT_CIDX[2048] (0x18000) 对软件 cidx 进行动态指针更新。这将传递到软件所使用中断聚合环指针的硬件。

   如果软件 cidx 不等于 pidx，硬件将传出另一条 PCIe MSI-X 报文，以使软件可再次读取中断聚合环。

软件使用寄存器 QDMA_DMAP_SEL_INT_CIDX[2048] (0x18000) 对中断聚合环进行动态指针更新时，会发送中断聚合环的环索引。

下图显示了间接中断流程。中断模块获取中断请求。它首先写入中断聚合环。然后等待写入完成。此后，它会传出 PCIe MSI-X 报文。中断请求可不断传入，中断模块会持续对其进行处理。同时，软件读取中断聚合环，并进行动态指针更新。如果软件 CIDX 不等于 PIDX，它将传出另一条 PCIe MSI-X 报文。

中断上下文结构

中断上下文结构 (0x8) 如下所示。

表 25：**中断上下文结构 (0x8)**

信号	位	所有者	描述
rsvd	[255:126]	驱动	保留。初始化为 0
func	[125:114]	驱动	功能编号
rsvd	[113:83]	驱动	保留。初始化为 0
at	[82]	驱动	1'b0：未转换的地址 1'b1：已转换的地址
pidx	[81:70]	DMA	生产者索引，由 DMA IP 更新。

表 25：**中断上下文结构 (0x8)** (续)

信号	位	所有者	描述
page_size	[69:67]	驱动	中断聚合环大小： 0：4 KB 1：8 KB 2：12 KB 3：16 KB 4：20 KB 5：24 KB 6：28 KB 7：32 KB
baddr_4k	[66:15]	驱动	中断聚合环的基址 – 位 [63:12]
color	[14]	DMA	颜色位
int_st	[13]	DMA	中断状态： 0：WAIT_TRIGGER 1：ISR_RUNNING
Rsvd	[12]	不适用	保留
vec	[11:1]	驱动	msix 表中的中断矢量索引
valid	[0]	驱动	有效

软件需要对中断聚合环大小进行相应调整。每个源均可向环发送最多 2 条报文。因此，环的大小需满足以下公式。

条目数量 ⩾ 队列数的 3 倍

Interrupt Context（中断上下文）由上下文访问进行编程。QDMA_IND_CTXT_CMD.Qid 包含来自中断源的环索引。MDMA_CTXT_CMD_CLR 操作可清除中断上下文中所有位。MDMA_CTXT_CMD_INV 可清除有效位。

- 通过 QDMA_TRQ_SEL_IND 执行上下文访问：
  - QDMA_IND_CTXT_CMD.Qid = 环索引
  - QDMA_IND_CTXT_CMD.Sel = MDMA_CTXT_SEL_INT_COAL (0x8)
  - QDMA_IND_CTXT_CMD.cmd.Op =

    MDMA_CTXT_CMD_WR

    MDMA_CTXT_CMD_RD

    MDMA_CTXT_CMD_CLR

    MDMA_CTXT_CMD_INV

中断引擎查找中断上下文后，中断引擎会写入中断聚合环。中断引擎还会以新的 PIDX、颜色和中断状态来更新中断上下文。

中断聚合条目

这是中断聚合环的条目结构。其中包含 8B 数据。

表 26：**中断聚合环条目结构**

信号	位	所有者	描述
Coal_color	[63]	DMA	中断聚合环的颜色位。每次在中断聚合环上 pidx 发生卷绕时，此位都会反相。
Qid	[62:39]	DMA	此信号来自中断源。队列 ID。
Int_type	[38:38]	DMA	0：H2C 1：C2H
Rsvd	[37:37]	DMA	保留
Stat_desc	[36:0]	DMA	这是中断源的状态描述符。

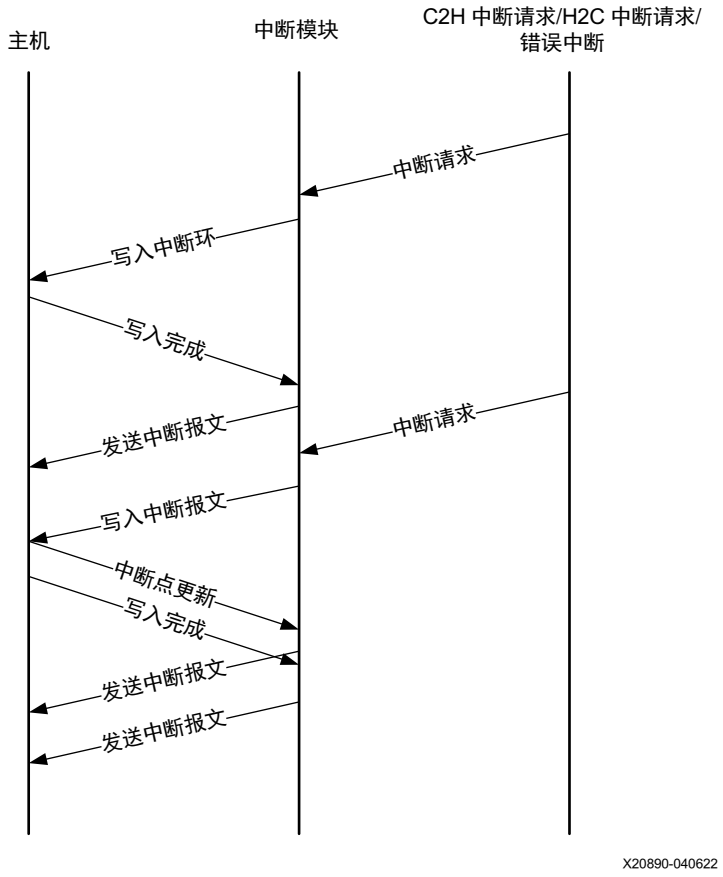
以下是 `stat_desc` 中的信息。

表 27：**stat_desc 信息**

信号	位	所有者	描述
错误	[36:35]	DMA	来自中断源：c2h_err[1:0] 或 h2c_err[1:0]。
Int_st	[34:33]	DMA	此信号来自中断源。中断状态。 0：WRB_INT_ISR 1：WRB_INT_TRIG 2：WRB_INT_ARMED
颜色	[32:32]	DMA	此信号来自中断源。每次 pidx 卷绕时，此位都会反相，并且此字段会被复制到描述符的颜色字段。
Cidx	[31:16]	DMA	此信号来自中断源。累积耗用的指针。
Pidx	[15:0]	DMA	此信号来自中断源。已写入的中断聚合环条目总数的累积指针。

中断流程

图 19：**中断流程**



错误中断

有多个 Leaf Error Aggregator（叶节点错误聚合器）位于不同位置。这些聚合器用于记录错误，并将错误传输到中央错误聚合器。每个叶节点错误聚合器都包含错误状态寄存器和错误掩码寄存器。错误掩码是使能掩码。错误状态寄存器始终都会记录错误，与使能掩码值无关。仅当启用错误掩码时，叶节点错误聚合器才会将错误传输至中央错误聚合器。

中央错误聚合器会将所有错误聚合在一起。发生任何错误时，如果在错误中断寄存器 QDMA_GLBL_ERR_INT (0B04) 中，`err_int_arm` 位已置位，那么中央错误聚合器就会生成 Error Interrupt（错误中断）。当中断引擎提取错误中断时，`err_int_arm` 位由软件置位，并由硬件清零。错误中断适用于所有错误，包括 H2C 错误和 C2H 错误。软件必须对该 `err_int_arm` 位进行置位，才能再次生成中断。

错误中断仅支持直接中断。寄存器 QDMA_GLBL_ERR_INT bit[23] `en_coal` 必须始终编程为 0（直接中断）。

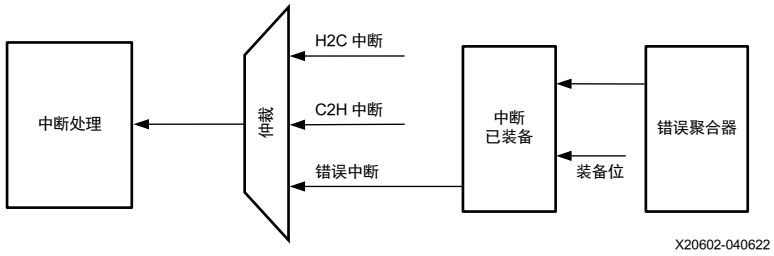
错误中断会从错误中断寄存器 QDMA_GLBL_ERR_INT 获取矢量。对于直接中断，该矢量是 MSI-X 表的中断矢量索引。

错误中断的进程如下。

1. 读取错误中断寄存器 QDMA_C2H_GLBL_INT (0B04) 以获取功能和矢量编号。
2. 传出 PCIe MSI-X 报文。

下图显示了错误中断寄存器模块框图。

图 20：**错误中断处理**



遗留中断

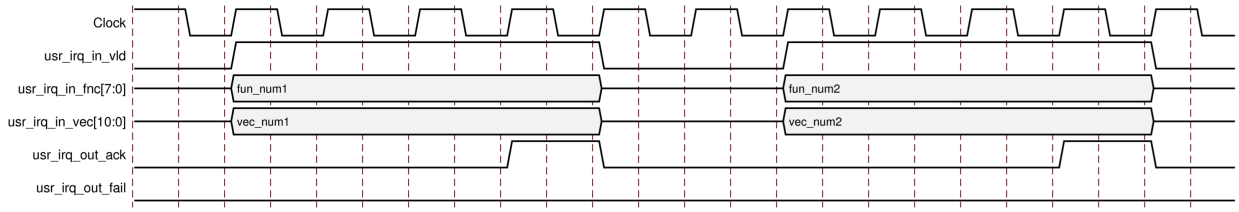
QDMA Subsystem for PCIe 针对物理功能支持遗留中断，并且将有一个队列与中断相关联。

为启用遗留中断，软件需对 QDMA_GLBL_GLBL_INTERRUPT_CFG(0x2C4) 中的 `en_lgcy_intr` 位进行置位。当 `en_lgcy_intr` 置位时，QDMA 将不会发出 MSI-X 中断。

当遗留中断连线 INTA、INTB、INTC 或 INTD 断言有效时，QDMA 硬件会在 QDMA_GLBL_GLBL_INTERRUPT_CFG (0x2C4) 寄存器中对 `lgcy_intr_pending` 位进行置位。当软件接收到遗留中断后，需将 `lgcy_intr_pending` 位清零。硬件将使遗留中断连线保持断言有效，直至 `lgcy_intr_pending` 位被清零为止。

用户中断

图 21：**中断**



队列管理

功能映射表

“Function Map Table”（功能映射表）用于为每项功能分配队列。RAM 中的索引即为功能编号。每个条目都包含物理 QID 的基本编号和分配给功能的队列数。它可提供基于功能的队列访问保护机制，方法是将针对逻辑队列的访问（通过 QDMA_TRQ_SEL_QUEUE_PF 和 QDMA_TRQ_SEL_QUEUE_VF 地址空间）转换为针对其物理队列的访问，并对这些访问执行检查。如果针对队列空间的直接寄存器访问超出表中分配给该功能的队列空间，那么这些访问将被取消，并记录错误。

功能映射可通过间接上下文寄存器空间 QMDA_IND_CTXT_CMD 寄存器来访问，其中 QDMA_IND_CTXT_CMD.sel = 0xC。通过间接上下文寄存器空间访问时，上下文结构由功能映射上下文结构表来定义。

1. 按下表所列方式，对 QDMA_IND_CTXT_DATA (0x804 - 0x820) 寄存器中的功能映射结构进行编程。

2. QDMA_IND_CTXT_CMD 寄存器编程
   a. [19:7]：功能编号
   b. [6:5]：2'h1（写入上下文数据）
   c. [4:1]：4'hC (FMAP)

如需了解有关 QDMA_IND_CTXT_CMD (0x844) 的更多信息，请参阅《Versal ACAP 寄存器参考资料》(AM012)。

由于这些空间仅存在于 PF 地址映射中，因此仅限物理功能才能修改此表。

表 28：**功能映射上下文结构 (0xC)**

位	位宽	字段名称	描述
[255:44]			保留。设为 0。
[43:32]	12	Qid_max	此功能包含的最大队列数。
[31:11]			保留。设为 0。
[10:0]	11	Qid_base	此功能的基本队列 ID。

上下文编程

- 将所有掩码寄存器编程为 1。这些寄存器包括从 QDMA_IND_CTXT_MASK_0 (0x824) 到 QDMA_IND_CTXT_MASK_7 (0x840) 在内的所有寄存器。
- 将上下文值编程到以下寄存器：QDMA_IND_CTXT_DATA_0 (0x804) 到 QDMA_IND_CTXT_DATA_7 (0x810)。
- 需先对主机配置文件表上下文进行编程，然后才能对任何上下文设置 QDMA_CTXT_SELC_HOST_PROFILE 进行编程。选择 QDMA_IND_CTXT_CMD (0x844) 中的 0xA，将所有数据字段写入 0，并对上下文进行编程。所有其它值均为保留值。
- 请参阅“软件描述符上下文结构”、“C2H 预取上下文”和“C2H 预取上下文结构”，了解如何对上下文数据寄存器进行编程。
- 对如下上下文命令寄存器中对应队列的任何上下文进行编程：QDMA_IND_CTXT_CMD (0x844)。
  - 在 `bits [17:7]` 中提供 Qid。
  - 操作码 `bits [6:5]` 用于选择必须完成的操作。
    - 0 = QDMA_CTXT_CLR：所有上下文内容都清零。将在 tm_dsc_sts 上传出 Qinv
    - 1 = QDMA_CTXT_WR：写入上下文
    - 2 = QDMA_CTXT_RD：读取上下文
    - 3 = QDMA_CTXT_INV：Qen 输入设为 0，其它上下文值保持不变。将在 tm_dsc_sts 上传出 Qinv，同时传出未使用的信用值。
  - 在 `bits [4:1]` 中将提供已访问的上下文。
    - 4'h0 = QDMA_CTXT_SELC_DEC_SW_C2H；C2H 描述符软件上下文
    - 4'h1 = QDMA_CTXT_SELC_DEC_SW_H2C：H2C 描述符软件上下文
    - 4'h2 = QDMA_CTXT_SELC_DEC_HW_C2H；C2H 描述符硬件上下文
    - 4'h3 = QDMA_CTXT_SELC_DEC_HW_H2C；H2C 描述符硬件上下文
    - 4'h4 = QDMA_CTXT_SELC_DEC_CR_C2H；C2H 描述符硬件上下文
    - 4'h5 = QDMA_CTXT_SELC_DEC_CR_H2C；H2C 描述符硬件上下文

- 4'h6 = QDMA_CTXT_SELC_WRB；CMPT/使用环上下文
- 4'h7 = QDMA_CTXT_SELC_PFTCH：C2H PFCH 上下文
- 4'h8 = QDMA_CTXT_SELC_INT_COAL：中断聚合上下文
- 4'h9 = 保留
- 4'hA = QDMA_CTXT_SELC_HOST_PROFILE；主机配置文件表（仅支持 QDMA_CTXT_CMD_WR 和 QDMA_CTXT_CMD_RD）
- 4'hB = QDMA_CTXT_SELC_TIMER；定时器上下文（仅支持 QDMA_CTXT_CMD_INV）
- 4'hC = QDMA_CTXT_SELC_FMAP FMAP 表写入（仅支持 QDMA_CTXT_CMD_WR 和 QDMA_CTXT_CMD_RD）
- 4'hD = QDMA_CTXT_SELC_FNC_STS（按功能 BME 启用/禁用）

当位 [0] 置位时，不发生上下文编程写入/读取。如需了解有关寄存器 0x844 的更多信息，请参阅寄存器参考文件中提供的 `qdma_v5_0_pf_registers.csv`。

相关信息

QDMA_CSR (0x0000)

队列设置

- 清除描述符软件上下文。
- 清除描述符硬件上下文。
- 清除描述符信用值上下文。
- 设置描述符软件上下文。
- 清除预取上下文。
- 清除完成上下文。
- 设置完成上下文。
  - 如需中断/状态写入（在完成上下文中启用），则需要通过初始完成 CIDX 更新来将硬件发送到对触发条件敏感的状态中。此初始 CIDX 更新是必需的，因为脱离复位时，硬件会初始化为未装备状态。
- 设置预取上下文。

队列拆解

队列拆解（C2H 串流）：

- 发送标记包以排空流水线。
- 等待标记完成。
- 使预取上下文失效或将其清除。
- 使完成上下文失效或将其清除。
- 使描述符软件上下文失效或将其清除。
- 使定时器上下文失效（不支持 clear cmd）。

队列拆解（H2C 串流和 MM）：

- 使描述符软件上下文失效或将其清除。

虚拟化

QDMA 可实现 SR-IOV 直通虚拟化，其中适配器会公开一个独立的虚拟功能 (VF) 以供虚拟机 (VM) 使用。可以选择为物理功能 (PF) 提供完整访问 QDMA 寄存器和资源的特权，但仅限 VF 才能实现逐队列指针更新寄存器和中断。VF 驱动程序必须通过邮箱与附加到 PF 的驱动程序进行通信，以进行配置、资源分配和异常处理。QDMA 可实现功能级别复位 (FLR)，在 VM 上支持操作系统进行器件复位，且不会对平台其余部分造成干扰。

表 29：**特权级访问**

类型	注释
队列上下文/其它控制寄存器	用于上下文访问的寄存器仅限由 PF（全部 4 个 PF）进行控制。
状态和统计数据寄存器	主要是仅限 PF 的寄存器。VF 需要与 PF 驱动程序协调以进行错误处理。VF 需要通过邮箱与附加到 PF 的驱动程序进行通信。
数据路径寄存器	PF 和 VF 都必须能直接写入数据路径中涉及的寄存器，而无需穿过虚拟机管理器。VF 或 PF 可使用自有 BAR 空间为与功能关联的队列直接完成 H2C/C2H 描述符提取的指针更新。针对不属于功能的队列执行的任何指针更新都将被丢弃，并记录错误。
其它保护建议	开启 IOMMU 以保护来自 VM 的错误存储器访问。
PF 驱动程序和 VF 驱动程序通信	VF 驱动程序需与 PF 驱动程序通信，以请求具有全局影响的操作。此通信通道需要该功能来传递报文和生成中断。此通信通道会为每个 VF 使用一组硬件邮箱。

邮箱

在虚拟环境中，连接到 PF 的驱动程序有足够的特权来编程和访问 QDMA 寄存器。对于所需特权较少的所有功能，部分 PF 和所有 VF 都必须使用邮箱机制来与有特权的驱动程序进行通信。通信 API 必须由驱动程序来定义。QDMA IP 并不会对其进行定义。

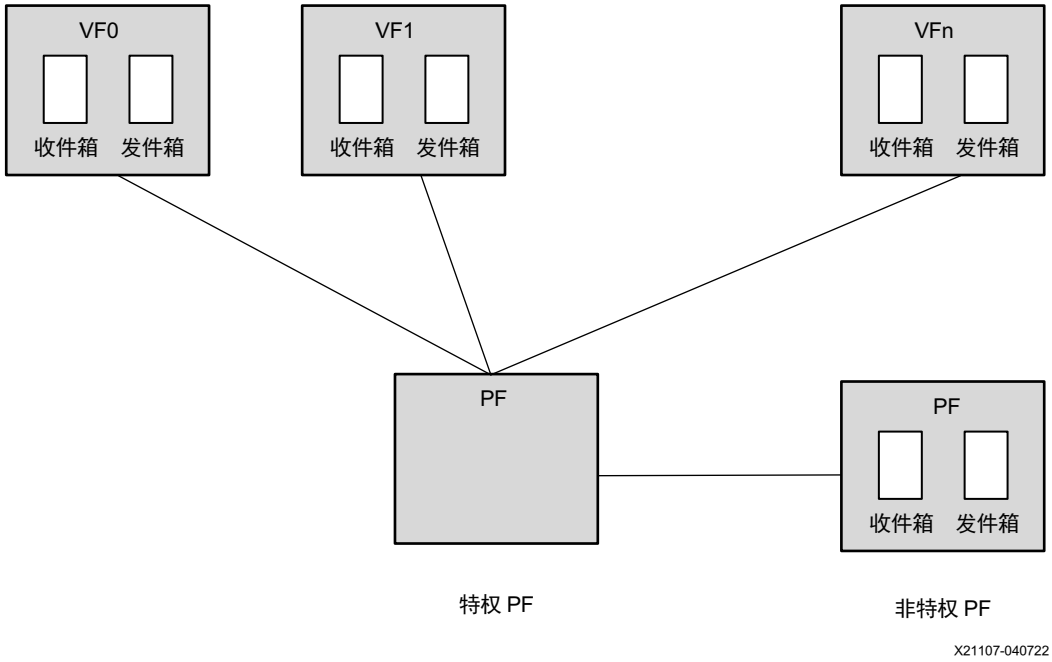
包括 PF 和 VF 在内的每个功能都有一个收件箱和一个发件箱，其中可以容纳大小为 128B 的报文。每个 VF 都各自访问自己的邮箱，每个 PF 不仅可访问其自己的邮箱，也可以访问与该 PF 关联的所有功能（PF 或 VF）。

注释：pcie_qdma_mailbox IP 支持最高 4PF 和 240VF。您可在 PL 中构建邮箱系统以支持更多 PF 和 VF（CPM5 限制为 16PF 和 2KVF）。添加 pcie_qdma_mailbox IP 可以提升 PL 利用率。

QDMA 邮箱允许以下访问：

- 从 VF 访问关联的 PF。
- 从 PF 访问属于其自有虚拟功能组 (VFG) 的任意 VF。
- 从某个 PF（通常是无权访问 QDMA 寄存器的驱动程序）访问另一个 PF。

图 22：**邮箱**

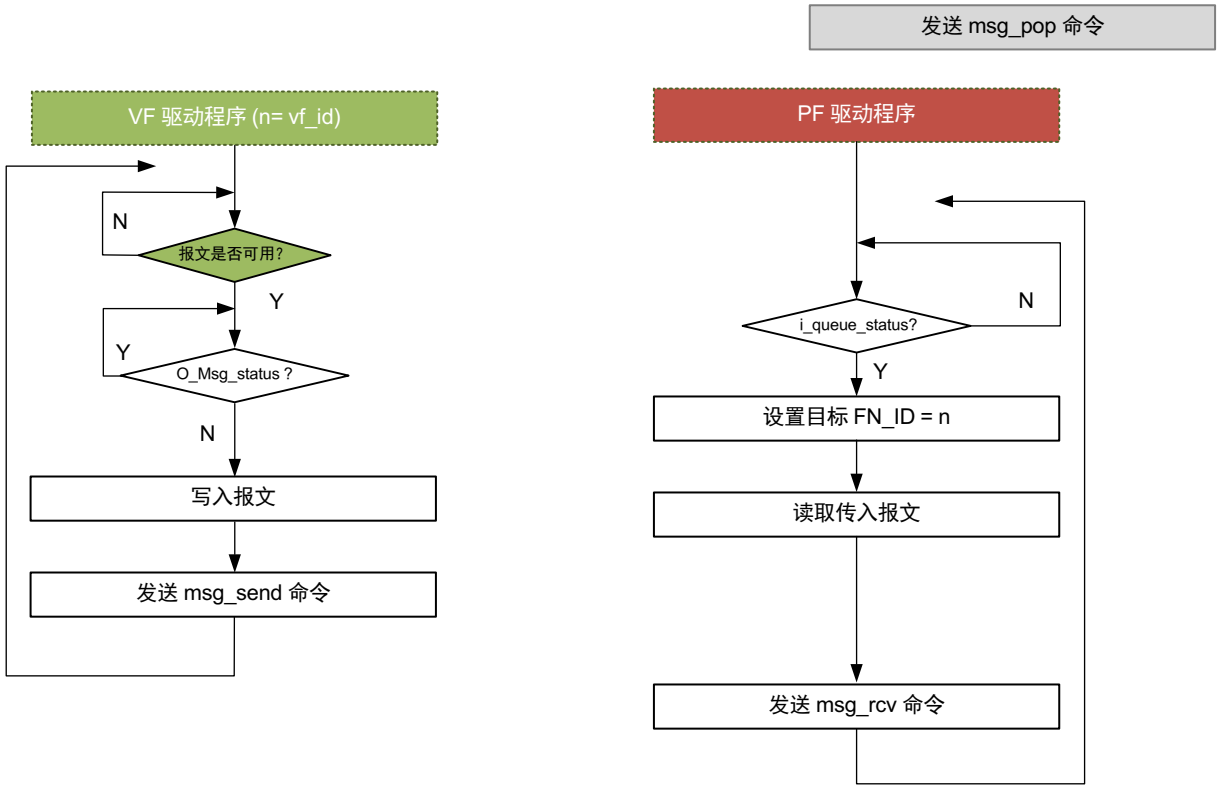


从 VF 到 PF 的报文传递

允许每个 VF 向目标 PF 邮箱发布一条报文，直至目标功能 (PF) 接受此报文为止。在发布报文之前，源功能应确保其 `o_msg_status` 处于已清零状态，然后 VF 才能将报文写入其传出报文寄存器。报文写入完成后，VF 驱动程序通过在控制/状态寄存器 (CSR) 地址 0x5004 写入 0x1 来发送 `msg_send` 命令。然后，邮箱硬件通过将 `i_msg_status` 字段断言有效来通知 PF 驱动程序。

功能驱动程序应启用 `i_msg_status` 的定期轮询，以检查传入报文的可用性。在 PF 侧，`i_msg_status = 0x1` 表示有一条或多条报文正在等待 PF 驱动程序提取。“Mailbox Status Register”（邮箱状态寄存器）中的 `cur_src_fn` 会提供第一条待处理报文的功能 ID。然后，PF 驱动程序应将“Mailbox Target Function Register”（邮箱目标功能寄存器）设置为第一条待处理报文的源功能 ID。随后即可对 PF 的“Incoming Message Registers”（传入报文寄存器）进行间接访问，这意味着邮箱硬件返回的报文字节数将始终与目标功能所发送的字节数相对应。完成报文读取后，PF 驱动程序还应通过在 CSR 地址处写入 0x2 来发送 `msg_rcv` 命令。硬件将在源功能侧断言 `o_msg_status` 无效。下图演示了从 VF 到源侧和目标侧的 PF 的报文传递流程。

图 23：**从 VF 到 PF 的报文传递流程**



VF (#n) 到 PF 报文流程
状态轮询可更改为由中断驱动

X21105-040722

从 PF 到 VF 的报文传递

从 PF 到属于其 VFG 的 VF 的报文传递流程与从 VF 到 PF 的流程略有不同，因为：

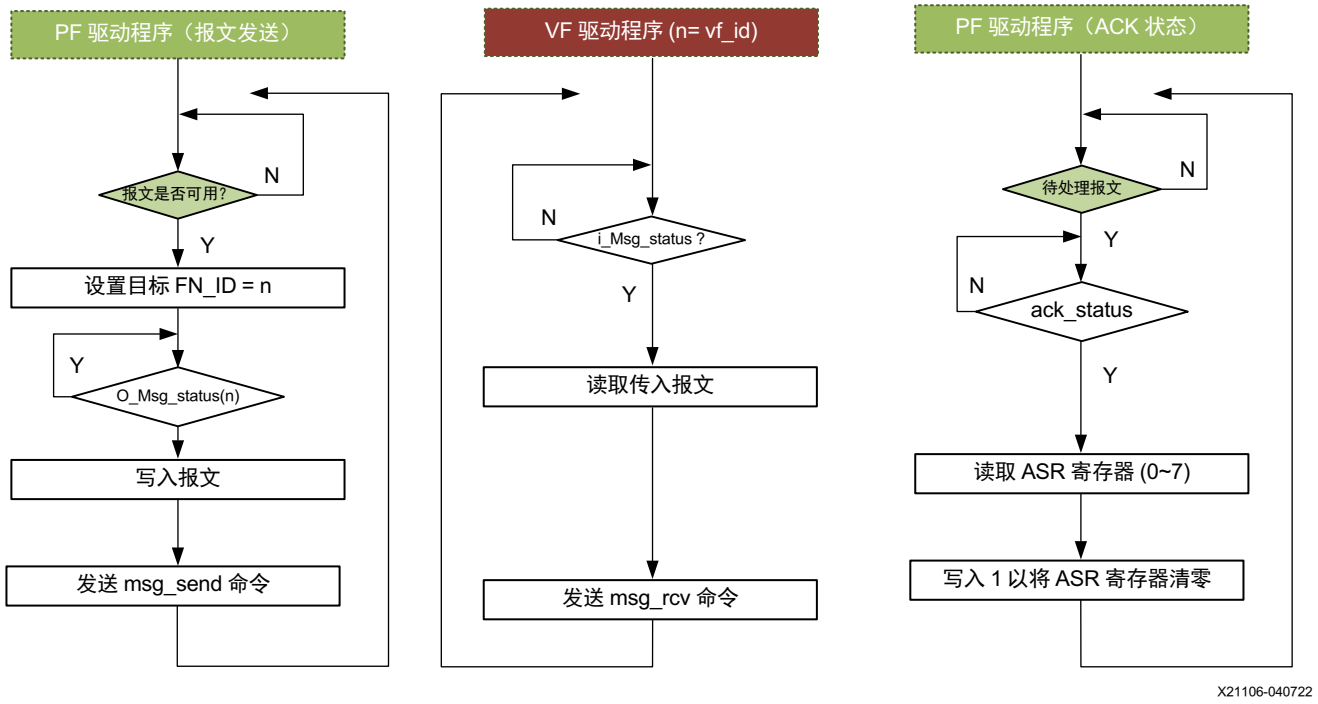
PF 可向多个目标功能发送报文，所以它在检查状态时可能会收到多个确认。如下图所示，PF 驱动程序必须先将“Mailbox Target Function Register”（邮箱目标功能寄存器）设置为目标功能 ID，然后才能执行任何报文传递操作；例如，检查传入报文状态、写入报文或发送命令。在 VF 侧（接收侧），每当 VF 驱动程序收到 `i_msg_status = 0x1` 时，VF 驱动程序都应读取其“Incoming Message Registers”（传入报文寄存器）以提取该报文。根据不同应用，VF 驱动程序可在读取报文后立即发送 `msg_rcv`，或者在处理完对应报文后再发送。

为了避免逐一轮询传出报文的状态，邮箱硬件为每个 PF 提供了一组“Acknowledge Status Registers (ASR)”（确认状态寄存器）。当邮箱收到来自 VF 的 `msg_rcv` 命令时，它就会将源 PF 的 `o_msg_status` 字段断言无效，同时也会在“Acknowledge Status Registers”中设置对应的位。对于功能 ID 为 <N> 的给定 VF，确认状态位于：

- Acknowledge Status Register 地址：<N> / 32 + <0x22420 寄存器地址>
- 确认状态位的位置为：<N> / 32

只要“Acknowledge Status Register (ASR)”中有任何位已断言有效，邮箱硬件就会将“Status Register”（状态寄存器）(0x22400) 中的 `ack_status` 断言有效。PF 驱动程序可先轮询 `ack_status`，而后再实际读出确认状态寄存器。PF 驱动程序可通过一次寄存器访问来检测多个完成情况。PF 驱动程序完成处理后，还应将该值写回同一寄存器地址以清除状态。

图 24：**从 PF 到 VF 的报文传递流程**



邮箱中断

邮箱模块支持中断作为备用事件通知机制。每个邮箱都有一个“Interrupt Control Register”（中断控制寄存器），对于 PF，该寄存器位于偏移 0x22410 处，对于 VF，则位于偏移 0x5010 处）。将此寄存器设为 1 即可启用中断。启用中断后，只要有任何待处理的事件等待邮箱处理（如任何待处理的传入报文或对应传出报文的任何确认），邮箱都将向 QDMA 发送中断。根据驱动程序配置，通过“Function Interrupt Vector Register”（功能中断矢量寄存器）（对应 PF 为 0x22408，对应 VF 则为 0x5008）来配置中断矢量。

启用中断并不会更改事件记录机制，这意味着用户必须通过读取“Function Interrupt Vector Register”（功能状态寄存器）来检查待处理事件。响应中断请求的第一步是禁用中断。可能待处理事件的实际数量比向邮箱发送中断时的事件数量更多。

建议：赛灵思建议用户应用中断处理程序对状态寄存器中存在的所有待处理事件进行处理。完成中断响应后，用户应用即可重新启用中断。

在中断控制从禁用变为启用时，邮箱会检查其事件状态。如果在读取中断状态和重新启用中断之间有任何新事件到达邮箱，那么邮箱将立即生成一个新的中断请求。

相关信息

QDMA_PF_MAILBOX (0x22400)
QDMA_VF_MAILBOX (0x5000)

功能等级复位

功能级别复位 (FLR) 机制支持软件按功能级别的粒度将端点硬件置于静止和复位状态。当 VF 复位时，仅复位与该 VF 关联的资源。当 PF 复位时，则复位 PF 的所有资源，包括其关联的 VF 的资源。由于 FLR 属特权操作，它必须由管理系统中运行的 PF 驱动程序来执行。

使用模式

- 当功能发生连接和断开连接（即上电和下电）时，虚拟机管理器会请求 FLR。
- 您可按如下方式请求 FLR：

```
echo 1 > /sys/bus/pci/devices/$BDF/reset
```

其中，`$BDF` 是目标功能的总线器件功能编号。

FLR 进程

完整的 FLR 进程涉及三个主要步骤。

1. “Pre-FLR”（FLR 前）：此步骤会复位目标功能的所有 QDMA 上下文结构、邮箱和用户逻辑。
   - 每个功能都有一个称为 MDMA_PRE_FLR_STATUS 的寄存器，用于保留功能的 FLR 前状态的记录。偏移的计算方式为 MDMA_PRE_FLR_STATUS_OFFSET = MB_base + 0x100，它位于功能的邮箱存储器空间的偏移 0x100 处。请注意，PF 和 VF 具有不同的 MB_base。MDMA_PRE_FLR_STATUS 的定义如下图表所示。
   - 软件将 1 写入目标功能的 MDMA_PRE_FLR_STATUS[0]（位 0）以启动 FLR 前步骤。当 FLR 前步骤完成时，硬件会将 MDMA_PRE_FLR_STATUS[0] 清零。软件不断轮询 MDMA_PRE_FLR_STATUS[0]，仅当它返回 0 时，才会继续下一个步骤。

   表 30：**MDMA_PRE_FLR_STATUS 寄存器**

偏移	字段	读/写类型	宽度	默认	描述
0x100	pre_flr_st	RW	32	0	[31:1] 保留。 [0]: 1 表示启动 FLR 前步骤。 [0]: 0 表示 FLR 前步骤完成。 bit[0] 由驱动程序置位，由硬件清零。

2. “Quiesce”（静止）：软件必须确保所有待处理的传输事务均已完成。方法是轮询器件状态寄存器中的传输事务待处理位（位于 PCIe 配置空间内），直到该位清零或者在一段时间后超时为止。
3. PCIe-FLR：PCIe-FLR 步骤会复位 PCIe 控制器中目标功能的所有资源。

   注释：目标功能的启动功能级别复位位（PCIe 器件控制寄存器的位 15）应设为 1，以触发 PCIe 中的 FLR 进程。

OS 支持

如果 PF 驱动程序已加载并处于活动状态（即，使用模式为 1），则上述所有三个步骤均由驱动程序来执行。但对于 UltraScale+，如果用户要在加载 PF 驱动程序之前执行 FLR（按上述“使用模式”中定义的方式），我们提供了操作系统内核补丁，允许操作系统通过 `//.../source/drivers/pci/quick.c` 中定义的功能按正确顺序来执行 FLR 步骤。

端口 ID

端口 ID 用于对 FPGA 侧的某些队列进行分类。如有多个用户应用共享 DMA，则端口 ID 可提供对 QID 的间接寻址，以便以更低成本对所有接口进行进一步解复用。但是，只有单个应用使用 DMA 时，则可忽略端口 ID 并将端口 id 输入驱动至 0。

主机配置文件

主机配置文件必须编程为表示 Root Port（根端口）主机。主机配置文件可通过上下文编程来进行编程。在 QDMA_IND_CTXT_CMD 中选择 QDMA_CTXT_SELC_HOST_PROFILE (4'hA)。下表中提供了主机配置文件上下文结构。

表 31：**主机配置文件上下文结构**

位	位宽	字段名称	描述
[255:188]	68	Reserved	保留
[187:186]	2		H2C AXI4-MM 写入 awprot
[185:182]	4		H2C AXI4-MM 写入 awcache
[181:178]	4		H2C AXIMM 导向
[177:104]	74	Reserved	保留
[103:102]	2		C2H AXI4-MM 读取 arprot
[101:98]	4		C2H AXI4-MM 读取 awcache
[97:94]	4		C2H AXIMM 导向
[0:93]	94	Reserved	保留

H2C AXI4-MM 导向位和 C2H AXI4-MM 导向位应设置为 0。否则，无法进行 DMA AXI4-MM 传输。在大多数情况下，主机配置文件上下文结构将全部为 0，主机配置文件仍必须通过编程来表示主机。

系统管理

复位

QDMA Subsystem for PCIe 支持所有 PCIe 定义的复位，例如，链路中断、复位、热复位和功能级别复位 (FLR)（仅限“静止”模式下才支持）。

软复位

通过 `soft_reset_n` 端口复位 QDMA 逻辑。此端口需保持复位状态至少 100 个时钟周期（`axi_aclk` 个周期）。

这不会将 PCIe 硬核块复位。它仅复位逻辑的 DMA 部分。如果存在 DMA 挂起或某些错误条件，则此复位可断言有效。

软复位用例

提示使用 `soft_reset` 的用例包括：

- DMA 挂起并且用户没有得到正确的值。
- DMA 传输有错误，但 PCIe 链路正常。
- DMA 记录某些异步错误

`soft_reset` 后，必须重新初始化队列并对所有队列上下文进行编程。

VDM

供应商定义报文 (VDM) 是对 PCI Express 现有报文传递功能的扩展。PCI Express 规范定义了针对供应商定义报文、报头格式和路由信息的附加要求。欲知详情，请参阅《PCI-SIG 规范》(https://www.pcisig.com/specifications)。

QDMA 允许传输和接收 VDM。要启用该功能，请选中 Vivado 的“Customize IP”（自定义 IP）对话框中的“Enable Bridge Slave Mode”（启用 Bridge 从模式）。这将启用 `st_rx_msg` 接口。

RX 供应商定义报文先存储在浅 FIFO 中，然后再发射到输出端口。如有大量连续 VDM 报文，FIFO 将发生上溢，这些报文将被丢弃。因此，最好定期重复 VDM 报文。

VDM 的吞吐量取决于几个因素：PCIe 速度、数据宽度、报文长度和内部 VDM 流水线。

内部 VDM 流水线必须替换为内部 RX VDM FIFO 接口，以便进行片上网络 (NoC) 访问，NoC 有 64B 的浅缓冲器。

注释：如果在通过 NoC 维护 FIFO 之前收到的 VDM 超过 64B，那么新 VDM 报文将被丢弃。

内部 RX VDM FIFO 接口不能处理连续报文。流水线吞吐量只能处理每四次访问中的一次访问，即约为主机访问效率的 25%。

重要提示！请勿使用连续 VDM 访问。

RX 供应商定义报文：

1. 当 QDMA 收到 VDM 时，将在 `st_rx_msg` 端口接收传入的报文。
2. 将在 `st_rx_msg_data` 端口捕获（按 DW）传入的数据串流。
3. 用户应用需驱动 `st_rx_msg_rdy` 以发出信号，指明它是否能接受传入 VDM。
4. 当 `st_rx_msg_rdy` 为高位时，传入 VDM 将被转发至用户应用。
5. 用户应用需存储此传入 VDM，并跟踪接收到的包数量。

TX 供应商定义报文：

1. 要启用从 QDMA 发射到 VDM 的操作，请在 Bridge 中通过从接口对 TX Message 寄存器进行编程。
2. Bridge 具有 TX Message Control、Header L（字节 8-11）、Header H（字节 12-15）和 TX Message Data 寄存器，如 PCIe TX Message Data FIFO Register (TX_MSG_DFIFO) 中所示。
3. 通过从接口对偏移 0xE64 发出 Write（写入），供 TX Message Header L 寄存器使用。
4. 对偏移 0xE68 进行编程，以供所需 VDM TX Header H 寄存器使用。
5. 向偏移 0xE6C 逐一发送 Write，对最多 16 个 DW 有效载荷进行编程，以供从 DW0 开始到 DW15 为止的 VDM 报文使用。
6. 对 TX_MSG_CTRL 寄存器中的偏移 0xE60 的 `msg_routing`、`msg_code`、数据长度、请求器功能字段和 `msg_execute` 字段进行编程以发送 VDM TX 包。
7. TX Message Control 寄存器还会在位 23 处指明报文的完成状态。用户需读取该位，以确认是否成功发射 VDM 包。
8. 这些寄存器中的所有字段均为 RW，TX Control 中的位 23 (`msg_fail`) 除外，写入 1 即可将该位清零。
9. VDM TX 包将在 AXI-ST RQ 发射接口上发送。

相关信息

VDM 端口
Bridge 寄存器空间

配置扩展

通过选择 PCIe 扩展接口即可增加配置空间。选中“Configuration Extend Interface”（配置扩展接口）时，由您负责添加接口扩展逻辑以使其正常工作。

Expansion ROM

如果选中“Expansion ROM”（扩展 ROM），就会将其激活，可将其设为 2 KB 到 4 GB 之间的值。根据《PCI 局部总线规范》（《PCI-SIG 规范》(https://www.pcisig.com/specifications)），扩展 ROM BAR 的最大大小应不超过 16 MB。选择大于 16 MB 的地址空间可能会导致核不符合标准。

错误

Bridge 错误

Slave Bridge 异常情况

Slave Bridge 异常情况分类为：违规突发类型和完成 TLP 错误。以下部分描述了 Bridge 处理这些错误的方式。

违规突发类型

Slave Bridge 监控 AXI 读取和写入突发类型输入，以确保仅请求 INCR（增量突发）类型。这些输入上的任何其它值都被视为错误情况，并且从接口违规突发 (SIB) 中断将断言有效。在读取请求情况下，Bridge 对所有数据节拍断言 SLVERR 有效，并且将任意数据布局在 `s_axi_rdata` 总线上。在写入请求情况下，Bridge 对写入响应断言 SLVERR 有效，并且会丢弃所有写入数据。

完成 TLP 错误

对 PCIe 的总线的任何请求（转发存储器写入除外）都需要完成 TLP，才能完成关联的 AXI 请求。Bridge 的 Slave 端会检查接收到的完成 TLP 是否有错误，并检查是否有任何完成 TLP 一直未返回（完成超时）。我们将在后续部分中讨论每个完成 TLP 错误类型。

意外完成

当 Slave Bridge 接收到完成 TLP 时，它会将报头 RequesterID 和 Tag 与未完成的 RequesterID 和 Tag 匹配。匹配失败表明 TLP 是意外完成，这会导致丢弃完成 TLP，并且从接口意外完成 (SUC) 中断选通将断言有效。然后继续正常操作。

请求不受支持

PCIe 的器件可能无法满足特定的读取请求。例如，如果读取请求以 PCIe 不支持的地址为目标，完成器将返回完成状态为 `0b001 - Unsupported Request` 的完成 TLP。根据 PCI Express 基本规范 v3.0，如果完成器返回的完成 TLP 的完成状态为 `Reserved`，则必须将其作为不受支持的请求状态来进行处理。当 Slave Bridge 接收到不受支持的请求响应时，从接口不受支持的请求 (SUR) 中断将断言有效，并且 DECERR 响应将断言有效，同时在 AXI4 存储器映射总线上布局任意数据。

完成超时

当处理 AXI 到 PCIe 存储器读取请求或 PCIe 配置读取/写入请求后未返回完成 (Cpl) 或具有数据的完成 (CplD) TLP 时，将发生完成超时。对于 PCIe 配置读取/写入请求，这些完成必须在 C_COMP_TIMEOUT 参数选定值范围内完成（从发出请求的时间算起）。对于 PCIe 存储器读取请求，这些完成必须在 PCIe 配置空间寄存器的器件控制 2 寄存器中设置的值范围内完成。如果发生完成超时，则 OKAY 响应将断言有效，并且在存储器映射 AXI4 总线上全部布局 1 数据。

完成包上接收到毒化位

当 TLP EP 位完成置位时如果发生错误毒化，则表明有效载荷中存在毒化数据。当 Slave Bridge 检测到毒化包时，从接口错误毒化 (SEP) 中断将断言有效，SLVERR 响应将断言有效，同时在存储器映射 AXI4 总线上布局任意数据。

完成器异常中止

当完成 TLP 的完成状态为 0b100（完成器异常中止）时，会发生完成器异常中止。这表明完成器遇到了无法完成传输事务的状态。当 Slave Bridge 接收到完成器异常中止响应时，从接口完成器异常中止 (SCA) 中断将断言有效，SLVERR 响应将断言有效，同时在存储器映射 AXI4 总线上布局任意数据。

表 32：**Slave Bridge 对异常情况的响应**

传输类型	异常情况	Bridge 响应
读取	违规突发类型	SIB 中断断言有效。 提供具有任意读取数据的 SLVERR 响应。
写入	违规突发类型	SIB 中断断言有效。 丢弃写入数据。 提供 SLVERR 响应。
读取	意外完成	SUC 中断断言有效。 丢弃完成。
读取	返回“不受支持的请求”状态	SUR 中断断言有效。 提供具有任意读取数据的 DECERR 响应。
读取	完成超时	SCT 中断断言有效。 提供具有任意读取数据的 SLVERR 响应。
读取	完成中的毒化位	丢弃完成数据。 SEP 中断断言有效。 提供具有任意读取数据的 SLVERR 响应。
读取	返回“完成器异常中止 (CA)”状态	SCA 中断断言有效。 提供具有任意读取数据的 SLVERR 响应。

Master Bridge 异常情况

以下部分描述了 Master Bridge 处理异常情况的方式。

AXI DECERR 响应

当 Master Bridge 接收到来自 AXI 总线的 DECERR 响应时，将丢弃该请求，并且主接口 DECERR (MDE) 中断将断言有效。如果请求为非转发请求，则在 PCIe 的总线上返回完成状态为“不受支持的请求 (UR)”的完成包。

AXI SLVERR 响应

当 Master Bridge 接收到来自寻址的 AXI 从接口的 SLVERR 响应时，将丢弃该请求，并且主接口 SLVERR (MSE) 中断将断言有效。如果请求为非转发请求，则在 PCIe 的总线上返回完成状态为“完成器异常中止 (CA)”的完成包。

PCIe 的最大有效载荷大小、最大读取请求大小

当 Master Bridge 接收到来自寻址的 AXI 从接口的 SLVERR 响应时，将丢弃该请求，并且主接口 SLVERR (MSE) 中断将断言有效。如果请求为非转发请求，则在 PCIe 的总线上返回完成状态为“完成器异常中止 (CA)”的完成包。

完成包

如果 MAX_READ_REQUEST_SIZE 大于 MAX_PAYLOAD_SIZE，则 PCIe 的读取请求可以请求的数据比 Master Bridge 可插入到单个完成包中的数据更多。发生这种情况时，会生成多个完成包，最多达到 MAX_PAYLOAD_SIZE 个，含观察到的读取完成边界 (RCB)。

毒化位

如果在传输事务层包 (TLP) 报头中毒化位已置位，则报头后接的有效载荷已损坏。如果 Master Bridge 接收到的存储器请求 TLP 中毒化位已置位，那么它会丢弃该 TLP，并且主接口错误毒化 (MEP) 中断选通将断言有效。

零长度请求

当 Master Bridge 接收到 Length = 0x1、FirstBE = 0x00 且 LastBE = 0x00 的读取请求时，它通过发送 Status = Successful Completion 的完成进行响应。

当 Master Bridge 接收到 Length = 0x1、FirstBE = 0x00 且 LastBE = 0x00 的写入请求时，不会有任何反应。

表 33：**Master Bridge 对异常情况的响应**

传输类型	异常情况	Bridge 响应
读取	DECERR 响应	MDE 中断选通断言有效。 返回具有“不受支持的请求”状态的完成。
写入	DECERR 响应	MDE 中断选通断言有效。
读取	SLVERR 响应	MSE 中断选通断言有效。 返回具有“完成器异常中止”状态的完成。
写入	SLVERR 响应	MSE 中断选通断言有效。
写入	请求中毒化位已置位	MEP 中断选通断言有效。 丢弃数据。
读取	DECERR 响应	MDE 中断选通断言有效。 返回具有“不受支持的请求”状态的完成。
写入	DECERR 响应	MDE 中断选通断言有效。

链路中断错误

如果在 DMA 操作期间 PCIe 链路中断，传输事务可能会丢失，DMA 可能无法完成。在此情况下，AXI4 接口将继续工作。C2H Bridge AXI4 MM 接口上未完成的读取请求会接收正确的完成或者含从接口错误响应的完成。DMA 将在状态寄存器中记录链路中断错误。驱动程序将负责设置超时，并处理链路中断情况的恢复。

数据路径错误

在主数据路径上支持数据保护。在 C2H 串流和 H2C 串流上可能发生 CRC 错误。在存储器映射接口、Bridge 主接口和 Bridge 从接口上都可能发生奇偶校验错误。在 C2H 串流接口、存储器映射接口和 Bridge 从接口上，则可能发生写入有效载荷错误。Bridge 从接口的写入有效载荷和读取完成上的双位元错误会导致奇偶校验错误。对 PCIe 发出的请求上的奇偶校验错误会被核丢弃，并且 PCIe 会记录致命错误。奇偶校验错误不可恢复，并可能导致意外行为。发生奇偶校验错误期间及之后的任何 DMA 都应被视为无效。如果出现奇偶校验错误，并且传输挂起或停止，那么 DMA 将记录该错误。您必须对奇偶校验问题进行调查并修复。问题得到解决后，请清除该队列，并重新打开队列以启动新的传输。

DMA 错误

所有 DMA 错误都记录在各自的错误状态寄存器中。每个块都有错误状态和错误掩码寄存器，因此错误可以传递到更高级别，并最终传递到 QDMA_GLBL_ERR_STAT 寄存器。

根据寄存器设置，错误可能是致命错误。如果存在致命错误，DMA 将停止传输，并发送中断（如已启用）。在调试和分析后，必须使队列无效并重新启动，以启动 DMA 传输。

错误聚合器

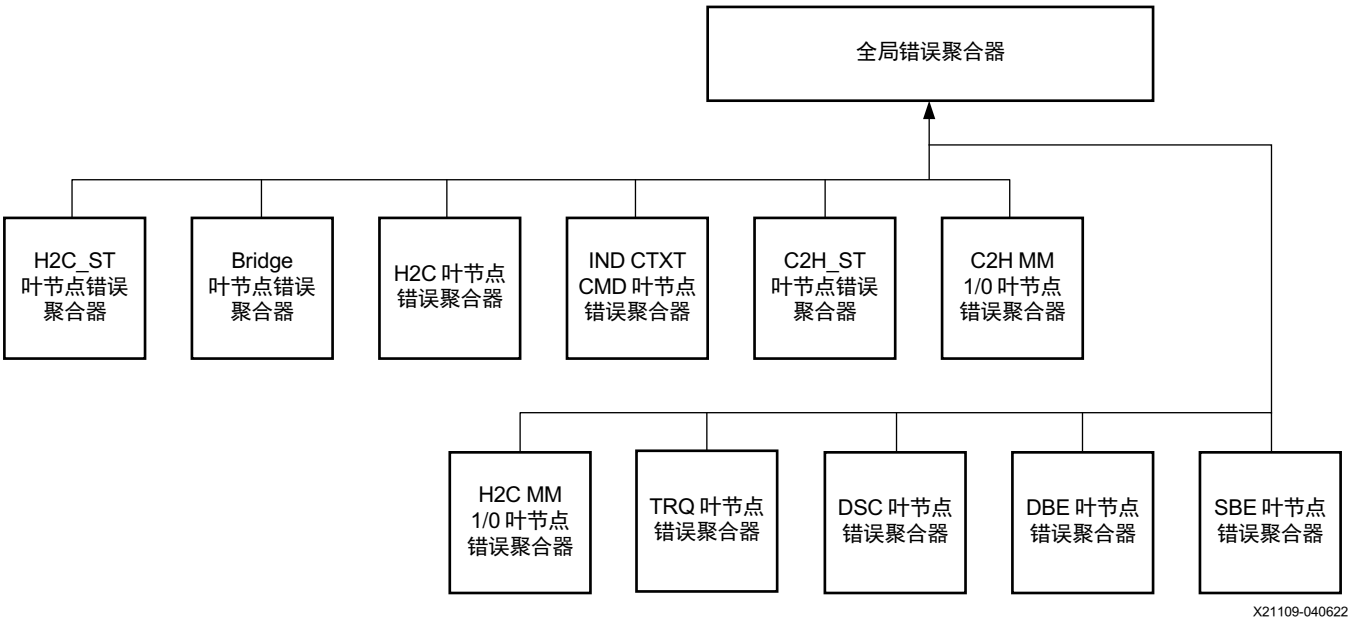
有多个 Leaf Error Aggregator（叶节点错误聚合器）位于不同位置。它们用于记录各种错误，并将其传输到集中位置。“Central Error Aggregator”（中央错误聚合器）用于聚合来自所有叶节点错误聚合器的错误。

QDMA_GLBL_ERR_STAT 寄存器是中央错误聚合器的错误状态寄存器。位字段用于指示叶节点错误聚合器的位置。随后，通过查找各叶节点错误聚合器的错误状态寄存器来找到精确错误。

寄存器 QDMA_GLBL_ERR_MASK 是中央错误聚合器的错误掩码寄存器。其中具有对应错误的掩码位。当掩码位设为 `1'b1` 时，它将启用对应错误，将其传输至下一级以生成中断。在中断部分会描述生成中断的错误的详细信息。错误中断由寄存器 QDMA_GLBL_ERR_INT (0xB04) 控制。

每个叶节点错误聚合器都包含错误状态寄存器和错误掩码寄存器。错误状态寄存器用于记录错误。发生错误时，硬件会对该位进行置位，软件可按需写入 `1'b1` 以将该位清零。错误掩码寄存器具有对应错误的掩码位。掩码位设为 `1'b1` 时，它将启用将对应错误传输至中央错误聚合器的操作。错误掩码寄存器不影响将错误记录到错误状态寄存器的操作。

图 25：**错误聚合器**



叶节点错误聚合器的错误状态寄存器和错误掩码寄存器如下。

C2H 串流错误

QDMA_C2H_ERR_STAT (0xAF0)：这是 C2H 串流错误的错误状态寄存器。

QDMA_C2H_ERR_MASK (0xAF4)：这是错误掩码寄存器。软件可对该位进行置位，以启用将对应 C2H 串流错误传输到中央错误聚合器的操作。

QDMA_C2H_FIRST_ERR_QID (0xB30)：这是第一个 C2H 串流错误的 QID。

C2H MM 错误

QDMA_C2H MM 状态 (0x1040)

C2H MM 错误代码使能掩码 (0x1054)

C2H MM 错误代码 (0x1058)

C2H MM 错误信息 (0x105C)

QDMA H2C0 MM 错误

H2C0 MM 状态 (0x1240)

H2C MM 错误代码使能掩码 (0x1254)

H2C MM 错误代码 (0x1258)

H2C MM 错误信息 (0x125C)

TRQ 错误

QDMA_GLBL_TRQ_ERR_STS (0x264)：这是 Trq 错误的错误状态寄存器。

QDMA_GLBL_TRQ_ERR_MSK (0x268)：这是错误掩码寄存器。

QDMA_GLBL_TRQ_ERR_LOG_A (0x26C)：这是错误记录寄存器。它显示错误发生时的地址的选择、功能和访问。

描述符错误

QDMA_GLBL_DSC_ERR_STS (0x254)

QDMA_GLBL_DSC_ERR_MSK (0x258)：这是错误记录寄存器。它具有 QID、DMA 方向，以及错误的使用者索引。

QDMA_GLBL_DSC_ERR_LOG0 (0x25C)

QDMA_GLBL_TRQ_ERR_STS (0x264)：这是 TRQ 错误的错误状态寄存器。

RAM 双位错误

QDMA_RAM_DBE_STS_A (0xFC)

QDMA_RAM_DBE_MSK_A (0xF8)

RAM 单位错误

QDMA_RAM_SBE_STS_A (0xF4)

QDMA_RAM_SBE_MSK_A (0xF0)

相关信息

寄存器空间

C2H 串流致命错误处理

- QDMA_C2H_FATAL_ERR_STAT (0xAF8)：C2H 串流致命错误的错误状态寄存器。
- QDMA_C2H_FATAL_ERR_MASK (0xAFC)：错误掩码寄存器。软件可对该位进行置位以启用将对应 C2H 致命错误发送到 C2H 致命错误处理逻辑的功能。
- QDMA_C2H_FATAL_ERR_ENABLE (0xB00)：此寄存器可启用 2 个 C2H 串流致命错误处理进程：
  - bit[0]：将 `enable_wrq_dis` 位 [0] 设为 1，禁用来自 C2H DMA 写入引擎的写入请求，以停止数据传输。
  - bit[1]：将 `enable_wpl_par_inv` 位 [1] 设为 1，使数据传输上的写入有效载荷奇偶校验反相。

相关信息

QDMA_CSR (0x0000)

端口描述

QDMA Subsystem for PCIe 可直接连接到 PCIe Integrated Block。根据 PCIe Integrated Block IP 的配置，到此 IP 的数据路径接口位宽为 128、256 或 512 位，运行速率上限为 250 MHz。数据路径宽度适用于所有数据接口。与该核关联的端口如下所述。

表 34：**参数**

参数名称	描述
PL_LINK_CAP_MAX_LINK_WIDTH	物理通道宽度
C_M_AXI_ADDR_WIDTH	AXI4 主接口地址宽度
C_M_AXI_ID_WIDTH	AXI4 主接口 ID 宽度

表 34：**参数** (续)

参数名称	描述
C_M_AXI_DATA_WIDTH	AXI4 主接口数据宽度 128、256 或 512 位
C_S_AXI_ID_WIDTH	AXI4 Bridge 从接口 ID 宽度
C_S_AXI_ADDR_WIDTH	AXI4 Bridge 从接口地址宽度
C_S_AXI_DATA_WIDTH	AXI4 Bridge 从接口数据宽度 128、256 或 512 位
C_S_AXI_ID_WIDTH	AXI4 Bridge 从接口 ID 宽度
AXI_DATA_WIDTH	AXI4 DMA 传输数据宽度。 示例：128、256 或 512 位

相关信息

QDMA 架构

QDMA 全局端口

表 35：**QDMA 全局端口描述**

端口名称	I/O	描述
sys_clk	输入	应从参考时钟 IBUFDS_GTE4 的 ODIV2 端口驱动。请参阅《UltraScale+ Integrated Block for PCI Express LogiCORE IP 产品指南》(PG213)。
sys_clk_gt	输入	PCIe 参考时钟。应从参考时钟 IBUFDS_GTE4 的端口驱动。请参阅《UltraScale+ Integrated Block for PCI Express LogiCORE IP 产品指南》(PG213)。
sys_rst_n	输入	从 PCIe 边缘连接器复位信号进行复位。
pci_exp_txp [PL_LINK_CAP_MAX_LINK_WIDTH-1:0]	输出	PCIe TX 串行接口。
pci_exp_txn [PL_LINK_CAP_MAX_LINK_WIDTH-1:0]	输出	PCIe TX 串行接口。
pci_exp_rxp [PL_LINK_CAP_MAX_LINK_WIDTH-1:0]	输入	PCIe RX 串行接口。
pci_exp_rxn [PL_LINK_CAP_MAX_LINK_WIDTH-1:0]	输入	PCIe RX 串行接口。
user_lnk_up	输出	输出处于高电平有效表示 PCI Express 核已正常链接至主机器件。
axi_aclk	输出	用户时钟输出。PCIe 衍生时钟输出，用于来自 QDMA 的所有接口信号输出和输入 QDMA 的所有接口信号。此时钟用于驱动来自 QDMA 的输入和门电路输出。
axi_aresetn	输出	用户复位输出。AXI 复位信号，与 axi_aclk 输出上提供的时钟同步。此复位应驱动所有对应的 AXI Interconnect aresetn 信号。
soft_reset_n	输入	软核复位（低电平有效）。此端口用于断言复位有效，并复位 DMA 逻辑。它仅复位 DMA 逻辑。用户应负责对此端口执行断言有效和断言无效。
phy_ready	输出	Phy 就绪输出状态。

表 35：**QDMA 全局端口描述** (续)

端口名称	I/O	描述
csr_prog_done	输出	仅当选中 IP 自定义 GUI 的“Basic”（基本）选项卡中的“AXI-Lite CSR Slave Interface”（AXI-Lite CSR 从接口）选项时，才会启用此端口。该端口指示是否可以访问 AXI4-Lite CSR 接口。 1'b0：AXI4-Lite CSR 从接口不可访问。 1'b1：AXI4-Lite CSR 从接口可访问。

所有 AXI 接口均由 `axi_aclk` 信号进行输入输出时钟设置。您负责使用 `axi_aclk` 将所有信号驱动到 DMA 内。

AXI Bridge 主端口

表 36：**AXI4 存储器映射主 Bridge 读取地址接口端口描述**

信号名称	I/O	描述
m_axib_araddr [C_M_AXI_ADDR_WIDTH-1:0]	输出	此信号为存储器映射读取地址（从主机到用户逻辑）。
m_axib_arid [C_M_AXI_ID_WIDTH-1:0]	输出	主读取地址 ID。
m_axib_arlen[7:0]	输出	主读取地址长度。
m_axib_arsize[2:0]	输出	主读取地址大小。
m_axib_arprot[2:0]	输出	主接口读取保护类型。
m_axib_arvalid	输出	此信号断言有效即表示存在发射到 m_axib_araddr 上的地址的有效读取请求。
m_axib_arready	输入	主读取地址就绪。
m_axib_arlock	输出	主接口读取锁定类型。
m_axib_arcache[3:0]	输出	主接口读取存储器类型。
m_axib_arburst[1:0]	输出	主读取地址突发类型。
m_axib_aruser[28:0]	输出	主接口读取用户位。 m_axib_aruser[10:0] = 保留 m_axib_aruser[11] = bridge 流量 m_axib_aruser[15:12] = BAR ID m_axib_aruser[18:16] = 保留 m_axib_aruser[30:19] = 功能编号 m_axib_aruser[31] = 保留 m_axib_aruser[39:32] = 总线编号 m_axib_aruser[42:40] = vf 组 m_axib_aruser[54:43] = vfg 偏移

表 37：**AXI4 存储器映射主 Bridge 读取接口端口描述**

信号名称	I/O	描述
m_axib_rdata [C_M_AXI_DATA_WIDTH-1:0]	输入	主读取数据。
m_axib_ruser [C_M_AXI_DATA_WIDTH/8-1:0]	输入	m_axib_ruser[C_M_DATA_WIDTH/8-1:0] = 读取数据奇校验（逐字节）。

表 37：**AXI4 存储器映射主 Bridge 读取接口端口描述** (续)

信号名称	I/O	描述
m_axib_rid [C_M_AXI_ID_WIDTH-1:0]	输入	主读取 ID。
m_axib_rresp[1:0]	输入	主读取响应。
m_axib_rlast	输入	主读取结束。
m_axib_rvalid	输入	主读取有效。
m_axib_rready	输出	主读取就绪。

表 38：**AXI4 存储器映射主 Bridge 写入地址接口端口描述**

信号名称	I/O	描述
m_axib_awaddr [C_M_AXI_ADDR_WIDTH-1:0]	输出	此信号为存储器映射写入地址（从主机到用户逻辑）。
m_axib_awid [C_M_AXI_ID_WIDTH-1:0]	输出	主写入地址 ID。
m_axib_awlen[7:0]	输出	主写入地址长度。
m_axib_awsize[2:0]	输出	主写入地址大小。
m_axib_awburst[1:0]	输出	主写入地址突发类型。
m_axib_awprot[2:0]	输出	主接口写入保护类型。
m_axib_awvalid	输出	此信号断言有效即表示存在发射到 m_axib_araddr 上的地址的有效写入请求。
m_axib_awready	输入	主写入地址就绪。
m_axib_awlock	输出	主接口写入锁定类型。
m_axib_awcache[3:0]	输出	主接口写入存储器类型。
m_axib_awuser[28:0]	输出	主接口写入用户位。 m_axib_aruser[10:0] = 保留 m_axib_aruser[11] = bridge 流量 m_axib_aruser[15:12] = BAR ID m_axib_aruser[18:16] = 保留 m_axib_aruser[30:19] = 功能编号 m_axib_aruser[31] = 保留 m_axib_aruser[39:32] = 总线编号 m_axib_aruser[42:40] = vf 组 m_axib_aruser[54:43] = vfg 偏移

表 39：**AXI4 存储器映射主 Bridge 写入接口端口描述**

信号名称	I/O	描述
m_axib_wdata [C_M_AXI_DATA_WIDTH-1:0]	输出	主写入数据。
m_axib_wuser [C_M_AXI_DATA_WIDTH/8-1:0]	输出	m_axib_wuser [C_M_AXI_DATA_WIDTH/8-1:0] = 写入数据的奇校验（逐字节）。
m_axib_wlast	输出	主写入结束。
m_axib_wstrb [C_M_AXI_DATA_WIDTH/8-1:0]	输出	主写入选通。

表 39：**AXI4 存储器映射主 Bridge 写入接口端口描述** (续)

信号名称	I/O	描述
m_axib_wvalid	输出	主写入有效。
m_axib_wready	输入	主写入就绪。

表 40：**AXI4 存储器映射主 Bridge 写入响应接口端口描述**

信号名称	I/O	描述
m_axib_bvalid	输入	主写入响应有效。
m_axib_bresp[1:0]	输入	主写入响应。
m_axib_bid [C_M_AXI_ID_WIDTH-1:0]	输入	主写入响应 ID。
m_axib_bready	输出	主响应就绪。

AXI Bridge 从端口

表 41：**AXI4 Bridge 从写入地址接口端口描述**

端口名称	I/O	描述
s_axib_awid [C_S_AXI_ID_WIDTH-1:0]	输入	从写入地址 ID。
s_axib_awaddr [C_S_AXI_ADDR_WIDTH-1:0]	输入	从写入地址。
s_axib_awuser[7:0]	输入	s_axib_awuser[7:0] 表示 function_number。
s_axib_awregion[3:0]	输入	从写入区域解码。
s_axib_awlen[7:0]	输入	从写入突发长度。
s_axib_awsize[2:0]	输入	从写入突发大小。
s_axib_awburst[1:0]	输入	从写入突发类型。仅支持 INCR 突发类型。
s_axib_awvalid	输入	从地址写入有效。
s_axib_awready	输出	从地址写入就绪。

表 42：**AXI4 Bridge 从写入接口端口描述**

端口名称	I/O	描述
s_axib_wdata [C_S_AXI_DATA_WIDTH-1:0]	输入	从写入数据。
s_axib_wstrb [C_S_AXI_DATA_WIDTH/8-1:0]	输入	从写入选通。
s_axib_wlast	输入	从写入结束。
s_axib_wvalid	输入	从写入有效。
s_axib_wready	输出	从写入就绪。
s_axib_wuser [C_S_AXI_DATA_WIDTH/8-1:0]	输入	s_axib_wuser [C_S_AXI_DATA_WIDTH/8-1:0] = 写入数据奇校验（逐字节）。

表 43：**AXI4 Bridge 从写入响应接口端口描述**

端口名称	I/O	描述
s_axib_bid [C_S_AXI_ID_WIDTH-1:0]	输出	从响应 ID。
s_axib_bresp[1:0]	输出	从写入响应。
s_axib_bvalid	输出	从写入响应有效。
s_axib_bready	输入	从响应就绪。

表 44：**AXI4 Bridge 从读取地址接口端口描述**

端口名称	I/O	描述
s_axib_arid [C_S_AXI_ID_WIDTH-1:0]	输入	从读取地址 ID。
s_axib_araddr [C_S_AXI_ADDR_WIDTH-1:0]	输入	从读取地址。
s_axib_arregion[3:0]	输入	从读取区域解码。
s_axib_arlen[7:0]	输入	从读取突发长度。
s_axib_arsize[2:0]	输入	从读取突发大小。
s_axib_arburst[1:0]	输入	从读取突发类型。仅支持 INCR 突发类型。
s_axib_arvalid	输入	从读取地址有效。
s_axib_arready	输出	从读取地址就绪。

表 45：**AXI4 Bridge 从读取接口端口描述**

端口名称	I/O	描述
s_axib_rid [C_S_AXI_ID_WIDTH-1:0]	输出	从读取 ID 标签。
s_axib_rdata [C_S_AXI_ID_WIDTH-1:0]	输出	从读取地址。
s_axib_ruser [C_S_AXI_DATA_WIDTH/8-1:0]	输出	s_axib_aruser[C_S_AXI_ID_WIDTH/8-1:0] = 读取数据奇校验（逐字节）。
s_axib_rresp[1:0]	输出	从读取响应。
s_axib_rlast	输出	从读取结束。
s_axib_rvalid	输出	从读取有效。
s_axib_rready	输入	从读取就绪。

AXI4-Lite 主端口

表 46：**配置 AXI4-Lite 存储器映射写入主接口端口描述**

信号名称	I/O	描述
m_axil_awaddr[31:0]	输出	此信号为存储器映射写入地址（从主机到用户逻辑）。
m_axil_awprot[2:0]	输出	保护类型。

表 46：**配置 AXI4-Lite 存储器映射写入主接口端口描述** (续)

信号名称	I/O	描述
m_axil_awvalid	输出	此信号断言有效即表示存在发射到 m_axil_awaddr 上的地址的有效写入请求。
m_axil_awready	输入	主写入地址就绪。
m_axil_awuser [54:0]	输出	m_axil_awuser[11:0] = 保留 m_axil_awuser[15:12] = bar id m_axil_awuser[18:16] = 保留 m_axil_awuser[30:19] = 功能编号 m_axil_awuser[31] = 保留 m_axil_awuser[39:32] = 总线编号 m_axil_awuser[42:40] = vf 组 m_axil_awuser[54:43] = vfg 偏移
m_axil_wdata[31:0]	输出	主写入数据。
m_axil_wstrb[3:0]	输出	主写入选通。
m_axil_wvalid	输出	主写入有效。
m_axil_wready	输入	主写入就绪。
m_axil_bvalid	输入	主响应有效。
m_axil_bresp[1:0]	输入	
m_axil_bready	输出	主响应有效。

表 47：**配置 AXI4-Lite 存储器映射读取主接口端口描述**

信号名称	I/O	描述
m_axil_araddr[31:0]	输出	此信号为存储器映射读取地址（从主机到用户逻辑）。
m_axil_aruser[54:0]	输出	m_axil_aruser[11:0] = 保留 m_axil_aruser[15:12] = bar id m_axil_aruser[18:16] = 保留 m_axil_aruser[30:19] = 功能编号 m_axil_aruser[31] = 保留 m_axil_aruser[39:32] = 总线编号 m_axil_aruser[42:40] = vf 组 m_axil_aruser[54:43] = vfg 偏移
m_axil_arprot[2:0]	输出	保护类型。
m_axil_arvalid	输出	此信号断言有效即表示存在发射到 m_axil_araddr 上的地址的有效读取请求。
m_axil_arready	输入	主读取地址就绪。
m_axil_rdata[31:0]	输入	主读取数据。
m_axil_rresp[1:0]	输入	主读取响应。
m_axil_rvalid	输入	主读取有效。
m_axil_rready	输出	主读取就绪。

AXI4-Lite 从端口

AXI4-Lite 从端口可用于访问 QDMA 队列空间寄存器（QDMA_TRQ_SEL_QUEUE_PF (0x18000) 和 QDMA_TRQ_SEL_QUEUE_VF (0x3000)）。

表 48：**配置 AXI4-Lite 存储器映射写入从接口信号**

信号名称	I/O	描述
s_axil_awaddr[31:0]	输入	此信号是存储器映射写入地址（从用户逻辑到 DMA 队列空间寄存器）。
s_axil_awvalid	输入	此信号断言有效即表示存在发射到 s_axil_awaddr 上的地址的有效写入请求。
s_axil_awuser[12:0]	输入	[12:8]：保留 [7:0]：功能编号
s_axil_awprot[2:0]	输入	保护类型。当前不使用此端口。
s_axil_awready	输出	从写入地址就绪。
s_axil_wdata[31:0]	输入	从写入数据。
s_axil_wstrb[3:0]	输入	从写入选通。
s_axil_wvalid	输入	从写入有效。
s_axil_wready	输出	从写入就绪。
s_axil_bvalid	输出	从写入响应有效。
s_axil_bresp[1:0]	输出	从写入响应。
s_axil_bready	输入	保存响应就绪。

表 49：**配置 AXI4-Lite 存储器映射读取从接口信号**

信号名称	I/O	描述
s_axil_araddr[31:0]	输入	此信号是存储器映射读取地址（从用户逻辑到 DMA 队列空间）。
s_axil_arprot[2:0]	输入	保护类型。当前不使用此端口。
s_axil_arvalid	输入	此信号断言有效即表示存在发射到 s_axil_araddr 上的地址的有效读取请求。
s_axil_aruser[12:0]	输入	[12:8]：保留 [7:0]：功能编号
s_axil_arready	输出	从读取地址就绪。
s_axil_rdata[31:0]	输出	从读取地址。
s_axil_rresp[1:0]	输出	从读取响应。
s_axil_rvalid	输出	从读取有效。
s_axil_rready	输入	从读取就绪。

AXI4-Lite CSR 从接口端口

表 50：**配置 AXI4-Lite 存储器映射写入 CSR 从接口信号**

信号名称	I/O	描述
s_axil_csr_awaddr[31:0]	输入	此信号为存储器映射写入地址（从用户逻辑到 DMA）。 s_axil_csr_awaddr[15]: 1'b1 – QDMA CSR 寄存器 1'b0 – Bridge 寄存器
s_axil_csr_awvalid	输入	此信号的断言有效表示存在有效的写入请求，该请求针对 s_axil_csr_awaddr 上的地址。
s_axil_csr_awprot[2:0]	输入	保护类型。当前不使用此端口。
s_axil_csr_awready	输出	从写入地址就绪。
s_axil_csr_wdata[31:0]	输入	从写入数据。
s_axil_csr_wstrb[3:0]	输入	从写入选通。
s_axil_csr_wvalid	输入	从写入有效。
s_axil_csr_wready	输出	从写入就绪。
s_axil_csr_bvalid	输出	从写入响应有效。
s_axil_csr_bresp[1:0]	输出	从写入响应。
s_axil_csr_bready	输入	保存响应就绪。

表 51：**配置 AXI4-Lite 存储器映射读取 CSR 从接口信号**

信号名称	I/O	描述
s_axil_csr_araddr[31:0]	输入	此信号为存储器映射读取地址（从用户逻辑到 DMA）。 s_axil_csr_araddr[15]: 1'b1 – QDMA 寄存器 1'b0 – Bridge 寄存器
s_axil_csr_arprot[2:0]	输入	保护类型。当前不使用此端口。
s_axil_csr_arvalid	输入	此信号的断言有效表示存在有效的读取请求，该请求针对 s_axil_csr_araddr 上的地址。
s_axil_csr_arready	输出	从读取地址就绪。
s_axil_csr_rdata[31:0]	输出	从读取地址。
s_axil_csr_rresp[1:0]	输出	从读取响应。
s_axil_csr_rvalid	输出	从读取有效。
s_axil_csr_rready	输入	从读取就绪。

AXI4 存储器映射 DMA 端口

表 52：**AXI4 存储器映射 DMA 读取地址接口信号**

信号名称	方向	描述
m_axi_araddr [C_M_AXI_ADDR_WIDTH-1:0]	输出	此信号为存储器映射读取地址（从 DMA 到用户逻辑）。
m_axi_arid [3:0]	输出	标准 AXI4 描述，可在 AXI4 协议规范《AMBA AXI4-Stream 协议规范》(ARM IHI 0051A) 中找到。

表 52：**AXI4 存储器映射 DMA 读取地址接口信号** (续)

信号名称	方向	描述
m_axi_aruser[31:0]	输出	m_axi_aruser[18:0] = 保留 m_axi_aruser[31:19] = 队列号
m_axi_arlen[7:0]	输出	主读取突发长度。
m_axi_arsize[2:0]	输出	主读取突发大小。
m_axi_arprot[2:0]	输出	保护类型。
m_axi_arvalid	输出	此信号断言有效即表示存在发射到 m_axi_araddr 上的地址的有效读取请求。
m_axi_arready	输入	主读取地址就绪。
m_axi_arlock	输出	锁定类型。
m_axi_arcache[3:0]	输出	存储器类型。
m_axi_arburst[1:0]	输出	主读取突发类型。

表 53：**AXI4 存储器映射 DMA 读取接口信号**

信号名称	方向	描述
m_axi_rdata [C_M_AXI_DATA_WIDTH-1:0]	输入	主读取数据。
m_axi_rid [3:0]	输入	主读取 ID。
m_axi_rresp[1:0]	输入	主读取响应。
m_axi_rlast	输入	主读取结束。
m_axi_rvalid	输入	主读取有效。
m_axi_rready	输出	主读取就绪。
m_axi_ruser [C_M_AXI_DATA_WIDTH/8-1:0]	输入	主接口读取奇校验（逐字节）。此端口仅在“Data Protection”（数据保护）模式下才启用。

表 54：**AXI4 存储器映射 DMA 写入地址接口信号**

信号名称	方向	描述
m_axi_awaddr [C_M_AXI_ADDR_WIDTH-1:0]	输出	此信号为存储器映射写入地址（从 DMA 到用户逻辑）。
m_axi_awid[3:0]	输出	主写入地址 ID。
m_axi_awuser[31:0]	输出	m_axi_awuser[18:0] = 保留 m_axi_awuser[31:19] = 队列号
m_axi_awlen[7:0]	输出	主写入地址长度。
m_axi_awsize[2:0]	输出	主写入地址大小。
m_axi_awburst[1:0]	输出	主写入地址突发类型。
m_axi_awprot[2:0]	输出	保护类型。
m_axi_awvalid	输出	此信号断言有效即表示存在发射到 m_axi_araddr 上的地址的有效写入请求。
m_axi_awready	输入	主写入地址就绪。
m_axi_awlock	输出	锁定类型。

表 54：**AXI4 存储器映射 DMA 写入地址接口信号** (续)

信号名称	方向	描述
m_axi_awcache[3:0]	输出	存储器类型。

表 55：**AXI4 存储器映射 DMA 写入接口信号**

信号名称	方向	描述
m_axi_wdata [C_M_AXI_DATA_WIDTH-1:0]	输出	主写入数据。
m_axi_wlast	输出	主写入结束。
m_axi_wstrb[31:0]	输出	主写入选通。
m_axi_wvalid	输出	主写入有效。
m_axi_wready	输入	主写入就绪。
m_axi_wuser [C_M_AXI_DATA_WIDTH/8-1:0]	输出	主接口写入用户。 m_axi_wuser[C_M_AXI_DATA_WIDTH/8-1:0] = 写入数据奇校验（逐字节）。此端口仅在“Data Protection”（数据保护）模式下才启用。

表 56：**AXI4 存储器映射 DMA 写入响应接口信号**

信号名称	方向	描述
m_axi_bvalid	输入	主写入响应有效。
m_axi_bresp[1:0]	输入	主写入响应。
m_axi_bid[3:0]	输入	主响应 ID。
m_axi_bready	输出	主响应就绪。

AXI4-Stream H2C 端口

表 57：**AXI4-Stream H2C 端口描述**

端口名称	I/O	描述
m_axis_h2c_tdata [AXI_DATA_WIDTH-1:0]	输出	H2C AXI4-Stream 的数据输出。
m_axis_h2c_tcrc [31:0]	输出	此节拍的 32 位 CRC 值。 IEEE 802.3 CRC-32 多项式
m_axis_h2c_tuser_qid[10:0]	输出	队列 ID
m_axis_h2c_tuser_port_id[2:0]	输出	端口 ID
m_axis_h2c_tuser_err	输出	如果该端口置位，则表示包存在错误。此错误可能来自 PCIe，或者可能在 DMA 传输中存在错误。赛灵思建议您查看错误寄存器和上下文以获取详细信息。 当 DMA 首次检测到错误时，错误位将设为 1。并且对于该包的其余部分，该错误位将置位。如果该包中没有错误，那么对于下一个包，该错误位将复位为 0。
m_axis_h2c_tuser_mdata[31:0]	输出	元数据 在内部模式下，QDMA 会在此字段上传递 H2C AXI4-Stream 描述符的低 32 位。
m_axis_h2c_tuser_mty[5:0]	输出	传输事务的最后一个节拍上无效的字节数。对于 64B 传输，此字段为 0。

表 57：**AXI4-Stream H2C 端口描述** (续)

端口名称	I/O	描述
m_axis_h2c_tuser_zero_byte	输出	当该端口置位时，表示当前节拍为空节拍（当前传输的字节数为 0）。
m_axis_h2c_tvalid	输出	有效
m_axis_h2c_tlast	输出	表示这是包传输中的最后一个周期
m_axis_h2c_tready	输入	就绪

AXI4-Stream C2H 端口

表 58：**AXI4-Stream C2H 端口描述**

端口名称	I/O	描述
s_axis_c2h_tdata [AXI_DATA_WIDTH-1:0]	输入	支持 128 位、256 位和 512 位数据宽度。每个 C2H 数据包都有对应的 C2H 完成包。
s_axis_c2h_tcrc [31:0]	输入	该节拍的 32 位 CRC 值。 IEEE 802.3 CRC-32 多项式 仅当断言 `s_axis_c2h_tlast` 有效时，IP 才会对 CRC 值进行采样。
s_axis_c2h_ctrl_len [15:0]	输入	包的长度。对于 0 字节写入，长度为 0。 C2H 串流包数据长度限制为 31 * c2h 缓冲器尺寸。 在旧版本（例如，2018.3）中，C2H 串流包数据长度限制为 7 * C2H 缓冲器大小。 ctrl_len 以字节为单位，应在包的第一拍内有效。
s_axis_c2h_ctrl_qid [10:0]	输入	队列 ID。
s_axis_c2h_ctrl_has_cmpt	输入	1'b1：数据包含完成。 1'b0：数据包不含完成。
s_axis_c2h_ctrl_marker	输入	此标记报文用于确保流水线已完全刷新。随后，您即可安全执行队列无效化。
s_axis_c2h_ctrl_port_id [2:0]	输入	端口 ID。
s_axis_c2h_ctrl_ecc[6:0]	输入	用于 C2H 控制信号的边带保护。赛灵思 Error Correction Code (ECC) 核的输出。ECC IP 输入描述如下。
s_axis_c2h_mty [5:0]	输入	空字节应在最后一拍置位。
s_axis_c2h_tvalid	输入	有效。
s_axis_c2h_tlast	输入	指示最后一个包。
s_axis_c2h_tready	输出	就绪。

要为 C2H 控制总线 `s_axis_c2h_ctrl_ecc[6:0]` 生成 ECC 信号，请使用赛灵思 Error Correction Code (ECC) IP。以下列出了 ECC IP 的输入信号，且所列信号顺序必须保留不变。

使用 ecc_data_in[56:0] 输入 ECC IP

```
assign ecc_data_in[56:0] = { 24'h0, //reserved
                    s_axis_c2h_ctrl_has_cmpt, //has compt
                    s_axis_c2h_ctrl_marker, //marker
                    s_axis_c2h_ctrl_port_id, //port_id
                    1'b0, // reserved should be set to 0.
                    s_axis_c2h_ctrl_qid, // Qid
                    s_axis_c2h_ctrl_len}; //length
```

AXI4-Stream C2H 完成端口

表 59：**AXI4-Stream C2H 完成端口描述**

端口名称	I/O	描述
s_axis_c2h_cmpt_tdata[511:0]	输入	来自用户应用的完成数据。其中包含写入主机中的完成环的信息。
s_axis_c2h_cmpt_size [1:0]	输入	00：8B 完成。 01：16B 完成。 10：32B 完成。 11：64B 完成
s_axis_c2h_cmpt_dpar [15:0]	输入	奇校验按位（以 32b 为单位）计算。 s_axis_c2h_cmpt_dpar[0] 是基于 s_axis_c2h_cmpt_tdata[31:0] 的奇偶校验。 s_axis_c2h_cmpt_dpar[1] 是基于 s_axis_c2h_cmpt_tdata[63:31] 的奇偶校验，以此类推。
s_axis_c2h_cmpt_ctrl_qid[10:0]	输入	完成队列 ID。
s_axis_c2h_cmpt_ctrl_marker	输入	此标记报文用于确保流水线已完全刷新。随后，您即可安全执行队列无效化。
s_axis_c2h_cmpt_ctrl_user_trig	输入	用户可以触发中断和状态描述符写入（如两者均已启用）。
s_axis_c2h_cmpt_ctrl_cmpt_type[1:0]	输入	2'b00：NO_PLD_NO_WAIT。CMPT 包不含对应有效载荷包，且无需等待。 2'b01：NO_PLD_BUT_WAIT。CMPT 包不含对应有效载荷包；但仍需等待有效载荷包完成发送后才能发送 CMPT 包。 2'b10：RSVD。 2'b11：HAS_PLD。CMPT 包具有对应的有效载荷包，需等待有效载荷包完成发送后才能发送 CMPT 包。
s_axis_c2h_cmpt_ctrl_wait_pld_pkt_id[15:0]	输入	CMPT 包发送前需等待的数据有效载荷包 ID。
s_axis_c2h_cmpt_ctrl_port_id[2:0]	输入	端口 ID。
s_axis_c2h_cmpt_ctrl_col_idx[2:0]	输入	该颜色索引用于定义用户是否希望在 CMPT 包内包含颜色位，以及颜色位（如果存在）的位元位置。
s_axis_c2h_cmpt_ctrl_err_idx[2:0]	输入	该错误索引用于定义用户是否希望在 CMPT 包内包含错误位，以及错误位（如果存在）的位元位置。
s_axis_c2h_cmpt_ctrl_no_wrb_marker	输入	在标记传输期间禁用 CMPT 包。 1'b0：CMPT 包发送至 CMPT 环路 1'b1：CMPT 包不发送至 CMPT 环路。
s_axis_c2h_cmpt_tvalid	输入	有效。s_axis_c2h_cmpt_tvalid 必须保持断言有效，直至 s_axis_c2h_cmpt_tready 断言有效为止。
s_axis_c2h_cmpt_tready	输出	就绪。

AXI4-Stream 状态端口

表 60：**AXI-ST C2H 状态端口描述**

端口名称	I/O	描述
axis_c2h_status_valid	输出	有效（按描述符）。
axis_c2h_status_qid [10:0]	输出	包的 QID。

表 60：**AXI-ST C2H 状态端口描述** (续)

端口名称	I/O	描述
axis_c2h_status_drop	输出	如果 QDMA Subsystem for PCIe 没有足够描述符用于将整个包传输到主机，就会将该包丢弃。该位用于指示包是否已丢弃。未丢弃的包则会被视为已接受。 0：包未被丢弃。 1：包已被丢弃。
axis_c2h_status_last	输出	最后一个描述符。
axis_c2h_status_cmp	输出	0：已丢弃的包或 has_cmpt 为 1'b0 的 C2H 包。 1：含完成的 C2H 包。
axis_c2h_status_error	输出	当 axis_c2h_status_error 设为 1 时，提取的描述符有一个错误。当设为 0 时，没有错误。

AXI4-Stream C2H 写入完成端口

表 61：**AXI-ST C2H 写入完成端口描述**

端口名称	I/O	描述
axis_c2h_dmawr_cmp	输出	当包的最后一个数据有效载荷写入请求变为写入完成时，此信号断言有效。每个包一个脉冲。

VDM 端口

表 62：**VDM 端口描述**

端口名称	I/O	描述
st_rx_msg_valid	输出	有效
st_rx_msg_data[31:0]	输出	节拍 1： {REQ_ID[15:0], VDM_MSG_CODE[7:0], VDM_MSG_ROUTING[2:0], VDM_DW_LENGTH[4:0]} 节拍 2： VDM Lower Header [31:0] 或 {(Payload_length=0), VDM Higher Header [31:0]} 节拍 3 到节拍 <n>： VDM Payload
st_rx_msg_last	输出	表示最后一拍
st_rx_msg_rdy	输入	就绪。 **注释：**不使用此接口时，Ready（就绪）必须绑定到 1。

RX 供应商定义报文先存储在浅 FIFO 中，然后再发射到输出端口。如有大量连续 VDM 报文，FIFO 会发生上溢，这些报文将被丢弃。最好定期重复 VDM 报文。

配置扩展接口端口

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。

表 63：**配置扩展接口端口描述**

端口名称	I/O	宽度	描述
cfg_ext_read_received	输出	1	已接收配置扩展读取 核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“User Defined Configuration Capabilities”（用户定义的配置功能）选项卡中选中“PCI Express Extended Configuration Space Enable”（PCI Express 扩展配置空间启用），即可设置此端口。 · 对于接收到的配置读取，如果其所含 cfg_ext_register_number 在 0xb0-0xbf 范围内，则所有这些配置读取都被视为 PCIe 遗留扩展配置空间。 · 对于接收到的配置读取，如果其所含 cfg_ext_register_number 在 0x120-13F 范围内，那么所有这些配置读取都被视为 PCIe 扩展配置空间。 · 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）。有效数据在 cfg_ext_register_number 和 cfg_ext_function_number 上驱动。 · IP 范围外的用户应用只需对接收到的上述两个范围内的配置读取进行响应即可。
cfg_ext_write_received	输出	1	已接收配置扩展写入 核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“Capabilities”（功能）选项卡中选中“PCI Express Extended Configuration Space Enable”，即可设置此端口。 · 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。 · 对于接收到的配置写入，如果所含 cfg_ext_register_number 在 0x120-13F 范围内，那么所有这些配置写入都会显示在 cfg_ext_register_number、cfg_ext_function_number、cfg_ext_write_data 和 cfg_ext_write_byte_enable 上。
cfg_ext_register_number	输出	10	配置扩展寄存器编号 读取或写入的配置寄存器的 10 位地址。当 cfg_ext_read_received 或 cfg_ext_write_received 为高电平时，数据有效。
cfg_ext_function_number	输出	8	配置扩展功能编号。 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。
cfg_ext_write_data	输出	32	配置扩展写入数据 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。
cfg_ext_write_byte_enable	输出	4	配置扩展写入字节使能 针对配置写入传输事务的字节使能。
cfg_ext_read_data	输入	32	配置扩展读取数据 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上对此数据进行采样。

表 63：**配置扩展接口端口描述** (续)

端口名称	I/O	宽度	描述
cfg_ext_read_data_valid	输入	1	配置扩展读取数据有效 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。如果此时没有收到响应，此核将发送使用含 'h0 有效载荷的自动响应，并且用户应用必须丢弃响应并立即终止该特定请求

FLR 端口

表 64：**FLR 端口描述**

端口名称	I/O	描述
usr_flr_fnc [7:0]	输出	功能 FLR 状态更改的功能编号。
usr_flr_set	输出	置位 断言有效并保持 1 个周期即表示 usr_flr_fnc[7:0] 上指示的功能当前处于 FLR 状态。
usr_flr_done_fnc [7:0]	输入	完成功能 用户逻辑已完成的 FLR 功能。
usr_flr_done_vld	输入	完成有效 断言有效并保持 1 个周期即表示 usr_flr_done_fnc[7:0] 上的功能的 FLR 已完成。

QDMA 描述符旁路输入端口

表 65：**QDMA H2C 串流旁路输入端口描述**

端口名称	I/O	描述
h2c_byp_in_st_addr [63:0]	输入	DMA 传输的 64 位起始地址。
h2c_byp_in_st_len [15:0]	输入	要传输的字节数。
h2c_byp_in_st_sop	输入	指示包起始。针对第一个描述符置位。针对描述符的剩余部分复位。
h2c_byp_in_st_eop	输入	指示包结束。针对最后一个描述符置位。针对描述符的剩余部分复位。
h2c_byp_in_st_sdi	输入	H2C 旁路输入状态描述符/中断 如果设置此端口，则将其视为用户应用指示 QDMA 将状态描述符发送到主机，并在 QDMA 提取与此描述符关联的数据的最后一个字节时向主机生成中断。仅当已在 H2C SW 上下文中为此 QID 启用中断，并且驱动程序已装备中断时，QDMA 才会遵循请求以生成中断。只能为 EOP 描述符设置此项。 如果最后一个没有 h2c_byp_in_st_sdi 的描述符存在错误，那么 QDMA 将挂起。这会导致写回和 hw_ctxt.dsc_pend 位两者均缺失并且无限期断言有效。变通方法是发送一个长度为零的描述符，以触发完成 (CMPT) 状态。 **建议：**出于性能原因，赛灵思建议在 32 或 64 个描述符中对该端口进行一次断言有效，如果没有剩余描述符，则在最后一个描述符处断言有效。

表 65：**QDMA H2C 串流旁路输入端口描述** (续)

端口名称	I/O	描述
h2c_byp_in_st_mrkr_req	输入	H2C 旁路输入标记请求 置位后，描述符在 H2C 引擎流水线中传递，完成后，在队列状态端口接口上生成标记响应。只能为 EOP 描述符设置此项。
h2c_byp_in_st_no_dma	输入	H2C 旁路输入无 DMA 当通过此接口传入描述符并将此信号断言有效时，它会通知 QDMA 不要发送针对该描述符的任何 PCIe 请求。由于没有传出任何 PCIe 请求，因此 H2C 串流输出接口上不会发出任何对应的 DMA 数据。 此端口通常与 h2c_byp_in_st_sdi 结合使用，这样当用户逻辑已用完实际描述符并且仍要驱动 h2c_byp_in_st_sdi 信号时，就会导致状态描述符/中断。 如果在传入非 DMA 描述符时将 h2c_byp_in_st_mrkr_req 和 h2c_byp_in_st_sdi 复位，则描述符将作为 NOP 来处理，并完全在 QDMA 内部使用，不会产生任何接口活动。 如果设置了 h2c_byp_in_st_no_dma，那么必须设置 h2c_byp_in_st_sop 和 h2c_byp_in_st_eop。 如果设置了 h2c_byp_in_st_no_dma，那么 QDMA 将忽略此接口的地址和长度字段。
h2c_byp_in_st_qid [10:0]	输入	与 H2C 描述符环关联的 QID。
h2c_byp_in_st_error	输入	通过对该位进行置位即可指示队列中存在错误。此描述符将不予处理。将更新上下文以反映队列中的错误。
h2c_byp_in_st_func [7:0]	输入	PCIe 函数 ID
h2c_byp_in_st_cidx [15:0]	输入	此 CIDX 将用于状态描述符更新和/或中断（聚合模式）。通常，在从描述符旁路输出接口接收到 CIDX 后，它就应保持不变。
h2c_byp_in_st_port_id [2:0]	输入	QDMA 端口 ID
h2c_byp_in_st_vld	输入	有效。High 表示描述符有效，一个脉冲针对一个描述符。
h2c_byp_in_st_rdy	输出	已准备好接收描述符

表 66：**QDMA H2C-MM 描述符旁路输入端口描述**

端口名称	I/O	描述
h2c_byp_in_mm_radr[63:0]	输入	DMA 数据的读取地址。
h2c_byp_in_mm_wadr[63:0]	输入	DMA 数据的写入地址。
h2c_byp_in_mm_no_dma	输入	H2C 旁路输入无 DMA 当通过此接口传入描述符并将此信号断言有效时，此信号会通知 QDMA 不要发送针对该描述符的任何 PCIe 请求。由于没有传出任何 PCIe 请求，因此 H2C MM 输出接口上不会发出任何对应的 DMA 数据。 此端口通常与 h2c_byp_in_mm_sdi 结合使用，这样当用户逻辑已用完实际描述符并且仍要驱动 h2c_byp_in_mm_sdi 信号时，就会导致状态描述符/中断。 如果在传入非 DMA 描述符时将 h2c_byp_in_mm_mrkr_req 和 h2c_byp_in_mm_sdi 复位，则描述符将作为“无操作”(NOP) 来处理，并完全在 QDMA 内部使用，不会产生任何接口活动。 如何设置了 h2c_byp_in_mm_no_dma，那么 QDMA 将忽略地址。长度字段应设置为 0。
h2c_byp_in_mm_len[27:0]	输入	DMA 数据长度。 高 12 位必须绑定到 0。因此，只有该字段的低 16 位可用于指定长度。

表 66：**QDMA H2C-MM 描述符旁路输入端口描述** (续)

端口名称	I/O	描述
h2c_byp_in_mm_sdi	输入	H2C-MM 旁路输入状态描述符/中断 如果设置此端口，则将其视为用户指示 QDMA 将状态描述符发送到主机，并在 QDMA 提取与此描述符关联的数据的最后一个字节时向主机生成中断。仅当已在 H2C 环上下文中为此 QID 启用中断，并且驱动程序已装备中断时，QDMA 才会遵循请求以生成中断。 如果最后一个没有 h2c_byp_in_mm_sdi 的描述符存在错误，那么 QDMA 将挂起。这会导致写回和 hw_ctxt.dsc_pend 位两者均缺失并且无限期断言有效。变通方法是发送一个长度为零的描述符，以触发完成 (CMPT) 状态。 **建议：**出于性能原因，赛灵思建议在 32 或 64 个描述符中对该端口进行一次断言有效，如果没有剩余描述符，则在最后一个描述符处断言有效。
h2c_byp_in_mm_mrkr_req	输入	H2C-MM 旁路输入标记请求 表示用户指示 QDMA，一旦完成此描述符的数据传输，就必须向用户发送完成状态。
h2c_byp_in_mm_qid [10:0]	输入	与 H2C 描述符环关联的 QID。
h2c_byp_in_mm_error	输入	通过对该位进行置位即可指示队列中存在错误。此描述符将不予处理。将更新上下文以反映队列中的错误。
h2c_byp_in_mm_func [7:0]	输入	PCIe 函数 ID
h2c_byp_in_mm_cidx [15:0]	输入	此 CIDX 将用于状态描述符更新和/或中断（聚合模式）。通常，在从描述符旁路输出接口接收到 CIDX 后，它就应保持不变。
h2c_byp_in_mm_port_id [2:0]	输入	QDMA 端口 ID
h2c_byp_in_mm_vld	输入	有效。High 表示描述符有效，一个脉冲针对一个描述符。
h2c_byp_in_mm_rdy	输出	已准备好接收描述符

表 67：**QDMA C2H 串流旁路输入端口描述** [1]

端口名称	I/O	描述
c2h_byp_in_st_csh_addr [63:0]	输入	DMA 在当前指定的 64 位地址中写入数据。
c2h_byp_in_st_csh_qid [10:0]	输入	与 C2H 描述符环关联的 QID。
c2h_byp_in_st_csh_error	输入	通过对该位进行置位即可指示队列中存在错误。此描述符将不予处理。将更新上下文以反映队列中的错误。
c2h_byp_in_st_csh_func [7:0]	输入	PCIe 函数 ID
c2h_byp_in_st_csh_port_id[2:0]	输入	QDMA 端口 ID
c2h_byp_in_st_csh_pfch_tag[6:0]	输入	预取标签。预取标签指向用于在预取引擎中存储活动队列的 CAM。采用高速缓存旁路模式时，必须将 `c2h_byp_out_pfch_tag[6:0]` 环回到 `c2h_byp_in_st_csh_pfch_tag[6:0]`。采用简单旁路模式时，用户需要从 MDMA_C2H_PFCH_BYP_TAG (0x140C) 寄存器传入预取标签值。
c2h_byp_in_st_csh_vld	输入	有效。High 表示描述符有效，一个脉冲针对一个描述符。
c2h_byp_in_st_csh_rdy	输出	已准备好接收描述符。

注释：

1. AXI-Stream C2H 简单旁路模式和高速缓存旁路模式使用相同的旁路端口 c2h_byp_in_st_csh_*。

表 68：**QDMA C2H-MM 描述符旁路输入端口描述**

端口名称	I/O	描述
c2h_byp_in_mm_raddr [63:0]	输入	DMA 数据的读取地址。
c2h_byp_in_mm_wadr[63:0]	输入	DMA 数据的写入地址。
c2h_byp_in_mm_no_dma	输入	C2H 旁路输入无 DMA 当通过此接口传入描述符并将此信号断言有效时，此信号会通知 QDMA 不要发送针对该描述符的任何 PCIe 请求。由于没有传出任何 PCIe 请求，因此不会从 C2H MM 接口读取任何对应的 DMA 数据。 此端口通常与 c2h_byp_in_mm_sdi 结合使用，这样当用户逻辑已用完实际描述符并且仍要驱动 c2h_byp_in_mm_sdi 信号时，就会导致状态描述符/中断。 如果在传入非 DMA 描述符时将 c2h_byp_in_mm_mrkr_req 和 c2h_byp_in_mm_sdi 复位，则描述符将作为 NOP 来处理，并完全在 QDMA 内部使用，不会产生任何接口活动。 如何设置了 c2h_byp_in_mm_no_dma，那么 QDMA 将忽略地址。长度字段应设置为 0。
c2h_byp_in_mm_len[27:0]	输入	DMA 数据长度
c2h_byp_in_mm_sdi	输入	C2H 旁路输入状态描述符/中断 如果设置此端口，则将其视为用户指示 QDMA 将状态描述符发送到主机，并在 QDMA 提取与此描述符关联的数据的最后一个字节时向主机生成中断。仅当已在 C2H 环上下文中为此 QID 启用中断，并且驱动程序已装备中断时，QDMA 才会遵循请求以生成中断。 **建议：**出于性能原因，赛灵思建议在 32 或 64 个描述符中对该端口进行一次断言有效，如果没有剩余描述符，则在最后一个描述符处断言有效。
c2h_byp_in_mm_mrkr_req	输入	C2H 旁路输入标记请求 表示用户指示 QDMA，一旦完成此描述符的数据传输，就必须向用户发送完成状态。
c2h_byp_in_mm_qid [10:0]	输入	与 C2H 描述符环关联的 QID
c2h_byp_in_mm_error	输入	通过对该位进行置位即可指示队列中存在错误。此描述符将不予处理。将更新上下文以反映队列中的错误。
c2h_byp_in_mm_func [7:0]	输入	PCIe 函数 ID
c2h_byp_in_mm_cidx [15:0]	输入	用户必须回显它在旁路输出接口上收到的描述符中的 CIDX。
c2h_byp_in_mm_port_id[2:0]	输入	QDMA 端口 ID
c2h_byp_in_mm_vld	输入	有效。High 表示描述符有效，一个脉冲针对一个描述符。
c2h_byp_in_mm_rdy	输出	已准备好接收描述符。

QDMA 描述符旁路输出端口

表 69：**QDMA H2C 描述符旁路输出端口描述**

端口名称	I/O	描述
h2c_byp_out_dsc [255:0]	输出	H2C 描述符提取自主机。 对于 H2C AXI-MM，该子系统使用全部 256 个位，并且这些位的结构与该表 相同。 对于 H2C AXI-ST，子系统使用 [127:0] 位，这些位的结构与 该表 相同。

表 69：**QDMA H2C 描述符旁路输出端口描述** (续)

端口名称	I/O	描述
h2c_byp_out_st_mm	输出	表示此描述符属于串流数据描述符还是存储器映射描述符。 0：串流 1：存储器映射
h2c_byp_out_dsc_sz [1:0]	输出	描述符大小。此字段表示描述符的大小。 0：8B 1：16B 2：32B 3：64B - 64B 描述符将随 2 个有效/就绪周期一起进行传输。第一个周期含最低有效位 32 字节。第二个周期含最高有效位 32 字节。CIDX 和其它队列信息仅在 64B 描述符的第二拍上有效。
h2c_byp_out_qid [10:0]	输出	与 H2C 描述符环关联的 QID。
h2c_byp_out_error	输出	表示描述符提取或执行前一个描述符时遇到错误。
h2c_byp_out_func [7:0]	输出	PCIe 函数 ID
h2c_byp_out_cidx [15:0]	输出	H2C 旁路输出使用者索引 提取的描述符的环索引。在旁路输入接口上提交描述符时，用户必须将该字段回显给 QDMA。
h2c_byp_out_port_id [2:0]	输出	QDMA 端口 ID
h2c_byp_out_fmt[2:0]	输出	格式 此字段的编码如下。 0x0：标准描述符 0x1 - 0x7：保留
h2c_byp_out_vld	输出	有效。High 表示描述符有效，一个脉冲针对一个描述符。
h2c_byp_out_rdy	输入	就绪。不使用此接口时，Ready（就绪）必须绑定到 1。

表 70：**QDMA C2H 描述符旁路输出端口描述**

端口名称	I/O	描述
c2h_byp_out_dsc [255:0]	输出	C2H 描述符提取自主机。 对于 C2H AXI-MM，该子系统使用全部 256 个位，并且这些位的结构与 该表 相同。 对于 C2H AXI-ST，子系统使用 [63:0] 位，这些位的结构与 该表 相同。其余位则忽略。
c2h_byp_out_st_mm	输出	表示此描述符属于串流数据描述符还是存储器映射描述符。 0：串流 1：存储器映射
c2h_byp_out_dsc_sz [1:0]	输出	描述符大小。该字段表示 h2c_byp_out_dsc 上的有效描述符信息量。 0：8B 1：16B 2：32B 3：64B - 64B 描述符将随 2 个有效/就绪周期一起进行传输。第一个周期含最低有效位 32 字节。第二个周期含最高有效位 32 字节。CIDX 和其它队列信息仅在 64B 描述符的第二拍上有效。
c2h_byp_out_qid [10:0]	输出	与 H2C 描述符环关联的 QID。
c2h_byp_out_error	输出	表示描述符提取或执行前一个描述符时遇到错误。
c2h_byp_out_func [7:0]	输出	PCIe 功能 ID。

表 70：**QDMA C2H 描述符旁路输出端口描述** (续)

端口名称	I/O	描述
c2h_byp_out_cidx [15:0]	输出	C2H 旁路输出使用者索引 提取的描述符的环索引。在旁路输入接口上提交描述符时，用户必须将该字段回显给 QDMA。
c2h_byp_out_port_id [2:0]	输出	QDMA 端口 ID
c2h_byp_out_pfch_tag[6:0]	输出	预取标签。预取标签指向用于在预取引擎中存储活动队列的 CAM
c2h_byp_out_fmt[2:0]	输出	格式 此字段的编码如下。 0x0：标准描述符 0x1 - 0x7：保留
c2h_byp_out_vld	输出	有效。High 表示描述符有效，一个脉冲针对一个描述符。
c2h_byp_out_rdy	输入	就绪。不使用此接口时，Ready（就绪）必须绑定到 1。

`h2c_byp_out_vld` 或 `c2h_byp_out_vld` 断言有效并包含 CIDX 值是很常见的；当在上下文编程选择中未设置描述符旁路模式选项时，会出现此状况。在 QDMA IP 核自定义期间，您必须在 Vivado® IDE 中设置描述符旁路模式才能看到描述符旁路输出端口。在 Vivado® IDE 中选中描述符旁路选项，但在上下文编程中，描述符旁路位未置位时，您将看到有效的信号被断言有效并伴随 CIDX 更新。

QDMA 描述符信用值输入端口

表 71：**QDMA 描述符信用值输入端口描述**

端口名称	I/O	描述
dsc_crdt_in_vld	输入	有效。当此端口断言有效时，用户必须在总线上呈现有效数据并保持总线值不变，直至在同一个周期内 valid（有效）和 ready（就绪）同时断言有效为止。
dsc_crdt_in_rdy	输出	就绪。此信号断言有效即表示 DMA 已准备好接受来自此总线的数据。
dsc_crdt_in_dir	输入	指示信用值是用于 H2C 描述符环还是 C2H 描述符环。 0：H2C 1：C2H
dsc_crdt_in_fence	输入	如果 fence 位完成置位，则信用值不会合并，并且保证队列会先生成描述符提取，然后再处理后续信用值更新。仅限针对已启用的队列才能将 fence 位加以置位，并且该位中描述符和信用值均可用，否则可能发生挂起。
dsc_crdt_in_qid [10:0]	输入	与当前正在添加的信用值的描述符环关联的 QID。
dsc_crdt_in_crdt [15:0]	输入	用户应用当前正在给予 QDMA Subsystem for PCIe 的描述符信用值数量，用于从主机提取描述符。

QDMA 流量管理器信用值输出端口

表 72：**QDMA TM 信用值输出端口描述**

端口名称	I/O	描述
tm_dsc_sts_vld	输出	有效。表明输出总线上存在有效数据。总线上的有效数据处于保持状态，直至用户将 tm_dsc_sts_rdy 断言有效为止。

表 72：**QDMA TM 信用值输出端口描述** (续)

端口名称	I/O	描述
tm_dsc_sts_rdy	输入	就绪。Assertion（断言有效）表明用户逻辑已准备好在该总线上接受数据。不使用此接口时，Ready（就绪）必须绑定到 1。 **注释：**不使用此接口时，Ready（就绪）必须绑定到 1。
tm_dsc_sts_byp	输出	显示软件描述符上下文中的旁路位
tm_dsc_sts_dir	输出	表示状态更新对应的是 H2C 还是 C2H 描述符环。 0：H2C 1：C2H
tm_dsc_sts_mm	输出	指示状态更新对应的是串流队列还是存储器映射队列。 0：串流 1：存储器映射
tm_dsc_sts_qid [10:0]	输出	环的 QID
tm_dsc_sts_avl [15:0]	输出	如果 tm_dsc_sts_qinv 置位，那么这表示描述符引擎中可用的信用值数。如果 tm_dsc_sts_qinv 未置位，则表示自从上次发送此更新后转发至环的新描述符的数量。
tm_dsc_sts_qinv	输出	如果设置此端口，表明队列已失效。此项供用户应用用于协调用户应用与 QDMA 之间的信用值审计。
tm_dsc_sts_qen	输出	当前队列启用状态。
tm_dsc_sts_irq_arm	输出	如果设置此端口，表明驱动程序已准备好接受中断
tm_dsc_sts_error	输出	如果 PIDX 更新滚动至关联队列的当前 CIDX，则设为 1。
tm_dsc_sts_pidx[15:0]	输出	队列的 PIDX
tm_dsc_sts_port_id [2:0]	输出	与来自队列上下文的队列关联的端口 ID。

用户中断

表 73：**用户中断端口描述**

端口名称	I/O	描述
usr_irq_in_vld	输入	有效 断言有效表示应向 PCIe 生成中断，此中断与总线上的矢量、功能和暂挂字段相关联。Usr_irq_in_vld 断言有效后，必须保持高位，直至 DMA 将 usr_irq_out_ack 断言有效为止。
usr_irq_in_vec [11:0]	输入	矢量 要发送的 MSIX 矢量。 矢量从 0 开始到 7 为止。矢量 0 是第一个矢量。
usr_irq_in_fnc [7:0]	输入	功能 要发送的矢量的功能。
usr_irq_out_ack	输出	中断确认 确认位断言有效表示链路上已发射中断，用户逻辑必须等待此脉冲出现后，才能发出另一个中断条件信号。
usr_irq_out_fail	输出	中断失败 失败断言有效表示在链路上发射中断请求前，该请求已异常中止。

每个功能最多允许 8 个矢量。

队列状态端口

表 74：**队列状态端口**

端口名称	I/O	描述
qsts_out_op[7:0]	输出	即操作码，表示当前发出的包的类型。该字段的编码如下。 0x0：CMPT 标记响应 0x1：H2C-ST 标记响应 0x2：C2H-MM 标记响应 0x3：H2C-MM 标记响应 0x4-0xff：保留
qsts_out_data[63:0]	输出	各操作码的数据字段定义如下表所示。
qsts_out_port_id[2:0]	输出	端口 ID
qsts_out_qid[11:0]	输出	队列 ID
qsts_out_vld	输出	队列状态有效
qsts_out_rdy	输入	队列状态就绪。Ready 必须绑定到 1，以免阻塞状态输出。即使不使用此接口，ready 端口也必须绑定到 1。

表 75：**队列状态数据**

qsts_out_data	字段	描述
[1:0]	err	CMPT 引擎报告的错误代码。 0：无错误 1：软件提供的完成 CIDX 更新错误 2：处理 C2H 包时收到描述符错误 3：由于完成环已满，C2H 引擎已将完成丢弃
[2]	retry_marker_req	虽然已启用中断，但无法生成。如果接收到标记请求时，队列中的中断已处于未完成状态，那么就会发生这种情况。如确需发送中断，则用户逻辑必须等待并重试标记请求。
[25:3]	marker_cookie	当 CMPT 引擎向队列状态端口接口发送标记式，它会将 CMPT 的低 23 位包含在队列状态端口接口上的标记响应中进行发送。因此，发出标记请求时，用户逻辑可将 23 位值布局在 CMPT 中，它将随标记响应接收到相同的 23 位。如果由于 CMPT 引擎遇到的错误而生成标记（而非用户逻辑提交的标记请求），那么此 23 位字段无效。 **注释：**即使用户已在对主机执行 CMPT 写入中启用错误位和/或颜色位戳记，marker_cookie 也不会包含这些位。用户逻辑提交标记请求时包含的位就是它提供给 QDMA 的 CMPT 的低 23 位。
[26]	is_mrkr_rsp	如果标记响应基于标记请求，则此位将为置位到 1。如果此位设为“0”，那么标记响应将基于错误。
[63:27]	rsv	保留

寄存器空间

本章节旨在提供 QDMA Subsystem for PCIe 的寄存器空间信息。

在寄存器空间描述中，配置寄存器属性定义如下：

- 不适用：保留
- RO：“Read-Only”（只读）：表示寄存器位是只读位，不能被软件修改。
- RW：“Read-Write”（读写）：表示寄存器位是读写位，允许由软件将其设为期望的状态：Set（置位）或 Clear（清零）。
- RW1C：“Write-1-to-clear-status”（写入 1 将状态清零）：读取寄存器位时，寄存器位会指示其状态。Set 位表示状态事件，通过写入 1b 即可将其 Clear（清零）。将 0b 写入 RW1C 位是无效的。
- W1C：“Non-readable-write-1-to-clear-status”（不可读，写入 1 将状态清零）：读取寄存器时，它将返回 0。写入 1b 会为该位索引状态清零。将 0b 写入 W1C 位是无效的。
- W1S：“Non-readable-write-1-to-set”（不可读，写入 1 以置位）：读取寄存器时，它将返回 0。写入 1b 会将该位索引的控制设置置位。将 0b 写入 W1S 位是无效的。

QDMA PF 地址寄存器空间

在寄存器参考文件上提供的 qdma_v5_0_pf_registers.csv 中列出了所有物理功能 (PF) 寄存器。

提示：在默认模式下生成 IP 时，并不会公开所有寄存器。例如，调试寄存器将缺失。请参阅 qdma_v5_0_pf_registers.csv 文件以识别调试寄存器。要公开所有寄存器，请在 IP 生成期间使用以下 Tcl 命令：

```
set_property CONFIG.debug_mode {DEBUG_REG_ONLY} [get_ips qdma_0]
```

表 76：**QDMA PF 地址寄存器空间**

寄存器名称	基址（十六进制）	字节大小（十进制）	寄存器列表和详细信息
QDMA_CSR	0x0000	9216	qdma_v5_0_pf_registers.csv 中提供的 QDMA 配置空间寄存器 (CSR)。
QDMA_TRQ_SEL_QUEUE_PF	0x18000	32768	也可在 QDMA_TRQ_SEL_QUEUE_PF (0x18000) 中找到。
QDMA_PF_MAILBOX	0x22400	16384	也可在 QDMA_PF_MAILBOX (0x22400) 中找到。
QDMA_TRQ_MSIX	0x30000	32768	也可在 QDMA_TRQ_MSIX (0x30000) 中找到。
QDMA_TRQ_MSIX_PBA	0x34000	256	QDMA_TRQ_MSIX_PBA (0x34000)

QDMA_CSR (0x0000)

用户可通过寄存器参考文件中提供的 qdma_v5_0_pf_registers.csv 来获取 QDMA 配置空间寄存器 (CSR) 描述。

QDMA_TRQ_SEL_QUEUE_PF (0x18000)

表 77：**QDMA_TRQ_SEL_QUEUE_PF (0x18000) 寄存器空间**

寄存器	地址	描述
QDMA_DMAP_SEL_INT_CIDX[2048] (0x18000)	0x18000-0x1CFF0	中断环使用者索引 (CIDX)
QDMA_DMAP_SEL_H2C_DSC_PIDX[2048] (0x18004)	0x18004-0x1CFF4	H2C 描述符生产者索引 (PIDX)
QDMA_DMAP_SEL_C2H_DSC_PIDX[2048] (0x18008)	0x18008-0x1CFF8	C2H 描述符生产者索引 (PIDX)
QDMA_DMAP_SEL_CMPT_CIDX[2048] (0x1800C)	0x1800C-0x1CFFC	C2H 完成使用者索引 (CIDX)

有 2048 个队列，每个队列将有 4 个以上寄存器。所有这些寄存器均可随时动态更新。这组寄存器可根据队列编号来访问。

队列编号为绝对值 Qnumber [0 到 2047]。
中断 CIDX 地址 = 0x18000 + Qnumber*16
H2C PIDX 地址 = 0x18004 + Qnumber*16
C2H PIDX 地址 = 0x18008 + Qnumber*16
写回 CIDX 地址 = 0x1800C + Qnumber*16

对于队列 0：

0x18000 对应于 QDMA_DMAP_SEL_INT_CIDX
0c18004 对应于 QDMA_DMAP_SEL_H2C_DSC_PIDX
0x18008 对应于 QDMA_DMAP_SEL_C2H_DSC_PIDX
0x1800C 对应于 QDMA_DMAP_SEL_CMPT_CIDX

对于队列 1：

0x18010 对应于 QDMA_DMAP_SEL_INT_CIDX
0c18014 对应于 QDMA_DMAP_SEL_H2C_DSC_PIDX
0x18018 对应于 QDMA_DMAP_SEL_C2H_DSC_PIDX
0x1801C 对应于 QDMA_DMAP_SEL_CMPT_CIDX

对于队列 2：

0x18020 对应于 QDMA_DMAP_SEL_INT_CIDX
0c18024 对应于 QDMA_DMAP_SEL_H2C_DSC_PIDX
0x18028 对应于 QDMA_DMAP_SEL_C2H_DSC_PIDX
0x1802C 对应于 QDMA_DMAP_SEL_CMPT_CIDX

QDMA_DMAP_SEL_INT_CIDX[2048] (0x18000)

表 78：**QDMA_DMAP_SEL_INT_CIDX[2048] (0x18000)**

位	默认	访问类型	字段	描述
[31:24]	0	不适用	Reserved	保留
[23:16]	0	RW	ring_idx	中断聚合环的环索引
[15:0]	0	RW	sw_cdix	软件使用者索引 (CIDX)

QDMA_DMAP_SEL_H2C_DSC_PIDX[2048] (0x18004)

表 79：**QDMA_DMAP_SEL_H2C_DSC_PIDX[2048] (0x18004)**

位	默认	访问类型	字段	描述
[31:17]	0	不适用	Reserved	保留
[16]	0	RW	irq_arm	中断装备。对于下一次中断生成，请将该位设为 1。
[15:0]	0	RW	h2c_pidx	H2C 生产者索引

QDMA_DMAP_SEL_C2H_DSC_PIDX[2048] (0x18008)

表 80：**QDMA_DMAP_SEL_C2H_DSC_PIDX[2048] (0x18008)**

位	默认	访问类型	字段	描述
[31:17]	0	不适用	Reserved	保留
[16]	0	RW	irq_arm	中断装备。对于下一次中断生成，请将该位设为 1。
[15:0]	0	RW	c2h_pidx	C2H 生产者索引

QDMA_DMAP_SEL_CMPT_CIDX[2048] (0x1800C)

表 81：**QDMA_DMAP_SEL_CMPT_CIDX[2048] (0x1800C)**

位	默认	访问类型	字段	描述
[31:29]	0	不适用	Reserved	保留
[28]	0	RW	irq_en_wrb	中断装备。对于下一次中断生成，请将该位设为 1。
[27]	0	RW	en_sts_desc_wrb	为 CMPT 启用状态描述符
[26:24]	0	RW	trigger_mode	中断和状态描述符触发模式： 0x0：Disabled 0x1：Every 0x2：User_Count 0x3：User 0x4：User_Timer 0x5：User_Timer_Count
[23:20]	0	RW	c2h_timer_cnt_index	索引到 QDMA_C2H_TIMER_CNT
[19:16]	0	RW	c2h_count_threshhold	索引到 QDMA_C2H_CNT_TH
[15:0]	0	RW	wrb_cidx	CMPT 使用者索引 (CIDX)

QDMA_PF_MAILBOX (0x22400)

表 82：**QDMA_PF_MAILBOX (0x22400) 寄存器空间**

寄存器	地址	描述
功能状态寄存器 (0x22400)	0x22400	状态位
功能命令寄存器 (0x22404)	0x22404	命令寄存器位
功能中断矢量寄存器 (0x22408)	0x22408	中断矢量寄存器
目标功能寄存器 (0x2240C)	0x2240C	目标功能寄存器
功能中断矢量寄存器 (0x22410)	0x22410	中断控制寄存器
RTL 版本寄存器 (0x22414)	0x22414	RTL 版本寄存器
PF 确认寄存器 (0x22420-0x2243C)	0x22420-0x2243C	PF 确认
FLR 控制/状态寄存器 (0x22500)	0x22500	FLR 控制和状态
传入报文存储器 (0x22C00-0x22C7C)	0x22C00-0x22C7C	传入报文（128 字节）
传出报文存储器 (0x23000-0x2307C)	0x23000-0x2307C	传出报文（128 字节）

邮箱寻址

- PF 寻址：`Addr = PF_Bar_offset + CSR_addr`
- VF 寻址：`Addr = VF_Bar_offset + VF_Start_offset + VF_offset + CSR_addr`

功能状态寄存器 (0x22400)

表 83：**功能状态寄存器 (0x22400)**

位	默认	访问类型	字段	描述
[31:12]	0	不适用	Reserved	保留
[11:4]	0	RO	cur_src_fn	该字段仅适用于 PF 使用。 传入请求队列基础上的报文的源功能编号。
[2]	0	RO	ack_status	该字段仅适用于 PF 使用。 当确认状态寄存器中的任意位断言有效时，状态位将置位。
[1]	0	RO	o_msg_status	对于 VF：当 VF 驱动程序将 msg_send 写入其命令寄存器时，状态位将置位。当关联 PF 驱动程序向此 VF 发送确认时，硬件会将此字段清零。当 o_msg_status 断言有效时，不允许 VF 驱动程序更新其传出邮箱存储器 (OMM) 中的任意内容。对该 OMM 执行的任何违规写入都将被丢弃，（可选）这可能导致在 AXI4-Lite 响应通道中记录错误。 对于 PF：该字段表示目标 FN 的报文状态，目标 FN 在目标 FN 寄存器中指定。当 PF 驱动程序发送 msg_send 命令时，状态位即置位。当对应功能驱动程序通过发送 msg_rcv 来发送确认时，硬件会将此字段清零。当 o_msg_status(target_fn_id) 断言有效时，不允许 PF 驱动程序更新其传出邮箱存储器 (OMM) 中的任意内容。对该 OMM 执行的任何违规写入都将被丢弃（可选在 AXI4L 响应通道中记录错误）。
[0]	0	RO	i_msg_status	对于 VF：断言有效时，VF 的传入邮箱存储器中的报文将等待处理。在 VF 驱动程序将 msg_rcv 写入其命令寄存器后，该字段将清零。 对于 PF：断言有效时，传入邮箱存储器中的报文将暂挂，等待处理。仅当事件队列为空时，该字段才会清零。

功能命令寄存器 (0x22404)

表 84：**功能命令寄存器 (0x22404)**

位	默认	访问类型	字段	描述
[31:3]	0	不适用	Reserved	保留
[2]	0	RO	Reserved	保留
[1]	0	RW	msg_rcv	对于 VF：VF 会在其传入邮箱存储器中将此报文标记为已接收。硬件将关联 PF 的确认位断言有效。 对于 PF：PF 会将 target_fn 发送的报文标记为已接收。硬件会刷新 PF 的 i_msg_status，将 target_fn 的 o_msg_status 清零。

表 84：**功能命令寄存器 (0x22404)** (续)

位	默认	访问类型	字段	描述
[0]	0	RW	msg_send	对于 VF：VF 会将其自己的传出邮箱中的当前报文标记为有效。 对于 PF： · 当前 target_fn_id 属于 VF：PF 已完成将报文写入含 target_fn_id 的 VF 的传入邮箱存储器的操作。硬件会将目标 FN 的状态寄存器的 i_msg_status 字段置位。 · 当前 target_fn_id 属于 PF：PF 已完成将报文写入其自己的传出邮箱存储器的操作。硬件会将报文推送到带有 target_fn_id 的 PF 的事件队列中。

功能中断矢量寄存器 (0x22408)

表 85：**功能中断矢量寄存器 (0x22408)**

位	默认	访问类型	字段	描述
[31:5]	0	不适用	Reserved	保留
[4:0]	0	RW	int_vect	由驱动程序分配的 5 位中断矢量。

目标功能寄存器 (0x2240C)

表 86：**目标功能寄存器 (0x2240C)**

位	默认	访问类型	字段	描述
[31:8]	0	不适用	Reserved	保留
[7:0]	0	RW	target_fn_id	该字段仅适用于 PF 使用。 当前操作的目标 FN 编号。

功能中断矢量寄存器 (0x22410)

表 87：**功能中断矢量寄存器 (0x22410)**

位	默认	访问类型	字段	描述
[31:1]	0	不适用	Reserved	保留
[0]	0	RW	int_en	中断使能。

RTL 版本寄存器 (0x22414)

表 88：**RTL 版本寄存器 (0x22414)**

位	默认	访问类型	字段	描述
[31:16]	0x1fd3	RO		QDMA ID

表 88：**RTL 版本寄存器 (0x22414)** (续)

位	默认	访问类型	字段	描述
[15:0]	0	RO		Vivado 版本 0x0100：QDMA 3.0 Vivado v2019.1 0x0201：QDMA 3.1 Vivado v2019.2 补丁 0x0010：QDMA 4.0 Vivado v2020.1

PF 确认寄存器 (0x22420-0x2243C)

表 89：**PF 确认寄存器 (0x22420-0x2243C)**

寄存器	地址	默认	访问类型	字段	宽度	描述
Ack0	0x22420	0	RW		32	来自 FN 31~0 的确认
Ack1	0x22424	0	RW		32	来自 FN 63~32 的确认
Ack2	0x22428	0	RW		32	来自 FN 95~64 的确认
Ack3	0x2242C	0	RW		32	来自 FN 127~96 的确认
Ack4	0x22430	0	RW		32	来自 FN 159~128 的确认
Ack5	0x22434	0	RW		32	来自 FN 191~160 的确认
Ack6	0x22438	0	RW		32	来自 FN 223~192 的确认
Ack7	0x2243C	0	RW		32	来自 FN 255~224 的确认

FLR 控制/状态寄存器 (0x22500)

表 90：**FLR 控制/状态寄存器 (0x22500)**

位	默认	访问类型	字段	描述
[31:1]	0	不适用	Reserved	保留
[0]	0	RW	Flr_status	软件写入 1 即可为关联的功能发起功能级别复位 (FLR)。在 FLR 处理期间，该字段保持有效。FLR 完成后，硬件将此字段断言无效。

传入报文存储器 (0x22C00-0x22C7C)

表 91：**传入报文存储器 (0x22C00-0x22C7C)**

寄存器	地址	默认	访问类型	字段	宽度	描述
i_msg_i	0x22C00 + i*4	0	RW		32	传入报文的第 i 个词 (0≤ i < 128)。

传出报文存储器 (0x23000-0x2307C)

表 92：**传出报文存储器 (0x23000-0x2307C)**

寄存器	地址	默认	访问类型	字段	宽度	描述
o_msg_i	0x23000 + i *4	0	RW		32	传出报文的第 i 个词 (0 ≤ i < 128)。

QDMA_TRQ_MSIX (0x30000)

表 93：**QDMA_TRQ_MSIX (0x30000)**

字节偏移	位	默认	访问类型	字段	描述
0x30000	[31:0]	0	RW	addr	MSI-X vector0 报文下位地址。 MSIX_Vector0_Address[63:32]
0x30004	[31:0]	0	RW	addr	MSI-X vector0 报文上位地址。 MSIX_Vector0_Address[63:32]
0x30008	[31:0]	0	RW	data	MSIX_Vector0_Data[31:0] MSI-X vector0 报文数据。
0x3000C	[31:0]	0	RW	control	MSIX_Vector0_Control[31:0] MSI-X vector0 控制。 位元位置： 31:1：保留。 0：掩码。设为 1 时，此 MSI-X 矢量不用于生成报文。复位为 0 时，此 MSI-X 矢量用于生成报文。

注释：上表显示了一个 MSI-X 表条目 0。对于 QDMA，有 2000 个 MSI-X 表条目。

QDMA_TRQ_MSIX_PBA (0x34000)

MSIX 暂挂位阵列 (PBA) 偏移为 0x34000。每个 MSIX 矢量均含有 1 位 PBA。对于 2K 矢量，有 2K 位的 256 字节 PBA。

QDMA VF 地址寄存器空间

在寄存器参考文件的 `qdma_v5_0_vf_registers.csv` 中列出了所有虚拟功能 (VF) 寄存器。

表 94：**QDMA VF 地址寄存器空间**

目标名称	基址（十六进制）	字节大小（十进制）	注释
QDMA_TRQ_SEL_QUEUE_VF (0x3000)	00003000	4096	VF 直接 QCSR（每个队列 16B，最高每个功能 256 个队列）
QDMA_TRQ_MSIX_VF (0x4000)	00004000	4096	可容纳 32 个 MSIX 矢量和 PBA 的空间
QDMA_TRQ_MSIX_VF_PBA (0x4800)	00004800	32	MSIX PBA
QDMA_VF_MAILBOX (0x5000)	00005000	8192	邮箱地址空间

QDMA_TRQ_SEL_QUEUE_VF (0x3000)

VF 函数可按队列访问直接更新寄存器，偏移量为 (0x3000)。此寄存器空间的描述与 QDMA_TRQ_SEL_QUEUE_PF (0x18000) 相同。

这组寄存器可基于队列编号来访问。队列编号是该 VF 的相对 Qnumber 值。

中断 CIDX 地址 = 0x3000 + Qnumber*16
H2C PIDX 地址 = 0x3004 + Qnumber*16
C2H PIDX 地址 = 0x3008 + Qnumber*16
完成 CIDX 地址 = 0x300C + Qnumber*16

对于队列 0：

0x3000 对应于 QDMA_DMAP_SEL_INT_CIDX
0x3004 对应于 QDMA_DMAP_SEL_H2C_DSC_PIDX
0x3008 对应于 QDMA_DMAP_SEL_C2H_DSC_PIDX
0x300C 对应于 QDMA_DMAP_SEL_WRB_CIDX

对于队列 1：

0x3010 对应于 QDMA_DMAP_SEL_INT_CIDX
0x3014 对应于 QDMA_DMAP_SEL_H2C_DSC_PIDX
0x3018 对应于 QDMA_DMAP_SEL_C2H_DSC_PIDX
0x301C 对应于 QDMA_DMAP_SEL_WRB_CIDX

QDMA_TRQ_MSIX_VF (0x4000)

VF 功能可按距离该功能的偏移 (0x0000) 来访问 MSIX 表。此寄存器空间的描述与 QDMA_TRQ_MSIX (0x30000) 相同。

QDMA_VF_MAILBOX (0x5000)

表 95：**QDMA_VF_MAILBOX (0x05000) 寄存器空间**

寄存器（地址）	地址	描述
功能状态寄存器 (0x5000)	0x5000	状态寄存器位
功能命令寄存器 (0x5004)	0x5004	命令寄存器位
功能中断矢量寄存器 (0x5008)	0x5008	中断矢量寄存器
目标功能寄存器 (0x500C)	0x500C	目标功能寄存器
功能中断控制寄存器 (0x5010)	0x5010	中断控制寄存器
RTL 版本寄存器 (0x5014)	0x5014	RTL 版本寄存器
传入报文存储器 (0x5800-0x587C)	0x5800-0x587C	传入报文（128 字节）
传出报文存储器 (0x5C00-0x5C7C)	0x5C00-0x5C7C	传出报文（128 字节）

功能状态寄存器 (0x5000)

表 96：**功能状态寄存器 (0x5000)**

位索引	默认	访问类型	字段	描述
[31:12]	0	不适用	Reserved	保留
[11:4]	0	RO	cur_src_fn	该字段仅适用于 PF 使用。 传入请求队列基础上的报文的源功能编号。
[2]	0	RO	ack_status	该字段仅适用于 PF 使用。 当确认状态寄存器中的任何位被断言有效时，状态位将置位。

表 96：**功能状态寄存器 (0x5000)** (续)

位索引	默认	访问类型	字段	描述
[1]	0	RO	o_msg_status	对于 VF：当 VF 驱动程序将 msg_send 写入其命令寄存器时，状态位将置位。当关联 PF 驱动程序向此 VF 发送确认时，硬件会将此字段清零。当 o_msg_status 断言有效时，不允许 VF 驱动程序更新其传出邮箱存储器 (OMM) 中的任意内容。对该 OMM 执行的任何违规写入都将被丢弃（可选在 AXI4-Lite 响应通道中记录错误）。 对于 PF：该字段指示目标 FN 的报文状态，目标 FN 在“Target FN Register”（目标 FN 寄存器）中指定。当 PF 驱动程序发送 msg_send 命令时，状态位即置位。当对应功能驱动程序通过 msg_rcv 发送确认时，硬件会将此字段清零。当 o_msg_status(target_fn_id) 断言有效时，不允许 PF 驱动程序更新其传出邮箱存储器 (OMM) 中的任意内容。对该 OMM 执行的任何违规写入都将被丢弃（可选在 AXI4L 响应通道中记录错误）。
[0]	0	RO	i_msg_status	对于 VF：断言有效时，VF 的传入邮箱存储器中的报文将等待处理。在 VF 驱动程序将 msg_rcv 写入其命令寄存器后，该字段将清零。 对于 PF：断言有效时，传入邮箱存储器中的报文将暂挂，等待处理。仅当事件队列为空时，该字段才会清零。

功能命令寄存器 (0x5004)

表 97：**功能命令寄存器 (0x5004)**

位索引	默认	访问类型	字段	描述
[31:3]	0	不适用	Reserved	保留
[2]	0	RO	Reserved	保留
[1]	0	RW	msg_rcv	对于 VF：VF 会在其传入邮箱存储器中将此报文标记为已接收。硬件将关联 PF 的确认位断言有效。 对于 PF：PF 会将 target_fn 发送的报文标记为已接收。硬件会刷新 PF 的 i_msg_status，将 target_fn 的 o_msg_status 清零。
[0]	0	RW	msg_send	对于 VF：VF 会将其自己的传出邮箱中的当前报文标记为有效。 对于 PF： 当前 target_fn_id 属于 VF：PF 已完成将报文写入含 target_fn_id 的 VF 的传入邮箱存储器的操作。硬件会将目标 FN 的状态寄存器的 i_msg_status 字段置位。 当前 target_fn_id 属于 PF：PF 已完成将报文写入其自己的传出邮箱存储器的操作。硬件会将报文推送到带有 target_fn_id 的 PF 的事件队列中。

功能中断矢量寄存器 (0x5008)

表 98：**功能中断矢量寄存器 (0x5008)**

位索引	默认	访问类型	字段	描述
[31:5]	0	不适用	Reserved	保留
[4:0]	0	RW	int_vect	由驱动程序软件分配的 5 位中断矢量。

目标功能寄存器 (0x500C)

表 99：**目标功能寄存器 (0x500C)**

位索引	默认	访问类型	字段	描述
[31:8]	0	不适用	Reserved	保留
[7:0]	0	RW	target_fn_id	该字段仅适用于 PF 使用。 当前操作的目标 FN 编号。

功能中断控制寄存器 (0x5010)

表 100：**功能中断控制寄存器 (0x5010)**

位索引	默认	访问类型	字段	描述
[31:1]	0	不适用	res	保留
[0]	0	RW	int_en	中断使能。

RTL 版本寄存器 (0x5014)

表 101：**RTL 版本寄存器 (0x5014)**

位	默认	访问类型	字段	描述
[31:16]	0x1fd3	RO	.	QDMA ID
[15:0]	0	RO	.	Vivado 版本 0x0100：QDMA 3.0 Vivado v2019.1 0x0201：QDMA 3.1 Vivado v2019.2 补丁 0x0010：QDMA 4.0 Vivado v2020.1

传入报文存储器 (0x5800-0x587C)

表 102：**传入报文存储器 (0x5800-0x587C)**

寄存器	地址	默认	访问类型	字段	宽度	描述
i_msg_i	0x5800 + i*4	0	RW		32	传入报文的第 i 个词 (i < 128)。

传出报文存储器 (0x5C00-0x5C7C)

表 103：**传出报文存储器 (0x5C00-0x5C7C)**

寄存器	地址	默认	访问类型	字段	宽度	描述
o_msg_i	0x5C00 + i *4	0	RW		32	传出报文的第 i 个词 (i < 128)。

AXI4-Lite CSR 从接口寄存器空间

Bridge 寄存器空间和 DMA 寄存器空间可通过 AXI4-Lite CSR 从接口来访问。仅当 `csr_prog_done` 端口设为 1 时，此接口才可访问。必须等待 `csr_prog_done` 端口置位后才能访问。

表 104：**AXI4-Lite CSR 从接口寄存器空间**

寄存器空间	AXI4-Lite CSR 从接口	详细信息
Bridge 寄存器	AXI4-Lite CSR 从接口地址 **bit [15]** 设为 0	可在寄存器参考文件的 `qdma_v5_0_bridge_registers.csv` 中找到。
DMA 寄存器	AXI4-Lite CSR 从接口地址 **bit [15]** 设为 1	欲知详情，请参阅 QDMA PF 地址寄存器空间 和 QDMA VF 地址寄存器空间。 **注释：**此接口仅限用于访问 DMA CSR 寄存器。DMA 队列空间寄存器只能通过 AXI4-Lite 从接口访问。

Bridge 寄存器空间

Bridge 寄存器地址从 0xE00 开始。从 0x00 到 0xE00 的地址均定向至 PCIe 核配置寄存器空间。

如需获取 QDMA Bridge 寄存器描述，请参阅寄存器参考文件中提供的 `qdma_v5_0_bridge_registers.csv`。

DMA 寄存器空间

以下章节旨在描述 DMA 寄存器空间：

- QDMA PF 地址寄存器空间
- QDMA VF 地址寄存器空间

AXI4-Lite 从接口寄存器空间

DMA 队列空间寄存器可通过 AXI4-Lite 从接口来访问。

如需了解有关 QDMA 队列空间 PF 寄存器地址和 QDMA 队列空间 VF 寄存器地址的描述，请参阅 QDMA_TRQ_SEL_QUEUE_PF (0x18000) 和 QDMA_TRQ_SEL_QUEUE_VF (0x3000)。

注释：通过此接口只能访问 DMA 队列空间寄存器。仅限通过 AXI4-Lite CSR 从接口访问 DMA CSR 寄存器。

AMD XILINX

第 4 章

利用子系统进行设计

通用设计指南

使用设计示例

由 Vivado® 设计工具所创建的子系统的每个实例交付时均包含设计示例，此设计示例可先在器件中实现，随后再对其进行仿真。您可将此设计作为起点来开展自己的设计，或者也可在遇到困难时将其用于对您的应用进行完整性检查。请参阅“设计示例”内容，以获取有关将设计示例用于子系统以及设计示例自定义的信息。

寄存信号

为了在可编程器件设计中简化时序并提升系统性能，请使用户应用与子系统之间的所有输入和输出保持处于已寄存状态。这意味着来自用户应用的所有输入和输出都应来自于或者连接至触发器。虽然可能并非所有路径都能寄存信号，但这样可简化时序分析，便于赛灵思工具对设计进行布局布线。

识别时序关键信号

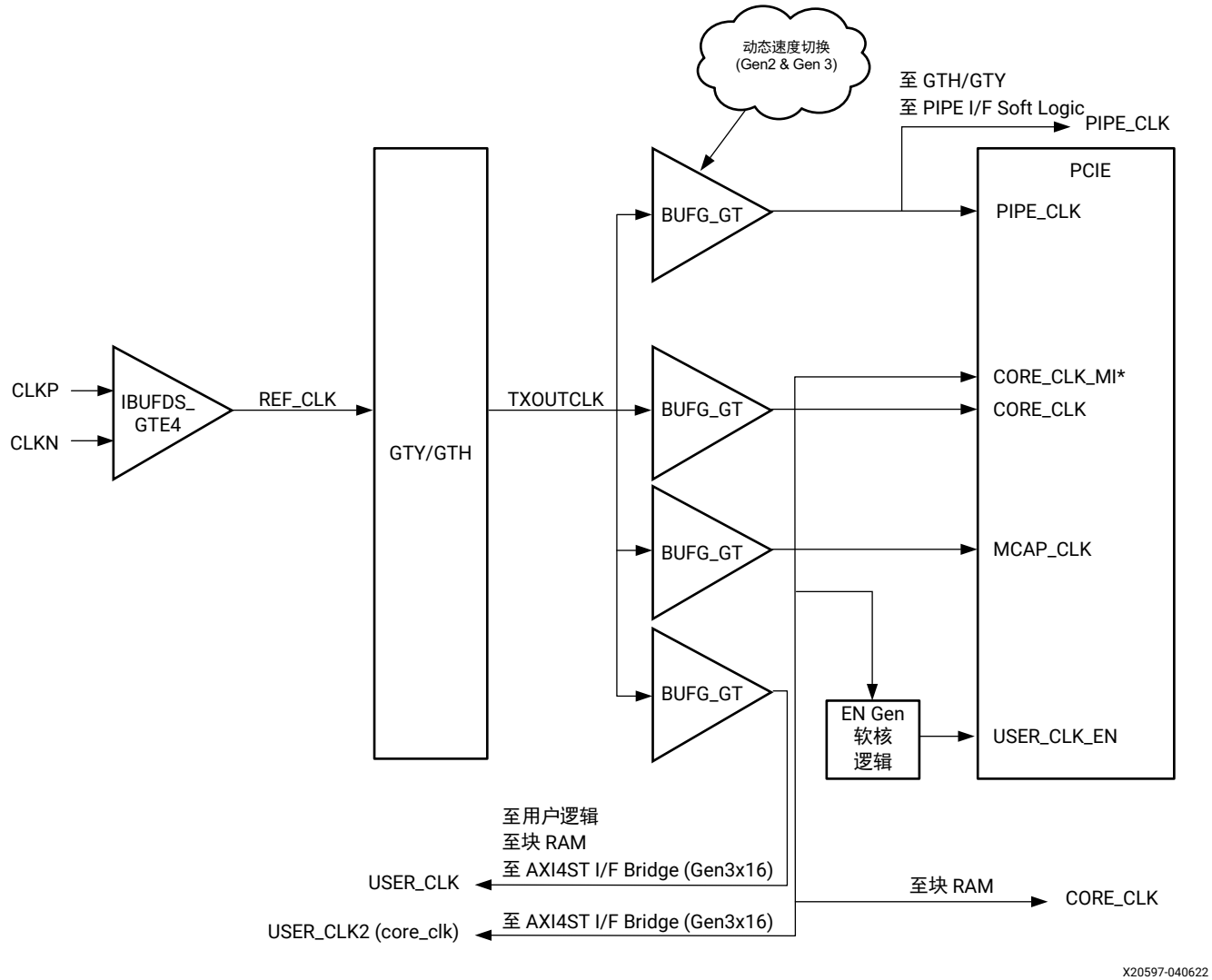
随设计示例所提供的约束可识别应应用的关键信号和时序约束。

仅限执行允许的修改

不得修改子系统。任何修改都可能对系统时序和协议合规性造成负面影响。只能通过在生成子系统时，选择“Customization IP”对话框中的选项来执行受支持的子系统用户配置。

时钟设置

图 26：**时钟设置**



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。此 IP 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。QDMA Subsystem for PCIe 和用户逻辑主要用于 `user_clk`。

串联配置

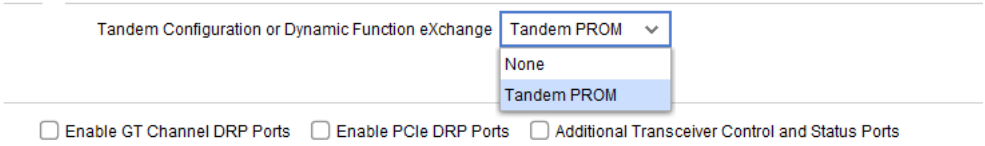
串联配置可供 UltraScale+™ 器件的 QDMA Subsystem for PCI Express 使用。对于其它赛灵思 IP（包括 UltraScale+ 器件 Integrated Block for PCI Express 和 DMA/Bridge Subsystem for PCI Express），均可使用双阶段方法论来快速配置此类 IP 以满足 PCI Express 规范中的配置时间要求。如需了解有关受支持的器件的更多信息，请参阅《UltraScale+ Integrated Block for PCI Express LogiCORE IP 产品指南》(PG213)。

为串联 PROM 配置核

要为 UltraScale+ 器件启用串联 PROM 功能，请在自定义该核时选择相应的 IP 目录选项。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”或“Dynamic Function eXchange”选项：
   - Tandem PROM：从闪存加载单一 2 阶比特流，目标是满足 100ms PCIe 枚举要求。

图 27：**Tandem PROM**



支持的器件

QDMA Subsystem for PCIe 支持的器件如下：

- 仅限 Zynq UltraScale+ RFSoC 和 KU19P 器件才支持串联配置（串联 PROM）。ZU39DR 器件需 X0Y1 PCIe 块位置，对于所有其它器件，您均可选择 X0Y0 PCIe 块位置。

如需了解有关其它 IP 和器件中可用的串联配置的更多信息，请参阅《UltraScale+ Integrated Block for PCI Express LogiCORE IP 产品指南》(PG213) 和《DMA/Bridge Subsystem for PCI Express 产品指南》(PG195)。

基于 PCIe 的 Dynamic Function eXchange

QDMA Subsystem for PCI Express 允许您启用 MCAP 以便为 Dynamic Function eXchange (DFX) 解决方案交付部分比特流。该功能特性使用器件的标准初始配置（而非串联配置），随后通过 PCIe 链路交付部分比特流，以便对可编程逻辑的各部分进行动态更新。Vivado DFX 设计流程用于创建和管理这些部分比特流，PCIe 端点保留在设计的静态部分中。

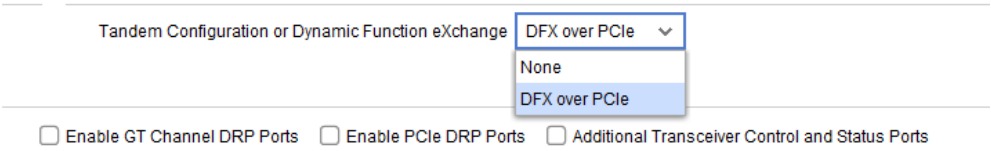
为 Dynamic Function eXchange 配置该核

要为 UltraScale+ 器件启用 DFX 功能，请在自定义该核时选择相应的 IP 目录选项。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：
   - DFX over PCIe（基于 PCIe 的 DFX）：该选项用于为 Dynamic Function eXchange 启用 MCAP 链路，而无需启用 Tandem Configuration。

图 28：**Dynamic Function eXchange 选项**



支持的器件

QDMA Subsystem for PCIe 支持的器件如下：

- 所有 Artix® UltraScale+、Kintex UltraScale+（KU19P 除外）以及 Virtex UltraScale+ 器件都支持“DFX over PCIe”，并且也支持含 PCIe 硬核块的 Zynq UltraScale+ MPSoC。

要了解可用于 DFX over PCIe 的 PCIe 块的位置，请参阅《UltraScale+ Integrated Block for PCI Express LogiCORE IP 产品指南》(PG213) 中的“串联配置”。如需了解有关 Dynamic Function eXchange 的更多信息，请参阅《Vivado Design Suite 用户指南：Dynamic Function eXchange》(UG909)。

AMD XILINX

第 5 章

设计流程步骤

本章节描述了子系统的自定义和生成方式、子系统的约束方式以及此 IP 子系统的仿真、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)

自定义和生成子系统

本节包含有关如何使用赛灵思工具在 Vivado® Design Suite 中自定义和生成子系统的信息。

如果要在 Vivado IP integrator 中自定义和生成子系统，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 子系统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。

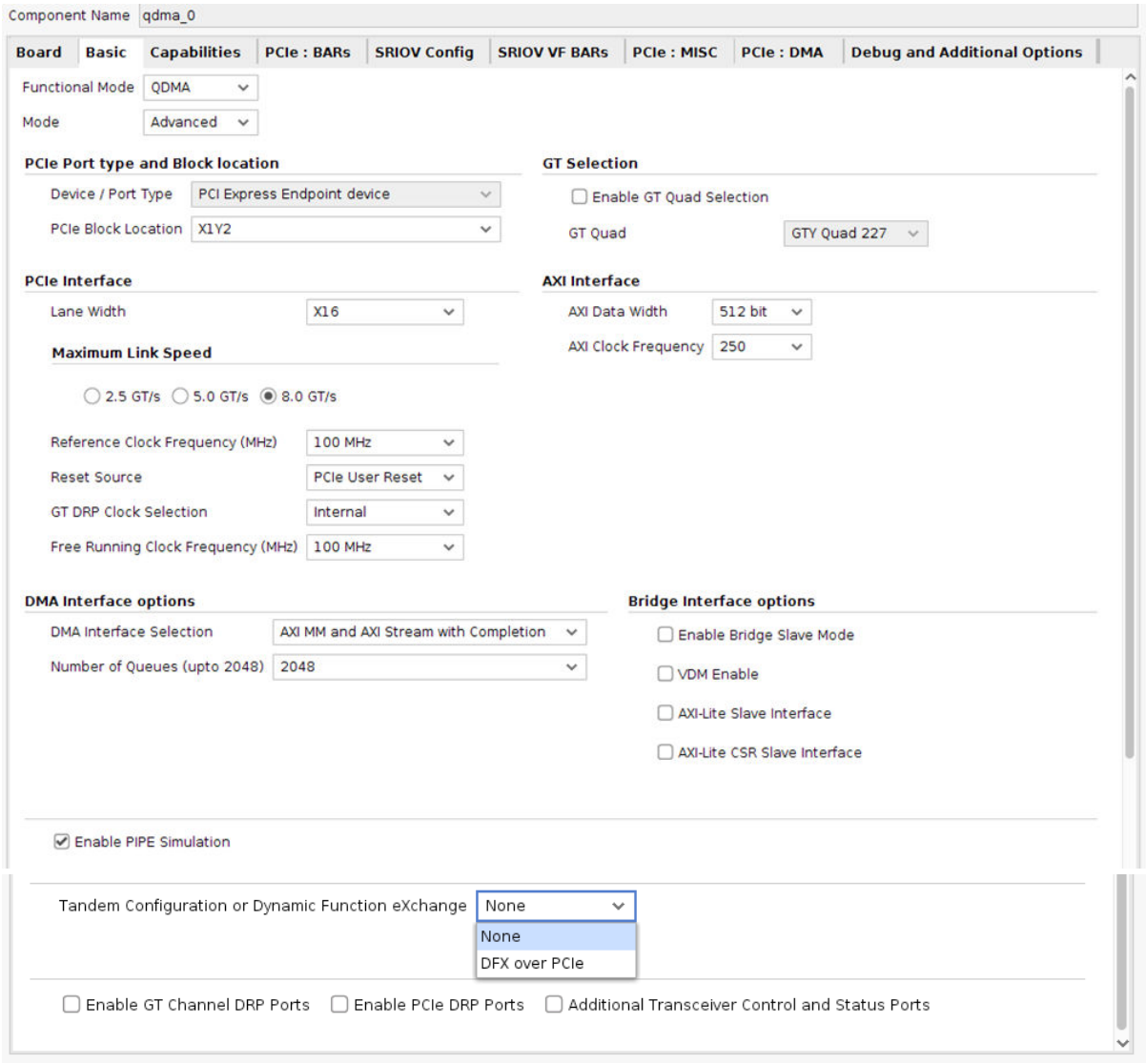
欲知详情，请参阅《Vivado Design Suite 用户指南：采用 IP 进行设计》(UG896) 和《Vivado Design Suite 用户指南：入门指南》(UG910)。

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。

“Basic”选项卡

“Basic”（基本）选项卡如下图所示。

图 29： **“Basic”选项卡**



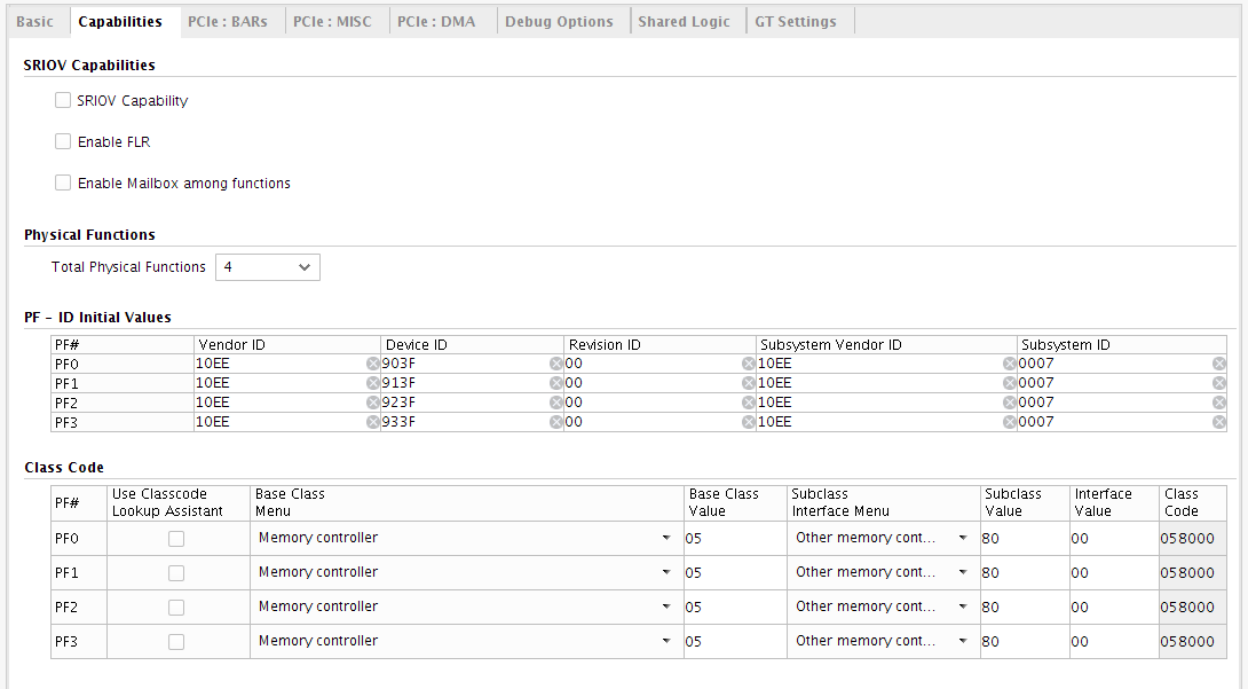
- “Functional Mode”（功能模式）：可选选项包括 QDMA 和 AXI Bridge。
- “Mode”（模式）：允许您为核的配置选择“Basic”（基本）模式或“Advanced”（高级）模式。
- “Device /Port Type”（器件/端口类型）：仅支持“PCI Express® Endpoint device”模式。
- “GT Selection/Enable GT Quad Selection”（GT 选择/启用 GT 四通道选择）：用于选择通道 0 所在的四通道。
- “PCIe Block Location”（PCIe 块位置）：用于从可用集成块中进行选择，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。如果选择赛灵思开发板，则该选项不可用。
- “Lane Width”（通道宽度）：核需要选择初始通道宽度。《UltraScale+ Integrated Block for PCI Express LogiCORE IP 产品指南》(PG213) 定义了可用的宽度和关联的生成核。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。选项为 4、8 或 16 通道。

- “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。《UltraScale+ Integrated Block for PCI Express LogiCORE IP 产品指南》(PG213) 定义了器件支持的通道宽度和链路速度。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。默认选项是 Gen3。
- “Reference Clock Frequency”（参考时钟频率）：默认值为 100 MHz。
- “Reset Source”（复位源）：可选下列任一选项：
  - “PCIe User Reset”（PCIe 用户复位）：用户复位来自链路建立后的 PCIe 核。当 PCIe 链路中断时，用户复位会被断言有效，核进入复位模式。当链路恢复时，用户复位将断言无效。
  - “Phy Ready”（物理就绪）：选中该选项后，核不受 PCIe 链路状态的影响。
- “AXI Data Width”（AXI 数据宽度）：选择 128、256 位或 512 位（仅适用于 UltraScale+）。该核允许您选择“Interface Width”（接口宽度），如需了解其定义，请参阅《UltraScale+ Integrated Block for PCI Express LogiCORE IP 产品指南》(PG213)。“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 Clock Frequency”（AXI 时钟频率）：值为 250 MHz，取决于通道宽度/速度。
- “DMA Interface Option”（DMA 接口选项）：可选下列任一选项：
  - AXI Memory Mapped and AXI Stream with Completion
  - AXI Memory Mapped only
  - AXI Stream with Completion
- “Number of Queues (up to 2048)”（队列数（最大数量为 2048））：选择最大队列数。选项为 512（默认值）、1024 和 2048。
- “Enable Bridge Slave Mode”（启用 Bridge 从模式）：选择此项以启用 AXI-MM 从接口。
- “VDM Enable”（VDM 启用）：选择此项以启用供应商定义报文。
- “AXI Lite Slave Interface”（AXI4-Lite 从接口）：选择此项以启用 AXI4-Lite 从接口，此接口可访问 DMA 队列空间。
- “AXI Lite CSR Slave Interface”（AXI4-Lite CSR 从接口）：选择此项以启用 AXI4-Lite CSR 从接口，此接口可访问 DMA 配置空间寄存器或 Bridge 寄存器。
- “Enable PIPE Simulation”（启用 PIPE 仿真）：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 PCI Express 块采用 Gen3 x8 和 Gen2 x8 配置进行 PIPE 模式仿真》(XAPP1184)。请参阅其中设计，了解如何将 UltraScale™ 器件核的外部 PIPE 接口端口连接到第三方 BFM。启用管道仿真以加速仿真。仅用于仿真。
- “Tandem Configuration or Dynamic Function eXchange”（串联配置或 Dynamic Function eXchange）：您可选择 Dynamic Function eXchange (DFX) over PCIe，DFX 使用 MCAP 接口。QDMA Subsystem for PCIe 不支持串联配置模式。
- “Enable GT Channel DRP Ports”（启用 GT 通道 DRP 端口）：选择该选项以启用 GT 专用的 DRP 端口。
- “Enable PCIe DRP Ports”（启用 PCIe DRP 端口）：选择该选项以启用 PCIe 专用的 DRP 端口。
- “Additional Transceiver Control and Status Ports”（其它收发器控制和状态端口）：选择此项以启用任何附加端口。

“Capabilities”选项卡

“Capabilities”（功能）选项卡如下图所示。

图 30：**“Capabilities”选项卡**



- “SRIOV Capability”（SRIOV 功能）：启用单根端口 I/O 虚拟化 (SR-IOV) 功能。该集成块用于实现扩展的 SR-IOV PCIe。启用该选项时，即可在所有选定物理功能上实现 SR-IOV。启用 SR-IOV 功能时，仅支持 MSI-X 中断。
  - “Enable Mailbox among functions”（在各功能之间启用邮箱）：这是一个邮箱系统，用于在不同功能之间进行通信。启用上方的“SR-IOV capability”时，默认启用该选项。邮箱的选择与“SR-IOV Capability”的选择无关。
  - “Enable FLR”（启用 FLR）：表示启用功能级别复位端口。启用上方的“SR-IOV capability”时，默认启用该选项。
- “Physical Functions”（物理功能）：默认可启用最多 4 个物理功能。
- “PF - ID Initial Values”（PF - 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 分配以保证每个标识都唯一。赛灵思的供应商 ID 默认值为 `10EEh`。请在此处输入供应商标识号。`FFFFh` 为保留编号。
  - “Device ID”（器件 ID）：表示应用的唯一标识；值取决于所选配置，默认值为 70h。该字段可包含任何值；请根据应用对该值进行更改。

    Device ID 参数根据下列条件进行评估：

    - 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

- 链接速度

例如，Device ID 为 B03F 表示 Versal B 的器件 ID，3 表示 Gen3，F 表示 X16（宽度）。

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。

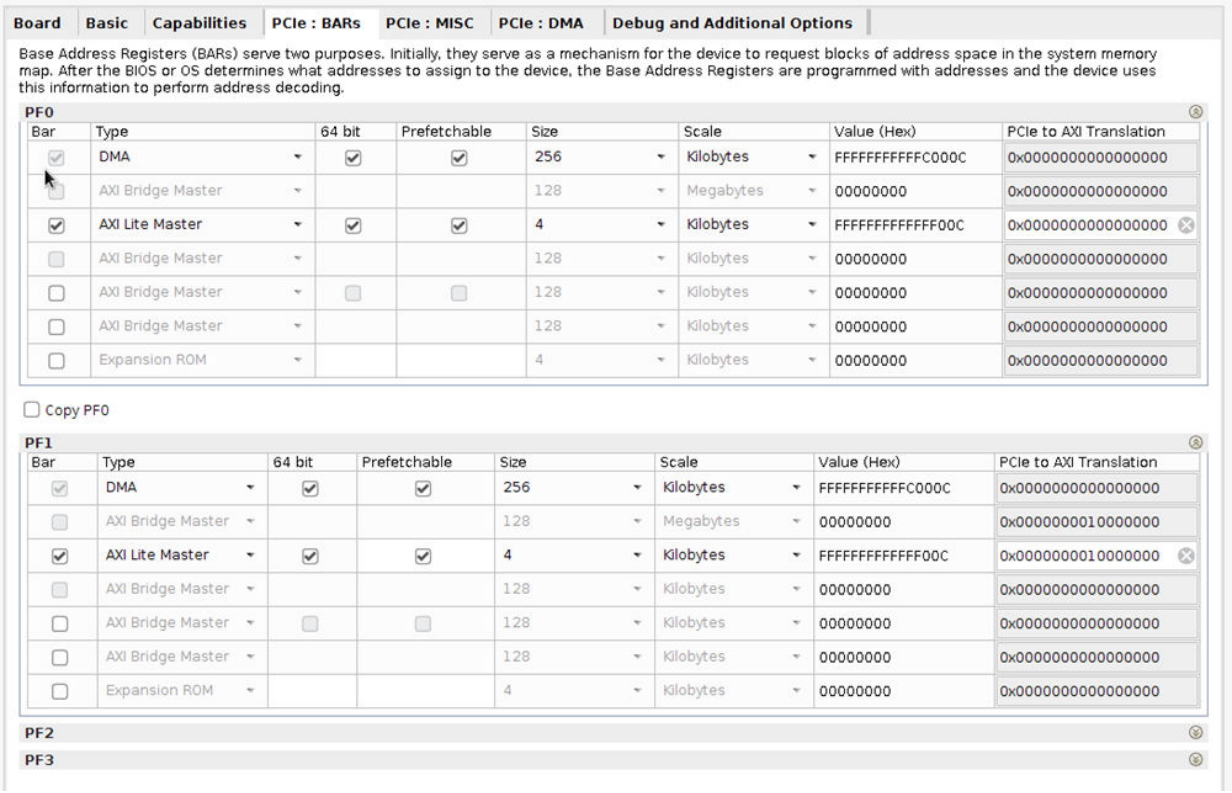
- “Revision ID”（版本 ID）：表示器件或应用的版本；作为器件 ID 的扩展。默认值为 00h；请针对应用输入相应的值。
- “Subsystem Vendor ID”（子系统供应商 ID）：用于进一步限定器件或应用的制造商。请在此处输入子系统供应商 ID；默认值为 10EEh。通常，该值与供应商 ID 相同。将该值设为 0000h 可能导致合规性测试出现问题。
- “Subsystem ID”（子系统 ID）：用于进一步限定器件或应用的制造商。该值通常与器件 ID 相同；默认值取决于所选通道宽度和链路速度。将该值设为 0000h 可能导致合规性测试出现问题。

- “Class Code”（类代码）：类代码用于识别器件的常规功能。
  - “Use Classcode Lookup Assistant”（使用类代码查找助手）：如果选中此项，那么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“Class Code”（类代码）中输入这 3 个值，才能将这些值转换为器件设置。
  - “Base Class”（基本类）：用于广泛识别器件执行的功能类型。
  - “Subclass”（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。

“PCIe BARs”选项卡

“PCIe BARs”（PCIe BAR）选项卡如下图所示。

图 31：**“PCIe BARs”选项卡**



- 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。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 DMA、AXI4-Lite 主接口或 AXI Bridge 主接口。
  - 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)。用于 DMA、AXI4-Lite 主接口或 AXI Bridge 主接口。

所有 BAR 寄存器都共享这些选项。

重要提示！DMA 需要大量空间来支持各种功能和队列。默认情况下，对于 DMA BAR，选择 64 位 BAR 空间。这适用于 PF BAR 和 VF BAR。在选择 64 位 BAR 空间还是 32 位 BAR 空间之前，您必须先计算自己的设计需求。

BAR 的选择是可配置的。默认情况下，DMA 位于 BAR 0（64 位），AXI-Lite 主接口则位于 BAR 2（64 位）。这些选择可根据用户需求而变。

- BAR：单击该复选框即可启用 BAR。反选该复选框即可禁用 BAR。
- 类型：可选：“DMA”（BAR0 中的默认选项）、“AXI Lite Master”（BAR1 中的默认选项，前提是启用该选项）或“AXI Bridge Master”（BAR2 中的默认选项，前提是启用该选项）。对于所有其它 BAR，可选项为“AXI Lite Master”和“AXI Bridge Master”。选择 BAR6 即可启用扩展 ROM

  对于 64 位 BAR（默认选项），可选：“**DMA**”（BAR0 中断默认选项）、“**AXI Lite Master**”（BAR2 中的默认选项，前提是启用 BAR2）和“**AXI Bridge Master**”（BAR4 中的默认选项，前提是启用 BAR4）。选择 BAR6 即可启用扩展 ROM。

  - DMA：DMA 默认分配给 BAR0 空间和所有 PF。在任意可用 BAR 中均可选择 DMA 选项（仅限一个 BAR 选择 DMA 选项）。前提是选中“DMA Mailbox Management”（DMA 邮箱管理）而不是 DMA；但“DMA Mailbox Management”将不允许您执行任何 DMA 操作。选中“DMA Mailbox Management”选项后，主机即可访问扩展邮箱空间。如需了解有关此空间的详细信息，请参阅 QDMA_PF_MAILBOX (0x22400) 寄存器空间。
  - AXI Lite Master：对任意 BAR 空间均可选择 AXI Lite Master 接口选项。大小、标度和地址转换均可配置。
  - AXI Bridge Master：对任意 BAR 空间均可选中 AXI Bridge Master 接口选项。大小、标度和地址转换均可配置。
  - “Expansion ROM”（扩展 ROM）：启用该选项后，即可在 AXI4-Lite Master 上访问此空间。这是一个只读空间。大小、标度和地址转换均可配置。
- “Size”（大小）：可用“Size”范围取决于选择的是 32 位 BAR 还是 64 位 BAR。DMA 需要 256 KB 的空间，这是固定的默认选项。其它 BAR 大小选项也可用，但必须指定。
- “Scale”（标度）：请选择字节、千字节或兆字节。
- “Value”（值）：表示基于当前选择分配给 BAR 的值。

注释：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。

相关信息

QDMA_PF_MAILBOX (0x22400)

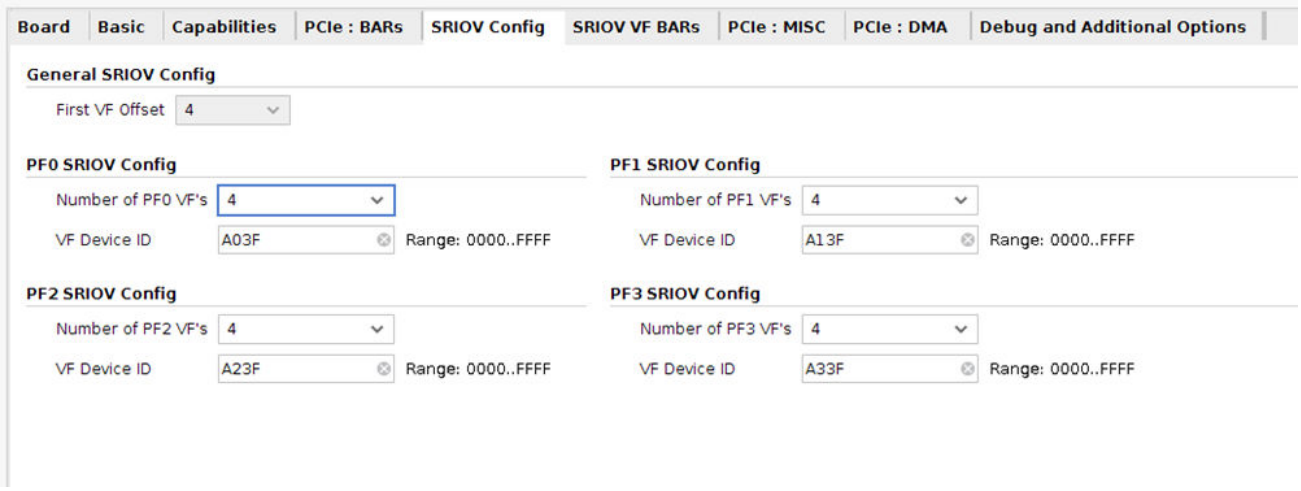
“SRIOV Config”选项卡

“SRIOV Config”（SRIOV 配置）选项卡允许您为物理功能 (PF) 指定 SR-IOV 功能。该信息用于构造 SR-IOV 功能结构。上电时不存在虚拟功能。由系统软件中的功能负责发现和启用基于系统功能的 VF。发现 VF 支持的方式是扫描每个 PF 的 SR-IOV 功能结构。

注释：在 “Capabilities”选项卡 中选中“SRIOV Capability”（SRIOV 功能）时，就会显示“SRIOV Config”（SRIOV 配置）选项卡。

“SRIOV Config”选项卡如下图所示。

图 32：**“SRIOV Config”选项卡**

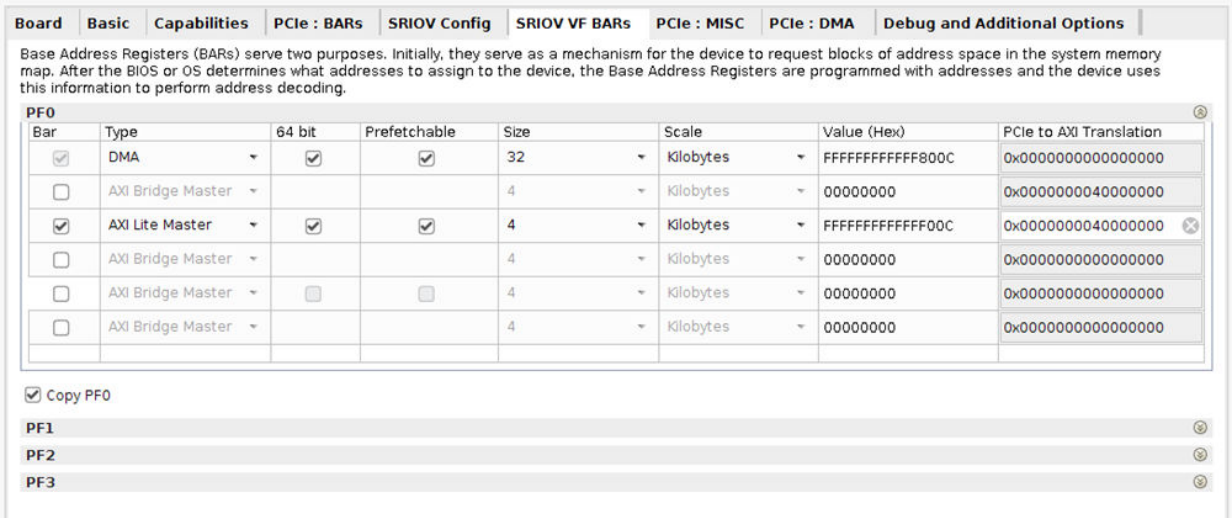


- “General SRIOV Config”（通用 SRIOV 配置）：该值用于指定第 1 项 PF 的偏移，其中含至少 1 项已启用的 VF。启用 ARI 时，允许的值为 'd4 或 'd64，所有 PF 中的 VF 总数加上该字段的值不得超过 256。禁用 ARI 时，该字段将设置为 1，这样仅支持 1PF 加上 7VF 的非 ARI SRIOV 配置。
- “Number of PFx VFs”（PFx VF 数量）：表示与物理功能关联的虚拟功能的数量。共有 252 项虚拟功能可供在 4 项物理功能之间灵活使用。
- “VF Device ID”（VF 器件 ID）：表示与物理功能关联的所有虚拟功能的 16 位器件 ID。

“SRIOV VF BARs”选项卡

“SRIOV VF BARs”选项卡如下图所示。

图 33：**“SRIOV VF BARs”选项卡**



“SRIOV VF BARs”选项卡支持您为单一虚拟功能组 (VFG) 内的所有虚拟功能 (VF) 配置基址寄存器 (BAR)。同一 VFG 内的所有 VF 都共享相同的基址寄存器 (BAR) 配置。每个虚拟功能支持最多 6 个 32 位 BAR 或 3 个 64 位 BAR。虚拟功能 BAR 的配置不依赖于关联物理功能 BAR 的设置。

重要提示！DMA 需要大量空间来支持各种功能和队列。默认情况下，对于 DMA BAR，选择 64 位 BAR 空间。这适用于 PF BAR 和 VF BAR。在选择 64 位 BAR 空间还是 32 位 BAR 空间之前，您必须先计算自己的设计需求。

BAR 的选择是可配置的。默认情况下，DMA 位于 BAR 0（64 位），AXI4-Lite 主接口则位于 BAR 2（64 位）。这些选择可根据用户需求而变。

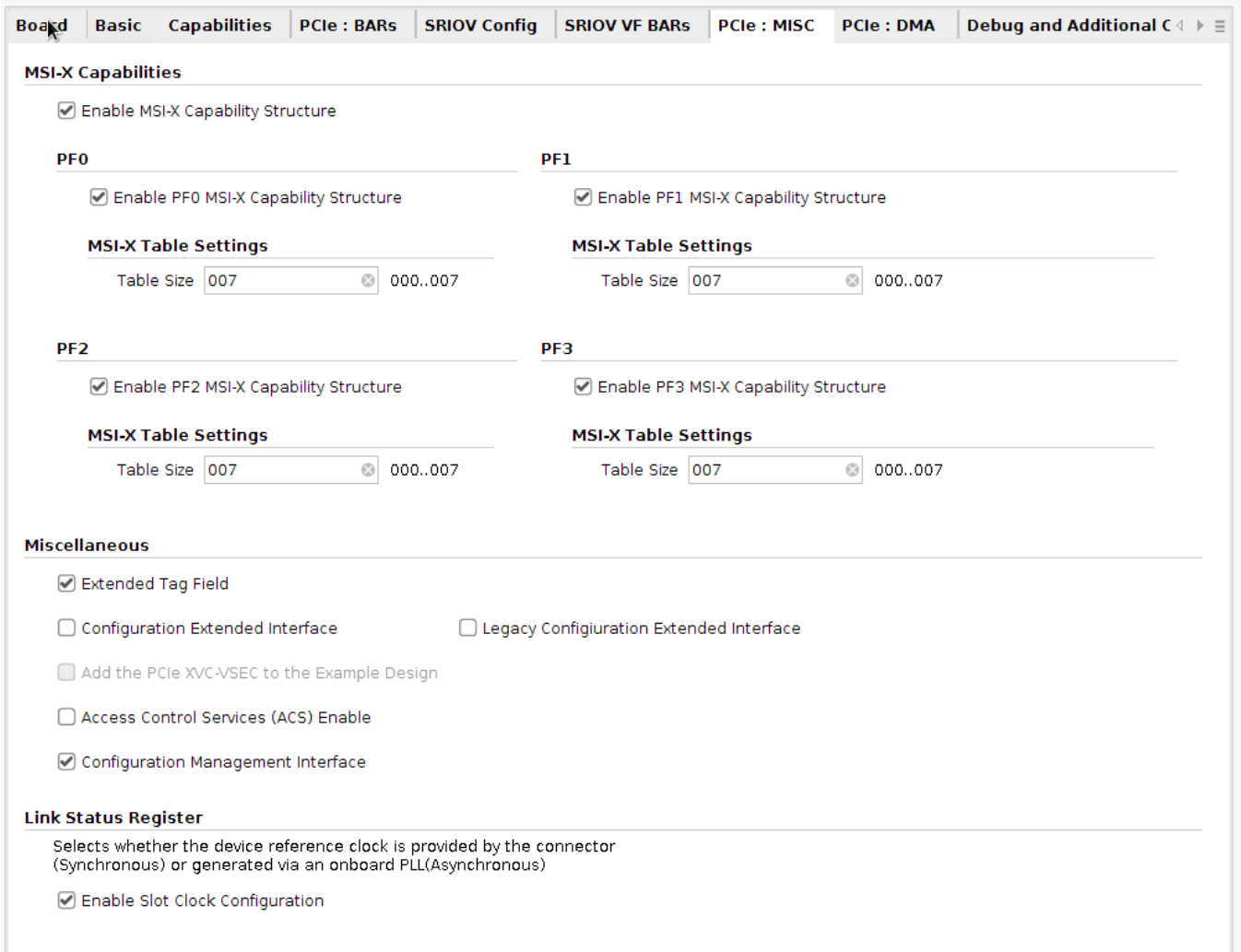
- “BAR”：使用复选框来选择适用的 BAR。
- “Type”（类型）：选择相关选项：
  - DMA：固定于 BAR0 空间。
  - AXI Lite Master：固定于 BAR1 空间。
  - AXI Bridge Master：固定于 BAR2 空间。对于所有其它 BAR，请选择 AXI Lite Master 或 AXI Bridge Master。

    注释：当前，IP 针对每个 VF 支持最多一个 DMA BAR（或一个管理 BAR，因为只需邮箱即可）。其它 BAR 则可配置为 AXI Lite Master，通过 AXI4-Lite 总线来访问所分配的存储器空间。虚拟功能 BAR 不支持 I/O 空间，必须配置为映射到适当的存储器空间。
- “64-bit”（64 位）：VF BAR 可采用 64 位或 32 位。默认为 64 位 BAR。
  - DMA BAR 支持 64 位寻址。
  - 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“Size”范围取决于选择的是 32 位 BAR 还是 64 位 BAR。“Supported Page Sizes”（支持的页面大小）字段表示 PF 所支持的所有页面大小，并且对于 SR-IOV 规范而言，这是必需字段。系统软件根据“Supported Page Size”字段来设置“System Page Size”（系统页面大小）字段，后者用于映射 VF BAR 存储器地址。每个 VF BAR 地址都与系统页面边界对齐。默认情况下，DMA 空间为 32 KB。分配如此多的空间后，用户逻辑即可为每个 VF 功能访问 256 个队列。
- “Value”（值）：表示基于当前选择分配给 BAR 的值。

“PCIe MISC”选项卡

“PCIe Miscellaneous”（PCIe 杂项）选项卡如下图所示。

图 34： **“PCIe MISC”选项卡**

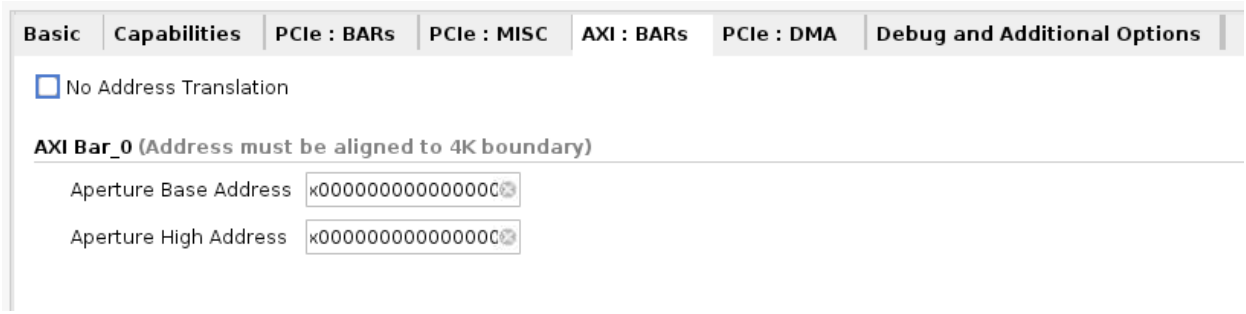


- “MSI-X Capabilities”（MSI-X 功能）：默认启用 MSI-X。可按需设置不同物理功能的 MSI-X 设置。
- “MSI-X Table Settings”（MSI-X 表设置）：用于定义 MSI-X 表结构。
  - “Table Size”（表大小）：用于指定 MSI-X 表的大小。默认值为 8（每个功能有 8 个中断矢量）。
- “Extended Tag Field”（扩展标签字段）：默认情况下，对于 UltraScale+™ 器件，“Extended Tag”（扩展标签）选项可提供 256 个标签。如果不选中“Extended Tag”选项，DMA 会使用 32 个标签。
- “Configuration Extended Interface”（配置扩展接口）：选中 PCIe 扩展接口即可增加配置空间。选中“Configuration Extend Interface”时，用户负责添加接口扩展逻辑以使其正常工作。
- “Access Control Server (ACS) Enable”（启用访问控制服务器 (ACS)）：表示默认情况下选中 ACS。
- “Configuration Management Interface”（配置管理接口）：此接口用于在配置空间寄存器上执行读取和写入。
- “Link Status Register”（链路状态寄存器）：默认情况下，选中“Enable Slot Clock Configuration”（启用时隙时钟配置）。这意味着在链路状态寄存器中启用时隙配置位。

“AXI BARs”选项卡

“SRIOV VF BARs”选项卡如下图所示。

图 35：**“AXI BARs”选项卡**



Slave Bridge 地址转换

- “No Address Translation”（无地址转换）：选中该选项时，DMA 不执行任何地址转换。其中会提供一个完整的 64 位 BAR 空间，您负责按需执行任何地址转换。如果 DMA 需要地址转换，请勿选择该选项。
- “AXI Bar_0 Address Translation”（AXI Bar_0 地址转换）：可按所需的值对 **Aperture Base Address**（间隙基址）和 **Aperture High Address**（间隙高位地址）进行编程。它可提供 AXI BAR 大小。高位地址（大于 BAR 大小）的地址转换可按需进行编程。

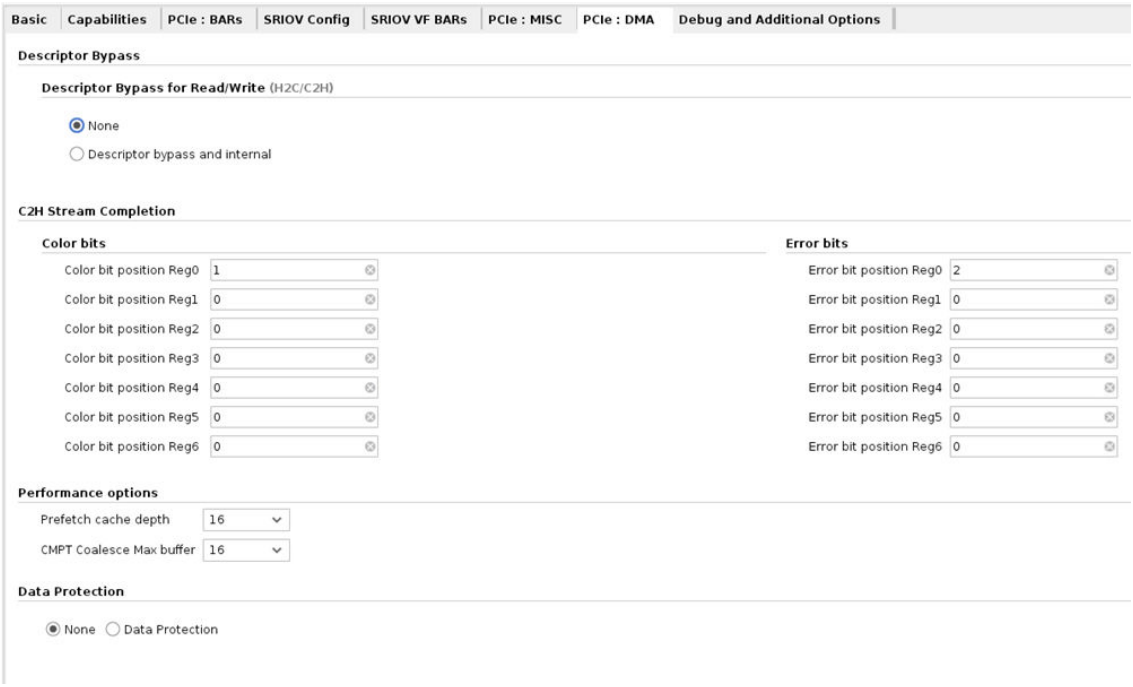
相关信息

Slave Bridge

“PCIe DMA”选项卡

“PCIe DMA”选项卡如下图所示。

图 36：**“PCIe DMA”选项卡**



- “Descriptor Bypass for Read/Write (H2C/C2H)”（对应读取/写入的描述符旁路 (H2C/C2H)）：有两个选项可供选择。

  注释：在此模式（内部模式）下，DMA 不会绕过任何 H2C 或 C2H 描述符。

  - “Descriptor bypass and Internal”（描述符旁路和内部）：在此模式下，启用旁路输出和旁路输入的描述符端口。根据上下文设置，可在描述符旁路输出上传出 H2C 或 C2H 描述符。用户可在描述符旁路输入端口上传入描述符。

- “C2H Stream Completion”（C2H 串流完成）：

  - “C2H Stream Completion Color bits”（C2H 串流完成颜色位）：完成条目中的完成颜色位的位置。对于 64 位完成，从位 0 到 511，有 7 个寄存器可用于编程。您可对这些位进行编程，并生成 BIT 文件。在 DMA 传输器件，输入管脚 `s_axis_c2h_cmpt_ctrl_color_idx[2:0]` 用于判定要使用的颜色位的位置。在寄存器 0 中选中默认位元位置 1。

  - “C2H Stream Completion Error bits”（C2H 串流完成错误位）：完成条目中的完成错误位的位置。对于 64 位完成，从位 0 到 511，有 7 个寄存器可用于编程。您可对这些位进行编程，并生成 BIT 文件。在 DMA 传输期间，输入管脚 `s_axis_c2h_cmpt_ctrl_err_idx[2:0]` 用于判定要使用的错误位的位置。在寄存器 0 中选中默认位元位置 2。

- 性能选项：

  - “Pre-fetch cache depth”（预取高速缓存深度）：预取高速缓存最多支持 64 个队列。选择 16 或 64（默认 16）。预取高速缓存可以在任意给定时间支持相应数量的队列保持活动状态。当某一个活动队列为该队列的所有包完成提取并交付所有描述符后，它就释放高速缓存条目以供其它活动队列使用。高速缓存大小越大，支持的活动队列数量越多，但面积也将增大。

  - “CMPT Coalesce Max buffer”（CMPT 合并最大缓冲器）：完成 (CMPT) 合并最大缓冲器支持多达 64 个缓冲器。选择 16 或 32（默认 16）。CMPT 合并缓冲器的每个条目都会将多个完成（最多 64B）合并组成单个队列，然后再写入主机，从而改善带宽利用率。CMPT 合并缓冲器越深，可合并的队列数就越多，但缺点是会增大面积。

- “Data Protection”（数据保护）：包括奇偶校验和端到端数据保护。默认情况下，不启用数据保护。

  不启用“**Data Protection**”时：

  - 无需在 C2H 数据和控制接口上提供任何 CRC/ECC 值。
  - 这将不会记录任何错误，也不会丢弃任何数据包。
  - 用户应将 ECC 和 CRC 端口接地。
  - CMPT 奇偶校验不受此参数影响。

    注释：您必须始终在 CMPT 上提供奇偶校验。

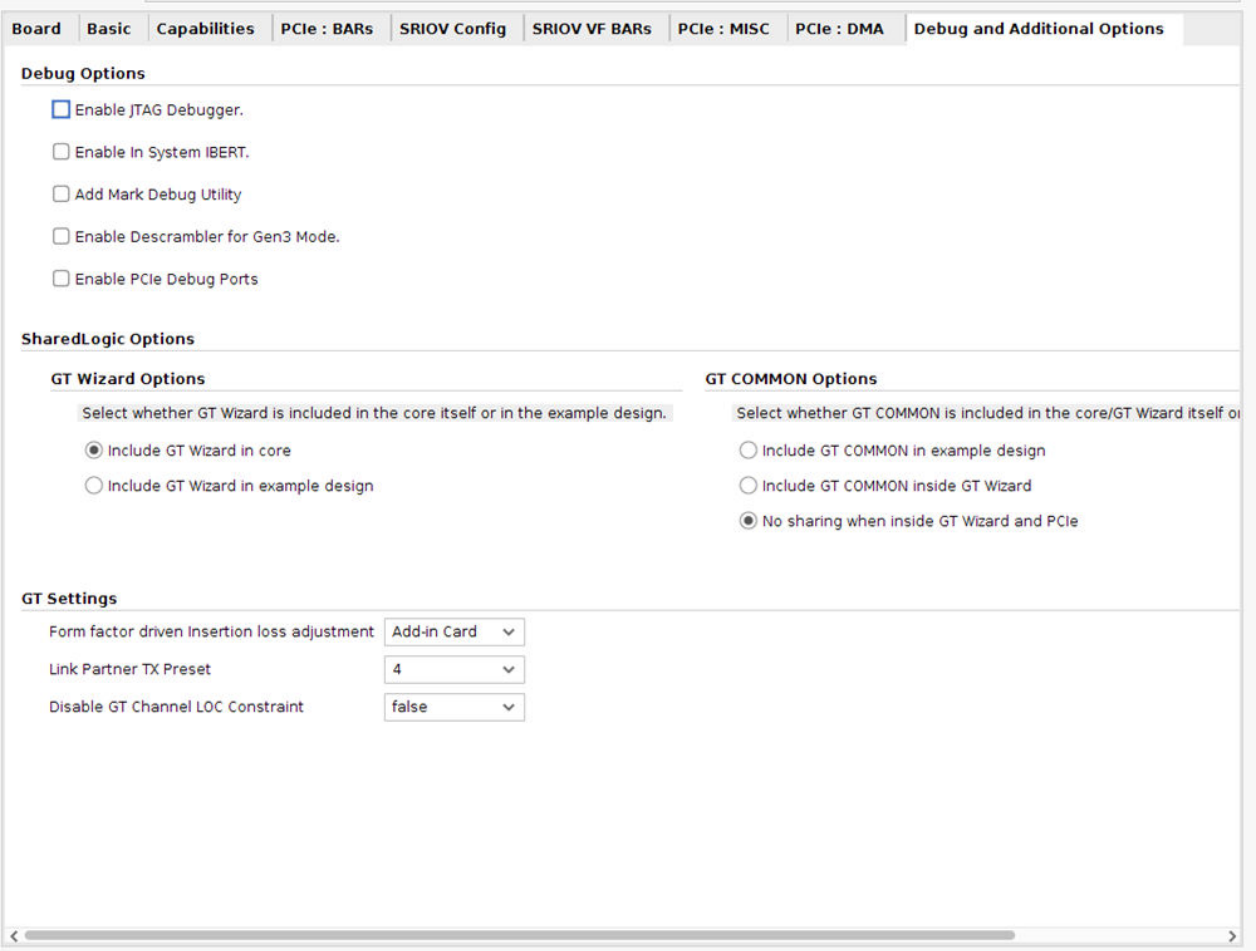
  启用“**Data Protection**”时：

  - 您必须在 C2H 数据和控制接口上发送 CRC/ECC 值。
  - 如有任何 ECC 或 CRC 错误，将记录错误位并将数据包发送给主机。
  - 如果启用错误中断，则将向主机发送中断。
  - FATAL 错误可在 QDMA_C2H_FATAL_ERR_ENABLE 寄存器中启用。
    - QDMA_C2H_FATAL_ERR_ENABLE[0]：如果该位完成置位，那么发生错误后将丢弃所有包。
    - QDMA_C2H_FATAL_ERR_ENABLE[1]：如果该位完成置位，那么奇偶校验将反相，并向 PCIe 发送错误包。

“Debug and Additional Options”选项卡

“Debug and Additional Options”（调试和其它选项）选项卡如下所示。

图 37：**“Debug and Additional Options”选项卡**



调试选项

- “Enable JTAG Debugging”（启用 JTAG 调试）：该功能提供了易于使用的调试功能，用于执行：
  - “**LTSSM state transitions**”（LTSSM 状态转换）：用于显示从链路建立开始后执行的所有 LTSSM 状态转换。
  - “**PHY Reset FSM transitions**”（PHY 复位 FSM 转换）：用于显示 PHY 复位 FSM（供 PCIe 解决方案 IP 使用的内部状态机）。
  - “**Receiver Detect**”（接收器检测）：用于显示已成功完成接收器检测的所有通道

  欲知详情，请参阅《UltraScale+ Integrated Block for PCI Express LogiCORE IP 产品指南》(PG213)。
- “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。
- “Add Mark Debug Utility”（添加标记调试实用工具）：可通过 `mark_debug` 属性添加预定义的 PCIe 信号，以便在 ILA 中添加这些信号用于调试。

- “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 PCIe Debug Ports”（启用 PCIe 调试端口）：保留。此版本支持此功能特性。

“Shared Logic”（共享逻辑）选项

- “GT Wizard Options”（GT 向导选项）：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 Options”（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。此特定选项可交付静态 GT COMMON 封装文件，其中包含最新设置。

“GT Settings”（GT 设置）

- “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）：值为 15 dB，这是默认选项。
  - “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。但对于部分系统，预置值 5 可能更适合。
- “Disable GT Channel LOC Constraint”（禁用 GT 通道 LOC 约束）：保留。在此版本中不予支持。

用户参数

另有其它核自定义选项可供使用。欲知详情，请参阅答复记录 72352。

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)。

约束子系统

所需约束

QDMA Subsystem for PCIe 子系统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。这些约束在赛灵思设计约束 (XDC) 文件中提供。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_GTE4 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 Integrated Block for PCIe 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。

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。

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 = xcvu9p-flgb2104-2-i

时钟频率

如需了解有关时钟要求的详细信息，请参阅《UltraScale+ Integrated Block for PCI Express LogiCORE IP 产品指南》(PG213)。

时钟管理

如需了解有关时钟要求的详细信息，请参阅《UltraScale+ Integrated Block for PCI Express LogiCORE IP 产品指南》(PG213)。

时钟布局

如需了解有关时钟要求的详细信息，请参阅《UltraScale+ Integrated Block for PCI Express LogiCORE IP 产品指南》(PG213)。

bank 分配

本节不适用于此 IP 子系统。

收发器布局

本节不适用于此 IP 子系统。

I/O 标准与布局

本节不适用于此 IP 子系统。

调整集成块核的位置

默认情况下，IP 核级约束可将块 RAM、收发器和 PCIe 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（即，新位置当前被另一个模块占据），有 2 种方法可用：

- 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)。

基本仿真

您可生成 AXI-MM 和 AXI-ST 选项的仿真模型并对其进行仿真。简单的仿真模型选项支持您开发复杂的设计。

AXI-MM 模式

AXI4 存储器映射 (AXI-MM) 模式的设计示例在用户侧有 512 KB 块 RAM，其中数据可写入块 RAM，也可从块 RAM 读取到主机。

主机到卡 (H2C) 传输启动后，DMA 会从主机存储器读取数据，并写入块 RAM。传输完成后，DMA 会更新写回状态并生成中断（如已启用）。随后会启动卡到主机 (C2H) 传输，DMA 会从块 RAM 读取数据并将其写入主机存储器。原始数据将用于与 C2H 写入数据进行比对。H2C 和 C2H 各设有 1 个描述符，总计传输大小为 128 字节。

AXI-ST 模式

AXI4-Stream (AXI-ST) 模式的设计示例具有数据检查，用于检查来自 H2C 传输的数据，并有数据生成器用于生成数据以执行 C2H 传输。

启动 H2C 传输后，DMA 引擎会从主机存储器读取数据，并将其写入用户侧。传输完成后，DMA 会更新写回状态并生成中断（如已启用）。用户侧的数据检查器会检查是否存在预定义的数据，并将结果发布到预定义地址中以供用户应用读取。

启动 C2H 传输，数据生成器会生成预定义数据和关联的控制信号，并将其发送到 DMA。DMA 会将数据传输到主机、更新完成 (CMPT) 环条目/状态，并生成中断（如已启用）。

H2C 和 C2H 各设有 1 个描述符，总计传输大小为 128 字节。

相关信息

参考软件驱动程序流程

PIPE 模式仿真

QDMA Subsystem 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 PIPE Simulation”（启用 PIPE 仿真）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 代替随设计示例提供的 Root Port 模型。

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。

表 105：**输入命令和端点 PIPE 信号映射**

输入命令	端点 PIPE 信号映射
common_commands_in[25:0]	不使用

表 106：**输出命令和端点 PIPE 信号映射**

输出命令	端点 PIPE 信号映射
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 107：**输入总线与端点 PIPE 信号映射**

输入总线	端点 PIPE 信号映射
pipe_rx_0_sigs[31:0]	pipe_rx0_data_gt
pipe_rx_0_sigs[33:32]	pipe_rx0_char_is_k_gt
pipe_rx_0_sigs[34]	pipe_rx0_elec_idle_gt
pipe_rx_0_sigs[35]	pipe_rx0_data_valid_gt
pipe_rx_0_sigs[36]	pipe_rx0_start_block_gt

表 107：**输入总线与端点 PIPE 信号映射** (续)

输入总线	端点 PIPE 信号映射
pipe_rx_0_sigs[38:37]	pipe_rx0_syncheader_gt
pipe_rx_0_sigs[83:39]	不使用

表 108：**输出总线与端点 PIPE 信号映射**

输出总线	端点 PIPE 信号映射
pipe_tx_0_sigs[31: 0]	pipe_tx0_data_gt
pipe_tx_0_sigs[33:32]	pipe_tx0_char_is_k_gt
pipe_tx_0_sigs[34]	pipe_tx0_elec_idle_gt
pipe_tx_0_sigs[35]	pipe_tx0_data_valid_gt
pipe_tx_0_sigs[36]	pipe_tx0_start_block_gt
pipe_tx_0_sigs[38:37]	pipe_tx0_syncheader_gt
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)。

AMD XILINX

第 6 章

设计示例

本章包含有关 Vivado® Design Suite 中提供的设计示例的信息。

可用设计示例

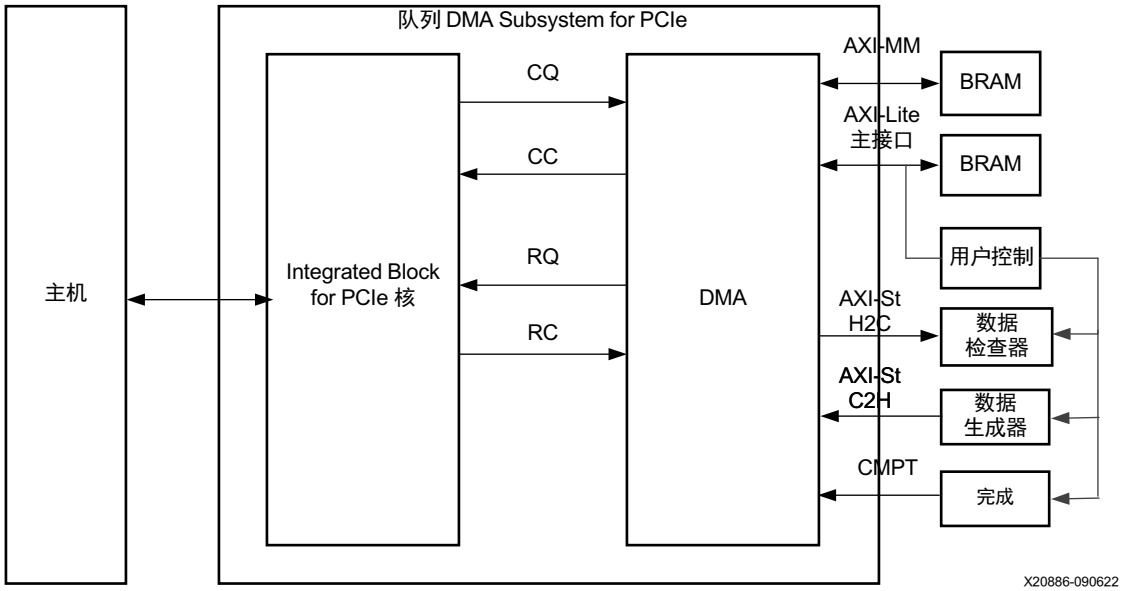
设计示例如下所述：

- 含完成的 AXI 存储器映射和 AXI4-Stream 的默认设计示例
- AXI 存储器映射设计示例
- 含完成的 AXI 串流设计示例
- AXI 串流环回设计示例
- 含描述符旁路输入/输出环回的设计示例
- AXI 串流性能设计示例

含完成的 AXI 存储器映射和 AXI4-Stream 的默认设计示例

以下设计示例是按如下设置生成的：“Basic”（基本）选项卡中的“DMA Interface Selection”（DMA 接口选择）选项设为“AXI Memory Mapped and AXI4-Stream with Completion”（含完成的 AXI 存储器映射和 AXI4-Stream）选项。

图 38：**默认设计示例**



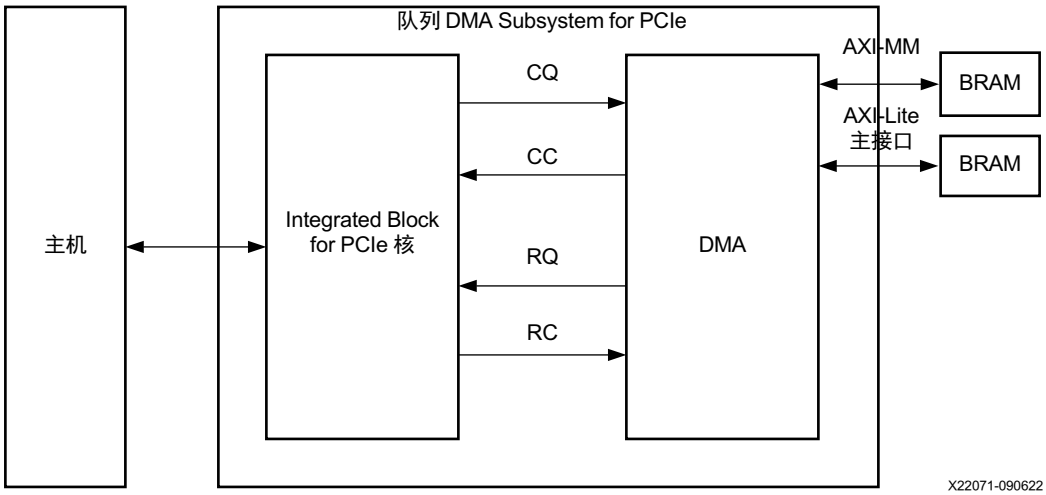
生成的设计示例可提供块，以便与 AXI 存储器映射和 AXI4-Stream 接口进行对接。

- AXI MM 接口连接到 512 KB 的块 RAM。
- AXI4-Stream 接口连接到定制数据生成器和数据检查器模块。
- CMPT 接口则连接到完成块生成器。
- 数据生成器和检查器仅适用于预定义模式，即 16 位增量模式（从 0 开始）。此数据文件包含在驱动程序包中。

模式生成器和检查器可使用 设计示例寄存器 中提供的寄存器来进行控制。这些寄存器只能通过 AXI4-Lite 主接口进行控制。要测试 QDMA Subsystem for PCIe 的 AXI4-Stream 接口，请确保 AXI4-Lite 主接口已存在。

AXI 存储器映射设计示例

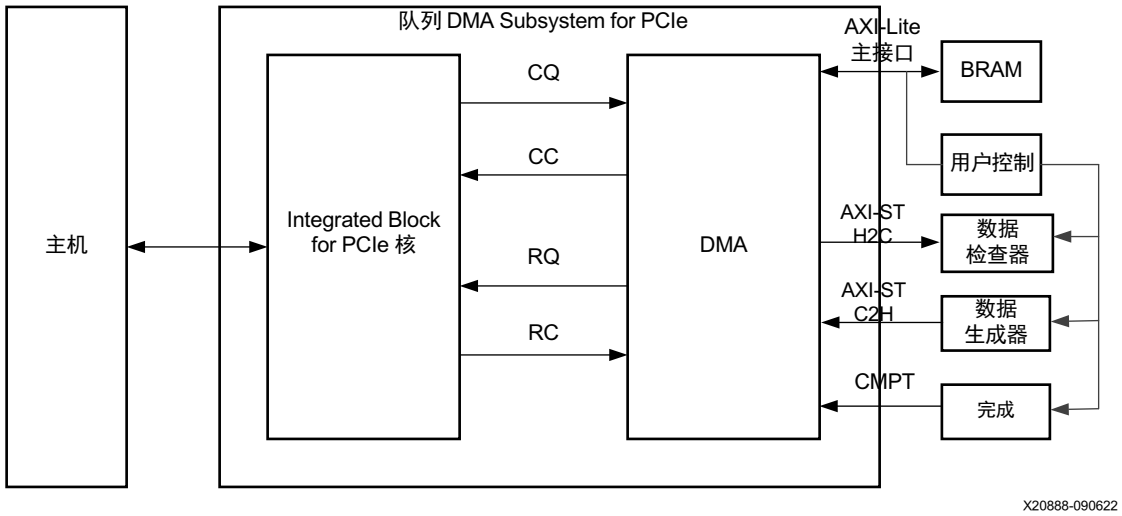
图 39：**AXI 存储器映射设计示例**



上述设计示例是按如下设置生成的：“Basic”（基本）选项卡中的“DMA Interface Selection”（DMA 接口选择）选项设为“AXI-MM only”（仅限 AXI-MM）。在此模式下，AXI MM 接口连接到 512 KB 的块 RAM。上图显示的是 AXI4-Lite 主接口连接到 4 KB 的块 RAM。对于主机到卡 (H2C) 传输，DMA 会从主机读取数据并写入块 RAM。对于卡到主机 (C2H) 传输，DMA 会从块 RAM 读取数据，并写入主机存储器。

含完成的 AXI 串流设计示例

图 40：**AXI4-Stream 设计示例**

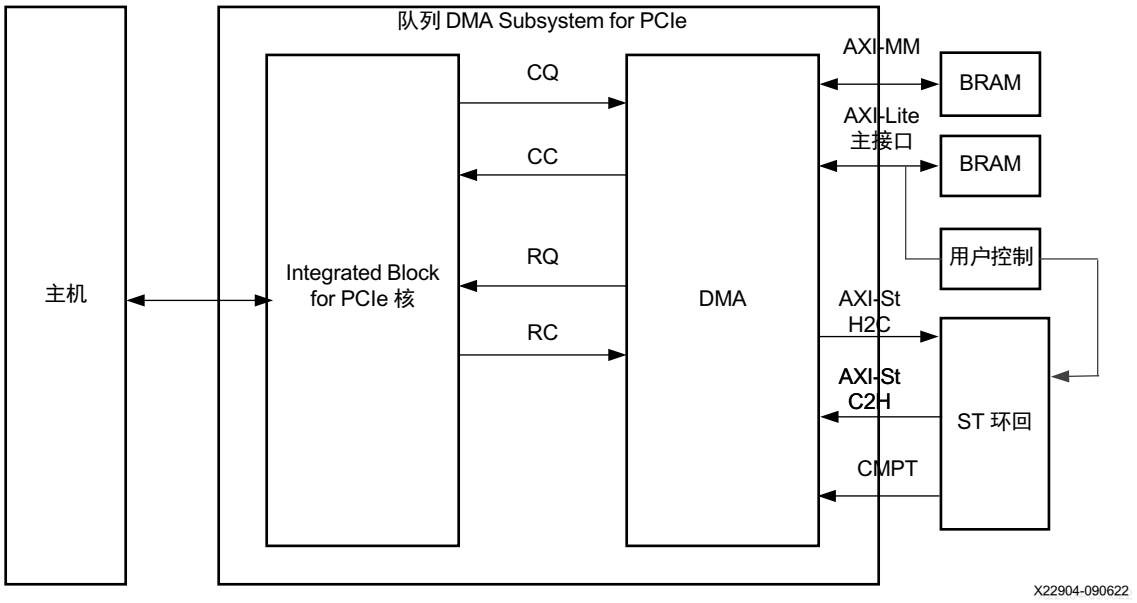


上述设计示例是按如下设置生成的：“Basic”（基本）选项卡中的“DMA Interface Selection”（DMA 接口选择）选项设为“AXI Stream with Completion”（含完成的 AXI 串流）。在此模式中，AXI-ST H2C 接口连接到数据检查器，AXI-ST C2H 接口连接到数据生成器，CMPT 接口则连接到完成生成器模块。该图显示 AXI4-Lite 主接口连接到 4 KB 的块 RAM 和用户控制逻辑。软件可通过 AXI4-Lite 主接口控制数据检查器和数据生成器。数据生成器和检查器仅适用于预定义模式，即 16 位增量模式（从 0 开始）。此数据文件包含在驱动程序包中。

模式生成器和检查器可使用 设计示例寄存器 中提供的寄存器来进行控制。

AXI 串流环回设计示例

图 41：**AXI4-Stream 环回设计示例**

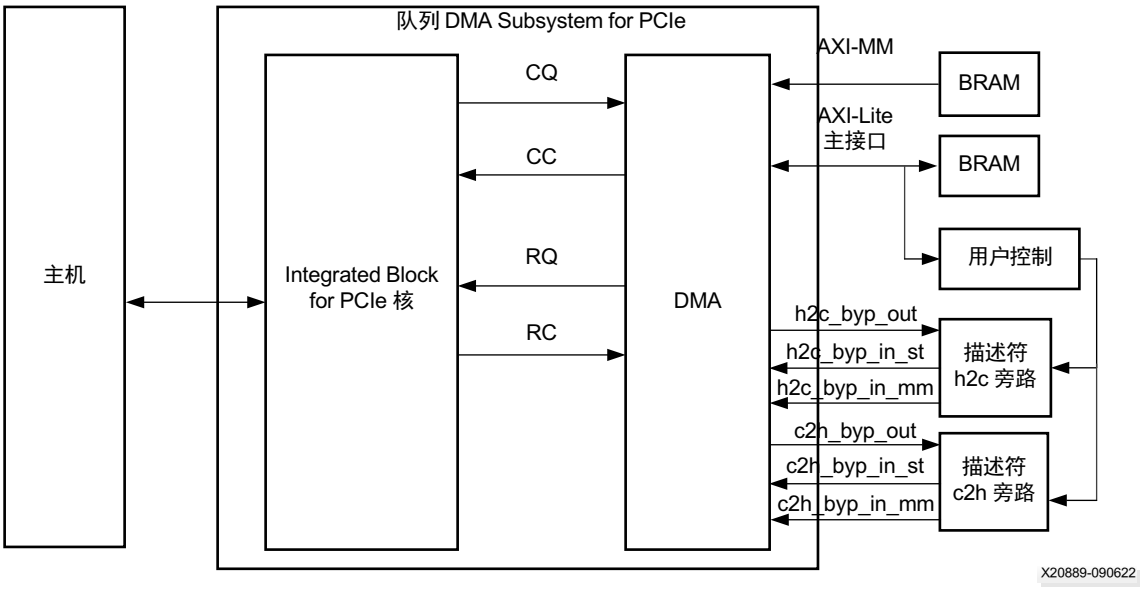


上述设计示例是按如下设置生成的：“Basic”（基本）选项卡中的“DMA Interface Selection”（DMA 接口选择）选项设为“AXI Stream with Completion”（含完成的 AXI 串流）。在此模式中，AXI-ST H2C 接口连接到数据检查器，AXI-ST C2H 接口连接到数据生成器，CMPT 接口则连接到完成生成器模块。但此设计示例也可用作串流环回设计。

将设计示例的寄存器 C2H_CONTROL_REG (0x008) bit[0] 设为 1 即可在串流环回设计中运行此设计示例。随后，设计示例会提取 H2C 串流包，并将其环回至 C2H 串流接口。完成数据包是从环回设计生成的。

含描述符旁路输入/输出环回的设计示例

图 42：**AXI 存储器映射和描述符旁路设计示例**



在“PCIe DMA”选项卡中选中“Descriptor Bypass for Read (H2C)”（对应读取的描述符旁路）选项和“Descriptor Bypass for Write (C2H)”（对应写入的描述符旁路）选项时，即可生成上述设计示例。选中这些选项卡的同时还可以搭配选中“Basic”（基本）选项卡中的任意 DMA 接口选项：

- AXI Memory Mapped and AXI4-Stream with Completion
- AXI Memory Mapped only
- AXI Stream with Completion
- AXI Memory Mapped with Completion

描述符旁路输入/输出环回由 AXI4-Lite 主接口进行控制，方法是写入设计示例寄存器 DESCRIPTOR_BYPASS (0x090) bit[0] 和 bit[1]。

要启用描述符旁路输出，需进行适当的上下文编程。欲知详情，请参阅 上下文编程。

C2H 串流简单旁路模式传输

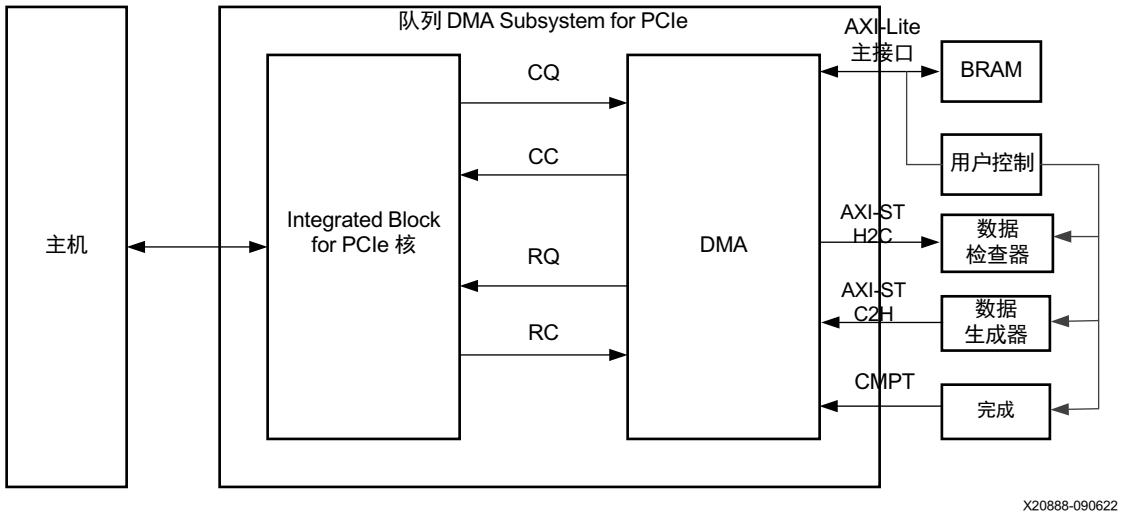
要将 QDMA 设置为以简单旁路模式进行数据传输，请执行以下操作

1. 将活动的 qid 写入寄存器 0x1408 (MDMA_C2H_PFCH_BYP_QID)。
2. 从 0x140C (MDMA_c2H_PFCH_BYP_TAG) 读取标签值。
3. 在设计示例寄存器 C3H_PREFETCH_TAG 0x24 中写入用于提取标签的标签值和 qid。qid 位为 [26:16]，标签位为 [6:0]。
4. 写入寄存器 DESCRIPTOR_BYPASS 0x90 bits [2:0] = `3'b100`，以设置简单旁路描述符环回。

设置完初始 C2H 串流数据传输后，prefetch（预取）标签即变为有效，直至 qid 变为有效为止。当前 qid 变为无效后，必须生成新标签。

AXI 串流性能设计示例

图 43：**AXI4-Stream 设计示例**



生成 QDMA_0 IP 后，您可在 Tcl 窗口中执行以下命令来获取性能设计示例。

```
set_property CONFIG.performance_exdes{true} [get_ips qdma_0]
```

为 QDMA IP 生成输出文件。然后为 QDMA IP 生成设计示例。

设计示例寄存器

表 109：**设计示例寄存器**

寄存器	地址	描述
C2H_ST_QID (0x000)	0x000	AXI-ST C2H 队列 ID
C2H_ST_LEN (0x004)	0x004	AXI-ST C2H 传输长度
C2H_CONTROL_REG (0x008)	0x008	AXI-ST C2H 模式生成器控制
H2C_CONTROL_REG (0x00C)	0x00C	AXI-ST H2C 控制
H2C_STATUS (0x010)	0x010	AXI-ST H2C 状态
C2H_STATUS (0x018)	0x018	AXI-ST C2H 状态
C2H_PACKET_COUNT (0x020)	0x020	要传输的 AXI-ST C2H 包数
C2H_COMPLETION_DATA_0 (0x030) 到 C2H_COMPLETION_DATA_7 (0x04C)	0x4C-0x030	AXI-ST C2H 完成数据
C2H_COMPLETION_SIZE (0x050)	0x050	AXI-ST 完成数据类型
SCRATCH_REG0 (0x060)	0x060	临时寄存器 0
SCRATCH_REG1 (0x064)	0x064	临时寄存器 1
C2H_PACKETS_DROP (0x088)	0x088	AXI-ST C2H 包丢弃计数
C2H_PACKETS_ACCEPTED (0x08C)	0x08C	AXI-ST C2H 包接受计数

表 109：**设计示例寄存器** (续)

寄存器	地址	描述
DESCRIPTOR_BYPASS (0x090)	0x090	C2H 和 H2C 描述符旁路环回
USER_INTERRUPT (0x094)	0x094	用户中断、矢量编号、功能编号
USER_INTERRUPT_MASK (0x098)	0x098	用户中断掩码
USER_INTERRUPT_VECTOR (0x09C)	0x09C	用户中断矢量
DMA_CONTROL (0x0A0)	0x0A0	DMA 控制
VDM_MESSAGE_READ (0x0A4)	0x0A4	VDM 报文读取

C2H_ST_QID (0x000)

表 110：**C2H_ST_QID (0x000)**

位	默认	访问类型	字段	描述
[31:11]	0	不适用		保留
[10:0]	0	RW	c2h_st_qid	AXI4-Stream C2H 队列 ID

C2H_ST_LEN (0x004)

表 111：**C2H_ST_LEN (0x004)**

位	默认	访问类型	字段	描述
[31:16]	0	不适用		保留
[15:0]	0	RW	c2h_st_len	AXI4-Stream 包长度

C2H_CONTROL_REG (0x008)

表 112：**C2H_CONTROL_REG (0x008)**

位	默认	访问类型	描述
[31:6]	0	不适用	保留
[5]	0	RW	C2H 串流标记请求 C2H 串流标记响应将寄存在地址 0x18 的位 [0]。
[4]	0	不适用	保留
[3]	0	RW	禁用完成。对于此包，将不会有任何完成。
[2]	0	RW	即时数据。 当该位完成置位后，数据生成器会立即发送数据。该位属于自清除位。写入 1 即可发起传输。
[1]	0	RW	启动 AXI-ST C2H 传输。该位属于自清除位。写入 1 即可发起传输。
[0]	0	RW	串流环回。当该位完成置位后，来自卡侧 H2C 串流端口的数据包会环回到 C2H 串流端口。

对于普通 C2H 串流包传输，请将地址偏移 0x08 设置为 0x2。

对于 C2H 即时数据传输，请将地址偏移 0x8 设置为 0x4。

对于 C2H/H2C 串流环回，请将地址偏移 0x8 设置为 0x1。

H2C_CONTROL_REG (0x00C)

表 113：**H2C_CONTROL_REG (0x00C)**

位	默认	访问类型	描述
[31:30]	0	不适用	保留
[0]	0	RW	将 H2C 传输的匹配位清零。

H2C_STATUS (0x010)

表 114：**H2C_STATUS (0x010)**

位	默认	访问类型	描述
[31:15]	0	不适用	保留
[14:4]	0	R	H2C 传输队列 ID
[3:1]	0	不适用	保留
[0]	0	R	H2C 传输匹配

C2H_STATUS (0x018)

表 115：**C2H_STATUS (0x018)**

位	默认	访问类型	描述
[31:30]	0	不适用	保留
[0]	0	R	C2H 标记响应

C2H_PACKET_COUNT (0x020)

表 116：**C2H_PACKET_COUNT (0x020)**

位	默认	访问类型	描述
[31:10]	0	不适用	保留
[9:0]	0	RW	要传输的 AXI-ST C2H 包的数量

C2H_PREFETCH_TAG(0x024)

表 117：**C2H_PREFETCH_TAG (0x024)**

位	默认	访问类型	描述
[31:27]	0	不适用	保留

表 117：**C2H_PREFETCH_TAG (0x024)** (续)

位	默认	访问类型	描述
[26:16]	0	RW	预取标签的 Qid
[15:7]	0	不适用	保留
[6:0]	0	RW	预取标签值

C2H_COMPLETION_DATA_0 (0x030)

表 118：**C2H_COMPLETION_DATA_0 (0x030)**

位	默认	访问类型	描述
[31:0]	0	不适用	AXI-ST C2H 完成数据 [31:0]

C2H_COMPLETION_DATA_1 (0x034)

表 119：**C2H_COMPLETION_DATA_1 (0x034)**

位	默认	访问类型	描述
[31:0]	0	不适用	AXI-ST C2H 完成数据 [63:32]

C2H_COMPLETION_DATA_2 (0x038)

表 120：**C2H_COMPLETION_DATA_2 (0x038)**

位	默认	访问类型	描述
[31:0]	0	不适用	AXI-ST C2H 完成数据 [95:64]

C2H_COMPLETION_DATA_3 (0x03C)

表 121：**C2H_COMPLETION_DATA_3 (0x03C)**

位	默认	访问类型	描述
[31:0]	0	不适用	AXI-ST C2H 完成数据 [127:96]

C2H_COMPLETION_DATA_4 (0x040)

表 122：**C2H_COMPLETION_DATA_4 (0x040)**

位	默认	访问类型	描述
[31:0]	0	不适用	AXI-ST C2H 完成数据 [159:128]

C2H_COMPLETION_DATA_5 (0x044)

表 123：**C2H_COMPLETION_DATA_5 (0x044)**

位	默认	访问类型	描述
[31:0]	0	不适用	AXI-ST C2H 完成数据 [191:160]

C2H_COMPLETION_DATA_6 (0x048)

表 124：**C2H_COMPLETION_DATA_6 (0x048)**

位	默认	访问类型	字段	描述
[31:0]	0	不适用	不适用	AXI-ST C2H 完成数据 [223:192]

C2H_COMPLETION_DATA_7 (0x04C)

表 125：**C2H_COMPLETION_DATA_7 (0x04C)**

位	默认	访问类型	描述
[31:0]	0	不适用	AXI-ST C2H 完成数据 [255:224]

C2H_COMPLETION_SIZE (0x050)

表 126：**C2H_COMPLETION_SIZE (0x050)**

位	默认	访问类型	描述
[31:13]	0	不适用	保留
[12]	0	RW	完成类型。 1'b1：NO_PLD_BUT_WAIT 1'b0：HAS PLD
[10:8]	0	RW	s_axis_c2h_cmpt_ctrl_err_idx[2:0] 完成错误位索引。 3'b000：选择第 0 个寄存器。 3'b111：未报告错误位。
[6:4]	0	RW	s_axis_c2h_cmpt_ctrl_col_idx[2:0] 完成颜色位索引。 3'b000：选择第 0 个寄存器。 3'b111：未报告颜色位。
[3]	0	RW	s_axis_c2h_cmpt_ctrl_user_trig 完成用户触发器
[1:0]	0	RW	AXI4-Stream C2H 完成数据大小。 00：8 字节 01：16 字节 10：32 字节 11：64 字节

相关信息

AXI4-Stream C2H 完成端口

SCRATCH_REG0 (0x060)

表 127：**SCRATCH_REG0 (0x060)**

位	默认	访问类型	描述
[31:0]	0	RW	临时寄存器

SCRATCH_REG1 (0x064)

表 128：**SCRATCH_REG1 (0x064)**

位	默认	访问类型	描述
[31:0]	0	RW	临时寄存器

C2H_PACKETS_DROP (0x088)

表 129：**C2H_PACKETS_DROP (0x088)**

位	默认	访问类型	描述
[31:0]	0	R	每次传输丢弃的 AXI-ST C2H 包（描述符）的数量

每次 AXI-ST C2H 传输均可包含一个或多个描述符，具体数量取决于传输大小和 C2H 缓冲器大小。该寄存器会显示当前传输中已丢弃的描述符数量。每次传输开始时，该寄存器都将复位为 0。

C2H_PACKETS_ACCEPTED (0x08C)

表 130：**C2H_PACKETS_ACCEPTED (0x08C)**

位	默认	访问类型	描述
[31:0]	0	R	每次传输接受的 AXI-ST C2H 包（描述符）的数量

每次 AXI-ST C2H 传输均可包含一个或多个描述符，具体数量取决于传输大小和 C2H 缓冲器大小。该寄存器会显示当前传输中已接受的描述符数量。每次传输开始时，该寄存器都将复位为 0。

DESCRIPTOR_BYPASS (0x090)

表 131：**描述符旁路 (0x090)**

位	默认	访问类型	字段	描述
[31:3]	0	不适用		保留

表 131：**描述符旁路 (0x090)** (续)

位	默认	访问类型	字段	描述
[2:1]	0	RW	c2h_dsc_bypass	C2H 描述符旁路环回。当该位完成置位后，C2H 描述符旁路输出端口将环回至 C2H 描述符旁路输入端口。 2'b00：无旁路环回。 2'b01：C2H MM 描述符旁路环回和 C2H 串流高速缓存旁路环回。 2'b10：C2H 串流简单描述符旁路环回。 2'b11：H2C 串流 64 字节描述符环回至完成接口。
[0]	0	RW	h2c_dsc_bypass	H2C 描述符旁路环回。当该位完成置位后，H2C 描述符旁路输出端口将环回至 H2C 描述符旁路输入端口。 1'b1：H2C MM 和 H2C 串流描述符旁路环回 1'b0：无描述符环回

USER_INTERRUPT (0x094)

表 132：**用户中断 (0x094)**

位	默认	访问类型	字段	描述
[31:20]	0	不适用		保留
[19:12]	0	RW	usr_irq_in_fun	用户中断功能编号
[11:9]	0	不适用		保留
[8:4]	0	RW	usr_irq_in_vec	用户中断矢量编号
[3:1]	0	不适用		保留
[0]	0	RW	usr_irq	用户中断。当该位进行置位时，设计示例会生成用户中断。

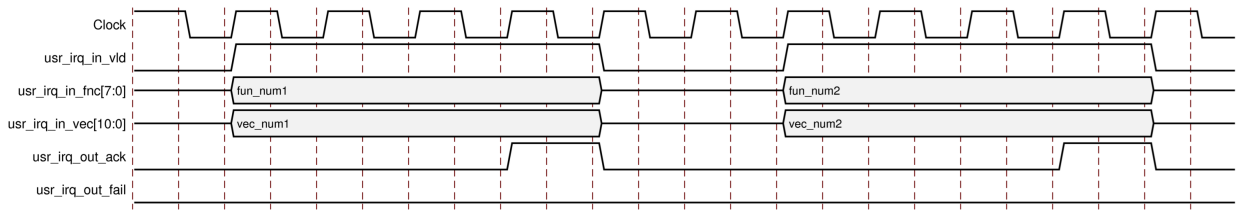
要生成用户中断，请执行以下操作：

1. 在位 [19:12] 中写入功能编号。对应于生成 `usr_irq_in_fnc` 用户中断的功能。
2. 在位 [8:4] 中写入 MSI-X 矢量编号。对应于为 `usr_irq_in_vec` 用户中断设置的 MSI-X 表中的条目。
3. 对位 [0] 写入 1 即可生成用户中断。从 DMA 生成 `usr_irq_out_ack` 后，该位会将自身清零。

上述三个步骤都是同时完成的，只需一次写入。

用户中断时序图如下所示：

图 44：**中断**



USER_INTERRUPT_MASK (0x098)

表 133：**用户中断掩码 (0x098)**

位	默认	访问类型	描述
[31:0]	0	RW	用户中断掩码

USER_INTERRUPT_VECTOR (0x09C)

表 134：**用户中断矢量 (0x09C)**

位	默认	访问类型	描述
[31:0]	0	RW	用户中断矢量

此处所提供的 `user_interrupt_mask[31:0]` 和 `user_interrupt_vector[31:0]` 寄存器均作为设计示例提供给用户中断聚合，用户中断聚合可为功能生成用户中断。`user_interrupt_mask[31:0]` 和 `user_interrupt_vector[31:0]` 两者间属于“与”关系（按位与），生成一个用户中断。`user_interrupt_vector[31:0]` 在读取寄存器上清零。

要生成用户中断，请执行以下操作：

1. 在 `user_interrupt[19:12]` 中写入功能编号。这对应于生成 `usr_irq_in_fnc` 用户中断的功能。
2. 在 `user_interrupt[8:4]` 中写入 MSI-X 矢量编号。这对应于 MSI-X 表中为 `usr_irq_in_vec` 用户中断设置的条目。
3. 在 `user_interrupt_mask[31:0]` 寄存器中写入掩码值。
4. 在 `user_interrupt_vector[31:0]` 寄存器中写入中断矢量值。

这样即可向 DMA 块生成用户中断。

有两种方法可用于生成用户中断：

- 写入 `user_interrupt[0]`，或者
- 写入已设置掩码的 `user_interrupt_vector[31:0]` 寄存器。

DMA_CONTROL (0x0A0)

表 135：**DMA 控制 (0x0A0)**

位	默认	访问类型	字段	描述
[31:1]		不适用		保留
[0]	0	RW	gen_qdma_reset	当 soft_reset 置位时，会向 DMA 块生成软核复位。此位在 100 个周期后会清零。

向 `DMA_control[0]` 写入 1 就会在 `soft_reset_n`（低电平有效）上生成软核复位。复位断言有效并保持 100 个周期，随后这些信号将断言无效。

VDM_MESSAGE_READ (0x0A4)

表 136：**VDM 报文读取 (0x0A4)**

位	默认	访问类型	描述
[31:0]		RO	VDM 报文读取

供应商定义报文 (VDM) 的 `st_rx_msg_data` 报文存储在设计示例的 FIFO 中。每次读取此寄存器 (0x0A4) 都将弹出一条 32 位报文。

自定义和生成设计示例

在“Customize IP”（自定义 IP）对话框中，使用 IP 设计示例的默认核参数值。

查看核参数后：

1. 右键单击组件名称。
2. 选中“Open IP Example Design”（打开 IP 设计示例）。

这样即可打开独立的设计示例。

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。

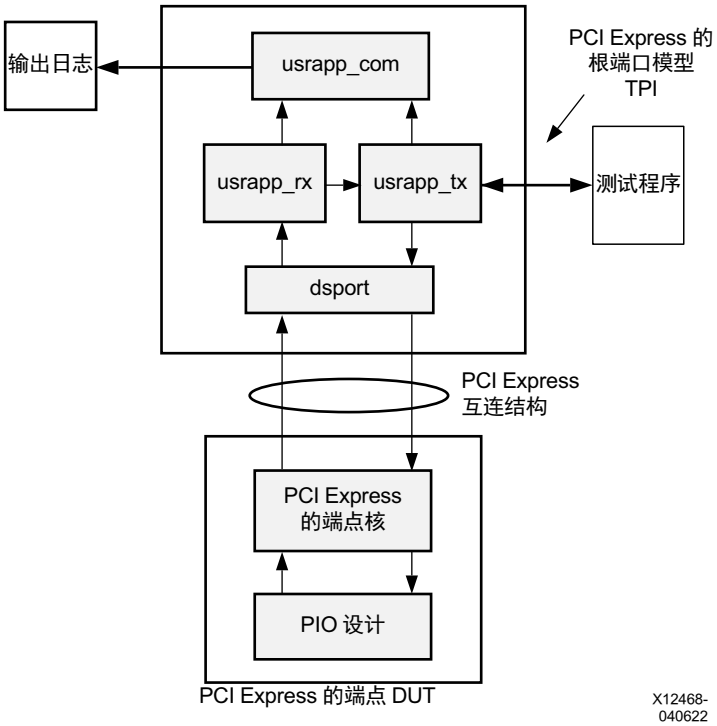
其中包含的根端口模型的源代码可提供模型，作为您的测试激励文件的起点。此外，包括初始化核配置空间、创建 TLP 传输事务、生成 TLP 日志和提供接口用于创建和验证测试在内的所有主要工作都已全部完成。这样，您即可集中精力来验证设计功能，而无需花时间来开发端点核测试激励文件基础架构。

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 45：**根端口模型和顶层端点**



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 块，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

所有仿真测试案例均以所提供的设计示例为基础。仿真任务会对特定设计示例寄存器进行读取和写入，以进行设置和检查。这些仿真任务可能并不一定适用于所有客户的设计。

可用测试

下表描述了为仿真提供的测试。这些测试是根据 QDMA IP 配置选择的。例如，如果选中仅限 AXI4-MM 选项，则会选中 qdma_mm_test0 测试用例，并在仿真期间执行。

表 137：**测试案例**

选项	测试名称	语言	描述
仅限 AXI4-MM	qdma_mm_test0	Verilog	1. 测试激励文件会将队列初始化，并在 H2C 方向执行 AXI4-MM 传输。 2. 然后，测试激励文件会将队列初始化，并在 C2H 方向执行 AXI4-MM 传输。 测试激励文件会将写入数据与读取数据进行比较，以检查数据是否正确。
仅限 AXI4-ST	qdma_st_test0	Verilog	1. 测试激励文件会将队列初始化、在 H2C 方向执行 AXI4-ST 传输，然后检查数据正确性。 2. 测试激励文件会将队列初始化、在 C2H 方向执行 AXI4-ST 传输，然后检查数据正确性。
含完成的 AXI4-MM 和 AXI4-ST	qdma_mm_st_test0	Verilog	1. 测试激励文件会将队列初始化，并在 H2C 方向执行 AXI4-MM 传输。随后，测试激励文件会将队列初始化、在 C2H 方向执行 AXI4-MM 传输，并通过比较来确认数据是否正确。 2. 测试激励文件会将队列初始化、在 H2C 方向执行 AXI4-ST 传输，然后检查数据正确性。 3. 随后，测试激励文件会将队列初始化、在 C2H 方向执行 AXI4-ST 传输，并检查数据正确性。 此测试适用于测试用例 `qdma_mm_test0` 与 `qdma_st_test0` 的组合。

Verilog 测试选项

您可通过编辑 `usr_pci_exp_usrapp_tx.v` 文件来更改测试用例。查找“testname”分配，并修改测试用例。

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

该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

仿真进程会创建 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` 文件各自分别包含接收和发射的每个 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。在 `error.dat` 中仅列出 PCIe 协议错误。DMA 数据不匹配或传输错误将打印在 log 日志文件中。

测试描述

此模型提供了一个测试程序接口 (TPI)。TPI 提供了通过调用一系列 Verilog 任务来创建测试的方法。所有测试都应遵循以下步骤：

1. 对唯一测试名称执行条件比较。
2. 等待复位和链路建立完成。
3. 为该队列初始化队列上下文。
4. 为队列发射包。
5. 验证测试是否成功。

模型任务列表

PCIe 相关任务

如需了解所有 PCIe 相关任务，请参阅《UltraScale+ Integrated Block for PCI Express LogiCORE IP 产品指南》(PG213)。

DMA 任务

表 138：**DMA 任务** [1]

名称	输入		输出	描述
TSK_QDMA_MM_H2C_TEST	qid, dsc_bypass irq_en	10:0 - -		此任务将在队列 ID“qid”上执行 H2C AXI-MM 传输。如果给定“dsc_bypass”，任务将执行描述符旁路 如果给定 irq_en，任务将设置 MSI-X 表，并将在传输完成时发送中断。
TSK_QDMA_MM_C2H_TEST	qid, dsc_bypass irq_en	10:0 - -		此任务将在队列 ID“qid”上执行 C2H AXI-MM 传输。如果给定“dsc_bypass”，任务将执行描述符旁路 如果给定 irq_en，任务将设置 MSI-X 表，并将在传输完成时发送中断。
TSK_QDMA_ST_C2H_TEST	qid, dsc_bypass	10:0 -		此任务将在队列 ID“qid”上执行 C2H AXI-ST 传输。如果给定“dsc_bypass”，任务将执行描述符旁路

表 138：**DMA 任务** [1] (续)

名称	输入		输出	描述
TSK_QDMA_ST_H2C_TEST	qid, dsc_bypass	10:0 -		此任务将在队列 ID“qid”上执行 H2C AXI-ST 传输。如果给定“dsc_bypass”，任务将执行描述符旁路

注释：

1. 建议仅运行该表中的 DMA 任务用于测试。

AMD XILINX

附录 A

升级

从 v3.1 到 v4.0 的更改

如需获取 QDMA Subsystem for PCIe 从 v3.1 到 v4.0 的更改列表，请参阅答复记录 75234。

与 DMA/Bridge Subsystem for PCI Express 的比较

下表描述了 DMA/Bridge Subsystem for PCI Express® 与 QDMA Subsystem for PCI Express 之间的差异。

表 139：**子系统比较**

	DMA/Bridge 子系统	QDMA 子系统
配置	最高 Gen3x16。	最高 Gen3x16。
通道/队列	4 条主机到卡 (H2C) 通道，4 条卡到主机 (C2H) 通道（含 1 个 PF）。	多达 2K 个队列（所有队列都可分配给 1 个 PF 或分布在全部 4 个通道中）。
SR-IOV	不支持。	支持（4 个 PF 和 252 个 VF）。
用户接口	配置为适用于 AXI4 存储器映射或 AXI4-Stream，但不得同时适用于这两者。	每个队列都将有上下文用于判定它进入 AXI4 存储器映射还是 AXI4-Stream。
用户中断	最多 16 个用户中断。	中断聚合（按功能）。
器件支持	支持 7 系列 Gen2 器件到 UltraScale+™ 器件。	仅支持 UltraScale+ 器件。
中断	支持遗留中断、MSI 中断和 MSI-X 中断。	支持 MSI-X。
驱动程序支持	Linux、Windows 示例驱动程序。	Linux、DPDK、Windows。

AMD XILINX

附录 B

GT 位置

如需了解有关 GT 位置的更多信息，请参阅《UltraScale+ Integrated Block for PCI Express LogiCORE IP 产品指南》(PG213) 中的“GT 位置”附录。

AMD XILINX

附录 C

调试

本附录包含有关赛灵思技术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

为了帮助您在使用子系统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

本产品指南是与该子系统相关的主要文档。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 Debug K-Map。

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”框找到对应该子系统的答复记录。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。

该子系统的主答复记录

答复记录 70927。

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。

调试工具

有许多工具可用于解决 QDMA Subsystem for PCIe 设计问题。至关重要的是要了解哪些工具可用于调试各种情况。

Vivado Design Suite 调试功能

Vivado® Design Suite 调试功能可以将逻辑分析器和虚拟 I/O 核直接插入到您的设计中。调试功能还支持您设置触发条件，以便在硬件中捕获应用和集成块端口信号。随后，您便可对捕获的信号进行分析。Vivado IDE 中的这个功能用来对在赛灵思器件中运行的设计进行逻辑调试和确认。

Vivado 逻辑分析器用于与下列逻辑调试 LogiCORE IP 核交互：

- ILA 2.0（及更高版本）
- VIO 2.0（及更高版本）

请参阅《Vivado Design Suite 用户指南：编程和调试》(UG908)。

硬件调试

硬件问题各不相同，可能是链路初始化问题，也可能是测试数小时后才会显现的问题。本节提供了常见问题的调试步骤。Vivado® 调试功能是可用于硬件调试的宝贵资源。可以通过调试功能来探测以下各个部分中提到的信号名称，以便对特定问题进行调试。

常规检查

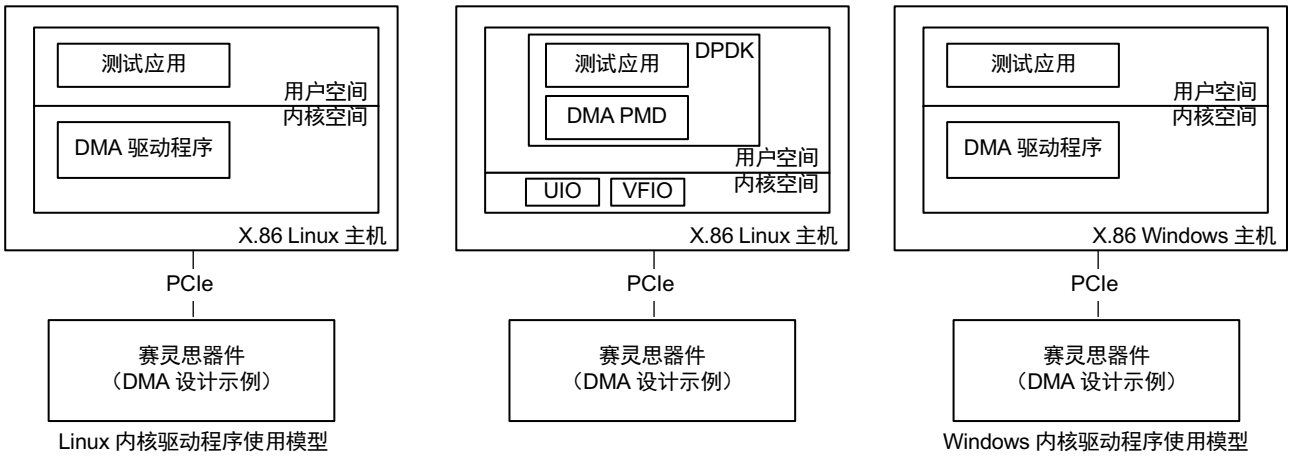
确保核的所有时序约束都已正确整合在设计示例中，并且在实现期间已满足所有约束条件。

- 在布局布线后的时序仿真中是否能够正常工作？如果在硬件中发现问题，但在时序仿真中没有出现问题，则可能表示存在 PCB 问题。确保所有时钟源均处于活动状态且无任何错误。
- 如果在设计中使用 MMCM，请通过监控 `locked` 端口确保所有 MMCM 都已被锁定。
- 如果您的输出为 0，请检查您的许可。

应用软件开发

器件驱动程序

图 46：**器件驱动程序**



- DPDK（数据平面开发套件）PMD（轮询模式驱动程序）使用模型
- DPDK 无需执行与系统调用关联的数据拷贝即可创建用户空间应用

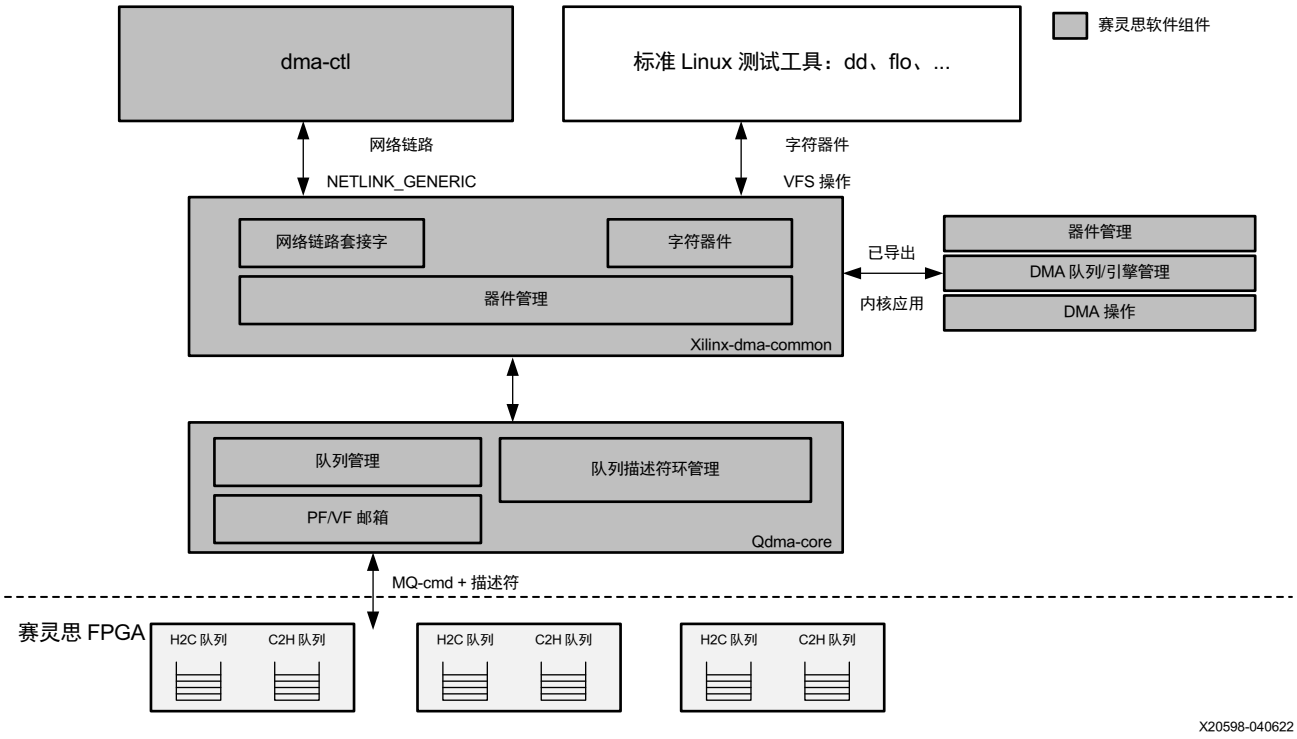
X20600-040622

上图显示了 Linux 和 Windows QDMA 软件驱动程序的使用模型。QDMA Subsystem for PCIe 设计示例是在通过 PCI Express® 连接到 X86 主机的赛灵思 FPGA 上实现的。

- 在第一种使用模式中，内核空间中的 QDMA 驱动程序在 Linux 上运行，而测试应用则在用户空间内运行。
- 在第二种使用模式中，使用 Data Plane Dev Kit (DPDK)（数据平面开发套件 (DPDK)）来开发完全在用户空间内运行的 QDMA Poll Mode Driver (PMD)（轮询模式驱动程序），并使用 UIO 和 VFIO 内核框架与 FPGA 进行通信。
- 在第三种使用模式中，内核空间中的 QDMA 驱动程序在 Windows 上运行，而测试应用则在用户空间内运行。

Linux QDMA 软件架构 (PF/VF)

图 47：**Linux DMA 软件架构**



QDMA 驱动程序由以下三个主要组件组成：

- “**Device control tool**”（器件控制工具）：用于创建 netlink socket（网络链路套接字），以供 PCIe 器件查询、查询管理、读取队列上下文等使用。
- “**DMA tool**”（DMA 工具）：它是用户空间应用，用于发起 DMA 传输事务。您可使用标准 Linux 实用工具 `dd` 或 `fio`，或者使用驱动程序包中的示例应用。
- “**Kernel space driver**”（内核空间驱动程序）：用于创建描述符并将用户空间功能转换为低级别命令，以便与器件进行交互。

使用驱动程序

赛灵思 DMA IP 驱动程序上提供了 Linux、DPDK 和 Windows 驱动程序和对应的文档。

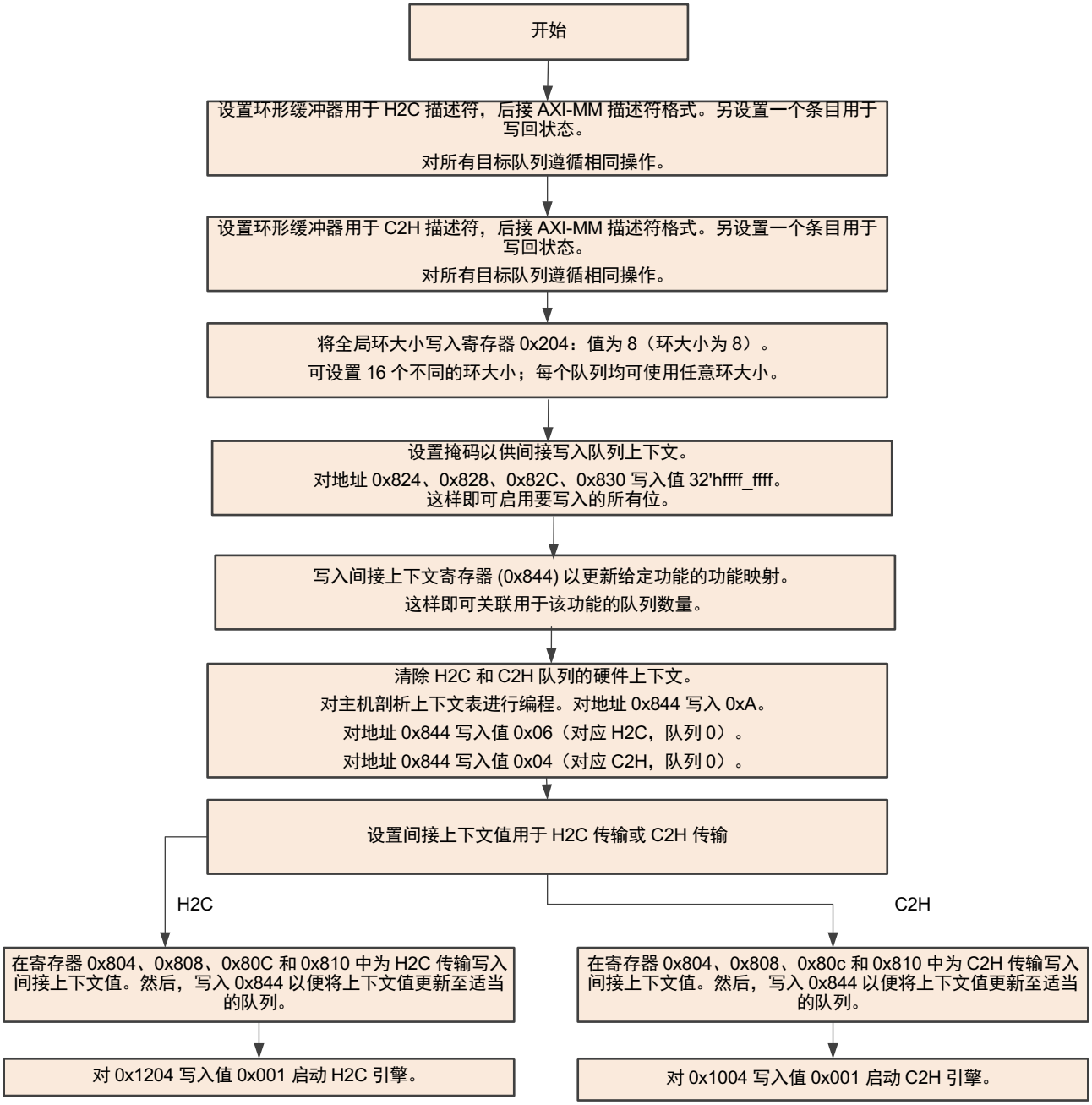
注释：从 QDMA 的 Linux 驱动程序 2022.1 版本起，如果设计使用的是串流队列，则必须通过 API 来显式启用这些队列，因为在模块加载时不会对这些队列进行配置。如果设计在上电时使用的是串联 PCIe 方法，那么在器件上加载完成阶段 2 之后必须完成启用操作。

重要提示！在所有功能 (PF/VF) 上都需要 8 MSI-X 矢量才能使用赛灵思 QDMA IP 驱动程序。

参考软件驱动程序流程

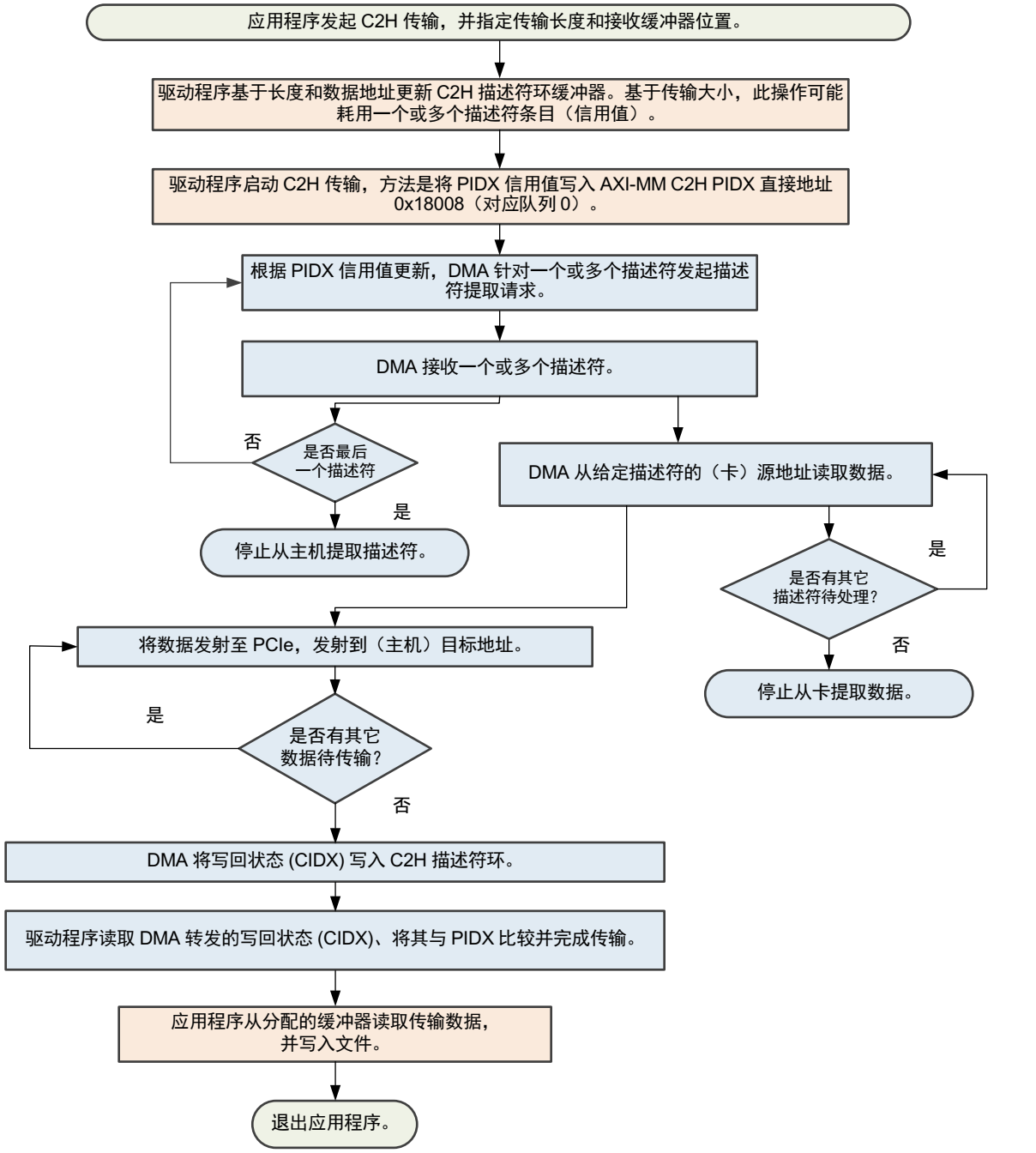
AXI4 存储器映射流程图

图 48：**AXI4 存储器映射流程图**



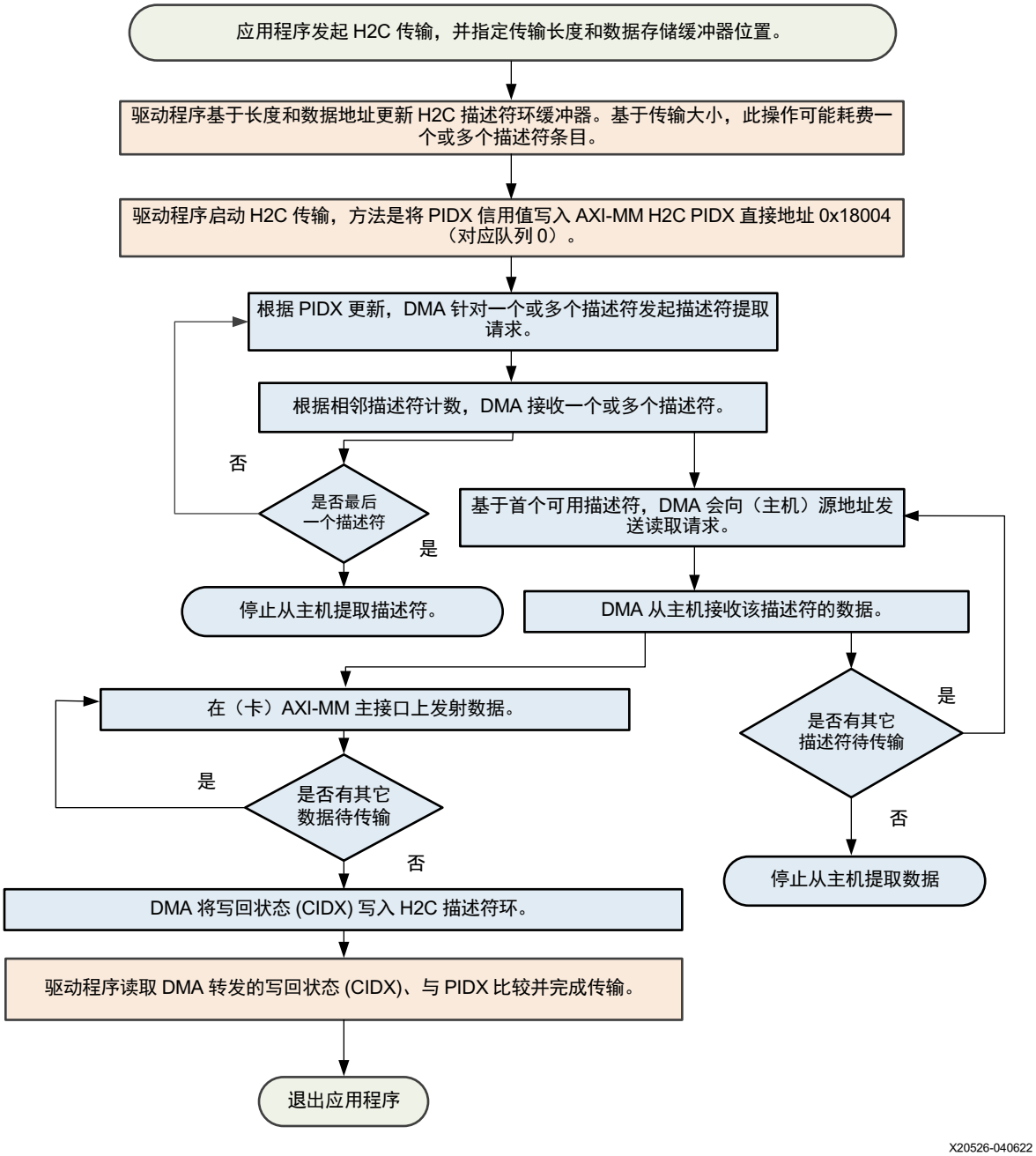
AXI4 存储器映射 C2H 流程

图 49：**AXI4 存储器映射卡到主机 (C2H) 流程图**



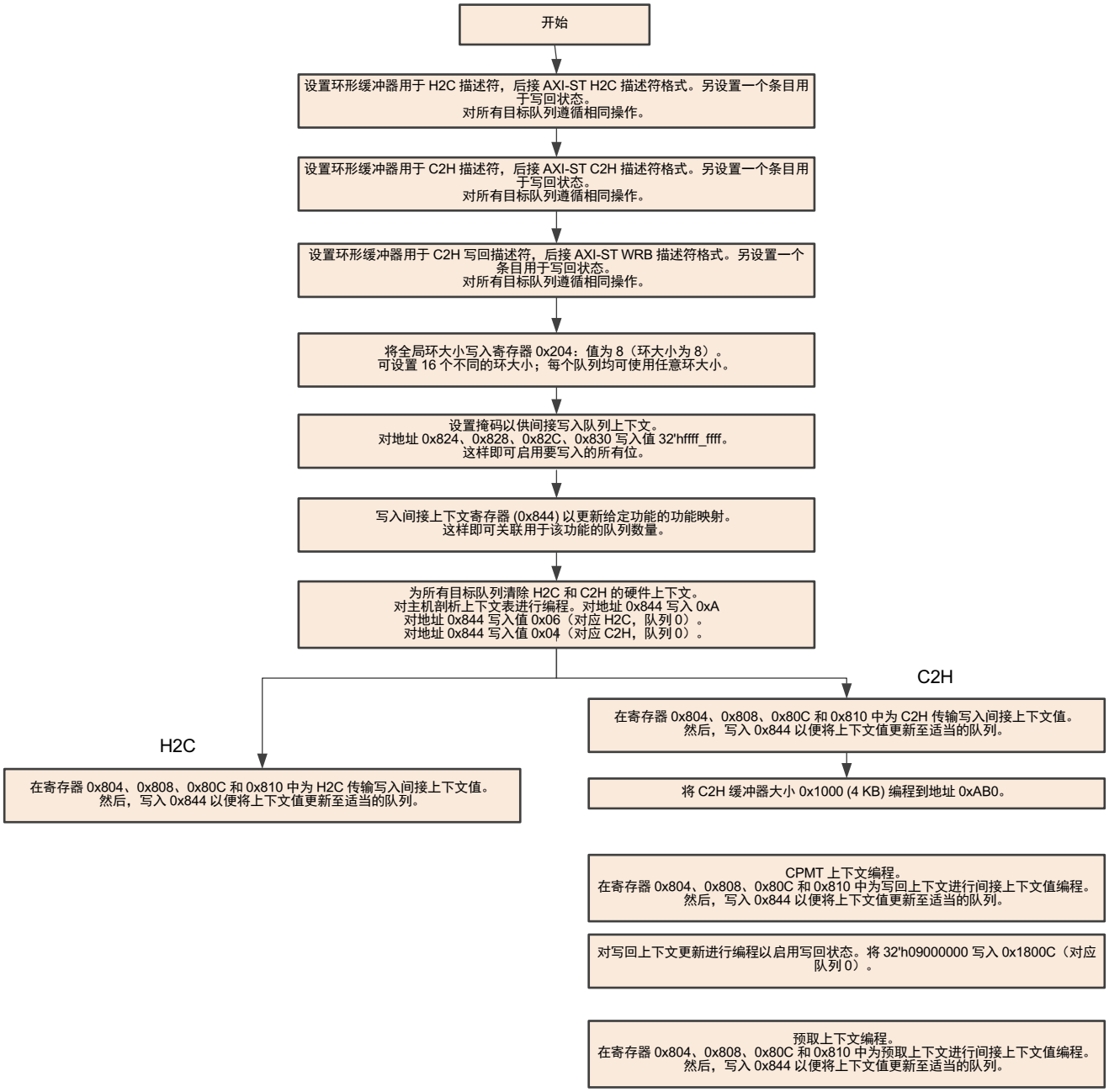
AXI4 存储器映射 H2C 流程

图 50：**AXI4 存储器映射主机到卡 (H2C) 流程图**



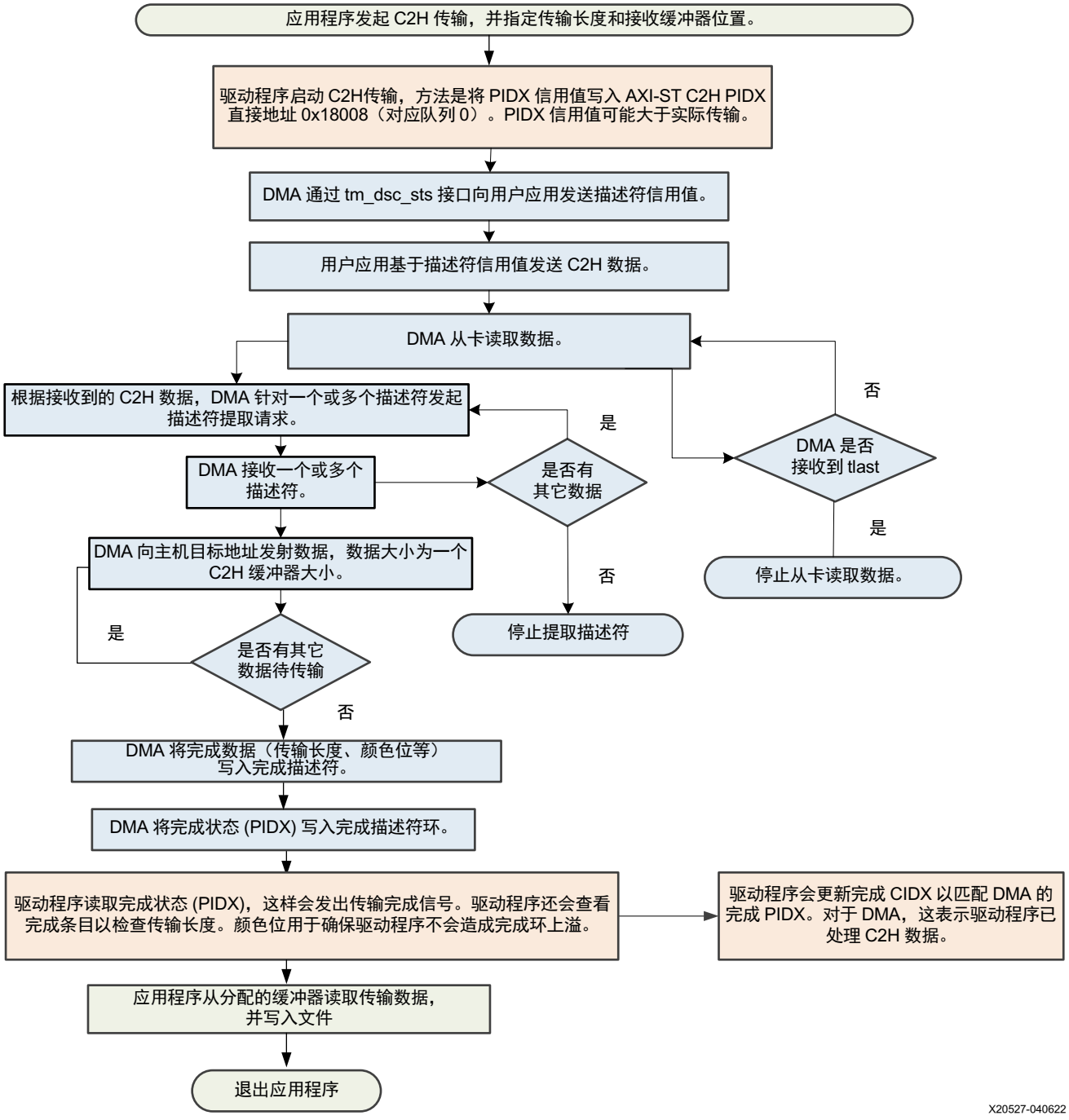
AXI4-Stream 流程图

图 51：**AXI4-Stream 流程图**



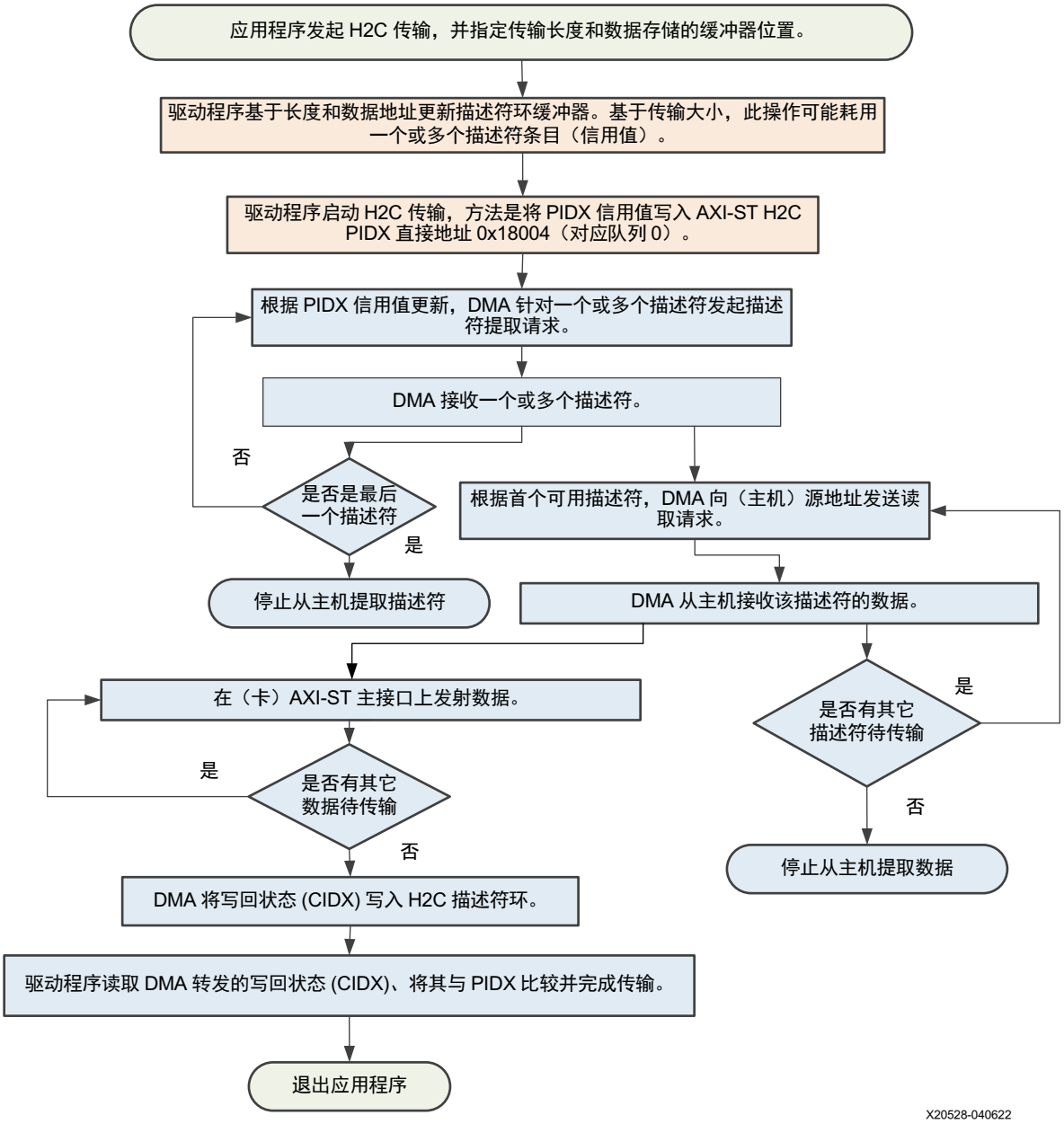
AXI4-Stream C2H 流程

图 52：**AXI4-Stream C2H 流程图**



AXI4-Stream H2C 流程

图 53：**AXI4-Stream H2C 流程图**



附录 E

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 Navigotor 页面。

参考资料

以下技术文档是非常实用的补充资料，可配合本产品指南一起使用：

1. 《Versal ACAP DMA and Bridge Subsystem for PCI Express 产品指南》(PG344)
2. 《AMBA AXI4-Stream 协议规范》(ARM IHI 0051A)
3. 《PCI-SIG 规范》(www.pcisig.com/specifications)
4. 《Virtex-7 FPGA Integrated Block for PCI Express LogiCORE IP 产品指南》(PG023)
5. 《7 系列 FPGA Integrated Block for PCI Express LogiCORE IP 产品指南》(PG054)
6. 《UltraScale 器件 Gen3 Integrated Block for PCI Express LogiCORE IP 产品指南》(PG156)

7. 《AXI Bridge for PCI Express Gen3 Subsystem 产品指南》(PG194)
8. 《DMA/Bridge Subsystem for PCI Express 产品指南》(PG195)
9. 《UltraScale+ Integrated Block for PCI Express LogiCORE IP 产品指南》(PG213)
10. 《Versal ACAP DMA and Bridge Subsystem for PCI Express 产品指南》(PG344)
11. 《Vivado Design Suite：AXI 参考指南》(UG1037)
12. 《Vivado Design Suite 用户指南：采用 IP integrator 设计 IP 子系统》(UG994)
13. 《Vivado Design Suite 用户指南：采用 IP 进行设计》(UG896)
14. 《Vivado Design Suite 用户指南：入门指南》(UG910)
15. 《Vivado Design Suite 用户指南：逻辑仿真》(UG900)
16. 《Vivado Design Suite 用户指南：使用约束》(UG903)
17. 《Vivado Design Suite 用户指南：编程和调试》(UG908)

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 10 月 27 日 5.0 版	
功能特性	不再支持 64 位 AXI 宽度。
FLR 端口	移除 `usr_flr_clr port`。
附录 B: GT 位置	新增附录。
AXI 串流性能设计示例	新增章节。
串联配置	已更新。
功能映射表	已更新。
上下文编程	已更新。
2022 年 5 月 20 日 4.0 版	
Slave Bridge	更新 BDF 表。
AXI4-Lite 主端口	更新“配置 AXI4-Lite 存储器映射读取主接口端口描述”表。
AXI4-Stream C2H 端口	更新“AXI4-Stream C2H 端口描述”表。
串联配置	更新新器件支持。
基于 PCIe 的 Dynamic Function eXchange	新增章节。
调试指南	新增章节。
2022 年 1 月 5 日 4.0 版	
常规更新	更新 PDF 关键字。
2021 年 12 月 17 日 4.0 版	
QDMA 全局端口	添加 csr_prog_done。
队列状态端口	更新 marker_cookie 端口。
2021 年 4 月 15 日 4.0 版	
Slave Bridge	添加有关 BDF 表和地址转换的更多详细信息，并提供示例。

章节	修订综述
串联配置 和 “Basic”选项卡	添加有关 Dynamic Function eXchange (DFX) over PCIe 支持的信息。
2020 年 7 月 1 日 4.0 版	
C2H 串流包类型	更新针对 QDMA 4.0 的标记响应（来自队列状态端口，而不是描述符旁路输出端口）。
主机配置文件	添加需编程的新主机配置文件上下文表。
寄存器空间	更新寄存器 CSV 文件。 更新寄存器地址。 添加有关公开所有调试寄存器的提示。
2020 年 6 月 10 日 4.0 版	
寄存器空间	对各章节进行重组。部分寄存器已更新。
QDMA_CSR (0x0000) 和 Bridge 寄存器空间	将寄存器描述移至产品指南外部的 CSV 文件。
描述符上下文 和 完成上下文结构	更新部分上下文表。
上下文编程	添加需编程的新主机配置文件上下文表。
端口描述	移除部分端口，并添加新端口。
自定义和生成子系统	更新有关 Vivado 2020.1 的选项和描述。
“PCIe BARs”选项卡	将 PF 中 QDMA BAR 大小增加到 256 KB，将 VF 中的 BAR 大小增加到 32 KB。
“Debug and Additional Options”选项卡	新增。
附录 A: 升级	添加 AR 引用，描述不同核版本之间的更改。
2019 年 11 月 22 日 3.0 版	
RTL 版本寄存器 (0x22414)	在文档中添加 PF RTL 版本寄存器
RTL 版本寄存器 (0x5014)	在文档中添加 VF RTL 版本寄存器
AXI4-Stream 状态端口	添加 axis_c2h_status_error 端口。从 2019.2 补丁版本开始，此端口可供使用。
QDMA C2H 描述符旁路输出标记响应描述表	为标记响应添加 C2H 串流 marker_cookie 字段。从 2019.2 补丁版本开始，此功能特性可供使用。
QDNA_GLBL2_MISC_CAP (0x134)	更新可用的位和描述。
VDM	添加有关不支持连续 VDM 访问的信息。
2019 年 5 月 22 日 3.0 版	
性能和资源使用情况	添加性能详细信息和性能报告答复记录。
器件最低要求	为 QDMA 启用 Gen4 器件。
用户参数	添加指向答复记录的链接，该答复记录中添加了额外的核自定义选项。
“Capabilities”选项卡	邮箱可单独选中，与 SR-IOV 选择无关。
AXI 串流环回设计示例	添加新设计示例。
2018 年 12 月 5 日 3.0 版	
IP 相关信息 和 使用驱动程序	添加 Windows 驱动程序支持。
寄存器空间	添加寄存器，并更新寄存器。
“PCIe MISC”选项卡 和 “PCIe DMA”选项卡	更新 2018.3 版本。
第 6 章：设计示例	添加两个设计示例，并更新寄存器。
附录 A: 升级	添加 AR 引用，描述不同核版本之间的更改。

章节	修订综述
2018 年 9 月 4 日 2.0 版	
端口描述	对于 tm_dsc_sts_rdy（VDM 端口）和 st_rx_msg_rdy（QDMA 流量管理器信用值输出端口），强调不使用此接口时，Ready 必须绑定到 1。
寄存器空间	添加寄存器，以便在未完成的数据量超出编程的阈值时，停止来自 H2C 串流引擎的读取请求。
	添加新的 C2H 完成中断触发模式，其中包括用户触发器、定时器到期或计数超阈值
2018 年 6 月 22 日 2.0 版	
“概述”章节	更新全文内容。
“端口描述”章节	更改部分表格内容，并对内容进行了重组。
“寄存器空间”章节	添加“存储器映射寄存器空间和 AXI4-Lite 从接口寄存器空间”章节。
“上下文结构定义”章节和“队列条目结构”章节	移除这些章节，并将其中内容移至“概述”章节中的“QDMA 操作”章节。
“设计流程步骤”章节	更新“Basic”选项卡、“Capabilities”选项卡、“PCIe BARs”选项卡、“PCIe Misc”选项卡和“PCIe DMA”选项卡的描述。
“设计示例”章节	添加两个新的设计示例，并添加设计示例寄存器。
2018 年 4 月 17 日 1.0 版	
初始赛灵思版本。	

请阅读：重要法律声明

版权声明