【PG054】7 Series Integrated Block for PCI Express IP核的学习

PCIe学习笔记系列：

1. PCIe基础知识及Xilinx相关IP核介绍
   概念了解：简单学习PCIe的数据链路与拓扑结构，另外看看有什么相关的IP核。
2. 【PG054】7 Series Integrated Block for PCI Express IP核的学习
   基础学习：关于Pcie IP核的数据手册，学习PCIe相关的IP核的配置参数及其对应的含义。
3. Xilinx PCIe IP核示例工程代码分析与仿真
   基础学习：关于PCIe IP核的仿真，学习PCIe的配置流程以及应用过程。
4. Xilinx XDMA 例程代码分析与仿真结果
   应用学习：关于Xilinx PCIe DMA IP核的仿真，学习 PCIe DMA 的配置过程以及具体的数据传输流程。
5. XDMA linux平台调试过程记录
   应用学习：关于XDMA的实际调试过程，可在此基础上定制自己的需求。

1 IP核说明

结构：

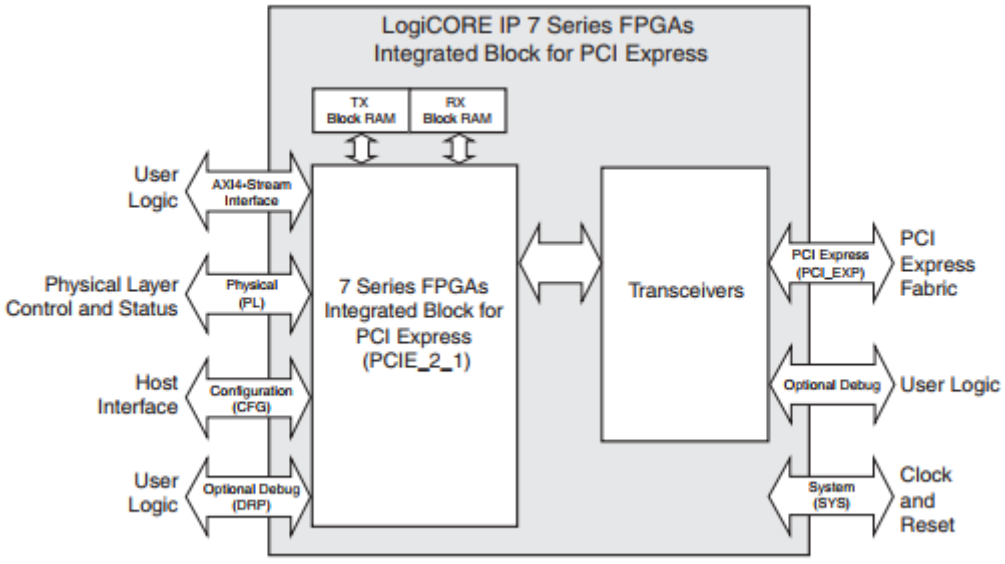


Figure 2-1: **Top-Level Functional Blocks and Interfaces**

主要的接口：

- 系统接口（SYS）。
- PCIe接口（PCI_EXP）。
- 配置接口（CFG）。
- 事务接口（AXI-Stream）。

- 物理层控制与状态接口（PL）。

该模块以数据包的形式与各模块进行通信。从发送到接收组件，携带信息（指的是用户的信息）的数据包在事务层以及数据链路层形成。在发送的过程中加了一些其他的必要的信息（各层处理数据包需要的信息）。在接收端，接收单元的每一层处理进入的包，剥离相关信息并将包转发给下一层。

1.1 遵循的标准

《PCI Express Base Specification, rev. 2.1 》
《PCI Express Card Electromechanical (CEM) v2.0》
《PCI Industrial Computer Manufacturers Group (PICMG) 3.4 specifications》

1.3 IP核的接口

2 设计例程

2.1 集成块Endpoint配置概览

PCI Express设备需要在上电后进行设置，然后系统中的设备才能开始与其他设备进行特定于应用程序的通信。通过PCI Express Link连接的至少两个设备必须初始化其配置空间并被枚举以进行通信。

Root Port通过给【像Endpoint或Switch这样的下游设备】发送【Configuration Read(CfgRd)和Write(CfgWr) TLP 并进行配置空间的设置】来实现【PCI Express枚举和配置】。当此过程完成时，才可以在PCI Express系统内发生更高级别的交互，如内存读取(MemRd TLPs)和写入(MemWr TLPs) 。

配置器示例设计 完成 枚举和配置单个连接的PCIe Endpoint配置空间 所需的配置事务，允许应用程序特定的交互发生。

Endpoint配置的示例的仿真设计包含两个部分：

- Root Port模型：生成、使用和检查PCI Express总线数据流的testbench。
- 可编程的输入输出（Programmed Input/Output ，PIO）示例设计：一个完整的PCI Express应用。PIO示例设计响应对其内存空间的读和写请求，可以在硬件测试中是可综合的。

仿真设计概览

在仿真设计中，事务从Root Port模型中发送到被配置为Endpoint的IP核中，示例设计的结构：

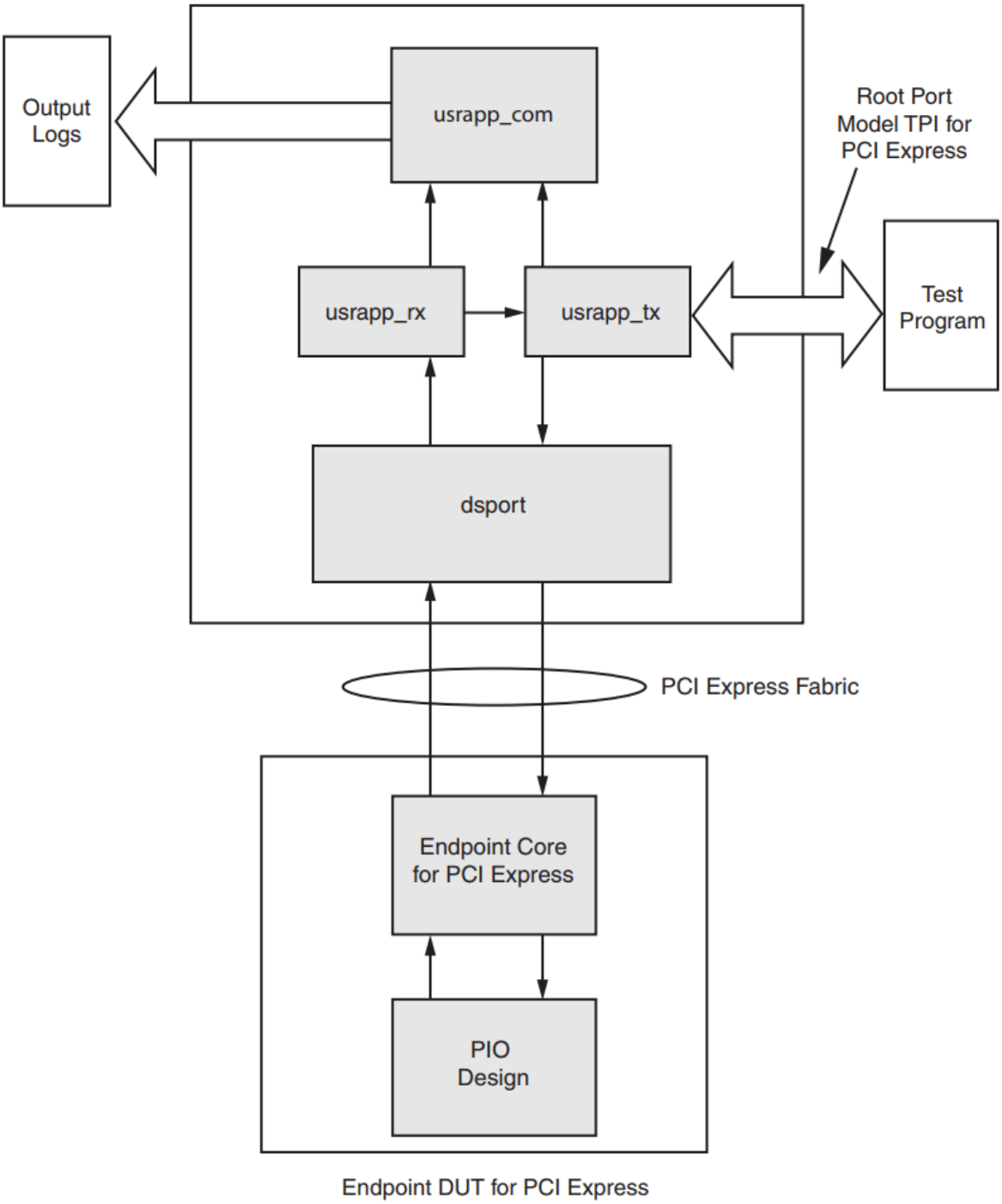
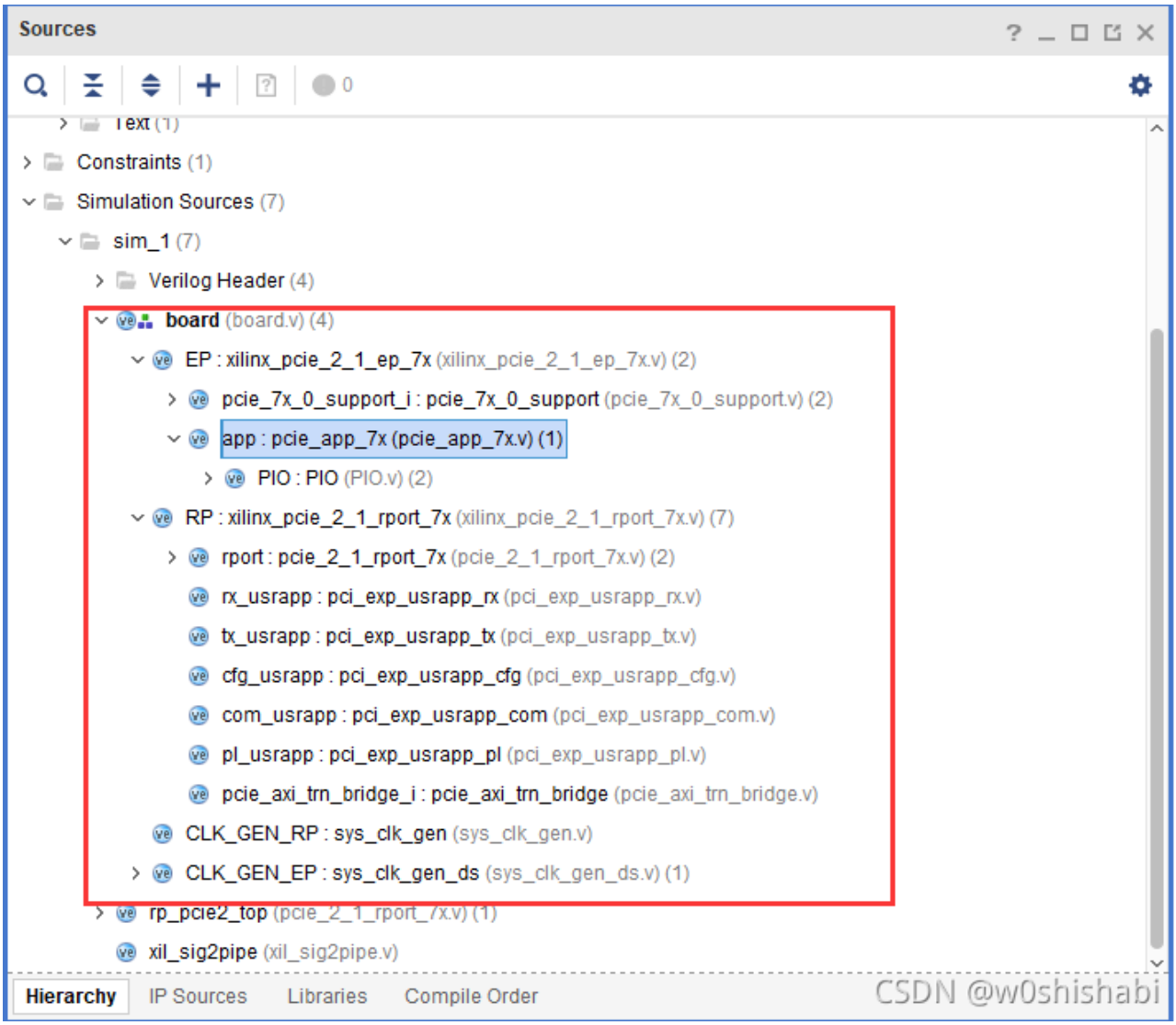


Figure 5-1: **Simulation Example Design Block Diagram**

对应的文件结构：



2.2 Programmed Input/Output: Endpoint 示例设计

Programmed Input/Output (PIO) 事务一般由PCI Express系统host CPU来访问内存映射输入/输出(Memory Mapped Input/Output，MMIO)和配置映射PCI Express逻辑中的输入/输出(Configuration Mapped Input/Output，CMIO)位置。PCI Express的Endpoint通过【完成数据事务】来 【接受内存和I/O写事务】 和 【响应内存和I/O读事务】 。

一般PIO是包含在IP核中的，但在7 Series FPGAs Integrated Block for PCI Express 、Endpoint Block Plus for PCI Express、Endpoint PIPE for PCI Express IP核中共享PIO设计端口模型，也就是pcie_app_7x中的PIO模块。可以在实际的板子中使用PIO设计来验证板的链路和功能。

2.2.1 系统概览

PIO设计是一个简单的应用程序，它连接【PCIe核心事务(AXI4-Stream)接口】与【Endpoint】，也可以作为构建用户设计的起点。特点包括:

- 使用Xilinx FPGA内部的block RAM 实现 四个特定事务的2 KB目标区域，提供8192个字节的总目标空间。

- 支持【单DWORD载荷】的到32/64位地址的内存空间或I/O空间的PCIe读写事务，并支持完成事务层包(TLPs)。
- 利用IP核的(rx_bar_hit[7:0]) m_axis_rx_tuser[9:2]信号来区分TLP目的基地地址寄存器。
- 提供针对单独实现32位、64位和128位优化的AXI4-Stream接口。

PCI Express系统架构组件如下图，包括一个Root Complex，一个PCIe switch器件，一个PCIe Endpoint。【PIO操作】将数据从 Root Complex(CPU寄存器)向下移动（downstream）到Endpoint，或者从端点向上移动（upstream）到 Root Complex((CPU寄存器)。无论哪种情况，移动数据的PCI Express协议请求都是由主机CPU发起的。

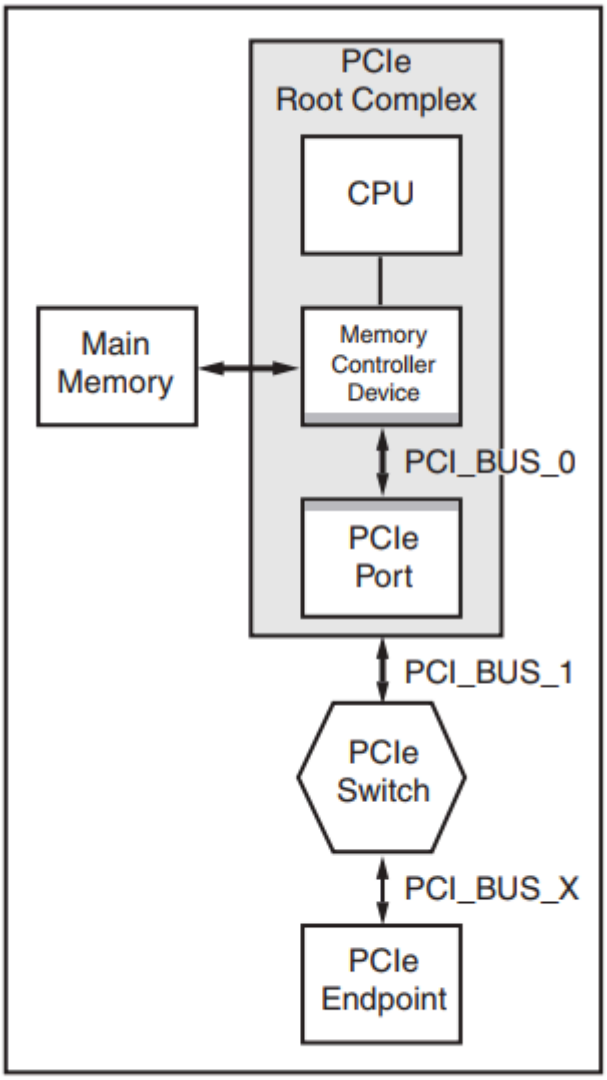


Figure 5-3: **PCI Express System Overview**

当CPU发出向【内存映射输入/输出（MMIO）地址】写寄存器 命令时，数据就下行移动。Root Complex通常生成一个Memory Write TLP，其中含MMIO单元的地址、字节使能和寄存器内容。当Endpoint接收到Memory Write TLP并更新相应的本地寄存器时，事务终止。

当CPU发出从一个MMIO地址读寄存器的命令时，数据就上行移动。Root Complex通常生成一个Memory Read TLP，其中包含MMIO单元的地址，以及字节使能。Endpoint 在收

到 Memory Read TLP 后产生一个 Completion with Data TLP 。 Completion被发送到Root Complex，有效负载被加载到目标寄存器中，完成事务。

2.2.2 PIO硬件

PIO设计在FPGA block RAM中实现了一个8192字节的目标空间，在PCIe Endpoint 之后。这个32位目标空间可以通过single DWORD I/O Read,I/O Write, Memory Read 64, Memory Write 64, Memory Read 32, 以及 Memory Write 32 TLP 进行访问。

PIO设计生成一个有效负载为一个DWORD的Completion，用来响应有效的Memory Read 32 TLP, Memory Read 64 TLP, or I/O Read TLP 请求。此外，PIO设计为I/O Write TLP请求返回一个没有数据且状态成功的ComCpletion。

PIO设计通过更新载荷到FPGA block RAM空间的目标地址来处理一个【有一个DWORD载荷的 Memory 或 I/O Write TLP】。

2.2.3 支持基地址寄存器（Base Address Register，BAR）

PIO设计支持4个不连续的目标空间，每个空间由一个独立的Base Address Register(BAR)表示的2KB内存块组成。使用默认参数时，Vivado IDE生成一个配置为配合PIO设计使用的IP核，它包括:

- 一个64位可寻址内存空间BAR
- 一个32位可寻址内存空间BAR

可以改变PIO设计使用的默认参数，但是，在某些情况下，可能需要根据系统更改用户应用程序。

BAR表示的四个2KB地址空间与PIO设计中的四个2kb地址区域对应。每个2kB区域都使用一个2KB 双端口 block RAM 来实现。当IP核收到事务时，IP核解码出地址以及确定四个区域中哪个时目标区域。IP核将TLP传递到PIO设计并断言相应的bitm_axis_rx_tuser[9:2]（rx_bar_hit[7:0]）。这些位的定义如下：

Block RAM	TLP事务类型	默认 BAR	rx_bar_hit[7:0]
ep_mem0	I/O TLP 事务	禁用	禁用
ep_mem1	32-bit address Memory TLP事务	0	0000_0001b
ep_mem2	64-bit address Memory TLP事务	禁用	禁用
ep_mem3	指定EROM（Expansion ROM）的32-bit address Memory TLP事务	扩展 ROM	0100_0000b

2.2.3.1 改变默认的BAR设置

可以改变参数然后继续使用PIO设计来创建自定义的HDL源文件来匹配选择的BAR设置。然而，由于PIO设计参数比IP核参数限制更大，在更改默认参数时，需要考虑以下示例设计限制:

- 示例设计支持一个I/O空间BAR，一个32位内存空间（不是外部ROM空间），以及一个64位内存空间。如果超出了这些限制，则只有给定类型的第一个空间是有效的，这意味这对其他空间的访问不会导致completion。
- 每个空间都有2KB内存的大小限制。如果相应的BAR配置为更大的大小，超过2kb限制的访问将与2kb内存空间交叠。
- PIO设计支持一个I/O空间BAR，默认情况下是禁用的，但如果需要可以更改。
-

尽管PIO设计有局限性，但提供了Verilog或VHDL源代码，以便可以根据自己的具体需求定制示例设计。

2.2.3.2 TLP数据流

本节定义了由PIO设计成功处理的TLP的数据流。有关PIO设计子块内的接口信号的详细信息，请参见接收路径和发送路径。

PIO 成功处理**单个DWORD**有效负载内存读写TLP和I/O读写TLP。内存读或内存写TLP长度**大于一个DWORD**不能被PIO设计正确处理。不过，IP核 会 接收这些TLP并传递给PIO。如果PIO接收到大于**单个DWORD**的TLP，丢弃并且不会产生相应的completion。

2.2.3.3 Memory and I/O Write TLP 的处理

当PCIe的Endpoint收到一个Memory或I/O Write TLP时，TLP的目的地址和事务类型会与IP核的BAR中的值进行比较。如果TLP通过了这个比较检查，核心将TLP传递到 PIO设计 的接收AXI4-Stream接口。PIO设计 以不同的方式处理内存写和I/O TLP写: PIO设计 通过生成无数据完成(cpl)响应I/O写，这是PCI Express规范的要求。

与包的开始、包的结束和准备握手信号一样，接收AXI4-Stream接口也断言(rx_bar_hit[7:0]) m_axis_rx_tuser[9:2] 以指示 PIO设计 传入 与TLP匹配的 特定目的BAR。在接收时，PIO设计 的RX状态机处理传入的Write TLP，并提取TLP数据和相关地址字段，以便将这些数据传递给PIO设计的内部block RAM写请求控制器。

根据断言的特定的`rx_bar_hit[7:0]`信号，RX状态机在断言写使能请求之前 指示【内部写控制器】要使用的适当的2KB block RAM。比如，如果IP核收到一个指向BAR0 的 Memory Write 32 Request，IP核将TLP传给PIO设计然后断言`rx_bar_hit[0]`。RX状态机从Memory Write 32 Request TLP中提取最低地址位以及的数据段，并指示内部内存写控制器开始对块进行写操作RAM。

在示例设计中，`rx_bar_hit[0]`的断言指示PIO内存写控制器访问`ep_mem1`(默认情况下，它代表2kb的Mem32空间)。当写入到FPGA block RAM时，PIO设计 的RX状态机取消断言m_axis_rx_tready，导致接收AXI4-Stream接口停止继续接收TLP，直到内部内存写控制器完成对block RAM的写操作。 不是所有使用IP核的设计都需要以这种方式取消断言`m_axis_rx_tready`，PIO设计采用这种方法简化了RX状态机的控制逻辑。

2.2.3.4 Memory and I/O Read TLP 的处理

当PCIe Endpoint 收到 Memory 或 I/O Read TLP，将TLP目的地址和事务类型与BARs中预设好的数据进行对比。如果TLP通过了这个比较，IP核将此TLP传递PIO设计的接收 AXI4-Stream 接口。

与包的开头，结尾以及 ready handshaking 信号一样，接收 AXI4-Stream 接口也断言对应的 rx_bar_hit[7:0] 信号，用来给PIO设计指明与输入的TLP匹配的目的BAR。在接收时，PIO设计的状态机处理接收到的Read TLP并提取相关信息TLP信息并将其传递给内部block RAM的读请求控制器。

根据断言的特定的 `rx_bar_hit[7:0]` 信号，RX状态机在读使能之前 指示【内部写请求控制器】使用正确的2KB block RAM。比如，如果一个IP核收到一个指向默认的MEM32 BAR2的Memory Read 32 Request TLP，IP核将TLP传给PIO设计并且断言`rx_bar_hit[0]`。RX状态机从Memory 32 Read TLP提取最低的地址位，并指示内部内存读请求控制器启动一个读操作。

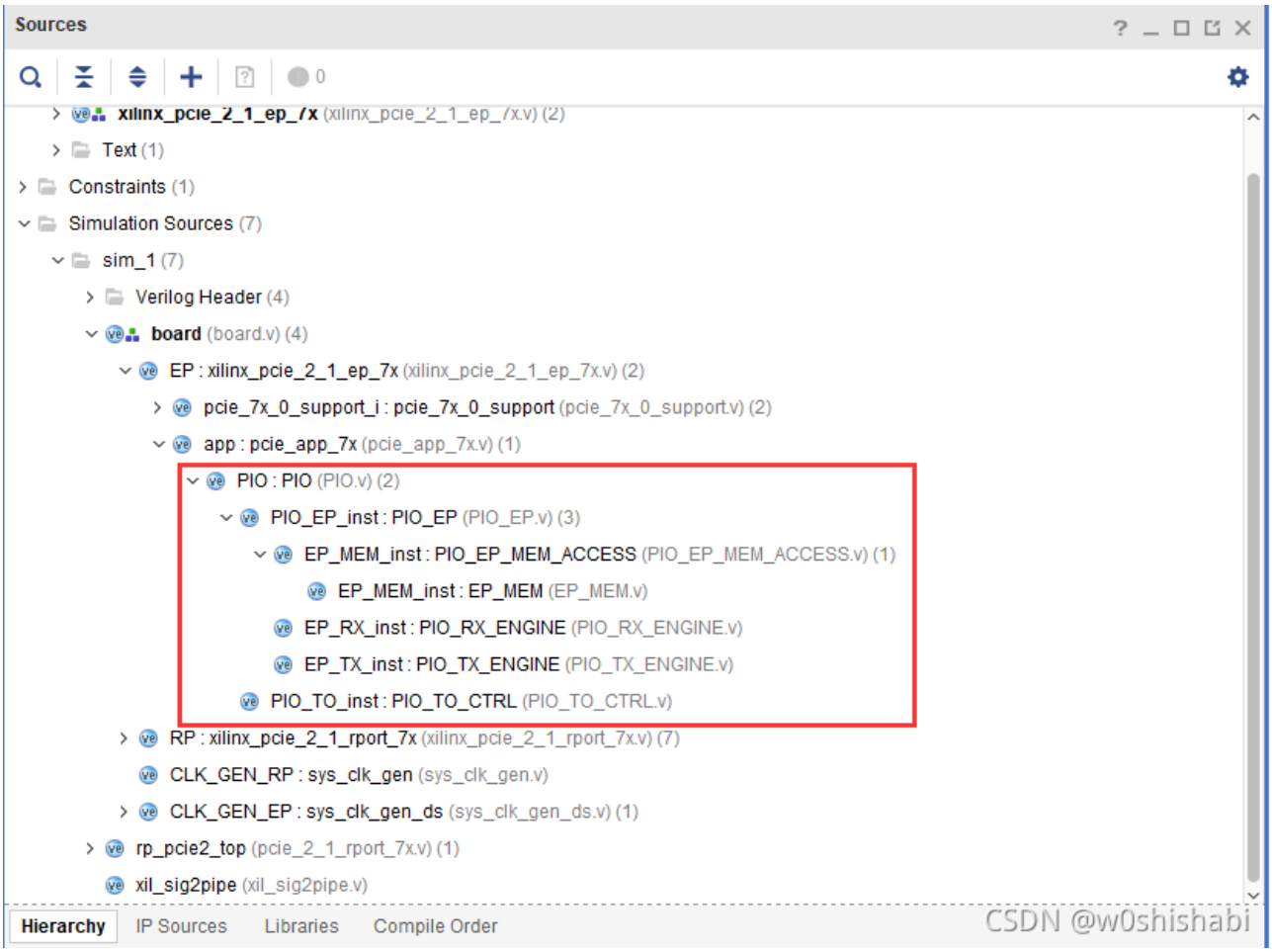
在示例设计中，`rx_bar_hit[0]`的断言指示PIO内存读控制器访问Mem32空间，该空间默认代表2kb的内存空间。处理内存写TLP和读TLP的一个显著区别是，**在内存或I/O读请求的情况下，接收设备需要返回一个Completion with Data TLP。**

当读取正在处理时，PIO设计RX状态机取消断言 m_axis_rx_tready，导致接收AXI4-Stream接口停止接收TLP，直到内部内存读控制器完成对block RAM的读访问并生成completion。不是所有使用IP核的设计都需要以这种方式取消断言 `m_axis_rx_tready`，PIO设计采用这种方法简化了RX状态机的控制逻辑。

2.2.3.5 PIO文件结构

于特定的核心目标，并非所有由Vivado IDE生成的文件都是必需的，有些文件可能无法产生。主要的区别在于，一些PCIe解决方案的端点使用64位用户数据路径，而另一些则使用128位的数据路径，PIO设计对这两种路径都适用。数据路径的宽度取决于所针对的特定核心。

文件	描述
pcie_app_7vx.v	顶层设计封装
PIO.v	PIO设计封装
PIO_EP.v	PIO应用模块
PIO_TO_CTRL.v	PIO关断控制
PIO_RX_ENGINE.v	64位/128位接收引擎
PIO_TX_ENGINE.v	64位/128位发送引擎
PIO_EP_MEM_ACCESS.v	Endpoint内存访问模块
PIO_EP_MEM.v	Endpoint内存



PIO提供3个配置：PIO_64,PIO_128分别对应64位和128位 AXI4-Stream 接口。生成的PIO配置取决于所选的Endpoint类型(即7系列fpga集成块)以及PCI Express通道的数量和所选的接口宽度。

IP核	**x1**	**x2**	**x4**	**x8**
7 Series FPGAs Integrated Block	PIO_64	PIO_64	PIO_64,PIO_128	PIO_64,PIO_128

PIO设计的各种组件，它被分为四个主要部分:TX引擎(TX Engine)，RX引擎(RX Engine)，内存访问控制器(Memory Access Controller)和电源管理断开控制器(Power Management Turn-Off Controller)。

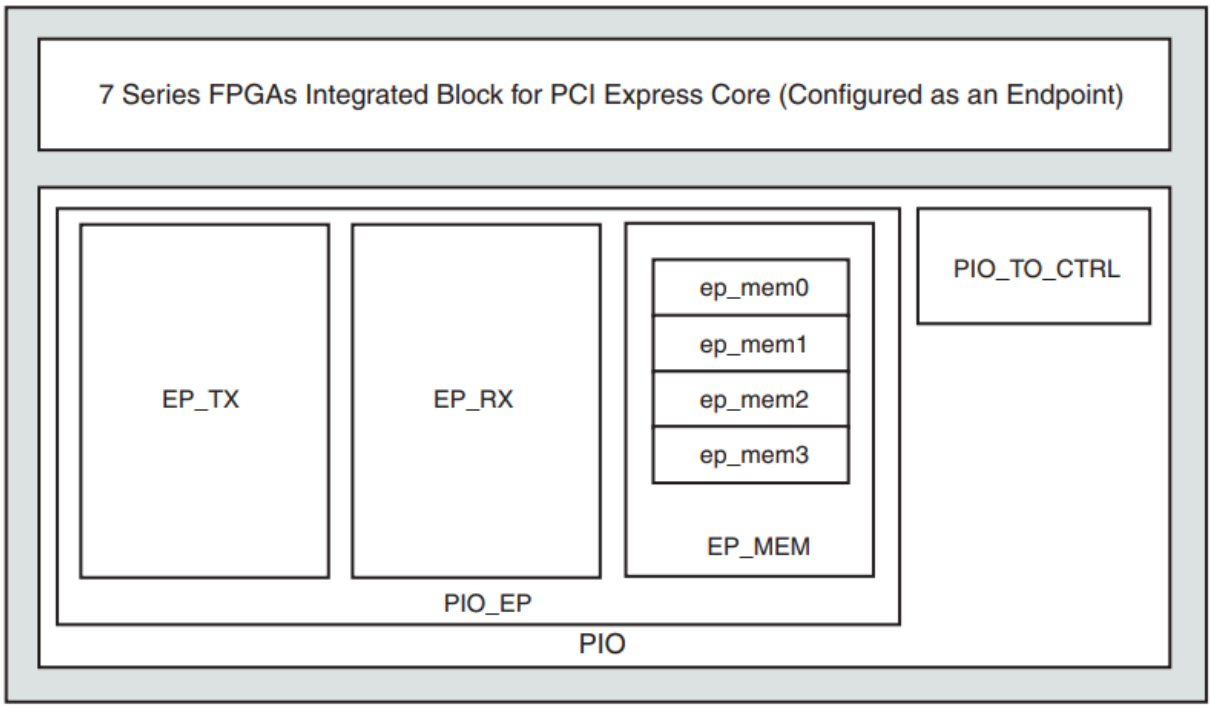


Figure 5-4: **PIO Design Components**

2.2.4 PIO应用

128位和64位PIO应用的顶层连接如下图。数据位宽（32，64或128位）取决于使用的PCIe Endpoint的类型。PIO_EP 模块包含了PIO FPGA block RAM模块和收发引擎。PIO_TO_CTRL 模块是Endpoint关闭控制器单元，它响应来自主机CPU的电源关闭消息并发出一个确认信号。

PIO_EP 模块连接到 Endpoint AXI4-Stream 和 配置（cfg）接口。

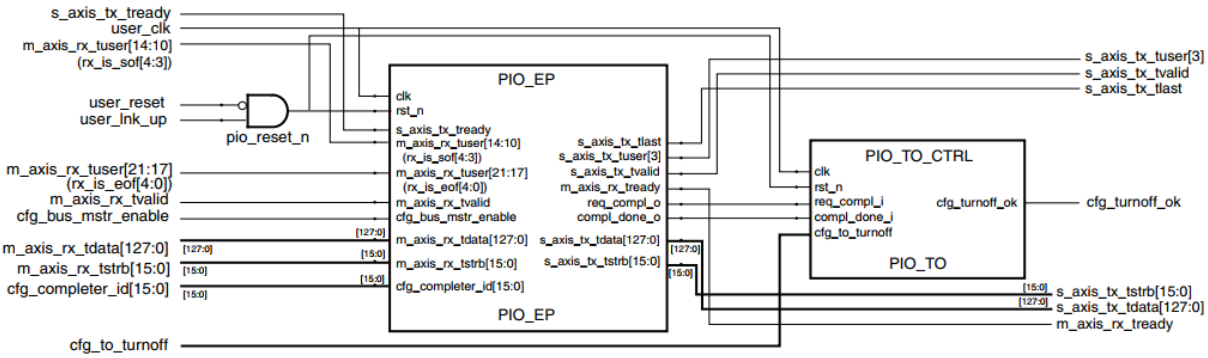


Figure 5-5: **PIO 128-Bit Application**

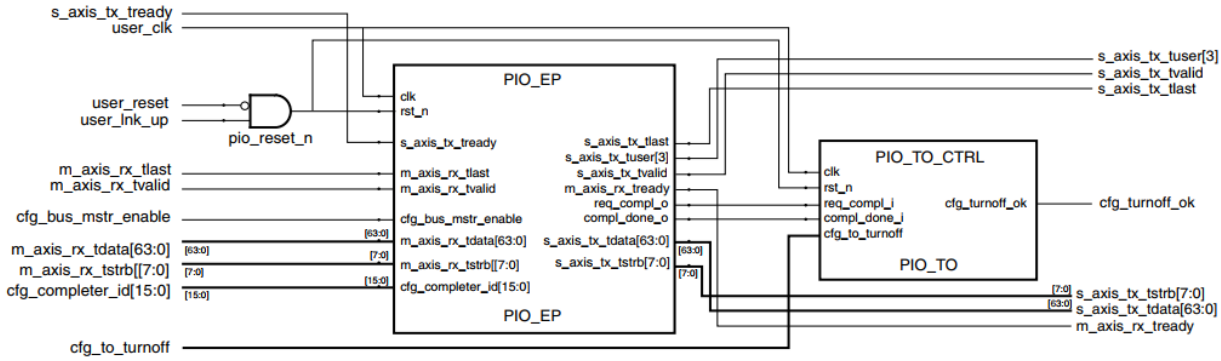


Figure 5-6: **PIO 64-Bit Application**

2.2.4.1 接收路径

PIO_RX_ENGINE 模块。模块的数据路径必须与正在使用的IP核的数据路径匹配。这些模块连接到Endpoint的PCIe接收接口。

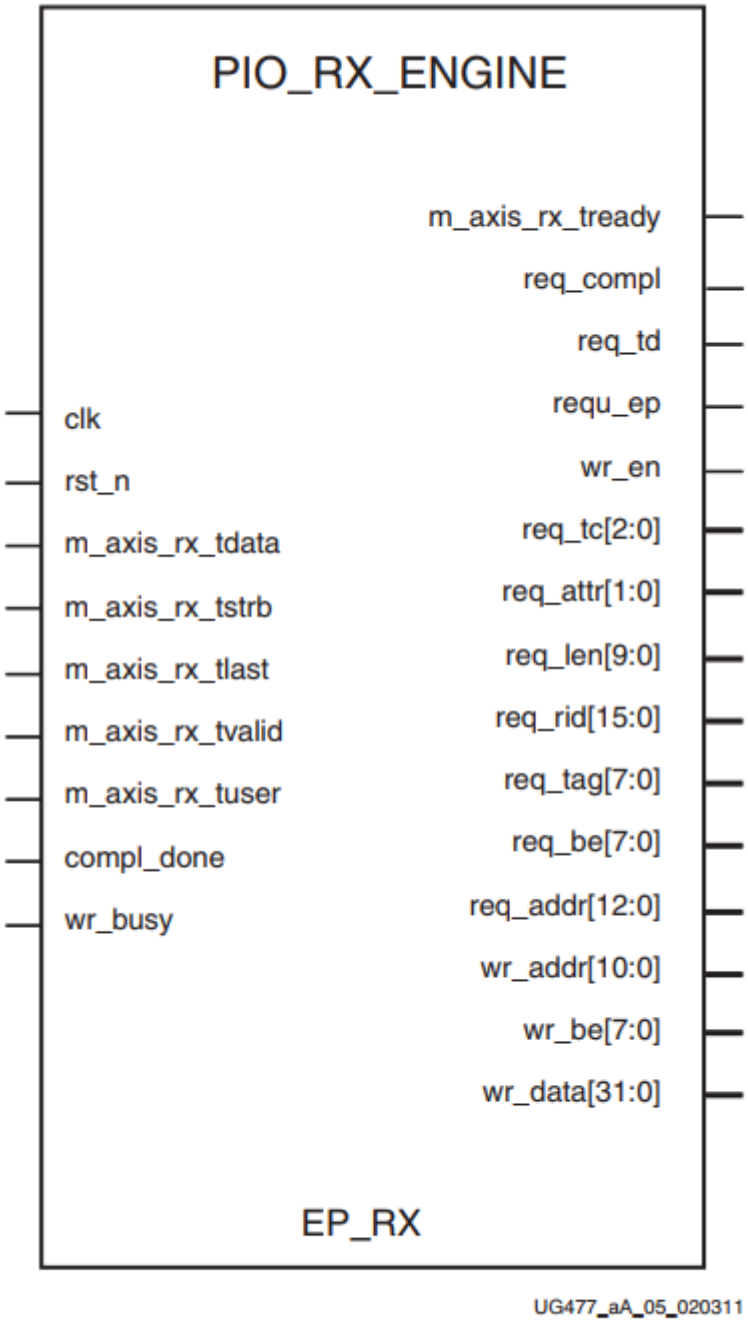


Figure 5-7: **RX Engine**

PIO_RX_ENGINE 模块接收并解析传入的读写TLP。

RX引擎解析一个DWORD 32和64位可寻址内存和I/O读请求。RX状态机从TLP中提取所需的信息，并将其传递给内存控制器。

RX Engine: Read Outputs端口：

端口	描述
req_compl	Completion请求
req_compl_wd	带数据的Completion请求

端口	描述
req_td	请求TLP的 Digest位
req_ep	请求TLP的Error Poisoning位
req_tc[2:0]	请求Traffic Class
req_attr[1:0]	请求Attributes
req_len[9:0]	请求Length
req_rid[15:0]	请求Requester Identifier
req_tag[7:0]	请求 Tag
req_be[7:0]	请求Byte Enable
req_addr[12:0]	请求Address

RX引擎解析一个DWORD 32和64位可寻址内存和I/O写请求。RX状态机从TLP中提取所需的信息，并将其传递给内存控制器。

RX Engine: write Outputs端口：

端口	描述
wr_en	写使能
wr_addr[10:0]	写地址
wr_be[7:0]	写字节使能
wr_data[31:0]	写数据

当应用程序正在处理当前TLP时，读数据路径停止接受来自IP核的新事务。这是通过断言 `m_axis_rx_tready` 完成的。对于正在进行的内存或I/O读事务，模块在接受下一个TLP之前等待`compl_done_i`输入被断言，而正在进行的内存或I/O写事务在`wr_busy_i`取消断言之后 视为完成。

2.2.4.2 发送路径

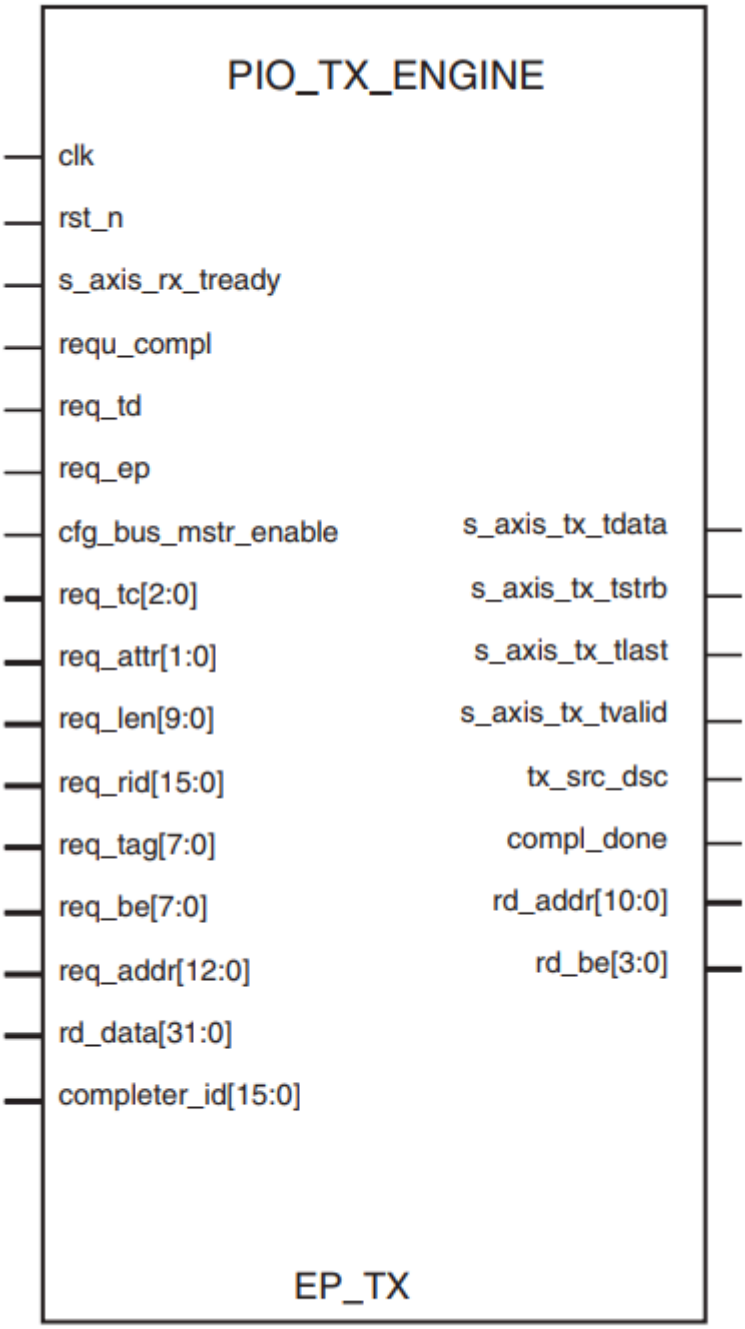


Figure 5-8: **TX Engine**

在发送完成后，TX引擎断言`compl_done_i`输出，指示RX引擎可以断言`m_axis_rx_tready`并继续接收TLP。

2.2.4.3 Endpoint 内存

`PIO_EP_MEM_ACCESS`模块包含Endpoint内存空间

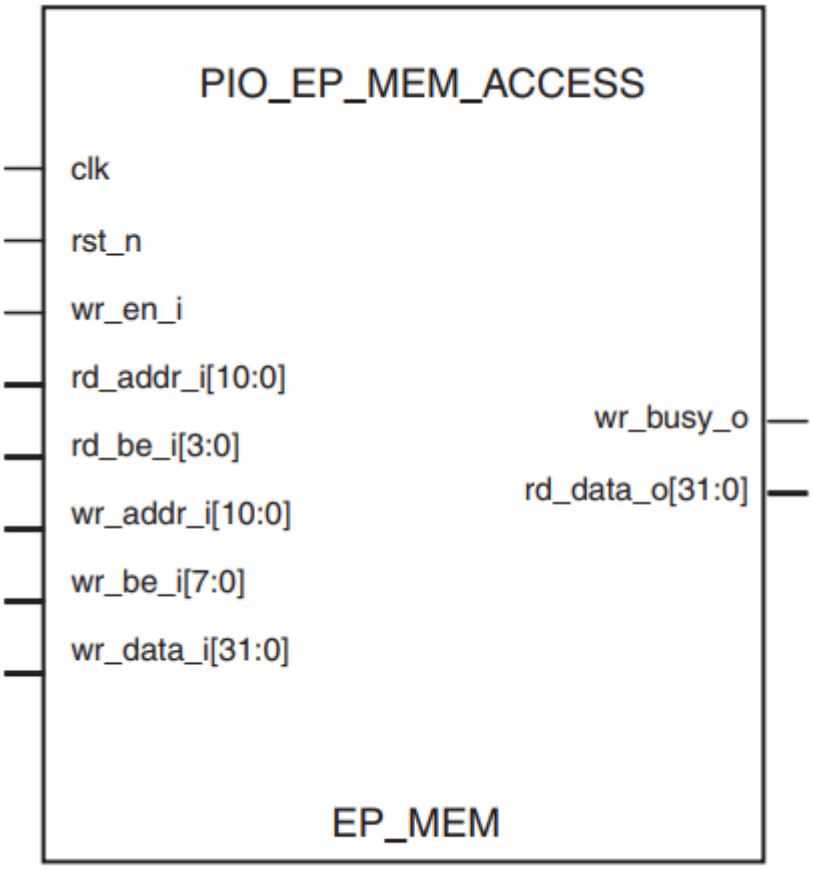


Figure 5-9: **EP Memory Access**

PIO_EP_MEM_ACCESS模块将【内存或I/O 写 TLP】 的数据写到内存，并提供从内存读取的数据作为【Memory和I/O读 TLP】的响应。

EP_MEM模块根据从RX引擎接收到的信息处理1个DWORD 32位和64位可寻址【内存和I/O写请求】。当内存控制器在处理写操作时，它断言wr_busy_o输出，表明正忙。

EP Memory: Write Inputs端口：

端口	描述
wr_en_i	写使能
wr_addr_i[10:0]	写地址
wr_be_i[7:0]	写字节使能
wr_data_i[31:0]	写数据

处理完读请求后，在rd_data_o[31:0]上返回数据。
EP Memory: read Inputs端口：

端口	描述
req_be_i[7:0]	请求 字节使能

端口	描述
req_addr_i[31:0]	请求 地址

2.2.5 PIO操作

2.2.5.1 PIO读事务

PIO设计 的连续内存读请求（Back-to-Back Memory Read request）。接收引擎一旦完全接收到第一个TLP，就取消对m_axis_rx_tready的断言。下一个读事务只有在传输引擎断言compl_done_o之后才会被接受，这表明第一个请求的Completion已经成功传输。

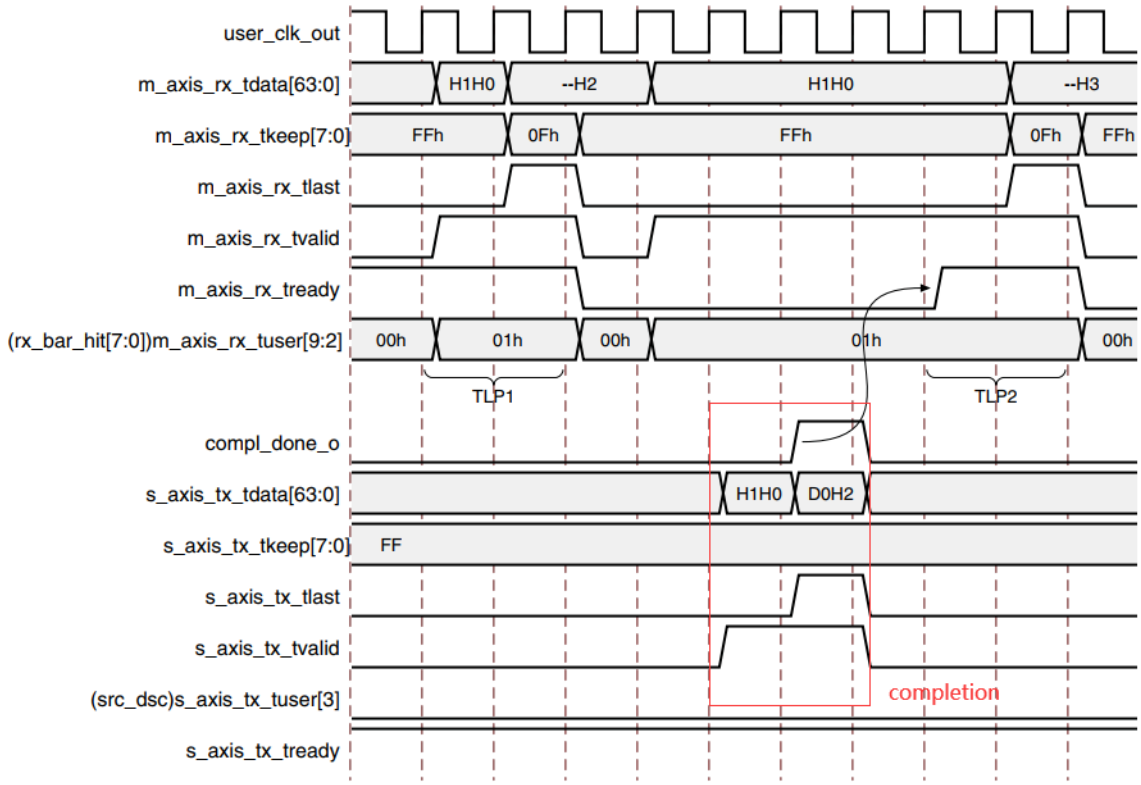


Figure 5-10: **Back-to-Back Read Transactions**

2.2.5.2 PIO写事务

PIO设计 的连续内存写请求（Back-to-Back Memory Write request）。只有当wr_busy_o被内存访问单元取消断言后，才会接受下一个写事务，这表明与第一个请求相关的数据已经成功写入内存。

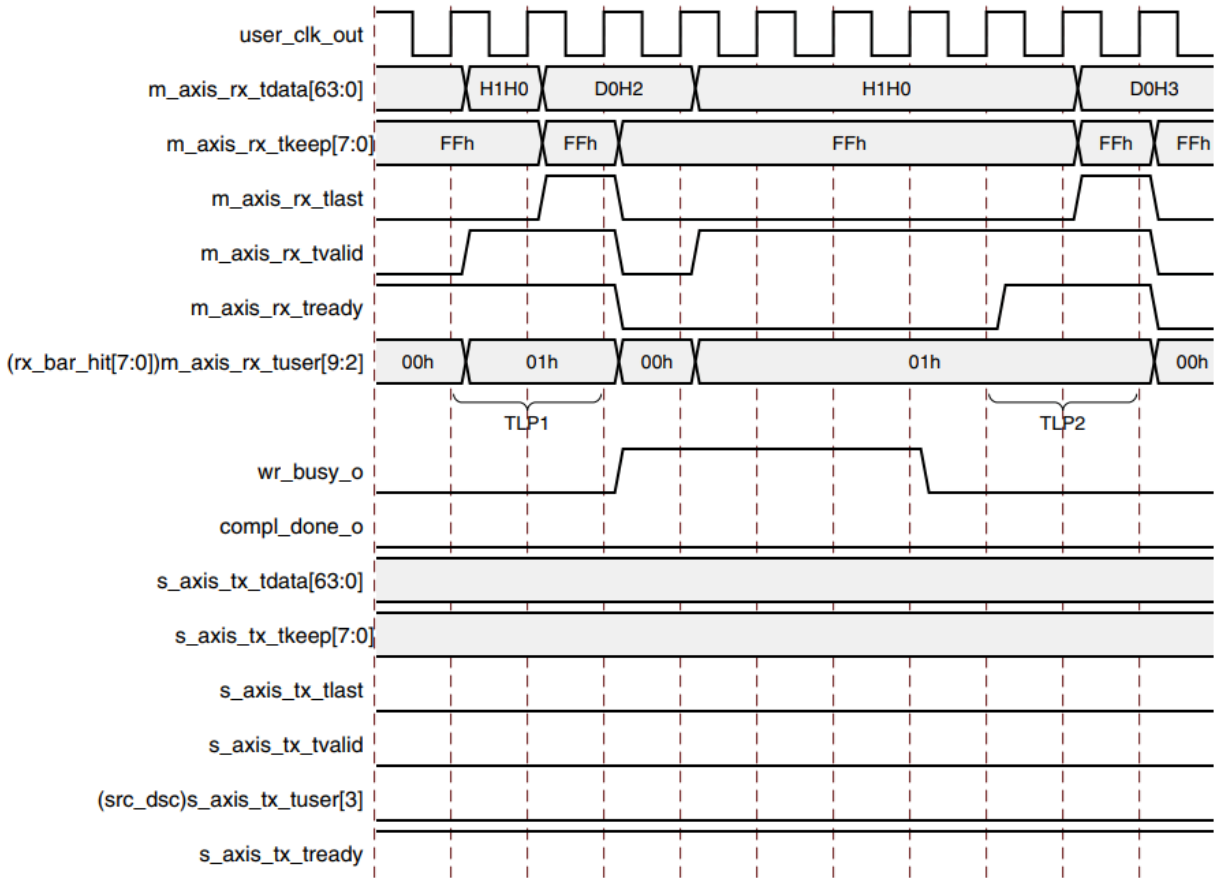


Figure 5-11: **Back-to-Back Write Transactions**

2.5 仿真示例设计

Endpoint配置

该仿真环境提供了7系列fpga的PCI Express 集成块IP核 执行 简单的内存访问测试 的 PIO 设计示例。事务由Root Port Model生成，并由PIO示例设计进行响应。

- PCIe事务层包（TLP）由testbench的发送程序（pci_exp_usrapp_tx）产生，在发送TLP的同时，也会产生相应的log文件（tx.dat）。
- PCIe TLP由teshbench的接收程序（pci_exp_usrapp_rx）接收，在接收TLP的同时，也会产生相应的Log文件（rx.dat）。

3 Endpoint 的 Root Port Model Test Bench

Endpoint的配置：

该仿真环境提供了7系列fpga集成块的PCI Express核心执行简单的内存访问测试PIO设计示例。事务由Root Port Model生成，并由PIO示例设计响应。

- PCI Express事务层包(TLPs)由test bench的 发送用户应用程序(pci_exp_usrapp_tx)生成。在传输TLP时，它还生成一个日志文件tx.dat。
- PCI Express TLP是由test bench的 接收用户应用程序(pci_exp_usrapp_rx) 接收的。当用户应用程序接收到TLP时，它生成一个日志文件rx.dat。

PCI Express Root Port Model提供了一个测试程序接口，可以与所提供的PIO设计或用户设计一起使用。根端口模型的目的是提供一个源机制，用于生成向下（downstream）的PCI Express TLP流来激励用户设计。同时也作为一个目标机制，用于在仿真环境中从用户设计接收向上（upstream）的PCI Express TLP流。

包含Root Port模型的源代码，可以在此基础上修改自己的Testbench。初始化配置空间、创建TLP事务、生成TLP日志以及提供创建和验证测试的接口的所有重要工作都已经完成。

Root Port模型包含：

- 测试编程接口（Test Programming Interface，TPI）：用来仿真Endpoint器件。
- 指示如何使用TPI的示例。
- 其他源码用来自定义testbench。

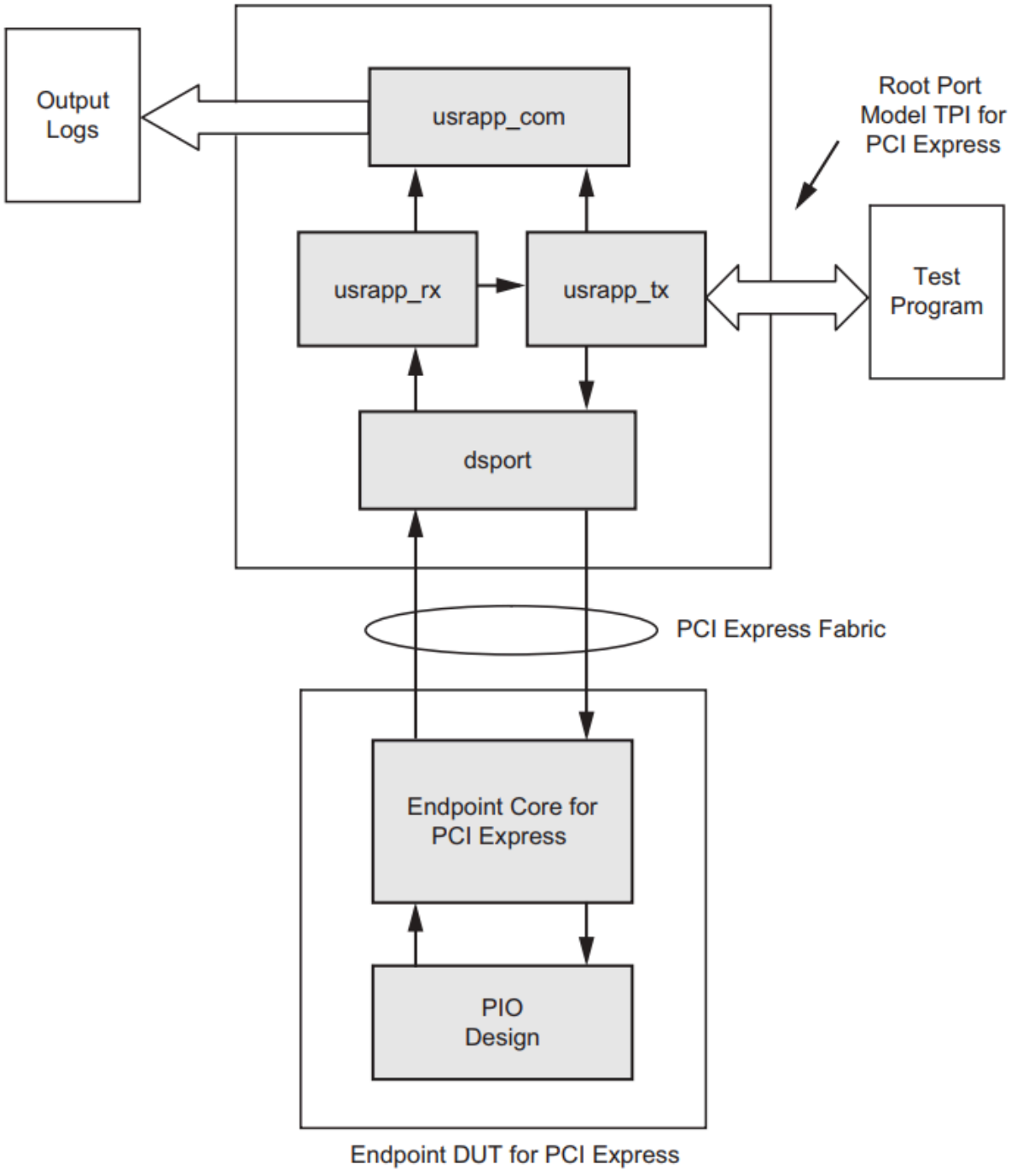


Figure 6-1: **Root Port Model and Top-Level Endpoint**

2.1 架构

usrapp_tx和usrapp_rx块与dsport块连接，用于传输和接收来自待验证的设计(Design Under Test ， DUT)Endpoint 的TLP。Endpoint DUT由PCIe Endpoint 和 PIO设计 或 用户设计的Endpoint组成。

usrapp_tx块将TLP发送到dsport块，以便在PCI Express Link上传输至Endpoint DUT。反过来，Endpoint DUT设备依次在PCI Express Link上传输TLP到dsport块，随后被传递给usrapp_rx块。当通过PCI Express逻辑进行通信时，dsport和IP核负责数据链路层和物理链路层的处理。usrapp_tx和usrapp_rx都利用usrapp_com块来实现共享功能，例如TLP处理和日志文件输出。

事务序列或测试程序由usrapp_tx块启动，以激励Endpoint设备的逻辑接口。Endpoint设备的TLP响应由usrapp_rx块接收。usrapp_tx和usrapp_rx块之间的通信, 允许usrapp_tx块验证正确的行为，并在usrapp_rx块从Endpoint设备接收到TLP时采取相应的行为。

2.2 仿真Timeout的缩放比例

用于PCI Express的7系列fpga集成块的仿真模型在链路训练期间使用缩小的时间，以允许链路在仿真期间的合理时间内进行训练。根据PCI Express Specification, rev. 2.1 规范，存在与链路训练和状态状态机(LTSSM)状态相关的各种超时。7系列fpga集成块在仿真中按256倍缩放这些超时，除了在恢复Speed_1 LTSSM状态，其中的超时没有缩放。

2.2 测试的选择

Root Port Model提供的测试，及VHDL和Verilog 的选择。

测试名称	语言	描述
sample_smoke_test0	Verilog 和 VHDL	发出PCI Type 0 Configuration Read TLP并等待完成TLP; 然后将返回的值与预期值进行比较，除了设备/供应商ID值
sample_smoke_test1	Verilog	执行与sample_smoke_test0相同的操作，但使用expectation任务。这个测试使用两个独立的测试程序线程: 1.第一个线程发出PCI Type 0 Configuration Read TLP 2.第二个线程发出Completion with Data TLP expectation任务。此测试演示了使用expectation望任务的并行测试的表单。通过此测试表格，可以确认从用户设计中收到的任何TLP。此外，当顺序不重要时，该方法可用于确认TLP的接收情况。

测试名称	语言	描述
pio_writeReadBack_test0	Verilog 和 VHDL	检查预期的设备/供应商ID、链路速度和链接宽度值。执行Endpoint枚举(扫描和配置BAR，设置PCIe配置寄存器。根据BAR类型使能 发出 内存/IO写和读数据包(IO，内存32位或64位内存)，并检查写入的值是否可以成功读回。

2.3 输出日志

当在示例或用户设计上的测试失败时，测试程序员将调试问题测试用例。通常，测试程序员会检查wave文件中的模拟，并将其与标准输出中显示的消息进行交叉引用。由于这种方法可能非常耗时，Root Port Model 提供了一种输出日志记录机制来帮助测试人员调试失败的测试用例，以加快过程。

Root Port Model在每次模拟运行期间创建三个输出日志文件。

- rx.dat和tx.dat：这些日志文件每个都包含Root Port模型分别接收和发送的每个TLP的详细记录。当在一个特定的测试用例中理解预期的TLP传输时，可以更容易地隔离故障。
- error.dat： 此日志文件与 expectation任务一起使用。使用 expectation任务的测试程序生成一个通用的错误消息到标准输出。由于 预期错误 而发生的 具体比较失败 的详细信息位于error.dat中。

2.4 并行测试程序

Root Port模型支持两类测试:

- Sequential tests: 存在于一个进程中的测试，其行为与顺序程序类似。当验证具有已知顺序的事件的行为时，顺序测试非常有用。
- Parallel tests: 涉及多个进程线程的测试。测试`sample_smoke_test1`是一个使用两个进程线程的并行测试示例。当验证一组特定的事件是否已经发生，但是这些事件的顺序未知时，并行测试非常有用。

典型的并行测试使用一个命令线程和一个或多个期望线程的形式。这些线程一起工作来验证设备的功能。命令线程的作用是创建必要的TLP事务，使设备接收和生成TLP。期望线程的作用是验证预期TLP的接收情况。根端口模型TPI有一组完整的期望任务，要与并行测试一起使用。

由于示例设计是一个仅针对目标的设备，当使用PIO设计时，并行测试程序只能预期完成tlp。然而，当与用户设计一起使用时，期望任务的完整库可以用于期望任何TLP类型(可以包括总线控制功能)。目前，VHDL版本的根端口模型测试台不支持并行测试。

2.5 测试描述

根端口模型提供了测试编程接口(TPI)。TPI提供了通过调用一系列Verilog任务来创建测试的方法。所有根端口模型测试都应该遵循相同的六个步骤:

1. 对唯一的测试名执行条件比较。
2. 在用例仿真挂起时设置主超时。
3. 等待复位和连接（link-up）。
4. 初始化Endpoint的配置空间。
5. 在根端口模型和端点DUT之间发送和接收TLP。
6. 验证测试是否成功。

测试用例编程

2.6 拓展Root Port Model

创建根端口模型是为了与PIO设计一起工作，因此对其进行了调整，以便根据PIO设计的限制进行特定的检查和警告。Vivado IDE生成根端口模型时，默认情况下将启用这些检查和警告。但是可以禁用这些限制，这样它们就不会影响设计。

因为PIO设计支持一个I/O BAR,一个Mem64 BAR, 和两个 Mem32 BARs(其中之一必须EROM空间), 默认Root Port模型在设备配置时进行检查, 确认IP核配置满足要求。如果违反此检查，将显示一条警告消息，并在test bench上适当地禁用违规的BAR。

这个检查可以通过设置`pci_exp_usrapp_tx.v`中的`pio_check_design`变量为0来取消。

2.6.1 Root Port模型TPI任务列表

Test Setup Tasks

名称	输入		描述
TSK_SYSTEM_INITIALIZATION	None		等待事务接口复位以及`Root Port`模型和`Endpoint DUT`之间的连接。必须在Endpoint IP核初始化之前调用此任务。
TSK_USR_DATA_SETUP_SEQ	None		将全局4096字节`DATA_STORE`数组项初始化为从0到4095的顺序值。
TSK_TX_CLK_EAT	clock count	31:30	等待clock_count事务接口时钟。
TSK_SIMULATION_TIMEOUT	timeout	31:0	以事务接口时钟为单位设置主仿真超时值。此任务应用于确保完成所有DUT测试。

TLP Tasks

名称	输入	描述

TSK_TX_TYPE0_CONFIGURATION_READ	tag_	7:0	从`Root Port`模型发送一个`Type 0 PCI Express Config Read TLP`到带有`tag_`和`first_dw_be_`输入的endpoint DUT。从`Endpoint DUT`返回的CplD使用全局COMPLETE_ID_CFG的内容作为完成ID
	reg_addr_	11:0	
	first_dw_be_	3:0	
TSK_TX_TYPE1_CONFIGURATION_READ	tag_	7:0	从`Root Port`模型发送一个`Type 1 PCI Express Config Read TLP`到带有`tag_`和`first_dw_be_`输入的endpoint DUT。从`Endpoint DUT`返回的CplD使用全局COMPLETE_ID_CFG的内容作为完成ID
	reg_addr_	11:0	
	first_dw_be_	3:0	
TSK_TX_TYPE0_CONFIGURATION_WRITE	tag_	7:0	从`Root Port`模型发送一个`Type 0 PCI Express Config Write TLP`到带有`tag_`和`first_dw_be_`输入的endpoint DUT。从`Endpoint DUT`返回的CplD使用全局COMPLETE_ID_CFG的内容作为完成ID
	reg_addr_	11:0	
	reg_data_	31:0	
	first_dw_be_	3:0	
TSK_TX_TYPE0_CONFIGURATION_WRITE	tag_	7:0	从`Root Port`模型发送一个`Type 1 PCI Express Config Write TLP`到带有`tag_`和`first_dw_be_`输入的endpoint DUT。从`Endpoint DUT`返回的CplD使用全局
	reg_addr_	11:0	
	reg_data_	31:0	

	first_dw_be_	3:0	COMPLETE_ID_CFG的内容作为完成ID
TSK_TX_MEMORY_READ_32	tag_	7:0	从`Root Port`模型发送一个` PCI Express Memory Read TLP `到32位内存地址`addr_`的endpoint DUT。 从`Endpoint DUT`返回的CplD使用全局COMPLETE_ID_CFG的内容作为完成ID
	tc_	2:0	
	len_	9:0	
	addr_	31:0	
	last_dw_be_	3:0	
	first_dw_be_	3:0	
TSK_TX_MEMORY_READ_64	tag_	7:0	从`Root Port`模型发送一个` PCI Express Memory Read TLP `到64位内存地址`addr_`的endpoint DUT。 从`Endpoint DUT`返回的CplD使用全局COMPLETE_ID_CFG的内容作为完成ID
	tc_	2:0	
	len_	9:0	
	addr_	63:0	
	last_dw_be_	3:0	
	first_dw_be_	3:0	
TSK_TX_MEMORY_WRITE_32	tag_	7:0	从`Root Port`模型发送一个`PCI Express Memory Write TLP`到32位内存地址`addr_`的endpoint DUT。 从`Endpoint DUT`返回的CplD使用全局COMPLETE_ID_CFG的内容作为完成ID。全局DATA_STORE字节数组用于向task传递写数据。
	tc_	2:0	
	len_	9:0	
	addr_	31:0	
	last_dw_be_	3:0	
	first_dw_be_	3:0	
	ep_	-	
TSK_TX_MEMORY_WRITE_64	tag_	7:0	从`Root Port`模型发送一个`PCI Express Memory Write TLP`到64位内存地址`addr_`的endpoint DUT。 从`Endpoint DUT`返回的CplD使用全局
	tc_	2:0	
	len_	9:0	
	addr_	63:0	

	last_dw_be_	3:0	COMPLETE_ID_CFG的内容作为完成ID。全局DATA_STORE字节数组用于向task传递写数据。
	first_dw_be_	3:0	
	ep_	-	
TSK_TX_TYPE0_CONFIGURATION_WRITE	tag_	7:0	从`Root Port`模型发送一个`PCI Express Completion TLP ` 到endpoint DUT。使用全局COMPLETE_ID_CFG的内容作为完成ID
	tc_	2:0	
	len_	9:0	
	comp_status_	2:0	

其他还有TSK_TX_COMPLETION_DATA、TSK_TX_MESSAGE、TSK_TX_MESSAGE_DATA、TSK_TX_IO_READ、TSK_TX_IO_WRITE、TSK_TX_BAR_READ、TSK_TX_BAR_WRITE、TSK_WAIT_FOR_READ_DATA、TSK_BAR_INIT、TSK_BAR_SCAN、TSK_BUILD_PCIE_MAP、TSK_DISPLAY_PCIE_MAP等,见PG054