(机器翻译成中文)

Getting started with the FPGA demo bundle for Xilinx

Xillybus Ltd.

www.xillybus.com

Version 3.3

本文档已由计算机自动翻译，可能会导致语言不清晰。 与原始文件相比，该文件也可能略微过时。

如果可能，请参阅英文文档。

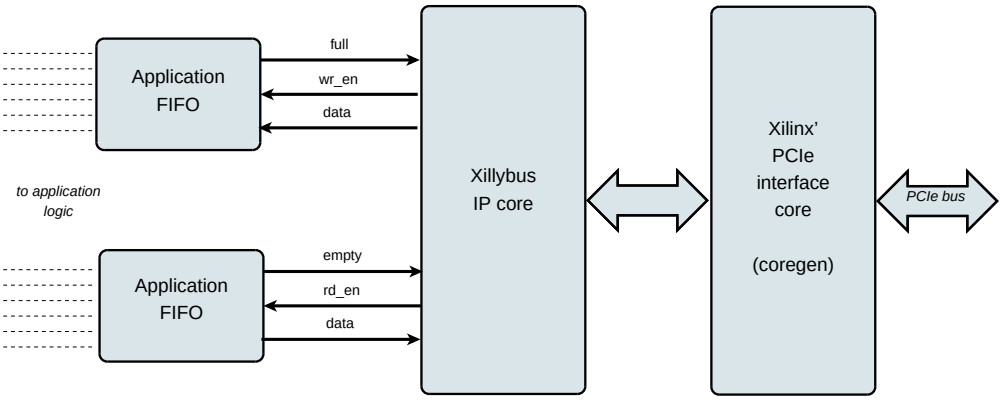
This document has been automatically translated from English by a computer, which may result in unclear language. This document may also be slightly outdated in relation to the original.

If possible, please refer to the document in English.

1

介绍

Xillybus 是基于 DMA 的端到端解决方案，用于 FPGA 和运行 Linux 或 Microsoft Windows 的 host 之间的数据传输。它为 FPGA logic 的设计者和软件的程序员提供了一个简单直观的界面。



如上图，FPGA上的application logic只需要与标准的FIFOs交互即可。

例如，将数据写入图中较低的 FIFO 会使 Xillybus IP core 感觉到数据可以在 FIFO 的另一端进行传输。很快，IP core 就从 FIFO 读取数据并发送给 host，让 userspace software 可以读取。数据传输机制对 FPGA 中的 application logic 是透明的，仅与 FIFO 交互。

另一方面，Xillybus IP core 利用 PCI Express 的 Transport Layer level 实现数据流，生成和接收 TLP 数据包。对于较低层，它依赖于 Xilinx 的官方 PCIe core，它是开发工具的一部分，并且不需要额外的许可证（即使使用 WebPACK 版本）。

计算机上的应用程序与 device files 交互，其行为类似于 named pipes。 Xillybus IP core 和 driver 在 FPGAs 中的 FIFOs 和 host 上的相关 device files 之间高效且直观地传输数据。

使用XillyUSB，一个MGT transceiver用于实现一个USB 3.0接口，用于数据传输，而不是上面提到的PCIe接口。

IP core 使用在线 Web 应用程序根据客户的规格即时构建。 streams 的数量、方向和其他属性由客户定义，以实现 design 的带宽性能、同步和简单性之间的最佳平衡。按照本指南中的说明使用 demo bundle 完成准备步骤后，建议在 http://xillybus.com/custom-ip-factory 上构建和下载您的自定义 IP core。

本指南介绍如何使用 Xillybus IP core 快速设置 FPGA，Xillybus IP core 可以附加到用户提供的数据源和数据消费者，用于实际应用场景测试。 IP core 包含在 demo bundle 中，可以在网站上下载。

尽管名称如此，demo bundle 并不是一个演示套件，而是一个功能齐全的 starter design，它可以按原样执行有用的任务。

对于那些好奇的人，可以在 Xillybus host application programming guide for Linux 或 Xillybus host application programming guide for Windows的附录 A 中找到关于如何实现 Xillybus 的简要说明。

2

先决条件

2.1 硬件

Xillybus FPGA demo bundle 封装后可与多个板和设备一起使用，如下载页面中所列（请参阅下面的 2.2 部分）。

在对引脚位置进行必要的更改并验证 MGT 的 reference clock 处理正确后，其他板的所有者可以在自己的硬件上运行 demo bundle。这对于任何经验丰富的 FPGA 工程师来说都应该是直截了当的。在 4.3 部分中了解更多信息。

2.2 FPGA项目

Xillybus demo bundle 可在 Xillybus 网站的下载页面下载。对于基于 PCIe 的 cores：

http://xillybus.com/pcie-download

对于 XillyUSB：

http://xillybus.com/usb-download

demo bundle包含Xillybus IP core的特定配置，用于简单测试。因此，它对于某些应用程序的性能相对较差。

可以使用 IP Core Factory Web 应用程序配置、自动构建和下载自定义 IP cores。请访问 http://xillybus.com/custom-ip-factory 以使用此工具。

任何下载的捆绑包，包括 Xillybus IP core，都可以免费使用，只要这种使用合理地匹配 “evaluation” 一词即可。这包括将 core 集成到最终用户 designs 中，运行真实数据和现场测试。 core 的使用方式没有限制，只要这种使用的唯一目的是评估其功能和对特定应用的适用性。

2.3 开发软件

下面列出了 Xillybus 的 demo bundle（以及其他涉及 Xillybus 的 designs）的 implementation 的推荐工具，具体取决于 FPGA 的系列。

Xillybus 用于 PCIe：

到今天为止，几乎可以肯定安装在您计算机上的 Vivado 版本适用于 Xillybus for PCIe。然而，这些是更详细的要求：

- 使用 Virtex-5 FPGAs 时，首选 Xilinx ISE 13.1 版本，参见 4.3.5 段落。
- 对于 Spartan-6 和 Virtex-6，请使用 Xilinx ISE 13.2 及更高版本。
- 对于带有 Gen2 接口的 Kintex-7 和 Virtex-7（所有不是 XT/HT 的 Virtex-7，以及 485T），Vivado 2014.1 及更高版本是首选工具。在 ISE 版本中，推荐使用 14.2 及更高版本。
- 对于带Gen3接口的Virtex-7（XT/HT，485T除外），首选Vivado 2014.1及更高版本。如果选择 ISE，则需要 14.6 及更高版本。
- 对于 Artix-7，应使用 Vivado 2014.1 及更高版本。 ISE 14.6 及更高版本也很好。
- 对于 Kintex / Virtex Ultrascale，应使用 Vivado 2015.2 及更高版本。没有 ISE 版本支持这些设备。
- Ultrascale+ FPGAs 需要 Vivado 2017.3 及更高版本。
- Versal APAC FPGAs 需要 Vivado 2021.2 及更高版本。

XillyUSB：

- 对于除 Ultrascale+ 之外的所有 FPGAs，应使用 Vivado 2015.2 及更高版本。
- 对于 Ultrascale+，应使用 Vivado 2018.3 及更高版本。

该软件可以直接从 Xilinx 的网站 (http://www.xilinx.com) 下载。

该软件的任何版本都适用。如果 FPGA 包含在 WebPACK 版本中，则此版本可能是首选，因为它可以无限期免费下载和使用。

Xillybus 的 implementation 依赖于 Xilinx 提供的一些 IP cores。所有软件版本均涵盖这些 IP cores，无需任何额外许可。

2.4 使用 FPGA design 的经验

当 design 适用于出现在 demo bundles 列表中的电路板时，无需以前的 FPGA design 经验即可让 demo bundle 在 FPGA 上运行。使用其他板卡时，需要具备一定的Xilinx工具使用知识，尤其是定义管脚位置和clocks。

要充分利用 demo bundle，必须充分了解 logic design 技术，以及掌握 HDL 语言（Verilog 或 VHDL）。尽管如此，Xillybus demo bundle 是学习这些的一个很好的起点，因为它提供了一个简单的 starter design 来进行实验。

3

demo bundle 的 implementation

3.1 概述

对于想要跳过阅读以下所有内容的 Vivado 用户，这些是运行第一个 implementation 的步骤：

- 将 demo bundle 解压缩到工作目录。
- 启动 Vivado，如果 Vivado 已经在运行，则关闭任何打开的项目。
- 选择 Tools > Run Tcl Script... 并在 verilog/ 或 vhdl/（在 demo bundle 内）中选择 **xillydemo-vivado.tcl**。
- 单击 “Generate Bitstream”（或 “Generate Device Image”）。
- implementation成功后，在vivado/xillydemo.runs/impl_1/中找到bitstream文件。

现在说长话：Xillybus 的 demo bundle 的 implementation 有三种可能的方法，并获取 bit stream 文件：

- 按原样使用包中的项目文件。这是最简单的方法，适用于 demo bundles 列表中出现的板，ML506 (Virtex-5) 除外。
- 修改文件以匹配不同的 FPGA。这适用于与其他板和/或其他 FPGAs 一起使用时。这在使用 Virtex-5 时也是必要的。 4.3 段中有关此的更多信息。
- 从头开始设置 Vivado（或 ISE）项目。将 demo bundle 与现有 application logic 集成时可能需要。 4.2 段中的更多细节。

在本节的其余部分中，将详细介绍第一个工作过程，这是最简单和最常选择的一个。其他两个工作程序基于第一个，差异在上文给出的段落中有详细说明。

重要的:

评估包的配置是为了简单而不是性能。对于需要持续和连续数据流的应用程序，特别是对于高带宽情况，可以获得明显更好的结果。对于这些场景，可以使用 *Web* 应用程序轻松构建和下载自定义 *IP core*。

3.2 文件大纲

该捆绑包由其中一些目录组成（存在哪些目录取决于预期的 FPGA ）：

- core——Xillybus IP core存放在这里
- instantiation templates– 包含 core 的 instantiation templates（在 Verilog 和 VHDL 中）
- verilog– 包含 demo bundle 的项目文件和 Verilog 中的源代码（在 'src' 子目录中）
- vhdl– 包含 demo bundle 的项目文件和 VHDL 中的源代码（在 'src' 子目录中）
- vivado-essentials——logic 的定义文件和构建目录，供 Vivado 使用。
- blockplus——该目录仅与 Virtex-5 相关。参见 4.3.5 段。

请注意，每个 demo bundle 都适用于特定的板，如下载 demo bundle 的站点网页中所列。如果使用了另一块板，或者在板上添加或移除了某些配置电阻，则必须相应地编辑 constraints 文件。

对于 Vivado 项目，此文件是 vivado-essentials/xillydemo.xdc，对于 ISE 项目，它是 verilog/ 或 vhdl/ 下所选 'src' 目录中的 UCF 文件。

另请注意，vhdl 目录包含 Verilog 文件，但这些文件都不需要进行重大更改。

Xillybus 的 IP core 和 application logic 之间的接口发生在 xillydemo.v 文件或 xillydemo.vhd 文件中（在各自的 'src' 子目录中）。这是要编辑的文件，以便用您自己的数据试用 Xillybus。

3.3 使用 Vivado 生成 bitstream 文件

ISE 用户：请跳至 3.4 段。

Vivado会生成很多中间文件，结构比较复杂，项目很难控制。为了保持包中的文件结构紧凑，提供了 Tcl 中的 script 用于创建 Vivado 项目。此 script 创建一个新的子目录“vivado”，并根据需要使用文件填充此目录。

该项目依赖于 src/ 子目录中的文件（没有制作这些文件的副本）。 PCIe 块以及 logic 使用的 FIFOs 在 vivado-essentials/ 中定义。 Vivado 还在项目的 implementation 期间使用中间文件填充此目录。

启动Vivado。在没有打开项目的情况下，选择 Tools > Run Tcl Script...，然后根据您的喜好在 verilog/ 或 vhdl/ 子目录中选择 **xillydemo-vivado.tcl**。一系列事件在不到一分钟的时间内发生。可以通过选择 Vivado 窗口底部的 “Tcl Console” 选项卡来验证项目部署是否成功，并验证其是否显示：

```
INFO: Project created: xillydemo
```

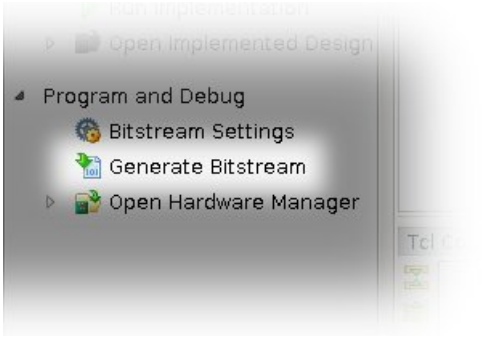
如果这不是 Tcl console 中的最后一行，则说明出了问题，很可能是因为使用了错误的 Vivado 修订版。

Warnings 会在此阶段出现，但不会出现错误。但是如果项目已经生成（即script已经运行），再次尝试运行script会出现如下错误：

```
ERROR: [Common 17-53] User Exception: Project already exists on disk,
       please use '-force' option to overwrite:
```

请注意，新的 “vivado” 子目录是在包含 Tcl script.1 的目录中创建的

创建项目后，启动 implementation：单击左侧 Flow Navigator 栏上的 “Generate Bitstream”（或 “Generate Device Image”）。



可能会出现一个询问是否可以启动 synthesis 和 implementation 的弹出窗口——选择 “Yes”。

Vivado 运行一系列进程。这通常需要几分钟。发布了几个 warnings，其中没有一个被归类为关键（但一些 critical warnings 可能仍保留在执行 Tcl script 的日志中）。

会出现一个弹窗，提示bitstream的implementation已成功完成。它将给出下一步做什么的选择。任何选项都可以，包括 “Cancel”。

bitstream 文件 xillydemo.bit 可以在 vivado/xillydemo.runs/impl_1/ 找到。对于 Versal FPGAs，文件是 xillydemo.pdi。

implementation 永远不会失败。然而，有一个错误情况值得一提：

集成自定义 logic 时可能会出现 “Timing constraints weren’t met” 错误。这意味着这些工具未能达到 timing 的要求。在这种情况下，design 在语法上是正确的，但需要更正以使某些路径相对于给定的 clock 速率和/或 I/O 要求足够快。将 design 校正为更好的 timing 的过程通常称为 *timing closure*。

timing constraint 故障通常被宣布为 critical warning，因此 Vivado 不会阻止用户生成不能保证 FPGA 可靠行为的 bitstream 文件。为了防止创建这样的 bitstream，timing 故障通过一个小的 Tcl script（即 “showstopper.tcl”）转化为错误，它在 route 运行结束时自动执行。要关闭此安全措施，请单击 Flow Navigator 中 “Project Manager” 下的 “Project Settings”。选择 “Implementation” 按钮，然后向下滚动到 “route_design” 的设置。然后从 tcl.post 中删除 showstopper.tcl。

Vivado 用户可跳过以下章节，直接进入 3.6 段落。

3.4 设置 Xilinx 的 PCIe IP core

此部分仅与 ISE 工具链有关，适用于 FPGAs 而非 Spartan-6。如果使用 Vivado，请参考 3.3 段落。那些使用 Spartan-6 FPGAs 的人可以直接跳到 3.5 段。

PCIe 的 Coregen IP core 有点特殊的组织不允许在 implementation 项目中包含 XCO 文件，而是 core 生成软件创建 Verilog 文件以供包含。只有在使用 Virtex-5、Virtex-6 和 series-7 时才会出现这种情况。

Virtex-5 FPGAs 需要对 XCO 文件进行一些特殊处理。参见第 4.3.5 段。

请在 blockplus 目录或 pcie_core 目录（在您的包中找到）中找到项目文件 (.xise)，然后双击它以打开 ISE。在 “Design Utilities” 下，单击 “Regenerate all cores” 并等待该过程完成。然后关闭ISE。无需任何进一步操作。

此过程生成一组 Verilog 文件，它们是 Xilinx PCIe core 的包装器。这些文件由创建 demo bundle 的 bitstream 的项目使用。这些文件在 Verilog 中生成，无论您选择 Verilog 还是 VHDL 作为首选语言。

这些不需要在主项目中手动包含这些 Verilog 文件，但这些文件必须在尝试整个项目的 implementation 之前生成一次。无需重复此过程，即使在重复主项目的 implementation 之前也无需重复。

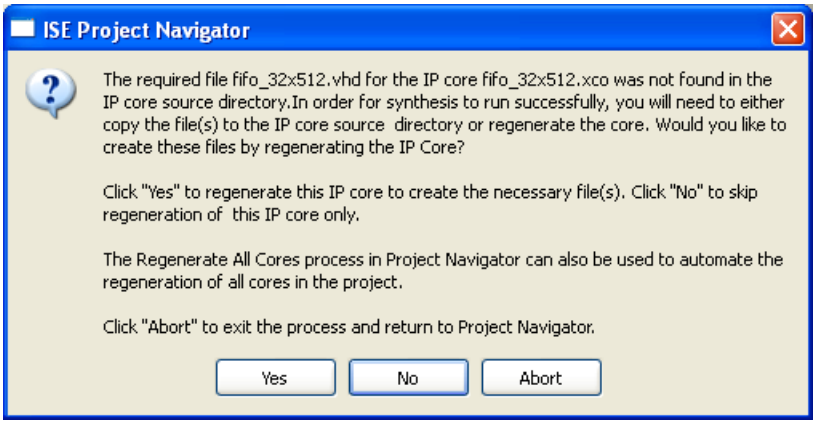
3.5 使用 ISE 套件生成位文件

使用属于 Virtex 或 series-7 系列的 FPGA 时，请确保您已按照上述 3.4 段落中的说明准备好 PCIe 包装器。

根据您的喜好，双击 'verilog' 子目录或 'vhdl' 子目录中的 'xillydemo.xise' 文件。 ISE 将开始运行并使用正确的设置打开项目。 ISE 不应该抱怨任何文件丢失。如果是这样，并且 FPGA 系列是 series-7 或任何 Virtex 系列之一，则您的 PCIe 包装器可能没有按照 3.4 段中所述进行准备。

单击 “Generate Programming File” 启动 implementation。

在第一个 implementation 期间，需要重新生成两个或三个 Xilinx Coregen IP cores。这个过程很耗时，但幸运的是它只完成了一次。一个看起来像这样的弹出窗口会出现两到三次：



在所有这些上单击 “Yes”。

该过程将产生多个 warnings（FPGA 项目的 implementation 总是如此）但不应出现任何错误。该过程完成后，可以发现 bitfile 为 xillydemo.bit。

始终通过在 log 中查找以下句子（接近结尾处）来验证 timing constraints 是否已实现：

```
All constraints were met.
```

这是至关重要的，尤其是在对项目进行更改之后。如果未实现 constraints，工具仍会生成 bitfile，但 FPGA 可能会以不可预知的方式运行（可能是由于温度变化在允许的温度范围内）。

在 ISE 的 design 摘要中可以找到类似的消息。

未能实现 timing constraints 可能是因为添加的 logic 的速度不足以承受 bus_clk 的速率。但是，如果没有明显原因发生故障，则可能是 Xilinx 的工具在尝试将 logic 组件放置在 FPGA 的 logic fabric 上时做出了错误的初始猜测，而稍后运行的优化算法无法解决此问题。

后一种情况可以通过更改 placer cost table figure（这只是 seed 到初始放置的随机性）来修复。在 ISE Project Navigator 内的 Processes 窗格中，右键单击 “Map” 并选择 “Process Properties...”。确保 Property display level 是 “Advanced” 并将 “Starting Placer Cost Table” 更改为尚未尝试过的任何其他数字。这个数字的大小没有意义。然后用 “Generate Programming File” 重新启动。

重要的:

在某些 *ISE* 版本（尤其是 *ISE 14.2*）上，*verilog/* 目录中的构建可能会失败，并显示 *ERROR:HDLCompiler:687 - “C:/try/xillybus-eval-kintex7-1.1/verilog/src/fifo_32x512_synth.v” Line 54: Illegal redeclaration of module fifo_32x512.*（或类似）错误。这是由于 *Xilinx* 工具中的 *bug*。要解决此问题，请删除 *src/* 目录中的 *fifo_8x2048_synth.v* 和 *fifo_32x512_synth.v*，然后重新启动 *“Generate Programming File”*。一些 *warnings* 将指示工具无法找到这些文件，但 *implementation* 仍应正常运行。

3.6 加载 bitfile

在早期开发阶段，建议通过 JTAG 加载 FPGA。在大多数主板上，在运行 Vivado 的计算机和主板面板上的 USB 连接器之间使用一条简单的 USB 电缆就足够了。使用 ISE 的人，使用 iMPACT 加载 bitfile。

有关如何通过 JTAG 加载 FPGA，请参阅您的主板说明。

对于 XillyUSB 项目，FPGA 可以随时加载和重新加载，即使 USB 接口连接到工作计算机也是如此。

对于 PCIe 项目，FPGA 必须在计算机上电前加载 bitfile：计算机期望 PCIe 外设在上电时处于正常状态，之后可能不会容忍任何意外。

因此，只要 host 正在运行，就**不要**重新加载 FPGA。尽管 PCIe 规范要求支持 hotplugging，但主板通常不会期望 PCIe 卡消失然后重新出现。因此，某些主板可能无法正确响应。不过，在操作系统运行时重新加载 FPGA 可以在某些主板上运行。

Xillybus的driver在设计上对hotplugging反应灵敏，但是没有什么可以保证电脑的整体稳定性。这是在这个页面上讨论的：

http://xillybus.com/doc/hot-reconfiguration

如果 FPGA 加电并从 flash memory 加载并同时启动计算机，则必须确保 FPGA 加载足够快，以便在 BIOS 扫描 bus 时存在 PCIe 设备。

4

修改

4.1 与定制 logic 集成

Xillybus demo bundle 的构造便于与 application logic 集成。连接数据的地方是 xillydemo.v 或 xillydemo.vhd 文件（取决于首选语言）。为了使用 Xillybus IP core 在 host（Linux 或 Windows）和 FPGA 之间传输数据，可以忽略捆绑包中的所有其他 HDL 文件。

可以将带有自定义 logic designs 的附加 HDL 文件添加到按照 3.5 或 3.3 段落所述准备的项目中，然后通过单击 “Generate Programming File” 或 “Generate Bitstream” 重新构建。无需重复初始部署的其他步骤，因此 logic 的开发周期相当快速和简单。

将 Xillybus IP core 连接到自定义 application logic 时，强烈建议仅通过 FIFOs 与 Xillybus IP core 交互，不要试图模仿 logic 的行为，至少在第一阶段不要。

一个例外是将存储器或 register arrays 连接到 Xillybus 时，在这种情况下，应遵循 xillydemo 模块中显示的示例。

在xillydemo模块中，FIFOs被用来执行一个data loopback。换句话说，从 host 到达的数据被发送回它。 FIFO的两边都连接到Xillybus IP core，所以core既是数据的来源，也是数据的消费者。

在实际使用场景中，只有 FIFO 的一侧连接到 Xillybus IP core。 FIFO 的另一端连接到 application logic，它提供或消耗数据。

xillydemo 模块中使用的 FIFOs 仅与两侧共用一个 clock 一起工作，因为两侧均由 Xillybus 的主 clock 驱动。在实际应用中，可能需要将它们替换为 FIFOs，FIFOs 具有单独的 clocks 用于读取和写入。这允许使用 bus_clk 以外的 clock 驱动数据源和数据消费者。通过这样做，FIFOs 不仅可以充当调解器，还可以用于正确的 clock domain 交叉。

请注意，对于从 FPGA 到 host 的 streams，Xillybus IP core 需要一个普通的 FIFO 接

口（与 First Word Fall Through, FWFT 相反）。

以下文档与集成自定义 logic 相关：

- API 用于 logic design：Xillybus FPGA designer's guide
- Getting started with Xillybus on a Linux host
- Getting started with Xillybus on a Windows host
- Xillybus host application programming guide for Linux
- Xillybus host application programming guide for Windows
- The guide to defining a custom Xillybus IP core

4.2 包含在自定义项目中

如果需要，可以将 Xillybus IP core 包含在现有 Vivado / ISE 项目中，或从头开始创建新项目。

如果该项目尚不存在，请启动一个新项目，并根据您首选的 HDL 语言和预期的 FPGA 进行设置。

要将 Xillybus IP core 包含在 Vivado 项目中，建议编辑 xillydemo-vivado.tcl 以反映自定义项目的源文件和设置，并通过运行此 script 创建一个新项目。

要将 Xillybus IP core 包含在 ISE 项目中：

- 当使用属于 Virtex-5/6 或 series-7 系列的 FPGAs 时，需要单独生成 PCIe 包装文件，如 3.4 段落（以及 Virtex-5 FPGAs 的 4.3.5 段落）中所述。生成的 Verilog 文件应该添加到自定义项目中（而不是 XCO 文件）。
- 将两个 src/ 子目录之一中的所有文件（取决于您的语言偏好）添加到项目中。
- 将目录添加到 Macro search path： 在 process 菜单中，在 “Implementation” 下，右键单击 “Translate” 并选择 “Process Properties...”。将 'core' 子目录添加到 Macro Search Path 属性（使用最右侧的按钮进行浏览）。未能设置此属性将使 Translate 阶段在 implementation 期间失败，因为它不会找到 xillybus_core.ngc 文件。
- 如果 xillydemo 模块不是项目的 top level module，请将其端口连接到 top level。
- 要将 Xillybus IP core 连接到自定义 application logic，请编辑 xillydemo 模块，将现有的 application logic 替换为所需的模块。

4.3 使用其他板

4.3.1 一般的

使用未出现在 demo bundles 列表中的电路板时，需要对捆绑包进行一些细微修改。

core 生成一些 GPIO LED 输出。建议将这些连接到板上的 LEDs，如果有任何这样的空缺。

4.3.2 将 Xillybus 用于 PCIe

大多数购买的板都有自己的 FPGA design 示例，它显示了如何在该板上使用 PCIe 接口。通常最容易在目标板的 XDC / UCF 文件中找到相关的引脚分配，并将引脚名称修改为 Xillybus 的 XDC / UCF 文件中使用的名称。然后可以替换 Xillybus 项目中使用的 XDC / UCF 文件中的相关行。

下面给出了有关如何放置引脚的详细信息。

请注意，最常见的错误是使用 PCIe bus 的 reference clock。仅连接任何具有相同频率的 clock 将不起作用：主板的 clock 与任何其他 clock 之间的微小频率差异会使 transceiver 偶尔失锁，导致通信不可靠，并且可能无法将 FPGA 检测为 PCIe 设备。

由于 Xillybus core 基于 Xilinx 的 PCIe core，Xilinx 的用户指南是与 PCIe 的 physical layer 相关的注意事项的有效来源。

4.3.3 使用 Spartan-6 PCIe 板

对于 Spartan-6，Xillybus core 通过 PCIe bus 端口与 host 接口，该端口由 7 条物理线组成，如下所示：

- 一对用于 reference clock 的 differential wires，名称为 PCIE_250M_P 和 PCIE_250M_N：频率为 125 MHz（尽管 net 的名称）的 clock 源自 PCIe bus clock，预计在这些电线上。如果应用不同的 clock，则必须重新配置 Xilinx PCIe Coregen core（由捆绑包中的 pcie.xco 定义）以期望获得真正的 clock 频率。此外，必须更新 timing constraint，以便 TS_PCIE_CLK 规范反映更改。
- PCIE_PERST_B_LS 上的 host 的 master bus reset
- 串行数据输入对，PCIE_RX0_P 和 PCIE_RX0_N
- 串行数据输出对，PCIE_TX0_P 和 PCIE_TX0_N

这些引脚的分配是根据电路板的接线设置的。

4.3.4 使用 Virtex-6 PCIe 板

对于 Virtex-6，接线类似：

- 一对用于 reference clock 的 differential wires，名称为 PCIE_REFCLK_P 和 PCIE_REFCLK_N：这些电线上预期有一个频率为 250 MHz 的 clock，它源自 PCIe bus clock。如果应用不同的 clock，则必须重新配置 Xilinx PCIe Coregen core（由捆绑包中的 pcie_v6_4x.xco 定义）以预期真正的 clock 频率。这样的变化也可能涉及 constraints 的变化。请参考由 Coregen 生成的示例 UCF 文件。
- PCIE_PERST_B_LS 上的 host 的 master bus reset
- 串行数据输入矢量对，PCIE_RX_P 和 PCIE_RX_N（各 4 线）
- 串行数据输出矢量对，PCIE_TX_P 和 PCIE_TX_N（各 4 线）

通过放置 transceiver logic 隐式进行引脚分配。 constraints 在 UCF 文件中定义 GTX 位置的 constraints 强制某个 pinout。同样，reference clock 引脚的位置通过约束 clock buffer (pcieclk_ibuf) 的位置来隐式设置。 Xillybus 捆绑包中的 UCF 文件包含指导性注释。

必须编辑 UCF 文件，以使它们的引脚位置与预期电路板的引脚位置相匹配。

4.3.5 使用 Virtex-5 PCIe 板

Virtex-5 系列中有两组设备，每组都需要稍有不同的 PCIe 接口。为了简单地处理这个问题，'blockplus' 子目录中有两个不同的 XCO 文件，应该只使用其中一个。

因此，在构建 PCIe core 之前，有必要重命名该子目录中的文件，如下所示：

- 对于 Virtex-5 LX 或 Virtex-5 SX：将 pcie_v5_gtp.xco 重命名为 pcie_v5.xco
- 对于 Virtex-5 FX 或 Virtex-5 TX：将 pcie_v5_gtx.xco 重命名为 pcie_v5.xco

重要的:

*PCIe Block Plus generator*的版本应该是*1.14*，绝对不是*1.15*。 *ISE 13.1* 有用于此目的的正确版本，但 *ISE 13.2* 随附的版本会产生错误代码。

如果整个 implementation 需要 ISE 13.1 以外的版本，则可以使用正确版本的 PCIe Block Plus（包含在 ISE 13.1 中）生成 Verilog 文件。整个项目的 implementation 然后可以在首选版本的 ISE 中完成。

UCF 文件有关于如何设置引脚的指导意见。 PCIe 引脚的位置是隐式的，并且由 constraint 强制在 GTP/GTX 组件的位置上。

PCIE_REFCLK 线对上应有一个频率为 100 MHz 的 clock。如果应用不同的 clock，则必须重新配置 Xilinx PCIe Coregen core（由 pcie_v5.xco 定义）以期望真正的 clock 频率。此外，必须更新 timing constraint，以便 TS_MGTCLK 规范反映更改。

4.3.6 使用 Kintex-7、Virtex-7 和 Artix-7 板 (PCIe)

series-7 系列中的所有 FPGAs 都具有相同的 PCIe 接口。

- 一对 differential wires 用于 reference clock，名称为 PCIE_REFCLK_P 和 PCIE_REFCLK_N：预计在这些电线上使用来自 PCIe bus clock（或直接连接）的频率为 100 MHz 的 clock。

  如果应用了不同的 clock，则必须重新配置 PCIe 块（由 pcie_k7_vivado.xci 或 demo bundle 中的类似内容定义）以期望真正的 clock 频率。该文件出现在项目的源列表中。这样的变化也可能涉及到 timing constraints 的变化。请参考示例 XCF 文件，由 Xilinx 的工具生成。

  如果使用 ISE，则 Xilinx 的 PCIe core 由例如 pcie_k7_8x.xco 定义。 UCF 文件，而不是 XDC，可能需要调整。

- PCIE_PERST_B_LS 上的 host 的 master bus reset
- 串行数据输入矢量对，PCIE_RX_P 和 PCIE_RX_N（各 8 或 4 线）
- 串行数据输出矢量对，PCIE_TX_P 和 PCIE_TX_N（各 8 或 4 线）

通过放置 transceiver logic 隐式进行引脚分配。在 UCF/XDC 文件中定义 GTX 位置的 constraints 强制某个 pinout。同样，reference clock 引脚的位置通过约束 clock buffer (pcieclk_ibuf) 的位置来隐式设置。 Xillybus 的 demo bundle 中的 UCF / XDC 文件包含指导性注释。

必须编辑 UCF/ XDC 文件，以使它们的引脚位置与预期电路板的引脚位置相匹配。

4.3.7 使用 Ultrascale 和 Ultrascale+ 板 (PCIe)

所有这些 FPGAs 都具有相同的 PCIe 接口。

- 一对differential wires为reference clock，分别命名为PCIE_REFCLK_P和PCIE_REFCLK_N：一个频率为100 MHz的clock，直接连接PCIe bus的clock。

如果应用了不同的 clock，则必须重新配置 PCIe 块（由 pcie_ku_vivado.xci 或 demo bundle 中的类似定义）以期望真正的 clock 频率，并且必须在 xillydemo.xdc 中更新此 clock 的 timing constraint。这些文件出现在项目的源列表中。

- PCIE_PERST_B_LS 上的 host 的 master bus reset
- 串行数据输入矢量对，PCIE_RX_P 和 PCIE_RX_N（各 8 或 4 线）
- 串行数据输出矢量对，PCIE_TX_P 和 PCIE_TX_N（各 8 或 4 线）

通过放置 transceiver logic 隐式进行引脚分配。在 XDC 文件中定义 GTX 位置的 constraints 强制某个 pinout。同样，reference clock 引脚的位置通过约束 clock buffer (pcieclk_ibuf) 的位置来隐式设置。 Xillybus 的 demo bundle 中的 XDC 文件包含指导性注释。

必须编辑 XDC 文件，以使它们的引脚位置与预期电路板的引脚位置相匹配。

4.3.8 使用 Versal ACAP 板 (PCIe)

这些 FPGAs 具有以下 PCIe 接口：

- 一对differential wires为reference clock，分别命名为PCIE_REFCLK_P和PCIE_REFCLK_N：一个频率为100 MHz的clock，直接连接PCIe bus的clock。

  如果应用不同的 clock，则必须重新配置 CPM block 中的 PCIe controller（位于 pcie_versal block design 中的 CIPS IP 内部）以期望真正的 clock 频率。
- 串行数据输入矢量对，PCIE_RX_P 和 PCIE_RX_N（各 8 线）
- 串行数据输出矢量对，PCIE_TX_P 和 PCIE_TX_N（各 8 线）

host 的 master bus reset 直接连接到 PMC MIO 38。如果另一个 MIO pin 用于此信号，则必须对 CIPS IP 进行相应配置（有关详细信息，请参阅 Xillybus 的 tutorial page）。

XDC 不包含任何与 PCIe 块相关的 constraints。 timing constraints 和 pin placement constraints 均由 CIPS IP 隐式提供。注意 pin placements 不能移动，因为 CPM 用于 PCIe 接口。

更改PCIe block参数的过程与其他FPGAs不同：PCIe部分作为block design实现。在这个block design中，有一个名为pcie_block_support的块。该块包含 PCIe 块使用的 transceivers 和 clock resources。因此，更改lanes或link speed的编号后，需要更新pcie_block_support。仅更新 pcie_block 是不够的。

更新pcie_block_support的方法是删除该块并让Vivado重新生成它。这样，就可以使用 PCIe 块的更新参数创建该块。

建议在开始此过程之前创建 block design 的可视副本。这将很有帮助，因为需要使用更新的 pcie_block_support 块重建相同的 block design。

此过程的步骤如下：

- 删除 pcie_block_support 块及其所有 external ports。这意味着删除未连接到任何东西的 ports （例如 “pcie_mgt” ），并删除连接到块的 ports （例如 “m_axis_cq_0” ）。
- 删除 versal_cips_0 和 pcie_block 之间 reset signal 的连接。
- 作为对上一步的响应，Vivado 应建议 “Run Block Automation”。单击该建议。Vivado 将打开 pop-up windows 作为响应。验证PCIe参数是否正确，然后单击““OK””。请注意，Vivado 也会建议 “Run Connection Automation”，但此选项还不够。
- Vivado 将添加一个新的 pcie_block_support 块并进行多个连接。
- 删除与“sys_reset”相关的external port。相反，将 versal_cips_0 的 pl_pcie0_resetn 连接到两个块的 sys_reset inputs：pcie_block 和 pcie_block_support。完成此操作后，“sys_reset” 像以前一样连接。
- 选择 PCIe 块中未连接到任何东西的所有 pins（可以使用 CTRL-click 来实现此目的）。通过使用 right-click > Make External 将这些 ports 外部化。 Vivado 将为所有 pins 创建 ports。每个 external port 的名称将与 net 的名称类似，并添加 “_0” 后缀。

4.3.9 使用 XillyUSB

XillyUSB 可用于具有 SFP+ 接口的其他板上。在这种情况下，只需将 design 的 constraints 设置为使用连接到 SFP+ 连接器的 MGT。

该板还应为 MGT 提供低 jitter 的 125 MHz reference clock。尽管 USB 规范中有要求，但不应启用 Spread Spectrum Clocking (SSC)（如果存在此类选项）：如果使用 SSC reference clock，MGT 无法正确锁定接收到的信号。

对于定制板，建议参考 sfp2usb 模块的原理图，因为 SFP+ 连接器的引脚直接连接到 FPGA 的 MGT。可以选择交换 SSRX 线，就像在 sfp2usb 模块上所做的那样。仅在简化 PCB design 时才推荐这样做。

如果需要，也可以交换 SSTX 线。这需要编辑 *_frontend.v 文件，以便反转传输位的极性，从而补偿线对交换。请注意，即使没有进行此编辑，USB 连接也很有可能会正

常工作，因为 USB 规范要求 link partners 即使在极性交换的情况下也能正常工作。但是建议不要依赖它。

4.4 PRSNT 引脚用于指示 PCIe lanes 的数量

根据 PCIe 规范，PCIe 连接器上有一个或多个引脚，表示 PCIe slot 中存在外设，以及 lanes 的数量。这些是 PRSNT 引脚。大多数开发板都有 DIP switches，通过这些引脚来调整 host 通知的 lanes 数量。

这些引脚的典型默认设置是板子可能的最大 lanes 数量。此设置通常有效，即使实际使用的 lanes 较少。这是因为 host 和外围设备之间的初始协商（这是 PCIe 规范所要求的）确保正确检测 lanes 的实际数量。

有关如何设置这些 DIP switches 的信息，请参阅您开发板的参考手册。重要的是不要将这些 DIP switches 设置为小于实际使用的 lanes，因为某些 hosts 可能会由于设置错误而忽略 lanes。

4.5 更改 PCIe lanes 和/或 link speed 的数量

4.5.1 介绍

重要的:

更改 *link* 的参数可能需要调整 *timing constraints*。不注意这个问题可能会导致 *PCIe link* 工作，但方式不可靠。
如有必要，请务必在进行更改后正确调整 *timing constraints*。下面详细介绍这个主题。

Xillybus 的 FPGA demo bundles 通常设置为预期板上可用的最大 lanes 数量，以及 2.5 GT/s 的 link speed (Gen1)。

基本原理是，如果 FPGA 板适合主板的 PCIe 连接器，则可以预期所有 lanes 都将用于与 host 连接。另一方面，在几乎所有情况下，这些 lanes 实现的带宽都高于 Xillybus IP core 可能使用的带宽，即使是 2.5 GT/s，因此设置更高的 link speed 是没有意义的。

由于 PCIe 规范需要回退功能以降低所涉及的所有 bus 组件的速度，因此选择 2.5 GT/s 可确保所有主板上的行为一致。

然而，经常需要更改 lanes 及其 link speed 的数量，特别是在定制板上使用 Xillybus IP core 时。较少的 lanes 和较高的 link speed 是常见的要求。

Xillybus IP core 依赖于 Xilinx 的 PCIe 模块与 PCIe bus 的低级接口。因此，只要 Xilinx 的 PCIe 块正常运行，无论 lanes 或其 link speed 的数量如何，IP core 都可以正常工作。

如果 PCIe 块配置了少量的 lanes，与 link speed 组合，使其带宽能力低于 Xillybus IP core 的，它仍然可以正常工作。在这种情况下，Xillybus 的 streams 提供的总带宽大约等于 PCIe 模块设置所施加的带宽限制。这是什么成为瓶颈的问题。

4.5.2 工作程序

对于 Versal ACAP FPGAs，请参阅 4.3.8 部分。以下描述适用于所有其他 FPGAs。

原则上，更改 lanes 和/或 link speed 的数量包括根据需要更改 PCIe 块的配置。但是有几个问题需要注意：

- 修改可能会影响 PCIe 块的其他参数，从而可能导致其无法正常运行。其中，Xilinx 的 GUI 工具中存在一个需要注意的错误，详情如下。
- 修改可能会更改驱动 PCIe 模块（PIPE clocks）的 clocks 的频率，因此需要更改 timing constraints。
- clock 频率的上述变化可能还需要对支持 PCIe 模块（pipe_clock 模块的 instantiation，如适用）的 Verilog 代码进行更改，否则 PCIe 模块将不起作用。

因此，阶段如下：

1. 在活动项目上制作 XCO 或 XCI 文件的副本（即在 Vivado 或 ISE 根据需要升级 IP 之后）。这将允许之后将更改与 diff 工具进行比较，并发现不需要的更改（如果发生这种情况）。
2. 在 Vivado（或 ISE）中打开 PCIe 块的 IP 进行配置（使用 Versal FPGAs，在 CIPS IP 中打开 CPM unit，这是 pcie_versal block design 中唯一的块）。
3. 根据需要更改 lanes 和/或 Maximum Link Speed 的数量，同时注意不要更改 (AXI) interface width。如果可以通过选择适合该任务的 lanes 和 link speed 的组合来避免更改 (AXI) Interface Frequency，那是可取的。
4. 进行所需的更改后，确认 Vendor ID 和 Device ID 没有更改（Subsystem 对应项都没有）。由于不相关的修改，Vivado 的某些修订版可能会将某些参数重置为默认值（这是一个错误）。
5. 确认更改（通常单击对话框底部的 “OK”）。如果在确认后建议这样做，则无需生成输出产品。

6. 使用文本 diff 工具比较更新和以前的 XCO 或 XCI 文件，并验证只有相关参数发生了变化。更多关于这下面。
7. 调整 xillybus.v 和 xillydemo.v/.vhd 中 PCIE_* 的信号矢量宽度，使其反映 lanes 的新编号。
8. 如有必要，调整 PIPE clock 模块的 instantiation，如下面的 4.5.3 段所述。
9. 如有必要，调整 timing constraints，如下面的 4.5.4 段所述。
10. 更新 PIPE clock 模块，如 4.5.5 段所述。

使用 Ultrascale 及更高版本时不需要最后三个步骤。

在比较新旧 XCI 文件时，应特别注意 `PARAM_VALUE.Device_ID`，因为它经常被意外更改。

XCI 文件的参数差异应与所需的匹配。这是根据 Vivado 中所做更改可接受更改的可能参数的简短列表。参数的名称应有所保留，因为 Vivado 的不同修订版（因此 PCIe 块的不同修订版）可能代表具有不同 XML 参数的 PCIe 块的属性。

- lanes数量相关：
  - `PARAM_VALUE.Maximum_Link_Width`
  - `MODELPARAM_VALUE.max_lnk_wdt`
- 与 link speed 相关：
  - `PARAM_VALUE.Link_Speed`
  - `PARAM_VALUE.Trgt_Link_Speed`
  - `MODELPARAM_VALUE.c_gen1`
  - `MODELPARAM_VALUE.max_lnk_spd`
- 与接口频率有关。这些参数的变化强烈表明第 4.5.3 和 4.5.4 段中的阶段是必要的。
  - `PARAM_VALUE.User_Clk_Freq`
  - `MODELPARAM_VALUE.pci_exp_int_freq`

4.5.3 PIPE频率变了吗?

使用 Ultrascale FPGAs 及更高版本时，不需要以下注意事项和操作，因为它们的 PCIe 块将 timing constraints 作为 IP 本身的一个组成部分提供。 PIPE 模块也是如此。

至于其他 FPGA 系列，请务必验证 PIPE clock 设置是否正确，如下所示：

更改后为 PCIe 块生成示例项目，并运行该项目的 synthesis。在 Vivado 中，这通常通过右键单击项目源层次结构中的 PCIe 块并选择 “Open IP Example Design...” 来完成。为 design 选择一个位置并生成后，单击左侧栏中的 “Run Synthesis” 启动 synthesis。

接下来，在 synthesis report 中获取 PIPE clock 模块的 instantiation parameters （在 Vivado 中，它被发现类似于 pcie_example/pcie_example.runs/synth_1/runme.log ）。在此报告中，搜索如下段：

```
INFO: [Synth 8-638] synthesizing module 'example_pipe_clock' [...]
Parameter PCIE_ASYNC_EN bound to: FALSE - type: string
Parameter PCIE_TXBUF_EN bound to: FALSE - type: string
Parameter PCIE_CLK_SHARING_EN bound to: FALSE - type: string
Parameter PCIE_LANE bound to: 4 - type: integer
Parameter PCIE_LINK_SPEED bound to: 3 - type: integer
Parameter PCIE_REFCLK_FREQ bound to: 0 - type: integer
Parameter PCIE_USERCLK1_FREQ bound to: 4 - type: integer
Parameter PCIE_USERCLK2_FREQ bound to: 4 - type: integer
Parameter PCIE_OOBCLK_MODE bound to: 1 - type: integer
Parameter PCIE_DEBUG_MODE bound to: 0 - type: integer
```

此报告中的参数必须与 pipe_clock 的 instantiation 中的参数匹配，因为它们出现在 xillybus.v 中，其形式为：

```
pcie_[...]_pipe_clock #
   (
    .PCIE_ASYNC_EN                  ( "FALSE" ),
    .PCIE_TXBUF_EN                  ( "FALSE" ),
    .PCIE_LANE                      ( 6'h08 ),
    .PCIE_LINK_SPEED               ( 3 ),
    .PCIE_REFCLK_FREQ               ( 0 ),
    .PCIE_USERCLK1_FREQ             ( 4 ),
    .PCIE_USERCLK2_FREQ             ( 4 ),
    .PCIE_DEBUG_MODE                ( 0 )
    )
   pipe_clock
     (
       [ ... ]
      );
```

要比较的三个参数是 PCIE_LINK_SPEED、PCIE_USERCLK1_FREQ 和 PCIE_USERCLK2_FREQ，它们必须匹配。如果他们这样做（如示例中所示），则所有设置都正确，包括 timing constraints。如果不是，则必须采取两项措施：

- xillybus.v 中的 instantiation parameters 必须更新以匹配示例项目的 synthesis report 中的那些。
- timing constraints 必须适应示例项目。这更加困难，因为未能正确执行此操作不一定会立即导致问题，但可能会影响 design 的可靠性。

如果 xillybus.v 中的 PCIE_LANE 参数大于示例项目的参数，那么就没有问题，而且这样做通常更容易。

4.5.4 适配 timing constraints

如果发生这种情况，必须调整 timing constraints 以反映 PCIe 块的 clocks 的变化。

由于 constraints 取决于所选的 FPGA 以及 Vivado 的修订版，因此正确完成此操作可能有些困难。避免这种调整是尝试通过选择 link speed 和 lanes 数量的组合（如果可能）来保持 PIPE clock 频率不变的主要动机。然而，即使 PIPE clock 的频率保持不变，更新 constraints 可能仍然是必要的。

再一次，当使用 Ultrascale 系列（及更高版本）的 FPGA 时，无需处理 timing constraints，因为其 PCIe 块的 IPs 在内部处理此问题。

为了调整timing constraints，首先找到示例项目的constraints。对于 Vivado，它通常是

example.srcs/constrs_1/imports/example_design/xilinx_*.xdc 形式的文件。

强烈建议在更改 lanes 和/或 link speed 的数量之前生成一个带有 PCIe 块设置的示例项目，并在这些更改之后生成一个示例项目。两个示例项目的 constraint 文件之间的简单 diff 给出了是否需要调整 constraints 以及如果需要，以何种方式调整的明确答案。

将 constraint 文件的 “Timing constraints” 部分与 xillydemo.xdc 中的部分进行比较。示例项目通过 logic 层次结构中的绝对位置选择 logic 元素，因此需要进行一些编辑。例如，假设示例项目中有这样的 timing constraint：

```
set_false_path -to [get_pins {pcie_vivado_support_i/pipe_clock_i/
pclk_i1_bufgctrl.pclk_i1/S0}]
```

在 xillydemo.xdc 中应该写成：

```
set_false_path -to [get_pins -match_style ucf */pipe_clock/
pclk_i1_bufgctrl.pclk_i1/S0]
```

主要区别在于 Xillybus 的 constraints 中使用的相对路径。可能还存在其他细微差别，因为某些 constraints 对于 Xilinx 工具的早期版本是必需的，而对于后来的工具则变得多余。

在 timing constraints 中进行更改后，重要的是通过在 design 的 implementation 之后查看 timing report 来验证它们是否对 logic 生效。

最后，值得解释的是以下两个 constraints，它们存在于一些 demo bundles 的 xillydemo.xdc 中：

```
set_case_analysis 1 [get_pins -match_style ucf */pipe_clock/
pclk_i1_bufgctrl.pclk_i1/S0]
set_case_analysis 0 [get_pins -match_style ucf */pipe_clock/
pclk_i1_bufgctrl.pclk_i1/S1]
```

这些 constraints 是 Gen1 PCIe 块所必需的，当使用 Vivado 的相当旧的版本时，如 Xilinx AR #62296 中所述。因此，最近的 Xilinx 工具可能会省略它们。

4.5.5 更新 PIPE clock 模块

如上所述，Ultrascale FPGAs 及更高版本不需要此步骤。

在某些情况下，特别是当从 2.5 GT/s (Gen1) 增加 link speed 或增加 2.5 GT/s (Gen1) 时，需要更新位于 vivado-essentials/ 目录中的 pcie_*_vivado_pipe_clock.v 文件。

该文件是通过执行 xillydemo-vivado.tcl 作为初始项目设置的一部分自动生成的。它可能会根据 PCIe 块的配置而略有变化，特别是它是否仅限于 2.5 GT/s。

推荐的方法是使用 xillydemo-vivado.tcl script 重新生成 Vivado 项目。即，从一个新的 demo bundle 开始，并将在前几个阶段更改的文件复制到其中：

- PCIe 块的 XCI 文件
- xillydemo.xdc
- 为适应新的 lanes 和 link speed 数量而编辑的 Verilog / VHDL 文件。

将这些文件复制到新的 demo bundle 中，使用 xillydemo-vivado.tcl script 生成项目可确保 PIPE clock module 与 PCIe 块的设置一致，并且项目不依赖于更改之前的任何剩余文件。

或者，从 4.5.3 段落中创建的示例项目更新 pcie_*_vivado_pipe_clock.v 文件。要使用的文件与 vivado-essentials/ 中的文件具有完全相同的名称，并且通常位于示例项目的文件层次结构的深处。将此文件复制到 vivado-essentials/（覆盖现有文件）。

4.6 更改 FPGA 部件号

从一个 FPGA 系列迁移到另一个系列时，必须从不同的 demo bundle 开始。存在与 PCIe 块（以及带有 XillyUSB 的 MGT 块）相关的差异（有时是细微的）。这些差异需要不同的 Xillybus IP core 以及不同的 wrapper modules。

尝试仅更改 Vivado / ISE 中的项目部分可能会导致项目在 implementation 期间出现错误。即使 implementation 成功，logic 也可能无法工作，或者工作不可靠。

然而，当保留在同一个 FPGA 系列中时，更改部件号通常就足够了（以及上面提到的关于引脚放置和 constraints 的注意事项）。

需要注意的是，对于某些 FPGA 系列（特别是 Ultrascale），PCIe 块在 logic fabric 中的位置（site）是 PCIe 块本身的属性，因此可能需要修改。此外，每个特定的 FPGA 和每个特定的 package 都有自己的一组有效 sites。因此，如果为 PCIe 块选择的 site 在新的 FPGA 上不存在，则 FPGA 的更改可能会重置 PCIe IP 的属性。

Vivado 对 PCIe 块位置 (site) 中的无效站点的反应非常具有破坏性。如果是这种情况，PCIe 块的 “upgrade”（在更改 FPGA 后解锁 IP 始终需要的操作）会导致将 PCIe 块的多个属性重置为任意值。这样做时，会生成如下所示的 Critical Warning：

```
CRITICAL WARNING: [IP_Flow 19-3419] Update of 'pcie_ku' to current project
options has resulted in an incomplete parameterization. Please review the
message log, and recustomize this instance before continuing with your design.
```

在不注意这个问题的情况下尝试项目的 implementation 是完全没有意义的，不仅会导致大量误导 warnings、Critical Warnings 和可能的错误，而且结果（如果有的话）将远远不能正常工作。

解决方案是在更改项目的部件号属性之前，分配一个在新 FPGA 上有效的 PCIe 块站点。如果更改前后 FPGA 部件之间没有公共站点，则可能需要手动编辑 XCI 文件（因为在这种情况下，无法在 GUI 中进行此更改，这会将设置限制为允许的站点在当前的 FPGA 部分）。

5

故障排除

5.1 implementation 期间的错误

Xilinx 工具版本之间的细微差异有时会导致无法运行 implementation 来创建 bitfile。

如果问题没有很快解决，请通过 Xillybus 网站上提供的电子邮件地址寻求帮助。请附上失败进程的输出日志，特别是在工具报告的第一个错误附近。此外，如果在 design 中进行了自定义更改（即从 demo bundle 转移），请详细说明这些更改。另请说明使用了哪个版本的 Vivado （或 ISE ）。

5.2 PCIe 硬件问题

正常情况下，host 的 BIOS 和/或操作系统正确检测到 PCIe 卡，host 的 driver 启动成功。

在大多数 PC 计算机上，BIOS 会在 boot 进程的早期简要显示检测到的外围设备列表。当成功检测到 Xillybus 接口时，列表中会出现一个带有 vendor ID 10EE 和 device ID EBEB 的外设。

至于操作系统对卡的检测，请参考以下两个文档之一，以适用者为准：

- Getting started with Xillybus on a Linux host
- Getting started with Xillybus on a Windows host

检测卡失败（或电脑的boot进程出现故障）与Xillybus IP core无关，Xillybus IP core依赖Xilinx的PCIe IP core与bus对接。

首先，建议验证以下内容：

- bitstream 在计算机开机时已经加载到 FPGA 中（或者在开机后不久，根据 PCI-SIG 规范）。
- PCIe 的 pinouts 线，包括 reference clock 是正确的（这可以在 placement report 中验证）。
- 该板为 FPGA 提供正确的 reference clock。

如果没有立即发现问题，建议尝试开发板随附的 PCIe 示例项目。这可能会显示错误的跳线设置和可能有缺陷的硬件。

如果使用此样本检测到卡，但未使用 Xillybus，则比较两个 designs 的 pinouts 可能会有所帮助。如果它们相等，下一步是通过调用 IP GUI 来比较 Xilinx 的 PCIe cores 的属性（在打开每个项目的情况下双击 Project Manager 中的 XCI / XCO 元素）。

以下配置元素可能需要调整:

- reference clock 的频率（在 GUI 中可能显示为 “Interface Frequency”）。
- base class 和 sub class（不太可能，但如果认为 class 未知，一些相对较旧的 PC 计算机在 boot 进程中失败）。
- 除基地址 register 设置、vendor ID、device ID 和中断设置外，任何其他配置不同的属性，不应更改。

如果问题仍然存在，请通过 Xillybus 网站上提供的电子邮件地址寻求帮助。