(机器翻译成中文)

Getting started with Xillybus on a Windows host

Xillybus Ltd.

www.xillybus.com

Version 4.1

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 This document may also be slightly outdated in relation to the original.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

本指南介绍了安装 driver 的步骤，以便在 Windows host 上运行 Xillybus / XillyUSB。还介绍了如何尝试 IP core 的基本功能。

本指南假设基于 Xillybus 的 demo bundle 的 bitstream 已加载到 FPGA 中，并且 FPGA 已被 host（通过 PCI Express 或 USB 3.x）识别为 peripheral。

以下文档之一概述了达到此阶段的步骤（取决于所选的 FPGA）：

- Getting started with the FPGA demo bundle for Xilinx
- Getting started with the FPGA demo bundle for Intel FPGA

host driver 生成 device files，其行为类似于 named pipes： 这些 device files 的打开、读取和写入就像任何文件一样。但是，这些文件在进程之间的行为方式与 pipes 类似。此行为也与 TCP/IP streams 类似。对于运行在host上的程序来说，不同的是，stream的另一端不是另一台process（在同一台计算机上或网络上的不同计算机上），而是另一端是FPGA内部的FIFO。就像 TCP/IP stream 一样，Xillybus stream 旨在高效地进行高速数据传输，但 stream 在偶尔传输少量数据时也能表现良好。

host 上的一个 driver 与通过 PCIe 与 host 通信的所有 Xillybus IP cores 一起使用。XillyUSB 与不同的 driver 一起使用。

当 FPGA 中使用不同的 IP core 时，无需更改 driver： 当 driver 加载到 host 的操作系统中时，driver 会自动检测 streams 及其属性。 device files 相应地创建，文件名格式为 \\.\xillybus_*something*。同样，driver for XillyUSB 创建格式为 \\.\xillyusb_00_*something* 的 device files。在这些文件名中，00部分是device的索引。当多于一台XillyUSB device同时连接电脑时，该部分替换为01、02等。

有关 host 相关主题的更深入信息，请参阅 Xillybus host application programming guide for Windows。

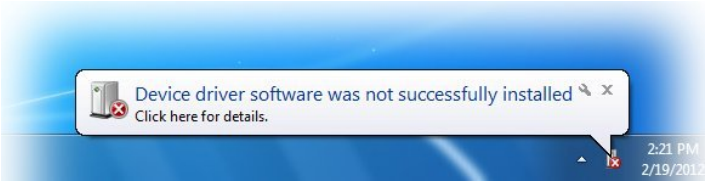
2

安装 host driver

2.1 安装程序

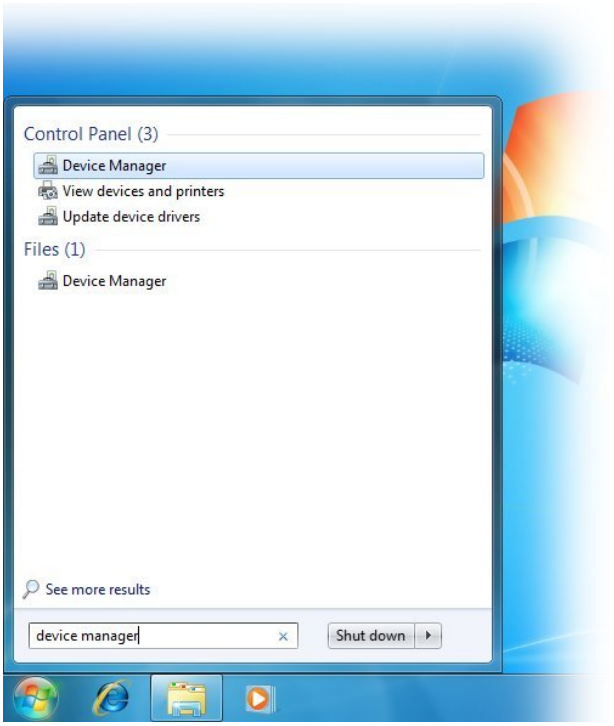
对于 Xillybus 或 XillyUSB 安装 Windows driver 没有什么特别的。下面描述的过程是从磁盘上的特定位置安装 device driver 的常用方法。

Windows在boot过程中第一次检测到PCIe Xillybus IP core时，很可能会出现如下所示的warning bubble：

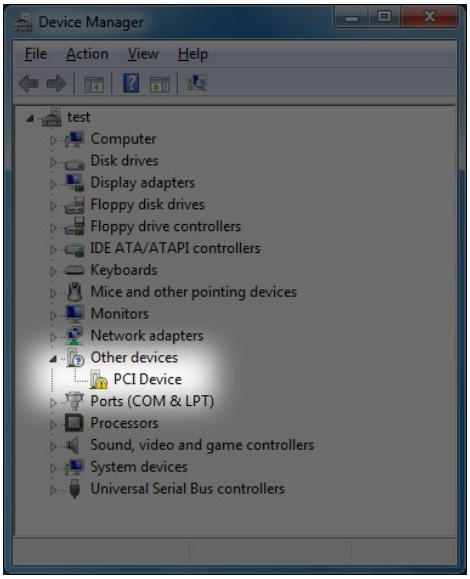


该通知表示已发现新硬件，但尚未安装相关的 driver。这种情况是正常的，并且是 Windows 检测到它尚未识别的东西的标志。当 XillyUSB device 首次连接到计算机时，预计会出现相同的结果。

为了响应此事件，首先运行 Device Manager。最简单的方法是单击 “Windows start” 按钮，然后输入 “device manager”，如下图所示。之后，单击顶部的菜单项。



打开的 Device Manager 看起来像这样（突出显示的重要部分）：



此屏幕截图与 PCIe 场景有关。对于 XillyUSB，“Universal Serial Bus controllers” 组

中会出现一个新项目。

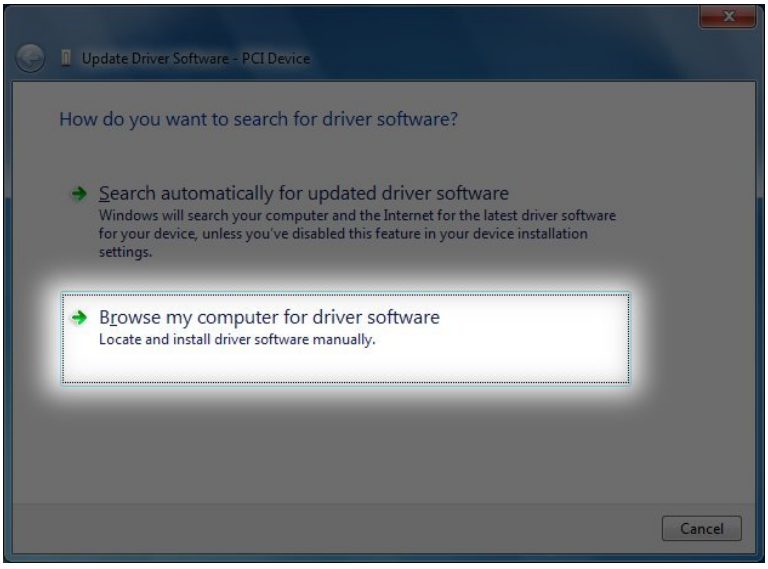
如果 Device Manager 中没有出现新条目，可能有以下几种原因：

- FPGA 加载了错误的 bitstream，或者根本没有加载任何 bitstream。
- 如果FPGA board从PC接收电源，则PC computer上电时将bitstream加载到FPGA中。在这种使用场景下，BIOS有可能在boot期间没有检测到PCIe接口，因为FPGA加载bitstream太慢： 当 BIOS 初始化计算机时，bitstream 必须已位于 FPGA 内部。

  执行 Windows restart（不关闭计算机）是解决此问题的安全方法。然而，执行 Action > Scan for New Hardware 也可能有效。
- board（jumpers、DIP switches 等）配置错误、pin assignment 错误、reference clock 频率错误等。

  请注意，此类问题是因为未检测到 FPGA 上的 PCIe block。此类问题与Xillybus无关，Xillybus使用此PCIe block（由AMD或Intel提供）作为与PCIe bus的接口。
- Xillybus / XillyUSB driver 已安装。在这种情况下，Device Manager 应类似于安装过程末尾所示的示例。

右键单击 “PCI Device” 项目并选择 “Update Driver Software...”。将打开以下窗口：



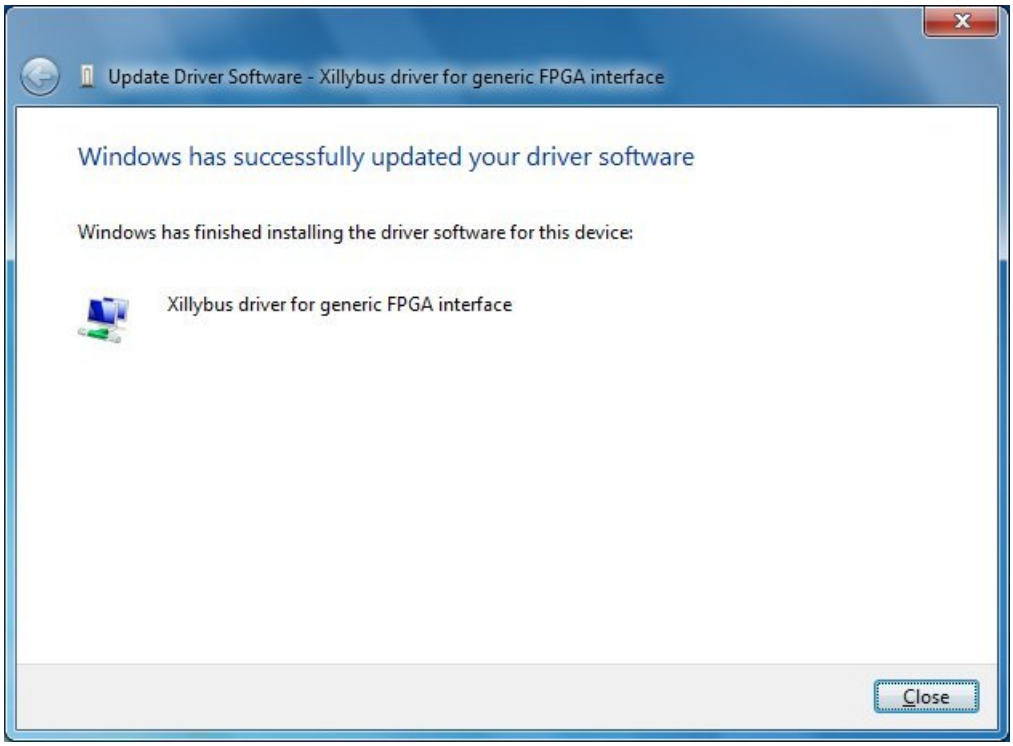
选择“Browse my computer for driver software”。这是下一个窗口：

使用 “Browse...” 按钮，导航到 driver 的存储位置（driver 解压缩后）。

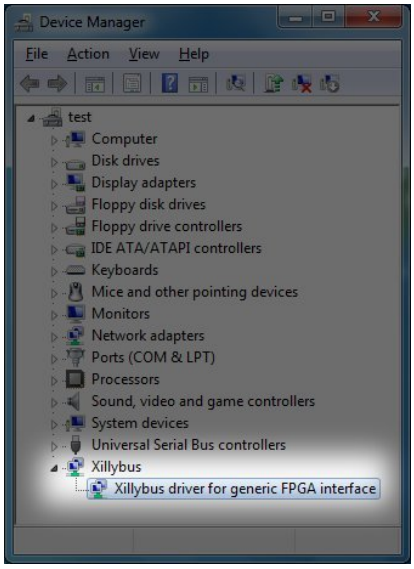
下一步是确认安装：

单击““Install””。在 Windows 7 上安装 driver 的过程需要 10-20 秒。较新版本的 Windows 通常需要更少的时间。

以下窗口宣布安装成功完成：



Device Manager 现在将显示新安装的 device：



至此，driver已安装完毕，并已自动加载到系统中。每次系统在PCIe bus上用Xillybus

IP core执行boot时都会加载这个driver。每次相关设备连接到 host 时，都会加载 XillyUSB 的 driver。

建议将 Event Viewer 设置为显示 Xillybus 的 log messages，如下所述。

上面显示的屏幕截图与 PCIe 的 Xillybus 相关。然而，XillyUSB 的过程是相同的，只有细微的差别。特别是，Device Manager 中的新组称为 “XillyUSB”，而不是 “Xillybus”。

2.2 device files

应用计算机程序通过标准file I/O API与Xillybus IP core进行通信。但 device files 不是访问常规文件，而是用于与 Xillybus 配合使用。

因此，Xillybus 的 driver 的目的是在操作系统内创建这些 device files，作为与 FPGA 通信的机制。这些device files在Microsoft的术语中被称为“Windows objects”。

直接从简单的计算机程序访问 Windows objects 似乎是一种不可靠的方法。不过熟悉Microsoft Windows内部的人都知道，软件与硬件的接口往往就是这样完成的。这些device files直接暴露于应用软件确实不常见。相反，硬件制造商通常会提供 DLL，允许程序通过 API 访问硬件。在幕后，DLL 使用 device files 来完成所需的功能。

因为Xillybus的接口非常简单，所以不需要这样的DLL。因此user application software直接访问device files。但不幸的是，Windows并没有提供简单的方法来获取Xillybus的device files的列表。这是因为访问 Windows objects 被认为是一种高级技术。因此，有必要下载实用程序才能从操作系统获取此信息。但如下所述，device files 的列表也可以从其他来源获得。

WinObj 实用程序（可在 Microsoft 网站下载）允许在 Window 的 object tree 中导航。 Xillybus / XillyUSB device files 在 “subdirectory” 中的名称为 symbolic links，名称为 GLOBAL??。其他知名的 Windows objects 也可以在同一地点找到，例如 C: 和 COM1:。

还有一个command-line工具，其名称为accesschk。该工具可以从Microsoft's website下载。获取Xillybus / XillyUSB device files名称的命令为：

```
> accesschk -o \\GLOBAL\?\?
```

请注意，此操作列出了许多其他全局 device files。

尽管可以使用这两个工具获取 device files 的列表，但没有必要这样做： device files的名字是预先知道的。

当使用 PCIe 接口时，Xillybus 的 device files 具有 `\\.\xillybus_` 形式的前缀。对于 XillyUSB，前缀是 `\\.\xillyusb_nn_`。

这是由 demo bundle 的 PCIe 变体生成的 device files 的列表：

- `\\.\xillybus_read_8`
- `\\.\xillybus_write_8`
- `\\.\xillybus_read_32`
- `\\.\xillybus_write_32`
- `\\.\xillybus_mem_8`

对于连接到 host 的单个 XillyUSB 设备，这些是 device files：

- `\\.\xillyusb_00_read_8`
- `\\.\xillyusb_00_write_8`
- `\\.\xillyusb_00_read_32`
- `\\.\xillyusb_00_write_32`
- `\\.\xillyusb_00_mem_8`

至于在IP Core Factory上生成的定制IP core： device files 的列表可以在 README 文件中找到，该文件包含在下载的 zip 文件中。

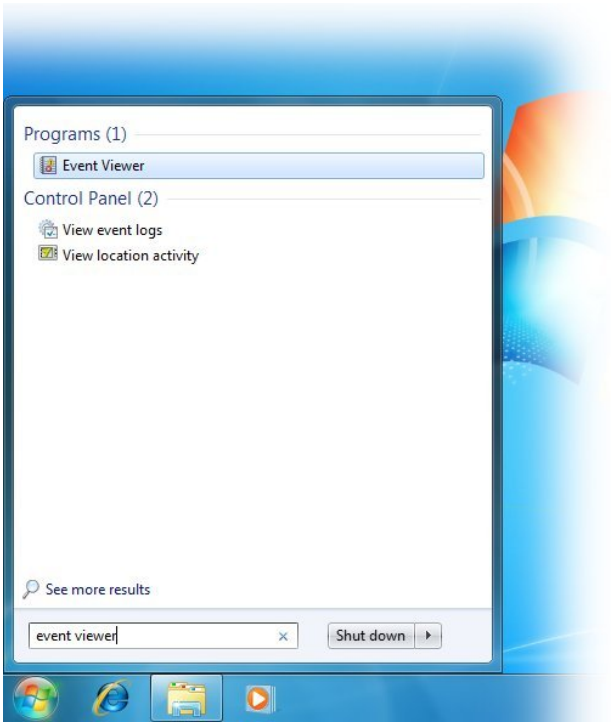
请注意，在许多编程语言（例如 C/C++ ）中，文件名中的 backslashes 之前需要 escape character。因此可能需要将device file的名称写为例如`\\\\.\\xillybus_read_8`。

2.3 获取诊断信息

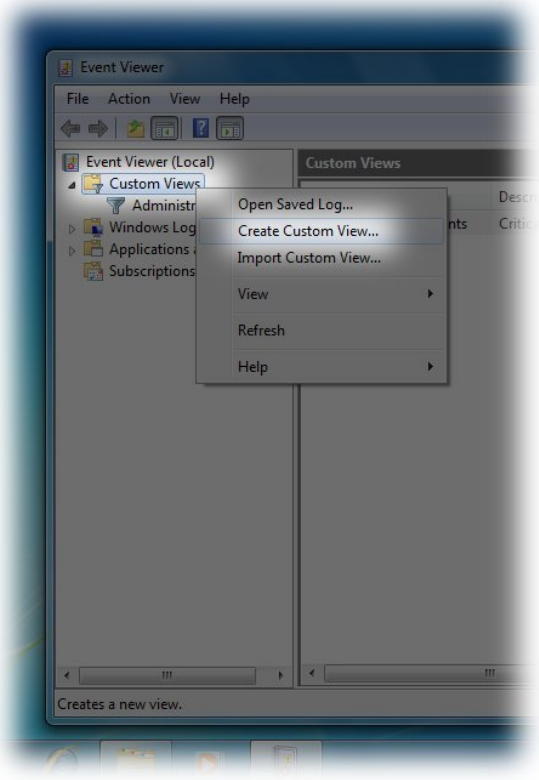
Xillybus / XillyUSB 的 driver 将诊断消息发送到操作系统的主 event logger。这些消息包括有关 driver 初始化失败时出现的问题的信息（例如，没有足够的内存用于 DMA buffers）。还发送有助于解决问题的其他消息。

接下来概述的过程显示了如何在 Event Viewer 中创建 custom view。此任务的目的是显示与 Xillybus 或 XillyUSB 相关的消息。

首先，打开Event Viewer。最简单的方法是单击 “Windows start” 按钮，然后键入 “event viewer”，如下所示。单击顶部的菜单项：

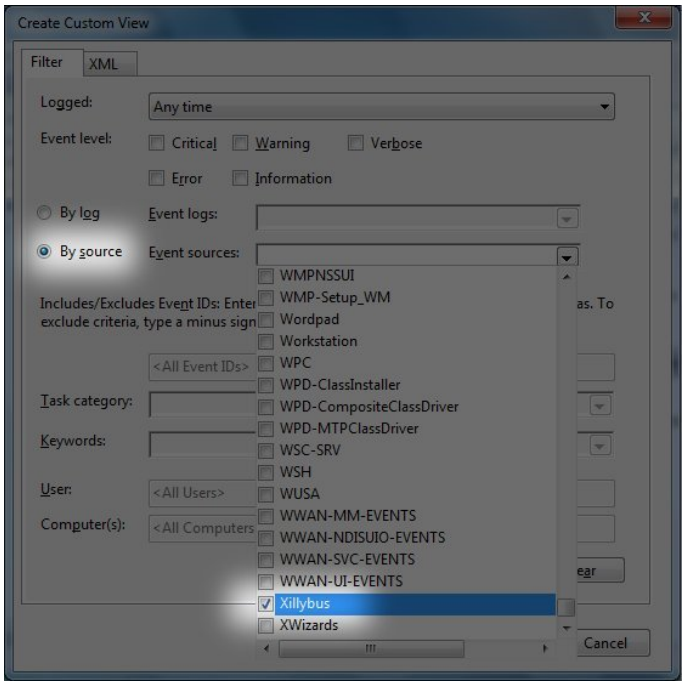


Event Viewer 将打开。右键单击 “Custom Views”，然后从菜单中选择 “Create Custom View...”。



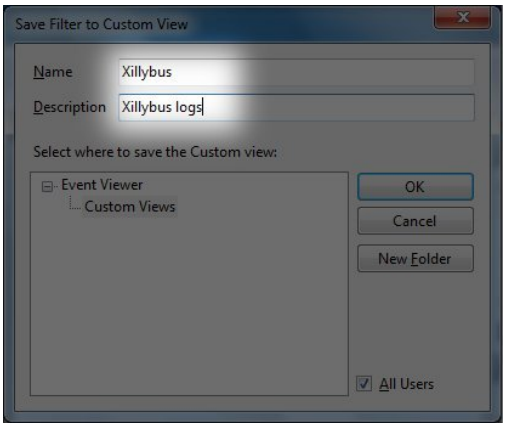
将打开一个标题为 “Create Custom View” 的窗口。此窗口的目的是定义选择显示哪些消息的过滤器。选择“By source”。在drop-down menu中选择Xillybus。保留其他选项的默认值。

要从 XillyUSB 获取消息，请在菜单中选择 XillyUSB。

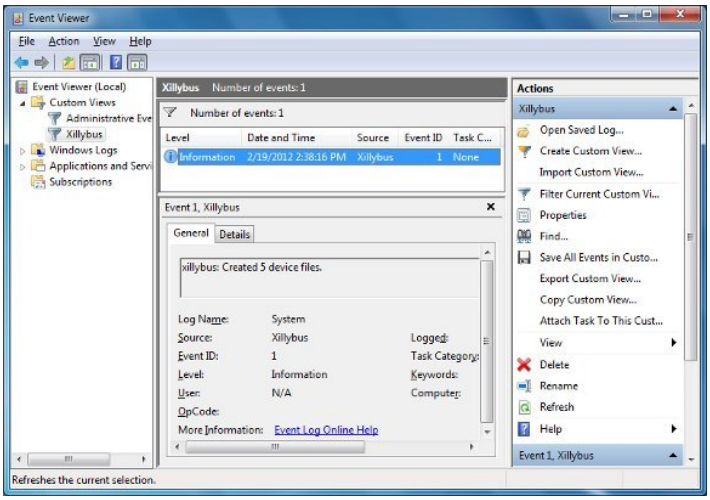


如果在drop-down menu中找不到Xillybus / XillyUSB条目，请检查driver是否正确安装。

单击 “OK” 后，将打开另一个窗口，用于为此 custom view 分配名称和描述。这是个人选择的问题：



点击 “OK” 后，Event Viewer 看起来是这样的：



上图显示了一条消息，通知 Xillybus 已正常启动，并且已创建 5 个 device files。这是成功安装 driver 后，当 FPGA 加载 demo bundle 时应该立即看到的情况。

在计算机的 reboot 上，消息不会被删除。因此，此 custom view 显示自安装 driver 以来的历史记录（除非故意删除历史记录）。

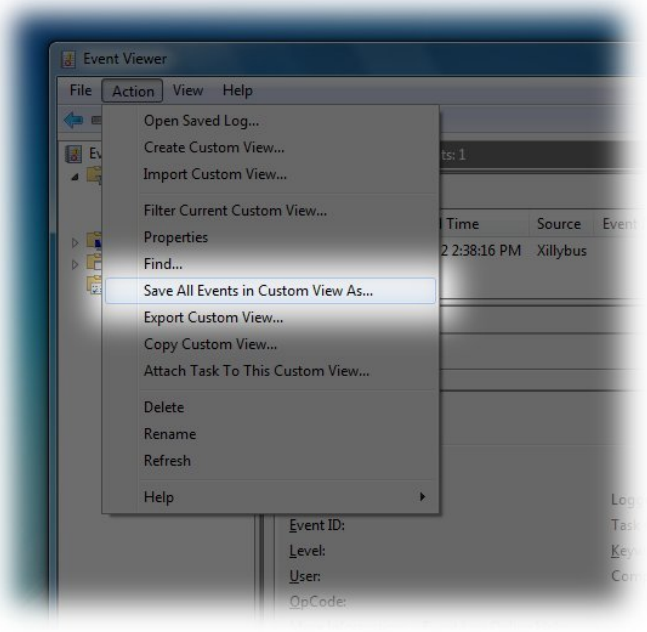
来自 PCIe driver 的消息列表及其解释可在以下位置找到：

http://xillybus.com/doc/list-of-kernel-messages

然而，通过在消息文本上使用 Google 可能更容易找到特定消息。

还可以将消息导出到文件中。请求支持时，最好发送包含来自 driver 的消息的文件。这些消息通常包含有价值的信息。

为了创建这样的文件，请选择 “Action” 菜单项，然后选择 “Save All Events in a Custom View As...”：



将打开一个文件选择窗口。为了寻求帮助，请选择 CSV 作为输出格式。

3

“Hello, world” 测试

3.1 目标

Xillybus 是一款旨在作为 logic design 中的构建块的工具。因此，了解 Xillybus 功能的最佳方法是将其与您自己的 user application logic 集成。 demo bundle 的目的是成为使用 Xillybus 的起点。

因此，在 demo bundle 中实现了最简单的应用： 两个 device files 之间的 loopback。这是通过将 FIFO 的两侧连接到 FPGA 中的 Xillybus IP Core 来实现的。这样，当host向一台device file写入数据时，FPGA会通过另一台device file将相同的数据返回给host。

接下来的几节将解释如何测试这个简单的功能。此测试是验证Xillybus是否正常运行的简单方法： FPGA 中的 IP Core 按预期工作，host 正确检测到 PCIe 外设，并且 driver 已正确安装。最重要的是，这次测试也是通过对FPGA中的logic design进行小修改来了解Xillybus如何工作的机会。

作为第一步，建议使用 demo bundle 进行简单的实验，以了解 FPGA 中的 logic 和 device files 如何协同工作。仅这一点就足以阐明如何使用 Xillybus 来满足您自己的应用程序的需求。

除了上面提到的 loopback 之外，demo bundle 还实现了 RAM 和额外的 loopback。下面简要讨论这个附加的 loopback。对于RAM，它演示了如何访问内存阵列或registers。有关此内容的更多信息，请参见 3.4 部分。

3.2 准备工作

“Hello, world” 测试包括使用 Command Prompt 窗口运行简单的 command-line 程序。

第一步，下载 Xillybus package for Windows。它是一个 zip 文件，可在提供 driver 的同一网页上找到。该 zip 文件包含这些程序的 source code 以及准备运行的 executable binaries。

最简单的方法是使用executable binaries来进行“Hello world”测试。但是，也可以执行这些程序的 compilation，如 4.2 节中详细介绍的。

另一种可能性是创建一个类似于 Linux 的工作环境，如 4.3 部分中所述。如果选择这种可能性，请按照 Getting started with Xillybus on a Linux host 指南中的 “Hello world” 测试说明进行操作。

3.3 琐碎的 loopback 测试

接下来显示了使用两个 Command Prompt 窗口进行 loopback 测试的简单示例。

打开常规 Command Prompt 窗口并将目录更改为 Xillybus package for Windows 的 “precompiled-demoapps” 子目录。为了使用您自己的 compilation 的结果（如 4.2 部分所示），请将目录更改为 XP32_DEBUG。

在此 Command Prompt 窗口中键入以下内容：

```
> streamread \\.\xillybus_read_8
```

这使得 “streamread” 程序打印出它从 xillybus_read_8 device file 读取的所有内容。预计现阶段不会发生任何事情。

请注意，backslashes 不重复。如果这是用 Cygwin 完成的（而不是在普通的 Command Prompt 窗口中），那么它将是 `\\\\.\\xillybus_read_8`。

现在打开另一个 Command Prompt 窗口。切换到与第一个 Command Prompt 相同的目录，然后输入：

```
> streamwrite \\.\xillybus_write_8
```

此命令将在第二个窗口中键入的任何内容发送到 device file （即 `\\.\xillybus_write_8` ）。

在第二个 Command Prompt 窗口中键入一些文本，然后按 ENTER。相同的文本将出现在第一个 Command Prompt 窗口中。请注意，在按下 ENTER 之前不会发送任何内容。这符合 standard input 的预期行为。

如果在尝试这两个命令时收到错误消息，建议执行以下操作：

- 检查是否有拼写错误。

- 验证 driver 是否已安装并且 FPGA 是否被检测为 Xillybus 设备： 打开 Device Manager 并与 2.1 部分中的最后一个图像进行比较。
- 检查 Event Viewer 中的错误，如 2.3 部分中所述。
- 确保已创建 device files，如 2.2 部分中所述（搜索 `xillybus_read_8` 和 `xillybus_write_8`）。

请注意，FPGA 内的 FIFOs 不会面临 overflow 或 underflow 的风险： core 尊重 FPGA 内部的 'full' 和 'empty' 信号。必要时，Xillybus driver 会强制计算机程序等待，直到 FIFO 为 I/O 做好准备。这称为blocking，意思是强制user space program休眠。

另请注意，streamwrite 单独作用于每一行，并且在按下 ENTER 之前不会向 FPGA 发送任何内容。这与 Linux 的同名程序不同。

还有另一对 device files，它们之间有一个 loopback：xillybus_read_32 和 xillybus_write_32。这些 device files 使用 32 位字，FPGA 内部的 FIFO 也是如此。因此，使用这些 device files 进行 “hello world” 测试将导致类似的行为，但有一个区别： 所有I/O都是以4个字节为一组进行的。因此，当输入未达到 4 字节边界时，输入的最后一个字节将保持不传输状态。

3.4 内存接口

memread和memwrite程序更有趣，因为它们演示了如何访问FPGA上的内存。这是通过对 device file 上的 _lseek() 进行函数调用来实现的。 Xillybus host application programming guide for Windows中有一节解释了这个API与Xillybus的device files的关系。

请注意，在demo bundle中，只有xillybus_mem_8允许seeking。这台device file也是唯一一款既可以读又可以写的。

本节使用一个名为“hexdump”的工具来显示FPGA、RAM的内容。该工具可以在Xillybus package for Windows的“unixutils”子目录中找到。或者，4.3 部分建议了获取此工具的其他选项。

在写入内存之前，可以使用hexdump观察现有情况。

```
> hexdump -C -v -n 32 \\.\xillybus_mem_8
00000000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000020
```

此输出是内存数组中的前 32 个字节：hexdump打开xillybus_mem_8并从这个device file读取32个字节。当打开允许_lseek()的文件时，初始位置始终为零。因此，输出由存储器阵列中的数据组成，从位置 0 到位置 31。

您的输出可能会有所不同： 此输出反映了 FPGA 的 RAM，其中可能包含其他值。特别是，由于之前使用 RAM 进行的实验，这些值可能不同于零。

简单说一下hexdump的flags： 上面显示的输出格式是“-C”和“-v”的结果。 “-n 32” 表示仅显示前 32 个字节。内存数组只有 32 个字节长，因此读取超过这个字节是没有意义的。

memwrite 可用于更改数组中的值。例如，使用以下命令将地址 3 处的值更改为 170（hex 格式的 0xaa）：

```
> memwrite \\.\xillybus_mem_8 3 170
```

为了验证该命令是否有效，可以重复上面的 hexdump 命令：

```
> hexdump -C -v -n 32 \\.\xillybus_mem_8
00000000  00 00 00 aa 00 00 00 00  00 00 00 00 00 00 00 00  |...Ãª............|
00000010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000020
```

显然，该命令起作用了。

在memwrite.c中，重要的部分是写着“_lseek(fd, address, SEEK_SET)”的地方。该函数调用更改 device file 的位置。因此，这会更改在 FPGA 内部访问的数组元素的地址。后续的读操作或写操作都从该位置开始。每次此类访问都会根据传输的字节数递增位置。

允许 seeking 的 device file 对于轻松向 FPGA 发送配置命令也很有用。这是通过将 registers 而不是 memory array 放入 FPGA 来完成的。这是 register 接收新值以响应对地址 2 的写入操作的示例:

```
reg [7:0] my_register;

always @(posedge bus_clk)
  if (user_w_mem_8_wren && (user_mem_8_addr == 2))
    my_register <= user_w_mem_8_data;
```

同样,可以通过在 FPGA 的 logic 中使用 “case” 语句来读回 registers 的值。

4

host 应用示例

4.1 一般的

有六个C程序演示了如何访问Xillybus的device files。这些程序可以在 Xillybus package for Windows 中找到，Xillybus package for Windows 是一个 zip 文件，可以在提供 driver 的同一网页上下载。

在该文件中，“precompiled-demoapps” 子目录中有 precompiled executables。

source code 可以在 “demoapps” 子目录中找到。这些 C 程序适用于 Microsoft’s Visual C++ compiler，可以作为 Microsoft 和 SDK 的一部分免费下载。这些程序也可以与 Visual Studio 一起使用。

也可以在 Cygwin 中运行示例程序（请参阅 4.3 部分）。 MinGW也可以用于此目的。如果选择这两个工具之一，则应使用 Linux 的 source code，并遵循 Getting started with Xillybus on a Linux host 中的说明。另请参阅 4.4 部分，了解有关用 `\\\\.\\` 替换 /dev/ 前缀的信息。

“demoapps”子目录包含以下文件：

- Makefile——该文件包含 “nmake” 实用程序用于程序 compilation 的规则。
- streamread.c– 从文件中读取，将数据发送到 standard output。
- streamwrite.c– 从 standard input 读取数据，发送到文件。
- memread.c– 执行 seek 后读取数据。演示如何访问 FPGA 中的内存接口。
- memwrite.c——执行seek后写入数据。演示如何访问 FPGA 中的内存接口。
- fifo.c——演示 userspace RAM FIFO 的实现。这个程序很少有用，因为device file的RAM buffers可以配置为足以满足几乎所有场景。因此，fifo.c 仅在非常高的数据速率以及 RAM buffer 需要非常大（即多个 gigabytes）时有用。

- winstreamread.c——从文件中读取数据，将数据发送到standard output。该程序的作用与streamread.c相同，但winstreamread.c使用并演示了Microsoft的file I/O API，而不是标准的API。

所有程序（除了 winstreamread.c ）都是以经典的 Linux 风格编写的，尽管它们是针对 compilation 和 Microsoft 的 compiler 设计的。

这些程序的目的是显示正确的 coding style。它们也可以用作编写您自己的程序的基础。然而，这些程序都不适用于现实生活中的应用，特别是因为这些程序在高数据速率下表现不佳。有关实现高带宽性能的指南，请参阅 5 章。

这些程序非常简单，仅演示访问文件的标准方法。这些方法在 Xillybus host application programming guide for Windows中进行了详细讨论。由于这些原因，这里没有对这些程序进行详细解释。

请注意，这些程序使用低级 API，例如 _open()、_read() 和 _write()。避免使用更知名的 API（fopen()、fread()、fwrite() 等），因为它依赖于由 C runtime library 维护的 data buffers。这些 data buffers 可能会造成混乱，特别是因为与 FPGA 的通信经常被 runtime library 延迟。

4.2 Compilation

正如已经提到的，不需要 compilation 来尝试示例程序： Xillybus package for Windows 包含可在 Windows 计算机上运行的文件。但显然，compilation的这些方案是有必要做出改变的。

那些习惯使用Microsoft Visual Studio的人可能会更喜欢使用这个compiler，并且知道如何使用这个工具。示例程序是简单的 Command Prompt 应用程序。

但是，以下指南基于 Microsoft 和 software development kit (SDK) 7.1。这是一个古老而简单的开发套件，可以免费下载。以下说明基于该软件，主要是因为完成任务需要少量步骤。

下载并安装Windows SDK 7.1。在“Start menu”中打开Program Files并选择Microsoft Windows SDK v7.1 > Windows SDK 7.1 Command Prompt。这将打开一个 Command Prompt 窗口，其中配置了多个用于 compilation 的 environment variables。该窗口中的文本以黄色字体显示。

将目录更改为 C 文件所在的位置：

```
> cd \path\to\demoapps
```

要在所有程序上运行 compilation，请键入 “nmake”。预计会有以下文字记录：

```
> nmake

Microsoft (R) Program Maintenance Utility Version 10.00.30319.01
Copyright (C) Microsoft Corporation.  All rights reserved.

    if not exist "XP32_DEBUG/" mkdir XP32_DEBUG
    cl -D_CRT_SECURE_NO_WARNINGS -c -DCRTAPI1=_cdecl -DCRTAPI2=_cdecl -nologo  [ ..
    link  /INCREMENTAL:NO /NOLOGO -subsystem:console,5.01 -out:XP32_DEBUG\ [ ... ]
    cl -D_CRT_SECURE_NO_WARNINGS -c -DCRTAPI1=_cdecl -DCRTAPI2=_cdecl -nologo  [ ..
    link  /INCREMENTAL:NO /NOLOGO -subsystem:console,5.01 -out:XP32_DEBUG\ [ ... ]
    cl -D_CRT_SECURE_NO_WARNINGS -c -DCRTAPI1=_cdecl -DCRTAPI2=_cdecl -nologo  [ ..
    link  /INCREMENTAL:NO /NOLOGO -subsystem:console,5.01 -out:XP32_DEBUG\ [ ... ]
    cl -D_CRT_SECURE_NO_WARNINGS -c -DCRTAPI1=_cdecl -DCRTAPI2=_cdecl -nologo  [ ..
    link  /INCREMENTAL:NO /NOLOGO -subsystem:console,5.01 -out:XP32_DEBUG\ [ ... ]
    cl -D_CRT_SECURE_NO_WARNINGS -c -DCRTAPI1=_cdecl -DCRTAPI2=_cdecl -nologo  [ ..
    link  /INCREMENTAL:NO /NOLOGO -subsystem:console,5.01 -out:XP32_DEBUG\ [ ... ]
    cl -D_CRT_SECURE_NO_WARNINGS -c -DCRTAPI1=_cdecl -DCRTAPI2=_cdecl -nologo  [ ..
    link  /INCREMENTAL:NO /NOLOGO -subsystem:console,5.01 -out:XP32_DEBUG\ [ ... ]
```

以 “cl” 开头的六行是 “nmake” 为了使用 compiler 所请求的命令。这些命令可单独用于compilation 的程序。然而，没有理由这样做。就用“nmake”吧。 “link” 命令也是如此，它在 object files 和 libraries 上执行 linking，从而创建 executables。

executables（和 object files）可以在 XP32_DEBUG 子目录中找到。如有必要，该子目录会在 compilation 进程期间创建。顾名思义，这些文件适用于 32-bit Windows XP。但是，executables 可在 Windows 的更高版本上运行，包括 64 位版本。

“nmake” 实用程序仅在必要时运行 compilation。如果仅更改一个文件，“nmake” 将仅请求该文件的 compilation。所以正常的工作方式是编辑你要编辑的文件，然后用“nmake”换recompilation。不会发生不必要的compilation。

使用 “nmake clean” 来删除由先前的 compilation 生成的 executables。

如上所述，Makefile 包含 compilation 的规则。该文件的语法并不简单，但幸运的是，通常只需使用常识即可对该文件进行更改。

Makefile 涉及与 Makefile 本身位于同一目录中的文件。因此，可以制作整个目录的副本，并处理该副本内的文件。目录的两个副本不会互相干扰。

还可以添加一个C文件并轻松更改Makefile，使“nmake”也运行这个新文件的compilation。

4.3 将 Linux 中的工具与 Windows 结合使用

使用 Linux 的人员倾向于使用标准 command line 工具来执行简单的任务。这些工

具对于主要使用 Windows 的人来说不太了解。不幸的是，主要原因是 Windows 的 command-line 工具不如每台 Linux 计算机上存在的工具有用。

如前所述，可以按照 Getting started with Xillybus on a Linux host 中的详细说明执行 “Hello world” 测试，而不是按照本指南中的说明进行。为了在 Windows 中做到这一点，有必要在计算机上提供一些可用的工具。有几种替代方案可以实现这一目标：

- 从 Gnuwin32 项目下载并安装两个软件包： Coreutils 和 Util-Linux-NG。这两个软件包满足了 “Hello world” 测试的需求（还提供了此任务不需要的程序）。请注意，即使使用Gnuwin32的setup工具安装这些软件包，Command Prompt的execution path也不会发生变化。
- 使用Xillybus提供的Windows工具： 这些可以在 Xillybus package for Windows 的 “unixutils” 子目录中找到。通过这种方式获得的工具是从 Gnuwin32 软件包中选择的，以满足 “Hello world” 测试的需要。
- 安装 Cygwin。选择这种方法意味着安装一个提供类似于 Linux 的 command-line 接口的整个系统。此类安装可能包括 GNU C compiler 和其他软件开发工具。对于那些习惯使用 Linux 和 command-line 的人来说，这是推荐的选择。

4.4 与Linux的差异

在 Windows 计算机上执行 Getting started with Xillybus on a Linux host 中所述的 “Hello world” 测试时，需要注意一些差异。

最重要的区别是path与device files是`\\.\`，而不是`/dev/`。例如，当 Linux 的指南提到 `/dev/xillybus_read_8` 时，Windows 的正确文件名是 `\\.\xillybus_read_8`。

由于device file的名称中包含backslashes，因此在某些情况下需要escape characters： backslash 本身通常被视为 escape character。因此，文件名中的每个 backslash 需要两个 backslashes。也就是说`\\.\`需要写成`\\\\.\\`。例如，当Linux的指南提到`/dev/xillybus_read_8`时，在某些情况下应该使用文件名`\\\\.\\xillybus_read_8`。

但情况并非总是如此： 当从 Command Prompt 执行程序时，不需要 escape characters。 Command Prompt 像对待任何其他 character 一样对待 backslash。

在大多数编程语言中，都需要额外的backslashes。 scripts 内部可能需要额外的 backslashes。这取决于 arguments 在 script 内部的处理方式。

4.5 Cygwin的警告信息

Cygwin 的 command-line 接口需要额外的反斜杠。然而，当第一次使用`\\\\.\\`前缀时，Cygwin可能会显示如下警告：

```
$ cat \\\\.\\xillybus_read_8
cygwin warning:
  MS-DOS style path detected: \\.\xillybus_read_8
  Preferred POSIX equivalent is: //./xillybus_read_8
  CYGWIN environment variable option "nodosfilewarning" turns off this warning.
  Consult the user's guide for more details about POSIX paths:
    \url{http://cygwin.com/cygwin-ug-net/using.html#using-pathnames}
```

应忽略此警告。

Xillybus已经与Cygwin进行了广泛的测试，上面显示的访问device files的方法是正确的。对于普通文件名，使用正斜杠确实是一个更好的主意。但Cygwin并没有将//./翻译成\\.\。因此，必须使用 backslashes。

为了避免此警告，请遵循有关 environment variable 的警告消息中的建议。

5

高带宽性能指南

Xillybus和IP cores的用户经常进行数据带宽测试，以确保确实达到宣传的数据传输速率。实现这些目标需要避免可能大大减慢数据流的瓶颈。

本节是指南的集合，它基于最常见的错误。遵循这些准则应该会导致带宽测量结果等于或略好于公布的结果。

当然，在基于Xillybus的项目实施过程中遵循这些准则非常重要，这样该项目才能充分利用IP core的功能。

通常的问题是 host 处理数据的速度不够快： 错误测量数据速率是抱怨无法达到公布数据的最常见原因。推荐的方法是使用Linux的“dd”命令，如下面的5.3部分所示。该工具在 Xillybus package for Windows 中作为 executable 提供。

本节中的信息对于 “Getting Started” 指南来说相对较先进。此讨论还引用了其他文档中解释的高级主题。尽管如此，本指南还是给出了这些指导原则，因为许多用户在熟悉 IP core 的早期阶段就进行了性能测试。

5.1 不要loopback

在 demo bundle（FPGA 内部）中，两对 streams 之间有一个 loopback。这使得 “Hello, world” 测试成为可能（参见 3 部分），但这对测试性能不利。

问题在于，Xillybus IP core 的数据传输突发很快就会充满 FPGA 内的 FIFO。由于该 FIFO 已满，因此数据流会暂时停止。

loopback是用这个FIFO实现的，所以这个FIFO的两侧都连接到IP core。为了响应 FIFO 中存在数据，IP core 从 FIFO 获取该数据并将其发送回 host。这也发生得非常快，所以 FIFO 就变空了。数据流再次暂时停止。

由于数据流中的这些短暂暂停，测得的数据传输速率低于预期。发生这种情况是因为

FIFO 太浅，并且 IP core 负责填充和清空 FIFO。

在现实场景中，不存在 loopback。相反，FIFO 的另一侧有 application logic。让我们考虑一下获得最大数据传输速率的使用场景： 在这种情况下，application logic 消耗来自 FIFO 的数据的速度与 IP core 填充该 FIFO 的速度一样快。因此，FIFO 永远不会满。

同样对于相反的方向： application logic 填充 FIFO 的速度与 IP core 消耗数据的速度一样快。因此，FIFO 永远不会是空的。

从功能角度来看，FIFO偶尔满或空是没有问题的。这只会导致数据流暂时停止。一切正常，只是不是以最大速度运行。

demo bundle 可以轻松修改以进行性能测试： 例如，为了测试 `\\.\xillybus_read_32`，请将 user_r_read_32_empty 与 FPGA 内部的 FIFO 断开。相反，将此 signal 连接到常量零。因此，IP core 会认为 FIFO 永远不会为空。因此，数据传输以最大速度执行。

这意味着 IP core 偶尔会从空的 FIFO 中读取数据。因此，到达 host 的数据并不总是有效（由于 underflow 的原因）。但对于速度测试来说，这并不重要。如果数据的内容很重要，一个可能的解决方案是 application logic 尽快填充 FIFO （例如，使用 counter 的输出）。

同样测试 `\\.\xillybus_write_32`： 断开 user_w_write_32_full 与 FIFO 的连接，并将该 signal 连接到常量零。 IP core 会认为 FIFO 永远不会满，因此数据传输以最大速度执行。发送到FIFO的数据会因overflow而部分丢失。

请注意，断开 loopback 允许单独测试每个方向。然而，这也是同时测试两个方向的正确方法。

5.2 不涉及磁盘或其他存储

磁盘、solid-state drives 和其他类型的计算机存储通常是无法满足带宽期望的原因。高估存储介质的速度是一个常见的错误。

操作系统的 cache 机制增加了混乱： 当数据写入磁盘时，并不总是涉及物理存储介质。相反，数据被写入 RAM。只有稍后这些数据才会写入磁盘本身。磁盘的读取操作也可能不涉及物理介质。当最近已读取相同数据时会发生这种情况。

cache 在现代计算机上可能非常大。因此，在磁盘的实际速度限制变得可见之前，可以传输几个 Gigabytes 的数据。这常常导致用户认为 Xillybus 的数据传输出现问题：对于数据传输速率的突然变化没有其他解释。

对于 solid-state drives (flash)，还有一个额外的混乱来源，特别是在长时间连续的写入操作期间： 在 flash drive 的低级实现中，必须擦除未使用的内存段 (blocks)，为写

入 flash 做准备。这是因为只有被擦除的 blocks 才允许向 flash memory 写入数据。

首先，flash drive 通常有很多 blocks 已被擦除。这使得写操作变得更快：有很多空间可以写入数据。然而，当不再有擦除的blocks时，flash drive被迫擦除blocks并且可能执行数据的defragmentation。这可能会导致明显的减速，而且没有明显的解释。

由于这些原因，测试 Xillybus 的带宽不应涉及任何存储介质。即使存储介质在短期测试中看起来足够快，这也可能会产生误导。

通过测量将数据从 Xillybus device file 复制到磁盘上的大文件所需的时间来估计性能是一个常见的错误。尽管此操作在功能上是正确的，但以这种方式测量性能可能会完全错误。

如果存储旨在作为应用程序的一部分（例如 data acquisition），建议彻底测试此存储介质：应对存储介质进行广泛、长期的测试，以验证其是否满足预期。较短的 benchmark test 可能会产生极大的误导。

5.3 读取和写入大部分内容

对 _read() 和 _write() 的每个函数调用都会生成操作系统的 system call。因此需要大量的 CPU cycles 来执行这些函数调用。因此，重要的是 buffer 的尺寸足够大，以便减少 system calls 的执行次数。对于带宽测试和高性能应用程序来说都是如此。

通常，128 kB 对于每个函数调用的 buffer 来说是一个合适的大小。这意味着每个此类函数调用的最大数量限制为 128 kB。然而，这些函数调用允许传输较少的数据。

需要注意的是，4.1 和 3.3（streamread 和 streamwrite）部分中提到的示例程序不适合测量性能：这些程序中的 buffer 大小为 128 字节（不是 kB）。这简化了示例，但使程序对于性能测试来说太慢。

在 Cygwin 中可以使用以下 shell 命令进行快速速度检查（根据需要替换 `\\\\.\\xillybus\_*` 名称）：

```
dd if=/dev/zero of=\\\\.\\xillybus_sink bs=128k
dd if=\\\\.\\xillybus_source of=/dev/null bs=128k
```

这些命令会一直运行，直到被 CTRL-C 停止为止。添加“count=”以进行固定数据量的测试。

有关使用 Cygwin 进行测试的更多信息，请参阅 4.3 部分。

5.4 注意CPU的消耗

在高数据速率的应用中，计算机程序通常是瓶颈，而不一定是数据传输。

高估 CPU 的功能是一个常见的错误。与普遍看法不同的是，当数据速率高于 100-200 MB/s 时，即使是最快的 CPUs 也难以对数据执行任何有意义的操作。 multi-threading 可以提高性能，但令人惊讶的是，这是必要的。

有时，buffers 的尺寸不足（如上所述）也会导致 CPU 消耗过多。

因此，密切关注 CPU 的功耗非常重要。例如，Task Manager 即可用于此目的。然而，该程序显示的信息在具有多个 processor cores 的计算机（即当今几乎所有计算机）上可能会产生误导。例如，如果有四个processor cores，那么25% CPU意味着什么？是低功耗的CPU，还是特定thread上的100%？ system monitoring 的不同工具以不同的方式显示此信息。

5.5 不要让读写相互依赖

当需要双向通信时，仅使用一台 thread 编写计算机程序是一个常见的错误。该程序通常有一个循环，既执行读取又执行写入： 对于每次迭代，数据朝 FPGA 写入，然后以相反方向读取数据。

有时这样的程序没有问题，例如如果两个 streams 功能独立。然而，此类程序背后的意图通常是 FPGA 应该执行 coprocessing。这种编程风格基于这样的误解：程序应该发送一部分数据进行处理，然后读回结果。因此，迭代构成了对每部分数据的处理。

这种方法不仅效率低下，而且程序经常会卡住。 Xillybus host application programming guide for Windows 的 6.5 节详细阐述了这个主题，并提出了更充分的编程技术。

5.6 了解 host 和 RAM 的限制

这在使用 revision XL / XXL IP core 时最为相关： 主板（或 embedded processor ）和 DDR RAM 之间的数据带宽有限。在计算机的日常使用中很少注意到这种限制。但对于 Xillybus 的要求非常高的应用来说，这个限制可能会成为瓶颈。

请记住，每次从 FPGA 到 user space program 的数据传输都需要在 RAM 上进行两次操作： 第一个操作是 FPGA 将数据写入 DMA buffer。第二个操作是 driver 将此数据复制到 user space program 可以访问的 buffer 中。出于类似的原因，当数据以相反方向传输时，也需要对RAM进行两次操作。

DMA buffers 和 user space buffers 之间的分离是操作系统要求的。所有使用_read()和_write()（或类似函数调用）的I/O都必须以这种方式进行。

例如，XL IP core 的测试预计会在每个方向产生 3.5 GB/s，即总共 7 GB/s。然而，RAM 的访问量是其两倍。因此，RAM 的带宽要求是 14 GB/s。并非所有主板都具有此功能。另请记住，host 同时使用 RAM 执行其他任务。

出于同样的原因，对于修订版 XXL，即使是一个方向的简单测试也可能超出 RAM 的带宽能力。

5.7 DMA buffers 足够大

这很少是一个问题，但仍然值得一提： 如果在 host 上为 DMA buffers 分配的 RAM 太少，可能会减慢数据传输速度。原因是host被迫将data stream分成小段。这就造成了CPU cycles的浪费。

所有demo bundles都有足够的DMA内存用于性能测试。对于在 IP Core Factory 上正确生成的 IP cores 也是如此： “Autoset Internals” 已启用，“Expected BW” 反映了所需的数据带宽。 “Buffering” 应选择为 10 ms，即使任何选项都很可能都可以。

一般来说，这对于带宽测试来说已经足够了： 至少有四个DMA buffers，其中RAM的总量对应于10 ms期间的数据传输。当然，必须考虑所需的数据传输速率。

5.8 使用正确的数据字宽度

很明显，对于 FPGA 内的每个 clock cycle，application logic 只能向 IP core 传输一个字的数据。因此，由于数据字的宽度和 bus_clk 的频率，数据传输速率受到限制。

除此之外，还有一个与默认版本 (revision A IP cores) 的 IP cores 相关的限制： 当字宽为 8 位或 16 位时，PCIe 的功能不能像字宽为 32 位时那样有效地使用。因此，需要高性能的应用程序和测试应仅使用 32 位。这不适用于 revision B IP cores 及更高版本。

从版本 B 开始，字宽最多可达 256 位。该字的宽度至少应与 PCIe block 的宽度一样。因此，对于数据带宽测试，需要这些数据字宽：

- 默认版本（Revision A）： 32 位。
- Revision B： 至少 64 位。
- Revision XL： 至少 128 位。
- Revision XXL： 256 位。

如果数据字比上面要求的宽（如果可能的话），通常会获得稍好的结果。原因是 application logic 和 IP core 之间的数据传输得到了改进。

5.9 参数调整

选择 demo bundles 中 PCIe block 的参数是为了支持宣传的数据传输速率。该性能是

在配备 x86 系列 CPU 的典型计算机上测试的。

此外，IP Core Factory 中生成的 IP cores 通常不需要任何微调： 启用 “Autoset Internals” 时，streams 可能会在 FPGA 的性能和资源利用率之间实现最佳平衡。因此，每个 stream 都能确保所需的数据传输速率。

因此，尝试微调 PCIe block 或 IP core 的参数几乎总是毫无意义的。对于 IP cores (revision A) 的默认版本，此类调整始终毫无意义。如果这样的调整提高了性能，则问题很可能是 application logic 或 user application software 中的缺陷。在这种情况下，纠正这个缺陷可以获得更多好处。

然而，在需要卓越性能的极少数情况下，可能需要稍微调整 PCIe block 的参数才能获得所需的数据速率。这对于从 host 到 FPGA 的 streams 尤其重要。 The guide to defining a custom Xillybus IP core 的 4.5 节讨论了如何执行此调整。

请注意，即使这种微调是有益的，也不会修改 Xillybus IP core 的参数。仅调整PCIe block。尝试通过调整 IP core 的参数来提高数据传输速率是一个常见的错误。相反，问题几乎总是本章上面提到的问题之一。

6

故障排除

drivers for Xillybus / XillyUSB 的设计目的是在系统的 event log 中产生有意义的 log messages。因此，建议在出现问题时搜索相关消息。这是通过在 event log 上应用滤波器来完成的，如 2.3 部分所述。

即使一切看起来都工作正常，也建议偶尔查看一下 event log。

来自 PCIe driver 的消息列表及其解释可在以下位置找到：

http://xillybus.com/doc/list-of-kernel-messages

然而，通过在消息文本上使用 Google 可能更容易找到特定消息。