(机器翻译成中文)

The guide to defining a custom Xillybus IP core

Xillybus Ltd.

www.xillybus.com

Version 3.1

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 This document may also be slightly outdated in relation to the original.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 **3**

1.1 一般的 3

1.2 如何集成自定义IP core 4

2 定义自定义 IP cores **5**

2.1 概述 5

2.2 device file 的名称 6

2.3 数据宽度 6

2.4 同步或异步 stream 7

2.5 缓冲时间 7

2.6 DMA buffers尺寸 8

2.7 DMA acceleration 10

3 可扩展性和 logic 资源消耗 **11**

3.1 一般的 11

3.2 Block RAMs 11

3.3 logic fabric的资源 12

4 IP cores 修订版 B、XL 和 XXL **15**

4.1 一般的 15

4.2 使用修订版 B/XL/XXL 16

4.3 数据字宽度 16

4.4 Logic 资源消耗 17

4.5 将 stream 的最佳带宽从 host 调整到 FPGA 20

1

介绍

1.1 一般的

Xillybus 是一个适用于各种应用的多用途平台。因此，每个用户都可以轻松创建和下载满足特定要求的自定义 IP core：streams 的数量、方向、与其性能和资源消耗相关的属性。

为了简化自定义 IP cores 的定义和创建，可以使用在线工具：IP Core Factory (http://xillybus.com/custom-ip-factory)。

该工具包含一个简单的 Web 应用程序，允许用户定义请求的 device files 及其配置。定义完成后，自动过程会生成包含在 FPGA 项目中的文件。稍后（通常是几分钟）后，自定义 IP core 即可作为 zip 文件下载。

下载的自定义 IP core 功能齐全。在实际应用中测试和使用此 IP core 没有技术限制。

Web 应用程序可以在不阅读本指南的情况下使用，但建议您先通过运行 demo bundle 来熟悉 Xillybus。希望更好地了解和控制 device files 属性的用户可以在本指南中找到一些背景信息。

对于尚未熟悉 demo bundle 的用户，建议先阅读以下部分文档：

- Getting started with the FPGA demo bundle for Xilinx
- Getting started with the FPGA demo bundle for Intel FPGA
- Getting started with Xillinux for Zynq-7000
- Getting started with Xillybus on a Linux host
- Getting started with Xillybus on a Windows host

即使明确需要定制 IP core，最好还是从 demo bundle 开始。这阐明了 IP core 如何与 application logic 集成，以及整个项目应该如何设置。

有关 IP core 自定义配置的所有信息都存储在 FPGA 中的 IP core 本身中。 host 上的 driver 在 driver 初始化时检索此信息。因此，更换 IP core 时无需更改 host 上的任何内容。

一个常见的错误是为了节省资源而尽量减少配置的 streams 的数量。 3 和 4.4 部分展示了 Xillybus IP core 如何扩展，并解释了为什么慷慨地分配 streams 是有意义的。

1.2 如何集成自定义IP core

从 IP Core Factory 下载自定义 IP core 后，需要修改 demo bundle 以包含此 IP core。这需要几个简单的步骤。这些说明是用 README.TXT 编写的，它是 IP Core 的 zip 文件的一部分。为方便起见，这些说明也列在下面。

此 README 文件还包含其他有用信息:

- Core ID，这是一个五位数字。此编号是 IP core 的唯一标识符。请求报价时应提及 Core ID。
- IP core的devices files都列出来了。还显示了有关每个 device file 的技术细节。这是关于IP core真实特性的准确信息。

为了将 custom IP core 集成到 demo bundle 中，请按照以下步骤操作:

1. 将 demo bundle 中的两个文件替换为 IP Core 的 zip 中的文件：xillybus.v 和 xillybus_core.v（或 xillybus_xl_core.v / xillybus_xxl_core.v）。
2. 更换 IP core 本身。该文件在 demo bundle 的 subdirectory 中，名称为 “core/”。要替换的文件类似于 xillybus_core.ngc、xillybus_core.edf、xillybus_core.qxp 或 xillybus_core.vqm。
3. 编辑 xillydemo.v（或 xillydemo.vhd）以便将所需的应用程序与此自定义 IP core 集成。如需指导，请查看名为 “instantiation templates” 的目录，该目录是 IP core 的 zip 文件的一部分。名为 template.v（或 template.vhd）的文件包含应遵循的 instantiation template。

2

定义自定义 IP cores

2.1 概述

IP Core Factory 是一个类似于 wizard 的 Web 应用程序，用于从头开始定义自定义 IP core，或使用 demo bundle 的 core 的配置作为起点。

对于绝大多数目的，建议依靠 IP Core Factory 来设置每个 stream 的属性，方法是保持 “Autoset internals” 选项处于启用状态。为了调整 stream 的参数而关闭此选项是一个非常常见的错误，这几乎总是会导致性能下降。

特别是，如果 IP core 未能达到预期的数据速率性能，则问题很可能出在其他地方。在这种情况下，建议参考以下两个指南之一，其中讨论了如何实现 IP core 的全部性能:

- Getting started with Xillybus on a Linux host中的第 5 节
- Getting started with Xillybus on a Windows host第5节

另一个常见的错误是关闭 “Autoset internals” 以调整 DMA buffers 的大小，使其与要传输的数据包的大小相匹配。这将在 2.6 节中讨论。

以下是使用此工具时值得强调的一些额外要点:

- 必须正确选择 IP core 所针对的 FPGA 系列，因为 IP core 作为 netlist 交付。
- 将每个 device file 的 “use” 属性设置为与预期用途相匹配的描述非常重要。这可确保正确设置 stream 的属性。
- 对于 XillyUSB IP cores，“Expected bandwidth” 属性应准确设置为 stream 请求的最大带宽，因为数据速率限制为该值。对于其他变体（ PCIe 和 AXI ），此属性仅影响性能调整。应该应用现实的数字，而不是试图通过夸大要求来获得更

好的结果。这种夸大可能会导致其他确实需要某些有限资源的 streams 的性能下降。

本节的其余部分讨论 device files 的一些属性。

2.2 device file 的名称

每个 stream 都被指定一个名称，该名称用作在 host 上创建的 device file 的名称。

名称总是采用 xillybus_* 的形式，例如 xillybus_mystream。对于 XillyUSB，名称类似于 xillyusb_NN_*，其中 NN 是一个索引——当只有一个 XillyUSB 设备连接到 host 时，通常两个零。

在 Linux 系统上，stream 作为普通文件打开，例如 /dev/xillybus_mystream。在 Windows 中，相同的 stream 显示为 \\.\xillybus_mystream。

一个 device file 可以代表两个方向相反的 streams，也就是两个 streams 恰好同名 device file。这两个 streams 可以在任一方向单独打开，也可以打开为读写。通常应避免此功能以防止混淆，但在将 device file 传递给需要双向 pipe 的软件时很有用。

2.3 数据宽度

数据宽度是从 FPGA 中读取或写入 FIFOs 的字的位数。允许的选择是 32、16 或 8 位。 B/XL/XXL 版本的 Xillybus IP cores（这些将在 4 部分讨论）以及 XillyUSB 允许更宽的数据宽度。

当 stream 需要高带宽性能时，并且当 IP core 的版本为 A for PCIe 或任何 IP core for AXI 时，数据宽度必须设置为 32 位：16 位和 8 位数据宽度的性能显着下降，导致底层传输（例如 PCIe bus 传输）的低效使用。

原因是字以 bus clock 的速率通过 Xillybus 的内部数据路径传输。因此，传输 8 位字与 32 位字占用相同的时隙，从而使其实际上慢了四倍。

这也会影响其他 streams 在给定时间竞争底层传输，因为数据路径被较慢的数据元素占用。

IP core 和 XillyUSB 的后续版本具有不同的内部数据路径结构，因此没有此限制。

无论如何，在 host application 中以与数据宽度匹配的粒度执行 I/O 操作是一种很好的做法，例如，如果数据宽度为 32 位，则使用数据长度为 4 的倍数的数据长度调用函数 read() 和 write()。

数据宽度选择不当可能会导致不良行为。例如，如果从 host 到 FPGA 的链接是 32 位宽，在 host 写入 3 个字节的数据将使 driver 在向 FPGA 发送任何内容之前无限期地

等待第四个字节。

2.4 同步或异步 stream

根据“use”设置的选择，选择“Autoset internals”选项时，将自动设置此属性。

在大多数情况下，异步 streams 足以用于连续数据流，而同步 streams 则足以用于命令、控制数据和获取状态信息。

对于同步 streams，所有 I/O（包括 FPGA 中的数据流）仅在对 read() 或 write() 的函数调用的调用和返回之间发生。这可以完全控制何时发生的事情，但在 CPU 执行其他操作时会保留未使用的数据传输资源。建议阅读这两个文档之一的第 2 节中有关此主题的详细说明：

- Xillybus host application programming guide for Linux
- Xillybus host application programming guide for Windows

使用同步 streams 使软件编程更直观，但对带宽利用率有负面影响。使用异步 streams，即使操作系统将 CPU 与在 user space 中运行一段时间的进程分开，也可以保持连续的数据流。

总结这个主题，这些是指导性问题：

- 对于 downstreams（host 到 FPGA）：write() 操作在数据到达 FPGA 之前返回是否可以？
- 对于 upstreams（FPGA 到 host）：在 host 中的 read() 操作请求数据之前，Xillybus IP core 是否可以开始从 FPGA 中的 user application logic 获取数据？

如果相应问题的答案是否定的，则需要同步 stream。否则，异步选项通常是首选，同时要理解对数据流的控制较少，而且它的直观性稍差。

2.5 缓冲时间

Xillybus 在 FPGA 和 host 之间保持连续 stream 数据的错觉。 DMA buffers的存在对于FPGA中的user application logic以及host上的应用软件都是透明的。它们仅对控制数据流的效率及其保持连续性的能力感兴趣，特别是在高数据速率下。

data acquisition 和 data playback 等应用程序需要 FPGA 上的连续数据流，否则数据会丢失。为了维持这个流程，user space application 需要足够频繁地对 read() 或 write() 进行函数调用，以防止 DMA buffers 因 FPGA 的活动而变满或变空（分别）。

然而，确保这些函数调用足够频繁存在一个问题：常见的操作系统，如 Linux 和 Windows，可能会在理论上任意时间段内剥夺 CPU 与任何 user-space application 的连接。无论如何，FPGA 都会不断填充或排出 driver 的 buffers。因此，DMA buffers 必须足够大，以维持连续的数据流，尽管 CPU 暂时被剥夺。

为方便讨论，缓冲时间是 stream 从所有 DMA buffers 都为空的状态变为所有 DMA buffers 都已满的状态所需的时间量，此时数据以速率填充 buffers stream 的预期用途（并且在此期间它们不会被耗尽）。

在启用 “Autoset internals”（推荐）设置 Xillybus stream 时，Web 应用程序中会出现一个标题为 “Buffering” 的选择框。这是选择所需缓冲时间的地方。

对于需要保持其连续性的异步 stream，所选时间应反映 CPU 可以从 user space application 中取出的预期最大时间。

选择 “Maximum” 告诉分配 buffers 的算法尝试分配尽可能多的 RAM，而对其他 streams 稍加考虑。

给定所需的缓冲时间 t 和预期带宽 W，该算法将尝试根据以下公式为 driver 的 DMA buffers 分配总量 RAM、M：

$$M = t \times W$$

然而，实际的 buffer 大小始终是 2 的幂（2^N）。也可能无法分配足够的内存来满足所需的缓冲时间。

因此，重要的是在 IP core 的 README 文件中查找分配的 buffer 大小，并验证它是否可以使用。手动设置 buffer 大小（即关闭 “Autoset internals”）可能需要强制 RAM 在 streams 中更好地分布，这更适合预期的应用程序。

2.6 DMA buffers尺寸

建议让工具通过在 Web 应用程序中启用 “Autoset internals” 来自动设置 DMA buffers 的参数（请参阅上面的 2.5 部分）。在某些情况下，自动设置可能不适合应用程序，在这种情况下，可以手动设置 DMA buffers 的大小和数量。

对于异步 streams，buffers 的参数有很大的影响，这在这两个文档中名为 “Continuous I/O at high rate” 的部分中进行了讨论：

- Xillybus host application programming guide for Linux
- Xillybus host application programming guide for Windows

无需使 DMA buffers 的大小适应 read() 和 write() 的预期函数调用的大小。正如这两个指南中所解释的，DMA buffers 的大小在对 read() 和 write() 的函数调用中是不相关且

透明的。特别是，如果有足够的数据到达 FPGA 中的 IP core（无论 DMA buffer 的填充水平如何），对 read() 的函数调用会立即返回。这要归功于 FPGA 和 host 之间的机制，它允许 FPGA 提交部分填充的 DMA buffer。当有助于立即完成对 read() 的函数调用时，使用此机制。

同样，通过明确的请求，可以确保从 host 到 FPGA 的数据立即到达 FPGA。

将 DMA buffers 的大小与预期数据交换的模式联系起来是一个常见的错误。有了 Xillybus，就不需要这样了，这也是为什么 “Autoset internals” 是设置 DMA buffers 尺寸的首选。

为了连续性，RAM 越多越好，因为 DMA buffers 中的总空间量即使在 CPU 被剥夺应用程序的情况下也能保持数据流的连续性。做出正确的决定涉及其他因素，这些因素在上面提到的编程指南中有详细说明。

然而，当 DMA buffers 的总尺寸过大时，buffering delay 存在风险，这是由于能够存储大量数据的结果。因此，当一侧填充 buffers 的速度快于另一侧清空 buffers 时，数据可能会在很长一段时间后到达另一端。这可以通过 Xillybus FPGA designer's guide中提到的技术来控制，在名为 “Monitoring the amount of buffered data” 的部分中。

对于 XillyUSB IP cores，每台 stream 对应一个 buffer，相当于一台由 driver 管理的大型 FIFO。其他 IP cores（ PCIe 和 AXI ）为每个 stream 维护多个 DMA buffers，因此它们的大小和数量都已定义。因此，DMA buffers 的有效尺寸是每个 DMA buffer 的尺寸乘以它们的数量。

因此，如果对基于 PCIe / AXI 的 IP cores 关闭 “Autoset internals”，则需要指定 DMA buffers 的数量和每个的大小。应考虑以下因素：

- 每个 DMA buffer 的大小在 streams 从 host 到 FPGA 都有自己的意义：当这些 buffers 已满时，数据被发送到 FPGA（除非软件明确请求 flush，或者 stream 空闲 10 毫秒）。因此，每个 DMA buffer 的大小都会对流动数据的典型 latency 产生影响。
- 对于慢速 streams（小于 10 MBytes/s），推荐的 DMA buffers 数量为 4。当需要更高的带宽时，选择 buffers 的数量以实现合适的整体 DMA buffer 分配。如果这允许每个 buffer 为 128 kBytes 或更少，则适用于高带宽 streams 的 DMA buffers 的足够数量在 16 到 64 之间。
- 所有 streams 的 DMA buffers 总分配不应超过 512 MBytes，除非在 host 上使用增强型 driver。否则，操作系统可能拒绝分配超过此值，导致 driver 初始化失败。
- 每次填充一个 buffer，就会向 host 发送一个 hardware interrupt。给定预期的数

据速率，应计算 interrupts 的速率并将其保持在与 processor 相同的水平（每秒不超过几千）

- 当通过增加它们的数量可以达到总大小时，每个 DMA Buffers 的大小不应超过 128 kBytes。

driver 的 DMA buffers 的问题对于同步 streams 来说不太重要。为此，经验法则是代表 stream 分配给 buffers 的总 RAM 应该是 read() 和 write() 的预期函数调用的数据长度的数量级。正如上面已经说过的，没有必要为这些函数调用调整 buffers 的大小，但是通过使它们变得更大来浪费 kernel RAM 很少有任何意义。

2.7 DMA acceleration

对于基于 PCIe 的 IP cores，从 host 到 FPGA 的 streams 可能需要加速 DMA 数据传输。

为了完成这个方向的数据交换，PCIe bus protocol 规定 FPGA 应该从 host 发出数据请求并等待数据到达。当请求在 bus 上传输、由 host 排队和处理以及数据传回时，会发生固有延迟。这种周转时间差距会导致 bus 的效率有所下降，有时会将单个 stream 的带宽降低到低至 40%。

为了解决这个问题，需要发送多个数据请求，以便 host 在连续传输期间始终有一个请求在其队列中。由于来自不同请求的数据可能以随机顺序到达，因此必须将其存储在 FPGA 上的 RAM buffers 中，以向 application logic 呈现有序的数据流。

FPGA 中的每个 buffer 都用于存储一段请求的数据。 DMA accelerations 当前可能的设置是

- 没有任何。 FPGA 上不存储任何数据。只有当所有数据都从前一个到达时，才会发送每个数据请求。
- 4 段，每段 512 字节。 FPGA 上分配了 2048 字节的 block RAM。在任何给定时刻最多可以激活四个数据请求。
- 8 段，每段 512 字节。 FPGA 上分配了 4096 字节的 block RAM。在任何给定时刻最多可以激活 8 个数据请求。
- 修订版 B 和更高版本的 IP cores 也有 16 段 512 字节的选项。

请求和数据到达之间的周转时间取决于 host 的硬件。因此，实际带宽性能可能会有所不同。

在 IP Core Factory 中使用 “Autoset internals” 时，加速资源的自动分配是基于典型 PC 计算机硬件上的测量结果，在极少数情况下可能需要手动细化。

3

可扩展性和 logic 资源消耗

3.1 一般的

Xillybus 的设计考虑了可扩展性。虽然为单个 stream 配置自定义 IP core 非常有意义，但扩展到大量 streams 对 Xillybus core 消耗的 logic 数量的影响相对较小。

为了测量 logic 的消耗，Xillybus IP core 的连续构建（基线，A 修订版）随着 streams 数量的增加而制作。在所有测试中，从 FPGA 到 host 的 streams 数量与另一个方向相同。 streams 的数量从 2 个（每个方向一个）到 64 个（每个方向 32 个）不等。

本节概述了 IP core 本身在 FPGAs 的三个系列上对 logic 的消耗，正如他们的工具所报告的那样。这些 FPGAs（由 Xilinx 提供）已经过时了，但是在更新的 FPGAs 上，由 Xilinx 和 Intel 获得了类似的结果。

有关修订版 B、XL 和 XXL 的 IP cores 的类似分析，请参阅 4 部分。

XillyUSB 不在任何这些分析中。

3.2 Block RAMs

Xillybus core 使用的 block RAMs 数量在零到几个之间变化（3 block RAMs 代表 64 个 streams）。每个 stream 的 Xillybus core 内部都没有 buffers。相反，Xillybus core 依靠与其连接的 FIFOs 来收集数据。在内部，core 有一个供所有 streams 使用的内存池。

随着streams数量的增加，block RAMs用于存储DMA buffers的地址。

从 host 到 FPGA 为 streams 为 DMA acceleration 分配了额外的 block RAMs，详见 core 的 README 文件。

3.3 logic fabric的资源

下图显示了 LUTs 和 registers (flip-flops) 的消耗，因为 streams 的数量从 2 变为 64。这些图中的每个点都是 synthesis 报告中所示的实际用途。从这些图中可以明显看出 logic 消耗量几乎呈线性增长。无论 FPGA 架构如何，每个 stream 平均增加了大约 110 个 LUTs 和 82 个 registers。

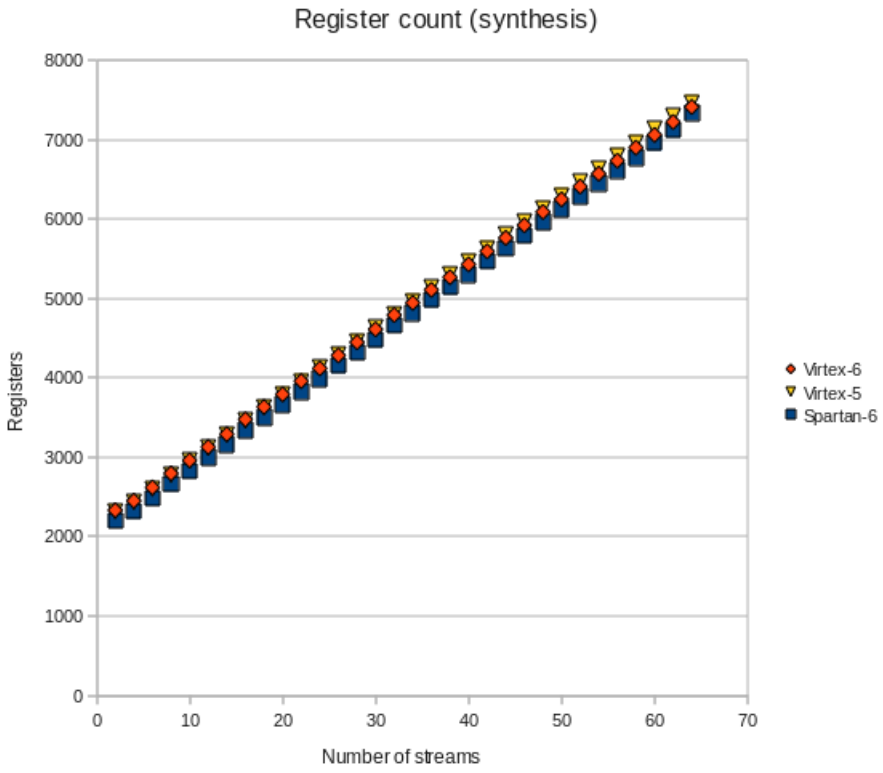
FPGA 上实际消耗的 slices 数量取决于 logic 的元素是如何打包到其中的。在 Spartan-6 或 Virtex-6 系列上，每个 slice 最多可以包含 8 个 LUTs 和 8 个 registers。因此，一个非常乐观的方法是假设 registers 包装完美，因此每个 stream 仅在消耗的资源中增加 110/8 = 14 slices。另一方面，以一半的效率打包是无需付出很大努力即可实现的。因此，stream 在 slices 中的预期成本可以估计在 14-28 slices 的范围内。

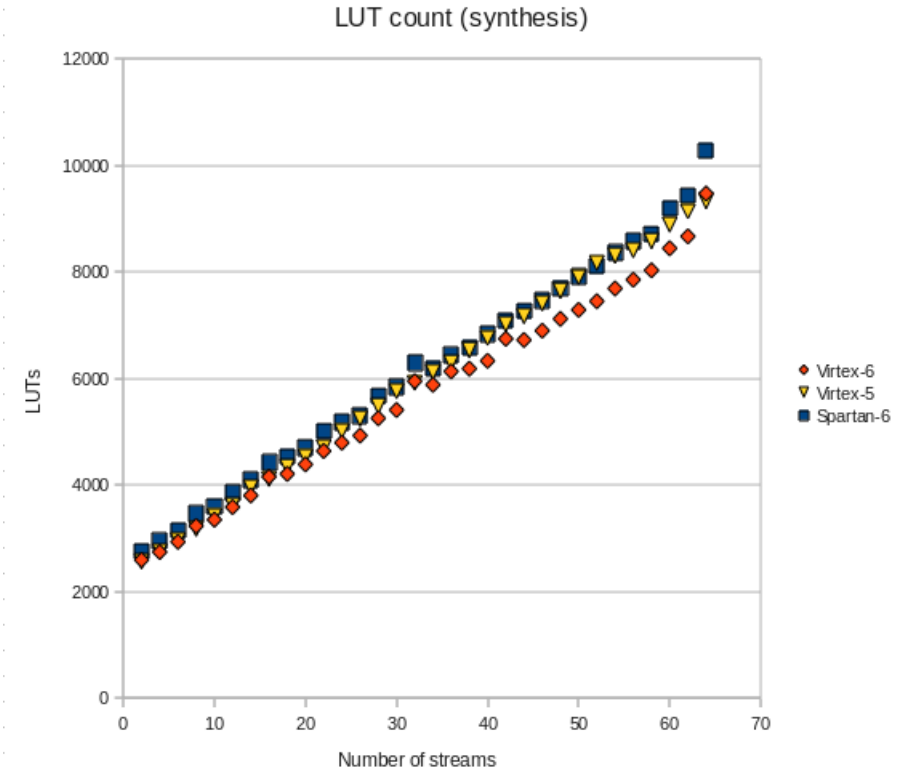
需要注意的是，当 FPGA 还没有装满时，执行 implementation 的工具不会费心将 logic 高效地打包到 slices 中，因此 slices 的数量增加可能会更加陡峭。在这种情况下，这些工具会浪费资源，因为它们太多了。

基准测试选择的设置是 upstreams 的 50% 和 downstreams 的相同设置。现实生活中的 IP cores 通常强调其中一个方向，但下面的结果可以让您了解预期的结果。

图表如下。坡度可能看起来很陡，但请注意 streams 的数量从最小的 IP core (2 streams) 变为相当重的 (64 streams)。

最重要的是，在 IP core 中分配额外的 streams 是有意义的，即使对于最琐碎的任务也是如此，因为它们在 slices 数量中的贡献相当低。


Register count (synthesis)
Registers
Number of streams
0
1000
2000
3000
4000
5000
6000
7000
8000
0
10
20
30
40
50
60
70
Virtex-6
Virtex-5
Spartan-6


LUT count (synthesis)
12000
10000
8000
6000
4000
2000
0
LUTs
0
10
20
30
40
50
60
70
Number of streams
Virtex-6
Virtex-5
Spartan-6

4

IP cores 修订版 B、XL 和 XXL

4.1 一般的

至此，本文档与 IP cores 的基线修订版（修订版 A ）相关，该修订版自 2010 年开始提供。修订版 B 和 XL 于 2015 年推出，以适应 Xillybus 用户的数据带宽需求。这些 cores 逐渐取代修订版 A。

修订版 XXL 于 2019 年推出。

与版本 A 相比，新版本（ B、XL 和 XXL ）提供了一个超集的功能，但在使用相同属性定义时在功能上是等效的（具有一些可能的性能改进）。

最显着的区别是：

- 增加的数据带宽：对于任何 FPGA、IP cores 修订版 B、XL 和 XXL，允许的总带宽分别约为修订版 A 的两倍、四倍和八倍。

  关于如何获得 Xillybus IP cores 的带宽能力，请参考 Getting started with Xillybus on a Linux host 或 Getting started with Xillybus on a Windows host 的第 5 节。

- 除了现有的 8、16 和 32 位选项外，还允许 64、128 和 256 位的用户界面数据宽度。无论 Xillybus 的 IP core 和正在使用的 PCIe 块之间的数据路径的宽度如何，这些宽度都是允许的。

- logic design 更快（更容易达到 timing constraints ），在最慢的 timing path 上延迟大约 1 ns。

- 在大多数情况下，logic 的消耗量较低（参见 4.4 部分）。

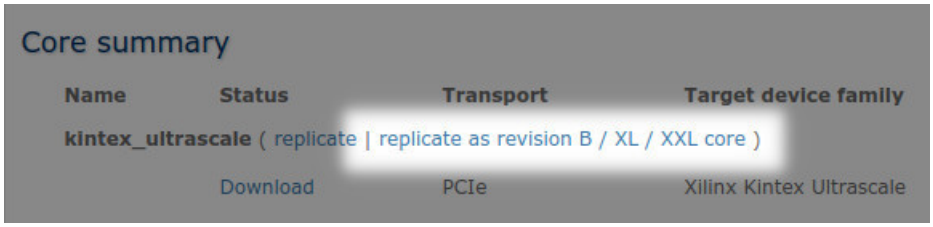
- 无论 IP core 和 application logic 之间信号的数据宽度如何，PCIe 模块的带宽都得到了有效利用。这与版本 A 的 8 位和 16 位字的 streams 的较低效率相反。

- 在 Xilinx 平台上，版本 B、XL 和 XXL 仅可用于 Vivado。

4.2 使用修订版 B/XL/XXL

通过复制 A 的 IP core 在 IP Core Factory 中创建 B/XL/XXL 的 IP core。

这是通过单击 “replicate as revision B / XL / XXL core” 来完成的，如 IP Core Factory 的截图所示：



无法降级回 A 版本。

只有请求访问这些高级 IP cores 的用户才可以升级到 B/XL/XXL。此类请求是使用网站上公布的联系信息通过普通电子邮件提出的。

获得此访问权限没有特别要求；这个请求的目的只是为了与高端用户进行更密切的接触。

B 版本的 IP cores 是 A 版本的 drop-in replacements。因此，应使用所需 FPGA 的基线 demo bundle 作为起点。由于此 demo bundle 与 A 版本的 IP core 一起提供，因此希望使用 B 版本的用户应从 IP Core Factory 进行配置和下载。

另一方面，修订版 XL 和 XXL 需要专用的 demo bundle 才能使用。这些 demo bundles 应通过电子邮件索取。

4.3 数据字宽度

虽然 A 版本的 IP cores 仅允许 8、16 和 32 位的应用程序数据宽度，但 B/ XL /XXL 版本还允许 64、128 和 256 位宽的接口。主要动机是使单个 stream 可以利用全部带宽容量。

尽管如此，也可以使用多个 streams（可能是 8、16 或 32 位宽）来划分带宽，以便总带宽利用全带宽能力（可能会降低 5-10%）。

应选择数据宽度以自然地与 application logic 配合使用。

宽字接口是允许的，无论它们是否有助于增加带宽能力。例如，在 B 版本的 IP cores 上允许 256 位字宽，即使 64 位足以利用 IP core 的带宽。这些数据宽度与 PCIe 模块

的接口信号无关。

当使用高于 32 位的字宽时，请务必注意，由于 PCIe bus 的自然数据元素是 32 位，因此 driver 中的一些保护措施可防止错误使用 streams，但当数据宽度高于32 位。例如，对字宽为 64 位的 stream 对 read() 或 write() 的任何函数调用，其长度必须是 8 的倍数。 同样，seek 操作在 64 位宽 stream 上请求的位置必须是 8 的倍数才能获得任何有意义的结果。然而，该软件只会强制它是 4 的倍数。

总之，当数据宽度在 32 位以上时，应用软件更负责执行与字宽对齐的 I/O。所有字宽的字对齐规则都相同，但与字宽为 32 位和 16 位的 streams 不同，driver 不一定会强制执行这些规则。

4.4 Logic 资源消耗

修订版 B/XL/XXL 的 IP cores 针对速度和 logic 的消耗量稍低进行了优化，但随着 streams 数量的增加，logic 的消耗量会略微增加。

为了量化 logic 资源的使用，生成了 cores for Kintex-7 和越来越多的 streams。cores经历了synthesis，logic的元素被计算在内。与 3.3 部分一样，upstreams 的基准测试是 50%，downstreams 也是如此。

以下三张图表显示了 logic 的消耗，比较了 IP cores 版本 A、B 和 XL 的相同设置。所有测试的 streams 都是 32 位宽。

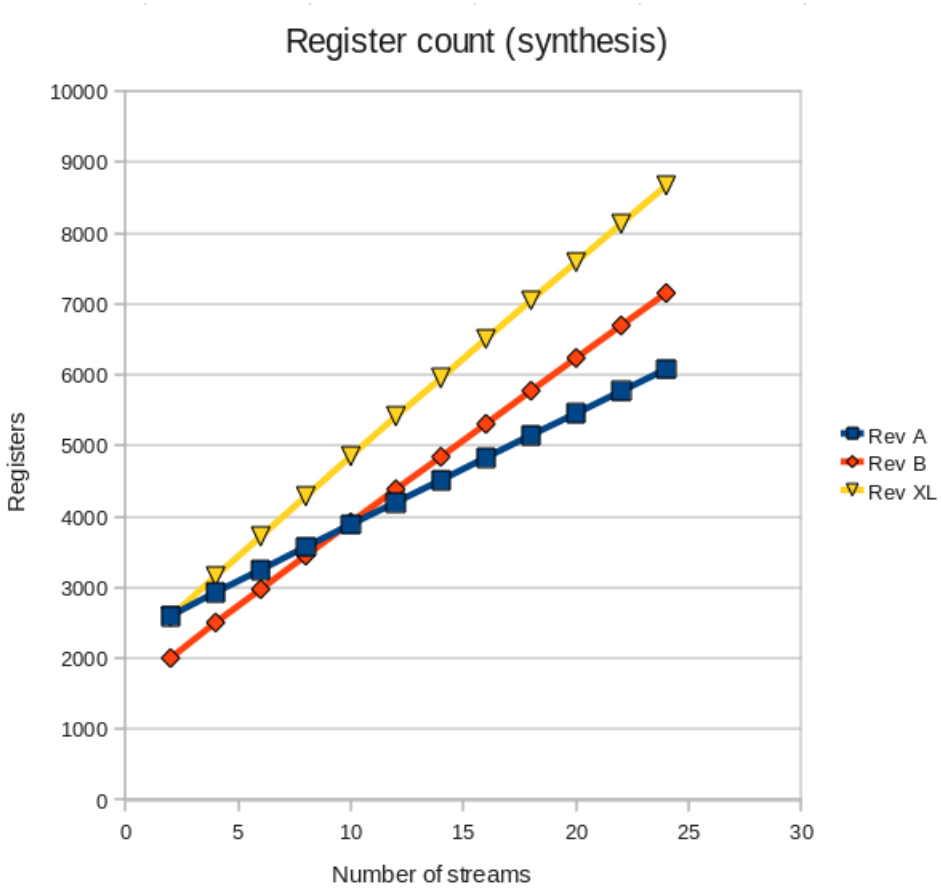
比较 registers 和 LUTs 的数量，B 版本在 streams 数量较少时优于 A 版本，但随着 streams 数量的增加而失去这一优势。

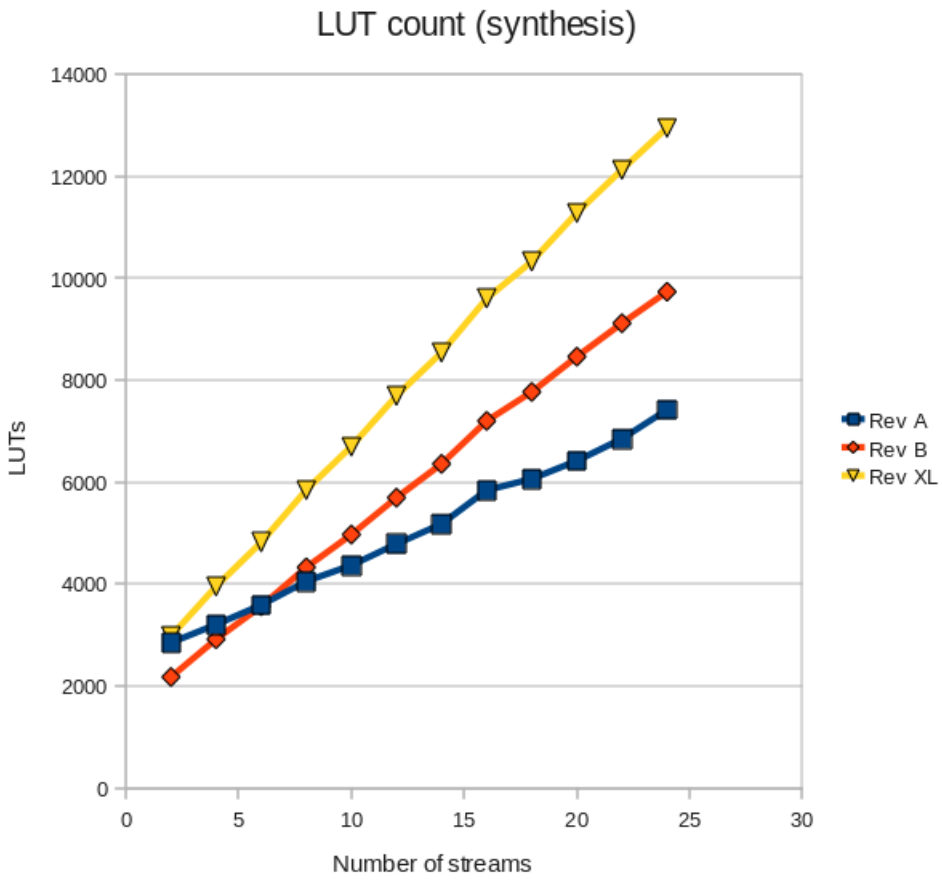
在所有情况下，修订版 XL 和 XXL 消耗的 logic 都比其他修订版多。

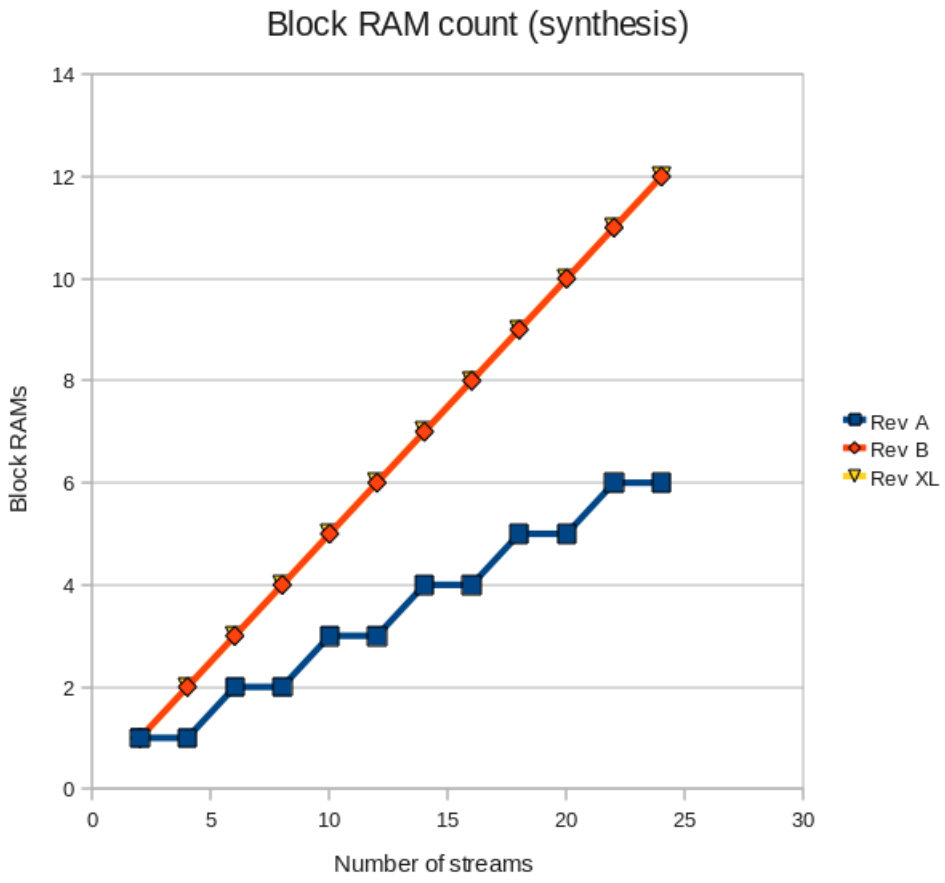
block RAMs 的图表显示，与版本 A 相比，版本 B 和 XL 消耗的 block RAMs 数量是 block RAMs 的两倍。

建议的结论是 B 版本几乎总是优于 A 版本。即使 logic 的功耗相反，B 的改进版 timing 也超过了这个差异。在大多数实际场景中都是如此，与 FPGA 的容量相比，IP core 的 logic 消耗可以忽略不计。

另一方面，修订版 XL 和 XXL 应该只用于需要其带宽容量的应用程序，因为它们消耗更多的 logic 并且相对于 timing constraints 也更加困难。


Register count (synthesis)
Registers
Number of streams
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
0
5
10
15
20
25
30
Rev A
Rev B
Rev XL


LUT count (synthesis)
14000
12000
10000
8000
6000
4000
2000
0
LUTs
0
5
10
15
20
25
30
Number of streams
Rev A
Rev B
Rev XL



4.5 将 stream 的最佳带宽从 host 调整到 FPGA

由于带有 B、XL 和 XXL 版本的 IP cores 允许更高的数据速率，因此它们对 PCIe 模块参数的正确调整越来越敏感。

demo bundles 中的 PCIe 模块已经设置为最佳性能。但是，如果 PCIe bus（它是 host 硬件的一部分）使用比正常时间更长的 latency 中继数据包，则可能需要进行轻微调整。

在 Xilinx 到 FPGAs 中，这仅适用于 B 或 XL 版本的 IP cores，与 Kintex-7 或 Virtex-7 一起使用，PCIe 块仅限于 Gen2。修订版 A 没有达到可以注意到任何改进的数据速率。

在这组有限的情况下，可能需要对 PCIe 模块的参数进行调整，以在 streams 上实现从 host 到 FPGA 的预期带宽。

这可能是必需的，因为数据在 host 到 FPGA 的方向上流动是由于 FPGA 发出的 DMA 传输请求。 host 通过发送数据来满足这些要求。请求和满足它们的数据传输 (completions) 之间的延迟取决于 host 对此类请求的响应，并且从一个 PCIe bus 到另一个不同。

为了有效利用 PCIe bus 提供的带宽，FPGA 会并行发出多个 DMA 请求，从而确保 host 始终有请求处理。但是，PCIe protocol 的流量控制对活动请求的数量有限制。由流控机制分配的有限资源称为*completion credits*，配置为PCIe endpoint。一般来说，更多的这些意味着更多的活动请求被允许，FPGA 也需要更多的资源来实现 PCIe 块。

FPGA 到 host 方向受 credits 分配的影响要小得多，因为 FPGA 将数据与 DMA 请求一起发送。因此，通过修改 credits 的设置来改善的可能性很小，因为它们几乎没有影响。

demo bundles 中的 PCIe 模块经过配置，可在普通台式计算机上实现最佳带宽利用率，processor 基于 x86 架构。尽管非常罕见，但可能需要更改配置以在 host 到 FPGA 方向上获得广告带宽。

通过增加 completion credits（header 和 data ）的数量可以获得改进。这在 Vivado 或 Quartus 中通过调用 PCIe block IP 的配置并修改其参数来完成。例如，对于在 Vivado 中配置的 Kintex-7，这是通过选择 “Core Capabilities” 选项卡并设置 “BRAM Configuration Options” 来完成的。 “Perf level” 已经设置为可能的最高值，因此没有改进的余地。然而，启用 “Buffering Optimized for Bus Mastering Application” 会增加 completion credits，但会牺牲其他类型的 credits。这可以提高 host 到 FPGA 方向的带宽性能，而不会对相反方向产生不利影响。