(机器翻译成中文)

Getting started with Xillybus on a Linux host

Xillybus Ltd.

www.xillybus.com

Version 4.0

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 This document may also be slightly outdated in relation to the original.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 **4**

2 安装 host driver **6**

2.1 安装 Xillybus 的 driver 的阶段 6
2.2 你真的需要安装任何东西吗? 6
2.2.1 一般的 6
2.2.2 预装 driver 的 Linux distributions 7
2.2.3 Linux kernels 包括 Xillybus 的 driver 7
2.3 检查先决条件 8
2.4 解压下载的文件 9
2.5 运行 kernel module 的 compilation 9
2.6 安装 kernel module 10
2.7 复制 udev rule 文件 11
2.8 加载和卸载模块 11
2.9 Linux kernel中的Xillybus drivers 12

3 “Hello, world” 测试 **14**

3.1 目标 14
3.2 准备工作 14
3.3 琐碎的 loopback 测试 15

4 host 应用示例 **17**

4.1 一般的 17
4.2 编辑和 compilation 18
4.3 运行程序 19
4.4 内存接口 20

5 高带宽性能指南 **22**

5.1 不要loopback 22
5.2 不涉及磁盘或其他存储 23
5.3 读取和写入大部分内容 24

5.4 注意CPU的消耗 . . . 24
5.5 不要让读写相互依赖 . . . 25
5.6 了解 host 和 RAM 的限制 . . . 26
5.7 DMA buffers 足够大 . . . 26
5.8 使用正确的数据字宽度 . . . 26
5.9 cache synchronization 导致速度减慢 . . . 27
5.10 参数调整 . . . 27

6 故障排除 **29**

A Linux command line 的简短生存指南 **30**
A.1 一些击键 . . . 30
A.2 获得帮助 . . . 30
A.3 显示和编辑文件 . . . 31
A.4 root 用户 . . . 32
A.5 选定的命令 . . . 33

1 介绍

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

为了简单起见，假设 host 是一台功能齐全的计算机，能够执行 compilation。 embedded platform 的过程类似，但有一些直接的差异（特别是 cross compilation 可能是必要的）。

本指南还假设基于 Xillybus 的 demo bundle 的 bitstream 已加载到 FPGA 中，并且 FPGA 已被 host 识别为 peripheral（通过相关的 PCI Express、AXI bus 或 USB 3.x）。

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
- Getting started with Xillinux for Zynq-7000
- Getting started with Xillinux for Cyclone V SoC (SoCKit)

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。不同的 driver 专用于 AXI 接口。 XillyUSB还有一个不同的driver。

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 /dev/xillybus_*something*。同样，driver for XillyUSB 创建格式为 /dev/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 Linux。

2 安装 host driver

2.1 安装 Xillybus 的 driver 的阶段

安装 Linux kernel driver 包括以下步骤:

- 检查是否需要安装。如果没有，请跳过下面的其他步骤（可能不跳过最后一步，即复制udev文件）
- 检查先决条件（即检查是否安装了 compiler 和 kernel headers）
- 将下载的包含 driver 的文件解压为 kernel module。
- 运行 kernel module 的 compilation
- 安装 kernel module
- 安装 udev 文件，以便任何用户（而不仅仅是 root ）都可以访问 Xillybus device files。

这些步骤是使用 command-line ("Terminal") 执行的。 Appendix A 中简短的 Linux 生存指南可能对那些对该界面缺乏经验的人有所帮助。

2.2 你真的需要安装任何东西吗?

2.2.1 一般的

大多数 Linux kernels 和 Linux 发行版都支持 Xillybus（适用于 PCIe 或 AXI），无需执行任何操作。这将在下面进一步解释。

也就是说，即使 Xillybus 已经受支持，有关 udev 文件安装的 2.7 部分还是值得一看的。

XillyUSB 的 driver 是版本 5.14（2021 年 8 月发布）的 Linux kernel 的一部分。

2.2.2 预装 driver 的 Linux distributions

大多数 Linux 发行版已安装 PCIe / AXI Xillybus driver (“out of the box”)。例如：

- Ubuntu 14.04 及更高版本
- 任何最近的 Fedora 发行版
- Xillinux（仅适用于 Zynq 和 Cyclone V SoC 平台）

为了快速检查 driver 是否已安装，请在 shell prompt 中键入以下内容：

```
$ modinfo xillybus_core
```

如果安装了 driver，则会打印有关它的信息。否则它会显示 “modinfo: ERROR: Module xillybus_core not found”。

同样，要检查 XillyUSB 的 driver，命令是：

```
$ modinfo xillyusb
```

XillyUSB 无需安装即可与 Ubuntu 22.04 及更高版本、Fedora 35 及更高版本以及从这两者派生的发行版一起使用。

请注意，如果 Linux 在 virtual machine 内部运行，它将不会检测到 PCIe bus 上的 Xillybus。 driver 的操作系统必须在 bare metal 上运行。 XillyUSB 可以在 virtual machine 内部工作。

2.7 部分介绍了如何永久更改 Xillybus device files 的 permissions。此更改使任何用户（而不仅仅是 root 用户）都可以访问这些文件。在台式计算机上使用 Xillybus 时，通常需要进行此修改。

2.2.3 Linux kernels 包括 Xillybus 的 driver

从版本 3.12 开始，Xillybus 的 driver（适用于 PCIe 和 AXI）包含在官方 Linux kernel 中。在3.12和3.17之间的kernels版本中，driver被列为“staging driver”，这是Linux社区完全接受新的driver之前的初步阶段。 Xillybus的driver在3.18版本中被承认为non-staging。尽管有一些与 coding style 相关的更改，但最早的 driver（kernel 的 3.12 版本）和当今可用的 driver 之间几乎没有功能差异。

加载 staging driver 时，kernel 在 system log 中发出警告。该警告表示 driver 的质量未知。对于Xillybus，可以安全地忽略此警告。

如上所述，XillyUSB 的 driver 从版本 5.14 开始包含在 Linux kernel 中。

关于属于 Linux distribution 一部分的 kernels： 即使 Xillybus 的 drivers 是 kernel 的 source code 的一部分，仅当 kernel 配置为包含这些 drivers 时，这些 drivers 才会包含在 compilation 中。 Xillybus 的 drivers 作为 kernel modules 包含在大多数主流 Linux distributions 中，但每个 distribution 对于选择包含在 kernel 中的内容都有自己的标准。因此，Xillybus 可能不包含在与 distribution 一起提供的 kernel 中。

本指南重点介绍如何通过 kernel modules 的单独 compilation 安装 drivers。这通常是最简单的方法。然而，无论如何，那些使用 kernel 的 compilation 的人可能更喜欢配置 kernel 以包含 Xillybus 的 drivers。该方法将在 2.9 节中讨论。

2.3 检查先决条件

Linux 系统可能缺少 kernel module compilation 的基本工具。了解这些工具是否存在的最简单方法是尝试运行它们。例如，在 command prompt 上键入 “make coffee”。这是正确的回应：

```
$ make coffee

make: *** No rule to make target ‘coffee’.  Stop.
```

尽管这是一个错误，但我们可以看到 “make” 实用程序存在。但如果 GNU make 丢失并且需要安装，输出将如下所示：

```
$ make coffee
bash: make: command not found
```

C compiler 也是需要的。输入 “gcc” 以检查是否安装了 compiler：

```
$ gcc
gcc: no input files
```

此响应表明 “gcc” 已安装。再次出现错误消息，但 “command not found” 没有。

除了这两个工具之外，还需要安装 kernel headers。这个检查起来有点困难。了解这些文件是否丢失的常见方法是当 kernel compilation 出现故障并显示 header file 丢失的错误时。

kernel module compilation 是一项常见任务，因此互联网上有大量针对每个 Linux distribution 的信息，介绍如何为 compilation 准备系统。

在 Fedora、RHEL、CentOS 和 Red Hat 的其他衍生产品上，此类命令可能会让计算机做好准备：

```
# yum install gcc make kernel-devel-$(uname -r)
```

对于 Ubuntu 和其他基于 Debian 的发行版：

```
# apt install gcc make linux-headers-$(uname -r)
```

重要的:

这些安装命令必须作为 *root* 发出。那些不熟悉 *root* 用户概念的人建议先了解一下。请参阅附录 A.4 部分。

2.4 解压下载的文件

从 Xillybus 的站点下载 driver 后，将目录更改为下载文件所在的位置。在 command prompt 上，键入（不包括 $ 符号）：

```
$ tar -xzf xillybus.tar.gz
```

对于 XillyUSB driver：

```
$ tar -xzf xillyusb.tar.gz
```

应该没有反应，只是一个新的command prompt。

2.5 运行 kernel module 的 compilation

将目录更改为 kernel module 的 source code 所在的位置。对于 Xillybus driver：

```
$ cd xillybus/module
```

对于 XillyUSB driver：

```
$ cd xillyusb/driver
```

键入 “make” 以执行模块的 compilation。文字记录应该是这样的：

```
$ make
make -C /lib/modules/3.10.0/build SUBDIRS=/home/myself/xillybus/module modules
make[1]: Entering directory '/usr/src/kernels/3.10.0'
  CC [M]  /home/myself/xillybus/module/xillybus_core.o
  CC [M]  /home/myself/xillybus/module/xillybus_pcie.o
  Building modules, stage 2.
  MODPOST 2 modules
  CC      /home/myself/xillybus/module/xillybus_core.mod.o
  LD [M]  /home/myself/xillybus/module/xillybus_core.ko
  CC      /home/myself/xillybus/module/xillybus_pcie.mod.o
  LD [M]  /home/myself/xillybus/module/xillybus_pcie.ko
make[1]: Leaving directory '/usr/src/kernels/3.10.0'
```

细节可能略有不同，但不会出现错误或 warnings。对于 XillyUSB，仅生成单个模块 xillyusb.ko。

请注意，kernel modules 的 compilation 特定于在 compilation 期间运行的 kernel。

如果打算使用另一个 kernel，请键入 “make TARGET=kernel-version”，其中 “kernel-version” 是所需 kernel 版本的名称。这是/lib/modules/中出现的名称。或者，在名为 “Makefile” 的文件中编辑以下行：

```
KDIR := /lib/modules/$(TARGET)/build
```

将 KDIR 的值更改为所需 kernel headers 的 path。

2.6 安装 kernel module

在不更改目录的情况下，将用户更改为 root （例如使用 “sudo su” ）。然后键入以下命令：

```
# make install
```

此命令可能需要几秒钟才能完成，但不应生成任何错误。

如果此操作失败，请将 compilation 生成的 *.ko 文件复制到 kernel modules 的现有子目录中。然后运行depmod。以下示例显示了如何对 PCIe driver 执行此操作（如果 kernel 的相关版本是 3.10.0）：

```
# cp xillybus_core.ko /lib/modules/3.10.0/kernel/drivers/char/
```

```
# cp xillybus_pcie.ko /lib/modules/3.10.0/kernel/drivers/char/

# depmod -a
```

安装不会立即将模块加载到 kernel 中。如果发现 Xillybus 外设，则会在系统的下一个 boot 上完成此操作。如何手动加载模块如 2.8 节所示。

对于 XillyUSB，不需要 reboot： 下次 USB device 连接到计算机时会自动加载该模块。

2.7 复制 udev rule 文件

默认情况下，Xillybus device files 只能由其所有者（即 root）访问。让任何用户都可以访问这些文件非常有意义，这样就可以避免以 root 的方式工作。当 device files 生成时，udev 机制通过遵守特定规则来更改 file permissions。

如何启用此功能： 保留在同一目录中，并保留 root 用户身份。将 udev rule 文件复制到系统中存储此类文件的位置（最有可能是 /etc/udev/rules.d/ ）。

例如：

```
# cp 10-xillybus.rules /etc/udev/rules.d/
```

该文件的内容很简单：

```
SUBSYSTEM=="xillybus", MODE="666", OPTIONS="last_rule"
```

这意味着 Xillybus device driver 生成的所有文件都应指定为 permission mode 0666。换句话说，每个人都可以阅读和写作。

对于 XillyUSB，文件为 10-xillyusb.rules，包含

```
SUBSYSTEM=="xilly*", KERNEL=="xillyusb_*", MODE="0666"
```

请注意，可以更改 udev 文件以获得不同的结果。例如，可以更改 device files 的所有者，这样只有特定用户才能访问这些文件。

2.8 加载和卸载模块

为了加载模块（并开始使用 Xillybus ），输入 root：

```
# modprobe xillybus_pcie
```

或者，对于 XillyUSB：

```
# modprobe xillyusb
```

这将使 Xillybus device files 出现（假设在 bus 上检测到 Xillybus device ）。

请注意，如果系统执行其 boot 进程时检测到 Xillybus PCIe / AXI 外围设备并且 driver 已按上述方式安装，则无需执行此操作。如果在已安装 driver 的情况下将 XillyUSB device 连接到计算机，则无需执行此操作。

要查看 kernel 中的模块列表，请键入 “lsmod”。要从 kernel 中移除模块，请键入（对于 PCIe driver ）

```
# rmmod xillybus_pcie xillybus_core
```

这将使 device files 消失。

如果出现问题，请检查 /var/log/syslog 文件中是否有包含单词 “xillybus” 或 “xillyusb”（如果适用）的消息。在此日志文件中经常可以找到有价值的线索。还可以使用 “dmesg” 命令访问相同的日志信息。

如果不存在 /var/log/syslog 日志文件，则可能是 /var/log/messages。可以尝试命令 “journalctl -k”。

2.9 Linux kernel中的Xillybus drivers

如前所述，从 v3.12.0 版本开始，Xillybus 的 driver 包含在 Linux kernel 中。因此可以把整个kernel进行一个compilation，这样这个kernel就支持Xillybus。这是单独安装 kernel modules 的替代方法，如上所示。

从功能角度来看，涉及 kernel compilation 的方法产生与 2.3 至 2.6 节中描述的步骤相同的结果。

为了将 Xillybus 的 driver 包含在面向 compilation 的 kernel 中，需要启用一些 kernel configuration options。有两种方法可以包含 driver： 作为 kernel modules 或作为 kernel image 的一部分。

例如，这是 kernel 的配置文件（ .config ）中为 PCIe 接口启用 Xillybus 的 driver 的部分：

```
CONFIG_XILLYBUS=m
CONFIG_XILLYBUS_PCIE=m
```

“m” 表示 driver 作为 kernel module 包含在内。 “y”意味着将driver包含在kernel image中。

同样，对于 XillyUSB（kernel v5.14 及更高版本）：

```
CONFIG_XILLYUSB=m
```

对.config进行更改的常见方法是使用kernel的配置工具： “make config”、“make xconfig” 或 “make gconfig”。

xconfig 和 gconfig 是更易于使用的 GUI 工具，因为它们允许搜索字符串 “xillybus” 以便找到 Xillybus 的 drivers。 driver 通过单击复选框来启用。 .config 文件的文本表示有助于验证是否已设置正确的选项。

在版本低于 3.18 的 kernels 上，可能需要先启用 staging drivers，然后再尝试启用 Xillybus。这会在 .config 文件中生成以下行。

```
CONFIG_STAGING=y
```

在.config文件中启用Xillybus的driver后，照常运行kernel compilation。

从 kernel 5.14 开始，当启用 Xillybus 或 XillyUSB 的 driver 时，会自动启用名称为 CONFIG_XILLYBUS_CLASS 的选项。这是配置系统的依赖性规则的结果。因此，手动更改此选项是不必要的（而且通常是不可能的）。

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 4.4 部分。

3.2 准备工作

执行 "Hello world" 测试需要一些准备工作：

- Xillybus 的 driver 安装在计算机上，如 2 部分所述。

- FPGA 必须装载从 demo bundle 创建的 bitstream（未经修改）。Getting started with the FPGA demo bundle for Xilinx 或 Getting started with the FPGA demo bundle for Intel FPGA中解释了如何实现这一点。

  使用 Xillinux（与 Zynq 或 Cyclone V SoC 一起使用）的用户，请参阅 Getting started with Xillinux for Zynq-7000 或 Getting started with Xillinux for Cyclone V SoC (SoCKit)：默认情况下，demo bundle 已包含在该系统中。

- 仅与 PCIe 相关：当计算机执行boot时，在PCIe bus上检测到FPGA。这可以使用 “lspci” 命令进行验证。

- 仅与 USB 相关：FPGA通过USB port连接到计算机，计算机已将FPGA检测为USB device。这可以使用 “lsusb” 命令进行验证。

- 您应该能够轻松使用 Linux command-line。附录 A 可能对此有所帮助。

如果这些准备工作都做得正确的话，Xillybus的device files应该是可用的。例如，名为 /dev/xillybus_read_8 的文件应该存在。

3.3 琐碎的 loopback 测试

执行此测试的最简单方法是使用名为 “cat” 的 Linux command-line utility：

打开两个terminal windows。在某些计算机上，可以通过双击名为 “Terminal” 的图标来完成此操作。如果没有这样的图标，请在桌面的菜单中搜索它。

在第一个 terminal window 上，在 command prompt 上键入以下命令（不要键入美元符号，它是 prompt）：

```
$ cat /dev/xillybus_read_8
```

这使得 “cat” 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。

使用 *XillyUSB* 的用户会发现 *device file* 是 *xillyusb_00_read_8*。*“xillyusb”*前缀很明显，*“00”*索引的目的是允许多个*USB devices*连接到同一个*host*。本指南中使用 *PCIe* 和 *AXI* 的命名约定。

在第二个 terminal 窗口中，键入

```
$ cat > /dev/xillybus_write_8
```

请注意 > 字符。它告诉 “cat” 将 console 上键入的所有内容发送到 xillybus_write_8 (redirection)。

现在在第二个 terminal 上键入一些文本，然后按 ENTER。相同的文本将出现在第一台 terminal 上。在按下 ENTER 之前，不会向 xillybus_write_8 发送任何内容。这是 Linux 计算机上的常见约定。

这两个 “cat” 命令均由 CTRL-C 停止。

如果在尝试这两个 “cat” 命令时遇到错误消息，请首先验证 device files 是否已创建（即 /dev/xillybus_read_8 和 /dev/xillybus_write_8 存在）。还要检查是否有拼写错误。

如果错误是 “permission denied”，则可以按照 2.7 部分所示进行修复。但请注意，只有当 kernel modules 加载到 kernel 中时，udev 文件才生效。有关如何重新加载 kernel modules 的信息，请参阅 2.8 部分。或者，在计算机上执行 reboot。

克服 “permission denied” 错误的另一种可能性是作为 root 用户使用 Xillybus device files。在台式计算机上不太推荐这样做（但通常在 embedded platforms 上这样做）。有关此内容的更多信息，请参见附录 A.4 部分。

对于其他错误，请遵循 2.8 部分中有关在 /var/log/syslog 中查找更多信息的指南或使用 “dmesg” 命令（或获取 kernel log 的类似方法）。

还可以执行简单的文件操作。例如，在不停止第一个 terminal 中的 “cat” 命令的情况下，在第二个 terminal 中键入以下命令：

```
$ date > /dev/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。

还有另一对 device files，它们之间有一个 loopback：/dev/xillybus_read_32 和 /dev/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。

4 host 应用示例

4.1 一般的

有四五个简单的C程序演示了如何访问Xillybus的device files。这些程序可以在包含 host driver for Xillybus / XillyUSB 的压缩文件中找到（可在网站上下载）。参考“demoapps”目录，该目录包含以下文件：

- Makefile——该文件包含 “make” 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。

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，仅演示在 Linux 计算机上访问文件的标准方法。这些方法在 Xillybus host application programming guide for Linux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。

那些为PCIe下载driver的人会在“demoapps”目录中找到第五个程序：fifo.c。该程序演示了 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 仅适用于非常高的数据速率，并且当 RAM buffer 需要非常大（即多个 gigabytes）时。

该程序不与 XillyUSB 的 driver 一起包含，因为 XillyUSB 无法实现 fifo.c 可能需要的数据速率。

4.2 编辑和 compilation

如果您对Linux中的compilation程序有经验，您可以跳到下一节： Xillybus的示例程序中的compilation是用“make”按照通常的方式完成的。

首先，将目录更改为 C 文件所在的位置：

```
$ cd demoapps
```

要运行 compilation 的所有五个程序，只需在 shell prompt 处键入 “make”。预计会有以下文字记录：

```
$ make
gcc  -g -Wall -O3 memwrite.c -o memwrite
gcc  -g -Wall -O3 memread.c -o memread
gcc  -g -Wall -O3 streamread.c -o streamread
gcc  -g -Wall -O3 streamwrite.c -o streamwrite
gcc  -g -Wall -O3 -pthread fifo.c -o fifo
```

以 “gcc” 开头的五行是 “make” 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。使用“make”就可以了。

在某些系统上，如果未安装 POSIX threads library（例如，在 Cygwin 的某些安装中），第五个 compilation（fifo.c ）可能会出现故障。如果您无意使用fifo.c，则可以忽略此错误。

“make” 实用程序仅在必要时运行 compilation。如果仅更改一个文件，“make” 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“make”换recompilation。不会发生不必要的compilation。

使用 “make 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，以便“make”也运行这个新文件的compilation。例如，假设 memwrite.c 被复制到一个名为 mygames.c 的新文件。这可以通过 GUI 接口或 command line 来完成：

```
$ cp memwrite.c mygames.c
```

下一步是编辑 Makefile。有许多文本编辑器和多种运行它们的方法。在大多数系统上，可以通过键入 “gedit” 或 “xed” 从 shell prompt 启动 GUI editor。不过，在计算机桌面的菜单中找到 GUI text editor 更容易。 terminal window 内部还支持许多文本编辑器，例如 vim、emacs、nano 和 pico。

使用哪款 editor 取决于品味和个人经验。例如，可以使用此命令开始编辑 Makefile：

```
$ xed Makefile &
```

命令末尾的“&”告诉 shell 不要等待程序完成： 下一个shell prompt立即出现。这适用于启动 GUI 应用程序等。

Makefile 中应更改的行是：

```
APPLICATIONS=memwrite memread streamread streamwrite
```

该行更改为：

```
APPLICATIONS=memwrite memread streamread streamwrite mygames
```

mygames.c 的 compilation 将在您下次键入 “make” 时发生。

4.3 运行程序

3.3 部分中显示的简单 loopback 示例可以使用两个示例程序来完成。

我们假设 “demoapps” 已经是 current directory，并且 compilation 已经通过“make”完成。

在第一个 terminal 中输入：

```
$ ./streamread /dev/xillybus_read_8
```

这是从 device file 读取的程序。

请注意，该命令以 “./” 开头： 有必要明确指向executable的目录。在此示例中，表达式 “./” 用于请求 current directory。

然后，在第二个 terminal window 中：

```
$ ./streamwrite /dev/xillybus_write_8
```

这或多或少类似于 “cat” 的示例。不同的是，“streamwrite” 不会等待 ENTER 才将数据发送到 device file。相反，该程序尝试在每个 character 上单独运行。为了实现这一点，程序使用了一个名为 config_console() 的函数。此功能仅用于立即响应键盘输入的目的。这与Xillybus无关。

上面的示例涉及 PCIe / AXI 的 Xillybus。对于 XillyUSB，device files 的名称前缀略有不同。例如，xillyusb_00_read_8 而不是 xillybus_read_8。

重要的:

由 *streamread* 和 *streamwrite* 执行的 *I/O* 操作效率较低： 为了使这些程序更加简单，*I/O buffer*的大小只有*128*字节。当需要高数据速率时，应使用更大的*buffers*。请参阅 *5.3* 部分。

4.4 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 Linux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。

在写入内存之前，可以使用hexdump实用程序观察现有情况：

```
$ hexdump -C -v -n 32 /dev/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/dev/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 /dev/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 /dev/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。

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，如下节5.3所示。

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 /dev/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 /dev/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，4.3 部分（streamread 和 streamwrite）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。

以下 shell 命令可用于速度检查（根据需要更换 /dev/xillybus_* names）：

```
dd if=/dev/zero of=/dev/xillybus_sink bs=128k
dd if=/dev/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。

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 的消耗非常重要。“top” 等实用程序可用于此目的。然而，该程序的输出（以及类似的替代方案）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CPU消耗低，还是特定thread上的100%？如果使用“top”，则取决于程序的版本。

另外需要注意的是，system calls 的处理时间是如何测量和显示的： 如果操作系统的 overhead 减慢了数据流速度，如何测量？

检查这一点的一个简单方法是使用 “time” 实用程序。例如，

```
$ time dd if=/dev/zero of=/dev/null bs=128k count=100k
102400+0 records in
102400+0 records out
13421772800 bytes (13 GB) copied, 1.07802 s, 12.5 GB/s

real 0m1.080s
user 0m0.005s
sys 0m1.074s
```

底部 “time” 的输出表明 “dd” 完成所需的时间为 1.080 秒。其中，processor 在 5 个 ms 期间执行了 user space program，并在 1.074 秒内与 system calls 忙碌。因此，在这个具体示例中，很明显 processor 几乎一直忙于执行 system calls。这并不奇怪，因为 “dd” 在这里没有做任何事情。

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 Linux 的 6.6 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

这主要与 embedded systems 和/或使用 revision XL / XXL IP core 时相关： 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 cache synchronization 导致速度减慢

此问题不适用于基于属于 x86 系列（32 和 64 位）的 CPUs 的计算机。那些将Xillybus与Zynq processor的AXI bus（例如与Xillinux）一起使用的人也可以忽略此主题。

然而，当使用 DMA buffers 时，一些 embedded processors 需要 cache 的显式同步。这会大大减慢 CPU 外设的数据传输速度。

此问题并非 Xillybus 特有： 所有基于 DMA 的 I/O（例如 Ethernet、USB 和其他外设）都观察到类似的行为。

通过查看 CPU 的消耗可以揭示 cache 造成的速度减慢。如果 CPU 在 system call 状态下花费不合理的时间（“time” 实用程序的 “sys” 行输出），则这可能表明 cache 存在问题。发生这种情况是因为 CPU 花费了大量时间来执行 cache synchronization。

然而，重要的是首先排除小型 buffers 的可能性（如上面 5.3 和 5.7 部分所述）。

x86 系列永远不会出现此问题，因为这些 CPUs 具有 coherent cache。因此不需要 cache synchronization。这同样适用于 Xillinux，因为 IP core 通过 ACP port 连接到 CPU。

但是当Zynq processor将Xillybus与PCIe bus一起使用时，就会出现此问题。其他几个 embedded processors 也受到影响，特别是 ARM processors。

5.10 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

Xillybus / XillyUSB 的 drivers 旨在产生有意义的 log messages。以下是获取它们的几种选择：

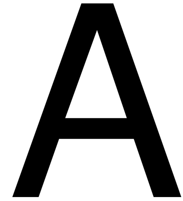
- “dmesg” 命令的输出。
- “journalctl -k” 命令的输出。
- 在log file中。这取决于操作系统，例如 /var/log/syslog 或 /var/log/messages。

当出现问题时，建议搜索包含单词 “xillybus” 或 “xillyusb” 的邮件。即使一切看起来都工作正常，也建议偶尔检查一下 system log。

来自 PCIe / AXI 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。



Linux command line 的简短生存指南

不习惯 command line 接口的人可能会发现很难在 Linux 计算机上完成工作。基本的 command-line 接口 30 多年来一直保持不变。因此，有大量关于如何使用每个命令的在线教程。这个简短的指南只是一个入门。

A.1 一些击键

这是最常用的击键的摘要。

- CTRL-C： 停止当前正在运行的程序
- CTRL-D： 完成此会话（关闭 terminal 窗口）
- CTRL-L： 清晰的屏幕
- TAB： 在 command prompt 上，尝试对已写入的内容进行 autocomplete。这对于例如长文件名很有用： 键入名称的开头，然后键入 [TAB]。
- 向上和向下箭头： 在 command prompt 上，建议历史记录中的先前命令。这对于重复刚刚完成的事情很有用。也可以编辑以前的命令，因此它也适合执行与以前的命令几乎相同的操作。
- space： 当计算机显示 terminal pager 时，[space] 表示 “page down”。
- q： “Quit”。在逐页显示时，“q” 用于退出该模式。

A.2 获得帮助

没有人真正记得所有 flags 和选项。有两种常见方法可以获得更多帮助。一种方式是 “man” 命令，第二种方式是 help flag。

例如，为了了解有关 “ls” 命令的更多信息（列出当前目录中的文件）：

```
$ man ls
```

请注意，“$”符号是 command prompt，计算机打印出来表示它已准备好执行命令。通常prompt较长，它包含一些有关用户和当前目录的信息。

manual page 与 terminal pager 一起显示。使用 [space]、箭头键、Page Up 和 Page Down 进行导航，使用 'q' 退出。

有关如何运行该命令的简短摘要，请使用 `--help` 标志。某些命令响应 `-h` 或 `-help`（带有一个破折号）。其他命令在语法不正确时打印帮助信息。这是一个反复试验的问题。对于 ls 命令：

```
$ ls --help
Usage: ls [OPTION]... [FILE]...
List information about the FILEs (the current directory by default).
Sort entries alphabetically if none of -cftuvSUX nor --sort.

Mandatory arguments to long options are mandatory for short options too.
  -a, --all                  do not ignore entries starting with .
  -A, --almost-all           do not list implied . and ..
      --author               with -l, print the author of each file
```

...

（它继续）

A.3 显示和编辑文件

如果文件预计很短（或者可以使用 terminal window 的滚动条），则可以通过以下方式在 console 上显示其内容：

```
$ cat filename
```

较长的文件需要 terminal pager：

```
$ less filename
```

至于编辑文本文件，有很多编辑器可供选择。最流行的（但不一定是最容易上手的）是 emacs （和 XEmacs ）以及 vi。 vi 编辑器很难学习，但它始终可用且始终有效。

推荐的简单 GUI 编辑器是 gedit 或 xed，以可用者为准。它可以通过桌面菜单或从 command line 启动：

```
$ gedit filename &
```

末尾的'&'表示该命令应该执行“in the background”。或者简单地说，下一个 command prompt 在命令完成之前出现。 GUI 应用程序最好这样开始。

当然，这些示例中的 'filename' 也可以是 path。例如查看系统主log file：

```
# less /var/log/syslog
```

要跳到文件末尾，请在 “less” 运行时按 shift-G。

请注意，日志文件只能由某些计算机上的 root 用户访问。

A.4 root 用户

所有 Linux 计算机都有一个名为 “root” 的用户，该用户拥有 ID 0。该用户也称为 superuser。该用户的特殊之处在于它可以执行任何操作。其他每个用户在访问文件和资源方面都有限制。并非所有操作都允许每个用户。这些限制不会强加给 root 用户。

这不仅仅是多用户计算机上的隐私问题。允许执行任何操作，包括在 shell prompt 上使用简单命令删除硬盘上的所有数据。它包括其他几种错误删除数据、使计算机无法使用或使系统容易受到攻击的方法。 Linux 系统的基本假设是，无论 root 用户是谁，都知道他或她在做什么。计算机不会询问 root 是否确定的问题。

作为 root 工作对于系统维护（包括软件安装）是必要的。避免搞砸的技巧是在按下 ENTER 之前进行思考，并确保命令完全按照要求输入。严格遵循安装说明通常是安全的。在不了解他们到底在做什么的情况下，不要进行任何修改。如果可以以非 root 的用户身份重复相同的命令（可能涉及其他文件），请尝试一下，看看以该用户身份会发生什么。

由于作为 root 的危险，作为 root 运行命令的常用方法是使用 sudo 命令。例如查看主日志文件：

```
$ sudo less /var/log/syslog
```

这并不总是有效，因为系统需要配置为允许用户使用 “sudo”。系统需要用户密码（不是root密码）。

第二种方法是键入 “su” 并启动一个会话，其中每个命令都以 root 给出。这在root需要完成几个任务时很方便，但也意味着有更大的机会忘记自己是root，想都不想就写错了。保持 root 会话简短。

```
$ su
```

```
Password:
# less /var/log/syslog
```

这次需要 root 密码。

shell prompt 的变化表明身份从普通用户更改为 root。如有疑问，请键入 “whoami” 以获取您当前的 user name。

在某些系统上，sudo 适用于相关用户，但可能仍然需要以 root 身份调用会话。如果 “su” 无法使用（主要是因为 root 密码未知），简单的替代方法是：

```
$ sudo su
#
```

A.5 选定的命令

最后，这里有几个常用的 Linux 命令。

一些文件操作命令（最好使用 GUI 工具）：

- cp——复制文件或文件。
- rm–删除一个或多个文件。
- mv——移动文件或文件。
- rmdir——删除目录。

以及一些建议一般了解的内容：

- ls——列出当前目录（或其他目录，如果指定）中的所有文件。 “ls -l” 列出文件及其属性。
- lspci — 列出 bus 上的所有 PCI（和 PCIe）设备。用于检查 Xillybus 是否已被检测为 PCIe 外设。还可以尝试 lspci -v、lspci -vv 和 lspci -n。
- lsusb — 列出 bus 上的所有 USB 设备。用于检查 XillyUSB 是否已被检测为外设。还可以尝试 lsusb -v 和 lsusb -vv。
- cd–更改目录
- pwd–显示当前目录

- cat–将文件（或多个文件）发送到 standard output。如果没有给出 argument，则使用 standard input。该命令的最初目的是连接文件，但它最终成为用于文件简单输入和输出的瑞士刀。
- man——显示命令的manual page。也可以尝试 “man -a”（有时一个命令有多个 manual page）。
- less——terminal pager。逐页显示 standard input 中的文件或数据。往上看。也用于显示命令的长输出。例如：

```
$ ls -l | less
```

- head–显示文件的开头
- tail——显示文件结尾。或者更好的是，带有 -f 标志：显示结尾+新行到达时。适合日志文件，例如（如 root）：

```
# tail -f /var/log/syslog
```

- diff——比较两个文本文件。如果它什么也没说，文件是相同的。
- cmp——比较两个二进制文件。如果它什么也没说，文件是相同的。
- hexdump——以简洁的格式显示文件的内容。首选标志 -v 和 -C。
- df——显示已安装的磁盘以及每个磁盘中剩余的空间。更好的是，“df -h”
- make– 尝试按照 Makefile 中的规则构建（运行 compilation）项目
- gcc——GNU C compiler。
- ps——获取正在运行的进程列表。 “ps a”、“ps au” 和 “ps aux” 将提供不同数量的信息。

还有一些高级命令：

- grep– 在文件中搜索 textual pattern 或 standard input。该模式是 regular expression，但如果它只是文本，那么它会搜索字符串。例如，在主日志文件中搜索 “xillybus” 作为 case insensitive 字符串，并逐页显示输出：

```
# grep -i xillybus /var/log/syslog | less
```

- find——查找文件。它具有复杂的参数语法，但它可以根据文件的名称、年龄、类型或您能想到的任何内容来查找文件。请参阅 man page。