(机器翻译成中文)

Getting started with Xillinux for Zynq-7000 v2.0

Xillybus Ltd.

www.xillybus.com

Version 4.2

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 This document may also be slightly outdated in relation to the original.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

1.1 Xillinux 分布

Xillinux 是一个完整的、图形化的、基于 Lubuntu 16.04 的 Linux 发行版，用于 Zynq-7000 设备，旨在作为快速开发混合软件/logic 项目的平台。目前支持的板卡有 Z-Turn Lite、Zedboard、MicroZed 和 Zybo。

与任何 Linux 发行版一样，Xillinux 是一个软件集合，支持与运行 Linux 的个人台式计算机大致相同的功能。与常见的 Linux 发行版不同，Xillinux 还包括一些硬件 logic，尤其是 VGA 适配器。

使用 Z-Turn Lite、Zedboard 和 Zybo，分布组织为经典的键盘、鼠标和显示器设置。它还允许从 USB UART 端口进行命令行控制，但此功能主要用于解决问题。

当与缺少 VGA/DVI 输出的 MicroZed 一起使用时，只有 USB UART 用作 console。

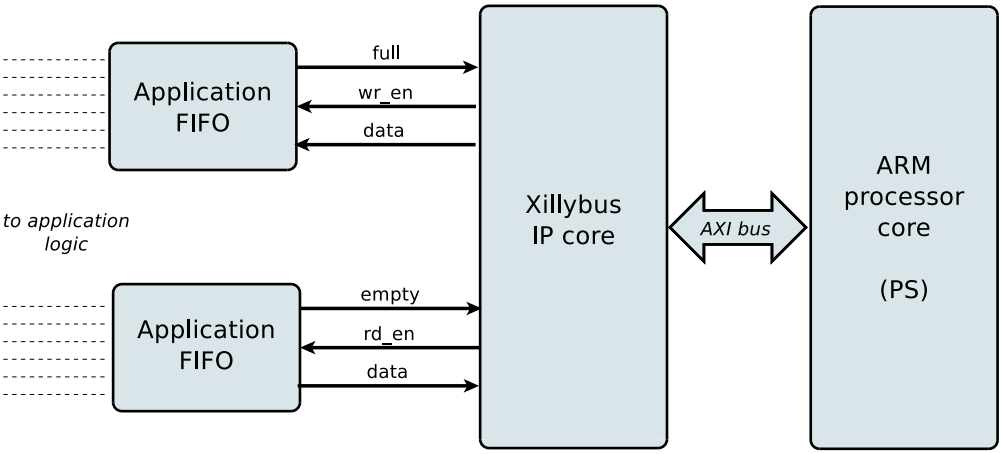
Xillinux 还是一个 kickstart 开发平台，用于集成设备的 FPGA logic fabric 和在 ARM processor 上运行的普通 user space applications。凭借其随附的 Xillybus IP core 和 driver，只需基本的编程和 logic design 技能即可完成 design 应用程序，其中 FPGA logic 和基于 Linux 的软件协同工作。

捆绑的 Xillybus IP cores 通过向应用程序设计人员提供简单而高效的工作环境，消除了处理 kernel programming 的低级内部以及 application logic 和 processor 之间的接口的需要。

1.2 Xillybus IP core

Xillybus 是基于 DMA 的端到端解决方案，用于在 FPGA 和运行 Linux 或 Microsoft Windows 的 host 之间传输数据。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。它适用于使用 PCI Express bus 作为底层传输的个人计算机和

embedded 系统，以及与 AMBA bus (AXI3/ AXI4) 接口的基于 ARM 的 processors。



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 AXI bus 实现数据流，在 processor core 的 bus 上生成 DMA 请求。

host 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。

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。

完成本指南中所述的准备工作后，建议在 http://xillybus.com/custom-ip-factory 上构建和下载您的自定义 IP core。

本指南说明如何快速设置 Xillinux 发行版，包括 Xillybus IP core。该IP core可以附加到用户提供的数据源和数据消费者，进行真实的应用场景测试。它不是演示套件，而是功能齐全的 starter design，可以按原样执行有用的任务。

用专为特殊应用定制的 IP core 替换现有 IP core 是一个快速过程，需要替换一个二进制文件并实例化一个模块。

有关使用 Xillybus IP core 的更多信息，请参阅以下文档：

- Getting started with Xillybus on a Linux host

- Xillybus host application programming guide for Linux
- The guide to defining a custom Xillybus IP core

好奇的朋友可以在 Xillybus host application programming guide for Linux的附录 A 中找到关于 Xillybus IP core 是如何实现的简要说明。

2

先决条件

2.1 硬件

Xillinux 是 Xillybus 为 Zynq 发行的 Linux，目前支持以下板卡：

- Z-Turn Lite（连同任何可用的 Zynq 设备）
- Zedboard
- 7010 MicroZed。 7020 MicroZed 也通过小修复得到支持：参见 5.5 部分。
- Zybo

如果您要购买 Z-Turn Lite 板，建议您访问此板上的 web page 以获得选择项目的帮助。

上面未列出的板的所有者可以在他们自己的硬件上运行分发，但可能需要进行某些更改，其中一些可能是不平凡的。在 5.2 部分中了解更多信息。

要将板子（不包括 MicroZed）用作带显示器、键盘和鼠标的台式计算机，需要以下项目：

- 一种显示器，能够显示符合 VESA 标准的 1024x768 @ 60Hz，带有模拟 VGA 或 HDMI 输入，具体取决于板卡的输出（即几乎任何 PC 显示器）。
- 用于显示器的 VGA 或 HDMI 电缆（如适用）
- USB 键盘
- 一只 USB 鼠标
- 一个 USB hub，如果键盘和鼠标没有组合在一个 USB 插头中

请注意，Z-Turn Lite 板没有显示器输出。因此，必须将具有 HDMI 端口的 Z-Turn Lite IO Cape board 连接到它以用于桌面使用场景。

使用 Zybo 时，显示器可以连接到 HDMI 端口以及 VGA 端口。不支持 Zedboard 的 HDMI 输出端口。

建议使用无线键盘/鼠标组合，因为它不需要 USB hub，并防止由于意外拉扯 USB 电缆而对板上的 USB 端口造成物理损坏。

在 Zedboard 和 Z-Turn Lite 上，键盘和鼠标的连接是通过 Micro B 到 Type A 母 USB 电缆完成的，该电缆随 Zedboard 和可能随 Z-Turn Lite 一起提供（取决于购买的物品）。

在另外两块板上，标准的 USB A 型母连接器（如 PC 的 USB 插头）可用于连接外围设备。

还需要：

- 一个可靠的 SD 卡（用于 Zedboard）或 MicroSD（用于 Z-Turn Lite、MicroZed 和 Zybo）具有 4GB 或更多，最优选由 Sandisk 制造的卡。不推荐（可能）随板子附带的卡，因为据报告将它与 Xillinux 一起使用会出现问题。
- 推荐：(Micro)SD 卡和 PC 之间的 USB 适配器，用于将 image 和 boot file 写入卡。如果 PC 计算机具有用于 SD 卡的内置插槽，则这可能是不必要的。 Zynq 板本身也可以用来写入 SD 卡，但这有点困难。

2.2 下载发行版

Xillinux 发行版可在 Xillybus 站点的下载页面下载：

http://xillybus.com/xillinux/

该发行版由两部分组成，它们作为两个单独的文件下载：

- (Micro)SD 卡的 raw image 由 file system 组成，Linux 在启动时可以看到
- boot partition kit 是一组文件，用于使用 Xilinx 的工具运行 implementation，以填充 boot partition。

在 3 部分中了解更多信息。

该发行版包括 Xillybus IP core 的演示，用于 processor 和 logic fabric 之间的轻松通信。此 demo bundle 的特定配置可能在某些应用程序上表现相对较差，因为它旨在用于简单的测试。

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 和 Xillinux 发行版，都可以免费使用，只要此使用与 “evaluation” 术语合理匹配。这包括将 IP core 集成到最终用户 designs 中，运行真实数据和现场测试。 IP 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

Vivado 2014.4及更高版本可用于 Xillinux 发行版中 logic design 的 compilation。

如果要使用 7z007s 或 7z014s 设备，则需要支持这些设备的 Vivado 版本（例如 Vivado 2016.4 和更高版本）。

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

Vivado 的任何版本都适用，包括

- WebPack Edition，假设目标设备被覆盖，可以无限期免费下载和使用。此版本涵盖了与 Z-Turn Lite、Zedboard、MicroZed 和 Zybo 配套的所有设备。
- Design Edition，需要购买许可证（但可以试用 30 天）。
- 任何可能已通过购买的主板专门许可的任何版本，因此仅限于特定 Zynq 设备。
- System 和任何其他提供 WebPack Edition 功能超集的版本也很好。

所有这些版本都涵盖了为 Zynq 实施 Xillybus 所需的 Xilinx 提供的 IP cores，无需额外的许可。

2.4 使用 FPGA design 的经验

使用 Z-Turn Lite、Zedboard、MicroZed 或 Zybo 时，无需使用 FPGA design 的经验即可在平台上运行发行版。使用另一块板需要一些使用 Xilinx 工具的知识，并且可能需要一些与 Linux kernel 相关的基本技能。

要充分利用该发行版，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 发行版是学习这些的一个很好的起点，因为它提供了一个简单的启动器 design 来进行试验。

3

建筑 Xillinux

3.1 概述

Xillinux 发行版旨在作为一个开发平台，而不仅仅是一个演示：在其准备在硬件上运行期间构建了一个用于定制 logic 开发和集成的即用型环境。因此，第一次测试运行的准备时间有点长（通常为 30 分钟，其中大部分是等待 Xilinx 的工具）。然而，这种漫长的准备工作缩短了集成定制 logic 的周期。

对于从 (Micro)SD 卡成功分发 Xillinux 的 boot 进程，它必须具有两个组件：

- boot partition 中的 FAT32 filesystem，由 boot loaders、FPGA 部分的 configuration bitstream（称为 PL ）和 Linux kernel 的 boot 的二进制文件组成。
- Linux安装的ext4 root file system。

下载的 Xillinux 的 raw image 几乎所有的东西都已经设置好了。它的boot partition只缺少三个文件，其中一个需要用Xilinx的工具生成，另外两个是从boot partition kit复制过来的。

本节将逐步详细介绍准备 (Micro)SD 的各种操作。

此过程包括以下步骤，这些步骤必须按照下面概述的顺序完成。

- 解压 boot partition kit
- 运行主 PL (FPGA) 项目的 implementation
- 将 Xillinux image 写入 (Micro)SD 卡
- 将三个文件复制到 (Micro)SD 卡的 boot partition

5.2 段讨论了如何与其他板一起工作。

3.2 解压 boot partition kit

将之前下载的 xillinux-eval-*board-XXX*.zip 文件解压到工作目录中。

重要的:

工作目录的路径不得包含空格。特别是 *Windows Desktop* 不合适，因为它的路径包括 *"Documents and Settings"*。

该捆绑包由以下目录（或其中一些）组成:

- verilog– 包含主要 logic 的项目文件和 Verilog 中的一些源代码（在 'src' 子目录中）
- vhdl– 包含主要 logic 的项目文件和一些源文件。 VHDL 中要编辑的文件位于 'src' 子目录中
- cores——Xillybus IP cores 的预编译二进制文件
- system–生成processor相关logic的目录
- bootfiles– 包含两个特定于板的文件，将被复制到 boot partition。
- vivado-essentials——processor 相关和通用 logic 的定义文件和构建目录，供 Vivado 使用。

有适用于 Z-Turn Lite、Zedboard、MicroZed 和 Zybo 板的捆绑包。如果使用另一块板，除了 5.2 部分列出的问题外，还必须相应地编辑 constraints 文件 vivado-essentials/xillydemo.xdc。

请注意，vhdl 目录包含 Verilog 文件，但它们都不需要用户编辑。

与 Xillybus IP core 的接口发生在相应 'src' 子目录中的 xillydemo.v 或 xillydemo.vhd 文件中。这是要编辑的文件，以便使用您自己的数据源和数据使用者尝试 Xillybus。

3.3 生成 bitstream 文件

3.3.1 介绍

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（不制作这些文件的副本）。 processor、它的互连和外围设备，以及 logic 使用的 FIFOs 都在 vivado-essentials/ 中定义，Vivado 在项目的 implementation 期间也填充了中间文件。

该项目的implementation可以基于Verilog或VHDL。

3.3.2 选择预期的 Zynq 部件（仅限 Z-Turn Lite）

只有demo bundle for Z-Turn Lite 需要此步骤。

使用文本编辑器在捆绑包的 root directory 中打开 select_part.tcl。该文件的最后四行是 Tcl 命令，用于设置为其创建 Vivado 项目的 Zynq 部件。这四行用 "#" 字符注释掉。

从这些行之一中取消注释一个 "#" 字符，以便选择 Z-Turn Lite 板上的 Zynq 部分。

如果您想稍后使用另一个 Zynq 部件，请相应地更改 Vivado 项目的设置并重新实现该项目。 select_part.tcl 仅在项目生成期间被引用，因此之后对其进行更改没有效果。

3.3.3 准备 xillydemo.vhd（仅限 VHDL 项目）

只有在以下情况下才需要执行此步骤：

- 该项目在VHDL
- demo bundle 适用于**除**Z-Turn Lite 之外的任何电路板。

如果这两个条件都满足，则必须编辑 vhdl/src/xillydemo.vhd：**删除**Xillydemo 的实体端口列表开头的以下三行：

```
PS_CLK : IN std_logic;
PS_PORB : IN std_logic;
PS_SRSTB : IN std_logic;
```

此外，**取消注释**体系结构定义中的以下行（删除 "--" 注释标记）：

```
-- signal PS_CLK :  std_logic;
-- signal PS_PORB :  std_logic;
-- signal PS_SRSTB :  std_logic;
```

无需对任何 Verilog 源文件进行更改。

3.3.4 生成 Vivado 项目

启动**Vivado 2014.4 或更高版本**。

在没有打开项目的情况下，选择 Tools > Run Tcl Script...，然后根据您的偏好在 verilog/ 或 vhdl/ 子目录中选择 **xillydemo-vivado.tcl**。一系列事件发生的时间不到一分钟。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
```

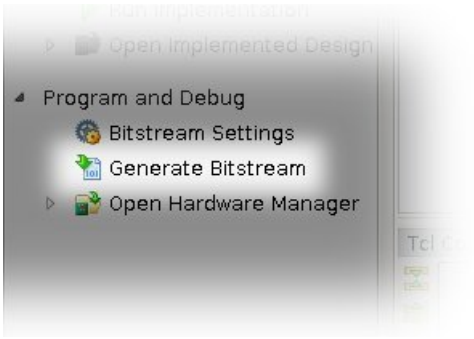
如果这不是 Tcl console 输出的最后一行，那就是出了问题。

在这个阶段可以有Critical Warnings，但没有错误。但是如果项目已经生成（即script已经运行），再次尝试运行script会导致如下错误：

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

3.3.5 Implementation项目

创建项目后，运行 implementation：单击左侧 Flow Navigator bar 上的 “Generate Bitstream”。



可能会出现一个弹出窗口，询问是否可以启动 synthesis 和 implementation——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，其中一些可能被归类为关键。应该没有错误。

将出现一个弹出窗口，通知 bitstream 已成功生成，并提供下一步操作的选择。任何选项都可以，包括选择 “Cancel”。

bitstream 文件 xillydemo.bit 可在 vivado/xillydemo.runs/impl_1/ 中找到。

implementation 永远不会失败。然而，有一些错误情况值得一提：

- 贴片机在 VHDL design 上出现 “IO placement is infeasible” 失败。如果在带有 VHDL 的 implementation 上发生这种情况，请确保按上述要求对 xillydemo.vhd 进行了编辑。
- write_bitstream 失败并出现 DRC 错误，抱怨 PS_CLK、PS_PORB 和 PS_SRSTB 未指定、未路由且未受约束，然后再说一次 - 请确保按上述要求编辑了 xillydemo.vhd。
- 说 “Timing constraints weren’t met” 时出错。集成自定义 logic 时可能会发生这种情况，导致工具无法满足 timing 要求。这意味着 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，允许用户生成 bitstream 文件，但 FPGA 的行为不能得到保证。为了防止产生这样的 bitstream，timing 故障会通过一个小的 Tcl script、“showstopper.tcl” 提升到错误级别，它会在 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 中移除 showstopper.tcl。
- 任何其他错误很可能是用户所做更改的结果，应按情况处理。

3.4 用 image 加载 (Micro)SD

3.4.1 一般的

此任务的目的是将 image file 写入 (micro)SD 卡。该文件名为 xillinux-2.0a.img.gz，下载为压缩文件（gzip 格式）。

(Micro)SD卡的这个image，是为boot准备的，除了三个文件，写入卡后添加。

这个 image 应该被解压缩，然后写入 (Micro)SD 卡的第一个 sector 和上。有几种方法和工具可以做到这一点。接下来推荐几种方法。

image 包含一个 partition table、一个用于放置初始 boot files 的部分填充 FAT file system 和 ext4 类型的 Linux root file system。第二个partition几乎被所有Windows电脑忽略了，所以(Micro)SD卡可能会显得容量很小（16 MB左右）。

编写 full disk image 不是针对普通计算机用户的操作，因此需要在 Windows 计算机上使用特殊软件，并在 Linux 上格外小心。以下段落解释了如何在任一操作系统上执行此操作。

如果 (Micro)SD 卡没有 USB 适配器（或计算机上的专用插槽），则板本身可用于写入 image，如 3.4.4 段所述。

重要的:

将 *image* 写入 *(Micro)SD* 会不可恢复地删除它之前可能包含的任何内容。强烈建议复制其现有内容，可能使用用于编写 *image* 的相同工具。

3.4.2 加载 image (Windows)

在 Windows 上，需要一个特殊的应用程序来复制 image，例如 USB Image Tool。此工具适用于使用 USB 适配器访问 (Micro)SD 卡的情况。

某些计算机（尤其是笔记本电脑）内置了 (Micro)SD 插槽，可能需要使用其他工具，例如 Win32 Disk Imager。运行 Windows 7 时也可能出现这种情况。

这两种工具都可以从网络上的各个站点免费下载。以下演练假设使用 USB Image Tool。

对于图形界面，运行 “USB Image Tool.exe”。当主窗口出现时，插入 USB 适配器，选择出现在左上角的设备图标。确保您位于 drop down menu 左上角的 “Device Mode”（与 “Volume Mode” 相对）中。单击 Restore 并将文件类型设置为 “Compressed (gzip) image files”。选择下载的 image file (xillinux-2.0a.img.gz)。整个过程大约需要4-5分钟。完成后，卸载设备（“安全移除硬件”）并拔下它。

在某些机器上，GUI 将无法运行，并显示软件初始化失败的错误。在这种情况下，可以使用命令行替代方案，或者需要安装 Microsoft .NET framework 组件。

或者，这可以从命令行完成（如果尝试运行 GUI 失败，这是一个快速的替代方法）。这分两个阶段完成。首先，获取设备编号。在 DOS Window 上，将目录更改为应用程序解压缩到的位置（典型会话如下）：

```
C:\usbimage>usbitcmd l

USB Image Tool 1.57
COPYRIGHT 2006-2010 Alexander Beug
http://www.alexpage.de

  Device    | Friendly Name              | Volume Name | Volume Path | Size
 ------------------------------------------------------------------------------
      2448 | USB Mass  Storage Device   |             | E:\         |   4024 MB
```

（注意 "usbitcmd" 后面的字符是字母 “l” 而不是数字 “1”）

现在，当我们有了设备号后，我们实际上可以写（”restore”）：

```
C:\usbimage>usbitcmd r 2448 \path\to\xillinux-2.0a.img.gz /d /g

USB Image Tool 1.57
COPYRIGHT 2006-2010 Alexander Beug
http://www.alexpage.de

Restoring backup to "USB Mass  Storage Device USB Device" (E:\)...ok
```

同样，这应该需要大约 4-5 分钟。当然，将数字 2448 更改为您在第一阶段获得的任何设备编号，并将 `\path\to` 替换为 (Micro)SD 卡的 image 在您的计算机上存储的路径。

3.4.3 加载 image (Linux)

重要的:

原始复制到设备是一项危险的任务：一个微不足道的人为错误（通常选择错误的目标磁盘）可能会导致计算机硬盘中的所有数据无法恢复地丢失。在按下 *Enter* 之前请三思，如果您不习惯 *Linux*，请考虑在 *Windows* 中执行此操作。

如前所述，将正确的设备检测为 (Micro)SD 卡非常重要。最好插入 USB 连接器，并在主日志文件（ /var/log/messages 或 /var/log/syslog ）中查找类似内容：

```
Sep  5 10:30:59 kernel: sd 1:0:0:0: [sdc] 7813120 512-byte logical blocks
Sep  5 10:30:59 kernel: sd 1:0:0:0: [sdc] Write Protect is off
Sep  5 10:30:59 kernel: sd 1:0:0:0: [sdc] Assuming drive cache: write through
Sep  5 10:30:59 kernel: sd 1:0:0:0: [sdc] Assuming drive cache: write through
Sep  5 10:30:59 kernel:  sdc: sdc1
Sep  5 10:30:59 kernel: sd 1:0:0:0: [sdc] Assuming drive cache: write through
Sep  5 10:30:59 kernel: sd 1:0:0:0: [sdc] Attached SCSI removable disk
Sep  5 10:31:00 kernel: sd 1:0:0:0: Attached scsi generic sg0 type 0
```

输出可能略有不同，但这里的重点是看 kernel 给新磁盘起什么名字。上例中的“sdc”。

解压 image file：

```
# gunzip xillinux-2.0a.img.gz
```

将 image 复制到 (Micro)SD 卡上很简单：

```
# dd if=xillinux-2.0a.img of=/dev/sdc bs=512
```

当然，您应该指向您发现是闪存驱动器的磁盘。

重要的:

以 */dev/sdc* 为例。请勿使用此设备，除非它恰好与您计算机上识别的设备相匹配。

并验证

```
# cmp xillinux-2.0a.img /dev/sdc
cmp: EOF on xillinux-2.0a.img
```

请注意响应：在 image file 上达到 EOF 的事实意味着其他所有内容比较正确，并且闪存比实际使用的空间更多。如果 cmp 什么也没说（这通常被认为是好的），它实际上意味着有问题。最有可能的是，生成了一个常规文件 "/dev/sdc"，而不是写入设备。

3.4.4 使用 Zynq 板加载 image

上面的 3.4.3 段落描述了如何使用 Linux 机器和 USB 适配器加载 image。可以使用 Zynq 板本身，运行板随附的示例 Linux 系统。基本上，可以遵循相同的指令，使用 /dev/mmcblk0 作为目标设备（而不是 /dev/sdc ）。

当从 QSPI flash 以及 SD 卡上的示例 Linux 系统（如果随板提供）完成 boot 过程时，这工作正常。这是因为与 Xillinux 不同，它完全从 RAM 运行，并且在 boot 完成后不使用 SD 卡。因此，如果 SD 卡用于 boot，则可以将其拔出，然后插入另一张用于写入 image。

如何让 Zynq 板访问 (Micro)SD 的 image 和 boot partition 文件是一个偏好和 Linux 知识的问题。有几种方法可以通过网络执行此操作，但最简单的方法是将这些文件写入密钥磁盘，并将其连接到 USB OTG 端口。安装 disk-on-key

```
> mkdir /mnt/usb
> mount /dev/sda1 /mnt/usb
```

然后可以在 /mnt/usb/ 上读取密钥磁盘上的文件。

在 Zedboard 上，确保已安装 JP2 跳线，以便 USB 端口由 5V 供电。

3.5 将文件复制到 boot partition

最后阶段放置 boot 的必要文件：

- 将 boot partition kit 的 bootfiles/ 子目录中的 boot.bin 和 devicetree.dtb 复制到 (Micro)SD 卡的 boot partition （第一个 partition ）中。
- 复制在 3.3 部分中生成的 xillydemo.bit（从 verilog/ 或 vhdl/ 子目录，以选择的为准）。

复制这些文件之前： 如果 (Micro)SD 的 image 刚刚写入卡，请拔下 USB 适配器，然后将其重新连接到计算机。如果 Zynq 板用于写入 raw image，请将 (Micro)SD 卡从插槽中拔出，然后重新插入。

这对于确保计算机是最新的 (Micro)SD 卡的 partition table 是必要的。

在 Linux 系统上，可能需要手动安装第一个 partition（例如 /dev/sdb1 ）。大多数计算机会自动执行此操作。

例如，当 Zynq 板本身用于此目的时，键入

```
> mkdir /mnt/sd
> mount /dev/mmcblk0p1 /mnt/sd
```

并将文件复制到 /mnt/sd/。

在 Windows 系统上，插入 (micro)SD 卡将显示单个 “disk”，带有单个文件 uImage。这是将文件复制到的正确目的地。

完成后，正确卸载 (Micro)SD 卡，然后将其从计算机上拔下，例如

```
> umount /mnt/sd
```

或 Windows 上的 “remove the disk safely”。

3.6 boot partition 中的文件

在尝试 boot 之前，请确认 boot partition 已按如下方式填充。

为了使 boot 成功，(micro)SD 卡的第一个 partition（boot partition）中需要存在四个文件：

- uImage——Linux kernel binary。这是 Xillinux 的 (Micro)SD image 写入卡后 boot partition 中唯一的文件。 kernel 是独立于板的。
- boot.bin——最初的 bootloader。该文件包含最初的 processor 初始化和 U-boot 实用程序，并且在板与板之间存在显着差异。
- devicetree.dtb– Device Tree Blob 文件，其中包含 Linux kernel 的硬件信息。
- xillydemo.bit– PL (FPGA) 编程文件，在 3.3 部分生成

4

启动 boot

4.1 跳线设置

为了让板子从 (Micro)SD 卡执行 boot，需要修改跳线设置。下面对每个板的设置进行了详细说明。

4.1.1 Zedboard

正确设置如下页的图像所示。

通常，需要进行以下跳线更改（但您的电路板可能一开始就设置不同）：

- 为 JP2 安装跳线，为 USB 设备提供 5V。
- JP10 和 JP9 从 GND 移动到 3V3 位置，该行中的其他三个仍然连接到 GND。
- 为 JP6 安装跳线（CES silicon 需要，请参阅 Zedboard 的 Hardware Guide 的第 34 页）。

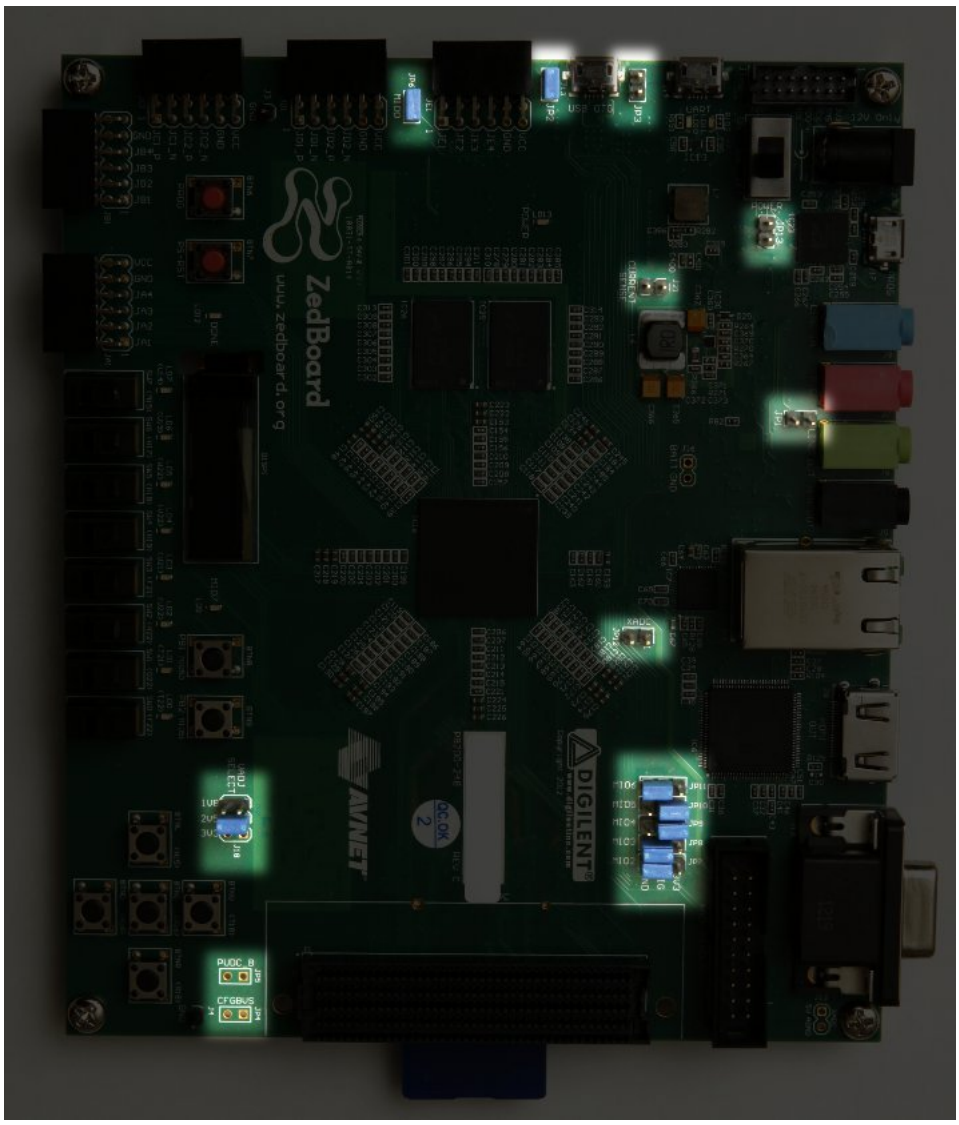
重要的:

所需设置与 *Zedboard Hardware User Guide* 中详述的设置不同，因为 *JP2* 是跨接的，因此连接到板上的 *USB* 设备（*USB* 键盘和鼠标）接收它们的 *5V* 电源。

4.1.2 MicroZed

从 MicroSD 卡执行 Xillinux 的 boot 的正确跳线设置如下：

- JP1: 1-2 (GND)

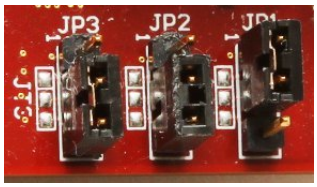


Zedboard 上突出显示的跳线设置

- JP2: 2-3 (VCC)
- JP3: 2-3 (VCC)

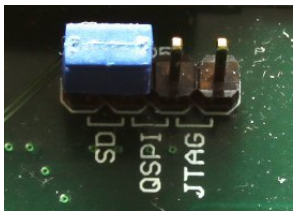
给定具有默认设置的板，只需移动 JP2。

正确的跳线设置如下所示：



4.1.3 Zybo

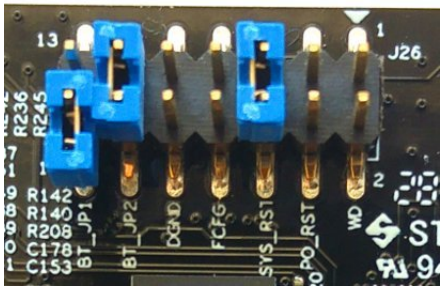
boot mode 由靠近 VGA 连接器的跳线选择，应设置在标有 “SD” 的两个引脚上，如下图所示：



其他跳线的设置取决于所需的操作模式。例如，可以将电源跳线设置为从外部 5V 电源或从 UART USB 插孔供电——对于成功的 boot 和 Xillinux，两者都适用。

4.1.4 Z-Turn Lite

Ethernet 连接器旁边的跳线（标记为 J26 ）确定 boot mode，应按如下方式设置：



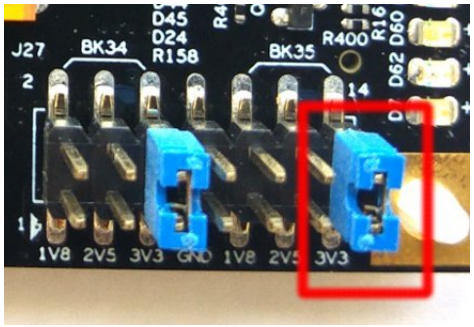
- BT_JP1 跳线不应放置（或如图所示放置，即仅连接其中一个引脚）
- BT_JP2 跳线应放置
- FCFG 和 PO_RST 不应放置

与 boot 操作无关，这两个跳线有些重要：

- SYS_RST 跳线可以通过按下板的 K2 按钮来重置 Zynq 的 processor。
- WD (Watchdog) 跳线允许从 processor 启用板载 watchdog 芯片。 Xillinux无论是否放置，都能正常执行boot。

DGND 引脚对只是接地的两个引脚。在这些上面放置一个跳线没有效果。

此外，如果使用 HDMI 视频输出（通过 Cape IO 板），则 Zynq 设备的 bank 35 必须由 3.3V 电源驱动。这是通过 J27 的 BK35 组上的 3V3 跳线建立的，位于靠近 MicroSD 卡的板角（见下图）。



4.2 连接外围设备

板卡可以连接以下通用硬件：

- Z-Turn Lite 和 Z-Turn Lite IO Cape board：连接到 Cape board 的 HDMI 连接器的计算机显示器。或者，通过 HDMI/DVI 适配器或电缆将 DVI 输入连接到板的 HDMI 插头。
- Zedboard / Zybo：模拟 VGA 连接器的计算机显示器。
- Zybo：带有 HDMI 输入的计算机显示器。或者，通过 HDMI/DVI 适配器或电缆将 DVI 输入连接到板的 HDMI 插头。

- Zedboard/Zybo/Z-Turn Lite：USB (OTG) 连接器的鼠标和键盘。

  在 Zybo 上，有一个类似 PC 的 USB 插头。在 Zedboard 和 Z-Turn Lite 上，这通过 Zedboard 随附的 USB 母电缆（较短）。当以 “kit” 配置购买时，此电缆还随 Z-Turn Lite 一起提供。

  如果没有这些，系统将成功执行 boot，并且在系统运行时连接和断开键盘和鼠标没有问题——系统在任何给定时刻检测并使用它连接的任何键盘和鼠标。

  请注意，在 Zedboard 上，必须安装 JP2 才能使此 USB 端口正常工作。

- Ethernet 端口对于常见的网络任务是可选的。如果连接的网络有 DHCP server，Linux 机器会自动配置网络。

- UART USB 端口可以连接到 PC，但在大多数情况下，对于 Zedboard 和 Zybo，这不是必需的。一些boot messages在那里发送，当boot完成时在这个接口上发出一个shell prompt。

  这对于在 boot 期间调试故障，或者当 PC 监视器或键盘丢失或无法正常工作时很有用。

  对于 Z-Turn Lite，需要一个外部 USB 适配器来连接板子的 3.3V UART 信号。该适配器可能包含在电路板上，如 Xillybus 的 web page 关于 Z-Turn Lite 的说明。

4.3 为电路板供电

4.3.1 初步诊断

当遵循上述构建说明时，boot 期间的故障很少见。常见的原因包括：

- 跳线设置不正确（参见第 4.1 段）。
- 使用不是 Sandisk 制造的 (Micro)SD 卡。即使卡看起来工作正常，分散的数据损坏也很容易被忽视，但会导致似乎有完全不同原因的错误。
- (Micro)SD image 写入卡不完整或错误
- U-boot 从开发板的 QSPI flash 加载旧的和不充分的环境变量。见 4.3.3 段。
- 未遵循说明，通常是由于在第一次尝试构建系统时尝试调整系统。

正确的 UART 设置是 115200 baud、8 data bits、1 stop bits，没有 flow control。一些文本应在板通电后不迟于 4 秒内出现。发生这种情况的唯一要求是在 (Micro)SD 卡上的第一个 (FAT32) partition 中找到 boot.bin 文件。

如果板子上电后没有任何反应：

- 验证是否从与电路板类型匹配的 boot partition kit 复制了正确的 boot.bin。该套件的文件名表明它应该与哪个板一起使用。
- 验证 UART 到计算机的链接是否正常工作，可能与 QSPI flash 上的示例 Linux 或（可能）与板一起到达的 SD 卡上。请注意，Linux 作为 UART terminal 应用程序的 host，可能无法与某些 UART/USB 转换器正常工作，因此在 Windows 下尝试 Tera Term 可能是唯一的选择。

如果 U-boot 在 console 上发出消息，但 boot 进程失败，则与 4.3.5 段中的脚本进行比较可能会有所帮助。本节的其余部分还可能包含相关信息，以帮助您了解问题所在。

4.3.2 boot 进程完成时

boot流程结束，在UART console上给出一个shell prompt，无需手动登录。即便如此，root user 的密码设置为空，因此如果需要，以 root 身份登录不需要密码。

重要的:

与主板随附的 *(Micro)SD* 卡上的 *Linux* 样本不同，*Xillinux* 的 *root file system* 永久驻留在 *(Micro)SD* 卡上，并在系统启动时写入。 *Linux* 系统应在关闭电路板电源之前正确关闭以保持系统稳定，就像任何 *PC* 计算机一样，即使在突然断电后通常会观察到正确的恢复。

Z-Turn Lite、Zedboard 和 Zybo 的注意事项：

- 在 shell prompt 处键入 “startx” 以启动 LXDE graphical desktop。 desktop 需要大约 15-30 秒来初始化。如果没有任何反应，监控 OLED 显示器上的活动表有助于判断是否发生了什么事（只有 Zedboard 有这个 OLED 显示器）。
- 从加载 logic fabric 到 Linux kernel 启动，屏幕上会显示带有白色背景的 Xillybus 徽标屏幕保护程序。它还会显示操作系统何时将屏幕置于 “blank” 模式，这是系统空闲时的正常情况，或者当 X-Windows 系统尝试操纵图形模式时。
- **蓝色**背景上的 Xillybus 屏幕保护程序或屏幕上的随机蓝色条纹表明图形界面存在数据不足的问题。除非知道明显的原因，否则永远不会发生这种情况，并且应该报告。

4.3.3 U-boot 环境变量

Xillinux在boot过程中依赖U-boot加载xillydemo.bit、kernel image和device tree。该实用程序提供了多种 boot 配置，并具有一个简单的命令行界面，允许对设置进行试验和

修改。

在 U-boot 启动后立即在 UART console 上键入任何字符即可到达 U-boot 的 shell：

```
U-Boot 2013.07 (Mar 15 2014 - 22:59:21)

Memory: ECC disabled
DRAM:  512 MiB
MMC:   zynq_sdhci: 0
SF: Detected S25FL129P_64K/S25FL128S_64K with page size 64 KiB, total 16 MiB
*** Warning - bad CRC, using default environment

In:    serial
Out:   serial
Err:   serial
Net:   Gem.e000b000
Hit any key to stop autoboot:  1
```

U-boot 总是尝试从 QSPI flash 检索保存的环境变量。显示 “bad CRC” 的 warning 表示未找到有效数据，因此 U-boot 恢复为其硬编码的默认环境。这对于来自 (Micro)SD 卡的 Xillinux 的 boot 不是错误，因为 Xillinux 的 U-boot 已正确设置所有环境变量。

重要的:

请注意，即使系统从 *(Micro)SD* 卡执行 *boot*，环境变量也是从板上的 *QSPI flash* 获取的。如果 *QSPI flash* 包含与不同 *boot* 场景匹配的环境变量，则 *U-boot* 可能会因变量不足而导致 *boot* 进程失败。

当一秒钟没有按下任何键时，U-boot 将根据其环境变量（从 QSPI flash 加载的环境变量或硬编码的默认设置）继续。更准确地说，它执行 “bootcmd” 变量的内容，默认情况下为 “run $modeboot”。反过来，“modeboot” 由 U-boot 动态设置，具体取决于从何处加载 U-boot，因此在常规 Xillinux boot 上显示 “sdboot”。 “sdboot”变量包含一系列执行Xillinux的boot进程的命令。

U-boot的command-line shell有一个“help”命令，里面列出了所有命令及其含义。一些有用的命令是

- help *command*——显示关于 *command* 的帮助
- env print–打印所有环境变量的当前值
- env set–设置某个环境变量的值

- env default -a——将所有环境变量设置为其硬编码默认值。
- saveenv–将当前环境变量保存到QSPI flash（不是MicroSD/SD卡）。

特别是，当 U-boot 使 boot 进程失败时，列表中的最后两个命令很重要。如果说 “bad CRC, using default environment” 的 warning不是由 U-boot 发布的，则它依赖于存储的变量。为了使用默认变量（对于 Xillinux 是正确的），去

```
xillinux-uboot> env default -a
## Resetting to default environment
xillinux-uboot> saveenv
Saving Environment to SPI Flash...
SF: Detected S25FL129P_64K/S25FL128S_64K with page size 64 KiB, total 16 MiB
Erasing SPI flash...Writing to SPI flash...done
```

如果存储的环境变量有任何理想的变化，它们当然也会被删除。

4.3.4 设置自定义 Ethernet MAC 地址

默认情况下，Linux 依赖于 U-boot 设置的 Ethernet MAC 地址。

为了更改 MAC 地址，可以添加由 Network Manager 读取的配置文件。例如，将以下内容复制到名为 /etc/NetworkManager/system-connections/eth0 的文件中：

```
[connection]
id=eth0
type=ethernet

[ethernet]
cloned-mac-address=00:11:22:33:44:55
mac-address=00:0A:35:00:01:22
```

有必要更改此文件的 permissions，以便 Network Manager 信任该配置：

```
# cd /etc/NetworkManager/system-connections/
# chmod 0600 eth0
```

除“eth0”外，该目录中不应有其他文件。如果还有其他文件，请将其删除。

当Linux重新启动时，MAC地址将为00:11:22:33:44:55。

还可以借助命令行实用程序 “nmcli” 执行相同的任务。然而，在图形桌面上执行此操作更容易：单击桌面右下角的 Network Manager 图标。该图标看起来像一个连接到墙上

的 Ethernet 插头。选择 “Edit Connections...”，然后选择 “Wired connection 1”，然后单击 “Edit”。在 “Cloned MAC address” 旁边写入新的 MAC 地址，然后单击 “Save”。Linux 重新启动后将使用新的 MAC 地址。

另一种方法是更改 U-boot 的环境变量之一。请注意，此方法不适用于某些版本的 Xillinux，因为 U-boot 并不总是能够访问 QSPI flash，因此并不总是支持 “saveenv” 命令。

由于环境变量存储在 QSPI flash 上，因此 MAC 地址永久绑定到硬件。这与之前的方法不同，之前的方法是将MAC地址绑定到(Micro)SD卡。

例如，在 U-boot 的 shell 上使用 USB UART console：

```
xillinux-uboot> set ethaddr 00:11:22:33:44:55
xillinux-uboot> saveenv
Saving Environment to SPI Flash...
Erasing SPI flash...Writing to SPI flash...done
```

稍后，在回收板的电源后，让 Linux 自动执行 boot：

```
root@localhost:~# ifconfig
eth1      Link encap:Ethernet  HWaddr 00:11:22:33:44:55
          inet addr:10.1.1.164  Bcast:10.1.1.255  Mask:255.255.255.0
          inet6 addr: fe80::211:22ff:fe33:4455/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:16 errors:0 dropped:0 overruns:0 frame:0
          TX packets:50 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:2720 (2.7 KB)  TX bytes:9230 (9.2 KB)
          Interrupt:54 Base address:0xb000
```

（上例中 IP 地址由 DHCP server 给出）

4.3.5 boot 期间的样本记录

作为参考，下面给出了 boot 期间的典型 UART 成绩单。该示例针对 Zedboard 显示，但板之间的差异很小。如果 boot 进程失败，一条错误消息可能会指出哪个阶段出了问题，以及可能的原因。

```
U-Boot 2013.07 (Aug 10 2014 - 11:28:31)

Zynq PS_VERSION = 0
Memory: ECC disabled
DRAM:  512 MiB
MMC:   zynq_sdhci: 0
```

```
SF: Detected S25FL256S_64K with page size 64 KiB, total 32 MiB
In:    serial
Out:   serial
Err:   serial
Net:   Gem.e000b000
Hit any key to stop autoboot:  1  0
Device: zynq_sdhci
Manufacturer ID: 3
OEM: 5344
Name: SL08G
Tran Speed: 50000000
Rd Block Len: 512
SD version 3.0
High Capacity: Yes
Capacity: 7.4 GiB
Bus Width: 4-bit
Booting Xillinux...
reading xillydemo.bit
4045675 bytes read in 295 ms (13.1 MiB/s)
  design filename = "xillydemo.ncd;HW_TIMEOUT=FALSE;UserID=0xFFFFFFFF"
  part number = "7z020clg484"
  date = "2014/03/11"
  time = "22:22:00"
  bytes in bitstream = 4045564
zynq_load: Align buffer at 10006f to 100080(swap 1)
reading uImage
4487928 bytes read in 326 ms (13.1 MiB/s)
reading devicetree.dtb
9531 bytes read in 16 ms (581.1 KiB/s)
## Booting kernel from Legacy Image at 03000000 ...
   Image Name:   Linux-4.4.30-xillinux-2.0
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    4487864 Bytes = 4.3 MiB
   Load Address: 00008000
   Entry Point:  00008000
   Verifying Checksum ... OK
## Flattened Device Tree blob at 02a00000
   Booting using the fdt blob at 0x2a00000
   Loading Kernel Image ... OK
   Loading Device Tree to 1fb4e000, end 1fb5353a ... OK

Starting kernel ...

Uncompressing Linux... done, booting the kernel.
[    0.000000] Booting Linux on physical CPU 0x0
[    0.000000] Initializing cgroup subsys cpuset
[    0.000000] Initializing cgroup subsys cpu
[    0.000000] Initializing cgroup subsys cpuacct
[    0.000000] Linux version 4.4.30-xillinux-2.0 (eli@ocho.localdomain) (gcc version 4.7.3 (Sourcery CodeBench Lite 2013.05-40) ) #1 SMP PREEMPT Tue D
[    0.000000] CPU: ARMv7 Processor [413fc090] revision 0 (ARMv7), cr=18c5387d
[    0.000000] CPU: PIPT / VIPT nonaliasing data cache, VIPT aliasing instruction cache
[    0.000000] Machine model: Xilinx Zynq
[    0.000000] bootconsole [earlycon0] enabled
[    0.000000] cma: Reserved 16 MiB at 0x1e800000
[    0.000000] Memory policy: Data cache writealloc
[    0.000000] PERCPU: Embedded 12 pages/cpu @dfb36000 s18880 r8192 d22080 u49152
[    0.000000] Built 1 zonelists in Zone order, mobility grouping on.  Total pages: 129920
[    0.000000] Kernel command line: console=ttyPS0,115200n8 console=tty0 consoleblank=0 root=/dev/mmcblk0p2 rw rootwait earlyprintk
[    0.000000] PID hash table entries: 2048 (order: 1, 8192 bytes)
[    0.000000] Dentry cache hash table entries: 65536 (order: 6, 262144 bytes)
[    0.000000] Inode-cache hash table entries: 32768 (order: 5, 131072 bytes)
[    0.000000] Memory: 493232K/524288K available (6155K kernel code, 294K rwdata, 2192K rodata, 312K init, 472K bss, 14672K reserved, 16384K cma-reser
[    0.000000] Virtual kernel memory layout:
[    0.000000]     vector  : 0xffff0000 - 0xffff1000   (   4 kB)
[    0.000000]     fixmap  : 0xffc00000 - 0xfff00000   (3072 kB)
[    0.000000]     vmalloc : 0xe0800000 - 0xff800000   ( 496 MB)
[    0.000000]     lowmem  : 0xc0000000 - 0xe0000000   ( 512 MB)
[    0.000000]     pkmap   : 0xbfe00000 - 0xc0000000   (   2 MB)
[    0.000000]     modules : 0xbf000000 - 0xbfe00000   (  14 MB)
[    0.000000]       .text : 0xc0008000 - 0xc082f0c4   (8349 kB)
[    0.000000]       .init : 0xc0830000 - 0xc087e000   ( 312 kB)
[    0.000000]       .data : 0xc087e000 - 0xc08c7840   ( 295 kB)
[    0.000000]        .bss : 0xc08c7840 - 0xc093da38   ( 473 kB)
[    0.000000] Preemptible hierarchical RCU implementation.
[    0.000000]  Build-time adjustment of leaf fanout to 32.
[    0.000000]  RCU restricting CPUs from NR_CPUS=4 to nr_cpu_ids=2.
```

```
[    0.000000] RCU: Adjusting geometry for rcu_fanout_leaf=32, nr_cpu_ids=2
[    0.000000] NR_IRQS:16 nr_irqs:16 16
[    0.000000] slcr mapped to e0800000
[    0.000000] L2C: platform modifies aux control register: 0x72360000 -> 0x72760000
[    0.000000] L2C: DT/platform modifies aux control register: 0x72360000 -> 0x72760000
[    0.000000] L2C-310 erratum 769419 enabled
[    0.000000] L2C-310 enabling early BRESP for Cortex-A9
[    0.000000] L2C-310 full line of zeros enabled for Cortex-A9
[    0.000000] L2C-310 ID prefetch enabled, offset 1 lines
[    0.000000] L2C-310 dynamic clock gating enabled, standby mode enabled
[    0.000000] L2C-310 cache controller enabled, 8 ways, 512 kB
[    0.000000] L2C-310: CACHE_ID 0x410000c8, AUX_CTRL 0x76760001
[    0.000000] zynq_clock_init: clkc starts at e0800100
[    0.000000] Zynq clock init
[    0.000000] clocksource: ttc_clocksource: mask: 0xffff max_cycles: 0xffff, max_idle_ns: 537538477 ns
[    0.000018] sched_clock: 16 bits at 54kHz, resolution 18432ns, wraps every 603975816ns
[    0.007925] ps7-ttc #0 at e0808000, irq=17
[    0.012173] sched_clock: 64 bits at 333MHz, resolution 3ns, wraps every 4398046511103ns
[    0.020052] clocksource: arm_global_timer: mask: 0xffffffffffffffff max_cycles: 0x4ce07af025, max_idle_ns: 440795209040 ns
[    0.031309] Console: colour dummy device 80x30
[    0.035629] console [tty0] enabled
[    0.039067] bootconsole [earlycon0] disabled
[    0.000000] Booting Linux on physical CPU 0x0
[    0.000000] Initializing cgroup subsys cpuset
[    0.000000] Initializing cgroup subsys cpu
[    0.000000] Initializing cgroup subsys cpuacct
[    0.000000] Linux version 4.4.30-xillinux-2.0 (eli@ocho.localdomain) (gcc version 4.7.3 (Sourcery CodeBench Lite 2013.05-40) ) #1 SMP PREEMPT Tue D
[    0.000000] CPU: ARMv7 Processor [413fc090] revision 0 (ARMv7), cr=18c5387d
[    0.000000] CPU: PIPT / VIPT nonaliasing data cache, VIPT aliasing instruction cache
[    0.000000] Machine model: Xilinx Zynq
[    0.000000] bootconsole [earlycon0] enabled
[    0.000000] cma: Reserved 16 MiB at 0x1e800000
[    0.000000] Memory policy: Data cache writealloc
[    0.000000] PERCPU: Embedded 12 pages/cpu @dfb36000 s18880 r8192 d22080 u49152
[    0.000000] Built 1 zonelists in Zone order, mobility grouping on.  Total pages: 129920
[    0.000000] Kernel command line: console=ttyPS0,115200n8 console=tty0 consoleblank=0 root=/dev/mmcblk0p2 rw rootwait earlyprintk
[    0.000000] PID hash table entries: 2048 (order: 1, 8192 bytes)
[    0.000000] Dentry cache hash table entries: 65536 (order: 6, 262144 bytes)
[    0.000000] Inode-cache hash table entries: 32768 (order: 5, 131072 bytes)
[    0.000000] Memory: 493232K/524288K available (6155K kernel code, 294K rwdata, 2192K rodata, 312K init, 472K bss, 14672K reserved, 16384K cma-reser
[    0.000000] Virtual kernel memory layout:
[    0.000000]     vector  : 0xffff0000 - 0xffff1000   (   4 kB)
[    0.000000]     fixmap  : 0xffc00000 - 0xfff00000   (3072 kB)
[    0.000000]     vmalloc : 0xe0800000 - 0xff800000   ( 496 MB)
[    0.000000]     lowmem  : 0xc0000000 - 0xe0000000   ( 512 MB)
[    0.000000]     pkmap   : 0xbfe00000 - 0xc0000000   (   2 MB)
[    0.000000]     modules : 0xbf000000 - 0xbfe00000   (  14 MB)
[    0.000000]       .text : 0xc0008000 - 0xc082f0c4   (8349 kB)
[    0.000000]       .init : 0xc0830000 - 0xc087e000   ( 312 kB)
[    0.000000]       .data : 0xc087e000 - 0xc08c7840   ( 295 kB)
[    0.000000]        .bss : 0xc08c7840 - 0xc093da38   ( 473 kB)
[    0.000000] Preemptible hierarchical RCU implementation.
[    0.000000]  Build-time adjustment of leaf fanout to 32.
[    0.000000]  RCU restricting CPUs from NR_CPUS=4 to nr_cpu_ids=2.
[    0.000000] RCU: Adjusting geometry for rcu_fanout_leaf=32, nr_cpu_ids=2
[    0.000000] NR_IRQS:16 nr_irqs:16 16
[    0.000000] slcr mapped to e0800000
[    0.000000] L2C: platform modifies aux control register: 0x72360000 -> 0x72760000
[    0.000000] L2C: DT/platform modifies aux control register: 0x72360000 -> 0x72760000
[    0.000000] L2C-310 erratum 769419 enabled
[    0.000000] L2C-310 enabling early BRESP for Cortex-A9
[    0.000000] L2C-310 full line of zeros enabled for Cortex-A9
[    0.000000] L2C-310 ID prefetch enabled, offset 1 lines
[    0.000000] L2C-310 dynamic clock gating enabled, standby mode enabled
[    0.000000] L2C-310 cache controller enabled, 8 ways, 512 kB
[    0.000000] L2C-310: CACHE_ID 0x410000c8, AUX_CTRL 0x76760001
[    0.000000] zynq_clock_init: clkc starts at e0800100
[    0.000000] Zynq clock init
[    0.000000] clocksource: ttc_clocksource: mask: 0xffff max_cycles: 0xffff, max_idle_ns: 537538477 ns
[    0.000018] sched_clock: 16 bits at 54kHz, resolution 18432ns, wraps every 603975816ns
[    0.007925] ps7-ttc #0 at e0808000, irq=17
[    0.012173] sched_clock: 64 bits at 333MHz, resolution 3ns, wraps every 4398046511103ns
[    0.020052] clocksource: arm_global_timer: mask: 0xffffffffffffffff max_cycles: 0x4ce07af025, max_idle_ns: 440795209040 ns
[    0.031309] Console: colour dummy device 80x30
[    0.035629] console [tty0] enabled
[    0.039067] bootconsole [earlycon0] disabled
```

```
[    0.043389] Calibrating delay loop... 1332.01 BogoMIPS (lpj=6660096)
[    0.130990] pid_max: default: 32768 minimum: 301
[    0.131116] Security Framework initialized
[    0.131135] Yama: becoming mindful.
[    0.131211] AppArmor: AppArmor initialized
[    0.131270] Mount-cache hash table entries: 1024 (order: 0, 4096 bytes)
[    0.131295] Mountpoint-cache hash table entries: 1024 (order: 0, 4096 bytes)
[    0.132028] Initializing cgroup subsys io
[    0.132059] Initializing cgroup subsys memory
[    0.132104] Initializing cgroup subsys devices
[    0.132133] Initializing cgroup subsys freezer
[    0.132156] Initializing cgroup subsys net_cls
[    0.132177] Initializing cgroup subsys perf_event
[    0.132200] Initializing cgroup subsys net_prio
[    0.132222] Initializing cgroup subsys pids
[    0.132274] CPU: Testing write buffer coherency: ok
[    0.132537] CPU0: thread -1, cpu 0, socket 0, mpidr 80000000
[    0.132602] Setting up static identity map for 0x82c0 - 0x82f4
[    0.310974] CPU1: thread -1, cpu 1, socket 0, mpidr 80000001
[    0.311078] Brought up 2 CPUs
[    0.311115] SMP: Total of 2 processors activated (2664.03 BogoMIPS).
[    0.311133] CPU: All CPU(s) started in SVC mode.
[    0.312116] devtmpfs: initialized
[    0.314713] evm: security.selinux
[    0.314734] evm: security.SMACK64
[    0.314748] evm: security.SMACK64EXEC
[    0.314761] evm: security.SMACK64TRANSMUTE
[    0.314775] evm: security.SMACK64MMAP
[    0.314804] evm: security.ima
[    0.314824] evm: security.capability
[    0.315239] VFP support v0.3: implementor 41 architecture 3 part 30 variant 9 rev 4
[    0.315600] clocksource: jiffies: mask: 0xffffffff max_cycles: 0xffffffff, max_idle_ns: 19112604462750000 ns
[    0.316774] pinctrl core: initialized pinctrl subsystem
[    0.318050] NET: Registered protocol family 16
[    0.320031] DMA: preallocated 256 KiB pool for atomic coherent allocations
[    0.323590] zynq_gpio e000a000.ps7-gpio: This is the Xillinux-1.3 compliant legacy GPIO driver.
[    0.324184] zynq_gpio e000a000.ps7-gpio: gpio at 0xe000a000 mapped to 0xe0814000
[    0.329041] hw-breakpoint: found 5 (+1 reserved) breakpoint and 1 watchpoint registers.
[    0.329099] hw-breakpoint: maximum watchpoint size is 4 bytes.
[    0.375012] vgaarb: loaded
[    0.377592] SCSI subsystem initialized
[    0.378094] usbcore: registered new interface driver usbfs
[    0.378220] usbcore: registered new interface driver hub
[    0.378369] usbcore: registered new device driver usb
[    0.378741] media: Linux media interface: v0.10
[    0.378849] Linux video capture interface: v2.00
[    0.379225] pps_core: LinuxPPS API ver. 1 registered
[    0.379263] pps_core: Software ver. 5.3.6 - Copyright 2005-2007 Rodolfo Giometti <giometti@linux.it>
[    0.379346] PTP clock support registered
[    0.379675] EDAC MC: Ver: 3.0.0
[    0.383475] NetLabel: Initializing
[    0.383515] NetLabel:  domain hash size = 128
[    0.383538] NetLabel:  protocols = UNLABELED CIPSOv4
[    0.383614] NetLabel:  unlabeled traffic allowed by default
[    0.384003] clocksource: Switched to clocksource arm_global_timer
[    0.384740] AppArmor: AppArmor Filesystem Enabled
[    0.399611] NET: Registered protocol family 2
[    0.400379] TCP established hash table entries: 4096 (order: 2, 16384 bytes)
[    0.400470] TCP bind hash table entries: 4096 (order: 3, 32768 bytes)
[    0.400579] TCP: Hash tables configured (established 4096 bind 4096)
[    0.400652] UDP hash table entries: 256 (order: 1, 8192 bytes)
[    0.400701] UDP-Lite hash table entries: 256 (order: 1, 8192 bytes)
[    0.400961] NET: Registered protocol family 1
[    0.402023] RPC: Registered named UNIX socket transport module.
[    0.402065] RPC: Registered udp transport module.
[    0.402090] RPC: Registered tcp transport module.
[    0.402114] RPC: Registered tcp NFSv4.1 backchannel transport module.
[    0.402789] hw perfevents: enabled with armv7_cortex_a9 PMU driver, 7 counters available
[    0.404341] futex hash table entries: 512 (order: 3, 32768 bytes)
[    0.404505] audit: initializing netlink subsys (disabled)
[    0.404585] audit: type=2000 audit(0.379:1): initialized
[    0.405111] Initialise system trusted keyring
[    0.405881] VFS: Disk quotas dquot_6.6.0
[    0.405984] VFS: Dquot-cache hash table entries: 1024 (order 0, 4096 bytes)
[    0.406373] squashfs: version 4.0 (2009/01/31) Phillip Lougher
[    0.407215] NFS: Registering the id_resolver key type
```

```
[    0.407288] Key type id_resolver registered
[    0.407315] Key type id_legacy registered
[    0.407361] nfs4filelayout_init: NFSv4 File Layout Driver Registering...
[    0.407468] jffs2: version 2.2. (NAND) (SUMMARY)  Â© 2001-2006 Red Hat, Inc.
[    0.407949] Allocating IMA MOK and blacklist keyrings.
[    0.409634] Key type asymmetric registered
[    0.409681] Asymmetric key parser 'x509' registered
[    0.409832] Block layer SCSI generic (bsg) driver version 0.4 loaded (major 248)
[    0.410040] io scheduler noop registered
[    0.410079] io scheduler deadline registered (default)
[    0.410137] io scheduler cfq registered
[    0.440104] Console: switching to colour frame buffer device 128x48
[    0.468985] xuartps e0001000.serial: clock name 'aper_clk' is deprecated.
[    0.469289] xuartps e0001000.serial: clock name 'ref_clk' is deprecated.
[    0.469614] e0001000.serial: ttyPS0 at MMIO 0xe0001000 (irq = 158, base_baud = 3125000) is a xuartps
[    1.278046] console [ttyPS0] enabled
[    1.282451] xdevcfg f8007000.ps7-dev-cfg: ioremap 0xf8007000 to e0872000
[    1.305388] brd: module loaded
[    1.315816] loop: module loaded
[    1.337113] libphy: Fixed MDIO Bus: probed
[    1.343139] libphy: XEMACPS mii bus: probed
[    1.348856] xemacps e000b000.ps7-ethernet: pdev->id -1, baseaddr 0xe000b000, irq 31
[    1.357688] ehci_hcd: USB 2.0 'Enhanced' Host Controller (EHCI) Driver
[    1.364433] ehci-pci: EHCI PCI platform driver
[    1.369044] ehci-platform: EHCI generic platform driver
[    1.381383] ohci_hcd: USB 1.1 'Open' Host Controller (OHCI) Driver
[    1.394522] ohci-pci: OHCI PCI platform driver
[    1.405966] ohci-platform: OHCI generic platform driver
[    1.418231] uhci_hcd: USB Universal Host Controller Interface driver
[    1.431756] usbcore: registered new interface driver usb-storage
[    1.445354] mousedev: PS/2 mouse device common for all mice
[    1.458696] i2c /dev entries driver
[    1.470432] device-mapper: uevent: version 1.0.3
[    1.482312] device-mapper: ioctl: 4.34.0-ioctl (2015-10-28) initialised: dm-devel@redhat.com
[    1.497966] sdhci: Secure Digital Host Controller Interface driver
[    1.511276] sdhci: Copyright(c) Pierre Ossman
[    1.522726] sdhci-pltfm: SDHCI platform and OF driver helper
[    1.537085] sdhci-arasan e0100000.ps7-sdio: No vmmc regulator found
[    1.550571] sdhci-arasan e0100000.ps7-sdio: No vqmmc regulator found
[    1.564036] mmc0: Invalid maximum block size, assuming 512 bytes
[    1.614085] mmc0: SDHCI controller on e0100000.ps7-sdio [e0100000.ps7-sdio] using ADMA
[    1.629895] ledtrig-cpu: registered to indicate activity on CPUs
[    1.644279] Key type dns_resolver registered
[    1.656171] Registering SWP/SWPB emulation handler
[    1.666380] mmc0: new high speed SDHC card at address aaaa
[    1.677100] mmcblk0: mmc0:aaaa SL08G 7.40 GiB
[    1.678486]  mmcblk0: p1 p2
[    1.703249] registered taskstats version 1
[    1.714453] Loading compiled-in X.509 certificates
[    1.727453] Key type encrypted registered
[    1.738464] AppArmor: AppArmor sha1 policy hashing enabled
[    1.750995] ima: No TPM chip found, activating TPM-bypass!
[    1.763635] evm: HMAC attrs: 0x1
[    1.774166] hctosys: unable to open rtc device (rtc0)
[    1.791487] md: Waiting for all devices to be available before autodetect
[    1.805427] md: If you don't use raid, use raid=noautodetect
[    1.819245] md: Autodetecting RAID arrays.
[    1.830359] md: Scanned 0 and added 0 devices.
[    1.841633] md: autorun ...
[    1.851105] md: ... autorun DONE.
[    1.861835] EXT4-fs (mmcblk0p2): couldn't mount as ext3 due to feature incompatibilities
[    1.877515] EXT4-fs (mmcblk0p2): couldn't mount as ext2 due to feature incompatibilities
[    1.906578] EXT4-fs (mmcblk0p2): mounted filesystem with ordered data mode. Opts: (null)
[    1.921636] VFS: Mounted root (ext4 filesystem) on device 179:2.
[    1.942290] devtmpfs: mounted
[    1.952285] Freeing unused kernel memory: 312K (c0830000 - c087e000)
[    2.204187] systemd[1]: System time before build time, advancing clock.
[    2.323264] NET: Registered protocol family 10
[    2.372338] random: systemd: uninitialized urandom read (16 bytes read, 6 bits of entropy available)
[    2.390906] random: systemd: uninitialized urandom read (16 bytes read, 6 bits of entropy available)
[    2.414805] systemd[1]: systemd 229 running in system mode. (+PAM +AUDIT +SELINUX +IMA +APPARMOR +SMACK +SYSVINIT +UTMP +LIBCRYPTSETUP +GCRYPT +GNU
[    2.447961] systemd[1]: Detected architecture arm.
[    2.491022] systemd[1]: Set hostname to <localhost.localdomain>.
```

它继续进行并进行 systemd 初始化。不超过 30 秒后，一个 shell prompt 出现如下。在最后一段输出之前可能会有几秒钟的暂停：

```
Ubuntu 16.04 LTS localhost.localdomain ttyPS0

localhost login: root (automatic login)

Last login: Thu Feb 11 16:28:21 UTC 2016 on ttyPS0
Welcome to the Xillinux-2.0 distribution for Xilinx Zynq.

You may communicate data with standard FPGA FIFOs in the logic fabric by
writing to or reading from the /dev/xillybus_* device files. Additional
pipe files of that sort can be set up with a custom Xillybus IP core.

For more information: http://www.xillybus.com.

To start a graphical X-Windows session, type "startx" at shell prompt.

root@localhost:~#
```

4.4 做第一台boot后不久

4.4.1 调整 file system 的大小

root file system 的 image 保持较小，以便尽可能快地将其写入设备。另一方面，没有理由不使用 (Micro)SD 卡的全部容量。

重要的:

尝试调整 *file system* 的大小时，存在擦除整个 *(Micro)SD* 卡内容的重大风险。因此建议尽早执行此操作，而此类事故的代价仅仅是重复 *(Micro)SD* 卡初始化（写入 *image* 并填充 *boot partition*）

起点通常如下：

```
# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/root       3.4G  2.8G  388M  89% /
devtmpfs        241M     0  241M   0% /dev
tmpfs           249M   72K  249M   1% /dev/shm
```

```
tmpfs           249M  7.2M  242M   3% /run
tmpfs           5.0M     0  5.0M   0% /run/lock
tmpfs           249M     0  249M   0% /sys/fs/cgroup
tmpfs            50M  4.0K   50M   1% /run/user/0
```

所以root filesystem就是2.8 GB，加上388 MB free。

第一阶段是重新分区 (Micro)SD 卡。在 shell prompt 处，键入

```
# fdisk /dev/mmcblk0
```

然后键入以下内容（另请参阅下面的会话记录）：

- d [ENTER]–删除partition
- 2 [ENTER]–选择partition编号2
- n [ENTER]——创建一个新的partition
- 按 [ENTER] 4 次以接受默认值：A primary partition，数字 2，从可能的最低 sector 开始，到可能最高的 sector 结束。
- w [ENTER] – 保存并退出。

如果在此序列中间出现问题，只需按 CTRL-C （或 q [ENTER] ）退出 fdisk 而不保存更改。在最后一步之前，(Micro)SD 卡上没有任何变化。

一个典型的会话如下所示。请注意，sector 编号可能会有所不同。

```
# fdisk /dev/mmcblk0

Welcome to fdisk (util-linux 2.27.1).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.


Command (m for help): d
Partition number (1,2, default 2): 2

Partition 2 has been deleted.

Command (m for help): n
Partition type
   p   primary (1 primary, 0 extended, 3 free)
```

```
   e   extended (container for logical partitions)
Select (default p):

Using default response p.
Partition number (2-4, default 2):
First sector (32768-15523839, default 32768):
Last sector, +sectors or +size{K,M,G,T,P} (32768-15523839, default 15523839):

Created a new partition 2 of type 'Linux' and of size 7.4 GiB.

Command (m for help): w
The partition table has been altered.
Calling ioctl() to re-read partition table.
Re-reading the partition table failed.: Device or resource busy

The kernel still uses the old table. The new table will be used at
the next reboot or after you run partprobe(8) or kpartx(8).
```

如果系统上显示的默认第一个 sector 与上面的不同，请选择系统的默认值，而不是此处显示的默认值。

在这个序列中，唯一可能偏离 fdisk 的默认值的地方是最后一个 sector，以使 file system 小于可能的最大值（但没有必要这样做）。

正如底部的 warning 所说，Linux 的 partition table 视图没有更新。按照建议，键入：

```
# partprobe
```

这应该不会对 console 产生任何输出。如果它确实抱怨无法通知 kernel partition 2 的变化，那么很可能是因为 partprobe 找不到它。换句话说，root partition 并不是 partition table 所说的那样。很可能是 fdisk 出了问题，应该修复，否则 Linux 将无法再次执行 boot。

如果 partprobe 没有声音，那么 partition table 没问题，但 file system 还没有调整大小；它只被赋予了调整大小的空间。所以在 shell prompt，键入

```
# resize2fs /dev/mmcblk0p2
```

预计会收到以下响应：

```
resize2fs 1.42.13 (17-May-2015)
Filesystem at /dev/mmcblk0p2 is mounted on /; on-line resizing required
old_desc_blocks = 1, new_desc_blocks = 1
The filesystem on /dev/mmcblk0p2 is now 1936384 (4k) blocks long.
```

block count 取决于 partition 的大小，因此可能会有所不同。

正如该实用程序所说，调整大小发生在积极使用的 file system 上。只要电源没有在中间丢失，这是安全的。

结果立即生效：无需重启。

使用 8 GB (Micro)SD 卡的典型会话：

```
# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/root       7.1G  2.8G  4.0G  42% /
devtmpfs        241M     0  241M   0% /dev
tmpfs           249M   72K  249M   1% /dev/shm
tmpfs           249M  7.2M  242M   3% /run
tmpfs           5.0M     0  5.0M   0% /run/lock
tmpfs           249M     0  249M   0% /sys/fs/cgroup
tmpfs            50M  4.0K   50M   1% /run/user/0
```

请注意，“df -h” 实用程序给出的大小是 1 GiB = 2^{30} 字节，它比 10^9 字节的 Gigabyte 大 7.1%。这就是 8 GB 卡在上面显示为 7.1 GiB 的原因。

4.4.2 允许远程 SSH 访问

root password 默认为无，允许任何用户以 root 身份登录，根本无需密码。因此，ssh 拒绝登录 root。

要纠正此问题，请在 shell prompt 上使用以下命令设置 root password：

```
# passwd
```

4.4.3 语言环境定义的 Compilation（如果需要）

在某些情况下，应用软件需要了解如何根据 locale settings 显示字符。最常见的原因是用ssh连接到板子的时候，因为ssh在设置shell session的时候把client的locale settings复制到server的环境中。

这可能会导致 warning messages 像

```
bash: warning: setlocale: LC_CTYPE: cannot change locale (en_US.UTF-8)
```

当出现这样的错误信息时，很明显缺少哪个 locale。要在发生错误之前解决此问题，请检查需要哪些 locales：

```
# locale
LANG=en_US.UTF-8
LANGUAGE=
LC_CTYPE="en_US.UTF-8"
LC_NUMERIC="en_US.UTF-8"
LC_TIME="en_US.UTF-8"
LC_COLLATE="en_US.UTF-8"
LC_MONETARY="en_US.UTF-8"
LC_MESSAGES="en_US.UTF-8"
LC_PAPER="en_US.UTF-8"
LC_NAME="en_US.UTF-8"
LC_ADDRESS="en_US.UTF-8"
LC_TELEPHONE="en_US.UTF-8"
LC_MEASUREMENT="en_US.UTF-8"
LC_IDENTIFICATION="en_US.UTF-8"
LC_ALL=
```

与可用的 locales 比较：

```
# locale -a
C
C.UTF-8
POSIX
```

在这个例子中，很明显缺少哪个 locale，所以让我们添加它：

```
# locale-gen en_US.UTF-8
Generating locales...
  en_US.UTF-8... done
```

请注意，必要的 locale 取决于建立 ssh 连接的计算机。来自世界各地的用户需要在他们的板上安装不同的 locales，才能实现流畅的 ssh 会话。 UART 端口上的 shell 基于默认包含的 POSIX locale。

由于 en_US.UTF-8 相当普遍，它已经安装在发行版中（尽管上面的示例会话显示相反）。

4.5 使用 desktop

Xillinux desktop（在 Z-Turn Lite、Zedboard 和 Zybo 上）就像任何 Lubuntu desktop 一样。由于 (Micro)SD 卡的数据带宽相对较低，应用程序的加载速度可能会有些慢，但 desktop 本身的响应速度相当快。

与任何 Ubuntu 发行版一样，可以使用 “apt-get” 安装其他软件包。

用 apt 升级整个 Ubuntu 操作系统是不可能的，并且会失败，使系统无法再次执行 boot。

4.6 关机/重启

要关闭系统电源，请选择桌面右下角的图标（如果可用），然后单击 “Shutdown”，或选择任何其他合适的选项。 “Lock Screen” 什么都不做。

或者，在 shell prompt 处键入以下内容：

```
# halt
```

当 UART console 上（以及屏幕上，如果存在）出现一条说明 “System Halted” 的文本消息时，可以安全地关闭电路板电源。

对于 reboot，包括将 bitstream 重新加载到 FPGA (PL) 部件，请在桌面菜单上选择 reboot 选项，或键入：

```
# reboot
```

请注意，这不一定会重置外部硬件组件，例如声音芯片。

4.7 从这里做什么

Zynq 板现在已成为一台运行 Linux 的计算机，用于各种用途。通过 Xillybus IP core 与 logic fabric 交互的基本步骤可以在 Getting started with Xillybus on a Linux host中找到。请注意，Xillybus 的 driver 已经安装在 Xillinux 发行版中，因此可以跳过指南中有关安装的部分。

5.1 段落是指将特定应用程序 logic 与 Linux 操作系统集成。

请注意，Xillinux 包括 gcc compiler 和 GNU make，因此可以直接在板上的 processors 上运行常规计算机程序的 compilation。 apt-get 也可以将其他软件包添加到发行版中。

5

进行修改

5.1 与定制 logic 集成

Xillinux 发行版设置为易于与应用程序 logic 集成。连接数据源和数据消费者的前端是 xillydemo.v 或 xillydemo.vhd 文件（取决于首选语言）。为了将 Xillybus IP core 用作 Linux host 和 logic fabric 之间的数据传输，可以忽略 boot partition kit 中的所有其他 HDL 文件。

可以将带有自定义 logic designs 的附加 HDL 文件添加到段落 3.3 中介绍的项目中，然后以与开始时相同的方式重新构建。要使用更新的 logic 执行系统的 boot，请将新的 xillydemo.bit 复制到 (Micro)SD 卡的 boot partition 中，覆盖现有的。请注意，可以使用 Zynq 板本身将 xillydemo.bit 复制到 boot partition 中，如 3.5 段所示。

无需重复初始分发部署的其他步骤，因此 logic 的开发周期相当快速和简单。

不支持通过 JTAG 对 PL 部件进行编程。

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，而不是试图模仿 FIFO 与 logic 的行为，至少在第一阶段不要。

将 Xillybus 与 block RAM 或 registers 连接时例外，在这种情况下应遵循 xillydemo 模块中所示的方法。

在 xillydemo 模块中，FIFOs 用于执行从 host 到达并返回给它的数据的 loopback。FIFOs 的两侧都连接到 Xillybus IP core，这使得 core 可以作为自己的数据源和数据消费者。

在更有用的场景中，FIFO 的一端只有一个连接到 Xillybus IP core，另一端连接到应用程序数据源或数据消费者。

xillydemo 模块中使用的 FIFOs 两边只接受一个通用的 clock，因为两边都由 Xillybus 的主 clock 驱动。在实际应用中，可能希望将它们替换为具有单独 clocks 用于读取和

写入的 FIFOs，从而允许数据源和数据消费者由 clock 而非 bus clock 驱动。通过这样做，FIFOs 不仅可以充当中介，还可以充当适当的 clock domain crossing。

请注意，对于从 FPGA 到 host 的 streams，Xillybus IP core 需要一个普通的 FIFO（与 First Word Fall Through 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
- Linux host 的基本概念：Getting started with Xillybus on a Linux host
- 编程应用：Xillybus host application programming guide for Linux
- 请求定制 Xillybus IP core：The guide to defining a custom Xillybus IP core

5.2 使用其他板

在尝试在 Z-Turn Lite、Zedboard、MicroZed 或 Zybo 以外的板上运行 Xillinux 之前，可能需要进行某些修改。

但是不建议尝试将 Xillinux 适配到其他硬件，因为过程很困难。经验表明，如果改装Xillinux的目的不是为了使用Xillybus IP core，那么从头开始比较容易。

这是需要注意的部分问题列表。

- 购买的板应该有一个 XML 文件作为参考（用作 ps7_system_prj.xml）。该文件包含 processor 的设置，包括 MIO 引脚的实际使用和 DDR 引脚的电气参数。推荐的做法是采用参考文件，至少作为起点。
- 如果采用 XML 文件作为参考，则 FPGA CLK1 (FCLK_CLK1) 必须设置为 100 MHz，无论参考文件说什么。
- 如果手动进行更改，则应注意 processor core 的 MIO 分配：ARM core 有 54 个 I/O pins，它们以固定位置路由到芯片上的物理引脚。 ARM core 在项目的 block design 中配置为将特定角色分配给这些引脚（例如 USB 接口、Ethernet 等），这些引脚必须与板上这些引脚的接线相匹配。
- 如果在 processor 的配置（即 XML file 中）中进行了更改，则必须根据从新 XML 文件派生的 FSBL (First Stage Boot Loader) 和 U-boot binary 重建 boot.bin。在 Vivado 的 block design 工具中所做的更改通过作为 FSBL 一部分的初始化例程生效。此例程写入 ARM processor 中的 registers，其值反映在 Vivado 中所做的设置，并导出到 SDK。注意Vivado项目中processor的参数可能不准确，所以FSBL要根据捆绑包中的XPS项目生成。要设置 U-boot 的源，请参阅 /usr/src/xillinux/uboot-patches/ 中的 README 文件。

- 或者，在某些情况下，借助 “poke” 功能，可以通过精确定位 registers 设置的更改来避免重建 boot.bin。见 5.6 段。
- 可能还需要在 devicetree.dtb 中进行更改，以反映新设置。现有 DTB 的源代码（DTS 格式）可以在 Linux kernel 的源代码中找到（参见 6.2 段落）。
- VGA/DVI 输出（如果适用）需要与预期的板相匹配。这是通过编辑 src/ 子目录中的 xillybus.v 文件来完成的。请注意，来自 “system” 模块的信号为 8 位宽，而在 xillybus.v 中截断为 4 位。因此，很容易将这些信号连接到 VGA/DVI 的任何编码器芯片。

5.3 改变系统中clocks的频率

ARM processor 的 core 提供四个 clocks 供 logic fabric 使用，通常称为 FCLK_CLKn。需要注意的是，它们的频率是由 FSBL (First Stage Boot Loader) 在加载 U-boot 之前设置的。

因此，即使 clocks 的频率设置在 Vivado 中，这些频率也仅对传播 timing constraints 和由 bare-metal 应用程序（即 SDK 上的 compiled）进行初始化有效。

如果硬件应用需要不同的频率，建议采取以下一系列措施:

- 在 Vivado 中更新 clock 的频率。
- 重建网表（这是更新 .ncf 文件中的 timing constraints 所必需的）
- 将工程导出到SDK，并以此为基础创建一个FSBL application project。
- 从 Vivado 的报告中了解所需设置所需的 registers 设置。
- 根据 “poke” 功能进行必要的调整，如第 5.6 段所述。

请参阅 Xilinx 的指南了解如何执行每个步骤（最后一步除外）的详细信息。

5.4 接管 PL logic 的 GPIO I/O 引脚

5.4.1 Z-Turn Lite

虽然 Z-Turn Lite 板本身没有提供方便的方式来访问 I/O 引脚以用于实验室目的，但将其连接到 Z-Turn Lite IO Cape board 会通过标准连接器暴露 68 I/O pins 和按钮，以及 HDMI 接口。

所有这些 68 针都连接到两个 40 针连接器 J3 和 J8，可以连接标准扁平带状电缆。 IO Cape board 有一些额外的连接器，它们与 J3 和 J8 共享引脚。由于 J3 和 J8 上提供

了所有附加连接器的引脚，因此 Xillydemo 的 top-level module 用于这些引脚的端口是名为 J3 和 J8 的向量，pin placement constraints 将它们路由到相应的连接器。

J3 和 J8 的 Verilog / VHDL 端口中的矢量对应于连接器的引脚号减 3：Verilog / VHDL 中的信号 J3[0] 进入物理引脚 J3/3。 J3[1] 到 J3/4 等，直到 J3[33] 到 J3/36。 J8 也是如此。

为简单起见，属于 J8 的所有管脚都在 xillydemo.v 和 xillydemo.vhd 中连接到 processor 的 GPIO 管脚，并且可以直接通过运行在 processor 上的软件进行控制。 J3 的所有引脚都由 xillydemo.v 和 xillydemo.vhd 驱动为低电平，并且可以通过修改相关的 xillydemo 模块文件轻松地被应用程序 logic 使用。

通过更改 xillydemo 模块中的接线，将引脚划分为 GPIO 和由 application logic 驱动的引脚也很容易更改。如果用作 GPIO 的管脚数量发生变化，则 xillybus 模块的 gpio_width 实例化参数（通用）也应相应更改。它目前为 35，占到 J8 连接器的 34 个 I/O 引脚，加上一个用于 Cape Board 按钮的 GPIO。

如前所述，板上还有额外的连接器，它们与 J3 和 J8 共享它们的引脚，仍然可以使用。但是，这需要查找 Cape Board 原理图中的哪些引脚。

这种引脚共享的一个副作用是，一些被替代连接器用作 I^2C 的引脚在 Cape board 上具有 pull-ups：J3[19]、J3[18]、J8[28] 和 J8[31]（使用 Verilog / VHDL 矢量信号表示法）。由于这些是板上带有电阻器的 pull-ups，因此即使在 J3 和 J8 连接器上使用这些引脚时它们也有效。

HDMI 连接器是独立的，不与 J3、J8 或任何其他连接器共用针脚。

5.4.2 Zedboard 和 Zybo

在 Zedboard 和 Zybo 板上，许多物理 I/O 引脚连接到 ARM processor 的 GPIO 端口 (PS)，这允许直接从 Linux 控制和监视这些引脚。然而，通常希望将这些物理引脚连接到 FPGA logic （即 PL ）。

将 Zynq 的 PL 引脚用于 I/O 的技术与任何 Xilinx FPGA 完全相同：信号暴露在顶层模块（xillydemo.v 或 xillydemo.vhd）中作为输入、输出或 inouts。这些信号的物理引脚分配发生在 xillydemo.xdc 中。

由于引脚用作 GPIO 信号，因此将它们从 processor 中取出并提供给 PL 部分。例如，XDC 文件中的以下行，如果我们希望 my_output 出现在引脚 U5 上，可以将

```
set_property -dict "PACKAGE_PIN U5 IOSTANDARD LVCMOS33" [get_ports "PS_GPIO[55]"]
```

替换为

```
set_property -dict "PACKAGE_PIN U5 IOSTANDARD LVCMOS33" [get_ports "my_output"]
```

。

但是进行这种替换会导致 PS_GPIO[55] 缺少引脚分配。尽管 Xilinx 的工具有可能会在 implementation 期间自动放置此端口，但建议为任何被驱逐的 PS_GPIO 分配一个 I/O 引脚。另一种方法是消除信号，如下所述。

因此，这些被驱逐的 PS_GPIO 信号有两种解决方案：

- 简单的方法：在设备上找到未使用的引脚，并将这些引脚分配给被驱逐的 PS_GPIO 信号。即使它不是一个非常干净的解决方案（GPIO 引脚只连接到板上的任何东西），它实际上是无害的，因为默认情况下 GPIOs 是输入。这些引脚上的电气状况仍然存在，除非 GPIO 被软件意外驱动（这不太可能）。例如，在 Zedboard 上，FMC 连接器通常会提供许多未使用的引脚。
- 更难的方法：减少 PS_GPIO 引脚的数量。这在 Zybo 上可能是必需的，它没有很多空引脚。

下面讨论第二种解决方案。例如，假设 PS_GPIO[55:48] 已从 XDC 文件中删除，以便将其引脚替换为来自 PL 的信号。请注意，如果需要来自较低 PS_GPIO 索引的引脚，则被驱逐的 PS_GPIO 信号应接管具有最高索引的引脚，然后将后者消除。没有可能消除一定范围的 PS_GPIO 指标，只能降低最大指标。

PS_GPIO 的宽度应在 xillydemo.v/vhd 中减小以反映在 XDC 文件中具有引脚分配的那些。

然而，这还不够。尝试在此状态下构建项目，将为这些引脚发布 critical warnings（可能声称这些引脚是多驱动的，带有 high-Z 和 GND ）。

要解决此问题，请在以下部分编辑 vivado-essentials/system.v：

```
generate
  for (i=0; i<56; i=i+1)
    begin: gpio
      assign gpio_tri_i[i] = processing_system7_0_GPIO[i];
      assign processing_system7_0_GPIO[i] = gpio_tri_t[i] ? 1'bz :
                                            gpio_tri_o[i];
    end
endgenerate
```

将索引范围（即 `i<56` 部分）减少到使用的 GPIOs 的数量（示例中为 48）。

根据 Vivado 的版本，可能还需要调整信号的宽度。

如果需要在 block design 中更正 GPIO 的宽度： 在 Vivado 的主窗口中，单击左侧栏中的 “Open Block Design”。右键单击 processor block（processing_system_7_0，带

有 ZYNQ 标记）并选择 “Customize block”。选择左列的 “MIO Configuration”，展开 “I/O Peripherals” 层次结构，展开 GPIO 层次结构（在底部）。 EMIO GPIO (Width) 参数目前为 56，即 GPIO 引脚数。将其减少到所需的数字（本例中为 48）。

5.5 使用 7020 MicroZed

可用于 MicroZed 的 boot partition kit 默认适用于 7010 MicroZed 板。但是，在对用于创建 Vivado 项目的 xillydemo-vivado.tcl 文件（即捆绑包中的 verilog/xillydemo-vivado.tcl 或 vhdl/xillydemo-vivado.tcl，取决于所选语言）进行微小更改后，可以使用 Vivado 使用 7020 MicroZed。

在解压缩套件之后（以及在 Vivado 中使用它之前），文件应该被编辑，改变行说

```
set thepart "xc7z010clg400-1"
```

（第 11 行附近）到

```
set thepart "xc7z020clg400-1"
```

其余的构建过程完全相同。

5.6 hardware registers (“poke”) 的前置 boot 操作

通常希望在不重新构建 boot.bin 文件的情况下对 ARM processor 的硬件设置进行轻微更改（第 5.3 段讨论了典型的重新构建顺序）。

例如，processor 的 MIO/EMIO 配置的微小变化会导致 registers 的设置发生一些变化，这可以通过查找系统设置导出到软件工具时生成的报告中的差异来很容易地推断出来。

processor 的 hardware registers 记录在 Xilinx 的 Zynq-7000 AP SoC Technical Reference Manual 中，也称为 TRM 或 ug585。

为了操作 registers，在 kernel 的 device tree 中添加了一个条目（通常通过编辑 Linux kernel 源中给出的相应 DTS 文件，请参阅第 6.2 段）。

在 device tree 的层次结构中的任何位置（最好在 “chosen” 条目之后）添加类似以下示例的条目：

```
poke {
        compatible = "xillybus,poke-1.0";
        sequence = < 0 0xf8002000 0
```

```
                0 0xf800200c 0
                0 0xf8002018 0
                1 0xf800200c 0x20
                0 0xf8002018 0
                0 0xf8002018 0
                1 0xf800200c 0x21
                0 0xf8002018 0
                0 0xf8002018 0
            >;
};
```

应更改 “sequence” 部分以设置所需的 register 读取和写入顺序。每个操作由 “sequence” 元素数组中的三个值定义。三元组的数量（以及操作）没有限制。上面的 “sequence” 条目的格式，tabs 和每行三个值，没有语法意义——重要的是每个三元组代表一个操作，如下所示：

- 第一个元素：读或写。值为 0 表示读取，否则表示写入。
- 第二个元素：地址。必须是 32 位对齐的（地址的 2 LSBs 必须为零）。
- 第三个元素：要写入的值。忽略读取操作。

这些操作按照 device tree 条目中列出的顺序执行，每个操作之间存在不可预知的延迟。

上面这个没有实际意义的例子，就是对processor的ttc2 (Triple Timer Counter 2)的registers进行了操作：前三个操作读取registers是为了演示。然后计数器被短暂启用，它的计数器值被读取两次以表明它正在改变，然后计数器被禁用。在此之后，再次读取计数器值两次，以表明它已停止。

这些操作的结果可以在 kernel 的 message log 中找到，它在 serial console (UART) 上可用，和/或在 shell prompt 上使用 dmesg 命令：

```
[    0.000000] poke read addr=f8002000: value=00000000
[    0.000000] poke read addr=f800200c: value=00000021
[    0.000000] poke read addr=f8002018: value=00000000
[    0.000000] poke write addr=f800200c: value=00000020
[    0.000000] poke read addr=f8002018: value=00000009
[    0.000000] poke read addr=f8002018: value=00004f68
[    0.000000] poke write addr=f800200c: value=00000021
[    0.000000] poke read addr=f8002018: value=000013ec
[    0.000000] poke read addr=f8002018: value=000013ec
```

当然，从 0xf8002018 读取的值会有所不同，因为它们是从运行的计数器中读取的。

register 修改发生在 kernel 的 boot 进程的早期，在任何 device driver 加载之前。但是请注意，U-boot 在 Linux 开始其 boot 进程之前设置了一些 ARM processor 的硬件外围设备，因此它们已经处于活动状态。另请注意，修改与 ARM processor 的基本功能（例如 clocks 和 interrupts ）相关的 registers 可能会破坏 processor 本身的正常功能。即使在执行 “poke” 时 kernel 禁用了 interrupts，也会发生这种情况。

尝试访问 kernel 的内存管理和/或硬件本身不允许的地址会导致 kernel Oops、kernel panic 并可能完全冻结。后两者导致 boot 故障。基于 “imprecise external abort (0x406)” 的 kernel panic 可能是由于试图访问硬件方面的非法地址。

此外，由于早期的 kernel console 消息在生成阶段存储在内部 memory buffer 中，并且仅在稍后阶段（设置 serial port 时）写入 console，因此早期冻结可能导致没有输出到 console —— 在 U-boot 的 “done, booting the kernel” 消息之后什么也没有出现。

因此，当 console 上没有出现 kernel messages 时，并不一定意味着 kernel 没有启动。这可能是在将存储在内存中的 kernel messages 写入 console 之前冻结的结果。原因可能是非法修改 register。

“poke” 功能是通过专门修补 Xillinux-2.0 的 kernel 来添加的，并且不是主线 Linux kernels 的一部分。

6

Linux笔记

6.1 一般的

本节包含特定于 Xillinux 的 Linux 相关主题。

可以在这些文档中找到有关 Xillybus 和 Linux 的更广泛的视图：

- Getting started with Xillybus on a Linux host
- Xillybus host application programming guide for Linux

6.2 Linux kernel 的 Compilation

由于它的大小，完整的 Linux kernel 不包含在 Xillinux 发行版中，但可以从 Github 下载

```
$ git clone https://github.com/xillybus/xillinux-kernel.git
```

有关与 Xillinux-2.0 一起使用的 kernel 的精确复制，请查看 “xillinux-2.0a” 标签。使用以下形式通知 kernel 构建环境所需的 cross compiler

```
$ export CROSS_COMPILE=/path/to/crosscompiler/arm-xilinx-linux-gnueabi-
```

然后获取Xillinux-2.0自带的kernel的compilation的.config文件：

```
$ make ARCH=arm xillinux_defconfig
```

然后运行 kernel 的 compilation：

```
$ make ARCH=arm -j 8 uImage modules LOADADDR=0x8000
```

为了构建 Device Tree Blobs，可以使用以下命令，在如上所述设置 cross compiler 和 .config 文件后：

```
$ make ARCH=arm dtbs
[ ... ]
  DTC     arch/arm/boot/dts/xillinux-microzed.dtb
  DTC     arch/arm/boot/dts/xillinux-zedboard.dtb
  DTC     arch/arm/boot/dts/xillinux-zybo.dtb
  DTC     arch/arm/boot/dts/xillinux-zturn-lite.dtb
```

DTB 文件可以在 arch/arm/boot/dts/ 中找到，正如命令响应所建议的那样。

6.3 Compilation kernel modules

Xillinux 发行版随附正在运行的 kernel 的 compilation headers。这还不足以执行 kernel 本身的 compilation，但允许 kernel modules 的 compilation 直接在平台上。

树外 kernel module 的 compilation 的标准方法是使用 Makefile，它在设置 environment variable 后调用 kernel 自己的构建环境，告诉它执行特定模块的 compilation。

由单个源文件 example.c 组成的 kernel module 的原生compilation（由 Zynq processor 本身执行）的最小 Makefile 如下：

```
ifneq ($(KERNELRELEASE),)
obj-m := example.o
else
TARGET := $(shell uname -r)
PWD := $(shell pwd)
KDIR := /lib/modules/$(TARGET)/build

default:
        @echo $(TARGET) > module.target
        $(MAKE) -C $(KDIR) SUBDIRS=$(PWD) modules
endif
```

请注意，模块名称 “example” 仅在 “obj-m” 行中提及。这是 Makefile 与另一个 Makefile 唯一不同的地方。

使用此 Makefile 通常会导致 compilation 会话如下（在板上运行；这不是 cross compilation）：

```
# make
make -C /lib/modules/4.4.30-xillinux-2.0/build SUBDIRS=/root/example modules
make[1]: Entering directory '/usr/src/linux-headers-4.4.30-xillinux-2.0'
  CC [M]  /root/example/example.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /root/example/example.mod.o
  LD [M]  /root/example/example.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.4.30-xillinux-2.0'
```

6.4 声音支持

6.4.1 一般的

Zedboard 和 Zybo 板卡分别凭借 Analog Devices 的 ADAU1761 和 SSM2603 芯片组支持录音和播放。这些仅连接到 Zynq 设备的 logic fabric (PL) 引脚。

除了其明显的功能外，声音支持包还演示了如何使用 Xillybus IP core 传输数据，以及通过 SMBus/I^2C 对芯片进行编程。

Xillinux 通过将专用 Xillybus streams 与当今 Linux 中最常见的声音工具包 Pulseaudio 连接，来原生支持声音。因此，几乎所有需要声卡的应用程序都可以正确使用板卡的声音芯片作为系统的默认输入和输出。

也可以关闭 Pulseaudio daemon，直接使用 Xillybus streams (/dev/xillybus_audio)。这提供了一个更简单的编程接口，只需打开 device file，代价是失去其他应用程序的本机功能。

与 Linux 声卡的常用方法不同，声音接口没有专用的 kernel driver（例如 ALSA ），因为 Xillybus host driver 无论如何处理数据传输。这没有任何意义，即使对于期望与 /dev/dsp 一起工作的程序也没有意义，因为 Pulseaudio 具有使用 “padsp” 实用程序伪造此接口的能力。

6.4.2 使用详情

默认情况下，声音会播放到耳机插孔（黑色）。在 Zedboard 上，同样的输出也进入 Line Out（绿色）。对于录音，仅使用麦克风输入（粉红色），但可以更改，如下所述。

6.4.3 相关boot scripts

在 Zedboard 和 Zybo 上设置声音有两个相关的任务：对音频芯片进行编程和启动 Pulseaudio daemon。

对于第一项任务，在 /etc/systemd/system/ 上安装了两个 systemd unit files：

- xillinux_sound.service：启动时启动 /usr/local/bin/xillinux-sound。
- xillinux_sound.path：指示systemd等待/dev/xillybus_smb出现，出现时启动刚才提到的service。请注意，这不是基于 udev，而是基于 inotify，它正在监视 /dev。

/usr/local/bin/xillinux-sound 识别它在哪个板上运行，并在适用时运行 script /usr/local/bin/zybo_sound_setup.pl 或 /usr/local/bin/zedboard_sound_setup.pl。

当任何用户在系统上创建第一个登录会话时，Pulseaudio daemon 就会启动，这要借助驻留在 /etc/systemd/user/ 中的用户服务单元文件 xillinux_pulseaudio.service。

由于Xillinux在串口console和console屏幕上自动登录root用户，所以Pulseaudio在系统boot完成后很快启动。

不建议在多用户计算机上将 Pulseaudio 作为 root 运行，但由于 Xillinux 以 root 作为默认用户运行，因此这是问题最少的选项。

如果需要直接访问 /dev/xillybus_audio，则应禁用该服务，例如

```
# systemctl --global disable xillinux_pulseaudio
Removed symlink /etc/systemd/user/default.target.wants/
xillinux_pulseaudio.service.
```

zybo_sound_setup.pl和zedboard_sound_setup.pl是Perl scripts，设置了音频芯片的registers才能正常工作。它们相当简单，即使对于不熟悉 Perl 的程序员也是如此。 script 使用 /dev/xillybus_smbus device file 在芯片的 I^2C bus 上发起交易。

可以编辑 zedboard_sound_setup.pl 以实现音频芯片的不同设置。特别是，如果 Line In 输入是所需的录音源，则应将

```
write_i2c(0x400a, 0x0b, 0x08);
write_i2c(0x400c, 0x0b, 0x08);
```

行替换为

```
write_i2c(0x400a, 0x01, 0x05);
write_i2c(0x400c, 0x01, 0x05);
```

同理，zybo_sound_setup.pl也可以进行编辑，将

```
write_i2c(0x04, 0x14);
```

替换为

```
write_i2c(0x04, 0x10);
```

，即可使用Line In进行录制。

6.4.4 直接访问 /dev/xillybus_audio

/dev/xillybus_audio可以直接写入播放，也可以读取录音。 sample format 每个音频样本为 32 位，分为两个 little Endian 格式的 16 位 signed integers。最重要的词对应于左声道。

采样率固定为 48000 Hz。

如果将具有此采样率的 Windows WAV 文件直接写入 /dev/xillybus_audio （大约 1 ms 也将播放 header ），例如使用

```
# cat song.wav > /dev/xillybus_audio
```

如果响应是

```
-bash: /dev/xillybus_audio: Device or resource busy
```

，则很可能另一个 process 正在打开 device file 进行写入，可能是 Pulseaudio daemon。打开 device file 以供一个进程读取并由另一个进程写入是没有问题的。

6.4.5 Pulseaudio详细信息

Pulseaudio 通过几个专用的 Pulseaudio 模块 module-file-sink 和 module-file-source 与 /dev/xillybus_audio device file 交互。它们的来源可以在 /usr/src/xillinux/pulseaudio/ 的 Xillinux 的 file system 中找到。

这些模块是标准 Pulseaudio 模块的轻微修改，用于将 UNIX pipes 用作数据 sinks 和源（ module-pipe-sink 和 module-pipe-source ）。

借助 /etc/pulse/default.pa 中的以下两行，当 Pulseaudio 启动时，模块会自动加载：

```
load-module module-file-sink file=/dev/xillybus_audio rate=48000
load-module module-file-source file=/dev/xillybus_audio rate=48000
```

这些模块被自动选择为系统的声音接口，因为没有其他选择。

6.5 OLED 实用程序（仅限 Zedboard）

默认情况下，Xillinux 在 boot 期间启动活动计量实用程序，显示 CPU 的近似使用百分比和 SD flash disk 上 I/O 速率的指示。

CPU 百分比基于 /proc/stat，其中所有未空闲的时间都被视为已使用的 CPU 时间。

SDIO 流量的估计基于中断发送到相应 driver 的速率。没有关于充分利用该资源的已知数字。相反，根据先前的测量，该实用程序显示 100% 的中断率似乎是最大的。

为了在开发板的 OLED 上显示图形输出，bitmap 被发送到由 Digilent 的 driver 创建的 /dev/zed_oled。值得一提的是，这款 driver 使用 bit-banging 机制将 SPI 数据发送到 OLED 设备，在软件中切换 clock 和数据。因此，每秒几次以这种方式发送 512 字节的 CPU 消耗很小，但对整体系统性能的影响很小。

要更改此实用程序的参数，请编辑 /usr/local/bin/start_zedboard_oled，归结为以下命令：

```
/usr/local/bin/zedboard_oled /proc/irq/$irqnum/spurious 4 800
```

应用程序 zedboard_oled 采用三个参数：

- 用于监视 SDIO 相关中断的 /proc 文件。 $irqnum 是 mmc0 设备的 IRQ 编号。
- OLED 显示更新的速率，以每秒次数为单位。
- 什么被认为是 SDIO 的 100% 中断率，以每秒中断数为单位。目前的数字是通过反复试验找到的。

要防止在 boot 期间启动此实用程序：

```
# systemctl disable zedboard_oled.path
Removed symlink /etc/systemd/system/paths.target.wants/zedboard_oled.path.
```

6.6 其他注意事项

- 尽管自 kernel 3.12 以来 Linux GPIO driver 发生了变化，但在 Xillinux-2.0 中使用与 Xillinux-1.3 相同的 GPIO driver，保留 Xillinux-1.3 的行为和编号。

  为了使用新的 GPIO driver，将 device tree 条目更改为 compatible = “xlnx,ps7-gpio-1.00.a”，因此它显示为 “xlnx,zynq-gpio-1.0”。
- Quad SPI flash 由 Linux kernel 以一位宽的 bus 访问，即使 driver 支持四位接口。可以更改相关的 device tree 条目以启用四位接口。这不是允许相同 device tree 与 kernel 3.12 一起使用的默认设置（即 Xillinux-1.3 ）。

7

故障排除

7.1 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，请在 Xillybus 的论坛寻求帮助：

http://forum.xillybus.com

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，请详细说明这些更改。另请说明使用了哪个版本的 ISE/Vivado 工具。

如果 Vivado 在 implementation 上为 VHDL 发出此错误：

```
ERROR: [Place 30-58] IO placement is infeasible. Number of unplaced terminals (3) is greater than number of available sites (2).
The following Groups of I/O terminals have not sufficient capacity:
Bank: 35:
 The following table lists all user constrained IO terminals
 Please analyze any user constraints (PACKAGE_PIN, LOC, IOSTANDARD ) which may cause a feasible placement to be impossible.
 The following table uses the following notations:
 c1 - is IOStandard compatible with bank? 1 - compatible, 0 is not
 c2 - is IO VREF compatible with INTERNAL_VREF bank? 1 - compatible, 0 is not
 c3 - is IO with DriveStrength compatible with bank? 1 - compatible, 0 is not
+--------+------------+----+----+----+-------------------------------------------------------------+
| BankId | IOStandard | c1 | c2 | c3 | Terminal Name                                               |
+--------+------------+----+----+----+-------------------------------------------------------------+
|   35   |LVCMOS18    |  1 |  1 |  1 |   PS_CLK                                                    |
|   35   |LVCMOS18    |  1 |  1 |  1 |   PS_PORB                                                   |
|   35   |LVCMOS18    |  1 |  1 |  1 |   PS_SRSTB                                                  |
+--------+------------+----+----+----+-------------------------------------------------------------+
```

请按照 3.3 段中的说明编辑 xillydemo.vhd。

由相同问题导致的另一个可能的错误：

```
ERROR: [Drc 23-20] Rule violation (NSTD-1) Unspecified I/O Standard
        ...
        Problem ports: PS_CLK, PS_PORB, PS_SRSTB.
```

```
ERROR: [Drc 23-20] Rule violation (RTSTAT-1) Unrouted net ...
ERROR: [Drc 23-20] Rule violation (UCIO-1) Unconstrained Logical Port ...
```

7.2 USB键盘和鼠标的问题

几乎所有 USB 键盘和鼠标都符合兼容行为的标准规范，因此不太可能遇到无法识别的设备问题。检查是否出现问题的第一件事是：

- 仅限 Zedboard：您使用的是正确的 USB 插头吗？应该是标有“USB OTG”的那个，离电源开关远一些。
- 仅限 Zedboard：设备是否有 5V 电源？是否安装了 JP2 跳线？在 Zedboard 通电的情况下，连接光学 USB 鼠标，并验证 LED 是否继续工作。
- 如果使用 USB hub，请尝试仅将键盘或鼠标直接连接到板上。

一般系统日志文件 /var/log/syslog 中可能会提供有用的信息。有时使用 “less /var/log/syslog” 查看其内容会很有帮助。更好的是，输入 “tail -f /var/log/syslog” 会在新消息到达时将它们转储到 console。这尤其有用，因为 USB bus 上的事件始终会在此日志中记录，包括有关检测到的内容和事件处理方式的详细描述。

请注意，shell prompt 也可以通过 USB UART 访问，因此如果连接键盘失败，可以使用串行终端查看日志。

7.3 file system mount 的问题

经验表明，如果使用合适的(Micro)SD卡，并且在下电前正确关闭系统，(Micro)SD卡中的数据完全没有问题。

在没有 root file system 的 unmounting 的情况下关闭电路板不太可能导致 file system 本身出现永久性不一致，因为 ext4 file system 在下一个 mount 上使用 journal 进行自我修复。然而，操作系统的功能正在累积损坏，因为在断电时打开以进行写入的文件可能会留下虚假内容或完全删除。这适用于任何突然关闭的计算机。

如果 root file system 无法执行 mount（导致在 boot 期间执行 kernel panic）或仅将 mount 执行为 read-only，最可能的原因是低质量的 (Micro)SD 卡。这种存储正常运行一段时间是很典型的，之后随机错误消息开始出现。如果 /var/log/syslog 包含这样的消息，(Micro)SD 卡很可能是原因：

```
EXT4-fs (mmcblk0p2): warning: mounting fs with errors, running ec2fsck
   is recommended
```

为避免这些问题，请坚持使用 Sandisk 设备。

7.4 “startx” 失败（Graphical desktop 无法启动）

虽然没有直接关系，但当(Micro)SD卡不是Sandisk制造时，这个问题经常被报告。图形软件在启动时会从卡中读取大量数据，因此很可能是 (Micro)SD 卡产生读取错误的主要受害者。

显而易见的解决方案是使用 Sandisk (Micro)SD 卡。

7.5 X desktop屏保后黑屏

如果 /root 目录已被清除，或者如果新用户正在使用桌面，或者如果已更改省电设置，则桌面可能无法从屏幕保护模式中恢复，尝试恢复时会留下黑屏。

修复：在 XFCE desktop 上，转到 Power Manager，并进行以下设置：

- 将 Display > “Put to sleep after” 设置为 “Never”
- 将 Display > “Put to sleep after” 设置为 “Never”
- 将 Display > “Switch off after” 设置为 “Never”。
- 将 Security > “Automatically lock the session” 设置为 “Never”。

这个问题不应该出现在新的 Xillinux 发行版上，因为这些设置已经为 root 用户设置好了。