

(机器翻译成中文)

---

## Getting started with Xilinx for Cyclone V SoC (SoCKit)

---

Xillybus Ltd.

[www.xillybus.com](http://www.xillybus.com)

Version 2.1

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

如果可能，请参阅英文文档。

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 This document may also be slightly outdated in relation to the original.*

*If possible, please refer to the document in English.*

<b>1</b>	<b>介绍</b>	<b>4</b>
1.1	Xilinx 分布	4
1.2	Xillybus IP core	5
<b>2</b>	<b>先决条件</b>	<b>7</b>
2.1	硬件	7
2.2	下载发行版	8
2.3	开发软件	8
2.4	使用 FPGA design 的经验	9
<b>3</b>	<b>建筑 Xilinx</b>	<b>10</b>
3.1	概述	10
3.2	解压缩 boot image 套件	11
3.3	生成 processor 的包装器	11
3.4	生成原始 bitstream 文件	12
3.5	用图像加载 microSD	14
3.5.1	一般的	14
3.5.2	加载图像 (Windows)	15
3.5.3	加载图像 (Linux)	16
3.6	将 soc_system.rbf 文件复制到 microSD 卡中	17
<b>4</b>	<b>执行 boot</b>	<b>19</b>
4.1	跳线和 DIP switch 设置	19
4.2	连接外围设备	19
4.3	为电路板供电	21
4.4	boot 期间 UART 输出	22
4.5	做第一台boot后不久	24
4.5.1	调整 file system 的大小	24
4.5.2	允许远程 SSH 访问	27
4.6	使用桌面	27
4.7	关机	28

---

4.8 从这里做什么 . . . . .	28
<b>5 进行修改</b>	<b>29</b>
5.1 与定制 logic 集成 . . . . .	29
5.2 使用其他板 . . . . .	30
5.3 自定义构建项目和 preflow.tcl . . . . .	31
5.4 Qsys 项目的变化 . . . . .	31
<b>6 故障排除</b>	<b>32</b>
6.1 端口“xillybus_0_conduit_...”不存在 . . . . .	32
6.2 USB键盘和鼠标的问题 . . . . .	32
6.3 File system . . . . .	33
6.4 “startx” . . . . .	33

# 1

## 介绍

---

### 重要的:

由于公众的兴趣相对较低，SoCKit 的 Xillinux 正在逐步淘汰：它开箱即用，没有已知问题，但它的实施仅限于用于构建 FPGA 捆绑包的相当过时的 Quartus II 13.0sp1。预计未来不会改变。对 Cyclone V SoC，尤其是 SoCKit 的总体支持是有限的。

### 1.1 Xillinux 分布

Xillinux 是一个完整的、图形化的、基于 Ubuntu 12.04 LTS 的 Linux 发行版，适用于 SoCKit 板，旨在作为快速开发混合软件/logic 项目的平台。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。

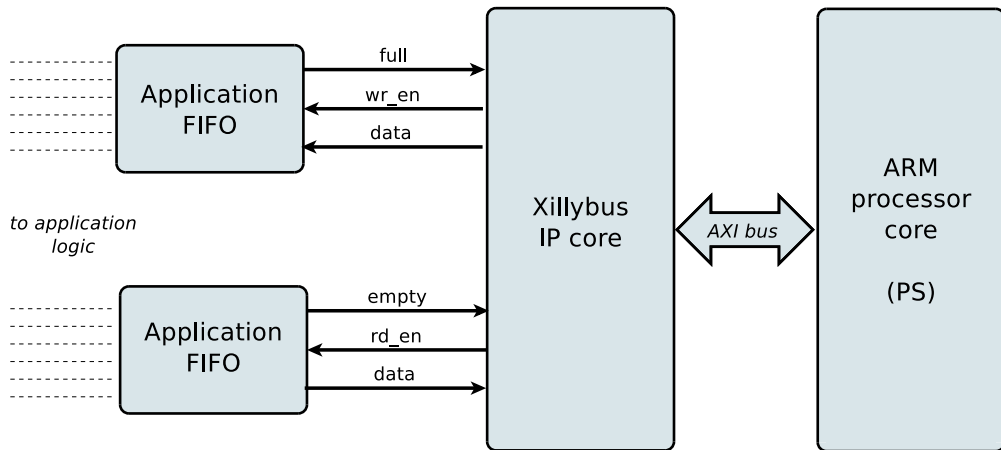
该发行版针对经典的键盘、鼠标和显示器设置进行组织。它还允许从 USB UART 端口进行命令行控制，但此功能主要用于解决问题。

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 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 <https://xillybus.com/custom-ip-factory> 上构建和下载您的自定义 IP core。

本指南解释了如何快速设置 Xilinx 发行版，包括 Xillybus IP core。这个 IP core 可以附加到用户提供的数据源和 data sinks，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。

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

Socket Linux 发行版 ( Xillinux ) 的 Xillybus 当前仅支持 SoCKit 板。

其他板的所有者可能会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。

还需要以下设备：

- 能够显示 VESA 兼容 1024x768 @ 60Hz 和模拟 VGA 输入的监视器（即几乎任何 PC 监视器）。
- 用于显示器的模拟 VGA 电缆
- USB 键盘
- 一只 USB 鼠标
- 如果键盘和鼠标未组合在单个 USB 插头中，则 Linux 3.8.0 可识别 USB 集线器
- USB 电缆到 SoCKit 板卡，A 型插座（母）到 USB Micro B 插头。购买 SoCKit 板时**不包括**此电缆。
- 具有 4 个或更多 GB 的可靠 microSD 卡，最好是由 Sandisk 制造的卡。  
不推荐使用其他品牌，因为据报道将它们与 Xillinux 一起使用会出现问题。
- microSD 卡和 PC 之间的 USB 适配器，用于将图像和 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。  
请注意，可以使用 SD 卡到 USB 适配器而不是 MicroSD 到 USB 适配器，与 SD 到 microSD 适配器一起使用，因为 SD 和 microSD 之间的区别只是物理外形尺寸。

建议使用无线键盘/鼠标组合，因为它无需 USB 集线器，并防止由于意外拉动 USB 电缆而对板上的 USB 端口造成物理损坏。

## 2.2 下载发行版

Xillinux 发行版可在 Xillybus 站点的下载页面下载：<https://xillybus.com/xillinux/>

该发行版由两部分组成：microSD 卡的 raw image 由 Linux 在启动时看到的 file system 组成，以及一组用于使用 Altera 工具实现以生成 first-stage boot image 的文件。有关这方面的更多信息是 3 部分。

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 <https://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特定应用的适用性。

## 2.3 开发软件

只有 Quartus II 13.0sp1 (Web 或 Subscription Edition) 可用于构建 Xillinux。尝试使用任何其他版本构建捆绑包将失败并出现错误。由于 ARM processor 是 Altera 设备的新产品，因此支持的软件工具变化很快，因此确保可靠结果的唯一方法是使用与 Xillinux 正确测试过的完全相同的版本。

由于 Cyclone V SoC 的 Xillinux 正在逐步淘汰，遗憾的是没有计划在未来取消此限制。

首选 Quartus II 的 64 位版本，特别是在 Windows 机器上，因为这些工具分配的 2GB 多于 RAM，这可能会在 32 位 Quartus II 上失败。如果使用 32 位 Windows XP，将 /3gb 标志添加到 boot.ini 可以使工具成功运行。参见 <https://msdn.microsoft.com/en-us/library/windows/hardware/gg487508.aspx>。

32 位 Windows 7 及更高版本可以使用 increaseuserva 参数进行修改，请参阅：

[https://msdn.microsoft.com/en-us/library/windows/hardware/ff542202\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/hardware/ff542202(v=vs.85).aspx)。

构建 Xillinux 不需要 Soc EDS 包。user space programs 和/或 kernel modules 的 compilation 可以直接在板子的 processor 上完成。

该软件可以直接从 Altera 的网站 (<https://www.altera.com>) 下载。

## 2.4 使用 FPGA design 的经验

使用 SoCKit 板时，无需以前的 FPGA design 经验即可在该平台上运行发行版。使用另一块板需要一些使用 Altera 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 AMD 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。

要从 microSD 卡执行 Xillinux 分发的 boot，它必须具有三个组件：

- 一个初始 boot image 环境，驻留在一个特殊的 partition 中，由 U-boot 加载程序组成
- 一个小型 FAT 文件系统，文件包含 FPGA 部分的配置 bitstream、Linux kernel 二进制文件及其 device tree。
- Linux 安装的 root file system。

所有这些，除了 FPGA 的 bitstream 之外，已经包含在 Xillinux 的 microSD 映像中。

本节将逐步详细介绍准备 microSD 的各种操作。大部分时间都花在准备 FPGA bitstream 上。

此过程假定使用 SoCKit 板。它由以下步骤组成，必须按以下列出的顺序完成：

- 解压缩 boot image 套件
- 生成 processor 的包装器和 bus 基础架构
- 实现主要的 FPGA 项目并将 bitstream 转换为正确的格式。
- 将原始 Xillinux 图像写入 microSD 卡

- 将 bitstream 文件复制到 microSD 卡中

5.2 段讨论了如何与其他板一起工作。

## 3.2 解压缩 boot image 套件

将之前下载的 xilinx-eval-socket-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。尤其是桌面不合适，因为它的路径包括 “*Documents and Settings*”。

该捆绑包包含以下目录:

- 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
- core——Xillybus IP cores 的预编译二进制文件
- soc\_system——ARM processor 包装器和 bus 基础设施
- instantiation templates——包含 Verilog 和 VHDL 中的 instantiation templates

请注意，'verilog' 和 'vhdl' 目录还包含 SoCKit 板的 QSF 文件。如果使用另一个板，则必须编辑此文件。

另请注意，vhdl 目录包含 Verilog 文件，但它们都不需要用户编辑。

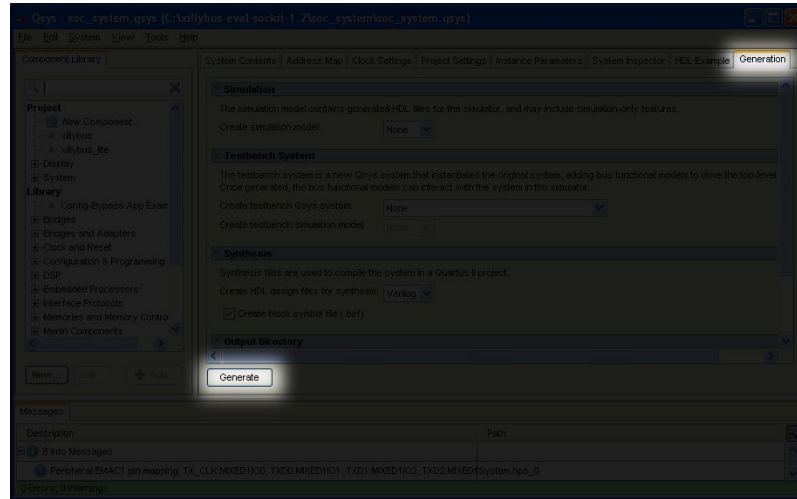
Xillybus IP core 的接口发生在相应 'src' 子目录中的 xillydemo.v 或 xillydemo.vhd 文件中。这是要编辑的文件，以便使用您自己的数据源和 sinks 尝试 Xillybus。

## 3.3 生成 processor 的包装器

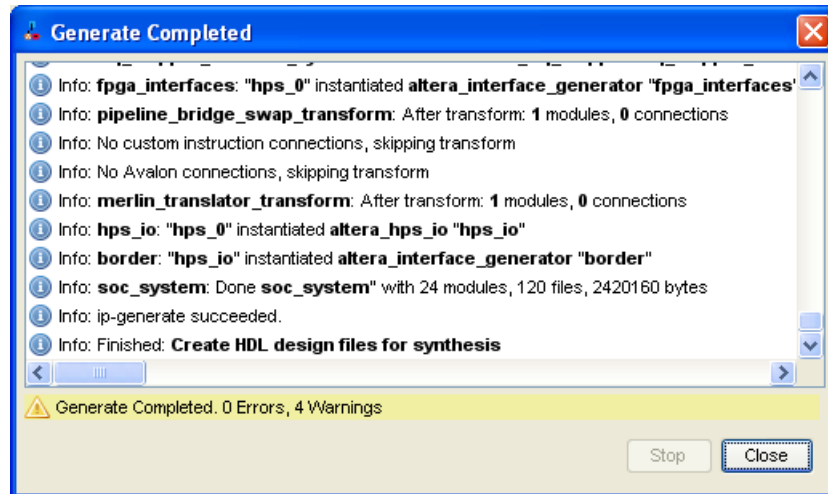
打开 Quartus II，可能通过双击 boot image 套件中 'verilog' 或 'vhdl' 子目录中的“xillydemo.qpf”文件。

在 Quartus 中，通过选择 “File > Open...” 打开 Qsys 项目并向上导航一个目录，进入 soc\_system 目录，然后选择 soc\_system.qsys。Qsys 工具启动并打开 processor 包装器项目。

选择窗口顶部附近的 “Generation” 选项卡，然后单击 “Generate”（靠近底部）



该过程需要几分钟左右，进度窗口如下所示：



关闭进度窗口以及 Qsys 窗口。以后无需重复此过程。

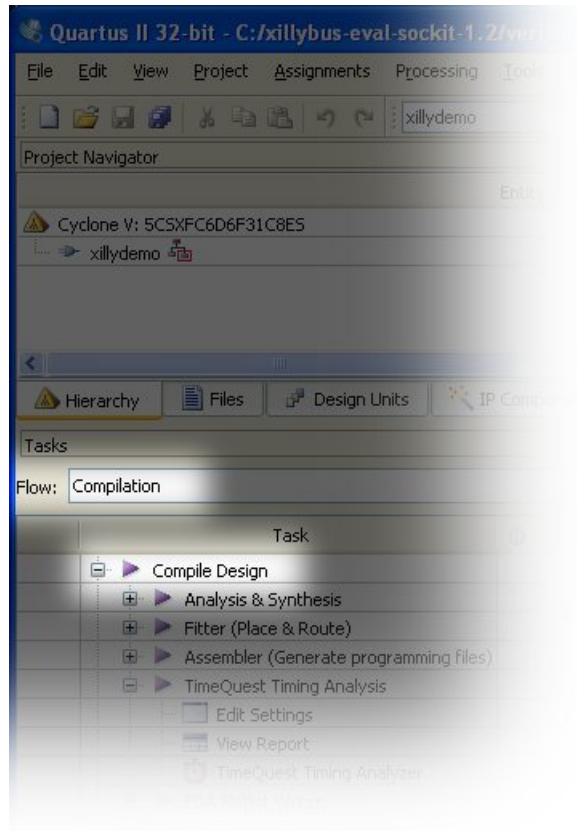
### 3.4 生成原始 bitstream 文件

此步骤在完成 processor 包装器的生成后执行，如 3.3 段所述。

根据您的偏好，双击 'verilog' 或 'vhd' 子目录中的 "xillydemo.qpf" 文件。Quartus II 将使用正确的设置启动并打开项目。如果您刚刚完成 Qsys，很可能您已经打开了

Quartus II。在这种情况下，只需验证是否选择了您的首选语言（如果您不关心，请选择 Verilog）。

确保将“flow”设置为“Compilation”，然后单击“Compile Design”以创建 FPGA 编程文件，如图所示。



该过程会产生大约 100 个警告，但应该以一个对话框结束，通知“Full Compilation was successful”。会产生严重警告，但不应容忍任何错误。此外，在 compilation 之后，始终必须验证没有警告说“Timing requirements not met” (332148)。稍后在您对 Verilog/VHDL 源进行更改后运行 design 的 compilation 时尤其如此。

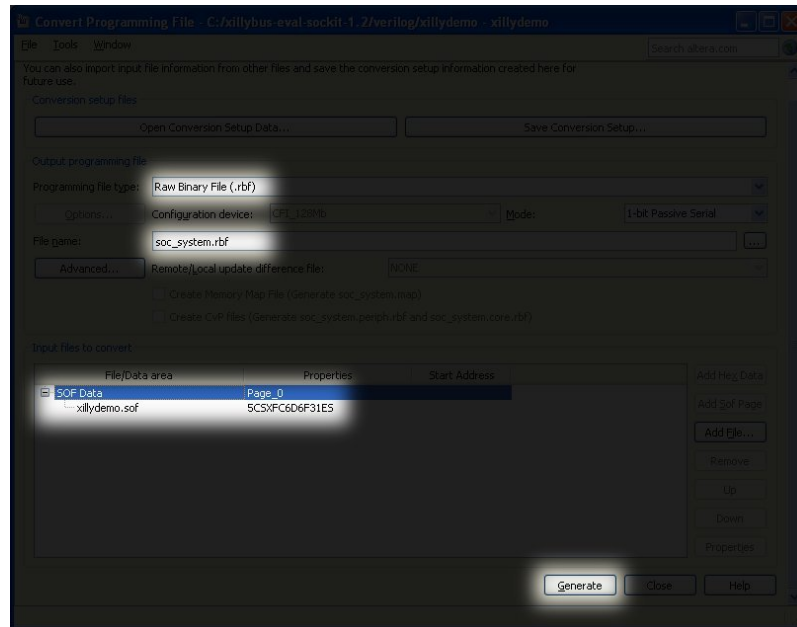
最后，将得到的编程文件转换成需要的格式。在 Quartus II 中，选择 File > Convert Programming Files... 并选择 Raw Binary File (.rbf) 作为编程文件类型。就在下面，将“File name”设置为 soc\_system.rbf。

在“Input files to convert”区域中，单击“SOF Data”，然后单击右侧的“Add File...”。选择 xillydemo.sof。

然后点击右下角的“Generate”。成功生成文件后关闭窗口。应将 `soc_system.rbf` 复制到 MicroSD 卡，如 3.6 段所述。

#### 重要的:

生成 `.rbf` 文件时不要启用压缩（保持默认）。`soc_system.rbf` 应该是 6-7 MBytes。



## 3.5 用图像加载 microSD

### 3.5.1 一般的

此任务的目的是将下载的 microSD 卡映像文件写入设备。该图像被下载为名为 `xillinux-1.1-sockit.img.gz`（或类似文件）的文件，并且是 microSD 卡的 gzip 压缩图像（但需要将 `soc_system.rbf` 文件添加到其中）。

此图像应未压缩，然后写入 microSD 卡的第一个 sector 及更高版本。有几种方法和工具可以做到这一点。接下来推荐几种方法。

该图像包含一个 `partition table`、一个用于放置 `boot image` 的部分填充的 `FAT file system`、一个 `raw boot partition` 和 `ext4` 类型的 `Linux root file system`。几乎所有的 Windows 电脑都只检测到 `FAT partitions`，所以 microSD 卡的容量会显得很小（47 MB 左右）。

写入完整的磁盘映像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。

**重要的:**

将图像写入 *microSD* 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，可能使用用于编写图像的工具。

### 3.5.2 加载图像 (Windows)

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

某些计算机（尤其是笔记本电脑）内置了 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。确保您位于左上角下拉菜单中的“[Device Mode](#)”（而不是“[Volume Mode](#)”）中。单击 [Restore](#) 并将文件类型设置为“[Compressed \(gzip\) image files](#)”。选择下载的图像文件（[xillinux-1.1-socket.img.gz](#)）。整个过程大约需要4-5分钟。完成后，卸载设备（“[safely remove hardware](#)”）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>usbimgcmd 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:\         | 2014 MB
```

(注意 "usbicmd" 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，我们就可以实际进行写入 ("restore")：

```
C:\usbimage>usbicmd r 2448 \path\to\xillinux-1.1-socket.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，将编号 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 microSD 卡图像在您的计算机上存储的路径。

### 3.5.3 加载图像 (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 microSD 卡非常重要。这最好通过插入 USB 连接器来完成，并寻找类似这样的主日志文件（/var/log/messages 或 /var/log/syslog）：

```
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-1.1-socket.img.gz
```

将图像复制到 microSD 卡很简单：

```
# dd if=xillinux-1.1-socket.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-1.1-socket.img /dev/sdc
cmp: EOF on xillinux-1.1-socket.img
```

注意响应：在图像文件上到达 EOF 的事实意味着其他所有内容比较正确，并且 flash drive 的空间比实际使用的空间多。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，生成了常规文件 `“/dev/sdc”`，而不是写入设备。

### 3.6 将 `soc_system.rbf` 文件复制到 microSD 卡中

拨下 USB 适配器，然后将其连接回计算机。这对于确保计算机是最新的 microSD 卡的 `partition table` 是必要的。如有必要，安装第一个 `partition`（例如 `/dev/sdb1`）。大多数计算机会自动执行此操作。

然后将 `soc_system.rbf`（在 3.4 段中生成）复制到 microSD 卡上的 FAT file system。在 Windows 系统上，这是系统将显示的唯一“disk”。在 Linux 系统上，它是第一个（和更小的）`partition`。无论哪种方式，正确的目的地都可以通过其现有内容轻松识别，即只有两个文件：“`socfpga.dtb`”和 `'ulimage'`。

完成后，正确卸载 microSD 卡，然后将其从计算机上拔下，例如

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

如果 SoCKit 板已经在运行 Xillinux，则可以从板本身更新 `soc_system.rbf`。要在 Xillinux 上安装 FAT 文件系统，请转到

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

并在 `/mnt/sd/` 访问 `file system`。请注意，如果新的 `soc_system.rbf` 本身有问题（例如它被压缩）或其他操作不当，板子下次可能无法执行 `boot`。因此，无论如何，访问 `microSD` 卡的替代方法必须放在手边。

# 4

## 执行 boot

### 4.1 跳线和 DIP switch 设置

要使主板能够从 microSD 卡执行 boot 功能,可能不需要对跳线进行任何更改,但最好确保它们与下图第一张图片中显示的设置相匹配。

只有跳线 J15-J19 与系统的 boot 相关。另外两个跳线与 LCD 背光和 HSMC 接口电压电平有关。

通常需要更换 DIP switches(位于电路板背面),如下图二所示。

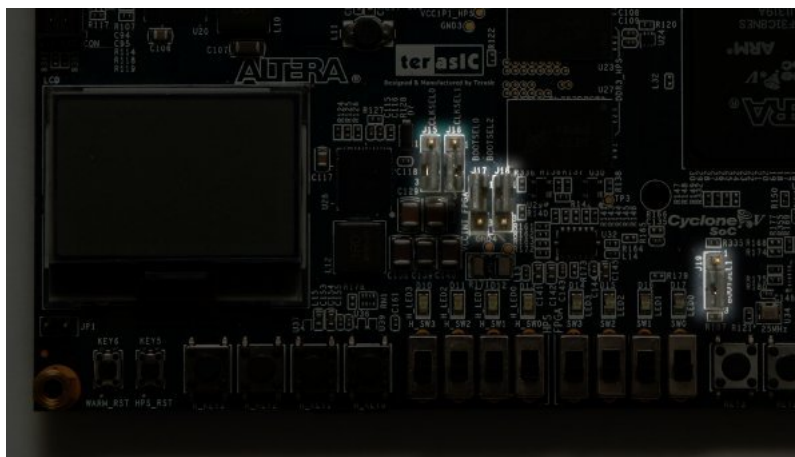
### 4.2 连接外围设备

#### 重要的:

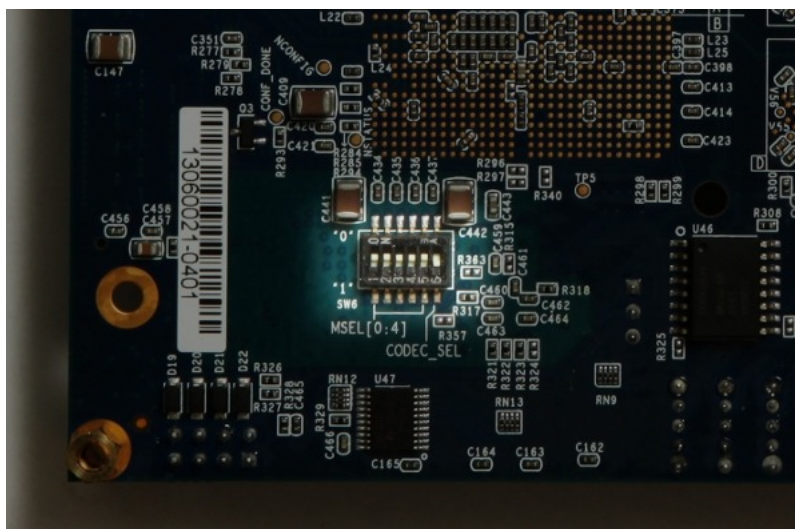
如果需要使用 USB 端口 (例如连接鼠标和键盘), 则在 Linux 执行 boot 时必须将某些外围设备连接到 OTG USB 端口 (即使是没有连接的 USB 集线器)。这是 Linux 确定 processor 和连接到它的对象之间的角色所必需的, 因为两者都可以是 OTG 端口上的 host。

应将以下通用硬件连接到板上:

- 连接到模拟 VGA 连接器的计算机显示器。由于 Xilinx 通过模拟 VGA 插头产生符合 VESA 的 1024x768 @ 60Hz 信号, 因此几乎可以肯定任何计算机显示器都足够了。
- 通过 USB 母头电缆 (不包含在 SocKit 硬件套件中) 连接到 USB OTG 连接器的鼠标和键盘。系统将在没有这些的情况下执行 boot, 并且在系统运行时连接和断开键盘和鼠标没有问题, 只要在 Linux 执行其 boot 序列时连接了一些外围设



电路板正面，突出显示跳线设置。



板的背面，DIP switch 突出显示。除最右边的开关外，所有开关均向上推。

备（或只是一个 USB 集线器）。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。

- 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 服务器，Linux 机器会自动配置网络。
- UART USB 端口可选择连接到 PC，但在大多数情况下是冗余的。boot 的一些消息在那里发送，当 boot 完成时，在此接口上发出 shell prompt。

当 PC 显示器或键盘丢失或无法正常工作时，这很有用。

请注意，如果此端口连接到计算机，则某些 terminal 软件必须在此计算机上运行。否则，boot 进程可能会在 Linux 等待其 boot 消息被读取时停止。在此端口未连接的情况下执行 boot 很好。

### 4.3 为电路板供电

本段描述了启动系统时会发生什么。在下面的描述中，查看板子时，红色电源按钮在左上角，8 个 LEDs 排在底部，如上一页的图像所示。

#### 重要的：

Xilinx 的 *root file system* 永久驻留在 *microSD* 卡上，并在系统启动时写入。因此，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，即使在突然断电时通常会观察到正确恢复。

将 microSD 卡插入 SoCKit 板，并按下红色按钮将其上电。预期的顺序如下：

- 以下 LEDs 立即开启：
  - 电源按钮旁边的绿色 LED
  - 板子底部全部 8 个 LEDs，带弱光
  - LCD 屏幕的背光 LED（除非已安装跳线来禁用此灯）

预计电源按钮右侧也会有一些短暂的闪光。这些是 UART LEDs。如果计算机连接到 UART USB 端口，它们将继续闪烁。

- 上电后不到一秒，最左边的四个 LEDs 就熄灭了，这表明 initial bootloader 已经对 processor 进行了初始化。如果这没有发生，可能的原因是 DIP 开关或跳线错误（请参阅 4.1 部分）或 microSD 未正确安装、出现故障或未正确加载 Xilinx image。

- 通电约 14 秒后，其他四个（最右边）LEDs 也熄灭，最右边的 LED 开始闪烁 1 个 Hz（大约）。一个“Xillybus”屏保在VGA屏幕上出现不到一秒，就换成了一个空白屏幕，左上角有两个Linux企鹅标志。如果这没有发生，请验证 `soc_system.rbf` 文件是否存在并且未压缩 (5-6 MBytes)。
- 开机约 26 秒后，VGA 屏幕上出现 `boot` 文字。
- `login prompt` 应在开机后 35 秒内出现。系统自动登录为 `root`，显示问候消息和 `shell prompt`。USB UART 链接上也展示了类似的 `shell prompt`，主要用于故障排除。

如果最左边的四个 LEDs 关闭后没有任何反应，则可能是 `soc_system.rbf` 文件有问题。查看 UART 的输出可能会有所帮助，如 4.4 段落中所述。

可以在以下位置查看显示电路板通电的简短视频剪辑

<https://youtu.be/mTDaAn4IX3I>。请注意，UART 的 LEDs 在剪辑中几乎没有活动，因为那里没有任何东西连接到 UART USB 端口。

在 `shell prompt` 处键入 “`startx`” 以启动 Gnome 图形桌面。桌面需要大约 15-30 秒来初始化。

笔记:

- `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。
- 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。

## 4.4 boot 期间 UART 输出

如果 `boot` 进程失败，UART 的输出可能会提示出现问题。计算机上的 `terminal` 应用程序应配置为 57600 baud, 8 bits, 1 stop bit, 没有 `parity` 和没有 `flow control` (“57600 8N1”)。

这是通常看到的:

```
U-Boot SPL 2012.10 (Dec 30 2013 - 18:03:34)
SDRAM: Initializing MMR registers
```

```
SDRAM: Calibrating PHY
SEQ.C: Preparing to start memory calibration
SEQ.C: CALIBRATION PASSED
DESIGNWARE SD/MMC: 0

U-Boot 2012.10 (Dec 30 2013 - 18:03:46)

CPU   : Altera SOCFPGA Platform
BOARD : Altera SOCFPGA Cyclone 5 Board
DRAM:  1 GiB
MMC:   DESIGNWARE SD/MMC: 0
*** Warning - bad CRC, using default environment

In:    serial
Out:   serial
Err:   serial
Net:   mii0
Warning: failed to set MAC address
```

最后一行倒计时几秒钟，然后自动继续读取三个文件。字节数以及显示的 **kernel** 版本可能会有所不同。

如果 **bootloader** 报告读取任何这些文件时出错，这可能是无法执行 **boot** 的原因。“**bad CRC**” 错误表明 **U-boot** 未能找到保存的一组环境变量，因此使用默认值，这很好。

```
reading uImage

3328400 bytes read
reading socfpga.dtb

15576 bytes read
reading soc_system.rbf

7007184 bytes read
## Booting kernel from Legacy Image at 00007fc0 ...
   Image Name:   Linux-3.8.0-xillinux-1.1
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    3328336 Bytes = 3.2 MiB
   Load Address: 00008000
   Entry Point:  00008000
```

```
## Flattened Device Tree blob at 00000100
   Booting using the fdt blob at 0x00000100
   XIP Kernel Image ... OK
OK
   Loading Device Tree to 0fff8000, end 0fffecd7 ... OK

Starting kernel ...
```

此时，**boot loader** 将控制权交给了 **Linux kernel**。下面仅给出 **kernel** 的 **boot** 消息的开头。**kernel** 完成 **boot** 序列后，给出一个 **shell prompt**。

如果“**Starting kernel...**”之后没有出现，请确认板上有蓝色 **LED** 闪烁，这表明 **FPGA** 部件已成功加载。最可能的原因是加载 **soc\_system.rbf** 文件失败。

```
Booting Linux on physical CPU 0x0
Initializing cgroup subsys cpuset
Linux version 3.8.0-00116-gffe44a8 (eli@ocho.localdomain) (gcc version 4.6.3 (S3
CPU: ARMv7 Processor [413fc090] revision 0 (ARMv7), cr=10c5387d
CPU: PIPT / VIPT nonaliasing data cache, VIPT aliasing instruction cache
Machine: Altera SOCFPGA, model: Altera SOCFPGA Cyclone V
Memory policy: ECC disabled, Data cache writealloc
PERCPU: Embedded 8 pages/cpu @80e77000 s10880 r8192 d13696 u32768
Built 1 zonelists in Zone order, mobility grouping on. Total pages: 260096
Kernel command line: console=ttyS0,57600 root=/dev/mmcblk0p3 rw rootwait
```

## 4.5 做第一台boot后不久

### 4.5.1 调整 file system 的大小

**root file system** 映像保持较小，以便尽可能快地将其写入设备。另一方面，没有理由不使用 **microSD** 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 **microSD** 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 **microSD** 卡初始化（写入映像和 **soc\_system.rbf**）

起点通常如下:

```
# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/root       2.3G  1.9G  304M  87% /
devtmpfs        505M  4.0K  505M   1% /dev
none            101M  736K  101M   1% /run
none            5.0M   0    5.0M   0% /run/lock
none            505M   0    505M   0% /run/shm
```

所以 root filesystem 是 2.3 GB，有 304 MB 免费。

第一阶段是重新分区 microSD 卡。在 shell prompt，输入：

```
# 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
- 3 [ENTER]—选择partition 3号
- n [ENTER]——创建一个新的partition
- 按 [ ENTER] 4 次接受默认值： primary partition，数字 3，从可能的最低 sector 开始，到可能最高的 sector 结束。
- w [ENTER]——保存退出。

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，microSD 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。

```
root@localhost:~# fdisk /dev/mmcblk0

Command (m for help): d
Partition number (1-4): 3

Command (m for help): n
Partition type:
   p   primary (2 primary, 0 extended, 2 free)
   e   extended
Select (default p):
Using default response p
Partition number (1-4, default 3):
```

```
Using default value 3
First sector (112455-15523839, default 112455):
Using default value 112455
Last sector, +sectors or +size{K,M,G} (112455-15523839, default 15523839):
Using default value 15523839
```

```
Command (m for help): w
The partition table has been altered!
```

```
Calling ioctl() to re-read partition table.
```

```
WARNING: Re-reading the partition table failed with error 16:
        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)
Syncing disks.
```

如果系统上显示的第一个 **sector** 的默认值与上面的不同，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，从 **fdisk** 的默认值转移可能有意义的唯一位置是最后一个 **sector**，以便使 **file system** 小于可能的最大值。

正如底部的警告所说，Linux 对 **partition table** 的看法无法更新，因为 **root partition** 正在使用中。因此需要重新启动：

```
# shutdown -h now
```

在 **UART console** 上出现 “**System halted.**” 的信息后（或光标在 **VGA** 屏幕上停止闪烁）后，关闭并重新打开电路板。系统应该像以前一样执行 **boot**，但是如果在重新分区期间执行不正确的操作，**boot** 可能会在任何阶段失败。

**file system** 尚未调整尺寸；它只被赋予了调整大小的空间。所以在 **shell prompt**，键入：

```
# resize2fs /dev/mmcb1k0p3
```

预计会收到以下响应：

```
resize2fs 1.42 (29-Nov-2011)
Filesystem at /dev/mmcb1k0p3 is mounted on /; on-line resizing required
old_desc_blocks = 1, new_desc_blocks = 1
The filesystem on /dev/mmcb1k0p3 is now 1926423 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。只要电源没有在中途丢失，这是安全的。

结果立即生效：无需重启。

使用 8 GB microSD 卡的典型会话：

```
# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/root       7.1G  1.9G  5.0G  28% /
devtmpfs        505M  4.0K  505M   1% /dev
none            101M  736K  101M   1% /run
none            5.0M   0    5.0M   0% /run/lock
none            505M   0    505M   0% /run/shm
```

请注意，“`df -h`”实用程序给出的尺寸是 1 GiB =  $2^{30}$  bytes，它比  $10^9$  bytes 的 Gigabyte 大 7.3%。这就是为什么一张 8 GB 卡在上面显示为 7.1 GiB。

#### 4.5.2 允许远程 SSH 访问

要在板上安装 `ssh` 服务器，请将板连接到 Internet 并在 shell prompt 处键入

```
# apt-get install ssh-server
```

。请注意，`root` 密码默认为无，`ssh` 理所当然地拒绝在没有密码的情况下登录某人。

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

```
# passwd
```

安装好 SSH 服务器后，建议在 `/etc/ssh/sshd_config` 底部添加如下一行：

```
UseDNS no
```

这将关闭 SSH 服务器所做的相当无意义的反向 DNS 检查，这会在尝试启动会话时导致延迟。

## 4.6 使用桌面

Xillinux 桌面就像任何 Ubuntu 桌面一样。由于 microSD 卡的数据带宽相对较低，应用程序加载速度会有些慢，但桌面本身的响应速度相当快。

要在桌面环境中运行应用程序，请单击桌面左上角的 Ubuntu 图标 (“Dash home”) 并输入所需应用程序的名称，例如 “terminal” 用于 shell prompt terminal 窗口，或 “edit” 用于 gedit 文本编辑器。

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

## 4.7 关机

要关闭系统，请选择桌面右上角的图标，然后单击 “Shut Down...”。或者，在 shell prompt 处键入

```
# shutdown -h now
```

当 UART console 上出现一条说明 “System Halted” 的文本消息时，可以安全地关闭电路板电源。或者，当 cursor 在 VGA 屏幕上的文本 console 上停止闪烁时，这也是 Linux 已关闭的标志。

## 4.8 从这里做什么

SoCKit 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 Xilinx 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。

注意 Xilinx 包括 gcc compiler 和 GNU make，所以 host 的应用可以直接在板子的 processors 上做 compiled。也可以使用 apt-get 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

**重要的:**

如果 *FPGA design* 是用 *QSF* 文件而不是 *demo bundle* 提供的文件构建的，请参阅 [5.3](#) 段落了解更多信息

Xillinux 发行版设置为易于与 *application logic* 集成。用于连接数据源和 *sinks* 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 image 套件中的所有其他 HDL 文件。

可以将带有自定义 *logic designs* 的附加 HDL 文件添加到第 3.4 段中处理的项目中，然后按照最初的不同方式重新构建。对于带有更新 *logic* 的系统的 boot，*soc\_system.rbf* 需要按照 3.4 段落中的说明重新生成，并按照 3.6 段落中的说明写入 microSD 卡。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。

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。

一个例外是在将存储器或 register 阵列连接到 Xillybus 时，应遵循 *xillydemo* 模块中所示的方法。

在 *xillydemo* 模块中，FIFOs 用于将来自 host 的数据环回给它。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 sink。

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 sink。

*xillydemo* 模块中使用的 FIFOs 两侧只接受一个通用 clock，因为两侧都是驱动 Xilly-

bus 的主 clock。在实际应用中，可能需要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 sinks 由除 bus clock 之外的 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 接口（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 SoCKit 板以外的板上运行 Xillinux 之前，可能需要进行某些修改。

其中包括更换引脚和 clocks：

- xillybus.v 中定义的 clock PLL 的属性必须设置为与连接到 clk\_bot1 的自由运行 clock 相匹配，如果它不是 50 MHz（并且相应地更新了 SDC 文件）。
- VGA 输出需要与预期的板相匹配。
- HPS' 多路复用管脚: ARM core 具有 I/O 管脚，这些管脚以固定位置路由到芯片上的物理管脚。ARM core 在 Qsys 中配置为将特定角色分配给这些引脚（例如 USB 接口、Ethernet 等），这些引脚必须与这些引脚在板上的接线相匹配。

后一个问题很重要，不仅因为关键的硬件功能可能会失败，而且电路板上的非法物理信号条件可能会导致硬件损坏（即使实际损坏非常罕见）。

HPS 引脚分配的不一致分三个步骤进行纠正：

- 在 Qsys 中更正这些分配。
- 构建整个项目（包括 FPGA 部分）并根据新的 handoff 文件生成更新的 boot loader（在 processor 上设置相关 registers 的是 U-boot 的 SPL 部分）。
- 更新 DTS 文件以匹配设备分配，并执行此文件的 compilation，生成 DTB 文件以执行系统的 boot。

### 5.3 自定义构建项目和 preflow.tcl

如果 **demo bundle** 的源被另一个 **Quartus II** 项目采用，则适用于 **QSF** 文件之间传输信息的常见做法。必须特别注意处理位于 **demo bundle** 的 **soc\_system** 子目录中的非标准 **script**、**preflow.tcl**。

**script** 的主要目的是重新连接由 **Qsys** 生成的 **SoC** 基础架构的 **toplevel module**，以绕过实现互连的错误 **AXI bus logic**。如果不重新布线，**AXI bus** 上的数据传输可能会正常工作，但会观察到零星的数据损坏。

**script** 在 **compilation** 发生之前由 **Quartus II** 自动执行。这是 **xillydemo.qsf** 中以下行的结果：

```
set_global_assignment -name PRE_FLOW_SCRIPT_FILE \  
    quartus_sh:../soc_system/preflow.tcl
```

为了确保基于 **Xillybus** 的 **FPGA design** 不会意外使用未重新布线的基础设施，**preflow.tcl** 还修改了一些 **top-level** 端口的名称，使其与原始端口不兼容。上述端口是那些名为 **xillybus\_0\_conduit\_\*** 的端口。

很容易判断模块是否被 **script** 修改过：修改文件的第一行，**soc\_system/soc\_system/synthesis/soc\_syst** 含

```
// soc_system.v (mangled by preflow.tcl)
```

如果文件确实被重新布线。

要将 **SoC** 基础架构包含在自定义项目中，请通过从已完全构建的 **demo bundle** 中复制文件来采用整个 **SoC** 模块树，并将以下行添加到 **QSF** 文件中

```
set_global_assignment -name QIP_FILE path/to/soc_system.qip
```

### 5.4 Qsys 项目的变化

由于 **Qsys** 生成的其中一个 **Verilog** 文件会自动修改，因此不建议修改 **Qsys** 工程。无法保证修改后的 **Qsys** 项目将被 **script** 正确修复，尤其是在项目拓扑发生更改的情况下。

更改元素的属性可能没问题，例如修改 **processor** 的引脚复用。但是请注意，如果对 **HPS processor** 的属性进行了更改，它们会通过预加载器中包含的 **C** 文件的更改生效，而不是通过 **logic**。

# 6

## 故障排除

### 6.1 端口“xillybus\_0\_conduit\_...”不存在

Xillydemo项目与Quartus II在compilation期间，可能会出现如下错误信息：

```
Error (12002): Port "xillybus_0_conduit_M_AXI_ARADDR" does not exist in macrofunction "u0" File: xillybus.v Line: 442
Error (12002): Port "xillybus_0_conduit_M_AXI_ARBURST" does not exist in macrofunction "u0" File: xillybus.v Line: 442
Error (12002): Port "xillybus_0_conduit_M_AXI_ARCACHE" does not exist in macrofunction "u0" File: xillybus.v Line: 442
Error (12002): Port "xillybus_0_conduit_M_AXI_ARID" does not exist in macrofunction "u0" File: xillybus.v Line: 442
Error (12002): Port "xillybus_0_conduit_M_AXI_ARLEN" does not exist in macrofunction "u0" File: xillybus.v Line: 442
Error (12002): Port "xillybus_0_conduit_M_AXI_ARLOCK" does not exist in macrofunction "u0" File: xillybus.v Line: 442
Error (12002): Port "xillybus_0_conduit_M_AXI_ARPROT" does not exist in macrofunction "u0" File: xillybus.v Line: 442
Error (12002): Port "xillybus_0_conduit_M_AXI_ARREADY" does not exist in macrofunction "u0" File: xillybus.v Line: 442
Error (12002): Port "xillybus_0_conduit_M_AXI_ARSIZE" does not exist in macrofunction "u0" File: xillybus.v Line: 442
...
```

找不到端口可能表明应该在 compilation 之前运行的 script 没有运行。这可能是由于 QSF 文件中的错误更改或自定义 Quartus II 项目中的源采用不当所致。有关详细信息，请参阅 5.3 段。

### 6.2 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：

- Linux 执行 boot 时设备是否连接？如果没有，请在连接鼠标和/或键盘的情况下重新启动 Linux。
- 您使用的是正确的 USB 插头吗？它应该是中间标有“HPS USB”的那个。
- 如果使用 USB hub，请尝试仅将键盘或鼠标直接连接到连接到 SoCKit 板的 OTG 端口的 USB 电缆。

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 `serial terminal` 查看日志。有关如何设置 `UART` 链接的信息，请参阅 `SoCKit` 板的文档。

### 6.3 File system 挂载问题

经验表明，如果使用合适的 `microSD` 卡，并且在下电前正确关闭系统，则永久存储完全没有问题。

在不卸载 `root file system` 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` 会在下一个 `mount` 上使用 `journal` 自行修复。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` 无法挂载（导致在 `boot` 期间出现 `kernel panic`）或执行 `mount` 只读，最可能的原因是低质量的 `microSD` 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。

### 6.4 “startx” 失败（图形桌面无法启动）

虽然没有直接关系，但当 `microSD` 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 `microSD` 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 microSD` 卡。