PetaLinux 工具文档

参考指南

UG1144 (v2019.1) 2019 年 5 月 22 日

查看所有版本

XILINX®

修订历史

下表列出了本文档的修订历史。

章节	修订总结
2019 年 5 月 22 日 2019.1 版	
第 6 章: 升级工作空间	新增“`petalinux-upgrade` 命令”一节。
第 12 章: 技术 FAQ	新增“封装管理”一节。
第 10 章: 高级设置	更新“Zynq® UltraScale+™ MPSoC 和 Zynq-7000 器件 FPGA 管理器配置和使用”。

目录

XILINX

第 1 章

简介

引言

PetaLinux 是一种嵌入式 Linux 软件开发套件 (SDK)，主要用于赛灵思 FPGA 基片上系统设计。本指南可帮助读者熟悉实现 PetaLinux 全面用途的工具。

我们假定您具有 Linux 基本知识，比如了解如何运行 Linux 命令。您应该知晓操作系统和主机系统功能，比如操作系统版本、Linux 分布、安全权限以及基本 Yocto 概念。

PetaLinux 工具包含：

- Yocto 可扩展 SDK (eSDK)
- 最少量的下载
- XSCT 和工具链
- PetaLinux CLI 工具

注释: 赛灵思软件开发套件 (SDK) (XSDK) 是集成设计环境 (IDE)，用于在赛灵思微处理器上创建嵌入式应用程序。

PetaLinux SDK 是一种赛灵思开发工具，含有构建、开发、测试和部署嵌入式 Linux 系统所需的所有功能。

Yocto 可扩展 SDK (eSDK)

下表详细说明了已安装的四个可扩展 SDK。

表 1: **可扩展 SDK**

路径	架构
`$PETALINUX/components/yocto/source/aarch64`	Zynq® UltraScale+™ MPSoC
`$PETALINUX/components/yocto/source/arm`	Zynq-7000 器件
`$PETALINUX/components/yocto/source/microblaze_full`	MicroBlaze™ 平台完整设计
`$PETALINUX/components/yocto/source/microblaze_lite`	MicroBlaze 平台简化设计

最少量的下载

BitBake 在查找上游任何源文件之前要校验 PREMIRRORS。当您拥有未使用 DL_DIR 变量定义的共享目录时，PREMIRRORS 即合适。工具的所有工程都使用这些 PREMIRRORS 并从中获取源代码。

工具中的 PREMIRROR 指向：`$PETALINUX/components/yocto/downloads`。下载目录拥有 Linux 内核、U-Boot 以及其他小型实用工具的源代码打包工具。如需了解更多信息，请参阅 下载镜像。

XSCT 和工具链

对于所有内嵌式软件应用程序，PetaLinux 工具都使用下面的 XSCT。所有三个架构的 Linux 工具链都来自 Yocto。

PetaLinux 命令行接口 (CLI) 工具

这包含您需要的所有 PetaLinux 命令。

XILINX

第 2 章

建立环境

安装要求

PetaLinux 工具安装要求如下：

- 工作站最低要求：
  - 8 GB RAM（推荐的赛灵思工具的最低要求）
  - 2 GHz CPU 时钟或同等频率（最低 8 核）
  - 100 GB 未使用 HDD 空间
  - 支持的操作系统：
    - Red Hat Enterprise 工作站/服务器 7.4, 7.5, 7.6（64 位）
    - CentOS 7.4、7.5、7.6 (64 位)
    - Ubuntu Linux 16.04.5、18.04.1 (64 位)
- 您需要拥有根访问权才能安装下表所示的所需软件包。PetaLinux 工具需要以非根用户的身份安装。
- PetaLinux 要求在您的 Linux 主机工作站中安装很多标准开发工具和库。在 Linux 主机上安装下表所列的库和工具。下文列出的所有 Linux 工作站环境都拥有 PetaLinux 工具所需的 32 位库。如果还有任何其他工具链软件包需要主机上有 32 位库，在发布 `petalinux-build` 之前要安装相同的库和工具。下表列出了所需的软件包，并介绍了在不同 Linux 工作站环境中的安装方法。
- PetaLinux 工具要求您的主机系统 `/bin/sh` 为“bash”。如果您在使用 Ubuntu 分布，而您的 `/bin/sh` 为“dash”，请咨询系统管理员，更改默认的系统 shell `/bin/sh`，请使用 `sudo dpkg-reconfigure dash` 命令更改。

表 2: **封装和 Linux 工作站环境**

工具/库	CentOS 7.4、7.5、7.6 (64 位)	Red Hat Enterprise 工作站/服务器 7.4、7.5、7.6 (64 位)	Ubuntu Linux 16.04.5、18.04.1 (64 位)
dos2unix	dos2unix-6.0.3-4.el7.x86_64.rpm	dos2unix-6.0.3-4.el7.x86_64.rpm	tofrodos_1.7.13+ds-2.debian.tar.xz
ip	iproute-3.10.0-74.el7.x86_64.rpm	iproute-3.10.0-74.el7.x86_64.rpm	iproute2 4.3.0-1ubuntu3
gawk	gawk-4.0.2-4.el7.x86_64.rpm	gawk-4.0.2-4.el7.x86_64.rpm	gawk (1:4.1.3+dfsg-0.1)
gcc	gcc-4.8.5-11.el7.x86_64	gcc-4.8.5-11.el7.x86_64	-
g++ (gcc-c++)	gcc-c++-4.8.5-11.el7.x86_64	gcc-c++-4.8.5-11.el7.x86_64	-
make	make 3.81	make 3.82	make 3.81
netstat	net-tools 2.0	net-tools 2.0	net-tools

表 2: **封装和 Linux 工作站环境** (续)

工具/库	CentOS 7.4、7.5、7.6 (64 位)	Red Hat Enterprise 工作站/服务器 7.4、7.5、7.6 (64 位)	Ubuntu Linux 16.04.5、18.04.1 (64 位)
ncurses devel	ncurses -devel 5.9-13	ncurses -devel 5.9-13	libncurses5 -dev
tftp server	tftp-server	tftp-server	tftpd
zlib devel（另，安装此版本的 32 位版本）	zlib-devel-1.2.7-17.el7.x86_64.rpm	zlib-devel-1.2.7-17.el7.x86_64.rpm	zlib1g:i386
openssl devel	openssl -devel 1.0	openssl -devel 1.0	libssl -dev
flex	flex 2.5.37	flex 2.5.37	flex
bison	bison-2.7	bison-2.7.4	bison
libselinux	libselinux 2.2.2	libselinux 2.2.2	libselinux1
gnupg	gnupg	gnupg	gnupg
wget	wget	wget	wget
diffstat	diffstat	diffstat	diffstat
chrpath	chrpath	chrpath	chrpath
socat	socat	socat	socat
xterm	xterm	xterm	xterm
autoconf	autoconf	autoconf	autoconf
libtool	libtool	libtool	libtool
tar	tar:1.24	tar:1.24	tar:1.24
unzip	unzip	unzip	unzip
texinfo	texinfo	texinfo	texinfo
zlib1g-dev	-	-	zlib1g-dev
gcc-multilib	-	-	gcc-multilib
build-essential	-	-	build-essential
SDL-devel	SDL-devel	SDL-devel	-
glibc-devel	glibc-devel	glibc-devel	-
32-bit glibc	glibc-2.17-157.el7_3.4.i686 glibc-2.17-157.el7_3.4.x86_64	glibc-2.17-157.el7_3.4.i686 glibc-2.17-157.el7_3.4.x86_64	-
glib2-devel	glib2-devel	glib2-devel	-
automake	automake	automake	-
screen	screen	screen	screen
pax	pax	pax	pax
gzip	gzip	gzip	gzip
libstdc++	libstdc++-4.8.5-11.el7.x86_64 libstdc++-4.8.5-11.el7.i686	libstdc++-4.8.5-11.el7.x86_64 libstdc++-4.8.5-11.el7.i686	-

快速安装软件包

以下几节将描述如何快速安装 Ubuntu 和 Redhat/CentOS 软件包。

Ubuntu

sudo apt-get install -y gcc git make net-tools libncurses5-dev tftpd zlib1g-dev libssl-dev flex bison libselinux1 gnupg wget diffstat chrpath socat xterm autoconf libtool tar unzip texinfo zlib1g-dev gcc-multilib build-essential -dev zlib1g:i386 screen pax gzip

Redhat/CentOS

sudo yum install gawk make wget tar bzip2 gzip python unzip perl patch diffutils diffstat git cpp gcc gcc-c++ glibc-devel texinfo chrpath socat perl-Data-Dumper perl-Text-ParseWords perl-Thread-Queue python34-pip xz which SDL-devel xterm autoconf libtool zlib-devel automake glib2-devel zlib ncurses-devel openssl-devel dos2unix flex bison glibc.i686 screen pax glibc-devel.i686 compat-libstdc+-33.i686 libstdc+.i686

注意! 如果不确定主机系统封装管理的正确流程，请咨询系统管理员。

重要提示! PetaLinux 2019.1 只适用于从 Vivado® Design Suite 2019.1. 导出的硬件设计。

安装步骤

要求

- 完成 PetaLinux 工具安装要求。如需了解更多信息，请参阅 安装要求。
- 下载 PetaLinux 发行包。可以从 PetaLinux 下载页面下载 PetaLinux 安装程序。
- Vivado® Design Suite、赛灵思 SDK 和 PetaLinux 版本同步。

运行 PetaLinux 工具安装程序

在不选择任何选项的情况下，PetaLinux 工具即被安装到当前工作目录中。此外，您还可以指定安装路径。

例如：若要将 PetaLinux 工具安装在 `/opt/pkg/petalinux/2019.1` 中：

```
$ mkdir -p /opt/pkg/petalinux/2019.1
$ ./petalinux-v2019.1-final-installer.run /opt/pkg/petalinux/2019.1
```

注释: 切勿将安装程序权限更改为 CHMOD 775，否则将产生 BitBake 错误。

这会将 PetaLinux 工具安装到 `/opt/pkg/petalinux/2019.1` 目录中。

重要提示! 一旦安装，您就无法移动或拷贝已安装的目录。在上述举例中，您无法移动或拷贝 `/opt/pkg/petalinux`，因为完整路径将被存储在 Yocto e-SDK 环境文件中。

注释: 您无法将工具作为根用户安装。确保 `/opt/pkg/petalinux` 可写入。您可以在安装之后更改权限，以便使其在全局可读取可执行 (0755)。将工具安装到 `/opt/pkg/petalinux` 目录不是强制要求。您可以在具有 755 权限的任何所需位置安装。

阅读并同意 PetaLinux 最终用户许可协议 (EULA) 是 PetaLinux 工具安装流程中的强制和不可分割的组成部分。您可以在安装之前阅读许可协议。如果您希望保存许可备查，可在以下文件中查找纯 ASCII 文本许可：

- `$PETALINUX/etc/license/petalinux_EULA.txt`: EULA 详细规定了 PetaLinux 拥有的权利和限制条件。
- `$PETALINUX/etc/license/Third_Party_Software_End_User_License_Agree ment.txt`: 该第三方协议详细说明了 PetaLinux 工具的可分发和不可分发组件。

默认设置是启用 WebTalk 选项，向赛灵思反馈工具使用统计数据。您可以运行 `petalinux-util --webtalk` 命令，关闭 Web Talk 功能：

重要提示! 在运行 PetaLinux 命令之前，您需要找到 PetaLinux 设置源。如需了解更多信息，请参阅 PetaLinux 工作环境建立。

```
$ petalinux-util --webtalk off
```

注释: 若需了解下载区中的共享状态，请参见 构建最优化。

故障排除

本节描述在安装 PetaLinux 工具时可能遇到的一些常见问题。如果 PetaLinux 工具安装失败，则会在 PetaLinux 安装目录中生成 `$PETALINUX/post-install.log` 文件。

表 3: **PetaLinux 安装故障排除**

描述/错误消息	描述和解决方案
`WARNING: You have less than 1 GB free space on the installation drive`	**问题描述** 此警告消息表明安装驱动几乎已满。安装之后，可能没有足够的空闲空间来开发硬件工程和/或软件工程。 **解决方案:** 清理安装驱动以腾出更多的空闲空间。 或者，将 PetaLinux 安装至另一个硬盘驱动。
`WARNING: No tftp server found`	**问题描述** 此警告消息表明工作站没有运行 TFTP 服务。如果没有 TFTP 服务，就不能使用 U-Boot 网络/ TFTP 功能将 Linux 系统镜像下载到目标系统。对于其他启动模式，可以忽略此警告。 **解决方案:** 工作站启用 TFTP 服务。如果不确定如何启用此服务，请与系统管理员联系。
`ERROR: GCC is not installed - unable to continue. Please install and retry`	**问题描述** 此警告消息表明主机工作站没有安装 gcc。 **解决方案:** 使用 Linux 工作站封装管理系统安装 gcc。如果不确定如何安装，请与系统管理员联系。请参阅 安装步骤。

`ERROR: You are missing the following system tools required by PetaLinux: missing-tools-list`
或
`ERROR: You are missing these development libraries required by PetaLinux: missing-library-list`
	问题描述
	此错误消息表明，“missing-tools-list”或“missing-library-list”中未列有所需的工具或库。
	解决方案:
	安装所缺少的工具包。如需了解更多信息，请参阅 安装要求。

表 3: **PetaLinux 安装故障排除** (续)

描述/错误消息	描述和解决方案
`./petalinux-v2019.1-final-installer.run: line 52: /proj/petalinux/petalinux-v2019.1_daily_latest/petalinux_installation_log: Permission denied`	**问题描述** 此错误消息表明 PetaLinux 安装目录没有写入权限。 **解决方案:** 授予安装目录 755 权限。

PetaLinux 工作环境建立

在安装之后，根据提供的 `settings` 脚本源，可自动完成剩余的建立。

要求

本节假设 PetaLinux 工具安装完成。如需了解更多信息，请参阅 安装步骤。

步骤建立 PetaLinux 工作环境

1. 获取适当的设置脚本：
   - Bash 作为用户登录 shell：

     ```
     $ source <path-to-installed-PetaLinux>/settings.sh
     ```

   - C shell 作为用户登录 shell：

     ```
     $ source <path-to-installed-PetaLinux>/settings.csh
     ```

     下面是第一次获取建立脚本时的一个输出例子：

     ```
     PetaLinux environment set to '/opt/pkg/petalinux'
     INFO: Checking free disk space
     INFO: Checking installed tools
     INFO: Checking installed development libraries
     INFO: Checking network and other services
     WARNING: No tftp server found - please see "PetaLinux SDK Installation
     Guide" for its
     impact and solution
     ```

2. 验证工作环境已经设置为：

   ```
   $ echo $PETALINUX
   ```

   输出：`/opt/pkg/petalinux`

   环境变量 `$PETALINUX` 应该指向已安装的 PetaLinux 路径。根据 PetaLinux 安装路径，输出可能与本例不同。

故障排除

本节介绍了在设定 PetaLinux 工作环境过程中您可能遇到的一些常见问题。

表 4: **PetaLinux 工作环境问题解答**

描述/错误消息	描述和解决方案
`WARNING: /bin/sh is not bash`	**问题描述** 该警告消息表示您的默认 shell 已连接到 dash。 **解决方案:** PetaLinux 工具要求您的主机系统 /bin/sh 为 bash。如果您在使用 Ubuntu 分布而且您的 /bin/sh 是 dash，请咨询您的系统管理员使用 `sudo dpkg-reconfigure dash` 命令来更改您的默认主机系统 /bin/sh。

`Failed to open PetaLinux lib`	**问题描述**
	该错误消息表示 PetaLinux 库加载失败。可能的原因如下:
	· PetaLinux `settings.sh` 未加载。
	· 正在运行的 Linux Kernel 已配置了 SELinux。这可造成安全性背景和加载库方面的问题。
	解决方案:
	1. 从顶层 PetaLinux 目录中找到 `settings.sh` 脚本源。如需了解更多信息，请参阅 PetaLinux 工作环境建立。
	2. 如果您已启用 SELinux，请确定 SELinux 是否处于强制模式。如果 SELinux 配置为强制模式，请重新配置 SELinux 至授权模式（参见 SELinux 手册）或更改库安全性环境以允许访问。
	`$ cd $PETALINUX/tools/xsct/lib/lnx64.o`
	`$ chcon -R -t textrel_shlib_t lib`

设计流程简介

一般而言，PetaLinux 工具遵从顺序工作流程模型。下表提供了一个示例设计工作流程，展示了任务应完成的顺序以及该任务的相应工具或工作流程。

表 5: **设计流程简介**

设计流程步骤	工具/工作流程
硬件平台创建（仅用于定制硬件）	Vivado® 设计工具
创建 PetaLinux 工程	`petalinux-create -t project`
初始化 PetaLinux 工程（仅用于定制硬件）	`petalinux-config --get-hw-description`
设置系统级选项	`petalinux-config`
创建用户组件	`petalinux-create -t COMPONENT`
设置 Linux 内核	`petalinux-config -c kernel`
配置根文件系统	`petalinux-config -c rootfs`
构建系统	`petalinux-build`
部署系统的封装	`petalinux-package`
启动系统进行测试	`petalinux-boot`

XILINX

第 3 章

创建工程

PetaLinux BSP 安装

PetaLinux 参考板级支持包 (BSP) 是受支持的电路板上的参考设计，以便您可以开始在自己的工程上工作并进行自定义。此外，这些设计可作为在受支持的电路板上创建自己的工程的基础使用。PetaLinux BSP 以可安装 BSP 文件的形式提供并包含所有必要的设计和配置文件、预建和测试硬件以及可随时下载到您的电路板或在 QEMU 系统仿真环境中启动的软件镜像。您可以将 BSP 下载到您选择的任何位置。

BSP 参考设计未包含在 PetaLinux 工具安装程序中，需要单独下载和安装。PetaLinux BSP 包可在 Xilinx.com 下载中心获得。在每个 BSP 中都有一个 README，介绍了 BSP 的详细内容。

注释: 只下载您需要的 BSP。

要求

本节假定已满足了以下要求：

- PetaLinux BSP 已下载。您可以从 PetaLinux 下载中下载 PetaLinux BSP。
- PetaLinux 工作环境建立已完成。有关更多详细信息，请参见 PetaLinux 工作环境建立。

从 BSP 中创建工程

1. 更改至您要在其中创建 PetaLinux 工程的目录。例如，如果您要在 `/home/user` 下创建工程：

   ```
   $ cd /home/user
   ```

2. 在命令控制台上运行 `petalinux-create` 命令：

   ```
   petalinux-create -t project -s <path-to-bsp>
   ```

   所引用的电路板基于已安装的 BSP。您将看到与下列输出类似的输出：

   ```
   INFO: Create project:
   INFO: Projects:
   INFO:   * xilinx-zcu102-v2019.1
   INFO: has been successfully installed to /home/user/
   INFO: New project successfully created in /home/user/
   ```

   在上述示例中，当命令运行时，它就告诉您从 BSP 中提取和安装工程。如果规定的位置在网络文件系统 (NFS) 上，它将 TMPDIR 更改为 `/tmp/<projname_timestamp>`；否则，它将被设置为 `$PROOT/build/tmp`。

   如果 `/tmp/<projname_timestamp>` 也在 NFS 上，则它会抛出一个错误。您可以通过 “petalinux-config → Yocto-settings” 随时更改 TMPDIR。切勿为两个不同的 PetaLinux 工程配置与 TMPDIR 相同的位置，否则可能造成构建错误。

如果您从 `/home/user` 运行 `ls`，您将看到已安装的工程。如需了解有关 PetaLinux 工程结构的详情，请参见 附录 B: PetaLinux 工程结构。

注意! 切勿在安装区创建 PetaLinux 工程，切勿将安装区作为临时构建区使用。

故障排除

本节描述在安装 PetaLinux BSP 时可能遇到的一些常见问题。

表 6: **PetaLinux BSP 安装故障排除**

描述/错误消息	描述和解决方案
`petalinux-create: command not found`	**问题描述**: 此消息表明无法找到 `petalinux-create` 命令，因此无法继续安装 BSP。 **解决方案**: 必须为 PetaLinux 工具建立环境。如需了解更多信息，请参阅 PetaLinux 工作环境建立。

使用 Vivado Design Suite 设置硬件平台

本节描述如何为 PetaLinux 工程设置硬件平台。

要求

本节假定已满足了以下要求：

- 已安装 Vivado® Design Suite。可以从 Vivado 设计工具下载页面下载 Vivado Design Suite。
- 已设置 Vivado 工具工作环境。如果还没有，请按照以下步骤获取适当的设置脚本：

```
$ source <path-to-installed-Xilinx-Vivado>/settings64.sh
```

- 熟悉 Vivado Design Suite 和赛灵思 SDK 工具。如需了解更多信息，请参阅 《Vivado Design Suite 用户指南：着手设计》 (UG910)。

设置 Linux 硬件平台

可以用 Vivado® 创建自己的硬件平台。无论如何创建和配置硬件平台，都需要更改少量的硬件 IP 和软件平台配置，以便硬件平台适用于 Linux。这些更改说明如下：

Zynq UltraScale+ MPSoC

以下列出 Zynq® UltraScale+™ MPSoC 硬件工程启动 Linux 的硬件要求：

1. 外部存储器至少有 64 MB 内存（必要）
2. 串行控制台 UART（必要）
3. 非易失性存储器（可选），如 QSPI 闪存和 SD/MMC

4. 以太网（可选，对网络访问必不可少）

重要提示! 如果使用带中断的软 IP 或带中断的外部 PHY 器件，确保中断信号已连接。

Zynq-7000 器件

以下列出 Zynq-7000 硬件工程启动 Linux 的硬件要求：

1. 一个三时序计数器 (TTC) （必要）

重要提示! 如果启用多个 TTC，Zynq-7000 Linux 内核将使用来自设备树的第一个 TTC 块。请确保 TTC 没有被使用。

2. 外部存储器至少有 32 MB 内存（必要）
3. 串行控制台 UART（必要）
4. 非易失性存储器（可选），如 QSPI 闪存和 SD/MMC
5. 以太网（可选，对网络访问必不可少）

重要提示! 如果使用软 IP，请确保中断信号已连接。如果使用带中断的软 IP 或带中断的外部 PHY 器件，确保中断信号已连接。

MicroBlaze 处理器 (AXI)

以下列出 MicroBlaze™ 硬件工程启动 Linux 的要求：

1. IP 核检查清单：
   - 外部存储器至少有 32 MB 内存（必要）
   - 带中断连接的双通道定时器（必要）
   - 串行控制台带中断连接的 UART（必要）
   - 非易失性存储器，如线性闪存或 SPI 闪存（必要）
   - 带中断连接的以太网（可选，但是网络访问的必要条件）
2. MicroBlaze 处理器设置：
   - 通过在 MicroBlaze 设置向导中选择带 MMU 的 Linux 或带 MMU 的低端 Linux 设置模板来支持带 MMU 的 MicroBlaze 处理器。

     注释: 不要禁用模板所启用的任何与指令集相关的选项，除非您理解此类变动的含义。
   - 系统从非易失性存储器启动时，MicroBlaze 处理器初始引导加载程序 fs-boot 的并行闪存至少需要 4 KB 的块 RAM，SPI 闪存至少需要 8 KB 的块 RAM。

将硬件平台导出到 PetaLinux 工程

本节介绍了如何向 PetaLinux 工程导出硬件平台。

注释: 器件支持存档 (DSA) 是一种硬件描述格式，将在 Vivado® Design Suite 2019.1 中推出。DSA 是一种超级 HDF 集，含有附加配置，可用 XSCT/XSDK 进行更改。

要求

本节假设用 Vivado Design Suite 创建硬件平台。如需了解更多信息，请参阅 使用 Vivado Design Suite 设置硬件平台。

导出硬件平台

在您配置了硬件工程之后，要构建硬件比特流。PetaLinux 工程要有硬件描述文件（`.hdf/.dsa` 文件），并包含有关处理器系统的信息。您可以从 Vivado® Design Suite 中运行导出硬件，获得硬件描述文件。

在工程初始化（或更新）过程中，PetaLinux 生成设备树源文件、U-Boot 配置报头文件，并根据硬件描述文件启用 Linux 内核驱动程序。这些详细信息可在 附录 B: PetaLinux 工程结构 中查阅。

对于 Zynq® UltraScale+™ MPSoC 平台，您需要利用平台管理单元 (PMU) 固件和 ATF 启动。有关构建 PMU 固件和 ATF 的信息，请参见 附录 C: 生成启动组件。如果您要为 Cortex™-R5F 启动构建第一阶段引导加载程序 (FSBL)，您还需要利用赛灵思 SDK 进行构建，因为利用 PetaLinux 工具构建的 FSBL 是用于 Cortex-A53 启动的。有关如何利用赛灵思 SDK 为 Cortex-R5F 构建 FSBL 的详细信息，请参见《Zynq UltraScale+ MPSoC：软件开发指南》 (UG1137)。

创建新的 PetaLinux 工程

本节介绍了如何创建新的 PetaLinux 工程。从模板中创建的工程在构建之前必须捆绑到实际硬件实例上。

要求

本节假定 PetaLinux 工作环境建立已完成。如需了解更多信息，请参阅 PetaLinux 工作环境建立。

创建新的工程

`petalinux-create` 命令用于创建新的 PetaLinux 工程：

```
$ petalinux-create --type project --template <PLATFORM> --name
<PROJECT_NAME>
```

参数如下：

- `--template <PLATFORM>` - 支持下列平台类型：
  - `zynqMP` (UltraScale+™ MPSoC)
  - `zynq` (Zynq-7000 器件)
  - `microblaze` (MicroBlaze™ CPU)

    注释: MicroBlaze 选项不得与 Zynq-7000 器件或可编程逻辑电路 (PL) 中的 Zynq UltraScale+ 设计配合使用。
- `--name <PROJECT_NAME>` - 您正在构建的工程名称。

该命令可从默认模板中创建新的 PetaLinux 工程文件夹。以下步骤可自定义这些设置，以便与以前创建的硬件工程匹配。

如果使用了 `--template` 选项，而未使用 BSP，您可以使用 `petalinux-config` 命令来选择与您的电路板设计接近的默认电路板配置，如下所示：

1. `petalinux-config --get-hw-description=<PATH-TO-HDF/DSA-DIRECTORY>`
2. 按需设置 CONFIG_SUBSYSTEM_MACHINE_NAME。
   - 可能的值：ac701-full、ac701-lite、kc705-full、kcu105、zc1275-revb、zcu1285-reva、zc1751-dc1、zc1751-dc2、zc702、zc706、avnet-ultra96-rev1、zcu100-revc、zcu102-rev1.0、zcu104-revc、zcu106-reva、zcu111-reva、zedboard、vcu118- rev2.0、sp701-rev1.0
   - 在 `petalinux-config` 中，选择“DTG Settings → (template) MACHINE_NAME”，将模板更改为任何上述可能的值。

提示: 如需了解有关 PetaLinux 工程结构的详细信息，请参阅 附录 B: PetaLinux 工程结构。

注意! 当在 NFS 上创建 PetaLinux 工程时，`petalinux-create` 自动将 TMPDIR 更改为 `/tmp/<projname_timestamp>`。如果 `/tmp` 也在 NFS 上，它会抛出一个错误。如果您要将 TMPDIR 更改至本地存储，请使用 “petalinux-config → Yocto-settings → TMPDIR”。切勿为两个不同的 PetaLinux 工程配置与 TMPDIR 相同的位置。这可能造成构建错误。如果 TMPDIR 位于 `/tmp/..`，删除工程不会将其清理干净。您必须显式执行此步骤或使用 `petalinux-build -x mrproper`。

XILINX

第 4 章

设置和构建

版本控制

本节详细介绍了有关 PetaLinux 工程中的版本管理/控制的内容。

要求

本节假定您已创建了一个新的 PetaLinux 工程或有现有的 PetaLinux 工程。如需了解更多创建 PetaLinux 工程的信息，请参见 创建新的 PetaLinux 工程。

版本控制

您可以对您的 PetaLinux 工程目录 `<plnx-proj-root>` 进行版本控制，但下列内容除外：

- `<plnx-proj-root>/.petalinux`
- `<plnx-proj-root>/!.petalinux/metadata`
- `<plnx-proj-root>/build/`
- `<plnx-proj-root>/images/linux`
- `<plnx-proj-root>/pre-built/linux`
- `<plnx-proj-root>/project-spec/meta-plnx-generated/`
- `<plnx-proj-root>/components/plnx-workspace/`
- `<plnx-proj-root>/*/*/config.old`
- `<plnx-proj-root>/*/*/rootfs_config.old`
- `<plnx-proj-root>/*.o`
- `<plnx-proj-root>/*.log`
- `<plnx-proj-root>/*.jou`

默认情况下，在创建工程的过程中，这些文件被添加到 `.gitignore` 中。

注释: 在提交源控制之前，应利用 `petalinux-build -x mrproper` 清理 PetaLinux 工程。

重要提示! 版本控制是目前在进程中的一项工作。建议您用 BSP 方法分享工程。

注释: 在并行开发中，在 `petalinux-config` 中的 TMPDIR 对每个用户都应该是唯一的。在将工程登入版本控制之前，使用 `${PROOT}` 作为参考来指定 TMPDIR 的相对路径。

导入硬件配置

本节介绍了利用新建硬件配置更新现有/新建 PetaLinux 工程的流程。这可让您使 PetaLinux 工具软件平台做好准备，以构建一个按照您的新硬件平台自定义的 Linux 系统。

要求

本节假定已满足了以下要求：

- 您已导出了硬件平台并生成了 `.hdf/.dsa` 文件。如需了解更多信息，请参阅 导出硬件平台。
- 您已创建了一个新的 PetaLinux 工程或有现有的 PetaLinux 工程。如需了解更多创建 PetaLinux 工程 的信息，请参阅 创建新的 PetaLinux 工程。

步骤导入硬件配置

导入硬件配置的步骤如下：

1. 更改至您的 PetaLinux 工程目录中。

```
$ cd <plnx-proj-root>
```

2. 利用 `petalinux-config` 命令导入硬件描述，按如下要求提供含有 `.hdf/.dsa` 文件的目录路径：

```
$ petalinux-config --get-hw-description=<path-to-directory-containing-
hardware description-file>
```

注释: 如果 DSA 和 HDF 文件都被放入硬件描述目录，则 DSA 文件所赋予的优先权要大于 HDF 文件。

当 `petalinux-config --get-hw-description` 为 PetaLinux 工程进行第一次运行或工具检测到系统主硬件候选项发生变化时，它会启动顶层系统配置菜单：

```
-*- ZYNQMP Configuration
Linux Components Selection --->
Auto Config Settings --->
 -*- Subsystem AUTO Hardware Settings --->
DTG Settings --->
ARM Trusted Firmware Compilation Configuration --->
PMU FIRMWARE Configuration --->
FPGA Manager --->
u-boot Configuration --->
Image Packaging Configuration --->
Firmware Version Configuration --->
Yocto Settings --->
```

确保选定了“Subsystem AUTO Hardware Settings”，然后进入菜单，与以下所示类似：

```
Subsystem AUTO Hardware Settings
System Processor (psu_cortexa53_0)  --->
Memory Settings  --->
Serial Settings  --->
Ethernet Settings  --->
Flash Settings  --->
SD/SDIO Settings  --->
RTC Settings  --->
[*]Advanced bootable images storage Settings  --->
```

“Subsystem AUTO Hardware Settings →”菜单可自定义整个系统范围内的硬件设置。

该步骤可能需要几分钟才能完成，因为工具将根据“自动配置设置 --->”和“子系统自动硬件设置 --->”设置，解析硬件描述文件，以获取更新设备树、PetaLinux U-Boot 配置文件以及内核配置文件所需的硬件信息。

例如，如果选定 `ps7_ethernet_0` 作为“Primary Ethernet”，而您启用了内核配置和 U-Boot 配置的自动更新，则该工具将自动启用其内核驱动并更新 U-Boot 头文件配置，以便 U-Boot 使用选定的以太网控制器。

注释: 如需了解有关自动配置设置菜单的详情，请参见 设置。

`--oldconfig/--silentconfig` 选项可允许您重复使用以前的配置。旧配置在含有无人值守更新的规定组件的目录内拥有文件名 `CONFIG.old`。

注释: `--oldconfig` 选项在未来版本中将被废弃。使用 `--silentconfig` 代替。

构建系统镜像

要求

本节假定您已使 PetaLinux 工具软件平台做好准备，以构建一个按照您的硬件平台自定义的 Linux 系统。如需了解更多信息，请参阅 导入硬件配置。

构建 PetaLinux 系统镜像的步骤

1. 更改至您的 PetaLinux 工程目录中。

   ```
   $ cd <plnx-proj-root>
   ```

2. 运行 `petalinux-build`，构建系统镜像：

   ```
   $ petalinux-build
   ```

   该步骤生成设备树 DTB 文件、第一阶段引导加载程序（如果选定的话）、U-Boot、Linux 内核以及根文件系统镜像。最后，它生成必需的启动镜像。

3. 编译进展在控制台上显示。等待编译完成。

提示: 详细的编译日志详见 `<plnx-proj-root>/build/build.log`。

在构建完成时，生成的镜像将位于 `<plnx-proj-root>/ images` 和 `/tftpboot` 目录之内。

控制台显示编译进展。例如：

```
[INFO] building project
[INFO] generating Kconfig for project
[INFO] silentconfig project
[INFO] sourcing bitbake
[INFO] generating plnxtool conf
[INFO] generating meta-plnx-generated layer
[INFO] generating bbappends for project . This may take time !
[INFO] generating u-boot configuration files
[INFO] generating kernel configuration files
[INFO] generating user layers
[INFO] generating kconfig for Rootfs
[INFO] silentconfig rootfs
[INFO] generating petalinux-user-image.bb
INFO: bitbake petalinux-user-image
Parsing recipes: 100% |
##################################################################################
########################################| Time: 0:00:29
Parsing of 2777 .bb files complete (0 cached, 2777 parsed). 3812 targets,
147 skipped, 0 masked, 0 errors.
NOTE: Resolving any missing task queue dependencies
Initialising tasks: 100% |
##################################################################################
########################################| Time: 0:00:05
Checking sstate mirror object availability: 100% |
##################################################################################
###############| Time: 0:00:10
Sstate summary: Wanted 923 Found 685 Missed 476 Current 0 (74% match, 0%
complete)
NOTE: Executing SetScene Tasks
NOTE: Executing RunQueue Tasks
NOTE: Tasks Summary: Attempted 3316 tasks of which 2254 didn't need to be
rerun and all succeeded.
INFO: Copying Images from deploy to images
INFO: Creating images/linux directory
NOTE: Failed to copy built images to tftp dir:  /tftpboot
[INFO] successfully built project
```

默认镜像

在您运行 `petalinux-build` 时，它为 Zynq®-7000 器件和 MicroBlaze™ 平台生成 FIT 镜像。还会生成 RAM 磁盘镜像 `rootfs.cpio.gz.u-boot`。

完整编译日志 `build.log` 存储在您的 PetaLinux 工程的构建子目录中。最终镜像 `<plnx-proj-root>/images/linux/image.ub`，是一种 FIT 镜像。内核镜像（包括 RootFS）是 Zynq® UltraScale+™ MPSoC 的“`Image`”、Zynq-7000 器件的“`zImage`”以及 MicroBlaze 处理器的“`image.elf`”。构建镜像位于 `<plnx-proj-root>/images/linux` 目录中。如果在 PetaLinux 工程的系统级配置中启用了该选项，其副本也被放置在 `/tftpboot` 目录中。

重要提示! 默认情况下，除了内核、RootFS 和 U-Boot 之外，PetaLinux 工程也被配置为生成和构建第一阶段引导加载程序。如需了解更多有关自动生成的第一阶段引导加载程序的详细信息，请参见 附录 C: 生成启动组件。

故障排除

本节介绍了在构建 PetaLinux 镜像过程中您可能遇到的一些常见问题/警告。

警告/错误:

```
<package-name> do_package: Could not copy license file /opt/pkg/petalinux/
components/yocto/source/<arch>/layers/core/meta/files/common-licenses/
to /opt/pkg/petalinux/build/tmp/work/<machine-name>-xilinx-linux/image/usr/
share/licenses/<package-name>/COPYING.MIT: [Errno 1] Operation not
permitted:
```

描述:

当安装了工具时，`/opt/pkg/petalinux/components/yocto/source/<arch>/layers/core/meta/files/common-licenses/` 中的所有许可证文件都将拥有“644”读写权限。因此，这些文件可由其他人读取，但不可写入。

解决方案:

- 方法 1：手动修改来自层级的许可证文件的权限

  ```
  $ chmod 666  /opt/pkg/petalinux/components/yocto/source/<arch>/layers/
  core/meta/files/common-licenses/*
  ```

  在创建硬链接时，用户将拥有对链接源的写入权限。

- 方法 2：禁用内核上的硬链接保护

  ```
  $ sysctl fs.protected_hardlinks=0
  ```

  在创建硬链接时，该内核将允许源不被当前用户写入。

- 方法 3：在 `<plnx-proj>/meta-user/conf/petalinuxbsp.conf` 中设置以下 Yocto 变量

  ```
  LICENSE_CREATE_PACKAGE_forcevariable = "0"
  SIGGEN_LOCKEDSIGS_TASKSIG_CHECK = "none"
  ```

  构建系统不会尝试创建链接，但许可证也不会在最终镜像上。

生成 uImage

如果想使用 uImage，请使用 `petalinux-package --image`。例如:

```
$ petalinux-package --image -c kernel --format uImage
```

注释: 该选项仅支持 Zynq-7000 器件和 MicroBlaze™ 处理器。

uImage 将生成到 PetaLinux 项目的子目录 `images/linux`。然后，需要设置 U-Boot，以使用 uImage 启动。如果已选择 `PetaLinux u-boot config` 作为 U-Boot 设置目标，则可以修改 PetaLinux 工程的 `<plnx-proj-root>/project-spec/meta-user/recipes-bsp/u-boot/files/platform-top.h` 以覆盖 `CONFIG_EXTRA_ENV_SETTINGS` 宏，从而定义 U-Boot 启动命令来使用 uImage 启动。

注释: 在未来版本中不推荐使用该选项，因为 `petalinux-build` 会生成 uImage。

生成 Zynq UltraScale+ MPSoC 的启动镜像

本节仅用于 Zynq® UltraScale+™ MPSoC，其中介绍了如何为 Zynq UltraScale+ MPSoC 生成 `BOOT.BIN`。

要求

本节假定您已经构建了 PetaLinux 系统镜像。如需了解更多信息，请参阅 构建系统镜像。

生成启动镜像

在执行此步骤之前，要确保您已构建了硬件比特流。启动镜像可放入闪存或 SD 卡。在您打开电路板的电源时，它可从启动镜像中启动。启动镜像通常含有第一阶段引导加载程序、FPGA 比特流（可选）、PMU 固件、ATF 和 U-Boot。

执行下列命令，生成 `.BIN` 格式的启动镜像。

```
petalinux-package --boot --format BIN --fsbl images/linux/zynqmp_fsbl.elf --
u-boot
images/linux/u-boot.elf --pmufw images/linux/pmufw.elf --fpga images/linux/
*.bit
--force
INFO: File in BOOT BIN:
"<plnx-proj-root>/images/linux/zynqmp_fsbl.elf"
INFO: File in BOOT BIN:
"/images/linux/pmufw.elf"
INFO: File in BOOT BIN:
"/images/linux/system.bit"
INFO: File in BOOT BIN:
"/images/linux/bl31.elf"
INFO: File in BOOT BIN:
"/images/linux/u-boot.elf"
INFO: Generating zynqmp binary package BOOT.BIN...
```

对于详细的用途，请参见 `--help` 选项或《PetaLinux 工具文档：PetaLinux 命令行参考》 (UG1157)。

生成 Zynq-7000 器件的启动镜像

本节仅用于 Zynq-7000 器件，描述如何生成 `BOOT.BIN`。

要求

本节假定您已经构建了 PetaLinux 系统镜像。如需了解更多信息，请参阅 构建系统镜像。

生成启动镜像

在执行此步骤之前，要确保您已构建了硬件比特流。启动镜像可放入闪存或 SD 卡。在您打开电路板的电源时，它可从启动镜像中启动。启动镜像通常包含第一阶段启动加载器镜像、FPGA 比特流（可选）和 U-Boot。

按照下面的步骤生成 .BIN 格式的启动镜像。

```
$ petalinux-package --boot --fsbl <FSBL image> --fpga <FPGA bitstream> --u-boot
```

有关详细用法，请参阅 --help 选项或《PetaLinux 工具文档：PetaLinux 命令行参考》 (UG1157)。

生成 MicroBlaze 处理器的启动镜像

本节仅用于 MicroBlaze™ 处理器，描述如何生成 MicroBlaze 处理器的 MCS 文件。

要求

本节假定您已经构建了 PetaLinux 系统镜像。如需了解更多信息，请参阅 构建系统镜像。

生成启动镜像

执行下列命令，为 MicroBlaze 处理器生成 MCS 启动文件。

```
$ petalinux-package --boot --fpga <FPGA bitstream> --u-boot --kernel
```

它可在您的工作目录中生成 boot.mcs，然后被拷贝到 <plnx-proj-root>/images/linux/ 目录中。利用上述命令，MCS 文件含有 FPGA 比特流、fs-boot、U-Boot 和内核镜像 image.ub。

生成只具有 fs-boot 和 FPGA 比特流的 MCS 文件的命令：

```
$ petalinux-package --boot --fpga <FPGA bitstream>
```

生成具有 FPGA 比特流、fs-boot 和 U-Boot 的 MCS 文件的命令：

```
$ petalinux-package --boot --fpga <FPGA bitstream> --u-boot
```

对于详细的用途，请参见 --help 选项或《PetaLinux 工具文档：PetaLinux 命令行参考》 (UG1157)。

生成 MicroBlaze 比特流文件

要求

本节假定您已经构建了 PetaLinux 系统镜像和 FSBL。如需了解更多信息，请参阅 构建系统镜像。

生成比特流

执行下列命令，为 MicroBlaze™ 处理器生成比特流文件。

```
$ petalinux-package --boot --fpga <FPGA bitstream> --fsbl <FSBL_ELF> --
format DOWNLOAD.BIT
```

这将在 `<plnx-proj-root>images/linux/` 目录中生成 `download.bit`。利用以上命令，它将 ELF 数据映射到设计中的块 RAM 的内存映射信息 (MMI)，从而将 fs-boot 合并到 FPGA 比特流中。对于详细的用途，请参见 `--help` 选项或参见《PetaLinux 工具文档：PetaLinux 命令行参考》 (UG1157)。

构建最优化

本节描述 PetaLinux 工具的构建最优化技术。

禁用默认组件

如果不需要，可以禁用默认组件。禁用 FSBL 和 PMU 固件方式：取消选择 “petalinux-config → Linux Components Selection”

- “FSBL → [ ] First Stage Boot Loader”
- “PMUFW → [ ] PMU Firmware”

取消选择这些组件将从默认构建流中删除这些组件。

注释: 如果 FSBL 和 PMU 固件不是用 PetaLinux 构建的，则必须在赛灵思 SDK 中构建。

本地镜像服务器

您可以在 NFS 或网络服务器上设置内部镜像，这样可以加快构建速度。默认情况下，PetaLinux 使用 sstate-cache 并从 petalinux.xilinx.com 下载镜像。通过以下步骤处理 PetaLinux 中 sstate 的本地、NFS 或内部网络服务器副本。可以从下载区域下载 sstate 和 PetaLinux。

表 7: **本地镜像服务器**

服务器	描述
downloads	下载文件源位于 http://petalinux.xilinx.com/sswreleases/rel-v${PETALINUX_VER}/downloads
aarch64	Zynq® UltraScale+™ MPSoC
arm	Zynq UltraScale+ MPSoC sstate 镜像
mb-full	MicroBlaze™ 处理器 sstate 镜像
mb-lite	MicroBlaze 设计 sstate 镜像

注意! 如要用视频编解码器构建 Zynq UltraScale+ MPSoC PetaLinux BSP，则必须通过 petalinux.xilinx.com 或本地访问该 sstate。

源镜像

可以通过 “petalinux-config → Yocto-settings → Add pre-mirror URL” 设置源镜像。`file:///<sstate path>/downloads` 用于所有工程。保存配置以使用下载镜像并验证 `build/conf/plnxtool.conf` 中的变动。例如：`file:///proj/petalinux/released/Petalinux-v${PETALINUX_VER}/sstate-rel-v${PETALINUX_VER}/downloads`。

减少构建时间

如需通过禁用网络 sstate feeds 来减少构建时间，请取消选择“petalinux-config → Yocto Settings → Enable Network sstate feeds”。

Sstate Feeds

可以通过 `petalinux-config` 设置 sstate feeds。

- NFS 上的 sstate feeds：转到 “petalinux-config → Yocto Settings → Local sstate feeds settings” 并输入 sstate 目录的完整路径。启用此选项后，即可指向 NFS /本地挂载点上可用的自有共享状态。
- 网络服务器上的 sstate feeds：转到 “petalinux-config → Yocto Settings → Enable Network sstate feeds → Network sstate feeds URL” 并输入 sstate feeds 的 URL。

注释: 默认情况下，这设置到 `http://petalinux.xilinx.com/sswreleases/rel-v${PETALINUX_VER}/aarch64/sstate-cache`。

构建忽略相依性

默认镜像配置启用了 initramfs。这可产生多重相依性，比如：

- 内核需要已为 initramfs 构建了 RootFS
- 构建 RootFS 的同时构建 FSBL、PMU 固件和 ATF，因为它们都是完整镜像的一部分
- 设备树需要内核报头
- U-Boot 需要设备树，因为它使用 Linux 设备树编译

您可以通过明确处理相依性来构建各个组件 (`petalinux-build -b component`)。应小心处理该选项，因为它可构建忽略其相依性的指定的配方/任务。如果用户未明确解决相依性问题，该选项的使用可导致多重间歇式错误。若要清理含有随机错误的工程，需使用 `petalinux-build -x mrproper`。

initramfs 模式

PetaLinux BSP 的默认模式是 initramfs 模式。该模式有多重相依性，比如：

- 内核需要已为 initramfs 构建了 RootFS
- 构建 RootFS 的同时构建 FSBL、PMU 固件和 ATF
- 设备树需要内核报头
- U-Boot 需要设备树，因为它使用 Linux 设备树编译

因此，构建设备树即构建所有组件。

例 1：只构建设备树

以下示例展示了从 PetaLinx 工程中生成设备树的步骤。设备树配方依赖 HDF、本机工具（dtc、python-yaml..）以及内核报头。

建立命令为：

1. 将 HDF 导入到工作空间：

   ```
   petalinux-config --get-hw-description=<PATH-to-HDF/DSA-DIRECTORY>
   ```

   以上命令只能从外部地点将硬件设计拷贝到 `petalinux project <proj-root>/project-spec/hw-description/`。外部 hdf 是 Yocto 中的一个配方，可将 HDF 从该位置导入到 Yocto 工作空间。所有 HDF 相依性配方都使用 Yocto 空间中的硬件设计。默认情况下，该相依性在配方内部处理。如果您在进行无相依性构建，则在硬件设计每次更新时，您都必须运行以下命令。

   ```
   petalinux-build -c external-hdf
   ```

2. 准备所有要求（本机实用程序）。

   该命令只需在首次时运行，只有在清理之后才需要重新运行

   ```
   petalinux-build -c device-tree -x do_prepare_recipe_sysroot
   ```

   注释: 在未来版本中，该功能被弃用。利用 `petalinux-build -c <app/package/component> -x <task>` 来构建组件的单个任务，因为 `petalinux-build` 命令的一部分将被弃用。

3. 利用以下命令构建忽略相依性的设备树任务：

   ```
   petalinux-build -b device-tree
   ```

   该命令可构建可忽略所有相依性的设备树并将其部署在 `images/linux/` 目录中。如果存在未满足的相依性，则会产生输出错误。以上命令也可用于增量构建。

注释: 以上各个命令需要利用 `-b` 选项运行。您可以在一次运行中获得所有上述功能：`petalinux-build -c device-tree`。它可自动取消所有相依性，从而导致构建更多相依性组件。

例 2：只构建 U-Boot

以下示例展示了如何构建忽略相依性的 U-Boot。`u-boot-xlnx` 配方依赖 HDF、设备树和本机工具（mkimage、dtc。）

1. 由于其强制性，您无法跳过设备树相依性。相反，您可以利用以上示例来构建设备树。
2. 为 U-Boot 配方建立本机工具。若要这么做，使用下列命令：

   ```
   petalinux-build -c u-boot-xlnx -x do_prepare_recipe_sysroot
   ```

   上述命令只需在首次运行或在每次清理后运行。

   注释: 在未来版本中，该功能被弃用。利用 `petalinux-build -c <app/package/component> -x <task>` 来构建组件的单个任务，因为 `petalinux-build` 命令的一部分将被弃用。

3. 构建忽略相依性的 U-Boot 任务。若要这么做，使用下列命令：

   ```
   petalinux-build -b u-boot-xlnx_2019.1
   ```

   上述命令可构建 `images/linux` 中的 U-Boot 和封装。

`-b` 选项需要配方的完整名称/路径；虚拟目标不起作用。

表 8: **配方的路径**

配方	路径
kernel，virtual/kernel	linux-xlnx_2019.1
u-boot，virtual/bootloader	u-boot-xlnx-2019.1
device-tree	device-tree

利用以下命令查找配方的路径：

```
petalinux-build -c "-e virtual/kernel" | grep "^FILE="
```

用任何虚拟目标或配方名称替换虚拟/内核。

注释: `petalinux-build -b` 需要用户明确满足所有要求。`petalinux-build -c` 自动处理所有相依性；不需要直接运行单个命令。

在未来版本要弃用的命令

- `petalinux-build -c rootfs`
- `petalinux-build -c <package_group>`
- `petalinux-build -x distclean(for image)`
- `petalinux-build -c component -x <task>`，其中任务是获取、解包、编译等。

XILINX

第 5 章

启动和封装

封装预建镜像

本节介绍了如何将新建镜像封装到预建目录中。

一般在您想要以 BSP 方式向其他用户分发您的工程时执行该步骤。

要求

本节假定已满足了以下要求：

- 对于 Zynq®-7000 器件，已生成启动镜像。如需了解更多信息，请参阅 生成 Zynq UltraScale+ MPSoC 的启动镜像。
- 对于 MicroBlaze™ CPU，已生成系统镜像。如需了解更多信息，请参阅 构建系统镜像。

封装预构建镜像的步骤

1. 切换到工程的根目录。

```
$ cd <plnx-proj-root>
```

2. 使用 `petalinux-package --prebuilt` 封装预构建镜像。

```
$ petalinux-package --prebuilt --fpga <FPGA bitstream>
```

有关详细用法，请参阅 `--help` 选项或《PetaLinux 工具文档：PetaLinux 命令行参考》 (UG1157)。

使用 petalinux-boot 命令处理预建镜像

您可以使用 `petalinux-boot` 命令启动 PetaLinux 镜像。在硬件电路板上使用 `--qemu` 选项和 `--jtag` 在软件仿真 (QEMU) 状态启动镜像。本节介绍了预建选项的不同启动级别。

要求

本节假设已封装预构建镜像。如需了解更多信息，请参阅 封装预建镜像。

预建选项的启动级别

`--prebuilt <BOOT_LEVEL>` 启动预建镜像（覆盖所有设置）。受支持的启动级别为 1 至 3。

- 1 级：下载预建 FPGA 比特流。
  - 它启动 Zynq® UltraScale+™ MPSoC 的 FSBL 和 PMU 固件。
  - 它启动 Zynq-7000 器件的 FSBL。
- 2 级：下载预建 FPGA 比特流并启动预建 U-Boot。
  - 对于 Zynq-7000 器件：它在启动 U-Boot 之前启动 FSBL。
  - 对于 Zynq UltraScale+ MPSoC：它在启动 U-Boot 之前启动 PMU 固件、FSBL 和 ATF。
- 3 级：
  - 对于 MicroBlaze™ 处理器：下载预建 FPGA 比特流并启动目标上的预建内核镜像。
  - 对于 Zynq-7000 器件：下载预建 FPGA 比特流和 FSBL、启动预建 U-Boot 并启动目标上的预建内核。
  - 对于 Zynq UltraScale+ MPSoC：下载 PMU 固件、预建 FSBL、预建内核、预建 FPGA 比特流、linux-boot.elf、DTB 以及目标上的预建 ATF。

展示预建选项启动级别用途的示例：

```
$ petalinux-boot --jtag --prebuilt 3
```

启动 QEMU 上的 PetaLinux 镜像

本节描述如何在软件仿真 (QEMU) 环境下启动 PetaLinux 镜像。

有关 QEMU 支持的赛灵思 IP 模型的详情，请参阅 附录 E: QEMU 支持的赛灵思 IP 模型。

要求

本节假定已满足了以下要求。

- 通过安装 PetaLinux BSP （请参阅 PetaLinux BSP 安装）或自行构建 PetaLinux 工程（请参阅 构建系统镜像）而拥有 PetaLinux 系统镜像。
- 如果打算使用 `--prebuilt` 选项进行 QEMU 启动，则需要将预先构建的镜像封装。如需了解更多信息，请参阅 封装预建镜像。

重要提示! 除非另有说明，PetaLinux 工具命令必须在工程目录 (`<plnx-proj-root>`) 中运行。

在 QEMU 上启动 PetaLinux 镜像的步骤

PetaLinux 提供 QEMU 支持，以便在不需要任何硬件的情况下在仿真环境中启用 PetaLinux 软件镜像的测试。

采用以下步骤利用 QEMU 测试 PetaLinux 参考设计：

1. 更改至您的工程目录并启动预建的 Linux 内核镜像：

```
$ petalinux-boot --qemu --prebuilt 3
```

如果您不希望使用预建能力进行 QEMU 启动，请参见 用于 QEMU 上引导的其他选项。

`--qemu` 选项可命令 `petalinux-boot` 启动 QEMU，而不是通过 JTAG 启动真实的硬件。`--prebuilt 3` 启动 Linux 内核，同时 PMUFW 在后台运行。

- `--prebuilt 1` 执行 1 级（FPGA 比特流）启动。该选项对 QEMU 无效。
- 2 级启动包括 U-Boot。
- 3 级启动包括预建 Linux 镜像。

如需了解更多有关预建选项的不同启动级别的信息，请参见 使用 petalinux-boot 命令处理预建镜像。

在成功运行 `petalinux-kernel` 期间在控制台上显示的内核启动日志消息的示例如下所示：

```
[    10.709243] Freeing unused kernel memory: 5568K (ffffffc000c20000 -
ffffffc001190000)
[    13.448003] udevd[1666]: starting version 3.2
[    13.458788] random: udevd: uninitialized urandom read (16 bytes read)
[    13.556064] udevd[1667]: starting eudev-3.2
[    14.045406] random: udevd: uninitialized urandom read (16 bytes read)
[    37.446360] random: dd: uninitialized urandom read (512 bytes read)
[    40.406936] IPv6: ADDRCONF(NETDEV_UP): eth0: link is not ready
[    41.460975] macb ff0e0000.ethernet eth0: link up (100/Full)
[    41.474152] IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready
[    44.787172] random: dropbearkey: uninitialized urandom read (32 bytes
read)

PetaLinux 2019.1 xilinx-zcu102-2019_1 /dev/ttyPS0

xilinx-zcu102-2019_1 login: root
Password:
root@xilinx-zcu102-2019_1:~#
root@xilinx-zcu102-2019_1:~#
```

2. 使用默认用户名 `root` 和密码 `root` 登录到 PetaLinux。

提示: 若要退出 QEMU，同时按下 “Ctrl+A”，然后按下 “X”。

用于 QEMU 上引导的其他选项

- 若要利用 QEMU 下载新构建的 `<plnx-proj-root>/images/linux/u-boot.elf`：

```
$ petalinux-boot --qemu --u-boot
```

  - 对于 Zynq® UltraScale+™ MPSoC，它加载 `<plnx-proj-root>/images/linux/u-boot.elf` 并利用 QEMU 启动 ATF 镜像 `<plnx-proj-root>/images/linux/bl31.elf`。然后 ATF 则启动已加载的 U-Boot 镜像。利用 `petalinux-build` 构建系统镜像。
  - 对于 MicroBlaze™ CPU 和 Zynq-7000 器件，它将利用 QEMU 启动 `<plnx-proj-root>/images/linux/u-boot.elf`。

- 若要利用 QEMU 下载新构建的内核：

```
$ petalinux-boot --qemu --kernel
```

- 对于 MicroBlaze 处理器，它利用 QEMU 启动 `<plnx-proj-root>/images/linux/image.elf`。
- 对于 Zynq-7000 器件，它将利用 QEMU 启动 `<plnx-proj-root>/images/linux/zImage`。
- 对于 Zynq UltraScale+ MPSoC，它利用 QEMU 加载内核镜像 `<plnx-proj-root>/images/linux/Image` 并启动 ATF 镜像 `<plnx-proj-root>/images/linux/bl31.elf`，然后 ATF 将启动已加载的内核镜像，同时 PMU 固件在后台运行。

注释: 对于 Zynq UltraScale+ MPSoC 内核引导，您需要建立 `pre-built/linux/images/` 文件夹并从任何 Zynq UltraScale+ MPSoC BSP 工程中拷贝 `pmu_rom_qemu_sha3.elf`。您还可以利用 `--pmu-qemu-args` 传递 `pmu_rom_qemu_sha3.elf`。

```
cd <project directory>
mkdir -p pre-built/linux/images
cp <zynq UltraScale+ bsp project
directory>/pre-built/linux/images/pmu_rom_qemu_sha3.elf pre-built/linux/
images/
```

或

```
petalinux-boot --qemu --uboot --pmu-qemu-args "-kernel
pmu_rom_qemu_sha3.elf"
```

在开机过程中，您会看到以登录提示结束的正常 Linux 启动进程，如下所示：

```
[ 10.709243] Freeing unused kernel memory: 5568K (ffffffc000c20000 -
ffffffc001190000)
[ 13.448003] udevd[1666]: starting version 3.2
[ 13.458788] random: udevd: uninitialized urandom read (16 bytes read)
[ 13.556064] udevd[1667]: starting eudev-3.2
[ 14.045406] random: udevd: uninitialized urandom read (16 bytes read)
[ 37.446360] random: dd: uninitialized urandom read (512 bytes read)
[ 40.406936] IPv6: ADDRCONF(NETDEV_UP): eth0: link is not ready
[ 41.460975] macb ff0e0000.ethernet eth0: link up (100/Full)
[ 41.474152] IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready
[ 44.787172] random: dropbearkey: uninitialized urandom read (32 bytes read)
PetaLinux 2019.1 xilinx-zcu102-2019_1 /dev/ttyPS0
xilinx-zcu102-2019_1 login: root
Password:
root@xilinx-zcu102-2019_1:~#
root@xilinx-zcu102-2019_1:~#
```

根据您测试的 Linux 镜像及其配置的不同，您看到的输出可能与上述所示略有不同。

当您看到仿真器控制台上出现登录提示时，请登录到虚拟系统中，登录名为 `root`，密码为 `root`。尝试使用 Linux 命令，比如 `ls`、`ifconfig`、`cat/proc/cpuinfo`，等等。其反应与在真实硬件上相同。在您完成后若要退出仿真器，请按下 “Ctrl + A”，松开后再按下 “X” 即可。

· 启动具体的 Linux 镜像：

`petalinux-boot` 工具利用镜像选项（`-i` 或 `--image`），也可启动具体的 Linux 镜像：

```
$ petalinux-boot --qemu --image <path-to-Linux-image-file>
```

例如：

```
$ petalinux-boot --qemu --image ./images/linux/zImage
```

- 利用具体 DTB 直接启动 Linux 镜像：

设备树（DTB 文件）用于描述 Linux 内核的硬件架构和地址映射。PetaLinux 系统仿真器还可利用 DTB 文件来动态配置仿真环境，以便与您的硬件平台匹配。

如果未提供 DTB 文件选项，`petalinux-boot` 从给定的 `image.elf` 中为 MicroBlaze 处理器、从 `<plnx-proj-root>/images/linux/system.dtb` 中为 Zynq-7000 器件和 Zynq UltraScale+ MPSoC 提取 DTB 文件。此外，您还可以按如下方式使用 `--dtb` 选项：

```
$ petalinux-boot --qemu --image ./images/linux/zImage --dtb ./images/linux/
system.dtb
```

注释: QEMU 版本已升级到 2.6。在新版本中旧选项已被弃用，但功能仍保持可用。PetaLinux 工具仍使用旧选项，因此会显示警告消息。您可以忽略这些消息。

Zynq UltraScale+ MPSoC：

```
qemu-system-aarch64: -tftp /home/user/xilinx-zcu102-2019.1/images/linux:
The -tftp option is deprecated. Please use '-netdev user,tftp=...' instead.g
```

利用 SD 卡在硬件上启动 PetaLinux 镜像

本节介绍了如何利用 SD 卡在硬件上启动 PetaLinux 镜像。

本节仅适用于 Zynq® UltraScale+™ MPSoC 和 Zynq-7000 器件，因为它们可让您从 SD 卡启动。

要求

本节假定已满足了以下要求：

- 已在 Linux 工作站上安装 PetaLinux 工具。如果尚未安装，请参阅 安装步骤。
- 已在 Linux 工作站上安装 PetaLinux BSP。如果尚未安装，请参阅 PetaLinux BSP 安装。
- 已安装了 minicom/kermit/gtkterm 等串行通讯程序；串行通讯程序的波特率已被设置为 115200 bps。

使用 SD 卡启动硬件上 PetaLinux 镜像的步骤

1. 将 SD 卡安装到主机上。
2. 将以下文件从 `<plnx-proj-root>/pre-built/linux/images/` 复制到 SD 卡中 FAT32 格式的第一分区根目录中：
   - `BOOT.BIN`
   - `image.ub`
3. 将电路板上的串行端口连接到您的工作站。
4. 打开工作站上的控制台，并启动首选串行通信程序（例如：kermit、minicom、gtkterm），该控制台的波特率设置为 115200。
5. 关闭电路板的电源。
6. 将电路板启动模式设定为 SD 启动。请参阅电路板文件，了解详细信息。

7. 将 SD 卡插入电路板。
8. 打开电路板的电源。
9. 查看串行控制台，将会看到如下启动信息：

```
[    5.546354] clk: Not disabling unused clocks
[    5.550616] ALSA device list:
[    5.553528]   #0: DisplayPort monitor
[    5.576326] sd 1:0:0:0: [sda] 312581808 512-byte logical blocks: (160
GB/149 GiB)
[    5.583894] sd 1:0:0:0: [sda] Write Protect is off
[    5.588699] sd 1:0:0:0: [sda] Write cache: enabled, read cache:
enabled, doesn't
support DPO or FUA
[    5.630942]  sda:
[    5.633210] sd 1:0:0:0: [sda] Attached SCSI disk
[    5.637897] Freeing unused kernel memory: 512K (ffffffc000c20000 -
ffffffc000ca0000)
INIT: version 2.88 booting
Starting udev
[    5.746538] udevd[1772]: starting version 3.2
[    5.754868] udevd[1773]: starting eudev-3.2
Populating dev cache
Starting internet superserver: inetd.
Running postinst /etc/rpm-postinsts/100-sysvinit-inittab...
Running postinst /etc/rpm-postinsts/libglib-2.0-0...
update-rc.d: /etc/init.d/run-postinsts exists during rc.d purge
(continuing)
INIT: Entering runlevel: 5
Configuring network interfaces... [    6.607236] IPv6:
ADDRCONF(NETDEV_UP): eth0:
link is not ready
udhcpc (v1.24.1) started
Sending discover...
[    7.628323] macb ff0e0000.ethernet eth0: link up (1000/Full)
[    7.633980] IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready
Sending discover...
Sending select for 10.10.70.1...
Lease of 10.10.70.1 obtained, lease time 600
/etc/udhcpc.d/50default: Adding DNS 172.19.128.1
/etc/udhcpc.d/50default: Adding DNS 172.19.129.1
Done.
Starting Dropbear SSH server: Generating key, this may take a while...
Public key portion is:
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQCxGtijKDWcJgnDxRCGiUPJJIMapFc0tcsCkMGyjJEDs
9LRugWzgaa
8XA+pGy4aTvZqHvGnFTvkMw4gZE/O
+BBgO8mMK9dFei2BvENbljm8M4NotG5LXRCFDaw6bXBCtg4ekCKWNU
6lUQU+PPdpmj9X+JgnTHnHnNB3jP6MrymCuS5wfFbyHfKdrwWXwfLmCycZr7DjRumee7T/
3SrBU3oRJoLcC
Vj2lf5Z7673+rOT1GM3QFzO2HWCCzyz/
3IUcEh9mhKpjzgs4iNEKmxwyi29rl37x7PD7zRsQbaW8uUtheCa
in3M1mjKfPnnygopdVh6IFsAT3FFMK4PYJ1GPL+h root@xilinx-zcu102-zu9-es2-
rev1_0-2019.1
Fingerprint: md5 f2:ce:1d:f2:50:e6:e2:55:5a:96:6f:bc:98:8f:82:99
dropbear.
Starting syslogd/klogd: done
```

```
Starting domain watchdog daemon: xenwatchdogd startup
PetaLinux 2019.1 xilinx-zcu102-zu9-es2-rev1_0-2019.1 /dev/ttyPS0
xilinx-zcu102-zu9-es2-rev1_0-2019.1 login: root
Password:
root@xilinx-zcu102-zu9-es2-rev1_0-2019:~#
```

提示: 如果希望停止自动启动，在控制台显示类似于下面的消息时，请按任意键：`Hit any key to stop autoboot:`

10. 在串行控制台上输入用户名 `root` 和密码 `root` 登录到 PetaLinux 系统。

故障排除

本节将描述使用 SD 卡启动硬件上的 PetaLinux 镜像时可能会遇到的一些常见问题。

表 9: **硬件上 PetaLinux 镜像故障排除**

描述/错误消息	描述和解决方案

`Wrong Image Format for boot command.`
`ERROR: Can't get kernel image!`
	问题描述
	此错误消息表明 U-Boot 引导加载程序无法找到内核镜像。这很可能是因为 `bootcmd` 环境变量设置不正确。
	解决方案:
	如要查看默认启动器件，请使用 U-Boot 控制台中的以下命令打印 `bootcmd` 环境变量。
	`U-Boot-PetaLinux> print bootcmd`
	如果没有使用 sdboot 流运行，有以下几个选项：
	· 无需重建 PetaLinux，设置 `bootcmd` 为从您期望的媒体启动，使用 `setenv` 命令。至于 SD 卡启动，请按照如下方式设置环境变量。
	`U-Boot-PetaLinux> setenv bootcmd 'run sdboot' ; saveenv`
	· 运行 `petalinux-config` 设置从 SD 卡加载内核镜像。如需了解更多信息，请参阅 启动镜像存储配置。重建 PetaLinux 并使用重建的 U-Boot 重新生成 `BOOT.BIN`，然后使用新的 `BOOT.BIN` 启动电路板。请参阅 生成 Zynq UltraScale + MPSoC 的启动镜像 了解如何生成 `BOOT.BIN`。

提示: 如要更多了解 U-Boot 选项，请使用命令：`$ U-Boot-PetaLinux> printenv`。

利用 JTAG 在硬件上启动 PetaLinux 镜像

本节介绍了如何利用 JTAG 在硬件上启动 PetaLinux 镜像。

JTAG 启动与 XSDB 通信，而 XSDB 与 hw_server 通信。所用的 TCP 端口是 3121；确保该端口的防火墙被禁用。

要求

本节假定已满足了以下要求：

· 通过安装 PetaLinux BSP （请参阅 PetaLinux BSP 安装）或自行构建 PetaLinux 工程（请参阅 构建系统镜像）而拥有 PetaLinux 系统镜像。

- 此为可选项，只有在希望使用 JTAG 启动的预构建功能时才需要。拥有已封装预构建镜像（请参阅 封装预建镜像）。
- 已安装了 minicom/kermit/gtkterm 等串行通讯程序；串行通讯程序的波特率已被设置为 115200 bps。
- 已安装了适当的 JTAG 电缆驱动程序。

JTAG 启动硬件上 PetaLinux 镜像的步骤

1. 关闭电路板的电源。
2. 将带有 JTAG 电缆的电路板上的 JTAG 端口连接到工作站。
3. 将电路板上的串行端口连接到您的工作站。
4. 如果系统有以太网，也要将电路板上的以太网端口连接到本地网络。
5. 对于 Zynq-7000 器件电路板，请确保模式开关设置为 JTAG 模式。请参阅电路板文件，了解详细信息。
6. 打开电路板的电源。
7. 打开工作站上的控制台，并启动首选串行通信程序（例如：kermit、minicom），该控制台的波特率设置为 115200。
8. 在工作站上运行 `petalinux-boot` 命令，如下所示：

   ```
   $ petalinux-boot --jtag --prebuilt 3
   ```

   注释: 如果不希望使用 JTAG 启动预构建功能，请参阅 使用 JTAG 启动的其他选项。

   `--jtag` 选项告诉 `petalinux-boot` 通过 JTAG 在硬件上启动，而 `--prebuilt 3` 选项则启动 Linux 内核。等待命令控制台出现 shell 提示符，表明命令完成。

   注释: 如需了解更多有关预建选项的不同启动级别的信息，请参见 使用 petalinux-boot 命令处理预建镜像。

   工作站命令控制台上显示 `petalinux-boot` 成功的消息示例如下所示：

   ```
   INIT: Entering runlevel: 5
   Configuring network interfaces... [    6.607236] IPv6:
   ADDRCONF(NETDEV_UP): eth0:
   link is not ready
   udhcpc (v1.24.1) started
   Sending discover...
   [    7.628323] macb ff0e0000.ethernet eth0: link up (1000/Full)
   [    7.633980] IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready
   Sending discover...
   Sending select for 10.10.70.1...
   Lease of 10.10.70.1 obtained, lease time 600
   /etc/udhcpc.d/50default: Adding DNS 172.19.128.1
   /etc/udhcpc.d/50default: Adding DNS 172.19.129.1
   Done.
   Starting Dropbear SSH server: Generating key, this may take a while...
   Public key portion is:
   ssh-rsa
   AAAAB3NzaC1yc2EAAAADAQABAAABAQCxGtijKDWcJgnDxRCGiUPJJIMapFc0tcsCkMGyjJEDs
   9LRugWzgaa
   8XA+pGy4aTvZqHvGnFTvkMw4gZE/O
   +BBgO8mMK9dFei2BvENbljm8M4NotG5LXRCFDaw6bXBCtg4ekCKWNU
   6lUQU+PPdpmj9X+JgnTHnHnNB3jP6MrymCuS5wfFbyHfKdrwWXwfLmCycZr7DjRumee7T/
   3SrBU3oRJoLcC
   Vj2lf5Z7673+rOT1GM3QFzO2HWCCzyz/
   3IUcEh9mhKpjzgs4iNEKmxwyi29rl37x7PD7zRsQbaW8uUtheCa
   in3M1mjKfPnnygopdVh6IFsAT3FFMK4PYJ1GPL+h root@xilinx-zcu102-zu9-es2-
   ```

```
rev1_0-2019.1

Fingerprint: md5 f2:ce:1d:f2:50:e6:e2:55:5a:96:6f:bc:98:8f:82:99
dropbear.
Starting syslogd/klogd: done
Starting domain watchdog daemon: xenwatchdogd startup
PetaLinux 2019.1 xilinx-zcu102-zu9-es2-rev1_0-2019.1 /dev/ttyPS0
xilinx-zcu102-zu9-es2-rev1_0-2019.1 login: root
Password:
root@xilinx-zcu102-zu9-es2-rev1_0-2019:~
```

默认情况下，使用 DHCP 设置 PetaLinux 参考设计的网络设置。所看到的输出可能与上面的示例略有不同，这取决于所测试的 PetaLinux 参考设计。

9. 在串行控制台上输入用户名 `root` 和密码 `root` 登录到 PetaLinux 系统。
10. 在系统控制台上运行 ifconfig 来确定 PetaLinux 的 IP 地址。

使用 JTAG 启动的其他选项

- 如要下载比特流到目标电路板：

```
$ petalinux-boot --jtag --fpga --bitstream <BITSTREAM>
```

- 如要下载新构建的 `<plnx-proj-root>/images/linux/u-boot.elf` 到目标电路板：

```
$ petalinux-boot --jtag --u-boot
```

- 如要下载新构建的内核到目标电路板：

```
$ petalinux-boot --jtag --kernel
```

  - 对于 MicroBlaze™ 处理器，这将启动目标电路板上的 `<plnx-proj-root>/images/linux/image.elf`。
  - 对于 Zynq® UltraScale+™ MPSoC，这将启动目标电路板上的 `<plnx-proj-root>/images/linux/Image`。
  - 对于 Zynq-7000 器件，这将启动目标电路板上的 `<plnx-proj-root>/images/linux/zImage`。

- 如要用 `--fpga --bitstream <BITSTREAM>` 选项下载带比特流的镜像：

```
$ petalinux-boot --jtag --u-boot --fpga --bitstream <BITSTREAM>
```

以上命令下载比特流，然后下载 U-Boot 镜像。

- 如要用 `-v` 选项查阅 JTAG 启动的详细输出：

```
$ petalinux-boot --jtag --u-boot -v
```

记录 JTAG 启动时的 Tcl/XSDB

使用以下命令记录 JTAG 启动期间使用的 XSDB 命令。它将 Tcl 脚本（然后调用 XSDB 命令）数据转储到 `test.txt`。

```
$ cd <plnx-proj-root>
$ petalinux-boot --jtag --prebuilt 3 --tcl test.txt
```

故障排除

本节描述 JTAG 启动硬件上的 PetaLinux 镜像时可能会遇到的一些常见问题。

表 10: **JTAG 启动硬件上 PetaLinux 镜像故障排除**

描述/错误消息	描述和解决方案
`ERROR: This tool requires ' xsdb' and it is missing. Please source Xilinx Tools settings first.`	**问题描述** 此错误消息表明 PetaLinux 工具无法找到作为赛灵思 Vivado 或 SDK 工具一部分的 xsdb 工具。 **解决方案:** 必须建立 Vivado 工具工作环境。如需了解更多信息，请参阅 PetaLinux 工作环境建立。
在试图启动硬件上的 U-Boot 或内核时看不到任何控制台输出，但在 QEMU 上启动正确。	**问题描述** 这个问题通常由以下一种或多种原因引起: · 串行通信终端应用的波特率设置错误。 · 硬件和软件平台之间不匹配。 **解决方案:** · 确保终端应用波特率正确，并与硬件配置相符。 · 确保使用正确的硬件平台构建 PetaLinux 工程。 ○ 正确导入硬件配置（请参阅 导入硬件配置）。 ○ 检查“Subsystem AUTO Hardware Settings →”子菜单，确保与硬件平台匹配。 ○ 检查“Subsystem AUTO Hardware Settings →”的“Serial settings →”子菜单下，确保 stdout, stdin 设定了正确的 UART IP 核。 ○ 重新构建系统镜像（请参阅 构建系统镜像）。

使用 TFTP 启动硬件上的 PetaLinux 镜像

本节描述如何使用普通文件传输协议 (TFTP) 启动 PetaLinux 镜像。

TFTP 启动可节省大量时间，因为比 JTAG 启动快很多，而且不必为内核源中的每次更改都刷新镜像。

要求

本节假定已满足了以下要求:

· 已建立 TFTP 服务器的主机环境，也构建了 TFTP 启动的 PetaLinux 镜像。如需了解更多信息，请参阅 设置 TFTP 启动。

· 您已封装了预建镜像。如需了解更多信息，请参阅 封装预建镜像。

· 已安装了 minicom/kermit/gtkterm 等串行通讯程序；串行通讯程序的波特率已被设置为 115200 bps。

· 在主机和 Linux 目标之间已正确建立了以太网连接。

· 已安装了适当的 JTAG 电缆驱动程序。

在具有 TFTP 的硬件上启动 PetaLinux 镜像

1. 关闭电路板的电源。
2. 使用 JTAG 电缆将电路板上的 JTAG 端口连接到工作站。
3. 将电路板上的串行端口连接到您的工作站。
4. 通过网络开关将电路板上的以太网端口连接到局域网。
5. 对于 Zynq®-7000 器件和 Zynq UltraScale+ MPSoC 器件电路板，要确保模式开关被设置为 JTAG 模式。请参阅电路板文件，了解详细信息。
6. 打开电路板的电源。
7. 打开您工作站上的控制台，启动首选串行通讯程序（例如，kermit、minicom），将该控制台上的波特率设定为 115200。
8. 在您的工作站上按如下所示运行 `petalinux-boot` 命令

   ```
   $ petalinux-boot --jtag --prebuilt 2
   ```

   `--jtag` 选项命令 `petalinux-boot` 通过 JTAG 在硬件上启动，同时 `--prebuilt 2` 选项下载预建的比特流（以及 Zynq-7000 器件的 FSBL）至目标电路板，然后在目标电路板上启动预建的 U-Boot。
9. 当自动启动开始时，可按下任意键停止。

   成功 U-Boot 下载的工作站控制台输出示例如下：

   ```
   U-Boot 2019.01-07106-gec1e403dd6 (Apr 29 2019 - 09:12:44 +0000)

   Board: Xilinx ZynqMP

   I2C:   ready
   DRAM:  4 GiB
   EL Level: EL2
   Chip ID: xczuunkn
   MMC:   Card did not respond to voltage select!
   sdhci@ff170000 - probe failed: -95
   Card did not respond to voltage select!

   zynqmp_qspi_ofdata_to_platdata: CLK 104156250
   SF: Detected n25q512a with page size 512 Bytes, erase size 128 KiB,
   total 128 MiB
   *** Warning - bad CRC, using default environment

   In:    serial
   Out:   serial
   Err:   serial
   Bootmode: JTAG_MODE
   Net:   ZYNQ GEM: ff0e0000, phyaddr c, interface rgmii-id

   Warning: ethernet@ff0e0000 MAC addresses don't match:
   Address in SROM is         ff:ff:ff:ff:ff:ff
   Address in environment is  00:0a:35:00:22:01
   eth0: ethernet@ff0e0000
   U-BOOT for xilinx-zcu102-2019_1

   BOOTP broadcast 1
   DHCP client bound to address 10.0.2.15 (2 ms)
   Hit any key to stop autoboot:  0
   ZynqMP>
   ```

10. 检查 TFTP 服务器 IP 地址是否设定为镜像所在的主机的 IP 地址。这可使用以下命令完成：

```
ZynqMP> print serverip
```

11. 使用以下命令将服务器 IP 地址设定为主机的 IP 地址：

```
ZynqMP> set serverip <HOST IP ADDRESS>; saveenv
```

12. 使用以下命令启动内核：

```
ZynqMP> run netboot
```

故障排除

表 11: **具有 TFTP 的硬件上的 PetaLinux 镜像**

描述/错误消息	描述和解决方案
`Error: "serverip" not defined.`	**问题描述** 该错误消息表示 U-Boot 环境变量 serverip 未设置。您必须将其设置为镜像所在的主机的 IP 地址。 **解决方案：** 使用以下命令设置 serverip： `ZynqMP> set serverip <HOST IP ADDRESS>;saveenv`

BSP 封装

BSP 对于团队和客户之间的分发非常有用。自定义的 PetaLinux 工程可以通过 BSP 交付给下一个级别的团队或外部客户。本节将举例说明如何何用 PetaLinux 工程封装 BSP。

要求

本节假定您已使 PetaLinux 工具软件平台做好准备，以构建一个按照您的硬件平台自定义的 Linux 系统。如需了解更多信息，请参阅 导入硬件配置。

步骤 BSP 封装

有关如何封装工程以便向 WTS 提交进行调试的步骤如下：

1. 您可以前往 PetaLinux 工程目录外部来运行 petalinux-package 命令。
2. 使用以下命令来封装 BSP。

```
$ petalinux-package --bsp -p <plnx-proj-root> --output MY.BSP
```

这将生成 MY.BSP，包括来自指定工程的下列要素：

- <plnx-proj-root>/project-spec/
- <plnx-proj-root>/config.project
- <plnx-proj-root>/.petalinux/

- `<plnx-proj-root>/pre-built/`
- `<plnx-proj-root>/.gitignore`
- `<plnx-proj-root>/components`

附加 BSP 封装选项

1. BSP 封装包含硬件资源。

```
$ petalinux-package --bsp -p <plnx-proj-root> --hwsource <hw-project-root> --output MY.BSP
```

   它不会修改指定的 PetaLinux 工程 `<plnx-proj-root>`。它会将指定的硬件工程源放入 `MY.BSP` 存档中的 `<plnx-proj-root>/hardware/` 中。

2. BSP 封装包含其他资源。

搜索路径支持已被废弃。你需要将其他资源拷贝到 components/ext_sources 目录下。如需了解更多信息，请参阅 将外部内核和 U-Boot 用于 PetaLinux。必须对 BSP 进行封装。

XILINX

第 6 章

升级工作空间

PetaLinux 工具系统软件组件（嵌入式 SW、ATF、Linux、U-Boot、OpenAMP 和 Yocto 框架）和主机工具组件（Vivado® Design Suite、赛灵思软件开发套件 (SDK)、HSI 以及更多组件）。若要升级至最新的系统软件组件，您必须安装相应的主机工具 (Vivado)。例如，如果您拥有随 2019.1 版本配套的 4.18 内核，但您想要升级至随 2019.2 版本配套的 4.19 内核，您必须安装 2019.2 PetaLinux 工具和 2019.2 Vivado 硬件工程。

`petalinux-upgrade` 命令可在无需更改主机工具组件的情况下升级系统软件组件，从而解决该问题。升级系统软件组件共有两步：第一步，升级已安装 PetaLinux 工具，然后升级单个 PetaLinux 工程。这可让您在不安装最新版 Vivado 硬件工程或 XSDK 的情况下升级。

注释: `petalinux-upgrade` 是 2019.1 版本中新采用的新命令。

重要提示! 该升级命令仅对微小升级有效。这意味着，尽管您能够使用 `petalinux-upgrade` 从 2019.1 升级至 2019.2，但您不能使用此命令从 2019.1 升级至 2020.1。

petalinux-upgrade 选项

表 12: **petalinux-upgrade 选项**

选项	功能描述	值范围	默认范围
`-h --help`	显示使用信息。	无	无
`-f --file`	目标系统软件组件的本地路径。	用户指定。目录结构应为： • `<file/esdks>` • `<file/downloads>`	无
`-u --url`	目标系统软件组件的 URL。	用户指定。URL 应为： • `<url/esdks>` • `<url/downloads>`	无
`-w, --wget-args`	将其他 `wget` 实参传递到命令。	其他 `wget` 选项	无

升级 PetaLinux 工具

从本地文件升级

从服务器 URL http://petalinux.xilinx.com/sswreleases/rel-v2019/ 下载目标系统软件组件内容。

`petalinux-upgrade` 命令将下载的路径作为输入。

1. 如果您还未安装，请安装该工具。

   注释: 确保安装区域可写入。

2. 使用 `cd <plnx-tool>` 切换到已安装的 PetaLinux 工具的目录。
3. 输入：`source settings.sh`。
4. 输入命令：`petalinux-upgrade -f <downloaded esdk path>`.

实例：

```
petalinux-upgrade -f “/scratch/ws/upgrade-workspace/eSDK”
```

注释: 此选项用于脱机升级。

从远程服务器中升级

按照这些步骤从远程服务器中升级已安装的工具目标系统软件组件。

1. 如果您还未安装，请安装该工具。

   注释: 该工具应具有 R/W 权限。

2. 前往已安装的工具。
3. 键入：source settings.sh。
4. 输入命令：`petalinux-upgrade -u <url>`.

实例：

```
petalinux-upgrade -u “http://petalinux.xilinx.com/sswreleases/rel-v2019/
sdkupdate/”
```

重要提示! 当前版本仅支持小型版本升级。

升级 PetaLinux 工程

利用升级工具升级现有工程

利用以下步骤使用升级工具升级现有工程。

1. 升级工具。从本地文件中升级，请参见 从本地文件升级。从远程服务器中升级，请参见 从远程服务器中升级。
2. 前往您要升级的 PetaLinux 工程。
3. 输入命令：`petalinux-build -x mrproper`.
4. 输入命令：`petalinux-build`，利用所有新的系统组件升级工程。

使用升级后的工具创建新工程

通过以下步骤使用升级后的工具创建新工程。

注意! 建议使用最新的 Vivado® Design Suite 和 PetaLinux 工具创建新工程。仅当您需要最新的 ssw 组件，但却是较早版本的 Vivado 硬件工程时，才使用以下选项。

1. 升级工具。从本地文件中升级，请参见 从本地文件升级。从远程服务器中升级，请参见 从远程服务器中升级。
2. 创建 PetaLinux 工程。
3. 使用 `petalinux-build` 命令构建包含所有新系统组件的工程。

XILINX

第 7 章

自定义工程

固件版本设置

本节解释如何使用 `petalinux-config` 命令配置固件版本。

要求

本节假定您已使 PetaLinux 工具软件平台做好准备，以构建一个按照您的硬件平台自定义的 Linux 系统。如需了解更多信息，请参阅 导入硬件配置。

固件版本配置步骤

1. 更改至您的 PetaLinux 工程根目录中。

```
$ cd <plnx-proj-root>
```

2. 启动顶层系统配置菜单。

```
$ petalinux-config
```

3. 选择“Firmware Version Configuration ”。
4. 按要求选择主机名称、产品名称、固件版本以进行编辑。
5. 在看到问题：“您是否希望保存您的新配置？”时，退出菜单并选择`<Yes>`。
6. 一旦目标被启动，请验证 `cat /etc/hostname` 中的主机名称、`cat /etc/petalinux/product` 中的产品名称以及 `cat /etc/petalinux/version` 中的固件版本。

根文件系统类型配置

本节详细介绍了使用 `petalinux-config` 命令进行 `RootFS` 类型配置的内容。

要求

本节假定您已使 PetaLinux 工具软件平台做好准备，以构建一个按照您的硬件平台自定义的 Linux 系统。如需了解更多信息，请参阅 导入硬件配置。

步骤根文件系统类型设置

1. 更改至您的 PetaLinux 工程根目录中。

```
$ cd <plnx-proj-root>
```

2. 启动顶层系统配置菜单。

```
$ petalinux-config
```

3. 选择“Image Packaging Configuration → Root File System Type”。
4. 按要求选择 “INITRAMFS”/“INITRD”/“JFFS2”/“NFS”/“SD card”。

   注释: SD 启动功能预期将 RootFS 加载到 ext4 分区，所有其他启动镜像加载到 FAT32 分区。
5. 保存设置。

启动镜像存储配置

本节提供有关启动器件配置的详细信息，例如闪存和 SD/MMC 使用 `petalinux-config` 命令。

要求

本节假定您已使 PetaLinux 工具软件平台做好准备，以构建一个按照您的硬件平台自定义的 Linux 系统。如需了解更多信息，请参阅 导入硬件配置。

步骤启动镜像设置

1. 更改至您的 PetaLinux 工程根目录中。

```
$ cd <plnx-proj-root>
```

2. 启动顶层系统配置菜单。

```
$ petalinux-config
```

3. 选择“Subsystem AUTO Hardware Settings → Advanced Bootable Images Storage Settings”。
4. 在高级可引导镜像存储设置子菜单中，您拥有如下选项：
   - “boot image settings” (BOOT.BIN - FSBL，PMU 固件，ATF，U-Boot)

     按要求选择根器件
     - 若要将闪存设置为根器件，则选择“primary flash”。
     - 若要将 SD 卡作为根器件，则选择“primary sd”。
   - “u-boot env partition settings”
   - “kernel image settings” (image.ub - Linux 内核，DTB 和 RootFS)

     按要求选择存储器件。
     - 若要将闪存设置为根器件，则选择“primary flash”。

- 若要将 SD 卡作为根器件，则选择“primary sd”。
- “jffs2 RootFS image settings”
- “DTB settings”

故障排除

本节描述配置启动器件时可能遇到的一些常见问题。

表 13: **启动镜像存储故障排除**

描述/错误消息	描述和解决方案
`ERROR: Failed to config linux/kernel!`	**问题描述** 此错误消息表明无法使用 menuconfig 设置 linux 内核组件。 **解决方案：** 检查所有必需的库/软件包是否安装正确。如需了解更多信息，请参阅 安装要求。

主闪存分区配置

本节详细介绍如何使用 PetaLinux menuconfig 设置闪存分区。

1. 更改至您的 PetaLinux 工程根目录中。

```
$ cd <plnx-proj-root>
```

2. 启动顶层系统配置菜单。

```
$ petalinux-config
```

3. 选择“Subsystem AUTO Hardware Settings → Flash Settings”。
4. 选择一个闪存器件作为主闪存。
5. 设置每个分区的名称和大小。

注释: PetaLinux 工具使用‘boot’、‘bootenv’和‘kernel’。'boot'存储‘BOOT.BIN’。‘bootenv’存储 u-boot env vars。‘kernel’存储‘image.ub’。

PetaLinux 工具使用并行闪存的起始地址或 SPI 闪存的起始偏移以及上述分区的大小来生成以下 U-Boot 命令：

- `update_boot`，如果启动镜像（MicroBlaze™ 处理器的 U-Boot 镜像和 Zynq®-7000 器件的 `BOOT.BIN` 镜像）被选定存储在主闪存中。
- `update_kernel` 和 `load_kernel`，如果内核镜像（FIT 镜像 `image.ub`）被选定存储在闪存中。

管理镜像大小

在嵌入式环境中，重要的是减小存储在闪存中的内核镜像大小以及 RAM 中内核镜像的静态大小。本节介绍了 `config` 项目对内核大小和 RAM 使用的影响。

FIT 镜像是默认可引导镜像格式。默认情况下，FIT 镜像由内核镜像、DTB 和 RootFS 镜像组成。

要求

本节假定您已使 PetaLinux 工具软件平台做好准备，以构建一个按照您的硬件平台自定义的 Linux 系统。如需了解更多信息，请参阅 导入硬件配置。

管理镜像大小的步骤

可以通过以下方法减小 FIT 镜像的大小：

1. 使用以下命令启动 RootFS 配置菜单：

```
$ cd <plnx-proj-root>
$ petalinux-config -c rootfs
```

2. 选择“File System Packages”。

   在此子菜单下，可以找到与 RootFS 包对应的选项列表。如果不需要某些软件包，可以通过禁用来缩小 RootFS 镜像的大小。

3. 使用以下命令启动内核配置菜单：

```
$ cd <plnx-proj-root>
$ petalinux-config -c kernel
```

4. 选择“General Setup”。

   在此子菜单下，可以找到设置 `config` 项的选项。可以禁用系统中不强制包含的任何项，以减小内核镜像的大小。例如：`CONFIG_SHMEM`, `CONFIG_AIO`, `CONFIG_SWAP`, `CONFIG_SYSVIPC`。如需了解更多详情，请参阅 Linux 内核文档。

   注释: 请注意，禁用某些 `config` 项可能会导致启动不成功。在禁用之前，应该了解 `config` 项。

   包含额外的 config 项和文件系统包分别会增加内核镜像大小和 RootFS 大小。

   如果内核或 RootFS 的大小增加，并且大于 128 MB，则需要执行以下操作：

   a. 在 `<plnx_proj>/project-spec/meta-user/recipes-bsp/u-boot/files/platform-top.h` 中提到 Bootm 长度。

```
#define CONFIG_SYS_BOOTM_LEN <value greater then image size>
```

   b. 取消 `CONFIG_SYS_BOOTMAPSZ` 在 `<plnx_proj>/project-spec/meta-user/recipes-bsp/u-boot/files/platform-top.h` 中的定义。

配置 INITRD BOOT

初始 RAM 磁盘 (INITRD) 提供了在 PetaLinux 开机过程期间利用引导加载程序加载 RAM 磁盘的能力。Linux 内核将其作为 RootFS 载入，并启动初始化流程。本节介绍了配置 INITRD 启动的程序。

要求

本节假设已创建新的 PetaLinux 工程（请参阅 创建新的 PetaLinux 工程）并已导入硬件平台（请参阅 导入硬件配置）。

步骤设置 INITRD 启动

1. 设置 RootFS 类型为 INITRD。如需了解更多信息，请参阅 根文件系统类型配置。
2. 设置 RAMDISK loadaddr。确保 loadaddr 不与内核或 DTB 地址重叠，并且是有效的 DDR 地址。
3. 构建系统镜像。如需了解更多信息，请参阅 构建系统镜像。
4. 使用以下方法之一来启动系统镜像：
   a. 利用 SD 卡在硬件上启动 PetaLinux 镜像，请参见 利用 SD 卡在硬件上启动 PetaLinux 镜像。
   b. 利用 JTAG 在硬件上启动 PetaLinux 镜像，请参见 利用 JTAG 在硬件上启动 PetaLinux 镜像。
      - 确保您已在主机中配置 TFTP 服务器。
      - 在 U-Boot 提示符下，使用以下命令设置服务器 IP 地址为主机 IP 地址：

        ```
        ZynqMP> set serverip <HOST IP ADDRESS>; saveenv
        ```

      - 使用以下命令读取镜像：

        ```
        ZynqMP> tftpb <dtb load address> system.dtb;tftpb <kernel load
        address> Image; tftpb <rootfs load address> rootfs.cpio.gz.u-boot.
        ```

      - 使用以下命令启动镜像：

        ```
        ZynqMP> booti <kernel load address> <rootfs loadaddress> <device
        tree load address>
        ```

重要提示! PetaLinux 的默认 RootFS 是 INITRAMFS。在 INITRD 模式下，内核镜像中不包含 RootFS。

设置 INITRAMFS 启动

初始 RAM 文件系统 (INITRAMFS) 是 INITRD 的继任者。在 PetaLinux 开机过程中，加载到内存中的是初始文件系统的 `cpio` 存档。Linux 内核将其作为 RootFS 载入，并启动初始化流程。

本节介绍了配置 INITRAMFS 启动的程序。

要求

本节假设已创建新的 PetaLinux 工程（请参阅 创建新的 PetaLinux 工程）并已导入硬件平台（请参阅 导入硬件配置）。

步骤设置 INITRAMFS 启动

1. 将 RootFS 类型设置为 `INITRAMFS`。如需了解更多信息，请参阅 根文件系统类型配置。

2. 构建系统镜像。如需了解更多信息，请参阅 构建系统镜像。
3. 利用以下方法之一启动系统镜像。
   a. 在 QEMU 上启动 PetaLinux 镜像，请参见 启动 QEMU 上的 PetaLinux 镜像。
   b. 利用 SD 卡在硬件上启动 PetaLinux 镜像，请参见 利用 SD 卡在硬件上启动 PetaLinux 镜像。
   c. 利用 JTAG 在硬件上启动 PetaLinux 镜像，请参见 利用 JTAG 在硬件上启动 PetaLinux 镜像。

重要提示! PetaLinux 的默认 RootFS 是 INITRAMFS。

在 INITRAMFS 模式中，RootFS 被包含在内核镜像中。

- 镜像 → 镜像（内核）+ `rootfs.cpio`（用于 Zynq® UltraScale+™ MPSoC）
- zImage → zImage（内核） + `rootfs.cpio`（用于 Zynq-7000 器件）
- image.elf → simpleImage.mb（内核） + `rootfs.cpio`（用于 MicroBlaze™ 处理器）

随着您选择 RootFS 组件，其大小按比例增加。

设置 TFTP 启动

本节描述如何为 TFTP 启动设置主机和 PetaLinux 镜像。

TFTP 启动可节省大量时间，因为比 JTAG 启动快很多，而且不必为内核源中的每次更改都刷新镜像。

要求

本节假定已满足了以下要求：

- 您已创建了一个新的 PetaLinux 工程（参见 创建新的 PetaLinux 工程）并导入了硬件平台（参见 导入硬件配置）。
- 主机上有 TFTP 服务器在运行。

PetaLinux 设置和构建系统镜像

为 TFTP 启动配置 PetaLinux 并构建系统镜像的步骤如下：

1. 更改至您的 PetaLinux 工程根目录中。

```
$ cd <plnx-proj-root>
```

2. 启动顶层系统配置菜单。

```
$ petalinux-config
```

3. 选择“Image Packaging Configuration”。
4. 选择“Copy final images to tftpboot” 并设置 tftpboot 目录。默认情况下，TFTP 目录 ID 为 /tftpboot。确保它与您主机的 TFTP 服务器建立相匹配。
5. 按 构建系统镜像 中的说明保存配置设置和构建系统。

设置 NFS 启动

Linux 系统最重要的组件之一是根文件系统。正确开发的根文件系统可为您提供在 PetaLinux 工程上工作的有用工具。因为根文件系统在规模上可变得很大，因此难以在闪存中存储。

最方便的方式是从网络中安装整个根文件系统，以便主机系统和目标共享相同的文件。根文件系统可以被快速修改，而且在联机时也可被修改（意味着即使系统在运行时也可修改文件系统）。建立一个上述系统的最常见方法是通过 NFS。

使用 NFS 的情况下，不需要手动刷新新文件。

要求

本节假定已满足了以下要求：

- 您已创建了一个新的 PetaLinux 工程（参见 创建新的 PetaLinux 工程）并导入了硬件平台（参见 导入硬件配置）。
- 拥有 Linux 文件和目录权限。
- 主机上已安装 NFS 服务器。假设在本例中设置为 `/home/NFSshare`。

PetaLinux 设置和构建系统镜像

为 NFS 启动配置 PetaLinux 和构建系统镜像的步骤如下：

1. 更改至您的 PetaLinux 工程根目录中。

   ```
   $ cd <plnx-proj-root>
   ```

2. 启动顶层系统配置菜单。

   ```
   $ petalinux-config
   ```

3. 选择“Image Packaging Configuration → Root File System Type”。
4. 选择 “NFS” 作为 RootFS 类型。
5. 选择“Location of NFS root directory”并将其设置到 `/home/NFSshare`。
6. 退出菜单配置并保存配置设置。自动生成的 DTSI 中的启动实参将自动更新。您可以检查 `<plnx-proj-root>/components/plnx_workspace/device-tree/device-tr ee/plnx_aarch64-system.dts`。
7. 启动内核配置菜单。

   ```
   $petalinux-config -c kernel
   ```

8. 选择“Networking support → IP: kernel level configuration”。
   - “IP:DHCP support”
   - “IP:BOOTP support”
   - “IP:RARP support”
9. 选择 NFS 上的“File systems → Network file systems → Root file systems”
10. 构建系统镜像。

    注释: 如需了解更多信息，请参阅 构建系统镜像。

11. 您只能在构建之后才能查看更新后的启动实参。

NFS 启动

对于 NFS 启动，RootFS 是通过 NFS，而非启动加载程序（FSBL、比特流、U-Boot）挂载的，可以使用下面提到的各种方法下载内核。

1. JTAG：在此情况下，启动加载程序和内核将通过 JTAG 下载到目标上。如需了解更多信息，请参阅 利用 JTAG 在硬件上启动 PetaLinux 镜像。

提示: 如果希望使用预构建的功能进行 JTAG 启动，请将镜像封装到预构建的目录中。如需了解更多信息，请参阅 封装预建镜像。

1. tftpboot：在此情况下，启动加载程序将通过 JTAG 下载，内核将通过 tftpboot 下载到到目标上。如需了解更多信息，请参阅 使用 TFTP 启动硬件上的 PetaLinux 镜像。
2. SD 卡：在此情况下，启动加载程序 (`BOOT.BIN`) 和内核镜像 (`image.ub`) 将复制到 SD 卡，并将从 SD 卡下载。如需了解更多信息，请参阅 利用 SD 卡在硬件上启动 PetaLinux 镜像。

设置 JFFS2 启动

日志登载闪存文件版本 2 或 JFFS2 是一种供闪存器件使用的日志结构的文件系统。本节介绍了配置 JFFS2 启动的程序。

要求

本节假设已创建新的 PetaLinux 工程（请参阅 创建新的 PetaLinux 工程）并已导入硬件平台（请参阅 导入硬件配置）。

步骤设置 JFFS2 启动

1. 将 RootFS 类型设置为 JFFS2。如需了解更多信息，请参阅 根文件系统类型配置。
2. 将主闪存设置为启动器件和启动镜像存储。如需了解更多信息，请参阅 启动镜像存储配置 和 主闪存分区配置。
3. 构建系统镜像。如需了解更多信息，请参阅 构建系统镜像。
4. 利用 JTAG 在硬件上启动 PetaLinux 镜像，请参见 利用 SD 卡在硬件上启动 PetaLinux 镜像。
5. 确保您已在主机中配置 TFTP 服务器。
6. 在 U-Boot 提示出现时，使用以下命令将服务器 IP 地址设定为主机的 IP 地址。

```
ZynqMP> set serverip <HOST IP ADDRESS>; saveenv
```

a. 检测闪存。

```
ZynqMP> sf probe 0 0 0
```

b. 擦除闪存。

```
ZynqMP> sf erase 0 0x5000000
```

c. 将镜像读取到内存上并写入闪存。

- 读取 BOOT.BIN。

```
ZynqMP> tftpboot 0x80000 BOOT.BIN
```

- 写入 BOOT.BIN。

```
ZynqMP> sf write 0x80000 0 <size of boot.bin>
```

实例： `sf write 0x80000 0 0x10EF48`

- 读取 image.ub。

```
ZynqMP> tftpboot 0x80000 image.ub
```

- 写入 image.ub。

```
ZynqMP>sf write 0x80000 <loading address of kernel> <size of
image.ub>
```

实例： `sf write 0x80000 0x580000 0x6cb0e4`

- 读取 rootfs.jffs2。

```
ZynqMP> tftpboot 0x80000 rootfs.jffs2
```

- 写入 rootfs.jffs2。

```
ZynqMP> sf write 0x80000 <loading address of rootfs.jffs2> <size of
rootfs.jffs2>
```

实例： `sf write 0x80000 0x1980000 0x7d4000`

注释: 检查 system.dts 内部的内核和 RootFS 加载地址。

7. 启用电路板上的 QSPI 闪存启动模式。
8. 复位电路板（启动将从闪存开始）。

配置 SD 卡 ext 文件系统启动

要求

本节假定已满足了以下要求：

- 您已创建了一个新的 PetaLinux 工程（参见 创建新的 PetaLinux 工程）并导入了硬件平台（参见 导入硬件配置）。
- 一张至少有 4 GB 存储空间的 SD 存储卡。建议使用速度等级为 6 或以上的卡，以获得最佳的文件传输性能。

准备 SD 卡

准备 SD 卡用于启动 PetaLinux SD 卡 ext 文件系统的步骤：

1. 使用分区编辑器（如 gparted）将 SD 卡格式化为两个分区。

2. 第一个分区的大小应该至少为 60 MB，并格式化为 FAT32 文件系统。确保分区之前有 4 MB 的空闲空间。第一个分区将包含引导加载程序、设备树和内核镜像。将这个分区标记为 BOOT。
3. 第二个分区应该格式化为 `ext4` 文件系统，可以占用 SD 卡上剩余的空间。这个分区将存储系统根文件系统。将这个分区标记为 RootFS。

为获得最佳性能，请确保 SD 卡分区为 4 MB 对齐。

PetaLinux 设置和构建系统镜像

为 SD 卡 ext 文件系统启动和构建系统镜像而配置 PetaLinux 的步骤如下：

1. 更改至您的 PetaLinux 工程根目录中。

```
$ cd <plnx-proj-root>
```

2. 启动顶层系统配置菜单。

```
$ petalinux-config
```

3. 选择“Image Packaging Configuration → Root filesystem type”。
4. 选择 “SD card”作为 RootFS 类型。
5. 退出菜单配置并保存配置设置。

   注释: 启动实参将在 `<plnx-proj-root>/components/plnx_workspace/device-tree/device-tree/syst em-conf.dtsi` 中自动更新。这些更改只有在构建之后才予以反映。
6. 构建 PetaLinux 镜像。如需了解更多信息，请参阅 构建系统镜像。
7. 生成启动镜像。如需了解更多信息，请参阅 生成 Zynq UltraScale+ MPSoC 的启动镜像。
8. 生成的 `rootfs.tar.gz` 文件将出现在 `images/linux` 目录中。若要提取，请使用 `tar xvf rootfs.tar.gz`。

拷贝镜像文件

本节介绍了如何向 SD 卡分区拷贝图像文件。假设有两个分区被安装到 `/media/BOOT` 和 `/media/rootfs`。

1. 更改至您的 PetaLinux 工程根目录中。

```
$ cd <plnx-proj-root>
```

2. 将 `BOOT.BIN` 和 `image.ub` 拷贝至 SD 卡的 BOOT 分区。`image.ub` 文件将具有设备树和内核镜像文件。

```
$ cp images/linux/BOOT.BIN /media/BOOT/
$ cp images/linux/image.ub /media/BOOT/
```

3. 将 `rootfs.tar.gz` 文件拷贝至 SD 卡的 RootFS 分区并提取文件系统。

```
$ sudo tar xvf rootfs.tar.gz -C /media/rootfs
```

为了启动此 SD 卡 ext 镜像，请参见 利用 SD 卡在硬件上启动 PetaLinux 镜像。

故障排除

表 14: **配置 SD 卡 ext 文件系统启动**

描述/错误消息	描述和解决方案
`EXT4-fs (mmcblk0p2): mounted filesystem with ordered data mode. Opts: (null) Kernel panic - not syncing: No working init found.`	**问题描述** 该消息表示 Linux 内核无法加载 EXT4 文件系统，也无法查找工作初始化文件。 **解决方案:** 提取 SD 卡的 RootFS 分区中的 RootFS。如需了解更多信息，请参阅 拷贝镜像文件。

XILINX

第 8 章

自定义 Rootfs

包含预构建库

本节解释如何将预先编译完成的库包含到 PetaLinux 根文件系统中。

如果某个库是在 PetaLinux 之外开发的，您可能只想将这个库添加到 PetaLinux 根文件系统中。在此情况下，创建一个应用模板，允许将现有内容复制到目标文件系统。

如果应用/库/模块名称有“_”，请参阅 含有“_”的配方名称。

要求

本节假定您已使 PetaLinux 工具软件平台做好准备，以构建一个按照您的硬件平台自定义的 Linux 系统。如需了解更多信息，请参阅 导入硬件配置。

包含预构建应用的步骤

如果预构建应用名称为 `mylib.so`，通过以下步骤可以将其包含到 PetaLinux 根文件系统中。

1. 确保已为 PetaLinux 目标架构编译了预编译代码，例如 MicroBlaze™ 处理器、Arm® 核等。
2. 使用以下命令创建应用。

```
$ petalinux-create -t apps --template install --name mylib --enable
```

3. 更改为新创建的应用程序目录。

```
$ cd <plnx-proj-root>/project-spec/meta-user/recipes-apps/mylib/files/
```

4. 删除现有的 `mylib` 文件，并将制预构建的 `mylib.so` 复制到 `mylib/files` 目录。

```
$ rm mylib
$ cp <path-to-prebuilt-mylib.so> ./
```

5. 编辑 `<plnx-proj-root>/project-spec/meta-user/recipes-apps/mylib/mylib.bb`。

   该文件看起来应该像下面这样。

```
# This file is the libs recipe.
#

SUMMARY = "Simple libs application"
SECTION = "PETALINUX/apps"
LICENSE = "MIT"
LIC_FILES_CHKSUM = "file://${COMMON_LICENSE_DIR}/
MIT;md5=0835ade698e0bcf8506ecda2f7b4f302"
```

```
SRC_URI = "file://mylib.so \
 "

S = "${WORKDIR}"

TARGET_CC_ARCH += "${LDFLAGS}"

do_install() {
      install -d ${D}${libdir}
      install -m 0655 ${S}/mylib.so ${D}${libdir}
}

FILES_${PN} += "${libdir}"
FILES_SOLIBSDEV = ""
```

6. 运行 `petalinux-build -c rootfs`。

   注释: 在将来版本中，不推荐使用 `petalinux-build -c rootfs` 构建 RootFS。使用 `petalinux-build` 代替。

重要提示! 需要确保由安装模板应用安装到目标文件系统中的二进制数据与系统的底层硬件实现兼容。

包含预构建应用

如果某个应用是在 PetaLinux 之外开发的（例如通过赛灵思 SDK），您可能只想在 PetaLinux 根文件系统中添加该应用的二进制文件。在此情况下，创建一个应用模板，允许将现有内容复制到目标文件系统。

本节解释如何将预先编译完成的应用包含到 PetaLinux 根文件系统中。

要求

本节假设 PetaLinux 工具软件平台已经准备好构建为硬件平台自定义的 Linux 系统。如需了解更多信息，请参阅 导入硬件配置。

包含预构建应用的步骤

如果预构建应用名称为 myapp，通过以下步骤可以将其包含到 PetaLinux 根文件系统中。

1. 确保已为 PetaLinux 目标架构编译了预编译代码，例如 MicroBlaze™ 处理器、Arm® 核等。
2. 使用以下命令创建应用。

```
$ petalinux-create -t apps --template install --name myapp --enable
```

3. 更改为新创建的应用程序目录。

```
$ cd <plnx-proj-root>/project-spec/meta-user/recipes-apps/myapp/files/
```

4. 删除现有的 `myapp` 应用，并将制预构建 myapp 复制到 `myapp/files` 目录。

```
$ rm myapp
$ cp <path-to-prebuilt-app> ./
```

重要提示! 需要确保由安装模板应用安装到目标文件系统中的二进制数据与系统的底层硬件实现兼容。

创建和添加定制库

本节介绍了如何向 PetaLinux 根文件系统添加定制库。

要求

本节假定您已使 PetaLinux 工具软件平台做好准备，以构建一个按照您的硬件平台自定义的 Linux 系统。如需了解更多信息，请参阅 导入硬件配置。

步骤添加定制库

基本步骤如下：

1. 从您工作站上的 PetaLinux 工程中运行 `petalinux-create -t apps`，创建用户应用程序：

```
$ cd <plnx-proj-root>
$ petalinux-create -t apps --template c --name <user-library-name> --
enable
```

例如：

```
$ petalinux-create -t apps --template c --name libsample --enable
```

注释: 如果应用程序名称有“_”，请参见 含有“_”的配方名称。

新的应用程序源可在 `<plnx-proj-root>/project-spec/meta-user/recipes-apps/libsample` 目录中找到。

2. 更改为新创建的应用程序目录。

```
$ cd <plnx-proj-root>/project-spec/meta-user/recipes-apps/libsample
```

3. 编辑 `project-spec/meta-user/recipes-apps/libsample/libsample.bb` 文件。

该文件看起来应该像下面这样。

```
    #
    # This file is the libsample recipe.
    #
    SUMMARY = "Simple libsample application"
    SECTION = "libs"
    LICENSE = "MIT"
    LIC_FILES_CHKSUM =
"file://${COMMON_LICENSE_DIR}/MIT;md5=0835ade698e0bcf8506ecda2f7b4f302"

    SRC_URI = "file://libsample.c \
              file://libsample.h \
              file://Makefile \
                    "

   S = "${WORKDIR}"
```

```
PACKAGE_ARCH = "${MACHINE_ARCH}"
PROVIDES = "sample"
TARGET_CC_ARCH += "${LDFLAGS}"

do_install() {
     install -d ${D}${libdir}
     install -d ${D}${includedir}
     oe_libinstall -so libsample ${D}${libdir}
     install -d -m 0655 ${D}${includedir}/SAMPLE
     install -m 0644 ${S}/*.h ${D}${includedir}/SAMPLE/
}

FILES_${PN} = "${libdir}/*.so.* ${includedir}/*"
FILES_${PN}-dev = "${libdir}/*.so"
```

4. 编辑 `project-spec/meta-user/recipes-apps/libsample/files/libsample.c` 文件。该文件看起来应该像下面这样：

```
#include <stdio.h>
   #include "libsample.h"

    int main(int argc, char **argv)
   {
     printf("Hello World!\n");
     return 0;
   }

   void samplelib()
   {
        printf("Hello, Welcome to PetaLinux -- samplelib !\n");
   }
```

5. 创建新文件 `project-spec/meta-user/recipes-apps/libsample/files/libsample.h` 并添加以下行。

```
void samplelib();
```

6. 编辑 `project-spec/meta-user/recipes-apps/libsample/files/Makefile` 文件。该文件看起来应该像下面这样。

```
APP = sample
  LIBSOURCES=*.c
  OUTS = *.o
  NAME := sample
  MAJOR = 1.0
  MINOR = 1
  VERSION = $(MAJOR).$(MINOR)

  all: lib$(NAME).so

  lib$(NAME).so.$(VERSION): $(OUTS)
        $(CC) $(LDFLAGS) $(OUTS) -shared -Wl,-soname,lib$(NAME).so.$
(MAJOR) -o lib
$(NAME).so.$(VERSION)

lib$(NAME).so: lib$(NAME).so.$(VERSION)
        rm -f lib$(NAME).so.$(MAJOR) lib$(NAME).so
        ln -s lib$(NAME).so.$(VERSION) lib$(NAME).so.$(MAJOR)
        ln -s lib$(NAME).so.$(MAJOR) lib$(NAME).so
```

```
%.o: %.c
        $(CC) $(CFLAGS) -c -fPIC $(LIBSOURCES)

clean:
        rm -rf *.o *.so *.so.*
```

7. 构建配方。

```
petalinux-build -c libsample
```

测试用户库

要求

本节假定您已构建和安装了预先编译完成的/定制用户应用程序。

步骤测试用户库

1. 使用以下命令创建应用。

```
petalinux-create -t apps --template c -n sampleapp --enable
```

2. 修改 `<plnx-proj-root>/project-spec/meta-user/recipes-apps/sampleapp/sampleapp.bb` 文件如下：

```
#
# This file is the sampleapp recipe.
#

SUMMARY = "Simple sampleapp application"
SECTION = "PETALINUX/apps"
LICENSE = "MIT"
LIC_FILES_CHKSUM =
"file://${COMMON_LICENSE_DIR}/MIT;md5=0835ade698e0bcf8506ecda2f7b4f302"

SRC_URI = "file://sampleapp.c \
                   "
S = "${WORKDIR}"

DEPENDS = " sample"

do_compile() {
        ${CC} ${CFLAGS} ${LDFLAGS} -o testsamplelib testsamplelib.c -
lsample
}

do_install() {
             install -d ${D}${bindir}
             install -m 0755 sampleapp ${D}${bindir}
}
FILES_${PN} += "sampleapp"
```

3. 编辑 `project-spec/meta-user/recipes-apps/sampleapp/files/sampleapp.c` 文件。

```
#include <stdio.h>
#include <SAMPLE/libsample.h>

int main(int argc, char **argv)
{
    printf("Hello World!\n");
    samplelib();
    return 0;
}
```

4. 应用 `petalinux-build -c sampleapp`。
5. 启动新创建的系统镜像。
6. 在目标系统控制台上运行用户应用。例如，如要运行用户应用 sampleapp：

```
# sampleapp
```

7. 确认应用结果符合预期。

创建和添加定制应用

本节介绍了如何向 PetaLinux 根文件系统添加定制应用程序。

要求

本节假定您已使 PetaLinux 工具软件平台做好准备，以构建一个按照您的硬件平台自定义的 Linux 系统。如需了解更多信息，请参阅 导入硬件配置。

步骤添加定制应用

基本步骤如下：

1. 从您工作站上的 PetaLinux 工程中运行 `petalinux-create -t apps`，创建用户应用程序：

```
$ cd <plnx-proj-root>
$ petalinux-create -t apps [--template TYPE] --name <user-application-
name> --enable
```

例如，若要用 C 语言（默认）创建名为 myapp 的用户应用程序：

```
$ petalinux-create -t apps --name myapp --enable
```

或：

```
$ petalinux-create -t apps --template c --name myapp --enable
```

若要创建 C++ 应用程序模板，请传递 `--template c++` 选项，如下所示：

```
$ petalinux-create -t apps --template c++ --name myapp --enable
```

若要创建 autoconf 应用程序模板，请传递 `--template autoconf` 选项，如下所示：

```
$ petalinux-create -t apps --template autoconf --name myapp --enable
```

新的应用程序源可在 `<plnx-proj-root>/project-spec/meta-user/recipes-apps/myapp` 目录中找到。

2. 更改为新创建的应用程序目录。

```
$ cd <plnx-proj-root>/project-spec/meta-user/recipes-apps/myapp
```

您可以看到以下 PetaLinux 模板生成的文件：

表 15: **添加定制应用文件**

模板	描述
`<plnx-proj-root>/project- spec/meta-user/ recipes-co re/images/petalinux-image - full.bbappend`	配置文件模板 - 该文件控制您的应用程序向 PetaLinux RootFS 菜单配置的集成。它还可让您选择或取消选择应用程序及其 dev、dbg 软件包进入到目标根文件系统中。
Makefile	编译文件模板 - 这是含有目标的基本 Makefile 文件，用于将您的应用程序构建和安装到根文件系统中。在向您的工程添加附加源代码文件时，该文件需要修改。
README	介绍构建用户应用程序方法的文件。
`myapp.c` 用于 C 语言； `myapp.cpp` 用于 C++ 语言	根据您的选择，采用 C 或 C++ 语言构建的简单应用程序。

注释: 如果您要使用构建伪像利用第三方实用程序进行调试，请在 `<plnx-proj-root>/project-spec/meta-user/conf/petalinuxbsp.conf` 中添加以下行：

```
RM_WORK_EXCLUDE += "myapp"
```

注释: 您可在 `${TMPDIR}/work/aarch64-xilinx-linux/myapp/1.0-r0/` 中找到全部构建伪像。

提示: 不建议在配方中使用 do_clean 映射 Make 文件清理。这是因为 Yocto 拥有其自己的 do_clean 功能。

3. 可采用您应用程序的真正源代码编辑或替换 `myapp.c/myapp.cpp` 文件。如果您要在以后修改您的定制用户应用程序，则应编辑此文件。

注意! 如果不再需要，您可以删除该应用程序目录。除了删除应用程序目录之外，您还必须将代码行：`IMAGE_INSTALL_append= " myapp"` 从 `<plnx-proj-root>/project-spec/meta-user/recipes-core/images/petalinux-image-full.bbappend` 中删除。删除目录而保留上述代码行将抛出一个错误。

创建和添加定制模块

本节解释如何将自定制内核模块添加到 PetaLinux 根文件系统。

要求

本节假定您已使 PetaLinux 工具软件平台做好准备，以构建一个按照您的硬件平台自定义的 Linux 系统。如需了解更多信息，请参阅 导入硬件配置。

步骤添加定制模块

1. 从您工作站上的 PetaLinux 内部运行 `petalinux-create -t modules`，创建用户模块：

```
$ cd <plnx-proj-root>
$ petalinux-create -t modules --name <user-module-name> --enable
```

例如，用 C 语言（默认）创建一个名为 mymodule 的用户模块：

```
$ petalinux-create -t modules --name mymodule --enable
```

您可以使用 `-h` 或`--help` 来查看 `petalinux-create -t modules` 的用途。您创建的新模块配方可在 `<plnx-proj-root>/project-spec/meta-user/recipes-modules/mymodule` 中找到。

注释: 如果模块名称有“_”，请参见 含有“_”的配方名称。

2. 更改至新创建的模块目录。

```
$ cd <plnx-proj-root>/project-spec/meta-user/recipes-modules/mymodule
```

您可以看到以下 PetaLinux 模板生成的文件：

表 16: **添加定制模块文件**

模板	描述
Makefile	编译文件模板 - 这是基本的 Makefile 文件，含有在根文件系统中构建和安装模块所需的目标。在向您的工程添加附加源代码文件时，该文件需要修改。单击这里自定义 Makefile 文件。
README	介绍建立用户模块方法的文件。
mymodule.c	简单的 C 语言内核模块。
`<plnx-proj-root>/project- spec/meta-user/ recipes-co re/images/petalinux-image - full.bbappened`	配置文件模板 - 该文件控制您的 `application/modules/ libs` 向 PetaLinux RooFS 菜单配置系统的集成。它还可让您将应用程序及其 dev、dbg 包选择或取消选择进入目标根文件系统。

3. 可用您模块的真实源代码编辑或替换 `mymodule.c` 文件。以后如果您想修改您的定制用户模块，您必须编辑此文件。

注释: 如果您要使用构建伪像利用第三方实用程序进行调试，请在 `project-spec/meta-user/conf/ petalinuxbsp.conf` 中添加以下行：

```
RM_WORK_EXCLUDE += "mymodule"
```

注释: 您可在 `${TMPDIR}/work/aarch64-xilinx-linux/mymodule/1.0-r0/` 中找到全部构建伪像。

注意! 如果不再需要模块目录，您可以将其删除。除了删除模块目录之外，您还必须从 `<plnx-proj-root>/ project-spec/meta-user/recipes-core/images/petalinux-image.bbappend` 中删除行：`IMAGE_INSTALL_append= "mymodule"`。删除目录但保留 `petalinux-image-full.bbappend` 中的上述程序行将抛出一个错误。

构建用户应用

本节解释如何构建和安装预编译/定制用户应用到 PetaLinux 根文件系统。

要求

本节假定您已经向 PetaLinux 根文件系统中纳入或添加了定制应用程序（参见 创建和添加定制应用）。

逐步构建用户应用

运行 petalinux-build（工程目录 <plnx-proj-root> 内）将重新构建系统镜像，包括选定的用户应用 myapp。（此构建进程的输出目录为 <TMPDIR>/work/aarch64-xilinx-linux/myapp/1.0-r0/。）

```
$ petalinux-build
```

将 myapp 构建到现有的系统镜像中：

```
$ cd <plnx-proj-root>
$ petalinux-build -c rootfs
$ petalinux-build -x package
```

注释: 在将来版本中，不推荐使用 petalinux-build -c rootfs 构建 RootFS。使用 petalinux-build 代替。

其他 petalinux-build 选项通过 --help 予以解释。其中一些构建选项包括：

- 如要清除选定的用户应用：

  ```
  $ petalinux-build -c myapp -x do_clean
  ```

- 如要重新构建选定的用户应用：

  ```
  $ petalinux-build -c myapp
  ```

  这将只是编译应用，编译后的可执行的文件将位于 ${TMPDIR}/work/aarch64-xilinx-linux/myapp/1.0-r0/ 目录中。

  如想使用构件与第三方工具进行调试，请将以下行：RM_WORK_EXCLUDE += "myapp" 添加到 <plnx-proj-root>/project-spec/meta-user/conf/petalinuxbsp.conf 中。如果没有这一行，BitBake 将在成功构建之后删除所有构件。

- 如要查看 myapp 的所有任务列表：

  ```
  petalinux-build -c myapp -x listtasks
  ```

- 如要安装选定的用户应用：

  ```
  $ petalinux-build -c myapp -x do_install
  ```

  注释: 在将来版本中，不建议使用 petalinux-build -c <app/package/component> -x <task>。不建议将某个组件的单个任务作为 petalinux-build 命令的一部分。

  这将把应用安装到目标 RootFS 主机副本：<TMPDIR>/work/<MACHINE_NAME>-xilinx-linux/petalinux-user-image/1.0-r0/rootfs/。

  可以在 “petalinux-config → Yocto-settings → TMPDIR” 中找到 TMPDIR。如果工程位于本地存储上，则 TMPDIR 是<plnx-proj-root>/build/tmp/。

  如想使用构件与第三方工具进行调试，请将以下行添加到 project-spec/meta-user/conf/petalinuxbsp.conf：

  ```
  RM_WORK_EXCLUDE += "myapp"
  ```

测试用户应用

要求

本节假定您已构建和安装了预先编译完成的/定制用户应用程序。如需了解更多信息，请参阅 构建用户应用。

步骤测试用户应用

1. 启动目标或 QEMU 上新创建的系统镜像。
2. 在目标系统登录控制台运行以下命令，确认用户应用在 PetaLinux 系统上：

   ```
   # ls /usr/bin
   ```

   除非已经通过其 Makefile 更改了用户应用位置，否则该用户应用将放在 `/usr/bin` 目录中。
3. 在目标系统控制台上运行用户应用。例如，如要运行用户应用 myapp：

   ```
   # myapp
   ```
4. 确认应用结果符合预期。

如果目标文件系统中缺少新应用，请确保已完成前一节所述的 `petalinux-build -x package` 步骤。这确保将应用二进制文件复制到根文件系统暂存区，并使用这个新文件系统更新目标系统镜像。

构建用户模块

本节解释如何构建和安装定制用户内核模块到 PetaLinux 根文件系统。

要求

本节假定您已经向 PetaLinux 根文件系统中纳入或添加了定制模块（参见 创建和添加定制模块）。

逐步构建用户模块

运行 `petalinux-build` （工程目录“`<plnx-proj-root>`”内）将重新构建系统镜像，包括选定的用户模块 mymodule。（此构建进程的输出目录为 `<TMPDIR>/work/<MANCHINE_NAME>-xilinx-linux/mymodule/1.0-r0/`。）

```
$ petalinux-build
```

将 mymodule 构建到现有的系统镜像中：

```
$ cd <plnx-proj-root>
$ petalinux-build -c rootfs
$ petalinux-build -x package
```

注释: 在将来版本中，不推荐使用 `petalinux-build -c rootfs` 构建 RootFS。使用 `petalinux-build` 代替。

其他 petalinux-build 选项通过 --help 予以解释。其中一些构建选项包括：

- 如要清除选定的用户模块：

```
$ petalinux-build -c mymodule -x do_cleansstate
```

- 如要重新构建选定的用户模块：

```
$ petalinux-build -c mymodule
```

  这将只是编译模块，编译后的可执行的文件将在：<TMPDIR>/work/<MANCHINE_NAME>-xilinx-linux/mymodule/1.0-r0/ 目录中。

- 如要查看该模块的所有任务列表：

```
$ petalinux-build -c mymodule -x listtasks
```

- 如要安装选定的用户模块：

```
$ petalinux-build -c mymodule -x do_install
```

  注释: 在将来版本中，不建议使用 petalinux-build -c <app/package/component> -x <task>。不建议将某个组件的单个任务作为 petalinux-build 命令的一部分。

  这将把模块安装到目标 RootFS 主机副本：<TMPDIR>/work/<MACHINE_NAME>-xilinx-linux/petalinux-user-image/1.0-r0/rootfs/。

  可以在 “petalinux-config → Yocto-settings → TMPDIR” 中找到 TMPDIR。如果工程位于本地存储上，则 TMPDIR 是 <${PROOT}>/build/tmp/。

  如想使用构件与第三方工具进行调试，请将以下行添加到 project-spec/meta-user/conf/petalinuxbsp.conf：

```
RM_WORK_EXCLUDE += "mymodule"
```

PetaLinux 自动登录

本节介绍了如何在不输入登录信息的情况下从启动中直接登录。

要求

本节假定您已使 PetaLinux 工具软件平台做好准备，以构建一个按照您的硬件平台自定义的 Linux 系统。如需了解更多信息，请参阅 导入硬件配置。

PetaLinux 自动登录步骤

按照以下步骤进行 PetaLinux 自动登录：

1. 切换到 PetaLinux 工程的根目录。

```
cd <plnx-proj-root>
```

2. 运行 petalinux-config。

3. 选择“Yocto-settings → Enable debug-tweaks”。
4. 保存配置并退出。
5. 运行 `petalinux-build`。

开机时应用程序自动运行

本节介绍了如何添加在系统开机时自动运行的应用程序。

要求

本节假定您已经添加并构建了 PetaLinux 应用程序。如需了解更多信息，请参阅 创建和添加定制应用 和 构建用户应用。

开机时应用程序自动运行的步骤

如果您在 PetaLinux 工程中的 `<plnx-proj-root>/project-spec/meta-user/recipes-apps/` 中拥有一个预建或新创建的定制用户应用程序，您可能希望在系统开机时执行它。启用程序的步骤如下：

如果您拥有预建应用程序，而您未将其包含在 PetaLinux 根文件系统中，请参见 包含预构建应用。如果您要创建定制应用程序并将其安装到 PetaLinux 根文件系统中，请参见 创建和添加定制应用。如果您的自动运行应用程序是一种永远不会退出的阻塞应用程序，可作为后台程序启动该程序。

1. 创建并安装名为 myapp-init 的新建应用程序

```
cd <plnx-proj-proot>/petalinux-create -t apps --template install -n
myapp-init  --enable
```

2. 编辑 `project-spec/meta-user/recipes-apps/myapp-init/myapp-init.bb` 文件。该文件看起来应该像下面这样：

```
#
# This file is the myapp-init recipe.
#
SUMMARY = "Simple myapp-init application"
SECTION = "PETALINUX/apps"
LICENSE = "MIT"
LIC_FILES_CHKSUM =
"file://${COMMON_LICENSE_DIR}/MIT;md5=0835ade698e0bcf8506ecda2f7b4f302"

SRC_URI = "file://myapp-init \
        "
S = "${WORKDIR}"

FILESEXTRAPATHS_prepend := "${THISDIR}/files:"

inherit update-rc.d

INITSCRIPT_NAME = "myapp-init"
INITSCRIPT_PARAMS = "start 99 S ."

do_install() {
```

```
        install -d ${D}${sysconfdir}/init.d
        install -m 0755 ${S}/myapp-init ${D}${sysconfdir}/init.d/myapp-
init
}
FILES_${PN} += "${sysconfdir}/*"
```

3. 若要作为后台程序运行 myapp，请编辑文件 `project-spec/meta-user/recipes-apps/myapp-init/files/myapp-init`。

   该文件看起来应该像下面这样：

```
#!/bin/sh
DAEMON=/usr/bin/myapp
start ()
{
        echo " Starting myapp"
        start-stop-daemon -S -o --background -x $DAEMON
}
stop ()
{
        echo " Stoping myapp"
        start-stop-daemon -K -x $DAEMON
}
restart()
{
        stop
        start
}
[ -e $DAEMON ] || exit 1

        case "$1" in
                start)
                        start; ;;
                stop)
                        stop; ;;
                restart)
                        restart; ;;
                *)
                        echo "Usage: $0 {start|stop|restart}"
                        exit 1
        esac
exit $?
```

4. 运行 `petalinux-build`。

添加层级

您可以将层级添加到 PetaLinux 工程中。THUD 版本的上游层级可在此处查找。

下列步骤展示了向 PetaLinux 工程添加 meta-my 层的方法。

1. 在 `<proj_root>/project-spec/meta-mylayer` 中拷贝或创建层级。
2. 运行 “petalinux-config → Yocto Settings → User Layers”。
3. 输入下列命令：

```
${proot}/project-spec/meta-mylayer
```

4. 保存并退出。
5. 查看 `<proj_root>/build/conf/bblayers.conf` 中的文件以便验证。

注释: 2019.1 PetaLinux 位于 THUD 基线上。层级/配方只应从 THUD 分支中选择。有些层级/配方可能与我们的架构不兼容。您对所有附加层级/配方负责。

注释: 您还可以在您的工程之外添加层级，这些层级可以在各工程之间共享。

重要提示! 如果您想更改层级优先权，您可以在此文件 BBFILE_PRIORITY_meta-mylayer = 6（0 至 10，更高的值具有更高的优先权）中更新 `${proot}/project-spec/meta-mylayer/conf/local.conf`。

将现有配方添加到 RootFS

大部分 RootFS 菜单配置是静态的。这些是赛灵思支持的实用程序。您可以在工程中添加您自己的层级或从 PetaLinux 中的现有层级中添加现有的附加配方。PetaLinux 中的层级可在 `/opt/pkg/petalinux/components/yocto/source/aarch64/` 中查找（用于 Zynq® UltraScale+™ MPSoC）。

默认情况下，`iperf3` 不在 RootFS 菜单配置中。以下示例展示了如何向 RootFS 菜单配置中添加 `iperf3`。

1. 配方的位置为 `/opt/pkg/petalinux/components/yocto/source/aarch64/layers/meta-openembedded/meta-oe/recipes-benchmark/iperf3/iperf3_3.2.bb`。
2. 在 `<plnx-proj-root>/project-spec/meta-user/recipes-core/images/petalinux-image-full.bbappend` 中添加以下行。

```
IMAGE_INSTALL_append = " iperf3"
```

重要提示! 在使用“`_append`”时，在`=` “之后应该有一个空格。

3. 运行 `petalinux-config -c rootfs`。
4. 选择“user packages → iperf3”。启用、保存并退出。
5. 运行 `petalinux-build`。

   注释: 除了 PetaLinux 默认 RootFS 菜单配置之外，您还负责在 PetaLinux 工具中可用的层级中添加配方。

   注释: 上述程序仅适用于来自用户层级的配方。

重要提示! 在 `petalinux-image-full` 中的所有配方都已锁定 sstate 状态。若要解锁，您必须将 `SIGGEN_UNLOCKED_RECIPES += "my-recipe"` 添加到 `project-spec/meta-user/conf/petalinuxbsp.conf` 中。

例如，您要在 mtd-utils 封装中做出更改，以便无需在 `project-spec/meta-user/conf/petalinuxbsp.conf` 中添加 SIGGEN_UNLOCKED_RECIPES += “mtd-utils”即可为其创建一个 .bbappend。在工程构建过程中，您会看到以下警告，而您对封装所做的更改不会包含在构建之中。

```
"The mtd-utils:do_fetch sig is computed to be
92c59aa3a7c524ea790282e817080d0a, but the sig is locked to
9a10549c7af85144d164d9728e8fe23f in SIGGEN_LOCKEDSIGS_t"
```

添加封装组

定制镜像的最好办法之一是创建一个用于构建镜像的定制封装组。一些封装组配方随 PetaLinux 工具配备。

例如：

```
$PETALINUX/components/yocto/source/aarch64/layers/meta-petalinux/recipes-core/packagegroups/packagegroup-petalinux-self-hosted.bb
```

封装组的名称应唯一，而且不得与现有配方名称冲突。

我们可以创建定制封装组，例如，ALSA 封装组看起来像这样：

```
DESCRIPTION = "PetaLinux ALSA supported Packages"

 inherit packagegroup

ALSA_PACKAGES = " \
        alsa-lib \
        alsa-plugins \
        alsa-tools \
        alsa-utils \
        alsa-utils-scripts \
        pulseaudio \
        "
RDEPENDS_${PN}_append += " \
        ${ALSA_PACKAGES} \
        "
```

这可以被添加到 `<plnx-proj-root>/meta-user/recipes-core/packagegroups/packagegroup-petalinux-alsa.bb`。

若要在 RootFS 菜单配置中添加该封装组，请将 `IMAGE_INSTALL_append = " packagegroup-petalinux-alsa"` 添加到 `<plnx-proj-root>/project-spec/meta-user/recipes-core/petalinux-image.bbappend` 中，以便在菜单配置中反映出来。

然后启动 `petalinux-config -c rootfs`，选择“user packages → packagegroup-petalinux-alsa”，保存并退出。然后运行 `petalinux-build`。

XILINX

第 9 章

调试

在 QEMU 中调试 Linux 内核

本节介绍了如何利用 GNU 调试器（GDB）在 QEMU 内部调试 Linux 内核。请注意，该功能仅采用 Zynq®-7000 器件进行了测试。

要求

本节假定您已经构建了 PetaLinux 系统镜像。如需了解更多信息，请参阅 构建系统镜像。

调试 QEMU 中 Linux 内核的步骤

1. 运行以下命令，用当前构建的 Linux 启动 QEMU：

```
$ petalinux-boot --qemu --kernel
```

2. 查看 QEMU 控制台。您应该看到 QEMU 命令的详细信息。从 `-gdb tcp:<TCP_PORT>` 获得 GDB TCP 端口。
3. 打开另一个命令控制台（确保已获得 PetaLinux 设置脚本），并切换到 Linux 目录：

```
$ cd "<plnx-proj-root>/images/linux"
```

4. 在命令模式下，在 vmlinux 内核镜像上启动 GDB：

```
$ petalinux-util --gdb vmlinux
```

您应该会看到 GDB 提示符。例如：

```
GNU gdb (Linaro GDB 2019.1) 7.12.1.20170130-git
Copyright (C) 2019 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/
gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show
copying"
and "show warranty" for details.
This GDB was configured as "--host=x86_64-unknown-linux-gnu
--target=aarch64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from vmlinux...done.
```

5. 通过运行以下 GDB 命令，连接到 GDB 中的 QEMU 目标：

```
(gdb) target remote :9000
```

6. 如要让 QEMU 继续执行：

```
(gdb) continue
```

7. 可以用 `Ctrl+C` 中断内核并恢复 GDB 提示符。
8. 可以设置断点并运行其他 GDB 命令来调试内核。

注意! 如果另一个进程正在使用端口 9000，`petalinux-boot` 将尝试使用不同的端口。请参阅 `petalinux-boot` 的输出，以确定使用了哪个端口。下面示例使用的是端口 9001：`INFO: qemu-system-arm ... -gdb tcp::9001 ...`

提示: 在内核配置菜单中启用内核调试可能会有帮助（“petalinux-config --kernel → Kernel hacking → Kernel debugging”），这样镜像中就会出现内核调试符号。

故障排除

本节介绍了在 QEMU 中调试 Linux 内核过程中您可能遇到的一些常见问题。

表 17: **在 QEMU 故障排除中调试 Linux 内核**

描述/错误消息	描述和解决方案
`(gdb) target remote W.X.Y.Z:9000:9000: Connection refused.`	**问题描述:** GDB 未能连接 QEMU 目标。这最有可能是因为端口 9000 不是 QEMU 正在使用的端口 **解决方案:** 检查 QEMU 控制台，以确保 QEMU 正在运行。 观察 Linux 主机命令控制台。它将显示完整 QEMU 命令，您应该能够看到 QEMU 在使用哪个端口。

使用 TCF 代理调试应用程序

本节介绍了利用 Eclipse 目标通信框架 (TCF) 代理调试用户应用程序。利用 TCF 代理调试应用程序的程序对于 Zynq® UltraScale+™ MPSoC 和 Zynq-7000 器件是相同的。本节介绍了 Zynq 平台用户应用程序 myapp 的基本调试程序。

要求

本节假定已满足了以下要求：

- XSDK 工具使用知识。如需了解更多信息，请参阅赛灵思软件开发套件 (SDK)。
- PetaLinux 工作环境已正确设置。如需了解更多信息，请参阅 PetaLinux 工作环境建立。
- 您已创建了一个用户应用程序并构建了包括选定的用户应用程序在内的系统镜像。如需了解更多信息，请参阅 构建用户应用。

为调试准备构建系统

1. 更改至工程目录：

```
$ cd <plnx-proj-root>
```

2. 在命令控制台上运行 `petalinux-config -c rootfs`：

```
$ petalinux-config -c rootfs
```

3. 向下滚动 Linux/RootFS 配置菜单至文件系统封装。

```
admin   --->
audio   --->
base   --->
baseutils   --->
benchmark   --->
bootloader   --->
console   --->
devel   --->
fonts   --->
kernel   --->
libs   --->
misc   --->
multimedia   --->
net   --->
network   --->
optional   --->
power management --->
utils   --->
x11   --->
```

4. 选择 “misc” 子菜单：

```
admin   --->
audio   --->
base   --->
baseutils   --->
benchmark   --->
bootloader   --->
console   --->
devel   --->
fonts   --->
kernel   --->
libs   --->
misc   --->
multimedia   --->
net   --->
network   --->
optional   --->
power management --->
utils   --->
x11   --->
```

5. 封装按字母顺序排列。导航至字母 “t”，如下所示：

```
serf   --->
sysfsutils   --->
sysvinit-inittab   --->
tbb   --->
tcf-agent --->
```

```
texi2html    --->
tiff    --->
trace-cmd    --->
util-macros    --->
v4l-utils    --->
```

6. 确保启用了 tcf 代理。

```
[*] tcf-agent
[ ] tcf-agent-dev
[ ] tcf-agent-dbg
```

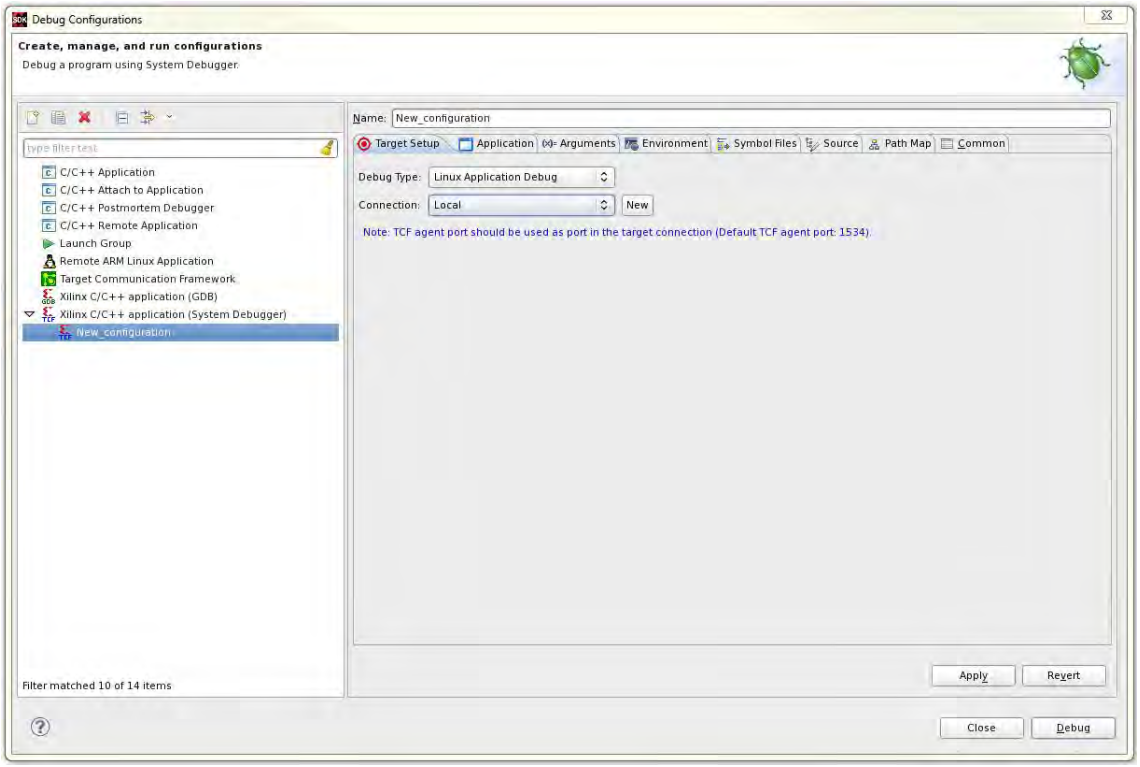
7. 选择 “console/network” 子菜单，然后单击进入 “dropbear” 子菜单。确保启用“dropbear-openssh-sftp-server”。

```
[*] dropbear
```

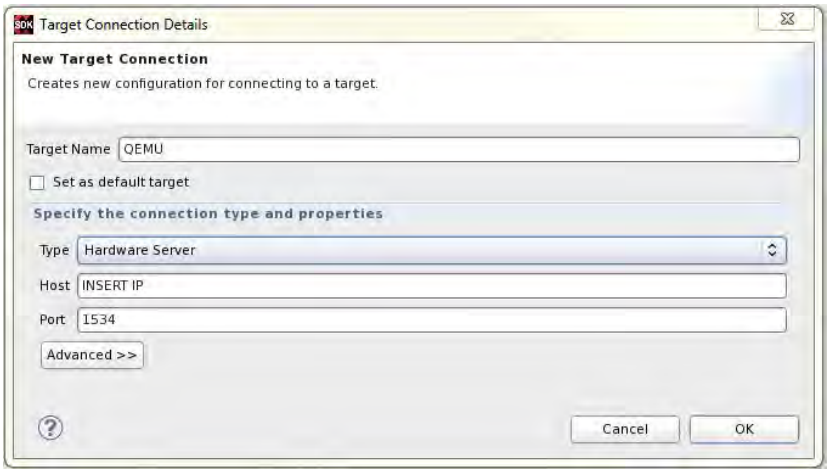
8. 选择“console/network → submenu → openssh”。确保启用“"openssh-sftp-server”。
9. 退出菜单。
10. 重新构建包括 myapp 在内的目标系统镜像。如需了解更多信息，请参阅 构建系统镜像。

执行调试会话

1. 使用新镜像启动电路板（或 QEMU）。
2. 启动日志应该表明 tcf-agent 已经启动。应该看到以下消息： “Starting tcf-agent: OK”
3. 启动赛灵思 SDK 并创建工作空间。
4. 通过选择“File → New → Project” 添加“Hardware Platform Specification”。
5. 在弹出窗口中，选择“Xilinx → Hardware Platform Specification”。
6. 给硬件工程起名。例如：ZC702。
7. 查找目标硬件的 `system.hdf/system.dsa`。这可以在 `<plnx-proj-root>/project-spec/hw-description` 中找到。
8. 选择“Run → Debug Configurations”，打开调试启动配置窗口。
9. 创建新的 “Xilinx C/C++ application (System Debugger)” 并启动配置：



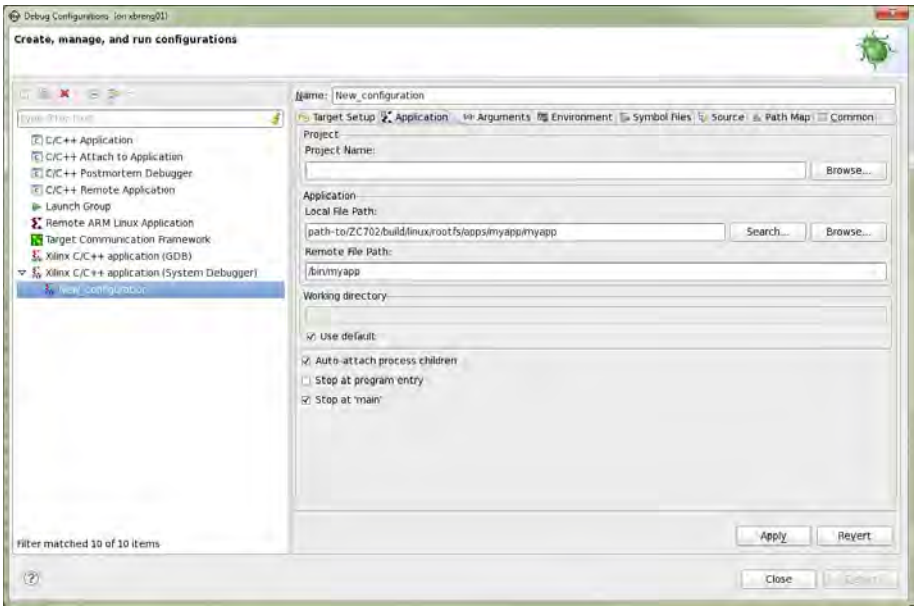
10. “Debug Type” 应该设置为 “Linux Application Debug”。
11. 选择“New”选项以输入连接详情。



12. 给目标连接起名，并指定主机（目标的 IP 地址）。
13. 设置 tcf-agent 端口并选择 “OK”。

重要提示! 如果在 QEMU 上进行调试，在非根模式（默认）或根模式下进行测试时，请参阅 附录 D: QEMU 虚拟网络模式 获取关于 IP 和端口重定向的信息。例如，如果在非根模式下测试，则需要在后续步骤中使用 localhost 作为目标 IP。

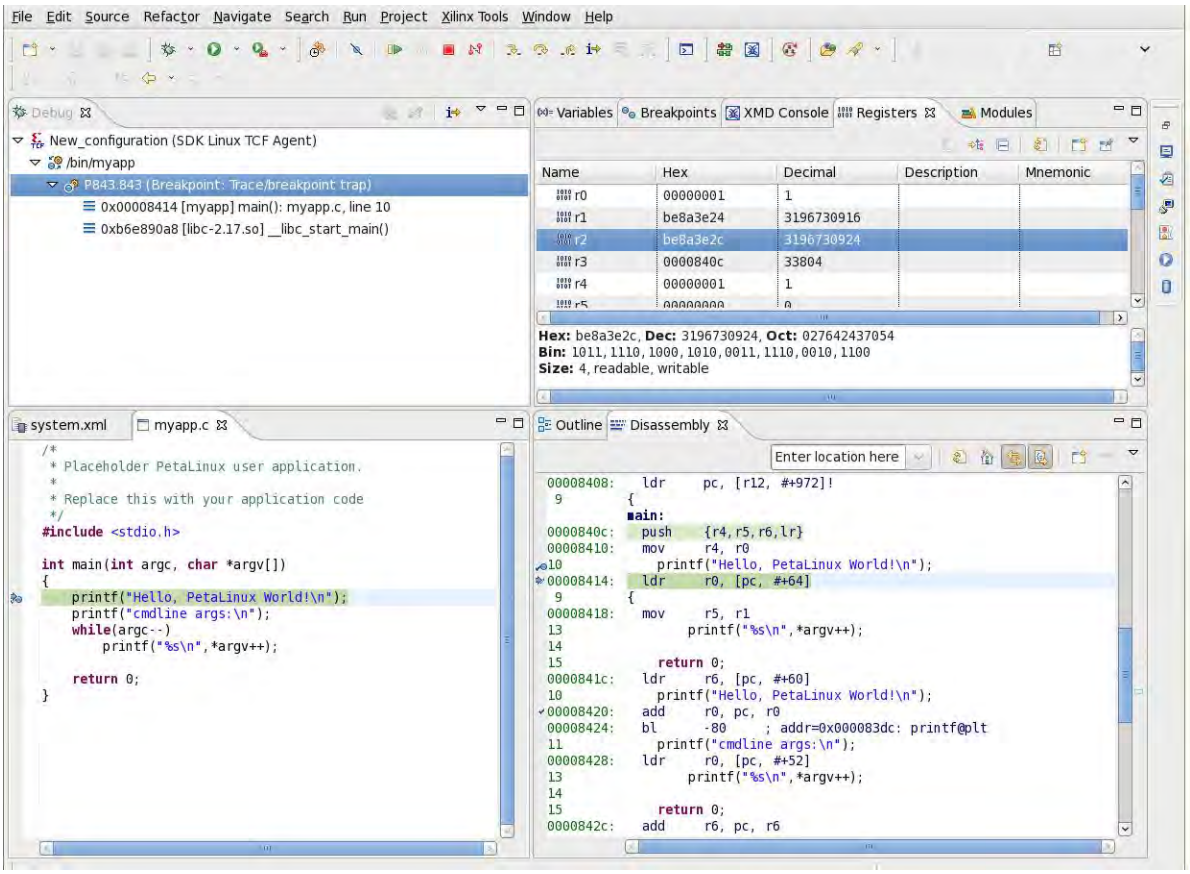
14. 切换到“Application”标签。



15. 在工程目录中输入已编译应用的“Local File Path”。例如：`<TMPDIR>/work/aarch64-xilinx-linux/myapp1/1.0-r0/image/usr/bin/`。

    注释: 创建应用时，需要将 `RM_WORK_EXCLUDE += "myapp"` 添加到 `project-spec/meta-user/conf/petalinuxbsp.conf`，否则镜像无法用于调试。

16. 目标文件系统上的“Remote File Path”应该是可以找到应用的位置。例如：`/usr/bin/myapp`。
17. 选择 “Debug” 以应用配置并开始调试会话。（如果被要求切换到“Debug Perspective”，请接受）。
18. 标准 XSDK 调试流程准备启动：



提示: 如要分析代码和调试，可以使用以下快捷键：

- Step Into (F5)
- Step Over (F6)
- Step Return (F7)
- Resume (F8)

使用 GDB 调试 Zynq UltraScale+ MPSoC 应用

PetaLinux 支持用 GDB 调试 Zynq® UltraScale+™ MPSoC 用户应用。本节描述的是基本的调试步骤。

要求

本节假定已满足了以下要求：

- PetaLinux 工作环境已正确设置。如需了解更多信息，请参阅 PetaLinux 工作环境建立。
- 您已创建了一个用户应用程序并构建了包括选定的用户应用程序在内的系统镜像。如需了解更多信息，请参阅 构建用户应用。

为调试准备构建系统

1. 更改至工程目录：

```
$ cd <plnx-proj-root>
```

2. 在 `<plnx-proj-root>/project-spec/meta-user/recipes-core/images/peta linux-image.bbappend` 中添加以下行：

```
IMAGE_INSTALL_append = " myapp-dev"
IMAGE_INSTALL_append = " myapp-dbg"
```

3. 在命令控制台上运行 `petalinux-config -c rootfs`：

```
$ petalinux-config -c rootfs
```

4. 向下滚动“user packages Configuration”菜单至“Debugging”：

```
Filesystem Packages    --->
PetaLinux Package Groups --->
apps    --->
user packages    --->
PetaLinux RootFS Settings   --->
```

5. 选择“user packages”。

   [X] myapp-dbg

   [ ] myapp-dev

6. 选择“myapp-dbg”。退出 myapp 子菜单。
7. 退出“user packages”子菜单，并选择“Filesystem Packages → misc → gdb”。
8. 选择 “gdb”，并确保启用 GDB 服务器：

   [ ] gdb

   [ ] gdb-dev

   [X] gdbserver

   [ ] gdb-dbg

9. 退出菜单并选择 “<Yes>”，以保存配置。
10. 重建目标系统镜像。在 `<plnx-proj-root>/project-spec/meta-user/conf/petalinuxbsp.conf` 中添加以下行。

```
RM_WORK_EXCLUDE += "myapp"
```

如需了解更多信息，请参阅 构建系统镜像。

执行调试会话

1. 使用上面创建的新镜像启动电路板（或 QEMU）。

2. 使用目标系统控制台上的用户应用，运行 `gdbserver`（设置为监听端口 1534）：

```
root@plnx_aarch64:~# gdbserver host:1534 /usr/bin/myapp
Process /bin/myapp created; pid = 73
Listening on port 1534
```

1534 是 `gdbserver` 端口 - 可以是任何未使用的端口号

3. 在工作站上，导航至已编译用户应用目录：

```
$ cd <<TMPDIR>/work/aarch64-xilinx-linux/myapp1/1.0-r0/image/usr/bin/
myapp
```

4. 运行 GDB 客户端。

```
$ petalinux-util --gdb myapp
```

5. GDB 控制台将会启动：

```
...
GNU gdb (crosstool-NG 1.18.0) 7.6.0.20130721-cvs
...
(gdb)
```

6. 在 GDB 控制台中，使用以下命令连接到目标机器：
   - 使用目标系统的 IP 地址，例如：192.168.0.10。如果不确定 IP 地址，可以在目标控制台上运行 `ifconfig` 核实。
   - 仕様端口 1534。如果在前面步骤中选择了不同的 GDB 服务器端口号，则使用该值。

重要提示! 如果在 QEMU 上调试，在非根模式（默认）或根模式下测试时，请参考“QEMU 虚拟网络模式”以获得关于 IP 和端口重定向的信息。例如，如果在非根模式下测试，则需要在后续步骤中使用 localhost 作为目标 IP。

```
(gdb) target remote 192.168.0.10:1534
```

GDB 控制台将连接到远程目标。目标控制台的 GDB 服务器将显示以下确认信息，其中包括主机 IP：

```
Remote Debugging from host 192.168.0.9
```

7. 在开始执行程序之前，请创建一些断点。使用 GDB 控制台时，可以使用功能名和行号在整个代码中创建断点。例如为 `main` 功能创建断点：

```
(gdb) break main
Breakpoint 1 at 0x10000444: file myapp.c, line 10.
```

8. 通过在 GDB 控制台中执行 continue 命令运行程序。GDB 将开始执行程序。

```
(gdb) continue
Continuing.
Breakpoint 1, main (argc=1, argv=0xbffffe64) at myapp.c:10
10  printf("Hello, PetaLinux World!\n");
```

9. 如要打印当前程序位置的代码列表，请使用 `list` 命令。

```
(gdb) list
5 */
6 #include <stdio.h>
7
8 int main(int argc, char *argv[])
9 {
```

```
10 printf("Hello, PetaLinux World!\n");
11 printf("cmdline args:\n");
12 while(argc--)
13 printf("%s\n",*argv++);
14
```

10. 尝试用 `step`、`next` 和 `continue` 命令。可以使用 `break` 命令设置和删除断点。可以使用 GDB 控制台 `help` 命令获取关于这些命令的更多信息。
11. 程序完成时，目标系统上的 GDB 服务器应用将退出。下面是控制台显示的消息示例：

```
Hello, PetaLinux World!
cmdline args:
/usr/bin/myapp
Child exited with status 0
GDBserver exiting
root@plnx_aarch64:~#
```

提示: 将自动创建 `.gdbinit` 文件来建立库路径。您可以在这个文件的末尾添加自己的 GDB 初始化命令。

进一步了解 GDB

如需了解更多信息，请访问 www.gnu.org。有关 GDB 的一般用法，请参阅 GDB 工程文档。

故障排除

本节介绍了在使用 GDB 调试应用程序过程中您可能遇到的一些常见问题。

表 18: **使用 GDB 故障排除调试 Zynq UltraScale+ MPSoC 应用程序**

描述/错误消息	描述和解决方案
`GDB error message: <IP Address>:<port>: Connection refused. GDB cannot connect to the target board using <IP>: <port>`	**问题描述** 该错误消息表明 GDB 客户端未能连接到 GDB 服务器。 **解决方案:** 检查 `gdbserver` 是否在目标系统上运行。 检查是否已有另一个 GDB 客户端已经连接到了 GDB 服务器上。查看目标控制台即可完成此项检查。如果您从主机 <IP> 中看到远程调试，则意味着有另一个 GDB 客户端连接到服务器上。 检查 IP 地址和端口是否正确设置。

调试单独的 PetaLinux 组件

PMU 固件

https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841724/PMU+Firmware#PMUFirmware-DebuggingPMUFWusingSDK

FSBL

https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842019/FSBL#FSBL-WhatarevariouslevelsofdebugprintsinFSBL

U-Boot

https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842557/Debug+U-boot

Linux

https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/123011167/Linux+Debug+infrastructure+KProbe+UProbe+LTTng

https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/123011146/Linux+Debug+infrastructure+Kernel+debugging+using+KGDB

XILINX

第 10 章

高级设置

Menuconfig 使用方法

如要选择以前未选择的菜单/子菜单，请按下箭头键向下滚动菜单或按上箭头键向上滚动菜单。光标在菜单上后，按下 “y”。如要取消选择菜单/子菜单，请遵循相同的过程并在最后按下 “n”。

PetaLinux 的 menuconfig 系统

在这个版本中，在子菜单中可用的 Linux 系统组件显示如下：

- 第一阶段启动加载器 (FSBL)
- PMU 固件（仅限 Zynq® UltraScale+™ MPSoC）
- U-Boot
- 内核
- ATF （仅限 Zynq UltraScale+ MPSoC）

对于 ATF、U-Boot 和内核，共有 3 个选项可用：

1. 默认

   默认组件随 PetaLinux 工具配备。

2. 外部源

   当您从任何指定位置下载组件时，您可以馈送您的组件，而不是通过本配置选项馈送默认组件。

   注释: 外部源文件夹对于某个工程或其用户必须唯一，但内容可以修改。如果外部源是 git 存储库、其检出状态应适合构建本工程。

3. 远程

   如果您要构建在定制 git 存储库中的组件，则必须使用本配置选项。

设置

当在系统级菜单配置中选定某个组件以启用自动配置 (autoconfig) 时，在 `petalinux-config` 运行时，其配置文件即被自动更新。

表 19: **组件及其配置文件**

菜单中的组件	启用了自动配置时受影响的文件
设备树	以下文件位于 <plnx-proj-root>/components/plnx_workspace/device-tree/device-tree/ • skeleton.dtsi (仅限 Zynq-7000 器件) • zynq-7000.dtsi (仅限 Zynq UltraScale+ MPSoC) • zynqmp-clk-ccf.dtsi (仅限 Zynq UltraScale+ MPSoC) • pcw.dtsi (Zynq-7000 器件 和 Zynq UltraScale+ MPSoC) • pl.dtsi • system-conf.dtsi • system-top.dts • <board>.dtsi
内核	以下文件位于 <plnx-proj-root>/project-spec/meta-plnx-generated/recipes-kernel/linux/configs/ plnx_kernel.cfg bsp.cfg
U-Boot	以下文件位于 <plnx-proj-root>/project-spec/meta-plnx-generated/recipes-bsp/u-boot/configs/ config.cfg config.mk (仅限 MicroBlaze™) platform-auto.h

“Subsystem AUTO Hardware Settings”菜单

“Subsystem AUTO Hardware Settings”菜单允许自定义 Linux 系统如何与底层硬件平台交互。

“System Processor”

“System Processor”菜单指定系统运行的 CPU 处理器。

“Memory Settings”

“Memory Settings”菜单允许：

- 选择哪个存储器 IP 是主系统存储器
- 设置系统存储器基址
- 设置系统存储器大小
- 将 U-Boot 文本基址偏移设置为存储器高地址

此菜单中的配置将影响设备树和 U-Boot 自动配置 (autoconfig) 文件中的存储器设置。

如果选择 manual 作为主存储器，则由您负责确保系统存储器的正确设置。

“Serial Settings”

“Serial Settings”子菜单允许选择哪个串行器件是系统的主 STDIN/STDOUT 接口。如果选择 `manual` 作为主串行接口，则由您负责确保系统串行接口的正确设置。

“Ethernet Settings”

“Ethernet Settings”子菜单允许：

- 选择哪个以太网是系统的主以太网
- 选择随机化 MAC 地址
- 设置主以太网的 MAC 地址

如果 MAC 地址被编程到 EEPROM 中，这里保持空。有关编程和设置 EEPROM 的命令，请参阅 U-Boot 文档。

- 设置在主以太网上使用 DHCP 或是静态 IP

如果选择 manual 作为主以太网，则由您负责确保系统以太网的正确设置。

“Flash Settings”

“Flash Settings”子菜单允许：

- 选择哪个闪存是系统的主闪存
- 设置闪存分区表

如果选择 manual 作为主闪存，则由您负责系统闪存设置。

“SD/SDIO Settings”

“SD/SDIO Settings”子菜单仅用于 Zynq-7000 器件和 Zynq UltraScale+ MPSoC。它允许选择哪个 SD 控制器是系统的主 SD 卡接口。

如果选择 manual 作为主闪存，则由您负责设置系统闪存。

“Timer Settings”

“Timer Settings”子菜单用于 MicroBlaze 器件和 Zynq UltraScale+ MPSoC。它允许选择哪个定时器是系统的主定时器。

重要提示! MicroBlaze 系统要求设有主定时器。

“Reset GPIO Settings”

“Reset GPIO Settings”子菜单仅用于 MicroBlaze 处理器。它允许选择哪个 GPIO 是系统复位 GPIO。

提示: MicroBlaze 系统使用 GPIO 作为复位输入。如果选定复位 GPIO，就可以从 Linux 启动系统。

“RTC Settings”

选择一个用作 Linux 内核主定时器的 RTC 实例。如果您首选的 RTC 不在列表中，请选择 “manual”，以便 RTC 启用适当的内核驱动。

“Advanced Bootable Images Storage Settings”

“Advanced Bootable Images Storage Settings”子菜单允许指定可启动镜像的位置。PetaLinux 使用此子菜单中的设置来设置 U-Boot。

如果禁用此子菜单，PetaLinux 将使用“Flash Settings”子菜单中指定的闪存分区表来定义可启动镜像的位置。

表 20: **闪存分区表**

可启动镜像 /U-Boot 环境分区	默认分区名	描述
启动镜像	boot	`BOOT.BIN` （Zynq-7000 器件 和 Zynq UltraScale+ MPSoC） 可重定位 U-Boot BIN 文件 (`u-boot-s.bin`)（MicroBlaze 处理器）
U-Boot 环境分区	bootenv	U-Boot 环境变量分区。如果选择 “primary sd”，则 U-Boot 环境保存在第一个分区。如果选择 “primary flash”，则 U-Boot 环境保存在闪存分区名称选项中提到的分区。
内核镜像	kernel	内核镜像 `image.ub` (FIT 格式)
DTB 镜像	dtb	如果禁用“Advanced bootable images storage Settings”，并且在闪存分区表设置中找到 DTB 分区，则 PetaLinux 将设置 U-Boot 从分区表加载 DTB。否则，它假定 DTB 包含在内核镜像中。

“Kernel Bootargs”

“Kernel Bootargs”子菜单允许 PetaLinux 在 DTS 中自动生成内核启动命令行设置，或者传递 PetaLinux 用户定义的内核启动命令行设置。以下是默认的 bootargs。

```
Microblaze-full -- console=ttyS0,115200 earlyprintk
Microblaze-lite -- console=ttyUL0,115200 earlyprintk
zynq            -- console=ttyPS0,115200 earlyprintk
zynqmp          -- earlycon clk_ignore_unused root=/dev/ram rw
```

注释: 在 Zynq UltraScale+ MPSoC 中，如果希望在控制台上看到内核错误打印信息，请添加 `earlycon console=<device>,<baud rate> clk_ignore_unused root=/dev/ram rw`。实例：`earlycon console=/dev/ttyPS0,115200 clk_ignore_unused root=/dev/ram rw` in `system_user.dtsi`。

如需了解更多信息，请参阅内核文档。

“ATF Compilation Configuration”

只有在 Zynq UltraScale+ MPSoC 平台上才会出现“ATF Compilation Configuration”。此子菜单允许设置：

- 额外的 ATF 编译设置
- 更改 bl31 二进制文件基址
- 更改 bl31 二进制文件大小

“Power Management Kernel Configuration”

“Power Management Kernel Configuration”选项允许 PetaLinux 添加与功耗相关的内核配置。

选择此选项可通过 `plnx-kernel.cfg` 启用/禁用与功耗管理相关的内核设置。这些设置稍后将应用于内核 defconfig (xilinx_zynqmp_defconfig)。如果没有选择这个选项，PetaLinux 将不会明确启用/禁用内核设置。

来自 Linux defconfig 的默认配置保留在 PetaLinux 工程中。

“u-Boot Configuration”

“U-Boot Configuration”子菜单允许选择由 PetaLinux 进行 U-Boot 自动配置 (autoconfig) 或 U-Boot 电路板配置目标。

“Image Packaging Configuration”

“Image Packaging Configuration”子菜单允许设置以下镜像封装配置：

- 根文件系统类型
- 生成的可启动内核镜像文件名
- Linux 内核镜像散列函数
- DTB 填充大小
- 是否将可启动镜像复制到主机 TFTP 服务器目录。

提示: `petalinux-build` 工具总是生成 FIT 镜像作为内核镜像。

“Firmware Version Configuration”

“Firmware Version Configuration”子菜单允许设置固件版本信息：

表 21: **固件版本选项**

固件版本选项	目标 RootFS 中的文件
主机名	`/etc/hostname`
产品名称	`/etc/petalinux/product`
固件版本	`/etc/petalinux/version`

提示: 主机名不会更新。如需了解更多详情，请参见赛灵思答复记录 69122。

Zynq-7000 器件和 Zynq UltraScale+ MPSoC 的 FPGA 管理器配置和使用

FPGA 管理器为 Linux 提供可编程逻辑 (PL) 设置接口。它将比特流和 dtbos 打包到 RootFS 中的 `/lib/firmware` 目录。

为 Zynq UltraScale+ MPSoC 创建 PetaLinux 工程后，按照以下步骤构建 FPGA 管理器支持：

1. 转到 `cd <proj root directory>`。
2. 在 `petalinux-config` 命令中，选择 “FPGA Manager → [*] Fpga Manager”

   注释: PetaLinux FPGA 管理器配置，如被选定：

   1. 将生成 `pl.dtsi` 节点作为设备数叠加 (dtbo)。
   2. 将 .bin 格式的比特流和 dtbos 打包到 RootFS 中的 `/lib/firmware/base` 目录。
   3. `BOOT.BIN` （使用 `petalinux-package` 命令生成的）不会有比特流。
3. 在“FPGA Manager” ---> “() Specify hw directory path”中指定其他 hw 文件。

注释: 这一步是可选的。只有当需要将相同的 PS 和相应的 dtbos 的多个比特流打包到 RootFS 中时，才是必要的。它将在 `/lib/firmware/<DSA/HDF name>` 的 RootFS 中生成并打包 .bin 格式的比特流及其 dtbo。确保 DSA/HDF 在 hw 目录路径和 `<PROOT>/project-spec/hw-description/system<.hdf/.dsa>` 上的 PS 设计是相同的。

4. 运行 `petalinux-build`。

在目标上加载全比特流的例子：

```
root@xilinx-zcu102-2019_1:~# fpgautil -o /lib/firmware/base/pl.dtbo -b
/lib/firmware/base/design_1_wrapper.bit.bin

Time taken to load DTBO is 239.000000 milli seconds. DTBO loaded through
ZynqMP FPGA manager successfully.
```

有关 BOOT.BIN 生成信息，请参阅 `petalinux-package` 命令。

通过 sysfs 加载完整的比特流 — 仅加载比特流：

```
root@xilinx-zcu102-2019_1:~# fpgautil  -b /mnt/design_1_wrapper.bit.bin

Time taken to load BIN is 213.000000 milli seconds. BIN FILE loaded through
zynqMP FPGA manager successfully.
```

有关更多选项，请参阅帮助部分：`root@xilinx-zcu102-2019_1:~# fpgautil -h`

如需了解更多信息，请参阅 http://www.wiki.xilinx.com/Solution+ZynqMP+PL+Programming。

Zynq-7000 器件和 Zynq UltraScale+ MPSoC 设备树叠加配置

选择此选项将 pl 与基础 DTB 分离，并构建 `pl.dtsi` 来生成 `pl.dtbo`。

创建 PetaLinux 工程后，按照以下步骤添加叠加支持：

1. 转到 `cd <proj root directory>`。
2. 在 `petalinux-config` 命令中，选择 “DTG Settings → Device tree overlay”
3. 运行 `petalinux-build`
4. 这将生成 `pl.dtbo`（在 `images/linux` 目录内）。

FPGA 管理器覆盖所有选项。只有在 FPGA 管理器未被选中时它才起作用。

从 .bit 向 .bin 转换比特流

1. 利用以下内容创建 bif 文件：

```
all:
{
        [destination_device = pl] <bitstream in .bit> ( Ex:
systemdesign_1_wrapper.bit )
}
```

2. 运行下列命令：

```
 bootgen -image bitstream.bif -arch zynqmp -process_bitstream bin
```

注释: Bit/bin 文件名应与 `pl.dtsi` (`design_1_wrapper.bit.bin`) 中规定的固件名称相同。

配置删除 PL 设备树

如果用户不依赖 PL 的 IP 地址，选择该配置选项，以跳过 PL 节点。此外，如果任何 DTG 的 PL IP 生成错误，则您只需启用此标识，而 DTG 不会生成任何 PL 节点。

1. 转到 `cd <proj root directory>`。
2. 在 `petalinux-config` 命令中，从设备树中选择“DTG Settings → Remove PL”。
3. 运行 `petalinux-build`。

注释: FPGA 管理器覆盖所有这些选项。只有在 FPGA 管理器未被选中时它才起作用。

注释: 如果您同时选择设备树覆盖和从设备树中删除 PI，则基础 DTB 会有覆盖支持的入口，但不会生成任何 PL DTBO。

Yocto 设置

Yocto 设置可让您配置工程中可用的各种 Yocto 功能。

表 22: **Yocto Settings**

参数	描述
TMPDIR Location	该目录被 BitBake 用于存储日志和构建伪像
YOCTO_MACHINE_NAME	指定工程的 Yocto 机器名称
Parallel thread execution	限制 BitBake 实例的线程数量
Add pre-mirror url	为下载组件的源代码添加镜像站点
Local sstate feeds settings	在具体位置使用本地 sstate 高速缓存
Enable Debug Tweaks	无需密码登录进入目标
Enable Network sstate feeds	启用的 NW sstate 馈送
User layers	将用户层添加到工程中
BB_NO_NETWORK	在启用时，互联网的访问权在构建机器上被禁用

配置树外构建

PetaLinux 能够从 git 库中自动下载最新的内核/U-Boot 源代码。本节介绍了该功能的工作方式及其在系统级菜单配置中的使用方法。它介绍了进行树外构建的两种方式。

要求

本节假定已满足了以下要求：

- PetaLinux 工具软件平台已经准备好构建为硬件平台自定义的 Linux 系统。如需了解更多信息，请参阅 导入硬件配置。
- `git` 访问的互联网连接可用。

设置 Out-of-tree 构建的步骤

请通过以下步骤设置 UBOOT/Kernel 的 out-of-tree 构建。

1. 更改至您的 PetaLinux 工程根目录中。

```
$ cd <plnx-proj-root>
```

2. 启动顶层系统配置菜单。

```
$ petalinux-config
```

3. 选择 “Linux Components Selection” 子菜单。
   - 对于内核，请选择“linux-kernel () → remote”。

     ( ) linux-xlnx

     (X) remote

     ( ) ext-local-src
   - 对于 U-Boot，请选择“u-boot () → remote”。

     ( ) u-boot-xlnx

     (X) remote

     ( ) ext-local-src
4. 对于内核，请选择“Remote linux-kernel settings → Remote linux-kernel git URL”，并输入 Linux 内核的 git URL。

   例如，如要使用 https://github.com/Xilinx/linux-xlnx，请输入：

   ```
   git://github.com/Xilinx/linux-xlnx.git;protocol=https
   ```

   对于 U-Boot，请选择“Remote U-Boot settings → Remote u-boot git URL”，并输入 Linux 内核的 git URL。例如：

   ```
   git://github.com/Xilinx/u-boot-xlnx.git;protocol=https
   ```

   提供远程 git 链接后，必须为“git TAG/Commit ID”选择提供以下任何值，否则将会出现错误消息。

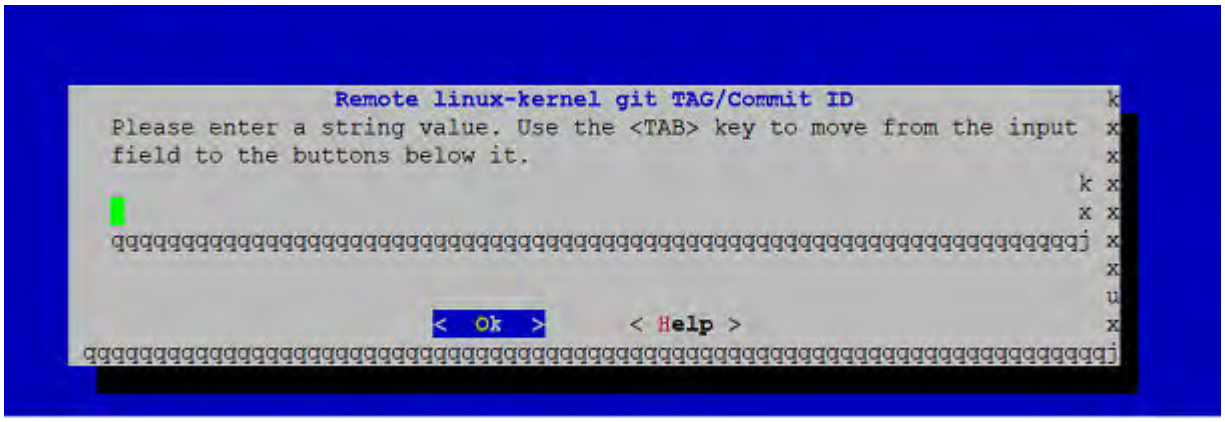
   必须为此设置设定以下任何值，否则将出现错误消息。
   - 如要指向当前已签出分支的库 HEAD：

     ```
     s${AUTOREV}
     ```
   - 如要指向任何标签：

     ```
     tag/mytag
     ```
   - 如要指向任何 commit id：

     ```
     commit id sha key
     ```

     选择“git TAG/Commit ID”后，可以看到输入字符串值的提示，如下图所示。输入上述任何一个设定值。



5. 退出菜单，保存设置。

将外部内核和 U-Boot 用于 PetaLinux

PetaLinux 包括内核源和 U-Boot 源。但是，可以使用 PetaLinux 构建自己的内核和 U-Boot。

PetaLinux 支持内核、U-Boot 和 ATF 的本地源。

为外部源创建目录 `<plnx-proj-root>/components/ext_sources/`。

1. 复制内核源目录：

   ```
   <plnx-proj-root>/components/ext_sources/<MY-KERNEL>
   ```

2. 复制 U-Boot 源目录：

   ```
   <plnx-proj-root>/components/ext_sources/<MY-U-BOOT>
   ```

3. 运行 `petalinux-config`，并进入“Linux Components Selection”子菜单。
   - 对于内核，请选择 “linux-kernel () --->”，然后选择 “ext-local-src”。

     ( ) linux-xlnx

     ( ) remote

     (X) ext-local-src
   - 对于 U-Boot，请选择“u-boot () --->”，然后选择“ext-local-src”。

     ( ) u-boot-xlnx

     ( ) remote

     (X) ext-local-src
4. 添加外部源路径。
   - 对于内核，选择“External linux-kernel local source settings --->”。输入路径：

     `${TOPDIR}/../components/ext_sources/<MY-KERNEL>`
   - 对于 U-Boot，选择“External u-boot local source settings --->”。输入路径：

     `${TOPDIR}/../components/ext_sources/<MY-U-BOOT>`

${TOPDIR} 是指向 <plnx-proj-root>/build 目录的一个 Yocto 变量。还可以指定源绝对路径。源也可以放在工程之外。

注释: 在工程中创建带有外部源的 BSP 时，您有责任将源复制到工程中并进行打包。如需了解更多信息，请参阅 BSP 封装。

重要提示! 在 components/ 下不一定要有外部源。还可以指定工程之外的任何位置。但是，在封装 BSP 时，您要负责将外部源复制到 components/ 并设置相对路径。

注释: 如果外部源是 git 库，那么其签出状态必须适合正在构建的工程。

故障排除

本节描述在设置 out-of-tree 构建时可能遇到的一些常见问题。

表 23: **设置 out-of-tree 构建故障排除**

描述/错误消息	
fatal: The remote end hung up unexpectedly ERROR: Failed to get linux-kernel	**问题描述** 此错误消息表明，系统无法使用远程 git URL 下载源代码 (Kernel/UBOOT)，因此无法继续 petalinux-build。 **解决方案:** 检查输入的远程 git URL 是否正确。 如果上述解决方案不能解决问题，请使用以下命令清除构建: $ petalinux-build -x mrproper 上述命令将删除以下目录。 < plnx-proj-root>/images/ <plnx-proj-root>/build/ 重新构建系统镜像。如需了解更多信息，请参阅 构建系统镜像。

设置工程组件

如果希望执行高级 PetaLinux 工程配置，例如启用 Linux 内核选项或修改闪存分区，请使用 petalinux-config 工具和适当的 -c COMPONENT 选项。

重要提示! 赛灵思技术支持仅支持 Linux 内核配置中的赛灵思驱动或最优化。如需了解更多有关 Linux 的赛灵思驱动的信息，请参阅 https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841873/Linux+Drivers。

下例显示如何使用 petalinux-config 来检查或修改 PetaLinux 工程配置。

1. 更改至您的 PetaLinux 工程根目录中。

```
$ cd <plnx-proj-root>
```

2. 启动顶层系统配置菜单，并按照自己的要求进行配置:

```
$ petalinux-config
```

3. 启动 Linux 内核配置菜单，并按照自己的要求进行配置:

```
$ petalinux-config -c kernel
```

4. 启动根文件系统配置菜单，并按照自己的要求进行配置：

```
$ petalinux-config -c rootfs
```

提示: 根据需要在 `petalinux-config` 菜单配置中为定制电路板设置 U-Boot 目标。根据需要设置 `$ petalinux-config Set MACHINE_NAME`。可能的值是 ac701-full、ac701-lite、kc705-full、kc705-lite、kcu105、zc1254-reva、zc1275-reva、zc1275-revb、zc1751-dc1、zc1751-dc2、zc702、zc706、avnet-ultra96-rev1、zcu100-reva、zcu100-revb、zcu100-revc、zcu102-rev1.0、zcu102-reva、zcu102-revb、zcu104-reva、zcu104-revc、zcu106-reva、zcu111-reva、zedboard、sp701-rev1.0、vcu118-rev2.0 和 zcu1285-reva。

注释: 请确保已将特定电路板和用户的 dtsi 条目添加到 `project-spec/meta-user/recipes-bsp/device-tree/files/system-user.dtsi`。

对于 zcu102 和 zcu106 电路板，使用模板流将以下行添加到 `<plnx-proj-root>/project-spec/meta-user/recipes-bsp/fsbl/fsbl_%.bba ppend` 以进行 FSBL 初始化。

```
YAML_COMPILER_FLAGS_append = " -DXPS_BOARD_ZCU102" #for zcu102
YAML_COMPILER_FLAGS_append = " -DXPS_BOARD_ZCU106" # for zcu106
```

PetaLinux 目前使系统自动操作，没有添加这些宏。

设备树设置

本节描述为设置设备树，修改哪些文件是安全的，以及如何将新信息添加到设备树中。

要求

本节假定您已使 PetaLinux 工具软件平台做好准备，以构建一个按照您的硬件平台自定义的 Linux 系统。如需了解更多信息，请参阅 导入硬件配置。必须掌握 DTS 语法知识才能自定义默认的 DTS。

设置设备树

用户可修改的 PetaLinux 设备树配置与以下 config 文件相关联，这些文件位于 `<plnx-projroot>/project-spec/meta-user/recipes-bsp/device-tree/files/`：

- `multi-arch/`
- `system-user.dtsi`
- `xen.dtsi`
- `zynqmp-qemu-arm.dts`
- `openamp.dtsi`
- `xen-qemu.dtsi`

生成的文件将在 `<plnx-projroot>/components/plnx_workspace/device-tree/device-tree/` 目录中。

注意! 上面提到的所有 dtsi 文件都是由该工具生成的。不建议编辑任何这些文件。

有关设备树文件的详细信息，请参阅 附录 B: PetaLinux 工程结构。

注意! 上列 DTSI 文件 `*.dtsi` 是自动生成的，不应该编辑。

如果希望添加信息，如以太网 PHY 信息，则应将其包含在 system-user.dtsi 文件中。在此情况下，设备树应该包含与特定平台相关的信息，因为信息（这里是以太网 PHY 信息）是电路板层面的，因电路板而异。

注释: 有必要进行这种手工交互是因为一些信息是“电路板层面的”，工具没有办法预测这里应该是什么。有关各个器件绑定的详细信息，请参考 Linux 内核设备树绑定文档（来自内核源根的 Documentation/devicetree/bindings）。

为 system-user.dtsi 提供的格式良好的设备树节点示例如下图所示：

```
/dts-v1/;
/include/ "system-conf.dtsi"
/ {
};
&gem0 {
    phy-handle = <&phy0>;
    ps7_ethernet_0_mdio: mdio {
        phy0: phy@7 {
            compatible = "marvell,88e1116r";
            device_type = "ethernet-phy";
            reg = <7>;
        };
    };
};
```

重要提示! 确保设备树节点名、MDIO 地址和兼容字符串与特定系统中使用的命名约定相对应。

下面的示例演示如何添加 sample-user-1.dtsi 文件：

1. 添加 /include/ “system-user-1.dtsi 到 project-spec/meta-user/recipes-bsp/device-tree/files/system-user.dtsi。该文件看起来应该像下面这样：

   ```
   /include/ "system-conf.dtsi"
   /include/ "system-user-1.dtsi"
   / {
   };
   ```

2. 添加 file://system-user-1.dtsi 到 project-spec/meta-user/recipes-bsp/device-tree/device-tree.bbappend。该文件看起来应该像下面这样：

   ```
   FILESEXTRAPATHS_prepend := "${THISDIR}/files:"

   SRC_URI += "file://system-user.dtsi"
   SRC_URI += "file://system-user-1.dtsi"
   ```

   不建议更改 <plnx-proj-root>/components/plnx_workspace/device-tree/device-tree/ 中的任何内容。

   建议使用系统用户 DTSI 添加、修改和删除节点或值。最后添加系统用户 DTSI，使其中的值具有更高的优先级。

   通过在系统用户 DTSI 中定义，可以覆盖其他 DTSI 中的任何现有值。

U-Boot 配置

本节介绍了哪些文件对于 U-Boot 配置可进行安全修改并讨论有关 U-Boot CONFIG_ 选项/设置的有关内容。

要求

本节假定您已使 PetaLinux 工具软件平台做好准备，以构建一个按照您的硬件平台自定义的 Linux 系统。如需了解更多信息，请参阅 导入硬件配置 章节。

设置 U-Boot

通用引导加载程序 (U-Boot) 的配置通常使用 C 预处理器定义：

- 配置 _OPTIONS_：

  可以选择配置选项。选项名称开头是“CONFIG_”。

- 配置 _SETTINGS_：

  这取决于硬件等。名称开头是“CONFIG_SYS_”。

提示: 有关 CONFIG_ 选项/设置文档的详细说明和 U-Boot 的 README，请参阅 Denx U-Boot 指南。

PetaLinux U-Boot 配置与 config.cfg 和 platform-auto.h 配置文件相关联，这些文件位于 <plnxproj_root>/project-spec/ meta-plnx-generated/recipes-bsp/u-boot/configs 和 platform-top.h，而后者则位于 <plnxproj_root>/project-spec/meta-user/recipes-bsp/u-boot/files/。

如要设置 U-Boot 环境变量，请在 platform-auto.h.中编辑 CONFIG_EXTRA_ENV_SETTINGS 变量。请注意，platform-auto.h 会在每次运行 petalinux-config 时再生成。

注意! config.cfg 和 platform-auto.h 文件自动生成，请谨慎编辑。

PetaLinux 目前还没有自动执行 U-Boot 配置的 CONFIG_ 选项/设置。可以将这些 CONFIG_ 选项/设置添加到 platform-top.h 文件。

将 CONFIG_ 选项（如 NFIG_CMD_MEMTEST）添加到 platform-top.h 的步骤：

- 更改至您的 PetaLinux 工程根目录中。

```
$ cd <plnx-proj-root>
```

- 打开文件 platform-top.h

```
$ vi project-spec/meta-user/recipes-bsp/u-boot/files/platform-top.h
```

- 如果想添加 CONFIG_CMD_MEMTEST 选项，请将以下行添加到文件中。保存更改。

```
#define CONFIG_CMD_MEMTEST
```

提示: 定义 CONFIG_CMD_MEMTEST 启用监控命令“mtest”，该命令用于简单的 RAM 测试。

- 构建 U-Boot 镜像。

```
$ petalinux-build -c u-boot
```

- 使用下列命令生成 BOOT.BIN：

```
$ petalinux-package --boot --fsbl <FSBL image> --fpga <FPGA bitstream> --
u-boot
```

- 启动硬件或 QEMU 上的镜像，并在 U-Boot 阶段停止。
- 在 U-Boot 控制台中输入 `mtest` 命令如下：

```
ZynqMP mtest
```

- U-Boot 控制台上的输出应该类似如下：

```
Testing 00000000 ... 00001000:
                        Pattern 00000000 Writing... Reading...Iteration:
20369
```

重要提示! 如果 `CONFIG_CMD_MEMTEST` 未定义，则 U-Boot 控制台上的输出将如下：

```
U-Boot-PetaLinux> mtest Unknown command ' mtest'  - try ' help'
```

如需了解更多有关 U-Boot 的信息，请参阅 https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842223/U-boot。

XILINX

第 11 章

Yocto 功能

SDK 生成（目标系统引导生成）

OpenEmbedded 构建系统使用 BitBake 来生成软件开发套件 (SDK) 安装程序脚本标准 SDK。PetaLinux 构建和安装 SDK。安装的 SDK 可作为应用程序开发的 sysroot（系统引导）使用。

这并非指赛灵思 SDK。

构建 SDK

以下命令构建 SDK 并将其复制到 `<proj_root>/images/linux/sdk.sh`。

```
petalinux-build --sdk
```

下面是等效的 BitBake 命令。

```
bitbake petalinux-user-image -c do_populate_sdk
```

安装 SDK

生成的 SDK 必须安装/提取到一个目录中。以下命令将 SDK 提取到指定的目录中。默认 SDK 是 `<proj_proot>/images/linux/sdk.sh`，默认的安装目录是 `<proj_proot>/images/linux/sdk/`。

```
petalinux-package --sysroot -s|--sdk <custom sdk path> -d|--dir <custom
directory path>
```

示例

1. 添加交叉编译 qt 工具链

   若要利用 qt 工具链构建 SDK：

   a. 创建 `<proj-root>/project-spec/meta-user/recipes-core/images/petalinux-user-image.bbappend` 文件。

   b. 在新建文件中添加 `inherit populate_sdk_qt5`。

   c. 运行 `petalinux-config -c rootfs` 并选择“packagegroup-petalinux-qt”。

   d. 运行 `petalinux-build -s`。

   e. 运行 `petalinux-package --sysroot`。

若要验证：

a. 打开新的终端。

b. 转到 `<plnx-proj>/image/linux/sdk`。

c. 运行 `source environment-setup-aarch64-xilinx-linux`。

d. 运行 `which qmake`。这即确认 qmake 是否来自 SDK。

2. 构建 OpenCV 应用

a. 创建 PetaLinux 工程。

b. 在 RootFS 菜单配置中添加 `packagegroup-petalinux-opencv`。

c. 构建 SDK

```
petalinux-build --sdk
```

该命令可构建 SDK 并将其部署在 `<proj_root>/images/linux/sdk.sh`。

d. 安装 SDK

```
petalinux-package --sysroot
```

该命令可将 SDK 安装在 `<proj_root>/images/linux/sdk`。

e. 使用 `images/linux/sdk` 目录作为系统根，用于构建 OpenCV 应用程序。

访问工程中的 BitBake

BitBake 只在 bash shell 中可用。

为 Zynq UltraScale+ MPSoC 获取 BitBake 实用程序的步骤

1. 在创建工程之后至少运行一次 `petalinux-config` 或 `petalinux-config --oldconfig` 或 `petalinux-config --silentconfig`，以便建立所需的环境。

2. 确定 PetaLinux 工具脚本的源：

```
source /opt/pkg/petalinux/settings.sh
```

3. 确定 Yocto e-SDK 源：

```
source /opt/pkg/petalinux/components/yocto/source/aarch64/env ironment-
setup
 -aarch64-xilinx-linux
```

4. 确定环境建立脚本的源：

```
source /opt/pkg/petalinux/components/yocto/source/aarch64/layers/core/
 oe-init-build-env
```

在上述步骤之后，您将被引导至构建目录。保持在构建目录中，以便运行 BitBake。

5. 导出 XSCT：

```
export PATH=/opt/pkg/petalinux/tools/hsm/bin:$PATH
```

6. 解析配方的 PetaLinux 变量：

```
export BB_ENV_EXTRAWHITE="$BB_ENV_EXTRAWHITE PETALINUX"
```

7. 若要测试 BitBake 是否可用，运行：

```
bitbake strace
```

生成的镜像将被放在部署目录中。您必须将生成的镜像拷贝到 `<plnx-proj-root>/images/linux` 目录中，以便与其他命令一起操作。

共享 sstate-cache

Yocto e-SDK 含有最小的共享 sstate 高速缓存。赛灵思托管完整的 `petalinux-image` sstate 高速缓存，位置为 http://petalinux.xilinx.com/sswreleases/rel-v2019.1/。

在 `petalinux-build` 过程中，BitBake 将在 PetaLinux 工具中搜索 sstate 高速缓存，这是最低设置。如果在该位置找不到 sstate 高速缓存，BitBake 则在 http://petalinux.xilinx.com/sswreleases/rel-v2019.1/ 中搜索该高速缓存。如果仍未找到 sstate 高速缓存，BitBake 将重新构建，sstate 被签字锁定。

对于您为任何 RootFS 组件创建的 `.bbappend` 文件，您必须在 `<plnx proj root>/project-spec/meta_user/conf/petalinuxbsp.conf` 中添加 `SIGGEN_UNLOCKED_RECIPES += "<component>"`。

下载镜像

赛灵思在 https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/embedded-design-tools.html c tar 文件。默认情况下，这个 URL 被添加到 `petalinux-config` 中的 Yocto 源镜像。

如果从头开始重建任何组件，BitBake 首先在预镜像（pre-mirror，即工具下载页面）中搜索组件源，然后在 petalinux.xilinx.com 下载镜像 URL 中搜索。随后，在 SRC_URI 中搜索下载该组件源的配方。

可以通过将 `SOURCE_MIRROR_URL += file:///home/you/your-download-dir/` 添加到 `<proj-root>/project-spec/meta-user/conf/petalinuxbsp.conf` 来添加更多镜像。

机器支持

Yocto 机器指定为其构建镜像的目标器件。该变量对应于同名的机器配置文件，通过该文件设定机器特定的设置。目前，PetaLinux 支持用户机器配置文件。

可以在 `<proj_root>/project-spec/meta-user/conf/machine/` 下添加自己的机器配置文件，也可以在任何附加层中添加机器配置文件，并通过 `petalinux-config` 将其添加到工程中。

通过以下步骤指定 PetaLinux 工程中的用户机器配置文件名：

1. 进入 PetaLinux 工程。
2. 选择“petalinux-config → Yocto settings → () MACHINE NAME”。

3. 指定机器配置文件名。

现在使用 meta-xilinx 机器更新 BSP。

表 24: **模板的机器名称更改**

模板	机器
zynq	plnx-zynq7
zynqmp	plnx-zynqmp
microblaze	plnx-microblazeel

表 25: **BSP 的机器名称更改**

BSP	机器
zc702	zc702-zynq7
zc706	zc706-zynq7
zcu102 (所有变体)	zcu102-zynqmp
zcu106	zcu106-zynqmp
zcu104	zcu104-zynqmp
kc705	plnx-microblazeel
ac701	plnx-microblazeel
kcu105	plnx-microblazeel
zcu111	zcu111-zynqmp
zcu1285	zcu1285-reva
zc1275	zc1275-revb
sp701	sp701-rev1.0
vcu118	vcu118-rev2.0

SoC 变体支持

赛灵思为每个 SoC 产品提供多个器件。Zynq® UltraScale+™ MPSoC 提供三种器件变体。如需了解更多信息请参阅此处。Zynq-7000 器件提供两个变体。如需了解更多信息请参阅此处。

SOC_VARIANT 可以扩展 ${SOC_FAMILY}${SOC_VARIANT} 覆盖。它进一步扩展 SoC 上的组件覆盖。（例如 mali、vcu）。这使得重用组件覆盖取决于 SoC。该功能主要用于当硬件设计具有相应的 IP （VCU 或 USP）时自动切换到硬件加速。赛灵思分销 SoC 的多个变体，如下所示。

1. Zynq-7000 器件提供 Zynq7000zs 和 Zynq7000z 变体。可用的 SOC_VARIANT 包括：
   - "7zs" - Zynq-7000 单个 A9 核

     "7z" - Zynq-7000 双 A9 核
   - Zynq-7000 器件的默认 SOC_VARIANT 为“7z”。对于 7000zs 器件，将 SOC_VARIANT = "7zs" 添加到 `petalinuxbsp.conf`

   Zynq-7000 器件没有其他覆盖。对于 EG 和 EV 器件，添加 mali440 覆盖。是否添加 VCU 覆盖取决于硬件设计中的 VCU IP。

2. Zynq UltraScale+ MPSoC 提供三种器件变体。可用的 SOC_VARIANT 包括：
   - "cg" - Zynq UltraScale+ MPSoC CG 器件
   - "eg" - Zynq UltraScale+ MPSoC EG 器件
   - "ev" - Zynq UltraScale+ MPSoC EV 器件
   - "dr" - RFSoC 器件

   默认值为 "eg"。PetaLinux 根据 HDF 中 IP 的存在自动分配“ev”和“dr”。

注释: 必须在 `petalinuxbsp.conf` 中为“cg”器件明确设置 SOC_VARIANT = "cg"。

镜像功能

由 OpenEmbedded 构建系统生成的镜像内容可利用您一般在镜像配方中配置的 IMAGE_FEATURES 和 EXTRA_IMAGE_FEATURES 变量予以控制。通过这些变量，您可以添加若干不同预定义的封装，比如开发实用程序或带有调查应用程序问题或剖析应用程序所需的调试信息的封装。

若要移除任何默认功能，在 `petalinuxbsp.conf` 中添加下列代码：

```
IMAGE_FEATURES_remove = "ssh-server-dropbear"
```

若要添加任何新的功能，在 `petalinuxbsp.conf` 中添加下列命令：

```
IMAGE_FEATURES_append = "  myfeature"
```

XILINX

第 12 章

技术 FAQ

故障排除

本节详细介绍使用 PetaLinux 命令时出现的常见错误，并详细列出恢复步骤。

有关补丁和 Yocto 的信息，请参阅 https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842475/PetaLinux+Yocto+Tips。

NFS 上的 TMPDIR

显示的错误是：

```
“ERROR: OE-core's config sanity checker detected a potential
misconfiguration”. Either fix the cause of this error or disable the
checker at your own risk (see sanity.conf). For the list of potential
problems or advisories.
```

TMPDIR：`/home/user/xilinx-kc705-axi-full-2019.1/build/tmp` 不能位于 NFS 上。

当 TMPDIR 在 NFS 上时，BitBake 在解析时将出现错误。您必须从 `petalinux-config` 中更改，然后提供任何本地存储。若要这么做，请选择“Yocto-settings → TMPDIR”。

切勿为两个不同的 PetaLinux 工程配置相同的 TMPDIR。这可造成构建错误。

含有“_”的配方名称

如果应用程序名称是 `plnx_myapp`，BitBake 则发生错误。必须在“_”之后输入版本号。

例如，`myapp_1` 是一个准确的应用程序/模块名称。

若要恢复，您必须 sstateclean 清理创建的应用程序，然后将其删除。此外，删除 `<plnx-proj-root>/project-spec/meta-user/recipes-core/images/petalinux-image.bbappend` 中的程序行。

```
IMAGE_INSTALL_append = " plnx_myapp"
```

从崩溃的终端中恢复

当按下两次 Ctrl+C 键，使 PetaLinux 强制退出时，会出现以下错误：

```
NOTE: Sending SIGTERM to remaining 1 tasks
Error in atexit._run_exitfuncs:
Traceback (most recent call last):
  File
"/opt/pkg/petalinux/components/yocto/source/aarch64/layers/core/
bitbake/lib/bb/ui/k
notty.py", line 313, in finish
    self.termios.tcsetattr(fd, self.termios.TCSADRAIN, self.stdinbackup)
termios.error: (5, 'Input/output error')
```

在该错误之后，控制台即崩溃，您无法看到刚键入的文本。若要恢复控制台，请输入 `stty sane`，然后按下两次 Ctrl+J 键。

Python 语言设置

语言设置缺失时会出现以下错误：

- 错误消息“`Could not find the /log/cooker/plnx_microblaze in the /tmp directory`”会出现在 `petalinux-config` 期间。
- 请使用支持 UTF-8 的区域设置（如 LANG=en_US.UTF-8）。Python 无法在加载后更改文件系统区域设置，因此，在 Python 启动时需要 UTF-8，否则将无法工作。

  ```
  ERROR: Failed to build project
  ```

  .
- 如要解决上述错误，请设置如下：

  ```
  export LC_ALL=en_US.UTF-8
  export LANG=en_US.UTF-8
  export LANGUAGE=en_US.UTF-8
  ```

内核和 U-Boot 的 Menuconfig 挂起

对于 `petalinux-config -c`，当内核和 U-Boot BitBake 试图在里面打开一个新终端时，有时会失败。下面是可能的错误消息。

1. `ERROR: Unable to spawn new terminal`
2. `ERROR: Continuing the execution without opening the terminal`

解决方案可以是：

1. 使用 `ssh -X <hostname>`。
2. 在 `<plnx-proj-root>/project-spec/meta-user/conf/petalinuxbsp.conf` 中取消 OE_TERMINAL 行的批注。可以设置任何适合的终端。有关更多详细信息，请参见 第 11 章: Yocto 功能。您必须更改 OE_TERMINAL，因为它无法通过缺省值。在 `<plnx-proj-root>/project-spec/meta-user/conf/petalinuxbsp.conf` 中取消 OE_TERMINAL 的批注，并将其设置为 xterm 或 screen。为此，需要在 PC 中安装相应的实用程序。

外部源设置

cfg 或 scc 文件不会与 Yocto 流程中的外部源一起应用（上游行为）。PetaLinux 需要应用配置方可处理外部源；它只有处理 cfg 文件的能力。因此，总是建议使用 cfg 文件，而非 scc 文件。

Xen 和 openamp 通过分布功能来处理。添加分布功能不能像在 scc 文件中处理的方式那样在内核中启用相应的配置。解决方案就是编辑 `<plnx-project-root>/project-spec/meta-user/recipes-kernel/linux/linux-xlnx_%.bbappend`。

添加以下行：

```
SRC_URI += "file://xilinx-kmeta/bsp/xilinx/xen.cfg"
```

在使用外部源方法时，必须使用相应的 cfg 文件替换所有 scc 文件。

错误消息“do_image_cpio: Function Failed”

CPIO 格式不支持 2 GB 以上的大小。因此，不能将 INITRAMFS 用于更大的尺寸。下面步骤描述较大镜像尺寸（2 GB 以上）的处理过程。

1. 将 RootFS 类型更改为 SD 卡。

   ```
   $ petalinux-config
   ```

   选择 “Image Packaging Configuration → Root filesystem type → SD card”

2. 在 `<proj-root>/project-spec/meta-user/conf/petalinuxbsp.conf` 中添加以下行。

   ```
   IMAGE_FSTYPES_remove = "cpio cpio.gz cpio.bz2 cpio.xz cpio.lzma cpio.lz4
   cpio.gz.u-boot"
   IMAGE_FSTYPES_DEBUGFS_remove = "cpio cpio.gz cpio.bz2 cpio.xz cpio.lzma
   cpio.lz4
   cpio.gz.u-boot"
   ```

3. 构建工程。

   ```
   $ petalinux-build
   ```

   注释: 与之前不同，目前 PetaLinux 不生成全局 DTS 文件。请使用以下命令生成全局 DTS 文件：

   ```
   dtc -I dtb -O dts -o system.dts system.dtb
   ```

注意! 对于任何构建操作，不要使用指向工程目录的符号链接路径，包括简单地“cd”到目录中。

封装管理

PetaLinux 支持 Zynq-7000 器件的 DNF 封装管理系统。利用以下步骤配置和使用封装管理系统：

1. 通过 `petalinux-config -c rootfs` 启用 DNF。在镜像功能项下启用“package management”，在文件系统封装/基础项下启用 “dnf”。
2. 构建 SDK 工程。

   ```
   #petalinux-build
   ```

3. 在 SD 或 JTAG 启动模式中启动 Linux。

4. 在 `/etc/yum.repos.d/` 目录中创建 `oe-remote-repo-sswreleases-rel-v2019.1-feeds-ultra96-zynqmp.repo`，从而在赛思灵平台上的目标上建立 `.repo` 文件。

```
[oe-remote-repo-sswreleases-rel-v2019.1-feeds-ultra96-zynqmp]
name=OE Remote Repo: sswreleases rel-v2019.1 feeds ultra96-zynqmp
baseurl=http://petalinux.xilinx.com/sswreleases/rel-v2019.1/feeds/
ultra96-zynqmp
gpgcheck=0
```

5. 列出所有可用封装。

```
#dnf repoquery
```

6. 安装某个具体封装。

```
#dnf install <pkg name>
```

示例：`#dnf install packagegroup-petalinux-matchbox`

一旦安装了 matchbox 封装，重启目标，然后您就能获得桌面环境。

在 Zynq-7000 器件和 Zynq UltraScale+ MPSoC 中有大型 INITRAMFS 镜像时 Linux 启动挂机

当发出 `petalinux-boot` 命令时，会显示以下警告消息：

```
"Linux image size is large (${imgsize}). It can cause boot issues. Please
refer to Technical FAQs. Storage based RootFilesystem is recommended for
large images."
```

如果您的 INITRAMFS 镜像尺寸很大，请使用基于存储的启动。

移植

本节描述当前版本与前一版本的移植详情。

工具目录结构

工具目录结构如下：

- `<path-to-installed-petalinux>/tools/linux-i386/petalinux` 已迁往 `<path-to-installed-petalinux>/tools/xsct/petalinux`。
- 因为 PetaLinux 工具没有使用可用的工具链，`<path-to-installed-petalinux>/tools/linux-i386`，它已经被删除。
- `<path-to-installed-petalinux>/tools/xsct/SDK/2018.3` 已迁往 `<path-to-installed-petalinux>/tools/xsct`。

DT 覆盖支持

- `<plnx-proj>/images/linux/` 中的比特流文件名是 `system.bit`，但是，如果您启用了 DT 覆盖支持，它将使用设计名称。
- 已为 Zynq®-7000 器件添加了 DT 覆盖支持。

更改构建

`petalinux-build` 将为 Zynq®-7000 器件和 MicroBlaze™ 处理器生成内核镜像。已移除了 Zynq® UltraScale+™ MPSoC 内核镜像支持。

XILINX

附录 B

PetaLinux 工程结构

本节将简要介绍 PetaLinux 工程的文件和目录结构。PetaLinux 工程一次支持一个 Linux 系统的开发。构建的 Linux 系统由以下组件组成：

- 设备树
- 第一阶段启动加载器（可选）
- U-Boot
- Linux 内核
- RootFS 由以下组件组成：
  - 预构建软件包
  - Linux 用户应用（可选）
  - 用户模块（可选）

PetaLinux 工程目录包含工程的配置文件、Linux 子系统和子系统的组件。`petalinux-build` 命令用那些配置文件来构建工程。可以运行 `petalinux-config` 来修改这些文件。以下是 PetaLinux 工程示例：

```
project-spec
   meta-plnx-generated
   hw-description
   configs
   meta-user
pre-built
   linux
      implementation
      images
hardware
   xilinx-zcu104-2019.1
components
   plnx_workspace
      device-tree
config.project
README
README.hw
```

表 26: **PetaLinux 工程描述**

PetaLinux 工程中的文件/目录	描述
`/.petalinux/`	保存工具使用情况和 WebTalk 数据的目录。
`/config.project/`	工程配置文件。
`/project-spec`	工程规格。
`/project-spec/hw-description`	从 Vivado® 设计工具导入的硬件描述。
`/project-spec/configs`	顶层 config 和 RootFS config 的配置文件

表 26: **PetaLinux 工程描述** (续)

PetaLinux 工程中的文件/目录	描述
/project-spec/configs/config	用于保存用户设置的配置文件
/project-spec/configs/rootfs_config	用于根文件系统的配置文件。
/components/plnx_workspace/device-tree/device-tree/	用于构建设备树的设备树文件。以下文件由 petalinux-config 自动生成： • skeleton.dtsi （仅限 Zynq-7000 器件） • zynq-7000.dtsi （仅限 Zynq-7000 器件） • zynqmp.dtsi （仅限 Zynq UltraScale+ MPSoC） • pcw.dtsi （仅限 Zynq-7000 器件和 Zynq UltraScale+ MPSoC） • pl.dtsi • system-conf.dtsi • system-top.dts • <bsp name>.dtsi 不建议编辑这些文件，因为这些文件是由工具重新生成的。
/project-spec/meta-user/recipes-bsp/device-tree/files/	system-user.dtsi 没有被任何 PetaLinux 工具所修改。此文件可以与版本控制系统一起安全使用。此外，可以将自己的 DTSI 文件添加到此目录。必须通过添加自己的 DTSI 文件来编辑 <plnx-proj-root>/project-spec/meta-user/recipes-bsp/device-tree/device-tree.bbappend。
/project-spec/meta-plnx-generated/recipes-bsp/u-boot/configs	U-Boot PetaLinux 配置文件。以下文件由 petalinux-config 自动生成： config.mk 仅用于 MicroBlaze 处理器 platform-auto.h config.cfg platform-top.h 不会被任何 PetaLinux 工具所修改。U-Boot 构建时，这些文件将复制到 U-Boot 构建目录 build/linux/u-boot/src/<U_BOOT_SRC>/，如下所示： config 是 U-Boot kconfig 文件。 config.mk 被复制到 board/xilinx/microblaze-generic/，用于 MicroBlaze。
/project-spec/meta-user/recipes-bsp/u-boot/files/platform-top.h	platform-auto.h and platform-top.h 被复制到 include/configs/ 目录。
/components/	嵌入式软件工作空间的目录，和封装 BSP 时存放外部源的位置。还可以手动将组件复制到此目录中。下面是放置外部组件的规则：<plnx-proj-root>/components/ext_source/<COMPONENT>
/project-spec/meta-user/conf/petalinuxbsp.conf	此配置文件包含构建环境的所有本地用户配置。这用来替代 Yocto 元层中的“local.conf”。

注释:

1. 所有路径都相对于 <plnx-projroot>。

工程构建后，将自动生成三个目录：

- <plnx-proj-root>/build 用于构建生成的文件。
- <plnx-proj-root>/images 用于可启动镜像。

- `<plnx-proj-root>/build/tmp` 用于 Yocto 生成的文件。此目录可通过 `petalinux-config` 设置。

下面给出一个实例：

```
<plnx-proj-root>
 -build
 -bitbake.lock
 -build.log
 -config.log
 -cache/
 -conf/
 -downloads/
 -misc/
 -config/
 -plnx-generated/
 -rootfs_config/
 -sstate-cache/
 -tmp/
 -components
 -plnx_workspace/
 -config.project
 -hardware
 -images
 -linux/
 -pre-built
 -linux/
 -project-spec
 -attributes
 -configs/
 -config
 -rootfs_config
 -hw-description/
 -meta-plnx-generated/
 -meta-user/
```

注释: `<plnx-proj-root>/build/` 自动生成。不要手动编辑此目录中的文件。运行 `petalinux-config` 或 `petalinux-build` 时此目录中的内容将更新。`<plnx-proj-root>/images/` 也将自动生成。运行 `petalinux-build` 时此目录中的文件将更新。

下表是 Zynq UltraScale+ MPSoC 示例。

默认情况下，`petalinux-build` 后，构件将被删除，以节省空间。如要保留构件，则必须将 `INHERIT += "rm_work"` 从 `build/conf/local.conf` 中删除，但是这会增加工程空间。

表 27: **PetaLinux 工程中的构建目录**

PetaLinux 工程中的构建目录	描述
`<plnx-proj-root>/build/build.log`	构建日志文件
`<plnx-proj-root>/build/misc/config/`	存放与 Linux 子系统构建相关文件的目录
`<plnx-proj-root>/build/misc/rootfs_config/`	存放与 RootFS 构建相关文件的目录
`${TMPDIR}/work/plnx_aarch64-xilinx-linux/petalinux-ser-image/1.0-r0/rootfs`	目标的 RootFS 副本。这是个暂存目录。
`${TMPDIR}/plnx_aarch64`	存放构建用户应用/库所需的库和报头文件的暂存目录
`${TMPDIR}/work/plnx_aarch64-xilinx-linux/linux-xlnx`	存放与内核构建相关文件的目录
`${TMPDIR}/work/plnx_aarch64-xilinx-linux/u-boot-xlnx`	存放与 U-Boot 构建相关文件的目录

表 27: **PetaLinux 工程中的构建目录** (续)

PetaLinux 工程中的构建目录	描述
`<plnx-proj-root>/components/plnx_workspace/device-tree/device-tree"`	存放与设备树构建相关文件的目录

表 28: **PetaLinux 工程中的镜像目录**

PetaLinux 工程中的镜像目录	描述
`<plnx-proj-root>/images/linux/`	存放 Linux 子系统可启动镜像的目录

工程层级

PetaLinux 工程在 `<proj-plnx-root>/project-spec` 下有以下两个层级

1. meta-plnx-generated

该层级保持了所有组件的所有 bbappends 和配置片段 (cfg)。该层级中的所有文件都是由工具根据 HDF/DSA 和用户配置创建的。在该层级中的文件不得予以手动更新，因为它是为 `petalinux-config` 和 `petalinux-build` 命令重新生成的。

2. meta-user

该层是所有针对具体用户更改的占位符。您可以在该层中添加您自己的 bbappend 和配置。

XILINX

附录 C

生成启动组件

第一阶段启动加载器 (FSBL)

默认情况下，顶层系统设置被设定为生成第一阶段启动加载器 (FSBL)。这是可选项。

注释: 如果不希望 PetaLinux 构建 FSBL/FS-BOOT，则需要自己手动构建。否则，系统将无法正常启动。

如果之前从 menuconfig 禁用了第一阶段启动加载器，则可以将工程设置为构建第一阶段启动加载器，如下所示：

1. 启动顶层系统设置配置菜单并配置：

```
$ petalinux-config
```

   a. 选择“Linux Components Selection --->”子菜单。

   b. 选择“First Stage Boot Loader”选项。

```
[*] First Stage Bootloader
```

   c. 退出菜单并保存更改。

2. 启动 `petalinux-build` 构建 FSBL：

构建工程时构建 FSBL：

```
$ petalinux-build
```

仅构建 FSBL：

```
$ petalinux-build -c fsbl (for MicroBlaze, it is fs-boot)
```

引导加载程序 ELF 文件将作为 `zynqmp_fsbl.elf` （Zynq® UltraScale+™ MPSoC）、`zynq_fsbl.elf` （Zynq®-7000 器件）和 `fs-boot.elf`（MicroBlaze™ 处理器）安装在工程根目录内的 `images/linux` 中。

如需了解更多有关 FSBL 的信息，请参阅 https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842019/FSBL。

Arm 可信固件 (ATF)

此项用于 Zynq® UltraScale+™ MPSoC。这是强制要求。默认情况下，顶层系统设置被设置为生成 ATF。

您可以按如下方式设置 ATF 可配置选项：

1. 启动顶层系统设置配置菜单并配置：

```
$ petalinux-config
```

   a. 选择“Arm Trusted Firmware Compilation Configuration --->”子菜单。
   b. 输入您的设置。
   c. 退出菜单并保存更改。

2. 在构建工程时构建 ATF：

```
$ petalinux-build
```

   只构建 ATF：

```
$ petalinux-build -c arm-trusted-firmware
```

   ATF ELF 文件将作为 Zynq UltraScale+ MPSoC 的 `bl31.elf` 安装在工程根目录内的 `images/linux` 中。

如需了解更多有关 ATF 的信息，请参见 https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842107/Arm+Trusted+Firmware。

PMU 固件

这仅用于 Zynq® UltraScale+™ MPSoC。这是可选项。默认情况下，顶层系统设置被设定为生成 PMU 固件。

注意! 如果用户不希望 PetaLinux 构建 PMU 固件，则需要自己手动构建。否则，系统将无法正常启动。

您可以设置工程构建 PMU 固件如下：

1. 启动顶层系统设置配置菜单并配置：

```
$ petalinux-config
```

   a. 选择“Linux Components Selection”。
   b. 选择“PMU Firmware”选项。

```
[*] PMU Firmware
```

   c. 退出菜单并保存更改。

2. 构建工程时构建 PMU 固件：

```
$ petalinux-build
```

   仅构建 PMU 固件：

```
$ petalinux-build -c pmufw
```

   PMU 固件 ELF 文件将作为 Zynq UltraScale+ MPSoC 的 `pmufw.elf` 安装于工程根目录内的 `images/linux` 中。

   如需了解更多有关 PMU 固件的信息，请参阅 http://www.wiki.xilinx.com%5CPMU+Firmware。

仅用于 MicroBlaze 平台的 FS-Boot

PetaLinux 中的 FS-Boot 是仅用于 MicroBlaze™ 平台的第一阶段引导加载程序演示。其目的是演示如何从闪存向内存加载镜像并跳转到其中。如果要试用 FS-Boot，至少需要 8 KB 区块 RAM。

FS-Boot 仅支持标准 SPI 模式和 Quad SPI 模式的并列式闪存和 SPI 闪存。

为了让 FS-Boot 知道在闪存中的什么位置获取镜像，需要定义宏 `CONFIG_FS_BOOT_START`。这将由 PetaLinux 工具完成。PetaLinux 工具在菜单配置主闪存分区表设置中的 `boot` 分区设置中自动设置该宏。对于并列式闪存，这是引导分区的开始地址。对于 SPI 闪存，这是 `boot` 分区的开始偏移地址。

闪存中的镜像要求 BIN 文件之前要有封装报头。FS-Boot 从封装中获取目标内存位置。封装需要包含以下信息：

表 29: **封装信息**

偏移	描述	值
0×0	FS-Boot 可启动镜像幻码	0×b8b40008
0×4	BIN 镜像大小	用户定义
0×100	FS-Boot 可启动镜像目标内存地址	用户定义。PetaLinux 工具根据菜单配置中的内存设置中的 U-Boot 文本基地址偏移自动计算此地址。
0×10c	BIN 文件的起点	无

FS-Boot 忽略封装报头中的其他字段。PetaLinux 工具生成封装报头来封装 U-Boot BIN 文件。

XILINX

附录 D

QEMU 虚拟网络模式

QEMU 中有两种执行模式：非根（默认）和根模式（需要 sudo 或根权限）。模式之间的差异与虚拟网络配置有关。

在非根模式下，QEMU 设置一个内部虚拟网络，限制来自主机和客户机的网络流量。其工作原理类似于 NAT 路由器。除非重定向 tcp 端口，否则无法访问此网络。

在根模式下，QEMU 在虚拟以太网适配器上创建子网，并依赖于主机系统上的 DHCP 服务器。

以下几节将详细介绍如何使用这些模式，包括重定向非根模式，以便从本地主机访问。

非根模式下重定向端口

如果以默认的非根模式运行 QEMU，并且希望从主机访问内部（虚拟）网络（例如，使用 GDB 或 TCF 代理进行调试），则需要将仿真的系统端口从 QEMU 虚拟机内部转发到本地机器。`petalinux-boot --qemu` 命令使用 `--qemu-args` 选项执行此重定向。下表列出一些重定向实参示例。这是标准的 QEMU 功能，请参阅 QEMU 文档了解更多详情。

表 30: **重定向实参**

QEMU 选项开关	目的	从主机访问客户机
`-tftp <path-to-directory>`	在指定目录下设置一个 TFTP 服务器，该服务器在 QEMU 内部 IP 地址 10.0.2.2 上可用。	
`-redir tcp:10021:10.0.2.15:21`	将主机上的端口 10021 重定向到客户机上的端口 21 ftp)	host> ftp localhost 10021
`-redir tcp:10023:10.0.2.15:23`	将主机上的端口 10023 重定向到客户机上的端口 23 telnet)	host> telnet localhost 10023
`-redir tcp:10080:10.0.2.15:80`	将主机上的端口 10080 重定向到客户机上的端口 80 http)	在网络浏览器中输入 `http://localhost:10080`
`-redir tcp:10022:10.0.2.15:22`	将主机上的端口 10022 重定向到客户机上的端口 22 ssh)	在主机上运行 `ssh -P 10022 localhost` 以打开到目标的 SSH 会话

下面的例子显示了用于重定向端口的命令行：

```
$ petalinux-boot --qemu --kernel --qemu-args "-redir tcp:1534::1534"
```

本文档假设 gdbserver 和 tcp -agent 使用端口 1534，但是可以重定向到任何未使用的端口。内部仿真端口也可以不同于本地机器上的端口：

```
$ petalinux-boot --qemu --kernel --qemu-args "-redir tcp:1444::1534"
```

指定 QEMU 虚拟子网络

默认情况下，PetaLinux 使用 `192.168.10.*` 作为 `--root` 模式中的 QEMU 虚拟子网络。如果它被您的局域网或其他虚拟子网络占用，您可以使用另一个子网络。您可以在命令控制台上按如下方式运行 `petalinux-boot`，从而配置 PetaLinux 来使用其他子网络设置：

注释: 该功能需要具有在本地机器上的 sudo 访问权，而且必须与 `--root` 选项配合使用。

```
$ petalinux-boot --qemu --root --u-boot --subnet <subnet gateway IP>/
<number of the bits of the subnet mask>
```

例如，要使用子网络 `192.168.20.*`：

```
$ petalinux-boot --qemu --root --u-boot --subnet 192.168.20.0/24
```

QEMU 支持的赛灵思 IP 模型

随 PetaLinux 工具配备的 QEMU 仿真器支持以下赛灵思 IP 型号：

- Zynq® UltraScale+™ MPSoC Arm® Cortex™-A53 MPCore
- Zynq UltraScale+ MPSoC Cortex-R5F
- Zynq-7000 Arm Cortex-A9 CPU
- MicroBlaze™ CPU (小尾数 AXI)
- 赛灵思 Zynq-7000/Zynq UltraScale+ MPSoC DDR Memory Controller
- 赛灵思 Zynq UltraScale+ MPSoC DMA Controller
- 赛灵思 Zynq UltraScale+ MPSoC SD/SDIO Peripheral Controller
- 赛灵思 Zynq UltraScale+ MPSoC Gigabit Ethernet Controller
- 赛灵思 Zynq UltraScale+ MPSoC NAND Controller
- 赛灵思 Zynq UltraScale+ MPSoC UART Controller
- 赛灵思 Zynq UltraScale+ MPSoC QSPI Controller
- 赛灵思 Zynq UltraScale+ MPSoC I2C Controller
- 赛灵思 Zynq UltraScale+ MPSoC USB Controller（仅限主机支持）
- 赛灵思 Zynq-7000 Triple Timer Counter
- 赛灵思 Zynq-7000 DMA Controller
- 赛灵思 Zynq-7000 SD/SDIO Peripheral Controller
- 赛灵思 Zynq-7000 Gigabit Ethernet Controller
- 赛灵思 Zynq-7000 USB Controller（仅限主机支持）
- 赛灵思 Zynq-7000 UART Controller
- 赛灵思 Zynq-7000 SPI Controller
- 赛灵思 Zynq-7000 QSPI Controller
- 赛灵思 Zynq-7000 I2C Controller
- 赛灵思 AXI Timer 和 Interrupt Controller 外设
- 赛灵思 AXI External Memory Controller（连接到并行闪存）
- 赛灵思 AXI DMA Controller
- 赛灵思 AXI Ethernet
- 赛灵思 AXI Ethernet Lite
- 赛灵思 AXI UART 16650 和 Lite

注释: 默认情况下，QEMU 将禁用无型号信息的任何器件。出于此原因，可以使用 QEMU 来测试您自己自定义的 IP 核（除非您按照 QEMU 标准为其开发 C/C++ 型号）。

如需了解更多信息，请参阅《赛灵思快速仿真器用户指南：QEMU》 (UG1169)。

XILINX

附录 F

Xen Zynq UltraScale+ MPSoC 示例

本节详细介绍了 Xen Zynq® UltraScale+™ MPSoC 示例。它介绍了如何获取 Linux，以便在 Zynq UltraScale+ MPSoC 上的 Xen 之上作为 dom0 启动。

要求

本节假定已满足了以下要求：

- 您已使 PetaLinux 工具软件平台做好准备，以构建一个按照您的硬件平台自定义的 Linux 系统。如需了解更多信息，请参阅 导入硬件配置。
- 您已从 ZCU102 参考 BSP 中创建了一个 PetaLinux 工程。
- 在 `pre-built/linux/images` 目录中有与 Xen 相关的预建项，它们是 `xen.dtb`、`xen.ub`、`xen-Image` 和 `xen-rootfs.cpio.gz.u-boot`。

作为 dom0 启动预建 Linux

1. 将预建 Xen 镜像拷贝到您的 TFTP 目录，以便您可以利用 TFTP 从 U-Boot 中将其加载。

```
$ cd <plnx-proj-root>
$ cp pre-built/linux/images/xen.dtb <tftpboot>/
$ cp pre-built/linux/images/xen.ub <tftpboot>/
$ cp pre-built/linux/images/xen-Image <tftpboot>/
$ cp pre-built/linux/images/xen-rootfs.cpio.gz.u-boot <tftpboot>/
```

2. 利用 JTAG 启动在电路板上启动预建 U-Boot 镜像或从 SD 卡启动。

   注释: 对于 SD 卡启动，请参见 利用 SD 卡在硬件上启动 PetaLinux 镜像，对于 JTAG 启动，请参见 利用 JTAG 在硬件上启动 PetaLinux 镜像。

3. 从 U-Boot 中建立 TFTP 服务器 IP：

```
ZynqMP> setenv serverip <TFTP SERVERIP>
```

4. 加载 Xen 并从 U-Boot 中：

```
ZynqMP> tftpboot 1000000 xen.dtb
ZynqMP> tftpboot 80000 xen-Image
ZynqMP> tftpboot 1030000 xen.ub
ZynqMP> tftpboot 2000000 xen-rootfs.cpio.gz.u-boot
ZynqMP> bootm 1030000 2000000 1000000
```

提示: 对于 RootFS 镜像大小不同的预建镜像，必须调整上述地址，以便在将这些镜像拷贝到 RAM 时无覆盖。

重新构建 Xen

在为 Zynq UltraScale+ MPSoC 创建 PetaLinux 工程之后，按照以下步骤来构建 Xen 镜像：

1. 转到 `cd <proj root directory>`。
2. 在 `petalinux-config` 命令中，选择“Image Packaging Configuration → Root filesystem type (INITRD)”。
3. 在 `petalinux-config -c rootfs` 中，选择“PetaLinux Package Groups → Packagegroup-petalinux-xen → [*] packagegroup-petalinux-xen”。
4. 编辑设备树，以便在额外 Xen 相关配置中进行构建。编辑此文件：`project-spec/meta-user/recipes-bsp/device-tree/files/system-user.dtsi`，并添加此行：`/include/ "xen.dtsi"`

   它看起来应该像下面这样：

   ```
   /include/ "system-conf.dtsi"
   /include/ "xen.dtsi"
   / {
   };
   ```

5. 编辑此文件：`project-spec/meta-user/recipes-bsp/device-tree/device-tree.bbapp end`，并在其中添加此行：`SRC_URI += "file://xen.dtsi"`

   该文件看起来应该像下面这样：

   ```
   FILESEXTRAPATHS_prepend := "${THISDIR}/files:"

   SRC_URI += "file://system-user.dtsi"
   SRC_URI += "file://xen.dtsi"
   ```

6. 运行 `petalinux-build`: `$ petalinux-build`。
7. 构建伪像将位于工程目录中的 `images/linux` 中。

注释: 默认情况下，`petalinux-build` 命令不构建 Xen。默认的根文件系统不含 Xen 工具。您必须使用 Xen RootFS。

重要提示! 您必须根据镜像/RootFS 大小来更新 `xen.dtsi` 文件中的 dom0 内存。此外，根据不重叠情况下的镜像/RootFS 大小调整上述加载地址。

作为 dom0 启动构建的 Linux

1. 将构建的 Xen 镜像拷贝到您的 TFTP 目录，以便您可以利用 TFTP 从 U-Boot 中将其加载。

   ```
   $ cd <plnx-proj-root>
   $ cp images/linux/system.dtb <tftpboot>/
   $ cp images/linux/xen.ub <tftpboot>/
   $ cp  images/linux/Image <tftpboot>/
   $ cp  images/linux/rootfs.cpio.gz.u-boot <tftpboot>/
   ```

2. 利用 JTAG 启动或从 SD 卡启动在电路板上启动构建 U-Boot 镜像。

   注释: 对于 SD 卡启动，请参见 利用 SD 卡在硬件上启动 PetaLinux 镜像，对于 JTAG 启动，请参见 利用 JTAG 在硬件上启动 PetaLinux 镜像。

3. 从 U-Boot 中建立 TFTP 服务器 IP：

   ```
   ZynqMP> setenv serverip <TFTP SERVERIP>
   ```

4. 从 U-Boot 中加载 Xen 镜像：

```
ZynqMP> tftpboot 1000000 xen.dtb
ZynqMP> tftpboot 80000 xen-Image
ZynqMP> tftpboot 1030000 xen.ub
ZynqMP> tftpboot 2000000 xen-rootfs.cpio.gz.u-boot
ZynqMP> bootm 1030000 2000000 1000000
```

注释: Xen 的加载地址仅是几个代表，有关具体步骤和加载地址，请参见 http://www.wiki.xilinx.com/XEN+Hypervisor。

执行 OpenAMP

通过以下步骤执行 OpenAMP：

1. 启动内核：

```
$ cd <plnx-proj-root>
$ cp pre-built/linux/images/openamp.dtb pre-built/linux/images/system.dtb
$ petalinux-boot --jtag --prebuilt 3 --hw_server-url <hostname:3121>
```

2. 如要加载任何固件并运行任何测试应用：

```
$ echo <echo_test_firmware> > /sys/class/remoteproc/remoteproc0/firmware
```

例如，如要加载 image_echo_test：

```
$ echo image_echo_test >  /sys/class/remoteproc/remoteproc0/firmware
$ echo start > /sys/class/remoteproc/remoteproc0/state
$ modprobe rpmsg_user_dev_driver
$ echo_test
```

3. 如要卸载应用：

```
modprobe -r rpmsg_user_dev_driver echo stop > /sys/class/remoteproc/
remoteproc0/state
```

如需了解更多有关 OpenAMP 的信息，请参阅《Zynq 器件的 Libmetal 和 OpenAMP 用户指南》 (UG1186)。

附录 G

附加资源与法律提示

赛灵思资源

如需了解答复记录、技术文档、下载以及论坛等支持性资源，请参阅赛灵思技术支持。

参考资料

以下技术文档是本指南非常实用的补充材料：

1. PetaLinux 文档 (china.xilinx.com/petalinux)
2. 赛灵思答复记录 (55776)
3. 《Zynq UltraScale+ MPSoC：软件开发指南》 (UG1137)
4. 《PetaLinux 工具文档：PetaLinux 命令行参考》 (UG1157)
5. 赛灵思快速仿真器用户指南 (QEMU) (UG1169)
6. 《Zynq 器件的 Libmetal 和 OpenAMP 用户指南》 (UG1186)
7. www.wiki.xilinx.com/Linux
8. PetaLinux Yocto 提示
9. Yocto Project 技术 FAQ

Documentation Navigator 与设计中心

赛灵思 Documentation Navigator (DocNav) 提供了访问赛灵思文档、视频和支持资源的渠道，您可以在其中筛选搜索信息。打开 DocNav 的方法：

- 在 Vivado® IDE 中，单击“Help → Documentation and Tutorials”。
- 在 Windows 中，单击“Start → All Programs → Xilinx Design Tools → DocNav”。
- 在 Linux 命令提示中输入“`docnav`”。

赛灵思设计中心提供了根据设计任务和其他话题整理的文档链接，您可以使用链接了解关键概念以及常见问题解答。访问设计中心：

- 在 DocNav 中，单击“Design Hub View”标签。
- 在赛灵思网站上，查看设计中心页面。

注释: 如需了解更多有关 DocNav 的信息，请参阅赛灵思网站上的 Documentation Navigotor。

请阅读：重要法律提示