MYS-ZU5EV Linux 系统开发指南

文件状态： [] 草稿 [√] 正式发布	**文件标识：**	MYIR-MYS-ZU5EV-SW-DG-ZH-L5.4.0
	当前版本：	V1.11
	作　者：	Fengyong
	创建日期：	2021-05-25
	最近更新：	2021-05-25

版 本 历 史

版本	作者	参与者	日期	备注
V1.10	Fengyong		20210525	初始版本，适用于 MYS-ZU5EV
V1.11	Fengyong		20210721	添加 VCU 等应用示例

目 录

MYS-ZU5EV Linux 系统开发指南 - 1 -
版 本 历 史 - 2 -
目 录 - 3 -
1. 概述 - 6 -
1.1. 软件资源 - 6 -
1.2. 文档资源 - 6 -
2. 开发环境准备 - 7 -
2.1. 开发主机环境 - 7 -
2.2. 软件开发工具介绍 - 8 -
2.3. 安装 Petalinux 工具 - 8 -
2.4. 安装米尔定制的 SDK - 8 -
3. 使用 Petalinux 构建开发板镜像 - 11 -
3.1. 简介 - 11 -
3.2. 获取源码 - 11 -
3.2.1. 从光盘镜像获取源码压缩包 - 11 -
3.2.2. 通过 github 获取源码 - 11 -
3.3. 快速编译开发板镜像 - 12 -
3.4. 构建 SDK - 12 -
4. 如何烧录系统镜像 - 13 -
4.1. 制作 SD 卡启动器 - 13 -
4.2. 制作 SD 卡烧录器 - 16 -
5. 如何修改板级支持包 - 18 -
5.1. Petalinux bsp 介绍 - 18 -
5.2. 板级支持包介绍 - 22 -
5.3. 板载 u-boot 编译与更新 - 23 -
5.3.1. 在 petalinux 项目下编译 u-boot - 23 -

5.3.2. 如何单独更新 Boot.bin ..... - 24 -
5.4. 板载 Kernel 编译与更新 ..... - 25 -
5.4.1. 在 Petalinux 项目下编译 Kernel ..... - 25 -
5.4.2. 如何单独更新 Kernel ..... - 25 -
5.5. Petalinux 项目下构建 FPGA 新工程的 BSP 软件 ..... - 27 -
6. 如何适配您的硬件平台 ..... - 35 -
6.1. 如何创建您的设备树 ..... - 35 -
6.1.1. 板载设备树 ..... - 35 -
6.1.2. 设备树的添加 ..... - 36 -
6.2. 如何根据您的硬件配置 CPU 功能管脚 ..... - 38 -
6.2.1. GPIO 管脚配置的方法 ..... - 38 -
6.2.2. 设备树中引用 GPIO ..... - 38 -
6.3. 如何使用自己配置的管脚 ..... - 40 -
6.3.1. 用户空间使用 GPIO 管脚 ..... - 40 -
7. FPGA PL 功能的 linux 应用示例 ..... - 53 -
7.1. axi-uartlite 中 Petalinux 软件实现流程 ..... - 53 -
8. 如何添加您的应用 ..... - 66 -
8.1. 基于 Makefile 的应用 ..... - 66 -
8.2. 基于 Qt 的应用 ..... - 70 -
8.3. 应用程序开机自启动 ..... - 71 -
8.4. 应用程序示例 ..... - 84 -
8.4.1. CAN 应用示例 ..... - 84 -
8.4.2. I2C 应用示例 ..... - 84 -
8.4.3. 网络应用示例 ..... - 85 -
8.4.4. Uart 应用示例 ..... - 85 -
8.4.5. Framebuffer 应用示例 ..... - 86 -
8.4.6. HDMIin 应用示例 ..... - 86 -
8.4.7. MIPI 摄像头应用示例 ..... - 87 -
8.4.8. VCU 应用示例 ..... - 87 -
9. 参考资料 ..... - 92 -

附录一联系我们 ..... - 93 -

附录二售后服务与技术支持 ..... - 94 -

1. 概述

Linux 系统平台上有许多开源的系统构建框架，这些框架方便了开发者进行嵌入式系统的构建和定制化开发，在 Xilinix 的 zynqMP 平台上，目前比较常见的有 Buildroot, Petalinux 等等。其中 Petalinux 项目使用更强大和定制化的方法，来构建出适合嵌入式产品的 Linux 系统。它不仅仅是一个制作文件系统的工具，同时提供整套的基于 Linux 的开发和维护工作流程，使底层嵌入式开发者和上层应用开发者在统一的框架下开发。

本文主要介绍基于 Petalinux 项目和米尔核心板定制一个完整的嵌入式 Linux 系统的完整流程，其中包括开发环境的准备，代码的获取，以及如何进行 Bootloader, Kernel 的移植，定制适合自身应用需求的 Rootfs 等。我们首先介绍如何基于我们提供的源代码构建适用于 MYS-ZU5EV 开发板的系统镜像，如何将构建好的镜像烧录到开发板。针对那些基于 MYS-ZU5EV 核心板进行项目开发的用户，我们重点介绍了将这一套系统移植到用户的硬件平台上的方法和一些要点，并通过一些实际的 BSP 移植案例和 Rootfs 定制的案例，使用户能够迅速定制适合自己硬件的系统镜像。

本文档并不包含 Petalinux 项目以及 Linux 系统相关基础知识的介绍，适合有一定开发经验的嵌入式 Linux 系统开发人员。

1.1. 软件资源

MYS-ZU5EV 搭载基于 Linux 5.4.0 版本内核的操作系统，提供了丰富的系统资源和其他软件资源。开发板出厂附带嵌入式 Linux 系统开发所需要的交叉编译工具链，U-boot 源代码，Linux 内核和各驱动模块的源代码，应用开发样例等。具体的包含的软件信息请参考《**MYS-ZU5EV SDK 发布说明**》中第 2 章软件信息中的说明。

1.2. 文档资源

根据用户使用开发板的各个不同阶段，SDK 中包含了发布说明，入门指南，评估指南，开发指南，应用笔记，FPGA 指导手册等不同类别的文档和手册。具体的文档列表参考 《**MYS-ZU5EV SDK 发布说明**》表 2-4 中的说明。

2. 开发环境准备

本章主要介绍基于 MYS-ZU5EV 开发板在开发流程所需的一些软硬件环境，包括必要的开发主机环境，必备的软件工具，代码和资料的获取等，具体的准备工作下面将进行详细介绍。

2.1. 开发主机环境

本节将介绍如何搭建适用于 zynqMP 系列处理器平台的开发环境。通过阅读本章节，您将了解相关硬件工具，软件开发调试工具的安装和使用。并能快速的搭建相关开发环境，为后面的开发和调试做准备。zynqMP 系列处理器包含 4 个 ARM Cortex A53 核和 2 个 ARM Cortex R5 核。

- **主机硬件**

Petalinux 项目的构建对开发主机的要求比较高，要求处理器具有 Pentium4 的 2GHz 以上 CPU，8GB 以上内存，100GB 硬盘或更高配置。可以是安装 Linux 系统的主机，也可以是运行 Linux 系统的虚拟机。

- **主机操作系统**

构建 Petalinux 项目的主机操作系统的构建，这里推荐的是 Ubuntu16.04 64bit 桌面版系统，后续开发也是以此系统为例进行介绍。

- **安装必备软件包**

```
# sudo apt-get install tofrodos  iproute2 gawk
# sudo apt-get install gcc git make
# sudo apt-get install xvfb
# sudo apt-get install net-tools  libncurses5-dev  tftpd
# sudo apt-get install zlib1g-dev zlib1g-dev:i386 libssl-dev
# sudo apt-get flex bison libselinux1
# sudo apt-get install gnupgwgetdiffstatchrpathsocatxterm
#sudo apt-get install autoconflibtool tar unzip texinfo
# sudo apt–get zlib1g-dev gcc-multilibbuild-essential
# sudo apt-get libsdl1.2-dev libglib2.0-dev
# sudo apt-get install screen pax gzip tar
```

- **确认 sh 为 bash**

请确保默认的 sh 为 bash，如果没有指向 bash，将会影响后面章节的某些操作执行。

执行以下命令查看 sh

```
# ls -al /bin/sh
/bin/sh -> dash
```

若 sh 指向的是 dash，则需要将其更改为 bash

```
sudo dpkg-reconfigure dash
# ls -al /bin/sh
/bin/sh ->bash
```

确认是指向 bash，然后才能进行后续章节的内容。

2.2. 软件开发工具介绍

在定制适用于 ARM Cortex A53 核心的 Linux 系统过程中会用到许多调试，烧写的工具，在米尔提供的光盘镜像目录 03-Tools 下提供了部分工具，除此之外还会用到如下的工具，简单介绍如下：

petalinux-v2020.1-final-installer.run

这是 petalinux 的安装工具，安装了 petalinux 工具，就可以构建 petalinux 的软件系统。

下载地址：https://www.xilinx.com/support/download.html

2.3. 安装 Petalinux 工具

请确保使用非 root 权限安装 petalinux 工具。执行如下命令，安装 petalinux2020.1。安装期间，将会有 PetaLinux End User License Agreement (EULA)等等提示，需要按键盘“q”，然后按“y”进行协议许可确认。本文中的<WORKDIR>用来表示主机上的工作目录，例如“/home/work/”，请保证目录访问权限。拷贝“petalinux-v2020.1-final-installer.run”到工作目录中。

```
# mkdir –p petalinux
#./petalinux-v2020.1-final-installer.run /home/work/petalinux
```

2.4. 安装米尔定制的 SDK

我们在使用 Petalinux 构建完系统镜像之后，还可以使用 Petalinux 构建一套可扩展的 SDK。在米尔提供的光盘镜像中包含一个编译好的 SDK 包，位于：03-Tools/sdk-qt.tar.xz，这个 SDK 中除了包含一个独立的交叉开发工具链还提供 qmake, 目标平台的 sysroot,

Qt 应用开发所依赖的库和头文件等。用户可以直接使用这个 SDK 来建立一个独立的开发环境，单独编译 Bootloader，Kernel 或者编译自己的应用程序，具体过程在后面的章节中将会详细介绍。这里先介绍 SDK 的安装步骤，如下：

- **拷贝 SDK 到 Linux 目录并解压**

将 SDK 压缩包拷贝到 Ubuntu 下的用户工作目录，如/home/work 下，解压文件，得到安装脚本文件，如下：

```
# cd /home/work
# tar –jxvf sdk-qt.tar.xz
sdk.sh
```

- **安装 SDK**

```
# ./sdk.sh
PetaLinux SDK installer version 2020.1
==========================================
You are about to install the SDK to "/opt/petalinux/2020.1". Proceed [Y/n]? Y
Extracting SD
K.....
.....
.....done
Setting it up...done
SDK has been successfully set up and is ready to be used.
Each time you wish to use the SDK in a new shell session, you need to source the
environment setup script e.g.
```

- **选择安装目录目录**

SDK 默认被安装到/opt/petalinux/2020.1/目录下，用户也可以自定义安装路径，例如安装到/home/work/sdk 目录下：

```
# ./sdk.sh -d /home/work/sdk -y
```

- **测试 SDK**

安装完成后，使用以下命令设置环境变量，测试 SDK 是否安装成功：

```
# source /opt/petalinux/2020.1/environment-setup-aarch64-xilinx-linux

# $CC --version
```

```
aarch64-xilinx-linux-gcc (GCC) 9.2.0
Copyright (C) 2019 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPO
SE.
```

米尔提供的 SDK 中除了包含交叉工具链，还包含 Qt 库，qmake 等开发 Qt 应用程序所需的资源，这些是后续使用 QT Creator 进行应用程序开发和调试的基础。

3. 使用 Petalinux 构建开发板镜像

3.1. 简介

Petalinux 是 Xilinx 公司推出的嵌入式 Linux 开发套件，包括了 Linux Kernel、u-boot、device-tree、rootfs 等源码，可以让客户很方便的生成、配置、编译 MYS-ZU5EV 开发板使用的 linux 系统。关于 Petalinux 的基础知识，请参考 https://www.xilinx.com/support/documentation/sw_manuals/xilinx2020_1/ug1144-petalinux-tools-reference-guide.pdf。

米尔提供的光盘镜像中 04_sources 目录下提供了适用于 MYC-ZU5EV 开发板的 petalinux 的 BSP 包，帮助开发者构建出可运行在 MYS-ZU5EV 开发板上的 Linux 系统镜像。下面以构建镜像为例进行介绍具体的开发流程，为后续定制适合自己的系统镜像打下基础。

3.2. 获取源码

直接从米尔光盘镜像 04-sources 目录中获取压缩包，请用户根据此压缩包进行构建。

3.2.1. 从光盘镜像获取源码压缩包

压缩的源码包位于米尔开发包资料 *04-Sources/Petalinux/mys_zu5ev2020_4G_core.bsp*。拷贝压缩包到用户指定目录，如/home/work/petalinux 目录，这个目录将作为后续构建的顶层目录：

```
#cd home/work/petalinux
```

3.2.2. 通过 github 获取源码

目前 MYS-ZU5EV 开发板的 Petalinux BSP、Kernel 以及 u-boot 源代码均使用了 github 托管并将保持长期更新，代码仓库地址请查看**《MYS-ZU5EV_SDK 发布说明》**。用户可以使用 git 命令和同步 github 上的代码。具体操作方法如下：

```
# mkdir  /home/work/github
# cd /home/work/github
# git clone https://github.com/MYiR-Dev/myir-zynqMP-uboot.git
# git clone https://github.com/MYiR-Dev/myir-zynqMP-kernel.git
```

3.3. 快速编译开发板镜像

在使用 Petalinux 项目进行系统构建之前都需要先设置相应的环境变量。

```
#source /home/work/petalinux/settings.sh
```

- **构建镜像**

我们选择 mys-zu5ev-core 镜像对应的 petalinux bsp 包 *mys_zu5ev2020_4G_core.bsp* 为例进行介绍。

```
# petalinux-create -t project -s mys_zu5ev2020_4G_core.bsp
# cd mys_zu5ev
# petalinux-build
```

- **生成镜像**

```
# petalinux-package --boot --fsbl images/linux/zynqmp_fsbl.elf --u-boot=images/linux/u-boot.elf --pmufw --atf --fpga images/linux/system.bit --force
```

系统构建完成后，在" images/linux "目录下生成各种格式的系统镜像文件，以下是构建后生成的文件信息列表：

表 3-1. 生成 images 文件清单说明

名称	描述
BOOT.bin	fsbl，pmufw，bl31，uboot，devicetree，fpgabit 烧录包生成文件
image.ub	内核与设备树烧录包生成文件
rootfs.tar.gz	文件系统

3.4. 构建 SDK

米尔已经提供较完整的 SDK 安装包，用户可直接使用。但当用户需要在 SDK 中引入新的库，则需要重新使用 Petalinux 构建出新的 SDK 工具。

本节只简单对米尔提供的 SDK 做构建说明，使用如下构建命令生成 SDK 包：

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

构建 SDK 需要较长时间，等待构建完成后，生成 SDK 安装包为“/images/linux/sdk.sh”，安装方法请查看 2.4 节。

4. 如何烧录系统镜像

米尔公司设计的 MYS-ZU5EV 系列核心板有两种启动方式，所以需要不同的更新系统工具与方法。

- 制作 SD 卡启动器: 适用于研发调试，快速启动等场景。
- 制作 SD 卡烧录器:适用于批量生产烧写 eMMC。

4.1. 制作 SD 卡启动器

以下步骤均在 Windows 系统下制作。

1）准备工作

- SD 卡（不少于 4GB）
- MYS-ZU5EV 开发板
- 制作镜像工具 Win32DiskImager-0.9.5-install.exe （路径：\03-Tools\）

表 4-1.镜像包列表

镜像名	包名
mys-zu5ev-core	mys-zu5ev-core.img.gz
mys-zu5ev-full	mys-zu5ev-full.img.gz
mys-zu5ev-mipi	mys-zu5ev-mipi.img.gz

2）制作 SD 卡启动器

以 mys-zu5ev-core 系统为例，其他镜像烧录方式类似。

- **解压下面资源**

mys-zu5ev-core.img.gz

- **将镜像文件写入 Micro SD Card**

将 Micro SD Card 放入读卡器读卡器，然后插入电脑，使用 Win32DiskImager-0.9.5-install.exe 安装 Win32DiskImager，然后双击打开 Win32DiskImager.exe 读出 U 盘分区，点击文件夹图标加载镜像文件。

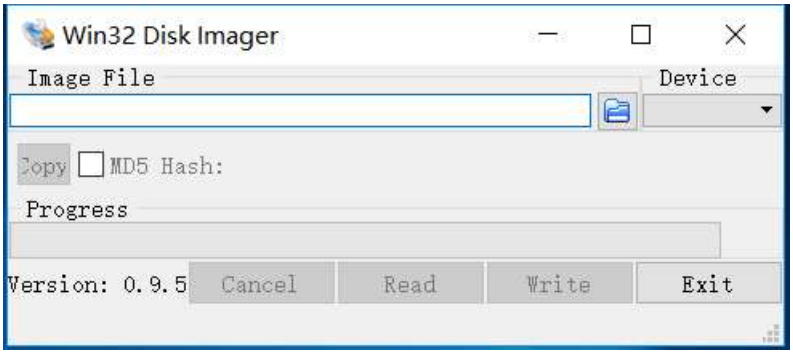

图 4-1.工具配置

选择好系统包后点击打开即可。

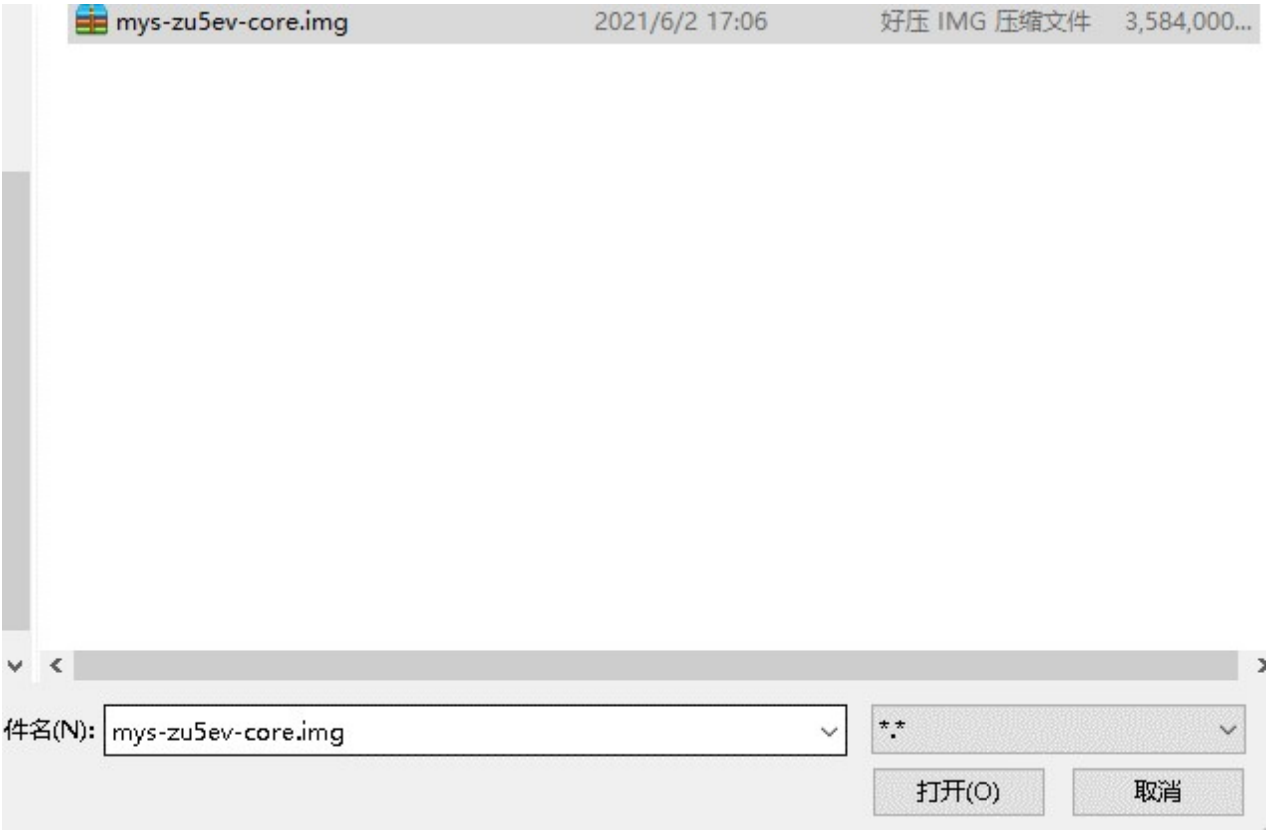

图 4-2.工具配置

加载完镜像后点击“Write”写入按钮即可，会弹出警告，点击“Yes”等待写入完成。

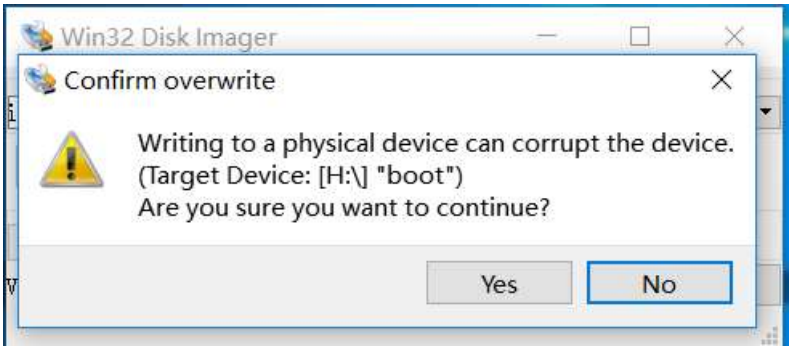

图 4-3.工具配置

等待写入完成，大约几分钟完成，此速度取决于 SD 的读写速度。

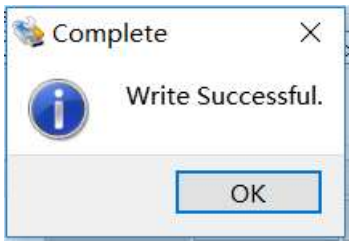

图 4-4.工具配置

● **检查是否烧写成功**

当写入完成后，即可使用此 SD 卡进行启动，将 SD 卡插入开发板 SD 卡槽，然后将开发板的启动模式 switch 开关 SW1 的 1 拨到 OFF，2 拨到 ON，3 拨到 OFF，4 拨到 ON，设置成 TF 卡启动模式。然后上电，即可 SD 卡启动系统。

● **烧写 petalinux 生成的待调试系统从 QSPI flash 启动**

（1）使用上一步制作的 tf 启动卡，将 petalinux 生成的 BOOT.bin、image.ub、rootfs.tar.gz 文件拷贝到 tf 卡的第一个分区（boot 分区）。

（2）将开发板的启动模式 switch 开关 SW1 的 1 拨到 OFF，2 拨到 ON，3 拨到 OFF，4 拨到 ON，设置成 TF 卡启动模式；

（3）插入已存入烧写文件的 TF 卡，连接串口波特率为 115200，开发板上电；

（4）开发板将引导进入 rootfs 文件系统，进入 Linux 命令行，输入命令开始更新：

#update /mnt/sd-mmcblk1p1

脚本将把 BOOT.bin，image.ub 烧写到 QSPI-Flash，把 rootfs.tar.gz 烧写到 eMMC。

（5）烧写完成之后，将开发板的启动模式 switch 开关 SW1 的 1 拨到 ON，2 拨到 OFF，3 拨到 ON，4 拨到 ON，设置成 Qspi flash 启动模式，重新上电，就可以进入烧写的 rootfs 文件系统。

4.2. 制作 SD 卡烧录器

为满足生产烧录的需要，米尔开发了适用于大批量生产的烧录方法。通过 SD 卡中的系统将需要烧录的系统刷写进板载的 Flash 中。具体制作过程请按照下列步骤完成。

➢ 制作更新包

使用 ubuntu 系统制作安装包，需要预先安装下列资源：

```
sudo apt-get install kpartxfdisk mount dosfstools e2fsprogs pv
```

从光盘中拷贝 03-Tools/SDCardUpdate-mys-zu5ev.tar.xz 资源包到 ubuntu 系统中，然后解压。

```
# tar -xvf SDCardUpdater-mys-zu5ev.tar.xz
# ls
```

```
BOOT.bin  CreateSDUpdateImage-myir  image.ub  rootfs.tar.gz  rootfs_update.tar.gz
```

图 4-5.源文件

➢ BOOT.BIN、image.ub、rootfs.tar.gz：由 Petalinux 生成的需要烧录的启动文件
➢ CreateSDUpdateImage-myir：制作脚本
➢ rootfs_update.tar.gz：包含自动烧录程序的根文件系统。

直接使用命令即可自动完成烧录包的制作。

```
# sudo ./CreateSDUpdateImage-myir
```

制作过程的截图如下。

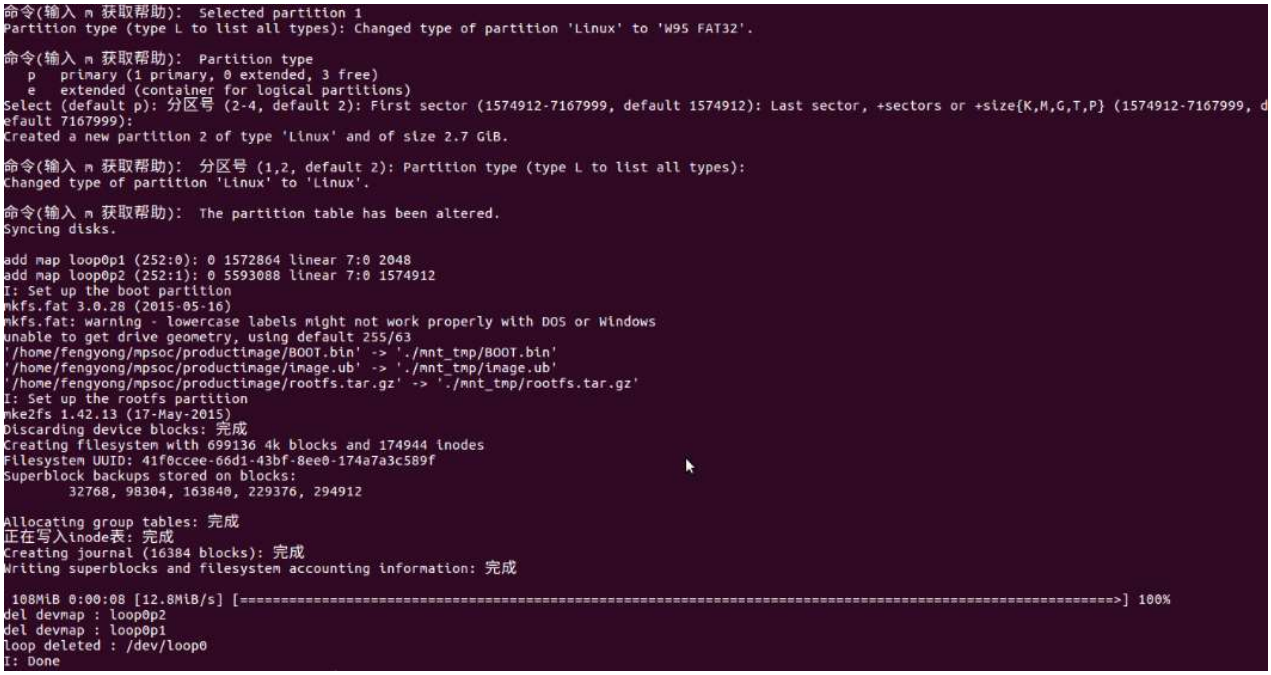

图 4-5.制作截图

制作完成后在同目录下生成了一个压缩包 myir-image-burn-mys-zu5ev.img.gz，这个文件就是用于 SD 卡烧录的镜像包。

➢ 制作 SD 烧录卡

将 myir-image-burn-mys-zu5ev.img.gz 拷贝到 windows 系统下并解压出来。

使用 Win32DiskImager.exe 烧写 myir-image-burn-mys-zu5ev.img 镜像包到 tf 卡。方法与“**4.1 制作 SD 卡启动器**”章节中的“**将镜像文件写入 Micro SD Card**”方法相同。

➢ SD 烧录卡烧写系统

将制作好的 SD 烧录卡插入开发板的 SD 卡槽，然后将开发板的启动模式 switch 开关 SW1 的 1 拨到 OFF，2 拨到 ON，3 拨到 OFF，4 拨到 ON，设置成 TF 卡启动模式，启动系统。插上电源，自动启动 SD Card 内的烧写系统，可以使用调试串口查看更新状态，或者查看复位按键旁的 led 灯，烧录过程中会慢闪，烧录完成后 led 灯会快速闪烁。

当烧写完成后，将开发板的启动模式 switch 开关 SW1 的 1 拨到 ON，2 拨到 OFF，3 拨到 ON，4 拨到 ON，设置成 Qspi flash 启动模式，重新上电，就可以从 qspiflash 模式启动板子，进入烧写的 rootfs 文件系统。

5. 如何修改板级支持包

前面的章节已经比较完整的讲述了基于 Petalinux 项目构建运行在 MYS-ZU5EV 开发板上的系统镜像，并且将镜像烧录到开发板上的完整流程。由于 MYS-ZU5EV 核心板的很多管脚都具有多种功能配置的特性，所以实际项目中总会有一些差异。除了硬件上的差异，还有一些软件系统上的差异，可能需要比较完备的图形系统，QT 库等，侧重后台管理应用的，可能需要更完备的网络应用等。这就需要系统开发人员在我们提供的代码基础上做一些裁剪和移植的工作。本章从一个系统开发人员的角度来讲述开发和定制自己系统的具体过程，为后面适配自己的硬件打下基础。

5.1. Petalinux bsp 介绍

Petalinux bsp 其中包含 BSP、中间件或应用程序的各种元数据和配方。用户可以在这个“层模型”的基础上适配基于 MYS-ZU5EV 核心板设计的硬件，定制自己的应用，从而构建适合自己的系统镜像，petalinux bsp 包含的具体内容如下：

```
mys_zu5ev$ tree -L 3
├── build
│   ├── bitbake-cookerdaemon.log
│   ├── bootgen.bif
│   ├── build.log
│   ├── build.log.old
│   ├── cache
│   │   ├── bb_codeparser.dat
│   │   ├── bb_persist_data.sqlite3
│   │   ├── bb_unihashes.dat
│   │   └── local_file_checksum_cache.dat
│   ├── conf
│   │   ├── bblayers.conf
│   │   ├── devtool.conf
│   │   ├── local.conf
│   │   ├── locked-sigs.inc
│   │   ├── plnxtool.conf
│   │   ├── sanity_info
│   │   ├── sdk-conf-manifest
```

```
|   |   ├── site.conf
|   |   ├── templateconf.cfg
|   |   └── unlocked-sigs.inc
|   ├── config.log
|   ├── downloads
|   ├── misc
|   |   ├── config
|   |   └── rootfs_config
|   ├── sstate-cache
|   └── tmp
|       ├── abi_version
|       ├── buildstats
|       ├── cache
|       ├── deploy
|       ├── hosttools
|       ├── log
|       ├── pkgdata
|       ├── saved_tmpdir
|       ├── sstate-control
|       ├── stamps
|       ├── sysroots
|       ├── sysroots-components
|       ├── sysroots-uninative
|       ├── work
|       └── work-shared
├── components
|   ├── plnx_workspace
|   |   └── device-tree
|   └── yocto
|       ├── cache
|       ├── conf
|       ├── downloads
|       ├── environment-setup-aarch64-xilinx-linux
```

```
|       ├── layers
|       ├── site-config-aarch64-xilinx-linux
|       ├── sysroots
|       ├── version-aarch64-xilinx-linux
|       └── workspace
├── config.project
├── images
|   └── linux
|       ├── bl31.bin
|       ├── bl31.elf
|       ├── BOOT.BIN
|       ├── boot.scr
|       ├── Image
|       ├── image.ub
|       ├── pmufw.elf
|       ├── pxelinux.cfg
|       ├── rootfs.cpio
|       ├── rootfs.cpio.gz
|       ├── rootfs.cpio.gz.u-boot
|       ├── rootfs.jffs2
|       ├── rootfs.manifest
|       ├── rootfs.tar.gz
|       ├── system.bit
|       ├── system.dtb
|       ├── u-boot.bin
|       ├── u-boot.elf
|       ├── vmlinux
|       ├── zynqmp_fsbl.elf
|       ├── zynqmp-qemu-arm.dtb
|       ├── zynqmp-qemu-multiarch-arm.dtb
|       └── zynqmp-qemu-multiarch-pmu.dtb
└── project-spec
    ├── attributes
```

```
├── configs
│   ├── busybox
│   ├── config
│   ├── init-ifupdown
│   ├── rootfs_config
│   └── rootfs_config.old
├── hw-description
│   ├── design_1_wrapper.bit
│   ├── metadata
│   ├── psu_init.c
│   ├── psu_init_gpl.c
│   ├── psu_init_gpl.h
│   ├── psu_init.h
│   ├── psu_init.html
│   ├── psu_init.tcl
│   └── system.xsa
└── meta-user
    ├── conf
    ├── COPYING.MIT
    ├── README
    ├── recipes-apps
    ├── recipes-bsp
    └── recipes-kernel
```

表 5-1. meta-user 层内容说明

源代码与数据	描述
conf	包括开发板软件配置资源信息
recipes-app	包含的应用程序
recipes-bsp	包含 uboot 和 devicetree 等配置资源
recipes-kernel	包含 linux 内核的资源

在进行系统移植时，需要重点关注的是负责硬件初始化和系统引导的 recipes-bsp 部分，负责 Linux 系统的内核和驱动实现的 recipes-kernel 部分以及应用程序定制的 recipes-app 部分。

5.2. 板级支持包介绍

板级支持包(BSP)是定义如何支持特定硬件设备、设备集或硬件平台的信息集合。BSP 包括有关设备上的硬件特性的信息和内核配置信息，以及所需的任何其他硬件驱动程序。

通常根据硬件启动的不同阶段，我们将 BSP 分成 Bootloader 部分和 Kernel 部分，采用 MYS-ZU5EV 核心板设计的硬件 BSP 代码可以查看 meta-user 中的 recipes-bsp 和 recipes-kernel 这两个配方的内容。

recipes-bsp 中只包含了 u-boot 和 device-tree，这一部分主要实现核心硬件，如 DDR，Clock 的初始化及内核的引导。基于 MYS-ZU5EV 核心板硬件修改这部分的内容。

```
recipes-bsp
├── device-tree
└── u-boot
```

recipes-kernel 中包含 Linux 内核，主要实现内核内容。

```
recipes-kernel/
└── linux
```

在使用米尔的核心板设计产品时，如无特殊的需求，bootloader 部分可不必修改。你需要更多的关注产品内核驱动的开发与调试以及应用软件的设计。后续章节将详细描述内核开发与应用开发。

5.3. 板载 u-boot 编译与更新

U-boot 是一个功能非常丰富的开源启动引导程序，包括内核引导，下载更新等众多方面，在嵌入式领域应用十分广泛，可查看官网获取更多信息 http://www.denx.de/wiki/U-Boot/WebHome

5.3.1. 在 petalinux 项目下编译 u-boot

当用户修改好 U-boot 的代码之后，也可以使用 petalinux 进行整个镜像的构建。参考示例如下。

```
# git add .
# git commit -m "demo"
# git format-patch -1
```

修改完成后，会产生 0001-demo.patch，将这个 patch 拷贝到 petalinux 的 project-spec/meta-user/recipes-bsp/u-boot/files/目录下，然后需要在 project-spec/meta-user/recipes-bsp/u-boot/u-boot-xlnx_%.bbappend 文件中加入如下内容，再重新编译 u-boot，就能将新修改的代码编译进来。

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

SRC_URI += "file://platform-top.h"
SRC_URI += "file://0001-add-config.patch"
SRC_URI += "file://0002-modify-config.patch"
SRC_URI += "file://0003-reset-ushhub.patch"
SRC_URI += "file://0001-demo.patch"

do_configure_append () {
	if [ "${U_BOOT_AUTO_CONFIG}" = "1" ]; then
		install ${WORKDIR}/platform-auto.h ${S}/include/configs/
		install ${WORKDIR}/platform-top.h ${S}/include/configs/
	fi
}

do_configure_append_microblaze () {
```

```
        if [ "${U_BOOT_AUTO_CONFIG}" = "1" ]; then
                install -d ${B}/source/board/xilinx/microblaze-generic/
                install ${WORKDIR}/config.mk ${B}/source/board/xilinx/microblaze-gen
eric/
        fi
}
```

可按照如下命令构建 u-boot。

```
# petalinux-build -c u-boot -x distclean
# petalinux-build -c u-boot
```

5.3.2. 如何单独更新 Boot.bin

1）生成 boot.bin

在更新 boot.bin 之前，我们需要先生成 boot.bin 文件，执行以下命令即可生成：

```
# petalinux-package --boot --fsbl images/linux/zynqmp_fsbl.elf --u-boot=images
/linux/u-boot.elf --pmufw --atf --fpga images/linux/system.bit  --force
```

2）单独更新 boot.bin

核心板上配备了一块可用空间为 32M qspiflash，其中默认 flash 的第一分区为 boot.bin 的分区，即/dev/mtd0。

将生成的 images/linux/boot.bin 通过网络或 U 盘或 SD 卡拷贝到开发板中后，这里我们将 boot.bin 拷贝到了用户主目录下，执行以下命令即可单独将 boot.bin 烧录到 qspiflash 中。

```
# flashcp -v boot.bin /dev/mtd0
```

5.4. 板载 Kernel 编译与更新

Linux kernel 是个十分庞大的开源内核，被应用在各种发行版操作系统上，Linux kernel 以其可移植性，多种网络协议支持，独立的模块机制，MMU 等诸多丰富特性，使 Linux kernel 能在嵌入式系统中被广泛采用。

MYS-ZU5EV 使用的 linux 版本为 Linux kernel 5.4.0 版本。

5.4.1. 在 Petalinux 项目下编译 Kernel

当用户修改好 kernel 的代码之后，也可以使用 petalinux 进行整个镜像的构建。参考示例如下。

```
# git add .
# git commit -m "demo"
# git format-patch -1
```

修改完成后，会产生 0001-demo.patch，将这个 patch 拷贝到 petalinux 的 project-spec/meta-user/recipes-kernel/linux/linux-xlnx 目录下，然后需要在 project-spec/meta-user/recipes-kernel/linux/linux-xlnx_%.bbappend 文件中加入如下内容，再重新编译 kernel，就能将新修改的代码编译进来。

```
SRC_URI += "file://bsp.cfg"
SRC_URI += "file://0001-add-watchdog.patch"
SRC_URI += "file://0002-adv7619-driver.patch"
SRC_URI += "file://0003-mplane.patch"
SRC_URI += "file://0001-demo.patch"

KERNEL_FEATURES_append = " bsp.cfg"
FILESEXTRAPATHS_prepend := "${THISDIR}/${PN}:"
```

修改完成后，可按照如下命令构建 kernel。

```
# petalinux-build -c kernel distclean
# petalinux-build -c kernel
```

5.4.2. 如何单独更新 Kernel

用户编译成功之后，可将 images/linux/image.ub 文件通过以太网、 U 盘等传输介质传输到开发板执行以下命令即可完成更新。

```
# flashcp -v image.ub /dev/mtd1
```

5.5. Petalinux 项目下构建 FPGA 新工程的 BSP 软件

在 MYS_ZU5EV 开发板开发了新的 FPGA 功能，要使 fpga 功能正常使用起来，就需要开发对应的软件。我们使用 petalinux 来快速构建项目，生成对应的一套软软件。具体的步骤流程如下：

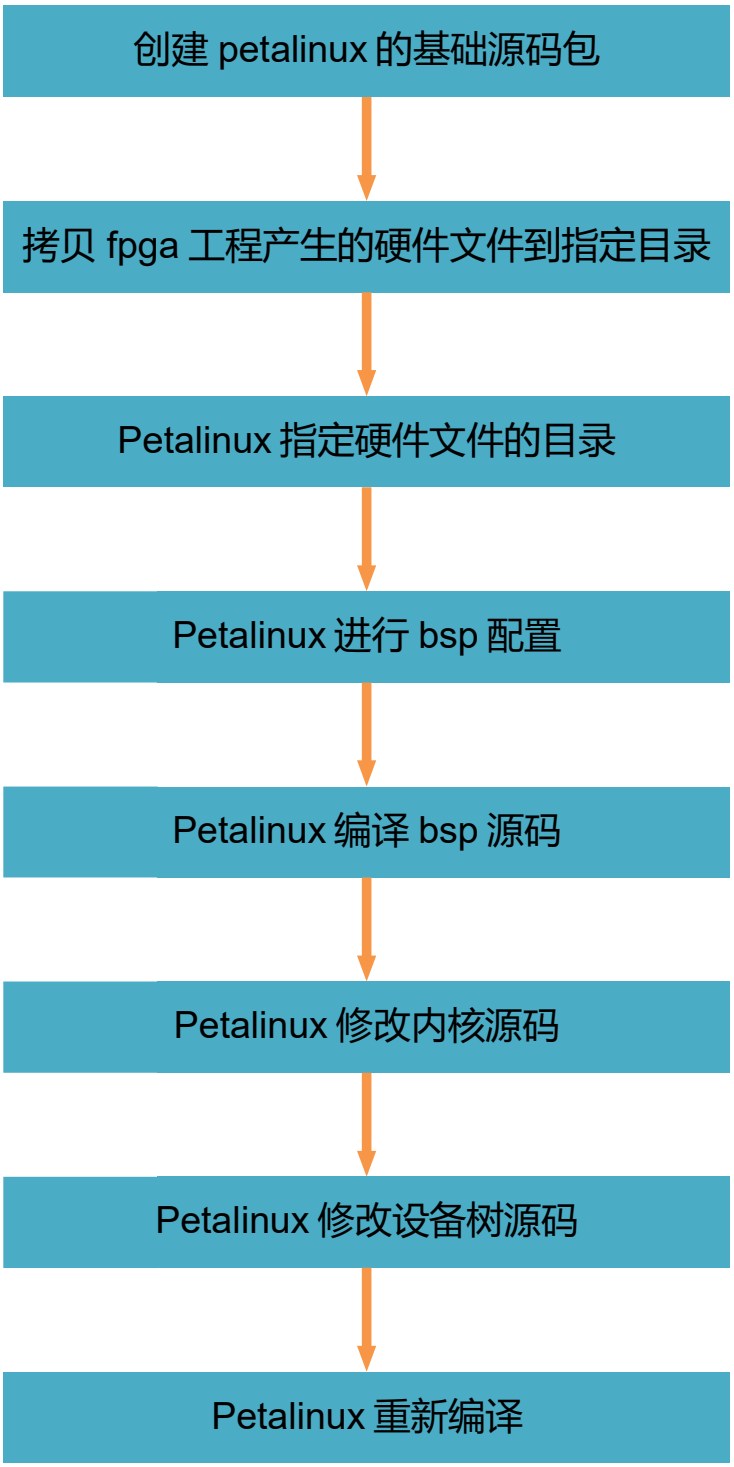

图 5-1 petalinux 构建项目流程

- **创建 petalinux 的基础源码包**

以光盘中的 bsp 作为基础源码包，这里是以 04-Source/Petalinux/mys_zu5ev2020_4G_core.bsp 为示例创建。

```
# petalinux-create -t project -s mys_zu5ev2020_4G_core.bsp
# cd mys_zu5ev
```

- **拷贝 fpga 工程产生的硬件文件到指定目录**

创建了 mys_zu5ev2020_4G_core.bsp 的源码包后，后面的操作都是在此基础上进行。将 fpga 工程产生的硬件描述文件 design_1_wrapper.xsa 拷贝到指定的目录下，此处以/home/work/zu5ev 目录为例。

```
# ls /home/work/zu5ev/design_1_wrapper.xsa
/home/ work/zu5ev /design_1_wrapper.xsa
```

- **Petalinux 指定硬件文件的目录**

Petalinux 指定硬件描述文件的目录，petalinux 编译时就会以此硬件描述文件的目录中的硬件描述文件 design_1_wrapper.xsa 来构建 petalinux 项目。操作如下：

```
# petalinux-config --get-hw-description=/home/work/zu5ev
```

- **Petalinux 进行 bsp 配置**

指定了硬件描述文件的目录后，上面的 petalinux-config 目录会自动进入 petalinux 的配置界面。在此界面中，可以根据需要进行配置。

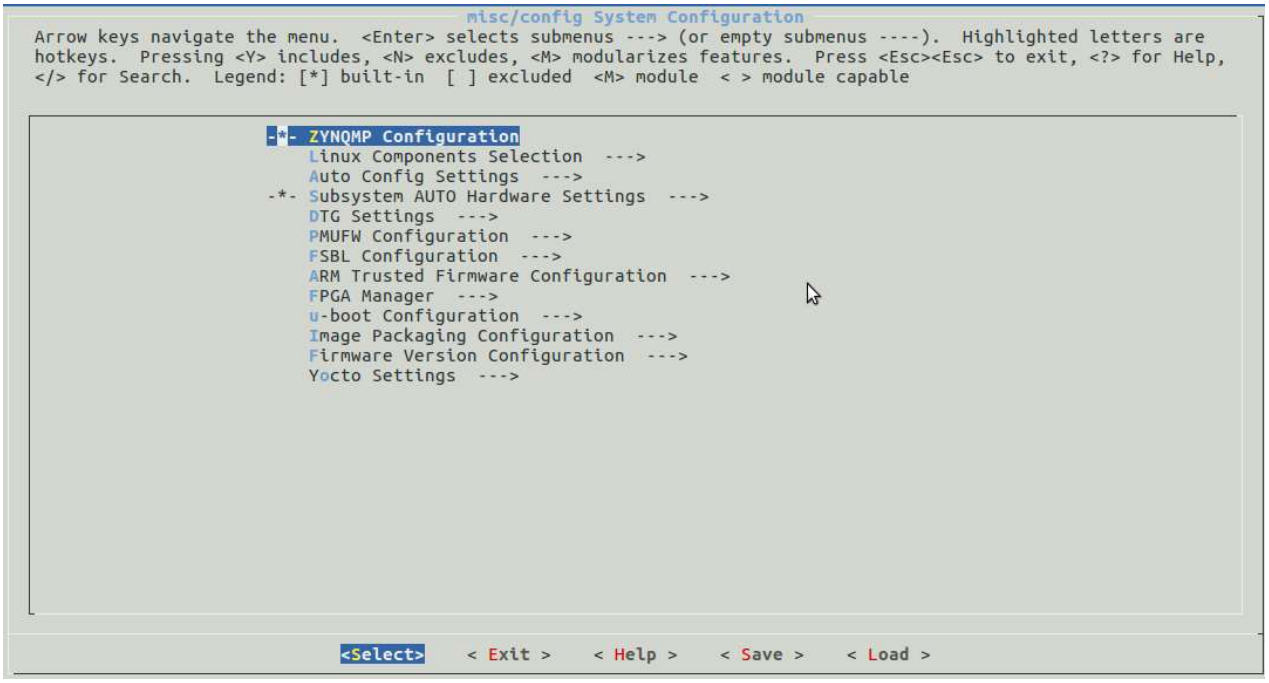

图 5-2 配置界面

这里列举几个重要的配置内容供参考。

（1）设备树配置为 petalinux 自动配置，kernel 和 uboot 配置为非自动配置，也就是我们手动可以配置。如下的选项：

```
[*] Device tree autoconfig
[ ] kernel autoconfig
[ ] u-boot autoconfig
```

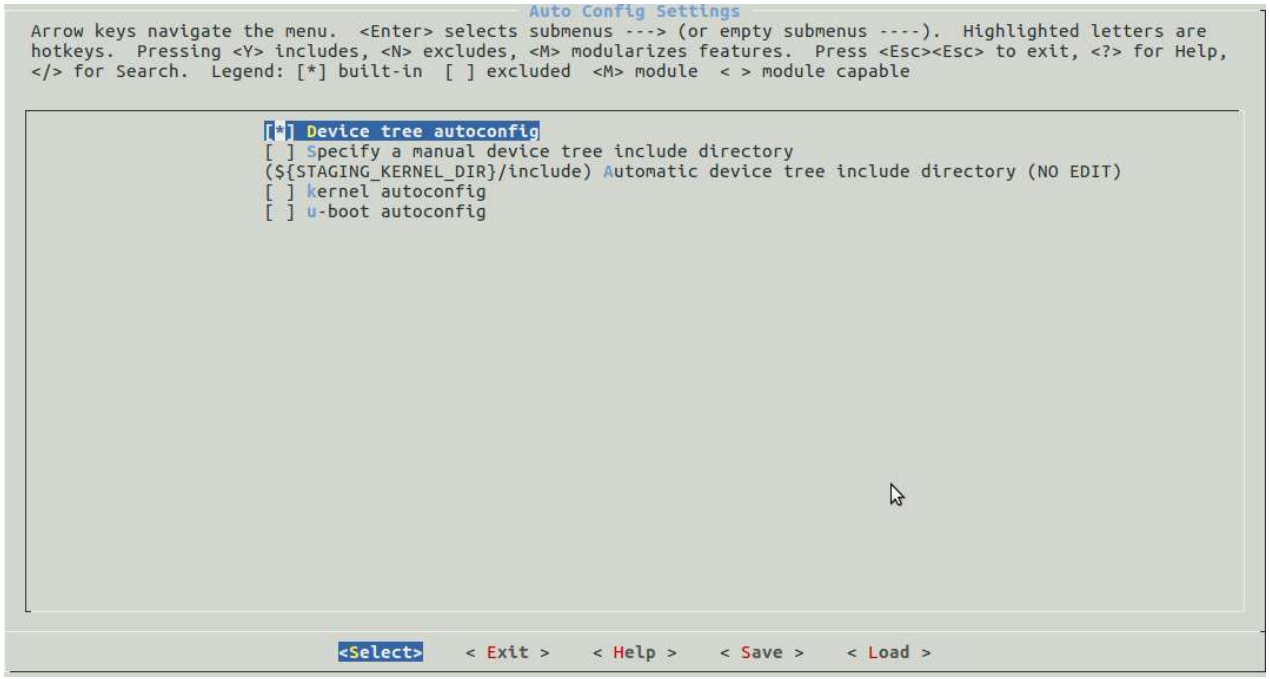

图 5-3 自动配置选择

(2)qspi flash 配置

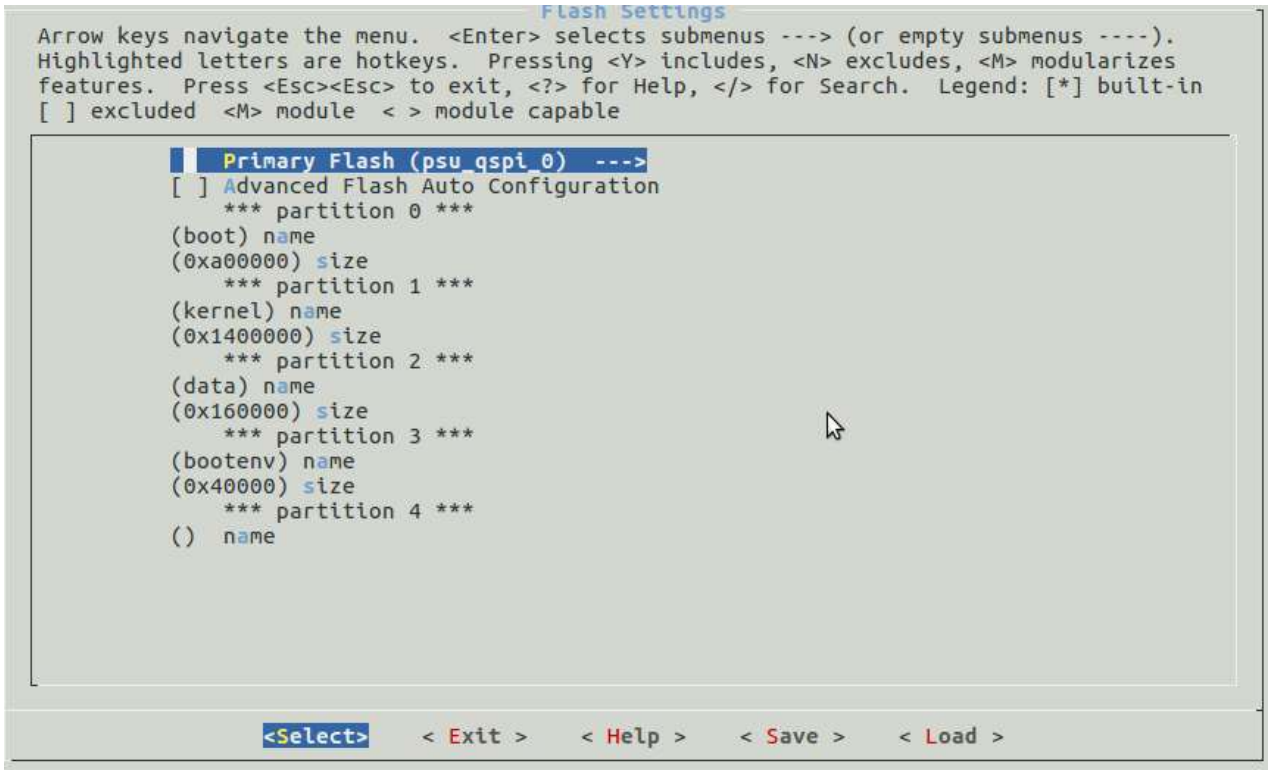

图 5-4 qspi 配置

如果需要配置文件系统可以输入如下命令配置：

```
#petalinux-config –c rootfs
```

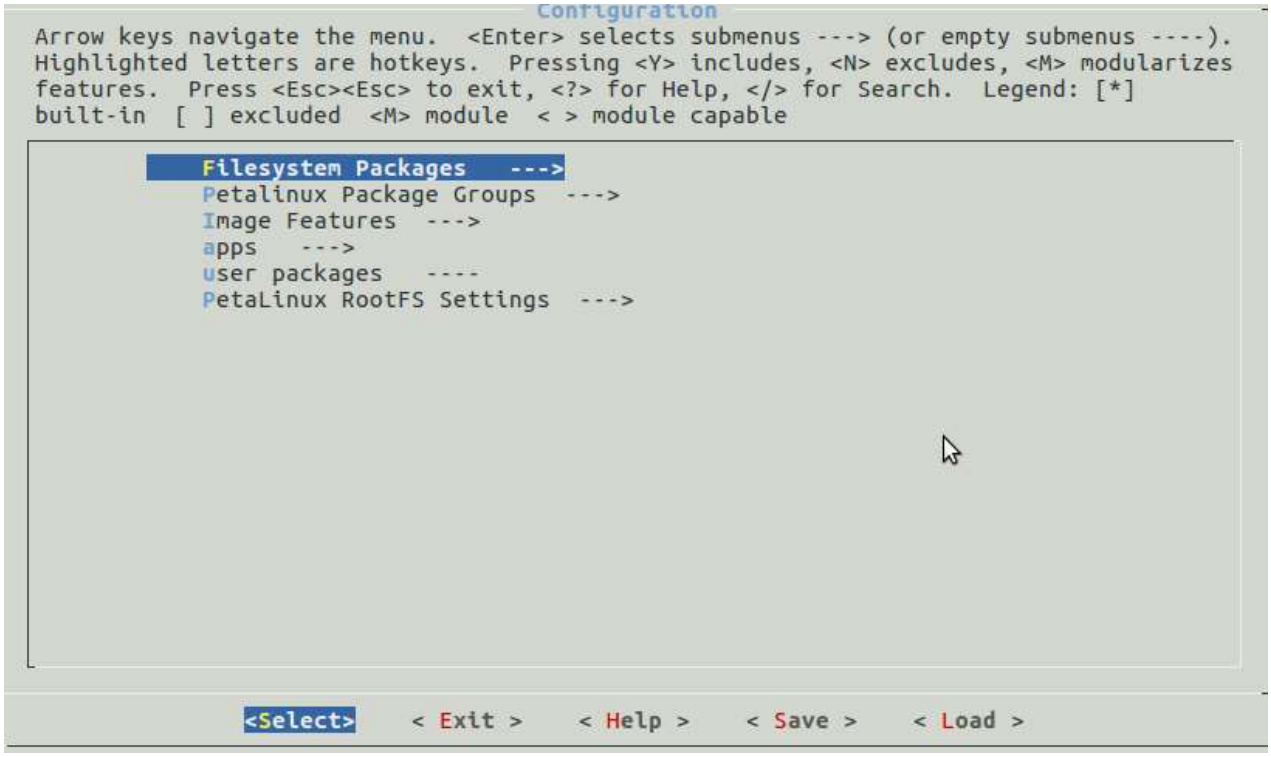

图 5-5 配置 rootfs

比如需要配置 qt 功能，应选择如下配置：

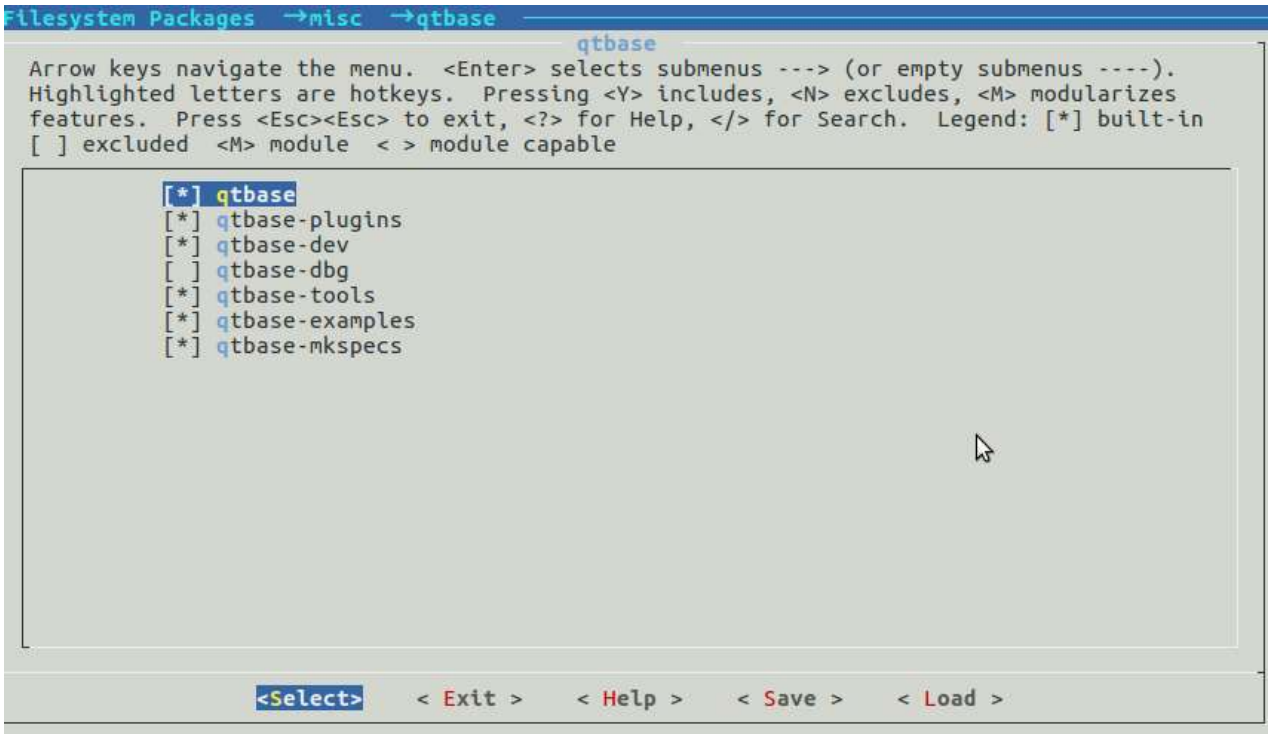

图 5-6 配置 qt

配置完成后，就可以构建 petalinux 项目，也就是 petalinux 自动下载并编译 petalinux 源码。

- **Petalinux 构建 bsp 源码**

```
#petalinux-build
```

编译完成后，根据需要对 kernel 源码和设备树源码进行修改，添加或者修改代码后，重新编译 petalinux 源码，就可以产生对应的 petalinux 镜像文件。

- **Petalinux 修改内核源码**

修改 petalinux 中内核源码，先进入到 petalinux 项目中的 linux 源码目录下 build/tmp/work/zynqmp_generic-xilinx-linux/linux-xlnx/5.4+gitAUTOINC+22b71b4162-r0/linux-zynqmp_generic-standard-build/，修改 linux 源码。然后按照如下步骤产生补丁文件 0001-demo.patch。

```
# git add .
# git commit -m "demo"
# git format-patch -1
```

将补丁文件加入到 petalinux 项目中的 project-spec/meta-user/recipes-kernel/linux/linux-xlnx 目录。然后需要在 project-spec/meta-user/recipes-kernel/linux/linux-xlnx_%.bbappend 文件中加入如下内容，再重新编译 kernel，就能将新修改的代码编译进来。

```
SRC_URI += "file://bsp.cfg"
SRC_URI += "file://0001-add-watchdog.patch"
SRC_URI += "file://0002-adv7619-driver.patch"
SRC_URI += "file://0003-mplane.patch"
SRC_URI += "file://0001-demo.patch"

KERNEL_FEATURES_append = " bsp.cfg"
FILESEXTRAPATHS_prepend := "${THISDIR}/${PN}:"
```

- **Petalinux 修改设备树源码**

按照类似如下的方法添加需要的设备树源码。

```
/include/ "system-conf.dtsi"

#include <dt-bindings/media/xilinx-vip.h>

/ {
        chosen {
```

```
            bootargs = "console=ttyPS0,115200 earlyprintk root=/dev/mmcblk1p2 r
wrootwaitclk_ignore_unused";
            stdout-path = "serial0:115200n8";
        };

leds {
            compatible = "gpio-leds";
            led1 {
                label = "rs485_de";
                gpios = <&gpio 12 0>;
                linux,default-trigger = "gpio";
            };
            led2 {
                label = "wdt_en";
                gpios = <&gpio 33 0>;
                linux,default-trigger = "gpio";
            };
            led3 {
                label = "led_sys";
                gpios = <&gpio 43 0>;
                linux,default-trigger = "gpio";
            };
        };
        ......
};

&gem3 {
phy-handle = <&phy0>;
phy-mode = "rgmii-id";
        phy0: phy@21 {
            reg = <4>;
        };
};
```

```
.....
```

至此，整个 petalinux 的构建完成，重新编译就能生成 petalinux 项目的镜像文件。

6. 如何适配您的硬件平台

为了适配用户新的硬件平台，首先需要了解米尔的 MYS-ZU5EV 开发板提供了哪些资源，具体的信息可以查看《**MYS-ZU5EV SDK 发布说明**》。除此之外用户还需要对 CPU 的芯片手册，以及 MYS-ZU5EV 核心板的产品手册，管脚定义有比较详细的了解，以便于根据实际的功能对这些管脚进行正确的配置和使用。

6.1. 如何创建您的设备树

6.1.1. 板载设备树

用户可以在 BSP 源码里创建自己的设备树，只需要根据实际的硬件资源对 Linux 内核设备树进行适当的调整即可。在此将 MYS-ZU5EV 的 BSP 各个部分中的设备树列表罗列出来，方便用户开发参考，具体内容如下表所示：

表 6-1 设备树文件说明

路径	设备树	说明
components/plnx_workspace/device-tree/device-tree/	zynqmp.dtsi	包含 zynqmp 系列器件的基础配置，由 petalinux 自动生成，请勿手动修改。
	pcw.dtsi	包含 PS 端的基础配置，由 petalinux 自动生成，请勿手动修改。
	system-conf.dtsi	petalinuxbsp 中对外设的配置，由 petalinux 自动生成，请勿手动修改。
	system-top.dts	顶层配置，由 petalinux 自动生成，请勿手动修改。
	pl.dtsi	包含 PL 端的配置，由 petalinux 自动生成，请勿手动修改。
project-spec/meta-user/recipes-bsp/device-tree/files	system-user.dtsi	开发板外设资源的配置，可根据实际硬件情况自行修改。
	pl-custom.dtsi	可根据实际 PL 端设计自行改动。

6.1.2. 设备树的添加

Linux 内核设备树是一种数据结构，它通过特有的语法格式描述片上片外的设备信息。由 BootLoader 传递给 kernel，kernel 进行解析后形成和驱动程序关联的 dev 结构供驱动代码使用。

用户如果想为自己的硬件添加一个新的设备树，只需在 project-spec/meta-user/recipes-bsp/device-tree/files 目录新建一个 dtsi 文件，然后包含在 system-user.dtsi 中即可。下面的示例是新建一个设备树并加入一个通过 mio 控制 led 的节点。

- **修改设备树**

```
# touch user-define.dtsi
```

然后在 system-user.dtsi 中包含这个设备树：

```
/include/ "system-conf.dtsi"
/include/ "user-define.dtsi"

/ {
    leds {
                compatible = "gpio-leds";
                led1 {
                        label = "rs485_de";
                        gpios = <&gpio 12 0>;
                        linux,default-trigger = "gpio";
                };
                led2 {
                        label = "wdt_en";
                        gpios = <&gpio 33 0>;
                        linux,default-trigger = "gpio";
                };
    };
    ........
};
```

- **新增 led 节点**

编辑设备树 user-define.dtsi，增加 led 节点，如下所示：

```
/ {
        gpio-leds {
            compatible =  “gpio-leds”;
            led3 {
                label = "led_sys";
                gpios = <&gpio 43 0>;
                linux,default-trigger = "gpio";
            };
        };
};
```

- **在配方中添加新建的设备树**

编辑配方“project-spec/meta-user/recipes-bsp/device-tree/device-tree.bbappend”，修改为如下所示:

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

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

python () {
    if d.getVar("CONFIG_DISABLE"):
        d.setVarFlag("do_configure", "noexec", "1")
}

export PETALINUX
do_configure_append () {
        script="${PETALINUX}/etc/hsm/scripts/petalinux_hsm_bridge.tcl"
        data=${PETALINUX}/etc/hsm/data/
        eval xsct -sdx -nodisp ${script} -c ${WORKDIR}/config \
        -hdf ${DT_FILES_PATH}/hardware_description.${HDF_EXT} -repo ${S} \
        -data ${data} -sw ${DT_FILES_PATH} -o ${DT_FILES_PATH} -a "soc_mapping"
}
```

6.2. 如何根据您的硬件配置 CPU 功能管脚

实现一个功能引脚的控制是一个较为复杂的系统开发过程之一，其中包含了驱动的开发，应用的实现等等步骤，本节不具体分析每个部分的开发过程，而是以实例来讲解功能管脚的控制实现。

6.2.1. GPIO 管脚配置的方法

GPIO: General-purpose input/output，通用的输入输出口，在嵌入式设备中是一个十分重要的资源，可以通过它们输出高低电平或者通过它们读入引脚的状态-是高电平或是低电平。

GPIO 配置方法

GPIO 的配置可通过米尔整理的 Datasheet 找到描述文件（01-Document\Datasheet\CPU\ug1085-zynq-ultrascale-trm.pdf）与 MYS_ZU5EV 开发板的原理图（01-Document\HardwareFiles\Schematic\MYS-ZU5EV-32E4D-EDGE_V11.pdf），说明如下：

GPIO 分为 78 个 MIO 和 96 个 EMIO，具体参考光盘中的 01-Document\Datasheet\CPU\ug1085-zynq-ultrascale-trm.pdf 文档中的“GeneralPurposeI/O”章节，里面有详细的说明。

- **通用计算方法**

MIO 的 GPIO number=MIO 端口号，例如 MIO0 的 GPIO number 为 GPIO0，EMIO 的 GPIO number = EMIO 端口号 + 78，例如 EMIO0 的 GPIO number 为 GPIO78。

6.2.2. 设备树中引用 GPIO

1）配置功能管脚为 GPIO 功能实例

此实例使用 PS_MIO43 作为测试 GPIO。介绍如何在设备树里配置 GPIO，并为后面章节供内核驱动使用。

只需在设备树 project-spec/meta-user/recipes-bsp/device-tree/file/system-user.dtsi 里增加节点即可。

```
.....
    leds {
        compatible = "gpio-leds";
        led1 {
            label = "rs485_de";
```

```
                gpios = <&gpio 12 0>;
                linux,default-trigger = "gpio";
        };
        led2 {
                label = "wdt_en";
                gpios = <&gpio 33 0>;
                linux,default-trigger = "gpio";
        };
        led3 {
                label = "led_sys";
                gpios = <&gpio 43 0>;
                linux,default-trigger = "gpio";
        };
    };
......
```

6.3. 如何使用自己配置的管脚

我们在 Kernel 的设备树中配置后的管脚，可以在 Kernel 中进行使用，从而实现对管脚的控制。

6.3.1. 用户空间使用 GPIO 管脚

Linux 操作系统的体系架构分为用户态和内核态（或者用户空间和内核）。用户态即上层应用程序的活动空间，应用程序的执行必须依托于内核提供的资源，包括 CPU 资源、存储资源、I/O 资源等。为了使上层应用能够访问到这些资源，内核必须为上层应用提供访问的接口：即系统调用。

Shell 是一个特殊的应用程序，俗称命令行，本质上是一个命令解释器，它下通系统调用，上通各种应用。使用 Shell 脚本，通常短短的几行 Shell 脚本就可以实现一个非常大的功能，原因就是这些 Shell 语句通常都对系统调用做了一层封装。为了方便用户和系统交互。

本节将讲述在用户态如何使用 GPIO 管脚的控制三种基本方式。

- Shell 命令
- 系统调用
- 库函数

1）Shell 实现管脚控制

Shell 控制管脚实质上是调用 Linux 提供的文件操作接口实现的，本节不做详细的说明，可查看**《MYS-ZU5EV_Linux 软件评估指南》**第 3.1 节描述。

2）库函数实现管脚控制

从 Linux 4.8 版本开始，Linux 引入了新的 gpio 操作方式，GPIO 字符设备。基于"文件描述符"的字符设备，每个 GPIO 组在"/dev"下有一个对应的 gpiochip 文件，例如"/dev/gpiochip0 对应 GPIOA, /dev/gpiochip1 对应 GPIOB"等等。

Libgpiod 库函数实现由于 gpiochip 的方式，基于 C 语言，所以开发者实现了 Libgpiod，提供了一些工具和更简易的 C API 接口。Libgpiod (Library General Purpose Input/Output device) 提供了完整的 API 给开发者，同时还提供了一些用户空间下的应用来操作 GPIO。

Libgpiod 常用基本接口描述：

- gpiodetect - 列出系统中出现的所有 gpiochip，它们的名称，标签和 GPIO 行数。

➢ gpioinfo - 列出指定的 gpiochips 的所有行、它们的名称、使用者、方向、活动状态和附加标志。
➢ gpioget -读取指定的 GPIO 行值。
➢ gpioset - 设置指定的 GPIO 行值，潜在地保持这些行导出并等待超时、用户输入或信号。
➢ gpiofind - 查找给定行名称的 gpiochip 名称和行偏移量。
➢ gpiomon - 等待 GPIO 行上的事件，指定要观察哪些事件，退出前要处理多少事件，或者是否应该将事件报告到控制台。

更多描述，可查看 libgpiod 源代码 https://git.kernel.org/pub/scm/libs/libgpiod/libgpiod.git/。

下列将以 MIO43 做为操作 GPIO 管脚来实现 C 语言的代码控制实例(交替置高置低)。

```
//example-gpio.c
#include <errno.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ioctl.h>
#include <unistd.h>
#include <linux/gpio.h>

int main(int argc, char **argv)
{
        struct gpiohandle_request req;
        struct gpiohandle_data data;
        char chrdev_name[20];
        int fd, ret;

        strcpy(chrdev_name, "/dev/gpiochip0");

        /* Open device: gpiochip0 for mio43*/
        fd = open(chrdev_name, 0);
```

```
if (fd == -1) {
        ret = -errno;
        fprintf(stderr, "Failed to open %s\n", chrdev_name);

return ret;
}

/* request GPIO line: mio43 */
req.lineoffsets[0] = 43;
req.flags = GPIOHANDLE_REQUEST_OUTPUT;
memcpy(req.default_values, &data, sizeof(req.default_values));
strcpy(req.consumer_label, "gpio_43");
req.lines = 1;

ret = ioctl(fd, GPIO_GET_LINEHANDLE_IOCTL, &req);
if (ret == -1) {
        ret = -errno;
        fprintf(stderr,"Failed to issue GET LINEHANDLE IOCTL (%d)\n",ret);
}
if (close(fd) == -1)
        perror("Failed to close GPIO character device file");

/* Start GPIO ctr*/
while(1) {
        data.values[0] = !data.values[0];
        ret = ioctl(req.fd, GPIOHANDLE_SET_LINE_VALUES_IOCTL, &data);
        if (ret == -1) {
                ret = -errno;
                fprintf(stderr,"Failed to issue %s (%d)\n", ret);
        }
        sleep(1);
}
```

```
        /* release line */
        ret = close(req.fd);
        if (ret == -1) {
                perror("Failed to close GPIO LINEHANDLE device file");
                ret = -errno;
        }
        return ret;
}
```

将上述代码拷贝到一个 gpioex.c 文件，加载 SDK 环境变量到当前 shell:

```
# source /opt/petalinux/2020.1/environment-setup-aarch64-xilinx-linux
```

使用编译命令$CC 可生成可执行文件 gpioex。

```
#$CC -o gpioex gpioex.c
```

只需在设备树 project-spec/meta-user/recipes-bsp/device-tree/file/system-user.dtsi 里屏蔽掉 led3 即可。

```
......
        leds {
                compatible = "gpio-leds";
                led1 {
                        label = "rs485_de";
                        gpios = <&gpio 12 0>;
                        linux,default-trigger = "gpio";
                };
                led2 {
                        label = "wdt_en";
                        gpios = <&gpio 33 0>;
                        linux,default-trigger = "gpio";
                };
                /*
                led3 {
                        label = "led_sys";
```

```
                gpios = <&gpio 43 0>;
                linux,default-trigger = "gpio";
            };
            */
        };
......
```

编译生成 image.ub，替换 tf 卡中 image.ub，tf 卡启动模式启动系统。

将可执行文件通过网络，u 盘或者 tf 卡等传输介质拷贝到开发板的目录下，即可在终端下输入命令可直接运行，可以看到复位按键旁的 led 灯闪烁。

```
# ./gpioex
```

3）系统调用实现管脚控制

操作系统提供给用户程序调用的一组“特殊”接口。用户程序可以通过这组“特殊”接口来获得操作系统内核提供的服务，比如用户可以通过文件系统相关的调用请求系统打开文件、关闭文件或读写文件，可以通过时钟相关的系统调用获得系统时间或设置定时器等。

MIO43 作为 gpio-leds 的功能时，可以通过驱动程序所控制的管脚进行系统调用控制。

```
/*****************************************************************************
 * Copyright (c) 2014-2017 MYIR Tech Ltd.
 *       File: led-test.c
 *       Date: 2014/11/3
 *     Author: Kevin Su
 * Description: A demo program to show how to control leds from user-space.
 */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>
#include <signal.h>
#include <linux/input.h>
```

```
#define DEBUG  1

#define ERR_MSG(fmt, args...)      fprintf(stderr, fmt, ##args)
#ifdef DEBUG
        #define DBG_MSG(fmt, args...)  fprintf(stdout, fmt, ##args)
#else
        #define DBG_MSG(fmt, args...)
#endif

#define LED_DIR         "/sys/class/leds/led_sys/"
#define NAME_MAX_LENGTH             64
#define LED_DELAY_US       (100*1000)

#define ARRAY_SIZE(x)       (sizeof(x)/sizeof(x[0]))

#define BITS_MASK(num)    ((1<<num) - 1)

typedef struct led_ctrl_s {
        char name[NAME_MAX_LENGTH];
        char brightness[NAME_MAX_LENGTH];
        char trigger[NAME_MAX_LENGTH];
        char trigger_backup[NAME_MAX_LENGTH];
        int state;
        int initialized;
} led_ctrl_t;

static led_ctrl_tleds[] = {
        /* name, brightness, trigger, trigger_str,  state, initialized */
        {"led_sys", LED_DIR "brightness", LED_DIR "trigger", "", 0, 0},
};

static int led_set_trigger(led_ctrl_t *led, const char *trigger)
```

```
{
        int ret;
        int fd = open(led->trigger, O_WRONLY);

        if (fd< 0) {
            ERR_MSG("Open %s failed!\n", led->trigger);
            return -1;
        }

        ret = write(fd, trigger, strlen(trigger));
        if (ret != strlen(trigger)) {
            ERR_MSG("Write %s failed!\n", led->trigger);
            close(fd);
            return -1;
        }
        close(fd);
        DBG_MSG("[%8s] Set trigger to '%s'\n",
            led->name,
            trigger);

        return 0;
}

static int led_get_trigger(led_ctrl_t *led, char *trigger)
{
        int ret;
        char tmp[1000] = {0};
        char *ptr[2] = {NULL};
        int fd = open(led->trigger, O_RDONLY);

        if (fd< 0) {
            ERR_MSG("Open %s failed!\n", led->trigger);
            return -1;
```

```
}

/* read back string with format like: "none cpu1 cpu2 [heartbeat] nand" */
ret = read(fd, tmp, sizeof(tmp));
if (ret <= 0) {
	ERR_MSG("Read %s failed!\n", led->trigger);
	close(fd);
	return -1;
}
close(fd);

/* find trigger from read back string, which is inside "[]" */
ptr[0] = strchr(tmp, '[');
if (ptr[0]) {
	ptr[0] += 1;
	ptr[1] = strchr(ptr[0], ']');
	if (ptr[1]) {
		*ptr[1] = '\0';
	} else {
		ERR_MSG("[%s] Can not find trigger in %s!\n", led->name, tmp);
		return -1;
	}
	strcpy(trigger, ptr[0]);
} else {
	ERR_MSG("[%s] Can not find trigger in %s!\n", led->name, tmp);
	return -1;
}

DBG_MSG("[%8s] Get trigger: '%s'\n",
	led->name,
	trigger);

return 0;
```

```
}

static int led_set_brightness(led_ctrl_t * led, int brightness)
{
        int ret;
        int fd = open(led->brightness, O_WRONLY);
        char br_str[2] = {0};

        br_str[0] = '0' + brightness;

        if (fd< 0) {
            ERR_MSG("Open %s failed!\n", led->brightness);
            return -1;
        }

        ret = write(fd, br_str, sizeof(br_str));
        if (ret != sizeof(br_str)) {
            ERR_MSG("Write %s failed!\n", led->brightness);
            close(fd);
            return -1;
        }
        close(fd);
        // DBG_MSG("[%s] Set brightness to %s successfully!\n",
            // led->name,
            // br_str);

        return 0;
}

static int led_init(void)
{
        int i;
        char tmp[NAME_MAX_LENGTH];
```

```
        for (i=0; i<ARRAY_SIZE(leds); i++) {
            memset(tmp, 0, sizeof(tmp));

            /* Backup all led triggers */
            if (led_get_trigger(&leds[i], tmp)) {
                return -1;
            }
            strcpy(leds[i].trigger_backup, tmp);

            /* Set all led triggers to 'none' */
            if (led_set_trigger(&leds[i], "none")) {
                return -1;
            }

            /* Set all brightness to 0 */
            if (led_set_brightness(&leds[i], 0)) {
                return -1;
            }
            leds[i].state = 0;
            leds[i].initialized = 1;
        }
        return 0;
}

static void led_restore(void)
{
        int i;

        /* set all brightness to '0', and restore triggers */
        for (i=0; i<ARRAY_SIZE(leds); i++) {
            if (leds[i].initialized) {
```

```
                led_set_brightness(&leds[i], 0);
                led_set_trigger(&leds[i], leds[i].trigger_backup);
            }
        }
}

/* Will be called if SIGINT(Ctrl+C) and SIGTERM(simple kill) signal is received */
static void signal_callback(int num)
{
        led_restore();
        exit(num);
}

int main(int argc, const char *argv[])
{
        /* Open button device */
        if(led_init()) {
            led_restore();
            return -1;
        }

        /* Register SIGINT(Ctrl+C) and SIGTERM(simple kill) signal and signal handl
er */
        signal(SIGINT, signal_callback);
        signal(SIGTERM, signal_callback);

        for (;;) {
            if (leds[0].state == 0) { /* if already ON, do nothing */
                leds[0].state = 1;
                led_set_brightness(&leds[0], leds[0].state);
            } else { /* this led should be turn OFF */
                leds[0].state = 0;
                led_set_brightness(&leds[0], leds[0].state);
```

```
            }
            usleep(LED_DELAY_US);
        }

        led_restore();

        return 0;
}
```

将上述代码拷贝到一个 led-test.c 文件，加载 SDK 环境变量到当前 shell:

```
# source /opt/petalinux/2020.1/environment-setup-aarch64-xilinx-linux
```

使用编译命令$CC 可生成可执行文件 led-test。

```
# $CC -o led-test led-test.c
```

只需在设备树 project-spec/meta-user/recipes-bsp/device-tree/file/system-user.dtsi 里配置 led3 即可。

```
......
    leds {
        compatible = "gpio-leds";
        led1 {
            label = "rs485_de";
            gpios = <&gpio 12 0>;
            linux,default-trigger = "gpio";
        };
        led2 {
            label = "wdt_en";
            gpios = <&gpio 33 0>;
            linux,default-trigger = "gpio";
        };
        led3 {
            label = "led_sys";
            gpios = <&gpio 43 0>;
```

```
                linux,default-trigger = "gpio";
            };
        };
......
```

编译生成 image.ub，替换 tf 卡中 image.ub，tf 卡启动模式启动系统。

将可执行文件通过网络，u 盘或者 tf 卡等传输介质拷贝到开发板的目录下，即可在终端下输入命令可直接运行。

```
# ./led-test
```

可以看到复位按键旁的 led 灯闪烁。

7. FPGA PL 功能的 linux 应用示例

通常我们在 fpga 中实现的 PL 功能，需要软件支持才能正常使用 PL 功能。本章以 axi-uartlite ip 核为例，讲述 FPGA PL 功能的 linux 应用流程。

7.1. axi-uartlite 中 Petalinux 软件实现流程

- **实现的功能大概步骤如下：**

（1）建立基础 bsp 包

（2）配置内核设备树

（3）配置内核驱动

（4）petalinux 编译

（5）测试 PL 的模块功能

- **建立基础 bsp 包**

将光盘中 05-ProgrammableLogic_Source/axi_uartlite.rar 示例工程生成的 design_1_wrapper.xsa 拷贝到目录/home/work/vivadoxsa/中

```
# petalinux-create -t project -s mys_zu5ev2020_4G_core.bsp
# petalinux-config --get-hw-description=/home/work/vivadoxsa/
```

- **配置内核设备树**

配置 petalinux 中 project-spec/meta-user/recipes-bsp/device-tree/files/system-user.dtsi 设备树文件如下：

```
/include/ "system-conf.dtsi"

#include <dt-bindings/media/xilinx-vip.h>

/ {
	chosen {
		bootargs = "console=ttyPS0,115200 earlyprintk root=/dev/mmcblk1p2 rw rootwait clk_ignore_unused";
		stdout-path = "serial0:115200n8";
	};
```

```
};

&sdhci0 {
	no-1-8-v;
};

&sdhci1 {
	disable-wp;
	no-1-8-v;
};

&qspi {
	flash@0 {
		compatible = "m25p80";
		#address-cells = <1>;
		#size-cells = <1>;
		reg = <0x0>;
		spi-tx-bus-width = <1>;
		spi-rx-bus-width = <4>;
		spi-max-frequency = <54000000>;
	};
};
```

Petalinux 会自动生成 axi uartlite 的设备树配置文件 components/plnx_workspace/device-tree/device-tree/pl.dtsi 如下：

```
/*
 * CAUTION: This file is automatically generated by Xilinx.
 * Version:
 * Today is: Fri Jun 11 08:34:40 2021
 */

/ {
	amba_pl: amba_pl@0 {
		#address-cells = <2>;
```

```
            #size-cells = <2>;
            compatible = "simple-bus";
            ranges ;
            axi_uartlite_0: serial@80000000 {
                clock-names = "s_axi_aclk";
                clocks = <&zynqmp_clk 71>;
                compatible = "xlnx,axi-uartlite-2.0", "xlnx,xps-uartlite-1.00.a";
                current-speed = <115200>;
                device_type = "serial";
                interrupt-names = "interrupt";
                interrupt-parent = <&gic>;
                interrupts = <0 89 1>;
                port-number = <1>;
                reg = <0x0 0x80000000 0x0 0x10000>;
                xlnx,baudrate = <0x1c200>;
                xlnx,data-bits = <0x8>;
                xlnx,odd-parity = <0x0>;
                xlnx,s-axi-aclk-freq-hz-d = "99.999001";
                xlnx,use-parity = <0x0>;
            };
        };
};
```

内核设备树已经配置完成。

- **配置内核驱动**

查看内核设备树，内核中已经配置了 axi uartlite 驱动。

```
# petalinux-config -c kernel
```

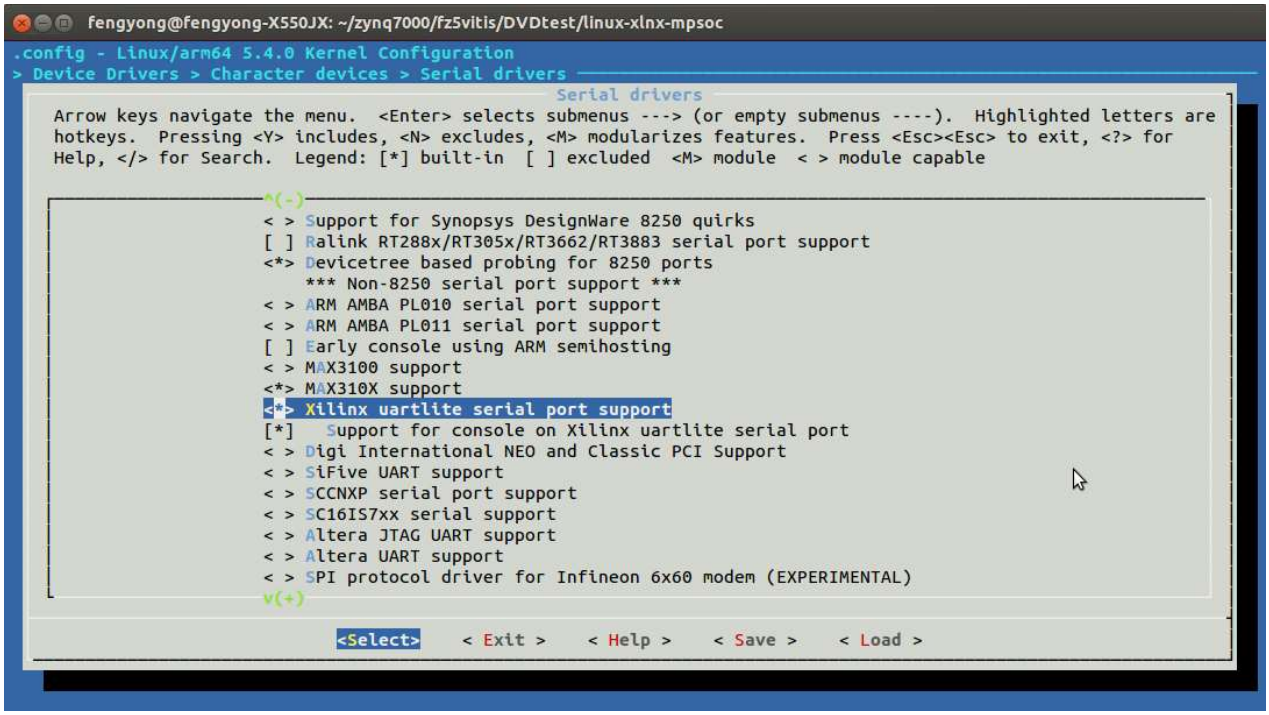

图 7-1 内核配置

- **Petalinux 编译**

```
# petalinux-build
```

- **测试 PL 的模块功能**

测试 PL 的模块功能，在硬件上，就是 RS232，即 MYS_ZU5EV 开发板的 J12 接口，测试时可以直接把 J12 接口的 1 脚和 3 脚短接，就可以在一块板子上进行 RS232 的测试了。

Rs232 的测试程序代码如下：

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <errno.h>
#include <getopt.h>
#include <string.h>
```

```
#define FALSE 1
#define TRUE 0

char *recchr="We received:\"";

void print_usage();

int speed_arr[] = {
	B921600, B460800, B230400, B115200, B57600, B38400, B19200,
	B9600, B4800, B2400, B1200, B300,
};

int name_arr[] = {
	921600, 460800, 230400, 115200, 57600, 38400,  19200,
	9600,  4800,  2400,  1200,  300,
};

void set_speed(int fd, int speed)
{
	int   i;
	int   status;
	struct termiosOpt;
	tcgetattr(fd, &Opt);

	for ( i= 0;  i<sizeof(speed_arr) / sizeof(int);  i++) {
		if  (speed == name_arr[i])   {
			tcflush(fd, TCIOFLUSH);
			cfsetispeed(&Opt, speed_arr[i]);
			cfsetospeed(&Opt, speed_arr[i]);
			status = tcsetattr(fd, TCSANOW, &Opt);
			if  (status != 0)
				perror("tcsetattr fd1");
				return;
```

```
            }
            tcflush(fd,TCIOFLUSH);
        }

        if (i == 12){
            printf("\tSorry, please set the correct baud rate!\n\n");
            print_usage(stderr, 1);
        }
}
/*
        *@brief  设置串口数据位，停止位和校验位
        *@param  fd 类型  int  打开的串口文件句柄
        *@param  databits 类型  int 数据位 取值为 7 或者 8
        *@param  stopbits 类型  int 停止位取值为 1 或者 2
        *@param  parity  类型  int  效验类型 取值为 N，E，O，S
*/
int set_Parity(int fd,intdatabits,intstopbits,int parity)
{
        struct termios options;
        if  (tcgetattr( fd,&options)  !=  0) {
            perror("SetupSerial 1");
            return(FALSE);
        }
        options.c_cflag&= ~CSIZE ;
        switch (databits) /*设置数据位数*/ {
        case 7:
            options.c_cflag |= CS7;
        break;
        case 8:
            options.c_cflag |= CS8;
        break;
        default:
            fprintf(stderr,"Unsupported data size\n");
```

```
        return (FALSE);
    }

    switch (parity) {
    case 'n':
    case 'N':
        options.c_cflag&= ~PARENB;   /* Clear parity enable */
        options.c_iflag&= ~INPCK;     /* Enable parity checking */
    break;
    case 'o':
    case 'O':
        options.c_cflag |= (PARODD | PARENB);  /* 设置为奇校验*/
        options.c_iflag |= INPCK;             /* Disnable parity checking */
    break;
    case 'e':
    case 'E':
        options.c_cflag |= PARENB;     /* Enable parity */
        options.c_cflag&= ~PARODD;   /* 转换为偶校验*/
        options.c_iflag |= INPCK;       /* Disnable parity checking */
    break;
    case 'S':
    case 's':  /*as no parity*/
        options.c_cflag&= ~PARENB;
        options.c_cflag&= ~CSTOPB;
    break;
    default:
        fprintf(stderr,"Unsupported parity\n");
        return (FALSE);
    }
    /* 设置停止位 */
    switch (stopbits) {
    case 1:
    options.c_cflag&= ~CSTOPB;
```

```
        break;
        case 2:
            options.c_cflag |= CSTOPB;
        break;
        default:
            fprintf(stderr,"Unsupported stop bits\n");
            return (FALSE);
        }
        /* Set input parity option */
        if (parity != 'n')
        options.c_iflag |= INPCK;
        options.c_cc[VTIME] = 150; // 15 seconds
        options.c_cc[VMIN] = 0;

        options.c_lflag&= ~(ECHO | ICANON);

        tcflush(fd,TCIFLUSH); /* Update the options and do it NOW */
        if (tcsetattr(fd,TCSANOW,&options) != 0) {
        perror("SetupSerial 3");
            return (FALSE);
        }
        return (TRUE);
}

/**
        *@breif 打开串口
*/
int OpenDev(char *Dev)
{
        int fd = open( Dev, O_RDWR );         //| O_NOCTTY | O_NDELAY
        if (-1 == fd) { /*设置数据位数*/
            perror("Can't Open Serial Port");
            return -1;
```

```
        } else
                return fd;
    }


    /* The name of this program */
    const char * program_name;

    /* Prints usage information for this program to STREAM (typically
     * stdout or stderr), and exit the program with EXIT_CODE. Does not
     * return.
     */

    void print_usage (FILE *stream, int exit_code)
    {
    fprintf(stream, "Usage: %s option [ dev... ] \n", program_name);
    fprintf(stream,
              "\t-h  --help     Display this usage information.\n"
              "\t-d  --device   The device ttyS[0-3] or ttySCMA[0-1]\n"
              "\t-b  --baudrate Set the baud rate you can select\n"
              "\t              [230400, 115200, 57600, 38400, 19200, 9600, 4800, 2400, 120
0, 300]\n"
              "\t-s  --string   Write the device data\n");
        exit(exit_code);
    }



    /*
          *@breif  main()
     */
    int main(int argc, char *argv[])
    {
```

```
        int  fd, next_option, havearg = 0;
        char *device;
        int i=0,j=0;
        int nread;              /* Read the counts of data */
        char buff[512];         /* Recvice data buffer */
        pid_tpid;
        char *xmit = "1234567890"; /* Default send data */
        int speed, send_mode = 0;
        const char *const short_options = "hd:s:b:m:";

        const struct option long_options[] = {
            { "help",   0, NULL, 'h'},
            { "device", 1, NULL, 'd'},
            { "string", 1, NULL, 's'},
            { "baudrate", 1, NULL, 'b'},
            { "send/recv mode", 0, NULL, 'm'},
            { NULL,     0, NULL, 0  }
        };

        program_name = argv[0];

        do {
            next_option = getopt_long (argc, argv, short_options, long_options, NU
LL);
            switch (next_option) {
                case 'h':
                    print_usage (stdout, 0);
                case 'd':
                    device = optarg;
                    havearg = 1;
                    break;
                case 'b':
                    speed = atoi(optarg);
```

```
                        break;
                case 's':
                        xmit = optarg;
                        havearg = 1;
                        break;
                case 'm':
                        send_mode = atoi(optarg);
                        break;
                case -1:
                        if (havearg)  break;
                case '?':
                        print_usage (stderr, 1);
                default:
                        abort ();
            }
        }while(next_option != -1);

        sleep(1);
        fd = OpenDev(device);

        if (fd> 0) {
            set_speed(fd, speed);
        } else {
            fprintf(stderr, "Error opening %s: %s\n", device, strerror(errno));
            exit(1);
        }

        if (set_Parity(fd,8,1,'N')== FALSE) {
            fprintf(stderr, "Set Parity Error\n");
            close(fd);
            exit(1);
        }
    #if 0
```

```
        pid = fork();

        if (pid< 0) {
            fprintf(stderr, "Error in fork!\n");
        } else if (pid == 0){
#endif
    if (send_mode){
            while(1) {
                printf("%s SEND: %s\n",device, xmit);
                write(fd, xmit, strlen(xmit));
                sleep(1);
                i++;
            }
}else {
            while(1) {
                nread = read(fd, buff, sizeof(buff));
                if (nread> 0) {
                    buff[nread] = '\0';
                    printf("%s RECV[%d]: %s\n", device, nread, buff);
                }
            }
        }
        close(fd);
        exit(0);
}
```

设置交叉编译工具

```
# source /opt/petalinux/2020.1/environment-setup-aarch64-xilinx-linux
```

编译 RS232 的测试程序

```
# $CC -o uart_test uart_test.c
```

使用编译产生的可执行程序 uart_test 来测试 axi uartlite 的功能。

```
# ./uart_test -d /dev/ttyUL1 -b 115200 -m 0 &

# ./uart_test -d /dev/ttyUL1 -b 115200 -m 1
/dev/ttyUL2 SEND: 1234567890
/dev/ttyUL2 RECV[1]: 1
/dev/ttyUL2 RECV[1]: 2
/dev/ttyUL2 RECV[1]: 3
/dev/ttyUL2 RECV[1]: 4
/dev/ttyUL2 RECV[1]: 5
/dev/ttyUL2 RECV[1]: 6
/dev/ttyUL2 RECV[1]: 7
/dev/ttyUL2 RECV[1]: 8
/dev/ttyUL2 RECV[1]: 9
/dev/ttyUL2 RECV[1]: 0
/dev/ttyUL2 SEND: 1234567890
/dev/ttyUL2 RECV[1]: 1
/dev/ttyUL2 RECV[1]: 2
/dev/ttyUL2 RECV[1]: 3
/dev/ttyUL2 RECV[1]: 4
/dev/ttyUL2 RECV[1]: 5
/dev/ttyUL2 RECV[1]: 6
/dev/ttyUL2 RECV[1]: 7
/dev/ttyUL2 RECV[1]: 8
/dev/ttyUL2 RECV[1]: 9
/dev/ttyUL2 RECV[1]: 0
```

通过测试的信息可以看到 axi uartlite 串口通讯正常，发送的数据都正常收到。

8. 如何添加您的应用

Linux 应用的移植通常分为两个阶段，开发调试阶段和生产部署阶段。开发调试阶段我们可以使用米尔构建好的 SDK 对我们编写好的应用进行交叉编译，然后远程拷贝到目标主机进行测试。生产部署阶段需要为应用编写配方文件，构建生产镜像。

8.1. 基于 Makefile 的应用

Makefile 其实就是一个文档，里面定义了一系列的编译规则，它记录了原始码如何编译的详细信息！Makefile 一旦写好，只需要一个 make 命令，整个工程完全自动编译，极大的提高了软件开发的效率。在开发 Linux 程序时，不管是内核，驱动，应用，Makefile 得到了普遍的应用。

make 是一个命令工具，是一个解释 makefile 中指令的命令工具。它可以简化编译过程里面所下达的指令，当执行 make 时，make 会在当前的目录下搜寻 Makefile (or makefile) 这个文本文件，执行对应的操作。make 会自动的判别原始文件是否经过了变动，从而自动重新编译更改的源代码。

下列将以一个实际的示例（在 MYS-ZU5EV 开发板上实现控制 LED 灯开关）来讲述 Makfile 的编写与 make 的执行过程。Makefile 有其自身的一套规则。

```
target ... : prerequisites ...
        command
```

- target 可以是一个 object file(目标文件)，也可以是一个执行文件，还可以是一个标签（label）。
- prerequisites 就是要生成那个 target 所需要的文件或是目标。
- command 也就是 make 需要执行的命令。

```
TARGET = $(notdir $(CURDIR))
objs := $(patsubst %c, %o, $(shell ls *.c))
$(TARGET)_test:$(objs)
            $(CC) -o $@ $^
%.o:%.c
            $(CC) -c -o $@ $<
clean:
            rm -f  $(TARGET)_test *.all *.o
      ${CC} -I . -c gpioctrl.c
```

- $(notdir $(path))： 表示把 path 目录去掉路径名，只留当前目录名，比如当前 Makefile 目录为*/home/examples/gpioapp*，执行为就变为 TARGET = gpioapp
- $(patsubst pattern, replacement,text) ：用 replacement 替换 text 中符合格式"pattern" 的字符，如$(patsubst %c, %o, $(shell ls *.c))，表示先列出当前目录后缀为.c 的文件，然后换成后缀为.o
- CC：C 编译器的名称
- CXX: C++编译器的名称
- clean: 是一个约定的目标

gpioctrl.c 实现代码如下：

```
//gpioctrl.c
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>

/*
* ./gpioctrl /sys/class/leds/led_sys on
* ./gpioctrl /sys/class/leds/led_sys off
 */
int main(int argc, char **argv)
{
        int fd;
        char status;

        /* 1. 判断参数 */
        if (argc != 3)
        {
                printf("Usage: %s <dev><on | off>\n", argv[0]);
                return -1;
```

```
        }

        /* 2. 打开文件 */
        fd = open(argv[1], O_RDWR);
        if (fd == -1)
        {
            printf("can not open file %s\n", argv[1]);
            return -1;
        }

        /* 3. 写文件 */
        if (0 == strcmp(argv[2], "on"))
        {
            status = '0';
            write(fd, &status, 1);
        }
        else
        {
            status = '1';
            write(fd, &status, 1);
        }

        close(fd);

        return 0;
}
```

使用 make 命令进行编译并生成目标机器上的可执行文件 gpioapp_test。

加载 SDK 环境变量到当前 shell:

```
# source /opt/petalinux/2020.1/environment-setup-aarch64-xilinx-linux
```

执行 make:

```
# make CC=aarch64-xilinx-linux-gcc
```

从上一个命令的结果可以看到，使用的编译器是通过设置脚本中定义的 CC 变量建立的编译器。

将 gpioapp_test 可执行文件通过网络，u 盘或者 tf 卡等传输介质拷贝到开发板的目录下:

```
# ./gpioapp_test /sys/class/leds/led_sys/brightness on
# ./gpioapp_test /sys/class/leds/led_sys/brightness off
```

在 MYS-ZU5EV 开发板可控制复位按键旁的 LED 灯开关。

8.2. 基于 Qt 的应用

Qt 是一个跨平台的图形应用开发框架，被应用在不同尺寸设备和平台上，同时提供不同版权版本供用户选择。MYS-ZU5EV 使用 Qt 5.13.2 版本进行应用开发。在 Qt 应用开发中，推荐使用 QtCreator 集成开发环境，可以在 Linux PC 下开发 Qt 应用，自动化地交叉编译为开发板的 ARM 架构程序。

1）QtCreator 安装与配置

从 QT 官网或 MYIR 官方包获得 qtcreator 安装包 QT 官网下载：http://download.qt.io/development_releases/qtcreator/4.1/4.1.0-rc1/。

QtCreator 安装包是一个二进制程序，直接执行就可以完成安装 ./qt-creator-opensource-linux-x86_64-4.1.0-rc1.run 即可，如需获得安装与配置详情请参考光盘中的文档《MYS-ZU5EV_QT 应用开发笔记》。

8.3. 应用程序开机自启动

1）应用程序在 Petalinux 的配置

通常我们的应用还需要实现开机自启动，这些也可以在配方中实现。下面以一个稍微复杂一点的 FTP 服务应用为例说明如何使用 Petalinux 构建包含特定应用的生产镜像，这里的 FTP 服务程序采用的是开源的 Proftpd，各个版本源码位于 ftp://ftp.proftpd.org/distrib/source/。

在我们从头开始写一个配方之前，我们可以在Petalinux已有层components/yocto/layers中查找有没有现成或者类似应用的配方，查找方法如下：

```
# find components/yocto/layers –name proftpd
```

我们也可以在 OpenEmbedded 的官方网站层索引（http://layers.openembedded.org/layerindex/branch/master/layers/）中查找是否有同样或者类似应用的配方。

本节重点描述如何移植 FTP 服务到目标机器中的方法。通过搜索 Petalinux 层发现已经存在 proftpd 的配方，只是没有添加到系统镜像中。下面详细描述具体的移植过程。

- **查找 Petalinux 的 proftpd 配方**

```
# find components/yocto/layers-name proftpd
components/yocto/layers/meta-openembedded/meta-networking/recipes-daemons/proftpd
```

注：这里可以看到 Petalinux 项目中已经存在 proftpd 配方。

- **打包 proftpd 到文件系统**

在<plnx-proj-root>/project-spec/meta-user/conf/user-rootfsconfig 添加一行：

```
CONFIG_proftpd
```

然后在 rootfs 的配置中使能 proftpd:

```
# petalinux-config -c rootfs
```

选择 user packages ->proftpd。

- **重新构建镜像**

```
# petalinux-build
```

- **烧录新镜像**

系统构建完成之后，需重新烧录镜像并查看 proftpd 服务是否运行：

```
# ps -axu | grep proftpd
nobody    673  0.0  0.1   3256  1336 ?        Ss   07:49   0:00 proftpd: (accepting con
nections)
root      741  0.0  0.0   2020   424 ttyPS0    S+   07:50   0:00 grep proftpd
```

这里补充说明一下 FTP 的账户设置。FTP 客户端有三种类型登录账户，分别为匿名账户，普通账户和 root 账户。

- **匿名账户**

用户名为 ftp ，不需要设置密码，用户登录后可以查看系统/var/lib/ftp 目录下的内容，默认没有写权限。由于系统默认不存在/var/lib/ftp 目录，所以需要用户在目标机器上创建一个目录/var/lib/ftp。为了尽量不修改 meta-openembbed, 我们可以通过为 proftpd 配方添加 Append 文件“proftpd_%.bbappend”来实现/var/lib/ftp 目录的创建。

```
do_install_append() {
        install -m 755 -d ${D}/var/lib/${FTPUSER}
        chownftp:ftp ${D}/var/lib/${FTPUSER}
}
```

编辑好的 proftpd_%.bbappend”需要放置到 meta-user 下面的 recipes-daemons\proftpd 目录。

- **普通账户**

在目标机器上使用 useradd 和 passwd 命令可以创建普通用户，并设置用户密码之后，客户端也可以使用该普通账户登录到该用户的 HOME 目录。Petalinux 默认会创建一个用户名和密码均为 petalinux 的普通账号。

- **root 账户**

如果需要开放 root 账户登录 FTP 服务器，需要先修改/etc/proftpd.conf 文件，在文件中增加一行配置 "RootLogin on"。与此同时，也需要为 root 账户设置密码，重启 proftpd 服务之后，客户端也可以使用 root 账户登录到目标机器上。

```
# /etc/init.d/proftpd restart
```

注意：修改/etc/proftpd.conf 使能 root 账户登录仅用于测试目的，关于/etc/proftpd.conf 的更多配置，参见 http://www.proftpd.org/docs/example-conf.html。

2）实现应用程序的自启动

本节将以 proftpd 配方为例从配方源码的层面介绍如何添加应用程序配方并实现程序的开机自启动。proftpd 配方位于 petalinux 项目 components/yocto/layers/meta-openembedded/meta-networking/recipes-daemons/proftpd，目录结构如下。

```
├── files
│   ├── basic.conf.patch
│   ├── build_fixup.patch
│   ├── close-RequireValidShell-check.patch
│   ├── contrib.patch
│   ├── default
│   ├── proftpd-basic.init
│   └── proftpd.service
└── proftpd_1.3.6.bb

1 directory, 8 files
```

- proftpd_1.3.6.bb 为构建 proftpd 服务的配方
- proftpd.service 为开机自启动服务
- proftpd-basic.init 为 proftpd 的启动脚本

proftpd_1.3.6.bb 配方中指定了获取 proftpd 服务程序的源代码路径以及针对该版本源码的一些补丁文件：

```
SRC_URI = "ftp://ftp.proftpd.org/distrib/source/${BPN}-${PV}.tar.gz \
           file://basic.conf.patch \
           file://proftpd-basic.init \
           file://default \
           file://close-RequireValidShell-check.patch \
file://contrib.patch  \
           file://build_fixup.patch \
           file://proftpd.service \
"
```

配方中还指定了 proftpd 的配置（do_configure)和安装过程（do_install）：

```
# proftpd uses libltdl which currently makes configuring using
# autotools.bbclass a pain...
do_configure () {
```

```
    install -m 0755 ${STAGING_DATADIR_NATIVE}/gnu-config/config.guess ${S}
    install -m 0755 ${STAGING_DATADIR_NATIVE}/gnu-config/config.sub ${S}
    oe_runconf
    cp ${STAGING_BINDIR_CROSS}/${HOST_SYS}-libtool ${S}/libtool
}

FTPUSER = "ftp"
FTPGROUP = "ftp"

do_install () {
oe_runmake DESTDIR=${D} install
rmdir ${D}${libdir}/proftpd ${D}${datadir}/locale
    [ -d ${D}${libexecdir} ]&&rmdir ${D}${libexecdir}
    sed -i '/ *User[ \t]*/s/ftp/${FTPUSER}/' ${D}${sysconfdir}/proftpd.conf
    sed -i '/ *Group[ \t]*/s/ftp/${FTPGROUP}/' ${D}${sysconfdir}/proftpd.conf
    install -d ${D}${sysconfdir}/init.d
    install -m 0755 ${WORKDIR}/proftpd-basic.init ${D}${sysconfdir}/init.d/proftpd
    sed -i 's!/usr/sbin/!${sbindir}/!g' ${D}${sysconfdir}/init.d/proftpd
    sed -i 's!/etc/!${sysconfdir}/!g' ${D}${sysconfdir}/init.d/proftpd
    sed -i 's!/var/!${localstatedir}/!g' ${D}${sysconfdir}/init.d/proftpd
    sed -i 's!^PATH=.*!PATH=${base_sbindir}:${base_bindir}:${sbindir}:${bindir}!'
${D}${sysconfdir}/init.d/proftpd

    install -d ${D}${sysconfdir}/default
    install -m 0755 ${WORKDIR}/default ${D}${sysconfdir}/default/proftpd

    # create the pub directory
mkdir -p ${D}/home/${FTPUSER}/pub/
chown -R ${FTPUSER}:${FTPGROUP} ${D}/home/${FTPUSER}/pub
    if ${@bb.utils.contains('DISTRO_FEATURES', 'pam', 'true', 'false', d)}; then
        # installproftpd pam configuration
        install -d ${D}${sysconfdir}/pam.d
```

```
        install -m 644 ${S}/contrib/dist/rpm/ftp.pamd ${D}${sysconfdir}/pam.d/proft
pd
        sed -i '/ftpusers/d' ${D}${sysconfdir}/pam.d/proftpd
        # specify the user Authentication config
        sed -i '/^MaxInstances/a\AuthPAM                    on\nAuthPAMConfigproft
pd' \
            ${D}${sysconfdir}/proftpd.conf
    fi

    install -d ${D}/${systemd_unitdir}/system
    install -m 644 ${WORKDIR}/proftpd.service ${D}/${systemd_unitdir}/system
    sed -e 's,@BASE_SBINDIR@,${base_sbindir},g' \
        -e 's,@SYSCONFDIR@,${sysconfdir},g' \
        -e 's,@SBINDIR@,${sbindir},g' \
        -i ${D}${systemd_unitdir}/system/*.service

    sed -e 's|--sysroot=${STAGING_DIR_HOST}||g' \
        -e 's|${STAGING_DIR_NATIVE}||g' \
        -e 's|-fdebug-prefix-map=[^ ]*||g' \
        -e 's|-fmacro-prefix-map=[^ ]*||g' \
        -i ${D}/${bindir}/prxs

    # ftpmailperl script, which reads the proftpd log file and sends
    # automatic email notifications once an upload finishs,
    # depends on an old perlMail::Sendmail
    # The Mail::Sendmail has not been maintained for almost 10 years
    # Other distribution not ship with ftpmail, so do the same to
    # avoid confusion about having it fails to run
    rm -rf ${D}${bindir}/ftpmail
    rm -rf ${D}${mandir}/man1/ftpmail.1
}
```

这两个函数对应 BitBake 构建过程的 config 和 install 任务(关于任务的更多信息，参见 https://docs.yoctoproject.org/ref-manual/tasks.html)。

当前目标机器采用 init 作为初始化管理子系统，init 是一个 Linux 系统基础组件的集合，提供了一个系统和服务管理器，运行为 PID 1 并负责启动其它程序。

Proftpd 服务的开机自启动服务文件 proftpd-basic.init 内容如下：

```
#!/bin/sh

### BEGIN INIT INFO
# Provides:          proftpd
# Required-Start:    $remote_fs $syslog $local_fs $network
# Required-Stop:     $remote_fs $syslog $local_fs $network
# Should-Start:      $named
# Should-Stop:       $named
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: Starts ProFTPD daemon
# Description:       This script runs the FTP service offered
#                    by the ProFTPD daemon
### END INIT INFO

# Start the proftpd FTP daemon.

PATH=/bin:/usr/bin:/sbin:/usr/sbin
DAEMON=/usr/sbin/proftpd
NAME=proftpd

# Defaults
RUN="no"
OPTIONS=""
CONFIG_FILE=/etc/proftpd.conf

PIDFILE=`grep -i '^pidfile' $CONFIG_FILE|awk '{ print $2 }'`
if [ "x$PIDFILE" = "x" ];
then
    PIDFILE=/var/run/proftpd.pid
```

```
fi

# Read config (will override defaults)
[ -r /etc/default/proftpd ] && . /etc/default/proftpd

trap "" 1
trap "" 15

test -f $DAEMON || exit 0

. /etc/init.d/functions

#
# Servertype could be inetd|standalone|none.
# In all cases check against inetd and xinetd support.
#
if ! egrep -qi "^[[:space:]]*ServerType.*standalone" $CONFIG_FILE
then
   if egrep -qi "server[[:space:]]*=[[:space:]]*/usr/sbin/proftpd" /etc/xinetd.conf 2>
/dev/null || \
      egrep -qi "server[[:space:]]*=[[:space:]]*/usr/sbin/proftpd" /etc/xinetd.d/* 2>/
dev/null || \
      egrep -qi "^ftp.*/usr/sbin/proftpd" /etc/inetd.conf 2>/dev/null
   then
      RUN="no"
      INETD="yes"
   else
      if ! egrep -qi "^[[:space:]]*ServerType.*inetd" $CONFIG_FILE
      then
         RUN="yes"
         INETD="no"
      else
         RUN="no"
```

```
            INETD="no"
        fi
    fi
fi

# /var/run could be on a tmpfs

[ ! -d /var/run/proftpd ] && mkdir /var/run/proftpd

inetd_check()
{
    if [ ! -x /usr/sbin/inetd -a ! -x /usr/sbin/xinetd ]; then
        echo "Neither inetd nor xinetd appears installed: check your configuration."
    fi
}

start()
{
    set -e
    echo -n "Starting ftp server $NAME... "
    start-stop-daemon --start --quiet --pidfile "$PIDFILE" --oknodo --exec $DAEM
ON -- -c $CONFIG_FILE $OPTIONS
    echo "done."
}

signal()
{

    if [ "$1" = "stop" ]; then
        SIGNAL="TERM"
        echo -n "Stopping ftp server $NAME... "
    else
        if [ "$1" = "reload" ]; then
```

```
            SIGNAL="HUP"
            echo -n "Reloading ftp server $NAME... "
        else
            echo "ERR: wrong parameter given to signal()"
            exit 1
        fi
    fi
    if [ -f "$PIDFILE" ]; then
        start-stop-daemon --stop --signal $SIGNAL --quiet --pidfile "$PIDFILE"
        if [ $? = 0 ]; then
            echo "done."
            return
        else
            SIGNAL="KILL"
            start-stop-daemon --stop --signal $SIGNAL --quiet --pidfile "$PIDFILE"
            if [ $? != 0 ]; then
                echo
                [ $2 != 0 ] || exit 0
            else
                echo "done."
                return
            fi
        fi
        if [ "$SIGNAL" = "KILL" ]; then
            rm -f "$PIDFILE"
        fi
    else
        echo "done."
        return
    fi
}

case "$1" in
```

```
    start)
        if [ "x$RUN" = "xyes" ] ; then
            start
        else
            if [ "x$INETD" = "xyes" ] ; then
                echo "ProFTPD is started from inetd/xinetd."
                inetd_check
            else
                echo "ProFTPD warning: cannot start neither in standalone nor in inetd/
xinetd mode. Check your configuration."
            fi
        fi
        ;;

    force-start)
        if [ "x$INETD" = "xyes" ] ; then
            echo "Warning: ProFTPD is started from inetd/xinetd (trying to start anywa
y)."
            inetd_check
        fi
        start
        ;;

    stop)
        if [ "x$RUN" = "xyes" ] ; then
            signal stop 0
        else
            if [ "x$INETD" = "xyes" ] ; then
                echo "ProFTPD is started from inetd/xinetd."
                inetd_check
            else
                echo "ProFTPD warning: cannot start neither in standalone nor in inetd/
xinetd mode. Check your configuration."
```

```
        fi
    fi
    ;;

  force-stop)
    if [ "x$INETD" = "xyes" ] ; then
        echo "Warning: ProFTPD is started from inetd/xinetd (trying to kill anywa
y)."
        inetd_check
    fi
    signal stop 0
    ;;

  reload)
    signal reload 0
    ;;

  force-reload|restart)
    if [ "x$RUN" = "xyes" ] ; then
        signal stop 1
        sleep 2
        start
    else
        if [ "x$INETD" = "xyes" ] ; then
            echo "ProFTPD is started from inetd/xinetd."
            inetd_check
        else
            echo "ProFTPD warning: cannot start neither in standalone nor in inetd/
xinetd mode. Check your configuration."
        fi
    fi
    ;;
```

```
    status)
        if [ "x$INETD" = "xyes" ] ; then
            echo "ProFTPD is started from inetd/xinetd."
            inetd_check
            exit 0
        else
            if [ -f "$PIDFILE" ]; then
                pid=$(cat $PIDFILE)
            else
                pid="x"
            fi
            if [ `pidof proftpd|grep "$pid"|wc -l` -ne 0 ] ; then
                echo "ProFTPD is started in standalone mode, currently running."
                exit 0
            else
                echo "ProFTPD is started in standalone mode, currently not running."
                exit 3
            fi
        fi
        ;;

    check-config)
        $DAEMON -t >/dev/null && echo "ProFTPD configuration OK" && exit 0
        exit 1
        ;;

    *)
        echo "Usage: /etc/init.d/$NAME {start|status|force-start|stop|force-stop|reloa
d|restart|force-reload|check-config}"
        exit 1
        ;;
esac
```

```
exit 0
```

- start()是启动服务时执行的函数。
- stop()是结束服务时执行的函数。
- restart()是重启服务时执行的函数。
- status()是查询进程状态时执行的函数。

用户在添加自己编写的应用时，可参考 UG1144 手册第 8 章节的“Application Auto Run at Startup”小节。

8.4. 应用程序示例

光盘中提供了常用的演示程序，程序以及源码都位于 04_Sources\Example 目录中。

8.4.1. CAN 应用示例

CAN 测试需要两块 MYS_ZU5EV 板子的 can 接口对接来测试。

进入 04_Sources\Example\can 目录，直接 make 编译代码就可产生可执行文件。

```
# source /opt/petalinux/2020.1/environment-setup-aarch64-xilinx-linux
# make
```

将编译产生的可执行文件 can_receive、can_send 拷贝到两个 tf 卡中，将两个 tf 卡分别插入两块板子，两块板子上电启动即可测试 can 通讯功能：

其中一块板子作为接收端进行如下操作：

```
# ip link set can0 up type can bitrate 500000
# ./can_receive
```

另一块板子作为发送端进行如下操作：

```
# ip link set can0 up type can bitrate 500000
# ./can_send -d can0 -i 123 33 44 55
```

8.4.2. I2C 应用示例

I2C 测试按照如下方式测试。

进入 04_Sources\Example\i2c 目录，直接 make 编译代码就可产生可执行文件。

```
# source /opt/petalinux/2020.1/environment-setup-aarch64-xilinx-linux
# make
```

将编译产生的可执行文件 i2c_flash 拷贝到 tf 卡中，启动板子测试：

```
# ./i2c_flash /dev/i2c-1
```

8.4.3. 网络应用示例

可以用板子和电脑来测试网络通讯。

进入 04_Sources\Example\network 目录，直接 make 编译代码就可产生可执行文件。

```
# source /opt/petalinux/2020.1/environment-setup-aarch64-xilinx-linux
# make
```

将编译产生的可执行文件 arm_server 拷贝到 tf 卡中，启动板子操作如下：

```
# ./arm_server
```

电脑端的 ubuntu 系统中输入如下命令，192.168.xxx.xxx 为板子的 ip 地址：

```
$ ./pc_client 192.168.xxx.xxx
```

8.4.4. Uart 应用示例

可以将板子的串口 J12 接口的 1 脚和 3 脚短接，进行测试。

进入 04_Sources\Example\uart 目录，直接 make 编译代码就可产生可执行文件。

```
# source /opt/petalinux/2020.1/environment-setup-aarch64-xilinx-linux
# make
```

将编译产生的可执行文件 uart_test 拷贝到 tf 卡中，启动板子操作如下：

```
# ./uart_test -d /dev/ttyUL1 -b 115200 -m 0 &
# ./uart_test -d /dev/ttyUL1 -b 115200 -m 1
/dev/ttyUL1 SEND: 1234567890
/dev/ttyUL1 RECV[1]: 1
/dev/ttyUL1 RECV[1]: 2
/dev/ttyUL1 RECV[1]: 3
/dev/ttyUL1 RECV[1]: 4
/dev/ttyUL1 RECV[1]: 5
/dev/ttyUL1 RECV[1]: 6
/dev/ttyUL1 RECV[1]: 7
/dev/ttyUL1 RECV[1]: 8
/dev/ttyUL1 RECV[1]: 9
/dev/ttyUL1 RECV[1]: 0
```

8.4.5. Framebuffer 应用示例

通过 framebuffer 程序，测试 DP 显示。

进入 04_Sources\Example\framebuffer 目录，直接 make 编译代码就可产生可执行文件。

```
# source /opt/petalinux/2020.1/environment-setup-aarch64-xilinx-linux
# make
```

将编译产生的可执行文件 framebuffer_test 拷贝到 tf 卡中，启动板子操作如下：

```
# ./framebuffer_test
```

8.4.6. HDMIin 应用示例

通过 qt-hdmi 程序，测试 hdmi 输入图像通过 DP 来显示。

进入 04_Sources\Example\qt-hdmi 目录，直接 make 编译代码就可产生可执行文件。

```
# source /opt/petalinux/2020.1/environment-setup-aarch64-xilinx-linux
# /opt/petalinux/2020.1/sysroots/x86_64-petalinux-linux/usr/bin/qmake
# make
```

按照本文中“4.1. 制作 SD 卡启动器”章节的方法将光盘中的 02_Images 目录中的 mys-zu5ev-full.img.gz 镜像烧写到 tf 卡中 ，tf 卡方式启动板子。将编译产生的可执行文件 qt-hdmi 拷贝到 tf 卡中，启动板子操作如下：

```
# media-ctl -v --set-format '"a0010000.v_tpg":0 [RBG24 1920x1080 field:none]'
# export DISPLAY=:0.0
# ./qt-hdmi
```

8.4.7. MIPI 摄像头应用示例

MIPI 摄像头应用需要有 IMX334 mipi 摄像头才能测试。

通过 qt-camera 程序，测试 IMX334 mipi 摄像头输入图像通过 DP 来显示。

进入 04_Sources\Example\qt-camera 目录，直接 make 编译代码就可产生可执行文件。

```
# source /opt/petalinux/2020.1/environment-setup-aarch64-xilinx-linux
# /opt/petalinux/2020.1/sysroots/x86_64-petalinux-linux/usr/bin/qmake
# make
```

按照本文中“4.1. 制作 SD 卡启动器”章节的方法将光盘中的 02_Images 目录中的 mys-zu5ev-mipi.img.gz 镜像烧写到 tf 卡中 ，tf 卡方式启动板子。将编译产生的可执行文件 qt-camera 拷贝到 tf 卡中，启动板子操作如下：

```
# export DISPLAY=:0.0
# ./qt-camera
```

8.4.8. VCU 应用示例

ZU5EV 芯片的 VCU 支持多标准视频编码和解码，包括支持高效视频编码(HEVC)和高级视频编码(AVC)H.264 标准。该单元包含编码（压缩）和解码（解压缩）功能。

在 FPGA 中实现 VCU 的逻辑功能，具体实现过程请参考光盘中《MYS-ZU5EV FPGA 指导手册》中“6.2 VCU”章节的内容。然后通过 petalinux 实现软件功能。

- **VCU 中 Petalinux 的软件实现流过程：**

（1）将光盘中 05-ProgrammableLogic_Source/project_fz5_vcu_707.rar 示例工程生成的 design_1_wrapper.xsa 拷贝到目录/home/work/vivadoxsa/中。

```
# petalinux-create -t project -s mys_zu5ev2020_4G_full.bsp
# petalinux-config --get-hw-description=/home/work/vivadoxsa/
```

（2）按如下方法配置 rootfs 系统：

```
# petalinux-config –c rootfs
```

配置 libmali；

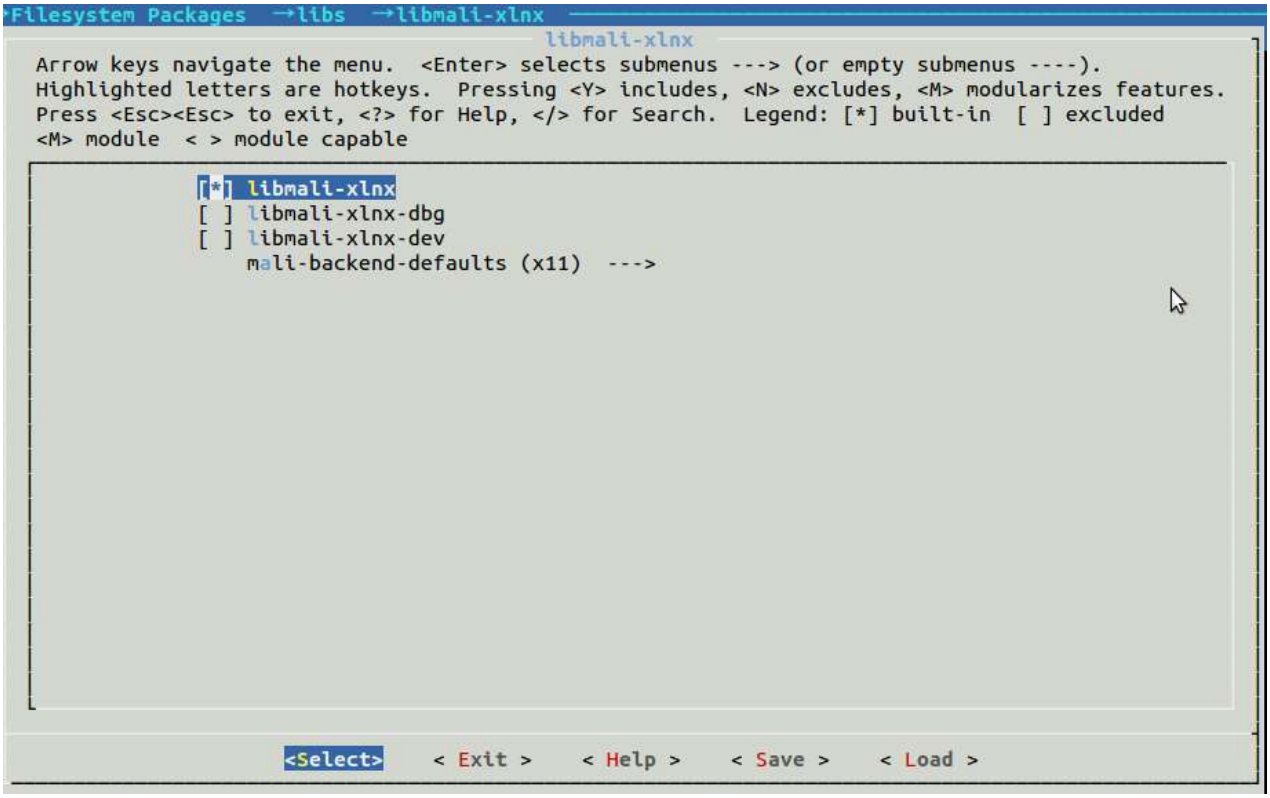

图 8-1 配置 libmaili

配置 multimedia；

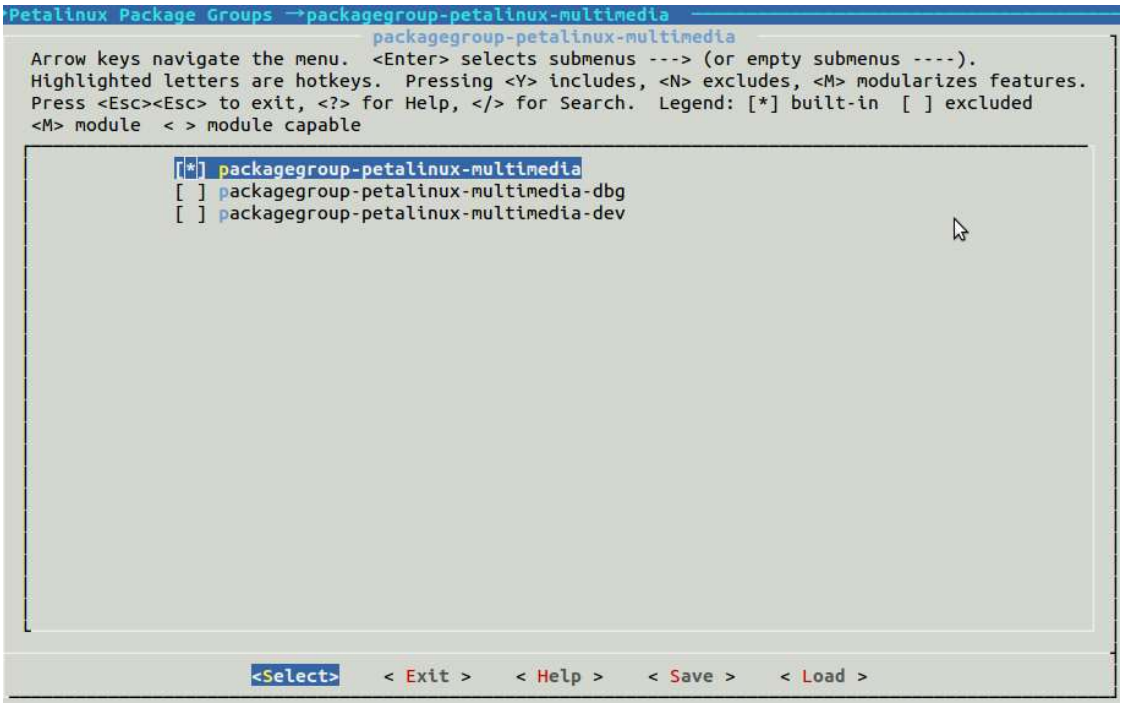

图 8-2 配置 multimedia

配置 gstreamer-vcu-examples；

在 project-spec/meta-user/conf/user-rootfsconfig 添加一行；

```
CONFIG_gstreamer-vcu-examples
```

然后在 rootfs 配置中使能 gstreamer-vcu-examples。

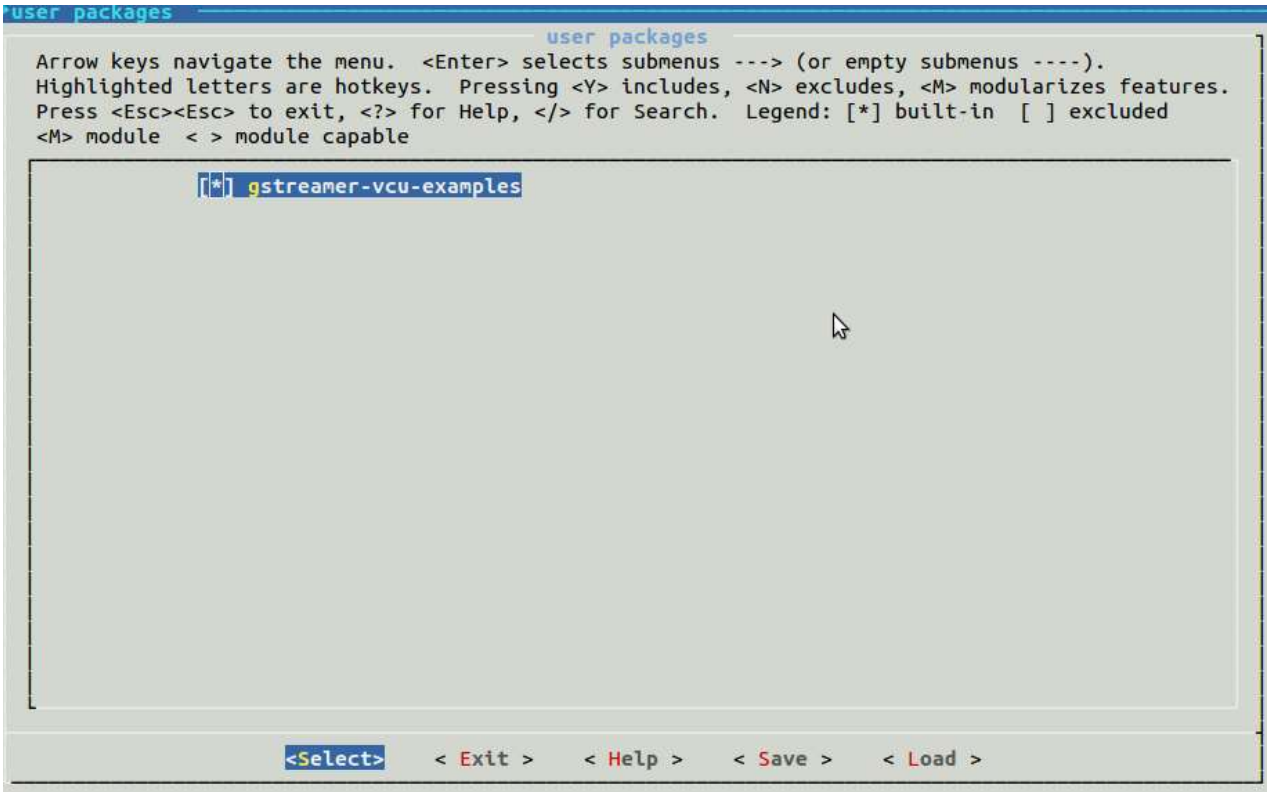

图 8-3 配置 gstreamer-vcu-examples

（3）配置 petalinux 中 project-spec/meta-user/recipes-bsp/device-tree/files/system-user.dtsi 设备树文件如下：

```
/include/ "system-conf.dtsi"

/ {
	chosen {
		bootargs = "console=ttyPS0,115200 earlyprintk root=/dev/mmcblk1p2 rw rootwait clk_ignore_unused";
		stdout-path = "serial0:115200n8";
	};

	leds {
```

```
            compatible = "gpio-leds";
            led1 {
                label = "rs485_de";
                gpios = <&gpio 12 0>;
                linux,default-trigger = "gpio";
            };
            led2 {
                label = "wdt_en";
                gpios = <&gpio 33 0>;
                linux,default-trigger = "gpio";
            };
            led3 {
                label = "led_sys";
                gpios = <&gpio 43 0>;
                linux,default-trigger = "gpio";
            };
        };

        watchdog {
            compatible = "gpio-watchdog";
        };
};

&gem3 {
        phy-handle = <&phy0>;
        phy-mode = "rgmii-id";
        phy0: phy@21 {
            reg = <4>;
        };
};

&sdhci0 {
        no-1-8-v;
```

```
};

&sdhci1 {
        disable-wp;
        no-1-8-v;
};

&qspi {
        flash@0 {
                compatible = "m25p80";
                #address-cells = <1>;
                #size-cells = <1>;
                reg = <0x0>;
                spi-tx-bus-width = <1>;
                spi-rx-bus-width = <4>;
                spi-max-frequency = <54000000>;
        };
};
```

（4）编译 petalinux bsp 就可以生成 vcu 的 petalinux 软件：

```
# petalinux-build
```

- **VCU 测试**

按照本文中“4.1. 制作 SD 卡启动器”章节的方法将光盘中的 04_Sources\Example\vcu 目录中的 mys-zu5ev-vcu.img.gz 镜像烧写到 tf 卡中 ，tf 卡方式启动板子，进入系统操作如下，就可以使用 vcu 解码功能播放 mp4 视频。

```
# vcu-demo-decode-display.sh -i /mnt/sd-mmcblk1p1/test.mp4
```

9. 参考资料

- **Linux kernel 开源社区**

  https://www.kernel.org/

- **Xilinx 官网**

  https://www.xilinx.com/

- **Petalinux 中涉及到 yocto 资源的参考网站**

  https://www.yoctoproject.org/

附录一联系我们

深圳总部

负责区域：广东 / 四川 / 重庆 / 湖南 / 广西 / 云南 / 贵州 / 海南 / 香港 / 澳门
电话：0755-25622735 18924653967
地址：深圳市龙岗区坂田街道发达路云里智能园 2 栋 6 楼 604 室

上海办事处

负责区域：上海 / 湖北 / 江苏 / 浙江 / 安徽 / 福建 / 江西
电话：021-62087019 18924632515
地址：上海市普陀区中江路 106 号北岸长风I 座 302

北京办事处

负责区域：北京/天津/陕西/辽宁/山东/河南/河北/黑龙江/吉林/山西/甘肃/内蒙古/宁夏
电话：010-84675491 13316862895
地址：北京市昌平区东小口镇中滩村润枫欣尚 1 号楼 505

销售联系方式

网址：www.myir-tech.com
邮箱：sales.cn@myirtech.com

技术支持联系方式

电话：0755-22316235（深圳），027-59621647/027-59621648（武汉）
邮箱：support.cn@myirtech.com

如果您通过邮件获取帮助时，请使用以下格式书写邮件标题：

[公司名称/个人--开发板型号]问题概述

这样可以使我们更快速跟进您的问题，以便相应开发组可以处理您的问题。

附录二售后服务与技术支持

凡是通过米尔电子直接购买或经米尔电子授权的正规代理商处购买的米尔电子全系列产品，均可享受以下权益：

1、6 个月免费保修服务周期
2、终身免费技术支持服务
3、终身维修服务
4、免费享有所购买产品配套的软件升级服务
5、免费享有所购买产品配套的软件源代码，以及米尔科技开发的部分软件源代码
6、可直接从米尔科技购买主要芯片样品，简单、方便、快速；免去从代理商处购买时，漫长的等待周期
7、自购买之日起，即成为米尔科技永久客户，享有再次购买米尔科技任何一款软硬件产品的优惠政策
8、OEM/ODM 服务

如有以下情况之一，则不享有免费保修服务：
1、超过免费保修服务周期
2、无产品序列号或无产品有效购买单据
3、进液、受潮、发霉或腐蚀
4、受撞击、挤压、摔落、刮伤等非产品本身质量问题引起的故障和损坏
5、擅自改造硬件、错误上电、错误操作造成的故障和损坏
6、由不可抗拒自然因素引起的故障和损坏

产品返修：

用户在使用过程中由于产品故障、损坏或其他异常现象，在寄回维修之前，请先致电米尔科技客服部，与工程师进行沟通以确认问题，避免故障判断错误造成不必要的运费损失及周期的耽误。

维修周期：

收到返修产品后，我们将即日安排工程师进行检测，我们将在最短的时间内维修或更换并寄回。一般的故障维修周期为 3 个工作日（自我司收到物品之日起，不计运输过程时间），由于特殊故障导致无法短期内维修的产品，我们会与用户另行沟通并确认维修周期。

维修费用：

在免费保修期内的产品，由于产品质量问题引起的故障，不收任何维修费用；不属于免费保修范围内的故障或损坏，在检测确认问题后，我们将与客户沟通并确认维修费用，我们仅收取元器件材料费，不收取维修服务费；超过保修期限的产品，根据实际损坏的程度来确定收取的元器件材料费和维修服务费。

运输费用：

产品正常保修时，用户寄回的运费由用户承担，维修后寄回给用户的费用由我司承担。非正常保修产品来回运费均由用户承担。