

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php



Linux 开发指南 V3.1

-正点原子 领航者 NAVIGATOR 开发板教程



原子哥在线教学: www.yuanzige.com





论坛:www.openedv.com/forum.php

正点原子公司名称: 广州市星翼电子科技有限公司

公司电话: 020-38271790

公司网址: <u>www.alientek.com</u>

在线教学平台: www.yuanzige.com

开源电子网: <u>www.openedv.com/forum.php</u>

天猫旗舰店: <u>https://zhengdianyuanzi.tmall.com</u>

正点原子 B 站: <u>https://space.bilibili.com/394620890</u>

扫码关注正点原子公众号,获取更多嵌入式学习资料 扫码下载原子哥 App,提供数千讲免费开源视频学习 扫码关注 B 站正点原子官方,所有视频均可免费在线观看 扫码关注抖音正点原子,技术与娱乐结合体验双倍快乐





原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

文档更新说明

版本	版本更新说明	负责人	校审	发布日期
V3.0	V3.0版本发布;	正点原子	正点原子	2023. 5. 1
V3. 1	增加: 1、《Petalinux 构建 Qt 和 OpenCV 交叉编 译环境》; 2、《搭建驱动开发使用的 ZYNQ 镜像》 3、《字符设备驱动开发》	正点原子	正点原子	2023. 6. 9



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

目录

前言	8
第一篇 Ubuntu 系统入门篇	9
第一章 Ubuntu 系统安装	10
1.1 安装虚拟机软件 VMware	11
1.2 创建虚拟机	
1.3 安装 Ubuntu 操作系统	
1.3.1 获取 Ubuntu 系统	
1.3.2 安装 Ubuntu 操作系统	
1.4 安装 Vmware Tools	43
第二章 Ubuntu 系统入门	50
2.1 Ubuntu 系统初体验	
2.1.1 hello Ubuntu.	
2.1.2 系统设置	
2.1.3 系统重启与关机	
2.1.4 中文输入测试	
2.2 Ubuntu 终端操作	
2.3 Shell 操作	60
2.3.1 Shell 简介	60
2.3.2 Shell 基本操作	61
2.3.3 常用 Shell 命令	62
2.4 APT 下载工具	70
2.5 Ubuntu 下文本编辑	75
2.5.1 Gedit 编辑器	75
2.5.2 VI/VIM 编辑器	75
2.6 Linux 文件系统	80
2.6.1 Linux 文件系统简介以及类型	80
2.6.2 Linux 文件系统结构	
2.6.3 文件操作命令	
2.6.4 文件压缩和解压缩	90
2.6.5 文件查询和搜索	97
2.6.6 文件类型	
2.7 Linux 用户权限管理	
2.7.1 Ubuntu 用户系统	
2.7.2 权限管理	100
2.7.3 权限管理命令	
2.8 Linux 磁盘管理	104
2.8.1 Linux 磁盘管理基本概念	104
2.8.2 磁盘管理命令	106
第三章 Linux C 编程入门	110

领航者 ZYNQ 之嵌入式 Linux 开发指南 原子哥在线教学: www ward



F子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php	
3.1 Hello World!	111
3.1.1 编写代码	111
3.1.2 编译代码	112
3.2 GCC 编译器	114
3.2.1 gcc 命令	114
3.2.2 编译错误警告	114
3.2.3 编译流程	115
3.3 Makefile 基础	116
3.3.1 何为 Makefile	116
3.3.2 Makefile 的引入	116
3.4 Makefile 语法	120
3.4.1 Makefile 规则格式	
3.4.2 Makefile 受量	
3.4.3 Makefile 模式规则	
3.4.4 Makefile 目动化变量	
3.4.5 Makefile 伤目标	
3.4.6 Makefile 条件判断	
3.4.7 Makefile 函数使用	
第二篇 Petalinux 使用篇	130
第四章开发环境搭建	131
4.1 Ubuntu 和 Windows 文件互传	
4.2 Ubuntu 和 Windows 文件本地共享	
4.3 Ubuntu 系统搭建 tftp 服务器	142
4.4 Ubuntu 下 NFS 和 SSH 服务开启	143
4.4.1 NFS 服务开启	143
4.4.2 SSH 服务开启	144
4.5 Visual Studio Code 软件的安装和使用	144
4.5.1 Visual Studio Code 的安装	144
4.5.2 Visual Studio Code 插件的安装	149
4.5.3 Visual Studio Code 新建工程	151
4.6 MobaXterm 软件安装和使用	157
4.6.1 MobaXterm 安装	157
4.6.2 MobaXterm 使用	160
第五章 Petalinux 的安装	165
5.1 Petalinux 简介	166
5.2 下载 Petalinux 安装包	166
5.3 安装 Petalinux	167
5.3.1 安装依赖库以及软件	167
5.3.2 修改 bash	169
5.3.3 安装 Petalinux	170
5.4 设置 Petalinux 环境变量	172
5.5 安装 Vitis 软件	

领航者 ZYNQ 之嵌入式 Linux 开发指南 原子哥在线教学: www.yuanzige.com 论坛:ww



(子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php	
5.5.1 下载 Vitis 在线安装包	173
5.5.2 在线安装 Vitis 软件	174
5.6 Linux 系统安装 JTAG cable 驱动	
第六章 Petalinux 设计流程实战	
6.1 Petalinux 工具的设计流程概述	
6.2 使用 Petalinux 定制 Linux 系统	
6.2.1 创建 Vivado 硬件平台	
6.2.2 设置 Petalinux 环境变量	
6.2.3 创建 petalinux 工程	
6.2.4 配置 petalinux 上程	
6.2.5 配直 Linux 内核	
6.2.6	
0.2./ 削直仅备树又针 6.2.9 疟语 Datalinux 工程	
0.2.8 编译 Petalliux 工程	
6.2.10 制作 SD 启动卡	202
6.2.10 两下 5D 冶动下	208
6.2.12 打开串口上位机, 进入 Linux 系统	
第七章 Linux 基础外设的使用	
	210
7.1 GPIO 之 LED 的使用 7.2 GPIO 之 按键的使用	
7.2 GHO 之设健时仅//1	220
74RTC的使用	221
7.5 OSPI 的使用	
7.5.1 读写测试	
7.5.2 复制文件	224
7.6 USB 的使用	224
7.7 以太网的使用	
7.7.1 查看网络设备	
7.7.2 外网连接测试(有路由器)	228
7.7.3 电脑直连测试(无路由器)	
7.8 eMMC 的使用	233
第八章 Linux 显示设备的使用	238
8.1 准备工作	239
8.2 创建 Petalinux 工程	240
8.3 配置 Petalinux 工程	
8.4 配置设备树	242
8.5 配置根文件系统	245
8.6 编译 Petalinux 工程	246
8.7 制作 BOOT.bin 启动文件并复制到 SD 卡	
8.8 开发板上启动 linux	247

领航者 ZYNQ 之嵌入式 Linux 开发指南 🛛 📀 正点原	子
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php	
8.8.1 启动 linux 并在 lcd 屏上显示打印信息	247
8.8.2 测试显示设备	248
第九章 使用 Vitis 开发 Linux 应用	250
9.1 创建基于 Vitis 的 Linux 平台工程	251
9.2 创建 Linux 应用工程	254
9.3 使用 TCF Agent 方式运行	259
9.4 使用 NFS 共享方式运行	
9.5 使用 SSH 方式运行	

原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php

正点原子

前言

如今的社会早已步入人工智能和物联网的时代。万物互联和人工智能如同洪流滚滚而来, 遍入生活的方方面面,各种各样的需求被创造出来,而这些需求又需要各种各样的软硬件来 现实,那么什么样的软硬件能满足这一需求呢?

首先从硬件方面而言,不同的需求需要不同的硬件平台,那么有没有一种硬件平台能满 足不同的需求或最大程度的适应不同需求呢?

从市场占有率高的 ARM 系来说,无论是主打工业控制的 M 系列和 R 系列还是主打市场 应用的 A 系列都是串行控制领域的首选,而在并行处理领域如大数据处理、图像处理、接口 设计等领域乏善可陈,也不支持硬件可编程,由此很难适应各种需求的需要。而作为可编程 器件的 FPGA 不仅具有硬件可编程的灵活性,能适应不同场景不同需求的需要,还具有天然 的并行优势,实现实时处理,当然了,控制领域就非 FPGA 所擅长了。既然如此,能否将两 者的优势结合起来呢?

为此 Xilinx 推出了 ZYNQ,这是一款将 ARM 与 FPGA 相结合的芯片,以领航者开发板所 使用的 ZYNQ7000 系列器件为例,内部不仅具有丰富的 FPGA 逻辑资源,还集成了双核 Cortex-A9 处理器。ARM 与 FPGA 的协同,不仅能实现优势互补,而且具有极大的灵活性。

现在从软件方面来看,这里的软件主要指操作系统体系。小型的嵌入式操作系统有 UCOS、FreeRTOS等等,这些操作系统都是一个个kernel,如果需要网络、文件系统、GUI等 这些就需要开发者自行移植。而移植又是非常麻烦的一件事情,而且移植完成以后的稳定性 也无法保证。即使移植成功,后续的开发工作也比较繁琐,因为不同的组件其 API 操作函数 都不同,没有一个统计的标准,使用起来的话学习成本比较高。这个时候一个功能完善的操 作系统显得尤为重要:具有统一的标准;提供完善的多任务管理、存储管理、设备管理、文 件管理和网络等。Linux 就是这样一个系统,当然这样的系统还有很多,比如 Windows, MacOS, UNIX 等等。本书我们讲解 Linux,而 Linux 开发可以分为底层驱动开发和应用开发, 本书以领航者开发板为硬件平台,从实现到驱动,从底层到应用的讲解 linux 开发。

从最初的 PL 部分的纯硬件的 FPGA 开发到后面的涉及到软件与硬件结合的 PS 部分的嵌入式裸机开发,到现在的 linux 系统开发,我们向前迈出了重要的一步。需要提醒的是,不要跳过前面两步,这既是由于涉及到不同的开发工具的交叉使用,也是因为前面两步能加深对 linux 开发的认识。



原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php

第一篇 Ubuntu 系统入门篇

本篇主要讲解 Ubuntu 系统,包括如何在虚拟机上安装 Ubuntu 操作系统,安装好以后 Ubuntu 的设置、基本操作等等。在正式进行嵌入式开发之前肯定是要先学会 Ubuntu 系统如何 使用的,由于 Ubuntu 系统功能很庞大,如果要详细讲解的话一本书都讲不完,因此本篇只做 Ubuntu 系统入门讲解,掌握我们进行嵌入式开发所需的计能即可,如果想详细的学习 Ubuntu 系统的话可以参考其他的书籍,比如经典的《鸟哥的linux私房菜》,《鸟哥的linux私房菜》 这本书使用的 CentOS 操作系统,但是 Ubuntu 下完全可以使用。

当 Ubuntu 系统入门以后,我们重点要学的就是如何在 Linux 下进行 C 语言开发,如何使 用 gcc 编译器、如何编写 Makefile 文件等等。

如果此前已经使用过 Ubuntu 操作系统,并且从事过 Linux C 编程工作的话本篇就不需要 看了,可以直接跳到第二篇,开始 Petalinux 使用篇的学习。

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

正点原子

第一章 Ubuntu 系统安装

Linux 的开发需要在 Linux 系统下进行,这就要求我们的 PC 主机安装 Linux 系统,本书 我们选择 Ubuntu 这个 Linux 发行版系统。本章讲解如何安装虚拟机,以及如何在虚拟机中安 装 Ubuntu 系统,安装完成以后如何做简单的设置。如果对于虚拟机以及 Ubuntu 基础操作已 经熟悉的话就可以跳过本章。

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

正点原子

1.1 安装虚拟机软件 VMware

不是安装 Ubuntu 吗? 怎么要先安装虚拟机呢? 虚拟机是个啥? 相信大部分第一次安装 Ubuntu 的人都会有这个疑问。我不能直接安装 Ubuntu 吗? 能不能不要虚拟机呢? 答案是肯定 可以的! 直接在电脑上安装 Ubuntu 以后你的电脑就是一个真真正正的 Ubuntu 电脑了,你可 以再安装一个 Windows 系统,这样你的电脑就是双系统了,在开机的时候可以选择不同的系 统启动。但是这样的话会有一个问题,那就是你每次只能选择其中的一个系统启动,要么 Windows 要么 Ubuntu, 但是我们在开发的时候经常需要在 Windows 和 Ubuntu 下来回切换, Windows 系统下的软件资源要比 Ubuntu 下丰富的多,比如我们在 Windows 用 Source Insight 这个神器编写代码,然后拿到 Ubuntu 下编译。这个就涉及到两个系统切换问题,显然如果你 直接在电脑上安装 Ubuntu 以后就没法做到,因为你每次开机只能在 Windows 和 Ubuntu 下二 选一。

如果 Ubuntu 系统能作为 Windows 下的一个软件就好了,我们默认启动 Windows 系统, 需要用到 Ubuntu 的话直接打开这个软件就行了。当然可以!这个就要借助虚拟机了,虚拟机 顾名思义就是虚拟出一个机器,然后你就可以在这个机器上安装任何你想要的系统,相当于 在克隆出一个你的电脑,这样在主机上运行 Windows 系统,当我们需要用到 Ubuntu 的话就打 开安装有 Ubuntu 系统的虚拟机。

虚拟机的实现我们可以借助其他软件,比如 Vmware Workstation, Vmware Workstation 是 收费软件,免费的虚拟机软件有 Virtualbox。本书我们使用 Vmware Workstation 软件来做虚拟 机。笔者使用的是 Vmware Workstation Pro 15,如果读者使用 Windows11 系统,则可以下载 Vmware Workstation Pro 16 或者更高版本。Vmware Workstation 软件可以在 Vmware 官网下载, 下载地址: https://www.vmware.com/cn/products/workstation-pro/workstation-pro-evaluation.html, 当前最新的版本是 Vmware Workstation Pro 17,我们下载 Windows 版本的,如下图所示:



图 1.1.1 Vmware 官网

我们已经在开发板光盘里面提供了 Vmware Workstation 15 软件,大家可以直接使用,在 光盘目录:资料盘(B盘) \VMware\VMware-workstation-full-15.5.0-14665864.exe。Vmware Workstation 的安装和普通软件安装一样,双击 VMware-workstation-full-15.5.0-14665864.exe 进 入安装界面,如下图所示:



图 1.1.2 VMware 安装界面



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php 点击图 1.1.2 中的"下一步",进入图 1.1.3 所示步骤:

☆ VMware Workstation Pro 安装 - □ ×
最终用户许可协议 ————————————————————————————————————
请仔细阅读以下许可协议。
VMWARE 最终用户许可协议 ^
请注意,在本软件的安装过程中无论可能会出现任何条款, 使用本软件都将受此最终用户许可协议各条款的约束。
重要信息,请仔细阅读:您一旦下载、安装或使用本软件,您(自然人或法人)即同意接受本最终用户许可协议("本协议")的约束。如果您不同意本协议的条款,请勿下载、安装或使用本软件,您必须删除本软件,或在三十(30)天、
□我接受许可协议中的条款(A) < 选中接受条款
打印(P) 上一步(B) 下一步(N) 取消

图 1.1.3 VMware 条款

先选择图 1.1.3 中的"我接受许可协议中的条款",然后在选择"下一步",进入图 1.1.4 所示步骤:

∰ VMware Workstation Pro 安装	—		×
自定义安装			5
选择安装目标及任何其他功能。			-
安装位置: C:\Program Files (x86)\\/Mware\\/Mware Workstation\		百改.	
		SEVA:	
□ 增强型键盘驱动程序(需要重新引导以使用此功能(E) 此功能要求主机驱动器上具有 10MB 空间。	更改安	装路	径
上一步(8) 下一	步(N)	取	肖

图 1.1.4 选择安装路径

在上图中选择软件的安装路径,点击"更改"按钮,然后根据自己的实际需要选择合适路径即可,我的安装路径如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

🛃 VMware Workstation Pro 安装	-	- [×
更改目标文件夹			
浏览到目标文件夹			Y
查找 📑 VMware Workstation	```	Ē	Ě
文件支名称作)・ D: \Program Files (x86) \VMware \VMware Workstation \			
	确定		取消

图 1.1.5 安装路径

选择好路径以后点击图1.1.5中的"确定"按钮,然后回到图1.1.4所示界面,点击图1.1.4 中的"下一步",进入图1.1.6所示界面:

Why are Workstation Pro 安装 − □	×
用户体验设置 编辑野认设罢以提高你的田白体验。	
建议不勾选这两个	-
□ 启动时检查产品更新(C) 在 VMware Workstation Pro 启动时,检查应用程序和已安装软件组件是否有 新版本。	
□加入 VMware 客户体验提升计划(3)	
VMware 客户体验提升计划 (CEIP) 将向 VMware 提供相 关信息,以帮助 VMware 改进产品和服务、解决问 题、并向您建议如何以最佳方式部署和使用我们的产	
品。作为 CEIP 的一部分,VMware 会定期收集和您所 持有的 VMware 密钥相关的使用 VMware 产品和服务的 >	
<u>了解更多信息</u>	
上一步(B) 下一步(N) 取消	

图 1.1.6 检查更新界面

在图 1.1.6 中,会有两个复选框,默认都是选中的,建议不要选中! 然后点击该图中的 "下一步"按钮,进入下图所示界面:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

🛃 VMware Workstation Pro 安装	_		×
快捷方式 选择您要拉入系统的快捷方式。			
			_
在以下位置创建 VMware Workstation Pro 的快捷方式:			
☑ 桌面(0)			
☑开始菜单程序文件夹(S)			
选中这两个			
上一步(8) 下一步	;(N)	取	肖

图 1.1.7 快捷方式设置

在图 1.1.7 中有两个选项,我们都选中,这样在安装完成以后就会在开始菜单和桌面上有 VMware 的图标,选中以后点击该图中的"下一步",进入图 1.1.8 界面:

🛃 VMware Workstation Pro 安装			×
已准备好安装 VMware Workstation Pro			Ð
单击"安装"开始安装。单击"上一步"查看或更改任何安装设置。 导。	单击"取	消"退出向	j
上一步(8) 安装(I)	取	肖

图 1.1.8 安装确定界面

前面几步已经设置好安装参数了,如果许需要修改安装参数的话就点击图 1.1.8 中的"安装"按钮开始安装 VMware,安装过程如下图所示:



正点原子

图 1.1.9 安装进行中

图 1.1.9 就是安装过程, 耐心等待几分钟, 等待安装完成, 安装完成以后会有如图 1.1.10 所示提示:



图 1.1.10 安装完成

点击图 1.1.10 中的"完成"按钮,完成 VMware 的安装,安装完成以后就会在桌面上出现 VMware Workstation Pro 的图标,如下图所示:

原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php

正点原子



图 1.1.11VMware 桌面图标

双击图 1.1.11 中的图标打开 VMware 软件,在第一次打开软件的时候会提示你输入许可 证密钥,如下图所示:

欢迎使用 VMware Workstation 15	×			
D VMware Workstation 15				
● 我有 VMware Workstation 15 的许可证密钥(H):				
是否需要许可证密钥?				
<u>立即购买</u> ————————————————————————————————————				
○ 我希望试用 VMware Workstation 15 30 天(W)				
♥继续(C) 取消				

图 1.1.12 输入许可证密钥

前面说了 VMware 是付费软件,是需要购买的,如果你购买了 VMware 的话就会有一串 许可密钥,如果没有购买的话就选择"我希望试用 VMware Workstation 15 30 天"选项,这样 你就可以体验 30 天 VMware。输入密钥以后点击"继续按钮",如果你的密钥正确的话就会 提示你购买成功,如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php



图 1.1.13 购买 VMware 成功

点击图 1.1.13 的"完成"按钮, VMware 软件正式打开, 界面如下图所示:



图 1.1.14VMwareWorkstation 主界面

至此, 虚拟机软件 VMware 安装成功。

1.2 创建虚拟机

安装好 VMware 以后我们就可以在 VMware 上创建一个虚拟机,打开 VMware,选择:文件->新建虚拟机,如图 1.2.1 所示:

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

正点原子



图 1.2.1 新建虚拟机

打开图 1.2.2 所示创建虚拟机向导界面:

新建虚拟机向导	×
	欢迎使用新建虚拟机向导
VMWARE 15 WORKSTATION	您希望使用什么类型的配置?
PRO™	 ○ 典型(推荐)(T) 通过几个简单的步骤创建 Workstation 15.x 虚拟机。
	自定义(高级)(C) 创建带有 SCSI 控制器类型、虚拟磁盘类 型以及与旧版 VMware 产品兼容性等高 级选项的虚拟机。
帮助	< 上一步(B) 下一步(N) > 取消

图 1.2.2 创建虚拟机向导

选中图 1.2.2 中的"自定义"选项,然后选择"下一步",进入图 1.2.3 所示硬件兼容性 选择界面:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

新建虚拟机向导			×
选择虚拟机硬件兼容性 该虚拟机需要何种	t 硬件功能?		
虚拟机硬件兼容性			
硬件兼容性(H):	Workstation	15.x	\sim
兼容:	ESX Serve	er(S)	
兼容产品:		限制:	
Fusion 11.x Workstation 15.x	~	64 GB 内存 16 个处理器 10 个网络适配器 8 TB 磁盘大小 3 GB 共享图形内存	~
帮助	<上一	步(B) 下一步(N) >	取消

图 1.2.3 硬件兼容性选择

在图 1.2.3 中我们使用默认值就行了,直接点击"下一步",进入下图所示的操作系统安装界面:

新建虚拟机向导	\times
安装客户机操作系统 虚拟机如同物理机,需要操作系统。您将如何安装客户机操作系统?	
安装来源:	
○ 安装程序光盘(D):	
无可用驱动器	
○ 安装柱序元盆映像文件(iso)(M):	
G:\软件\ubuntu操作系统\ubuntu-16.04-desktop-amd6 > 浏览(R)	
● 稍后安装操作系统(S)。	
创建的虚拟机将包含一个空白硬盘。	

图 1.2.4 安装客户机操作系统



原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php

在上图中选择你新创建的虚拟机要安装什么系统? windows 还是 linux, 如果你现在就安 装系统的话,需要准备好系统文件,一般是.iso文件。我们现在不安装系统,因此选择"稍后 安装操作系统(S)"这个选项,然后选择"下一步",进入下图所示界面:

新建虚拟机向导	\times
选择客户机操作系统 此虚拟机中将安装哪种操作系统?	
客户机操作系统 Microsoft Windows(W) ● Linux(L) VMware ESX(X) 其他(O)	
版本(V) Ubuntu 64 位	~
帮助 < 上一步(B) 下一步(N) > 取	₹消

图 1.2.5 客户机操作系统选择

上图中依旧是让你选择你要在虚拟机中装什么系统,图 1.2.5 是和图 1.2.4 配合在一起使 用的,在图 1.2.4 中放入系统文件(.iso 文件),然后在图 1.2.5 中选择你图 1.2.4 中放入的是什么 系统,然后 VMware 就会稍后自动安装所设置的系统。在图 1.2.4 中我们没有设置系统文件, 因此图 1.2.5 是没用的,不过我们还是在图 1.2.5 中的客户机操作系统一栏选择"Linux",版 本选择 Ubuntu 64 位, 然后点击"下一步", 进入图 1.2.6 所示界面:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

新建虚拟机向导	×
命名虚拟机 您希望该虚拟机使用什么名称?	
虚拟机名称(V): Ubuntu 64 位	
位置(L): C:\Users\zuozh\Documents\Virtual Machines\Ubuntu 64 位 浏览(R) 在"编辑">"首选项"中可更改默认位置。	
选择虚拟机使用的磁盘!一定要是一个空磁盘!!	
< 上一步(B) 下一步(N) > 取消	

图 1.2.6 命名虚拟机

在图 1.2.6 中上面是命名虚拟机名字,大家可以根据自己的使用习惯给虚拟机命名,重点 是下面的虚拟机位置选择!我们要给虚拟机单独清理出一块磁盘,做嵌入式开发建议这块空 磁盘的大小不小于 100GB,比如我清理除了一个 196GB 的 I 盘给虚拟机使用,如图 1.2.7 所示:

Ubuntu (I:)
 196 GB 可用 , 共 196 GB

图 1.2.7 虚拟机所使用的磁盘

清理出虚拟机专用的磁盘以后然后就在图 1.2.6 中的位置出选择这个磁盘,比如我的位置 选择如图 1.2.8 所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

新建虚拟机向导	×
命名虚拟机 您希望该虚拟机使用什么名称?	
虚拟机名称(V):	
Ubuntu 64 位	
位置(L):	
I:\	浏览(R)
在"编辑">"首选项"中可更改默认位置。	
_<上一步(B) 下一步(N) >	取消

图 1.2.8 选择虚拟机磁盘位置

设置好图 1.2.8 中的虚拟机磁盘位置以后点击"下一步",进入图 1.2.9 所示的处理器配置选择界面:

新建虚拟机向导		×
处理器配置 为此虚拟机指定处理器	牧 量。	
处理器		
处理器数量(P):	2 ~	
每个处理器的内核数量(C):	2 ~	
处理器内核总数:	4	
•		
帮助	< 上一步(B) 下一步(N) >	取消

图 1.2.9 处理器配置界面



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

图 1.2.9 中就是配置你的虚拟机所使用的处理器数量,以及每个处理器的内核数量,这个 要根据自己实际使用的电脑 CPU 配置来设置。比如我的电脑 CPU 是 I7-4720HQ,这是个 4 核 8 线程的 CPU,因此我就可以分 2 个核给 VMware,然后 I7-4720HQ 每个物理核有两个逻辑 核,因此每个处理器的内核数量就是 2,所以的 VMware 虚拟机配置就如图 1.2.9 所示,大家 根据自己的实际电脑 CPU 配置来设置即可,设置好以后点击"下一步",进入图 1.2.10 所示 内存配置界面:

新建虚拟机向导	≩	×	
此虚拟机的内存 您要为此虚拟机使用多少内存?			
指定分配给此	虚拟机的内存量。内存大小必须为 4 MB 的倍数。		
64 GB - 32 GB - 16 GB - 8 GB - 2 GB - 1 GB - 2 GB - 1 GB - 512 MB - 256 MB - 128 MB - 128 MB - 32 MB - 16 MB - 8 MB - 4 MB -	此虚拟机的内存(M): 8192 → MB 调整滑块,选择内存大小 ■最大推荐内存: 13.4 GB ■ 推荐内存: 2 GB ■ 客户机操作系统最低推荐内存: 1 GB		
帮助	< 上一步(B) 下一步(N) > 取消		

图 1.2.10 内存配置

同样的在图 1.2.10 中根据自己电脑的实际内存配置来设置分给虚拟机的内存大小,比如 我的电脑是 16GB 的内存,因此我可以给虚拟机分配 8GB 的内存。配置好虚拟机的内存大小 以后点击"下一步",进入图 1.2.11 所示的网络类型选择界面:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

新建虚拟机向导 ×
网络类型 要添加哪类网络?
网络连接
使用挤接网络(R) 为客户机操作系统提供直接访问外部以太网网络的权限。客户机在外部网络 上必须有自己的 IP 地址。
○ 使用网络地址转换(NAT)(E) 为客户机操作系统提供使用主机 IP 地址访问主机拨号连接或外部以太网网络 连接的权限。
○ 使用仅主机模式网络(H) 将客户机操作系统连接到主机上的专用虚拟网络。
○ 不使用网络连接(T)
帮助 < 上一步(B) 下一步(N) > 取消

图 1.2.11 网络类型选择界面

在图 1.2.11 中我们选择"使用桥接网络",然后点击"下一步",进入图 1.2.12 所示的选择 I/O 控制器类型界面:

新建虚拟机向导		×
选择 I/O 控制器 您要使用何种	类型 ·类型的 SCSI 控制器?	
- I/O 控制器类型		
SCSI 控制器:		
O BusLogic(U)	(不适用于 64 位客户机)	
LSI Logic(L)	(推荐)	
O LSI Logic SAS	(S)	
帮助	< 上一步(B) 下一步(N) >	取消

图 1.2.12 I/O 控制器选择



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php I/O 控制器类型选择默认值就行,也就是"LSI Logic",然后点击"下一步",进入磁盘 类型选择界面,如图 1.2.13 所示:

新建虚拟机向导			×
选择磁盘类型 您要创建何和	中磁盘?		
虚拟磁盘类型			
◯ IDE(I)			
● SCSI(S) (持	谁荐)		
⊖ SATA(A)			
○ NVMe(V)			
帮助	< 上一步(B)	下一步(N) >	取消

图 1.2.13 选择磁盘类型

图 1.2.13 中选择磁盘类型,使用默认值 "SCSI"即可,然后点击"下一步",进入选择磁盘界面,如图 1.2.14 所示:

新建虚拟机向导	\times
选择磁盘 您要使用哪个磁盘?	
磁盘	
创建新虚拟磁盘(V) 虚拟磁盘由主机文件系统上的一个或多个文件组成,客户机操作系统会将 其视为单个硬盘。虚拟磁盘可在一台主机上或多台主机之间轻松复制或移 动。	
 ○ 使用现有虚拟磁盘(E) 选择此选项可重新使用以前配置的磁盘。 	
 ○ 使用物理磁盘 (适用于高级用户)(P) 选择此选项可为虚拟机提供直接访问本地硬盘的权限。需要具有管理员特权。 	
帮助 < 上一步(B) 下一步(N) > 取消	

图 1.2.14 磁盘选择



原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php

图 1.2.14 中使用默认值,即"创建新虚拟磁盘",这样我们前面设置好的那个空的磁盘 就会被创建为一个新的磁盘,设置要以后点击"下一步",进入磁盘容量设置界面,如图 1.2.15 所示:

新建虚拟机向导 ×				
指定磁盘容量 磁盘大小为多少?				
最大磁盘大小 (GB)(S): 196.0 →				
针对 Ubuntu 64 位 的建议大小: 20 GB				
立即分配所有磁盘空间(A)。 分配所有容量可以提高性能,但要求所有物理磁盘空间立即可用。如果不立即 分配所有空间,虚拟磁盘的空间最初很小,会随着您向其中添加数据而不断变 大。				
○ 将虚拟磁盘存储为单个文件(O)				
● 将虚拟磁盘拆分成多个文件(M) 七八道盘氏 司以更轻松地在计算机之间移动虚拟机 伊可维合喀斯士家黑道				
拆分磁盈后,可以更轻松地在计算机之间移动虚拟机,但可能会降低大谷重磁 盘的性能。				
帮助 < 上一步(B) 下一步(N) > 取消				

图 1.2.15 磁盘容量设置

图 1.2.15 是用来设置我们清出的空的磁盘多少是给虚拟机用的,我们清出了一个空磁盘 肯定是全部给虚拟机用的,因此设置最大磁盘大小为空磁盘的大小,比如图 1.2.7 中我的那个 I 盘是 196GB 的,因此图 1.2.15 中就设置最大磁盘大小为 196GB,然后点击"下一步",进入图 1.2.16 所示界面指定磁盘文件,



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

新建虚拟机向导	<
指定磁盘文件 您要在何处存储磁盘文件?	
磁盘文件(F)	
将使用多个磁盘文件创建一个 196 GB 虚拟磁盘。将根据此文件名自动命名这些磁盘文件。	
Ubuntu 64 位.vmdk 浏览(R)	
帮助 < 上一步(B) 下一步(N) > 取消	

图 1.2.16 指定磁盘文件

图 1.2.16 使用默认设置,不要做任何修改,直接点击"下一步",进入已准备好创建虚 拟机界面,如图 1.2.17 所示:

	刘建虚拟机:
名称:	Ubuntu 64 位
位置:	I:\
版本:	Workstation 15.x
操作系统:	Ubuntu 64 位
硬盘:	196 GB, 拆分
内存:	8192 MB
网络适配器:	桥接模式 (自动)
其他设备 <mark>:</mark>	4 个 CPU 内核, CD/DVD, USB 控制器, 打印机, 声卡

图 1.2.17 准备创建虚拟机

在图 1.2.17 中确认自己的虚拟机配置,如果确认无误就点击"完成",如果有误的话就 返回有误的配置界面做修改,点击"完成"按钮以后就会创建一个虚拟机,如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

🔁 Ubuntu 64位 - VMware Workstation 🛛 📃							
文件(F) 编辑(E) 查看(V) 虚拟机(M)	选项卡(T) 帮助(H) 🕨 🔻	₽ Q 🎴	♀ □ □ □ ▷ ≥ ₽ ·				
文件(F) 骗辑(E) 重看(V) 虚拟机(M) <u>库</u> × <u>○</u> 在此处键入内容进行搜索 ▼ □ <u>我的计算机</u> □ <u>Ubuntu 64 位</u> <u>□</u> 共享的虚拟机 刚刚创建的虚拟机	远坝卡(1) 幣助(H) 前面 主页 × □ Ubuntu 64 位 ① Ubuntu 64 位 〕 Ubuntu 64 位 〕 开启此虚拟机 □ 編輯虚拟机设置 、 设备 内存 □ 妙理器 □ 硬盘 (SCSI) ③ CD/DVD (SATA) □ 网络适配器 ④ 声卡	♀ 45 ♀ 8 GB 4 196 GB 自动检测 桥接模式 (自 存在 自动检测	▲ 山 山 口, D, D, L L				
	□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □	存在 自动检测	主 IP 地址:网络信息不可用 →				

图 1.2.18 新创建的虚拟机

创建虚拟机成功以后就会在右侧的:我的计算机下出现刚刚创建的虚拟机"Ubuntu 64 位",点击一下就会在右侧打开这个虚拟机的详细信息,如下图所示:

1			— —	×			
选项卡(T) 帮助(H) 🕨 🔻	₽ P 🚇	<u>_</u>					
☆主页 × □ Ubuntu 64 位 ×							
[Ubuntu 64 位							
▶ 开启此虚拟机		^					
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□							
┃ ▼ 设备							
IIII 四 内存	8 GB						
1 处理器	4						
☐ 硬盘 (SCSI)	196 GB						
💿 CD/DVD (SATA)	自动检测						
网络适配器	桥接模式 (自		▼ 虚拟机详细信息				
── USB 控制器	存在		状态: 已关机				
↓ 声卡	自动检测		配置文件: I:\Ubuntu 64 位.vmx				
🛛 🔓 打印机	存在		····································				
	自动检测						
]		~					

图 1.2.19 新建虚拟机配置信息



原子哥在线教学: www.yuanzige.com

在图 1.2.19 中的设备一栏我们可以看到虚拟机详细的配置信息,图 1.2.20 所示的两个按 钮就是虚拟机的开关。

论坛:www.openedv.com/forum.php



图 1.2.20 虚拟机开关

上图中的这两个绿色三角按钮都可以打开虚拟机,但是此时虚拟机没有安装任何操作系统,因此没法打开,接下来我们就是要在刚刚新建的这个虚拟机中安装 Ubuntu 操作系统。

1.3 安装 Ubuntu 操作系统

1.3.1 获取 Ubuntu 系统

前面虚拟机已经创建成功了,相当于硬件已经准备好了,接下来就是要在虚拟机中安装 Ubuntu系统了,首先肯定是获取到 Ubuntu 的系统镜像。Ubuntu系统镜像肯定是在 Ubuntu 官 网获取,下载地址为: <u>https://www.ubuntu.com/download/desktop</u>,如图 1.3.1 所示:



图 1.3.1 Ubuntu 最新版系统下载界面



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

从上图中可以看出,目前最新 LTS(长期支持)版本的 Ubuntu 系统是 22.04.2,我们能够 使用该版本么?答案是要看我们后面安装的重要的开发工具支不支持该版本。

我们后面安装的重要的开发工具是 2020.2 版本的 Petalinux,要与使用的 Vivado 版本相同 (PetaLinux 是一款嵌入式 Linux 系统开发套件,主要针对基于 Xilinx FPGA 的片上系统设计)。 我们打开 2020.2 版本的 Petalinux 手册,查看第 2 章的安装需求如下图所示:



Chapter 2

Setting Up Your Environment

Installation Steps

Installation Requirements

The PetaLinux tools installation requirements are:

- Minimum workstation requirements:
 - . 8 GB RAM (recommended minimum for Xilinx® tools)
 - . 2 GHz CPU clock or equivalent (minimum of eight cores)
 - 100 GB free HDD space
 - Supported OS:
 - Red Hat Enterprise Workstation/Server 7.4, 7.5, 7.6, 7.7, 7.8 (64-bit)
 - CentOS Workstation/Server 7.4, 7.5, 7.6, 7.7, 7.8 (64-bit)

 Ubuntu Linux Workstation/Server 16.04.5, 16.04.6, 18.04.1, 18.04.2, 18.04.3, 18.04.4 (64-bit)

 You need to have root access to install the required packages mentioned in the release notes. The PetaLinux tools need to be installed as a non-root user.

图 1.3.2 支持的 Ubuntu 系统

在支持的操作系统 Supported OS 中,显示支持的 Ubuntu 系统不包括当前最新的 22.04.2 版本的 Ubuntu,所以不能使用该 Ubuntu。那如果非要使用不支持的 Ubuntu 版本会怎样呢? 答案是会出现各种各样的问题,缺少各种各样的包和库,不要问笔者是怎么知道的,因为笔 者吃过这方面的亏,最后只好选择使用 Xilinx 官方支持的 Ubuntu 版本。

读者可以使用上面支持的 6 个 Ubuntu 版本中选一个,笔者选择的是相对较新的 Ubuntu Linux 18.04.2 版本,后面所有的例程和教程均在 18.04.2 (64-bit)版本下完成。18.04.2 (64-bit)版本的 Ubuntu 下载地址为: <u>http://old-releases.ubuntu.com/releases/18.04.2/</u>,下载"ubuntu-18.04.2-desktop-amd64.iso"这个版本,该版本也已经提供在开发板光盘中,路径为:开发板资料盘(B盘)\Ubuntu\ ubuntu-18.04.2-desktop-amd64.iso。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

1.3.2 安装 Ubuntu 操作系统

Ubuntu 系统获取到以后就可以安装了,打开 VMware 软件,选择:虚拟机->设置,如下 图所示:

🔁 Ubuntu 64 位 - VMware	Wo	rkstation										—		\times
文件(F) 编辑(E) 查看(V)	虚	拟机(M)	选项卡(T)	帮助(H)	•	4	p 😐	<u>_</u>			IR	>2^2	-	
库 ● 在此处键入内容进行搜索	() ()	电源(P) 可移动设 暂停(U)	备(D)		Ctrl-	> > ⊦Shift+P				,				
 以 我的计算机 Ubuntu 64 位 中 共享的虚拟机 	4	发送 Ctrl 抓取输入	+Alt+De 内容(l)	I(E)		Ctrl+G		^						
	6	快照(N) 捕获屏幕	(C)		Ctrl+Alt	+PrtScn								
	B	管理(M) 安装 VM	ware Toc	ols(T)		>								
		设置(S)				Ctrl+D								
			💿 CI	D/DVD (SA	ATA)	自动检	测			_				
			5- M	络适配器		桥接模	式 (自		▼虚	拟机详	细信息			
			🔁 U 🕾	SB 控制器		存在			-	状态	已关机			
			⇒⊅	Ŧ		自动检	测		硬件	心直又(+) 牛兼容性	: I:\Ubun : Workst	tu 64 <u>1⊻</u> .vm ation 15.x ∉	× ē拟机	
			骨打	印机		存在			ŧ	IP 地址	网络信息	急不可用		
				示器		自动检	测	~						

图 1.3.3 打开虚拟机设置对话框

打开以后的虚拟机设置对话框如下图所示:

拟机设置 更件 选项			>
设备 ■ 内存 ① 处理器 → 硬盘 (SCSI) ③ CD/DVD (SATA) ● 网络适配器 ④ USB 控制器 ④ 声卡 ● 打印机 ■ 显示器	摘要 8 GB 4 196 GB 自动检测 桥接模式 (自动) 存在 自动检测 存在 自动检测	连接 USB 兼容性(C): USB 2.0 □ 显示所有 USB 输入设备(S) ☑ 与虚拟机共享蓝牙设备(B)	

图 1.3.4 虚拟机对话框

首先设置"USB 控制器"选项,默认 USB 控制器的 USB 兼容性为 USB2.0,这样当你使 用 USB3.0 的设备的时候 Ubuntu 可能识别不出来,因此我们需要调整 USB 兼容性为 USB3.0, 如图 1.3.5 所示:

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

虚拟机设置 硬件 选项		×
设备 ■ 内存 ① 处理器 → 硬盘 (SCSI) ③ CD/DVD (SATA) ● 网络适配器 ← USB 控制器 ♥ JSB 控制器 ♥ 声卡 合 打印机 ■ 显示器	摘要 8 GB 4 196 GB 自动检测 桥接模式 (自动) 存在 自动检测 存在 自动检测	连接 USB 兼容性(C): USB 3.0 ∨ 显示所有 USB 输入设备(S) ☑ 与虚拟机共享蓝牙设备(B) □ 要通过 USB 3.0 控制器使用 USB 设备,必须具备 Linux 内核 3.2 或更新版本。 USB兼容性选择USB 3.0

正点原子

图 1.3.5 USB 兼容性设置

设置好 USB 兼容性以后就开始安装 Ubuntu 系统了,选中虚拟机设置对话框中的 "CD/DVD(SATA)"选项,然后在右侧选中"使用 ISO 映像文件",如图 1.3.6 所示:

虚拟机设置		×
硬件 选项		
 设备 摘要 通中存 8 GB ● 处理器 4 ● 硬盘 (SCSI) 196 GI ③ CD/DVD (SATA) 自动检 ○ M络适配器 桥接機 ← USB 控制器 存在 ④ 声卡 自动检 ● 打印机 存在 □ 显示器 自动检 	B 2 <u>洲</u> 1式 (目动) 2 2 2 洲	设备状态 □ 已连接(C) ☑ 启动时连接(O) 连接 ● 使用物理驱动器(P): 自动检测 ● 使用 ISO 映像文件(M):

图 1.3.6 系统镜像设置

在上图中的"使用 ISO 映像文件"里面添加我们刚刚下载的 Ubuntu 系统镜像,点击"浏览"按钮,选择 Ubuntu 系统镜像,完成以后如下图所示:

虚拟机设置		×
虚拟机设置 硬件 选项 设备 四内存 碰处理器 一硬盘 (SCSI) ③ CD/DVD (SATA) ⑤ CD/DVD (SATA) ⑤ CD/DVD (SATA) ⑤ CD/DVD (SATA) ⑤ DS 控制器 ④ 声卡 ⑤ 打印机 □ 显示器	摘要 8 GB 4 196 GB 自动检测 桥接模式 (自动) 存在 自动检测 存在 自动检测 存在 自动检测	★ 设备状态 已连接(C) ✓ 启动时连接(O) 连接 使用物理驱动器(P): 自动检测 ④ 使用 ISO 映像文件(M): G:\IMX6\IMX6UL\开发板光盘\ ✓ 浏览(B)
		高级(V)…

图 1.3.7 Ubuntu 镜像选择



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

设置好以后点击"确定"按钮退出,退出以后就可以打开虚拟机了,虚拟机就会自动的 安装 Ubuntu 系统,如下图所示:



图 1.3.8 Ubuntu 安装开始

Ubuntu开始安装以后首先是语言选择,如图1.3.9所示。Ubuntu默认语言是英文,习惯中 文的我们,选择"中文(简体)",选择好以后点击右侧的"安装 Ubuntu"按钮,进入安装过 程。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php



图 1.3.9 语言选择

安装一开始会有 7 个配置步骤。首先配置是键盘布局,保持默认即可,点击"继续"按 钮,如下图所示:

Tue	15:10 ? 📢 🗸 🔫
键盘布局	
选择您的键盘布局:	
法语(刚果民主共和国,刚果(金))	汉语
法语(加拿大)	汉语 - Tibetan
菲律宾语	汉语 - Tibetan (with ASCII numerals)
芬兰语	汉语 - Uyghur
哈萨克语	
汉语	
荷兰语	
里山海	
在这里输入以测试您的键盘	
探测键盘布局	
	退出(Q) 后退(B) 继续
• • •	• • • •

图 1.3.10 键盘布局



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

接下来配置是否选择在安装 Ubuntu 时下载更新,以及是否为图形或者无线硬件安装其它 第三方软件,如下图所示,我们不勾选这两个,否则安装过程很慢。

更新和其他软件	
 您希望先安装哪些应用? 正常安装 网络浏览器、工具、办公软件、游戏和媒体播放器。 最小安装 网络浏览器和基本工具 其他选项 安装 Ubuntu 时下载更新 这能节约安装后的时间。 为图形或无线硬件,以及其它媒体格式安装第三方软件 此软件及其文档遵循许可条款。其中一些系专有的。 	不要选择,否则安装过程很慢。 另外,从这一步开始要断开电脑 的网络连接,因为有网络连接会 自动下载更新
	退出(Q) 后退(B) 继续

图 1.3.11 是否安装是下载更新

另外从这一步开始请断开电脑的网络连接,因为有网络连接会自动下载更新,一方面会 导致安装变慢,另一方面可能会让后面安装其他工具如 Petalinux 时出现问题。

直接点击图 1.3.11 中的"继续"按钮,弹出安装类型,使用默认的"清除整个磁盘并安 装 Ubuntu",如下图所示:


原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php



图 1.3.12 安装类型选择

设置好安装类型以后点击"现在安装"按钮,会弹出"将改动写入磁盘吗?"对话框, 点击"继续"即可,下一步会让你输入你在哪个位置,默认在上海,保持默认即可,如下图 所示:



图 1.3.13 输入所在位置



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

点击"继续"按钮,进入下一步设置用户名和密码,设置自己的用户名和密码,可随意 设置,但要记得密码。我的设置如图 1.3.14 所示:

Tue 23:29	.?. ♥) 🕛 👻
您是谁?	
您的姓名: zuozhongkai 您的计算机名: zuozhongkai-virtual-mi ✔ 与其他计算机联络时使用的名称。 选择一个用户名: zuozhongkai ✔ 选择一个密码: ●●●●●●● 密码强度 :合理 确认您的密码: ●●●●●●● 《 ●●●●●●● 《 ●●●●●●● 《 ●●●●●●● 《 ●●●●●● 《 ●●●●●●● 《 ●●●●●● 《 ●●●●● 《 ●●●●●● 《 ●●●●●● 《 ●●●●●● 《 ●●●●●● 《 ●●●●●● 《 ●●●●●● 《 ●●●●●● 《 ●●●●●● 《 ●●●●●●● 《 ●●●●●●● 《 ●●●●●● 《 ●●●●●●● 《 ●●●●●● 《 ●●●●●● 《 ●●●●●●● 《 ●●●●●● 《 ●●●●●● 《 ●●●●●●● 《 ●●●●●● 《 ●●●●●●● 《 ●●●●●●● 《 ●●●●●●● 《 ●●●●●●● 《 ●●●●●●● 《 ●●●●●●● 《 ●●●●●●● 《 ●●●●●●● 《 ●●●●●●● 《 ●●●●●● 《 ●●●●●● 《 ●●●●●●● 《 ●●●●●● 《 ●●●●●●● 《 ●●●●●●● 《 ●●●●●●● 《 ●●●●●●● 《 ●●●●●●● 《 ●●●●●●	
后退(B)	继续

图 1.3.14 设置用户名和密码

设置好用户名和密码以后点击"继续"按钮,系统就会开始正式安装,如下图所示:

Ubuntu 有着令人惊艳的 Rhythmbox 音乐					Categories
播放器。通过局级的播放模式选坝,排列 应你理相的变曲十公应目。只如一定能与	✓ Local collection	Artist			Album
CD 和価推式充斥援协器组织协善应 所以	■ Play Queue	All 3 artis	its (11)		1 album (11)
你干论去哪里都可以欣赏你所有的音乐。	də Music	Bleachers	(9)		Strange Desire (1
心元论在帝王即马林欣风心川自时自小。	🔊 Podcasts	Bleachers	Feat. Grimes (1)		
7714 46 /4	▼ Online sources	Bleachers	Feat. Yoko Ono (1)		
预装软件	🖄 Radio				
Bhythmbox 音乐播放器	∞ Last.fm				
	₽ Libre.fm	40 Track	Title	Genre	Artist
	▼ Playlists	0.1	Wild Heart	Pop	Bleachers
6	A Mu Tan Datad	2	Rollercoaster	Pop	Bleachers
		CLIE	DC	Pop	Bleachers
		CHE	IN Better	Pop	Bleachers
				Pop	Bleachers
			ve	Pop	Bleachers
			Runs	Pop	Bleachers
			1 Myster	у Рор	Bleachers
	AP .		You to L	ove Pop	Bleachers
	4		ay	Рор	Bleachers Feat. G
			A Move (20 POD	Hieachers Feat. Yr

图 1.3.15 系统安装中



原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php

等待系统安装完成,安装过程中会下载一些文件,所以要电脑能够正常上网,如果不能 正常上网的话也没有问题,对于系统的安装没有任何影响,安装完成以后提示重启系统,如 下图所示:



击或按 Ctrl+G。

图 1.3.16 安装完成,重启系统

重启系统以后会提示移除安装媒介,然后重启,如下图所示:



图 1.3.17 提示移除安装媒介, 然后重启

此时,我们先按"Ctrl"+"Alt"键,将鼠标定向到虚拟机界面,然后点击菜单栏中"虚 拟机"选项,找到"电源"选项,然后选择"关闭客户机",如下图所示:

原子哥在	线教学: ww	W.yu 54 位 - VN	an	zige.com	论坛	:www.ope	ne	dv.com/forun	n.php	
	文件(F) 编辑(E) i	查看(V)	虚拟	(机(M) 选项卡(T) ;	帮助(H)	• ₽ ₽	4	₽ □ □ □		
	库 ② 在此处键入内容; □ □ 我的计算机 □ Ubuntu 6- □ Ubuntu 1- □ Ubuntu 20 □ 共享的虚拟机	进行搜索 4 位 6 64 位 0_04_4 64	U S 子 C C	电源(P) 可移动设备(D) 暫停(U) 发送 Ctrl+Alt+Del(E) 抓取输入内容(I) SSH(H) 快照(N) 捕获屏幕 (C) 管理(M))	Ctrl+Shift+P Ctrl+G Ctrl+Alt+PrtScn		启动客户机(T) 关闭客户机(D) 挂起客户机(N) 重新启动客户机(E) 开机(P) 关机(O) 挂起(S) 重置(R) 打开电源时进入固件(F)	Ctrl+B Ctrl+E Ctrl+J Ctrl+R	
在弹	山的确认关标	北窗□	7	型新安装 VMware 10 设置(s) 図 1 □ 点击"关材	.3.18 关 几",如	_{Ctrl+D} 闭客户机 1下图所示:				

VMware Workstation					
?	确实要关闭"Ubuntu 64 位″的客户机操作系统吗?				
	□不再显示此消息(S)				
	关机取消				

图 1.3.19 关闭虚拟机

关闭 Ubuntu 操作系统后,打开 VMware 的虚拟机设置界面,然后选中 "CD/DVD(SATA)",在右侧的"连接"选项中选择"使用物理驱动器",然后点击"确定",如下图所示:

领航者 ZYNQ 之嵌入式 Linux 开发指南ご正原子哥在线教学: www.yuanzige.com论坛:www.openedv.com/forum.php

虚拟机设置						×
硬件 选项						
设备 ■ 内存 碰盘 (SCSI) ④ 碰盘 (SCSI) ● 阿络道歐器 ④ USB 控制器 ④ 声卡 合打印机 ■ 显示器	摘要 8 GB 8 449 GB 自动检测 存在 自动检测 存在 自动检测	移除(R)	 设备状态 □ 已连报 ② 使用 ③ 使用 ③ 使用 	 ^(C) ^ji 连接(O) ^j加理驱动器(P): か理驱动器(P): か短辺 		→ 浏览(B) 高级(V)
				确定	取消	帮助

正点原子

图 1.3.20 弹出 Ubuntu 安装镜像

接下来点击图 1.2.20 中的"开启此虚拟机"按钮,打开安装好的 Ubuntu 系统。在弹出的 "无法连接虚拟设备 sata0:1"提示窗口中,点击"否",以后每次打开系统就不会再弹出此 窗口了,如下图所示:



图 1.3.21 关闭提示窗口

进入登陆页面后,点击用户名,输入密码进入系统,如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php ^{星期二 23 : 49} 分 マ 注 砂 ウ マ

zuozhongkai 未列出?	
ubuntu®	

图 1.3.22 输入密码登陆系统

进入系统后,点击"欢迎使用 Ubuntu"→"退出",退出欢迎界面,如下图所示:



图 1.3.23 退出欢迎界面

注意,如果系统弹出升级 Ubuntu,我们直接选择"不升级"。如果这里升级系统,后面 Petalinux 安装会有问题,如下图所示:

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

正点原子



图 1.3.24 不升级系统





图 1.3.25 Ubuntu 系统

到这里,VMware 虚拟机以及 Ubuntu 系统就安装完成了。接下来我们安装 Vmware Tools, 方便使用 Ubuntu 系统。

1.4 安装 Vmware Tools

首先在 Vmware 中启动 Ubuntu 系统, 然后在 Vmware 的菜单栏, 选择"虚拟机(M)"选项 下的"安装 VMware Tools(T)",如下图所示:



原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php



图 1.4.1 安装 VMware Tools(T)

稍等片刻后,在桌面生成"Vmware Tools"图标,如下图所示:





双击"VMware Tools"图标,进入VMware Tools文件浏览器,如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php < > 🖣 💿 VMware Tools 🕨 Q 0 最近使用 tar.gz 主目录 企 manifest. run_ VMwareTo vmwarevmware-桌面 txt upgrader. ols-10.3.10toolstoolssh 13959562. upgraderupgrader-视频 tar.gz 32 64 图片 Ō D ⇒ 下载 99 Î 回收站 ▲ 其他位置 +

图 1.4.3 VMware Tools 文件浏览器界面

双击上图箭头所指的文件,在弹出的界面中选择"提取",如下图所示:

提取 -	⊦ VMwareT	ools-10.3	3.10-13959562	.tar.gz [只读]	ୟ ≡	
$\langle \rangle$	✿ 位置(L):	i /				
名称		•	大小	类型	已修改	
w mware	-tools-distrib		164.3 MB	文件夹	2019年6月	13日 20:03

图 1.4.4 在弹出的界面中选择"提取"

弹出选择提取的目录,这里选择"主目录",然后点击右上角的"提取(E)"按钮,如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

取消	肖(C)	提取		۹	提取(E)
Ø	最近使用	 ▲ 1 sqd 		/	53
<u>۵</u>	主目录	名称	▲ 大小	-	修改日期
	桌面	② 公共的			14:12
	视频	顧 模板			14:12
٥	图片	■ 视频			14:12
۵	文档	■ 图片			14:12
∻	下载	▶ 文档			14:12
99	音乐	▶ 下载			14:12
	回收站	🚺 音乐			14:12
0		■ 桌面			14:12
		examples.desktop	9.0 KE	3	14:10

图 1.4.5 选择主目录

提取结束后,进入主目录,可以看到有一个"vmware-tools-distrib"文件夹,如下图所示:

<		企主	文件夹	vmwa	are-tools-dis	strib 🕨		٩	:=	≡
Ø	最近使用]						EB	-	
۵			vmwa	are-	公共的	模材	U .	视频		图片
	桌面		too dist	ls- rib	A7(b)	17.1	~	17077		
H	视频								-	
ø	图片									.
	文档		文材	当	下载	音法	к :	泉面		示例
`	下载				\mathbf{X}					
99	音乐									
	回收站									

图 1.4.6 "vmware-tools-distrib"文件夹

进入该文件夹,在空白处单击鼠标右键,在弹出的菜单中,选择"在终端打开(E)",如下 图所示:

正点原子 领航者 ZYNQ 之嵌入式 Linux 开发指南 原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php く > (企 主文件夹 vmware-tools-distrib) Q **:**= | ≡ 0 最近使用 ŵ 主目录 bin caf doc etc installer 桌面 视频 Perl lib vgauth FILES INSTALL vmware-图片 0 install.pl D 文档 ⇒ 下载 新建文件夹(F) 99 音乐 回收站 圙 全选(A) 0 ▲

图 1.4.7 选择 "在终端打开(T)"

属性(P)

在终端打开(E)

还原丢失的文件..

在打开的终端中输入"II"命令(注:两个小写的L)或"ls-I"命令,列出该文件夹下的文件结构,如下图所示:

选中了"VMwareTools-

+

其他位置

具他似直

😣 🖱 💷 guest-6uve8f@qinqingzhou-virtual-machine: ~/vmware-tools-distrib						
guest-6uve8f@qinqingzhou-virtual-machine 总用量 372	:~/vmware-t	ools-distrib\$ ll 🔶				
drwxr-xr-x 9 guest-6uve8f guest-6uve8f	240 6月	13 20:03 ./				
drwx 19 guest-6uve8f guest-6uve8f	540 11月	1 16:35/				
drwxr-xr-x 2 guest-6uve8f guest-6uve8f	100 6月	13 20:03 bin/				
drwxr-xr-x 5 guest-6uve8f guest-6uve8f	100 6月	13 20:03 caf/				
drwxr-xr-x 2 guest-6uve8f guest-6uve8f	100 6月	13 20:03 doc/				
drwxr-xr-x 5 guest-6uve8f guest-6uve8f	400 6月	13 20:03 etc/				
-rw-rr 1 guest-6uve8f guest-6uve8f :	146996 6月	13 20:03 FILES				
-rw-rr 1 guest-6uve8f guest-6uve8f	2538 6月	13 20:03 INSTALL				
drwxr-xr-x 2 guest-6uve8f guest-6uve8f	120 6月	13 20:03 installer/				
drwxr-xr-x 14 guest-6uve8f guest-6uve8f	280 6月	13 20:03 lib/				
drwxr-xr-x 3 guest-6uve8f guest-6uve8f	60 6月	13 20:03 vgauth/				
-rwxr-xr-x 1 guest-6uve8f guest-6uve8f	227024 6月	13 20:03 vmware-install.pl*				
guest-6uve8f@qinqingzhou-virtual-machine	:~/vmware-t	ools-distrib\$				

图 1.4.8 输入" 11" 命令

然后输入命令"sudo ./vmware-install.pl",会提示输入用户密码,输入密码后并回车,弹出下图所示内容:

qinqingzhou@qinqingzhou_virtual-machine:~/vmware-tools-distrib\$ sudo ./vmware-install.pl
[sudo] qinqingzhou的密码: 🖕 🐂
open-vm-tools packages are available from the OS vendor and VMware recommends 📃 🥄
using open-vm-tools packages. See http://kb.vmware.com/kb/2073803 for more
information.
Do you still want to proceed with this installation? [no] yes

图 1.4.9 命令"sudo ./vmware-install.pl"

这里提示该操作系统可以直接安装 open-vm-tools 安装包(可通过命令 apt install open-vm-tools 安装),不需要安装 VMware-Tools,这里笔者根据多年的使用经验建议不要安装 open-vm-tools 安装包,因为在使用的过程中会出现一些问题,比如找不到共享的文件夹,没必要



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

浪费时间在解决这种问题上。如果是安装 VMware-Tools,虽然没有安装 open-vm-tools 安装包 方便,但笔者使用至今也没出现过什么问题。所以这里的"Do you still want to proceed with this installation?[no]"后面输入"yes"。

回车后,后面出现这种停下来进行选择的,直接一路回车即可,如下图所示,箭头所指处都是直接回车。



图 1.4.10 使用默认配置

留意一下,可以看到下图所示的信息,这里就是是否启用虚拟机和主机共享文件夹的功能和是否启用在虚拟机和主机之间拖曳和复制文件的功能,默认为 yes,也是一路回车即可。

The VMware Host-Guest Filesystem allows for shared folders between the host OS and the guest OS in a Fusion or Workstation virtual environment. Do you wish to enable this feature? [yes] INPUT: [yes] default

The vmxnet driver is no longer supported on kernels 3.3 and greater. Please upgrade to a newer virtual NIC. (e.g., vmxnet3 or e1000e)

The vmblock enables dragging or copying files between host and guest in a Fusion or Workstation virtual environment. Do you wish to enable this feature? [yes]

INPUT: [yes] default

图 1.4.11 启用文件共享功能

最后弹出下图所示信息,也就是安装完成了,需要重启图形界面,直接重启虚拟机里的 Ubuntu 系统就好了。



原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php

You must restart your X session before any mouse or graphics changes take effect. To enable advanced X features (e.g., guest resolution fit, drag and drop, and file and text copy/paste), you will need to do one (or more) of the following: 1. Manually start /usr/bin/vmware-user 2. Log out and log back into your desktop session 3. Restart your X session.

Found VMware Tools CDROM mounted at /media/qinqingzhou/VMware Tools. Ejecting device /dev/sr0 ... Enjoy,

-the VMware team

图 1.4.12 完成安装

重启 Ubuntu 之后, 接下来我们就要学习如何使用 Ubuntu 了。

另外可以发现安装完 Vmware Tools 后,Ubuntu 桌面的大小会自动适应屏幕的大小,而不 是像原先那样显示很小的一部分。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

第二章 Ubuntu 系统入门

在上一章我们已经安装好虚拟机,并且在虚拟机中安装好了 Ubuntu 操作系统了,本章我 们就来学习 Ubuntu 系统的基本使用,通过本章的学习为我们以后的开发做准备。Ubuntu 系统 是和 Windows 一样的大型桌面操作系统,因此功能非常强大,不是一章就能介绍完的,因此 本章叫做《Ubuntu 系统入门》。本章的主要目的是教会读者掌握后续嵌入式开发所需的 Ubuntu 基本技能,比如系统的基本设置、常用的 shell 命令、vim 编辑器的基本操作等等,如 果想详细的学习 Ubuntu 操作系统请参考其它更为详细的书籍,本章参考了《Ubuntu Linux 从 入门到精通》,这本书不厚,很适合用来作 Ubuntu 入门。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

2.1 Ubuntu 系统初体验

2.1.1 hello Ubuntu

上一章我们已经安装好了 Ubuntu 操作系统,我们再来回顾一下如何开机: 1、打开 VMware 虚拟机软件,打开以后如图 2.1.1 所示:

Ubuntu 64 1 <u>v</u> - VMware Workstation				- L X
文件(F) 编辑(E) 查看(V) 虚拟机(M)	选项卡(T) 帮助(H) 🕨 🗸	육 🗘 🔑	₽	
库 × <	 ☆ 主页 × □ Ubuntu 64 位 □ Ubuntu 64 位 ▶ 开启此虚拟机 □ 編載虚拟机设置 	×	^	
	 ▼ 设备 □ 内存 □ 砂址理器 □ 硬盘 (SCSI) ③ CD/DVD (SATA) □ 内 网络适配器 ← USB 控制器 ◆ 小 声卡 □ 日印机 □ 显示器 	8 GB 4 196 GB 自动检测 桥接模式 (自 存在 自动检测 存在 自动检测		 ✓ 虚拟机详细信息 状态:已关机 配置文件: I\Ubuntu 64 位.vmx 硬件兼容性: Workstation 15.x 虚拟机 主 IP 地址:网络信息不可用
,, ,	Later IN		-	

图 2.1.1WMware 主界面

2、打开 VMware 上的开机按钮, 打开方式如图 2.1.2 所示:

选项卡(T) 帮助(H) 🕨 🗸	두 우 우 우 🔲 🗆 🖸 🛛 ▷ 🖉 -	
🕜 主页 🛛 🗋 Ubuntu 64 🖄		
「」 Ubuntu 64 位	两个都可以开机	
 ▶ 开启此虚拟机 □ 编辑虚拟机设置 		
▼ 设备		
興 内存	8 GB	
心 处理器	4	
□ @ 硬盘 (SCSI)	196 GB	
💿 CD/DVD (SATA)	自动检测	
	桥接模式 (自	

图 2.1.2VMware 开机按钮

3、点击上图中两个开机按钮中的任意一个就会打开 Ubuntu 操作系统,首先进入图 2.1.3 所示的登陆界面,点击使用的用户名,输入密码即可进入系统。





图 2.1.3Ubuntu 登陆界面

在登陆界面输入密码,进入系统主界面,如图 2.1.4 所示:



图 2.1.4Ubuntu 主界面

进入主界面以后大家就可以看到和 Windows 基本一样, 左侧有一列 APP, 第一个是火狐 浏览器, 可以用来上网, 比如我们登陆百度网站, 如图 2.1.5 所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php



图 2.1.5firefox 浏览器

第二个是邮箱。第三个是文件浏览器,打开以后可以浏览 Ubuntu 系统中的文件,打开以 后如图 2.1.6 所示:

<	→ < 企主	文件夹			٩	=	•••
\odot	最近使用						
ŵ		vmware-	公开的		シロル市		
	桌面	tools-	公共的	保収	↑光 少贝	图片	
-	视频		·				
ø	图片			53		3	
D	文档	文档	下载	音乐	桌面	示例	
⇒	下载						
99	音乐						
	回收站						
+	其他位置						

图 2.1.6 文件浏览

这里还有其它一些 APP,大家可以自行打开看一下这些 APP 都是干啥的,这里就不一一详细的介绍了。

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

正点原子

2.1.2 系统设置

我们以设置取消锁屏为例来讲解如何使用设置。打开系统设置界面,打开方式为点击桌 面最右上角的图标,然后在弹出的界面中选择工具按钮,如图 2.1.7 所示:



图 2.1.7 打开系统设置

打开以后的系统设置界面如下图所示:

٩	设置	● © <u>⊗</u>
()-	Wi-Fi	
*	蓝牙	
4	背景	
D	Dock	
	通知	
۹	搜索	
0	区域和语言	
0	通用辅助功能	
€Ds	在线帐户	未发现 Wi-Fi 适配器 请确认您已插入 Wi-Fi 适配器并打开
4	隐私	
<	共享	

图 2.1.8 系统设置界面

系统设置界面可以完成系统的大部分设置。我们找到"电源"设置并点击,点击以后如 下图所示:

原子哥在线教	教学: www.yuanz	ige.com	论坛:www.op	enedv.com/for	um.php
	Q 设置	_	电》	原	
	◙ 区域和语言		节电		
	● 通用辅助功能		空白屏幕(B)	5分钟 ▼	
	● 在线帐户		体打到大型体包		
	≝ 隐私		白动性纪(4)	¥	
				~	
	■1) 声音				
	健 电源				
	₽ 网络				
	动 设备	>			
	〕 详细信息	>			

🔁 正点原子

图 2.1.9 电源设置界面

从上图中可以看到,空白屏幕的时间是5分钟,也就是5分钟不动 Ubuntu 系统,就会进入锁屏界面,而每次从锁屏进入桌面都要输入密码,比较麻烦,所以笔者将其设置为"从不",如下图所示:

٩	设置		ŧ	ョ源		
◎ 区域和语	÷	节电	1			
☑ 通用辅助	功能	호	空白屏幕(B)		从不 🔻	
● 在线帐户		+± ±2	110 - 24 - 40 - 40 - 611	1		
≝ 隐私		住起		/		
< 共享		E	目动挂起(A)	/_	<u>天</u>	
() 声音						
Cŧ 电源						
₽ 网络						
☜ 设备	>					
■ 详细信息	>					

图 2.1.10 调整空白屏幕时间

也就是无论多长时间不动 Ubuntu 系统,都不会进入锁屏界面。通过设置取消锁屏这个例 子,我们就知道了如何设置 Ubuntu 系统,如果有需要设置其它东西的话都可以到系统设置里 面去进行,这里就不一一详细的介绍了。

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

正点原子

2.1.3 系统重启与关机

当我们不使用 Ubuntu 系统以后就需要将其关机,就和我们使用 Windows 系统一样,千万 不要通过直接退出 VMware 软件来关机!!关机很简单,在主界面,点击右上角的图标,如 下图所示:



图 2.1.11 关机

弹出下图所示界面:

	秒后自动关机。	(从不)、				
取消	重启	关机				

图 2.1.12 关机确认界面

注意,该界面显示系统将在60秒后自动关机,也就是此时什么都不做,60秒后系统将会 自动关机。

在图 2.1.12 中可以看到有三个选项:取消、重启和关机,这个和 Windows 下是一样的, 你如果想要取消就点击"取消"按钮,想要关机就点击"关机"按钮。以关机为例,点击 "关机"以后会直接关闭 Ubuntu 系统。

2.1.4 中文输入测试

我们是中国人,平时用的最多的肯定是中文,那么 Ubuntu 下中文输入是否和 Windows 一 样呢?如何在 Ubuntu 下使用中文输入法?



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

我们在安装 Ubuntu 系统的时候就已经使用过中文输入法了,就是选择我们所在地的时候。 本节我们就以创建一个文本为例,介绍如何在 Ubuntu 中使用中文输入法。

在 Ubuntu 虚拟机中,点击左下角的"显示应用程序"图标,如下图所示:



图 2.1.13 点击左下角的"显示应用程序"

进入活动界面,输入"text",找到文本编辑器工具,如下图所示:



图 2.1.14 文本编辑器工具

单击打开文本编辑器工具,打开以后如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

打开(0) ▼	Æ	无标题文	と档 1	保存(s	5) =	•	• 😣
	纯文本 🔻	制表符宽度:	8 🔻	第1行,	第1列	•	插入

图 2.1.15 打开文档

打开文本编辑器工具后,我们可以尝试在里面输入一些英文和数字,输入英文和数字是 没有任何问题的,输入中文的话需要切换到 Ubuntu 自带的拼音输入法,有两种方式切换,一 种是使用快捷键:Windows+空格键,一种是使用鼠标点击设置输入法,如下图所示:



图 2.1.16 切换拼音输入法

这两种方法都可以切换输入法,切换到拼音输入法以后就可以输入中文了,如下图所示:

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

正点原子

zuozhongkai 123456789 士中ण				
左志凯 ARM <u>逻辑</u> luo ji 1 逻辑 2	洛基 3 罗技 4	罗5洛〈〉		
47	·····································	· شنب م	5.4 亿	 任く

图 2.1.17 输入中文文本

大家会发现 Ubuntu 下的拼音输入法使用起来跟 Windows 下的输入法相差不大。

通过上面几个小节中对 Ubuntu 的基本操作来看,基本和 Windows 下的操作差不多,我们 真正要使用 Ubuntu 的不是通过图形界面操作, 而是通过命令行操作的。这也是我们接下来着 重要讲的: Ubuntu(Linux)终端操作,会涉及到很多命令,但是常用的命令就那几十个,不需 要刻意的去背,使用习惯了就自然记住了。不要看到要记命令就觉得可怕。根据 2080 原则, 80%情况下只使用那 20%的命令,实际情况会更少,常用的可能就那 5%~10%的命令。

2.2 Ubuntu 终端操作

本节就是我们学习 Ubuntu 操作系统的重点了,终端操作,也就是俗称的"敲命令",不 管是哪个版本的 Linux 发行版系统,它都会提供终端操作, Linux 下的终端操作类似与 Windows 下的 DOS 操作。要使用终端首先肯定是要打开终端,在主界面上点击鼠标右键,然 后选择打开终端,如图 2.2.1 所示:

新建文件夹(F)	Shift+Ctrl+N
粘贴(P)	
全选(A)	Ctrl+A
✔保持对齐(K)	
按名称整理桌面(D)	
更换背景(B)	
打开终端(E)	

图 2.2.1 打开终端



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

打开终端以后如图 2.2.2 所示:



图 2.2.2 终端界面

我们就是在图 2.2.2 所示界面上输入命令的,终端默认会有类似下面一行所示的一串提示符:

zuozhongkai@zuozhongkai-virtual-machine: ~\$

上述字符串中,@前面的"zuozhongkai"是当前的用户名字,@后面的 zuozhongkaivirtual-machine 是我的机器名字。最后面的符号"\$"表示当前用户是普通用户,我们可以在 提示符后面输入命令,比如输入命令"ls",命令"ls"是打印出当前所在目录中所有文件和 文件夹,如图 2.2.3 所示:



图 2.2.3 1s 命令

在图 2.2.3 中我们输入了"ls"这个命令,然后打印出了当前目录下的所有文件和文件夹, 后面我们学习命令的时候就是在终端中输入相应命令的。

2.3 Shell 操作

2.3.1 Shell 简介

学习 linux 的时候会频繁的看到 Shell 这个词语?那么什么是 Shell 呢?网上搜索一下,各种专业的解释一堆,但是对于第一次接触 Linux 的人来说这些专业的词语只会让人更晕。简单的说 Shell 就是敲命令。国内把 Linux 下通过命令行输入命令叫做"敲命令",国外人玩的比较洋气,人家叫做"Shell"。因此以后看到 Shell 这个词语第一反应就是在终端中敲命令,将多个 Shell 命令按照一定的格式放到一个文本中,那么这个文本就叫做 Shell 脚本。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

严格意义上来讲, Shell 是一个应用程序, 它负责接收用户输入的命令, 然后根据命令做 出相应的动作, Shell 负责将应用层或者用户输入的命令传递给系统内核, 由操作系统内核来 完成相应的工作, 然后将结果反馈给应用层或者用户。

2.3.2 Shell 基本操作

前面我们说 Shell 就是"敲命令",那么既然是命令,那肯定是有格式的, Shell 命令的格式如下:

command -options [argument]

command: Shell 命令名称。

options: 选项,同一种命令可能有不同的选项,不同的选项其实现的功能不同。

argument: Shell 命令是可以带参数的,也可以不带参数运行。

同样以命令"ls"为例,下面"ls"命令的三种不同格式其结果也不同:

ls

ls - l

ls /usr

这三种命令的运行结果如图 2.3.1 所示:

😣 🔿 🗉 🛛 zuozhongkai@ubuntu: ~			
zuozhongkai@ubuntu:~\$ ls			
examples.desktop tmp 公共的 模板	视频	图片	文档 下载 音乐 桌面
zuozhongkai@ubuntu:~\$ ls -l			
总用量 48			
-rw-rr 1 zuozhongkai zuozhongkai	8980	12月	18 02:08 examples.desktop
drwxrwxr-x 2 zuozhongkai zuozhongkai	4096	12月	19 00:29 tmp
drwxr-xr-x 2 zuozhongkai zuozhongkai	4096	12月	18 02:21 公共的
drwxr-xr-x 2 zuozhongkai zuozhongkai	4096	12月	18 02:21 模板
drwxr-xr-x 2 zuozhongkai zuozhongkai	4096	12月	18 02:21 视频
drwxr-xr-x 2 zuozhongkai zuozhongkai	4096	12月	18 02:21 图片
drwxr-xr-x 2 zuozhongkai zuozhongkai	4096	12月	18 02:21 文档
drwxr-xr-x 2 zuozhongkai zuozhongkai	4096	12月	18 02:21 下载
drwxr-xr-x 2 zuozhongkai zuozhongkai	4096	12月	18 02:21 音乐
drwxr-xr-x 2 zuozhongkai zuozhongkai	4096	12月	19 00:25 桌面
<pre>zuozhongkai@ubuntu:~\$ ls /usr</pre>			
bin games include lib local loca	ale s	bin	share src
zuozhongkaj@ubuntu:~\$			

图 2.3.1 ls 命令

在图 2.3.1 中 "ls" 命令用来打印出当前目录下的所有文件和文件夹,而 "ls -l" 同样是 打印出当前目录下的所有文件和文件夹,但是此命令会列出所有文件和文件夹的详细信息, 比如文件大小、拥有者、创建日期等等。最有一个 "ls /usr" 是用来打印出目录 "/usr"下的 所有文件和文件夹。

Shell 命令是支持自动补全功能的,因为 Shell 命令非常多,如果不作自动补全的话就需要 用户去记忆这些命令的全部字母。使用自动补全功能以后我们只需要输入命令的前面一部分 字母,然后按下 TAB 键,如果只有一个命令匹配的话就会自动补全这个命令剩下的字母。如 果有多个命令匹配的话系统就会发出报警声音,此时再按下一次 TAB 键就会列出所有匹配的 命令,比如我们输入字母"ip" (注意,ip命令后面有空格),然后按下两次 TAB 键,结果 如图 2.3.2 所示:

maddress

monitor



vrf

xfrm

原子哥在线教学:	www.yuanzige.co	m 论坛:v	www.openedv.c	om/forum.php	
sqd@sqd-	virtual-machine	:~\$ ip			
address	l2tp	mroute	ntable	token	
addrlabe	l link	mrule	route	tunnel	
fou	macsec	neigh	rule	tuntap	

netconf

netns

图 2.3.2 "if"开始的命令

sг

tcp_metrics

从上图可以看出, ip 工具命令有 25 个, 我们以"ip address"为例, 输入"ip a", 按一 下 TAB 键, 会自动补全"ip addr",但不是我们想要的"ip address",这是因为以 a 开头的命 令有"address"和"addrlabel"两个,继续按两次 TAB 键可以看到这两个命令,如下图所示:

<mark>sqd@sqd-</mark> v address	r <mark>irtual-machine:~</mark> \$ ip addr addrlabel	
	图 2.3.3 "ip a"开始的命令	

此时,我们继续输入"e",然后按下 TAB 键,就会自动补全"ip address",按下回车 键,就能查看 ip 地址,如下图所示:

sqd@sqd-virtual-machine:~\$ ip address 👞
1: lo: <loopback,up,lower_up> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000</loopback,up,lower_up>
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
inet 127.0.0.1/8 scope host lo
valid_lft forever preferred_lft forever
inet6 ::1/128 scope host
valid_lft forever preferred_lft forever
2: ens33: <broadcast,multicast,up,lower_up> mtu 1500 qdisc fq_codel state UP group default qlen 1000</broadcast,multicast,up,lower_up>
<pre>link/ether 00:0c:29:7a:26:76 brd ff:ff:ff:ff:ff</pre>
inet 192.168.2.131/24 brd 192.168.2.255 scope global dynamic noprefixroute ens33
valid_lft 85333sec preferred_lft 85333sec
inet6 fe80::bb2b:2d44:963c:ae63/64 scope link noprefixroute
valid_lft forever preferred_lft forever

图 2.3.4 ip address 命令结果

2.3.3 常用 Shell 命令

help

ila

我们做嵌入式开发用的最多就是 Shell 命令, Shell 命令是所有的 Linux 系统发行版所通用 的,并不是说我在 Ubuntu 下学会了 Shell 命令,换另外一个 Linux 发行版操作系统以后就没用 了(不同的发行版 Linux 系统可能会自定义一些命令)。本节我们先来介绍一些 Shell 下常用的 命令:

1、目录信息查看命令 ls

文件浏览是最基本的操作了, Shell 下文件浏览命令为 ls, 格式如下:

ls [选项] [路径]

ls 命令主要用于显示指定目录下的内容,列出指定目录下包含的所有的文件以及子目录, 它的主要参数有:

-a 显示所有的文件以及子目录,包括以"."开头的隐藏文件。

- -**l** 显示文件的详细信息,比如文件的形态、权限、所有者、大小等信息。
- -t 将文件按照创建时间排序列出。
- -A 和-a一样, 但是不列出"."(当前目录)和".."(父目录)。

-R 递归列出所有文件,包括子目录中的文件。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

Shell 命令里面的参数是可以组合在一起用的,比如组合"-al"就是显示所有文件的详细 信息,包括以"."开头的隐藏文件,ls命令使用如图 2.3.5 所示:

<pre>zuozhongkai@ubuntu:~/tmp\$</pre>	ls						
a b c							
<pre>zuozhongkai@ubuntu:~/tmp\$</pre>	ls -a						
a b c							
<pre>zuozhongkai@ubuntu:~/tmp\$</pre>	ls -al						
总用量 8							
drwxrwxr-x 2 zuozhongkai	zuozhongkai	4096	12月	19	21:41		
drwxr-xr-x 20 zuozhongkai	zuozhongkai	4096	12月	19	20:31		
-rw-rw-r 1 zuozhongkai	zuozhongkai	0	12月	19	21:41	а	
-rw-rw-r 1 zuozhongkai	zuozhongkai	0	12月	19	21:41	b	
-rw-rw-r 1 zuozhongkai	zuozhongkai	0	12月	19	21:41	С	

图 2.3.5 ls 命令演示

注意上图中 tmp 文件夹是我为了演示方便,自己创建的,里面的文件 a, b 和 c 也是我创建的,关于文件夹和文件的创建后面会详细的讲解。

2、目录切换命令 cd

要想在 Shell 中切换到其它的目录,使用的命令是 cd,命令格式如下:

cd [路径]

路径就是我们要进入的目录路径,比如下面所示操作:

cd / //进入到根目录"/"下, Linux 系统的根目录为"/",

cd /usr //进入到目录 "/usr" 里面。

- cd .. //进入到上一级目录。
- cd ~ //切换到当前用户主目录

比如我们要进入到目录"/usr"下去,并且查看"/usr"下有什么文件,操作如下图所示:

zuozhongkai@ubuntu:~\$ cd /usr zuozhongkai@ubuntu:/usr\$ ls bin games include lib local locale sbin share src

图 2.3.6 cd 命令演示

在上图中,我们先使用命令"cd /usr"进入到"/usr"目录下,然后使用"ls"命令显示 "/usr"目录下的所有文件。仔细观察上图可以看到,当我们切换到其它目录以后在符号"\$" 前面就会以蓝色的字体显示出当前目录名字,如下图所示:

zuozhongkai@ubuntu:/usr\$ls bin games include lib local locale sbin share src

图 2.3.7 目录路径显示

3、当前路径显示命令 pwd

pwd 命令用来显示当前工作目录的绝对路径,不需要任何的参数,使用如下图所示:



论坛:www.openedv.com/forum.php

zuozhongkai@ubuntu:~\$ pwd
/home/zuozhongkai

图 2.3.8 pwd 命令

4、系统信息查看命令 uname

原子哥在线教学: www.yuanzige.com

要查看当前系统信息,可以使用命令 uname,命令格式如下:

uname [选项]

可选的选项参数如下:

- -r 列出当前系统的具体内核版本号。
- -s 列出系统内核名称。
- **-o**列出系统信息。

使用如下图所示:



图 2.3.9 uname 命令操作

5、清屏命令 clear

clear 命令用于清除终端上的所有内容,只留下一行提示符。

6、切换用户执行身份命令 sudo

Ubuntu(Linux)是一个允许多用户的操作系统,其中权限最大的就是超级用户 root,有时候我们执行一些操作的时候是需要用 root 用户身份才能执行,比如安装软件。通过 sudo 命令可以使我们暂时将身份切换到 root 用户。当使用 sudo 命令的时候是需要输入密码的,这里要注意输入密码的时候是没有任何提示的!命令格式如下:

sudo [选项] [命令]

选项主要参数如下:

- -h 显示帮助信息。
- -1 列出当前用户可执行与不可执行的命令
- -p 改变询问密码的提示符。

假如我们现在要创建一个新的用户 test,创建新用户的命令为"adduser",创建新用户 的权限只有 root 用户才有,我们在装系统的时候创建的那个用户是没有这个权限的,比如我 的"zuozhongkai"用户。所以创建新用户的话需要使用"sudo"命令以 root 用户执行 "adduser"这个命令,如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

zuozhongkai@ubuntu:~\$ adduser test adduser: 只有 root 才能将用户或组添加到系统。 zuozhongkai@ubuntu:~\$ sudo adduser test 正在添加用户"test"... 正在添加新组"test" (1001)... 正在添加新用户"test" (1001) 到组"test"... 创建主目录"/home/test"... 正在从"/etc/skel"复制文件. 输入新的 UNIX 密码: 重新输入新的 UNIX 密码: passwd: 已成功更新密码 正在改变 test 的用户信息 请输入新值,或直接敲回车键以使用默认值 · []: 名 房间号码 11: ii: []: 作电话 电话 []: 正确? [Y/n] y

图 2.3.10 sudo 命令演示

在图 2.3.10 中,我们一开始直接使用"adduser test"命令添加用户的时候提示我们 "adduser:只有 root 才能将用户或组添加到系统。"所以我们要在前面加上"sudo"命令, 表示以 root 用户执行 adduser 操作。

7、添加用户命令 adduser

在讲解 sudo 命令的时候我们已经用过命令"adduser",此命令需要 root 身份去运行。命 令格式如下:

adduser [参数] [用户名]常用的参数如下:-system 添加一个系统用户-home DIR DIR 表示用户的主目录路径-uid ID ID 表示用户的uid。-ingroup GRP 表示用户所属的组名。

adduser 的使用我们前面已经演示过了,大家可以试着再添加一个用户。

8、删除用户命令 deluser

前面讲了添加用户的命令,那肯定也有删除用户的命令,删除用户使用命令"deluser", 命令参数如下:

deluser [参数] [用户名]主要参数有:-system当用户是一个系统用户的时候才能删除。-remove-home删除用户的主目录-remove-all-files删除与用户有关的所有文件。-backup备份用户信息同样的,命令 "deluser" 也要使用 "sudo" 来以 root 用户运行,以删除我们前面创建的

用户 test 为例, deluser 使用如下图所示:



原子哥在线教学:	www.yuanzige.com	论坛:www.openedv.com/forum.php
zuozhongka: 正在寻找要看	L@ubuntu:~\$ sudo deluser · 备份或删除的文件	remove-all-files test
/usr/sbin/o	deluser: 无法处理特殊文件	/sys/kernel/security/apparmor/.null
/usr/sbin/d	deluser:无法处理特殊文件 deluser:无法处理特殊文件	/run/udev/static_node-tags/uaccess/snd\x21seq /run/udev/static_node-tags/uaccess/snd\x2ftimer
/usr/sbin/o /usr/sbin/o	deluser: 无法处理特殊文件 deluser: 无法处理特殊文件	/dev/vcsa7 /dev/vcs7
/usr/sbin/d /usr/sbin/d /usr/sbin/d 正在删除文f 正在删除用 警告:组"te 完成。	deluser: 无法处理特殊文件 deluser: 无法处理特殊文件 deluser: 无法处理特殊文件 deluser: 无法处理特殊文件 件 户 'test' est"没有其他成员了。	/lib/systemd/system/single.service /lib/systemd/system/cryptdisks.service /lib/systemd/system/cryptdisks-early.service

图 2.3.11 命令 deluser 演示

9、切换用户命令 su

前面在讲解命令"sudo"的时候说过,"sudo"是以 root 用户身份执行一个命令,并没 有更改当前的用户身份,所有需要 root 身份执行的命令都必须在前面加上"sudo"。命令 "su"可以直接将当前用户切换为 root 用户,切换到 root 用户以后就可以尽情地尽情任何操 作了!因为你已经获得了系统最高权限,在 root 用户下,所有的命令都可以无障碍执行,不 需要在前面加上"sudo","su"命令格式如下:

su [选项] [用户名]

常用选项参数如下:

-c -command 执行指定的命令,执行完毕以后回复原用户身份。

-login 改变用户身份,同时改变工作目录和 PATH 环境变量。

-m 改变用户身份的时候不改变环境变量

-h 显示帮助信息

以切换到 root 用户为例,使用如下图所示:

zuozhongkai@ubuntu:~\$ sudo su [sudo] zuozhongkai 的密码: root@ubuntu:/home/zuozhongkai#

图 2.3.12 su 命令演示

在图 2.3.12 中,先使用命令 "sudo su" 切换到 root 用户, su 命令不写明用户名的话默认 切换到 root 用户。然后输入密码,密码正确的话就会切换到 root 用户,可以看到切换到 root 用户以后提示符的 "@"符号前面的用户名变成了 "root",表示当前的用户是 root 用户。并 且以 "#"结束。

注意!! 由于 root 用户权限太大,稍微不注意就可能删除掉系统文件,导致系统奔溃,因此强烈建议大家,不要以 root 用户运行 Ubuntu。当要用到 root 身份执行某些命令的时候使用 "sudo"命令即可。

要切换回原来的用户,使用命令"sudo su 用户名"即可,比如我要从 root 切换回 zuozhongkai 这个用户,操作如下图所示:

root@ubuntu:/home/zuozhongkai# sudo su zuozhongkai
zuozhongkai@ubuntu:~\$

图 2.3.13 切换回原来用户



正点原子

10、显示文件内容命令 cat

查看文件内容是最常见的操作了,在 windows 下可以直接使用记事本查看一个文本文件 内容, linux 下也有类似记事本的软件, 叫做 gedit, 找到一个文本文件, 双击打开, 默认使用 的就是 gedit, 如下图所示:

😣 🖻 🗊 *无标题文体	档 (~/桌面) - gedit		
打开(0) ▼ 用			保存(S)
zuozhongkai 123456789			
ARM裸机与嵌入式Li	Inux驱动开发		
纯文本 ▼ 制	则表符宽度:8 ▼	行4,列1	▼ 插入

图 2.3.14 gedit 打开文档

我们现在讲解的是 Shell 命令, 那么 Shell 下有没有办法读取文件的内容呢? 肯定有的, 那就是命令"cat",命令格式如下:

cat [选项] [文件]

选	项	主	要	参	数	如	下	:
-n	由1开始对所表	有输出的行	进行编号。					

-b 和-n 类似, 但是不对空白行编号。

-s 当遇到连续两个行以上空白行的话就合并为一个行空白行。

比如我们以查看文件"/etc/environment"的内容为例,结果如下图所示:

PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games" zuozhongkai@ubuntu:~\$ cat /etc/environment -n PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games"

图 2.3.15 命令 cat 演示

11、显示和配置网络属性命令 if config

ifconfig 是一个跟网络属性配置和显示密切相关的命令,通过此命令我们可以查看当前网 络属性,也可以通过此命令配置网络属性,比如设置网络 IP 地址等等,此命令格式如下:

ifconfig interface options | address



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

主要参数如下:

interface 网络接口名称,比如 eth0 等。

up 开启网络设备。

down 关闭网络设备。

add IP 地址,设置网络 IP 地址。

netmask add 子网掩码。

命令 if config 的使用如下图所示:

zuozhongka	ai@ubuntu:~\$ ifconfig
ens33	Link encap:以太网 硬件地址 00:0c:29:96:89:d6 inet 地址:192.168.31.235 广播:192.168.31.255 掩码:255.255.255.0 inet6 地址: fe80::e92a:d882:e1b:7402/64 Scope:Link UP BROADCAST RUNNING MULTICAST MTU:1500 跃点数:1 接收数据包:13404 错误:0 丢弃:0 过载:0 帧数:0 发送数据包:967 错误:0 丢弃:0 过载:0 载波:0 碰撞:0 发送队列长度:1000 接收字节:810508 (810.5 KB) 发送字节:75728 (75.7 KB)
lo	Link encap:本地环回 inet 地址:127.0.0.1 掩码:255.0.0.0 inet6 地址: ::1/128 Scope:Host UP LOOPBACK RUNNING MTU:65536 跃点数:1 接收数据包:248 错误:0 丢弃:0 过载:0 帧数:0 发送数据包:248 错误:0 丢弃:0 过载:0 载波:0 碰撞:0 发送队列长度:1000 接收字节:19004 (19.0 KB) 发送字节:19004 (19.0 KB)
zuozhonaka	ai@ubuntu:~\$ ifconfig ens33
ens33	Link encap:以太网 硬件地址 00:0c:29:96:89:d6 inet 地址:192.168.31.235 广播:192.168.31.255 掩码:255.255.255.0 inet6 地址: fe80::e92a:d882:e1b:7402/64 Scope:Link UP BROADCAST RUNNING MULTICAST MTU:1500 跃点数:1 接收数据包:13406 错误:0 丢弃:0 过载:0 帧数:0 发送数据包:971 错误:0 丢弃:0 过载:0 载波:0 碰撞:0 发送队列长度:1000 接收字节:810658 (810.6 KB) 发送字节:76218 (76.2 KB)

图 2.3.16 if config 命令演示

在图 2.3.16 中有两个网卡: ens33 和 lo, ens33 是我的电脑实际使用的网卡, lo 是回测网 卡。可以看出网卡 ens33 的 IP 地址为 192.168.31.235, 我们使用命令 "ifconfig" 将网卡 ens33 的 IP 地址改为 192.168.31.20, 操作如下图所示:

zuozhongk	ai@ubuntu:~\$ sudo ifconfig ens33 192.168.31.20
zuozhongk	ai@ubuntu:~\$ ifconfig ens33
ens33	Link encap:以太网 硬件地址 00:0c:29:96:89:d6
	inet 地址:192.168.31.20 广播:192.168.31.255
	inet6 地址: fe80::e92a:d882:e1b:7402/64
	UP BROADCAST RUNNING MULTICAST MTU:1500 跃点数:1
	接收数据包:15188 错误:0 丢弃:0 过载:0 帧数:0
	发送数据包 :1040 错误 :0 丢弃 :0 过载 :0 载波 :0
	碰撞:0发送队列长度:1000
	接收字节:917930 (917.9 KB) 发送字节:84590 (84.5 KB)

图 2.3.17 修改网卡 IP 地址



原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php

从上图可以看出,我们在使用命令"ifconfig"修改网卡 ens33 的 IP 地址的时候使用了 "sudo",说明在 Ubuntu 下修改网卡 IP 地址是需要 root 用户权限的。当修改完以后使用命令 "ifconfig ens33"再次查看网卡 ens33 的命令,发现网卡 ens33 的 IP 地址变成了 192.168.31.20。

12、系统帮助命令 man

Ubuntu 系统中有很多命令,这些命令都有不同的格式,不同的格式对应不同的功能,要 完全记住这些命令和格式几乎是不可能的,必须有一个帮助手册,当我们需要了解一个命令 的详细信息的时候查阅这个帮助手册就行了。Ubuntu 提供了一个命令来帮助用户完成这个功 能,那就是"man"命令,通过"man"命令可以查看其它命令的语法格式、主要功能、主要 参数说明等,"man"命令格式如下:

man [命令名]

比如我们要查看命令"ifconfig"的说明,输入"man ifconfig"即可,如下图所示:

zuozhongkai@ubuntu:~\$ man ifconfig

图 2.3.18 man 命令演示

在终端中输入上图所示的命令,然后点击回车键就会打开"ifconfig"这个命令的详细说明,如下图所示:



图 2.3.19 命令 "ifconfig" 详细介绍信息

上图就是命令"ifconfig"的详细介绍信息,按"q"键退出到终端。

13、系统重启命令 reboot

通过点击 Ubuntu 主界面右上角的齿轮按钮来选择关机或者重启系统,同样的我们也可以 使用 Shell 命令 "reboot" 来重启系统,直接输入命令 "reboot" 然后点击回车键接口,如下图 所示:

zuozhongkai@ubuntu:~\$ reboot

图 2.3.20 reboot 命令演示



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

14、系统关闭命令 poweroff

使用命令"reboot"可以重启系统,使用命令"poweroff"就可以关闭系统,在终端中输 入命令 "poweroff" 然后按下回车键即可关闭 Ubuntu 系统,如下图所示:

```
zuozhongkai@ubuntu:~$ poweroff
```

图 2.3.21 poweroff 命令演示

15、软件安装命令 install

截至目前,我们都没有讲过 Ubuntu 下如何安装软件,因为 Ubuntu 安装软件不像 Windows 下那样,直接双击.exe 文件就开始安装了。Ubuntu 下很多软件是需要先自行下载源码,下载 源码以后自行编译,编译完成以后使用命令"intsall"来安装。当然 Ubuntu 下也有其它的软 件安装方法,但是用的最多的就是自行编译源码然后安装,尤其是嵌入式 Linux 开发。命令

"install"格式如下:

	install [选项]	[-T]	源文件	目标文件
或:	install [选项]		源文件	目录
或:	install [选项]	-t	目录	源文件
或:	install [选项]	-d	目录	

"install"命令是将文件(通常是编译后的文件)复制到目的位置,在前三种形式中,将源 文件复制到目标文件或将多个源文件复制到一个已存在的目录中同时设置其所有权和权限模 式。在第四种形式会创建指定的目录。命令"install"通常和命令"apt-get"组合在一起使用 的,关于"apt-get"命令我们稍后会讲解。

以上就是 Shell 最基本一些命令,还有一些其它的命令我们在后面在讲解,循序渐进嘛。

2.4 APT 下载工具

对于长时间使用 Windows 的我们,下载安装软件非常容易,Windows 下有很多的下载软 件, Ubuntu 同样有不少的下载软件,本节我们讲解 Ubuntu 下我们用的最多的下载工具: APT 下载工具,APT 下载工具可以实现软件自动下载、配置、安装二进制或者源码的功能。APT 下载工具和我们前面讲解的"install"命令结合在一起构成了 Ubuntu 下最常用的下载和安装 软件方法。它解决了 Linux 平台下一安装软件的一个缺陷,即软件之间相互依赖。

APT 采用的 C/S 模式,也就是客户端/服务器模式,我们的 PC 机作为客户端,当需要下 载软件的时候就向服务器请求,因此我们需要知道服务器的地址,也叫做安装源或者更新源。

打开"软件和更新"设置,打开方式如下图所示:



原子哥在线教学:w	ww.yuanzi	ge.com	论	云:www.	openedv.	com/foru	m.php	
			星	期三 01:42		I	‡ - _?	●) () -
۵.		Q 输入以	!搜索					
9								
	尽 待办…	<mark>☆☆</mark> 电源	<mark>89996</mark> 对对碰	Ţ	计算器	茄子		
0	/ 启动	一 启动…	<mark>27</mark> 日历	会 _{软件}	软件	黨 扫雷		
	_ /	-			3 点击	"软件与更新'		
	扫描易	设置	视频	输入法	数独	文 本…		0
?	一 文件	- 小 系统	(语言…	>_ 终端	Aisl	a, Ama		
a								
	祖 养		常用	全部	2 选择 '	'全部"		
要将输入定向到该虚拟机,请将	鼠标指针移入其中或	波 Ctrl+G。) 🔓 🖶 🕼	

图 2.4.1 打开软件和更新设置

打开以后的界面如下图所示:

软件和更新 💿 🗈 😣									
Ubuntu 软件	其它软件	更新	身份验证	附加驱动	开发者选项				
可从互联网下载									
🗹 Canonio	☑ Canonical 支持的免费和开源软件 (main)								
☑ 社区维持	护的免费和	开源软	件 (universe	e)					
☑ 设备的	专有驱动 (re	estricte	ed)						
☑ 有版权利	和合法性问题	题的的?	软件 (multiv	verse)					
□ 源代码									
下载自:	中国的服务	务器			•				
可从光驱安装	ŧ								
Ubuntu 18.04 'Bionic Beaver' 的 CD 光盘 □ 官方支持 版权受限									
				还原(V)	关闭(C)				

图 2.4.2 软件和更新设置



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

在上图中的"Ubuntu 软件"选项卡下面的"下载自"就是 APT 工具的安装源,因为我们 是在中国,所以需要选择中国的服务器,否则的话可能会导致下载失败!这个也就是网上说 的 Ubuntu 安装成功以后要更新源。

在我们使用 APT 工具下载安装或者更新软件的时候,首先会在下载列表中与本机软件对比,看一下需要下载哪些软件,或者升级哪些软件,默认情况下 APT 会下载最新的软件包,被安装的软件包所依赖的其它软件也会被下载安装。说了这么多,APT 下载工具究竟怎么用呢? APT 工具常用的命令如下:

1、更新本地数据库

如果想查看本地哪些软件可以更新的话可以使用如下命令:

sudo apt-get update

这个命令会访问源地址,并且获取软件列表并保存在本电脑上,过程如下图所示:

zuozhongkai@ubuntu:~\$ sudo apt-get update
[sudo] zuozhongkai 的密码:
 命中:1 http://security.ubuntu.com/ubuntu xenial-security InRelease
 命中:2 http://cn.archive.ubuntu.com/ubuntu xenial InRelease
 获取:3 http://cn.archive.ubuntu.com/ubuntu xenial-updates InRelease [109 kB]
 获取:4 http://cn.archive.ubuntu.com/ubuntu xenial-backports InRelease [107 kB]
 已下载 216 kB, 耗时 13秒 (16.5 kB/s)
 正在读取软件包列表...完成

图 2.4.3 更新本地数据库

2、检查依赖关系

有时候本地某些软件可能存在依赖关系,所谓依赖关系就是 A 软件依赖于 B 软件。通过如下命令可以查看依赖关系,如果存在依赖关系的话 APT 会提出解决方案:

sudo apt-get check

上述命令的执行结果如下图所示:

zuozhongkai@ubuntu:~\$ sudo apt-get check 正在读取软件包列表... 完成 正在分析软件包的依赖关系树 正在读取状态信息... 完成

图 2.4.4 检查依赖关系

3、软件安装

这个是重点了,安装软件,使用如下命令:

sudo apt-get install package-name

可以看出上述命令是由"apt-get"和"install"组合在一起的,"package-name"就是要 安装的软件名字,"apt-get"负责下载软件,"install"负责安装软件。比如我们要安装软件 Ubuntu 下的串口工具"minicom",我们就可以使用如下命令:

sudo apt-get install minicom

执行上述命令以后就会自动下载和安装 minicom 软件,如下图所示:


原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

zuozhongkai@ubuntu:~\$ sudo apt-get install minicom 正在读取软件包列表... 完成 正在分析软件包的依赖关系树 正在读取状态信息... 完成 将会同时安装下列软件: lrzsz 下列【新】软件包将被安装: lrzsz minicom 升级了 0 个软件包,新安装了 2 个软件包,要卸载 0 个软件包,有 92 个软件包未被升 级。 需要下载 306 kB 的归档。 服 反 缩 后 会 消 耗 1,193 kB 的 额 外 空 间 。 您 希 望 继 续 执 行 吗 ? [Y/n] y 获取:1 http://cn.archive.ubuntu.com/ubuntu xenial/universe amd64 lrzsz amd64 0.1 2.21-8 [73.8 kB] 获取:2 http://cn.archive.ubuntu.com/ubuntu xenial-updates/universe amd64 minicom amd64 2.7-1+deb8u1build0.16.04.1 [232 kB] 已下载 306 kB,耗时 3秒 (80.8 kB/s) 正在选中未选择的软件包 lrzsz。 (正在读取数据库 ... 系统当前共安装有 217399 个文件和目录。) 正准备解包 .../lrzsz_0.12.21-8_amd64.deb 正在解包 lrzsz (0.12.21-8) ... 正在选中未选择的软件包 minicom。 正准备解包 .../minicom 2.7-1+deb8u1build0.16.04.1 amd64.deb ... 正在解包 minicom (2.7-1+deb8u1build0.16.04.1) ... 正在处理用于 man-db (2.7.5-1) 的触发器 正在设置 lrzsz (0.12.21-8) 正在设置 minicom (2.7-1+deb8u1build0.16.04.1) ...

图 2.4.5 安装 minicom 软件

上图就是安装 minicom 这个软件的过程,在安装的过程中,会有如下所示询问:

您希望继续执行吗? [Y/n]

如果希望继续执行的话就输入 y,如果不希望继续执行的话就输入 n。安装完成以后我们 直接在终端输入如下命令打开 minicom 这个串口软件:

minicom -s

打开以后的 minicom 软件如下图所示:

+[configuration]	L
Filenames and paths	
File transfer protocols	
Serial port setup	
Modem and dialing	
Screen and keyboard	
Save setup as dfl	
Save setup as	
Exit	
Exit from Minicom	

图 2.4.6 minicom 软件

关于 minicom 的使用大家可以上网搜索一下,这里就不详细讲解了,要退出 minicom 可 以直接按下 ESC 键。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

注: 推荐使用 apt 替代 apt-get。

4、软件更新

有时候我们需要更新软件,更新软件的话使用命令:

sudo apt-get upgrade package-name

其中 package-name 为要升级的软件名字,比如我们升级刚刚安装的 minicom 这个软件, 如下图所示:

<pre>zuozhongkai@ubuntu:~\$ sudo apt-get upgrade minicom</pre>
正在读取软件包列表完成
正在分析软件包的依赖关系树
正在读取状态信息完成
minicom 已经是最新版 (2.7-1+deb8u1build0.16.04.1)。
正在计算更新完成
下列软件包的版本将保持不变:
ubuntu-minimal

图 2.4.7 更新 minicom 软件

从上图可以看出, minicom 已经是最新的了, 不用更新, 不过有其它软件需要更新, 因 此会自动更新其它的软件。

5、卸载软件

如果要卸载某个软件的话使用如下命令:

sudo apt-get remove package-name

其中 package-name 是要卸载的软件,比如卸载前面安装的 minicom 这个软件,操作如下 图所示:



图 2.4.8 卸载软件

从上图中可以看出软件 minicom 被卸载掉了。关于 APT 下载工具就讲解到这里,我们用 的最多的就是"sudo apt-get install package-name"来下载和安装软件。有关 Ubuntu 其它的安 装软件的方法打开可以自行上网查阅学习,这里就不一一详解了。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

2.5 Ubuntu 下文本编辑

2.5.1 Gedit 编辑器

进行文本编辑是最常用的操作, Windows 下我们会使用记事本来完成, 或者其它一些优 秀的文本编辑器,比如 notepad++, Ubuut 下有一个自带的文本编辑器,那就是 Gedit。Gedit 是一个窗口式的编辑器,关于 Gedit 的使用前面我们已经讲解了。本节我们重点讲解的是另外 一个编辑器: VI/VIM 编辑器。

2.5.2 VI/VIM 编辑器

我们如果要在终端模式下进行文本编辑或者修改文件就可以使用 VI/VIM 编辑器, Ubuntu 自带了 VI 编辑器,但是 VI 编辑器对于习惯了 Windows 下进行开发的人来说不方便,比如竟 然不能使用键盘上的上下左右键调整光标位置。因此我推荐大家使用 VIM 编辑器, VIM 编辑 器是 VI 编辑器升级版本, VI/VIM 编辑器都是一种基于指令式的编辑器, 不需要鼠标, 也没 有菜单,仅仅使用键盘来完成所有的编辑工作。

我们需要先安装 VIM 编辑器, 命令如下:

sudo apt-get install vim

安装完成以后就可以使用 VIM 编辑器了, VIM 编辑器有 3 中工作模式: 输入模式、指令 模式和底行模式,通过切换不同的模式可以完成不同的功能,我们就以编辑一个文本文档为 例讲解 VIM 编辑器的使用。打开终端,输入命令: vi test.txt,如下图所示:

zuozhongkai@ubuntu:~\$ vim test.txt

图 2.5.1 新建 test.txt 文档

在终端中输入上图中所示的命令以后就会创建一个 test.txt 文档,并且用 VIM 打开了,如 下图所示:



图 2.5.2VIM 打开的 test.txt 文档



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

我们试着在上图中输入数字,发现根本没法输入,这不是因为你的键盘坏了。因为 VIM 默认是以只读模式打开的文档,因此我们要切换到输入模式,切换到输入模式的命令如下:

- i 在当前光标所在字符的前面,转为输入模式。
- I 在当前光标所在行的行首转换为输入模式。
- a 在当前光标所在字符的后面,转为输入模式。
- A 在光标所在行的行尾,转换为输入模式。
- o 在当前光标所在行的下方,新建一行,并转为输入模式。
- O 在当前光标所在行的上方,新建一行,并转为输入模式。
- s 删除光标所在字符。
- r 替换光标处字符。

最常用的就是"a",我们在图 2.5.2 中按下键盘上的"a"键,这时候终端左下角会提示"插入"字样,表示我们进入到了输入模式,如下图所示:



图 2.5.3 切换到插入模式

上图表明我们可以正常输入文本了,我们可以输入图 2.5.4 所示文本:

原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php

正点原子



图 2.5.4 输入文本

在上图中我们在 test.txt 中输入了字母、数字和中文,我们输入完成以后需要保存文本啊, Windows下的记事本可以使用快捷键 Ctrl+S 来保存,VIM 是否也可以使用 Ctrl+S 来保存呢? 你会发现当你按下 Ctrl+S 键以后你的终端不能操作了!!!这是因为在 Ubuntu 下 Ctrl+S 快捷 键不是用来完成保存的功能的,而是暂停该终端!所以你一旦在使用终端的时候按下 Ctrl+S 快捷键,那么你的终端肯定不会再有任何反应,如果你按下 Ctrl+S 关闭了当前终端的话可以 按下 Ctrl+Q 来重新打开终端。

既然 Ctrl+S 不能保存文本文档,那么有没有其它方法保存文本文档呢?肯定是有的,我 们需要从 VIM 现在的输入模式切换到指令模式,方式就是按下键盘的 ESC 键,按下 ESC 键 以后终端坐下角的"插入"字样就会消失,此时你就不能在输入任何文本了,如果想再次输 入文本的话就按下"a"键重新进入到输入模式。指令模式顾名思义就是输入指令的模式,这 些指令是控制文本的指令,我们将这些指令进行分类,如下所示:

1、移动光标指令:

h(或左方向键) 光标左移一个字符。 l(或右方向键) 光标右移一个字符。 j(或下方向键) 光标下移一行。 k(或上方向键) 光标上移一行。 nG 光标移动到第n行首。 n+ 光标下移n行。 n- 光标上移n行。

2、屏幕翻滚指令

Ctrl+f	屏幕向下翻一页,	相当于下一页。
Ctrl+b	屏幕向上翻一页,	相当于上一页。

3、复制、删除和粘贴指令

cc 删除整行,并且修改整行内容。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

- dd 删除改行,不提供修改功能。
- ndd 删除当前行向下 n 行。
- x 删除光标所在的字符。
- **X** 删除光标前面的一个字符。
- nyy 复制当前行及其下面 n 行。
- **p** 粘贴最近复制的内容。

上面就是 VI/VIM 的命令模式下最常用的一些命令,还有一些不常用的我没有列出来,感兴趣的可以自行上网查阅。从上面的命令可以看出,并没有保存文本的命令,那是因为保存 文档的命令是在底行模式中,我们要先进入到指令模式,进入底行模式的方式是先进入指令 模式下,然后在指令模式下输入":"进入底行模式,如下图所示:

😣 🖨 💷 zuozhongkai@ubuntu: ~
zuozhongkai
123456789°
ARM裸机与嵌入式Linux驱动开发。
~
~
~
~
~
~
~
~
~
~

图 2.5.5":"底行模式

在上图中当进入底行模式以后会在终端的左下角就会出现符号":",我们可以在":" 后面输入命令,常用的命令如下:

- **x** 保存当前文档并且退出。
- **q** 退出。
- w 保存文档。
- q! 退出 VI/VIM,不保存文档。

如果我们要退出并保存文本的话需要在":"底行模式下输入"wq",如下图所示:





图 2.5.6 保存并退出 VIM

在":"底行模式下输入"wq"以后按下回车键就保存 test.txt 并退出 VI/VIM 编辑器, 退出以后我们可以使用命令"cat"来查看刚刚新建的 test.txt 文档的内容, 如下图所示:



图 2.5.7 查看文档内容

从上图中可以看出,test.txt 中的内容就是我们用 VIM 输入的内容,至此我们就完整的进行了一遍 VI/VIM 创建文档、编辑文档和保存文档。

在上面讲解进入 VIM 的底行模式的时候使用了在指令模式下输入":"的方法,还可以 在指令模式下输入"/"进入底行模式,输入"/"以后如下图所示。



图 2.5.8"/"底行模式



原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php

在"/"底行模式下我们可以在文本中搜索指定的内容,比如搜索 test.txt 文件中"嵌入式" 三个字,使用方法如下图所示:



图 2.5.9 搜索文本

在"/"后面输入要搜索的内容,然后按下回车键就会在 test.txt 中找到与字符串"嵌入式" 匹配的部分,如下图所示:

😣 🖨 💷 zuozhongkai@ubuntu: ~		
zuozhongkai		
12345678 <mark>9</mark>		
ARM裸机与嵌入式Linux驱动开发。		
~		
~		
~		
已查找到文件再从开头继续查找	3,13-10	全部

图 2.5.10 查找到指定内容

从上图中可以看出,在 test.txt 中找到了"嵌入式"这个词,并且标记出来位置。我们以 后要在一个文档中搜素是否存在某个字符串的时候就可以使用这种方法。有关 VI/VIM 编辑器 的讲解就到这里,我们完整的练习了一遍如何使用 VIM 创建文档、编辑文档、保存文档和在 文档中搜索字符串。有关更多更详细的 VIM 编辑器的操作大家自行上网查阅相关文档和博客。

2.6 Linux 文件系统

操作系统的基本功能之一就是文件管理, 而文件的管理是由文件系统来完成的。Linux 支 持多种文件系统,本节我们就来讲解 Linux 下的文件系统、文件系统类型、文件系统结构和 文件系统相关 Shell 命令。

2.6.1 Linux 文件系统简介以及类型

1、Linux 文件系统简介

操作系统就是处理各种数据的,这些数据在硬盘上就是二进制,人类肯定不能直接看懂 这些二进制数据,要有一个翻译器,将这些二进制的数据还原为人类能看懂的文件形式,这 个工作就是由文件系统来完成的,文件系统的目的就是实现数据的查询和存储,由于使用场 合、使用环境的不同, Linux 有多种文件系统, 不同的文件系统支持不同的体系。文件系统是



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

管理数据的,而可以存储数据的物理设备有硬盘、U盘、SD 卡、NAND FLASH、NOR FLASH、网络存储设备等。不同的存储设备其物理结构不同,不同的物理结构就需要不同的 文件系统去管理,比如管理 NAND FLASH 的话使用 YAFFS 文件系统,管理硬盘、SD 卡的话 就是 ext 文件系统等等。

我们在使用 Windows 的时候新买一个硬盘回来一般肯定是将这个硬盘分为好几个盘,比如 C 盘、D 盘等。这个叫磁盘的分割,Linux 下也支持磁盘分割,Linux 下常用的磁盘分割工 具为: fdisk, fdisk 这个工具我们后面会详细讲解怎么用,因为我们移植 Linux 的时候需要将 SD卡分为两个分区来存储不同的东西。在 Windows 下我们创建一个新的盘符以后都要做格式 化处理,格式化其实就是给这个盘符创建文件系统的过程,我们在 Windows 格式化某个盘的 时候都会让你选择文件系统,如下图所示:

格式化 ESD-USB (J:)	×
容量(P):	
28.8 GB	\sim
文件系统(F)	
FAT32 (默认)	\sim
分配单元大小(A)	
16 КВ	\sim
还原设备的默认值(D)	

图 2.6.1 格式化磁盘

上图就是格式化磁盘的时候选择的文件系统,Windows下一般有 FAT、NTFS 和 exFAT 这些文件系统。同样的,在 Linux 下我们使用 fdisk 创建好分区以后也是要先在创建好的分区 上面创建文件系统,也就是格式化。

在 Windows 下有磁盘分区的概念,比如 C, D, E 盘等,在 Linux 下没有这个概念,因此 Linux 下你找不到像 C、D、E 盘这样的东西。前面我们说了 Linux 下可以给磁盘分割,但是 没有 C、D、E 盘那怎么访问这些分区呢?在 Linux 下创建一个分区并且格式化好以后我们要 将其"挂载"到一个目录下才能访问这个分区。Windows 的文件系统挂载过程是其内部完成 的,用户是看不到的,Linux 下我们使用 mount 命令来挂载磁盘。挂载磁盘的时候是需要确定 挂载点的,也就是你的这个磁盘要挂载到哪个目录下。

2、Linux 文件系统类型

前面我们说了,在Windows下有 FAT、NTFS 和 exFAT 这样的文件系统,在 Linux下又 有哪些文件系统呢,Linux下的文件系统主要有 ext2、ext3、ext4 等文件系统。Linux 还支持其 他的 UNIX 文件系统,比如 XFS、JFS、UFS 等,也支持 Windows 的 FAT 文件系统和网络文 件系统 NFS 等。这里我们主要讲一下 Linux 自带的 ext2、ext3 和 ext4 文件系统。

ext2 文件系统:

ext2 是 Linux 早期的文件系统,但是随着技术的发展 ext2 文件系统已经不推荐使用了, ext2 是一个非日志文件系统,大多数的 Linux 发行版都不支持 ext2 文件系统了。



ext3 文件系统:

ext3 是在 ext2 的基础上发展起来的文件系统,完全兼容 ext2 文件系统, ext3 是一个日志 文件系统, ext3 支持大文件, ext3 文件系统的特点有如下:

高可靠性: 使用 ext3 文件系统的话,即使系统非正常关机、发生死机等情况,恢复 ext3 文件系统也只需要数十秒。

数据完整性: ext3 提高了文件系统的完整性,避免意外死机或者关机对文件系统的伤害。 文件系统速度: ext3 的日志功能对磁盘驱动器读写头进行了优化,文件系统速度相对与 ext2 来说没有降低。

数据转换:从 ext2 转换到 ext3 非常容易,只需要两条指令就可以完成转换。用户不需要 花时间去备份、恢复、格式化分区等,用 ext3 文件系统提供的工具 tune2fs 即可轻松的将 ext2 文件系统转换为 ext3 日志文件系统。ext3 文件系统不需要经过任何修改,可以直接挂载成 ext2 文件系统。

ext4 文件系统:

ext4 文件系统是在 ext3 上发展起来的, ext4 相比与 ext3 提供了更佳的性能和可靠性,并 且功能更丰富, ext4 向下兼容 ext3 和 ext2,因此可以将 ext2 和 ext3 挂载为 ext4。那么我们安 装的 Ubuntu 使用的哪个版本的文件系统呢?在终端中输入如下命令来查询当前磁盘挂载的是 啥文件系统:

df -T -h

结果如下图所示:

zuozhongkai@ub	untu:/\$ df	-T -h			
文件系统	类型	容量	已用	可用	已用% 挂载点
udev	devtmpfs	3.9G	0	3.9G	0% /dev
tmpfs	tmpfs	796M	9.3M	787M	2% / run
/dev/sda1	ext4	192G	5.2G	177G	3% /
tmpfs	tmpfs	3.9G	252K	3.9G	1% /dev/shm
tmpfs	tmpfs	5.0M	4.0K	5.0M	1% /run/lock
tmpfs	tmpfs	3.9G	0	3.9G	0% /sys/fs/cgroup
tmpfs	tmpfs _	796M	56K	796M	1% /run/user/1000

图 2.6.2Ubuntu 使用的文件系统

在图 2.6.2 中, 框起来的就是我们安装 Ubuntu 的这个磁盘, 在 Linux 下一切皆为文件, "/dev/sda1"就是我们的磁盘分区,可以看出这个磁盘分区类型是 ext4, 它的挂载点是"/", 也就是根目录。

2.6.2 Linux 文件系统结构

在 Windows 下直接打开 C 盘,我们进入的就是 C 盘的根目录,打开 D 盘进入的就是 D 盘的根目录,比如 C 盘根目录如下:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

本地磁盘 (C:)			
名称	修改日期	类型	大小
\$WINDOWS.~BT	2017-01-18 10:25	文件夹	
\$Windows.~WS	2017-12-17 11:34	文件夹	
📕 AppData	2018-08-27 14:47	文件夹	
DRMsoft	2017-09-07 11:24	文件夹	
ESD	2017-12-17 12:11	文件夹	
📕 Intel	2016-09-07 15:13	文件夹	
M1530_MFP_Series_Basic_Solution	2016-12-29 23:30	文件夹	
📜 PerfLogs	2016-07-16 19:47	文件夹	
📙 Program Files	2017-12-13 15:08	文件夹	
📙 Program Files (x86)	2018-12-18 0:04	文件夹	
📕 ProgramData	2018-12-17 21:57	文件夹	
📜 touchgfx-env	2017-12-05 11:37	文件夹	
TouchGFXProjects	2018-01-07 11:22	文件夹	
📜 uacdump	2016-11-28 12:39	文件夹	
Users	2016-09-08 13:00	文件夹	
Windows	2018-10-31 23:49	文件夹	
🔬 InstallConfig.ini	2017-10-13 23:12	配置设置	1 KB

图 2.6.3C 盘根目录

在 Linux 下因为没有 C、D 盘之说,因此 Linux 只有一个根目录,没有 C 盘根目录、D 盘 根目录之类的。其实如果你的 Windows 只有一个 C 盘的话那么整个系统也就只有一个根目录。 Windows下的C盘根目录就是"C:",在Linux下的根目录就是"/",你没有看错,Linux 根目录就是用"/"来表示的,打开 Ubuntu 的文件浏览器,文件浏览器在左侧的导航栏,图标 如下图所示:



图 2.6.4 文件浏览器

打开以后的文件浏览器如图 2.6.5 所示:

原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php

正点原子

<	→ 《 企主	文件夹			٩	= =	•••
Ø	最近使用						
ŵ	主目录	vmware-			加中市		
	桌面	tools-	公共的	假加	个光少贝	图 万	
-	视频		_				
ø	图片	State	-	53	-		
۵	文档	文档	下载	音乐	桌面	示例	
⇒	下载						
13	音乐						
	回收站						
+	其他位置						

图 2.6.5 文件浏览器

直接打开文件浏览器以后,我们默认不是处于根目录中的,不像 Windows,我们直接打 开 C 盘就处于 C 盘根目录下。Ubuntu 是支持多用户的, Ubuntu 为每个用户创建了一个根目 录,比如我电脑现在登陆的是"zuozhongkai"这个用户,因此默认进入的是"zuozhongkai" 这个用户的根目录。我们点击图 2.6.5 中左侧的"其他位置",打开以后如图 2.6.6 所示:

<	> ◀ 其他(立置 ▶		Q	= ≡	
Ø	最近使用	位于本机				
ŵ	主目录	🖾 计算机	43.2	GB/52.0	6 GB 可用	/
	桌面	网络 2 点击"计算机	"进入根目录		根目影	₹ 3
H	视频	一 Windows 网络				
٥	图片	• Windows Might				
D	文档					
÷	下载					
	音乐					
▣	回收站					
	其他位置					
	1					
		连接到服务器(S)	输入服务器地址		• •	连接(N)

图 2.6.6 根目录"/"



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

图 2.6.6 就是 Ubuntu 的根目录 "/",这时候肯定就有人有疑问,刚刚说 Ubuntu 会给每个 用户创建一个根目录,那这些用户的根目录在哪里?是不是和根目录 "/"是一个地位的?其 实所谓的给每个用户创建一个根目录只是方便说而已,这个所谓的用户根目录其实就是"/" 下的一个文件夹,以我的"zuozhongkai"这个用户为例,其用户根目录就是: /home/zuozhongkai。只要你创建了一个用户,那么系统就会在/home 这个目录下创建一个以这 个用户名命名的文件夹,这个文件夹就是这个用户的根目录。

用户可以对自己的用户根目录下的文件进行随意的读写操作,但是如果要修改根目录"/"下的文件就会提示没有权限。打开终端以后默认进入的是当前用户根目录,比如我们打开终端以后输入"ls"命令查看当前目录下有什么文件,结果如图 2.6.7 所示:

zuozhongkai@ubunt	u:~\$ ls				
examples.desktop	tmp	模板	图片	下载	桌面
test.txt	公共的	视频	文档	音乐	

图 2.6.7 目录查看

可以看出图 2.6.7 中的文件和图 2.6.5 中的一模一样,都是"zuozhongkai"这个账户的根目录。我们来看一下根目录"/"下都有哪些文件,在终端中输入如下命令:

- cd / //进入到根目录"/
- ls //查看根目录"/"下的文件以及文件夹
 - 执行上述两行命令以后,终端如所示:

zuozhongkai@ubuntu:~\$ cd / zuozhongkai@ubuntu:/\$ ls									
bin boot cdrom	dev etc home	initrd.img initrd.img.old lib	lib64 lost+found media	mnt opt proc	root run sbin	snap srv sys	tmp usr var	vmlinuz vmlinuz.old	

图 2.6.8 查看根目录"/"

图 2.6.8 中列举出了根目录"/"下面的所有文件夹,这里我们仔细观察一下,当我们进入 到根目录"/"里面以后终端提示符"\$"前面的符号"~"变成了"/",这是因为当我们在终 端中切换了目录以后"\$"前面就会显示切换以后的目录路径。我们来看一下根目录"/"中 的一些重要的文件夹:

/bin 存储一些二进制可执行命令文件,/usr/bin 也存放了一些基于用户的命令文件。

/sbin 存储了很多系统命令,/usr/sbin 也存储了许多系统命令。

/root 超级用户 root 的根目录文件。

/home 普通用户默认目录,在该目录下,每个用户都有一个以本用户名命名的文件夹。 **/boot** 存放 Ubuntu 系统内核和系统启动文件。

/mnt 通常包括系统引导后被挂载的文件系统的挂载点。

/dev 存放设备文件,我们后面学习 Linux 驱动主要是跟这个文件夹打交道的。

/etc 保存系统管理所需的配置文件和目录。

/lib 保存系统程序运行所需的库文件,/usr/lib下存放了一些用于普通用户的库文件。

/lost+found 一般为空,当系统非正常关机以后,此文件夹会保存一些零散文件。

/var 存储一些不断变化的文件,比如日志文件

/usr 包括与系统用户直接有关的文件和目录,比如应用程序和所需的库文件。

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

/media 存放 Ubuntu 系统自动挂载的设备文件。 /proc 虚拟目录,不实际存储在磁盘上,通常用来保存系统信息和进程信息。

正点原子

/tmp 存储系统和用户的临时文件,该文件夹对所有的用户都提供读写权限。

/opt 可选文件和程序的存放目录。

/sys 系统设备和文件层次结构,并向用户程序提供详细的内核数据信息。

2.6.3 文件操作命令

本节我们来学习一下在终端进行文件操作的一些常用命令:

1、创建新文件命令一touch

在前面学习 VIM 的时候我们知道可以用 vi 指令来创建一个文本文档,本节我们就学习一个功能更全面的文件创建命令一touch。touch 不仅仅可以用用来创建文本文档,其它类型的文档也可以创建,命令格式如下:

touch [参数] [文件名]

使用 touch 创建文件的时候,如果[文件名]的文件不存在,那就直接创建一个以[文件名] 命名的文件,如果[文件名]文件存在的话就仅仅修改一下此文件的最后修改日期,常用的命 令参数如下:

-a 只更改存取时间。

-c 不建立任何文件。

-d<日期> 使用指定的日期,而并非现在日期。

-t<时间> 使用指定的时间,而并非现在时间。

进入到用户根目录下,直接使用命令 "cd~"即可快速进入用户根目录,进入用户根目 录以后使用 touch 命令创建一个名为 test 的文件,创建过程如图 2.6.9 所示:

zuozhongkai@ubuntu:~\$ cd ~ //进入用户根目录
zuozhongkai@ubuntu:~\$ ls //查看当前目录下的文件
examples.desktop tmp 模板 图片 下载 桌面
test.txt 公共的 视频 文档 音乐
zuozhongkai@ubuntu:~\$ touch test //创建test文件
zuozhongkai@ubuntu:~\$ ls test //查看创建的test文件
test
zuozhongkai@ubuntu:~\$ ls test -l //查看创建的test文件详细信息
-rw-rw-r 1 zuozhongkai zuozhongkai 0 12月 22 01:24 test

图 2.6.9touch 命令操作

2、文件夹创建命令一mkdir

既然可以创建文件,那么肯定也可以创建文件夹,创建文件夹使用命令"mkdir",命令格式如下:

mkdir [参数] [文件夹名目录名]

主要参数如下:

-p 如所要创建的目录其上层目录目前还未创建,那么会一起创建上层目录。

我们在用户根目录下创建两个分别名为"testdir1"和"testdir2"的文件夹,操作如图 2.6.10 所示:



于贵	任线教学:www.y	uanzige.co	m 14	达:www	w.open	edv.co	om/forum.ph	p
	zuozhongkai@ubunt	u:~\$ ls //	查看当前目录	下的文件				
	examples.desktop	test.txt	公共的 礼	见频 文材	当 音乐	÷		
	test	tmp	模板 图	图片 下载	哉 桌面	ī		
	zuozhongkai@ubunt	u:~\$ mkdir	testdir1	//创建testo	dir1文件到	E		
	zuozhongkai@ubunt	u:~\$ mkdir	testdir2	//创建testc	lir2文件到	ξ		
	zuozhongkai@ubunt	u:~\$ ls		//查看当前日	目录下的所	府文件	,看看文件夹创建是	否成功
	examples.desktop	testdir1	test.txt	公共的	视频	文档	音乐	
	test	testdir2	tmp	模板	图片	下载	桌面	

图 2.6.10 创建文件夹

在图 2.6.10 中,我们使用命令"mkdir"创建了"testdir1"和"testdir2"这两个文件夹。

3、文件及目录删除命令一rm

既然有创建文件的命令,那肯定有删除文件的命令,要删除一个文件或者文件夹可以使 用命令"m",此命令可以完成删除一个文件或者多个文件及文件夹,它可以实现递归删除。 对于链接文件,只删除链接,原文件保持不变,所谓的链接文件,其实就是 Windows 下的快 捷方式文件,此命令格式如下:

rm [参数] [目的文件或文件夹目录名]

命令主要参数如下:

-d 直接把要删除的目录的硬连接数据删成 0, 删除该目录。

- -f 强制删除文件和文件夹(目录)。
- -i 删除文件或者文件夹(目录)之前先询问用户。
- -r 递归删除,指定文件夹(目录)下的所有文件和子文件夹全部删除掉。
- -v 显示删除过程。

我们使用 rm 命令来删除前面使用命令"touch"创建的 test 文件,操作过程如图 2.6.11 所

示:

zuozhongkai@ubunt	u:~\$ ls //ª	看当前日录	下的所有	ī文件			
examples.desktop	testdir1	test.txt	: 公共	的 礼	视频	文档	音乐
test	testdir2	tmp	模板	i B	图片	下载	桌面
zuozhongkai@ubunt	u:~\$ rm te	st //删除文	(件test				
zuozhongkai@ubunt	u:~ \$ ls	//查看当	前日录,	看看te	st文件题	是否删除	
examples.desktop	testdir2	tmp	模板	图片	下载	桌面	
testdir1	test.txt	公共的	视频	文档	音乐		

图 2.6.11 删除文件

命令 "rm"也可以直接删除文件夹,我们可以试一下删除前面创建的 testdir1 文件夹,先 直接使用命令 "rm testdir1"测试一下是否可以删除,结果如图 2.6.12 所示:

zuozhongkai@ubuntu:~\$ rm testdir1 rm: 无法删除'testdir1': 是一个目录

图 2.6.12 删除文件夹

在图 2.6.12 中可以看出,直接使用命令"rm"是无法删除文件夹(目录)的,我们需要加上参数"-rf",也就是强制递归删除文件夹(目录),操作结果如图 2.6.13 所示:



图 2.6.13 带参数删除文件夹

从图 2.6.13 可以看出,当在命令"rm"中加入参数"-rf"以后就可以删除掉文件夹"testdir1"了。

4、文件夹(目录)删除命令一rmdir

上面我们讲解了如何使用命令"m"删除文件夹,那就是要加上参数"-rf",其实 Linux 提供了直接删除文件夹(目录)的命令一rmdir,它可以不加任何参数的删除掉指定的文件 夹(目录),命令格式如下:

rmdir [参数] [文件夹(目录)]

命令主要参数如下:

-p 删除指定的文件夹(目录)以后,若上层文件夹(目录)为空文件夹(目录)的话就将其一起删除。

我们使用命令"rmdir"删除掉前面创建的"testdir2"文件夹,操作过程如图 2.6.14 所示:

zuozhongkai@ubuntu	I:~\$ ls	//查看当前E	录下的所有	与文件	
examples.desktop	test.txt	公共的	视频	文档 音	音乐 二十二十二十二十二十二十二十二十二十二十二十二十二十二十二十二十二十二十二十
testdir2	tmp	模板	图片 '	下载 舅	夏面
zuozhongkai@ubuntu	ı:∼\$ rmdi	r testdir	2 //删除了	<mark>て件夹test</mark>	dir2
zuozhongkai@ubuntu	: ∼\$ ls		//查看删	除文件夹	以后的目录文件
examples.desktop	tmp	模板 图片	ト 下 载	桌面	
test.txt	公共的	视频 文権	皆 音乐		

图 2.6.14 命令 rmdir 删除文件夹

5、文件复制命令一cp

在 Windows 下我们可以通过在文件上点击鼠标右键来进行文件的复制和粘贴,在 Ubuntu 下我们也可以通过点击文件右键进行文件的复制和粘贴。但是本节我们来讲解如何在终端下 使用命令来进行文件的复制,Linux 下的复制命令为"cp",命令描述如下:

cp [参数] [源地址] [目的地址]

主要参数描述如下:

- -a 此参数和同时指定"-dpR"参数相同
- -d 在复制有符号连接的文件时,保留原始的连接。
- -f 强行复制文件,不管要复制的文件是否已经存在于目标目录。
- -I 覆盖现有文件之前询问用户。
- -p 保留源文件或者目录的属性。

-r或-R递归处理,将指定目录下的文件及子目录一并处理

我们在用户根目录下,使用前面讲解的命令"mkdir"创建两个文件夹: test1和 test2,过程如图 2.6.15 所示。



图 2.6.15 创建 test1 和 test2 两个文件夹

进入上面创建的 test1 文件夹, 然后在 test1 文件夹里面创建一个 a.c 文件, 操作过程如图 2.6.16 所示:

zuozhongkai@ubuntu:~\$ cd test1 //进入test1文件夹 zuozhongkai@ubuntu:~/test1\$ touch a.c//创建a.c文件 zuozhongkai@ubuntu:~/test1\$ ls //查看当前目录下所有文件 a.c

图 2.6.16 创建 a.c 文件

我们先将图 2.6.16 中的 a.c 这个文件做个备份,也就是复制到同文件夹 test1 里面,新的文件命名为 b.c。然后在将 test1 文件夹中的 a.c 和 b.c 这两个文件都复制到文件夹 test2 中,操作如图 2.6.17 所示

zuozhongkai@ubuntu ~/test1\$	cp	a.c b.c	//拷贝a.c到本文件夹中,并重命名为b.c
zuozhongkai@ubuntu ~/test1\$	ls		//查看当前目录下的所有文件
a.c b.c zuozhongkai@ubuntu:~/test1\$ zuozhongkai@ubuntu:~/test1\$ zuozhongkai@ubuntu:~/test2\$ a.c b.c	cp cd ls	*.c/test2 /test2	2 //拷贝a.c和b.c到文件夹test2中 //进入上级目录中的test2文件夹 //查看test2文件夹下的文件

图 2.6.17 拷贝文件

在图 2.6.17 中,我们添加了一些高级使用技巧,首先是拷贝 a.c 和 b.c 文件到 test2 文件夹 中,我们使用了通配符 "*", "*.c"就表示 test1 下的所有以 ".c"结尾的文件,也就是 a.c 和 b.c。"../test2"中的 "../"表示上级目录,因此 "../test2"就是上级目录下的 test2 文件夹。 上面都是文件复制,我们接下来学习一下文件夹复制,我们将 test2 文件夹复制到同目录

下,新拷贝的文件夹命名为 test3,操作如图 2.6.18 所示:

zuozhongkai@ubunt	u:~\$ cp	-rf test2	/ test3/	//复制	jtest2文	件夹为test3文件夹
zuozhongkai@ubunt	u:~\$ ls			//查看	当前日录	录下的所有文件
examples.desktop	test2	test.txt	公共的	视频	文档	音乐
test1	test3	tmp	模板	图片	下载	桌面
zuozhongkai@ubunt	u:~\$ cd	test3		//进入	test3文(件夹
zuozhongkai@ubunt	u:~/tesi	t 3 \$ ls		//杏毛	tost3cbd	的形有文件
a.c b.c		_		//里相	(COLO-TI	

图 2.6.18 拷贝文件夹

6、文件移动命令一mv



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

有时候我们需要将一个文件或者文件夹移动到另外一个地方去,或者给一个文件或者文件夹进行重命名,这个时候我们就可以使用命令"mv"了,此命令格式如下:

mv [参数] [源地址] [目的地址]

主要参数描述如下:

-b 如果要覆盖文件的话覆盖前先进行备份。

-f 若目标文件或目录与现在的文件重复,直接覆盖目标文件或目录。

-I 在覆盖之前询问用户。

使用上面讲解"cp"命令的时候创建了三个文件夹,在上面创建的 test1 文件夹里面创建 一个 c.c 文件,然后将 c.c 这个文件重命名为 d.c。最后将 d.c 这个文件移动到 test2 文件夹里 面,操作如下图所示:



图 2.6.19 移动文件操作

我们再将 test1 中的 d.c 文件移动到 test2 文件夹里面,操作如下图所示:

zuozhongkai@ubuntu:~/test2\$ ls//查看test2下所有文件a.c b.czuozhongkai@ubuntu:~/test2\$ cd ../test1//进入test1文件夹中zuozhongkai@ubuntu:~/test1\$ ls//查看test1文件夹所有文件a.c b.c d.c../test2//查看test2下的所有文件zuozhongkai@ubuntu:~/test1\$ mv d.c ../test2//查看test2下的所有文件zuozhongkai@ubuntu:~/test1\$ ls ../test2//查看test2下的所有文件a.c b.c d.c../test2//查看test2下的所有文件

图 2.6.20 移动文件操作

2.6.4 文件压缩和解压缩

文件的压缩和解压缩是非常常见的操作,在 Windows 下我们有很多压缩和解压缩的工具,比如 zip、360 压缩等等。在 Ubuntu 下也有压缩工具,本节我们学习 Ubuntu 下图形化以及命 令行这两种压缩和解压缩操作。

1、图形化压缩和解压缩

图形化压缩和解压缩和 Windows 下基本一样,在要压缩或者解压的文件上点击鼠标右键,然后选择要进行的操作,我们先讲解一下如何进行文件的压缩。首先找到要压缩的文件,然后在要压缩的文件上点击鼠标右键,选择"压缩"选项,如下图所示:

正点原子

原子哥在线教学	🗄: www.yua	nzige.co	om 论坛:w	ww.opened	v.com/forum.php
<	> 4 企主3	文件夹 🕨			Q ः: ≡ ● © ⊗
0	最近使用				al 📕
w.	土日末	testi	打开	回车模	板视频
	泉面		在新标签页中打开(T)	Ctrl+回车	
×80	视频	-	在新窗口中打开(W)	Shift+回车	
Ø	图片		使用其他程序打开(A)		
D	文档	图片	剪切(T)	Ctrl+X 音	乐 桌面
*	下载		复制(C)	Ctrl+C	
ត	音乐	三 万	移动到		
	回收站	11 12	复制到		
+	其他位置		移动到回收站(V)	删除	
			重命名(M)	F2	
			压缩(O)		进力了"tast?" (今右 0 顶)
			在终端打开(E)		远中」(ESt2(日有 0 项)
			恢复到上一版本…		
			发送到		
			本地网络共享		
			属性(P)	Ctrl+I	

图 2.6.21 文件压缩

在上图中我们要对 test2 这个文件夹进行压缩,点击"压缩"以后会弹出下图所示界面让 选择压缩后的文件名和压缩格式。

取消	创建归档		创建
归档名称			
test2			
o.zip	○ .tar.xz	○ .7z	
与所有系统	兼容。		

图 2.6.22 压缩命名和格式选择

在上图中,设置好压缩以后的文件名,然后选择压缩格式,可选的压缩格式如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

取消	创建归档		创建
归档名称 test2			
•.zip	○ .tar.xz	○ .7z	
与所有系统	兼容。		

图 2.6.23 可选压缩格式

从上图中可以看出,可以选择的压缩格式只有三个。这是因为这三个压缩格式与所有系 统兼容。挑选一个格式进行压缩,比如我选择的".zip"这个格式,压缩完成以后如下图所示:



图 2.6.24 压缩完成的文件

上面就是使用图形化进行文件压缩的过程,我们接下来对刚刚压缩的 test2.zip 进行解压 缩, 鼠标放到 test2.zip 上然后点击鼠标右键, 选择"提取到此处", 如下图所示:



正点原子

图 2.6.25 解压缩文件

点击上图中的"提取到此处"以后,系统就会自动进行解压缩,上面就是在 Ubuntu 中使用图形化工具进行文件的压缩和解压缩。

2、命令行进行文件的压缩和解压缩

上面我们学习了如何使用图形化工具在 Ubuntu 下进行文件的压缩和解压缩,本节我们学 学如何使用命令行进行压缩和解压缩,我们后面的开发中所有涉及到压缩和解压缩的操作都 是在命令行下完成的。命令行下进行压缩和解压缩常用的命令有三个: zip、unzip 和 tar,我 们依次来学习:

①、命令 zip

zip命令看名字就知道是针对.zip文件的,用于将一个或者多个文件压缩成一个.zip结尾的 文件,命令格式如下:

zip [参数] [压缩文件名.zip] [被压缩的文件]

主要参数函数如下:

-b<工作目录> 指定暂时存放文件的目录。

- -d 从 zip 文件中删除一个文件。
- -F 尝试修复已经损毁的压缩文件。
- -g 将文件压缩入现有的压缩文件中,不需要新建压缩文件。
- -h 帮助。

-j 只保存文件的名,不保存目录。

-m 压缩完成以后删除源文件。

-n<字尾符号> 不压缩特定扩展名的文件。

-q 不显示压缩命令执行过程。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

-r 递归压缩,将指定目录下的所有文件和子目录一起压缩。

-v 显示指令执行过程。

-num 压缩率,为1~9的数值。

上面讲解了如何使用图形化压缩工具对文件夹 test2 进行压缩,这里我们使用命令 "zip" 对 test2 文件夹进行压缩,操作如下图所示:

zuozhongkai@ubunt	u:~\$ ls 🏼 //	/显示当前目录下	所有文件		
examples.desktop	test2 te	st.txt 🖄	もい 视频	文档 音乐	£
test1	test3 tm	p 模材	反 图片	下载 桌面	面
zuozhongkai@ubunt	u:~\$ zip -	rv test2.zi	p test2	//使用zip命令i	进行压缩
adding: test2/	(in	=0) (out=0)	(stored	0%)	
adding: test2/d	.c (in	=0) (out=0)	(stored	0%)	
adding: test2/b	.c (in	=0) (out=0)	(stored	0%)	
adding: test2/a	.c (in	=0) (out=0)	(stored	0%)	
total bytes=0, com	mpressed=0) -> 0% savi	ngs		
zuozhongkai@ubunt	u:~\$ ls			//显示压缩以后	的当前目录下所有文件
examples.desktop	test2	test3	tmp	模板 图片	下载 桌面
test1	test2.zip	test.txt	公共的	视频 文档	音乐

图 2.6.26 使用 zip 进行文件压缩

上图就是使用 zip 命令进行 test2 文件夹的压缩,我们使用的命令如下:

zip -rv test2.zip test2

上述命令中,-rv 表示递归压缩并且显示压缩命令执行过程。

② 命令 unzip

unzip 命令用于对.zip 格式的压缩包进行解压, 命令格式如下:

unzip [参数] [压缩文件名.zip]

主要参数如下:

- -1 显示压缩文件内所包含的文件。
- -t 检查压缩文件是否损坏,但不解压。
- -v 显示命令显示的执行过程。
- -Z 只显示压缩文件的注解。
- -C 压缩文件中的文件名称区分大小写。
- -j 不处理压缩文件中的原有目录路径。
- -L 将压缩文件中的全部文件名改为小写。
- -n 解压缩时不要覆盖原有文件。
- -**P<密码>** 解压密码。
- -q 静默执行,不显示任何信息。
- -x<文件列表> 指定不要处理.zip 中的哪些文件。
- -d<目录> 把压缩文件解到指定目录下。

对上面压缩的 test2.zip 文件使用 unzip 命令进行解压缩,操作如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

zuozhongkai@ubuntu:~\$ rm -rf test2 //删除以前的test2文件夹,防止干扰解压过程										
examples.desktop	test2.zip	test.txt	公共的	视频	文档	音乐				
test1	test3	tmp	模板	图片	下载	桌面				
zuozhongkai@ubunt	u:~\$ unzip	test2.zip	//解压缩 t e	st2.zip3	7件					
Archive: test2.z	ip									
creating: test	2/									
extracting: test	2/d.c									
extracting: test	2/b.c									
extracting: test	2/a.c									
zuozhongkai@ubunt	u:~\$ ls		//显示当前	目录下所	有文件					
examples.desktop	test2	test3	tmp	模板	图片	下载	桌面			
test1	test2.zip	test.txt	公共的	视频	文档	音乐				

图 2.6.27 命令 unzip 使用演示

③、命令 tar

我们前面讲的 zip 和 unzip 这两个是命令只适用于.zip 格式的压缩和解压,其它压缩格式 就用不了了,比如 Linux 下最常用的.bz2 和.gz 这两种压缩格式。其它格式的压缩和解压使用 命令 tar,tar 将压缩和解压缩集合在一起,使用不同的参数即可,命令格式如下:

tar [参数] [压缩文件名] [被压缩文件名] 常用参数如下: -c 创建新的压缩文件。 -C<目的目录> 切换到指定的目录。 -f<备份文件> 指定压缩文件。 -j 用 tar 生成压缩文件, 然后用 bzip2 进行压缩。 -k 解开备份文件时,不覆盖已有的文件。 -m 还原文件时,不变更文件的更改时间。 **-r** 新增文件到已存在的备份文件的结尾部分。 -t 列出备份文件内容。 -v 显示指令执行过程。 -w 遭遇问题时先询问用户。 -x 从备份文件中释放文件,也就是解压缩文件。 -z 用 tar 生成压缩文件,用 gzip 压缩。 -Z 用 tar 生成压缩文件,用 compress 压缩。

使用 tar 命令来进行.zip 和.gz 格式的文件压缩,操作如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

zuozhongkai@ubunt	u:~\$ ls	//显示当	前目录下的	的所有文件				
examples.desktop	test2	test.tx	t 公共	的 视频	文档 音	乐		
test1	test3	tmp	模板	图片	下载 桌	面		
zuozhongkai@ubuntu	u:~\$ tau	r -vcjf i	test1.ta	ar.bz2 tes	st1 //压缩	为.bz2格;	式	
test1/								
test1/c.c								
test1/b.c								
test1/a.c								
zuozhongkai@ubuntu	ı:~ \$ ls				//查看日	缩后的文	件是否存	在
examples.desktop	test1.	tar.bz2	test3	tmp	模板	图片 「	下载	桌面
test1	test2		test.t	xt 公共的	│ 视频	文档 i	音乐	
zuozhongkai@ubunt	ı:∼\$ taı	r -vczf i	test1.ta	ar.gz test	1 //压缩为	.gz格式		
test1/								
test1/c.c								
test1/b.c								
test1/a.c								
zuozhongkai@ubuntu	ı:~\$ ls				//查看压缩	宿后的文件	牛是否存	在
examples.desktop	test1.	tar.bz2	test2	test.txt	公共的	视频	文档	音乐
test1	test1.	tar.gz	test3	tmp	模板	图片	下载	桌面

图 2.6.28 tar 命令进行压缩

在上图中,我们使用如下两个命令将 test1 文件夹压缩为.bz2 和.gz 这两个格式:

tar-vcjf test1.tar.bz2 test1

tar -vczf test1.tar.gz test1

在上面两行命令中,-vcjf表示创建 bz2 格式的压缩文件,-vczf表示创建.gz 格式的压缩文 件。学习了如何使用 tar 命令来完成压缩,我们再来学习使用 tar 命令完成文件的解压,操作 如下图所示:

zuozhongkai@ubuntu	ı:~\$ls //	显示当前目录了	下所有文件				
examples.desktop	test2.tar.gz	test.txt	公共的	视频	文档	音乐	
test1.tar.bz2	test3	tmp	模板	图片	下载	桌面	
zuozhongkai@ubuntu	ı:~\$ tar -vxjf	test1.tar.	.bz2 //#	压缩.bz2	2格式文件		
test1/							
test1/c.c							
test1/b.c							
test1/a.c							
zuozhongkai@ubuntu	ı:~\$ ls		//检查	≦test1.t a	ar.bz2是	否解压缩成功	
examples.desktop	test1.tar.bz2	test3	tmp	模板	图片	下载桌	面
test1	test2.tar.gz	test.txt	公共的	视频	文档	音乐	
zuozhongkai@ubuntu	ı:~\$ tar -vxzf	test2.tar.	.gz //解	压缩.gz格	試文件		
test2/							
test2/d.c							
test2/b.c							
test2/a.c	+ 1		//絵	杳test2.t	ar oz 🗠	李解压综成功	
zuozhongkai@ubuntu	1:~\$ LS						
examples.desktop	test2	test.txt	<u> </u>	档 臬	面		
test1	test2.tar.gz	tmp	祝频 下	報			
testi.tar.bz2	test3	公共的	图方 音				

图 2.6.29 tar 解压缩命令

上图中我们使用如下所示两行命令完成.bz2和.gz格式文件的解压缩:

tar -vxjf test1.tar.bz2

tar -vxzf test2.tar.gz



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

上述两行命令中,-vxif用来完成.bz2格式压缩文件的解压,-vxzf用来完成.gz格式压缩文 件的解压。关于 Ubuntu 下的命令行压缩和解压缩就讲解到这里, 重点是 tar 命令, 要熟练掌 握使用 tar 命令来完成.bz2 和.gz 格式的文件压缩和解压缩。

2.6.5 文件查询和搜索

文件的查询和搜索也是最常用的操作,在嵌入式 Linux 开发中常常需要在 Linux 源码文件 中查询某个文件是否存在,或者搜索哪些文件都调用了某个函数等等。本节我们就讲解两个 最常用的文件查询和搜索命令: find 和 grep。

1、命令 find

find 命令用于在目录结构中查找文件,其命令格式如下:

[路径] [参数] [关键字] find

路径是要查找的目录路径,如果不写的话表示在当前目录下查找,关键字是文件名的-部分,主要参数如下:

-name<filename> 按照文件名称查找,查找与 filename 匹配的文件,可使用通配符。

-depth 从指定目录下的最深层的子目录开始查找。

-gid<群组识别码> 查找符合指定的群组识别码的文件或目录。

-group<群组名称> 查找符合指定的群组名称的文件或目录。

-size<文件大小> 查找符合指定文件大小的文件。

-tvpe<文件类型> 查找符合指定文件类型的文件。

-user<拥有者名称>查找符合指定的拥有者名称的文件或目录。

find 命令的参数有很多,常用的就这些,关于其它的参数大家可以自行上网查找,我们 来看一下如何使用 find 命令进行文件搜索,我们搜索目录/etc 中以"vim"开头的文件为例, 操作如下图所示:

<pre>zuozhongkai@ubuntu:~\$ find /etc/ -name vim*</pre>	
/etc/vim	
/etc/vim/vimrc	
/etc/vim/vimrc.tiny	
find: `/etc/cups/ssl': 权限不够	
/etc/alternatives/vim	
/etc/alternatives/vimdiff	
find: `/etc/polkit-1/localauthority': 权限不够	
find: `/etc/ssl/private': 权限不够	

图 2.6.30 find 命令操作

从上图可以看出,在目录/etc下,包含以"vim*"开头的文件有/etc/vim、/etc/vim/vimrc 等等,就不一一列出了。

2、命令 grep

find 命令用于在目录中搜索文件,我们有时候需要在文件中搜索一串关键字, grep 就是 完成这个功能的, grep 命令用于查找包含指定关键字的文件, 如果发现某个文件的内容包含 所指定的关键字, grep 命令就会把包含指定关键字的这一行标记出来, grep 命令格式如下:

grep [参数] 关键字 文件列表

grep 命令一次只能查一个关键字, 主要参数如下:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

-b 在显示符合关键字的那一列前,标记处该列第1个字符的位编号。

-c 计算符合关键字的列数。

-d<进行动作> 当指定要查找的是目录而非文件时,必须使用此参数! 否则 grep 指令将 回报信息并停止搜索。

- -i 忽略字符大小写。
- -v 反转查找,只显示不匹配的行。
- -r 在指定目录中递归查找。

比如我们在目录/usr下递归查找包含字符"Ubuntu"的文件,操作如下图所示:

```
zuozhongkai@ubuntu:-$ grep -ir "Ubuntu" /usr
匹配到二进制文件 /usr/bin/fcitx-qimpanel-configtool
匹配到二进制文件 /usr/bin/nsupdate
/usr/bin/apport-bug:# Author: Martin Pitt <martin.pitt@ubuntu.com>
/usr/bin/apport-bug:# this so that confined applications using ubuntu-browsers.d
/ubuntu-integration
匹配到二进制文件 /usr/bin/locale
/usr/bin/fcitx-configtool: LXDE|Lubuntu)
匹配到二进制文件 /usr/bin/x86 64-linux-gnu-elfedit
```

图 2.6.31 命令 grep 演示

2.6.6 文件类型

这里的文件类型不是说这个文件是音乐文件还是文本文件,在用户根目录下使用命令 "ls-1"来查看用户根目录下所有文件的详细信息,如下图所示:

zuozhongkai(@u	i <mark>buntu:~</mark> \$ le	5 -l					
总用量 72								
-rw-rr	1	zuozhongkai	zuozhongkai	8980	12月	18	02:08	examples.desktop
drwxrwxr-x	2	zuozhongkai	zuozhongkai	4096	12月	24	21:56	test1
- rw- rw- r	1	zuozhongkai	zuozhongkai	194	12月	24	21:58	test1.tar.bz2
drwxrwxr-x	2	zuozhongkai	zuozhongkai	4096	12月	23	01:00	test2
- rw- rw- r	1	zuozhongkai	zuozhongkai	171	12月	24	22:09	test2.tar.gz
drwxrwxr-x	2	zuozhongkai	zuozhongkai	4096	12月	23	00:45	test3
- rw- rw- r	1	zuozhongkai	zuozhongkai	68	12月	21	01:40	test.txt
drwxrwxr-x	2	zuozhongkai	zuozhongkai	4096	12月	19	21:41	tmp
drwxr-xr-x	2	zuozhongkai	zuozhongkai	4096	12月	18	02:21	公共的
drwxr-xr-x	2	zuozhongkai	zuozhongkai	4096	12月	18	02:21	模板
drwxr-xr-x	2	zuozhongkai	zuozhongkai	4096	12月	18	02:21	视频
drwxr-xr-x 3	2	zuozhongkai	zuozhongkai	4096	12月	18	02:21	图片
drwxr-xr-x	2	zuozhongkai	zuozhongkai	4096	12月	18	02:21	文档
drwxr-xr-x 🕻	2	zuozhongkai	zuozhongkai	4096	12月	18	02:21	下载
drwxr-xr-x	2	zuozhongkai	zuozhongkai	4096	12月	18	02:21	音乐
drwxr-xr-x 2	2	zuozhongkai	zuozhongkai	4096	12月	22	02:08	桌面

图 2.6.32 文件详细信息

在上图中,每个文件的详细信息占一行,每行最前面都是一个符号就标记了当前文件类型,比如 test1 的第一个字符是"d", test1.tar.bz2 文件第一个字符是"-"。这些字符表示的文件类型如下:

- 普通文件,一些应用程序创建的,比如文档、图片、音乐等等。
- d 目录文件。
- c 字符设备文件, Linux 驱动里面的字符设备驱动,比如串口设备,音频设备等。
- **b** 块设备文件,存储设备驱动,比如硬盘,U盘等。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

1 符号连接文件,相当于 Windwos 下的快捷方式。

- s 套接字文件。
- p 管道文件,主要指 FIFO 文件。

我们后面学习 Linux 驱动开发的时候基本是在和字符设备文件和块设备文件打交道。

2.7 Linux 用户权限管理

2.7.1 Ubuntu 用户系统

Ubuntu 是一个多用户系统,我们可以给不同的使用者创建不同的用户账号,每个用户使 用各自的账号登陆,使用用户账号的目的一是方便系统管理员管理,控制不同用户对系统的 访问权限,另一方面是为用户提供安全性保护。

我们前面在安装 Ubuntu 系统的时候被要求创建一个账户,当我们创建好账号以后,系统 会在目录/home下以该用户名创建一个文件夹,所有与该用户有关的文件都会被存储在这个文 件文件夹中。同样的,创建其它用户账号的时候也会在目录/home下生成一个文件夹来存储该 用户的文件,下图就是我的电脑上"zuozhongkai"这个账户的文件夹。

zuozhongkai@ubunt	u:~\$	ls								
examples.desktop	tmp	公共的	模板	视频	图片	文档	下载	音乐	桌面	

图 2.7.1 用户账号根目录

装系统的时候创建的用户其权限比后面创建的用户大一点,但是没有 root 用户权限大, Ubuntu 下用户类型分为以下 3 类:

● 初次创建的用户,此用户可以完成比普通用户更多的功能。

● root 用户,系统管理员,系统中的玉皇大帝,拥有至高无上的权利。

● 普通用户,安装完操作系统以后被创建的用户。

以上三种用户,每个用户都有一个 ID 号,称为 UID,操作系统通过 UID 来识别是哪个用户,用户相关信息可以在文件/etc/passwd 中查看到,如下图所示:

```
zuozhongkai@ubuntu:/$ cat /etc/passwd
root:x:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
hplip:x:115:7:HPLIP system user,,:/var/run/hplip:/bin/false
kernoops:x:116:65534:Kernel Oops Tracking Daemon,,:/:/bin/false
pulse:x:117:124:PulseAudio daemon,,:/var/run/pulse:/bin/false
rtkit:x:118:126:RealtimeKit,,:/proc:/bin/false
saned:x:119:127::/var/lib/saned:/bin/false
usbmux:x:120:46:usbmux daemon,,:/var/lib/usbmux:/bin/false
zuozhongkai:x:1000:1000:zuozhongkai,,:/home/zuozhongkai:/bin/bash
guest-sjhdig:x:999:999:ij客:/tmp/guest-sjhdig:/bin/bash
```

图 2.7.2 passwd 文件内容



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

从配置文件 passwd 中可以看到,每个用户名后面都有两个数字,比如用户"zuozhongkai" 后面"1000:1000",第一个数字是用户的 ID,另一个是用户的 GID,也就是用户组 ID。 Ubuntu 里面每个用户都属于一个用户组里面,用户组就是一组有相同属性的用户集合。

2.7.2 权限管理

在使用 Windows 的时候我们很少接触到用户权限,最多就是打开某个软件出问题的时候 会选择以"管理员身份"打开。Ubuntu 下我们会常跟用户权限打交道,权限就是用户对于系 统资源的使用限制情况,root 用户拥有最大的权限,可以为所欲为,装系统的时候创建的用 户拥有 root 用户的部分权限,其它普通用户的权限最低。对于我们做嵌入式开发的人一般不 关注用户的权限问题,因为嵌入式基本是单用户,做嵌入式开发重点关注的是文件的权限问题。

对于一个文件通常有三种权限:读(r)、写(w)和执行(x),使用命令"ls-l"可以查看某个目录下所有文件的权限信息,如下图所示:

<mark>zuozhongka</mark> 过 总用量 48	i@1	ubuntu:~\$ ls	-1					
- rw- r r	1	zuozhongkai	zuozhongkai	8980	12月	18	02:08	examples.desktop
- rw- rw- r	1	zuozhongkai	zuozhongkai	0	12月	25	20:44	test.c
drwxrwxr-x	2	zuozhongkai	zuozhongkai	4096	12月	19	21:41	tmp
drwxr-xr-x	2	zuozhongkai	zuozhongkai	4096	12月	18	02:21	公共的
drwxr-xr-x	2	zuozhongkai	zuozhongkai	4096	12月	18	02:21	模板
drwxr-xr-x	2	zuozhongkai	zuozhongkai	4096	12月	18	02:21	视频
drwxr-xr-x	2	zuozhongkai	zuozhongkai	4096	12月	18	02:21	图片
drwxr-xr-x	2	zuozhongkai	zuozhongkai	4096	12月	18	02:21	文档
drwxr-xr-x	2	zuozhongkai	zuozhongkai	4096	12月	18	02:21	下载
drwxr-xr-x	2	zuozhongkai	zuozhongkai	4096	12月	18	02:21	音乐
drwxr-xr-x	2	zuozhongkai	zuozhongkai	4096	12月	22	02:08	桌面

图 2.7.3 文件权限信息

在上图中我们以文件 test.c 为例讲解, 文件 test.c 文件信息如下:

-rw-rw-r-- 1 zuozhongkai zuozhongkai 0 12 月 25 20:44 test.c

其中 "-rw-rw-r--"表示文件权限与用户和用户组之间的关系,第一位表示文件类型,上一 小节已经说了。剩下的 9 位以 3 位为一组,分别表示文件拥有者的权限,文件拥有者所在用 户组的权限以及其它用户权限。后面的 "zuozhongkai zuozhongkai"分别代表文件拥有者(用 户)和该用户所在的用户组,因此文件 test.c 的权限情况如下:

①、文件 test.c 的拥有者是用户 zuozhongkai,其对文件 tesst.c 的权限是 "rw-",也就是 对该文件拥有读和写两种权限。

②、用户 zuozhongkai 所在的用户组也叫做 zuozhongkai,其组内用户对于文件 test.c 的权限是"rw-",也是拥有读和写这两种权限。

③、其它用户对于文件 test.c 的权限是"r--",也就是只读权限。

对于文件,可读权限表示可以打开查看文件内容,可写权限表示可以对文件进行修改,可执行权限就是可以运行此文件(如果是软件的话)。对于文件夹,拥有可读权限才可以使用命令 ls 查看文件夹中的内容,拥有可执行权限才能进入到文件夹内部。

如果某个用户对某个文件不具有相应的权限的话就不能进行相应的操作,比如根目录"/"下的文件只有 root 用户才有权限进行修改,如果以普通用户去修改的话就会提示没有权限。比如我们要在根目录"/"创建一个文件 mytest,使用命令"touch mytest",结果如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

zuozhongkai@ubuntu:/\$ touch mytest
touch: 无法创建'mytest': 权限不够

图 2.7.4 创建文件

在上图中,我以用户"zuozhongkai"在根目录"/"创建文件 mytest,结果提示我无法创 建"mytest",因为权限不够,因为只有 root 用户才能在根目录"/"下创建文件。我们可以 使用命令"sudo"命令暂时切换到 root 用户,这样就可以在根目录"/"下创建文件 mytest 了, 如下图所示:

zuozhor [sudo] zuozhor	ngkai@ zuozho ngkai@	<mark>ubuntu:/\$</mark> sudo t ongkai 的密码: ubuntu:/\$ ls	ouch mytest					
bin	dev	<pre>initrd.img initrd.img.old lib</pre>	lib64	mnt	proc	sbin	sys	var
boot	etc		lost+found	mytest	root	snap	tmp	vmlinuz
cdrom	home		media	opt	run	srv	usr	vmlinuz.old

图 2.7.5 使用 sudo 命令创建文件

在上图中我们使用命令"sudo"以后就可以在根目录"/"创建文件 mytest,在进行其它的操作的时候,遇到提示权限不够的时候都可以使用 sudo 命令暂时以 root 用户身份去执行。

上面我们讲了,文件的权限有三种:读(r)、写(w)和执行(x),除了用r、w和x表示以外, 我们也可以使用二进制数表示,三种权限就可以使用 3 位二进制数来表示,一种权限对应一 个二进制位,如果该位为 1 就表示具备此权限,如果该位为 0 就表示没不具备此权限,如下 表所示:

字母	二进制	八进制
r	100	4
W	010	2
Х	001	1

表 2.7.1 文件权限数字表示方法

如果做过单片机开发的话,就会发现和单片机里面的寄存器位一样,将三种权限 r、w 和 x 进行不同的组合,即可得到不同的二进制数和八进制数,3 位权限可以组出 8 种不同的权限 组合,如下表所示:

权限	二进制数字	八进制数字
	000	0
x	001	1
—w—	010	2
-wx	011	3
r	100	4
r-x	101	5
rw-	110	6
rwx	111	7

表 2.7.2 文件所有权限组合



原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php

表 2.7.2 中权限所对应的八进制数字就是每个权限对应的位相加,比如权限 rwx 就是 4+2+1=7。前面的文件 test.c 其权限为 "rw-rw-r--",因此其十进制表示就是: 664。

另外我们也开始使用 a、u、g 和 o 表示文件的归属关系,用=、+和-表示文件权限的变化,如下表所示:

字母	意义
r	可读权限
W	可写权限
Х	可执行权限
a	所有用户
u	归属用户
g	归属组
0	其它用户
=	具备权限
+	添加某权限
_	去除某权限

表 2.7.3 权限修改字母表示方式

对于文件 test.c,我们想要修改其归属用户(zuozhongkai)对其拥有可执行权限,那么就可以使用: u+x。如果希望设置归属用户及其所在的用户组都对其拥有可执行权限就可以使用: gu+x。

2.7.3 权限管理命令

我们也可以使用 Shell 来操作文件的权限管理,主要用到"chmod"和"chown"这两个 命令,我们一个一个来看。

1、权限修改命令 chmod

命令 "chmod" 用于修改文件或者文件夹的权限,权限可以使用前面讲的数字表示也可以使用字母表示,命令格式如下:

chmod [参数] [文件名/目录名]

主要参数如下:

-c 效果类似"-v"参数,但仅回显更改的部分。

-f 不显示错误信息。

-R 递归处理,指定目录下的所有文件及其子文件目录一起处理。

-v 显示指令的执行过程。

我们先来学习以下如何使用命令"chmod"修改一个文件的权限,在用户根目录下创建一个文件 test,然后查看其默认权限,操作如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

zuozhongkai@	u <mark>buntu:~</mark> \$ to	uch test						
LibreOffice Calc	ubuntu:~\$ ls	-1						
芯田里 40								
-rw-rr 1	zuozhongkai	zuozhongkai	8980	12月	18	02:08	examples.de	sktop
-rw-rw-r 1	zuozhongkai	zuozhongkai	0	12月	25	22:24	test	
drwxrwxr-x 2	zuozhongkai	zuozhongkai	4096	12月	19	21:41	tmp	
drwxr-xr-x 2	zuozhongkai	zuozhongkai	4096	12月	18	02:21	公共的	
drwxr-xr-x 2	zuozhongkai	zuozhongkai	4096	12月	18	02:21	模板	
drwxr-xr-x 2	zuozhongkai	zuozhongkai	4096	12月	18	02:21	视频	
drwxr-xr-x 2	zuozhongkai	zuozhongkai	4096	12月	18	02:21	图片	
drwxr-xr-x 2	zuozhongkai	zuozhongkai	4096	12月	18	02:21	文档	
drwxr-xr-x 2	zuozhongkai	zuozhongkai	4096	12月	18	02:21	下载	
drwxr-xr-x 2	zuozhongkai	zuozhongkai	4096	12月	18	02:21	音乐	
drwxr-xr-x 2	zuozhongkai	zuozhongkai	4096	12月	22	02:08	桌面	

图 2.7.6 创建文件 test

在上图中我们创建了一个文件: test, 这个文件的默认权限为 "rw-rw-r--", 我们将其权 限改为"rwxrw-rw",对应数字就是766,操作如下:

zuozhongkai@ubuntu:~\$ ls -l test -rw-rw-r-- 1 zuozhongkai zuozhongkai 0 12月 25 22:24 test zuozhongkai@ubuntu:~\$ chmod 766 test //修 zuozhongkai@ubuntu:~\$ ls -l test -rwxrw-rw- 1 zuozhongkai zuozhongkai 0 12月 25 22:24 test

图 2.7.7 修改权限

在上图中,我们修改文件 test 的权限为 766,修改完成以后的 test 文件权限为 "rwxrwrw-",和我们设置的一样,说明权限修改成功。

上面我们是通过数字来修改权限的,我们接下来使用字母来修改权限,操作如下图所示:

zuozhongkai@ubuntu:~\$ touch a.c / zuozhongkai@ubuntu:~\$ ls -l a.c -rw-rw-r-- 1 zuozhongkai zuozhongkai 0 12月 25 22:33 a.c zuozhongkai@ubuntu:~\$ chmod u+x a.c zuozhongkai@ubuntu:~\$ ls -l a.c -rwxrw-r-- 1 zuozhongkai zuozhongkai 0 12月 25 22:33 a.c

图 2.7.8 使用字母修改文件权限

上面两个例子都是修改文件的权限,接下来我们修改文件夹的权限,新建一个 test 文件 夹,在文件夹 test 里面创建 a.c、b.c 和 c.c 三个文件,如下图所示:

zuozhongkaj@ubuntu:~\$ ls _l test/
-rw-rw-r l zuozhongkai zuozhongkai 0 12月 26 00:20 a.c
-rw-rw-r 1 zuozhongkai zuozhongkai 0 12月 26 00:20 b.c
-rw-rw-r 1 zuozhongkai zuozhongkai 0 12月 26 00:20 c.c

图 2.7.9 test 文件夹

在上图中 test 文件夹下的文件 a.c、b.c 和 c.c 的权限均为 "rw-rw-r--", 我们将 test 文件夹 下的所有文件权限都改为"rwxrwxrwx",也就是数字777,操作如下图所示:



尽 了	可仁线钗子: www.yuanzige.com 化坛:www.openeuv.com/forum.pnp	
	zuozhongkai@ubuntu:~\$ chmod -R 777 test/ //递归修改文件权限	
	zuozhongkai@ubuntu:~\$ ls -l test/ //查看修改权限以后的文件	
	心府重 0 -rwxrwxrwx 1 zuozhongkai zuozhongkai 0 12月 26 00:20 a.c	
	-rwxrwxrwx 1 zuozhongkai zuozhongkai 0 12月 26 00:20 b.c	
	-rwxrwxrwx 1 zuozhongkai zuozhongkai 0 12月 26 00:20 c.c	

图 2.7.10 递归修改文件夹权限

2、文件归属者修改命令 chown

命令 chown 用来修改某个文件或者目录的归属者用户或者用户组,命令格式如下: chown [参数] [用户名.<组名>] [文件名/目录]

其中[用户名.<组名>]表示要将文件或者目录改为哪一个用户或者用户组,用户名和组名用"."隔开,其中用户名和组名中的任何一个都可以省略,命令主要参数如下:

- -c 效果同-v类似,但仅回显更改的部分。
- -f 不显示错误信息。
- -h 只对符号连接的文件做修改,不改动其它任何相关的文件。

-R 递归处理,将指定的目录下的所有文件和子目录一起处理。

-v 显示处理过程。

在用户根目录下创建一个 test 文件, 查看其文件夹所属用户和用户组, 如下图所示:

zuozhongkai@ubuntu:~\$ touch test **zuozhongkai@ubuntu:**~\$ ls -l test -rw-rw-r-- 1 zuozhongkai zuozhongkai 0 12月 26 00:36 test

图 2.7.11 test 文件信息查询

从上图中可以看出,文件 test 的归属用户为 zuozhongkai,所属的用户组为 zuozhongkai,将文件 test 归属用户改为 root 用户,所属的用户组也改为 root,操作如下图所示:

zuozhongkai@ubuntu:~\$ ls -l test //显示文件test的归属用户和归属组
-rw-rw-r 1 zuozhongkai zuozhongkai 0 12月 26 00:45 test
zuozhongkai@ubuntu:~\$ sudo chown root.root test //修改文件test的归属用户和归属组
zuozhongkai@ubuntu:~\$ ls -l test //查看修改以后的文件test归属用户和归属组
-rw-rw-r 1 root root_0 12月 26 00:45 test

图 2.7.12 修改文件归属用户和归属组

命令 shown 同样也可以递归处理来修改文件夹的归属用户和用户组,用法和命令 chown 一样,这里就不演示了。

2.8 Linux 磁盘管理

2.8.1 Linux 磁盘管理基本概念

Linux 的磁盘管理体系和 Windows 有很大的区别,在 Windows 下经常会遇到"分区"这个概念,在 Linux 中一般不叫"分区"而叫"挂载点"。"挂载点"就是将一个硬盘的一部分做成文件夹的形式,这个文件夹的名字就是"挂载点",不管在哪个发行版的 Linux 中,用户是绝对看到不到 C 盘、D 盘这样的概念的,只能看到以文件夹形式存在的"挂载点".

文件/etc/fstab 详细的记录了 Ubuntu 中硬盘分区的情况,如图 2.8.1 所示:



正点原子

图 2.8.1 文件 fstab

在图 2.8.1 中有一行"/ was on /dev/sda1 during installation", 意思是根目录"/"是在 /dev/sda1上的, 其中"/"是挂载点, "/dev/sda1"就是我们装 Ubuntu 系统的硬盘。由于我们 的系统是安装在虚拟机中的, 因此图 2.8.1 没有出现实际的硬盘。可以通过如下命令查看当前 系统中的磁盘:

ls /dev/sd*

上述命令就是打印出所有以/dev/sd开头的设备文件,如图 2.8.2 所示:

zuozhongkai@ubuntu:~\$ ls /dev/sd* /dev/sda /dev/sda1 /dev/sda2 /dev/sda5

图 2.8.2 查看硬盘设备文件

在图 2.8.2 中有四个磁盘设备文件,其中 sd 表示是 SATA 硬盘或者其它外部设备,最后面的数字表示该硬盘上的第 n 个分区,比如/dev/sda1 就表示磁盘 sda 上的第一个分区。图 2.8.2 中都是以/dev/sda 开头的,说明当前只有一个硬盘。如果再插上 U 盘、SD 卡啥的就可能会出现/dev/sdb,/dev/sdc 等等。如果你的 U 盘有两个分区那么可能就会出现/dev/sdb1、dev/sdb2 这样的设备文件。比如我现在插入我的 U 盘,插入 U 盘会提示 U 盘是接到主机还是虚拟机,如图 2.8.3 所示:

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

正点原子

检测到新的 USB 设备 >	<						
选择您希望将 Kingston DataTraveler 3.0 连接到的位置							
 ○ 连接到主机 ● 连接到虚拟机 1、连接到虚拟机 							
虚拟机名称 Ubuntu 64 位 2、选中要连接的虚拟机名字							
☑记住我的选择,以后不再询问 3、确定							
确定							

图 2.8.3 U 盘连接选择

设置好图 2.8.3 以后,点击"确定"按钮 U 盘就会自动连接到虚拟机中,也就是连接到 Ubuntu 系统中,我们再次使用命令"ls /dev/sd*"来查看当前的"/dev/sd*"设备文件,如图 2.8.4 所示:

zuozhongkai@ubuntu:~\$ ls /dev/sd*									
/dev/sda	/dev/sda1	/dev/sda2	/dev/sda5	/dev/sdb	/dev/sdb1				

图 2.8.4 插入 U 盘后的设备文件

从图 2.8.4 可以看出,相比图 2.8.2 多了/dev/sdb 和/dev/sdb1 这两个文件,其中/dev/sdb 就 是 U 盘文件,/dev/sdb1 表示 U 盘的第一个分区,因为我的 U 盘就一个分区。

2.8.2 磁盘管理命令

本节我们学习以下跟磁盘操作有关的命令,这些命令如下:

1、磁盘分区命令 fdisk

如果要对某个磁盘进行分区,可以使用命令 fdisk,命令格如下:

fdisk [参数]

主要参数如下:

-b<分区大小> 指定每个分区的大小。

-l 列出指定设备的分区表。

-s<分区编号> 将指定的分区大小输出到标准的输出上,单位为块。

-u 搭配"-l"参数,会用分区数目取代柱面数目,来表示每个分区的起始地址。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php 比如我要对 U 盘进行分区,千万不要对自己装 Ubuntu 系统进行分区!!!可以使用如下

命令:

sudo fdisk /dev/sdb 结果如图 2.8.5 所示:

> zuozhongkai@ubuntu:~\$ sudo fdisk /dev/sdb [sudo] zuozhongkai 的密码: Welcome to fdisk (util-linux 2.27.1). Changes will remain in memory only, until you decide to write them. Be careful before using the write command. 命令(输入 m 获取帮助):

> > 图 2.8.5 U 盘分区界面

在图 2.8.5 中提示我们输入"m"可以查看帮助,因为 fdisk 还有一些字命令,通过输入"m"可以查看都有哪些子命令,如图 2.8.6 所示:

命令(输入 m 获取帮助): m Help: DOS (MBR) toggle a bootable flag а b edit nested BSD disklabel toggle the dos compatibility flag С Generic delete a partition d F list free unpartitioned space list known partition types ι add a new partition print the partition table n р change a partition type t verify the partition table V i print information about a partition Misc m print this menu change display/entry units U. extra functionality (experts only) х Script load disk layout from sfdisk script file dump disk layout to sfdisk script file 0 Save & Exit write table to disk and exit W quit without saving changes q Create a new label create a new empty GPT partition table g G create a new empty SGI (IRIX) partition table create a new empty DOS partition table 0 create a new empty Sun partition table S

图 2.8.6 fdisk 命令的子命令

图 2.8.6 中常用的命令如下:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

- **p** 显示现有的分区
- n 建立新分区
- t 更改分区类型
- d 删除现有的分区
- a 更改分区启动标志
- w 对分区的更改写入到硬盘或者存储器中。
- q 不保存退出。

由于我的 U 盘里面还有一些重要的文件,所以现在不能进行分区,所以就不演示 fdisk 的 分区操作了,后面我们讲解裸机例程的时候需要将可执行的 bin 文件烧写到 SD 卡中,烧写到 SD 卡之前需要对 SD 卡进行分区,到时候再详细讲解如何使用 fdisk 命令对磁盘进行分区。

2、格式化命令 mkfs

使用命令 fdisk 创建好一个分区以后,我们需要对其格式化,也就是在这个分区上创建一 个文件系统, Linux 下的格式化命令为 mkfs, 命令格式如下:

[参数] [-t 文件系统类型] [分区名称] mkfs

主要参数如下:

fs 指定建立文件系统时的参数

-V 显示版本信息和简要的使用方法。

-v 显示版本信息和详细的使用方法。

比如我们要格式化U盘的分区/dev/sdb1为FAT格式,那么就可以使用如下命令:

mkfs -t vfat /dev/sdb1

3、挂载分区命令 mount

我们创建好分区并且格式化以后肯定是要使用硬盘或者 U 盘的,那么如何访问磁盘呢? 比如我的 U 盘就一个分区,为/dev/sdb1,如果直接打开文件/dev/sdb1 会发现根本就不是我们 要的结果。我们需要将/dev/sdb1 这个分区挂载到一个文件夹中, 然后通过这个文件访问 U 盘, 磁盘挂载命令为 mount, 命令格式如下:

mount [参数] -t [类型] [设备名称] [目的文件夹]

命令主要参数有:

-V 显示程序版本。

- -h 显示辅助信息。
- -v 显示执行过程详细信息。
- -oro 只读模式挂载。
- -orw 读写模式挂载。
- -s-r 等于-oro。
- -w 等于-orw。

挂载点是一个文件夹,因此在挂载之前先要创建一个文件夹,一般我们把挂载点放到 "/mnt" 目录下,在 "/mnt" 下创建一个 tmp 文件夹, 然后将 U 盘的/dev/sdb1 分区挂载到 /mnt/tmp 文件夹里面, 操作如图 2.8.7 所示:


原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

zuozhongkai@ubuntu:~\$ ls /mnt //查看/mnt目录 zuozhongkai@ubuntu:~\$	下的所有文件
zuozhongkai@ubuntu:~\$ sudo mkdir /mnt/tmp [sudo] zuozhongkai 的家码:	//创建文件夹/mnt/tmp
<pre>zuozhongkai@ubuntu:~\$ ls /mnt</pre>	//查看/mnt目录下的所有文件
<pre>tmp zuozhongkai@ubuntu:~\$ sudo mount -t vfat /o</pre>	dev/sdb1 /mnt/tmp //挂载U盘到/mnt/tmp中
zuozhongkai@ubuntu:~\$ ls /mnt/tmp //查看/mnt	/tmp中的文件,也就是U盘里面的文件
AD IMX	6UL_ZERO_V1.0.zip 开发板光盘
ARM裸机与嵌入式Linux驱动开发V1.0.docx Sys	tem Volume Information

图 2.8.7 挂载 U 盘

在图 2.8.7 中我们将 U 盘以 fat 格式挂载到目录/mnt/tmp 中, 然后我们就可以通过访问 /mnt/tmp 来访问 U 盘了。

4、卸载命令 umount

当我们不在需要访问已经挂载的 U 盘,可以通过 umount 将其从卸载点卸除,命令格式如 下:

umount [参数] -t [文件系统类型] [设备名称]

-a 卸载/etc/mtab 中的所有文件系统。

-h 显示帮助。

-n 卸载时不要将信息存入到/etc/mtab 文件中

-r 如果无法成功卸载,则尝试以只读的方式重新挂载。

-t <文件系统类型> 仅卸载选项中指定的文件系统。

-v 显示执行过程。

上面我们将 U 盘挂载到了文件夹/mnt/tmp 里面,这里我们使用命令 umount 将其卸载掉, 操作如图 2.8.8 所示:



图 2.8.8 卸载 U 盘

在图 2.8.8 中,我们使用命令 umount 卸载了 U 盘,卸载以后当我们再去访问文件夹 /mnt/tmp 的时候发现里面没有任何文件了,说明我们卸载成功了。



正点原子

第三章 Linux C 编程入门

在 Windows 下我们可以使用各种各样的 IDE 进行编程,比如强大的 Visual Studio。但是 在 Ubuntu 下如何进行编程呢? Ubuntu 下也有一些可以进行编程的工具,但是大多都只是编辑 器,也就是只能进行代码编辑,如果要编译的话就需要用到 GCC 编译器,使用 GCC 编译器 肯定就要接触到 Makefile。本章就讲解如何在 Ubuntu 下进行 C 语言的编辑和编译、GCC 和 Makefile 的使用和编写。通过本章的学习可以掌握 Linux 进行 C 编程的基本方法,为以后的 Linux 驱动学习做准备。



正点原子

3.1 Hello World!

我们所说的编写代码包括两部分:代码编写和编译,在Windows下可以使用Visual Studio 来完成这两部,可以在 Visual Studio 下编写代码然后直接点击编译就可以了。但是在 Linux 下 这两部分是分开的,比如我们用 VIM 进行代码编写,编写完成以后再使用 GCC 编译器进行 编译,其中代码编写工具很多,比如 VIM 编辑器、Emacs 编辑器、VScode 编辑器等等,本教 程使用 Ubuntu 自带的 VIM 编辑器。先来编写一个最简单的"Hello World"程序,把 Linux 下 的C编程完整的走一遍。

3.1.1 编写代码

先在用户根目录下创建一个工作文件夹: C_Program,所有的C语言练习都保存到这个工 作文件夹下, 创建过程如图 3.1.1.1 所示:

zuozhongkai@ubuntu:~\$ mkdir C Program zuozhongkai@ubuntu:~\$ ls 模板 图片 C Program test 下载 桌面 examples.desktop 公共的 视频 文档 音乐

图 3.1.1.1 创建工作目录

进入图 3.1.1.1 创建的 C_Program 工作文件夹,为了方便管理,我们后面每个例程都创建 一个文件夹来保存所有与本例程有关的文件, 创建一个名为"3.1"的文件夹来保存我们的 "Hello World"程序相关的文件,操作如图 3.1.1.2 所示:

zuozhongkai@ubuntu:~\$ cd C Program/ zuozhongkai@ubuntu:~/C_Program\$ mkdir 3.1
zuozhongkai@ubuntu:~/C_Program\$ ls 3.1 zuozhongkai@ubuntu:~/C Program\$

图 3.1.1.2 创建工程文件夹

前面说了我们使用 VI 编辑器,在使用 VI 编辑器之前我们先做如下设置:

1、设置 TAB 键为 4 字节

VI编辑器默认 TAB 键为 8 空格,我们改成 4 空格,用 vi 打开文件/etc/vim/vimrc,在此文 件最后面输入如下代码:

set ts=4

添加完成如图 3.1.1.3 所示:

Source a global configuration file if available if filereadable("/etc/vim/vimrc.local") source /etc/vim/vimrc.local endif set ts=4

图 3.1.1.3 设置 TAB 为四个空格

修改完成以后保存并关闭文件。

2、VIM 编辑器显示行号



原子哥在线教学: www.yuanzige.com 论坛:

论坛:www.openedv.com/forum.php

VIM 编辑器默认是不显示行号的,不显示行号不利于代码查看,我们设置 VIM 编辑器显示行号,同样是通过在文件/etc/vim/vimrc 中添加代码来实现,在文件最后面加入下面一行代码即可:

set nu

添加完成以后的/etc/vim/vimrc 文件如图 3.1.1.4 所示:



图 3.1.1.4 设置 VIM 编辑器显示行号

VIM 编辑器可以自行定制,网上有很多的博客讲解如何设置 VIM,感兴趣的可以上网看一下。设置好 VIM 编辑器以后就可以正式开始编写代码了,进入前面创建的"3.1"这个工程 文件夹里面,使用 vi 指令创建一个名为"main.c"的文件,然后在里面输入如下代码:

示例代码 3.1

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

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

编写完成以后保存退出 vi 编辑器,可以使用 "cat"命令查看代码是否编写成功,如图 3.1.1.5 所示:

```
zuozhongkai@ubuntu:~/C_Program/3.1$ cat main.c
#include <stdio.h>
int main(int argc, char *argv[])
{
    printf("Hello World!\n");
}
```

图 3.1.1.5 查阅程序源码

从图 3.1.1.5 可以看出 main.c 文件是编辑成功的,代码编辑成功以后我们需要对其进行编译。

3.1.2 编译代码

Ubuntu下的 C 语言编译器是 GCC, GCC 编译器在我们安装 Ubuntu 的时候就已经默认安装好了,可以通过如下命令查看 GCC 编译器的版本号:

gcc -v

在终端中输入上述命令以后终端输出如图 3.1.2.1 所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

zuozhongkai@ubuntu:~/C Program/3.1\$ gcc -v
Using built-in specs.
COLLECT GCC=gcc
COLLECT_LTO WRAPPER=/usr/lib/gcc/x86 64-linux-gnu/5/lto-wrapper
Target: x86 64-linux-gnu
Configured with:/src/configure -vwith-pkgversion='Ubuntu 5.4.0-6ubuntu1~16.04.11'with-bugurl=file:///usr/shar
/doc/gcc-5/README.Bugsenable-languages=c,ada,c++,java,go,d,fortran,objc,obj-c++prefix=/usrprogram-suffix=-5
enable-sharedenable-linker-build-idlibexecdir=/usr/libwithout-included-gettextenable-threads=posixlibdi
=/usr/libenable-nlswith-sysroot=/enable-clocale=qnuenable-libstdcxx-debugenable-libstdcxx-time=yeswi
h-default-libstdcxx-abi=newenable-gnu-unique-objectdisable-vtable-verifyenable-libmpxenable-pluginwith-
vstem-zlibdisable-browser-pluginenable-java-awt=gtkenable-gtk-cairowith-java-home=/usr/lib/jvm/java-1.5.0-
cj-5-amd64/jreenable-java-homewith-jvm-root-dir=/usr/lib/jvm/java-1.5.0-gcj-5-amd64with-jvm-jar-dir=/usr/lib/
vm-exports/java-1.5.0-gcj-5-amd64with-arch-directory=amd64with-ecj-jar=/usr/share/java/eclipse-ecj.jarenable-
bjc-qcenable-multiarchdisable-werrorwith-arch 32=i686with-abi=m64with-multilib-list=m32,m64,mx32enab
e-multilibwith-tune=genericenable-checking=releasebuild=x86 64-linux-gnuhost=x86 64-linux-gnutarget=x86
64-linux-anu
Thread model: posix
gcc version 5.4.0 20160609 (Ubuntu 5.4.0-6ubuntul~16.04.11)

图 3.1.2.1 gcc 版本查询

如果输入命令"gcc-v"命令以后,你的终端输出类似图图 3.1.2.1 中的信息,那么说明你的电脑已经有 GCC 编译器了。最后下面的"gcc version 5.4.0"说明本机的 GCC 编译器版本为 5.4.0 的。注意观察在图 3.1.2.1 中有"Target: x86_64-linux-gnu"一行,这说明 Ubuntu 自带的 GCC 编译器是针对 X86 架构的,因此只能编译在 X86 架构 CPU 上运行的程序。如果想要编译在 ARM 上运行的程序就需要针对 ARM 架构的 GCC 编译器,也就是交叉编译器!我们进行 ARM 开发,因此肯定要安装针对 ARM 架构的 GCC 交叉编译器,当然了,这是后面的事,现在我们不用管这些,只要知道不同的目标架构,其 GCC 编译器是不同的。

如何使用 GCC 编译器来编译 main.c 文件呢? GCC 编译器是命令模式的,因此需要输入 命令来使用 gcc 编译器来编译文件,输入如下命令:

gcc main.c

上述命令的功能就是使用 gcc 编译器来编译 main.c 这个 c 文件, 过程如图 3.1.2.2 所示:

zuozhongkai@ubuntu:~/C_Program/3.1\$ gcc main.c zuozhongkai@ubuntu:~/C_Program/3.1\$ ls a.out main.c

图 3.1.2.2 编译 main.c 文件

在图 3.1.2.2 中可以看到,当编译完成以后会生成一个 a.out 文件,这个 a.out 就是编译生成的可执行文件,执行此文件看看是否和我们代码的功能一样,执行的方法很简单使用命令: "./+可执行文件",比如本例程就是命令: ./a.out,操作如图 3.1.2.3 所示:

zuozhongkai@ubuntu:~/C_Program/3.1\$ ls
a.out main.c
zuozhongkai@ubuntu:~/C_Program/3.1\$./a.out
Hello World

图 3.1.2.3 执行编译得到的文件

在图 3.1.2.3 中执行 a.out 文件以后终端输出了"Hello World!",这正是 main.c 要实现的 功能,说明我们的程序没有错误。a.out 这个文件名是 GCC 编译器自动命名的,那我们能不能 决定编译完生成的可执行文件名字呢?肯定可以的,在使用 gcc 命令的时候加上-o 来指定生成的可执行文件名字,比如编译 main.c 以后生成名为"main"的可执行文件,操作如图 3.1.2.4 所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

zuozhongkai@ubuntu:~/C_Program/3.1\$ ls
a.out main.c
zuozhongkai@ubuntu:~/C_Program/3.1\$ gcc main.c -o main //指定编译生成的可执行文件名字为main
zuozhongkai@ubuntu:~/C_Program/3.1\$ ls
a.out main main.c
zuozhongkai@ubuntu:~/C_Program/3.1\$./main
Hello World!

图 3.1.2.4 指定可执行文件名字

在图 3.1.2.4 中,我们使用 "gcc main.c –o main"来编译 main.c 文件,使用参数 "-o"来指 定编译生成的可执行文件名字,至此我们就完成 Linux 下 C 编程和编译的一整套过程。

3.2 GCC 编译器

3.2.1 gcc 命令

在上一小节我们已经使用过 GCC 编译器来编译 C 文件了,我们使用的是 gcc 命令, gcc 命令格式如下:

gcc [选项] [文件名字]

主要选项如下:

-c 只编译不链接为可执行文件,编译器将输入的.c 文件编译为.o 的目标文件。

-o<输出文件名> 用来指定编译结束以后的输出文件名,如果不使用这个选项的话 GCC 默认编译出来的可执行文件名字为 a.out。

-g 添加调试信息,如果要使用调试工具(如 GDB)的话就必须加入此选项,此选项指示编译的时候生成调试所需的符号信息。

-O 对程序进行优化编译,如果使用此选项的话整个源代码在编译、链接的的时候都会进行优化,这样产生的可执行文件执行效率就高。

-O2 比-O更幅度更大的优化,生成的可执行效率更高,但是整个编译过程会很慢。

3.2.2 编译错误警告

在 Windows 下不管我们用啥编译器,如果程序有语法错误的话编译的时候都会指示出来, 比如开发 STM32 的时候所使用的 MDK 和 IAR,我们可以根据错误信息方便的修改 bug。那 GCC 编译器有没有错误提示呢?肯定是有的,我们可以测试以下,新名为"3.2"的文件夹, 使用 vi 在文件夹"3.2"中创建一个 main.c 文件,在文件里面输入如下代码:

```
示例代码 3.2 mian.c 文件代码
#include <stdio.h>
int main(int argc, char *argv[])
{
    int a, b;
    a = 3;
    b = 4 (1)
    printf("a+b=\n", a + b);
}
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

在上述代码中有两处错误:

(1) 第一处是"b=4"少写了个一个";"号。

(2) 第二处应该是 printf("a+b=%d\n", a + b);

我们编译一下上述代码,看看 GCC 编译器是否能够检查出错误,编译结果如图 3.2.2.1 所示:

图 3.2.2.1 错误提示

从图 3.2.2.1 中可以看出有一个 error,提示在 main.c 文件的第 9 行有错误,错误类型是在 printf 之前没有";"号,这就是第一处错误,我们在"b = 4"后面加上分号,然后接着编译, 结果又提示有一个错误,如图 3.2.2.2 所示:

zuozhongkai@ubuntu:~/C_Program/3.2\$ gcc main.c -o main main.c: In function 'main': main.c:9:12: warning: too many arguments for format [-Wformat-extra-args] printf("a+b=\n", a + b);

图 3.2.2.2 错误提示

在图 3.2.2.2 中,提示我们说文件 main.c 的第 9 行: printf("a+b=\n", a + b)有 error,错误是 因为太多参数了,我们将其改为:

printf(" $a+b=%d\n$ ", a+b);

修改完成以后接着重新编译一下,结果如图 3.2.2.3 所示:

zuozhongkai@ubuntu:~/C_Program/3.2\$ gcc main.c -o main zuozhongkai@ubuntu:~/C_Program/3.2\$ ls main main.c

图 3.2.2.3 编译成功

在图 3.2.2.3 中我们编译成功,生成了可执行文件 main,执行一下 main,看看结果和我们 设计的是否一样,如图 3.2.2.4 所示:

zuozhongkai@ubuntu:~/C_Program/3.2\$./main
a+b=7

图 3.2.2.4 执行结果

可以看出,GCC 编译器和其它编译器一样,不仅能够检测出错误类型,而且标记除了错误发生在哪个文件、哪一行,方便我们去修改代码。

3.2.3 编译流程

GCC 编译器的编译流程是:预处理、汇编、编译和链接。预处理就是对程序中的宏定义 等相关的内容先进行前期的处理。汇编是先将 C 文件转换为汇编文件。当 C 文件转换为汇编 文件以后就是文件编译了,编译过程就是将 C 源文件编译成.o 结尾的目标文件。编译生成的.o 文件不能直接执行,而是需要最后的链接,如果你的工程有很多个 c 源文件的话最终就会有 很多.o 文件,将这些.o 文件链接在一起形成完整的一个可执行文件。

上一小节演示的例程都只有一个文件,而且文件非常简单,因此可以直接使用 gcc 命令 生成可执行文件,并没有先将 c 文件编译成.o 文件,然后在链接在一起。

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

正点原子

3.3 Makefile 基础

3.3.1 何为 Makefile

上一小节我们讲了如何使用 GCC 编译器在 Linux 进行 C 语言编译,通过在终端执行 gcc 命令来完成 C 文件的编译,如果我们的工程只有一两个 C 文件还好,需要输入的命令不多, 当文件有几十、上百甚至上万个的时候用终端输入 GCC 命令的方法显然是不现实的。如果我 们能够编写一个文件,这个文件描述了编译哪些源码文件、如何编译那就好了,每次需要编 译工程的时只需要使用这个文件就行了。这种问题怎么可能难倒聪明的程序员,为此提出了 一个解决大工程编译的工具: Make, 描述哪些文件需要编译、哪些需要重新编译的文件就叫 做 Makefile, Makefile 就跟脚本文件一样, Makefile 里面还可以执行系统命令。使用的时候只 需要一个 make 命令即可完成整个工程的自动编译,极大的提高了软件开发的效率。如果大家 以前一直使用 IDE 来编写 C 语言的话肯定没有听说过 Makefile 这个东西,其实这些 IDE 是有 的,只不过这些 IDE 对其进行了封装,提供给大家的是已经经过封装后的图形界面了,我们 在 IDE 中添加要编译的 C 文件, 然后点击按钮就完成了编译。在 Linux 下用的最多的是 GCC 编译器,这是个没有 UI 的编译器,因此 Makefile 就需要我们自己来编写了。作为一个专业的 程序员,是一定要懂得 Makefile 的,一是因为在 Linux 下你不得不懂 Makefile,再就是通过 Makefile 你就能了解整个工程的处理过程。

由于 Makefile 的知识比较多,完全可以单独写本书,因此本章我们只讲解 Makefile 基础 入门,如果想详细的研究 Makefile,推荐大家阅读《跟我一起写 Makefile》这份文档,文档已 经放到了开发板资料盘(A盘)\ 8_ZYNQ&FPGA 参考资料文件夹了,本章也有很多地方参考了 此文档。

3.3.2 Makefile 的引入

我们完成这样一个小工程,通过键盘输入两个整形数字,然后计算他们的和并将结果显 示在屏幕上,在这个工程中我们有 main.c、input.c 和 calcu.c 这三个 C 文件和 input.h、calcu.h 这两个头文件。其中 main.c 是主体, input.c 负责接收从键盘输入的数值, calcu.h 进行任意两 个数相加,其中 main.c 文件内容如下:

```
示例代码 3.3.2.1 main.c 文件代码
1 #include <stdio.h>
2 #include "input.h"
3 #include "calcu.h"
4
5 int main(int argc, char *argv[])
6 {
     int a, b, num;
8
9
      input int(&a, &b);
10
      num = calcu(a, b);
      printf("%d + %d = %d\r\n", a, b, num);
12 }
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

```
input.c 文件内容如下:
```

示例代码 3.3.2.2 input.c 文件代码

```
1 #include <stdio.h>
2 #include "input.h"
3
4 void input_int(int *a, int *b)
5 {
6     printf("input two num:");
7     scanf("%d %d", a, b);
8     printf("\r\n");
9 }
```

calcu.c 文件内容如下:

```
示例代码 3.3.2.3 calcu.c 文件代码
```

```
1 #include "calcu.h"
2
3 int calcu(int a, int b)
4 {
5 return (a + b);
6 }
```

文件 input.h 内容如下:

示例代码 3.3.2.4 input.h 文件代码

```
1 #ifndef _INPUT_H
2 #define _INPUT_H
3
4 void input_int(int *a, int *b);
5 #endif
文件 calcu.h 内容如下:
```

示例代码 3.3.2.5 calcu.h 文件代码

```
1 #ifndef _CALCU_H
2 #define _CALCU_H
3
4 int calcu(int a, int b);
5 #endif
```

以上就是我们这个小工程的所有源文件,我们接下来使用 3.1 节讲的方法来对其进行编译, 在终端输入如下命令:

gcc main.c calcu.c input.c -o main

上面命令的意思就是使用 gcc 编译器对 main.c、calcu.c 和 input.c 这三个文件进行编译, 编译生成的可执行文件叫做 main。编译完成以后执行 main 这个程序,测试一下软件是否工作 正常,结果如图 3.3.2.1 所示: 领航者 ZYNQ 之嵌入式 Linux 开发指南
② 正点原子
③
FF (A to be shown on the shown

图 3.3.2.1 程序测试

可以看出我们的代码按照我们所设想的工作了,使用命令"gcc main.c calcu.c input.c -o main"看起来很简单是吧,只需要一行就可以完成编译,但是我们这个工程只有三个文件啊!如果几千个文件呢?再就是如果有一个文件被修改了呢,使用上面的命令编译的时候所有的文件都会重新编译,如果工程有几万个文件(Linux 源码就有这么多文件!),想想这几万个文件编译一次所需要的时间就可怕。最好的办法肯定是哪个文件被修改了,只编译这个修改的文件即可,其它没有修改的文件就不需要再次重新编译了,为此我们改变我们的编译方法,如果第一次编译工程,我们先将工程中的文件都编译一遍,然后后面修改了哪个文件就编译哪个文件,命令如下:

gcc -c main.c

gcc -c input.c

gcc -c calcu.c

gcc main.o input.o calcu.o -o main

上述命令前三行分别是将 main.c、input.c 和 calcu.c 编译成对应的.o 文件,所以使用了"c"选项,"-c"选项我们上面说了,是只编译不链接。最后一行命令是将编译出来的所有.o 文件链接成可执行文件 main。假如我们现在修改了 calcu.c 这个文件,只需要将 caclue.c 这一 个文件重新编译成.o 文件,然后再将所有的.o 文件链接成可执行文件,只需要下面两条命令 即可:

gcc -c calcu.c

gcc main.o input.o calcu.o -o main

但是这样就又有一个问题,如果修改的文件一多,我自己可能都不记得哪个文件修改过 了,然后忘记编译,然后.....,为此我们需要这样一个工具:

1、如果工程没有编译过,那么工程中的所有.c文件都要被编译并且链接成可执行程序。

2、如果工程中只有个别 C 文件被修改了,那么只编译这些被修改的 C 文件即可。

3、如果工程的头文件被修改了,那么我们需要编译所有引用这个头文件的C文件,并且 链接成可执行文件。

很明显,能够完成这个功能的就是 Makefile 了,在工程目录下创建名为"Makefile"的文件, 文件名一定要叫做 "Makefile" !! 区分大小写的哦!如图 3.3.2.2 所示:

zuozhongka:	i@ubuntu	:~/C Prod	<pre>ram/3.3\$</pre>	ls .	
calcu.c c	alcu.h	input.c	input.h	<u>m</u> ain.c	Makefile

图 3.3.2.2 Makefile 文件

在图 3.3.2.2 中 Makefile 和 C 文件是处于同一个目录的,在 Makefile 文件中输入如下代码: 示例代码 3. 3. 2.6 Makefile 文件代码

1 main: main.o input.o calcu.o

```
2 gcc -o main main.o input.o calcu.o
```

3 main.o: main.c



原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php

4	gcc -c main.c
5	input.o: input.c
6	gcc -c input.c
7	calcu.o: calcu.c
8	gcc -c calcu.c
9	
10	clean:
11	rm *.0
12	rm main

上述代码中所有行首需要空出来的地方一定要使用"TAB"键!不要使用空格键!这 是 Makefile 的语法要求,编写好的 Makefile 如图 3.3.2.3 所示:



图 3.3.2.3 Makefile 源码

Makefile 编写好以后我们就可以使用 Make 命令来编译我们的工程了,直接在命令行中输 入"make"即可, Make 命令会在当前目录下查找是否存在"Makefile"这个文件, 如果存在 的话就会按照 Makefile 里面定义的编译方式进行编译, 如图 3.3.2.4 所示:



图 3.3.2.4 Make 编译工程

在图 3.3.2.4 中, 使用命令 "Make" 编译完成以后就会在当前工程目录下生成各种.o 和可 执行文件,说明我们编译成功了。使用 make 命令编译工程的时候可能会提示如图 3.3.2.5 所示 错误:

zuozhongkai@ubuntu:~/C_Program/3.3\$ make Makefile:2: *** missing separator。 停止。

图 3.3.2.5 Make 失败

图 3.3.2.5 中的错误来源一般有两点:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

1、Makefile 中命令缩进没有使用 TAB 键!

2、VI/VIM 编辑器使用空格代替了 TAB 键,修改文件/etc/vim/vimrc,在文件最后面加上如下所示代码:

set noexpandtab

我们修改一下 input.c 文件源码,随便加几行空行就行了,保证 input.c 被修改过即可,修 改完成以后再执行一下"make"命令重新编译一下工程,结果如图 3.3.2.6 所示:

zuozhongkai@ubuntu:~/C_Program/3.3\$ make
gcc -c input.c
gcc -o main main.o input.o calcu.o

图 3.3.2.6 重新编译工程

从图 3.3.2.6 中可以看出因为我们修改了 input.c 这个文件,所以 input.c 和最后的可执行文件 main 重新编译了,其它没有修改过的文件就没有编译。而且我们只需要输入"make"这个命令即可,非常方便,但是 Makefile 里面的代码都是什么意思呢?这就是接下来我们要讲解的。

3.4 Makefile 语法

3.4.1 Makefile 规则格式

Makefile 里面是由一系列的规则组成的,这些规则格式如下:

目标.....:依赖文件集合......

命令1 命令2

.

比如下面这条规则:

main: main.o input.o calcu.o

gcc -o main main.o input.o calcu.o

这条规则的目标是 main, main.o、input.o 和 calcu.o 是生成 main 的依赖文件,如果要更新目标 main,就必须要先更新它的所有依赖文件,如果依赖文件中的任何一个有更新,那么目标也必须更新,"更新"就是执行一遍规则中的命令列表。

命令列表中的每条命令必须以 TAB 键开始,不能使用空格!

make 命令会为 Makefile 中的每个以 TAB 开始的命令创建一个 Shell 进程去执行。

了解了 Makefile 的基本运行规则以后我们再来分析一下 3.3 节中"示例代码 3.3.2.6"中的 Makefile,代码如下:

1 main: main.o input.o calcu.o
2 gcc -o main main.o input.o calcu.o
3 main.o: main.c
4 gcc -c main.c
5 input.o: input.c
6 gcc -c input.c
7 calcu.o: calcu.c
8 gcc -c calcu.c



原子哥在线教学: www.yuanzige.com 论坛:www.openedy.com/forum.php

10 clean: 11 rm *.o

9

12 rm main

上述代码中一共有 5 条规则, 1~2 行为第一条规则, 3~4 行为第二条规则, 5~6 行为第三 条规则, 7~8 行为第四条规则, 10~12 为第五条规则, make 命令在执行这个 Makefile 的时候 其执行步骤如下:

首先更新第一条规则中的 main, 第一条规则的目标成为默认目标,只要默认目标更新了 那么就完成了 Makefile 的工作,完成了整个 Makefile 就是为了完成这个工作。在第一次编译 的时候由于 main 还不存在,因此第一条规则会执行,第一条规则依赖于文件 main.o、input.o 和 calcu.o 这个三个.o 文件,这三个.o 文件目前还都没有,因此必须先更新这三个文件。make 会查找以这三个.o 文件为目标的规则并执行。以 main.o 为例,发现更新 main.o 的是第二条规 则,因此会执行第二条规则,第二条规则里面的命令为"gcc -c main.c",这行命令很熟悉了 吧,就是不链接编译 main.c,生成 main.o,其它两个.o 文件同理。最后一个规则目标是 clean, 它没有依赖文件,因此会默认为依赖文件都是最新的,所以其对应的命令不会执行,当我们 想要执行 clean 的话可以直接使用命令"make clean",执行以后就会删除当前目录下所有的.o 文件以及 main,因此 clean 的功能就是完成工程的清理,"make clean"的执行过程如图 3.4.1.1 所示:

zuozhongkai@ubuntu:~/C_Program/3.3\$ ls
calcu.c calcu.o input.h main main.o
calcu.h input.c input.o main.c Makefile
zuozhongkai@ubuntu:~/C_Program/3.3\$ make clean
rm *.o
rm main
zuozhongkai@ubuntu:~/C_Program/3.3\$ ls
calcu.c calcu.h input.c input.h main.c Makefile

图 3.4.1.1 make clean 执行过程

从图 3.4.1.1 可以看出,当执行"make clean"命令以后,前面编译出来的.o 和 main 可执行文件都被删除掉了,也就是完成了工程清理工作。

我们在来总结一下 Make 的执行过程:

- 1) make 命令会在当前目录下查找以 Makefile(makefile 其实也可以)命名的文件。
- 2) 当找到 Makefile 文件以后就会按照 Makefile 中定义的规则去编译生成最终的目标文件。
- 当发现目标文件不存在,或者目标所依赖的文件比目标文件新(也就是最后修改时间 比目标文件晚)的话就会执行后面的命令来更新目标。

这就是 make 的执行过程, make 工具就是在 Makefile 中一层一层的查找依赖关系,并执行相应的命令。编译出最终的可执行文件。Makefile 的好处就是"自动化编译",一旦写好了 Makefile 文件,以后只需要一个 make 命令即可完成整个工程的编译,极大的提高了开发效率。把 make 和 Makefile 和做菜类似,目标都是呈现出一场盛宴,它们之间的对比关系如表 3.4.1.1 所示:

make	做菜	描述
make 工具	厨师	负责将材料加工成"美味"。
gcc 编译器	厨具	加工"美味"的工具。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

Makefile	食材	加工美味所需的原材料。
最终的可执行文	最终的菜肴	最终目的,呈现盛宴
件		

表 3.4.1.1 make 和做菜对比

总结一下, Makefile 中规则用来描述在什么情况下使用什么命令来构建一个特定的文件, 这个文件就是规则的"目标",为了生成这个"目标"而作为材料的其它文件称为"目标" 的依赖,规则的命令是用来创建或者更新目标的。

除了 Makefile 的"终极目标"所在的规则以外,其它规则的顺序在 Makefile 中是没有意义的,"终极目标"就是指在使用 make 命令的时候没有指定具体的目标时,make 默认的那个目标,它是 Makefile 文件中第一个规则的目标,如果 Makefile 中的第一个规则有多个目标,那么这些目标中的第一个目标就是 make 的"终极目标"。

3.4.2 Makefile 变量

跟 C 语言一样 Makefile 也支持变量的,先看一下前面的例子:

```
main: main.o input.o calcu.o
```

```
gcc -o main main.o input.o calcu.o
```

上述 Makefile 语句中, main.o input.o 和 calcue.o 这三个依赖文件,我们输入了两遍,我们 这个 Makefile 比较小,如果 Makefile 复杂的时候这种重复输入的工作就会非常费时间,而且 非常容易输错,为了解决这个问题,Makefile 加入了变量支持。不像 C 语言中的变量有 int、 char 等各种类型,Makefile 中的变量都是字符串!类似C语言中的宏。使用变量将上面的代码 修改,修改以后如下所示:

示例代码 3.4.2.1 Makefile 变量使用

```
1 #Makefile 变量的使用
2 objects = main.o input.o calcu.o
3 main: $(objects)
4 gcc -o main $(objects)
```

我们来分析一下"示例代码 3.4.2.1",第1行是注释,Makefile 中可以写注释,注释开头要用符号"#",不能用 C 语言中的"//"或者"/**/"!第2行我们定义了一个变量 objects,并且给这个变量进行了赋值,其值为字符串"main.o input.o calcu.o",第3和4行使用到了变量 objects,Makefile 中变量的引用方法是"\$(变量名)",比如本例中的"\$(objects)"就是使用变量 objects。

在"示例代码 3.4.2.1"中我们在定义变量 objects 的时候使用"="对其进行了赋值, Makefile 变量的赋值符还有其它两个":="和"?=",我们来看一下这三种赋值符的区别:

1、赋值符"="

使用 "=" 在给变量的赋值的时候,不一定要用已经定义好的值,也可以使用后面定义的 值,比如如下代码:

示例代码 3.4.2.1 赋值符"="使用

```
1 name = zzk
2 curname = $(name)
3 name = zuozhongkai
4
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

5 print:

6 @echo curname: \$(curname)

我们来分析一下上述代码,第1行定义了一个变量 name,变量值为"zzk",第2行也定 义了一个变量 curname, curname 的变量值引用了变量 name,按照我们 C 写语言的经验此时 curname 的值就是"zzk"。第3行将变量 name 的值改为了"zuozhongkai",第5、6行是输 出变量 curname 的值。在 Makefile 要输出一串字符的话使用 "echo",就和C语言中的"printf" 一样,第6行中的"echo"前面加了个"@"符号,因为 Make 在执行的过程中会自动输出命 令执行过程,在命令前面加上"@"的话就不会输出命令执行过程,大家可以测试一下不加 "@"的效果。使用命令"make print"来执行上述代码,结果如图 3.4.2.1:

zuozhongkai@ubuntu:~/C_Program\$ make print
curname: zuozhongkai
zuozhongkai@ubuntu:~/C_Program\$

图 3.4.2.1 make 执行结果

在图 3.4.2.1 中可以看到 curname 的值不是"zzk",竟然是"zuozhongkai",也就是变量 "name"最后一次赋值的结果,这就是赋值符"="的神奇之处!借助另外一个变量,可以 将变量的真实值推到后面去定义。也就是变量的真实值取决于它所引用的变量的最后一次有 效值。

2、赋值符":="

在"示例代码 3.4.2.1"上来测试赋值符":=",修改"示例代码 3.4.2.1"中的第2行,将 其中的"="改为":=",修改完成以后的代码如下:

```
示例代码 3.4.2.2 ":="的使用
```

1 name = zzk
2 curname := \$(name)
3 name = zuozhongkai
4
5 print:
6 @echo curname: \$(curname)

修改完成以后重新执行一下 Makefile,结果如图 3.4.2.2 所示:

zuozhongkai@ubuntu:~/C_Program\$ make print
curname: zzk
zuozhongkai@ubuntu:~/C_Program\$

图 3.4.2.2 make 执行结果

从图 3.4.2.2 中可以看到此时的 curname 是 zzk,不是 zuozhongkai 了。这是因为赋值符 ":="不会使用后面定义的变量,只能使用前面已经定义好的,这就是 "="和 ":="两个的 区别。

3、赋值符"?="

"?="是一个很有用的赋值符,比如下面这行代码:

curname ?= zuozhongkai

上述代码的意思就是,如果变量 curname 前面没有被赋值,那么此变量就是 "zuozhongkai",如果前面已经赋过值了,那么就使用前面赋的值。

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

```
4、变量追加"+="
```

Makefile 中的变量是字符串,有时候我们需要给前面已经定义好的变量添加一些字符串进去,此时就要使用到符号 "+=",比如如下所示代码:

正点原子

objects = main.o inpiut.o

objects += calcu.o

一开始变量 objects 的值为 "main.o input.o",后面我们给他追加了一个 "calcu.o",因此变量 objects 变成了 "main.o input.o calcu.o",这个就是变量的追加。

3.4.3 Makefile 模式规则

在 3.3.2 小节中我们编写了一个 Makefile 文件用来编译工程,这个 Makefile 的内容如下: 示例代码 3.4.3.1 Makefile 文件代码

```
1 main: main.o input.o calcu.o
2
      gcc -o main main.o input.o calcu.o
3 main.o: main.c
     gcc -c main.c
4
5 input.o: input.c
     gcc -c input.c
6
7 calcu.o: calcu.c
8
     gcc -c calcu.c
9
10 clean:
      rm *.o
      rm main
```

上述 Makefile 中第 3~8 行是将对应的.c 源文件编译为.o 文件,每一个 C 文件都要写一个 对应的规则,如果工程中 C 文件很多的话显然不能这么做。为此,我们可以使用 Makefile 中 的模式规则,通过模式规则我们就可以使用一条规则来将所有的.c 文件编译为对应的.o 文件。

模式规则中,至少在规则的目标定定义中要包涵 "%",否则就是一般规则,目标中的 "%"表示对文件名的匹配, "%"表示长度任意的非空字符串,比如 "%.c"就是所有的 以.c 结尾的文件,类似与通配符, a.%.c 就表示以 a.开头,以.c 结束的所有文件。

当 "%" 出现在目标中的时候,目标中 "%" 所代表的值决定了依赖中的 "%" 值,使用 方法如下:

%.o:%.c

命令

因此"示例代码 3.4.3.1"中的 Makefile 可以改为如下形式:

```
示例代码 3.4.3.2 模式规则使用
```

```
1 objects = main.o input.o calcu.o
2 main: $(objects)
3 gcc -o main $(objects)
4
5 %.o : %.c
6 #命令
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

8 clean: 9 rm *.o 10 rm main

"示例代码 3.4.3.2" 中第 5、6 这两行代码替代了"示例代码 3.4.3.1" 中的 3~8 行代码, 修改以后的 Makefile 还不能运行,因为第6行的命令我们还没写呢,第6行的命令我们需要 借助另外一种强大的变量一自动化变量。

3.4.4 Makefile 自动化变量

上面讲的模式规则中,目标和依赖都是一系列的文件,每一次对模式规则进行解析的时 候都会是不同的目标和依赖文件,而命令只有一行,如何通过一行命令来从不同的依赖文件 中生成对应的目标呢? 自动化变量就是完成这个功能的! 所谓自动化变量就是这种变量会把 模式中所定义的一系列的文件自动的挨个取出,直至所有的符合模式的文件都取完,自动化 变量只应该出现在规则的命令中,常用的自动化变量如表 3.4.4.1:

自动化变 量	描述
\$@	规则中的目标集合,在模式规则中,如果有多个目标的话,"\$@" 表示匹配模式中定义的目标集合。
\$%	当目标是函数库的时候表示规则中的目标成员名,如果目标不是函 数库文件,那么其值为空。
\$<	依赖文件集合中的第一个文件,如果依赖文件是以模式(即"%")定 义的,那么"\$<"就是符合模式的一系列的文件集合。
\$?	所有比目标新的依赖目标集合,以空格分开。
\$ ^	所有依赖文件的集合,使用空格分开,如果在依赖文件中有多个重 复的文件,"\$ [^] "会去除重复的依赖文件,值保留一份。
\$+	和"\$ [^] "类似,但是当依赖文件存在重复的话不会去除重复的依赖 文件。
\$*	这个变量表示目标模式中"%"及其之前的部分,如果目标是 test/a.test.c,目标模式为 a.%.c,那么"\$*"就是 test/a.test。

表 3.4.4.1 自动化变量

表 3.4.4.1 中的 7 个自动化变量中,常用的三种: \$@、\$<和\$^,我们使用自动化变量来完 成"示例代码 3.4.3.2"中的 Makefile, 最终的完整代码如下所示:

示例代码 3.4.4.1 自动化变量

```
1 objects = main.o input.o calcu.o
2 main: $(objects)
     gcc -o main $(objects)
4
5 %.0 : %.C
     gcc -c $<
8 clean:
```

正点原子

9 rm *.0

rm main

上述代码就是修改后的完成的 Makefile,可以看出相比 3.3.2 小节中的要精简了很多,核 心就在于第5、6这两行,第5行使用了模式规则,第6行使用了自动化变量。

3.4.5 Makefile 伪目标

Makefile 有一种特殊的目标——伪目标,一般的目标名都是要生成的文件,而伪目标不 代表真正的目标名,在执行 make 命令的时候通过指定这个伪目标来执行其所在规则的定义的 命令。

使用伪目标的主要是为了避免 Makefile 中定义的只执行命令的目标和工作目录下的实际 文件出现名字冲突,有时候我们需要编写一个规则用来执行一些命令,但是这个规则不是用 来创建文件的,比如在前面的"示例代码 3.4.4.1"中有如下代码用来完成清理工程的功能:

clean:

rm *.o

rm main

上述规则中并没有创建文件 clean 的命令,因此工作目录下永远都不会存在文件 clean, 当我们输入"make clean"以后,后面的"rm*.o"和"rm main"总是会执行。可是如果我们 "手贱",在工作目录下创建一个名为"clean"的文件,那就不一样了,当执行"make clean" 的时候,规则因为没有依赖文件,所以目标被认为是最新的,因此后面的 rm 命令也就不会执 行,我们预先设想的清理工程的功能也就无法完成。为了避免这个问题,我们可以将 clean 声 明为伪目标,声明方式如下:

.PHONY : clean

我们使用伪目标来更改"示例代码 3.4.4.1",修改完成以后如下: 示例代码 3.4.5.1 伪目标

```
1 objects = main.o input.o calcu.o
2 main: $(objects)
       gcc -o main $ (objects)
4
5 .PHONY : clean
6
7 %.0 : %.C
     qcc −c $<
8
9
10 clean:
      rm *.0
12
       rm main
```

上述代码第5行声明 clean 为伪目标,声明 clean 为伪目标以后不管当前目录下是否存在 名为"clean"的文件,输入"make clean"的话规则后面的 rm 命令都会执行。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

3.4.6 Makefile 条件判断

在 C 语言中我们通过条件判断语句来根据不同的情况来执行不同的分支, Makefile 也支 持条件判断, 语法有两种如下:

<条件关键字>

<条件为真时执行的语句>

endif

以及:

<条件关键字>

<条件为真时执行的语句>

else

<条件为假时执行的语句>

endif

其中条件关键字有 4 个: ifeq、ifneq、ifdef 和 ifndef,这四个关键字其实分为两对、ifeq 与 ifneq、ifdef 与 ifndef, 先来看一下 ifeq 和 ifneq, ifeq 用来判断是否相等, ifneq 就是判断是 否不相等, ifeq 用法如下:

ifeq (<参数 1>, <参数 2>) ifeq '<参数 1 >',' <参数 2>' ifeq "<参数 1>", "<参数 2>" ifeq "<参数 1>", '<参数 2>' ifeq '<参数 1>', "<参数 2>"

上述用法中都是用来比较"参数1"和"参数2"是否相同,如果相同则为真,"参数1" 和"参数 2"可以为函数返回值。ifneq 的用法类似,只不过 ifneq 是用来了比较"参数 1"和 "参数 2"是否不相等,如果不相等的话就为真。

ifdef 和 ifndef 的用法如下:

ifndef <变量名>

如果"变量名"的值非空,那么表示表达式为真,否则表达式为假。"变量名"同样可 以是一个函数的返回值。ifndef 用法类似,但是含义与 ifdef 相反。

3.4.7 Makefile 函数使用

Makefile 支持函数, 类似 C 语言一样, Makefile 中的函数是已经定义好的, 我们直接使 用,不支持我们自定义函数。make 所支持的函数不多,但是绝对够我们使用了,函数的用法 如下:

\$(函数名参数集合)

或者

\${函数名参数集合}

可以看出,调用函数和调用普通变量一样,使用符号"\$"来标识。参数集合是函数的多 个参数,参数之间以逗号","隔开,函数名和参数之间以"空格"分隔开,函数的调用以 "\$"开头。接下来我们介绍几个常用的函数,其它的函数大家可以参考《跟我一起写 Makefile》这份文档。

1、函数 subst



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

函数 subst 用来完成字符串替换,调用形式如下:

\$(subst <from>,<to>,<text>)

此函数的功能是将字符串<text>中的<from>内容替换为<to>,函数返回被替换以后的字符串,比如如下示例:

\$(subst zzk,ZZK,my name is zzk)

把字符串"my name is zzk"中的"zzk"替换为"ZZK",替换完成以后的字符串为"my name is ZZK"。

2、函数 patsubst

函数 patsubst 用来完成模式字符串替换,使用方法如下:

\$(patsubst <pattern>,<replacement>,<text>)

此函数查找字符串<text>中的单词是否符合模式<pattern>,如果匹配就用<replacement>来 替换掉,<pattern>可以使用包括通配符"%",表示任意长度的字符串,函数返回值就是替 换后的字符串。如果<replacement>中也包涵"%",那么<replacement>中的"%"将是 <pattern>中的那个"%"所代表的字符串,比如:

\$(patsubst %.c,%.o,a.c b.c c.c)

将字符串"a.c b.c c.c"中的所有符合"%.c"的字符串, 替换为"%.o", 替换完成以后的字符串为"a.o b.o c.o"。

3、函数 dir

函数 dir 用来获取目录,使用方法如下:

\$(dir <names...>)

此函数用来从文件名序列<names>中提取出目录部分,返回值是文件名序列<names>的目录部分,比如:

\$(dir </src/a.c>)

提取文件"/src/a.c"的目录部分,也就是"/src"。

4、函数 notdir

函数 notdir 看名字就是知道去除文件中的目录部分,也就是提取文件名,用法如下:

\$(notdir <names...>)

此函数用与从文件名序列<names>中提取出文件名非目录部分,比如:

\$(notdir </src/a.c>)

提取文件"/src/a.c"中的非目录部分,也就是文件名"a.c"。

5、函数 foreach

foreach 函数用来完成循环,用法如下:

\$(foreach <var>, <list>,<text>)

此函数的意思就是把参数<list>中的单词逐一取出来放到参数<var>中,然后再执行<text>所包含的表达式。每次<text>都会返回一个字符串,循环的过程中,<text>中所包含的每个字符串会以空格隔开,最后当整个循环结束时,<text>所返回的每个字符串所组成的整个字符串将会是函数 foreach 函数的返回值。

6、函数 wildcard



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php 通配符"%"只能用在规则中,只有在规则中它才会展开,如果在变量定义和函数使用

时,通配符不会自动展开,这个时候就要用到函数 wildcard,使用方法如下:

\$(wildcard PATTERN...)

比如:

\$(wildcard *.c)

上面的代码是用来获取当前目录下所有的.c文件,类似"%"。

关于 Makefile 的相关内容就讲解到这里,本节只是对 Makefile 做了最基本的讲解,确保 大家能够完成后续的学习,Makefile 还有大量的知识没有提到,有兴趣的可以自行参考《跟 我一起写 Makefile》这份文档来深入学习 Makefile。



正点原子

第二篇 Petalinux 使用篇

前面几章都是 Ubuntu/Linux 的基础操作,没有涉及到开发。从本篇开始我们就开始实战 操作。本篇主要讲解 Petalinux 工具的使用。首先介绍下什么是 Petalinux 工具。

"PetaLinux 工具提供在 Xilinx 处理系统上定制、构建和调配嵌入式 Linux 解决方案所需 的所有组件。该解决方案旨在提升设计生产力,可与 Xilinx 硬件设计工具配合使用,以简化 针对 Versal、Zynq UltraScale MPSoC、Zynq 7000 SoC 和 MicroBlaze 的 Linux 系统开发。"

以上是 Xilinx 官方对 Petalinux 的介绍。简单地说就是 Petalinux 工具可帮助简化基于 Linux 产品的开发。下面我们从开发环境搭建和 Petalinux 的安装和使用入手, 配以若个例程看下 Petalinux 如何简化基于 Linux 产品的开发。

需要提醒的是,学习是一步一个台阶。从《领航者 ZYNQ 之 FPGA 开发指南》到《领航 者 ZYNQ 之嵌入式 Vitis 开发指南》,我们经历了从如何使用 FPGA 到 FPGA 和 ARM 的协同 设计,虽然没有涉及到 Linux 系统,但裸机层面的使用为使用 Linux 系统打下了基础。从本篇 开始,默认读者已经学过了《领航者 ZYNQ 之 FPGA 开发指南》和《领航者 ZYNQ 之嵌入式 Vitis 开发指南》,对前面已经讲解过的概念和工具的使用将不再赘述。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

第四章 开发环境搭建

要进行 ZYNQ 开发肯定要先搭建好开发环境,我们在开始学习 STM32 的时候肯定需要安 装一堆的软件,比如MDK、IAR、串口调试助手等等,这个就是STM32的开发环境搭建。同 样的,要想在 Ubuntu 下进行 ZYNO 开发也需要安装一些软件,也就是网上说的开发环境搭 建,环境搭建好以后我们就可以进行开发了。环境搭建分为 Ubuntu 和 Windows,因为我们最 熟悉 Windows, 所以代码编写、查找资料啥的肯定是在 Windows 下进行的。但是 Linux 开发 又必须在 Ubuntu 下进行,所以还需要搭建 Ubuntu 下的开发环境。本章我们就分为 Ubuntu 和 Windows,讲解这两种操作系统下的环境搭建。



原子哥在线教学: www.yuanzige.com 论坛:www.

论坛:www.openedv.com/forum.php

4.1 Ubuntu 和 Windows 文件互传

在开发的过程中会频繁的在 Windows 和 Ubuntu 下进行文件传输,比如在 Windows 下进行代码编写,然后将编写好的代码拿到 Ubuntu 下进行编译。Windows 和 Ubuntu 下的文件互传我们需要使用 FTP 服务,设置方法如下:

1、开启 Ubuntu 下的 FTP 服务

打开 Ubuntu 的终端窗口, 然后执行如下命令来安装 FTP 服务:

sudo apt-get install vsftpd

等待软件自动安装,安装完成以后使用如下 VI 命令打开/etc/vsftpd.conf,命令如下:

sudo vi /etc/vsftpd.conf

打开以后 vsftpd.conf 文件以后找到如下两行:

 $local_enable=YES$

write_enable=YES

确保上面两行前面没有"#",有的话就取消掉,完成以后如图 4.1.1 所示:



图 4.1.1 vsftpd. conf 修改

修改完 vsftpd.conf 以后保存退出,使用如下命令重启 FTP 服务:

sudo /etc/init.d/vsftpd restart

2、Windows下 FTP 客户端安装

Windows 下 FTP 客户端我们使用 FileZilla,这是个免费的 FTP 客户端软件,可以在 FileZilla 官网下载,下载地址如下: <u>https://www.filezilla.cn/download</u>,下载界面如图 4.1.2 所示:



我们已经下载好 FileZilla 并放到开发板光盘中了,路径为:开发板资料盘(A 盘)/6_常用软件/FileZilla_3.39.0_win64-setup_bundled.exe,双击安装即可。安装完成以后找到安装目录,找到图标,然后发送图标快捷方式到桌面,完成以后如图 4.1.3 所示:

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

正点原子



图 4.1.3 FileZilla 图标

打开 FileZilla 软件,界面如图 4.1.4 所示:

E FileZilla										-		×
文件(F) 编辑(E)	文件(F) 編辑(E) 查看(V) 传输(T) 服务器(S) 书签(B) 帮助(H)											
₩ • 111	€ № 8	1 🕄 🗐 🗐	A 🖉 🦚									
主机(H):	用户名(U)	:	密码(W):		端口(P):		快速连接	€(Q) ▼				
												^
												\sim
本地站点: C:\Use	rs\zuozh\			~	远程站点;							~
	a zuozh			^								
<u> </u> <u>⊕</u> - <u></u> \	Windows											
⊞~∽ D:	「作)											
⊞ - ⇒ F: (≦	LII) L活)											
🖶 🥪 G: (1	仓库)											
🗎 🚽 🖶 H: (i	新加卷)											
	buntu)			~								
文件名	文件大小 文件类型	最近修改		^	文件名	文件大	小 文件类型	最近修改	权限	所有者/	组	
<u> </u>												
android	文件夹	2017-12-13					15	没有连接到任何	可服务器			
idlerc	文件夹	2016-12-19										
oracle ire	文件夹	2016-09-07										
.p2	文件夹	2016-12-19										
stm32cub	文件夹	2016-09-07										
📜 .stmcufinder	文件夹	2017-05-17										
👆 3D Objects	文件夹	2016-10-26										
📜 AppData	文件夹	2018-10-20										
Applicatio	文件夹			~								
18 个文件 和 39 个	1日录。大小尽计:27,550),389字节			禾连接。							
服务器/本地文件	方向 远程文	(件	大小 优先级	状态	ž							
列队的文件 传输	命牛败 成功的传输											
VINCHOX TT 194												
									(1) 队列	:空		••

图 4.1.4 FileZilla 软件界面

3、FileZilla 软件设置

Ubuntu 作为 FTP 服务器, FileZilla 作为 FTP 客户端, 客户端肯定要连接到服务器上, 打 开站点管理器,点击:文件->站点管理器,打开以后如图 4.1.5 所示:

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

正点原子

站点管理器			×
选择项(S):		常规 高级 传输设置 字符集	
我的站京		协议(T): FTP - 文件传输协议	\sim
		主机(H): 端口(P):	
		加密(E): 如果可用 , 使用显式的 FTP over TLS	\sim
		登录类型(L): 正常	\sim
		用户(U):	
		密码(W):	
		背景颜色(B) 无 ~	
		注释(M):	
新站点(N)	新文件夹(F)		^
新建书签(M)	重命名(R)		
删除(D)	复制(I)		\sim
		连接(C) 确定(O) 取消	

图 4.1.5 站点管理器

点击图 4.1.5 中的"新站点(N)"按钮来创建站点,新建站点以后就会在"我的站点"下 出现新建的这个站点,站点的名称可以自行修改,比如我将新的站点命名为"Ubuntu"如图 4.1.6 所示:

选择项(S):
<mark>●●</mark> 我的站点 □□□□ 및 Ubuntu

图 4.1.6 新建站点

选中新创建的"Ubuntu"站点,然后对站点的"常规"进行设置,设置如图 4.1.7 所示:



正点原子

图 4.1.7 站点设置

上图中主机 ip 地址可以在 Ubuntu 系统中的设置界面查询,如下图所示:

Q	、 设置	·····································	
٩	背景	+	
Q	Dock	取消(C) 有线 应用(A)	
Ļ	通知	IF#细信息 身份 IPv4 IPv6 安全	
۹	搜索	链路速度 1000 Mb/秒 +	
٩	区域和语言	IPv6地址 fe80::bb2b:2d44:963c:ae63	
0	通用辅助功能	硬件地址 00:0C:29:7A:26:76	
₽s	在线帐户	默认路由 192.168.2.1 DNS 192.168.2.1	
4	隐私	☑ 自动连接(A)	
<	共享	☑ 对其他用户可用(O)	
40	声音	Restrict background data usage Appropriate for connections that have data charges or limits.	
Ge	电源		
Ō5	网络	忘记连接配置	
Ē			

图 4.1.8 主机 IP 地址

按照图 4.1.7 中设置好以后,点击"连接"按钮,第一次连接可能会弹出"不安全的 FTP 连接",点击确定即可。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

连接成功以后如图 4.1.9 所示,其中左边就是 Windows 文件目录,右边是 Ubuntu 文件目录,默认进入用户根目录下(比如我电脑的"/home/sqd")。但是注意观察在图 4.1.9 中Ubuntu 文件目录下的中文目录都是乱码的,这是因为编码方式没有选对,先断开连接,点击:服务器(S)->断开连接,然后打开站点管理器,选中要设置的站点"Ubuntu",选择"字符集",设置如图 4.1.10 所示。

🔁 Ubuntu - sqd@192.16	8.2.131 - FileZilla								-		\times
文件(F) 编辑(E) 查看(V)	传输(T) 服务器(S)	书签(B) 帮助(H) 有新版	版本! (N)								
出 - 同一二社	🖸 🔁 🎼 😆 🕵	1, 🗉 🔍 🔗 🦚									
主机(H):	用户名(U):	密码(W):	端口(P):	快速	连接(Q) ▼						
状态: 读取目录列表											^
状态: 计算服务器时差…	状态: 计算服务器时差										
状态: Timezone offset of	server is 0 seconds.										- 11
状态:列出"/home/sqd"的目录成功 Windows下的文件目录 Ubuntu下的文件目录									~		
本地站点: C:\Users\Admin	nistrator\			远程站点: //	nome/sqd						Ý
🕀 🐍 Admin	nistrator										
🗈 🔤 All Use	ers			<u>⊨</u> . hc	ome						
⊞ <mark>.</mark> Defau	lt			±	sqd						
Defau	lt User										
😟 🔤 Public											
XmpCach	ie										
E. (4(1+)			Ļ								
	7件十小 文件米刑	是近後改		☆# 2	文件十小	立件举刑	是沂修政	权限	所有考/织		
		ARCEL IS RX		XIHA	XIHAD	XIHXE	ARCZ IN EX	TXPK	///H/H//2E		
altera cht/le	文件本	2022年7日23日				立件主	2010年6日13	dowyr-yr-y	1000 1000		
lisse	文件夹	2022年5月10日		於明濟		文件夹	2023年3月7日	drwxr-xr-x	1000 1000		
profile	文件夹	2022年3月28日		妯℃潜		文件夹	2023年3月7日	drwxr-xr-x	1000 1000		
ssh	文件夹	2022年10月1日		温燥流		文件夹	2023年3月7日	drwxr-xr-x	1000 1000		
vscode	文件夹	2022年9月27日		遥唾		文件夹	2023年3月7日	drwxr-xr-x	1000 1000		
Xilinx	文件夹	2022年6月13日		2 創得		文件夹	2023年3月7日	drwxr-xr-x	1000 1000		
3D Objects	文件夹	2022年3月12日		線剧墖		文件夹	2023年3月7日	drwxr-xr-x	1000 1000		
AppData	文件夹	2022年3月12日		謳囨。		文件夹	2023年3月7日	drwxr-xr-x	1000 1000		
Application Data	文件夹			開充擎		文件夹	2023年3月7日	drwxr-xr-x	1000 1000		
2 Contacts	文件夹	2022年3月12日		🗋 exampl	8,980	DESKTOP	2023年3月7日	-rw-rr	1000 1000		
Cookies	文件夹										
Desktop	文件夹	2023年3月6日 星	ļ								
17 个文件 和 32 个目录。大	小总计: 64,664,452 字节	5		1 个文件 和 9) 个目录。大小	总计: 8,980 字	ŧ				
服务器/本地文件	方向 沅程文	件	大小 优先级 光才	5							
AND ANY TODACI I	7567 201EX			~							
列头的文件 传输失败 成功的传输											
								۵ 🔅	队列: 空	4	

图 4.1.9 连接成功

原子哥在线教学:	www.yuanzige.com	论坛:www.openedv.com/forum.php
	站点管理器	×
	选择项(S):	常规 高级 传输设置 字符集
	我的站点 Ubuntu	服务器使用以下的字符集编码来处理文件名:
		○ 目 动 检测(A) 如果服务器支持就使用 UTF-8,否则使用本 地编码。
		 ● 强制 UTF-8
	选择"强制UTF-8	· 编码(E):
		使用烘浸的字灯集可能导动文件名息云不正
	新站点(N) 新文件夹(F)	
	新建书签(M) 重命名(R)	
	删除(D) 复制(I)	
		连接(C) 确定(O) 取消
	图	4.1.10设置字符集

🙋 正点原子

按照图 4.1.10 设置好字符集以后重新连接到 FTP 服务器上,重新链接到 FTP 服务器以后 Ubuntu 下的文件目录中文显示就正常了,如图 4.1.11 所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

🔁 Ubuntu - sqd@192	2.168.2.131 - FileZilla									-		×
文件(F) 编辑(E) 查看	文件(F) 編環(E) 查看(V) 传输(T) 服务器(S) 书签(B) 帮助(H) 有新版本! (N)											
紐・■□□□ # 2 排 3 1, 1) 〒 点 9 め												
主机(H):	用户名(U):	密码(W):	端口((P):	快速	连接(Q) 🔻						
状态: 读取目录列表												^
状态: 计算服务器时差												
状态: Timezone offset	of server is 0 seconds.											
状态: 列出"/home/sqd	"的目录成功							<u> </u>				~
本地站点: C:\Users\Ad	ministrator			~	远程站点: /h	ome/sqd						~
	ministrator			^	B- ? /							-
All	Users					me						
	fault				÷	sqd						
De	fault User											
	blic											
. Windo	ows											
·····································	ache (쿱)											
□ □ □ E: (软件)												
)			~								
文件名	文件大小 文件类型	最近修改		^	文件名	文件大小	文件类型	最近修改	权限	所有者/组		_
<mark>.</mark>					. .							
📙 .altera.sbt4e	文件夹	2022年7月23日			vmware		文件夹	2019年6月13	drwxr-xr-x	1000 1000		
. jssc	文件夹	2022年5月10日			下载		文件夹	2023年3月7日…	drwxr-xr-x	1000 1000		
.profile	文件夹	2022年3月28日			公共的		文件夹	2023年3月7日	drwxr-xr-x	1000 1000		
.ssh	文件夹	2022年10月1日			图片		文件夹	2023年3月7日	drwxr-xr-x	1000 1000		
.vscode	文件夹	2022年9月27日			文档		文件夹	2023年3月7日	drwxr-xr-x	1000 1000		
Xilinx	文件夹	2022年6月13日			桌面		文件夹	2023年3月7日	drwxr-xr-x	1000 1000		
3D Objects	文件夹	2022年3月12日			模板		文件夹	2023年3月7日	drwxr-xr-x	1000 1000		
AppData	又件失	2022年3月12日			例刻		又件类	2023年3月7日	drwxr-xr-x	1000 1000		
Application Data	又件关	2022年2日12日				0.000		2023年3月7日	drwxr-xr-x	1000 1000		
Cookies	21+天 文仕平	202243/121			exampi	0,900	DESKTOP	. 202343月7日	-100-11	1000 1000		
	文件夹	2023年3月6日 星										
				~								
17 个文件 和 32 个目录。	大小总计: 64,664,452 字节				1 个文件 和 9	个目录。大小	息计: 8,980 字	-17 -				
服务器/本地文件	方向 远程文件	4	大小 优先级	状态								
列队的文件 传输失败	成功的传输											
									00	이 귀		
									Û (J	队列:学		••

图 4.1.11 Ubuntu 下文件目录中文显示正常

如果要将 Windows 下的文件或文件夹拷贝到 Ubuntu 中,只需要在图 4.1.11 中左侧的 Windows 区域选中要拷贝的文件或者文件夹,然后直接拖到右侧的 Ubuntu 中指定的目录即可。将 Ubuntu 中的文件或者文件夹拷贝到 Windows 中也是直接拖放。

4.2 Ubuntu 和 Windows 文件本地共享

对于在Windows上安装虚拟机软件,在虚拟机软件中运行Ubuntu系统这类场景,Ubuntu和Windows文件互传可以使用本地共享的方式。这种共享的方式极大的免除了不同系统文件之间的文件复制和磁盘空间的双重占用。下面笔者将介绍如何使用Vmware虚拟机来实现Ubuntu和Windows文件之间的共享。

注: 请先完成 1.4 小节安装 Vmware Tools 的内容,因为本小节是基于 1.4 小节的。

首先在 Vmware 中启动 Ubuntu 系统, 然后在 Vmware 的菜单栏中, 选择"设置(S)", 如下图所示:

正点原子 领航者 ZYNQ 之嵌入式 Linux 开发指南 原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php 文件(F) 编辑(E) 查看(V) 虚拟机(M) 选项卡(T) 帮助(H) 귝 -Ð ① 电源(P) Þ 隼 ③ 可移动设备(D) Þ ○ 在此处键入内容进行搜索 暂停(U) Ctrl+Shift+P □ 🖵 我的计算机 母 发送 Ctrl+Alt+Del(E) 🕞 Ubuntu 64 位 抓取输入内容(I) Ctrl+G □ 共享的虚拟机 Ⅰ 使照(N) 捕获屏幕 (C) Ctrl+Alt+PrtScn ⊘ 管理(M) Þ 重新安装 Mware Tools(T)...

Ctrl+D

[] 设置(S)...

在弹出的菜单中,选择"选项",在该界面中,单击"共享文件夹",在右边界面中选择"总是启用(E)",如下图所示:

虚拟机设置	/	×
硬件 选项		
设置 □ 常规 ▶ 电源 ④ 共享文件共 ④ 自动保护 合 客户机隔离 ☞ 访问控制 ☞ VMware Tools 班 VNC 连接 □ Unity 5 自动登录 ☞ 高級	摘要 Ubuntu 64 位 已禁用 已禁用 未加密 关闭时间同步 已禁用 不受支持 默认/默认	文件夹共享 ▲ 共享文件夹会将您的文件显示给虚拟机中 的程序。这可能为您的计算机和数据带来 风险。请仅在您信任虚拟机使用您的数据 时启用共享文件夹。 ④ 已禁用(D) ④ 总是启用(E) ④ 在下次关机或挂起前一直启用(U) 文件夹(F) 名称 主机路径 添加(A) 移除(R) 属性(P)

图 4.2.2 启用共享

现在我们需要添加共享的文件夹了。这里我们在 Windows 的 G 盘里新建一个名为 "share" 的文件夹,如图 4.2.3 所示。这个文件夹就是以后我们用来在 Windows 和 Ubuntu 系统之间共 享文件的地方。当然了,任何一个文件夹都是可以的,读者可以按自己的实际情况选择。

图 4.2.1 选则"设置(S)"



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

👝 🛃 📙 🖛	管理 tmp (G:)		
文件 主页 共享 查看	驱动器工具		
← → • ↑ 👝 > 此电脑 > t	mp (G:)		
share ^		修改日期	类型
OpeDrive	tk	2021/5/6 10:13	文件夹
	FPGA	2021/7/5 16:59	文件夹
💻 此电脑	gitoffice	2021/7/12 17:14	文件夹
	📙 hls	2021/7/5 17:00	文件夹
	📙 jianguoyun	2021/3/2 12:36	文件夹
	npsoc	2021/10/27 10:08	文件夹
	📙 pcie	2021/8/16 19:18	文件夹
🖆 文档	pynq	2021/10/9 13:48	文件夹
🔶 下载	pynq_old	2021/4/23 16:36	文件夹
♪ 音乐	share	2021/10/26 14:42	文件夹
📃 桌面	share_for_office	2021/10/23 17:10	文件夹
🏪 本地磁盘 (C:)	📑 test	2021/9/27 19:14	文件夹
一 软件 (D·)	🔄 tmp	2021/10/25 12:29	文件夹
	🔄 vivado	2021/7/5 20:25	文件夹
work (E:)	📙 work	2021/9/9 17:32	文件夹
doc (F:)	🔄 zhanjiam01	2021/3/2 12:41	文件夹
👝 tmp (G:)	zynq	2021/9/30 15:35	文件夹

图 4.2.3 新建用于共享的文件夹"share"

现在我们单击图中的"添加(A)…"按钮,弹出下图所示界面



图 4.2.4 添加共享的文件夹

直接点击"下一步(N)->",进入下图所示界面:

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

添加共享文件夹向导	\times
命名共享文件夹 如何命名此共享文件夹?	
主机路径(H) G:\share 2	
名称(A) share 3	
4	
<上一步(B) 下一步(N) > 取消	

正点原子

图 4.2.5 完成添加

点击"浏览(R)…",选择我们刚才新建的 share 文件夹,选择好以后点击确定。上图的 名称此处保持默认 share,按个人喜好可以修改。点击"下一步(N)->",进入下图所示界面:

添加共享文件夹向导		23
指定共享文件夹属性 指定此共享文件夹的范围。		
其他属性 ☑ 启用此共享(E)		
□□ 只读(R)		
[< 上一步(B) 完成	取消

图 4.2.6 点击"完成"按钮

默认勾选"启用此共享",如果不想该文件夹内的内容被修改,可以勾选只读,此处我 们不勾选"只读",点击"完成"按钮。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

现在如果我们需要向 Ubuntu 系统传文件,就可以直接将该文件复制到 share 目录就可以 了。以后我们需要向 Ubuntu 系统传递的文件就都放在 share 文件夹下。那 Ubuntu 如何访问 share 文件夹呢?

该文件夹在 Ubuntu 系统中对应的是/mnt/hgfs/share/目录,我们在终端中输入命令:

ll /mnt/hgfs/share/

如下图所示:可以看到该文件夹为空。如果需要从 Ubuntu 系统向 Windows 传递文件,可以用 cp 命令或 mv 命令文件到该目录。

sqd@sqd-vir 台田島 45	tu	ial-ma	achine	e:~\$ 1	ll /r	nnt/h	ngfs/sl	nare/
心用里 13					_			_
drwxrwxrwx	1	root	root	4096	3月	7	17:22	. /
dr-xr-xr-x	1	root	root	4192	3月	7	17:23	/
drwxrwxrwx	1	root	root	4096	2月	21	10:44	linux_tool/
drwxrwxrwx	1	root	root	0	3月	2	15:04	设备树文件/

图 4.2.7 共享文件夹对应的 Ubuntu 目录/mnt/hgfs/share/

别忘了,在 1.4 小节安装 Vmware Tools 中我们还启用了拖曳和复制功能,可以用鼠标直接将文件或文件夹从 Windows 拖到 Ubuntu 或从 Ubuntu 拖到 Windows 中。复制功能则可用于在 Ubuntu 和 Windows 中共享粘贴板。

4.3 Ubuntu 系统搭建 tftp 服务器

TFTP 作为一种简单的文件传输协议,在嵌入式开发中会经常使用到,而且后面我们在安装 Petalinux 工具时也会提示需要 tftp 服务,所以我们需要在 Ubuntu 上搭建 TFTP 服务器。需要安装 tftp-hpa (客户端软件包,如果不用可不装)和 tftpd-hpa 软件包,命令如下:

sudo apt install tftp-hpa tftpd-hpa

TFTP 需要一个文件夹来存放文件,我们在根目录下新建一个/tftpboot 目录做为 TFTP 文件存储目录,之所以使用该目录是因为后面使用的 Petalinux 工具默认使用该目录,省得我们每次建 Petalinux 工程的时候手动修改。创建/tftpboot 目录命令如下:

sudo mkdir -p /tftpboot

sudo chmod 777 /tftpboot

这样笔者就在电脑上创建了一个名为tftpboot的目录(文件夹),路径为/tftpboot。需要注意 的是我们要给 tftpboot 文件夹权限,否则的话后面在使用过程中会遇到问题,所以使用了 chmod 777 命令。

最后配置 tftp。使用 chmod 666 命令将/etc/default/tftpd-hpa 文件属性改为可读可写,打开 /etc/default/tftpd-hpa 文件,将其内容修改如下:

示例代码/etc/default/tftpd-hpa 文件内容

```
1 # /etc/default/tftpd-hpa
2
3 TFTP_USERNAME="tftp"
```

```
4 TFTP_DIRECTORY="/tftpboot"
```

```
5 TFTP_ADDRESS=":69"
```

```
6 TFTP_OPTIONS="-1 -c -s"
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

TFTP_DIRECTORY 就是我们上面创建的/tftpboot 文件夹目录,以后我们就将所有需要通过 TFTP 传输的文件都放到该文件夹里面。

最后输入如下命令,重启tftp服务器:

sudo service tftpd-hpa restart

至此,tftp服务器就已经搭建好了。

4.4 Ubuntu 下 NFS 和 SSH 服务开启

4.4.1 NFS 服务开启

后面进行 Linux 驱动开发的时候需要 NFS 启动,因此要先安装并开启 Ubuntu 中的 NFS 服务,使用如下命令安装 NFS 服务:

sudo apt install nfs-kernel-server

等待安装完成。安装完成以后在用户根目录下创建一个名为"workspace/nfs"的文件夹, 命令如下:

cd

mkdir -p workspace/nfs cd workspace/nfs pwd

结果如下图所示:

sqd@sqd-virtual-machine:~\$ cd
sqd@sqd-virtual-machine:~\$ mkdir -p workspace/nfs
sqd@sqd-virtual-machine:~\$ cd workspace/nfs
sqd@sqd-virtual-machine:~/workspace/nfs\$ pwd
/home/sqd/workspace/nfs

图 4.4.1nfs 目录

以后所有需要使用 nfs 的东西都放到这个"nfs"文件夹里面。

上面创建的 nfs 文件夹供 nfs 服务器使用,以后我们可以在开发板上通过网络文件系统来 访问 nfs 文件夹。

使用前需要先配置 nfs。NFS 允许挂载的目录及权限在文件/etc/exports 中进行定义,使用 如下命令打开 nfs 配置文件/etc/exports:

sudo vi /etc/exports

打开/etc/exports 以后在后面添加如下所示内容:

/home/sqd/workspace/nfs *(rw,sync,no_root_squash)

/home/sqd/workspace/nfs 是要刚才创建的 nfs 的目录,也就是上面命令 pwd 输出的内容, *代表允许所有的网络段访问,rw 是可读写权限,sync 是文件同步写入存储器, no_root_squash 是 nfs 客户端分享目录使用者的权限。如果客户端使用的是 root 用户,那么对 于该共享目录而言,该客户端就具有 root 权限。

添加完成以后的/etc/exports 如下图所示:



原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php



图 4.4.2 修改文件/etc/exports

重启 NFS 服务,使用如下命令:

sudo systemctl start nfs-kernel-server.service

此时可以运行以下命令来显示共享的目录:

showmount -e

在 nfs 运行的过程中,修改了/etc/exports 配置文件,可以使用 exportfs 命令使改动生效, 具体命令:

sudo exportfs -rv

4.4.2 SSH 服务开启

开启 Ubuntu 的 SSH 服务以后我们就可以在 Windwos 下使用终端软件登陆到 Ubuntu,比如使用 MobaXterm, Ubuntu 下使用如下命令开启 SSH 服务:

sudo apt install openssh-server

上述命令安装 ssh 服务, ssh 的配置文件为/etc/ssh/sshd_config, 使用默认配置即可。

4.5 Visual Studio Code 软件的安装和使用

4.5.1 Visual Studio Code 的安装

Visual Stuio Code 本教程简称为 VSCode, VSCode 是微软推出的一款免费编辑器。 VSCode 有 Windows、Linux 和 macOS 三个版本,是一个跨平台的编辑器。VSCode 下载地址 是: <u>https://code.visualstudio.com/</u>,下载界面如下图所示:
领航者 ZYNQ)之嵌)	入式 Li	inux 刃	F发指南		② 正点原子			
原子哥在线教学:	www.y	uanzige	e.com	论坛:www.op	enedv.com/for	um.php			
🗙 Visual Studio	Code Docs	Updates	Blog API Ex	tensions FAQ Learn	م se	earch Docs J Download			
			is now available	Read about the new features	and fixes from January.				
Code editing. L At h TW Redefined. Free Built on open source. Runs everywhere.									
2、选择要下载 的版本 Download Stab	for Windows le Build	~		Eric Amodio Install C/C++ 0.240 \$\Phi23M \times 3.5\$ C/C++ Install Install Microsoft Install Install ESLint 1.9.0 \$\Phi219M \times 4.5\$ Integrates ESLint JavaScript into VS \$\Phi219M \times 4.5\$	45 Image: productSub 46 Image: provestiteSpecific 47 Image: provestiteSpecific 48 Image: provestiteSpecific 49 Image: provestiteSpecific 50 Image: provestiteSpecific 51 Image: provestiteSpecific	TrackingException ingException temAccess • operty) Navigator.serviceWorke.O			
macOS	Universal	itable Insiders	s to the second se	Dirk Baeumer Install Debugger for Ch 4.11.6 @20.6M * 4 Debug your JavaScript code in the C Microsoft Install	52 G storesiteSpecificT 53 G storeWebWideTracki 54] @ userAgent 55 } @ vendor	rackingException ngException			
Windows x64 Linux x64	User Installer .deb .rpm	π π π π π π π π		Language Supp 0470 €71874 *45 Java Linting, Intellisense, formatting, Red Hat Install Viscode-icons 880 €71724 *5 Icons for Visual Studio Code VSCode Icons Team Install	57 function registerValid5% 58 navigator.serviceWorke 59 .register(swUrl) 60 .then(registration = TERMINAL	((swUrl, config) (m > { e • • + □ @ ^ ×			
	Other downloads or open on web		දිදිදි Prmaster C	Vue tooling for VS Code Pine Wu C 4 Table 2000 Pine Wu C 4 for VSual Studio Code (powered Microsoft C 0 0 & 0	You can now view create-react- Local: http://loc On Your Network: http://lo. Note that the development buil Ln43.Col1	app in the browser. alhost:3000/ 211.55.3:3000/ d is not optimized. 9 Spaces 2 UTF-8 UF javaScript 😁 🌲			

图 4.5.1 VSCode 下载界面

在上图中下载自己想要的版本,本教程需要 Windows 和 Linux 这两个版本,所以下载这两个即可,我们已经下载好并放入了开发板光盘中,路径为:开发板资料盘(A 盘)/6_常用软件/Visual Studio Code。

1、Windows 版本安装

Windows 版本的安装和容易,和其他 Windows 一样,双击.exe 安装包,然后一路"下一步"即可,安装完成以后在桌面上就会有 VSCode 的图标,如下图所示:



图 4.5.2 VSCode 图标

双击上图所以的图标打开 VSCode, 默认界面如下图所示:



_	= Sattinas 🗙			о m
D	= settings A			л ш
ρ				296 Settings Found
00	User Settings			
8	Commonly Used	Commonly Used		
B	 Workbench 	Files: Auto Save		
	Window	Controls auto save of dirty files. Rea	d more about autosave here.	
	Features	off		
	Application			
	Extensions	Files: Auto Save Delay		
		Controls the delay in ms after which	a dirty file is saved automatically. Only applies when Files:	Auto Save is set to afterDela
		1000		
		Editor: Font Size		
		Controls the font size in pixels.		
		14		
		Editor: Font Family		
		Controls the font family.		
		Consolas, 'Courier New', monospac	re	
		Editor: Tab Size		
		The number of spaces a tab is equal on.	l to. This setting is overridden based on the file contents w	nen Editor: Detect Indentation

图 4.5.3 VSCode 默认界面

2、Linux 版本安装

我们有时候也需要在 Ubuntu 下阅读代码,所以还需要在 Ubuntu 下安装 VSCode。Linux 下的 VSCode 安装包我们也放到了开发板光盘中,将开发板光盘中的.deb 软件包拷贝到 Ubuntu 系统中,然后使用如下命令安装:

sudo dpkg -i code_1.32.3-1552606978_amd64.deb 等待安装完成,如下图所示:

```
sqd@sqd-virtual-machine:~/linux-tool$ ls
code_1.32.3-1552606978_amd64.deb
sqd@sqd-virtual-machine:~/linux-tool$ sudo dpkg -i code_1.32.3-1552606978_amd64.deb
[sudo] sqd 的密码:
正在选中未选择的软件包 code。
(正在读取数据库 ... 系统当前共安装有 131502 个文件和目录。)
正准备解包 code_1.32.3-1552606978_amd64.deb ...
正在解包 code (1.32.3-1552606978) ...
正在设置 code (1.32.3-1552606978) ...
gpg: WARNING: unsafe ownership on homedir '/home/sqd/.gnupg'
正在处理用于 gnome-menus (3.13.3-11ubuntu1.1) 的触发器 ...
正在处理用于 desktop-file-utils (0.23-1ubuntu3.18.04.2) 的触发器 ....
正在处理用于 mime-support (3.60ubuntu1) 的触发器 ...
```

图 4.5.4 VSCode 安装过程

安装完成以后搜索"code"就可以找到,如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php Q codel

图 4.5.5 VSCode 安装完成

Visual Studio ...

每次打开 VSCode 都要搜索,太麻烦了,我们可以将图标添加到 Ubuntu 桌面上,安装的 所有软件图标都在目录/usr/share/applications 中,如下图所示:



图 4.5.6 软件图标

在上图中找到 Visual Studio Code 的图标,然后点击鼠标右键,选择复制到->桌面,如下 图所示:

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

正点原子

取消	肖(C)	选择复制的目标位置	C	ス 选择(S)
Ø	最近使用	 ▲ sqd ▲ 桌面 		
企	主目录	名称	▲ 大小	修改日期
	桌面			
H	视频			
٥	图片			
۵	文档			
∻	下载			
99	音乐			
Î	回收站			
+	其他位置			

图 4.5.7 复制图标到桌面

按照上图所示方法将 VSCode 图标复制到桌面,以后直接双击图标即可打开 VSCode,如 果出现未信任的应用程序启动器,点击"Trust and Lanch",Ubuntu下的 VSCode 打开以后如 下图所示:



图 4.5.8 Linux 下的 VSCode

可以看出Linux下的VSCode和Windows下的基本是一样的,所以使用方法也是一样的。



原子哥在线教学:www.yuanzige.com

论坛:www.openedv.com/forum.php

4.5.2 Visual Studio Code 插件的安装

VSCode 支持多种语言,比如 C/C++、Python、C#等等,本教程我们主要用来编写 C/C++ 程序的,所以需要安装 C/C++的扩展包,扩展包安装很简单,如下图所示:



图 4.5.9 VSCode 插件安装

我们需要安装的插件有下面几个:

- 1)、C/C++,这个肯定是必须的。
- 2)、C/C++ Snippets,即C/C++重用代码块。
- 3)、C/C++ Advanced Lint,即C/C++静态检测。
- 4)、Code Runner,即代码运行。
- 5)、Include AutoComplete,即自动头文件包含。
- 6)、Rainbow Brackets,彩虹花括号,有助于阅读代码。
- 7)、One Dark Pro, VSCode 的主题。
- 8)、GBKtoUTF8,将GBK转换为UTF8。
- 9)、ARM,即支持 ARM 汇编语法高亮显示。
- 10)、Chinese(Simplified),即中文环境。
- 11)、vscode-icons, VSCode 图标插件,主要是资源管理器下各个文件夹的图标。
- 12)、compareit,比较插件,可以用于比较两个文件的差异。
- 13)、DeviceTree,设备树语法插件。

如果要查看已经安装好的插件,可以按照下图所示方法查看:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

শ	File	e Edit	Selection	View		Debug	Terminal	Help		Visual Studio Code					
ß		EXTENS	ions: Marke	TPLACE		×					8234	****			
~		@insta	lled				Show	Installed	Extensions	,		दस्तराश्रात	14		
		C/C++	0.22.1				Show	Outdated	Extensions						
v		C/C++ Microso	IntelliSense, ft	debug	ging, a	and code l	Show	Enabled I	Extensions						
8		C/C++	Advanced	Lint 1.2.	.2		Show	Disabled	Extensions						
		An adva	anced, mode	ern, stat	ic ana	lysis exten	Show	Built-in E	xtensions						
9			Senden	0.10			Show	Recomme		S					
¢		Code sr	nippets for C	.0.13 C/C++											
		Harsh					Sort B	y: Install (Iount						
		Code R	unner 0.9.7 C++ Java J	S. PHP.	Pvthor	n Perl. Rul	Sort B	y: Rating							
		Jun Han		-,,		.,		y. Name							
		GBKtol	JTF8 0.0.2				Check	for Exten	sion Updates						
		a vscod bukas	e extension	to con	/ert gb	ok to utf8	Disabl	le Auto U	pdating Extensio	ns mmands	Ctd y Shift				
		Include	Autocomp	olete 0.0).4			Trom VSI.	A		cui + siin				
		Autoco	mpletion fo	r C++ ir	nclude	s	Disabl	e All Insta	alled Extensions)pen File	Ctrl + O				
		One Da	rk Pro 2.21	o			Enable	e All Exter		n Folder	Ctrl + K	Ctrl + O			
		Atom's binaryify	iconic One l	Dark th	eme fo	or Visual S	tud ¢			Open Recent	Ctrl + R				
		Rainbo	w Brackets	0.0.6											
مله		A rainb 2gua	ow brackets	extensi	ion for	VS Code.	ø								
*															
Ø 0	A 0									Ln 1, Col	1 Tab Size: 4	UTF-8 L	F JSOI	N 😬	

图 4.5.10 显示已安装的插件

安装好插件以后就可以进行代码编辑了,截至目前,VSCode界面都是英文环境,我们已 经安装了中文插件了,最后将 VSCode 改为中文环境,使用方法如下图所示:



根据上图的提示,按下"Ctrl+Shift+P"打开搜索框,在搜索框里面输入"config",然 后选择"Configure Display Language",如下图所示:

原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php

正点原子



图 4.5.12 配置语言

在打开的 local.json 文件中将 locale 修改为 zh-cn,如下图所示:



图 4.5.13 修改 locale 变量

修改完成以后保存 local.json, 然后重新打开 VSCode, 测试 VSCode 就变成了中文的了, 如下图所示:

শ	文件(F)	编辑(E)	选择(S)	查看(V)	转到(G)	调试(D)	终端(T)	帮助(H)	Visual Stud	lio Code				
ı آ	资源	管理器												
	▲ 打开													
	回乙	KIJTIXIH	光。											
			打开文	(件夹										
\$														
છ 0	▲ 0	-						行6,列2	制表符长度:4	UTF-8	LF J	ISON	۲	<u>۽</u>

图 4.5.14 中文环境

4.5.3 Visual Studio Code 新建工程

新建一个文件夹用于存放工程,比如我新建了文件夹目录为 E:\VScode_Program\1_test, 路径尽量不要有中文和空格打开 VSCode。然后在 VSCode 上点击"文件->打开文件夹...",刚 刚创建的"1_test"文件夹,打开以后如下图所示:

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

正点原子



图 4.5.15 打开的文件夹

从上图可以看出此时的文件夹"1_TEST"是空的,点击文件->将工作区另存为...,打开 工作区命名对话框,输入要保存的工作区路径和工作区名字,如下图所示:

▲ 保存工作区		×
← → ◇ ↑ 📙 > 此电脑 > 工作 (E:) > VScode_Program > 1_test	✓ ひ 捜索"1_test"	Q
组织 ▼ 新建文件夹	工作区路径	
▶ 音乐 ^ 名称 ^	修改日期	<u>u</u>
 ▶ 車車 ▶ 本地磁盘 (C:) ▶ 本地磁盘 (D:) 	家件匹配的项。	
↓ 工作 (E:) ✓ <		>
文件名(N) test	/ 工作区名子	~
保存类型(T): Code 工作区 (*.code-workspace)		~
▲ 隐藏文件夹	保存(S)	取消

图 4.5.16 工作区保存设置

工作区保存成功以后,点击图 4.5.15 中的"新建文件"按钮创建 main.c 和 main.h 这两个 文件, 创建成功后的 VSCode 如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php



图 4.5.17 新建文件以后的 VSCode

从上图可以看出,此时"实验1TEST"中有.vscode文件夹、mian.c和mian.h,这三个文 件和文件夹同样会出现在"实验1test"文件夹中,如下图所示:

▶ ☑ ▶ = 1_1 文件 主页	test 共享 查看		_		× ~ ?
← → • ↑	> 此电脑 > 工作 (E:) > VScode_Program > 1_test		✓ ひ 搜索"1_test"		Q
◆ 快速访问	^ 名称 ^	修改日期	类型	大小	
	.vscode	2019-04-02 18:38	文件夹		
le OneDrive 🍊	i main.c	2019-04-02 18:38	C Source File		0 KB
▶ 此电脑	📄 main.h	2019-04-02 18:38	C/C++ Header File		0 KB
🔥 3D 对象	test.code-workspace	2019-04-02 18:32	CODE-WORKSPAC		1 KB
4 个项目	•				

图 4.5.18 实验文件夹

在 main.h 中输入如下所示内容:

示例代码 4.5.3.1 main.h 文件代码

```
1 #include <stdio.h>
```

3 int add(int a, int b);

在 main.c 中输入如下所示内容:

示例代码 4.5.3.2 main.c 文件代码

```
1 #include <main.h>
2
3 int add(int a, int b)
4 {
      return (a + b);
6 }
8 int main(void)
9 {
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

```
10 int value = 0;
11
12 value = add(5, 6);
13 printf("5 + 6 = %d", value);
14 return 0;
15 }
```

代码编辑完成以后 VSCode 界面如下图所示:

2	文件(F) 编辑(E) 选择(S) 查看(V)	转到(G) 调试(D) 终端(T) ··· main.c - test (工作区) - Visual Studio - [X C
ı آ	资源管理器	C main.c × h main.h ▶ tì	⊞ …
	▲ 打开的编辑器	1 #include <main.h></main.h>	e de la constante de la consta
Q	× C main.c		
-	n main.h	<pre>3 int add(int a. int b)</pre>	_
Ŷ		4 {	_
0	▶ iipch	$\frac{1}{5}$	_
) C main.c	ϵ	_
	h main.h		_
Ē	<pre>{ } test.code-workspace</pre>		- 1
		8 int main(void)	_
		9 {	
		10 int value = 0;	
		12 value = add(5, 6);	
		<pre>13 printf("5 + 6 = %d", value);</pre>	
		14 return 0;	
_		15 }	
*	▶ 大纲	16	

图 4.5.19 代码编辑完成以后的界面

从上图可以看出,VSCode 的编辑的代码高亮很漂亮,阅读起来很舒服。但是此时提示找不到"stdio.h"这个头文件,如下图所示错误提示:

<pre>#include <main.h></main.h></pre>
<pre>#include errors detected. Please update your includePath. IntelliSense features for this translation unit (E:\VScode程序\实验1 test\main.c) will be provided by the Tag Parser.</pre>
cannot open source file "stdio.h" (dependency of "main.h") 供读修复 读监问题

图 4.5.20 头文件找不到

上图中提示找不到"main.h",同样的在 main.h 文件中会提示找不到"stdio.h"。这是因为我们没有添加头文件路径。按下"Ctrl+Shift+P"打开搜索框,然后输入"Edit configurations",选择"C/C++:Edit configurations...",如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

>Edit configurations
C/C++:编辑配置
C/C++: Edit configurations

图 4.5.21 打开 C/C++编辑配置文件

C/C++的配置文件是个 json 文件, 名为: c_cpp_properties.json, 此文件默认内容如下图所

示:

খ	文件(F) 编辑(E) 选择(S) 查看(V) 转到	驯(G) 调试(D)	终端(T) 帮助(H) c_cpp_properties.json - 实验1 test - Visual Studio Code	- 🗆 ×
R	资源管理器		{} c_cpp_properties.json × h main.h	▶ ហ្ Ⅲ …
	▲ 打开的编辑器			a second
	C main.c		"configurations", [
	★ { } c_cpp_properties.json .vscode			
00	h main.h			T
X	▲ 实验1 TEST		"name": "Win32",	
~	🖌 🛃 .vscode 🛛 🛛 main		"includePath": [
) ▶ 📫 ipch		"\${workspaceFolder}/**"	
	<pre>{} c_cpp_properties.json</pre>			
	C main.c), ,	
	h main.h		"defines":	
			"_DEBUG",	
			"UNICODE",	
			"_UNICODE"	
],	
			"intelliSenseMode": "msvc-x64"	
],	
			"version": 4	
يعو				
*	▶ 大纲			
Ø 0	▲ 0 ④ 4		行 6, 列 40 空格:4 UTF-8 CRI	F JSON Win32 😁 🌲 4

图 4.5.22c_cpp_properties.json 内容

c cpp properties.json 中的变量"includePath"用于指定工程中的头文件路径,但是 "stdio.h"是C语言库文件,而VSCode只是个编辑器,没有编译器,所以肯定是没有 stdio.h 的,除非我们自行安装一个编译器,比如 CygWin,然后在 includePath 中添加编译器的头文 件。这里我们就不添加了,因为我们不会使用 VSCode 来编译程序,这里主要知道如何指定 头文件路径就可以了,后面有实际需要的时候再来讲。

我们在 VScode 上打开一个新文件的话会覆盖掉以前的文件,这是因为 VSCode 默认开启 了预览模式,预览模式下单击左侧的文件就会覆盖掉当前的打开的文件。如果不想覆盖的话 采用双击打开即可,或者设置 VSCode 关闭预览模式,设置如下图所示:



原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php



图 4.5.23 取消预览

我们在编写代码的时候有时候会在右下角有如下图所示的警告提示:

(Unable to activate Flexelint analyzer.	\$ ^	×
Inable to activate CppCheck analyzer.		
A Unable to activate Clang analyzer.		

图 4.5.24 错误提示

这是因为插件 C/C++ Lint 打开了几个功能,我们将其关闭就可以了,顺便也可以学习一下 VSCode 插件配置方法,如下图所示:



正点原子

图 4.5.25 C/C++ Lint 配置界面

在 C/C++ Lint 配置界面上找到 CLang:Enable、Cppcheck:Enable、Flexelint:Enable 这个三个,然后取消掉勾选即可,如下图所示:

Clang: Enable Enable or disable the Clang linter
Flexelint: Enable Enable or disable the Flexelint linter
Cppcheck: Enable Enable or disable the CppCheck linter

图 4.5.26 C/C++ Lint 配置

按照图 4.5.25 所示取消这三个有关 C/C++ Lint 的配置以后就不会有图 4.5.24 所示的错误 提示了。但是关闭 Cppcheck:Enable 以后 VSCode 就不能实时检查错误了,大家根据实际情况 选择即可。

4.6 MobaXterm 软件安装和使用

4.6.1 MobaXterm 安装

在后续的开发过程中我们需要在 Windows 下使用串口终端,用来查看信息以及进行操作。 常用的串口终端软件有 MobaXterm、Putty 和 SecureCRT 等,这里我们使用免费且功能全面的 MobaXterm 软件。

MobaXterm 下载地址为: <u>https://mobaxterm.mobatek.net/download-home-edition.html</u>, 下载 界面如下图所示:

领航者 ZYNQ	之嵌入式 Linux	x 开发指南		②正点原子
原子哥在线教学:	www.yuanzige.com	论坛:www.	openedv.com/for	um.php
MobaXterm	Home Demo Features	Download Plugins Help	Contact f 🎐 🗩 😤	Customer area Buy
MobaXterm Hor	me Edition		点击下载	
Download Mo	baXterm Home Edition (current version):		\sim	
	MobaXterm Home Edition v23. (Portable edition)	0	B MobaXterm Home Editi (Installer edition)	ion v23.0
Download pre	vious stable version: MobaXterm Porta	ble v22.3 MobaXterm Installer	<u>v22.3</u>	
By downloadir	ng MobaXterm software, you accept <u>Mob</u>	aXterm terms and conditions		
You can down	lload the third party plugins and compone	ents sources <u>here</u>		
if y giv ver Ple	rou use MobaXterm inside your company, you s e you access to professional support and to the rsions of MobaXterm including your own logo, y ease <u>contact us</u> for more information.	should consider subscribing to Moba) e "Customizer" software. This custom your default settings and your welcom	Clerm Professional Edition: your subsc izer will allow you to generate persona le message.	ription will lilized

图 4.6.1 下载软件

我们已经下载好放到开发板光盘中了,路径为:开发板资料盘(A 盘)/6_常用软件/MobaXterm_Installer_v23.0,将"MobaXterm_installer_23.0"文件夹复制到英文路径下,然后 双击文件夹中的"MobaXterm_installer_23.0.msi"开始安装,安装界面如下图所示:



图 4.6.2 MobaXterm 安装界面

点击上图中的"Next"按钮,进入 License 许可界面,如下图所示:



图 4.6.3 License 界面

在上图中,选择"I accept the terms in the license agreement",然后点击"Next"按钮, 进入安装路径选择界面,大家根据自己的实际情况选择安装路径,如下图所示:

🛃 MobaXterm Setup	_		×
Destination Folder Click Next to install to the default folder or click Change to choose	e another.	>.	\$
Install MobaXterm to:			
D:\Program Files\MobaXterm20\ Change			
Back N	lext	Cano	el

图 4.6.4 选择安装路径

选择好路径后点击"Next"按钮,进入安装确认界面,然后点击"Install"开始安装,入 下图所示:



正点原子

图 4.6.5 点击 Install 安装

安装完成后,点击"Finish"退出安装。到这里,MobaXterm软件就安装成功了,此时桌面上也生成相应图标,如下图所示:



图 4.6.6 MobaXterm 图标

4.6.2 MobaXterm 使用

1、查看开发板当前使用的串口号

在使用 MobaXterm 之前,我们需要先查看串口使用的端口号。首先通过 USB 线将开发板的串口和电脑连接起来,打开"设备管理器",在设备管理器中查看当前连接到电脑的端口都有哪些,如下图所示:



原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php



图 4.6.7 设备管理器

在上图中可以看到有多个 COM 口,哪个才是我们开发板的呢?开发板使用 CH340 芯片完成串口转 USB,所以"USB-SERIAL CH340(COM4)"就是我的开发板所使用的端口,串口 号为 COM4。如果你的电脑连接了多个 CH340 做的 USB 转串口设备,无法区分哪个才是开发 板所使用的,只需要把你的开发板串口拔掉,看看哪个串口号消失了,然后再重新插上开发 板的串口线,再看一下那个消失的串口号会不会重新出现,如果会的话那你的开发板就是用 的这个串口号。

2、设置 MobaXterm

打开 MobaXterm 软件,界面如下图所示:

正点原子 领航者 ZYNQ 之嵌入式 Linux 开发指南 原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php × KobaXterm - 菜单栏 Servers X X server (U) Exit 4 N. Tools ++ —工具栏 xec Tunneling (1) 📩 🔝 User sessions ¢ 1 MobaXterm Start local terminal Select your favorite theme Light Dark

图 4.6.8 MobaXterm 软件界面

点击"Session"图标,创建一个新会话,如下图所示:

💐 Mob	aXterm												
Terminal	Sessions	View	X server	Tools	Games	Settings	Macros	Help					
<u>.</u>	*	1	× 😜	*			Ý	* •	1	**	?		
Session	Servers	Tools	Games	Sessions	View	Split	MultiExec	Tunneling	Packages	Settings	Help		
Quick	connect.					¢							
* 🗈	User session	ns											
												~	MobaXterm

图 4.6.9 创建新会话

"Session Settings"界面用来选择会话类型,常用的类型有 SSH、Telnet、Serial(串口) 等,这里我们选择"串口",入下图所示:



图 4.6.10 会话类型选择串口

然后在"Basic Serial settings"选项中,设置串口端口号,这里我选择图 4.6.7 中的 COM4, 读者可根据自己的实际情况选择,然后设置波特率为"115200",如下图所示:

ion seu	ungs													
٩.		•	X	==	v ĉ	3	۲		@	>	3	X	6 60	-
SSH	Telnet	Rsh	Xdmcp	RDP	VNC	FTP	SFTP	Serial	File	Shell	Browser	Mosh	Aws S3	WSL
🖋 Ba	sic Serial	settings												
	Serial por	t * COM	4 (USB-SI	ERIAL CH	1340 (CON	14))	~	\$	Speed (bp	os) * 1152	200 ~			
۸d 🔊	vanced Se	rial cotti	nas 🛃	l Termina	l cottings	F	Bookmark	eattinge						
			5 _											
	Serial (COM) session													
					C	ОК		😣 Car	ncel					

图 4.6.11 设置端口号和波特率

继续点击下面的"Advanced Serial settings"标签页,设置数据的比特数和停止位,这里 我们使用默认的 8bit 数据和 1bit 停止位,接着设置"Flow control"为"None",如下图所示:

正点原子 领航者 ZYNQ 之嵌入式 Linux 开发指南 原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php Session settings × N ٩, ۰ X v 2 8 <u>ф</u> . **e** ۲ > 2 SSH Telnet Rsh Xdmcp RDP VNC FTP SFTP File Shell Browser Mosh Aws S3 WSL Serial Serial settings Serial port * COM4 (USB-SERIAL CH340 (COM4)) Speed (bps) * 115200 ~ \sim Advanced Serial settings 📓 Terminal settings 🛛 🔶 Bookmark settings Serial engine: PuTTY (allows manual COM port setting) \sim Data bits 8 If you need to transfer files (e.g. router Stop bits 1 configuration file), you can use MobaXterm embedded TFTP server Parity None Flow control Non "Servers" window --> TFTP server O Reset defaults Execute macro at session start: \sim 🕑 OK 😣 Cancel

图 4.6.12 设置串口数据格式

最后点击"OK",串口会话就创建好了,如下图所示:



图 4.6.13 创建好的串口会话

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

正点原子

第五章 Petalinux 的安装

本章将带大家来安装 Petalinux 开发工具,需要注意的是与 Vivado 工具不同的是我们不是 将 petalinux 工具安装在 Windows 系统下,而是安装在 Ubuntu 操作系统中,所以在此之前确 保大家已经在虚拟机中安装了 Ubuntu 18.04 64 位操作系统,这也是官方推荐的版本,那么对 于其他版本可能会在安装和使用过程中出现莫名其妙的错误,所以这里不推荐大家使用其他 版本。

在前面的篇章当中已经向大家介绍了 Windows 下虚拟机的安装以及 Ubuntu 操作系统的安 装,如果你还没做好这些准备工作,那么可以先回到前面的章节进行学习。

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

正点原子

5.1 Petalinux 简介

Petalinux 工具是 Xilinx 公司推出的嵌入式 Linux 开发套件,包括了 u-boot、Linux Kernel、 device-tree、rootfs 等源码和库,以及 Yocto recipes,可以让客户很方便的生成、配置、编译及 自定义 Linux 系统。Petalinux 支持 Versal 、Zyng UltraScale+ MPSoC、Zyng-7000 SoC 以及 MicroBlaze,可与Xilinx硬件设计工具Vivado协同工作,大大简化了Linux系统的开发工作。 具体的介绍可访问 Petalinux 工具网站: https://china.xilinx.com/products/design-tools/embeddedsoftware/petalinux-sdk.html

5.2 下载 Petalinux 安装包

这里先给大家说明一下, petalinux (2020.2) 的安装包文件比较大, 有 1.85G 左右, 如果 大家网速太慢或者不想浪费时间下载,我们已经在开发板光盘资料中提供了 Petalinux 安装包 软件,读者可以直接使用,在光盘路径:"开发板工具盘(B盘)->Petalinux-> petalinux-> v2020.2-final-installer.run",使用光盘提供的安装包文件的可以跳过该小节;如果读者想亲自 下 可 以 到 载 Xilinx 的 官 方 XX 站 https://china.xilinx.com/support/download/index.html/content/xilinx/zh/downloadNav/embeddeddesign-tools/archive.html 进行下载进行下载,如下图所示(注:需要注册 Xilinx 账号并登陆才 能下载):

Vivado (硬件开 发者)	Vitis (软件开发者)	Vitis 嵌入式平台	Alveo 软件包	PetaLinux	器件模型	文档导航
Version	不支持档案下载	或。我们强烈建议统	怒使用最新版本。			
2022.2	2020					
2022.1 2021.2	2020.3					
2021.1	2020.2					
	PetaLinu: 重要信息 PetaLinu: 对用户接受 使用任何所 先安装 PetaLinux MD5 SUM Va	x 工具 - 安装程) 工具安装程序可使用以 ¹ 许可证遵循的所需主机 需的路径进行安装。注 aLinux 工具。 * 2020.2 安装程序 (TAR/G) lue : 687b018f7502a4258	字 - 2020.2 Full Pro 下链接下载。该安装程/ 数据包要求进行检查。 意:所有 BSP(如下) ZIP - 1.85 GB) bd633dc483bde79	oduct Installation 下载类型 它可 都需要 答案 技术文林	멛 dated 当	Full Product Installation 2020-11-24 发布说明和已知问题 PetaLinux 工具文档 安装信息

在下载页面中,可以看到左侧 Petalinux 版本中列出了"2022.2"、"2022.1"、"2021.2" 和"2021.1"几个版本。由于 Petalinux 的版本要求与 Vivado 的版本一致,而我们使用的

图 5.2.1 下载页面及安装文件



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

Vivado 选用的是 2020.2 版本,所以这里点击"存档",找到我们需要的"2020.2"版本,然 后点击"Petalinux 2020.2 安装程序"下载 Petalinux。

如果弹出的账号信息确认界面,直接点击"Download",软件就开始下载了,如下图所示:

居住地*	州/省	
China ~		
城市*	Postal Code	
•		
电话		
工作职能*		
	~	
如需了解我们如何使用您的个人信息、如何保护您个人信息的隐私权、以及如	如何与我们联系,请参阅 <mark>隐私政策</mark> 。	
Download		
		•

图 5.2.2 点击 Download 下载

下载完成之后就获得了 petalinux 的安装文件 petalinux-v2020.2-final-installer.run,如下图 所示:

petalinux-v2020.2-final-installer.run	2023年3月7日 星期二	RUN 文件	1,937,308

图 5.2.3 Petalinux 安装包

5.3 安装 Petalinux

首先将 petalinux 安装包文件 petalinux-v2020.2-final-installer.run 拷贝到 share 共享目录(见 《4.2 Ubuntu 和 Windows 文件本地共享》小节),在 Ubuntu 系统中,打开终端,切换到 /mnt/hgfs/share/目录,可以看到 petalinux 的安装文件 petalinux-v2020.2-final-installer.run 已在该 目录,可直接访问,如下图所示:

<mark>sqd@sqd-virtual-machine:</mark> ~\$ ls -l /mnt/h 总用量 1937312	ngfs/share/
drwxrwxrwx 1 root root 4096 3月	7 18:54 linux tool
-гwxгwxrwx 1 root root 1983802642 3月	7 18:51 petalinux-v2020.2-final-installer.run 🍊
drwxrwxrwx 1 root root 0 3月	2 15:04 设备树文件
sqd@sqd-virtual-machine:~\$	

图 5.3.1 安装包复制到 share 共享目录中

5.3.1 安装依赖库以及软件

在安装 Petalinux 之前我们需要为 Ubuntu 系统安装一些必要的运行软件以及依赖库,所以 大家需要确保 Ubuntu 能够正常上网,打开 Ubuntu Terminal 终端执行如下命令(该命令是一 行,不要分开成多行):

sudo apt-get install iproute2 gawk python3 python build-essential gcc git make net-tools libncurses5-dev tftpd zlib1g-dev libssl-dev flex bison libselinux1 gnupg wget git-core diffstat chrpath socat xterm autoconf libtool tar unzip



原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php

texinfo zlib1g-dev gcc-multilib automake zlib1g:i386 screen pax gzip cpio python3-pip python3-pexpect xz-utils

debianutils iputils-ping python3-git python3-jinja2 libeg11-mesa libsd11.2-dev pylint3

执行结果如下图所示:

<pre>sqd@sqd-virtual-machine:~\$ sudo apt-get install iproute2 gawk python3 python bui</pre>
ld-essential gcc git make net-tools libncurses5-dev tftpd zlib1g-dev libssl-dev .
flex bison libselinux1 gnupg wget git-core diffstat chrpath socat xterm autoconf
libtool tar unzip texinfo zlib1g-dev gcc-multilib automake zlib1g:i386 screen p
ax gzip cpio python3-pip python3-pexpect xz-utils debianutils iputils-ping pytho
n3-git python3-jinja2 libegl1-mesa libsdl1.2-dev pylint3
正在读取软件包列表 完成
正在分析软件包的依赖关系树
正在读取状态信息 完成
注意,选中 'git' 而非 'git-core'
debianutils 已经是最新版 (4.8.4)。
diffstat 已经是最新版 (1.61-1build1)。
libselinux1 已经是最新版 (2.7-2build2)。
python3-pexpect 已经是最新版 (4.2.1-1)。
python3 已经是最新版 (3.6.7-1~18.04)。
将会同时安装下列软件:
autotools-dev blt cpp cpp-7 dh-python dirmngr dpkg-dev fakeroot g++ g++-7
gcc-7 gcc-7-base gcc-7-multilib gcc-8-base gcc-8-base:i386 gir1.2-snapd-1
git-man gnupg-l10n gnupg-utils gpg gpg-agent gpg-wks-client gpg-wks-server
gpgconf gpgsm gpgv lib32asan4 lib32atomic1 lib32cilkrts5 lib32gcc-7-dev
图 5.3.2 运行结果

除了上面使用命令的方法安装依赖库,也可以使用 Xilinx 提供的脚本 plnx-env-setup.sh 安装。该脚本可以从 <u>https://www.xilinx.com/support/answers/73296.html</u>处下载,如下图所示:

+ Files (1)			Download
FILE NAME	SIZE	ACTION	1
plnx-env-setup.sh	5.95 KB	•	

图 5.3.3 使用脚本安装依赖库

将下载后的脚本拷贝到 Ubuntu 虚拟机中,打开 Ubuntu Terminal 终端执行输入如下命令以执行此脚本:

sudo ./plnx-env-setup.sh 如下图所示:



正点原子

图 5.3.4Ubuntu 安装必要依赖库及软件

这里需要等待大概1-2分钟,等待软件以及库安装完成之后即可进入下一步。安装完成后 如下图所示:



图 5.3.5 脚本方式安装库

5.3.2 修改 bash

Petalinux 工具需要主机系统的/bin/sh 是 bash,而 Ubuntu 默认的/bin/sh 是 dash,所以这里 需要进行更改。运行 sudo dpkg-reconfigure dash 命令进行更改,执行结果如下图所示:

领航者 ZYNQ 之嵌入式 Linux 开发指南 ②正点原子						
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php						
正在设定 dash						
The system shell is the default command interpreter for shell scripts.						
Using dash as the system shell will improve the system's overall performance. It does not alter the shell presented to interactive users.						
Use dash as the default system shell (/bin/sh)?						
<是> <否>						

图 5.3.6 禁用 dash

选择"否"按下回车即可。

5.3.3 安装 Petalinux

安装 Petalinux 就要考虑安装位置了,对于 Petalinux 这种体积庞大的工具,我们将其放在 /opt 目录下。在/opt 目录下新建专门存放 Petalinux 的文件夹,如/opt/pkg/petalinux/2020.2,在 终端输入以下命令即可:

sudo chown -R \$USER:\$USER /opt

mkdir -p /opt/pkg/petalinux/2020.2

chown 命令将/opt 目录的属主和属组更改为当前的用户名,如笔者的 Ubuntu 用户名为 sqd,执行的 chown 命令就相当于 "sudo chown -R sqd:sqd /opt",然后通过 mkdir 创建安装目录。现在我们将 petalinux 安装在/opt/pkg/petalinux/2020.2 目录下,在终端中输入如下命令:

./petalinux-v2020.2-final-installer.run -d /opt/pkg/petalinux/2020.2

执行 Petalinux 的安装,如下图所示:

<pre>sqd@sqd-virtual-machine:/mnt/hgfs/share\$./petalinux-v2020.2-final-installer.run</pre>
-d /opt/pkg/petalinux/2020.2
INFO: Checking installation environment requirements
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 refer to "UG1144 PetaLinux Tools Documen
tation Reference Guide" for its impact and solution
INFO: Checking installer checksum
INFO: Extracting PetaLinux installer

图 5.3.7 安装 petalinux

运行上述命令后,需要等待一段时间,当出现"Press Enter to display the license agreements"字样的时候,如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

LICENSE AGREEMENTS PetaLinux SDK contains software from a number of sources. Please review the following licenses and indicate your acceptance of each to continue. You do not have to accept the licenses, however if you do not then you may not use PetaLinux SDK. Use PgUp/PgDn to navigate the license viewer, and press 'q' to close Press Enter to display the license agreements

图 5.3.8 按下回车显示许可协议

从显示的意思可以知道,让我们按下回车键显示软件许可协议,按下键盘上的回车键 Enter,显示协议内容如下:

XILINX, INC. END USER LICENSE AGREEMENT FOR PETALINUX TOOLS CAREFULLY READ THIS END USER LICENSE AGREEMENT FOR PETALINUX TOOLS ("AGREEMENT") . BY CLICKING THE "ACCEPT" OR "AGREE" BUTTON, ENTERING <mark><93></mark>YES<mark><94></mark> OR <mark><93></mark>Y<94> TO ACCEPT THIS AGREEMENT, OR OTHERWISE ACCESSING, DOWNLOADING, INSTALLING OR US ING THE SOFTWARE, YOU AGREE ON BEHALF OF LICENSEE TO BE BOUND BY THIS AGREEMENT. IF LICENSEE DOES NOT AGREE TO ALL OF THE TERMS AND CONDITIONS OF THIS AGREEMENT, DO NOT CLICK THE "ACCEPT" OR "AGREE" BUTTON, ENTER <93>YES<94> OR <93>Y<94>, OR ACCESS, DOWNLOAD, INSTALL OR USE THE SOFTWARE. Definitions 1. "Bitstream" means a machine-executable, binary form of a core used to program a Xilinx Device. "Licensee" means the individual, corporation or other legal entity who has downl oaded and installed the Software. "User" means a specific human being who is identified by Licensee as a person wh o is authorized to use the applicable Software on behalf of Licensee. In cases /tmp/tmp.vGB76emr9w/Petalinux EULA.txt

图 5.3.9 许可协议内容

这些内容如果你感兴趣可以看看,此处我们就不详细看了,直接按下键盘上的Q键退出,回到之前的界面之后会出现一个选择项,询问我们是否接受 xilinx 最终用户协议。

Use PgUp/PgDn to navigate the license viewer, and press 'q' to close Press Enter to display the license agreements Do you accept Xilinx End User License Agreement? [y/N] > y

图 5.3.10xilinx 最终用户协议

这个显然是没得选,必须接受,否则无法进行下面的安装;输入 y 按回车接受。除了 xilinx 最终用户协议之外,还有两个协议也需要大家接受,会依次显示在终端上,同理也是输入 Y 按下回车接受。



接受所有协议之后 Petalinux 安装工具便会继续安装,直到安装完成。安装完成之后,我 们进入到安装目录下,目录内容如下图所示:

sqd@sqd-virtual-machine:/opt/pkg/petalinux/2020.2\$ ls components doc etc settings.csh settings.sh tools sqd@sqd-virtual-machine:/opt/pkg/petalinux/2020.2\$

图 5.3.12petalinux 安装目录内容

5.4 设置 Petalinux 环境变量

在正式使用 petalinux 工具之前,需要先运行 petalinux 安装目录下的 settings.sh 脚本文件 设置 petalinux 工作环境, settings.sh 脚本用于 bash,还有一个 settings.csh 用于 C shell,如下图 所示:

sqd@sqd-virtual-machine:/opt/pkg/petalinux/2020.2\$ is components doc etc settings.csh settings.sh tools sqd@sqd-virtual-machine:/opt/pkg/petalinux/2020.2\$

图 5.4.1 settings.sh 脚本文件

一般默认情况下,我们使用 bash 作为登录 shell,所以 source 脚本文件 settings.sh 对 petalinux 所需的运行环境进行配置,命令如下:

source settings.sh

需要注意的是该命令只对当前终端有效,重新打开终端后需要重新执行这一步。执行结果如下图所示:

sqd@sqd-virtual-machine:/opt/pkg/petalinux/2020.2\$ source settings.sh
PetaLinux environment set to '/opt/pkg/petalinux/2020.2'
INF0: Checking free disk space
INF0: Checking installed tools
INF0: Checking installed development libraries
INF0: Checking network and other services

图 5.4.2 运行 settings.sh 脚本文件

我们来验证下工作环境是否已设置,在终端输入如下命令: echo \$PETALINUX

结果如下图所示:

sqd@sqd-virtual-machine:/opt/pkg/petalinux/2020.2\$ echo \$PETALINUX
/opt/pkg/petalinux/2020.2

图 5.4.3 显示 ETALINUX 变量

显示 Petalinux 的安装目录,表明工作环境已设置。现在可以使用 Petalinux 工具了。



原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php

鉴于每次打开终端使用 Petalinux 都需要设置相应的环境变量,我们为了方便,将设置 Petalinux 环境变量的命令设置成别名,这样我们使用起来就方便些。设置别名方法的很简单, 在终端输入如下命令:

echo "alias sptl='source \$PETALINUX/settings.sh'" >> ~/.bashrc

以后我们打开终端后,输入 sptl 就可以设置 Petalinux 的环境变量了,无需输入长长的路径。sptl 的记忆法是 Source PeTaLinux 的环境变量。提醒:在使用 linux 的时候要善用别名但不要滥用别名。

5.5 安装 Vitis 软件

5.5.1 下载 Vitis 在线安装包

Vitis 是用来开发 Linux 应用的软件,使用这个软件开发 Linux 应用,既方便了工程管理,也免去了自己写 Makefile 的必要。这个软件跟我们在《领航者 ZYNQ 嵌入式 Vitis 开发指南》中使用的 Vitis 是一样的,只不过那个是安装在 Windows 系统下的。

本小节我们将介绍如何下载并安装 Linux 版本 Vitis。同样我们在开发板光盘资料中提供 了该在线安装程序的安装包文件,读者可以跳过下载步骤,直接使用此安装包在线安装。

安装包文件放在光盘路径: "开发板工具盘(B盘) -> Petalinux-> Xilinx_Unified_2020.2_1118_1232_Lin64.bin"。

点 击 网 址

<u>https://china.xilinx.com/support/download/index.html/content/xilinx/zh/downloadNav/vitis/archive-vitis.html</u> 进入 Vitis 软件官方下载页面,在页面左边 Verison 一栏中找到 Vitis 存档选项并点击, 然后找到 2020.2 版本,如下图所示(注意:下载之前需要注册并登陆 Xilinx 账号):

amd zi Xilinx	产品	解决方案	下载与支持 i	商城			L	Q
脅 / 技术支持 /	下载							
下载								
			许可支持		件与驱动	*		
Vivado (硬 者)	•叶开发 	Vitis (软件开发者)	Vitis 嵌入式平台	Alveo 软件包	PetaLinux	器件模型	文档导航	
Version		自 2019.2 起, Xilinx SDSoC ヲ	Xilinx SDSoC™ ヲ ∓发环境的 2019.	F发环境将统一到一个 2 或更高版本。	~多合一的 Vitis™ 统	一软件平台中。将不	云有	
2022.2 2022.1		2020						
2021.2		2020.3						
Vitis 存档		2020.2						-
SDAccel 存档	¥1							反復
SDK/PetaLin	ux 存档	Vitio Core	、工生在供。		U - 41			

图 5.5.1 Vitis 下载页面

下拉找到 Linux 版本在线安装包,点击下载,如下图所示:



⋛正点原子

原子哥在线教学: www.yuanzige.com 论坛:www.

论坛:www.openedv.com/forum.php



图 5.5.2 下载 Linux 版本 Vitis 安装包文件

此时会弹出个人信息确认界面,确认无误后点击"Download"即可下载,如下图所示:

	居住地*	州/省		
	China ~			
	城市*	Postal Code		
	1			
	电话			
	工作职能*		_	
	SHX HE	~	剱	
	如需了解我们如何使用您的个人信息、如何保护您个人信息的隐私权、以及如	何与我们联系,请参 <mark>阅隐私政策</mark> 。	N N	
[Download			
			^) •
) Xilinx_Unifie 已下載 58.3/354	ed_20bin 人 4M8, 还要		全部显示	×



5.5.2 在线安装 Vitis 软件

下载完成后,将"Xilinx_Unified_2020.2_1118_1232_Lin64.bin"安装包复制到 4.2 小节创 建的 share 共享文件夹下。进入 Ubuntu 系统中,打开终端,切换到/mnt/hgfs/share/目录,可以 看到 Vitis 在线安装的安装包文件 Xilinx_Unified_2020.2_1118_1232_Lin64.bin 已在该目录,可 直接访问,如下图所示:

sqd@sqd-virtual-machine:/mnt/hgfs/share\$ ls Linux tool Xilinx_Unified_2020.2_1118_1232_Lin64.bin 设备树文件 sqd@sqd-virtual-machine:/mnt/hgfs/share\$					
图 5.5.4 在线安装包复制到 share 目录下					

安装需要联网,确保安装之前 ubuntu 可以访问网络。输入如下命令开始安装: ./Xilinx_Unified_2020.2_1118_1232_Lin64.bin 执行结果如下图所示:



正点原子

弹出如下图所示的安装界面:

	Xilinx Unified 2020.2 Installer - Wel	come	
UNIFIED	Welcome		
Xilinx Installer	We are glad you've chosen Xilinx as your platform development partner Vitis, Vivado Design Environment, Lab Edition, Bootgen, and Documen Supported operating systems for 2020.2 are: - Windows 10 Professional and Enterprise versions 1809, 1903, 1909 a - Red Hat Enterprise Linux 7.4-7.8, and 8.1-8.2: 64-bit - CentOS Linux 7.4-7.8, and 8.1-8.2: 64-bit - SUSE Enterprise Linux 12.4: 64-bit - Amazon Linux 2 AL2 LTS: 64-bit - Ubuntu Linux 16.04.5, 16.04.6, 18.04.1, 18.04.2, 18.04.3, 18.04.4 and	r. This program can install the Xilinx p tation Navigator. and 2004: 64-bit 20.04 LTS: 64-bit - Additional library.	roducts including installation required.
	Note: This release requires upgrading your license server tools to the F admin that the correct version of the license server tools are installed at Note: This installation program will not install cable drivers on Linux. Th administrative privileges. To reduce installation time, we recommend that you disable any anti-vir	iex 11.14.1 versions. Please confirm n nd available, before running the tools. is item will need to be installed separ rus software before continuing.	with your license ately, with
E XILINX.			
Copyright © 1986-2023 Xilinx, I	Inc. All rights reserved.	Preferences < Back	<u>N</u> ext > <u>C</u> ancel

图 5.5.6 安装界面

点击上图中"Next",在接下来的安装类型界面中,先输入 xilinx 账号和密码,然后选择 默认安装类型"Download and Install Now",点击"Next",如下图所示:

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

正点原子

Copyright © 1980-2023 Xilms, Inc. Al rights reserved.	Select Install Type	
User Authentication Please provide your Xllinx user account credentials to download the required files. If you don't have an account, please create one. If you forgot your password, you can reself there. Imail Address Imail Address	Please select install type and provide your Xilinx.com E-mail Address and password for authentication.	
Copyright © 1986-2023 Xilinx, Inc. All rights reserved.	Please select install type and provide your Xilinx.com E-mail Address and password for authentication. User Authentication Please provide your Xilinx user account credentials to download the required files. If you don't have an account, please create one, If you forgot your password, you can reset it here, E-mail Address Password Password Password Password Please and Install Now Password Please device and tool installation options and the installer will download and install just what is required O Download Image (Install Separately) The installer will download an image containing all devices and tool options for later installation. Use this option If you network drive or allow different users maximum flexibility when installing.	wish to install a full image on a
	Copyright © 1986-2023 Xilinx, Inc. All rights reserved.	< <u>B</u> ack <u>N</u> ext > <u>C</u> ancel

图 5.5.7 选择安装类型

在弹出的安装产品选择界面中,我们选择默认的"Vitis",注意 Vitis 软件包含了 Vivado 软件, 然后点击"Next", 如下图所示:

Select Product to Install
Select a product to continue installation. You will be able to customize the content in the next page.
Vitis
Installs Vitis Core Development Kit for embedded software and application acceleration development on Xilinx platforms. Vitis installation includes Vivado Design Suite. Users can optionally add "Xilinx Add-On for MATLAB & Simulink" that includes Model Composer and System Generator to design for Al Engines and Programmable Logic.
🔾 Vivado
Includes the full complement of Vivado Design Suite tools for design, including C-based design with Vitis High-Level Synthesis, implementation, verification and device programming. Complete device support, cable driver, and Document Navigator included.
On-Premises Install for Cloud Deployments (Linux only)
Install on-premises version of tools for cloud deployments.
BootGen
Installs Bootgen for creating bootable images targeting Xilinx SoCs and FPGAs.
🔾 Lab Edition
Installs only the Xilinx Vivado Lab Edition. This standalone product includes the Vivado Device Programmer and Vivado Logic Analyzer tools.
O Hardware Server
Installs hardware server and JTAG cable drivers for remote debugging.
PetaLinux (Linux only)
PetaLinux SDK is a Xilinx development tool that contains everything necessary to build, develop, test, and deploy embedded Linux systems.
O Documentation Navigator (Standalone)
Xilinx Documentation Navigator (DocNav) provides access to Xilinx technical documentation both on the Web and on the Desktop. This is a standalone installation without Vivado Design Suite.
Copyright © 1986-2023 Xilinx, Inc. All rights reserved.

图 5.5.8 产品选择



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

进入 Vitis Software Platform 设置界面,包含的器件库选项中只保留 Zynq-7000 一项,其 它器件都取消勾选,如下图所示:

Vitis Unified Software Platform Customize your installation by (de)selecting items in the tree below. Moving cursor over selections below provide additional information.	3	XILINX
The Vitis unified software platform enables the development of embedded software and accelerated applications on heterog FPGAs, SoCs, and Versal ACAPs. It provides a unified programming model for accelerating Edge, Cloud, and Hybrid compu- is a superset that includes the Vivado Design Suite as well. Users can optionally add "Xilinx Add-On for MATLAB & Simulink and System Generator to design for Al Engines and Programmable Logic.	geneous Xilinx p uting application " that includes N	latforms including s. This installation Aodel Composer
 P ➤ Design Tools P ➤ Vitis Unified Software Platform Y Vitis do Y Vitis ALS Add-On for MATLAB and Simulink Model Composer and System Generator DocNav P Devices Install devices for Alveo and Xilinx edge acceleration platforms P Devices for Custom Platforms P Zynq UltraScale+ MPSoC Zynq UltraScale+ RFSoC P UltraScale UltraScale UltraScale Versal ACAP F Engineering Sample Devices for Custom Platforms P Installation Options NOTE: Cable Drivers are not installed on Linux. Please follow the instructions in UG973 to install Linux cable of the set of the se	trivers	
Download Size: 17.36 GB Disk Space Required: 77.67 GB	E	eset to Defaults
Copyright © 1986-2023 Xilinx, Inc. All rights reserved.	< <u>B</u> ack <u>N</u> e	ext > <u>C</u> ancel

图 5.5.9 取消勾选不需要的工具和器件

点击"Next"进入许可协议界面,全部勾选同意,如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

RE, I AGREE on behalf of
WebTalk and have been afforded stand that I am able to disable ling the Software or using the able steps to prevent such as described in Section 13(b).
RE, I AGREE on behalf of
<u>\</u>
Death Newton Connect

图 5.5.10 接受协议

点击"Next",进入安装路径选择界面,这里设置为"/opt/pkg/tools/Xilinx",如下图所示:

Installation Options Select the installation directory /opt/pkg/tools/Xilinx/ Installation location(s), /opt/pkg/tools/Xilinx/Vita/2020.2 /opt/pkg/tools/Xilinx/Vivado/2020.2	Select shortcut and file association options Create program group entries Xilinx Design Tools Create desktop shortcuts
/opt/pkg/tools/Xilinx/Vitis_HLS/2020.2 Download location /opt/pkg/tool/Downloads/Vitis_2020.2 Disk Space Required Download Size: 17.36 GB Disk Space Required: 77.67 GB Final Disk Usage: 45.01 GB	
Disk Space Available: 422.56 GB	
Copyright © 1986-2023 Xilinx, Inc. All rights reserved.	< <u>B</u> ack <u>N</u> ext > <u>C</u> ancel

图 5.5.11 设置安装路径

点击"Next",在弹出的路径确认窗口中点击"yes",如下图所示:



原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php



图 5.5.12 确认路径

接下来进入"Installation Summary"界面,点击"Install"开始下载并安装,如下图所示:

	Installation Summary			
Edition: Vitis Unified Software Platform Xilinx Installer				
	Devices for Custom Platforms (SoCs)			
	Design Tools			
	• Vitis Unified Software Platform (Vitis, Vivado, Vitis HLS)			
	Installation location			
	 /opt/pkg/tools/Xilinx/Vitis/2020.2 			
	 /opt/pkg/tools/Xilinx/Vivado/2020.2 			
	 /opt/pkg/tools/Xilinx/Vitis_HLS/2020.2 			
	Download location			
	 /opt/pkg/tools/Xilinx/Downloads/Vitis_2020.2 			
	Disk Space Required			
	Download Size: 17.36 GB			
	 Disk Space Required: 77.67 GB 			
	 Final Disk Usage: 45.01 GB 			
E XILINX.		$\mathbf{X}_{\mathbf{r}}$		
Copyright © 1986-2023 Xilinx, I	nc. All rights reserved.	Preferences < <u>B</u> ack <u>I</u> nstall <u>C</u> ancel		

图 5.5.13 开始下载并安装

下载和安装需要的时间比较长,我们只需要等待其安装完成就行。

安装完成后输入下面命令,在桌面创建"Vitis"、"Vitis HLS"和"Vivado"桌面快捷 方式:

cp ~/.local/share/applications/'Xilinx Vitis 2020.2_1678269588565.desktop' ~/桌面

cp ~/.local/share/applications/'Vitis HLS 2020.2_1678269588885.desktop' ~/桌面

cp ~/.local/share/applications/'Vivado 2020.2_1678269588584.desktop' ~/桌面

命令执行结果如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

sqd@sqd-virtual-machine:~\$ cp /home/sqd/.local/share/applications/'Xilinx Vitis
2020.2_1678269588565.desktop' /home/sqd/桌面
sqd@sqd-virtual-machine:~\$ cp /home/sqd/.local/share/applications/'Vitis HLS 202
0.2_1678269588885.desktop' /home/sqd/桌面
sqd@sqd-virtual-machine:~\$ cp /home/sqd/.local/share/applications/'Vivado 2020.2
1678269588584.desktop' /home/sqd/桌面

图 5.5.14 命令执行结果

此时三个图标文件已经复制到桌面上了,如下图所示:



图 5.5.15 desktop 文件

双击即可打开软件,如果出现未信任的应用程序启动器,点击"Trust and Lanch"即可。

5.6 Linux 系统安装 JTAG cable 驱动

在嵌入式开发中,我们都是通过在线 jtag 进行调试的,这种调试方式方便快捷,在使用 Petalinux 进行 Linux 开发中,其实也是可以使用 JTAG 的,不过对于 Linux 系统,由于安装驱 动程序需要 root 或 sudo 访问权限,因此从 Vivado 2015.4 版本开始,默认不安装 jtag 驱动。这 样 Vivado 安装程序和 Petalinux 安装程序可以在没有 root 或 sudo 特权的 Linux 系统上运行。这 也导致了在 linux 系统中,jtag 驱动需要手动安装。下面我们介绍如何在 Ubuntu 主机中安装 jtag 驱动(其他 Linux 系统同样适用)。

需要注意的是在安装jtag驱动之前,请不要将jtag下载器连接到电脑,已经接到电脑的,最好先拔掉。下面开始安装。

在 Ubuntu 系统中,打开终端,以普通用户运行即可。进入到 petalinux 安装目录,如下图 所示:

sqd@sqd-virtual-machine:~\$ cd /opt/pkg/petalinux/2020.2/ sqd@sqd-virtual-machine:/opt/pkg/petalinux/2020.2\$ ls components doc etc settings.csh settings.sh tools

图 5.6.1 Petalinux 安装目录


原子哥在线教学:www.yuanzige.com

论坛:www.openedv.com/forum.php

可以看到有一个名为"tools"的目录, jtag 驱动程序在该目录下, 我们输入以下命令进入到 itag 驱动程序所在位置:

cd tools/xsct/data/xicom/cable_drivers/lin64/install_script/install_drivers

可以看到该目录下有一个名为"install_drivers"的文件,如下图所示:

sqd@sqd-virtual-machine:/opt/pkg/petalinux/2020.2\$ cd tools/xsct/data/xicom/cabl
e_drivers/lin64/install_script/install_drivers
sqd@sqd-virtual-machine:/opt/pkg/petalinux/2020.2/tools/xsct/data/xicom/cable_dr
ivers/lin64/install_script/install_drivers\$ ls
52-xilinx-digilent-usb.rules install_digilent.sh setup_xilinx_ftdi
52-xilinx-ftdi-usb.rules install_drivers
52-xilinx-pcusb.rules setup_pcusb

图 5.6.2 "install drivers"文件

输入如下命令,以 root 权限执行该文件,安装 jtag 驱动程序:

sudo ./install_drivers

执行结果如下图所示:

sqd@sqd-virtual-machine:/opt/pkg/petalinux/2020.2/tools/xsct/data/xicom/cable dr ivers/lin64/install script/install_drivers\$ sudo ./install_drivers [sudo] sqd 的密码: INFO: Installing cable drivers. INF0: Script name = ./install drivers INFO: HostName = sqd-virtual-machine INF0: Current working dir = /opt/pkg/petalinux/2020.2/tools/xsct/data/xicom/cabl e drivers/lin64/install script/install drivers $\overline{INF0}$: Kernel version = $\overline{4.18.0}$ -15-generic. INF0: Arch = x86 64. Successfully installed Digilent Cable Drivers --File /etc/udev/rules.d/52-xilinx-ftdi-usb.rules does not exist. --File version of /etc/udev/rules.d/52-xilinx-ftdi-usb.rules = 0000. --Updating rules file. --File /etc/udev/rules.d/52-xilinx-pcusb.rules does not exist. --File version of /etc/udev/rules.d/52-xilinx-pcusb.rules = 0000. --Updating rules file. INFO: Digilent Return code = 0 INFO: Xilinx Return code = 0 INFO: Xilinx FTDI Return code = 0 INFO: Return code = 0 INF0: Driver installation successful. CRITICAL WARNING: Cable(s) on the system must be unplugged then plugged back in order for the driver scripts to update the cables.

图 5.6.3 jtag 驱动安装成功

可以看到 jtag 驱动安装成功。对于最下面一行的警告,如果已经拔掉 jtag 下载器与电脑 的连接就可以不用看。至此 jtag 驱动程序就安装完成了。

下面说下在遇到 jtag 驱动问题时,如何删除 jtag 驱动。注意如无特需情况,请不要执行 下面的命令:

sudo rm -f /etc/udev/rules.d/52-xilinx-digilent-usb.rules

sudo rm -f /etc/udev/rules.d/52-xilinx-ftdi-usb.rules

sudo rm -f /etc/udev/rules.d/52-xilinx-pcusb.rules

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

正点原子

第六章 Petalinux 设计流程实战

PetaLinux 工具提供了在 Xilinx 处理系统上自定义、构建和部署嵌入式 Linux 解决方案所 需的一切。该解决方案旨在提高设计生产力,可与 Xilinx 硬件设计工具一起使用,以简化针 对 Zynq SoC 的 Linux 系统的开发。本章我们以使用 Petalinux 定制 Linux 系统为例,实战 Petalinux 的设计流程,看下 Petalinux 如何简化 Linux 系统的开发。

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

正点原子

6.1 Zynq-7000 嵌入式软件栈概述

要想在 Zynq-7000 SoC 中搭建运行 Linux, 我们需要先简单的了解下其嵌入式软件栈, 如 下图所示:



图 6.1.1 Zyng-7000 嵌入式软件栈

处理器系统引导是一个分两个阶段的过程。第一个阶段是一个内部 BootROM, 它存储 stage-0 的引导代码。BootROM 在 CPU 0 上执行, CPU 1 执行等待事件(WFE)指令。 BootROM 还配置必要的外围设备,以开始从其中一个引导设备获取第一阶段引导加载程序

(FSBL) 引导代码。可编程逻辑(PL) 不是由 BootROM 配置的。第二种状态是 FSBL 引导 代码。这通常存储在闪存、SD卡中,或者可以通过 JTAG 下载。BootROM 代码将 FSBL 引导 代码从选定的非易失性存储器复制到片上存储器(OCM)。加载到 OCM 中的 FSBL 的大小限 制为 192KB。在 FSBL 开始执行后,完整的 256 KB 可用。有一个可选的第二阶段引导加载程 序,它是可选的,由用户设计。常见的第二阶段引导加载程序是 U-boot。

(1) FSBL

Zyng-7000 的第一阶段引导加载程序(FSBL)使用硬件比特流(如果存在)配置 FPGA, 并将操作系统(OS)映像或第二阶段引导加载器映像从非易失性存储器 (NAND/SD/eMC/OSPI)加载到存储器(DDR/OCM)。它支持多个分区,每个分区可以是 代码映像、位流或通用数据。如果需要,可以对这些分区中的每个分区进行身份验证和/或解 密。

(2) U-Boot

U-Boot,通用引导加载程序的缩写,是一种开源的主引导加载程序,用于嵌入式设备, 以引导Linux社区中经常使用的设备操作系统内核。Xilinx在Zynq-7000设备中使用U-Boot作 为第二阶段引导加载程序。

(3) Linux

Linux, 全称 GNU/Linux, 是一种免费使用和自由传播的类 UNIX 操作系统, 是我们本开 发指南的重点。

以上就简单的介绍了Zyng-7000嵌入式软件栈,如果没有看懂没关系,笔者这里简单的概 括下。Zynq-7000上电后,首先由 BootROM 对 Zynq 设备进行初始启动,然后引导加载 fsbl 到



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

OCM 并启动 fsbl; fsbl 启动后将 uboot 加载到 DDR 并启动 uboot; uboot 启动后加载 linux 系统 镜像到 DDR 并启动 linux,至此整个 linux 系统启动完成。

综上,就是需要在 Zynq-7000 运行 linux 系统所需要搭建的软件栈。如果这些软件栈由我 们一个个手工搭建,任务量极其庞大,所幸的是,Xilinx 推出了 Petalinux 开发工具,可以让 我们方便快捷的完成这些软件栈的搭建,从而加快 linux 的使用和开发。

6.2 Petalinux 工具的设计流程概述

通常 PetaLinux 工具遵循顺序设计流程模型,如下表所示:

表 6.2.1 设	计流程表
-----------	------

Design Flow Step	Tool / Workflow
Hardware Platform Creation	Vivado
Create PetaLinux Project	petalinux-create -t project
Initialize PetaLinux Project	<pre>petalinux-configget-hw-description</pre>
Configure System-Level Options	petalinux-config
Create User Components	petalinux-create -t COMPONENT
Configure the Linux Kernel	petalinux-config -c kernel
Configure the Root Filesystem	petalinux-config -c rootfs
Build the System	petalinux-build
Package for Deploying the System	petalinux-package
Boot the System for Testing	petalinux-boot

从上表可以看到,使用 Vivado 搭建好硬件平台后,通过几个命令就完成了 Linux 系统的 定制,极其方便。

需要说明的是以上设计流程不是按部就班的每一步都执行一遍,可以根据使用场景有选 择的执行。一般的设计流程如下:

- 1. 通过 Vivado 创建硬件平台,得到 xsa 文件;
- 2. 运行 source <petalinux 安装路径>/settings.sh, 设置 Petalinux 运行环境
- 3. 通过 petalinux-create -t project 创建 petalinux 工程;
- 4. 使用 petalinux-config --get-hw-description,将 xsa 文件导入到 petalinux 工程当中并配置 petalinux 工程;
- 5. 使用 petalinux-config -c kernel 配置 Linux 内核;
- 6. 使用 petalinux-config -c rootfs 配置 Linux 根文件系统;
- 7. 配置设备树文件;
- 8. 使用 petalinux-build 编译整个工程;
- 9. 使用 petalinux-package --boot 制作 BOOT.BIN 启动文件;
- 10. 制作 SD 启动卡,将 BOOT.BIN 和 image.ub 以及根文件系统部署到 SD 卡中;
- 11. 将 SD 卡插入开发板,并将开发板启动模式设置为从 SD 卡启动;
- 12. 开发板连接串口线并上电启动,串口上位机打印启动信息,登录进入 Linux 系统。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

6.3 使用 Petalinux 定制 Linux 系统

现在我们以使用 Petalinux 定制 Linux 系统为例,来实战下 Petalinux 的设计流程,体验 Petalinux 对 Linux 系统开发的简便之处。

6.3.1 创建 Vivado 硬件平台

在《领航者 ZYNQ 之嵌入式 Vitis 开发指南》中我们创建了很多 Vivado 工程,相信大家 对 vivado 工程的创建已经非常熟悉了,为了节省时间,这里我们统一使用正点原子为领航者 开发板配置的 vivado 工程,该工程在开发板资料包中已经给大家提供了,路径为: 4_SourceCode\3_Embedded_Linux\vivado_prj,在该目录下有两个.zip 压缩文件 Navigator_7010.zip和Navigator_7020.zip,分别对应领航者7010和领航者7020的vivado工程, 这里大家根据自己所使用的开发板来选择,笔者以领航者 7020 开发板为例,将 Navigator_7020.zip 在 Windows 系统下解压得到 vivado 工程目录 Navigator_7020,进入到该目 录下,如下所示:

	ip_repo	2023年3月3日 星期五	文件夹	
	Navigator_7020.cache	2023年3月6日 星期一	文件夹	
	Navigator_7020.gen	2023年3月6日 星期一	文件夹	
	Navigator_7020.hw	2023年3月6日 星期一	文件夹	
	Navigator_7020.ioplanning	2023年3月6日 星期一	文件夹	
	Navigator_7020.ip_user_files	2023年3月6日 星期一	文件夹	
	Navigator_7020.runs	2023年3月6日 星期一	文件夹	
	Navigator_7020.sim	2023年3月6日 星期一	文件夹	
	Navigator_7020.srcs	2023年3月6日 星期一	文件夹	
A	Navigator_7020.xpr	2023年3月6日 星期一	Vivado Project Fi	126 KB
	system_wrapper.xsa	2023年3月6日 星期一	XSA 文件	1,088 KB

图 6.3.1 vivado 工程目录

该目录下有一个 system_wrapper.xsa 文件,该文件包含着 Vivado 工程所对应的硬件平台 信息,Petalinux 根据这些信息来配置 fsbl、uboot、内核等。

将该文件拷贝到 Ubuntu 系统下,譬如笔者在 Ubuntu 系统用户根目录下创建了一个名为 "petalinux"的目录,并在该目录下创建 "xsa_7020" 目录用于存放 system_wrapper.xsa 文件。将 system_wrapper.xsa 文件拷贝到 petalinux/xsa_7020 目录中,如下图所示:

sqd@sqd-virtual-machine:~/petalinux/xsa_7020\$ cp /mnt/hgfs/share/system_wrapper.xsa .
sqd@sqd-virtual-machine:~/petalinux/xsa_7020\$ ls
system_wrapper.xsa

图 6.3.2 将 xsa 文件复制到 petalinux/xsa_7020 共享目录下

6.3.2 设置 Petalinux 环境变量

现在进入到 Ubuntu 系统中,打开终端,以普通用户运行即可,不需要使用 root 用户。在 正式使用 petalinux 工具之前,需要先设置 petalinux 的作环境,在终端输入如下命令即可:

source /opt/pkg/petalinux/2020.2/settings.sh

#或者

sptl



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

执行结果如下图所示:

sqd@sqd-virtual-machine:~\$ sptl
PetaLinux environment set to '/opt/pkg/petalinux/2020.2'
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 refer to "UG1144 2020.2 PetaLinux Tools D
ocumentation Reference Guide" for its impact and solution

图 6.3.3 设置 petalinux 工作环境

6.3.3 创建 petalinux 工程

我们在 6.3.1 小节中创建了 petalinux 目录,现在我们在该目录中创建一个名为 "ALIENTEK-ZYNQ"的 Petalinux 工程,在终端中输入如下命令:

petalinux-create -t project --template zynq -n ALIENTEK-ZYNQ

-t 是 "--type"的简写。template 参数表明创建的 petalinux 工程使用的平台模板,此处的 zynq 表明使用的是 zynq 平台模板的 petalinux 工程,用于 zynq-7000 系列的芯片。name 参数 (此处简写为 "-n") 后接的是 petalinux 工程名,如此处的 "ALIENTEK-ZYNQ"。执行结 果如下图所示:

可以看到该命令会自动在 petalinux 目录下创建一个名为 ALIENTEK-ZYNQ 的文件夹,也就是 ALIENTEK-ZYNQ 对应的 Petalinux 工程所在目录。

6.3.4 配置 petalinux 工程

首次配置 Petalinux 工程是将 xsa 文件导入到 Petalinux 工程中, Petalinux 工具会解析 xsa 文件并弹出配置窗口。在终端中输入如下命令配置 Petalinux 工程:

cd ALIENTEK-ZYNQ

petalinux-config --get-hw-description ../xsa_7020/

即进入到 ALIENTEK-ZYNQ 文件夹,并配置 petalinux 工程。"petalinux-config --get-hw-description"命令后面的文件夹就是我们复制到 Ubuntu 系统下 system_wrapper.xsa 文件所在的位置。如果后面修改了 Vivado 工程,重新生成 xsa 文件后,可以重新执行"petalinux-config --get-hw-description < xsa 文件所在的位置>"以重新配置 Petalinux 工程。

执行结果如下图所示:



原子哥在线教学:	www.yuanzige.com	论坛:www.openedv.com/forum.php
sqd@sqd-virt	ual-machine:~/petalinux	<pre>\$ cd ALIENTEK-ZYNQ/</pre>
sqd@sqd-virt	<pre>ual-machine:~/petalinux</pre>	<pre>«/ALIENTEK-ZYNQ\$ ls</pre>
config.proje	ct project-spec	
sqd@sqd-virt	ual-machine:~/petalinux	<pre></pre>
cription/	xsa 7020/	
INFO: Sourci	ng build tools	
INFO: Gettin	g hardware description.	
INFO: Rename	system wrapper.xsa to	system.xsa
[INF0] Gener	ating Kconfig for proje	ect
[INFO] Menuc	onfig project	

图 6.3.5 配置 petalinux 工程

弹出 petalinux 工程配置窗口,如下图所示:



图 6.3.6 petalinux 工程配置窗口

需要注意的是该窗口不可以使用鼠标操作,只能通过键盘操作,界面上方的英文就是简 单的操作说明,操作方法如下:

通过键盘上的"↑"和"↓"键来选择要配置的菜单,按下"Enter"键进入子菜单。菜 单中蓝色高亮的首字母就是此菜单的热键,在键盘上按下此高亮字母对应的键可以快速选中 对应的菜单。选中子菜单以后按下"Y"键就会将相应的配置选项写入配置文件中,菜单前 面变为"<*>"。按下"N"键不编译相应的代码,按下"M"键就会将相应的代码编译为 模块,菜单前面变为"<M>"。按两下"Esc"键退出,也就是返回到上一级,按下"?"键 查看此菜单的帮助信息,按下"/"键打开搜索框,可以在搜索框输入要搜索的内容。

在配置界面下方会有五个按钮,这五个按钮的功能如下: <Select>: 选中按钮,和"Enter"键的功能相同,负责选中并进入某个菜单。 <Exit>: 退出按钮,和按两下"Esc"键功能相同,退出当前菜单,返回到上一级。 <Help>: 帮助按钮,查看选中菜单的帮助信息。 <Save>: 保存按钮,保存修改后的配置文件。 <Load>: 加载按钮,加载指定的配置文件。

正点原子 领航者 ZYNQ 之嵌入式 Linux 开发指南 原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php 本实验我们无需更改该窗口的配置信息。不过由于该窗口菜单不多,我们就从上到下的 简单地介绍下这些菜单。 首先按键盘上的下方向键移动到"Linux Components Selection",然后按键盘上的 "Enter" 进入子菜单,子菜单内容如下图所示: Linux Components Selection Arrow keys navigate the menu. < Enter> selects submenus ---> (or empty submenus ----). Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] built-in [] [] Pre FSBL [*] First Stage Bootloader [*] Auto update ps init u-boot (u-boot-xlnx) ---> linux-kernel (linux-xlnx) ---> <Select> < Exit > < Help > < Save > < Load >

图 6.3.7 LinuxComponentsSelection 子菜单

中括号里的"*"表示已使能配置。第二项和第三项表示会自动生成我们在《领航者 ZYNQ之嵌入式 Vitis 开发指南》程序固化实验中使用的 fsbl.elf 文件和自动更新 ps_init。下面 两个选项用来配置 u-boot 和 linux-kernel 的来源,本实验保持默认来源配置,不做改动,后面 的实验需要更改的时候再做介绍。按键盘上的"Esc"按键连按两次退出该子菜单。

"Auto Config Settings"菜单主要就是选择是否使能 Device tree、Kernel 和 u-boot 的自动 配置,这里使用默认配置,无需更改,就不看了。

"Subsystem AUTO Hardware Settings"子菜单的内容如下图所示:



图 6.3.8 SubsystemAUTOHardwareSettings 子菜单

进入到该界面的各个外设子菜单中,可以发现都已经设置好了默认配置,这些默认配置 是根据 xsa 文件的信息自动配置的,基本上无需我们手动配置; "Serial Settings"配置项用于 配置开发板的调试串口和串口波特率等参数,对于领航者开发板来说,这里需要修改一下, 首先光标移动到该配置项按回车进入,如下所示:



图 6.3.9 配置串口

把 "FSBL Serial stdin/stdout"和 "DTG Serial stdin/stdout"两项中默认使用的 "ps7_uart_1"改成 "ps7_uart_0",也就是 USB 串口。这是因为 uart1 是 PL 端的 RS232 和 RS485 接口。修改完成之后连按两次 ESC 键回到上一级菜单。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

在"Advanced bootable images storage Settings"菜单中可配置启动引导镜像和内核镜像的存储媒介,默认为 Primary SD,这里保持默认即可。



图 6.3.10 Advanced bootable images storage Settings 菜单

返回到主界面(按四次"ESC"按键), "DTG Settings"、FSBL 配置菜单"FSBL Configuration"、FPGA 管理器菜单"FPGA Manager"、u-boot 配置菜单"u-boot Configuration"和 linux 配置菜单"Linux Configuration"一般保持默认即可。

我们进入"Image Packaging Configuration"子菜单,如下图所示:



图 6.3.11 ImagePackagingConfiguration 子菜单



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

第一个选项便是根文件系统的类型的配置,默认为 INITRD,一般默认即可。如果我们需要运行 Ubuntu 或 Debian 的根文件系统时,就需要配置成 EXT4(SD/eMMC/ SATA/USB), NFS 挂载启动需要配置成 NFS。

注: INITRD 类型的根文件系统每次重新启动 linux 系统都是全新的、未改动过的,也就 是说启动系统后进行的所有修改掉电后就全部丢失了,再次重新启动还是之前未修改过的根 文件系统,选择"EXT4"可以将根文件系统放在 SD 卡、eMMC 的 ext4 分区,这样启动系统 后进行的所有修改掉电后就不会丢失了。后面会讲解如何将根文件系统放在 SD 卡的 ext4 分 区。本章还是先从简单的 INITRD 类型用起。

另外从该界面我们可以看到,有"Copy final images to tftpboot"选项,当在 Ubuntu 的根 文件下创建一个名为 tftpboot 的文件夹时,工程生成镜像后会自动将相关文件复制到/tftpboot 目录中。

回到主界面, "Firmware Version Configuration"可以用来修改定制的 linux 系统的主机名 和产品名,默认与该 Petalinux 工程同名,如果需要可修改。"Yocto Settings"进行与 Yocto 相关的设置,这里就不做介绍了,一般保持默认即可。

按键盘上的右方向键(即右箭头),移动到底部的"Save",按键盘上的"Enter"键,进入如下图所示的保存配置文件界面:



图 6.3.12 保存配置文件界面

按键盘上的"Enter"键确认,进入下图所示界面:



图 6.3.13 退出保存配置文件界面

再次按键盘上的"Enter"键确认,直接返回到主界面,到这里配置修改就完成了。最后 按两次键盘上的"Esc"退出配置窗口,Petalinux工具开始自动配置工程。

这一步可能需要几分钟才能完成。这是因为 PetaLinux 会根据 "Auto Config Settings --->"和"Subsystem AUTO Hardware Settings --->"来解析 xsa 文件,以获取更新设备树、U-Boot 配置 文件和内核配置文件所需的硬件信息。

等待一段时间后,完成 petalinux 工程的配置,如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:w

论坛:www.openedv.com/forum.php

sqd@sqd-virtual-machine:~/petalinux/ALIENTEK-ZYNQ\$ petalinux-config --get-hw-des cription ../xsa 7020/ INFO: Sourcing build tools INFO: Getting hardware description... INFO: Rename system wrapper.xsa to system.xsa [INFO] Generating Kconfig for project [INFO] Menuconfig project *** End of the configuration. *** Execute 'make' to start the build or try 'make help'. [INF0] Extracting yocto SDK to components/yocto [INF0] Sourcing build environment [INFO] Generating kconfig for Rootfs [INF0] Silentconfig rootfs [INFO] Generating plnxtool conf [INF0] Adding user layers [INF0] Generating workspace directory

图 6.3.14 完成 petalinux 工程的配置

如果后面想重新配置,只需输入"petalinux-config"命令即可重新配置。

6.3.5 配置 Linux 内核

现在我们开始定制 Linux 内核,在终端输入如下命令:

petalinux-config -c kernel

执行结果如下:



等待一段时间后会弹出 Linux 内核的配置界面,如下图所示:



图 6.3.16 内核配置界面

可以看到 Petalinux 默认使用的内核版本为 5.4.0,当然也可以换成其它版本的内核,不过修改起来比较麻烦,Petalinux 对内核版本有要求,读者如需使用其他的内核版本可以在网上查找关于 Petalinux 使用非默认内核版本的方法。一般使用默认内核版本就可以了。

这里使用的内核 Xilinx 官方已经做好了基础配置,如无特定需求,无需更改。另外关于 Linux 内核的配置在后面的 Linux 内核移植章节进行讲解,此处就不多做介绍了。这里采用 Xilinx 官方的默认配置即可,保存配置并退出。

注意,如果配置内核时,出现下图所示的报错信息,参考错误!未找到引用源。小节的 Petalinux 错误解决办法,按照这3种解决办法,重新配置 Petalinux 工程。



原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php WARNING: Failed to fetch URL https://ftp.gnu.org/gnu/autoconf/autoconf-2.69.tar. gz, attempting MIRRORS if available WARNING: Failed to fetch URL https://downloads.sourceforge.net/flex/flex-2.6.0.t ar.bz2, attempting MIRRORS if available WARNING: Failed to fetch URL https://ftp.gnu.org/gnu/bison/bison-3.4.1.tar.xz, a ttempting MIRRORS if available WARNING: Failed to fetch URL git://anongit.freedesktop.org/pkg-config, attemptin g MIRRORS if available WARNING: Failed to fetch URL https://downloads.sourceforge.net/libpng/zlib/1.2.1 1/zlib-1.2.11.tar.xz, attempting MIRRORS if available WARNING: Failed to fetch URL https://ftp.gnu.org/gnu/automake/automake-1.16.1.ta r.gz, attempting MIRRORS if available WARNING: Failed to fetch URL https://ftp.gnu.org/gnu/m4/m4-1.4.18.tar.gz, attemp ting MIRRORS if available WARNING: Failed to fetch URL git://git.savannah.gnu.org/config.git, attempting M IRRORS if available WARNING: Failed to fetch URL https://ftp.gnu.org/gnu/libtool/libtool-2.4.6.tar.g z, attempting MIRRORS if available WARNING: Failed to fetch URL https://gmplib.org/download/gmp/gmp-6.1.2.tar.bz2, attempting MIRRORS if available WARNING: Failed to fetch URL https://download.savannah.gnu.org/releases/quilt/qu ilt-0.66.tar.gz, attempting MIRRORS if available WARNING: Failed to fetch URL https://tukaani.org/xz/xz-5.2.4.tar.gz, attempting MIRRORS if available

图 6.3.17 配置内核时报错

6.3.6 配置 Linux 根文件系统

在终端输入下面的命令可配置根文件系统,如果不需要配置可不执行该命令。

petalinux-config -c rootfs

下图就是根文件系统的配置界面:



图 6.3.18 根文件系统的配置界面

默认配置可满足一般使用,也可以根据需求来定制根文件系统,本实验保持默认配置。 需要说明的是"PetaLinux RootFS Settings"可以用来设置 root 用户的密码,默认为"root"。 后面登录的时候会用到。

保存配置并退出。

6.3.7 配置设备树文件

关于设备树的概念,这里先不做介绍。设备树的概念源自于 Linux 内核当中,当然其实 在 U-Boot 当中也已经使用了。如果需要配置设备树,可以编辑当前 petalinux 工程目录下的 project-spec/meta-user/recipes-bsp/device-tree/files/system-user.dtsi 文件。

我们可以打开这个文件进行编辑,将一些简单外设添加到系统当中,譬如按键、led和 IIC 设备。设备树用于保存 Linux 系统中的各种设备信息,内核在启动过程当中会去解析设备 树文件,获取设备所需的配置信息完成设备的初始化工作。

设备树的概念以及相关配置、语法涉及到了 Linux 内核驱动相关知识,并不是本篇学习 的重点,所以这里并不会去深入给大家介绍,将会在 Linux 驱动部分的章节做详细解说。

使用 vi 或 gedit 命令打开 system-user.dtsi 文件,如下所示:

vi project-spec/meta-user/recipes-bsp/device-tree/files/system-user.dtsi 默认的文件内容如下,可见该文件需要我们自己手动配置。

/include/ "system-conf.dtsi"
/ {
}:

图 6.3.19 system-conf.dtsi 文件初始内容

我们把按键、led 灯、蜂鸣器、USB、串口、sd 卡、Flash、PL 端网口、PS 端网口、 EEPROM 和 RTC 这两个 IIC 设备添加到 system-user.dtsi 设备树当中, system-user.dtsi 文件内 容如下:



```
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

/include/"system-conf.dtsi"

#include <dt-bindings/gpio/gpio.h>

#include <dt-bindings/input/input.h>

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

#include <dt-bindings/phy/phy.h>

/ {

model = "Alientek Navigator Zynq Development Board";

compatible = "xlnx,zynq-zc702", "xlnx,zynq-7000";

leds {

compatible = "gpio-leds";

gpio-led1 {

label = "led2";

gpios = <&gpio0 0 GPIO_ACTIVE_HIGH>;

default-state = "on";
```

```
};
```

```
gpio-led2 {
  label = "led1";
  gpios = <&gpio0 54 GPIO_ACTIVE_HIGH>;
  linux,default-trigger = "heartbeat";
};
gpio-led3 {
  label = "pl_led0";
  gpios = <&axi_gpio_0 0 0 GPIO_ACTIVE_HIGH>;
  default-state = "on";
};
gpio-led4 {
  label = "pl_led1";
  gpios = <&axi_gpio_0 1 0 GPIO_ACTIVE_HIGH>;
  linux,default-trigger = "timer";
};
gpio-led5 {
```

```
label = "ps_led0";
gpios = <&gpio0 7 GPIO_ACTIVE_HIGH>;
default-state = "on";
```



```
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
        };
        gpio-led6 {
          label = "ps_led1";
          gpios = <&gpio0 8 GPIO_ACTIVE_HIGH>;
          linux,default-trigger = "timer";
        };
      };
      keys {
        compatible = "gpio-keys";
        autorepeat;
        gpio-key1 {
          label = "pl_key0";
          gpios = <&gpio0 55 GPIO_ACTIVE_LOW>;
          linux,code = <KEY_LEFT>;
          gpio-key,wakeup;
          autorepeat;
        };
        gpio-key2 {
          label = "pl_key1";
          gpios = <&gpio0 56 GPIO_ACTIVE_LOW>;
          linux,code = <KEY_RIGHT>;
          gpio-key,wakeup;
          autorepeat;
        };
        gpio-key3 {
          label = "ps_key1";
          gpios = <&gpio0 12 GPIO_ACTIVE_LOW>;
          linux,code = <KEY_UP>;
          gpio-key,wakeup;
          autorepeat;
        };
        gpio-key4 {
          label = "ps_key2";
          gpios = <&gpio0 11 GPIO_ACTIVE_LOW>;
          linux,code = <KEY_DOWN>;
```



```
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
          gpio-key,wakeup;
          autorepeat;
        };
        touch-key {
          label = "touch_key";
          gpios = <&gpio0 57 GPIO_ACTIVE_HIGH>;
          linux,code = <KEY_ENTER>;
          gpio-key,wakeup;
          autorepeat;
        };
      };
      beep {
        compatible = "gpio-beeper";
        gpios = <&gpio0 58 GPIO_ACTIVE_HIGH>;
      };
      usb_phy0: phy0@e0002000 {
        compatible = "ulpi-phy";
        \#phy-cells = <0>;
        reg = <0xe0002000 0x1000>;
        view-port = <0x0170>;
        drv-vbus;
      };
    };
    &uart0 {
      u-boot,dm-pre-reloc;
      status = "okay";
    };
    &sdhci0 {
      u-boot,dm-pre-reloc;
      status = "okay";
    };
    &usb0 {
      dr_mode = "otg";
      usb-phy = <&usb_phy0>;
    };
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

```
&qspi {
  u-boot,dm-pre-reloc;
  flash@0 { /* 16 MB */
    compatible = "w25q256", "jedec, spi-nor";
    reg = <0x0>;
    spi-max-frequency = <50000000>;
    #address-cells = <1>;
    \#size-cells = <1>;
    partition@0x00000000 {
      label = "boot";
      reg = <0x0000000 0x00100000>;
    };
    partition@0x00100000 {
      label = "bootenv";
      reg = <0x00100000 0x00020000>;
    };
    partition@0x00120000 {
      label = "bitstream";
      reg = <0x00120000 0x00400000>;
    };
    partition@0x00520000 {
      label = "device-tree";
      reg = <0x00520000 0x00020000>;
    };
    partition@0x00540000 {
      label = "kernel";
      reg = <0x00540000 0x00500000>;
    };
    partition@0x00A40000 {
      label = "space";
      reg = <0x00A40000 0x00000000;
    };
  };
};
&gem0 {
  local-mac-address = [00 0a 35 00 8b 87];
  phy-handle = <&ethernet_phy>;
  ethernet_phy: ethernet-phy@7 { /* yt8521 */
```



```
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
        reg = <0x7>;
        device_type = "ethernet-phy";
      };
    };
    &gem1 {
      local-mac-address = [00 0a 35 00 11 55];
      phy-reset-gpio = <&gpio0 63 GPIO_ACTIVE_LOW>;
      phy-reset-active-low;
      phy-handle = <&pl_phy>;
      pl_phy: pl_phy@4 {
        reg = <0x4>;
        device_type = "ethernet-phy";
      };
    };
    &watchdog0 {
      status = "okay";
      reset-on-timeout;
                         // Enable watchdog reset function
    };
    &adc {
      status = "okay";
      xlnx,channels {
        #address-cells = <1>;
        #size-cells = <0>;
        channel@0 {
          reg = <0>;
        };
      };
    };
    &i2c0 {
      clock-frequency = <100000>;
      eeprom@50 {
        compatible = "atmel,24c64";
        reg = <0x50>;
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

```
pagesize = <32>;

};

rtc@51 {

compatible = "nxp,pcf8563";

reg = <0x51>;

};

};

设备树配置内容在我们的资料盘当中已经提供了,路径为: "开发板资料盘(A
```

反 奋 树 配 直 內 谷 住 找 们 的 贡 科 盈 当 中 C 经 提 供 」, 路 径 万: 开 友 板 贡 科 盈 (A 盘)\4_SourceCode\3_Embedded_Linux\zynq_petalinux\zynq7020\1_customize_linux\ devicetree\system-user.dtsi",大家可以用资料盘中的文件替换当前 Petalinux 工程中的该文件。

下面简单的讲解下 gpio led 的配置。

6个 gpio led 灯分别对应开发板的 6个 led 灯,配置信息主要包括 compatible (用于与内核 驱动匹配的名字)、label (别名)、gpios (对应的 GPIO 管脚)、默认状态以及触发状态。

例如 linux,default-trigger = " timer "表示默认的触发状态是 timer 模式,也可以改为 heartbeat 模式。除此之外,还有其他一些内核定义好的触发状态; default-state = "on"表示默认 led 灯是亮着的; gpios = <& gpio0 0 GPIO_ACTIVE_HIGH>表示该 led 的控制管脚是 gpio-mio0, GPIO_ACTIVE_HIGH 表示高电平有效(也就是高电平的时候 led 灯才会亮)。

其他的就不讲解了,后面驱动部分文档中会详细讲解。

内容编辑完成之后保存退出即可。

6.3.8 编译 Petalinux 工程

现在我们就可以编译整个 Petalinux 工程了,在终端输入如下命令:

petalinux-build

该命令将生成设备树 DTB 文件、fsbl 文件、U-Boot 文件、boot.scr 文件、Linux 内核和根 文件系统文件。编译完成后,生成的镜像文件将位于工程的 images 目录下。需要说明的是 fsbl、U-Boot 这两个我们在工程中并没有配置,这是因为 Petalinux 会根据 xsa 文件和 6.3.4 节 的配置 petalinux 工程自动配置 fsbl 和 uboot,如无特需要求,不需要再手动配置。

执行结果如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php sqd@sqd-virtual-machine:~/petalinux/ALIENTEK-ZYNQ\$ petalinux-build INFO: Sourcing build tools [INFO] Building project [INF0] Sourcing build environment [INFO] Generating workspace directory INFO: bitbake petalinux-image-minimal Loaded 4264 entries from dependency cache. Parsing of 2995 .bb files complete (2993 cached, 2 parsed). 4265 targets, 204 sk ipped, 0 masked, 0 errors. **NOTE:** Resolving any missing task queue dependencies ete) **NOTE:** Executing Tasks NOTE: linux-xlnx: compiling from external source tree /home/sgd/petalinux/ALIENT EK-ZYNQ/components/yocto/workspace/sources/linux-xlnx NOTE: Setscene tasks completed NOTE: Tasks Summary: Attempted 3509 tasks of which 2621 didn't need to be rerun and all succeeded. INFO: Successfully copied built images to tftp dir: /tftpboot [INFO] Successfully built project

图 6.3.20 编译整个 Petalinux 工程

从上图可以看出,工程已经成功编译。如果编译遇到报错,请参考扩展阅读部分。

6.3.9 制作 BOOT.BIN 启动文件

Petalinux 提供了 petalinux-package 命令将 PetaLinux 项目打包为适合部署的格式,其中 "petalinux-package --boot"命令生成可引导映像,该映像可直接与 Zynq 系列设备(包括 Zynq-7000和 Zynq UltraScale + MPSoC)或基于 MicroBlaze 的 FPGA 设计一起使用。对于 Zynq 系列设备,可引导格式为 BOOT.BIN,可以从 SD 卡引导启动。对于基于 MicroBlaze 的设计,默认格式为 MCS PROM 文件,适用于通过 Vivado 或其他 PROM 编程器进行编程。

ZYNQ 的启动文件 BOOT.BIN 一般包含 fsbl 文件、bitstream 文件和 uboot 文件。使用下面 的命令可生成 BOOT.BIN 文件:

petalinux-package --boot --fsbl --fpga --u-boot --force

选项"--fsbl"用于指定 fsbl 文件所在位置,后面接文件对应的路径信息,如果不指定文件位置,默认对应的是 images/linux/zynq_fsbl.elf;选项"--fpga"用于指定 bitstream 文件所在位置,后面接该文件对应的路径信息,默认对应的是 images/linux/system.bit,实际可能有区别;选项"--u-boot"用于指定 U-Boot 文件所在位置,后面接该文件所在路径信息,默认为 images/linux/u-boot.elf。这里笔者均没有指定对应的文件的路径信息,那么 Petalinux 会自动使用默认文件。执行结果如下图所示:



尽丁可任线教子:www.yuanzige.com	形式:www.openeav.com/forum.pnp
<pre>sqd@sqd-virtual-machine:~/petalinux/ALIENTEK-; INFO: Sourcing build tools</pre>	ZYNQ\$ petalinux-packagebootfsblfpgau-bootforce
INFO: File in BOOT BIN: "/home/sqd/petalinux/ INFO: File in BOOT BIN: "/home/sqd/petalinux/	ALIENTEK-ZYNQ/images/linux/zynq_fsbl.elf" ALIENTEK-ZYNO/project-spec/hw-description/system_wrapper.bit"
INFO: File in BOOT BIN: "/home/sqd/petalinux/	ALIENTEK-ZYNQ/images/linux/u-boot.elf" ALIENTEK-ZYNQ/images/linux/system dth"
INFO: Generating Zynq binary package BOOT.BIN	···
***** Xilinx Bootgen v2020.2 **** Build date : Nov 15 2020-06:11:24	
** Copyright 1986-2020 Xilinx, Inc. All R	ights Reserved.
[INFO] : Bootimage generated successfully	
INFO: Binary is ready.	

图 6.3.21 生成 BOOT 文件

生成的 BOOT.BIN 文件放在 Petalinux 工程的 images/linux 目录下,上一小节编译 Petalinux 工程生成的文件同样存放在 images/linux 目录下,如下图所示:

<pre>sqd@sqd-virtual-machine:~/petalinux/ALIENTEK-ZYNQ/images/linux\$ ls -l</pre>								
总用量 8626	50							
- rw- rw- r	1	sqd	sqd	4925156	3月	13	16:08	BOOT.BIN
- rw- r r	1	sqd	sqd	2010	3月	13	15:46	boot.scr 🛻
- rw- r r	1	sqd	sqd	11568184	3月	13	15:49	image.ub 👞
drwxr-xr-x	2	sqd	sqd	4096	3月	13	15:46	pxelinux.cfg
- rw- r r	1	sqd	sqd	13970432	3月	13	15:49	rootfs.cpio
- rw- r r	1	sqd	sqd	7211071	3月	13	15:49	rootfs.cpio.gz
- rw- r r	1	sqd	sqd	7211135	3月	13	15:49	rootfs.cpio.gz.u-boot
- rw- r r	1	sqd	sqd	8650752	3月	13	15:49	rootfs.jffs2
- rw- r r	1	sqd	sqd	6487	3月	13	15:49	rootfs.manifest
- rw- r r	1	sqd	sqd	7235523	3月	13	15:49	rootfs.tar.gz
- rw- r r	1	sqd	sqd	4045674	3月	13	11:19	system.bit 🔶
- rw- r r	1	sqd	sqd	29025	3月	13	15:46	system.dtb
- rw- r r	1	sqd	sqd	746277	3月	13	15:47	u-boot.bin
- rw- r r	1	sqd	sqd	812240	3月	13	15:47	u-boot.elf ←
- rw- r r	1	sqd	sqd	4326048	3月	13	15:49	uImage
- rw- r r	1	sqd	sqd	12671032	3月	13	15:49	vmlinux
- rw- r r	1	sqd	sqd	4325984	3月	13	15:49	zImage
- rw-rr	1	sqd	sqd	551832	3月	13	15:47	zynq_fsbl.elf 🖡

图 6.3.22 编译工程生成的启动镜像文件

6.3.10 制作 SD 启动卡

如果使用 SD 卡引导 linux 系统启动,一般需要在 SD 卡上有 2 个分区。一个分区使用 FAT32文件系统,用于放置启动镜像文件(如 BOOT.BIN, boot.scr 文件和 linux 镜像文件等), 另一分区使用 EXT4 文件系统,用于存放根文件系统。

需要说明的是在 6.2.4 节配置 petalinux 工程中, "Image Packaging Configuration"子菜单 根文件系统的类型的配置使用的是默认的 INITRD,所以只需要一个使用 FAT32 文件系统的 分区就可以了。当设置为"EXT4"则需要另一个存放根文件系统的分区。

本小节先讲解 SD 卡的分区和格式化,然后说明将哪些文件复制到 SD 卡中。此处的 SD 卡指的是那种小卡,也称为 TF 卡。

注: 在使用 SD 卡前需要先将 sd 卡中的数据做备份, 否则会丢失 SD 卡中的数据。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

将 SD 卡插入到读卡器中、并将读卡器插入电脑并连接到 Ubuntu 系统,在 Ubuntu 系统中 找到 SD 卡所对应的设备节点,笔者插入的 SD 卡对应的设备节点为/dev/sdb。在终端中输入如 下命令:

umount /dev/sdb*
sudo fdisk /dev/sdb
输入"p"执行结果如下图所示:
<pre>sqd@sqd-virtual-machine:~\$ umount /dev/sdb* </pre>
umount: /dev/sdb: not mounted.
squesqu-virtuat-machine:~\$ sudo ruisk /dev/sub 🚛
欢迎使用 fdisk (util-linux 2.31.1)。
更
命令(输入 m 获取帮助): p Dick (dow/cdb:14.0.CiP 15021520456 安英 21116288 会自区
单元: 扇区 / 1 * 512 = 512 字节
扇区大小(逻辑/物理): 512 字节 / 512 字节
I/0 大小(最小/最佳): 512 字节 / 512 字节
磁盘标签类型: dos 磁盘标记符: 0vb8/f7531
设备 启动 起点 末尾 扇区 大小 Id 类型
/dev/sdb1 * 2048 1026047 1024000 500M c W95 FAT32 (LBA)
/dev/sub2 1020048 3111028/ 30090240 14.46 / HPFS/NTFS/EXFAT
命令(输入 m 获取帮助):

图 6.3.23 查看 SD 卡分区

可以看到当前的分区表,有两个分区,一个 FAT32 的分区和一个 exFAT 分区。在开始新 分区之前需要将以前的分区删除,键入"d",然后输入1,删除1分区,再次键入"d"删除 第2个分区,如果读者的 SD 卡原本就只有一个分区,前面只需执行一次删除就行。当再次键 入"d"并出现提示"还没有定义分区!"时,表明已无存在的分区,如图 6.3.24 中红色字体 提示所示。

正点原子 领航者 ZYNQ 之嵌入式 Linux 开发指南 原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php 命令(输入 m 获取帮助): р Disk /dev/sdb: 14.9 GiB, 15931539456 字节, 31116288 个扇区 单元: 扇区 / 1 * 512 = 512 字节 扇区大小(逻辑/物理): 512 字节 / 512 字节 I/0 大小(最小/最佳): 512 字节 / 512 字节 磁盘标签类型: dos 磁 盘 标 识 符: 0xb84f7531 设备 启动 起点 末尾 扇区 大小 Id 类型 2048 1026047 1024000 500M c W95 FAT32 (LBA) /dev/sdb1 * 1026048 31116287 30090240 14.4G 7 HPFS/NTFS/exFAT <u>分</u>区 /dev/sdb2 命令(输入 m 获取帮助): d 分区号 (1,2, 默认 2):1 分区 1 已删除。 命令(输入 m 获取帮助): d 已选择分区 2 分区 2 已删除。 命令(输入 m 获取帮助): d 命令(输入 m 获取帮助):

图 6.3.24 将以前的分区删除

下面开始新建分区。输入"n"创建一个新分区。通过选择"p"使其为主,使用默认分区号 1 和第一个扇区 2048。设置最后一个扇区,也就是设置第一个分区的大小,一般设置 500M 足够了,通过输入"+500M",为该分区预留 500MB,如果提示分区包含 vfat 签名并询问是否移除该签名,则输入"y",如下图所示:

命令(输入 m 获取帮助): n ——
分区类型
p 主分区(0个主分区,0个扩展分区,4空闲)
e 扩展分区(逻辑分区容器)
选择 (默认 p): p 🚛
分区号 (1-4, 默认 1): 1 ┥
第一个扇区(2048-31116287,默认 2048): 🔶
上个扇区,+sectors 或 +size{K,M,G,T,P}(2048-31116287, 默认 31116287): +500M —
创建了一个新分区 1, 类型为"Linux", 大小为 500 MiB。
分区 #1 包含一个 VTat 金名。
你 相 移 吟 达 饺 夕 回 ? 旦 [V] /不 [N] • _ v ,
E) 命令 炮移 险 该 签 夕
命令(输入 m 获取帮助):

图 6.3.25 创建第一个新分区

现在设置分区类型,输入"t",然后输入"c",设置为"W95FAT32(LBA)",如下图 所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

命令(输入 m 获取帮助): t ◀—— 已选择分区 1 Hex 代码(输入 L 列出所有代码): c ◀—— 已将分区"Linux"的类型更改为"W95 FAT32 (LBA)"。 命令(输入 m 获取帮助): ■

图 6.3.26 设置分区类型

输入"a",设为引导分区,如下图所示:

命令(输入 m 获取帮助): a ____ 已选择分区 1 分区 1 的 可启动 标志已启用。

图 6.3.27 设为引导分区

第一个分区就创建好了,开始创建第二个分区。 通过键入"n"来创建根文件系统分区。后面一路默认就可以了,如下图所示:

命令(输入 m 获取帮助): n
p 主分区 (1个主分区,0个扩展分区,3空闲)
e 扩展分区 (逻辑分区容器)
选择 (默认 p):
将使用默认回应 p。
分区号 (2-4, 默认 2):
第一个扇区 (1026048-31116287, 默认 1026048):
上个扇区, +sectors 或 +size{K,M,G,T,P} (1026048-31116287, 默认 31116287):
创建了一个新分区 2, 类型为"Linux",大小为 14.4 GiB。
劳区 #2 包含一个 ntfs 签名。
您想移除该签名吗? 是[Y]/否[N]: y
写入命令将移除该签名。
命令(输入 m 获取帮助):

图 6.3.28 创建第二个分区

如果现在输入"p"检查分区表,会看到刚刚创建的 2 个分区。如果没问题,键入"w" 以写入到 SD 卡并退出。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

命令(输入 m 获取帮助): p Disk /dev/sdb: 14.9 GiB, 15931539456 字节, 31116288 个扇区 单元:扇区 / 1 * 512 = 512 字节 扇区大小(逻辑/物理): 512 字节 / 512 字节 I/0 大小(最小/最佳): 512 字节 / 512 字节 磁盘标签类型: dos 磁盘标识符: 0xb84f7531 设备 启动 起点 末尾 扇区 大小 Id 类型 (day/sdb1 * 2048 1026047 1024000 500M s 2005 54722

/dev/sdb1 * 2048 1026047 1024000 500M c W95 FAT32 (LBA) /dev/sdb2 1026048 31116287 30090240 14.4G 83 Linux

Filesystem/RAID signature on partition 1 will be wiped. Filesystem/RAID signature on partition 2 will be wiped.

命令(输入 m 获取帮助): w ◀—— 分区表已调整。 将调用 ioctl() 来重新读分区表。 正在同步磁盘。

图 6.3.29 写入分区到 SD 卡

完成了分区创建后,就可以格式化分区了。在终端输入如下命令:

sudo mkfs.vfat -F 32 -n boot /dev/sdb1

sudo mkfs.ext4 -L rootfs /dev/sdb2

将第一个分区格式化成 FAT32 分区并命名为 boot,将第二个分区格式化成 ext4 分区并命 名为 rootfs。执行结果如下图所示:

图 6.3.30 SD 卡分区格式化

格式化分区之后就可以挂载分区了(重新插拔读卡器或者使用 mount 命令进行挂载)。 挂载完成后,我们将该工程 image/linux 目录下的 BOOT.BIN、boot.scr 和 image.ub 文件拷贝到 名为 boot 的分区也即/dev/sdb1 分区中,结果如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

> 🖪 🗖 boot					
最近使用		1 10 101	1 10 101	1 10 101	
主目录	System	1010 boot.scr	BOOT.BIN	image.ub	
桌面	Volume				J
视频	n				
图片					
文档					
下载					
音乐					
回收站					
boot 🔺					

图 6.3.31 拷贝启动镜像到第一个分区

本实验只需要这三个文件即可,现在可以使用 umount 命令卸载 SD 卡了。

6.3.11 开发板启动模式设置

将 SD 卡插入领航者开发板的 SD 卡槽, 然后使用 USB Type-C 连接线将开发板左侧的 USB_UART 接口与电脑连接,用于串口通信。领航者开发板连接 SD (TF)卡的正面图如下 图所示:



图 6.3.32 开发板连接 tf 卡和串口

接下来**将领航者底板上的启动模式开关 BOOT_CFG 的两个开关均拨到下面(都置为** OFF),即设置为从 SD 卡启动。不同的启动方式与两个拨码开关的状态对应关系如下图所示:

原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php

正点原子

	MIO 4	MIO 5
BOOT_CFG	1	2
JTAG	ON	ON
NAND	OFF	ON
QSPI	ON	OFF
SD Card	OFF	OFF

图 6.3.33 启动模式设置

最后连接开发板的电源线给开发板上电。

6.3.12 打开串口上位机,进入 Linux 系统

打开 MobaXterm 串口上位机或其它串口上位机。上位机打印 Linux 启动信息如下:



图 6.3.34 打印 Linux 启动信息

停留在登录处,此处使用 root 用户登录,登录密码为 "root" (去掉双引号),登录进去 后,界面如下:





图 6.3.35 使用 root 用户登录

如果我们把视线移到开发板,会看到板上的 LED 灯全都是亮的,其中,底板上的 PL_LED1和PS_LED1同频闪烁,这是设备树配置文件产生的结果,还记得那个"timer"么。

至此我们就走完了 Petalinux 开发 Linux 的整个流程。正如俗语所说师傅领进门,修行靠 个人,Petalinux 的功能远不止如此,其他功能读者有兴趣可进行探索,建议参考 ug1144 参考 手册,也就是 Xilinx 官方编写的 PetaLinux 工具使用说明文档,已提供在开发板光盘资料:开 发板资料盘(A盘)\8_ZYNQ&FPGA参考资料\Xilinx\User Guide\ug1144-petalinux-tools-reference-guide.pdf。

特别说明:以后我们使用主机终端指代电脑上 Ubuntu 系统的终端,串口终端指代通过串口线连接到开发板显示在串口上位机中的终端。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

6.4 解决 Petalinux 由于网络原因产生的编译错误

6.4.1 没有找到合适的 staging package:

如果编译过程中报如图 6.4.1 所示 "libxau-1_1.0.9" 或者图 6.4.2 所示 "autoconf-native-2.69-r11"包找不到或者错误,这是因为网络不好或者提供的网址无法访问的原因,导致编译 时需要的暂存包无法获取。

ERROR libyau-1 1 0 9-r0 do package write rpm setscepe. Frror everyting a pythom
function in ever withon funct) autogenerated:
Tunetton in exec_python_rune() autogenerated.
The stack trace of python calls that resulted in this excention/failure was:
File: 'ever python func() sutogenerated' lineno; 2 function; andules
oppi.
0001:
0002:00_package_writte_rpiii_setscene(u)
File: '/home/zy/petalinux/wavigator_/010_v3/ALIEWTEK-2YWQ/components/yocto/tayer
s/core/meta/classes/package_rpm.bbclass', lineno: /31, function: do_package_writ
e_rpm_setscene
0727:# but we need to stop the rootfs/solver from running while we do
<pre>0728:do_package_write_rpm[sstate-lockfile-shared] += "\${DEPLOY_DIR_RPM}/rpm</pre>
,lock"
0729:
0730:python do_package_write_rpm_setscene () {
*** 0731: sstate_setscene(d)
0732:}
0733:addtask do_package_write_rpm_setscene
0734:
0735:python do_package_write_rpm () {
File: '/home/zy/petalinux/Navigator_7010_v3/ALIENTEK-ZYNQ/components/yocto/layer
<pre>s/core/meta/classes/sstate.bbclass', lineno: 744, function: sstate_setscene</pre>
0740: pass
0741:
0742:def sstate setscene(d):
0743: shared state = sstate state fromvars(d)
<pre>*** 0744: accelerate = sstate installpkg(shared state, d)</pre>
0745: if not accelerate:
0746: bb.fatal "No suitable staging package found")
0747:

图 6.4.1 缺少暂存包

A.S. And IN LIA



领航清	皆 ZYI	NQ 之嵌入录	Linux	开发指南	菊	E	 			
原子哥	在线教	学: www.yuan	zige.com	论坛:	www.openedv	.com/forum.php				
ER na 69 _p at f0 s	ROR: au ble to :r11:x8 opulate ive:x86 5d25fa9 earched /home						ailure) U H-linux:2. 662ce84194 autoconf-n aabbf9e8fb that were cate-cache			
ER	/home ROR: au					lriver <u>/build/sst</u> cene: <mark>No suita</mark> t	t <u>ate-cache</u> Dle stagin			
g ER EK po	package ROR: Lo -ZYNQ-d pulate_						v3/ALIENT p/log.do_			
图 6.4.2 缺少暂存包										
我		们下	IJ	以	到	xilinx	官	XX		
https://c	<u>china.xil</u>	inx.com/suppor	t/download	/index.htm	<u>l/content/xilinx</u>	/zh/downloadNa	v/embedded-	-		
<u>design-tools/archive.html</u> 下载 2020.2版本 petalinux 对应的"sstate-cache"如下图所示:										
	PetaLinux 工具 sstate-cache 工件 - 2020.2									
		重要信息								
		只有在 PetaLir	nux 工具 / BS	P 构建无法词	方问因特网时,才	需要使用				
		sstate-cache 肴 无关。	问下载。sstai	te-cache 文作	H按架构提供,提	供的下载与架构				
		请参阅 READM	IE, 了解有关	如何按架构	使用 sstate-cache	的更多详情。				
		🛓 aarch64 ssta	te-cache (TAR/	/GZIP - 25.88 (GB)					
		MD5 SUM Value	: d7e3fddb914	f6db1c91159	105f80eb85					

▲ arm sstate-cache (TAR/GZIP - 9.09 GB)

MD5 SUM Value : c75f23b5d9685ed20c572a5731a776ec

图 6.4.3 暂存包 sstate-cache

暂存包中包含了 Petalinux 编译时需要的各种软件包和库。该暂存包在资料盘中已经提供, 路径为"资料盘(B盘)/SState-cache/sstate_arm_2020.2.tar.gz"。下载完成后解压到 Ubuntu 和 Windows 的共享文件夹中,如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

share18 > sstate > sstate_arm_2020.2 > arm >

名称	修改日期	类型
universal-4.8	2023/5/5 13:57	文件夹
📊 universal	2023/5/5 13:57	文件夹
📙 ff	2023/5/5 13:57	文件夹
📊 fe	2023/5/5 13:57	文件夹
📙 fd	2023/5/5 13:57	文件夹
📙 fc	2023/5/5 13:57	文件夹
📙 fb	2023/5/5 13:57	文件夹
📙 fa	2023/5/5 13:57	文件夹
<mark></mark> f9	2023/5/5 13:57	文件夹
f8	2023/5/5 13:57	文仕中

图 6.4.4 解压后的文件

从 Ubuntu 中访问该文件夹路径为"/mnt/hgfs/share18/sstate/sstate_arm_2020.2/arm"。 下面将解压后的路径添加到 Petalinux 工程中。

进入 Petalinux 工程, 输入 "petalinux-config" 配置工程, 配置"Yocto Settings --->Local sstate feeds settings--->local sstate feeds url",添加解压后文件的路径,添加格式为"<路径>"。 对于笔者而言,路径为"/mnt/hgfs/share18/sstate/sstate arm 2020.2/arm",如下图所示:



图 6.4.5 添加解压后包文件路径

保存配置,返回到"Yocto Settings"界面,取消"Enable Network sstate feeds"使能,如 下图所示:



图 6.4.6 取消通过网络获取暂存包



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

保存配置并退出。

重新编译就不会再报"libxau-1_1.0.9"包相关的错误。

6.4.2 Fetcher failure:Unable to find file 错误

如果编译工程遇到如下图所示的"Fetcher failure for URL"或相似错误

Cloning into bare repository '/home/cx/workspace/petalinux/ALIENTEK-ZYNQ/build/d
ownloads/git2/github.com.Xilinx.linux-xlnx.git'...
fatal: unable to access 'https://github.com/Xilinx/linux-xlnx.git/': Empty reply
from server
ERROR: Fetcher failure for URL: 'git://github.com/Xilinx/linux-xlnx.git;protocol
=https;name=machine;branch=xlnx_rebase_v5.4'. Unable to fetch URL from any sourc
e.
ERROR: Logfile of failure stored in: /home/cx/workspace/petalinux/ALIENTEK-ZYNQ/
build/tmp/work/zynqmp_generic-xilinx-linux/linux-xlnx/5.4+gitAUTOINC+62ea514294r0/devtooltmp-8lylrfrk/temp/log.do_fetch.13346
NOTE: Tasks Summary: Attempted 20 tasks of which 0 didn't need to be rerun and 4
failed.
ERROR: Extracting source for linux-xlnx failed
ERROR: Failed to config kernel.

图 6.4.7 Fetcher failure:Unable to find file 错误

出现这种错误的原因是 Petalinux 在配置和编译的时候,需要联网下载一些文件,由于网络原因这些文件不能正常下载,导致编译出错。解决的方法如下:

到 Xilinx 官 网 https://china.xilinx.com/support/download/index.html/content/xilinx/zh/downloadNav/embeddeddesign-tools/archive.html 下载 2020.2 版本 petalinux 对应的"下载"压缩包,如下图所示:

🛓 下载 (TAR/GZIP - 36.01 GB)

MD5 SUM Value : 67e3547808cbfdf9a4f21c481dd532d2

图 6.4.8 Downloads 下载包

同样, 该下载包在资料盘中已经提供, 路径为"资料盘(B盘)/Downloads/downloads_2020.2.tar.gz"。下载完成后解压到 Ubuntu 和 Windows 的共享文件夹中, 如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

harelo > downloads > downloads		~ 0
名称	修改日期	类型
git2	2023/5/5 15:43	文件夹
uninative	2023/5/5 15:43	文件夹
xsct	2023/5/5 15:43	文件夹
acl-2.2.52.src.tar.gz	2020/11/17 17:51	WinRAR 压缩文件
acl-2.2.52.src.tar.gz.done	2020/11/17 17:51	DONE 文件
acpica-unix2-20190816.tar.gz	2020/11/17 17:51	WinRAR 压缩文件
adwaita-icon-theme-3.32.0.tar.xz	2020/11/17 17:51	WinRAR 压缩文件
🚍 alsa-lib-1.1.9.tar.bz2	2020/11/17 17:51	WinRAR 压缩文件
alsa-plugins-1.1.9.tar.bz2	2020/11/17 17:51	WinRAR 压缩文件
alsa-tools-1.1.7.tar.bz2	2020/11/17 17:51	WinRAR 压缩文件
alca-utile_1 1 0 tar bz?	2020/11/17 17-51	WinRAR 压缩文件

图 6.4.9 解压后的 downloads 包文件

从 Ubuntu 虚拟机中访问该文件夹路径为"/mnt/hgfs/share18/downloads/downloads"。下面将解压后的路径添加到 Petalinux 工程中。

进入 Petalinux 工程,输入 "petalinux-config" 配置工程,配置 "Yocto Settings --->Add pre-mirror url",删除原来的内容,添加 downloads 包文件路径,添加格式为 "file://<路径>"。 对于笔者而言,路径为 "file:///mnt/hgfs/share18/downloads/downloads",如下图所示:



图 6.4.10 添加 pre-mirror url

保存配置,返回到 Yocto Settings 界面,使能"Enable BB NO NETWORK",如下图所示:



图 6.4.11 使能 BB NO NETWORK

保存配置并退出。

重新编译就不会再报"Fetcher failure:Unable to find file"相关的错误。



原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php

6.4.3 获取 qemu-xilinx-system-native 失败

编译时遇到 qemu-xilinx-system-native 包获取失败,如下图所示:

ERROR: [qemu-xilinx-system-native-v5.1.0] xilinx-v2020.2+gitAUTOINC+7e3e3ae09a-r0
do_fetch: Fetcher failure for URL: 'gitsm://github.com/Xilinx/qemu.git;protocol=
https;branch=branch/xilinx-v2020.2'. Unable to fetch URL from any source.
ERROR: Logfile of failure stored in: /home/zy/petalinux/Navigator_7010_v3/ALIENT
EK-ZYNQ/build/tmp/work/x86_64-linux/qemu-xilinx-system-native/v5.1.0-xilinx-v202
0.2+gitAUTOINC+7e3e3ae09a-r0/temp/log.do_fetch.17958
ERROR: Task (/home/zy/petalinux/Navigator_7010_v3/ALIENTEK-ZYNQ/components/yocto
/layers/meta-xilinx/meta-xilinx-bsp/recipes-devtools/qemu/qemu-xilinx-system-native/v5.1.0.



解决这种错误方法如下:

进入 Petalinux 工程,编辑工程下的 project-spec/meta-user/conf/petalinuxbsp.conf 文件,在 文件末尾添加如下内容:

 $PREMIRRORS_prepend = " \ \ \ \\$

git://.*/.* file:///mnt/hgfs/share18/downloads/downloads $\n \$

gitsm://.*/.* file:///mnt/hgfs/share18/downloads/downloads $\n \$

ftp://.*/.* file:///mnt/hgfs/share18/downloads/downloads $\n \$

 $\label{eq:http://.*/.* file:///mnt/hgfs/share18/downloads/downloads \n \$

 $https://.*/.*\ file:///mnt/hgfs/share18/downloads/downloads \n''$

文件中 file:///mnt/hgfs/share18/downloads/downloads 要与 6.4.2 小节 pre-mirror url 的路径相同。读者在添加时,需要将其替换成自己的包文件路径。结果如下图所示:

#User Configuration	
#OE_TERMINAL = "tmux"	
<pre>PREMIRRORS_prepend = " \ git://.*/.* file:///mnt/hgfs/share18/downloads/downloads \n ' gitsm://.*/.* file:///mnt/hgfs/share18/downloads/downloads \n'</pre>	\ n \
ftp://.*/.* file:///mnt/hgfs/share18/downloads/downloads \n http://.*/.* file:///mnt/hgfs/share18/downloads/downloads \n https://.*/.* file:///mnt/hgfs/share18/downloads/downloads \n	\ \ n"

图 6.4.13 修改 petalinuxbsp.conf 文件

重新编译就不会出现 qemu 相关的报错。

6.5 扩展阅读

- 1) Xilinx 官方技术支持: <u>https://china.xilinx.com/support/service-portal/contact-support.html</u>
- 2) Petalinux 用户手册 UG1144: <u>Revision History PetaLinux Tools Documentation</u> <u>Reference Guide (UG1144) • 阅读器 • AMD 自适应计算文档门户 (xilinx.com)</u>


原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

- 3) Petalinux 工程目录结构详解: Petalinux 用户手册 <u>UG1144</u>的附录 B (Appendix B) — PetaLinux 工程结构
- 4) Zynq-7000 FSBL 介绍: Zynq-7000 FSBL Xilinx Wiki Confluence (atlassian.net)
- 5) Zynq-7000 资源汇总: Zynq-7000 Xilinx Wiki Confluence (atlassian.net)

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

正点原子

第七章 Linux 基础外设的使用

上一章我们成功使用 Petalinux 搭建了 Linux 系统,有了系统就可以在系统上使用相应外 设以及运行应用程序。本章我们学习如何通过 Linux 系统控制领航者开发板上的基础外设, 如 LED、按键、EEPROM、以太网等。这些外设都可以在终端通过 Shell 来控制,非常方便。



原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php

7.1 GPIO 之 LED 的使用

GPIO 驱动程序通过 sysfs 文件系统提供了用户空间对 GPIO 的访问,因而通过终端控制 LED 极其方便。我们启动领航者开发板,进入第六章定制的 Linux 系统后在串口终端中输入 如下命令进入到 sysfs 文件系统的 leds 接口处:

cd /sys/class/leds

ls

通过 ls 命令可以看到我们在设备树中配置的 led,如下图所示:

root@ALIENTEK-ZYNQ:~# cd /sys/class/leds
root@ALIENTEK-ZYNQ:/sys/class/leds# ls
led1 led2 mmc0:: mmc1:: pl_led0 pl_led1 ps_led0 ps_led1

图 7.1.1 查看 led

这 6 个 LED 对应开发板上的 6 个 LED 灯,名字的命名跟板子上对应 LED 丝印上的名字 是一样的。我们先来看下 leds 下的内容,以 ps_led1 为例,输入"ls ps_led1",执行结果如下图 所示:

prightness delay_on max_brightness subsystem ueve	t
delay_off device power trigger	
root@ALIENTEK-ZYNQ:/sys/class/leds#	

图 7.1.2 ps_led1 下的内容

可以看到有 brightness 和 trigger。Brightness 可以控制 led 灯的亮灭, trigger 可以选择触发方式。我们向 ps_led1 的 brightness 写入 0, 即输入 "echo 0 > ps_led1/brightness" 命令, 如下图所示:

root@ALIENTEK-ZYNQ:/sys/class/leds# echo 0 > ps_led1/brightness root@ALIENTEK-ZYNQ:/sys/class/leds# |

图 7.1.3 输入 "echo0>ps led1/brightness"命令

可以看到开发板上的 ps_led1 被熄灭,现在输入 "echo 1 > ps_led1/brightness" 命令,如下图所示:

root@ALIENTEK-ZYNQ:/sys/class/leds#
root@ALIENTEK-ZYNQ:/sys/class/leds# echo 1 > ps_led1/brightness
root@ALIENTEK-ZYNQ:/sys/class/leds# |

图 7.1.4 输入 "echo1>ps led1/brightness"命令

可以看到开发板上的 ps_led1 灯亮,这就是通过 brightness 来控制 led 灯的亮灭,其他 led 灯也可以以此种方式控制。

现在我们来看下如何通过 trigger 来控制 led。首先输入 "cat ps_led1/trigger" 命令看下有多少种触发方 式,如下图所示:

root@ALIENTEK-ZYNQ:/sys/class/leds# cat ps_led1/trigger [none] kbd-scrolllock kbd-numlock kbd-capslock kbd-kanalock kbd-shiftlock kbd-altgrlock kbd-ctrllock kbd-altlock kbd-shiftllock kbd-shiftrlock kbd-ctrlllock kbd-ctrlrlock mmc0 mmc1 timer oneshot heartbe at backlight gpio cpu cpu0 cpu1 default-on transient flash torch

图 7.1.5 查看触发方式

可以看到触发方式非常多,其中"[]"中的内容表示当前触发方式。可知当前触发方式 为 none,也就是表示无任何触发反应。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php 现在试一下 timer 触发, 输入命令 "echo timer > ps_led1/trigger", 如下图所示:

root@ALIENTEK-ZYNQ:/sys/class/leds#
root@ALIENTEK-ZYNQ:/sys/class/leds# echo timer > ps_led1/trigger
root@ALIENTEK-ZYNQ:/sys/class/leds# |

图 7.1.6 timer 触发

可以看到开发板上 ps_led1 灯以定时器的方式每秒闪烁一次,这就是 timer 触发方式的效果,与我们在 设备树中配置的效果一样。现在来看下 heartbeat 方式的效果,输入命令 "echo heartbeat > ps_led1/trigger",如下图所示:

root@ALIENTEK-ZYNQ:/sys/class/leds#
root@ALIENTEK-ZYNQ:/sys/class/leds# echo heartbeat > ps_led1/trigger
root@ALIENTEK-ZYNQ:/sys/class/leds# |

图 7.1.7heartbeat 方式

此时可以看到开发板上的 PS_LED1 灯像心跳一样的闪烁这就是 heartbeat 触发方式的效果。其它触发方 式,这里就不一一列举了,需要提醒的是并不是所有的触发方式都能有反应,必须满足相应触发条件才行, 有兴趣的读者可以自行探索。

7.2 GPIO 之按键的使用

GPIO 按键的使用非常简单,通过读取文件/dev/input/event0 可以获取由 GPIO 按键生成的按键事件。在 串口终端中输入"cat /dev/input/event0 | hexdump"命令,然后按下一个按键,输入事件将打印到控制 台;如下图所示:

root@AL	IENTE	(-ZYN):~# (cat /o	dev/ir	nput/e	event@	9 h	exdump		
0000000	b3e0	620d	0000	0000	75d4	0000	0000	0000			
0000010	0001	0069	0001	0000	b3e0	620d	0000	0000			
0000020	75d4	0000	0000	0000	0000	0000	0000	0000			
0000030	b3e0	620d	0000	0000	448f	0003	0000	0000			
0000040	0001	0069	0000	0000	b3e0	620d	0000	0000			
0000050	448f	0003	0000	0000	0000	0000	0000	0000			

图 7.2.1 获取按键事件

可以通过这种方式快速测试按键。

7.3 EEPROM 的使用

在 6.3.7 小节配置设备树的时候,我们在设备树文件中配置了两个 I2C 外设,一个是 EEPROM,另一个是 RTC。首先我们来看下如何读写 EEPROM。

在/sys/class/i2c-adapter 目录下有两个 I2C 总线控制器,如下图所示:

root@ALIENTEK-ZYNQ:~# cd /sys/class/i2c-adapter root@ALIENTEK-ZYNQ:/sys/class/i2c-adapter# ls i2c-1 i2c-2

图 7.3.1 I2C 总线控制器



subsystem uevent

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

i2c-1和i2c-2分别对应 ZYNQ PS 端的两个 I2C 总线控制器,开发板上的 EEPROM 挂载在i2c-1 总线下。我们进入到/sys/class/i2c-adapter/i2c-1/目录下,查看该目录下的内容,如下图所示:

root@ALIENTEK-ZYNQ:/sys/class/i2c-adapter# cd i2c-1 root@ALIENTEK-ZYNQ:/sys/class/i2c-adapter/i2c-1# ls 1-0050 delete_device i2c-dev new_device power uevent 1-0051 device name of_node subsystem

图 7.3.2 i2c-1 总线下挂载的器件

其中 1-0050 中的 1 代表 i2c-1, 50 代表器件地址 50, 也就是 eeprom 的器件地址。

进入到 1-0050/目录下,可以看到该目录有一个 eeprom 文件,如下图所示:

root@ALIENTEK-ZYNQ:/sys/class/i2c-adapter/i2c-1# cd 1-0050/ root@ALIENTEK-ZYNQ:/sys/class/i2c-adapter/i2c-1/1-0050# ls 1-00500 driver eeprom modalias name of_node power

图 7.3.3 eeprom 文件

可以通过 eeprom 文件对 EEPROM 设备进行读写操作,譬如向 eeprom 设备中写入 "hello world", 然后读取出来, 输入命令如下:

echo "hello world" > eeprom head -1 eeprom 执行结果如下图所示: #向 eeprom 设备中写入 "hello world" #读取 eeprom

root@ALIENTEK-ZYNQ:/sys/class/i2c-adapter/i2c-1/1-0050# echo "hello world" > eeprom root@ALIENTEK-ZYNQ:/sys/class/i2c-adapter/i2c-1/1-0050# head -1 eeprom hello world

root@ALIENTEK-ZYNQ:/sys/class/i2c-adapter/i2c-1/1-0050#

图 7.3.4 读写 eeprom

至此我们 eeprom 设备的读写就完成了。

7.4 RTC 的使用

在上一小节中,/sys/class/i2c-adapter/i2c-1/目录下除了 1-0050 文件夹,还有一个 1-0051 文 件夹,对应 RTC 设备。进入 1-0051/目录,查看其目录内容,如下图所示:

root@ALIE	ENTEK-ZYNQ:/ ENTEK-ZYNQ:/	sys/class sys/class	/i2c-adapter/i /i2c-adapter/i	i2c-1# c i2c-1/1-	d 1-0051 0051# ls			
driver	modalias	name	of_node	power	rtc	subsystem	uevent	
			图 7.4	.1 RTC	文件			

可以看到有一个 rtc 文件夹,那么操作 rtc 是不是像 eeprom 那样呢?

当然不是,对于 rtc, linux 有一个专用的命令: hwclock。

在 Linux 中有硬件时钟与系统时钟两种时钟。硬件时钟是指电路板上的时钟设备,也就 是通常可在电脑 BIOS 画面设定的时钟。系统时钟则是指 kernel 中的时钟。当 Linux 启动时, 系统时钟会去读取硬件时钟的设定,之后系统时钟独立运作。所有 Linux 相关指令与函数都 是读取系统时钟的设定。

使用 date 和 hwclock 命令可分别查看和设定系统时钟和硬件时钟。

在串口终端中输入下面的指令查看系统时间:

date

串口终端如下图所示:



原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php

正点原子

root@ALIENTEK-ZYNQ:~# date
Mon Mar 13 09:07:13 UTC 2023

图 7.4.2 date 查看系统时钟

上图中箭头所指示的位置就是当前的系统时间。

在串口终端中输入下面的指令查看硬件(RTC)时钟:

hwclock

串口终端如下图所示:

root@ALIENTEK-ZYNQ:~# hwclock Mon Mar 13 09:07:55 2023 0.000000 seconds

图 7.4.3 查看 RTC 时钟

上图中显示的便是当前 RTC 时钟芯片中的时间。

在串口终端中输入下面的指令将系统时间设置为当前日期和时间(2023/3/01,00:00:00),然后使用 date 指令查看所设置的系统时间:

date -s "2023-03-01 00:00:00"

date

执行结果如下图所示:



图 7.4.4 系统时间设置

在串口终端中输入下面的命令将系统时间写入 RTC 时钟芯片中,然后使用 hwclock 命令 查看硬件时钟。

hwclock -w# 将系统时钟同步至硬件时钟hwclock# 查看硬件时钟

执行结果如下图所示:

root@ALIENTEK-ZYNQ:~# hwclock -w 🔶
root@ALIENTEK-ZYNQ:~# hwclock 🔶
Wed Mar 1 00:00:21 2023 0.000000 seconds
root@ALIENTEK-ZYNQ:~#

图 7.4.5 写入 RTC 时钟

7.5 QSPI 的使用

首先执行"cat /proc/mtd"命令查看 qspi flash 分区信息,如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

root@	ALIENTEK-Z	ZYNQ:~# ca	at /proc/mtd	
dev:	size	erasesize	e name	
mtd0:	00100000	00001000	"boot"	
mtd1:	00020000	00001000	"bootenv"	
mtd2:	00400000	00001000	"bitstream"	
mtd3:	00020000	00001000	"device-tree"	
mtd4:	00500000	00001000	"kernel"	
mtd5:	015c0000	00001000	"space"	
root@	ALIENTEK-Z	ZYNQ:~#		

图 7.5.1 查看 qspi flash 分区信息

linux 系统将 qspi flash 分成了 6 个分区: mtd0、mtd1、mtd2、mtd3、mtd4 以及 mtd5 分区, 分区 名分别是: boot、bootenv、bitstream、device-tree、kernel 以及 space, 不同分区用于存放不同镜像文件。本小节使用第 6 个, 即 space 分区。

7.5.1 读写测试

执行下面这条命令读取 flash 的第六个分区,由于该分区比较大,为了节省时间,我们只 读取 64 个字节:

hexdump	-C -	n 64	l /de	v/m	tdbl	ock	5	#i	卖 fla	sh 豸	新六	个分	NX ·	长度	专为	64 /	〉字节
root@ALIEN 00000000	NTEH 00	<-Z` 00	YNQ : 00	:~# 00	he> 00	cdun 00	р 00	-C - 00	n 64 00	4 /0 00	dev, 00	/mto 00	dblo 00	ock5 00	5 00	00	
* 00000040 root@ALIEM	f ITE	lash <-Z`	ynd :	: :~#			1		司数排								↑ 对应数据的ASCII字符

图 7.5.2 读取 flash

左边红箭头处显示的是十六进制表示的相对地址,一行地址共有 16 字节数据;中间红框 中显示的是十六进制表示的数据,而右边箭头处则是对应的 ASCII 字符,由于有些数据没有 对应的 ASCII 字符,所以用"....."来表示。

执行下面这条命令将字符串写入 QSPI 的第六个分区中:

echo "ALIENTEK ZYNQ 7020" > /dev/mtdblock5 #将字符串写入 flash

hexdump -C -n 64 /dev/mtdblock5 #读取 flash 中内容



图 7.5.3 读写 flash

双引号("")中间的字符串就是我们要写入的,从上面可以知道,当写入之后再次读取 flash,发现写入与读取的数据是相同的,所以 qspi flash 读写测试成功!

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

正点原子

7.5.2 复制文件

首先在用户"root"根目录下创建"test.txt"文件,并向文件中写入"flash test"内容, 命令如下所示:

touch test.txt #创建 test.txt 文本

echo "flash test" > test.txt #向文本中写入内容

结果如下图所示:

<pre>root@ALIENTEK-ZYNQ:~#</pre>	touch test.txt
<pre>root@ALIENTEK-ZYNQ:~#</pre>	<pre>echo "flash test" > test.txt</pre>
<pre>root@ALIENTEK-ZYNQ:~#</pre>	cat test.txt_
flash test	
<pre>root@ALIENTEK-ZYNQ:~#</pre>	查看是否写入到文本中

图 7.5.4 创建文本文件

使用"flashcp"命令复制文本文件到 flash 第六个分区,命令如下所示:

flashcp test.txt /dev/mtd5 #复制文本文件到 flash 中

hexdump -C -n 64 /dev/mtdblock5 #查看分区中的内容是否和文本中一致

执行结果如下图所示:

root@ALIEN	ITEk	(-Z)	(NQ :	~#	fla	ishc	p t	test.	.txt	: /0	dev/	'mtc	15 🔺						
root@ALIEN	ITEK	(-Z)	(NQ:	~#	he>	dun	np -	-C -r	n 64	/ /	dev/	'mtc	blo	ock5	5 🔶				
00000000	66	6c	61	73	68	20	74	65	73	74	0a	ff	ff	ff	ff	ff	flash	test.	
00000010	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff			
*																			
00000040																			
root@ALIEN	ITEK	(-Z)	(NQ :	~#															

图 7.5.5 复制文本文件到 flash 分区

从上图中可以看到, 文本中的"flash test"内容已经成功复制到 flash 分区中了。

7.6 USB 的使用

领航者底板上有四个 USB HOST 接口和一个 USB SLAVE 接口,本小节我们通过访问 USB 设备——U 盘,来学习 USB HOST 接口的使用。

开发板上 USB HOST 和 SLAVE 接口共用相同 ZYNQ 引脚,使用前要通过跳线帽手动选 择。使用跳线帽将 P4 端子的 DN 与 HN 相连、DP 与 HP 相连, 注意先连接 DN 与 HN,此时 USB HOST 接口被激活。

将 fat32 格式的 U 盘插入底板上 4 个 USB HOST 接口中的一个。注意, U 盘文件系统格式 不支持 ntfs, ntfs 格式的用户请将 U 盘重新格式化成 fat32, 格式化之前请备份 U 盘中的数据。

此时串口终端会有打印信息,如下所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

root@ALIENTEK-ZYNQ:~# usb 1-1.1: new high-speed USB device number 4 using ci_hdrc usb-storage 1-1.1:1.0: USB Mass Storage device detected scsi host0: usb-storage 1-1.1:1.0 scsi 0:0:0:0: Direct-Access Generic STORAGE DEVICE 1532 PQ: 0 ANSI: 6 sd 0:0:0:0: Attached scsi generic sg0 type 0 sd 0:0:0:0: [sda] 31116288 512-byte logical blocks: (15.9 GB/14.8 GiB) sd 0:0:0:0: [sda] Write Protect is off sd 0:0:0:0: [sda] Write cache: disabled, read cache: enabled, doesn't support DP0 or FUA sda: sda1 sd 0:0:0:0: [sda] Attached SCSI removable disk

图 7.6.1 插入 U 盘后串口打印信息

信息打印出来之后,插入的 U 盘就已经被挂载到根文件系统中了,此时可通过 df 命令查 看挂载点,如下所示:

root@ALIENTEK-ZYN0):~# df -Th 👞				
Filesystem	Туре	Size	Used /	Available	Use% Mounted on
devtmpfs	devtmpfs	490.0M	4.0K	490.0M	0% /dev
tmpfs	tmpfs	502.1M	140.0K	501.9M	0% /run
tmpfs	tmpfs	502.1M	56.0K	502.0M	0% /var/volatile
/dev/mmcblk0p1	vfat	499.0M	71.7M	427.3M	14% /media/sd-mmcblk0p1
/dev/mmcblk0p2	ext4	14.1G	55.3M	13.3G	0% /media/sd-mmcblk0p2
/dev/mmcblk1p1	ext4	7.1G	32.8M	6.7G	0% /media/sd-mmcblk1p1
/dev/sda1	vfat	14.8G	8.0K	14.8G	0% /run/media/sda1
root@ALTENTEK-7YN():~#				

图 7.6.2 查看 U 盘挂载点

红色框标识出来的就是笔者插入的 USB 设备——U 盘,显示 U 盘文件系统格式是 vfat, 也就 fat32,实际容量为 14.8G,挂载点是/run/media/sda1,进入到该目录可以看到 U 盘中存放 的文件以及文件夹,如下所示:

> root@ALIENTEK-ZYNQ:~# cd /run/media/sda1/ root@ALIENTEK-ZYNQ:/run/media/sda1# ls -l total 0 4—— 空U盘,没有文件

> > 图 7.6.3 查看 U 盘中的文件

通过下面这条命令创建一个文件 hello.txt,内容为"Hello World www.openedv.com",如下所示:

echo "Hello World www.openedv.com" > hello.txt

root@ALIENTEK-	ZYNQ:/run/m	nedia/sda1#	echo "Hel	llo Wor	ld www	v.openedv.com"	<pre>> hello.txt</pre>
root@ALIENTEK-	ZYNQ:/run/m	nedia/sda1#	ls -l				
total 8							
-rwxrwx	1 root	disk	28 1	lar 27	10:57	hello.txt	
root@ALIENTEK-	ZYNQ:/run/m	nedia/sda1#					

图 7.6.4 在 U 盘中创建新文件

文件创建之后,使用 cat 命令读取文件的内容,如下:

cat hello.txt

root@ALIENTEK-ZYNQ:/run/media/sda1# cat hello.txt Hello World www.openedv.com root@ALIENTEK-ZYNQ:/run/media/sda1#

图 7.6.5 读取 hello.txt 内容

领	航者 ZYNQ 之	嵌入式 Linu	x 开发指南		止京原子	
原于	子哥在线教学:ww	ww.yuanzige.com	论坛:www.op		n.php	F -t-
接打	与人的内谷与读 巴U盘拔了,需要	取出米的内谷一; 「先把 U 盘卸载,	政,说明 U 盘读与 在串口终端中输入切	测试没有问题。 如下命令:	测试元成乙后个岁	芝且
	cd	#退出U盘所在	目录			
	umount /run/media/s	/sda1/ #卸载U盘	ī		_	
	ro ro	ot@ALIENTEK-ZY ot@ALIENTEK-ZY	NQ:/run/media/sda NQ:/run/media/sda NO:.#umount/rum	a1# a1# cd a/madia/ada1/		
	ro ro ro	oot@ALIENTEK-ZY oot@ALIENTEK-ZY oot@ALIENTEK-ZY	NQ:/run/media/sda NQ:/run/media/sda NQ:~# umount /rur	a1# a1# cd h/media/sda1/		

root@ALIENTEK-ZYNQ:~#

图 7.6.6 卸载 U 盘

卸载成功之后就可以把U盘拔掉了。

7.7 以太网的使用

领航者开发板有两路千兆以太网接口,GE_PS和GE_PL;由于GE_PS网口使用了PS端IO资源,而GE_PL网口使用了PL端IO资源,所以这里也把GE_PS网口称为PS网口、而把GE_PL网口称为PL网口,如下所示:



图 7.7.1 开发板上的网口示例图

注: 连接网口的网线要使用千兆网线,譬如 CAT-5E 类网线或 CAT-6 类网线,笔者在实际测试当中,发现 CAT-5E 类网线并不稳定,所以这里推荐使用 CAT-6 类网线进行测试。

7.7.1 查看网络设备

在串口终端执行下面这条命令可以查看系统中的所有网络设备,如下所示: ifconfig-a



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php root@ALIENTEK-ZYNQ:~# ifconfig -a can0 NOARP MTU:16 Metric:1 RX packets:0 errors:0 dropped:0 overruns:0 frame:0 TX packets:0 errors:0 dropped:0 overruns:0 carrier:0 collisions:0 txqueuelen:10 RX bytes:0 (0.0 B) TX bytes:0 (0.0 B) Interrupt:21 Link encap:Ethernet HWaddr 00:0A:35:00:8B:87 UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1 eth0 RX packets:1702 errors:0 dropped:0 overruns:0 frame:0 TX packets:468 errors:0 dropped:0 overruns:0 carrier:0 collisions:0 txqueuelen:1000 RX bytes:83962 (81.9 KiB) TX bytes:161172 (157.3 KiB) Interrupt:30 Base address:0xb000 Link encap:Ethernet HWaddr 00:0A:35:00:11:55 BROADCAST MULTICAST MTU:1500 Metric:1 RX packets:0 errors:0 dropped:0 overruns:0 frame:0 eth1 TX packets:0 errors:0 dropped:0 overruns:0 carrier:0 collisions:0 txqueuelen:1000 RX bytes:0 (0.0 B) TX bytes:0 (0.0 B) Interrupt:31 Base address:0xc000 Link encap:Local Loopback inet addr:127.0.0.1 Mask:255.0.0.0 inet6 addr: ::1/128 Scope:Host UP LOOPBACK RUNNING MTU:65536 Metric:1 RX packets:0 errors:0 dropped:0 overruns:0 frame:0 lo TX packets:0 errors:0 dropped:0 overruns:0 carrier:0 collisions:0 txqueuelen:1000 RX bytes:0 (0.0 B) TX bytes:0 (0.0 B) sit0 Link encap: IPv6-in-IPv4 NOARP MTU:1480 Metric:1 RX packets:0 errors:0 dropped:0 overruns:0 frame:0 TX packets:0 errors:0 dropped:0 overruns:0 carrier:0 collisions:0 txqueuelen:1000 RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

图 7.7.2 查看所有网络设备

图 7.7.2 中显示了当前系统中所有的网络设备,其中 eth0表示开发板上的 PS 网口、而 eth1 则表示开发板上的 PL 网口。还可以直接使用 ifconfig 命令不加任何选项查看当前系统已经激活 (打开)的网络设备,如下图所示:

root@ALI eth0	ENTEK-ZYNQ:~# ifconfig Link encap:Ethernet HWaddr 00:0A:35:00:8B:87 UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1 RX packets:1908 errors:0 dropped:0 overruns:0 frame:0 TX packets:552 errors:0 dropped:0 overruns:0 carrier:0 collisions:0 txqueuelen:1000
	RX bytes:93819 (91.6 KiB) IX bytes:190236 (185.7 KiB) Interrupt:30 Base address:0xb000
lo	Link encap:Local Loopback inet addr:127.0.0.1 Mask:255.0.0.0 inet6 addr: ::1/128 Scope:Host UP LOOPBACK RUNNING MTU:65536 Metric:1 RX packets:0 errors:0 dropped:0 overruns:0 frame:0 TX packets:0 errors:0 dropped:0 overruns:0 carrier:0 collisions:0 txqueuelen:1000 RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

图 7.7.3 查看当前打开的网络设备

上图中的 eth0 就是开发板上的 PS 网口,可以通过 ifconfig 命令来关闭或激活对应的网口。

原子	·哥在线教学:www.yua	nzige.com	论坛	:www.openedv.com/f	orum.php	
	ifconfig 命令打开或关闭	引 PS 网口的命令	∲如下,	所示:		
	ifconfig eth0 down	#关闭 eth0(PS 🕅	网口)			
	ifconfig eth0 up #打	开 eth0(PS 网口))			
	ifconfig 命令打开或关闭	引 PL 网口的命令	∲如下	所示:		
	ifconfig eth1 down	#关闭 eth1 (PL)	网口)			
	ifconfig eth1 up #打	开 eth1(PL 网口))			
	如果是用 ip 命令,则对	讨应命令如下:				
	ip link set eth0 down	#关闭 eth0(PS 图	网口)			
	ip link set eth0 up	#打开 eth0(PS 🕅	网口)			

正点原子

下面我们在使用 PS 网口的时候,需要先把 PL 网口给关闭,只打开 PS 网口;同理使用 PL 网口的时候,需要把 PS 网口给关闭,只打开 PL 网口;在后面的使用当中,笔者以 PS 网口为例进行介绍,PL 网口使用方式相同。

7.7.2 外网连接测试(有路由器)

在测试之前,我们先使用网线将开发板 PS 网口连接到能够上网的路由器设备上,如果没 有能够上网的路由器设备的请跳过该小节,进入下一小节,因为访问外网需要联网的路由器。 执行下面的命令打开开发板的 PS 网口,并且关闭 PL 网口:

ip link set eth1 down#关闭 PL 网口ip link set eth0 up#打开 PS 网口ip link show up结果如下图所示:

root@ALIENTEK-ZYNQ:~# ip link set eth1 down root@ALIENTEK-ZYNQ:~# ip link set eth0 up [10501.162415] pps pps0: new PPS source ptp0 [10501.166478] macb ff0b0000.ethernet: gem-ptp-timer ptp clock registered. [10501.173370] IPv6: ADDRCONF(NETDEV_UP): eth0: link is not ready root@ALIENTEK-ZYNQ:~# [10504.241614] macb ff0b0000.ethernet eth0: unable to generate target frequenc y: 125000000 Hz [10504.249898] macb ff0b0000.ethernet eth0: link up (1000/Full) [10504.255584] IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready root@ALIENTEK-ZYNQ:~# ip link show up 1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000 link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00 3: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode DEFAULT group default qle n 1000 link/ether de:7a:01:81:ba:0e brd ff:ff:ff:ff:ff:ff:ff:ff: root@ALIENTEK-ZYNQ:~# |

图 7.7.4 打开 PL 网口关闭 PS 网口

接下来我们需要给 PS 网口分配一个 IP 地址,使用 udhcpc 命令从 DHCP 服务器中动态获取一个 IP 地址,如下所示:

udhcpc -i eth0 #eth0 动态获取 IP 地址



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

root@ALIENTEK-ZYNQ:~#
root@ALIENTEK-ZYNQ:~# udhcpc -i eth0
udhcpc: started, v1.29.2
udhcpc: sending discover
udhcpc: sending select for 192.168.2.219
udhcpc: lease of 192.168.2.219 obtained, lease time 86400
/etc/udhcpc.d/50default: Adding DNS 192.168.2.1
root@ALIENTEK-ZYNQ:~#

图 7.7.5 动态获取 IP 地址

从图 7.7.5 中可以看到,笔者这里 PS 网口动态获取得到的 IP 地址为 192.168.2.219。获取 到 IP 地址之后,接下来我们需要测试下开发板是不是能够上网,也就是测试开发板 PS 网口 是否工作正常、是否能够连接外网。当然首先确定开发板 PS 网口连接到的路由器是能够连接 外网的,我们可以使用 ping 命令来测试开发板与另一台主机的网络连接是否通畅。

ping 命令是基于 ICMP(Internet Control Message Protocol)协议来工作的,执行 ping 命令本地主机会向目标主机发送一份 ICMP 回显请求报文,并等待目标主机返回 ICMP 应答;因为 ICMP 协议会要求目标主机收到消息之后,必须返回 ICMP 应答消息给本地主机,如果本地主机收到了目标主机的应答,则表示两台主机之间的网络运行、网络连接是正常的。

例如在串口终端执行 ping 命令来测试开发板(本地主机)与百度 <u>www.baidu.com</u>服务器(目标主机)之间的网络连接是否通畅,如下所示:

	ping -c 10 www.baidu.com	#测试开发板与百度服务器之间网络连接情况
×	root@ATK_ZYNQ:~# root@ATK_ZYNQ:~# ping -c 10 w PING www.baidu.com (14.215.177.39: s 64 bytes from 14.215.177.39: s	ww.baidu.com 7.39): 56 data bytes seq=0 ttl=55 time=5.955 ms seq=1 ttl=55 time=6.140 ms seq=2 ttl=55 time=6.113 ms seq=4 ttl=55 time=5.686 ms seq=5 ttl=55 time=5.686 ms seq=6 ttl=55 time=5.552 ms seq=6 ttl=55 time=5.752 ms seq=8 ttl=55 time=5.772 ms seq=9 ttl=55 time=5.772 ms
	www.baidu.com ping statist 10 packets transmitted, 10 pac round-trip min/avg/max = 5.55 root@AIK_ZYNQ:~# root@ATK_ZYNQ:~#	tics ckets received, 0% packet loss 2/5.900/6.180 ms 求。接收到10次应答

图 7.7.6 执行 ping 命令

从上图可以知道,我们通过开发板对百度服务器主机(IP 地址: 14.215.177.39)发送了 10 次应答请求,并且每次都收到了它的应答消息(64 字节数据),并且没有数据丢失,说明 开发板与百度服务器主机之间的网络运行、网络连接是 OK 的,也就意味着我们的开发板与 外网是连通的。

PL 网口的外网连接测试同理。

7.7.3 电脑直连测试 (无路由器)

在开发过程中,电脑和开发板互相访问是经常需要的,这可以通过路由器来实现,连接 到同一路由器的设备是可以互相访问的,如果没有路由器,也可以使用网线将开发板 PS 网口



原子哥在线教学:www.yuanzige.com 论坛:www.openedv.com/forum.php 或者 PL 网口直接连接到电脑的以太网接口上,也就是电脑直连,不过这种方式不能访问外网。 笔者以 PS 网口为例,用网线将开发板的 PS 网口和电脑的以太网接口相连接。 连接好网线之后,需要设置电脑以太网的 IP 地址(设置方法可以参考该视频链接:Linux

<u>开发板网络直连电脑的设置方法</u>)。笔者设置电脑的 IP 地址如下图所示:

Internet 协议版本 4 (TCP/IPv4) 属性		×
常规		
如果网络支持此功能,则可以获取自动指 络系统管理员处获得适当的 IP 设置。	派的 IP 设置。否则,你需要从网	
○ 自动获得 IP 地址(O)		
-● 使用下面的 IP 地址(S):		
IP 地址(I):	192.168.1.89	
子网掩码(U):	255.255.255.0	
默认网关(D):	192.168.1.1	
○ 自动获得 DNS 服务器地址(B)		
● 使用下面的 DNS 服务器地址(E):		
首选 DNS 服务器(P):	1.1.1.1	
备用 DNS 服务器(A):	· · ·	
□退出时验证设置(L)	高级(V)	
	确定取消	í

图 7.7.7 设置 IPv4 地址

配置完成后,在串口终端中执行下面这些命令打开开发板的 PS 网口,并且关闭 PL 网口: ifconfig eth1 down #关闭 PL 网口

ifconfig eth0 up #打开 PS 网口

执行下面的命令设置开发板 eth0 网口的静态 IP 地址为 192.168.1.10:

ifconfig eth0 192.168.1.10 netmask 255.255.255.0

设置完成后,开发板的 IP 地址和电脑的 IP 就在同一网段。接下来进行 Ping 测试,看开发板和电脑能不能相互 Ping 通。

首先开发板 ping 电脑,命令如下:

ping -c4 192.168.1.89

结果如下图所示:

```
root@ALIENTEK-ZYNQ:~# ifconfig eth0 192.168.1.10 netmask 255.255.255.0
root@ALIENTEK-ZYNQ:~# ping -c4 192.168.1.89
PING 192.168.1.89 (192.168.1.89): 56 data bytes
64 bytes from 192.168.1.89: seq=0 ttl=128 time=0.401 ms
64 bytes from 192.168.1.89: seq=1 ttl=128 time=0.239 ms
64 bytes from 192.168.1.89: seq=2 ttl=128 time=0.226 ms
64 bytes from 192.168.1.89: seq=3 ttl=128 time=0.233 ms
--- 192.168.1.89 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 0.226/0.274/0.401 ms
root@ALIENTEK-ZYNQ:~#
```

图 7.7.8 开发板 ping 电脑结果

原子哥在线教学:www.yuanzige.com 如果 Ping 不通电脑,请关闭电脑的防火	论坛:www.openedv.com/forum.php (墙后再尝试。
电脑 ping 开友侬。自无打开电脑的 cmc	1.叩受旋小何, 26. 加利人如下叩受:
ping 192.168.1.10	
结果如下:	
 C:\Users\Administrator>ping 正在 Ping 192.168.1.10 具有 来自 192.168.1.10 的回复: 年 来自 192.168.1.10 的回复: 年 来自 192.168.1.10 的回复: 年 来自 192.168.1.10 的回复: 年 书自 192.168.1.10 的回复: 年 192.168.1.10 的回复: 4 大百百万万百万万万万万万万万万万万万万万万万万万万万万万万万万万万万万万万万	192.168.1.10 32 字节的数据: 字节=32 时间<1ms TTL=64 字节=32 时间<1ms TTL=64 字节=32 时间<1ms TTL=64 字节=32 时间<1ms TTL=64 字节=32 时间<1ms TTL=64 息: 度收 = 4, 丢失 = 0 (0% 丢失), 可单位): 恶丧地 = 0ms

🔁 正点原子

图 7.7.9 电脑 ping 开发板结果

开发板和电脑相互 Ping 通,说明双方的通信基本上没啥问题了。

C:\Users\Administrator>_

如果想测试 Ubuntu 虚拟机与开发板的网络连接,可以设置 Ubuntu 虚拟机的网络,设置 方法可以继续参考该视频链接: <u>Linux 开发板网络直连电脑的设置方法</u>。笔者设置 Ubuntu 虚 拟机的 IP 地址如下图所示:

详细信息 身份 IPv4 I	Pv6 安全		
I IPv4 方式 2	○ 自动 (DHCP) ● 手动	○ 仅本地链路 ○ 禁用	
地址 地址	3 子网掩码	网关	
192.168.1.20	255.255.255.0	192.168.1.1	3
			3
DNS		自动打开	
使用逗号分隔 IP 地址			
路由		自动打开	

图 7.7.10Ubuntu 虚拟机静态 IP 地址

首先开发板 ping Ubuntu 虚拟机,命令如下: ping -c4 192.168.1.20 结果如下图所示:





原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php root@ALIENTEK-ZYNQ:~# ping -c4 192.168.1.20 PING 192.168.1.20 (192.168.1.20): 56 data bytes 64 bytes from 192.168.1.20: seq=0 ttl=64 time=0.522 ms 64 bytes from 192.168.1.20: seq=1 ttl=64 time=0.431 ms 64 bytes from 192.168.1.20: seq=2 ttl=64 time=0.306 ms 64 bytes from 192.168.1.20: seq=3 ttl=64 time=0.310 ms --- 192.168.1.20 ping statistics ---4 packets transmitted, 4 packets received, 0% packet loss round-trip min/avg/max = 0.306/0.392/0.522 ms root@ALIENTEK-ZYNQ:~#

图 7.7.11 开发板 ping Ubuntu 虚拟机结果

Ubuntu 虚拟机 ping 开发板。首先打开 Ubuntu 虚拟机的终端, 然后输入如下命令: ping -c4 192.168.1.10

结果如下:

sqd@sqd-virtual-machine:~\$ ping -c4 192.168.1.10 PING 192.168.1.10 (192.168.1.10) 56(84) bytes of data. 64 bytes from 192.168.1.10: icmp_seq=1 ttl=64 time=0.332 ms 64 bytes from 192.168.1.10: icmp_seq=2 ttl=64 time=0.301 ms 64 bytes from 192.168.1.10: icmp_seq=3 ttl=64 time=0.263 ms 64 bytes from 192.168.1.10: icmp_seq=4 ttl=64 time=0.589 ms --- 192.168.1.10 ping statistics ---4 packets transmitted, 4 received, 0% packet loss, time 3051ms rtt min/avg/max/mdev = 0.263/0.371/0.589/0.128 ms sqd@sqd-virtual-machine:~\$

图 7.7.12 Ubuntu 虚拟机 ping 开发板结果

开发板和 Ubuntu 虚拟机相互 Ping 通,说明双方的通信基本上没啥问题了。

如果不能 ping 通,点击虚拟机菜单栏"编辑→虚拟网络编辑器",检查 Vmnet0 是否桥接至以太网网卡,如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

🕀 虚拟网络编辑器

🕀 虚拟网络	各编辑器					×
名称 VMnet0 VMnet1 VMnet8	类型 桥接模式 自定义 NAT 模式	外部连接 Realtek PCIe GbE Family Co - NAT 模式	主机连接 - - 已连接	DHCP - - 已启用	子网地址 - 192.168.72 192.168.14	.0 7.0
VMnet 信息 ● 桥接樹 已桥掛	1 (式(将虚拟 (接至(G): Rea (与虚拟	<u>れ直接连接到外部网络)(B)</u> altek PCIe GbE Family Controller 11共享主机的 IP 地址)(N)	添加网络(E)	移除网络	\$(O) 重 ~ 自 N/	命名网络(W) 动设置(U) IT 设置(S)
 ○ 仅主机 ○ 将主机 主机 ○ 使用本 → 子网 IP (I 	 1.模式(在专月 1.虚拟适配器 塩拟适配器 塩 DHCP 服); (二); (1); (1);<	用网络内连接虚拟机()(H) 法 接到此网络(V) 高称: VMware 网络适配器 VMnet 务将 IP 地址分配给虚拟机(D) 子网箍码(M) 局入(T)	· ; 确定	取消	DH 应用(A)	CP 设置(P) 帮助

图 7.7.13 Vmnet0 桥接至以太网网卡

PL 网口的电脑直连测试同理,此处就不赘述了。

7.8 eMMC 的使用

开发板板载 8GB eMMC,可以把 eMMC 当作固定在板子上的 SD 卡,其使用方式跟 SD 卡一样。接下来通过简单的读写来学习 eMMC 的使用。

由于本小节需要对 eMMC 进行格式化操作, 而之前 6.3.8 小节生成的根文件系统不支持格 式化命令,需要重新配置。运行"petalinux-config -c rootfs"命令打开根文件系统配置界面,进 入 "Filesystem Packages → base → e2fsprogs" 选项下, 勾选 "e2fsprogs-mke2fs" 选项, 如下 图所示:



图 7.8.1 重新配置根文件系统

配置完成后保存并退出,重新编译工程,将编译后生成的 image.ub 复制到 sd 卡替换原来的 image.ub 文件,并重新启动系统。

系统重新启动后,进入系统,输入"df-Th"命令,查看挂载的文件系统,如下图所示:

root@ALIENTEK-ZYNQ:~# df -Th					
Filesystem	Туре	Size	Used	Available	Use% Mounted on
devtmpfs	devtmpfs	490.1M	4.0K	490.1M	0% /dev
tmpfs	tmpfs	502.1M	120.0K	502.0M	0% /run
tmpfs	tmpfs	502.1M	52.0K	502.0M	0% /var/volatile
/dev/mmcblk0p1	vfat	499.0M	49.1M	449.9M	10% /media/sd-mmcblk0p1
/dev/mmcblk1p1	ext4	7.1G	332.1M	6.4G	5% /media/sd-mmcblk1p1
/dev/mmcblk0p2	e <u>x</u> t4	14.1G	55.3M	13.3G	0% /media/sd-mmcblk0p2

图 7.8.2 查看文件系统

其中"/dev/mmcblk1p1"就是 eMMC 设备,代表 eMMC 的第一个分区,挂载在 "/media/sd-mmcblk1p1"挂载点。接下来参考 6.3.10 章节制作 SD 启动卡的步骤,重新将 eMMC 格式化成一个分区。

分别执行下面两个命令,先从挂载点卸载 eMMC,再对 eMMC分区,结果如下图所示: umount /media/sd-mmcblk1p1

fdisk /dev/mmcblk1



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php root@ALIENTEK-ZYNQ:~# umount /media/sd-mmcblk1p1/ root@ALIENTEK-ZYNQ:~# df Filesystem 1K-blocks Used Available Use% Mounted on devtmpfs 501908 - 4 501904 0% /dev tmpfs 514132 120 514012 0% /run 52 tmpfs 514132 514080 0% /var/volatile /dev/mmcblk0p1 510984 50268 460716 10% /media/sd-mmcblk0p1 /dev/mmcblk0p2 14743024 56660 13917724 0% /media/sd-mmcblk0p2 root@ALIENTEK-ZYNQ:~# fdisk /dev/mmcblk1 The number of cylinders for this disk is set to 238592. There is nothing wrong with that, but this is larger than 1024, and could in certain setups cause problems with: 1) software that runs at boot time (e.g., old versions of LILO) 2) booting and partitioning software from other OSs (e.g., DOS FDISK, OS/2 FDISK) Command (m for help):

图 7.8.3 卸载 eMMC 并对 eMMC 分区

执行后输入"p",结果如下图所示:

Command (m for he Disk /dev/mmcblk1 238592 cylinders, Units: sectors of	lp): p : 7456 MB, 78 4 heads, 16 1 * 512 = 52	318182656 byte: sectors/track 12 bytes	s, 15269888	sectors			
Device Boot	StartCHS	EndCHS	StartLBA	EndLBA	Sectors	Size	Id Type
/dev/mmcblk1p1	32,0,1	1023,3,16	2048	15269887	15267840	7455M	83 Linux

图 7.8.4 查看分区信息

可以看到 eMMC 已经有一个 p1 (mmcblk1p1) 分区,为了演示如何新建分区,这里我们 先删除该分区。输入 "d" 删除第一个分区,再次输入 "d" 提示没有分区可以删除,如下图 所示:



图 7.8.5 删除原有分区

接下来输入"n"创建新分区,输入"p"选择创建主分区,按回车键,然后输入"1"设置分区号,接下来设置分区起始扇区,都按回车键,选择默认即可,如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

图 7.8.6 新建分区

输入 "p" 检查分区表,可以看到新建的分区,如果没有问题,输入 "w" 保存并退出,如下图所示:



图 7.8.7 查看并保存分区信息

分区创建完成后,就可以格式化分区了。在终端输入如下命令:

mkfs.ext4 -L rootfs /dev/mmcblk1p1

该命令将分区格式化成 ext4 分区并命名为 "rootfs",如果提示该分区已经存在一个 ext4 文件系统,询问是否继续,直接输入"y"并回车,如下图所示:



501780 4 0% /dev devtmpfs 501776 tmpfs 514132 128 514004 0% /run 52 tmpfs 514132 514080 0% /var/volatile /dev/mmcblk0p1 510984 73436 437548 14% /media/sd-mmcblk0p1 /dev/mmcblk0p2 0% /media/sd-mmcblk0p2 14743024 56660 13917724 7449440 33540 7017772 /dev/mmcblk1p1 0% /media/sd-mmcblk1p1 root@ALIENTEK-ZYNQ:~#

图 7.8.8 格式化分区

格式化完成后还需要重新挂载 eMMC 到系统,如上图所示。此时就可以对 eMMC 进行读 写操作了。

输入 echo "www.openedv.com" > /media/sd-mmcblk1p1/test.txt 命令在 eMMC 中创建名为 test.txt 的 文本文件,并向文件中写入内容为"www.openedv.com"的文本内容,然后使用 cat /media/sdmmcblk1p1/test.txt 命令,将文件中的内容打印出来,可以看到写入和读出的内容相同,如下图 所示:

root@ALIENTEK-ZYNQ:~# echo "www.openedv.com" > /media/sd-mmcblk1p1/test.txt root@ALIENTEK-ZYNQ:~# cat /media/sd-mmcblk1p1/test.txt www.openedv.com root@ALIENTEK-ZYNQ:~#

图 7.8.9 eMMC 读写操作

可以看到 eMMC 的使用方式跟 SD 卡没啥区别。

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

正点原子

第八章 Linux 显示设备的使用

上一章我们介绍了基础外设的使用,本章学习显示设备的使用。领航者开发板上有两个 显示设备接口,分别为 HDMI 接口和 LCD 接口,这两个接口可以用来接 HDMI 显示器和正点 原子的 LCD 液晶屏。本章我们讲解如何使用 Petalinux 配置 Linux 内核和设备树来驱动 HDMI 显示器和 LCD 液晶屏。



8.1 准备工作

LCD 或者 HDMI 的显示首先需要硬件层面的支持,也就是需要搭建对应的 vivado 工程, 在第六章当中我们统一使用了正点原子为领航者开发板所配置 vivado 工程,同样该工程也是 开发板出厂时烧录的系统中所使用的工程,对于领航者 7020 来说,正点原子提供的 vivado 工 程支持 LCD 和 HDMI,但是对于领航者 7010 来说,其提供的工程只支持 LCD 不支持 HDMI, 原因在于 7010 PL 端逻辑资源较少,为了其它外设能够配置上,故而取消了 HDMI。

正点原子

除了需要该实验对应的 Vivado 工程的硬件平台支持外,还需要相应的 Linux 驱动程序, LCD 和 HDMI 显示设备的驱动程序我们放在了提供的内核源码中。光盘资料 ZYNQ 开发板资料盘(A 盘)\4_SourceCode\3_Embedded_Linux\资源文件\出厂镜像相关目录下,有一个名为 atkzynq-linux-xlnx.tar.gz 的内核源码压缩包。这是正点原子针对领航者开发板进行移植、修改的 Linux 内核源码包,里面添加了适用于领航者开发板的驱动程序。

执行下面的命令,在 workspace 目录下新建 kernel-ch8 文件夹,然后将 atk-zynq-linuxxlnx.tar.gz 拷贝到该文件夹下:

mkdir kernel-ch8

#创建 kernel-ch8 目录

cp /mnt/hgfs/share/kernel/atk-zynq-linux-xlnx.tar.gz kernel-ch8/ #将内核源码压缩包复制到 kernel-ch8 执行结果如下图:

sqd@sqd-virtual-machine:~/workspace/kernel-ch8\$ ls
atk-zynq-linux-xlnx.tar.gz
sqd@sqd-virtual-machine:~/workspace/kernel-ch8\$

图 8.1.1 复制内核源码到 ubuntu 系统中

接下来将其解压,对应的解压目录就是内核源码目录,这个解压目录大家可以自己设置, 笔者这里选择将其解压到当前 kernel-ch8 目录下:

tar -xzf atk-zynq-linux-xlnx.tar.gz #解压内核源码

sync #同步

rm atk-zynq-linux-xlnx.tar.gz #删除压缩包

结果如下图所示:

sqd@sqd-virtual-machine:~/workspace/kernel-ch8\$ tar -xzf atk-zynq-linux-xlnx.tar.gz sqd@sqd-virtual-machine:~/workspace/kernel-ch8\$ sync sqd@sqd-virtual-machine:~/workspace/kernel-ch8\$ ls atk-zynq-linux-xlnx atk-zynq-linux-xlnx.tar.gz sqd@sqd-virtual-machine:~/workspace/kernel-ch8\$ rm atk-zynq-linux-xlnx.tar.gz sqd@sqd-virtual-machine:~/workspace/kernel-ch8\$ ls atk-zynq-linux-xlnx.tar.gz



解压完成后,进入 workspace/kernel-ch8 目录下,可看到 linux 内核源码目录结构,如下图 所示:



图 8.1.3 linux 内核源码目录

至此我们准备工作就已经完成了,接下来需要对 petalinux 工程进行配置。

8.2 创建 Petalinux 工程

参考 6.3.3 小节创建一个名为 "ALIENTEK-ZYNQ-disp-dev" 的工程。

8.3 配置 Petalinux 工程

参考 6.3.4 小节,使用 xsa 文件配置 petalinux 工程,然后按照图 6.3.9 所示配置串口,串口 配置完成后返回主配置界面。

进入到"Linux Components Selection--->linux-kernel(linux-xlnx)"菜单下,配置Linux内核来源。此处选择"ext-local-src",也就是本地存放的Linux内核源码,如下图所示:



图 8.3.1 选择 "ext-local-src"

按键盘上的下方向键移到"ext-local-src",然后按键盘上的"Enter"键确定,返回到上一界面,如下图所示:



图 8.3.2 返回到上一界面

进入"External linux-kernel local source settings"子菜单,如下图所示:



图 8.3.3 "External linux-kernel local source settings"子菜单

按键盘上的"Enter"键配置"External linux-kernel local source path",如下图所示:

Ex Please enter a st field to the butt	t <mark>ernal linux-kern</mark> ring value. Use th ons below it.	el local source he <tab> key to n</tab>	path move from the input
/home/sqd/workspa	ce/kernel-ch8/atk	-zynq-linux-xlnx	
L			
	< 0k >	< Help >	

图 8.3.4 填写 Linux 内核源码的本地路径



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

也就是填写 Linux 内核源码的本地路径,上一节我们将 Linux 内核源码解压到了 Ubuntu 系统/home/sqd/workspace/kernel-ch8/atk-zynq-linux-xlnx 目录,所以此处填写该目录,大家根据自己的路径填写即可,填写完成后,按键盘上的"Enter"键完成配置,连续按四次"ESC"键返回到主界面,如下图所示:

<pre>misc/config System Configuration Arrow keys navigate the menu. <enter> selects submenus> (or empty submenus). Highlighted letters are hotkeys. Pressing <y> includes, <n> excludes, <m> modularizes features. Press <esc><esc> to exit, <?> for Help, for Search. Legend: [*] built-in []</esc></esc></m></n></y></enter></pre>
<pre>-*- ZYNQ Configuration Linux Components Selection> Auto Config Settings> -*- Subsystem AUTO Hardware Settings> DTG Settings> FSBL Configuration> FPGA Manager> u-boot Configuration> Linux Configuration> Image Packaging Configuration> t(+)</pre>
<pre><select> < Exit > < Help > < Save > < Load ></select></pre>

图 8.3.5 返回主界面

需要配置的选项已经修改完成了,选择 Save 保存,然后按两次"ESC"退出配置界面。

8.4 配置设备树

本章节我们直接使用内核中配置好的设备树文件, 文件在"atk-zynq-linuxxlnx/arch/arm/boot/dts/"路径下, 对于 ZYNQ 7020 来说, 需要"atk-navigator-7020.dts"、 "system-user.dtsi"、"navigator-7020-pl.dtsi"、"navigator-7020-pcw.dtsi"和"zynq-7000.dtsi"这几个设备树文件。除了"system-user.dtsi"外, 其它几个设备树文件已经包含在 petalinux 工程中且不需要修改, 所以只需要把内核中"system-user.dtsi"文件复制到 petalinux 工程"project-spec/meta-user/recipes-bsp/device-tree/files/"路径下就行。

先删除 petalinux 工程下原有的 system-user.dtsi 文件,再将内核中的 system-user.dtsi 复制 到工程路径下,如下图所示:

<pre>sqd@sqd-virtual-machine:~/petalinux/ALIENTEK-ZYNQ-disp-dev/project-spec/meta-use</pre>
r/recipes-bsp/device-tree/files\$ ls
pl-custom.dtsi system-user.dtsi
<pre>sqd@sqd-virtual-machine:~/petalinux/ALIENTEK-ZYNQ-disp-dev/project-spec/meta-use</pre>
r/recipes-bsp/device-tree/files\$ rm system-user.dtsi
<pre>sqd@sqd-virtual-machine:~/petalinux/ALIENTEK-ZYNQ-disp-dev/project-spec/meta-use</pre>
<pre>r/recipes-bsp/device-tree/files\$ cp /home/sqd/workspace/kernel-ch8/atk-zynq-linu</pre>
x-xlnx/arch/arm/boot/dts/system-user.dtsi .
<pre>sqd@sqd-virtual-machine:~/petalinux/ALIENTEK-ZYNQ-disp-dev/project-spec/meta-use</pre>
r/recipes-bsp/device-tree/files\$ ls
pl-custom.dtsi system-user.dtsi
团的人工转换几个社文件

图 8.4.1 替换设备树又件

这里需要对 system-user.dtsi 做一些修改,如下所示:

```
原子哥在线教学: www.yuanzige.com
                                              论坛:www.openedv.com/forum.php
                                        示例代码 system-user.dtsi
    1 #include <dt-bindings/gpio/gpio.h>
    2 #include <dt-bindings/input/input.h>
    3 #include <dt-bindings/media/xilinx-vip.h>
    4 #include <dt-bindings/phy/phy.h>
    5
    6 / {
    7
        model = "Alientek Navigator Zynq Development Board";
    8
         compatible = "xlnx,zynq-zc702", "xlnx,zynq-7000";
    9
        //此处添加 chosen 节点方便用于直接拷贝 system-user.dtsi 用于 petalinux 工程
    10
    11
         //chosen {
         // bootargs = "console=ttyPS0,115200 cma=50M earlycon root=/dev/mmcblk0p2 rw rootwait";
    12
         // stdout-path = "serial0:115200n8";
    13
    14
         //};
    15
    16
         leds {
    17
           compatible = "gpio-leds";
    18 ...
```

F点原子

将第 11 行到第 14 行注释掉,这是因为第 12 行 bootargs 环境变量将根文件系统指定到 tf 卡第二个分区,而本小节实验根文件系统只需要放在 image.ub 中就行。

此外,对于 ZYNQ 7010 来说,修改 system-user.dtsi 之后,还需要在替换后的 system-user.dtsi 中添加宏定义 "#define Z7010",如下所示:

示例代码 system-user.dtsi

```
2 #include <dt-bindings/gpio/gpio.h>
3 #include <dt-bindings/input/input.h>
4 #include <dt-bindings/media/xilinx-vip.h>
5 #include <dt-bindings/phy/phy.h>
6
7 /{
8
     model = "Alientek Navigator Zynq Development Board";
9
     compatible = "xlnx,zynq-zc702", "xlnx,zynq-7000";
10
     //此处添加 chosen 节点方便用于直接拷贝 system-user.dtsi 用于 petalinux 工程
11
12
     //chosen {
     // bootargs = "console=ttyPS0,115200 cma=50M earlycon root=/dev/mmcblk0p2 rw rootwait";
13
     // stdout-path = "serial0:115200n8";
14
15
     //};
16 ...
```

1 #define Z7010



```
原子哥在线教学: www.yuanzige.com
                                               论坛:www.openedv.com/forum.php
    . . .
                      //7010的开发板因为 pl 资源不够,没有添加 hdmi
    298 #ifndef Z7010
    299 &hdmi_out_clk_wiz_dyn{
          compatible = "xlnx,clocking-wizard";
    300
    301
          xlnx,nr-outputs = <2>;
    302 };
    303
    304 &hdmi_out_hdmi_vtc {
    305
          compatible = "xlnx,bridge-v-tc-6.1";
    306
          xlnx, pixels-per-clock = <1>;
    307 };
    308 #endif
    ...
    416 #ifndef Z7010
    417
          drm_pl_disp_hdmi {
    418
            compatible = "xlnx,pl-disp";
    419
            dmas = <&hdmi_out_v_frmbuf_rd_0 0>;
    420
            dma-names = "dma0";
    421
            xlnx,vformat = "BG24";
    422
            xlnx,bridge = <&hdmi_out_hdmi_vtc>;
    423
    424
            ports {
    425
              #address-cells = <1>;
    426
              \#size-cells = <0>;
    427
              port@0 {
    428
                reg = <0>;
    429
                pl_disp_crtc_hdmi:endpoint {
                   remote-endpoint = <&hdmi_encoder>;
    430
    431
                };
    432
              };
    433
            };
          };
    434
    435
    436
          atk_hdmi_drm{
    437
            compatible = "atk,atk-hdmi";
    438
            status = "okay";
    439
    440
            clocks = <&hdmi_out_clk_wiz_dyn 0>, <&hdmi_out_clk_wiz_dyn 1>;
    441
            clock-names = "hdmi_pclk", "hdmi_pclk_5x";
    442
            clk-num = <2>;
                               //时钟数
    443
            ddc-i2c-bus = \langle \&i2c1 \rangle;
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

444	hpd-gpios = <&gpio0 64 GPIO_ACTIVE_HIGH>;			
445				
446	ports {			
447	#address-cells = <1>;			
448	<pre>#size-cells = <0>;</pre>			
449	port@0 {			
450	reg = <0>;			
451	hdmi_encoder: endpoint {			
452	remote-endpoint = <&pl_disp_crtc_hdmi>;			
453	};			
454	};			
455	};			
456	};			
457 #	457 #endif			

第1行,添加宏定义"Z7010",这是因为 ZYNQ7010 资源少,需要去掉 HDMI 显示部分。第298行到第308行和第416到第457行就是设备树中 HDMI 显示部分相关的节点。 到这里设备树就添加好了。

8.5 配置根文件系统

本章需要用到 libdrm 库中的 modetest 命令来对显示设备进行基本的显示测试,所以需要 在根文件系统中添加 libdrm 库。

输入下面命令配置根文件系统:

petalinux-config -c rootfs

打开根文件系统配置界面,如下图所示:



图 8.5.1 配置根文件系统

依次进入 "Filesystem Packages--- >x11--- >base--- >libdrm" 菜单中, 然后使能 "libdrm-tests", 如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

Libdrm Arrow keys navigate the menu. <enter> selects submenus> (or empty submenus). Highlighted letters are hotkeys. Pressing <y> includes, <n> excludes, <m> modularizes features. Press <esc><esc> to exit, <? > for Help, for Search. Legend: [*] built-in []</esc></esc></m></n></y></enter>
<pre>[] libdrm [] libdrm-omap [] libdrm-amdgpu [] libdrm-dev [] libdrm-dbg [] libdrm-drivers [] libdrm-nouveau [*] libdrm-nouveau [*] libdrm-freedreno [] libdrm-radeon + (+)</pre>
<pre><select> < Exit > < Help > < Save > < Load ></select></pre>

图 8.5.2 添加 libdrm-tests 工具

保存配置并退出。

8.6 编译 Petalinux 工程

在终端输入如下命令编译整个 petalinux 工程:

petalinux-build 执行结果如下图所示: sqd@sqd-virtual-machine:~/petalinux/ALIENTEK-ZYNQ-disp-dev\$ petalinux-build INFO: Sourcing build tools [INFO] Building project [INF0] Sourcing build environment [INFO] Generating workspace directory INFO: bitbake petalinux-image-minimal Parsing of 2995 .bb files complete (0 cached, 2995 parsed). 4265 targets, 204 sk ipped, 0 masked, 0 errors. NOTE: Resolving any missing task queue dependencies Checking sstate mirror object availability: 100% |################ Time: 0:00:18 Sstate summary: Wanted 154 Found 25 Missed 129 Current 826 (16% match, 86% compl ete) NOTE: Executing Tasks NOTE: Setscene tasks completed NOTE: Tasks Summary: Attempted 3541 tasks of which 3126 didn't need to be rerun and all succeeded. INFO: Successfully copied built images to tftp dir: /tftpboot [INFO] Successfully built project

图 8.6.1 编译 petalinux 工程

8.7 制作 BOOT.bin 启动文件并复制到 SD 卡

使用下面命令生成 BOOT.bin 文件:

petalinux-package --boot --fsbl --fpga --u-boot --force

image.ub	rootfs.cpio.gz.u-boot	system.bit	uImage		
pxelinux.c	fg rootfs.jffs2	system.dtb	vmlinux		
sqd@sqd-vi	rtual-machine:~/petalinux/	ALIENTEK-ZYNQ-	<pre>-disp-dev/images</pre>	/linux\$	cp BOOT
.BIN boot.s	scr image.ub /media/sqd/boo	ot/ 🔶 🛛 将启:	动镜像复制到SD卡		
sqd@sqd-vi	rtual-machine:~/petalinux//	ALIENTEK-ZYNQ-	disp-dev/images	/linux\$	sync
sqd@sqd-vi	rtual-machine:~/petalinux/	ALIENTEK-ZYNQ-	disp-dev/images	/linux\$	umount
/dev/sdb*	←→ 卸载SD卡				
umount: /de	ev/sdb: not mounted.				
sad@sad-vi	rtual-machine:~/petalinux/	AL TENTEK-ZYNO-	disn_dev/images	/linux\$	

图 8.7.1 复制镜像文件到 sd 卡

8.8 开发板上启动 linux

8.8.1 启动 linux 并在 lcd 屏上显示打印信息

将 SD 卡插入开发板的 SD 卡槽,然后通过 USB Type-C 连接线将开发板 USB 调试串口与电脑 USB 口相连,连接电源线、设置开发板启动模式为 SD Card,使用软排线将 LCD 屏连接到开发板,对于 7020 开发板,也可以将 HDMI 线接到开发板。接下来打开电源开关、启动开发板,连接串口终端,打印信息如下所示:



图 8.8.1 启动打印信息

此时,LCD 屏也会显示打印信息,如下图所示:



原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php



图 8.8.2 LCD 屏显示信息

此时可以在开发板 USB HOST 接口上连接一副 USB 键盘,使用键盘输入用户名和密码就可以登录 Linux 系统了(登录的用户名为 root,密码为: root),登录成功之后大家就可以通过 LCD 终端来进行相关的操作了,譬如输入 Linux 命令、执行 Linux 命令并打印显示 Linux 命令运行结果,此处我们就不演示了。

笔者这里是以 7 寸 1024*600 屏为例的,如果大家手上有其它正点原子推出的 LCD RGB 屏,也可以连接测试。

8.8.2 测试显示设备

DRM 是 Linux 下的图形显示架构, libdrm 是 DRM 架构下的接口库,为上层应用程序提供 API 接口。本小节使用 libdrm 库提供的"modetest"测试工具。

运行下面命令测试 LCD 屏显示:

modetest -D amba_pl:drm_pl_disp_lcd -s 35@33:1024x600@BG24

上面命令中参数 D 表示设备,参数 s 格式为<connector_id>@<crtc_id>:<mode>@<format>, 这些参数可以通过 "modetest -D amba_pl:drm_pl_disp_lcd -c" 命令查询。

LCD 屏显示效果如下所示:



原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php



图 8.8.3 LCD 屏显示测试

对于 ZYNQ7020 开发板,可以运行下面两个命令测试 HDMI 显示: modetest -D amba_pl:drm_pl_disp_hdmi -s 35@33:1920x1080@BG24 -P 32@33:1920x1080 或者

modetest -D amba_pl:drm_pl_disp_hdmi -s 35@33:1280x720@BG24 -P 32@33:1280x720 分别是 HDMI 的 1080P 和 720P 显示,同样可以看到 HDMI 显示器能显示彩条。 原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

正点原子

第九章 使用 Vitis 开发 Linux 应用

上一章我们学习了在 Petalinux 搭建的 Linux 系统上基础外设的使用,本章我们以创建 "Hello World" 工程为例学习如何使用 Vitis 开发 Linux 应用以及如何让应用程序运行在 Petalinux 搭建的 Linux 系统上。运行方式本章介绍了三种,可根据个人喜好选择。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

9.1 创建基于 Vitis 的 Linux 平台工程

双击 Ubuntu 系统桌面的 Xilinx Vitis IDE 2020.2 图标打开 Vitis 软件,弹出如下图所示的 Launcher 界面:

Vitis IDE Launcher 😣 😣			
Select a directory as workspace			
Vitis IDE uses the workspace directory to store its preferences and development artifacts.			
Workspace: //home/sqd/workspace/linux_app/hello_world			
Use this as the default and do not ask again			
Restore other Workspace			
Recent Workspaces			
Cancel Launch			

图 9.1.1 打开 Vitis Launcher 界面

设置 Workspace 为"/workspace/linux_app/hello_world",如图 9.1.1 所示,然后点击"Launch"按钮,进入如下图所示界面:

File Edit Search Xilinx Project Window	Help		
P □ Welcome S VILINX VITIS.			-
	VITIS IDE		
	PROJECT	PLATFORM	RESOURCES
	Create Application Project	Add Custom Platform	Vitis Documentation
	Create Platform Project		Xilinx Developer
	Create Library Project		
	Import Project		

图 9.1.2 创建 Platform 工程

单击"Create Platform Project",即创建平台工程,进入下图所示界面:



.

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

Create	new	platform	project
			project

Enter a name for your platform project

This wizard will guide you through creation of a platform project from the output of Vivado [Xilinx Shell Archive (XSA)] or from an existing platform. A platform will enable you to specify options for the kernels, BSPs, as well as settings required for creating new applications. Platforms are currently supported for embedded software developers. Platform project name: Dev_xc7z020 • A platform provides hardware information and software System Platform environment settings. Project Project • A system project contains one or more applications that run at the same time. Domain Арр • A domain provides runtime for applications, such as operating system or BSP. • A workspace can contain unlimited platforms and unlimited XSA system projects. A new platform project can be created from one of the two inputs: From hardware specification (XSA) Create a new platform project from a hardware specification file. You can specify the OS and processor to start with. The platform can be customized later from the platform project editor. From existing platform Load the platform definition from an existing platform. You can choose any platform from the platform repository as a base for your platform project. ? < Back Next > Cancel Finish

图 9.1.3 平台工程配置选项

"Platform project name"栏输入平台工程名"Dev_xc7z020",其他保持默认,然后单击 底部的"Next>",进入下一页面,如下图所示。

Platform

oose a platform for y SA)' tab.	our project. You can also create an application from XSA through the 'Create a new platform from hardware
Create a new plat	form from hardware (XSA) 🔄 Select a platform from repository
Hardware Specificat XSA File: /home/s	on
Software Specificati Specify the details clicking the platfor	on for the initial domain to be added to the platform. More domains can be after the platform is created by double m.spr file
Operating system:	linux • 9 选择Linux系统
Processor:	ps7_cortexa9
details in the Boot components Generate boot of 3, I[X	omponents 消勾选
)	<back next=""> Cancel Finish</back>
	图 9.1.4 创建新的平台工程


原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php

在 "Create a new platform from hardware(XSA)"标签页中选择创建平台需要的 xsa 文件, 这里我们使用 6.3.4 小节使用的 "system_wrapper.xsa"文件, "Operating system"选择 "linux", 然后取消勾选 "Generate boot components", 如图 9.1.4 所示。设置完成后点击 "Finish", 进入下图所示界面:

	hello_world - Dev_	_xc7z020/platform.spr - V	itis IDE		
File Edit Search Xilinx Project Windov	v Help				
🖻 🕶 🔄 🔞 👻 🔸 🗸 🖬 🗊 🗊 🏘 🕶 🔿 🗸 🤞	? ▼ ∜⊃ ⇔ ⇒ ⇒			Q 🛛 🕜 Desig	,亦 称 Debug
🖳 Explorer 🛛 🛛 🖻 🕾 🕴 🗆 🖬	✓ Dev_xc7z020 🛛			🗄 Outline 🛿	- 0
 ✓ Dev_xc7z020 > hw > blogs > resources ✓ platform.spr ※ platform.tcl 	I Dev_xc7z020 ▼ ☐ Dev_xc7z020 ▼ ☐ Inux on ps7_cortexa9 ☐ Libraries	Platform: Dev_xc7z02 Name: Hardware Specification: Description: Samples: Generate boot comp Pre-built Component FSBL:	20 Dev_xc7z020 system_wrapper.xsa Dev_xc7z020 connents ts	There is no act	ive editor an outline.
		. Oall			
	Platform Tcl Console	s Log 🕕 Guidance		Ex 🏭 🖙 🔽 🖵 🔻	<u> </u>
	platform create -name {Dev_xc7 platform read {/home/sqd/works platform active {Dev_xc7z020}	z020} -hw {/home/sqd/pe pace/linux_app/hello_we	etalinux/xsa_7020/system_wrapper.xsa} orld/Dev_xc7z020/platform.spr}	· -proc {ps7_co	ortexa9} -

图 9.1.5 进入 Vitis 工程界面

现在我们将开发板资料盘(A 盘)\4_SourceCode\3_Embedded_Linux\zynq_petalinux\zynq7020\1_customize_linux\software\linux _boot文件夹拷贝到 Vitis Workspace 文件夹/home/sqd/workspace/linux_app/hello_world下,如 下图所示:



图 9.1.6 解压缩后的 linux_boot 文件夹

回到 Vitis 工程界面,设置 "Bif File"为 linux_boot 文件夹下的 linux_image.bif 文件,设置 "Boot Components Directory"为 linux_boot 文件夹,如下图所示:



原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php

🔁 🕶 🗐 🔞 🕶 🐔 🕶 🖸 💋 🖬 🎋 🕶 🕻) ▼					Q
强 Explorer 🛛 🛛 🖯 🕄 🖿 🗆	✓ Dev_xc7z020 ⊠					
 ✓ Image: Performance of the second se	type filter text ■ ⊕ ♥ ¥ ■ ⊕ ev_xc7z020 ■ ☐ ps7_cortexa9 ■ Cibraries	Domain: linux_domain OS: Processor: Supported Runtimes: Display Name:	linux ps7_cortexa9 C/C++ linux on ps7_cortexa9			
		Description:	linux domain			
						/
		Bif File:	/home/sqd/workspace/linux_app/hello_world/linux_boot/linux_image.bif	<u>B</u> rowse	Q	
		Boot Components Directory:	/home/sqd/workspace/linux_app/hello_world/linux_boot	B <u>r</u> owse	Q	R
		Linux Image Directory:		Br <u>o</u> wse	Q	R
Dev xc7z020 [Platform]		Linux Rootfs:		Browse	Q,	
		Bootmode	SD •			
		Sysroot Directory:		Bro <u>w</u> se	Q	
	Main Hardware Specification	OFMU Data:	I II h I het her kanadi i I i i I i e I i e e	•		

图 9.1.7 设置 Vitis 平台工程

设置完成后编译平台工程如下图所示:



图 9.1.8 编译平台工程

编译完平台工程后,平台工程的搭建就完成了。现在可以创建 Linux 应用工程了。

9.2 创建 Linux 应用工程

点击 Vitis 菜单栏的 "File->New->Application Project...", 创建应用工程, 如下图所示:



图 9.2.1 创建应用工程



进入平台选择界面后可以看到我们之前创建好的平台,这里不需要修改其它设置,点击 "Next",如下图所示:

		eace a new placion	in from naroware (A3A)	
d:					🕂 Add 🏶 Manage
ame	Boar	d Flow	Vendor	Path	
Dev_xc7z020 [custor	ו]	Embedded SW	/ Dev xilinx	/home/sqd/workspace/li	inux_app/hello_world/
tform Info	切延好的十百				
atform Info eneral Info Name: Dev_xc72	创建好的干古	Acceleration Reso The selected plat	Irces	Domain Details Domains	
tform Info eneral Info Name: Dev_xc7z Part: xc7z020c	的 20 1g400-2	Acceleration Reso The selected plat application accel	urces- form does not have eration capabilities	Domain Details Domains Domain name	e Details
Ltform Info eneral Info Name: Dev_xc7z Part: xc7z020c Family: zynq	回建好的下音 020 1g400-2	Acceleration Reso The selected plat application accel	urces form does not have eration capabilities	Domain Details Domains Domain name linux on ps7_corte	e Details exa9 CPU: cortex-a9 OS: linux
atform Info eneral Info Name: Dev_xc7z Part: xc7z020c Family: zynq Description:	020 1g400-2	Acceleration Reso The selected plat application accel	urces form does not have eration capabilities	Domain Details Domains Domain nam linux on ps7_corte	e Details exa9 CPU: cortex-a9 OS: linux

图 9.2.3 选择平台

领航者 ZYNQ 之嵌入式 Linux 开发指南	② 正点原子
原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com	n/forum.php
设置工程名为"hello world",点击"Next",如下图所示:	T T
Application Project Details Specify the application project name and its system project properties	
Application project name: hello_world	
System Project Create a new system project for the application or select an existing one from the workpsace ()	
Select a system project System project details	
Create new System project name: hello_world_system	
Target processor	
Select target processor for the Application project.	
Processor Associated applications ps7_cortexa9 SMP hello_world	
Show all processors in the hardware specification	0
(?) < Back	ancel Finish

图 9.2.4 设置工程名

进入 Domain 设置页面,该设置页面中我们不做修改,直接点击"Next",如下图所示:

•••

Domain

Note: New domain created by this wizard will have all the requirements of the application template selected i	in the next step
---	------------------

Select a domain	Domain details			
linux on ps7_cortexa9				
- Create any	Name:	linux_domain		
Create new	Display Name:	linux on ps7_cortexa9		
	Operating System:	linux	*	
	Processor:	ps7_cortexa9		
	Application settings			
	Sysroot path:			Browse
	Root FS:			Browse
	Kernel Image:			Browse
•	< E	Back Next >	Cancel	Finish

图 9.2.5 Domain 设置界面



原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php

进入应用工程模板选择页面,选择"Linux Hello World",然后点击"Finish",如下图 所示:

Available Templates:		
Find:	Linux Hello World	
 SW development templates Empty Application (C++) Linux Empty Application Linux Hello World 	Let's say 'Hello World' in C.	

图 9.2.6 选择应用工程模板

创建好应用工程后,选中应用工程然后点击编译,如下图所示:



图 9.2.7 编译应用工程

到这里应用工程就创建好了。展开左侧的"hello_world"下的 src,可以看到 helloworld.c 文件,双击可以看到右侧的源码,是不是很简单,跟我们平时写的 C 应用没区别。



原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php



图 9.2.8Vitis 界面

下面我们将介绍如何将该工程的 elf 文件运行在我们第六章搭建的 Linux 上。

有三种方式,一种是通过 Vitis 软件自带的 TCF Agent,另一种是通过其他方式如 NFS、 ssh 等将文件共享到或传到开发板上的 Linux 系统中。无论使用什么方式,都需要开发板连接 网线,且与 Windows 系统处于同一网络,也就是 Ubuntu 虚拟机和开发板要相互 ping 通。

首先用网线连接电脑和开发板 PS 端网口,运行我们第六章搭建的 Linux 系统。系统启动 后,在串口上位机中输入"ifconfig eth0 192.168.1.10 netmask 255.255.255.0"命令设置开发板静态 IP 地址为 192.168.1.10,使用"ifconfig"命令查看设置的 IP 地址,运行结果如下图所示:



原子哥在线教学:	www.yuanzige.com	论坛:www.openedv.com/forum.php
root@ALIE root@ALIE eth0	NTEK-ZYNQ:~# ifconfig eth0 NTEK-ZYNQ:~# ifconfig eth0 Intek-ZYNQ:~# ifconfig eth0 inet addr:192.168.1.10 B UP BROADCAST RUNNING MULT RX packets:849 errors:0 d TX packets:374 errors:0 d collisions:0 txqueuelen:1 RX bytes:39783 (38.8 KiB) Interrupt:30 Base address	192.168.1.10 netmask 255.255.255.0 dr 00:0A:35:00:8B:87 ccast:192.168.1.255 Mask:255.255.255.0 ICAST MTU:1500 Metric:1 ropped:0 overruns:0 frame:0 ropped:0 overruns:0 carrier:0 000 TX bytes:89579 (87.4 KiB) :0xb000
ιο	Link encap:Local Loopback inet addr:127.0.0.1 Mask inet6 addr: ::1/128 Scope UP LOOPBACK RUNNING MTU: RX packets:0 errors:0 dro TX packets:0 errors:0 dro collisions:0 txqueuelen:1 RX bytes:0 (0.0 B) TX by	:255.0.0.0 :Host 65536 Metric:1 pped:0 overruns:0 frame:0 pped:0 overruns:0 carrier:0 000 tes:0 (0.0 B)
root@ALIE PING 192. 64 bytes 64 bytes 64 bytes 64 bytes 64 bytes ^C 192.1 5 packets round-tri root@ALIE	<pre>ENTEK-ZYNQ:~# ping 192.168. 168.1.20 (192.168.1.20): 5 from 192.168.1.20: seq=0 t from 192.168.1.20: seq=1 t from 192.168.1.20: seq=2 t from 192.168.1.20: seq=3 t from 192.168.1.20: seq=4 t 4.68.1.20 ping statistics</pre>	1.20 6 data bytes tl=64 time=0.397 ms tl=64 time=0.313 ms tl=64 time=0.309 ms tl=64 time=0.318 ms - ceived, 0% packet loss 3/0.397 ms

图 9.2.9 开发板的 IP 地址

设置好后,此时开发板若能 ping 通 Ubuntu 主机,那么网络环境就搭建好了,如果 ping 不通,参考 7.7 小节重新设置网络环境。

完成准备工作后,我们先来看下 TCF Agent 方式,然后试下 NFS 方式和 SSH 方式。

9.3 使用 TCF Agent 方式运行

在 Vitis 界面,右键点击 "hello_world"应用工程,在弹出的菜单中选择 "Run As—>Run Configuration",如图 9.3.1 所示。



图 9.3.1 运行"Run Configuartion"

the state of the second second		プエネホー
`	nzige.com 论坛:www.openedv.com/foru	m.php
在弹出的界面中	,双击"Single Applicatuin Debug"	, 会自动创
ebugger_hello_world-Def	fault",然后点击"New"创建 Linux Agent 连	连接,如下图所示
Create, manage, and run configurations		
Debug a program using Application Debug	gger.	
☞ 200 × 87 - 1、双击	Name: Debugger_hello_world-Default	
type filter text	🛛 🗱 Main 🔲 Application 🞯 Target Setup 🕬 Arguments 📼 Environment 🚋 Symb	ool Files 🖗 Source
✓ ♣ Single Application Debug ♣ Debugger hello world-Default	2. 占击"New	** ₂
Single Application Debug (GDB)	Debug Type: Linux Application Debug	
	Connection:	
	Note: TCF agent port should be used as port in the target connection (Default TCF a	gent port: 1534).
	Project: hello_world	Browse
	Configuration: Debug	•]
	Emulation Performance Analysis	
Filter matched 4 of 4 items	Reve	Apply
0	Clo	se Run
	図 0 2 2 创建 Linux A cont	
	图 9.3.2 创建 LinuxAgent	
New Target Connection	图 9.3.2 创建 LinuxAgent	
New Target Connection Creates new configuration for co	图 9.3.2 创建 LinuxAgent	
New Target Connection Creates new configuration for co	图 9.3.2 创建 LinuxAgent	
New Target Connection Creates new configuration for co Target Name linux	图 9.3.2 创建 LinuxAgent Innecting to a target. 、输入目标名称,任意名都可以	
New Target Connection Creates new configuration for co Target Name linux 1. Set as default target	图 9.3.2 创建 LinuxAgent mnecting to a target. 输入目标名称,任意名都可以	
New Target Connection Creates new configuration for co Target Name linux 1. Set as default target Specify the connection type and	图 9.3.2 创建 LinuxAgent mnecting to a target. 输入目标名称,任意名都可以	
New Target Connection Creates new configuration for co Target Name linux 1. Set as default target Specify the connection type an Type Linux TCF Agent	图 9.3.2 创建 LinuxAgent annecting to a target. 、输入目标名称,任意名都可以 nd propertles	
New Target Connection Creates new configuration for co Target Name linux 1. Set as default target Specify the connection type ar Type Linux TCF Agent Host 192.168.1.10	图 9.3.2 创建 LinuxAgent mnecting to a target. 、输入目标名称,任意名都可以 nd properties 2、输入开发板 ip 地址	
New Target Connection Creates new configuration for co Target Name linux 1 Set as default target Specify the connection type an Type Linux TCF Agent Host 192.168.1.10 Port 1534	图 9.3.2 创建 LinuxAgent mnecting to a target. 、输入目标名称,任意名都可以 nd properties 2、输入开发板 ip 地址:	
New Target Connection Creates new configuration for co Target Name linux	图 9.3.2 创建 LinuxAgent	
New Target Connection Creates new configuration for co Target Name linux 1. Set as default target Specify the connection type and Type Linux TCF Agent Host 192.168.1.10 Port 1534 Advanced >> 3.	图 9.3.2 创建 LinuxAgent Innecting to a target. 、输入目标名称,任意名都可以 nd properties 2、输入开发板 ip 地址 测试连接是否成功	

图 9.3.3 输入 Host 的 IP 地址

输入Host的IP地址,也就是开发板的IP地址,笔者的为192.168.1.10。输入IP地址完成后,点击"Test Connection",如果出现下图所示的结果,表示连接测试成功,Linux TCF Agent 服务运行正常,可以与主机(开发板)连接。点击"OK"按钮,完成设置。





原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

点击图 9.3.5 中"OK"完成 Linux Agent 连接的创建。然后分别点击"Run Configurations" 界面中的"Apply"和"Run"按钮开始运行程序,如下图所示:

	Name: Debugger_hello_world-Default	
type filter text	🕅 🚺 Application 📀 Target Setup 🏁 Arguments 🖾 Environment 🚡	Symbol Files 🦻 Source
🕫 💱 Single Application Debug		»» ₂
Lebugger_hello_world-Default		
Single Application Debug (GDB)		
SPM Analysis	Connection: Linux New	
	Note: TCF agent port should be used as port in the target connection (Default T	CF agent port: 1534).
	Project: ballo world	Browro
	Hello_world	Diowse
	Configuration: Debug	
	E renominance Analysis	
		1
ilter matched 4 of 4 items		Revert Apply

图 9.3.5 运行

在 Console 终端可以看到 Hello World 输出,表明程序在 Linux 上运行成功,如下图所示:



图 9.3.6Console 终端输出运行结果

9.4 使用 NFS 共享方式运行

在 4.4.1 节我们使用/home/sqd/workspace/nfs 文件夹供 nfs 服务器使用,所以我们首先将工 程的 elf 文件复制到 Ubuntu 虚拟机的该文件夹下。

复制完成后,我们在开发板上挂载 Ubuntu 主机的 NFS 目录,在挂载之前要知道 Ubuntu 主机的 IP 地址, 使用 ifconfig 命令或"ip a s"命令查询到笔者的 Ubuntu 主机的 IP 地址为 192.168.1.20。在连接到开发板的串口上位机中输入如下命令进行挂载:

mount -t nfs -o nolock 192.168.1.20:/home/sqd/workspace/nfs /mnt 其中 192.168.1.20 为笔者的 Ubuntu 主机 IP 地址,/mnt 为挂载到开发板的目录。 在串口终端中输入"ls /mnt"可以看到 hello_world.elf 文件,输入如下命令执行该 elf 文件: chmod +x /mnt/hello world.elf #赋予可执行权限 /mnt/hello_world.elf #执行 执行结果如下图所示:



原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php

root@ALIENTEK-ZYNQ:~# mount -t nfs -o nolock 192.168.1.20:/home/sqd/workspace/nfs /mnt root@ALIENTEK-ZYNQ:~# ls /mnt/ hello_world.elf root@ALIENTEK-ZYNQ:~# chmod +x /mnt/hello_world.elf root@ALIENTEK-ZYNQ:~# /mnt/hello_world.elf Hello World

图 9.4.1 执行 elf 文件

可以看到打印出"Hello World"信息,表明执行成功。

9.5 使用 SSH 方式运行

SSH方式是将工程的 elf 文件传到开发板上的 Linux 系统中,在 Ubuntu 主机终端中输入如下命令将 elf 文件传到开发板的 Linux 系统中(上一节已经工程的 elf 文件复制到 /home/sqd/workspace/nfs 文件夹):

 $scp \ /home/sqd/workspace/nfs/hello_world.elf \ \underline{root@192.168.1.10:/home/root}$

上面命令的 192.168.1.10 为笔者使用的开发板的 IP 地址,/home/root 表示传到开发板的该目录下,也就是 root 用户所在的根目录。

执行结果如下图所示:



图 9.5.1scp 命令传输文件

需要输入开发板上 Linux 系统的 root 用户密码,默认为"root"。传输完成后,在串口终端 中输入 "ls" 会看到有一个 hello_world.elf 文件。输入 "./hello_world.elf" 会打印出 "Hello World" 信息,表明运行成功。在串口上位机中输入的命令及结果如下图所示:



图 9.5.2 命令及结果



原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php

第十章 Petalinux 构建 Qt 和 OpenCV 交叉编译开发环境

我们在前面实验使用的 Petalinux 工具中包含 linux 交叉编译链,可以用来编译 uboot、 kernel 和 linux 应用程序。

对于 2019.1 之前版本的 Petalinux,设置 Petalinux 工作环境变量后可以直接使用 arm 的 linux 交叉编译工具链,然而此后版本的 Petalinux 包括我们当前使用的 Petalinux 在设置环境 变量后只能得到裸机的交叉编译工具链,无法获得 linux 交叉编译工具链,所以本章以构建 Qt 和 OpenCV 交叉编译开发环境为例讲解如何获得 linux 交叉编译工具链。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

10.1 简介

10.1.1 Petalinux 的交叉编译工具链

交叉编译是编译技术发展过程中的一个重要分支。通俗的说,交叉编译就是在一个平台 上生成另一个平台上的可执行代码。这里提到的平台有两方面的含义:处理器的体系结构和 所运行的操作系统。

通常,程序是在一台计算机上编译,然后再分布到将要使用的其它计算机上。当主机系 统(运行编译器的系统)和目标系统(产生的程序将在其上运行的系统)不兼容时,该过程 就叫做交叉编译。嵌入式系统应用的日益广泛和需求推动了交叉编译技术的发展。非 X86 的 嵌入式系统开发,一般都是基于 X86 的 PC 平台进行,这种开发方式就是交叉编译。嵌入式系 统中的应用程序都是通过交叉编译得到的。

我们使用 Petalinux 工具编译,显示 Petalinux 工具是带有交叉编译工具链的。设置 Petalinux 的工作环境后,输入"arm",然后按两次 TAB 键,会显示当前环境中,以"arm" 开头的工具和命令,包括交叉编译的工具和命令,如下图所示:

zy@zy-virtual-machine:~\$ arm	
arm2hpdl	armr5-none-eabi-addr2line
arm-none-eabi-addr2line	armr5-none-eabi-ar
arm-none-eabi-ar	armr5-none-eabi-as
arm-none-eabi-as	armr5-none-eabi-c++filt
arm-none-eabi-c++filt	armr5-none-eabi-cpp
arm-none-eabi-cpp	armr5-none-eabi-dwp
arm-none-eabi-dwp	armr5-none-eabi-elfedit
arm-none-eabi-elfedit	armr5-none-eabi-g++
arm-none-eabi-g++	armr5-none-eabi-gcc
arm-none-eabi-gcc	armr5-none-eabi-gcc-ar
arm-none-eabi-gcc-ar	armr5-none-eabi-gcc-nm
arm-none-eabi-gcc-nm	armr5-none-eabi-gcc-ranlib
arm-none-eabi-gcc-ranlib	armr5-none-eabi-gcov
arm-none-eabi-gcov	armr5-none-eabi-gcov-dump
arm-none-eabi-gcov-dump	armr5-none-eabi-gcov-tool
arm-none-eabi-gcov-tool	armr5-none-eabi-gdb
arm-none-eabi-gdb	armr5-none-eabi-gdb-add-index
arm-none-eabi-gdb-add-index	armr5-none-eabi-gprof
arm-none-eabi-gprof	armr5-none-eabi-ld
arm-none-eabi-ld	armr5-none-eabi-ld.bfd
arm-none-eabi-ld.bfd	armr5-none-eabi-ld.gold
arm-none-eabi-ld.gold	armr5-none-eabi-nm
arm-none-eabi-nm	armr5-none-eabi-objcopy

图 10.1.1 以"arm"开头的工具和命令

其中以"arm"开头的是 arm 的交叉编译工具链,显示的 arm 交叉编译工具链只有 none, 也就是裸机下的交叉编译工具链,不能编译 linux,那么用于 linux 的交叉编译工具链是啥样的 呢,如下图所示:

· · · · · · · · · · · · · · · ·	·····
原子哥在线教学:www.yuanzige.com 论	论坛:www.openedv.com/forum.php
<pre>sqd@sqd-virtual-machine:~\$ arm</pre>	
Display all 117 possibilities? (y or n)	
arm2hpdl	arm-xilinx-linux-gnueabi-addr2line
arm-none-eabi-addr2line	arm-xilinx-linux-gnueabi-ar
arm-none-eabi-ar	arm-xilinx-linux-gnueabi-as
arm-none-eabi-as	arm-xilinx-linux-gnueabi-c++filt
arm-none-eabi-c++filt	arm-xilinx-linux-gnueabi-cpp
arm-none-eabi-cpp	arm-xilinx-linux-gnueabi-dwp
arm-none-eabi-dwp	arm-xilinx-linux-gnueabi-elfedit
arm-none-eabi-elfedit	arm-xilinx-linux-gnueabi-g++
arm-none-eabi-g++	arm-xilinx-linux-gnueabi-gcc
arm-none-eabi-gcc linux交叉编译工具和命令	arm-xilinx-linux-gnueabi-gcc-ar
arm-none-eabi-gcc-ar	arm-xilinx-linux-gnueabi-gcc-nm
arm-none-eabi-gcc-nm 🛛 🔪	arm-xilinx-linux-gnueabi-gcc-ranlib
arm-none-eabi-gcc-ranlib	arm-xilinx-linux-gnueabi-gcov
arm-none-eabi-gcov	arm-xilinx-linux-gnueabi-gcov-dump
arm-none-eabi-gcov-dump	arm-xilinx-linux-gnueabi-gcov-tool
arm-none-eabi-gcov-tool	arm-xilinx-linux-gnueabi-gdb
arm-none-eabi-gdb	arm-xilinx-linux-gnueabi-gdb-add-index
arm-none-eabi-gdb-add-index	arm-xilinx-linux-gnueabi-gprof
arm-none-eabi-gprof	arm-xilinx-linux-gnueabi-ld
arm-none-eabi-ld	arm-xilinx-linux-gnueabi-ld.bfd
arm-none-eabi-ld.bfd	arm-xilinx-linux-gnueabi-ld.gold
arm-none-eabi-ld.gold	arm-xilinx-linux-gnueabi-nm
arm-none-eabi-nm	arm-xilinx-linux-gnueabi-objcopy
arm-none-eabi-objcopy	arm-xilinx-linux-gnueabi-objdump

🔁 正点原子

图 10.1.2 linux 交叉编译工具链

如果我们想编译 linux 内核或 linux 应用就可以用其中的 arm-xilinx-linux-gnueabi-gcc 进行 编译。既然设置 Petalinux 工作环境没有,那么 Petalinux 是怎么构建 linux 系统的呢?其实 Petalinux 在构建 linux 系统过程中会编译生成 linux 交叉编译工具链,然后使用其构建 linux 系 统,可以在 Petalinux 工程下使用 "find . -name "arm-xilinx-linux-gnueabi-gcc"" 命令找到相应 的痕迹。不过这种构建过程中编译生成的 linux 交叉编译工具链是不方便使用的,而且其功能 完整性还不能得到保证,毕竟我们不知道 Petalinux 在构建完成后会怎么处理这些中间件。那 么怎么才能获得实用的 linux 编译工具链呢? 这就是本章的重点了。因为有了 linux 交叉编译 工具链就可以摆脱对 Petalinux 的依赖,直接使用 linux 交叉编译工具链进行编译,方便快捷。

获取 linux 交叉编译工具链需要用 Petalinux 构建 SDK, 然后安装 SDK。本章以开发 Qt 和 OpenCV 为例,构建安装 SDK,所以本章我们就将 Qt 和 OpenCV 也打包进 SDK。所谓的 SDK,也就是软件开发工具集,与 Petalinux 构建的根文件系统息息相关,里面不仅包含有 Petalinux 构建的根文件系统、各种库和头文件,还包含 linux 交叉编译工具链,用来编译 linux 内核及 linux 应用以使其能在 ZYNQ 开发板上运行。

10.1.2 Qt 简介

Qt 是一个跨平台的 C++图形用户界面应用程序开发框架。它既可以开发 GUI 图形用户界面程序,也可用于开发非 GUI 程序,比如控制台工具和服务器。Qt 是面向对象的框架,使用



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php 特殊的代码生成扩展(称为元对象编译器 (Meta Object Compiler, moc))以及一些宏, Qt 很容 易扩展,并且允许真正地组件编程。

Qt 是一个完整的开发框架,其目的旨在简化桌面、嵌入式和移动平台的应用程序和用户 界面的创建。Qt 除了可以绘制漂亮的界面(包括控件、布局、交互),还包含很多其它功能, 比如多线程、访问数据库、图像处理、音频视频处理、网络通信、文件操作等。

大部分应用程序都可以使用 Qt 实现,除了与计算机底层结合特别紧密的,例如驱动开发, 它直接使用硬件提供的编程接口,而不能使用操作系统自带的函数库。

由于Qt良好的跨平台特性,基本上不做修改就可以在Linux上实现同样的界面。Linux操作系统是嵌入式的主力军,广泛应用于消费类电子、工业控制、军工电子、电信/网络/通讯、航空航天、汽车电子、医疗设备、仪器仪表等相关行业。

除了 Qt 的跨平台特性和功能丰富外,笔者选择 Qt 的另一个原因是开源。Qt 有商业版和 <u>开源版</u>(使用 GPL 和 LGPL 开源协议),开源版 Qt 足以满足一般需求。关于 Qt 不同许可证 模式和开发平台下的不同特性,可访问该网页: <u>https://www.qt.io/cn/product/features</u>。进入该 网页后,点击左侧的选择菜单,选择对应的许可证模式和开发平台后,右侧会显示支持的特 性,不支持的特性会变灰。

Q 查找	 ● 商业许可证: 商业许可证保证您的代码的专有性,只有您才能控制最终产品的开发、用户体验保护您的知识产权。 					
软件开发生命周期核心阶段						
Design Develop Deploy	设计工目					
Qt 程序包	以口上只					
All	助力实现酷炫的用户界面和极致	的交互体验。				
许可证 模式	Qt Design Studio 🛛 🔊	Qt Designer 🛛 🛛	Qt Quick Designer			
们可加快式	用于创建动画用户界面的 UI 设计和	使用 Qt Widgets 设计和构建图形用	使用 Qt Quick 设计和构建图形用户			
Commercial ~	开发环境。	户界面(GUI)的工具。已集成到 Qt Creator 中。	界面(GUI)的工具。已集成到 Qt Creator 中。			
开发平台						
All 🗸						
	廾友丄具					
目标平台	Ot 有白己的腔平台 IDF 和主宣的	T.且田它开发应田程序和田户男	而			
All	可以实现一次开发到处发布。		, mi.			
编程语言支持	Qt QmlLive 🗾	GammaRay 🛛	Emulator			
All ~	为快速开发 Qt Quick 应用程序提供 了一个实时重加载环境,大大减少	用于查看和修改 Qt 应用内部属性的 内省工具。	提供目标设备的仿真,以便在应用 程序中测试,调试,			
	了部署和测试 UI 设计中的更改所需		- and a constraint of the berger			
Clear Filters	的时间。					

图 10.1.3 Qt 版本特性

开发 Qt 需要搭建 Qt 开发环境。Qt 有自己的集成开发环境——Qt Creator。Qt Creator 是一 个跨平台的、完整的 Qt 集成开发环境(IDE),其中包括了高级 C++代码编辑器、项目和生成管 理工具、集成的上下文相关的帮助系统、图形化调试器、代码管理和浏览工具等。与常见的 IDE 不同的是, Qt Creator 本身是一个空壳,不包含编译器和 Qt 库,所以后面还需要配置 Qt Creator。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

由于使用 Qt 的人很多, 市面上关于 Qt 的教程也很多, 入门的、深入的都有, 所以本章 就不浪费时间讲解如何安装 Ot 了,读者可自行上网搜索 Ot 的安装,也可阅读本章扩展阅读 部分的"安装 Qt"。

10.1.3 OpenCV 简介

OpenCV 全称 Open Source Computer Vision Library, 是一个开源的计算机视觉库,实现了 图像处理和计算机视觉方面的很多通用算法,我们可以调用它提供的各种 API 接口对图像进 行算法处理,例如图像模糊处理、边缘检测、滤波以及人脸识别等相关应用场景,笔者对此 并不是很了解,大家有兴趣的话可以在网上百度 get 这些知识。

OpenCV 基于 BSD 协议,因此它可免费用于学术和商业用途,其提供 C++、C、Python 和 Java 接口,支持 Windows、Linux、Mac OS、iOS 和 Android 等多种操作系统。

OpenCV 提供了丰富的 API 可简化我们对图像处理相关的编程工作,同样这些 API 也是 封装成链接库的形式,可以是动态链接库(Dynamic link library)或者静态链接库(Static link library);对于嵌入式 Linux 系统来说,我们需要通过交叉编译将 OpenCV 源码编译成可适用 于 ARM 平台的链接库文件,那么如何获取适用于 ARM 平台的链接库文件呢?

答案是可以使用 Petalinux 构建,通过定制根文件系统的方式将 OpenCV 库添加到根文件 系统中,这样就获取到了用于我们开发板的链接库文件。

10.2 定制根文件系统 rootfs

前面介绍了 SDK 与 Petalinux 构建的根文件系统息息相关, 这是因为 SDK 中包含了根文 件系统,而根文件系统中又包含了我们添加的各种各样的软件库,特别是链接库,我们在电 脑主机中开发编译 Ot 或 OpenCV 时就需要这些链接库。下面我们定制本章的根文件系统。

进入到第八章 Linux 显示设备的使用中创建的 petalinux 工程目录下,然后按照 6.3.2 小节 设置 Petalinux 环境变量。输入如下命令配置工程

petalinux-config

配置"Image Packaging Configuration ---> Root filesystem type"类型为"EXT4",将根文 件系统放到 SD 卡第二个分区,如下图所示:



图 10.2.1 修改根文件系统类型

保存设置并退出。

正点原子 领航者 ZYNQ 之嵌入式 Linux 开发指南 原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php 输入如下命令配置根文件系统: petalinux-config -c rootfs 执行结果如下图所示: Configuration Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty submenus ----). Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] built-in [] Filesystem Packages ---> Petalinux Package Groups ---> Image Features ---> apps ---> user packages PetaLinux RootFS Settings ---> <Select> < Exit > < Help > < Save > < Load >

图 10.2.2 根文件系统配置界面

下面定制本章需要的根文件系统。

10.2.1 添加 Qt

```
添加 Qt 需要对如下配置项进行配置:
Petalinux Package Groups --->
    packagegroup-petalinux-qt --->
        packagegroup-petalinux-qt (Y)
        populate_sdk_qt5 (Y)
结果如下图所示:
                              packagegroup-petalinux-qt
           Arrow keys navigate the menu. <Enter> selects submenus ---> (or
           empty submenus ----). Highlighted letters are hotkeys. Pressing
           <Pre><P> includes, <N> excludes, <M> modularizes features. Press
           <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*]
               [*] packagegroup-petalinux-qt
                  packagegroup-petalinux-qt-dbg
                  packagegroup-petalinux-qt-dev
               [*] populate_sdk_qt5
                <Select>
                           < Exit >
                                       < Help >
                                                   < Save >
                                                               < Load >
```

图 10.2.3 添加 qt



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

"packagegroup-petalinux-qt"中包含了 linux 系统使用 Qt 需要的基本包。移动到 "packagegroup-petalinux-qt", 然后用界面下方的 "< Help >" 查看其含义, 也可以按下键盘上 的H键,结果如下图所示:

<pre>packagegroup-pe CONFIG_packagegroup-petalinux-qt:</pre>	talinux-qt
<pre>Includes Qt supported packages qtbase qtbase-plugins qtbase-examples qtquick1 qtquick1-plugins qtquickcontrols-qmlplugins qtcharts</pre>	
Symbol: packagegroup-petalinux-qt [=y Type : boolean Prompt: packagegroup-petalinux-qt]
< <mark>E</mark> xit	> (01%) .

图 10.2.4 包含的 Qt 包

可以看到"packagegroup-petalinux-qt"中包含了 Qt 的基本包有哪些。 "populate_sdk_qt5" 表示包含 Qt5 主机工具链,同样可以用<Help>查看其含义。

其中的 dev 和 dbg 应该对应着开发和调试,由于没有具体说明,此处就不配置了。 对于 Qt,还有一个 Qt 扩展"qt-extended"的包组,名为 packagegroup-petalinux-qt-extended, 此处笔者没有添加该包组,有需要的读者可自行添加。

10.2.2 添加 OpenCV

添加 OpenCV 需要对如下配置项进行配置: Petalinux Package Groups ---> packagegroup-petalinux-opencv ---> packagegroup-petalinux-opencv (Y) 结果如下图所示:



图 10.2.5 添加 OpenCV

10.2.3 添加 Python

当今社会,随着深度学习及 AI 的兴起,使用 Python 的人越来越多,在根文件系统中必然 不可缺少对 Python 的支持,添加 Python 需要对如下配置项进行配置:

Petalinux Package Groups --->

packagegroup-petalinux-python-modules --->

packagegroup-petalinux-python-modules (Y)

结果如下图所示:



图 10.2.6 添加 Python

10.2.4 添加视频工具库 v4lutils (可选)

如果想使用开发板对视频进行采集开发等,可以添加视频工具库 v4lutils。添加视频工具 库 v4lutils 需要进行如下配置:

Petalinux Package Groups --->

领航者 ZYNQ 之嵌入式 Linux 开发指南 ②正点原子
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
packagegroup-petalinux-v4lutils>
packagegroup-petalinux-v4lutils (Y)
结果如下图所示:
<pre>packagegroup-petalinux-v4lutils Arrow keys navigate the menu. <enter> selects submenus> (or empty submenus). Highlighted letters are hotkeys. Pressing <y> includes, <n> excludes, <m> modularizes features. Press <esc><esc> to exit, <?> for Help, for Search. Legend: [*] </esc></esc></m></n></y></enter></pre> <pre>[*] packagegroup-petalinux-v4lutils [] packagegroup-petalinux-v4lutils-dbg [] packagegroup-petalinux-v4lutils-dev </pre>
<pre><select> < Exit > < Help > < Save > < Load ></select></pre>

图 10.2.7 添加视频工具库

10.2.5 添加 gcc 编译工具链(可选)

要想在开发板上直接编译 C/C++源码,可以添加 gcc 编译工具链,也就是 packagegroup-petalinux-self-hosted, self-hosted 直译就是自主机的,也就是把开发板当作主机来开发使用。添加 packagegroup-petalinux-self-hosted 需要进行如下配置:

Petalinux Package Groups --->

```
packagegroup-petalinux-self-hosted --->
```

```
packagegroup-petalinux-self-hosted (Y)
```

结果如下图所示:



图 10.2.8 添加编译工具链



原 十 廿 田 在	线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
用< H	Help >可以看到 packagegroup-petalinux-self-hosted 包含如下包:
CO	<pre>packagegroup-petalinux-self-hosted NFIG_packagegroup-petalinux-self-hosted:</pre>
In	cludes self hosted tools packages packagegroup-self-hosted whetstone vim
Syı Tyj Pro	<pre>mbol: packagegroup-petalinux-self-hosted [=n] pe : boolean ompt: packagegroup-petalinux-self-hosted Location: -> Petalinux Package Groups -> packagegroup-petalinux-self-hosted Defined at /home/zy/petalinux/Navigator_7020_v3/ALIENTEK-ZYNQ-driver/ (100%)</pre>
	< <mark>E</mark> xit >

图 10.2.9

packagegroup-self-hosted 中包含着 gcc 编译工具链;对于 whetstone,笔者查到的信息好像 是一种性能测试工具;vim,相信不需要笔者进行介绍了,linux 中一种流行的文本编辑器。 注意该包是可选的,添加会增大根文件系统的大小,需要的话可以添加。

10.2.6 添加网络工具(可选)

要想丰富根文件系统的网络相关方面的功能,如访问 mdio 寄存器等,可以添加网络栈包 networking-stack,需要进行的配置如下:

Petalinux Package Groups --->

packagegroup-petalinux-networking-stack --->

packagegroup-petalinux-networking-stack (Y)

结果如下图所示:



图 10.2.10 添加网络工具包

配置完成后,保存配置并退出。

10.3 编译根文件系统并构建 SDK

使用如下命令编译刚才定制的根文件系统:

petalinux-build -c rootfs

结果如下图所示:



图 10.3.1 编译根文件系统

如果编译过程中报错,参考错误!未找到引用源。小节扩展阅读部分。Petalinux 工程编译 生成的根文件系统在 images/linux/目录下。

接下来构建 SDK, 输入如下命令:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

petalinux-build --sdk

该命令可构建 SDK 并将其部署在当前 Petalinux 工程的 images/linux/sdk.sh 中。编译过程 中可能会遇到如下图所示的错误:



图 10.3.2 Network access disable 错误

错误的原因是缺少 qt3d 包,而工程又不能联网下载。解决方法是,运行"petalinux-config"命令,进入"Yocto Settings"选项界面,取消使能"BB_NO_NETWORK",如下图 所示:

Vactor Costerner
Arrow keys navigate the menu. <enter> selects submenus> (or empty submenus). Highlighted letters are hotkeys. Pressing <y> includes, <n> excludes, <m> modularizes features. Press <esc><esc> to exit, <? > for Help, for Search. Legend: [*] built-in []</esc></esc></m></n></y></enter>
<pre>(zynq-generic) YOCTO_MACHINE_NAME TMPDIR Location> Devtool Workspace Location> Parallel thread execution> Add pre-mirror url> Local sstate feeds settings> [] Enable Network sstate feeds [] Enable BB NO NETWORK User Layers></pre>
<pre><select> < Exit > < Help > < Save > < Load ></select></pre>

图 10.3.3 取消使能 BB_NO_NETWORK

保存配置并退出,继续构建 SDK。 成功构建的 SDK 文件在工程的 images/linux 目录下,名为 sdk.sh,如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

<pre>sqd@sqd-virtual-machine:~/petalinux/ALIENTEK-ZYNQ-cross-compile/images/linux\$ ls -l</pre>								
总用量 2593	3188	}						
- rw- r r	1	sqd	sqd	2967848	5月	21	12:25	BOOT.BIN
- rw- r r	1	sqd	sqd	2010	5月	8	07:40	boot.scr
- rw- r r	1	sqd	sqd	11568880	5月	8	08:19	image.ub
drwxr-xr-x	2	sqd	sqd	4096	5月	8	07:40	pxelinux.cfg
drwxr-xr-x	17	root	root	4096	5月	17	10:32	rootfs_20230519
- rw- r r	1	sqd	sqd	252153344	5月	22	10:46	rootfs.cpio
- rw- r r	1	sqd	sqd	93776375	5月	22	10:46	rootfs.cpio.gz
- rw- r r	1	sqd	sqd	93776439	5月	22	10:46	rootfs.cpio.gz.u-boot
- rw- r r	1	sqd	sqd	122159104	5月	22	10:46	rootfs.jffs2
- rw- r r	1	sqd	sqd	20926	5月	22	10:45	rootfs.manifest
- rw- r r	1	sqd	sqd	94134678	5月	22	10:46	rootfs.tar.gz
-rwxr-xr-x	1	sqd	sqd	1959246680	5月	22	11:13	sdk.sh 👞
- rw- r r	1	sqd	sqd	2083850	5月	5	06:50	system.bit
- rw- r r	1	sqd	sqd	31917	5月	8	08:18	system.dtb
- rw- r r	1	sqd	sqd	749169	5月	8	08:18	u-boot.bin
- rw- r r	1	sqd	sqd	815132	5月	8	08:18	u-boot.elf
- rw- r r	1	sqd	sqd	4326048	5月	8	08:19	uImage
- rw-rr	1	sqd	sqd	12671032	5月	8	08:19	vmlinux
- rw- r r	1	sqd	sqd	4325984	5月	8	08:19	zImage
- rw- r r	1	sqd	sqd	551916	5月	8	08:18	zynq fsbl.elf

图 10.3.4 SDK 文件

注意,由于 Petalinux 本身 bug 的原因会出现一些 ERROR,有些 ERROR 不好解决,另外 构建 SDK 太耗硬盘空间,推荐使用我们提供的已经构建好的 SDK 文件,路径为资料盘 B 盘: SDK\202002\ sdk.sh。

从图 10.3.4 中可以看到,构建的 SDK 文件是一个脚本文件,需要执行才能使用,也就是 安装 sdk。命令如下:

cd images/linux/

./sdk.sh

首先进入到当前 Petalinux 工程的 images/linux 目录下,然后执行 sdk.sh 进行安装,结果如下图所示:

<pre>zy@zy-virtual-machine:~/Navigator_7010_v3/ALIENTEK-ZYNQ-disp-dev\$ cd images/ linux/</pre>
<pre>zy@zy-virtual-machine:~/Navigator_7010_v3/ALIENTEK-ZYNQ-disp-dev/images/linu m\$/sdk.sh</pre>
PetaLinux SDK installer version 2020.2
Enter target directory for SDK (default: /opt/petalinux/2020.2):

图 10.3.5 安装 SDK

如果使用的是网盘提供的 sdk.sh 文件,需要先将其拷贝到 Ubuntu 系统中,然后执行 sdk.sh 进行安装。

安装路径默认在/opt/petalinux/2020.2 目录下,如果想安装在其他目录下,可以输入相应的路径,此处笔者保持默认,按回车键继续,接下来提示确认路径,按回车键确认,结果如下图所示:



原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php zy@zy-virtual-machine:~/Navigator 7010 v3/ALIENTEK-ZYNQ-disp-dev\$ cd images/ linux/ zy@zy-virtual-machine:~/Navigator 7010 v3/ALIENTEK-ZYNQ-disp-dev/images/linu m\$./sdk.sh PetaLinux SDK installer version 2020.2 Enter target directory for SDK (default: /opt/petalinux/2020.2): You are about to install the SDK to "/opt/petalinux/2020.2". Proceed [Y/n]? Extracting SDK.....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. \$. /opt/petalinux/2020.2/environment-setup-cortexa9t2hf-neon-xilinx-linuxgnueabi

图 10.3.6 SDK 安装路径设置

安 装 完 成 后 提 示 在 一 个 新 终 端 中 使 用 SDK 前 需 要 先 执 行 ". /opt/petalinux/2020.2/environment-setup-cortexa9t2hf-neon-xilinx-linux-gnueabi"以设置相应的环 境变量,其中的"."和 source 具有相同含义。

10.4 SDK 中的交叉编译工具链

安装完成后,我们进入到安装目录/opt/petalinux/2020下,该目录下有一些文件和文件夹, 如下图所示:

zy@zy-virtu 总用量 40	ıal-ma	achi	ine:/o	ot/pe	etali	inux/20	020.2\$ ls -l
- rw- r r	1 zy	zy	4407	5月	27	22:23	<pre>environment-setup-cortexa9t2hf-neon-x</pre>
ilinx-linu>	k-gnue	eab	i				
- rw- r r	1 zy	zy	21921	5月	27	22:23	<pre>site-config-cortexa9t2hf-neon-xilinx-</pre>
linux-gnuea	abi						
drwxr-xr-x	4 zy	zy	4096	5月	27	22:21	sysroots
- rw- r r	1 zy	zy	96	5月	27	22:23	version-cortexa9t2hf-neon-xilinx-linu
x-gnueabi	_						

图 10.4.1 SDK 安装目录

该目录下有一个环境变量配置文件 environment-setup-cortexa9t2hf-neon-xilinx-linuxgnueabi每次使用之前都需要使用 source 来运行这个配置文件,例如:

source environment-setup-cortexa9t2hf-neon-xilinx-linux-gnueabi

当 SDK 环境变量配置完成之后,我们可以在终端输入 arm,然后按两次 TAB 键查看交叉 编译相关的命令,如果配置成功则会显示如下信息,表示环境变量已经配置成功了:



原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php

zy@zy-virtual-machine:/opt/petalinux/2020.2\$ source environment-setup-cortexa9t2hf-neon-xilin
x-linux-gnueabi
zy@zy-virtual-machine:/opt/petalinux/2020.2\$ arm
arm2hpdl arm-xilinx-linux-musl-addr2line

arm-xilinx-linux-gnueabi-addr2line	arm-xilinx-linux-musl-ar
arm-xilinx-linux-gnueabi-ar	arm-xilinx-linux-musl-as
arm-xilinx-linux-gnueabi-as	arm-xilinx-linux-musl-c++filt
arm-xilinx-linux-gnueabi-c++filt	arm-xilinx-linux-musl-cpp
arm-xilinx-linux-gnueabi-cpp	arm-xilinx-linux-musl-dwp
arm-xilinx-linux-gnueabi-dwp	arm-xilinx-linux-musl-elfedit
arm-xilinx-linux-gnueabi-elfedit	arm-xilinx-linux-musl-g++
arm-xilinx-linux-gnueabi-g++	arm-xilinx-linux-musl-gcc
arm-xilinx-linux-gnueabi-gcc	arm-xilinx-linux-musl-gcc-ar
arm-xilinx-linux-gnueabi-gcc-ar	arm-xilinx-linux-musl-gcc-nm
arm-xilinx-linux-gnueabi-gcc-nm	arm-xilinx-linux-musl-gcc-ranlib
arm-xilinx-linux-gnueabi-gcc-ranlib	arm-xilinx-linux-musl-gcov
arm-xilinx-linux-gnueabi-gcov	arm-xilinx-linux-musl-gcov-dump
arm-xilinx-linux-gnueabi-gcov-dump	arm-xilinx-linux-musl-gcov-tool
arm-xilinx-linux-gnueabi-gcov-tool	arm-xilinx-linux-musl-gdb
arm-xilinx-linux-gnueabi-gdb	arm-xilinx-linux-musl-gdb-add-index
arm-xilinx-linux-gnueabi-gdb-add-index	arm-xilinx-linux-musl-gprof
arm-xilinx-linux-gnueabi-gprof	arm-xilinx-linux-musl-ld
arm-xilinx-linux-gnueabi-ld	arm-xilinx-linux-musl-ld.bfd
arm-xilinx-linux-gnueabi-ld.bfd	arm-xilinx-linux-musl-ld.gold

图 10.4.2 查看交叉编译工具链

输入如下命令查看交叉编译工具所在位置,例如:

whereis arm-xilinx-linux-gnueabi-gcc

结果如下图所示:

zy@zy-virtual-machine:/opt/petalinux/2020.2\$ whereis arm-xilinx-linux-gnueabi-gcc arm-xilinx-linux-gnueabi-gcc: /opt/petalinux/2020.2/sysroots/x86_64-petalinux-linux/usr/bin/a rm-xilinx-linux-gnueabi/arm-xilinx-linux-gnueabi-gcc zy@zy-virtual-machine:/opt/petalinux/2020.2\$

图 10.4.3 查看交叉编译工具位置

此时,我们就得到了 linux 交叉编译工具链,可以摆脱 Petalinux,直接使用 linux 交叉编译工具链编译 uboot、linux 内核和 linux 应用。

注意,如果不想每次都手动设置 SDK 环境变量,可以在终端输入如下命令:

echo '. /opt/petalinux/2020.2/environment-setup-cortexa9t2hf-neon-xilinx-linux-gnueabi' | tee -a ~/.bashrc

输出结果: . /opt/petalinux/2020.2/ environment-setup-cortexa9t2hf-neon-xilinx-linux-gnueabi, 这样设置以后,我们就可以直接使用 arm 的交叉编译器,无需再设置 SDK 的工作环境。

10.5 SDK 中的 Qt 和 OpenCV 库

sysroots 目录下有两个文件夹,其中 cortexa9t2hf-neon-xilinx-linux-gnueabi文件下是我们构建的用于 arm 开发板的根文件系统。在 cortexa9t2hf-neon-xilinx-linux-gnueabi/usr/lib 目录下有很多库文件,其中就包含了 Qt 和 OpenCV 相关的链接库文件。

输入"ls libQt*"命令,查看 Qt 相关链接库文件,如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

zy@zy-virtual-machine:/opt/petalin	ux/2020.2/sysroots\$ ls
cortexa9t2hf-neon-xilinx-linux-gnu	eabi x86_64-petalinux-linux
zy@zy-virtual-machine:/opt/petalin	<pre>ux/2020.2/sysroots\$ cd cortexa9t2hf-neon-xilinx-linux-gnuea</pre>
bi/usr/lib	
zy@zy-virtual-machine:/opt/petalin	ux/2020.2/sysroots/cortexa9t2hf-neon-xilinx-linux-gnueabi/u
sr/lib\$ ls libQt* 🛶	
libQt53DAnimation.prl	libQt5PacketProtocol.a
libQt53DAnimation.so	libQt5PacketProtocol.prl
libQt53DAnimation.so.5	libQt5PlatformCompositorSupport.a
libQt53DAnimation.so.5.13	libQt5PlatformCompositorSupport.prl
libQt53DAnimation.so.5.13.2	libQt5Positioning.prl
libQt53DCore.prl	libQt5PositioningQuick.prl
libQt53DCore.so	libQt5PositioningQuick.so
libQt53DCore.so.5	libQt5PositioningQuick.so.5
libQt53DCore.so.5.13	libQt5PositioningQuick.so.5.13
libQt53DCore.so.5.13.2	libQt5PositioningQuick.so.5.13.2
libQt53DExtras.prl	libQt5Positioning.so
libQt53DExtras.so	libQt5Positioning.so.5
libQt53DExtras.so.5	lib0t5Positioning.so.5.13
libQt53DExtras.so.5.13	libQt5Positioning.so.5.13.2
libQt53DExtras.so.5.13.2	libQt5PrintSupport.prl
libQt53DInput.prl	libQt5PrintSupport.so
libQt53DInput.so	libQt5PrintSupport.so.5
libQt53DInput.so.5	libQt5PrintSupport.so.5.13
libQt53DInput.so.5.13	libQt5PrintSupport.so.5.13.2
libQt53DInput.so.5.13.2	libQt5PublishSubscribe.prl
libQt53DLogic.prl	libQt5PublishSubscribe.so

图 10.5.1 Qt 链接库文件

输入"ls libopencv*"命令,查看 OpenCV 相关链接库文件,如下图所示:

zy@zy-virtual-machine:/opt/petalinu	x/2020.2/sysroots/cortexa9t2hf-neon-xilinx-linux-gnueabi/u
<pre>sr/lib\$ ls libopencv* </pre>	
libopencv_aruco.so	libopencv_objdetect.so.3.4.3
libopencv_aruco.so.3.4	libopencv_optflow.so
libopencv_aruco.so.3.4.3	libopencv_optflow.so.3.4
libopencv_bgsegm.so	libopencv_optflow.so.3.4.3
libopencv_bgsegm.so.3.4	libopencv_phase_unwrapping.so
libopencv_bgsegm.so.3.4.3	libopencv_phase_unwrapping.so.3.4
libopencv_bioinspired.so	libopencv_phase_unwrapping.so.3.4.3
libopencv_bioinspired.so.3.4	libopencv_photo.so
libopencv_bioinspired.so.3.4.3	libopencv_photo.so.3.4
libopencv_calib3d.so	libopencv_photo.so.3.4.3
libopencv_calib3d.so.3.4	libopencv_plot.so
libopencv_calib3d.so.3.4.3	libopencv_plot.so.3.4
libopencv_ccalib.so	libopencv_plot.so.3.4.3
libopencv_ccalib.so.3.4	libopencv_reg.so
libopencv_ccalib.so.3.4.3	libopencv_reg.so.3.4
libopencv_core.so	libopencv_reg.so.3.4.3
libopencv_core.so.3.4	libopencv_rgbd.so
libopencv_core.so.3.4.3	libopencv_rgbd.so.3.4
libopencv_datasets.so	libopencv_rgbd.so.3.4.3
libopencv_datasets.so.3.4	libopencv_saliency.so
libopencv_datasets.so.3.4.3	libopencv_saliency.so.3.4
libopencv_dnn_objdetect.so	libopencv_saliency.so.3.4.3
libopencv_dnn_objdetect.so.3.4	libopencv_shape.so
libopencv_dnn_objdetect.so.3.4.3	libopencv_shape.so.3.4

图 10.5.2 opencv 链接库文件



原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php

10.6 交叉编译工具链的使用

输入如下命令设置 SDK 工作环境:

source /opt/petalinux/2020.2/environment-setup-cortexa9t2hf-neon-xilinx-linux-gnueabi

设置完 SDK 的工作环境后,就可以获得 linux 交叉编译工具链。不过使用的时候并不是 直接使用交叉编译工具链,而是使用\${CC}和\${LD}两个环境变量(可以简写成\$CC 和 \$LD),如下图所示:

zy@zy-virtual-machine:~\$ source /opt/petalinux/2020.2/environment-setup-cortexa9
t2hf-neon-xilinx-linux-gnueabi

zy@zy-virtual-machine:~\$ echo \$CC

arm-xilinx-linux-gnueabi-gcc -mthumb -mfpu=neon -mfloat-abi=hard -mcpu=cortex-a9
 -sysroot=/opt/petalinux/2020.2/sysroots/cortexa9t2hf-neon-xilinx-linux-gnueabi
zy@zy-virtual-machine:~\$ echo \$CXX

arm-xilinx-linux-gnueabi-g++ -mthumb -mfpu=neon -mfloat-abi=hard -mcpu=cortex-a9
 --sysroot=/opt/petalinux/2020.2/sysroots/cortexa9t2hf-neon-xilinx-linux-gnueabi
zy@zy-virtual-machine:~\$

图 10.6.1 CC 和 CXX 环境变量定义

可以看到\$CC 环境变量是带参数的 arm-xilinx-linux-gnueabi-gcc 交叉编译工具链的定义, 特别是其中的 sysroot 参数,没有该参数,直接使用 arm-xilinx-linux-gnueabi-gcc 编译 C 源文件 会报错。这就是为何不能直接使用 arm-xilinx-linux-gnueabi-gcc,而是需要使用\$CC 的原因, 所以后面需要编译用于开发板上的 C 程序源码文件时,应使用如下方式编译:

\$CC c 源码文件

#或者

\$CXX c++源码文件

\$CXX 环境变量是带参数的 arm-xilinx-linux-gnueabi-g++交叉编译工具链的定义,可直接用于编译 C++程序源码文件。

调试工具为 arm-xilinx-linux-gnueabi-gdb,对应的环境变量为\$GDB。其它工具就不一一介绍了。注意,每打开一个新的终端,需要重新设置 SDK 工作环境。当前终端中的设置只在当前终端有效。

10.7 配置 QT Creator

要想用 QT Creator 开发用于开发板的 Qt demo 需要配置 QT Creator。在 Ubuntu 系统中打 开 QT Creator (如果没有安装 QT Creator,请阅读本章扩展阅读部分)。



原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php



图 10.7.1 打开 Qt Creator

打开后的 Qt Creator 界面如下图所示:



图 10.7.2 Qt Creator 界面

现在开始配置 QT Creator。如图 10.7.3 所示,点击 QT Creator 菜单栏的"工具(Tools)"菜单,在弹出的菜单中点击"选项(Options)..."以打开选项配置界面。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php



图 10.7.3 打开配置选项

打开的选项配置界面如下图所示:

	选项 — Qt Creator	8
Filter	Kits	
🖬 Kits	构建套件(Kit) Qt Versions 编译器 Debuggers	Qbs CMake
🖵 环境	名称	添加
■ 文本编辑器		克隆
🌠 FakeVim	■ Desktop Qt 5.14.2 GCC 64bit (款认) 手动设置	副陸
❷ 帮助		
{} C++		设置为默认
🖈 Qt Quick		Settings Filter
▶ 构建和运行		Default Settings Filter
ᆥ 调试器		
✔ 设计师		
Python		
	Apply	X Cancel ✓ OK

图 10.7.4 选项配置界面

1、配置编译器(Compilers)

配置编译器也就是配置 linux 交叉编译工具 arm-xilinx-linux-gnueabi-gcc 和 arm-xilinxlinux-gnueabi-g++.

点击左侧分类栏中"Kits",然后在右侧点击"编译器"选项,如下图所示:



正点原子

图 10.7.5 编译器配置

从图 10.7.5 中红框 3 处可以看出,Qt Creator 自动检测到 C 和 C++编译器,这些编译器在 Ubuntu 系统的/usr/bin 目录中,且用于 64 位的 x86 架构的 CPU,用这些编译工具编译出来的 代码只能在 Ubuntu 虚拟机上运行。所以还需要在红框 4 处手动添加用于编译 32 位 arm 处理 器程序的编译器。

点击红框 5 处的"添加"按钮,在下拉选项中选择"GCC",然后再选择 GCC 子项中 "C",如下图所示:



原子哥在线教学:www.yuanzige.com 论

论坛:www.openedv.com/forum.php

	选项 — Qt Creator 🛛 😣
Filter	Kits
TH Kits	构建套件(Kit) Qt Versions 编译器 Debuggers Qbs CMake
 □ 环境 ■ 文本编辑器 ■ 文本编辑器 ■ FakeVim ④ 帮助 () C++ √ Qt Quick ✓ 构建和运行 爺 调试器 ✓ 设计师 Python 	Name 添加 * Auto-detected ICC * C GCC (C, x86 64bit in /usr/bin) Clang (C, x86 64bit in /opt/Qt5.14.2/Tools ICC * C++ GCC (C++, x86 64bit in /usr/bin) * Manual C C C++ C++ GCC (C++, x86 64bit in /usr/bin) * Manual C C++ Auto-detection Settings
	✓ Apply X Cancel ✓ OK

图 10.7.6 手动添加 GCC 编译工具

点击之后,在出现的属性栏中,修改编译器名称为 zynq_GCC,然后点击浏览,设置编译器路径为 "/opt/petalinux/2020.2/sysroots/x86_64-petalinux-linux/usr/bin/ arm-xilinx-linux-gnueabi / arm-xilinx-linux-gnueabi",如下图所示::

	选项一(Qt Creator	•
Filter	Kits		
🖬 Kits	构建套件(Kit) Qt Versions 编译器 Debuggers Qbs CMake		
🖵 环境	Name	Туре	▲ 添加 →
■ 文本编辑器	 C++ GCC (C++, x86 64bit in /usr/bin) ✓ Manual 	GCC	克隆
K. FakeVim	* C	666	删除
帮助	C++	GCC	Remove All
{} C++			
🛃 Qt Quick			Re-detect
▶ 构建和运行	名称: zynq_GCC		Auto-detection Settings
✤ 调试器	编译器路径(C): 0.2/sysroots/x86_64-petalinux-linux/usr/bin	/arm-xilinx-linux-gnueabi/arm-xilinx-linux-gnueabi-gcc 浏览	
✔ 设计师	Platform codegen flags:		
Python	Platform linker flaos:		•
			✓ Apply ¥ Cancel ✓ OK

图 10.7.7 配置 gcc 编译工具

同样的方法添加 G++编译器。再次点击"添加",在下拉选项中选择"GCC",然后选择 GCC 子项中"C++",如下图所示:



原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php

Kits											
构建套件(Kit)	Qt Versions	编译器	Debuggers	Qbs	CMake						
构建套件(Kit) Name ▼ Auto-detec ▼ C Clang ▼ C++ GCC (▼ Manual ▼ C ← G C++	Qt Versions ted C, x86 64bit in / I(C, x86 64bit in C++, x86 64bit i	/ 编详故 /usr/bin) I /opt/Qt5. n /usr/bin)	14.2/Tools/Qt	QDS	(libexec/qt	creator/clar	ت ng/bin) C G	ype cc lang .cc		添加 ICC MinGW GCC Clang Custom QCC Auto-detectio	a v
									🖌 Ap	ply X <u>C</u> ance	еl <u>∢ о</u> к

图 10.7.8 手动添加 G++编译工具

点击之后,在出现的属性栏中,修改编译器名称为 zynq_G++,然后点击浏览,设置编译 器路径为 "/opt/petalinux/2020.2/sysroots/x86_64-petalinux-linux/usr/bin/ arm-xilinx-linux-gnueabi /arm-xilinx-linux-gnueabi-g++",如下图所示:

选项 — Qt Crea	tor	
Kits		
构建套件(Kit) Qt Versions 编译器 Debuggers Qbs CMake		
Name	Туре	▲ 添加 ▼
GCC (C++, x86 64bit in /usr/bin)	GCC	10100
✓ Manual ✓ C		克隆
• GCC • C++	GCC	删除
0 GCC	GCC	Remove All
		Re-detect
		Auto-detection Settings
名称: zynq_G++		
编译器路径(C): 0.2/sysroots/x86_64-petalinux-linux/usr/bin/arm-xil	linx-linux-gnueabi/arm-xilinx-linux-gnueabi-g++ 浏览	
Platform codegen flags:		
Platform linker flags:		•
	۲ ۲ 	
	✓	Apply 🗱 <u>C</u> ancel 🖌 <u>O</u> K

图 10.7.9 配置 g++编译工具

编译器设置好了。

2、配置 Debuggers

配置 Debuggers,也就是配置调试工具 arm-xilinx-linux-gnueabi-gdb。

将配置选项切换到"Debuggers"标签页,然后点击"Add",弹出配置列表后,将调试 工具名称修改为"zynq_Debugger",然后点击"浏览",添加调试工具路径,为

"/opt/petalinux/2020.2/sysroots/x86_64-petalinux-linux/usr/bin/arm-xilinx-linux-gnueabi/armxilinx-linux-gnueabi-gdb",如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

	选项 — Qt Creator	×
Kits		
构建套件(Kit) Qt	Versions 编译器 Debuggers Qbs CMake	2
Name	Location	Add
 Auto-detected System GDB a 	ut /usr/bin/qdb /usr/bin/qdb	Clone
4		Remove
Name: 3	zynq_Debugger	
Path: 5	//opt/petalinux/2020.2/sysroots/x86_64-petalinux-linux/usr/bin/arm-xilinx-linux-gnueabi/arm-xilinx-linux-gnueabi-gdb 浏览	
Туре:	GDB	
ABIs:	arm-linux-generic-elf-32bit	
Version:	8.3.1	
Working directory	6 浏览	-
		cel 🥑 OK

图 10.7.10 配置 Debuggers

3、配置 Qt Versions

切换到"Qt Versions"标签页,如下图所示:	
选项 — Qt Creator	8
Kits 构建套件(Kit) Qt Versions 编译器 Debuggers Qbs CMake	
Name • qmake Location	添加
 手动设置 ▼ 自动检测 	删除
Qt 5.14.2 GCC 64bit /opt/Qt5.14.2/5.14.2/gcc_64/bin/qmake	Clean Up
	Clean op
✓ Apply X Cance	l

图 10.7.11 Qt Versions 配置页面

该配置页面用于配置 qmake 工具的路径,这里简单地介绍一下 Qt 程序的编译过程,大致 可以分为以下三个步骤:

1. 使用 qmake-project 生成与平台无关的 pro 文件;

2. 利用 pro 文件生成与平台相关的 Makefile 文件。Makefile 文件关系到整个工程的编译规 则,一个工程中的源文件不计数,其按类型、功能、模块分别放在若干个目录中,Makefile 定义了一系列的规则来指定,哪些文件需要先编译,哪些文件需要后编译,哪些文件需要重



原子哥在线教学:www.yuanzige.com 论坛:www.openedv.com/forum.php 新编译,文件之间的依赖关系,甚至于进行更复杂的功能。而在 Qt Creator 中,使用 qmake 工 具自动生成 Makefile 文件。

3. 使用 make 命令完成自动编译, make 就是通过解析 Makefile 文件的内容来执行编译工作的, 会为每个源文件生成一个对应的.o 文件, 最后将目标文件链接生成最终的可执行文件。

点击右边"添加(Add)..."按钮添加一个 qmake 工具,工具路径为

"/opt/petalinux/2020.2/sysroots/x86-64-petalinux-linux/usr/bin/qmake"。"版本名称"栏的 "Qt%{Qt:Version}(系统)"可以改为其他名字,也可以不改,此处笔者改为

"Qt%{Qt:Version}(zynq)", 配置完成后点击 "Apply", 如下图所示:

		选项 — Q	t Creator					8
Kits								
构建套件(Kit)	Qt Versions	编译器	Debuggers	Qbs	CMake		\ \	
Name	▼ qm	ake Locatio	n					添加
▼ 手动设置								
Qt 5.13.	2 (zynq) /op	ot/petalinu	ıx/2020.2/sys	roots/	.4-petali	nux-linux/usr	/bin/qmake	删除
▼ 自动检测								
Qt 5.14.	2 GCC 64bit /op	t/Qt5.14.2	/5.14.2/gcc_64	4 /bin/q n	nake			Clean Ur
L D								
版本名称:	Qt %{Qt:Version	n} (zyną)					<mark></mark> дв	
amaka Pt /Z	/opt/potalipux/	2020 2/646	COOLE /296 64	potalipu	v linuv/u	cr/bip/amako	SHUK	
qmake 哈哈?	opt/petaunux/	2020.2/595	0015/880_04-	petatinu	x-unux/u	siybiiiyqiilake	测免	
							•	
桌面的Qt 版本	5.13.2						详情 ▼	
4						\		
								0
						🗸 Apply	₩ <u>C</u> ancel	<u> </u>

图 10.7.12 配置 qmake

4、配置 Kits

点击切换到"构建套件(Kit)"标签页,如下图所示:

Kits							
构建套	\$件(Kit)	Qt Versions	编译器	Debuggers	Qbs	CMake	
名称		-					添加
▼ 自	动检测 및 Deski	ton Ot 5 14 2 Ci	°C 64hit (₹	+;{)			克隆
手	动设置	ιορ Qt 3. 14.2 Ο	ις σ <i>ησι</i> ς (#/				删除
							设置为默认
							Settings Filter.
							Default Settings Fi

🔁 正点原子

图 10.7.13 构建套件

Kit 其实就是对前面的各个配置项进行打包,变成一个 Kit 套件,而当我们使用 Qt Creator 创建 Qt 工程的时候会选择一个 Kit 套件,并基于它创建我们的工程。上图中的"自动 检测"列表下已经存在一个配置好的 Kit 套件 Desktop Qt 5.14.2 GCC 64bit,从名字可以知 道,该 Kit 适用于创建在 Ubuntu 桌面系统上运行的 Qt 程序,所以不能用于创建一个能够在 ZYNQ 开发板上运行的 Qt 程序,所以这里需要添加一个新的 Kit。

点击图 10.7.13 中"添加",出现下图所示的配置列表:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

		选项 — 🤇	ot Creator						
lits									
勾建套件(Kit)	Qt Versions	编译器	Debuggers	Qbs	CMake				
名称								家加] [
▼ 自动检测	top Qt 5.14.2 GC	.C 64bit (黒	til)				Ę	ē隆	
手动设置							H	削除	
🔺 zynq							设置	为默认	
							Setting	ıs Filter]
							Default Se	tings Filter]
名称:	1	zynq						₽.	
File system na	ame:								
Device type:		2 Gener	ic Linux Devic	e			-		
Device:							*	Manage	
Sysroot:		3 ;/2020	.2/sysroots/co	rtexa9t2	2hf-neon->	xilinx-lin	ux-gnueabi	浏览	
Compiler:		4 <u>c:</u> 5 c++: ;	zynq_GCC zynq_G++				*	Manage	
Environment:		No cha	inges to apply					Change	
Debugger:		6 zynq_	Debugger				¥	Manage	
Qt version:		7 Qt 5.1	Qt 5.13.2 (zynq)						
Qt mkspec:									
Additional Ob	- Deofile Cottie	<u></u>						Change	_

图 10.7.14 配置套件

在第一栏的名称栏为 Kit 套件命名,例如 zynq。

Device type: 选择设备的类型,这里有四个选择项,分别为 Desktop (PC 机)、Android Device (安卓设备)、Generic Linux Device (通用 Linux 设备)和 QNX Device (QNX 设备); 对于 ZYNQ7000 系列来说,选择 Generic Linux Device。

Sysroot: 系统镜像的根目录,为 SDK 安装路径下的 sysroots/ cortexa9t2hf-neon-xilinxlinux-gnueabi 文件夹。

Compiler C: 选择前面手动添加的 zynq_GCC。

Compiler C++: 选择前面手动添加的 zynq_G++。

Debugger 选择 zynq_Debugger,

Qt version 选择 Qt 5.13.2(zynq)

配置完成之后点击"Apply"应用。

点击"OK"按钮配置完成退出界面,至此我们的 Qt Creator 就配置完成了。下一节我们 将对前面所做的配置工作进行测试。


原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

10.8 搭建 Qt 测试工程

首先设置 SDK 环境变量,然后进入"/opt/Qt5.14.2/Tools/QtCreator/bin"目录,运行 "./qtcreator"命令启动 Qt Creator,如下图所示:

zy@zy-virtual-	m <mark>achine:~</mark> \$ source /o	pt/petalinux/2020.2/en\	/ironment-setup-cortexa9				
t2hf-neon-xili	nx-linux-gnueabi 🔶 🔶		- The second				
<pre>zy@zy-virtual-machine:~\$ cd /opt/Qt5.14.2/Tools/QtCreator/bin</pre>							
zy@zy-virtual-	<pre>machine:/opt/Qt5.14.</pre>	2/Tools/QtCreator/bin\$	ls				
qbs	qbs-create-project	qbs-setup-toolchains	qtcreator.sh				
qbs-config	qbs-setup-android	qt.conf					
qbs-config-ui	qbs-setup-qt	qtcreator					
zy@zy-virtual-machine:/opt/Qt5.14.2/Tools/QtCreator/bin\$./qtcreator +							



打开 Qt Creator 后,点击菜单栏"文件->新建文件或项目"(或者直接使用快捷键 Ctrl+ N)进入模板选择界面,如下所示:

	New File or Project — Qt Creator	8
选择一个模板:		所有模板
项目 Application Library 其他项目 Non-Qt Project Import Project 文件和类 C++ Modeling Qt GLSL General Java Python	Qt Widgets Application Qt Console Application Qt for Python - Empty Qt for Python - Window Qt Quick Application - Empty Qt Quick Application - Scroll Qt Quick Application - Stack Qt Quick Application - Swipe	Creates a Qt application for the desktop. Includes a Qt Designer- based main window. Preselects a desktop Qt for building the application if available. 支持的平台 : Generic Linux Device 桌面
		🗱 <u>C</u> ancel 🚽 Choose

图 10.8.2 创建工程

左边选择工程类型,这里我们选择的是一个 Application 应用程序,右边选择工程模板, 这里我们选择 Qt Widgets Application, 点击"Choose..."按钮进入下一步。 设置 qt 工程名为 "qt_test", 路径使用默认设置, 然后点击"下一步", 如下图所示:

	Qr widger	
Location	Project Location	
Build System Details Translation Kits Summary	This wizard generates a default from QApplicatio	Qt Widgets Application project. The application derives by on and includes an empty widget.
	名称: qt_test	
	名称: qt_test 创建路径: /home/zy	浏览
	名称: qt_test 创建路径: /home/zy	浏览
	名称: qt_test 创建路径: /home/zy 设为默认的项目路径	浏览

2 正点原子

图 10.8.3 设置 qt 工程名和路径

"Build system" 默认为 qmake,此处保持默认即可,点击"下一步",如下图所示:

	Qt Widge	ts Application — Qt Crea	tor	8
Location	Define Build Syst	em		
Build System Details Translation Kits Summary	Build system: qmake			¥
			< 上一步(B) 下一步(N) >	取消

图 10.8.4 选择构建工具

接下来的页面使用默认配置,直接点击"下一步",如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

		Qt Widgets Application — Qt Creator
Location	Class Inf	formation
Build System		Specify basic information about the classes for which you want to generate skeleton source code files.
Translation	Class name:	e: MainWindow
Kits Summary	Base class:	QMainWindow
	Header file:	: mainwindow.h
	Source file:	: mainwindow.cpp
		✓ Generate form
	Form file:	mainwindow.ui
		< 上一步(B) 下一步(N) > 取消
		图 10.8.5 配置工程的类
$T \pm H = T + H$		
下米的贝迪也	史用款认能_	L直,继续只击 下一步 ,如下图则亦:
Location	Translatio	ion File
Build System	If you plan to tool, please s	o provide translations for your project's user interface via the Qt Linguist select a language here. A corresponding translation (.ts) file will be
Details	generated fo	OF YOU.
Details Translation	generated fo Language:	<pre>or you. </pre>
Details Translation Kits Summary	generated fo Language: Translation fi	file: <none> .ts</none>
Details Translation Kits Summary	generated fo Language: Translation fi	file: <none> .ts</none>
Details Translation Kits Summary	generated fo Language: Translation fi	<pre>or you. </pre> <pre></pre> <pre>select the select test of the select test of the select test of t</pre>
Details Translation Kits Summary	generated fo Language: Translation fi	<pre>or you. </pre> <pre></pre> <pre>select the select text and te</pre>
Details Translation Kits Summary	generated fo Language: Translation fi	<pre>or you. </pre> <pre></pre> <pre>file: <none> </none></pre> <pre>.ts</pre>
Details Translation Kits Summary	generated fo Language: Translation fi	file: <none> .ts</none>
Details Translation Kits Summary	generated fo Language: Translation fi	<pre>or you. </pre> <pre> </pre> <pre> file: <<none> </none></pre> <pre> .ts </pre>
Details Translation Kits Summary	generated fo Language: Translation fi	<pre>or you. </pre> <pre> file: <none> .ts </none></pre>
Details Translation Kits Summary	generated fo Language: Translation fi	<pre>or you. </pre> <pre> file: <none> .ts </none></pre>
Details Translation Kits Summary	generated fo Language: Translation fi	<pre>or you. </pre> <pre> file: <none> </none></pre> .ts
Details Translation Kits Summary	generated fo Language: Translation fi	or you. <pre> file: <none> .ts </none></pre> <pre> <p< td=""></p<></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre>



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

前面介绍过,在创建 Qt 应用程序的时候需要选择 Kit,并基于该 Kit 创建我们的工程。由 于本小节是测试之前配置的 QT Creator 能否正常工作,所以这里选择"zynq"。点击"下一 步"按钮,如下图所示:

	Qt Widgets Application — Qt Creator	8
Location Build System Details Translation	Kit Selection The following kits can be used for project qt_test : Type to filter kits by name	
Kits Summary	■ Select all kits □ □ Desktop Qt 5.14.2 GCC 64bit ✓ □ zynq	详情 ▼ 详情 ▼
	<上一步(B) 下一步(N) >	取消

图 10.8.7 选择 Kit

在"Summary"页面点击"完成"按钮,完成工程的创建,如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

	Qt Widgets	Application — Qt Creato	or	8
Location	Project Manageme	ent		
Build System Details Translation Kits Summary	作为子项目添加到项目中: 添加到版本控制系统(<u>V</u>):	<none></none>	~	Configure
	要添加的文件 /home/zy/qt_test: main.cpp mainwindow.cpp mainwindow.h mainwindow.ui qt_test.pro			
			<上一步(B) 完成(F) 取消

图 10.8.8 Summary 页面

工程创建好之后,出现如下所示界面:



图 10.8.9 工程界面

双击图 10.8.9 左边文件列表 Forms 中的 mainwindow.ui 文件, 打开 Qt Creator 的图形化编 程界面,如下所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

X1+(E)	编相(E) 彻廷(E) 朐氏(E) Analyze		2) 市助(日)							
	🗈 📝 mainwindow.ui	+ × 🖬 🖫	🔻 🌉 III 🗏 M 🕱 8	III 🐝 🔣						
	Filter				•		Filter			
欢迎	 Layouts 	▲ 仕凶里制入					7+45		*	
_	Vertical Layout						∧jsk ▼ Mair	Window	OMai ndo	
	III Horizontal Layout						E E	centralwidg	et QWidget	1
1673	Grid Layout						n	nenubar	QMenuBar	
1	Form Layout						S	tatusbar	QStatusBar	
设计	* Spacers									
÷.	🚧 Horizontal Spacer									
Debug	📓 Vertical Spacer									
بر	 Buttons 									
項目	Push Button									
•	I Tool Button									Ŧ
U Min	Radio Button						4		· · ·	s
19-10	Scheck Box						- Filter	adaus Ottais		-
	📀 Command Link Button		. 		Filter	•	Mainw	ndow : Qmair	window	
	Dialog Button Box			***	TT/t 65	て日相二	属性		徂	
	 Item Views (Model-Based) 	古柳 1	使用 又平	状定键	可延的	上 具旋亦	 QOL 	ject	Maintalia	
	List View						ODJe	daot	Mainwindow	
	Tree View						wind	lowModality	NonModal	
qt_test	🔠 Table View						enat	oled	✓	
_ L ,	Column View						→ geo	metry	[(0, 0), 800	
Debug	Undo View						sizel	Policy	[Preferred,	
	 Item Widgets (Item-Based) 						mini	mumSize	0 x 0	
	List Widget						▶ max	imumSize	16777215 x	
	Tree Widget	 Action Editor 	Signals_Slots E				1	ncrement	0 × 0	Í
PHE	Would you like to take a quick UI too	ur? This tour highl	ights important user inte	erface elements	and shows how	they are used. To take	Take UI To	Jr Do Not S	how Again 🛛 🗙	<
	the tour later, select Help > UI Tour.									

图 10.8.10 图形化编程界面

这是 Qt Creator 中集成的 Qt Design 图形化编程工具, 左边栏显示了 Qt 所支持的控件, 中间显示 GUI 控件编程、布局界面,而右边界面显示了 Qt 控件所支持的各种属性,可以对 其进行修改。

图 10.8.10 中笔者随便拖入几个控件,包括一个日历控件 Calendar Widget、一个按钮控件 Push Button 以及一个 Label 控件,并对它们进行了简单地布局、调整、修改了 Label 控件的 显示的文字内容和大小(选中 Label 控件,可以在右下角列出的属性表中找到对应的设置 项),如下图所示:

在这里输入						•			
	<			五月	2023			>	
		周日	周一	周二	周三	周四	周五	周六	
	18	30	1	2	3	4	5	6	
	19	7	8	9	10	11	12	13	
	20	14	15	16	17	18	19	20	
	21	21	22	23	24	25	26	27	
	22	28	29	30	31	1	2	3	
	23	4	5	6	7	8	9	10	
			Duch	Dutter					
			Push	Button			Qt Test	t	
							.		

图 10.8.11 搭建显示界面

Qt 工程编程完成后使用 Ctrl+S 快捷键进行保存, 然后点击左下角的小锤子按钮编译工 程,如下所示:



原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php



图 10.8.12 编译工程

编译过程中可以点击下方的"编译输出(Compile Output)"按钮查看编译输出信息,如下 所示:

laiβia	^	⊡
D_REENTRANT -fPIC -DQT_DEPRECATED_WARNINGS -DQT_QML_DEBUG -DQT_WIDGETS_LIB -DQT_GUI_LIB -DQT_CORE_LIB -I/qt_test -II/opt/petalinux/2020.2/sysroots/ cortexa9t2hf-neon-xilinx-linux-gnueabi/usr/include/QtWidgets -I/opt/petalinux/2020.2/sysroots/cortexa9t2hf-neon-xilinx-linux-gnueabi/usr/include/QtGui -I/opt/	,	-
petalinux/2020.2/sysroots/cortexa9t2hf-neon-xilinx-linux-gnueabi/usr/include/QtCore -III/opt/petalinux/2020.2/sysroots/cortexa9t2hf-neon-xilinx-linux-gnueabi/usr/lib/mkspecs/linux-oe_gt+ -o moc_mainwindow.com		
arm=xilinx-linux=gnueabi-g+t -=mthumb -=mfpu=neon -=mfloat=abi=and ==now=rotex-a9sysroot=/opt/petalinux/2020.2/sysroots/cortexa9t2hf-neon-xilinx-linux-gnueabi-g+t -=mthumb -=mfpu=neon -=mfloat=abi=and ==now=rotex-a9sysroot=/opt/petalinux/2020.2/sysroots/cortexa9t2hf-neon-xilinx-linux-gnueabi-g+t -=mthumb -=mfpu=neon -=mfloat=abi=and ==now=rotex-a9sysroot=/opt/petalinux/2020.2/sysroots/cortexa9t2hf-neon-xilinx-linux-gnueabi-g+t -=mthumb -=mfloat=abi=and ==now=rotex-a9sysroot=/opt/petalinux/2020.2/sysroots/cortexa9t2hf-neon-xilinx-linux-gnueabi/usr/lib/libQt5Widgets.so /opt/petalinux/2020.2/sysroots/cortexa9t2hf-neon-xilinx-linux-gnueabi/usr/lib/libQt5Widgets.so /opt/petalinux/2020.2/sysroots/cortexa9t2hf-neon-xilinx-linux-gnueabi/usr/lib/libQt5Core.so /opt/petalinux/2020.2/sysroots/cortexa9t2hf-neon-xilinx-linux-gnueabi/usr/lib/libQt5Core.so /opt/petalinux/2020.2/sysroots/cortexa9t2hf-neon-xilinx-linux-gnueabi/usr/lib/libQt5Core.so /opt/petalinux/2020.2/sysroots/cortexa9t2hf-neon-xilinx-linux-gnueabi/usr/lib/libQt5Core.so /opt/petalinux/2020.2/sysroots/cortexa9t2hf-neon-xilinx-linux-gnueabi/usr/lib/libQt5Core.so /opt/petalinux/2020.2/sysroots/cortexa9t2hf-neon-xilinx-linux-gnueabi/usr/lib/libQt5Core.so /opt/petalinux/2020.2/sysroots/cortexa9t2hf-neon-xilinx-linux-gnueabi/usr/lib/libQt5Core.so /opt/petalinux/2020.2/sysroots/cortexa9t2hf-neon-xilinx-linux-gnueabi/usr/lib/libQt5Core.so /opt/petalinux/2020.2/sysroots/cortexa9t2hf-neon-xilinx-linux-gnueabi/usr/lib/libQt5Core.so /opt/petalinux/2020.2/sysroots/cortexa9t2hf-neon-xilinx-linux-gnueabi/usr/lib/libGtESv2.so -lpthread	abi	
		-
Would you like to take a quick UI tour? This tour highlights important user interface elements and shows how they are used. To take Take UI Tour Do Not Show Agai the tour later, select Help > UI Tour.	in	×
🛛 🖉 🥂 Type to locate (Ctrl		

图 10.8.13 编译输出信息

到这里 Qt 工程就编译好了。我们将编译生成的可执行文件拷贝到开发板根文件系统目录中,然后在 ZYNQ 开发板系统下运行。

打开 Ubuntu 终端, 进入到 Qt 工程所在目录, 如下:

zy@zy-virtual-m	machine:~\$ l	s				
build-qt test-z	zynq-Debug	qt test		模板	文档	桌面
examples.desktd	op	vmware-to	ols-distrib	视频	下载	
Navigator_7010_	v3	公共的		图片	音乐	
zy@zy-virtual-m	m <mark>achine:~</mark> \$ c	d build-q	t test-zynq-I	Debug/		
zy@zy-virtual-m	<pre>nachine:~/bu</pre>	ild-qt_tes	st-zynq-Debug	g\$ ls		
main.o №	lakefile	m	oc mainwindo	v.o q1	t_test	4
mainwindow.o m	<pre>noc_mainwind</pre>	ow.cpp mo	<pre>predefs.h</pre>	u	i_mainv	window.h
zy@zy-virtual-m	<pre>nachine:~/bu</pre>	ild-qt_tes	st-zynq-Debug	g\$		

图 10.8.14 应用程序所在路径

图 10.8.14 中,qt_test 就是笔者创建的工程,build-qt_test-zynqMP-Debug 目录下存放的就 是工程编译过程中所生成的文件,包括最终的可执行文件都在这个目录下。

其中 qt_test 是最终的可执行文件,可以使用 file 命令查看该文件的属性,如下图所示:



原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php

zy@zy-virtual-machine:~/build-qt_test-zynq-Debug\$ file qt_test _____ qt_test: ELF 32-bit LSB shared object, ARM, EABI5 version 1 (SYSV), dynamically linked, interpreter /lib/ld-linux-armhf.so.3, BuildID[sha1]=37cd113002481844c244 f6105e6b595f2247e7e3, for GNU/Linux 3.2.0, with debug_info, not stripped

图 10.8.15 查看文件属性

由此可以知道该文件是一个 ELF 格式的可执行文件,可以运行在 32 位 ARM 架构的 Linux 系统下,所以我们需要把该文件放置到开发板上运行。

10.9 上板测试

开发板设置为 SD 卡启动模式,并和 LCD 屏连接。

进入 10.3 小节的工程中,设置环境变量,然后运行 "petalinux-package --boot --fsbl --fpga --u-boot --force"命令生成启动文件。

将"BOOT.bin"、"boot.scr"和"image.ub"三个文件复制到 SD 卡第一个分区。删除 SD 卡 ext4 分区原有内容,使用如下命令将 10.3 小节工程中 image/linux 目录下根文件系统 rootfs.tar.gz 解压到 SD 卡 ext4 分区。

sudo tar -xzf rootfs.tar.gz -C /media/zy/rootfs/

上述命令中"/media/zy/rootfs"是 SD 卡挂载到笔者电脑中的路径,读者可根据自己电脑 实际情况修改。在终端中输入"df-Th"命令可查看 SD 卡在 Ubuntu 中挂载的路径,如下图 所示:

vmhgfs-fuse	fuse.vmhgfs-fuse	200G	132G	69G	66%	/mnt/hgfs
tmpfs	tmpfs	1.2G	48K	1.2G	1%	/run/user/1000
/dev/sdb1	vfat	4.0G	8.8M	4.0G	1%	/media/zy/boot
/dev/sdb2	ext4	11G	16M	10G	1%	/media/zy/rootfs
zy@zy-virtual-i	machine:~\$ df -Th ┥					
文件系统	类型	容量	已用	可用	已用%	挂载点
udev	devtmpfs	5.8G	Θ	5.8G	<u>0</u> %	/dev

图 10.9.1 查看 SD 卡挂载点

接下来将上一小节编译得到的 qt_test 复制到 SD 卡 ext4 分区的 home/root 目录中。 启动开发板后进入 linux 系统。输入如下命令设置 Qt 运行环境变量,(如果没有配置

X11,环境变量使用 export QT_QPA_PLATFORM=linuxfb):

```
export DISPLAY=:0.0
结果如下图所示:
```

```
root@ALIENTEK-ZYNQ-cross-compile:~# export QT_QPA_PLATFORM=linuxfb
root@ALIENTEK-ZYNQ-cross-compile:~# ls
qt_test
root@ALIENTEK-ZYNQ-cross-compile:~# ./qt_test
QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/var/volatile/tmp/runtime-ro
ot'
```

图 10.9.2 设置 Qt 运行环境变量

运行 qt_test 程序, LCD 屏显示如下图所示:

领航者 ZYNQ 之嵌入式 Linux 开发指南 ②正 原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

e			May _	2023			•
17	24	25	26	27	28	29	30
18	1	2	3	4	5	6	7
19	8	9	10	11	12	13	14
20	15	16	17	18	19	20	21
21	22	23	24	25	26	27	28

正点原子

图 10.9.3 显示 Qt 界面

在串口终端按 Ctrl + C 快捷键可结束程序运行。

10.10 扩展阅读

10.10.1 安装 Qt

打开 Qt 官网 <u>https://download.qt.io/archive/qt/</u>,选择要下载的 Qt 版本。由于是在 Ubuntu 中安装,所以选择后缀名为 run 的安装包。笔者选择 5.14.2 版本,如下图所示:

Name	Last modified	Size	Metadata
↑ Parent Directory		-	
submodules/	31-Mar-2020 09:27	-	
single/	31-Mar-2020 10:10	-	
t-opensource-windows-x86-5.14.2.exe	31-Mar-2020 10:18	2.3G	Details
t-opensource-mac-x64-5.14.2.dmg	31-Mar-2020 10:16	2.6G	Details
t-opensource-linux-x64-5.14.2.run	31-Mar-2020 10:14	1.2G	Details
Md5sums.txt	31-Mar-2020 10:32	207	Details

图 10.10.1 下载 Qt 安装包

注:不要安装 Qt6 版本,因为 Qt6 不支持 Ubuntu18 版本。

下载完成后,将安装包放到共享文件夹中,然后运行"sudo./qt-opensource-linux-x64-5.14.2.run"命令开始安装,如下图所示:



正点原子

图 10.10.2 安装 Qt

在弹出的窗口中点击"Next",如下图所示:



图 10.10.3 点击 Next

在接下来的界面中输入官网注册好的账号和密码,然后点击"Next",如下图所示:

领航者 ZYNQ	之嵌入式 Linux	开发	指南	② 正点原子
原子哥在线教学:	www.yuanzige.com	论	坛:www.openedv.com/f	orum.php
		Qt 5.14.	2 安装程序	8
	Qt Account - Your unified lo	gin to eve	rything Qt	
	Qt	Login	lease log in to Qt Account	
			Password	
			Confirm Password	
			I accept the <u>service terms</u> .	
	设置	(<上一步(B) Next 取	7消

图 10.10.4 输入账号和密码

在接下来的使用声明界面,勾选同意,然后点击"Next",如下图所示:



继续点击"下一步",来到安装目录选择界面,这里不做修改,使用默认路径,然后点击"下一步",如下图所示:

领航者 ZYNQ 之	嵌入式 Linux 法	开发指南	② 正点原子	-
原子哥在线教学:w	ww.yuanzige.com	论坛:www.opened	v.com/forum.php	
		Qt 5.14.2 安装程序	8	
	安装文件夹			
	<mark>请指定将安装 Qt 5.14.2 的</mark> 目录。	,		
	/opt/Qt5.14.2		浏览(<u>R</u>)	
		\backslash		
-		<上一步(B) 下一步(N)>	取消	

图 10.10.6 选择安装目录

在接下来的组件选择界面,红框里的3项为必选,其它组件为可选,笔者这里只勾选必选项,然后点击"下一步",如下图所示:



图 10.10.7 选择要安装的组件

接下来的许可协议界面,勾选同意,然后点击"下一步",如下图所示:

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

正点原子

许可协议

请阅读以下许可协议。 您必须接受这些协议中的条款才能继续安装。

Qt 5.14.2 安装程序

	Qt Installer LGPL License Agreement GPLv3 with Qt Company GPL Exception License Agreement	
	GENERAL	1
	Qt is available under a commercial license with various pricing models and packages that meet a variety of needs. Commercial Qt license keeps your code proprietary where only you can control and monetize on your end product's development, user experience and distribution. You also get great perks like additional functionality, productivity enhancing tools, world-class support and a close strategic relationship with The Qt Company to make sure your product and development goals are met.	
_	Qt has been created under the belief of open development and providing freedom and choice to developers. To support that, The Qt Company also licenses Qt under open source licenses, where most of the functionality is	Ŧ
L	I have read and agree to the terms contained in the license agreements.	
	○ I <u>d</u> o not accept the terms and conditions of the above license agreements.	
	<上一步(B) 下一步(N) > 取消	

图 10.10.8 同意协议

在接下来的开始安装界面,点击"安装",等待一段时间后,弹出如下图所示界面:



图 10.10.9 安装完成

点击"完成"结束安装。到这里 Qt 就安装好了,点击 Qt 图标可以打开软件,如下图所示:



图 10.10.10 Qt Creator



论坛:www.openedv.com/forum.php

正点原子

第三篇 系统移植篇

在上一篇中我们学习了 Petalinux 的使用,从使用中可以体会到 Petalinux 对开发 Linux 的 方便,通过几个简单的命令就完成了 Linux 系统的搭建,极大的提升了 Linux 开发的效率,是 开发 linux 系统的利器,然而就像硬币的两面一样,有利的一面当然也存在着不利的一面。 Petalinux 封装了细节,只提供了配置接口,对开发者是友好的,但对学习者就未必了。就像 我们使用 Petalinux 搭建完 Linux 系统后,在开发板上成功运行,但对 U-boot、linux 内核、根 文件系统一无所知。一般我们在搭建 Linux 系统时需要移植 Linux,在移植 Linux 之前我们需 要先移植一个 bootloader 代码,这个 bootloader 代码用于启动 Linux 内核, bootloader 有很多, 常用的是 U-Boot。移植好 U-Boot 以后再移植 Linux 内核,移植完 Linux 内核以后 Linux 还不 能正常启动,还需要再移植一个根文件系统(rootfs),根文件系统里面包含了一些最常用的命 令和文件,所以 U-Boot、Linux kernel 和 rootfs 这三者一起构成了一个完整的 Linux 系统,一 个可以正常使用、功能完善的 Linux 系统。在本篇我们就来讲解 U-Boot 的使用以及 Linux Kernel 和 rootfs 的移植,与其说是"移植",倒不如说是"适配",因为大部分的移植工作都 由 Xilinx 完成了,我们这里所谓的"移植"主要是使其能够在开发板上跑起来。

讲解移植的另一个原因是虽然 Petalinux 功能比较全面,但是在编译速度上有点慢,在后面讲解驱动开发的时候,完全使用 Petalinux 会非常不方便,所以讲解了 linux 内核的移植。在后面进行内核驱动开发的时候,就可以单独编译内核,无需借助 Petalinux。

特别提醒:在每个使用 Petalinux 或 arm 交叉编译器 arm-xilinx-linux-gnueabi-的终端都需要先设置 Petalinux 的环境变量,如何设置,见 5.4 节设置 Petalinux 环境变量,如果只需要 arm 的交叉编译器,不需要 Petalinux,可以在终端输入如下命令:

sptl

echo 'export PATH=\$PATH': `which arm-xilinx-linux-gnueabi-gcc | xargs dirname` | tee -a ~/.bashrc

输出结果: export PATH=\$PATH:/opt/petalinux/2020.2/sysroots/x86_64-petalinux-linux/usr/bin/arm-xilinx-linux-gnueabi

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

正点原子

第十一章 U-Boot 使用实验

在移植 U-Boot 之前,我们肯定要先使用一下 U-Boot,体验一下 U-Boot 是什么。领航者 开发板光盘资料里面已经提供了一个已经移植好的 U-Boot,本章我们就直接编译这个移植好 的 U-Boot, 然后烧写到 SD 卡里面启动, 启动 U-Boot 以后就可以学习使用 U-Boot 的命令。



原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php

11.1 U-Boot 简介

对于计算机系统而言,从开机上电到操作系统启动需要一个引导过程,这个引导过程由 引导程序指定。引导程序是系统上电启动运行的第一段软件代码。在 PC 体系结构中,引导程 序由主板上的 BIOS 和位于硬盘 MBR 中的启动代码组成。系统上电后,首先运行 BIOS,在 完成硬件检测和资源分配后,将硬盘 MBR 中的引导程序读到系统的 RAM 中,然后将控制权 交给引导程序。引导程序的主要运行任务就是将内核映像从硬盘读到 RAM 中,然后跳转到内 核的入口点去运行,也即开始启动操作系统。嵌入式 Linux 系统同样离不开引导程序,这个 引导程序一般我们叫作启动加载程序(Bootloader)。

Bootloader 是在操作系统运行之前执行的一段小程序。通过这段小程序,可以初始化硬件 设备、建立内存空间的映射表,从而建立适当的系统软硬件环境,为最终调用操作系统内核 做好准备。也就是说芯片上电以后先运行一段 bootloader 程序。这段 bootloader 程序会先初始 化 DDR 等外设,然后将 Linux 内核从 flash(NAND, NOR FLASH, SD, MMC 等)拷贝到 DDR 中,最后启动 Linux 内核。当然了,bootloader 的实际工作要复杂的多,但是它最主要的工作 就是启动 Linux 内核。

对于嵌入式系统,bootloader 是基于特定硬件平台实现的。因此,几乎不可能为所有的嵌入式系统建立一个通用的 bootloader,不同的处理器架构都有不同的 bootloader。bootloader 不 仅依赖于 CPU 的体系结构,而且依赖于嵌入式系统板级设备的配置。对于两块不同的嵌入式 板而言,即使它们使用同一种处理器,要想让运行在一块板子上的 bootloader 程序也能运行在 另一块板子上,一般需要修改 bootloader 的源程序。庆幸的是,大部分 bootloader 仍具有很多 共性,某些 bootloader 能够支持多种体系结构的嵌入式系统。

现成的 bootloader 软件有很多,比如 U-Boot、vivi、RedBoot 等等,其中以 U-Boot 使用最为广泛,为了方便书写,本书会将 U-Boot 写为 uboot。特别说明的是对于 ZYNQ 而言,在引导过程中,先运行 FSBL 来设置 PS,然后运行 U-Boot 用于加载 Linux 内核映像并引导 Linux,所以 uboot 对于 zynq 而言是第二阶段引导程序,FSBL 是第一阶段引导程序。

uboot 的全称是 Universal Boot Loader, uboot 是一个遵循 GPL 协议的开源软件。uboot 是一个裸机代码,可以看作是一个裸机综合项目。现在的uboot 已经支持液晶屏、网络、USB等高级功能。uboot 官方网址: <u>http://www.denx.de/wiki/U-Boot/</u>,如下图所示:



图 11.1.1 uboot 官网 307



原子哥在线教学: www.yuanzige.com 论坛:www

论坛:www.openedv.com/forum.php

我们可以在 <u>https://ftp.denx.de/pub/u-boot/</u>这个网址下载自己想要的 uboot 源码,下载界面

如下图所示:

C 🗄 https://ftp.denx.de/pub/u-boot/

Index of /pub/u-boot/

<u>1-boot-2020.01.tar.bz2</u>	06-Jan-2020 23	2:57 14M	
<u>1-boot-2020.01.tar.bz2.sig</u>	06-Jan-2020 23	2:57 586	
<u>1-boot-2020.04-rc1.tar.bz2</u>	29-Jan-2020 0	0:00 14M	
1-boot-2020.04-rc1.tar.bz2.sig	29-Jan-2020 0	0:00 586	
1-boot-2020.04-rc2.tar.bz2	12-Feb-2020 1	6:31 14M	
1-boot-2020.04-rc2.tar.bz2.sig	12-Feb-2020 1	6:31 586	
1-boot-2020.04-rc3.tar.bz2	26-Feb-2020 1-	4:56 14M	
1-boot-2020.04-rc3.tar.bz2.sig	26-Feb-2020 1-	4:56 586	
1-boot-2020.04-rc4.tar.bz2	31-Mar-2020 0	1:30 14M	
1-boot-2020.04-rc4.tar.bz2.sig	31-Mar-2020 0	1:30 586	
1-boot-2020.04-rc5.tar.bz2	06-Apr-2020 23	3:45 14M	
1-boot-2020.04-rc5.tar.bz2.sig	06-Apr-2020 2	3:45 586	
1-boot-2020.04.tar.bz2	13-Apr-2020 1	7:03 14M	
<u>1-boot-2020.04.tar.bz2.sig</u>	13-Apr-2020 1	7:03 586	
1-boot-2020.07-rc1.tar.bz2	28-Apr-2020 23	2:02 14M	

图 11.1.2 uboot 源码

上图显示的就是 uboot 原汁原味的源码文件,目前最新的版本是 2023.04。但是我们一般 不会直接用 uboot 官方的 uboot 源码。uboot 官方的 uboot 源码是给半导体厂商准备的,半导体 厂商会下载 uboot 官方的 uboot 源码,然后将自家相应的芯片移植进去。也就是说半导体厂商 会自己维护一个为其设计的芯片定制的 uboot 版本。既然是定制的,那么肯定对自家的芯片支 持会很全,虽然 uboot 官网的源码中一般也会支持他们的芯片,但是绝对是没有半导体厂商自 己维护的 uboot 全面。

Xilinx 维护的 uboot 版本可在网站 <u>https://github.com/Xilinx/u-boot-xlnx</u> 查看,下载地址为: <u>https://github.com/Xilinx/u-boot-xlnx/tags</u>,下载界面如下图所示:

Xilinx / u-boot-xlnx Public	
<> Code 11 Pull requests 3 (•) Actions (!) Security	,
Releases Tags	
🟷 Tags	
xlnx_rebase_v2020.01_2020.1 🚥	
🕚 on Jun 3, 2020 - O- 86c84c0 📓 zip 📓 tar.gz	
xilinx-v2020.1 🚥	
🕚 on Jun 3, 2020 🗢 e44c2bc 📓 zip 📓 tar.gz	
() on Jun 3, 2020 - 86c84c0 [3] zip [3] tar.gz xilinx-v2020.1 () on Jun 3, 2020 - e44c2bc [3] zip [3] tar.gz	

图 11.1.3 Xilinx 官方 uboot 下载界面

点击上图中箭头所指的 zip 或 tar.gz 就可以下载 uboot 源码, xilinx-v2020.1 是 xilinx 自己 维护的一个版本号,并不是 U-Boot 官方对应的版本号。之所以下载这个版本的 uboot,就是 为了和我们使用的 petalinux 工具的版本要匹配。虽然我们使用的 petalinux 的版本为 2020.2,



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

但是使用 petalinux 工具生成的 uboot 源码的版本却为 2020.1,所以这里我们下载 uboot 的版本 与 petalinux 工具生成的 uboot 版本保持一致,这样可以避免因 petalinux 工具与 uboot 版本不匹 配而导致的一系列问题。

后面我们学习 uboot 移植的时候使用的就是上图中的 2020.1 的 uboot。可以从该网址下载 xilinx-v2020.1.tar.gz,也可以使用我们已经下载完成的,放在开发板提供的光盘资料中,路径为:领航者 ZYNQ 开发板资料盘(A 盘)\4_SourceCode\3_Embedded_Linux\资源文件\资源文件\资源文件\资源文件\的

图 11.1.3 中的 uboot 基本支持了 Xilinx 当前所有可以运行 Linux 的芯片,而且支持各种启动方式,比如 EMMC、NAND、NOR FLASH 等等,这些都是 uboot 官方所不支持的。但是图 11.1.3 中的 uboot 是针对 Xilinx 自家评估板的,如果是我们自己做的板子就需要修改 Xilinx 官方的 uboot,使其支持我们自己做的板子,正点原子的 ZYNQ 开发板就是自己做的板子,虽然大部分都参考了 Xilinx 官方的 ZYNQ 开发板,但是还是有很多不同的地方,所以需要修改 Xilinx 官方的 uboot,使其适配正点原子的 ZYNQ 开发板。所以当我们拿到开发板以后,是有三种 uboot 的,这三种 uboot 的区别如下表所示:

种类	描述
uboot 宣方的 uboot 代码	由 uboot 官方维护开发的 uboot 版本,版本更新快,基本包含
uboot 自力的 uboot 气响	所有常用的芯片
半已休厂商的 -	半导体厂商维护的一个 uboot,专门针对自家的芯片,在对
十寻评)间的uboot代码	自家芯片支持上要比 uboot 官方的好
工生板 厂 商 的 wheret 代 印	开发板厂商在半导体厂商提供的 uboot 基础上加入了对自家
开及极广 同时 uboot 代码	开发板的支持

表 11.1.1 三种 uboot 的区别

那么这三种 uboot 该如何选择呢? 首先 uboot 官方的基本是不会用的,因为支持太弱了。 最常用的就是半导体厂商或者开发板厂商的 uboot,如果你用的半导体厂商的评估板,那么就 使用半导体厂商的 uboot,如果你是购买的第三方开发板,比如正点原子的 ZYNQ 开发板,那 么就使用正点原子提供的 uboot 源码(也是在半导体厂商的 uboot 上修改的)。当然了,你也 可以在购买了第三方开发板以后使用半导体厂商提供的 uboot,只不过有些外设驱动可能不支 持,需要自己移植,这个就是我们常说的 uboot 移植。

本节是 uboot 的使用,可以直接使用正点原子已经移植好的 uboot,该 uboot 已经放到了 开发板光盘中,路径为:领航者 ZYNQ 开发板资料盘(A 盘)\4_SourceCode\3_Embedded_Linux\ 资源文件\资源文件\资源文件\uboot\alientek-uboot-v2020.1.tar.gz。

11.2 Petalinux 配置和编译 U-Boot

uboot 的功能有很多,虽然默认的功能一般而言已经够用了,但在项目开发过程中,可能 会使用到 uboot 平时不常用的功能,这时就需要配置 uboot 以使能相应功能。在 Petalinux 中配 置 uboot 的方法很简单,在第六章搭建好的 petalinux 工程目录下,执行如下命令即可进入 uboot 图形配置界面:

petalinux-config -c u-boot

执行后会在终端中打开另一个配置 uboot 的标签页,如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

.config - U-Boot 2020.01 Configuratio

U-Boot 2020.01 Configuration Arrow keys navigate the menu. <enter> selects submenus> (or empty submenus). Highlighted letters are hotkeys. Pressing <y> includes, <n> excludes, <m> modularizes features. Press <esc><esc> to exit, <? > for Help, for Search. Legend: [*] built-in [] excluded <m> module < > module</m></esc></esc></m></n></y></enter>
Apphitesture colort (ADM apphitesture)
Architecture setect (ARM architecture)>
Beet in security and a security and
Boot thages>
Boot timing>
BOOT MEDIA>
(4) delay in seconds before automatically booting
[] Enable boot arguments
[*] Enable a default value for bootcMd
(run distro_bootcmd) bootcmd value
- V(+)
<pre><select> < Exit > < Help > < Save > < Load ></select></pre>

图 11.2.1 uboot 配置界面

这里我们将"delay in seconds before automatically booting"选项配置成"4",其余选项 保持默认即可,若读者有配置 uboot 的其它需求可以自行配置。这里说一下配置项"delay in seconds before automatically booting"的含义,该配置是用来设置启动 uboot 后,延时多长时间 自动启动 linux 系统,默认为 2 秒,这里我们将其配置成 4 秒,如果读者觉得延时时间太长或 者太短,可以重新设置。

配置完成后,保存配置并退出。对于 Petalinux20.2 版本而言,执行上面的配置操作后, 会在当前 Petalinux 工程的 components/yocto/workspace/sources/目录下生成 uboot 的源码,如下 图所示:

wmq@Linux:~/petalinux/ALIENTEK-ZYNQ\$ ls	-l components/yocto/workspace/sources/
心用里 8 drwxr-xr-x 28 wmq wmq 4096 6月 29 15:3%	3 linux-xlnx
drwxr-xr-x 28 wmq wmq 4096 7月 29 14:0	3 u-boot-xlnx
wmq@Linux:~/petalinux/ALIENTEK-ZYNQ\$	

图 11.2.2 uboot 源码

接下来执行如下命令编译 uboot:

petalinux-build -c u-boot

也可以只使用"petalinux-build"命令,用来编译fsbl、uboot、设备树、Linux内核和根文件系统等,如果是重新创建 Petalinux 工程,推荐使用"petalinux-build"命令,不带"-cuboot"参数。

命令执行完以后 uboot 也就编译成功了,如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php q@Linux:~/petalinux/ALIENTEK-ZYNQ\$ petalinux-build -c u-boot INFO: Sourcing build tools [INFO] Building u-boot [INFO] Sourcing build environment [INFO] Generating workspace directory INFO: bitbake virtual/bootloader Parsing of 2995 .bb files complete (2992 cached, 3 parsed). 4265 targets, 204 skipped, 0 masked, 0 e rrors. NOTE: Resolving any missing task queue dependencies JARNING: tainted from a forced run Time: 0:00:00 Sstate summary: Wanted 119 Found 118 Missed 1 Current 810 (99% match, 99% complete) NOTE: Executing Tasks NOTE: Setscene tasks completed NOTE: u-boot-xlnx: compiling from external source tree /home/wmq/petalinux/ALIENTEK-ZYNQ/components/ yocto/workspace/sources/u-boot-xlnx NOTE: Tasks Summary: Attempted 2926 tasks of which 2901 didn't need to be rerun and all succeeded. Summary: There was 1 WARNING message shown. INFO: Successfully copied built images to tftp dir: /tftpboot [INFO] Successfully built u-boot

图 11.2.3 编译 U-Boot

注:有一个警告,Petalinux版本的问题,不影响使用。

编译完成以后会在当前 Petalinux 工程目录的 images/linux 目录下生成 u-boot.elf 文件,如下图所示:

wmq@Linux:~/petalinux/ALIENTEK-ZYNQ\$ ls -l images/linux/u-boot.elf
-rwxrwxr-x 1 wmq wmq 797720 7月 28 14:01 images/linux/u-boot.elf
wmq@Linux:~/petalinux/ALIENTEK-ZYNQ\$

图 11.2.4 u-boot 镜像文件

11.3 U-Boot 烧写与启动

uboot 编译好以后就可以烧写到板子上使用了。我们将 u-boot.elf 文件打包到 ZYNQ 的启 动文件 BOOT.BIN 中,在终端中输入如下命令:

petalinux-package --boot --fsbl --fpga --u-boot --force

生成 BOOT.BIN 文件后,将该工程 image/linux 目录下的 BOOT.BIN 文件拷贝到 SD 卡的 第一个分区也即 FAT32 分区。

拷贝完成后将 SD 卡插到领航者开发板上,启动模式设置从 SD 卡启动,使用 USB 线将开 发板的串口和电脑连接,上电启动开发板;打开串口调试助手,设置好串口的相关参数,最 后按核心板上的 PS_RST 复位按钮复位开发板。

之后串口终端会出现"Hit any key to stop autoboot:" 倒计时,如下所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php



图 11.3.1 U-Boot 启动内核倒计时

在倒计时为 0 之前按下键盘上的回车键来打断倒计时,一旦倒计时完成后没有按下回车 按键,uboot 将会自动启动 Linux 内核,这里我们不让其启动内核,所以要打断它;默认是 4 秒倒计时,当然这个倒计时时间长短可以通过配置 uboot 环境变量进行修改,倒计时被打断之 后就会进入 uboot 的命令行模式,如下图所示:



图 11.3.2 U-Boot 命令行模式

从上图可以看出,当进入到 uboot 的命令行模式以后,左侧会出现一个 "zynq>"标志。 uboot 启动的时候会输出一些信息,这些信息如下所示:

- 1 U-Boot 2020.01 (Aug 23 2023 10:49:37 +0800)
- 2

```
3 CPU: Zynq 7z020
```

4 Silicon: v3.1

- 5 Model: Alientek Navigator Zynq Development Board
- 6 DRAM: ECC disabled 1 GiB
- 7 Flash: 0 Bytes
- 8 NAND: 0 MiB
- 9 MMC: mmc@e0100000: 0, mmc@e0101000: 1

10 Loading Environment from SPI Flash... SF: Detected w25q256 with page size 256 Bytes, erase size 4 KiB,

total 32 MiB

11 *** Warning - bad CRC, using default environment

12



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

13 In: serial@e0000000

14 Out: serial@e0000000

15 Err: serial@e0000000

16 Net:

17 ZYNQ GEM: e000b000, mdio bus e000b000, phyaddr 7, interface rgmii-id

18

19 Warning: ethernet@e000b000 MAC addresses don't match:

20 Address in DT is 00:0a:35:00:8b:87

21 Address in environment is 00:0a:35:00:01:22

22 eth0: ethernet@e000b000

23 ZYNQ GEM: e000c000, mdio bus e000c000, phyaddr 4, interface gmii

24

25 Warning: ethernet@e000c000 using MAC address from DT

26, eth1: ethernet@e000c000

27 Hit any key to stop autoboot: 0

28 Zynq>

第 1 行是 uboot 版本号和编译时间,可以看出,当前的 uboot 版本号是 2020.01,编译时间是 2023 年 8 月 23 日上午 10 点 49 分。

第 3~5 行是开发板相关信息,可以看出当前使用的开发板名称为"Alientek Navigator Zynq Development Board", CPU为"Zynq 7z020",核心板的芯片版本为 3.1。

第6行提示当前板子的DRAM(内存)为1GB,如果是7010的核心板的话内存为512MB。 第7行说明 Flash为0Byte,第8行说明 NAND为0MiB

第 9 行提示当前有两个 MMC 控制器: mmc@e0100000 和 mmc@e0101000, 且 mmc@e010000 接的是 SD(TF)卡, mmc@e0101000 接的是 eMMC。

第10检测到 QSPI--w25q256, 大小为 32MB。

第 13~15 是标准输入、标准输出和标准错误所使用的终端,这里都使用串口(serial)作为终端, serial@e0000000 对应的就是开发板的 USB 调试串口。

第16~26行是网口信息,提示我们当前使用的是 eth0 这个网口, ZYNQ 支持两个网口。

第 24 行是倒计时提示,默认倒计时 4 秒,倒计时结束之前按下回车键就会进入 Uboot 命令行模式。如果在倒计时结束以后没有按下回车键,那么 Linux 内核就会启动, Linux 内核一旦启动,uboot 就会寿终正寝。这个就是 uboot 默认输出信息的含义。

如果在启动时看到 uboot 打印出: "Warming-bad CRC, using default environment",说明 uboot 没有在存放环境变量的固态存储器(一般为 Flash)中找到有效的环境变量,只好使用编译时定义的默认环境变量。如果 uboot 存放环境变量的固态存储器的驱动没问题,那么只要运行 saveenv 就可以把默认环境变量写入固态存储器,下次启动就不会有这个警告了,当然了,也可以忽略该警告。

uboot 是来干活的,我们现在已经进入 uboot 的命令行模式了,进入命令行模式以后就可 以给 uboot 发号施令了。当然了,不能随便发号施令,得看看 uboot 支持哪些命令,然后使用 这些 uboot 所支持的命令来做一些工作。下一节就讲解 uboot 命令的使用。

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

正点原子

11.4 U-Boot 命令使用

进入 uboot 的命令行模式以后输入 "help" 或者 "?", 然后按下回车即可查看当前 uboot 所支持的命令,如图 11.4.1 所示。截图只是 uboot 的一部分命令,并不是 uboot 所支持的 所有命令。前面说过 uboot 是可配置的,需要什么命令就使能什么命令,所以该图中的命令是 正点原子提供的 uboot 中使能的命令, uboot 支持的命令还有很多, 而且也可以在 uboot 中自 定义命令。这些命令后面都有对应的使用说明,用于描述此命令的作用,但是命令具体怎么 用呢?我们输入"help(或?)命令名"就可以查看对应命令的详细用法。

Zynq> help	
?	alias for 'help'
base	print or set address offset
bdinfo	print Board Info structure
blkcache	block cache diagnostics and control
boot	boot default, i.e., run 'bootcmd'
bootd	boot default, i.e., run 'bootcmd'
bootefi	Boots an EFI payload from memory
bootelf	Boot from an ELF image in memory
bootm	boot application image from memory
bootp	boot image via network using BOOTP/TFTP protocol
bootvx	Boot vxWorks from an ELF image
bootz	boot Linux zImage image from memory
clk	CLK sub-system
cmp	memory compare
coninfo	print console devices and information
cp	memory copy
crc32	checksum calculation
dcache	enable or disable data cache
dfu	Device Firmware Upgrade
dhcp	boot image via network using DHCP/TFTP protocol
dm	Driver model low level access
echo	echo args to console
editenv	edit environment variable
env	environment handling commands
erase	erase FLASH memory
exit	exit script
ext2load	load binary file from a Ext2 filesystem
ext21s	list files in a directory (default /)
ext4load	load binary file from a Ext4 filesystem
ext41s	list files in a directory (default /)
ext4size	determine a file's size
ext4write	create a file in the root directory
false	do nothing, unsuccessfully
fatinfo	print information about filesystem
fatload	load binary file from a dos filesystem
fatls	list files in a directory (default /)
fatmkdir	create a directory

图 11.4.1 查看 U-Boot 支持的命令

以"bootm"这个命令为例,我们输入如下命令即可查看"bootm"这个命令的用法: ? bootm 或 help bootm

结果如下图所示:

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

正点原子

yng> help bootm
ootm - boot application image from memory
isage:
ootm [addr [arg]]
 boot application image stored in memory
passing arguments 'arg'; when booting a Linux kernel,
'arg' can be the address of an initrd image
When booting a Linux kernel which requires a flat device-tree
a third argument is required which is the address of the
device-tree blob. To boot that kernel without an initrd image,
use a '-' for the second argument. If you do not pass a third
a bd_info struct will be passed instead
or the new multi component uImage format (FIT) addresses
must be extended to include component or configuration unit name:
addr: <subimg_uname> - direct component image specification</subimg_uname>
addr# <conf_uname> - configuration specification</conf_uname>
Use iminfo command to get the list of existing component
images and configurations.
ub-commands to do part of the bootm sequence. The sub-commands must be
ssued in the order below (it's ok to not issue all sub-commands):
start [addr [arg]]
loados – load OS image
ramdisk - relocate initrd, set env initrd_start/initrd_end
fdt – relocate flat device tree
cmdline - OS specific command line processing/setup
bdt - OS specific bd_t processing
prep - OS specific prep before relocation or go
go – start OS
vng>

图 11.4.2 查看 bootm 命令使用说明

上图详细的列出了"bootm"这个命令的用法,其它的命令也可以使用此方法查询具体的使用方法。接下来我们学习一下一些常用的 uboot 命令。

11.4.1 信息查询命令

常用的和信息查询有关的命令有 3 个: bdinfo、printenv 和 version。先来看一下 bdinfo 命 令,此命令用于查看板子信息,直接输入 "bdinfo"即可,结果如下图所示:

Zemer helinfa
arch_number = 0x00000000
boot_params = 0x00000000
DRAM bank = 0x00000000
-> start = 0x0000000
-> size = 0x40000000
baudrate = 115200 bps
TLB addr = 0x3fff0000
relocaddr = 0x3ff2e000
reloc off = 0x3bf2e000
<pre>irq_sp = 0x3eb05bf0</pre>
sp start = 0x3eb05be0
ARM frequency = 766 MHz
DSP frequency = 0 MHz
DDR frequency = 533 MHz
Early malloc usage: 658 / 800
fdt_blob = 0x3eb05c08
Zyng>

图 11.4.3 bdinfo 命令

从上图中可以得出 DRAM 的起始地址和大小、启动参数保存起始地址、波特率、sp(堆栈 指针)起始地址等信息。

命令 "printenv" 用于输出环境变量信息,uboot 也支持 TAB 键自动补全功能,输入 "print" 然后按下 TAB 键就会自动补全命令,直接输入 "print" 也可以打印出 uboot 的环境 变量,打印出 uboot 环境变量如下图所示:



原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php



图 11.4.4 print 命令

在上图中有很多的环境变量,比如 baudrate、board_name、bootdelay、bootcmd 等等。 uboot 中的环境变量都是字符串,既然叫做环境变量,那么它的作用就和"变量"一样。比如 bootdelay 这个环境变量就表示 uboot 启动延时时间,默认 bootdelay=4,也就默认延时 4 秒。 前面说的 4 秒倒计时就是由 bootdelay 定义的,如果将 bootdelay 改为 5 的话就会倒计时 5s 了。 uboot 中的环境变量是可以修改的,有专门的命令来修改环境变量的值,稍后我们会讲解。

命令 version 用于查看 uboot 的版本号, 输入 "version", uboot 版本号如下图所示:



图 11.4.5 version 命令

从上图可以看出,当前 uboot 版本号为 2020.01,2023 年 6 月 28 日编译的,编译器为 armxilinx-linux-gnueabi-gcc 等信息。

11.4.2 环境变量操作命令

1、修改环境变量

环境变量的操作涉及到两个命令: setenv 和 saveenv, 命令 setenv 用于设置或者修改环境 变量的值。命令 saveenv 用于保存修改后的环境变量, 一般环境变量是存放在外部 flash 中的, uboot 启动的时候会将环境变量从 flash 读取到 DRAM 中。所以使用命令 setenv 修改的是 DRAM 中的环境变量值, 修改以后要使用 saveenv 命令将修改后的环境变量保存到 flash 中, 否则的话 uboot 下一次重启会继续使用以前的环境变量值。

命令 saveenv 使用起来很简单,格式为:

saveenv 环境变量 值



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

saveenv 环境变量 '值 1 值 2 值 3'

比如我们要将环境变量 bootdelay 该为 5, 就可以使用如下所示命令:

setenv bootdelay 5

saveenv

或

上述命令执行过程如下图所示:



图 11.4.6 修改并保存环境变量

在上图中,当我们使用命令 saveenv 保存修改后的环境变量的话会有保存过程提示信息, 根据提示可以看出环境变量保存到了 SPI flash 中,也就是开发板上的 QSPI Flah--w25q256 中, 当然环境变量不一定保存在 QSPI 中,也可以保存在 SD 卡或其它外部存储设备中,可以通过 petalinux 配置。

修改 bootdelay 以后,重启开发板,uboot 就是变为 5 秒倒计时,如下图所示:



图 11.4.7 bootdelay 环境变量修改之后

2、新建环境变量

命令 setenv 也可以用于新建命令,用法和修改环境变量一样,比如我们新建一个环境变量 author, author 的值为 "alientek",那么就可以使用如下命令:

setenv author alientek

saveenv

新建命令 author 完成以后重启 uboot, 然后使用命令 printenv 查看当前环境变量, 如下图 所示:

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php



正点原子

图 11.4.8 添加环境变量

从上图可以看到新建的环境变量: author, 其值为: alientek。

3、删除环境变量

既然可以新建环境变量,那么就可以删除环境变量,删除环境变量也是使用命令 setenv, 要删除一个环境变量只要给这个环境变量赋空值即可,比如我们删除掉上面新建的 author 这 个环境变量,命令如下:

setenv author

saveenv

上面命令中通过 setenv 给 author 赋空值,也就是什么都不写来删除环境变量 author。重启 uboot 就会发现环境变量 author 没有了。

11.4.3 内存操作命令

内存操作命令就是用于直接对 DRAM 进行读写操作的,常用的内存操作命令有 md、nm、mm、mw、cp和 cmp。我们依次来看一下这些命令都是做什么的。

1、md 命令

md 命令用于显示内存值,格式如下:

md [.b, .w, .l] address [# of objects]

命令中的[.b.w.l]对应 byte、word 和 long,也就是分别以 1 个字节、2 个字节、4 个字节 来显示内存值。address 就是要查看的内存起始地址,[# of objects]表示要查看的数据长度,这 个数据长度单位不是字节,而是跟你所选择的显示格式有关。比如你设置要查看的内存长度 为 20(十六进制为 0x14),如果显示格式为.b 的话那就表示 20 个字节;如果显示格式为.w 的话 就表示 20 个 word,也就是 20*2=40 个字节;如果显示格式为.l 的话就表示 20 个 long,也就 是 20*4=80 个字节。另外要注意:

uboot 命令中的数字都是十六进制的,不是十进制的。

比如你想查看以 0X8000000 开始的 20 个字节的内存值,显示格式为.b 的话,应该使用如下所示命令:

md.b 8000000 14

而不是:

md.b 8000000 20

上面说了,uboot 命令里面的数字都是十六进制的,所以可以不用写"0x"前缀,十进制的 20 其十六进制为 0x14,所以命令 md 后面的数应该是 14,如果写成 20 的话就表示查看 32(十六进制为 0x20)个字节的数据。分析下面三个命令的区别:

md.b 8000000 10



原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php

正点原子

md.w 8000000 10

md.1 8000000 10

上面这三个命令都是查看以 0X8000000 为起始地址的内存数据, 第一个命令以.b 格式显 示,长度为 0x10,也就是 16 个字节; 第二个命令以.w 格式显示,长度为 0x10,也就是 16*2=32个字节;最后一个命令以.1格式显示,长度也是 0x10,也就是 16*4=64 个字节。这三 个命令的执行结果如下图所示:

			4		
Zynq> md.	b 8000000	10 🔶			
08000000:	ff ff ff	ff ff ff	ff ff ff	ff ff ff ff	ff ff ff
Zynq> md.	w 8000000 w	10 🔶			
08000000:	ffff ffff	ffff fff	f ffff ff	ff ffff fff	f
08000010:	ffff ffff	ffff fff	f ffff ff	ff ffff fff	f
Zynq> md.	1 8000000	10 🔶			
08000000:	fffffff	fffffff	fffffff	fffffff	
08000010:	fffffff	fffffff	fffffff	fffffff	
08000020:	fffffff	fffffff	fffffff	fffffff	
08000030:	fffffff	fffffff	fffffff	fffffff	
Zynq>					

图 11.4.9 md 命令使用示例

2、nm 命令

nm 命令用于修改指定地址的内存值, 命令格式如下:

nm [.b, .w, .l] address

nm 命令同样可以以.b、.w 和.1 来指定操作格式,比如现在以.1 格式修改 0x8000000 地址的 数据为 0x12345678。输入命令:

nm.1 8000000

输入上述命令以后如下图所示:

Zynq> nm.1 8000000 08000000: ffffffff

图 11.4.10 nm 命令

在上图中,8000000 表示现在要修改的内存地址,ffffffff 表示地址 0x8000000 现在的数 据,?后面就可以输入要修改后的数据 0x12345678,输入完成以后按下回车,然后再输入 'q'即可退出,如下图所示:

Zynq> nm.1 8000000		ĺ
08000000: fffffff	? 12345678 🔶	
080000 <u>0</u> 0: 12345678	? q 🔶	
Zynq>		

图 11.4.11 修改内存数据

修改完成以后在使用命令 md 来查看一下有没有修改成功,如下图所示:

Zynq> md.1 8000000	1	
080000 <u>0</u> 0: 12345678		xV4.
Zynq>		

图 11.4.12 查看修改后的值

从上图可以看出,此时地址 0X8000000 的值变为了 0x12345678。

3、mm 命令



原子哥在线教学: www.yuanzige.com 说

论坛:www.openedv.com/forum.php

mm命令也是修改指定地址内存值的,使用mm修改内存值的时候地址会自增,而使用命令 nm的话地址不会自增。比如以.1格式修改从地址 0x8000000 开始的连续 3 个内存块(3*4=12 个字节)的数据为 0X05050505,操作如下图所示:

Zynq> mm.]	L 8000000				
:00000080	12345678	?	05050505		
08000004:	fffffff	?	05050505		
:8000008	fffffff	?	05050505		
080000c:	fffffff	?	q		
Zynq>					

图 11.4.13 命令 mm

从上图可以看出,修改了地址 0X8000000、0X8000004 和 0X8000008 的内容为 0x05050505。使用命令 md 查看修改后的值,结果如下图所示:

Zynq> md.1 8000000 3 08000000: 05050505 05050505 05050505 Zynq>

图 11.4.14 查看修改后的内存数据

从上图可以看出内存数据修改成功。

4、mw 命令

命令 mw 用于使用一个指定的数据填充一段内存,命令格式如下:

mw [.b, .w, .l] address value [count]

mw 命令同样可以以.b、.w 和.l 来指定操作格式, address 表示要填充的内存起始地址, value 为要填充的数据, count 是填充的长度。比如使用.l 格式将以 0X8000000 为起始地址的 0x10 个内存块(0x10*4=64字节)填充为 0X0A0A0A0A, 命令如下:

mw.1 8000000 0A0A0A0A 10

然后使用命令 md 来查看,如下图所示:

Zynq> m	nw.l	8000000	0A0A0A0A	10		
Zynq> m	nd.l	8000000	10			
0800000	00:	0a0a0a0a	0a0a0a0a	0a0a0a0a	0a0a0a0a	
0800001	10:	0a0a0a0a	0a0a0a0a	0a0a0a0a	0a0a0a0a	
0800002	20:	0a0a0a0a	0a0a0a0a	0a0a0a0a	0a0a0a0a	
0800003	30:	0a0a0a0a	0a0a0a0a	0a0a0a0a	0a0a0a0a	
Zynq>						

图 11.4.15 查看修改后的内存数据

从上图可以看出内存数据修改成功。

5、cp 命令

cp 是数据拷贝命令,用于将 DRAM 中的数据从一段内存拷贝到另一段内存中。命令格式如下:

cp [.b, .w, .l] source target count

cp 命令同样可以以.b、.w 和.l 来指定操作格式, source 为源地址, target 为目的地址, count 为拷贝的长度。我们使用.l 格式将 0x8000000 处的地址拷贝到 0X8000100 处, 长度为 0x10 个内存块(0x10 * 4=64 个字节), 命令如下所示:

cp.1 8000000 8000100 10



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

结果如下图所示:

Zynq> md.1 8000000	10			
08000000: 0a0a0a0a	0a0a0a0a	0a0a0a0a	0a0a0a0a	
08000010: 0a0a0a0a	0a0a0a0a	0a0a0a0a	0a0a0a0a	
08000020: 0a0a0a0a	0a0a0a0a	0a0a0a0a	0a0a0a0a	
08000030: 0a0a0a0a	0a0a0a0a	0a0a0a0a	0a0a0a0a	
Zynq> md.1 8000100	10			
08000100: ffffffff	fffffff	fffffff	fffffff	
08000110: ffffffff	fffffff	fffffff	fffffff	
08000120: ffffffff	fffffff	fffffff	fffffff	
08000130: ffffffff	fffffff	fffffff	fffffff	
Zynq> cp.1 8000000	8000100 1	10 🔶 ———————————————————————————————————		
Zynq> md.1 8000100	10			
08000100: 0a0a0a0a	0a0a0a0a	0a0a0a0a	0a0a0a0a	
08000110: 0a0a0a0a	0a0a0a0a	0a0a0a0a	0a0a0a0a	
08000120: 0a0a0a0a	0a0a0a0a	0a0a0a0a	0a0a0a0a	
08000130: 0a0a0a0a	0a0a0a0a	0a0a0a0a	0a0a0a0a	
Zynq>				

图 11.4.16 cp 命令操作结果

在上图中, 先使用 md.l 命令打印出地址 0x8000000 和 0x8000100 处的数据, 然后使用命 令 cp.1 将 0x8000000 处的数据拷贝到 0x8000100 处。最后使用命令 md.1 查看 0x8000100 处的 数据有没有变化,检查拷贝是否成功。

6、cmp 命令

cmp是比较命令,用于比较两段内存的数据是否相等,命令格式如下:

cmp [.b, .w, .l] addr1 addr2 count

cmp 命令同样可以以.b、.w 和.l 来指定操作格式, addr1 为第一段内存首地址, addr2 为第 二段内存首地址, count 为要比较的长度。我们使用.1 格式来比较 0x8000000 和 0X8000100 这 两个地址数据是否相等,比较长度为0x10个内存块(16*4=64个字节),命令如下所示:

cmp.1 8000000 8000100 10

结果如下图所示:

Zynq>	cmp.1	8000000	8000100 10 🔶
Total	of 16	word(s)	were the same
Zynq>			

图 11.4.17 cmp 命令比较结果

从上图可以看出两段内存的数据相等。我们再随便挑两段内存比较一下,比如地址 0x8000000 和 0x8000200,长度为 0X10,比较结果如下图所示:

> cmp.1 8000000 8000200 10 word at 0x08000000 (0xa0a0a0a) != word at 0x08000200 (0xffffffff) Fotal of 0 word(s) were the same Zvna>

从上图可以看出,0x8000000处的数据和0x8000200处的数据就不一样。

11.4.4 网络操作命令

uboot 是支持网络的,我们在移植 uboot 的时候一般都要调通网络功能,因为在移植 linux kernel的时候需要使用到uboot的网络功能做调试。uboot支持大量的网络相关命令,比如dhcp、 ping、nfs 和 tftpboot,我们接下来依次学习一下这几个和网络有关的命令。

图 11.4.18 cmp 命令比较结果



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

在使用 uboot 的网络功能之前先用网线将开发板的以太网 GE PS 接口和电脑或者路由器 连接起来,领航者开发板有两个网口:GE_PS和GE_PL,一定要连接GE_PS,不能连接错了, GE_PS 接口如下图所示。



图 11.4.19 GE PS 网络接口

建议开发板和主机 PC 都连接到同一个路由器上。与网络相关的环境变量如下表所示:

表 11.4.1 网络相关环境变量

环境变量	描述
ipaddr	开发板 ip 地址,可以不设置,使用 dhcp 命令来从路由器获取 IP 地址
ethaddr	开发板的 MAC 地址,一定要设置
gatewayip	网关地址
netmask	子网掩码
serverip	服务器 IP 地址,也就是 Ubuntu 主机 IP 地址,用于调试代码

连接网线,开发板上电后 uboot 默认通过 dhcp 获取网络 ip 地址(与路由器连接时有效), 若与电脑直连,可以通过以下命令手动设置:

setenv ipaddr 192.168.1.10

setenv ethaddr 00:0a:35:00:1e:53

setenv gatewayip 192.168.1.1

setenv netmask 255.255.255.0

setenv serverip 192.168.1.20

saveenv

注意,如果连接到电脑的以太网接口需要同时设置电脑以太网适配器的 IPv4 属性。另外 网络地址环境变量的设置要根据自己的实际情况,确保 Ubuntu 主机和开发板的 IP 地址在同一 个网段内,比如笔者现在的开发板和电脑都在 192.168.1.0 这个网段内,所以设置开发板的 IP 地址为 192.168.1.10, 笔者的 Ubuntu 主机的地址为 192.168.1.20, 因此 serverip 就是 192.168.1.20。ethaddr 为网络 MAC 地址, 是一个 48bit 的地址, 如果在同一个网段内有多个开



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

发板的话一定要保证每个开发板的 ethaddr 是不同的,否则通信会有问题。设置好网络相关的 环境变量以后就可以使用网络相关命令了。

以上有一些环境变量其实不需要我们手动设置,我们所使用的 U-Boot 会帮我们设置,譬如 ethaddr,大家可以打印出 U-Boot 的环境变量,查找有没有设置以上环境变量,如果有的话就不用自己手动设置了。

与路由器连接时需要设置 serverip,也就是 TFTP 服务器的 IP 地址。由于 TFTP 服务器运行在 Ubuntu 主机上,所以 serverip 为 Ubuntu 主机的 IP 地址(如笔者的为 192.168.1.20):

setenv serverip 192.168.1.20

saveenv

1、ping 命令

开发板的网络能否使用,是否可以和服务器(Ubuntu 主机)进行通信,通过 ping 命令就可以验证,直接 ping 服务器的 IP 地址即可,比如我的服务器 IP 地址为192.168.1.20,命令如下:

ping 192.168.1.20

结果如下图所示:



图 11.4.20 ping 命令

从上图可以看出,192.168.1.20 这个主机存在,说明 ping 成功,uboot 的网络工作正常。 注意:如果是通过网线直接将领航者开发板和电脑连接,ping 不通的情况,除了以上设 置不正确外,还有可能是电脑的防火墙导致的。可以关闭防火墙或者允许文件和打印共享应 用通过防火墙(针对 Windows 系统)。

2、dhcp 命令

dhcp 用于从路由器获取 IP 地址,前提是开发板连接到路由器,如果开发板是和电脑直连的,那么 dhcp 命令就会失效。直接输入 dhcp 命令即可通过路由器获取到 IP 地址,如下图所示:

Zynq> dhcp 🔶	
BOOTP broadcast 1	
BOOTP broadcast 2	
DHCP client bound to address 192.1	.68.1.117 (286 ms)
Zynq>	

图 11.4.21 dhcp 命令

从上图可以看出,开发板通过 dhcp 获取到的 IP 地址为 192.168.1.117,和我们手动设置的一样。

3、nfs 命令

nfs 也就是网络文件系统,通过 nfs 可以在计算机之间通过网络来分享资源,比如我们将 linux 镜像和设备树文件放到 Ubuntu 中,然后在 uboot 中使用 nfs 命令将 Ubuntu 中的 linux 镜 像和设备树下载到开发板的 DRAM 中。这样做的目的是为了方便调试 linux 镜像和设备树, 也就是网络调试,通过网络调试是 Linux 开发中最常用的调试方法。原因是嵌入式 linux 开发 不像单片机开发,可以直接通过 JLINK 或 STLink 等仿真器将代码直接烧写到单片机内部的



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

flash 中,嵌入式 Linux 通常是烧写到 SD 卡、NAND Flash、SPI Flash 等外置 flash 中,但是嵌入式 Linux 开发也没有 MDK、IAR 这样的 IDE,更没有烧写算法,因此不可能通过点击一个"download"按钮就将固件烧写到外部 flash 中。虽然半导体厂商一般都会提供一个烧写固件的软件,但是这个软件使用起来比较复杂,这个烧写软件一般用于量产的。其远没有 MDK、IAR 的一键下载方便,在 Linux 内核调试阶段,如果用这个烧写软件的话将会非常浪费时间,而这个时候网络调试的优势就显现出来了,可以通过网络将编译好的 linux 镜像和设备树文件下载到 DRAM 中,然后就可以直接运行。

我们一般使用 uboot 中的 nfs 命令将 Ubuntu 中的文件下载到开发板的 DRAM 中,在使用 之前需要开启 Ubuntu 主机的 NFS 服务,并且要新建一个 NFS 使用的目录,以后所有要通过 NFS 访问的文件都需要放到这个 NFS 目录中。Ubuntu 的 NFS 服务开启我们在 4.4.1 小节已经 详细讲解过了,包括 NFS 文件目录的创建,如果忘记的话可以去查看一下该小节。笔者设置 的/home/wmq/workspace/nfs 这个目录为笔者的 NFS 文件目录。uboot 中的 nfs 命令格式如下所 示:

nfs [loadAddress] [[hostIPaddr:]bootfilename]

loadAddress 是要保存的DRAM地址, [[hostIPaddr:]bootfilename]是要下载的文件地址。这 里我们将当前工程目录 images/linux 下的 zImage 文件复制到 NFS 目录下,比如笔者放到 /home/wmq/workspace/nfs 这个目录下,完成后的 NFS 目录如下图所示:

wmq@Linux:~/workspace/nfs\$	pwd
/home/wmq/workspace/nfs	
wmq@Linux:~/workspace/nfs\$	ls
zImage	
wmg@Linux:~/workspace/nfs\$	

图 11.4.22 NFS 目录中的 zImage 文件

准备好以后就可以使用 nfs 命令来将 zImage 下载到开发板 DRAM 的 0x00000000 地址处, 命令如下:

nfs 00000000 192.168.1.20:/home/wmq/workspace/nfs/zImage

命令中的"0000000"表示 zImage 保存地址, "192.168.1.20:/home/wmq/workspace/nfs/z Image"表示 zImage 在 192.168.1.20 这个主机中,路径为/home/wmq/workspace/nfs/zImage。下 载过程如下图所示:

7.mm => =================================
Zynd> nis 0000000 192.188.1.201/nome/wmd/workspace/nis/zimage
Using ethernet@e000b000 device
File transfer via NFS from server 192.168.1.20; our IP address is 192.168.1.10
Filename '/home/wmq/workspace/nfs/zImage'.
Load address: 0x0

done
Bytes transferred = 4325984 (420260 hex)

图 11.4.23 nfs 命令下载文件


原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

在上图中会以"#"提示下载过程。如果出现"ERROR: `serverip' not set"的错误,就需要设置 serverip为 Ubuntu 主机的 IP 地址。笔者的 Ubuntu 主机的 IP 地址为 192.168.1.20,因此使用命令"setenv serverip 192.168.1.20"设置 serverip为 192.168.1.20。下载完成以后会提示下载的数据大小,这里下载的 4325984 字节,而 zImage 的大小就是 4325984 字节,如下图所示:

<pre>wmq@Linux:~/workspace/nfs\$ ls -l zImage</pre>	
-rw-rr 1 wmq wmq 432598 <u>4</u> 6月 27 15:02 zImage	
wmq@Linux:~/workspace/nfs\$ 🥄	
wmq@Linux:~/workspace/nfs\$	

图 11.4.24 zImage 文件大小

下载完成以后查看 0x0000000 地址处的数据,使用命令 md.b 来查看前 0x100 个字节的数据,如下图所示:

md.b 00000000 100

Zung> md k	- 00	000	000	110	10 -	-											
agings mail	~~	000	- 0	- 1	00	00	- 0	- 1	~~	~~		- 1	~~	00		- 1	
00000000:	00	00	au	еī	00	00	au	еı	00	00	au	еı	00	00	au	eī	
00000010:	00	00	a 0	e1	00	00	a 0	e1	00	00	a 0	e1	00	00	a 0	e1	
00000020:	05	00	00	ea	18	28	6f	01	00	00	00	00	60	02	42	00	(о`.В.
00000030:	01	02	03	04	45	45	45	45	a 0	31	00	00	00	90	0f	e1	EEEE.1
00000040:	46	0b	00	eb	01	70	a0	e1	02	80	a 0	e1	00	20	0f	e1	Fp
00000050:	03	00	12	e3	01	00	00	1a	17	00	a 0	e3	56	34	12	ef	v4
00000060:	00	00	0f	e1	1a	00	20	e2	1f	00	10	e3	1f	00	c 0	e3	
00000070:	d3	00	80	e3	04	00	00	1a	01	0c	80	e3	0c	e0	8f	e2	
:08000000	00	f0	6f	e1	0e	£3	2e	e1	6e	00	60	e1	00	f0	21	e1	on.`!.
00000090:	09	£0	6f	e1	00	00	00	00	00	00	00	00	00	00	00	00	
000000a0:	0f	40	a 0	e1	3e	43	04	e2	02	49	84	e2	0f	00	a 0	e1	.@>cI
:000000b0	04	00	50	e1	b0	01	9f	35	0f	00	80	30	00	00	54	31	р50т1
000000c0:	01	40	84	33	6d	00	00	2b	5f	0f	8f	e2	4e	1c	90	e8	.@.3m+N
:000000d0	1c	d0	90	e5	01	00	40	e0	00	60	86	e0	00	a 0	8a	e0	· · · · · · · @ · · ` · · · · · · ·
000000e0:	00	90	da	e5	01	e0	da	e5	0e	94	89	e1	02	e0	da	e5	
000000f0:	03	a 0	da	e5	0e	98	89	e1	0a	9c	89	e1	00	d0	8d	e0	
Zynq>																	

图 11.4.25 查看 DDR 内存中的数据

使用 od 命令或 xxd 命令来查看 Ubuntu 下的 zImage 文件,检查一下下载到开发板 DDR 中的数据是否与 zImage 原文件一致,命令如下:

od -tx1 -vN 0x100 zImage

或

xxd -g 1 -1 0x100 zImage 结果如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

1.4																	
wmq@Linux	:~/\	worl	kspa	ace	/nfs	\$\$	кхd	- g	1 ·	-1 (9x1(90 i	zIma	age	\blacklozenge		
00000000:	00	00	a0	e1	00	00	a0	e1	00	00	a0	e1	00	00	a0	e1	
00000010:	00	00	a0	e1	00	00	a0	e1	00	00	a0	e1	00	00	a0	e1	
00000020:	05	00	00	ea	18	28	бf	01	00	00	00	00	60	02	42	00	(o`.B.
00000030:	01	02	03	04	45	45	45	45	a0	31	00	00	00	90	0f	e1	EEEE.1
00000040:	46	øЬ	00	eb	01	70	a0	e1	02	80	a0	e1	00	20	0f	e1	Fp
00000050:	03	00	12	e3	01	00	00	1 a	17	00	a0	e3	56	34	12	ef	V4
00000060:	00	00	0f	e1	1 a	00	20	e2	1f	00	10	e3	1f	00	c 0	e3	
00000070:	d3	00	80	e3	04	00	00	1 a	01	0c	80	e3	0c	e0	8f	e2	
00000080:	00	f0	бf	e1	0e	f3	2e	e1	бе	00	60	e1	00	f0	21	e1	on.`!.
00000090:	09	f0	бf	e1	00	00	00	00	00	00	00	00	00	00	00	00	
000000a0:	0f	40	a0	e1	3e	43	04	e2	02	49	84	e2	0f	00	a0	e1	.@>CI
000000b0:	04	00	50	e1	ЬO	01	9f	35	0f	00	80	30	00	00	54	31	P50T1
000000c0:	01	40	84	33	бd	00	00	2b	5f	0f	8f	e2	4e	1c	90	e8	.@.3m+N
000000d0:	1c	d0	90	e5	01	00	40	e0	00	60	86	e0	00	a0	8a	e0	@`
000000e0:	00	90	da	e5	01	e0	da	e5	0e	94	89	e1	02	e0	da	e5	
000000f0:	03	a0	da	e5	0e	98	89	e1	0a	9c	89	e1	00	d 0	8d	e0	
wmq@Linux	:~/\	worl	kspa	ace	/nfs	\$											

图 11.4.26 zImage 文件数据

可以看出图 11.4.25 和图 11.4.26 的前 0x100 个字节的数据一致,说明 nfs 命令下载到的 zImage 是正确的。

4、tftpboot 命令

tftpboot 命令的作用和 nfs 命令一样,都是用于通过网络下载文件到 DRAM 中,只是 tftpboot 命令使用的是 TFTP 协议, Ubuntu 主机作为 TFTP 服务器。

将 zImage 镜像文件拷贝到 tftpboot 文件夹中,这一步我们在 6.3.8 节编译完成后 Petalinux 会自动将 zImage 镜像拷贝到 Ubuntu tftp 服务器所对应的目录中,笔者 Ubuntu 系统配置的 tftp 服务器路径为/tftpboot,如下所示:

wmq@Linux:/tftpboot\$ ls										
BOOT.BIN pxelinux.cfg	rootfs.cpio.gz.u-boot	rootfs.tar.gz	u-boot.bin	vmlinux						
boot.scr rootfs.cpio	rootfs.jffs2	system.bit	u-boot.elf	zImage 📥						
image.ub rootfs.cpio.gz	rootfs.manifest	system.dtb	uImage	zyng fsbl.elf						
wmq@Linux:/tftpboot\$										

图 11.4.27 tftp 服务器目录下的 zImage 文件

万事俱备,只剩验证了,uboot中的tftp命令格式如下:

tftpboot [loadAddress] [[hostIPaddr:]bootfilename]

看起来和 nfs 命令格式一样的, loadAddress 是文件在 DRAM 中的存放地址, [[hostIPaddr:]bootfilename]是要从 Ubuntu 中下载的文件。但是和 nfs 命令的区别在于, tftpboot 命令不需要输入文件在 Ubuntu 中的完整路径,只需要输入文件名即可。比如我们现在将 tftpboot 文件夹里面的 zImage 文件下载到开发板 DRAM 的 0x00000000 地址处,命令如下:

tftpboot 00000000 zImage

下载过程如下图所示:



正点原子

图 11.4.28 tftp 下载

从上图可以看出,zImage 下载成功了,网速为 3.6MiB/s,文件大小为 4325984 字节。同样的,可以使用 md.b 命令来查看前 100 个字节的数据是否和上图中的相等。有时候使用 tftpboot 命令从 Ubuntu 中下载文件的时候会出现如 "TFTP error: 'Permission denied' (0)" 这样的错误提示,提示没有权限,出现这个错误一般有两个原因:

①、在 Ubuntu 中创建 tftpboot 目录的时候没有给予 tftboot 相应的权限。

②、tftpboot 目录中要下载的文件没有给予相应的权限。

针对上述两个问题,使用命令 "chmod 777 xxx" 来给予权限,其中 "xxx" 就是要给予权限的文件或文件夹。

好了,uboot中关于网络的命令就讲解到这里,我们最常用的就是 ping、nfs 和 tftpboot 这 三个命令。使用 ping 命令来查看网络的连接状态,使用 nfs 和 tftp 命令来从 Ubuntu 主机中下 载文件。

11.4.5 EMMC 和 SD 卡操作命令

uboot 支持 EMMC 和 SD 卡,因此也要提供 EMMC 和 SD 卡的操作命令。一般认为 EMMC 和 SD 卡是同一个东西,所以没有特殊说明,本教程统一使用 MMC 来代指 EMMC 和 SD 卡。 uboot 中常用于操作 MMC 设备的命令为"mmc"。

mmc 是一系列的命令,其后可以跟不同的参数,输入"?mmc"即可查看 mmc 有关的命 令,如下图所示:

Zynq> ? mmc	
mmc - MMC sub system	
Usage:	
mmc info - display info of the current MMC device	
mmc read addr blk# cnt	
mmc write addr blk# cnt	
mmc erase blk# cnt	
mmc rescan	
mmc part - lists available partition on current mmc device	
mmc dev [dev] [part] - show or set current mmc device [partition]	
mmc list – lists available devices	
mmc hwpartition [args] - does hardware partitioning	
arguments (sizes in 512-byte blocks):	
[user [enh start cnt] [wrrel {on off}]] - sets user data area att	ributes
[gp1 gp2 gp3 gp4 cnt [enh] [wrrel {on off}]] - general purpose pa	rtition
[check set complete] - mode, complete set partitioning completed	
WARNING: Partitioning is a write-once setting once it is set to com	plete.
Power cycling is required to initialize partitions after set to com	plete.
mmc setdsr <value> - set DSR register value</value>	
Zynq>	

图 11.4.29 mmc 命令

从上图可以看出, mmc 后面跟不同的参数可以实现不同的功能, 如下表所示:



正点原子

	表	11	.4.2	mmc	命	ş
--	---	----	------	-----	---	---

命令	描述
mmc info	输出 MMC 设备信息
mmc read	读取 MMC 中的数据
mmc wirte	向 MMC 设备写入数据
mmc erase	擦除 MMC 中的数据
mmc rescan	扫描 MMC 设备
mmc part	列出 MMC 设备的分区
mmc dev	切换 MMC 设备
mmc list	列出当前有效的所有 MMC 设备
mmc hwpartition	设置 MMC 设备的分区
mmc setdsr	设置 DSR 寄存器的值

1、mmc info 命令

mmc info 命令用于输出当前选中的 mmc info 设备的信息,输入命令"mmc info"即可, 如下图所示:



图 11.4.30 mmc info 命令

从上图可以看出,当前选中的 MMC 设备是 SD 卡,版本为 3.0,容量为 14.8GiB,速度为 50000000Hz=50MHz, 4 位宽的总线。还有一个与 mmc info 命令相同功能的命令: mmcinfo, "mmc"和"info"之间没有空格。

2、mmc rescan 命令

mmc rescan 命令用于扫描当前开发板上所有的 MMC 设备,包括 EMMC 和 SD 卡,输入 "mmc rescan"即可。

3、mmc list 命令

mmc list 命令用于来查看当前开发板一共有几个 MMC 设备, 输入"mmc list", 结果如 下图所示:



图 11.4.31 扫描 MMC 设备

可以看出当前开发板有两个 MMC 设备: mmc@e0100000:0 (SD)和 mmc@e0101000:1, mmc@e0100000: 0 (SD)是 SD 卡, mmc@e0101000: 1 是 EMMC。默认会将 SD 卡设置为当前 MMC 设备,这就是为什么输入"mmc info"查询到的是 SD 卡信息,而不是 EMMC 设备。要 想查看 SD 卡信息,就要使用命令"mmc dev"来将 EMMC 设置为当前的 MMC 设备。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

4、mmc dev 命令

mmc dev 命令用于切换当前 MMC 设备, 命令格式如下:

mmc dev [dev] [part]

[dev]用来设置要切换的 MMC 设备号, [part]是分区号。如果不写分区号的话默认为分区 0。使用如下命令切换到 eMMC:

mmc dev 1 //切换到 eMMC, 0为 SD 卡, 1为 eMMC

结果如下图所示:

Zyng> mmc	dev 1 🔶
switch to	partitions #0, OK
mmc1(part	0) is current device
Zynq>	

图 11.4.32 切换到 eMMC

从上图可以看出,切换到 eMMC 成功, mmc1 为当前的 MMC 设备,输入命令 "mmc info" 即可查看 eMMC 的信息,结果如下图所示:



图 11.4.33 eMMC 信息

从上图可以看出当前 eMMC 为 5.1 版本,容量为 7.3GiB(8GB 的 eMMC),4 位宽的总线。

5、mmc part 命令

有时候 SD 卡或者 EMMC 会有多个分区,可以使用命令"mmc part"来查看其分区,比 如查看 SD 卡的分区情况,输入如下命令:

//切换到 SD 卡 mmc dev 0

//查看 SD 卡分区 mmc part

结果如下图所示:

Zynq> m switch f mmc0 is Zynq> m	mc dev 0 (to partitions # 0 current device mc part (, OK		
Partiti	on Map for MMC d	evice 0	Partition Type: DO	s
Part 1 2	Start Sector 2048 1026048	Num Sectors 1024000 30090240	UUID 00000000-01 00000000-02	Type Oc Boot 83
Zyng>				

图 11.4.34 查看 SD 卡分区



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

从上图中可以看出,此时 SD 卡有两个分区,扇区 2048~1024000 为第一个分区,扇区 1026048~30090240 为第二个分区。

6、mmc read 命令

mmc read 命令用于读取 mmc 设备的数据,命令格式如下:

mmc read addr blk# cnt

addr 是数据读取到 DRAM 中的地址, blk 是要读取的块起始地址(十六进制), 一个块是 512 字节, 这里的块和扇区是一个意思, 在 MMC 设备中我们通常说扇区, cnt 是要读取的块 数量(十六进制)。比如从 SD 卡的第 2048(0x800)个块开始, 读取 16(0x10)个块的数据到 DRAM 的 0X00000000 地址处, 命令如下:

mmc dev $0\,0$

//切换到 SD 卡的 0 分区 //读取数据

结果如下图所示:

mmc read 00000000 800 10



图 11.4.35 mmc read 命令

7、mmc write 命令

要将数据写到 MMC 设备里面,可以使用命令"mmc write",格式如下:

mmc write addr blk# cnt

addr 是要写入 MMC 中的数据在 DRAM 中的起始地址, blk 是要写入 MMC 的块起始地址 (十六进制), cnt 是要写入的块大小, 一个块为 512 字节。注意千万不要写 SD 卡或者 EMMC 的前两个块(扇区), 里面保存着分区表信息。

8、mmc erase 命令

如果要擦除 MMC 设备的指定块就是用命令"mmc erase",命令格式如下:

mmc erase blk# cnt

blk 为要擦除的起始块, cnt 是要擦除的数量。没事不要用 mmc erase 来擦除 MMC 设备。

关于 MMC 设备相关的命令就讲解到这里,表 11.4.2 中还有一些跟 MMC 设备操作有关的 命令,但是很少用到,这里就不讲解了,感兴趣的可以上网查一下,或者在 uboot 中查看这些 命令的使用方法。

11.4.6 FAT 格式文件系统操作命令

有时候需要在 uboot 中对 SD 卡或者 EMMC 中存储的文件进行操作,这时候就要用到文件操作命令,跟文件操作相关的命令有: fatinfo、fatls、fstype、fatload 和 fatwrite,但是这些文件操作命令只支持 FAT 格式的文件系统。

1、fatinfo 命令

fatinfo 命令用于查询指定 MMC 指定分区的文件系统信息,格式如下: fatinfo <interface> [<dev[:part]>]



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

interface 表示接口,比如 mmc, dev 是查询的设备号, part 是要查询的分区。比如我们要 查询 SD 卡分区 1 的文件系统信息, 命令如下:

fatinfo mmc 0:1

结果如下图所示:

Zynq> fatin:	fo mmc 0:1 🛑
Interface:	MMC
Device 0:	Vendor: Man 000003 Snr 3fab4801 Rev: 13.12 Prod: SC16G≣ Type: Removable Hard Disk
• -	Capacity: 15193.5 MB = 14.8 GB (31116288 x 512)
Filesystem: Zynq>	FAT32 "boot "

图 11.4.36 SD 卡分区 1 文件系统信息

从上图可以看出,SD卡分区1的文件系统为FAT32格式的。

2、fatls 命令

fatls 命令用于查询 FAT 格式设备的目录和文件信息, 命令格式如下:

```
fatls <interface> [<dev[:part]>] [directory]
```

interface 是要查询的接口,比如 mmc, dev 是要查询的设备号, part 是要查询的分区, directory 是要查询的目录。比如查询 SD 卡分区 1 中的所有的目录和文件,输入命令:

fatls mmc 0:1

结果如下图所示:

Zyng> fatl	s mmc 0:1 🔶
2961576	BOOT.BIN
	.Trash-1000/
4359168	image.ub
2 file(s),	1 dir(s)
Zynq>	

图 11.4.37 SD 卡分区 1 文件查询

从上图可以看出,SD卡的分区1中存放着2个文件:BOOT.BIN和 image.ub,1个目录。

3、fstype 命令

fstype 用于查看 MMC 设备某个分区的文件系统格式,命令格式如下:

fstype <interface> <dev>:<part>

在 6.3.10 小节制作 SD 启动卡时我们将 SD 卡分成两个分区,我们来查看一下这两个分区 的文件系统格式,输入命令:

fstype mmc 0:1 fstype mmc 0:2

结果如下图所示:



图 11.4.38 fstype 命令



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

从上图可以看出,分区1的格式为 fat,分区1用于存放 BOOT.BIN 文件。分区2的格式为 ext4,用于存放 Linux 的根文件系统。

4、fatload 命令

fatload 命令用于将指定的文件读取到 DRAM 中,命令格式如下:

fatload <interface> [<dev[:part]> [<addr> [<filename> [bytes [pos]]]]]

interface 为接口,比如 mmc,dev 是设备号,part 是分区,addr 是保存在 DRAM 中的起始 地址,filename 是要读取的文件名字。bytes 表示读取多少字节的数据,如果 bytes 为 0 或者省 略的话表示读取整个文件。pos 是要读的文件相对于文件首地址的偏移,如果为 0 或者省略的 话表示从文件首地址开始读取。我们将 SD 卡分区 1 中的 BOOT.BIN 文件读取到 DRAM 中的 0X00000000 地址处,命令如下:

fatload mmc 0:1 00000000 BOOT.BIN

操作过程如下图所示:



图 11.4.39 读取过程

从上图可以看出在 181ms 内读取了 2947944 个字节的数据,速度为 15.5MiB/s,速度还是非常快的。

5、fatwrite 命令

fatwirte 命令用于将 DRAM 中的数据写入到 MMC 设备中,命令格式如下:

fatwrite <interface> <dev[:part]> <addr> <filename> <bytes>

interface 为接口,比如 mmc,dev 是设备号,part 是分区,addr 是要写入的数据在 DRAM 中的起始地址,filename 是写入的数据文件名字,bytes 表示要写入多少字节的数据。我们可 以通过 fatwrite 命令在 uboot 中更新 linux 镜像文件和设备树。我们以更新 linux 镜像文件 image.ub 为例,首先将 Petalinux 工程目录 images/linux 下的 image.ub 镜像文件拷贝到 Ubuntu 中的 tftpboot 目录下,petalinux 工程编译完成后 Petalinux 会自动将其拷贝到 tftp 服务器目录 下,如下图所示:

wma@Linux	:/tftpboot\$ ls								
BOOT.BIN boot.scr	pxelinux.cfg rootfs.cpio	<pre>rootfs.cpio.gz.u-boot rootfs.jffs2</pre>	<mark>rootfs.tar.gz</mark> system.bit	u-boot.bin u-boot.elf	vmlinux zImage				
image.ub	rootfs.cpio.gz	rootfs.manifest	system.dtb	uImage	zynq_fsbl.elf				
wmq@Linux:/tftpboot\$ ls -l image.ub									
-rw-rr 1 wmq wmq 11838820 6月 28 10:21 image.ub 🦛 👘 👘 👘 👘 👘 👘									
wmq@Linux	:/tftpboot\$								

图 11.4.40 tftp 服务器目录下的 image.ub 文件

使用命令 tftpboot 将 image.ub 镜像文件下载到 DRAM 的 0X00000000 地址处,命令如下: tftpboot 00000000 image.ub

image.ub 大小为 11838820(0XB4A564)个字节, 接下来使用命令 fatwrite 将其写入到 SD 卡 的分区 1 中, 文件名字为 image.ub, 命令如下:

fatwrite mmc 0:1 00000000 image.ub 0XB4A564

结果如下图所示:



图 11.4.42 查看 SD 第一个分区的文件

11.4.7 EXT 格式文件系统操作命令

uboot 有 ext2 和 ext4 这两种格式的文件系统的操作命令,常用的就四个命令,分别为: ext2load、ext2ls、ext4load、ext4ls 和 ext4write。这些命令的含义和使用与 fatload、fatls 和 fatwrit 一样,只是 ext2 和 ext4 都是针对 ext 文件系统的。比如 ext4ls 命令, SD 卡的分区 2 就 是 ext4 格式的,使用 ext4ls 就可以查询 SD 卡的分区 2 中的文件和目录,输入命令:

结果如下图所示:

Zynq>	ext41s mmc	0:2
<dir></dir>	4096	
<dir></dir>	4096	
<dir></dir>	16384	lost+found
<dir></dir>	4096	var
<dir></dir>	4096	usr
<dir></dir>	4096	sys
<dir></dir>	4096	lib
<dir></dir>	4096	tmp
<dir></dir>	4096	dev
<dir></dir>	4096	mnt
<dir></dir>	4096	etc
<dir></dir>	4096	sbin
<dir></dir>	4096	home
<dir></dir>	4096	run
<dir></dir>	4096	proc
<dir></dir>	4096	bin
<dir></dir>	4096	boot
<dir></dir>	4096	media
Zvnα≥		

图 11.4.43 ext41s 命令

关于 ext 格式文件系统其他命令的操作可参考 11.4.6 小节或者通过 help 命令查看这些命令 的使用帮助信息,这里就不讲解了。

11.4.8 系统引导命令

uboot 的本质工作是引导操作系统如 Linux,所以 uboot 肯定有相关的引导(boot)命令来启动操作系统。常用的跟系统引导有关的命令有: bootm、bootz 和 boot。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

要启动 Linux,需要先将 Linux 镜像文件拷贝到 DRAM 中,如果使用到设备树的话也需要将设备树拷贝到 DRAM 中。可以从 SD 卡中将 Linux 镜像和设备树文件加载到 DRAM,也可以通过 nfs 或者 tftp 协议将 Linux 镜像文件和设备树文件下载到 DRAM 中。无论用哪种方法,只要能将 Linux 镜像和设备树文件存到 DRAM 中就行,然后使用 bootm 命令来启动。

一般更多的是通过 tftp 协议从网络获取,这样调试内核和设备树时极其方便。需要注意的是通过 tftp 协议从网络获取时需要先设置 serverip 变量值为内核和设备树文件所在的 Ubuntu 主机的 IP 地址,如何设置见 11.4.4 节的网络操作命令。另外使用 tftp 协议时需要将对应文件拷贝到 Ubuntu 主机的 tftp 服务器存储文件夹下,如我们在 4.3 节 Ubuntu 系统搭建 tftp 服务器中的/tftpboot 目录,这一步通常我们在编译 Petalinux 工程的时候 Petalinux 工具已经帮我们做好了。如果/tftpboot 目录下没有对应文件,可以将 Petalinux 工程所在目录的 image/linux 目录下对应文件拷贝到/tftpboot 目录中。下面我们学习如何通过网络启动 Linux。

1、bootm 命令

bootm 命令用于启动在内存中的用 mkimage 工具处理过的内核镜像。由于 zynq 使用 image.ub 镜像文件,而 image.ub 镜像文件属于 U-Boot fitImage (参考 18.3 节 image.ub 的来 源),里面通常包括 linux 内核和设备树,所以可以将 image.ub 镜像文件写到 DRAM 中,然 后使用 bootm 命令来启动。启动 Linux 内核的命令如下:

bootm addr

addr是 image.ub 镜像在 DRAM 中的首地址。

现在我们使用 tftp 网络协议来启动 Linux 系统。image.ub 文件可以在/tftpboot 文件夹找到, 如下图所示:

wmq@Linux:/tftpboot\$	ls -l image.ub 🔶
-rw-rr 1 wmq wmq	11838820 6月 28 10:21 image.ub
wmq@Linux:/tftpboot\$	
wma@Linux:/tftpbootS	

图 11.4.44 tftp 服务器目录下的 image 镜像文件

如果没有该文件,可以将 Petalinux 工程所在目录的 image/linux 目录下对应文件拷贝到 /tftpboot 文件夹。万事俱备后我们将 image.ub 下载到 DRAM 的 0x10000000 地址处,然后使用 命令 bootm 启动,命令如下:

tftpboot 10000000 image.ub bootm 10000000

命令运行结果如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

Zynq> tft	tpboot 1	0000000	image	.ub	-	_												
Using eth	hernet@e	000b000	devio	e														
TFTP from	n server	192.16		; ou	ır II	ad? ad	ldre	ss	is	19		68.						
Filename	'image.	ub'.																
Load add:	ress: 0x	1000000	0															
Loading:	#######	#######	######	####	####	****	###	:##	###	###	###	###	###	###	###	###	##:	###
	#######	#######	*****	####	####	;;;;	###	:##:	###	###	###	###	###	###	###	###	##	###
	#######	******	*****	####	####	::::	###	:##:	###	###	###	###	###	###	###	###	##	###
	#######	#######	*****	####	####	;;;;	###	:##:	###	###	###	###	###	###	###	###	##	###
	#######	******	*****	####	####	****	###	:##:	###	###	###	###	###	###	###	###	##:	###
	#######	******	*****	####	####	::::	###	:##:	###	###	###	###	###	###	###	###	##:	###
	#######	******	*****	####	####	****	###	*##	###	###	###	###	###	###	###	###	##:	###
	#######	******	*****	####	####	****	###	:##:	###	###	###	###	###	###	###	###	##:	###
	#######	******	*****	####	####	::::	###	:##:	###	###	###	###	###	###	###	###	##	###
	#######	#######	*****	####	####	::::	###	:##:	###	###	###	###	###	###	###	###	##	###
	#######	******	*****	####	####	;;;;	###	:##:	###	###	###	###	###	###	###	###	##:	###
	#######	#######	*****	####	####	::::	###	:##:	###:	###	###	###	###	###	###	###	##:	###
	#######	******	*****	####	###													
	3.6 MiB	/s																
done																		
Bytes tra	ansferre	d = 118	38820	(b4a	564	hex	:)											
Zyng> boo	otm 1000	0000 🔶																
## Loadin	ng kerne	l from	FIT Im	lage	at :	LOOC	0000	00										
Using	conf@s	vstem-to	op.dtk	' co	nfi	jura	tic	on										
Verify	ving Has	- h Integ	ritv .	o	K.													
Trving	y 'kerne	101' ke:	rnel s	ubim	lage													
Desc	cription	: Linu	x kern	el														
TVD	a:	Kerne	el Ima	are														
Com	oression	: uncor	moress	ed														
Data	a Start:	0 x 10	0000e8															
Data	a Size:	4325	984 By	tes	= 4	.1 M	ſiв											
Arch	hitectur	e: ARM																
os:		Linu	x															
Load	d Addres	s: 0x00	200000															
Entr	ry Point	: 0x00	200000															

图 11.4.45 通过 tftpboot+bootm 启动内核

上图就是我们通过 tftpboot 和 bootm 命令来从网络启动 Linux 系统。

2、bootz 命令

bootz 和 bootm 功能类似,只是 bootz 命令用于启动 zImage 镜像文件,zynq 使用的不多, 了解即可。bootz 命令格式如下:

bootz [addr [initrd[:size]] [fdt]]

命令 bootz 有三个参数, addr 是 Linux zImage 镜像文件在 DRAM 中的位置, initrd 是 initrd 文件在 DRAM 中的地址,这个其实就是前面给大家提到的 INITRAM 根文件系统的在内存中 的地址,如果不使用 initrd 的话使用 '-'代替即可, fdt 就是设备树文件在 DRAM 中的地址。可以在/tftpboot 目录找到 zImage 和 system.dtb 文件,如下图所示:

wmq@Linux:	/tftpboot\$ ls				
BOOT.BIN	pxelinux.cfg	rootfs.cpio.gz.u-boot	rootfs.tar.gz	u-boot.bin	vmlinux
boot.scr	rootfs.cpio	rootfs.jffs2	system.bit	u-boot.elf	zImage 🔶 📂
image.ub	rootfs.cpio.gz	rootfs.manifest	system.dtb	uImage	zynq_fsbl.elf
wmq@Linux:	/tftpboot\$ _			_	

图 11.4.46 zImage 和 systemdtb 文件

使用 tftpboot 命令将 zImage 下载到 DRAM 的 0x00000000 地址处,然后将设备树 system.dtb 下载到 DRAM 中的 0x05000000 地址处,最后使用命令 bootz 启动,命令如下:

tftpboot 00000000 zImage tftpboot 05000000 system.dtb

bootz 00000000 - 05000000

命令运行结果如下图所示:

论坛:www.openedv.com/forum.php

正点原子



图 11.4.47 通过 tftpboot+bootz 命令启动内核

上图就是我们通过 tftpboot 和 bootz 命令来从网络启动 Linux 系统。如果我们要从 SD 中启 动 Linux 系统的话只需要使用命令 fatload 将 zImage 和 system.dtb 从 SD 卡的分区 1 中拷贝到 DRAM 中,然后使用命令 bootz 启动即可, bootm 同理。

3、boot 命令

boot 命令也是用来启动 Linux 系统的,只是 boot 会读取环境变量 bootcmd 来启动 Linux 系 统, bootcmd 是一个很重要的环境变量。其名字分为"boot"和"cmd",也就是"引导"和 "命令",说明这个环境变量保存着引导命令,其实就是启动的命令集合,具体的引导命令 内容是可以修改的。比如我们要想使用 tftpboot 命令从网络启动 Linux 那么就可以设置 bootcmd为"tftpboot 10000000 image.ub;bootm",然后使用 saveenv 将 bootcmd 保存起来。然 后直接输入 boot 命令即可从网络启动 Linux 系统, 命令如下:

setenv bootcmd 'tftpboot 10000000 image.ub;bootm'

boot

运行结果如下图所示:



正点原子

图 11.4.48 设置 bootcmd 从网络启动 Linux

前面说过 uboot 倒计时结束以后就会启动 Linux 系统,其实就是执行的 bootcmd 中的启动 命令。

11.4.9 其他常用命令

uboot 中还有其他一些常用的命令,比如 reset、run 和 mtest 等。

1、reset 命令

reset 命令顾名思义就是复位, 输入"reset"即可复位重启, 如下图所示:



图 11.4.49 reset 命令运行结果 337

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

2、run 命令

run 命令用于运行环境变量中定义的命令,比如可以通过"run bootcmd"来运行 bootcmd 中的启动命令。我们在讲解 bootm 命令时就使用过 "run netboot" 命令以使用 tftp 网络协议下 载 linux 镜像并启动。

run 命令运行环境变量,那么其实就是解析环境变量,将变量的内容解析出来,如果是变 量的内容是可执行的命令那么就运行此命令,所以不是什么环境变量都可以使用 run 来运行 的,必须要求该变量的内容是可执行的命令或其它可运行的逻辑语句。

常用的逻辑语句也就是 if...then...else 等,这些内容就大家自己去看了,这里就不详细介 绍了。

3、mtest 命令

mtest 命令是一个简单的内存读写测试命令,可以用来测试自己开发板上的 DDR,命令格 式如下:

mtest [start [end [pattern [iterations]]]]

start 是要测试的 DRAM 开始地址, end 是结束地址, 比如我们测试 0X0000000~0X00001000 这段内存,输入"mtest 00000000 00001000",结果如下图所示:



图 11.4.50 mtest 命令运行结果

从上图可以看出,测试范围为0X0000000~0X00001000,已经测试了8156次,如果要结 束测试就按下键盘上的"Ctrl+C"键。

至此, uboot 常用的命令就讲解完了, 如果要使用 uboot 的其他命令, 可以查看 uboot 中 的帮助信息,或者上网查询相应的资料。

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

正点原子

第十二章 U-Boot 顶层 Makefile 详解

上一章我们详细的讲解了 uboot 的使用方法,其实就是各种命令的使用,学会 uboot 使用 以后就可以尝试移植 uboot 到自己的开发板上了。在移植之前我们先来分析 uboot 顶层 Makefile 文件,理清 uboot 的编译流程。本章我们使用正点原子提供的 uboot 源码,先了解 uboot 的目录结构,再来分析顶层 Makefile 文件。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

12.1 U-Boot 源码目录分析

12.1.1 U-Boot 工程目录分析

讲解顶层 Makefile 文件之前,我们需要先了解 uboot 源码工程目录结构。此处我们将正点 原子移植好的 uboot 解压到家目录下的 uboot/alientek-uboot-v2020.1,如下图所示:

wmq@Linux:~/uboot/alientek-uboot-v2020.1\$ ls					
api	configs	examples	lib	post	zynq_zc702.sh
arch	disk	fs	Licenses	README	
board	doc	hw-description	linux.tcl	scripts	
cmd	drivers	include	MAINTAINERS	test	
common	dts	Kbuild	Makefile	tools	
config.mk	env	Kconfig	net	uboot.tcl	
wmq@Linux:~/uboot/alientek-uboot-v2020.1\$					

图 12.1.1 解压后的 uboot

上图中的文件夹或文件的含义如下表所示:

类型	名字	描述	备注
	api	与硬件无关的 API 函数	
	arch	与芯片架构体系相关的代码	
	Board	不同板子(开发板)的定制代码	
	cmd	uboot 命令相关代码	
	common	通用代码	
	configs	配置文件	
	disk	磁盘分区相关代码	
	doc	文档	
	drivers	通用设备驱动代码	
	dts	设备树	
文件夹	env	环境变量相关代码	uboot 自带
	examples	示例代码	
	fs	文件系统	
	include	头文件	
	lib	库文件	
	Licenses	许可证相关文件	
	net	网络相关代码	
	post	上电自检程序	
	scripts	脚本文件	
	test	测试代码	
	tools	工具文件夹	
	. checkpatch. conf	路径检查配置文件	uboot 自带
	.gitignore	git 工具相关文件	uboot 自带
	.mailmap	邮件列表	
	config.mk	某个 Makefile 会调用此文件	uboot 自带
文件	Kbuild	用于生成一些和汇编有关的文件	uboot 自带
	Kconfig	图形配置界面描述文件	
	MAINTAINERS	维护者联系方式文件	
	Makefile	顶层 Makefile 文件	
	README	相当于帮助文档	uboot 自带

上表中的很多文件夹和文件我们都不需要去关注,我们要关注的文件夹或文件如下:

1. arch 文件夹



从上图可以看出有很多架构,比如 arm、x86、riscv 等,我们现在用的是 ARM 芯片,所 以只需要关注 arm 文件夹即可,进入 arm 文件夹里面内容如下图所示:

wmq@Linux:~/uboot/alientek-uboot-v2020.1\$ cd arch/arm/					
wmq@Linux:~/ub	oot/alientek-uboo	t-v2020.1/arch/	arm\$ ls		
config.mk	mach-bcm283x	mach-kirkwood	mach-rockchip	mach-uniphier	
сри	mach-bcmstb	mach-mediatek	mach-s5pc1xx	mach-versal	
dts	mach-davinci	mach-meson	mach-snapdragon	mach-versatile	
include	mach-exynos	mach-mvebu	mach-socfpga	mach-zynq	
Kconfig	mach-highbank	mach-omap2	mach-sti	mach-zynqmp	
Kconfig.debug	mach-imx	mach-orion5x	mach-stm32	mach-zynqmp-r5	
lib	mach-integrator	mach-owl	mach-stm32mp	Makefile	
mach-aspeed	mach-k3	mach-qemu	mach-sunxi	thumb1	
mach-at91	mach-keystone	mach-rmobile	mach-tegra		

图 12.1.3 arm 文件夹

mach 开头的文件夹是跟具体的设备有关的,比如"mach-exynos"就是跟三星的 exyons 系列 CPU 有关的文件。我们使用的是 ZYNQ,所以要关注"mach-zynq"这个文件夹。另外 "cpu"这个文件夹也是和 cpu 架构有关的,文件夹内容如下图所示:

wmq@Linux	:~/uboot/al	ientek-ubo	ot-v2020.:	1/arch/arm\$	ls cpu/
arm11	arm720t	arm946es	armv8	sa1100	
arm1136	arm920t	armv7	Makefile	u-boot.lds	
arm1176	arm926ejs	armv7m	рха	u-boot-spl.	lds
wmq@Linux	:~/uboot/al	ientek-ubo	ot-v2020.:	1/arch/arm\$	

图 12.1.4 cpu 文件夹

从上图可以看出有多种 ARM 架构相关的文件夹, ZYNQ 使用的 Cortex-A9 内核, Cortex-A9 属于 armv7,所以我们要关心 "armv7"这个文件夹。cpu 文件夹里面有个名为 "u-boot.lds"的链接脚本文件,这个就是 ARM 类架构处理器所使用的 u-boot 链接脚本文件。armv7 文件夹里面的文件都是跟 ARMV7 架构有关的,是我们分析 uboot 启动源码的时候需要重点关注的。

2. board 文件夹

board 文件夹就是和具体的开发板有关的,打开此文件夹,里面全是不同的板子,borad 文件夹里面有个名为 "xilinx" 的文件夹,里面存放的是 Xilinx 厂商所有支持 uboot 的芯片系 列,如下图所示:

wmq@Linux:~/	uboot/alientek-uboot-	v2020.1	<pre>5 ls board/xilinx/</pre>
bootscripts	Kconfig	versal	zynqmp
common	microblaze-generic	zynq	zynqmp_r5
wmq@Linux:~/	uboot/alientek-uboot-	v2020.1	5

图 12.1.5 xilinx 文件夹

所有使用 xilinx 芯片的板子对应的板级文件都放到此文件夹中。该文件夹下有 3 个文件 夹,这 3 个文件夹对应 3 种类型的开发板。"microblaze-generic"表示使用 microblaze 软核 IP 的 FPGA 开发板、"zynq"表示使用 ZYNQ-7000 系列芯片的开发板、"zynqmp"表示使用



原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php

ZYNQ MP 系列芯片的开发板。正点原子的领航者开发板是 ZYNQ-7000 系列的开发板,我们 后面移植 uboot 的时候就是参考的 Xilinx 官方的开发板,也就是要参考"zynq"这个文件夹来 定义我们的开发板。

看看 zvng 目录下有哪些文件,如下所示:

wmq@Linux:~/	uboot/alientek-uboot-v	2020.1\$ ls board/xilin	ix/zynq
board.c	zynq-altk	zynq-zc702	zynq-zed
bootimg.c	zynq-cc108	zynq-zc706	zynq-zturn
cmds.c	zynq-cse-nand	zynq-zc770-xm010	zynq-zybo
Kconfig	zynq-cse-nor	zynq-zc770-xm011	zynq-zybo-z7
MAINTAINERS	zynq-cse-qspi-single	zynq-zc770-xm011-x16	
Makefile	zynq-dlc20-rev1.0	zynq-zc770-xm012	
xil_io.h	zynq-microzed	zynq-zc770-xm013	
wmq@Linux:~/	uboot/alientek-uboot-v	2020.1\$	

图 12.1.6 zynq 文件夹下的内容

3. configs 文件夹

此文件夹为 uboot 的配置文件, uboot 是可配置的, 但是你要是自己从头开始一个一个项 目的配置,那就太麻烦了,因此一般半导体或者开发板厂商都会制作好一个配置文件。我们 可以在这个做好的配置文件基础上来添加自己想要的功能,这些半导体厂商或者开发板厂商 制作好的配置文件统一命名为"xxx_defconfig", xxx一般表示开发板名字,这些defconfig文 件都存放在 configs 文件夹,因此, Xilinx 官方的开发板配置文件肯定也在这个文件夹中,如 下图所示:

<pre>wmq@Linux:~/uboot/alientek-uboot-v2020.1\$ ls configs/xilinx_*</pre>
configs/xilinx_versal_mini_defconfig
configs/xilinx_versal_mini_emmc0_defconfig
configs/xilinx_versal_mini_emmc1_defconfig
configs/xilinx_versal_mini_ospi_defconfig
configs/xilinx_versal_mini_qspi_defconfig
configs/xilinx_versal_virt_defconfig
configs/xilinx_zynqmp_mini_defconfig
configs/xilinx_zynqmp_mini_emmc0_defconfig
configs/xilinx_zynqmp_mini_emmc1_defconfig
configs/xilinx_zynqmp_mini_nand_defconfig
configs/xilinx_zynqmp_mini_nand_single_defconfig
configs/xilinx_zynqmp_mini_qspi_defconfig
configs/xilinx_zynqmp_r5_defconfig
configs/xilinx_zynqmp_virt_defconfig
configs/xilinx_zynq_virt_defconfig
wmq@Linux:~/uboot/alientek-uboot-v2020.1\$

图 12.1.7 Xilinx 芯片系列开发板配置文件

上图中的以"xilinx"开头的文件都是 xilinx 芯片系列的配置文件。

4. Makefile 文件

这个是顶层 Makefile 文件, Makefile 是支持嵌套的, 也就是顶层 Makefile 可以调用子目 录中的 Makefile 文件。Makefile 嵌套在大项目中很常见,一般大项目里面所有的源代码都不 会放到同一个目录中,各个功能模块的源代码都是分开的,各自存放在各自的目录中。每个 功能模块目录下都有一个 Makefile, 这个 Makefile 只处理本模块的编译链接工作, 这样所有 的编译链接工作就不用全部放到一个 Makefile 中,可以使得 Makefile 变得简洁明了。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

uboot 源码根目录下的 Makefile 是顶层 Makefile,它会调用其它的模块的 Makefile 文件,比如 drivers/cpu/Makefile。当然了,顶层 Makefile 要做的工作可远不止调用子目录 Makefile 这么简单。

5. README

README 文件描述了 uboot 的详细信息,包括 uboot 该如何编译、uboot 中各文件夹的含义、相应的命令等等。建议大家详细的阅读此文件,可以进一步增加对 uboot 的认识。关于 uboot 根目录中的文件和文件夹的含义就讲解到这里。

12.1.2 VScode 创建 uboot 工程

为了方便查看 uboot 源码,我们使用 VScode 创建 uboot 工程。打开 VScode,选择:文件 ->打开文件夹...,选中 uboot 文件夹,如下图所示:

Car	ncel	打开文件夹	٩	ОК
\odot	最近使用	▲ 🛱 wmq uboot 🕨		2
ŵ	主目录	名称 1	大小	修改日期
	桌面	🖻 alientek-uboot-v2020.1		08:27
H	视频			
٥	图片			
۵	文档			
÷	下载			
5	音乐			
	boot			
+	其他位置			

图 12.1.8 选择 U-Boot 源码目录

打开 uboot 目录以后, VSCode 界面如下图所示:



图 12.1.9 U-Boot 源码 VSCode 工程



原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php

点击"文件->将工作区另存为...",打开保存工作区对话框,将工作区保存到 uboot 源码 根目录下,设置文件名为"uboot_vscode",如下图所示:



图 12.1.10 保存工作区

保存成功以后就会在 uboot 源码根目录下存在一个名为 uboot_vscode.code-workspace 的文件。这样一个完整的 VSCode 工程就建立起来了。但是这个 VSCode 工程包含了 uboot 的所有文件, uboot 中有些文件是不需要的, 比如 arch 目录下是各种架构的文件夹, 如下图所示:

						_vscode	(工作区) -	Visual	Studio (ode:
文件(F)	编辑(E)	选择(S)	查看(V)	转到(G)	调试(D)	终端(T)	帮助(H)			
ð	资源管理器	물								
	▲ 打开的编辑	器								
Ω	▲ UBOOT_V	SCODE (I	作区)							
~	🕨 👩 .gi	thub								
88	🕨 🛤 ap									
x	🔺 📹 ar	ch								
\sim	🕨 🖬 a	rc								
8	🕨 📹 a	rm								
	🕨 📹 n	n68k								
	🕨 📹 n	nicroblaze								
	🕨 📫 n	nips								
	🕨 📹 n	ds32								
	🕨 📹 n	ios2								
	🕨 📹 p	owerpc								

图 12.1.11 arch 目录

在 arch 目录下,我们只需要 arm 文件夹,所以需要将其它的目录从 VSCode 中给屏蔽掉, 比如将 arch/nios2 这个目录给屏蔽掉,在 VSCode 上建名为".vscode"的文件夹,如下图所示:



图 12.1.12 新建.vscode 文件夹

输入新建文件夹的名字,完成以后如下图所示。



图 12.1.13 .vscode 文件创建完成

在.vscode 文件夹中新建一个名为"settings.json"的文件,然后在 settings.json 中输入如下内

```
示例代码 settings.json 文件代码
1 {
2
    "search.exclude": {
3
       "**/node_modules": true,
4
       "**/bower_components": true,
5
    },
6
    "files.exclude": {
       "**/.git": true,
7
8
       "**/.svn": true,
9
       "**/.hg": true,
10
       "**/CVS": true,
       "**/.DS_Store": true,
11
12 }
13 }
```

结果如下图所示:

容:

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

正点原子



图 12.1.14 settings. json 文件内容

其中"search.exclude"里面是需要在搜索结果中排除的文件或者文件夹,"files.exclude"是 左侧工程目录中需要排除的文件或者文件夹。我们需要将 arch/nios2 文件夹下的所有文件从搜 索结果和左侧的工程目录中都排除掉,因此在"search.exclude"和"files.exclude"中输入如下图 所示内容:

<pre>{} setting</pre>	s.json ×
🔰 aliente	ek_uboot > .vscode > { } settings.json >
1	[
	"search.exclude": {
	"**/node_modules": true,
	"**/bower_components": true,
	"arch/nios2":true
	},
	"files.exclude": {
	"**/.git": true,
	"**/.svn": true,
10	"**/.hg": true,
11	"**/CVS": true,
12	"**/.DS Store": true,
13	"arch/nios2":true
14	}
15	}

图 12.1.15 添加 arch/nios2 排除选项

此时再看一下左侧的工程目录,发现 arch 目录下没有 nios2 这个文件夹了,说明 nios2 这 个文件夹被排除掉了,如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php 📹 arch 📹 arc 📫 arm m68k microblaze mips nds32 powerpc sandbox x86 🛾 xtensa 🚯 .gitignore Kconfig 图 12.1.16 arch/nios2 目录已被排除

我们只是在"search.exclude"和"files.exclude"中加入了"arch/nios2": true, 冒号前面的是要排除的文件或者文件夹, 冒号后面为是否将文件排除, true 表示排除, false 表示不排除。用这种方法即可将不需要的文件, 或者文件夹排除掉, 对于本章我们分析 uboot 而言, 在 "search.exclude"和"files.exclude"中需要输入的内容如下:

示例代码 settings.json 文件代码

"**/*.o": true, "**/*.su": true, "**/*.cmd": true, "arch/arc": true, "arch/m68k": true, "arch/microblaze": true, "arch/mips": true, "arch/nds32": true, "arch/nios2": true, "arch/powerpc": true, "arch/sandbox": true, "arch/sh": true, "arch/xtensa": true, "arch/x86": true, "arch/arm/mach*": true, "arch/arm/mach-zynq": false, "arch/arm/cpu/arm11*": true, "arch/arm/cpu/arm720t": true, "arch/arm/cpu/arm9*": true, "arch/arm/cpu/armv7m": true, "arch/arm/cpu/armv8": true, "arch/arm/cpu/pxa": true, "arch/arm/cpu/sa1100": true,



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

"board/[a-w]*": true,
"board/[y-z]*": true,
"board/[0-9]*": true,
"board/[A-Z]*": true,
"board/xe*": true,
"board/xilinx/m*": true,
"configs/[a-y]*": true,
"configs/[0-9]*": true,

上述代码用到了通配符 "*",比如 "**/*.o"表示所有.o 结尾的文件。"configs/[a-y]*" 表示 configs 目录下所有以 'a' ~ 'y'开头的文件或者文件夹。上述配置只是排除了一部分 文件夹,大家在实际的使用中可以根据自己的实际需求来选择将哪些文件或者文件夹排除掉。 排除以后我们的工程就会清爽很多,搜索的时候也不会跳出很多文件了。

12.2 U-Boot 顶层 Makefile 分析

在阅读 uboot 源码之前,肯定要先看一下项层 Makefile,分析 gcc 编译代码的时候一定是 先从项层 Makefile 开始,然后是子 Makefile,这样通过层层分析 Makefile 即可了解整个工程 的组织结构。项层 Makefile 也就是 uboot 根目录下的 Makefile 文件,由于项层 Makefile 文件 内容比较多,所以我们将其分开来看。

12.2.1 版本号

顶层 Makefile 一开始是版本号,内容如下(为了方便分析,顶层 Makefile 代码段前段行号 采用 Makefile 中的行号,因为 uboot 会更新,因此行号可能会与你所看的顶层 Makefile 有所不 同):

- 3 VERSION = 2020
- 4 PATCHLEVEL = 01
- 5 SUBLEVEL =
- 6 EXTRAVERSION =
- 7 NAME =

VERSION 是主版本号, PATCHLEVEL 是补丁版本号, SUBLEVEL 是次版本号, 这三个 一起构成了 uboot 的版本号, 比如当前的 uboot 版本号就是"2020.01", 没有次版本号。 EXTRAVERSION 是附加版本信息, NAME 是和名字有关的, 一般不使用这两个。

12.2.2 MAKEFLAGS 变量

make 是支持递归调用的,也就是在 Makefile 中使用"make"命令来执行其他的 Makefile 文件,一般都是子目录中的 Makefile 文件。假如在当前目录下存在一个"subdir"子目录,这个子目录中又有其对应的 Makefile 文件,那么这个工程在编译的时候其主目录中的 Makefile 就可以调用子目录中的 Makefile,以此来完成所有子目录的编译。

有时候我们需要向子 make 传递变量,这个时候使用 "export"来导出要传递给子 make 的 变量即可,如果不希望哪个变量传递给子 make 的话就使用 "unexport"来声明不导出:

export VARIABLE //导出变量给子 make。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

unexport VARIABLE..... //不导出变量给子 make。

有两个特殊的变量: "SHELL"和"MAKEFLAGS",这两个变量除非使用"unexport" 声明,否则的话在整个 make 的执行过程中,它们的值始终自动的传递给子 make。在 uboot 的 主 Makefile 中有如下代码:

18 MAKEFLAGS += -rR --include-dir=\$(CURDIR)

上述代码使用 "+=" 来给变量 MAKEFLAGS 追加了一些值, "-rR" 表示禁止使用内置 的隐含规则和变量定义, "--include-dir" 指明搜索路径, "\$(CURDIR)"表示当前目录。

12.2.3 命令输出

为了更好的体会下面讲解的内容,我们在讲解命令输出之前先介绍下不使用 Petalinux 时 通常的编译 uboot 方式,在传统的嵌入式 Linux 开发方式中,一般都是直接编译 uboot 源码, 而不像我们前面章节中使用了 petalinux 来编译 Uboot。在不使用 Petalinux 的情况下,可以使 用如下的 make 命令来编译 uboot (需要提醒的是在使用下面的命令前先在 ubuntu 终端运行该 命令"./opt/petalinux/2020.2/environment-setup-cortexa9t2hf-neon-xilinx-linux-gnueabi",否则终 端将不认识交叉编译器而报错),然后输入下面三条语句编译 uboot 源码。

make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi- distclean

make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi- atk_7020_defconfig

make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi- -j8

这三条命令中"ARCH=arm"设置目标为 arm 架构, CROSS_COMPILE 指定所使用的交 叉编译器,只需要指明编译器前缀就行了,比如 arm-xilinx-linux-gnueabi-gcc 编译器的前缀就 是"arm-xilinx-linux-gnueabi-"。第一条命令相当于"make distclean",目的是清除工程,一 般在第一次编译的时候最好清理一下工程。第二条指令相当于"make atk_7020_defconfig", 用于配置 uboot,配置文件为 atk_7020_defconfig,这里笔者是以领航者 7020 为例,如果大家 使用的是领航者 7010 开发板,则对应的配置为 atk_7010_defconfig,大家根据自己所使用的开 发板进行设置。uboot 支持其它的架构和外设,比如 USB、网络、SD 卡等。这些都是可以配 置的,需要什么功能就使能什么功能,所以在编译 uboot之前,可以根据自己的需求对配置文 件进行修改,atk_7020_defconfig和 atk_7010_defconfig两个配置文件便是正点原子针对自家的 开发板配置所对应的配置文件,当然也可以使用图形化界面配置,这个后面会讲,这个配置 文件在 U-Boot 源码根目录下的 configs 目录中。最后一条指令相当于"make -j8"也就是使用 8 核来编译 uboot。"-j"参数用于设置主机使用多少个核来编译 uboot,设置的核越多,编译 速度越快。-j8 表示使用 8 个核编译 uboot,具体设置多少个要根据自己的虚拟机或者电脑配 置,如果你给 VMware 分配了 4 个核,那么最多只能使用-j4。

当这三条命令执行完以后 uboot 也就编译成功了,如下图所示:

原子哥在线教学:v	www.yuanzige.com	论坛:www.openedv.com/forum.php
CC	spl/lib/hexdu	יp.o
LD	spl/drivers/m	td/spi/spi-nor.o
LD	spl/drivers/m	td/spi/built-in.o
CC	spl/lib/uuid.o	D
LD	spl/drivers/m	td/built-in.o
LD	spl/drivers/b	uilt-in.o
CC	spl/lib/rand.c	0
CC	spl/lib/panic	.0
CC	spl/lib/tiny-	printf.o
CC	spl/lib/strto	.0
CC	spl/lib/date.c	0
LD	spl/lib/built	-in.o
LD	spl/u-boot-sp	l
OBJ	COPY spl/u-boot-sp	l-nodtb.bin
CAT	spl/u-boot-sp	l-dtb.bin
COP	Y spl/u-boot-sp	l.bin
MKI	MAGE spl/boot.bin	
CFG	CHK u-boot.cfg	
wmq@L	inux:~/uboot/alient	tek-uboot-v2020.1\$

正点原子

图 12.2.1 U-Boot 编译成功

可以看到 uboot 默认编译是不会在终端中显示完整的命令,都是短命令,很简洁。

在终端中输出短命令虽然看起来很清爽,但是不利于分析 uboot 的编译过程。可以通过设置变量"V=1"来实现完整的命令输出,也就是使用命令"make V=1 ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi--j8",这个在调试 uboot 的时候很有用,结果如下图所示:



图 12.2.2 显示 U-Boot 完整编译信息

顶层 Makefile 中控制命令输出的代码如下:

示例代码 顶层 Makefile 代码

```
90 ifeq ("$(origin V)", "command line")
91 KBUILD_VERBOSE = $(V)
92 endif
93 ifndef KBUILD_VERBOSE
94 KBUILD_VERBOSE = 0
95 endif
96
97 ifeq ($(KBUILD_VERBOSE),1)
98 quiet =
99 Q =
100 else
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

101 quiet=quiet_

102 Q = @

103 endif

上述代码中先使用 ifeq 来判断"\$(origin V)"和"command line"是否相等。这里用到了 Makefile 中的函数 origin, origin 和其他的函数不一样,它不操作变量的值, origin 用于告诉你 变量是哪来的, 语法为:

\$(origin <variable>)

variable 是变量名, origin 函数的返回值就是变量来源, 因此\$(origin V)就是变量 V 的来 源。如果变量 V 是在命令行定义的那么它的来源就是"command line",这样"\$(origin V)"和 "command line"就相等了。当这两个相等的时候变量 KBUILD_VERBOSE 就等于 V 的值,比 如在命令行中输入"V=1"的话那么 KBUILD_VERBOSE=1。如果没有在命令行输入 V 的话 KBUILD VERBOSE=0.

第 97 行判断 KBUILD VERBOSE 是否为 1,如果 KBUILD VERBOSE 为 1 的话变量 quiet 和Q都为空,如果KBUILD_VERBOSE=0的话变量quiet为"quiet_",变量Q为"@",综上 所述: V=1 的话:

KBUILD_VERBOSE=1

quiet= 空

Q= 空

V=0或者命令行不定义 V 的话:

KBUILD_VERBOSE=0

quiet= quiet_

Q= @

Makefile 中会用到变量 quiet 和 Q 来控制编译的时候是否在终端输出完整的命令,在顶层 Makefile 中有很多如下所示的命令:

\$(Q)\$(MAKE) \$(build)=tools

如果 V=0 的话上述命令展开就是"@ make \$(build)=tools", make 在执行的时候默认会 在终端输出命令,但是在命令前面加上"@"就不会在终端输出命令了。当 V=1 的时候 Q 就 为空,上述命令就是"make \$(build)=tools",因此在 make 执行的过程,命令会被完整的输出 在终端上。

有些命令会有两个版本,比如:

quiet_cmd_sym ?= SYM \$@

cmd_sym ?= \$(OBJDUMP) -t \$<>\$@

sym 命令分为 "quiet_cmd_sym" 和 "cmd_sym" 两个版本,这两个命令的功能都是一样 的,区别在于 make 执行的时候输出的命令不同。quiet_cmd_xxx 命令输出信息少,也就是短 命令,而 cmd xxx 命令输出信息多,也就是完整的命令。

如果变量 quiet 为空的话,整个命令都会输出。

如果变量 quiet 为 "quiet_"的话, 仅输出短版本。

如果变量 quiet 为 "silent_" 的话, 整个命令都不会输出。

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

正点原子

12.2.4 静默输出

上一小节讲了,设置 V=0 或者在命令行中不定义 V 的话,编译 uboot 的时候终端中显示 的短命令,但是还是会有命令输出,有时候我们在编译 uboot 的时候不需要输出命令,这个时 候就可以使用 uboot 的静默输出功能。编译的时候使用"make -s"即可实现静默输出,顶层 Makefile 中相应的代码如下:

示例代码 顶层 Makefile 代码

- 105 # If the user is running make -s (silent mode), suppress echoing of 106 # commands 107 108 ifneq (\$(filter 4.%,\$(MAKE_VERSION)),) # make-4 109 ifneq (\$(filter %s ,\$(firstword x\$(MAKEFLAGS))),) 110 quiet=silent
- 111 endif
- 112 **else** # make-3.8x
- 113 ifneq (\$(filter s% -s%,\$(MAKEFLAGS)),)
- 114 quiet=silent_
- 115 endif
- 116 endif
- 117

118 export quiet Q KBUILD_VERBOSE

第 108 行判断当前正在使用的编译器版本号是否为 4.x,判断 \$(filter 4.%, \$(MAKE VERSION))和""(空)是否相等,如果不相等的话就成立,执行里面的语句。 也就是说\$(filter 4.%,\$(MAKE_VERSION))不为空的话条件就成立,这里用到了 Makefile 中的 filter 函数,这是个过滤函数,函数格式如下:

\$(filter <pattern...>,<text>)

filter 函数表示以 pattern 模式过滤 text 字符串中的单词, 仅保留符合模式 pattern 的单词, 可以有多个模式。函数返回值就是符合 pattern 的字符串。因此 \$(filter 4.%,\$(MAKE_VERSION))的含义就是在字符串"MAKE_VERSION"中找出符合"4.%"的字 符(%为通配符), MAKE VERSION 是 MAKE 工具的版本号,我们当前使用的 MAKE 工具版 本号为 4.1 (make -v 命令查看),所以肯定可以找出"4.%"。因此\$(filter 4.%,\$(MAKE_VERSION))不为空,条件成立,执行109~110行的语句。

第 109 行也是一个判断语句,如果\$(filter %s,\$(firstword x\$(MAKEFLAGS)))不为空的话 条件成立,变量 quiet 等于"silent_"。这里也用到了函数 filter,在 \$(firstword x\$(MAKEFLAGS)))中过滤出符合"%s"的单词。到了函数 firstword,函数 firstword 是获取首 单词,函数格式如下:

\$(firstword <text>)

firstword 函数用于取出 text 字符串中的第一个单词,函数的返回值就是获取到的单词。当 使用"make -s"编译的时候,"-s"会作为 MAKEFLAGS 变量的一部分传递给 Makefile。在 顶层 Makefile 中添加如下图所示的代码:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

107	
108	<pre>ifneq (\$(filter 4.%,\$(MAKE_VERSION)),) # make-4</pre>
109	<pre>ifneq (\$(filter %s ,\$(firstword x\$(MAKEFLAGS))),)</pre>
110	<pre>quiet=silent_</pre>
111	endif
112	else # make-3.8x
113	<pre>ifneq (\$(filter s% -s%,\$(MAKEFLAGS)),)</pre>
114	<pre>quiet=silent_</pre>
115	endif
116	endif
117	·····································
118	export quiet Q KBUILD_VERBOSE
119	
120	mytest:
121	<pre>@echo 'firstword=' \$(firstword x\$(MAKEFLAGS))</pre>

图 12.2.3 添加 mytest 目标

上图中的两行代码用于输出\$(firstword x\$(MAKEFLAGS))的结果,最后执行命令"make-s mytest",结果如下图所示:



图 12.2.4 修改顶层 Makefile 后的执行结果

从上图可以看出第一个单词是 "xrRs", 将\$(filter %s, \$(firstword x\$(MAKEFLAGS)))展 开就是\$(filter %s, xrRs), 而\$(filter %s, xrRs)的返回值肯定不为空, 条件成立, quiet=silent_。 第 118 行使用 export 导出变量 quiet、Q 和 KBUILD_VERBOSE。

12.2.5 设置编译结果输出目录

uboot可以将编译出来的目标文件输出到单独的目录中,在make的时候使用"O"来指定输出目录,比如"make O=out"就是设置目标文件输出到 out 目录中。这么做是为了将源文件和编译产生的文件分开,当然也可以不指定 O 参数,不指定的话源文件和编译产生的文件都在同一个目录内,一般我们不指定 O 参数。顶层 Makefile 中相关的代码如下:

示例代码 顶层 Makefile 代码

120 # kbuild supports saving output files in a separate directory.

121 # To locate output files in a separate directory two syntaxes are supported.

122 # In both cases the working directory must be the root of the kernel src.

123 # 1) O=

124 # Use "make O=dir/to/store/output/files/"

125 #

126 # 2) Set KBUILD_OUTPUT

127 # Set the environment variable KBUILD_OUTPUT to point to the directory

128 # where the output files shall be placed.

129 # export KBUILD_OUTPUT=dir/to/store/output/files/

130 # make

131 #

132 # The O= assignment takes precedence over the KBUILD_OUTPUT environment

134

138

144

148

151

157

.....

```
F点原子
原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php
    133 # variable.
    135 # KBUILD_SRC is set on invocation of make in OBJ directory
    136 # KBUILD_SRC is not intended to be used by the regular user (for now)
    137 ifeq ($(KBUILD_SRC),)
    139 # OK, Make called in directory where kernel src resides
    140 # Do we want to locate output files in a separate directory?
    141 ifeq ("$(origin O)", "command line")
    142 KBUILD OUTPUT := (O)
    143 endif
    145 # That's our default target when none is given on the command line
    146 PHONY := _all
    147 _all:
    149 # Cancel implicit rules on top Makefile
    150 $(CURDIR)/Makefile Makefile: ;
    152 ifneq ($(KBUILD_OUTPUT),)
    153 # Invoke a second make in the output directory, passing relevant variables
    154 # check that the output directory actually exists
    155 saved-output := $(KBUILD_OUTPUT)
    156 KBUILD_OUTPUT := $(shell mkdir -p $(KBUILD_OUTPUT) && cd $(KBUILD_OUTPUT) \
                         && /bin/pwd)
    172 endif # ifneq ($(KBUILD_OUTPUT),)
```

```
173 endif # ifeq ($(KBUILD_SRC),)
```

第 141 行判断"O"是否来自于命令行,如果来自命令行的话条件成立, KBUILD_OUTPUT 就为\$(O),因此变量 KBUILD_OUTPUT 就是输出目录。

第152行判断 KBUILD OUTPUT 是否为空。

第 156 行调用 mkdir 命令, 创建 KBUILD_OUTPUT 目录,并且将创建成功以后的绝对路 径赋值给 KBUILD_OUTPUT。至此,通过 O 指定的输出目录就存在了。

12.2.6 代码检查

uboot 支持代码检查,使用命令"make C=1"使能代码检查,检查那些需要重新编译的文 件。"make C=2"用于检查所有的源码文件,顶层 Makefile 中的代码如下:

示例代码 顶层 Makefile 代码

183 # Call a source code checker (by default, "sparse") as part of the

184 # C compilation.

185 #



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php 186 # Use 'make C=1' to enable checking of only re-compiled files. 187 # Use 'make C=2' to enable checking of *all* source files, regardless 188 # of whether they are re-compiled or not. 189 # 190 # See the file "doc/sparse.txt" for more details, including 191 # where to get the "sparse" utility. 192 193 ifeq ("\$(origin C)", "command line") 194 KBUILD_CHECKSRC = \$(C) 195 endif 196 ifndef KBUILD_CHECKSRC 197 KBUILD_CHECKSRC = 0 198 endif

第 193 行判断 C 是否来源于命令行,如果 C 来源于命令行,那就将 C 赋值给变量 KBUILD_CHECKSRC,如果命令行没有 C 的话 KBUILD_CHECKSRC 就为 0。

12.2.7 模块编译

在 uboot 中允许单独编译某个模块,使用命令 "make M=dir"即可,旧语法 "make SUBDIRS=dir"也是支持的。顶层 Makefile 中的代码如下:

```
示例代码 顶层 Makefile 代码
```

200 # Use make M=dir to specify directory of external module to build

201 # Old syntax make ... SUBDIRS=\$PWD is still supported

202 # Setting the environment variable KBUILD_EXTMOD take precedence

203 ifdef SUBDIRS

```
204 KBUILD_EXTMOD ?= $(SUBDIRS)
```

205 endif

206

207 ifeq ("\$(origin M)", "command line")

```
208 KBUILD_EXTMOD := $(M)
```

209 endif

```
210
```

211 # If building an external module we do not care about the all: rule

```
212 # but instead _all depend on modules
```

213 PHONY += all

```
214 ifeq ($(KBUILD_EXTMOD),)
```

215 _all: all

216 else

217 _all: modules

```
218 endif
```

219

```
220 ifeq ($(KBUILD_SRC),)
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

```
221
        # building in the source tree
222
        srctree := .
223 else
224
        ifeq ($(KBUILD_SRC)/,$(dir $(CURDIR)))
225
             # building in a subdirectory of the source tree
226
             srctree := ..
227
        else
228
             srctree := $(KBUILD_SRC)
229
        endif
230 endif
231 objtree := .
232 src := $(srctree)
233 obj := $(objtree)
234
235 VPATH
               := $(srctree)$(if $(KBUILD_EXTMOD),:$(KBUILD_EXTMOD))
236
237 export srctree objtree VPATH
```

第 203 行判断是否定义了 SUBDIRS,如果定义了 SUBDIRS,变量 KBUILD_EXTMOD=SUBDIRS,这里是为了支持老语法 "make SUBIDRS=dir"

第207行判断是否在命令行定义了M,如果定义了的话KBUILD_EXTMOD=\$(M)。

第 214 行判断 KBUILD_EXTMOD 是否为空,如果为空的话目标_all 依赖 all,因此要先 编译出 all。否则的话默认目标_all 依赖 modules,要先编译出 modules,也就是编译模块。一 般情况下我们不会在 uboot 中编译模块,所以此处会编译 all 这个目标。

第 220 行判断 KBUILD_SRC 是否为空,如果为空的话就设置变量 srctree 为当前目录,即 srctree 为".",一般不设置 KBUILD_SRC。

第231行设置变量 objtree 为当前目录。

第 232 和 233 行分别设置变量 src 和 obj,都为当前目录。

第 235 行设置 VPATH。

第 237 行导出变量 scrtree、objtree 和 VPATH。

12.2.8 获取主机架构和系统

接下来顶层 Makefile 会获取主机架构和系统,也就是我们电脑的架构和系统,代码如下:示例代码 顶层 Makefile 代码

244 HOSTARCH := (shell uname -m |)

245 sed -e s/i.86/x86/ \

- -e s/sun4u/sparc64/
- 247 -e s/arm.*/arm/∖
- 248 -e s/sa110/arm/∖
- 249 -e s/ppc64/powerpc/ \
- 250 -e s/ppc/powerpc/ \
- 251 -e s/macppc/powerpc/∖



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

-e s/sh.*/sh/) 252

253

254 HOSTOS := \$(shell uname -s | tr '[:upper:]' '[:lower:]' |\

255 sed -e 's/\(cygwin\).*/cygwin/')

256

257 export HOSTARCH HOSTOS

第 244 行定义了一个变量 HOSTARCH,用于保存主机架构,这里调用 shell 命令 "uname -m"获取主机 CPU 体系架构的名称,结果如下图所示:

wmg@Linux:~/uboot/alientek-uboot-v2020.1\$ uname -m x86 64 wmg@Linux:~/uboot/alientek-uboot-v2020.1\$ wmq@Linux:~/uboot/alientek-uboot-v2020.1\$

图 12.2.5 获取主机架构类型

从上图可以看出当前电脑主机架构为"x86_64", shell 中的"|"表示管道, 意思是将左 边的输出作为右边的输入, sed -e 是替换命令, "sed -e s/i.86/x86/"表示将管道输入的字符串 中的"i.86"替换为"x86",其他的"sed-s"命令同理。对于笔者的电脑而言, HOSTARCH=x86 64.

第 254 行定义了变量 HOSTOS,此变量用于保存主机操作系统 OS 的值,先使用 shell 命 令 "name-s" 可以获取主机 OS, 结果如下图所示:

<pre>wmq@Linux:~/uboot/alientek-uboot-v2020.1\$ uname -s</pre>
Linux
wmq@Linux:~/uboot/alientek-uboot-v2020.1\$
wmq@Linux:~/uboot/alientek-uboot-v2020.1\$

图 12.2.6 获取主机 OS

从上图可以看出笔者主机的 OS 为"Linux",使用管道将"Linux"作为后面"tr '[:upper:]' '[:lower:]'"的输入, "tr '[:upper:]' '[:lower:]'" 表示将所有的大写字母替换为小写字 母,因此得到"linux"。最后同样使用管道,将"linux"作为"sed -e 's/(cygwin/).*/cygwin/"" 的输入,用于将 cygwin.*替换为 cygwin。因此,HOSTOS=linux。

第257行导出 HOSTARCH=x86_64, HOSTOS=linux。

12.2.9 设置目标架构、交叉编译器和配置文件

编译 uboot 的时候需要设置目标板架构和交叉编译器, " make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi-"就是用于设置 ARCH 和 CROSS_COMPILE,在 顶层 Makefile 中代码如下:

示例代码 顶层 Makefile 代码

261 # set default to nothing for native builds 262 ifeq (\$(HOSTARCH),\$(ARCH)) 263 CROSS COMPILE ?= 264 endif 265 266 KCONFIG_CONFIG ?= .config



原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php

267 export KCONFIG_CONFIG

第 262 行判断 HOSTARCH 和 ARCH 这两个变量是否相等, 主机架构(变量 HOSTARCH) 是 x86_64, 而我们编译的是 ARM 版本 uboot, 肯定不相等, 所以需要设置 CROS_COMPILE= arm-xilinx-linux-gnueabi-。从上面的示例代码可以看出,手动编译的话每次编译 uboot 的时候 都要在 make 命令后面设置 ARCH 和 CROS COMPILE, 使用起来很麻烦, 可以直接修改顶层 Makefile, 在里面加入 ARCH 和 CROSS_COMPILE 的定义, 如下图所示:

<pre>261 # set default to nothing for native builds 262 ifeq (\$(HOSTARCH),\$(ARCH)) 263 CROSS_COMPILE ?= 264 endif</pre>
265
266 ARCH ?= arm
<pre>267 CROSS_COMPILE ?= arm-xilinx-linux-gnueabi-</pre>
268
269 KCONFIG_CONFIG ?= .config
270 export KCONFIG_CONFIG
271

图 12.2.7 定义 ARCH 和 CROSS COMPILE

按照上图所示,直接在顶层 Makefile 里面定义 ARCH 和 CROSS COMPILE,这样就不用 每次编译的时候都要在 make 命令后面定义 ARCH 和 CROSS COMPILE。因为我们使用 Petalinux 工具, 使用 Petalinux 工具编译 uboot 很方便, 所以就不需要在顶层 Makefile 里面定 义 ARCH 和 CROSS_COMPILE。

第 269 行定义变量 KCONFIG_CONFIG, uboot 是可以配置的,这里设置配置文件 为.config。.config默认是没有的,需要使用命令"make xxx_defconfig"对uboot进行配置,配 置完成以后就会在 uboot 根目录下生成.config。默认情况下.config 和 xxx_defconfig 内容是一样 的,因为.config 就是从 xxx_defconfig 复制过来的。如果后续自行调整了 uboot 的一些配置参 数,那么这些新的配置参数就添加到了.config 中,而不是 xxx_defconfig。相当于 xxx_defconfig 只是一些初始配置,而.config 里面的才是实时有效的配置。

12.2.10 调用 scripts/Kbuild.include

主 Makefile 会调用 scripts/Kbuild.include 这个文件,顶层 Makefile 中代码如下:

示例代码 顶层 Makefile 代码

379 # We need some generic definitions (do not try to remake the file).

380 scripts/Kbuild.include: ;

381 include scripts/Kbuild.include

该段代码中使用"include"包含了文件 scripts/Kbuild.include,此文件里面定义了很多变 量,如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

```
1 ####
 2 # kbuild: Generic definitions
 3
  # Convenient variables
 4
           := ;
 5
  сотта
  quote
           :=
              .
   squote
  empty
           :=
  space
           := $(empty) $(empty)
  pound := \#
10
11
12 ###
13 # Name of target with a '.' as filename prefix. foo/bar.o => foo/.bar.o
14 dot-target = $(dir $@).$(notdir $@)
15
16 ###
17 # The temporary file to save gcc -MD generated dependencies must not
18 # contain a comma
19 depfile = $(subst $(comma),_,$(dot-target).d)
20
21 ###
22 # filename of target with directory and extension stripped
23 basetarget = $(basename $(notdir $@))
```

图 12.2.8 Kbuild. include 文件部分内容

在 uboot 的编译过程中会用到 scripts/Kbuild.include 中的这些变量,后面用到的时候再分析。

12.2.11 交叉编译工具变量设置

上面我们只是设置了 CROSS_COMPILE 的名字,但是交叉编译器其他的工具还没有设置,顶层 Makefile 中相关代码如下:

```
示例代码 顶层 Makefile 代码
383 # Make variables (CC, etc...)
384
385 AS
        = $(CROSS_COMPILE)as
386 # Always use GNU ld
387 ifneq ($(shell $(CROSS_COMPILE)ld.bfd -v 2>/dev/null),)
388 LD
        = $(CROSS_COMPILE)ld.bfd
389 else
390 LD
        = $(CROSS_COMPILE)ld
391 endif
392 CC = $(CROSS COMPILE)gcc
393 \text{ CPP} = (CC) - E
394 AR = $(CROSS COMPILE)ar
395 NM = $(CROSS_COMPILE)nm
396 LDR = $(CROSS COMPILE)ldr
397 STRIP = $(CROSS_COMPILE)strip
398 OBJCOPY = $(CROSS_COMPILE)objcopy
399 OBJDUMP = $(CROSS_COMPILE)objdump
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

12.2.12 导出其他变量

接下来在顶层 Makefile 会导出很多变量,代码如下:

示例代码 顶层 Makefile 代码

429 export VERSION PATCHLEVEL SUBLEVEL UBOOTRELEASE UBOOTVERSION

430 export ARCH CPU BOARD VENDOR SOC CPUDIR BOARDDIR

431 export CONFIG_SHELL HOSTCC HOSTCFLAGS HOSTLDFLAGS CROSS_COMPILE AS LD CC

432 export CPP AR NM LDR STRIP OBJCOPY OBJDUMP

433 export MAKE LEX YACC AWK PERL PYTHON PYTHON2 PYTHON3

434 export HOSTCXX HOSTCXXFLAGS CHECK CHECKFLAGS DTC DTC_FLAGS

435

436 export KBUILD_CPPFLAGS NOSTDINC_FLAGS UBOOTINCLUDE OBJCOPYFLAGS LDFLAGS

437 export KBUILD_CFLAGS KBUILD_AFLAGS

这些变量中大部分都已经在前面定义了,我们重点来看一下下面这几个变量:

ARCH CPU BOARD VENDOR SOC CPUDIR BOARDDIR

这 7 个变量在顶层 Makefile 是找不到的,说明这 7 个变量是在其他文件里面定义的,先 来看一下这 7 个变量都是什么内容,在顶层 Makefile 中输入如下图所示的内容:



图 12.2.9 输出变量值

修改好顶层 Makefile 以后执行如下命令:

make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi- mytest

结果如下图所示:



图 12.2.10 变量结果

从上图可以看到这7个变量的值,这7个变量是从哪里来的呢?在uboot根目录下有个文件叫做 config.mk,这7个变量就是在 config.mk 里面定义的,打开 config.mk 内容如下:

示例代码 config.mk 代码

1 # SPDX-License-Identifier: GPL-2.0+
```
原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php
   2 #
   3 # (C) Copyright 2000-2013
   4 # Wolfgang Denk, DENX Software Engineering, wd@denx.de.
   6
   7 # This file is included from ./Makefile and spl/Makefile.
   8 # Clean the state to avoid the same flags added twice.
   9 #
   10 # (Tegra needs different flags for SPL.
   11 # That's the reason why this file must be included from spl/Makefile too.
   12 # If we did not have Tegra SoCs, build system would be much simpler...)
   13 PLATFORM_RELFLAGS :=
   14 PLATFORM_CPPFLAGS :=
   15 PLATFORM_LDFLAGS :=
   16 LDFLAGS :=
   17 LDFLAGS_FINAL :=
   18 LDFLAGS STANDALONE :=
   19 OBJCOPYFLAGS :=
   20 # clear VENDOR for tcsh
   21 VENDOR :=
   23
   24 ARCH := $(CONFIG_SYS_ARCH:"%"=%)
   25 CPU := $(CONFIG_SYS_CPU:"%"=%)
   26 ifdef CONFIG_SPL_BUILD
   27 ifdef CONFIG_TEGRA
   28 CPU := arm720t
   29 endif
   30 endif
   31 BOARD := $(CONFIG_SYS_BOARD:"%"=%)
   32 ifneq ($(CONFIG_SYS_VENDOR),)
   33 VENDOR := $(CONFIG_SYS_VENDOR:"%"=%)
   34 endif
   35 ifneq ($(CONFIG_SYS_SOC),)
   36 SOC := $(CONFIG_SYS_SOC:"%"=%)
   37 endif
   38
   39 # Some architecture config.mk files need to know what CPUDIR is set to,
   40 # so calculate CPUDIR before including ARCH/SOC/CPU config.mk files.
   41 # Check if arch/$ARCH/cpu/$CPU exists, otherwise assume arch/$ARCH/cpu contains
```

正点原子

42 # CPU-specific code.



```
原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php
   43 CPUDIR=arch/$(ARCH)/cpu$(if $(CPU),/$(CPU),)
   44
   45 sinclude $(srctree)/arch/$(ARCH)/config.mk # include architecture dependend rules
   46 sinclude $(srctree)/$(CPUDIR)/config.mk
                                           # include CPU
                                                          specific rules
   47
   48 ifdef SOC
   49 sinclude $(srctree)/$(CPUDIR)/$(SOC)/config.mk # include SoC specific rules
   50 endif
   51 ifneq ($(BOARD),)
   52 ifdef VENDOR
   53 BOARDDIR = $(VENDOR)/$(BOARD)
   54 else
   55 BOARDDIR = $(BOARD)
   56 endif
   57 endif
   58 ifdef BOARD
   59 sinclude $(srctree)/board/$(BOARDDIR)/config.mk # include board specific rules
   60 endif
   61
   62 ifdef FTRACE
   63 PLATFORM CPPFLAGS += -finstrument-functions -DFTRACE
   64 endif
   65
   67
   68 RELFLAGS := $(PLATFORM_RELFLAGS)
   69
   70 PLATFORM_CPPFLAGS += $(RELFLAGS)
   71 PLATFORM_CPPFLAGS += -pipe
   72
   73 LDFLAGS += $(PLATFORM_LDFLAGS)
   74 LDFLAGS_FINAL += -Bstatic
   75
   76 export PLATFORM_CPPFLAGS
   77 export RELFLAGS
   78 export LDFLAGS_FINAL
   79 export LDFLAGS STANDALONE
   80 export CONFIG_STANDALONE_LOAD_ADDR
    第 24 行 定 义 变 量 ARCH, 值 为 $(CONFIG_SYS_ARCH:"%"=%), 也 就 是 提 取
```

```
CONFIG_SYS_ARCH 里面双引号""之间的内容。比如 CONFIG_SYS_ARCH="arm"的话, ARCH=arm。
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

第 25 行定义变量 CPU,值为\$(CONFIG_SYS_CPU:"%"=%)。

第 31 行定义变量 BOARD, 值为(CONFIG_SYS_BOARD:"%"=%)。

第 33 行定义变量 VENDOR, 值为\$(CONFIG_SYS_VENDOR:"%"=%)。

第36行定义变量SOC,值为\$(CONFIG_SYS_SOC:"%"=%)。

第 43 行定义变量 CPUDIR, 值为 arch/\$(ARCH)/cpu\$(if \$(CPU),/\$(CPU),)。

第45行 sinclude 和 include 的功能类似,在 Makefile 中都是读取指定文件内容,这里读取

文件\$(srctree)/arch/\$(ARCH)/config.mk 的内容。sinclude 读取的文件如果不存在的话不会报错。 第 46 行读取文件\$(srctree)/\$(CPUDIR)/config.mk 的内容。

第49行读取文件\$(srctree)/\$(CPUDIR)/\$(SOC)/config.mk的内容。

第 53 行定义变量 BOARDDIR,如果定义了 VENDOR 那么 BOARDDIR=\$(VENDOR)/\$(B OARD),否则的 BOARDDIR=\$(BOARD)。

第59行读取文件\$(srctree)/board/\$(BOARDDIR)/config.mk。

接下来需要找到 CONFIG_SYS_ARCH、CONFIG_SYS_CPU、CONFIG_SYS_BOARD、 CONFIG_SYS_VENDOR 和 CONFIG_SYS_SOC 这 5 个变量的值。这 5 个变量在 uboot 根目录 下的.config 文件中有定义,定义如下:

示例代码 .config 文件代码

19 CONFIG_SYS_ARCH="arm" 20 CONFIG_SYS_CPU="armv7" 21 CONFIG_SYS_SOC="zynq" 22 CONFIG_SYS_VENDOR="xilinx" 23 CONFIG_SYS_BOARD="zynq" 24 CONFIG_SYS_CONFIG_NAME="zynq_altk" 根据上面这段代码可知: ARCH = armCPU = armv7BOARD = zynq VENDOR = xilinx SOC = zynq CPUDIR = arch/arm/cpu/armv7 BOARDDIR = xilinx/zynq 在 config.mk 中读取的文件有: arch/arm/config.mk arch/arm/cpu/armv7/config.mk arch/arm/cpu/armv7/zynq/config.mk (此文件不存在) board/xilinx/zynq/config.mk (此文件不存在)

12.2.13 make xxx_defconfig 过程

在编译 uboot 之前要使用"make xxx_defconfig"命令来配置 uboot,那么这个配置过程是 如何运行的呢? 在顶层 Makefile 中有如下代码:

示例代码 顶层 Makefile 代码段

475 # To make sure we do not include .config for any of the *config targets



```
原子哥在线教学: www.yuanzige.com
                                                 论坛:www.openedv.com/forum.php
    476 # catch them early, and hand them over to scripts/kconfig/Makefile
    477 # It is allowed to specify more targets when calling make, including
    478 # mixing *config targets and build targets.
    479 # For example 'make oldconfig all'.
    480 # Detect when mixed targets is specified, and make a second invocation
    481 # of make so .config is not included in this case either (for *config).
    482
    483 version_h := include/generated/version_autogenerated.h
    484 timestamp_h := include/generated/timestamp_autogenerated.h
    485 defaultenv h := include/generated/defaultenv autogenerated.h
    486 dt_h := include/generated/dt.h
    487
    488 no-dot-config-targets := clean clobber mrproper distclean \
    489
               help %docs check% coccicheck \
    490
               ubootversion backup tests check qcheck
    491
    492 config-targets := 0
    493 mixed-targets := 0
    494 dot-config := 1
    495
    496 ifneq ($(filter $(no-dot-config-targets), $(MAKECMDGOALS)),)
    497
          ifeq ($(filter-out $(no-dot-config-targets), $(MAKECMDGOALS)),)
    498
            dot-config := 0
    499
          endif
    500 endif
    501
    502 ifeq ($(KBUILD_EXTMOD),)
    503
             ifneq ($(filter config %config,$(MAKECMDGOALS)),)
    504
                 config-targets := 1
    505
                 ifneq ($(words $(MAKECMDGOALS)),1)
    506
                      mixed-targets := 1
    507
                 endif
    508
            endif
    509 endif
    510
    511 ifeq ($(mixed-targets),1)
    513 # We're called with mixed targets (*config and build targets).
    514 # Handle them one by one.
    515
    516 PHONY += $(MAKECMDGOALS) __build_one_by_one
```

```
原子哥在线教学: www.yuanzige.com
                                                论坛:www.openedv.com/forum.php
    517
    518 $(filter-out __build_one_by_one, $(MAKECMDGOALS)): __build_one_by_one
    519
          @:
    520
    521 __build_one_by_one:
    522 $(Q)set -e; \
    523
         for i in $(MAKECMDGOALS); do \
    524
            $(MAKE) -f $(srctree)/Makefile $$i; \
    525
          done
    526
    527 else
    528 ifeq ($(config-targets),1)
    529 # =========
    530 # *config targets only - make sure prerequisites are updated, and descend
    531 # in scripts/kconfig to make the *config target
    532
    533 KBUILD_DEFCONFIG := sandbox_defconfig
    534 export KBUILD_DEFCONFIG KBUILD_KCONFIG
    535
    536 config: scripts_basic outputmakefile FORCE
          $(Q)$(MAKE) $(build)=scripts/kconfig $@
    537
    538
    539 %config: scripts_basic outputmakefile FORCE
          $(Q)$(MAKE) $(build)=scripts/kconfig $@
    540
    541
    542 else
    543 # ======
    544 # Build targets only - this includes vmlinux, arch specific targets, clean
    545 # targets and others. In general all targets except *config targets.
    546
    547 # Additional helpers built in scripts/
    548 # Carefully list dependencies so we do not try to build scripts twice
    549 # in parallel
    550 PHONY += scripts
    551 scripts: scripts_basic include/config/auto.conf
    552
          $(Q)$(MAKE) $(build)=$(@)
    553
    554 ifeq ($(dot-config),1)
    555 # Read in config
    556 -include include/config/auto.conf
```

正点原子



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

第 483 行定义了变量 version_h,这变量保存版本号文件,此文件是自动生成的。文件 include/generated/version_autogenerated.h内容如下图所示:



图 12.2.11 版本号文件

第 484 行定义了变量 timestamp_h,此变量保存时间戳文件,此文件也是自动生成的。文 件 include/generated/timestamp autogenerated.h 内容如下图所示:

wmq@Linux: ~/uboot/alientek-uboot-v2020.1/include/generated

文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H) 1 #define U_BOOT_DATE "Aug 02 2023" 2 #define U_BOOT_TIME "10:43:29" 3 #define U_BOOT_TZ "+0800" 4 #define U_BOOT_DMI_DATE "08/02/2023" 5 #define U BOOT BUILD DATE 0x20230802

图 12.2.12 时间戳文件

第488行定义了变量 no-dot-config-targets。

第492行定义了变量 config-targets, 初始值为 0。

第493行定义了变量 mixed-targets, 初始值为 0。

第494行定义了变量 dot-config, 初始值为1。

第496行将 MAKECMDGOALS 中不符合 no-dot-config-targets 的部分过滤掉,剩下的如果 不为空的话条件就成立。MAKECMDGOALS 是 make 的一个环境变量,这个变量会保存你所 指定的终极目标列表,比如执行"make atk 7020 defconfig",那么 MAKECMDGOALS 就为 atk_7020_defconfig。很明显过滤后为空,所以条件不成立,变量 dot-config 依旧为 1。

第 502 行判断 KBUILD_EXTMOD 是否为空,如果 KBUILD_EXTMOD 为空的话条件成 立,经过前面的分析,我们知道 KBUILD_EXTMOD 为空,所以条件成立。

第 503 行将 MAKECMDGOALS 中不符合 "config" 和 "%config" 的部分过滤掉, 如果 剩下的部分不为空条件就成立,很明显此处条件成立,变量 config-targets=1。

第 505 行统计 MAKECMDGOALS 中的单词个数,如果不为 1 的话条件成立。此处调用 Makefile 中的 words 函数来统计单词个数, words 函数格式如下:

\$(words <text>)

很明显,MAKECMDGOALS的单词个数是1个,所以条件不成立,mixed-targets继续为 0。综上所述,这些变量值如下:

```
config-targets = 1
mixed-targets = 0
dot-config = 1
```

第 511 行如果变量 mixed-targets 为 1 的话条件成立,很明显,条件不成立。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

第528行如果变量 config-targets 为1的话条件成立,很明显,条件成立,执行这个分支。 第536行,没有目标与之匹配,所以不执行。

第 539 行,有目标与之匹配,当输入"make xxx_defconfig"的时候就会匹配到%config目标,目标"%config"依赖于 scripts_basic、outputmakefile 和 FORCE。FORCE 在顶层 Makefile 的 2177 行有如下定义:

示例代码 顶层 Makefile 代码段

2177 PHONY += FORCE

2178 FORCE:

可以看出 FORCE 是没有规则和依赖的,所以每次都会重新生成 FORCE。当 FORCE 作为 其他目标的依赖时,由于 FORCE 总是被更新过的,因此依赖所在的规则总是会执行的。

依赖 scripts_basic 和 outputmakefile 在顶层 Makefile 中的内容如下:

示例代码 顶层 Makefile 代码段

455 # Basic helpers built in scripts/

456 PHONY += scripts_basic

457 scripts_basic:

458 \$(Q)\$(MAKE) \$(build)=scripts/basic

459 \$(Q)rm -f .tmp_quiet_recordmcount

460

461 # To avoid any implicit rule to kick in, define an empty command.

462 scripts/basic/%: scripts_basic ;

463

464 PHONY += outputmakefile

465 # outputmakefile generates a Makefile in the output directory, if using a

466 # separate output directory. This allows convenient use of make in the

467 # output directory.

468 outputmakefile:

469 ifneq (\$(KBUILD_SRC),)

470 \$(Q)ln -fsn \$(srctree) source

471 \$(Q)\$(CONFIG_SHELL) \$(srctree)/scripts/mkmakefile \

472 \$(srctree) \$(objtree) \$(VERSION) \$(PATCHLEVEL)

473 endif

第469行,判断 KBUILD_SRC 是否为空,只有变量 KBUILD_SRC 不为空的时候 outputm akefile 才有意义,经过我们前面的分析 KBUILD_SRC 为空,所以 outputmakefile 无效,只有 s cripts_basic 是有效的。

第 457~459 行是 scripts_basic 的规则,其对应的命令用到了变量 Q、MAKE 和 build,其中:

Q=@或为空 MAKE=make 变量 build 是在 scripts/Kbuild.include 文件中有定义,定义如下: 示例代码 Kbuild.include 代码段

178 ###



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php 179 # Shorthand for \$(Q)\$(MAKE) -f scripts/Makefile.build obj= 180 # Usage: 181 # \$(Q)\$(MAKE) \$(build)=dir 182 build := -f \$(srctree)/scripts/Makefile.build obj 从上面的示例代码可以看出 build=-f \$(srctree)/scripts/Makefile.build obj, 经过前面的分析 可知,变量 srctree 为".",因此: build=-f ./scripts/Makefile.build obj scripts basic 展开以后如下: scripts basic: @make -f ./scripts/Makefile.build obj=scripts/basic //可以没有@,视配置而定 @rm -f .tmp_quiet_recordmcount //可以没有@ scripts basic 会调用文件./scripts/Makefile.build,这个我们后面在分析。 接着回到 Makefile 第 539 的% config 处,内容如下: %config: scripts_basic outputmakefile FORCE \$(Q)\$(MAKE) \$(build)=scripts/kconfig \$@ 将命令展开就是: @make -f ./scripts/Makefile.build obj=scripts/kconfig xxx_defconfig //可以没有@ 同样也跟文件./scripts/Makefile.build 有关,我们后面再分析此文件。使用如下命令配置 uboot,并观察其配置过程: make atk_7020_defconfig V=1 配置过程如下图所示: q@Linux:~/uboot/alientek-uboot-v2020.1\$ make atk_7020 defconfig V=1 make -f ./scripts/Makefile.build obj=scripts/basic -f .tmp_quiet_recordmcount make -f ./scripts/Makefile.build obj=scripts/kconfig atk_7020_defconfig scripts/kconfig/conf --defconfig=arch/../configs/atk_7020_defconfig Kconfig arch/../configs/atk_7020_defconfig:86:warning: symbol value '3 ' invalid for B00

> configuration written to .config mg@Linux:~/uboot/alientek-uboot-v2020.1\$

> > 图 12.2.13 uboot 配置过程

从上图可以看出,我们的分析是正确的,接下来就要结合下面两行命令重点分析一下文件 scripts/Makefile.build。

①、scripts_basic 目标对应的命令

@make -f ./scripts/Makefile.build obj=scripts/basic

②、%config 目标对应的命令

 $@make -f \ ./scripts/Makefile.build \ obj=scripts/kconfig \ xxx_def config$

12.2.14 Makefile.build 脚本分析

TDELAY

从上一小节可知, "make xxx_defconfig" 配置 uboot 的时候如下两行命令会执行脚本 scripts/Makefile.build:

@make -f ./scripts/Makefile.build obj=scripts/basic

@make -f ./scripts/Makefile.build obj=scripts/kconfig xxx_defconfig



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

依次来分析一下:

1、scripts basic 目标对应的命令

scripts_basic 目标对应的命令为: @make -f ./scripts/Makefile.build obj=scripts/basic。打开 文件 scripts/Makefile.build,有如下代码:

示例代码 Makefile.build 代码段

```
6 # Modified for U-Boot
```

```
7 prefix := tpl
```

8 src := \$(patsubst \$(prefix)/%,%,\$(obj))

9 ifeq (\$(obj),\$(src))

10 prefix := spl

11 src := \$(patsubst \$(prefix)/%,%,\$(obj))

12 ifeq (\$(obj),\$(src))

13 prefix :=.

14 endif

15 endif

第7行定义了变量 prefix 值为 tpl。

第8行定义了变量 src,这里用到了函数 patsubst,此行代码展开后为:

\$(patsubst tpl/%,%, scripts/basic)

patsubst 是替换函数,格式如下:

\$(patsubst <pattern>,<replacement>,<text>)

此函数用于在 text 中查找符合 pattern 的部分,如果匹配的话就用 replacement 替换掉。 pattern 是可以包含通配符"%",如果 replacement 中也包含通配符"%",那么 replacement 中的这个 "%" 将是 pattern 中的那个 "%" 所代表的字符串。函数的返回值为替换后的字符 串。因此, 第8行就是在"scripts/basic"中查找符合"tpl/%"的部分, 然后将"tpl/"取消掉, 但是"scripts/basic"没有"tpl/",所以 src= scripts/basic。

第9行判断变量 obj 和 src 是否相等,相等的话条件成立,很明显,此处条件成立。

第10行和第7行一样,只是这里处理的是"spl","scripts/basic"里面也没有"spl/", 所以 src 继续为 scripts/basic。

第13行因为变量 obj 和 src 相等,所以 prefix=.。

继续分析 scripts/Makefile.build,有如下代码:

示例代码 Makefile.build 代码段

54 # The filename Kbuild has precedence over Makefile

55 kbuild-dir := \$(if \$(filter /%,\$(src)),\$(src),\$(srctree)/\$(src))

56 kbuild-file := \$(if \$(wildcard \$(kbuild-dir)/Kbuild),\$(kbuild-dir)/Kbuild,\$(kbuild-dir)/Makefile)

57 include \$(kbuild-file)

将 kbuild-dir 展开后为:

\$(if \$(filter /%, scripts/basic), scripts/basic, ./scripts/basic),

因为没有以"/"为开头的单词,所以\$(filter /%, scripts/basic)的结果为空, kbuilddir=./scripts/basic。

将 kbuild-file 展开后为:

\$(if \$(wildcard ./scripts/basic/Kbuild), ./scripts/basic/Kbuild, ./scripts/basic/Makefile)



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

因为 scrpts/basic 目录中没有 Kbuild 这个文件,所以 kbuild-file= ./scripts/basic/Makefile。 最后将 57 行展开,即:

include ./scripts/basic/Makefile

也就是读取 scripts/basic 下面的 Makefile 文件。

继续分析 scripts/Makefile.build,如下代码:

示例代码 Makefile.build 代码段

114 __build: \$(if \$(KBUILD_BUILTIN),\$(builtin-target) \$(lib-target) \$(extra-y)) \

115 \$(if \$(KBUILD_MODULES),\$(obj-m) \$(modorder-target)) \

116 \$(subdir-ym) \$(always)

117 @:

__build 是默认目标,因为命令"@make -f ./scripts/Makefile.build obj=scripts/basic"没有 指定目标,所以会使用到默认目标: __build。在项层 Makefile 中,KBUILD_BUILTIN 为 1, KBUILD_MODULES 为 0,因此展开后目标__build 为:

__build:\$(builtin-target) \$(lib-target) \$(extra-y)) \$(subdir-ym) \$(always)

@:

可以看出目标__build 有 5 个依赖: builtin-target、lib-target、extra-y、subdir-ym 和 always。 这 5 个依赖的具体内容我们就不通过源码来分析了,直接在 scripts/Makefile.build 中输入下图 所示内容,将这 5 个变量的值打印出来:



图 12.2.14 输出变量

执行如下命令:

make atk_7020_defconfig V=1

结果如下图所示:



图 12.2.15 输出结果

从上图可以看出,只有 always 有效,因此_build 最终为:

②正点原子

原子哥在线教学: www.yuanzige.com 论坛:www

论坛:www.openedv.com/forum.php

__build: scripts/basic/fixdep

@:

__build 依赖于 scripts/basic/fixdep,所以要先编译 scripts/basic/fixdep.c,生成 fixdep,前面已经读取了 scripts/basic/Makefile 文件。

综上所述, scripts_basic 目标的作用就是编译出 scripts/basic/fixdep 这个软件。

2、%config 目标对应的命令

%config 目标对应的命令为: @make -f ./scripts/Makefile.build obj=scripts/kconfig xxx_defconfig,各个变量值如下:

src= scripts/kconfig

kbuild-dir = ./scripts/kconfig

kbuild-file = ./scripts/kconfig/Makefile

include ./scripts/kconfig/Makefile

可以看出, Makefilke.build 会读取 scripts/kconfig/Makefile 中的内容, 此文件有如下所示 内容:

示例代码 scripts/kconfig/Makefile 代码段

127 %_defconfig: \$(obj)/conf

128 \$(Q)\$< \$(silent) --defconfig=arch/\$(SRCARCH)/configs/\$@ \$(Kconfig)

129

130 # Added **for** U-Boot (backward compatibility)

131 %_config: %_defconfig

132 @:

目标%_defconfig 刚好和我们输入的 xxx_defconfig 匹配,所以会执行这条规则。依赖为 \$(obj)/conf,展开后就是 scripts/kconfig/conf。接下来就是检查并生成依赖 scripts/kconfig/conf。 conf 是主机软件,到这里我们就打住,不要纠结 conf 是怎么编译出来的,否则就越陷越深, 像 conf 这种主机所使用的工具类软件我们一般不关心它是如何编译产生的。如果一定要看是 conf 是怎么生成的,可以输入如下命令重新配置 uboot,在重新配置 uboot 的过程中就会输出 conf 编译信息。

make distclean

make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi- atk_7020_defconfig V=1 结果如下图所示:





图 12.2.16 编译过程

得到 scripts/kconfig/conf 以后就要执行目标%_defconfig 的命令:

\$(Q)\$< \$(silent) --defconfig=arch/\$(SRCARCH)/configs/\$@ \$(Kconfig)

相关的变量值如下:

silent=-s 或为空

SRCARCH= ..

Kconfig=Kconfig

将其展开就是:

@ scripts/kconfig/conf --defconfig=arch/../configs/xxx_defconfig Kconfig

上述命令用到了 xxx_defconfig 文件,比如 xilinx_zynq_virt_defconfig。这里会将 atk_7020_defconfig 中的配置输出到.config 文件中,最终生成 uboot 根目录下的.config 文件。 这个就是命令 make xxx_defconfig 执行流程,总结一下如下图所示:

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php scripts_basic make -f ./scripts/Makefile.build obj=scripts/basic 生成: scripts/basic/fixdep outputmakefile 依赖 FORCE 顶层Makefile make xxx_defconfig %config -->make -f ./scripts/Makefile.build obj=scripts/kconfig xxx_defconfig 命令 生成: scripts/kconfig/conf : scripts/kconfig/conf --defconfig=arch/../configs/xxx_defconfig Kconfig 生成:.config

正点原子

图 12.2.17 make xxx_defconfig 执行流程图

至此, make xxx_defconfig 就分析完了, 接下来就要分析一下 u-boot.bin 是怎么生成的了。

12.2.15 make 过程

配置好 uboot 以后就可以直接 make 编译了,因为没有指明目标,所以会使用默认目标, 主 Makefile 中的默认目标如下:

示例代码 顶层 Makefile 代码段

148 # That's our default target when none is given on the command line
149 PHONY := _all
150 _all:
目标_all 又依赖于 all,如下所示:
示例代码 顶层 Makefile 代码段

214 # If building an external module we do not care about the all: rule

215 # but instead _all depend on modules

216 PHONY += all

```
217 ifeq ($(KBUILD_EXTMOD),)
```

218 _all: all

219 else

220 _all: modules

```
221 endif
```

如果 KBUILD_EXTMOD 为空的话_all 依赖与 all 。这里不编译模块,所以 KBUILD_EXTMOD 肯定为空, _all 的依赖就是 all 。在主 Makefile 中 all 目标规则如下:

示例代码 顶层 Makefile 代码段

```
859 all: $(ALL-y) cfg
860 ifeq ($(CONFIG_DM_I2C_COMPAT)$(CONFIG_SANDBOX),y)
```

862 @echo "This board uses CONFIG_DM_I2C_COMPAT. Please remove"

863 @echo "(possibly in a subsequent patch in your series)"

864 @echo "before sending patches to the mailing list."

京子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php	
865 @echo "====================================	
866 endif	
867 @# Check that this build does not use CONFIG options that we do not	
868 @# know about unless they are in Kconfig. All the existing CONFIG	
869 @# options are whitelisted, so new ones should not be added.	
870 \$(call cmd,cfgcheck,u-boot.cfg)	
从 859 行可以看出, all 目标依赖\$(ALL-y)和 cfg, 而在顶层 Makefile 中, ALL-y 如下:	
示例代码 顶层 Makefile 代码段	
846 # Always append ALL so that arch config.mk's can add custom ones	
847 ALL-y += u-boot.srec u-boot.bin u-boot.sym System.map binary_size_check	
848	
849 ALL-\$(CONFIG_ONENAND_U_BOOT) += u-boot-onenand.bin	
850 ifeq (\$(CONFIG_SPL_FSL_PBL),y)	
851 ALL-\$(CONFIG_RAMBOOT_PBL) += u-boot-with-spl-pbl.bin	
852 else	
853 ifneq (\$(CONFIG_SECURE_BOOT), y)	
854 # For Secure Boot The Image needs to be signed and Header must also	
855 # be included. So The image has to be built explicitly	
856 ALL-\$(CONFIG_RAMBOOT_PBL) += u-boot.pbl	
857 endif	
858 endif	
859 ALL-\$(CONFIG_SPL) += spl/u-boot-spl.bin	
860 ifeq (\$(CONFIG_MX6)\$(CONFIG_IMX_HAB), yy)	
861 ALL-\$(CONFIG_SPL_FRAMEWORK) += u-boot-ivt.img	
862 else	
863 ifeq (\$(CONFIG_MX7)\$(CONFIG_IMX_HAB), yy)	
864 ALL-\$(CONFIG_SPL_FRAMEWORK) += u-boot-ivt.img	
865 else	
866 ALL-\$(CONFIG_SPL_FRAMEWORK) += u-boot.img	
867 endif	
868 endif	
869 ALL-\$(CONFIG_TPL) += tpl/u-boot-tpl.bin	
870 ALL-\$(CONFIG_OF_SEPARATE) += u-boot.dtb	
871 ifeq (\$(CONFIG_SPL_FRAMEWORK),y)	
872 ALL-\$(CONFIG_OF_SEPARATE) += u-boot-dtb.img	
873 endif	
874 ALL-\$(CONFIG_OF_HOSTFILE) += u-boot.dtb	
875 ifneq (\$(CONFIG_SPL_TARGET),)	
876 ALL-\$(CONFIG_SPL) += \$(CONFIG_SPL_TARGET:"%"=%)	
877 endif	
878 ALL-\$(CONFIG_REMAKE_ELF) += u-boot.elf	

正点原子



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php	
879 ALL-\$(CONFIG_EFI_APP) += u-boot-app.efi	
880 ALL-\$(CONFIG_EFI_STUB) += u-boot-payload.efi	
881	
882 ifneq (\$(BUILD_ROM)\$(CONFIG_BUILD_ROM),)	
883 ALL-\$(CONFIG_X86_RESET_VECTOR) += u-boot.rom	
884 endif	
885	
886 # Build a combined spl + u-boot image for sunxi	
887 ifeq (\$(CONFIG_ARCH_SUNXI)\$(CONFIG_SPL),yy)	
888 ALL-y += u-boot-sunxi-with-spl.bin	
889 endif	
890	
891 # enable combined SPL/u-boot/dtb rules for tegra	
892 ifeq (\$(CONFIG_TEGRA)\$(CONFIG_SPL),yy)	
893 ALL-y += u-boot-tegra.bin u-boot-nodtb-tegra.bin	
894 ALL-\$(CONFIG_OF_SEPARATE) += u-boot-dtb-tegra.bin	
895 endif	
896	
897 ALL-\$(CONFIG_ARCH_MEDIATEK) += u-boot-mtk.bin	
898	
899 # Add optional build target if defined in board/cpu/soc headers	
900 ifneq (\$(CONFIG_BUILD_TARGET),)	
901 ALL-y += \$(CONFIG_BUILD_TARGET:"%"=%)	
902 endif	
从上面的示例代码可以看出, ALL-y 包含 u-boot.srec、u-boot.bin、u-boot.syn	m 、

System.map 和 binary_size_check 这几个文件。根据 uboot 的配置情况也可能包含其他的文件, 比如:

ALL-\$(CONFIG_ONENAND_U_BOOT) += u-boot-onenand.bin

CONFIG_ONENAND_U_BOOT 就是 uboot 中跟 ONENAND 配置有关的,如果我们使能 了 ONENAND,那么在.config 配置文件中就会有 "CONFIG_ONENAND_U_BOOT=y" 这一 句。相当于 CONFIG_ONENAND_U_BOOT 是个变量,这个变量的值为 "y",所以展开以后 就是:

ALL-y += u-boot-onenand.bin

这个就是.config 里面的配置参数的含义,这些参数其实都是变量,后面跟着变量值,会在顶层 Makefile 或者其他 Makefile 中调用这些变量。

ALL-y 里面有个 u-boot.bin,这个就是我们最终需要的 uboot 二进制可执行文件,所作的 所有工作就是为了它。在顶层 Makefile 中找到 u-boot.bin 目标对应的规则,如下所示:

示例代码 顶层 Makefile 代码段

1093 ifeq (\$(CONFIG_MULTI_DTB_FIT),y) 1094

1095 ifeq (\$(CONFIG_MULTI_DTB_FIT_LZO),y)



```
原子哥在线教学: www.yuanzige.com
                                               论坛:www.openedv.com/forum.php
    1096 FINAL_DTB_CONTAINER = fit-dtb.blob.lzo
    1097 else ifeq ($(CONFIG_MULTI_DTB_FIT_GZIP),y)
    1098 FINAL_DTB_CONTAINER = fit-dtb.blob.gz
    1099 else
    1100 FINAL_DTB_CONTAINER = fit-dtb.blob
    1101 endif
    1102
    1103 fit-dtb.blob.gz: fit-dtb.blob
    1104 @gzip -kf9 $<>$@
    1105
    1106 fit-dtb.blob.lzo: fit-dtb.blob
    1107 @lzop -f9 $<>$@
    1108
    1109 fit-dtb.blob: dts/dt.dtb FORCE
    1110 $(call if_changed,mkimage)
    1111 ifneq ($(SOURCE_DATE_EPOCH),)
    1112 touch -d @$(SOURCE_DATE_EPOCH) fit-dtb.blob
    1113 chmod 0600 fit-dtb.blob
    1114 endif
    1115
    1116 MKIMAGEFLAGS_fit-dtb.blob = -f auto -A $(ARCH) -T firmware -C none -O u-boot \
    1117 -a 0 -e 0 -E \
    1118 $(patsubst %,-b arch/$(ARCH)/dts/%.dtb,$(subst ",,$(CONFIG_OF_LIST))) -d /dev/null
    1119
    1120 ifneq ($(EXT_DTB),)
    1121 u-boot-fit-dtb.bin: u-boot-nodtb.bin $(EXT_DTB)
    1122
            $(call if_changed,cat)
    1123 else
    1124 u-boot-fit-dtb.bin: u-boot-nodtb.bin $(FINAL_DTB_CONTAINER)
    1125 $(call if_changed,cat)
    1126 endif
    1127
    1128u-boot.bin: u-boot-fit-dtb.bin FORCE
    1129 $(call if_changed,copy)
    1130
    1131u-boot-dtb.bin: u-boot-nodtb.bin dts/dt.dtb FORCE
    1132 $(call if_changed,cat)
    1133
    1134else ifeq ($(CONFIG_OF_SEPARATE),y)
    1135u-boot-dtb.bin: u-boot-nodtb.bin dts/dt.dtb FORCE
    1136 $(call if_changed,cat)
```



原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php

1137

1138u-boot.bin: u-boot-dtb.bin FORCE

1139 \$(call if_changed,copy)

1140else

1141u-boot.bin: u-boot-nodtb.bin FORCE

1142 \$(call if changed,copy)

1143endif

第 1093 行判断 CONFIG MULTI DTB FIT 是否等于 y, 如果相等, 那条件就成立, 在.config 中搜索 "CONFIG_MULTI_DTB_FIT",未设置,说明条件不成立。

第 1134 行判断 CONFIG_OF_SEPARATE 是否等于 y,如果相等,那条件就成立, 在.config 中搜索"CONFIG_OF_SEPARATE",未设置,说明条件不成立。

第 1141 行就是目标 u-boot.bin 的规则,目标 u-boot.bin 依赖于 u-boot-nodtb.bin,命令为 \$(call if_changed,copy),这里调用了 if_changed, if_changed 是一个函数,这个函数在 scripts/Kbuild.include 中有定义,而顶层 Makefile 中会包含 scripts/Kbuild.include 文件,这个前 面已经说过了。

if_changed 在 Kbuild.include 中的定义如下:

示例代码 Kbuild.include 代码段

227###

- execute command if any prerequisite is newer than 228 # if_changed 229 # target, or command line has changed 230 # if_changed_dep - as if_changed, but uses fixdep to reveal dependencies 231 # including used config symbols 232 # if_changed_rule - as if_changed but execute rule instead 233 # See Documentation/kbuild/makefiles.txt for more info 234 235 ifneq (\$(KBUILD_NOCMDDEP),1) 236 # Check if both arguments has same arguments. Result is empty string if equal. 237 # User may override this check using make KBUILD_NOCMDDEP=1 238 arg-check = \$(strip \$(filter-out \$(cmd_\$(1)), \$(cmd_\$@)) \ 239 \$(filter-out \$(cmd_\$@), \$(cmd_\$(1)))) 240 else 241 arg-check = \$(if \$(strip \$(cmd_\$@)),,1) 242 endif 243 244 # Replace >\$< with >\$\$< to preserve \$ when reloading the .cmd file 245 # (needed for make) 246 # Replace >#< with >\#< to avoid starting a comment in the .cmd file 247 # (needed for make) 248 # Replace >'< with >'\"< to be able to enclose the whole string in '...' 249 # (needed for the shell) $250 \text{ make-cmd} = (\text{call escsq}, (\text{subst }#, \\\#, (\text{subst }$, $$, $(\text{cmd}_(1)))))$

领航者 ZYNQ 之嵌入式 Linux 开发指南	② 正点原子
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.	.com/forum.php
251	
252 # Find any prerequisites that is newer than target or that does not exist	st.
253 # PHONY targets skipped in both cases.	
254 any-prereq = \$(filter-out \$(PHONY),\$?) \$(filter-out \$(PHONY) \$(wi	ildcard \$^),\$^)
255	
256 # Execute command if command has changed or prerequisite(s) are u	ipdated.
257 #	
258 if_changed = \$(if \$(strip \$(any-prereq) \$(arg-check)),	
259 @set -e;	
260 \$(echo-cmd) \$(cmd_\$(1)); \	
261 printf '%s\n' 'cmd_\$@ := $(make-cmd)' > (dot-target).cmd)$	
第 228 行为 if_changed 的描述,根据描述,在一些先决条件 有改变的时候, if changed 就会执行一些命令。	比目标新的时候,或者命令行
第 258 行就是函数 if_changed, if_changed 函数引用的变量	比较多,也比较绕,我们只需
要知道它可以从 u-boot-nodtb.bin 生成 u-boot.bin 就行了。	
既然 u-boot.bin 依赖于 u-boot-nodtb.bin, 那么肯定要先生质	戎 u-boot-nodtb.bin 文件, 顶层
Makefile 中相关代码如下:	
示例代码 顶层 Makefile 代码段	
1236 u-boot-nodtb.bin: u-boot FORCE	
<pre>1237 \$(call if_changed,objcopy)</pre>	
1238 \$ (call DO_STATIC_RELA, \$<, \$@, \$ (CONFIG_SYS_	TEXT_BASE))
1239 \$ (BOARD_SIZE_CHECK)	
目标 u-boot-nodtb.bin 又依赖于 u-boot, 顶层 Makefile 中 u-b	poot相关规则如下:
示例代码 顶层 Makefile 代码段	
1670 u-boot: \$(u-boot-init) \$(u-boot-main) u-boot.lds FO	RCE
<pre>1671 +\$(call if_changed,u-boot)</pre>	
<pre>1672 ifeq (\$ (CONFIG_KALLSYMS),y)</pre>	
1673 \$(call cmd, smap)	
<pre>1674 \$(call cmd, u-boot) common/system_map.o</pre>	
1675 endif	
目标 u-boot 依赖于 u-boot_init、u-boot-main 和 u-boot.lds。	u-boot_init 和 u-boot-main 是两
个变量,在顶层 Makefile 中有定义,值如下:	
示例代码 顶层 Makefile 代码段	
782 u-boot-init:=\$(head-y)	
783 u-boot-main := \$(libs-y)	
\$(head-y)跟 CPU 架构有关,我们使用的是 ARM 芯片,所中被指定为:	以 head-y 在 arch/arm/Makefile
head-y := arch/arm/cpu/\$(CPU)/start.o	

根据 0 小节的分析,我们知道 CPU=armv7,因此 head-y 展开以后就是:

head-y := arch/arm/cpu/armv7/start.o

因此:

原子哥在线教学: www.yuanzige.com



论坛:www.openedv.com/forum.php

```
u-boot-init= arch/arm/cpu/armv7/start.o
$(libs-y)在顶层 Makefile 中被定义为 uboot 所有子目录下 build-in.o 的集合,代码如下:
                                   示例代码 顶层 Makefile 代码段
725 libs-y += lib/
726 libs-$(HAVE_VENDOR_COMMON_LIB) += board/$(VENDOR)/common/
727 libs-$(CONFIG OF EMBED) += dts/
728 libs-y += fs/
729 libs-y += net/
730 libs-y += disk/
731 libs-y += drivers/
732 libs-y += drivers/dma/
733 libs-y += drivers/gpio/
734 libs-y += drivers/i2c/
735 libs-y += drivers/net/
736 libs-y += drivers/net/phy/
737 libs-y += drivers/power/ \setminus
     drivers/power/domain/ \
738
739
     drivers/power/fuel_gauge/ \
740 drivers/power/mfd/ \setminus
741
     drivers/power/pmic/ \
742 drivers/power/battery/
743 drivers/power/regulator/
744 libs-y += drivers/spi/
745 libs-$(CONFIG_FMAN_ENET) += drivers/net/fm/
746 libs-$(CONFIG_SYS_FSL_DDR) += drivers/ddr/fsl/
747 libs-$(CONFIG_SYS_FSL_MMDC) += drivers/ddr/fsl/
748 libs-$(CONFIG_$(SPL_)ALTERA_SDRAM) += drivers/ddr/altera/
749 libs-y += drivers/serial/
750 libs-y += drivers/usb/cdns3/
751 libs-y += drivers/usb/dwc3/
752 libs-y += drivers/usb/common/
753 libs-y += drivers/usb/emul/
754 libs-y += drivers/usb/eth/
755 libs-$(CONFIG_USB_GADGET) += drivers/usb/gadget/
756 libs-$(CONFIG_USB_GADGET) += drivers/usb/gadget/udc/
757 libs-y += drivers/usb/host/
758 libs-y += drivers/usb/musb/
759 libs-y += drivers/usb/musb-new/
760 libs-y += drivers/usb/phy/
761 libs-y += drivers/usb/ulpi/
762 libs-y += cmd/
```



这里调用了函数 patsubst,将 libs-y 中的"/" 替换为"/built-in.o",比如"drivers/dma/" 就 变为了"drivers/dma/built-in.o",相当于将 libs-y 改为所有子目录中 built-in.o 文件的集合。那 么 u-boot-main 就等于所有子目录中 built-in.o 的集合。

这个规则就相当于将以 u-boot.lds 为链接脚本,将 arch/arm/cpu/armv7/start.o 和各个子目录 下的 built-in.o 链接在一起生成 u-boot。

u-boot.lds 的规则如下:

示例代码 顶层 Makefile 代码段

1832 u-boot.lds: \$(LDSCRIPT) prepare FORCE

1833 \$(call if_changed_dep,cpp_lds)

接下来的重点就是各子目录下的 built-in.o 是怎么生成的,以 drivers/gpio/built-in.o 为例, 在 drivers/gpio/目录下会有个名为.built-in.o.cmd 的文件,此文件内容如下:

示例代码 drivers/gpio/.built-in.o.cmd 代码

cmd_drivers/gpio/built-in.o := arm-linux-gnueabihf-ld.bfd -r -o drivers/gpio/built-in.o drivers/gpio/gpiouclass.o drivers/gpio/zynq_gpio.o

从命令"cmd_drivers/gpio/built-in.o"可以看出, drivers/gpio/built-in.o 这个文件是使用 ld 命令由文件 drivers/gpio/gpio-uclass.o 和 drivers/gpio/zynq_gpio.o 生成而来的,其中 zynq_gpio.o 是 zynq_gpio.c 编译生成的.o 文件,这个是 Xilinx 的 ZYNQ 系列的 GPIO 驱动文件。这里用到 了 ld 的"-r"参数,参数含义如下:

-r -relocateable: 产生可重定向的输出,比如,产生一个输出文件它可再次作为 'ld' 的 输入,这经常被叫做"部分链接",当我们需要将几个小的.o 文件链接成为一个.o 文件的时 候,需要使用此选项。



原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php

最终将各个子目录中的 built-in.o 文件链接在一起就形成了 u-boot, 使用如下命令编译 uboot 就可以看到链接的过程:

make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi- atk_7020_defconfig V=1

make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi- V=1

编译的时候会有如下图所示内容输出:

arm-xilinx-linux-gnueabi-gcc -E -Wp,-MD,./.u-boot-elf.lds.d -D KERNEL -D UBOOT
-DARMmarm -mno-thumb-interwork -mabi=aapcs-linux -mword-relocations -fno-pic
-mno-unaligned-access -ffunction-sections -fdata-sections -fno-common -ffixed-r9 -msof
t-float -pipe -march=armv7-a -D_LINUX_ARM_ARCH=7 -mtune=generic-armv7-a -I./arch/
arm/mach-zyng/include -Iinclude -I./arch/arm/include -include ./include/linux/kconfig.
h -nostdinc -isystem /opt/petalinux/2020.2/sysroots/x86 64-petalinux-linux/usr/lib/arm-
xilinx-linux-gnueabi/gcc/arm-xilinx-linux-gnueabi/9.2.0/include -ansi -include ./include
/u-boot/u-boot.lds.h -DCPUDIR=arch/arm/cpu/armv7 -D ASSEMBLY -x assembler-with-cpp -
std=c99 -P -o u-boot-elf.lds arch/u-boot-elf.lds
arm-xilinx-linux-qnueabi-objcopy -I binary -B arm -O elf32-littlearm u-boot.bin u-boot-e
lf.o
arm-xilinx-linux-gnueabi-ld.bfd u-boot-elf.o -o u-boot.elf -T u-boot-elf.ldsdefsvm=
" start"=0x4000000 -Ttext=0x4000000
./scripts/check-confia.sh_u-boot.cfg/scripts/config_whitelist.txt
wmg@Linux:~/uboot/alientek-uboot-v2020.1\$

图 12.2.18 U-Boot 链接过程

将其整理一下,内容如下:

arm-xilinx-linux-gnueabi-ld.bfd -pie --gc-sections -Bstatic --no-dynamic-linker -Ttext 0x4000000

-o u-boot -T u-boot.lds

arch/arm/cpu/armv7/start.o

--start-group arch/arm/cpu/built-in.o

arch/arm/cpu/armv7/built-in.o

arch/arm/lib/built-in.o

arch/arm/mach-zynq/built-in.o

board/xilinx/zynq/built-in.o

cmd/built-in.o

common/built-in.o

disk/built-in.o

drivers/built-in.o

drivers/dma/built-in.o

drivers/gpio/built-in.o

.

drivers/usb/phy/built-in.o

drivers/usb/ulpi/built-in.o

dts/built-in.o

env/built-in.o

fs/built-in.o

lib/built-in.o

net/built-in.o

test/built-in.o

test/dm/built-in.o

--end-group arch/arm/lib/eabi_compat.o



正点原子

原子哥在线教学: www.yuanzige.com arch/arm/lib/lib.a -Map u-boot.map; true

可以看出最终是用 arm-xilinx-linux-gnueabi-ld.bfd(arm-xilinx-linux-gnueabi-ld.bfd)命令将 arch/arm/cpu/armv7/start.o 和其他众多的 built_in.o 链接在一起,形成 u-boot。

目标 all 除了 u-boot.bin 以外还有其他的依赖,比如 u-boot.srec、u-boot.sym、System.map、u-boot.cfg 和 binary_size_check 等等,这些依赖的生成方法和 u-boot.bin 很类似,大家自行查看一下顶层 Makefile,我们就不详细的讲解了。

总结一下"make"命令的流程,如下图所示:



图 12.2.19 U-Boot make 命令流程

上图就是"make"命令的执行流程,关于 uboot 的顶层 Makefile 就分析到这里,重点是 "make xxx_defconfig"和 "make" 这两个命令的执行流程:

make xxx_defconfig: 用于配置 uboot, 这个命令最主要的目的就是生成.config 文件。

make:用于编译 uboot,这个命令的主要工作就是生成二进制的 u-boot.bin 文件和其他的一些与 uboot 有关的文件,如 u-boot.elf 等等。

关于 uboot 的顶层 Makefile 就分析到这里,有些内容我们没有详细、深入的去研究,因为 我们的重点是使用 uboot,而不是 uboot 的研究者,我们要做的是缕清 uboot 的流程。至于更 具体的实现,有兴趣的可以上网查阅其他 uboot 相关资料。



正点原子

第十三章 U-Boot 启动流程详解

上一章我们详细的分析了 uboot 的顶层 Makefile,理清了 uboot 的编译流程。本章我们来 详细的分析一下 uboot 的启动流程,理清 uboot 是如何启动的。通过对 uboot 启动流程的梳理, 我们就可以掌握一些外设是在哪里被初始化的,这样当我们需要修改这些外设驱动的时候就 会心里有数。另外,通过分析 uboot 的启动流程可以了解 Linux 内核是如何被启动的。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

13.1 使用 Vitis 工具编译和调试 uboot

古语云:工欲善其事,必先利其器。要想对 uboot 启动流程进行准确而细致的分析,少不 了调试尤其是单步调试,虽然 vscode 也能调试代码,但 vscode 偏向于纯软件的调试,而对于 uboot 这种与开发板硬件进行交互的 vscode 显然无能为力,所以我们需要祭出大杀器——Vitis。 下面我们讲解如何使用 Vitis 工具编译和调试 uboot。注:这里我们使用 Ubuntu 中的 Vitis 工 具,而不是 Windows 下的 Vitis 工具。

首先我们将开发板资料包路径: 4_SourceCode\3_Embedded_Linux\vivado_prj\vivado_prj 下的 Navigator_7020 工程下的 system_wrapper.xsa 文件夹复制到 Ubuntu 家目录 ~/petalinux/uboot_start下。

现在我们打开 Ubuntu 中的 Vitis 软件, Workspace 选择~/petalinux/uboot_start 目录,如下 图所示:

	Vitis IDE Launcher
Select a direct	ory as workspace
Vitis IDE uses	the workspace directory to store its preferences and development artifacts.
<u>W</u> orkspace:	/home/wmq/petalinux/uboot_start
Use this as	s the default and do not ask again
Restore oth	er Workspace
Recent Wor	kspaces
	Cancel Launch
	图 13.1.1 选择 Workspace

点击"Launch"按钮,进入 Vitis 工程界面,如下图所示:



原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php



图 13.1.2 Vitis 工程界面

界面与 Vivado 中的 Vitis 一样,操作也是一样的。现在我们创建一个名为"uboot"的空 应用工程。工程创建后的界面如下所示:



图 13.1.3 "uboot" 应用工程

可以看到 src 目录为空,现在我们往 src 目录添加 uboot 源码。此处我们使用正点原子移 植好的 uboot 源码。拷贝 uboot 源码到上图中的 src 目录下。添加完 uboot 源码,然后在 src 目 录下按键盘上的 F5 键刷新目录,刷新后的 src 目录如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php



图 13.1.4 src 目录

可以看到,uboot 源码已经添加进来了。现在我们来配置 uboot。在 Vitis 工具中单击下图 中箭头所指的位置,打开 bash shell。



图 13.1.5 打开 bash shell

弹出如下图所示的 bash shell 界面,输入 "pwd" 命令,得到当前的工作路径,如下图所示:

原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php

正点原子

bash --noprofile --norc

bash-4.4\$ pwd /home/wmq/petalinux/uboot_start bash-4.4\$ hash-4.4\$

图 13.1.6 当前的工作路径

可以看到当前路径处于 petalinux/uboot_start 目录下,我们切换路径到 src 目录下,输入命 令 "cd uboot/src" 切换到 src 目录下,如下图所示:

bash --noprofile --norc bash-4.4\$ pwd /home/wmq/petalinux/uboot_start bash-4.4\$ cd uboot/src/ bash-4.4\$ pwd /home/wmq/petalinux/uboot_start/uboot/src

图 13.1.7 切换到 src 目录下

然后先使用命令启动 vitis 环境下的交叉编译器

sh−4.4\$

. /opt/petalinux/2020.2/environment-setup-cortexa9t2hf-neon-xilinx-linux-gnueabi 当输入以上命令按回车后,在 shell 终端会打印出如下图所示的提示信息:

bashnoprofilenorc	00
<pre>bash=4.4\$ pwd /home/wmq/petalinux/uboot_start bash=4.4\$ cd uboot/src/ bash=4.4\$ pwd /home/wmq/petalinux/uboot_start/uboot/src bash=4.4\$, /opt/petalinux/2020,2/environment-setup-cortexa9t2hf-neon-xilinx-linux-s /our environment is misconfigured, you probably need to 'unset LD_LIBRARY_PATH' but please check why this was set in the first place and that it's safe to unset. The SDK will not operate correctly in most cases when LD_LIBRARY_PATH is set. For more references see: http://tldp.org/HOWTO/Program-Library-HOWTO/shared-libraries.html#AEN80 http://xahlee.info/UnixResource_dir/_/ldpath.html</pre>	gnueabi
bash-4.4\$	

图 13.1.8 启动交叉编译器

提示信息的大致意思为,若想使用交叉编译器,请先执行下列命令:

unset LD_LIBRARY_PATH

命令执行完后紧接着使用 echo \$CROSS_COMPILE 打印交叉编译器的名称,若打印结果为空,则再次输入. /opt/petalinux/2020/environment-setup-cortexa9t2hf-neon-xilinx-linux-gnueabi 命令,然后再 次输入 echo \$CROSS_COMPILE 语句,按回车后就可以打印出交叉编译器的名称,打印结果如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php



图 13.1.9 打印交叉编译器的名称

然后输入命令 "make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi-atk_7020_defconfig" 配置 uboot。配置完成后, 鼠标右键在弹出的菜单中选择"C/C++ Build Settings"或者 "Properties",对应的快捷键是"Alt+Enter",如下图所示:



图 13.1.10 选择 "C/C++ Build Settings" 或者 "Properties" 在弹出的属性界面中,单击"C/C++ Build",如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

😣 💷 Properties for ubo	ot
type filter text 🛛 🗷	C/C++ Build 🗘 🔻 🖒 🔻 🔫
 Resource Builders C/C++ Build Ruild Variables 	Configuration: Debug [Active]
Environment Logging	Builder Settings Behavior Refresh Policy Builder
Settings Tool Chain Editor ▶ C/C++ General	Builder type: External builder Image: Second sec
Project References Run/Debug Settings	Build command: make Variables Makefile generation Image: Command state st
	Generate Makefiles automatically Expand Env. Variable Refs in Makefile Build location
	Build directory: \${workspace_loc:/uboot/src} Workspace File system
?	Cancel

图 13.1.11 配置 "C/C++ Build"

取消勾选"Generate Makefiles automatically",然后点击"Workspace..."按钮,选择路径 为"uboot/src",修改完成后,"Build directory"的路径如上图所示。紧接着添加环境变量, 在属性界面的左栏选择"Environment",添加环境变量 CROSS_COMPILE,环境变量的值为 交叉编译器的绝对路径,笔者的交叉编译器的绝对路径为"/opt/petalinux/2020.2/sysroots/x86_ 64-petalinux-linux/usr/bin/arm-xilinx-linux-gnueabi/arm-xilinx-linux-gnueabi-gcc"如图 13.1.12 所 示,若读者不知道自己交叉编译器的绝对路径,可以在 ubuntu 的终端页面输入 whereis arm-xi linx-linux-gnueabi-gcc 命令后按回车就可以打印出交叉编译器的绝对路径,打印出的绝对路径 如图 13.1.13 所示:

	Properties for uboot	06
type filter text	Environment	
Resource Builders C/C++ Build Build Variables	Configuration: Debug [Active]	Configurations)
Environment	Environment variables to set	2 Add
Settings	Variable Value Origin	Select
Tool Chain Editor		Edit
C/C++ General	Name: CROSS_COMPILE 3	Delete
Project References	Value: -linux-gnueabi/arm-xilinx-linux-gnueabi- Variables	Undefine
Refactoring History Run/Debug Settings Validation	Cancel OK	5
	 Append variables to native environment Replace native environment with specified one 	
	Restore <u>D</u> efaults	6 <u>A</u> pply
?	Cancel	7 Apply and Close
	图 13.1.12 添加环境变量 CROSS_COMPILE	
	wmq@Linux: ~	
 件(F) 编辑(E) <u> 查</u> ā		
	reis arm-viliny-linux-onueahi-occ	

2 正点原子

wmq@Linux:~\$

图 13.1.13 打印交叉编译器的绝对路径

添加完环境变量 CROSS_COMPILE 后,单击上图 13.1.12 中 5 所示的 "OK" 按钮,然后单击 6 所示 "Apply" 按钮,最后单击 7 所示的 "Apply and Close" 按钮退出该界面。然后编译整个应用工程(可使用快捷键 Ctrl+B),编译结果若出现下图所示的报错信息,我们根据错误提示信息,将对应文件的权限改成最大,如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

🗐 Console 🛱 🔐 Problems 📗 Vitis Log 🛈 Guidance	000
Build Console [uboot, Debug]	
MKIMAGE spl/boot.bin MKIMAGE u-boot.img COPY u-boot.dtb	
MKIMAGE u-boot-dtb.img LDS u-boot-elf.lds LD u-boot.elf CFGCHK u-boot.cfg /bin/sh: ./scripts/check-config.sh: 权限不够	
Makefile:1079: recipe for target 'all' failed make: *** [all] Error 126	

15:58:28 Build Finished (took 43s.904ms)

图 13.1.14 编译报错

在 Vitis shell 中修改./scripts/check-config.sh 文件的权限为 777, 操作命令如下图所示:

bash-4.4\$	chmod	777	./scripts/check-config.sh
bash-4₊4\$			
bash-4₊4\$			

图 13.1.15 修改.scripts/check-config.sh 的权限

然后在 Vitis 中清除一下工程,若不清除,再次编译也会报错,操作步骤所下图所示:



图 13.1.16 清除上一次编译的工程

	-		2 A .	
原子哥在线教		v.yuanzige.com 说	论坛:www.openedv.com/foru	ım.php
最后再次练	编译,编	译结果如卜图所示:		
🗐 (Console	🛿 🔝 Problems 📗 Viti	is Log 🧴 Guidance	- 12 🔁 🖓 - 4
Buil	d Consol	e [uboot, Debug]		
0	PICODY	 	hin	
0	DJCOPT	spt/u-boot-spt-noutb	0.01II	
C	OPY	spl/u-boot-spl.dtb		
C	AT	spl/u-boot-spl-dtb.b	pin	
C	0PY	spl/u-boot-spl.bin		
м	KIMAGE	spl/boot.bin		
м	KIMAGE	u-boot.img		
C	OPY	u-boot.dtb		
м	KIMAGE	u-boot-dtb.img		
L	DS	u-boot-elf.lds		
L	D	u-boot.elf		
C	FGCHK	u-boot.cfg		

🔁 正点原子

10:10:17 Build Finished (took 42s.347ms)

图 13.1.17 编译没有出错

现在调试 uboot。将领航者开发板的启动模式设置为"JTAG"启动,连接 JTAG、串口和 电源,然后开发板上电。打开串口软件如 MobaXterm 或 CuteCom,设置好领航者开发板所使 用的串口并打开。

在 Vmware 软件的菜单栏点击虚拟机(M)菜单,在弹出的子菜单中点击"可移动设备(D)",在弹出的相应的移动设备子菜单选择"Future Devices Digilent USB Device", "Future Devices Digilent USB Device"是 JTAG 的 USB 接口,将该 USB 接口连接到虚拟机,如下图所示:

🔁 Ubuntu 64 位 - VMware Workstation							
文件(E) 编辑(E) 查看(V)	虚拟	\机(<u>M)</u> 选项卡(<u>T</u>) 帮助(<u>H</u>)	<mark> </mark> - 母 ₽	æ	🕰 🔲 🗆 🔁 🛛 ⊵	_7 ·	
库	\bigcirc	电源(P)	>				
	٢	可移动设备(D)	>		CD/DVD (SATA)	>	
		暂停(U)	Ctrl+Shift+P	~	网络适配器	>	11
■ □ 我的计算机 「 <mark>P</mark> Ubuntu 64 位	æ	发送 Ctrl+Alt+Del(E) 抓取輸入内容(I)	Ctrl+G		打印机 声卡	> >	
		ссп(п)	``````````````````````````````````````		Realtek 802.11n WLAN Adapter	>	
	F	551(1) 中昭(N)	>		Future Devices Diailent USB Device		连接(断开与 主机 的连接)(C)
	40	捕获屏幕 (C)	Ctrl+Alt+PrtScn		QinHeng USB Serial	>	更改图标(I)
	ß	答理(M)	```				✓ 在状态栏中显示(S)
	0	重新安装 VMware Tools(T)	,				
	[]	设置(S)	Ctrl+D				

图 13.1.18 在 Vmware 中连接 JTAG 的 USB 接口到虚拟机内

在应用工程 uboot 目录上鼠标右键单击,在弹出的菜单中选择 "Debug As" → "Debug Configurations",如下图所示:





图 13.1.19 打开 Debug Configurations 界面

弹出 Debug Configurations 界面,如下图所示,双击"single Application Debug",创建 "Debugger_uboot-Default",然后鼠标点击下图所示的方框 2 所指的"Target Setup"栏,将 方框 3 中的复选框都选中,配置信息如下图所示:

		Debug Configura	tions			•
Create, manage, and run configurations (Application]: Application 'Debug/ub 'uboot'.	s oot.el	f' specified for 'ps	7_cortexa	9_0'	doesn't exist in proje	ect 🔊
ご 證 急 圖 業 目 ♡ ▼ type filter text ▼ ≦ Single Application Debug	Nam	e: Debugger_ubo	ot-Default	et Se	tup (⋈= Arguments	»»5
🚡 Debugger_uboot-Default	Ha	rdware Platform:	\${sdxTcfLa		Search	B <u>r</u> owse
Single Application Debug (GDB)	Bit	stream File:			Search	B <u>r</u> owse
	PL	Device:	Auto Detect		Select	
	PS	Device:	Auto Detect		Select	
		Use FSBL flow for				
	Ini	tialization File:	_ide/psinit/		Search	B <u>r</u> owse
	3	Reset entire syste Program FPGA Skip Revision C Run ps7_init Run ps7_post_cor Enable Cross-Trig	rm Theck hfig gering	F F L I I Z Z	Summary: Following operations the debugger. 1. Resets entire syste 2. Program FPGA fab oetalinux/zynq_petal system_wrapper.bit). 3. Sources the init tcl zyna_petalinux/uboo	will be performe m. Clears the FPC ric (PL) using the inux/uboot/_ide, file (/home/wmq t/ ide/psinit/ps7

图 13.1.20 配置 "Target Setup" 栏

"Target Setup"栏信息配置完成后,鼠标点击上图方框 4 所指的"Application"栏,配置 信息如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

			Debug Configu	rations			۵ 🛚
Create, manage, and run configu	rations						The second
Debug a program using Applicat	ion Debugger.						- - - - - - - - - - -
type filter text	Name: Debugge	r_uboot-Default ication () Target Setu	up 🕬 Arguments 🕅	🖥 Environment े 📅 Symbol File	es 💱 Source 🎄 Pa	ath Map 🔲 <u>C</u> omn	non
 Single Application Debug Debugger_uboot-Default 	Summary	n'					
Single Application Debug (Download	Processor	Project	Application	Details		
		ps7_cortexa9_0	uboot	src/u-boot.elf	stop at eni	try = true, relocate	e elf = false
	Project:	iboot				Browse	
	Application:	rc/u-boot.elf				Search	B <u>r</u> owse
	Reset proce	ssor gram entry ons: Edit					
Filter matched 4 of 4 items						Revert	Apply
0					(Close	Debug

图 13.1.21 配置 "Application" 栏

配置完成后,依次单击 Debug Configurations 界面右下方的"Apply"和"Debug"按钮, 然后会弹出如下图所示的提示框,提示打开调试界面,点击"Yes"按钮。

	WARNING	8
?	Debug session is already active. Do you want to relaunch?	
	not show this warning again	
	Cancel OK	

图 13.1.22 提示打开调试界面

调试界面如图 13.1.23 所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

zynq_petalinux - Disassembly - Vitis IDE 🛛 😑 😑								
File Edit <u>R</u> un Search Xilin	Project W	indow Help						
🖻 🕶 🛛 🕤 🖉 🖛 🛛	🗏 N 🕹 🔻 3.	∿.¢≂,∞⊡,	1 🖬 🔅 🕇	• 🔿 • 🛷	▼ *\$-\$-	> •	🔍 🛛 🖓 Design 🕸 Deb	
🎋 Debug 🏼 📃 🗆	メ uboot	📟 Disassemb	y ⊠ "₁	- 8	(x)= Vari ≿	Bre	🛠 Exp 🕮 Regi 🗖 🗄	
2	Fr	nter location he	е 🚽 🔕	0 4 2 Q			2010 - 20	
Debugger_uboot-Default		_vector_tab	.e:	-	Name	Туре	Value	
▼ 🔐 APU	\$	b +	92 ; 48 ·	addr=			•	
ARM Cortex-A9 MPC		b +	72 :	addr=			9	
≡ 0x00000000_vecto		b +2	36 ;	addr=				
■ 0xffffff20 3		b +2 nop	:00;	addr≕	Memory	я	- 6	
=		b +() ;	addr=		ন্ত্রী	1010 1010 📑 🚽 🚮 📰 🐼 👯 👻	
ARM Cortex-A9 MPC	4	b +6	i0 ;	addr=	Moni 👍 🗙	¥.		
æxc7z010		IRQHandler:	0 51 52	r2 r1		<u></u>		
		vpush {	0,11,12	,13,11		6		
		vpush {	16-d31}			•		
		vmrs r	, fpscr					
		push {	1}					
	·	vmrs r	. tpexc					

图 13.1.23 调试界面

方框1是常用的调试命令,方框2是 Debug 窗口,方框3处显示的是当前 PC 指针的值和 当前运行的命令对应的源码文件, 方框 4 是源码文件, 可以看到箭头指向_vector_table: 的下 一行,方框 5 是常用的信息框,可以显示变量 Variables、断点 Breakpoint、寄存器 Register 信 息等内容, 方框 6显示内存 Memory 的内容。

对于方框1我们只要知道单步调试按键盘上的F5,遇到函数时不想进入就按F6,进入函 数想返回时按 F7 即可。我们具体的来看下方框 5。方框 5 中的 Variables 显示变量内容, 尤其 当进入函数时非常有用。当然了更有信息价值的是寄存器 Register 栏,所以我们看下寄存器 Register 栏的信息有哪些。

^{(x)=} Variable	s °e	Breakpoints 😚	€ E)	kpressions 🔤 Reg	jis	sters 🛛	
Name		Hex		Decimal		Description	Mnemonic
1010 0101 FO		f8007028		4160778280			
8888 г1		fffffff		4294967295			
1919 г2		4e00e07f		1308680319			
1919 гЗ		f8007000		4160778240			
¹⁰¹⁰ г4		fffff00		4294967040			
3939 r 5		ffffff04		4294967044			
1010 гб		00000018		24			
1919 г7		f8000004		4160749572			
1919 г8		0000767b		30331			
¹⁰¹⁰ г9		e3e0f0d3		3823169747			
^{រព្រព} r10		f8000244		4160750148			
^{រពរ} ា ក11		9fa90ddc		2678656476			
1919 г12		00000000		0			
¹⁰¹⁰ sp		e5801000		3850375168			
1919 Lr		ffffff20		4294967072			
¹⁹¹⁹ pc		00000000		Θ			
▶ 1010 cpsr		200001df		536871391			
▶ 1010 USF							

图 13.1.24 寄存器 Register 栏



原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php

从上图可以看出,Register栏中包括 Name、Hex、Decimal、Description 和 Mnemonic。我 们常用的是 Name 用来显示寄存器名,Hex 显示寄存器的 16 进制值和 Desciption,描述寄存器 内容。我们以 cpsr 寄存器为例,展开 cpsr 如下图所示:

(x)= Variables	🛛 Breakpoints 🖋 E	xpressions	iiii Regis	sters ස			
Name	Hex	Decimal		Descrip	Description		
▼ ¹⁰¹⁰ cpsr	200001df	536871	391				
19191 n	Θ	0		Negal	tive condition code flag		
0101 Z	Θ	Θ		Zего (condition code flag		
1010 0101 C	1	1		Carry	condition code flag		
1010 0101 V	0	0		Overf	low condition code flag		
1010 q	0	0		Cumu	lative saturation flag		
1010 it	00	0		If-The	n execution state bits		
1010 j	0	0		Jazell	e bit		
1010 ge	Θ	0		SIMD	Greater than or Equal flags		
1010 e	Θ	0		Endia	nness execution state bit		
1010 a	1	1		Async	hronous abort disable bit		
1010	1	1		Interr	upt disable bit		
1010 F	1	1		Fast i	nterrupt disable bit		
1010 L	Θ	0		Thum	b execution state bit		
1919 m	lf	31		Mode	field		

图 13.1.25 Vitis 调试界面中的 cpsr 寄存器

可以看到 cpsr 中各个标志位的内容,如果不清楚 cpsr 寄存器的 n 标志位代表什么意思,可以看下 Desciption 栏的内容。其他的寄存器如果不清楚的也可以照例参详。

可以看到 Vitis 真的很方便,我们在调试 uboot 的时候,可以一目了然的知道寄存器的变化,不用自己计算,也不用翻手册查找寄存器中的标志位代表什么意思。另外调试界面是可以调整的,可以按照自己想要的方式布局调试界面,此处笔者限于篇幅就不介绍了。

需要说明的是在完成 relocate_code 函数实现代码动态重定位后, Vitis 就不能很好的调试 了,这时只在反汇编文件中运行,不能对照源码,不过欣慰的是,这时已经完成动态重定位, 最令人苦恼的汇编已基本结束了,后面基本上都是C函数了。

13.2 链接脚本

13.2.1 链接脚本 u-boot.lds 详解

要分析 uboot 的启动流程,首先要找到"入口",找到第一行程序在哪里。程序的链接是 由链接脚本来决定的,所以通过链接脚本可以找到程序的入口。如果没有编译过 uboot 的话链 接脚本为 arch/arm/cpu/u-boot.lds。但是这个不是最终使用的链接脚本,最终的链接脚本是在 这个链接脚本的基础上生成的。编译一下 uboot 后就会在 uboot 根目录下生成 u-boot.lds 文件, 如下图所示:
冬一度

须	预航者 ZYNQ 之嵌入式 Linux 开发指南 ②正点原子						
原子	子哥在线教学	±: www.yua	nzige.com	论坛:www.openedv.com/	forum.php		
	wmq@Linux:~ api arch board cmd config.mk configs disk doc drivers wmq@Linux:~	<pre>~/petalinux/ dts env examples fs include Kbuild Kconfig lib Licenses lscript.ld ~/petalinux/ t.lds, 内容如</pre>	Yuboot_start/u MAINTAINERS Makefile net post README README.txt scripts spl System.map test Yuboot_start/u 图下:	boot/src\$ ls tools u-boot u-boot.bin u-boot.cfg u-boot.cfg.configs uboot_Debug.build.ui.log u-boot.dtb u-boot.dtb.bin u-boot-dtb.bin u-boot-dtb.img u-boot.elf boot/src\$ 13.2.1 链接脚本	<pre>u-boot-elf.lds u-boot.elf.o u-boot.img u-boot.lds u-boot.map u-boot-nodtb.bin u-boot.srec u-boot.sym Xilinx.spec</pre>		
			示例在	代码 u-boot.lds 文件代码			
	 OUTPUT_I OUTPUT_A ENTRY(_si SECTIONS { = 0x00000 = ALIGN .text : { *(imag *(.vectors arch/arm/ 	FORMAT("elf3 ARCH(arm) tart) 00000; (4); ge_copy_start) s)	32-littlearm", "eli	f32-littlearm", "elf32-littlearm")			
	12 arch/arm/	cpu/armv7/star	t.o (.text*)				
	13 } 14 . efi run	time_start · {					
	15 *(efi_r	runtime_start)					
	16 }	_ /					
	17 .efi_runtin	ne : {					
	18 *(.text.efi	_runtime*)					
	19 *(.rodata.	efi runtime*)					

- 20 *(.data.efi_runtime*)

```
21 }
```

```
22 .__efi_runtime_stop : {
```

```
23 *(.__efi_runtime_stop)
```

```
24 }
```

```
25 .text_rest :
```

```
26 {
```

```
27 *(.text*)
```

```
28 }
```

```
29 . = ALIGN(4);
```

```
30 .rodata : { *(SORT_BY_ALIGNMENT(SORT_BY_NAME(.rodata*))) }
```



```
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
    31 . = ALIGN(4);
    32 .data : {
    33 *(.data*)
    34 }
    35 . = ALIGN(4);
    36 . = .;
    37 . = ALIGN(4);
    38 .u_boot_list : {
    39 KEEP(*(SORT(.u_boot_list*)));
    40 }
    41 . = ALIGN(4);
   42 .efi_runtime_rel_start :
   43 {
    44 *(.__efi_runtime_rel_start)
    45 }
    46 .efi_runtime_rel : {
    47 *(.rel*.efi_runtime)
    48 *(.rel*.efi_runtime.*)
    49 }
    50 .efi_runtime_rel_stop :
    51 {
    52 *(.__efi_runtime_rel_stop)
    53 }
    54 . = ALIGN(4);
    55 .image_copy_end :
    56 {
    57 *(.__image_copy_end)
    58 }
    59 .rel_dyn_start :
    60 {
    61 *(.__rel_dyn_start)
    62 }
    63 .rel.dyn : {
    64 *(.rel*)
    65 }
    66 .rel_dyn_end :
    67 {
    68 *(.__rel_dyn_end)
    69 }
    70 .end :
    71 {
```



```
原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php
   72 *(.__end)
   73 }
   74 _image_binary_end = .;
   75 .bss_start __rel_dyn_start (OVERLAY) : {
   76 KEEP(*(.__bss_start));
   77 __bss_base = .;
   78 }
   79 .bss __bss_base (OVERLAY) : {
   80 *(.bss*)
   81 . = ALIGN(4);
   82 __bss_limit = .;
   83 }
   84 .bss_end __bss_limit (OVERLAY) : {
   85 KEEP(*(.__bss_end));
   86 }
   87 /DISCARD/: { *(.dynsym) }
   88 /DISCARD/: { *(.dynbss*) }
   89 /DISCARD/: { *(.dynstr*) }
   90 /DISCARD/: { *(.dynamic*) }
   91 /DISCARD/: { *(.plt*) }
   92 /DISCARD/: { *(.interp*) }
   93 /DISCARD/: { *(.gnu*) }
   94 /DISCARD/: { *(.ARM.exidx*) }
   95 /DISCARD/: { *(.gnu.linkonce.armexidx.*) }
   96 }
   链接脚本可以决定生成的 u-boot 镜像中所有.o 文件和.a 文件的链接地址。下面简单说明
上述链接脚本中各行代码的含义。
   第1行,指定输出可执行文件是32位ARM指令、小端模式的ELF格式。
   第2行,指定输出可执行文件的平台为 ARM
   第3行为代码入口点:__start, __start 在文件 arch/arm/lib/vectors.S 中有定义, 具体代码见
示例代码 13.3.1 vectors.S 代码段。
   第6行,指明目标代码的起始地址从 0x0 位置开始,"."代表当前位置。
   第7行,表示此处4字节对齐。
   对于第10行,使用如下命令在 uboot 中查找"__image_copy_start":
   grep -nR "__image_copy_start"
   搜索结果如下图所示:
```

u-boot.map:1337: *(image_copy_start)	
u-boot.map:1338: . image copy start	
u-boot.map:1340: 0x000000004000000	image_copy_start
grep: oe-workdir/pseudo/pseudo.socket: 没有那个设备或地址	

图 13.2.2 查找结果

打开 u-boot.map, 找到如下图所示位置:

原子哥在线教学:w	ww.yuanzige.com 论坛	:www.openedv.com/forum.php			
1332 节 .text 的地址设置到 0x4000000					
1333	0x00000000000000000	. = 0×0			
1334	0x00000000000000000	. = ALIGN (0x4)			
1335					
1336 .text	0x000000004000000	0x3a8			
1337 *(image_copy_start)				
1338im	age_copy_start				
1339	0x0000000004000000	0x0 arch/arm/lib/built-in.o			
1340	0x000000004000000	image_copy_start			
1341 *(.ve	ctors)				
1342 .vect	ors 0x000000004000000	0x2e8 arch/arm/lib/built-in.o			
1343	0x000000004000000	_start			
1344	0x000000004000020	_undefined_instruction			
1345	0x0000000004000024	_software_interrupt			
1346	0x000000004000028	_prefetch_abort			
1347	0x000000000400002c	_data_abort			
1348	0x0000000004000030	_not_used			
1349	0x0000000004 <u>000034</u>	_irq			
1350	0x000000004000038	_fiq			
1351	0x000000004000040	IRQ_STACK_START_IN			

正点原子

图 13.2.3 u-boot.map

u-boot.map 是 uboot 的映射文件,可以从该文件看到某个文件或者函数链接到了哪个地址,从上图的1338行可以看到__image_copy_start为0x4000000,而.text的起始地址也是0x4000000。

第 11 行是 vectors 段, vectors 段保存中断向量表。从图 13.2.3 可以看出, vectors 段的起始地址也是 0x4000000, 说明整个 uboot 的起始地址就是 0x4000000。

第12行将 arch/arm/cpu/armv7/start.o 编译出来的代码放到中断向量表后面。

第30行,只读数据段。

第32行,数据段。

第75行, bss_start 标号指向 bss 段的开始位置。

第 79 行, bss 段。BSS 是英文 Block Started by Symbol 的缩写。BSS 段通常是指用于存放 程序中未初始化的全局变量的一块内存区域,该段中的变量在使用前由系统初始化为 0。

在 u-boot.lds 中有一些跟地址有关的"变量"需要我们注意一下,后面分析 u-boot 源码的 时候会用到,这些变量要最终编译完成才能确定的。比如笔者编译完成以后这些"变量"的 值如下表所示:

	数值 数值	描述
image_copy_start	0x4000000	uboot 拷贝的首地址
image_copy_end	0x409edcc	uboot 拷贝的结束地址
rel_dyn_start	0x409edcc	.rel.dyn 起始地址
rel_dyn_end	0x40af1c4	.rel.dyn 结束地址
_image_binary_end	0x40af1c4	二进制镜像结束地址
bss_start	0x409edcc	bss 段起始地址
bss_end	0x40c0a80	bss 段结束地址

表 13.2.1 uboot 相关变量表

上表中的"变量"值可以在 u-boot.map 文件中查找,表中除了__image_copy_start 以外, 其他的变量值每次编译的时候可能会变化,如果修改了 uboot 代码、修改了 uboot 配置、选用 不同的优化等级等都会影响到这些值,所以一切以实际值为准。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

13.3 U-Boot 启动流程详解

13.3.1 reset 函数源码详解

从 u-boot.lds 文件中我们已经知道了入口点是 arch/arm/lib/vectors.S 文件中的_start,代码 如下:

示例代码 13.3.1 vectors.S 代码段

- 1 /* SPDX-License-Identifier: GPL-2.0+ */
- 2 /*
- 3 * vectors Generic ARM exception table code
- 4 *
- 5 * Copyright (c) 1998 Dan Malek <dmalek@jlc.net>
- 6 * Copyright (c) 1999 Magnus Damm <kieraypc01.p.y.kie.era.ericsson.se>
- 7 * Copyright (c) 2000 Wolfgang Denk <wd@denx.de>
- 8 * Copyright (c) 2001 Alex Züpke <azu@sysgo.de>
- 9 * Copyright (c) 2001 Marius Gröger <mag@sysgo.de>
- 10 * Copyright (c) 2002 Alex Züpke <azu@sysgo.de>
- 11 * Copyright (c) 2002 Gary Jennejohn <garyj@denx.de>
- 12 * Copyright (c) 2002 Kyle Harris <kharris@nexus-tech.net>
- 13 */
- 14

```
15 #include <config.h>
```

- 16
- 17 /*
- 18 * A macro to allow insertion of an ARM exception vector either
- 19 * for the non-boot0 case or by a boot0-header.
- 20 */
- 21 .macro ARM_VECTORS
- 22 #ifdef CONFIG_ARCH_K3
- 23 ldr pc, _reset
- 24 #else

```
25
     b reset
```

- 26 #endif
- 27 ldr pc, _undefined_instruction
- 28 ldr pc, _software_interrupt
- 29 ldr pc, _prefetch_abort
- 30 ldr pc, _data_abort
- ldr pc, _not_used 31
- 32 ldr pc, _irq
- 33 ldr pc, _fiq
- 34 .endm
- 35



原子哥在线教学: www.yuanzige.com 论坛:www.openedy.com/forum.php 36 37 /* 39 * 40 * Symbol _start is referenced elsewhere, so make it global 41 * 43 */ 44 45 .globl start 46 47 /* 49 * 50 * Vectors have their own section so linker script can map them easily 51 * 53 */ 54 55 .section ".vectors", "ax" 56 57 #if defined(CONFIG_ENABLE_ARM_SOC_BOOT0_HOOK) 58 /* 59 * Various SoCs need something special and SoC-specific up front in 60 * order to boot, allow them to set that in their boot0.h file and then 61 * use it here. 62 * 63 * To allow a boot0 hook to insert a 'special' sequence after the vector 64 * table (e.g. for the socfpga), the presence of a boot0 hook supresses 65 * the below vector table and assumes that the vector table is filled in 66 * by the boot0 hook. The requirements for a boot0 hook thus are: 67 * (1) defines '_start:' as appropriate 68 * (2) inserts the vector table using ARM_VECTORS as appropriate 69 */ 70 #include <asm/arch/boot0.h> 71 #else 72 73 /* 74 ********* 75 * 76 * Exception vectors as described in ARM reference manuals



<pre>wmq@Linux:~/petalinux/uboot_start/uboot/src\$ grep -nR "CONFIG_ENABLE_ARM_SOC_BOOT0_HOOK"</pre>
configs/gose_defconfig:3:CONFIG_ENABLE_ARM_SOC_BOOT0_HOOK=y
configs/bcm28155_w1d_defconfig:2:CONFIG_ENABLE_ARM_SOC_BOOT0_HOOK=y
configs/stout_defconfig:3:CONFIG_ENABLE_ARM_SOC_BOOT0_HOOK=y
configs/blanche_defconfig:3:CONFIG_ENABLE_ARM_SOC_BOOT0_HOOK=y
configs/bcm28155_ap_defconfig:2: CONFIG_ENABLE_ARM_SOC_BOOT0_HOOK=y
configs/lager_defconfig:3:CONFIG_ENABLE_ARM_SOC_BOOT0_HOOK=y
configs/koelsch_defconfig:3:CONFIG_ENABLE_ARM_SOC_BOOT0_HOOK=y
configs/silk_defconfig:3:CONFIG_ENABLE_ARM_SOC_BOOT0_HOOK=y
<pre>configs/porter_defconfig:3:CONFIG_ENABLE_ARM_SOC_BOOT0_HOOK=y</pre>
configs/alt_defconfig:3:CONFIG_ENABLE_ARM_SOC_BOOT0_HOOK=y
configs/bcm23550_w1d_defconfig:4:CONFIG_ENABLE_ARM_SOC_B00T0_H00K=y
config:50:# CONFIG_ENABLE_ARM_SOC_BOOTO_HOOK is not set
arch/arm/lib/vectors.S:57: #if defined(CONFIG_ENABLE_ARM_SOC_BOOT0_HOOK)

图 13.3.1 CONFIG ENABLE ARM SOC BOOTO HOOK

直接看"#else"部分。第83行 start 开始的是中断向量表, CONFIG_SYS_DV_NOR_BOOT_CFG 未定义,直接看第 87 行的 ARM_VECTORS, 对应第 21~33 行的中断向量表。当一个异常或中断发生时, CPU 根据异常, 在异常(中断)向量表 中找到对应的异常向量,然后执行异常向量处的跳转指令,CPU 跳转到对应的异常处理程序 执行。使用 grep -Nr "CONFIG ARCH K3"命令查找 CONFIG ARCH K3 变量在 uboot 中没有 定义,所以当uboot启动后程序会跳转到第25行的复位异常向量的指令"breset"处,紧接着 执行 reset 代码段。标号 reset 在 arch/arm/cpu/armv7/start.S 中定义,代码如下:

示例代码 13.3.2 start.S 代码段

21	/**************************************
22	*
23	* Startup Code (reset vector)
24	*
25	* Do important init only if we don't start from memory!
26	* Setup memory and board specific bits prior to relocation.
27	* Relocate armboot to ram. Setup stack.
28	*
29	***************************************
30	
31	.globl reset



领航者 ZYNQ 之嵌入式 Linux 开发指南 原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php .globl save_boot_params_ret 32 .type save_boot_params_ret,%function 33 34 #ifdef CONFIG_ARMV7_LPAE .global switch_to_hypervisor_ret 35 36 #endif 37 38 reset: 39 /* Allow the board to save important registers */ 40 b save_boot_params 第38行就是标号 reset。注:为了方便,后面对标号和 ENTRY 统一称为函数。 第40行从标号 reset 跳转到了 save_boot_params 函数,而 save_boot_params 函数同样定义 在 start.S 中, 定义如下: 示例代码 start.S 代码段 111 * 112 * void save_boot_params(u32 r0, u32 r1, u32 r2, u32 r3) 113 * __attribute__((weak)); 114 * 115 * Stack pointer is not yet initialized at this moment 116 * Don't save anything to stack even if compiled with -O0 117 * 119 ENTRY(save_boot_params) 120 b save_boot_params_ret @ back to my caller save_boot_params 函数也是只有一句跳转语句,跳转到 save_boot_params_ret 函数, save_boot_params_ret 函数代码如下: 示例代码 start.S 代码段 41 save_boot_params_ret: 42 #ifdef CONFIG_ARMV7_LPAE 43 /* 44 * check for Hypervisor support 45 */ 46 mrc p15, 0, r0, c0, c1, 1 @ read ID PFR1 and r0, r0, #CPUID_ARM_VIRT_MASK @ mask virtualization bits 47 cmp r0, #(1 << CPUID_ARM_VIRT_SHIFT) 48 49 beq switch_to_hypervisor 50 switch_to_hypervisor_ret: 51 #endif 52 /* * disable interrupts (FIQ and IRQ), also set the cpu to SVC32 mode, 53 54 * except if in HYP mode already



原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php

- */ 55
- 56 mrs r0, cpsr
- and r1, r0, #0x1f 57 @ mask mode bits
- teq r1, #0x1a 58 @ test for HYP mode
- 59 bicne r0, r0, #0x1f @ clear all mode bits
- orrne r0, r0, #0x13 @ set SVC mode 60
- orr r0, r0, #0xc0 @ disable FIQ and IRQ 61
- 62 msr cpsr,r0

第 42 行表示是否定义宏 CONFIG_ARMV7_LPAE。LPAE(Large Physical Address Extensions)是 ARMv7 系列的一种地址扩展技术,可以让 32 位的 ARM 最大能支持到 1TB 的 内存空间,由于嵌入式 ARM 需求的内存空间一般不大,所以一般不使用 LPAE 技术。

第56行,读取寄存器 cpsr 中的值,并保存到 r0 寄存器中。

第57行,将寄存器r0中的值与0X1F进行与运算,结果保存到r1寄存器中,目的是提取 cpsr 的 bit0~bit4 这 5 位, 这 5 位为 M4 M3 M2 M1 M0, M[4:0]这五位用来设置处理器的工作模 式,如下表所示:

衣 15.5.1 Cortex-A9 上作 候 式				
M[4:0]	模式			
10000	User(usr)			
10001	FIQ(fiq)			
10010	IRQ(irq)			
10011	Supervisor(svc)			
10110	Monitor(mon)			
10111	Abort(abt)			
11010	Hyp(hyp)			
11011	Undefined(und)			
11111	System(sys)			

10 工作性中

第58行,判断rl寄存器的值是否等于0x1A(0b11010),也就是判断当前处理器模式是否 处于 Hyp 模式。

第 59 行,如果 r1 和 0x1A 相等,也就是 CPU 处于 Hyp 模式的话就将 r0 寄存器的 bit0~4 进行清零,其实就是清除模式位。

第 60 行,如果处理器不处于 Hyp 模式的话就将 r0 的寄存器的值与 0x13 进行或运算, 0x13=0b10011,也就是设置处理器进入 SVC 模式。

第 61 行, r0 寄存器的值再与 0xC0 进行或运算, 那么 r0 寄存器此时的值就是 0xD3, cpsr 的 I 为和 F 位分别控制 IRO 和 FIO 这两个中断的开关,设置为 1 就关闭了 FIO 和 IRO。

第 62 行,将 r0 寄存器写回到 cpsr 寄存器中。完成设置 CPU 处于 SVC32 模式,并且关闭 FIQ 和 IRQ 这两个中断。

继续执行执行下面的代码:

示例代码 start.S 代码段

64 /*

65 * Setup vector:

66 * (OMAP4 spl TEXT_BASE is not 32 byte aligned.

```
原子哥在线教学: www.vuanzige.com
                                             论坛:www.openedv.com/forum.php
   67 * Continue to use ROM code vector only in OMAP4 spl)
   68 */
   69 #if !(defined(CONFIG_OMAP44XX) && defined(CONFIG_SPL_BUILD))
       /* Set V=0 in CP15 SCTLR register - for VBAR to point to vector */
   70
         mrc p15, 0, r0, c1, c0, 0 @ Read CP15 SCTLR Register
   71
   72
        bic r0, \#CR V @ V = 0
   73
         mcr p15, 0, r0, c1, c0, 0 @ Write CP15 SCTLR Register
   74
   75 #ifdef CONFIG HAS VBAR
        /* Set vector address in CP15 VBAR register */
   76
        ldr r0, =_start
   77
        mcr p15, 0, r0, c12, c0, 0 @Set VBAR
    78
```

正点原子

79 #endif

第 69 行,如果没有定义 CONFIG_OMAP44XX 和 CONFIG_SPL_BUILD 则条件成立。此 处简单的说下 SPL, SPL 是 Secondary Program Loader 的简称,也就是第二阶段程序加载器, 可用来初始化 DDR 内存并加载 uboot 到内存中,功能类似于 FSBL,可使用 CONFIG SPL BUILD 来选择编译,一般我们不使用 SPL。

第71行读取 CP15 中 c1 寄存器的值到 r0 寄存器中。

第72行, CR_V在 arch/arm/include/asm/system.h 中有如下所示定义:

#define CR_V (1 << 13) /* Vectors relocated to 0xffff0000 */

因此这一行的目的就是清除r0寄存器中的bit13,SCTLR寄存器结构如下图所示:



图 13.3.2 SCTLR 寄存器结构图

从图可以看出, bit13 为 V 位, 此位是向量表控制位, 当为 0 的时候向量表基地址为 0X00000000, 软件可以重定位向量表。为 1 的时候向量表基地址为 0XFFFF0000, 软件不能 重定位向量表。这里将 V 清零, 目的就是为了接下来的向量表重定位。

第73行将r0寄存器的值重写入到寄存器SCTLR中。

第78行设置r0寄存器的值为_start,_start就是整个uboot的入口地址,其值为0x4000000,相当于uboot的起始地址,因此0x4000000也是向量表的起始地址。

第78行将r0寄存器的值(向量表值)写入到CP15的c12寄存器中,也就是VBAR寄存器。因此第71~78行就是设置向量表重定位的。



原子哥在线教学:	www.yuanzige.com
	±1. 4 [→]

论坛:www.openedv.com/forum.php

代码继续往下执行:

示例代码 start.S 代码段

- 82 /* the mask ROM code should have PLL and others stable */
- 83 #ifndef CONFIG_SKIP_LOWLEVEL_INIT
- 84 #ifdef CONFIG_CPU_V7A
- 85 bl cpu_init_cp15
- 86 #endif
- 87 #ifndef CONFIG_SKIP_LOWLEVEL_INIT_ONLY
- 88 bl cpu_init_crit
- 89 #endif
- 90 #endif
- 91
- 92 bl _main

第 83 行如果没有定义 CONFIG_SKIP_LOWLEVEL_INIT 的话条件成立。此时我们执行 #ifdef CONFIG_CPU_V7A 语句,该语句的意思是判断 CONFIG_CPU_V7A 这个宏是否被定义, 若定义了,则执行 bl cpu_init_cp15 这条语句。

此处代码中的内容比较简单,就是分别调用函数 cpu_init_cp15、cpu_init_crit和_main。

函数 cpu_init_cp15 用来设置 CP15 相关的内容,比如关闭 MMU 啥的,此函数同样在 start.S 文件中定义的,代码如下:

```
示例代码 start.S 代码段
132 *
133 * cpu_init_cp15
134 *
135 * Setup CP15 registers (cache, MMU, TLBs). The I-cache is turned on unless
136 * CONFIG_SYS_ICACHE_OFF is defined.
137 *
139 ENTRY(cpu_init_cp15)
140 /*
141
    * Invalidate L1 I/D
142
   */
143 mov r0, #0
               @ set up for MCR
144 mcr p15, 0, r0, c8, c7, 0 @ invalidate TLBs
145 mcr p15, 0, r0, c7, c5, 0 @ invalidate icache
146 mcr p15, 0, r0, c7, c5, 6 @ invalidate BP array
147
    mcr p15, 0, r0, c7, c10, 4 @ DSB
148
    mcr p15, 0, r0, c7, c5, 4 @ ISB
149
150
    /*
151
    * disable MMU stuff and caches
```

```
原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php
   152
        */
   153 mrc p15, 0, r0, c1, c0, 0
   154 bic r0, r0, #0x00002000 @ clear bits 13 (--V-)
   155 bic r0, r0, #0x00000007 @ clear bits 2:0 (-CAM)
   156 orr r0, r0, #0x00000002 @ set bit 1 (--A-) Align
   157 orr r0, r0, #0x00000800 @ set bit 11 (Z---) BTB
   158 #if CONFIG_IS_ENABLED(SYS_ICACHE_OFF)
   159
        bic r0, r0, #0x00001000 @ clear bit 12 (I) I-cache
   160 #else
   161
        orr r0, r0, #0x00001000 @ set bit 12 (I) I-cache
   162 #endif
   163
        mcr p15, 0, r0, c1, c0, 0
   .....
                    @ back to my caller
   320 mov pc, r5
   321 ENDPROC(cpu_init_cp15)
    函数 cpu_init_cp15 都是一些和 CP15 有关的内容,我们不用关心,有兴趣的可以详细的看
一下。
    函数 cpu_init_crit 也在是定义在 start.S 文件中, 函数内容如下:
                                  示例代码 start.S 代码段
   326 *
   327 * CPU_init_critical registers
   328 *
   329 * setup important registers
   330 * setup memory timing
   331 *
   333 ENTRY(cpu_init_crit)
   334 /*
   335
        * Jump to board specific initialization...
        * The Mask ROM will have already initialized
   336
        * basic memory. Go here to bump up clock rate and handle
   337
        * wake up conditions.
   338
   339
        */
   340 b lowlevel_init
                       @ go setup pll,mux,memory
   341 ENDPROC(cpu_init_crit)
   可以看出函数 cpu_init_crit 内部仅仅是调用了函数 lowlevel_init, 接下来我们详细的分析
```

F点原子

一下 lowlevel_init 和_main 这两个函数。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

13.3.2 lowlevel_init 函数详解

函数 lowlevel_init 的目的一般是为了允许执行函数 board_init_f 所做的基本初始化,在文件 arch/arm/mach-zynq/lowlevel_init.S 中定义,内容如下:

示例代码 lowlevel_init.S 代码段

```
6 #include <asm-offsets.h>
   7 #include <config.h>
   8 #include <linux/linkage.h>
   9
    10 ENTRY(lowlevel_init)
    11
    12 /* Enable the the VFP */
    13 mrc p15, 0, r1, c1, c0, 2
    14 orr r1, r1, #(0x3 << 20)
    15 orr r1, r1, #(0x3 << 20)
    16 mcr p15, 0, r1, c1, c0, 2
    17 isb
    18 fmrx r1, FPEXC
    19 orr r1,r1, #(1<<30)
   20 fmxr FPEXC, r1
   21
   22 /* Move back to caller */
   23 mov pc, lr
    24
    25 ENDPROC(lowlevel_init)
    可以看到 ZYNQ 的 lowlevel_init 函数功能很简单。第 13-20 行使能 VFP。第 24 行返回调
用。
    函数调用路径如下图所示:
```



图 13.3.3 uboot 函数调用路径

从上图可知,接下来要执行的是 save_boot_params_ret 中的_main 函数,接下来分析_main 函数。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

13.3.3 _main 函数详解

```
main 函数定义在文件 arch/arm/lib/crt0.S 中, 函数内容如下:
                                    示例代码 crt0.S 代码段
87 /*
88 * entry point of crt0 sequence
89 */
90
91 ENTRY(_main)
92
93 /*
94 * Set up initial C runtime environment and call board_init_f(0).
95 */
96
97 #if defined(CONFIG_TPL_BUILD) && defined(CONFIG_TPL_NEEDS_SEPARATE_STACK)
98 ldr r0, =(CONFIG_TPL_STACK)
99 #elif defined(CONFIG_SPL_BUILD) && defined(CONFIG_SPL_STACK)
100 ldr r0, =(CONFIG_SPL_STACK)
101 #else
102 ldr r0, =(CONFIG_SYS_INIT_SP_ADDR)
103 #endif
104 bic r0, r0, #7 /* 8-byte alignment for ABI compliance */
105 mov sp, r0
106 bl board_init_f_alloc_reserve
107 mov sp, r0
108 /* set up gd here, outside any C code */
109 mov r9, r0
110 bl board_init_f_init_reserve
111
112 #if defined(CONFIG_SPL_EARLY_BSS)
113 SPL_CLEAR_BSS
114 #endif
115
116 mov r0, #0
117 bl board_init_f
118
119 #if ! defined(CONFIG_SPL_BUILD)
120
121 /*
122 * Set up intermediate environment (new sp and gd) and call
123 * relocate_code(addr_moni). Trick here is that we'll return
124 * 'here' but relocated.
```



正点原子



```
原子哥在线教学: www.yuanzige.com
                                       论坛:www.openedv.com/forum.php
   166 #if ! defined(CONFIG_SPL_BUILD)
   167 bl coloured LED init
   168 bl red_led_on
   169 #endif
   170 /* call board_init_r(gd_t *id, ulong dest_addr) */
   171 mov r0, r9 /* gd t */
   172 ldr r1, [r9, #GD_RELOCADDR] /* dest_addr */
   173 /* call board_init_r */
   174 #if CONFIG_IS_ENABLED(SYS_THUMB_BUILD)
   175 ldr lr, =board init r /* this is auto-relocated! */
   176 bx lr
   177 #else
   178 ldr pc, =board_init_r /* this is auto-relocated! */
   179 #endif
   180 /* we should not return here. */
   181 #endif
   182
   183 ENDPROC(_main)
    第102行,加载 CONFIG_SYS_INIT_SP_ADDR 到r0。CONFIG_SYS_INIT_SP_ADDR 在
```

```
include/configs/zynq-common.h 文件中有如下所示定义:
```

示例代码 zynq-common.h 代码段

```
223 #define CONFIG_SYS_INIT_RAM_ADDR 0xFFFF0000
```

224 #define CONFIG_SYS_INIT_RAM_SIZE 0x2000

 $\label{eq:config_sys_init_sp_addr} \mbox{(Config_sys_init_ram_addr} + \end{tabular}$

226 CONFIG_SYS_INIT_RAM_SIZE - \

227 GENERATED_GBL_DATA_SIZE)

还需要知道 GENERATED_GBL_DATA_SIZE 的值,在文件 include/generated/generic-asm-offsets.h 中有定义如下:

示例代码 generic-asm-offsets.h 代码段

```
1 #ifndef __GENERIC_ASM_OFFSETS_H__
2 #define __GENERIC_ASM_OFFSETS_H__
3 /*
4 * DO NOT MODIFY.
5 *
6 * This file was generated by Kbuild
7 */
8
9 #define GENERATED_GBL_DATA_SIZE 208 /* (sizeof(struct global_data) + 15) & ~15 @ */
10 #define GENERATED_BD_INFO_SIZE 80 /* (sizeof(struct bd_info) + 15) & ~15 @ */
11 #define GD_SIZE 200 /* sizeof(struct global_data) @ */
12 #define GD_BD 0 /* offsetof(struct global_data, bd) @ */
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

13 #define GD_MALLOC_BASE 148 /* offsetof(struct global_data, malloc_base) @ */

14 #define GD_RELOCADDR 44 /* offsetof(struct global_data, relocaddr) @ */

15 #define GD_RELOC_OFF 64 /* offsetof(struct global_data, reloc_off) @ */

16 #define GD_START_ADDR_SP 60 /* offsetof(struct global_data, start_addr_sp) @ */

17 #define GD_NEW_GD 68 /* offsetof(struct global_data, new_gd) @ */

18

19 #endif

GENERATED_GBL_DATA_SIZE=208, GENERATED_GBL_DATA_SIZE 的含义为 (sizeof(struct global_data) + 15) & ~15。

综上所述, CONFIG_SYS_INIT_SP_ADDR 值如下:

CONFIG_SYS_INIT_SP_ADDR = 0xFFFF0000+ 0x2000 - 208= 0xFFFF1F30,

此时 r0 的值为 0xFFFF1F30。

第104行,清除r0的bit[2:0]位。

第 105 行,读取 r0 寄存器的数据到寄存器 sp 里面,此时 sp=0xFFFF1F30,属于 ZYNQ 的内部 ram

第 106 行,调用函数 board_init_f_alloc_reserve,此函数有一个参数,参数为 r0 中的值, 也就是 0xFFFF1F30,此函数定义在文件 common/init/board init.c 中,内容如下:

示例代码 board_init.c 代码段

62 ulong board_init_f_alloc_reserve(ulong top)

63 {

64 /* Reserve early malloc arena */

65 #if CONFIG_VAL(SYS_MALLOC_F_LEN)

66 top -= CONFIG_VAL(SYS_MALLOC_F_LEN);

67 #endif

68 /* LAST : reserve GD (rounded up to a multiple of 16 bytes) */

69 top = rounddown(top-sizeof(struct global_data), 16);

70

71 **return** top;

72 }

函数 board_init_f_alloc_reserve 主要是从内存顶部地址分配保留空间以用作全局变量使用, 返回分配空间的底部地址。其中 SYS_MALLOC_F_LEN=0x800(在文件 include/generated/autoc onf.h 中定义), sizeof(struct global_data)=200(GD_SIZE 值)。

函数 board_init_f_alloc_reserve 是有返回值的,返回值为新的 top 值,此时 top=0xffff166 8。

继续回到_main 函数,第107行,将r0写入到 sp 里面,r0 保存着函数 board_init_f_alloc_ reserve 的返回值,所以这一句也就是设置 sp=0xffff1660(注因为需要16字节对齐,所以是0 xffff1660)。

第 109 行,将 r0 寄存器的值写到寄存器 r9 里面,因为 r9 寄存器存放着全局变量 gd 的地址,在文件 arch/arm/include/asm/global_data.h 中有如下图所示宏定义:



尿	、十司	T 任 线 教 字: W	www.yuanzige.com	形压:www.op	enedv.com	/torun	n.phj	р		
	109	<pre>#ifdef CONFI</pre>	IG_ARM64							
	110	#define DECI	LARE_GLOBAL_DATA_PTR	register	volatile	gd_t	*gd	asm	("x18"))
	111	#else								
	112	#define DECI	LARE_GLOBAL_DATA_PTR	register	volatile	gd_t	*gd	asm	("r9")	
	113	#endif								
	114	#endif								

图 13.3.4 DECLARE_GLOBAL_DATA_PTR 宏定义

从上图可以看出,uboot 中定义了一个指向 gd_t 的指针 gd, gd 存放在寄存器 r9 里面的,因此 gd 是个全局变量。gd_t 是个结构体,在 include/asm-generic/global_data.h 里面有定义,gd_t 定义如下:

示例代码 global_data.h 代码段

```
27 typedef struct global_data {
    28
         bd_t *bd;
    29
         unsigned long flags;
         unsigned int baudrate;
    30
    31
         unsigned long cpu_clk;
                                /* CPU clock in Hz!
                                                     */
    32
         unsigned long bus_clk;
         /* We cannot bracket this with CONFIG_PCI due to mpc5xxx */
    33
    34
         unsigned long pci_clk;
    35
         unsigned long mem_clk;
    36 #if defined(CONFIG_LCD) || defined(CONFIG_VIDEO)
         unsigned long fb_base; /* Base address of framebuffer mem */
    37
    38 #endif
    .....
    117 #ifdef CONFIG_LOG
          int log_drop_count; /* Number of dropped log messages */
    118
    119
          int default_log_level; /* For devices with no filters */
         struct list_head log_head; /* List of struct log_device */
    120
    121 #endif
    122 } gd_t;
    因此这一行代码就是设置 gd 所指向的位置,也就是 gd 指向 0xffff1660。
     回到_main 函数,第110行调用函数 board_init_f_init_reserve,此函数在文件 common/init
/board_init.c 中有定义,函数内容如下:
                                      示例代码 board_init.c 代码段
    116 void board_init_f_init_reserve(ulong base)
    117 {
    118 struct global_data *gd_ptr;
    119
    120 /*
    121 * clear GD entirely and set it up.
    122 * Use gd_ptr, as gd may not be properly set yet.
```

123 */



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

```
124
125 gd_ptr = (struct global_data *)base;
126 /* zero the area */
127 memset(gd_ptr, '\0', sizeof(*gd));
128 /* set GD unless architecture did it already */
129 #if !defined(CONFIG ARM)
130 arch_setup_gd(gd_ptr);
131 #endif
132 /* next alloc will be higher by one GD plus 16-byte alignment */
133 base += roundup(sizeof(struct global data), 16);
134
135 /*
136 * record early malloc arena start.
137 * Use gd as it is now properly set for all architectures.
138 */
139
140 #if CONFIG_VAL(SYS_MALLOC_F_LEN)
141 /* go down one 'early malloc arena' */
142 gd->malloc_base = base;
143 /* next alloc will be higher by one 'early malloc arena' size */
144 base += CONFIG_VAL(SYS_MALLOC_F_LEN);
145 #endif
146
147 if (CONFIG_IS_ENABLED(SYS_REPORT_STACK_F_USAGE))
```

board_init_f_init_stack_protection(); 148

149}

可以看出,此函数用于初始化已分配的全局变量(gd)的保留空间,其实就是清零处理。 另外,此函数还设置了 gd->malloc_base 为 gd 基地址+gd 大小=0xffff1660+200=0xffff1728,并 做 16 字节对齐,最终 gd->malloc_base=0xffff1730,这个也就是 early malloc 的起始地址。

继续回到_main函数,第116行设置R0为0。

第 117 行,调用 board_init_f 函数,此函数定义在文件 common/board_f.c 中。主要用来初 始化 DDR, 定时器, 完成代码拷贝等等, 此函数我们后面进行详细的分析。

第 127 行,重新设置环境(sp 和 gd)、获取 gd->start_addr_sp 的值赋给 r0,在函数 board_init_f 中会初始化 gd 的所有成员变量,其中 gd->start_addr_sp=0x3eb19ec0,所以这里相 当于设置 r0=gd->start_addr_sp=0x3eb19ec0。0x3eb19ec0 是 DDR 中的地址, 说明新的 sp 和 gd 将会存放到 DDR 中,而不是内部的 RAM 了。GD_START_ADDR_SP=60,参考示例代码 generic-asm-offsets.h.

第 128-129 行, r0 做 8 字节对齐并赋值给 sp。

第 130 行, 获取 gd->bd 的地址赋给 r9, 此时 r9 存放的是老的 gd, 这里通过获取 gd->bd 的地址来计算出新的 gd 的位置。GD_BD=0,参考示例代码 generic-asm-offsets.h。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

第 131 行,新的 gd 在 bd 下面,所以 r9 减去 gd 的大小就是新的 gd 的位置,获取到新的 gd 的位置以后赋值给 r9。

第133行,设置 lr 寄存器为 here,这样后面执行其他函数返回的时候就返回到了第141行的 here 位置处。

第 134,读取 gd->reloc_off 的值给 r0 寄存器,GD_RELOC_OFF=64,参考示例代码 generic-asm-offsets.h。

第 135 行, lr 寄存器的值加上 r0 寄存器的值,重新赋值给 lr 寄存器。因为接下来要重定 位代码,也就是把代码拷贝到新的地方去(现在的 uboot 存放的起始地址为 0x4000000,下面要 将 uboot 拷贝到 DDR 最后面的地址空间处,将 0x4000000 开始的内存空出来),其中就包括 here,因此 lr 中的 here 要使用重定位后的位置。

第 139 行,读取 gd->relocaddr 的值赋给 r0 寄存器,此时 r0 寄存器就保存着 uboot 要拷贝 的目的地址,为 0x3ff3a000。GD_RELOCADDR=44,参考示例代码 generic-asm-offsets.h。

第 140 行,调用函数 relocate_code,也就是代码重定位函数,此函数负责将 uboot 拷贝到 新的地方去,此函数定义在文件 arch/arm/lib/relocate.S 中稍后会详细分析此函数。

第 146 行,调用函数 relocate_vectors,对中断向量表做重定位,此函数定义在文件 arch/arm/lib/relocate.S 中,稍后会详细分析此函数。

第 150 行,调用函数 c_runtime_cpu_setup,此函数定义在文件 arch/arm/cpu/armv7/start.S 中,函数作用为如果启用了 I-cache,则使其无效。

第152~164行,清除BSS段。

第 171~172 行,设置函数 board_init_r 的两个参数,函数 board_init_r 声明如下:

board_init_r(gd_t *id, ulong dest_addr)

第一个参数是 gd,因此读取 r9 保存到 r0 里面。

第二个参数是目的地址,因此 r1= gd->relocaddr。

第 178 行,调用函数 board_init_r,此函数定义在文件 common/board_r.c 中,稍后会详细的分析此函数。

这个就是_main 函数的运行流程,在_main 函数里面主要调用了 board_init_f、 relocate_code、relocate_vectors 和 board_init_r 这4个函数,接下来依次来看下这4个函数的作用。

13.3.4 board_init_f 函数详解

_main 中会调用 board_init_f 函数, board_init_f 函数主要有两个工作:

①、初始化一系列外设,比如串口、定时器,或者打印一些消息等。

②、初始化 gd 的各个成员变量,uboot 会将自己重定位到 DRAM 最后面的地址区域,也 就是将自己拷贝到 DRAM 最后面的内存区域中。这么做的目的是给 Linux 腾出空间,防止 Linux kernel 覆盖掉 uboot,将 DRAM 前面的区域完整的空出来。在拷贝之前肯定要给 uboot 各部分分配好内存位置和大小,比如 gd 应该存放到哪个位置,malloc 内存池应该存放到哪个 位置等等。这些信息都保存在 gd 的成员变量中,因此要对 gd 的这些成员变量做初始化。最 终形成一个完整的内存"分配图",在后面重定位 uboot 的时候就会用到这个内存"分配图"。

此函数定义在文件 common/board_f.c 中定义,代码如下:

示例代码 board_f.c 代码段

1012 void board_init_f(ulong boot_flags)



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

```
1013 {
1014 gd->flags = boot flags;
1015 gd->have_console = 0;
1016
1017 if (initcall_run_list(init_sequence_f))
1018
       hang();
1019
1020 #if !defined(CONFIG_ARM) && !defined(CONFIG_SANDBOX) && \
       !defined(CONFIG_EFI_APP) && !CONFIG_IS_ENABLED(X86_64) && \
1021
1022
       !defined(CONFIG ARC)
1023 /* NOTREACHED - jump_to_copy() does not return */
1024 hang();
1025 #endif
1026 }
第 1014 行,初始化 gd->flags=boot_flags=0。
第1015行,设置gd->have_console=0。
重点在第 1017 行。通过函数 initcall_run_list 来运行初始化序列 init_sequence_f 里面的一
```

系列函数。init_sequence_f 里面包含了一系列的初始化函数, init_sequence_f 也是定义在文件 common/board_f.c 中,由于 init_sequence_f 的内容比较长,里面有大量的条件编译代码,这里 为了缩小篇幅,将条件编译部分删除掉了,去掉条件编译以后的 init_sequence_f 定义如下:

/************************去掉条件编译语句后的 init_sequence_f*****************/

- 1 static const init_fnc_t init_sequence_f[] = {
- 2 setup_mon_len,
- 3 fdtdec_setup,
- 4 initf_malloc,
- 5 log_init,
- 6 initf_bootstage, /* uses its own timer, so does not need DM */
- 7 initf_console_record,
- 8 arch_cpu_init, /* basic arch cpu dependent setup */
- 9 mach_cpu_init, /* SoC/machine dependent CPU setup */
- 10 initf_dm,
- 11 arch_cpu_init_dm,
- 12 timer_init, /* initialize timer */
- 13 env_init, /* initialize environment */
- 14 init_baud_rate, /* initialze baudrate settings */
- 15 serial_init, /* serial communications setup */
- 16 console_init_f, /* stage 1 init of console */
- 17 display_options, /* say that we are here */
- 18 display_text_info, /* show debugging info if required */

```
19 show_board_info,
```

20 INIT_FUNC_WATCHDOG_INIT



~ ~ ~ ~ ~ ~	
原子哥	午在线教学:www.yuanzige.com 论坛:www.openedv.com/forum.php
21	INIT_FUNC_WATCHDOG_RESET
22	init_func_i2c,
23	announce_dram_init,
24	dram_init, /* configure available RAM banks */
25	INIT_FUNC_WATCHDOG_RESET
26	INIT_FUNC_WATCHDOG_RESET
27	INIT_FUNC_WATCHDOG_RESET
28	/*
29	* Now that we have DRAM mapped and working, we can
30	* relocate the code and continue running from DRAM.
31	*
32	* Reserve memory at end of RAM for (top down in that order):
33	* - area that won't get touched by U-Boot and Linux (optional)
34	* - kernel log buffer
35	* - protected RAM
36	* - LCD framebuffer
37	* - monitor code
38	* - board info struct
39	*/
40	setup_dest_addr,
41	reserve_round_4k,
42	reserve_mmu,
43	reserve_video,
44	reserve_trace,
45	reserve_uboot,
46	reserve_malloc,
47	reserve_board,
48	setup_machine,
49	reserve_global_data,
50	reserve_fdt,
51	reserve_bootstage,
52	reserve_arch,
53	reserve_stacks,
54	dram_init_banksize,
55	show_dram_config,
56	display_new_sp,
57	INIT_FUNC_WATCHDOG_RESET
58	reloc_fdt,
59	reloc_bootstage,
60	setup_reloc,
61	NULL,

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

正点原子

<mark>62</mark> };

接下来分析以上函数执行完以后的结果:

第2行, setup_mon_len函数设置 gd 的 mon_len 成员变量,变量值为__bss_end -_start,也就是整个代码的长度。代码长度为 0x40c0a80-0x409edcc=0X21cb4。

第3行,调用 fdtdec_setup 函数配置 gd->fdt_blob 指针(即 device tree 所在的存储位置)。 对于 ARM 平台,uboot 的 Makefile 会通过连接脚本,将 dtb 文件打包到 u-boot image 的 "dtb dt begin"位置处。

第4行, initf_malloc 函数初始化 gd 中跟 malloc 有关的成员变量,比如 malloc_limit,此 函数会设置 gd->malloc_limit = CONFIG_SYS_MALLOC_F_LEN=0x800。malloc_limit 表示 malloc 内存池大小。

第5行, log_init 函数将 struct log_driver 结构体加入到 gd->log_head 的循环链表中,并初 始化 gd->default_log_level。

第6行, initf_bootstage函数为gd->bootstage分配空间,并初始化gd->bootstage。

第 7 行, initf_console_record, 如果定义了宏 CONFIG_CONSOLE_RECORD 和宏 CONFIG_SYS_MALLOC_F_LEN,此函数就会调用函数 console_record_init,但是领航者的 uboot 没有定义宏 CONFIG_CONSOLE_RECORD,所以此函数直接返回 0。

第8行, arch_cpu_init 函数, 初始化架构相关的内容, CPU 级别的操作。

第9行, mach_cpu_init 函数, 初始化 SOC 相关的内容, CPU 级别的操作。

第 10 行, initf_dm 函数, 驱动模型有关的初始化操作, 如果定义了 CONFIG_DM, 则调用 dm_init_and_scan 初始化并扫描系统所有的 device。如果定义了 CONFIG_TIMER_EARLY, 调用 dm_timer_init 初始化 driver model 所需的 timer。

第11行, arch_cpu_init_dm函数未实现。

第 12 行,timer_init,初始化定时器,Cortex-A9 内核有一个定时器,这里初始化的就是Cortex-A 内核的定时器。通过这个定时器来为 uboot 提供时间。关于 Cortex-A 内部定时器的详细内容,请参考文档《ARM ArchitectureReference Manual ARMv7-A and ARMv7-R edition.pdf》的"Chapter B8 The Generic Timer"章节。

第 13 行, env_init 函数是和环境变量有关的,设置 gd 的成员变量 env_addr,也就是环境 变量的保存地址。

第 14 行, init_baud_rate 函数用于初始化波特率,根据 CONFIG_BAUDRATE 的值来初始化 gd->baudrate。

第15行, serial_init, 初始化串口。

第16行, console_init_f, 设置 gd->have_console 为1, 表示有1个控制台,此函数也将前面暂存在缓冲区中的数据通过控制台打印出来。

第17行、display_options,通过串口输出信息,如下图所示:

U-Boot 2020.01 (Aug 23 2023 - 10:49:37 +0800)

图 13.3.5 串口信息输出

第19行, show_board_info 函数用于打印开发板信息,会调用 checkboard 函数,结果如下图所示:

原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php

正点原子

CPU: Zynq 7z020 Silicon: v3.1 Model: Alientek Navigator Zynq Development Board

图 13.3.6 开发板信息

第20行,INIT_FUNC_WATCHDOG_INIT,初始化看门狗,对于ZYNQ而言未定义。

第21行, INIT_FUNC_WATCHDOG_RESET, 复位看门狗, 对于 ZYNQ 而言未定义。

第 22 行, init_func_i2c 函数用于初始化 I2C,

第23行, announce_dram_init, 此函数很简单, 就是输出字符串 "DRAM:"

第 24 行,dram_init,并非真正的初始化 DDR,只是设置 gd->ram_size 的值并检测 ecc,对于正点原子 ZYNQ 开发板 7020 核心板来说,DDR 大小为 1GB,所以 gd->ram_size 的值为 0x40000000。

第 40 行, setup_dest_addr 函数,设置目的地址,设置 gd->ram_top, gd->relocaddr 的值。gd->ram_top = 0x40000000, gd->relocaddr = 0x40000000,也就是说重定位后的最高地址为 0x40000000。

第 41 行, reserve_round_4k 函数用于对 gd->relocaddr 做 4KB 对齐,因为 gd->relocaddr=0x40000000,已经是 4K 对齐了,调整后不变。

第42行, reserve_mmu, 留出 MMU的 TLB 表的位置,分配 MMU的 TLB 表内存以后会对 gd->relocaddr 做 64K 字节对齐。完成以后 gd->arch.tlb_size、gd->arch.tlb_addr 和 gd->relocaddr 的值如下:

gd->arch.tlb_size= 0X4000 //MMU 的 TLB 表大小 gd->arch.tlb_addr=0x3fff0000 //MMU 的 TLB 表起始地址, 64KB 对齐以后 gd->relocaddr=0x3fff0000 //relocaddr 地址

第43行, reserve_video函数为视频显示开辟帧空间, zynq未用到。

第44行, reserve_trace 函数, 留出跟踪调试的内存, zynq 未用到。

第 45 行, reserve_uboot, 留出重定位后的 uboot 所占用的内存区域, uboot 所占用大小由 gd->mon_len 所指定, 留出 uboot 的空间以后还要对 gd->relocaddr 做 4K 字节对齐,并且重新设置 gd->start_addr_sp, 结果如下:

gd->mon_len = 0x000b58e8 (使能 Debug 时为 0xcb1ec)

gd->start_addr_sp = 0x3ff3a000 (使能 Debug 时为 0x3ff24000)

gd->relocaddr = 0x3ff3a000 (使能 Debug 时为 0x3ff24000)

第 46 行, reserve_malloc, 留出 malloc 区域, 调整 gd->start_addr_sp 位置, malloc 区域由宏 TOTAL_MALLOC_LEN 定义, 宏定义如下:

#define TOTAL_MALLOC_LEN (CONFIG_SYS_MALLOC_LEN + CONFIG_ENV_SIZE)

宏CONFIG_SYS_MALLOC_LEN值为20MB=0x1400000,宏CONFIG_ENV_SIZE值为128KB=0x20000,因此TOTAL_MALLOC_LEN=0x1420000。调整以后gd->start_addr_sp如下:

TOTAL_MALLOC_LEN=0x1420000

gd->start_addr_sp=0x3eb1a000 //0x3ff3a000-0x1420000=0x3eb1a000 (使能 Debug 时为 0x3eb04000)

第 47 行, reserve_board 函数, 留出板子 bd 所占的内存区, bd 是结构体 bd_t, bd_t 大小为 80 字节, 结果如下:

gd->start_addr_sp=0x3eb19fb0(使能 Debug 为 0x3eb03fb0)

gd->bd=0x3eb19fb0(使能 Debug 为 0x3eb03fb0)

第 48 行, setup_machine,设置机器 ID, linux 启动的时候会和这个机器 ID 匹配,如果匹配的话 linux 就会启动正常。但是 ZYNQ 已经不采用这种方式了,这是以前老版本的 uboot 和 linux 使用的,新版本使用 设备树了,因此此函数不起作用。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

第 49 行, reserve_global_data 函数, 保留出 gd_t 的内存区域, gd_t 结构体大小为 200B, 结果如下: gd->start_addr_sp=0x3eb19ee8 //0x3eb19fb0-0xc8=0x3eb19ee8(使能 Debug 时为 0x3eb03ee8) gd->new_gd=3eb19ee8(使能 Debug 为 0x3eb03ee8)

第50行, reserve_fdt, 留出设备树相关的内存区域,领航者的 uboot 没有用到, 因此此函数无效。

第51行, reserve_bootstage 保留 bootstage 空间,领航者的 uboot 没有用到。

第52行, reserve_arch 是个空函数。

第 53 行, reserve_stacks, 留出栈空间, 先对 gd->start_addr_sp 减去 16, 然后做 16 字节对齐。如果使 能 IRQ 的话还要留出 IRQ 相应的内存,具体工作是由 arch/arm/lib/stack.c 文件中的函数 arch_reserve_stacks 完成。结果如下(使能 Debug):

gd->start_addr_sp=0x3eb19ec0

第 54 行, setup_dram_config 函数设置 dram 信息,也就是设置 gd->bd->bi_dram[0].start 和 gd->bd->bi_dram[0].size,后面会传递给 linux 内核,告诉 linux DRAM 的起始地址和大小。DRAM 的起始地 址为 0x00000000,大小为 0x40000000 (1GB)。

第55行, show_dram_config 函数,用于显示 DRAM 的配置,如下图所示:



图 13.3.7 信息输出

第56行, display_new_sp函数,显示新的 sp 位置。

上图中的 gd->start_addr_sp 值和我们前面分析的最后一次修改的值一致。

第58行, reloc_fdt函数用于重定位fdt,没有用到。

第 59 行, reloc_bootstage 函数用于重定位 bootstage,没有用到。

第 60 行, setup_reloc, 设置 gd 其他一些成员变量,供后面重定位的时候使用,并且将以前的 gd 拷贝 到 gd->new_gd 处。

至此, board_init_f 函数就执行完成了, 最终的内存分配如下图所示:



正点原子

图 13.3.8 最终的内存分配图

13.3.5 relocate_code 函数详解

relocate_code 函数是用于代码拷贝的,此函数定义在文件 arch/arm/lib/relocate.S 中,代码如下:示例代码 relocate.S 代码段 68 /*

```
* void relocate code(addr moni)
69
70
    *
    * This function relocates the monitor code.
71
    *
72
73
    * NOTE:
    * To prevent the code below from containing references with an R_ARM_ABS32
74
    * relocation record type, we never refer to linker-defined symbols directly.
75
    * Instead, we declare literals which contain their relative location with
76
    * respect to relocate_code, and at run time, add relocate_code back to them.
77
78
     */
```

```
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
    79
    80 ENTRY(relocate code)
    81
          ldr r1, =__image_copy_start /* r1 <- SRC &__image_copy_start */</pre>
    82
          subs r4, r0, r1 /* r4 <- relocation offset */
          beq relocate_done /* skip relocation */
    83
          ldr r2, =__image_copy_end /* r2 <- SRC &__image_copy_end */
    84
    85
    86 copy_loop:
    87
         Idmia r1!, {r10-r11} /* copy from source address [r1] */
         stmia r0!, {r10-r11} /* copy to target address [r0] */
    88
                      /* until source end address [r2] */
    89
          cmp r1, r2
    90
          blo copy_loop
    91
    92
         /*
    93
          * fix .rel.dyn relocations
    94
          */
    95
         ldr r2, =__rel_dyn_start /* r2 <- SRC &__rel_dyn_start */
          ldr r3, =__rel_dyn_end /* r3 <- SRC &__rel_dyn_end */
    96
    97 fixloop:
    98
         ldmia r2!, {r0-r1} /* (r0,r1) <- (SRC location,fixup) */
    99
          and r1, r1, #0xff
    100 cmp r1, #R_ARM_RELATIVE
    101 bne fixnext
    102
    103 /* relative fix: increase location by offset */
    104 add r0, r0, r4
    105 ldr r1, [r0]
    106 add r1, r1, r4
    107 str r1, [r0]
    108 fixnext:
    109
          cmp r2, r3
    110
          blo fixloop
    111
    112 relocate_done:
    113
    114 #ifdef __XSCALE__
    115 /*
         * On xscale, icache must be invalidated and write buffers drained,
    116
    117
           * even with cache disabled - 4.2.7 of xscale core developer's manual
    118
           */
    119
          mcr p15, 0, r0, c7, c7, 0 /* invalidate icache */
```

正点原子



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

120 mcr p15, 0, r0, c7, c10, 4 /* drain write buffer */

121 #endif

122

123 /* ARMv4- don't know bx lr but the assembler fails to see that */

124

125 #ifdef __ARM_ARCH_4__

126 mov pc, lr

- 127 #else
- 128 bx lr
- 129 #endif

130

131 ENDPROC(relocate_code)

第 81 行, r1=__image_copy_start, 也就是 r1 寄存器保存源地址, 由表 13.2.1 可知, __image_copy_start=0x4000000。

第 82 行, r0=0x3ff3a000, 这个地址就是 uboot 拷贝的目标首地址。r4=r0-r1=0x3ff3a000-0x400000=0x3fb3a000,因此r4保存偏移量。

第83行,如果在第82中,r0-r1等于0,说明r0和r1相等,也就是源地址和目的地址是一样的,那肯定就不需要拷贝了,执行relocate_done函数

第 84 行, r2=__image_copy_end, r2 中保存拷贝之前的代码结束地址, 由表 13.2.1 可知, __image_copy_end =0x409edcc。

第86行,标号 copy_loop 完成代码拷贝工作。从r1,也就是__image_copy_start 开始,读取 uboot 代码 保存到 r10 和 r11 中,一次只拷贝 2 个 32 位的数据。拷贝完成以后 r1 的值会更新,保存下一个要拷贝的数 据地址。

第88行,将r10和r11的数据写到r0开始的地方,也就是目的地址。写完以后r0的值会更新,更新为下一个要写入的数据地址。

第 89 行,比较 r1 是否和 r2 相等,也就是检查是否拷贝完成,如果不相等的话说明没有拷贝完成,没 有拷贝完成的话就跳转到 copy_loop 接着拷贝,直至拷贝完成。

接下来的第 95 行~129 行是.rel.dyn 段。.rel.dyn 是.relocation.dynamic 的简写,也就是动态重定位的意思。.rel.dyn 段是存放.text 段中需要重定位地址的集合。动态重定位就是 uboot 在运行时将自身拷贝到 DRAM 的另一个地方去继续运行,需要解决运行地址和链接地址不同导致的寻址问题。在具体的实现时涉 及到链接工具和相对寻址的内容。如在文件 arch/arm/config.mk 中有如下代码:

102 # needed for relocation

103 LDFLAGS_u-boot += -pie

从注释可以看到第 103 行的 "-pie" 选项用于重定位。编译链接 uboot 的时候会使用到 "-pie" 选项,如 下图所示:

arm-xilinx-linux-gnueabi-ld.bfd -gc-sections -Bstatic --no-dynamic-link er -Ttext 0x4000000 -o u-boot -T u-boot.lds arch/arm/cpu/armv7/start.o --start-grou p arch/arm/cpu/built-in.o arch/arm/cpu/armv7/built-in.o arch/arm/lib/built-in.o arch/arm/mach-zynq/built-in.o board/xilinx/zynq/built-in.o cmd/built-in.o commo n/built-in.o disk/built-in.o drivers/built-in.o drivers/dma/built-in.o drivers/

图 13.3.9 链接命令



原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php

使用"-pie"选项以后会生成一个.rel.dyn 段, .rel.dyn 段中存放着需要重定位的地址。uboot 在运行时 relocate_code 函数遍历.rel.dyn 段,根据.rel.dyn 段中存储的地址进行重定位,从而完成对所有需要重定位的 地址的修改。下面我们继续来看 relocate_code 函数下面地址重定位的实现。

第 95 行, r2= rel dyn start, 也就是.rel.dyn 段的起始地址, 执行结果见图 13.3.10。

第 96 行, r3= rel dyn end,也就是.rel.dyn 段的结束地址,执行结果见图 13.3.10。

第98行,从.rel.dyn 段起始地址开始,每次读取两个4字节的数据存放到r0和r1寄存器中。从下图的 执行结果可以看出 r0 存放的是代码拷贝前的地址; r1 存放的是需要重定位的标志。

Name	Hex	Decimal	
1919 rO	00400020	4194336	
1989 r 1	00000017	23	
1919 r2	004747a0	4671392	
¹⁰¹⁰ r3	00480cb0	4721840	
¹⁰¹⁰ r4	3fb3a000	1068736512	

图 13.3.10 第 98 行执行结果

第99行,r1中给的值与0xff进行与运算,其实就是取r1的低8位,从图13.3.10可以看到r1值不变。

第 100 行,判断 r1 中的值是否等于 R ARM RELATIVE。R ARM RELATIVE 定义在 include/elf.h 文 件中, 值为 0X17(23), 笔者推测是判断地址是否需要进行重定位的标志符号。

第 101 行,如果 rl 不等于 23 的话就说明不需要重定位,执行函数 fixnext,否则的话继续执行下面的 代码进行重定位。

第 104 行,r0 保存代码拷贝前的地址,r4 保存着代码拷贝的偏移量,r0+r4 就得到了代码拷贝后的地 址。执行结果见图 13.3.11 的 r0。

第 105, 读取 r0 所指向的代码拷贝后的地址处的值到 r1 中,从下图可以看到该值是地址,而且还是代 码拷贝前的地址。

Name	Hex	Decimal
¹⁰¹⁰ FO	3ff3a020	1072930848
1919 r1	00400060	4194400
¹⁰¹⁰ r2	004747a0	4671392
1010 гЗ	00480cb0	4721840
¹⁰¹⁰ r4	3fb3a000	1068736512

图 13.3.11 第 105 行执行结果

第106行,将r1中代码拷贝前的地址加上代码拷贝的偏移量r4即可得到代码拷贝后的地址,执行结果 见图 13.3.12。

第 107 行,將得到的代码拷贝后的地址值 r1 写入到代码拷贝后的 r0 所指地址处。



正点原子

Name		Hex	Decimal
	1919 FO	3ff3a020	1072930848
	¹⁰¹⁰ r1	3ff3a060	1072930912
	¹⁰¹⁰ г2	004747a0	4671392
	¹⁰¹⁰ гЗ	00480cb0	4721840
	¹⁰¹⁰ г4	3fb3a000	1068736512

图 13.3.12 第 107 行执行结果

第109行,比较r2和r3,查看.rel.dyn段重定位是否完成。

第110行,如果r2和r3不相等,说明.rel.dyn重定位还未完成,因此跳到fixloop继续重定位.rel.dyn段。 从上面的分析我们大概的知道了代码动态重定位是怎么一回事了。首先 uboot 在运行时将自身拷贝到 DRAM 的另一个地方去继续运行,有一个地址偏移,如从 0 拷贝到 100,地址偏移为 100。代码拷贝完成 后,所有指向代码拷贝前的地址都是无效的,所以需要进行地址重定位,由于是在 uboot 运行中进行的, 所以是动态重定位。那么如何进行重定位呢?譬如说代码拷贝前地址 32 处存放着地址值 54,代码拷贝后 的重定位就是将需要重定位的地址 32 加上地址偏移 100 等于 132, 然后读取地址 132 处的地址值 54, 使其 也加上地址偏移 100 等于 154, 最后更新地址 132 处的地址值 54 为 154, 即实现了地址动态重定位。

13.3.6 relocate_vectors 函数详解

```
函数 relocate_vectors 用于重定位向量表,此函数定义在文件 arch/arm/lib/relocate.S 中,函数源码如下:
                                   示例代码 relocate.S 代码段
28 ENTRY(relocate_vectors)
29
30 #ifdef CONFIG_CPU_V7M
31
     /*
32
     * On ARMv7-M we only have to write the new vector address
     * to VTOR register.
33
34
     */
     ldr r0, [r9, #GD_RELOCADDR] /* r0 = gd->relocaddr */
35
36
     ldr r1, =V7M_SCB_BASE
37
     str r0, [r1, V7M_SCB_VTOR]
38 #else
39 #ifdef CONFIG_HAS_VBAR
40
   /*
     * If the ARM processor has the security extensions,
41
42
     * use VBAR to relocate the exception vectors.
43
     */
     ldr r0, [r9, #GD_RELOCADDR] /* r0 = gd->relocaddr */
44
     mcr p15, 0, r0, c12, c0, 0 /* Set VBAR */
45
46 #else
     /*
47
     * Copy the relocated exception vectors to the
48
                                              426
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

49 * correct address
50 * CP15 c1 V bit gives us the location of the vectors:
51 * 0x0000000 or 0xFFFF0000.
52 */
53 ldr r0, [r9, #GD_RELOCADDR] /* r0 = gd->relocaddr */
54 mrc p15, 0, r2, c1, c0, 0 /* V bit (bit[13]) in CP15 c1 */

```
55 ands r2, r2, #(1 << 13)
```

```
56 ldreg r1, =0x00000000 /* If V=0 */
```

```
57 ldrne r1, =0xFFFF0000 /* If V=1 */
```

```
58 ldmia r0!, {r2-r8,r10}
```

- 59 stmia r1!, {r2-r8,r10}
- 60 Idmia r0!, {r2-r8,r10}
- 61 stmia r1!, {r2-r8,r10}
- 62 #endif
- 63 #endif
- 64 bx **lr**
- 65

66 ENDPROC(relocate_vectors)

第 30 行,如果定义了 CONFIG_CPU_V7M 的话就执行第 35~37 行的代码。从注释可以看出这是用于 Cortex-M 系列的 ARM,因此对于 ZYNQ 来说是无效的。

第 39 行,如果定义了 CONFIG_HAS_VBAR 的话就执行下面的语句,这个是向量表偏移,Cortex-A9 是支持向量表偏移的。而且,在.config 里面定义了 CONFIG_HAS_VBAR,因此会执行这个分支。

第 44 行, r0=gd->relocaddr,也就是重定位后 uboot 的首地址 0x3ff3a000,向量表从这个地址开始存放。 第 45 行,将 r0 的值写入到 CP15 的 VBAR 寄存器中,也就是将新的向量表首地址写入到寄存器 VBAR 中,设置向量表偏移。

13.3.7 board_init_r 函数详解

13.3.4 小节讲解了 board_init_f 函数,在此函数里面会调用一系列的函数来初始化一些外 设和 gd 的成员变量。但是 board_init_f 并没有初始化所有的外设,还需要做一些后续工作,这 些后续工作就是由函数 board_init_r 来完成的, board_init_r 函数定义在文件 common/board_r.c 中,代码如下:

示例代	式码 board_r.c 代码段				
844 void board_init_r(gd_t *new_gd, ulong dest_addr)					
845 {					
846 /*					
847 * Set up the new global data pointer. So far	only x86 does this				
848 * here.					
849 * TODO(sjg@chromium.org): Consider do	ing this for all archs, or				
850 * dropping the new_gd parameter.					
851 */					
852 #if CONFIG_IS_ENABLED(X86_64)					

427



```
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
    853 arch_setup_gd(new_gd);
   854 #endif
    855
   856 #ifdef CONFIG_NEEDS_MANUAL_RELOC
   857 int i;
   858 #endif
    859
   860 #if !defined(CONFIG_X86) && !defined(CONFIG_ARM) && !defined(CONFIG_ARM64)
   861 \text{ gd} = \text{new}_{\text{gd}};
    862 #endif
   863 gd->flags &= ~GD_FLG_LOG_READY;
    864
    865 #ifdef CONFIG_NEEDS_MANUAL_RELOC
   866 for (i = 0; i < ARRAY_SIZE(init_sequence_r); i++)
         init_sequence_r[i] += gd->reloc_off;
    867
    868 #endif
    869
   870 if (initcall_run_list(init_sequence_r))
   871 hang();
    872
   873 /* NOTREACHED - run_main_loop() does not return */
   874 hang();
   875 }
```

第870行调用 initcall_run_list 函数来执行初始化序列 init_sequence_r, init_sequence_r 是一个函数集合, init_sequence_r 同样定义在文件 common/board_r.c 中,由于 init_sequence_f 的内容比较长,里面有大量的条件编译代码,这里为了缩小篇幅,将条件编译部分删除掉了,去掉条件编译以后的 init_sequence_r 定义如下:

示例代码 board_r.c 代码段

```
1 static init_fnc_t init_sequence_r[] = {
```

- 2 initr_trace,
- 3 initr_reloc,
- 4 initr_caches,
- 5 initr_reloc_global_data,
- 6 initr_barrier,
- 7 initr_malloc,
- 8 log_init,
- 9 initr_bootstage, /* Needs malloc() but has its own timer */
- 10 initr_console_record,
- 11 bootstage_relocate,
- 12 initr_dm,
- 13 board_init,



X7X/11/1	<u> </u>	-HA/ CALLENIUM /			
原子哥	F在线教学:w	ww.yuanzige.com	论坛:www.ope	nedv.com/forum.php	
14	set_cpu_clk_inf	o, /* Setup clock informa	tion */		
15	efi_memory_ini	t,			
16	stdio_init_tables	,			
17	initr_serial,				
18	initr_announce,				
19	INIT_FUNC_W	ATCHDOG_RESET			
20	INIT_FUNC_W	ATCHDOG_RESET			
21	power_init_boar	·d,			
22	INIT_FUNC_W	ATCHDOG_RESET			
23	initr_mmc,				
24	initr_env,				
25	INIT_FUNC_W	ATCHDOG_RESET			
26	initr_secondary_	_cpu,			
27	INIT_FUNC_W	ATCHDOG_RESET			
28	stdio_add_devic	es,			
29	initr_jumptable,				
30	console_init_r,	/* fully init console as	a device */		
31	console_announ	ce_r,			
32	show_board_inf	0,			
33	INIT_FUNC_W	ATCHDOG_RESET			
34	interrupt_init,				
35	initr_enable_inte	errupts,			
36	initr_ethaddr,				
37	board_late_init,				
38	INIT_FUNC_W	ATCHDOG_RESET			
39	initr_net,				
40	run_main_loop,				
41	};				
第	2行, initr_trace	函数,如果定义了宏(CONFIG_TRACE 的记	f就会调用函数 trace_init,	初始化和调试
跟踪有	关的内容。				
第	3行, initr_reloc	函数用于设置 gd->flag	s,标记重定位完成。		
第	4行, initr_cach	es 函数用于初始化 cach	e,使能 cache。		
第	5行, initr_reloc	_global_data 函数,初如	台化重定位后 gd 的一	些成员变量。	
第	6行, initr_barri	er函数,ZYNQ未用到	o		
第	7行, initr_malle	oc 函数,初始化 malloc	0		

第 8 行, log_init 函数, 将所有的 struct log_driver 结构体都加入到 gd->log_head 的循环链表中, 并初始 化 gd->default_log_level。

第9行, initr_bootstage 函数, 初始化 bootstage 标志 ID。

第10行, initr_console_record 函数, 初始化控制台相关的内容, ZYNQ 未用到, 空函数。

第 11 行, bootstage_relocate 函数,复制所有的字符串。因为字符串可能指向 program.text 的旧位置,最终可能被破坏。



- 原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
 - 第12行, initr_dm 函数, 保存先前的重定位驱动程序模型并开始新的驱动模型。
 - 第13行, board_init 函数, 板级初始化。
 - 第14行, set_cpu_clk_info函数,建立时钟信息。
 - 第15行, efi_memory_init 函数, 初始化话 efi memory。
 - 第16行, stdio_init_tables 函数, stdio 相关初始化。
 - 第17行, initr_serial 函数, 初始化串口。
 - 第18行, initr_announce 函数, 与调试有关, 通知已经在 RAM 中运行。
 - 第21行, power_init_board 函数, 初始化电源芯片, 正点原子的 ZYNQ 开发板没有用到。
 - 第23行, initr_mmc函数, 初始化 EMMC。
 - 第24行, initr_env函数, 初始化环境变量。
 - 第26行, initr_secondary_cpu函数, 初始化其他 CPU 核。
 - 第28行, stdio_add_devices 函数, 各种输入输出设备的初始化。
 - 第29行, initr_jumptable 函数, 初始化跳转表。

第30行, console_init_r函数, 控制台初始化, 初始化完成以后此函数会调用 stdio_print_current_devices 函数来打印出当前的控制台设备, 如下图所示:

In:	serial@e0000000	
Out:	serial@e0000000	
Err:	serial@e0000000	

图 13.3.13 控制台信息

第 31 行, console_announce_r

第32行, show_board_info函数,显示开发板信息,空函数。

第34行, interrupt_init函数, 初始化中断。

第 35 行, initr_enable_interrupts 函数, 使能中断。

第 36 行, initr_ethaddr 函数, 初始化网络地址, 也就是获取 MAC 地址。读取环境变量 "ethaddr"的 值。

第 37 行, board_late_init 函数, 板子后续初始化, 此函数定义在文件 board/xilinx/zynq/board.c 中。读取 启动模式并设置环境变量。

第 39 行, initr_net 函数, 初始化网络设备, 函数调用顺序为: initr_net->eth_initialize, 串口输出如下图 所示信息:

Net: ZYNQ GEM: e000b000, mdio bus e000b000, phyaddr 7, interface rgmii-id Warning: ethernet@e000b000 MAC addresses don't match: Address in DT is 00:0a:35:00:8b:87 Address in environment is 00:0a:35:00:01:22 eth0: ethernet@e000b000 ZYNQ GEM: e000c000, mdio bus e000c000, phyaddr 4, interface gmii , eth1: ethernet@e000c000

图 13.3.14 网络信息输出

第40行,run_main_loop主循环,处理命令。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

13.3.8 run_main_loop 函数详解

uboot 启动以后会进入 4 秒倒计时,如果在 4 秒倒计时结束之前按下回车键,那么就会进入 uboot 的命令模式,如果倒计时结束以后都没有按下回车键,那么就会自动启动 Linux 内核,这个功能就是由run_main_loop 函数定义在文件 common/board_r.c 中,函数内容如下:

```
示例代码 board_r.c 文件代码段
636 static int run_main_loop(void)
637 {
638 #ifdef CONFIG_SANDBOX
639 sandbox_main_loop_init();
640 #endif
641 /* main_loop() can return to retry autoboot, if so just run it again */
642 for (;;)
643 main_loop();
644 return 0;
645 }
第 642 行是个死循环 "for(;;)",死循环里面就一个 main_loop 函数, main_loop 函数定义
```

示例代码 main loop 函数

在文件 common/main.c 里面,代码如下:

```
40 /* We come here after U-Boot is initialised and ready to process commands */
41 void main_loop(void)
42 {
43 const char *s;
44
45 bootstage_mark_name(BOOTSTAGE_ID_MAIN_LOOP, "main_loop");
46
47 if (IS_ENABLED(CONFIG_VERSION_VARIABLE))
     env_set("ver", version_string); /* set version variable */
48
49
50 cli_init();
51
52 if (IS_ENABLED(CONFIG_USE_PREBOOT))
     run_preboot_environment_command();
53
54
55 if (IS_ENABLED(CONFIG_UPDATE_TFTP))
56
     update_tftp(0UL, NULL, NULL);
57
58 s = bootdelay_process();
59 if (cli_process_fdt(&s))
     cli_secure_boot_cmd(s);
60
61
```

```
62 autoboot_command(s);
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

63

64 cli_loop();

65 panic("No CLI available");

<u>66</u> }

第45行,调用 bootstage_mark_name 函数,打印出启动进度。

第47行,如果定义了宏 CONFIG_VERSION_VARIABLE 的话就会执行函数 setenv,设置 变量 ver 的值为 version_string,也就是设置版本号环境变量。领航者的 zynq 没有定义该宏。

第50行, cli_init 函数, 与命令初始化有关, 初始化 hush shell 相关的变量。

第53行,run_preboot_environment_command函数,获取环境变量perboot的内容,perboot 是一些预启动命令,一般不使用这个环境变量。

第55行,如果定义了宏 CONFIG_UPDATE_TFTP,就更新 tftp, zynq 没有定义改宏。

第58行, bootdelay_process函数,此函数会读取环境变量 bootdelay 和 bootcmd 的内容,

然后将 bootdelay 的值赋值给全局变量 stored_bootdelay,返回值为环境变量 bootcmd 的值。

第59行,函数 cli_process_fdt 读取设备树中的 bootcmd,覆盖之前读取的 bootcmd 值。

第 62 行, autoboot_command 函数, 此函数就是检查倒计时是否结束? 倒计时结束之前有 没有被打断? 此函数定义在文件 common/autoboot.c 中。

第 64 行, cli_loop 函数是 uboot 的命令行处理函数,我们在 uboot 中输入各种命令,进行 各种操作就是有 cli_loop 来处理的,此函数定义在文件 common/cli.c 中,具体内容我们就不看 了

13.4 bootz 启动 Linux 内核过程

13.4.1 images 全局变量

不管是 bootz 还是 bootm 命令,在启动 Linux 内核的时候都会用到一个重要的全局变量: images, images 在文件 cmd/bootm.c 中有如下定义:

示例代码 images 全局变量

1 bootm_headers_t images; /* pointers to os/initrd/fdt images */

images 是 bootm_headers_t 类型的全局变量, bootm_headers_t 是个 boot 头结构体, 在文件 include/image.h 中的定义如下(删除了一些条件编译代码):

示例代码 bootm_headers_t 结构体

332 typedef struct bootm_headers {

333 /*

* Legacy os image header, if it is a multi component image

* then boot_get_ramdisk() and get_fdt() will attempt to get

* data from second and third component accordingly.

337 */

338 image_header_t *legacy_hdr_os; /* image header pointer */

339 image_header_t legacy_hdr_os_copy; /* header copy */

340 ulong legacy_hdr_valid;

•••••

362 #ifndef USE_HOSTCC


```
原子哥在线教学: www.yuanzige.com
                                         论坛:www.openedv.com/forum.php
        image_info_t os; /* os image info */
   363
   364
                ep; /* entry point of OS */
        ulong
   365
                rd_start, rd_end;/* ramdisk start/end */
   366
        ulong
   367
                *ft addr; /* flat dev tree address */
   368
        char
                ft_len; /* length of flat device tree */
   369
        ulong
   370
   371
                initrd_start;
        ulong
   372
        ulong
                initrd end;
   373
        ulong
                cmdline_start;
                cmdline_end;
   374
        ulong
   375
        bd t
                *kbd;
   376 #endif
   377
   378
        int verify; /* env_get("verify")[0] != 'n' */
   379
   380 #define BOOTM_STATE_START (0x0000001)
   381 #define BOOTM_STATE_FINDOS (0x0000002)
   382 #define BOOTM_STATE_FINDOTHER (0x00000004)
   383 #define BOOTM_STATE_LOADOS (0x0000008)
   384 #define BOOTM_STATE_RAMDISK (0x0000010)
   385 #define BOOTM_STATE_FDT (0x0000020)
   386 #define BOOTM_STATE_OS_CMDLINE (0x00000040)
   387 #define BOOTM_STATE_OS_BD_T (0x0000080)
   388 #define BOOTM_STATE_OS_PREP (0x00000100)
   389 #define BOOTM_STATE_OS_FAKE_GO (0x0000200) /* 'Almost' run the OS */
   390 #define BOOTM_STATE_OS_GO (0x00000400)
   391
        int state:
   392
   393 #ifdef CONFIG_LMB
   394 struct lmb lmb;
                        /* for memory mgmt */
   395 #endif
   396 } bootm_headers_t;
    第363行的 os 成员变量是 image_info_t 类型的,为系统镜像信息。
    第 380~390 行这 11 个宏定义表示 BOOT 的不同阶段。
    接下来看一下结构体 image_info_t, 也就是系统镜像信息结构体, 此结构体在文件
include/image.h 中的定义如下:
```

示例代码 image_info_t 结构体

336 typedef struct image_info {

337 ulong start, end; /* start/end of blob */



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php image_start, image_len; /* start of image within blob, len of image */ 338 ulong 339 ulong load: /* load addr for the image */ comp, type, os; /* compression, type of image, os type */ 340 uint8_t /* CPU architecture */ 341 uint8 t arch; 342 } image_info_t; 全局变量 images 会在 bootz 命令的执行中频繁使用到,相当于 Linux 内核启动的"灵魂"。 13.4.2 do_bootz 函数 bootz 命令的执行函数为 do_bootz, 在文件 cmd/bootz.c 中有如下定义: 示例代码 do_bootz 函数 61 int do_bootz(cmd_tbl_t *cmdtp, int flag, int argc, char * const argv[]) **62** { 63 int ret; 64 65 /* Consume 'bootz' */ 66 argc--; argv++; 67 68 if (bootz_start(cmdtp, flag, argc, argv, &images)) **69** return 1; 70 71 /* 72 * We are doing the BOOTM_STATE_LOADOS state ourselves, so must 73 * disable interrupts ourselves 74 */ 75 bootm_disable_interrupts(); 76 77 images.os.os = IH_OS_LINUX; 78 ret = do_bootm_states(cmdtp, flag, argc, argv, 79 #ifdef CONFIG_SYS_BOOT_RAMDISK_HIGH 80 BOOTM_STATE_RAMDISK | 81 #endif 82 BOOTM_STATE_OS_PREP | BOOTM_STATE_OS_FAKE_GO | 83 BOOTM_STATE_OS_GO, 84 &images, 1); 85 86 return ret; 87 } 第 68 行,调用 bootz start 函数, bootz start 函数执行过程参考下一小节。 第75行,调用函数 bootm_disable_interrupts 关闭中断。

第 77 行,设置 images.os.os 为 IH_OS_LINUX,也就是设置系统镜像为 Linux,表示我们 要启动的是 Linux 系统。后面会用到 images.os.os 来选择具体的启动函数。



原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php

第 78 行,调用函数 do_bootm_states 来执行不同的 BOOT 阶段,这里要执行的 BOOT 阶 段有:BOOTM_STATE_OS_PREP、BOOTM_STATE_OS_FAKE_GO 和 BOOTM_STATE_OS _GO。

13.4.3 bootz_start 函数

```
bootz_start 函数用于支持 zImage 镜像的启动, 定义在文件 cmd/bootz.c 中, 函数内容如下:
                                     示例代码 bootz_start 函数
25 static int bootz_start(cmd_tbl_t *cmdtp, int flag, int argc,
26
        char * const argv[], bootm_headers_t *images)
27 {
28 int ret;
29 ulong zi_start, zi_end;
30
31 ret = do_bootm_states(cmdtp, flag, argc, argv, BOOTM_STATE_START,
32
           images, 1);
33
34 /* Setup Linux kernel zImage entry point */
35 if (!argc) {
36
     images->ep = load_addr;
     debug("* kernel: default image load address = 0x\%081x\n",
37
38
          load_addr);
39 } else {
     images->ep = simple_strtoul(argv[0], NULL, 16);
40
     debug("* kernel: cmdline image address = 0x\%081x\n",
41
42
        images->ep);
43 }
44
45 ret = bootz_setup(images->ep, &zi_start, &zi_end);
46 if (ret != 0)
47
     return 1;
48
49 lmb_reserve(&images->lmb, images->ep, zi_end - zi_start);
50
51 /*
52 * Handle the BOOTM_STATE_FINDOTHER state ourselves as we do not
53 * have a header that provide this information.
54 */
55 if (bootm_find_images(flag, argc, argv))
56
     return 1;
57
58 return 0;
```

```
正点原子
领航者 ZYNQ 之嵌入式 Linux 开发指南
原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php
   59 }
   第 31 行,调用函数 do_bootm_states,执行 BOOTM_STATE_START 阶段。
   第36行,设置 images 的 ep 成员变量,也就是系统镜像的入口点,使用 bootz 命令启动系
统的时候就会设置系统在 DRAM 中的存储位置,这个存储位置就是系统镜像的入口点,因此
images->ep=0X8000.
   第45行,调用 bootz setup 函数,此函数会判断当前的系统镜像文件是否为 Linux 的镜像
文件,并且会打印出镜像相关信息,bootz_setup 函数稍后会讲解。
   第55行,调用函数 bootm find images 查找 ramdisk 和设备树(dtb)文件,但是我们没有用
到 ramdisk,因此此函数在这里仅仅用于查找设备树(dtb)文件,此函数稍后也会讲解。
   先来看一下 bootz_setup 函数,此函数定义在文件 arch/arm/lib/zimage.c 中,函数内容如下:
                              示例代码 bootz_setup 函数
   11 #define LINUX_ARM_ZIMAGE_MAGIC 0x016f2818
   12 #define BAREBOX_IMAGE_MAGIC 0x00786f62
   13
   14 struct arm_z_header {
   15 uint32_t code[9];
   16 uint32_t zi_magic;
   17 uint32_t zi_start;
   18 uint32_t zi_end;
   19 } __attribute__ ((__packed__));
   20
   21 int bootz_setup(ulong image, ulong *start, ulong *end)
   22 {
   23 struct arm_z_header *zi = (struct arm_z_header *)image;
   24
   25 if (zi->zi_magic != LINUX_ARM_ZIMAGE_MAGIC &&
       zi->zi_magic != BAREBOX_IMAGE_MAGIC) {
   26
   27 #ifndef CONFIG_SPL_FRAMEWORK
       puts("zimage: Bad magic!\n");
   28
   29 #endif
   30
       return 1;
   31 }
   32
   33 *start = zi->zi_start;
   34 *end = zi->zi_end;
   35 #ifndef CONFIG_SPL_FRAMEWORK
   36 printf("Kernel image @ %#081x [ %#081x - %#081x ]\n",
   37
        image, *start, *end);
   38 #endif
```

39

```
40 return 0;
```

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

41 }

第11行,宏LINUX_ARM_ZIMAGE_MAGIC是 ARM Linux 系统镜像的重要标志。

正点原子

第 21 行,从传递进来的参数 image(也就是系统镜像首地址)中获取 zImage 头。zImage 头 结构体为 arm_z_header。

第 25~30 行,判断 image 是否为 ARM 的 Linux 系统镜像,如果不是的话就直接返回,并 且打印出 "zimage: Bad magic!",比如我们输入一个错误的启动命令:

bootz 8000 - 90000

因为我们并没有在 0X8000 处存放 Linux 镜像文件(zImage),因此上面的命令肯定会执行 出错的。

第 33、34 行初始化函数 bootz_setup 的参数 start 和 end。

第36行,打印启动信息,如果 Linux 系统镜像正常的话就会输出相应的打印信息。

接下来看一下函数 bootm_find_images。boot_find_images 函数将尝试加载可用的 ramdisk, 设备树以及带有特殊标记的"可加载"映像,此函数定义在文件 common/boot m.c 中,函数 内容如下:

示例代码 bootm_find_images 函数

239 int bootm_find_images(int flag, int argc, char * const argv[]) 240 { 241 int ret; 242 243 /* find ramdisk */ 244 ret = boot_get_ramdisk(argc, argv, &images, IH_INITRD_ARCH, 245 &images.rd_start, &images.rd_end); 246 **if** (ret) { puts("Ramdisk image is corrupt or invalid\n"); 247 248 return 1; 249 } 250 251 #if IMAGE_ENABLE_OF_LIBFDT 252 /* find flattened device tree */ 253 ret = boot_get_fdt(flag, argc, argv, IH_ARCH_DEFAULT, & images, &images.ft_addr, &images.ft_len); 254 255 if (ret) { 256 puts("Could not find a valid device tree\n"); 257 return 1; 258 } 259 if (CONFIG_IS_ENABLED(CMD_FDT)) set_working_fdt_addr(map_to_sysmem(images.ft_addr)); 260 261 #endif 262 263 #if IMAGE_ENABLE_FIT 264 #if defined(CONFIG FPGA)



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

265 /* find bitstreams */
266 ret = boot_get_fpga(argc, argv, &images, IH_ARCH_DEFAULT,
267 NULL, NULL);
268 if (ret) {
269 printf("FPGA image is corrupted or invalid\n");
270 return 1;
271 }
272 #endif
.....
283 return 0;
284 }
第 244~249 行是跟查找 ramdisk, 但是我们没有用到 ramdisk, 因此这部分代码不用管。

第 251~261 行是查找设备树(dtb)文件,找到以后就将设备树的起始地址和长度分别写到 images 的 ft_addr 和 ft_len 成员变量中。我们使用 bootz 启动 Linux 的时候已经指明了设备树在 DRAM 中的存储地址,长度根据具体的设备树文件而定。

第 264~272 行是查找 FPGA 的 bitstreams 文件。

bootz_start 函数就讲解到这里,bootz_start 主要用于初始化 images 的相关成员变量。

13.4.4 do_bootm_states 函数

do_bootz 最后调用的就是函数 do_bootm_states, 而且在 bootz_start 中也调用了 do_bootm_states 函数, 看来 do_bootm_states 函数还是个香饽饽。此函数定义在文件 common/bootm.c 中, 函数 do_bootm_states 根据不同的 BOOT 状态执行不同的代码段, 通过如下代码来判断 BOOT 状态:

states & BOOTM_STATE_XXX

在 do_bootz 函数中会用到 BOOTM_STATE_OS_PREP、BOOTM_STATE_OS_FAKE_GO 和 BOOTM_STATE_OS_GO 这 三 个 BOOT 状态, bootz_start 函 数 中 会 用 到 BOOTM_STATE_START 状态。由于此函数代码较长,为了精简代码,方便分析,因此我们 将函数 do_bootm_states 进行精简,只留下下面这 4 个 BOOT 状态对应的处理代码:

BOOTM_STATE_OS_PREP

BOOTM_STATE_OS_FAKE_GO

BOOTM_STATE_OS_GO

BOOTM_STATE_START

精简以后的 do_bootm_states 函数如下所示:

示例代码 do_bootm_states 函数

603 int do_bootm_states(cmd_tbl_t *cmdtp, int flag, int argc, char * const argv[],

604 int states, bootm_headers_t *images, int boot_progress)

```
605 {
```

606 boot_os_fn *boot_fn;

```
607 ulong iflag = 0;
```

```
608 int ret = 0, need_boot_fn;
```

```
609
```



```
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
    610
          images->state |= states;
    611
    612
         /*
    613
          * Work through the states and see how far we get. We stop on
    614
          * any error.
          */
    615
    616
          if (states & BOOTM_STATE_START)
    617
            ret = bootm_start(cmdtp, flag, argc, argv);
    . . . . . .
    661
          /* From now on, we need the OS boot function */
    662
          if (ret)
    663
           return ret;
    664
          boot_fn = bootm_os_get_boot_func(images->os.os);
    . . . . . .
    683
          if (!ret && (states & BOOTM_STATE_OS_PREP)) {
    684 #if defined(CONFIG_SILENT_CONSOLE) && !defined(CONFIG_SILENT_U_BOOT_ONLY)
            if (images->os.os == IH_OS_LINUX)
    685
              fixup_silent_linux();
    686
    687 #endif
    688
            ret = boot_fn(BOOTM_STATE_OS_PREP, argc, argv, images);
          }
    689
    690
    691 #ifdef CONFIG_TRACE
          /* Pretend to run the OS, then run a user command */
    692
          if (!ret && (states & BOOTM_STATE_OS_FAKE_GO)) {
    693
    694
            char *cmd_list = env_get("fakegocmd");
    695
    696
            ret = boot_selected_os(argc, argv, BOOTM_STATE_OS_FAKE_GO,
    697
                 images, boot_fn);
    698
            if (!ret && cmd_list)
    699
              ret = run_command_list(cmd_list, -1, flag);
    700 }
    701 #endif
          /* Check for unsupported subcommand. */
    703
    704
          if (ret) {
    705
            puts("subcommand not supported\n");
    706
            return ret;
    707
          }
    708
    709
          /* Now run the OS! We hope this doesn't return */
    710
          if (!ret && (states & BOOTM_STATE_OS_GO))
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

711 ret = boot_selected_os(argc, argv, BOOTM_STATE_OS_GO,

```
712 images, boot_fn);
.....
724 return ret;
```

```
725 }
```

第 616~617 行,处理 BOOTM_STATE_START 阶段,bootz_start 会执行这一段代码,这 里调用函数 bootm_start,用来清零 images 并重新设置 images 的 state 和 verify。

第 664 行非常重要。通过函数 bootm_os_get_boot_func 来查找系统启动函数。参数 images->os.os 就是系统类型,根据这个系统类型来选择对应的启动函数,在 do_bootz 中设置 images.os.os= IH_OS_LINUX。函数返回值就是找到的系统启动函数,这里找到的 Linux 系统 启动函数为 do_bootm_linux,关于此函数查找系统启动函数的过程请参考 13.4.5 小节。因此 boot_fn=do_bootm_linux,后面执行 boot_fn 函数的地方实际上是执行的 do_bootm_linux 函数。

第 683 行,处理 BOOTM_STATE_OS_PREP 状态,调用函数 do_bootm_linux。 do_bootm_linux 函数内部也是调用 boot_prep_linux 来完成具体的处理过程。boot_prep_linux 主要用于处理环境变量 bootargs, bootargs 保存着传递给 Linux kernel 的参数。

第 691~701 行是处理 BOOTM_STATE_OS_FAKE_GO 状态的,但是要我们没用使能 TRACE 功能,因此宏 CONFIG_TRACE 也就没有定义,所以这段程序不会编译。

第711行,调用函数 boot_selected_os 启动 Linux 内核,此函数第4个参数为 Linux 系统镜像头,第5个参数就是 Linux 系统启动函数 do_bootm_linux。boot_selected_os 函数定义在文件 common/bootm_os.c 中,函数内容如下:

示例代码 boot_selected_os 函数

551 int boot_selected_os(int argc, char * const argv[], int state,

552 bootm_headers_t *images, boot_os_fn *boot_fn)

553 {

```
554 arch_preboot_os();
```

555 boot_fn(state, argc, argv, images);

•••••

```
566 return BOOTM_ERR_RESET;
```

567 }

第555行调用 boot_fn 函数,也就是 do_bootm_linux 函数来启动 Linux 内核。

13.4.5 bootm_os_get_boot_func 函数

do_bootm_states 会调用 bootm_os_get_boot_func 来查找对应系统的启动函数,此函数定义 在文件 common/bootm_os.c 中,函数内容如下:

示例代码 bootm_os_get_boot_func 函数

569 boot_os_fn *bootm_os_get_boot_func(int os)
570 {
571 #ifdef CONFIG_NEEDS_MANUAL_RELOC
572 static bool relocated;
573
574 if (!relocated) {



```
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
```

```
575
      int i;
576
577 /* relocate boot function table */
      for (i = 0; i < ARRAY_SIZE(boot_os); i++)
578
        if (boot_os[i] != NULL)
579
580
           boot os[i] += gd->reloc off;
581
582
      relocated = true;
583 }
584 #endif
585 return boot_os[os];
586 }
```

第 571~584 行是条件编译,在 zynq 的 uboot 中没有用到,因此这段代码无效,只有第 585 行有效。第 585 行中的 boot_os 是个数组,这个数组里面存放着不同的系统对应的启动函数。boot_os 也定义在文件 common/bootm_os.c 中,如下所示:

```
示例代码 boot_os 数组
```

```
501 static boot_os_fn *boot_os[] = {
502 [IH_OS_U_BOOT] = do_bootm_standalone,
503 #ifdef CONFIG_BOOTM_LINUX
504 [IH_OS_LINUX] = do_bootm_linux,
505 #endif
.....
531 #ifdef CONFIG_BOOTM_OPENRTOS
532 [IH_OS_OPENRTOS] = do_bootm_openrtos,
533 #endif
534 #ifdef CONFIG_BOOTM_OPTEE
535 [IH_OS_TEE] = do_bootm_tee,
536 #endif
537 };
第 504 行就是 Linux 系统对应的启动函数: do_bootm_linux.
```

13.4.6 do_bootm_linux 函数

经过前面的分析,我们知道了 do_bootm_linux 就是最终启动 Linux 内核的函数,此函数 定义在文件 arch/arm/lib/bootm.c,函数内容如下:

示例代码 do_bootm_linux 函数

```
418 int do_bootm_linux(int flag, int argc, char * const argv[],
419 bootm_headers_t *images)
420 {
421 /* No need for those on ARM */
422 if (flag & BOOTM_STATE_OS_BD_T || flag & BOOTM_STATE_OS_CMDLINE)
423 return -1;
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

```
424
425 if (flag & BOOTM STATE OS PREP) {
      boot_prep_linux(images);
426
427
      return 0;
428 }
429
430 if (flag & (BOOTM_STATE_OS_GO | BOOTM_STATE_OS_FAKE_GO)) {
431
      boot_jump_linux(images, flag);
432
      return 0;
433 }
434
435 boot_prep_linux(images);
436 boot_jump_linux(images, flag);
437 return 0;
438 }
```

第 430 行,如果参数 flag 等于 BOOTM_STATE_OS_GO 或者 BOOTM_STATE_OS_FAKE _GO 的话就执行 boot_jump_linux 函数。boot_selected_os 函数在调用 do_bootm_linux 的时候会 将 flag 设置为 BOOTM_STATE_OS_GO。

第 436 行,执行函数 boot_jump_linux,此函数定义在文件 arch/arm/lib/bootm.c 中,函数 内容如下:

```
示例代码 boot_jump_linux 函数
```

```
327 /* Subcommand: GO */
328 static void boot_jump_linux(bootm_headers_t *images, int flag)
329 {
330 #ifdef CONFIG ARM64
. . . . . .
368 #else
369 unsigned long machid = gd->bd->bi_arch_number;
370 char *s;
371 void (*kernel_entry)(int zero, int arch, uint params);
372 unsigned long r2;
373 int fake = (flag & BOOTM_STATE_OS_FAKE_GO);
374
375 kernel_entry = (void (*)(int, int, uint))images->ep;
376 #ifdef CONFIG_CPU_V7M
377 ulong addr = (ulong)kernel_entry | 1;
378 kernel_entry = (void *)addr;
379 #endif
380 s = env_get("machid");
381 if (s) {
```

```
382 if (strict_strtoul(s, 16, &machid) < 0) {
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

<pre>383 debug("strict_strtoul failed!\n");</pre>
384 return;
385 }
386 printf("Using machid 0x%1x from environment\n", machid);
387 }
388
389 debug("## Transferring control to Linux (at address %08lx)" \
390 "\n", (ulong) kernel_entry);
391 bootstage_mark(BOOTSTAGE_ID_RUN_OS);
392 announce_and_cleanup(fake);
393
394 if (IMAGE_ENABLE_OF_LIBFDT && images->ft_len)
395 r2 = (unsigned long)images->ft_addr;
396 else
$r^2 = gd \rightarrow bd \rightarrow bi_boot_params;$
407 kernel_entry(0, machid, r2);
408 }
409 #endif
410.1

第 330~367 行是 64 位 ARM 芯片对应的代码, Cortex-A9 是 32 位芯片,因此用不到。

第 369 行,变量 machid 保存机器 ID,如果不使用设备树的话这个机器 ID 会被传递给 Linux 内核,Linux 内核会在自己的机器 ID 列表里面查找是否存在与 uboot 传递进来的 machid 匹配的项目,如果存在就表明Linux 内核支持这个机器,那么Linux 就会启动。如果使用设备 树的话这个 machid 就无效了,设备树有一个"兼容性"属性,Linux 内核会比较"兼容性" 属性的值(字符串)来查看是否支持这个机器。

第 371 行,函数 kernel_entry,看名字"内核_进入",说明此函数是进入 Linux 内核的, 也就是最终的大 boos。此函数有三个参数: zero, arch, params,第一个参数 zero 同样为 0; 第二个参数为机器 ID;第三个参数 ATAGS 或者设备树(DTB)首地址,ATAGS 是传统的方法, 用于传递一些命令行信息啥的,如果使用设备树的话就要传递设备树(DTB)。

第 375 行,获取 kernel_entry 函数,函数 kernel_entry 并不是 uboot 定义的,而是 Linux 内核定义的,Linux 内核镜像文件的第一行代码就是函数 kernel_entry,而 images->ep 保存着 Linux 内核镜像的起始地址,而起始地址保存的不正是 Linux 内核第一行代码么。

第 392 行,调用函数 announce_and_cleanup 来打印一些信息并做一些清理工作,此函数定 义在文件 arch/arm/lib/bootm.c 中,函数内容如下:

示例代码 announce_and_cleanup 函数

91 static void announce_and_cleanup(int fake)

92 {

93 bootstage_mark_name(BOOTSTAGE_ID_BOOTM_HANDOFF, "start_kernel");

94 #ifdef CONFIG_BOOTSTAGE_FDT

95 bootstage_fdt_add_report();



原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php 96 #endif 97 #ifdef CONFIG BOOTSTAGE REPORT 98 bootstage_report(); 99 #endif 100 101 #ifdef CONFIG USB DEVICE 102 udc_disconnect(); 103 #endif 104 105 board quiesce devices(); 106 107 printf("\nStarting kernel ...%s\n\n", fake ? **108** "(fake run for tracing)" **:** ""); 109 /* 110 * Call remove function of all devices with a removal flag set. 111 * This may be useful for last-stage operations, like cancelling 112 * of DMA operation or releasing device internal buffers. 113 */ 114 dm_remove_devices_flags(DM_REMOVE_ACTIVE_ALL); 115 116 cleanup_before_linux(); 117 } 第 107 行,在启动 Linux 之前输出 "Starting kernel ..."信息,如下图所示: Verifying Hash Integrity ... sha256+ OK Booting using the fdt blob at 0x2428454 Loading Kernel Image Loading Ramdisk to 1e3e5000, end 1eb07ce7 ... OK Loading Device Tree to le3db000, end le3e4de6 ... OK Starting kernel ...

Booting Linux on physical CPU 0x0

图 13.4.1 启动内核提示信息

第116行调用 cleanup_before_linux 函数做一些清理工作。

继续回到示例代码 boot_jump_linux 函数,第 394~397 行是设置寄存器 r2 的值?为什么要 设置 r2 的值呢?Linux 内核一开始是汇编代码,因此函数 kernel_entry 就是个汇编函数。向汇 编函数传递参数要使用 r0、r1 和 r2(参数数量不超过 3 个的时候),所以 r2 寄存器就是函数 kernel_entry 的第三个参数。

第 394 行,如果使用设备树的话,r2 应该是设备树的起始地址,而设备树地址保存在 images 的 ftd_addr 成员变量中。

第 396 行,如果不使用设备树的话,r2 应该是 uboot 传递给 Linux 的参数起始地址,也就 是环境变量 bootargs 的值,

②正点原子

原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php

407 行,调用 kernel_entry 函数进入 Linux 内核,此行将一去不复返,uboot 的使命也就完成了,它可以安息了!

总结一下 bootz 命令的执行过程,如下图所示:

bootz命令



图 13.4.2 bootz 命令执行过程

到这里 uboot 的启动流程我们就讲解完成了,加上 uboot 顶层 Makefile 的分析,洋洋洒洒 近 100 页,还是不少的。这也仅仅是 uboot 启动流程分析,当缕清了 uboot 的启动流程以后, 后面移植 uboot 就会轻松很多。其实在工作中我们基本不需要这么详细的去了解 uboot,半导 体厂商提供给我们的 uboot 一般是可以直接用的,只要能跑起来,可以使用就可以了。但是作 为学习,我们是必须了解一下 uboot 的启动流程,否则如果在工作中遇到问题我们连解决的方 法都没有,都不知道该从哪里看起。但是呢,如果第一次就想弄懂 uboot 的整个启动流程还是 有点困难的,所以如果没有看懂的话,不要紧!不要气馁,大多数人第一次看 uboot 启动流程 基本都有各种各样的问题。

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

正点原子

第十四章 U-Boot 移植

上一章节我们详细的分析了 uboot 的启动流程,对 uboot 有了一个初步的了解。前两章我 们都是使用的正点原子提供的 uboot,本章我们就来学习如何将 Xilinx 官方的 uboot 移植到正 点原子的 ZYNQ 开发板上,学习如何在 uboot 中添加我们自己的开发板。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

14.1 Xilinx 官方开发板 uboot 编译测试

14.1.1 查找 Xilinx 官方的开发板默认配置文件

uboot 的移植并不是说我们完完全全的从零开始将 uboot 移植到我们现在所使用的开发板 或者开发平台上。这个对于我们来说基本是不可能的,这个工作一般是半导体厂商做的,半 导体厂商负责将 uboot 移植到他们的芯片上,因此半导体厂商都会自己做一个开发板,这个开 发板就叫做原厂开发板,比如大家学习 STM32 的时候听说过的 discover 开发板就是 ST 自己 做的。半导体厂商会将 uboot 移植到他们自己的原厂开发板上,测试好以后就会将这个 uboot 发布出去,这就是大家常说的原厂 BSP 包。我们一般做产品的时候就会参考原厂的开发板做 硬件,然后在原厂提供的 BSP 包上做修改,将 uboot 或者 linux kernel 移植到我们的硬件上。 这个就是 uboot 移植的一般流程:

- ① 在 uboot 中找到参考的开发平台,一般是原厂的开发板。
- ② 参考原厂开发板移植 uboot 到我们所使用的开发板上。

正点原子的 ZYNQ 开发板参考的是 Xilinx 官方的 ZYNQ ZC702 开发板做的硬件,因此我 们在移植 uboot 的时候就可以以 Xilinx 官方的 ZYNQ ZC702 开发板为标本。

本章我们是将Xilinx 官方的 uboot 移植到正点原子的 ZYNQ 开发板上, Xilinx 官方的 uboot 放到了开发板光盘中,路径为:领航者 ZYNQ 开发板资料盘(A盘)\4_SourceCode\3_Embedded_Linux\资源文件\资源文件\资源文件\boot\u-boot-xlnx-xilinx-v2020.1.tar.gz。将 u-boot-xlnx-xilinx-v2020.1.tar.g 拷贝到 Ubuntu 下的 petalinux/uboot 中并解压。 解压 Xilinx 官方的 uboot 后得到的 configs 目录下有很多跟 zyng 有关的配置,如下图所示,

wmq@Linux: ~/petalinux/uboot/u-boot-xlnx-xilinx-v2020.1/configs
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
<pre>xilinx_zynqmp_mini_defconfig xilinx_zynqmp_mini_emmc0_defconfig xilinx_zynqmp_mini_emmc1_defconfig xilinx_zynqmp_mini_nand_defconfig xilinx_zynqmp_mini_qspi_defconfig xilinx_zynqmp_r5_defconfig xilinx_zynqmp_virt_defconfig xilinx_zynqmp_virt_defconfig xilinx_zynq_virt_defconfig xpedite517x_defconfig xpedite520x_defconfig xpedite550x_defconfig xpress_defconfig xpress_spl_defconfig xpress_spl_defconfig xtfpaa_defconfig</pre>
Yones_Toptech_BD1078_defconfig
zc5202_defconfig
zc5601_defconfig zmx25_defconfig
zynq_cse_nand_defconfig zynq_cse_nor_defconfig
zynq_cse_qspi_defconfig wmg@Linux:~/petalinux/uboot/u-boot-xlnx-xilinx-v2020.1/configs\$

图 14.1.1 Xilinx 官方 zynq 默认配置文件

从上图可以看出有很多的默认配置文件,其中以 zynq 开头的是 ZYNQ 相关开发板的配置 文件。由于我们的 ZYNQ 开发板参考的是 Xilinx 官方的 ZYNQ ZC702 开发板做的硬件,所以



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

我们在移植的时候就以赛灵思 zynq 开发板的配置文件(xilinx_zynq_virt_defconfig)为模板进行移植。

14.1.2 编译 Xilinx 官方开发板对应的 uboot

首先是配置 uboot, 然后编译 uboot, 命令如下(首先需要启动交叉编译器, 需要 source petalinux 安装目录下的 settings.sh 脚本文件或者运行". /opt/petalinux/2020.2/environment-setup-cortexa9t2hf-neon-xilinx-linux-gnueabi"命令):

make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi- xilinx_zynq_virt_defconfig make V=1 ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi--j8 编译完成以后结果如下图所示:



图 14.1.2 执行命令

注: 上图终端中的~petalinux/uboot/u-boot-xlnx-xilinx-v2020.1 目录为笔者解压 Xilinx 官方 uboot 后的目录

编译完成以后输入"ls"或"l"命令,结果如下图所示:

~/petalinu	x/uboot/u-boo	t-xlnx-xilin	x-v2020.15 ls	
<pre>~/petalinu doc drivers dts env examples fs include Kbuild</pre>	X/Uboot/U-boo Kconfig lib Licenses MAINTAINERS Makefile net post README	scripts spl System.map test tools u-boot u-boot.bin u-boot.cfg	<pre>x-v2020.1\$ ls u-boot.cfg.configs u-boot.dtb u-boot-dtb.bin u-boot-dtb.img u-boot.elf u-boot.elf.lds u-boot.elf.o u-boot.img</pre>	u-boot.lds u-boot.map u-boot-nodtb.bin u-boot.srec u-boot.sym
	<pre>~/petalinu doc drivers dts env examples fs include Kbuild</pre>	<pre>c~/petalinux/uboot/u-boo doc Kconfig drivers lib dts Licenses env MAINTAINERS examples Makefile fs net include post Kbuild README</pre>	<pre>% /petalinux/uboot/u-boot-xlnx-xilin doc Kconfig scripts drivers lib spl dts Licenses System.map env MAINTAINERS test examples Makefile tools fs net u-boot include post u-boot.bin Kbuild README u-boot.cfg</pre>	<pre>c~/petalinux/uboot/u-boot-xlnx-xilinx-v2020.1\$ ls doc Kconfig scripts u-boot.cfg.configs drivers lib spl u-boot.dtb dts Licenses System.map u-boot-dtb.bin env MAINTAINERS test u-boot-dtb.img examples Makefile tools u-boot.elf fs net u-boot u-boot.elf.lds include post u-boot.bin u-boot-elf.o Kbuild README u-boot.cfg u-boot.img </pre>

图 14.1.3 编译结果

从上图可以看出,生成了 u-boot 文件,编译成功。

我们在编译的时候需要输入 ARCH 和 CORSS_COMPILE 这两个变量的值,有点麻烦,如 果使用 Petalinux 就没有这个问题了,现在我们没有使用 Petalinux,不过也可以通过以下两种 方式解决:

方式一: (不推荐)可以直接在顶层 Makefile 中直接给 ARCH 和 CORSS_COMPILE 赋 值,如下图所示的那样:

原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php

正点原子



图 14.1.4 添加 ARCH 和 CROSS_COMPILE 值

上图中的 266、267 行就是直接给 ARCH 和 CROSS_COMPILE 赋值,这样我们就可以使 用如下简短的命令来编译 uboot 了:

make xilinx_zynq_virt_defconfig

make V=1 -j8

方式二: (推荐) 创建 shell 脚本。在 uboot 根目录(对应笔者的就是~/petalinux/uboot/u-boot-xlnx-xilinx-v2020.1 目录) 下创建一个名为 zynq_zc702.sh 的 shell 脚本, 然后在 shell 脚本 里面输入如下内容:

示例代码 zynq_zc702.sh 文件

1 #!/bin/bash

2 make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi- distclean

3 make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi- xilinx_zynq_virt_defconfig

4 make V=1 ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi- -j8

记得给 zynq_zc702.sh 这个文件可执行权限,使用 zynq_zc702.sh 脚本编译 uboot 的时候每 次都会清理一下工程,然后全部重新编译,编译的时候直接执行这个脚本就行了,命令如下:

./zynq_zc702.sh

编译完成以后会生成 u-boot.bin、u-boot 等文件,但是这些文件是 Xilinx 官方 ZYNQ 开发板的。在第六章我们没有单独配置 uboot,使用的是 Petalinux 工具默认的配置,使用上是没有问题的。不过那时我们使用了 Petalinux 工具,Petalinux 会根据硬件描述文件自动配置 uboot,现在我们没有使用 Petalinux 工具来编译 uboot,编译完成后看能不能用到正点原子的 ZYNQ 开发板上呢?下面我们拭目以待。

14.1.3 验证与驱动测试

如果使用 Petalinux 工具,测试还是比较方便的,可以直接生成 BOOT.bin 文件,现在我 们没有使用 Petalinux 工具,也不在 Petalinux 工程目录下编译,所以测试起来还是比较麻烦的。 当然了,也可以借助 Vitis 软件,这种方法是可行的,不过还是较麻烦,有没有一种更简便的 方法呢?

答案是有的,不过首先说明一点,这种方法笔者测试是可行的,各人可能因为软硬件的问题不一定能够顺利运行,这时可以使用 Vitis 软件来下载。下面我们介绍下这种方法。

这种方法是使用 JTAG 下载,跟使用 Vitis 软件一样的,不过这种方式更便捷,因为是脚本化。需要提醒的是 Linux 需要手动安装 JTAG 驱动,安装方式见第五章的 5.6 小节 Linux 系 统安装 JTAG cable 驱动。

首先我们进入第六章建立的 Petalinux 工程目录中,在工程的 project-spec 目录下有一个 hw-description 文件夹,如下图所示:

领航者 ZYNQ 之嵌入式 Linux 开发指南
 ② 正点原子
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③
 ③

图 14.1.5 hw-description 文件夹

该文件夹的内容和我们从 Vivado 软件打开 Vitis 时创建的 system_wrapper_hw_platform_0 文件夹的内容基本相同,我们需要的是里面的 PS 初始化文件。我们将该文件夹复制到 petalinux/uboot 根目录下,如下图所示:

wmq@Linux:~	~/pe	etali	inux,	/uboot/u·	-boot	-xlr	ıx-xili	<pre>Inx-v2020.1\$ ll hw-description/</pre>
总用量 7032	2							
drwxr-xr-x	3	wmq	wmq	4096	6月	29	15:32	./
drwxrwxr-x	26	wmq	wmq	4096	8月	1	11:36	••/
drwxrwxr-x	3	wmq	wmq	4096	6月	29	15:32	drivers/
- rw- r r	1	wmq	wmq	17	6月	21	09:21	metadata
- rw- rw- r	1	wmq	wmq	562362	6月	29	15:32	ps7_init.c
- rw- rw- r	1	wmq	wmq	562979	6月	29	15:32	ps7_init_gpl.c
- rw-rw-r	1	wmq	wmq	4418	6月	29	15:32	ps7_init_gpl.h
- rw- rw- r	1	wmq	wmq	3800	6月	29	15:32	ps7_init.h
- rw- rw- r	1	wmq	wmq	3088462	6月	29	15:32	ps7_init.html
- rw- rw- r	1	wmq	wmq	37538	6月	29	15:32	ps7_init.tcl
- rw-rw-r	1	wmq	wmq	2083850	6月	29	15:32	system_wrapper.bit
-rwxr-xr-x	1	wmq	wmq	821986	6月	29	15:32	system.xsa*
wmq@Linux:~	~/pe	etali	inux,	/uboot/u·	-boot	-xlr	nx-xili	lnx-v2020.1\$

图 14.1.6 uboot 根目录下的 hw-description 文件夹

现在我们在 petalinux/uboot 根目录下新建一个名为 linux.tcl 的文件,用来下载 fgpa 的 bitstream 文件和 uboot 的 elf 文件以启动 linux 内核,文件内容如下:

```
connect
source hw-description/ps7_init.tcl
targets -set -filter {name =~"APU*" && jtag_cable_name =~ "Digilent*"} -index 0
rst -system
after 3000
targets -set -filter {jtag_cable_name =~ "Digilent*" && level==0} -index 1
fpga -file hw-description/system_wrapper.bit
targets -set -filter {name =~"APU*" && jtag_cable_name =~ "Digilent*"} -index 0
loadhw -hw hw-description/system.xsa-mem-ranges [list {0x40000000 0xbffffff]]
configparams force-mem-access 1
targets -set -filter {name =~"APU*" && jtag_cable_name =~ "Digilent*"} -index 0
ps7_init
ps7_post_config
targets -set -nocase -filter {name =~ "ARM*#0"}
```



原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php

dow u-boot.elf

configparams force-mem-access 0

targets -set -nocase -filter {name =~ "ARM*#0"}

con

然后新建一个名为 uboot.tcl 的文件。当我们只是调试 uboot,不启动内核的时候或没有使 用 fpga 部分时启动内核的时候,可使用该文件,文件内容如下(该文件除去了下载 bitstream 文件的步骤,更节省时间):

connect source hw-description/ps7_init.tcl targets -set -filter {name =~''APU''} loadhw hw-description/system.xsa stop ps7_init targets -set -nocase -filter {name =~ ''ARM*#0''} #rst -processor dow u-boot.elf con

至此,我们可以开始下载 uboot。

将领航者开发板的启动模式设置为"JTAG"启动,连接 JTAG、串口和电源,然后开发 板上电。打开串口软件如 MobaXterm 或 cuteCom,设置好领航者开发板所使用的串口并打开。

在Vmware软件的菜单栏点击"虚拟机(M)"菜单,在弹出的子菜单中移动到"可移动 设备(D)":会弹出相应的移动设备,选择里面带有"Digilent USB"的是 JTAG 的 USB 接 口,将该接口连接到虚拟机中,如下图所示:

🖬 Ubuntu 64 位 - VMware Workstation								
文件(F) 编辑(E) 查看(V)	虚	以机(<u>M)</u> 选项卡(<u>T</u>)帮助(<u>H</u>)	<mark> -</mark> 母 の	Į 🚇		2		
	\bigcirc	电源(P)	>					
	\odot	可移动设备(D)	>		CD/DVD (SATA)	>		
		暂停(U)	Ctrl+Shift+P	~	网络适配器	>	11	
	귝	发送 Ctrl+Alt+Del(E)			打印机	>		
	Ľ	抓取輸入内容(I)	Ctrl+G		声卡	>		
		(H)H22	>		Realtek 802.11n WLAN Adapter	>		
	G	(N)	>		Future Devices Diailent USB Device	Þ		连接(断开与 主机 的连接)(C)
	-109	捕获屏幕 (C)	Ctrl+Alt+PrtScn		QinHeng USB Serial	>		更改图标(I)
	ß	答理(M)	```				~	在状态栏中显示(S)
	6	百姓(M) 重新安装 VMware Tools(T)	/					
		主机交换 viniware roois(i)						
	1	设置(S)	Ctrl+D				1	

图 14.1.7 在 Vmware 中连接 JTAG 的 USB 接口到虚拟机内

现在我们在终端中输入如下命令配置 Petalinux 的环境变量:

source /opt/pkg/petalinux/2020.2/settings.sh //或使用 spl 别名

因为这种方法使用的工具是 Petalinux 自带的, Vitis 软件也有, 这里我们使用 Petalinux 中 的,所以需要配置 Petalinux 的环境变量。

在 uboot 根目录下,输入如下命令下载 u-boot 文件,也就是 u-boot.elf 文件: /opt/pkg/petalinux/2020.2/tools/xsct/bin/xsct uboot.tcl



原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php

注意:若读者使用的是 7010 的板子,则应该重新配置第六章的 petalinux 工程,否则通过 xsct 下载 u-boot.elf 文件会报错。

首先将领航者开发板 A 盘->4_Source_Code->3_Embedded_Linux->vivado_prj->vivado_prj 下的 7010 工程的 system_wrapper.xsa 文件导入到第六章的 petalinux 工程中:

petalinux-config -get-hw-description ../xsa_7010

然后重新编译 petalinux 工程

Petalinux-build

最后将第六章重新生成的 hw-description 文件夹复制到 uboot 的根目录下(笔者的 uboot 目录为~/petalinux/uboot/alientek-uboot-v2020.1)。

xsct 是 Xilinx 软件命令行工具,基于 tcl 脚本,方便开发和调试。若执行结果处出现下图 所示的报错信息,说明 JTAG 下载器没有接入到虚拟机中:



图 14.1.8 JTAG 下载器没有接入到虚拟机中

重新将 JTAG 的 USB 接口连接到虚拟机,然后重新通过 xsct 命令下载 uboot,操作结果如下图所示:

wmq@Linux:~/petalinux/uboot/u-boot-xlnx-xilinx-v2020.1\$ /opt/pkg/petalinux/2020.2/ tools/xsct/bin/xsct uboot.tcl attempting to launch hw_server ***** Xilinx hw_server v2020.2 **** Build date : Nov 8 2020 at 18:16:39 ** Copyright 1986-2020 Xilinx, Inc. All Rights Reserved. INFO: hw_server application started INFO: Use Ctrl-C to exit hw_server application INFO: To connect to this hw_server instance use url: TCP:127.0.0.1:3121 INFO: [Hsi 55-2053] elapsed time for repository (/opt/pkg/petalinux/2020.2/tools/x sct/data/embeddedsw) loading 0 seconds hsi::open_hw_design: Time (s): cpu = 00:00:02 ; elapsed = 00:00:06 . Memory (MB): peak = 1740.590 ; gain = 452.352 ; free physical = 191 ; free virtual = 6615 wmq@Linux:~/petalinux/uboot/u-boot-xlnx-xilinx-v2020.1\$

图 14.1.9 下载 bitstream 文件和 u-boot 文件

串口软件接收到的情况如下图所示:

原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php

正点原子



图 14.1.10 串口软件接收情况

从上图可以看到,串口软件没有接收到任何启动信息,有两种可能,一是下载有问题, 二是这种命令行编译确实不适用于我们自己的开发板。笔者经过多次测试之后排除了第一种 可能,不信可以看后面我们下载移植好的,就能正常打印启动信息。那么就是第二种可能了。

虽然我们可以直接在官方的开发板上配置文件中直接修改,使uboot可以完整的运行在我们的板子上。但是从学习的角度来讲,这样我们就不能了解到uboot是如何添加新平台的。接下来我们就参考 Xilinx 官方的 ZYNQ 开发板,学习如何在 uboot 中添加我们自己的开发板或开发平台。

14.2 在 U-Boot 中添加自己的开发板

14.2.1 添加开发板默认配置文件

先在 configs 目录下创建领航者开发板的默认配置文件。复制 xilinx_zynq_virt_defconfig, 然后重命名为 zynq_altk_defconfig, 命令如下:

cd configs

cp xilinx_zynq_virt_defconfig zynq_altk_defconfig

然后修改 zynq_altk_defconfig 文件,修改后的文件内容如下:

示例代码 zynq_altk_defconfig 文件

- 1 CONFIG_ARM=y
- 2 CONFIG_SYS_CONFIG_NAME="zynq_altk"
- 3 CONFIG_SPL_SYS_DCACHE_OFF=y
- 4 CONFIG_ARCH_ZYNQ=y
- 5 CONFIG_SYS_TEXT_BASE=0x4000000
- 6 CONFIG_SPL_STACK_R_ADDR=0x200000
- 7 CONFIG_SPL=y
- 8 CONFIG_CMD_ZYNQ_AES=y
- 9 CONFIG_DISTRO_DEFAULTS=y
- 10 CONFIG_SYS_CUSTOM_LDSCRIPT=y
- 11 CONFIG_SYS_LDSCRIPT="arch/arm/mach-zynq/u-boot.lds"



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

```
12 CONFIG_FIT=y
```

- 13 CONFIG_FIT_SIGNATURE=y
- 14 CONFIG_FIT_VERBOSE=y
- 15 CONFIG_BOOTCOMMAND="run default_bootcmd"
- 16 CONFIG_SPL_FIT_PRINT=y
- 17 CONFIG_SPL_LOAD_FIT=y
- 18 CONFIG_LEGACY_IMAGE_FORMAT=y
- 19 CONFIG_USE_PREBOOT=y
- 20 CONFIG_SPL_STACK_R=y
- 21 CONFIG_SPL_FPGA_SUPPORT=y
- 22 CONFIG_SPL_OS_BOOT=y
- 23 CONFIG_SPL_SPI_LOAD=y
- 24 CONFIG_SYS_SPI_U_BOOT_OFFS=0x100000
- 25 # CONFIG_BOOTM_NETBSD is not set
- 26 CONFIG_CMD_IMLS=y
- 27 CONFIG_CMD_THOR_DOWNLOAD=y
- 28 CONFIG_CMD_MEMTEST=y
- 29 CONFIG_SYS_ALT_MEMTEST=y
- 30 CONFIG_CMD_DFU=y
- 31 CONFIG_CMD_FPGA_LOADBP=y
- 32 CONFIG_CMD_FPGA_LOADFS=y
- 33 CONFIG_CMD_FPGA_LOADMK=y
- 34 CONFIG_CMD_FPGA_LOADP=y
- 35 CONFIG_CMD_GPIO=y
- 36 CONFIG_CMD_I2C=y
- 37 CONFIG_CMD_MMC=y
- 38 CONFIG_CMD_NAND_LOCK_UNLOCK=y
- 39 CONFIG_CMD_USB=y
- 40 # CONFIG_CMD_SETEXPR is not set
- 41 CONFIG_CMD_TFTPPUT=y
- 42 CONFIG_CMD_CACHE=y
- 43 CONFIG_CMD_EXT4_WRITE=y
- 44 CONFIG_DEFAULT_DEVICE_TREE="zynq-altk"
- 45 CONFIG_OF_LIST="zynq-altk zynq-zc702 zynq-zc706 zynq-zc770-xm010 zynq-zc770-xm011 zynq-zc770-xm012 zynq-zc770-xm013 zynq-cc108"
 - 46 CONFIG_ENV_IS_IN_SPI_FLASH=y
 - 47 CONFIG_SYS_RELOC_GD_ENV_ADDR=y
 - 48 CONFIG_NET_RANDOM_ETHADDR=y
 - 49 CONFIG_SPL_DM_SEQ_ALIAS=y
 - 50 CONFIG_DFU_MMC=y
 - 51 CONFIG_DFU_RAM=y



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

- 52 CONFIG_FPGA_XILINX=y
- 53 CONFIG_FPGA_ZYNQPL=y
- 54 CONFIG_DM_GPIO=y
- 55 CONFIG_DM_I2C=y
- 56 CONFIG_SYS_I2C_CADENCE=y
- 57 CONFIG_I2C_MUX=y
- 58 CONFIG_I2C_MUX_PCA954x=y
- 59 CONFIG_LED=y
- 60 CONFIG_LED_GPIO=y
- 61 CONFIG_MISC=y
- 62 CONFIG_I2C_EEPROM=y
- 63 CONFIG_SYS_I2C_EEPROM_ADDR=0x0
- 64 CONFIG_SYS_I2C_EEPROM_ADDR_OVERFLOW=0x0
- 65 CONFIG_MMC_SDHCI=y
- 66 CONFIG_MMC_SDHCI_ZYNQ=y
- 67 CONFIG_MTD=y
- 68 CONFIG_MTD_NOR_FLASH=y
- 69 CONFIG_FLASH_CFI_DRIVER=y
- 70 CONFIG_CFI_FLASH=y
- 71 CONFIG_SYS_FLASH_USE_BUFFER_WRITE=y
- 72 CONFIG_SYS_FLASH_CFI=y
- 73 CONFIG_MTD_RAW_NAND=y
- 74 CONFIG_NAND_ZYNQ=y
- 75 CONFIG_SF_DEFAULT_SPEED=30000000
- 76 CONFIG_SPI_FLASH_ISSI=y
- 77 CONFIG_SPI_FLASH_MACRONIX=y
- 78 CONFIG_SPI_FLASH_SPANSION=y
- 79 CONFIG_SPI_FLASH_STMICRO=y
- 80 CONFIG_SPI_FLASH_SST=y
- 81 CONFIG_SPI_FLASH_WINBOND=y
- 82 CONFIG_PHY_MARVELL=y
- 83 CONFIG_PHY_REALTEK=y
- 84 CONFIG_PHY_XILINX=y
- 85 CONFIG_MII=y
- 86 CONFIG_ZYNQ_GEM=y
- 87 CONFIG_DEBUG_UART_ZYNQ=y
- 88 CONFIG_DEBUG_UART_BASE=0xe0000000
- 89 CONFIG_DEBUG_UART_CLOCK=100000000
- 90 CONFIG_ZYNQ_SERIAL=y
- 91 CONFIG_ZYNQ_SPI=y
- 92 CONFIG_ZYNQ_QSPI=y



原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php

93 CONFIG_USB=y

94 CONFIG USB EHCI HCD=y

95 CONFIG_USB_ULPI_VIEWPORT=y

96 CONFIG USB ULPI=v

97 CONFIG_USB_GADGET=y

98 CONFIG USB GADGET MANUFACTURER="Xilinx"

99 CONFIG_USB_GADGET_VENDOR_NUM=0x03fd

100 CONFIG_USB_GADGET_PRODUCT_NUM=0x0300

101 CONFIG_CI_UDC=y

102 CONFIG USB GADGET DOWNLOAD=y

103 CONFIG_USB_FUNCTION_THOR=y

104 CONFIG_DISPLAY=y

105 CONFIG SPL GZIP=y

可以看出, zynq_altk_defconfig文件基本和 xilinx_zynq_virt_defconfig 文件中的内容一样。 修改的内容如下:

首先将所有的 zc702 字样替换为 altk, 然后修改第 2 行,将 CONFIG_SYS_CONFIG_NA ME 设为 zyng altk;

修改了第 44 行,指定领航者开发板的设备树文件 zynq-altk,对应的是我们在 14.2.4 节创 建的领航者开发板的设备树文件名:

第 15 行的添加 CONFIG_BOOTCOMMAND="run default_bootcmd";

第87行添加 CONFIG_DEBUG_UART_ZYNQ=y;

第 88 行添加 CONFIG_DEBUG_UART_BASE=0xe0000000,该定义表示调试串口寄存器 的基地址:

第 89 行添加 CONFIG_DEBUG_UART_CLOCK=100000000。

领航者开发板使用的是 PS 的串口 0 作为 PS 的调试串口。查阅 ug585《Zynq-7000 SoC Technical Reference Manual》参考手册, UARTO 的寄存器基地址是 0xe0000000, 如下图所示:

uart0	UART	0xE0000000	Universal Asynchronous Receiver Transmitter
uart1	UART	0xE0001000	Universal Asynchronous Receiver Transmitter

图 14.2.1 UARTO 寄存器地址

因而我们需要将宏 CONFIG DEBUG UART BASE 的值设为 0xe0000000。

14.2.2 添加开发板对应的头文件

在目录 include/configs 下添加领航者开发板对应的头文件。我们的领航者开发板是参考 xilinx 的 zc702 开发板的硬件设计的,我们应当以 zynq_zc702.h 文件为模板进行修改,但是这 里并没有 zynq_zc702.h 头文件。所以这里我们只能自定义一个 zynq_zc702.h 文件,在终端输 入如下命令:

cd include/configs/ vim zynq_zc702.h 执行结果如下图所示:



原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php

wmq@Linux:~/petalinux/uboot/u-boot-xlnx-xilinx-v2020.1/configs\$ cd ../include/configs wmq@Linux:~/petalinux/uboot/u-boot-xlnx-xilinx-v2020.1/include/configs\$ vim zynq zc702.h vmq@Linux:~/petalinux/uboot/u-boot-xlnx-xilinx-v2020.1/include/configs\$

图 14.2.2 创建 zyng zc702.h 文件

紧接着在 zyng zc702.h 文件中输入下列代码:

示例代码 zyng zc702.h 文件

#ifndef __CONFIG_ZYNQ_ZC70X_H #define __CONFIG_ZYNQ_ZC70X_H

#include <configs/zynq-common.h>

#endif /* __CONFIG_ZYNQ_ZC70X_H */

然后复制 zynq_zc702.h 文件,并重命名为 zynq_altk.h,操作命令如下图所示:

wmq@Linux:~/petalinux/uboot/u-boot-xlnx-xilinx-v2020.1/include/configs\$ cp zynq zc702.h zynq_altk.h wmq@Linux:~/petalinux/uboot/u-boot-xlnx-xilinx-v2020.1/include/configsS

图 14.2.3 复制 zvng zc702.h

然后将 zynq_altk.h 文件中的<configs/zynq-common.h>改为<configs/altk-common.h>。因为 我们需要修改 zynq-common.h 的内容,为了防止破坏原有文件的内容,所以将其复制一份, 名为 altk-common.h。

修改完成后保存退出,操作命令如下所示:

cp zynq-common.h altk-common.h

vim altk-common.h

wmq@Linux:~/petalinux/uboot/u-boot-xlnx-xilinx-v2020.1/include/configs\$ cp zynq-common.h altk-common.h wmq@Linux:~/petalinux/uboot/u-boot-xlnx-xilinx-v2020.1/include/configs\$ vim altk-common.h wmq@Linux:~/petalinux/uboot/u-boot-xlnx-xilinx-v2020.1/include/configs\$

图 14.2.4 修改 altk-common.h 文件

修改的内容主要是第 200 行起的 CONFIG_EXTRA_ENV_SETTINGS 内容。修改后的内容 如下所示:

198 /* Default environment */	
199/* Default environment */	
200#:fndaf CONEIC EVTDA	END

200#ifndef CONFIG_EXTRA_ENV_SETTINGS

201#define CONFIG_EXTRA_ENV_SETTINGS \

"fdt high=0x2000000\0" 202

- 203 "initrd_high=0x2000000\0" \
- "scriptaddr=0x20000\0" 204
- "script_size_f= $0x40000\0$ " 205
- 206 "fdt_addr_r=0x1f00000\0"
- "pxefile addr r=0x200000\0" 207
- "kernel addr r=0x200000\0" 208



原子哥在	生线教学: www.yuanzige.com 论	坛:www.openedv.com/forum.php
209	"scriptaddr=0x300000\0" \	
210	"ramdisk_addr_r=0x3100000\0" $\$	
211	"ethaddr=00:0a:35:00:01:22\0" \	
212	"kernel_image=image.ub\0" \	
213	"kernel_load_address= $0x2008000\0"$	
214	"ramdisk_image=uramdisk.image.gz 0 " \	
215	"ramdisk_load_address=0x4000000\0" $\$	
216	"devicetree_image=devicetree.dtb\0" \	
217	"devicetree_load_address=0x2000000\0" \	
218	"bitstream_image=system.bit.bin\0" \	
219	"boot_image=BOOT.bin\0" \	
220	"loadbit_addr=0x100000\0" \	
221	"loadbootenv_addr=0x2000000\0" \	
222	"kernel_size=0x500000\0" \	
223	"devicetree_size=0x20000\0" \	
224	"ramdisk_size=0x5E0000\0" \	
225	"boot_size=0xF00000\0" \	
226	"fdt_high=0x2000000\0" \	
227	"initrd_high=0x2000000\0" \	
228	"bootenv=uEnv.txt\0" \	
229	"loadbootenv=load mmc 0 \${loadbootenv_ad	dr} ${bootenv} 0" $
230	"importbootenv=echo Importing environmen	from SD; " \
231	"env import -t \${loadbootenv_addr} \$files	ze\0" \
232	"sd_uEnvtxt_existence_test=test -e mmc 0 /u	Env.txt\0" \
233	"preboot=if test \$modeboot = sdboot && env	run sd_uEnvtxt_existence_test; "\
234	"then if env run loadbootenv; " \	
235	"then env run importbootenv; " \	
236	"fi; " \	
237	"fi; \0" \	
238	"mmc_loadbit=echo Loading bitstream from	SD/MMC/eMMC to RAM && " \
239	"mmcinfo && " \	
240	"load mmc 0 \${loadbit_addr} \${bitstream_	image} && " \
241	"fpga load 0 \${loadbit_addr} \${filesize}\0	' \
242	"norboot=echo Copying Linux from NOR fla	sh to RAM && " \
243	"cp.b 0xE2100000 \${kernel_load_address	\${kernel_size} && "\
244	"cp.b 0xE2600000 \${devicetree_load_addr	ress} \${devicetree_size} && "\
245	"echo Copying ramdisk && " \	
246	"cp.b 0xE2620000 \${ramdisk_load_addres	s} \${ramdisk_size} && " \
247	"bootm \${kernel_load_address} \${ramdisk	<pre>x_load_address } \${devicetree_load_address }\0" \</pre>
248	"qspiboot=echo Copying Linux from QSPI fl	ash to RAM && " \
0.10		

249 "sf probe 0 0 0 && " \

领航者 ZYNQ 之嵌入式 Linux 开发指南 原子哥在线教学: www.yuanzige.com 论坛:w



〔 子哥	午线教学:www.yuanzige.com 论坛:www.openedv.com/forum.php
250	"sf read \${kernel_load_address} 0x100000 \${kernel_size} && " \
251	"sf read \${devicetree_load_address} 0x600000 \${devicetree_size} && "\
252	"echo Copying ramdisk && " \
253	"sf read \${ramdisk_load_address} 0x620000 \${ramdisk_size} && " \
254	"bootm \${kernel_load_address} \${ramdisk_load_address} \${devicetree_load_address}\0" \
255	"uenvboot=" \
256	"if run loadbootenv; then "
257	"echo Loaded environment from \${bootenv}; "\
258	"run importbootenv; " \
259	"fi; " \
260	"if test -n \$uenvcmd; then "
261	"echo Running uenvcmd; " \
262	"run uenvcmd; "
263	"fi\0" \
264	"sdboot=if mmcinfo; then " \
265	"run uenvboot; " \
266	"echo Copying Linux from SD to RAM && " \
267	"load mmc 0 \${kernel_load_address} \${kernel_image} && " \
268	"bootm \${kernel_load_address}; " \
269	"fi\0" \
270	"netboot=tftpboot \${kernel_load_address} \${kernel_image} && bootm\0" \
271	"default_bootcmd=run sdboot;\0" \
272	"usbboot=if usb start; then " \
273	"run uenvboot; " \
274	"echo Copying Linux from USB to RAM && " \
275	"load usb 0 \${kernel_load_address} \${kernel_image} && " \
276	"load usb 0 \${devicetree_load_address} \${devicetree_image} && "\
277	"load usb 0 \${ramdisk_load_address} \${ramdisk_image} && " \
278	"bootm \${kernel_load_address} \${ramdisk_load_address} \${devicetree_load_address}; " \
279	"fi\0" \
280	"nandboot=echo Copying Linux from NAND flash to RAM && " \
281	"nand read \${kernel_load_address} 0x100000 \${kernel_size} && "\
282	"nand read \${devicetree_load_address} 0x600000 \${devicetree_size} && "\
283	"echo Copying ramdisk && " \
284	"nand read \${ramdisk_load_address} 0x620000 \${ramdisk_size} && " \
285	"bootm \${kernel_load_address} \${ramdisk_load_address} \${devicetree_load_address}\0" \
286	"jtagboot=echo TFTPing Linux to RAM && " \
287	"tftpboot \${kernel_load_address} \${kernel_image} && " \
288	"tftpboot \${devicetree_load_address} \${devicetree_image} && " \
289	"tftpboot \${ramdisk_load_address} \${ramdisk_image} && " \
290	"bootm \${kernel_load_address} \${ramdisk_load_address} \${devicetree_load_address}\0" \



314#endif

可以看到 CONFIG EXTRA ENV SETTINGS 宏设置的是 uboot 如何加载内核镜像,并且 设置了各种启动方式如 NOR 启动的 norboot、QSPI 启动的 qspiboot、SD 卡启动的 sdboot,甚 至是 usb 启动的 usbboot,不过 zyng-7000 系列是不支持 usb 启动的, zyng ultrascale 系列是支 持的。此外各种启动方式还有 rsa 启动如 rsa_sdboot,这是一种 rsa 认证的安全启动方式,通常 用于工业安全方面,具体的笔者也没研究过。下面讲解要修改的内容。

将第 212 行的 uImage 修改为 image.ub,因为 zyng 使用的内核镜像文件是 image.ub 文件。

由于我们一般都是通过 SD卡启动,可以看到, SD卡启动方式先从 mmc(此处指 SD卡) 中加载内核镜像文件 kernel_image, 然后加载设备树文件 devicetree_image, 最后加载根文件 系统镜像文件 ramdisk_image,从而启动 linux 系统,但我们使用的内核镜像文件 image.ub 中 已经包括了内核和设备树。另外对于根文件系统,一般都是直接放到 SD 卡的 ext4 分区,或 者使用 INITRAMFS 格式,而不是通过加载 ramdisk_image 文件启动的,所以对于 sdboot 我们 将其修改为:

"sdboot=if mmcinfo; then " $\$ "run uenvboot; " \ "echo Copying Linux from SD to RAM... && " \ "load mmc 0 \${kernel_load_address} \${kernel_image} && "\ "bootm \${kernel load address}; "\



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

"fi\0" \

除了经常使用的 SD 卡启动方式外,有时方便调试也会用网络启动,所以我们还需添加网络启动内核方式 netboot,当然了默认启动还是设置为 SD 卡启动,所以在 sdboot 后面添加下面两行:

"netboot=tftpboot \${kernel_load_address} \${kernel_image} && bootm\0" \

"default_bootcmd=run sdboot;\0" \

如下图所示:

255	"sdboot=if mmcinfo; then " 🔪
256	"run uenvboot; " \
257	"echo Copying Linux from SD to RAM && "
258	"load mmc 0 \${kernel_load_address} \${kernel_image} && " \
259	"bootm \${kernel load address}; " \
260	"fi\0" \
261	<pre>"netboot=tftpboot \${kernel_load_address} \${kernel_image} && bootm\0" \</pre>
262	"default bootcmd=run sdboot;\0" \

图 14.2.5 修改 sdboot 并添加 netboot

修改完成后保存退出即可。

14.2.3 添加开发板对应的板级文件夹

uboot 中每个板子都有一个对应的文件夹来存放板级文件,比如开发板上外设驱动文件等。 Xilinx 的 ZYNQ 系列芯片的所有板级文件夹都存放在 board/xilinx/zynq 目录下,在这个目录下 有个名为 zynq-zc702 的文件夹,这个文件夹就是 Xilinx 官方 ZC702 开发板的板级文件夹。复 制 zynq-zc702,将其重命名为 zynq-altk,命令如下:

cd board/xilinx/zynq/

cp -r zynq-zc702 zynq-altk

进入 zynq-altk 目录中,可以看到只有一个名为 "ps7_init_gpl.c" 的文件,该文件是 PS 的 初始化文件,可以用于我们的领航者开发板。

14.2.4 添加开发板对应的设备树

uboot 支持设备树,每个开发板都有一个对应的设备树文件。Xilinx 的 ZYNQ 系列芯片的 所有设备树文件夹都存放在 arch/arm/dts 目录下,在这个目录下有个名为 zynq-zc702.dts 的文 件,该文件是 ZC702 开发板的设备树文件。这里我们就不参照 zynq-zc702.dts 文件,而是参照 zynq-zed.dts 文件,这是因为 zynq-zed.dts 是在 zynq-zc702.dts 文件基础上修改而来,能极大的 方便我们的移植。我们将 zynq-zed.dts 重命名为 zynq-altk.dts,命令如下:

cd arch/arm/dts

```
cp zynq-zed.dts zynq-altk.dts
拷贝完成以后将 zynq-altk.dts 文件的内容修改成如下所示:
示例代码 zynq-altk.dts 文件
6 /dts-v1/;
7 #include "zynq-7000.dtsi"
8
```

9 / {



```
原子哥在线教学: www.yuanzige.com
                                                  论坛:www.openedv.com/forum.php
          model = "Zynq Alientek Development Board";
    10
          compatible = "avnet,zynq-altk", "xlnx,zynq-altk", "xlnx,zynq-7000";
    11
    12
    13
          aliases {
    14
            ethernet0 = \&gem0;
    15
            serial0 = \&uart0;
    16
            spi0 = \&qspi;
    17
            mmc0 = \& sdhci0;
    18
          };
    19
    20
          memory@0 {
    21
            device_type = "memory";
    22
            reg = \langle 0x0 \ 0x40000000 \rangle;
    23
          };
    24
    25
          chosen {
            bootargs = "";
    26
    27
            stdout-path = "serial0:115200n8";
    28
          };
    29
    30
          usb_phy0: phy0@e0002000 {
    31
            compatible = "ulpi-phy";
    32
            #phy-cells = <0>;
    33
            reg = <0xe0002000 0x1000>;
    34
            view-port = \langle 0x0170 \rangle;
    35
            drv-vbus;
    36
         };
    37 };
    38
    39 &clkc {
          ps-clk-frequency = <33333333>;
    40
    41 };
    42
    43 & gem0 {
          status = "okay";
    44
    45
          phy-mode = "rgmii-id";
          phy-handle = <&ethernet_phy>;
    46
    47
          ethernet_phy: ethernet-phy@0 {
    48
    49
            reg = <0>;
```

```
50 device_type = "ethernet-phy";
```



```
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
    51 };
    52 };
    53
    54 &qspi {
    55
          u-boot,dm-pre-reloc;
          status = "okay";
    56
    57
         is-dual = <0>;
    58
          num-cs = <1>;
    59
          flash@0 {
            compatible = "spansion,s25fl256s1", "jedec,spi-nor";
    60
    61
            reg = <0>;
    62
            spi-tx-bus-width = <1>;
    63
            spi-rx-bus-width = <4>;
            spi-max-frequency = <50000000>;
    64
            m25p,fast-read;
    65
    66
            #address-cells = <1>;
            \#size-cells = <1>;
    67
    68
            partition@0 {
    69
              label = "qspi-fsbl-uboot";
    70
              reg = <0x0 0x100000>;
    71
            };
    72
            partition@100000 {
    73
              label = "qspi-linux";
    74
              reg = <0x100000 0x500000>;
    75
            };
    76
            partition@600000 {
    77
              label = "qspi-device-tree";
    78
              reg = <0x600000 0x20000>;
    79
            };
    80
            partition@620000 {
    81
              label = "qspi-rootfs";
    82
              reg = <0x620000 0x5E0000>;
    83
            };
    84
            partition@c00000 {
    85
              label = "qspi-bitstream";
    86
              reg = <0xC00000 0x400000>;
    87
            };
    88
          };
    89 };
    90
    91 &sdhci0 {
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

92 u-boot,dm-pre-reloc;
93 status = "okay";
94 };
95
96 &uart0 {
97 u-boot,dm-pre-reloc;
98 status = "okay";
99 };
100
101 &usb0 {
102 status = "okay";
$103 dr_mode = "host";$
104 usb-phy = $<$ &usb_phy0>;
105 };
修改的内容如下:
首先将 zed 字样替换成 altk, 然后

将第 10 行的 model 改成"Zynq Alientek Development Board",当然也可以使用其他的名称。 将第 15 行 uart1 改成 uart0;因为我们使用的是 PS 端的 uart0 串口。

将第 22 行 memory 的大小 (PS DDR 的大小) 设置为 7020 的 1GB,也就是 "reg = <0x0 0x4000000>",如果是 7010 的核心板,就是 "reg = <0x0 0x20000000>",也就是 512MB。 将第 96 行的 uart1 改成 uart0。

由于我们在 14.2.1 节添加开发板默认配置文件中修改了配置文件中的设备树的名称为 zynq-altk.dts,这里又成功构建了 zynq-altk.dts,所以我们应该修改 arch/arm/dts下的 Makefile, 否则编译会报错,修改内容见下图所示:

234	<pre>dtb-\$(CONFIG_ARCH_ZYNQ) += \</pre>
235	bitmain-antminer-s9.dtb \
236	zynq-cc108.dtb \
237	zynq-cse-nand.dtb \
238	zynq-cse-nor.dtb \
239	zynq-cse-qspi-single.dtb \
240	zynq-cse-qspi-parallel.dtb \
241	zynq-cse-qspi-stacked.dtb \
242	zynq-cse-qspi-x1-single.dtb \
243	zynq-cse-qspi-x1-stacked.dtb \
244	zynq-cse-qspi-x2-single.dtb \
245	zynq-cse-qspi-x2-stacked.dtb \
246	zynq-dlc20-rev1.0.dtb \
247	zynq-microzed.dtb \
248	zynq-minized.dtb \
249	zynq-picozed.dtb \
250	zynq-syzygy-hub.dtb \
251	zynq-topic-miami.dtb \
252	zynq-topic-miamilite.dtb \
253	zyng-topic-miamiplus.dtb \
254	zynq-altk.dtb \
255	zvna-zc706.dtb

图 14.2.6 修改设备树文件下的 Makefile 文件



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php 修改内容为将上图中 254 行 zynq-zc702.dtb 替换为 zynq-altk.dtb 即可。 关于设备树的内容后面在讲解 linux 驱动的时候会详细讲解,这里就不细述了。

14.2.5 修改 U-Boot 图形界面配置文件

uboot 支持图形界面配置,关于 uboot 的图形界面配置下一章会详细的讲解。修改文件 arch/arm/mach-zynq/Kconfig,将第 49 行的 SYS_CONFIG_NAME 配置项的内容修改如下:

示例代码 Kconfig 文件

49	config SYS_CONFIG_NAME
50	string "Board configuration name"
51	default "zynq_altk"
52	help
53	This option contains information about board configuration name.
54	Based on this option include/configs/ <config_sys_config_name>.h header</config_sys_config_name>
55	will be used for board configuration.
修	改的是第51行的内容,修改完成以后的 Kconfig 文件如下图所示:
46	config SYS_SOC
48	deradte zyng
49	config SYS_CONFIG_NAME
50	string "Board configuration name"
51	default "zynq_altk"
52	help
53	This option contains information about board configuration name.

Based on this option include/configs/<CONFIG_SYS_CONFIG_NAME>.h header will be used for board configuration.

图 14.2.7 修改后的 Kconfig 文件

到此为止,领航者开发板就已经添加到 uboot 中了,接下来就是编译这个新添加的开发板。

14.2.6 使用新添加的板子配置编译 uboot

在 uboot 源码根目录下新建一个名为 zynq.sh 的 shell 脚本,在这个 shell 脚本里面输入如下内容:

示例代码 zynq.sh 脚本文件

#!/bin/bash

55

make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi- distclean

make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi- zynq_altk_defconfig

make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi- -j8

第3行使用的配置文件 zynq_altk_defconfig 就是 14.2.1 节中新建的 zynq_altk_defconfig 配置文件。给予 zynq.sh 可执行权限,然后运行脚本来完成编译,命令如下:

chmod +x zynq.sh //给予可执行权限

./zynq.sh //运行脚本编译 uboot

等待编译完成,编译完成以后输入如下命令,查看一下 14.2.2 小节中添加的 zynq_altk.h 这个头文件有没有被引用。

grep -nR "zynq_altk.h"



原子哥在线教学: www.yuanzige.com 论坛:ww

论坛:www.openedv.com/forum.php

如果有很多文件都引用了 zynq_altk.h 这个头文件,那就说明新板子添加成功,如下图所

示:

wmq@Linux:~/petalinux/uboot/u-boot-xlnx-xilinx-v2020.1\$ grep -nR "zynq_altk.h"
<pre>env/.attr.o.cmd:33: include/configs/zynq_altk.h \</pre>
env/.common.o.cmd:36: include/configs/zynq_altk.h \
env/.sf.o.cmd:42: include/configs/zynq_altk.h \
env/.env.o.cmd:45: include/configs/zyng_altk.h \
env/.callback.o.cmd:33: include/configs/zynq_altk.h \
env/.flags.o.cmd:41: include/configs/zyng_altk.h \
lib/.hang.o.cmd:35: include/configs/zynq_altk.h \
lib/.list_sort.o.cmd:72: include/configs/zynq_altk.h \
lib/.fdtdec_common.o.cmd:32: include/configs/zynq_altk.h \
lib/.sha256.o.cmd:34: include/configs/zynq_altk.h \
lib/zlib/.zlib.o.cmd:33: include/configs/zynq_altk.h \
lib/libfdt/.fdt.o.cmd:36: include/configs/zynq_altk.h \
lib/libfdt/.fdt_strerror.o.cmd:36: include/configs/zynq_altk.h \

图 14.2.8 查找结果

编译完成以后就可以使用"/opt/pkg/petalinux/2020.2/tools/xsct/bin/xsct uboot.tcl"或者 "/opt/pkg/petalinux/2020.2/tools/xsct/bin/xsct linux.tcl"命令进行下载测试。串口软件输出结果 如下图所示:



图 14.2.9 uboot 启动过程

从上图可以看到,我们基本移植成功了,也验证了这种下载方式是没有问题的。uboot 的 最终目的就是启动 Linux 内核,所以还是需要通过启动 Linux 内核来判断 uboot 移植是否真的 成功。在启动 Linux 内核之前我们先来学习两个重要的环境变量 bootcmd 和 bootargs。

14.3 bootcmd 和 bootargs 环境变量

uboot 中有两个非常重要的环境变量 bootcmd 和 bootargs,接下来看一下这两个环境变量。 bootcmd 和 bootagrs 是采用类似 shell 脚本语言编写的,里面有很多的变量引用,这些变量其 实都是环境变量,有很多是 Xilinx 自己定义的。文件 include/configs/altk-common.h 中的宏 CONFIG_EXTRA_ENV_SETTINGS 保存着这些环境变量的默认值,详细内容见 14.2.2 节添加 开发板对应的头文件处。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

宏 CONFIG_EXTRA_ENV_SETTINGS 中有很多有价值的信息,比如第 270 行的 netboot 变量,就可以让我们从网络启动 linux。

14.3.1 环境变量 bootcmd

bootcmd保存着 uboot 默认命令,uboot 倒计时结束以后就会执行 bootcmd 中的命令。这些 命令一般都是用来启动 Linux 内核的,比如读取 SD 或者 eMMC 中的 Linux 内核镜像文件和设 备树文件到 DRAM 中,然后启动 Linux 内核。可以在 uboot 启动以后进入命令行设置 bootcmd 环境变量的值。如果 QSPI 中没有保存 bootcmd 的值,那么 uboot 就会使用默认的值,开发板 第一次运行 uboot 的时候都会使用默认值来设置 bootcmd 环境变量,默认环境变量在文件 include/env_default.h 中定义。打开文件 include/env_default.h,在此文件中有如下所示内容:

示例代码 默认环境变量

14 env_t environment __UBOOT_ENV_SECTION__ = { 15 ENV_CRC, /* CRC Sum */ 16 #ifdef CONFIG_SYS_REDUNDAND_ENVIRONMENT 17 1, /* Flags: valid */ 18 #endif 19 { 20 #elif defined(DEFAULT_ENV_INSTANCE_STATIC) 21 static char default_environment[] = { 22 #else 23 const uchar default_environment[] = { 24 #endif 31 #ifdef CONFIG USE BOOTARGS 32 "bootargs=" CONFIG_BOOTARGS "\0" 33 #endif 34 #ifdef CONFIG_BOOTCOMMAND 35 "bootcmd=" CONFIG_BOOTCOMMAND "\0" 36 #endif 37 #ifdef CONFIG_RAMBOOTCOMMAND "ramboot=" CONFIG_RAMBOOTCOMMAND "\0" 38 39 #endif 40 #ifdef CONFIG_NFSBOOTCOMMAND "nfsboot=" CONFIG NFSBOOTCOMMAND "\0" 41 42 #endif 43 #if defined(CONFIG BOOTDELAY) && (CONFIG BOOTDELAY >= 0) 44 "bootdelay=" __stringify(CONFIG_BOOTDELAY) "\0" 45 #endif 91 #ifdef CONFIG_ENV_VARS_UBOOT_CONFIG "arch=" CONFIG_SYS_ARCH 92 "\0"



```
原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php
   93 #ifdef CONFIG_SYS_CPU
   94 "cpu=" CONFIG SYS CPU
                                    "\0"
   95 #endif
   96 #ifdef CONFIG_SYS_BOARD
      "board=" CONFIG SYS BOARD
   97
                                      "\0"
      "board name=" CONFIG SYS BOARD
                                          "\0"
   98
   99 #endif
   100 #ifdef CONFIG_SYS_VENDOR
   101 "vendor=" CONFIG_SYS_VENDOR
                                        "\0"
   102 #endif
   103 #ifdef CONFIG_SYS_SOC
   104 "soc="
               CONFIG_SYS_SOC
                                    "\0"
   105 #endif
   106 #endif
   107 #ifdef CONFIG_EXTRA_ENV_SETTINGS
   108 CONFIG_EXTRA_ENV_SETTINGS
   109 #endif
   110 "\0"
   111 #ifdef DEFAULT_ENV_INSTANCE_EMBEDDED
   112 }
   113 #endif
   114 };
   从上述代码中的第14行可以看出 environment 是 env t 类型的变量, env t 类型如下:
                                   示例代码 env_t 结构体
   148 typedef struct environment_s {
   149 uint32_t crc; /* CRC32 over data bytes */
   150 #ifdef CONFIG_SYS_REDUNDAND_ENVIRONMENT
   151 unsigned char flags; /* active/obsolete flags */
   152 #endif
```

```
153 unsigned char data[ENV_SIZE]; /* Environment data */
```

154 } env_t;

env_t 结构体中的 crc 为 CRC 值, flags 是标志位, data 数组就是环境变量值。因此, environment 就是用来保存默认环境变量的, 在示例代码默认环境变量中指定了很多环境变量 的默认值, 比如 bootcmd 的默认值就是 CONFIG_BOOTCOMMAND, bootargs 的默认值就是 CONFIG_BOOTARGS 。 我 们 在 zynq_altk_defconfig 文 件 中 通 过 设 置 CONFIG_BOOTCOMMAND 来设置 bootcmd 的默认值, 如下所示:

示例代码 CONFIG_BOOTCOMMAND 默认值

15 CONFIG_BOOTCOMMAND="run default_bootcmd"

看起来很简单,我们来分析下。以下分析的内容都来自 include/configs/altk-common.h 中的宏 CONFIG_EXTRA_ENV_SETTINGS 定义处。


THANK-	
原子哥上	在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php 面的 "run default_bootcmd"使用的是 uboot 的 run 命令来运行 default_bootcmd。
default	_bootcmd 是我们在 14.2.2 小节添加的,其内容如下:
278	default_bootcmd=run sdboot;\0" \
ru	n sdboot表示默认使用 sd 卡启动方式。sdboot 变量的内容如下:
264	sdboot=if mmcinfo; then " \
265	"run uenvboot; "
266	"echo Copying Linux from SD to RAM && " \
267	"load mmc 0 \${kernel_load_address} \${kernel_image} && "\
268	<pre>B "bootm \${kernel_load_address}; "\</pre>
269	"fi\0" \
ub	oot使用了类似 shell 脚本语言的方式来编写变量。sdboot 启动方式首先执行 mmcinfo 命
令,打	印 mmc 信息,执行成功后然后执行 uenvboot。uenvboot 变量的内容如下:
255	"uenvboot=" \
256	"if run loadbootenv; then "
257	"echo Loaded environment from \${bootenv}; "\
258	"run importbootenv; " \
259	"fi; " \
260) "if test -n \$uenvcmd; then " \
261	"echo Running uenvcmd; " \
262	2 "run uenvcmd; " \
263	3 "fi\0" \
ue	nvboot 首先运行 loadbootenv, loadbootenv 内容如下:
221	"loadbootenv_addr=0x200000\0" \
••••	
228	"bootenv=uEnv.txt $0"$
229	<pre>"loadbootenv=load mmc 0 \${loadbootenv_addr} \${bootenv}\0" \</pre>
也	就是从 SD 卡中加载 uEnv.txt 文件,显然 SD 卡中没有 uEnv.txt 文件,执行失败,所以
第一个	· if 条件不成立,执行第二个 if 语句,判断 uenvcmd 变量值的长度是否为零,由于
uenvcn	nd 变量未定义,所以第二个 if 语句条件不成立,直接返回到 sdboot。
在	sdboot 中执行"echo Copying Linux from SD to RAM && "输出"Copying Linux from SD
to RAN	M", 执行成功后从 mmc0 中加载内核镜像文件 kernel_image 到内存 DRAM 的
kernel_	_load_address 处。 具中 kernel_image 和 kernel_load_address 变量的内容如下:
212	2 "kernel_image=image.ub\0" \
213	"kernel_load_address=0x2080000\0" \

可以看到 kernel_load_address 变量值为 0x2080000,是文件加载到内存中的地址。

对于 bootcmd 的值,也可以在启动 uboot 后直接在 uboot 命令行中设置 bootcmd 的值,命 令如下:

setenv bootcmd ' mmcinfo && fatload mmc 0 0x2080000 image.ub; bootm 0x2080000;'

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

正点原子

14.3.2 环境变量 bootargs

bootargs保存着 uboot 传递给 Linux 内核的参数。zynq 的 bootargs 由设备树指定,在 14.2.4 节我们可以看到 bootargs 的值为空,也就是说 zynq 一般不用向 linux 内核传递参数。不过此处我们还是简单地讲解下 bootargs。以下面的命令做介绍:

setenv bootargs 'console=\${console},\${baudrate} root=\${mmcroot} rootfstype=ext4'

假设 console=ttyPS0, baudrate=115200, mmcroot=/dev/mmcblk0p2 rootwait rw, 因此将其 展开后就是:

setenv bootargs 'console=ttyPS0,115200 root=/dev/mmcblk0p2 rootwait rw rootfstype=ext4'

可以看出 bootargs 的值为 "console= ttyPS0,115200 root= /dev/mmcblk0p2 rootwait rw rootfstype=ext4。bootargs设置了很多的参数的值,这些参数Linux内核会使用到,常用的参数有:

1、 console

console 用来设置 linux 终端(或者叫控制台),也就是通过什么设备来和 Linux 进行交互, 是串口还是 LCD 屏幕。如果是串口的话应该是串口几等等。一般设置串口作为 Linux 终端, 这样我们就可以在电脑上通过串口工具来和 linux 交互了。这里设置 console 为 ttyPS0,因为 linux 启动以后 ZYNQ 的串口 0 在 linux 下的设备文件就是/dev/ttyPS0,在 Linux 下,一切皆文 件。

ttyPS0 后面有个",115200",这是设置串口的波特率,console=ttyPS0,115200 综合起来就是设置ttyPS0(也就是串口0)作为Linux的终端,并且串口波特率设置为115200。

2, root

root 用来设置根文件系统的位置,root=/dev/mmcblk0p2 用于指明根文件系统存放在mmcblk0设备的分区2中。领航者开发板启动linux以后会存在/dev/mmcblk0、/dev/mmcblk1、/dev/mmcblk0p1、/dev/mmcblk0p2、/dev/mmcblk1p1 和/dev/mmcblk1p2 这样的文件,其中/dev/mmcblkx(x=0~n)表示mmc设备,而/dev/mmcblkxpy(x=0~n,y=1~n)表示mmc设备x的分区y。在领航者开发板中/dev/mmcblk0表示SD卡,而/dev/mmcblk0p2表示SD卡的分区2。

root 后面有"rootwait rw", rootwait 表示等待 mmc 设备初始化完成以后再挂载,否则的 话 mmc 设备还没初始化完成就挂载根文件系统会出错的。rw 表示根文件系统是可以读写的, 不加 rw 的话可能无法在根文件系统中进行写操作,只能进行读操作。

3, rootfstype

此选项一般配置 root 一起使用, rootfstype 用于指定根文件系统类型, 如果根文件系统为 ext 格式的话此选项无所谓。如果根文件系统是 yaffs、jffs 或 ubifs 的话就需要设置此选项, 指 定根文件系统的类型。

bootargs 常设置的选项就这三个,后面遇到其他选项的话再讲解。

14.4 uboot 启动 Linux 测试

uboot 已经移植好了,bootcmd 和 bootargs 这两个重要的环境变量也讲解了,接下来就要 测试一下 uboot 能不能完成它的工作:启动 Linux 内核。我们测试两种启动 Linux 内核的方法,一种是直接从 SD 卡启动,一种是从网络启动。

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

正点原子

14.4.1 从 SD 卡启动 Linux 系统

首先我们将编译生成的 u-boot.elf 文件拷贝到 petalinux 工程目录的 image/linux 下,操作命 令如下图所示:

wmg@Linux:~/petalinux/uboot/u-boot-xlnx-xilinx-v2020.1\$ cp u-boot.elf ~/petalinu x/ALIENTEK-ZYNQ/images/linux/ wmq@Linux:~/petalinux/uboot/u-boot-xlnx-xilinx-v2020.1\$

图 14.4.1 拷贝 u-boot.elf 到 petalinux 工程目录下

然后使用如下命令生成 BOOT.BIN 文件, 用于从 SD 卡启动 linux 内核:

petalinux-package --boot --fsbl --fpga --u-boot --force

从 SD 卡启动也就是将编译出来的 BOOT.BIN 保存在 SD 卡中,由于目前我们还没有讲解 如何移植 linux,所以这里我们将第六章生成的 image.ub 文件也拷贝到 SD 卡中,供 uboot 从 SD卡中读取 linux 镜像文件,并启动 linux 内核。先检查一下 SD卡的分区 1 中有没有 image.ub 文件, 输入命令"ls mmc 0:1", 结果如下图所示:



图 14.4.2 SD 卡分区 1 文件

从上图中可以看出,此时 SD 卡分区 1 中存在 BOOT.BIN 和 image.ub 这两个文件,所以 我们可以测试新移植的 uboot 能不能启动 linux 内核。

直接输入 boot, 或者 run bootcmd 即可启动 Linux 内核, 如果 Linux 内核启动成功的话就 会输出如图下图所示的启动信息:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

Zynq> boot			
Device: mmc@e010000	00		
Manufacturer ID: 91	E		
OEM: 5449			
Name: SD32G			
Bus Speed: 2500000)		
Mode: SD Legacy			
Rd Block Len: 512			
SD version 3.0			
High Capacity: Yes			
Capacity: 29.1 GiB			
Bus Width: 4-bit			
Erase Group Size: S	512 Bytes		
Copying Linux from	SD to RAM		
11838820 bytes read	1 in 1003 ms (11.3 MiB/s)		
## Loading kernel f	from FIT Image at 02008000		
Using 'conf@system-top.dtb' configuration			
Verifying Hash 1	Integrity OK		
Trying 'kernel@	l' kernel subimage		
Description:	Linux kernel		
Type:	Kernel Image		
Compression:	uncompressed		
Data Start:	0x020080e8		
Data Size:	4325984 Bytes = 4.1 MiB		
Architecture:	ARM		
OS:	Linux		
Load Address:	0x00200000		
Entry Point:	0x00200000		
Hash algo:	sha256		
Hash value:	d614711fae4f373ff4181b15c184847e80ff534bfa5fb9bcb1ef7941191086dd		
Verifying Hash 1	Integrity sha256+ OK		

图 14.4.3 linux 内核启动成功

14.4.2 从网络启动 Linux 系统

从网络启动 linux 系统的唯一目的就是为了调试。不管是为了调试 linux 系统还是 linux 下 的驱动。每次修改 linux 系统文件或者 linux 下的某个驱动以后都要将其烧写到 SD 中去测试, 这样太麻烦了。我们可以设置 linux 从网络启动,也就是将 linux 镜像文件和根文件系统都放 到 Ubuntu 下某个指定的文件夹中,这样每次重新编译 linux 内核或者某个 linux 驱动以后只需 要使用 cp 命令将其拷贝到这个指定的文件夹中即可,这样就不需要频繁的烧写 SD,从而加 快开发速度。我们可以通过 nfs 或者 tftp 从 Ubuntu 中下载 image.ub 文件或者 zImage 和设备树 文件,根文件系统的话也可以通过 nfs 挂载,不过本小节我们不讲解如何通过 nfs 挂载根文件 系统,这个在讲解根文件系统移植的时候再讲解。这里我们使用 tftp 从 Ubuntu 中下载 image.ub 文件或者 zImage 和设备树文件, 前提是要将 image.ub 文件或者 zImage 和设备树文件 放到 Ubuntu 下的 tftpboot 目录中,这些文件我们在第六章编译 Petalinux 工程的时候 Petalinux 工具已经帮我们复制到/tftpboot目录中了。

首先我们参考 20.6.1 设置网络环境小节,设置网络环境,使板子与 ubuntu 系统之间能够 ping通。

然后我们通过 tftpboot 命令下载 linux 镜像(image.ub)文件到 DDR 中,操作步骤如下图 所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

Zynq> tftpboot 2080000 image.ub
Using ethernet@e000b000 device
TFTP from server 192.168.1.20; our IP address is 192.168.1.10
Filename 'image.ub'.
Load address: 0x2080000
Loading: ####################################

3.5 MiB/s
done
Bytes transferred = 11838820 (b4a564 hex)
Zynq>

图 14.4.4 使用 tftp 命令下载 image.ub 文件 紧接着使用 bootm 命令启动 linux 内核,操作步骤如下如所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

_		
2	yng> bootm 208000) 5 FIT T
	# Loading kernel : Using loopf@sus	from FII Image at 02080000
	Varifuing Wash	Integrity OV
	Truing IkernelA	libegrity OA
	Description.	i keinei subimaye
	Tume:	Linux Keinei
	Compression.	uncompressed
	Data Start.	
	Data Siza:	4325094 Bytes = 4 1 MiB
	Architecture:	1323901 Byces - 1.1 MIB
	Architecture.	Tipuy
	Load Address.	0x00200000
	Entry Point:	0x00200000
	Hash algo:	sha256
	Hash value:	d614711fae4f373ff4181b15c184847e80ff534bfa5fb9bcb1ef7941191086dc
	Verifving Hash	Integrity sha256+ OK
#	# Loading ramdisk	from FIT Image at 02080000
	Using 'conf@svs	tem-top.dtb' configuration
	Verifving Hash	Integrity OK
	Trving 'ramdisk	l' ramdisk subimage
	Description:	petalinux-image-minimal
	Type:	RAMDisk Image
	Compression:	uncompressed
	Data Start:	0x024a7308
	Data Size:	7482599 Bvtes = 7.1 MiB
	Architecture:	ARM
	OS:	Linux
	Load Address:	unavailable
	Entry Point:	unavailable
	Hash algo:	sha256
	Hash value:	4e7b95e8c7b817bc6bec7b640b627576d25b1fbb0ebe9735b364518dbd406bca
	Verifying Hash	Integrity sha256+ OK
#	# Loading fdt from	n FIT Image at 02080000
	Using 'conf@sys	tem-top.dtb' configuration
	Verifying Hash	Integrity OK
	Trying 'fdt@sys	tem-top.dtb' fdt subimage
	Description:	Flattened Device Tree blob
	Type:	Flat Device Tree
	Compression:	uncompressed
	Data Start:	0x024a0454
	Data Size:	28135 Bytes = 27.5 KiB
	Architecture:	ARM
	Hash algo:	sha256
	Hash value:	af393ddd932718bb09909acc47bb74368ad384947b090f742d457ce62b5c2948
	Verifying Hash	Integrity sha256+ OK
	Booting using th	ne fdt blob at 0x24a0454
	Loading Kernel	Image
	Loading Ramdisk	to 1e3e2000, end 1eb04ce7 OK
	Loading Device 1	<pre>Free to le3d8000, end le3elde6 OK</pre>
S	tarting kernel	
	a a b d m an T d m soon a m	

图 14.4.5 使用 "bootm" 命令启动 linux

uboot 移植到此结束,简单总结一下 uboot 移植的过程:

- 不管是购买的开发板还是自己做的开发板,基本都是参考半导体厂商的 dmeo 板,而 半导体厂商会在他们自己的开发板上移植好 uboot、linux kernel 和 systemfs 等,最终 制作好 BSP 包提供给用户。我们可以在官方提供的 BSP 包的基础上添加我们的板子, 也就是俗称的移植。
- ② 我们购买的开发板或者自己做的板子一般都不会原封不动的照抄半导体厂商的 demo 板,都会根据实际的情况来做修改,既然有修改就必然涉及到 uboot 下驱动的移植。
- ③ 一般 uboot 中需要解决串口、QSPI、EMMC 或 SD 卡、网络和 LCD 驱动,因为 uboot 的主要目的就是启动 Linux 内核,所以不需要考虑太多的外设驱动。
- ④ 在 uboot 中添加自己的板子信息,根据自己板子的实际情况来修改 uboot 中的驱动。



正点原子

第十五章 U-Boot 图形化配置及其原理

在前两章中我们是通过配置文件来配置 uboot 的,这里我们再介绍另外一种配置 uboot 的 方法,就是图形化配置,以前的 uboot 是不支持图形化配置,只有 Linux 内核才支持图形化配 置。不过不知从什么时候开始, uboot 也支持图形化配置了, 本章我们就来学习一下如何通过 图形化配置 uboot,并且学习一下图形化配置的原理,因为后面学习 Linux 驱动开发的时候可 能要修改图形配置文件。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

15.1 U-Boot 图形化配置体验

uboot 或 Linux 内核可以通过输入"make menuconfig"来打开图形化配置界面, menuconfig 是一套图形化的配置工具, 需要 ncurses 库支持。ncurses 库提供了一系列的 API 函 数供调用者生成基于文本的图形界面,因此需要先在 Ubuntu 中安装 ncurses 库(我们在安装 Petalinux 工具的时候已经安装,因而不需要再安装),命令如下:

sudo apt-get install libncurses*

menuconfig 重点会用到两个文件: .config 和 Kconfig。.config 文件前面已经说了,这个文 件保存着 uboot 的配置项,使用 menuconfig 配置完 uboot 以后肯定要更新.config 文件。Kconfig 文件是图形界面的描述文件,也就是描述界面应该有什么内容,很多目录下都会有 Kconfig 文 件。

在打开图形化配置界面之前,要先使用"make xxx_defconfig"对 uboot 进行一次默认配 置,只需要一次即可。如果使用"make clean"清理了工程的话就那就需要重新使用"make xxx_defconfig"再对 uboot 进行一次配置。进入 uboot 源码根目录,输入如下命令:

make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi- zynq_altk_defconfig

make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi- menuconfig

如果显示如下所示错误信息:

Your display is too small to run Menuconfig!

It must be at least 19 lines by 80 columns.

scripts/kconfig/Makefile:37: recipe for target 'menuconfig' failed

make[1]: *** [menuconfig] Error 1

Makefile:479: recipe for target 'menuconfig' failed

make: *** [menuconfig] Error 2

这是因为终端窗口有点小了,用鼠标调大点就可以了。

打开后的图形化界面如下图所示:

U-Boot 2020.01 Configuration Arrow keys navigate the menu. <enter> selects submenus> (or empty submenus). Highlighted letters are hotkeys. Pressing <y> includes, <n> excludes, <m> modularizes features. Press <esc> to exit, <? > for Help, for Search. Legend: [*] built-in []</esc></m></n></y></enter>
Architecture select (ARM architecture)> ARM architecture> General setup> Boot images> API> Boot timing> Boot media> (2) delay in seconds before automatically booting
<pre>[] Enable boot arguments [*] Enable a default value for bootcmd</pre>



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

上图就是主界面,主界面上方的英文就是简单的操作说明,操作方法我们在使用 Petalinux时已经介绍过,具体见第六章配置 petalinux 章节。

uboot 有许多配置项,可通过键盘上的上下键移动到配置项。后面跟着"--->"表示此配置项有子配置项,按回车键就可以进入子配置项。

我们以如何使能 dns 命令为例,讲解一下如何通过图形化界面来配置 uboot。进入 "Command line interface --->"这个配置项,此配置项用于配置 uboot 的命令,进入以后如下 图所示:



图 15.1.2 Command line interface 配置项

从上图可以看出,有很多配置项,这些配置项也有子配置项,选择"Network commands --->",进入网络相关命令配置项,如下图所示:

.config - U-Boot 2020.01 Configuration
Arrow keys navigate the menu. < Enter> selects submenus> (or empty
submenus). Highlighted letters are hotkeys. Pressing <y></y>
includes, <n> excludes, <m> modularizes features. Press <esc><esc> to</esc></esc></m></n>
exit, for Help, for Search. Legend: [*] built-in []
<u>(-)</u>
[] tftpsrv
[*] Control TFTP timeout and count through environment
[] rarpboot
[*] Mil
[] cdp
[] dns
<pre><select> < Exit > < Help > < Save > < Load ></select></pre>

图 15.1.3 Network commands 配置项



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

从上图可以看出,uboot 中有很多和网络有关的命令,比如 bootp、tftpboot、dhcp 等等。 选中 dns,然后按下键盘上的"Y"键,此时 dns 前面的"[]"变成了"[*]",如下图所示:

<pre>>Command line interface →Network commands Network commands Arrow keys navigate the menu. <enter> selects submenus> (or empty submenus). Highlighted letters are hotkeys. Pressing <y> includes, <n> excludes, <m> modularizes features. Press <esc> to exit, <?> for Help, for Search. Legend: [*] built-in [] (-) [*] Control TFTP timeout and count through environment [] rarpboot [*] nfs [*] mit [*] mit [*] modio -*- ping [*]</esc></m></n></y></enter></pre>
Arrow keys navigate the menu. <enter> selects submenus> (or empty submenus>). Highlighted letters are hotkeys. Pressing <y> includes, <n> excludes, <m> modularizes features. Press <esc> to exit, <? > for Help, for Search. Legend: [*] built-in [] (-) [*] Control TFTP timeout and count through environment [] rarpboot [*] nfs [*] mit [*] mit [*] modo -*- ping</esc></m></n></y></enter>
Arrow keys navigate the menu. <enter> selects submenus> (or empty submenus>). Highlighted letters are hotkeys. Pressing <y> includes, <n> excludes, <m> modularizes features. Press <esc><to exit, <? > for Help, for Search. Legend: [*] built-in [] (-) [*] Control TFTP timeout and count through environment [] rarpboot [*] nfs [*] mii [*] mdio -*- ping</to </esc></m></n></y></enter>
<pre>submenus). Highlighted letters are hotkeys. Pressing <y> includes, <n> excludes, <m> modularizes features. Press <esc><esc> to exit, <?> for Help, for Search. Legend: [*] built-in [] [*] Control TFTP timeout and count through environment [] rarpboot [*] nfs [*] mii [*] mdio -*- ping [*]</esc></esc></m></n></y></pre>
<pre>includes, <n> excludes, <m> modularizes features. Press <esc><esc> to exit, <?> for Help, for Search. Legend: [*] built-in [] [*] Control TFTP timeout and count through environment [] rarpboot [*] nfs [*] mii [*] mdio -*- ping [*]</esc></esc></m></n></pre>
exit, for Help, for Search. Legend: [*] built-in [] [*] Control TFTP timeout and count through environment [] rarpboot [*] nfs [*] mii [*] mdio -*- ping [*]
<pre>[*] Control TFTP timeout and count through environment [] rarpboot [*] nfs [*] mii [*] mdio -*- ping [*]</pre>
<pre>[] rarpboot [*] nfs [*] mii [*] mdio -*- ping [*]</pre>
[*] nfs [*] mi [*] mi [*] mdio -*- ping
[*] mii [*] mdio -*- ping
[*] mdio -*- ping
-*- ping
[] sntp
[] linklocal
Contraction of the second seco

图 15.1.4 选中 dns 命令

每个选项有 3 种编译选项,分别为:编译进 uboot、取消编译(也就是不编译这个功能模块)、编译为模块。按下"Y"键表示编译进 uboot,此时"[]"变成了"[*]";按下"N"表示不编译,"[]"默认表示不编译;有些功能模块是支持编译为模块的,这个一般在 Linux 内核里面很常用,uboot 下面不使用,如果要将某个功能编译为模块,那就按下"M",此时"[]"就会变为"<M>"。

选中 dns, 然后按下"H"或者"?"键可以打开 dns 命令的提示信息, 如下图所示:



图 15.1.5 dns 命令提示信息

按两下 ESC 键即可退出提示界面,相当于返回上一层。选择 dns 命令以后,按两下 ESC 键 (按两下 ESC 键相当于返回上一层),退出当前配置项,进入到上一层配置项。如果没有 要修改的就按两下 ESC 键,退出到主配界面,如果其他也没有要修改的,那就再次按两下



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

ESC 键退出 menuconfig 配置界面。如果修改过配置的话,在退出主界面的时候会有如下图所 示提示:



图 15.1.6 是否保存新的配置文件对话框

上图询问是否保存新的配置文件,通过键盘的←或→键来选择"Yes"或"No"项,此 处我们保持默认选择"Yes",然后按下键盘上的回车键确认保存。至此,我们就完成了通过 图形界面使能 uboot 的 dns 命令,打开.config 文件,会发现多了"CONFIG_BOOTP_DNS=y" 这一行,如下图中的601行所示:



图 15.1.7. config 文件

使用如下命令编译 uboot:

make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi--j8

编译完成以后下载到领航者开发板中运行,下载方式见 14.1.3。下载完成后输入"?"查 看是否有"dns"命令,一般肯定有的。测试一下 dns 命令工作是否正常(需要开发板连接路 由器),使用 dns 命令来查看一下百度官网"www.baidu.com"的 IP 地址。要先设置一下 dns 服务器的 IP 地址,也就是设置环境变量 dnsip 的值,命令如下:

```
setenv dnsip 114.114.114.114
```

saveenv

设置好以后就可以使用 dns 命令查看百度官网的 IP 地址了, 输入命令:

dns www.baidu.com

结果如下图所示:

```
Zyng> setenv dnsip 114.114.114.114
Zynq> dns www.baidu.com
180.101.49.12
Zynq>
```

图 15.1.8 dns 命令



原子哥在线教学: www.yuanzige.com 论:

论坛:www.openedv.com/forum.php

从上图可以看出, "www.baidu.com"的 IP 地址为 180.101.49.12, 说明 dns 命令工作正常。这个就是通过图形化命令来配置 uboot, 一般用来使能一些命令还是很方便的, 这样就不需要到处找命令的配置宏是什么, 然后再到配置文件里面去定义。

15.2 menuconfig 图形化配置原理

15.2.1 make menuconfig 过程分析

当输入"make menuconfig"以后会匹配到顶层 Makefile 的如下代码:

示例代码 顶层 Makefile 代码段

536 config: scripts_basic outputmakefile FORCE

537 \$(Q)\$(MAKE) \$(build)=scripts/kconfig \$@

这个在 12.2.13 小节已经详细的讲解过了,其中 build=-f./scripts/Makefile.build obj,将 537 行的规则展开就是:

@ make -f ./scripts/Makefile.build obj=scripts/kconfig menuconfig //可以没有@

Makefilke.build 会读取 scripts/kconfig/Makefile 中的内容,在 scripts/kconfig/Makefile 中可以找到如下代码:

示例代码 scripts/kconfig/Makefile 代码段

```
34 menuconfig: $(obj)/mconf
```

35 \$< \$(silent) \$(Kconfig)

其中 obj= scripts/kconfig, silent 是设置静默编译的, 在这里可以忽略不计, Kconfig=Kconfig, 因此扩展以后就是:

menuconfig: scripts/kconfig/mconf

scripts/kconfig/mconf Kconfig

目标 menuconfig 依赖 scripts/kconfig/mconf,因此 scripts/kconfig/mconf.c 这个文件会被编译,生成 mconf 这个可执行文件。目标 menuconfig 对应的规则为 scripts/kconfig/mconf Kconfig,也就是说 mconf 会调用 uboot 源码根目录下的 Kconfig 文件开始构建图形配置界面。

15.2.2 Kconfig 语法简介

上一小节我们已经知道了 scripts/kconfig/mconf 会调用 uboot 根目录下的 Kconfig 文件开始 构建图形化配置界面,接下来简单学习一下 Kconfig 的语法。因为后面学习 Linux 驱动开发的 时候可能会涉及到修改 Kconfig,对于 Kconfig 语法我们不需要太深入的去研究,关于 Kconfig 的详细语法介绍,可以参考 linux 内核源码(uboot 源码中没有这个文件)中的文件 Documentation/kbuild/kconfig-language.txt,本节我们大概了解其原理即可。打开 uboot 根目录 下的 Kconfig,这个 Kconfig 文件就是顶层 Kconfig,我们就以这个文件为例来简单学习一下 Kconfig 语法。

1, mainmenu

故名思议 mainmenu 就是主菜单,也就是输入"make menuconfig"以后打开的默认界面, 在顶层 Kconfig 中有如下代码:

示例代码顶层 Kconfig 代码段

6 mainmenu "U-Boot \$UBOOTVERSION Configuration"

示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php 上述代码就是定义了一个名为"U-Boot \$UBOOTVERSION Configuration"的主菜单,其 中 UBOOTVERSION=2020.01,因此主菜单名为"U-Boot 2020.01 Configuration",如下图所



图 15.2.1 主菜单名字

2、调用其他目录下的 Kconfig 文件

和 makefile 一样, Kconfig 也可以调用其他子目录中的 Kconfig 文件,调用方法如下: source "xxx/Kconfig" //xxx 为具体的目录名,相对路径

在顶层 Kconfig 中有如下代码:

示例代码 顶层 Kconfig 代码段 source 代码

```
13 source "arch/Kconfig"
.....
583 source "api/Kconfig"
584
585 source "common/Kconfig"
586
587 source "cmd/Kconfig"
588
589 source "disk/Kconfig"
590
591 source "dts/Kconfig"
592
593 source "env/Kconfig"
594
595 source "net/Kconfig"
596
597 source "drivers/Kconfig"
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

598

599 source "fs/Kconfig"600601 source "lib/Kconfig"

602

603 source "test/Kconfig"

从上述示例代码中可以看出,顶层 Kconfig 文件调用了很多其他子目录下的 Kcofig 文件,这些子目录下的 Kconfig 文件在主菜单中生成各自的菜单项。

3、menu/endmenu 条目

menu 用于生成菜单, endmenu 就是菜单结束标志, 这两个成对出现, 用于定义一个拥有 若干配置选项或子菜单的配置菜单。在顶层 Kconfig 中有如下代码:

```
示例代码 顶层 Kconfig 代码段 menu 代码
```

```
15
  menu "General setup"
16
17 config BROKEN
18
       bool
19
       help
20
          This option cannot be enabled. It is used as dependency
21
          for broken and incomplete features.
. . . . . .
304 endmenu
                # General setup
305
306 menu "Boot images"
308 config ANDROID BOOT IMAGE
309 bool "Enable support for Android Boot Images"
. . . . . .
                # Boot images
581 endmenu
```

上述示例代码中有两个 menu/endmenu 代码块,这两个代码块就是两个子菜单。第 15 行的 "menu "General setup""表示子菜单 "General setup"。第 306 行的 "menu "Boot images""表示子菜单 "Boot images"。体现在主菜单界面的效果如下图所示:



正点原子

图 15.2.2 子菜单

在"General setup"菜单上面还有"Architecture select (ARM architecture)"和"ARM architecture"这两个子菜单,但是在顶层 Kconfig 中并没有看到这两个子菜单对应的 menu/endmenu 代码块,那这两个子菜单是怎么来的呢?这两个子菜单就是 arch/Kconfig 文件 生成的。包括主界面中的"Boot timing"、"Console recording"等等这些子菜单,都是分别 由顶层 Kconfig 所调用的 common/Kconfig、cmd/Kconfig 等这些子 Kconfig 文件来创建的。

3、config 条目

顶层 Kconfig 中的"General setup"子菜单内容如下:

```
示例代码 顶层 Kconfig 代码段
15
    menu "General setup"
16
   config BROKEN
17
18
        bool
19
        help
20
          This option cannot be enabled. It is used as dependency
21
          for broken and incomplete features.
22
23
   config LOCALVERSION
24
        string "Local version - append to U-Boot release"
25
        help
. . . . . .
134 config SYS MALLOC F
         bool "Enable malloc() pool before relocation"
136
         default y if DM
137
         help
. . . . . .
```

```
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
   144 config SYS MALLOC F LEN
           hex "Size of malloc() pool before relocation"
   145
   146
          depends on SYS MALLOC F
          default 0x1000 if AM33XX
   147
          default 0x2800 if SANDBOX
   148
   149
           default 0x2000 if (ARCH IMX8 || ARCH IMX8M || ARCH MX7 || \backslash
              ARCH_MX7ULP || ARCH_MX6 || ARCH_MX5)
           default <mark>0</mark>x400
   150
   151
           help
   . . . . . .
   187 menuconfig EXPERT
   188
         bool "Configure standard U-Boot features (expert users)"
   189
         default y
   190
         help
   . . . . . .
   196 if EXPERT
         config SYS MALLOC CLEAR ON INIT
   197
         bool "Init with zeros the memory reserved for malloc (slow)"
   198
   199
         default y
   200 help
   . . . . . .
   221 endif # EXPERT
   . . . . . .
   304 endmenu # General setup
```

正点原子

可以看出,在 menu/endmenu 代码块中有大量的 "config xxxx" 的代码块,也就是 config 条目。config 条目就是 "General setup" 菜单的具体配置项,如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php .config - U-Boot 2020.01 Configuration General setup General setup Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty submenus ----). Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] built-in [] () Local version - append to U-Boot release [*] Automatically append version information to the version strin [*] Optimize for size [*] Select defaults suitable for booting general purpose Linux di -*- Add arch, board, vendor and soc variables to default environm (4) Number of DRAM banks [] Enable kernel command line setup [] Enable kernel board information setup [*] Enable malloc() pool before relocation (0x800) Size of malloc() pool before relocation <Select> < Exit > < Help > < Save > < Load >

图 15.2.3 General setup 配置项

可以看到"config LOCALVERSION"对应着第一个配置项, "config LOCALVERSION_AUTO"对应着第二个配置项,以此类推,至于为什么没有"config BROKEN"的配置项,后面会说明。我们以"config LOCALVERSION"和"config LOCALVERSION_AUTO"这两个为例来分析一下 config 配置项的语法:

示例代码 顶层 Kconfig 代码段

```
config LOCALVERSION
   31
          string "Local version - append to U-Boot release"
   32
          help
            Append an extra string to the end of your U-Boot version.
   . . . . . .
   38
      be a maximum of 64 characters.
   40 config LOCALVERSION AUTO
            bool "Automatically append version information to the version
   41
string"
   42
          default y
   43
          help
           This will try to automatically determine if the current tree is a
   44
   45
           release tree by looking for Git tags that belong to the current
           top of tree revision.
   46
        which is done within the script "scripts/setlocalversion".)
   58
   第 30 和 40 行,这两行都以"config"关键字开头,后面跟着"LOCALVERSION"和
"LOCALVERSION_AUTO", 这两个是配置项名字。 假如我们使能了
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

LOCALVERSION_AUTO 这个功能, 那么就会在.config 文件中生成 CONFIG_LOCALVERSION_AUTO,这个在上一小节讲解如何使能 dns 命令的时候讲过了。 由此可知,.config 文件中的 "CONFIG_xxx" (xxx 是具体的配置项名字)就是 Kconfig 文件中 config 关键字后面的配置项名字加上 "CONFIG_" 前缀。

config 关键字下面的几行是配置项属性,第 31~38 行是 LOCALVERSION 的属性,第 41~58 行是 LOCALVERSION_AUTO 的属性。属性里面描述了配置项的类型、输入提示、依 赖关系、帮助信息和默认值等。

第 31 行的 string 是变量类型,也就是"CONFIG_LOCALVERSION"的变量类型。变量 类型一共有5种,分别是: bool、tristate、string、hex和int,最常用的是 bool、tristate和 string 这三种。bool 类型有两种值: y和n,当为 y的时候表示使能该配置项,当为n的时候表示禁 止该配置项。tristate 类型有三种值: y、m和n,其中 y和n的涵义与 bool 类型一样,m表示 将这个配置项编译为模块。string 为字符串类型,所以LOCALVERSION 是个字符串变量,用 来存储本地字符串,选中以后即可输入用户定义的本地版本号,如下图所示:



图 15.2.4 本地版本号配置

可以在上图中输入本地版本号,此处我们只是演示下,不输入,直接按"OK"退出。

另外,可以看到"string"后面的"Local version - append to U-Boot release"对应该配置项 在图形界面上显示出来的标题。也就是说变量类型后面的输入提示会显示在图形配置界面上。 由于"config BROKEN"只有"boot"变量类型,没有对应输入提示,所以没有显示在图形 配置界面上。

第 32 行, help 表示帮助信息,告诉我们配置项的含义,当我们按下"h"或"?"弹出来的帮助界面就是 help 的内容。

第 41 行,说明 "CONFIG_LOCALVERSION_AUTO" 是个 bool 类型,可以通过按下 Y 或 N 键来使能或者禁止 CONFIG_LOCALVERSION_AUTO。

第 42 行, "default y" 表示 CONFIG_LOCALVERSION_AUTO 的默认值就是 y, 所以这 一行默认会被选中。

4、depends on 和 select

打开 arch/arm/Kconfig 文件,在里面有这如下代码:

示例代码 arch/arm/Kconfig 代码段

1 menu "ARM architecture"



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

2	depends on ARM
3	
4	config SYS_ARCH
5	default "arm"
6	
7	config ARM64
8	bool
9	select PHYS_64BIT
10	select SYS_CACHE_SHIFT_6

第 2 行的 "depends on" 说明 "ARM architecture" 项依赖于 "ARM",也就是说 "ARM" 被选中以后 "ARM architecture" 才会显示。

第 9~10 行的 "select" 表示反向依赖,当选中 "ARM64" 以后, "PHYS_64BIT"、 "SYS_CACHE_SHIFT_6" 这两个也会被选中。

4、 choice/endchoice

在 arch/Kconfig 文件中有如下代码:

```
示例代码 arch/ Kconfig 代码段
```

```
7 choice
8
     prompt "Architecture select"
9
     default SANDBOX
10
11 config ARC
     bool "ARC architecture"
12
.....
20 config ARM
     bool "ARM architecture"
21
.....
30 config MICROBLAZE
31 bool "MicroBlaze architecture"
•••••
1156
1157 endchoice
```

choice/endchoice 代码段用来定义一组可选择项,将多个类似的配置项组合在一起,供用 户单选或者多选。上述示例代码就是选择处理器架构,可以从 ARC、ARM、MICROBLAZE 等这些架构中选择,这里是单选。在 uboot 图形配置界面上选择 "Architecture select",进入 以后如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php .config - U-Boot 2020.01 Configuration Architecture select Use the arrow keys to navigate this window or press the hotkey of the item you wish to select followed by the <SPACE BAR>. Press <?> for additional information about this ARC architecture (X) ARM architecture) M68000 architecture) MicroBlaze architecture MIPS architecture NDS32 architecture <Select> < Help >

图 15.2.5 架构选择界面

可以在上图中通过移动光标来选择所使用的 CPU 架构。第 8 行的 prompt 给出这个 choice/endchoice 段的提示信息为"Architecture select"。

5, menuconfig

15

menuconfig 和 menu 很类似,但是 menuconfig 是个带选项的菜单,其一般用法为:

示例代码 menuconfig 用法

```
1 menuconfig MODULES
      bool "菜单"
2
3 if MODULES
4 . . .
5 endif # MODULES
```

第1行,定义了一个可选的菜单 MODULES,只有选中了 MODULES 后第 3~5 行的 if 到 endif之间的内容才会显示。在顶层 Kconfig 中有如下代码:

示例代码 顶层 Kconfig 代码段

```
menu "General setup"
.....
187 menuconfig EXPERT
188
      bool "Configure standard U-Boot features (expert users)"
189
      default y
190
      help
      This option allows certain base U-Boot options and settings
191
       to be disabled or tweaked. This is for specialized
192
       environments which can tolerate a "non-standard" U-Boot.
193
194
       Use this only if you really know what you are doing.
195
196 if EXPERT
```

领航者	ZYNQ 之嵌入式 Linux 开发指南 ② 正点原子		
原子哥在	送教学:www.yuanzige.com 论坛:www.openedv.com/forum.php		
197	config SYS_MALLOC_CLEAR_ON_INIT		
198	bool "Init with zeros the memory reserved for malloc (slow)"		
199	default y		
200	help		
201	This setting is enabled by default. The reserved malloc		
202	memory is initialized with zeros, so first malloc calls		
212			
213 config TOOLS_DEBUG			
214	bool "Enable debug information for tools"		
215	help		
216	Enable generation of debug information for tools such as mkimage.		
217	This can be used for debugging purposes. With debug information		
218	it is possible to set breakpoints on particular lines, single-step		
219	debug through the source code, etc.		
220			
221 e i	ndif # EXPERT		
304	endmenu # General setup		
第1	87~221 行使用 menuconfig 实现了一个菜单,路径如下:		

General setup

-> Configure standard U-Boot features (expert users) --->

如下图所示:

.config - U-Boot 2020.01 Configuration →Connection
General setup
Arrow keys navigate the menu. <enter> selects submenus> (or empty submenus). Highlighted letters are hotkeys. Pressing <y> includes, <n> excludes, <m> modularizes features. Press <esc><esc> to exit, <? > for Help, for Search. Legend: [*] built-in []</esc></esc></m></n></y></enter>
<pre>-*- Add arch, board, vendor and soc variables to default environm (4) Number of DRAM banks [] Enable kernel command line setup [] Enable kernel board information setup [*] Enable malloc() pool before relocation (0x800) Size of malloc() pool before relocation (0x1400000) Define memory for Dynamic allocation (0x800) Size of malloc() pool in SPL before relocation [*] Configure standard U-Boot features (expert users)></pre>
L [] 64bit physical address support
<pre><select> < Exit > < Help > < Save > < Load ></select></pre>

图 15.2.6 菜单 Configure standard U-Boot features (expert users)



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

从上图可以看到,前面有"[]"说明这个菜单是可选的,当选中这个菜单以后就可以进入到子选项中,也就是示例代码"menuconfig EXPERT"的第187~221行所描述的配置项,如下图所示:



图 15.2.7 EXPERT 子菜单

如果不使能"Configure standard U-Boot features (expert users)"菜单,那么示例代码"menuconfig EXPERT"的第 187~221 行所描述的配置项就不会显示出来,进去以后是空白的,不显示配置项。

6, comment

comment 用于注释,也就是在图形化界面中显示一行注释,打开文件 cmd/Kconfig,有如下所示代码:

示例代码 cmd/K config 代码段

```
186 config BUILD_BIN2C
```

187 bool

188

189 comment "Commands"

第 189 行使用 comment 标注了一行注释,注释内容为: "Commands",该行注释在配置项 Command line interface 的下面。在 uboot 图形配置界面上选择"Command line interface --->",进入后如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

.config - U-Boot 2020.01 Configuration →Command line interface ————————————————————————————————————
Command line interface Arrow keys navigate the menu. <enter> selects submenus> (or empty submenus). Highlighted letters are hotkeys. Pressing <y> includes, <n> excludes, <m> modularizes features. Press <esc><esc> to exit, <? > for Help, for Search. Legend: [*] built-in []</esc></esc></m></n></y></enter>
<pre>[*] Support U-Boot commands -*- Use hush shell -*- Enable command line editing -*- Enable auto complete using TAB -*- Enable long help messages (Zynq>) Shell prompt (y) Command execution tracer Autoboot options> *** Commands *** Info commands></pre>
<pre><select> < Exit > < Help > < Save > < Load ></select></pre>

图 15.2.8 注释 "Commands"

从上图可以看出,在配置项"Autoboot support"下面有一行注释,注释内容为"*** Commands ***"。

7, source

source 用于读取另一个 Kconfig, 比如:

source "arch/Kconfig"

这个在前面已经讲过了。

Kconfig 语法就讲解到这里,基本上常用的语法就是这些,因为 uboot 相比 Linux 内核要 小很多,所以配置项也要少很多,所以建议大家使用 uboot 来学习 Kconfig。一般无需修改 uboot 中的 Kconfig 文件,甚至都不会去使用 uboot 的图形化界面配置工具,本小节学习 Kconfig 的目的主要还是为了 Linux 内核作准备。

15.3 添加自定义菜单

图形化配置工具的主要工作是在.config 文件下面生成前缀为"CONFIG_"的变量,这些变量一般都有对应的值,为y、m、字符串或数字,在 uboot 源码里面会根据这些变量来决定编译哪些文件。本小节我们就来学习一下如何添加自定义菜单。添加的自定义菜单需求如下:

1) 在主界面中添加一个名为"My test menu"的菜单,此菜单内部有一个配置项。

2) 配置项为"MY_TESTCONFIG",此配置项处于菜单"My test menu"中。

3) 配置项的变量类型为 bool, 默认值为 y。

4) 配置项标题名(变量名)为"This is my test config"。

5) 配置项的帮助信息为"This is a empty config, just for tset!"。

打开顶层 Kconfig, 在最后面加入如下代码:

示例代码 自定义菜单

1	menu	"My	test menu"	
~				

3 config MY_TESTCONFIG

4 bool "This is my test config"



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

5 **default** y

- 6 help
- This is a empty config, just for tset! 7
- 8

9 endmenu # my test menu

添加完成以后打开图形化配置界面,如下图所示:



图 15.3.1 主界面

从上图可以看出, 主配置界面最后面出现了一个名为"My test menu"的菜单, 这个就是 我们添加进来的自定义菜单。进入此菜单,如下图所示:

config - U-Boot 2020.01 Configuration →My test menu		
My test menu Arrow keys navigate the menu. <enter> selects submenus> (or empty submenus). Highlighted letters are hotkeys. Pressing <y> includes, <n> excludes, <m> modularizes features. Press <esc><esc> to exit, <? > for Help, for Search. Legend: [*] built-in []</esc></esc></m></n></y></enter>		
[*] This is my test config (NEW)		
<pre><select> < Exit > < Help > < Save > < Load ></select></pre>		

图 15.3.2 "My test menu" 子菜单



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

从上图可以看出,配置项添加成功,选中"This is my test config"配置项,然后按下"H" 键打开帮助文档,如下图所示:



图 15.3.3 帮助信息

从上图可以看出,帮助信息也显示正确。配置项"MY_TESTCONFIG"默认也是被选中 的,因此在图形界面中选中"Save",然后按回车,最后退出图形化配置界面。重新编译一 下 uboot, 在.config 文件中会有 "CONFIG_MY_TESTCONFIG=y" 这一行, 如下图所示:

1604	#	
1605	# My test	menu
1606	#	
1607	CONFIG_MY	_TESTCONFIG=y

图 15.3.4. config 文件

至此,我们成功的在主菜单添加的自定义菜单。以后大家如果去半导体原厂工作的话, 如果要编写 Linux 驱动,那么很有可能需要你来修改甚至编写 Kconfig 文件。Kconfig 语法其 实不难,重要的点就是15.2.2小节中的那几个,最主要的是记住: Kconfig 文件的最终目的是 在.config 文件中生成以"CONFIG_"开头的变量。

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

正点原子

第十六章 Linux 内核顶层 Makefile 详解

前几章我们重点讲解了如何移植 uboot 到领航者开发板上,从本章开始我们就开始学习如 何移植 Linux 内核。同 uboot 一样,在具体移植之前,我们先来学习一下 Linux 内核的顶层 Makefile 文件,因为项层 Makefile 控制着 Linux 内核的编译流程。



论坛:www.openedv.com/forum.php

正点原子

16.1 Linux 内核获取

关于 Linux 的起源以及发展历史,这里就不啰嗦了,网上相关的介绍太多了。即使写到 这里也只是水一下教程页数而已,没有任何实际的意义。有限的时间还是放到有意义的事情 上吧,Linux 由 Linux 基金会管理与发布,Linux 官网为 <u>https://www.kernel.org</u>,所以你想获取 最新的 Linux 版本就可以在这个网站上下载,网站界面如下图所示:



图 16.1.1 Linux 官网

从上图可以看出最新的稳定版 Linux 已经到了 6.4.9,大家没必要追新,因为 5.x 版本的 Linux 和 6.x 版本没有本质上的区别, 6.x 更多的是加入了一些新的平台、新的外设驱动而已。 Xilinx 会从 https://www.kernel.org 下载某个版本的 Linux 内核,然后将其移植到自己的芯 片平台上,测试成功以后就会将其开放给 Xilinx 的芯片平台开发者。开发者下载 Xilinx 提供 的 Linux 内核,下载地址为 <u>https://github.com/Xilinx/linux-xlnx/tags</u>。后面的移植我们使用 Xili nx 提供的 Linux 源码, Xilinx 提供的 Linux 源码已经放到了开发板光盘中,路径为: ZYNQ 开 发板资料盘(A 盘)\4_SourceCode\3_Embedded_Linux\资源文件\kernel\linux-xlnx-xlnx_rebase_5.4 _2020.2.tar.gz。

16.2 Linux 工程目录分析

这里我们使用正点原子移植好的 Linux 源码。移植好的 Linux 源码已经放到了开发板光盘 中,路径为: ZYNQ开发板资料盘(A 盘)\4_SourceCode\3_Embedded_Linux\资源文件\出场镜像 相关 \atk-zynq-linux-xlnx.tar.gz。可以在 Ubuntu 中新建一个名为"atk-zynq"的文件夹,然后 将 atk-zynq-linux-xlnx.tar.gz 这个压缩包拷贝到新建的 atk-zynq 文件夹中并解压。查看内核目 录结构如下图所示:



S

原子哥在线教学:	www.yuanzige	e.com 论坛:www	.openedv.c	om/forum.php
wmq@Linux	:~/petalinux/a	tk-zynq/atk-zynq-linu	x-xlnx\$ ls	
arch	drivers	kernel	net	usr
block	fs	lib	README	virt
certs	include	LICENSES	samples	zynq-fit-image.it
COPYING	init	MAINTAINERS	scripts	
CREDITS	ipc	Makefile	security	
crypto	Kbuild	ጦጦ	sound	
Documenta	<mark>tion</mark> Kconfig	<pre>mpsoc-fit-image.its</pre>	tools	
wmq@Linux	:~/petalinux/a	tk-zynq/atk-zynq-linu	x-xlnx\$	

图 16.2.1 未编译的 Linux 源码目录

上图中重要的文件夹或文件的含义如下表所示:

表 16.2.1 Linux 目录

类型	名字	描述	备注	
	arch	架构相关目录		
	block	块设备相关目录		
	crypto	加密相关目录		
	Documentation	文档相关目录		
	drivers	驱动相关目录		
	firmeare	固件相关目录		
	fs	文件系统相关目录		
	include	头文件相关目录		
	init 初始化相关目录			
	ipc	进程间通信相关目录		
	kernel	内核相关目录	Linux 自带	
又忤夹	lib	库相关目录		
	mm	内存管理相关目录		
	net	网络相关目录		
	samples	例程相关目录		
	scripts	脚本相关目录		
	security	安全相关目录		
	sound	音频处理相关目录		
	tools	工具相关目录		
	usr	与 initramfs 相关的目录, 用于生成		
		initramfs		
	virt	提供虚拟机技术(KVM)		
	.config	Linux 最终使用的配置文件	编译生成的文件	
	.gitignore	git 工具相关文件	
	.mailmap	邮件列表	- Linux 自带	
	.tmp_xx	这是一系列的文件,作用目前笔者 还不是很清禁	编译生成的文件	
	version	—————————————————————————————————————		
	vmlinux cmd	emd 文件。田子连接生成 ymlinuy	编译生成的文件 Linux 自带	
	COPVING	いれて、 いれて、 い ゴーン に が ゴーン と は 工 成 い minux い な し 、 の の 、 の 、 の 、 の 、 の 、 の 、 の 、 の 、 の 、 の 、 の 、 の 、 の 、 の 、 の 、 の 、 の の 、 の 、 の 、 の 、 の 、 の 、 の 、 の 、 の 、 の 、 の 、 、 の 、 の 、 の 、 の 、 、 、 の 、 、 の 、 の 、 の 、 の の の 、 の 、 の の の 、 の の の の の の の の の の の の の		
	CREDITS	Linux 贡献老		
	Khuild	Makefile 会遗取业文件		
文件	Koonfig	图形化配置界面的配置文件	-	
211	MAINTAINERS		Linux 自带	
	Maharah	モリ 石 石 平 Linux 顶目 Makefile	编译生成的文件	
	Module xx	Makefile Linux 坝层 Makefile		
	modules.xx	一系列文件,和模块有关		
	zvng.sh	正点原子提供的, Linux 编译脚本	正点原子提供	
	README	Linux 描述文件	Linux 自带	
	REPORTING-BUGS	BUG上报指南	Linux 自带	
	System.map	符号表	编译生成的文件	
	venlinuv	编译出来的、未压缩的 ELF 格式	编译生成的文件	
	vmiinux	Linux 文件	狮 年生 成 的 义 件	



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php vmlinux.o 如果你们的问题,你们的问题,你们的问题。

上表中的很多文件夹和文件我们都不需要去关心,我们要关注的文件夹或文件如下:

1、arch 目录

这个目录是和架构有关的目录,比如 arm、arm64、avr32、x86 等等架构。每种架构都对 应一个目录,在这些目录中又有很多子目录,比如 boot、common、configs 等等,以 arch/arm 为例,其子目录如下图所示:

wmq@Linux:~/petalinux/atk	-zynq/atk-zynq-linux-xlnx\$	ls -d arch/arm/*
arch/arm/boot	arch/arm/mach-imx	arch/arm/mach-sa1100
arch/arm/common	arch/arm/mach-integrator	arch/arm/mach-shmobile
arch/arm/configs	arch/arm/mach-iop32x	arch/arm/mach-socfpga
arch/arm/crypto	arch/arm/mach-ixp4xx	arch/arm/mach-spear
arch/arm/include	arch/arm/mach-keystone	arch/arm/mach-sti
arch/arm/Kconfig	arch/arm/mach-lpc18xx	arch/arm/mach-stm32
arch/arm/Kconfig.debug	arch/arm/mach-lpc32xx	arch/arm/mach-sunxi
arch/arm/Kconfig-nommu	arch/arm/mach-mediatek	arch/arm/mach-tango
arch/arm/kernel	arch/arm/mach-meson	arch/arm/mach-tegra
arch/arm/kvm	arch/arm/mach-milbeaut	arch/arm/mach-u300
arch/arm/lib	arch/arm/mach-mmp	arch/arm/mach-uniphier
arch/arm/mach-actions	arch/arm/mach-moxart	arch/arm/mach-ux500
arch/arm/mach-alpine	arch/arm/mach-mv78xx0	arch/arm/mach-versatile
arch/arm/mach-artpec	arch/arm/mach-mvebu	arch/arm/mach-vexpress
arch/arm/mach-asm9260	arch/arm/mach-mxs	arch/arm/mach-vt8500

图 16.2.2 arch/arm 子目录

上图是 arch/arm 的一部分子目录,这些子目录用于控制系统引导、系统调用、动态调频、 主频设置等。arch/arm/configs 目录是不同平台的默认配置文件: xxx_defconfig,如下图所示:

wmq@Linux:~/petalinux/atk·	-zynq/atk-zynq-linux-xlnx	ls arch/arm/configs/
am200epdkit_defconfig	imx_v6_v7_defconfig	pxa255-idp_defconfig
aspeed_g4_defconfig	integrator_defconfig	pxa3xx_defconfig
aspeed_g5_defconfig	iop32x_defconfig	pxa910_defconfig
assabet_defconfig	ixp4xx_defconfig	pxa_defconfig
at91_dt_defconfig	jornada720_defconfig	qcom_defconfig
axm55xx_defconfig	keystone_defconfig	realview_defconfig
badge4_defconfig	lart_defconfig	rpc_defconfig
bcm2835_defconfig	lpc18xx_defconfig	s3c2410_defconfig
cerfcube_defconfig	lpc32xx_defconfig	s3c6400_defconfig
clps711x_defconfig	lpd270_defconfig	s5pv210_defconfig
cm_x2xx_defconfig	lubbock_defconfig	sama5_defconfig
cm_x300_defconfig	magician_defconfig	shannon_defconfig

图 16.2.3 配置文件

在 arch/arm/configs 中就包含有领航者开发板的默认配置文件: xilinx_zynq_defconfig,执行 "make xilinx_zynq_defconfig"即可完成配置。arch/arm/boot/dts 目录里面是对应开发平台的 设备树文件,正点原子领航者开发板对应的设备树文件如下所示:

领航者 7010 开发板对应的设备树文件为: arch/arm/boot/dts/atk-navigator-7010.dts

领航者 7020 开发板对应的设备树文件为: arch/arm/boot/dts/atk-navigator-7020.dts

arch/arm/boot 目录中有编译出来的 Image 和 zImage 镜像文件,而 zImage 就是我们要用的 linux 镜像文件。

arch/arm/mach-xxx 目录分别为相应平台的驱动和初始化文件,比如 mach-zynq 目录里面 就是 ZYNQ 系列 CPU 的驱动和初始化文件。

2、block 目录



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

block 是 Linux 下块设备目录,像 SD 卡、EMMC、NAND、硬盘等存储设备就属于块设备, block 目录中存放着管理块设备的相关文件。

3、crypto 目录

crypto 目录里面存放着加密文件,比如常见的 crc、crc32、md4、md5、hash 等加密算法。

4、Documentation 目录

此目录里面存放着 Linux 相关的文档,如果要想了解 Linux 某个功能模块或驱动架构的功能,就可以在 Documentation 目录中查找有没有对应的文档。

5、drivers 目录

设备驱动程序目录,此目录根据驱动类型的不同,分门别类进行整理,比如 drivers/i2c 就 是 I2C 相关驱动目录,drivers/gpio 就是 GPIO 相关的驱动目录,这是我们学习的重点。

6、firmware 目录

此目录用于存放固件。

7、fs 目录

此目录存放文件系统相关代码,比如fs/ext2、fs/ext4、fs/f2fs等,分别是ext2、ext4和f2fs 等文件系统。

8、include 目录

头文件目录。

9、init 目录

此目录存放 Linux 内核启动的时候初始化代码。

10、ipc 目录

IPC 为进程间通信, ipc 目录是进程间通信的具体实现代码。

11、kernel 目录

Linux 内核代码。与平台相关的部分代码放在 arch/*/kernel 目录下,其中*代表各种处理器 平台

12、lib 目录

lib 是库的意思, lib 目录都是一些公用的库函数。

13、mm 目录

此目录存放与平台无关的内存管理代码,与平台相关的内存管理代码放在 arch/*/mm 目录下。

14、net 目录

此目录存放网络相关代码。

15、samples 目录

此目录存放一些示例代码文件。

16、scripts 目录



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

脚本目录,Linux编译的时候会用到很多脚本文件,这些脚本文件就保存在此目录中。

17、security 目录

此目录存放安全相关的文件。

18、sound 目录

此目录存放音频相关驱动文件, 音频驱动文件并没有存放到 drivers 目录中, 而是单独的目录。

19、tools 目录

此目录存放一些编译的时候使用到的工具。

20、usr 目录

此目录存放与 initramfs 有关的代码。

21、virt 目录

此目录存放虚拟机相关文件。

22、.config 文件

跟 uboot 一样,.config 保存着 Linux 最终的配置信息,编译 Linux 的时候会读取此文件中的配置信息。最终根据配置信息来选择编译 Linux 哪些模块,哪些功能。

23、Kbuild 文件

有些 Makefile 会读取此文件。

24、Kconfig 文件

图形化配置界面的配置文件。

25、Makefile 文件

Linux 顶层 Makefile 文件,建议好好阅读一下此文件。

26、README 文件

此文件详细讲解了如何编译 Linux 源码,以及 Linux 源码的目录信息,建议仔细阅读一下 此文件。

关于 Linux 源码目录就分析到这里。

16.3 Linux 内核初次编译

现在我们讲解一下如何编译出对应的 Linux 镜像文件。新建名为"zynq.sh"的 shell 脚本, 然后在这个 shell 脚本里面输入如下所示内容:

示例代码 zynq.sh 文件内容

1 #!/bin/sh

2 make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi- distclean

3 make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi- xilinx_zynq_defconfig

4 make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi- menuconfig

5 make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi- all -j8



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

第 2 行,执行"make distclean",清理工程,所以 zynq.sh 每次都会清理一下工程。如果 通过图形界面配置了 Linux,但是还没保存新的配置文件,那么就要慎重使用 zynq.sh 编译脚 本了,因为它会把你的配置信息都删除掉。

第3行,执行"make xxx_defconfig",配置工程。

第 4 行,执行"make menuconfig",打开图形配置界面,对 Linux 进行配置,如果不想每次编译都打开图形配置界面的话可以将这一行删除掉。

第5行,执行"make",编译Linux源码。

可以看出, Linux 的编译过程基本和 uboot 一样,都要先执行"make xxx_defconfig"来配置一下,然后在执行"make"进行编译。如果需要使用图形界面配置的话就执行"make menuconfig"。

使用 chmod 给予 zynq.sh 可执行权限, 然后运行此 shell 脚本, 命令如下:

./zynq.sh

编译的时候会弹出 Linux 图形配置界面,如下图所示:

.config - Linux/arm 5.4.0 Kernel Configuration



图 16.3.1 Linux 图形配置界面

Linux 的图行界面配置和 uboot 是一样的,这里我们不需要做任何的配置,直接按两下 ESC 键退出,退出图形界面以后会自动开始编译 Linux。等待编译完成,完成以后如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

	SHIPPED	arch/arm/boot/compressed/hyp-stub.S
	сс	arch/arm/boot/compressed/decompress.o
	сс	arch/arm/boot/compressed/string.o
	SHIPPED	arch/arm/boot/compressed/lib1funcs.S
	SHIPPED	arch/arm/boot/compressed/ashldi3.S
	SHIPPED	arch/arm/boot/compressed/bswapsdi2.S
	AS	arch/arm/boot/compressed/hyp-stub.o
	AS	arch/arm/boot/compressed/lib1funcs.o
	AS	arch/arm/boot/compressed/ashldi3.o
	AS	arch/arm/boot/compressed/bswapsdi2.o
	AS	arch/arm/boot/compressed/piggy.o
	LD	arch/arm/boot/compressed/vmlinux
	OBJCOPY	arch/arm/boot/zImage
	Kernel:	arch/arm/boot/zImage is ready
WI	nq@Linux:	~/petalinux/atk-zynq/atk-zynq-linux-xlnx\$

图 16.3.2 Linux 编译完成

编译完成以后就会在 arch/arm/boot 这个目录下生成一个叫做 zImage 的文件, zImage 就是 我们要用的 Linux 镜像文件。另外也会在 arch/arm/boot/dts 下生成很多.dtb 文件,这些.dtb 就是 设备树文件,其中 atk-navigator 开头的就是领航者开发板对应的设备树文件。

16.4 VSCode 工程创建

在分析 Linux 的顶层 Makefile 之前,先创建 VSCode 工程,创建过程和 uboot 一样。创建 好以后将文件.vscode/settings.json 改为如下所示内容:

示例代码 settings.json 文件内容

```
1 {
2
    "search.exclude": {
   "**/node_modules": true,
3
    "**/bower_components": true,
4
5
    "**/*.o":true,
   "**/*.su":true.
6
   "**/*.cmd":true,
7
    "Documentation":true,
8
9
    /* 屏蔽不用的架构相关的文件 */
10
     "arch/alpha":true,
11
    "arch/arc":true.
12
13
    "arch/arm64":true,
    "arch/avr32":true,
14
     "arch/[b-z]*":true,
15
```

- 16 "arch/arm/plat*":true,
- 17 "arch/arm/mach-[a-y]*":true,
- 18 "arch/arm/mach-zx":true,

19

20 /* 屏蔽不用的配置文件 */

21 "arch/arm/configs/[a-w]*":true,



```
原子哥在线教学: www.yuanzige.com
                                               论坛:www.openedv.com/forum.php
    22
         "arch/arm/configs/[y-z]*":true,
    23
    24
        /* 屏蔽不用的 DTB 文件 */
         "arch/arm/boot/dts/[a-y]*":true,
    25
         "arch/arm/boot/dts/.*":true,
    26
    27
         },
    28
    29
         "files.exclude": {
         "**/.git": true,
    30
         "**/.svn": true,
    31
    32
         "**/.hg": true,
         "**/CVS": true,
    33
         "**/.DS_Store": true,
    34
    35
         "**/*.0":true,
         "**/*.su":true.
    36
         "**/*.cmd":true,
    37
         "Documentation":true,
    38
    39
        /* 屏蔽不用的架构相关的文件 */
    40
    41
         "arch/alpha":true,
         "arch/arc":true,
    42
    43
         "arch/arm64":true,
    44
         "arch/avr32":true,
    45
         "arch/[b-z]*":true,
    46
         "arch/arm/plat*":true,
    47
         "arch/arm/mach-[a-y]*":true,
         "arch/arm/mach-zx":true,
    48
    49
            /* 屏蔽不用的配置文件 */
    50
    51
         "arch/arm/configs/[a-w]*":true,
    52
         "arch/arm/configs/[y-z]*":true,
    53
        /* 屏蔽不用的 DTB 文件 */
    54
         "arch/arm/boot/dts/[a-y]*":true,
    55
         "arch/arm/boot/dts/.*":true,
    56
    57
         }
```

58 }

创建好 VSCode 工程以后就可以开始分析 Linux 的顶层 Makefile 了。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

16.5 顶层 Makefile 详解

Linux 的顶层 Makefile 和 uboot 的顶层 Makefile 非常相似,因为 uboot 参考了 Linux,前 610 行几乎一样,所以前面部分我们大致看一下就行了。

1、版本号

顶层 Makefile 一开始就是 Linux 内核的版本号,如下所示:

示例代码 顶层 Makefile 代码段

1 # SPDX-License-Identifier: GPL-2.0

```
2 VERSION = 5
```

3 PATCHLEVEL = 4

4 SUBLEVEL = 0

5 EXTRAVERSION =

可以看出, Linux 内核版本号为 5.4.0。

2、MAKEFLAGS 变量

MAKEFLAGS 变量设置如下所示:

示例代码 顶层 Makefile 代码段

```
38 MAKEFLAGS += -rR
```

3、命令输出

Linux 编译的时候也可以通过"V=1"来输出完整的命令,这个和 uboot 一样,相关代码 如下所示:

示例代码 顶层 Makefile 代码段

```
75 ifeq ("$(origin V)", "command line")
76 KBUILD_VERBOSE = $(V)
77 endif
78 ifndef KBUILD_VERBOSE
79 KBUILD_VERBOSE = 0
80 endif
81
82 ifeq ($(KBUILD_VERBOSE),1)
83 quiet =
84 Q =
85 else
86 quiet=quiet_
87 Q = @
88 endif
```

4、静默输出

Linux 编译的时候使用"make-s"就可实现静默编译,编译的时候就不会打印任何的信息,同 uboot 一样,相关代码如下:

示例代码 顶层 Makefile 代码段

90 # If the user is running make -s (silent mode), suppress echoing of



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

```
91 # commands
92
```

93 ifneq (\$(findstring s,\$(filter-out --%,\$(MAKEFLAGS))),)

```
94 quiet=silent_
```

95 endif

96

97 export quiet Q KBUILD_VERBOSE

5、设置编译结果输出目录

Linux 编译的时候使用 "O=xxx" 即可将编译产生的过程文件输出到指定的目录中,相关 代码如下:

示例代码 顶层 Makefile 代码段

```
119 # Do we want to locate output files in a separate directory?
120 ifeq ("$(origin 0)", "command line")
121 KBUILD_OUTPUT := $(0)
122 endif
```

6、代码检查

Linux 也支持代码检查,使用命令"make C=1"使能代码检查,检查那些需要重新编译的文件。"make C=2"用于检查所有的源码文件,顶层 Makefile 中的代码如下:

示例代码 顶层 Makefile 代码段

```
202 ifeq ("$(origin C)", "command line")
203 KBUILD_CHECKSRC = $(C)
204 endif
205 ifndef KBUILD_CHECKSRC
206 KBUILD_CHECKSRC = 0
207 endif
```

7、模块编译

Linux 允许单独编译某个模块,使用命令"make M=dir"即可,顶层 Makefile 中的代码如下:

示例代码 顶层 Makefile 代码段

```
209 # Use make M=dir or set the environment variable KBUILD_EXTMOD to specify the
210 # directory of external module to build. Setting M= takes precedence.
211 ifeq ("$(origin M)", "command line")
212 KBUILD_EXTMOD := $(M)
213 endif
214
215 export KBUILD_CHECKSRC KBUILD_EXTMOD
216
217 extmod-prefix = $(if $(KBUILD_EXTMOD),$(KBUILD_EXTMOD)/)
218
```

```
219 ifeq ($(abs_srctree),$(abs_objtree))
```


原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

```
220
         # building in the source tree
221
         srctree := .
222
      building_out_of_srctree :=
223 else
224
         ifeq ($(abs_srctree)/,$(dir $(abs_objtree)))
225
              # building in a subdirectory of the source tree
226
              srctree := ..
227
         else
228
              srctree := $(abs_srctree)
229
         endif
      building_out_of_srctree := 1
230
231 endif
232
233 ifneq ($(KBUILD_ABS_SRCTREE),)
234 srctree := $(abs_srctree)
235 endif
236
237 objtree := .
238 VPATH := $(srctree)
239
240 export building_out_of_srctree srctree objtree VPATH
```

外部模块编译过程和 uboot 也一样,最终导出 srctree、objtree 和 VPATH 这三个变量的值, 其中 srctree=.,也就是当前目录,objtree 同样为"."。

8、设置目标架构和交叉编译器

有时为了方便,可以直接修改项层 Makefile 中的 ARCH 和 CROSS_COMPILE,将其设置为对应的架构和编译器,比如本教程可以将 ARCH 设置为 arm, CROSS_COMPILE 设置为 arm-xilinx-linux-gnueabi-,如下所示:

示例代码 顶层 Makefile 代码段

 365 ARCH ?= arm

 366 CROSS_COMPILE ?= arm-xilinx-linux-gnueabi

 设置好以后就可以使用如下命令编译 Linux 了:

 make xxx_defconfig
 //使用默认配置文件配置 Linux

 make menuconfig
 //启动图形化配置界面

 make -j8
 //编译 Linux

9、调用 scripts/Kbuild.include 文件

同 uboot 一样, Linux 顶层 Makefile 也会调用文件 scripts/Kbuild.include, 顶层 Makefile 相 应代码如下:

示例代码 顶层 Makefile 代码段

333 include scripts/Kbuild.include

10、交叉编译工具变量设置



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

顶层 Makefile 中和交叉编译器有关的变量设置如下:

示例代码 顶层 Makefile 代码段

407 # Make variables (CC, etc...)

408 AS = \$(CROSS_COMPILE)as

409 LD = \$(CROSS_COMPILE)ld

410 CC = \$(CROSS_COMPILE)gcc

411 CPP = (CC) - E

412 AR = \$(CROSS_COMPILE)ar

413 NM = \$(CROSS_COMPILE)nm

414 STRIP = \$(CROSS_COMPILE)strip

415 OBJCOPY = \$(CROSS_COMPILE)objcopy

416 OBJDUMP = \$(CROSS_COMPILE)objdump

AS、LD、CC等这些都是交叉编译器所使用的工具。

11、头文件路径变量

顶层 Makefile 定义了两个变量保存头文件路径: USERINCLUDE 和 LINUXINCLUDE, 相关代码如下:

示例代码 顶层 Makefile 代码段

441 # Use USERINCLUDE when you must reference the UAPI directories only.

442 USERINCLUDE := \setminus

```
443 -I\(srctree)/arch/\(SRCARCH)/include/uapi \)
```

445 -I $(srctree)/include/uapi \setminus$

```
446 -I$(objtree)/include/generated/uapi \
```

```
447 -include $(srctree)/include/linux/kconfig.h
```

```
448
```

449 # Use LINUXINCLUDE when you must reference the include/ directory.

450# Needed to be compatible with the O= option

451 LINUXINCLUDE := \

452 $-I\(srctree)/arch/\(srcARCH)/include \)$

```
453 -I$(objtree)/arch/$(SRCARCH)/include/generated \
```

454 \$(if \$(building_out_of_srctree),-I\$(srctree)/include) \

```
455 -I$(objtree)/include \setminus
```

```
456 $(USERINCLUDE)
```

第 441~448 行的 USERINCLUDE 是 UAPI 相关的头文件路径,第 449~456 行的 LINUXINCLUDE 是 Linux 内核源码的头文件路径。重点来看下 LINUXINCLUDE,其中 srctree=., objtree=., SRCARCH=arm, hdr-arch=arm, KBUILD_SRC 为空,因此,将 USERINCLUDE 和 LINUXINCLUDE 展开以后为:

USERINCLUDE := \setminus

-I./arch/arm/include/uapi \
-I./arch/arm/include/generated/uapi \
-I./include/uapi \



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

-I./include/generated/uapi \ -include ./include/linux/kconfig.h

LINUXINCLUDE := \setminus

-I./arch/arm/include \

-I./arch/arm/include/generated $\$

-I./include $\$

-I./arch/arm/include/uapi \

-I./arch/arm/include/generated/uapi $\$

-I./include/uapi \

-I./include/generated/uapi \

-include ./include/linux/kconfig.h

12、导出变量

顶层 Makefile 会导出很多变量给子 Makefile 使用,导出的这些变量如下:

示例代码 顶层 Makefile 代码段

475 export ARCH SRCARCH CONFIG_SHELL BASH HOSTCC KBUILD_HOSTCFLAGS CROSS_COM PILE AS LD CC

476 export CPP AR NM STRIP OBJCOPY OBJDUMP OBJSIZE PAHOLE LEX YACC AWK INSTALLKE RNEL

477 export PERL PYTHON PYTHON2 PYTHON3 CHECK CHECKFLAGS MAKE UTS_MACHINE HOS TCXX

478 export KBUILD_HOSTCXXFLAGS KBUILD_HOSTLDFLAGS KBUILD_HOSTLDLIBS LDFLAGS_ MODULE

479

480 export KBUILD_CPPFLAGS NOSTDINC_FLAGS LINUXINCLUDE OBJCOPYFLAGS KBUILD_LD FLAGS

481 export KBUILD_CFLAGS CFLAGS_KERNEL CFLAGS_MODULE

482 export CFLAGS_KASAN CFLAGS_KASAN_NOSANITIZE CFLAGS_UBSAN

483 export KBUILD_AFLAGS AFLAGS_KERNEL AFLAGS_MODULE

484 export KBUILD_AFLAGS_MODULE KBUILD_CFLAGS_MODULE KBUILD_LDFLAGS_MODULE

485 export KBUILD_AFLAGS_KERNEL KBUILD_CFLAGS_KERNEL

16.5.1 make xxx_defconfig 过程

第一次编译 Linux 之前都要先使用"make xxx_defconfig"配置 Linux 内核,在顶层 Makefile 中有"%config"这个目标,如下所示:

示例代码 16.5.1 顶层 Makefile 代码段

262 config-build := 263 mixed-build := 264 need-config := 1 265 may-sync-config := 1 266 single-build :=



```
原子哥在线教学: www.yuanzige.com
                                                论坛:www.openedv.com/forum.php
    267
    268 ifneq ($(filter $(no-dot-config-targets), $(MAKECMDGOALS)),)
          ifeq ($(filter-out $(no-dot-config-targets), $(MAKECMDGOALS)),)
    269
    270
            need-config :=
    271
          endif
    272 endif
    273
    274 ifneq ($(filter $(no-sync-config-targets), $(MAKECMDGOALS)),)
    275
          ifeq ($(filter-out $(no-sync-config-targets), $(MAKECMDGOALS)),)
    276
            may-sync-config :=
    277
          endif
    278 endif
    279
    280 ifneq ($(KBUILD_EXTMOD),)
    281
          may-sync-config :=
    282 endif
    283
    284 ifeq ($(KBUILD_EXTMOD),)
    285
            ifneq ($(filter config %config,$(MAKECMDGOALS)),)
    286
            config-build := 1
    287
                 ifneq ($(words $(MAKECMDGOALS)),1)
    288
               mixed-build := 1
    289
                 endif
    290
            endif
    291 endif
    292
    293 # We cannot build single targets and the others at the same time
    294 ifneq ($(filter $(single-targets), $(MAKECMDGOALS)),)
    295
          single-build := 1
    296
          ifneq ($(filter-out $(single-targets), $(MAKECMDGOALS)),)
    297
            mixed-build := 1
    298
          endif
    299 endif
    300
    301 # For "make -j clean all", "make -j mrproper defconfig all", etc.
    302 ifneq ($(filter $(clean-targets),$(MAKECMDGOALS)),)
    303
            ifneq ($(filter-out $(clean-targets),$(MAKECMDGOALS)),)
    304
            mixed-build := 1
    305
            endif
    306 endif
    307
```



```
原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php
    308 # install and modules_install need also be processed one by one
   309 ifneq ($(filter install,$(MAKECMDGOALS)),)
           ifneq ($(filter modules_install,$(MAKECMDGOALS)),)
   310
   311
           mixed-build := 1
           endif
   312
   313 endif
   314
   315 ifdef mixed-build
   316 #=====
    . . .
    554# ====
   555 # *config targets only - make sure prerequisites are updated, and descend
   556 # in scripts/kconfig to make the *config target
   557
   558 # Read arch specific Makefile to set KBUILD_DEFCONFIG as needed.
   559 # KBUILD_DEFCONFIG may point out an alternative default configuration
   560 # used for 'make defconfig'
   561 include arch/$(SRCARCH)/Makefile
   562 export KBUILD_DEFCONFIG KBUILD_KCONFIG CC_VERSION_TEXT
   563
   564 config: outputmakefile scripts_basic FORCE
   565
         $(Q)$(MAKE) $(build)=scripts/kconfig $@
   566
   567 % config: outputmakefile scripts_basic FORCE
         $(Q)$(MAKE) $(build)=scripts/kconfig $@
   568
   550
       else
    . . . . . .
   627 endif # KBUILD EXTMOD
    第 262~313 行和 uboot 一样, 都是设置定义变量 config-build、mixed-build 和 single-build
的值,最终这三个变量的值为:
   config-build=1
   mixed-build=0
   need-config= 1
    因为 config-build=1,因此第 561 行~568 行成立。第 561 行引用 arch/arm/Makefile 这个文
件,这个文件很重要,以为 zImage、uImage 等这些文件就是由 arch/arm/Makefile 来生成的。
    第562行导出变量 KBUILD_DEFCONFIG KBUILD_KCONFIG CC_VERSION_TEXT。
    第564行,没有目标与之匹配,因此不执行。
```

第 567 行, "make xxx_defconfig"与目标"%config"匹配,因此被执行。"%config" 依赖 scripts_basic、outputmakefile 和 FORCE,真正有意义的依赖是 scripts_basic。scripts_basic 的规则如下:

示例代码 16.5.2 顶层 Makefile 代码段

原于	·哥在线教学:	www.yuan	zige.com	论坛:www	w.openedv.c	om/forum.p	hp	
	500 scripts_	basic:						
	501 \$ (Q) \$	(MAKE) \$(1	build)=scrip	pts/basic				
	502 \$ (Q) r	m -f .tmp	_quiet_reco	rdmcount				
obj,	build 定义在 因此将上述	文件 scripts/ 示例代码展3	Kbuild.include 千就是:	中,值为	build := -f S	S(srctree)/scr	ipts/Makefile.	build
	scripts_basic:							
	@make -f ./	scripts/Makefi	ile.build obj=scri	pts/basic /	/也可以没有(@,视配置而第	定	
	@rm -f . tm	p_quiet_recore	dmcount		//也可以没有	有@		
	接着回到 Ma	kefile 的目标	示"%config"	处,内容如	口下:			
	%config: scripts	_basic outputm	nakefile FORCE					
	\$(Q)\$(MAH	KE) \$(build)=s	cripts/kconfig \$@	<u>a</u>				
	将命令展开家	t是:						
	@make -f ./scrip	ts/Makefile.bu	ild obj=scripts/k	config xxx_d	lefconfig			

正点原子

16.5.2 Makefile.build 脚本分析

从上一小节可知, "make xxx_defconfig "配置 Linux 的时候如下两行命令会执行脚本 scripts/Makefile.build:

@make -f ./scripts/Makefile.build obj=scripts/basic

@make -f ./scripts/Makefile.build obj=scripts/kconfig xxx_defconfig

我们依次来分析一下:

1、scripts_basic 目标对应的命令

scripts_basic 目标对应的命令为: @make -f ./scripts/Makefile.build obj=scripts/basic。打开 文件 scripts/Makefile.build,有如下代码:

示例代码 16.5.3 Makefile.build 代码段

39 # The filename Kbuild has precedence over Makefile

```
40 kbuild-dir := $(if $(filter /%,$(src)),$(src),$(srctree)/$(src))
```

41 kbuild-file := \$(if \$(wildcard \$(kbuild-dir)/Kbuild),\$(kbuild-

dir)/Kbuild,\$(kbuild-dir)/Makefile)

42 include \$(kbuild-file)

将 kbuild-dir 展开后为:

kbuild-dir=./scripts/basic

将 kbuild-file 展开后为:

kbuild-file= ./scripts/basic/Makefile

最后将 42 行展开, 即:

include ./scripts/basic/Makefile

继续分析 scripts/Makefile.build,如下代码:

示例代码 16.5.4 Makefile.build 代码段

```
497 __build: $(if $(KBUILD_BUILTIN),$(builtin-target) $(lib-target)
```

 $(extra-y) \setminus$

499

```
498 $ (if $(KBUILD_MODULES),$(obj-m) $(modorder-target)) \
```

```
$(subdir-ym) $(always)
```

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

500 @:

__build 是默认目标,因为命令"@make -f ./scripts/Makefile.build obj=scripts/basic"没有 指定目标,所以会使用到默认目标__build。在顶层 Makefile 中,KBUILD_BUILTIN 为 1, KBUILD MODULES 为空,因此展开后目标 build 为:

正点原子

__build:\$(builtin-target) \$(lib-target) \$(extra-y)) \$(subdir-ym) \$(always)

@:

可以看出目标__build 有 5 个依赖: builtin-target、lib-target、extra-y、subdir-ym 和 always。 这 5 个依赖的具体内容如下:

builtin-target =

lib-target =

extra-y =

subdir-ym =

 $always = scripts/basic/fixdep\ scripts/basic/bin2c$

只有 always 有效,因此__build 最终为:

__build: scripts/basic/fixdep scripts/basic/bin2c

@:

__build 依赖于 scripts/basic/fixdep 和 scripts/basic/bin2c,所以要先将 scripts/basic/fixdep.c 和 scripts/basic/bin2c.c 这两个文件编译成 fixdep 和 bin2c。

综上所述, scripts_basic 目标的作用就是编译 scripts/basic/fixdep 和 scripts/basic/bin2c 这两个文件。

2、%config 目标对应的命令

%config 目标对应的命令为: @make -f ./scripts/Makefile.build obj=scripts/kconfig xxx_defconfig, 此命令会使用到的各个变量值如下:

src= scripts/kconfig

kbuild-dir = ./scripts/kconfig

kbuild-file = ./scripts/kconfig/Makefile

include ./scripts/kconfig/Makefile

可以看出, Makefilke.build 会读取 scripts/kconfig/Makefile 中的内容, 此文件有如下所示 内容:

示例代码 16.5.5 scripts/kconfig/Makefile 代码段

89 %_defconfig: \$(obj)/conf

90 \$(Q)\$< \$(silent) --defconfig=arch/\$(SRCARCH)/configs/\$@ \$(Kconfig) 目标%_defconfig 与 xxx_defconfig 匹配,所以会执行这条规则,将其展开就是:

%_defconfig: scripts/kconfig/conf

@ scripts/kconfig/conf --defconfig=arch/arm/configs/%_defconfig Kconfig

%_defconfig 依赖 scripts/kconfig/conf,所以会编译 scripts/kconfig/conf.c 生成 conf 这个软件。此软件就会将%_defconfig 中的配置输出到.config 文件中,最终生成 Linux kernel 根目录下的.config 文件。

正点原子 原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php 16.5.3 make 过程 使用命令 "make xxx_defconfig" 配置好 Linux 内核以后就可以使用 "make" 或者 "make all"命令进行编译。顶层 Makefile 有如下代码: 示例代码 16.5.6 顶层 Makefile 代码段 14 # That's our default target when none is given on the command line 15 PHONY := _all 16 _all: 575 # If building an external module we do not care about the all: rule 576 # but instead _all depend on modules 577 **PHONY** += all 578 ifeq (\$(KBUILD EXTMOD),) 579 _all: all 580 else 581 _all: modules 582 endif 633 all: vmlinux 第16行,_all是默认目标,如果使用命令"make"编译Linux的话此目标就会被匹配。 第 578 行,如果 KBUILD EXTMOD 为空的话 579 行的代码成立。 第 579 行,默认目标 all 依赖 all。 第633行,目标 all 依赖 vmlinux,所以接下来的重点就是 vmlinux。 顶层 Makefile 中有如下代码: 示例代码 16.5.7 顶层 Makefile 代码段 1038 # Externally visible symbols (used by link-vmlinux.sh) 1039 export KBUILD_VMLINUX_OBJS := \$(head-y) \$(init-y) \$(core-y) \$(libs-y2) \ 1040 \$(drivers-y) \$(net-y) \$(virt-y) 1041 export KBUILD_VMLINUX_LIBS := \$(libs-y1) 1042 export KBUILD_LDS := arch/\$(SRCARCH)/kernel/vmlinux.lds 1043 export LDFLAGS_vmlinux 1044 # used by scripts/Makefile.package 1045 export KBUILD_ALLDIRS := \$(sort \$(filter-out arch/%,\$(vmlinux-alldirs)) LICENSES arch include scripts tools) 1046 1047 vmlinux-deps := \$(KBUILD_LDS) \$(KBUILD_VMLINUX_OBJS) \$(KBUILD_VMLINUX_LIBS) 1048 1049 # Recurse until adjust_autoksyms.sh is satisfied 1050 PHONY += autoksyms_recursive 1051 ifdef CONFIG_TRIM_UNUSED_KSYMS

1052 autoksyms_recursive: descend modules.order

1053 \$(Q)\$(CONFIG_SHELL) \$(srctree)/scripts/adjust_autoksyms.sh \

```
原子哥在线教学: www.yuanzige.com
                                          论坛:www.openedv.com/forum.php
         "$(MAKE) -f $(srctree)/Makefile vmlinux"
    1054
    1055 endif
    1056
   1057 # For the kernel to actually contain only the needed exported symbols,
   1058 # we have to build modules as well to determine what those symbols are.
   1059 # (this can be evaluated only once include/config/auto.conf has been included)
   1060 ifdef CONFIG_TRIM_UNUSED_KSYMS
    1061 KBUILD_MODULES := 1
    1062 endif
    1063
    1064 autoksyms_h := $(if $(CONFIG_TRIM_UNUSED_KSYMS), include/generated/autoksyms.h)
    1065
    1066 $(autoksyms_h):
          $(Q)mkdir -p $(dir $@)
    1067
          $(Q)touch $@
    1068
    1069
   1070 ARCH_POSTLINK := $(wildcard $(srctree)/arch/$(SRCARCH)/Makefile.postlink)
    1071
    1072 # Final link of vmlinux with optional arch pass after final link
    1073 cmd_link-vmlinux =
    1074 $(CONFIG_SHELL) $< $(LD) $(KBUILD_LDFLAGS) $(LDFLAGS_vmlinux); \
   1075 $(if $(ARCH_POSTLINK), $(MAKE) -f $(ARCH_POSTLINK) $@, true)
   1076
   1077 vmlinux: scripts/link-vmlinux.sh autoksyms_recursive $(vmlinux-deps) FORCE
   1078 +$(call if_changed,link-vmlinux)
   从第 1077 行可以看出目标 vmlinux 依赖 scripts/link-vmlinux.sh autoksyms_recursive
$(vmlinux-deps) FORCE。第 1047 行定义了 vmlinux-deps, 值为:
   vmlinux-deps := $(KBUILD_LDS) $(KBUILD_VMLINUX_OBJS) $(KBUILD_VMLINUX_LIBS)
    第 1039 行, KBUILD_VMLINUX_OBJS = $(head-y) $(init-y) $(core-y) $(libs-y2) $(drivers-
y) $(net-y) $(virt-y).
    第 1041 行, export KBUILD_VMLINUX_LIBS = $(libs-y1)。
    第 1042 行, KBUILD_LDS = arch/$(SRCARCH)/kernel/vmlinux.lds, 其中 SRCARCH=arm,
因此 KBUILD_LDS= arch/arm/kernel/vmlinux.lds。
    综上所述, vmlinux 的依赖为: scripts/link-vmlinux.sh、$(head-y)、$(init-y)、$(core-y)、
$(libs-y1)、$(libs-y2)、$(drivers-y)、$(net-y)、$(virt-y)、arch/arm/kernel/vmlinux.lds 和 FORCE。
    第 1078 行的命令用于链接生成 vmlinux。
    重点来看下$(head-y)、$(init-y)、$(core-y)、$(libs-y1)、$(libs-y2)、$(drivers-y)和$(net-y)
这七个变量的值。
```

F点原子

1、head-y

head-y 定义在文件 arch/arm/Makefile 中,内容如下:

示例代码 16.5.8 arch/arm/Makefile 代码段



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

144 head-y := arch/arm/kernel/head\$(MMUEXT).o

当不使能 MMU 的话 MMUEXT=-nommu,如果使能 MMU 的话为空,因此 head-y 最终的 值为:

head-y = arch/arm/kernel/head.o

2、init-y、drivers-y和net-y

在顶层 Makefile 中有如下代码:

示例代码 16.5.9 顶层 Makefile 代码段

1030 init-y := (patsubst %/, %/built-in.a, (init-y))

1031 core-y := \$(patsubst %/, %/built-in.a, \$(core-y))

1032 drivers-y := \$(patsubst %/, %/built-in.a, \$(drivers-y))

1033 net-y := \$(patsubst %/, %/built-in.a, \$(net-y))

1034 libs-y1 := \$(patsubst %/, %/lib.a, \$(libs-y))

1035 libs-y2 := \$(patsubst %/, %/built-in.a, \$(filter-out %.a, \$(libs-y)))

1036 virt-y := \$(patsubst %/, %/built-in.a, \$(virt-y))

从上述示例代码可知, init-y、libs-y、drivers-y和 net-y最终的值为:

init-y = init/built-in.a

drivers-y = drivers/built-in.a sound/built-in.a firmware/built-in.a

net-y = net/built-in.a

3、libs-y1 和 libs-y2

libs-y1和libs-y2基本和init-y一样,在顶层Makefile中存在如下代码:

示例代码 16.5.10 顶层 Makefile 代码段

```
1034 libs-y1 := $(patsubst %/, %/lib.a, $(libs-y))
```

1035 libs-y2 := \$(patsubst %/, %/built-in.a, \$(filter-out %.a, \$(libs-y)))

根据上述示例代码可知, libs-y1应该等于"lib/lib.a", libs-y2应该等于"lib/built-in.a"。

这个只正确了一部分。因为在 arch/arm/Makefile 中向 libs-y 追加了一些值,代码如下:

297 libs-y := arch/arm/lib/ \$(libs-y)

因此可知, libs-y1、libs-y2 最终应该为:

libs-y1 = arch/arm/lib/lib.a lib/lib.a

libs-y2 = arch/arm/lib/built-in.a lib/built-in.a

4、 core-y

core-y和 init-y也一样,在顶层 Makefile 中有如下代码:

示例代码 16.5.11 顶层 Makefile 代码段

```
1016 ifeq ($(KBUILD_EXTMOD),)
```

1017 core-y += kernel/ certs/ mm/ fs/ ipc/ security/ crypto/ block/

但是在 arch/arm/Makefile 中会对 core-y 进行追加,代码如下:

示例代码 16.5.12 arch/arm/Makefile 代码段

276 core-\$(CONFIG_FPE_NWFPE) += arch/arm/nwfpe/

277 # Put arch/arm/fastfpe/ to use this.

```
278 core-$(CONFIG_FPE_FASTFPE) += $(patsubst $(srctree)/%,%,$(wildcard $(srctree)/arch/arm/fastfpe/))
279 core-$(CONFIG_VFP) += arch/arm/vfp/
```

原子哥在线教学	Ź: www.yuanzige.com	论坛:www.openedv.com/forum.php				
280 core-\$(C	ONFIG_XEN) += arch/arm/2	xen/				
281 core-\$(C	281 core-\$(CONFIG_KVM_ARM_HOST) += arch/arm/kvm/					
282 core-\$(C	ONFIG_VDSO) += arch/arr	n/vdso/				
283						
284 # If we ha	ave a machine-specific direct	ory, then include it in the build.				
285 core-y	+= arch/arm/kernel/ and	rch/arm/mm/ arch/arm/common/				
286 core-y	+= arch/arm/probes/					
287 core-y	+= arch/arm/net/					
288 core-y	+= arch/arm/crypto/					
289 core-y	+= \$(machdirs) \$(plat	dirs)				
第 276~282	行根据不同的配置向。	core-y 追加不同的值,比如使能 VFP 的话就会在.config				
中有 CONFIG_	VFP=y 这一行,那么 co	ore-y 就会追加"arch/arm/vfp/"。				
第 285~289)行就是对 core-y 直接追	追加的值。				
在顶层 Ma	kefile 中有如下一行:					
	示例代	码 16.5.13 顶层 Makefile 代码段				
1031 core-y	:= \$(patsubst %/, %/built-i	n.a, \$(core-y))				
经过上述代	码的转换,最终 core-y	的值为:				
core-y =usr/b	uilt-in.a	arch/arm/vfp/built-in.a \				
cer	ts/built-in.a arch/a	arm/kernel/built-in.a \				
arch	n/arm/mm/built-in.a	arch/arm/common/built-in.a \				
arch	n/arm/probes/built-in.a	arch/arm/net/built-in.a \				
arch	n/arm/crypto/built-in.a	arch/arm/firmware/built-in.a \				
arch	n/arm/mach-zynq/built-in.a	kernel/built-in.a\				
mm	/built-in.a	fs/built-in.a \				
ipc/	built-in.a	security/built-in.a \				
cry	oto/built-in.a	block/built-in.a				

2 正占原子

关于 head-y、init-y、core-y、libs-y1、libs-y2、drivers-y和 net-y这7个变量就讲解到这里。 这些变量都是一些 built-in.o 或.a 等文件,这个和 uboot 一样,都是将相应目录中的源码文件进 行编译,然后在各自目录下生成 built-in.o 文件,有些生成了.a 库文件。最终将这些 built-in.o 和.a 文件进行链接即可形成 ELF 格式的可执行文件,也就是 vmlinux。但是链接是需要连接脚 本的,vmlinux 依赖的 arch/arm/kernel/vmlinux.lds 就是整个 Linux 的链接脚本。

在示例代码 16.5.7 的第 1078 行的命令 "+\$(call if_changed,link-vmlinux)"表示将\$(call if_changed,link-vmlinux)的结果作为最终生成 vmlinux 的命令,前面的 "+"表示该命令结果不可忽略。\$(call if_changed,link-vmlinux)调用函数 if_changed, link-vmlinux 是函数 if_changed 的参数,函数 if_changed 定义在文件 scripts/Kbuild.include 中,如下所示:

示例代码 16.5.14 scripts/Kbuild.include 代码段

\

```
218 if_changed = $(if $(any-prereq)$(cmd-check),
```

219 \$(cmd);

220 printf '%s\n' 'cmd_\$@ := \$(make-cmd)' > \$(dot-target).cmd, @:)

any-prereq 用于检查依赖文件是否有变化,如果依赖文件有变化那么 any-prereq 就不为空, 否则就为空。cmd-check 用于检查参数是否有变化,如果没有变化那么 cmd-check 就为空。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

第 219 行, \$(cmd)用于打印命令执行过程,比如在链接 vmlinux 的时候就会输出 "LINK vmlinux"。\$(cmd)执行 cmd_link-vmlinux 的内容。cmd_link-vmlinux 在顶层 Makefile 中有如下 所示定义:

示例代码 16.5.15 顶层 Makefile 代码段

1072 # Final link of vmlinux with optional arch pass after final link

1073 cmd_link-vmlinux =

1074 \$(CONFIG_SHELL) \$< \$(LD) \$(LDFLAGS) \$(LDFLAGS_vmlinux); \

1075 \$(if \$(ARCH_POSTLINK), \$(MAKE) -f \$(ARCH_POSTLINK) \$@, true)

第 1074~1075 行就是 cmd_link-vmlinux 的值,其中 CONFIG_SHELL=/bin/bash、\$<表示目标 vmlinux 的第一个依赖文件,根据示例代码 16.5.7 的第 1077 行可知,这个文件为 scripts/link-vmlinux.sh。

LD= arm-xilinx-linux-gnueabi-ld -EL, LDFLAGS 为空。LDFLAGS_vmlinux 的值由顶层 Makefile 和 arch/arm/Makefile 这两个文件共同决定,最终 LDFLAGS_vmlinux=-p--no-undefined -X --pic-veneer --build-id。第 1075 行的语句因为 ARCH_POSTLINK 为空,所以不起作用。因此 cmd_link-vmlinux 最终的值为:

cmd_link-vmlinux = /bin/bash scripts/link-vmlinux.sh arm-xilinx-linux-gnueabi-ld -EL -p --no-undefined -X -- pic-veneer --build-id

cmd_link-vmlinux会调用 scripts/link-vmlinux.sh 脚本来链接出 vmlinux 文件。在 scripts/link-vmlinux.sh 中有如下所示代码:

示例代码 16.5.16 scripts/link-vmlinux.sh 代码段

```
58 # Link of vmlinux
59 # ${1} - output file
60 # \{2\}, \{3\}, \dots - optional extra .0 files
61 vmlinux_link()
62 {
63 local lds="${objtree}/${KBUILD_LDS}"
64 local output=\frac{1}{1}
65 local objects
66
67 info LD ${output}
68
69 # skip output file argument
70 shift
71
72 if [ "${SRCARCH}" != "um" ]; then
73
     objects="--whole-archive
                                  \
74
        {KBUILD VMLINUX OBJS}
                                          \
75
       --no-whole-archive
76
        --start-group
        {KBUILD_VMLINUX_LIBS}
77
                                          \
78
        --end-group
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

79	\${@}"
80	
81 \$	<pre>{LD} \${KBUILD_LDFLAGS} \${LDFLAGS_vmlinux} \</pre>
82	-o <mark>\${output}</mark> \
83	-T \${lds} \${objects}
84 else	
<mark>85</mark> o	bjects="-Wl,whole-archive \
86	\${KBUILD_VMLINUX_OBJS} \
87	-Wl,no-whole-archive
88	-Wl,start-group
89	\${KBUILD_VMLINUX_LIBS}
90	-Wl,end-group
91	\${@}"
92	
93 \$	{CC} \${CFLAGS_vmlinux}
94	-o <mark>\${output}</mark> \
95	-Wl,-T, <mark>\${lds}</mark>
96	\${objects}
97	-lutil -lrt -lpthread
98 ri	m -f linux
99 fi	
100 }	
299 vm	linux_link vmlinux "\${kallsymso}" \${btf_vmlinux_bin_o}
	$I \to I \to$

vmliux_link 就是最终链接出 vmlinux 的函数,第72行判断 SRCARCH 是否等于"um",如果不相等的话就执行 73~83 行的代码。第81~83 行的代码应该很熟悉了,就是普通的链接操作,连接脚本为 lds= ./arch/arm/kernel/vmlinux.lds,需要链接的文件由变量KBUILD_VMLINUX_LIBS 来决定,这个变量在本节中已经讲解过了。

第299行调用 vmlinux_link 函数来链接出 vmlinux。

使用命令"make V=1"编译 Linux, 会有如下图所示的编译信息:

+ arm-xilinx-linux-gnueabi-ld -EL --no-undefined -X --pic-veneer --build-id -o v mlinux -T ./arch/arm/kernel/vmlinux.lds --whole-archive arch/arm/kernel/head.o i nit/built-in.a usr/built-in.a arch/arm/vfp/built-in.a arch/arm/kernel/built-in.a

图 16.5.1 link-vmlinux.sh 链接 vmlinux 的过程

至此我们基本理清了 make 的过程,重点就是将各个子目录下的 built-in.o、.a 等文件链接 在 一 起 , 最 终 生 成 vmlinux 这 个 ELF 格 式 的 可 执 行 文 件 。 链 接 脚 本 为 arch/arm/kernel/vmlinux.lds, 链接过程是由 shell 脚本 scripts/link-vmlinux.s 来完成的。接下来 的问题就是这些子目录下的 built-in.o、.a 等文件又是如何编译出来的呢?



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

16.5.4 built-in.o 文件编译生成过程

根据示例代码 16.5.7 的第 1077 行可知, vmliux 依赖 vmlinux-deps, 而 vmlinux-deps= \$(KBUILD_LDS) \$(KBUILD_VMLINUX_INIT) \$(KBUILD_VMLINUX_MAIN) \$(KBUILD_VMLINUX_LIBS)。 KBUILD_LDS 是连接脚本,这里不考虑,剩下的 KBUILD_VMLINUX_INIT、KBUILD_VMLINUX_MAIN 和 KBUILD_VMLINUX_LIBS 就是 各个子目录下的 built-in.o 和.a 等文件。最终 vmlinux-deps 的值如下:

vmlinux-deps = arch/arm/kernel/vmlinux.lds arch/arm/kernel/head.o

init/built-in.o	usr/built-in.o
arch/arm/vfp/built-in.o	arch/arm/kernel/built-in.o
arch/arm/mm/built-in.c	arch/arm/common/built-in.o
arch/arm/probes/built-i	in.o arch/arm/net/built-in.o
arch/arm/crypto/built-i	n.o arch/arm/firmware/built-in.o
arch/arm/lib/lib.a	arch/arm/mach-zynq/built-in.o
lib/lib.a	kernel/built-in.o
certs/built-in.o	mm/built-in.o
fs/built-in.o	ipc/built-in.o
security/built-in.o	crypto/built-in.o
block/built-in.o	arch/arm/lib/built-in.o
lib/built-in.o	drivers/built-in.o
sound/built-in.o	firmware/built-in.o
net/built-in.o	virt/built-in.o

除了 arch/arm/kernel/vmlinux.lds 以外,其他都是要编译链接生成的。在顶层 Makefile 中有 如下代码:

示例代码 16.5.17 顶层 Makefile 代码段

1084 \$(sort \$(vmlinux-deps)): descend ;

sort 是排序函数,用于对 vmlinux-deps 的字符串列表进行排序,并且去掉重复的单词。 vmlinux-deps 依赖 vmlinux-dirs, vmlinux-dirs 也在顶层 Makefile 中定义,定义如下:

示例代码 16.5.18 顶层 Makefile 代码段

1019 vmlinux-dirs := $(patsubst \%/,\%,$(filter \%/, $(init-y) $(init-m) \)$

1020 (core-y) (core-m) (drivers-y) (drivers-m)

```
1021 $(net-y) $(net-m) $(libs-y) $(libs-m) $(virt-y)))
```

vmlinux-dirs 看名字就知道和目录有关,此变量保存着生成 vmlinux 所需源码文件的目录, 值如下:

vmlinux-dir	rs= init		usr		arch/arm/vfp
arc	arch/arm/kernel		arch/arm/mm		arch/arm/common
arc	h/arm/probes		arch/arr	n/net	arch/arm/crypto
arc	h/arm/firmwa	are	drivers	5	arch/arm/mach-zynq
arc	h/arm/lib	ke	rnel	(certs
mn	n f	fs		ipc	
sec	curity	cryp	to	blo	ock
sou	und	firmv	vare	ne	et



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

virt

在顶层 Makefile 中有如下代码:

示例代码 16.5.19 顶层 Makefile 代码段

1650 PHONY += descend \$(build-dirs)

1651 descend: \$(build-dirs)

lib

1652 \$(build-dirs): prepare

1653 \$(Q)\$(MAKE) \$(build)=\$@ single-build=\$(single-build) need-builtin=1 need-modorder=1

目标 vmlinux-dirs 依赖 prepare 和 scripts,这两个依赖就不去浪费时间分析了,重点看一 下第 1653 行的命令。build 前面已经说了, 值为 "-f./scripts/Makefile.build obj", 因此将 1653 行的命令展开就是:

@ make -f ./scripts/Makefile.build obj=\$@

\$@表示目标文件,也就是 vmlinux-dirs 的值,将 vmlinux-dirs 中的这些目录全部带入到命 令中,结果如下:

@ make -f ./scripts/Makefile.build obj=init

@ make -f ./scripts/Makefile.build obj=usr

@ make -f ./scripts/Makefile.build obj=arch/arm/vfp

@ make -f ./scripts/Makefile.build obj=arch/arm/kernel

@ make -f ./scripts/Makefile.build obj=arch/arm/mm

@ make -f ./scripts/Makefile.build obj=arch/arm/common

@ make -f ./scripts/Makefile.build obj=arch/arm/probes

@ make -f ./scripts/Makefile.build obj=arch/arm/net

@ make -f ./scripts/Makefile.build obj=arch/arm/crypto

@ make -f ./scripts/Makefile.build obj=arch/arm/firmware

@ make -f ./scripts/Makefile.build obj=arch/arm/mach-zynq

@ make -f ./scripts/Makefile.build obj=kernel

@ make -f ./scripts/Makefile.build obj=certs

@ make -f ./scripts/Makefile.build obj=mm

@ make -f ./scripts/Makefile.build obj=fs

@ make -f ./scripts/Makefile.build obj=ipc

@ make -f ./scripts/Makefile.build obj=security

@ make -f ./scripts/Makefile.build obj=crypto

@ make -f ./scripts/Makefile.build obj=block

@ make -f ./scripts/Makefile.build obj=drivers

@ make -f ./scripts/Makefile.build obj=sound

@ make -f ./scripts/Makefile.build obj=firmware

@ make -f ./scripts/Makefile.build obj=net

@ make -f ./scripts/Makefile.build obj=arch/arm/lib

@ make -f ./scripts/Makefile.build obj=lib

这些命令运行过程其实都是一样的,我们就以"@ make -f ./scripts/Makefile.build obj=init" 这个命令为例,讲解一下详细的运行过程。这里又要用到 Makefile.build 这个脚本了,此脚本



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php 默认目标为__build,这个在 16.5.2 小节已经讲过了,我们再来看一下,__build 目标对应的规则如下:

示例代码 16.5.20 scripts/Makefile.build 代码段

497 __build: \$(if \$(KBUILD_BUILTIN),\$(builtin-target) \$(lib-target)
\$(extra-y)) \

```
97 $ (if $(KBUILD_MODULES),$(obj-m) $(modorder-target)) \
```

98 \$(subdir-ym) \$(always)

99 @:

当只编译 Linux 内核镜像文件,也就是使用"make zImage"编译的时候, KBUILD_BUILTIN=1,KBUILD_MODULES 为空。"make"命令是会编译所有的东西,包括 Linux 内核镜像文件和一些模块文件。所以如果只编译 Linux 内核镜像的话,__build 目标可简化为:

__build: \$(builtin-target) \$(lib-target) \$(extra-y)) \$(subdir-ym) \$(always)

@:

重点来看一下 builtin-target 这个依赖, builtin-target 同样定义在文件 scripts/Makefile.build 中, 定义如下:

示例代码 16.5.21 scripts/Makefile.build 代码段

69 ifneq (\$(strip \$(real-obj-y) \$(need-builtin)),)

70 builtin-target := \$(obj)/built-in.a

71 endif

第 70 行就是 builtin-target 变量的值,为 "\$(obj)/built-in.a",这就是这些 built-in.a 的来源 了。要生成 built-in.a,要求 obj-y、obj-m、obj-、subdir-m 和 lib-target 这些变量不能全为空。 最后一个问题: built-in.a 是怎么生成的? 在文件 scripts/Makefile.build 中有如下代码:

示例代码 16.5.22 scripts/Makefile.build 代码段

386 #

```
387 # Rule to compile a set of .o files into one .a file (without symbol table)
388 #
389 ifdef builtin-target
390
391 quiet_cmd_ar_builtin = AR $@
392 cmd_ar_builtin = rm -f $@; $(AR) cDPrST $@ $(real-prereqs)
393
394 $(builtin-target): $(real-obj-y) FORCE
395 $(call if_changed,ar_builtin)
396
397 targets += $(builtin-target)
398 endif # builtin-target
```

第 394 行 的 目 标 就 是 builtin-target, 依 赖 为 real-obj-y, 命 令 为 " \$(call if_changed,link_o_target)",也就是调用函数 if_changed,参数为 ar_builtin,其返回值就是具体的命令。cmd_ar_builtin 使用 LD 将某个目录下的所有.o 文件链接在一起,最终形成 built-in.o。



正点原子

16.5.5 make zImage 过程

1、vmlinux、Image, zImage、uImage的区别

前面几小节重点是讲 vmlinux 是如何编译出来的, vmlinux 是 ELF 格式的文件, 但是在实 际中我们不会使用 vmlinux, 而是使用 zImage 或 uImage 这样的 Linux 内核镜像文件。那么 vmlinux、zImage、uImage它们之间有什么区别呢?

vmlinux 是编译出来的最原始的内核文件,是未压缩的,比如正点原子提供的 Linux 源码 编译出来的 vmlinux 差不多有 13MB,如下图所示:

wmg@Linux:~/petalinux/atk-zyng/atk-zyng-linux-xlnx\$ ls -l vmlinux -rwxrwxr-x 1 wmq wmq 13950072 8月 17 14:22 vmlinux wmq@Linux:~/petalinux/atk-zynq/atk-zynq-linux-xlnx\$ wmq@Linux:~/petalinux/atk-zynq/atk-zynq-linux-xlnx\$

图 16.5.2 vmlinux 信息

Image 是 Linux 内核镜像文件,但是 Image 仅包含可执行的二进制数据。Image 就是使用 objcopy 去掉 vmlinux 中的一些信息,比如符号表什么的。但是 Image 是没有压缩过的, Image 保存在 arch/arm/boot 目录下,其大小大概在 12MB 左右,如下图所示:

wmq@Linux:~/petalinux/atk-zynq/atk-zynq-linux-xlnx\$ ls -l arch/arm/boot/Image -rwxrwxr-x 1 wmq wmq 12769856 8月 17 14:22 arch/arm/boot/Image wmq@Linux:~/petalinux/atk-zynq/atk-zynq-linux-xlnx\$ wmq@Linux:~/petalinux/atk-zynq/atk-zynq-linux-xlnx\$

图 16.5.3 Image 镜像信息

可见相比 vmlinux 的 13MB, Image 缩小了 1MB, 然而在嵌入式中, Image 可能还是比较 大,所以需要压缩。

zImage 是经过 gzip 压缩后的 Image,经过压缩以后其大小为 4.8MB,如下图所示: wmq@Linux:~/petalinux/atk-zynq/atk-zynq-linux-xlnx\$ ls -l arch/arm/boot/zImage -rwxrwxr-x 1 wmq wmq 4830352 8月 17 14:22 arch/arm/boot/zImage wmq@Linux:~/petalinux/atk-zynq/atk-zynq-linux-xlnx\$ wmq@Linux:~/petalinux/atk-zynq/atk-zynq-linux-xlnx\$

图 16.5.4 zImage 镜像信息

uImage 是老版本 uboot 专用的镜像文件, uImag 是在 zImage 前面加了一个长度为 64 字节 的"头",这个头信息描述了该镜像文件的类型、加载位置、生成时间、大小等信息。但是 新的 uboot 已经支持了 zImage 启动,所以已经很少用到 uImage 了。

使用"make"、"make all"、"make zImage"这些命令就可以编译出 zImage 镜像,在 arch/arm/Makefile 中有如下代码:

示例代码 16.5.23 arch/arm/Makefile 代码段

339 \$(BOOT TARGETS): vmlinux

340 \$(Q)\$(MAKE) \$(build)=\$(boot) MACHINE=\$(MACHINE) \$(boot)/\$@

341 @\$(kecho) ' Kernel: \$(boot)/\$@ is ready'

第 339 行, BOOT_TARGETS 依赖 vmlinux, 因此如果使用"make zImage"编译 Linux 内 核的话,首先肯定要先编译出 vmlinux。

第 340 行,具体的命令,比如要编译 zImage,那么命令展开以后如下所示:

@ make -f ./scripts/Makefile.build obj=arch/arm/boot MACHINE=arch/arm/boot/zImage 看来又是使用 scripts/Makefile.build 文件来完成 vmliux 到 zImage 的转换。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

关于 Linux 顶层 Makefile 就讲解到这里,基本和 uboot 的顶层 Makefile 一样,重点在于 vmlinux 的生成。最后将 vmlinux 压缩成我们最常用的 zImage 或 uImage 等文件。

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

正点原子

第十七章 Linux 内核启动流程

看完Linux内核的顶层 Makefile 文件以后我们再来看下Linux内核的大致启动流程。Linux 内核的启动流程要比 uboot 复杂的多,涉及到的内容也更多,因此本章我们大致的了解一下 Linux 内核的启动流程即可。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

17.1 链接脚本 vmlinux.lds

要分析 Linux 启动流程,同样需要先编译一下 Linux 源码,因为有很多文件是需要编译才 会生成的。首先分析 Linux 内核的连接脚本文件 arch/arm/kernel/vmlinux.lds,通过链接脚本可 以找到 Linux 内核的第一行程序是从哪里执行的。vmlinux.lds 链接脚本中有如下代码:

1 OUTPUT_ARCH(arm)

2 ENTRY(stext)

3 jiffies = jiffies_64;

4 SECTIONS

5 {

6 /DISCARD/: {

7 *(.ARM.exidx.exit.text) *(.ARM.extab.exit.text) *(.ARM.exidx.text.exit) *(.ARM.extab.text.exit) *(.exitcall.exit) *(.discard) *(.discard.*)

8 }

第2行的 ENTRY 指明了了 Linux 内核入口,入口为 stext, stext 定义在文件 arch/arm/kernel/head.S 中,因此要分析 Linux 内核的启动流程,就得先从文件 arch/arm/kernel/head.S 的 stext 处开始分析。

17.2 Linux 内核启动流程分析

17.2.1 Linux 内核入口 stext

stext 是 Linux 内核的入口地址,在文件 arch/arm/kernel/head.S 中有如下所示提示内容:

/*

* Kernel startup entry point.

* _____

*

* This is normally called from the decompressor code. The requirements

- * are: MMU = off, D-cache = off, I-cache = dont care, r0 = 0,
- * r1 = machine nr, r2 = atags or dtb pointer.

```
·····
*/
```

根据该提示可知 Linux 内核启动之前要求如下:

- ①、关闭 MMU。
- ②、关闭 D-cache。
- ③、I-Cache 无所谓。
- ④、r0=0∘
- ⑤、r1=machine nr(也就是机器 ID)。
- ⑥、r2=atags或者设备树(dtb)首地址。

Linux 内核的入口点 stext 其实相当于内核的入口函数, stext 函数内容如下:

示例代码 17.2.1 arch/arm/kernel/head.S 代码段

77 ENTRY(stext)



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php 85 #ifdef CONFIG_ARM_VIRT_EXT 86 bl __hyp_stub_install 87 #endif 88 @ ensure svc mode and all interrupts masked 89 safe svcmode maskall r9 90 91 mrc p15, 0, r9, c0, c0 @ get processor id 92 bl __lookup_processor_type @ r5=procinfo r9=cpuid 93 movs r10, r5 @ invalid processor (r5=0)? 94 THUMB(it eq) @ force fixup-able long branch encoding 95 beq __error_p @ yes, error 'p' • • • • • • 105 #ifndef CONFIG_XIP_KERNEL 110 #else 111 ldr r8, =PLAT_PHYS_OFFSET @ always constant in this case 112 #endif 113 114 /* 115 * r1 = machine no, r2 = atags or dtb, 116 * r8 = phys_offset, r9 = cpuid, r10 = procinfo */ 117 118 bl __vet_atags 119 #ifdef CONFIG_SMP_ON_UP 120 bl __fixup_smp 121 #endif 122 #ifdef CONFIG_ARM_PATCH_PHYS_VIRT 123 bl __fixup_pv_table 124 #endif bl __create_page_tables 125 126 127 /* 128 * The following calls CPU specific code in a position independent 129 * manner. See arch/arm/mm/proc-*.S for details. r10 = base of130 * xxx_proc_info structure selected by __lookup_processor_type 131 * above. 132 * 133 * The processor init function will be called with: 134 * r1 - machine type 135 * r2 - boot data (atags/dt) pointer

```
原子哥在线教学: www.yuanzige.com
                                            论坛:www.openedv.com/forum.php
    136 * r4 - translation table base (low word)
    137 * r5 - translation table base (high word, if LPAE)
    138 * r8 - translation table base 1 (pfn if LPAE)
    139 * r9 - cpuid
    140 * r13 - virtual address for enable mmu -> turn mmu on
    141 *
    142 * On return, the CPU will be ready for the MMU to be turned on,
    143 * r0 will hold the CPU control register value, r1, r2, r4, and
    144 * r9 will be preserved. r5 will also be preserved if LPAE.
    145 */
    146 ldr r13, = __mmap_switched
                                   @ address to jump to after
                   @ mmu has been enabled
    147
    148 badr lr, 1f
                        @ return (PIC) address
    149 #ifdef CONFIG_ARM_LPAE
    150
       mov r5, #0
                         @ high TTBR0
    151 mov r8, r4, lsr #12
                             @ TTBR1 is swapper_pg_dir pfn
    152 #else
    153 mov r8, r4
                         @ set TTBR1 to swapper_pg_dir
    154 #endif
    155 ldr r12, [r10, #PROCINFO_INITFUNC]
    156 add r12, r12, r10
    157 ret r12
    158 : b __enable_mmu
    159 ENDPROC(stext)
    第 85 行,如果配置了 CONFIG ARM VIRT EXT (ARM 虚拟化扩展)则跳转到
__hyp_stub_install 处。__hyp_stub_install 定义在 arch\arm\kernel\hyp-stub.S 文件中。
    第 89 行,调用函数 safe_svcmode_maskall 确保 CPU 处于 SVC 模式,并且关闭了所有的
```

正点原子

```
中断。safe_svcmode_maskall 定义在文件 arch/arm/include/asm/assembler.h 中。
```

第91行,读处理器 ID, ID 值保存在 r9 寄存器中。

第 92 行,调用函数__lookup_processor_type 检查当前系统是否支持此 CPU,如果支持的 就获取 procinfo 信息。procinfo 是 proc_info_list 类型的结构体, proc_info_list 在文件 arch/arm/include/asm/procinfo.h中的定义如下:

示例代码 17.2.2 proc_info_list 结构体

```
struct proc_info_list {
  unsigned int
                       cpu_val;
  unsigned int
                   cpu_mask;
  unsigned long
                                                   /* used by head.S */
                       ___cpu_mm_mmu_flags;
  unsigned long
                       __cpu_io_mmu_flags; /* used by head.S */
  unsigned long
                        __cpu_flush;
                                          /* used by head.S */
                       *arch_name;
  const char
  const char
                       *elf name;
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

unsigned int e	II_nwcap;
const char	*cpu_name
struct processor	*proc;
<pre>struct cpu_tlb_fns</pre>	*tlb;
<pre>struct cpu_user_fn</pre>	s *user;
struct cpu_cache_f	ns *cache;

};

Linux 内核将每种处理器都抽象为一个 proc_info_list 结构体,每种处理器都对应一个 procinfo。因此可以通过处理器 ID 来找到对应的 procinfo 结构, __lookup_processor_type 函数 找到对应处理器的 procinfo 以后会将其保存到 r5 寄存器中。

继续回到示例代码 17.2.1 中,第 118 行,调用函数__vet_atags 验证 atags 或设备树(dtb)的 合法性。函数__vet_atags 定义在文件 arch/arm/kernel/head-common.S 中。

第125行,调用函数__create_page_tables创建页表。

第 146 行,将函数__mmap_switched 的地址保存到 r13 寄存器中。__mmap_switched 定义 在文件 arch/arm/kernel/head-common.S, __mmap_switched 最终会调用 start_kernel 函数。

第 158 行, 调用 __enable_mmu 函数使能 MMU, __enable_mmu 定义在文件 arch/arm/kernel/head.S 中。__enable_mmu 最终会通过调用__turn_mmu_on 来打开 MMU, __turn_mmu_on 最后会执行 r13 里面保存的__mmap_switched 函数。

17.2.2 __mmap_switched 函数

___mmap_switched 函数定义在文件 arch/arm/kernel/head-common.S 中,函数代码如下: 示例代码 17.2.3 __mmap_switched 函数

```
77 __mmap_switched:
78
79 mov r7, r1
80 mov r8, r2
81 mov r10, r0
82
83 adr r4, __mmap_switched_data
84 mov fp, #0
85
86 #if defined(CONFIG_XIP_DEFLATED_DATA)
87 ARM( ldr sp, [r4], #4 )
88 THUMB( ldr sp, [r4] )
89 THUMB( add r4, #4 )
```

```
90 bl __inflate_kernel_data @ decompress .data to RAM
```

```
91 teq r0, #0
```

```
92 bne __error
```

```
93 #elif defined(CONFIG_XIP_KERNEL)
```

```
94 ARM( ldmia r4!, {r0, r1, r2, sp})
```

```
95 THUMB( ldmia r4!, {r0, r1, r2, r3} )
```



```
原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php
    96 THUMB( mov sp, r3)
    97 sub r2, r2, r1
    98 bl memcpy
                        @ copy .data to RAM
    99 #endif
    100
    101 ARM( ldmia r4!, {r0, r1, sp})
    102 THUMB( ldmia r4!, {r0, r1, r3})
    103 THUMB( mov sp, r3)
    104 sub r2, r1, r0
    105 mov r1, #0
    106 bl memset
                        @ clear .bss
    107
    108 ldmia r4, {r0, r1, r2, r3}
    109 str r9, [r0]
                      @ Save processor ID
    110 str r7, [r1]
                      @ Save machine type
    111 str r8, [r2]
                      @ Save atags pointer
    112 cmp r3, #0
    113 strne r10, [r3]
                          @ Save control register values
    114 mov lr, #0
    115 b start_kernel
    116 ENDPROC(__mmap_switched)
    第 115 行最终调用 start_kernel 来启动 Linux 内核, start_kernel 函数定义在文件 init/main.c
中。
```

17.2.3 start_kernel 函数

start_kernel 通过调用众多的子函数来完成 Linux 启动之前的一些初始化工作,由于 start_kernel 函数里面调用的子函数太多,而这些子函数又很复杂,因此我们简单的来看一下 一些重要的子函数。精简并添加注释后的 start_kernel 函数内容如下:

示例代码 17.2.4 start_kernel 函数

```
asmlinkage __visible void __init start_kernel(void)
{
    char *command_line;
    char *after_dashes;

set_task_stack_end_magic(&init_task);/* 设置任务栈结束魔术数,用于栈溢出检测*/
    smp_setup_processor_id(); /* 跟 SMP 有关(多核处理器),设置处理器 ID.
        * 有很多资料说 ARM 架构下此函数为空函数,那是因
        * 为他们用的老版本 Linux,而那时候 ARM 还没有多
        * 核处理器。
*/
```

```
debug_objects_early_init(); /* 做一些和 debug 有关的初始化 */
```



```
原子哥在线教学: www.yuanzige.com
                                    论坛:www.openedv.com/forum.php
     cgroup_init_early(); /* cgroup 初始化, cgroup 用于控制 Linux 系统资源*/
    local_irq_disable(); /* 关闭当前 CPU 中断 */
    early_boot_irqs_disabled = true;
    /*
     *中断关闭期间做一些重要的操作,然后打开中断
     */
     boot_cpu_init();
                       /* 跟 CPU 有关的初始化 */
                       /* 页地址相关的初始化 */
     page_address_init();
     pr notice("%s", linux banner);/* 打印 Linux 版本号、编译时间等信息 */
     setup_arch(&command_line); /* 架构相关的初始化,此函数会解析传递进来的
                   * ATAGS 或者设备树(DTB)文件。会根据设备树里面
                   *的 model 和 compatible 这两个属性值来查找
                   *Linux 是否支持这个单板。此函数也会获取设备树
                   *中 chosen 节点下的 bootargs 属性值来得到命令
                   * 行参数,也就是 uboot 中的 bootargs 环境变量的
   * 值, 获取到的命令行参数会保存到
   *command line 中。
                   */
     mm_init_cpumask(&init_mm); /* 看名字, 应该是和内存有关的初始化 */
     setup command line(command line); /* 好像是存储命令行参数 */
     setup_nr_cpu_ids();
                       /* 如果只是 SMP(多核 CPU)的话,此函数用于获取
                    * CPU 核心数量, CPU 数量保存在变量
                    * nr_cpu_ids 中。
   */
     setup_per_cpu_areas(); /* 在 SMP 系统中有用,设置每个 CPU 的 per-cpu 数据 */
   boot_cpu_state_init();
   smp_prepare_boot_cpu();
```

build_all_zonelists(NULL, NULL); /* 建立系统内存页区(zone)链表 */ page_alloc_init(); /* 处理用于热插拔 CPU 的页 */

/* 打印命令行信息 */



下点原子

```
原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php
     profile_init();
     call function init();
     WARN(!irqs_disabled(), "Interrupts were enabled early\n");
     early_boot_irqs_disabled = false;
     local_irq_enable(); /* 使能中断 */
     kmem_cache_init_late(); /* slab 初始化, slab 是 Linux 内存分配器 */
                      /* 初始化控制台,之前 printk 打印的信息都存放
     console_init();
                    *缓冲区中,并没有打印出来。只有调用此函数
                    *初始化控制台以后才能在控制台上打印信息。
                    */
     if (panic_later)
       panic("Too many boot %s vars at `%s"", panic_later,
          panic_param);
     lockdep_info();/* 如果定义了宏 CONFIG_LOCKDEP,那么此函数打印一些信息。*/
     locking_selftest()
                      /* 锁自测 */
     .....
     page_ext_init();
                          /* kmemleak 初始化, kmemleak 用于检查内存泄漏 */
     kmemleak_init();
   debug_objects_mem_init();
   setup_per_cpu_pageset();
     numa_policy_init();
     if (late_time_init)
       late_time_init();
     calibrate_delay(); /* 测定 BogoMIPS 值,可以通过 BogoMIPS 来判断 CPU 的性能
                 * BogoMIPS 设置越大,说明 CPU 性能越好。
                 */
     pidmap_init();
                      /* PID 位图初始化 */
                      /* 生成 anon vma slab 缓存 */
     anon_vma_init();
     acpi_early_init();
     •••••
     thread_stack_cache_init();
                  /* 为对象的每个用于赋予资格(凭证) */
     cred_init();
     fork_init();
                  /* 初始化一些结构体以使用 fork 函数
                                                     */
     proc caches init(); /* 给各种资源管理结构分配缓存
                                                         */
                     /* 初始化缓冲缓存
                                                         */
     buffer_init();
                     /* 初始化密钥
     key_init();
                                                         */
                     /* 安全相关初始化
     security_init();
                                                         */
     dbg_late_init();
```

下点原子



```
原子哥在线教学: www.yuanzige.com
                                       论坛:www.openedv.com/forum.php
     vfs_caches_init(totalram_pages);
                                /* 为 VFS 创建缓存
                                                   */
                             /* 初始化信号
     signals init();
                                               */
                                /* 注册并挂载 proc 文件系统 */
     proc_root_init();
     nsfs_init();
                    /* 初始化 cpuset, cpuset 是将 CPU 和内存资源以逻辑性
     cpuset init();
                 *和层次性集成的一种机制,是cgroup使用的子系统之一
                 */
                             /* 初始化 cgroup */
     cgroup_init();
     taskstats_init_early();
                        /* 进程状态初始化 */
     delayacct_init();
                         /* 检查写缓冲一致性 */
     check_bugs();
     acpi_subsystem_init();
     sfi_init_late();
     if (efi_enabled(EFI_RUNTIME_SERVICES)) {
       efi_free_boot_services();
     }
     rest_init();
                     /* rest_init 函数 */
   }
    start_kernel 里面调用了大量的函数,每一个函数都是一个庞大的知识点,如果想要学习
```

start_kernel 里面调用了大量的函数,每一个函数都是一个庞大的知识点,如果想要学习 Linux 内核,那么这些函数就需要去详细的研究。本教程注重于嵌入式 Linux 入门,因此不会 去讲太多关于 Linux 内核的知识。start_kernel 函数最后调用了 rest_init,接下来简单看一下 rest_init 函数。

17.2.4 rest_init 函数

```
rest_init 函数定义在文件 init/main.c 中, 函数内容如下:
```

```
示例代码 17.2.5 rest_init 函数
406 noinline void ____ref rest_init(void)
407 {
408
      struct task struct *tsk;
409
      int pid;
410
411
      rcu_scheduler_starting();
412
      /*
413
      * We need to spawn init first so that it obtains pid 1, however
       * the init task will end up wanting to create kthreads, which, if
414
415
       * we schedule it before we create kthreadd, will OOPS.
```



正点原子

第417行,调用函数kernel_thread创建kernel_init线程,也就是大名鼎鼎的init内核进程。 init 进程的 PID 为 1。init 进程一开始是内核进程(也就是运行在内核态),后面 init 进程会在根



原子哥在线教学:www.yuanzige.com 论坛:www.openedv.com/forum.php 文件系统中查找名为 "init" 这个程序,这个 "init" 程序处于用户态,通过运行这个 "init" 程序, init 进程就会实现从内核态到用户态的转变。

第 429 行,调用函数 kernel_thread 创建 kthreadd 内核进程,此内核进程的 PID 为 2。 kthreadd 进程负责所有内核进程的调度和管理。

第 451 行,最后调用函数 cpu_startup_entry 来进入 idle 进程, cpu_startup_entry 会调用 cpu_idle_loop, cpu_idle_loop 是个 while 循环,也就是 idle 进程代码。idle 进程的 PID 为 0, idle 进程叫做空闲进程,如果学过 FreeRTOS 或者 UCOS 的话应该听说过空闲任务。idle 空闲 进程就和空闲任务一样,当 CPU 没有事情做的时候就在 idle 空闲进程里面"瞎逛游",反正 就是给 CPU 找点事做。当其他进程要工作的时候就会抢占 idle 进程,从而夺取 CPU 使用权。其实可以看到 idle 进程并没有使用 kernel_thread 或者 fork 函数来创建,因为它是由主进程演 变而来的。

在 Linux 终端中输入 "ps -A" 就可以打印出当前系统中的所有进程,其中就能看到 kthreadd 进程,如下图所示:

1	?	00:00:02	systemd	
2	?	00:00:00	kthreadd	
3	?	00:00:00	rcu_gp	
4	?	00:00:00	rcu_par_gp	
б	?	00:00:00	kworker/0:0H	l-kb
8	?	00:00:00	mm_percpu_wo	1
9	?	00:00:00	ksoftirqd/0	

图 17.2.1 Linux 系统当前进程

从上图可以看出, kthreadd 进程的 PID 为 2。之所以上图中没有显示 PID 为 0 的 idle 进程, 那是因为 idle 进程是内核进程。我们接下来重点看一下 init 进程, kernel_init 就是 init 进程的进程函数。

17.2.5 init 进程

kernel_init 函数就是 init 进程具体做的工作,定义在文件 init/main.c 中,函数内容如下: 示例代码 17.2.6 kernel_init 函数

```
1105 static int __ref kernel_init(void *unused)
1106 {
1107 int ret;
1108
1109 kernel_init_freeable();
1110 /* need to finish all async __init code before freeing the memory */
1111 async_synchronize_full();
1112 ftrace_free_init_mem();
1113 free_initmem();
1114 mark_readonly();
1115
1116 /*
1117 * Kernel mappings are now finalized - update the userspace page-table
1118 * to finalize PTI.
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php 1119 */ 1120 pti_finalize(); 1121 1122 system_state = SYSTEM_RUNNING; 1123 numa_default_policy(); 1124 1125 rcu_end_inkernel_boot(); 1126 1127 if (ramdisk_execute_command) { ret = run_init_process(ramdisk_execute_command); 1128 1129 if (!ret) 1130 return 0; 1131 pr_err("Failed to execute %s (error %d)\n", 1132 ramdisk_execute_command, ret); 1133 } 1134 1135 /* * We try each of these until one succeeds. 1136 1137 1138 * The Bourne shell can be used instead of init if we are * trying to recover a really broken machine. 1139 1140 */ 1141 if (execute_command) { 1142 ret = run_init_process(execute_command); 1143 if (!ret) 1144 return 0; 1145 panic("Requested init %s failed (error %d).", execute_command, ret); 1146 1147 } 1148 **if** (!try_to_run_init_process("/sbin/init") || !try_to_run_init_process("/etc/init") || 1149 !try_to_run_init_process("/bin/init") || 1150 !try_to_run_init_process("/bin/sh")) 1151 1152 return 0; 1153 1154 panic("No working init found. Try passing init= option to kernel. " 1155 "See Linux Documentation/admin-guide/init.rst for guidance."); 1156 } 第 1109 行, kernel_init_freeable 函数用于完成 init 进程的一些其他初始化工作, 稍后再来

具体看一下此函数。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

第 1127 行, ramdisk_execute_command 是一个全局的 char 指针变量,此变量值为"/init",也就是根目录下的 init 程序。ramdisk_execute_command 也可以通过 uboot 传递,在 bootargs 中使用"rdinit=xxx"即可, xxx 为具体的 init 程序名字。

第1128行,如果存在"/init"程序的话就通过函数 run_init_process 来运行此程序。

第 1141 行,如果 ramdisk_execute_command 为空的话就看 execute_command 是否为空,反正不管如何一定要在根文件系统中找到一个可运行的 init 程序。execute_command 的值是通过 uboot 传递,在 bootargs 中使用 "init=xxxx" 就可以了,比如 "init=/linuxrc" 表示根文件系统中的 linuxrc 就是要执行的用户空间 init 程序。

第 1148~1151 行,如果 ramdisk_execute_command 和 execute_command 都为空,那么就依次查找 "/sbin/init"、 "/etc/init"、 "/bin/init"和 "/bin/sh",这四个相当于备用 init 程序,如果这四个也不存在,那么 Linux 启动失败。

第1154行,如果以上步骤都没有找到用户空间的 init 程序,那么就提示错误发生。

最后来简单看一下 kernel_init_freeable 函数,前面说了,kernel_init 会调用此函数来做一些 init 进程初始化工作。kernel_init_freeable 定义在文件 init/main.c 中,缩减后的函数内容如下:

示例代码 17.2.7 kernel_init_freeable 函数

```
1158 static noinline void __init kernel_init_freeable(void)
1159 {
1160 /*
1161 * Wait until kthreadd is all set-up.
1162
       */
1163 wait_for_completion(&kthreadd_done);
1183
1184 smp init();
                      //SMP 初始化
1185 sched_init_smp(); //多核(SMP)调度初始化
1186
      page_alloc_init_late();
1187
1188 /* Initialize page ext after all struct pages are initialized. */
1189
      page_ext_init();
1190
1191 do_basic_setup(); //设备初始化都在此函数中完成
1192
1193 /* Open the /dev/console on the rootfs, this should never fail */
      if (ksys_open((const char __user *) "/dev/console", O_RDWR, 0) < 0)
1194
1195
         pr_err("Warning: unable to open an initial console.\n");
1196
1197 (void) ksys_dup(0);
1198 (void) ksys_dup(0);
1199 /*
1200 * check if there is an early userspace init. If yes, let it do all
```



领航者	ZYNQ 之嵌入式 Linux 开	发指南	② 正点原
原子哥在	线教学: www.yuanzige.com	论坛:www.openedv.com/f	forum.php
1201	* the work		
1202	*/		
1203			
1204	if (!ramdisk_execute_command)		
1205	ramdisk_execute_command = "/init";		
1206			
1207	<pre>if (ksys_access((const charuser *)</pre>		
1208	ramdisk_execute_command, 0) != 0	0) {	
1209	ramdisk_execute_command = NULL	;	
1210	prepare_namespace();		
1211	}		
1212			
1213	/*		
1214	* Ok, we have completed the initial boo	otup, and	
1215	* we're essentially up and running. Get	rid of the	

* initmem segments and start the user-mode stuff..

* rootfs is available now, try loading the public keys

* and default modules

1222 integrity_load_keys();

1216

1217

1218

1219 1220

1221

1223

*/

第 1191 行, do_basic_setup 函数用于完成 Linux 下设备驱动初始化工作。do_basic_setup 会调用 driver init 函数完成 Linux 下驱动模型子系统的初始化。

第 1194 行,打开设备"/dev/console",在 Linux 中一切皆为文件。因此"/dev/console" 也是一个文件,此文件为控制台设备。每个文件都有一个文件描述符,此处打开的 "/dev/console" 文件描述符为 0, 作为标准输入(0)。

第 1197 和 1198 行, ksys_dup 函数将标准输入(0)的文件描述符复制了 2 次, 一个作为标 准输出(1),一个作为标准错误(2)。这样标准输入、输出、错误都是/dev/console 了。console 通过 uboot 的 bootargs 环境变量设置, "console=ttyPS0,115200"表示将/dev/ ttyPS0 设置为 console,也就是 ZYNQ 的串口 0。当然,也可以设置其他的设备为 console,比如虚拟控制台 tty1,设置 tty1为 console 就可以在 LCD 屏幕上看到系统的提示信息。

第1210行,调用函数 prepare_namespace 来挂载根文件系统。跟文件系统也是由命令行参 数指定的,也就是 uboot 的 bootargs 环境变量。比如 "root=/dev/mmcblk1p2 rootwait rw"就表 示根文件系统在/dev/mmcblk1p2中,也就是EMMC的分区2中。

Linux 内核启动流程就分析到这里, Linux 内核最终是需要和根文件系统打交道的, 需要 挂载根文件系统,并且执行根文件系统中的init程序,以此来进入用户态。这里就正式引出了 根文件系统,根文件系统也是我们系统移植的最后一片拼图。Linux 移植三巨头: uboot、 Linux kernel、rootfs(根文件系统)。关于根文件系统后面章节会详细的讲解,这里我们只需要 知道 Linux 内核移植完成以后还需要构建根文件系统即可。

原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php

正点原子

第十八章 Linux 内核移植

前两章我们简单了解了一下 Linux 内核顶层 Makefile 和 Linux 内核的启动流程,本章我们 就来学习一下如何将 Xilinx 官方提供的 Linux 内核移植到正点原子的领航者开发板上。需要说 明的是当我们使用 Petalinux 工具的时候是不需要移植内核的,因为 Petalinux 工具可以根据硬 件描述文件 system_wrapper.xsa 使能相应配置。本章讲解内核移植(更准确的说是内核适配) 是为了了解一般情况下(不使用 Petalinux)的内核移植过程。通过本章的学习,我们将掌握 如何将半导体厂商提供的 Linux BSP 包移植到我们自己的平台上。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

18.1 Xilinx 官方开发板 Linux 内核编译

本章的移植我们使用 Xilinx 提供的 Linux 源码,将其移植到正点原子领航者开发板上。 Xilinx 提供的 Linux 源码已经放到了开发板光盘中,路径为: ZYNQ 开发板资料盘(A 盘)\4_SourceCode\3_Embedded_Linux\资源文件\kernel\linux-xlnx-xlnx_rebase_v5.4_2020.2.tar.gz。 将其发送到 Ubuntu 中并解压到名为~/petalinux/atk-linux 的目录(没有该目录可创建,位置任 意)下。若读者想从官网下载 linux 源码,可以从该网址 <u>https://github.com/Xilinx/linux-xlnx/tags</u>下载 linux 源码,注意 linux 源码的版本要正点原子提供的 linux 版本一样,这样在后 期遇到问题后可以参考我们的文档解决。正点原子下载的 linux 源码版本如下图所示:

Co Tags · Xilinx/linux-xlnx · GitHub x +					
← C ↔ https://github.com/Xilinx/linux-xlnx/tags?after=zynqmp-soc-for-v5.12					
<> Code 11 Pull requests 2 ② Actions 🕀 Projects ① Security 🗠 Insights					
Releases Tags					
🖏 Tags					
zynqmp-dt-for-v5.12 🚥					
() on Feb 3, 2021 ↔ 5556339 []) zip []] tar.gz					
zynqmp-soc-for-v5.11 🚥					
⊙ on Dec 9, 2020 •					
xlnx_rebase_v5.4_2020.2 🚥					
🕐 on Nov 23, 2020 🗢 62ea514 👔 zip 👔 tar.gz					

图 18.1.1 linux 源码下载

Xilinx 提供的 Linux 源码肯定可以在 Xilinx 的 ZYNQ EVK(Evaluation Kit)开发板上运行的,我们以 ZYNQ EVK 开发板为参考,然后将 Linux 内核移植到我们的领航者开发板上。

18.1.1 修改顶层 Makefile

修改顶层 Makefile,直接在顶层 Makefile 文件里面定义 ARCH 和 CROSS_COMPILE 这两个的变量值为 arm 和 arm-xilinx-linux-gnueabi-,结果如下图所示:



图 18.1.2 修改顶层 Makefile

先注释掉原先的设置,方便后面我们复原,然后分别设置 ARCH 和 CROSS_COMPILE 这两个变量的值,这样在编译的时候就不用输入很长的命令了。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

18.1.2 配置并编译 Linux 内核

和 uboot 一样,在编译 Linux 内核之前要先配置 Linux 内核。每个板子都有其对应的默认 配置文件,这些默认配置文件保存在 arch/arm/configs 目录中。xilinx_zynq_defconfig 作为 ZYNQ EVK 开发板所使用的默认配置文件。

进入到 Ubuntu 中的 Linux 源码根目录下,执行如下命令配置 Linux 内核: //第一次编译 Linux 内核之前先清理一下 make clean make xilinx_zynq_defconfig //配置 Linux 内核 配置完成以后如下图所示: q@Linux:~/petalinux/atk-linux/linux-xlnx_rebase_v5.4_2020.2\$ make xilinx_zynq_defconfig HOSTCC scripts/basic/fixdep HOSTCC scripts/kconfig/conf.o HOSTCC scripts/kconfig/confdata.o HOSTCC scripts/kconfig/expr.o LEX scripts/kconfig/exer.lex.c YACC scripts/kconfig/parser.tab.[ch] HOSTCC scripts/kconfig/lexer.lex.o HOSTCC scripts/kconfig/parser.tab.o scripts/kconfig/preprocess.o HOSTCC scripts/kconfig/symbol.o HOSTCC HOSTLD scripts/kconfig/conf configuration written to .config q@Linux:~/petalinux/atk-linux/linux-xlnx-xlnx_rebase_v5.4_2020.2\$ 图 18.1.3 配置 Linux 内核 配置完成以后就可以编译了,使用如下命令编译 Linux 内核: //编译 Linux 内核 make -j8 等待编译完成,结果如下图所示: SHIPPED arch/arm/boot/compressed/hyp-stub.S SHIPPED arch/arm/boot/compressed/lib1funcs.S CC arch/arm/boot/compressed/string.o SHIPPED arch/arm/boot/compressed/ashldi3.S SHIPPED arch/arm/boot/compressed/bswapsdi2.S arch/arm/boot/compressed/hyp-stub.o AS arch/arm/boot/compressed/lib1funcs.o AS arch/arm/boot/compressed/ashldi3.o AS arch/arm/boot/compressed/bswapsdi2.o AS arch/arm/boot/compressed/piggy.o AS arch/arm/boot/compressed/vmlinux LD OBJCOPY arch/arm/boot/zImage Kernel: arch/arm/boot/zImage is ready wmq@Linux:~/petalinux/atk-linux/linux-xlnx-xlnx_rebase_v5.4_2020.2\$

图 18.1.4 Linux 编译完成

Linux 内核编译完成以后会在 arch/arm/boot 目录下生成 zImage 镜像文件,如果使用设备 树的话还会在 arch/arm/boot/dts 目录下生成开发板对应的.dtb(设备树)文件,比如 zynq-zc702.dtb 就是 Xilinx 官方的 ZYNQ ZC702 EVK 开发板对应的设备树文件。至此我们得到两个 文件:

① Linux 内核镜像文件: zImage。

② Xilinx 官方 ZYNQ ZC702 EVK 开发板对应的设备树文件: zynq-zc702.dtb。


原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

18.1.3 Linux 内核启动测试

在上一小节我们已经得到了 Xilinx 官方 ZYNQ ZC702 EVK 开发板对应的 zImage 和 zynqzc702.dtb 这两个文件。这两个文件能不能在正点原子的领航者开发板上启动呢?我们得测试 一下,在测试之前需要先设置 uboot 中的环境变量 bootargs,设置命令如下:

setenv bootargs 'console= ttyPS0, 115200'

将上一小节编译出来的 zImage 和 zynq-zc702.dtb 复制到 Ubuntu 中的/tftpboot 目录下,因为我们要在 uboot 中使用 tftpboot 命令将其下载到开发板中,拷贝命令如下:

cp arch/arm/boot/zImage /tftpboot/ -f

cp arch/arm/boot/dts/zynq-zc702.dtb /tftpboot/ -f

拷贝完成以后就可以测试了,启动开发板,进入uboot命令行模式,然后输入如下命令将 zImage 和 zynq-zc702.dtb 下载到开发板中并启动:

tftpboot 8000 zImage

tftpboot 103cdda8 zynq-zc702.dtb

bootz 8000 - 103cdda8

结果下图所示:



图 18.1.5 启动 Linux 内核

从上图可以看出,此时 Linux 内核启动停止输出信息了,这是因为 Xilinx 官方 ZYNQ ZC702 EVK 开发板使用的输出串口是 UART1,而我们领航者使用的是 UART0。

根据读者 Road 反馈,使用上面的地址没有显示"Starting kernel",而是显示 axi_gpio <addr:地址>,解决方法是将下载到 DDR 中的地址更换成如下地址即可:

tftpboot 00000000 zImage tftpboot 05000000 zynq-zc702.dtb bootz 00000000 - 05000000



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

18.2 在 Linux 中添加自己的开发板

在上一节中我们通过编译 Xilinx 官方 ZYNQ EVK 开发板对应的 Linux 内核,发现其不可以在正点原子的 ZYNQ 开发板上启动,所以本节我们就参考 Xilinx EVK 开发板的设置,修改相应的配置使其能够在正点原子的领航者开发板上启动。

18.2.1 添加开发板默认配置文件

将 arch/arm/configs 目录下的 xilinx_zynq_defconfig 重新复制一份,命名为 alientek_zynq_defconfig,命令如下:

cd arch/arm/configs

cp xilinx_zynq_defconfig alientek_zynq_defconfig

以后 alientek_zynq_defconfig 就是正点原子的领航者开发板默认配置文件了,可以使用如下命令来配置正点原子领航者开发板对应的 Linux 内核:

make alientek_zynq_defconfig

18.2.2 添加开发板对应的设备树文件

Linux 支持设备树,每个开发板都有一个对应的设备树文件。Xilinx 的 ZYNQ 系列芯片的 所有设备树文件夹都存放在 arch/arm/boot/dts 目录下,在这个目录下有个名为 zynq-zc702.dts 的文件,该文件是 ZC702 开发板的设备树文件。这里我们就不参照 zynq-zc702.dts 文件,而是 参照 zynq-zed.dts 文件,这是因为 zynq-zed.dts 是在 zynq-zc702.dts 文件基础上修改而来,能极 大的方便我们的移植。我们将 zynq-zed.dts 重命名为 zynq-alientek.dts,命令如下:

cd arch/arm/boot/dts

cp zynq-zed.dts zynq-alientek.dts

```
.dts 是设备树源码文件,编译 Linux 的时候会将其编译为.dtb 文件。
```

拷贝完成以后修改 zynq-alientek.dts 文件的内容。修改部分内容如下所示:

9 / {

```
10 model = "Zynq Alientek Development Board";
```

```
11 compatible = "xlnx,zynq-alientek", "xlnx,zynq-7000";
```

12

```
13 aliases {
```

```
14 ethernet0 = \&gem0;
```

```
15 serial0 = \&uart0;
```

```
16 spi0 = \&qspi;
```

```
17 mmc0 = \& sdhci0;
```

```
18 };
```

19

```
20 memory@0 {
```

```
21 device_type = "memory";
```

```
22 reg = \langle 0x0 \ 0x40000000 \rangle;
```

```
23 };
```

24



```
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
    25 chosen {
    26
          bootargs = "";
          stdout-path = "serial0:115200n8";
    27
    28 };
    29
    30 usb_phy0: phy0@e0002000 {
    31
        compatible = "ulpi-phy";
    32
        \#phy-cells = <0>;
    33
        reg = \langle 0xe0002000 \ 0x1000 \rangle;
    34
         view-port = \langle 0x0170 \rangle;
    35
         drv-vbus;
    36 };
    37 };
    38
    39 &clkc {
    40 ps-clk-frequency = <33333333>;
    41 };
    42
    43 & gem0 {
    44 status = "okay";
    45 phy-mode = "rgmii-id";
    46 phy-handle = <&ethernet_phy>;
    47
    48 ethernet_phy: ethernet-phy@0 {
    49
         reg = <0>;
    50
         device_type = "ethernet-phy";
    51 };
    52 };
    53
    54 &qspi {
    55 u-boot,dm-pre-reloc;
    56 status = "okay";
    57 is-dual = <0>;
    58 num-cs = <1>;
    59 flash@0 {
    60
         compatible = "spansion,s25fl256s1", "jedec,spi-nor";
    61
         reg = <0>;
         spi-tx-bus-width = <1>;
    62
          spi-rx-bus-width = <4>;
    63
    64
          spi-max-frequency = <50000000>;
```

```
65 m25p,fast-read;
```



```
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
    66
          #address-cells = <1>;
    67
          \#size-cells = <1>;
          partition@0 {
    68
    69
            label = "qspi-fsbl-uboot";
    70
            reg = \langle 0x0 \ 0x100000 \rangle;
    71
          };
    72
          partition@100000 {
    73
            label = "qspi-linux";
    74
            reg = <0x100000 0x500000>;
    75
          };
    76
          partition@600000 {
    77
            label = "qspi-device-tree";
            reg = <0x600000 0x20000>;
    78
    79
          };
          partition@620000 {
    80
    81
            label = "qspi-rootfs";
            reg = <0x620000 0x5E0000>;
    82
    83
          };
    84
          partition@c00000 {
    85
           label = "qspi-bitstream";
            reg = <0xC00000 0x400000>;
    86
    87
        };
    88 };
    89 };
    90
    91 &sdhci0 {
    92 u-boot,dm-pre-reloc;
    93 status = "okay";
    94 };
    95
    96 &uart0 {
    97 u-boot,dm-pre-reloc;
    98 status = "okay";
    <mark>99</mark> };
    100
    101 &usb0 {
    102 status = "okay";
    103 dr_mode = "host";
    104 usb-phy = <&usb_phy0>;
    105 };
```



原子哥在线教学: www.yuanzige.com 论坛:www.op

论坛:www.openedv.com/forum.php

首先将 15 行和 96 行的 uart1 改为 uart0, 然后根据自己板子 DDR 的大小修改 22 行的数 值, 若你的板子是领航者 7010, 则应修改为 reg = <0x0 0x20000000>;若你的板子是领航者 7020, 则应修改为 reg = <0x0 0x4000000>;

需要注意的是这里的 zynq-alientek.dts 文件只是配置了 ZYNQ 的 PS 端,使 linux 内核能够 启动,PL 端并没有配置,如何配置 PL 端可以进入第八章的 Petalinux 工程,在工程的 "components\plnx_workspace\device-tree\device-tree" 目录下有很多 dts 文件,可以参考该目录 下的 system-top.dts 文件(特别是该文件所 include 的文件)来配置 zynq-alientek.dts 文件。

zynq-alientek.dts 修改好以后我们还需要修改文件 arch/arm/boot/dts/Makefile, 找到"dtb-\$(CONFIG_ARCH_ZYNQ)"配置项,在此配置项中加入"zynq-alientek.dtb\",如下所示:

1201 dtb-\$(CONFIG_ARCH_ZYNQ) += \

- 1202 zynq-cc108.dtb $\$
- $1203 \qquad zynq-microzed.dtb \ \backslash$
- $1204 \qquad zynq-parallella.dtb \ \backslash$
- 1205 zynq-zc702.dtb \setminus
- $1206 \qquad zynq-alientek.dtb \ \backslash$
- $1207 \qquad zynq-zc706.dtb \ \backslash$

- 1210 zynq-zc770-xm012.dtb \setminus
- 1211 zynq-zc770-xm013.dtb $\$
- 1212 zynq-zed.dtb \setminus
- 1213 zynq-zybo.dt

在第 1205 行的 "zynq-zc702.dtb \"下方添加 "zynq-alientek.dtb \",这样编译 Linux 的时 候就可以从 zynq-alientek.dts 编译出 zynq-alientek.dtb 文件了。

额外提一下,如果后面测试的时候只修改设备树的话可以只重新编译设备树,在 Linux 内核源码根目录下输入如下命令编译设备树:

make dtbs

命令"make dtbs"只编译设备树文件,也就是将.dts 编译为.dtb,编译完成以后就可以使用新的设备树文件。

18.2.3 编译测试

经过前两个小节,我们已经在 Linux 内核里面已经添加了正点原子领航者开发板的配置, 接下接进行编译测试。我们可以创建一个名为 zynq.sh 的编译脚本,脚本内容如下:

1 #**!/bin/sh**

2 make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi- distclean

3 make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi- alientek_zynq_defconfig

4 make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi- menuconfig

5 make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi- all -j8

第2行,清理工程。

第3行,使用默认配置文件 alientek_zynq_defconfig 来配置 Linux 内核。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

第 4 行,打开 Linux 的图形配置界面,如果不需要每次都打开图形配置界面可以删除此行。

第5行,编译Linux。

执行 shell 脚本 zynq.sh 编译 Linux 内核前,先使用 chmod 命令给 zynq.sh 可执行权限,然 后才可以使用 zynq.sh 编译 Linux 内核,编译完成以后就会在目录 arch/arm/boot 下生成 zImage 镜像文件,在 arch/arm/boot/dts 目录下生成 zynq-alientek.dtb 文件。将这两个文件拷贝到 /tftpboot 目录下,然后参考 20.6.1 章节配置虚拟机与开发板的网络环境,然后在 uboot 命令模 式中使用 tftpboot 命令下载这两个文件并启动,命令如下:

tftpboot 8000 zImage

tftpboot 103cdda8 zynq-alientek.dtb

bootz 8000 - 103cdda8

只要出现如下图所示内容就表示 Linux 内核启动成功:

Starting kernel ...

Booting Linux on physical CPU 0x0 Linux version 5.4.0-xilinx (wmq@Linux) (gcc version 9.2.0 (GCC)) #1 SMP PREEMPT Mon Aug 7 13:17:14 CST 2023 CPU: ARMv7 Processor [413fc090] revision 0 (ARMv7), cr=18c5387d CPU: PIPT / VIPT nonaliasing data cache, VIPT aliasing instruction cache OF: fdt: Machine model: Zynq Alientek Development board Memory policy: Data cache writealloc

图 18.2.1 Linux 内核启动

如果上面的地址启动有问题,可以使用下面的地址:

tftpboot 00000000 zImage

tftpboot 05000000 zynq-alientek.dtb

bootz 00000000 - 05000000

Linux 内核启动成功,说明我们已经在 Xilinx 提供的 Linux 内核源码中成功添加了正点原 子领航者开发板。

18.2.4 根文件系统缺失错误

Linux 内核启动以后是需要根文件系统的,根文件系统存在哪里是由 uboot 或设备树文件 的 bootargs 环境变量指定,bootargs 会传递给 Linux 内核作为命令行参数。比如设置 root=/dev/mmcblk0p2,也就是说根文件系统存储在/dev/mmcblk0p2 中,也就是 SD 卡的分区 2 中。我们在 6.3.10 节制作 SD 启动卡时将 SD 卡分为两个分区,分区 2 就是用来存放根文件系统的,所以设置 root=/dev/mmcblk0p2。如果我们不设置根文件系统路径,或者说根文件系统 路径设置错误的话会出现什么问题?这个问题是很常见的,我们在实际的工作中开发一个产品,这个产品的第一版硬件出来以后我们是没有对应的根文件系统可用的,必须要自己做根 文件系统。在构建出对应的根文件系统之前 Linux 内核是没有根文件系统可用的,此时 Linux 内核启动以后会出现什么问题呢?带着这个问题,我们将 bootargs 环境变量改为 "console= ttyPS0,115200",也就是不填写 root 的内容了,命令如下:

setenv bootargs 'console= ttyPS0, 115200'

修改完成以后重新从网络启动,启动以后会有如下图所示错误:



原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php

Kernel panic - not syncing: VFS: Unable to mount root fs on unknown-block(0,0) CPU0: stopping CPU: 0 PID: 0 Comm: swapper/0 Not tainted 5.4.0-xilinx #1 Hardware name: Xilinx Zynq Platform [<c010e35c>] (unwind backtrace) from [<c010allc>] (show stack+0x10/0x14) [<c010allc>] (show stack) from [<c066ab04>] (dump stack+0xb4/0xd0) [<c066ab04>] (dump_stack) from [<c010c944>] (ipi_cpu_stop+0x3c/0x98) [<c010c944>] (ipi_cpu_stop) from [<c010d190>] (handle_IPI+0x64/0x80) [<c010dl90>] (handle_IPI) from [<c03128a8>] (gic_handle_irq+0x84/0x90) [<c03128a8>] (gic handle_irq) from [<c0101a8c>] (__irq_svc+0x6c/0xa8) Exception stack(0xc0a0lee0 to 0xc0a01f28) lee0: 00000000 00000000 le28e000 debcfl40 c0a284fc 00000000 debce538 00000000 1f00: 4cdf17b9 4ce9928b 00000000 00000000 ffffffff5 c0a01f30 c04fa90c c04fa930 1f20: 60000013 ffffffff [<c0101a8c>] (__irq_svc) from [<c04fa930>] (cpuidle_enter_state+0xec/0x288) [<c04fa930>] (cpuidle_enter_state) from [<c04fab08>] (cpuidle_enter+0x28/0x38) [<c04fab08>] (cpuidle_enter) from [<c013eafc>] (do idle+0x230/0x258) [<c013eafc>] (do_idle) from [<c013ec88>] (cpu_startup_entry+0x18/0x1c) [<c013ec88>] (cpu_startup_entry) from [<c0900c9c>] (start_kernel+0x384/0x420) [<c0900c9c>] (start_kernel) from [<00000000>] (0x0) -[end Kernel panic - not syncing: VFS: Unable to mount root fs on unknown-block(0,0)

图 18.2.2 根文件系统缺失错误

在上图中最后会有下面这一行:

end Kernel panic - not syncing: VFS: Unable to mount root fs on unknown-block(0,0)

也就是提示内核崩溃,因为根文件系统目录不存在,VFS(虚拟文件系统)不能挂载根文件 系统。即使根文件系统目录存在,如果根文件系统目录里面是空的依旧会提示内核崩溃。这 个就是根文件系统缺失导致的内核崩溃,但是内核是启动了的,只是根文件系统不存在而已。

18.3 **image.ub** 的来源

从 16.5.5 节 make zImage 过程我们知道,编译 linux 内核后得到的是 vmlinux、Image 和 zImage 镜像文件,而我们使用 Petalinux 启动 linux 的时候使用的是 image.ub 镜像文件,那么 image.ub 文件是怎么产生的呢,是否一定需要 image.ub 文件才能启动内核呢?

由于我们编译内核并没有生成 image.ub 镜像文件,显然不是由内核编译产生的。在 uboot 移植章节的从网络启动 Linux 小节中我们只使用了 image.ub 文件结果是可以正常启动 linux, 说明 image.ub 文件一定是包含 zImage 文件和设备树文件信息的。接下来,我们来探索下 image.ub 的来源。

既然 image.ub 文件包含 zImage 文件和设备树文件信息,那么 image.ub 文件一定是由某个 工具打包制作而得到的,常用的镜像制作工具是 mkimage, mkimage 工具位于 uboot 源码根目 录下的/tools 目录下,初次使用前,要把 mkimage 文件复制到系统可执行命令文件夹内(也就 是/usr/bin/),如下图所示:

<pre>wmq@Linux:~/petalinux/uboot/alientek-uboot-v2020.1/tools\$</pre>	sudo	cp mkimage	/usr/bin/
[sudo] wmq 的密码:			
wmq@Linux:~/petalinux/uboot/alientek-uboot-v2020.1/tools\$			
<pre>wmq@Linux:~/petalinux/uboot/alientek-uboot-v2020.1/tools\$</pre>			

图 18.3.1 复制 mkimage 文件到可执行命令文件夹内

接下来我们使用 mkimage 来分析 image.ub 文件, image.ub 文件在第六章 petalinux 工程目录下有,所以我们将路径切到第六章的 petalinux 工程目录下,笔者的 petalinux 工程目录为: /home/wmq/petalinux/ALIENTEK-ZYNQ。然后使用命令 mkimage -l images/linux/image.ub 来分析 image.ub 文件,分析结果如下图 18.3.2 所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php wmq@Linux:~/petalinux/ALIENTEK-ZYNQ\$ mkimage -l images/linux/image.ub

FIT description: U-Boot fitImage for PetaLinux/5.4+git999/zynq-generic Wed Jun 28 10:21:53 2023 Created: Image 0 (kernel@1) Description: Linux kernel Wed Jun 28 10:21:53 2023 Created: Kernel Image Type: Compression: uncompressed Data Size: 4325984 Bytes = 4224.59 KiB = 4.13 MiB Architecture: ARM **OS:** Linux Load Address: 0x00200000 Entry Point: 0x00200000 Hash algo: sha256 Hash value: _____d614711fae4f373ff4181b15c184847e80ff534bfa5fb9bcb1ef7941191086dd Image 1 (fdt@system-top.dtb) Description: Flattened Device Tree blob Wed Jun 28 10:21:53 2023 Created: Flat Device Tree Type: Compression: uncompressed 28135 Bytes = 27.48 KiB = 0.03 MiB Data Size: Architecture: ARM Hash algo: sha256 Hash value: af393ddd932718bb09909acc47bb74368ad384947b090f742d457ce62b5c2948 Image 2 (ramdisk@1) Description: petalinux-image-minimal Created: Wed Jun 28 10:21:53 2023 RAMDisk Image Type: Compression: uncompressed Data Size: 7482599 Bytes = 7307.23 KiB = 7.14 MiB Architecture: ARM 0S: Linux Load Address: unavailable Entry Point: unavailable Hash algo: sha256 4e7b95e8c7b817bc6bec7b640b627576d25b1fbb0ebe9735b364518dbd406bca Hash value: Default Configuration: 'conf@system-top.dtb' Configuration 0 (conf@system-top.dtb) Description: 1 Linux kernel, FDT blob, ramdisk Kernel: kernel@1 Init Ramdisk: ramdisk@1 fdt@system-top.dtb FDT:

图 18.3.2 分析 image. ub 文件

从显示的内容来看, image.ub 确实是包括 linux 内核镜像和设备树的, 另外从 "FIT description"的信息来看, image.ub 文件是 U-Boot fitImage。现在我们根据 "FIT description"的信息来搜索哪个文件包含该内容, 搜索结果如下图所示:



原于哥任线教字: www.yuanzige.com	论坛:www.openedv.com/forum.php
<pre>wmq@Linux:~/petalinux/ALIENTEK-ZYNQ\$ grep -irnI</pre>	"U-Boot fitImage for PetaLinux/5.4+git999/zynq-generic"
build/tmp/work/zynq_generic-xilinx-linux-gnueab	i/linux-xlnx/5.4+git999-r0/deploy-linux-xlnx/fitImage-it
s5.4+git999-r0-zynq-generic-20230628021954.it	s:4: description = "U-Boot fitImage for PetaLinux
/5.4+git999/zynq-generic";	
build/tmp/work/zynq_generic-xilinx-linux-gnueab	l/linux-xinx/5.4+git999-r0/depioy-linux-xinx/fitimage-it
s-petalinux-image-minimal-zynq-generic5.4+git	999-r0-zynq-generic-20230628021954.its:4: descrip
tion = "U-Boot Titimage for Petalinux/S.4+git99	<pre>9/zynq-generic"; i/liawy wlaw/f_4.cit000_c0/tame/swa_da_accomble_fitimace</pre>
initeration and a second	c/ttnux-xtnx/5.4+gtt999-r0/temp/run.do_assemble_rtttmage
	i/lipux xlpx/E 4.git000 c0/tomo/log do percemble fitimped
initramfs 12395:21:ETT description: IL-Root fit	Trace for Petalinux/5 4+git999-10/temp/tog.do_assemble_fittimage
	i/lipux_xlpx/5_4+git000_r0/temp/log_do_assemble_fitimage
33187:14: FIT description: U-Boot fitTmage for	Detalioux/5 4+git999/zvng-generic
huild/tmp/work/zvpg_generic_vilipy_lipux_gnuesh	i/lipuy-ylpy/5_4+git999-r0/temp/log_do_assemble_fitimage
initramfs. 31497:30:FIT description: U-Boot fit	Tmage for Petalinux/5.4+git999/zvng-generic
build/tmp/work/zvng generic-xilinx-linux-gnueab	i/linux-xlnx/5.4+git999-r0/temp/log.do assemble fitimage
.14595:29:FIT description: U-Boot fitImage for	PetaLinux/5.4+git999/zvng-generic
build/tmp/work/zvng generic-xilinx-linux-gnueab	i/linux-xlnx/5.4+git999-r0/temp/run.do assemble fitimage
.33187:324: description = "U-Boot fitIma	ge for PetaLinux/5.4+git999/zvng-generic":
build/tmp/work/zynq generic-xilinx-linux-gnueab	i/linux-xlnx/5.4+git999-r0/temp/run.do assemble fitimage
_initramfs.22666:325: description = "U-B	oot fitImage for PetaLinux/5.4+git999/zynq-generic";

图 18.3.3 搜索结果

可见包含该内容的文件不少。大致的看了一下以上文件,以".its"结尾的文件是由 run.do_assemble_fitimage_initramfs*文件生成的。我们打开 build/tmp/work/zynq_generic-xilinxlinux-gnueabi/linux-xlnx/5.4+git999-r0/deploy-linux-xlnx/fitImage-its--5.4+git999-r0-zynq-generic-20230628021954.its 文件,内容如下:

```
1 /dts-v1/;
    2
    3 / {
    4
            description = "U-Boot fitImage for PetaLinux/5.4+git999/zynq-generic";
     5
            #address-cells = <1>;
    6
    7
            images {
    8
                 kernel@1 {
    9
                      description = "Linux kernel";
     10
                      data = /incbin/("linux.bin");
                      type = "kernel";
     11
                      arch = "arm";
     12
     13
                      os = "linux";
                      compression = "none";
     14
     15
                      load = \langle 0x200000 \rangle;
                      entry = <0x200000>;
     16
     17
                      hash@1 {
                           algo = "sha256";
     18
     19
                      };
    20
                 };
                  fdt@system-top.dtb {
    21
     22
                      description = "Flattened Device Tree blob";
     23
                      data = /incbin/("/home/wmq/petalinux/ALIENTEK-ZYNQ/build/tmp/work/zynq_generic-
xilinx-linux-gnueabi/linux-xlxn/5.4+git999-r0/recipe-sysroot/boot/devicetree/system-top.dtb");
```



```
原子哥在线教学: www.yuanzige.com
                                                论坛:www.openedv.com/forum.php
    24
                    type = "flat_dt";
    25
                    arch = "arm";
                    compression = "none";
    26
    27
                    hash@1 {
    28
                         algo = "sha256";
    29
                    };
    30
                };
    31
         };
    32
    33
           configurations {
    34
                default = "conf@system-top.dtb";
    35
                conf@system-top.dtb {
                  description = "1 Linux kernel, FDT blob";
    36
                  kernel = "kernel@1";
    37
                  fdt = "fdt@system-top.dtb";
    38
    39
    40
    41
                    hash@1 {
    42
                         algo = "sha256";
    43
                    };
    44
                };
    45
          };
    46 }
```

可以看出该文件的内容与我们使用"mkimage -l images/linux/image.ub"得到的信息基本 一致。另外,从该文件内容可以看到镜像的 kernel 来源于 linux.bin,而设备树来源于工程目录 下 的 build/tmp/work/zynq_generic-xilinx-linux-gnueabi/linux-xlnx/5.4+git999-r0/recipesysroot/boot/devicetree/system-top.dtb 。 从 build/tmp/work/zynq_generic-xilinx-linuxgnueabi/linux-xlnx/5.4+git999-r0 /temp/run.do_assemble_fitimage_initramfs.12395 文件可知 linux.bin 是由 vmlinux 生成的。其实我们也可以用 zImage 来代替 linux.bin,下面我们使用 zImage 和 system-top.dtb 生成 imagde.ub。

将我们打开的.its 文件另存为 fitimage.its 并将其放到 linux 源码根目录下,也就是 "~/petalinux/atk-linux/linux-xlnx_reabase_v5.4_2020.2"目录下,然后修改相应的内容, 修改后的 fitimage.its 文件内容如下:

```
1 /dts-v1/;
2
3 / {
4     description = "U-Boot fitImage for Alientek ZYNQ ";
5     #address-cells = <1>;
6
7     images {
8         kernel-1 {
```



```
原子哥在线教学: www.yuanzige.com
                                                   论坛:www.openedv.com/forum.php
    9
                     description = "Linux kernel";
    10
                     data = /incbin/("./arch/arm/boot/zImage");
    11
                     type = "kernel";
    12
                     arch = "arm";
    13
                     os = "linux";
    14
                     compression = "none";
                     load = <0x8000>;
    15
    16
                     entry = <0x8000>;
    17
                     hash-1 {
                          algo = "sha256";
    18
    19
                     };
    20
                 };
    21
                 fdt-zynq-alientek.dtb {
    22
                     description = "Flattened Device Tree blob";
                     data = /incbin/("./arch/arm/boot/dts/zynq-alientek.dtb");
    23
    24
                     type = "flat_dt";
                     arch = "arm";
    25
    26
                     compression = "none";
    27
                     hash-1 {
    28
                          algo = "sha256";
    29
                     };
    30
                 };
    31
          };
    32
    33
            configurations {
    34
                default = "conf-zynq-alientek.dtb";
    35
                conf-zynq-alientek.dtb {
                   description = "1 Linux kernel, FDT blob";
    36
                   kernel = "kernel-1";
    37
    38
                   fdt = "fdt-zynq-alientek.dtb";
    39
    40
    41
                     hash-1 {
                          algo = "sha256";
    42
    43
                     };
    44
                };
          };
    45
    46 };
```

保存文件内容后,在终端中输入"cd ~/petalinux/atk-linux/linux-xlnx-xlnx_reabase_v5.4_2020.2" 进入到 Linux 源码根目录下,然后输入命令"mkimage -f fitimage.its image.ub"来生成 image.ub 文 件,如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

wmq@Linux:~/peta	alinux/atk-linux/linux-xlnx-xlnx_rebase_v5.4_2020.2\$ mkimage -f fitimage.its image.ub
FIT description	: U-Boot fitImage for Alientek ZYNQ
Created:	Mon Aug 21 16:16:58 2023
Image 0 (kerne	l-1)
Description:	Linux kernel
Created:	Mon Aug 21 16:16:58 2023
Туре:	Kernel Image
Compression:	uncompressed
Data Size:	4183480 Bytes = 4085.43 KiB = 3.99 MiB
Architecture:	ARM
OS:	Linux
Load Address:	0×00008000
Entry Point:	0×00008000
Hash algo:	sha256
Hash value:	3f2204bd39e93ad7c2b0b7bff6d2e92b85de03faa03cb4b2d08b12cf9421a669
Image 1 (fdt-z	yng-alientek.dtb)
Description:	Flattened Device Tree blob
Created:	Mon Aug 21 16:16:58 2023
Туре:	Flat Device Tree
Compression:	uncompressed
Data Size:	11879 Bytes = 11.60 KiB = 0.01 MiB
Architecture:	ARM
Hash algo:	sha256
Hash value:	f415e1e465a2461b931fcf569b07ff26cd0251af95b5ef0c7d8581ac03030cab
Default Config	uration: 'conf-zynq-alientek.dtb'
Configuration	0 (conf-zynq-alientek.dtb)
Description:	1 Linux kernel, FDT blob
Kernel:	kernel-1
FDT:	fdt-zynq-alientek.dtb
Hash algo:	sha256
Hash value:	unavailable
wmq@Linux:~/peta	alinux/atk-linux/linux-xlnx-xlnx_rebase_v5.4_2020.2\$

图 18.3.4 生成 image. ub 文件

在内核根目录下可以找到生成的 image.ub 文件,如下图所示:

wmq@Linux:~/petalinux/atk-linux/linux-xlnx-xlnx_rebase_v5.4_2020.2\$ ls -l image.ub -rw-rw-r-- 1 wmq wmq 4197236 8月 21 16:16 image.ub wmq@Linux:~/petalinux/atk-linux/linux-xlnx-xlnx_rebase_v5.4_2020.2\$ wmq@Linux:~/petalinux/atk-linux/linux-xlnx-xlnx_rebase_v5.4_2020.2\$

图 18.3.5 生成的 image. ub 文件

如何验证该文件确实可行呢。如果是通过 SD 卡启动领航者开发板可以将生成的 image.ub 文件复制到 SD 卡中,如果是通过网络启动领航者开发板,可以将其复制到/tftpboot 目录下, 启动领航者开发板测试。经测试,是可以启动内核的,启动结果如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

Zyng> tftpboot 10000000 image.ub
TFTP from server 192.168.1.20; our IP address is 192.168.1.10
Filename 'image.ub'.
Load address: 0x10000000
Loading: ####################################

3.5 MiB/s
done
Bytes transferred = 4195816 (4005e8 hex)
Zyng> bootm 10000000
<pre>## Loading kernel from FIT Image at 10000000</pre>
Using 'conf-lzyng-alientek.dtb' configuration
Verifying Hash Integrity OK
Trying 'kernel-1' kernel subimage
Description: Linux kernel
Type: Kernel Image
Compression: uncompressed
Data Statt: 0x100000df
Data Size: Histood Bytes - 4 MB
Alchitecture. And
Losd Address. 0x00008000
Hash alon: sha256
Haoh algo. Shazoo Hash value. ac5sahqlh2drla746a241f0h72dfh262729a2382rl6a4782a9a24df8904h9a1
Varies Cooperative abased of the state of th
ti Loding fdt from FIT Inge at 1000000
Using 'confeiguration
Verifying Hash Integrity OK
Trving 'fdt-lzvng-alientek.dtb' fdt subimage
Description: Flattened Device Tree blob
Type: Flat Device Tree
Compression: uncompressed
Data Start: 0x103fd15c
Data Size: 12060 Bytes = 11.8 KiB
Architecture: ARM
Hash algo: sha256
Hash value: bc343d3ef9d2504laa386cd7b74db0c64ab00cc0537c70b60a6447caf8d5cla5
Verifying Hash Integrity sha256+ OK
Booting using the fdt blob at 0x103fd15c
Loading Kernel Image
Loading Device Tree to leb02000, end leb07flb OK
Starting kernel
Booting Linux on physical CPU 0x0
Linux version 5.4.0-xilinx (wmg@Linux) (gcc version 9.2.0 (GCC)) #1 SMP PREEMPT Mon Aug 7 13:17:14 CST 2023
CPU: ARMv7 Processor [413fc090] revision 0 (ARMv7), cr=18c5387d

图 18.3.6 验证结果

至此,我们终于知道了 image.ub 是怎么来的,怎么生成的了。另外 image.ub 既然是 U-Boot fitImage,那 U-Boot fitImage 又是什么呢?限于篇幅就不再介绍了,可自行上网搜索,笔者找到一篇不错的介绍,推荐下,链接如下:

http://www.wowotech.net/u-boot/fit_image_overview.html

从该链接可知 U-Boot fitImage 主要来源于 Unify Kernel 的思想,其思想如下:

"在编译 linux kernel 的时候,不必特意的指定具体的架构和 SOC,只需要告诉 kernel 本 次编译需要支持哪些板级的 platform 即可,最终将会生成一个 Kernel image,以及多个和具体 的板子(哪个架构、哪个 SOC、哪个版型)有关的 FDT image (dtb 文件)。

bootloader 在启动的时候,根据硬件环境,加载不同的 dtb 文件,即可使 linux kernel 运行 在不同的硬件平台上,从而达到 unify kernel 的目标。"



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

18.4 CPU 调频策略及配置文件的修改

18.4.1 CPU 调频策略

1、设置领航者开发板的 CPU 调频策略

我们知道: 主频越高, 功耗也越高。为了节省 CPU 的功耗和减少发热, 我们有必要根据 当前 CPU 的负载状态, 动态地提供刚好足够的主频给 CPU。调频或称变频技术应运而生。变 频技术作为电源管理技术以节能为目的加入 linux 内核。我们可以通过配置内核来选择不同的 调频策略。

我们来看一下如何通过图形化界面配置 Linux 内核的 CPU 调频策略,输入"make menuconfig"打开 Linux 内核的图形化配置界面,如下图所示:

.config - Linux/arm 5.4.0 Kernel Configuration
Linux/arm 5.4.0 Kernel Configuration Arrow keys navigate the menu. <enter> selects submenus> (or empty submenus). Highlighted letters are hotkeys. Pressing <y> includes, <n> excludes, <m> modularizes features. Press <esc><esc> to exit, <? > for Help, for Search. Legend: [*] built-in [] excluded <m> module <> module</m></esc></esc></m></n></y></enter>
<pre>*** Compiler: arm-xilinx-linux-gnueabi-gcc (GCC) 9.2.0 *** General setup> -*- Patch physical to virtual translations at runtime System Type> Bus support> Kernel Features> Boot options> CPU Power Management> Floating point emulation></pre>
<pre>Power management options> Firmware Drivers> [] ARM Accelerated Cryptographic Algorithms (+) </pre> <pre> <</pre></pre></pre></pre></pre></pre></pre></pre></pre></pre>

图 18.4.1 Linux 内核图形化配置界面

进入如下路径:

CPU Power Management

-> CPU Frequency scaling

-> CPU Frequency scaling

-> Default CPUFreq governor

打开默认调频策略选择界面,如下图所示:



论坛:www.openedv.com/forum.php



图 18.4.2 默认调频策略选择

可以看到 Linux 内核一共有 6 种调频策略,

- ① Performance,最高性能,直接用最高频率,不考虑耗电。
- ② Powersave,省电模式,通常以最低频率运行,系统性能会受影响,一般不会用这个。
- ③ Userspace,可以在用户空间手动调节频率,可以看到这是默认的配置。
- ④ Ondemand, 定时检查负载, 然后根据负载来调节频率。负载低的时候降低 CPU 频率以省 电,负载高的时候提高 CPU 频率,增加性能。
- ⑤ conservative: 保守模式, 类似于 ondemand, 但调整相对较缓。
- ⑥ schedutil: 4.7 版本内核新增加的一种调度策略,可以直接使用调度程序提供的信息做出 调整 cpu 频率的决策;也可以调用 cpufreq 驱动程序更改频率以立即调整 CPU 的性能,无 需生成进程上下文或其他工作项。

我们可以根据实际需求选择合适的调频策略。

18.4.2 配置文件的修改

在 18.2.1 节添加开发板默认配置文件时我们直接复制 ZC702 的配置文件,该文件能够使 我们的领航者开发板启动 linux 内核, 然而却不能保证所有的硬件都能够工作。因为官方的这 份配置只是基础配置, 随底层 Vivado 工程的不同需要添加额外的配置, 如使能 GPIO、IIC、 以太网驱动等。当然了我们可以在 linux 内核的图形化配置界面配置, 使能以后会保存 到.config 文件中。这样做是可以的,不过有两个问题,一个是我们得一个个的手动配置,比 较麻烦,还要确保配置正确,二是当我们执行"make clean"清理工程以后.config 文件就会被 删除掉,因此我们所有的配置内容都会丢失,结果就是前功尽弃,一"删"回到解放前,所 以我们可以直接使用第八章 Linux 显示设备的使用的 Petalinux 工程里的 linux 配置。这个是 Petalinux 工具配置好的,使用第八章 petalinux 工程里的 linux 配置的方法有两种。

1、直接另存为.config 文件为 alientek_zynq_defconfig

既然 linux 的配置项保存在.config 文件中,那么就可以直接将第八章的 Petalinux 工程里的 linux 配置文件.config 另存为 alientek_zynq_defconfig, 然后将其复制到 arch/arm/configs 目录下,



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

替换以前的 alientek_zynq_defconfig。这样以后执行 "make alientek_zynq_defconfig" 重新配置 Linux 内核的时候就会使用新的配置文件。此种方式了解即可,不推荐。

2、通过图形界面保存配置文件

相比于第 1 种直接另存为.config 文件,第 2 种方法就很"文雅"了。进入第八章的 Petalinux 工程,在终端输入 sptl,先配置 petalinux 的环境变量,然后输入 petalinux-config -c kernel 配置 linux 内核,过一会就会弹出图形化配置界面,选择图形界面中的"< Save >"选项,在图形界面中保存配置文件,如下图所示:

.config - Linux/arm 5.4.0 Kernel Configuration
Linux/arm 5.4.0 Kernel Configuration Arrow keys navigate the menu. <enter> selects submenus> (or empty submenus). Highlighted letters are hotkeys. Pressing <y> includes, <n> excludes, <m> modularizes features. Press <esc><esc> to exit, <? > for Help, for Search. Legend: [*] built-in []</esc></esc></m></n></y></enter>
<pre>*** Compiler: arm-xilinx-linux-gnueabi-gcc (GCC) 9.2.0 ***</pre>
General setup>
-*- Patch physical to virtual translations at runtime
System Type>
Bus support>
Rent options
Floating point emulation>
Power management options>
V(+)
<select> < Exit > < Help > < Save > < Load ></select>

图 18.4.3 保存配置

通过键盘的"→"键,移动到"<Save>"选项,然后按下回车键,打开文件名输入对话框,如下图所示:

.config - Lir	nux/arm 5.4.0 Kernel Configuration
	Enter a filename to which this configuration should be saved as an alternate. Leave blank to abort.
	.config
	< 0k > < Help >

图 18.4.4 输入文件名

在上图中输入要保存的文件名,可以带路径。比如我们要将新的配置文件保存到目录 /home/wmq/Linux/linux-xlnx-xilinx-v2020.1/arch/arm/configs 下 , 文 件 名 为 alientek_zynq_defconfig,也就是用新的配置文件替换掉旧的默认配置文件。那么我们在上图



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

中输入"/home/wmq/Linux/linux-xlnx-xilinx-v2020.1/arch/arm/configs/alientek_zynq_defconfig" 即可,如下图所示:

.config - L	inux/arm 5.4.0 Kernel Configuration
	Enter a filename to which this configuration should be saved as an alternate. Leave blank to abort.
	v2020.1/arch/arm/configs/alientek_zynq_defconfig
	< 0k > < Help >

图 18.4.5 输入文件名

设置好文件名以后选择下方的" < Ok >"按钮,保存文件并退出。这样当我们使用 "make alientek_zynq_defconfig"重新配置 Linux 内核的时候就可以使用 Petalinux 配置好的内 核。

关于 Linux 内核的移植就讲解到这里,简单总结一下移植步骤:

- ① 在 Linux 内核中查找可以参考的板子,一般都是半导体厂商自己做的开发板。
- ② 编译出参考板子对应的 zImage 和.dtb 文件。
- ③ 使用参考板子的 zImage 文件和.dtb 文件在我们所使用的板子上启动 Linux 内核,看能否启 动。
- ④ 如果能启动的话就万事大吉,如果不能启动就需要调试 Linux 内核和修改设置树。不过一 般都会参考半导体官方的开发板设计自己的硬件,所以大部分情况下都会启动起来。启 动 Linux 内核用到的外设不多,一般就 DRAM(Uboot 都初始化好的)和串口。
- ⑤ 修改相应的驱动,像 USB、EMMC、SD 卡等驱动官方的 Linux 内核都是已经提供好了, 基本不会出问题。重点是网络驱动,因为 Linux 驱动开发一般都要通过网络调试代码,所 以一定要确保网络驱动工作正常。如果是处理器内部 MAC+外部 PHY 这种网络方案的话, 一般网络驱动都很好处理,因为在 Linux 内核中是有外部 PHY 通用驱动的。只要设置好 复位引脚、PHY 地址信息基本上都可以驱动起来。
- ⑥ Linux 内核启动以后需要根文件系统,如果没有根文件系统的话肯定会崩溃,所以确定 Linux 内核移植成功以后就要开始根文件系统的构建。



正点原子

第十九章 根文件系统构建

Linux"三巨头"已经完成了2个了,就剩最后一个 rootfs(根文件系统)了,本章我们就来 学习一下根文件系统的组成以及如何构建根文件系统。这是 Linux 移植的最后一步,根文件 系统构建好以后就意味着我们已经拥有了一个完整的、可以运行的最小系统。以后我们就在 这个最小系统上编写、测试 Linux 驱动,移植一些第三方组件,逐步的完善这个最小系统。 最终得到一个功能完善、驱动齐全、相对完善的操作系统。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

19.1 根文件系统简介

根文件系统一般也叫做 rootfs, 那么什么叫根文件系统? 看到"文件系统"这四个字, 很 多人,包括我第一反应就是FATFS、FAT、EXT4、YAFFS和NTFS等这样的文件系统。在这 里,根文件系统并不是FATFS、EXT4这样的文件系统,这些文件系统属于Linux内核的一部 分。Linux 中的根文件系统更像是一个文件夹或者叫做目录(在我看来就是一个文件夹,只不 过是特殊的文件夹),在这个目录里面会有很多的子目录。根目录下和子目录中会有很多的文 件,这些文件是 Linux 运行所必须的,比如库、常用的软件和命令、设备文件、配置文件等 等。以后我们说到文件系统,如果不特别指明,统一表示根文件系统。对于根文件系统专业 的解释,百度百科上是这么说的(原谅我把百度百科引用为专业解释,因为我实在找不到根文 件系统的最初定义,也不要建议我到哪些 404 网站去查找,毕竟我胖,我怕翻到一般梯子不 稳把我摔丑了):

根文件系统首先是内核启动时所 mount(挂载)的第一个文件系统,内核代码映像文件保存 在根文件系统中,而系统引导启动程序会在根文件系统挂载之后从中把一些基本的初始化脚 本和服务等加载到内存中去运行。

百度百科上说内核代码镜像文件保存在根文件系统中,但是我们嵌入式 Linux 并没有将 内核代码镜像保存在根文件系统中,而是保存到了其他地方。比如 NAND Flash 的指定存储地 址、EMMC 专用分区中。根文件系统是 Linux 内核启动以后挂载(mount)的第一个文件系统, 然后从根文件系统中读取初始化脚本,比如 rcS, inittab 等。根文件系统和 Linux 内核是分开 的,单独的 Linux 内核是没法正常工作的,必须要搭配根文件系统。如果不提供根文件系统, Linux 内核在启动的时候就会提示内核崩溃(Kernel panic)的提示,这个在 Linux 内核移植章节 已经说过了。

根文件系统的这个"根"字就说明了这个文件系统的重要性,它是其他文件系统的根, 没有这个"根",其他的文件系统或者软件就别想工作。比如我们常用的 ls、mv、ifconfig 等 命令其实就是一个个小软件,只是这些软件没有图形界面,而且需要输入命令来运行。这些 小软件就保存在根文件系统中,这些小软件是怎么来的呢?这个就是我们本章教程的目的, 教大家来构建自己的根文件系统,这个根文件系统是满足 Linux 运行的最小根文件系统,后 续我们可以根据自己的实际工作需求不断的去填充这个最小根文件系统,最终使其成为一个 相对完善的根文件系统。

在构建根文件系统之前,我们先来看一下根文件系统里面大概都有些什么内容,以 Ubuntu 为例,根文件系统的目录名字为'/',没看错就是一个斜杠,所以输入如下命令就可 以讲入根目录中:

cd / //进入根目录 进入根目录以后输入"ls"命令查看根目录下的内容都有哪些,结果如下图所示: wmq@Linux:~\$ cd / wmq@Linux:/\$ ls tmp home lib64 bin opt snap

DOOT	ιπιτΓα.ιmg	LIDX32	ргос	SEV	USF
cdrom	initrd.img.old	lost+found	root	swapfile	var
dev	lib	media	run	sys	vmlinuz
etc	lib32	mnt	sbin	tftpboot	vmlinuz.old
wmq@Li	nux:/\$				

图 19.1.1 Ubuntu 根目录 559



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

上图中根目录下子目录和文件不少,但是这些都是 Ubuntu 所需要的,其中有很多子目录 和文件我们嵌入式 Linux 是用不到的,所以这里就讲解一些常用的子目录:

1、/bin 目录

看到"bin"大家应该能想到 bin 文件, bin 文件是可执行文件, 所以此目录下存放着系统 需要的可执行文件,一般都是一些命令,比如 ls、mv 等命令。此目录下的命令所有的用户都 可以使用。

2、/dev 目录

dev 是 device 的缩写,所以此目录下的文件都是和设备有关的,是设备文件。在 Linux 下 一切皆文件,即使是硬件设备,也是以文件的形式存在的,比如/dev/ttvPS0(ZYNO 根目录会 有此文件)就表示 ZYNO 的串口 0, 我们要想通过串口 0 发送或者接收数据就要操作文件 /dev/ttyPS0, 通过对文件/dev/ttyPS0的读写操作来实现串口0的数据收发。

3、/etc 目录

此目录下存放着各种服务程序的配置文件,里面的配置文件非常多。但是在嵌入式 Linux 下此目录会很简洁。

4、/lib 目录

lib 是 library 的简称,也就是库的意思,因此此目录下存放着 Linux 所必须的库文件。这 些库文件是共享库,命令和用户编写的应用程序要使用这些库文件。

5、/mnt 目录

临时挂载目录,一般用于安装移动介质,可以在此目录下创建空的子目录,比如/mnt/sd、 /mnt/usb,这样就可以将 SD 卡或者 U 盘挂载到/mnt/sd 或者/mnt/usb 目录中。

6、/proc 目录

此目录一般是空的,当Linux系统启动以后会将此目录作为proc文件系统的挂载点,proc 是个虚拟文件系统,没有实际的存储设备。proc 里面的文件都是临时存在的,一般用来存储 系统运行信息文件。

7、/usr 目录

要注意, usr 不是 user 的缩写, 而是 Unix Software Resource 的缩写, 也就是 Unix 操作系 统软件资源目录,用于存放用户级的数据和程序。这里有个小知识点,那就是 Linux 一般被 称为类 Unix 操作系统,苹果的 MacOS 也是类 Unix 操作系统。关于 Linux 和 Unix 操作系统的 渊源可以直接在网上找 Linux 的发展历史。

8、/var 目录

此目录用于存放一些动态的程序数据,如系统日志等。

9、/sbin 目录

此目录用于存放系统管理员常用的管理程序。具有管理员权限才可以使用,主要用户系 统管理。

10、/sys 目录



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

系统启动以后此目录作为 sysfs 文件系统的挂载点, sysfs 是一个类似于 proc 文件系统的 特殊文件系统, sysfs 也是基于 ram 的文件系统,也就是说它也没有实际的存储设备。此目录 是系统设备管理的重要目录,此目录通过一定的组织结构向用户提供详细的内核数据结构信 息。

11、/opt

可选的文件、软件存放区,由用户选择将哪些文件或软件放到此目录中。

12, /home

普通用户的工作目录。

13, /root

超级用户的工作目录

关于 Linux 的根目录就介绍到这里, 接下来的构建根文件系统就是研究如何创建上面这 些子目录以及子目录中的文件。

19.2 Petalinux 构建根文件系统

构建根文件系统有两种方式,一种是手工创建根目录并编译配置相关文件,另一种是通 过相应的工具来构建。手工构建方式耗时耗力并无多大意义,这里我们采用相应的工具来构 建。构建根文件系统的工具有很多,如 BusyBox、buildroot、Yocto 等。由于 Petalinux 工具已 经包含了 Yocto, 所以我们可以直接使用 Petalinux 工具来构建根文件系统。

实际上我们在 6.3.6 节配置 Linux 根文件系统就是在构建根文件系统,当时我们使用的是 默认配置,在后面的使用中可以看到默认配置是可以满足一般的需求的。现在我们进入第六 章创建的 Petalinux 工程目录下,具体看下 Petalinux 的根文件系统构建。

进入 Petalinux 工程目录后,在终端输入下面的命令配置根文件系统:

petalinux-config -c rootfs

下图就是根文件系统的配置界面:



图 19.2.1 根文件系统的配置界面 561



原子哥在线教学: www.yuanzige.com 论坛:

论坛:www.openedv.com/forum.php

可以看到有 6 个子菜单,其中 "Filesystem Packages"菜单主要是配置根文件常用的工具软件,包括内核调试软件、流媒体软件、Python 软件以及图形界面软件等,具体的我们就不 细看了。

"Petalinux Package Groups"菜单可以使能 Petalinux 提供的软件包组,所谓的软件包组即 将与该软件相关的组件和模块放入一组,譬如"packagegroup-petalinux-opencv"包含 opencv 需要的包: opencv、libopencv-core、libopencv-highgui、libopencv-imgproc、libopencvobjdetect、libopencv-ml、libopencv-calib3d、libopencv-ccalib,当使能"packagegrouppetalinux-opencv"时,这些包也会自动使能,这样做的好处是解决了软件依赖的问题。

"Image Features"菜单可以配置根文件系统的某些功能,如是否支持 ssh,使用 dropbear 的 ssh 还是 openssh 等。如果不希望每次启动 linux 后都得输入密码验证,可以使能该菜单下 的 "auto-login"选项。如下图所示:

/home/wmq/petalinux/ALIENTEK-ZYNQ/project-spec/configs/rootfs_config - Configurati o→Image Features						
Image Features Arrow keys navigate the menu. <enter> selects submenus> (or empty submenus). Highlighted letters are hotkeys. Pressing <y> includes, <n> excludes, <m> modularizes features. Press <esc> to exit, <? > for Help, for Search. Legend: [*] built-in [] excluded <m> module</m></esc></m></n></y></enter>						
<pre>Tor Help, Tor Search. Legend: [*] built-in [] excluded <m> module [*] ssh-server-dropbear [] ssh-server-openssh [*] hwcodecs [] package-management -*- debug-tweaks [*] auto-login</m></pre>						
<pre><select> < Exit > < Help > < Save > < Load ></select></pre>						

图 19.2.2 配置自动登录

这样配置后,会自动登录,不用再手动输入用户名和密码,方便调试。

"apps"菜单可用来构建根文件系统的用户应用。

"user packages"菜单可用来配置用户自定义的软件包,因为我们没有自定义软件包,所 以该菜单为空。

"PetaLinux RootFS Settings" 主要是用来设置 root 用户的密码, 默认为"root"。

介绍完这 6 个菜单后,我们退出配置界面。如果没有做修改,就不需要重新编译根文件 系统,如果做了修改可以使用如下命令编译根文件系统:

petalinux-build -c rootfs

Petalinux 工程生成的根文件系统在 images/linux/目录下,如下图所示:

领航者 ZYNQ 之间	嵌入式 Linux 开始	发指南		② 正点原子	-			
原子哥在线教学:www	w.yuanzige.com	论坛:ww	w.openedv.com/	/forum.php				
wmq@Linux:~/p	<pre>wmq@Linux:~/petalinux/ALIENTEK-ZYNQ/images/linux\$ ls</pre>							
BOOT.BIN	rootfs.cpio.gz		system.bit	vmlinux				
boot.scr	rootfs.cpio.gz.	u-boot	system.dtb	zImage				
image.ub	rootfs.jffs2		u-boot.bin	zynq_fsbl.elf				
pxelinux.cfg	rootfs.manifest		u-boot.elf					
rootfs.cpio	rootfs.tar.gz		uImage					
<pre>wmq@Linux:~/petalinux/ALIENTEK-ZYNQ/images/linux\$</pre>								
wmq@Linux:~/p	etalinux/ALIENTE	K-ZYNQ/1	images/linux	\$				

图 19.2.3 Petalinux 工程生成的根文件系统

图中带有"rootfs"的文件名都是 Petalinux 生成的根文件系统。一般我们使用 rootfs.tar.gz 文件。我们可以将 rootfs.tar.gz 文件解压到 SD 卡的第二个分区——ext4 文件系统分区来测试 根文件系统,可参考第八章 Linux 显示设备的使用。不过我们在 Linux 驱动开发的时候一般都 是通过 nfs 挂载根文件系统的,当产品最终上市的时候才会将根文件系统烧写到 EMMC 或者 SD 中,这样做的好处是方便调试,免得多次插拔 SD 卡。

我们在开发环境搭建的 4.4 节设置的 nfs 服务器目录下创建一个名为 rootfs 的子目录(名字 随意,此处为了方便使用了 rootfs),比如笔者的电脑中 "/home/wmq/workspace/nfs"就是笔 者设置的 NFS 服务器目录,使用如下命令创建名为 rootfs 的子目录: ~

mkdir rootfs

创建好 rootfs 子目录就可以用来存放我们的根文件系统了。我们将 rootfs.tar.gz 文件解压 到 rootfs 目录。命令如下

tar -xzvf images/linux/rootfs.tar.gz -C ~/workspace/nfs/rootfs/ 解压完成的 rootfs 目录内容如下图所示:

wmq@Linux:	~/w	orksp	pace	/nfs/r	ootf	s\$ 1	1		
总用量 68									
drwxr-xr-x	17	wmq	wmq	4096	8月	14	10:32	./	
drwxr-xr-x	3	wmq	wmq	4096	8月	14	10:28	/	
drwxr-xr-x	2	wmq	wmq	4096	8月	14	10:32	bin/	
drwxr-xr-x	3	wmq	wmq	4096	8月	14	10:32	boot/	
drwxr-xr-x	2	wmq	wmq	4096	8月	14	10:06	dev/	
drwxr-xr-x	24	wmq	wmq	4096	8月	14	10:32	etc/	
drwxr-xr-x	4	wmq	wmq	4096	8月	14	10:32	home/	
drwxr-xr-x	5	wmq	wmq	4096	8月	14	10:32	lib/	
drwxr-xr-x	2	wmq	wmq	4096	8月	14	10:06	media/	
drwxr-xr-x	2	wmq	wmq	4096	8月	14	10:06	mnt/	
dr-xr-xr-x	2	wmq	wmq	4096	8月	14	10:06	ргос/	
drwxr-xr-x	2	wmq	wmq	4096	8月	14	10:06	run/	
drwxr-xr-x	2	wmq	wmq	4096	8月	14	10:32	sbin/	
dr-xr-xr-x	2	wmq	wmq	4096	8月	14	10:06	sys/	
drwxrwxr-x	2	wmq	wmq	4096	8月	14	10:06	tmp/	
drwxr-xr-x	10	wmq	wmq	4096	8月	14	10:10	usr/	
drwxr-xr-x	8	wmq	wmq	4096	8月	14	10:23	var/	
wma@Linux:-	~ / wo	orksi	bace	/nfs/r	ootf	s\$			

图 19.2.4 rootfs 目录

可以看到该有的目录都有了。接下来我们测试该根文件系统。

注:如果读者使用的 image.ub 是使用 Petalinux 软件生成的,请重新配置 petalinux,重新 设置根文件系统类型,输入如下命令:

petalinux-config



原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php

在弹出的界面中,进入到"Image Packaging Configuration"菜单下的"Root filesystem type(INITRAMFS)" 子菜单下,如下图所示:

/home/wm C→Image	q/petalinux/ALIENTEK-ZYNQ/project-spec/configs/config - misc/config System Packaging Configuration
	Root filesystem type Use the arrow keys to navigate this window or press the hotkey of the item you wish to select followed by the <space BAR>. Press <? > for additional information about this</space
	() INITRAMFS (X) INITRD () JFFS2 () NFS () EXT4 (SD/EMMC/SATA/USB) () other
	<pre><select> < Help ></select></pre>

图 19.2.5 选择"SD card"

将根文件系统配置成 SD 卡, 然后重新编译 Petalinux 工程, 将生成的 image.ub 文件复制 到 SD 卡的 boot 分区。

19.3 根文件系统测试

测试根文件系统 rootfs 的方法有两种——本地测试和网络测试。本地测试就是将根文件系 统打包到启动镜像中如第六章的 image.ub, 或者将其烧写到本地设备中如第六章制作的 SD 卡 ext4 分区;网络测试就是使用 NFS 挂载启动,这在 uboot 中进行设置。这里我们使用网络测 试的方式测试根文件系统。

uboot 里面的 bootargs 环境变量会设置"root"的值,所以我们将 root 的值改为 NFS 挂载 即可。在 Linux 内核源码里面有相应的文档讲解如何设置, 文档为 Documentation/filesystems/nfs/nfsroot.txt, 格式如下:

root=/dev/nfs nfsroot=[<server-ip>:]<root-dir>[,<nfs-options>] ip=<client-ip>:<server-ip>:<gwip>:<netmask>:<hostname>:<device>:<autoconf>:<dns0-ip>:<dns1-ip>

root=/dev/nfs 这是启用 nfs 挂载所必需的。注意,它不是真正的设备,只是告诉内核使用 NFS 加载根文件系统。

nfsroot:

<server-ip>: nfs 服务器 IP 地址,也就是存放根文件系统主机的 IP 地址,比如笔者的存 放根文件系统主机 Ubuntu 的 IP 地址为 192.168.1.20。

<root-dir>: nfs 服务器上根文件系统的存放路径,比如笔者的是 /home/wmq/workspace/nfs/rootfs。

<nfs-options>: NFS 选项,所有选项以逗号分隔,一般不设置,使用默认设置即可。

ip:

可以自动配置(开发板连接路由器,自动获取 IP 地址),也可以手动配置(开发板直连 电脑,手动设置静态 IP 地址),由下面的<autoconf>选项决定。使用自动配置时,可直接变 成"ip=dhcp"。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

<hostname>: 客户机的名字,一般不设置,此值可以空着。

<**device>:** 设备名,也就是网卡名,一般是 eth0, eth1....,正点原子的领航者开发板的 ENET2 为 eth0, ENET1 为 eth1。如果你的电脑只有一个网卡,那么基本只能是 eth0。这里我 们使用 ENET2,所以网卡名就是 eth0。

autoconf>: 用于自动配置 ip 的方法。自动配置的情况下可以使用 DHCP 协议,如果手动配置,需要设置为 off。使用自动配置时,可直接变成"ip=dhcp",无需配置前面的

<dns0-ip>: DNS0服务器 IP 地址,不使用。

<dns1-ip>: DNS1 服务器 IP 地址,不使用。

根据上面的格式设置 bootargs 环境变量的 root 值如下:

root=/dev/nfs rw nfsroot=192.168.1.20:/home/wmq/workspace/nfs/rootfs ip=192.168.1.10:192.168. 1.20:192.168.1.1:255.255.255.0::eth0:off'

启动开发板,进入uboot命令行模式,然后重新设置 bootargs 环境变量,命令如下:

setenv bootargs 'console=ttyPS0,115200 root=/dev/nfs rw nfsroot=192.168.1.20:/home/wmq/workspace /nfs/rootfs,nfsvers=3 ip=192.168.1.10:192.168.1.20:192.168.1.1:255.255.255.0::eth0:off' //设置 bootargs

saveenv //保存环境变量

如果领航者开发板是通过网线与路由器相连接,可以使用下面的 bootargs 环境变量:

setenv bootargs 'console=ttyPS0,115200 root=/dev/nfs nfsroot=192.168.1.20:/home/wmq/workspace/nfs/rootf s,tcp ip=dhcp rw'

saveenv //保存环境变量

设置好以后使用"boot"命令启动 Linux 内核,结果如下图所示:



原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php



图 19.3.1 进入根文件系统

从上图可以看出,系统已经进入了根文件系统,说明我们的根文件系统工作了。如果没 有启动进入根文件系统的话可以重启一次开发板试试。我们可以输入"ls/"命令测试一下, 结果如下图所示:

root@ALIENTEK-ZYNQ:~# 1s /														
bin	boot	dev	etc	home	lib	media	mnt	proc	run	sbin	sys	tmp	usr	var
root@ALIENTEK-ZYNQ:~#														
root@Al	LIENTEK-	-ZYNQ:^	~ #											

图 19.3.2 1s 命令测试

可以看出 ls 命令工作正常,说明根文件系统基本没问题。接下来我们进行其他功能测试。

19.4 根文件系统其他功能测试

19.4.1 软件运行测试

我们使用 Linux 的目的就是运行我们自己的软件,我们编译的应用软件一般都使用动态 库,使用动态库的话应用软件体积就很小,但是得提供库文件,库文件我们已经添加到了根 文件系统中。我们编写一个小小的测试软件来测试一下库文件是否工作正常,在根文件系统 下创建一个名为"drivers"的文件夹,以后我们学习 Linux 驱动的时候就把所有的实验文件放 到这个文件夹里面。

在 ubuntu 下的~/workspace/nfs/rootfs/drivers 目录下使用 vim 编辑器新建一个 hello.c 文件, 在 hello.c 里面输入如下内容:

```
1 #include <stdio.h>
```

- 2 #include <unistd.h>
- 3 int main(void)
- 4 {

```
5 int i=0;
```

```
6 while(i < 50){
```

7 printf("Hello World!\r\n");



原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php

8 i++; 9 sleep(2);10 }

11 return 0;

12 }

hello.c 内容很简单,就是循环输出"hello world"50次。sleep 相当于 Linux 的延时函数, 单位为秒,所以 sleep(2)就是延时 2 秒。编写完代码后执行编译,因为我们是要在 ARM 芯片 上运行的,所以要用交叉编译器去编译,也就是使用 arm-xilinx-linux-gnueabi-gcc 编译(需要 在当前终端中建立 Petalinux 的环境变量), 命令如下:

. /opt/petalinux/2020.2/environment-setup-cortexa9t2hf-neon-xilinx-linux-gnueabi

设置完环境后,就可以使用交叉编译工具链去编译 hello.c.但在这里我们不能直接使用交 叉编译器(arm-xilinx-linux-gnueabi-gcc)去编译,而是使用\${CC}和\${LD}两个环境变量(可 以简写成\$CC 和\$LD) 去编译 hello.c。接下来我们将\$(CC)和\$(LD)打印出来看看,打印结果 如下图所示:



```
图 19.4.1 打印$(CC)和$(LD)的值
```

可以看到\$CC 环境变量是带参数的 arm-xilinx-linux-gnueabi-gcc 交叉编译工具链的定义, 特别是其中的 sysroot 参数,没有该参数,直接使用 arm-xilinx-linux-gnueabi-gcc 编译 C 源文件 会报错。这就是为何不能直接使用 arm-xilinx-linux-gnueabi-gcc, 而是需要使用\$CC 的原因, 所以后面需要编译用于开发板上的C程序源码文件时,应使用如下方式编译:

\$CC c 源码文件

使用 arm-xilinx-linux-gnueabi-gcc 将 hello.c 编译为 hello 可执行文件。

\$CC hello.c -o hello

这个 hello 可执行文件究竟是不是 ARM 使用的呢? 使用"file"命令查看文件类型以及编 码格式:

```
//查看 hello 的文件类型以及编码格式
file hello
结果如下图所示:
```

```
wmq@Linux:~/workspace/nfs/rootfs/drivers$ file hello
hello: ELF 32-bit LSB shared object, ARM, EABI5 version 1 (SYSV), dynamically linke
d, interpreter /lib/ld-linux-armhf.so.3, BuildID[sha1]=36477d6352208a9c37a63cc3c1c5
674375189fe4, for GNU/Linux 3.2.0, with debug_info, not stripped
wmq@Linux:~/workspace/nfs/rootfs/drivers$
wmq@Linux:~/workspace/nfs/rootfs/drivers$
```

图 19.4.2 查看 hello 编码格式

从上图可以看出,输入"file hello"后输出了如下信息:

hello: ELF 32-bit LSB shard object, ARM, EABI5 version 1 (SYSV), dynamically linked.....

意思是 hello 文件是 32 位的 LSB 可执行文件,基于 ARM 架构,并且是动态链接的,所 以我们编译出来的 hello 可以在领航者开发板上运行。在 drivers 目录下执行这个可执行文件:



原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php

//执行 hello ./hello 结果如下图所示:

root@ALIENTEK-ZYNQ:/drivers# ./hello
Hello World!

图 19.4.3 hello运行结果

可以看出,程序 hello 运行正常,说明我们的根文件系统中的共享库是没问题的。在程序 运行的时候应该能感觉到, hello 程序执行的时候终端是没法用的, 除非使用"ctrl+c"组合键 来关闭 hello 程序,那么有没有办法既能让 hello 正常运行,而且终端能够正常使用?答案是 有的,让 hello 程序进入后台运行就行了,让一个软件进入后台的方法很简单,运行软件的时 候加上 "&" 即可, 比如 "./hello &" 就是让 hello 程序在后台运行。在后台运行的软件可以使 用"kill-9 pid(进程 ID)"命令来关闭掉,首先使用"ps"命令查看要关闭的软件 PID 是多少, ps 命令用于查看所有当前正在运行的进程,并且会给出进程的 PID。输入"ps"命令,结果 如下图所示:

446	root	0:00	/sbin/getty 38400 ttyl	
460	root	0:00	/bin/login	
462	root	0:00	-sh	
484	root	0:00	[kworker/1:1-eve]	
508	root	0:00	[kworker/1:2-eve]	
509	root	0:00	[kworker/1:0-mm_]	
512	root	0:00	./hello	
513	root	0:00	ps	

图 19.4.4 ps 命令结果

从上图可以看出 hello 程序对应的 PID 为 512,因此我们使用如下命令关闭在后台运行的 hello 软件: kill -9 512

因为 hello 在不断的输出 "hello world" 所以我们的输入看起来会被打断,其实是没有的, 因为我们是输入,而 hello 是输出。在数据流上是没有打断的,只是显示在串口终端上就好像 被打断了,所以只管输入"kill -9 512"即可。hello被 kill 以后会有提示,如下图所示:

root(ALIENT	EK-ZYN	IQ:/di	rivers	s# kill -9 512		
-sh:	kill:	(512)	- No	such	process		
[1]+	Kille	d			./hello		
root@ALIENTEK-ZYNQ:/drivers#							

图 19.4.5 提示 hello 被 kill 掉

再去用 ps 命令查看一下当前的进程,发现没有 hello 了。这个就是 Linux 下的软件后台运 行以及如何关闭软件的方法,重点就是3个操作:程序后面加"&"、使用 ps 查看要关闭的 软件 PID、使用"kill -9 pid"来关闭指定的软件。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

19.4.2 开机自启动测试

在上一小节测试 hello 软件的时候都是等 Linux 启动进入根文件系统以后手动输入命令 "./hello"来完成的。我们一般做好产品以后都是需要开机自动启动相应的软件,本节我们就 以 hello 这个软件为例, 讲解一下如何实现开机自启动。开机启动后进入根文件系统的时候会 运行/etc/init.d/rcS 这个 shell 脚本, Linux 内核启动以后需要启动一些服务, 而 rcS 就是规定启 动哪些文件的脚本文件。因此我们可以在这个脚本里面添加自启动相关内容。添加完成以后 的/etc/init.d/rcS 文件内容如下:

```
1 #!/bin/sh
2 #
3 \# rcS
           Call all S??* scripts in /etc/rcS.d in
4 #
        numerical/alphabetical order.
5 #
6 # Version: @(#)/etc/init.d/rcS 2.76 19-Apr-1999 miquels@cistron.nl
7 #
8
9 PATH=/sbin:/usr/sbin:/usr/bin
10 runlevel=S
11 prevlevel=N
12 umask 022
13 export PATH runlevel prevlevel
14
15 # Make sure proc is mounted
16 #
17 [ -d "/proc/1" ] || mount /proc
18
19 #
20 # Source defaults.
21 #
22 . /etc/default/rcS
23
24 #开机自启动
25 cd /drivers
26 ./hello &
27 cd
28
29 #
      Trap CTRL-C &c only in this shell so we can interrupt subprocesses.
30 #
31 #
32 trap ":" INT QUIT TSTP
33
34 #
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

35 # Call all parts in order.

36 #

37 exec /etc/init.d/rc S

第25行,进入 drivers 目录,因为要启动的软件存放在 drivers 目录下。

第26行,以后台方式执行 hello 程序。

第27行,退出 drivers 目录,进入到当前用户目录下。

自启动代码添加完成以后就可以重启开发板,看看 hello 这个软件会不会自动运行。结果 如下图所示:



图 19.4.6 hello 开机自启动

从上图可以看出, hello 程序开机自动运行了, 说明开机自启动成功。

19.4.3 网络连接测试

本小节的网络连接测试是指测试领航者开发板能否连接到互联网。需要提醒的是,测试 的领航者开发板需要和能连接到互联网的路由器用网线连接,如果领航者开发板是和我们的 主机电脑连接的话,本小节可跳过,因为这种情况下领航者开发板是不能连接到互联网的, 除非电脑主机有路由器的功能。

测试方法很简单,就是通过 ping 命令来 ping 百度的官网:<u>www.baidu.com</u>。输入如下命 令:

```
ping www.baidu.com
结果如下图所示:
root@zynq_linux:~# ping www.baidu.com
ping: bad address 'www.baidu.com'
root@zynq_linux:~#
```

图 19.4.7 ping 测试结果

可以看出,测试失败,提示 <u>www.baidu.com</u>是个"bad address",也就是地址不对,显然我们的地址是正确的。之所以出现这个错误提示是因为 <u>www.baidu.com</u>的地址解析失败,并没有解析出其对应的 IP 地址。我们需要配置域名解析服务器的 IP 地址,一般域名解析地址可



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

以设置为所处网络的网关地址,比如 192.168.2.1,也可以设置为公共 DNS 服务器地址如 114.114.114.114。

设置很简单,可以直接在串口终端中输入"udhcpc"命令,该命令可自动获取 IP 地址,并且修改 nameserver 的值,一般是将其设置为对应的网关地址。输入"udhcpc"命令后,执行结果如下图所示:

root@zynq_linux:~# udhcpc udhcpc (v1.24.1) started Sending discover... Sending select for 192.168.2.124... Lease of 192.168.2.124 obtained, lease time 86400 /etc/udhcpc.d/50default: Adding DNS 192.168.2.1 root@zynq_linux:~#

图 19.4.8 "udhcpc" 命令

从上图可以看到,执行"udhcpc"命令后,我们的领航者开发板获得了 IP 地址 "192.168.2.124",添加了 DNS 地址为网关地址"192.168.2.1"。现在我们重新 ping 一下百 度官网,结果如下图所示:

root@zynq_linux:~# ping www.baidu.com -c 4
PING www.baidu.com (180.101.49.11): 56 data bytes
64 bytes from 180.101.49.11: seq=0 ttl=53 time=6.294 ms
64 bytes from 180.101.49.11: seq=1 ttl=53 time=5.449 ms
64 bytes from 180.101.49.11: seq=2 ttl=53 time=5.514 ms
64 bytes from 180.101.49.11: seq=3 ttl=53 time=5.435 ms

--- www.baidu.com ping statistics ---4 packets transmitted, 4 packets received, 0% packet loss round-trip min/avg/max = 5.435/5.673/6.294 ms root@zynq_linux:~#

图 19.4.9 ping 百度官网结果

参数 "-c4"代表 ping 4 次就停止,否则无限次的 ping。从上图可以看出 ping 百度官网成功了。域名也成功的解析了,至此,我们的根文件系统就彻底的制作完成。需要说明的是本章制作的根文件系统是能满足基本需求的,如果想运行图形界面的其他功能,可以自行移植,推荐使用我们在第八章 Linux 显示设备的使用中使用的根文件系统。

uboot、Linux kernel、rootfs 这三个共同构成了一个完整的 Linux 系统,现在的系统至少是一个可以正常运行的系统,后面我们就可以在这个系统上完成 Linux 驱动开发的学习。

随着本章的结束,也宣告着本书第三篇的内容也正式结束了,第三篇是系统移植篇,重 点就是 uboot、Linux kernel 和 rootfs 的移植,看似简简单单的"移植"两个字,引出的却是一 篇 300 多页的"爱恨情仇"。授人以鱼不如授人以渔,本可以简简单单的教大家修改哪些文 件、添加哪些内容,怎么去编译,然后得到哪些文件。但是这样只能看到表象,并不能深入 的了解其原理,为了让大家能够详细的了解整个流程,笔者义无反顾的选择了这条最难走的 路,不管是 uboot 还是 Linux kernel,从 Makefile 到启动流程,都尽自己最大的努力去阐述清 楚。奈何,笔者水平有限,还是有很多的细节没有处理好,大家有疑问的地方可以到正点原 子论坛 www.openedv.com/forum.php



论坛:www.openedv.com/forum.php

正点原子

第二十章 搭建驱动开发使用的 ZYNQ 镜像

前面我们一直都是使用 petalinux 工具编译镜像文件,例如包括 u-boot、fsbl、linux 内核、设备树、ZYNQ PL 端的 bitstream 文件等。petalinux 功能上比较全面,但是编译速度较慢。本章采用分步式的方式编译启动开发板所需要的各种镜像文件,虽然步骤比较繁琐,但灵活性比较高。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

20.1 创建 Petalinux 工程

对于 Zynq 而言,一个完整的 linux 系统包含 PS 和 PL 两个构件,其中 PS 构件包含 fsbl、 uboot、设备树文件、linux 内核、根文件系统共 5 个要素,PL 构件包含 bit 文件一个要素,当 不使用 PL 的时候,该要素非必须。

编写 linux 驱动的时候,经常改动的要素有设备树文件、linux 内核、根文件系统,当然如 果改动 PL 的话还需改动 bit 文件。因而我们将这些要素独立出来,从而方便修改变更。也就 是说我们将 bit 文件从原先的 BOOT.BIN 文件独立出来,将 image.ub 文件分开为内核 zImage 和设备树 dtb。另外将根文件系统放到 SD 卡的 EXT4 分区,加快启动速度。下面我们正式开 始。

将开发板资料盘(A

盘)\4_SourceCode\3_Embedded_Linux\vivado_prj\Navigator_7010\system_wrapper.xsa 复制到 Ubuntu 目录下,例如共享文件夹"/mnt/hgfs/share18/xsa/Navigator_7010/"目录下,大家根据 自己实际情况选择。对于 7020 开发板来说,则将开发板资料盘(A

盘)\4_SourceCode\3_Embedded_Linux\vivado_prj\Navigator_7020\system_wrapper.xsa 复制到 Ubuntu 目录下。如下图所示:

zy@zy-virtual-machine:~\$ ls /mnt/hgfs/share18/xsa/Navigator_7010/ system_wrapper.xsa

图 20.1.1 复制资料盘 xsa 文件到 Ubuntu 中

创建 Petalinux 工程的步骤在第六章 Petalinux 设计流程实战中已讲解,本章就不细述。输入如下命令创建和配置工程:

sptl

//设置 petalinux 工作环境

petalinux-create -t project --template zynq -n ALIENTEK-ZYNQ-driver //创建 petalinux 工程 cd ALIENTEK-ZYNQ-driver //进入 petalinux 工程目录下

petalinux-config --get-hw-description /mnt/hgfs/share18/xsa/Navigator_7010/ //导入 xsa 文件

注: 后面更新 vivado 工程的时候,只需要从 vivado 工程中导出 xsa 文件,并在该

Petalinux 工程下使用"petalinux-config --get-hw-description <xsa 文件路径>"命令重新配置即可。

xsa 文件导入成功之后会自动弹出 petalinux 工程配置窗口,如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php



图 20.1.2 petalinux 工程配置窗口

进入 "Subsystem AUTO Hardware Settings ---> Serial Settings"选项,将"FSBL"和 "DTG"两项使用的串口改成"uart 0",如下图所示:



图 20.1.3 修改串口

保存配置。按四次"ESC"键返回顶层配置界面,进入"Image Packaging Configuration--->Root filesystem type (INITRD)"配置选项,将其更改为"EXT4 (SD/eMMC/SATA/USB)",如下图所示:



图 20.1.4 修改根文件存放位置

此时可以看到下面"Device node of SD device"选项的值变为"/dev/mmcblk0p2",表示 sd 卡第二个分区。

到这里, Petalinux 工程就配置好了。保存配置并退出。

20.1.1 生成 BOOT.BIN

运行如下命令编译 fsbl 和 uboot:

petalinux-build -c bootloader

petalinux-build -c u-boot

然后执行下面命令生成 BOOT.bin:

petalinux-package --boot --fsbl --u-boot --dtb no --force

注意,petalinux 工具在打包的时候会自动把 dtb 文件加进入,所以这里加入"--dtb no" 参数把 dtb 文件除去。

执行结果如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php zy@zy-virtual-machine:~/petalinux/Navigator_7010_v3/ALIENTEK-ZYNQ-driver\$ petalinu x-package --boot --fsbl --u-boot --dtb no --force INFO: Sourcing build tools INFO: File in BOOT BIN: "/home/zy/petalinux/Navigator 7010 v3/ALIENTEK-ZYNQ-driver /images/linux/zynq fsbl.elf" INFO: File in BOOT BIN: "/home/zy/petalinux/Navigator 7010 v3/ALIENTEK-ZYNQ-driver /images/linux/u-boot.elf" INFO: Generating Zynq binary package BOOT.BIN... ****** Xilinx Bootgen v2020.2 **** Build date : Nov 15 2020-06:11:24 ** Copyright 1986-2020 Xilinx, Inc. All Rights Reserved. [INF0] : Bootimage generated successfully INFO: Binary is ready. WARNING: Unable to access the TFTPB00T folder /tftpboot!!! WARNING: Skip file copy to TFTPBOOT folder!!! 图 20.1.5 生成 BOOT.bin

从图 20.1.5 可以看出,BOOT.BIN 包含 zynq_fsbl.elf 和 u-boot.elf 两个文件,没有包含 bit 文件,这两个文件我们基本不会再改动,所以后面无论编写任何 linux 驱动以及更改 Vivado 工程,都不会改动 BOOT.BIN 文件了。

20.1.2 生成 boot.scr

最后执行"petalinux-build"命令编译工程,并生成 boot.scr 脚本文件,如下图所示:


原子哥在线教学	: www.yuanzige.com	论坛:www.openedv	.com/forum.pl	hp 📃	
zy@zy-virtual-	machine:~/petalinux/Nav	/igator_7010_v3/AL	IENTEK-ZYNQ-	driver\$ peta	li
nux-build 🔶					
INFO: Sourcing	build tools				
[INFO] Buildin	g project				
[INFO] Sourcin	g build environment				
[INFO] Generat	ing workspace directory	/			
INFO: DITDAKE	petalinux-image-minimal			T :	~~
Loading cache:		<i></i>	*############	Time: 0:00:0	90
Loaded 4264 en	tries from dependency c	cache.			00
Parsing recipe	5: 100% ################ 5	"#####################################	+############## 	Time: 0:00:0	93 64
inned 0 macka	d A orrors	1995 Cacheu, 2 par	seu). 4205 i	argets, 204	sк
NOTE: Pecolvin	a any missing task queu	le dependencies			
Initialicing t	y any missing task queu acks: 100% ############		<i></i>	Time: 0.00.	ດວ
Checking sstat	e mirror object availah	oilitv∙ 100% ####	************	Time: 0.00.	02
Sstate summarv	· Wanted 127 Found 92 M	lissed 35 Current	841 (72% mat	ch 96% comp	le
te)		HISSER SS carrent	041 (720 mat	.en, 500 comp	
NOTE: Executin	g Tasks				
NOTE: Setscene	tasks completed				
NOTE: Tasks Su	mmary: Attempted 3509 t	asks of which 292	20 didn't nee	ed to be reru	n
and all succee	ded.				
INFO: Failed t	o copy built images to	tftp dir: /tftpbc	oot		
[INF0] Success	fully built project				
zy@zy-virtual-	<pre>machine:~/petalinux/Nav</pre>	/igator_7010_v3/AL	IENTEK-ZYNQ-	driver\$ ls in	na
ges/linux/					
BOOT.BIN	rootfs.cpio	rootfs.manifest	u-boot.bin	zImage	
boot.scr	rootfs.cpio.gz	rootfs.tar.gz	u-boot.elf	zynq_fsbl.el	f
image.ub	rootfs.cpio.gz.u-boot	system.bit	uImage		
pxelinux.cfg	rootfs.jffs2	system.dtb	vmlinux		

图 20.1.6 生成 boot.scr

这里需要对 boot.scr 做一些修改。

复制 boot.scr 并命名为 boot.cmd.default,如下图所示:

<pre>zy@zy-virtual-machine:~/petalinux/Navigator_7010_v3/ALIENTEK-ZYNQ-driver/images/ linux\$ cp boot.scr boot.cmd.default zy@zy-virtual-machine:~/petalinux/Navigator_7010_v3/ALIENTEK-ZYNQ-driver/images/</pre>				
linux\$ ls				
BOOT.BIN	rootfs.cpio	<pre>rootfs.tar.gz</pre>	uImage	
<pre>boot.cmd.default</pre>	rootfs.cpio.gz	system.bit	vmlinux	
boot.scr	rootfs.cpio.gz.u-boot	system.dtb	zImage	
image.ub	rootfs.jffs2	u-boot.bin	zynq fsbl.elf	
pxelinux.cfg	rootfs.manifest	u-boot.elf		

图 20.1.7 创建 boot.cmd.default 文件

打开 boot.cmd.default 文件,将光标移动到第一行,输入"dd"命令删除第一行,删除后 如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php



图 20.1.8 删除第一行

然后将第19行和第20行内核镜像名 uImage 改成 zImage,同时第30行 bootm 改成 bootz,如下图所示:



图 20.1.9 修改内核镜像名和启动命令

接下来在第 29 行添加 system.bit 的加载和启动命令,命令如下: if test -e \${devtype} \${devnum}:\${distro_bootpart} /system.bit; then fatload \${devtype} \${devnum}:\${distro_bootpart} 0x00800000 system.bit; fpga loadb 0 \${fileaddr} \${filesize} fi



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php 添加后如下图所示:

28	exit;
29	fi
30	if test -e \${devtype} \${devnum}:\${distro_bootpart} /system.b
	it; then
31	fatload \${devtype} \${devnum}:\${distro bootpart} 0x00
	800000 system.bit;
32	<pre>fpga loadb 0 \${fileaddr} \${filesize}</pre>
33	fi
34	bootz 0x00200000 - 0x00100000
35	exit;
20	f :

图 20.1.10 添加 bit 文件启动命令

修改完成后保存退出,运行如下命令重新生成 boot.scr 文件:

mkimage -c none -A arm -T script -d boot.cmd.default boot.scr

结果如下图所示:

zy@zy-virtual-	machine:~/petalinux/Navigat	tor_7010_v3/ALI	ENTEK-ZYNO-dr	<pre>iver/images/l</pre>
<pre>inux\$ mkimage</pre>	-c none -A arm -T script -d	d boot.cmd.defa	ult boot.scr	
Image Name:				
Created:	Wed Jun 7 11:42:31 2023			
Image Type:	ARM Linux Script (gzip com	pressed)		
Data Size:	2135 Bytes = 2.08 KiB = 0.0	90 MiB		
Load Address:	0000000			
Entry Point:	0000000			
Contents:				
Image 0: 21	27 Bytes = 2.08 KiB = 0.00	MiB		
zy@zy-virtual-	machine:~/petalinux/Naviga	tor_7010_v3/ALI	ENTEK-ZYNQ-dr	<code>iver/images/l</code>
inux\$ ls				
BOOT.BIN	rootfs.cpio	<pre>rootfs.tar.gz</pre>	uImage	
boot.cmd.defau	lt rootfs.cpio.gz	system.bit	vmlinux	
boot.scr	rootfs.cpio.gz.u-boot	system.dtb	zImage	
image.ub	rootfs.jffs2	u-boot.bin	zynq_fsbl.el	.f
pxelinux.cfg	rootfs.manifest	u-boot.elf		

图 20.1.11 重新生成 boot.scr

20.2 设备树文件

在 Petalinux 工程中执行编译 uboot 后, 会在工程的 "components/plnx_workspace/devicetree/device-tree/"目录下生成设备树文件,如下图所示:

<pre>zy@zy-virtual-machine:~/petalinux/Navigator_7010_v3/ALIENTEK-ZYNQ-driver/c omponents/plnx workspace/device-tree/device-tree\$ ls</pre>				
device-tree.mss	ps7_init.c	skeleton.dtsi		
hardware_description.xsa	ps7_init_gpl.h	system-top.dts		
include	ps7_init.h	system_wrapper.bit		
pl.dtsi	ps7_init.tcl	2ynq-7000.dtsi		

图 20.2.1 设备树文件



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

上图中的设备树文件是 Petalinux 根据 xsa 文件产生,我们在前面 Uboot 移植和内核移植 章节中有使用到。其中红框中的设备树将在下一小节编译内核中用到。

注意,设备树文件只有编译完成后才会产生,例如执行编译 uboot。

20.3 编译内核

本小节不使用前面移植好的 Linux 内核源码,我们用一份新的 Xilinx 官方 2020.2 版本内核 源码(这个版本是 Xilinx 设定的版本,其 Linux 版本为 5.4.0),源码已经提供给大家了,路 径为:开发板资料盘(A 盘)\4_SourceCode\3_Embedded_Linux\资源文件\Kernel\linux-xlnxxlnx_rebase_v5.4_2020.2.tar.gz。大家也可以通过 <u>https://github.com/Xilinx/linux-xlnx</u>网址进行 下载,如下图所示:

Xilinx / linux-xlnx Public			
<> Code	រ៉ៀ Pull re	quests 2 🕑 Actions 🖽 Pro	jects 🤃
		<mark>알 master →</mark> 알 33 branches	🔊 184 tag
		Switch branches/tags	×
		Find a tag	
		Branches Tags	
		xlnx_rebase_v5.10_2021.1_update1	^
	:	2 xlnx_rebase_v5.4_2020.2	

图 20.3.1 下载内核源码

将内核源码压缩包文件复制到 ubuntu 系统中,笔者这里放到/home/zy/worksapce/kernel-driv er 目录下,然后解压文件压缩包,生成 linux-xlnx-rebase_v5.4_2020.2 文件夹,如下图所示:

```
zy@zy-virtual-machine:~/worksapce/kernel-driver$ ll
总用量 169540
drwxrwxr-x 2 zy zy
                        4096 6月
                                  5 19:11 ./
drwxrwxr-x 3 zy zy
                        4096 5月
                                  29 16:34 .../
                                 5 17:07 linux-xlnx-xlnx rebase v5.4 2020.2.ta
-rwxrwxr-x 1 zy zy 173593124 6月
r.gz*
zy@zy-virtual-machine:~/worksapce/kernel-driver$ tar -xzf linux-xlnx-xlnx rebase
v5.4 2020.2.tar.gz
zy@zy-virtual-machine:~/worksapce/kernel-driver$ ll
总用量 169544
                         4096 6月
                                    5 19:11 ./
           3 zy zy
drwxrwxr-x
drwxrwxr-x 3 zy zy
                         4096 5月
                                   29 16:34
                                  4 2020 linux-xlnx-xlnx rebase v5.4 2020.2/
drwxrwxr-x 25 zy zy
                         4096 11月
-rwxrwxr-x 1 zy zy 173593124 6月
                                    5 17:07 linux-xlnx-xlnx rebase v5.4 2020.2.t
ar.gz*
```

图 20.3.2 解压内核压缩包



原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php 下面开始编译内核。

20.3.1 添加设备树文件

将 20.2 小节中 pcw.dtsi, pl.dtsi, system-top.dts, zynq-7000.dtsi 和 system-conf.dtsi 五个设 备树文件直接复制到内核源码"arch/arm/boot/dts"目录下,如下图所示:

zy@zy-virtual-machine:~/petalinux/Navigator_7010_v3/ALIENTEK-ZYNQ-driver/compon ents/plnx_workspace/device-tree/device-tree\$ cp pcw.dtsi pl.dtsi system-top.dts zyng-7000.dtsi system-conf.dtsi /home/zy/worksapce/kernel-driver/linux-xlnx-xl nx rebase v5.4 2020.2/arch/arm/boot/dts/

图 20.3.3 添加设备树文件

此外还要将工程"project-spec/meta-user/recipes-bsp/device-tree/files"目录下设备树文件 "svstem-user.dtsi"复制到内核源码"arch/arm/boot/dts"目录下,如下图所示:

zy@zy-virtual-machine:~/petalinux/Navigator 7010 v3/ALIENTEK-ZYNQ-driver/projec t-spec/meta-user/recipes-bsp/device-tree/files\$ cp system-user.dtsi /home/zy/wo rksapce/kernel-driver/linux-xlnx-xlnx rebase v5.4 2020.2/arch/arm/boot/dts/

图 20.3.4 添加设备树文件

接下来对"system-user.dtsi"进行修改,修改后的内容如下:

示例代码 修改后的 system-user.dtsi

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

- 2 #include <dt-bindings/gpio/gpio.h>
- 3 #include <dt-bindings/input/input.h>
- 4 #include <dt-bindings/media/xilinx-vip.h>
- 5 #include <dt-bindings/phy/phy.h>
- 6

```
7 / {
```

```
8 model = "Alientek Navigator Zynq Development Board";
```

```
9 compatible = "xlnx,zynq-zc702", "xlnx,zynq-7000";
```

10

```
11 chosen {
```

```
bootargs = "console=ttyPS0,115200 earlycon root=/dev/mmcblk0p2 rw rootwait";
12
```

```
stdout-path = "serial0:115200n8";
13
```

```
14 };
```

15

- 16 };
- 17

```
18 &uart0 {
```

```
19 u-boot, dm-pre-reloc;
```

```
20 status = "okay";
```

```
21 };
```

22



论坛:www.openedv.com/forum.php n

٦	千骨在线教学: www.yuanzige.com
	23 &sdhci0 {
	24 u-boot,dm-pre-reloc;
	25 status = "okay";
	26 };
	27
	28 &gem0 {
	29 local-mac-address = [00 0a 35 00 8b 87];
	30
	31 phy-handle = <ðernet_phy>;
	32 ethernet_phy: ethernet-phy@7 { /* yt8521 */
	33 reg = $<0x7>$;

34 device_type = "ethernet-phy";

35 };

36 };

首先注释掉第一行"system-conf.dtsi"设备树文件的引用,并将该设备树文件中 bootargs 变量直接放到 system-user.dtsi 中。第8行 model 属性表示设备名称。第9行 compatible 表示 兼容性属性,和驱动程序有关,只有和驱动匹配,才能使用该节点。第11行添加"chosen" 节点,并增加 bootargs 和 stdout-path 属性。

第18行到第36行,分别对 uart0 串口、sd0 控制器和 gem0 以太网控制器节点增加了部 分属性。

修改完成后保存退出。

下面修改 arch/arm/boot/dts 目录下 Makefile 文件,将添加后的设备树顶层文件放到 Makefile 中,如下图所示:

1200	wm8850-w70v2.dtb
1201	<pre>dtb-\$(CONFIG_ARCH_ZYNQ) += \</pre>
1202	zynq-cc108.dtb \
1203	zynq-microzed.dtb \
1204	zynq-parallella.dtb \
1205	zynq-zc702.dtb \
1206	zynq-zc706.dtb \
1207	zynq-zc770-xm010.dtb \
1208	zynq-zc770-xm011.dtb \
1209	zynq-zc770-xm012.dtb \
1210	zynq-zc770-xm013.dtb \
1211	zynq-zed.dtb \
1212	zynq-zturn.dtb \
1213	zynq-zybo.dtb \
1214	zynq-zybo-z7.dtb \
1215	system-top.dtb
1216	dtb-\$(CONFIG_MACH_ARMADA_370) += \
1217	armada-370-db.dtb \

图 20.3.5 Makefile 中添加设备树

修改完成后保存退出。

领航者 ZYNQ 之嵌入式 Linux 开发指南 原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php 20.3.2 运行 defconfig 配置 首先设置 SDK 环境变量。如果已经按照 10.4 小节在 bashrc 文件中添加 SDK 环境变量,则不用再手动设置了。 然后在内核源码根目录下执行如下所示命令,对内核进行配置:

make xilinx_zynq_defconfig

执行结果如下图所示:

<pre>zy@zy-virtual-machine:~/worksapce/kernel-driver/linux-xlnx_rebase_v</pre>	5.4_2020
<pre>.2\$ make xilinx_zynq_defconfig </pre>	
HOSTCC scripts/basic/fixdep	
HOSTCC scripts/kconfig/conf.o	
HOSTCC scripts/kconfig/confdata.o	
HOSTCC scripts/kconfig/expr.o	
LEX scripts/kconfig/lexer.lex.c	
YACC scripts/kconfig/parser.tab.[ch]	
HOSTCC scripts/kconfig/lexer.lex.o	
HOSTCC scripts/kconfig/parser.tab.o	
HOSTCC scripts/kconfig/preprocess.o	
HOSTCC scripts/kconfig/symbol.o	
HOSTLD scripts/kconfig/conf	
#	
# configuration written to .config	
<pre># zv@zv-virtual-machine:~/worksapce/kernel-driver/linux-xlnx-xlnx rebase v</pre>	5.4 2020
.2\$	

图 20.3.6 配置内核

20.3.3 menuconfig 配置

这里我们暂时不进行配置,先保持默认。

20.3.4 编译内核

执行下面命令编译内核源码: make-j8 执行结果如下图所示:



,	、于骨在线	教学: www.yuanzige.com 论坛:www.openedv.com/forum.php 📃
	zy@zy-vir	tual-machine:~/worksapce/kernel-driver/linux-xlnx-xlnx_rebase_v5.4_2020
	.2\$ make	-j8 🛻
	SYSHDR	arch/arm/include/generated/uapi/asm/unistd-oabi.h
	SYSHDR	arch/arm/include/generated/uapi/asm/unistd-eabi.h
	SYSHDR	arch/arm/include/generated/uapi/asm/unistd-common.h
	WRAP	arch/arm/include/generated/uapi/asm/kvm_para.h
	WRAP	arch/arm/include/generated/uapi/asm/bitsperlong.h
	WRAP	arch/arm/include/generated/uapi/asm/errno.h
	WRAP	arch/arm/include/generated/uapi/asm/bpf_perf_event.h
	WRAP	arch/arm/include/generated/uapi/asm/ioctl.h
	WRAP	arch/arm/include/generated/uapi/asm/ipcbuf.h
	WRAP	arch/arm/include/generated/uapi/asm/param.h
	WRAP	arch/arm/include/generated/uapi/asm/msgbuf.h
	WRAP	arch/arm/include/generated/uapi/asm/poll.h

图 20.3.7 编译内核

AS	arch/arm/boot/compressed/hyp-stub.o
AS	arch/arm/boot/compressed/lib1funcs.o
AS	arch/arm/boot/compressed/ashldi3.o
AS	arch/arm/boot/compressed/bswapsdi2.o
AS	arch/arm/boot/compressed/piggy.o
LD	arch/arm/boot/compressed/vmlinux
OBJCOPY	arch/arm/boot/zImage
Kernel:	arch/arm/boot/zImage is ready
zy@zy-vir	tual-machine:~/worksapce/kernel-driver/linux-xlnx-xlnx rebase v5.4 2020
.2\$ ls ar	ch/arm/boot/
bootp	deflate xip data.sh Image Makefile
compresse	d dts install.sh zImage
zy@zy-vir	tual-machine:~/worksapce/kernel-driver/linux-xlnx-xlnx rebase v5.4 2020
.2\$ ls ar	ch/arm/boot/dts/system-top*
arch/arm/l	<pre>boot/dts/system-top.dtb arch/arm/boot/dts/system-top.dts</pre>
zy@zy-vir	tual-machine:~/worksapce/kernel-driver/linux-xlnx-xlnx rebase v5.4 2020
.2\$	

图 20.3.8 内核编译完成

内核编译完成后会在 arch/arm/boot 目录下生成内核镜像文件 zImage,在 arch/arm/boot/dts 目录下生成设备树镜像文件 system-top.dtb,如图 20.3.8 所示。

额外提一下,如果内核源码中只修改了设备树,可以只编译设备树。在内核源码根目录 下输入如下命令编译设备树:

make dtbs

命令"make dtbs"将.dts编译成.dtb,编译完成后就可以使用新的设备树镜像文件。

20.4 编译根文件系统

进入 20.1 小节的 petalinux 工程中, 输入如下命令配置根文件系统:

petalinux-config -c rootfs

这里我们使用默认配置,不做修改。

退出配置界面,输入如下命令编译根文件系统:

petalinux-build -c rootfs

编译结果如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
<pre>zy@zy-virtual-machine:~/petalinux/Navigator_7010_v3/ALIENTEK-ZYNQ-driver\$ petali</pre>
nux-build -c rootfs
INFO: Sourcing build tools
[INFO] Bullaing rootts
[INFO] Sourcing build environment [INFO] Cenerating workspace directory
INFO: bitbake petalipux-image-minimal -c do image complete
loading cache: 100% ###################################
Loaded 4264 entries from dependency cache.
Parsing recipes: 100% ###################################
Parsing of 2995 .bb files complete (2993 cached, 2 parsed). 4265 targets, 204 sk
ipped, 0 masked, 0 errors.
NOTE: Resolving any missing task queue dependencies
Initialising tasks: 100% ###################################
Checking sstate mirror object availability: 100% ###################################
Sstate summary: wanted 297 Found 88 Missed 209 Current 601 (29% match, 76% complete)
ele) NOTE: Executing Tasks
NOTE: Setscene tasks completed
NOTE: linux-xlnx: compiling from external source tree /home/zv/petalinux/Navigat
or 7010 v3/ALIENTEK-ZYNQ-driver/components/yocto/workspace/sources/linux-xlnx
NOTE: Tasks Summary: Attempted 2700 tasks of which 2038 didn't need to be rerun
and all succeeded.
INFO: Failed to copy built images to tftp dir: /tftpboot
[INFO] Successfully built rootfs
zy@zy-virtual-machine:~/petalinux/Navigator_7010_v3/ALIENTEK-ZYNQ-driver\$
图 20.4.1 编译根文件系统
编译成功后,在 images/linux 目录下生成根文件系统压缩包,如下图所示:
zy@zy-virtual-machine:~/petalinux/Navigator 7010 v3/ALIENTEK-ZYNQ-driver\$ cd ima
ges/linux/
<pre>zy@zy-virtual-machine:~/petalinux/Navigator_7010_v3/ALIENTEK-ZYNQ-driver/images/ linux\$ ls</pre>
BOOT.BIN rootfs.cpio.gz.u-boot system.bit uImage
<pre>image.ub rootfs.jffs2 system.dtb vmlinux</pre>
<pre>rootfs.cpio rootfs.manifest u-boot.bin zImage</pre>
<pre>rootfs.cpio.gz rootfs.tar.gz u-boot.elf zynq_fsbl.elf</pre>
<pre>zy@zy-virtual-machine:~/petalinux/Navigator_7010_v3/ALIENTEK-ZYNQ-driver/images/</pre>

图 20.4.2 生成根文件系统压缩包

这里我们使用 rootfs.tar.gz 文件。将 rootfs.tar.gz 解压到 SD 卡 ext4 分区即可。

20.5 启动开发板

在第六章实验中,我们使用 petalinux 工具生成的 BOOT.BIN, boot.scr 和 image.ub 三个 文件启动开发板。为了方便后面驱动开发,本章前面几小节,我们生成单独的内核镜像文 件、设备树镜像文件和根文件系统文件代替 image.ub,同时将 system.bit 从 BOOT.BIN 中独 立出来。

从本章开始,我们使用 BOOT.BIN, boot.scr, system.bit, zImage, system.dtb 和根文件 系统这六个文件启动开发板。

Step1: 制作 SD 启动卡



原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php

参考 6.3.10 小节制作 SD 启动卡,制作好后使用 "df -Th" 命令查看 SD 卡分区信息,如下 图所示:

tmpfs	tmpfs	1.2G	16K	1.2G	1% /run/user/121
tmpfs	tmpfs	1.2G	32K	1.2G	1% /run/user/1000
/dev/sdb1	vfat	4.0G	4.0K	4.0G	1% /media/zy/boot
/dev/sdb2	ext4	11G	24K	11G	1% /media/zy/rootfs
zy@zy-virtua	al-machine:~\$		2.00		10, modiu, 29, 100000

图 20.5.1 查看 SD 卡分区信息

Step2: 拷贝镜像到 FAT 分区

将前面过程当中生成的各种镜像文件拷贝到 SD 启动卡的 FAT 分区,包括 20.3 小节内核 源码目录下 zImage 和 system-top.dtb 镜像文件,20.1 小节 petalinux 工程下 system.bit、 BOOT.BIN 和 boot.scr 文件。大家根据自己前面步骤当中文件存放的目录去找到相应的镜像文件,注意复制之前确保 FAT 分区没有上述文件。

复制完成后,看看我们的FAT分区有哪些文件:

<pre>zy@zy-virtual-machine:~\$ ls -l /media/zy/boot/</pre>									
总用量 6976									
- rw- r	r	1	zy	zy	846796	6月	6	13:27	BOOT.BIN
- rw- r	r	1	zy	zy	2010	6月	6	13:59	boot.scr
- rw- r	r	1	zy	zy	2083850	6月	6	13:27	system.bit
- rw- r	r	1	zy	zy	17338	6月	6	13:35	system-top.dtb
- rw- r	r	1	zy	zy	4183488	6月	6	13:27	zImage

图 20.5.2 SD 卡 fat 分区中的文件

然后修改设备树文件名,运行如下命令,将 system-top.dtb 重命名为 system.dtb: mv system-top.dtb system.dtb

结果如下图所示:

zy@zy-virtual-machine:/media/zy/boot\$ ls
BOOT.BIN boot.scr system.bit system-top.dtb zImage
<pre>zy@zy-virtual-machine:/media/zy/boot\$ mv system-top.dtb system.dtb</pre>
<pre>zy@zy-virtual-machine:/media/zy/boot\$ ls</pre>
BOOT.BIN boot.scr system.bit system.dtb zImage
zy@zy-virtual-machine:/media/zy/boot\$

图 20.5.3 设备树重命名

Step3: 将根文件系统解压到 SD 卡 ext4 分区

接下来我们需要将 20.4 小节编译的根文件系统压缩包解压到 SD 启动卡的 EXT4 分区,这 里笔者使用 rootfs.tar.gz 压缩包文件,进入到 rootfs.tar.gz 压缩包文件所在目录,执行解压命令 (解压之前确保 EXT4 分区没有根文件系统):

```
sudo tar -xzf rootfs.tar.gz -C /media/zy/rootfs
结果如下图所示:
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

图 20.5.4 根文件系统解压到 ext4 分区

/media/zy/rootfs 是笔者 SD 启动卡对应的 ext4 分区的挂载点,解压时需要使用 sudo,也就 是要以 root 权限进行解压。解压完成之后执行 sync 命令将数据同步到 SD 卡中,之后卸载 SD 启动卡。

Step4: 启动开发板

将 SD 启动卡插入开发板,启动模式设置为 sd 卡启动,连接串口,最后连接电源,并给 开发板上电,开发板启动后打印信息如下所示:



原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php

U-Boot 2020.01 (Jun 01 2023 - 10:01:41 +0000) Zynq 7z010 CPU: Silicon: v3.1 DRAM: ECC disabled 512 MiB Flash: 0 Bytes NAND: 0 MiB MMC: mmcGe0100000: 0.mmc(MMC: mmc@e01000000: 0, mmc@e01010000: 1 Loading Environment from SPI Flash... SF: Detected w25q256 with page size 256 Bytes, erase size 4 KiB, to MMC: tal 32 MiB *** Warning - bad CRC, using default environment In: serial@e0000000 Out: serial@e0000000 Err: serial@e0000000 Net: ZYNQ GEM: e000b000, mdio bus e000b000, phyaddr -1, interface rgmii-id warning: ethernet@e000b000 using MAC address from DT eth0: ethernet@e000b000 ZYNQ GEM: e000c000, mdio bus e000c000, phyaddr -1, interface gmii Could not get PHY for eth1: addr -1 Hit any key to stop autoboot: 0 switch to partitions #0, OK mmc0 is current device Scanning mmc 0:1.. Found U-Boot script /boot.scr 2199 bytes read in 15 ms (142.6 KiB/s) ## Executing script at 0300000 4183488 bytes read in 247 ms (16.2 MiB/s) 17338 bytes read in 19 ms (890.6 KiB/s) 2083850 bytes read in 127 ms (15.6 MiB/s) design filename = "system wrapper;UserID=0XFFFFFFF;Version=2020.2" part purples = #3201021a400" part number = "7z010clg400" date = "2023/03/22" time = "18:25:05" bytes in bitstream = 2083740 zynq_align_dma_buffer: Align buffer at 80006e to 7fff80(swap 1) INFO:post config was not run, please run manually if needed
Flattened Device Tree blob at 00100000
Booting using the fdt blob at 0x100000
Loading Device Tree to leaff000, end leb063b9 ... 0K Starting kernel ... Booting Linux on physical CPU 0x0 Linux version 5.4.0-xilinx (zy@zy-virtual-machine) (gcc version 9.2.0 (GCC)) #1 SMP PREEMPT Tue Jun 6 09: 58:51 CST 2023 CPU: ARMv7 Processor [413fc090] revision 0 (ARMv7), cr=18c5387d

图 20.5.5 启动后打印信息

输入用户名和密码后进入系统,如下图所示:



原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php

udevd[106]: starting version 3.2.8 random: udevd: uninitialized urandom read (16 bytes read) random: udevd: uninitialized urandom read (16 bytes read) random: udevd: uninitialized urandom read (16 bytes read) udevd[107]: starting eudev-3.2.8 FAT-fs (mmcblk0p1): Volume was not properly unmounted. Some data may be corrupt. Please run fsck EXT4-fs (mmcblk1p1): recovery complete EXT4-fs (mmcblk1p1): mounted filesystem with ordered data mode. Opts: (null) EXT4-fs (mmcblk0p2): re-mounted. Opts: (null) hwclock: can't open '/dev/misc/rtc': No such file or directory Wed Jun 7 03:03:44 UTC 2023 hwclock: can't open '/dev/misc/rtc': No such file or directory urandom_read: 2 callbacks suppressed random: dd: uninitialized urandom read (512 bytes read) Configuring packages on first boot.... (This may take several minutes. Please do not power off the machine.) Running postinst /etc/rpm-postinsts/100-sysvinit-inittab... update-rc.d: /etc/init.d/run-postinsts exists during rc.d purge (continuing) Removing any system startup links for run-postinsts ... /etc/rcS.d/S99run-postinsts INIT: Entering runlevel: 5 Configuring network interfaces... macb e000b000.ethernet eth0: link up (1000/Full) IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready udhcpc: started, v1.31.0 udhcpc: sending discover udhcpc: sending discover udhcpc: sending discover udhcpc: no lease, forking to background done. Starting haveged: haveged: listening socket at 3 haveged: haveged starting up Starting Dropbear SSH server: random: dropbearkey: uninitialized urandom read (32 bytes read) random: dropbearkey: uninitialized urandom read (32 bytes read) random: dropbearkey: uninitialized urandom read (32 bytes read) Generating 2048 bit rsa key, this may take a while... random: dropbearkey: uninitialized urandom read (32 bytes read) Public key portion is: ssh-rsa AAAAB3NzaClyc2EAAAADAQABAAABAQCgVtYeIme910A9JI0pq65/VcypGD7CteEs1A6DcOmKCeKlMpYVCjGMr2BS) MPcb1xs1m2eUUlFFcuzJYU05xPb90KlBVehY/xBVfZAF3bCRDH1xW8BpB/uatlFyQSwT+pKmcKr/GPQ1EuLKDoe9Qwupo66F0 Q9Q/q0Z0ZJf510Fu3GsfcEjzLf03SfgZTbmViz2R/6XrltvTC8CHRfEWyEzop3/bqVUxr7glE9DZUqvK8BSThpNrtfbVkiJK0 eiuv3qaq/TeQH3+Wo2vNbiXrQ0nACRWgoBx8nN3g50Jy4nC1/2WrZ1lXltA3glhhh root@ALIENTEK-ZYNQ-driver Fingerprint: shal!! 82:b5:de:ef:d7:41:cc:46:d9:56:a0:a9:3c:a1:9d:49:16:bf:8e:8e dropbear. hwclock: can't open '/dev/misc/rtc': No such file or directory Starting internet superserver: inetd. Starting syslogd/klogd: done Starting tcf-agent: OK PetaLinux 2020.2 ALIENTEK-ZYNQ-driver /dev/ttyPS0 ALIENTEK-ZYNQ-driver login: random: crng init done random: 1 urandom warning(s) missed due to ratelimiting PetaLinux 2020.2 ALIENTEK-ZYNQ-driver /dev/ttyPS0 ALIENTEK-ZYNQ-driver login: root 🔶 Password: root@ALIENTEK-ZYNQ-driver:~#

图 20.5.6 登录名和密码都是 root

20.6 NFS 挂载根文件并启动开发板

我们在 20.5 小节步骤 3(Step3)中将根文件系统放到 SD 卡 ext 分区,为了方便后面开发 驱动,我们换一种方式,通过 NFS 方式挂载根文件系统。NFS(Network File Network)也就



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

是网络文件系统,将根文件系统放在 Ubuntu 主机中,开发板通过网络挂载放在 Ubuntu 中的根文件系统,方便驱动程序开发。

20.6.1 设置网络环境

本小节需要设置开发板、Windows 系统和 Ubuntu 虚拟机的网络环境。

1、设置 Ubuntu 虚拟机网络环境

打开 Ubuntu 系统设置界面,找到网络配置选项,然后点击箭头处的网络配置按钮,如下 图所示:

Q 设置	网络	● ® ⊗
● 背景	有线连接	打开网络设置选项
🖸 Dock	已连接 - 1000 Mb/秒	打开 📄 🌣
▲ 通知		
Q 搜索	VPN	+
◎ 区域和语言	未设置	
● 通用辅助功能	网络代理	¥ 🏟
● 在线帐户		
≝ 隐私		
◀ 共享		
●》 声音		
С∉ 电源		
₽ 网络		

图 20.6.1 打开网络设置界面

在打开的有线网络配置界面中,首先切换到 IPv4 标签页,然后将 IPv4 方式设置为手动, 在地址栏中输入 IP 地址、子网掩码和网关,最后点击"应用"按钮,如下图所示

②正点原子

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

取消(C)	有线		应用(A)
详细信息 身份 IPv4 IP	2v6 安全	/	
IPv4 方式 2 C	〕自动 (DHCP) ▶手动	○ 仅本地链路 ○ 禁用	
地址 3 地址	子网掩码	网关	
192.168.1.11	255.255.255.0	192.168.1.1	8
			8
DNS		自动打开	Ŧ
使用逗号分隔 IP 地址			
路由		自动打开	Ŧ

图 20.6.2 配置有线网络

返回到系统配置界面,点击箭头处的开关按钮重新打开有线网络,如下图所示:

Q 设置	网络	
◎ 区域和语言	右线连接	±
✿ 通用辅助功能	日 按 注 - 1000 Mb/秒	T E
● 在线帐户		•
≝ 隐私	VPN	+
ሩ 共享	未设置	
●) 声音	网络代理 关 🔀	8
ጬ 电源		
₽ 网络		
현 设备 💦 👌		

图 20.6.3 重新打开有线连接



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

点击虚拟机菜单栏"编辑-->虚拟网络编辑器",在打开的设置界面中,将 VMnet0 桥接 目标设置为本地以太网网卡,注意这里不要连接到无线网网卡。设置好后点击"确定",如 下图所示:

🖳 虚拟网络	各编辑器				>
名称	类型	外部连接	主机连接	DHCP	子网地址
VMnet0	桥接模式	Realtek PCIe GbE Family Co	-	-	-
VMnet1	仅主机	-	已连接	已启用	192.168.92.0
VMnet8	NAT 模式	NAT 模式	已连接	已启用	192.168.149.0
			添加网络(E)	移除网络	(O) 重命名网络(W)
VMnet信息					
● 桥接模	。 [式(将虚拟树	几直接连接到外部网络)(B)			
口桥拉	至(G): Per	Itek PCIe ChE Eamily Controller			> 白动沿罟(1)
		Intek POIe ODE Failing Controller			·
◯NAT 模	式(与虚拟机		NAT 设置(S)		
○仅主机	.模式 <mark>(</mark> 在专用	月网络内连接虚拟机)(H)			
── 将主机	虎拟话函哭				
主机虎	拟话西器纲	3称: VMware 网络话配器 VMneti)		
一体田本		冬悠 m 抽屉公面必占切如 /m)			DUCD 辺里(D)
4	OR DHOP HE	方有 - 地址为肖娟虚拟机(U)			Uncr 反应(P)
子网 IP (I));	子网掩码(M):			
还原默认设	と <mark>置(R)</mark> 특	导入(T) 导出(X)	▲ 确定	限消	应用(A) 帮助

图 20.6.4 VMnet0 桥接至以太网网卡

如果打开后提示需要"管理员权限才能修改",则关掉虚拟机软件,重新以管理员身份 运行软件即可,如下图所示:



名称	类型	外部连接	主机连接	DHCP	子网地址
VMnet0 VMnet1 VMnet8	<u>自定义</u> 仅主机 NAT 模式	- - NAT 模式	- 已连接 已连接	- 已启用 已启用	192.168.10.0 192.168.92.0 192.168.149.0
			添加网络(E)	移除网络	络(O) 重命名
VMnet 信息 ○ 桥接模	】 式(将虚拟t	〔〕「「」」 〔〕」 〔〕」 [1] [1] [1] [1] [1] [1] [1] [1] [1] [1]	各)(B)		
已桥接	轾(G);				~ 自动设
◯ NAT 模	式(与虚拟树	ጊ共享主机的 IP 地址)	(N)		NAT is
◎ 仅主机	模式 (在专用	月网络内连接虚拟机)((H)		
 □ 将主机 主机處	虚拟适配器 拟话配器名	连接到此网络(V) 3称: VMware 网络话西	2器 VMnet0		
□使用本	地DHCP服	务将IP 地址分配给虚	封以机,(D)		DHCP 1
	400.45	10 0 75		0	

图 20.6.5 需要以管理员身份运行软件

2、设置 Windows 系统网络环境

参考领航者 FPGA 开发指南《MDIO 接口读写测试实验》下载验证部分,设置以太网网络适配器。

3、设置开发板网络环境

SD卡插入开发板并启动,出现 uboot 启动倒计时,按回车键进入 uboot 模式,如下图所示:



原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php



图 20.6.6 进入 uboot 模式

输入如下命令设置开发板网络环境变量:

setenv ipaddr 192.168.1.10	//开发板 ip 地址
setenv ethaddr 00:0a:35:00:1e:53	//开发板 mac 地址
setenv gatewayip 192.168.1.1	//开发板网关
setenv netmask 255.255.255.0	//开发板 ip 地址掩码
setenv serverip 192.168.1.11	//ubuntu ip 地址
saveenv	

设置完成后保存环境变量。此时如果能 ping 通 ubuntu 主机,则说明网络环境已经搭建好了,如下图所示:



图 20.6.7 设置网络环境变量

20.6.2 启动开发板

如果 Ubuntu 系统没有开启 NFS 服务,参考 4.4.1 小节进行相关设置。

进入 NFS 共享目录"/home/zy/workspace/nfs"下,新建 rootfs 文件夹,运行如下命令将 20.1 小节中根文件系统解压到该文件夹下:

tar -xzf /home/zy/petalinux/Navigator_7010_v3/ALIENTEK-ZYNQ-driver/images/linux/rootfs.tar.gz -C rootfs/



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

结果如下图所示:

Zy@Zy-VITtuat-machine:~/workspace/miss mkuir rootis	
zy@zy-virtual-machine:~/workspace/nfs\$ ls	
rootfs	
zy@zy-virtual-machine:~/workspace/nfs\$ tar -xzf /home/zy/petalinux/Navigator	701
0 v3/ALIENTEK-ZYNQ-driver/images/linux/rootfs.tar.gz -C rootfs/ 🔶	
zy@zy-virtual-machine:~/workspace/nfs\$ sync	
zy@zy-virtual-machine:~/workspace/nfs\$ ls rootfs/	
bin boot dev etc home lib media mnt proc run sbin sys tmp usr	var

图 20.6.8 将根文件解压到 NFS 共享目录下

将 20.6.1 小节做好的启动卡插入开发板,启动模式设置为 sd 卡启动,连接串口,用网线 连接开发板 ps 网口和电脑,最后开发板连接电源启动,进入 uboot 模式。

输入如下命令设置环境变量 bootargs, 让内核通过 NFS 方式从 Ubuntu 主机中挂载根文件 系统:

setenv bootargs 'console=ttyPS0,115200 root=/dev/nfs rw

nfsroot=192.168.1.11:/home/zy/workspace/nfs/rootfs,nfsvers=3

ip=192.168.1.10:192.168.1.11:192.168.1.1:255.255.255.0::eth0:off

执行结果如下图所示:

U-Boot 2020.01 (Jun 01 2023 - 10:01:41 +0000)

```
CPU:
         Zynq 7z010
Silicon: v3.1
DRAM: ECC disabled 512 MiB
Flash: 0 Bytes
NAND: 0 MiB
         mmc@e0100000: 0, mmc@e0101000: 1
MMC:
 oading Environment from SPI Flash... SF: Detected w25q256 with page size 256 Bytes, erase size 4 KiB, to
tal 32 MiB
0K
In:
          serial@e0000000
          serial@e0000000
Out:
          serial@e0000000
Err:
Net:
ZYNQ GEM: e000b000, mdio bus e000b000, phyaddr -1, interface rgmii-id
eth0: ethernet@e000b000
ZYNQ GEM: e000c000, mdio bus e000c000, phyaddr -1, interface gmii
Could not get PHY for eth1: addr -1
Hit any key to stop autoboot: 0
Zynq> setenv bootargs 'console=ttyPS0,115200 root=/dev/nfs rw
> nfsroot=192.168.1.11:/home/zy/workspace/nfs/rootfs,nfsvers=3
> ip=192.168.1.10:192.168.1.11:192.168.1.1:255.255.255.0::eth0:off'
```

图 20.6.9

输入"boot"命令启动 linux 系统,启动过程如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php Hit any key to stop autoboot: 0 Zynq> setenv bootargs 'console=ttyPS0,115200 root=/dev/nfs rw > nfsroot=192.168.1.11:/home/zy/workspace/nfs/rootfs,nfsvers=3 ip=192.168.1.10:192.168.1.11:192.168.1.1:255.255.255.0::eth0:off' Zynq> Zynq> boot 🖣 switch to partitions #0, OK nmc0 is current device Scanning mmc 0:1... Found U-Boot script /boot.scr 2199 bytes read in 15 ms (142.6 KiB/s) ## Executing script at 03000000 4183488 bytes read in 246 ms (16.2 MiB/s) 17338 bytes read in 19 ms (890.6 KiB/s) 2083850 bytes read in 127 ms (15.6 MiB/s) design filename = "system_wrapper;UserID=0XFFFFFFFF;Version=2020.2" part number = "7z010clg400" date = "2023/03/22" time = "18:25:05" bytes in bitstream = 2083740 zynq_align_dma_buffer: Align buffer at 80006e to 7fff80(swap 1) INFO:post config was not run, please run manually if needed ## Flattened Device Tree blob at 00100000 Booting using the fdt blob at 0x100000 Loading Device Tree to leaff000, end leb063b9 ... OK Starting kernel ... Booting Linux on physical CPU 0x0 Linux version 5.4.0-xilinx (zy@zy-virtual-machine) (gcc version 9.2.0 (GCC)) #1 SMP PRE 58:51 CST 2023 CPU: ARMv7 Processor [413fc090] revision 0 (ARMv7), cr=18c5387d CPU: PIPT / VIPT nonaliasing data cache, VIPT aliasing instruction cache OF: fdt: Machine model: Alientek Navigator Zynq Development Board Memory policy: Data cache writealloc cma: Reserved 16 MiB at 0x1f000000 percpu: Embedded 15 pages/cpu s31820 r8192 d21428 u61440 Built 1 zonelists, mobility grouping on. Total pages: 130048 Kernel command line: console=ttyPS0,115200 root=/dev/nfs rw nfsroot=192.168.1.11:/home/zy/workspace/nfs/rootfs,nfsvers=3 ip=192.168.1.10:192.168.1.11:192.168.1.1:255.255.255.0::eth0:off Dentry cache hash table entries: 65536 (order: 6, 262144 bytes, linear) Inode-cache hash table entries: 32768 (order: 5, 131072 bytes, linear) mem auto-init: stack:off, heap alloc:off, heap free:off Memory: 493120K/524288K available (6144K kernel code, 195K rwdata, 1808K rodata, 1024K 784K reserved, 1634K come reserved, 0K bighmem) 784K reserved, 16384K cma-reserved, 0K highmem) rcu: Preemptible hierarchical RCU implementation. RCU restricting CPUs from NR_CPUS=4 to nr_cpu_ids=2. Tasks RCU enabled. rcu: rcu: RCU calculated value of scheduler-enlistment delay is 10 jiffies. rcu: Adjusting geometry for rcu_fanout_leaf=16, nr_cpu_ids=2 图 20.6.10 输入 boot 命令启动系统

系统启动完成后,进入登录界面,输入用户名和密码进入系统,如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php mmcblk1rpmb: mmc1:0001 8GTF4R partition 3 512 KiB, chardev (245:0) bootserver=192.168.1.11, rootserver=192.168.1.11, rootpath= mmcblk1: pl ALSA device list: No soundcards found. VFS: Mounted root (nfs filesystem) on device 0:13. devtmpfs: mounted Freeing unused kernel memory: 1024K Run /sbin/init as init process random: fast init done IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready INIT: version 2.88 booting Starting udev udevd[110]: starting version 3.2.8 random: udevd: uninitialized urandom read (16 bytes read) random: udevd: uninitialized urandom read (16 bytes read) random: udevd: uninitialized urandom read (16 bytes read) udevd[111]: starting eudev-3.2.8 FAT-fs (mmcblk0p1): Volume was not properly unmounted. Some data may be corrupt. Pleas FAT-fs (mmcblk0p1): Volume was not properly unmounted. Some data may be corre EXT4-fs (mmcblk0p2): recovery complete EXT4-fs (mmcblk0p2): mounted filesystem with ordered data mode. Opts: (null) EXT4-fs (mmcblk1p1): recovery complete EXT4-fs (mmcblk1p1): mounted filesystem with ordered data mode. Opts: (null) hwclock: can't open '/dev/misc/rtc': No such file or directory Wed Jun 7 03:03:44 UTC 2023 hwclock: can't open '/dev/misc/rtc': No such file or directory urandom read: 2 callbacks suppressed urandom_read: 2 callbacks suppressed random: dd: uninitialized urandom read (512 bytes read) INIT: Entering runlevel: 5 Configuring network interfaces... RTNETLINK answers: File exists ifup skipped for nfsroot interface eth0 run-parts: /etc/network/if-pre-up.d/nfsroot: exit status 1 Starting haveged: haveged: listening socket at 3 haveged: haveged starting up Starting Dropbear SSH server: random: dropbear: uninitialized urandom read (32 bytes n dropbear. hwclock: can't open '/dev/misc/rtc': No such file or directory Starting internet superserver: inetd. Starting syslogd/klogd: done Starting tcf-agent: OK PetaLinux 2020.2 ALIENTEK-ZYNQ-driver /dev/ttyPS0 ALIENTEK-ZYNQ-driver login: random: crng init done PetaLinux 2020.2 ALIENTEK-ZYNQ-driver /dev/ttyPS0 ALIENTEK-ZYNQ-driver login: root Password: root@ALIENTEK-ZYNQ-driver:~#

图 20.6.11 登录系统

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

正点原子

第二十一章 字符设备驱动开发

本章我们从 Linux 驱动开发中最基础的字符设备驱动开始,重点学习 Linux 下字符设备 驱动开发框架。本章会以一个虚拟的设备为例,讲解如何进行字符设备驱动开发,以及如何 编写测试 APP 来测试驱动工作是否正常,为以后的学习打下坚实的基础。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

21.1 字符设备驱动简介

字符设备是 Linux 驱动中最基本的一类设备驱动,字符设备就是一个一个字节,按照字 节流进行读写操作的设备,读写数据是分先后顺序的。比如我们最常见的 LED、按键、IIC、 SPI, LCD 等等都是字符设备,这些设备的驱动就叫做字符设备驱动。

在详细的学习字符设备驱动架构之前,我们先来简单的了解一下 Linux 下的应用程序是 如何调用驱动程序的, Linux 应用程序对驱动程序的调用如下图所示:



图 21.1.1 Linux 应用程序对驱动程序的调用流程

在 Linux 中一切皆为文件, 驱动加载成功以后会在"/dev"目录下生成一个相应的文件, 应用程序通过对这个名为"/dev/xxx"(xxx是具体的驱动文件名字)的文件进行相应的操作即 可实现对硬件的操作。比如现在有个叫做/dev/led 的驱动文件,此文件是 led 的驱动文件。应 用程序使用 open 函数来打开文件/dev/led,使用完成以后使用 close 函数关闭/dev/led 这个文 件。open 和 close 就是打开和关闭 led 驱动的函数,如果要点亮或关闭 led,那么就使用 write 函数来操作,也就是向此驱动写入数据,这个数据就是要关闭还是要打开 led 的控制参数。 如果要获取 led 灯的状态,就用 read 函数从驱动中读取相应的状态。

应用程序运行在用户空间,而 Linux 驱动属于内核的一部分,因此驱动运行于内核空间。 当我们在用户空间想要实现对内核的操作,比如使用 open 函数打开/dev/led 这个驱动,因为用 户空间不能直接对内核进行操作,因此必须使用一个叫做"系统调用"的方法来实现从用户 空间陷入到内核空间,这样才能实现对底层驱动的操作。open、close、write 和 read 等这些函 数是有 C 库提供的,在 Linux 系统中,系统调用作为 C 库的一部分。当我们调用 open 函数的 时候流程如下图所示:



图 21.1.2 open 函数调用流程

其中关于 C 库以及如何通过系统调用陷入到内核空间这个我们不用去管,我们重点关注的是应用程序和具体的驱动,应用程序使用到的函数在具体驱动程序中都有与之对应的函数,比如应用程序中调用了 open 这个函数,那么在驱动程序中也得有一个名为 open 的函数。每一个系统调用,在驱动中都有与之对应的一个驱动函数,在 Linux 内核文件 include/linux/fs.h 中有个叫做 file_operations 的结构体,此结构体就是 Linux 内核驱动操作函数集合,内容如下所示:

示例代码 file_operations 结构体

1692 struct file_operations {

1693 struct module *owner;

1694 loff_t (*llseek) (struct file *, loff_t, int);

1695 ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);

1696 ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);

1697 ssize_t (*read_iter) (struct kiocb *, struct iov_iter *);

1698 ssize_t (*write_iter) (struct kiocb *, struct iov_iter *);

1699 int (*iterate) (struct file *, struct dir_context *);

1700 int (*iterate_shared) (struct file *, struct dir_context *);

1701 unsigned int (*poll) (struct file *, struct poll_table_struct *);

1702 long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);

1703 long (*compat_ioctl) (struct file *, unsigned int, unsigned long);

1704 int (*mmap) (struct file *, struct vm_area_struct *);

1705 int (*open) (struct inode *, struct file *);

1706 int (*flush) (struct file *, fl_owner_t id);

1707 int (*release) (struct inode *, struct file *);

1708 int (*fsync) (struct file *, loff_t, loff_t, int datasync);

1709 int (*fasync) (int, struct file *, int);

1710 int (*lock) (struct file *, int, struct file_lock *);

1711 ssize_t (*sendpage) (struct file *, struct page *, int, size_t, loff_t *, int);

1712 unsigned long (*get_unmapped_area)(struct file *, unsigned long, unsigned long, unsigned long);

1713 int (*check_flags)(int);

1714 int (*flock) (struct file *, int, struct file_lock *);

1715 ssize_t (*splice_write)(struct pipe_inode_info *, struct file *, loff_t *, size_t, unsigned int);

1716 ssize_t (*splice_read)(struct file *, loff_t *, struct pipe_inode_info *, size_t, unsigned int);

1717 int (*setlease)(struct file *, long, struct file_lock **, void **);

1718 long (*fallocate)(struct file *file, int mode, loff_t offset,

1719 loff_t len);



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

1720 void (*show_fdinfo)(struct seq_file *m, struct file *f);
1721 #ifndef CONFIG MMU

1722 unsigned (*mmap_capabilities)(struct file *);

1723 #endif

1724 ssize_t (*copy_file_range)(struct file *, loff_t, struct file *,

1725 loff_t, size_t, unsigned int);

1726 int (*clone_file_range)(struct file *, loff_t, struct file *, loff_t,

1727 u64);

1728 ssize_t (*dedupe_file_range)(struct file *, u64, u64, struct file *,

1729 u64);

1730 } __randomize_layout;

简单介绍一下 file_operation 结构体中比较重要的、常用的函数:

第1693行,owner拥有该结构体的模块的指针,一般设置为THIS_MODULE。

第1694行, llseek函数用于修改文件当前的读写位置。

第1695行, read 函数用于读取设备文件。

第1696行, write 函数用于向设备文件写入(发送)数据。

第 1701 行, poll 是个轮询函数,用于查询设备是否可以进行非阻塞的读写。

第 1702 行, unlocked_ioctl 函数提供对于设备的控制功能, 与应用程序中的 ioctl 函数对应。

第1703行, compat_ioctl函数与 unlocked_ioctl函数功能一样,区别在于在64位系统上, 32 位的应用程序调用将会使用此函数。在32 位的系统上运行32 位的应用程序调用的是 unlocked_ioctl。

第1704行,mmap函数用于将设备的内存映射到进程空间中(也就是用户空间),一般帧缓冲设备会使用此函数,比如 LCD 驱动的显存,将帧缓冲(LCD 显存)映射到用户空间中以后应用程序就可以直接操作显存了,这样就不用在用户空间和内核空间之间来回复制。

第1705行, open 函数用于打开设备文件。

第1707行, release 函数用于释放(关闭)设备文件, 与应用程序中的 close 函数对应。

第1709行,fsync函数用于刷新待处理的数据,用于将缓冲区中的数据刷新到磁盘中。

第1605行, fasync 函数与 fsync 函数的功能类似,只是 fasync 是异步刷新待处理的数据。

在字符设备驱动开发中最常用的就是上面这些函数,关于其他的函数大家可以查阅相关 文档。我们在字符设备驱动开发中最主要的工作就是实现上面这些函数,不一定全部都要实 现,但是像 open、release、write、read 等都是需要实现的,当然了,具体需要实现哪些函数 还是要看具体的驱动要求。

21.2 字符设备驱动开发步骤

上一小节我们简单的介绍了一下字符设备驱动,那么字符设备驱动开发都有哪些步骤 呢?我们在学习裸机或者 STM32的时候关于驱动的开发就是初始化相应的外设寄存器,在 Linux 驱动开发中肯定也是要初始化相应的外设寄存器,这个是毫无疑问的。只是在 Linux 驱 动开发中我们需要按照其规定的框架来编写驱动,所以说学 Linux 驱动开发重点是学习其驱 动框架。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

21.2.1 驱动模块的加载和卸载

Linux 驱动有两种运行方式, 第一种就是将驱动编译进 Linux 内核中, 这样当 Linux 内核 启动的时候就会自动运行驱动程序。第二种就是将驱动编译成模块(Linux 下模块扩展名为.ko), 在Linux内核启动以后使用"insmod"命令加载驱动模块。在调试驱动的时候一般都选择将其 编译为模块,这样我们修改驱动以后只需要编译一下驱动代码即可,不需要编译整个 Linux 代码。而且在调试的时候只需要加载或者卸载驱动模块即可,不需要重启整个系统。总之, 将驱动编译为模块最大的好处就是方便开发,当驱动开发完成,确定没有问题以后就可以将 驱动编译进 Linux 内核中,当然也可以不编译进 Linux 内核中,具体看自己的需求。

模块有加载和卸载两种操作,我们在编写驱动的时候需要注册这两种操作函数,模块的 加载和卸载注册函数如下:

module init(xxx init); //注册模块加载函数 //注册模块卸载函数 module exit(xxx exit);

module init 函数用来向 Linux 内核注册一个模块加载函数,参数 xxx init 就是需要注册 的具体函数,当使用"insmod"命令加载驱动的时候, xxx init 这个函数就会被调用。 module_exit()函数用来向 Linux 内核注册一个模块卸载函数,参数 xxx_exit 就是需要注册的具 体函数,当使用"rmmod"命令卸载具体驱动的时候 xxx exit 函数就会被调用。字符设备驱 动模块加载和卸载模板如下所示:

示例代码 字符设备驱动模块加载和卸载函数模板

```
1 /* 驱动入口函数 */
2 static int __init xxx_init(void)
3 {
4 /* 入口函数具体内容 */
5
   return 0;
6 }
7
8 /* 驱动出口函数 */
9 static void ___exit xxx_exit(void)
10 {
11 /* 出口函数具体内容 */
12 }
13
14 /* 将上面两个函数指定为驱动的入口和出口函数 */
15 module init(xxx init);
16 module_exit(xxx_exit);
第2行,定义了个名为 xxx_init 的驱动入口函数,并且使用了"__init"来修饰。
第9行,定义了个名为 xxx_exit 的驱动出口函数,并且使用了"__exit"来修饰。
```

第 15 行,调用函数 module init 来声明 xxx init 为驱动入口函数,当加载驱动的时候 xxx init 函数就会被调用。

第16行,调用函数 module_exit 来声明 xxx_exit 为驱动出口函数,当卸载驱动的时候 xxx_exit 函数就会被调用。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

驱动编译完成以后扩展名为.ko,有两种命令可以加载驱动模块: insmod 和 modprobe, insmod 是最简单的模块加载命令,此命令用于加载指定的.ko 模块,比如加载 drv.ko 这个驱动模块,命令如下:

insmod drv.ko

insmod 命令不能解决模块的依赖关系,比如 drv.ko 依赖 first.ko 这个模块,就必须先使用 insmod 命令加载 first.ko 这个模块,然后再加载 drv.ko 这个模块,但是 modprobe 就不会存在 这个问题。modprobe 会分析模块的依赖关系,然后将所有依赖的模块都加载到内核中,因此 modprobe 命令相比 insmod 要智能一些。modprobe 命令主要智能在提供了模块的依赖性分 析、错误检查、错误报告等功能,推荐使用 modprobe 命令来加载驱动。modprobe 命令默认 会去/lib/modules/<kernel-version>目录中查找模块,比如本书使用的 Linux kernel 的版本号为 5.4.0, modprobe 命令默认会到/lib/modules/ 5.4.0-150-generic 这个目录中查找相应的驱动模 块,一般自己制作的根文件系统中是不会有这个目录的,所以需要自己手动创建。

驱动模块的卸载使用命令"rmmod"即可,比如要卸载 drv.ko,使用如下命令即可: rmmod drv.ko

也可以使用"modprobe-r"命令卸载驱动,比如要卸载 drv.ko,命令如下:

modprobe -r drv.ko

使用 modprobe 命令可以卸载掉驱动模块所依赖的其他模块,前提是这些依赖模块已经没有被 其他模块所使用,否则就不能使用 modprobe 来卸载驱动模块。所以对于模块的卸载,还是推 荐使用 rmmod 命令。

21.2.2 字符设备注册与注销

对于字符设备驱动而言,当驱动模块加载成功以后需要注册字符设备,同样,卸载驱动 模块的时候也需要注销掉字符设备。字符设备的注册和注销函数原型如下所示:

static inline int register_chrdev(unsigned int major, const char *name, const struct file_operations *fops) static inline void unregister_chrdev(unsigned int major, const char *name)

register_chrdev 函数用于注册字符设备,此函数一共有三个参数,这三个参数的含义如下: major: 主设备号,Linux 下每个设备都有一个设备号,设备号分为主设备号和次设备号 两部分,关于设备号后面会详细讲解。

name: 设备名字, 指向一串字符串。

fops: 结构体 file_operations 类型指针,指向设备的操作函数集合变量。

unregister_chrdev函数用于注销字符设备,此函数有两个参数,这两个参数含义如下:

major:要注销的设备对应的主设备号。

name: 要注销的设备对应的设备名。

一般字符设备的注册在驱动模块的入口函数 xxx_init 中进行,字符设备的注销在驱动模块的 出口函数 xxx_exit 中进行。字符设备的注册和注销示例代码内容如下所示:

示例代码 字符设备注册和注销

```
1 static struct file_operations test_fops;
```

```
2
```

```
3 /* 驱动入口函数 */
```

```
4 static int __init xxx_init(void)
```

```
5 {
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

```
/* 入口函数具体内容 */
6
7
   int retvalue = 0:
8
9
   /* 注册字符设备驱动 */
10 retvalue = register_chrdev(200, "chrtest", &test_fops);
   if(retvalue < 0){
11
       /* 字符设备注册失败,自行处理 */
12
13 }
14 return 0;
15 }
16
17 /* 驱动出口函数 */
18 static void __exit xxx_exit(void)
19 {
20 /* 注销字符设备驱动 */
21 unregister_chrdev(200, "chrtest");
22 }
23
24 /* 将上面两个函数指定为驱动的入口和出口函数 */
25 module_init(xxx_init);
26 module exit(xxx exit);
```

第1行,定义了一个 file_operations 结构体变量 test_fops, test_fops 就是设备的操作函数 集合,只是此时我们还没有初始化 test_fops 中的 open、release 等这些成员变量,所以这个操 作函数集合还是空的。

第 10 行,调用函数 register_chrdev 注册字符设备,主设备号为 200,设备名为 "chrtest",设备操作函数集合就是第 1 行定义的 test_fops。要注意的一点就是,选择没有被 使用的主设备号,输入命令"cat /proc/devices"可以查看当前已经被使用掉的设备号,如下 图所示(限于篇幅原因,只展示一部分):

```
/ # cat /proc/devices
Character devices:
  1 mem
    /dev/vc/0
  4
  4
    tty
   /dev/tty
  5
 5
   /dev/console
  5
   /dev/ptmx
  7 vcs
10 misc
 13 input
 29 fb
81 video4linux
  图 21.2.1 查看设备号
```

上图列出了当前系统中所有的字符设备和块设备,其中第1列就是设备对应的主设备 号。200这个主设备号在我的开发板中并没有被使用,所以我这里就用了200这个主设备 号。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php 第 21 行,调用函数 unregister_chrdev 注销主设备号为 200 的这个设备。

21.2.3 实现设备的具体操作函数

file_operations 结构体就是设备的具体操作函数,在字符设备注册和注销的示例代码中我 们定义了 file_operations 结构体类型的变量 test_fops,但是还没对其进行初始化,也就是初始 化其中的 open、release、read 和 write 等具体的设备操作函数。本小节我们就完成变量 test_fops 的初始化,设置好针对 chrtest 设备的操作函数。在初始化 test_fops 之前我们要分析 一下需求,也就是要对 chrtest 这个设备进行哪些操作,只有确定了需求以后才知道我们应该 实现哪些操作函数。假设对 chrtest 这个设备有如下两个要求:

1、能够对 chrtest 进行打开和关闭操作

设备打开和关闭是最基本的要求,几乎所有的设备都得提供打开和关闭的功能。因此我 们需要实现 file_operations 中的 open 和 release 这两个函数。

2、对 chrtest 进行读写操作

假设 chrtest 这个设备控制着一段缓冲区(内存),应用程序需要通过 read 和 write 这两个函数对 chrtest 的缓冲区进行读写操作,所以需要实现 file_operations 中的 read 和 write 这两个函数。

需求很清晰了,修改字符设备注册和注销的示例代码,在其中加入 test_fops 这个结构体 变量的初始化操作,完成以后的内容如下所示:

```
示例代码 加入设备操作函数
```

```
1 /* 打开设备 */
2 static int chrtest_open(struct inode *inode, struct file *filp)
3 {
4 /* 用户实现具体功能 */
    return 0;
5
6 }
7
8 /* 从设备读取 */
9 static ssize_t chrtest_read(struct file *filp, char __user *buf, __size_t cnt, loff_t *offt)
10 {
11 /* 用户实现具体功能 */
12 return 0;
13 }
14
15 /* 向设备写数据 */
16 static ssize t chrtest write(struct file *filp, const char user *buf, size t cnt, loff t *offt)
17 {
18 /* 用户实现具体功能 */
19 return 0;
20 }
21
```



```
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
   22 /* 关闭/释放设备 */
   23 static int chrtest_release(struct inode *inode, struct file *filp)
   24 {
   25 /* 用户实现具体功能 */
   26 return 0;
   27 }
   28
   29 static struct file_operations test_fops = {
   30 .owner = THIS_MODULE,
   31 .open = chrtest_open,
   32 .read = chrtest_read,
    33 .write = chrtest_write,
   34 .release = chrtest_release,
   35 };
   36
   37 /* 驱动入口函数 */
   38 static int __init xxx_init(void)
   39 {
   40 /* 入口函数具体内容 */
   41 int retvalue = 0;
   42
   43 /* 注册字符设备驱动 */
   44 retvalue = register_chrdev(200, "chrtest", &test_fops);
   45 if (retvalue < 0)
   46
           /* 字符设备注册失败,自行处理 */
   47 }
   48 return 0;
   49 }
   50
   51 /* 驱动出口函数 */
   52 static void __exit xxx_exit(void)
   53 {
   54 /* 注销字符设备驱动 */
   55 unregister_chrdev(200, "chrtest");
   56 }
   57
   58 /* 将上面两个函数指定为驱动的入口和出口函数 */
   59 module_init(xxx_init);
   60 module_exit(xxx_exit);
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

在上面的示例代码中我们一开始编写了四个函数: chrtest_open、chrtest_read、 chrtest_write 和 chrtest_release。这四个函数就是 chrtest 设备的 open、read、write 和 release 操 作函数。第 29 行~35 行初始化 test_fops 的 open、read、write 和 release 这四个成员变量。

21.2.4 添加 LICENSE 和作者信息

最后我们需要在驱动中加入 LICENSE 信息和作者信息,其中 LICENSE 是必须添加的, 否则的话编译的时候会报错,作者信息可以添加也可以不添加。LICENSE 和作者信息的添加 使用如下两个函数:

```
MODULE_LICENSE() //添加模块 LICENSE 信息
MODULE_AUTHOR() //添加模块作者信息
最后给加入设备操作函数的示例代码加入 LICENSE 和作者信息,完成以后的内容如下:
示例代码 字符设备驱动最终的模板
1 /* 打开设备 */
2 static int chrtest_open(struct inode *inode, struct file *filp)
```

3 {

```
4 /* 用户实现具体功能 */
```

```
5 return 0;
```

6 }

```
•••••
```

```
57
```

58 /* 将上面两个函数指定为驱动的入口和出口函数 */

59 module_init(xxx_init);

60 module_exit(xxx_exit);

61

62 MODULE_LICENSE("GPL");

63 MODULE_AUTHOR("alientek");

第62行,LICENSE采用GPL协议。

第63行,添加作者名字。

至此,字符设备驱动开发的完整步骤就讲解完了,而且也编写好了一个完整的字符设备 驱动模板,以后字符设备驱动开发都可以在此模板上进行。

21.3 Linux 设备号

21.3.1 设备号的组成

为了方便管理,Linux 中每个设备都有一个设备号,设备号由主设备号和次设备号两部分 组成,主设备号表示某一个具体的驱动,次设备号表示使用这个驱动的各个设备。Linux 提供 了一个名为 dev_t 的数据类型表示设备号,dev_t 定义在文件 include/linux/types.h 里面,定义 如下:

示例代码 设备号 dev_t

```
12 typedef __u32 __kernel_dev_t;
```

正点原子

原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php

15 typedef __kernel_dev_t dev_t;

可以看出 dev_t 是__u32 类型的, 而__u32 定义在文件 include/uapi/asm-generic/int-ll64.h 里 面, 定义如下:

示例代码 u32 类型

26 **typedef** unsigned int ___u32;

综上所述, dev t 其实就是 unsigned int 类型, 是一个 32 位的数据类型。这 32 位的数据构 成了主设备号和次设备号两部分,其中高 12 位为主设备号,低 20 位为次设备号。因此 Linux 系统中主设备号范围为 0~4095, 所以大家在选择主设备号的时候一定不要超过这个范围。在 文件 include/linux/kdev_t.h 中提供了几个关于设备号的操作函数(本质是宏),如下所示:

示例代码 设备号操作函数

7 #define MINORBITS 20

8 #define MINORMASK ((1U << MINORBITS) - 1)

9

10 #define MAJOR(dev) ((unsigned int) ((dev) >> MINORBITS))

11 #define MINOR(dev) ((unsigned int) ((dev) & MINORMASK))

12 #define MKDEV(ma,mi) (((ma) << MINORBITS) | (mi))

第7行, 宏 MINORBITS 表示次设备号位数, 一共是 20 位。

第8行, 宏 MINORMASK 表示次设备号掩码。

第10行, 宏 MAJOR 用于从 dev_t 中获取主设备号, 将 dev_t 右移 20 位即可。

第 11 行, 宏 MINOR 用于从 dev t 中获取次设备号, 取 dev t 的低 20 位的值即可。

第12行, 宏 MKDEV 用于将给定的主设备号和次设备号的值组合成 dev_t 类型的设备 号。

21.3.2 设备号的分配

1、静态分配设备号

本小节讲的设备号分配主要是主设备号的分配。前面讲解字符设备驱动的时候说过了, 注册字符设备的时候需要给设备指定一个设备号,这个设备号可以是驱动开发者静态的指定 一个设备号,比如选择 200 这个主设备号。有一些常用的设备号已经被 Linux 内核开发者给分 配掉了,具体分配的内容可以查看文档 Documentation/devices.txt。并不是说内核开发者已经 分配掉的主设备号我们就不能用了,具体能不能用还得看我们的硬件平台运行过程中有没有 使用这个主设备号,使用"cat /proc/devices"命令即可查看当前系统中所有已经使用了的设 备号。

2、动态分配设备号

静态分配设备号需要我们检查当前系统中所有被使用了的设备号,然后挑选一个没有使 用的。而且静态分配设备号很容易带来冲突问题, Linux 社区推荐使用动态分配设备号, 在注 册字符设备之前先申请一个设备号,系统会自动给你一个没有被使用的设备号,这样就避免 了冲突。卸载驱动的时候释放掉这个设备号即可,设备号的申请函数如下:

int alloc_chrdev_region(dev_t *dev, unsigned baseminor, unsigned count, const char *name)

函数 alloc chrdev region 用于申请设备号,此函数有4个参数:

dev: 保存申请到的设备号。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

baseminor:次设备号起始地址,alloc_chrdev_region 可以申请一段连续的多个设备号,这些设备号的主设备号一样,但是次设备号不同,次设备号以 baseminor 为起始地址并逐次递增。一般 baseminor 为 0,也就是说次设备号从 0 开始。

count:要申请的设备号数量。
name:设备名字。
注销字符设备之后要释放掉设备号,设备号释放函数如下:
void unregister_chrdev_region(dev_t from, unsigned count)
此函数有两个参数:
from:要释放的设备号。
count:表示从 from 开始,要释放的设备号数量。

21.4 chrdevbase 字符设备驱动开发实验

字符设备驱动开发的基本步骤我们已经了解了,本节我们就以 chrdevbase 这个虚拟设备为例,完整的编写一个字符设备驱动模块。chrdevbase不是实际存在的一个设备,是笔者为了方便讲解字符设备的开发而引入的一个虚拟设备。chrdevbase设备有两个缓冲区,一个为读缓冲区,一个为写缓冲区,这两个缓冲区的大小都为100字节。在应用程序中可以向 chrdevbase 设备的写缓冲区中写入数据,从读缓冲区中读取数据。chrdevbase 这个虚拟设备的功能很简单,但是它包含了字符设备的最基本功能。

21.4.1 编写驱动程序

本实验对应的例程路径为: ZYNQ 开发板光盘资料(A 盘) \4_SourceCode\3_Embedded_Linux\Linux 驱动例程\1_chrdevbase。

应用程序调用 open 函数打开 chrdevbase 这个设备,打开以后可以使用 write 函数向 chrdevbase 的写缓冲区 writebuf 中写入数据(不超过 100 个字节),也可以使用 read 函数读取读 缓冲区 readbuf 中的数据操作,操作完成以后应用程序使用 close 函数关闭 chrdevbase 设备。

1、创建工程

在 Ubuntu 中创建一个目录用来存放 Linux 驱动程序,比如我在/home/zy/linux 目录下创建 了一个名为 drivers 的目录来存放所有的 Linux 驱动。在 drivers 目录下新建一个名为 1_chrdevbase 的子目录来存放本实验所有文件,如下图所示:

zy@zy-virtual-machine:~\$ mkdir -p linux/drivers/1_chrdevbase
zy@zy-virtual-machine:~\$ cd linux/drivers/1_chrdevbase
zy@zy-virtual-machine:~/linux/drivers/1_chrdevbase\$ ls
zy@zy-virtual-machine:~/linux/drivers/1_chrdevbase\$

图 21.4.1 创建 linux 驱动目录

2、编写实验程序

工程建立好以后就可以开始编写驱动程序了,在 1_chrdevbase 目录下新建的 chrdevbase.c 源文件,打开该文件里面输入如下内容,关于在 Ubuntu 下编写代码,笔者一直喜欢用 vim 和 Windows 下的 Notepad++软件,当然每个人都有自己喜欢的开发方式,看个人了!

示例代码 chrdevbase.c 文件

1 #include <linux/types.h>



```
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
   2 #include <linux/kernel.h>
   3 #include <linux/delay.h>
   4 #include <linux/ide.h>
   5 #include <linux/init.h>
   6 #include <linux/module.h>
   8 Copyright © ALIENTEK Co., Ltd. 1998-2029. All rights reserved.
   9 文件名 : chrdevbase.c
   10 作者 : 正点原子
   11 版本 : V1.0
   12 描述 : chrdevbase 驱动文件。
   13 其他 :无
   14 论坛 :www.openedv.com
   15 日志 : 初版 V1.0 2019/1/30 左忠凯创建
   17
   18 #define CHRDEVBASE_MAJOR 200
                                         // 主设备号
   19 #define CHRDEVBASE_NAME "chrdevbase"
                                           // 设备名
   20
   21 static char readbuf[100];
                                // 读缓冲区
   22 static char writebuf[100];
                                // 写缓冲区
   23 static char kerneldata[] = {"kernel data!"};
   24
   25 /*
   26 * @description : 打开设备
   27 * @param - inode : 传递给驱动的 inode
   28 * @param - filp : 设备文件, file 结构体有个叫做 private_data 的成员变量
   29 *
               一般在 open 的时候将 private_data 指向设备结构体。
               :0 成功;其他 失败
   30 * @return
   31 */
   32 static int chrdevbase_open(struct inode *inode, struct file *filp)
   33 {
   34
       printk("chrdevbase open!\r\n");
       return 0;
   35
   36 }
   37
   38 /*
   39 * @description :从设备读取数据
   40 * @param - filp : 要打开的设备文件(文件描述符)
   41 * @param - buf :返回给用户空间的数据缓冲区
   42 * @param - cnt : 要读取的数据长度
```



```
原子哥在线教学: www.yuanzige.com
                                           论坛:www.openedv.com/forum.php
   43 * @param - offt : 相对于文件首地址的偏移
   44 * @return
                    :读取的字节数,如果为负值,表示读取失败
   45 */
   46 static ssize_t chrdevbase_read(struct file *filp, char __user *buf, size_t cnt, loff_t *offt)
   47 {
   48
        int retvalue = 0;
   49
   50
        /* 向用户空间发送数据 */
   51
         memcpy(readbuf, kerneldata, sizeof(kerneldata));
         retvalue = copy_to_user(buf, readbuf, cnt);
   52
    53
        if(retvalue == 0){
    54
           printk("kernel senddata ok!\r\n");
   55
         }else{
    56
          printk("kernel senddata failed!\r\n");
   57
        }
   58
   59
        //printk("chrdevbase read!\r\n");
   60
         return 0;
   61 }
   62
   63 /*
   64 * @description : 向设备写数据
   65 * @param - filp : 设备文件,表示打开的文件描述符
   66 * @param - buf : 要写给设备写入的数据
   67 * @param - cnt : 要写入的数据长度
   68 * @param - offt : 相对于文件首地址的偏移
                   :写入的字节数,如果为负值,表示写入失败
   69 * @return
   70 */
   71 static ssize_t chrdevbase_write(struct file *filp, const char __user *buf, size_t cnt, loff_t *offt)
   72 {
   73
        int retvalue = 0;
        /* 接收用户空间传递给内核的数据并且打印出来 */
   74
   75
         retvalue = copy_from_user(writebuf, buf, cnt);
        if(retvalue == 0){
   76
          printk("kernel recevdata:%s\r\n", writebuf);
   77
   78
        }else{
   79
           printk("kernel recevdata failed!\r\n");
   80
        }
   81
   82
        //printk("chrdevbase write!\r\n");
   83
        return 0;
```



```
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
   84 }
   85
    86 /*
   87 * @description :关闭/释放设备
   88 * @param - filp : 要关闭的设备文件(文件描述符)
   89 * @return : 0 成功;其他 失败
   90 */
   91 static int chrdevbase_release(struct inode *inode, struct file *filp)
   92 {
        //printk("chrdevbase release! \r\n");
   93
   94
         return 0;
   95 }
   96
   97 /*
   98 * 设备操作函数结构体
   99 */
    100 static struct file_operations chrdevbase_fops = {
    101 .owner = THIS_MODULE,
    102 .open = chrdevbase_open,
    103 .read = chrdevbase_read,
    104 .write = chrdevbase_write,
    105 .release = chrdevbase_release,
    106 };
    107
    108 /*
    109 * @description : 驱动入口函数
    110 * @param
                    :无
    111 * @return : 0 成功;其他 失败
    112 */
    113 static int __init chrdevbase_init(void)
    114 {
    115 int retvalue = 0;
    116
    117 /* 注册字符设备驱动 */
    118 retvalue = register_chrdev(CHRDEVBASE_MAJOR, CHRDEVBASE_NAME, & chrdevbase_fops);
    119 if(retvalue < 0){
    120
           printk("chrdevbase driver register failed\r\n");
    121
         }
    122
         printk("chrdevbase_init()\r\n");
    123
         return 0;
    124
```


原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

125
126 /*
127 * @description : 驱动出口函数
128 * @param :无
129 * @return :无
130 */
131 static voidexit chrdevbase_exit(void)
132 {
133 /* 注销字符设备驱动 */
<pre>134 unregister_chrdev(CHRDEVBASE_MAJOR, CHRDEVBASE_NAME);</pre>
<pre>135 printk("chrdevbase_exit()\r\n");</pre>
136 }
137
138 /*
139 * 将上面两个函数指定为驱动的入口和出口函数
140 */
141 module_init(chrdevbase_init);
142 module_exit(chrdevbase_exit);
143
144 /*
145 * LICENSE 和作者信息
146 */
147 MODULE_LICENSE("GPL");
148 MODULE_AUTHOR("alientek");
第 32~36 行, chrdevbase_open 函数, 当应用程序调用 open 函数的时候此函数就会调用,
本例程中我们没有做任何工作,只是输出一串字符,用于调试。这里使用了 printk 来输出信
息,而不是 printf。因为在 Linux 内核中没有 printf 这个函数。printk 相当于 printf 的孪生兄妹,

息,而不是printf。因为在Linux内核中没有printf这个函数。printk相当于printf的孪生兄妹, printf运行在用户态,printk运行在内核态。在内核中想要向控制台输出或显示一些内容,必须使用printk这个函数。不同之处在于,printk可以根据日志级别对消息进行分类,一共有8 个消息级别,这8个消息级别定义在文件 include/linux/kern_levels.h 里面,定义如下:

#define KERN_SOH "\001" #define KERN_SOH_ASCII '\001' /* ASCII Start Of Header */

#define KERN_EMERG KERN_SOH "0"	/* system is unusable */
#define KERN_ALERT KERN_SOH "1"	/* action must be taken immediately */
#define KERN_CRIT KERN_SOH "2"	/* critical conditions */
#define KERN_ERR KERN_SOH "3"	/* error conditions */
#define KERN_WARNING KERN_SOH "4"	/* warning conditions */
#define KERN_NOTICE KERN_SOH "5"	/* normal but significant condition */
#define KERN_INFO KERN_SOH "6"	/* informational */
#define KERN DEBUG KERN SOH "7"	/* debug-level messages */



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

一共定义了8个级别,其中0的优先级最高,7的优先级最低。如果要设置消息级别,参考如下示例:

printk(KERN_EMERG "gsmi: Log Shutdown Reason\n");

上述代码就是设置"gsmi: Log Shutdown Reason\n"这行消息的级别为 KERN_EMERG。 在具体的消息前面加上 KERN_EMERG 就可以将这条消息的级别设置为 KERN_EMERG。如 果使用 printk 的时候不显式的设置消息级别,那么 printk 将会采用默认级别 CONFIG_MESSAGE_LOGLEVEL_DEFAULT, CONFIG_MESSAGE_LOGLEVEL_DEFAULT 默认为4。

在 include/linux/printk.h 中有个宏 CONSOLE_LOGLEVEL_DEFAULT,定义如下: /*

* Default used to be hard-coded at 7, we're now allowing it to be set from

* kernel config.

*/

#define CONSOLE_LOGLEVEL_DEFAULT CONFIG_CONSOLE_LOGLEVEL_DEFAULT

CONSOLE_LOGLEVEL_DEFAULT 控制着哪些级别的消息可以显示在控制台上,从注释可知此宏默认为7,意味着只有优先级高于7的消息才能显示在控制台上。

这个就是 printk 和 printf 的最大区别,可以通过消息级别来决定哪些消息可以显示在控制 台上。默认消息级别为 4,4 的级别比 7 高,所示直接使用 printk 输出的信息是可以显示在控 制台上的。

参数 filp 有个叫做 private_data 的成员变量, private_data 是个 void 指针, 一般在驱动中将 private_data 指向设备结构体, 设备结构体会存放设备的一些属性。

第46~61 行, chrdevbase_read 函数,应用程序调用 read 函数从设备中读取数据的时候此 函数会执行。参数 buf 是用户空间的内存,读取到的数据存储在 buf 中,参数 cnt 是要读取的 字节数,参数 offt 是相对于文件首地址的偏移。kerneldata 里面保存着用户空间要读取的数据, 第52 行先将 kerneldata 数组中的数据拷贝到读缓冲区 readbuf 中,第53 行通过函数 copy_to_user 函数将 readbuf 中的数据复制到参数 buf 中。因为内核空间不能直接操作用户空间 的内存,因此需要借助 copy_to_user 函数来完成内核空间的数据到用户空间的复制。 copy_to_user 函数原型如下:

static inline long copy_to_user(void __user *to, const void *from, unsigned long n)

参数 to 表示目的,参数 from 表示源,参数 n 表示要复制的数据长度。如果复制成功,返 回值为 0,如果复制失败则返回负数。

第71~84行, chrdevbase_write 函数,应用程序调用 write 函数向设备写数据的时候此函数 就会执行。参数 buf 就是应用程序要写入设备的数据,也是用户空间的内存,参数 cnt 是要写 入的数据长度,参数 offt 是相对文件首地址的偏移。第75 行通过函数 copy_from_user 将 buf 中的数据复制到写缓冲区 writebuf 中,因为用户空间内存不能直接访问内核空间的内存,所 以需要借助函数 copy_from_user 将用户空间的数据复制到 writebuf 这个内核空间中。

第 91~95 行, chrdevbase_release 函数,应用程序调用 close 关闭设备文件的时候此函数会执行,一般会在此函数里面执行一些释放操作。如果在 open 函数中设置了 filp 的 private_data 成员变量指向设备结构体,那么在 release 函数最终就要释放掉。

第 100~106 行,新建 chrdevbase 的设备文件操作结构体 chrdevbase_fops,初始化 chrdevbase_fops。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

第 113~124 行, 驱动入口函数 chrdevbase_init, 第 118 行调用函数 register_chrdev 来注册 字符设备。

第 131~136 行, 驱动出口函数 chrdevbase_exit, 第 134 行调用函数 unregister_chrdev 来注 销字符设备。

第 141~142 行,通过 module_init 和 module_exit 这两个函数来指定驱动的入口和出口函数。

第 147~148 行,添加 LICENSE 和作者信息。

21.4.2 编写测试 APP

1、C 库文件操作基本函数

编写测试 APP 就是编写 Linux 应用,需要用到 C 库里面和文件操作有关的一些函数,比 如 open、read、write 和 close 这四个函数。

①、open 函数

open 函数原型如下:

int open(const char *pathname, int flags)

open 函数参数含义如下:

pathname: 要打开的设备或者文件名。

flags: 文件打开模式,以下三种模式必选其一:

O_RDONLY	只读模式
—	

O_WRONLY 只写模式

O_RDWR 读写模式

因为我们要对 chrdevbase 这个设备进行读写操作,所以选择 O_RDWR。除了上述三种模式以外还有其他的可选模式,通过逻辑或来选择多种模式:

O_APPEND 每次写操作都写入文件的末尾

O_CREAT 如果指定文件不存在,则创建这个文件

O_EXCL 如果要创建的文件已存在,则返回 -1,并且修改 errno 的值

O_TRUNC 如果文件存在,并且以只写/读写方式打开,则清空文件全部内容

O_NOCTTY 如果路径名指向终端设备,不要把这个设备用作控制终端。

O_NONBLOCK 如果路径名指向 FIFO/块文件/字符文件,则把文件的打开和后继 I/O 设置为非阻塞

DSYNC 等待物理 I/O 结束后再 write。在不影响读取新写入的数据的前提下,不等待文件属性更新。

O_RSYNC read 等待所有写入同一区域的写操作完成后再进行。

O_SYNC 等待物理 I/O 结束后再 write,包括更新文件属性的 I/O。 **返回值**:如果文件打开成功的话返回文件的文件描述符。

在 Ubuntu 中输入"man 2 open"即可查看 open 函数的详细内容,如下图所示:

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

8 🗖 🗊	zuozhongkai@ubuntu: ~
OPEN(2) Linux Programmer's Manual OPEN(2)
NAME	open, openat, creat - open and possibly create a file
SYNOPS	IS #include <sys types.h=""> #include <sys stat.h=""> #include <fcntl.h></fcntl.h></sys></sys>
	<pre>int open(const char *pathname, int flags); int open(const char *pathname, int flags, mode_t mode);</pre>
	<pre>int creat(const char *pathname, mode_t mode);</pre>
	int openat(int <u>dirfd</u> , const char * <u>pathname</u> , int <u>flags</u>); int openat(int <u>dirfd</u> , const char * <u>pathname</u> , int <u>flags</u> , mode_t <u>mode</u>);
Fea	ture Test Macro Requirements for glibc (see feature_test_macros (7)):
	openat(): Since glibc 2.10:
Manua	l page open(2) line 1 (press h for help or q to quit)

正点原子

图 21.4.2 open 函数帮助信息

②、read 函数

read 函数原型如下:

ssize_t read(int fd, void *buf, size_t count)

read 函数参数含义如下:

fd:要读取的文件描述符,读取文件之前要先用 open 函数打开文件, open 函数打开文件 成功以后会得到文件描述符。

buf: 数据读取到此 buf 中。

count: 要读取的数据长度,也就是字节数。

返回值: 读取成功的话返回读取到的字节数; 如果返回 0 表示读取到了文件末尾; 如果 返回负值, 表示读取失败。在 Ubuntu 中输入 "man 2 read" 命令即可查看 read 函数的详细内 容。

③、write 函数

write 函数原型如下:

ssize_t write(int fd, const void *buf, size_t count);

write 函数参数含义如下:

fd:要进行写操作的文件描述符,写文件之前要先用 open 函数打开文件, open 函数打开 文件成功以后会得到文件描述符。

buf: 要写入的数据。

count: 要写入的数据长度,也就是字节数。

返回值: 写入成功的话返回写入的字节数;如果返回 0 表示没有写入任何数据;如果返回负值,表示写入失败。在 Ubuntu 中输入 "man 2 write" 命令即可查看 write 函数的详细内容。

④、close 函数

close 函数原型如下:

int close(int fd);

close 函数参数含义如下:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

fd: 要关闭的文件描述符。

返回值: 0 表示关闭成功,负值表示关闭失败。在 Ubuntu 中输入"man 2 close"命令即 可查看 close 函数的详细内容。

2、编写测试 APP 程序

驱动编写好以后是需要测试的,一般编写一个简单的测试 APP,测试 APP 运行在用户空 间。测试 APP 很简单通过输入相应的指令来对 chrdevbase 设备执行读或者写操作。在 1_chrdevbase 目录中新建 chrdevbaseApp.c 文件,在此文件中输入如下内容:

示例代码 chrdevbaseApp.c 文件

1 #include "stdio.h" 2 #include "unistd.h" 3 #include "sys/types.h" 4 #include "sys/stat.h" 5 #include "fcntl.h" 6 #include "stdlib.h" 7 #include "string.h" 9 Copyright © ALIENTEK Co., Ltd. 1998-2029. All rights reserved. 10 文件名 : chrdevbaseApp.c 11 作者 : 正点原子 12版本 : V1.0 13 描述 : chrdevbase 驱测试 APP。 14 其他 : 使用方法: ./chrdevbaseApp /dev/chrdevbase <1>|<2> argv[2] 1:读文件 15 16 argv[2] 2:写文件 17论坛 : www.openedv.com 18 日志: 初版 V1.0 2019/1/30 正点原子创建 19 *********** 20 21 static char usrdata[] = {"usr data!"}; 22 23 /* 24 * @description : main 主程序 25 * @param - argc : argv 数组元素个数 26 * @param - argv : 具体参数 27 * @return : 0 成功;其他 失败 28 */ 29 int main(int argc, char *argv[]) 30 { 31 int fd, retvalue; 32 char *filename; 33 char readbuf[100], writebuf[100];



```
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
    34
    35
         if(argc != 3){
           printf("Error Usage!\r\n");
    36
    37
            return -1;
    38
         }
    39
    40
         filename = argv[1];
    41
         /* 打开驱动文件 */
    42
    43
         fd = open(filename, O_RDWR);
         if(fd < 0){
    44
    45
           printf("Can't open file %s\r\n", filename);
    46
           return -1;
         }
    47
    48
         if(atoi(argv[2]) == 1){ /* 从驱动文件读取数据 */
    49
    50
           retvalue = read(fd, readbuf, 50);
    51
           if(retvalue < 0){</pre>
    52
                printf("read file %s failed!\r\n", filename);
    53
            }else{
                /* 读取成功,打印出读取成功的数据 */
    54
    55
                printf("read data:%s\r\n",readbuf);
    56
           }
    57
         }
    58
    59
         if(atoi(argv[2]) == 2){
           /* 向设备驱动写数据 */
    60
    61
           memcpy(writebuf, usrdata, sizeof(usrdata));
           retvalue = write(fd, writebuf, 50);
    62
    63
           if(retvalue < 0){</pre>
                printf("write file %s failed!\r\n", filename);
    64
           }
    65
    66
         }
    67
    68
         /* 关闭设备 */
    69
         retvalue = close(fd);
    70
         if(retvalue < 0){
    71
           printf("Can't close file %s\r\n", filename);
    72
           return -1;
    73
         }
    74
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

75 **return** 0;

76 }

第 21 行,数组 usrdata 是测试 APP 要向 chrdevbase 设备写入的数据。

第 35 行,判断运行测试 APP 的时候输入的参数是不是为 3 个,main 函数的 argc 参数表示参数数量,argv[]保存着具体的参数,如果参数不为 3 个的话就表示测试 APP 用法错误。比如,现在要从 chrdevbase 设备中读取数据,需要输入如下命令:

./chrdevbaseApp /dev/chrdevbase 1

上述命令一共有三个参数"./chrdevbaseApp"、"/dev/chrdevbase"和"1",这三个参数分别对应 argv[0]、argv[1]和 argv[2]。第一个参数表示运行 chrdevbaseAPP 这个软件,第二 个参数表示测试 APP 要打开/dev/chrdevbase 这个设备。第三个参数就是要执行的操作,1表示 从 chrdevbase 中读取数据,2表示向 chrdevbase 写数据。

第40行,获取要打开的设备文件名字, argv[1]保存着设备名字。

第43行,调用C库中的open函数打开设备文件:/dev/chrdevbase。

第 49 行,判断 argv[2]参数的值是 1 还是 2,因为输入命令的时候其参数都是字符串格式 的,因此需要借助 atoi 函数将字符串格式的数字转换为真实的数字。

第 50 行,当 argv[2]为 1 的时候表示要从 chrdevbase 设备中读取数据,一共读取 50 字节 的数据,读取到的数据保存在 readbuf 中,读取成功以后就在终端上打印出读取到的数据。

第59行,当 argv[2]为2的时候表示要向 chrdevbase 设备写数据。

第69行,对 chrdevbase 设备操作完成以后就关闭设备。

chrdevbaseApp.c内容还是很简单的,就是最普通的文件打开、关闭和读写操作。

21.4.3 编译驱动程序和测试 APP

1、编译驱动程序

上面我们已经把本章的驱动代码和测试代码都编写好之后,在 1_chrdevbase 目录下,如下所示:

zy@zy-virtual-machine:~/linux/drivers/1_chrdevbase\$ zy@zy-virtual-machine:~/linux/drivers/1_chrdevbase\$ ls chrdevbaseAPP.c chrdevbase.c zy@zy-virtual-machine:~/linux/drivers/1_chrdevbase\$

图 21.4.3 源文件

首先编译驱动程序,也就是 chrdevbase.c 这个文件,我们需要将其编译为.ko 模块,在 1_chrdevbase 目录中创建一个 Makefile 文件,然后在其中输入如下内容:

示例代码 Makefile 内容示例

```
1 KERN_DIR := /home/zy/workspace/kernel-driver/linux-xlnx_rebase_v5.4_2020.2
2
3 obj-m := chrdevbase.o
4
5 all:
6 make -C $(KERN_DIR) M=`pwd` modules
7
```



正点原子

8 clean:

9 make -C \$(KERN_DIR) M=`pwd` clean

原子哥在线教学: www.yuanzige.com

第1行,KERN_DIR 表示开发板所使用的 Linux 内核源码目录,使用绝对路径,大家根据自己的实际情况填写即可。

第3行, obj-m 表示将 chrdevbase.c 这个文件编译为模块。

第6行,具体的编译命令,后面的 modules 表示编译模块,-C 表示将当前的工作目录切换到指定目录中,也就是 KERN_DIR 目录。M 表示模块源码目录`pwd`也就是当前目录,

"make modules" 命令中加入 M=dir 以后程序会自动到指定的 dir 目录中读取模块的源码并将 其编译为.ko 文件。

注意:不要直接 copy 文档中的 Makefile 内容,因为复制后格式不一定正确,该 Makefile 文件在我们提供的资料包中已经有了,路径:开发板光盘资料(A 盘) \4_SourceCode\3_Embedded_Linux\Linux 驱动例程\1_chrdevbase 目录中。

Makefile 编写好以后输入"make"命令编译驱动模块,编译过程如下图所示:

zy@zy-virtual-machine:~/linux/drivers/1_chrdevbase\$ ls

chrdevbaseAPP.c chrdevbase.c Makefile

zy@zy-virtual-machine:~/linux/drivers/1_chrdevbase\$ make +

make -C /home/zy/workspace/kernel-driver/linux-xlnx-xlnx_rebase_v5.4_2020.2 M=`p wd` modules

make[1]: 进入目录"/home/zy/workspace/kernel-driver/linux-xlnx-xlnx_rebase_v5.4_2 020.2"

CC [M] /home/zy/linux/drivers/1_chrdevbase/chrdevbase.o

Building modules, stage 2.

MODPOST 1 modules

CC [M] /home/zy/linux/drivers/1_chrdevbase/chrdevbase.mod.o

LD [M] /home/zy/linux/drivers/1_chrdevbase/chrdevbase.ko

make[1]: 离开目录"/home/zy/workspace/kernel-driver/linux-xlnx-xlnx_rebase_v5.4_2 020.2"

zy@zy-virtual-machine:~/linux/drivers/1_chrdevbase\$

图 21.4.4 驱动编译过程

编译成功以后就会在当期目录生成一个叫做 chrdevbaes.ko 的文件,此文件就是 chrdevbase 设备的驱动模块。至此, chrdevbase 设备的驱动就编译成功。

2、编译测试 APP

测试 APP 比较简单,只有一个文件,因此就不需要编写 Makefile 了,直接输入命令编译。因为测试 APP 是要在 ARM 开发板上运行的,所以需要使用交叉编译工具 arm-linux-gnueabihf-gcc 来编译,还记得 10.6 小节中交叉编译工具的使用吗? 输入如下命令:

\$CC chrdevbaseApp.c -o chrdevbaseApp

编译完成以后会生成一个叫做 chrdevbaseApp 的可执行程序,输入如下命令查看 chrdevbaseApp 这个程序的文件信息:

file chrdevbaseApp 结果如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php zy@zy-virtual-machine:~/linux/drivers/1_chrdevbase\$ \$CC chrdevbaseApp.c -o chrde vbaseApp _____ zy@zy-virtual-machine:~/linux/drivers/1_chrdevbase\$ ls chrdevbaseApp chrdevbase.ko chrdevbase.mod.o modules.order

chrdevbaseApp.c chrdevbase.mod chrdevbase.o Module.symvers
chrdevbase.c chrdevbase.mod.c Makefile
zy@zy-virtual-machine:~/linux/drivers/1 chrdevbase\$ file chrdevbaseApp _____
chrdevbaseApp: ELF 32-bit LSB shared object, ARM, EABI5 version 1 (SYSV), dynami
cally linked, interpreter /lib/ld-linux-armhf.so.3, BuildID[sha1]=95d2696db29449
19039b16151de40ed86d682edd, for GNU/Linux 3.2.0, with debug_info, not stripped
zy@zy-virtual-machine:~/linux/drivers/1 chrdevbase\$

图 21.4.5 chrdevbaseApp 文件信息

从上图可以看出, chrdevbaseApp 这个可执行文件是 32 位 LSB 格式, ARM 版本的, 因此 chrdevbaseApp 只能在 ARM 芯片下运行。

21.4.4 运行测试

1、加载驱动模块

驱动模块 chrdevbase.ko 和测试程序 chrdevbaseApp 都已经准备好了,接下来就是运行测试。

进入 Ubuntu 系统 NFS 共享目录下的根文件系统中,在/lib/modules/文件夹下新建名为 "5.4.0-xilinx"的文件夹,然后将 21.4.3 小节中编译好的驱动模块 chrdevbase.ko 和测试程序 chrdevbaseApp 复制到 "5.4.0-xilinx" 文件夹中,如下图所示:

```
zy@zy-virtual-machine:~/workspace/nfs/rootfs/lib/modules$ ls
5.4.0-xilinx-v2020.2
zy@zy-virtual-machine:~/workspace/nfs/rootfs/lib/modules$ mkdir 5.4.0-xilinx
zy@zy-virtual-machine:~/workspace/nfs/rootfs/lib/modules$ cd 5.4.0-xilinx
zy@zy-virtual-machine:~/workspace/nfs/rootfs/lib/modules$ cd 5.4.0-xilinx
zy@zy-virtual-machine:~/workspace/nfs/rootfs/lib/modules/5.4.0-xilinx$ cp /home/
zy/linux/drivers/1_chrdevbase/chrdevbase.ko .
zy@zy-virtual-machine:~/workspace/nfs/rootfs/lib/modules/5.4.0-xilinx$ cp /home/
zy/linux/drivers/1_chrdevbase/chrdevbaseApp .
zy@zy-virtual-machine:~/workspace/nfs/rootfs/lib/modules/5.4.0-xilinx$ cp /home/
zy/linux/drivers/1_chrdevbase/chrdevbaseApp .
zy@zy-virtual-machine:~/workspace/nfs/rootfs/lib/modules/5.4.0-xilinx$ ls
chrdevbaseApp chrdevbase.ko
zy@zy-virtual-machine:~/workspace/nfs/rootfs/lib/modules/5.4.0-xilinx$
```

图 21.4.6 将驱动模块和测试程序复制到创建的文件夹中

启动开发板进入 uboot 模式,设置环境变量,通过 NFS 方式加载根文件系统。系统启动 完成后,进入"/lib/modules/5.4.0-xilinx"目录下,此时便能看到驱动模块和测试程序两个文 件,如下图所示:



F占原子

或

modprobe chrdevbase.ko

如果使用 modprobe 加载驱动的话,可能会出现如下图所示的提示:

```
root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx# ls
chrdevbase.ko chrdevbaseApp
root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx# modprobe chrdevbase.ko
modprobe: can't open 'modules.dep': No such file or directory
root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx#
root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx#
```

图 21.4.8 modprobe 错误提示

从上图可以看出, modprobe 提示无法打开"modules.dep"这个文件, 因此驱动挂载失败 了。我们不用手动创建 modules.dep 这个文件, 直接输令"depmod"命令即可自动生成 modules.dep, 有些根文件系统可能没有 depmod 这个命令, 如果没有这个命令就只能重新配置 根文件系统, 使能此命令, 然后重新编译根文件系统。输入"depmod"命令以后会自动生成 modules.alias、modules.symbols 和 modules.dep 这三个文件, 如下图所示:

root@ALIENTEK	-ZYNQ-driver:/lib/mod	dules/5.4.0-xilin	ix#	
root@ALIENTEK	-ZYNQ-driver:/lib/mod	dules/5.4.0-xilin	x# depmod	
root@ALIENTEK	-ZYNQ-driver:/lib/mog	ules/5.4.0-xilin	x#ls	
chrdevbase.ko	chrdevbaseApp	modules.alias	modules.dep	modules.symbols
root@ALIENTEK	-ZYNQ-driver:/lib/mod	ules/5.4.0-xilin	x#	

图 21.4.9 depmod 执行结果

重新使用 modprobe 加载 chrdevbase.ko, 结果如下图所示:

```
root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx#
root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx# modprobe chrdevbase.ko
chrdevbase: loading out-of-tree module taints kernel.
chrdevbase_init()
root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx#
```

图 21.4.10 驱动加载成功

从上图可以看到"chrdevbase_init()"这一行,这一行正是 chrdevbase.c 中模块入口函数 chrdevbase_init 输出的信息,说明模块加载成功 (chrdevbase: loading out-of-tree module taints kernel.并不是我们的驱动模块打印出来的,这个打印信息跟内核的签名机制有关系,这里并 不影响我们的使用,先不用管)。

输入"lsmod"命令即可查看当前系统中存在的模块,结果如下图所示:



原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php

root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx#
root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx# lsmod
Tainted: G
chrdevbase 16384 0 - Live 0xbf000000 (0)
root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx#

图 21.4.11 当前系统中的模块

从上图可以看出,当前系统只有"chrdevbase"这一个模块。输入如下命令查看当前系统中有没有 chrdevbase 这个设备:

cat /proc/devices | grep 'chr' //查看当前系统中的所有设备 结果如下图所示:

root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx# root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx# cat /proc/devices | grep 'chr' 200 chrdevbase root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx#

图 21.4.12 当前系统中的设备

从上图可以看出,当前系统存在 chrdevbase 这个设备,主设备号为 200,跟我们设置的主 设备号一致。

2、创建设备节点文件

驱动加载成功需要在/dev 目录下创建一个与之对应的设备节点文件,应用程序就是通过 操作这个设备节点文件来完成对具体设备的操作。输入如下命令创建/dev/chrdevbase这个设备 节点文件:

mknod /dev/chrdevbase c 200 0

其中"mknod"是创建节点命令,"/dev/chrdevbase"是要创建的节点文件,"c"表示 这是个字符设备,"200"是设备的主设备号,"0"是设备的次设备号。创建完成以后就会 存在/dev/chrdevbase 这个文件,可以使用"ls /dev/chrdevbase -1"命令查看,结果如下图所示:

```
root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx#
root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx# mknod /dev/chrdevbase c 200 0
root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx#
root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx# ls /dev/chrdevbase -l
crw-r--r- 1 root root 200, 0 Jun 7 03:11 /dev/chrdevbase
root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx# _
```

图 21.4.13 chrdevbase 的设备文件

如果 chrdevbaseApp 想要读写 chrdevbase 设备,直接对/dev/chrdevbase 进行读写操作即可。 相当于/dev/chrdevbase 这个文件是 chrdevbase 设备在用户空间中的实现。前面一直说 Linux 下 一切皆文件,包括设备也是文件,现在大家应该是有这个概念了吧。

3、chrdevbase 设备操作测试

一切准备就绪,接下来就是"大考"的时刻了。使用 chrdevbaseApp 软件操作 chrdevbase 这个设备,看看读写是否正常,首先进行读操作,输入如下命令:

```
./chrdevbaseApp /dev/chrdevbase 1
```

结果如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx#

root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx# ./chrdevbaseApp /dev/chrdevbase 1 chrdevbase open! kernel senddata ok!

read data:kernel data!

root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx#

图 21.4.14 读操作结果

从上图可以看出,首先输出"kernel senddata ok!"这一行信息,这是驱动程序中 chrdevbase_read函数输出的信息,因为 chrdevbaseApp 使用 read函数从 chrdevbase 设备读取数据,因此 chrdevbase_read函数就会执行。chrdevbase_read函数向 chrdevbaseApp 发送"kernel data!"数据, chrdevbaseApp 接收到以后就打印出来,"read data:kernel data!"就是 chrdevbaseApp 打印出来的接收到的数据。说明对 chrdevbase 的读操作正常,接下来测试对 chrdevbase 设备的写操作,输入如下命令:

./chrdevbaseApp /dev/chrdevbase 2 结果如下图所示:

```
root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx#
root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx# ./chrdevbaseApp /dev/chrdevbase 2
chrdevbase open!
kernel recevdata:usr data!
root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx#
```

图 21.4.15 写操作结果

第二行"kernel recevdata:usr data!",这个是驱动程序中的 chrdevbase_write 函数输出的。 chrdevbaseApp 使用 write 函数向 chrdevbase 设备写入数据"usr data!"。chrdevbase_write 函数 接收到以后将其打印出来。说明对 chrdevbase 的写操作正常,既然读写都没问题,说明我们 编写的 chrdevbase 驱动是没有问题的。

4、卸载驱动模块

如果不再使用某个设备的话可以将其驱动卸载掉,比如输入如下命令卸载掉 chrdevbase 这个设备:

rmmod chrdevbase.ko

卸载以后使用 lsmod 命令查看 chrdevbase 这个模块还存不存在,结果如下图所示:

```
root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx#
root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx# rmmod chrdevbase.ko
chrdevbase_exit()
root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx# lsmod
Tainted: G
root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx#
```

图 21.4.16 卸载模块

从上图可以看出,此时系统已经没有任何模块了,chrdevbase这个模块也不存在了,说明 模块卸载成功。

至此, chrdevbase这个设备的整个驱动就验证完成了, 驱动工作正常。本章我们详细的讲解了字符设备驱动的开发步骤, 并且以一个虚拟的 chrdevbase 设备为例, 带领大家完成了第一个字符设备驱动的开发, 掌握了字符设备驱动的开发框架以及测试方法, 以后的字符设备驱动实验基本都以此为蓝本。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

第二十二章 嵌入式 Linux LED 驱动开发实验

上一章我们详细的讲解了字符设备驱动开发步骤,并且用一个虚拟的 chrdevbase 设备为 例带领大家完成了第一个字符设备驱动的开发。本章我们就开始编写第一个真正的 Linux 字 符设备驱动。在领航者开发板上有6个LED灯,在《领航者 ZYNQ之嵌入式开发指南》中已 经编写过 LED 灯的裸机驱动,本章我们就来学习一下如何编写 Linux 下的 LED 灯驱动程序, 本章我们只驱动底板上的 PS_LED0 灯。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

22.1 Linux 下 LED 灯驱动原理

Linux 下的任何外设驱动,最终都是要配置相应的硬件寄存器。所以本章的 LED 灯驱动 最终也是对 ZYNQ 的 IO 口进行配置,与裸机实验不同的是,在 Linux 下编写驱动要符合 Linux 的驱动框架。领航者开发板上的 PS LED0 连接到 ZYNQ 的 MIO7 这个引脚上,因此本章实验 的重点就是编写 Linux 下 ZYNQ GPIO 引脚控制驱动。关于 ZYNQ 的 GPIO 详细讲解请参考 《领航者 ZYNO 之嵌入式开发指南》第二章。

22.1.1 地址映射

在编写驱动之前,我们需要先简单了解一下 MMU 这个神器, MMU 全称叫做 Memory Manage Unit,也就是内存管理单元。在老版本的 Linux 中要求处理器必须有 MMU,但是现在 Linux 内核已经支持无 MMU 的处理器了。MMU 主要完成的功能如下:

① 完成虚拟空间到物理空间的映射。

② 内存保护,设置存储器的访问权限,设置虚拟存储空间的缓冲特性。

我们重点来看一下第(1)点,也就是虚拟空间到物理空间的映射,也叫做地址映射。首先 了解两个地址概念: 虚拟地址(VA, Virtual Address)、物理地址(PA, Physcical Address)。对于 32 位的处理器来说, 虚拟地址范围是 2^32=4GB, 我们的 7010 核心板搭配的是 512MB 的 DDR3(7020是1GB),这 512MB的内存就是物理内存,经过 MMU 可以将其映射到整个 4GB的虚拟空间,如下图所示:



图 22.1.1 内存映射



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

物理内存只有 512MB,虚拟内存有 4GB,那么肯定存在多个虚拟地址映射到同一个物理 地址上去,虚拟地址范围比物理地址范围大的问题处理器自会处理,这里我们不要去深究, 因为 MMU 是很复杂的一个东西,后续有时间的话正点原子 Linux 团队会专门做 MMU 专题教 程。

Linux 内核启动的时候会初始化 MMU,设置好内存映射,设置好以后 CPU 访问的都是虚 拟地址。比如 ZYNQ 的 GPIO 模块寄存器基地址为 0xE000A000。如果没有开启 MMU 的话直 接读写 0xE000A000 这个地址是没有问题的,现在开启了 MMU,并且设置了内存映射,因此 就不能直接向 0xE000A000 这个地址写入数据了。我们必须得到 0xE000A000 这个物理地址在 Linux 系统里面对应的虚拟地址,这里就涉及到了物理内存和虚拟内存之间的转换,需要用到 两个函数: ioremap 和 iounmap。

1、ioremap 函数

ioremap 函数用于获取指定物理地址空间对应的虚拟地址空间,定义在arch/arm/include/asm/io.h文件中,定义如下:

示例代码 22.1.1 ioremap 函数

1 #define ioremap(cookie,size) __arm_ioremap((cookie), (size), MT_DEVICE)

2

3 void __iomem * __arm_ioremap(phys_addr_t phys_addr, size_t size, unsigned int mtype)

4 {

5 return arch_ioremap_caller(phys_addr, size, mtype, __builtin_return_address(0));

6 }

ioremap 是个宏,有两个参数: cookie 和 size,真正起作用的是函数__arm_ioremap,此函数有三个参数和一个返回值,这些参数和返回值的含义如下:

phys_addr: 要映射给的物理起始地址。

size: 要映射的内存空间大小。

mtype: ioremap 的类型,可以选择 MT_DEVICE、MT_DEVICE_NONSHARED、 MT_DEVICE_CACHED 和 MT_DEVICE_WC, ioremap 函数选择 MT_DEVICE。

返回值:___iomem 类型的指针,指向映射后的虚拟空间首地址。

假如我们要获取 ZYNQ 的 APER_CLK_CTRL 寄存器对应的虚拟地址,使用如下代码即可:

#define APER_CLK_CTRL 0xF800012C

static void __iomem *aper_clk_ctrl_addr;

aper_clk_ctrl_addr = ioremap(APER_CLK_CTRL, 4);

宏定义 APER_CLK_CTRL 是寄存器物理地址, aper_clk_ctrl_addr 是该物理地址映射后的 虚拟地址。对于 ZYNQ 来说一个寄存器是 4 字节(32 位)的,因此映射的内存长度为 4。映射完 成以后直接对 aper_clk_ctrl_addr 进行读写操作即可。

2、iounmap 函数

卸载驱动的时候需要使用 iounmap 函数释放掉 ioremap 函数所做的映射, iounmap 函数原型如下:

示例代码 22.1.2 iounmap 函数原型

```
void iounmap (volatile void __iomem *addr)
```



原子哥在线教学: www.vuanzige.com 论均

论坛:www.openedv.com/forum.php

iounmap只有一个参数 addr,此参数就是要取消映射的虚拟地址空间首地址。假如我们现在要取消掉 APER_CLK_CTRL 寄存器的地址映射,使用如下代码即可:

iounmap(aper_clk_ctrl_addr);

22.1.2 I/O 内存访问函数

这里说的 I/O 是输入/输出的意思,并不是我们学习单片机的时候讲的 GPIO 引脚。这里涉及到两个概念: I/O 端口和 I/O 内存。当外部寄存器或内存映射到 IO 空间时,称为 I/O 端口。 当外部寄存器或内存映射到内存空间时,称为 I/O 内存。但是对于 ARM 来说没有 I/O 空间这 个概念,因此 ARM 体系下只有 I/O 内存(可以直接理解为内存)。使用 ioremap 函数将寄存器 的物理地址映射到虚拟地址以后,我们就可以直接通过指针访问这些地址,但是 Linux 内核 不建议这么做,而是推荐使用一组操作函数来对映射后的内存进行读写操作。

1、读操作函数

读操作函数有如下几个:

示例代码 22.1.3 读操作函数

1 u8 readb(const volatile void __iomem *addr)

2 u16 readw(const volatile void __iomem *addr)

3 u32 readl(const volatile void __iomem *addr)

readb、readw 和 readl 这三个函数分别对应 8bit、16bit 和 32bit 读操作,参数 addr 就是要 读取写内存地址,返回值就是读取到的数据。

2、写操作函数

写操作函数有如下几个:

示例代码 22.1.4 写操作函数

1 void writeb(u8 value, volatile void __iomem *addr)

2 void writew(u16 value, volatile void __iomem *addr)

3 void writel(u32 value, volatile void __iomem *addr)

writeb、writew 和 writel 这三个函数分别对应 8bit、16bit 和 32bit 写操作,参数 value 是要 写入的数值, addr 是要写入的地址。

22.2 ZYNQ GPIO 相关寄存器讲解

在《领航者 ZYNQ 之嵌入式开发指南》文档的第二章中给大家详细的介绍过 ZYNQ 的 GPIO 模块,如果大家不明白的可以看看该文档;因为本小节我们将直接对 GPIO 相关的寄存 器进行操作,所以在此之前有必要向大家简单地说明下 ZYNQ GPIO 相关的寄存器控制。在我 们提供给大家的资料包中有一份文档,路径为:开发板资料盘(A 盘)\8_ZYNQ&FPGA 参考资料 \Xilinx\User Guide\ug585-Zynq-7000-TRM.pdf;找到 Appx. B: Register Details 章中的 General Purpose I/O (gpio)小节,也就是文档的 1346页,如下所示:

🗄 🖳 🗖 Appx. B: Register Details U Overview

CAN Controller (can)

Debug Access Port (dap)

.... DMA Controller (dmac) Gigabit Ethernet Controller (GEM) General Purpose I/O (gpio) ☐ Interconnect QoS (qos301)

... CoreSight Embedded Trace Buffer (etb) PL Fabric Trace Monitor (ftm) ... CoreSight Trace Funnel (funnel) 💭 CoreSight Intstrumentation Trace Macroc ... CoreSight Trace Packet Output (tpiu) Device Configuration Interface (devcfg)

NIC301 Address Region Control (nic301_a

Acronyms Module Summary



Register Summary

Register Name	Address	Width	Туре	Reset Value
XGPIOPS_DATA_LSW_O FFSET	0x0000000	32	mixed	x
XGPIOPS_DATA_MSW_ OFFSET	0x00000004	32	mixed	x
MASK_DATA_1_LSW	0x0000008	32	mixed	x
MASK_DATA_1_MSW	0x000000C	22	mixed	x

图 22.2.1 gpio 寄存器

在这里可以看到 ZYNO GPIO 寄存器的基地址是 0xE000A000, 接下来重点给大家介绍 DATA 寄存器、DIRM 寄存器、OUTEN 寄存器以及 INTDIS 寄存器。

22.2.1 DATA 寄存器

XGPIOPS_DATA_OFFSET	0x0000040	32	rw	x	Output Data (GPIO Bank0, MIO)
DATA_1	0x0000044	22	rw	x	Output Data (GPIO Bank1, MIO)
DATA_2	0x0000048	32	rw	0x0000000	Output Data (GPIO Bank2, EMIO)
DATA_3	0x000004C	32	rw	0x0000000	Output Data (GPIO Bank3, EMIO)

图 22.2.2 DATA 寄存器

从上面的描述信息就可以知道, DATA 寄存器就是控制管脚输出高低电平的, 所以在输 出模式下,就可以通过对寄存器的相应 bit 位写 0 或 1 来控制某个 GPIO 输出电平为高还是低。 这里面有4个,前面2个(XGPIOPS DATA、DATA1)控制的是ZYNQ的Bank0和Bank1两 组的 GPIO, 而 DATA_2 和 DATA_3 则是控制 EMIO 中的 GPIO; 因为本实验以 PS_LED0 为 例,它连接的是 MIO7,所以可以通过控制第一个 DATA 寄存器(寄存器地址偏移量为 0x40) 的 bit7 位来控制该管脚,例如下面的代码可以将 MIO7 管脚拉低:

val = readl(data_addr);	// 读取 data 寄存器数据
val &= ~(0x1U << 7);	// 将 bit7 位数据清零
writel(val, data_addr);	// 将数据写入寄存器

22.2.2 DIRM 寄存器

DIRM 是 Direction mode 的缩写,所以从名字可以知道该寄存器是控制 GPIO 的输入、输 出方向的:



原子哥在线教学:ww	w.yuanzige.com	n	论坛:w	www.openedv.c	om/forum.php
XGPIOPS_DIRM_OFFSE	0x0000204	32	rw	0x0000000	Direction mode (GPIO Bank0,
<u>T</u>					MIO)

图 22.2.3 DIRM 寄存器

同样 DIRM 寄存器也有 4 个,因为我们控制的是 MIO7 管脚,所以只需要对第一个 DIRM 寄存器进行控制即可,它对应的寄存器地址偏移量为 0x204。

Field Name	Bits	Туре	Reset Value	Description
DIRECTION_0	31:0	rw	0x0	Direction mode
				0: input
				1: output
				Each bit configures the corresponding pin within the 32-bit bank
				NOTE: bits[8:7] of bank0 cannot be used as inputs.
				The DIRM bits can be set to 0, but reading DATA_RO does not reflect the input value. See the GPIO chapter for more information.

图 22.2.4 DIRM 寄存器描述

从寄存器的描述信息可以知道,向寄存器相应的 bit 位写入 0 则表示将该 GPIO 作为输入 模式,写入 1 则表示将该 GPIO 作为输出模式,是吧,非常的简单,没有任何花里胡哨的东 西!

22.2.3 OUTEN 寄存器

看这个名字就知道这个寄存器是干啥的,输出使能嘛;也就是啥呢,你上面将GPIO设置 为输出模式后还不能真正的作为输出使用,你还得使能它才行。

图 22.2.5 OUTEN 寄存器

Field Name	Bits	Туре	Reset Value	Description
OP_ENABLE_0	31:0	rw	0×0	Output enables 0: disabled 1: enabled Each bit configures the corresponding pin within the 32-bit bank

图 22.2.6 OUTEN 寄存器描述

从上面的描述信息可以知道,将 bit 位置 1 就是使能输出功能,置 0 则禁止输出功能;同 样这个寄存器也有 4 个,对于 MIO7 我们只需要控制第一个即可,地址偏移量为 0x208。

22.2.4 INTDIS 寄存器

INTDIS 就是 Interrupt Disable 的缩写,所以该寄存器是用来禁止 GPIO 中断功能的:

XGPIOPS_INTDIS_OFFS ET	0x00000214	32	wo	0x0000000	Interrupt Disable/Mask (GPIO Bank0, MIO)		
图 22.2.7 INTDIS 寄存器							



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

同样写入1表示禁止中断,但是写入0并不是使能中断功能,它的中断使能和禁止是使 用两个不同的寄存器来控制的,这里大家要清楚。

Field Name	Bits	Туре	Reset Value	Description
INT_DISABLE_0	31:0	wo	0x0	Interrupt disable 0: no change 1: set interrupt mask Each bit configures the corresponding pin within the 32-bit bank

图 22.2.8 INTDIS 寄存器描述

同样该寄存器也有 4 个,对于 MIO7,我们只需要控制第一个即可,它的地址偏移量为 0x214。

22.2.5 GPIO 时钟

我们需要把GPIO模块的时钟给打开,GPIO才能正常工作,那么如何使能GPIO时钟呢? 我们可以通过控制 APER_CLK_CTRL 寄存器。APER_CLK_CTRL 寄存器属于系统级别的控 制寄存器(system level control register),打开收据手册的 Apx. B: Register Details 章中的 System Level Control Register 小节中找到该寄存器:

🛨 📕 Appx. A: Additional Resources	
🖻 📜 Appx. B: Register Details	
Overview	
Module Summary	
AXI_HP Interface (AFI) (axi_hp)	
CAN Controller (can)	
DDR Memory Controller (ddrc)	
CoreSight Cross Trigger Interface (cti)	
Performance Monitor Unit (cortexa9_pmu)	
CoreSight Program Trace Macrocell (ptm)	
Debug Access Port (dap)	
CoreSight Embedded Trace Buffer (etb)	
PL Fabric Trace Monitor (ftm)	
CoreSight Trace Funnel (funnel)	
CoreSight Intstrumentation Trace Macrocell (itm)	
CoreSight Trace Packet Output (tpiu)	
Device Configuration Interface (devcfg)	
DMA Controller (dmac)	
Gigabit Ethernet Controller (GEM)	
General Purpose I/O (gpio)	
Interconnect QoS (gos301)	
NIC301 Address Region Control (nic301 addr region ctrl r	
I2C Controller (IIC)	
L2 Cache (L2Cpl310)	
Application Processing Unit (mpcore)	
On-Chip Memory (ocm)	
Quad-SPI Flash Controller (gspi)	
SD Controller (sdio)	
System Level Control Registers (slcr)	
Static Memory Controller (pl353)	

Description System Level Cont Vendor Info Xilinx Zyng slcr

Register Summary

Register Name	Address	V
<u>SCL</u>	0x0000000	3
SLCR_LOCK	0x0000004	3
SLCR_UNLOCK	0x0000008	3
SLCR_LOCKSTA	0x000000C	3
ARM_PLL_CTRL	0x00000100	3
DDR_PLL_CTRL	0x00000104	3
IO_PLL_CTRL	0x00000108	3
PLL_STATUS	0x0000010C	3

图 22.2.9 SLRC 部分寄存器

我们可以看到 SLRC 部分寄存器的基地址是 0xF8000000, 往下看可以找到我们这里说的 APER CLK CTRL 寄存器,如下所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

DCI_CLK_CTRL	0x00000128	32	rw	0x01E03201	DCI clock control
APER_CLK_CTRL	0x0000012C	32	rw	0x01FFCCCD	AMBA Peripheral Clock Control
USB0_CLK_CTRL	0x00000130	32	rw	0x00101941	USB 0 ULPI Clock Control

图 22.2.10 APER CLK CTRL 寄存器

该寄存器的地址偏移量为 0x12C,加上前面的基地址,所以可以知道该寄存器地址的绝 对偏移量为0xF800012C。点击蓝色字体跳转到该寄存器的描述页面:

Field Name	Bits	Туре	Reset Value	Description	
reserved	31:25	rw	0x0	Reserved. Writes are ignored, read data is zero	0.
SMC_CPU_1XCLKACT	24	rw	0x1	SMC AMBA Clock control 0: disable, 1: enable	
LQSPI_CPU_1XCLKACT	23	rw	0x1	Quad SPI AMBA Clock control 0: disable, 1: enable	
GPIO_CPU_1XCLKACT	22	rw	0x1	GPIO AMBA Clock control 0: disable, 1: enable	
UART1_CPU_1XCLKACT	21	rw	0x1	UART 1 AMBA Clock control 0: disable, 1: enable	
UART0_CPU_1XCLKACT	20	rw	0x1	UART 0 AMBA Clock control 0: disable, 1: enable	
I2C1_CPU_1XCLKACT	19	rw	0x1	I2C 1 AMBA Clock control 0: disable, 1: enable	
I2C0_CPU_1XCLKACT	18	rw	0x1	I2C 0 AMBA Clock control	

图 22.2.11 APER CLK CTRL 描述信息

该寄存器用于控制 ZYNQ AMBA 外设时钟,从图中可以知道该寄存器的 bit22 位 GPIO 时 钟控制位,向该位写入0禁止GPIO时钟,写入1则使能GPIO时钟,所以目标就很明确了! 上面已经向大家介绍了本章需要使用到的所有寄存器了,那么在驱动源码中就不给大家 再去一一讲解了! 细心的同学可能会发现, 为什么没有关于管脚电气特性相关的初始化呢? 例如什么上下拉、什么 IO 速率等之类的,那么这些问题留给大家去想一想,我们会在后面的 章节给大家解答!

22.3 硬件原理图分析

本章我们以领航者底板上的 PS_LED0 为例,打开我们提供给大家的领航者开发板的底板 原理图,LED部分原理图如下所示:



原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php

LED



图 22.3.1 LED 原理图

	SD D3	J2 88	00
PS KEY0	PS MIO12	J2 90	00
PS LED0	PS MIO7	J2 92	90
PS KEY1	PS MIO11	J2 94	92
OTG RESETN	PS MIO9	J2 96	94

图 22.3.2 PS_LEDO 绑定的管脚

从图 22.3.1 中可以知道,当 PS_LED0 输出为高电平的时候点亮 LED,相反低电平的时候 LED 灭。从图 22.3.2 中知道 PS_LED0 与 ZYNQ 的 MIO7 引脚相连。所以接下来我们需要做的 就是对 MIO7 引脚进行控制输出高电平点亮 LED,输出低电平则 LED 灭!

22.4 实验程序编写

本实验对应的例程路径为: 开发板光盘资料 (A 盘) \4_SourceCode\3_Embedded_Linux\Linux 驱动例程\2_led。

本章实验编写 Linux 下的 LED 灯驱动,可以通过应用程序对开发板上的 LED 灯进行开关操作。

22.4.1 编写 LED 灯驱动程序

在 drivers 目录下新建名为"2_led"的文件夹,如下所示:

```
zy@zy-virtual-machine:~/linux/drivers$ ls
1_chrdevbase
zy@zy-virtual-machine:~/linux/drivers$ mkdir 2_led
zy@zy-virtual-machine:~/linux/drivers$ ls
1_chrdevbase 2_led
zy@zy-virtual-machine:~/linux/drivers$
图 22.4.1 2 led 文件夹
```

进入到 2_led 目录下,新建一个名为 led.c 的驱动源文件,输入如下内容: 示例代码 22.4.1 led.c 驱动示例代码



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php 2 Copyright © ALIENTEK Co., Ltd. 1998-2029. All rights reserved. 3 文件名 : led.c 4 作者 :邓涛 5 版本 : V1.0 6 描述 : ZYNQ LED 驱动文件。 7 其他 :无 8 论坛 : www.openedv.com 9 日志 :初版 V1.0 2019/1/30 邓涛创建 11 12 #include <linux/types.h> 13 #include <linux/kernel.h> 14 #include <linux/delay.h> 15 #include <linux/ide.h> 16 #include <linux/init.h> 17 #include <linux/module.h> 18 #include <linux/errno.h> 19 #include <linux/gpio.h> 20 #include <asm/mach/map.h> 21 #include <asm/uaccess.h> 22 #include <asm/io.h> 23 24 #define LED_MAJOR 200 /* 主设备号 */ 25 #define LED_NAME "led" /* 设备名字 */ 26 27 /* 28 * GPIO 相关寄存器地址定义 29 */ 30 #define ZYNQ_GPIO_REG_BASE0xE000A000 31 #define DATA_OFFSET 0x00000040 32 #define DIRM_OFFSET 0x0000204 33 #define OUTEN_OFFSET 0x0000208 34 #define INTDIS_OFFSET 0x00000214 35 #define APER_CLK_CTRL 0xF800012C 36 37 /* 映射后的寄存器虚拟地址指针 */ 38 static void __iomem *data_addr; 39 static void __iomem *dirm_addr; 40 static void __iomem *outen_addr; 41 static void __iomem *intdis_addr;

```
原子哥在线教学: www.yuanzige.com
                                   论坛:www.openedv.com/forum.php
   42 static void __iomem *aper_clk_ctrl_addr;
   43
   44
   45 /*
   46 * @description
                        :打开设备
   47 * @param – inode
                            :传递给驱动的 inode
                            :设备文件, file 结构体有个叫做 private_data 的成员变量
   48 * @param – filp
   49 *
                             一般在 open 的时候将 private_data 指向设备结构体。
                         :0 成功;其他 失败
   50 * @return
   51 */
   52 static int led_open(struct inode *inode, struct file *filp)
   53 {
   54 return 0;
   55 }
   56
   57 /*
                       :从设备读取数据
   58 * @description
   59 * @param – filp
                            :要打开的设备文件(文件描述符)
   60 * @param – buf
                            :返回给用户空间的数据缓冲区
   61 * @param – cnt
                            :要读取的数据长度
   62 * @param – offt
                            :相对于文件首地址的偏移
   63 * @return
                         :读取的字节数,如果为负值,表示读取失败
   64 */
   65 static ssize_t led_read(struct file *filp, char __user *buf,
                size_t cnt, loff_t *offt)
   66
   67 {
   68 return 0;
   69 }
   70
   71 /*
   72 * @description
                    : 向设备写数据
   73 * @param – filp
                            :设备文件,表示打开的文件描述符
   74 * @param – buf
                            :要写给设备写入的数据
   75 * @param – cnt
                            :要写入的数据长度
   76 * @param – offt
                            :相对于文件首地址的偏移
                        :写入的字节数,如果为负值,表示写入失败
   77 * @return
   78 */
   79 static ssize_t led_write(struct file *filp, const char __user *buf,
   80
                size_t cnt, loff_t *offt)
   81 {
   82 int ret;
```

正点原子



```
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
    83
         int val;
    84
         char kern_buf[1];
    85
         ret = copy_from_user(kern_buf, buf, cnt);// 得到应用层传递过来的数据
    86
        if(0 > ret) {
    87
    88
           printk(KERN_ERR "kernel write failed!\r\n");
    89
           return -EFAULT;
    90
        }
    91
    92 val = readl(data_addr);
    93
        if (0 == kern_buf[0])
                                 // 如果传递过来的数据是 0 则关闭 led
    94
          val \& = \sim (0 \times 10 \ll 7);
    95
        else if (1 == \text{kern}_{\text{buf}})
          val |= (0x1U << 7);
                             // 如果传递过来的数据是 1 则点亮 led
    96
    97
    98
        writel(val, data_addr);
    99
        return 0;
    100 }
    101
    102 /*
    103 * @description
                               :关闭/释放设备
    104 * @param – filp
                                 :要关闭的设备文件(文件描述符)
    105 * @return
                                 :0 成功;其他 失败
    106 */
    107 static int led_release(struct inode *inode, struct file *filp)
    108 {
    109 return 0;
    110 }
    111
    112 /* 设备操作函数 */
    113 static struct file_operations led_fops = {
    114 .owner = THIS_MODULE,
    115 .open = led_open,
    116 .read = led_read,
    117 .write = led_write,
    118 .release = led_release,
    119 };
    120
    121 static int __init led_init(void)
    122 {
    123 u32 val;
```



```
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
    124
         int ret;
    125
         /* 1.寄存器地址映射 */
    126
    127
         data_addr = ioremap(ZYNQ_GPIO_REG_BASE + DATA_OFFSET, 4);
    128
         dirm_addr = ioremap(ZYNQ_GPIO_REG_BASE + DIRM_OFFSET, 4);
    129
         outen addr = ioremap(ZYNQ GPIO REG BASE + OUTEN OFFSET, 4);
         intdis_addr = ioremap(ZYNQ_GPIO_REG_BASE + INTDIS_OFFSET, 4);
    130
    131
         aper_clk_ctrl_addr = ioremap(APER_CLK_CTRL, 4);
    132
         /* 2.使能 GPIO 时钟 */
    133
    134
         val = readl(aper_clk_ctrl_addr);
         val = (0x1U << 22);
    135
    136
         writel(val, aper_clk_ctrl_addr);
    137
    138
         /* 3.关闭中断功能 */
    139
         val |= (0x1U << 7);
    140
         writel(val, intdis_addr);
    141
    142
         /* 4.设置 GPIO 为输出功能 */
    143
         val = readl(dirm_addr);
    144
         val |= (0x1U << 7);
    145
         writel(val, dirm_addr);
    146
    147
         /* 5.使能 GPIO 输出功能 */
    148
         val = readl(outen_addr);
    149
         val = (0x1U << 7);
    150
         writel(val, outen_addr);
    151
         /* 6.默认关闭 LED */
    152
    153
         val = readl(data_addr);
         val \& = \sim (0 \times 10 \ll 7);
    154
    155
         writel(val, data_addr);
    156
         /* 7.注册字符设备驱动 */
    157
    158
         ret = register_chrdev(LED_MAJOR, LED_NAME, &led_fops);
    159
         if(0 > ret){
    160
           printk(KERN_ERR "Register LED driver failed!\r\n");
    161
           return ret;
    162
         }
    163
    164
         return 0;
```



原子哥在线教学·www.yuanzige.com 论坛·www.onenedy.com/forum.php
165 }
166
167 static voidexit led_exit(void)
168 {
169 /* 1.卸载设备 */
170 unregister_chrdev(LED_MAJOR, LED_NAME);
171
172 /* 2.取消内存映射 */
173 iounmap(data_addr);
174 iounmap(dirm_addr);
175 iounmap(outen_addr);
176 iounmap(intdis_addr);
177 iounmap(aper_clk_ctrl_addr);
178 }
179
180 /* 驱动模块入口和出口函数注册 */
181 module_init(led_init);
182 module_exit(led_exit);
183
184 MODULE_AUTHOR("DengTao <773904075@qq.com>");
185 MODULE_DESCRIPTION("Alientek ZYNQ GPIO LED Driver");
186 MODULE_LICENSE("GPL");
第 24~25 行,定义了两个宏,设备名字和设备的主设备号。
第30~35行,本头验要用到的奇存器宏定义。
第 38~42 行,经过内存映射以后的奇存器地址指针。
第 52~55 行,led_open 函数,为全函数,可以目行任此函数屮添加相天内谷,一般任此
图数中将设备结构体作为参数 filp 的私有数据(filp->private_data)。
弗 65~69 行,led_read 函数,为全函数,如未忽住应用柱序中读取 LED 的状态,那么别 可以在业函数中还加出应的优现。比加法取 MO 的 DATA 客方照的店。就后近回公应用租
可以任此函数中你加阳匹的气屿, 比如误取 MIO 的 DATA 苛仔奋的值, 然后返凹宕应用柱 它
了。
另 / 7~100 门,IEU_WIIIE 函数,天坑/Y LEU 灯的月天深下, 当应用在序调用 WIIIE 函数问 动身体的 化合物合物 合地 行。 首先通过函数 convition waar 基面应用程序会送过本
icu 以田与奴殖的时候此函数机云环门。目元迪过函数 Copy_Hom_user 获取应用柱序及运过术

16 亨发送过来 的操作信息(打开还是关闭 LED),最后根据应用程序的操作信息来控制寄存器打开或关闭 LED灯。

第107~110行, led_release 函数, 为空函数, 可以自行在此函数中添加相关内容, 一般关 闭设备的时候会释放掉 led_open 函数中添加的私有数据。

第113~119行,设备文件操作结构体 led_fops 的定义和初始化。

第 121~165 行, 驱动入口函数 led_init, 此函数实现了 LED 的初始化工作, 127~131 行通 过 ioremap 函数获取物理寄存器地址映射后的虚拟地址,得到寄存器对应的虚拟地址以后就可 以完成相关初始化工作了。比如使能 GPIO 时钟、关闭 MIO7 的中断功能、配置并使能 MIO7 的输出功能等。最后,最重要的一步!使用 register_chrdev 函数注册 led 这个字符设备。



原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php

第167~178行,驱动出口函数 led_exit,首先使用函数 unregister_chrdev 注销 led 这个字符 设备,然后调用 iounmap 函数取消内存映射,因为设备已经被卸载,也就意味用不到了,必 须要取消映射;需要注意的是这两顺序不要反了,不能在设备没有卸载的情况下,你就把人 家的内存映射给取消了,这是不合理的!

第 181~182 行,使用 module_init 和 module_exit 这两个函数指定 led 设备驱动加载和卸载 函数。

第184~186行,添加模块LICENSE、作者信息以及模块描述信息。

22.4.2 编写测试 App

编写测试 APP, led 驱动加载成功以后手动创建/dev/led 节点,应用 APP 通过操作/dev/led 文件来完成对 LED 设备的控制。向/dev/led 文件写 0 表示关闭 LED 灯,写 1 表示打开 LED 灯。 在我们的 drivers 目录下新建一个名为 led App.c 测试文件,在里面输入如下内容:

示例代码 22.4.2 ledApp.c 示例代码

```
2 Copyright © ALIENTEK Co., Ltd. 1998-2029. All rights reserved.
3 文件名
              : led-app.c
4 作者
             :邓涛
5 版本
             : V1.0
6 描述
             :LED 驱测试 APP。
7 其他
             :无
8 使用方法
             : ./ledApp /dev/led 0 关闭 LED
9
             ./ledApp /dev/led 1 打开 LED
10 论坛
             : www.openedv.com
11 日志
             :初版 V1.0 2019/1/30 邓涛创建
13
14 #include <stdio.h>
15 #include <unistd.h>
16 #include <sys/types.h>
17 #include <sys/stat.h>
18 #include <fcntl.h>
19 #include <stdlib.h>
20 #include <string.h>
21
22 /*
23 * @description
               :main 主程序
24 * @param - argc : argv 数组元素个数
25 * @param - argv : 具体参数
26 * @return
               :0 成功;其他 失败
27 */
28 int main(int argc, char *argv[])
```



```
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
    29 {
    30
         int fd, ret;
         unsigned char buf[1];
    31
    32
    33
         if(3 != argc) {
           printf("Usage:\n"
    34
    35
               "\t./ledApp /dev/led 1
                                    @ close LED\n"
    36
               "\t./ledApp /dev/led 0
                                   @ open LED\n"
    37
               );
    38
           return -1;
    39
        }
    40
    41 /* 打开设备 */
    42 fd = open(argv[1], O_RDWR);
         if(0 > fd) \{
    43
    44
           printf("file %s open failed!\r\n", argv[1]);
    45
           return -1;
    46
        }
    47
    48
        /* 将字符串转换为 int 型数据 */
    49
         buf[0] = atoi(argv[2]);
    50
    51 /* 向驱动写入数据 */
    52 ret = write(fd, buf, sizeof(buf));
    53 if(0 > ret){
    54
           printf("LED Control Failed!\r\n");
           close(fd);
    55
    56
           return -1;
    57 }
    58
    59 /* 关闭设备 */
    60 close(fd);
    61
         return 0;
    62 }
```

ledApp.c的内容还是很简单的,就是对 led 的驱动文件进行最基本的打开、关闭、写操作等,具体代码就不给大家进行讲解了!

22.5 运行测试

22.5.1 编译驱动程序和测试 App

1、编译驱动程序



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

首先我们还是得需要一个 Makefile 文件,直接将 21.4.3 小节使用的 Makefile 文件拷贝到本实验目录下(2_led 目录下),然后打开 Makefile 文件,将 obj-m 变量的值改为 led.o, Makefile 内容如下所示:

示例代码 22.5.1 Makefile 文件内容示例

1 KERN_DIR := /home/zy/workspace/kernel-driver/linux-xlnx_rebase_v5.4_2020.2
3 obj-m := led.o
4
5 all:
6 make -C \$(KERN_DIR) M=`pwd` modules
7
8 clean:
9 make -C \$(KERN_DIR) M=`pwd` clean
第 3 行,设置 obj-m 变量的值为 led.o。
修改完成之后保存保存退出即可,那么此时我们的 2_led 目录下就已经有 3 个文件了,如
下所示:

zy@zy-virtual-machine:~/linux/drivers/2_led\$
zy@zy-virtual-machine:~/linux/drivers/2_led\$ ls
ledApp.c led.c Makefile
zy@zy-virtual-machine:~/linux/drivers/2_led\$

图 22.5.1 2_led 目录下的文件

输入如下命令编译驱动模块文件:

make

编译成功以后就会生成一个名为"led.ko"的驱动模块文件,如下所示:

```
zy@zy-virtual-machine:~/linux/drivers/2 led$
zy@zy-virtual-machine:~/linux/drivers/2 led$ make
make -C /home/zy/workspace/kernel-driver/linux-xlnx-xlnx rebase v5.4 2020.2 M=`p
wd` modules
make[1]: 进入目录"/home/zy/workspace/kernel-driver/linux-xlnx-xlnx rebase v5.4 2
020.2"
  CC [M] /home/zy/linux/drivers/2 led/led.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M] /home/zy/linux/drivers/2 led/led.mod.o
  LD [M] /home/zy/linux/drivers/2 led/led.ko
make[1]: 离开目录"/home/zy/workspace/kernel-driver/linux-xlnx-xlnx rebase v5.4 2
020.2"
zy@zy-virtual-machine:~/linux/drivers/2 led$ ls
ledApp.c led.ko led.mod.c led.o
                                       modules.order
          led.mod led.mod.o Makefile Module.symvers
led.c
zy@zy-virtual-machine:~/linux/drivers/2 led$
```

图 22.5.2 编译 led 驱动

2、编译测试 APP

输入如下命令编译测试 ledApp.c 这个测试程序:



原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php

\$CC ledApp.c -o ledApp

编译成功以后就会生成 ledApp 这个应用程序。

22.5.2 运行测试

将上一小节编译出来的 led.ko 和 ledApp 这两个文件拷贝到 NFS 共享目录下根文件系统的 /lib/modules/5.4.0-xilinx 文件夹下,如下图所示:

zy @;	zy-v	/irtı	Jal	mac	hine:-	-/works	spac	e/nf	s/ <mark>roo</mark> t	fs/lib/modules/5.4.0-xilinx\$
zy @:	zy-v	/irtu	Jal	-macl	hine:-	-/works	spac	e/nf	s/ <mark>roo</mark> t	fs/lib/modules/5.4.0-xilinx\$ ls -l
总用	量	44								
- rw	xrwx	(r-x	1	zy	zy	11856	6月	8	19:02	chrdevbaseApp
- rw	-rw-	r	1	zy	zy	6256	6月	8	19:02	chrdevbase.ko
- rw	xrwx	(r-x	1	zy	zy	11768	6月	9	13:57	/ ledApp 🔶
- rw	xrwx	(r-x	1	zy	zy	6620	6月	9	13:57	led.ko ←
- rw	-rw-	rw-	1	root	root	0	6月	9	13:58	8 modules.alias
- rw	-rw-	rw-	1	root	root	23	6月	9	13:58	8 modules.dep
- rw	-rw-	rw-	1	root	root	0	6月	9	13:58	modules.symbols
zy@:	zy-v	irtu	Jal	macl	nine:-	-/works	spac	e/nf	s/root	fs/lib/modules/5.4.0-xilinx\$

图 22.5.3 将驱动和测试程序复制到根文件系统中

启动开发板,进入"/lib/modules/5.4.0-xilinx"目录下,输入如下命令加载 led.ko 驱动模

块:

modprobe led.ko //加载驱动

驱动加载成功以后创建"/dev/led"设备节点,命令如下:

mknod /dev/led c 200 0

驱动节点创建成功以后就可以使用 ledApp 软件来测试驱动是否工作正常,输入如下命令 点亮底板上的 PS_LED0 小灯:

./ledApp /dev/led 1 //点亮 LED 灯

输入上述命令以后查看底板上的 PS_LED0 小灯是否点亮,如果点亮的话说明驱动工作正常。在输入如下命令关闭 LED 灯:

./ledApp /dev/led 0 //关闭 LED 灯

输入上述命令以后查看底板上的 PS_LED0 小灯是否熄灭,如果熄灭的话说明我们编写的 LED 驱动工作完全正常!至此,我们成功编写了第一个真正的 Linux 驱动设备程序。

如果要卸载驱动的话输入如下命令即可:

```
rmmod led.ko
```



原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php

第二十三章 新字符设备驱动实验

经过前两章实验的实战操作,我们已经掌握了 Linux 字符设备驱动开发的基本步骤,字 符设备驱动开发重点是使用 register_chrdev函数注册字符设备,当不再使用设备的时候就使用 unregister_chrdev函数注销字符设备,驱动模块加载成功以后还需要手动使用 mknod 命令创建 设备节点。register_chrdev和 unregister_chrdev这两个函数是老版本驱动使用的函数,现在新 的字符设备驱动已经不再使用这两个函数,而是使用 Linux 内核推荐的新字符设备驱动 API 函 数。本节我们就来学习一下如何编写新字符设备驱动,并且在驱动模块加载的时候自动创建 设备节点文件。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

23.1 新字符设备驱动原理

23.1.1 分配和释放设备号

使用 register_chrdev 函数注册字符设备的时候只需要给定一个主设备号即可,但是这样会 带来两个问题:

①、需要我们事先确定好哪些主设备号没有使用。

②、会将一个主设备号下的所有次设备号都使用掉,比如现在设置 LED 这个主设备号为 200, 那么 0~1048575(2^20-1)这个区间的次设备号就全部都被 LED 一个设备分走了。这样太 浪费次设备号了! 一个 LED 设备肯定只能有一个主设备号, 一个次设备号。

解决这两个问题最好的方法就是要使用设备号的时候向 Linux 内核申请,需要几个就申 请几个,由Linux内核分配设备可以使用的设备号。这个就是我们在21.3.2小节讲解的设备号 的分配,如果没有指定设备号的话就使用如下函数来申请设备号:

int alloc_chrdev_region(dev_t *dev, unsigned baseminor, unsigned count, const char *name)

如果给定了设备的主设备号和次设备号就使用如下所示函数来注册设备号即可:

int register chrdev region(dev t from, unsigned count, const char *name)

参数 from 是要申请的起始设备号,也就是给定的设备号;参数 count 是要申请的数量, 一般都是一个;参数 name 是设备名字。

注销字符设备之后要释放掉设备号,不管是通过 alloc_chrdev_region 函数还是 register_chrdev_region 函数申请的设备号,统一使用如下释放函数:

void unregister_chrdev_region(dev_t from, unsigned count)

新字符设备驱动下,设备号分配示例代码如下:

示例代码 23.1.1 新字符设备驱动下设备号分配

1 int major; /* 主设备号	± */
2 int minor; /* 次设备号	± */
3 dev_t devid; /* 设备号 *	:/
4	
5 if (major) {	/* 定义了主设备号 */
6 devid = MKDEV(major, 0);	/* 大部分驱动次设备号都选择 0 */
7 register_chrdev_region(devid	, 1, "test") ;
8 } else {	/* 没有定义设备号 */
9 alloc_chrdev_region(&devid, 0	,1, "test"); /* 申请设备号 */
10 major = MAJOR(devid);	/* 获取分配号的主设备号 */
11 minor = MINOR(devid);	/* 获取分配号的次设备号 */
12 }	

第1~3行,定义了主/次设备号变量 major 和 minor,以及设备号变量 devid。

第5行,判断主设备号 major 是否有效,在 Linux 驱动中一般给出主设备号的话就表示这 个设备的设备号已经确定了,因为次设备号基本上都选择0,这算个Linux驱动开发中约定俗 成的一种规定了。

第6行,如果 major 有效的话就使用 MKDEV 来构建设备号,次设备号选择 0。

第7行,使用 register_chrdev_region 函数来注册设备号。



原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php

第 9~11 行,如果 major 无效,那就表示没有给定设备号。此时就要使用 alloc_chrdev_region 函数来申请设备号。设备号申请成功以后使用 MAJOR 和 MINOR 来提取 出主设备号和次设备号,当然了,第10和11行提取主设备号和次设备号的代码可以不要。 如果要注销设备号的话,使用如下代码即可:

/* 注销设备号 */

示例代码 23.1.2 注销设备号

1 unregister_chrdev_region(devid, 1);

注销设备号的代码很简单。

23.1.2 新的字符设备注册方法

1、字符设备结构

在 Linux 中使用 cdev 结构体表示一个字符设备, cdev 结构体在 include/linux/cdev.h 文件 中的定义如下:

示例代码 23.1.3 cdev 结构体

1 struct cdev {

- 2 struct kobject kobj;
- 3 struct module *owner;
- 4 const struct file_operations *ops;
- 5 struct list_head list;
- 6 dev_t dev;
- 7 unsigned int count;

8};

在 cdev 中有两个重要的成员变量: ops 和 dev,这两个就是字符设备文件操作函数集合 file_operations 以及设备号 dev_t。编写字符设备驱动之前需要定义一个 cdev 结构体变量,这 个变量就表示一个字符设备,如下所示:

struct cdev test_cdev;

2、cdev_init 函数

定义好 cdev 变量以后就要使用 cdev_init 函数对其进行初始化, cdev_init 函数原型如下: void cdev_init(struct cdev *cdev, const struct file_operations *fops)

参数 cdev 就是要初始化的 cdev 结构体变量,参数 fops 就是字符设备文件操作函数集合。 使用 cdev_init 函数初始化 cdev 变量的示例代码如下:

```
示例代码 23.1.4 cdev_init 函数使用示例代码
```

```
1 struct cdev testcdev;
2
3 /* 设备操作函数 */
4 static struct file_operations test_fops = {
5 .owner = THIS_MODULE,
6 /* 其他具体的初始项 */
7 };
8
9 testcdev.owner = THIS_MODULE;
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php 10 cdev_init(&testcdev, &test_fops); /* 初始化 cdev 结构体变量 */

3、cdev_add 函数

cdev_add函数用于向Linux系统添加字符设备(cdev结构体变量),首先使用 cdev_init函数 完成对 cdev结构体变量的初始化,然后使用 cdev_add函数向Linux系统添加这个字符设备。cdev_add函数原型如下:

int cdev_add(struct cdev *p, dev_t dev, unsigned count)

参数 p 指向要添加的字符设备(cdev 结构体变量),参数 dev 就是设备所使用的设备号,参数 count 是要添加的设备数量。完善示例代码 23.1,加入 cdev_add 函数,内容如下所示:

示例代码 23.1.5 cdev_add 函数使用示例

```
1 struct cdev testcdev;
```

```
2
```

```
3 /* 设备操作函数 */
```

```
4 static struct file_operations test_fops = {
```

```
5 .owner = THIS_MODULE,
```

```
6 /* 其他具体的初始项 */
```

7 };

```
8
```

9 testcdev.owner = THIS_MODULE;

10 cdev_init(&testcdev, &test_fops); /* 初始化 cdev 结构体变量 */

```
11 cdev_add(&testcdev, devid, 1); /* 添加字符设备 */
```

示例代码 23.1.5 就是新的注册字符设备代码段, Linux 内核中大量的字符设备驱动都是采 用这种方法向 Linux 内核添加字符设备。如果在加上示例代码 23.1.1 中分配设备号的程序, 那 么就它们一起实现的就是函数 register_chrdev 的功能。

3、cdev_del函数

卸载驱动的时候一定要使用 cdev_del 函数从 Linux 内核中删除相应的字符设备, cdev_del 函数原型如下:

void cdev_del(struct cdev *p)

参数 p 就是要删除的字符设备。如果要删除字符设备,参考如下代码:

示例代码 23.1.6 cdev_del 函数使用示例

1 cdev_del(&testcdev); /* 删除 cdev */

cdev_del 和 unregister_chrdev_region 这两个函数合起来的功能相当于 unregister_chrdev 函数。

23.2 自动创建设备节点

在前面的 Linux 驱动实验中,当我们使用 modprobe 加载驱动程序以后还需要使用命令"mknod"手动创建设备节点。本节就来讲解一下如何实现自动创建设备节点,在驱动中实现自动创建设备节点的功能以后,使用 modprobe 加载驱动模块成功的话就会自动在/dev 目录下创建对应的设备文件。

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

正点原子

23.2.1 mdev 机制

udev 是一个用户程序,在 Linux 下通过 udev 来实现设备文件的创建与删除, udev 可以检 测系统中硬件设备状态,可以根据系统中硬件设备状态来创建或者删除设备文件。驱动注册 和注销时信息会被传给 udev, 由 udev 在应用层进行设备文件的创建和删除, 比如使用 modprobe 命令成功加载驱动模块以后就自动在/dev 目录下创建对应的设备节点文件,使用 rmmod命令卸载驱动模块以后就删除掉/dev目录下的设备节点文件。使用 busybox 构建根文件 系统的时候, busybox 会创建一个 udev 的简化版本——mdev, 所以在嵌入式 Linux 中我们使 用 mdev 来实现设备节点文件的自动创建与删除, Linux 系统中的热插拔事件也由 mdev 管理。

关于 udev 或 mdev 更加详细的工作原理我们不去讨论,本章我们重点来学习设备文件节 点的自动创建与删除。

23.2.2 创建和删除类

自动创建设备节点的工作是在驱动程序的入口函数中完成的,一般在 cdev_add 函数后面 添加自动创建设备节点相关代码。首先要创建一个 class 类(class 的概念这里暂时不去细说, 大家可以先简答地理解为设备类即可,就是某个设备属于某个类,后面我们会详细的讲), class 是个结构体, 定义在文件 include/linux/device.h 里面。class create 是类创建函数, class_create 是个宏定义,内容如下:

示例代码 23.2.1 class create 函数

1 #define class_create(owner, name) \ 2 ({ 3 static struct lock_class_key __key; \ 4 __class_create(owner, name, &_key); \ **5**}) 6 7 struct class *__class_create(struct module *owner, const char *name, struct lock_class_key *key) 8 根据上述代码,将宏 class create 展开以后内容如下: struct class *class_create (struct module *owner, const char *name) class_create 一共有两个参数,参数 owner 一般为 THIS_MODULE,参数 name 是类名字。

返回值是个指向结构体 class 的指针,也就是创建的类。

卸载驱动程序的时候需要删除掉类,类删除函数为 class destroy,函数原型如下:

void class destroy(struct class *cls);

参数 cls 就是要删除的类。

23.2.3 创建和删除设备

上一小节创建好类以后还不能实现自动创建设备节点,我们还需要在这个类下创建一个 设备(device)。设备是个结构体,定义在文件 include/linux/device.h 里面。使用 device create 函数在类下面创建设备, device_create 函数原型如下:

struct device *device_create(struct class *class, struct device *parent, dev_t devt, void *drvdata, const char *fmt, ...)



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

device_create 是个可变参数函数,参数 class 就是设备要创建哪个类下面;参数 parent 是 父设备,一般为 NULL,也就是没有父设备;参数 devt 是设备号;参数 drvdata 是设备可能会 使用的一些数据,一般为 NULL;参数 fmt 是设备名字,如果设置 fmt=xxx 的话,就会生成 /dev/xxx 这个设备文件。返回值就是创建好的设备。

同样的,卸载驱动的时候需要删除掉创建的设备,设备删除函数为 device_destroy,函数 原型如下:

void device_destroy(struct class *class, dev_t devt)

参数 classs 是要删除的设备所处的类,参数 devt 是要删除的设备号。

23.2.4 参考示例

在驱动入口函数里面创建类和设备,在驱动出口函数里面删除类和设备,参考示例如下: 示例代码 23.2.2 创建/删除类/设备参考代码

```
1 struct class *class;
                          /* 类 */
2 struct device *device; /* 设备 */
                         /* 设备号 */
3 dev_t devid;
4
5 /* 驱动入口函数 */
6 static int __init xxx_init(void)
7 {
  /* 创建类
                 */
8
9
    class = class_create(THIS_MODULE, "xxx");
10 /* 创建设备 */
    device = device_create(class, NULL, devid, NULL, "xxx");
11
12
    return 0;
13
14
15 /* 驱动出口函数 */
16 static void __exit led_exit(void)
17 {
18 /* 删除设备 */
     device_destroy(newchrled.class, newchrled.devid);
19
    /* 删除类 */
20
21
     class_destroy(newchrled.class);
22 }
23
24 module init(led init);
25 module_exit(led_exit);
```

23.3 设置设备文件私有数据

每个硬件设备都有一些属性,比如主设备号(dev_t),类(class)、设备(device)、开关状态 (state)等等,在编写驱动的时候你可以将这些属性全部写成变量的形式,如下所示:


R T	丁可任线驭子: WWW	yuanzige.com	化坛	:www.openeav	.com/iorum.pnp		
		示例代码	23. 2. 3 💈	变量形式的设备属	禹性		
	dev_t devid;	/* 设备号 */					
	struct cdev cdev;	/* cdev */					
	struct class *class;	/* 类 */					
	<pre>struct device *device;</pre>	/* 设备 */					
	int major;	/* 主设备号 */					
	int minor;	/* 次设备号 */					
	这样它告宁恐方间	師 但見读择它7	てキオレエ	对王一个语文	的底方屋灶信自	自我们是抗肉	Ħ

这样写肯定没有问题,但是这样写不专业!对于一个设备的所有属性信息我们最好将其做成一个结构体。编写驱动 open 函数的时候将设备结构体作为私有数据添加到设备文件中,如下所示:

示例代码 23.2.4 设备结构体作为私有数据

```
/* 设备结构体 */
1 struct test_dev{
     dev_t devid;
                             /* 设备号 */
2
3
     struct cdev cdev;
                         /* cdev */
    struct class *class;
                             /* 类 */
4
     struct device *device; /* 设备 */
5
6
     int major;
                            /* 主设备号 */
                             /* 次设备号 */
7
     int minor;
8 };
9
```

```
10 struct test_dev testdev;
```

```
11
```

大化共

12 /* open 函数 */ 13 static int test_open(struct inode *inode, struct file *filp)

```
14 {
```

15 filp->private_data = &testdev; /* 设置私有数据 */

```
16 return 0;
```

17 }

在 open 函数里面设置好私有数据以后,在 write、read、close 等函数中直接读取 private_data 即可得到设备结构体。

23.4 硬件原理图分析

本章实验硬件原理图参考 22.3 小节即可。

23.5 实验程序编写

本实验对应的例程路径为:开发板光盘资料(A盘)\4_SourceCode\3_Embedded_Linux\Linux驱动例程\3_newchrled。

本章实验在上一章实验的基础上完成,重点是使用了新的字符设备注册方法、设置了文件私有数据、添加了自动创建设备节点相关内容。



正点原子



```
原子哥在线教学: www.yuanzige.com
                                        论坛:www.openedv.com/forum.php
    38 /* 映射后的寄存器虚拟地址指针 */
    39 static void iomem *data addr;
   40 static void __iomem *dirm_addr;
   41 static void __iomem *outen_addr;
    42 static void __iomem *intdis_addr;
   43 static void iomem *aper clk ctrl addr;
    44
   45 /* newchrled 设备结构体 */
   46 struct newchrled dev {
   47 dev t devid;
                         /* 设备号 */
                        /* cdev */
    48 struct cdev cdev;
                         /* 类 */
    49 struct class *class;
    50 struct device *device; /* 设备 */
    51 int major;
                             /* 主设备号 */
                        /* 次设备号 */
    52 int minor;
    53 };
    54
    55 static struct newchrled_dev newchrled; /* led 设备 */
    56
    57 /*
    58 * @description
                         :打开设备
    59 * @param – inode
                            :传递给驱动的 inode
                             :设备文件, file 结构体有个叫做 private_data 的成员变量
    60 * @param – filp
    61 *
                              一般在 open 的时候将 private_data 指向设备结构体。
    62 * @return
                         :0 成功;其他 失败
    63 */
    64 static int led_open(struct inode *inode, struct file *filp)
    65 {
    66 filp->private_data = &newchrled; /* 设置私有数据 */
    67
        return 0;
    68 }
    69
    70 /*
    71 * @description
                        :从设备读取数据
   72 * @param – filp
                             :要打开的设备文件(文件描述符)
   73 * @param – buf
                             :返回给用户空间的数据缓冲区
   74 * @param – cnt
                             :要读取的数据长度
                             :相对于文件首地址的偏移
   75 * @param – offt
                     :读取的字节数,如果为负值,表示读取失败
    76 * @return
    77 */
```

```
78 static ssize_t led_read(struct file *filp, char __user *buf,
```



```
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
           size_t cnt, loff_t *offt)
    79
    80 {
    81 return 0;
    82 }
    83
    84 /*
    85 * @description
                     : 向设备写数据
   86 * @param – filp
                             :设备文件,表示打开的文件描述符
    87 * @param – buf
                             :要写给设备写入的数据
    88 * @param – cnt
                            :要写入的数据长度
    89 * @param – offt
                             :相对于文件首地址的偏移
   90 * @return : 写入的字节数,如果为负值,表示写入失败
    91 */
    92 static ssize_t led_write(struct file *filp, const char __user *buf,
           size_t cnt, loff_t *offt)
    93
    94 {
    95 int ret;
    96 int val;
    97
       char kern_buf[1];
    98
        ret = copy_from_user(kern_buf, buf, cnt); // 得到应用层传递过来的数据
    99
   100 if(0 > ret) {
          printk(KERN_ERR "kernel write failed!\r\n");
   101
   102
          return -EFAULT;
   103
        }
   104
   105 val = readl(data_addr);
   106 if (0 == kern_buf[0])
         val &= ~(0x1U << 7); // 如果传递过来的数据是 0 则关闭 led
   107
   108
        else if (1 == kern_buf[0])
          val |= (0x1U << 7); // 如果传递过来的数据是 1 则点亮 led
   109
   110
   111 writel(val, data_addr);
   112 return 0;
   113 }
   114
   115 /*
                     :关闭/释放设备
   116 * @description
   117 * @param – filp
                            :要关闭的设备文件(文件描述符)
                     :0 成功;其他 失败
   118 * @return
   119 */
```



```
原子哥在线教学: www.yuanzige.com
                                              论坛:www.openedv.com/forum.php
    120 static int led_release(struct inode *inode, struct file *filp)
    121 {
    122
         return 0;
    123 }
    124
    125 static inline void led ioremap(void)
    126 {
    127
         data_addr = ioremap(ZYNQ_GPIO_REG_BASE + DATA_OFFSET, 4);
    128
         dirm_addr = ioremap(ZYNQ_GPIO_REG_BASE + DIRM_OFFSET, 4);
         outen_addr = ioremap(ZYNQ_GPIO_REG_BASE + OUTEN_OFFSET, 4);
    129
    130
         intdis_addr = ioremap(ZYNQ_GPIO_REG_BASE + INTDIS_OFFSET, 4);
    131
         aper_clk_ctrl_addr = ioremap(APER_CLK_CTRL, 4);
    132 }
    133
    134 static inline void led_iounmap(void)
    135 {
    136
         iounmap(data_addr);
    137
         iounmap(dirm_addr);
    138
         iounmap(outen_addr);
    139
         iounmap(intdis_addr);
    140
         iounmap(aper_clk_ctrl_addr);
    141 }
    142
    143 /* 设备操作函数 */
    144 static struct file_operations newchrled_fops = {
    145 .owner = THIS_MODULE,
    146 .open = led_open,
    147 .read = led_read,
    148 .write = led_write,
    149
         .release = led_release,
    150 };
    151
    152 static int __init led_init(void)
    153 {
         u32 val;
    154
    155
         int ret;
    156
         /* 1.寄存器地址映射 */
    157
    158
         led_ioremap();
    159
    160 /* 2.使能 GPIO 时钟 */
```



```
原子哥在线教学: www.yuanzige.com
                                              论坛:www.openedv.com/forum.php
    161
          val = readl(aper_clk_ctrl_addr);
    162
          val |= (0x1U << 22);
    163
          writel(val, aper_clk_ctrl_addr);
    164
    165
         /* 3.关闭中断功能 */
    166
         val = (0x1U << 7);
    167
          writel(val, intdis_addr);
    168
    169
         /* 4.设置 GPIO 为输出功能 */
    170
         val = readl(dirm addr);
    171
          val = (0x1U << 7);
    172
          writel(val, dirm_addr);
    173
    174
         /* 5.使能 GPIO 输出功能 */
    175
         val = readl(outen_addr);
    176
         val |= (0x1U << 7);
    177
          writel(val, outen_addr);
    178
    179
         /* 6.默认关闭 LED */
    180
         val = readl(data_addr);
    181
          val &= ~(0x1U << 7);
    182
          writel(val, data_addr);
    183
    184
          /* 7.注册字符设备驱动 */
    185
          /* 创建设备号 */
    186
          if (newchrled.major) {
    187
            newchrled.devid = MKDEV(newchrled.major, 0);
            ret = register_chrdev_region(newchrled.devid, NEWCHRLED_CNT, NEWCHRLED_NAME);
    188
    189
            if (ret)
    190
              goto out1;
    191
          } else {
    192
            ret = alloc_chrdev_region(&newchrled.devid, 0, NEWCHRLED_CNT, NEWCHRLED_NAME);
    193
            if (ret)
    194
              goto out1;
    195
    196
            newchrled.major = MAJOR(newchrled.devid);
    197
            newchrled.minor = MINOR(newchrled.devid);
    198
          }
    199
    200
          printk("newcheled major=%d,minor=%d\r\n",newchrled.major, newchrled.minor);
    201
```



```
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
         /* 初始化 cdev */
    202
    203
         newchrled.cdev.owner = THIS_MODULE;
         cdev_init(&newchrled.cdev, &newchrled_fops);
    204
    205
    206
         /* 添加一个 cdev */
    207
         ret = cdev add(&newchrled.cdev, newchrled.devid, NEWCHRLED CNT);
    208
         if (ret)
    209
           goto out2;
    210
    211
          /* 创建类 */
    212
         newchrled.class = class_create(THIS_MODULE, NEWCHRLED_NAME);
    213
         if (IS_ERR(newchrled.class)) {
    214
           ret = PTR_ERR(newchrled.class);
    215
           goto out3;
    216 }
    217
    218
         /* 创建设备 */
    219
         newchrled.device = device_create(newchrled.class, NULL,
    220
                newchrled.devid, NULL, NEWCHRLED_NAME);
    221
         if (IS_ERR(newchrled.device)) {
    222
           ret = PTR_ERR(newchrled.device);
    223
           goto out4;
    224
         }
    225
    226
         return 0;
    227
    228 out4:
    229
         class_destroy(newchrled.class);
    230
    231 out3:
    232
         cdev_del(&newchrled.cdev);
    233
    234 out2:
    235
         unregister_chrdev_region(newchrled.devid, NEWCHRLED_CNT);
    236
    237 out1:
    238
         led_iounmap();
    239
    240
         return ret;
    241 }
    242
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php 243 static void __exit led_exit(void) 244 { 245 /* 注销设备 */ 246 device_destroy(newchrled.class, newchrled.devid); 247 /* 注销类 */ 248 249 class_destroy(newchrled.class); 250 /* 删除 cdev */ 251 cdev del(&newchrled.cdev); 252 253 254 /* 注销设备号 */ 255 unregister_chrdev_region(newchrled.devid, NEWCHRLED_CNT); 256 257 /* 取消地址映射 */ 258 led_iounmap(); 259 } 260 261 /* 驱动模块入口和出口函数注册 */ 262 module_init(led_init); 263 module exit(led exit); 264 265 MODULE_AUTHOR("DengTao <773904075@qq.com>"); 266 MODULE_DESCRIPTION("Alientek ZYNQ GPIO LED Driver"); 267 MODULE LICENSE("GPL");

第 25 行,宏 NEWCHRLED_CNT 表示设备数量,在申请设备号或者向 Linux 内核添加字 符设备的时候需要设置设备数量,一般我们一个驱动一个设备,所以这个宏为1。

第 26 行,宏 NEWCHRLED_NAME 表示设备名字,本实验的设备名为"newchrdev",为了方便管理,所有使用到设备名字的地方统一使用此宏,当驱动加载成功以后就生成/dev/newchrled 这个设备文件。

第46~53行,创建设备结构体 newchrled_dev。

第55行,定义一个设备结构体变量 newchrdev,此变量表示 led 设备。

第 64~68 行,在 led_open 函数中设置文件的私有数据 private_data 指向 newchrdev。

第 152~241 行,根据前面讲解的方法在驱动入口函数 led_init 中申请设备号、添加字符设备、创建类和设备。186~198 行代码去申请设备号,如果提供了设备号则申请静态设备号,如果我们提供的设备号为0则采用动态申请设备号的方法,第 200 行使用 printk 在终端上显示出申请到的主设备号和次设备号。

第 241~245 行,根据前面讲解的方法,在驱动出口函数 led_exit 中注销字符新设备、删除 类和设备。

总体来说 newchrled.c 文件中的内容不复杂,LED 灯驱动部分的程序和上一章一样。重点 就是使用了新的字符设备驱动方法。

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

正点原子

驱动中的倒退式处理方法

细心的同学会发现在上面的代码当中用到了 C 语言中 goto 语句,不知道大家对这个 goto 熟悉不熟悉, goto 顾名思义其实即使跳转的意思, 例如 goto out1, 就是跳转到 out1 地址所在 的地方,那么 goto 语句在 linux 内核当中用的特别多,因为它非常符合 linux 下这种软件设计 上的要求,因为内核中一个函数可能包含了很多个操作,这些操作每一步都有可能出错,如 果出错之后那么后面的步骤就没有进行下去的必要性了,但是你不能直接退出,你得把你前 面做过的操作给"复原"了。

例如在上面的代码当中,如果在调用 device_create 函数注册设备的时候失败了,没有成 功,那么你就得把前面做的工作给"复原",你创建了 class 类,那你就得调用 class_destroy 删除这个类,你添加了 cdev,那你就得删除 cdev,而且他这个顺序还是倒退式的,大家需要 去好好理解下,以后自己开发驱动也需要养成这样的习惯。

23.5.2 编写测试 App

本章直接使用上一章的测试 App,将上一章的 ledApp.c 文件复制到本章实验目录下 (3 newchrled) 即可。

23.6 运行测试

23.6.1 编译驱动程序和测试 App

1、编译驱动程序

将上一章使用的 Makefile 文件拷贝到本实验的目录下,然后打开这个 Makefile 文件,将 obj-m 变量的值改为 newchrled.o, 最终 Makefile 内容如下所示:

示例代码 23.6.1 Makefile 文件内容

```
1 KERN_DIR := /home/zy/workspace/kernel-driver/linux-xlnx-xlnx_rebase_v5.4_2020.2
2
3 obj-m := newchrled.o
4
5 all:
   make -C $(KERN_DIR) M=`pwd` modules
6
7
8 clean:
9
   make -C $(KERN_DIR) M=`pwd` clean
第3行,设置 obj-m 变量的值为 newchrled.o。
执行 make 命令编译出驱动模块文件:
make
编译成功以后就会生成一个名为"newchrled.ko"的驱动模块文件,如下所示:
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php zy@zy-virtual-machine:~/linux/drivers/3 newchrled\$ ls ledApp.c Makefile newchrled.c zy@zy-virtual-machine:~/linux/drivers/3 newchrled\$ make make -C /home/zy/workspace/kernel-driver/linux-xlnx-xlnx rebase v5.4 2020.2 M=`p wd` modules make[1]: 进入目录"/home/zy/workspace/kernel-driver/linux-xlnx-xlnx rebase v5.4 2 020.2" CC [M] /home/zy/linux/drivers/3 newchrled/newchrled.o Building modules, stage 2. MODPOST 1 modules CC [M] /home/zy/linux/drivers/3 newchrled/newchrled.mod.o LD [M] /home/zy/linux/drivers/3 newchrled/newchrled.ko make[1]: 离开目录"/home/zy/workspace/kernel-driver/linux-xlnx-xlnx rebase v5.4 2 020.2" zy@zy-virtual-machine:~/linux/drivers/3 newchrled\$ ls ledApp.c modules.order <u>newchrled.c</u> newchrled.mod Makefile Module.symvers newchrled.ko newchrled.mod.c newchrled.mod.o newchrled.o zy@zy-virtual-machine:~/linux/drivers/3 newchrled\$ 图 23.6.1 编译过程

2、编译测试 APP

输入如下命令编译测试 ledApp.c 这个测试程序:

\$CC ledApp.c -o ledApp

编译成功以后就会生成 ledApp 这个应用程序,如下图所示:

```
zy@zy-virtual-machine:~/linux/drivers/3 newchrled$ ls
ledApp.c modules.order
                         newchrled.c
                                       newchrled.mod
                                                        newchrled.mod.o
Makefile Module.symvers
                         newchrled.ko
                                       newchrled.mod.c
                                                        newchrled.o
zy@zy-virtual-machine:~/linux/drivers/3 newchrled$ $CC ledApp.c -o ledApp
zy@zy-virtual-machine:~/linux/drivers/3 newchrled$ ls
                         newchrled.ko
         modules.order
                                          newchrled.mod.o
ledApp
ledApp.c Module.symvers
                         newchrled.mod
                                          newchrled.o
                         newchrled.mod.c
Makefile newchrled.c
zy@zy-virtual-machine:~/linux/drivers/3 newchrled$
```

图 23.6.2 编译测试 App

23.6.2 运行测试

将上一小节编译出来的 newchrled.ko 和 ledApp 两个文件拷贝到 NFS 共享目录下根文件系 统的/lib/modules/5.4.0-xilinx 文件夹中。

将 SD 启动卡插到开发板中,连接开发板串口、网口和电源线,启动开发板,通过串口终端进入"/lib/modules/5.4.0-xilinx"目录下,输入如下命令加载 newchrled.ko 驱动模块:

depmod //第一次加载驱动的时候需要运行此命令

modprobe newchrled.ko //加载驱动

驱动加载成功以后会输出申请到的主设备号和次设备号,如下图所示:



正点原子

图 23.6.3 加载驱动模块

从上图可以看出,申请到的主设备号为 244,次设备号为 0。驱动加载成功以后会自动在 /dev 目录下创建设备节点文件/dev/newchrdev,输入如下命令查看/dev/newchrdev 这个设备节 点文件是否存在:

ls /dev/newchrled -l

结果如下图所示:

root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx# root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx# ls /dev/newchrled -l crw------ 1 root root 244, 0 Jun 7 03:04 /dev/newchrled root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx# root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx#

图 23.6.4 自动创建设备节点

从图中可以看出,/dev/newchrled 这个设备文件存在,而且主设备号为 244,此设备号为 0,说明设备节点文件创建成功。

驱动节点创建成功以后就可以使用 ledApp 软件来测试驱动是否工作正常,输入如下命令 打开 LED 灯:

./ledApp /dev/newchrled 1 //打开 LED 灯

输入上述命令以后查看底板上的 PS_LED0 灯是否点亮,如果点亮的话说明驱动工作正常。 在输入如下命令关闭 LED 灯:

./ledApp /dev/newchrled 0 //关闭 LED 灯

输入上述命令以后查看底板上的 PS_LED0 灯是否熄灭。如果要卸载驱动的话输入如下命令即可:

rmmod newchrled.ko



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

第二十四章 Linux 设备树

前面章节中我们多次提到"设备树"这个概念,因为时机未到,所以当时并没有详细的 讲解什么是"设备树",本章我们就来详细的谈一谈设备树。掌握设备树是 Linux 驱动开发 人员必备的技能!因为在新版本的 Linux 内核中,设备驱动基本全部采用了设备树的方式(也 有支持老式驱动的,比较少),最新出的CPU其驱动开发也基本都是基于设备树的,我们所使 用的 Linux 版本为 5.4.0, 肯定是支持设备树的, 所以正点原子领航者开发板的所有 Linux 驱 动都是基于设备树的。本章我们就来了解一下设备树的起源、重点学习一下设备树语法。



原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php

24.1 什么是设备树

在旧版本(大概是 3.x 以前的版本)的 linux 内核当中,ARM 架构的板级硬件设备信息被 硬编码在 arch/arm/plat-xxx 和 arch/arm/mach-xxx 目录下的文件当中,例如板子上的 platform 设 备信息、设备 I/O 资源 resource、板子上的 i2c 设备的描述信息信息 i2c_board_info、板子上 spi 设备的描述信息 spi_board_info 以及各种硬件设备的 platform_data 等,所以就导致在 Linux 内 核源码中大量的 arch/arm/mach-xxx 和 arch/arm/plat-xxx 文件夹,这些文件夹里面的文件就描述了对应平台下的板级硬件设备信息。比如在 arch/arm/mach-s3c24xx/mach-smdk2440.c 文件中 有如下内容(有缩减):

```
示例代码 24.1.1 mach-smdk2440.c 文件代码片段
90 static struct s3c2410fb_display smdk2440_lcd_cfg __initdata = {
91
92
     .lcdcon5 = S3C2410 LCDCON5 FRM565
93
          S3C2410_LCDCON5_INVVLINE
94
          S3C2410_LCDCON5_INVVFRAME
          S3C2410_LCDCON5_PWREN
95
96
          S3C2410 LCDCON5 HWSWP,
.....
113 };
114
115 static struct s3c2410fb_mach_info smdk2440_fb_info __initdata = {
116
     .displays = &smdk2440_lcd_cfg,
117
     .num_displays = 1,
     .default_display = 0,
118
•••••
133 };
134
135 static struct platform_device *smdk2440_devices[] __initdata = {
```

```
136 &s3c_device_ohci,
```

```
137 &s3c_device_lcd,
```

```
138 &s3c_device_wdt,
```

```
139 &s3c_device_i2c0,
```

```
140 &s3c_device_iis,
```

```
141 };
```

上述代码中的结构体变量 smdk2440_fb_info 就是描述 SMDK2440 这个开发板上的 LCD 硬件信息的,结构体指针数组 smdk2440_devices 描述的是 SMDK2440 这个开发板上的所有硬件相关信息。这个仅仅是使用 2440 这个芯片的 SMDK2440 开发板下的 LCD 信息, SMDK2440 开发板还有很多的其他外设硬件和平台硬件信息。使用 2440 这个芯片的板子有很多,每个板子都有描述相应板级硬件信息的文件,这仅仅只是一个 2440。随着智能手机的发展,每年新出的 ARM 架构芯片少说都在数十、数百款, Linux 内核下板级信息文件将会成指数级增长!这些板级信息文件都是.c 或.h 文件,都会被硬编码进 Linux 内核中,导致 Linux 内核"虚胖"。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

就好比你喜欢吃自助餐,然后花了 100 多到一家宣传看着很不错的自助餐厅,结果你想吃的 牛排、海鲜、烤肉基本没多少,全都是一些凉菜、炒面、西瓜、饮料等小吃,相信你此时肯 定会脱口而出一句"F*k!"、"骗子!"。

这些板级硬件信息代码对 linux 内核来说只不过是垃圾代码而已,所以当 Linux 之父 linus 看到 ARM 社区向 Linux 内核添加了大量"无用"、冗余的板级信息文件,不禁的发出了一句"This whole ARM thing is a f*cking pain in the ass"。从此以后 ARM 社区就开始引入设备树 DTS 了。

DTS 即 Device Tree Source 设备树源码, Device Tree 是一种描述硬件的数据结构,它起源于 OpenFirmware(OF),用于实现驱动代码与设备信息相分离;在设备树出现以前,所有关于板 子上硬件设备的具体都要硬编码在 arch/arm/plat-xxx 和 arch/arm/mach-xxx 目录下的文件当中,或者直接硬编码在驱动代码当中,例如我们前面编写的 LED 驱动就是直接将 led 的信息(用 的哪个管脚、GPIO 寄存器的基地址等)直接编码在了驱动源码当中,一旦外围设备变化(例 如 PS_LED0 换成另一个 MIO 引脚了),驱动代码就要重写。

引入了设备树之后,驱动代码只负责处理驱动的逻辑,而关于设备的具体信息存放到设 备树文件中,这样,如果只是硬件接口信息的变化而没有驱动逻辑的变化,驱动开发者只需 要修改设备树文件信息,不需要改写驱动代码。使用设备树之后,许多硬件设备信息可以直 接通过它传递给 Linux,而不需要在内核中堆积大量的冗余代码。

设备树,将这个词分开就是"设备"和"树",描述设备树的文件叫做 DTS(Device Tree Source),这个 DTS 文件采用树形结构描述板级设备,也就是开发板上的硬件设备信息,比如 CPU 数量、内存基地址、IIC 接口上接了哪些设备、SPI 接口上接了哪些设备等等,如图 24.1.1 所示:



图 24.1.1 设备树结构示意图

在图 24.1.1 中,树的主干就是系统总线,IIC 控制器、GPIO 控制器、SPI 控制器等都是接到系统主线上的分支。IIC 控制器有分为 IIC1 和 IIC2 两种,其中 IIC1 上接了 FT5206 和 AT24C02 这两个 IIC 设备,IIC2 上只接了 MPU6050 这个设备。DTS 文件的主要功能就是按照



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

图 24.1.1 所示的结构来描述板子上的设备信息,DTS 文件描述设备信息是有相应的语法规则 要求的,稍后我们会详细的讲解 DTS 语法规则。

设备树文件的扩展名为.dts,一个.dts(device tree source)文件对应一个开发板,一般放置在内核的"arch/arm/boot/dts/"目录下,比如 exynos4412 开发板的板级设备树文件就是"arch/arm/boot/dts/exynos4412-origen.dts",再比如I.MX6ULL-EVK开发板的板级设备树文件就是arch/arm/boot/dts/imx6ull-14x14-evk.dts。那本篇驱动开发我们所使用的板级设备树文件就是arch/arm/boot/dts/system-top.dts,这个文件是在第二十章时候使用 Petalinux 工具自动生成的,前面已经跟大家讲过了,除了 system-top.dts 文件之外,还生成了另外五个文件 pl.dtsi、pcw.dtsi、zynq-7000.dtsi(system-top.dts 包含它们三个,后面会说到), system-conf.dtsi, system-uer.dtsi并且一并把它们放入了 linux 内核源码 arch/arm/boot/dts 目录下了。

前面也跟大家讲过,除了内核支持设备树之外,新版的 u-boot 也是支持设备树的,如果 有机会也可以跟大家讲一讲 U-Boot 的设备树。

24.2 设备树的基本知识

24.2.1 dts

设备树的源文件的后缀名就是.dts,每一款硬件平台可以单独写一份 xxxx.dts,所以在 Linux 内核源码中存在大量.dts 文件,对于 arm 架构可以在 arch/arm/boot/dts 找到相应的 dts。

24.2.2 dtsi

值得一提的是,对于一些相同的 dts 配置可以抽象到 dtsi 文件中,这个 dtsi 文件其实就类 似于 C 语言当中的.h 头文件,可以通过 C 语言中使用 include 来包含一个.dtsi 文件,例如 arch/arm/boot/dts/system-top.dts 文件有如下内容:

示例代码 24.2.1 system-top.dts

```
1 /*
2 * CAUTION: This file is automatically generated by Xilinx.
3 * Version:
4 * Today is: Thu Jun 8 05:45:26 2023
5 */
6
7
8 /dts-v1/;
9 #include "zynq-7000.dtsi"
10 #include "pl.dtsi"
11 #include "pcw.dtsi"
12/{
13 chosen {
     bootargs = "earlycon";
14
15
     stdout-path = "serial0:115200n8";
16 };
17 aliases {
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

```
18 ethernet0 = \& \text{gem}0;
```

- 19 ethernet1 = & gem1;
- 20 i2c0 = &i2c2;
- 21 i2c1 = &i2c0;
- 22 i2c2 = &i2c1;
- 23 serial0 = &uart0;
- 24 serial1 = &uart1;
- 25 spi0 = &qspi;

```
26 };
```

- 27 memory {
- 28 device_type = "memory";
- 29 $reg = \langle 0x0 \ 0x40000000 \rangle;$
- 30 };
- 31 };
- 32 #include "system-user.dtsi"

第 9~11 行中,通过#include 包含了同目录下的三个.dtsi 文件,分别为: zynq-7000.dtsi、pl.dtsi、pcw.dtsi。这里简答地给大家说一下这三个文件的内容有啥不同,首先 zynq-7000.dtsi 文件中的内容是 zynq-7000 系列处理器相同的硬件外设配置信息(PS 端的),pl.dtsi 的内容是 我们在 vivado 当中添加的 pl 端外设对应的配置信息,而 pcw.dtsi 则表示我们在 vivado 当中已 经使能的 PS 外设,这两个文件都是由 petalinux 根据 xsa 硬件描述自动生成的,不需要我们去 编写。

那么除此之外,使用#include 除了可以包含.dtsi 文件之外,还可以包含.dts 文件以及 C 语言当中的.h 文件,这些都是可以的,可以这么理解.dtsi 和.dts 文件语法各方面都是一样的,但是不能直接编译一个.dtsi 文件。

24.2.3 dtc

dtc 其实就是 device-tree-compiler,也就是设备树文件.dts 的编译器。将.c 文件编译为.o 文件需要用到 gcc 编译器,那么将.dts 文件编译为相应的二进制文件则需要 dtc 编译器,dtc 工具在 Linux 内核的 scripts/dtc 目录下,当然必须要编译了内核源码之后才会生成,如下所示:

zy@zy-virtual-ma	<mark>chine:</mark> ~/workspace/l	kernel-driver/linux	x-xlnx-xlnx_rebase_v5.4_2020
.2/scripts/dtc\$	ls		
checks.c	dtc-parser.tab.c	fstree.c	treesource.c
checks.o	dtc-parser.tab.h	fstree.o	treesource.o
data.c	dtc-parser.tab.o	include-prefixes	update-dtc-source.sh
data.o	dtc-parser.y	libfdt	util.c
dtc	dt_to_config	livetree.c	util.h
dtc.c	dtx_diff	livetree.o	util.o
dtc.h	fdtdump.c	Makefile	version_gen.h
dtc-lexer.l	fdtget.c	Makefile.dtc	yamltree.c
dtc-lexer.lex.c	fdtput.c	srcpos.c	
dtc-lexer.lex.o	flattree.c	srcpos.h	
dtc.o	flattree.o	srcpos.o	
<pre>zy@zy-virtual-machine:~/workspace/kernel-driver/linux-xlnx-xlnx_rebase_v5.4_2020 .2/scripts/dtc\$</pre>			

图 24.2.1 dtc 编译器



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

我们来看看 scripts/dtc/Makefile 文件,如下所示:

示例代码 24.2.2 scripts/dtc/Makefile 文件代码段

1 hostprogs-\$(CONFIG_DTC) := dtc

 $2 \text{ always} \quad := (hostprogs-y)

3

4 dtc-objs:= dtc.o flattree.o fstree.o data.o livetree.o treesource.o \setminus

5 srcpos.o checks.o util.o

6 dtc-objs += dtc-lexer.lex.o dtc-parser.tab.o

•••••

可以看出,dtc 工具依赖于 dtc.c、flattree.c、fstree.c 等文件,最终编译并链接出 dtc 这个主机 文件。如果要编译 dts 文件的话只需要进入到 Linux 源码根目录下,然后执行如下命令:

make all

或者:

.....

make dtbs

"make all"命令是编译 Linux 源码中的所有东西,包括 zImage, ko 驱动模块以及设备 树,如果只是编译设备树的话建议使用"make dtbs"命令。

在内核源码 arch/arm/boot/dts 目录下有很多的 dts 文件,那我们编译的时候如何确定编译 的是哪个或者说哪些 dts 文件的呢?大家可以打开 arch/arm/boot/dts/Makefile 文件,找到 CONFIG_ARCH_ZYNQ 宏所在的位置,内容如下所示:

示例代码 24.2.3 arch/arm/boot/dts/Makefile 文件部分内容

```
1003 dtb-(CONFIG_ARCH_ZYNQ) +=
```

```
1004 zynq-cc108.dtb \setminus
```

```
1005 zynq-microzed.dtb \setminus
```

```
1006 zynq-parallella.dtb \setminus
```

```
1007 zynq-zc702.dtb \setminus
```

```
1008 zynq-zc706.dtb \setminus
```

```
1009 zynq-zc770-xm010.dtb \setminus
```

```
1010 zynq-zc770-xm011.dtb \setminus
```

```
1011 zynq-zc770-xm012.dtb \setminus
```

```
1012 zynq-zc770-xm013.dtb \setminus
```

```
1013 zynq-zed.dtb \setminus
```

```
1014 \qquad zynq-zybo.dtb \ \backslash
```

```
1015 system-top.dtb
```

```
.....
```

由上面可以知道,当 CONFIG_ARCH_ZYNQ=y 时,对应下面列举的所有.dts 文件会被编译进去,也包括我们之前添加上去的 system-top.dtb 文件;但是 CONFIG_ARCH_ZYNQ=y 又怎么确定呢?在 20.3.2 小节编译内核的时候我们执行了"make xilinx_zynq_defconfig"这样一条命令,后面的 xilinx_zynq_defconfig (arch/arm/configs/xilinx_zynq_defconfig)文件就是我们zynq 平台对应的 defconfig 配置文件,那么打开它,找到 CONFIG_ARCH_ZYNQ,如下:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

10 CONFIG_BLK_DEV_INITRD=y
11 CONFIG_CC_OPTIMIZE_FOR_SIZE=y
12 CONFIG_SYSCTL_SYSCALL=y
13 # CONFIG_BUG is not set
14 CONFIG_EMBEDDED=y
15 CONFIG_PERF_EVENTS=y
16 CONFIG_SLAB=y
17 CONFIG_ARCH_VEXPRESS=y
18 CONFIG_ARCH_ZYNQ=y
19 CONFIG_PL310_ERRATA_588369=y
20 CONFIG_PL310_ERRATA_727915=y
21 CONFIG_PL310_ERRATA_754322=y
23 CONFIG_ARM_ERRATA_754327=y
24 CONFIG_ARM_ERRATA_764369=y

图 24.2.2 CONFIG_ARCH_ZYNQ

在这个文件当中定义了 CONFIG_ARCH_ZYNQ=y。所以对于 ZYNQ 平台来说,当我们在 内核源码目录下执行"make all"或者是"make dtbs"命令的时候,arch/arm/boot/dts/Makefile 文件 中对应的那些 dts 文件就会被编译成 dtb (二进制文件)文件。但是需要注意并不是说不在这 个 Makefile 文件中列举出来的 dts 文件就不能被编译,我们在使用 make 命令的时候也可以指 定需要编译的 dts 文件,例如下面:

make system-top.dtb

system-top.dtb 表示需要编译出这个 dtb 文件,那么系统就会去 arch/arm/boot/dts 目录下找 到相对应的.dts 文件,也就是 system-top.dts,而 system-top.dts 文件就可以不用记录在 arch/arm/boot/dts/Makefile 文件中,我们编译的时候已经指定了。

关于编译 dts 文件的问题就说到这里了,这里讲的有点太多了,如果大家还不明白可以去 网上找找资料。

24.2.4 dtb

.dtb 文件就是将.dts 文件编译成二进制数据之后得到的文件,这就跟.c 文件编译为.o 文件 是一样的道理,关于.dtb 文件怎么使用这里就不多说了,前面讲解 Uboot 移植、Linux 内核移 植的时候已经无数次的提到如何使用.dtb 文件了(uboot 中使用 bootz 或 bootm 命令向 Linux 内 核传递二进制设备树文件(.dtb))。

24.3 dts 语法

虽然我们基本上不会从头到尾重写一个.dts 文件,大多时候是直接在 SOC 厂商提供的.dts 文件上进行修改。但是 DTS 文件语法我们还是需要详细的学习一遍,因为我们肯定需要修改.dts 文件。大家不要看到要学习新的语法就觉得会很复杂,DTS 语法非常的人性化,是一种 ASCII 文本文件,不管是阅读还是修改都很方便。

本节我们就以 system-top.dts 这个文件为例来讲解一下 DTS 语法。关于设备树详细的语法规则请参考《Devicetree SpecificationV0.2.pdf》和《Power_ePAPR_APPROVED_v1.12.pdf》这两份文档,此两份文档已经放到了开发板光盘中,路径为: ZYNQ 开发板资料盘(A

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php 盘)\8_ZYNQ&FPGA 参考资料\ARM\Devicetree SpecificationV0.2.pdf、领航者 ZYNQ 开发板资 料盘(A 盘)\8_ZYNQ&FPGA 参考资料\ARM\Power_ePAPR_APPROVED_v1.12.pdf。

24.3.1 设备树的结构

设备树用树状结构描述设备信息,组成设备树的基本单元是 node(设备节点),这些 node 被组织成树状结构,有如下一些特征:

- ➤ 一个 device tree 文件中只有一个 root node (根节点);
- ▶ 除了 root node,每个 node 都只有一个 parent node (父节点);
- ➤ 一般来说,开发板上的每一个设备都能够对应到设备树中的一个 node;
- ▶ 每个 node 中包含了若干的 property-value (键-值对,当然也可以没有 value)来描述该 node 的一些特性;
- ➢ 每个 node 都有自己的 node name(节点名字);
- ▶ node之间可以是平行关系,也可以嵌套成父子关系,这样就可以很方便的描述设备间的关系;

下面给出一个设备树的简单的结构示意图:

示例代码 24.3.1 设备树结构示意

	0.1001.000
2	node1{
3	property1=value1; // node1 节点的属性 property1
4	property2=value2; // node1 节点的属性 property2
5	
6	};
7	
8	node2{
9	property3=value3; // node2 节点的属性 property3
10	
11	node3{ // node2的子节点 node3
12	property4=value4; // node3 节点的属性 property4
13	
14	};
15	};
16 };	
4 5 6 7 8 9 10 11 12 13 14 15 16};	property2=value2; // nodel 节点的属性 property2 }; node2{ // node2 节点 property3=value3; // node2 节点的属性 property2 node3{ // node2 的子节点 node3 property4=value4; // node3 节点的属性 property4 };

第1行当中的'/'就表示设备树的 root node(根节点),所以可知 node1 节点和 node2 节点的父节点都是 root node,而 node3 节点的父节点则是 node2, node2 与 node3 之间形成了父子 节点关系。Root node下面的子节点 node1 和 node2 可以表示为 SoC 上的两个控制器,而 node3 则可以表示挂在 node2 控制器上的某个设备,例如 node2 表示 ZYNQ PS 的一个 I2C 控制器,而 node3 则表示挂在该 I2C 总线下的某个设备,例如 eeprom、RTC 等。

24.3.2 节点与属性

在设备树文件中如何定义一个节点,节点的命名有什么要求呢?在设备树中节点的命名 格式如下:



原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php

[label:]node-name[@unit-address] { [properties definitions] [child nodes] };

"[]"中的内容表示可选的,可有也可以没有;节点名字前加上"label"则方便在 dts 文件中 被其他的节点引用,我们后面会说这个;其中"node-name"是节点名字,为ASCII字符串, 节点名字应该能够清晰的描述出节点的功能,比如 "uart1" 就表示这个节点是 UART1 外设。 "unit-address"一般表示设备的地址或寄存基地址,如果某个节点没有地址或者寄存器的话 "unit-address"可以不要,比如"cpu@0"、"interrupt-controller@00a01000"。

每个节点都有若干属性,属性又有相对应的值(值不是必须要有的),而一个节点当中 又可以嵌套其它的节点,形成父子节点。例如下面:

示例代码 24.3.2 设备树节点示例

```
20 cpus {
```

```
#address-cells = <1>;
21
     \#size-cells = <0>;
22
23
24
    cpu0: cpu@0 {
25
       compatible = "arm,cortex-a9";
26
       device_type = "cpu";
27
       reg = <0>;
       clocks = <\&clkc 3>;
28
29
       clock-latency = <1000>;
       cpu0-supply = <&regulator_vccpint>;
30
31
       operating-points = <
32
         /* kHz uV */
33
         666667 1000000
34
         333334 1000000
35
         >;
36
    };
37
     cpu1: cpu@1 {
38
39
       compatible = "arm,cortex-a9";
       device_type = "cpu";
40
       reg = <1>;
41
       clocks = <\&clkc 3>:
42
43
       };
44 };
```

每一个节点(包括 root node)都会使用一组括号"{}"将自己的属性以及子节点包含在里 边,注意括号外需要加上一个分号";",包括每一个属性都使用一个分号来结束。有点像 C 语言中的表达式后面的分号。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

第 20 行当中的 cpus 节点,它的名字只有" [label:]node-name[@unit-address]"当中的"node-name"部分,并没有其它两部分;第 24 行节点的定义包含了所有的组成部分,包括 label 以及 unit-address;关于 label 的作用的我们后面专门讲,这里先不说。

cpus 节点有两个属性" #address-cells"和" #size-cells",它们的值分别为" <1>"和" <0>"。例如 cpu@0 节点中有 compatible、device_type、reg、clocks 属性等,它们都有对应的值,大家看到这些值可能有点不明白,为啥有的是字符串,有的是尖括号"<>"括起来的东西,下面单独给大家讲解一波。

每个节点都有不同属性,不同的属性又有不同的值,那么设备树当中值有哪些形式呢?

● 字符串

compatible = "arm,cortex-a9";

字符串使用双引号括起来,例如上面的这个 compatible 属性的值是" arm, cortex-a9"字符串。

● 32 位无符号整形数据

clock-latency = <1000>;

reg = <0x0000000 0x00500000>;

32 位无符号整形数据使用尖括号括起来,例如属性 clock-latency 的值是一个 32 位无符号 整形数据 1000,而 reg 属性有两个数据,使用空格隔开,那么这个就可以认为是一个数组, 很容易理解!

● 二进制数据

local-mac-address = [00 0a 35 00 1e 53];

二进制数据使用方括号括起来,例如上面这个就是一个二进制数据组成的数组。

● 字符串数组

compatible = "n25q512a","micron,m25p80";

属性值也可以使用字符串列表,例如上面的这个属性,它的值是一个字符串列表,字符 串之间使用逗号分割;

● 混合值

mixed-property = "a string", [0x01 0x23 0x45 0x67], <0x12345678>;

除此之外不同的数据类型还可以混合在一起,以逗号分隔。

● 节点引用

除了上面一些数据类型之外,还有一种非常常见的形式,如下所示:

clocks = <&clkc 3>;

这其实就是我们上面说到的引用节点的一种形式,"&clkc"就表示引用"clkc"这个节点, 而 clkc 就是前面提到的"label"。引用节点也是使用尖括号来表示,关于节点之间的引用,我 们后面还会再讲,这里先告一段落。

24.3.3 使用注释和宏定义

在设备树文件中也可以使用注释,注释的方法和 C 语言当中是一样的,可以使用" // "进行单行注释,也可以使用" // * */ "进行多行注释,如下所示:

示例代码 24.3.3 设备树中注释示例

1 // SPDX-License-Identifier: GPL-2.0+

2 /*

3 * Copyright (C) 2011 - 2015 Xilinx

```
原子哥在线教学: www.yuanzige.com
                                                 论坛:www.openedv.com/forum.php
     4 *
     5 * This software is licensed under the terms of the GNU General Public
     6 * License version 2, as published by the Free Software Foundation, and
     7 * may be copied, distributed, and modified under those terms.
     8 *
     9 * This program is distributed in the hope that it will be useful,
     10 * but WITHOUT ANY WARRANTY; without even the implied warranty of
     11 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
     12 * GNU General Public License for more details.
     13 */
     14
    15/{
     16
           #address-cells = <1>;
     17
           \#size-cells = <1>;
           compatible = "xlnx,zynq-7000";
     18
     19
    20
           cpus {
     21
             #address-cells = <1>;
             #size-cells = <0>;
     22
     23
     24
             cpu0: cpu@0 {
    25
               compatible = "arm,cortex-a9";
    26
               device_type = "cpu";
    27
               reg = <0>;
    28
               clocks = \langle \&clkc 3 \rangle;
    29
               clock-latency = <1000>;
               cpu0-supply = <&regulator_vccpint>;
    30
    31
               operating-points = <
                  /* kHz uV */
     32
     33
                  666667 1000000
     34
                  333334 1000000
     35
                  >;
    36
             };
    37
    38
             cpu1: cpu@1 {
     39
               compatible = "arm,cortex-a9";
    40
               device_type = "cpu";
    41
               reg = <1>;
    42
               clocks = <\&clkc 3>;
    43
             };
     44
           };
```

正点原子



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php 前面跟大家讲过,设备树中可以使用"#include"包含 dtsi、dts 以及 C 语言的头文件,那我 们为什么要包含一个.h 的头文件呢?因为在设备树中可以使用宏定义,所以你在 arch/arm/boot/dts 目录下你会看到很多的设备树文件中都包含了.h 头文件,例如下面这个:



图 24.3.1 头文件包含

10		100000	11/3000	
80		>;		
81		clocks = <&clks	IMX6UL_CLK_ARM>,	
82		<&clks	IMX6UL_CLK_PLL2_BUS>,	
83		<&clks	IMX6UL_CLK_PLL2_PFD2>,	
84		<&clks	IMX6UL_CA7_SECONDARY_SEL>,	
85		<&clks	IMX6UL_CLK_STEP>,	
86		<&clks	IMX6UL_CLK_PLL1_SW>,	
87		<&clks	IMX6UL_CLK_PLL1_SYS>,	
88		<&clks	IMX6UL_PLL1_BYPASS>,	
89		<&clks	IMX6UL_CLK_PLL1>,	
90		<&clks	IMX6UL_PLL1_BYPASS_SRC>,	
91		<&clks	IMX6UL_CLK_OSC>;	
92		clock-names = "a	arm", "pll2_bus", "pll2_pfd2_3	396m",
93			secondary_sel", "step", "pll1_	sw",
94		"	oll1_sys", "pll1_bypass", "pll.	1",
95		"	oll1_bypass_src", "osc";	
96		arm-supply = <&	reg_arm>;	
97		soc-supply = <&	reg_soc>;	
98	};			
99	};			

图 24.3.2 使用宏定义

关于头文件包含以及宏定义的使用这里就不多说了,本身也非常简单。

24.3.4 标准属性

节点的内容是由一堆的属性组成,不同的设备需要的属性不同,用户可以自定义属性。 除了用户自定义属性,有很多属性是标准属性,Linux下的很多外设驱动都会使用这些标准属 性,本节我们就来学习一下几个常用的标准属性。

1、compatible 属性



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

compatible 属性也叫做"兼容性"属性,这是非常重要的一个属性! compatible 属性的值可以是一个字符串,也可以是一个字符串列表;一般该字符串使用"<制造商>,<型号>"这样的形式进行命名,当然这不是必须要这样,这是要求大家按照这样的形式进行命名,目的是为了指定一个确切的设备,并且包括制造商的名字,以避免命名空间冲突,如下所示:

compatible = "xlnx,xuartps", "cdns,uart-r1p8";

例子当中的 xlnx 和 cdns 就表示制造商,而后面的 xuartps 和 uart-r1p8 就表示具体设备的型号。compatible 属性用于将设备和驱动绑定起来,例如该设备首先使用第一个兼容值

(xlnx,xuartps)在 Linux 内核里面查找,看看能不能找到与之匹配的驱动文件,如果没有找到的话就使用第二个兼容值(cdns,uart-r1p8)查找,直到找到或者查找完整个 Linux 内核也没有找到对应的驱动。

一般驱动程序文件都会有一个 OF 匹配表,此 OF 匹配表保存着一些 compatible 值,如果 设备树中的节点的 compatible 属性值和 OF 匹配表中的任何一个值相等,那么就表示设备可以 使用这个驱动。比如在驱动文件 drivers/tty/serial/xilinx_uartps.c 中有如下内容:

示例代码 24.3.4 drivers/tty/serial/xilinx_uartps.c 内容片段

```
1342
```

```
1343 /* Match table for of_platform binding */
```

```
1344 static const struct of_device_id cdns_uart_of_match[] = {
```

```
1345 {.compatible = "xlnx,xuartps", },
```

1346 {.compatible = "cdns,uart-r1p8", },

1347 {.compatible = "cdns,uart-r1p12", .data = &zynqmp_uart_def },

1348 {.compatible = "xlnx,zynqmp-uart", .data = &zynqmp_uart_def },

1349 {}

1350 };

1351 MODULE_DEVICE_TABLE(of, cdns_uart_of_match);

•••••

```
1703
```

1704 static struct platform_driver cdns_uart_platform_driver = {

```
1705 .probe = cdns_uart_probe,
```

```
1706 .remove = cdns_uart_remove,
```

```
1707 .driver = {
```

```
1708 .name = CDNS_UART_NAME,
```

```
1709 .of_match_table = cdns_uart_of_match,
```

```
.pm = &cdns_uart_dev_pm_ops,
```

```
1711 },
```

```
1712 };
```

这个驱动文件是 ZYNQ PS 端的 UART 设备对应的驱动文件。

第 1344~1350 行定义的数组 cdns_uart_of_match 就是 xilinx_uartps.c 这个驱动文件的匹配 表,此匹配表有 4 个匹配值 "xlnx,xuartps"、"cdns,uart-r1p8"、"cdns,uart-r1p12"以及 "xlnx,zynqmp-uart"。如果在设备树中有哪个节点的 compatible 属性值与这 4 个字符串中的 某个相同,那么这个节点就会与此驱动文件匹配成功。



原子哥在线教学: www.yuanzige.com 论均

论坛:www.openedv.com/forum.php

第 1704 行, UART 驱动基于 platform_driver 总线框架,关于 platform_driver 驱动后面会 讲解。此行设置.of_match_table为cdns_uart_of_match,也就是设置这个 platform_driver 所使用 的 OF 匹配表。

2、model 属性

model 属性值也是一个字符串描述信息,它指定制造商的设备型号,model 属性一般定义 在根节点下,一般就是对板子的描述信息,没啥实质性的作用,内核在解析设备树的时候会 把这个属性对应的字符串信息打印出来。

```
示例代码 24.3.5 arch/arm/boot/dts/system-top.dts 内容片段
```

```
8 /dts-v1/;
9 #include "zynq-7000.dtsi"
10 #include "pl.dtsi"
11 #include "pcw.dtsi"
12/{
        model = "Alientek ZYNQ Development Board";
13
14
15
        chosen {
             bootargs = "console=ttyPS0,115200 earlyprintk root=/dev/mmcblk0p2 rw rootwait";
16
             stdout-path = "serial0:115200n8";
17
        };
18
19
        aliases {
20
             ethernet0 = \& \text{gem}0;
21
             i2c0 = \&i2c2;
22
             i2c1 = \&i2c0;
23
             i2c2 = \&i2c1;
             serial0 = \&uart0;
24
25
             serial1 = \&uart1;
26
             spi0 = \&qspi;
27
        };
28
        memory {
29
             device_type = "memory";
             reg = \langle 0x0 \ 0x20000000 \rangle;
30
31
        };
32 };
```

之前我在 system-top.dts 设备树文件加了一个 model 属性,它的值等于 "Alientek ZYNQ Development Board",内核启动过程中就会打印出来,如下所示:



原子哥在线教学:www.yuanzige.com 论坛:www.openedv.com/forum.php yng align dma buffer: Align buffer at 10007d to fff80(swap 1) reading zImage 3965560 bytes read in 253 ms (14.9 MiB/s) reading system.dtb 3457 bytes read in 17 ms (772.5 KiB/s) # Flattened Device Tree blob at 02000000 Booting using the fdt blob at 0x2000000 Loading Device Tree to 1eb12000, end 1eb18490 ... OK starting kernel ... Booting Linux on physical CPU 0x0 Linux version 4.14.0-xilinx (zynq@ubuntu) (gcc version 7.3.1 20180314 (Linaro GCC 7.3-2018.04-rc3)) #1 SMP PREE CPU: ARMv7 Processor [413fc090] revision 0 (ARMv7), cr=18c5387d CPU: PIPT / VIPT nonaliasing data cache. VIPT aliasing instruction cache OF: fdt: Machine model: Alientek ZYNQ Development Board emory policy: Data cache writeall ma: Reserved 16 MiB at 0x1f000000 random: fast init done percpu: Embedded 16 pages/cpu @debc5000 s34764 r8192 d22580 u65536 Built 1 zonelists, mobility grouping on. Total pages: 130048 Kernel command line: console=ttyPS0,115200 earlyprintk root=/dev/mmcblk0p2 rw rootwait ID hash table entries: 2048 (order: 1, 8192 bytes) entry cache hash table entries: 65536 (order: 6, 262144 bytes)

图 24.3.3 打印 model 字符串

3、status 属性

status 属性看名字就知道是和设备状态有关的, device tree 中的 status 标识了设备的状态, 使用 status 可以去禁止设备或者启用设备, 看下设备树规范中的 status 可选值:

表 24.3.1 status 属性值

值	描述
okay	表明设备是可操作的。启动设备
disabled	表明设备当前是不可操作的,但是在未来可以变为可操作的,比如 热插拔设备插入以后。至于 disabled 的具体含义还要看设备的绑 定文档。
fail	表明设备不可操作,设备检测到了一系列的错误,而且设备也不大 可能变得可操作。
fail-sss	含义和"fail"相同,后面的 sss 部分是检测到的错误内容。

注意如果节点中没有添加 status 属性,那么它默认就是"status = okay"。

4、#address-cells 和#size-cells 属性

这两个属性的值都是无符号 32 位整形, #address-cells 和#size-cells 这两个属性可以用在任何拥有子节点的设备节点中,用于描述子节点的地址信息。

▶ #address-cells,用来描述子节点"reg"属性的地址表中首地址 cell 的数量;

▶ #size-cells,用来描述子节点"reg"属性的地址表中地址长度 cell 的数量。

#address-cells 和#size-cells 表明了子节点应该如何编写 reg 属性值,一般 reg 属性都是和地 址有关的内容,和地址相关的信息有两种:起始地址和地址长度,有了这两个属性,子节点 中的"reg"属性就可以描述一块连续的地址区域了; reg 属性的格式一般为:

reg = <address1 length1 address2 length2 address3 length3.....>

每个 "address length" 组合表示一个地址范围,其中 address 是起始地址, length 是地址 长度, #address-cells 表明 address 字段占用的字长, #size-cells 表明 length 这个字段所占用的字 长,比如:

示例代码 24.3.6 #address-cells 和#size-cells 属性

<mark>38 &</mark>qspi {



```
原子哥在线教学: www.yuanzige.com
                                                    论坛:www.openedv.com/forum.php
     39
          #address-cells = <1>;
     40
          \#size-cells = <0>;
          flash0: flash@0 {
     41
             compatible = "n25q512a", "micron, m25p80";
     42
             reg = <0x0>;
     43
             #address-cells = <1>;
     44
             \#size-cells = <1>;
     45
     46
             spi-max-frequency = <50000000>;
             partition@0x00000000 {
     47
               label = "boot";
     48
     49
               reg = \langle 0x0000000 \ 0x00500000 \rangle;
     50
             };
             partition@0x00500000 {
     51
               label = "bootenv";
     52
               reg = \langle 0x00500000 \ 0x00020000 \rangle;
     53
     54
             };
             partition@0x00520000 {
     55
               label = "kernel";
     56
     57
               reg = <0x00520000 0x00a80000>;
     58
             };
     59
             partition@0x00fa0000 {
     60
               label = "spare";
               reg = \langle 0x00fa0000 \ 0x00000000 \rangle;
     61
     62
             };
     63
          };
     64 };
```

第 39~40 行,节点 qspi 的#address-cells = <1>, #size-cells = <0>,说明 qspi 的子节点 reg 属性中起始地址使用一个 32bit 数据来表示,地址长度没有;第 43 行,qspi 的子节点 flash0:flash@0 的 reg 属性值为<0>,因为父节点设置了#address-cells = <1>, #size-cells = <0>,因此 addres=0,没有 length 的值,相当于设置了起始地址,而没有设置地址长度。

第 44~45 行,设置 flash0:flash@0 节点#address-cells = <1>, #size-cells = <1>,说明 flash0:flash@0 的子节点起始地址长度所占用的字长为 1,地址长度所占用的字长也为 1。第 49 行,flash0:flash@0 的子节点 partition@0x00000000 的 reg 属性值为 reg = <0x00000000 0x00500000>,因为父节点设置了#address-cells = <1>, #size-cells = <1>,所以 address 使用一 个 32bit 数据来表示,也就是 address=0x00000000,而 length 也使用一个 32bit 数据来表示,也就是 length=0x00500000,相当于设置了起始地址为 0x00000000,地址长度为 0x00500000。

5、reg 属性

reg 属性前面已经提到过了, reg 属性的值一般是(address, length)对。reg 属性一般用于描述设备地址空间资源信息,一般都是描述某个外设的寄存器地址范围信息、flash 设备的分区信息等,比如在 arch/arm/boot/dts/zynq-7000.dts 文件中有如下内容:

示例代码 24.3.7 uart0 节点信息

原子哥在线教学: www.yuanzige.com 论坛:www.openedy.com/forum.php

174 uart0: serial@e0000000 {

175 compatible = "xlnx,xuartps", "cdns,uart-r1p8";

领航者 ZYNQ 之嵌入式 Linux 开发指南

- 176 status = "disabled";
- 177 clocks = <&clkc 23>, <&clkc 40>;
- 178 clock-names = "uart_clk", "pclk";
- 179 reg = <0xE000000 0x1000>;
- 180 interrupts = <0 27 4>;

181 };

上述代码是节点 uart0, uart0 节点描述了 ZYNQ PS 端的 UART0 相关信息,重点是第 179 行的 reg 属性。其中 uart0 的父节点 amba 设置了#address-cells = <1>、#size-cells = <1>,因此 reg 属性中 address= 0xE0000000, length= 0x1000。查阅 ZYNQ 的数据手册(ZYNQ 开发板资 料盘(A 盘)\8_ZYNQ&FPGA 参考资料\Xilinx\User Guide\ug585-Zynq-7000-TRM.pdf)可知, ZYNQ 的 UART0 寄存器首地址确实为 0xE0000000,但是 UART0 的地址长度(范围)并没有 0x1000 这么多,这里我们重点是获取 UART0 寄存器首地址,只要地址空间没有跨越到其它 外设的地址空间也没什么影响。

正点原子

6、ranges 属性

ranges 是地址转换表,其中的每个项目是一个子地址、父地址以及在子地址空间的大小的映射。ranges 属性值可以为空或者按照(child-bus-address,parent-bus-address,length)格式编写的数字矩阵。映射表中的子地址、父地址占用的字长分别由 ranges 属性所在节点的#address-cells 属性和 ranges 属性所在节点的父节点的#address-cells 属性来确定。而子地址空间长度占用的字长由 ranges 属性所在节点的#address-cells 属性决定。

child-bus-address: 子总线地址空间的物理地址,由 ranges 属性所在节点的#address-cells 属性确定此物理地址占用的字长。

parent-bus-address: 父总线地址空间的物理地址,由 ranges 属性所在节点的父节点的 #address-cells 属性确定此物理地址所占用的字长。

length: 子地址空间的长度,由 ranges 属性所在节点的#address-cells 属性确定此地址长度 所占用的字长。

如果 ranges 属性值为空值,说明子地址空间和父地址空间完全相同,不需要进行地址转换,对于我们所使用的 ZYNQ 来说,子地址空间和父地址空间完全相同,因此会在 zynq-7000.dtsi 文件中找到大量的值为空的 ranges 属性,如下所示:

```
示例代码 24.3.8 zynq-7000.dtsi 内容片段
```

```
46 fpga_full: fpga-full {
```

```
47 compatible = "fpga-region";
```

- 48 fpga-mgr = $\langle \& devcfg \rangle$;
- 49 #address-cells = <1>;

```
50 #size-cells = <1>;
```

51 ranges;

```
52 };
```

第51行定义了 ranges 属性,但是 ranges 属性值为空。

ranges 属性不为空的示例代码如下所示:

```
示例代码 24.3.9 ranges 属性不为空
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

1 soc {

- 2 compatible = "simple-bus";
- 3 #address-cells = <1>;
- 4 #size-cells = <1>;
- 5 ranges = <0x0 0xe0000000 0x00100000>;

```
6
7 se
```

```
serial {
```

```
8 device_type = "serial";
```

- 9 compatible = "ns16550";
- 10 reg = <0x4600 0x100>;
- 11 clock-frequency = $\langle 0 \rangle$;
- 12 interrupts = $\langle 0xA \ 0x8 \rangle$;
- 13 interrupt-parent = <&ipic>;

14 };

```
15 };
```

第5行,节点 soc 定义的 ranges 属性,值为<0x0 0xe0000000 0x00100000>,此属性值指定 了一个 1024KB(0x00100000)的地址范围,子地址空间的物理起始地址为 0x0,父地址空间的 物理起始地址为 0xe0000000。

第10行, serial 是串口设备节点, reg 属性定义了 serial 设备寄存器的起始地址为0x4600, 寄存器长度为 0x100。经过地址转换, serial 设备可以从 0xe0004600 开始进行读写操作, 0xe0004600=0x4600+0xe0000000。

7、device_type 属性

device_type 属性值为字符串,表示节点的类型;此属性在设备树当中用的比较少,一般用于 cpu 节点或者 memory 节点。zynq-7000.dtsi 文件中的 cpu0 和 cpu1 节点用到了此属性,内容如下所示:

```
示例代码 24.3.10 zynq-7000.dtsi 内容片段
```

```
24 cpu0: cpu@0 {
```

```
25 compatible = "arm,cortex-a9";
```

```
26 device_type = "cpu";
```

```
27 reg = <0>;
```

```
28 clocks = <&clkc 3>;
```

```
29 clock-latency = <1000>;
```

30 cpu0-supply = <®ulator_vccpint>;

```
31 operating-points = <
```

```
32 /* kHz uV */
```

```
33 666667 1000000
```

```
34 333334 1000000
```

>;

```
35
```

```
36   };
37
```

```
38 cpu1: cpu@1 {
```



原子哥在线教学:www.yuanzige.com 论:

论坛:www.openedv.com/forum.php

```
39 compatible = "arm,cortex-a9";
```

```
40 device_type = "cpu";
```

```
41 reg = <1>;
```

```
42 clocks = <\&clkc 3>;
```

```
43 };
```

关于标准属性就讲解这么多,后面还会跟大家介绍一些常常会使用到的节点,例如设备 树中的中断控制器、GPIO、I2C总线等。

24.3.5 根节点 compatible 属性

每个节点都有 compatible 属性(除了一些特殊用途的节点),根节点"/"也不例外,在 zynq-7000.dtsi 文件中根节点的 compatible 属性内容如下所示:

示例代码 24.3.11 zynq-7000.dtsi 根节点 compatible 属性

```
15/{
```

```
16 #address-cells = <1>;
17 #size-cells = <1>;
```

18 compatible = "xlnx,zynq-7000";

```
431 };
```

可以看出, compatible 有一个值: "xlnx,zynq-7000"。前面我们说了,设备节点的 compatible 属性值是为了匹配 Linux 内核中的驱动程序,那么根节点中的 compatible 属性是为 了做什么工作的? 同样根节点下的 compatible 属性的值可以是一个字符串,也可以是一个字符串列表;该字符串也要求以"<制造商>,<型号>"这样的形式进行命名;比如这里使用的是 "xlnx"制造的"zynq-7000"系列处理器。

通过根节点的 compatible 属性可以知道我们所使用的处理器型号, Linux 内核会通过根节 点的 compoatible 属性查看是否支持此该处理器,因为内核在启动初期会进行校验,必须要支 持才会启动 Linux 内核。接下来我们就来学习一下 Linux 内核在使用设备树之前以及使用设备 树之后是如何判断是否支持某款处理器的。

1、使用设备树之前的校验方法

在没有使用设备树以前,uboot 会向 Linux 内核传递一个叫做 machine id 的值,machine id 可以认为就是一个机器 ID 编码,告诉 Linux 内核自己是个什么硬件平台,看看 Linux 内核是 否支持。Linux 内核是支持很多硬件平台的,但是针对每一个特定的板子,Linux 内核都用 MACHINE_START 和 MACHINE_END 来定义一个 machine_desc 结构体来描述这个硬件平台,比如在文件 arch/arm/mach-imx/mach-mx35 3ds.c 中有如下定义:

示例代码 24.3.12 MX35_3DS 设备

```
613 MACHINE_START(MX35_3DS, "Freescale MX35PDK")
```

```
614 /* Maintainer: Freescale Semiconductor, Inc */
```

```
615 .atag_offset = 0x100,
```

```
616 .map_io = mx35_map_io,
```

```
617 .init_early = imx35_init_early,
```

```
618 .init_irq = mx35_init_irq,
```

```
619 .init_time = mx35pdk_timer_init,
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

620 .init_machine = $mx35_3ds_init$,

621 .reserve = $mx35_3ds_reserve$,

 $622 \quad .restart = mxc_restart,$

```
623 MACHINE_END
```

上述代码就是定义了"Freescale MX35PDK"这个硬件平台,其中 MACHINE_START 和 MACHINE_END 定义在文件 arch/arm/include/asm/mach/arch.h 中,内容如下:

\

示例代码 24.3.13 MACHINE_START 和 MACHINE_END 宏定义

#define MACHINE_START(_type,_name)

\

static const struct machine_desc __mach_desc_##_type \

__used

```
__attribute__((__section__(".arch.info.init"))) = { \
```

```
.nr = MACH_TYPE_##_type,
```

.name = _name,

#define MACHINE_END

```
};
```

根据 MACHINE_START 和 MACHINE_END 的宏定义,将上面示例代码展开后如下所示: 示例代码 24.3.14 展开以后

```
1 static const struct machine_desc __mach_desc_MX35_3DS \
```

2 __used

```
3 __attribute__((__section__(".arch.info.init"))) = {
```

```
4 .nr = MACH_TYPE_MX35_3DS,
```

```
5 .name = "Freescale MX35PDK",
```

```
6 /* Maintainer: Freescale Semiconductor, Inc */
```

```
7 .atag_offset = 0x100,
```

```
8 .map_io = mx35_map_io,
```

```
9 .init_early = imx35_init_early,
```

```
10 .init_irq = mx35_init_irq,
```

```
init_time = mx35pdk_timer_init,
```

```
12 .init_machine = mx35_3ds_init,
```

```
13 .reserve = mx35_3ds_reserve,
```

```
14 .restart = mxc_restart,
```

```
15 };
```

从示例代码 24.3.14 中可以看出,这里定义了一个 machine_desc 类型的结构体变量 __mach_desc_MX35_3DS,这个变量存储在".arch.info.init"段中。第4行的 MACH_TYPE_MX35_3DS 就是"Freescale MX35PDK"这个板子的 machine id。 MACH_TYPE_MX35_3DS 定义在文件 include/generated/mach-types.h 中,此文件定义了大量的 machine id,内容如下所示:

0

示例代码 24.3.15 mach-types.h 文件中的 machine id

15 #define MACH_TYPE_EBSA110

16#define MACH_TYPE_RISCPC



原子语	哥在线教学: www.yuanzige.com	论坛:www.openedv.com/forum.php	
17	#define MACH_TYPE_EBSA285	4	
18	#define MACH_TYPE_NETWINDER	5	
19	#define MACH_TYPE_CATS	6	
20) #define MACH_TYPE_SHARK	15	
21	#define MACH_TYPE_BRUTUS	16	
22	2 #define MACH_TYPE_PERSONAL_SERV	VER 17	
28	87 #define MACH_TYPE_MX35_3DS	1645	
1(000 #define MACH_TYPE_PFLA03	4575	

第 287 行就是 MACH_TYPE_MX35_3DS 的值,为 1645。

前面说了,uboot会给Linux内核传递machine id这个参数,Linux内核会检查这个machine id,其实就是将machine id 与示例代码 24.3.15 中的这些 MACH_TYPE_XXX 宏进行对比,看 看有没有相等的,如果相等的话就表示Linux内核支持这个硬件平台,如果不支持的话就没 法启动Linux内核。

2、使用设备树以后的设备匹配方法

当 Linux 内核引入设备树以后就不再使用 MACHINE_START 了,而是换为了 DT_MACHINE_START 。 DT_MACHINE_START 也 定 义 在 文 件 arch/arm/include/asm/mach/arch.h 里面,定义如下:

```
示例代码 24.3.16 DT MACHINE START 宏
```

```
#define DT_MACHINE_START(_name, _namestr) \
static const struct machine_desc __mach_desc_##_name \
__used \
__attribute__((__section__(".arch.info.init"))) = { \
.nr = ~0, \
.name = _namestr, \
```

可以看出,DT_MACHINE_START和 MACHINE_START 基本相同,只是.nr 的设置不同, 在 DT_MACHINE_START 里面直接将.nr 设置为~0。说明引入设备树以后不会再根据 machine id 来检查 Linux 内核是否支持某个硬件平台了。

打开文件 arch/arm/mach-zynq/common.c,有如下所示内容:

```
示例代码 24.3.17 arch/arm/mach-zynq/common.c
```

```
191 static const char * const zynq_dt_match[] = {
192 "xlnx,zynq-7000",
193 NULL
194 };
195
196 DT_MACHINE_START(XILINX_EP107, "Xilinx Zynq Platform")
197 /* 64KB way size, 8-way associativity, parity disabled */
198 #ifdef CONFIG_XILINX_PREFETCH
199 .l2c_aux_val = 0x30400000,
200 .l2c_aux_mask = 0xcfbfffff,
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

201 #else

```
202
        .12c aux val = 0x00400000,
203
        .l2c_aux_mask = 0xffbfffff,
204 #endif
205
        .smp
                   = smp_ops(zynq_smp_ops),
206
        .map io
                    = zyng map io,
207
        .init_irq
                   = zynq_irq_init,
208
        .init_machine = zynq_init_machine,
209
        .init late
                   = zynq_init_late,
210
        .init time
                    = zyng timer init,
211
        .dt_compat
                      = zynq_dt_match,
212
        .reserve
                    = zynq_memory_init,
```

213 MACHINE_END

machine_desc 结构体中有个.dt_compat 成员变量,此成员变量保存着本硬件平台的兼容属性,示例代码 24.3 中中设置.dt_compat = zynq_dt_match, zynq_dt_match 数组的定义在第 191~194 行中,可以看到它匹配的字符串是"xlnx,zynq-7000"。只要某个板子的设备树根节 点"/"的 compatible 属性值与 zynq_dt_match 表中的任何一个值相等,那么就表示 Linux 内核 支持这个开发板、支持这个硬件平台。前面也跟大家说过了,我们使用的设备树文件是 system-top.dts,该文件中使用 include 包含了 zynq-7000.dtsi,在 zynq-7000.dtsi 文件中根节点的 compatible 属性值就是"xlnx,zynq-7000",所以内核是支持我们开发板的

如果将 zynq-7000.dtsi 根节点的 compatible 属性改为其他的值,那么它就启动不了了。

当我们修改了根节点 compatible 属性内容以后,因为 Linux 内核找不到对应的硬件平台,因此 Linux 内核无法启动。

接下来我们简单看一下 Linux 内核是如何根据设备树根节点的 compatible 属性来匹配出对 应的 machine_desc, Linux 内核调用 start_kernel 函数来启动内核, start_kernel 函数会调用 setup_arch 函数来匹配 machine_desc, setup_arch 函数定义在文件 arch/arm/kernel/setup.c 中, 函数内容如下(有缩减):

示例代码 24.3.18 setup_arch 函数内容

```
913 void __init setup_arch(char **cmdline_p)
914 {
915
     const struct machine_desc *mdesc;
916
917
      setup_processor();
      mdesc = setup_machine_fdt(__atags_pointer);
918
919
      if (!mdesc)
920
      mdesc = setup_machine_tags(___atags_pointer, ___machine_arch_type);
      machine desc = mdesc;
921
922
      machine_name = mdesc->name;
.....
986 }
```



原子哥在约	线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
第 91	8行,调用 setup_machine_fdt 函数来获取匹配的 machine_desc,参数就是 atags 的首
地址,也就	就是 uboot 传递给 Linux 内核的 dtb 文件首地址, setup_machine_fdt 函数的返回值就
是找到的E	已经匹配成功的 machine_desc。
函数。	setup_machine_fdt 定义在文件 arch/arm/kernel/devtree.c 中,内容如下(有缩减):
	示例代码 24.3.19 setup_machine_fdt 函数内容
204 con	nst struct machine_desc *init setup_machine_fdt(unsigned int dt_phys)
205 {	
206 c	<pre>const struct machine_desc *mdesc, *mdesc_best = NULL;</pre>
•••••	
214	
215 i	f (!dt_phys !early_init_dt_verify(phys_to_virt(dt_phys)))
216	return NULL;
217	
218 r	mdesc = of_flat_dt_match_machine(mdesc_best, arch_get_next_mach);
219	
•••••	
247	machine_arch_type = mdesc->nr;
248	
249 r	return mdesc;
250 }	
第 21	18 行,调用函数 of_flat_dt_match_machine 来获取匹配的 machine_desc,参数
1 1	

mdesc_best 是默认的 machine_desc,参数 arch_get_next_mach 是个函数,此函数定义在 arch/arm/kernel/devtree.c 文件中。找到匹配的 machine_desc 的过程就是用设备树根节点的 compatible 属性值和 Linux 内核中保存的所有的 machine_desc 结构体的.dt_compat 中的值比较, 看看哪个相等,如果相等的话就表示找到匹配的 machine_desc, arch_get_next_mach 函数的工 作就是获取 Linux 内核中下一个 machine_desc 结构体。

最后再来看一下 of_flat_dt_match_machine 函数,此函数定义在文件 drivers/of/fdt.c 中,内容如下(有缩减):

示例代码 24.3.20 of_flat_dt_match_machine 函数内容

705 const void * __init of_flat_dt_match_machine(const void *default_match,

```
706 const void * (*get_next_compat)(const char * const**))
```

```
707 {
```

```
708 const void *data = NULL;
```

```
709 const void *best_data = default_match;
```

```
710 const char *const *compat;
```

```
711 unsigned long dt_root;
```

```
712 unsigned int best_score = \sim 1, score = 0;
```

```
713
```

714 dt_root = of_get_flat_dt_root();

715 while ((data = get_next_compat(&compat))) {

```
716 score = of_flat_dt_match(dt_root, compat);
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

717	<pre>if (score > 0 && score < best_score) {</pre>
718	best_data = data;
719	<pre>best_score = score;</pre>
720	}
721	}
•••••	
739	
740	pr_info("Machine model: %s\n", of_flat_dt_get_machine_name());
741	
742	return best_data;
743 }	
第 7	14行,通过函数 of_get_flat_dt_root 获取设备树根节点。

第 715~720 行,此循环就是查找匹配的 machine_desc 过程,第 716 行的 of_flat_dt_match 函数会将根节点 compatible 属性的值和每个 machine_desc 结构体中.dt_compat 的值进行比较,直至找到匹配的那个 machine_desc。

总结一下,Linux内核通过根节点 compatible 属性找到对应的 machine_desc 结构体的函数 调用过程,如下图所示:

具体的查找machine_desc的过程

图 24.3.4 查找匹配 machine_desc 的过程

24.3.6 引用节点

前面说到节点的命名格式如下所示:

[label:]node-name[@unit-address]

也多次给大家提到"label"字段,引入 label 的目的就是为了方便访问节点,可以直接通过&label 来访问这个节点,例如下面这个模板:

示例代码 24.3.21 设备树模板

```
1 / {
2 aliases {
3 can0 = &flexcan1;
4 };
5
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

```
6
    cpus {
7
       #address-cells = <1>;
8
       \#size-cells = <0>;
9
10
       cpu0: cpu@0 {
          compatible = "arm,cortex-a7";
11
          device_type = "cpu";
12
13
          reg = <0>;
14
       };
15
     };
16
17
     intc: interrupt-controller@00a01000 {
        compatible = "arm,cortex-a7-gic";
18
19
        #interrupt-cells = <3>;
        interrupt-controller;
20
        reg = <0x00a01000 0x1000>,
21
22
            <0x00a02000 0x100>;
23
     };
```

```
24 };
```

通过&cpu0 就可以访问"cpu@0"这个节点,而不需要输入完整的节点名字。再比如节 点"intc: interrupt-controller@00a01000",节点 label 是 intc,而节点名字就很长了,为 "interrupt-controller@00a01000"。很明显通过&intc 来访问"interrupt-controller@00a01000" 这个节点要方便很多!

所以如果我们要在设备树中引用其它的节点,那么就可以在这个被引用的节点前加上 "label:",这样我们就可以很方便的通过"&label"的方式进行引用了。

24.3.7 向节点追加或修改内容

这里面有两个知识点:向节点追加内容,也就是添加属性;另一个就是修改节点的内容。 我相信大家都理解我这里说的意思。在实际的开发当中肯定是有这样的需求存在的,例如在 我们的领航者开发板上有一个 eeprom 器件(24c64)和一个 rtc 器件(pcf8563),假如它俩都 是挂在 ZYNQ 的 i2c0 总线下的。那么现在要把这两个设备添加到 i2c0 总线下,打开 zynq-7000.dtsi 文件,可以看到 PS 的两组 i2c 控制器节点定义,如下所示:

示例代码 24.3.22 zynq-7000.dtsi i2c 节点

```
122 i2c0: i2c@e0004000 {
```

- 123 compatible = "cdns,i2c-r1p10";
- 124 status = "disabled";
- 125 clocks = <&clkc 38>;
- 126 interrupt-parent = <&intc>;
- 127 interrupts = <0 25 4>;
- 128 $reg = \langle 0xe0004000 \ 0x1000 \rangle;$
- 129 #address-cells = <1>;


原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

```
130 #size-cells = <0>;
131 };
132
133 i2c1: i2c@e0005000 {
134 compatible = "cdns,i2c-r1p10";
135 status = "disabled";
136 clocks = <&clkc 39>;
137 interrupt-parent = <&intc>;
138 interrupts = <0 48 4>;
139 reg = <0xe0005000 0x1000>;
140 #address-cells = <1>;
```

141 #size-cells = <0>;

142 };

因为现在要把开发板的两个 i2c 器件添加到 i2c0 总线下,直接在 i2c0 节点下创建两个子 节点即可,一个子节点对应的是 eeprom,另一个子节点对应的是 rtc,那么最简单的方法就是 直接在 zynq-7000.dtsi 文件的 i2c0 节点中添加这两个节点子节点即可,如下所示:

示例代码 24.3.23 zynq-7000.dtsi 添加 i2c 器件

```
122 i2c0: i2c@e0004000 {
```

```
123 compatible = "cdns,i2c-r1p10";
```

```
124 status = "disabled";
```

```
125 clocks = <&clkc 38>;
```

```
126 interrupt-parent = <&intc>;
```

```
127 interrupts = <0 25 4>;
```

```
128 reg = \langle 0xe0004000 \ 0x1000 \rangle;
```

```
129 #address-cells = <1>;
```

```
130 \#size-cells = \langle 0 \rangle;
```

```
132 24c64@50 {
```

```
133 compatible = "atmel,24c64";
```

```
134 reg = <0x50>;
```

```
135 pagesize = <32>;
```

```
136 };
```

131

```
137
```

```
138 rtc@51 {
```

```
139 compatible = "nxp,pcf8563";
```

```
140 reg = <0x51>;
```

```
141 };
```

```
142 };
143
```

```
144 i2c1: i2c@e0005000 {
```

```
145 compatible = "cdns,i2c-r1p10";
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

- 146 status = "disabled";
- 147 clocks = <&clkc 39>;
- 148 interrupt-parent = <&intc>;
- 149 interrupts = <0 48 4>;
- 150 $reg = \langle 0xe0005000 \ 0x1000 \rangle;$
- 151 #address-cells = <1>;
- 152 #size-cells = <0>;

153 };

第132~136行就是在i2c0总线下添加了 eeprom 设备,138~141 行添加了 rtc 设备(注意: 我这里只是给大家做演示,你们不要去改这个文件);但是这样会有个问题,i2c0 节点是定 义在 zynq-7000.dtsi 文件中的,而 zynq-7000.dtsi 是设备树头文件,前面也跟大家说到过,该文 件是 zynq-7000 系列处理器的一个通用设备树头文件,也就是说它是会被其他 dts 文件所包含 的,直接在 i2c0 节点中添加这两个子节点就相当于在所有的 zynq-7000 系列处理器开发板上 都添加了这两个设备,如果其他的板子并没有这两个设备呢!因此,按照示例代码 24.3.23 这 样写肯定是不行的。

这里就要引入另外一个内容,那就是向节点追加数据,我们现在要解决的就是如何向 i2c0节点追加两个子节点,而且不能影响到其它使用 zynq-7000 系列处理器的开发板。在本篇 中我们使用的设备树文件为 system-top.dts,因此我们需要在 system-top.dts 文件中完成数据追 加的内容,方式如下:

示例代码 24.3.24 节点追加数据方法

1 &i2c0 {

```
2 /* 要追加或修改的内容 */
```

3 };

第1行, &i2c0表示要引用到 i2c0 这个 label 所对应的节点,也就是 zynq-7000.dtsi 文件中的 "i2c0: i2c@e0004000"。

第2行,花括号内就是要向i2c0这个节点添加的内容,包括修改某些属性的值。

```
打开 system-top.dts,这样我们就可以直接在该文件中追加内容了:
```

```
示例代码 24.3.25 system-top.dts 向 i2c0 节点追加内容
```

```
8 /dts-v1/;
9 #include "zynq-7000.dtsi"
10 #include "pl.dtsi"
11 #include "pcw.dtsi"
12/{
13
     model = "Alientek ZYNQ Development Board";
14
15
     chosen {
       bootargs = "console=ttyPS0,115200 earlyprintk root=/dev/mmcblk0p2 rw rootwait";
16
       stdout-path = "serial0:115200n8";
17
18
    };
19
     aliases {
```

```
20 ethernet0 = \& \text{gem}0;
```



原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php 21 $i2c0 = \&i2c_2;$ 22 i2c1 = &i2c0;i2c2 = &i2c1;23 24 serial0 = &uart0;25 serial1 = &uart1; spi0 = &qspi;26 27 }; 28 memory { 29 device_type = "memory"; $reg = \langle 0x0 \ 0x20000000 \rangle;$ 30 31 }; 32 }; 33 34 &i2c0 { clock-frequency = <100000>; 35 status = "okay"; 36 37 38 24c64@50 { 39 compatible = "atmel,24c64"; 40 reg = <0x50>;pagesize = <32>; 41 42 }; 43 44 rtc@51 { 45 compatible = "nxp,pcf8563"; 46 reg = <0x51>;47 }; 48 }; 49 50 & gem0 { 51 local-mac-address = $[00 \ 0a \ 35 \ 00 \ 1e \ 53];$ 52 }; 第 34~48 行就是向 i2c0 节点添加/修改数据,比如 35 的属性 "clock-frequency = <100000>"

就表示将 i2c0 的时钟设置为 100KHz, "clock-frequency"就是新添加的属性。 第 36 行,将 status 属性的值由原来的 disabled 改为 okay,这是修改节点的属性值。 第 38~47 行,我们向 i2c0 子节点追加了两个子节点,"24c64@50"和"rtc@51"。 除此之外,第 12~32 行,其实就是向 zynq-7000.dtsi 中定义的根节点中追加了一些节点。 注意,这里只是给大家演示,大家不要去修改这些文件,后面用到的时候我会再说!!! 因为示例代码 24.3.25 中的内容是 system-top.dts 这个文件内的,所以不会对使用 ZYNQ-7000 系列处理器的其它板子造成任何影响。这个就是向节点追加或修改内容,重点就是通过



原子哥在线教学:www.yuanzige.com 论坛:www.openedv.com/forum.php &label 来访问节点,然后直接在里面编写要追加或者修改的内容。例如在 pcw.dtsi 文件中,可 以看到很多的节点引用、向节点追加内容、修改节点内容的示例,如下所示:

<pre>19 phy-mode = "rgmii-id";</pre>
<pre>20 status = "okay";</pre>
<pre>21 xlnx,ptp-enet-clock = <0x69f6bcb>;</pre>
22 };
23 &gpio0 {
<pre>24 emio-gpio-width = <13>;</pre>
<pre>25 gpio-mask-high = <0x0>;</pre>
<pre>26 gpio-mask-low = <0x5600>;</pre>
27 };
28 &i2c0 {
<pre>29 clock-frequency = <400000>;</pre>
30 status = "okay";
31 };
32 &i2c1 {
<pre>33 clock-frequency = <400000>;</pre>
34 status = "okay";
35 };
36 &intc {
37 num_cpus = <2>;
<pre>38 num_interrupts = <96>;</pre>
39 };
40 &qspi {
41 is-dual = <0>;
42 num-cs = <1>;
<pre>43 spi-rx-bus-width = <4>;</pre>
<pre>44 spi-tx-bus-width = <4>;</pre>
45 status = "okay";
46 };

图 24.3.5 pcw.dtsi 示例

24.3.8 特殊节点

在根节点"/"中有那么几个特殊的子节点: aliases、chosen 以及 memory,我们接下来看一下这三个比较特殊的节点,我们会发现这三个节点都是没有 compatible 属性,也就是说它们对应的是特殊设备。

1、aliases 节点

打开 system-top.dts 文件,可以看到 aliases 节点的内容如下所示:

示例代码 24.3.26 system-top.dts aliases 节点

```
19 aliases {
```

- 20 ethernet0 = & gem0;
- 21 i2c0 = &i2c_2;
- 22 i2c1 = &i2c0;
- 23 i2c2 = &i2c1;
- serial0 = &uart0;
- 25 serial1 = &uart1;

```
26 spi0 = &qspi;
```

```
27 };
```

单词 aliases 的意思是"别名",因此 aliases 节点的主要功能就是定义别名,定义别名的 目的就是为了方便访问节点。但是需要注意的是,这里说的方便访问节点并不是在设备树中 访问节点,例如前面说到的使用"&label"的方式访问设备树中的节点,而是内核当中方便 定位节点,例如在内核中通过 ethernet0 就可以定位到 gem0 节点(&gem0 引用的节点),再 例如内核通过 serial0 就可以找到 uart0 节点。

2、chosen 节点



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

chosen 节点一般会有两个属性, "bootargs" 和 "stdout-path"。打开 system-conf.dtsi 文 件,找到 chosen 节点,内容如下所示:

示例代码 24.3.27 chosen 节点

15 chosen {

16 bootargs = "console=ttyPS0,115200 earlycon root=/dev/mmcblk0p2 rw rootwait";

stdout-path = "serial0:115200n8"; 17

18 };

在 chosen 节点当中, 属性 stdout-path = "serial0:115200n8", 表示标准输出设备使用串口 serial0, 在 system-top.dts 文件当中, serial0 其实是一个别名, 指向的就是 uart0; "115200" 则表示串口的波特率为 115200, "n"表示无校验位, "8"则表示有 8 位数据位, 相信大家 都明白这些是什么意思。

当你看到 chosen 节点中的 bootargs 属性的时候有没有想到 U-Boot 的 bootargs 环境变量呢? 内核的 bootargs 参数不是由 U-Boot 传给它的吗?为什么要在内核设备树根节点下的 chosen 节 点中定义呢?他们俩有什么区别呢?那么关于这些问题稍后再给大家解释,这里大家想思考 另一个问题: "stdout-path"属性指定了标准输出设备,而 bootargs 参数当中也指定了标准输 出设备(console=ttyPS0,115200, ttyPS0 其实指的就是根文件系统下的/dev/ttyPS0 这个设备文 件,那么它对应的硬件设备其实就是板子的 uart0),那么内核在初始化标准输出设备的时候 到底听谁的呢? 关于这个问题,笔者开始也想不明白,于是呼去内核源码中找了找,在内核 源码 drivers/of/base.c 文件中看到了下面这段代码:

示例代码 24.3.28 of console check 函数

```
1822 /**
```

1823 * of_console_check() - Test and setup console for DT setup

1824 * @dn - Pointer to device node

1825 * @name - Name to use for preferred console without index. ex. "ttyS"

1826 * @index - Index to use for preferred console.

1827 *

1828 * Check if the given device node matches the stdout-path property in the

1829 * /chosen node. If it does then register it as the preferred console and return

1830 * TRUE. Otherwise return FALSE.

```
1831 */
```

1832 bool of_console_check(struct device_node *dn, char *name, int index) 1833 {

```
1834
       if (!dn || dn != of_stdout || console_set_on_cmdline)
```

```
return false;
1835
```

1836

```
1837 /*
```

* XXX: cast `options' to char pointer to suppress complication 1838

* warnings: printk, UART and console drivers expect char pointer. 1839

1840 */

1841 return !add_preferred_console(name, index, (char *)of_stdout_options);

1842



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

看这个函数的名字"of_console_check",意思是控制台校验(控制台大家可以理解为 linux 的标准输入、输出终端),第 1834 行当中的 of_stdout 其实是内核解析 stdout-path = "serial0:115200n8"时得到的 serial0 指向的设备节点,也就是我们的串口 0;而 console_set_on_cmdline 是一个 int 类型的变量,如果 bootargs 字符串当中指定了 console=xxxxx,那么内核也会解析到,并且将 console_set_on_cmdline 变量设置为 1;所以根据代码中的第 1834 行以及函数定义前面的注释信息,我的猜想如下:

在 of_console_check 函数中会判断设备树 stdout-path 属性是否定义了,如果定义了则它拥 有优先级。

当然这是我的猜测,我并没有去验证,不想花这个时间去研究了,如果大家有时间可以 去找找看,这里就不说这个问题了。

现在给大家解释前面说到的那些问题:内核的 bootargs 参数不是由 U-Boot 传给它的吗? 为什么还要在内核设备树根节点下的 chosen 节点中定义 bootargs 呢?他们俩有什么区别呢? 下面给大家一一解释一下。

前面讲解 uboot 的时候说过, uboot 在启动 Linux 内核的时候会将 bootargs 的值传递给 Linux 内核, bootargs 会作为 Linux 内核的命令行参数, Linux 内核启动的时候会打印出命令行参数(也就是 uboot 传递进来的 bootargs 的值), 如下图所示:



图 24.3.6 内核启动打印命令行参数

但是我们使用的这个 U-Boot, 它的环境变量当中并没有定义 bootargs 变量,大家可以进入 U-Boot 命令行,通过 print 命令打印出所有的环境变量,你会发现并没有定义 bootargs,那 这跟我们前面说的不相符了呀,而事实并不如此。

在 uboot 源码中全局搜索 "chosen"这个字符串,看看能不能找到一些蛛丝马迹,果然在 U-Boot 源码目录的 common/fdt_support.c 文件中有个 fdt_chosen 函数,此函数内容如下所示:

示例代码 24.3.29 uboot 源码中的 fdt_chosen 函数

```
275 int fdt_chosen(void *fdt)
276 {
277    int nodeoffset;
278    int err;
279    char *str;    /* used to set string properties */
280
281    err = fdt_check_header(fdt);
282    if (err < 0) {</pre>
```



原子哥在	送教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
283	<pre>printf("fdt_chosen: %s\n", fdt_strerror(err));</pre>
284	return err;
285	}
286	
287	/* find or create "/chosen" node. */
288	<pre>nodeoffset = fdt_find_or_add_subnode(fdt, 0, "chosen");</pre>
289	if (nodeoffset < 0)
290	return nodeoffset;
291	
292	<pre>str = getenv("bootargs");</pre>
293	if (str) {
294	err = fdt_setprop(fdt, nodeoffset, "bootargs", str,
295	strlen(str) + 1);
296	if (err < 0) {
297	printf("WARNING: could not set bootargs %s.\n",
298	fdt_strerror(err));
299	return err;
300	}
301	}
302	
303	<pre>return fdt_fixup_stdout(fdt, nodeoffset);</pre>
304 }	

第 288 行,调用函数 fdt_find_or_add_subnode 从内核设备树(.dtb,因为此时内核 dtb 文件 已经被拷贝到 DDR 中了)中找到 chosen 节点,如果没有找到的话就会自己创建一个 chosen 节点。

第292行,读取 uboot 中 bootargs 环境变量的内容。

第 293 行,判断如果读取 bootargs 环境变量成功,则执行 if { }中的代码。

第 294 行,调用函数 fdt_setprop 向内核设备的 chosen 节点添加 bootargs 属性,并且 bootargs 属性的值就是环境变量 bootargs 的内容。(因为此时内核 dtb 文件已经被拷贝到 DDR 中了,U-Boot 可以通过内核设备树 dtb 的起始地址对 dtb 数据进行修改)。

所以从上面这段代码可以看出来,如果 U-Boot 定义了 bootargs 环境变量,则会通过 fdt_setprop 函数在内核设备树的 chosen 节点追加 bootargs 属性,它的值就是 U-Boot 环境变量 bootargs 的值,如果是这样,那么内核设备树 chosen 节点的 bootargs 属性就会被修改。但是对 于我们使用这个 U-Boot 来说,它并没有定义 bootargs 环境变量,所以使用的就是内核设备树 chosen 节点下的 bootargs 属性,也就是说 U-Boot 的环境变量 bootargs 拥有最高的优先级。

接下来我们顺着 fdt_chosen 函数一点点的抽丝剥茧,看看都有哪些函数调用了 fdt_chosen, 一直找到最终的源头。这里我就不卖关子了,直接告诉大家整个流程是怎么样的,如下图所 示:



图 24.3.7 fdt_chosen 函数调用流程

上图中框起来的部分就是函数 do_bootm_linux 函数的执行流程,也就是说 do_bootm_linux 函数会通过一系列复杂的调用,最终通过 fdt_chosen 函数在内核设备树 chosen 节点中添加 bootargs 属性。而 U-Boot 的 bootcmd 命令最终会执行 bootz 命令,而 bootz 命令启动 Linux 内核的时候会运行 do_bootm_linux 函数,至此,真相大白!

3、memory 节点

memory 节点看名字就知道跟内存是有关系的,如下所示:示例代码 24.3.30 memory 节点

```
28 memory {
29 device_type = "memory";
30 reg = <0x0 0x20000000>;
31 };
```

memory 节点描述了系统内存的基地址以及系统内存大小, "reg = <0x0 0x20000000>"就 表示系统内存的起始地址为 0x0,大小为 0x20000000,也就是 512MB,该节点一般只有这两 个属性,device_type 属性的值固定为"memory"。

24.3.9 常用节点

本来这小节给大家讲一些常用到的节点,例如中断控制器、GPIO 控制器以及在节点当中 如何使用中断、如何使用 gpio 等。当想了想还是放在后面我们用到的时候再给大家介绍。

24.4 驱动与设备节点的匹配

这部分内容已经在前面跟大家讲过了,具体请看 24.3.5 小节中的第一个小点 compatible 属性介绍。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

24.5 内核启动过程中解析设备树

Linux 内核在启动的时候会解析内核 DTB 文件,然后在根文件系统的/proc/device-tree (后面给大家演示)目录下生成相应的设备树节点文件。接下来我们简单分析一下 Linux 内核是如何解析 DTB 文件的,流程如下图所示:



图 24.5.1 设备树中节点解析流程

从上图中可以看出,在 start_kernel 函数中完成了设备树节点解析的工作,最终实际工作的函数为 unflatten_dt_node。那么具体如何进行设备树解析的这里就不给大家进行一一分析了,如果大家有时间可以自个去研究研究!

24.6 设备树在系统中的体现

Linux 内核启动的时候会解析设备树中各个节点的信息,并且在根文件系统的/proc/device-tree 目录下根据节点名字创建不同文件夹,如下图所示:

root@ALIENTEK-Z	YNQ:/proc/device	-tree#					
root@ALIENTEK-Z	root@ALIENTEK-ZYNQ:/proc/device-tree# pwd						
/proc/device-tr	ee						
root@ALIENTEK-Z	YNQ:/proc/device	-tree#					
root@ALIENTEK-Z	YNQ:/proc/device	-tree# ls					
#address-cells	aliases	amba_pl	compatible	fixedregulator	memory	name	
#size-cells	amba	chosen	cpus	fpga-full	model	pmu@f8891000	
root@ALIENTEK-Z	YNQ:/proc/device	-tree#					
root@ALIENTEK-Z	YNQ:/proc/device	-tree#					
root@ALIENTEK-Z	YNQ:/proc/device	-tree#					

图 24.6.1 根节点的属性以及子节点

上图列出来就是目录/proc/device-tree 目录下的内容,/proc/device-tree 目录下是根节点"/"的所有属性和子节点,我们依次来看一下这些属性和子节点。

1、根节点"/"各个属性

在图 24.6.1 中,根节点下的属性表现为一个个的文件(大家可以用 ls -1 查看到文件的类型),比如图 24.6.1 中的"#address-cells"、"#size-cells"、"compatible"、"model"和"name"这 5 个文件,它们在设备树中就是根节点的 5 个属性。既然是文件那么肯定可以查看其内容,输入 cat 命令来查看 model 和 compatible 这两个文件的内容,结果如下图所示:

原子哥在线教学: www.yuanzige.com 论坛:www.openedy.com/forum.php

			-		
root@ALIENTEK-	ZYNQ:/proc/device-	tree#			
root@ALIENTEK-	ZYNQ:/proc/device-	tree# cat model			
Alientek ZYNQ	Development Board	oot@ALIENTEK-ZYNQ:/j	proc/device-t	ree#	
root@ALIENTEK-	ZYNQ:/proc/device-	tree#			
root@ALIENTEK-	ZYNQ:/proc/device-	tree#			
root@ALIENTEK-	ZYNQ:/proc/device-	tree# cat compatible	e		
xlnx,zynq-7000	coot@ALIENTEK-ZYNQ	:/proc/device-tree#			
root@ALIENTEK-	ZYNQ:/proc/device-	tree#			
root@ALTENTEK-	ZYNO:/proc/device-	tree#			

图 24.6.2 model 和 compatible 文件内容

从图 24.6.2 可以看出,文件 model 的内容是"Alientek ZYNQ Development Board",文件 compatible 的内容为"xlnx,zynq-7000"。这跟 system-user.dtsi 文件根节点的 model 属性值、以 及 zynq-7000.dtsi 文件根节点的 compatible 属性值是完全一样的。

2、根节点"/"各子节点

图 24.6.1 中列出的各个文件夹就是根节点"/"的各个子节点,比如"aliases"、"cpus"、 "chosen"和"amba"等等。大家可以查看我们用到的设备树文件,看看根节点的子节点都 有哪些,看看是否和图 24.6.1 中的一致。

/proc/device-tree 目录就是设备树在根文件系统中的体现,同样是按照树形结构组织的,进入/proc/device-tree/amba目录中就可以看到 amba 节点的所有子节点,如所示:

root@ALIENTEK-ZYNQ:/proc/dev	vice-tree#			
root@ALIENTEK-ZYNQ:/proc/dev	vice-tree# cd amba			
root@ALIENTEK-ZYNQ:/proc/dev	vice-tree/amba# ls			
#address-cells	devcfg@f8007000	i2c@e0005000	name	S
#size-cells	dmac@f8003000	interrupt-controller@f8f01000	ocmc@f800c000	S
adc@f8007100	efuse@f800d000	interrupt-parent	ranges	t:
cache-controller@f8f02000	ethernet@e000b000	memory-controller@e000e000	serial@e0000000	t:
can@e0008000	ethernet@e000c000	memory-controller@f8006000	serial@e0001000	t:
can@e0009000	gpio@e000a000	mmc@e0100000	slcr@f8000000	t:
compatible	i2c@e0004000	mmc@e0101000	spi@e0006000	u
root@ALIENTEK-ZYNQ:/proc/dev	vice-tree/amba#			
root@ALIENTEK-ZYNQ:/proc/dev	vice-tree/amba#			

图 24.6.3 amba 节点的所有属性和子节点

和根节点"/"一样,图 24.6.3 中的所有文件分别为 amba 节点的属性文件和子节点文件 夹。大家可以自行查看一下这些属性文件的内容是否和我们使用的设备树中 amba 节点的属性 值相同。

24.7 绑定信息文档

设备树是用来描述板子上的硬件设备信息的,不同的设备其信息不同,反映到设备树中就是属性不同。那么我们在设备树中添加一个硬件对应的节点的时候从哪里查阅相关的说明呢?在Linux内核源码中有详细的.txt文档描述了如何添加节点,这些.txt文档叫做绑定文档,路径为:Linux源码目录/Documentation/devicetree/bindings,如所示:



原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php

📕 🔽 📕 🖛	bindings									-	×
文件 主页	共享	查看									~ ?
★ 复制 适定到"快 复制 速访问"		剪切 复制路径 粘贴快捷方式	秋 动到 复制到	★ ■ ● <th>□ 1 新建项目 * ○ 1 轻松访问 * ○ 新建 文件夹</th> <th>属性</th> <th>↓ 打开 ▼ 】 编辑 ◎ 历史记录</th> <th></th> <th>全部选择 全部取消 反向选择</th> <th></th> <th></th>	□ 1 新建项目 * ○ 1 轻松访问 * ○ 新建 文件夹	属性	↓ 打开 ▼ 】 编辑 ◎ 历史记录		全部选择 全部取消 反向选择		
	剪贴板		组织	Ŗ	新建	Ŧ	J开		选择		
$\leftarrow \ \rightarrow \ \checkmark \ \uparrow$	📜 « Do	cumentation >	devicetree > bi	ndings			~	ري ا	搜索"bin	dings"	Ą
	^	名称	^		修改日期	类	型		大小		^
▼ 快速切回		📕 arc			2019-06-03 15:0)7 文	件夹				
山本 一世		📜 arm			2019-06-03 15:0)7 文	件夹				
◆ 11転 		📜 ata			2019-06-03 15:0)6 文	件夹				
■ 又档	*	📜 bus			2019-06-03 15:0)7 文	件夹				
▶ 图片	*	📜 с6х			2019-06-03 15:0)6 文	件夹分	门别	类存放的	绑定	
📜 第15讲 Li	inux C!	📜 clock			2019-06-03 15:0)6 文	件夹 文	档			
📜 第16讲 m	nake⊥	📜 cpufreq			2019-06-03 15:0)7 文	件夹				
📜 开发手册		📜 cris			2019-06-03 15:0	06 文	件夹				
1 工长主即。	vico Rel 🗡	📜 crypto			2019-06-03 15:0)6 文	件夹				~
88 个项目											

图 24.7.1 绑定文档

比如我们现在要想在 ZYNO 7010/7020 这颗 SOC 的 I2C 下添加一个节点,那么就可以查 看 Documentation/devicetree/bindings/i2c/i2c-cadence.txt(文件的名字一般都是以 i2c-xxx.txt 命 名的, xxx 一般是制造商), 此文档详细的描述了 ZYNQ-7000 系列处理器如何在设备树中添 加 I2C 设备节点, 文档内容如下所示:

Binding for the Cadence I2C controller

Required properties:

- reg: Physical base address and size of the controller's register area.
- compatible: Should contain one of:
 - * "cdns,i2c-r1p10"
 - Note: Use this when cadence i2c controller version 1.0 is used.
 - * "cdns,i2c-r1p14"

Note: Use this when cadence i2c controller version 1.4 is used.

- clocks: Input clock specifier. Refer to common clock bindings.
- interrupts: Interrupt specifier. Refer to interrupt bindings.
- #address-cells: Should be 1.
- #size-cells: Should be 0.

Optional properties:

- clock-frequency: Desired operating frequency, in Hz, of the bus.
- clock-names: Input clock name, should be 'pclk'.

Example:

```
i2c@e0004000 {
     compatible = "cdns,i2c-r1p10";
     clocks = \langle \&clkc | 38 \rangle;
     interrupts = <GIC_SPI 25 IRQ_TYPE_LEVEL_HIGH>;
```



论坛:www.openedv.com/forum.php

```
reg = <0xe0004000 0x1000>;
clock-frequency = <400000>;
#address-cells = <1>;
#size-cells = <0>;
```

原子哥在线教学: www.yuanzige.com

};

有时候使用的一些芯片在 Documentation/devicetree/bindings 目录下找不到对应的文档,这个时候就要咨询芯片的提供商,让他们给你提供参考的设备树文件。

24.8 设备树常用 of 操作函数

设备树描述了设备的详细信息,这些信息包括数字类型的、字符串类型的、数组类型的, 我们在编写驱动的时候需要获取到这些信息。比如设备树使用 reg 属性描述了某个外设的寄存 器地址为 0X02005482,长度为 0X400,我们在编写驱动的时候需要获取到 reg 属性的 0X02005482 和 0X400 这两个值,然后初始化外设。Linux 内核给我们提供了一系列的函数来 获取设备树中的节点或者属性信息,这一系列的函数都有一个统一的前缀"of_",所以在很 多资料里面也被叫做 OF 函数。这些 OF 函数原型都定义在 include/linux/of.h 文件中。

24.8.1 查找节点的 OF 函数

设备都是以节点的形式"挂"到设备树上的,因此要想获取这个设备的属性信息,必须 先获取到这个设备的节点。Linux 内核使用 device_node 结构体来描述一个节点,此结构体定 义在文件 include/linux/of.h 中,定义如下:

示例代码 24.8.1 device node 节点 49 struct device_node { 50 const char *name; /* 节点名字 */ /* 设备类型 */ 51 const char *type; 52 phandle phandle; const char *full name; /* 节点全名 */ 53 struct fwnode handle fwnode; 54 55 56 struct property *properties; /* 属性 */ /* removed 属性 */ 57 struct property *deadprops; struct device node *parent; /* 父节点 */ 58 /* 子节点 */ struct device_node *child; 59 struct device_node *sibling; 60 struct kobject kobj; 61 unsigned long _flags; 62 63 void *data; 64 #if defined(CONFIG_SPARC) 65 const char *path_component_name; 66 unsigned int unique_id; struct of_irq_controller *irq_trans; 67

68 #endif

发指南 砂正点原子 论坛:www.openedv.com/forum.php

69 };

与查找节点有关的 OF 函数有 5 个,我们依次来看一下。

1、of_find_node_by_name 函数

原子哥在线教学: www.yuanzige.com

of_find_node_by_name 函数通过节点名字查找指定的节点,函数原型如下: struct device_node *of_find_node_by_name(struct device_node *from,

const char *name);

函数参数和返回值含义如下:

from:开始查找的节点,如果为 NULL 表示从根节点开始查找整个设备树。 name:要查找的节点名字。

返回值:找到的节点,如果为 NULL 表示查找失败。

2、of_find_node_by_type 函数

of_find_node_by_type 函数通过 device_type 属性查找指定的节点,函数原型如下: struct device_node *of_find_node_by_type(struct device_node *from, const char *type) 函数参数和返回值含义如下: from: 开始查找的节点,如果为 NULL 表示从根节点开始查找整个设备树。 type: 要查找的节点对应的 type 字符串,也就是 device_type 属性值。 返回值: 找到的节点,如果为 NULL 表示查找失败。

3、of_find_compatible_node 函数

of_find_compatible_node 函数根据 device_type 和 compatible 这两个属性查找指定的节点, 函数原型如下:

struct device_node *of_find_compatible_node(struct device_node *from,

const char

const char *compatible)

*type,

函数参数和返回值含义如下:

from:开始查找的节点,如果为 NULL 表示从根节点开始查找整个设备树。

type:要查找的节点对应的 type 字符串,也就是 device_type 属性值,可以为 NULL,表示忽略掉 device_type 属性。

compatible: 要查找的节点所对应的 compatible 属性列表。

返回值:找到的节点,如果为 NULL 表示查找失败

4、of_find_matching_node_and_match 函数

of_find_matching_node_and_match 函数通过 of_device_id 匹配表来查找指定的节点,函数 原型如下:

struct device_node *of_find_matching_node_and_match(struct device_node *from,

const struct of_device_id *matches,

const struct of_device_id **match)

函数参数和返回值含义如下:

from:开始查找的节点,如果为 NULL 表示从根节点开始查找整个设备树。 **matches**: of_device_id 匹配表,也就是在此匹配表里面查找节点。 **match**:找到的匹配的 of_device_id。



原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php 返回值:找到的节点,如果为 NULL 表示查找失败

5、of_find_node_by_path 函数

of_find_node_by_path 函数通过节点路径来查找指定的节点,函数原型如下:

inline struct device_node *of_find_node_by_path(const char *path)

函数参数和返回值含义如下:

path:带有全路径的节点名,可以使用节点的别名(用 aliens 节点中定义的别名)。 **返回值**:找到的节点,如果为 NULL 表示查找失败

24.8.2 查找父/子节点的 OF 函数

Linux 内核提供了几个查找节点对应的父节点或子节点的 OF 函数,我们依次来看一下。

1、of get parent 函数

of_get_parent 函数用于获取指定节点的父节点(如果有父节点的话), 函数原型如下: struct device_node *of_get_parent(const struct device_node *node) 函数参数和返回值含义如下: node: 要查找的父节点的节点。 返回值:找到的父节点。

2、of_get_next_child 函数

of_get_next_child 函数用迭代的查找子节点,函数原型如下: struct device_node *of_get_next_child(const struct device_node *node, *prev)

struct device_node

函数参数和返回值含义如下:

node: 父节点。

prev: 前一个子节点, 也就是从哪一个子节点开始迭代的查找下一个子节点。可以设置 为NULL,表示从第一个子节点开始。

返回值:找到的下一个子节点。

24.8.3 提取属性值的 OF 函数

设备树节点的属性保存了驱动所需要的内容,因此对于属性值的提取非常重要,Linux内 核中使用结构体 property 表示属性,此结构体同样定义在文件 include/linux/of.h 中,内容如下:

示例代码 24.8.2 property 结构体

35 s	truct p	roper	ty {						
36	char	*nar	ne;		/*	属性	名字	*/	
37	int ler	ngth;		/*	属性	长度	*/		
38	v	void	*va	lue;	/*	属性	值 */		
39	struct	prop	erty	*next;		/*	ᡯᢇ╯	个属性	*/
40	unsig	ned lo	ong_	_flags;					
41	unsig	ned ir	nt un	ique_i	1;				
42	struct	bin_a	attrił	oute att	r;				
43 }	:								



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php Linux 内核也提供了提取属性值的 OF 函数,我们依次来看一下。

1、of_find_property 函数

of_find_property 函数用于查找指定的属性,函数原型如下:

property *of_find_property(const struct device_node *np,

const char

函数参数和返回值含义如下: np:设备节点。 name:属性名字。 lenp:属性值的字节数 返回值:找到的属性。

2、of_property_count_elems_of_size 函数

int

of_property_count_elems_of_size 函数用于获取属性中元素的数量,比如 reg 属性值是一个数组,那么使用此函数可以获取到这个数组的大小,此函数原型如下:

*name.

*lenp)

int of_property_count_elems_of_size(const struct device_node *np,

const char	*propname,	
int	elem_size)	

函数参数和返回值含义如下: np:设备节点。 proname: 需要统计元素数量的属性名字。 elem_size: 元素长度。 返回值:得到的属性元素数量。

3、of_property_read_u32_index 函数

of_property_read_u32_index 函数用于从属性中获取指定下标(属性值是一个u32数据组成的数组)的u32 类型数据值(无符号 32 位),比如某个属性有多个u32 类型的值,那么就可以使用此函数来获取指定下标的数据值,此函数原型如下:

int of_property_read_u32_index(const struct device_node *np,

	const char	*propname,
	u32	index,
	u32	*out_value)
函数参数利	口返回值含义如下:	
np: 设备 [†]	力点。	
proname:	要读取的属性名字。	

index: 要读取的值的下标。

out_value: 读取到的值

返回值:0 读取成功,负值,读取失败,-EINVAL 表示属性不存在,-ENODATA 表示没有要读取的数据,-EOVERFLOW 表示属性值列表太小。

```
4、of_property_read_u8_array 函数
```

```
of_property_read_u16_array 函数
```

of_property_read_u32_array 函数



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

of_property_read_u64_array 函数

这 4 个函数分别是读取属性中 u8、u16、u32 和 u64 类型的数组数据,比如大多数的 reg 属性都是数组数据,可以使用这 4 个函数一次读取出 reg 属性中的所有数据。这四个函数的原 型如下:

int of_property_read_u8_array(const struct device_node	*np,
const char	*propname,
u8 *out	_values,
size_t	sz)
int of_property_read_u16_array(const struct device_node	*np,
const char	*propname,
u16	*out_values,
size_t	sz)
int of_property_read_u32_array(const struct device_node	*np,
const char *propnan	ne,
u32	*out_values,
size_t sz)	
int of_property_read_u64_array(const struct device_node	*np,
const char *propnan	ne,
u64	*out_values,
size_t sz)	
函数参数和返回值含义如下:	
np : 设备节点。	
proname: 要读取的属性名字。	
out_value:读取到的数组值,分别为u8、u16、	u32和u64。
sz: 要读取的数组元素数量。	
返回值: 0,读取成功,负值,读取失败,-EIN	VAL 表示属性不存在,-ENODATA 表示
没有要读取的数据,-EOVERFLOW 表示属性值列表	太小。
5、of_property_read_u8 函数	
of_property_read_u16 函数	
of_property_read_u32 函数	
of_property_read_u64 函数	
有些属性只有一个整形值,这四个函数就是用	于读取这种只有一个整形值的属性,分别
用于读取 u8、u16、u32 和 u64 类型属性值,函数原	型如下:
int of_property_read_u8(const struct device_node *np,	
const char *pro	pname,
u8 *out_valu	ie)
int of property read u16(const struct device node *np	

int of_property_read_uro(const struct device_node	np,
const char	*propname,
u16	*out_value)
int of_property_read_u32(const struct device_node	*np,
const char	*propname,



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

u32 int of_property_read_u64(const struct device_node

u64

const char

*out_value) *np,

*propname,

*out_value)

函数参数和返回值含义如下:

np:设备节点。

proname: 要读取的属性名字。

out_value: 读取到的数组值。

返回值:0,读取成功,负值,读取失败,-EINVAL 表示属性不存在,-ENODATA 表示 没有要读取的数据,-EOVERFLOW 表示属性值列表太小。

6、of_property_read_string 函数

of_property_read_string 函数用于读取属性中字符串值,函数原型如下: int of_property_read_string(struct device_node *np,

const char*propname,const char**out_string)函数参数和返回值含义如下:

np: 设备节点。

proname: 要读取的属性名字。

out_string: 读取到的字符串值。

返回值: 0, 读取成功, 负值, 读取失败。

7、of_n_addr_cells 函数

of_n_addr_cells 函数用于获取#address-cells 属性值,函数原型如下:

int of_n_addr_cells(struct device_node *np)

函数参数和返回值含义如下:

np:设备节点。

返回值:获取到的#address-cells 属性值。

8、of_n_size_cells 函数

of_size_cells 函数用于获取#size-cells 属性值,函数原型如下:

int of_n_size_cells(struct device_node *np)

函数参数和返回值含义如下:

np:设备节点。

返回值: 获取到的#size-cells 属性值。

24.8.4 其它常用的 OF 函数

1、of_device_is_compatible 函数

of_device_is_compatible 函数用于查看节点的 compatible 属性是否有包含 compat 指定的字符串,也就是检查设备节点的兼容性,函数原型如下:

int of_device_is_compatible(const struct device_node *device,

const char

*compat)



原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php

函数参数和返回值含义如下:

device: 设备节点。

compat: 要查看的字符串。

返回值: 0, 节点的 compatible 属性中不包含 compat 指定的字符串; 正数, 节点的 compatible 属性中包含 compat 指定的字符串。

2、of get address 函数

of get address 函数用于获取地址相关属性,主要是"reg"或者"assigned-addresses"属 性值,函数属性如下:

const __be32 *of_get_address(struct device_node *dev.

> int index, u64 *size, unsigned int *flags)

函数参数和返回值含义如下:

dev: 设备节点。

index: 要读取的地址标号。

size: 地址长度。

flags: 参数,比如 IORESOURCE_IO、IORESOURCE_MEM 等

返回值:读取到的地址数据首地址,为 NULL 的话表示读取失败。

3、of_translate_address 函数

of_translate_address 函数负责将从设备树读取到的地址转换为物理地址,函数原型如下: u64 of_translate_address(struct device_node *dev,

const __be32 *in_addr) 函数参数和返回值含义如下:

dev: 设备节点。

in_addr: 要转换的地址。

返回值:得到的物理地址,如果为 OF BAD ADDR 的话表示转换失败。

4、of_address_to_resource 函数

IIC、SPI、GPIO 等这些外设都有对应的寄存器,这些寄存器其实就是一组内存空间, Linux 内核使用 resource 结构体来描述一段内存空间, "resource"翻译出来就是"资源",因 此用 resource 结构体描述的都是设备资源信息, resource 结构体定义在文件 include/linux/ioport.h 中, 定义如下:

示例代码 24.8.3 resource 结构体

```
18 struct resource {
```

```
19 resource_size_t start;
```

- 20 resource_size_t end;
- 21 const char *name;
- 22 unsigned long flags;
- 23 struct resource *parent, *sibling, *child;
- 24 };



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

源标志

对于 32 位的 SOC 来说, resource_size_t 是 u32 类型的。其中 start 表示开始地址, end 表 示结束地址, name 是这个资源的名字, flags 是资源标志位, 一般表示资源类型, 可选的资源 标志定义在文件 include/linux/ioport.h 中,如下所示:

	示例代码 24.8.4 贷
1 #define IORESOURCE_BITS	0x000000ff
2 #define IORESOURCE_TYPE_BI	TS 0x00001f00
3 #define IORESOURCE_IO	0x00000100
4 #define IORESOURCE_MEM	0x00000200
5 #define IORESOURCE_REG	0x00000300
6 #define IORESOURCE_IRQ	0x00000400
7 #define IORESOURCE_DMA	0x00000800
8 #define IORESOURCE_BUS	0x00001000
9 #define IORESOURCE_PREFETC	CH 0x00002000
10 #define IORESOURCE_READO	NLY 0x00004000
11 #define IORESOURCE_CACHE	ABLE 0x00008000
12 #define IORESOURCE_RANGE	LENGTH 0x00010000
13 #define IORESOURCE_SHADO	WABLE 0x00020000
14 #define IORESOURCE_SIZEALI	GN 0x00040000
15 #define IORESOURCE_STARTA	LIGN 0x00080000
16 #define IORESOURCE_MEM_64	0x00100000
17 #define IORESOURCE_WINDO	W 0x00200000
18 #define IORESOURCE_MUXED	0x00400000
19 #define IORESOURCE_EXCLUS	SIVE 0x08000000
20 #define IORESOURCE_DISABL	ED 0x1000000
21 #define IORESOURCE_UNSET	0x20000000
22 #define IORESOURCE_AUTO	0x40000000
23 #define IORESOURCE_BUSY	0x8000000

大家一般最常见的资源标志就是 IORESOURCE_MEM、IORESOURCE_REG 和 IORESOURCE_IRQ 等。接下来我们回到 of_address_to_resource 函数,此函数看名字像是从设 备树里面提取资源值,但是本质上就是将 reg 属性值,然后将其转换为 resource 结构体类型, 函数原型如下所示

int of_address_to_reso	urce(struct device_n	node *dev,		
	int	index,		
	struct resource	*r)		
函数参数和返回值	ī含义如下:			
dev: 设备节点。				
index: 地址资源标号。				
r:得到的 resource 类型的资源值。				
返回值: 0,成功	; 负值, 失败。			
5、of_iomap 函数				



原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php

of iomap 函数用于直接内存映射,以前我们会通过 ioremap 函数来完成物理地址到虚拟地 址的映射,采用设备树以后就可以直接通过 of_iomap 函数来获取内存地址所对应的虚拟地址, 不需要使用 ioremap 函数了。当然了,你也可以使用 ioremap 函数来完成物理地址到虚拟地址 的内存映射,只是在采用设备树以后,大部分的驱动都使用 of iomap 函数了。of iomap 函数 本质上也是将 reg 属性中地址信息转换为虚拟地址,如果 reg 属性有多段的话,可以通过 index 参数指定要完成内存映射的是哪一段, of iomap 函数原型如下:

void __iomem *of_iomap(struct device_node *np. int

index)

函数参数和返回值含义如下:

np:设备节点。

index: reg 属性中要完成内存映射的段,如果 reg 属性只有一段的话 index 就设置为 0。 返回值:经过内存映射后的虚拟内存首地址,如果为 NULL 的话表示内存映射失败。

常用的 OF 函数就先讲解到这里, Linux 内核中关于设备树的 OF 函数不仅仅只有前面讲 的这几个,还有很多 OF 函数我们并没有讲解,这些没有讲解的 OF 函数要结合具体的驱动, 比如获取中断号的 OF 函数、获取 GPIO 的 OF 函数等等,这些 OF 函数我们在后面的驱动实 验中再详细的讲解。

关于设备树就讲解到这里,关于设备树我们重点要了解以下几点内容:

①、DTS、DTB 和 DTC 之间的区别,如何将.dts 文件编译为.dtb 文件。

②、设备树语法,这个是重点,因为在实际工作中我们是需要修改设备树的。

③、设备树的几个特殊子节点。

④、关于设备树的 OF 操作函数,也是重点,因为设备树最终是被驱动文件所使用的,而 驱动文件必须要读取设备树中的属性信息,比如内存信息、GPIO 信息、中断信息等等。要想 在驱动中读取设备树的属性值,那么就必须使用 Linux 内核提供的众多的 OF 函数。

从下一章开始所有的 Linux 驱动实验都将采用设备树,从最基本的点灯,到复杂的音频、 网络或块设备等驱动。将会带领大家由简入深,深度剖析设备树,最终掌握基于设备树的驱 动开发技能。

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

正点原子

第二十五章 设备树下的 LED 驱动实验

上一章我们详细的讲解了设备树语法以及在驱动开发中常用的 OF 函数,本章我们就开始 第一个基于设备树的 Linux 驱动实验。本章在第二十三章实验的基础上完成,只是将其驱动 开发改为设备树形式而已。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

25.1 设备树 LED 驱动原理

在《第二十三章 新字符设备驱动实验》中,我们直接在驱动文件 newchrled.c 中定义有关 寄存器物理地址,然后使用 io_remap 函数进行内存映射,得到对应的虚拟地址,最后操作寄 存器对应的虚拟地址完成对 GPIO 的初始化。本章我们在第二十三章实验基础上完成,本章我 们使用设备树来向 Linux 内核传递相关的寄存器物理地址, Linux 驱动文件使用上一章讲解的 OF函数从设备树中获取所需的属性值,然后使用获取到的属性值来初始化相关的 IO。本章实 验还是比较简单的,本章实验重点内容如下:

① 在 system-user.dts 文件中创建相应的设备节点。

- ② 编写驱动程序(在第二十三章实验基础上完成),获取设备树中的相关属性值。
- ③ 使用获取到的有关属性值来初始化 LED 所使用的 GPIO 以及初始状态。

25.2 硬件原理图分析

本章实验硬件原理图参考 22.3 小节即可。

25.3 实验程序编写

本实验对应的例程路径为:开发板资料盘(A 盘)\4_SourceCode\3_Embedded_Linux\Linux 驱动例程\4 dtsled。

本章实验在第二十三章实验的基础上完成,重点是将驱动改为基于设备树形式。

25.3.1 修改设备树文件

打开 linux 内核源码目录下的 arch/arm/boot/dts/system-user.dtsi 文件,在根节点"/"下创 建一个名为"led"的子节点, led 节点内容如下所示:

```
示例代码 增加 led 节点
```

```
. . . . . .
   7 / {
   8 model = "Alientek Navigator Zynq Development Board";
       compatible = "xlnx, zynq-zc702", "xlnx, zynq-7000";
   9
   10
   11 chosen {
              bootargs = "console=ttyPS0,115200 earlycon root=/dev/mmcblk0p2 rw
   12
rootwait";
           stdout-path = "serial0:115200n8";
   13
   14 };
   15
   16
          led{
   17
          compatible="alientek,led";
   18
           status="okay";
   19
           default-state="on";
   20
   21
         reg = <0 \times e000 a 0 4 0 0 \times 4
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

22	0xe000a204	0x4
23	0xe000a208	0x4
24	0xe000a214	0x4
25	0xf800012c	0x4>;
26 };		
27 };		

.

第16~26行,在根节点下定义了一个 led 子节点。

第17行,添加 compatible 属性,并将属性值设置为"alientek,led"。

第18行,添加 status 属性,并将属性值设置为"okay"。

第19行,添加 default-state 属性,并将属性值设置为"on"。

第 21~25 行,添加 reg 属性,非常重要! reg 属性设置了驱动里面所要使用的寄存器物理 地址,比如第 21 行的"0xE000A040 0x04"表示 ZYNQ的 GPIO 模块的寄存器 DATA 寄存器, 其中寄存器首地址为 0xE000A040,长度为 4 个字节;第 22 行表示 DIRM 寄存器的首地址为 0xE000A204,长度为 4 个字节;第 23 行表示 OUTEN 寄存器的首地址为 0xE000A208;第 24

行表示 INTDIS 寄存器的首地址为 0xE000A214,长度为 4 个字节;第 25 行表示 APER_CLK_CTRL 寄存器的首地址为 0xF800012C,长度为 4 个字节。

设备树修改完成后保存退出,在内核源码根目录下执行下面这条命令重新编译一下 system-user.dtsi 设备树源文件:

make dtbs

执行结果如下图所示:

<pre>zy@zy-virtual-machine:~/workspace/kernel-driver/linux-xlnx-xlnx_rebase_v5 .4 2020.2\$</pre>
zy@zy-virtual-machine:~/workspace/kernel-driver/linux-xlnx-xlnx_rebase_v5
.4_2020.2\$ make dtbs DTC arch/arm/boot/dts/system-top.dtb
zy@zy-virtual-machine:~/workspace/kernel-driver/linux-xlnx-xlnx_rebase_v5
.4_2020.25 zy@zy-virtual-machine:~/workspace/kernel-driver/linux-xlnx-xlnx_rebase_v5
.4 2020.2\$

图 25.3.1 编译设备树

编译完成以后得到 system-top.dtb 二进制文件,将 system-top.dtb 文件重命名为 system.dtb, 然后拷贝到 SD 启动卡的 FAT 分区替换掉之前的 system.dtb 文件,替换完成之后重新启动开发 板 linux 系统。Linux 启动成功以后进入到/proc/device-tree/目录中查看是否有"led"这个节点,结果如下图所示:



图 25.3.2 led 节点

领航者 ZYNQ 之嵌入式 Linux 开发指南	② 正点原子
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/fo	orum.php
如果没有"led"节点的话请重点注意下面两点:	• •
1、检查设备树修改是否成功,也就是 led 节点是否为根节点	"/"的子节点。
2、检查是否使用的是新的设备树启动 Linux 内核。	
可以进入到图 25.3.2 中的 led 目录中,查看一下都有哪些属性文件	-,结果如下图所示:
root@ALIENTEK-ZYNQ-driver:/proc/device-tree# root@ALIENTEK-ZYNQ-driver:/proc/device-tree# cd led root@ALIENTEK-ZYNQ-driver:/proc/device-tree/led# root@ALIENTEK-ZYNQ-driver:/proc/device-tree/led# ls	
compatible default-state name reg	status
root@ALIENTEK-ZYNQ-driver:/proc/device-tree/led#	
rool@ALIENTEK-ZTNQ-driver:/proc/device-tree/ted#	

图 25.3.3 led 节点下的属性

大家可以用 cat 命令查看一下 compatible、status、default-state 等属性值是否和我们设置的 一致。细心的同学会发现这里边多了一个"name"属性,我们在添加 led 节点的时候并没设 置 name 属性呀,那这是怎么回事呢? name 属性其实也是一个标准属性,**但是现在被弃用了**, 其实不止 led 这个节点多了 name 属性,其它所有的节点也都多了这个属性,但它的值是空的, 这是内核在解析设备树的时候给加上去的,**注意:现在已经不用这个属性了,被弃用了!所 以我们不用管它。**

25.3.2 LED 灯驱动程序编写

设备树准备好以后就可以编写驱动程序了,本章实验在第二十三章实验的驱动文件 newchrled.c 的基础上修改而来。首先在 drivers 目录下新建名为 "4_dtsled" 文件夹,进入到 4_dtsled 目录,新建名为 dtsled.c 的 C 源文件,在 dtsled.c 里面输入如下内容:

```
示例代码 dtsled.c 文件内容
2 Copyright © ALIENTEK Co., Ltd. 1998-2029. All rights reserved.
3 文件名 : dtsled.c
4 作者
       :邓涛
5 版本 : V1.0
6 描述
       : ZYNQ LED 驱动文件。
7 其他 :无
8 论坛
       : www.openedv.com
9 日志
       :初版 V1.0 2019/1/30 邓涛创建
11
12 #include <linux/types.h>
13 #include <linux/kernel.h>
14 #include <linux/delay.h>
15 #include linux/ide.h>
16 #include <linux/init.h>
17 #include <linux/module.h>
18 #include <linux/errno.h>
19 #include <linux/gpio.h>
```



```
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
   20 #include <asm/mach/map.h>
   21 #include <asm/uaccess.h>
   22 #include <asm/io.h>
   23 #include <linux/cdev.h>
   24 #include <linux/of address.h>
   25 #include <linux/of.h>
   26
   27 #define DTSLED_CNT 1 /* 设备号个数 */
   28 #define DTSLED_NAME "dtsled" /* 名字 */
   29
   30 /* 映射后的寄存器虚拟地址指针 */
   31 static void __iomem *data_addr;
   32 static void __iomem *dirm_addr;
   33 static void __iomem *outen_addr;
   34 static void __iomem *intdis_addr;
   35 static void __iomem *aper_clk_ctrl_addr;
   36
   37 /* dtsled 设备结构体 */
   38 struct dtsled_dev {
   39
       dev_t devid;
                      /* 设备号 */
       struct cdev cdev; /* cdev */
   40
   41
        struct class *class; /* 类 */
        struct device *device; /* 设备 */
   42
   43
        int major;
                      /* 主设备号 */
   44
        int minor;
                       /* 次设备号 */
   45
        struct device_node *nd; /* 设备节点 */
   46 };
   47
   48 static struct dtsled_dev dtsled; /* led 设备 */
   49
   50 /*
   51 * @description : 打开设备
   52 * @param – inode : 传递给驱动的 inode
   53 * @param – filp : 设备文件, file 结构体有个叫做 private_data 的成员变量
   54 *
                    一般在 open 的时候将 private_data 指向设备结构体。
   55 * @return
                      :0 成功;其他 失败
   56 */
   57 static int led_open(struct inode *inode, struct file *filp)
   58 {
   59
        filp->private_data = &dtsled; /* 设置私有数据 */
   60
       return 0;
```

```
正点原子
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
   64 * @description :从设备读取数据
   65 * @param - filp : 要打开的设备文件(文件描述符)

      66 * @param – buf
      :返回给用户空间的数据缓冲区

      67 * @param – cnt
      :要读取的数据长度

   68 * @param - offt : 相对于文件首地址的偏移
                    :读取的字节数,如果为负值,表示读取失败
   71 static ssize_t led_read(struct file *filp, char __user *buf,
           size_t cnt, loff_t *offt)
                    :要写给设备写入的数据
                      :要写入的数据长度
                :写入的字节数,如果为负值,表示写入失败
        ret = copy_from_user(kern_buf, buf, cnt); // 得到应用层传递过来的数据
```

72 73 { 74 return 0;

75 } 76 77 /*

84 */

86

89

90

91 92

93

94

95 96

97 98 }

87 { 88

70 */

69 * @return

61 } 62 63 /*

```
78 * @description : 向设备写数据
79 * @param - filp : 设备文件,表示打开的文件描述符
80 * @param – buf
81 * @param – cnt
82 * @param - offt : 相对于文件首地址的偏移
83 * @return
85 static ssize_t led_write(struct file *filp, const char __user *buf,
         size_t cnt, loff_t *offt)
    int ret;
     int val;
     char kern_buf[1];
     if(0 > ret) {
       printk(KERN_ERR "kernel write failed!\r\n");
       return -EFAULT;
     val = readl(data_addr);
```

```
99
     if (0 == kern_buf[0])
```

```
val &= ~(0x1U << 7); // 如果传递过来的数据是 0 则关闭 led
100
```

```
101 else if (1 == kern_buf[0])
```



```
原子哥在线教学: www.yuanzige.com
                                              论坛:www.openedv.com/forum.php
            val |= (0x1U << 7); // 如果传递过来的数据是 1 则点亮 led
    102
    103
         writel(val, data_addr);
    104
    105
         return 0;
    106 }
    107
    108 /*
    109 * @description
                        :关闭/释放设备
    110 * @param - filp : 要关闭的设备文件(文件描述符)
                        :0 成功;其他失败
    111 * @return
    112 */
    113 static int led_release(struct inode *inode, struct file *filp)
    114 {
    115 return 0;
    116 }
    117
    118 static inline void led_ioremap(void)
    119 {
    120 data_addr = of_iomap(dtsled.nd,0);
    121
         dirm_addr = of_iomap(dtsled.nd,1);
    122 outen_addr = of_iomap(dtsled.nd,2);
    123
         intdis_addr = of_iomap(dtsled.nd,3);
    124
         aper_clk_ctrl_addr = of_iomap(dtsled.nd,4);
    125 }
    126
    127 static inline void led_iounmap(void)
    128 {
    129
         iounmap(data_addr);
    130
         iounmap(dirm_addr);
    131
         iounmap(outen_addr);
    132
         iounmap(intdis_addr);
    133
         iounmap(aper_clk_ctrl_addr);
    134 }
    135
    136 /* 设备操作函数 */
    137 static struct file_operations dtsled_fops = {
    138
         .owner = THIS_MODULE,
    139
         .open = led_open,
    140 .read = led_read,
    141
         .write = led_write,
```

```
142 .release = led_release,
```



```
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
    143 };
    144
    145 static int __init led_init(void)
    146 {
    147
         const char *str;
    148
         u32 val;
    149
         int ret;
    150
    151
         /*1.获取 led 设备节点*/
    152
         dtsled.nd = of_find_node_by_path("/led");
    153
          if(NULL == dtsled.nd){
    154
            printk(KERN_ERR "led node can not found!\r\n");
    155
            return -EINVAL;
    156
         }
    157
    158
         /*2.读取 status 属性*/
    159
         ret = of_property_read_string(dtsled.nd,"status",&str);
    160
         if(!ret){
    161
            if(strcmp(str,"okay"))
    162
            return -EINVAL;
         }
    163
    164
         /*3.获取 compatible 属性值并进行匹配*/
    165
    166
          ret = of_property_read_string(dtsled.nd,"compatible",&str);
    167
          if(0>ret)
    168
            return -EINVAL;
    169
    170
          if(strcmp(str,"alientek,led"))
    171
            return -EINVAL;
    172
    173
          printk(KERN_ERR "led device matching successful!\r\n");
    174
    175
         /* 4.寄存器地址映射 */
    176
         led_ioremap();
    177
    178
         /* 5.使能 GPIO 时钟 */
    179
         val = readl(aper_clk_ctrl_addr);
    180
          val |= (0x1U << 22);
    181
          writel(val, aper_clk_ctrl_addr);
    182
    183 /* 6.关闭中断功能 */
```



```
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
    184
          val |= (0x1U << 7);
    185
          writel(val, intdis_addr);
    186
    187
         /* 7.设置 GPIO 为输出功能 */
    188
         val = readl(dirm_addr);
    189
         val = (0x1U << 7);
          writel(val, dirm_addr);
    190
    191
         /* 8.使能 GPIO 输出功能 */
    192
    193
         val = readl(outen addr);
    194
          val = (0x1U << 7);
    195
          writel(val, outen_addr);
    196
    197
         /* 9.初始化 LED 的默认状态 */
    198
         val = readl(data_addr);
    199
    200
          ret = of_property_read_string(dtsled.nd,"default-state",&str);
    201
          if(!ret){
    202
            if(!strcmp(str,"on"))
    203
              val |= (0x1U << 7);
    204
            else
    205
              val &= ~(0x1U << 7);
    206
          }else
    207
            val &= ~(0x1U << 7);
    208
    209
          writel(val, data_addr);
    210
    211
          /* 10.注册字符设备驱动 */
    212
          /* 创建设备号 */
         if (dtsled.major) {
    213
    214
            dtsled.devid = MKDEV(dtsled.major, 0);
            ret = register_chrdev_region(dtsled.devid, DTSLED_CNT, DTSLED_NAME);
    215
    216
            if (ret)
    217
              goto out1;
    218
          } else {
    219
            ret = alloc_chrdev_region(&dtsled.devid, 0, DTSLED_CNT, DTSLED_NAME);
    220
            if (ret)
    221
              goto out1;
    222
    223
            dtsled.major = MAJOR(dtsled.devid);
    224
            dtsled.minor = MINOR(dtsled.devid);
```



```
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
    225 }
    226
          printk("newcheled major=%d,minor=%d\r\n",dtsled.major, dtsled.minor);
    227
    228
    229
          /* 初始化 cdev */
    230
          dtsled.cdev.owner = THIS MODULE;
    231
          cdev_init(&dtsled.cdev, &dtsled_fops);
    232
          /* 添加一个 cdev */
    233
    234
         ret = cdev_add(&dtsled.cdev, dtsled.devid, DTSLED_CNT);
    235
          if (ret)
    236
            goto out2;
    237
    238
         /* 创建类 */
    239
          dtsled.class = class_create(THIS_MODULE, DTSLED_NAME);
    240
         if (IS_ERR(dtsled.class)) {
    241
            ret = PTR_ERR(dtsled.class);
    242
            goto out3;
    243
         }
    244
    245
         /* 创建设备 */
    246
         dtsled.device = device_create(dtsled.class, NULL,
    247
                dtsled.devid, NULL, DTSLED_NAME);
    248
          if (IS_ERR(dtsled.device)) {
    249
            ret = PTR_ERR(dtsled.device);
    250
            goto out4;
    251
         }
    252
    253
         return 0;
    254
    255 out4:
    256
          class_destroy(dtsled.class);
    257
    258 out3:
    259
          cdev_del(&dtsled.cdev);
    260
    261 out2:
    262
          unregister_chrdev_region(dtsled.devid, DTSLED_CNT);
    263
    264 out1:
    265
         led_iounmap();
```



```
领航者 ZYNQ 之嵌入式 Linux 开发指南
原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php
   266
   267
        return ret;
   268 }
   269
   270 static void __exit led_exit(void)
   271 {
   272 /* 注销设备 */
   273
        device_destroy(dtsled.class, dtsled.devid);
   274
        /* 注销类 */
   275
   276
        class_destroy(dtsled.class);
   277
   278 /* 删除 cdev */
        cdev_del(&dtsled.cdev);
   279
   280
        /* 注销设备号 */
   281
   282
        unregister_chrdev_region(dtsled.devid, DTSLED_CNT);
   283
   284
        /* 取消地址映射 */
   285 led_iounmap();
   286 }
   287
   288 /* 驱动模块入口和出口函数注册 */
   289 module_init(led_init);
   290 module_exit(led_exit);
   291
   292 MODULE_AUTHOR("DengTao <773904075@qq.com>");
   293 MODULE_DESCRIPTION("Alientek ZYNQ GPIO LED Driver");
   294 MODULE LICENSE("GPL");
```

dtsled.c 文件中的内容和第二十三章的 newchrled.c 文件中的内容基本一样,只是 dtsled.c 中包含了处理设备树的代码,我们重点来看一下这部分代码。

第45行,在设备结构体 dtsled dev 中添加了成员变量 nd, nd 是 device node 结构体类型 指针,表示设备节点。如果我们要读取设备树某个节点的属性值,首先要先得到这个节点, 一般在设备结构体中添加 device_node 指针变量来存放这个节点。

第 118~125 行,通过使用 of_iomap 函数替换之前使用 ioremap 函数来实现物理地址到虚 拟地址的映射,它能够直接解析给定节点的 reg 属性,并将 reg 属性中存放的物理地址和长度 进行映射,使用不同的下标依次对 reg 数组中记录的不同组"物理地址-长度"地址空间进行 映射,非常的方便!

第 152~156 行,通过 of_find_node_by_path 函数获取设备树根节点下的 led 节点,这里我 们用的是绝对路径"/led",因为 led 节点就在根节点"/"下;只有获取成功了才会进行下面 的步骤!



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

第 159~163 行,通过 of_property_read_string 函数获取 led 节点的"status"属性的内容, 当节点中定义了"status"属性,并且值为"okay"时表示设备是可用的,才会进行下面的操 作;如果没有定义"status"属性则默认设备树可用的。

第 166~173 行,通过 of_property_read_string 函数获取 led 节点的 "compatible" 属性的内容,如果节点中没有定义这个属性(也就是获取失败),则表示这个节点不支持我们的驱动 直接退出;如果获取成功了,则使用 strcmp 函数进行比较,看是否等于 "alientek,led",如 果相同则表示匹配成功,可以接着进行下面的步骤了。

第176行,调用自定义的 led_ioremap 函数进行物理地址到虚拟地址的映射。

第 200~209 行,通过 of_property_read_string 函数获取 led 节点的"default-state"属性的内容,根据读取到的内容来设置 LED 灯的初始状态。

那么其他的内容前面都已经讲过了,没什么好说的了,本身驱动也非常的简单。

25.3.3 编写测试 App

本章直接使用第二十三章的测试 App,将第二十三章实验工程目录下的 ledApp.c 源文件和 ledApp 可执行文件一并复制到本章实验工程下即可,这样就不用再去编译 ledApp.c 了。

25.4 运行测试

25.4.1 编译驱动程序和测试 App

1、编译驱动程序

编写 Makefile 文件,本章实验的 Makefile 文件和第二十三章实验基本一样,我们直接将 第二十三章实验目录下的 Makefile 文件拷贝到本实验目录中,修改 Makefile 文件,只是将 objm 变量的值改为 dtsled.o, Makefile 内容如下所示:

示例代码 25.4.1 Makefile 文件内容

```
1 KERN_DIR := /home/zy/workspace/kernel-driver/linux-xlnx_rlpase_v5.4_2020.2
2
3 obj-m := dtsled.o
4
5 all:
6 make -C $(KERN_DIR) M=`pwd` modules
7
8 clean:
9 make -C $(KERN_DIR) M=`pwd` clean
第 3 行,设置 obj-m 变量的值为 dtsled.o。
输入如下命令编译出驱动模块文件:
make
编译成功以后就会生成一个名为 "dtsled.ko" 的驱动模块文件,如下所示:
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php zy@zy-virtual-machine:/mnt/hgfs/share18/linux驱动例程/4 dtsled\$ zy@zy-virtual-machine:/mnt/hgfs/share18/linux驱动例程/4_dtsled\$ make make -C /home/zy/workspace/kernel-driver/linux-xlnx-xlnx_rebase_v5.4_2020 .2 M=`pwd` modules make[1]: 进入目录"/home/zy/workspace/kernel-driver/linux-xlnx-xlnx rebase v5.4 2020.2' CC [M] /mnt/hgfs/share18/linux驱动例程/4 dtsled/dtsled.o Building modules, stage 2. MODPOST 1 modules LD [M] /mnt/hgfs/share18/linux驱动例程/4 dtsled/dtsled.ko make[1]: 离开目录"/home/zy/workspace/kernel-driver/linux-xlnx-xlnx rebase v5.4 2020.2" zy@zy-virtual-machine:/mnt/hgfs/share18/linux驱动例程/4 dtsled\$ zy@zy-virtual-machine:/mnt/hgfs/share18/linux驱动例程/4 dtsled\$ zy@zy-virtual-machine:/mnt/hgfs/share18/linux驱动例程/4_dtsled\$ ls dtsled.mod.c ledApp modules.order system-user.c dtsled.mod.o ledApp.c Module.symvers system-user-dri.c dtsled.c dtsled.ko system-user.dtsi Makefile newchrled.c dtsled.mod dtsled.o zy@zy-virtual-machine:/mnt/hgfs/share18/linux驱动例程/4_dtsled\$

图 25.4.1 编译驱动模块

2、编译测试 App

直接使用第二十三章编译好的可执行文件 ledApp。

25.4.2 运行测试

将上面编译出来的 dtsled.ko 和 ledApp 这两个文件拷贝到 NFS 共享目录下根文件系统的 /lib/modules/5.4.0-xilinx 文件夹中。

SD卡 FAT 分区 system.dtb 文件替换完成后,将 SD 启动卡插到开发板中,连接开发板串口、网口和电源线,设置拨码开关为 SD 卡启动模式,然后打开电源,启动开发板,通过串口终端进入"/lib/modules/5.4.0-xilinx"目录下,输入如下命令加载 dtsled.ko 驱动模块:

depmod //第一次加载驱动的时候需要运行此命令

modprobe dtsled.ko //加载驱动

驱动加载成功以后会在终端中输出一些信息,如下图所示:



图 25.4.2 加载驱动

从上图中可以看出, led 驱动已经和 led 设备节点匹配成功了!并且开发板上的 PS_LED0 被点亮了,因为我们在设备树中将 led 节点的 "default-state" 属性的值设置为 "on",所以初 始化 LED 的时候会将其点亮。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

驱动加载成功以后就可以使用 ledApp 软件来测试驱动是否工作正常,输入如下命令打开 LED 灯:

./ledApp /dev/dtsled 0 //关闭 LED 等

输入上述命令以后查看开发板上的 PS_LED0 灯是否熄灭,如果熄灭的话说明驱动工作正常。在输入如下命令点亮灯:

./ledApp /dev/dtsled 1 //点亮 LED 灯

输入上述命令以后查看开发板上的 PS_LED0 灯是否被点亮。如果要卸载驱动的话输入如下命令即可:

rmmod dtsled.ko



原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php

第二十六章 gpio 子系统简介

上一章我们编写了基于设备树的 LED 驱动,但是驱动的本质还是没变,都是配置 LED 灯 所使用的 GPIO 寄存器, 驱动开发方式和裸机基本没啥区别。在驱动程序用到了 GPIO 就直接 去读写 GPIO 相关的寄存器,这样会引发一个问题,大家有没有想过,如果另外一个驱动工程 师写了一个驱动也用到这个相同的管脚,那么它也会去操作这些GPIO寄存器,也就是说多个 驱动代码中都用了这个 GPIO, 那么这会乱套的, 对于 linux 系统来说是绝对不允许的事情!

内核维护者在内核中设计了一些统一管控系统资源的体系,这些体系让内核能够对系统 资源在各个驱动之间的使用统一协调和分配,保证整个内核的稳定健康运行。例如系统中所 有的 GPIO 就属于系统资源,每个驱动模块如果要使用某个 GPIO 就要先调用特殊的接口先申 请,申请到后使用,使用完后要释放。又譬如中断号也是一种资源,驱动在使用前也必须去 申请,只能申请成功之后才能使用,否则不能使用。

所以本章就引入了 gpio 子系统,那么 gpio 子系统其实就是内核中管控 GPIO 资源的一套 软件设计体系。



正点原子

26.1 gpio 子系统

26.1.1 gpio 子系统简介

gpio 子系统是 linux 内核当中用于管理 GPIO 资源的一套系统,它提供了很多 GPIO 相关 的 API 接口。驱动程序中使用 GPIO 之前需要向 gpio 子系统申请,申请成功之后才可以使用, 例如设置 GPIO 的输入、输出方向,设置 GPIO 输出高或低电平、读取 GPIO 输入电平等等。

gpio 子系统的主要目的就是方便驱动开发者使用 gpio,驱动开发者在设备树中添加 gpio 相关信息,然后就可以在驱动程序中使用 gpio 子系统提供的 API 函数来操作 GPIO, Linux 内 核向驱动开发者屏蔽掉了 GPIO 的设置过程,极大的方便了驱动开发者使用 GPIO。

26.1.2 ZYNQ 的 gpio 子系统驱动

gpio 子系统虽然方便了驱动开发者使用 gpio, 但是最终还是得去操作硬件寄存器; 所以 在使用 gpio 子系统之前,我们需要向内核 gpio 子系统注册这一套操作硬件寄存器的"方法", 关于这个后面再说,我们先来看看在设备树中是如何描述 gpio 信息。

1、设备树中的 gpio 信息

我们来看个示例,如下所示:

```
示例代码 26.1.1 gpio 信息
```

```
1 led {
```

```
compatible = "alientek,led";
2
   status = "okay";
3
   default-state = "on";
4
5 led-gpio = <&gpio0 7 GPIO_ACTIVE_HIGH>;
6};
7
8 beeper {
9
    compatible = "alientek,beeper";
10 status = "okay";
11 default-state = "off";
12 beeper-gpio = <&gpio0 60 GPIO_ACTIVE_HIGH>;
```

```
13 };
```

在上面的示例当中,定义了两个节点 led 和 beeper,其中 led 节点是对应了一个 led 设备, beeper 节点对应了一个蜂鸣器设备; 第5行, led 节点中定义了一个属性"led-gpio", 该属性 用于描述 LED 设备由那个 GPIO 控制,属性值一共有三个,我们来看一下这三个属性值的含 义, "&gpio0" 表示 led 引脚所使用的 IO 属于 gpio0, "7" 表示 gpio0 的第 7 号 IO, 通过这 两个值 led 驱动程序就知道 led 引脚使用了 GPIO0 IO07 这 GPIO。"GPIO ACTIVE HIGH" 表示高电平有效,如果改为"GPIO_ACTIVE_LOW"就表示低电平有效 (GPIO_ACTIVE_HIGH 和 GPIO_ACTIVE_LOW 其实是宏定义, GPIO_ACTIVE_HIGH 等于 0, GPIO_ACTIVE_LOW 等于1); 高电平有效的意思就是,当 GPIO0_IO07 管脚输出高电平 时 led 才会被点亮。


原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

根据上面这些信息,LED 驱动程序就可以使用 GPIO0 IO07 来控制 LED 灯的亮灭状态了,

打开 zynq-7000.dtsi, 在里面找到如下所示内容:

示例代码 26.1.2 gpio0 节点

- 110 gpio0: gpio@e000a000 {
- compatible = "xlnx,zynq-gpio-1.0"; 111
- 112 #gpio-cells = <2>;
- 113 clocks = <&clkc 42>;
- 114 gpio-controller;
- 115 interrupt-controller;
- 116 #interrupt-cells = $\langle 2 \rangle$;
- 117 interrupt-parent = <&intc>;
- 118 interrupts = <0.20.4>;
- 119 $reg = \langle 0xe000a000 0x1000 \rangle;$

120 };

gpio0节点信息描述了 ZYNQ PS GPIO 控制器的所有信息,重点就是 GPIO 外设寄存器基 地址以及兼容属性。关于 ZYNQ 系列 SOC 的 PS GPIO 控制器绑定信息请查看文档 Documentation/devicetree/bindings/gpio/gpio-zynq.txt.

第 111 行,设置 gpio0 节点的 compatible 属性为 "xlnx,zynq-gpio-1.0",在 Linux 内核中 搜索这两个字符串就可以找到 ZYNQ 的 GPIO 驱动程序。

第 119 行, reg 属性设置了 GPIO 控制器的寄存器基地址为 0xE000A000, 大家可以打开 ug585-Zynq-7000-TRM.pdf 手册, 找到"Appx. B: Register Details"章节的 General Purpose I/O (gpio)小节,如下图所示:

Ch. 30: XADC Interface
Ch. 31: PCI Express
Ch. 32: Device Secure Boot
Appx. A: Additional Resources
Appx. B: Register Details
Overview
Acronyms
Module Summary
AXI_HP Interface (AFI) (axi_hp)
CAN Controller (can)
DDR Memory Controller (ddrc)
CoreSight Cross Trigger Interface (cti)
Performance Monitor Unit (cortexa9_pmu)
CoreSight Program Trace Macrocell (ptm)
Debug Access Port (dap)
CoreSight Embedded Trace Buffer (etb)
PL Fabric Trace Monitor (ftm)
CoreSight Trace Funnel (funnel)
CoreSight Intstrumentation Trace Macrocell (itm)
CoreSight Trace Packet Output (tpiu)
Device Configuration Interface (devcfg)
DMA Controller (dmac)
Gigabit Ethernet Controller (GEM)
General Purpose I/O (gpio)
Interconnect Qu3 (qus301)
NIC301 Address Region Control (nic301_addr_region_ctrl_r
I2C Controller (IIC)
L2 Cache (L2Cpl310)
E The second se second second sec

B.19 General Purpose I/O (gpio)

Module Name Software Name Base Address Description

Vendor Info

General Purpose I/O (gpio) XGPIOPS 0xE000A000 gpio

General Purpose Input / Output

Register Summary

Register Name	Address	Width	Туре	Reset Value
XGPIOPS_DATA_LSW_O FFSET	0x0000000	32	mixed	x
VODIODO DATA MOM	000000004	22		

图 26.1.1 ZYNQ GPIO 寄存器基地址

从上图可以看出, GPIO 控制器的基地址就是 0xE000A000。

第 114 行, "gpio-controller"表示 gpio0 节点是个 GPIO 控制器,表示这个节点对应的驱 动程序是 gpio 驱动。

第 112 行, "#gpio-cells"属性和 "#address-cells" 类似,在 gpio0 节点中#gpio-cells 的值 等于 2, 表示一共有两个 cell, 大家可以这样理解, 使用 gpio0 的时候, 需要传递 2 个参数过



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

去,第一个参数为 GPIO 编号,比如 "&gpio0 7" 就表示 GPIO0_IO07。第二个参数表示 GPIO 极性,如果为 0(GPIO_ACTIVE_HIGH)的话表示高电平有效,如果为 1(GPIO_ACTIVE_LOW) 的话表示低电平有效。

所以在 led 节点中 "led-gpio = <&gpio0 7 GPIO_ACTIVE_HIGH>;"就表示使用了 GPIO0 IO07 这个管脚,并且是高电平有效。

以上就讲解了在设备树如何指定、描述一个 gpio。

2、GPIO 驱动程序简介

在前面给大家说过,gpio 子系统虽然方便了驱动开发者使用 gpio,但是最终还是得去操作硬件寄存器;所以在使用 gpio 子系统之前,我们需要向内核 gpio 子系统注册这一套操作硬件寄存器的"方法",那么这套"方法"就是由 GPIO 驱动程序去实现、并注册到 gpio 子系统,所以 GPIO 驱动程序就负责实现 GPIO 操控硬件寄存器的代码,并注册到内核 gpio 子系统中由 gpio 子系统进行统一管控。

所以怎么去控制 GPIO 的高低电平、怎么去设置输入输出方向,怎么读取 GPIO 高低电平等这些代码都是在 GPIO 驱动程序中实现的。

gpio0节点的 compatible 属性描述了兼容性,在 Linux 内核中搜索 "xlnx,zynq-gpio-1.0"这个字符串,找打我们 ZYNQ 的 GPIO 驱动程序代码。drivers/gpio/gpio-zynq.c 就是 ZYNQ 的 PS 端 GPIO 驱动程序,在此文件中有如下所示 of_device_id 匹配表:

示例代码 26.1.3 zynq_gpio_of_match 配置表

781 static const struct of_device_id zynq_gpio_of_match[] = {

782 {.compatible = "xlnx,zynq-gpio-1.0", .data = &zynq_gpio_def },

783 { .compatible = "xlnx,zynqmp-gpio-1.0", .data = &zynqmp_gpio_def },

```
784 { /* end of table */ }
```

785 };

786 MODULE_DEVICE_TABLE(of, zynq_gpio_of_match);

第782行的 compatible 值为"xlnx,zynq-gpio-1.0",和 gpio0节点的 compatible 属性匹配,因此 gpio-zynq.c 就是 ZYNQ 的 GPIO 控制器驱动文件。gpio-zynq.c 所在的目录为 drivers/gpio,进入到这个目录下可以看到很多芯片的 gpio 驱动文件,"gpiolib"开始的文件是 gpio 驱动的 核心文件,如下图所示:

🥝 gpiolib.c	2019-05-25 10:26	sourceinsight.c_file
🥝 gpiolib.h	2019-05-25 10:26	H文件
🥝 gpiolib-acpi.c	2019-05-25 10:26	sourceinsight.c_file
🥝 gpiolib-legacy.c	2019-05-25 10:26	sourceinsight.c_file
🥝 gpiolib-of.c	2019-05-25 10:26	sourceinsight.c_file
🥝 gpiolib-sysfs.c	2019-05-25 10:26	sourceinsight.c_file

图 26.1.2 gpio 驱动核心文件

gpio-zynq.c 文件的内容这里先不分析,等后面时机成熟的时候会专门介绍 gpio-zynq.c 程序以及如何编写一个 GPIO 驱动程序。

26.1.3 gpio 子系统 API 函数

对于驱动开发人员,设置好设备树以后就可以使用 gpio 子系统提供的 API 函数来操作指定的 GPIO,gpio 子系统向驱动开发人员屏蔽了具体的读写寄存器过程。这就是驱动分层与分



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php 离的好处,大家各司其职,做好自己的本职工作即可。gpio 子系统提供的常用的 API 函数有 下面几个:

1、gpio_request 函数

gpio_request 函数用于申请一个 GPIO 管脚, 在使用一个 GPIO 之前一定要使用 gpio_request 进行申请, 函数原型如下:

int gpio_request(unsigned gpio, const char *label)

函数参数和返回值含义如下:

gpio:要申请的 gpio 标号,使用 of_get_named_gpio 函数从设备树获取指定 GPIO 属性信息,此函数会返回这个 GPIO 的标号。

label:给 gpio 设置个名字。 **返回值:**0,申请成功;其他值,申请失败。

2、gpio_free 函数

如果不使用某个 GPIO 了,那么就可以调用 gpio_free 函数进行释放。函数原型如下:

void gpio_free(unsigned gpio) 函数参数和返回值含义如下:

gpio: 要释放的 gpio 标号。

返回值:无。

3、gpio_direction_input 函数

此函数用于设置某个 GPIO 为输入,函数原型如下所示:

int gpio_direction_input(unsigned gpio)

函数参数和返回值含义如下:

gpio: 要设置为输入的 GPIO 标号。

返回值: 0,设置成功;负值,设置失败。

4、gpio_direction_output 函数

此函数用于设置某个 GPIO 为输出,并且设置默认输出值,函数原型如下: int gpio_direction_output(unsigned gpio, int value) 函数参数和返回值含义如下: gpio:要设置为输出的 GPIO 标号。 value: GPIO 默认输出值。 返回值: 0,设置成功;负值,设置失败。

5、gpio_get_value 函数

此函数用于获取某个 GPIO 的值(0 或 1),此函数是个宏,定义所示: #define gpio_get_value __gpio_get_value int __gpio_get_value(unsigned gpio) 函数参数和返回值含义如下: gpio: 要获取的 GPIO 标号。 返回值: 非负值,得到的 GPIO 值;负值,获取失败。

6、gpio_set_value 函数



 原子哥在线教学:www.yuanzige.com
 论坛:www.openedv.com/forum.php

 此函数用于设置某个 GPIO 的值,此函数是个宏,定义如下

 #define gpio_set_value __gpio_set_value

 void __gpio_set_value (unsigned gpio, int value)

 函数参数和返回值含义如下:

 gpio: 要设置的 GPIO 标号。

 value: 要设置的值。

 返回值: 无

 关于 gpio 子系统常用的 API 函数就讲这些,这些是我们用的最多的。

26.1.4 与 gpio 相关的 OF 函数

示例代码 26.1.1 中使用 led-gpio 属性指定了 LED 灯对应的 GPIO,使用 beeper-gpio 属性 指定了蜂鸣器对应的 GPIO,那么在驱动程序中就需要去读取这些属性的内容,Linux 内核提 供了几个与 GPIO 有关的 OF 函数,常用的几个 OF 函数如下所示:

1、of_gpio_named_count 函数

of_gpio_named_count 函数用于获取设备树某个属性里面定义了几个 GPIO 信息,要注意 的是空的 GPIO 信息也会被统计到,比如:

gpios = <0 &gpio1 1 2 0 &gpio2 3 4>;

上述代码的 "gpios" 节点一共定义了 4个 GPIO,但是有 2个是空的,没有实际的含义。 通过 of_gpio_named_count 函数统计出来的 GPIO 数量就是 4个,此函数原型如下:

int of_gpio_named_count(struct device_node *np, const char *propname)

函数参数和返回值含义如下:

np: 设备节点。

propname: 要统计的 GPIO 属性。

返回值:正值,统计到的 GPIO 数量;负值,失败。

2、of_gpio_count 函数

和 of_gpio_named_count 函数一样,但是不同的地方在于,此函数统计的是 "gpios" 这个 属性的 GPIO 数量,而 of_gpio_named_count 函数可以统计任意属性的 GPIO 信息,函数原型 如下所示:

int of_gpio_count(struct device_node *np)
函数参数和返回值含义如下:
np:设备节点。
返回值:正值,统计到的 GPIO 数量;负值,失败。

3、of_get_named_gpio 函数

此函数获取 GPIO 编号,gpio 子系统为了方便管理系统中的 GPIO 资源,每一个 GPIO 管脚都有一个对应的编号,Linux 内核中关于 GPIO 的 API 函数都要使用 GPIO 编号,此函数会



原子哥在线教学:www.yuanzige.com 论坛:www.openedv.com/forum.php

将设备树中类似<&gpio07 GPIO_ACTIVE_LOW>的属性信息转换为对应的 GPIO 编号,此函数在驱动中使用很频繁!函数原型如下:

int of_get_named_gpio(struct device_node *np, const char *propname, int index)

函数参数和返回值含义如下:

np: 设备节点。

propname: 包含要获取 GPIO 信息的属性名。

index: GPIO 索引,因为一个属性里面可能包含多个 GPIO,此参数指定要获取哪个 GPIO 的编号,如果只有一个 GPIO 信息的话此参数为 0。

返回值:正值,获取到的 GPIO 编号;负值,失败。

26.2 pinctrl 子系统

gpio 子系统是用于管理系统中的 GPIO 资源的,那么 pinctrl 子系统又是做什么的呢? pinctrl 其实就是 PIN control 的一个缩写形式。

如果使用过 STM32 的话应该都记得, STM32 也是要先设置某个 PIN 的复用功能以及电气 特性,例如 IO 速率、上下拉等,其实对于大多数的 32 位 SOC 而言, PIN 都是需要设置复用 功能和电气特性,因为大多数 SOC 的 pin 都是支持复用的,同一个 PIN 可以作为多种功能,例如 vivado 中配置 SD0 时,可以使用 MIO40~45,也可以选择 MIO16~21 等,以及配置它们 的电气特性,如所示:

Peripheral	10	Signal	Ю Туре	Speed	Pullup	Direction	Polarity
> ENET 1							
> 🕑 USB 0	MIO 28 39 🗸 🗸						
USB 1							
∨ 🖌 SD 0	MIO 40 45 🛛 🗸						
> 🕑 CD	EMIO						
WP	MIO 16 21						
Power	MIO 28 33						
SD 0	MIO 40 45	clk	LVCMOS 1.8V 🗸	slow 🗸	enabled 🗸 🗸	inout	
SD 0	MIO 41	cmd	LVCMOS 1.8V 🗸 🗸	slow 🗸	enabled 🗸 🗸	inout	
SD 0	MIO 42	data[0]	LVCMOS 1.8V 🗸 🗸	slow 🗸	enabled 🗸 🗸	inout	
SD 0	MIO 43	data[1]	LVCMOS 1.8V 🗸 🗸	slow 🗸	enabled 🗸 🗸	inout	
SD 0	MIO 44	data[2]	LVCMOS 1.8V 🗸 🗸	slow 🗸	enabled 🗸 🗸	inout	
SD 0	MIO 45	data[3]	LVCMOS 1.8V 🗸 🗸	slow 🗸	enabled 🗸 🗸	inout	

图 26.2.1 vivado 中配置 SD0

因此 Linux 内核针对 PIN 的配置推出了 pinctrl 子系统,对于 GPIO 的配置推出了 gpio 子系统,所以说到这里就知道了,pinctrl 子系统是内核中专门用于管理、配置 PIN 的一套子系统。

配置 PIN 的复用功能和电器特性也是通过控制硬件寄存器来实现的,所以 pinctrl 子系统 最终也是如此。有 gpio 驱动程序,那必然也有 pinctrl 驱动程序,pinctrl 驱动程序中实现了 PIN 的配置方法并,并注册到 pinctrl 子系统,所以 pinctrl 驱动程序就负责实现配置 PIN 的底层代码(主要就是寄存器控制),并注册到内核 pinctrl 子系统中由 pinctrl 子系统进行统一管理。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

在内核源码 drivers/pinctrl 目录下有很多的 pinctrl-xxx.c 文件,它们都是不同 SoC 所对应的 pinctrl 驱动程序源文件,例如对于 ZYNQ 来说,pinctrl-zynq.c 就是它的 pinctrl 驱动程序文件。pinctrl-zynq.c 文件的内容这里先不分析,等后面时机成熟的时候会专门介绍 pinctrl-zynq.c 程序 以及如何编写一个 pinctrl 驱动程序。

对于 ZYNQ 来说,我们使用了 vivado 图形化完成了对 PIN 的配置并在 fsbl 阶段将配置信息写入了硬件寄存器中(具体的过程就不分析了),所以不需要在内核阶段进行配置,所以就不详细介绍 pinctrl 子系统了,本小节的只是告诉大家 pinctrl 子系统的存在以及它的作用,对于 ZYNQ 来说,我们可以忽略它!





正点原子



在上一章当中已经给大家介绍了 linux 的 gpio 子系统,那本章我们就来编写一个基于 gpio 子系统 API 的 LED 驱动程序,本章将在第二十五章实验的基础上进行修改完成。

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

正点原子

27.1 实验简介

在第二十五章中,虽然我们在 dtsled.c 驱动代码中获取到了 GPIO 有关寄存器物理地址, 然后使用 of_iomap 函数进行内存映射,得到对应的虚拟地址,最后操作寄存器对应的虚拟地 址完成对 GPIO 的初始化,但本质上还是跟裸机没啥区别,还非常的麻烦。所以本章就带大家 完成一个 gpio 子系统下的 led 驱动程序,本章我们在第二十五章实验基础上完成,本章实验 重点内容如下:

1、修改第二十五章创建的 led 节点,在节点中指定 led 使用的 GPIO。

2、去掉 led 节点中的 reg 属性,因为不需要用到了。

3、在第二十五章驱动实验的基础上修改驱动代码,获取设备树中传入的 gpio,调用 gpio 子系统提供的接口函数对 gpio 进行操控。

27.2 硬件原理图分析

本章实验硬件原理图参考 22.3 小节即可。

27.3 实验程序编写

本实验对应的例程路径为:开发板资料盘(A 盘)\4_SourceCode\3_Embedded_Linux\Linux 驱动例程\5 gpioled。

本章实验在第二十五章实验的基础上完成,重点是将驱动改为基于 gpio 子系统的框架下 的 led 驱动代码。

27.3.1 修改设备树文件

1、添加 led-gpio 属性指定 led 使用的 GPIO

打开 linux 内核源码目录下的 arch/arm/boot/dts/system-user.dtsi 文件,找到第二十五章创建 在根节点"/"下的led节点,我们需要对led节点内容进行修改,去掉之前reg属性,添加ledgpio 节点指定 LED 所使用的 GPIO 管脚,修改完成之后如下所示:

示例代码 27.3.1 system-user.dtsi 文件 led 节点

```
1 #define GPIO_ACTIVE_HIGH 0
```

```
2 #define GPIO_ACTIVE_LOW 1
```

3

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

5 #include <dt-bindings/gpio/gpio.h>

6 #include <dt-bindings/input/input.h>

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

8 #include <dt-bindings/phy/phy.h>

9

10/{

11 model = "Alientek Navigator Zynq Development Board";

```
12 compatible = "xlnx,zynq-zc702", "xlnx,zynq-7000";
```

```
13
```



第1和第2行,在设备树文件中定义了两个宏,这两个宏用来表示 GPIO 是高电平有效还 是低电平有效,GPIO_ACTIVE_HIGH 表示高电平有效。

第 19~25 行,将之前定义的 reg 属性给去掉了,加上了一个"led-gpio"属性,它的值等 于 "<&gpio0 7 GPIO_ACTIVE_HIGH>",led-gpio 属性指定了 LED 灯所使用的 GPIO,在这 里就是 GPIO0 的 MIO 7,高电平有效,这些东西在第二十五章给大家讲过了,这里不再重复 啰嗦! 稍后编写驱动程序的时候会获取 led-gpio 属性的内容来得到 GPIO 编号,因为 gpio 子系 统的 API 操作函数需要 GPIO 编号。

2、添加 pinctrl 节点

对于 ZYNQ 来说,我们不需要 pinctrl,原因前面已经给大家讲过了!

3、重新编译设备树

上面修改完成之后,执行下面的命令重新编译设备树,如下所示:

make dtbs

zy@zy-virtual-machine:~/workspace/kernel-driver/linux-xlnx_rlnx_rebase_v5.4_2020.2\$
zy@zy-virtual-machine:~/workspace/kernel-driver/linux-xlnx-xlnx_rebase_v5.4_2020.2\$ make dtbs
DTC arch/arm/boot/dts/system-top.dtb
zy@zy-virtual-machine:~/workspace/kernel-driver/linux-xlnx-xlnx_rebase_v5.4_2020.2\$

图 27.3.1 重新编译设备树

编译完成之后将 system-top.dtb 文件重命名为 system.dtb, 然后将其拷贝到开发板的 SD 启动卡 Fat 分区, 替换掉之前的 system.dtb 文件, 然后重启开发板, 进入到/proc/device-tree 目录下, 可以看到我们的 led 节点, 进入到 led 节点目录下, 如下所示:

领航者 ZYN	Q之嵌入式I	.inux 开发指	南	۲ 🕑 🖾	点原子
原子哥在线教学	: www.yuanzig	e.com 论坛	:www.open	edv.com/forum.php	
root@ALIENTEK-ZY root@ALIENTEK-ZY root@ALIENTEK-ZY root@ALIENTEK-ZY root@ALIENTEK-ZY #address-cells	NQ-driver:~# NQ-driver:~# NQ-driver:~# cd NQ-driver:/proc, NQ-driver:/proc, amba amba	/proc/device-tre /device-tree# /device-tree# ls compatible	ee 👞	model	replicator
aliases root@ALIENTEK-ZY root@ALIENTEK-ZY root@ALIENTEK-ZY root@ALIENTEK-ZY compatible	amba_pt chosen /NQ-driver:/proc, /NQ-driver:/proc, /NQ-driver:/proc, /NQ-driver:/proc, default-state 10	fixedregulator /device-tree# /device-tree# cd /device-tree/leda /device-tree/leda ed-gpio na	memory led/ # # ls ne	pmu@f8891000 status	
root@ALIENTEK-ZY	NQ-driver:/proc	/device-tree/led	#	0 20 200	

图 27.3.2 led 节点下的属性

可以看到有一个"led-gpio"文件,这就是我们前面新添加的属性。因为 led-gpio 属性的 值是引用别的节点,所以直接用 cat 命令看不到这个文件的东西,我们可以使用 od 命令查看,如下所示:



图 27.3.3 od 命令查看属性

所以可以知道它里面是有东西的。

27.3.2 编写 LED 灯驱动程序

设备树准备好以后就可以编写驱动程序了,本章实验在第二十五章实验驱动文件 dtsled.c 的基础上修改而来。首先在 drivers 目录下新建名为 "5_gpioled" 文件夹,然后在 5_gpioled 文 件夹里面新建 gpioled.c 文件,在 gpioled.c 里面输入如下内容:

示例代码 27.3.2 gpioled.c 文件内容

```
2 Copyright © ALIENTEK Co., Ltd. 1998-2029. All rights reserved.
3 文件名 : gpioled.c
4 作者 : 邓涛
5版本
     : V1.0
6 描述 : ZYNQ LED 驱动文件。
7 其他 :无
8 论坛 : www.openedv.com
9 日志
      :初版 V1.0 2019/1/30 邓涛创建
11
12 #include <linux/types.h>
13 #include <linux/kernel.h>
14 #include <linux/delay.h>
```



```
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
    15 #include <linux/ide.h>
    16 #include <linux/init.h>
    17 #include <linux/module.h>
    18 #include <linux/errno.h>
    19 #include linux/gpio.h>
    20 #include <asm/mach/map.h>
    21 #include <asm/uaccess.h>
    22 #include <asm/io.h>
    23 #include <linux/cdev.h>
    24 #include <linux/of.h>
    25 #include <linux/of_address.h>
    26 #include <linux/of_gpio.h>
    27
    28 #define GPIOLED_CNT
                                           /* 设备号个数 */
                               1
    29 #define GPIOLED_NAME
                               "gpioled" /* 名字 */
    30
    31 /* dtsled 设备结构体 */
    32 struct gpioled_dev {
    dev_t devid;
                            /* 设备号 */
    34 struct cdev cdev;
                           /* cdev */
                           /* 类 */
    35 struct class *class;
    36 struct device *device; /* 设备 */
                           /* 主设备号 */
    37 int major;
    38 int minor;
                           /* 次设备号 */
    39
        struct device_node *nd; /* 设备节点 */
                           /* LED 所使用的 GPIO 编号 */
    40
        int led_gpio;
    41 };
    42
    43 static struct gpioled_dev gpioled; /* led 设备 */
    44
    45 /*
    46 * @description
                          :打开设备
    47 * @param – inode : 传递给驱动的 inode
    48 * @param – filp
                          :设备文件, file 结构体有个叫做 private_data 的成员变量
                           一般在 open 的时候将 private_data 指向设备结构体。
    49 *
    50 * @return
                           :0 成功;其他 失败
    51 */
    52 static int led_open(struct inode *inode, struct file *filp)
    53 {
    54 filp->private_data = &gpioled; /* 设置私有数据 */
    55 return 0;
```

```
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
    56 }
    57
    58 /*
    59 * @description
                         :从设备读取数据
    60 * @param – filp
                         :要打开的设备文件(文件描述符)
                         :返回给用户空间的数据缓冲区
    61 * @param – buf
    62 * @param – cnt
                         :要读取的数据长度
    63 * @param – offt
                         :相对于文件首地址的偏移
                          :读取的字节数,如果为负值,表示读取失败
    64 * @return
    65 */
    66 static ssize_t led_read(struct file *filp, char __user *buf,
            size_t cnt, loff_t *offt)
    67
    68 {
    69
          return 0;
    70 }
    71
    72 /*
    73 * @description
                      : 向设备写数据
                         :设备文件,表示打开的文件描述符
    74 * @param – filp
    75 * @param – buf
                         :要写给设备写入的数据
    76 * @param – cnt
                         :要写入的数据长度
    77 * @param – offt
                         :相对于文件首地址的偏移
                         :写入的字节数,如果为负值,表示写入失败
    78 * @return
    79 */
    80 static ssize_t led_write(struct file *filp, const char __user *buf,
    81
            size_t cnt, loff_t *offt)
    82 {
    83 int ret;
    84
        char kern_buf[1];
    85
    86 ret = copy_from_user(kern_buf, buf, cnt); // 得到应用层传递过来的数据
       if(0 > ret) \{
    87
    88
          printk(KERN_ERR "kernel write failed!\r\n");
    89
          return -EFAULT;
    90
       }
    91
    92 if (0 == \text{kern}_{buf}[0])
          gpio_set_value(gpioled.led_gpio, 0); // 如果传递过来的数据是 0 则关闭 led
    93
    94
        else if (1 == \text{kern}_{buf[0]})
          gpio_set_value(gpioled.led_gpio, 1); // 如果传递过来的数据是 1 则点亮 led
    95
    96
```

正点原子



```
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
    97
        return 0;
    98 }
    99
    100 /*
    101 * @description
                               :关闭/释放设备
                                 :要关闭的设备文件(文件描述符)
    102 * @param – filp
                                 :0 成功;其他失败
    103 * @return
    104 */
    105 static int led_release(struct inode *inode, struct file *filp)
    106 {
    107 return 0;
    108 }
    109
    110 /* 设备操作函数 */
    111 static struct file_operations gpioled_fops = {
    112 .owner = THIS_MODULE,
    113 .open = led_open,
    114 .read = led_read,
    115 .write = led_write,
    116 .release = led_release,
    117 };
    118
    119 static int __init led_init(void)
    120 {
    121
         const char *str;
    122
         int ret;
    123
    124 /* 1.获取 led 设备节点 */
         gpioled.nd = of_find_node_by_path("/led");
    125
    126
         if(NULL == gpioled.nd) {
    127
            printk(KERN_ERR "gpioled: Failed to get /led node\n");
    128
            return -EINVAL;
    129
         }
    130
    131
         /* 2.读取 status 属性 */
    132
         ret = of_property_read_string(gpioled.nd, "status", &str);
    133
         if(!ret) {
    134
            if (strcmp(str, "okay"))
    135
              return -EINVAL;
    136 }
    137
```



```
原子哥在线教学: www.yuanzige.com
                                                论坛:www.openedv.com/forum.php
        /* 2、获取 compatible 属性值并进行匹配 */
    138
    139
          ret = of_property_read_string(gpioled.nd, "compatible", &str);
    140
          if(0 > ret) {
    141
            printk(KERN_ERR "gpioled: Failed to get compatible property\n");
    142
            return ret;
    143
          }
    144
    145
          if (strcmp(str, "alientek,led")) {
    146
            printk(KERN_ERR "gpioled: Compatible match failed\n");
    147
            return -EINVAL;
    148
          }
    149
    150
          printk(KERN_INFO "gpioled: device matching successful!\r\n");
    151
    152
          /* 4.获取设备树中的 led-gpio 属性,得到 LED 所使用的 GPIO 编号 */
    153
          gpioled.led_gpio = of_get_named_gpio(gpioled.nd, "led-gpio", 0);
    154
          if(!gpio_is_valid(gpioled.led_gpio)) {
    155
            printk(KERN_ERR "gpioled: Failed to get led-gpio\n");
    156
            return -EINVAL;
    157
          }
    158
    159
          printk(KERN_INFO "gpioled: led-gpio num = %d\r\n", gpioled.led_gpio);
    160
    161
          /* 5.向 gpio 子系统申请使用 GPIO */
          ret = gpio_request(gpioled.led_gpio, "LED-GPIO");
    162
    163
          if (ret) {
    164
            printk(KERN_ERR "gpioled: Failed to request led-gpio\n");
    165
            return ret;
    166
          }
    167
          /* 6.将 led gpio 管脚设置为输出模式 */
    168
    169
          gpio_direction_output(gpioled.led_gpio, 0);
    170
          /* 7.初始化 LED 的默认状态 */
    171
          ret = of_property_read_string(gpioled.nd, "default-state", &str);
    172
    173
          if(!ret) {
    174
            if (!strcmp(str, "on"))
    175
              gpio_set_value(gpioled.led_gpio, 1);
    176
            else
    177
              gpio_set_value(gpioled.led_gpio, 0);
    178
          } else
```



```
原子哥在线教学: www.yuanzige.com
                                               论坛:www.openedv.com/forum.php
    179
            gpio_set_value(gpioled.led_gpio, 0);
    180
         /* 8.注册字符设备驱动 */
    181
         /* 创建设备号 */
    182
         if (gpioled.major) {
    183
    184
            gpioled.devid = MKDEV(gpioled.major, 0);
    185
            ret = register_chrdev_region(gpioled.devid, GPIOLED_CNT, GPIOLED_NAME);
    186
            if (ret)
    187
              goto out1;
    188
          } else {
    189
            ret = alloc_chrdev_region(&gpioled.devid, 0, GPIOLED_CNT, GPIOLED_NAME);
    190
            if (ret)
    191
              goto out1;
    192
    193
            gpioled.major = MAJOR(gpioled.devid);
    194
            gpioled.minor = MINOR(gpioled.devid);
    195
          }
    196
    197
          printk("gpioled: major=%d,minor=%d\r\n",gpioled.major, gpioled.minor);
    198
          /* 初始化 cdev */
    199
    200
          gpioled.cdev.owner = THIS_MODULE;
    201
          cdev_init(&gpioled.cdev, &gpioled_fops);
    202
    203
          /* 添加一个 cdev */
    204
          ret = cdev_add(&gpioled.cdev, gpioled.devid, GPIOLED_CNT);
    205
          if (ret)
    206
            goto out2;
    207
    208
          /* 创建类 */
    209
          gpioled.class = class_create(THIS_MODULE, GPIOLED_NAME);
    210
         if (IS_ERR(gpioled.class)) {
    211
            ret = PTR_ERR(gpioled.class);
    212
            goto out3;
    213
          }
    214
    215
          /* 创建设备 */
    216
          gpioled.device = device_create(gpioled.class, NULL,
    217
                gpioled.devid, NULL, GPIOLED_NAME);
    218
         if (IS_ERR(gpioled.device)) {
    219
            ret = PTR_ERR(gpioled.device);
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php 220 goto out4; 221 } 222 223 return 0; 224 225 out4: 226 class_destroy(gpioled.class); 227 228 out3: 229 cdev_del(&gpioled.cdev); 230 231 out2: 232 unregister_chrdev_region(gpioled.devid, GPIOLED_CNT); 233 234 out1: 235 gpio_free(gpioled.led_gpio); 236 237 return ret; 238 } 239 240 static void __exit led_exit(void) 241 { 242 /* 注销设备 */ 243 device_destroy(gpioled.class, gpioled.devid); 244 245 /* 注销类 */ 246 class_destroy(gpioled.class); 247 248 /* 删除 cdev */ 249 cdev_del(&gpioled.cdev); 250 251 /* 注销设备号 */ 252 unregister_chrdev_region(gpioled.devid, GPIOLED_CNT); 253 254 /* 释放 GPIO */ 255 gpio_free(gpioled.led_gpio); 256 } 257 258 /* 驱动模块入口和出口函数注册 */ 259 module_init(led_init); 260 module_exit(led_exit);

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

261

262 MODULE_AUTHOR("DengTao <773904075@qq.com>");

263 MODULE_DESCRIPTION("Alientek ZYNQ GPIO LED Driver");

264 MODULE_LICENSE("GPL");

第26行,使用 include 包含 of_gpio.h 头文件,因为我们的代码中使用到了与 gpio 相关的 OF 函数。

正点原子

第 28~29 行, 定义 led 设备的设备号数量以及 led 设备的名字。

第 32~41 行,定义了 struct gpioled_dev 结构体,在第二十五章的基础上加入 led_gpio 用来 表示 GPIO 的编号。

第 80~98 行,使用 gpio_set_value 函数替代第二十五章直接操作 GPIO 寄存器的做法,那 么这个函数我们在前面一张给大家介绍过了。

第 125~129 行,跟之前是一样的,从设备树中获取 led 节点,包括后面读取"status"属性以及"compatible"属性。

第 153~157 行,重点在这里,使用 of_get_named_gpio 函数获取设备树中 led 节点指定的 gpio 管脚,得到一个 GPIO 编号,那么这个编号是 linux 内核 GPIO 子系统对 gpio 的编号,一个编号就对应一个 gpio,那么我们这里得到的这个编号就对应设备树中 led-gpio 指定的那个管 脚(gpio0 7---MIO7)。得到 GPIO 编号之后使用 gpio is valid 函数判断编号是否是有效编号。

第162~166行,通过gpio_request函数去申请对该gpio管脚的使用权。

第169行,通过gpio_direction_output 函数将gpio设置为输出模式。

第 172~179 行,根据 led 节点的"default-state"属性初始化 LED 的状态,使用 gpio_set_value 函数设置 gpio 输出高或低电平。

后面跟第二十五章就是一样的,需要注意的是,第 240~256 行,驱动模块的出口函数中 一定要把申请的 gpio 管脚给释放,通过 gpio_free 函数释放 gpio。

27.3.3 编写测试 App

本章直接使用第二十五章的测试 App,将第二十五章实验目录下的测试 APP 源程序 ledApp.c 以及编译好的可执行文件 ledApp 全部拷贝到本章实验目录下,这样就不用再次编译 ledApp.c 文件了。

27.4 运行测试

27.4.1 编译驱动程序和测试 App

1、编译驱动程序

编写 Makefile 文件,同样也是将第二十五章实验目录下的 Makefile 文件直接拷贝到本章 实验目录下,只是将 obj-m 变量的值改为 gpioled.o, Makefile 内容如下所示:

示例代码 27.4.1 Makefile 文件

```
1 KERN_DIR :=/home/zy/workspace/kernel-driver/linux-xlnx-rebase_v5.4_2020.2
2
3 obj-m :=gpioled.o
4
5 all:
```



原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php

6 make -C \$(KERN_DIR) M=`pwd` modules

7 clean:

8 make -C \$(KERN_DIR) M=`pwd` clean

第3行,设置 obj-m 变量的值为 gpioled.o。

修改完成之后保存退出,此时在我们的工程目录(5_gpioled)下有如下文件:

zy@zy-virtual-machine:~/linux/drivers/5_gpioled\$	
zy@zy-virtual-machine:~/linux/drivers/5_gpioled\$	ls
gpioled.c ledApp ledApp.c Makefile	
zy@zy-virtual-machine:~/linux/drivers/5 gpioled\$	
zy@zy-virtual-machine:~/linux/drivers/5_gpioled\$	

图 27.4.1 5_gpioled 目录下文件

输入如下命令编译出驱动模块文件:

make

编译成功以后就会生成一个名为"gpioled.ko"的驱动模块文件,如下所示:

zv@zv_virtual_machine/linux/drivers/5_gnioled\$
zygzy-virtuat-machine.~/tinux/drivers/5_gpicled\$
zy@zy-virtual-machine:~/linux/drivers/5_gpioled\$ make
make -C /home/zy/workspace/kernel-driver/linux-xlnx-xlnx_rebase_v5.4_2020.2 M=`p
wd` modules
make[1]: 进入目录"/home/zy/workspace/kernel-driver/linux-xlnx-xlnx rebase v5.4 2
020.2"
CC [M] /home/zy/linux/drivers/5 gpioled/gpioled.o
Building modules, stage 2.
MODPOST 1 modules
CC [M] /home/zy/linux/drivers/5 gpioled/gpioled.mod.o
LD [M] /home/zy/linux/drivers/5 gpioled/gpioled.ko
make[1]: 离开目录"/home/zy/workspace/kernel-driver/linux-xlnx-xlnx rebase v5.4 2
020.2"
zy@zy-virtual-machine:~/linux/drivers/5 gpioled\$
zy@zy-virtual-machine:~/linux/drivers/5 gpioled\$ ls
<pre>gpioled.c gpioled.mod.c ledApp modules.order</pre>
<pre>gpioled.ko gpioled.mod.o ledApp.c Module.symvers</pre>
<pre>gpioled.mod gpioled.o Makefile</pre>
zy@zy-virtual-machine:~/linux/drivers/5_gpioled\$

图 27.4.2 编译驱动模块

2、编译测试 APP

不用编译直接上一章的 ledApp 可执行文件即可!

27.4.2 运行测试

将上面编译出来的 gpioled.ko 和 ledApp 这两个文件拷贝到 NFS 共享目录下根文件系统的 /lib/modules/5.4.0-xilinx 文件夹中。

复制 system.dtb 到 SD 卡 FAT 分区,然后将 SD 启动卡插到开发板中,连接开发板串口、网口和电源线,设置拨码开关为 SD 卡启动模式,然后打开电源,启动开发板。

系统启动完成后,通过串口终端进入"/lib/modules/5.4.0-xilinx"目录下,输入如下命令 加载 dtsled.ko 驱动模块:

depmod //第一次加载驱动的时候需要运行此命令

正点原子 领航者 ZYNQ 之嵌入式 Linux 开发指南 原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php modprobe dtsled.ko //加载驱动 驱动加载成功以后会在终端中输出一些信息,如下图所示: root@ALIENTEK-ZYNQ-driver:~# root@ALIENTEK-ZYNQ-driver:~# root@ALIENTEK-ZYNQ-driver:~# cd /lib/modules/5.4.0-xilinx root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx# root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx# ls gpioled.ko ledApp root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx# root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx# depmod root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx# root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx# modprobe gpioled.ko ppioled: loading out-of-tree module taints kernel. pioled: device matching successful! gpioled: led-gpio num = 905 pioled: major=244,minor=0 root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx# root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx#

图 27.4.3 加载 gpioled 驱动模块

从上图中可以看出,通过 of_get_named_gpio 函数得到 gpio0_7(MIO7) 这个 GPIO 的编 号为 905。看起来这个数字有点大哈,不过没关系,能用就行!驱动加载成功以后就可以使 用 ledApp 软件来测试驱动是否工作正常,输入如下命令打开 LED 灯:

./ledApp /dev/gpioled 0 //关闭 LED 灯

输入上述命令以后查看开发板上的 PS_LED0 灯是否熄灭,如果熄灭的话说明驱动工作正常。在输入如下命令打开 LED 灯:

./ledApp /dev/gpioled 1 //打开 LED 灯 输入上述命令以后查看开发板上的 PS_LED0 灯是否点亮。 如果要卸载驱动的话输入如下命令即可: rmmod gpioled.ko



正点原子

第二十八章 Linux 蜂鸣器驱动实验

上一章实验中我们借助 gpio 子系统编写了 LED 灯驱动,领航者开发板上还有一个蜂鸣器, 从软件的角度考虑,蜂鸣器驱动和 LED 灯驱动其实是一摸一样的,都是控制 IO 输出高低电 平。本章我们就来学习编写蜂鸣器的 Linux 驱动,也算是对上一章讲解的 gpio 子系统的巩固。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

28.1 有源蜂鸣器简介

蜂鸣器常用于计算机、打印机、报警器、电子玩具等电子产品中,常用的蜂鸣器有两种: 有源蜂鸣器和无源蜂鸣器,这里的有"源"不是电源,而是震荡源,有源蜂鸣器内部带有震 荡源,所以有源蜂鸣器只要通电就会叫。无源蜂鸣器内部不带震荡源,直接用直流电是驱动 不起来的, 需要 2K-5K 的方波去驱动。领航者开发板使用的是有源蜂鸣器, 因此只要给其供 电就会工作,开发板所使用的有源蜂鸣器如下图所示:



图 28.1.1 有源蜂鸣器

有源蜂鸣器只要通电就会叫,所以我们可以做一个供电电路,这个供电电路可以由一个 IO 来控制其通断,一般使用三极管来搭建这个电路。为什么我们不能像控制 LED 灯一样,直 接将 GPIO 接到蜂鸣器的负极,通过 IO 输出高低来控制蜂鸣器的通断。因为蜂鸣器工作的电 流比 LED 灯要大, 直接将蜂鸣器接到开发板的 GPIO 上有可能会烧毁 IO, 所以我们需要通过 一个三极管来间接的控制蜂鸣器的通断,相当于加了一层隔离。本章我们就驱动开发板上的 有源蜂鸣器,使其周期性的"滴、滴、滴、流…"鸣叫。

本节我们来看一下如果在 Linux 下编写蜂鸣器驱动需要做哪些工作:

① 在设备树中创建蜂鸣器节点 beeper;

② 在蜂鸣器节点 beeper 中指定 gpio;

③ 编写驱动程序和测试 APP,和第二十六章的 LED 驱动程序和测试 APP 基本一样。

接下来我们就根据上面这三步来编写蜂鸣器 Linux 驱动程序。

28.2 硬件原理图分析

打开领航者底板原理图文件,找到蜂鸣器电路原理图,如下所示:



原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php



图 28.2.1 蜂鸣器原理图

上图中通过一个 NPN 型的三极管 S8050 来驱动蜂鸣器,通过 BEEP 这个 IO 来控制三极管 Q1 的导通,当 BEEP 输出高电平的时候 Q1 导通,相当于蜂鸣器的负极连接到 GND,蜂鸣器 形成一个通路,因此蜂鸣器会鸣叫。同理,当 BEEP 输出低电平的时候 Q1 不导通,那么蜂鸣器就没有形成一个通路,因此蜂鸣器也就不会鸣叫。

在原理图中搜索 "BEEP" 标号,可知 BEEP 连接到了 ZYNQ Bank35 的 M14 引脚,如下 图所示:

B35 IOO GBC KEY G14 J2	46 46
B35 IO25 UART3 TX J15 J2	48 40
B35 L11 N LCD RST L17 J2	50 50
B35 L11 P CAN RX L16 J2	52 52
B35 L20 N CAN TX J14 J2	54 54
B35 L20 P UART2 RX K14 J2	56 56
B35 L23 N UART2 TX M15 J2	58 58
B35 L23 P BEEP M14 J2	<u>60</u>

图 28.2.2 BEEP 管脚

M14 并不是 PS 端 MIO 引脚,而是 PL 端的 IO 引脚,但是 PS 可以通过 EMIO 来连接 PL 端引脚,关于 MIO 和 EMIO 的详细内容请大家阅读《领航者 ZYNQ 之嵌入式开发指南》的第二和第三章的内容,这里就不给大家做过多的介绍。

本篇驱动开发篇使用的 xsa 文件对应的 vivado 工程,是笔者配置的,使能了 EMIO,并将 M14 引脚绑定到了 EMIO,如下所示:

	Direction	Neg Diff	Package Pin		Fixed	Bank	I/O Std		Vcco	Vref	Drive Stre	ngth
gpio0_tri_io[6]	INOUT		U19	~	~	34	LVCMOS33*	*	3.300		12	~
gpio0_tri_io[5]	INOUT		M19	~	~	35	LVCMOS33*	~	3.300		12	~
gpio0_tri_io[4]	INOUT		M14	~	~	35	LVCMOS33*		3.300		12	~
gpio0_tri_io[3]	INOUT		F16	~	~	35	LVCMOS33*	~	3.300		12	~
gpio0_tri_io[2]	INOUT		K16	~	~	35	LVCMOS33*	~	3.300		12	~
gpio0_tri_io[1]	INOUT		L14	~	~	35	LVCMOS33*	~	3.300		12	~
gpio0_tri_io[0]	INOUT		J16	~	~	35	LVCMOS33*	*	3.300		12	~

图 28.2.3 EMIO 连接 M14 引脚



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

所以由上面可以知道, M14 连接到了 EMIO 的第 5 个引脚 emio[4], 由于 EMIO 对应的 GPIO 起始编号是从 54 开始的,所以由此可知 emio[4]对应的就是 GPIO 58 (54 + 4),关于 MIO 和 EMIO 编号的问题在《领航者 ZYNQ 之嵌入式开发指南》的第二和第三章有详细的说 明。

所以目标很明确了,蜂鸣器就是通过 GPIO 58 控制的,那么在我们的驱动代码当中就是 对 GPIO 58 进行控制从而控制蜂鸣器鸣叫和关闭。

28.3 实验程序编写

本实验对应的例程路径为:开发板资料盘(A 盘)\4_SourceCode\3_Embedded_Linux\Linux 驱动例程\6_beeper。

本章也是直接在前面的实验当中进行修改。

28.3.1 修改设备树文件

打开 system-user.dtsi 文件,在根节点"/"下创建一个名为 beeper 的节点,节点内容如下:示例代码 28.3.1 system-user.dtsi 蜂鸣器节点 beeper

27 beeper{

```
28 compatible = "alientek,beeper";
```

- **29** status = "okay";
- **30 default**-state = "off";
- 31 beeper-gpio = <&gpio0 58 GPIO_ACTIVE_HIGH>;

32 };

第 28 行,设置 beeper 节点的 compatible 属性为 "alientek, beeper",我们在驱动代码中会 区匹配这个属性的值。

第 30 行,将 default-state 属性的值设置为 "off",我们在代码中会根据 default-state 属性的值来设置蜂鸣器的初始化状态。

第31行, beeper-gpio 属性指定所需的 GPIO。

设备树编写完成以后使用下面这条命令重新编译设备树,如下所示:



图 28.3.1 重新编译设备树

将编译出来的 system-top.dtb 文件重命名为 system.dtb, 然后将 system.dtb 文件拷贝到开发 板 SD 启动卡的 FAT 分区, 替换旧的 system.dtb 文件。替换成功之后重新启动成功开发板, 以 后进入"/proc/device-tree"目录中查看"beeper"节点是否存在,如果存在的话就说明设备树 基本修改成功(具体还要驱动验证),结果如下图所示:



图 28.3.2 设备树 beeper 节点

28.3.2 编写蜂鸣器驱动程序

设备树准备好以后就可以编写驱动程序了,在我们的 drivers 目录下新建名为"6_beeper"的文件夹,然后在 6_beeper 文件中新建 beeper.c 文件,在 beeper.c 里面输入如下内容:示例代码 28.3.2 beeper.c 文件内容

1 /******	*****	**********	*****				
2 Copyrigh	2 Copyright © ALIENTEK Co., Ltd. 1998-2029. All rights reserved.						
3 文件名	: beeper.c						
4 作者	:邓涛						
5 版本	: V1.0						
<mark>6</mark> 描述	:领航者开发板蜂鸣	器驱动文件。					
<mark>7</mark> 其他	: 无						
<mark>8</mark> 论坛	: www.openedv.com						
<mark>9</mark> 日志	: 初版 V1.0 2019/1/30)邓涛创建					
10 ******	******	********	***************************************				
11							
12 #include	linux/types.h>						
13 #include	<linux kernel.h=""></linux>						
14 #include	linux/delay.h>						
15 #include	linux/ide.h>						
16 #include	<linux init.h=""></linux>						
17 #include	linux/module.h>						
18 #include	linux/errno.h>						
19 #include	linux/gpio.h>						
20 #include	<asm mach="" map.h=""></asm>						
21 #include	<asm uaccess.h=""></asm>						
22 #include	<asm io.h=""></asm>						
23 #include	linux/cdev.h>						
24 #include	linux/of.h>						
25 #include	linux/of_address.h>						
26 #include	linux/of_gpio.h>						
27							
28 #define H	BEEPER_CNT	1	/* 设备号个数 */				
29 #define H	BEEPER_NAME	"beeper"	/* 名字 */				
30							
31 /* dtsled	设备结构体 */						



```
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
    32 struct beeper_dev {
                          /* 设备号 */
    33 dev_t devid;
    34 struct cdev cdev;
                        /* cdev */
    35 struct class *class; /* 类 */
    36 struct device *device; /* 设备 */
                         /* 主设备号 */
    37 int major;
    38 int minor;
                         /* 次设备号 */
    39
       struct device_node *nd;/* 设备节点 */
                         /* LED 所使用的 GPIO 编号 */
    40 int gpio;
   41 };
    42
   43 static struct beeper_dev beeper; /* led 设备 */
   44
   45 /*
   46 * @description
                             :打开设备
   47 * @param – inode
                             :传递给驱动的 inode
   48 * @param – filp
                             :设备文件, file 结构体有个叫做 private_data 的成员变量
    49 *
                              一般在 open 的时候将 private_data 指向设备结构体。
                             :0 成功;其他失败
    50 * @return
    51 */
    52 static int beeper_open(struct inode *inode, struct file *filp)
    53 {
    54 return 0;
    55 }
    56
    57 /*
                             :从设备读取数据
   58 * @description
   59 * @param – filp
                             :要打开的设备文件(文件描述符)
    60 * @param – buf
                             :返回给用户空间的数据缓冲区
    61 * @param – cnt
                             :要读取的数据长度
    62 * @param – offt
                             :相对于文件首地址的偏移
                             :读取的字节数,如果为负值,表示读取失败
    63 * @return
    64 */
    65 static ssize_t beeper_read(struct file *filp, char __user *buf,
            size_t cnt, loff_t *offt)
    66
    67 {
    68 return 0;
    69 }
    70
    71 /*
    72 * @description : 向设备写数据
```



```
原子哥在线教学: www.yuanzige.com
                                      论坛:www.openedv.com/forum.php
    73 * @param – filp
                              :设备文件,表示打开的文件描述符
                              :要写给设备写入的数据
   74 * @param – buf
    75 * @param – cnt
                              :要写入的数据长度
    76 * @param – offt
                              :相对于文件首地址的偏移
                              :写入的字节数,如果为负值,表示写入失败
    77 * @return
    78 */
    79 static ssize_t beeper_write(struct file *filp, const char __user *buf,
    80
            size_t cnt, loff_t *offt)
    81 {
    82
        int ret;
    83
        char kern_buf[1];
    84
        ret = copy_from_user(kern_buf, buf, cnt); // 得到应用层传递过来的数据
    85
       if(0 > ret) \{
    86
    87
          printk(KERN_ERR "kernel write failed!\r\n");
    88
          return -EFAULT;
    89
       }
    90
    91
        if (0 == kern_buf[0])
          gpio_set_value(beeper.gpio, 0); // 如果传递过来的数据是 0 则关闭 led
    92
    93
        else if (1 == kern_buf[0])
          gpio_set_value(beeper.gpio, 1); // 如果传递过来的数据是1则点亮 led
    94
    95
    96 return 0;
    97 }
    98
    99 /*
   100 * @description
                              :关闭/释放设备
   101 * @param – filp
                              :要关闭的设备文件(文件描述符)
                              :0 成功;其他 失败
   102 * @return
   103 */
   104 static int beeper_release(struct inode *inode, struct file *filp)
   105 {
   106 return 0;
   107 }
   108
   109 /* 设备操作函数 */
   110 static struct file_operations beeper_fops = {
   111 .owner = THIS_MODULE,
   112 .open = beeper_open,
   113 .read = beeper_read,
```



```
原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php
    114
          .write = beeper_write,
    115
          .release = beeper_release,
    116 };
    117
    118 static int __init beeper_init(void)
    119 {
    120
          const char *str;
    121
          int ret;
    122
    123
         /* 1.获取 beeper 设备节点 */
    124
          beeper.nd = of_find_node_by_path("/beeper");
    125
         if(NULL == beeper.nd) {
    126
            printk(KERN_ERR "beeper: Failed to get beeper node\n");
    127
            return -EINVAL;
    128
         }
    129
    130
         /* 2.读取 status 属性 */
    131
          ret = of_property_read_string(beeper.nd, "status", &str);
    132
         if(!ret) {
    133
            if (strcmp(str, "okay"))
    134
              return -EINVAL;
    135
          }
    136
    137
         /* 2、获取 compatible 属性值并进行匹配 */
    138
          ret = of_property_read_string(beeper.nd, "compatible", &str);
    139
          if(0 > ret) {
    140
            printk(KERN_ERR "beeper: Failed to get compatible property\n");
    141
            return ret;
    142
          }
    143
    144
          if (strcmp(str, "alientek, beeper")) {
            printk(KERN_ERR "beeper: Compatible match failed\n");
    145
    146
            return -EINVAL;
          }
    147
    148
    149
          printk(KERN_INFO "beeper: device matching successful!\r\n");
    150
          /* 4.获取设备树中的 beeper-gpio 属性,得到蜂鸣器所使用的 GPIO 编号 */
    151
    152
          beeper.gpio = of_get_named_gpio(beeper.nd, "beeper-gpio", 0);
    153
          if(!gpio_is_valid(beeper.gpio)) {
    154
            printk(KERN_ERR "beeper: Failed to get beeper-gpio\n");
```



```
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
    155
            return -EINVAL;
    156
          }
    157
    158
          printk(KERN_INFO "beeper: beeper-gpio num = %d\r\n", beeper.gpio);
    159
         /* 5.向 gpio 子系统申请使用 GPIO */
    160
    161
          ret = gpio_request(beeper.gpio, "Beeper gpio");
    162
         if (ret) {
    163
            printk(KERN_ERR "beeper: Failed to request gpio num %d\n", beeper.gpio);
    164
            return ret;
    165
         }
    166
         /* 6.将 gpio 管脚设置为输出模式 */
    167
    168
          gpio_direction_output(beeper.gpio, 0);
    169
         /* 7.设置蜂鸣器的初始状态 */
    170
    171
         ret = of_property_read_string(beeper.nd, "default-state", &str);
    172
         if(!ret) {
    173
            if (!strcmp(str, "on"))
    174
              gpio_set_value(beeper.gpio, 1);
    175
            else
    176
              gpio_set_value(beeper.gpio, 0);
    177
          } else
    178
            gpio_set_value(beeper.gpio, 0);
    179
    180
         /* 8.注册字符设备驱动 */
          /* 创建设备号 */
    181
    182
         if (beeper.major) {
    183
            beeper.devid = MKDEV(beeper.major, 0);
    184
            ret = register_chrdev_region(beeper.devid, BEEPER_CNT, BEEPER_NAME);
    185
            if (ret)
    186
              goto out1;
    187
          } else {
    188
            ret = alloc_chrdev_region(&beeper.devid, 0, BEEPER_CNT, BEEPER_NAME);
    189
            if (ret)
    190
              goto out1;
    191
    192
            beeper.major = MAJOR(beeper.devid);
    193
            beeper.minor = MINOR(beeper.devid);
    194
          }
    195
```



```
原子哥在线教学: www.yuanzige.com
                                              论坛:www.openedv.com/forum.php
          printk("beeper: major=%d,minor=%d\r\n", beeper.major, beeper.minor);
    196
    197
    198
         /* 初始化 cdev */
    199
          beeper.cdev.owner = THIS_MODULE;
          cdev_init(&beeper.cdev, &beeper_fops);
    200
    201
         /* 添加一个 cdev */
    202
    203
         ret = cdev_add(&beeper.cdev, beeper.devid, BEEPER_CNT);
    204
         if (ret)
    205
            goto out2;
    206
         /* 创建类 */
    207
    208
         beeper.class = class_create(THIS_MODULE, BEEPER_NAME);
         if (IS_ERR(beeper.class)) {
    209
    210
            ret = PTR_ERR(beeper.class);
    211
            goto out3;
    212
         }
    213
    214
          /* 创建设备 */
    215
          beeper.device = device_create(beeper.class, NULL,
    216
                beeper.devid, NULL, BEEPER_NAME);
    217
          if (IS_ERR(beeper.device)) {
            ret = PTR_ERR(beeper.device);
    218
    219
            goto out4;
    220
          }
    221
    222
          return 0;
    223
    224 out4:
    225
          class_destroy(beeper.class);
    226
    227 out3:
    228
          cdev_del(&beeper.cdev);
    229
    230 out2:
    231
          unregister_chrdev_region(beeper.devid, BEEPER_CNT);
    232
    233 out1:
    234
          gpio_free(beeper.gpio);
    235
    236
         return ret;
```



原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php 237 } 238 239 static void __exit beeper_exit(void) 240 { 241 /* 注销设备 */ 242 device destroy(beeper.class, beeper.devid); 243 244 /* 注销类 */ 245 class_destroy(beeper.class); 246 247 /* 删除 cdev */ 248 cdev_del(&beeper.cdev); 249 250 /* 注销设备号 */ 251 unregister_chrdev_region(beeper.devid, BEEPER_CNT); 252 253 /* 释放 GPIO */ 254 gpio_free(beeper.gpio); 255 } 256 257 /* 驱动模块入口和出口函数注册 */ 258 module_init(beeper_init); 259 module_exit(beeper_exit); 260 261 MODULE_AUTHOR("DengTao <773904075@qq.com>"); 262 MODULE_DESCRIPTION("Alientek ZYNQ GPIO Beeper Driver"); 263 MODULE_LICENSE("GPL"); beeper.c 中的内容和上一章的 gpioled.c 中的内容基本一样,基本上就是改了下命名。

28.3.3 编写测试 APP

测试 APP 程序直接将第二十七章工程目录下的 ledApp.c 文件拷贝到当前实验目录,将其重命名为 beeperApp.c 即可。

28.4 运行测试

28.4.1 编译驱动程序和测试 APP

1、编译驱动程序

编写 Makefile 文件,直接将上一章实验目录下的 Makefile 文件拷贝到本实验目录,修改 Makefile 文件,将 obj-m 变量的值改为 beeper.o,修改完成之后 Makefile 内容如下所示:示例代码 28.4.1 Makefile 文件内容

1 KERN_DIR :=/home/zy/workspace/kernel-driver/linux-xlnx-xlnx_rebase_v5.4_2020.2



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

```
3 obj-m :=beeper.o
```

```
4
```

2

5 all:

6 make -C \$(KERN_DIR) M=`pwd` modules

7 clean:

8 make -C \$(KERN_DIR) M=`pwd` clean

第3行,设置 obj-m 变量的值为 beeper.o。

修改完成之后保存退出, 输入如下命令编译出驱动模块文件:

make

编译成功以后就会生成一个名为"beeper.ko"的驱动模块文件。

2、编译测试 APP

输入如下命令编译测试 beeperApp.c 这个测试程序:

\$CC beeperApp.c -o beeperApp

编译成功以后就会生成 beeperApp 这个应用程序,最终在本章实验实验目录下有如下文件:

zy@zy-virtual	L-machine:/mr	1t/hgfs/share18/1	Linux驱动例程/6_beeper\$	
zy@zy-virtual	L-machine:/mr	nt/hgfs/share18/1	Linux驱动例程/6_beeper\$	<pre>\$CC beeperApp.c</pre>
-o beeperApp	—			
zy@zy-virtual	L-machine:/mr	nt/hgfs/share18/1	Linux驱动例程/6 beeper\$	
zy@zy-virtual	L-machine:/mr	nt/hgfs/share18/1	Linux驱动例程/6 beeper\$	ls
beeperApp	beeper.ko	beeper.mod.o	modules.order	
beeperApp.c	beeper.mod	beeper.o	Module.symvers	
beeper.c	beeper.mod.c	c Makefile	system-user.dtsi	
zy@zy-virtual	-machine:/mr	nt/hgfs/share18/1	Linux驱动例程/6_beeper\$	

图 28.4.1 beeperApp 和 beeper.ko 文件

28.4.2 运行测试

将上面编译出来的 beeper.ko 和 beeperApp 这两个文件拷贝到 NFS 共享目录下根文件系统的/lib/modules/5.4.0-xilinx 文件夹中。

复制 system.dtb 到 SD 卡 FAT 分区,然后将 SD 启动卡插到开发板中,连接开发板串口、网口和电源线,设置拨码开关为 SD 卡启动模式,然后打开电源,启动开发板。

系统启动完成后,通过串口终端进入"/lib/modules/5.4.0-xilinx"目录下,输入如下命令 加载 dtsled.ko 驱动模块:

depmod//第一次加载驱动的时候需要运行此命令modprobe dtsled.ko//加载驱动驱动加载成功以后会在终端中输出一些信息,如下图所示:

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

A HEADY A HAND A
root@ALIENTEK-ZYNQ-driver:~#
root@ALIENTEK-ZYNQ-driver:~#
root@ALIENTEK-ZYNQ-driver:~# cd /lib/modules/5.4.0-xilinx
root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx#
root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx# ls
beeper.ko beeperApp
root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx#
root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx# depmod
root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx#
root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx# modprobe beeper.ko
beeper: loading out-of-tree module taints kernel.
beeper: device matching successful!
beeper: beeper-gpio num = 956
beeper: major=244,minor=0
root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx#
root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx#

E点原子

图 28.4.2 加载 beeper.ko 驱动模块

从上图中可以看出,在内核驱动中获取到的 GPIO0_58 的编号为 956,使用 beeperApp 软件来测试驱动是否工作正常,输入如下命令打开蜂鸣器:

./beeperApp /dev/beeper 1 //打开蜂鸣器

执行上述命令之后,开发板的蜂鸣器是会鸣叫的,如果鸣叫的话说明驱动工作正常。在 输入如下命令关闭蜂鸣器:

./beeperApp /dev/beeper 0 //关闭蜂鸣器

执行上述命令后,开发板的蜂鸣器会停止鸣叫。如果要卸载驱动的话输入如下命令即可: rmmod beeper.ko



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

第二十九章 Linux 并发与竞争

Linux 是一个多任务操作系统,肯定会存在多个任务共同操作同一段内存或者设备的情况, 多个任务甚至中断都能访问的资源叫做共享资源,就和共享单车一样。在驱动开发中要注意 对共享资源的保护,也就是要处理对共享资源的并发访问。比如共享单车,大家按照谁扫谁 骑走的原则来共用这个单车,如果没有这个并发访问共享单车的原则存在,只怕到时候为了 一辆单车要打起来了。在 Linux 驱动编写过程中对于并发控制的管理非常重要,本章我们就 来学习一下如何在 Linux 驱动中处理并发。



原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php

29.1 并发与竞争

1、并发与竞争简介

并发就是多个"用户"同时访问同一个共享资源,比如你们公司有一台打印机,你们公 司的所有人都可以使用。现在小李和小王要同时使用这一台打印机,都要打印一份文件。小 李要打印的文件内容如下:

示例代码 29.1.1 小李要打印的内容

我叫小李

电话: 123456

工号: 16

小王要打印的内容如下:

示例代码 29.1.2 小王要打印的内容

我叫小王

电话: 678910

工号: 20

这两份文档肯定是各自打印出来的,不能相互影响。当两个人同时打印的话如果打印机 不做处理的话可能会出现小李的文档打印了一行,然后开始打印小王的文档,这样打印出来 的文档就错乱了,可能会出现如下的错误文档内容:

示例代码 29.1.3 小王打印出来的错误文档

我叫小王

电话: 123456

工号: 20

可以看出,小王打印出来的文档中电话号码错误了,变成小李的了,这是绝对不允许的。 如果有多人同时向打印机发送了多份文档,打印机必须保证一次只能打印一份文档,只有打 印完成以后才能打印其他的文档。

Linux 系统是个多任务操作系统,会存在多个任务同时访问同一片内存区域,这些任务可 能会相互覆盖这段内存中的数据,造成内存数据混乱。针对这个问题必须要做处理,严重的 话可能会导致系统崩溃。现在的 Linux 系统并发产生的原因很复杂,总结一下有下面几个主 要原因:

①、多线程并发访问, Linux 是多任务(线程)的系统, 所以多线程访问是最基本的原因。

②、抢占式并发访问,从 2.6 版本内核开始,Linux 内核支持抢占,也就是说调度程序可 以在任意时刻抢占正在运行的线程,从而运行其他的线程。

③、中断程序并发访问,这个无需多说,学过 STM32 的同学应该知道,硬件中断的权利 可是很大的。

④、SMP(多核)核间并发访问,现在 ARM 架构的多核 SOC 很常见,多核 CPU 存在核间 并发访问。

并发访问带来的问题就是竞争,学过 FreeRTOS 和 UCOS 的同学应该知道临界区这个概 念,所谓的临界区就是共享数据段,对于临界区必须保证一次只有一个线程访问,也就是要 保证临界区是原子访问的,注意这里的"原子"不是正点原子的"原子"。我们都知道,原 子化学反应不可再分的基本微粒,这里的原子访问就表示这一个访问是一个步骤,不能再进 行拆分。如果多个线程同时操作临界区就表示存在竞争,我们在编写驱动的时候一定要注意



原子哥在线教学:www.yuanzige.com 论坛:www.openedv.com/forum.php 避免并发和防止竞争访问。很多 Linux 驱动初学者往往不注意这一点,在驱动程序中埋下了 隐患,这类问题往往又很不容易查找,导致驱动调试难度加大、费时费力。所以我们一般在 编写驱动的时候就要考虑到并发与竞争,而不是驱动都编写完了然后再处理并发与竞争。

2、保护内容是什么

前面一直说要防止并发访问共享资源,换句话说就是要保护共享资源,防止进行并发访问。那么问题来了,什么是共享资源?现实生活中的公共电话、共享单车这些是共享资源,我们都很容易理解,那么在程序中什么是共享资源?也就是保护的内容是什么?我们保护的不是代码,而是数据!某个线程的局部变量不需要保护,我们要保护的是多个线程都会访问的共享数据。一个整形的全局变量 a 是数据,一份要打印的文档也是数据,虽然我们知道了要对共享数据进行保护,那么怎么判断哪些共享数据要保护呢?找到要保护的数据才是重点,而这个也是难点,因为驱动程序各不相同,那么数据也千变万化,一般像全局变量,设备结构体这些肯定是要保护的,至于其他的数据就要根据实际的驱动程序而定了。

当我们发现驱动程序中存在并发和竞争的时候一定要处理掉,接下来我们依次来学习一下 Linux 内核提供的几种并发和竞争的处理方法。

29.2 原子操作

29.2.1 原子操作简介

首先看一下原子操作,原子操作就是指不能在进一步分割的操作,一般原子操作用于变量或者位操作。假如现在要对无符号整形变量 a 赋值,值为 3,对于 C 语言来讲很简单,直接就是:

a = 3

但是 C 语言要先编译为成汇编指令,ARM 架构不支持直接对寄存器进行读写操作,比如 要借助寄存器 R0、R1 等来完成赋值操作。假设变量 a 的地址为 0X3000000, "a=3"这一行 C 语言可能会被编译为如下所示的汇编代码:

示例代码 29.2.1 汇编示例代码

1 ldr r0, =0X3000000	/* 变量 a 地址 */
2 ldr r1, = 3	/* 要写入的值 */
3 str r1, [r0]	/* 将3写入到a变量中*;

示例代码 29.2.1 只是一个简单的举例说明,实际的结果要比示例代码复杂的多。从上述 代码可以看出,C语言里面简简单单的一句"a=3",编译成汇编文件以后变成了3句,那么 程序在执行的时候肯定是按照示例代码29.2.1 中的汇编语句一条一条的执行。假设现在线程 A要向 a 变量写入10这个值,而线程 B 也要向 a 变量写入20这个值,我们理想中的执行顺序 如下图所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

线程A	线程B
ldr r0, =0X3000000	
ldr r1, = 10	
str r1, [r0]	
	ldr r0, =0X3000000
	ldr r1, = 20
	str r1, [r0]



按照图 29.2.2 所示的流程,确实可以实现线程 A 将 a 变量设置为 10,线程 B 将 a 变量设 置为 20。但是实际上的执行流程可能如下图所示:

线程A	线程B
ldr r0, =0X30000000	
ldr r1, = 10	ldr r0, =0X30000000
	ldr r1, = 20
str r1, [r0]	
	str r1, [r0]

图 29.2.3 可能的执行流程

按照上图所示的流程,线程 A 最终将变量 a 设置为了 20,而并不是要求的 10!线程 B 没 有问题。这就是一个最简单的设置变量值的并发与竞争的例子,要解决这个问题就要保证示 例代码 29.2.1 中的三行汇编指令作为一个整体运行,也就是作为一个原子存在。Linux 内核提 供了一组原子操作 API 函数来完成此功能, Linux 内核提供了两组原子操作 API 函数, 一组是 对整形变量进行操作的,一组是对位进行操作的,我们接下来看一下这些 API 函数。

29.2.2 原子整形操作 API 函数

Linux 内核定义了叫做 atomic t 的结构体来完成整形数据的原子操作,在使用中用原子变 量来代替整形变量,此结构体定义在 include/linux/types.h 文件中,定义如下:

示例代码 29.2.2 atomic t 结构体

175 typedef struct { 176 int counter; 177 } atomic_t; 如果要使用原子操作 API 函数,首先要先定义一个 atomic_t 的变量,如下所示: //定义 a atomic t a; 也可以在定义原子变量的时候给原子变量赋初值,如下所示: atomic_t b = ATOMIC_INIT(0); //定义原子变量 b 并赋初值为 0


原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

可以通过宏 ATOMIC_INIT 向原子变量赋初值。

原子变量有了,接下来就是对原子变量进行操作,比如读、写、增加、减少等等,Linux 内核提供了大量的原子操作 API 函数,如下表所示:

函数	描述
ATOMIC_INIT(int i)	定义原子变量的时候对其初始化。
int atomic_read(atomic_t *v)	读取 v 的值,并且返回。
<pre>void atomic_set(atomic_t *v, int i)</pre>	向v写入i值。
void atomic_add(int i, atomic_t *v)	给v加上i值。
void atomic_sub(int i, atomic_t *v)	从v减去i值。
<pre>void atomic_inc(atomic_t *v)</pre>	给 v 加 1, 也就是自增。
<pre>void atomic_dec(atomic_t *v)</pre>	从 v 减 1, 也就是自减
int atomic_dec_return(atomic_t *v)	从 v 减 1,并且返回 v 的值。
int atomic_inc_return(atomic_t *v)	给 v 加 1,并且返回 v 的值。
int atomic_sub_and_test(int i, atomic_t *v)	从 v 减 i,如果结果为 0 就返回真,否 则返回假
<pre>int atomic_dec_and_test(atomic_t *v)</pre>	从 v 减 1,如果结果为 0 就返回真,否 则返回假
<pre>int atomic_inc_and_test(atomic_t *v)</pre>	给 v 加 1, 如果结果为 0 就返回真, 否则返回假
int atomic_add_negative(int i, atomic_t *v)	给 v 加 i,如果结果为负就返回真,否则返回假

表 29.2.1 原子整形操作 API 函数表

如果使用 64 位的 SOC 的话,就要用到 64 位的原子变量,Linux 内核也定义了 64 位原子 结构体,如下所示:

示例代码 29.2.3 atomic64_t 结构体

typedef struct {

long long counter;

} atomic64_t;

相应的也提供了 64 位原子变量的操作 API 函数,这里我们就不详细讲解了,和表 29.2.1 中的 API 函数有用法一样,只是将"atomic_"前缀换为"atomic64_",将 int 换为 long long。 如果使用的是 64 位的 SOC,那么就要使用 64 位的原子操作函数。ZYNQ 是 32 位的架构,所 以本书中只使用表 29.2.1 中的 32 位原子操作函数。原子变量和相应的 API 函数使用起来很简 单,参考如下示例:

示任	列代码 29.2.4 原子变量和 API 函数使用
<pre>atomic_t v = ATOMIC_INIT(0);</pre>	/* 定义并初始化原子变零 v=0 */
atomic_set(&v, 10);	/* 设置 v=10 */
atomic_read(&v);	/* 读取 v 的值, 肯定是 10 */



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php atomic_inc(&v); /* v 的值加 1, v=11 */

29.2.3 原子位操作 API 函数

位操作也是很常用的操作,Linux内核也提供了一系列的原子位操作 API 函数,只不过原子位操作不像原子整形变量那样有个 atomic_t 的数据结构,原子位操作是直接对内存进行操作,API 函数如下表所示:

函数	描述
void set_bit(int nr, void *p)	将 p 地址的第 mr 位置 1。
<pre>void clear_bit(int nr,void *p)</pre>	将 p 地址的第 nr 位清零。
void change_bit(int nr, void *p)	将p地址的第nr位进行翻转。
int test_bit(int nr, void *p)	获取 p 地址的第 nr 位的值。
int test_and_set_bit(int nr, void	将 p 地址的第 nr 位置 1,并且返回 nr 位
*p)	原来的值。
int test_and_clear_bit(int nr,	将 p 地址的第 nr 位清零,并且返回 nr 位
void *p)	原来的值。
int test_and_change_bit(int nr,	将 p 地址的第 nr 位翻转,并且返回 nr 位
void *p)	原来的值。

表 29.2.2 原子位操作函数表

29.3 自旋锁

29.3.1 自旋锁简介

原子操作只能对整形变量或者位进行保护,但是,在实际的使用环境中怎么可能只有整 形变量或位这么简单的临界区。举个最简单的例子,设备结构体变量就不是整型变量,我们 对于结构体中成员变量的操作也要保证原子性,在线程 A 对结构体变量使用期间,应该禁止 其他的线程来访问此结构体变量,这些工作原子操作都不能胜任,需要本节要讲的锁机制, 在 Linux 内核中就是自旋锁。

当一个线程要访问某个共享资源的时候首先要先获取相应的锁,锁只能被一个线程持有, 只要此线程不释放持有的锁,那么其他的线程就不能获取此锁。对于自旋锁而言,如果自旋 锁正在被线程A持有,线程B想要获取自旋锁,那么线程B就会处于忙循环-旋转-等待状态, 线程 B 不会进入休眠状态或者说去做其他的处理,而是会一直傻傻的在那里"转圈圈"的等 待锁可用。比如现在有个公用电话亭,一次肯定只能进去一个人打电话,现在电话亭里面有 人正在打电话,相当于获得了自旋锁。此时你到了电话亭门口,因为里面有人,所以你不能 进去打电话,相当于没有获取自旋锁,这个时候你肯定是站在原地等待,你可能因为无聊的 等待而转圈圈消遣时光,反正就是哪里也不能去,要一直等到里面的人打完电话出来。终于, 里面的人打完电话出来了,相当于释放了自旋锁,这个时候你就可以使用电话亭打电话了, 相当于获取到了自旋锁。

自旋锁的"自旋"也就是"原地打转"的意思,"原地打转"的目的是为了等待自旋锁可以用,可以访问共享资源。把自旋锁比作一个变量 a,变量 a=1 的时候表示共享资源可用,当 a=0 的时候表示共享资源不可用。现在线程 A 要访问共享资源,发现 a=0(自旋锁被其他线



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

程持有),那么线程 A 就会不断的查询 a 的值,直到 a=1。从这里我们可以看到自旋锁的一个缺点:那就等待自旋锁的线程会一直处于自旋状态,这样会浪费处理器时间,降低系统性能,所以自旋锁的持有时间不能太长。所以自旋锁适用于短时期的轻量级加锁,如果遇到需要长时间持有锁的场景那就需要换其他的方法了,这个我们后面会讲解。

Linux 内核使用结构体 spinlock_t 表示自旋锁,结构体定义如下所示:

```
示例代码 29.3.1 spinlock_t 结构体
```

64 typedef struct spinlock {
65 union {
66 struct raw_spinlock rlock;
67
68 #ifdef CONFIG_DEBUG_LOCK_ALLOC
69 # define LOCK_PADSIZE (offsetof(struct raw_spinlock, dep_map))
70 struct {
71 u8padding[LOCK_PADSIZE];
72 struct lockdep_map dep_map;
73 };
74 #endif
75 };
76 } spinlock_t;
在使用自旋锁之前,肯定要先定义一个自旋锁变量,定义方法如下所示:
spinlock t lock: //定义自旋锁

定义好自旋锁变量以后就可以使用相应的 API 函数来操作自旋锁。

29.3.2 自旋锁 API 函数

最基本的自旋锁 API 函数如下表所示:

函数	措述
DEFINE_SPINLOCK(spinloc	宁 \) 并初始化一个白进峦晶
k_t lock)	定入升的如凡 日起又重。
int spin_lock_init(spinlock_t	初始化白诺瑞
*lock)	初知礼日派领。
void spin_lock(spinlock_t	本面也完的白海绵,也则他加绵
*lock)	犹取1日足的日展顿,也叫做加钡。
void spin_unlock(spinlock_t	双 故
*lock)	件双1日足口日灰切。
int spin_trylock(spinlock_t	尝试获取指定的自旋锁,如果没有获取
*lock)	到就返回0
int spin_is_locked(spinlock_t	检查指定的自旋锁是否被获取,如果没
*lock)	有被获取就返回非0,否则返回0。
主 20.2.1.1	白花继甘木和国家教主

表 29.3.1 自旋锁基本 API 函数表

表 29.3.1 中的自旋锁 API 函数适用于 SMP 或支持抢占的单 CPU 下线程之间的并发访问, 也就是用于线程与线程之间,被自旋锁保护的临界区一定不能调用任何能够引起睡眠和阻塞 的 API 函数,否则的话会可能会导致死锁现象的发生。自旋锁会自动禁止抢占,也就说当线



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

程 A 得到锁以后会暂时禁止内核抢占。如果线程 A 在持有锁期间进入了休眠状态,那么线程 A 会自动放弃 CPU 使用权。线程 B 开始运行,线程 B 也想要获取锁,但是此时锁被 A 线程持有,而且内核抢占还被禁止了!线程 B 无法被调度出去,那么线程 A 就无法运行,锁也就无法释放,好了,死锁发生了!

表 29.3.1 中的 API 函数用于线程之间的并发访问,如果此时中断也要插一脚,中断也想 访问共享资源,那该怎么办呢? 首先可以肯定的是,中断里面可以使用自旋锁,但是在中断 里面使用自旋锁的时候,在获取锁之前一定要先禁止本地中断(也就是本 CPU 中断,对于多核 SOC 来说会有多个 CPU 核),否则可能导致锁死现象的发生,如下图所示:



图 29.3.1 中断打断线程

在图 29.3.1 中,线程 A 先运行,并且获取到了 lock 这个锁,当线程 A 运行 functionA 函数的时候中断发生了,中断抢走了 CPU 使用权。右边的中断服务函数也要获取 lock 这个锁,但是这个锁被线程 A 占有着,中断就会一直自旋,等待锁有效。但是在中断服务函数执行完之前,线程 A 是不可能执行的,线程 A 说"你先放手",中断说"你先放手",场面就这么僵持着,死锁发生!

最好的解决方法就是获取锁之前关闭本地中断,Linux内核提供了相应的API函数,如下表所示:

函数	描述
void spin_lock_irq(spinlock_t *lock)	禁止本地中断,并获取自旋锁。
void spin_unlock_irq(spinlock_t *lock)	激活本地中断,并释放自旋锁。
void spin_lock_irqsave(spinlock_t *lock, unsigned long flags)	保存中断状态,禁止本地中断,并获取 自旋锁。
void spin_unlock_irqrestore(spinlock_t *lock, unsigned long flags)	将中断状态恢复到以前的状态,并且激 活本地中断,释放自旋锁。

表 29.3.2 线程与中断并发访问处理 API 函数

使用 spin_lock_irq/spin_unlock_irq 的时候需要用户能够确定加锁之前的中断状态,但实际 上内核很庞大,运行也是"千变万化",我们是很难确定某个时刻的中断状态,因此不推荐 使用 spin_lock_irq/spin_unlock_irq。建议使用 spin_lock_irqsave/ spin_unlock_irqrestore,因为这 一组函数会保存中断状态,在释放锁的时候会恢复中断状态。一般在线程中使用 spin_lock_irqsave/ spin_unlock_irqrestore,在中断中使用 spin_lock/spin_unlock,示例代码如下 所示:

示例代码 29.3.2 自旋锁使用示例

原子哥在线教学: www.yuanzige.com	论坛:www.openedv.com/forum.php	
1 DEFINE_SPINLOCK(lock);	/* 定义并初始化一个自旋锁 */	
2		
3 /* 线程 A */		
4 void functionA (){		
5 unsigned long flags;	/* 中断状态 */	
<pre>6 spin_lock_irqsave(&lock, flags);</pre>	/* 获取锁 */	
7 /* 临界区 */		
<pre>8 spin_unlock_irqrestore(&lock, flags);</pre>	/* 释放锁 */	
9 }		
10		
11 /* 中断服务函数 */		
12 void irq() {		
<pre>13 spin_lock(&lock);</pre>	/* 获取锁 */	
14 /* 临界区 */		
<pre>15 spin_unlock(&lock);</pre>	/* 释放锁 */	
16 }		

🔁 正点原子

下半部(BH)也会竞争共享资源,有些资料也会将下半部叫做底半部。关于下半部后面的 章节会讲解,如果要在下半部里面使用自旋锁,可以使用表 29.3.3 中的 API 函数:

函数	描述
void spin_lock_bh(spinlock_t	土闭下半刻 光基取白旋锚
*lock)	大肉下干部,并获取日灰顿。
void spin_unlock_bh(spinlock_t *lock)	打开下半部,并释放自旋锁。

表 29.3.3 下半部竞争处理函数

29.3.3 其它类型的锁

在自旋锁的基础上还衍生出了其他特定场合使用的锁,这些锁在驱动中其实用的不多, 更多的是在 Linux 内核中使用,本节我们简单来了解一下这些衍生出来的锁。

1、读写自旋锁

现在有个学生信息表,此表存放着学生的年龄、家庭住址、班级等信息,此表可以随时 被修改和读取。此表肯定是数据,那么必须要对其进行保护,如果我们现在使用自旋锁对其 进行保护。每次只能一个读操作或者写操作,但是,实际上此表是可以并发读取的。只需要 保证在修改此表的时候没人读取,或者在其他人读取此表的时候没有人修改此表就行了。也 就是此表的读和写不能同时进行,但是可以多人并发的读取此表。像这样,当某个数据结构 符合读/写或生产者/消费者模型的时候就可以使用读写自旋锁。

读写自旋锁为读和写操作提供了不同的锁,一次只能允许一个写操作,也就是只能一个 线程持有写锁,而且不能进行读操作。但是当没有写操作的时候允许一个或多个线程持有读 锁,可以进行并发的读操作。Linux 内核使用 rwlock_t 结构体表示读写锁,结构体定义如下 (删除了条件编译):

示例代码 29.3.3 rwlock_t 结构体

typedef struct {



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

arch_rwlock_t raw_lock;

} rwlock_t;

读写锁操作 API 函数分为两部分,一个是给读使用的,一个是给写使用的,这些 API 函 数如表 29.3.4 所示:

函数	描述
DEFINE_RWLOCK(rwlock_t lock)	定义并初始化读写锁
<pre>void rwlock_init(rwlock_t *lock)</pre>	初始化读写锁。
	读锁
<pre>void read_lock(rwlock_t *lock)</pre>	获取读锁。
void read_unlock(rwlock_t *lock)	释放读锁。
<pre>void read_lock_irq(rwlock_t *lock)</pre>	禁止本地中断,并且获取读锁。
void read_unlock_irq(rwlock_t *lock)	打开本地中断,并且释放读锁。
void read_lock_irqsave(rwlock_t *lock, unsigned long	保存中断状态,禁止本地中断,并 获取读锁。
flags) void read_unlock_irqrestore(rwlock_t *lock, unsigned long flags)	将中断状态恢复到以前的状态,并 且激活本地中断,释放读锁。
void read_lock_bh(rwlock_t *lock)	关闭下半部,并获取读锁。
void read_unlock_bh(rwlock_t *lock)	打开下半部,并释放读锁。
	写创
<pre>void write_lock(rwlock_t *lock)</pre>	获取读锁。
<pre>void write_unlock(rwlock_t *lock)</pre>	释放读锁。
<pre>void write_lock_irq(rwlock_t *lock)</pre>	禁止本地中断,并且获取读锁。
void write_unlock_irq(rwlock_t *lock)	打开本地中断,并且释放读锁。
void write_lock_irqsave(rwlock_t *lock, unsigned long flags)	保存中断状态,禁止本地中断,并 获取读锁。
void write_unlock_irqrestore(rwlock_t *lock, unsigned long flags)	将中断状态恢复到以前的状态,并 且激活本地中断,释放读锁。
void write_lock_bh(rwlock_t *lock)	关闭下半部,并获取读锁。
void write_unlock_bh(rwlock_t *lock)	打开下半部,并释放读锁。

表 29.3.4 读写锁 API 函数

2、顺序锁

顺序锁在读写锁的基础上衍生而来的,使用读写锁的时候读操作和写操作不能同时进行。 使用顺序锁的话可以允许在写的时候进行读操作,也就是实现同时读写,但是不允许同时进



原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php

行并发的写操作。虽然顺序锁的读和写操作可以同时进行,但是如果在读的过程中发生了写操作,最好重新进行读取,保证数据完整性。顺序锁保护的资源不能是指针,因为如果在写操作的时候可能会导致指针无效,而这个时候恰巧有读操作访问指针的话就可能导致意外发生,比如读取野指针导致系统崩溃。Linux 内核使用 seqlock_t 结构体表示顺序锁,结构体定义如下:

示例代码 29.3.4 seqlock_t 结构体

typedef struct {

struct seqcount seqcount;

spinlock_t lock;

} seqlock_t;

关于顺序锁的 API 函数如表 29.3.5 所示:

函数	描述
<pre>DEFINE_SEQLOCK(seqlock_t sl)</pre>	定义并初始化顺序锁
<pre>void seqlock_ini seqlock_t *sl)</pre>	初始化顺序锁。
顺序	<i>演写操作</i>
<pre>void write_seqlock(seqlock_t *sl)</pre>	获取写顺序锁。
<pre>void write_sequnlock(seqlock_t *sl)</pre>	释放写顺序锁。
void write_seqlock_irq(seqlock_t *sl)	禁止本地中断,并且获取写顺序锁
void write_sequnlock_irq(seqlock_t *sl)	打开本地中断,并且释放写顺序 锁。
void write_seqlock_irqsave(seqlock_t *sl, unsigned long flags)	保存中断状态,禁止本地中断,并 获取写顺序锁。
<pre>void write_sequnlock_irqrestore(seqlock_t *sl, unsigned long flags)</pre>	将中断状态恢复到以前的状态,并 且激活本地中断,释放写顺序锁。
void write_seqlock_bh(seqlock_t *sl)	关闭下半部,并获取写读锁。
void write_sequnlock_bh(seqlock_t *sl)	打开下半部,并释放写读锁。
顺序	(演读操作) · · · · · · · · · · · · · · · · · · ·
unsigned read_seqbegin(const seqlock_t *sl)	读单元访问共享资源的时候调用此 函数,此函数会返回顺序锁的顺序号。
unsigned read_seqretry(const seqlock_t *sl, unsigned start)	读结束以后调用此函数检查在读的 过程中有没有对资源进行写操作,如果 有的话就要重读

表 29.3.5 顺序锁 API 函数表

29.3.4 自旋锁使用注意事项

综合前面关于自旋锁的信息,我们需要在使用自旋锁的时候要注意一下几点:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

①、因为在等待自旋锁的时候处于"自旋"状态,因此锁的持有时间不能太长,一定要短,否则的话会降低系统性能。如果临界区比较大,运行时间比较长的话要选择其他的并发处理方式,比如稍后要讲的信号量和互斥体。

②、自旋锁保护的临界区内不能调用任何可能导致线程休眠的 API 函数,否则的话可能导致死锁。

③、不能递归申请自旋锁,因为一旦通过递归的方式申请一个你正在持有的锁,那么你就必须"自旋",等待锁被释放,然而你正处于"自旋"状态,根本没法释放锁。结果就是自己把自己锁死了!

④、在编写驱动程序的时候我们必须考虑到驱动的可移植性,因此不管你用的是单核的还是多核的 SOC,都将其当做多核 SOC 来编写驱动程序。

29.4 信号量

29.4.1 信号量简介

大家如果有学习过 FreeRTOS 或者 UCOS 的话就应该对信号量很熟悉,因为信号量是同步的一种方式。Linux 内核也提供了信号量机制,信号量常常用于控制对共享资源的访问。举一个很常见的例子,某个停车场有 100 个停车位,这 100 个停车位大家都可以用,对于大家来说这 100 个停车位就是共享资源。假设现在这个停车场正常运行,你要把车停到这个这个停车场肯定要先看一下现在停了多少车了?还有没有停车位?当前停车数量就是一个信号量,具体的停车数量就是这个信号量值,当这个值到 100 的时候说明停车场满了。停车场满的时你可以等一会看看有没有其他的车开出停车场,当有车开出停车场的时候停车数量就会减一,也就是说信号量减一,此时你就可以把车停进去了,你把车停进去以后停车数量就会加一,也就是信号量加一。这就是一个典型的使用信号量进行共享资源管理的案例,在这个案例中使用的就是计数型信号量。

相比于自旋锁,信号量可以使线程进入休眠状态,比如 A 与 B、C 合租了一套房子,这 个房子只有一个厕所,一次只能一个人使用。某一天早上 A 去上厕所了,过了一会 B 也想用 厕所,因为 A 在厕所里面,所以 B 只能等到 A 用来了才能进去。B 要么就一直在厕所门口等 着,等 A 出来,这个时候就相当于自旋锁。B 也可以告诉 A,让 A 出来以后通知他一下,然 后 B 继续回房间睡觉,这个时候相当于信号量。可以看出,使用信号量会提高处理器的使用 效率,毕竟不用一直傻乎乎的在那里"自旋"等待。但是,信号量的开销要比自旋锁大,因 为信号量使线程进入休眠状态以后会切换线程,切换线程就会有开销。总结一下信号量的特 点:

①、因为信号量可以使等待资源线程进入休眠状态,因此适用于那些占用资源比较久的场合。

②、因此信号量不能用于中断中,因为信号量会引起休眠,中断不能休眠。

③、如果共享资源的持有时间比较短,那就不适合使用信号量了,因为频繁的休眠、切 换线程引起的开销要远大于信号量带来的那点优势。

信号量有一个信号量值,相当于一个房子有 10 把钥匙,这 10 把钥匙就相当于信号量值 为 10。因此,可以通过信号量来控制访问共享资源的访问数量,如果要想进房间,那就要先 获取一把钥匙,信号量值减 1,直到 10 把钥匙都被拿走,信号量值为 0,这个时候就不允许 任何人进入房间了,因为没钥匙了。如果有人从房间出来,那他要归还他所持有的那把钥匙,



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php 信号量值加1,此时有1把钥匙了,那么可以允许进去一个人。相当于通过信号量控制访问资 源的线程数,在初始化的时候将信号量值设置的大于1,那么这个信号量就是计数型信号量, 计数型信号量不能用于互斥访问,因为它允许多个线程同时访问共享资源。如果要互斥的访 问共享资源那么信号量的值就不能大于1,此时的信号量就是一个二值信号量。

29.4.2 信号量 API 函数

Linux 内核使用 semaphore 结构体表示信号量,结构体内容如下所示: 示例代码 29.4.1 semaphore 结构体

struct semaphore {
 raw_spinlock_t lock;

unsigned int count;

struct list head wait list;

};

要想使用信号量就得先定义,然后初始化信号量。有关信号量的 API 函数如表 29.4.1 所

示:

函数	描述
	定义一个信号量,并且设置信号量
DEFINE_SEAMPHORE(liaille)	的值为1。
void sema_init(struct semaphore	初始化信号量 sem,设置信号量值
*sem, int val)	为 val。
void down(struct somenhore *som)	获取信号量,因为会导致休眠,因
void down(struct semaphore *sem)	此不能在中断中使用。
	尝试获取信号量,如果能获取到信
int down_trylock(struct semaphore	号量就获取,并且返回 0。如果不能就
sem),	返回非0,并且不会进入休眠。
	获取信号量,和 down 类似,只是
int down_interruptible(struct	使用 down 进入休眠状态的线程不能被
semaphore *sem)	信号打断。而使用此函数进入休眠以后
	是可以被信号打断的。
void up(struct semaphore *sem)	释放信号量
<pre>*sem); int down_interruptible(struct semaphore *sem) void up(struct semaphore *sem)</pre>	专重 就 获取, 开 且 返回 0。 如果 个 能 就 返 回 非 0, 并 且 不 会 进 入 休 眠 。 获 取 信 号 量, 和 down 类 似, 只 是 使 用 down 进 入 休 眠 状 态 的 线 程 不 能 被 信 号 打 断 。 而 使 用 此 函 数 进 入 休 眠 以 后 是 可 以 被 信 号 打 断 的。 释 放 信 号 量

表 29.4.1 信号量 API 函数

信号量的使用如下所示:

```
示例代码 29.4.2 信号量使用示例
struct semaphore sem; /* 定义信号量 */
sema_init(&sem, 1); /* 初始化信号量 */
down(&sem); /* 申请信号量 */
/* 临界区 */
up(&sem); /* 释放信号量 */
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

正点原子

29.5 互斥体

29.5.1 互斥体简介

在 FreeRTOS 和 UCOS 中也有互斥体,互斥体也可以叫互斥锁。将信号量的值设置为1就 可以使用信号量进行互斥访问了,虽然可以通过信号量实现互斥,但是 Linux 提供了一个比 信号量更专业的机制来进行互斥,它就是互斥体一mutex。互斥访问表示一次只有一个线程可 以访问共享资源,不能递归申请互斥体。在我们编写 Linux 驱动的时候遇到需要互斥访问的 地方建议使用 mutex。Linux 内核使用 mutex 结构体表示互斥体, 定义如下(省略条件编译部 分):

示例代码 29.5.1 mutex 结构体

struct mutex {

/* 1: unlocked, 0: locked, negative: locked, possible waiters */

atomic t count;

spinlock t wait_lock;

};

在使用 mutex 之前要先定义一个 mutex 变量。在使用 mutex 的时候要注意如下几点: ①、mutex 可以导致休眠,因此不能在中断中使用 mutex,中断中只能使用自旋锁(因为 中断不参与进程调度,如果一旦在中断服务函数执行过程中休眠了,休眠了则意味着交 出了 CPU 的使用权, CPU 使用权则跑到了其它线程了, 那么就不能再回到中断断点处 了)。

③ 、和信号量一样, mutex 保护的临界区可以调用引起阻塞的 API 函数。

④ 、因为一次只有一个线程可以持有 mutex,因此,必须由 mutex 的持有者释放 mutex。 并且 mutex 不能递归上锁和解锁。

29.5.2 互斥体 API 函数

有关互斥体的 API 函数如表 29.5.1 所示:

函数	描述
DEFINE_MUTEX(name)	定义并初始化一个 mutex 变量。
<pre>void mutex_init(mutex *lock)</pre>	初始化 mutex。
<pre>void mutex_lock(struct mutex *lock)</pre>	获取 mutex,也就是给 mutex 上锁。如果获取不到就进休眠。
void mutex_unlock(struct mutex *lock)	释放 mutex,也就给 mutex 解锁。
int mutex_trylock(struct mutex *lock)	尝试获取 mutex,如果成功就返回 1,如果失败就返回 0。
int mutex_is_locked(struct mutex *lock)	判断 mutex 是否被获取,如果是的话就返回1,否则返回0。
int mutex_lock_interruptible(struct mutex *lock)	使用此函数获取信号量失败进入 休眠以后可以被信号打断。

表 29.5.1 互斥体 API 函数

互斥体的使用如下所示:



原子	² 哥在线教学:www.yu	anzige.com 论	坛:www.openedv.com/forum.ph	ւթ
		示例代码 29.5	5.2 互斥体使用示例	
	1 struct mutex lock;	/* 定义一个互斥体 */	/	
	2 mutex_init(&lock);	/* 初始化互斥体 */		
	3			
	4 mutex_lock(&lock);	/* 上锁 */		
	5 /* 临界区 */			
	6 mutex_unlock(&lock);	/* 解锁 */		
	关手 Linux 由的并生手	山主名前计级到达田	Linux 由核还方组夕甘仙的丛	、珊光生和喜名放

关于 Linux 中的并发和竞争就讲解到这里, Linux 内核还有很多其他的处理并发和竞争的 机制,本章我们主要讲解了常用的原子操作、自旋锁、信号量和互斥体。以后我们在编写 Linux 驱动的时候就会频繁的使用到这几种机制,希望大家能够深入理解这几个常用的机制。 原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

正点原子

第三十章 Linux 并发与竞争实验

在上一章中我们学习了 Linux 下的并发与竞争,并且学习了四种常用的处理并发和竞争 的机制:原子操作、自旋锁、信号量和互斥体。本章我们就通过四个实验来学习如何在驱动 中使用这四种机制。

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

正点原子

30.1 原子操作实验

本实验对应的例程路径为:开发板资料盘(A 盘)\4 SourceCode\3 Embedded Linux\Linux 驱动例程\7 atomic。

本例程我们在第二十七章的 gpioled.c 文件基础上完成。在本节中我们使用原子操作来实 现对 LED 这个设备的互斥访问,也就是一次只允许一个应用程序可以使用 LED 灯。

30.1.1 实验程序编写

1、修改设备树文件

因为本章实验是在第二十七章实验的基础上完成的,因此不需要对设备树做任何的修改。

2、LED 驱动修改

本节实验在第二十七章实验驱动文件 gpioled.c 的基础上修改而来。首先在我们的 drivers 目录下新建名为"7_atomic"的文件夹,将第二十七章实验目录下的gpioled.c复制到7_atomic 文件夹中,并且重命名为 atomic.c。

本节实验重点就是使用 atomic 来实现一次只能允许一个应用访问 LED, 所以我们只需要 在 atomic.c 文件源码的基础上加上添加 atomic 相关代码即可,完成以后的 atomic.c 文件内容 如下所示:

示例代码 30.1.1 atomic.c 文件内容 2 Copyright © ALIENTEK Co., Ltd. 1998-2029. All rights reserved. 3 文件名 : atomic.c 4 作者 : 邓涛 5版本 : V1.0 6 描述 :原子操作实验,使用原子变量来实现对实现设备的互斥访问 7 其他 :无 8 论坛 : www.openedv.com 9 日志 :初版 V1.0 2019/1/30 邓涛创建 11 12 #include <linux/types.h> 13 #include <linux/kernel.h> 14 #include <linux/delay.h> 15 #include <linux/ide.h> 16 #include <linux/init.h> 17 #include <linux/module.h> 18 #include <linux/errno.h> 19 #include linux/gpio.h> 20 #include <asm/mach/map.h> 21 #include <asm/uaccess.h> 22 #include <asm/io.h> 23 #include <linux/cdev.h>



```
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
    24 #include <linux/of.h>
    25 #include <linux/of address.h>
    26 #include <linux/of_gpio.h>
    27
    28 #define GPIOLED CNT 1 /* 设备号个数 */
                              "gpioled" /* 名字 */
    29 #define GPIOLED NAME
    30
    31 /* dtsled 设备结构体 */
    32 struct gpioled_dev {
                              /* 设备号 */
    33 dev t devid;
                              /* cdev */
    34 struct cdev cdev;
                             /* 类 */
    35 struct class *class;
    36 struct device *device;
                            /* 设备 */
    37 int major;
                             /* 主设备号 */
                             /* 次设备号 */
    38 int minor;
    39 struct device_node *nd; /* 设备节点 */
                             /* LED 所使用的 GPIO 编号 */
   40 int led_gpio;
    41
        atomic_t lock;
                             /* 原子变量 */
   42 };
    43
                                 /* led 设备 */
   44 static struct gpioled_dev gpioled;
   45
   46 /*
   47 * @description
                              :打开设备
   48 * @param – inode
                              :传递给驱动的 inode
                              :设备文件, file 结构体有个叫做 private_data 的成员变量
   49 * @param – filp
    50 *
                              一般在 open 的时候将 private_data 指向设备结构体。
                              :0 成功;其他 失败
   51 * @return
    52 */
    53 static int led_open(struct inode *inode, struct file *filp)
    54 {
       /* 通过判断原子变量的值来检查 LED 有没有被别的应用使用 */
    55
    56
        if (!atomic_dec_and_test(&gpioled.lock)) {
          printk(KERN_ERR "gpioled: Device is busy!\n");
    57
    58
          atomic_inc(&gpioled.lock); /* 小于 0 的话就加 1,使其原子变量等于 0 */
    59
          return -EBUSY;
                                         /* LED 被其他应用使用,返回忙 */
    60 }
    61
    62
        return 0;
    63 }
    64
```

```
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
    65 /*
    66 * @description
                              :从设备读取数据
    67 * @param – filp
                              :要打开的设备文件(文件描述符)
    68 * @param – buf
                              :返回给用户空间的数据缓冲区
    69 * @param – cnt
                             :要读取的数据长度
    70 * @param – offt
                             :相对于文件首地址的偏移
                              :读取的字节数,如果为负值,表示读取失败
    71 * @return
    72 */
    73 static ssize_t led_read(struct file *filp, char __user *buf,
            size t cnt, loff t *offt)
    74
    75 {
    76 return 0;
    77 }
    78
    79 /*
                             : 向设备写数据
    80 * @description
                             :设备文件,表示打开的文件描述符
    81 * @param – filp
    82 * @param – buf
                             :要写给设备写入的数据
    83 * @param – cnt
                             :要写入的数据长度
    84 * @param – offt
                              :相对于文件首地址的偏移
                              :写入的字节数,如果为负值,表示写入失败
    85 * @return
    86 */
    87 static ssize_t led_write(struct file *filp, const char __user *buf,
    88
            size_t cnt, loff_t *offt)
    89 {
    90
       int ret;
    91
        char kern_buf[1];
    92
        ret = copy_from_user(kern_buf, buf, cnt); // 得到应用层传递过来的数据
    93
    94
        if(0 > ret) {
    95
          printk(KERN_ERR "kernel write failed!\r\n");
          return -EFAULT;
    96
    97
       }
    98
        if (0 == kern_buf[0])
    99
   100
          gpio_set_value(gpioled.led_gpio, 0); // 如果传递过来的数据是 0 则关闭 led
   101
        else if (1 == kern_buf[0])
          gpio_set_value(gpioled.led_gpio, 1); // 如果传递过来的数据是 1 则点亮 led
   102
   103
   104
        return 0;
   105 }
```

正点原子

106



```
107 /*
108 * @description
                             :关闭/释放设备
109 * @param – filp
                              :要关闭的设备文件(文件描述符)
110 * @return
                             :0 成功;其他 失败
111 */
112 static int led_release(struct inode *inode, struct file *filp)
113 {
114 /* 关闭驱动文件的时候释放原子变量 */
115
     atomic_inc(&gpioled.lock);
116
117 return 0;
118 }
119
120 /* 设备操作函数 */
121 static struct file_operations gpioled_fops = {
               = THIS_MODULE,
122 .owner
123 .open
                 = led_open,
124 .read
                = led_read,
125 .write
                 = led_write,
126 .release = led_release,
127 };
128
129 static int __init led_init(void)
130 {
131
     const char *str;
132
     int ret;
133
134
     /* 1.获取 led 设备节点 */
135
     gpioled.nd = of_find_node_by_path("/led");
     if(NULL == gpioled.nd) {
136
        printk(KERN_ERR "gpioled: Failed to get /led node\n");
137
138
        return -EINVAL;
139
     }
140
141 /* 2.读取 status 属性 */
142
     ret = of_property_read_string(gpioled.nd, "status", &str);
143
     if(!ret) {
144
        if (strcmp(str, "okay"))
145
          return -EINVAL;
146 }
```



```
原子哥在线教学: www.yuanzige.com
                                                论坛:www.openedv.com/forum.php
    147
    148
          /* 2、获取 compatible 属性值并进行匹配 */
    149
          ret = of_property_read_string(gpioled.nd, "compatible", &str);
    150
          if(0 > ret) {
            printk(KERN_ERR "gpioled: Failed to get compatible property\n");
    151
    152
            return ret;
    153
          }
    154
    155
          if (strcmp(str, "alientek,led")) {
    156
            printk(KERN_ERR "gpioled: Compatible match failed\n");
    157
            return -EINVAL;
          }
    158
    159
    160
          printk(KERN_INFO "gpioled: device matching successful!\r\n");
    161
          /* 4.获取设备树中的 led-gpio 属性,得到 LED 所使用的 GPIO 编号 */
    162
    163
          gpioled.led_gpio = of_get_named_gpio(gpioled.nd, "led-gpio", 0);
          if(!gpio_is_valid(gpioled.led_gpio)) {
    164
    165
            printk(KERN_ERR "gpioled: Failed to get led-gpio\n");
    166
            return -EINVAL;
          }
    167
    168
          printk(KERN_INFO "gpioled: led-gpio num = %d\r\n", gpioled.led_gpio);
    169
    170
    171
          /* 5.向 gpio 子系统申请使用 GPIO */
    172
          ret = gpio_request(gpioled.led_gpio, "LED-GPIO");
    173
          if (ret) {
    174
            printk(KERN_ERR "gpioled: Failed to request led-gpio\n");
    175
            return ret;
    176
          }
    177
          /* 6.将 led gpio 管脚设置为输出模式 */
    178
    179
          gpio_direction_output(gpioled.led_gpio, 0);
    180
    181
          /* 7.初始化 LED 的默认状态 */
    182
          ret = of_property_read_string(gpioled.nd, "default-state", &str);
    183
          if(!ret) {
    184
            if (!strcmp(str, "on"))
    185
               gpio_set_value(gpioled.led_gpio, 1);
    186
            else
    187
              gpio_set_value(gpioled.led_gpio, 0);
```



```
原子哥在线教学: www.yuanzige.com
                                              论坛:www.openedv.com/forum.php
    188
         } else
    189
            gpio_set_value(gpioled.led_gpio, 0);
    190
    191
         /* 8.注册字符设备驱动 */
    192
          /* 创建设备号 */
    193
         if (gpioled.major) {
    194
            gpioled.devid = MKDEV(gpioled.major, 0);
    195
            ret = register_chrdev_region(gpioled.devid, GPIOLED_CNT, GPIOLED_NAME);
    196
            if (ret)
    197
              goto out1;
    198
          } else {
            ret = alloc_chrdev_region(&gpioled.devid, 0, GPIOLED_CNT, GPIOLED_NAME);
    199
    200
            if (ret)
    201
              goto out1;
    202
    203
            gpioled.major = MAJOR(gpioled.devid);
    204
            gpioled.minor = MINOR(gpioled.devid);
    205
          }
    206
    207
          printk("gpioled: major=%d,minor=%d\r\n",gpioled.major, gpioled.minor);
    208
    209
          /* 初始化 cdev */
          gpioled.cdev.owner = THIS_MODULE;
    210
    211
          cdev_init(&gpioled.cdev, &gpioled_fops);
    212
    213
         /* 添加一个 cdev */
    214
         ret = cdev_add(&gpioled.cdev, gpioled.devid, GPIOLED_CNT);
    215
          if (ret)
    216
            goto out2;
    217
    218
          /* 创建类 */
    219
         gpioled.class = class_create(THIS_MODULE, GPIOLED_NAME);
    220
         if (IS_ERR(gpioled.class)) {
    221
            ret = PTR_ERR(gpioled.class);
    222
            goto out3;
    223
         }
    224
    225
         /* 创建设备 */
    226
          gpioled.device = device_create(gpioled.class, NULL,
    227
                gpioled.devid, NULL, GPIOLED_NAME);
    228
          if (IS_ERR(gpioled.device)) {
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php 229 ret = PTR_ERR(gpioled.device); 230 goto out4; 231 } 232 233 /* 9.初始化原子变量 */ 234 atomic_set(&gpioled.lock, 1); /* 原子变量初始值为 1 */ 235 236 return 0; 237 238 out4: 239 class_destroy(gpioled.class); 240 241 out3: 242 cdev_del(&gpioled.cdev); 243 244 out2: 245 unregister_chrdev_region(gpioled.devid, GPIOLED_CNT); 246 247 out1: 248 gpio_free(gpioled.led_gpio); 249 250 return ret; 251 } 252 253 static void __exit led_exit(void) 254 { 255 /* 注销设备 */ 256 device_destroy(gpioled.class, gpioled.devid); 257 258 /* 注销类 */ 259 class_destroy(gpioled.class); 260 261 /* 删除 cdev */ 262 cdev_del(&gpioled.cdev); 263 264 /* 注销设备号 */ 265 unregister_chrdev_region(gpioled.devid, GPIOLED_CNT); 266 /* 释放 GPIO */ 267 gpio_free(gpioled.led_gpio); 268 269 }



原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php

270

271 /* 驱动模块入口和出口函数注册 */

272 module_init(led_init);

273 module_exit(led_exit);

274

275 MODULE AUTHOR("DengTao <773904075@qq.com>");

276 MODULE_DESCRIPTION("Alientek ZYNQ GPIO LED Driver");

277 MODULE_LICENSE("GPL");

第 41 行,在 struct gpioled_dev 结构体中定义了一个原子变量 lock,用来实现一次只能允 许一个应用访问 LED 灯, led_init 驱动入口函数会将 lock 的值设置为 1。

第 53~63 行,每次调用 open 函数打开驱动设备的时候先申请 lock,如果申请成功的话就 表示 LED 灯还没有被其他的应用使用,如果申请失败就表示 LED 灯正在被其他的应用程序使 用,并打印 "gpioled: Device is busy!"。每次打开驱动设备的时候先使用 atomic_dec_and_test 函数将 lock 减 1, 如果 atomic_dec_and_test 函数返回值为真就表示 lock 当前值为 0, 说明设备 可以使用。如果 atomic_dec_and_test 函数返回值为假,就表示 lock 当前值为负数(lock 值默认 是 1), lock 值为负数的可能性只有一个,那就是其他设备正在使用 LED。其他设备正在使用 LED 灯,那么就只能退出了,在退出之前调用函数 atomic_inc 将 lock 加 1,因为此时 lock 的 值被减成了负数,必须要对其加1,将 lock 的值变为0。

第 115 行, LED 灯使用完毕,应用程序调用 close 函数关闭的驱动文件, led_release 函数 执行,调用 atomic_inc 释放 lcok,也就是将 lock 加 1。

第234行,初始化原子变量lock,初始值设置为1,这样每次就只允许一个应用使用LED 灯。

3、编写测试 APP

在 7_atomic 实验目录下新建名为 atomicApp.c 的测试 APP, 在里面输入如下所示内容: 示例代码 30.1.2 atomicApp.c 文件内容

2 Copyright © ALIENTEK Co., Ltd. 1998-2029. All rights reserved. 3 文件名 : atomicApp.c 4 作者 :邓涛 5 版本 : V1.0

- 6 描述 :原子变量测试 APP,测试原子变量能不能实现一次
- 7 只允许一个应用程序使用 LED。 8 其他 :无
- 9 使用方法 : ./atomicApp /dev/gpioled 0 关闭 LED 灯
- ./atomicApp /dev/gpioled 1 打开 LED 灯 10
- 11 论坛 : www.openedv.com
- 12 日志 :初版 V1.0 2019/1/30 邓涛创建

14

15 #include <stdio.h>

16 #include <unistd.h>



```
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
    17 #include <sys/types.h>
    18 #include <sys/stat.h>
    19 #include <fcntl.h>
    20 #include <stdlib.h>
    21 #include <string.h>
    22
    23 /*
    24 * @description
                                  :main 主程序
    25 * @param – argc
                                  : argv 数组元素个数
    26 * @param – argv
                                  :具体参数
    27 * @return
                                  :0 成功;其他 失败
    28 */
    29 int main(int argc, char *argv[])
    30 {
         int fd, ret;
    31
    32
         int cnt = 0;
    33
         unsigned char buf[1];
    34
    35
         if(3 != argc) {
    36
           printf("Usage:\n"
    37
               "\t./atomicApp /dev/gpioled 1
                                               @ close led\n"
    38
               "\t./atomicApp /dev/gpioled 0
                                               @ open led\n"
    39
               );
    40
           return -1;
    41
         }
    42
        /* 打开设备 */
    43
         fd = open(argv[1], O_RDWR);
    44
    45
         if(0 > fd) \{
    46
           printf("ERROR: file %s open failed!\r\n", argv[1]);
    47
           return -1;
         }
    48
    49
         /* 将字符串转换为 int 型数据 */
    50
    51
         buf[0] = atoi(argv[2]);
    52
    53 /* 向驱动写入数据 */
    54
         ret = write(fd, buf, sizeof(buf));
    55
         if(0 > ret)
    56
           printf("ERROR: LED Control Failed!\r\n");
    57
           close(fd);
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

58	return -1;
59	}
60	
61	/* 模拟占用 25 秒 LED 设备 */
62	for (;;) {
63	sleep(5);
64	cnt++;
65	<pre>printf("App running times:%d\r\n", cnt);</pre>
66	<pre>if(cnt >= 5) break;</pre>
67	}
68	
69	<pre>printf("App running finished!\n");</pre>
70	
71	/* 关闭设备 */
72	close(fd);
73	return 0;
74 }	

atomicApp.c 中的内容就是在第二十七章的 ledAPP.c 的基础上修改而来的,重点是加入了 第 62~67 行的模拟占用 25 秒 LED 设备的代码。测试 APP 在获取到 LED 设备使用权以后会使 用 25 秒,在使用的这段时间如果有其他的应用也去获取 LED 灯使用权的话肯定会失败!

30.1.2 运行测试

1、编译驱动程序

编写 Makefile 文件,将第二十七章实验目录下的 Makefile 文件拷贝到 7_atomic 目录下, 修改 Makefile 文件的 obj-m 变量,修改完成之后 Makefile 内容如下所示:

示例代码 30.1.3 Makefile 文件内容

1 KERN_DIR :=/home/zy/workspace/kernel-driver/linux-xlnx-xlnx_rebase_v5.4_2020.2
2
3 obj-m :=atomic.o
4
5 all:
6 make -C \$(KERN_DIR) M=`pwd` modules
7 clean:
8 make -C \$(KERN_DIR) M=`pwd` clean
第 3 行,设置 obj-m 变量的值为 atomic.,其它的都没改。
修改完成之后保存退出,输入如下命令编译出驱动模块文件:
make
编译成功以后就会生成一个名为 "atomic.ko" 的驱动模块文件,如下所示:



原子哥在线教学: www.yuanzige.com 论坛:ww

论坛:www.openedv.com/forum.php

zy@zy-virtual-machine:/mnt/hgfs/share18/linux驱动例程/7 atomic\$ ls atomicApp.c atomic.c Makefile zy@zy-virtual-machine:/mnt/hgfs/share18/linux驱动例程/7 atomic\$ make make -C /home/zy/workspace/kernel-driver/linux-xlnx-xlnx rebase v5.4 2020.2 M=`p wd` modules make[1]: 进入目录"/home/zy/workspace/kernel-driver/linux-xlnx-xlnx rebase v5.4 2 020.2" /mnt/hgfs/share18/linux驱动例程/7 atomic/atomic.o CC [M] Building modules, stage 2. MODPOST 1 modules CC [M] /mnt/hgfs/share18/linux驱动例程/7 atomic/atomic.mod.o LD [M] /mnt/hgfs/share18/linux驱动例程/7 atomic/atomic.ko make[1]: 离开目录"/home/zy/workspace/kernel-driver/linux-xlnx-xlnx rebase v5.4 2 020.2" zy@zy-virtual-machine:/mnt/hgfs/share18/linux驱动例程/7 atomic\$ ls atomicApp.c atomic.ko atomic.mod.c atomic.o modules.order atomic.mod atomic.mod.o Makefile Module.symvers atomic.c

图 30.1.1 编译 atomic 驱动模块

2、编译测试 APP

输入如下命令编译测试 atomicApp.c 这个测试程序:

\$CC atomicApp.c -o atomicApp

编译成功以后就会生成 atomicApp 这个应用程序。

3、运行测试

将上面编译出来的 atomic.ko 和 atomicApp 这两个文件拷贝到 NFS 共享目录下根文件系统的/lib/modules/5.4.0-xilinx 文件夹中。

将 SD 启动卡插到开发板中,连接开发板串口、网口和电源线,设置拨码开关为 SD 卡启 动模式,然后打开电源,启动开发板。

系统启动完成后,通过串口终端进入"/lib/modules/5.4.0-xilinx"目录下,输入如下命令 加载 atomic.ko 驱动模块:

depmod //第一次加载驱动的时候需要运行此命令

modprobe atomic.ko //加载驱动

驱动加载成功以后就可以使用 atomicApp 软件来测试驱动是否工作正常,输入如下命令 以后台运行模式运行 atomicApp 程序, "&"表示将程序放置后台运行,不占用终端:

./atomicApp /dev/gpioled 0 & //熄灭 LED 灯

输入上述命令以后查看开发板上的 PS_LED0 灯是否熄灭(驱动成功加载之后 LED 会被 点亮),然后每隔 5 秒都会输出一行 "App running times",如图 30.1.2 所示:

```
root@ALIENTÉK-ZYNQ-driver:/lib/modules/5.4.0-xilinx#
root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx# ./atomicApp /dev/gpioled 0 &
[1] 521
root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx# App running times:1
App running times:2
App running times:3
App running times:4
```

图 30.1.2 熄灭 LED 灯



原子哥在线教学: www.yuanzige.com 论坛:ww

论坛:www.openedv.com/forum.php

从图 30.1.2 中可以看出, atomicApp 运行正常, 输出了"App running times:1"和"App running times:2",这就是模拟 25 秒占用 LED 设备,说明 atomicApp 这个软件正在使用 LED 灯。此时再输入如下命令关闭 LED 灯:

./atomicApp /dev/gpioled 1 //点亮 LED 灯

输入上述命令以后会发现如图 30.1.3 所示输入信息:

root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx# ./atomicApp /dev/gpioled 0 & [1] 526 root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx# App running times:1 ./atomicApp /dev/gpioled 1App running times:2 gpioled: Device is busy! ERROR: file /dev/gpioled open failed! root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx#

图 30.1.3 点亮 LED 灯

从图 30.1.3 可以看出,打开/dev/gpioled 失败! 原因是在图 30.1.2 中运行的 atomicApp 软件正在占用/dev/gpioled,也就是它正在占用 LED 设备,如果再次运行 atomicApp 软件去操作 /dev/gpioled 肯定会失败。必须等待图 30.1.2 中的 atomicApp 运行结束,也就是 25S 结束以后 其他软件才能去操作/dev/gpioled。这个就是采用原子变量实现一次只能有一个应用程序访问 LED 灯。

如果要卸载驱动的话输入如下命令即可:

rmmod atomic.ko

30.2 自旋锁实验

上一节我们使用原子变量实现了一次只能有一个应用程序访问 LED 灯,本节我们使用自旋锁来实现此功能。在使用自旋锁之前,先回顾一下自旋锁的使用注意事项:

①、自旋锁保护的临界区要尽可能的短,因此在 open 函数中申请自旋锁,然后在 release 函数中释放自旋锁的方法就不可取。我们可以使用一个变量来表示设备的使用情况,如果设备被使用了那么变量就加一,设备被释放以后变量就减 1,我们只需要使用自旋锁保护这个变量即可。

②、考虑驱动的兼容性,合理的选择 API 函数。

综上所述,在本节例程中,我们通过定义一个变量 dev_stats 表示设备的使用情况, dev_stats为0的时候表示设备没有被使用,dev_stats大于0的时候表示设备被使用。驱动 open 函数中先判断 dev_stats 是否为0,也就是判断设备是否可用,如果为0的话就使用设备,并 且将 dev_stats 加1,表示设备被使用了。使用完以后在 release 函数中将 dev_stats 减1,表示 设备没有被使用了。因此真正实现设备互斥访问的是变量 dev_stats,但是我们要使用自旋锁 对 dev_stats 来做保护。

本实验对应的例程路径为:开发板资料盘(A 盘)\4_SourceCode\3_Embedded_Linux\Linux 驱动例程\8_spinlock。

30.2.1 实验程序编写

1、修改设备树文件

本章实验是在上一节实验的基础上完成的,同样不需要对设备树做任何的修改。

2、LED 驱动修改



原子哥在线教学: www.yuanzige.com 论

.....

论坛:www.openedv.com/forum.php

本节实验在第上一节实验驱动文件 atomic.c 的基础上修改而来。在 drivers 目录下新建名为 "8_spinlock"的文件夹,将 7_atomic 实验中的 atomic.c 复制到 8_spinlock 文件夹中,并且 重命名为 spinlock.c。将原来使用 atomic 的地方换为 spinlock 即可,其他代码不需要修改,完 成以后的 spinlock.c 文件内容如下所示(有省略):

示例代码 30.2.1 spinlock.c 文件内容(片段)

```
28 #define GPIOLED_CNT
                                          /* 设备号个数 */
                          1
29 #define GPIOLED_NAME
                                 "gpioled" /* 名字 */
30
31 /* dtsled 设备结构体 */
32 struct gpioled_dev {
                             /* 设备号 */
33 dev_t devid;
34 struct cdev cdev;
                             /* cdev */
                             /* 类 */
35 struct class *class;
                             /* 设备 */
36 struct device *device;
                             /* 主设备号 */
37 int major;
38 int minor;
                             /* 次设备号 */
39
    struct device_node *nd;
                           /* 设备节点 */
40
    int led_gpio;
                             /* LED 所使用的 GPIO 编号 */
                            /* 设备状态: 0.设备未使用; >0.设备已经被使用 */
    int dev_stats;
41
42
     spinlock_t lock;
                             /* 自旋锁 */
43 };
44
45 static struct gpioled_dev gpioled; /* led 设备 */
.....
54 static int led_open(struct inode *inode, struct file *filp)
55 {
56
     unsigned long flags;
57
     int ret = 0;
58
59
    /* 自旋锁上锁 */
60
     spin_lock_irqsave(&gpioled.lock, flags);
61
    /* 如果设备被使用了 */
62
    if (gpioled.dev_stats) {
63
64
       printk(KERN_ERR "gpioled: Device is busy!\n");
65
       ret = -EBUSY;
       goto out;
66
```



```
原子哥在线教学: www.yuanzige.com
                                         论坛:www.openedv.com/forum.php
    67
        }
    68
    69
        /* 如果设备没有打开,那么就标记已经打开了 */
    70
         gpioled.dev_stats++;
    71
    72 out:
    73 /* 解锁 */
    74
         spin_unlock_irqrestore(&gpioled.lock, flags);
    75
         return ret;
    76 }
    .....
    125 static int led_release(struct inode *inode, struct file *filp)
    126 {
    127
         unsigned long flags;
    128
    129
         /* 上锁 */
    130
         spin_lock_irqsave(&gpioled.lock, flags);
    131
    132
         /* 关闭驱动文件的时候将 dev_stats 减 1 */
    133
          if (gpioled.dev_stats)
    134
            gpioled.dev_stats--;
    135
    136
         /* 解锁 */
    137
          spin_unlock_irqrestore(&gpioled.lock, flags);
    138
    139
          return 0;
    140 }
    .....
    151 static int __init led_init(void)
    152 {
        .....
    255 /* 9.初始化自旋锁 */
    256 spin_lock_init(&gpioled.lock);
        .....
    273 }
```

第 41 行, dev_stats 变量表示设备状态,如果为 0 的话表示设备还没有被使用,如果大于 0 的话就表示设备已经被使用了。



原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php

第42行,定义自旋锁变量 lock。

第 54~76 行,在 open 函数中使用自旋锁实现对设备的互斥访问,第 60 行调用 spin_lock_irqsave 函数(我们假设 dev_stats 变量在中断函数中被使用到了)获取锁,为了考虑 到驱动兼容性,这里并没有使用 spin_lock 函数来获取锁。第 63 行判断 dev_stats 是否大于 0, 如果是的话表示设备已经被使用了,那么就使用 goto 跳转到 out 处执行 spin_unlock_irqrestore 函数释放锁,并且返回-EBUSY。如果设备没有被使用的话就在第 70 行将 dev_stats 加 1,表 示设备要被使用了,然后调用 spin_unlock_irqrestore 函数释放锁。自旋锁的工作就是保护 dev stats 变量,真正实现对设备互斥访问的是 dev stats。

第 125~140 行,在 release 函数中将 dev_stats 减 1,表示设备被释放了,可以被其他的应 用程序使用。将 dev_stats 减 1 的时候需要自旋锁对其进行保护。

第256行,在驱动入口函数 led_init 中调用 spin_lock_init 函数初始化自旋锁。

3、编写测试 APP

测试 APP 使用 30.1.1 小节中的 atomicApp.c 即可,将 7_atomic 目录中的 atomicApp.c 文件 拷贝到本例程目录中,并将 atomicApp.c 重命名为 spinlockApp.c 即可。

30.2.2 运行测试

1、编译驱动程序

编写 Makefile 文件,将7_atomic 目录下的 Makefile 文件拷贝本实验目录中,打开 Makefile 文件将 obj-m 变量的值改为 spinlock.o, Makefile 内容如下所示:

示例代码 30.2.2 Makefile 文件内容

1 KERN_DIR :=/home/zy/workspace/kernel-driver/linux-xlnx_rebase_v5.4_2020.2

2

3 obj-m :=spinlock.o

4

5 all:

6 make -C \$(KERN_DIR) M=`pwd` modules

7 clean:

8 make -C \$(KERN_DIR) M=`pwd` clean

第3行,设置 obj-m 变量的值为 spinlock.o。

修改完成之后保存退出,输入如下命令编译出驱动模块文件:

make

编译成功以后就会生成一个名为"spinlock.ko"的驱动模块文件。

2、编译测试 APP

输入如下命令编译测试 spinlockApp.c 这个测试程序: \$CC spinlockApp.c -o spinlockApp 编译成功以后就会生成 spinlockApp 这个应用程序。

3、运行测试

将上面编译出来的 spinlock.ko 和 spinlockApp 这两个文件拷贝到 NFS 共享目录下根文件 系统的/lib/modules/5.4.0-xilinx 文件夹中。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

将 SD 启动卡插到开发板中,连接开发板串口、网口和电源线,设置拨码开关为 SD 卡启动模式,然后打开电源,启动开发板。

系统启动完成后,通过串口终端进入"/lib/modules/5.4.0-xilinx"目录下,输入如下命令 加载 spinlock.ko 驱动模块:

depmod //第一次加载驱动的时候需要运行此命令

modprobe spinlock.ko //加载驱动

驱动加载成功以后就可以使用 spinlockApp 软件测试驱动是否工作正常,测试方法和 30.1.2 小节中一样,先输入如下命令让 spinlockApp 软件模拟占用 25 秒的 LED 设备:

./spinlockApp /dev/gpioled 0 & //关闭 LED 灯

紧接着再输入如下命令打开 LED 灯:

./spinlockApp /dev/gpioled 1 //打开 LED 灯

看一下能不能打开 LED 设备,驱动正常工作的话并不会打开 LED 设备,会提示你"file /dev/gpioled open failed!",必须等待第一个 spinlockApp 软件运行完成(25S 计时结束)才可以 再次操作 LED 设备。

如果要卸载驱动的话输入如下命令即可:

rmmod spinlock.ko

30.3 信号量实验

本节我们来使用信号量实现了一次只能有一个应用程序访问 LED 灯,信号量可以导致休眠,因此信号量保护的临界区没有运行时间限制,可以在驱动的 open 函数申请信号量,然后 在 release 函数中释放信号量。但是信号量不能用在中断中,本节实验我们不会在中断中使用 信号量。

本实验对应的例程路径为:开发板资料盘(A 盘)\4_SourceCode\3_Embedded_Linux\Linux 驱动例程\9_semaphore。

30.3.1 实验程序编写

1、修改设备树文件

本章实验是在上一节实验的基础上完成的,同样不需要对设备树做任何的修改。

2、LED 驱动修改

本节实验在上一节实验驱动文件 spinlock.c 的基础上修改而来。在 drivers 目录下新建名为 "9_semaphore"的文件夹,将 8_spinlock 实验中的 spinlock.c 复制到 9_semaphore 文件夹中,并且重命名为 semaphore.c。将原来使用到自旋锁的地方换为信号量即可,其他的内容基本不变,完成以后的 semaphore.c 文件内容如下所示(有省略):

示例代码 30.3.1 semaphore.c 文件代码

•••••

12 #include <linux/types.h>

13 #include <linux/kernel.h>

14 #include <linux/delay.h>

15 #include <linux/ide.h>



原子哥在线教学:www.yuanzig	e.com 论:	坛:www.openedv.com/forum.php
16 #include <linux init.h=""></linux>		
17 #include <linux module.h=""></linux>		
18 #include <linux errno.h=""></linux>		
19 #include <linux gpio.h=""></linux>		
20 #include <asm mach="" map.h=""></asm>		
21 #include <asm uaccess.h=""></asm>		
22 #include <asm io.h=""></asm>		
23 #include <linux cdev.h=""></linux>		
24 #include <linux of.h=""></linux>		
25 #include <linux of_address.h=""></linux>		
26 #include <linux of_gpio.h=""></linux>		
27 #include <linux semaphore.h=""></linux>		
28		
29 #define GPIOLED_CNT	1	/* 设备号个数 */
30 #define GPIOLED_NAME	"gpioled"	/* 名字 */
31		
32 /* dtsled 设备结构体 */		
33 struct gpioled_dev {		
34 dev_t devid;	/* 设备号 */	
35 struct cdev cdev;	/* cdev */	
36 struct class *class;	/* 类 */	
37 struct device *device;	/* 设备 */	
38 int major;	/* 主设备号 */	
39 int minor;	/* 次设备号 */	
40 struct device_node *nd;	/* 设备节点 */	
41 int led_gpio;	/* LED 所使用的	GPIO 编号 */
42 struct semaphore sem;	/* 信号量 */	
43 };		

•••••

54 static int led_open(struct inode *inode, struct file *filp)
55 {
56 /* 获取信号量,如果获取不到则会进入休眠状态 */
57 if (down_interruptible(&gpioled.sem))
58 return -ERESTARTSYS;
59
60 #if 0
61 down(&gpioled.sem);
62 #endif
63



```
原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php
    64
         return 0;
    65 }
    . . . . . .
    114 static int led release(struct inode *inode, struct file *filp)
    115 {
    116 /* 释放信号量,信号量值加1*/
         up(&gpioled.sem);
    117
    118
    119 return 0;
    120 }
    .....
    131 static int __init led_init(void)
    132 {
        .....
   235 /* 初始化信号量 */
    236 sema_init(&gpioled.sem, 1);
       .....
    253 }
```

第29行,要使用信号量必须添加<linux/semaphore.h>头文件。

第42行,在设备结构体中添加一个信号量成员变量 sem。

第 54~65 行,在 open 函数中申请信号量,可以使用 down 函数,也可以使用 down_interruptible 函数。如果信号量值大于或等于1就表示可用,那么应用程序就会开始使用 LED 设备。如果信号量值为0就表示应用程序不能使用 LED 设备,此时应用程序就会进入到 休眠状态。等到信号量值大于或等于1 的时候应用程序就会唤醒,申请到信号量,获取 LED 设备使用权。

第 117 行,在 release 函数中调用 up 函数释放信号量,这样其他因为没有得到信号量而进入休眠状态的应用程序就会唤醒,获取信号量。

第236行,在驱动入口函数中调用 sema_init 函数初始化信号量 sem 的值为1,相当于 sem 是个二值信号量。

总结一下,当信号量 sem为1的时候表示 LED 设备还没有被使用,如果应用程序 A 要使用 LED 设备,先调用 open 函数打开/dev/gpioled,这个时候会获取信号量 sem,获取成功以后 sem 的值减1变为0。如果此时应用程序 B 也要使用 LED 设备,调用 open 函数打开/dev/gpioled 就会因为信号量无效(值为 0)而进入休眠状态。当应用程序 A 运行完毕,调用 close 函数关闭 /dev/gpioled 的时候就会释放信号量 sem,此时信号量 sem 的值就会加1,变为1。信号量 sem



原子哥在线教学:www.yuanzige.com 论坛:www.openedv.com/forum.php 再次有效,表示其他应用程序可以使用 LED 设备了,此时在休眠状态的应用程序 A 就会被唤

再次有效,表示其他应用程序可以使用 LED 设备了,此时在怀眠状态的应用程序 A 就会被唤醒获取到信号量 sem,获取成功以后就开始控制 LED 设备了。

3、编写测试 APP

测试 APP 使用 30.1.1 小节中的 atomicApp.c 即可,将 7_atomic 中的 atomicApp.c 文件到本 实验目录下中,并将 atomicApp.c 重命名为 semaphoreApp.c 即可。

30.3.2 运行测试

1、编译驱动程序

编写 Makefile 文件,将上一小节实验目录下的 Makefile 文件拷贝到本实验目录 9_semaphore 下,打开 Makefile 文件将 obj-m 变量的值改为 semaphore.o, Makefile 内容如下所示:

示例代码 30.3.2 Makefile 文件内容

1 KERN_DIR :=/home/zy/workspace/kernel-driver/linux-xlnx-rebase_v5.4_2020.2

2

3 obj-m :=semaphore.o

4

5 all:

6 make -C \$(KERN_DIR) M=`pwd` modules

7 clean:

8 make -C \$(KERN_DIR) M=`pwd` clean

第3行,设置 obj-m 变量的值为 semaphore.o。

修改完成保存退出,输入如下命令编译出驱动模块文件:

make

编译成功以后就会生成一个名为"semaphore.ko"的驱动模块文件。

2、编译测试 APP

输入如下命令编译测试 semaApp.c 这个测试程序:

\$CC semaphoreApp.c -o semaphoreApp

编译成功以后就会生成 semaphoreApp 这个应用程序。

3、运行测试

将上面编译出来的 semaphore.ko 和 semaphoreApp 这两个文件拷贝到 NFS 共享目录下根文 件系统的/lib/modules/5.4.0-xilinx 文件夹中。

将 SD 启动卡插到开发板中,连接开发板串口、网口和电源线,设置拨码开关为 SD 卡启动模式,然后打开电源,启动开发板。

系统启动完成后,通过串口终端进入"/lib/modules/5.4.0-xilinx"目录下,输入如下命令 加载 semaphore.ko 驱动模块:

depmod		//第一次加载驱动的时候需要运行此命令			
	modprobe semaphore.ko	//加载驱动			
	亚马加鲁马拉马拉马拉马				

驱动加载成功以后就可以使用 semaphoreApp 软件测试驱动是否工作正常,测试方法和 30.1.2 小节中一样,先输入如下命令让 semaphoreApp 软件模拟占用 25S 的 LED 灯:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

./semaphoreApp /dev/gpioled 0 & //关闭 LED 灯

紧接着再输入如下命令打开 LED 灯:

./semaphoreApp /dev/gpioled 1 & //打开 LED 灯

注意两个命令都是运行在后台,第一条命令先获取到信号量,因此可以操作 LED 设备,将 LED 灯打开,并且占有 25S。第二条命令因为获取信号量失败而进入休眠状态,等待第一条命令运行完毕并释放信号量以后才拥有 LED 设备使用权,将 LED 灯关闭,运行结果如图 30.3.1 所示:

root@ALIENTEK-ZYNQ-drive	er:/lib/modules/5.4.0-xilinx#						
root@ALIENTEK-ZYNQ-drive	er:/lib/modules/5.4.0-xilinx#						
root@ALIENTEK-ZYNQ-drive	er:/lib/modules/5.4.0-xilinx#	./semaphoreApp	/dev/gpioled 0 &				
[1] 521							
root@ALIENTEK-ZYNQ-drive	./semaphoreApp	/dev/gpioled 1 &					
[2] 522	[2] 522						
root@ALIENTEK-ZYNQ-drive	root@ALIENTEK-ZYNO-driver:/lib/modules/5.4.0-xilinx# App running times:1						
App running times:2							
App running times:3							
App running times:4							
App running times:5							
App running finished!							
App running times:1							
App running times:2							
App running times:3							
App running times:4							
App running times:5							
App running finished!							

图 30.3.1 两条命令运行过程

如果要卸载驱动的话输入如下命令即可:

rmmod semaphore.ko

30.4 互斥体实验

前面我们使用原子操作、自旋锁和信号量实现了对 LED 设备的互斥访问,但是最适合互斥的就是互斥体 mutex 了。本节我们来学习一下如何使用 mutex 实现对 LED 设备的互斥访问。

本实验对应的例程路径为:开发板资料盘(A 盘)\4_SourceCode\3_Embedded_Linux\Linux 驱动例程\10_mutex。

30.4.1 实验程序编写

.....

1、修改设备树文件

本章实验是在上一节实验的基础上完成的,同样不需要对设备树做任何的修改。

2、LED 驱动修改

本节实验在上一节实验驱动文件 semaphore.c 的基础上修改而来。在 drivers 目录下新建名为 "10_mutex"的文件夹,将 9_semaphore 实验中的 semaphore.c 复制到 10_mutex 文件夹中,并且重命名为 mutex.c。将原来使用到信号量的地方换为 mutex 即可,其他的内容基本不变,完成以后的 mutex.c 文件内容如下所示(有省略):

示例代码 30.4.1 mutex.c 文件代码

29 #define GPIOLED_CNT 1 /* 设备号个数 */



```
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
                                                /* 名字 */
    30 #define GPIOLED_NAME
                                    "gpioled"
    31
    32 /* dtsled 设备结构体 */
    33 struct gpioled_dev {
    34 dev_t devid;
                               /* 设备号 */
                               /* cdev */
    35 struct cdev cdev;
                               /* 类 */
    36 struct class *class;
    37 struct device *device;
                               /* 设备 */
                                /* 主设备号 */
    38 int major;
                               /* 次设备号 */
    39 int minor;
        struct device_node *nd; /* 设备节点 */
    40
                               /* LED 所使用的 GPIO 编号 */
    41 int led_gpio;
    42 struct mutex lock; /* 互斥体 */
    43 };
    44
    45 static struct gpioled_dev gpioled; /* led 设备 */
    .....
    54 static int led_open(struct inode *inode, struct file *filp)
    55 {
    56 /* 获取互斥体,可以被信号打断 */
        if (mutex_lock_interruptible(&gpioled.lock))
    57
    58
           return -ERESTARTSYS;
    59
    60 #if 0
        mutex_lock(&gpioled.lock); /* 不能被信号打断 */
    61
    62 #endif
    63
    64
        return 0;
    65 }
    .....
    114 static int led_release(struct inode *inode, struct file *filp)
    115 {
    116 /* 释放互斥锁 */
    117
         mutex_unlock(&gpioled.lock);
    118
    119 return 0;
    120 }
```



E点原子

```
131 static int __init led_init(void)
132 {
133     const char *str;
134     int ret;
......
```

235 /* 初始化互斥体 */

236 mutex_init(&gpioled.lock);

•••••

```
252 return ret;
```

253 }

.....

第42行, 定义互斥体 lock。

第 54~65 行,在 open 函数中调用 mutex_lock_interruptible 或者 mutex_lock 获取 mutex,成 功的话就表示可以使用 LED 设备,失败的话就会进入休眠状态,和信号量一样。

第 117 行,在 release 函数中调用 mutex_unlock 函数释放 mutex,这样其他应用程序就可以获取 mutex 了。

第236行,在驱动入口函数中调用 mutex_init 初始化 mutex。

互斥体和二值信号量类似,只不过互斥体是专门用于互斥访问的。

3、编写测试 APP

测试 APP 使用 30.1.1 小节中的 atomicApp.c 即可,将 7_atomic 中的 atomicApp.c 文件拷贝 到本实验目录 10_mutex 中,并将 atomicApp.c 重命名为 mutexApp.c 即可。

30.4.2 运行测试

1、编译驱动程序

编写 Makefile 文件,将上一节实验目录下的 Makefile 文件拷贝到本实验目录下,打开 Makefile 文件将 obj-m 变量的值改为 mutex.o, Makefile 内容如下所示:

示例代码 30.4.2 Makefile 文件

1 KERN_DIR :=/home/zy/workspace/kernel-driver/linux-xlnx-xlnx_rebase_v5.4_2020.2

```
2
3 obj-m :=mutex.o
4
```

5 all:

6 make -C \$(KERN_DIR) M=`pwd` modules

7 clean:

8 make -C \$(KERN_DIR) M=`pwd` clean

第3行,设置 obj-m 变量的值为 mutex.o。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

Makefile 修改完成之后保存退出,输入如下命令编译出驱动模块文件:

make

编译成功以后就会生成一个名为"mutex.ko"的驱动模块文件。

2、编译测试 APP

输入如下命令编译测试 mutexApp.c 这个测试程序:

\$CC mutexApp.c -o mutexApp

编译成功以后就会生成 mutexApp 这个应用程序。

3、运行测试

将上面编译出来的 mutex.ko 和 mutexApp 这两个文件拷贝到 NFS 共享目录下根文件系统 的/lib/modules/5.4.0-xilinx 文件夹中。

将 SD 启动卡插到开发板中,连接开发板串口、网口和电源线,设置拨码开关为 SD 卡启动模式,然后打开电源,启动开发板。

系统启动完成后,通过串口终端进入"/lib/modules/5.4.0-xilinx"目录下,输入如下命令 加载 mutex.ko 驱动模块:

depmod //第一次加载驱动的时候需要运行此命令

modprobe mutex.ko //加载驱动

驱动加载成功以后就可以使用 mutexApp 软件测试驱动是否工作正常,测试方法和 30.3.2 中测试信号量的方法一样,这里不详细介绍了。关闭和打开 LED 命令如下:

./mutexApp /dev/gpioled 0 & //关闭 LED 灯
./mutexApp /dev/gpioled 1 & //打开 LED 灯
如果要卸载驱动的话输入如下命令即可:
rmmod mutex.ko



正点原子

第三十一章 Linux 按键输入实验

在前几章我们都是使用的 GPIO 输出功能,还没有用过 GPIO 输入功能,本章我们就来学 习一下如果在 Linux 下编写 GPIO 输入驱动程序。领航者开发板上有 5 个按键, 四个轻触式按 键和一个触摸按键,本章我们就以 PS KEY0 按键为例,使用此按键来完成 GPIO 输入驱动程 序,同时利用第二十九章讲的互斥锁来对按键值进行保护。


原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

31.1 Linux 下按键驱动原理

按键驱动和 LED 驱动原理上来讲基本都是一样的,都是操作 GPIO,只不过一个是读取 GPIO 的高低电平,一个是从 GPIO 输出高低电平。本章我们实现按键输入,在驱动程序中实 现 read 函数,读取按键值并将数据发送给上层应用测试程序,在 read 函数中,使用了互斥锁 对读数据过程进行了保护,后面会讲解为什么使用互斥锁进行保护。Linux 下的按键驱动原理 很简单,接下来开始编写驱动。

注意,本章例程只是为了演示 Linux 下 GPIO 输入驱动的编写,实际中的按键驱动并不会 采用本章中所讲解的方法,Linux 下的 input 子系统专门用于输入设备!

31.2 硬件原理图分析

打开领航者底板原理图,找到 PS_KEY0 按键原理图,如下所示:

	KEY	+ <u>3.3</u> V		
	PL RESET PL KEY0 PL KEY0 PL KEY1 PS KEY0 PS KEY0 PS KEY1 PS KEY1 PS KEY1	$\frac{10K}{10K} \frac{R5}{R6}$		
	SD D3	J2	88	00
PS KEY0	PS MIO12	J2	90	00
PS LED0	PS MIO7	J2	92	90
PS KEY1	PS MIO11	J2	94	92
OTG RESETN	PS MIO9	J2	96	96
		10	00	

图 31.2.1 按键原理图

从原理图可知,当 PS_KEY0 按键按下时,对应的管脚 MIO12 为低电平状态,松开的时候 MIO12 为高电平状态,所以可以通过读取 MIO 管脚的电平状态来判断按键是否被按下或松开。

31.3 实验程序编写

本实验对应的例程路径为:开发板资料盘(A 盘)\4_SourceCode\3_Embedded_Linux\Linux 驱动例程\11_key。

31.3.1 修改设备树文件

打开 system-user.dtsi 文件,在根节点"/"下创建一个按键节点,节点名为"key",节点 内容如下:

示例代码 31.3.1 创建 key 节点



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

34 key {

35 compatible = "alientek,key";

36 status = "okay";

37 key-gpio = <&gpio0 12 GPIO_ACTIVE_LOW>;

38 };

这个节点内容很简单。

第 35 行,设置节点的 compatible 属性为 "alientek,key"。

第37行,key-gpio 属性指定了 PS_KEY0 按键所使用的 GPIO。

设备树编写完成以后使用,在 linux 内核源码目录下执行下面这条命令重新编译设备树: make dtbs

<pre>zy@zy-virtual-machine:~/workspace/kernel-driver/linux-xlnx_rebase_v5.</pre>
4_2020.2\$
zy@zy-virtual-machine:~/workspace/kernel-driver/linux-xlnx_rebase_v5.
4_2020.2\$ make dtbs
DTC arch/arm/boot/dts/system-top.dtb
zy@zy-virtual-machine:~/workspace/kernel-driver/linux-xlnx-xlnx_rebase_v5.
4_2020.2\$
zy@zy-virtual-machine:~/workspace/kernel-driver/linux-xlnx_rebase_v5.
4_2020.2\$

图 31.3.1 重新编译设备树

然后将新编译出来的 system-top.dtb 文件重命名为 system.dtb,将 system.dtb 文件拷贝到 SD 启动卡的 Fat 分区,替换以前的 system.dtb 文件,替换完成之后重启开发板。启动成功以 后进入"/proc/device-tree"目录中查看"key"节点是否存在,如果存在的话就说明设备树基 本修改成功(具体还要驱动验证),结果如图 31.3.2 所示:

root@ALIENTEK-Z root@ALIENTEK-Z	YNQ-driver:~# cd YNQ-driver:/proc/	/proc/device-tre /device-tree#	e		
root@ALIENIEK-Z	YNQ-driver:/proc/	device-tree# ls	fixedrogulator	lod	nomo
#size-cells	amba pl	compatible	fpga-full	memorv	pmu@f8891000
aliases	beeper	cpus	key	model	replicator
root@ALIENTEK-ZYNQ-driver:/proc/device-tree#					
root@ALIENTEK-ZYNQ-driver:/proc/device-tree# _					

图 31.3.2 key 节点

31.3.2 按键驱动程序编写

设备树准备好以后就可以编写驱动程序了,在 drivers 目录下新建名为"11_key"的文件 夹,然后在 11_key 文件夹里面新建一个名为 key.c 的源文件,在 key.c 里面输入如下内容:

示例代码 31.3.2 key.c 文件代码

2 Copyright © ALIENTEK Co., Ltd. 1998-2029. All rights reserved.

- 3 文件名 : key.c
- 4 作者 : 邓涛
- 5 版本 : V1.0
- 6 描述 : Linux 按键输入驱动实验
- 7 其他 :无



Contraction of C	
原子哥在线教学: www.yuanzi	ge.com 论坛:www.openedv.com/forum.php
8 论坛 :www.openedv.com	1
9 日志 :初版 V1.0 2019/1/	30 邓涛创建
10 ***************	***************************************
11	
12 #include <linux types.h=""></linux>	
13 #include <linux kernel.h=""></linux>	
14 #include <linux delay.h=""></linux>	
15 #include <linux ide.h=""></linux>	
<pre>16 #include <linux init.h=""></linux></pre>	
17 #include <linux module.h=""></linux>	
<pre>18 #include <linux errno.h=""></linux></pre>	
19 #include <linux gpio.h=""></linux>	
20 #include <asm mach="" map.h=""></asm>	
21 #include <asm uaccess.h=""></asm>	
22 #include <asm io.h=""></asm>	
23 #include <linux cdev.h=""></linux>	
24 #include <linux of.h=""></linux>	
25 #include <linux of_address.h<="" td=""><td>></td></linux>	>
26 #include <linux of_gpio.h=""></linux>	
27	
28 #define KEY_CNT	1 /* 设备号个数 */
29 #define KEY_NAME	"key" /* 名字 */
30	
31 /* dtsled 设备结构体 */	
32 struct key_dev {	
dev_t devid;	/* 设备号 */
34 struct cdev cdev;	/* cdev */
35 struct class *class;	/* 类 */
36 struct device *device;	/* 设备 */
37 int major;	/* 主设备号 */
38 int minor;	/* 次设备号 */
<pre>39 struct device_node *nd;</pre>	/* 设备节点 */
40 int key_gpio;	/* GPIO 编号 */
41 int key_val;	/* 按键值 */
42 struct mutex mutex;	/* 互斥锁 */
43 };	
44	
45 static struct key_dev key;	/* led 设备 */
46	
47 /*	
48 * @description	:打开设备



```
原子哥在线教学: www.yuanzige.com
                                         论坛:www.openedv.com/forum.php
    49 * @param – inode
                              :传递给驱动的 inode
                              :设备文件, file 结构体有个叫做 private_data 的成员变量
    50 * @param – filp
                              一般在 open 的时候将 private_data 指向设备结构体。
    51 *
                              :0 成功:其他 失败
    52 * @return
    53 */
    54 static int key open(struct inode *inode, struct file *filp)
    55 {
    56 return 0;
    57 }
    58
    59 /*
                              :从设备读取数据
    60 * @description
    61 * @param – filp
                              :要打开的设备文件(文件描述符)
    62 * @param – buf
                              :返回给用户空间的数据缓冲区
    63 * @param – cnt
                              :要读取的数据长度
    64 * @param – offt
                              :相对于文件首地址的偏移
                              :读取的字节数,如果为负值,表示读取失败
    65 * @return
    66 */
    67 static ssize_t key_read(struct file *filp, char __user *buf,
    68
            size_t cnt, loff_t *offt)
    69 {
    70
        int ret = 0;
    71
    72 /* 互斥锁上锁 */
    73
        if (mutex_lock_interruptible(&key.mutex))
    74
          return -ERESTARTSYS;
    75
    76 /* 读取按键数据 */
    77
        if (!gpio_get_value(key.key_gpio)) {
    78
          while(!gpio_get_value(key.key_gpio));
    79
          key.key_val = 0x0;
       } else
    80
    81
          key.key_val = 0xFF;
    82
    83
        /* 将按键数据发送给应用程序 */
    84
        ret = copy_to_user(buf, &key.key_val, sizeof(int));
    85
        /* 解锁 */
    86
    87
        mutex_unlock(&key.mutex);
    88
    89
        return ret;
```

```
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
    90 }
    91
    92 /*
    93 * @description
                              : 向设备写数据
                              :设备文件,表示打开的文件描述符
    94 * @param – filp
    95 * @param – buf
                              :要写给设备写入的数据
                              :要写入的数据长度
    96 * @param – cnt
    97 * @param – offt
                              :相对于文件首地址的偏移
    98 * @return
                               :写入的字节数,如果为负值,表示写入失败
    <mark>99</mark> */
    100 static ssize_t key_write(struct file *filp, const char __user *buf,
             size_t cnt, loff_t *offt)
    101
    102 {
    103 return 0;
    104 }
    105
    106 /*
    107 * @description
                            :关闭/释放设备
    108 * @param – filp
                              :要关闭的设备文件(文件描述符)
    109 * @return
                              :0 成功;其他 失败
    110 */
    111 static int key_release(struct inode *inode, struct file *filp)
   112 {
    113 return 0;
    114 }
    115
   116 /* 设备操作函数 */
   117 static struct file_operations key_fops = {
    118 .owner = THIS_MODULE,
    119 .open
                  = key_open,
    120 .read
                   = key_read,
    121 .write
                   = key_write,
    122 .release
                   = key_release,
    123 };
    124
    125 static int __init mykey_init(void)
   126 {
   127 const char *str;
    128 int ret;
    129
    130 /* 初始化互斥锁 */
```

正点原子



```
原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php
    131
          mutex_init(&key.mutex);
    132
    133 /* 1.获取 key 节点 */
    134
         key.nd = of_find_node_by_path("/key");
    135
         if(NULL == key.nd) {
    136
            printk(KERN_ERR "key: Failed to get key node\n");
    137
            return -EINVAL;
    138
         }
    139
         /* 2.读取 status 属性 */
    140
          ret = of_property_read_string(key.nd, "status", &str);
    141
    142
         if(!ret) {
    143
            if (strcmp(str, "okay"))
    144
              return -EINVAL;
    145
         }
    146
         /* 3.获取 compatible 属性值并进行匹配 */
    147
    148
          ret = of_property_read_string(key.nd, "compatible", &str);
    149
         if(ret) {
    150
            printk(KERN_ERR "key: Failed to get compatible property\n");
    151
            return ret;
    152
          }
    153
    154
          if (strcmp(str, "alientek,key")) {
    155
            printk(KERN_ERR "key: Compatible match failed\n");
    156
            return -EINVAL;
    157
          }
    158
    159
          printk(KERN_INFO "key: device matching successful!\r\n");
    160
          /* 4.获取设备树中的 key-gpio 属性,得到按键所使用的 GPIO 编号 */
    161
          key.key_gpio = of_get_named_gpio(key.nd, "key-gpio", 0);
    162
    163
          if(!gpio_is_valid(key.key_gpio)) {
            printk(KERN_ERR "key: Failed to get key-gpio\n");
    164
    165
            return -EINVAL;
    166
          }
    167
    168
          printk(KERN_INFO "key: key-gpio num = % d\r\n", key.key_gpio);
    169
    170
         /* 5.申请 GPIO */
```

```
171
     ret = gpio_request(key.key_gpio, "Key Gpio");
```



```
原子哥在线教学: www.yuanzige.com
                                              论坛:www.openedv.com/forum.php
    172
         if (ret) {
    173
            printk(KERN_ERR "key: Failed to request key-gpio\n");
    174
            return ret;
    175
         }
    176
    177
         /* 6.将 GPIO 设置为输入模式 */
    178
         gpio_direction_input(key.key_gpio);
    179
    180
         /* 7.注册字符设备驱动 */
    181
         /* 创建设备号 */
    182
         if (key.major) {
    183
            key.devid = MKDEV(key.major, 0);
            ret = register_chrdev_region(key.devid, KEY_CNT, KEY_NAME);
    184
    185
            if (ret)
    186
              goto out1;
    187
         } else {
    188
            ret = alloc_chrdev_region(&key.devid, 0, KEY_CNT, KEY_NAME);
    189
            if (ret)
    190
              goto out1;
    191
    192
            key.major = MAJOR(key.devid);
    193
            key.minor = MINOR(key.devid);
    194
         }
    195
    196
         printk(KERN_INFO "key: major=%d, minor=%d\r\n", key.major, key.minor);
    197
          /* 初始化 cdev */
    198
    199
         key.cdev.owner = THIS_MODULE;
    200
         cdev_init(&key.cdev, &key_fops);
    201
    202
          /* 添加 cdev */
    203
         ret = cdev_add(&key.cdev, key.devid, KEY_CNT);
    204
         if (ret)
    205
            goto out2;
    206
    207
         /* 创建类 */
   208
         key.class = class_create(THIS_MODULE, KEY_NAME);
    209
         if (IS_ERR(key.class)) {
    210
            ret = PTR_ERR(key.class);
    211
            goto out3;
    212 }
```



```
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
   213
   214
         /* 创建设备 */
   215 key.device = device_create(key.class, NULL,
   216
               key.devid, NULL, KEY_NAME);
   217
         if (IS_ERR(key.device)) {
           ret = PTR_ERR(key.device);
   218
   219
           goto out4;
   220 }
    221
    222
         return 0;
   223
   224 out4:
   225
         class_destroy(key.class);
   226
   227 out3:
   228
         cdev_del(&key.cdev);
   229
   230 out2:
   231
         unregister_chrdev_region(key.devid, KEY_CNT);
   232
   233 out1:
   234
         gpio_free(key.key_gpio);
   235
   236 return ret;
   237 }
   238
   239 static void __exit mykey_exit(void)
   240 {
   241 /* 注销设备 */
   242
         device_destroy(key.class, key.devid);
    243
    244 /* 注销类 */
   245
         class_destroy(key.class);
   246
   247
         /* 删除 cdev */
   248
         cdev_del(&key.cdev);
   249
         /* 注销设备号 */
   250
   251
         unregister_chrdev_region(key.devid, KEY_CNT);
    252
    253 /* 释放 GPIO */
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

254 gpio_free(key.key_gpio);

255 }

256

257 /* 驱动模块入口和出口函数注册 */

258 module_init(mykey_init);

259 module_exit(mykey_exit);

260

261 MODULE_AUTHOR("DengTao <773904075@qq.com>");

262 MODULE_DESCRIPTION("Alientek Gpio Key Driver");

263 MODULE_LICENSE("GPL");

第 32~43 行,结构体 key_dev 为按键的设备结构体,第 40 行的 key_gpio 表示按键对应的 GPIO 编号,第 41 行 key_val 用来保存读取到的按键值,第 42 行定义了一个互斥锁变量 mutex,用来保护按键读取过程。

第67~90行,在key_read函数中,通过gpio_get_value函数读取按键值,如果当前为按下状态,则使用while循环等待按键松开,松开之后将key_val变量置为0x0,从按键按下状态到松开状态视为一次有效状态;如果当前为松开状态,则将key_val变量置为0xFF,表示为无效状态。使用 copy_to_user 函数将 key_val 值发送给上层应用;第73行,调用mutex_lock_interruptible函数上锁(互斥锁),第87行解锁,对整个读取按键过程进行保护,因为在于用于保存按键值的key_val是一个全局变量,如果上层有多个应用对按键进行了读取操作,将会出现第二十九章说到的并发访问,这对系统来说是不利的,所以这里使用了互斥锁进行了保护。应用程序通过 read函数读取按键值的时候 key_read函数就会执行!

第131行,调用 mutex_init 函数初始化互斥锁。

第 178 行,调用 gpio_direction_input 函数将按键对应的 GPIO 设置为输入模式。

key.c 文件代码很简单,重点就是 key_read 函数读取按键值,要对读取过程进行保护。

31.3.3 编写测试 APP

×

在本章实验目录下新建名为 keyApp.c 的文件, 然后输入如下所示内容:

示例代码 31.3.3 keyApp.c 文件代码

2 Copyright © ALIENTEK Co., Ltd. 1998-2029. All rights reserved.

3	又件名	: keyApp.c
4	作者	:邓涛
5	版本	: V1.0
6	描述	:按键测试应用程序
7	其他	:无
8	使用方法	: ./keyApp /dev/key
9	论坛	: www.openedv.com
10	日志	:初版 V1.0 2019/1/30 邓涛创建
11	******	***************************************
12		

13 #include <stdio.h>



```
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
    14 #include <unistd.h>
    15 #include <sys/types.h>
    16 #include <sys/stat.h>
    17 #include <fcntl.h>
    18 #include <stdlib.h>
    19 #include <string.h>
    20
    21 /*
    22 * @description
                                 : main 主程序
    23 * @param – argc
                                 : argv 数组元素个数
    24 * @param – argv
                                 :具体参数
    25 * @return
                                 :0 成功;其他 失败
    26 */
    27 int main(int argc, char *argv[])
    28 {
    29
        int fd, ret;
    30
        int key_val;
    31
    32 /* 判断传参个数是否正确 */
    33 if(2 != argc) {
           printf("Usage:\n"
    34
    35
               "\t./keyApp /dev/key\n"
    36
               );
    37
           return -1;
    38
        }
    39
        /* 打开设备 */
    40
         fd = open(argv[1], O_RDONLY);
    41
         if(0 > fd) 
    42
    43
           printf("ERROR: %s file open failed!\n", argv[1]);
    44
           return -1;
    45
        }
    46
         /* 循环读取按键数据 */
    47
    48
         for(;;){
    49
    50
           read(fd, &key_val, sizeof(int));
    51
           if (0x0 == key_val)
             printf("PS_KEY0 Press, value = 0x%x\n", key_val);
    52
    53
        }
    54
```



原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php

55 /* 关闭设备 */

56 close(fd);

57 return 0;

```
58 }
```

第 48~53 行,循环读取/dev/key 文件,也就是循环读取按键值,如果读取到的值为 0,则 表示是一次有效的按键(按下之后松开标记为一次有效状态),并打印信息出来。

31.4 运行测试

31.4.1 编译驱动程序和测试 APP

1、编译驱动程序

编写 Makefile 文件,将 10_mutex 实验目录下的 Makefile 文件拷贝到本实验目录下,打开 Makefile 文件,将 obj-m 变量的值改为 key.o,修改完之后 Makefile 内容如下所示:

示例代码 31.4.1 Makefile.c 文件内容

1 KERN DIR :=/home/zy/workspace/kernel-driver/linux-xlnx-xlnx rebase v5.4 2020.2

2

3 obj-m :=key.o

4

5 all:

6 make -C \$(KERN_DIR) M=`pwd` modules

7 clean:

8 make -C \$(KERN_DIR) M=`pwd` clean

第3行,设置obj-m变量的值为kev.o。

Makefile 文件修改完成之后保存退出,在本实验目录下输入如下命令编译出驱动模块文 件:

make

编译成功以后就会生成一个名为"key.ko"的驱动模块文件。

2、编译测试 APP

输入如下命令编译测试 keyApp.c 这个测试程序:

\$CC keyApp.c -o keyApp

编译成功以后就会生成 keyApp 这个应用程序。

31.4.2 运行测试

将上面编译出来的 key.ko 和 keyApp 这两个文件拷贝到 NFS 共享目录下根文件系统的 /lib/modules/5.4.0-xilinx 文件夹中。

将 SD 启动卡插到开发板中,连接开发板串口、网口和电源线,设置拨码开关为 SD 卡启 动模式,然后打开电源,启动开发板。

系统启动完成后,通过串口终端进入"/lib/modules/5.4.0-xilinx"目录下,输入如下命令 加载 key.ko 驱动模块:

depmod //第一次加载驱动的时候需要运行此命令

//加载驱动



原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php

如下所示:

modprobe key.ko



图 31.4.1 加载 key 驱动模块

驱动加载成功以后如下命令来测试:

./keyApp /dev/key

按下开发板上的 PS_KEY0 按键, keyApp 就会获取并且输出按键信息, 如图 31.4.2 所示:

root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx#
root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx#
<pre>root@ALIENTEK-ZYNO-driver:/lib/modules/5.4.0-xilinx# ./keyApp /dev/key</pre>
PS_KEY0 Press, value = 0x0
PS_KEYO Press, value = 0x0
PS_KEY0 Press, value = 0x0
^C
root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx#

图 31.4.2 打印按键值

从图 31.4.2 可以看出,当我们按下 PS_KEYO 再松开以后就会打印出"KEYO Press, value = 0x0",表示这是一次完整的按键按下、松开事件。但是大家在测试过程可能会发现,有时 候按下 PS_KEY0 会输出好几行"KEY0 Press, value = 0x0",这是因为我们的代码没有做按键 消抖处理,是属于正常情况。

如果要卸载驱动的话输入如下命令即可:

rmmod key.ko

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

正点原子

第三十二章 Linux 内核定时器实验

定时器是我们最常用到的功能,一般用来完成定时功能,本章我们就来学习一下 Linux 内核提供的定时器 API 函数,通过这些定时器 API 函数我们可以完成很多要求定时的应用。 Linux 内核也提供了短延时函数,比如微秒、纳秒、毫秒延时函数,本章我们就来学习一下这 些和时间有关的功能。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

32.1 Linux 时间管理和内核定时器简介

32.1.1 内核时间管理简介

学习过 UCOS 或 FreeRTOS 的同学应该知道, UCOS 或 FreeRTOS 是需要一个硬件定时器 提供系统时钟,一般使用 Systick 作为系统时钟源。同理, Linux 要运行,也是需要一个系统 时钟的,至于这个系统时钟是由哪个定时器提供的,笔者没有去研究过 Linux 内核,但是在 Cortex-A7 内核中有个通用定时器,在《Cortex-A7 Technical ReferenceManua.pdf》的 "9:Generic Timer"章节有简单的讲解,关于这个通用定时器的详细内容,可以参考《ARM ArchitectureReference Manual ARMv7-A and ARMv7-R edition.pdf》的"chapter B8 The Generic Timer"章节。这个通用定时器是可选的,按照笔者学习 FreeRTOS 和 STM32 的经验,猜测 Linux 会将这个通用定时器作为 Linux 系统时钟源(前提是 SOC 得选配这个通用定时器)。具体 是怎么做的笔者没有深入研究过,这里仅仅是猜测!不过对于我们 Linux 驱动编写者来说, 不需要深入研究这些具体的实现,只需要掌握相应的 API 函数即可,除非你是内核编写者或 者内核爱好者。

Linux 内核中有大量的函数需要时间管理,比如周期性的调度程序、延时程序、对于我们驱动编写者来说最常用的定时器。硬件定时器提供时钟源,时钟源的频率可以设置,设置好以后就周期性的产生定时中断,系统使用定时中断来计时。中断周期性产生的频率就是系统频率,也叫做节拍率(tick rate)(有的资料也叫系统频率),比如 1000Hz, 100Hz 等等说的就是系统节拍率。系统节拍率是可以设置的,单位是 Hz,我们在编译 Linux 内核的时候可以通过图形化界面设置系统节拍率,在内核源码目录下执行下面这条命令进入到 menuconfig 配置界面:

make menuconfig
按照如下路径打开配置界面:
-> Kernel Features
-> Timer frequency (<choice> [=y])
选中 "Timer frequency",打开以后如下图所示:



图 32.1.1 系统节拍率设置

从上图中可以看出,可选的系统节拍率为 100Hz、200Hz、250Hz、300Hz、500Hz 和 1000Hz,默认情况下选择 100Hz。设置好以后打开 Linux 内核源码根目录下的.config 文件,在此文件中有如下图所示定义:

领航者 ZYNQ 之嵌入式 Linux 开发指南
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
417 CONFIG HZ_FIXED=0
418 CONFIG HZ 100=y
419 # CONFIG HZ_200 is not set
420 # CONFIG HZ_250 is not set
421 # CONFIG HZ_300 is not set
422 # CONFIG HZ_500 is not set
423 # CONFIG HZ_1000 is not set
423 # CONFIG HZ_1000 is not set
424 CONFIG HZ=100 → 系统节拍率
425 CONFIG SCHED HRTICK=y
426 # CONFIG THUMB2 KERNEL is not set

图 32.1.2 系统节拍率

图 32.1.2 中的 CONFIG_HZ 为 100, Linux 内核会使用 CONFIG_HZ 来设置自己的系统时 钟。打开文件 include/asm-generic/param.h, 有如下内容:

示例代码 32.1.1 include/asm-generic/param.h 文件代码段

6 # undef HZ

7 # define HZ CONFIG_HZ

8 # define USER_HZ 100

9 # define CLOCKS_PER_SEC (USER_HZ)

第7行定义了一个宏HZ,宏HZ就是 CONFIG_HZ,因此 HZ=100,我们后面编写 Linux 驱动的时候会常常用到 HZ,因为 HZ 表示一秒的节拍数,也就是频率。

大多数初学者看到系统节拍率默认为 100Hz 的时候都会有疑问,怎么这么小? 100Hz 是可选的节拍率里面最小的。为什么不选择大一点的呢?这里就引出了一个问题:高节拍率和低节拍率的优缺点:

①、高节拍率会提高系统时间精度,如果采用 100Hz 的节拍率,时间精度就是 10ms,采 用 1000Hz 的话时间精度就是 1ms,精度提高了 10 倍。高精度时钟的好处有很多,对于那些 对时间要求严格的函数来说,能够以更高的精度运行,时间测量也更加准确。

②、高节拍率会导致中断的产生更加频繁,频繁的中断会加剧系统的负担,1000Hz 和 100Hz 的系统节拍率相比,系统要花费 10 倍的"精力"去处理中断。中断服务函数占用处理 器的时间增加,但是现在的处理器性能都很强大,所以采用 1000Hz 的系统节拍率并不会增加 太大的负载压力。根据自己的实际情况,选择合适的系统节拍率,本教程我们全部采用默认 的 100Hz 系统节拍率。

Linux 内核使用全局变量 jiffies 来记录系统从启动以来的系统节拍数,系统启动的时候会将 jiffies 初始化为 0, jiffies 定义在文件 include/linux/jiffies.h 中,定义如下:

示例代码 32.1.2 include/jiffies.h 文件代码段

76 extern u64 __jiffy_data jiffies_64;

77 extern unsigned long volatile __jiffy_data jiffies;

第76行, 定义了一个64位的jiffies_64。

第77行, 定义了一个 unsigned long 类型的 32 位的 jiffies。

jiffies_64 和 jiffies 其实是同一个东西, jiffies_64 用于 64 位系统, 而 jiffies 用于 32 位系 统。为了兼容不同的硬件, jiffies 其实就是 jiffies_64 的低 32 位, jiffies_64 和 jiffies 的结构如 图 32.1.3 所示:





jiffies_64

图 32.1.3 jiffies_64 和 jiffies 结构图

当我们访问 jiffies 的时候其实访问的是 jiffies_64 的低 32 位,使用 get_jiffies_64 这个函数 可以获取 jiffies_64 的值。在 32 位的系统上读取 jiffies 的值,在 64 位的系统上 jiffes 和 jiffies_64 表示同一个变量,因此也可以直接读取 jiffies 的值。所以不管是 32 位的系统还是 64 位系统,都可以使用 jiffies。

前面说了 HZ 表示每秒的节拍数, jiffies 表示系统运行的 jiffies 节拍数,所以 jiffies/HZ 就 是系统运行时间,单位为秒。不管是 32 位还是 64 位的 jiffies,都有溢出的风险,溢出以后会 重新从 0 开始计数,相当于绕回来了,因此有些资料也将这个现象也叫做绕回。假如 HZ 为最 大值 1000 的时候,32 位的 jiffies 只需要 49.7 天就发生了绕回,对于 64 为的 jiffies 来说大概需 要 5.8 亿年才能绕回,因此 jiffies_64 的绕回忽略不计。处理 32 位 jiffies 的绕回显得尤为重要, Linux 内核提供了如表 32.1.1.1 所示的几个 API 函数来处理绕回。

函数	描述		
time_after(unkown,			
known)			
time_before(unkown,			
known)	unkown 通常为 jiffies, known 通常是需要对比		
time_after_eq(unkown,	的值。		
known)			
time_before_eq(unkown			
, known)			

表 32.1.1.1 处理绕回的 API 函数

如果 unkown 超过 known 的话, time_after 函数返回真, 否则返回假。如果 unkown 没有超过 known 的话 time_before 函数返回真, 否则返回假。time_after_eq 函数和 time_after 函数类似, 只是多了判断等于这个条件。同理, time_before_eq 函数和 time_before 函数也类似。比如我们要判断某段代码执行时间有没有超时,此时就可以使用如下所示代码:

示例代码 32.1.3 使用 jiffies 判断超时



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

12 /* 超时发生 */

13 }

timeout 就是超时时间点,比如我们要判断代码执行时间是不是超过了 2 秒,那么超时时间点就是 jiffies+(2*HZ),如果 jiffies 大于 timeout 那就表示超时了,否则就是没有超时。第 4~6 行就是具体的代码段。第 9 行通过函数 time_before 来判断 jiffies 是否小于 timeout,如果小于的话就表示没有超时。

为了方便开发,Linux 内核提供了几个 jiffies 和 ms、us、ns 之间的转换函数,如表 32.1.1.2 所示:

描述	
将 jiffies 类型的参数 j 分别转换为对	
」应的毫秒、微秒、纳秒。	
将毫秒、微秒、纳秒转换为 jiffies 类	
_ 型。	

表 32.1.1.2 jiffies 和 ms、us、ns 之间的转换函数

32.1.2 内核定时器简介

定时器是一个很常用的功能,需要周期性处理的工作都要用到定时器。Linux内核定时器 采用系统时钟来实现,用软件的方式来实现,并不是SoC提供硬件定时器。Linux内核定时器 使用很简单,只需要提供超时时间(相当于定时值)和定时处理函数即可,当超时时间到了以 后设置的定时处理函数就会执行,和我们使用硬件定时器的套路一样,只是使用内核定时器 不需要做一大堆的寄存器初始化工作。在使用内核定时器的时候要注意一点,内核定时器并 不是周期性运行的,超时以后就会自动关闭,因此如果想要实现周期性定时,那么就需要在 定时处理函数中重新开启定时器。Linux内核使用timer_list结构体表示内核定时器,timer_list 定义在文件 include/linux/timer.h 中,定义如下(省略掉条件编译):

示例代码 32.1.4 timer_list 结构体

```
struct timer_list {
    /*
    * All fields that change during normal runtime grouped to the
    * same cacheline
    */
    struct hlist_node entry;
    unsigned long expires;    /* 定时器超时时间,单位是节拍数 */
    void (*function)(struct timer_list *);    /* 定时处理函数 */
    u32 flags;
```



原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php

#ifdef CONFIG_LOCKDEP

struct lockdep_map lockdep_map;

#endif

};

要使用内核定时器首先要先定义一个 timer_list 变量,表示定时器, tiemr_list 结构体的 expires 成员变量表示超时时间,单位为节拍数。比如我们现在需要定义一个周期为 2 秒的定 时器,那么这个定时器的超时时间就是 jiffies+(2*HZ),因此 expires=jiffies+(2*HZ)。function 就是定时器超时以后的定时处理函数,我们要做的工作就放到这个函数里面,需要我们编写 这个定时处理函数。

定义好定时器以后还需要通过一系列的 API 函数来初始化此定时器,这些函数如下:

1、timer_setup 函数

timer_setup 函数负责初始化 timer_list 类型变量,当我们定义了一个 timer_list 变量以后 一定要先用 timer_setup 初始化一下。timer_setup 函数原型如下:

#define timer_setup(timer, callback, flags) __init_timer((timer), (callback), (flags))

函数参数和返回值含义如下:

timer: 要初始化定时器。

callback: 定时器回调函数,当定时器计数到设定的时间后,运行此函数。

flags: 表示 cpu id, 一般为 0。

返回值: 没有返回值。

2、add_timer 函数

add_timer 函数用于向 Linux 内核注册定时器,使用 add_timer 函数向内核注册定时器以后, 定时器就会开始运行,函数原型如下:

void add_timer(struct timer_list *timer)

函数参数和返回值含义如下: timer: 要注册的定时器。 返回值: 没有返回值。

3、del_timer 函数

del_timer 函数用于删除一个定时器,不管定时器有没有被激活,都可以使用此函数删除。 在多处理器系统上,定时器可能会在其他的处理器上运行,因此在调用 del_timer 函数删除定 时器之前要先等待其他处理器的定时处理器函数退出。del_timer 函数原型如下:

int del_timer(struct timer_list * timer)

函数参数和返回值含义如下:

timer: 要删除的定时器。

返回值: 0, 定时器还没被激活; 1, 定时器已经激活。

4、del_timer_sync 函数

del_timer_sync 函数是 del_timer 函数的同步版,会等待其他处理器使用完定时器再删除, del_timer_sync 不能使用在中断上下文中。del_timer_sync 函数原型如下所示:

int del_timer_sync(struct timer_list *timer)

函数参数和返回值含义如下:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

timer: 要删除的定时器。

返回值: 0, 定时器还没被激活: 1, 定时器已经激活。

5、mod_timer 函数

mod_timer 函数用于修改定时值,如果定时器还没有激活的话,mod_timer 函数会激活定 时器! 函数原型如下:

int mod_timer(struct timer_list *timer, unsigned long expires)

函数参数和返回值含义如下:

timer: 要修改超时时间(定时值)的定时器。

expires: 修改后的超时时间。

返回值: 0,调用 mod_timer 函数前定时器未被激活; 1,调用 mod_timer 函数前定时器已 被激活。

关于内核定时器常用的 API 函数就讲这些,内核定时器一般的使用流程如下所示:

```
示例代码 32.1.5 内核定时器使用方法演示
                      /* 定义定时器 */
1 struct timer_list timer;
2
3 /* 定时器回调函数 */
4 void function(unsigned long arg)
5 {
6 /*
7 * 定时器处理代码
8
    */
9
10 /* 如果需要定时器周期性运行的话就使用 mod timer
11 *函数重新设置超时值并且启动定时器。
12 */
13 mod_timer(&dev->timertest, jiffies + msecs_to_jiffies(2000));
14 }
15
16 /* 初始化函数 */
17 void init(void)
18 {
                                /* 初始化定时器 */
19 init_timer(&timer);
20
   timer.function = function;
                                     /* 设置定时处理函数 */
21
22
   timer.expires=iffies + msecs_to_jiffies(2000);/* 超时时间 2 秒 */
23
   timer.data = (unsigned long)&dev;
                                    /* 将设备结构体作为参数 */
24
25 add_timer(&timer);
                                     /* 启动定时器 */
26 }
27
28 /* 退出函数 */
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

```
29 void exit(void)
30 {
31 del_timer(&timer); /* 删除定时器 */
32 /* 或者使用 */
33 del_timer_sync(&timer);
34 }
```

32.1.3 Linux 内核短延时函数

有时候我们需要在内核中实现短延时,尤其是在 Linux 驱动中。Linux 内核提供了毫秒、 微秒和纳秒延时函数,这三个函数如表 32.1.3.1 所示:

	函数		描述
void	ndelay(unsigned	long	
nsecs)			
void	udelay(unsigned	long	油 新 曲新和宣称环时函数
usecs)			的抄、佩伊相笔抄延时困致。
void	mdelay(unsigned	long	
mseces)			

表 32.1.3.1 内核短延时函数

32.2 硬件原理图分析

本章使用通过设置一个定时器来实现周期性的闪烁 LED 灯,还是使用 PS_LED0 为例,关于 PS_LED0 的硬件原理图参考 22.3 小节即可。

32.3 实验程序编写

本实验对应的例程路径为:开发板资料盘(A 盘)\4_SourceCode\3_Embedded_Linux\Linux 驱动例程\12_timer。

本章实验我们使用内核定时器周期性的点亮和熄灭开发板上的 PS_LED0, LED 灯的闪烁 周期由内核定时器来设置,测试应用程序可以控制内核定时器周期。

32.3.1 修改设备树文件

本章实验使用到了 LED 灯, LED 灯的设备树节点信息使用 27.3.1 小节创建的即可。

32.3.2 定时器驱动程序编写

在 drivers 目录下新建名为 "12_timer"的文件夹,在 12_timer 目录下创建一个名为 timer.c 的源文件,在 timer.c 里面输入如下内容:

示例代码 32.3.1 timer.c 文件代码段

2 Copyright © ALIENTEK Co., Ltd. 1998-2029. All rights reserved.

```
3 文件名 : timer.c
```

```
4 作者 :邓涛
```

5 版本 : V1.0



```
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
            : linux 内核定时器测试
   6 描述
   7 其他
            :无
   8 论坛 : www.openedv.com
   9 日志
            :初版 V1.0 2019/1/30 邓涛创建
   11
   12 #include <linux/kernel.h>
   13 #include <linux/module.h>
   14 #include <linux/errno.h>
   15 #include <linux/gpio.h>
   16 #include <asm/uaccess.h>
   17 #include <linux/cdev.h>
   18 #include linux/of.h>
   19 #include <linux/of_gpio.h>
   20 #include <linux/timer.h>
   21 #include <linux/types.h>
   22
   23 #define LED_CNT 1 /* 设备号个数 */
   24 #define LED_NAME
                         "led" /* 名字 */
   25
   26 /* ioctl 函数命令定义 */
   27 #define CMD_LED_CLOSE (_IO(0XEF, 0x1)) /* 关闭 LED */
   28 #define CMD_LED_OPEN (_IO(0XEF, 0x2)) /* 打开 LED */
   29 #define CMD_SET_PERIOD (_IO(0XEF, 0x3)) /* 设置 LED 闪烁频率 */
   30
   31
   32 /* led 设备结构体 */
   33 struct led_dev {
                             /* 设备号 */
   34
        dev_t devid;
   35
       struct cdev cdev;
                             /* cdev */
       struct class *class;
                             /* 类 */
   36
   37
        struct device *device;
                            /* 设备 */
                             /* 主设备号 */
   38
        int major;
                             /* 次设备号 */
   39
        int minor;
        struct device_node *nd;
                            /* 设备节点 */
   40
   41
        int led_gpio;
                             /* GPIO 编号 */
                             /* 定时周期,单位为 ms */
   42
        int period;
                            /* 定义一个定时器 */
   43
        struct timer_list timer;
                             /* 定义自旋锁 */
   44
        spinlock_t spinlock;
   45 };
```

46



```
原子哥在线教学: www.yuanzige.com
                                          论坛:www.openedv.com/forum.php
   47 static struct led_dev led; /* led 设备 */
   48
   49 /*
   50 * @description
                         :打开设备
   51 * @param - inode : 传递给驱动的 inode
   52 * @param – filp : 设备文件, file 结构体有个叫做 private data 的成员变量
                         一般在 open 的时候将 private_data 指向设备结构体。
   53 *
   54 * @return
                        :0 成功;其他 失败
   55 */
   56 static int led_open(struct inode *inode, struct file *filp)
   57 {
        return 0;
   58
   59 }
   60
   61 /*
   62 * @description :从设备读取数据
   63 * @param - filp : 要打开的设备文件(文件描述符)
   64 * @param – buf : 返回给用户空间的数据缓冲区
   65 * @param - cnt : 要读取的数据长度
   66 * @param – offt : 相对于文件首地址的偏移
                   :读取的字节数,如果为负值,表示读取失败
   67 * @return
   68 */
   69 static ssize_t led_read(struct file *filp, char __user *buf,
   70
             size_t cnt, loff_t *offt)
   71 {
   72
         return 0;
   73 }
   74
   75 /*
   76 * @description : 向设备写数据

      77 * @param - filp
      : 设备文件,表示打开的文件描述符

      78 * @param - buf
      : 要写给设备写入的数据

      79 * @param - cnt
      :要写入的数据长度

      80 * @param - offt
      :相对于文件首地址的偏移

                          :写入的字节数,如果为负值,表示写入失败
   81 * @return
   82 */
   83 static ssize_t led_write(struct file *filp, const char __user *buf,
             size_t cnt, loff_t *offt)
   84
   85 {
        return 0;
    86
```

```
87 }
```



```
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
    88
   89 /*
   90 * @description
                       :关闭/释放设备
   91 * @param - filp : 要关闭的设备文件(文件描述符)
   92 * @return
                      :0 成功;其他 失败
   93 */
   94 static int led_release(struct inode *inode, struct file *filp)
   95 {
   96
        return 0;
   97 }
   98
   99 /*
    100 * @description
                      : ioctl 函数,
    101 * @param - filp : 要打开的设备文件(文件描述符)
    102 * @param - cmd : 应用程序发送过来的命令
    103 * @param – arg :参数
    104 * @return
                        :0 成功;其他 失败
    105 */
    106 static long led_unlocked_ioctl(struct file *filp, unsigned int cmd,
    107
             unsigned long arg)
    108 {
    109
         unsigned long flags;
    110
    111
         /* 自旋锁上锁 */
    112
         spin_lock_irqsave(&led.spinlock, flags);
    113
    114
         switch (cmd) {
    115
         case CMD_LED_CLOSE:
    116
    117
           del_timer_sync(&led.timer);
           gpio_set_value(led.led_gpio, 0);
    118
           break;
    119
    120
    121
         case CMD_LED_OPEN:
    122
           del_timer_sync(&led.timer);
    123
           gpio_set_value(led.led_gpio, 1);
    124
           break;
    125
    126
         case CMD_SET_PERIOD:
    127
           led.period = arg;
```



```
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
    129
            break;
    130
    131
         default: break;
    132
         }
    133
    134
         /* 自旋锁解锁 */
    135
         spin_unlock_irqrestore(&led.spinlock, flags);
    136
    137
         return 0;
    138
    139
    140 /* 设备操作函数 */
    141 static struct file_operations led_fops = {
                     = THIS_MODULE,
    142 .owner
    143 .open
                      = led_open,
    144 .read
                     = led_read,
    145 .write
                      = led_write,
    146 .release
                      = led_release,
    147
         .unlocked_ioctl = led_unlocked_ioctl,
    148 };
    149
    150 /* 定时器回调函数 */
    151 static void led_timer_function(struct timer_list *unused)
    152 {
    153
         static bool on = 1;
    154
         unsigned long flags;
    155
    156
         /* 每次都取反, 实现 LED 灯反转 */
    157
         on = !on;
    158
    159
         /* 自旋锁上锁 */
    160
         spin_lock_irqsave(&led.spinlock, flags);
    161
         /* 设置 GPIO 电平状态 */
    162
    163
         gpio_set_value(led.led_gpio, on);
    164
    165
         /* 重启定时器 */
    166
         mod_timer(&led.timer, jiffies + msecs_to_jiffies(led.period));
    167
    168
         /* 自旋锁解锁 */
    169
         spin_unlock_irqrestore(&led.spinlock, flags);
```



```
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
    170 }
    171
    172 static int __init led_init(void)
    173 {
    174
         const char *str;
    175
          int val;
    176
          int ret;
    177
    178
          /* 初始化自旋锁 */
    179
          spin_lock_init(&led.spinlock);
    180
         /* 1.获取 led 设备节点 */
    181
    182
          led.nd = of_find_node_by_path("/led");
         if(NULL == led.nd) {
    183
    184
            printk(KERN_ERR "led: Failed to get led node\n");
    185
            return -EINVAL;
    186
          }
    187
    188
          /* 2.读取 status 属性 */
    189
          ret = of_property_read_string(led.nd, "status", &str);
    190
          if(!ret) {
    191
            if (strcmp(str, "okay"))
    192
              return -EINVAL;
    193
          }
    194
    195
          /* 3.获取 compatible 属性值并进行匹配 */
    196
          ret = of_property_read_string(led.nd, "compatible", &str);
    197
          if(ret) {
    198
            printk(KERN_ERR "led: Failed to get compatible property\n");
    199
            return ret;
    200
          }
    201
    202
          if (strcmp(str, "alientek,led")) {
            printk(KERN_ERR "led: Compatible match failed\n");
    203
    204
            return -EINVAL;
    205
          }
    206
    207
          printk(KERN_INFO "led: device matching successful!\r\n");
    208
          /* 4.获取设备树中的 led-gpio 属性,得到 LED 所使用的 GPIO 编号 */
    209
    210
          led.led_gpio = of_get_named_gpio(led.nd, "led-gpio", 0);
```



```
原子哥在线教学: www.yuanzige.com
                                            论坛:www.openedv.com/forum.php
    211
          if(!gpio_is_valid(led.led_gpio)) {
    212
            printk(KERN_ERR "led: Failed to get led-gpio\n");
    213
            return -EINVAL;
    214
         }
    215
    216
          printk(KERN_INFO "led: led-gpio num = %d\r\n", led.led_gpio);
    217
    218
         /* 5.向 gpio 子系统申请使用 GPIO */
    219
         ret = gpio_request(led.led_gpio, "LED Gpio");
    220
         if (ret) {
    221
            printk(KERN_ERR "led: Failed to request led-gpio\n");
    222
            return ret;
    223
         }
    224
    225
         /* 6.设置 LED 灯初始状态 */
    226
         ret = of_property_read_string(led.nd, "default-state", &str);
    227
         if(!ret) {
    228
            if (!strcmp(str, "on"))
    229
              val = 1;
    230
            else
    231
              val = 0;
    232
          } else
    233
            val = 0;
    234
    235
          gpio_direction_output(led.led_gpio, val);
    236
    237
         /* 7.注册字符设备驱动 */
    238
         /* 创建设备号 */
    239
         if (led.major) {
    240
            led.devid = MKDEV(led.major, 0);
    241
            ret = register_chrdev_region(led.devid, LED_CNT, LED_NAME);
    242
            if (ret)
    243
              goto out1;
    244
          } else {
            ret = alloc_chrdev_region(&led.devid, 0, LED_CNT, LED_NAME);
    245
    246
            if (ret)
    247
              goto out1;
    248
    249
            led.major = MAJOR(led.devid);
    250
            led.minor = MINOR(led.devid);
    251
          }
```



```
原子哥在线教学: www.yuanzige.com
                                             论坛:www.openedv.com/forum.php
    252
   253
         printk(KERN_INFO "led: major=%d, minor=%d\r\n", led.major, led.minor);
    254
    255
         /* 初始化 cdev */
         led.cdev.owner = THIS_MODULE;
    256
    257
         cdev_init(&led.cdev, &led_fops);
    258
   259
          /* 添加 cdev */
    260
         ret = cdev_add(&led.cdev, led.devid, LED_CNT);
    261
         if (ret)
    262
            goto out2;
    263
          /* 创建类 */
   264
   265
         led.class = class_create(THIS_MODULE, LED_NAME);
    266
         if (IS_ERR(led.class)) {
    267
           ret = PTR_ERR(led.class);
    268
           goto out3;
    269
         }
    270
    271
          /* 创建设备 */
    272
         led.device = device_create(led.class, NULL,
    273
                led.devid, NULL, LED_NAME);
    274
         if (IS_ERR(led.device)) {
   275
           ret = PTR_ERR(led.device);
   276
           goto out4;
   277
         }
   278
         /* 8.初始化 timer,绑定定时器处理函数,此时还未设置周期,所以不会激活定时器 */
   279
   280
         timer_setup(&led.timer,led_timer_function,0);
    281
    282
         return 0;
    283
   284 out4:
   285
         class_destroy(led.class);
    286
   287 out3:
   288
         cdev_del(&led.cdev);
   289
   290 out2:
         unregister_chrdev_region(led.devid, LED_CNT);
    291
    292
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php 293 out1: 294 gpio_free(led.led_gpio); 295 296 return ret; 297 } 298 299 static void __exit led_exit(void) 300 { 301 /* 删除定时器 */ 302 del_timer_sync(&led.timer); 303 304 /* 注销设备 */ 305 device_destroy(led.class, led.devid); 306 307 /* 注销类 */ 308 class_destroy(led.class); 309 310 /* 删除 cdev */ 311 cdev_del(&led.cdev); 312 313 /* 注销设备号 */ 314 unregister_chrdev_region(led.devid, LED_CNT); 315 316 /* 释放 GPIO */ gpio_free(led.led_gpio); 317 318 } 319 320 /* 驱动模块入口和出口函数注册 */ 321 module_init(led_init); 322 module_exit(led_exit); 323 324 MODULE_AUTHOR("DengTao <773904075@qq.com>"); 325 MODULE_DESCRIPTION("Alientek Gpio LED Driver"); 326 MODULE_LICENSE("GPL"); 第 33~45 行, led 设备结构体, 在 43 行定义了一个定时器成员变量 timer; 在 44 行定义了

一个自旋锁变量,用于对必要的代码段进行保护。

第 106~138 行,函数 led_unlocked_ioctl,对应应用程序的 ioctl 函数,应用程序调用 ioctl 函数向驱动发送控制信息,此函数响应并执行。此函数有三个参数: filp, cmd 和 arg,其中 filp 是对应的设备文件, cmd 是应用程序发送过来的命令信息, arg 是应用程序发送过来的参数,在本章例程中 arg 参数表示定时周期。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

本 驱 动 成 需 一 共 定 义 了 三 种 命 令 : CMD_LED_CLOSE 、 CMD_LED_OPEN 和 CMD_SET_PERIOD,这三个命令分别为熄灭 LED 灯、点亮 LED 灯(常亮)、LED 灯闪烁。 这三个命令的作用如下:

CMD_LED_CLOSE: 熄灭 LED 灯,首先调用 del_timer_sync 函数关闭定时器,然后再将 LED 熄灭。

CMD_LED_OPEN: LED 灯常亮,首先也是调用 del_timer_sync 函数关闭定时器,然后 再将 LED 点亮。

CMD_SET_PERIOD: 让 LED 灯闪烁,参数 arg 就是闪烁周期,单位为毫秒(ms);从 应用层传递过来的,将 led 的 period 成员变量设置为 arg 所表示定时周期,然后使用 mod_timer 打开定时器,使定时器以新的周期运行。

在 led_unlocked_ioctl 函数中使用了自旋锁对代码段进行保护。

第 141~148 行, led 设备驱动操作函数集 led_fops, 在 led 的操作函数集中, led_read 和 led_write 函数在本驱动程序中都没被用到,因为应用程序使用了 ioctl 函数对设备进行控制,所以驱动要定义 led_unlocked_ioctl。

第 151~170 行,函数 led_timer_function,定时器服务函数,此函有一个参数 arg,在初始 化定时器的时候可以设置传递给 led_timer_function 函数的参数,不过在本例中我们没有用到。 当定时周期到了以后此函数就会被调用;第 157 行,每次进入定时器服务函数都会将变量取 反,实现 LED 灯闪烁的效果。因为内核定时器不是循环的定时器,执行一次以后就结束了, 因此在 166 行又调用了 mod_timer 函数重新开启定时器;同样在这个服务函数中也使用了自旋 锁进行保护!

第 172~298 行,函数 led_init,驱动入口函数。在 179 行初始化自旋锁;第 280 行,初始 化定时器,设置定时器的定时处理函数为 led_timer_function,在 led_init 函数中并没有调用 timer_add 函数来开启定时器,因此定时器默认是关闭的,除非应用程序发送打开命令。

第 299~318 行,驱动出口函数。第 302 行调用 del_timer_sync 函数删除定时器,也可以使用 del_timer 函数。

32.3.3 编写测试 APP

测试 APP 我们要实现的内容如下:

- 运行 APP 以后提示我们输入 LED 灯控制命令,输入 0 表示熄灭 LED、输入 1 表示点亮 LED,输入 2 表示让 LED 灯周期性闪烁,并且此时提示再次输入闪烁周期,单位为 毫秒。
- ② 输入3则表示退出测试 APP 程序。

好了搞清楚我们的逻辑、需求之后就可以开始编写测试程序了,在 12_timer 目录下新建 名为 timerApp.c 的文件,然后输入如下所示内容:

示例代码 32.3.2 timerApp.c 文件代码段

- 2 Copyright © ALIENTEK Co., Ltd. 1998-2029. All rights reserved.
- 3 文件名 : timerApp.c
- 4 作者 : 邓涛
- 5版本 : V1.0
- 6 描述 : linux 内核定时器测试程序



```
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
    7 其他
             :无
    8 使用方法 : ./timerApp /dev/led
    9 论坛
             : www.openedv.com
    10 日志
               :初版 V1.0 2019/1/30 邓涛创建
    12
    13 #include <stdio.h>
    14 #include <unistd.h>
    15 #include <sys/types.h>
    16 #include <sys/stat.h>
    17 #include <fcntl.h>
    18 #include <stdlib.h>
    19 #include <string.h>
    20 #include <sys/ioctl.h>
    21
    22 /* ioctl 命令 */
    23 #define CMD_LED_CLOSE
                               (_IO(0XEF, 0x1)) /* 关闭 LED */
                               (_IO(0XEF, 0x2)) /* 打开 LED */
    24 #define CMD_LED_OPEN
                               (_IO(0XEF, 0x3)) /* 设置 LED 闪烁频率 */
    25 #define CMD_SET_PERIOD
    26
    27 /*
    28 * @description
                          : main 主程序
    29 * @param – argc
                         : argv 数组元素个数
    30 * @param – argv
                          :具体参数
    31 * @return
                          :0 成功;其他 失败
    32 */
    33 int main(int argc, char *argv[])
    34 {
    35 int fd, ret;
    36
        unsigned int cmd;
        unsigned int period;
    37
    38
    39
        if(2 != argc) {
    40
          printf("Usage:\n"
    41
             "\t./timerApp /dev/led @ open LED device\n"
    42
          );
    43
          return -1;
    44
        }
    45
    46 /* 打开设备 */
    47 fd = open(argv[1], O_RDWR);
```



```
原子哥在线教学: www.yuanzige.com
                                         论坛:www.openedv.com/forum.php
    48
        if(0 > fd) \{
    49
          printf("ERROR: %s file open failed!\r\n", argv[1]);
    50
          return -1;
    51
        }
    52
        /* 通过命令控制 LED 设备 */
    53
    54
        for(;;){
    55
          printf("Input CMD:");
    56
          scanf("%d", &cmd);
    57
    58
    59
          switch (cmd) {
    60
    61
          case 0:
    62
            cmd = CMD_LED_CLOSE;
    63
            break;
    64
    65
          case 1:
    66
            cmd = CMD_LED_OPEN;
    67
            break;
    68
    69
          case 2:
    70
            cmd = CMD_SET_PERIOD;
    71
            printf("Input Timer Period:");
    72
            scanf("%d", &period);
    73
            break;
    74
    75
          case 3:
    76
            close(fd);
    77
            return 0;
    78
    79
          default: break;
    80
          }
    81
    82
          ioctl(fd, cmd, period);
    83
        }
    84 }
    第 23~25 行, ioctl 命令值, 这个命令值跟驱动中定义的是一样的。
```

第54~83行,在for循环中,首先让用户输入要测试的命令,例如输入0表示关闭LED,将cmd设置为CMD_LED_CLOSE;输入1表示打开LED灯,将cmd设置为CMD_LED_OPEN;输入2表示让LED灯周期性闪烁,让后再提示用户输入闪烁周期。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

ge.com 论坛:www.openedv.com/forum.php 第 82 行通过调用 :cott 函数发送 and 绘版动程序

上面的命令输入完成之后,第 82 行通过调用 ioctl 函数发送 cmd 给驱动程序,并且 ioctl 函数的 arg 参数就是用户输入的周期值(当用户输入命令为 2 时)。

测试完成之后用户可以输入3命令退出测试程序。

32.4 运行测试

32.4.1 编译驱动程序和测试 APP

1、编译驱动程序

编写 Makefile 文件,将第三十一章实验目录 11_key 下的 Makefile 文件拷贝到当前实验目 录下,打开 Makfile 文件,将 obj-m 变量的值改为 timer.o,修改完成之后 Makefile 内容如下所 示:

示例代码 32.4.1 Makefile 文件

1 KERN_DIR :=/home/zy/workspace/kernel-driver/linux-xlnx_rebase_v5.4_2020.2 2 3 obj-m :=timer.o 4 5 all: 6 make -C \$(KERN_DIR) M=`pwd` modules 7 clean: 8 make -C \$(KERN_DIR) M=`pwd` clean 第3行,设置 obj-m 变量的值为 timer.o。 修改完成之后保存退出,在实验目录下输入如下命令编译出驱动模块文件: make 编译成功以后就会生成一个名为"timer.ko"的驱动模块文件,如下所示: zy@zy-virtual-machine:~/linux/drivers/12 timer\$ ls Makefile timer.c zy@zy-virtual-machine:~/linux/drivers/12 timer\$ zy@zy-virtual-machine:~/linux/drivers/12 timer\$ make make -C /home/zy/workspace/kernel-driver/linux-xlnx-xlnx rebase v5.4 2020.2 M=`p wd` modules make[1]: 进入目录"/home/zy/workspace/kernel-driver/linux-xlnx-xlnx rebase v5.4 2 020.2" CC [M] /home/zy/linux/drivers/12 timer/timer.o Building modules, stage 2. MODPOST 1 modules CC [M] /home/zy/linux/drivers/12 timer/timer.mod.o LD [M] /home/zy/linux/drivers/12 timer/timer.ko make[1]: 离开目录"/home/zy/workspace/kernel-driver/linux-xlnx-xlnx rebase v5.4 2 020.2" zy@zy-virtual-machine:~/linux/drivers/12 timer\$ zy@zy-virtual-machine:~/linux/drivers/12 timer\$ ls Makefile Module.symvers timer.ko timer.mod.c timer.o modules.order timer.c timer.mod timer.mod.o zy@zy-virtual-machine:~/linux/drivers/12 timer\$



2、编译测试 APP



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

输入如下命令编译测试 timerApp.c 这个测试程序:

\$CC timerApp.c -o timerApp

编译成功以后就会生成 timerApp 这个应用程序。

32.4.2 运行测试

将上面编译出来的 timer.ko 和 timerApp 这两个文件拷贝到 NFS 共享目录下根文件系统的 /lib/modules/5.4.0-xilinx 文件夹中。

将 SD 启动卡插到开发板中,连接开发板串口、网口和电源线,设置拨码开关为 SD 卡启动模式,然后打开电源,启动开发板。

系统启动完成后,通过串口终端进入"/lib/modules/5.4.0-xilinx"目录下,输入如下命令 加载 key.ko 驱动模块:

depmod //第一次加载驱动的时候需要运行此命令

modprobe timer.ko //加载驱动

驱动加载成功以后如下命令来测试:

./timerApp /dev/led

输入上述命令以后终端提示输入命令,如所示:

root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx#
root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx# depmod
root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx# modprobe timer.ko
timer: loading out-of-tree module taints kernel.
led: device matching successful!
led: led-gpio num = 905
led: major=244, minor=0
root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx#
root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx# ./timerApp /dev/led
Input CMD:

图 32.4.2 运行 timerApp 测试程序

输入"0"回车,关闭LED;输入1回车,点亮LED;输入2回车之后,又会提示用户输入一个闪烁周期值,以毫秒为单位,操作如下所示:

root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx#	
root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx#	./timerApp /dev/led
Input CMD:0	
Input CMD:1	
Input CMD:2	
Input Timer Period:50	
Input CMD:	

图 32.4.3 输入相应数字执行命令

上面输入"50",表示设置定时器周期值为 50ms,设置好以后 LED 灯就会以 50ms 为间 隔,开始闪烁。测试完成之后我们可以输入 3 退出测试 APP。这里需要注意的是,我们的测 试程序代码中并没有对输入的内容做检测,所以如果输入了其它的字符可能会导致奇怪的现 象! 当然大家可以对测试程序代码进行完善。

通过下面的命令卸载驱动模块:

rmmod timer.ko



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

第三十三章 Linux 中断实验

不管是裸机实验还是 Linux 下的驱动实验,中断都是频繁使用的功能,关于 ZYNQ 的中 断原理已经在《领航者 ZYNQ 嵌入式开发指南》第四章中做了详细的讲解,在裸机中使用中 断我们需要做一大堆的工作,比如配置寄存器,使能 IRQ 等等(SDK 提供的库函数中已经封 装好了)。Linux 内核提供了完善的中断框架,我们只需要申请中断,然后注册中断处理函数 即可,使用非常方便,不需要一系列复杂的寄存器配置。本章我们就来学习一下如何在 Linux 下使用中断。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

33.1 Linux 中断简介

33.1.1 Linux 中断 API 函数

先来回顾一下裸机实验里面按键中断的处理方法:

- ①、使能中断异常。
- ② 、注册中断服务函数。
- ③、设置中断的触发方式。
- ④、最后使能按键中断。

当按键中断发生以后进入到前面绑定的 IRO 中断服务函数当中, 然后执行相应的中断处 理代码。在 Linux 内核中也提供了大量的中断相关的 API 函数,我们来看一下这些跟中断有 关的 API 函数:

1、中断号

每个中断都有一个中断号,通过中断号即可区分不同的中断,有的资料也把中断号叫做 中断线。在 Linux 内核中使用一个 int 变量表示中断号。

2、request irq 函数

在 Linux 内核中要想使用某个中断是需要申请的, request irg 函数用于申请中断, request irg 函数可能会导致睡眠,因此不能在中断上下文或者其他禁止睡眠的代码段中使用 request_irq 函数。request_irq 函数会激活(使能)中断,所以不需要我们手动去使能中断, request_irq 函数原型如下:

int request_irq(unsigned int		irq,
	irq_handler_t	handler
	unsigned long	flags,
	const char	*name,
	void	*dev)

函数参数和返回值含义如下:

irg: 要申请中断的中断号。

handler: 中断处理函数,当中断发生以后就会执行此中断处理函数。

flags: 中断标志,可以在内核源码目录 include/linux/interrupt.h 文件中查看所有的中断标 志,这里我们介绍几个常用的中断标志,如表 33.1.1.1 所示:

标志	描述
	多个设备共享一个中断线, 共享的所有中断都必
IRQF_SHARED	须指定此标志。如果使用共享中断的话,request_irq
	函数的 dev 参数就是唯一区分他们的标志。
IRQF_ONESHOT	单次中断,中断执行一次就结束。
IRQF_TRIGGER_N	于触发
ONE	
IRQF_TRIGGER_RI	上升沉舳岩
SING	
IRQF_TRIGGER_FA	下降沿鲉发
LLING	I F牛1日 / 五 / 入 。

原子哥在线教学: www.yuanzige	.com 论坛:www.openedv.com/forum.php
IRQF_TRIGGER_HI	高电平触发。
GH	
IRQF_TRIGGER_L OW	低电平触发。

表 33.1.1.1 常用的中断标志

正点原子

比如开发板上的 PS_KEY0 使用 GPIO0_IO12(也就是 MIO 12),按下 PS_KEY0 以后为低电平,因此可以设置为下降沿触发,也就是将 flags 设置为 IRQF_TRIGGER_FALLING。表 33.1.1.1 中的这些标志可以通过"|"来实现多种组合。

name: 中断名字,设置以后可以在/proc/interrupts文件中看到对应的中断名字。

dev: 如果将 flags 设置为 IRQF_SHARED 的话, dev 用来区分不同的中断, 一般情况下将 dev 设置为设备结构体, dev 会传递给中断处理函数 irg handler t 的第二个参数。

返回值: 0 中断申请成功,其他负值 中断申请失败,如果返回-EBUSY 的话表示中断已 经被申请了。

3、free_irq函数

使用中断的时候需要通过 request_irq 函数申请,使用完成以后就要通过 free_irq 函数释放 掉相应的中断。如果中断不是共享的,那么 free_irq 会删除中断处理函数并且禁止中断。 free_irq 函数原型如下所示:

void free_irq(unsigned int irq, void *dev)

函数参数和返回值含义如下:

irq: 要释放的中断。

dev:如果中断设置为共享(**IRQF_SHARED**)的话,此参数用来区分具体的中断。共享中断只有在释放最后中断处理函数的时候才会被禁止掉。

返回值:无。

4、中断处理函数

使用 request_irq 函数申请中断的时候需要设置中断处理函数,中断处理函数格式如下所示:

irqreturn_t (*irq_handler_t) (int, void *)

第一个参数是要中断处理函数要相应的中断号。第二个参数是一个指向 void 的指针,也就是个通用指针,需要与 request_irq 函数的 dev 参数保持一致。用于区分共享中断的不同设备,dev也可以指向设备数据结构。中断处理函数的返回值为irqreturn_t类型,irqreturn_t类型定义如下所示:

示例代码 33.1.1 irqreturn_t 结构体

```
10 enum irqreturn {11IRQ_NONE
```

```
11 IRQ_NONE = (0 << 0),
12 IRQ_HANDLED = (1 << 0),
```

```
13 IRQ_WAKE_THREAD = (1 \ll 1),
```

14 };

15

16 **typedef** enum irqreturn irqreturn_t;

可以看出irqreturn_t 是个枚举类型,一共有三种返回值。一般中断服务函数返回值使用如下形式:


论坛:www.openedv.com/forum.php

return IRQ_RETVAL(IRQ_HANDLED)

原子哥在线教学: www.yuanzige.com

5、中断使能与禁止函数

常用的中断使用和禁止函数如下所示:

void enable_irq(unsigned int irq)

void disable_irq(unsigned int irq)

enable_irq 和 disable_irq 用于使能和禁止指定的中断, irq 就是要禁止的中断号。 disable_irq 函数要等到当前正在执行的中断处理函数执行完才返回,因此使用者需要保证不会 产生新的中断,并且确保所有已经开始执行的中断处理程序已经全部退出。在这种情况下, 可以使用另外一个中断禁止函数:

void disable_irq_nosync(unsigned int irq)

disable_irq_nosync 函数调用以后立即返回,不会等待当前中断处理程序执行完毕。上面 三个函数都是使能或者禁止某一个中断,有时候我们需要关闭当前处理器的整个中断系统, 也就是在学习 STM32 的时候常说的关闭全局中断,这个时候可以使用如下两个函数:

local_irq_enable()

local_irq_disable()

local_irq_enable 用于使能当前处理器中断系统, local_irq_disable 用于禁止当前处理器中断系统。假如 A 任务调用 local_irq_disable 关闭全局中断 10S, 当关闭了 2S 的时候 B 任务开始运行, B 任务也调用 local_irq_disable 关闭全局中断 3S, 3 秒以后 B 任务调用 local_irq_enable 函数将全局中断打开了。此时才过去 2+3=5 秒的时间, 然后全局中断就被打开了, 此时 A 任务要关闭 10S 全局中断的愿望就破灭了, 然后 A 任务就"生气了",结果很严重,可能系统都要被 A 任务整崩溃。为了解决这个问题, B 任务不能直接简单粗暴的通过 local_irq_enable 函数来打开全局中断, 而是将中断状态恢复到以前的状态, 要考虑到别的任务的感受, 此时就要用到下面两个函数:

local_irq_save(flags)

local_irq_restore(flags)

这两个函数是一对,local_irq_save 函数用于禁止中断,并且将中断状态保存在 flags 中。 local_irq_restore 用于恢复中断,将中断到 flags 状态。

33.1.2 上半部与下半部

在有些资料中也将上半部和下半部称为顶半部和底半部,都是一个意思。我们在使用 request_irq申请中断的时候注册的中断服务函数属于中断处理的上半部,只要中断触发,那么 中断处理函数就会执行。我们都知道中断处理函数一定要快点执行完毕,越短越好,但是现 实往往是残酷的,有些中断处理过程就是比较费时间,我们必须要对其进行处理,缩小中断 处理函数的执行时间。比如电容触摸屏通过中断通知 SOC 有触摸事件发生,SOC 响应中断, 然后通过 IIC 接口读取触摸坐标值并将其上报给系统。但是我们都知道 IIC 的速度最高也只有 400Kbit/S,所以在中断中通过 IIC 读取数据就会浪费时间。我们可以将通过 IIC 读取触摸数据 的操作暂后执行,中断处理函数仅仅相应中断,然后清除中断标志位即可。这个时候中断处 理过程就分为了两部分:

上半部:上半部就是中断处理函数,那些处理过程比较快,不会占用很长时间的处理就可以放在上半部完成。



原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php

下半部:如果中断处理过程比较耗时,那么就将这些比较耗时的代码提出来,交给下半 部去执行,这样中断处理函数就会快进快出。

因此, Linux 内核将中断分为上半部和下半部的主要目的就是实现中断处理函数的快进快 出,那些对时间敏感、执行速度快的操作可以放到中断处理函数中,也就是上半部。剩下的 所有工作都可以放到下半部去执行,比如在上半部将数据拷贝到内存中,关于数据的具体处 理就可以放到下半部去执行。至于哪些代码属于上半部,哪些代码属于下半部并没有明确的 规定,一切根据实际使用情况去判断,这个就很考验驱动编写人员的功底了。这里有一些可 以借鉴的参考点:

①、如果要处理的内容不希望被其他中断打断,那么可以放到上半部。

②、如果要处理的任务对时间敏感,可以放到上半部。

③、如果要处理的任务与硬件有关,可以放到上半部

④、除了上述三点以外的其他任务,优先考虑放到下半部。

上半部处理很简单,直接编写中断处理函数就行了,关键是下半部该怎么做呢? Linux 内 核提供了多种下半部机制,接下来我们来学习一下这些下半部机制。

1、软中断

一开始 Linux 内核提供了"bottom half"机制来实现下半部,简称"BH"。后面引入了 软中断和 tasklet 来替代 "BH" 机制,完全可以使用软中断和 tasklet 来替代 BH,从 2.5 版本的 Linux 内核开始 BH 已经被抛弃了。Linux 内核使用结构体 softirq_action 表示软中断, softirg action 结构体定义在文件 include/linux/interrupt.h 中,内容如下:

示例代码 33.1.2 softing action 结构体

```
433 struct softirg action
434 {
435 void (*action)(struct softirq_action *);
436 };
在 kernel/softirg.c 文件中一共定义了 10 个软中断,如下所示:
                            示例代码 33.1.3 softirq_vec 数组-1
056 static struct softirq_action softirq_vec[NR_SOFTIRQS];
NR_SOFTIRQS 是枚举类型,定义在文件 include/linux/interrupt.h 中,定义如下:
                            示例代码 33.1.4 softirg vec 数组-2
458 enum
459 {
460 HI_SOFTIRQ=0,
```

```
461 TIMER SOFTIRO,
```

```
462 NET_TX_SOFTIRQ,
```

```
463
    NET_RX_SOFTIRQ,
```

```
464
    BLOCK_SOFTIRQ,
```

```
IRQ POLL SOFTIRQ,
465
```

```
TASKLET_SOFTIRQ,
466
```

```
SCHED_SOFTIRQ,
467
```

```
468
      HRTIMER_SOFTIRQ, /* Unused, but kept as tools rely on the
```

numbering. Sigh! */ 469



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

RCU_SOFTIRQ, /* Preferable RCU should always be the last softirq */

470 471

472 NR_SOFTIRQS

473 };

可以看出,一共有 10 个软中断,因此 NR_SOFTIRQS 为 10,因此数组 softirq_vec 有 10 个元素。softirq_action 结构体中的 action 成员变量就是软中断的服务函数,数组 softirq_vec 是 个全局数组,因此所有的 CPU(对于 SMP 系统而言)都可以访问到,每个 CPU 都有自己的触发 和控制机制,并且只执行自己所触发的软中断。但是各个 CPU 所执行的软中断服务函数确是 相同的,都是数组 softirq_vec 中定义的 action 函数。要使用软中断,必须先使用 open_softirq 函数注册对应的软中断处理函数, open_softirq 函数原型如下:

void open_softirq(int nr, void (*action)(struct softirq_action *))

函数参数和返回值含义如下:

nr: 要开启的软中断, 在示例代码 33.1.4 中选择一个。

action: 软中断对应的处理函数。

返回值:没有返回值。

注册好软中断以后需要通过 raise_softirq 函数触发, raise_softirq 函数原型如下:

void raise_softirq(unsigned int nr)

函数参数和返回值含义如下:

nr: 要触发的软中断, 在示例代码 33.1.4 中选择一个。

返回值:没有返回值。

软中断必须在编译的时候静态注册! Linux 内核使用 softirq_init 函数初始化软中断, softirq_init 函数定义在 kernel/softirq.c 文件里面,函数内容如下:

示例代码 33.1.5 softirq_init 函数内容

```
634 void __init softirq_init(void)
635 {
636 int cpu;
637
638 for_each_possible_cpu(cpu) {
639 per_cpu(tasklet_vec, cpu).tail =
640 & &per_cpu(tasklet_vec, cpu).head;
```

```
641 per_cpu(tasklet_hi_vec, cpu).tail =
```

```
642 &per_cpu(tasklet_hi_vec, cpu).head;
```

- 643 }
- 644

645 open_softirq(TASKLET_SOFTIRQ, tasklet_action);

646 open_softirq(HI_SOFTIRQ, tasklet_hi_action);

647 }

从示例代码 33.1.5 可以看出, softirq_init 函数默认会打开 TASKLET_SOFTIRQ 和 HI_SOFTIRQ。

2 tasklet



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

tasklet 是利用软中断来实现的另外一种下半部机制,在软中断和 tasklet 之间,建议大家 使用 tasklet。Linux 内核使用结构体

示例代码 33.1.6 tasklet_struct 结构体

484 s	truct tasklet_struct			
485 {				
486	<pre>struct tasklet_struct *next;</pre>	/* 下一个 tasklet */		
487	unsigned long state;	/* tasklet 状态 */		
488	atomic_t count;	/* 计数器,记录对 tasklet	t 的引用数 */	
489	<pre>void (*func)(unsigned long);</pre>	/* tasklet 执行的函数 */		
490	unsigned long data;	/* 函数 func 的参数 */		
491 }	;			
- 44	100 仨的 fame 函粉 計目 teel	1.4 再抽 行的 肋 理 丞 粉	田白空立丞粉山宓	相坐工由账户

第 489 行的 func 函数就是 tasklet 要执行的处理函数,用户定义函数内容,相当于中断处 理函数。如果要使用 tasklet,必须先定义一个 tasklet,然后使用 tasklet_init 函数初始化 tasklet, taskled_init 函数原型如下:

void tasklet_init(struct tasklet_struct *t, void (*func)(unsigned long), unsigned long data);

函数参数和返回值含义如下:

t: 要初始化的 tasklet

func: tasklet 的处理函数。

data: 要传递给 func 函数的参数

返回值:没有返回值。

也可以使用宏 DECLARE_TASKLET 来一次性完成 tasklet 的定义和初始化, DECLARE_TASKLET 定义在 include/linux/interrupt.h 文件中, 定义如下:

DECLARE_TASKLET(name, func, data)

其中 name 为要定义的 tasklet 名字,这个名字就是一个 tasklet_struct 类型的时候变量, func 就是 tasklet 的处理函数, data 是传递给 func 函数的参数。

在上半部,也就是中断处理函数中调用 tasklet_schedule 函数就能使 tasklet 在合适的时间运行,tasklet_schedule 函数原型如下:

void tasklet_schedule(struct tasklet_struct *t)

函数参数和返回值含义如下:

t: 要调度的 tasklet,也就是 DECLARE_TASKLET 宏里面的 name。

返回值:没有返回值。

关于 tasklet 的参考使用示例如下所示:

示例代码 33.1.7 tasklet 使用示例

/* 定义 taselet */

struct tasklet_struct testtasklet;

```
/* tasklet 处理函数 */
void testtasklet_func(unsigned long data)
{
    /* tasklet 具体处理内容 */
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

```
/* 中断处理函数 */
irqreturn_t test_handler(int irq, void *dev_id)
{
    .....
    /* 调度 tasklet */
    tasklet_schedule(&testtasklet);
    .....
}
/* 驱动入口函数 */
static int __init xxxx_init(void)
{
    .....
    /* 初始化 tasklet */
    tasklet_init(&testtasklet, testtasklet_func, data);
    /* 注册中断处理函数 */
```

```
.....
```

2、工作队列

工作队列是另外一种下半部执行方式,工作队列在进程上下文执行,工作队列将要推后 的工作交给一个内核线程去执行,因为工作队列工作在进程上下文,因此工作队列允许睡眠 或重新调度。因此如果你要推后的工作可以睡眠那么就可以选择工作队列,否则的话就只能 选择软中断或 tasklet。

Linux 内核使用 work_struct 结构体表示一个工作,该结构体定义在内核源码目录 include/linux/workqueue.h 头文件中,内容如下:

示例代码 33.1.8 work_struct 结构体

```
101 struct work_struct {
102 atomic_long_t data;
```

103 struct list_head entry;

```
104 work_func_t func; /* 工作处理函数 */
```

request_irq(xxx_irq, test_handler, 0, "xxx", &xxx_dev);

105 #ifdef CONFIG_LOCKDEP

```
106 struct lockdep_map lockdep_map;
```

107 #endif

108 };

这些工作组织成工作队列,工作队列使用 workqueue_struct 结构体表示,该结构体定义在内核源码目录 kernel/workqueue.c 文件中,内容如下:

示例代码 33.1.9 workqueue_struct 结构体

239 struct workqueue_struct {

240 struct list_head pwqs; /* WR: all pwqs of this wq */



下点原子



原子哥在线教学:www.yuanzige.com

论坛:www.openedv.com/forum.php

Linux 内核使用工作者线程(worker thred)来处理工作队列中的各个工作, Linux 内核使用 worker 结构体表示工作者线程, 该结构体定义在内核源码目录 kernel/workqueue_internal.h 头 文件中, worker 结构体内容如下:

示例代码 33.1.10 worker 结构体

```
239 struct workqueue_struct {
240
      struct list head pwqs;
                                 /* WR: all pwgs of this wg */
241
      struct list_head list;
                                 /* PR: list of all workqueues */
242
      struct mutex mutex;
243
                                 /* protects this wg */
                                 /* WQ: current work color */
244
      int work color;
                                 /* WQ: current flush color */
245
      int flush_color;
246
                                      /* flush in progress */
      atomic_t nr_pwqs_to_flush;
247
      struct wq_flusher *first_flusher;
                                           /* WQ: first flusher */
      struct list_head flusher_queue;
                                           /* WQ: flush waiters */
248
      struct list_head flusher_overflow; /* WQ: flush overflow list */
249
024 struct worker {
025
      /* on idle list while idle, on busy hash table while busy */
026
      union {
027
         struct list head entry;
                                      /* L: while idle */
028
         struct hlist_node hentry; /* L: while busy */
029
      };
030
031
      struct work_struct *current_work; /* L: work being processed */
032
      work_func_t current_func;
                                           /* L: current work's fn */
033
      struct pool_workqueue *current_pwq; /* L: current_work's pwq */
                                 /* ->desc is valid */
034
      bool desc_valid;
      struct list_head scheduled; /* L: scheduled works */
035
036
037
      /* 64 bytes boundary on 64bit, 32 on 32bit */
038
      struct task_struct *task;
                                      /* I: worker task */
039
040
      struct worker_pool *pool;
                                      /* I: the associated pool */
041
                        /* L: for rescuers */
042
                                      /* A: anchored at pool->workers */
      struct list_head node;
043
                             /* A: runs through worker->node */
044
045
      unsigned long last_active;
                                       /* L: last active timestamp */
046
      unsigned int flags;
                                       /* X: flags */
047
      int id;
                                       /* I: worker id */
```

原子哥	「在线教学: www.yuanzige.com	论坛:www.openedv.com/forum.php	
048	8		
049	9 /*		
050	• * Opaque string set with work_set_desc	c(). Printed out with task	
05	* dump for debugging - WARN, BUG,	panic or sysrq.	
052	2 */		
053	3 char desc[WORKER_DESC_LEN];		
054	4		
055	5 $/*$ used only by rescuers to point to the ta	arget workqueue */	
050	6 struct workqueue_struct *rescue_wq;/*	I: the workqueue to rescue */	
057	7 };		

正点原子

从示例代码 33.1.10 可以看出,每个 worker 都有一个工作队列,工作者线程处理自己工作队列中的所有工作。在实际的驱动开发中,我们只需要定义工作(work_struct)即可,关于工作队列和工作者线程我们基本不用去管。简单创建工作很简单,直接定义一个 work_struct 结构体变量即可,然后使用 INIT_WORK 宏来初始化工作, INIT_WORK 宏定义如下:

#define INIT_WORK(_work, _func)

_work 表示要初始化的工作, _func 是工作对应的处理函数。

也可以使用 DECLARE_WORK 宏一次性完成工作的创建和初始化, 宏定义如下:

#define DECLARE_WORK(n, f)

n表示定义的工作(work_struct),f表示工作对应的处理函数。

和 tasklet 一样,工作也是需要调度才能运行的,工作的调度函数为 schedule_work,函数 原型如下所示:

```
bool schedule_work(struct work_struct *work)
```

函数参数和返回值含义如下:

work: 要调度的工作。

返回值:0成功,其他值失败。

关于工作队列的参考使用示例如下所示:

示例代码 33.1.11 工作队列使用示例

```
/* 定义工作(work) */
```

struct work_struct testwork;

```
/* work 处理函数 */
```

void testwork_func_t(struct work_struct *work);

```
{
    /* work 具体处理内容 */
}
```

```
/* 中断处理函数 */
```

irqreturn_t test_handler(int irq, void *dev_id)

```
{
```

.....

```
/* 调度 work */
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

```
schedule_work(&testwork);
......
}
/* 驱动入口函数 */
static int __init xxxx_init(void)
{
.....
/* 初始化 work */
INIT_WORK(&testwork, testwork_func_t);
/* 注册中断处理函数 */
request_irq(xxx_irq, test_handler, 0, "xxx", &xxx_dev);
......
}
```

33.1.3 设备树中断信息节点

如果使用设备树的话就需要在设备树中设置好中断属性信息,Linux内核通过读取设备树中的中断属性信息来配置中断。对于中断控制器而言,设备树绑定信息参考文档 Documentation/devicetree/bindings/interrupt-controller/arm,gic.yam。打开 arch/arm/boot/dts/zynq-7000.dtsi文件,其中的 intc 节点就是 ZYNQ 7010/7020 的中断控制器节点,节点内容如下所示:示例代码 33.1.12 中断控制器 intc 节点

```
177 intc: interrupt-controller@f8f01000 {
```

```
178 compatible = "arm,cortex-a9-gic";
```

```
179 #interrupt-cells = \langle 3 \rangle;
```

```
180 interrupt-controller;
```

```
181 reg = <0xF8F01000 0x1000>,
```

```
182 <0xF8F00100 0x100>;
```

183 };

第 178 行, compatible 属性值为"arm,cortex-a9-gic"在 Linux 内核源码中搜索 "arm,cortex-a9-gic"即可找到 GIC 中断控制器驱动文件。

第 179 行, #interrupt-cells 和#address-cells、#size-cells 一样。表示此中断控制器下设备的 cells 大小,对于设备而言,会使用 interrupts 属性描述中断信息,#interrupt-cells 描述了 interrupts 属性的 cells 大小,也就是一条信息有几个 cells。每个 cells 都是 32 位整形值,对于 ARM 处理的 GIC 来说,一共有 3 个 cells,这三个 cells 的含义如下:

第一个 cells: 中断类型, 0 表示 SPI 中断(共享外设中断), 1 表示 PPI 中断(私有外设中断)。

第二个 cells: 中断号,在《领航者 ZYNQ 之嵌入式开发指南》第四章中介绍过, SPI 中断大约有 60 个,它们对应的中断号范围为 32~92(61 个),对于 PPI 中断来说中断号的范围为 27~31。但是该 cell 描述的中断号是从 0 开始。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

第三个 cells:标志,bit[3:0]表示中断触发类型,为1的时候表示上升沿触发,为2的时候表示下降沿触发,为4的时候表示高电平触发,为8的时候表示低电平触发。bit[15:8]为PPI中断的 CPU 掩码。

第180行,使用 interrupt-controller 属性来表示当前节点是中断控制器节点。

对于 gpio 来说, gpio 节点也可以作为中断控制器,比如 zynq-7000.dtsi 文件中的 gpio0 节 点内容如下所示:

示例代码 33.1.13 gpio0 设备节点

143 gpio0: gpio@e000a000 {

144 compatible = "xlnx,zynq-gpio-1.0";

```
145 #gpio-cells = <2>;
```

- 146 clocks = <&clkc 42>;
- 147 gpio-controller;
- 148 interrupt-controller;
- 149 #interrupt-cells = <2>;
- 150 interrupt-parent = <&intc>;
- 151 interrupts = <0 20 4>;
- 152 $reg = \langle 0xe000a000 \ 0x1000 \rangle;$

```
153 };
```

第151行,使用 interrupts 属性来描述 gpio 中断源的信息,"interrupts = <0 20 4>"中,0 表示中断类型,也就是前面说到的 SPI 中断;20 对应的就是 ZYNQ 7010/7020 GPIO 外设所对 应的中断号;而4表示中断触发类型是高电平触发。打开 ZYNQ 7010/7020 的参考手册,路径 为:领航者 ZYNQ 开发板资料盘(A 盘)\8_ZYNQ&FPGA 参考资料\Xilinx\User Guide\ug585-Zynq-7000-TRM.pdf,找到"Ch.7: Interrupts"章节,找到 Table 7-4,如下图所示:



原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php

Table 7-4: PS and PL Shared Peripheral Interrupts (SPI)

Source	Interrupt Name	IRQ ID#	Status Bits (mpcore Registers)	Required Type	PS-PL Signal Name	ı/o
	CPU 1, 0 (L1, TLB, BTAC)	33:32	spi_status_0[1:0]	Rising edge	~	~
APU	L2 Cache	34	spi_status_0[2]	High level	~	~
	ОСМ	35	spi_status_0[3]	High level	~	~
Reserved	~	36	spi_status_0[3]	~	~	~
PMU	PMU [1,0]	38, 37	spi_status_0[6:5]	High level	~	~
XADC	XADC	39	spi_status_0[7]	High level	~	~
DevC	DevC	40	spi_status_0[8]	High level	~	~
SWDT	SWDT	41	spi_status_0[9]	Rising edge	~	~
Timer	TTC 0	44:42	spi_status_0[12:10]	High level	~	~
DMAC	DMAC Abort	45	spi_status_0[13]	High level	IRQP2F[28]	Output
DIVIAC	DMAC [3:0]	49:46	spi_status_0[17:14]	High level	IRQP2F[23:20]	Output
Momony	SMC	50	spi_status_0[18]	High level	IRQP2F[19]	Output
wemory	Quad SPI	51	spi_status_0[19]	High level	IRQP2F[18]	Output
Reserved	~	2	~	Always driven Low	IRQP2F[17]	Output
	GPIO	52	spi_status_0[20]	High level	IRQP2F[16]	Output
	USB 0	53	spi_status_0[21]	High level	IRQP2F[15]	Output
	Ethernet 0	54	spi_status_0[22]	High level	IRQP2F[14]	Output
	Ethernet 0 Wake-up	55	spi_status_0[23]	Rising edge	IRQP2F[13]	Output

图 33.1.1 中断号

从图 33.1.1 中可以看出, GPIO 外设所对应的中断号为 52, 那么前面的 20 是怎么计算出 来的呢? 其实就是将中断号编号从 0 开始,将 52 减去 SPI 类型中断的起始编号 32 即可得出 20。

第 148 行, interrupt-controller 属性表明了 gpio0 节点也是个中断控制器,用于控制 ZYNQ 7010/7020 的所有 IO 的中断。第 147 行, gpio-controller 属性表明该节点是 gpio 中断控制器。

第149行,将#interrupt-cells修改为2,所以在设备树中使用gpio中断需要提供2个参数,第一个表示 GPIO 编号,第二个表示该 GPIO 的中断触发类型。

在设备树中如何表示一个 GPIO 中断呢? 我们来看看下面这个示例:

示例代码 33.1.14 gpio 中断使用示例

```
1 test-node {
```

```
2 compatible = "test-node";
```

3 reg = <0x1e>;

```
4 interrupt-parent = \langle \& gpio0 \rangle;
```

5 interrupts = <12 2>;

<mark>6</mark>};

第4行, interrupt-parent 属性设置中断控制器, 这里使用 gpio0 作为中断控制器。

第 5 行, interrupts 设置中断信息, 12 表示一个具体的 GPIO 引脚编号,也就是对应 GPIO0_12; 而 2 表示该 GPIO 的中断触发类型为下降沿触发。

简单总结一下与中断有关的设备树属性信息:

1、#interrupt-cells,指定中断源的信息 cells(参数)个数。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

2、interrupt-controller,表示当前节点为中断控制器。

3、interrupts,指定中断号,触发方式等。

4、interrupt-parent,指定父中断,也就是中断控制器。

33.1.4 获取中断号

编写驱动的时候需要用到中断号,我们用到中断号,中断信息已经写到了设备树里面,因此可以通过 irq_of_parse_and_map 函数从 interupts 属性中提取到对应的设备号,函数原型如下:

unsigned int irq_of_parse_and_map(struct device_node *dev, int index)

函数参数和返回值含义如下:

dev: 设备节点。

index: 索引号, interrupts 属性可能包含多条中断信息, 通过 index 指定要获取的信息。 返回值: 中断号。

如果使用 GPIO 的话,可以使用 gpio_to_irq 函数来获取 gpio 对应的中断号,函数原型如

下:

int gpio_to_irq(unsigned int gpio)
函数参数和返回值含义如下:
gpio: 要获取的 GPIO 编号。
返回值: GPIO 对应的中断号。

33.2 硬件原理图分析

本章实验硬件原理图参考 31.2 小节即可。

33.3 实验程序编写

本实验对应的例程路径为:开发板资料盘(A 盘)\4_SourceCode\3_Embedded_Linux\Linux 驱动例程\13_irq。

本章实验我们驱动开发板上的 PS_KEY0 按键,不过我们采用中断的方式,并且采用定时器来实现按键消抖,应用程序读取按键值并且通过终端打印出来。通过本章我们可以学习到 Linux 内核中断的使用方法,以及对 Linux 内核定时器的回顾。

33.3.1 修改设备树文件

将 31.3.1 小节中的 system-user.dtsi 文件复制到 13_irq 文件夹中。本章实验使用到了按键 PS_KEY0,按键 PS_KEY0 使用中断模式,因此需要在 system-user.dtsi 文件中 "key" 子节点 下添加中断相关属性,添加完成以后的 "key" 节点内容如下所示:

示例代码 33.3.1 key 节点信息

```
    #define GPIO_ACTIVE_HIGH
    #define GPIO_ACTIVE_LOW
    4 ///include/ "system-conf.dtsi"
```

5 #include <dt-bindings/gpio/gpio.h>



领航者 ZYNQ 之嵌入式 Linux 开发指南 原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php 6 #include <dt-bindings/input/input.h> 7 #include <dt-bindings/media/xilinx-vip.h> 8 #include <dt-bindings/phy/phy.h> 9 #include <dt-bindings/interrupt-controller/irq.h> 10 11/{ 12 model = "Alientek Navigator Zynq Development Board"; 13 compatible = "xlnx,zynq-zc702", "xlnx,zynq-7000"; ... 35 key { 36 compatible = "alientek,key"; status = "okay"; 37 key-gpio = <&gpio0 12 GPIO_ACTIVE_LOW>; 38 39 40 interrupt-parent=<&gpio0>; 41 interrupts=<12 IRQ_TYPE_EDGE_BOTH>; 42 }; 43 }; 第9行,使用 include 包含了一个头文件,在内核源码目录 include/dt-bindings/interrupt-

controller/irq.h 中。 第 40 行,设置 interrupt-parent 属性值为 "gpio0",因为 PS_KEY0 所使用的 GPIO 为 GPIO0_IO12,也就是设置 PS_KEY0 的 GPIO 中断控制器为 gpio0。

第41行,设置 interrupts 属性,也就是设置中断源,第一个 cells 的 12 表示 GPIO0 组的 12 号 IO。IRQ_TYPE_EDGE_BOTH 定义在头文件 include/dt-bindings/interrupt-controller/irq.h 中, 定义如下:

示例代码 33.3.2 irq.h 头文件内容

13 #define IRQ_TYPE_NONE 0 14 #define IRQ_TYPE_EDGE_RISING 1 15 #define IRQ_TYPE_EDGE_FALLING 2 16 #define IRQ_TYPE_EDGE_BOTH (IRQ_TYPE_EDGE_FALLING | IRQ_TYPE_EDGE_RISING) 17 #define IRQ_TYPE_LEVEL_HIGH 4

18 #define IRQ_TYPE_LEVEL_LOW 8

从示例代码 33.3.2 中可以看出, IRQ TYPE EDGE BOTH 表示上升沿和下降沿同时有效, 相当于 PS KEY0 按下和释放都会触发中断。

设备树编写完成以后使用下面这条命令重新编译设备树,如下所示: make dtbs



原子哥在线教学: www.yuanzige.com 论均

论坛:www.openedv.com/forum.php



图 33.3.1 重新编译设备树

将编译的到的 system-top.dtb 文件重命名为 system.dtb,用这个文件替换 SD 启动卡 FAT 分 区中的 dtb 文件,然后重新启动开发板。

33.3.2 按键驱动程序编写

在 drivers 目录下新建名为"13_irq"的文件夹, 然后在 13_irq 目录中新建 keyirq.c 源文件, 在 keyirq.c 里面输入如下内容:

```
示例代码 33.3.3 keyirq.c 文件代码
2 Copyright © ALIENTEK Co., Ltd. 1998-2029. All rights reserved.
3 文件名 : keyirq.c
4 作者
       :邓涛
5 版本
       : V1.0
6 描述
       :GPIO 按键中断驱动实验
7 其他 :无
8 论坛
       : www.openedv.com
9 日志
       :初版 V1.0 2019/1/30 邓涛创建
11
12 #include <linux/types.h>
13 #include <linux/kernel.h>
14 #include <linux/delay.h>
15 #include <linux/ide.h>
16 #include <linux/init.h>
17 #include <linux/module.h>
18 #include <linux/errno.h>
19 #include <linux/gpio.h>
```



```
原子哥在线教学: www.yuanzige.com
                                          论坛:www.openedv.com/forum.php
    20 #include <asm/mach/map.h>
    21 #include <asm/uaccess.h>
    22 #include <asm/io.h>
    23 #include <linux/cdev.h>
    24 #include <linux/of.h>
    25 #include <linux/of address.h>
    26 #include <linux/of_gpio.h>
    27 #include <linux/of_irq.h>
    28 #include <linux/irq.h>
    29
                                  /* 设备号个数 */
    30 #define KEY_CNT
                           1
                                  /* 名字 */
    31 #define KEY_NAME
                           "key"
    32
    33 /* 定义按键状态 */
    34 enum key_status {
                          // 按键按下
    35 KEY_PRESS = 0,
                           // 按键松开
    36 KEY_RELEASE,
    37 KEY_KEEP,
                           // 按键状态保持
    38 };
    39
    40 /* 按键设备结构体 */
    41 struct key_dev {
    42 dev_t devid;
                          /* 设备号 */
    43 struct cdev cdev;
                         /* cdev 结构体 */
    44 struct class *class;
                          /* 类 */
    45 struct device *device; /* 设备 */
    46 int key_gpio;
                          /* GPIO 编号 */
                          /* 中断号 */
    47 int irq_num;
    48 struct timer_list timer; /* 定时器 */
    49
        spinlock_t spinlock; /* 自旋锁 */
    50 };
    51
    52 static struct key_dev key;
                                  /* 按键设备 */
    53 static int status = KEY_KEEP;
                                  /* 按键状态 */
    54
    55 /*
    56 * @description
                           :打开设备
    57 * @param – inode
                           :传递给驱动的 inode
                           :设备文件, file 结构体有个叫做 private_data 的成员变量
    58 * @param – filp
                           一般在 open 的时候将 private_data 指向设备结构体。
    59 *
                           :0 成功;其他 失败
    60 * @return
```



```
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
    61 */
    62 static int key_open(struct inode *inode, struct file *filp)
    63 {
    64
        return 0;
    65 }
    66
    67 /*
    68 * @description
                     :从设备读取数据
                     :要打开的设备文件(文件描述符)
    69 * @param – filp
    70 * @param – buf
                     :返回给用户空间的数据缓冲区
    71 * @param – cnt
                     :要读取的数据长度
    72 * @param – offt
                     :相对于文件首地址的偏移
   73 * @return
                      :读取的字节数,如果为负值,表示读取失败
   74 */
    75 static ssize_t key_read(struct file *filp, char __user *buf,
    76
            size_t cnt, loff_t *offt)
    77 {
    78
        unsigned long flags;
    79
        int ret;
    80
        /* 自旋锁上锁 */
    81
    82
        spin_lock_irqsave(&key.spinlock, flags);
    83
    84
       /* 将按键状态信息发送给应用程序 */
    85
        ret = copy_to_user(buf, &status, sizeof(int));
    86
       /* 状态重置 */
    87
    88
        status = KEY_KEEP;
    89
    90
        /* 自旋锁解锁 */
    91
        spin_unlock_irqrestore(&key.spinlock, flags);
    92
    93
        return ret;
    94 }
    95
    96 /*
    97 * @description
                         : 向设备写数据
                         :设备文件,表示打开的文件描述符
   98 * @param – filp
   99 * @param – buf
                         :要写给设备写入的数据
   100 * @param - cnt
                         :要写入的数据长度
   101 * @param – offt
                         :相对于文件首地址的偏移
```



```
原子哥在线教学: www.yuanzige.com
                                             论坛:www.openedv.com/forum.php
                            :写入的字节数,如果为负值,表示写入失败
    102 * @return
    103 */
    104 static ssize_t key_write(struct file *filp, const char __user *buf,
    105
             size_t cnt, loff_t *offt)
    106 {
    107
           return 0;
    108 }
    109
    110 /*
    111 * @description
                        : 关闭/释放设备
    112 * @param – filp
                            :要关闭的设备文件(文件描述符)
    113 * @return
                            :0 成功;其他 失败
    114 */
    115 static int key_release(struct inode *inode, struct file *filp)
    116 {
    117 return 0;
    118 }
    119
    120 static void key_timer_function(struct timer_list *unused)
    121 {
    122
         static int last_val = 1;
    123
         unsigned long flags;
    124
         int current_val;
    125
    126 /* 自旋锁上锁 */
    127
         spin_lock_irqsave(&key.spinlock, flags);
    128
    129
         /* 读取按键值并判断按键当前状态 */
    130
         current_val = gpio_get_value(key.key_gpio);
    131
         if (0 == current_val && last_val) // 按下
    132
           status = KEY_PRESS;
    133
         else if (1 == current_val && !last_val)
    134
           status = KEY_RELEASE; // 松开
    135
         else
    136
           status = KEY_KEEP;
                                     // 状态保持
    137
    138
         last val = current val;
    139
    140
         /* 自旋锁解锁 */
    141
         spin_unlock_irqrestore(&key.spinlock, flags);
    142
```



```
原子哥在线教学: www.yuanzige.com
                                              论坛:www.openedv.com/forum.php
    143
    144 static irqreturn_t key_interrupt(int irq, void *dev_id)
    145 {
    146 /* 按键防抖处理, 开启定时器延时 15ms */
    147
          mod_timer(&key.timer, jiffies + msecs_to_jiffies(15));
          return IRQ HANDLED;
    148
    149 }
    150
    151 static int key_parse_dt(void)
    152 {
    153
         struct device_node *nd;
    154
         const char *str;
    155
         int ret;
    156
    157
         /* 获取 key 节点 */
    158
         nd = of_find_node_by_path("/key");
    159
         if(NULL == nd) \{
    160
            printk(KERN_ERR "key: Failed to get key node\n");
    161
            return -EINVAL;
    162
         }
    163
    164
         /* 读取 status 属性 */
         ret = of_property_read_string(nd, "status", &str);
    165
    166
         if(!ret) {
    167
            if (strcmp(str, "okay"))
    168
              return -EINVAL;
    169
         }
    170
         /* 获取 compatible 属性值并进行匹配 */
    171
    172
          ret = of_property_read_string(nd, "compatible", &str);
    173
          if(ret)
    174
            return ret;
    175
          if (strcmp(str, "alientek,key")) {
    176
            printk(KERN_ERR "key: Compatible match failed\n");
    177
    178
            return -EINVAL;
    179
         }
    180
         /* 获取设备树中的 key-gpio 属性,得到按键的 GPIO 编号 */
    181
    182
         key.key_gpio = of_get_named_gpio(nd, "key-gpio", 0);
    183
         if(!gpio_is_valid(key.key_gpio)) {
```



```
原子哥在线教学: www.yuanzige.com
                                        论坛:www.openedv.com/forum.php
           printk(KERN_ERR "key: Failed to get key-gpio\n");
    184
    185
           return -EINVAL;
    186 }
    187
    188 /* 获取 GPIO 对应的中断号 */
    189
         key.irq_num = irq_of_parse_and_map(nd, 0);
    190
         if (!key.irq_num)
    191
           return -EINVAL;
    192
    193 return 0;
    194 }
    195
    196 static int key_gpio_init(void)
    197 {
    198
         unsigned long irq_flags;
    199
         int ret;
    200
    201
        /* 申请使用 GPIO */
    202
        ret = gpio_request(key.key_gpio, "Key Gpio");
    203
         if (ret)
    204
           return ret;
    205
         /* 将 GPIO 设置为输入模式 */
    206
    207
         gpio_direction_input(key.key_gpio);
    208
   209
         /* 获取设备树中指定的中断触发类型 */
   210
         irq_flags = irq_get_trigger_type(key.irq_num);
   211
         if (IRQF_TRIGGER_NONE == irq_flags)
    212
           irq_flags = IRQF_TRIGGER_FALLING | IRQF_TRIGGER_RISING;
    213
   214 /* 申请中断 */
   215 ret = request_irq(key.irq_num, key_interrupt, irq_flags, "PS Key0 IRQ", NULL);
   216 if (ret) {
   217
           gpio_free(key.key_gpio);
   218
           return ret;
   219
        }
   220
   221 return 0;
   222 }
    223
   224 /* 设备操作函数 */
```



```
原子哥在线教学: www.yuanzige.com
                                             论坛:www.openedv.com/forum.php
    225 static struct file_operations key_fops = {
   226
         .owner
                    = THIS_MODULE,
   227 .open
                    = key_open,
   228 .read
                    = key_read,
   229 .write
                    = key_write,
   230 .release
                    = key_release,
   231 };
   232
   233 static int __init mykey_init(void)
   234 {
   235
         int ret;
    236
   237 /* 初始化自旋锁 */
   238
         spin_lock_init(&key.spinlock);
   239
   240 /* 设备树解析 */
   241 ret = key_parse_dt();
    242
        if (ret)
    243
           return ret;
    244
   245 /* GPIO、中断初始化 */
   246
         ret = key_gpio_init();
   247
         if (ret)
   248
           return ret;
    249
   250 /* 初始化 cdev */
   251
         key.cdev.owner = THIS_MODULE;
   252
         cdev_init(&key.cdev, &key_fops);
    253
    254
         /* 添加 cdev */
    255
         ret = alloc_chrdev_region(&key.devid, 0, KEY_CNT, KEY_NAME);
         if (ret)
    256
    257
           goto out1;
    258
   259
         ret = cdev_add(&key.cdev, key.devid, KEY_CNT);
    260
         if (ret)
    261
           goto out2;
    262
   263
        /* 创建类 */
    264
         key.class = class_create(THIS_MODULE, KEY_NAME);
    265 if (IS_ERR(key.class)) {
```



```
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
    266
           ret = PTR_ERR(key.class);
    267
           goto out3;
         }
    268
    269
   270 /* 创建设备 */
   271
         key.device = device_create(key.class, NULL,
    272
                key.devid, NULL, KEY_NAME);
   273
         if (IS_ERR(key.device)) {
    274
           ret = PTR_ERR(key.device);
    275
           goto out4;
    276
         }
    277
   278
         /* 初始化定时器 */
   279
         timer_setup(&key.timer, key_timer_function,0);
    280
    281
         return 0;
    282
   283 out4:
   284
         class_destroy(key.class);
    285
   286 out3:
    287
         cdev_del(&key.cdev);
    288
   289 out2:
   290
         unregister_chrdev_region(key.devid, KEY_CNT);
   291
   292 out1:
   293
         free_irq(key.irq_num, NULL);
    294
         gpio_free(key.key_gpio);
    295
    296 return ret;
   297 }
   298
   299 static void __exit mykey_exit(void)
   300 {
    301 /* 删除定时器 */
   302
         del_timer_sync(&key.timer);
    303
         /* 注销设备 */
    304
    305
         device_destroy(key.class, key.devid);
    306
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

/* 注销类 */ 307 308 class destroy(key.class); 309 310 /* 删除 cdev */ cdev_del(&key.cdev); 311 312 313 /* 注销设备号 */ 314 unregister_chrdev_region(key.devid, KEY_CNT); 315 316 /* 释放中断 */ 317 free_irq(key.irq_num, NULL); 318 319 /* 释放 GPIO */ 320 gpio_free(key.key_gpio); 321 } 322 323 /* 驱动模块入口和出口函数注册 */ 324 module_init(mykey_init); 325 module_exit(mykey_exit); 326 327 MODULE_AUTHOR("DengTao <773904075@qq.com>"); 328 MODULE_DESCRIPTION("Gpio Key Interrupt Driver");

329 MODULE_LICENSE("GPL");

第 34~38 行,定义了一个枚举类型,包含 3 个常量 KEY_PRESS、KEY_RELEASE、 KEY_KEEP,分别用来表示按键的 3 种不同的状态,即按键按下、按键松开以及按键状态保持。

第 41~50 行,结构体 key_dev 为按键设备所对应的结构体,key_gpio 为按键 GPIO 编号, irq_num 为按键 IO 对应的中断号;除此之外,结构体当中还定义了一个定时器用于实现按键 的去抖操作,还定义了一个自旋锁用于实现对关键代码的保护操作。

第52行, 定义一个按键设备 key。

第53行,定义一个 int 类型的静态全局变量 status 用来表示按键的状态。

第 75~94 行, key_read 函数, 对应应用程序的 read 函数。此函数向应用程序返回按键状态信息数据; 这个函数其实很简单, 使用 copy_to_user 函数直接将 status 数据发送给应用程序, status 变量保存了按键当前的状态,发送完成之后再将按键状态重置即可! 需要注意的是, 该函数中使用了自旋锁进行保护。

第 120~142 行, key_timer_function 函数为定时器定时处理函数。该函数中定义了一个静态局部变量 last_val 用来保存按键上一次读取到的值,变量 current_val 用来存放当前按键读取到的值;第 130~136 行,通过读取到的按键值以及上一次读取到的值来判断按键当前所属的状态,如果本次读取的值为 0,而上一次读取的值 1,则表示按键按下;如果本次读取的值为 1,而上一次读取的值 0,则表示按键松开;如果本次读取的值为 0,而上一次读取的值也是 1,则表示没有



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php 触碰按键。第 138 行,当状态判断完成之后,会将 current_val 的值赋值给 last_val。本函数中 也使用自旋锁对全局变量 status 进行加锁保护!

第 144~149 行, key_interrupt 函数是按键 PS_KEY0 中断处理函数,参数 dev_id 是一个 void 类型的指针,本驱动程序并没使用到这个参数;这个中断处理函数很简单直接开启定时 器,延时 15 毫秒,用于实现按键的软件防抖。

第 151~194 行, key_parse_dt 函数中主要是对设备树中的属性进行了解析, 获取设备树中的 key 节点, 通过 of_get_named_gpio 函数得到按键的 GPIO 编号, 通过 irq_of_parse_and_map 函数获取按键的中断号, irq_of_parse_and_map 函数会解析 key 节点中的 interrupt-parent 和 interrupts 属性然后得到一个中断号, 后面就可以使用这个中断号去申请以及释放中断了。

第 196~222 行, key_gpio_init 函数中主要对 GPIO 以及中断进行了相关的初始化。使用 gpio_request 函数申请 GPIO 使用权,通过 gpio_direction_input 将 GPIO 设置为输出模式; irq_get_trigger_type 函数可以获取到 key 节点中定义的中断触发类型,最后使用 request_irq 申 请中断,并设置 key_interrupt 函数作为我们的按键中断处理函数,当按键中断发生之后便会 跳转到该函数执行; request_irq 函数会默认使能中断,所以不需要 enable_irq 来使能中断,当 然我们也可以在申请成功之后先使用 disable_irq 函数禁用中断,等所有工作完成之后再来使 能中断,这样会比较安全,建议大家这样使用。

第 225~231 行,按键设备的 file_operations 结构体。

第 233~298 行, mykey_init 是驱动入口函数, 第 238 行调用 spin_lock_init 初始化自旋锁变 量, 第 279 行对定时器进行初始化并将 key_timer_function 函数绑定为定时器定时处理函数, 当定时时间到了之后便会跳转到该函数执行。

第 299~321 行, mykey_exit 驱动出口函数, 第 302 行调用 del_timer_sync 函数删除定时器, 代码中已经注释得非常详细了, 这里便不再多说!

33.3.3 编写测试 APP

测试 APP 要实现的内容很简单,通过不断的读取/dev/key 设备文件来获取按键值来判断 当前按键的状态,从按键驱动上传到应用程序的数据可以有 3 个值,分别为 0、1、2;0 表示 按键按下时的这个状态,1 表示按键松开时对应的状态,而2 表示按键一直被按住或者松开; 搞懂数据代表的意思之后,我们开始编写测试程序,在 13_irq 目录下新建名为 keyApp.c 的文 件,然后输入如下所示内容:

示例代码 33.3.4 keyApp.c 文件代码

1 /************************************					
2 Copyright © ALIENTEK Co., Ltd. 1998-2029. All rights reserved.					
3 文件名	: keyApp.c				
4 作者	:邓涛				
5 版本	: V1.0				
6 描述	: 按键测试应用程序				
7 其他	:无				
8 使用方法	: ./keyApp /dev/key				
<mark>9</mark> 论坛	: www.openedv.com				
10 日志	:初版 V1.0 2019/1/30 邓涛创建				
11 ************************************					



```
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
    12
    13 #include <stdio.h>
    14 #include <unistd.h>
    15 #include <sys/types.h>
    16 #include <sys/stat.h>
    17 #include <fcntl.h>
    18 #include <stdlib.h>
    19 #include <string.h>
    20
    21 /*
    22 * @description
                            : main 主程序
    23 * @param – argc
                             : argv 数组元素个数
    24 * @param – argv
                             :具体参数
    25 * @return
                             :0 成功;其他 失败
    26 */
    27 int main(int argc, char *argv[])
    28 {
    29
        int fd, ret;
    30
        int key_val;
    31
    32
        /* 判断传参个数是否正确 */
        if(2 != argc) {
    33
    34
           printf("Usage:\n"
    35
              "\t./keyApp /dev/key\n"
    36
             );
    37
           return -1;
    38
        }
    39
    40 /* 打开设备 */
    41
         fd = open(argv[1], O_RDONLY);
    42
         if(0 > fd) \{
    43
           printf("ERROR: %s file open failed!\n", argv[1]);
    44
           return -1;
    45
        }
    46
    47
        /* 循环读取按键数据 */
         for(;;){
    48
    49
    50
           read(fd, &key_val, sizeof(int));
    51
           if (0 == key_val)
    52
             printf("Key Press\n");
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

```
53 else if (1 == key_val)
54 printf("Key Release\n");
55 }
56
57 /* 关闭设备 */
58 close(fd);
59 return 0;
60 }
```

第 48~55 行使用 for 循环不断的读取按键值,如果读取到的值是 0 则打印 "Key Press"字符串,而过读取到的值是 1 则打印 "Key Release"字符串。

33.4 运行测试

33.4.1 编译驱动程序和测试 APP

1、编译驱动程序

编写 Makefile 文件,直接将上一章实验目录 12_timer 下的 Makefile 文件拷贝到本实验目 录下,打开该 Makefile 文件,将 obj-m 变量的值改为 keyirq.o, Makefile 内容如下所示:

示例代码 33.4.1 Makefile 文件

1 KERN_DIR :=/home/zy/workspace/kernel-driver/linux-xlnx-xlnx_rebase_v5.4_2020.2
2
3 obj-m :=keyirq.o
4
5 all:
6 make -C \$(KERN_DIR) M=`pwd` modules
7 clean:
8 make -C \$(KERN_DIR) M=`pwd` clean
第 3 行,设置 obj-m 变量的值为 keyirq.o。
修改完成之后保存退出,输入如下命令编译出驱动模块文件:
make
编译成功以后就会生成一个名为 "keyirq.ko" 的驱动模块文件,如下所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php zy@zy-virtual-machine:~/linux/drivers/13 irg\$ zy@zy-virtual-machine:~/linux/drivers/13 irq\$ ls keyApp.c keyirq.c Makefile zy@zy-virtual-machine:~/linux/drivers/13 irq\$ zy@zy-virtual-machine:~/linux/drivers/13 irq\$ make make -C /home/zy/workspace/kernel-driver/linux-xlnx-xlnx rebase v5.4 2020.2 M=`p wd` modules make[1]: 进入目录"/home/zy/workspace/kernel-driver/linux-xlnx-xlnx rebase v5.4 2 020.2" CC [M] /home/zy/linux/drivers/13 irg/keyirg.o Building modules, stage 2. MODPOST 1 modules CC [M] /home/zy/linux/drivers/13 irq/keyirq.mod.o LD [M] /home/zy/linux/drivers/13_irq/keyirq.ko make[1]: 离开目录"/home/zy/workspace/kernel-driver/linux-xlnx-xlnx_rebase_v5.4_2 020.2" zy@zy-virtual-machine:~/linux/drivers/13 irq\$ zy@zy-virtual-machine:~/linux/drivers/13_irq\$ zy@zy-virtual-machine:~/linux/drivers/13 irq\$ ls keyApp.c keyirq.ko keyirq.mod.c keyirq.o modules.order keyirq.c keyirq.mod keyirq.mod.o Makefile Module.symvers zy@zy-virtual-machine:~/linux/drivers/13_irq\$ 图 33.4.1 编译驱动模块

2、编译测试 APP

输入如下命令编译测试 keyApp.c 这个测试程序:

\$CC keyApp.c -o keyApp

编译成功以后就会生成 keyApp 这个应用程序。

33.4.2 运行测试

将上面编译出来的 keyirq.ko 和 keyApp 这两个文件拷贝到 NFS 共享目录下根文件系统的 /lib/modules/5.4.0-xilinx 文件夹中。

将 SD 启动卡插到开发板中,连接开发板串口、网口和电源线,设置拨码开关为 SD 卡启动模式,然后打开电源,启动开发板。

系统启动完成后,通过串口终端进入"/lib/modules/5.4.0-xilinx"目录下,输入如下命令 加载 keyirq.ko 驱动模块:

depmod //第一次加载驱动的时候需要运行此命令

modprobe keyirq.ko //加载驱动

驱动加载成功以后可以通过查看/proc/interrupts 文件来检查一下对应的中断有没有被注册 上,输入如下命令:

cat /proc/interrupts 结果如下图所示:



原子哥在	线教学: www.	yuanzige.co	om t	论坛:www.op	penedv.com/forum.php
root@A	LIENTEK-ZYNQ-	driver:/li	b/module:	s/5.4.0-xil	linx#
root@A	LIENTEK-ZYNQ-	driver:/li	b/module:	s/5.4.0-xil	linx# cat /proc/interrupts
	CPU0	CPU1			
16:	Θ	Θ	GIC-0	27 Edge	gt _
17:	4507	5896	GIC-0	29 Edge	twd
18:	Θ	Θ	GIC-0	37 Level	arm-pmu
19:	Θ	Θ	GIC-0	38 Level	arm-pmu
20:	43	Θ	GIC-0	39 Level	f8007100.adc
23:	Θ	Θ	GIC-0	57 Level	cdns-12c
24:	Θ	Θ	GIC-0	80 Level	çdns-12c
26:	Θ	Θ	GIC-0	35 Level	†800c000.ocmc
27:	133	Θ	GIC-0	59 Level	xuartps
29:	Θ	Θ	GIC-0	51 Level	e000d000.spi
30:	10121	Θ	GIC-0	54 Level	eth0
31:	Θ	Θ	GIC-0	77 Level	ethl
32:	419	Θ	GIC-0	56 Level	mmcΘ
33:	500	Θ	GIC-0	79 Level	mmcl
34:	Θ	Θ	GIC-0	45 Level	f8003000.dmac
35:	Θ	Θ	GIC-0	46 Level	f8003000.dmac
36:	Θ	Θ	GIC-0	47 Level	f8003000.dmac
37:	Θ	Θ	GIC-0	48 Level	f8003000.dmac
38:	Θ	Θ	GIC-0	49 Level	f8003000.dmac
39:	Θ	Θ	GIC-0	72 Level	f8003000.dmac
40:	Θ	Θ	GIC-0	73 Level	f8003000.dmac
41:	Θ	Θ	GIC-0	74 Level	f8003000.dmac
42:	Θ	Θ	GIC-0	75 Level	f8003000.dmac
43:	Θ	Θ	GIC-0	40 Level	f8007000.devcfg
45:	Θ	Θ	GIC-0	43 Level	ttc_clockevent
51:	Θ	Θ	GIC-0	41 Edge	18005000.watchdog
56:	Θ	Θ	GIC-0	63 Level	xilinx_framebuffer
50:	<u>e</u>		GIC 9	61 Lovel	x1linx_tramebuttor
60:	0	Θz	ynq-gpio	12 Edge	PS Key0 IRQ
1P11.	2240	5202	Deschodu	ling inter	terrupts
IPIZ:	2249	5393	Function	cing intern	up ts
1P13:	1	2	CDU oton	internet	rupts
1P14:	U	0	TPO STOP	interrupts	
IPI5:	U	0	IRU WORK	interrupts	
1P10:	0	Θ	compteri	on interrup	
EII:		drivor, (1.	h (modula	o /5 / 0 vit	inv#
LOOT @A	LIENIEK-ZYNQ-	ariver:/li	b/module	s/5.4.0-X1	LINX#

图 33.4.2 查看当前系统已经使用的中断号

图 33.4.2 中红框标识的中断信息就是 keyirq 按键中断驱动程序中使用的中断,中断号为 60,触发方式为跳边沿(Edge),中断命名为 "PS Key0 IRQ"。

接下来使用如下命令来测试按键中断工作是否正常了:

./keyApp /dev/key

按下开发板上的 PS_KEY0 键,终端就会打印出字符串,如图 33.4.3 所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

roo	t@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx# ./keyApp /dev/key
Кеу	Press
Key	Release
Кеу	Press
Кеу	Release
Кеу	Press
Кеу	Release
Кеу	Press
Кеу	Release

图 33.4.3 按键中断测试结果

从图 33.4.3 可以看出,当我们按下 PS_KEY0 按键时,终端会打印出"Key Press"字符串, 当松开按键时,中断会打印出"Key Release"字符串;由于我们在驱动程序中加入了软件防 抖处理,所以效果会比第三十一章测试时好很多。

如果要卸载驱动的话输入如下命令即可:

rmmod keyirq.ko



正点原子

第三十四章 Linux 阻塞和非阻塞 IO 实验

阻塞和非阻塞 IO 是 Linux 驱动开发里面很常见的两种设备访问模式,在编写驱动的时候 一定要考虑到阻塞和非阻塞。本章我们就来学习一下阻塞和非阻塞 IO,以及如何在驱动程序 中处理阻塞与非阻塞,如何在驱动程序使用等待队列和 poll 机制。



原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php

34.1 阻塞和非阻塞 IO

34.1.1 阻塞和非阻塞简介

这里的"IO"并不是我们学习 STM32 或者其他单片机的时候所说的"GPIO"(也就是引脚)。这里的 IO 指的是 Input/Output,也就是输入/输出,是应用程序对驱动设备的输入/输出操作。当应用程序对设备驱动进行操作的时候,如果不能获取到设备资源,那么阻塞式 IO 就会将应用程序对应的线程挂起,直到设备资源可以获取为止。对于非阻塞 IO,应用程序对应的线程不会挂起,它要么一直轮询等待,直到设备资源可以使用,要么就直接放弃。阻塞式 IO 如图 34.1.1 所示:





图 34.1.1 中应用程序调用 read 函数从设备中读取数据,当设备不可用或数据未准备好的时候就会进入到休眠态。等设备可用的时候就会从休眠态唤醒,然后从设备中读取数据返回 给应用程序。非阻塞 IO 如图 34.1.2 所示:





从图 34.1.2 可以看出,应用程序使用非阻塞访问方式从设备读取数据,当设备不可用或 数据未准备好的时候会立即向内核返回一个错误码,表示数据读取失败。应用程序会再次重 新读取数据,这样一直往复循环,直到数据读取成功。

应用程序可以使用如下所示示例代码来实现阻塞访问:

示例代码 34.1.1 应用程序阻塞读取数据



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

从示例代码 34.1.1 可以看出,对于设备驱动文件的默认读取方式就是阻塞式的,所以我 们前面所有的例程测试 APP 都是采用阻塞 IO。

如果应用程序要采用非阻塞的方式来访问驱动设备文件,可以使用如下所示代码:

示例代码 34.1.2 应用程序非阻塞读取数据

```
1 int fd;
2 int data = 0;
3
4 fd = open("/dev/xxx_dev", O_RDWR | O_NONBLOCK); /* 非阻塞方式打开 */
5 ret = read(fd, &data, sizeof(data));
                                                       /* 读取数据 */
```

第 4 行使用 open 函数打开"/dev/xxx_dev"设备文件的时候添加了参数 "O NONBLOCK",表示以非阻塞方式打开设备,这样从设备中读取数据的时候就是非阻 塞方式的了。

34.1.2 等待队列

1、等待队列头

阻塞访问最大的好处就是当设备文件不可操作的时候进程可以进入休眠态,这样可以将 CPU 资源让出来。但是,当设备文件可以操作的时候就必须唤醒进程,一般在中断函数里面 完成唤醒工作。Linux 内核提供了等待队列(wait queue)来实现阻塞进程的唤醒工作,如果我们 要在驱动中使用等待队列,必须创建并初始化一个等待队列头,等待队列头使用结构体 wait queue head t 表示, wait queue head t 结构体定义内核源码目录 include/linux/wait.h 头文 件中,结构体内容如下所示:

示例代码 34.1.3 wait queue head t 结构体

34 struct wait_queue_head { 35 spinlock_t lock;

36 struct list_head head;

37 };

38 **typedef** struct wait_queue_head wait_queue_head_t;

定义好等待队列头以后需要初始化,使用 init_waitqueue_head 初始化等待队列头, init_waitqueue_head 其实是一个宏定义,也是定义在 include/linux/wait.h 头文件中,原型如下: \

#define init_waitqueue_head(wq_head)

do {

static struct lock_class_key __key;

\

__init_waitqueue_head((wq_head), #wq_head, &__key);

 $\}$ while (0)

参数 wq_head 就是要初始化的等待队列头。

也可以使用宏 DECLARE WAIT QUEUE HEAD 来一次性完成等待队列头的定义和初始 化。

2、等待队列项



原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php

等待队列头就是一个等待队列的头部,每个访问设备的进程都是一个队列项,当设备不 可用的时候就要将这些进程对应的等待队列项添加到等待队列里面。结构体 wait_queue_entry_t表示等待队列项,该结构体内容如下:

示例代码 34.1.4 wait queue entry t 结构体

```
14 typedef struct wait_queue_entry wait_queue_entry_t;
```

..... 24 /*

25 * A single wait-queue entry structure:

26 */

27 struct wait queue entry {

28 unsigned int flags;

29 void *private;

30 wait_queue_func_tfunc;

31 struct list_head entry;

32};

使用宏 DECLARE_WAITQUEUE 定义并初始化一个等待队列项, 宏的内容如下: #define DECLARE WAITQUEUE(name, tsk) \

struct wait_queue_entry name = __WAITQUEUE_INITIALIZER(name, tsk)

name 就是等待队列项的名字, tsk 表示这个等待队列项属于哪个任务(进程), 一般设置为 current,在 Linux 内核中 current 相当于一个全局变量,表示当前进程。因此宏 DECLARE_WAITQUEUE 就是给当前正在运行的进程创建并初始化了一个等待队列项。

3、将队列项添加/移除等待队列头

当设备不可访问的时候就需要将进程对应的等待队列项添加到前面创建的等待队列头中, 只有添加到等待队列头中以后进程才能进入休眠态。当设备可以访问以后再将进程对应的等 待队列项从等待队列头中移除即可,等待队列项添加 API 函数如下:

void add_wait_queue(struct wait_queue_head *wq_head, struct wait_queue_entry *wq_entry);

函数参数和返回值含义如下:

wq head: 等待队列项要加入的等待队列头。

wq_entry:要加入的等待队列项。

返回值:无。

等待队列项移除 API 函数如下:

void remove_wait_queue(struct wait_queue_head *wq_head, struct wait_queue_entry *wq_entry); 函数参数和返回值含义如下:

wq_head: 要删除的等待队列项所处的等待队列头。

wq entry: 要删除的等待队列项。

返回值:无。

4、等待唤醒

当设备可以使用的时候就要唤醒进入休眠态的进程,唤醒可以使用如下:

void wake_up(wait_queue_head_t *q)

void wake up interruptible(wait queue head t *q)



原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php

参数 q 就是要唤醒的等待队列头,这两个函数会将这个等待队列头中的所有进程都唤醒。 wake_up 函数可以唤醒处于 TASK_INTERRUPTIBLE 和 TASK_UNINTERRUPTIBLE 状态的进程,而 wake_up_interruptible 函数只能唤醒处于 TASK_INTERRUPTIBLE 状态的进程。

5、等待事件

除了主动唤醒以外,也可以设置等待队列等待某个事件,当这个事件满足以后就自动唤醒等待队列中的进程,和等待事件有关的 API 函数如表 34.1.2.1 所示:

函数	描述
	等待以 wq 为等待队列头的等待队列
wait event(wa condition)	被唤醒,前提是 condition 条件必须满足(为
wan_event(wq, condition)	真),否则一直阻塞。此函数会将进程设置
	为 TASK_UNINTERRUPTIBLE 状态
	功能和 wait_event 类似,但是此函数
	可以添加超时时间,以 jiffies 为单位。此
wait_event_timeout(wq,	函数有返回值,如果返回 0 的话表示超时
condition, timeout)	时间到,而且 condition 为假。为1的话表
	示 condition 为真,也就是条件满足了。
	与 wait_event 函数类似,但是此函数
wait_event_interruptible(wq,	将进程设置为 TASK_INTERRUPTIBLE,
condition)	就是可以被信号打断。
	与 wait_event_timeout 函数类似,此函
wait_event_interruptible_timeout	数 也 将 进 程 设 置 为
(wq, condition, timeout)	TASK_INTERRUPTIBLE,可以被信号打
	断。

表 34.1.2.1 等待事件 API 函数

34.1.3 轮询

如果用户应用程序以非阻塞的方式访问设备,设备驱动程序就要提供非阻塞的处理方式,也就是轮询。poll、epoll和 select可以用于处理轮询,应用程序通过 select、epoll或 poll函数 来查询设备是否可以操作,如果可以操作的话就从设备读取或者向设备写入数据。当应用程序调用 select、epoll或 poll函数的时候设备驱动程序中的 poll函数就会执行,因此需要在设备驱动程序中编写 poll函数。我们先来看一下应用程序中使用的 select、poll和 epoll这三个函数。

1、select 函数

select 函数原型如下:

int select(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout) 函数参数和返回值含义如下:

nfds: 要操作的文件描述符个数。

readfds、writefds 和 exceptfds: 这三个指针指向描述符集合,这三个参数指明了关心哪些描述符、需要满足哪些条件等等,这三个参数都是 fd_set 类型的, fd_set 类型变量的每一个位都代表了一个文件描述符。readfds 用于监视指定描述符集的读变化,也就是监视这些文件是否可以读取,只要这些集合里面有一个文件可以读取那么 seclect 就会返回一个大于 0 的值



原子哥在线教学: www.vuanzige.com 论坛:www.openedv.com/forum.php

表示文件可以读取。如果没有文件可以读取,那么就会根据 timeout 参数来判断是否超时。可以将 readfs 设置为 NULL,表示不关心任何文件的读变化。writefds 和 readfs 类似,只是 writefs 用于监视这些文件是否可以进行写操作。exceptfds 用于监视这些文件的异常。

比如我们现在要从一个设备文件中读取数据,那么就可以定义一个 fd_set 变量,这个变量要传递给参数 readfds。当我们定义好一个 fd_set 变量以后可以使用如下所示几个宏进行操作:

void FD_ZERO(fd_set *set)

void FD_SET(int fd, fd_set *set)

void FD_CLR(int fd, fd_set *set)

int FD_ISSET(int fd, fd_set *set)

FD_ZERO用于将 fd_set 变量的所有位都清零, FD_SET 用于将 fd_set 变量的某个位置 1, 也就是向 fd_set 添加一个文件描述符,参数 fd 就是要加入的文件描述符。FD_CLR 用户将 fd_set 变量的某个位清零,也就是将一个文件描述符从 fd_set 中删除,参数 fd 就是要删除的文 件描述符。FD_ISSET 用于测试 fd_set 的某个位是否置 1,也就是判断某个文件是否可以进行 操作,参数 fd 就是要判断的文件描述符。

timeout:超时时间,当我们调用 select 函数等待某些文件描述符可以设置超时时间,超时时间使用结构体 timeval 表示,结构体定义如下所示:

struct timeval { long tv_sec; /* 秒 */ long tv_usec; /* 微妙 */ };

当 timeout 为 NULL 的时候就表示无限期的等待。

返回值: 0,表示的话就表示超时发生,但是没有任何文件描述符可以进行操作;-1,发 生错误;其他值,可以进行操作的文件描述符个数。

使用 select 函数对某个设备驱动文件进行读非阻塞访问的操作示例如下所示:

示例代码 34.1.5 select 函数非阻塞读访问示例

```
1 void main(void)
```

```
2 {
                      /* 要监视的文件描述符 */
3
   int ret, fd;
4
   fd_set readfds;
                      /* 读操作文件描述符集 */
   struct timeval timeout; /* 超时结构体 */
5
6
7
   fd = open("dev xxx", O RDWR | O NONBLOCK); /* 非阻塞式访问 */
8
9
   FD_ZERO(&readfds);
                         /* 清除 readfds */
10
   FD_SET(fd, &readfds);
                          /* 将 fd 添加到 readfds 里面 */
11
   /* 构造超时时间 */
12
13
   timeout.tv_sec = 0;
14
    timeout.tv usec = 500000; /* 500ms */
15
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

```
ret = select(fd + 1, &readfds, NULL, NULL, &timeout);
16
17
    switch (ret) {
                    /* 超时 */
18
    case 0:
19
      printf("timeout!\r\n");
20
      break;
                    /* 错误 */
21
    case -1:
22
      printf("error!\r\n");
23
      break;
24
    default:
                    /* 可以读取数据 */
                                   /* 判断是否为 fd 文件描述符 */
      if(FD ISSET(fd, &readfds)) {
25
                                     /* 使用 read 函数读取数据 */
26
27
      }
      break;
28
29 }
30 }
```

2、poll 函数

在单个线程中, select 函数能够监视的文件描述符数量有最大的限制, 一般为1024, 可以 修改内核将监视的文件描述符数量改大, 但是这样会降低效率! 这个时候就可以使用 poll 函 数, poll 函数本质上和 select 没有太大的差别, 但是 poll 函数没有最大文件描述符限制, Linux 应用程序中 poll 函数原型如下所示:

```
int poll(struct pollfd *fds, nfds_t nfds, int timeout)
```

```
函数参数和返回值含义如下:
```

fds:要监视的文件描述符集合以及要监视的事件,为一个数组,数组元素都是结构体 pollfd 类型的, pollfd 结构体如下所示:

struct pollfd {

```
int fd; /* 文件描述符 */
short events; /* 请求的事件 */
short revents; /* 返回的事件 */
```

};

fd 是要监视的文件描述符,如果 fd 无效的话那么 events 监视事件也就无效,并且 revents 返回 0。events 是要监视的事件,可监视的事件类型如下所示:

POLLIN	有数据可以读取。
POLLPRI	有紧急的数据需要读取。
POLLOUT	可以写数据。
POLLERR	指定的文件描述符发生错误
POLLHUP	指定的文件描述符挂起。
POLLNVAL	无效的请求。
POLLRDNORM	等同于 POLLIN

revents 是返回参数,也就是返回的事件,有 Linux 内核设置具体的返回事件。

nfds: poll 函数要监视的文件描述符数量。

timeout: 超时时间,单位为ms。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php 返回值:返回 revents 域中不为 0 的 pollfd 结构体个数,也就是发生事件或错误的文件描 述符数量; 0, 超时; -1, 发生错误,并且设置 errno 为错误类型。 使用 poll 函数对某个设备驱动文件进行读非阻塞访问的操作示例如下所示: 示例代码 34.1.6 poll 函数读非阻塞访问示例 1 void main(void) 2 { 3 int ret; 4 int fd; /* 要监视的文件描述符 */ 5 struct pollfd fds; 6 7 fd = open(filename, O_RDWR | O_NONBLOCK); /* 非阻塞式访问 */ 8 9 /* 构造结构体 */ fds.fd = fd;10 fds.events = POLLIN; /* 监视数据是否可以读取 */ 11 12 /* 轮询文件是否可操作, 超时 500ms */ 13 ret = poll(&fds, 1, 500);14 if (ret) { /* 数据有效 */ 15 /* 读取数据 */ 16 17 ••••• 18 } else if (ret == 0) { /* 超时 */ 19 20 } else if (ret < 0) { /* 错误 */ 21 22 } 23 }

3、epoll 函数

传统的 selcet 和 poll 函数都会随着所监听的 fd 数量的增加,出现效率低下的问题,而且 poll 函数每次必须遍历所有的描述符来检查就绪的描述符,这个过程很浪费时间。为此, epoll 因运而生, epoll 就是为处理大并发而准备的,一般常常在网络编程中使用 epoll 函数。应 用程序需要先使用 epoll_create 函数创建一个 epoll 句柄, epoll_create 函数原型如下:

int epoll_create(int size)

函数参数和返回值含义如下:

size:从 Linux2.6.8 开始此参数已经没有意义了,随便填写一个大于 0 的值就可以。

返回值: epoll 句柄,如果为-1 的话表示创建失败。

epoll 句柄创建成功以后使用 epoll_ctl 函数向其中添加要监视的文件描述符以及监视的事件, epoll_ctl 函数原型如下所示:

int epoll_ctl(int epfd, int op, int fd, struct epoll_event *event) 函数参数和返回值含义如下:

epfd: 要操作的 epoll 句柄,也就是使用 epoll_create 函数创建的 epoll 句柄。


原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

op: 表示要对 epfd(epoll 句柄)进行的操作,可以设置为: EPOLL CTL ADD 向 epfd 添加文件参数 fd 表示的描述符。

EPOLL_CTL_MOD 修改参数 fd 的 event 事件。

EPOLL_CTL_DEL 从 epfd 中删除 fd 描述符。

fd: 要监视的文件描述符。

event:要监视的事件类型,为 epoll_event 结构体类型指针, epoll_event 结构体类型如下 所示:

struct epoll_event {

uint32_t events; /* epoll 事件 */

epoll_data_t data; /* 用户数据 */

};

结构体 epoll_event 的 events 成员变量表示要监视的事件,可选的事件如下所示:

EPOLLIN 有数据可以读取。

EPOLLOUT 可以写数据。

EPOLLPRI 有紧急的数据需要读取。

EPOLLERR 指定的文件描述符发生错误。

EPOLLHUP 指定的文件描述符挂起。

EPOLLET 设置 epoll 为边沿触发,默认触发模式为水平触发。

EPOLLONESHOT 一次性的监视,当监视完成以后还需要再次监视某个 fd,那么就需要将 fd 重新添加 到 epoll 里面。

上面这些事件可以进行"或"操作,也就是说可以设置监视多个事件。

返回值: 0,成功; -1,失败,并且设置 errno 的值为相应的错误码。

一切都设置好以后应用程序就可以通过 epoll_wait 函数来等待事件的发生,类似 select 函数。epoll wait 函数原型如下所示:

int epoll_wait	(int	epfd,
	struct epoll_event	*events,
	int	maxevents
	int	timeout)

函数参数和返回值含义如下:

epfd:要等待的 epoll。

events: 指向 epoll_event 结构体的数组,当有事件发生的时候 Linux 内核会填写 events, 调用者可以根据 events 判断发生了哪些事件。

maxevents: events 数组大小,必须大于 0。

timeout: 超时时间,单位为ms。

返回值: 0, 超时; -1, 错误; 其他值, 准备就绪的文件描述符数量。

epoll 更多的是用在大规模的并发服务器上,因为在这种场合下 select 和 poll 并不适合。 当涉及到的文件描述符(fd)比较少的时候就适合用 selcet 和 poll,本章我们就使用 sellect 和 poll 这两个函数。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

34.1.4 Linux 驱动下的 poll 操作函数

当应用程序调用 select 或 poll 函数来对驱动程序进行非阻塞访问的时候,驱动程序 file_operations 操作集中的 poll 函数就会执行。所以驱动程序的编写者需要提供对应的 poll 函

数, poll函数原型如下所示:

unsigned int (*poll) (struct file *filp, struct poll_table_struct *wait)

函数参数和返回值含义如下:

filp: 要打开的设备文件(文件描述符)。

wait: 结构体 poll_table_struct 类型指针,由应用程序传递进来的。一般将此参数传递给 poll_wait 函数。

返回值:向应用程序返回设备或者资源状态,可以返回的资源状态如下:

POLLIN 有数据可以读取。

POLLPRI 有紧急的数据需要读取。

POLLOUT 可以写数据。

POLLERR 指定的文件描述符发生错误。

POLLHUP 指定的文件描述符挂起。

POLLNVAL 无效的请求。

POLLRDNORM 等同于 POLLIN, 普通数据可读

我们需要在驱动程序的 poll 函数中调用 poll_wait 函数, poll_wait 函数不会引起阻塞, 只 是将应用程序添加到 poll_table 中, poll_wait 函数原型如下:

void poll_wait(struct file * filp, wait_queue_head_t * wait_address, poll_table *p)

参数 wait_address 是要添加到 poll_table 中的等待队列头,参数 p 就是 poll_table,就是 file_operations 中 poll 函数的 wait 参数。

34.2 阻塞 IO 实验

在上一章Linux中断实验中,我们直接在应用程序中通过read函数不断的读取按键状态,当按键有效的时候就打印出按键值。这种方法有个缺点,那就是 keyApp 这个测试应用程序拥有很高的 CPU 占用率,大家可以在开发板中加载上一章的驱动程序模块 keyirq.ko,然后以后台运行模式运行上一章的应用测试程序 keyApp,命令如下:

./keyApp /dev/key &

测试驱动是否正常工作,如果驱动工作正常的话输入"top"命令查看 keyApp 这个应用程序的 CPU 使用率,结果如所示:



原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php

root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx# ./keyApp /dev/key & [1] 522 root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx# top Mem: 26448K used, 484080K free, 140K shrd, 268K buff, 7928K cached CPU: 9.0% usr 45.4% sys 0.0% nic 45.4% idle 0.0% io 0.0% irq 0.0% sirq Load average: 0.10 0.10 0.05 2/76 523

	PID	PPID USER	STAT	VSZ	%VSZ	CPU	%CPU	COMMAND
ĺ	522	512 root	R	1236	0.2	0	49.9	./keyApp /dev/key
	523	512 root	R	2304	0.4	1	4.5	top
	491	1 root	S	13020	2.5	1	0.0	/usr/sbin/tcf-agent -d -Ll0
	462	1 root	S	7224	1.4	1	0.0	/usr/sbin/haveged -w 1024 -v 1
	511	496 root	S	3144	0.6	Θ	0.0	/bin/login
	512	511 root	S	2716	0.5	1	0.0	-sh
	81	1 root	S	2416	0.4	1	0.0	/sbin/udevd -d
	496	1 root	S	2400	0.4	Θ	0.0	<pre>{start getty} /bin/sh /bin/start getty 115</pre>
	477	1 root	S	2304	0.4	1	0.0	/sbin/syslogd -n -0 /var/log/messages
	480	1 root	S	2304	0.4	1	0.0	/sbin/klogd -n
	473	1 root	S	2304	0.4	1	0.0	/usr/sbin/inetd
	497	1 root	S	2304	0.4	0	0.0	/sbin/getty 38400 tty1
	466	1 root	S	1952	0.3	0	0.0	/usr/sbin/dropbear -r /etc/dropbear/dropbe
	1	0 root	S	1388	0.2	1	00	init [5]

图 34.2.1 keyApp 程序 CPU 使用率

从图 34.2.1 可以看出, keyApp 这个应用程序的 CPU 使用率竟然高达 49.9%, 这仅仅是一个读取按键值的应用程序,这么高的 CPU 使用率显然是有问题的!原因就在于我们是直接在 while 循环中通过 read 函数读取按键值,因此 keyApp 这个软件会一直运行,一直读取按键值, CPU 使用率肯定就会很高。最好的方法就是在没有有效的按键事件发生的时候,让 keyApp 这个应用程序应该处于休眠状态,当有按键事件发生以后 keyApp 这个应用程序才运行,打印出 按键值,这样就会降低 CPU 使用率,本小节我们就使用阻塞 IO 来实现此功能。

34.2.1 硬件原理图分析

本章实验硬件原理图参考 31.2 小节即可。

34.2.2 实验程序编写

1、驱动程序编写

本 实 验 对 应 的 例 程 路 径 为 : ZYNQ 开 发 板 资 料 盘 (A 盘)\4_SourceCode\3_Embedded_Linux\Linux 驱动例程\14_blockio。

本章实验我们在上一章的"13_irq"实验的基础上完成,主要是对其添加阻塞访问相关的代码。在 drivers 目录下新建名为"14_blockio"的文件夹,将"13_irq"实验中的 keyirq.c 复制到 14_blockio 文件夹中,并重命名为 blockio.c。接下来我们就修改 blockio.c 这个文件,在其中添加阻塞相关的代码,完成以后的 blockio.c 内容如下所示(因为是在上一章实验的 keyirq.c 文件的基础上修改的,为了减少篇幅,下面的代码有省略):

示例代码 34.2.1 blockio.c 文件代码(有省略)

40 /* 按键设备结构体 */	
41 struct key_dev {	
42 dev_t devid;	/* 设备号 */



```
原子哥在线教学: www.yuanzige.com
                                             论坛:www.openedv.com/forum.php
                                 /* cdev 结构体 */
    43
         struct cdev cdev;
    44
        struct class *class;
                                 /* 类 */
    45 struct device *device;
                                /* 设备 */
    46 int key_gpio;
                                 /* GPIO 编号 */
    47 int irq_num;
                                 /* 中断号 */
                                /* 定时器 */
        struct timer list timer;
    48
         wait_queue_head_t r_wait; /* 读等待队列头 */
    49
    50 };
    51
                                 /* 按键设备 */
    52 static struct key_dev key;
    53 static atomic_t status;
                                 /* 按键状态 */
    .....
    75 static ssize_t key_read(struct file *filp, char __user *buf,
    76
             size_t cnt, loff_t *offt)
    77 {
    78
        int ret;
    79
    80 /* 加入等待队列,当有按键按下或松开动作发生时,才会被唤醒*/
    81
         ret = wait_event_interruptible(key.r_wait, KEY_KEEP != atomic_read(&status));
    82
         if (ret)
    83
           return ret;
    84
        /* 将按键状态信息发送给应用程序 */
    85
    86
         ret = copy_to_user(buf, &status, sizeof(int));
    87
    88
         /* 状态重置 */
    89
         atomic_set(&status, KEY_KEEP);
    90
    91
         return ret;
    92 }
    . . . . . .
    118 static void key_timer_function(unsigned long arg)
    119 {
    120
         static int last_val = 1;
    121
         int current_val;
    122
    123 /* 读取按键值并判断按键当前状态 */
    124
         current_val = gpio_get_value(key.key_gpio);
    125 if (0 == current_val \&\& last_val) {
                                                  // 按下
    126
           atomic_set(&status, KEY_PRESS);
                                               // 唤醒 r wait 队列头中的所有队列
    127
           wake_up_interruptible(&key.r_wait);
```

```
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
    128
        }
    129
         else if (1 == current_val && !last_val) {
                                                  // 松开
            atomic_set(&status, KEY_RELEASE);
    130
                                                  // 唤醒 r_wait 队列头中的所有队列
    131
            wake_up_interruptible(&key.r_wait);
    132
         }
    133
         else
    134
                                                  // 状态保持
            atomic_set(&status, KEY_KEEP);
    135
    136
         last_val = current_val;
    137 }
    .....
    228 static int __init mykey_init(void)
    229 {
        .....
    232 /* 初始化等待队列头 */
    233 init_waitqueue_head(&key.r_wait);
        .....
    273 /* 初始化按键状态 */
    274 atomic_set(&status, KEY_KEEP);
    275
    276 /* 初始化定时器 */
    277
         init_timer(&key.timer);
         key.timer.function = key_timer_function;
    278
    279
    280
         return 0;
    281
    282 out4:
    283
         class_destroy(key.class);
    284
    285 out3:
    286
         cdev_del(&key.cdev);
    287
    288 out2:
    289
         unregister_chrdev_region(key.devid, KEY_CNT);
    290
    291 out1:
    292
         free_irq(key.irq_num, NULL);
    293
         gpio_free(key.key_gpio);
    294
    295
         return ret;
    }
```

正点原子

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

第 41~50 行,我们删除了设备结构体 struct key_dev 中的自旋锁变量,本章驱动代码我们 不用自旋锁,而改用原子变量来实现对相应变量的保护操作。第 49 行,添加了一个等待队列 头 r_wait,因为在 Linux 驱动中处理阻塞 IO 需要用到等待队列,因为等待队列会使进程进入 休眠状态,所以不能使用自旋锁!

正点原子

第53行,定义了一个原子变量 status,可以实现对该变量的原子操作,保护数据。

第 75~92 行, key_read 函数, 在这个函数中我们会判断按键是否有按下或松开动作发生时, 如果没有则调用 wait_event_interruptible 把它加入等待队列当中,进行阻塞, 如果等待队 列被唤醒并且条件 "KEY_KEEP != atomic_read(&status)"成立则会解除阻塞, 继续下面的操 作, 也就是读取按键状态数据将其发送给应用程序, 因为采用了 wait_event_interruptible 函数, 因此进入休眠态的进程可以被信号打断; 在该函数中我们读取 status 原子变量必须要使用 atomic_read 函数进行操作。

第 118~137 行,定时器定时处理函数 key_timer_function,对按键状态进行判断,如果是 按下动作或是松开动作则会使用 wake_up_interruptible 函数唤醒等待队列,并且将 status 变量 设置为 KEY_PRESS 或 KEY_RELEASE;这样在 key_read 函数中阻塞的进程就会解除阻塞继 续进行下面的操作。

第 233 行,在驱动入口函数中我们会调用 init_waitqueue_head 初始化等待队列头;第 274 行使用原子操作 atomic_set 设置按键的初始状态 status 为 KEY_KEEP。

使用等待队列实现阻塞访问重点注意两点:

①、将任务或者进程加入到等待队列头,

②、在合适的点唤醒等待队列,一般是中断处理函数里面。

2、编写测试 APP

本节实验的测试 APP 直接使用 13_irq 实验目录下的 keyApp.c 测试程序,将 keyApp.c 复制到本实验目录 14_blockio 中,不需要修改任何内容。

34.2.3 运行测试

1、编译驱动程序和测试 APP

①、以编译驱动程序

编写 Makefile 文件,将 13_irq 实验目录下的 Makefile 文件拷贝到本实验目录下,打开该 Makefile 文件,将 obj-m 变量的值改为 blockio.o, Makefile 内容如下所示:

示例代码 34.2.2 Makefile 文件

```
1 KERN_DIR :=/home/zy/workspace/kernel-driver/linux-xlnx-xlnx_rebase_v5.4_2020.2
```

```
2
3 obj-m :=blockio.o
4
```

5 all:

```
6 make -C $(KERN_DIR) M=`pwd` modules
```

```
7 clean:
```

```
8 make -C $(KERN_DIR) M=`pwd` clean
```



make[1]: 进入目录"/home/zy/workspace/kernel-driver/linux-xlnx-xlnx rebase v5.4 2 020.2" CC [M] /home/zy/linux/drivers/14_blockio/blockio.o

Building modules, stage 2.

make

wd` modules

MODPOST 1 modules

CC [M] /home/zy/linux/drivers/14 blockio/blockio.mod.o

LD [M] /home/zy/linux/drivers/14 blockio/blockio.ko make[1]: 离开目录"/home/zy/workspace/kernel-driver/linux-xlnx-xlnx rebase v5.4 2

020.2" zy@zy-virtual-machine:~/linux/drivers/14 blockio\$ zy@zy-virtual-machine:~/linux/drivers/14 blockio\$ zy@zy-virtual-machine:~/linux/drivers/14_blockio\$ ls blockio.c blockio.mod blockio.mod.o keyApp.c modules.order

blockio.ko blockio.mod.c blockio.o Makefile Module.symvers zy@zy-virtual-machine:~/linux/drivers/14 blockio\$

图 34.2.2 编译驱动模块

③、编译测试 APP

输入如下命令编译测试 keyApp.c 这个测试程序:

\$CC keyApp.c -o keyApp

编译成功以后就会生成 keyApp 这个应用程序。

2、运行测试

将上面编译出来的 blockio.ko 和 keyApp 这两个文件拷贝到 NFS 共享目录下根文件系统的 /lib/modules/5.4.0-xilinx 文件夹中。

将 SD 启动卡插到开发板中,连接开发板串口、网口和电源线,设置拨码开关为 SD 卡启 动模式,然后打开电源,启动开发板。

系统启动完成后,通过串口终端进入"/lib/modules/5.4.0-xilinx"目录下,输入如下命令 加载 blockio.ko 驱动模块:

//第一次加载驱动的时候需要运行此命令 depmod

modprobe blockio.ko //加载驱动

驱动加载成功以后使用如下命令打开 keyApp 这个测试 APP,并且以后台模式运行:

./keyApp /dev/key &

按下开发板上的 PS KEY0 按键,结果如图 34.2.3 所示:



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
<pre>root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx# depmod</pre>
<pre>root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx# modprobe blockio.ko</pre>
blockio: loading out-of-tree module taints kernel.
root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx# ./keyApp /dev/key &
[1] 521
<pre>root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx# Key Press</pre>
Key Release
Key Press
Key Release
Key Press
Key Release

图 34.2.3 测试 APP 运行测试

当按下或者松开 PS_KEY0 按键动作发生的时候,终端就会打印出相应的字符串信息。输入"top"命令,查看 keyAPP 这个应用 APP 的 CPU 使用率,如所示:

root@ Key Re Key Pi Key Re	ALIENTE elease ress elease ress	EK-ZYN	Q-driver:	:/lib/m	odule	es/5.	4.0->	kilinx# Key Press
Key R	alease							
top	ceuse							
Mem: 2	26464K	used.	484064K	free.	140K	shrd	. 268	3K buff. 7932K cached
CPU:	0.0% ι	ısr 0	.0% svs	0.0% n	ic 1	00%	idle	0.0% io 0.0% ira 0.0% sira
Load a	average	e: 0.0	1 0.08 0.	.04 1/7	5 522	2		
PID	PPID	USER	STAT	VSZ	%VSZ	CPU	%CPU	COMMAND
522	511	root	R	2304	0.4	1	0.0	top
490	1	root	S	13020	2.5	1	0.0	/usr/sbin/tcf-agent -d -Ll0
461	1	root	S	7224	1.4	1	0.0	/usr/sbin/haveged -w 1024 -v 1
510	495	root	S	3144	0.6	0	0.0	/bin/login
511	510	root	S	2616	0.5	1	0.0	-sh
81	1	root	S	2576	0.5	1	0.0	/sbin/udevd -d
495	1	root	S	2400	0.4	0	0.0	<pre>{start getty} /bin/sh /bin/start getty 115</pre>
479	1	root	S	2304	0.4	1	0.0	/sbin/klogd -n
472	1	root	S	2304	0.4	0	0.0	/usr/sbin/inetd
476	1	root	S	2304	0.4	1	0.0	/sbin/syslogd -n -0 /var/log/messages
496	1	root	S	2304	0.4	0	0.0	/sbin/getty 38400 tty1
465	1	root	S	1952	0.3	0	0.0	/usr/sbin/dropbear -r /etc/dropbear/dropbe
1	0	root	S	1388	0.2	1	0.0	init [5]
521	511	root	S	1368	0.2	1	0.0	./keyApp /dev/key
55	2	root	IW<	0	0.0	Θ	0.0	kworker/u5:3-xpl

图 34.2.4 keyApp 程序 CPU 占用率

从图 34.2.4 可以看出,当我们在按键驱动程序里面加入阻塞访问以后,keyApp 这个应用程序的 CPU 使用率从图 34.2.1 中的 49.9%降低到了 0.0%。大家注意,这里的 0.0%并不是说keyApp 这个应用程序不使用 CPU 了,只是因为使用率太小了,CPU 使用率可能为 0.00001%,但是图 34.2.4 中只能显示出小数点后一位,因此就显示成了 0.0%。

我们可以使用"kill"命令关闭后台运行的应用程序,比如我们关闭掉 keyApp 这个后台运行的应用程序。首先输出"Ctrl+C"关闭 top 命令界面,进入到命令行模式。然后使用"ps"命令查看一下 keyApp 这个应用程序的 PID,如下图所示:



Trestate	
472 root 0:00 /usr/sbin/inetd	
476 root 0:00 /sbin/syslogd -n -0 /var/log/messages	
479 root 0:00 /sbin/klogd -n	
490 root 0:00 /usr/sbin/tcf-agent -d -Ll0	
495 root 0:00 {start_getty} /bin/sh /bin/start_getty 115200 ttyPS0 vt1	.02
496 root 0:00 /sbin/getty 38400 tty1	
510 root 0:00 /bin/login	
_511_root0:00sh	
521 root 0:00 ./keyApp /dev/key	
523 root 0:00 [kworker/1:0-eve]	
524 root 0:00 ps	
root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx#	

图 34.2.5 查看进程 PID

从图 34.2.5 可以看出, keyApp 这个应用程序的 PID 为 521, 使用 "kill -9 PID" 即可"杀死"指定 PID 的进程, 比如我们现在要"杀死" PID 为 521 的 keyApp 应用程序, 可是使用如下命令:

kill -9 521

输入上述命令以后终端显示如图 34.2.6 所示:

521	root	0:00 ./keyApp /dev/key			
523	root	0:00 [kworker/1:0-eve]			
524	root	0:00 ps			
root@/	ALIENTEK-Z	NQ-driver:/lib/modules/5.4.0-xilinx#	kill	-9	521
[1]+	Killed	./keyApp /dev/key			

图 34.2.6 kill 命令输出结果

从图 34.2.6 可以看出, "./keyApp /dev/key"这个应用程序已经被"杀掉"了,在此输入 "ps"命令查看当前系统运行的进程,会发现 keyApp 已经不见了。这个就是使用 kill 命令 "杀掉"指定进程的方法。

34.3 非阻塞 IO 实验

34.3.1 硬件原理图分析

本章实验硬件原理图参考 31.2 小节即可。

34.3.2 实验程序编写

1、驱动程序编写

本 实 验 对 应 的 例 程 路 径 为 : ZYNQ 开 发 板 资 料 盘 (A 盘)\4_SourceCode\3_Embedded_Linux\Linux 驱动例程\15_noblockio。

本章我们将在上一节实验"14_blockio"实验的基础上完成,上一节实验我们已经在驱动中添加了阻塞 IO 的代码,本小节我们继续完善驱动,加入非阻塞 IO 驱动代码。在 drivers 目录下新建名为"15_noblockio"的文件夹,将"14_blockio"实验中的 blockio.c 复制到 15_noblockio 文件夹中,并重命名为 noblockio.c。接下来我们就修改 noblockio.c 这个文件,在其中添加非阻塞相关的代码,完成以后的 noblockio.c 内容如下所示(因为是在上一小节实验的 blockio.c 文件的基础上修改的,为了减少篇幅,下面的代码有省略):

示例代码 34.3.1 noblockio.c 文件(有省略)

2 Copyright © ALIENTEK Co., Ltd. 1998-2029. All rights reserved.



原子	² 哥在线教	(学:www.y	uanzige.com	论坛:www.openedv.com/forum.php	
	3 文件名	: noblockio.	c		
	4 作者	:邓涛			
	5 版本	: V1.0			
	6 描述	:非阻塞 IO	访问驱动实验		
	7 其他	:无			
	8 论坛	: www.opene	dv.com		
	9 日志	:初版 V1.02	2019/1/30 邓涛创建		
	10 *****	*****	***************	***************************************	
	11				
	12 #includ	e <linux td="" types<=""><td>.h></td><td></td><td></td></linux>	.h>		
	13 #includ	e <linux kerne<="" td=""><td>l.h></td><td></td><td></td></linux>	l.h>		
	14 #includ	e <linux delay<="" td=""><td>.h></td><td></td><td></td></linux>	.h>		
	15 #includ	e <linux ide.h<="" td=""><td>></td><td></td><td></td></linux>	>		
	16 #includ	e <linux init.h<="" td=""><td>></td><td></td><td></td></linux>	>		
	17 #includ	e <linux modu<="" td=""><td>lle.h></td><td></td><td></td></linux>	lle.h>		
	18 #includ	e <linux errno<="" td=""><td>.h></td><td></td><td></td></linux>	.h>		
	19 #includ	e <linux gpio.<="" td=""><td>h></td><td></td><td></td></linux>	h>		
	20 #includ	e <asm <="" mach="" td=""><td>map.h></td><td></td><td></td></asm>	map.h>		
	21 #includ	e <asm td="" uacces<=""><td>s.h></td><td></td><td></td></asm>	s.h>		
	22 #includ	e <asm 10.h=""></asm>	L.		
	25 #includ	e < linux/cdev.	n>		
	24 #includ	e < linux/or.n>	ldragg b		
	25 #includ	e < linux/of_ac	via h		
	20 #Includ	e linux/of_jr	n0.11>		
	27 #includ	e linux/or_ira b	1.11.2		
	20 #includ	e <u>linux</u>			
	2) milleruu	c <mux pon.<="" td=""><td></td><td></td><td></td></mux>			
	68 /*				
	69 * @des	scription	:从设备读取数	据	
	70 * @pai	am – filp	:要打开的设备	文件(文件描述符)	
	71 * @pai	am – buf	:返回给用户空	间的数据缓冲区	
	72 * @pai	ram – cnt	:要读取的数据	长度	
	73 * @pai	ram – offt	:相对于文件首	地址的偏移	
	74 * @ret	urn	:读取的字节数	,如果为负值,表示读取失败	
	75 */				
	76 static ss	ize_t key_read	l(struct file *filp, cha	aruser *buf,	
	77	size_t cnt, loff	_t *offt)		
	78 {				
	79 int re	t;			
	80				



```
原子哥在线教学: www.yuanzige.com
                                          论坛:www.openedv.com/forum.php
        if (filp->f_flags & O_NONBLOCK) { // 非阻塞方式访问
    81
    82
          if(KEY_KEEP == atomic_read(&status))
            return -EAGAIN;
    83
    84
       } else {
                                       // 阻塞方式访问
          /* 加入等待队列,当有按键按下或松开动作发生时,才会被唤醒 */
    85
          ret = wait event interruptible(key.r wait, KEY KEEP != atomic read(&status));
    86
    87
          if (ret)
    88
            return ret;
    89
       }
    90
    91
        /* 将按键状态信息发送给应用程序 */
    92
        ret = copy_to_user(buf, &status, sizeof(int));
    93
    94
       /* 状态重置 */
    95
        atomic_set(&status, KEY_KEEP);
    96
    97
        return ret;
    98 }
   •••••
   100 /*
                          : poll 函数,用于处理非阻塞访问
   101 * @description
   102 * @param – filp
                          :要打开的设备文件(文件描述符)
   103 * @param – wait
                          : 等待列表(poll_table)
   104 * @return
                           :设备或者资源状态
   105 */
   106 static unsigned int key_poll(struct file *filp, struct poll_table_struct *wait)
   107 {
         unsigned int mask = 0;
   108
   109
   110
        poll_wait(filp, &key.r_wait, wait);
   111
        if(KEY_KEEP != atomic_read(&status)) // 按键按下或松开动作发生
   112
   113
           mask = POLLIN | POLLRDNORM; // 返回 PLLIN
   114
   115
         return mask;
   116 }
   .....
   245 /* 设备操作函数 */
   246 static struct file_operations key_fops = {
   owner.
                   = THIS_MODULE,
   248 .open
                   = key_open,
```



```
原子哥在线教学: www.yuanzige.com
                                          论坛:www.openedv.com/forum.php
   249
        .read
                   = key_read,
   250 .write
                   = key_write,
   251 .release
                   = key_release,
   252 .poll
                   = key_poll,
   253 };
   .....
   325 static void __exit mykey_exit(void)
   326 {
   327 /* 删除定时器 */
   328
        del_timer_sync(&key.timer);
   329
   330 /* 注销设备 */
   331
        device_destroy(key.class, key.devid);
   332
   333 /* 注销类 */
   334
        class_destroy(key.class);
   335
   336
        /* 删除 cdev */
   337
        cdev_del(&key.cdev);
   338
   339
        /* 注销设备号 */
   340
         unregister_chrdev_region(key.devid, KEY_CNT);
   341
   342 /* 释放中断 */
   343
        free_irq(key.irq_num, NULL);
   344
   345 /* 释放 GPIO */
   346 gpio_free(key.key_gpio);
   347 }
   348
   349 /* 驱动模块入口和出口函数注册 */
   350 module_init(mykey_init);
   351 module_exit(mykey_exit);
   352
   353 MODULE_AUTHOR("DengTao <773904075@qq.com>");
   354 MODULE_DESCRIPTION("Gpio Key Interrupt Driver");
   355 MODULE_LICENSE("GPL");
    第29行,使用 include 将内核源码目录 include/linux/poll.h 头文件包含进来。
```

第 76~98 行, key_read 函数中判断是否为非阻塞式读取访问,如果是的话就判断按键状态是否有效,也就是判断是否产生了按下或松开这样的动作,如果没有的话就返回-EAGAIN。



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

第 106~116 行, key_poll 函数就是 file_operations 驱动操作集中的 poll 函数,当应用程序 调用 select 或者 poll 函数的时候 key_poll 函数就会执行。第 110 行调用 poll_wait 函数将等待队 列头添加到 poll_table 中,第 112~113 行判断按键是否有效,如果按键有效的话就向应用程序 返回 POLLIN 这个事件,表示有数据可以读取。

第 252 行,设置 file_operations 的 poll 成员变量为 key_poll。

2、编写测试 APP

将上一节实验目录 14_blockio 下的 keyApp.c 源文件拷贝到本实验目录下,然后打开 keyApp.c 文件进行修改,修改完成之后内容如下所示:

```
示例代码 34.3.2 keyApp.c 文件代码
2 Copyright © ALIENTEK Co., Ltd. 1998-2029. All rights reserved.
3 文件名
           : keyApp.c
4 作者
           :邓涛
5 版本
          : V1.0
6 描述
          :以非阻塞方式读取按键状态
7 其他
          :无
8 使用方法
         : ./keyApp /dev/key
9 论坛
          : www.openedv.com
10 日志
           :初版 V1.0 2019/1/30 邓涛创建
12
13 #include <stdio.h>
14 #include <unistd.h>
15 #include <sys/types.h>
16 #include <sys/stat.h>
17 #include <fcntl.h>
18 #include <stdlib.h>
19 #include <string.h>
20 #include <poll.h>
21
22 /*
                : main 主程序
23 * @description
24 * @param – argc
               : argv 数组元素个数
25 * @param – argv : 具体参数
26 * @return
                :0 成功;其他 失败
27 */
28 int main(int argc, char *argv[])
29 {
30 fd_set readfds;
31 int key_val;
```

32 int fd;



```
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
    33
        int ret;
    34
    35
        /* 判断传参个数是否正确 */
    36
        if(2 != argc) {
    37
           printf("Usage:\n"
    38
              "\t./keyApp /dev/key\n"
    39
              );
    40
           return -1;
    41
        }
    42
    43 /* 打开设备 */
    44
        fd = open(argv[1], O_RDONLY | O_NONBLOCK);
    45
        if(0 > fd) {
           printf("ERROR: %s file open failed!\n", argv[1]);
    46
    47
           return -1;
    48
        }
    49
    50
        FD_ZERO(&readfds);
    51
        FD_SET(fd, &readfds);
    52
    53
        /* 循环轮询读取按键数据 */
    54
         for(;;){
    55
    56
           ret = select(fd + 1, &readfds, NULL, NULL);
    57
           switch (ret) {
    58
    59
                       // 超时
           case 0:
             /* 用户自定义超时处理 */
    60
    61
             break;
    62
    63
           case -1:
                        // 错误
             /* 用户自定义错误处理 */
    64
    65
             break;
    66
    67
           default:
    68
             if(FD_ISSET(fd, &readfds)) {
    69
               read(fd, &key_val, sizeof(int));
    70
               if (0 == key_val)
    71
                 printf("Key Press\n");
    72
               else if (1 == key_val)
    73
                 printf("Key Release\n");
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

74	}
75	
76	break;
77	}
78	}
79	
80	/* 关闭设备 */
81	close(fd);
82	return 0;
83 }	

第20行,添加 poll.h 头文件。

第30行,定义 readfds 变量。

第 54~78 行,在本测试程序中我们使用 select 函数来实现非阻塞访问,在 for 循环中使用 select 函数不断的轮询,检查驱动程序是否有数据可以读取,如果可以读取的话就调用 read 函数读取按键数据。大家也可以试试使用 poll 函数来实现!

34.3.3 运行测试

1、编译驱动程序和测试 APP

①编译驱动程序

编写 Makefile 文件,将上一节实验目录 14_blockio 下的 Makefile 文件拷贝到本实验目录下,打开 Makefile 文件,将 obj-m 变量的值改为 noblockio.o, Makefile 内容如下所示:

示例代码 34.3.3 Makefile 文件

1 KERN_DIR :=/home/zy/workspace/kernel-driver/linux-xlnx_rebase_v5.4_2020.2

2

3 obj-m :=noblockio.o

4

5 all:

6 make -C \$(KERN_DIR) M=`pwd` modules

7 clean:

8 make -C \$(KERN_DIR) M=`pwd` clean

第3行,设置 obj-m 变量的值为 noblockio.o。

文件修改完成之后保存退出,输入如下命令编译出驱动模块文件:

make

编译成功以后就会生成一个名为"noblockio.ko"的驱动模块文件。

②编译测试 APP

输入如下命令编译测试 keyApp.c 这个测试程序:

\$CC keyApp.c -o keyApp

编译成功以后就会生成 keyApp 这个应用程序。

2、运行测试



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php 将上面编译出来的 noblockio.ko 和 keyApp 这两个文件拷贝到 NFS 共享目录下根文件系统 的/lib/modules/5.4.0-xilinx 文件夹中。 将 SD 启动卡插到开发板中,连接开发板串口、网口和电源线,设置拨码开关为 SD 卡启 动模式,然后打开电源,启动开发板。 系统启动完成后,通过串口终端进入"/lib/modules/5.4.0-xilinx"目录下,输入如下命令 加载 noblockio.ko 驱动模块: //第一次加载驱动的时候需要运行此命令 depmod modprobe noblockio.ko//加载驱动 驱动加载成功以后使用如下命令打开 keyApp 这个测试 APP,并且以后台模式运行: ./keyApp /dev/key & 按下开发板上的 PS_KEY0 按键,结果如图 34.3.1 所示: root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx# ./keyApp /dev/key & [1] 545 root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx# Key Press Key Release Key Press Key Release

图 34.3.1 测试 APP 运行测试

当产生按下按键或松开按键动作的时候,终端就会打印出相应的字符串信息。输入"top" 命令,查看 keyAPP 这个应用 APP 的 CPU 使用率,如所示:

Kev I	ress							
Kev F	Release							
top								
Mem	26336K	used	484192K	free	140K	shre	1 268	3K buff 7976K cached
	0.0%	isr 4	7% svs	0 0% r	ic Q	5 2%	idle	A A% in A A% ing A A% sing
	average	- 0 0	0 0 00 0	0.001/6	A 54	7	fuce	0.00 10 0.00 114 0.00 3114
PT		USER	STAT	VS7	%VS7	CPIL	%CPI1	COMMAND
496	1	root	S	13020	2 5	0	0 0	/usr/shin/tcf-agent -d -Ll0
461	, <u>1</u>	root	S	7224	1.4	1	0.0	/usr/sbin/haveged $-w$ 1024 $-v$ 1
510	495	root	Š	3144	0.6	ō	0.0	/bin/login
511	510	root	Š	2720	0.5	1	0.0	-sh
81	1	root	Š	2576	0.5	1	0.0	/sbin/udevd -d
495	5 1	root	ŝ	2400	0.4	õ	0.0	{start getty} /bin/sh /bin/start getty 115
479	9 1	root	s	2304	0.4	1	0.0	/sbin/kload -n
547	511	root	R	2304	0.4	1	0.0	top
472	2 1	root	S	2304	0.4	0	0.0	/usr/sbin/inetd
476	5 I	root	S	2304	0.4	1	0.0	/sbin/syslogd -n -0 /var/log/messages
496	5 1	root	S	2304	0.4	0	0.0	/sbin/getty 38400 tty1
465	5 1	root	S	1952	0.3	0	0.0	/usr/sbin/dropbear -r /etc/dropbear/dropbe
1	L 0	root	S	1388	0.2	1	0.0	init [5]
545	5 511	root	S	1368	0.2	1	0.0	./keyApp /dev/key
55) 2	root	1W<	0	0.0	Θ	0.0	[kworker/u5:3-xp]

图 34.3.2 应用程序 CPU 使用率



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

从图 34.3.2 可以看出,采用非阻塞方式读处理以后,keyApp 的 CPU 占用率也低至 0.0%,和图 34.2.4 中的一样,这里的 0.0%并不是说 keyApp 这个应用程序不使用 CPU 了,只是因为使用率太小了,而图中只能显示出小数点后一位,因此就显示成了 0.0%。

如果要"杀掉"处于后台运行模式的 keyApp 这个应用程序,可以参考 34.2.3 小节讲解的方法。



正点原子

第三十五章 异步通知实验

在前面使用阻塞或者非阻塞的方式来读取驱动中按键值都是应用程序主动读取的,对于 非阻塞方式来说还需要应用程序通过 poll 函数不断的轮询。最好的方式就是驱动程序能主动 向应用程序发出通知,报告自己可以访问,然后应用程序在从驱动程序中读取或写入数据, 类似于我们在 SDK 例程中讲解的中断。Linux 提供了异步通知这个机制来完成此功能,本章 我们就来学习一下异步通知以及如何在驱动中添加异步通知相关处理代码。



正点原子

35.1 异步通知

35.1.1 异步通知简介

我们首先来回顾一下"中断",中断是处理器提供的一种异步机制,我们配置好中断以 后就可以让处理器去处理其他的事情了,当中断发生以后会触发我们事先设置好的中断服务 函数,在中断服务函数中做具体的处理。比如我们在《领航者 ZYNO 之嵌入式开发指南》第 四章中讲解的 GPIO 按键中断实验,我们不再循环读取 GPIO 数据获取按键状态,而是采用中 断方式,采用中断方式以后处理器就不需要时刻的去查看按键有没有被按下,因为按键按下 以后会自动触发中断。同样的, Linux 应用程序可以通过阻塞或者非阻塞这两种方式来访问驱 动设备,通过阻塞方式访问的话应用程序会处于休眠态,等待驱动设备可以使用,非阻塞方 式的话会通过 poll 函数来不断的轮询,查看驱动设备文件是否可以使用。这两种方式都需要 应用程序主动的去查询设备的使用情况,如果能提供一种类似中断的机制,当驱动程序可以 访问的时候主动告诉应用程序那就最好了。

"信号"为此应运而生,信号类似于我们硬件上使用的"中断",只不过信号是软件层 次上的。算是在软件层次上对中断的一种模拟,驱动可以通过主动向应用程序发送信号的方 式来报告自己可以访问了,应用程序获取到信号以后就可以从驱动设备中读取或者写入数据 了。整个过程就相当于应用程序收到了驱动发送过来了的一个中断,然后应用程序去响应这 个中断,在整个处理过程中应用程序并没有去查询驱动设备是否可以访问,一切都是由驱动 设备自己告诉给应用程序的。

阻塞、非阻塞、异步通知,这三种是针对不同的场合提出来的不同的解决方法,没有优 劣之分,在实际的工作和学习中,根据自己的实际需求选择合适的处理方法即可。

异步通知的核心就是信号,在include/uapi/asm-generic/signal.h头文件中定义了Linux所支 持的信号,这些信号如下所示:

		示例代码 35.1.1 Linux 信号
34 #define SIGHUP	1	/*终端挂起或控制进程终止 */
35 #define SIGINT	2	/*终端中断(Ctrl+C组合键)*/
36 #define SIGQUIT	3	/* 终端退出(Ctrl+\组合键) */
37 #define SIGILL	4	/* 非法指令 */
38 #define SIGTRAP	5	/* debug 使用,有断点指令产生 */
39 #define SIGABRT	6	/* 由 abort(3)发出的退出指令 */
40 #define SIGIOT	б	/* IOT 指令 */
41 #define SIGBUS	7	/* 总线错误 */
42 #define SIGFPE	8	/* 浮点运算错误 */
43 #define SIGKILL	9	/* 杀死、终止进程 */
44 #define SIGUSR1	10	/* 用户自定义信号 1 */
45 #define SIGSEGV	11	/* 段违例(无效的内存段) */
46 #define SIGUSR2	12	/* 用户自定义信号 2 */
47 #define SIGPIPE	13	/* 向非读管道写入数据 */
48 #define SIGALRM	14	/* 闹钟 */
49 #define SIGTERM	15	/* 软件终止 */
50 #define SIGSTKFL1	Г <u>16</u>	/* 栈异常 */

原子哥在线教学:www	.yuanzig	e.com 论坛:www.openedv.com/forum.php
51 #define SIGCHLD	17	/* 子进程结束 */
52 #define SIGCONT	18	/* 进程继续 */
53 #define SIGSTOP	19	/* 停止进程的执行,只是暂停 */
54 #define SIGTSTP	20	/*停止进程的运行(Ctrl+Z组合键)*/
55 #define SIGTTIN	21	/* 后台进程需要从终端读取数据 */
56 #define SIGTTOU	22	/* 后台进程需要向终端写数据 */
57 #define SIGURG	23	/* 有"紧急"数据 */
58 #define SIGXCPU	24	/* 超过 CPU 资源限制 */
59 #define SIGXFSZ	25	/* 文件大小超额 */
60 #define SIGVTALR	M 26	/* 虚拟时钟信号 */
61 #define SIGPROF	27	/* 时钟信号描述 */
62 #define SIGWINCH	28	/* 窗口大小改变 */
63 #define SIGIO	29	/* 可以进行输入/输出操作 */
64 #define SIGPOLL	SIGIO	
65 /* #define SIGLOS	29 */	
66 #define SIGPWR	30	/* 断点重启 */
67 #define SIGSYS	31	/* 非法的系统调用 */
68 #define SIGUNUSE	D 31	/* 未使用信号 */

2 正点原子

在示例代码 35.1.1 的这些信号中,除了 SIGKILL(9)和 SIGSTOP(19)这两个信号不能被忽略外,其他的信号都可以忽略。这些信号就相当于中断号,不同的中断号代表了不同的中断,不同的中断所做的处理不同,因此,驱动程序可以通过向应用程序发送不同的信号来实现不同的功能。

我们使用中断的时候需要设置中断处理函数,同样的,如果要在应用程序中使用信号, 那么就必须设置信号所使用的信号处理函数,在应用程序中使用 signal 函数来设置指定信号 的处理函数, signal 函数原型如下所示:

sighandler_t signal(int signum, sighandler_t handler)

函数参数和返回值含义如下:

signum: 要设置处理函数的信号。

handler: 信号的处理函数。

返回值:设置成功的话返回信号的前一个处理函数,设置失败的话返回 SIG_ERR。

信号处理函数原型如下所示:

typedef void (*sighandler_t)(int)

我们前面讲解的使用"kill -9 PID"杀死指定进程的方法就是向指定的进程(PID)发送 SIGKILL 这个信号。当按下键盘上的 CTRL+C 组合键以后会向当前正在占用终端的应用程序 发出 SIGINT 信号, SIGINT 信号默认的动作是关闭当前应用程序。这里我们修改一下 SIGINT 信号的默认处理函数,当按下 CTRL+C 组合键以后先在终端上打印出"SIGINT signal!"这 行字符串,然后再关闭当前应用程序。在 Ubuntu 中新建一个 signaltest.c 文件,然后输入如下 所示内容:

示例代码 35.1.2 信号测试

1 #include <stdlib.h>

2 #include <stdio.h>



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

```
3 #include <signal.h>
    4
    5 void sigint_handler(int num)
    6 {
       printf("\r\nSIGINT signal!\r\n");
    7
    8
       exit(0);
    9}
    10
    11 int main(void)
    12 {
    13 signal(SIGINT, sigint_handler);
    14 while(1);
    15 return 0;
    16 }
   在示例代码 35.1.2 中我们设置 SIGINT 信号的处理函数为 sigint handler, 当按下 CTRL+C
向 signaltest 发送 SIGINT 信号以后 sigint_handler 函数就会执行,此函数先输出一行"SIGINT
```

signal!"字符串,然后调用 exit 函数关闭 signaltest 应用程序。

使用如下命令编译 signaltest.c:

gcc signaltest.c -o signaltest

在 Ubuntu 中执行"./signaltest"命令运行 signaltest 这个应用程序, 然后按下键盘上的 CTRL+C 组合键, 结果如下图所示:

```
zy@zy-virtual-machine:~/linux$ gcc signaltest.c -o signaltest
zy@zy-virtual-machine:~/linux$
zy@zy-virtual-machine:~/linux$ ls
drivers signaltest signaltest.c
zy@zy-virtual-machine:~/linux$
zy@zy-virtual-machine:~/linux$ ./signaltest
^C
SIGINT signal!
zy@zy-virtual-machine:~/linux$
```

图 35.1.1 signaltest 软件运行结果

从图 35.1.1 可以看出,当按下 CTRL+C 组合键以后 sigint_handler 这个 SIGINT 信号处理 函数执行了,并且输出了 "SIGINT signal!"这行字符串。

35.1.2 驱动中的信号处理

1、fasync_struct 结构体

首先我们需要在驱动程序中定义一个 fasync_struct 结构体指针变量,该结构体定义在内核 源码目录 include/linux/fs.h 头文件中, fasync_struct 结构体内容如下:

示例代码 35.1.3 fasync_struct 发结构体

1247 struct fasync_struct { 1248 spinlock_t fa_lock;

	spinioti_t	14_10 till,
1249	int	magic;
1250	int	fa_fd;



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

1251 struct fasync_struct *fa_next; /* singly linked list */
1252 struct file *fa file;

1253 struct rcu_head fa_rcu;

1254 };

一般将 fasync_struct 结构体指针变量定义到设备结构体中,比如在上一章节驱动实验 key_dev 结构体中添加一个 fasync_struct 结构体指针变量,结果如下所示:

示例代码 35.1.4 在设备结构体中添加 fasync_struct 类型指针变量

41 /* 按键设备结构体 */

42 st	truct key_dev {	
43	dev_t devid;	/* 设备号 */
44	struct cdev cdev;	/* cdev 结构体 */
45	struct class *class;	/* 类 */
46	struct device *device;	/* 设备 */
47	int key_gpio;	/* GPIO 编号 */
48	int irq_num;	/* 中断号 */
49	struct timer_list timer;	/* 定时器 */
50	<pre>wait_queue_head_t r_wait;</pre>	/* 读等待队列头 */
51	<pre>struct fasync_struct *async_queue;</pre>	/* 异步相关结构体 *
521		

第 51 行就是在 key_dev 中添加了一个 fasync_struct 结构体指针变量。

2、fasync 函数

如果要使用异步通知,需要在设备驱动中实现 file_operations 操作集中的 fasync 函数,此 函数格式如下所示:

int (*fasync) (int fd, struct file *filp, int on)

fasync 函数里面一般通过调用 fasync_helper 函数来初始化前面定义的 fasync_struct 结构体指针, fasync_helper 函数原型如下:

int fasync_helper(int fd, struct file * filp, int on, struct fasync_struct **fapp)

fasync_helper 函数的前三个参数就是 fasync 函数的那三个参数,第四个参数就是要初始 化的 fasync_struct 结构体指针变量。当应用程序通过"fcntl(fd, F_SETFL, flags | FASYNC)"改变 fasync 标记的时候,驱动程序 file_operations 操作集中的 fasync 函数就会执行。

驱动程序中的 fasync 函数参考示例如下:

```
示例代码 35.1.5 驱动中 fasync 函数参考示例

1 struct xxx_dev {

2 .....

3 struct fasync_struct *async_queue; /* 异步相关结构体 */

4 };

5

6 static int xxx_fasync(int fd, struct file *filp, int on)

7 {

8 struct xxx_dev *dev = (xxx_dev)filp->private_data;

9
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php if (fasync_helper(fd, filp, on, &dev->async_queue) < 0) 10 11 return -EIO; return 0; 12 13 } 14 15 static struct file operations $xxx ops = {$ 16 17 $fasync = xxx_fasync$, 18 19 };

在关闭驱动文件的时候需要在 file_operations 操作集中的 release 函数中释放 fasync_struct, fasync_struct 的释放函数同样为 fasync_helper, release 函数参数参考实例如下:

示例代码 35.1.6 释放 fasync_struct 参考示例

```
1 static int xxx_release(struct inode *inode, struct file *filp)
2 {
```

3 return xxx_fasync(-1, filp, 0); /* 删除异步通知 */

4 }

```
5
```

```
6 static struct file_operations xxx_ops = {
```

7

```
8 .release = xxx_release,
```

<mark>9</mark>};

第3行通过调用示例代码35.1.5中的xxx_fasync函数来完成fasync_struct的释放工作,但是,其最终还是通过fasync_helper函数完成释放工作。

3、kill_fasync 函数

当设备可以访问的时候,驱动程序需要向应用程序发出信号,相当于产生"中断"。 kill_fasync函数负责发送指定的信号,kill_fasync函数原型如下所示:

void kill_fasync(struct fasync_struct **fp, int sig, int band)

函数参数和返回值含义如下:

fp:要操作的 fasync_struct。
sig:要发送的信号。
band:可读时设置为 POLL_IN,可写时设置为 POLL_OUT。
返回值:无。

35.1.3 应用程序对异步通知的处理

应用程序对异步通知的处理包括以下三步:

1、注册信号处理函数

应用程序根据驱动程序所使用的信号来设置信号的处理函数,应用程序使用 signal 函数 来设置信号的处理函数。前面已经详细的讲过了,这里就不细讲了。

2、将本应用程序的进程号告诉给内核



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

使用 fcntl(fd, F_SETOWN, getpid())将本应用程序的进程号告诉给内核。

3、开启异步通知

使用如下两行程序开启异步通知:

fcntl(fd, F_SETFL, flags | FASYNC); /* 开启当前进程异步通知功能 */

重点就是通过 fcntl 函数设置进程状态为 FASYNC,经过这一步,驱动程序中的 fasync 函数就会执行。

35.2 硬件原理图分析

本章实验硬件原理图参考 31.2 小节即可。

35.3 实验程序编写

本 实 验 对 应 的 例 程 路 径 为 : ZYNQ 开 发 板 资 料 盘 (A 盘)\4_SourceCode\3_Embedded_Linux\Linux 驱动例程\16_asyncnoti。

本章实验我们在上一章实验"15_noblockio"的基础上完成,在其中加入异步通知相关内容即可,当按键按下以后驱动程序向应用程序发送 SIGIO 信号,应用程序获取到 SIGIO 信号 以后读取按键数据并打印出相应的字符串信息。

35.3.1 修改设备树文件

因为是在实验"15_noblockio"的基础上完成的,因此不需要修改设备树。

35.3.2 实验程序编写

在 drivers 目录下新建名为"16_asyncnoti"的文件夹,将"15_noblockio"实验中的 noblockio.c 拷贝到 16_asyncnoti 文件夹中,并重命名为 asyncnoti.c。接下来我们就修改 asyncnoti.c 这个文件,在其中添加异步通知关的代码,完成以后的 asyncnoti.c 内容如下所示 (因为是在上一章实验的 noblockio.c 文件的基础上修改的,因为了减少篇幅,下面的代码有省 略):

12 #include <linux/types.h>



原子哥在	E线教学: www.yuanzige.com	n 论坛:www.openedv.com/forum.php
13 #	include <linux kernel.h=""></linux>	
14 #	include <linux delay.h=""></linux>	
15 #	include <linux ide.h=""></linux>	
<mark>16</mark> #	include <linux init.h=""></linux>	
17 #	include <linux module.h=""></linux>	
18 #	include <linux errno.h=""></linux>	
<mark>19</mark> #	include <linux gpio.h=""></linux>	
20 #	include <asm mach="" map.h=""></asm>	
21 #	include <asm uaccess.h=""></asm>	
22 #	include <asm io.h=""></asm>	
23 #	include <linux cdev.h=""></linux>	
24 #	include <linux of.h=""></linux>	
25 #	include <linux of_address.h=""></linux>	
<mark>26</mark> #	include <linux of_gpio.h=""></linux>	
27 #	include <linux of_irq.h=""></linux>	
28 #	include <linux irq.h=""></linux>	
<mark>29</mark> #	include <linux poll.h=""></linux>	
<mark>30</mark> #	include <linux fcntl.h=""></linux>	
31		
32 #	define KEY_CNT 1	/* 设备号个数 */
<mark>33</mark> #	define KEY_NAME "key"	/* 名字 */
34		
35 /*	* 定义按键状态 */	
36 e	num key_status {	
37	KEY_PRESS = 0, // 按键按 ⁻	下
38	KEY_RELEASE, // 按键松子	Ŧ
39	KEY_KEEP, // 按键状系	态保持
40 }	;	
41		
42 /*	* 按键设备结构体 */	
43 s	truct key_dev {	
44	dev_t devid;	/* 设备号 */
45	struct cdev cdev;	/* cdev 结构体 */
46	struct class *class;	/* 类 */
47	struct device *device;	/* 设备 */
48	int key_gpio;	/* GPIO 编号 */
49	int irq_num;	/* 中断号 */
50	<pre>struct timer_list timer;</pre>	/* 定时器 */
51	<pre>wait_queue_head_t r_wait;</pre>	/* 读等待队列头 */
52	<pre>struct fasync_struct *async_queue</pre>	; /* fasync_struct 结构体 */
53 }	•	



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php 134 /* 135 * @description : fasync 函数,用于处理异步通知 136 * @param – fd : 文件描述符 137 * @param – filp :要打开的设备文件(文件描述符) 138 * @param – on :模式 139 * @return :负数表示函数执行失败 140 */ 141 static int key_fasync(int fd, struct file *filp, int on) 142 { 143 **return** fasync_helper(fd, filp, on, &key.async_queue); 144 } 145 146 /* 147 * @description :关闭/释放设备 148 * @param – filp :要关闭的设备文件(文件描述符) 149 * @return :0 成功;其他 失败 150 */ 151 static int key_release(struct inode *inode, struct file *filp) 152 { 153 return key_fasync(-1, filp, 0); 154 } 155 156 static void key_timer_function(unsigned long arg) 157 { 158 static int last_val = 1; 159 int current_val; 160 /* 读取按键值并判断按键当前状态 */ 161 162 current_val = gpio_get_value(key.key_gpio); **if** (0 == current_val && last_val) { // 按下 163 atomic_set(&status, KEY_PRESS); 164 165 wake_up_interruptible(&key.r_wait); if(key.async_queue) 166 167 kill_fasync(&key.async_queue, SIGIO, POLL_IN); 168 } 169 else if (1 == current_val && !last_val) { // 松开 170 atomic_set(&status, KEY_RELEASE); 171 wake_up_interruptible(&key.r_wait); 172 if(key.async_queue)

kill_fasync(&key.async_queue, SIGIO, POLL_IN);



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php 174 } 175 else // 状态保持 176 atomic_set(&status, KEY_KEEP); 177 178 last val = current val; 179 } 261 /* 设备操作函数 */ 262 static struct file_operations key_fops = { = THIS MODULE, 263 .owner 264 .open = key_open, = key_read, **265** .read 266 .write = key_write, = key_release, 267 .release 268 .poll = key_poll, 269 .fasync = key_fasync, 270 }; 366 /* 驱动模块入口和出口函数注册 */ 367 module_init(mykey_init); 368 module_exit(mykey_exit); 369 370 MODULE_AUTHOR("DengTao <773904075@qq.com>"); 371 MODULE_DESCRIPTION("Gpio Key Interrupt Driver"); 372 MODULE LICENSE("GPL"); 第30行,添加fcntl.h头文件,因为要用到相关的API函数。 第52行,在设备结构体 key dev 中添加 fasync struct 指针变量。

第 141~144 行,设备操作函数集 file_operations 结构体中的 fasync 函数 key_fasync,该函数中直接调用 fasync_helper 函数进行相关处理。

第153行,在key_release函数中也调用key_fasync函数释放fasync_struct指针变量。

第 156~175 行,在 key_timer_function 函数中,当按键按下或松开动作发生时调用 kill_fasync 函数向应用程序发送 SIGIO 信号,通知应用程序按键数据可以进行读取了。

第269行,将key_fasync函数绑定到key_fops变量的fasync函数指针中。

35.3.3 编写测试 APP

测试 APP 要实现的内容很简单,设置 SIGIO 信号的处理函数为 sigio_signal_func,当驱动程序向应用程序发送 SIGIO 信号以后 sigio_signal_func 函数就会执行。sigio_signal_func 函数内容很简单,就是通过 read 函数读取按键状态数据并打印信息。在 16_asyncnoti 目录下新建 名为 asyncKeyApp.c 的文件,然后输入如下所示内容:

示例代码 35.3.2 asyncKeyApp.c 文件代码段



```
原子哥在线教学: www.yuanzige.com
                                         论坛:www.openedv.com/forum.php
    2 Copyright © ALIENTEK Co., Ltd. 1998-2029. All rights reserved.
    3 文件名
                   : asyncKeyApp.c
    4 作者
                   :邓涛
    5 版本
                   : V1.0
    6 描述
                   :异步通知测试程序
    7 其他
                   :无
    8 使用方法
                   : ./asyncKeyApp /dev/key
    9 论坛
                   : www.openedv.com
    10 日志
                   :初版 V1.0 2019/1/30 邓涛创建
    12
    13 #include <stdio.h>
    14 #include <unistd.h>
    15 #include <sys/types.h>
    16 #include <sys/stat.h>
    17 #include <fcntl.h>
    18 #include <stdlib.h>
    19 #include <string.h>
    20 #include <signal.h>
    21
    22 static int fd;
    23
    24 /*
    25 * SIGIO 信号处理函数
                         :信号值
    26 * @param – signum
                          :无
    27 * @return
    28 */
    29 static void sigio_signal_func(int signum)
    30 {
    31
        unsigned int key_val = 0;
    32
    33
        read(fd, &key_val, sizeof(unsigned int));
    34
        if (0 == key_val)
    35
          printf("Key Press\n");
    36
        else if (1 == key_val)
    37
          printf("Key Release\n");
    38 }
    39
    40
    41 /*
    42 * @description
                    : main 主程序
```



```
原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php
    43 * @param – argc
                         : argv 数组元素个数
   44 * @param – argv
                          :具体参数
   45 * @return
                          :0 成功;其他失败
    46 */
    47 int main(int argc, char *argv[])
    48 {
    49
        int flags = 0;
    50
        /* 判断传参个数是否正确 */
    51
    52
       if(2 != argc) {
    53
          printf("Usage:\n"
    54
             "\t./asyncKeyApp /dev/key\n"
    55
             );
    56
          return -1;
    57
       }
    58
    59 /* 打开设备 */
        fd = open(argv[1], O_RDONLY | O_NONBLOCK);
    60
    61
        if(0 > fd) \{
    62
          printf("ERROR: %s file open failed!\n", argv[1]);
    63
          return -1;
    64 }
    65
    66 /* 设置信号 SIGIO 的处理函数 */
    67
        signal(SIGIO, sigio_signal_func);
                                         // 将当前进程的进程号告诉给内核
    68
        fcntl(fd, F_SETOWN, getpid());
                                         // 获取当前的进程状态
        flags = fcntl(fd, F_GETFD);
    69
        fcntl(fd, F_SETFL, flags | FASYNC);
                                         // 设置进程启用异步通知功能
    70
    71
    72
    73
       /* 循环轮询读取按键数据 */
    74 for (;;) {
    75
    76
          sleep(2);
    77 }
    78
    79 /* 关闭设备 */
    80
       close(fd);
    81
        return 0;
    82 }
```



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

第 29~38 行, sigio_signal_func 函数, SIGIO 信号的处理函数, 当驱动程序有效按键按下 以后就会发送 SIGIO 信号, 此函数就会执行。此函数通过 read 函数读取按键状态数据, 然后 通过 printf 函数打印在终端上。

第67行,通过 signal 函数设置 SIGIO 信号的处理函数为 sigio_signal_func。

第68~70行,设置当前进程的状态,开启异步通知的功能。

第74~77行, for循环,等待信号产生。

35.4 运行测试

35.4.1 编译驱动程序和测试 APP

1、编译驱动程序

编写 Makefile 文件,将 15_noblockio 实验目录下的 Makefile 文件拷贝本实验目录下,打 开 Makefile 文件,将 obj-m 变量的值改为 asyncnoti.o, Makefile 内容如下所示:

示例代码 35.4.1 Makefile 文件

1 KERN_DIR :=/home/zy/workspace/kernel-driver/linux-xlnx-xlnx_rebase_v5.4_2020.2

2

3 obj-m := asyncnoti.o

4

5 all:

6 make -C \$(KERN_DIR) M=`pwd` modules

7 clean:

8 make -C \$(KERN_DIR) M=`pwd` clean

第3行,设置 obj-m 变量的值为 asyncnoti.o。

文件修改完成之后保存退出,输入如下命令编译出驱动模块文件:

make

编译成功以后就会生成一个名为"asyncnoti.ko"的驱动模块文件,如下所示:

```
y@zy-virtual-machine:~/linux/drivers/16_asyncnoti$ ls
asyncKeyApp.c asyncnoti.c Makefile
zy@zy-virtual-machine:~/linux/drivers/16 asyncnoti$
zy@zy-virtual-machine:~/linux/drivers/16_asyncnoti$_make
make -C /home/zy/workspace/kernel-driver/linux-xlnx-xlnx rebase_v5.4 _2020.2 M=`
pwd` modules
make[1]: 进入目录"/home/zy/workspace/kernel-driver/linux-xlnx-xlnx rebase_v5.4
2020.2"
  CC [M] /home/zy/linux/drivers/16 asyncnoti/asyncnoti.o
  Building modules, stage 2.
  MODPOST 1 modules
CC [M] /home/zy/linux/drivers/16_asyncnoti/asyncnoti.mod.o
LD [M] /home/zy/linux/drivers/16_asyncnoti/asyncnoti.ko
make[1]: 离开目录"/home/zy/workspace/kernel-driver/linux-xlnx-xlnx_rebase_v5.4_
2020.2
zy@zy-virtual-machine:~/linux/drivers/16_asyncnoti$
zy@zy-virtual-machine:~/linux/drivers/16_asyncnoti$ ls
asyncKeyApp.c asyncnoti.ko
                                   asyncnoti.mod.c asyncnoti.o modules.order
asyncnoti.c
                 asyncnoti.mod asyncnoti.mod.o Makefile
                                                                        Module.symvers
zy@zy-virtual-machine:~/linux/drivers/16 asyncnoti$
```

图 35.4.1 编译驱动模块



原子哥在线教学: www.yuanzige.com 论

论坛:www.openedv.com/forum.php

2、编译测试 APP

输入如下命令编译测试 asyncnotiApp.c 这个测试程序:

\$CC asyncKeyApp.c -o asyncKeyApp

编译成功以后就会生成 asyncKeyApp 这个应用程序。

35.4.2 运行测试

将上面编译出来的 asyncnoti.ko 和 asyncKeyApp 这两个文件拷贝到 NFS 共享目录下根文 件系统的/lib/modules/5.4.0-xilinx 文件夹中。

将 SD 启动卡插到开发板中,连接开发板串口、网口和电源线,设置拨码开关为 SD 卡启 动模式,然后打开电源,启动开发板。

系统启动完成后,通过串口终端进入"/lib/modules/5.4.0-xilinx"目录下,输入如下命令 加载 asyncnoti.ko 驱动模块:

depmod //第一次加载驱动的时候需要运行此命令

modprobe asyncnoti.ko //加载驱动

驱动加载成功以后使用如下命令来测试中断:

./asyncKeyApp /dev/key

按下开发板上的 PS_KEY0 键,当按键按下或按键松开时终端就会打印出相应的信息,如 图 35.4.2 所示:

root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx# depmod root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx# modprobe asyncnoti.ko root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx# root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx# lsmod Tainted: G asyncnoti 16384 0 - Live 0xbf000000 (0) root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx# ./asyncKeyApp /dev/key Key Press Key Release Key Press Key Release

图 35.4.2 应用程序测试结果

从图 35.4.2 可以看出,捕获到 SIGIO 信号,并且按键值获取成功,大家可以自行以后台 模式运行 asyncKeyApp,使用 top 命令查看一下这个应用程序的 CPU 使用率,这里就不给大 家演示了。

如果要卸载驱动的话输入如下命令即可:

rmmod asyncnoti.ko

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

正点原子

第三十六章 驱动静态编译

前面所有章节的驱动实验当中,我们都是将驱动代码编译成.ko 驱动模块,然后在 linux 系统下使用 insmod 或 modprobe 命令加载驱动模块之后进行测试; 驱动开发工程师将驱动代 码编译成.ko 驱动模块可以很方便的进行驱动调试,所以在驱动调试阶段一般都选择将其编译 为模块,这样我们修改驱动以后只需要编译一下驱动代码即可,不需要编译整个 Linux 源代 码;而且在调试的时候只需要加载或者卸载驱动模块即可,不需要重启整个系统。总之,将 驱动编译为模块最大的好处就是方便开发,当驱动开发完成,确定没有问题以后就可以将驱 动编译进 Linux 内核中,当然也可以不编译进 Linux 内核中,具体看自己的需求以及设备的特 点来决定!

那么本章将给大家介绍如何将驱动源码编译进 Linux 内核中,这里我把它叫做静态编译, 本章就以上一章(第三十五章)驱动实验为例,将驱动源码编译进 linux 内核中。



36.1 修改 Makefile

首先将上一章实验目录 16_asyncnoti 下的驱动源文件 asyncnoti.c 拷贝到 linux 内核源码目 录 drivers/char 目录下,如下所示:

zy@zy-virtual-	<pre>machine:~/linux/d</pre>	rivers/16_asyncno [·]	ti\$ ls
asyncKeyApp	asyncnoti.ko	asyncnoti.mod.o	modules.order
asyncKeyApp.c	asyncnoti.mod	asyncnoti.o	Module.symvers
asyncnoti.c	asyncnoti.mod.c	Makefile	
zy@zy-virtual-	<pre>machine:~/linux/d</pre>	rivers/16_asyncno [.]	<pre>ti\$ cp -a asyncnoti.c /home/zy/</pre>
workspace/kern	el-driver/linux-x	lnx-xlnx rebase v	5.4 2020.2/drivers/char/
zy@zy-virtual-	<pre>machine:~/linux/d</pre>	rivers/16_asyncno	ti\$
zy@zy-virtual-	<pre>machine:~/linux/d</pre>	rivers/16_asyncno	ti\$

图 36.1.1 拷贝驱动源文件到内核源码目录

/home/zy/workspace/kernel-driver/linux-xlnx-xlnx rebase v5.4 2020.2/drivers/char/是笔者内 核源码所在目录, linux 内核源码目录下的 drivers 文件夹中存放了驱动相关的代码, 此目录根 据驱动类型的不同,分门别类进行整理,比如 drivers/i2c 就是 I2C 相关驱动目录, drivers/gpio 就是 GPIO 相关的驱动目录,而 char 目录下存放的就是字符设备相关的驱动代码,所以这里 我们将驱动源文件 asyncnoti.c 拷贝到了 drivers/char 目录。

打开 drivers/char 目录下的 Makefile 文件,将 asyncnoti.c 文件添加进来,如下所示:

51		
52 obj-\$(CONFIG_XILLYBUS)	+= xillybus/	
53 obj-\$(CONFIG_POWERNV_OP_PANEL)	+= powernv-op-panel	0
54 obj-\$(CONFIG ADI)	+= adi.o	
55 obj-\$(CONFIG_ASYNCNOTI)	+= asyncnoti.o	
插入		55,47

图 36.1.2 将 asyncnoti.o 添加到 Makefile 文件中

内核源码 drivers 目录下的各个子目录以及子目录的子目录中一般都会存在一个 Makefile 文件,在这些 Makefile 文件中我们会看到很多 "obj-\$(CONFIG_XXX) += xxxx.o",例如图 36.1.2 中所示。CONFIG_XXX 变量一般可以取 3 个不同的值: y、m、n; 所以将 \$(CONFIG_XXX)替换之后也就是定义了三个变量 obj-y、obj-m、obj-n。

当我们编译内核源码的时候, obj-y 变量中所有的 xxxx.o 所对应的 xxxx.c 文件都会被编译 进内核镜像;当在内核源码目录执行"make modules"编译内核模块的时候,obj-m变量中所 有的 xxxx.o 文件对应的 xxxx.c 源文件会被编译成.ko 驱动模块文件; 而 obj-n 变量中所有的 xxxx.o 文件对应的 xxxx.c 源文件既不会编译进内核镜像文件,也不会编译成驱动模块。所以 由此可以知道 CONFIG_XXX 好比是一个"选择器",你可以选择将源文件编译进内核中, 也可以选择编译成一个.ko驱动模块文件,也可以不编译。

在图 36.1.2 中我们将 "obj-\$(CONFIG_ASYNCNOTI) += asyncnoti.o" 添加到了 Makefile 文件的最后, CONFIG_ASYNCNOTI 可以自定义, 但是不能和其它名字冲突, 一般就是 "CONFIG_"加上.c 源文件的大写字母;将内容添加到 Makefile 文件之后保存退出即可!

36.2 修改 Kconfig

打开内核源码 drivers/char 目录下的 Kconfig 文件,添加如下内容:



原子哥在线教学: www.yuanzige.com



图 36.2.1 修改 Kconfig 文件

关于 Kconfig 文件的语法在《U-Boot 图形化配置及其原理》章节中给大家介绍过,这里 不再说明! "config ASYNCNOTI" 其实就是 CONFIG_ASYNCNOTI 变量的配置项,用于配 置 CONFIG_ASYNCNOTI 变量的值是 y、m 还是 n。tristate 表示三态,也就是可以将 CONFIG_ASYNCNOTI 变量的值设置为 y、m 或者 n 中的任何一个,后边携带的字符串信息 表示的是对该 config 配置项的一个描述信息。

"default y" 表示该配置项的默认值是 y,关于 Kconfig 文件的语法如果大家有什么不懂 的可以看看《U-Boot 图形化配置及其原理》章节。在 Kconfig 文件中添加配置项之后,当我 们在内核源码目录执行"make menuconfig"的时候就可以通过图形化界面的方式对其进行配 置了。

36.3 menuconfig 配置

在内核源码目录下执行下面这些命令进入到 menuconfig 配置界面:

make distclean	// 清理整个工程
make xilinx_zynq_defconfig	// defconfig
make menuconfig	// menuconfig



zy@zy-vir	tual-machine:~/workspace/kernel	-driver/atk-zynq-linux-xlnx\$ make distclean
CLEAN	scripts/basic	
CLEAN	scripts/kconfig	
CLEAN	include/config include/generate	ed
CLEAN	.config	
zy@zy-vir	tual-machine:~/workspace/kernel	-driver/atk-zynq-linux-xlnx\$ make xilinx_zynq_defconfig
HOSTCC	scripts/basic/fixdep	
HOSTCC	scripts/kconfig/conf.o	
HOSTCC	scripts/kconfig/confdata.o	
HOSTCC	scripts/kconfig/expr.o	
LEX	<pre>scripts/kconfig/lexer.lex.c</pre>	
YACC	<pre>scripts/kconfig/parser.tab.[ch]</pre>	
HOSTCC	<pre>scripts/kconfig/lexer.lex.o</pre>	
HOSTCC	scripts/kconfig/parser.tab.o	
HOSTCC	scripts/kconfig/preprocess.o	
HOSTCC	scripts/kconfig/symbol.o	
HOSTLD	scripts/kconfig/conf	
#		
# contigu #	ration written to .config	
zy@zy-vir	<pre>tual-machine:~/workspace/kernel</pre>	-driver/atk-zynq-linux-xlnx\$ make menuconfig
UPD	scripts/kconfig/mconf-cfg	
HOSTCC	scripts/kconfig/mconf.o	
scripts/k	config/mconf.c: In function 'se	arch_conf':
scripts/k	config/mconf.c:425:6: warning: '	implicit declaration of function <code>`strncasecmp' -Wimplic</code>

图 36.3.1 执行 menuconfig 图形化配置界面命令

注意,如果执行图形化配置命令出现报错或者警告,请参考 36.5 小节中提供的解决办法。 进入 menuconfig 图形化界面之后,找到 Character devices 配置界面:

Device Drivers --->

Character devices --->

如下图所示:

Character devices

Arrow keys navigate the menu. < Enter> selects submenus ---> (or empty submenus ----). Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] built-in [] excluded <M> module < > module capable [] Non-standard serial port support HSDPA Broadband Wireless Data Card - Globe Trotter < > GSM MUX line discipline support (EXPERIMENTAL) < > Trace data sink for MIPI P1149.7 cJTAG standard < > < > NULL TTY driver Automatically load TTY Line Disciplines <*> Character device key driver support *] /dev/mem virtual device support [] /dev/kmem virtual device support

图 36.3.2 Character devices 配置界面

在图 36.3.2 中可以看到我们在上一个小节为 asyncnoti.c 驱动源文件添加的配置项 "Character device key driver support", "<*>"尖括号里边的"*"符号表示该配置项默认是 选中的,也就是配置为"y";将光标移动到该配置条目上,分别按键盘上的"Y"、"M"、 "N"键可以分别选择将该驱动编译进内核、编译成单独的驱动模块、不编译驱动源文件。 因为本章我们需要将驱动编译进内核中,所以这里选"Y"即可!

对于 menuconfig 界面中的每一个配置项,都可以查看它的 help 帮助信息,如下所示:

原子哥在线教学: www.yuanzige.com

论坛:www.openedv.com/forum.php

正点原子

Character device key driver support —

CONFIG_ASYNCNOTI: Say Y here if you want to support the key device. If unsure, say N. If you need a drive module, say M Symbol: ASYNCNOTI [=y] Type : tristate Prompt: Character device key driver support Location: -> Device Drivers -> Character devices Defined at drivers/char/Kconfig:10

图 36.3.3 menuconfig 配置项帮助信息

从图 36.3.3 中可以知道,该配置项就是我们在图 36.2.1 中所加入到 Kconfig 文件中的配置 项。按回车键退出帮助页面,保存当前配置,然后退出 menuconfig 界面。

36.4 编译测试

为了验证我们加入的 asyncnoti.c 驱动源文件在内核启动的时候会自动运行,我们可以在 asyncnoti.c 文件的驱动入口函数 mykey_init 中加入打印语句:

printk(KERN_INFO "##### key driver registration is successful ! #####\n");

如下所示:



图 36.4.1 加入打印语句

修改完成之后保存退出即可,接下来就可以进行编译了,执行下面两条命令编译内核源 码得到 zImage 和设备树镜像文件:

make zImage -j8 make dtbs 编译完成之后,如下所示:
领航者 ZYNQ 之嵌入式 Linux 开发指南



Ţ	仕线教子:	www.yuanzige.com			
	AS	arch/arm/boot/compressed/hyp-stub.o			
	AS	arch/arm/boot/compressed/lib1funcs.o			
	AS	arch/arm/boot/compressed/ashldi3.o			
	AS	arch/arm/boot/compressed/bswapsdi2.o			
	AS	arch/arm/boot/compressed/piggy.o			
	LD	arch/arm/boot/compressed/vmlinux			
	OBJCOPY	/ arch/arm/boot/zImage			
	Kernel:	: arch/arm/boot <mark>/</mark> zImage <mark>i</mark> s ready			
<pre>zy@zy-virtual-machine:~/workspace/kernel-driver/linux-xlnx_</pre>					

图 36.4.2 内核源码编译完成

编译得到的 zImage 和 system-top.dtb 文件所在目录分别为 arch/arm/boot/zImage 和 arch/arm/boot/dts/system-top.dtb,将这两个文件拷贝到开发板 SD 启动卡的 Fat 分区中,替换旧 的镜像文件,然后重新启动开发板;系统在启动过程当中就会自动运行 asyncnoti.c 驱动模块,打印信息如下所示:

xilinx-frmbuf 43c40000.v_frmbuf_wr: Xilinx AXI FrameBuffer Engine Drive	r Probed!!
xilinx-frmbuf 43c20000.v_frmbuf_rd: Xilinx AXI frmbuf DMA_MEM_TO_DEV	
xilinx-frmbuf 43c20000.v_frmbuf_rd: Xilinx AXI FrameBuffer Engine Drive	r Probed!!
##### key driver registration is successful ! #####	
brd: module loaded	
loop: module loaded	
libphy: Fixed MDIO Bus: probed	

图 36.4.3 内核启动打印信息

图 36.4.3 中标识出来的打印信息就是我们在 asyncnoti.c 驱动源文件中加入的打印信息, 所以可以证明我们的驱动程序已经成功运行了,那么接下来可以进行驱动程序的测试工作了; 输入用户名和密码登录开发板 linux 系统,进入到根文件系统/lib/modules/5.4.0-xilinx 目录下, 执行下面这条命令运行上一章的驱动测试程序 asyncKeyApp:

./asyncKeyApp /dev/key 按下或者松开开发板的 PS_KEY0 按键,终端就会打印相应的信息,如下所示:

root@ALIENTEK-Z	YNQ-driver:~# co	d /lib/modules/5.4.	0-xilinx			
root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx# ls						
asyncKeyApp	blockio.ko	keyirq.ko	modules.dep	noblockio.ko		
asyncnoti. <mark>ko</mark>	k eyApp	modules.alias	modules.symbols			
root@ALIENTEK-ZYNQ-driver:/lib/modules/5.4.0-xilinx# ./asyncKeyApp /dev/key						
Key Press						
Key Release						
Key Press						
Key Release						

图 36.4.4 运行驱动测试程序

测试的效果跟上一章测试情况是一样的,说明的驱动工作正常。测试完成之后按住键盘 Ctrl+C组合键退出 asyncKeyApp测试程序。将驱动代码静态编译到 linux 内核中之后,我们就 不需要使用 insmod 或 modprobe 等方式加载.ko 驱动模块了,内核启动的过程中会自动运行驱 动程序!

36.5 解决图形化配置时出现的报错和警告问题

运行 make menuconfig 命令时,会出现如下图所示的警告和报错:

领航者 ZYNQ 之嵌入式 Linux 开发指南



原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

zy@zy-virtual-machine:~/workspace/kernel-driver/atk-zyng-linux-xlnx\$ make menuco
nfig
UPD scripts/kconfig/mconf-cfg
HOSTCC scripts/kconfig/mconf.o
<pre>scripts/kconfig/mconf.c: In function 'search_conf':</pre>
<pre>scripts/kconfig/mconf.c:423:6: warning: implicit declaration of function 'strnca</pre>
secmp' [-Wimplicit-function-declaration]
<pre>if (strncasecmp(dialog_input_result, CONFIG_, strlen(CONFIG_)) == 0)</pre>
Annanana
<pre>scripts/kconfig/mconf.c: In function 'main':</pre>
<pre>scripts/kconfig/mconf.c:1021:8: warning: implicit declaration of function 'strca</pre>
<pre>secmp' [-Wimplicit-function-declaration]</pre>
if (!strcasecmp(mode, "single_menu"))
Awwwwwww
HOSTCC scripts/kconfig/lxdialog/checklist.o
HOSTCC scripts/kconfig/lxdialog/inputbox.o
HOSTCC scripts/kconfig/lxdialog/menubox.o
HOSTCC scripts/kconfig/lxdialog/textbox.o
HOSTCC scripts/kconfig/lxdialog/util.o
HOSTCC scripts/kconfig/lxdialog/yesno.o
_HOSTLD_scripts/kconfig/mconf
/usr/bin/ld: 找不到 -lncursesw
collect2: error: ld returned 1 exit status
scripts/Makefile.host:116: recipe for target 'scripts/kconfig/mconf' failed
make[1]: *** [scripts/kconfig/mconf] Error 1
Makefile:567: recipe for target 'menuconfig' failed
make: *** [menuconfig] Error 2
zy@zy-virtual-machine:~/workspace/kernel-driver/atk-zynq-linux-xlnx\$

图 36.5.1 使用内核图形化配置界面报错

由于前面两个警告不影响配置界面的启动,所以这里我们只解决找不到 -lncursesw 的问 题。"-Incursesw"表示名为"ncursesw"的库,我们使用"locate libncursesw.so"命令定位库 文件的位置,结果如下图所示:



图 36.5.2 定位库文件的位置

使用软连接将 ncursesw.so 链接到第一个库, 命令如下:

sudo ln -s /lib/x86_64-linux-gnu/libncursesw.so.5 /usr/lib/libncursesw.so

设置完成后,重新运行"make menuconfig"命令便能启动图形配置界面。

原子哥在线教学: www.yuanzige.com 论坛:www.openedv.com/forum.php

②正点原子