ARM Linux 在 EP7312 上的移植

李程远， 刘文峰， 李善平

（浙江大学计算机科学与技术学院，浙江 杭州 310027）

摘 要：Linux 是一种支持多种体系结构处理器的操作系统，有很强的移植性。描述了将 ARM Linux 移植到基于 EP7312 处理器的目标板上的方法与过程。首先介绍了 EP7312 处理器和 ARM Linux，并简单说明了如何搭建移植环境，然后着重讨论了在该开发板上 bootloader 的设计实现以及 ARM Linux 内核的移植的方法，最后对在这种基于 Linux 的嵌入式系统环境下如何开发应用程序作了探索和展望。

关键词：嵌入式系统；移植；ARM Linux；EP7312；bootloader

Porting ARM Linux to EP7312

LI Cheng-yuan， LIU Wen-feng， LI Shan-ping

(College of Computer Science， Zhejiang University, Zhejiang Hangzhou 310027, China)

Abstract: Linux supports various architecture and is an operation system which is easy to be ported. The technique of porting ARM Linux to one board based on Cirrus Logic EP7312 SoC processor is introduced in this paper. All the work consists of the design and implementation of bootloader, the porting of ARM Linux kernel, and some relative research about the development of embedded applications.

Key words: embedded system; porting; ARM linux, EP7312; bootloader

1 引 言

ARM 处理器是一种 32 位的嵌入式 RISC 处理器。在众多同种类型的处理器中，ARM 系列已经成为了当今应用范围最广的嵌入式芯片。第 1 枚 ARM 芯片诞生于 1983 年 10 月到 1985 年 4 月之间的英国[1]，当时属于 Acorn 公司，是 Acorn RISC Machine 的简写。为了扩展市场，ARM 产品线在成为 Acorn 公司的核心之后，被独立分割出来，成立了 ARM 公司，变成了 Advanced RISC Machine 的缩写。

根据市场调查，2001 年初，ARM 的 32 位处理器的市场占有率已经超过了 75%。

ARM 公司是知识产权（Intellectual Property）供应商，它负责 ARM 处理器的芯片设计，转让设计许可，由合作伙伴公司来生产使用 ARM 处理器核的各具特色的芯片。

ARM 公司在全世界范围内有超过 100 个这样的合作伙伴。在芯片特点上，ARM 处理器核耗电少，成本低，功能强，还特有 16/32 位双指令集，这使得 ARM 成为了移动通信、手持计算、多媒体数字消费等嵌入式解决方案的 RISC 标准。ARM 的产品线主要包括 ARM7 Thumb 家族和 ARM9 Thumb 家族、ARM10 Thumb 家族以及 StrongARM 家族。

2 基于 ARM 的 EP7312 处理器和 ARM Linux

在本文中使用的目标平台 EP7312 是 Cirrus Logic 公司使用 ARM7 Thumb 家族中 ARM 720T 处理器内核开发的一块 SoC（System-on-Chip）嵌入式微处理器。EP7312 专门为 PDA、Internet 设备、移动电话和手持设备等设计成超低功耗和高性能的微处理器。它的核心逻辑部件 ARM720T 采用了具有 8K 字节的 4 路集合关联（set-associative）独立 cache 和一个写缓存，并且还包含了一块加强的 MMU（内存管理部件）。这些特性使得开发人员可以将 Windows CE 和 Linux 等操作系统移植到基于这块微处理器的目标系统中。

Linux 是一种被广泛移植到各种嵌入式平台的开

收稿日期：2002-06-11

作者简介：李程远（1978-），男，浙江杭州人，硕士生，主要研究方向为嵌入式系统、操作系统； 刘文峰，硕士生，主要研究方向为嵌入式系统，操作系统； 李善平，教授，博士生导师，主要研究领域为操作系统、嵌入式系统、CIMS。

放源代码操作系统。ARM Linux 是一个将 Linux 内核移植到各种基于 ARM 处理器的目标系统的项目，由 Russell King<rmk@arm.linux.org.uk>主持，已经为超过 100 种不同的目标机器成功完成了移植工作，包括有基于 ARM 的计算机、网络设备和目标板等。

本文的工作主要包括 bootloader 的设计实现、ARM Linux 内核移植、嵌入式系统应用开发方法探索 3 个部分。

3 移植环境的建立

嵌入式系统的开发与一般的应用开发最大的差别就在于:前者需要建立特殊的硬件环境，而后者一般基于特定的操作系统或者分布式平台。后者的平台已经对硬件或者网络媒质做了抽象，从而不需要由系统开发者来完成这些工作。而在嵌入式系统开发中，这也由开发者完成。

嵌入式系统开发环境一般分成主机端(HOST)和目标板(TARGET)两个部分。主机端是开发平台，用于运行开发过程中的各种工具；目标板是运行和测试平台，是嵌入式系统的最终驻留环境。在主机端和目标板之间需要通过某种方式进行通信，如使用 RS232C 串口。这种通信的目的在于发送控制指令和传输数据，同时获得反馈信息。图 1 是本文中系统移植工作的硬件环境：

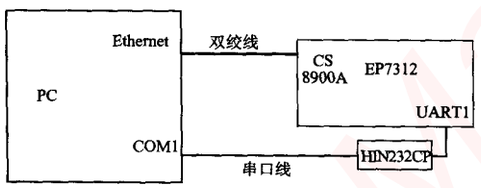

图 1 系统移植工作的硬件环境

主机端的 PC 使用 COM1 和 EP7312 的 UART1 相连接，因为 RS232C 和 UART 的电气特性不同，在连接两者的串口线上使用了一块 HIN232CP 芯片完成电平转换。通过 RS232C 串口完成对目标板的必要控制功能。EP7312 开发板上配备有一块 CS8900A 以太网卡芯片，和主机端建立原始(raw)IP 连接，使用链路层地址(以太网 MAC 地址)完成大批量数据的传送。

硬件环境建立之后，就需要创建软件开发环境。软件环境主要是指 ARM 体系结构的交叉编译环境，在主机端使用 RedHat Linux 7.1 操作系统，并在其上建立 gcc 的 ARM 体系结构的交叉开发环境。交叉编译环境所需的源代码软件包有下面这些：

(1)binutils-2.11.2.tar.gz

binutils 里面包括 GNU 的链接器 ld、汇编代码编译器 as、用来将文件打包重组的 ar 以及为 ar 打包的文件建立符号表的 ranlib 等工具。

(2)linux-2.4.17.tar.gz, patch-2.4.17-rmk5.gz

交叉开发环境的编译过程中需要 Linux 内和源代码中的头文件。

(3)gcc-2.95.3.tar.gz 和 gcc-2.95.3.diff.bz2

gcc 是 GNU 的 C 编译器，可以配置编译为多种体系结构目标的交叉编译器。这里配置为 arm-elf-编译目标(TARGET)。

(4)glibc-2.2.3.tar.gz 和 glibc-linuxthreads-2.2.3.tar.gz

gcc 使用的 C 函数库和针对 Linux 的线程库。

将这些软件包按照如下顺序进行编译：

binutils → 内核头文件 → gcc → glibc

交叉开发环境建立后，在目标板上运行的可执行代码都由它来编译链接。

4 Bootloader 的设计实现

Bootloader 被用作系统从硬件启动到操作系统启动的过渡，是嵌入式系统中必不可少的一段程序。它相当于 PC 机中的 BIOS 和 OS Loader，用于初始化运行硬件和启动操作系统，因此其实现方式由硬件的特性决定。和 BIOS/OS Loader 一样，它需要固化在目标板中，每次启动目标板时，首先会运行 bootloader，在它完成 CPU 和相关硬件的初始化之后，才从事先规定的地址启动操作系统或者嵌入式应用的固化程序。

在嵌入式系统开发过程中，bootloader 还担任了与主机端通信的任务，它相当于一个的“服务器”，不断监听从主机端传来的控制信息和数据信息，完成相应的操作。

EP7312 处理器根据 nMEDCHG 引脚接入电平的不同，触发两种启动模式：内部启动模式和外部启动模式。启动模式的不同会使系统地址映射发生变化，表 1 是两种启动模式下的系统地址映射表。

表 1 EP7312 系统地址映射表

地址		对应存储片
内部模式	外部模式	
0x0000.0000	0x7000.0000	片内 ROM(nCS[7])
0x1000.0000	0x6000.0000	片内 SRAM(nCS[6])
0x2000.0000	0x5000.0000	扩展(nCS[5])
0x3000.0000	0x4000.0000	扩展(nCS[4])
0x4000.0000	0x3000.0000	扩展(nCS[3])
0x5000.0000	0x2000.0000	扩展(nCS[2])
0x6000.0000	0x1000.0000	FLASH Bank 1 (nCS[1])
0x7000.0000	0x0000.0000	FLASH Bank 0 (nCS[0])

不论是采用哪一种启动方式，系统都是从地址 0x0000.0000，也就是地址映射的最起始位置开始执行指令。

由于外部启动是从外部Flash开始的，本文的bootloader不考虑这种启动的情况，而只针对于内部启动。EP7312采用内部启动时，首先会执行片内ROM中的一段只有128个字节长的汇编代码[2]。这128个字节的程序只能从串口中读取2K字节并把这2K的代码存放到地址值为0x1000.0000的内部SRAM中，然后PC就跳转到地址0x1000.0000开始运行这下载下来的2K代码，因此我们用于在目标板运行的bootloader的大小必须限制在2K字节以内，并用这段可执行程序来实现下面的功能：

(1) 设置EP7312的控制寄存器SYSCON2和SYSCON3;

(2)初始化SDRAM控制器，完成SDRAM的侦测;

(3)初始化地址扩展控制寄存器;

(4)实现网卡控制芯片CS8900A的驱动;

(5)将PC寄存器指向Linux内核存放的地址，运行操作系统。

为了使存放于目标板上的可执行代码小于2K字节，我们将bootloader的代码分成两部分，一部分存放于主机端，另一部分存放于目标板上，两者通过串口进行通信。通过这种方法来减小目标板上的代码量，并且依靠主机端与目标板的协同工作来启动Linux内核。在目标板上的程序主要完成下面的任务：

(1)从串口接收从主机端发送过来的信息;

(2) 根据主机端不同的命令调用相应的写控制寄存器函数来设置各个控制芯片;

(3)通过串口或者网卡芯片接收大块数据，并把它写到SDRAM中的指定地址;

(4) 完成一个任务后通过串口向主机端发送一个反馈信息。

在主机端完成的任务如下：

(1)解释输入的命令行参数;

(2)打开主机端的串口和网卡控制芯片;

(3) 根据命令行参数决定向目标板发送命令的顺序。

(4) 向目标板传送需要改写的控制寄存器的地址和控制字;

(5)读入内核映像文件，通过串口或者网卡将数据发送给目标板。

图2描述了bootloader的工作流程。

5 Linux内核的移植

我们使用的Linux内核版本是2.4.17，采用ARM Linux的补丁是patch-2.4.17-rmk5。完成ARM Linux内核到EP7312体系结构的移植，需要修改所有和体系结构

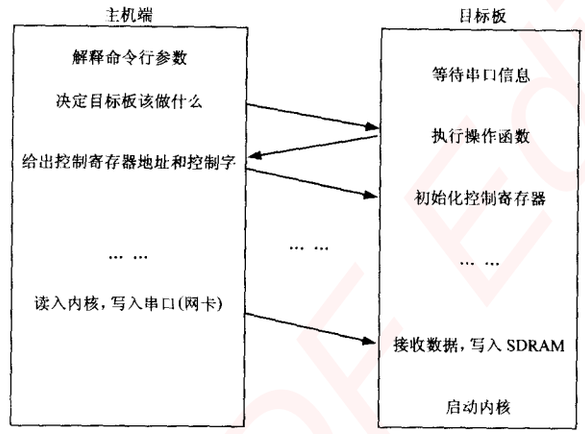

图2 Bootloader的运行流程

相关的代码。其中主要的部分是内核入口部分、处理器和体系结构初始化部分、IO端口映射部分和中断初始化部分。除此之外，还有EP7312相关设备的驱动程序，如CS8900A以太网接口的驱动等。

要把ARM Linux移植到EP7312上，首先必须搞清楚内核入口代码head-armv. S (arch/arm/kernel/)。head-armv.S所需要完成的任务主要集中在下面的一段代码中：

```
mov r0, #F_BIT | I_BIT | MODE_SVC   @ make sure svc mode
msr cpsr_c, r0                      @ and all irqs disabled
bl __lookup_processor_type
teq r10, #0                         @ invalid processor?
moveq r0, #'p'                      @ yes, error 'p'
beq __error
bl __lookup_architecture_type
teq r7, #0                          @ invalid architecture?
moveq r0, #'a'                      @ yes, error 'a'
beq __error
bl __create_page_tables
adr lr, __ret                       @ return address
add pc, r10, #12                    @ initialise processor
                                    @ (return control reg)
```

在这里内核检查Linux内核代码中的相关信息与目标板处理器中寄存器中的相关值是否相匹配。处理器寄存器中的值由bootloader传递过来，里面的具体内容是从EP7312的控制寄存器中得到。Linux内核中的相关信息存放在内核的.proc.info段和.arch.info段中(Linux内核映像文件的布局由GNU的连接脚本所决定)，它们用__proc_info_begin、__arch_info_begin和__proc_info_end、__arch_info_end作为开始与结束的标识。.proc.info段中的具体的内容定义在arch/arm/mm/proc-arm720.S文件中，主要包括CPU值、MMU的控制字、一组处理器的操作函数等信息。.arch.info段的内容则是

通过系统中的一组宏来定义，定义的内容包括体系类型、内存和I/O 的起始地址、系统参数存放的起始地址、体系修正函数、I/O映射函数和中断初始化函数。Linux内核和bootloader做好协调，才能在运行过程中查找到正确的函数，启动Linux内核。

当Linux内核能够在EP7312芯片上运行起来后，还需要在内核中加入开发板上其它控制器和设备的驱动，这里最重要的是网卡驱动。

EP7312开发板中所带的网卡芯片是Cirrus Logic公司的CS8900A。Linux2.4.17中有i386体系结构下的CS8900A驱动程序代码，不过没有在ARM体系结构下的驱动，因此需要把它移植到EP7312开发板上。

在Linux内核代码中CS8900A的侦测函数是cs89x0_probe，首先需要让内核在启动时能够执行这个函数，用以侦测开发板上的CS8900A芯片。

Linux内核用net_device结构来描述一块网卡，结构中的init项指向（或者间接指向）每个网卡驱动的侦测函数。系统中存在一个net_device结构的链表，如图3所示：

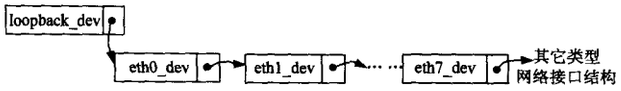

图3 内核中网络接口结构链表

表明在内核启动时可以最多侦测8块网卡，通过扫描这个链表，就可以执行网卡驱动的侦测函数。Linux利用GNU C编译器的C扩展属性功能，将一系列设备初始化函数统一放到initcall.init段中，然后在内核启动时将存放在这个段中函数逐个执行。因此，只要将eth0_dev中的init项指向CS8900A的侦测函数，并且将扫描net_device结构链表的函数放到initcall.init段中就可以在内核启动时执行CS8900A的侦测函数。

在修改CS8900A的侦测函数时需要注意的是，CS8900A有两种工作模式：存储模式和I/O模式。由于在bootloader中并没有设定存储模式，因此只能用CS8900A缺省的I/O模式来访问其内部的Packet Page。侦测函数中原先使用存储模式访问Packet Page的地方，需要改用I/O模式访问而且还要考虑8个I/O访问端口的地址问题。

6 结 语

嵌入式Linux系统的根文件系统通常是以Ramdisk的方式存在的，Ramdisk的映像文件一般以压缩的形式存放在Flash中，在内核启动时将这个压缩的文件解压存放到内存中再作为根文件系统装载。由于在应用程序开发期间常常需要改动Ramdisk中的内容，采用前面所说的方法就会经常写Flash而比较费时。我们可以通过使用bootloader将Ramdisk直接下载到SDRAM中的方法来解决这个问题。

Linux内核启动时，有几个重要的函数与Linux的内存管理相关，这两个函数的实现方式会影响到将Ramdisk直接下载到SDRAM中的方法的实现。

这些函数的处理过程是：在建立的内核内存分配器时，它会统计系统SDRAM的大小然后建立一张相应大小的位图来跟踪内存中的空闲页面。由于开发板的SDRAM的大小是16MBytes的，位图只需占用一个页面，大小为4K，存放的位置紧跟着内核，如图4所示。接着，系统把存放内核之后的内存区域，也就是_end标签之后的空间全部视为空闲区域，位图也会把对应这片空间内存页的位置为0。有了这张跟踪内存使用情况的位图之后，系统可以很方便地将所有置0位所对应的内存页面清空。

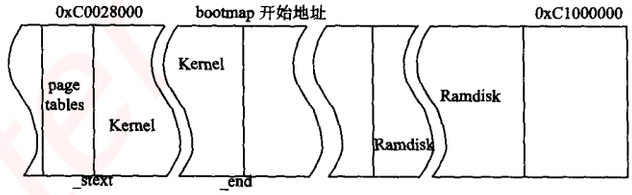

图4 系统内存布局

由于上面的这种内存初始化方法，使得存放Ramdisk的内存区域也被清空了，因此在内核启动之后就无法找到Ramdisk了。可以根据存放Ramdisk的起始和结束地址来计算出位图中相对应的位，并在系统将这些位置0之后，重新将它置1。这样，之后的系统操作中就不会将Ramdisk给清除了。

为了进一步方便应用程序的开发和调试，在内核加上网卡驱动后可以使用NFS的方法。在主机端开启一个NFS服务之后，不需要将编译好的程序放到Ramdisk的映像文件中而只要把它放到主机端的一个NFS输出目录，在目标板上再将这个目录挂载上就可以了。

参 考 文 献

[1] Steve Furber. ARM System-on-Chip Architecture, 2nd Edition[M]. Addison-Wesley Press, 2000.

[2] EP73XX User's Guide[R]. Cirrus Logic, 2001.

[3] CS8900A Product Data Sheet[R]. Cirrus Logic, 2001.

[4] 李善平，刘文峰，李程远，等. Linux内核2.4版源代码分析大全[M].北京:机械工业出版社,2002.

嵌入式资源免费下载

总线协议：

1. 基于 PCIe 驱动程序的数据传输卡 DMA 传输
2. 基于 PCIe 总线协议的设备驱动开发
3. CANopen 协议介绍
4. 基于 PXI 总线 RS422 数据通信卡 WDM 驱动程序设计
5. FPGA 实现 PCIe 总线 DMA 设计
6. PCI Express 协议实现与验证
7. VPX 总线技术及其实现
8. 基于 Xilinx FPGA 的 PCIE 接口实现
9. 基于 PCI 总线的 GPS 授时卡设计

VxWorks：

1. 基于 VxWorks 的多任务程序设计
2. 基于 VxWorks 的数据采集存储装置设计
3. Flash 文件系统分析及其在 VxWorks 中的实现
4. VxWorks 多任务编程中的异常研究
5. VxWorks 应用技巧两例
6. 一种基于 VxWorks 的飞行仿真实时管理系统
7. 在 VxWorks 系统中使用 TrueType 字库
8. 基于 FreeType 的 VxWorks 中文显示方案
9. 基于 Tilcon 的 VxWorks 简单动画开发
10. 基于 Tilcon 的某武器显控系统界面设计
11. 基于 Tilcon 的综合导航信息处理装置界面设计
12. VxWorks 的内存配置和管理
13. 基于 VxWorks 系统的 PCI 配置与应用

Linux：

1. Linux 程序设计第三版及源代码

2. NAND FLASH 文件系统的设计与实现
3. 多通道串行通信设备的 Linux 驱动程序实现
4. Zsh 开发指南-数组
5. 常用 GDB 命令中文速览
6. 嵌入式 C 进阶之道

Windows CE:

1. Windows CE.NET 下 YAFFS 文件系统 NAND Flash 驱动程序设计
2. Windows CE 的 CAN 总线驱动程序设计
3. 基于 Windows CE.NET 的 ADC 驱动程序实现与应用的研究
4. 基于 Windows CE.NET 平台的串行通信实现
5. 基于 Windows CE.NET 下的 GPRS 模块的研究与开发
6. win2k 下 NTFS 分区用 ntldr 加载进 dos 源代码
7. Windows 下的 USB 设备驱动程序开发
8. WinCE 的大容量程控数据传输解决方案设计
9. WinCE6.0 安装开发详解
10. DOS 下仿 Windows 的自带计算器程序 C 源码
11. G726 局域网语音通话程序和源代码
12. WinCE 主板加载第三方驱动程序的方法
13. WinCE 下的注册表编辑程序和源代码
14. WinCE 串口通信源代码
15. WINCE 的 SD 卡程序[可实现读写的源码]

PowerPC:

1. Freescale MPC8536 开发板原理图
2. 基于 MPC8548E 的固件设计
3. 基于 MPC8548E 的嵌入式数据处理系统设计
4. 基于 PowerPC 嵌入式网络通信平台的实现
5. PowerPC 在车辆显控系统中的应用

ARM：

1. 基于 DiskOnChip 2000 的驱动程序设计及应用
2. 基于 ARM 体系的 PC-104 总线设计
3. 基于 ARM 的嵌入式系统中断处理机制研究
4. 设计 ARM 的中断处理
5. 基于 ARM 的数据采集系统并行总线的驱动设计
6. S3C2410 下的 TFT LCD 驱动源码
7. STM32 SD 卡移植 FATFS 文件系统源码
8. STM32 ADC 多通道源码

Hardware：

1. DSP 电源的典型设计
2. 高频脉冲电源设计
3. 电源的综合保护设计
4. 任意波形电源的设计
5. 高速 PCB 信号完整性分析及应用
6. DM642 高速图像采集系统的电磁干扰设计