基于IntelPXA270的系统引导程序移植与实现

于 明,盛 俊,刘建设
(武汉理工大学 能源与动力工程学院,湖北 武汉 430063)

摘 要:为了满足嵌入式操作系统与硬件平台配置的多样性和灵活性,要求设计引导程序Bootloader,实现基本硬件初始化和引导操作系统内核。系统总结了基于PXA270嵌入式系统引导程序的移植方法和步骤;介绍了引导程序Bootloader的设计和实现方法;讨论了位置无关代码在引导程序中的应用;简述了用于测试PXA270硬件器件的Bootloader扩展功能。该设计方法除用于完成Bootloader基本功能外,位置无关代码实现了程序的快速运行,扩展功能可以测试PXA270基本硬件设备。该引导程序Bootloader已成功运行于PXA270嵌入式平台,可提供类似系统开发的直接借鉴。

关键词:引导程序;PXA270;位置无关代码;扩展功能;Linux
中图分类号:TP368.2 **文献标识码**:A **文章编号**:1673-629X(2010)06-0032-04

Transplant and Realization of Bootloader Object Based on PXA270 Embedded Systems

YU Ming, SHENG Jun, LIU Jian-she
(School of Energy and Power Engineering, Wuhan University of Technology, Wuhan 430063, China)

Abstract: In order to meet the configuration diversity and flexibility of embedded operating system and hardware platform, Bootloader is designed to initialize basic hardware and boot the operating system kernel. Based on PXA270, transport methods and procedures of embedded system Linux Bootloader is summarized; The design and implementation of Bootloader is introduced; The application of position independent code in the Bootloader is discussed; Based on Bootloader, extended function for testing embedded hardware interface is described. In addition to complete the basic functions of Bootloader, position independent code is applied to boot fast, expanded function is used to test the basic hardware of PXA270. Direct reference can be provided to the similar system.

Key words: Bootloader; PXA270; position independent code; extended function; Linux

0 引 言

嵌入式系统是以应用为中心,以计算机技术为基础,软硬件可裁剪,适合应用系统对功能、可靠性、成本、体积、功耗等有严格要求的专用计算机系统。嵌入式操作系统种类众多,为了满足其与硬件平台配置的多样性和灵活性,设计Bootloader引导程序,完成基本硬件初始化和引导操作系统内核[1]。文中基于Intel Xscale体系架构的PXA270处理器,Xscale本身与ARM V5TE架构兼容,具有代表性。

1 硬件平台介绍

PXA270是Intel推出的32位Xscale系列微处理器,工作频率高达520MHz[2]。处理器时钟可达1GHz,功耗1.6W,并能达到1200兆条指令/秒[3]。基于PXA270的嵌入式系统结构:Flash、SDRAM、CAN接口、CF卡、LCD液晶屏、以太网等。系统结构如图1所示。

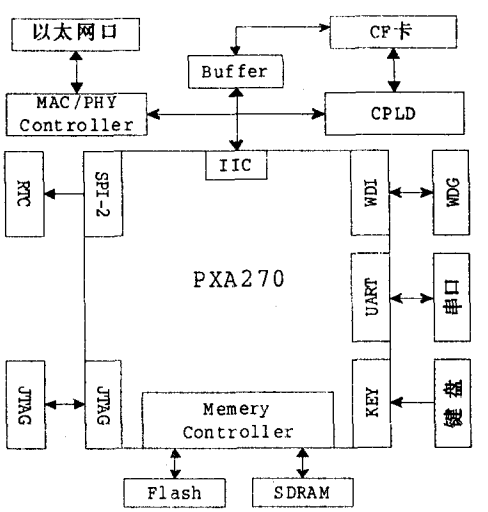

图1 PXA270系统结构图

收稿日期:2009-10-13;**修回日期**:2010-01-18
基金项目:湖北省科技攻关计划项目(2004AA101c28)
作者简介:于 明(1983-),男,黑龙江人,硕士,研究方向为轮机自动化及信息融合;导师:喻方平,教授,研究方向为轮机自动化。

2　Bootloader 的工作原理与启动过程

Bootloader 是系统运行的第一段代码,其作用是完成系统硬件初始化,建立内存空间映像图,使系统处于良好、稳定的工作环境,为加载操作系统内核或用户应用程序做好准备工作。介绍启动过程之前,先说明嵌入式系统各映像文件在存储空间中的位置[4],如图 2 所示。

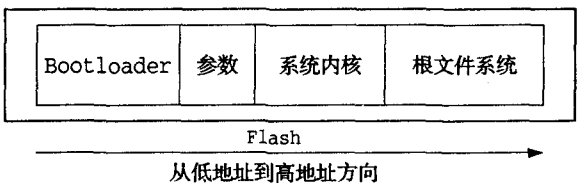

图 2　Flash 空间各映像分布图

Flash 各映像地址分布如下:

Bootloader:0X0000 0000—0X0003 8F00

Parameters:0X0003 8F00—0X0004 0000

Kernel:0X0004 0000—0X0024 0000

Filesystem:0X0024 0000—0X0200 0000

Bootloader 的启动过程分为两个阶段[5,6],Stage1 和 Stage2:

Stage1 主要完成工作:

*通用 IO、SDRAM 及系统时钟等基本硬件的初始化;

*为加载 Stage2 准备 RAM 空间;

*将 Stage2 拷贝到 SDRAM 中运行,设置好堆栈指针;

*跳转到 Stage2 的 C 程序入口点。

Stage2 主要完成工作:

*将内核和文件系统映像从 Flash 读入 SDRAM;

*为内核设置启动参数;

*初始化堆栈。

3　Bootloader 的设计与实现

3.1　位置无关代码在引导程序 Bootloader 中的应用

在设计 Bootloader 程序时,必须在裸机环境中进行,这时 Bootloader 映像文件的运行地址必须由程序员设定。通常将 Bootloader 程序下载到 ROM 的 0x0 地址进行启动,而为了实现快速启动,首先将 Bootloader 程序拷贝到 SDRAM 中再运行。由于这两者的地址并不相同,程序在 SDRAM 中的地址重定位过程必须由程序员完成。

实际上,由于 Bootloader 是系统上电后要执行的第一段程序,Bootloader 程序 Stage1 中的基本硬件初始化和完成 Stage2 拷贝之前的所有工作都是在 ROM 中执行的。也就是说,这些代码即使不在链接时所指定的运行时地址空间,也可以正确执行。这就是位置无关代码,它是一段加载到任意地址空间都能正常执行的特殊代码[7]。

Bootloader 映像结构由链接脚本决定,链接脚本用于设定映像文件的内存映射。决定如何将各目标文件合并起来并安排数据和符号的位置。脚本文件 boot.lds 的结构:

```
OUTPUT_ARCH(arm)
ENTRY(_start)
SECTIONS {
  . = BOOTADDR; /* Bootloader 起始地址 */
  _boot_start = .;
  .text ALIGN(4): { /* 代码段.text */
    *(.text)}
  .data ALIGN(4): { /* 数据段.data */
    *(.data)}
  .got ALIGN(4): { /* 全局偏移表 */
    *(.got)}
  _boot_end = .; /* Bootloader 结束地址 */
  .bss ALIGN(16): { /* 堆栈段.bss */
  _bss_start = .;
    *(.bss)
  _bss_end = .; } }
```

上面的链接脚本指定了 Bootloader 映像在执行时,将被重定位到 Bootloader 开始的存储空间,以保证在相关位置对符号进行正确引用,使程序正常运行。ARM 微处理器复位后总是从 0x0 地址取第 1 条指令,只需把 Bootloader 地址设置为 0x0,再把编译后生成的可执行文件下载到 Flash 的 0x0 地址空间,程序便可正常运行。但是,一旦链接时指定映像文件从 0x0 地址执行,那么 Bootloader 就只能在 Flash 中运行,无法拷贝到 SDRAM 实现快速运行。

利用 ARM 的 PIC 程序设计可以解决上述问题。只需在程序链接时,将 Bootloader 设置为 SDRAM 空间的地址(一般情况下利用 SDRAM 中最高的 1MB 存储空间为起始地址),这样 ARM 处理器上电复位后 Bootloader 仍然可以从 0x0 地址执行,并将其拷贝到指定的_boot_start 起始的 SDRAM 中运行。

实现上述功能的链接脚本所对应的启动代码架构如下:

```
.section .text
    .globl _start
    _start:
    B reset /* 复位异常 */
    … /* 其他异常处理 */
    reset:
    … /* 复位处理程序 */
```

```
copy_boot: /* 拷贝 Bootloader 到 SDRAM */
  LDR R0, =0x0
  LDR R1, = _boot_start
  LDR R2, = _boot_end
  1: LDRMIA R0!, { R3 - R10 }
  STRMIA R1!, { R3 - R10 }
  CMP R1, R2
  BLT 1b
  clear_bss:
  ... /* 清零.bss 段 */
  BL init_Stack /* 初始化堆栈 */
  LDR PC, = main /* 跳转到阶段 2 入口 */
  .end
```

程序入口为_start,即复位异常,其他异常向量都使用相对跳转指令 B 来实现,以保证位置无关性。引导程序 Stage1 在 Flash 中运行,为保证代码的位置无关性,在完成 GPIO、时钟和存储器等基本硬件初始化时,使用相对跳转指令。第二阶段在 SDRAM 中运行,利用链接脚本参数_boot_start 和_boot_end,将 Bootloader 映像拷贝到 SDRAM,并清除.bbs 段,初始化堆栈后,程序将 main 函数入口的绝对地址赋给 PC,进而跳转到 SDRAM 中继续运行。

3.2 Bootloader 的功能实现

Bootloader 一般分为两个部分:汇编部分和 C 语言部分[8,9]:其中汇编完成简单硬件初始化,C 负责复制数据和设置启动参数,串口通讯。

3.2.1 汇编部分

(1)CPU 工作模式切换和屏蔽中断:将 CPU 切换到超级用户(SVC)工作模式;将中断处理交给操作系统,Bootloader 不响应中断,屏蔽所有中断。

(2)初始化 GPIO:PXA270 提供 121 个 GPIO,根据硬件平台 GPIO 的连接情况,设计 Bootloader 中 GPIO 的初始化过程,用于产生和捕捉应用的输入、输出信号。

(3)初始化存储系统:初始化存储设备是引导程序要完成的主要工作,下面分析 SDRAM 的初始化。系统上电后 PXA270 将 SDRAM 置于自我刷新无时钟状态(Self_Refresh/Clk_stop)。要完成 SDRAM 正常工作,通过配置 MDREFR、MDCNFG 等寄存器完成状态机转换,使 SDRAM 从睡眠状态到正常状态(NOP)。状态机:自我刷新无时钟 Self_Refresh/Clk_stop ;自我刷新 Self_Refresh;低功耗 Powerdown;退出低功耗 PWRDNX;无操作 NOP。SDRAM 状态机转换如图 3 所示。

(4)初始化内核时钟频率:PXA270 提供时钟管理,控制所有时钟的产生、选通、变频,为存储控制器、系统总线、LCD 控制器提供时钟信号。

(5)复制 Boot 代码到 SDRAM:为调用 C 代码做准备,需将 Boot 代码复制到 SDRAM。

(6)建立堆栈和清除数据区:堆栈、DATA、BSS 段初始化是运行 C 代码前首先完成的工作。C 语言函数编译必须提供堆栈存放临时变量、调用位置、寄存器信息。除堆栈,需 DATA 段和 BSS 段存放 C 代码中有初始值的全局或静态变量以及没有指定初始值的全局变量。

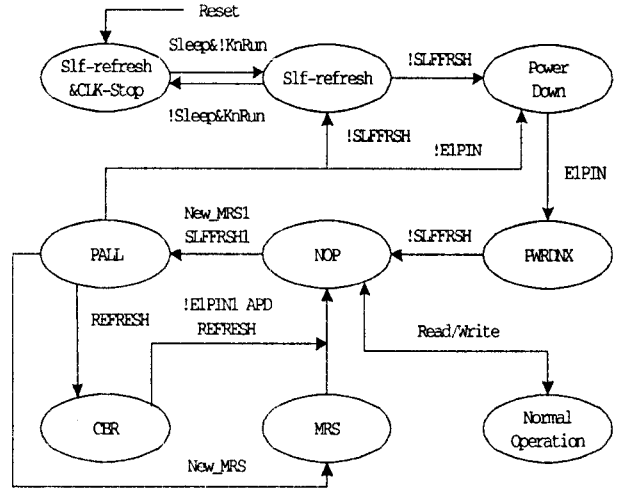

图 3 SDRAM 上电状态机

3.2.2 C 语言部分

完成启动所用设备的初始化。

(1)初始化本阶段要使用到的硬件设备:初始化标准串口,以便和终端用户进行 I/O 通信;初始化计时器;初始化网络等。

(2)系统内存映射:PXA270 提供 4G 物理寻址空间,实际的嵌入式系统只将其中一部分映射到内存单元上。Bootloader 程序利用 map_bsetup parts[]结构体对内存的分配进行配置。

(3)加载内核和根文件系统映像:包括内核映像和根文件系统映像所占用的内存范围。对于内核映像,将其拷贝到从(MEM_START + 0x8000)这个基地址开始的大约 1MB 大小的内存范围内;对于根文件系统,将其拷贝到(MEM_START + 0x00100000)开始的地方。

(4)设置内核启动参数:将映像复制到内存后,就准备启动 Linux 内核。调用内核前,需设置 Linux 内核启动参数。Linux2.4.x 以后的内核以标记列表(tagged list)形式来传递启动参数。启动参数标记列表以标记 ATAG_CORE 开始,以标记 ATAG_NONE 结束。每个标记由 tag_header 结构和随后的特定参数值数据结构来组成。

4 Bootloader移植及功能扩展

4.1 Bootloader的移植

(1)安装Toolchain。

由于ARM系统的特有局限，不具备友好的人机界面，一般开发环境都安装在PC上，而通过交叉编译工具Toolchain生成的最终目标文件才可以运行在相应的ARM平台上。

在PC机Linux系统根目录创建一个名为PXA270的目录：

[root@root]# mkdir /PXA270

将Toolchain内容拷贝到该目录下：

[root@PXA270]# cp Toolchain /PXA270

将Toolchain目录下的xscalev1拷贝到/opt目录：

[root@PXA270/Toolchain]# cp xscalev1 /opt

通过设置/root/.bash_profile，使Toolchain下的编译工具在任何目录下都能使用：

[root@ root]# vi /.bash_profile

用VI编辑器打开/root/.bash_profile文件并添加下述的路径：

PATH= $ PATH:/opt/xscalev1/bin

(2)创建JTAG。

JTAG的一个最主要的功能就是将Bootloader烧写到flash中，在这里使用编译后生成的Jflashmm将Bootloader烧写到flash中。

进入PXA270/Jflash目录下：

[root@root]# cd /PXA270/Jflash

将/PXA270/Bootloader/Boot - PXA270目录下的boot映像拷贝到/PXA270/Jflash目录下：

[root@Jflash]# cp /PXA270/Bootloader/Boot - PXA

270/boot /PXA270/Jflash

(3)烧写boot。

烧写Bootloader到目标板：

[root@Jflash]# ./jflashmm boot PXA270

4.2 引导程序的扩展功能

为了保证引导程序Bootloader在PXA270硬件平台的正常运行，文中对Bootloader功能进行扩展，用于测试PXA270基本硬件，包括：Flash和Sdram。

存储系统包括外置的32MB Flash和128MB Sdram，主要完成以下两部分的测试[10]：

(1)数据总线的测试。

选择有代表性的写入值0b1(单位)，0b11(2个相邻位)，0b111(3个相邻位)，0b1111(4个相邻位)。将0b1写入内存地址，将写入值与读出值相比较，相等则写入值逻辑左移一位，作为下一次写入值。如不相等则报错，将写入前后的值显示出来。

(2)地址总线的测试。

取测试值0xaaaaaaaa，按一定方法选择测试的地址，分别写入测试值，比较写入前后值。然后再取其反值为测试值写入测试地址测试。改换测试地址，按地址从低向高的顺序依次写入1的递增值，比较写入前后的值；再按上一步的地址顺序写入1递增值反值，比较写入前后的值；在每步的写入前后值的比较中如果不一致的话，就输出出错的地址和写入前后的值。

5 结束语

根据引导程序的运行机理，在不改变Bootloader结构的前提下，修改与PXA270硬件的相关代码，使之成功引导嵌入式操作系统。通过分析引导程序的启动流程和运行机理，设计并实现了以PXA270为处理器的嵌入式系统的引导程序Bootloader。目前移植的Bootloader能够稳定运行于嵌入式平台，顺利引导Linux操作系统，达到了嵌入式系统设计要求。

参考文献：

[1] 韩艳芬，吴援明，王斯瑶，等.一种二次Bootloader升级和回退的设计与实现[J].计算机技术与发展，2009，19(10)：90-91.

[2] Intel PXA270 Processor Family Developer's Manual[M]. USA：Intel，2004.

[3] 陈章龙.嵌入式技术与系统—Intel Xscale结构与开发[M].北京：北京航空航天大学出版社，2004.

[4] 刘　淼.嵌入式系统接口设计与Linux驱动程序开发[M].北京：北京航空航天大学出版社，2006.

[5] Suthar S. U - boot Porting guide[J]. Einforchips technology，2007(4)：6-7.

[6] 白伟平，包启亮.基于ARM的嵌入式Bootloader浅析[J].微计算机信息，2006，22(4)：99-100.

[7] 黄振华，李外云，刘锦高.ARM的位置无关程序设计在Bootloader中的应用[J].单片机与嵌入式系统应用，2008(1)：22-23.

[8] 周　慰.基于Xscale处理器的嵌入式硬件平台设计与引导程序研究[D].西安：西安电子科技大学，2006.

[9] 王文东，李竹林，尚建人.汇编语言与C语言的混合程序设计技术[J].计算机技术与发展，2006，16(8)：19-20.

[10] Common Flash Memory Interface Specification Release2. 0 [M]. [s.l.]：AMD，2001.