基于 PowerPC处理器 SMP系统的 U - Boot移植

李相国[1,2] 杨树元[1]

([1]中国科学院声学研究所 北京 100190 [2]中国科学院研究生院 北京 100039)

摘要:引导加载程序是计算机系统加电后运行的第一段程序代码,负责系统硬件的初始化和加载操作系统,它和系统硬件结构密切相关。本文针对一个基于 PowerPC处理器的对称多处理系统平台,进行了 U - Boot移植。并介绍了对称多处理系统的不同之处和移植方法。

关键词: PowerPC 对称多处理 U - Boot 移植

U - Boot Porting for SMP System Based on PowerPC

LI Xiangguo[1,2], YANG Shuyuan[1]

([1] Institute of Acoustics, Chinese Academy of Sciences, Beijing, 100190, China;
[2] Graduate School of the Chinese Academy of Sciences, Beijing, 100039, China)

Abstract: The bootloader is the first piece of program run after a computer is powered on, which is mainly aimed at initializing system hardware and booting up operating system, and it is highly dependent on hardware. This paper introduces the U - Boot porting process for a SMP system based on PowerPC, also discusses the characteristic of SMP system and the according porting method.

Keywords: PowerPC, SMP, U - Boot, Porting

1 前言

引导加载程序 (Bootloader)是系统加电后运行的第一段程序代码,其作用是初始化系统硬件、建立系统的地址映射,将系统的软硬件环境带到一个合适的状态,为最终加载操作系统内核准备好正确的环境[1]。在 PC机系统上,该功能由保存在 EEPROM或 Flash中的 BIOS代码和硬盘主引导记录 (MBR)上的加载程序 (NTLDR、GRUB、LILO等)共同完成;在嵌入式系统中,该功能一般由单一程序来完成,常见的有 U - Boot、RedBoot等。U - Boot是采用 GPL版权的自由软件,目前能够支持 ARM、PowerPC、MIPS、x86等多种处理器构架,支持上百种系统平台,能够引导加载 Linux、VxWorks、QNX等多种操作系统。

作为系统运行的第一段代码,引导加载程序和系统硬件密切相关,因此,不同的系统硬件组成结构需要不同的代码,即便采用同一 CPU而系统的资源配置不相同,都需要不同的配置代码。对称多处理 (SMP)系统是并行处理系统中的重要结构,目前,伴随着多核处理器的在 PC机上的普及, SMP系统得到广泛应用;在一些高性能嵌入式系统中, SMP也开始得到应用。 SMP系统和单核处理器系统的 Bootloader在实现上有细微差别。

本文以一个基于 PowerPC处理器的 SMP系统为例,介绍了该系统的组成结构,分析了 U - Boot的启动过程,并详细介绍了基于该系统平台的 U - Boot移植步骤。

2 硬件平台简介

2.1 硬件平台简介

该系统的组成框图如图 1所示。

系统采用 Freescale半导体公司的高性能 PowerPC处理器 MPC7448。该芯片采用高性能的超标量 e600内核,并集成了矢量运算单元 AltiVec,芯片集成了32KB的指令 数据一级缓存和 1MB的二级缓存,目前最高主频可达 1.7GHz。系统使用 Tsi109作为主桥。Tsi109是 Tundra公司专为 PowerPC处理器设计的主桥芯片,提供了处理器接口、DDR2内存接口、HLP接口、PCI/X总线接口,并集成了千兆以太网控制器和串口,集成了时钟发生器,可以为处理器接口、内存接口和 PCI/X接口提供时钟信号,简化了系统设计。该平台充分利用了 Tsi109提供的接口功能,在 PCI/X总线上通过 PCI/X - VME桥接芯 Tsi148扩展了 VME总线,能够支持 2eSST协议。

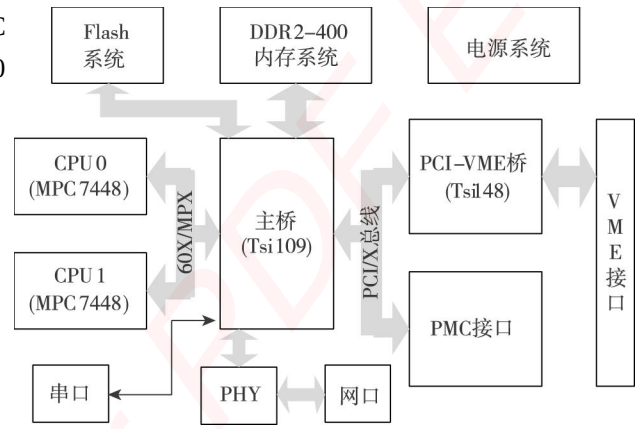

图 1 系统结构图

系统上电后,MPC7448在完成内部寄存器初始化后开始向 0x0_FFF0_0100位置读取第一条外部指令,这时的处理器接口读取命令被 Tsi109自动引导到 HLP接口与 HLP_CS0连接设备上。为了兼容 Tsi107和 Tsi108, Tsi109支持两种映射地址: 0x0_0FF0_0100和 0x0_0000_0100。必须在上电时进行设定。

3 U - Boot启动过程分析

3.1 U - Boot目录结构简介

U - Boot是自由软件,其源代码开放的特点使得它非常适合嵌入式系统的移植。目前由德国 DENX软件工程中心 Wolfgang Denk工程师负责版本维护。U - Boot具有和 Linux源文件相似的目录结构,如图 2所示。

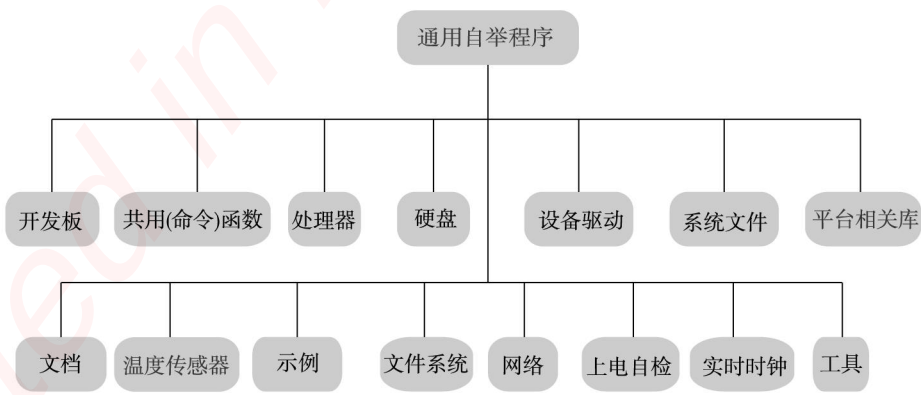

图 2 U - Boot目录结构示意图

通过目录名就可以大致了解目录内容,不再介绍。其中,和移植相关的目录主要包括顶层目录、board、cpu、include和 lib_ppc等。

3.2 U - Boot启动过程简述

如前所述,Bootloader一般具备两个基本功能:初始化系统硬件和加载操作系统。从实现过程上看,现代

多数 Bootloader都由多级组成,U - Boot也不例外。可以把 U - Boot的执行过程分为两个阶段: Stage1和 Stage2。 Stage1代码在系统 Flash内执行, Stage2代码在系统内存中执行。 Stage2能够执行功能更复杂初始化程序,同时执行速度更快。在 Stage1和 Stage2之间有一个对自身的加载程序。U - Boot执行流程如图 3 所示。

(1)U - Boot的函数入口

在 board目录下每一个平台子目录内,都有一个链接脚本文件 u - boot. lds,从中可以知道 U - Boot的函数入口:对于使用 MPC7448 系统平台来说,其函数入口地址位于 . /U - Boot/cpu/74xx_7xx/ start. S文件, start. S是一个汇编程序。

函数入口处代码为:

```
. = EXC_OFF_SYS_RESET
. globl _start
_start:
li r21, BOOTFLAG_COLD /* Normal Power - On: Boot from FLASH */
b boot_cold
sync
```

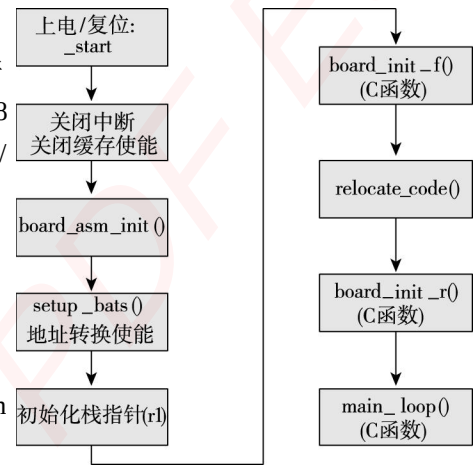

图 3 U - Boot执行流程

函数执行开始后,一个立即数读取指令后就是一个跳转语句。这是因为:MPC7448的上电 /复位处理通过调用系统复位中断 (System reset偏移向量 0x00100)来实现 [2]。每个中断向量拥有 256字节空间,不足以装下全部代码,为了代码结构清晰,并不使用完所有可用空间即跳转到中断向量空间外执行。

(2) Stage1代码

系统初始化的第一件事就是对 CPU进行关闭所有中断响应,关闭缓存使能,关闭 MMU地址转换使能等操作,为系统创建一个干净可靠的初始环境。这主要对 CPU的硬件实现相关寄存器 0(HID0)和机器状态寄存器 (MSR)置零,并对缓存、MMU相关寄存器清零。在 bootloader阶段 PowerPC运行在特权模式下,可以对所有寄存器进行操作。

然后调用 board_asm_init()函数。该函数主要进行 Tsi109内存控制接口的初始化,使系统能够尽快使用内存系统。内存接口初始化完成后,打开内存接口使能,然后,设置处理器接口的 PB_SDRAM_BAR寄存器,映射 CPU接口的系统内存空间范围。

board_asm_init()是汇编程序。为了能够执行功能复杂的 C程序,需要初始化系统栈指针。在 PowerPC嵌入式应用二进制接口规范中 [3],规定由通用寄存器 1(GPR1)来充任栈指针,因此,对 GPR1设置一个高位内存地址。

board_init_f()是一个 C函数。该函数主要为了打通终端控制接口 (串口),使 U - Boot能够尽早向外界打印信息,方便调试;还进行内存空间的分配,为代码搬运到内存执行做好准备。该函数仍在 Flash内执行,系统全局变量为只读。

relocate_code()函数是一个搬运程序,把 U - Boot代码从 Flash复制到系统内存中。该函数结束时,直接跳转到内存对应位置继续执行。

(3) Stage2代码

函数 board_init_r()是主要的初始化程序。在函数内调用 board_early_init_r(),对 Tsi109进行各接口的地址映射寄存器的正确配置,完成系统的地址映射。进行 flash命令初始化,进行设备初始化等工作。

系统初始化结束,进入 main_loop()命令行状态,或按照内核启动参数自动加载操作系统。至此,U - Boot启动完毕。

3.3 SMP系统的启动过程

在 bootloader阶段,大多数 SMP系统的启动过程都由一个处理器来完成,其它处理器处于待命状态。在 PowerPC平台规范中[4],负责启动的处理器称作主处理器(Master),其余为从处理器(Slaves)。主处理器完成操作系统初始化后激活从处理器,从处理器走快速通道跳转到其操作系统入口进行初始化。因此,从 bootloader角度看,SMP和单核处理器系统没有什么本质区别,但是,系统必须进行"主"/"从"处理器设定,并在 bootloader中给出不同的执行路径。

4 SMP系统的 U-Boot移植

4.1 开发环境的建立

嵌入式系统的开发一般需要交叉开发环境。本文工作在 Fedora 7环境下使用 DENX提供的开发套件 ELDK进行。首先安装并配置 ELDK开发环境;下载最新 U-Boot源码包,解压缩到 eldk/usr/src下;配置串口通信程序 kermit。

4.2 U-Boot的移植过程

(1)确定系统的地址映射

地址映射表反映系统资源。对系统地址映射进行正确配置,是 U-Boot初始化工作的重要内容,是系统初始化后正确运行的前提。本文平台的几个重要地址范围为:

系统内存空间:0x0_0000_0000～0x0_1FFF_FFFF

系统 flash空间:0x0_FFF0_0000～0x0_FFF7_FFFF

Tsi109寄存器:0x0_C000_0000～0x0_C000_FFFF

(2)添加目标板文件

移植工作主要是在参考板基础上进行修改,本文使用 HPC II作为参考平台。进入到 board目录下创建目标板文件夹(本文使用 smp7448):cp -r mpc7448hpc2 smp7448;然后进入到 smp7448目录内,修改文件名:mv mpc7448hpc2.c smp7448.c;然后在 include/configs目录下创建目标板配置文件:cp mpc7448hpc2.h smp7448.h;最后在顶层目录 Makefile文件内添加目标板信息:

```
smp7448_config: unconfig
@ $(MKCONFIG) $(@:_config=) ppc 74xx_7xx smp7448
```

注意,第二行开始空位是一个 <TAB>键。

(3)代码修改

首先修改 include/configs/smp7448.h配置文件。修改目标板打印信息;主要根据地址映射表进行相应参数设置,包括 Tsi109的接口参数和 CPU的 BATs映射参数等;配置系统资源参数;设置 flash命令等等;

然后根据前述执行流程进行修改。函数 board_asm_init()位于 board/smp7448/asm_init.S,本平台使用内存颗粒构建内存系统,没有内存条上 SPD可用,需要仔细检查参数数值。函数 board_init_f()位于 lib_ppc/board.c内,这是个通用文件,一般不需要做修改。board_init_r()位于 lib_ppc/board.c中,提供一个平台无关框架,一般不需要修改;board_early_init_r()是修改的重点,设置系统资源对 CPU和 PCI接口的映射参数,位于 board/smp7448/tsi108_init.c中。

设备驱动初始化是系统初始化的另一部分内容,但 bootloader关心最小系统,完整的设备初始化将由操作系统完成。系统 flash的驱动可以通过标准驱动函数 drivers/cfi_flash.c完成,串口和网口集成在 Tsi109上,参考平台原设置可以使用。由于网口部分 PHY和参考板不同,需要适当修改 drivers/tsi108_eth.c文件。

(4)添加 SMP支持代码

本文平台的"主"/"从"处理器确定通过电路硬件实现,需在 U-Boot代码中设置不同的执行路径:在 board_asm_init()函数内添加基于处理器识别的分支处理语句,让"主"处理器正常执行,"从"处理器跳转到

其操作系统入口。从处理器的 linux入口地址为标号 __secondary_hold,地址 0xc4。

4.3 U-Boot的编译

对 U-Boot的编译,通过三个命令完成:

make distclean

make smp7448_config

make

编译成功后,生成 u-boot、u-boot.bin、u-boot.map和 u-boot.srec四个文件,其中 u-boot.bin用于下载,可以通过编程器离线烧写或者仿真器下载。板子加电后,结合仿真器和 U-Boot打印信息,进行分析和调试,直至能够正确启动。

5 结束语

Bootloader是系统上电后执行的第一段程序,它和系统硬件配置密切相关。目前,SMP系统得到越来越普遍的应用,而且,SMP具有和单核处理器系统不同之处,在 bootloader中需要适当处理。本文以一个基于 PowerPC的 SMP平台为例,重点分析了 U-Boot的启动过程,以及相应的移植过程。分析了 SMP系统的启动过程和不同之处,并介绍了在 U-Boot中移植方法。本文可以为嵌入式 SMP系统的 bootloader移植提供参考。

参 考 文 献

1 詹荣开."嵌入式系统 BootLoader技术内幕". http://www.ibm.com/developerworks/cn/linux/l-btlouder/index.html 2003.

2 Freescale, "Programming Environments Manual for 32-Bit Implementations of the PowerPC(tm) Architecture", Rev. 3, 2005.

3 S. Sobek and K. Burke, "PowerPC Embedded Application Binary Interface: 32-Bit Implementation", Version 1.0, 1995.

4 Power.org, "Power.org(tm) Standard for Power Architecture(tm) Platform Requirements (Workstation, Server)", Version 2.0, 2006.

作者简介

李相国,男,博士研究生,主要研究方向为嵌入式系统和信号处理。

杨树元,男,研究员,博士生导师。研究方向为模式识别、数字图像处理、大规模并行处理及 VLSI信号处理。

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
10. 基于 CPCI 标准的 6U 信号处理平台的设计
11. USB30 电路保护
12. USB30 协议分析与框架设计
13. USB 30 中的 CRC 校验原理及实现
14. 基于 CPLD 的 UART 设计
15. IPMI 在 VPX 系统中的应用与设计
16. 基于 CPCI 总线的 PMC 载板设计
17. 基于 VPX 总线的工件台运动控制系统研究与开发
18. PCI Express 流控机制的研究与实现
19. UART16C554 的设计
20. 基于 VPX 的高性能计算机设计
21. 基于 CAN 总线技术的嵌入式网关设计
22. Visual C 串行通讯控件使用方法与技巧的研究
23. IEEE1588 精密时钟同步关键技术研究
24. GPS 信号发生器射频模块的一种实现方案
25. 基于 CPCI 接口的视频采集卡的设计
26. 基于 VPX 的 3U 信号处理平台的设计
27. 基于 PCI Express 总线 1394b 网络传输系统 WDM 驱动设计
28. AT89C52 单片机与 ARINC429 航空总线接口设计
29. 基于 CPCI 总线多 DSP 系统的高速主机接口设计

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
14. 基于 MPC8270 的 VxWorks BSP 的移植
15. Bootrom 功能改进经验谈
16. 基于 VxWorks 嵌入式系统的中文平台研究与实现
17. VxBus 的 A429 接口驱动
18. 基于 VxBus 和 MPC8569E 千兆网驱动开发和实现
19. 一种基于 vxBus 的 PPC 与 FPGA 高速互联的驱动设计方法
20. 基于 VxBus 的设备驱动开发
21. 基于 VxBus 的驱动程序架构分析
22. 基于 VxBus 的高速数据采集卡驱动程序开发

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
7. Linux 串口编程实例
8. 基于 Yocto Project 的嵌入式应用设计
9. Android 应用的反编译
10. 基于 Android 行为的加密应用系统研究
11. 嵌入式 Linux 系统移植步步通
12. 嵌入式 CC++语言精华文章集锦
13. 基于 Linux 的高性能服务器端的设计与研究
14. S3C6410 移植 Android 内核

15. Android 开发指南中文版
16. 图解 Linux 操作系统架构设计与实现原理（第二版）
17. 如何在 Ubuntu 和 Linux Mint 下轻松升级 Linux 内核
18. Android 简单 mp3 播放器源码
19. 嵌入式 Linux 系统实时性的研究
20. Android 嵌入式系统架构及内核浅析
21. 基于嵌入式 Linux 操作系统内核实时性的改进方法研究
22. Linux TCP IP 协议详解
23. Linux 桌面环境下内存去重技术的研究与实现

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]
16. 基于 WinCE 的 BootLoader 研究

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

6. 基于 PowerPC 的单板计算机的设计
7. 用 PowerPC860 实现 FPGA 配置
8. 基于 MPC8247 嵌入式电力交换系统的设计与实现
9. 基于设备树的 MPC8247 嵌入式 Linux 系统开发
10. 基于 MPC8313E 嵌入式系统 UBoot 的移植

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
9. ARM Linux 在 EP7312 上的移植
10. ARM 经典 300 问
11. 基于 S5PV210 的频谱监测设备嵌入式系统设计与实现
12. Uboot 中 start.S 源码的指令级的详尽解析
13. 基于 ARM9 的嵌入式 Zigbee 网关设计与实现
14. 基于 S3C6410 处理器的嵌入式 Linux 系统移植

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
7. 使用 COMExpress Nano 工控板实现 IP 调度设备
8. 基于 COM Express 架构的数据记录仪的设计与实现
9. 基于 COM Express 的信号系统逻辑运算单元设计
10. 基于 COM Express 的回波预处理模块设计
11. 基于 X86 平台的简单多任务内核的分析与实现

12. 基于 UEFI Shell 的 PreOS Application 的开发与研究