基于 ARM 的嵌入式系统 Bootloader 的设计与实现①

Design and Implementation of Bootloader Based on ARM Embedded System

潘孝帮 刘连浩 (中南大学 信息科学与工程学院 湖南 长沙 410083)

摘 要： 为了解决在设计 Bootloader 时调试程序不方便的问题，作者自行设计了 ARM9 核心板，扩展和增加了 SD 卡具体硬件模块上的功能支持。设计并实现了一个支持从 SD 卡加载系统镜像并启动系统的 Bootloader。在启动模式、中断处理、硬件初始化以及内存映射和最终引导 linux 内核等一系列关键技术上做了详细的设计。该设计已成功运用到湖南省科技厅一款手持导航设备项目中，运行稳定。

关键词： ARM9 Bootloader linux SD 卡

Bootloader 常常是嵌入式系统研发中可能遇到的第一个技术难点[1]。应用程序运行环境能否正确构建,内核能否成功启动,都取决于 Bootloader 能否正确的工作。本文在基于 ARM9 嵌入式系统的硬件平台与 linux 嵌入式操作系统基础上,描述了系统引导程序的设计原理,阐述了设计时应考虑的因素和需解决的技术难点并给出了一套可行的引导程序流程。在设计 ARM 核心板时，扩展和增加了 SD 卡具体硬件模块上的功能支持，以方便开发人员进行开发和调试。

笔者在参与湖南省科技厅一款手持导航设备项目研发中,设计了一款基于 S3C2410 的 ARM9 核心板,并为该设备设计了 Bootloader。目前,该 Bootloader 在核心板上运行稳定，完全实现了设计目的，达到了嵌入式系统的设计要求。

1 ARM9核心板设计

作者设计的 ARM9 核心板[2]主要由 CPU、电源管理、存储单元、SD 卡接口和串口五部分组成。 图 1 是其结构图:

CPU 采用的是 Samsung 的 S3C2410 微处理器[2],它是由 Samsung Electronics Co,Ltd 为手持设备设计的低功耗、高度集成的基于 ARM9TDMI 核的微处理器,采用精简指令系统(RISC)和 5 级流水线结构,且具有丰富的内置部件。电源管理部分采用开关电源将 AC220V 转换成 DC5V。存储单元由 Flash 存储器和 SDRAM 存储器构成。Flash 存储器主要用来存放已经调试好的应用程序、嵌入式操作系统和需要保存的用户数据。SDRAM 存储器选用的是 SAMSUNG 公司的 8MB K4S641632H,16 位数据宽度,工作电压 3.3V。由于 SDRAM 执行速度比较快，其通常作为系统运行时的主要区域，用来存放系统及用户数据、堆栈等。S3C2410[2]内部具有丰富的系统外围设备控制器，包括 SD 卡控制器。SD 卡支持 SPI 和 BUS 两种接口模式，本系统采用 SPI 接口模式。

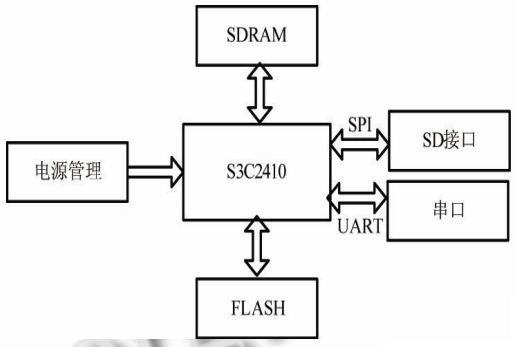

图 1 ARM9 核心板结构图

① 收稿时间:2008-09-20

© 中国科学院软件研究所 http://www.c-s-a.org.cn

2 Bootloader设计

2.1 Bootloader 的操作模式(OperationMode)

Bootloader 一般要实现两种启动模式[3]：启动加载模式和下载模式。

(1)启动加载(Bootloading)模式：也称为“内核启动”模式。

(2)下载(Downloading)模式:在这种模式下，目标机上的Bootloader将通过串口或网络连接或者USB等从主机下载操作系统文件,也可以直接从 SD/MMC 卡、CF 卡等通信手段从主机(Host)下载内核映像和根文件系统映像等，然后保存到目标机上的 Flash 类固态存储设备中，也可以将 Flash 中的映像文件上传到主机。

在我们的 Bootloader 设计中我们同时支持这两种工作模式，采用的方法是：一开始启动时处于正常的启动加载模式，但并不立即启动进入 linux 内核，而是提示延时 8 秒，等待终端用户如果按下某一特定按键，则切换到下载模式，否则继续启动 linux 内核。

2.2 Bootloader 的启动及初使化

由于 BootLoader 的实现依赖于 CPU 的体系结构。因此，为了方便移植和增加系统的执行效率一般分为两个阶段 stage1 和 stage2。stage1 用汇编语言写，主要进行与 CPU 与存储设备相关的工作进行必要的初始化工作，是一些依赖于 CPU 体系结构的代码，初始化 CPU 运行的时钟频率，初始化 Flash 和内存的数据宽度、读写访问周期和刷新周期，初始化中断系统,初始化系统中各种片内片外设备和 I/O 口，初始化系统各种运行模式下的寄存器和堆栈。stage2 是用 C 语言实现一般的流程以及对板级驱动的支持，这样设计代码具有很好的移植性和可读性，对于相同的 CPU 只需修改 stage2，对于不同的 CPU 只需修 stage1。

3 stage1设计

3.1 建立二级中断向量表

ARM 处理器的中断向量表从地址 0x0 处开始存放，连续有 8×4 字节的空间。在 ARM 存储空间里每个字 32 位，占 4 个字节。每当有中断或者异常发生时，ARM 处理器便强制把 PC 指针指向向量表中对应中断类型的地址值，但是这样中断响应时间相对较慢。为了加快中断响应，我们在 Flash 的 0x0 地址存放能跳转到 0x0c000008 地址处中断向量的跳转指令，即在 RAM 中建立一个二级中断向量表，起始地址为 0x0c000008，除复位外，其它异常入口地址由 Flash 跳转得到。如图 2 所示：

B ResetHander	0x0c000008	b ResetHander	Reset vector
LDR PC, =0x0c00000c	0x0c00000c	b HanderUndef	Undefined
LDR PC, =0x0c000010	0x0c000010	b HanderSWI	SWI
LDR PC, =0x0c000014	0x0c000014	b HanderPabort	Prefetch abort
LDR PC, =0x0c000018	0x0c000018	b HanderDabort	Data abort
LDR PC, =0x0c00001c	0x0c00001c	b HanderAbort	Reserved
LDR PC, =0x0c000020	0x0c000020	b HanderIRQ	IRQ
LDR PC, =0x0c000024	0x0c000024	b HanderFIQ	FIQ

图 2　二级中断向量表

4 stage2设计

4.1 可执行映像 stage2 的入口

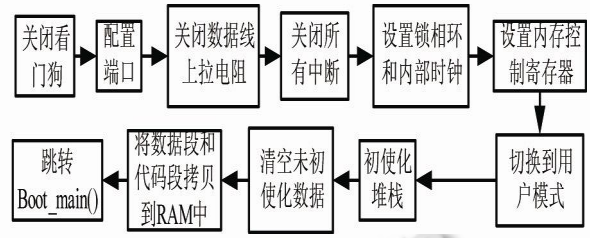

图 3　硬件初使化流程图

如图 3 所示，这一段代码是用汇编完成的，其后的代码都是用 C 语言编写的。这样的设计有利于 Bootloader 的移植。跳转到 boot_main 最简单的实现方法是直接把 boot_main()函数的起始地址作为执行映象的入口点。但是这样做会有两个缺点：一是无法通过 boot_main()函数传递函数参数；二是无法处理 boot_main()函数返回的情况。笔者在实现的过程中采用了弹簧床的概念。实现代码如下:

```
.text
.globl_trampoline
_trampoline
bl boot_main
b _trampoline
```

这样，在转到 C 语言编写的 boot_main 函数之前，可以通过寄存器 R0～R3 来传递参数，参数返回使用寄存器 R0 和 R1。当 boot_main 函数返回的时

© 中国科学院软件研究所 http://www.c-s-a.org.cn

候，实际上是重新再次调用 boot_main 并执行。程序也可以根据需要，在 boot_main 函数返回后做相应的后续处理(如通知用户等)。这样就完全克服了直接跳转的两个缺点。

4.2 内存映射

一般 S3C2410 上配置的 SDRAM 大小为 64M，该 SDRAM 的物理地址范围 0x30000000~0x33FFFFFF(属于 Bank6)，由于 1 个 Section 的大小是 1M，所以该物理空间可以被分成 64 个物理段(页框)。由于 Bootloader 没有对 MMU 的管理代码，处理器在运行时直接访问物理地址。同时，因为 ARM 体系结构中数据缓冲(Dcache)必须通过 MMU 开启，所以 Bootloader 效率比较低，可通过平板映射(flat，既虚拟地址和物理地址相同)方式开启 MMU，从而使用内存空间的 Dcache，提高 Bootloader 的运行速度。如图 4 所示：

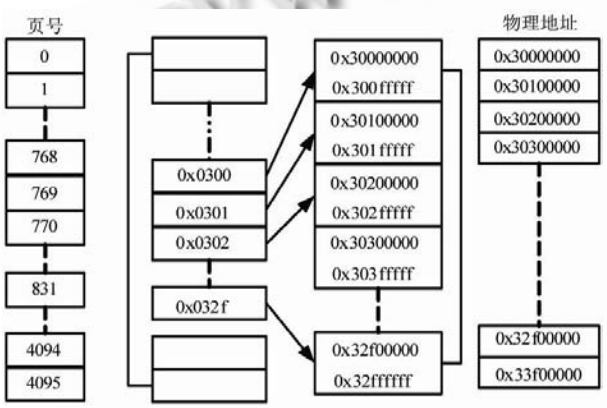

图 4　平板映射关系图

映射关系代码如下：

```
Void mem_mapping_linear(void)
{unsigned long descriptor_index, section_base,
sdram_base,sdram_size;
    sdram_base=0x30000000;
    sdram_size=0x4000000;
    for(section_base=sdram_base,
descriptor_index=sec-tion_base>>20;
    Ssection_base<sdram_base+sdram_size;
    descriptor_index+=1;
    section_base+=0x100000)
    {*(mmu_tlb_base+(descriptor_index))=(sect
ion_base>>20)|MMU_OTHER_SECDESC;}
}
```

4.3 通过 SD 卡加载系统镜像的实现

当用户选取 SD 卡[5]作为下载系统镜像的目标后，Bootloader 就进入对 SD 处理的流程。首先通过 SD 卡检测引脚判断是否有 SD 卡插在插槽。如果有就要对 SD 控制的硬件进行初始化。

接着是对 SD 卡协议栈[5]的软件实现，为了减少 Bootloader 中 SD Host 驱动的复杂性，能够易于调试，实现总线驱动模块、客户端驱动模块和 FAT16 文件系统模块，所以在我们的实现中对 SD 卡上的文件系统要有一定的限制，必须是格式化成 FAT16 的文件系统才能被我们的 Bootloader 识别。其协议栈结构如图 5 所示：

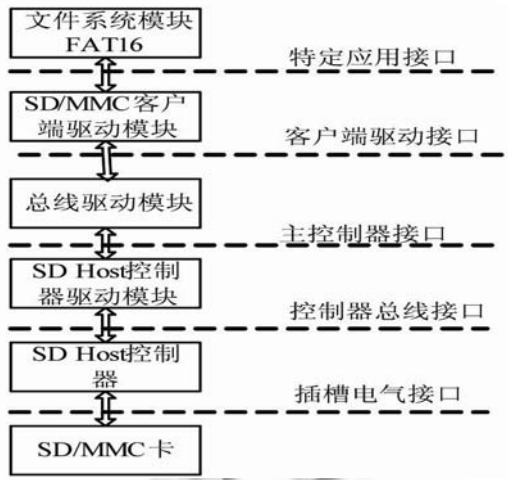

图 5　SD 卡的协议栈

4.4 仿真实验结果

我们使用一个 linux 镜像文件，在 Bootloader 中分别用 USB、SD 卡、TFTP、本地四种使用方式进行加载启动测试，分别测试了 100 次。由于在 Bootloader 中 SD Host 的实现没有使用 DMA 方式，为了进行性能比较，我们又调用 linux 系统下使用 DMA 的 SD Host 驱动加载同样大小的文件进行了 100 次测试。图 6 是我们测试的结果。图 6 中数据为平均值。

	SD卡	USB	TFTP	使用DMA的SD卡	本地启动
加载时间(秒)	49	100	56	28	5

图 6　仿真测试结果

© 中国科学院软件研究所 http://www.c-s-a.org.cn

从表中数据可知，本地启动是最快的。其次是通过 SD 卡启动。而 SD 卡的数据通信传输完全由硬件实现，这也是 SD 卡加载相对比较快的主要原因。而且使用 TFTP 下载还需要进行相对复杂的配置。而在使用 SD 的加载中 Bootloader 会自动去搜寻系统镜像，自动下载。这对用户的使用来说是方便快捷的。另外从使用 DMA 的 SD 驱动下载文件和 Bootloader 中的实现做比较，可以看出使用 DMA 后大大提高了使用 SD 下载的性能。当然我们也可以在 Bootloader 中使用 DMA 方式来实现以提高性能。但这样一来会大大增加 Bootloader 的复杂性。我们在 Bootloader 中实现使用 SD 做加载启动的主要目的是方便开发和调试 SD 硬件模块，我们的实现中目的已经达到。

5 结束语

本论文创新观点是：为了解决设计 Bootloader 时调试程序不方便的问题，设计了具有较高性价比的 ARM9 核心板，该 ARM 板支持从 SD 卡加载并启动 linux 系统的 Bootloader，能提高 linux 操作系统移植的稳定性并加快 linux 操作系统移植的周期。在设计 Bootloader 时采用延时的方法，进行两种加载模式的切换；提出了在 RAM 中建立二级中断向量表的响应中断的方式，增强了系统实时性，加快了系统响应时间；在 Stag1 跳转到 boot_main 时采用了弹簧床的概念，克服了直接跳转的缺点；在内存映射方面，采用了平板映射的方式，提高了 Bootloader 的运行速度。并且在 Bootloader 中实现了 SD 卡的协议栈，实现了从 SD 卡加载并启动 linux 嵌入式系统镜像的功能，提高了开发效率。

该 Bootloader 具有典型代表性，良好的健壮性和可移植性，对于相同的 CPU 只需修改 stage2，对于不同的 CPU 只需修 stage1。对进一步开发复杂系统的 Bootloader 具有很好的借鉴和启发作用。该设计已成功运用到湖南省科技厅一款手持导航设备项目中，运行稳定。

参考文献

1 王田苗.嵌入式系统设计与实例开发—基于 ARM 微处理器与 uC/OS 实时操作系统.北京:清华大学出版社, 2006:5－10.

2 SUMSUANG ELECTRONICS.S3C2410 User's Manual. Republic of Korea: Sumsang, 2006: 25－36.

3 Henkel J, Hu XBS, Bhattacharyya SS. Taking on the Embedded System Design Challenge, IEEE Computer, 2006,(4):35－37.

4 王学龙.嵌入式 Linux 系统设计与应用.北京:清华大学出版社,2006:93－126.

5 SD-Memory Card Specifications /Part1 Physical Layer Specification Version 1.01.SD Group, 2007:12－15.

© 中国科学院软件研究所 http://www.c-s-a.org.cn