基于 ARM11 的精简 BootLoader 的设计

汪猛 程翔

（厦门大学 物理与机电工程学院，福建 厦门 361005）

摘 要：嵌入式系统一般有三个部分构成:系统引导加载程序、嵌入式操作系统内核以及根文件系统。由于嵌入式系统引导加载程序严重依赖于 CPU 体系结构和嵌入式板级设备的配置，所以没有一个通用的嵌入式引导加载程序。嵌入式系统引导加载程序一般分为二个阶段，第一阶段主要是和 CPU 体系结构相关的代码，第二阶段和板级硬件配置相关代码。本文基于 ARM 体系结构的硬件平台主要分析引导加载程序第一阶段，进而研究引导加载程序的实质。

关键词：中断向量表； BootLoader； ARM

中图分类号：TP368.12 **文献标识码**：A **文章编号**：1672-4801(2011)03-024-03

嵌入式系统的启动设备主要有三种，分别是 NOR FLASH, NAND FLASH 以及 SD 卡，可通过配置引脚的不同状态确定启动设备。由于 SD 卡烧写方式不依赖 JTAG 和并行接口，以及其可插拔的特点，故其应用更加广泛和便捷，大大提高了调试的效率。本文研究基于 S3C6410 硬件平台，启动方式为 SD 卡启动。

1 开发环境介绍

本文基于 SC36410 硬件平台，S3C6410 是三星公司推出的基于 ARM1 微处理器的一款高端开发板，专为多媒体消费类电子产品的开发而设计。S3C6410 采用了 64/32 位内部总线架构，由 AXI、AHB 和 APB 总线组成，内置视频处理、音频处理、2D 图形、3D 图形等硬件加速器[1]。本文软件开发环境基于 Linux 操作系统 Ubuntu-9.10，ARM 嵌入式交叉工具链、scratchbox2、minicom、kermit 等软件开发包。

2 ARM 汇编程序编程要点

(1) 位置无关性代码 PIC(Position-Independent Code)

对于 C、C++以及其他类似语言编写的程序，要经过编译、汇编、链接才成为可执行程序。在链接阶段编译器会给数据、函数等分配相应的内存地址。在有操作系统的系统环境中，编译阶段分配的地址和程序运行时候的物理地址通常是不一样的，这种情况并不会产生错误。因为现代 CPU 体系结构一般都有 MMU 模块，操作系统以保护模式的方式运行而非实地址模式，提供给 CPU 使用的地址需要经过 MMU 页表映射才能得到相应的物理地址。而 Bootloader 是为正确加载操作系统前初始化硬件的一段程序，一般情况下嵌入式的 Bootloader 是没有开启 MMU 的，所以要让 Bootloader 正确运行，在链接阶段产生的地址应该和运行时候的物理地址是一致的才行。

ARM 体系架构的指令集采用的是 RISC 指令集，内存和 CPU 寄存器之间交换数据通过 LDR/STR 及其扩展指令来实现的，在链接阶段产生的地址和运行时候的物理地址不相同时，如果指令中地址值的编码计算方式是通过绝对值地址取得的时候就会产生不可预知的错误。在实际应用中，嵌入式 Bootloader 以及 Linux 启动代码采用汇编编写的程序中很多地方的链接阶段分配的地址和运行时候的物理地址不相同，但是程序却可以正确运行。这是因为这部分代码是位置无关性代码，其原理是指令所存放的内存地址是根据 PC 指令的偏移方式计算出来的，和链接阶段产生的地址无关。地址无关性代码用途很广，最常用的是程序的起始阶段通过地址无关性代码，实现代码的重定向功能。我们可根据 ARM 指令的编码方式，确定哪些是地址无关性的指令，编写出地址无关性的程序。

(2) AAPCS 规范

AAPCS 文档描述了基于 ARM 体系结构下的应用程序二进制接口的过程调用标准，该文档包含了第五次修订版的 APCS 和第三次修订版的 TPCS 规范的主要内容[2]。同时支持 Thumb 和 ARM 指令，Thumb 和 ARM 指令运行状态之间的切换，以及 ARM 体系结构高性能的执行效率是

作者简介：汪猛(1987-)，男， 硕士研究生，研究方向:嵌入式 Linux 系统。
程翔(1977-)，女，硕士生导师，研究方向:硅基光电子集成(OEIC)，嵌入式系统。

AACPS 规范的目标。在编写程序时，要想生成一致性的机器代码，必须遵循 AAPCS 的一致性要求。本文在编写程序时，在 ARM 汇编中调用了 C 语言中的函数，属于混合编程。所以要使程序能够分别编译并且链接成功，以及正确运行，那么在对寄存器的使用，函数调用参数的传递和返回，栈的保存等方面都必须按照 AAPCS 的规范来编程。

3　嵌入式引导加载程序的设计

引导加载程序是系统上电后，最先运行的一段初始化程序代码，其主要功能是初始化硬件设备并建立内存空间的映射图，为调用操作系统内核或应用程序建立正确的环境。引导加载程序一般分为两个阶段：第一个阶段主要是和 CPU 体系结构等硬件密切相关的代码，大部分用汇编语言完成；第二个阶段和板级配置相关的代码，主要用 C 语言编写。在嵌入式领域非常流行的一款开源软件是 Bootloader，然而 U-BOOT 并不支持中断的功能，其最显著特点就是主机不能 ping 通目标机。要实现这个功能，一种方式是用轮询，另一种方式是采用中断。国内一款优秀的开源软件 g-bios 在 S3C24X0、ATMEL926X 实现了中断，S3C6410 采用的是轮询的方式[3]。本文设计了一种精简的嵌入式 Bootloader，以最小的代码量来研究嵌入式的加载程序的实质。

(1) 建立中断向量表

ARM 体系结构提供了多种方法实现中断的功能，其中之一就是建立中断向量表。S3C6410 上电后，映射地址从 0x0c000000 开始，而不是从 0x00000000 地址开始的，这点和 S3C24x0 系列平台有所不同。S3C6410 设置中断向量表研究办法如下：

一种方法是开启 MMU，建立页表，实现地址重定向功能。另外，在 ARM V5 以上架构的 CPU，ARM 的中断向量表不仅可以映射在 0x00000000~0x0000001C 低地址，而且可以映射到 0xFFFF0000~0xFFFF0001C。通过设置协处理器 CP15 寄存器 1 的第十三位的值便可设定中断向量表是映射在低地址空间还是高地址空间[4]。对于 IRQ 和 FIQ 模式，ARM 架构还提供了在执行时定义这两个模式的地址映射的空间。通过设置协处理 CP15 寄存器 1 的第二十四位的值可实现 Bootloader 中断功能。然而，无论是映射在低地址空间还是高地址空间，映射的内存范围都不包含 SRAM 或者 SDRAM 芯片挂载的内存空间。由于开启 MMU 过程太过复杂，而本文的目的是设计一个精简的 bootloader，权衡考虑，放弃建立中断向量表。

(2) LED 点亮控制

在裸机环境下，调试程序是一件很困难的事情；另一方面，本实验平台没有建立 ARM 的中断向量表，所以无法捕捉到程序出现了何种异常，加大了程序调试的难度。本文通过在程序的不同阶段对 LED 点亮进行控制，对程序的调试起到了一定的辅助作用。通过对相应的 Gpio 管脚的电平写操作即可完成对 LED 的控制。

(3) 看门狗和相关总线的时钟初始化

系统加载程序需要关闭看门狗，关闭看门狗的编程和 LED 的点亮控制一样简单。S3c6410 的相关总线要比 S3c24x0 系列复杂，在编程实现的时候也相应的更复杂。如果不考虑功耗的因数，正确设置总线时钟的初始化值是很多的，一步一步按照 Datasheet 的要求实现即可。

(4) 串口编程和 kermit 串口协议实现

串口控制器的初始化包括时钟初始化、波特率的设置、数据收发以及状态寄存器的编程等。在 kermit 串口协议的实现中，数据的读写是对串口控制器的数据收发寄存器以及状态寄存器的编程。

(5) SDRAM 的初始化

本文采用的是 SD 启动方式，系统上电后，第一条指令运行在 0xc000000 地址上。S3c6410 的 0xc000000 映射的是一块 SRAM，SRAM 不需要初始化设置，通过运行在 SRAM 上的程序完成必要的硬件初始化。如果本文的加载程序加载一段非常小的程序，可以将其直接加载到 SRAM 的地址空间，不需要对 SDRAM 进行编程。如果要加载比较大的程序则必须要对 SDRAM 进行初始化，将程序加载到 SDRAM 的地址空间。

(6) 精简嵌入式 Bootloader 的实现

本文实现的嵌入式精简 Bootloader 由 s3c6410.h、s3c6410.S、kermit.c、boot.lds、Makefile 组成。s3c6410.h 包含了大量关于地址的宏定义，s3c6410.S 和 kermit.c 是核心部分，为本文第三部分的设计的实现，其中 kermit.c 为 kermit 串口协议的实现，boot.lds 为链接脚本，Makefile 为程序的自动编译脚本。

boot.lds:

```
ENTRY(start)
SECTIONS
{
    .text 0x0c000000:
    {
        *(.text);
    }
}
Makefile:
CROSS_COMPILE = arm-linux-
AS = $(CROSS_COMPILE)as
CC = $(CROSS_COMPILE)gcc
LD = $(CROSS_COMPILE)ld
OBJCOPY = $(CROSS_COMPILE)objcopy
OBJDUMP = $(CROSS_COMPILE)objdump
INPUT = s3c6410
CFLAGS = -c -nostdinc -nostdlib
LDFLAGS = -m armelf_linux_eabi
all: $(INPUT).bin
$(INPUT).bin: $(INPUT).elf
    $(OBJCOPY) -O binary –S $< $@
    cp $(INPUT).bin /tftp -v
$(INPUT).elf: $(INPUT).o kermit.o
    $(LD) $(LDFLAGS) $^ -T boot.lds -o $@
$(INPUT).o: $(INPUT).S
    $(CC) $(CFLAGS) $< -o $@
kermit.o: kermit.c
    $(CC) $(CFLAGS) $< -o $@
```

(7) 烧录和测试

用 SD 卡的烧写软件将编译出来的 s3c6410.bin 烧录到 SD 卡，将 SD 插入开发板的 SD 卡插槽准备测试。在测试前需要编写一个程序作为串口下载的测试程序，本文用 arm 汇编编写了一个精简的 hello world C 程序，作为加载测试的文件。其中字符串输出所调用的函数是对串口读写操作来实现的，不能调用标准库下的 printf 函数来输出字符串。也可以用 G-bios，U-boot 等其他嵌入式加载程序作为测试程序。

(8) 加载和运行 Linux 内核

本文设计的加载程序没有实现网卡驱动程序和 tftp 下载应用程序，只能通过串口下载内核 image，由于串口下载速度比 tftp 下载速度慢很多，所以下载需要很长的时间。ARM 体系架构要求在启动内核前，必须保证 CPU 处于 SVC 模式、IRQ 和 FIQ 中断禁止，MMU 关闭、数据 cache 关闭、r0 寄存器值为 0，r1 为机器类型，r2 为内核参数列表的物理地址[5]，所以在嵌入式加载程序的设计中要特别注意这一点。

4 结语

本文基于 ARM11 平台提出了一种较简单的方法，设计和实现了这种精简的嵌入式加载程序，对于分析和理解嵌入式加载程序的实质有一定的作用，基于此工作的基础，扩展和丰富程序的功能是下一步要做的工作。

参考文献：

[1] USER'S MANUAL S3C6410X[S], 2008: 60-63.

[2] ARM Procedure Call Standard for the ARMArchitecture[S], 2009,:9-10.

[3] MaxWit Corporation[CP]http://code.google.com/p/g-bios.

[4] ARM Architecture Reference Manual[S], 2005:54-67, 684-697.

[5] Booting ARM Linux[S], 2004:5-8.