文章编号:1671-4598(2016)09-0012-03 DOI:10.16526/j.cnki.11—4762/tp.2016.09.004 中图分类号:TP273 文献标识码:A

基于嵌入式 S3C2440 系统 Bootloader 设计与实现

范蟠果，邢保亳，米晓亮，余书宝，王 婷
（西北工业大学 自动化学院，西安 710129）

摘要：Bootloader 是嵌入式系统的一个重要环节，对不同的硬件平台，其 Bootloader 都不尽相同，因此设计 Bootloader 是嵌入式系统开发的难点；文中分析 S3C2440 嵌入式系统的硬件组成和 u—boot 源码对 linux 内核的启动流程，得出 u—boot 启动内核两个阶段必备阶段：第一个阶段是用汇编初始与具体硬件平台相关的操作等，第二阶段是用 C 语言编写复杂功能以及启动内核；以加载 linux—2.6.22.6 内核为例，根据 u—boot 启动内核两个阶段所做的工作，设计出适用于 S3C2440 嵌入式系统的精简 Bootloader；通过实验表明，该设计的 Bootloader 成功启动 linux 内核，具有良好的稳定性，可靠性和简洁性。

关键词：系统设计；分析 u—boot；实现 Bootloader；启动内核

Development and Implement of Bootloader Based on S3C2440 and Embedded Linux System

Fan Panguo, Xing Baohao , Mi Xiaoliang, Yu Shubao, Wang Ting
(College of Automation, Northwestern Polytechnical University, Xi'an 710129, China)

Abstract: Bootloader is the important part of embedded system. For the different hardware platforms, Bootloader is different. So bootloader is the difficult for the developmemt of embedded system. This paper mainly analyzes the design of embedded S3C2440 system and the process of u—boot startup kernel, drawing two essential phases of u—boot startup kernel, the first phase is to use assembler language to write some functions of the specific hardware platforms , The second phase is to use C language to write codes about some complicated functions and loading the kernel. For example, Loading the linux—2.6.22.6, according to the two stages work of u—boot startup kernel. Designing of streamlined Bootloader is suitable for the S3C2440 embedded system. Experiment shows that the design of Bootloader successfully starts the linux kernel , and has a good stability, reliability and simplicity.

Keywords: design system; analyze u—boot; realize Bootloader; load kernel

0 引言

Bootloader 是嵌入式系统内核运行前的一段程序，这段程序初始化硬件设备，并且建立一个内存空间映射图，从而建立适当的系统软硬件环境，为最终启动内核和加载文件系统做准备[1]。由于 Bootloader 依赖于硬件，它与处理器架构，具体设计的硬件平台资源相关，因此设计一个适合某平台的 Bootloader 是开发嵌入式系统重要工作。

在设计引导程序时，一般会移植 u—boot 开源代码，但是这样代码量大，占用较大存储空间；本文通过分析 u—boot 启动流程，根据硬件平台资源，设计一个精简，稳定的 Bootloader。

1 u—boot 分析

1.1 u—boot 启动流程分析

分析支持 S3C2410 的 u—boot 源代码，具体启动流程分析如图 1 所示。

根据分析得出启动过程一般分为两个阶段。

stage1：主要通过汇编来实现和硬件相关代码：硬件设备初始化，加载 u—boot 第二段代码到 RAM 空间，设置好栈，跳转到第二段代码入口[2]。

stage2：主要用 C 语言来实现一些复杂的功能：初始化本阶段使用的硬件设备，检测系统内存映射，将内核从 flash 读取到 RAM 中，为内核设置启动参数，调用内核[2]。

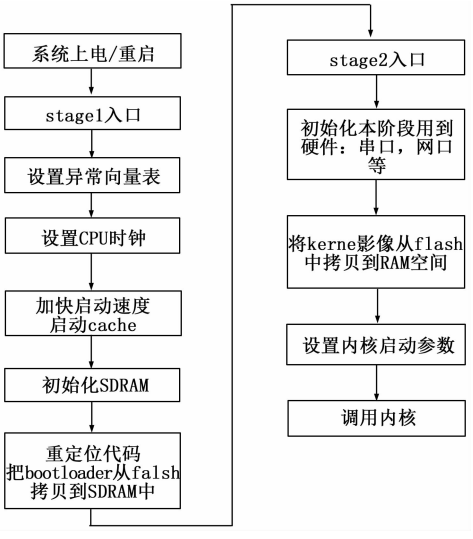

图 1 u—boot 启动流程

1.2 u—boot 编译以及大小分析

u—boot 在编译之前，需要编写好 Makefile 和连接文件 u—boot.lds，然后通过 make 命令编译生成 u—boot.bin；Makefile 规定 u—boot 所有函数的依赖关系；连接文件 u—boot.lds 指定 u—boot 编译的连接地址，第一个被编译的文

收稿日期:2016-03-30； 修回日期:2016-04-18。

作者简介:范蟠果(1960-),男,陕西西安人,硕士生导师,副教授,主要从事计算机测控方向的研究。

件，存放的代码段，数据段以及 BSS 段的位置。

如下面一个连接文件 u－boot. lds：

```
SECTIONS
{
. = 33f00000;//指定的连接地址
. text ://代码段
{
_start = . ;//代码段开始位置
arch/arm/cpu/arm920t/start. o (. text)//执行编译的第一个文件 start. S
... }
. data ://数据段
.....
__bss_start = . ;//BSS 段开始位置,此处为 u－boot. bin 大小结束位置
.....}
```

BSS 段是由静态和未初始化的全局变量组成，不会被编译到 u－boot. bin 中。

由连接文件可知 u－boot 的大小为__bss_start 地址减去_start 的地址，可以查看 u－boot 的反汇编得到具体的大小，执行 arm－linux－objdump －D u－boot ＞ u－boot. dis 命令来生成如下 u－boot. dis 反汇编文件，：{

```
Disassembly of section . text:
33f00000 <_start>:注释:代码开始位置,和连接文件中的连接地址一致
33f00000:  ea000013    b   33f00054 <start_code>
......
33f00044 <_bss_start_ofs>:
33f00044:  00069c4c    . word   0x00069c4c
.......} 注释:在启动文件 start. S 中定义一个表示 u－boot. bin 大小的全局变:
. globl _bss_start_ofs
  _bss_start_ofs:
. word __bss_start － _start
```

编译的 u－boot. bin 大小为（0x00069c4c）423 KB，占有很大存储空间。由于 u－boot 是一个支持多平台的源代码，所以它的结构复杂，若要让这个 u－boot 成功应用到 S3C2440 系统上，需要对 u－boot 源代码进行裁剪，并且添加支持平台外设的代码，工作量比较大，而且不便于调试。所以很有必要设计适用 S3C2440 系统的 Bootloader，具体设计如下。

2　系统设计

2.1　硬件设计

图 2 是一个基于 ARM 嵌入式系统硬件框图，S3C2440 芯片是基于 ARM920T 的架构处理器，片内有 4 k 的 SRAM，nandflash 控制器，SDRAM 控制器等资源。板级设备包含：全功能的串口，JTAG，Nandflash（256 M），SDRAM（64 M）等。

2.2　系统软件内存设计

1）根据 S3C2440 手册可知 SDRAM 映射在 BANK7，所以起始地址为 0x30000000；结束地址为 0x34000000。在调用 C 语言之前要设置栈指针，ARM 栈是往下增长的，设计栈指针为 0x34000000。为给内核以及文件系统留有足够的内存，设置启动代码连接地址为 0x33f80000。内核编译后大小不超过 2 M，设计内核连接地址为 0x30008000。内核把启动参数放在 0x30000100 位置处，具体内存分布如图 3。

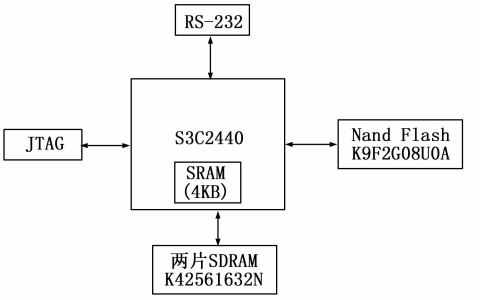

图 2　硬件平台框图

2）Nandflash 是存储程序和数据，掉电后能保存数据和程序，所以启动代码和内核都存放 nandflash 中，为了保证两者代码不覆盖，设计 Bootloader 存储的起始地址为 0x0，内核存储的起始地址为 0x60000。

根据设计的存储空间和内存分配，两者的映射关系如图 3 所示。

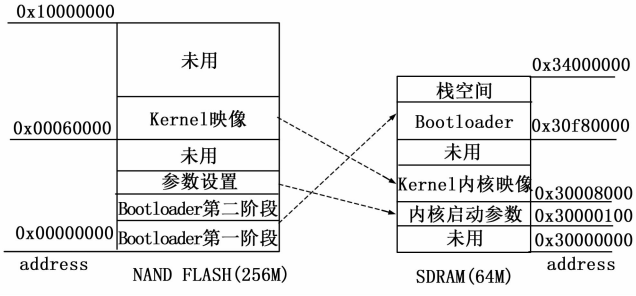

图 3　映射空间图

3）代码重定位。

根据两者映射关系，需要对代码进行重新定位，详细分析如下：

由于 cpu 对 nandflash 读取是以块（2 048 KB）为单位的，所以不能在 nandflash 上运行程序。当系统上电或者重启时，硬件自动把存储器中前 4 KB 代码拷贝到 cpu 内部 SRAM（映射地址为 0）；对于连接地址为 0x33f80000 的代码若要成功运行在地址为 0 的内存中，在设计代码时必须使用位置无关指令。通过运行在 SRAM 程序把 nandflash 中启动代码和内核拷贝到 SDRAM，然后通过一条跳转指令，使 PC 指向 SDRAM。

3　Bootlaoder 软件实现

根据分析得到 u－boot 启动流程，以及分配好的内存，下面针对 S3C2440 特定的嵌入式系统，实现一个完整的 Bootloader。

3.1　第一阶段

Stage1 主要是完成，建立向量表，初始化硬件，设置堆栈，把 Bootloader 拷贝到内存中，具体流程如图 4 所示。

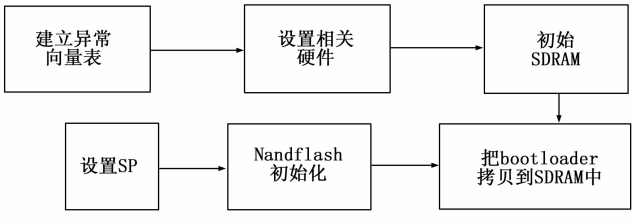

图 4　Bootloader 一阶段流程

（1）设置 ARM 异常向量表。

根据 ARM920T 的架构可知，异常向量入口地址从 0x00000000 开始到 0x0000001c 结束，分别设置对应入口地址，当一个异常或中断发生时，处理器会把 pc 指向对应中断向量的入口地址。

2）设置相关硬件。

设置时钟，使 CPU 主频为 400 MHz；屏蔽中断；初始化串口以便于调试。

3）初始化 SDRAM。

SDRAM 用来运行启动代码和内核，必须对其进行设置，SDRAM 控制器的寄存器，包含处理器对 SDRAM 读写访问的时序，数据宽度；系统使用两片 16 位数据接口的 K4S561632N，其数据宽度为 32 位。

4）设置堆栈指针。

Bootloader 在启动阶段是运行在管理模式下，只需设置管理模式下的堆栈指针（向下增长），代码为：ldr sp，=0x34000000；设置成功后就可以调用 C 语言代码。

5）nandflash 的初始化。

由于需要把存储在 nandflash 的启动代码和内核读入 SDRAM，所以要设置 nandflash 控制器的寄存器，设置 cpu 对存储器读写时序，以及数据宽度。

6）把启动代码拷贝到 SDRAM 执行。

SDRAM 配置成功后，就可以从 nandflash 拷贝 Bootloader 启动代码到内存中运行；具体代码如下：

```
mov r0,   0 /* 从 nandflash 的 0x0 地址开始拷贝 */
  ldr r1, =_start /* 这个值是由连接脚本确定为 0x33f80000 */
  ldr r2, =__bss_start /* Bootlaoder 代码结束位置 */
  sub r2, r2, r1 /* 获得拷贝的大小 */
  bl copy_code_to_sdram /  * 从 nandflash 拷贝到 SDRAM */
  ldr pc, =main / * 重新给 pc 赋值,跳转到 SDRAM 中,开始执行第二阶段代码 */
```

3.2 第二阶段

Stage2 的主要工作设置内核启动参数，拷贝内核到内存中，然后启动内核，具体流程如如图 5 所示。

图 5 Bootloader 二阶段流程

1）设置内核启动参数。

Bootloader 和内核之间参数传递是单向的，Bootloader 将参数放在某个约定地址[3]（0x30000100），再启动内核，启动后会从 0x30000100 地址处取参数。

内核启动前会检查传入的参数，比如内存的大小，命令等参数。因为 Linux2.6.22.6 内核检查参数以标记 ATAG _ CODE 开始，以标记 ATAG _ NONE 结束。详细结构体可以参考 linux 内核源码 include/asm/setup.h 头文件。

以设置参数开始 ATAG _ CODE 标记为例：

```
void setup_start_tag(void)
{
  params = (struct tag *)0x30000100; / * 约定的存放参数的起始地址 */
  params->hdr.tag = ATAG_CODE; / * 参数标记 */
  .......
  params = tag_next (params); / * 指向下一个参数 */
}
```

其他参数设置 如内存标记 ATAG _ MEM：告诉内核开发平台外设 SDRAM 内存起始地址和大小；命令行标记 ATAG _ CMDLINE：用来控制内核一些行为，结束标记 ATAG _ NONE 用来标记参数结束[3]。

最后配置的参数结束标记：

```
void setup_end_tag(void)
{ params->hdr.tag = ATAG_NONE;
}
```

设置参数具体代码：

```
puts("Set boot param\n\r"); / * 打印提示信息 */
setup_start_tag(); / * 设置起始参数 */
setup_memory_tags(); / * 设置内存参数 */
setup_commandline_tag("noinitrd root=/dev/mtdblock3 init=/linuxrc console=ttySAC0"); / * 设置命令参数 */
setup_end_tag(); / * 参数设置结束 */
```

2）把内核从 flash 中拷贝到内存的 0x30008000 地址处。

编译内核是通过执行：make uImage 生成内核影像 uImage；用 JTAG 把内核影像 uImage 下载到 nandflash 的 0x60000 处，但是 uImage 开始的 64 字节表示内核头帧数据，真正的内核是从 0x60000+64 地址开始。具体拷贝代码如下：

```
nand _ read (0x60000 + 64, (unsigned char *) 0x30008000, 0x200000);
```

第一个参数：0x60000+64 代表的是内核起始地址

第二个参数：0x30008000 代表是在 SDRAM 中内核的运行起始地址

第三个参数：0x200000 代表从 nandflash 拷贝 2M（内核大小一般小于 2M）大小程序到 SDRAM 中。

3）启动内核。

根据分析 u-boot 源程序可知；启动内核的过程就是把内核在 SDRAM 的连接地址赋值给一个能启动内核的函数指针；然后通过调用此函数指针把机器类型 ID（S3C2440 的 ID 为 362）和启动参数在 SDRAM 中的起始基地址（0x30000100）传递给内核[4]。

调用内核具体代码：

```
void (*theKernel)(int zero, int arch, unsigned int params);/ * 声明启动内核的函数指针 */
theKernel = (void (*)(int, int, unsigned int))0x30008000;/ * 把内核在 SDRAM 中起始地址赋值给函数指针 */
puts("Boot kernel\n\r");/ * 打印提示信息 */
theKernel(0, 362, 0x30000100); / * 启动内核 */
puts("error \n\r");/ * 如果内核启动成功,就不会执行这一句,否则会打印出 error:表示没有成功启动内核 */
```

4 实验结果与分析

为了测试设计的 Bootloader 是否具备稳定和可靠特点，需要在嵌入式 S3C2440 平台多次测试是否能启动内核，并分析 Bootloader 是否具备精简性。

4.1 Bootloader 精简性分析

在编写 Bootloader 的 start.S 文件中定义一个全局变量来计算 Bootloader 大小，具体代码如下：

```
.globl _boot_size
_boot_size:
.word __bss_start - _start
```

（下转第 18 页）