嵌入式系统的Boot Loader技术

陈文智
浙江大学计算机学院
2009年5月

内容提要

- 1. Boot Loader程序的基本概念
- 2. Boot Loader的典型结构框架
- 3. Boot Loader实验
  - 实验一 Boot Loader应用实验
  - 实验二 U-BOOT的分析和移植

1. Boot Loader程序的基本概念

- Boot Loader就是在操作系统内核运行之前运行的一段小程序
  - 初始化硬件设备和建立内存空间的映射图
  - 将系统的软硬件环境带到一个合适的状态，以便为最终调用操作系统内核准备好正确的环境
- 系统的Boot Loader程序通常安排在地址0x00000000 处

- Boot Loader所支持的硬件环境
  - 每种不同的CPU体系结构都有不同的Boot Loader
- Boot Loader的安装地址
- Boot Loader相关的设备和机制
  - 主机和目标机之间一般通过串口建立连接
- Boot Loader的启动过程

- Boot Loader的操作模式
  - 启动加载模式
  - 下载模式
- Boot Loader与主机之间的通信设备及协议

2. Boot Loader的典型结构框架

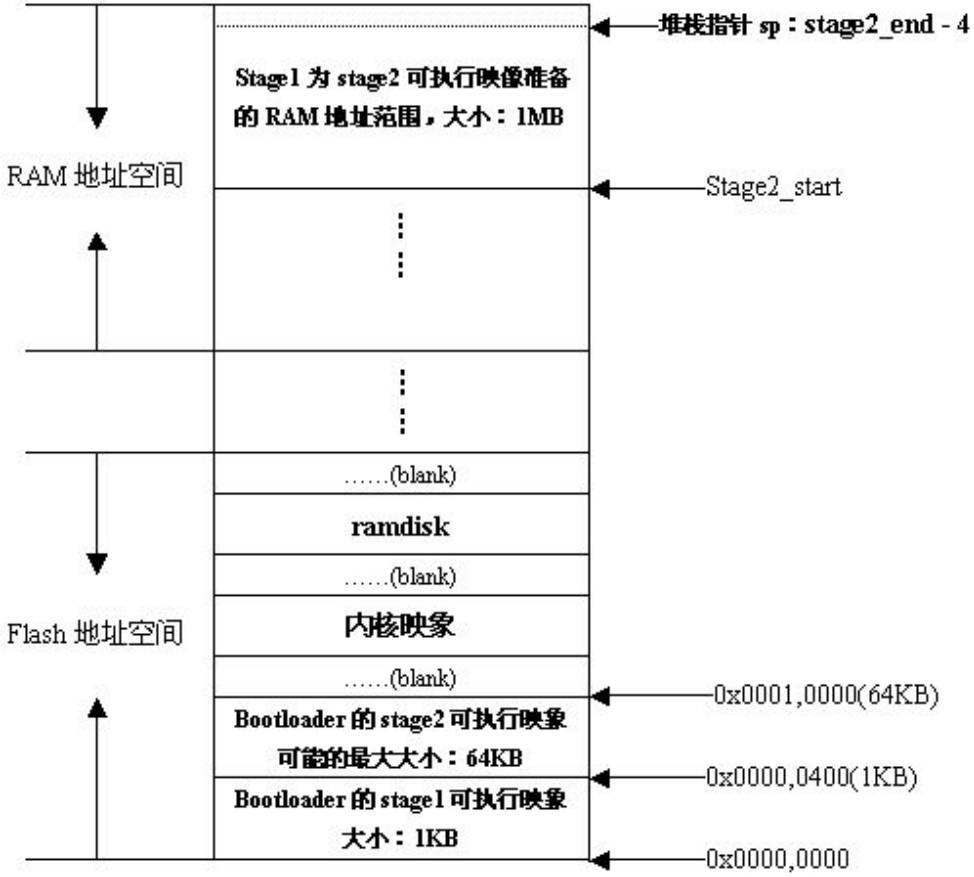
- 操作系统的角度看，Boot Loader的总目标就是正确地调用内核来执行
- 大多数Boot Loader都分为阶段1和阶段2两大部分
  - 阶段1实现依赖于CPU体系结构的代码
  - 阶段2实现一些复杂的功能

2.1 Boot Loader阶段1介绍

- Boot Loader 的阶段1通常包括以下步骤:
- 1)硬件设备初始化。
  - 屏蔽所有的中断
  - 设置CPU的速度和时钟频率
  - RAM初始化
  - 初始化LED
  - 关闭CPU内部指令／数据Cache

- 2）为加载阶段2准备RAM空间
  - 除了阶段2可执行映象的大小外，还必须把堆栈空间也考虑进来
  - 必须确保所安排的地址范围的的确确是可读写的RAM空间

- 3）拷贝阶段2到RAM中
- 4）设置堆栈指针sp
- 5）跳转到阶段2的C入口点
- Boot Loader 的 阶段2 可执行映象刚被拷贝到 RAM 空间时的系统内存布局，如下图：


堆栈指针 sp：stage2_end - 4
Stage1 为 stage2 可执行映像准备的 RAM 地址范围，大小：1MB
RAM 地址空间
Stage2_start
……(blank)
ramdisk
……(blank)
Flash 地址空间
内核映象
……(blank)
0x0001,0000(64KB)
Bootloader 的 stage2 可执行映象可能的最大大小：64KB
0x0000,0400(1KB)
Bootloader 的 stage1 可执行映象大小：1KB
0x0000,0000

2.2 Boot Loader阶段2介绍

- 1）初始化本阶段要使用到的硬件设备
  - 初始化至少一个串口，以便和终端用户进行I/O输出信息
  - 初始化计时器等

```
typedef struct memory_area_struct {
    u32 start; /* 内存空间的基址 */
    u32 size;  /* 内存空间的大小 */
    int used;
} memory_area_t;
```

- 2）检测系统的内存映射
  - 内存映射的描述
  - 可以用如下数据结构来描述RAM地址空间中的一段连续的地址范围：

```
typedef struct memory_area_struct {
    u32 start; /* 内存空间的基址 */
    u32 size;  /* 内存空间的大小 */
    int used;
} memory_area_t;
```

  - 内存映射的检测

- 3）加载内核映像和根文件系统映像
  - 规划内存占用的布局
    - 内核映像所占用的内存范围
      - MEM_START + 0X8000
    - 根文件系统所占用的内存范围
      - MEM_START + 0X00100000
  - 从Flash上拷贝
    - While循环

- 4）设置内核的启动参数
  - 标记列表(tagged list)的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
  - 嵌入式Linux系统中，通常需要由Boot Loader设置的常见启动参数有：ATAG_CORE、ATAG_MEM、ATAG_CMDLINE、ATAG_RAMDISK、ATAG_INITRD

```
params->u.core.flags = 0;
params->u.core.pagesize = 0;
params->u.core.rootdev = 0;
params = tag_next(params);
```

- 例:设置ATAG_CORE的代码如下:

```
params = (struct tag *)BOOT_PARAMS;
    params->hdr.tag = ATAG_CORE;
    params->hdr.size = tag_size(tag_core);
    params->u.core.flags = 0;
    params->u.core.pagesize = 0;
    params->u.core.rootdev = 0;
    params = tag_next(params);
```

BOOT_PARAMS 表示内核启动参数在内存中的起始基地址，指针params是一个struct tag类型的指针。宏tag_next()将以指向当前标记的指针为参数，计算出当前标记的下一个标记的起始地址

HOW TO CALL ?

- 5）调用内核
  - CPU寄存器的设置：
    - R0＝0；
    - R1＝机器类型ID；关于机器类型号，可以参见：
    - linux/arch/arm/tools/mach-types。
    - R2＝启动参数标记列表在RAM中起始基地址；
  - CPU 模式：
    - 必须禁止中断（IRQs和FIQs）；
    - CPU必须SVC模式；
  - Cache和MMU的设置：
    - MMU必须关闭；
    - 指令Cache可以打开也可以关闭；
    - 数据Cache必须关闭；

HOW TO CALL ?

2.3 关于串口终端

- 向串口终端打印信息也是一个非常重要而又有效的调试手段
- 如果碰到串口终端显示乱码或根本没有显示的问题，可能是因为：
  - Boot Loader 对串口的初始化设置不正确
  - 运行在host 端的终端仿真程序对串口的设置不正确

- Boot Loader 启动内核后却无法看到内核的启动输出信息：
  - 确认内核在编译时是否配置了对串口终端的支持，并配置了正确的串口驱动程序
  - Boot Loader 对串口的初始化设置是否和内核对串口的初始化设置一致
  - 还要确认 Boot Loader 所用的内核基地址必须和内核映像在编译时所用的运行基地址一致

3. Boot Loader实验

- 实验一 Boot Loader应用实验
- 实验二 U-BOOT的分析和移植

实验一 Boot Loader应用实验(1)

- 烧写XsBase255的BootLoader
  - 编译生成XsBase255专用的JTAG程序Jflash-XSBase255
  - 编译生成XSBase的Boot Loader x-boot255
  - 正确连线
  - 利用JTAG烧写BootLoader

```
[root@XSBase JTAG]# ./Jflash-XSBase255 x-boot255
```

实验一 Boot Loader应用实验(2)

- 熟悉使用 Bootloader 指令，执行各个指令后将其结果与下表的 description进行比较

Usage	Help
Description	对各个指令的简单的说明。
Arguments	None
Example	X-HYPER255> Help

X-HYPER255> reload kernel

Usage	Reload [kernel/ramdisk]
Description	将Flash中纪录的image复制到SDRAM 为了复制 kernel image到 SDRAM， Autoboot时自动执行
Arguments	Kernel －将flash的 kernel image复制到 SDRAM 0xa0008000 Ramdisk －将flash的 ramdisk复制到 SDRAM 0xa0800000
Example	X-HYPER255> reload kernel

X-HYPER255> tftp zImage kernel
X-HYPER255> tftp zImage 0xa0000000

Usage	Tftp [file] [loader/kernel/root/ramdisk] Tftp [file] [addr]
Description	通过Ethernet将 Host的映像文件下载到SDRAM中
Arguments	Loader－将接收到的文件储存到loader的SDRAM 0xa0000000 Kernel－将接收到的文件储存到kernel的 SDRAM 0xa0008000 Root － 将接收到的文件储存到 0xa0000000 Ramdisk － 将接收到的文件储存到 0xA0800000。 Addr － SDRAM上纪录接收到的文件的地址
Example	X-HYPER255> tftp zImage kernel X-HYPER255> tftp zImage 0xa0000000

```
XSBASE255> flash kernel
XSBASE255> flash 0xc0000 0xa0000000 0x100000
```

Usage	Flash [loader/kernel/root/ramdisk] Flash [dest] [src] [len]
Description	将SDRAM上的数据储存到flash的相应地址
Arguments	Loader-将SDRAM的loader 0xa00000000储存到flash的0x0地址 Kernel-将SDRAM的Kernel 0xa00080000储存到flash的0xc0000 地址 Root-将SDRAM的root 0xa0000000储存到flash的0x1c0000地址 Ramdisk-将SDRAM的ramdisk 0xA0800000储存到Flash的0x1c00 00地址 Dest-储存到flash上的地址 Src-原来的数据所在地址 Len-复制的长度
Example	XSBASE255> flash kernel XSBASE255> flash 0xc0000 0xa0000000 0x100000

```
XSBASE255> boot
XSBASE255> boot 0 200
XSBASE255> boot 0xa0008000 0 200
```

Usage	Boot Boot [opt1] [opt2] Boot [addr] [opt1] [opt2]
Description	驱动SDRAM上的 kernel 通过相应 arguments 驱动 或者驱动相应地址的 kernel。
Arguments	Opt1 - kernel option(Only 0) Opt2 - machine type(X-Hyer255 : 200) Addr - kernel image address
Example	XSBASE255> boot XSBASE255> boot 0 200 XSBASE255> boot 0xa0008000 0 200

实验二 U-BOOT的分析和移植(1)

- U-BOOT的特点
  - 在线读写Flash、DOC、IDE、IIC、EEROM、RTC 。其他一般的BOOT-LOADER不支持IDE和DOC的在线读写。
  - 支持串行口kermit和S-record下载代码
  - 识别二进制、ELF32、uImage格式的Image，对Linux引导有特别的支持
  - 单任务软件运行环境

- 脚本语言支持(类似BASH脚本)
- 支持WatchDog、LCD logo和状态指示功能
- 支持MTD和文件系统
- 支持中断
- 详细的开发文档

实验二 U-BOOT的分析和移植(2)

- U-BOOT源代码结构
  - /board：和一些已有开发板相关的文件
  - /common：与体系结构无关的文件，实现各种命令的C文件
  - /cpu：CPU相关文件
  - /disk：disk驱动的分区处理代码
  - /doc：文档
  - /drivers：通用设备驱动程序，如网卡串口USB等

- /fs:支持文件系统的文件
- /net：与网络有关的代码
- /lib_arm：与ARM体系结构相关的代码
- /tools：创建S-Record格式文件 和U-BOOT images的工具

实验二 U-BOOT的分析和移植(3)

- 对U-BOOT的移植
  - 建立自己开发板的目录和相关文件
    - 在include/configs目录中添加头文件xsbase.h
    - 在board/目录下新建xsbase目录，创建如下文件：flash.c、memsetup.S、xsbase.c、Makefile和u-boot.lds
  - 添加网口设备控制程序，cs8900网口设备的控制程序cs8900.c 和cs8900.h

实验二 U-BOOT的分析和移植(4)

- 修改Makefile
  - 在u-boot-1.1.2/Makefile中加入:
  - xsbase_config : unconfig
  - @./mkconfig $(@:_config=) arm pxa xsbase

实验二 U-BOOT的分析和移植(5)

- 生成目标文件
  - 先运行make clean
  - 然后运行make xsbase_config
  - 再运行make all
  - 生成三个文件：
    - u-boot——ELF格式的文件，可以被大多数Debug程序识别。
    - u-boot.bin——二进制bin文件，这个文件一般用于烧录到用户开发板中。
    - u-boot.srec——Motorola S-Record格式，可以通过串行口下载到开发板中

实验二 U-BOOT的分析和移植(6)

- 通过JTAG口将u-boot.bin烧写到Flash的零地址，复位后执行u-boot
- 输入help得到所有命令列表