

U-Boot 在 S3C2410 上的移植分析

李 夏 王化深 易海旺

(北京交通大学 电子信息工程学院 北京 100044)

摘 要:从较为通用的开源 u-boot 引导程序移的结构代码入手,并且以移植到三星的 smdk2410 开发板引导 arm-linux kernel 为例子,分析说明一个嵌入式系统中 Boot loader 典型的结构、作用及基本功能。

关键词:内核引导 嵌入式引导程序 嵌入式 Linux boot loader 移植

Analyzing the Transplantation of U-Boot on S3C2410

LI Xia, WANG Huashen, YI Haiwang

(School of Electronics and Information Engineering, Beijing JiaoTong University, Beijing, 100044, China)

Abstract: In an embedded system the role of the boot loader is most important. This paper measures the more universal open source boot loader u-boot by analyzing the construction of the u-boot, then ports the u-boot into smdk2410 in order to boot the armlinux kernel and explains the typical construction, action and basic function of the boot loader in an embedded system.

Keywords: boot kernel, boot loader, embedded linux, boot loader porting

1 引言

在嵌入式系统中的引导代码 (boot loader) 的开发和移植是嵌入式系统开发的难点之一,同时也是系统运行的基本条件。这段代码是和硬件紧密相关的,就好比是 PC 机上的 BIOS。但是在嵌入式系统中没有像 PC 那样的 BIOS 固件程序 (有的嵌入式 CPU 会在芯片内部嵌入一段短小的程序,用来将 boot loader 装载到 RAM 中),简单地说,boot loader 就是在操作系统内核运行之前运行的一段小程序。通过这段小程序,可以初始化硬件设备、建立内存空间的映射图,从而将系统的软硬件环境设置成一个合适的状态,以便为最终调用操作系统内核准备好正确的环境。Boot Loader 是严重地依赖于硬件而实现的,因此,在嵌入式定位中建立一个通用的 Boot Loader 非常困难。

由图 1 可以看出一个典型的嵌入式 Linux 系统一般要有 4 个部分组成^[1]:

(1) 引导加载程序 包括固化在固件 (firmware) 中的 boot 代码 (可选),和 Boot Loader 两大部分。

(2) Linux 内核 特定于嵌入式系统的定制内核以及内核的启动参数。

(3) 文件系统 包括根文件系统和建立于 Flash 内存设备



图 1 boot loader 在系统中的层次结构

之上的文件系统。通常用 RAM disk 作为根文件系统。

(4) 用户应用程序 有时在用户应用程序和内核层之间可能还会包括一个嵌入式图形用户界面。

在这里作者使用的硬件平台是广州友善之臂电子有限公司的 SBC2410 开发板,它是仿照韩国三星公司的 sanumg smdk2410 而制作的。

在嵌入式世界,因为硬件环境各不相同,找到一个通用的 boot loader 是非常困难的,u-boot 是目前比较流行且开源的 boot loader 项目,它被视为较为通用的 Boot loader, u-boot 的功能十分强大,u-boot 是在 PPCboot 的基础上进化来的一个开放源码的嵌入式 BootROM 程序,由德国的工程师 Wolfgang Denk 从 8XXROM 代码发展而来的,它支持很多处理器,比如 PowerPC、ARM、MIPS 和 x86。目前 u-boot 的最新版源代码可以在 Sourceforge 的 CVS 服务器中匿名获

本文于 2005-05-19 收到。

得。这里使用的是最新版本 u-boot-1.1.2, 它已经对 smdk2410 支持了, 所以需要改动的地方并不是很多。

2 boot loader 的分析和 u-boot 特点

2.1 boot loader 的典型框架和功能

Boot loader 是用来初始化硬件和引导内核的, 并且为操作系统准备好运行的环境。在运行操作系统的时候, 有两种方式来运行内核的镜像^[2], 如果内核镜像文件装在 flash 中, 它可以在 flash 中正确链接, 则 boot loader 可以在引导内核直接在 flash 中运行。当然内核镜像也可以放在系统的 RAM 区并且从 RAM 区运行。需要注意的是内核用 RAM 的前 16KB 存放它运行所需要的数据表, 这里推荐 RAM 区给它们留 32KB 的空间, 见表 1。

表 1 boot loader 应满足的条件

| 选项 | 条件 |
|-----------|--|
| CPU 寄存器设置 | r0 = 0, r1 = 机器型号 r2 = RAM 物理地址 |
| CPU 模式 | 关闭所有中断 (IRQs and FIQs) CPU 必须是 SVC 模式 |
| Caches | 指令 cache 可以开或者关。 数据 cache 必须关闭 |
| MMUs | MMU 要关闭 |

2.2 u-boot 的特点

u-boot 支持 SCC/FEC 以太网、OFTP/TFTP 引导、IP 和 MAC 的预置功能, 这一点和其他 boot loader (如 BLOB 和 RedBoot 等) 类似。但 u-boot 还具有一些特有的功能^[3]。

(1) 在线读写 Flash、DOC、IDE、IIC、EEROM、RTC, 其他的 boot loader 根本不支持 IDE 和 DOC 的在线读写。

(2) 支持串行口 kermit 和 S-record 下载代码, u-boot 本身的工具可以把 ELF32 格式的可执行文件转换成为 S-record 格式, 直接从串口下载并执行。

(3) 识别二进制、ELF32、uImage 格式的 Image, 对 Linux 引导有特别的支持。U-BOOT 对 Linux 内核进一步封装为 uImage。封装如下: # {CROSS_COMPILE} - objcopy - O binary -R .note -R .comment -S vmlinux \ linux. bin # gzip -9 linux. bin # tools/mkimage -A arm -O linux -T kernel -C gzip -a 0xc0008000 -e \0xc0008000 -n "Linux -2.4.20" -d linux. bin. gz /tftpboot/uImage 即在 Linux 内核镜像 vmlinux 前添加了一个特殊的头, 这个头在 include/image. h 中定义, 包括目标操作系统的种类 (比如 Linux, Vx-Works 等)、目标 CPU 的体系机构 (比如 ARM、PowerPC 等)、映像文件压缩类型 (比如 gzip、bzip2 等)、加载地址、入口地址、映像名称和映像的生成时间。当系统引导时, U-BOOT 会对这个文件头进行 CRC 校验, 如果正确, 才会跳到

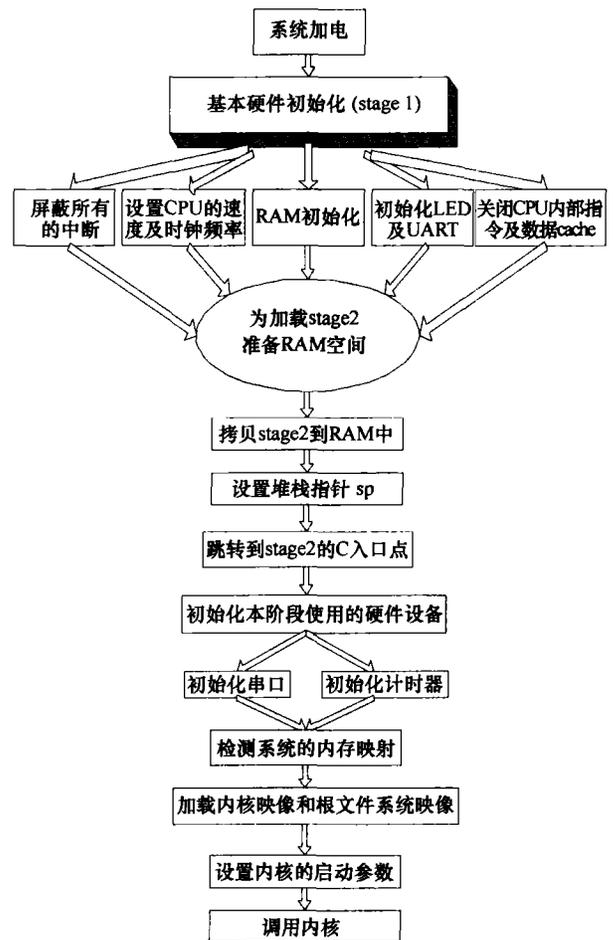


图 2 boot loader 工作流程图

内核执行。

3 u-boot 代码结构和分析

u-boot 代码结构, 见图 3。

我们结合启动流程和代码结构来分析在 smdk2410 上使用 u-boot 所涉及到可能要修改的文件。

3.1 u-boot 代码结构和分析

u-boot 从 cpu/arm920t/start. s 开始运行, 它的任务是设置处理器状态、初始化中断和内存时序并确定是否对整个 u-boot 重定位, 是它从 flash 跳到定位好的 ram 中去执行^[4]。start. s 中的重要代码:

```

_TEXT_BASE
.word TEXT_BASE
.globl armboot_start;
.word _start
.globl _bass_start
_bss_start;
.word _bss_start

```

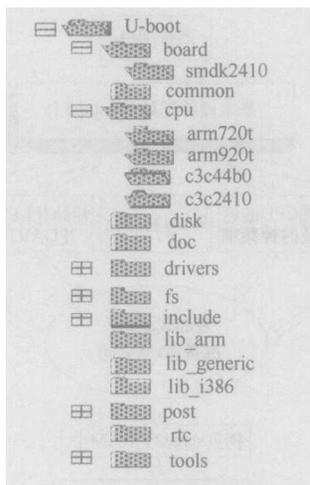


图 3 u-boot 代码树结构图 [5]

```
. globl _bss_end
_bss_end;
.word_end
```

这段中的 `TEXT_BASE = 0x33F80000`, 这个定义可以在 `/board/smdk2410/config.mk` 中找到, 因为 `smdk2410` 中的 `ram` 地址是 `0x30000000-0x34000000` 而且规定了 `linux kernel` 是从 `0x30008000` 开始进入的, `0x33F80000` 是复制 `boot-loader` 的地方, 因为考虑到 `bootloader` 的大小一般不会超过 `1M`, 所以从 `0x33F80000` 到 `0x34000000` 空间完全够用的。 `_armboot_start` 的函数作用是什么呢? `boot loader` 的启动一般分为两个阶段, 第一阶段主要是依赖 `CPU` 的体系结构硬件初始化的代码, 一般用汇编语言编写。第二阶段通常用 `C` 语言完成, 完成更为复杂的功能, 这个阶段主要完成检测系统的内存映射, 把内核的 `image` 和 `rootfile` 从 `flash` 读到 `ram` 区。这里用的 `u-boot` 代码并没有明显的阶段区分, 但是可以根据代码的功能大致地区分开来, 它是用下面语句实现两个阶段地工作交接, 而且 `_start` 的地址和 `TEXT_BASE` 是一样的:

```
ldr pc, _start_armboot
_start_armboot; .word start_armboot
```

编译调试好的 `u-boot.bin` 要固化到 `flash` 里面, 则 `u-boot` 就应该具有将自己从 `flash` 复制到 `ram` 中运行的能力。

复制代码如下:

```
copy_loop:
ldmia r0!, {r3-r10}
stmia r1!, {r3-r10}
cmp r0, r2
ble copy_loop
```

重新定位的代码是为了把 `u-boot` 重 `flash` 转到 `ram` 中运行。代码如下:

```
relocate
```

```
adr r0, _start
ldr r1, _TEXT_BASE
cmp r0, r1
beq stack_setup
ldr r2, _armboot_start
ldr r3, _bss_start
sub r2, r3, r2
add r2, r0, r2
```

`r0` 是当前代码的位置, 如果从 `flash` 开始运行 `r0 = 0x0000000`, 如果从 `ram` 开始则 `r0 = TEXT_BASE = 0x33f80000`, 这时就要执行 `copy_loop` 代码了^[5]。

运行 `stack_setup`, 建立堆栈, `start.s` 中剩下的代码是对处理器的设置和中断, 他们的参数需要参考处理器的数据手册, 因为这个版本的 `boot loader` 已经支持 `smdk2410` 了, 所以其余的这些和移植关系并不是很大。

其中 `board` 目录存放了 `u-boot` 支持的目标板子的子目录, 在这里关心的是 `board/smdk2410/` 下的内容。这里主要有以下文件:

(1) `u-boot.lds` 这个文件是内核链接器的脚本文件。由于内核可执行文件由许多链接在一起的对象文件组成, 而文件由许多如文本、数据、`bss` 等组成, 所有的这些对象文件是由这个链接器脚本文件来负责链接装入的, 就是输入对象文件的各节映射到输出文件中。 `u-boot` 的各个节通过这个文件被链接, 装入到内存中的特定的偏移量处。

(2) `memsetup.s` 这个文件是配置开发板的参数的, 从这个文件的名字就可以了解到, 它的作用是和存储器和寄存器相关的, 如装载程序运行的起始地址和设置总线宽度状态寄存器等。

(3) `smdk2410.c` 这个文件有两个函数:

`board_init()` 和 `dram_init()`。

`board_init()` 主要是对一个目标板参数结构 `gd_t` 进行初始化, 这个结构在 `linclude/asm-arm/global_data.h` 中进行了定义, 代码如下。

```
Typedef struct global_data{
    bd_t * bd;
    /* bd_t{} 是关于目标板的参数, 它在 include/asm-arm/u-boot.h 中定义 */
    unsigned long flags;
    unsigned long baudrate; /* 设置波特率 */
    unsigned long have_console;
    /* 调用 serial_init() */
    unsigned long reloc_off; /* 重定向的偏址 */
    unsigned long env_addr;
    /* 环境结构的地址 */
};
```

```

unsigned long env_valid;
/* 环境是否合法的校验 */
unsigned long fb_base;
/* frame buffer 的基地址 */
#ifdef CONFIG_VFD
unsigned char vfd_type; /* 显示类型 */
#endif
}gd_t;
bd_t{}数据结构如下:
typedef struct bd_info{
int bi_baudrate; /* 串口控制台波特率 */
unsigned long bi_ip_addr; /* IP 地址 */
unsigned char bi_enetaddr[6]; /* mac 地址 */
struct environment_s * bi_env;
ulong bi_arch_number; /* 板子标识符 */
ulong bi_boot_params; /* 启动参数 */
struct /* RAM 配置 */
{
ulong start; /* RAM 开始地址 */
ulong size; /* RAM 的大小 */
}bi_dram[CONFIG_NR_DRAM_BANKS]
}bd_t;

```

Board_init 定义了一个 gd_t 类型的变量 gd,地址由 r8 寄存器指向,并对其中的 bi_arch_number 域和 bi_boot_params 域进行了设置。

dram_init()定义了 SDRAM 的其实地址和大小:

```

gd->bd->bi_dram[0].start = PHYS_SDRAM_1;
gd->bd->bi_dram[0].size = PHYS_SDRAM_1_
SIZE;

```

(4) flash.c 该文件主要完成了对 flash_info_t{}数据结构的初始化:

```

typedef struct{
ulong size; //容量的大小
ushort sector_count; //擦除单元
ulong flash_id; //芯片信息,由厂商提供
ulong start[CFG_MAX_FLASH_SECT]; //物理扇区的起始地址
uchar protect[CFG_MAX_FLASH_SECT]; //扇区保护状态
}flash_info_t;

```

SDRAM 和 Flash 的信息分别在 smdk2410.c 和 flash.c 文件中定义,最后通过 lib_arm/board.c 文件中的 display_

dram_config()和 display_flash_config()函数来显示。

cpu 目录下存放了 u-boot 所支持的 cpu 类型,用到的是 cpu/arm920t/下的文件,这些文件的作用是初始化一个可执行的环境。开始的时候已经把目录下非常关键的 start.s 的作用和功能做了介绍。cpu 目录中除了 start.s 还有:

(5)cpu.c 它是和 cpu 相关的文件,是用来对处理器进行操作,其中比较重要的函数是 cpu_init(),它是用来建立中断栈的,如果使用了 IRQ 和 FIQ 机制,就要用到这个函数。

(6)interrupts.c 改文件是处理中断的,例如打开、关闭中断,前面的时候在 start.s 中已经看到了它准备了异常处理向量表,而这些异常中断所采取的动作就定义在 interrupts.c 文件中。

(7)serial.c 是对 UART 相关的寄存器进行配置,serial_init()中调用了 serial_setbrg()函数,设置波特率。

(8)speed.c 和 usb_ohci.c 这两个文件和移植的关系并不是很大,其中 speed.c 提供了 CLK 频率,而 usb_ohci.c 则主要是处理 USB 的。

lib_arm 存放的是 ARM 平台的公共接口代码。前面分析 start.s 时,曾经看到了 start_armboot()函数,该函数定义在/lib_arm/board.c 文件中。该函数的第一条语句是

“DECLARE_GLOBAL_DATA_PTR”它在/include/adm_arm/global_data.h 文件中有宏定义。

```

#define DECLARE_GLOBAL_DATA_PTR register
volatile gd_t * asm (“r8”)

```

这里还有一个重要的函数 mem_malloc_init,它是为动态分配内存做准备,只有 mem_malloc_init 的调用参数正确了,u-boot 才能正确地引导开发板运行。其函数原型是 static void mem_malloc_init(ulong dest_addr)

```

{mem_malloc_start = dest_addr;
mem_malloc_end = dest_addr + CFG_MALLOC_LEN;
mem_malloc_brk = mem_malloc_start;
memset((void *)mem_malloc_start,0, mem_malloc_end - mem_malloc_start);
}

```

common 目录存放了 u-boot 的一些公用命令函数原型。例如系统初始化 flash 后,系统就执行 env_relocate()函数,重新定位环境变量参数区,还有很多 u-boot 命令的原型都是在这里定义。

drivers 这个目录存放了各种外设接口的驱动程序。fs 目录中存放的是 u-boot 支持的文件系统。include 是头文件公用目录,其中 include/configs/smdk2410.h 定义了 s3c2410 相关的资源配置参数,可能根据目标板的参数修改这些文件。

tools 目录里面提供了可能用到的实用工具,例如做内核镜像 uImage 的工具 imagewrite。

参考文献

- 1 Anand K Santhanam, Vishal Kulkarni. 嵌入式设备上的 Linux 系统开发. <http://www-128.ibm.com/developerworks/cn/linux/embed/embclev>. 2002.
- 2 Curt Brune. The Universal Boot Loader. <http://linuxdevice.com> 2002.
- 3 Karim, Yaghmour Building Embedded Linux Systems. O' Reilly Media Inc. 2003.
- 4 Russell King. Booting ARM linux. <http://www.arm.linux.org.uk/developer/booting.php>. 2002.
- 5 孙天泽. 嵌入式设计及 linux 驱动程序开发指南. 北京:电子工业出版社, 2005.
- 6 田志刚. 嵌入式 Linux 系统及其应用研究. 微计算机应用, 2003, 24(15): 290~295