Uboot 在 S3C2440 上的移植

卢 伟，潘 炼

（武汉科技大学 信息科学与工程学院自动化系，湖北 武汉 434200）

摘 要：通过分析 Uboot 的文件结构及其启动流程，详细给出了 Uboot 在基于 ARM920T 开发板上的移植方案，包括编译、调试全过程，最终能够在 Uboot 命令方式下加载映像文件，完成 Linux 内核与 yaffs 映像文件的调试，具有 Bootloader 移植的通用性。

关键词：Uboot；S3C2440；ARM920T；引导过程；启动代码

中图分类号：TP368 **文献标识码**：B **文章编号**：1674-7720(2010)24-0004-05

Porting of Uboot for S3C2440

LU Wei, PAN Lian

(Dept.of Information Science and Engineering, Wuhan University of Science and Technology, Wuhan 434200, China)

Abstract: By analyzing the structure and starting process of Uboot, described in details the Uboot transplant programs to development platform based on ARM920T, including compilation as well as debug during the whole process, finally it could be able to load image file in the Uboot command mode, and also complete the debug of Linux kernel and yaffs image file, which performs the universal character of Bootloader transplantation.

Key words: Uboot; S3C2440; ARM920T; boot process; boot code

1 Uboot 移植环境准备

1.1 移植平台的硬件组成

硬件平台是 ARM9 的体系结构，ARM920T 的 CPU，SOC 芯片是三星的 S3C2440，支持 Nand Flash 与 Nor Flash 的可选启动方式，其主要硬件资源如表 1 所示[1]。

支持 Nand Flash 与 Nor Flash 启动，可以通过跳线来选择启动方式。Nand Flash 启动时，最开始 4 KB 数据被硬拷贝到内部 Boot Internal SRAM，且被映射到 nGCS0 的片选空间 0x0000,0000—0x0800,0000；Nor Flash 方式启动时，它直接被映射到 nGCS0 的片选空间。所以，在 Uboot 移植时，要考虑将 Uboot 烧写到 Nor flash 上还是 Nand Flash 上。

1.2 Uboot 工作原理

Uboot 的整体结构如图 1 所示。

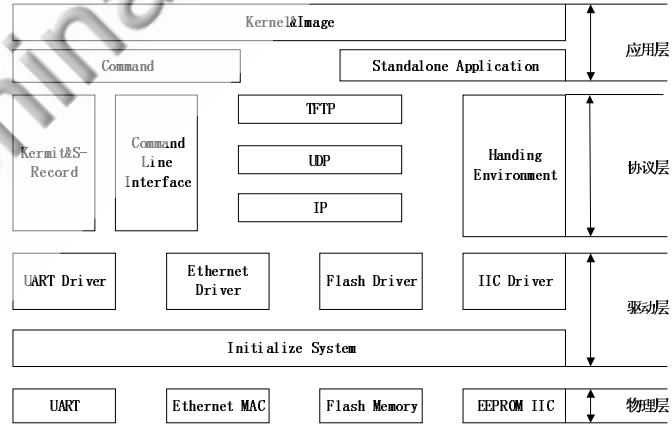

图 1 U-boot 的整体结构

从图 1 可以看出，这种分层结构的 Uboot 分模块化了，给移植带来了很大的方便。由于协议层与应用层是与目标硬件无关的，因此移植工作主要集中在物理层和驱动层上面的修改。而 Uboot 支持串口下载、网络下载，并提供了很多交互式命令。整个 Uboot 编译、连接过程如下：

(1)创建编译环境

在 MAKEFILE 中会调用根目录下

表 1 实验板的主要硬件组成

CPU	S3C2440A	主频 400 MHz，最高 533 MHz
SDRAM	hy57v561620ftp-h	2 片 16 bit 32 M 串联，最高 100 MHz
Nand Flash	K9F1208U00-PCB0	64 M×8 bit
Nor Flash	AM29LV160DB-90EC	2 M×16 bit
LCD	HST-TPA3.5F	3.5 英寸真彩色 TFT 电阻式触摸屏 LCD
NET CARD	CS8900A-CQ3Z	10M 网卡控制芯片 16 bit 数据传输

的 mkconfig 文件，如下：

```
MKCONFIG:= $(SRCTREE)/mkconfig
qq2440v3_config:unconfig
@$(MKCONFIG)$(@:_config=)arm ARM920T qq2440v3 NULL s3c24x0
```

Mkconfig 文件引用传入的参数 $1=qq2440v3、$2=arm、$3=arm920t、$4=qq2440v3、$5=NULL、$6=s3c24x0，流程如图 2 所示。

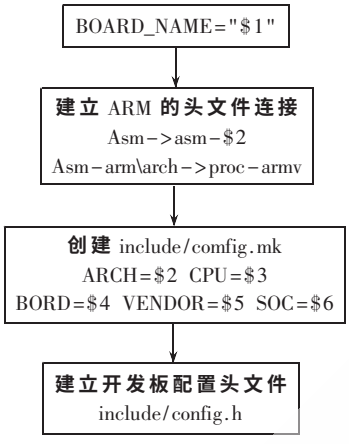

图 2 创建文件链接

(2)编译流程

编译流程如图 3 所示。

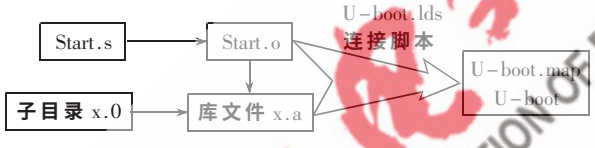

图 3 Uboot 的编译流程

最终生成内存映像图文件 U-boot.map 和可执行二进制映像 elf 文件 U-boot[2]，可以直接将生成的 U-boot 下载到 SDRAM 来单步调试。

2 Uboot 的移植操作

2.1 存储器映射与存储器重映射

存储器映射，实现了统一编址，方便了程序在 32 bit 寻址(4 GB 寻址空间)的范围内能够寻址到任意的物理存储区。

S3C2440 芯片不带片内 Flash，带片内4 KB 的 SRAM，被映射到了 0x4000_0000~0x4000_1000 的地址空间，外部的 SDRAM 被映射到 bank6，网卡被映射到 bank3，Flash 被映射到 bank0。

由于 Uboot 是上电后就运行，因此需要将代码定位在 Flash 从 0x0000_0000 的上电入口处。为了提高系统加载速度并且实现在线编程功能，需要将整个 Uboot 从 Flash 中搬到 RAM 运行，即代码从定位，将整个代码定位到 SDRAM 的 0x3300_0000 之后，来作为其实际的运行地址，具体如图 4 所示。

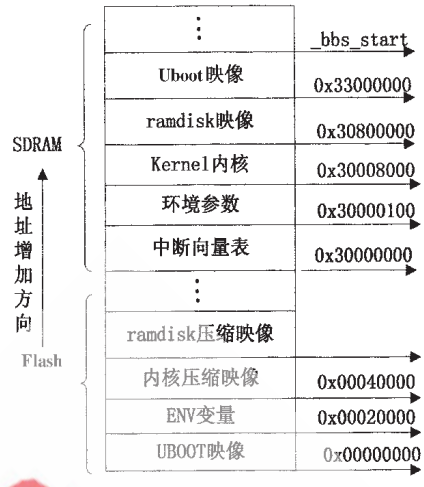

图 4 uboot 代码从定位后的 SDRAM 分布

2.2 配置主机运行环境

Uboot 与 Linux 系统密切相关，笔者在 RH Linux 的虚拟机中搭建了整个运行环境，采用的是 2.2.4 的 Linux 内核，arm-linux-gcc-3.4.1 的交叉编译工具[3]，需要在/root/.bashrc 文件中做一下交叉编译工具路径的声明，即加上如下一句：

```
export PATH=$PATH:/usr/local/arm/3.4.1/bin
```

保存并退出，在终端下输入“arm-linux-gcc-version”并回车，如果能看到输出版本信息为 3.4.1，则代表路径设置正确，交叉编译工具链已经成功安装。

2.3 修改 CPU 相关代码

在调试 Uboot 时，如果每次都将二进制映像烧录到 Flash 中，不仅需要等待，而且操作麻烦，本文是在调试阶段将二进制映像直接烧录到外部存储器 SDRAM 中，然后直接从该处运行，这样直接在内存中运行，可以很方便地完成 Uboot 调试。

Uboot 启动的第一阶段，从.\cpu\arm920t\start.s 开始执行，依次完成关闭看门狗、关闭中断、设置 CPU 分频比、初始化 SDRAM、代码重定位、设置堆栈，最后跳转到 C 函数的入口点。当在 SDRAM 中调试时，内存的初始化已经预先完成了，因此不需要初始化 SDRAM 和代码重定位的功能。

在.\include\configs\qq2430.h 添加宏定义 define CONFIG_SKIP_LOWLEVEL_INIT，就会跳过 cpu_init_crit 处的初始化 SDRAM 函数，代码如下所示：

```
#ifndef CONFIG_SKIP_LOWLEVEL_INIT
cpu_init_crit:
…
```

当 Uboot 在 SDRAM 中运行时，代码的入口地址 _start 与代码在 SDRAM 中重定位的地址 _TEXT_BASE 相同，直接跳转到堆栈初始化处 stack_setup，跳过了代码的 Flash 到 RAM 的搬运。代码如下所示：

```
adr r0, _start
```

```
ldr  r1, _TEXT_BASE
cmp  r0, r1
beq  stack_setup
```

2.4 添加平台相关代码

从汇编跳转到C程序的入口点start_armboot时，即位于./Lib_arm/Board.c中的start_armboot函数，通过函数初始化数组来依次完成CPU(cpu_init)、板级(board_init)、中断(interrupt_init)、环境变量(env_init)、串口(serial_init)、波特率(init_buardrate)、显示(disp_banner)、Flash初始化(nand_init)，每个初始化后返回不为0的值，否则永远在死循环中挂起，需要重新启动开发板。

2.4.1 锁相环时钟的配置

在smdk2410开发板的基础上修改。在./board/qq2440v3下新建qq2440v3.c，并且将Uboot中的./board/smdk2410/smdk2410.c内容全部复制给qq2440v3.c。在include/configs下新建qq2440v3.h配置文件，同时将include/configs/smdk2410.h全部复制给qq2440v3.h。因为S3C2440与S3C2410的最高运行速率不一样，系统时钟设置也不一样，即锁相环配置有差异，因此内部总线的分频比系数是不同的。S3C2440可运行于400 MHz，而S3C2410则是200 MHz，需要更改系统时钟部分，使其增加对S3C2440的支持。

锁相环输出的MPLL(fclk时钟频率寄存器)与UPLL(USB控制时钟频率寄存器)计算式如下：

```
MPLL=(2*m*Fin)/(p*2^s)        /* 锁相环输出fclk频率
UPLL=(m*Fin)/(p*2^s)          /* 锁相环输出USB控制频率
m=M (the value for divider M)+8,p=P (the value for divider P)+2,s=S
```

其中m、p、s为锁相环的预置值，控制输出频率，Fin为晶振频率12 MHz。通过下面的宏定义分别给M、P、S赋值0x5c、0x01、0x01就能轻松完成时钟配置。

```
#define S3C2440_MPLL_400MHz  ((0x5c<<12)|(0x01<<4)|(0x01))
#define S3C2440_UPLL_48MHz   ((0x38<<12)|(0x02<<4)|(0x02))
#define S3C2440_CLKDIV  0x05  /* FCLK:HCLK:PCLK = 1:4:8, UCLK = UPLL
```

2.4.2 串口的配置

S3C2440带有3个UART，用异步通用串口uart0与上位机通信。串口时钟用pclk时钟，即1/2 hclk时钟，1/8 fclk时钟(mpll时钟)，get_PCLK()函数返回pclk时钟频率。串口0工作于中断模式，通过FIFO缓存收发，没有校验位，1位停止位，通过设置ULCON0寄存器与UFCON0来配置。为了提高数据传送的可靠性，使用了错误接收中断机制，能够检测溢出错误、奇偶校验错误和帧错误，相关错误状态保存在错误状态寄存器UTRSTAT0的对应位中。

串口的波特率用115 200 b/s，通过给波特率除数寄存器UBRDIV0赋值，能够确定串行发送接收波特率。计算公式如下：

UBRDIVn=(int) (UART clock/(buad rate*16))-1；

其中UART clock为pclk时钟，buard rate为115 200。

2.5 编译并调试

为了在内存中运行，先要把链接脚本文件./Board/qq2440v3/U-boot.lds中的程序初始化运行地址.=0x0000,0000改为.=0x3300,0000，以方便直接在RAM中运行。在终端控制台中先进入Uboot的根目录，然后执行命令make qq2440v3_config，通过make all来编译和连接程序。编译没有错误的情况下，会在Uboot的根目录下生成u-boot、u-boot.srec和u-boot.bin三个文件，分别对应于ELF格式、S-Record格式和二进制格式。

直接使用JTAG烧写二进制u-boot.bin到SDRAM的0x3300,0000处，烧写完成之后，将pc的指针指向该处运行，会在串口终端上显示板子的自检信息，并给出提示符等待用户输入命令，如图5所示。

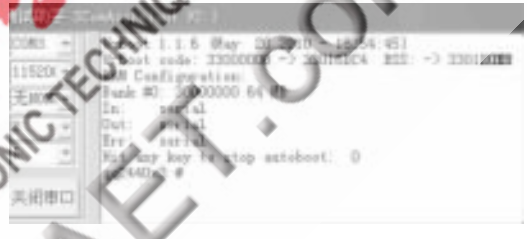

图5 RAM运行串口输出

3 Uboot从Flash中启动

S3C2440既支持从Nor Flash中启动，也支持从Nand Flash中启动。Nor Flash的地址线与数据线分开，方便代码存取且速度很快，上电复位时直接把Nor Flash映射到了0x0000,0000地址处。但是Nand Flash数据线与地址线复用，运行速度慢，为了提高Nand Flash启动效率，S3C2440芯片加入了一个特殊机制，会在上电复位时，把Nand Flash前4 KB代码硬拷贝到内部SRAM的4 KB空间，然后将SRAM的4 KB空间映射到0x0000,0000处，这样直接在SRAM中启动Uboot，节省了启动时间。

3.1 从Nor Flash中运行

3.1.1 添加Nor Flash驱动

board/qq2440v3/Flash.c中的驱动只支持Nor Flash的AMDLV400和AMDLV800两种芯片，不支持本文板子上的AM29LV160，更不支持Nand Flash，只能用CFI标准接口连接，在/drivers/cfiflash.c中定义了该接口标准下的读写函数的具体实现。要调用该驱动，应在配置头文件/include/configs/qq2440v3.h中添加CFI的宏定义：

```
#define CFG_FLASH_CFI_DRIVER 1
```

并在board/qq2440v3/Makefile中去掉原来的Nor flash驱动的编译，即：

COBJS:= qq2440v3.o flash.o 变量中去掉flash.o的连接。

3.1.2 SDRam 初始化并实现代码从定位

Uboot 在 Nor Flash 中启动后，在 start.s 阶段除了要完成必要的寄存器设置外，还要完成 SDRAM 的初始化以及代码从定位，即把 Flash 空间的 Uboot 映像搬运到 SDRAM 高地址空间中，然后在 SDRAM 中运行 Uboot。可以直接从 Nor Flash 启动 Uboot，但从 Nand Flash 启动要实现重定位，在这里就一起实现了。

首先在.\include\configs\qq2430.h 中去掉刚才添加宏定义 define CONFIG_SKIP_LOWLEVEL_INIT，则会在 start.s 阶段进入 cpu_init_crit 函数以完成 I/D caches 设置以及禁止 MMU，随后进入 lowlevel_init 完成内存寄存器组的设置，如 SDRAM 位宽、刷新率等的初始化工作。

在.\include\configs\qq2430.h 中去掉 CONFIG_SKIP_RELOCATE_UBOOT 的宏定义,来完成整个代码的重定位[5]。

Uboot 代码区的长度为 _bss_start-_armboot_start，其中 _bss_start 与 _armboot_start 变量保存的都是代码段的起始地址与终止地址。_start+_bss_start-_armboot_start 为代码区结束的绝对地址，通过地址绝对寻址来复制代码区的数据到内存中 TEXT_BASE 地址区域，其中 TEXT_BASE 在.\Board\QQ2440v3\Config.mk 中被赋值，即 TEXT_BASE=0x33000000，表示代码重定位在 SDRAM 中的运行起始地址。

3.1.3 编译并调试

Uboot 已经能够成功在 SDRAM 中启动运行了，为了能够从 Nor Flash 中启动，需要做如下工作。

先要把链接脚本文件./Board/qq2440v3/U-boot.lds 中的程序初始化运行地址.=0x3300,0000 改为.=0x0000,0000，通过硬件开关选择开机启动方式为 Nor Flash，完成 Nor Flash 映射到 0 地址处。然后在终端控制台编译连接，直到没有错误。通过 JTAG 烧录进 Nor Flash 里面，开机运行后串口终端输出界面如图 6 所示。

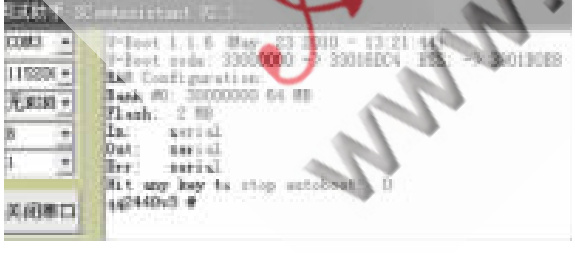

图 6 Nor Flash 中启动串口输出

3.2 从 Nand Flash 中运行

3.2.1 添加 Nand Flash 驱动

S3C2440 支持从 Nand Flash 启动，考虑到移植的通用性，对于没有 Nor Flash 的板子，就需要从 Nand Flash 启动。在.\drivers 目录下有两种 Nand 的驱动，.\Nand 和.\Nand_legacy 两种驱动可以选择，其中.\Nand 能够自动识别很多型号的 Nand Flash，并且是更新版本，因此选择这种驱动。根据 Nand.c 中的

```
#if(CONFIG_COMMANDS&CFG_CMD_NAND) && !defined
(CFG_NAND_LEGACY)
#include <nand.h>
```

条件编译选择 Nand 驱动，首先在板级配置头文件 qq2440v3.h 中的宏定义 CONFIG_COMMAND 中添加 CFG_CMD_NAND，并且不定义 CFG_NAND_LEGACY。在 start_armboot()函数中会对外设逐一初始化，Nand 初始化代码如下：

```
#ifdefined(CFG_MAX_NAND_DEVICE) nand_init;
#endif
```

需要在板级配置头文件 qq2440v3.h 中宏定义 CFG_MAX_NAND_DEVICE，因为 smdk2410 开发板不支持 Nand Flash，因此需要自己来编写 Nand Flash 驱动函数 board_nand_init 来被 nand_init 以及 nand_init_chip 调用，以完成 Nand Flash 的硬件初始化，包括使能 Nand Flash 控制器、初始化 ECC、使能片选信号、设置时序等。

3.2.2 添加 cmd 命令

为了丰富 Nand 与网卡功能，还需要在配置文件中添加 Nand 与网卡相关命令来调用相关函数。在板级配置头文件 qq2440v3.h 中的 CONFIG_COMMANDS 宏定义中以逻辑“或”的形式添加 CFG_CMD_NAND 与 CFG_CMD_NET，这样便可以通过命令方式实现 Nand Flash 的读写以及网络下载功能。

Uboot 的网络功能很强大，可以方便地通过 TFTP 引导或者是 NFS 引导内核映像或者文件系统到 SDRAM，然后直接 go 到此处执行，在 SDRAM 中调试完成后，再将映像文件烧录到 Flash 中，不仅调试方便，而且还节省下载时间。

3.2.3 编译并调试

编译过程跟 Nor Flash 启动一样，最后串口输出信息如图 7 所示。

图 7 Nand Flash 中启动串口输出

此时，整个 Uboot 的移植就算完成了，由于支持串口跟网卡驱动，可以很方便地用这个 Uboot 来通过网卡下载内核映像与文件系统到 Flash，通过串口输出信息调试 Uboot。

4 Uboot 引导 Linux 内核

4.1 内核启动参数的传递

Uboot 在引导内核启动时,通过标记列表的方式向内核传递启动参数。这些启动参数预先以环境变量的方式保存在 Flash 中,在./Lib_arm/Board.c 中的初始化环境变量函数 env_init()初始化,下面的函数来实现向 kernel 跳转。

theKernel (0, bd->bi_arch_number, bd->bi_boot_params);

thekernel 其实不是个函数,而是指向内核入口地址的指针,为 0x30008000。这里把它强行转化为带三个参数的函数指针,会把三个参数保存到通用寄存器中,实现了向 kernel 传递信息的功能,R0 赋值为 0,R1 赋值为机器号,R2 赋值为启动参数数据结构的首地址[6]。

标记列表实际上是内存中的结构体组成的列表,在./Lib_arm/Armlinux.c 中函数 setup_start_tag()来创建标记列表。

4.2 tftp 加载内核映像

对于已经编译好了的内核映像文件 zImage,其格式是 ELF 的可执行文件,首先要把它转化成 U-boot 格式的文件 uImage,实际是添加了一个 header 定义,直接用 tools 目录下的工具 mkimage 就可用实现,具体在终端中执行如下操作:

①arm-linux-objcopy -O binary -R .note -R .comment -S vmlinux linux.bin

②gzip -9 linux.bin

③mkimage -A arm -O linux -T kernel -C gzip -a 0x30008000 -e 0x30008040 -n "Linux Kernel Image" -d linux.bin.gz uImage

先从内核文件中提取二进制文件,然后压缩,最后构造文件头信息,包括名称、大小、类型、CRC 校验码等,添加的头信息占用 0x40 大小空间。完成后下载内核映像 uImage,如下操作:

开启主机 tftp 服务,将 uImage 放置 tftp 目录下,然后启动 Uboot,运行 tftp 下载,擦除、烧写 Nand Flash,具体如图 8 所示。

图 8 tftp 下载内核

最后烧写文件系统映像,与烧写内核映像一样,先 tftp 下载到内存,然后再烧写,不同类型的文件系统 nand 烧写命令不一样,本文用到的是 yaffs 文件类型,则通过 Nand write.yaffs 0x30000000 0x1d0000 $ (filesize)命令来烧写。

本文研究了 Uboot 在基于 S3C2440 系统上的移植,并且提出了可行性方案,通过边搭建硬件环境边调试 Uboot,使 Uboot 在嵌入式系统板上正常运行,实现了串口通信、网口下载、Flash 烧录等功能,并且成功引导了 Linux 系统,为后续的系统驱动程序开发奠定了基础,使得 Uboot 的移植具有开发的通用性。

参考文献

[1] 刘淼.嵌入式系统接口设计与 Linux 驱动程序开发[M].北京:北京航空航天大学出版社,2006.

[2] YAGHMOUR K. Building embedded Linux system[M]. [S.l.]:O'Reilly,2004.

[3] HENKEL J, XIAOBO SHARON HU, SHUVRA S. BHATTACHARYYA. Taking on the embedded system design challenge[J].IEEE Computer,2003,5(4):35-37.

[4] SD-Memory Card Specifications /Part1 Physical Layer Specification Version 1.01[R]. [S.l.]:SD Group, 2001.

[5] SUMSUANG ELECTRONICS. S3C2410X User's Manual[Z]. Republic of Korea: Sumsang,2003.

[6] Configurations.Denx software engineering[EB/OL]. (2006-7-23) http://www.denx.de/wiki/DULG/Manual.

(收稿日期:2010-08-25)

作者简介:

卢伟,男,1986 年生,硕士研究生,主要研究方向:无线传感器网络、ARM 传感器应用。

潘炼,男,1964 年生,教授,硕士,主要研究方向:智能检测及计算机应用。