

面向 Cortex-A8 微处理器的 U-boot 移植与研究

周 力, 姚茂群

(杭州师范大学, 浙江 杭州 311121)

摘 要:在 PC 机上电后会执行 BIOS 程序, BIOS 程序负责初始化 DDR 内存和硬盘, 从硬盘上将操作系统镜像读取到 DDR 中, 然后跳转到 DDR 中去执行操作系统直到 PC 机启动。在嵌入式系统中的部署和启动都是参考 PC 机的, U-boot 程序类似于 BIOS 程序, 部署在 Flash 上, 是内核运行前一段小程序, 完成对必要硬件的初始化, 传递内核启动信息, 提供命令行供人操作, 最终调用操作系统, 起到引导和加载内核的功能。文中详细介绍了基于 CortexA8 微处理器的 s5pv210 的移植与研究, 包括 U-boot 的启动、移植、配置等过程, 最终通过 tftp 加载内核和 nfs 远程挂载根文件系统, 在开发板上实现 Linux 系统的启动。

关键词:嵌入式系统; U-boot; s5pv210; 移植; 配置

中图分类号: TP368

文献标识码: A

文章编号: 1673-629X(2018)12-0179-06

doi: 10.3969/j.issn.1673-629X.2018.12.038

U-boot Porting and Research for Cortex-A8 Microprocessor

ZHOU Li, YAO Mao-qun

(Hangzhou Normal University, Hangzhou 311121, China)

Abstract: The BIOS program is responsible for initializing DDR memory and hard disk, reading the operating system image from the hard disk into DDR, and then jumping to DDR to execute the operating system until the PC starts. The deployment and startup in the embedded system are based on PC. The U-boot program is similar to the BIOS program and deployed on Flash. It is a small program before the kernel runs, completes the initialization of necessary hardware, passes the kernel startup information, provides the command line for human operation, and finally calls the operating system, playing boot and loading kernel functions. In this paper, we introduce the transplantation and research of s5pv210 based on CortexA8 microprocessor, including the process of U-boot startup, porting and configuration. Finally, we implement the startup of Linux system on the development board by tftp loading kernel and nfs remote loading root file system.

Key words: embedded systems; U-boot; s5pv210; transplantation; configuration

0 引言

U-boot 就是 universal bootloader 的简称, 即通用的启动代码, 在源代码级别具有移植性, 可以针对多个开发板进行移植。U-boot 起先是 SourceForge 的一个开源项目, 由一个人发起, 后由整个网络上所有感兴趣的人共同维护发展而来的一个 bootloader。U-boot 经过多年的发展, 已经成为 bootloader 的标准, 现在大部分的嵌入式设备都使用 U-boot 做为 bootloader。U-boot 的终极目的是启动内核, 在 U-boot 中事先给 Linux 内核准备一些启动参数放在内存中, 然后这些参数将被用来指导 Linux 内核的启动, 借助 U-boot 可以部署整个 Linux 系统, 包括 kernel、rootfs 的镜像, 也

可以进行 cpu 级和板级硬件管理。当启动内核以后, U-boot 的生命周期结束^[1]。

1 U-boot 移植准备

U-boot 源代码是从 U-boot 官网下载的, 由很多目录构成, 一些主要目录的功能如表 1 所示。本次实验中使用的 U-boot 版本是 U-boot 官方的 2013-10 版本, 在 windows 和 ubuntu 的共享文件下创建 u-boot-2013-10 目录, 并从官网下载 U-boot, 解压到共享文件夹下, 在 ubuntu 中也解压一份代码, 在 windows 中修改代码, 在 ubuntu 中编译代码, 以方便整个实验进行^[1-2]。

收稿日期: 2018-01-10

修回日期: 2018-05-16

网络出版时间: 2018-06-29

基金项目: 国家自然科学基金(61771179); 浙江省自然科学基金(LY15F010011)

作者简介: 周 力(1992-), 男, 硕士, 研究方向为嵌入式软件编程; 姚茂群, 教授, 研究方向为数字集成电路与系统、嵌入式系统与应用。

网络出版地址: <http://kns.cnki.net/kcms/detail/61.1450.TP.20180629.1707.070.html>

表 1 U-boot 主要源码目录与作用

源码	作用
arch	存放 cpu 相关信息和初始化代码
board	存放开发板相关信息
drivers	存放开发板必须用的驱动
fs	文件系统,用来管理 flash 等资源
include	所有的头文件存放在 include 目录下
lib	存放架构相关的库文件
net	网络相关的代码

2 U-boot 启动过程

一个同时安装有 bootloader、内核镜像和根文件系统镜像的固态存储设备的典型空间分配结构如图 1 所示。从固态存储设备上启动 bootloader 一般可以分为两个阶段,即阶段 1 和阶段 2。阶段 1 为汇编阶段、阶段 2 为 c 语言阶段。阶段 1 注重 SoC 内部,在 SRAM 中完成,阶段 2 注重 SoC 外部,在 DRAM 中完成^[2]。

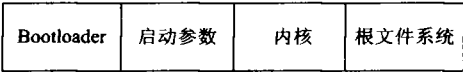


图 1 固态存储设备的典型空间分配结构

U-boot 的第一阶段主要完成异常向量表构建,设置 cpu 为 SVC 模式,初始化时钟,重定位,加载 U-boot 第二阶段代码到 DRAM 等一些工作,具体步骤如图 2 所示。

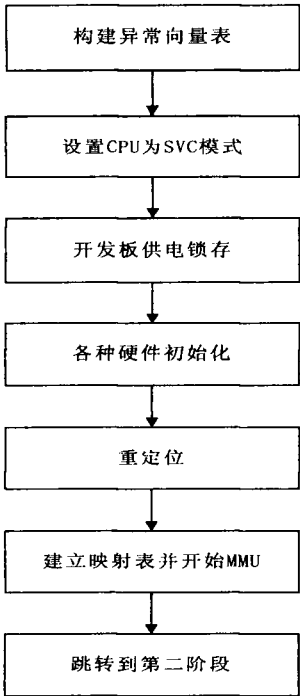


图 2 U-boot 第一阶段

U-boot 的第二阶段主要完成对开发板级别的硬件、软件数据结构进行初始化,包括网卡、控制台、SD 卡、环境变量等的初始化^[3]。具体步骤如图 3 所示。

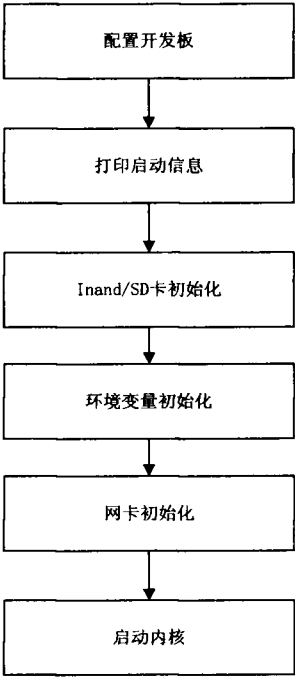


图 3 U-boot 第二阶段

3 U-boot 分析

U-boot 本质上是一个 C 语言项目,由很多个项目组成,可以通过 Makefile 来做项目管理,方便编译链接。在进行移植时,U-boot 中差别最大的是 board 和 cpu 这两个目录,这两个目录是和开发板有关的,在本次移植中使用的开发板 cpu 是三星公司的 s5pv210,所以要找 U-boot 中针对 s5pv210 或者 s5pc110 的 cpu 进行移植的作为参考^[4]。

3.1 U-boot 主 Makefile 分析

根据移植的一般规律,参考 include/configs/s5p_goni.h 这个头文件,找到对应的 board 在 board/samsung/goni 这个目录下。

这个版本中的 U-boot 的 Makefile 使用了 boards.cfg 文件,因此在 make xxx_config 编译生成 U-boot.bin 的可执行二进制文件时,这个 xxx 要到 boards.cfg 文件中查找。经过分析为 s5p_goni,配置时对应的 cpu、board 文件夹分别为:

```
cpu:u-boot-2013.10/arch/arm/cpu/armv7
board:u-boot-2013.10/board/samsung/goni
修改 Makefile 及 CROSS_COMPILE:
```

由于使用的 cpu 是三星公司的 s5pv210,属于 Cortex-A8 系列架构,所以它的 ARCH 属于 arm 系列,在 ubuntu 系统上找到交叉编译工具链安装的地方,复制相关路径到主 Makefile 中的 CROSS_COMPILE,具体代码修改如下^[5]:

```
ARCH=arm
CROSS_COMPILE=/usr/local/arm/arm-2009q3/bin/arm-
```

none-linux-gnueabi-

3.2 mkconfig 脚本分析

在 U-boot 的主 Makefile 中存在以下两行代码:

```
%_config::unconfig
@ $(MKCONFIG) -A $(@:_config=)
```

实际配置开发板时主 Makefile 传 2 个参数: -A 和 s5p_goni 到 mkconfig 脚本中, 在 mkconfig 脚本中使用 awk 正则表达式将 boards.cfg 文件中与刚才 s5p_goni 能够匹配上的那一行截取出来赋值给变量 line, 然后将 line 的内容以空格为间隔依次分开, 分别赋值给 \$1, \$2, ..., \$8。在解析完 boards.cfg 之后, \$1 到 \$8 就有了新的值, 将这些值用于开发板的配置, 具体的值如下所示^[5]:

```
$1=Active, $2=arm, $3=armv7, $4=s5pc1xx, $5=samsung, $6=goni, $7=s5p_goni, $8=-
```

4 U-boot 移植

在 C 语言中整个项目的入口就是 main 函数, 而在 U-boot 阶段由于有汇编阶段的参与, 整个程序的入口取决于 u-boot.lds 中 ENTRY 声明的地方。因此 start.S 即是整个 U-boot 的入口。在 start.S 中通过一句跳转指令 bl cpu_init_crit, 短跳转到 lowlevel_init 函数。lowlevel_init 函数中完成的工作有: 关看门狗、调用 uart_asm_init 来初始化串口、初始化时钟。跳转出来以后在 cpu_init_crit 函数中初始化完 DDR, 从而进入到第二阶段, 所以在这里执行重定位代码, 其中的 TEXT_BASE 是 U-boot 起始链接地址。U-boot 的第二阶段就在 crt0.S 文件中, 第二阶段的入口就是 _main 函数。第二阶段分成了 board_init_f 和 board_init_r 这两个函数, board_init_f 函数主要是各种板级初始化。初始化控制台、时钟、DRAM 等等。board_init_r 函数主要是各种外设的初始化, 如初始化 iNand/SD、环境变量、网卡等等。在本次实验中移植和分析 U-boot 最主要的几个模块。

4.1 DDR 移植

在 s5pv210 的核心板中有两块 DDR 内存, 每块 128 MB, 总共 256 MB。将三星版本的 U-boot 中的 DDR 内存初始化函数有关的 cpu_init.S 和头文件 s5pc110.h 复制到 U-boot-2013-10/board/Samsung/goni 下, 这部分代码必须在程序运行代码前 8 kb 以内, 否则校验会不通过, 导致 U-boot 无法正常启动。在 /board/Samsung/goni/Makefile 中和 /arch/arm/cpu/u-boot.lds 中分别做相应修改, 如下:

```
LOW:=lowlevel_init.o cpu_init.o
board/samsung/goni/lowlevel_init.o (.text *) board/samsung/goni/cpu_init.o (.text *)
```

下一步修改相应的 DDR 配置参数。将三星版本

的 include/configs/smdkv210single.h 中的 DDR 相关参数复制到 include/configs/s5p_goni.h 下, 查相关数据手册得, 在 s5pv210 的核心板中 MEMORY_BASE_ADDRESS 为 0x30000000^[6], 而在三星版本中 MEMORY_BASE_ADDRESS 为 0x20000000, 做如下修改:

```
#define MEMORY_BASE_ADDRESS 0x20000000 改为:
#define MEMORY_BASE_ADDRESS 0x30000000
#define DMC0_MEMCONFIG_0 0x20F01323 改为:
#define DMC0_MEMCONFIG_0 0x30F01323
```

4.2 重定位

从逻辑上来说, 重定位部分代码应该在 DDR 初始化之后和 U-boot 第二阶段来临前之间。U-boot 的第一阶段和第二阶段的划分并不是绝对的, 唯一必须遵循的就是第一阶段代码不能大于 8 KB。所以 U-boot 的第一阶段至少要完成 DDR 初始化和重定位。在满足这些条件之后, 选择在 u-boot-2013-10/board/samsung/goni/movi.c 中完成重定位工作。将三星版本中的 cpu/s5pc11x/movi.c 和 include/movi.h 分别复制到 u-boot-2013-10/board/samsung/goni 和 u-boot-2013-10/include 下^[6-7]。重定位完成以后, 清理完 bss 段, 数据段, 设置好栈, 用一句 ldrpc, __main 跳转到 _main 函数, 这也是 U-boot 第一阶段和第二阶段的分界线。修改相应的 makefile 和 u-boot.lds, 做如下修改:

```
LOW:=movi.o
board/samsung/goni/movi.o (.text *)
```

4.3 SD 卡驱动移植

4.3.1 SD 卡驱动移植分析

SD 卡驱动工作包含 2 部分内容, 一部分是 drivers/mmc 目录下的驱动, 另外一部分是 U-boot 自己提供的初始化代码, 譬如 GPIO 初始化、时钟初始化等。在 u-boot-2013-10 中, 驱动相关的文件主要有^[7]:

```
drivers/mmc/mmc.c
drivers/mmc/sdhci.c
drivers/mmc/s5p_sdhci.c
board/samsung/goni/goni.c
```

在三星移植版本中, 驱动相关的文件主要有:

```
drivers/mmc/mmc.c
drivers/mmc/s3c_hsmmc.c
cpu/s5pc11x/cpu.c
cpu/s5pc11x/setup_hsmmc.c
```

4.3.2 复制文件以及修改 Makefile

将三星的 drivers/mmc 复制到 u-boot-2013-10 的 drivers/mmc 中, 为了使原来三星中的 sdhci.o 和 s5p_sdhci.o 不发生编译错误, 需要将 drivers/mmc/Makefile 中涉及到的相应宏在 s5p_goni.h 中注释掉, 如下:

```
//#define CONFIG_SDHCI
```

```
//#define CONFIG_SSP_SDHCI
```

在 drivers/mmc/Makefile 中添加从三星复制过来的文件的编译项 COBJS-\$(CONFIG_S3C_HSMMC) += s3c_hsmmc., 再在 u-boot-2013-10/include/configs/s5p_goni.h 中添加原来在三星定义的关于 SD 卡的宏, 主要如下^[8]:

```
#define CONFIG_S3C_HSMMC
#define SDMMC_BLK_SIZE      (0xD003A500)
#define COPY_SDMMC_TO_MEM  (0xD003E008)
#define USE_MMC0
#define USE_MMC2
#define MMC_MAX_CHANNEL4
```

4.3.3 驱动相关移植

对比两个 U-boot, 找到 u-boot-2013-10 中没有定义的函数的头文件所在路径, 将三星版本的 include 目录下的 mmc.h 和 s3c_hsmmc.h 移植到 u-boot-2013-10 中 include 目录下。将三星的 drivers/mmc.c 中的 cpu_mmc_init 函数复制到 board/samsung/goni 中 board_mmc_init 函数中, 移植三星的 cpu/s5p11x/setup_hsmmc.c 到 board/Samsung/goni 下, 在 goni 目录下的 Makefile 中添加 COBJS-y:=goni.o setup_hsmmc.o。

在本次实验中用 SD 卡从 ubuntu 中拷贝程序, 整个 U-boot 能成功启动, 即代表 SD 卡驱动移植成功。

4.4 环境变量的移植

4.4.1 环境变量理解

环境变量有 2 份, 一份在 Flash 中, 另一份在 DDR 中。U-boot 开机时一次性从 Flash 中读取全部环境变量到 DDR 中作为环境变量的初始化值, 使用过程中都是用 DDR 中这一份, 可以用 saveenv 指令将 DDR 中的环境变量重新写入 Flash 中去更新 Flash 中环境变量。下次开机时又会从 Flash 中再读取一次^[8]。

4.4.2 环境变量保存位置分析

U-boot 烧录时使用的扇区数是 SD 卡的扇区 1-

16 和 49-x(x-49 大于等于 U-boot 的大小)。从 U-boot 的烧录情况来看, SD 卡的扇区 0 空闲, 扇区 1-16 被 U-boot 的 BL1 占用, 扇区 17-48 空闲, 扇区 49-x 被 U-boot 的阶段二占用。再往后就是 kernel, rootfs 等镜像的分区了。所以 ENV 不能往扇区 1-16 或者 49-x 中来放置, 其他地方都可以。ENV 的大小是 16 K 字节也就是 32 个扇区。在环境变量移植中宏 CONFIG_ENV_OFFSET 决定了环境变量在 SD 卡中相对 SD 卡扇区 0 的偏移量, 也就是 ENV 写到 SD 卡的哪里去了。修改这个 CONFIG_ENV_OFFSET 的值, 将 ENV 写到从第 17 扇区开始的地方, 具体如下:

```
#define CONFIG_ENV_OFFSET 17 * 512 //第17扇区开始的位置
```

```
#define MOVI_BL2_POS ((eFUSE_SIZE/MOVI_BLK_SIZE)+MOVI_BL1_BLKCNT+ MOVI_ENV_BLKCNT)
```

宏 MOVI_BL2_POS 决定了 U-boot 的 BL2 开始扇区, 宏(eFUSE_SIZE/MOVI_BLK_SIZE)为 1, 代表扇区 0, 宏 MOVI_BL1_BLKCNT 为 16, 存放 U-boot 的 BL1, 宏 MOVI_ENV_BLKCNT 为 32, 存放 ENV, 从扇区 49 开始就是 U-boot 的 BL2 了。为了实现环境变量不冲突, 将 ENV 放到 17 扇区起始的地方即可。

4.4.3 环境变量的测试

程序修改重新编译后启动, 为了确保 iNand 里面没有之前保存过的环境变量, 对 iNand 的前 49 个扇区进行擦除。使用命令: mmc write 0 30000000 0# 49 来擦除 SD 卡的扇区 0-48, 这样以前的环境变量都没有了。重新开机后先修改 bootdelays 等于 3 作为标记后 saveenv 保存环境变量后重启。测试方法是, 使用命令 mmc read 0 30000000 17# 32 将 iNand 的 17 开始的 32 个扇区读出来到内存 30000000 处, 用 md 命令查看。找到显示区域里面的各个环境变量, 看到读出来的值与刚才标记的值一样, 说明修改成功^[9]。具体如图 4 所示。

```
x210 # mmc read 0 30000000 17# 32
MMC read: dev # 0, block # 17, count 32 ... 32 blocks read: OK
x210 # md
30000000: 64a236a2 746f6f62 3d646d63 69766f6d .6..bootcmd=movi
30000010: 61657220 656b2064 6c656e72 30303320 read kernel 300
30000020: 30303830 6d203b30 2069766f 64616572 08000: movi read
30000030: 6f6f7220 20736674 30423033 30303030 rootfs 30800000
30000040: 30303320 3b303030 6f6f6220 33206d74 300000: bootm 3
30000050: 38303030 20303030 30423033 30303030 0008000 30800000
30000060: 64746d00 74726170 3030383d 34203030 .mtdpart=80000 4
30000070: 30303030 3032030 30303030 61620030 00000 3000000.ba
30000080: 61726475 313d6574 30323531 74650030 udrate=115200.et
30000090: 64646168 30303d72 3a30343a 323a6335 haddr=00:40:5c:2
300000a0: 61303a36 0062353a 6d74656e 3d6b7361 6:0a:5b.netmask=
300000b0: 2e353532 2e353532 00302e30 64617069 253.253.0.0.ipad
300000c0: 313d7264 312e3237 34362e37 3838312e dr=172.17.64.188
300000d0: 72657300 69726576 37313d70 37312e32 .server=172.17
300000e0: 2e34362e 00373631 65746167 69796177 .64.167.gateway1
300000f0: 37313d70 37312e32 2e34362e 6f620031 p=172.17.64.1.bo
30000100: 6564746f 3d79616c 6f620033 7261746f otdelay=3.bootar
30000110: 633d7367 6f736e6f 743d656c 41537974 gs-console=ttySA
30000120: 312c3243 30323531 6f722030 2f3d746f C2.115200 root=/
30000130: 2f766564 62636d6d 70306b6c 77722032 dev/mmcblkOp2 rw
30000140: 696e6920 6c2f3d74 78756e69 72206372 init=/linuxrc r
30000150: 6e746f6f 70797473 78653d65 00003274 ootfstype=ext2..
30000160: 2e37312e 312e3436 313a3736 312e3237 17.64.167:172.1
30000170: 34362e37 323a312e 222e3535 322e3235 7.64.1:255.255.2
30000180: 302e3535 74653a3a 6f3a3068 6920666e 55.0:eth0:off 1
30000190: 3d74696e 6e596c2f 63727875 6e6f6320 nit=/linuxrc con
300001a0: 656c6f73 7974743d 3234153 3531312c sole=ttySAC2.115
300001b0: 00303032 746f6f62 616c6564 00333d79 200.bootdelay=3.
```

图4 环境变量测试效果

4.5 网卡移植

4.5.1 网卡移植分析

U-boot 中对各种功能的实现是一种可配置可裁剪的设计,默认情况下 U-boot 没有选择支持网络。当在配置头文件中添加一行 `#define CONFIG_CMD_NET`,即添加了网络支持宏之后,U-boot 的 `board_init_r` 函数中初始化时就会执行 `eth_initialize` 函数,从而网络相关代码初始化就会被执行,将来网络就可以使用^[10]。

4.5.2 设置网络相关的环境变量

```
#define CONFIG_ETHADDR00 40:5c:26:0a:5b
#define CONFIG_NETMASK 255.255.255.0 //子网掩码
#define CONFIG_IPADDR 172.17.64.188 //开发板的地址
#define CONFIG_SERVERIP 172.17.64.167 //虚拟机 IP 地址
#define CONFIG_GATEWAYIP 172.17.64.1 //网关
```

4.5.3 添加 ping 和 tftp 命令

在 Linux 系统中网络底层驱动被上层应用调用的接口是 `socket`,是一个典型的分层结构,底层和上层是完全被 `socket` 接口隔离的。但是在 U-boot 中网络底层驱动和上层应用是不分层的^[11-12]。即上层网络的每一个应用都是自己去调用底层驱动中的操作硬件的代码来实现的。U-boot 中有很多预先设计的需要用到网络的命令,直接相关的就是 `ping` 和 `tftp` 这两个命令。这两个命令在 U-boot 中也是需要用相应的宏开关来打开或者关闭的。

经分析,发现 `ping` 命令开关宏为 `CONFIG_CMD_PING`,而 `tftp` 命令的开关为 `CONFIG_CMD_NET`,已经存在。所以只需在 `include/configs/S5p_goni.h` 中添加 `ping` 命令相关宏定义即可,如下:

```
#define CONFIG_CMD_PING
```

4.5.4 移植及注册网卡驱动

为了让网卡初始化函数 `dm9000_pre_init` 在 `u-boot-2013-10/board/samsung/goni.c` 中的 `board_init` 函数下调用,添加 `dm9000_pre_init` 函数到 `goni.c` 下,将网卡相关的宏添加到 `s5p_goni.h` 下,并且添加上相应的头文件。添加的宏如下:

```
#define CONFIG_DRIVER_DM9000 1
#define CONFIG_DM9000_BASE(0x88000300) //bank1 基地址
#define DM9000_IO(CONFIG_DM9000_BASE) //IO 基地址
#define DM9000_DATA(CONFIG_DM9000_BASE+4) //ADD2 加 4 为了得到地址
```

在 Linux 的网卡驱动体系中,有一个数据结构

(`struct eth_device`)用来封装一个网卡的所有信息,系统中注册一个网卡时就是要建立一个这个结构体的实例,然后填充这个实例中的各个元素,最后将这个结构体实例加入到 `eth_devices` 链表上,就完成了注册^[13]。在 `/drivers/net/dm9000x.c` 中的最后一个函数 `int dm9000_initialize(bd_t *bis)`,就是用来注册 `dm9000` 网卡驱动的。自己定义一个 `board_eth_init` 函数用来做网卡驱动添加工作,即可以注册驱动成功。在本次实验中能使用 `tftp` 加载内核最终启动 Linux 系统则说明网卡驱动移植成功。

4.6 实验结果

当 U-boot 移植完以后,需要通过设置 `tftp` 加载 `ubuntu` 中编译生成的内核镜像和用 `nfs` 的方式远程挂载 `ubuntu` 中制作好的根文件系统。配置好 `tftp` 和 `nfs` 以后,设置 U-boot 的 `bootargs`,如根目录所在路径,主机 IP,开发板 IP,网关,串口,波特率等以支持 `nfs` 方式挂载 `rootfs`,具体设置如下^[13-14]:

```
setenv bootargs root=/dev/nfs
nfsroot=192.168.1.141:/root/porting_x210/rootfs
ip=172.17.64.188:172.17.64.167:172.17.64.1:255.255.255.0::eth0:off
init=/linuxrc console=ttySAC2,115200
```

设置完成后,成功加载内核和根文件系统,整个 Linux 系统启动起来。成功启动 U-boot 如图 5 所示,成功启动 Linux 系统如图 6 所示。

```
OK
U-Boot 1.3.4-dirty (May 21 2016 - 18:17:00) for x210

CPU: S5PV210@1000MHz(OK)
APLL = 1000MHz, Hclkmsys = 200MHz, Pclkmsys = 100MHz
MPLL = 667MHz, EPPLL = 96MHz
Hclkdsys = 166MHz, Pclkdsys = 83MHz
Hclkpsys = 133MHz, Pclkpsys = 66MHz
SCLKA2M = 200MHz

Serial = CLKUART
Board: x210
DRAM: 512 MB
Flash: 8 MB
SD/MMC: 3728MB
In: serial
Out: serial
Err: serial
[LEFT UP] boot mode
checking mode for fastboot ...
Hit any key to stop autoboot: 0
```

图 5 U-boot 成功启动

```
5.516997 smdkc110-rtc smdkc110-rtc: rtc disabled, re-enabling
5.522833 smdkc110-rtc smdkc110-rtc: setting system clock to 2010-01-01 12:00:04 UTC (1262347204)
5.533334 FIMCO registered successfully
5.537140 FIMC1 registered successfully
5.541165 FIMC2 registered successfully
5.543881 <db>S5PC110 TVOUT Driver, (c) 2009 Samsung Electronics
5.550574 S5P-TVOUT S5P-TVOUT: hpd status is cable removed
5.561627 DRUG_PORT must not use APC!
5.565323 EXT2-fs (mmcblk0p2): warning: mounting unchecked fs, running e2fsck is recommended
5.574199 VFS: Mounted root (ext2 filesystem) on device 179:2.
5.578769 Freeing init memory: 172K
6.459117 eth0: link down
6.460663 ADDRCONF(NETDEV_UP): eth0: link is not ready
zhouli login: root
login[58]: root login on 's3c2410_serial2'
[root@zhouli ~]# ls
Makefile hello.c hello_dynamic hello_static
```

图 6 Linux 系统成功启动

5 结束语

在 Linux 系统中 U-boot 是必不可少的一部分,完成了包括 `cpu` 级和板级必要硬件设备的初始化,为操

作系统的启动提供了必要的条件和参数。在本次移植过程中,详细阐述了 U-boot 的启动过程,分析了 U-boot 的主 Makefile 和 mkconfig,介绍了 U-boot 移植中主要几个移植对象的移植方法。实验结果表明,将官方 U-boot 移植到基于 s5pv210 的 x210 的开发板上可行,同时通过 tftp 的方式加载内核和 nfs 方式挂载启动 ext2 的根文件系统,从而启动整个 Linux 系统,为以后嵌入式系统开发提供必要环境。

参考文献:

- [1] WELSH M, KAUFMAN L. Linux 权威指南[M]. 洪峰,译. 第3版. 北京:中国电子出版社,2000.
- [2] 邹思秩. 嵌入式 Linux 设计与应用[M]. 北京:清华大学出版社,2002.
- [3] 马丽洁. 嵌入式 Linux 应用编程[M]. 北京:北京理工大学出版社,2016.
- [4] 熊星星,何月顺. 基于 S5PV210 的 U-boot 分析与移植[J]. 计算机系统应用,2015,24(1):199-205.
- [5] 张伟,刘斌,董群峰. 基于 S3C2410 上 U-Boot 的移植与实现[J]. 计算机系统应用,2014,23(9):204-207.
- [6] 史巧硕,范东月,柴欣,等. 嵌入式 Linux 根文件系统的构建与分析[J]. 计算机测量与控制,2015,23(2):656-

659.

- [7] 杜海星. 基于 ARM 的嵌入式 Bootloader 分析与移植[J]. 微计算机信息,2010,26(29):58-59.
- [8] 申爽. 基于 S3C2440 的 Uboot 分析与移植[J]. 计算机系统应用,2012,21(5):222-225.
- [9] 朱吉庆. 工程项目中的 VxWorks 操作系统移植与网络驱动分析实现[D]. 西安:西安电子科技大学,2008.
- [10] 刘启军,程明. 嵌入式 linux 中以太网设备驱动的设计与实现[J]. 通信技术,2009,42(9):145-147.
- [11] GENG Qingtian, SUN Zhanchen, ZHAO Hongwei, et al. The U-boot transplantation based on S3C2440[C]//International conference on mechatronic science, electric engineering and computer. Jilin, China: IEEE, 2011: 2168-2171.
- [12] WANG Lei. Realization of U-Boot booting through NAND flash[J]. Electronic Design Engineering, 2010, 18(5): 98-100.
- [13] BLUM R. Linux command line and shell scripting bible[M]. [s. l.]: John Wiley & Sons, 2015.
- [14] NAM S, YOON S K, KIM S D. Fast bootstrapping method for the memory-disk integrated memory system[C]//IEEE/ACIS international conference on computer and information science. Las Vegas, NV, USA: IEEE, 2015: 167-172.

(上接第 178 页)

参考文献:

- [1] CHAN T F, SHEN J. Mathematical models for local nontexture inpaintings[J]. SIAM Journal on Applied Mathematics, 2002, 62(3): 1019-1043.
- [2] CHAN T F, SHEN J. Nontexture inpainting by curvature-driven diffusions (CDD)[J]. Journal of Visual Communication and Image Representation, 2001, 12(4): 436-449.
- [3] ZHANG X, CHAN T F. Wavelet inpainting by nonlocal total variation[J]. Inverse Problems and Imaging, 2010, 4(1): 191-210.
- [4] 方宝龙. 基于纹理合成的图像修复算法研究[D]. 济南:山东大学, 2013.
- [5] DRORI I, COHEN-OR D, YESHURUN H. Fragment-based image completion[J]. ACM Transactions on Graphics, 2003, 22(3): 303-312.
- [6] TANG Feng, YING Yiting, WANG Jin, et al. A novel texture synthesis based algorithm for object removal in photographs[C]//Proceedings of the 9th Asian computing science conference on advances in computer science. ChiangMai, Thailand: Springer-Verlag, 2004: 248-254.
- [7] 张东,唐向宏,张少鹏,等. 小波变换与纹理合成相结合

的图像修复[J]. 中国图象图形学报, 2015, 20(7): 882-894.

- [8] 翟东海, 鱼江, 段维夏, 等. 破损区域分块划分的图像修复[J]. 中国图象图形学报, 2014, 19(6): 835-842.
- [9] 王猛, 翟东海, 聂洪玉, 等. 邻域窗口权重变分的图像修复[J]. 中国图象图形学报, 2015, 20(8): 1000-1007.
- [10] STARCK J L, ELAD M, DONOHO D L. Image decomposition via the combination of sparse representations and a variational approach[J]. IEEE Transactions on Image Processing, 2005, 14(10): 1570-1582.
- [11] MAIRAL J, ELAD M, SAPIRO G. Sparse representation for color image restoration[J]. IEEE Transactions on Image Processing, 2008, 17(1): 53-69.
- [12] 李民, 程建, 李小文, 等. 非局部学习字典的图像修复[J]. 电子与信息学报, 2011, 33(11): 2672-2678.
- [13] SAHOO S K, LU W. Image inpainting using sparse approximation with adaptive window selection[C]//7th international symposium on intelligent signal processing. [s. l.]: IEEE, 2011: 1-4.
- [14] 李志丹, 和红杰, 尹忠科, 等. 基于 Curvelet 方向特征的样本块图像修复算法[J]. 电子学报, 2016, 44(1): 150-154.