第 29卷第 4期
2008年 4月
微 计 算 机 应 用
MICROCOMPUTER APPLICATIONS
Vol. 29 No. 4
Apr. 2008

U - Boot在 S3c2410上的移植

王淑贞 姚 铭 周结华

(厦门大学信息科学与技术学院 厦门 361005)

摘要:基于 Samsung公司的板子 S3c2410,实现 U - Boot的移植,并成功启动内核和加载文件系统。首先了解移植环境,本文主要是针对最小硬件系统的移植;分析 U - Boot的启动原理和启动流程,深刻剖析 U - Boot阶段 1和阶段 2的运行过程;在通用的 U - Boot基础上修改其硬件配置,对应于 S3c2410板子上的硬件;编译 U - Boot,用 JTAG口将 U - Boot烧写到板子;最后利用 U - Boot命令下载,烧写内核和文件系统,复位后成功启动内核,这一步同时证实了所移植的 U - Boot的正确性。

关键词: U - Boot S3c2410板 交叉编译工具 硬件配置

Transplant U - BOOT to S3c2410 Board

WANG Shuzhen, YAO Ming, ZHOU Jiehua

(College of Information Science and Technology, Xiamen University, Xiamen, 361005, China)

Abstract: Abstract In this paper, the transplantation of U - Boot which is based on Samsung Company's board S3C2410 is implemented, moreover it leads the system to startup the kernel and load the Ramdisk filesystem successfully. First, come to know the environment of transplantation, this paper mainly relates to the smallest hardware system. Analyze the startup theory and flow of the U - Boot, dissect its operation process of the stage1 and stage2. Modify the hardware configuration based on the universal U - Boot - 1.1.6 according to the hardwares that are on the board S3C2410. Compile the U - Boot and then download the U - Boot to the board by JTAG. Lastly, utilize the U - Boot commands to download the linux kernel and filesystem to flash, then reset and it can startup kernel successfully. This also indicates that the U - Boot is accurate.

Keywords: U - Boot S3c2410 board, crossstool, hardware configuration

U - BOOT类似 Windows下的 BIOS,全称 Universal Boot Loader,是遵循 GPL条款的开放源码项目。从 FADSROM、8xxROM、PPCBOOT逐步发展演化而来。U - Boot不仅仅支持嵌入式 Linux系统的引导,当前,它还支持 NetBSD, VxWorks, QNX, RTEMS, ARTOS, LynxOS嵌入式操作系统。这是 U - Boot中 Universal的一层含义,另外一层含义则是 U - Boot除了支持 PowerPC系列的处理器外,还能支持 MIPS、x86、ARM、NIOS、XScale等诸多常用系列的处理器。这两个特点正是 U - Boot项目的开发目标,即支持尽可能多的嵌入式处理器和嵌入式操作系统。就目前来看,U - Boot对 PowerPC系列处理器支持最为丰富,对 Linux的支持最完善。U - Boot有其独特的优势,源码开放,支持多种嵌入式操作系统,支持多个处理器系列,有较高的可靠性和稳定性,有丰富的开发调试文档与强大的网络技术支持。这些优点是致使它被广泛应用的原因,也是选择移植到 S3c2410板子上的原因。本文下载了 U - Boot - 1.1.6,修改并移植到 S3c2410板子上,成功地启动内核。

1 移植环境

硬件环境: cpu: SAMSUNG S3C2410; flash: 8M 16bit intel E28F128J3A - 1; (16MB),地址范围: 0x01000000 - 0x02000000; sdram: 16M 16bit HY57V561620BT X2 (64MB),地址范围: 0x30000000 - 0x34000000; ethernet: DM9000; serial: RS232; JTAG: JTAG连接线。

本文于 2008 - 01 - 18收到。

软件环境:U-Boot-1.1.6:http://sourceforge.net(下载地址);交叉编译工具:arm-9tdmi-linux-gnu-gcc(通过下载工具自己制作的);烧写工具:sjf2410;内核(移植到板子上):Linux-2.6.20.1。

本项目所采用是华恒 HHARM2410开发板,此开发板采用 203MHz的 ARM920T内核的处理器 S3C2410,其内部集成了微处理器和一些手持设备的常用外围组件,可用于各种领域。它是应用于手持设备的低成本实现,提供了更高性价比。系统具有体积小、耗电低、处理能力强、等特点,能够装载和运行嵌入式 Linux操作系统。

嵌入式系统的具体硬件设计会随着设计应用系统的不同而有所差别。一般情况下,用户可以根据自己的要求,选用合适的微处理器类型,根据相应的接口电路,搭配不同的外设,构成不同用途、不同规模的应用系统。而本项目采用最小硬件系统设计,其由处理器与 Flash和 SDRAM等外围电路构成的设计原理图如图 1所示。

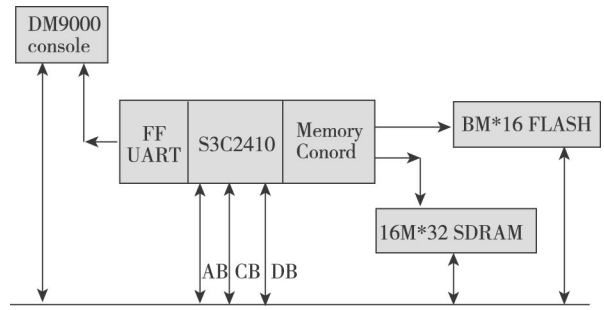

图 1 硬件总体连接图

2 U-Boot启动原理

(1)启动模式。U-Boot提供启动加载和下载两种工作模式。启动加载模式也称自主模式,一般是将存储在目标板 Flash中的内核和文件系统的镜像装载到 SDRAM中,整个过程无需用户的介入。在使用嵌入式产品时,一般工作在该模式下。工作在下载模式时,目标板往往受外设(一般是 PC机)的控制,从而将外设中调试好的内核和文件系统下载到目标板中去。在烧写、调试 U-Boot时就是工作在此模式下,进入该模式可以对 flash进行操作,比如写保护,擦写,也可以设定环境变量等等。U-Boot允许用户在这两种工作模式间进行切换。通常目标板启动时,会延时等待一段时间,如果在设定的延时时间范围内,用户没有按键,U-Boot就进入启动加载模式。

(2)U-Boot启动流程。Bootloader是系统加电后运行的第一段代码,其最终目标是正确地调用内核来执行。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带到一个合适的状态,以便为最终调用内核和应用程序准备好正确的环境。

Bootloader的实现依赖于 CPU的体系结构,因此,大多数 Bootloader的执行都可分为两个阶段。阶段 1通常都是用汇编语言来实现,而阶段 2通常是用 C语言来实现,以实现复杂的功能,而且代码有更好的可读性和可移植性。阶段 1的代码是在 Flash中运行的,而阶段 2的代码是将其从 Flash复制到 RAM中运行的。其阶段 1完成的任务包括如下:①硬件设备初始化;②为加载 U-Boot的阶段 2代码准备 RAM空间;③复制 U-Boot的阶段 2代码到 RAM空间中;④设置好堆栈;⑤跳转到阶段 2代码的 C入口点。

U-Boot中有一脚本文件 U-Boot.lds,该文件用来设置 U-Boot中各个目标文件的连接地址。这个脚本文件定义目标程序各部分链接顺序。打开 U-Boot.lds可以看出第一要链接的是 cpu/arm920t/start.o,即阶段 1的执行过程主要就是指 cpu/arm920t/start.S的执行过程。start.S文件主要是 CPU的初始化,用汇编语言编写,其主要代码流程如图 2所示。阶段 2完成的任务包括。①初始化本阶段要使用到的硬件设备;②检测系统内存映射(Memory map);③将内核映像和根文件系统映像从 Flash上读到 RAM空间中;④为内核设置启动参数;⑤启动内核。

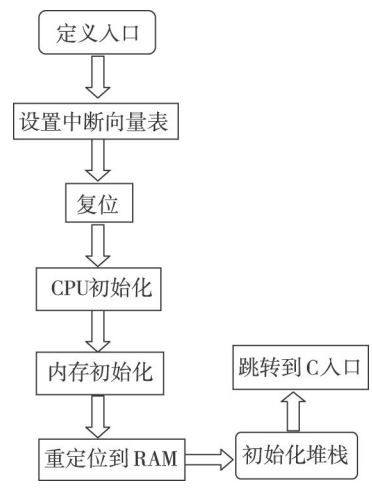

图 2 阶段 1启动流程图

start_armboot()是 U-Boot执行的第一个 C语言函数,完成系统初始化工作,即该阶段所用的硬件设备的初始化和命令控制台的初始化,其中设备初始化如 serial_init(),cpu_init(),flash_init(),eth_init()等等,而命令控制台初始化主要由 main_loop()函数来完成,该函数主要是处理命令控制台,包括初始化命令控制台、接收命令输入、自动延时等待、自动执行 bootmcmd命令等。main_loop()函数主要用于设置延时等待,从而确定目

标板是进入下载操作模式还是装载镜像文件启动内核。在设定的延时时间范围内,目标板将在串口等待输入命令。若没有接到相关命令,系统将自动进入装载模式,执行 go 30008000命令,程序进入 do_go()函数,启动内核并加载文件系统,或者执行 bootm 30008000,程序进入 do_bootm_linux()函数,启动内核。具体执行哪个命令,就看头文件 S3c2410中环境变量 bootcmd如何设置而定。

3 U - Boot移植过程

Bootloader除了依赖于 CPU的体系结构外,Bootloader实际上也依赖于具体的嵌入式板极设备的配置。移植 U - Boot只需修改通用的 U - Boot - 1.1.6,它本身就适用于很多不同类型的板子。我们可以充分利用 U - Boo资源,选择与自己的板子最为相近的作为基础。在这里我选择 SMDK2410的板子,因为它跟我所要用的板子的硬件配置是最接近的,两者的 CPU体系结构是一样的,也就是有相同核 arm920t。但两者的板极设备的配置不尽相同,比如 flash, sdram,网卡等不同啊,我们需要修改它的硬件配置才能为己所用。具体的修改过程如下:

(1)在 board目录下创建一新目录,命名为 S3c2410(此名可以根据自己喜好所命)。

(2)修改 U - Boot - 1.1.6目录下的 Makefile,它负责配置 U - Boot的编译方式。Makefile的内容从上到分别是:定义编译环境:使用何种编译器、编译方式、目标文件的生成及它们最终镜像中的链接次序等,因此需在该文件添加如下内容:

```
S3c2410_config : unconfig
@ $(MKCONFIG) $(@: _config = ) arm arm920t S3c2410 NULL S3c24x0
```

其中 arm指 cpu体系结构,arm920t指 cpu, S3c2410是自己主板对应的目录,S3c24x0指片上系统。编译后会自动在 U - Boot - 1.1.6/include/下产生 config.mk文件,内容如下:

```
ARCH = arm
CPU = arm920t
BOARD = S3c2410
SOC = S3c24x0
```

把 smdk2410下的所有文件复制到 S3c2410目录下:

(1)修改 S3c2410下的 Makefile和 config.mk,Makefile是把 smdk2410.o改为 S3c2410.o, onfig.mk中的 TEXT_BASE是指 U - Boot被复制到内存的基址,把它改为 0x33A00000。

(2) S3c2410下的 lowlevel_init.S,该文件是用来配置和初始化内存的,根据自己的 SDRAM型号修改,本项目所用的是两片 HY57V561620BT级联而成,数据位是 32位,总容量是 64MB,因此应该把 .word 0x32改为 .word 0x31,因为 word 0x32对应是的 128MB。

(3) S3c2410下的 flash.c是读、写和删除 Flash的源代码文件,在源代码中有一系列对 flash进行操作的函数,如: flash_init(),flash_print_info()打印 flash信息, flash_erase(), write_buff()从内存中复制数据, write_word()将数据写入 flash中, flash_real_protect()等等。

不同型号的 Flash其初始化,读,写方式可能就不同。在 board目录中 CMI主板对应的 FLASH正是本板子上 flash的型号 INTEL E28F128,因此只需把 cmi中 flash.c拷到 S3c2410目录代换原来的 flash.c, cmi中的 flash.c在写入时要交换字节,所以删除它的 write_short()和 wirte_buff()函数,把 ep7312主板目录中 flash.c的 wirte_word()和 wirite_buff()函数复制过来。把 flash.c中的 FLASH_BASE0_PRELM改为 CFG_FLASH_BASE。把 FLASH_BLOCK_SIZE改为 0x20000,(E28F128J3A flash中块的大小是 128K)。

(4)复制 /include/configs/smdk2410.h,重新命名为 S3c2410.h,并对它进行修改,该文件和硬件配置息息相关,具体修改如下:

①由于本项目中 S3c2410板子用的网卡是 dm9000而不是 CS8900,因此把有关 cs8900的宏定义删除,加上 dm9000的宏定义,如下所示:

```
#define CONFIG_DRIVER_DM9000 1
#define CONFIG_DM9000_BASE 0x8000300
#define DM9000_IO CONFIG_DM9000_BASE
#define DM9000_DATA (CONFIG_DM9000_BASE + 4)
#define CONFIG_DM9000_USE_16BIT 1
#define COMMAND_LINE " initrd = 0x31000000,
```

0x440000 mem = 64M root = /dev/ram init = /linuxrc console = ttySAC0"

②修改控制台命令提示符,把"SMDK2410 #" - - >"S3C2410 #"这个完全可以根据自己的喜好去改。

#define CFG_PROMPT "S3C2410 # "

③由于 smdk2410.h中有关 SDRAM的容量,起始地址等都与 S3c2410板子对应的 SDRAM一致,所以这一部分不需要修改。

④关于 flash,因为 S3C2410用的 flash是 intel E28F128J3A - 150,而不是 amd lv400和 lv800,所以删除此文件中有关 lv400和 lv800的所有宏定义。另外添加以下对 intel E28F128J3A - 150的宏定义语句(根据此型号的 flash作的如下修改):

```
#define PHYS_FLASH_SIZE 0x01000000
/* 16MB 3/ FLASH的容量大小 */
#define CFG_MAX_FLASH_SECT (128)
/* 最大扇区数 */
#define CFG_ENV_ADDR (CFG_FLASH_BASE + 0x60000) /
* 环境变量的地址 */
#define CFG_MONITOR_BASE PHYS_FLASH_1
#define CFG_FLASH_PROTECTION 1
#define CFG_ENV_SIZE 0x20000
/* 环境变量所占的内存大小 */
#define CFG_FLASH_ERASE_TOUT (2 * CFG_HZ) /*
FLASH擦除超时时间 */
#define CFG_FLASH_WRITE_TOUT (2 * CFG_HZ) /* FLASH
写超时时间 */
```

CFG_FLASH_PROTECTION这个宏定义,如果定义了,那么当 U - Boot成功运行后,flash永远被保护起来,不管你用命令 protect off all怎么开启,它永远被保护。

⑤cpu/arm920t/start.S,在将 flash上的代码复制到 RAM之前,有:

```
#ifndef CONFIG_SKIP_LOWLEVEL_INIT
bl cpu_init_crit
#endif
```

这一段是判断是否进行 cpu包括内存在内的初始化,若想在内存中调试 U - Boot,则此时不能对内存进行初始化,否则就相当于把刚下载到内存的代码又删除,故此时应在上述语句之前加上 #defined CONFIG_SKIP_LOWLEVEL_INIT。若是想烧写到 flash中直接调试的,则一定要内存初始化,否则串口将无输出,看了一下程序,发现将代码复制完之后下面还有这么一段:

```
#ifndef CONFIG_SKIP_LOWLEVEL_INIT
cpu_init_crit:
```

也就是说即使没有定义 CONFIG_SKIP_LOWLEVEL_INIT,它也会初始化两次内存,就相当于刚复制到内存中的代码又被初始化掉,所以需要把后面这一部分屏蔽使它无法执行。即去掉文件中的条件语句 #ifndef CONFIG_SKIP_LOWLEVEL_INIT。这样整个过程就只执行一次内存初始化。

4 编译、烧写 U - Boot,并启动内核

(1)编译 U - Boot编译 U - Boot的目的是产生可运行于板子上的映像文件,其所用的交叉编译工具是 arm - 9tdmi - linux - gnu - gcc,此工具是通过参考网上的资料自己制作的。在编译之前需把它的路径添加到 PATH环境变量中,即把路径添加到 .bash_profile中。或者通过修改 U - Boot - 1.1.6目录的 Makefile文件,指定 Makefile中的 CROSS_COMPILE宏。设置完路径后,接下来按顺序打入如下命令:

```
make distclean清除一下之前编译产生的所有文件
make S3c2410_config编译自己的板子 S3c2410
make开始编译
```

这种情况下产生的文件是在默认的文件夹下,当然也可以指定目录,具体方法可以查看 /U - Boot - 1.1.6/README,里面有详细的说明。编译成功之后会产生三个重要映象文件:U - Boot.bin,U - Boot和 U - Boot.srec文件,其中 U - Boot是一个二进制的源映像文件,U - Boot.bin是个 elf格式的二进制映像文件,而U - Boot.srec是个摩托罗拉 S - Record格式的映像文件,而烧写到板子上的是 U - Boot.bin。

(2)烧写 U - Boot。移植 U - Boot的最后一步,其实就是将 U - Boot.bin烧写到板子的 flash中。有两种

烧写方法:

方法一:即用 JTAG烧写,当开发板没有 Bootloader,或板子上的 Bootloader被损坏时,这时无法用 U - Boot或 ppcboot命令烧写,此时只能用 JTAG接口烧写 Bootloader。由于板子有 JTAG口,用板子的专用 JTAG烧写软件 sjf2410,就可以将 U - Boot. bin烧写到 flash中,步骤如下:

①首先将 JTAG烧写软件 sjf2410和 U - Boot. bin放在同一个目录下,然后进入这个目录,打入命令语句:. / sjf2410 /f: U - Boot. bin即可立马准备烧写程序。

②选择 flash对应的型号,本项目中选择的是 intel公司的 E28F128J3A - 150。

③选择烧写到 flash的地址,由于处理器启动时,是从 flash的起始开始执行代码,即 0x00000000。因此需把 U - Boot下载到 flash的 0地址。

这种方法烧写速度慢,此法只在没其他办法的情况下才使用。

方法二:即当板子已有能正常运行的 BOOTLOADER后,进入命令控制台,利用 U - Boot命令烧写。U - Boot的一大突出特点就是支持网络功能,可以通过网络下载数据,比用串口下载要快得多。用这种方法烧写比用 JTAG接口烧写要快很多,因此只要引导程序能正确运行就都用的是这种方法。其烧写过程如下:

```
SMDK2410 # protect off all
SMDK2410 # tftp 30008000 U - Boot. bin
SMDK2410 # erase 1: 0
SMDK2410 # cp. b 30008000 01000000 20000
```

U - Boot隔一段时间后会自动将 flash所有扇区保护起来,而要被保护的扇区是无法被擦除的,因此在烧写之前必须先开启。flash只有先被擦除了才能被重新烧写,U - Boot是放在 flash的 0地址的,也就是第一个扇区,只需擦除这个即可。cp. b是一个内存与 flash之间的一个数据拷贝命令,30008000代表要拷贝的数据在内存中的地址,而 01000000是指要拷贝到 flash的地址,而 20000是指数据量的大小。

(3)启动内核和加载 Ramdisk文件系统。U - Boot所支持的内核格式是 uImage,不是 zImage,如果不对代码进行任何修改,则此时只能用 bootm命令启动经过 mkimage工具处理过的内核映像文件 uImage。也可以修改 /common/cmd_boot. c文件,这样就可以使用 go命令来启动 zImage内核映像文件。具体命令如下:

```
SMDK2410 # tftp 30008000 zImage
SMDK2410 # tftp 31000000 initrd. gz
SMDK2410 # go 30008000
```

这样即可引导内核,并加载文件系统,但这只是人为将内核下载到内存让其运行,一旦断电就随之消失。若是想每次启动时让 U - Boot自动启动内核,则需先把内核映象文件烧写到 flash中,每次启动时再把它从 flash加载到内存去执行,而这一步的实现就需要通过 U - Boot命令设定环境变量 bootcmd,或者在头文件 S3c2410. h中预先设置好 bootcmd的值,这样 U - Boot就会自动启动内核。

5 结束语

实验结果表明,CPU,串口,sdram,Flash,DM9000等主要硬件,它们的配置都正确,因为能顺利把内核和文件系统烧写到板子,并且能成功启动内核,这足以证明 U - Boot的正确性,完成了它的最终目的。本文选择最小硬件系统进行移植,此种系统是最基本的系统,最具有代表性的系统,同时它所有的硬件配置也是 U - Boo引导内核启动所必须涉及到的。当然本文所作的修改只是让 U - Boot实现最基本的功能,也可以通过修改代码实现其他的功能,比如设置发光二极管的显示方式来达到调试的目的。将 U - Boot移植到 ARM板相对简单点,只需修改其硬件配置,而移植到 FPGA就相对较复杂点,因为 FPGA具有更大的灵活性,今后将往这一方面努力,以达到对 U - Boot更深的理解,以致不管移植何种类型的板子都能得心应手。

参 考 文 献

1 郑灵翔. 嵌入式系统设计与应用开发. 北京:北京航空航天大学出版社, 2006.
2 刘峥嵘 . 嵌入式 Linux应用开发详界解 . 北京:机械工业出版社, 2004(7).
3 周立功 . ARM嵌入式 Linux系统构件与驱动开发范例 . 北京:北京航天航空大学出版社, 2006, 1.
4 李善平. Linux操作系统及实验教程. 北京:机械工业出版社, 1999.