

doi:10.3969/j.issn.1007-7162.2014.04.016

基于 S3C2440 的嵌入式系统小型 U-Boot 的研究

吴 伟,周延周

(广东工业大学 自动化学院,广东 广州 510006)

摘要: 针对嵌入式系统中硬件设备和使用环境等条件限制使得系统整体资源量相对比较少的问题,提出基于系统 CPU 为 S3C2440 的一种占用资源非常小的 U-Boot 设计方案.通过初始化本实验平台的硬件设备,设置好实验的软件环境等步骤后,成功启动 Linux2.6.30.4 版本的内核,而该 U-Boot 本身大小仅仅为 2.32 kB.在整个嵌入式系统中占用资源少的 U-Boot 就可以为其他功能让出更多的资源,使得系统功能可以变得更加强大.

关键词: U-Boot; S3C2440;资源;设计方案

中图分类号: TP311.1

文献标志码: A

文章编号: 1007-7162(2014)04-0085-05

Research on the Mini U-Boot of the Embedded System Based on S3C2440

Wu Wei,Zhou Yan-zhou

(School of Automation, Guangdong University of Technology, Guangzhou 510006, China)

Abstract: The embedded system is limited by system resources for its hardware and operating environment. It proposes the design of U-Boot with low system resources. By initializing the experimental platform and setting the software environment, it used the kernel of Linux on version 2.6.30.4. The U-Boot size is only 2.32 kB. Thus, the U-Boot, which uses fewer system resources on the embedded system, leaves more resources for other modules and makes the embedded system more powerful.

Key words: U-Boot;S3C2440;resources;design proposal

嵌入式系统的建立主要包括硬件平台的搭建和软件环境的实现.在软件环境中,又包括 Bootloader、操作系统内核和文件系统的构建以及应用程序开发等几个阶段.其中 Bootloader 的建立是后续几个步骤的基础,它为操作系统内核、文件系统提供下载方式,为操作系统、文件系统的引导、运行提供合适的硬件环境配置信息^[1].U-Boot,全称是 Universal Boot Loader,是一个遵循 GPL 条款的开源通用 Bootloader.在嵌入式系统中,U-Boot 是应用最广泛的一款 Bootloader.它是由德国 DENX 软件工程中心的 Wolfgang Denk 基于 8xxROM 的源码创建的 PPC-BOOT 工程发展而来的,支持 ARM、PowerPC、MIPS、X86 等众多架构的 CPU 和 Linux、VxWorks、NetBSD 等操作系统.U-Boot 是在系统复位后执行的第一段程序,功能和 PC 机的 BIOS 程序相同^[2-3].先初始化硬件设备,然后准备好软件环境等,最后完成它的最终目标:加载并启动操作系统内核.U-Boot 与 CPU 芯片的体系结构、板级外设的芯片配置和使用的操

作系统等因素有关,它的功能的实现非常依赖于硬件.但是在嵌入式系统中,由于硬件平台类型的搭建与选择十分灵活,因此在嵌入式系统领域建立一种通用的 U-Boot 几乎是不可能的^[4].开发时需要开发人员根据不同的开发环境进行不同的程序配置.开发时,可以采用交叉编译开发模式,即程序的编写和编译在装有 Linux 系统的服务器上完成,开发工具链选用 arm-linux-gcc,编译生成的可执行文件通过工具下载到嵌入式目标板上运行^[5].

1 U-Boot 的实现

1.1 硬件资源

处理器采用 ARM9 系列结构,版本是 CPU 核为 ARM920T 的 S3C2440 芯片.该芯片支持 32 位的 ARM 指令集和 16 位的 Thumb 指令集,常用主频 400 MHz.内部带有 4 kB 的 SRAM,并且支持 norflash 启动和 nandflash 启动两种方式.SDRAM 是 32 Bit 数据宽度,64 MB 大小,由两片 16 Bit 的 32 MB 的

HY57V561620 芯片组成. Nandflash 和 Norflash 分别采用 256 MB 大小的 K9F2G08U0B 芯片和 2 MB 大小的 SST39VF1601 芯片.

1.2 总体流程

本文参照 U-Boot 思想的主要流程,并紧密结合实验中用到的实验平台的硬件资源特性,设计的 U-Boot 主要流程分为第一阶段和第二阶段^[6]. 整体流程如图 1 所示,在第一阶段中,完成了 SDRAM、nandflash 等硬件资源相关的初始化工作,使得它们可以正常工作. 并从 norflash 或者 nandflash 中读出 U-Boot 程序重定位到 SDRAM 中;在第二阶段主要完成串口 0 的初始化,然后从 nandflash 中 Linux 操作系统内核读取到其指定的链接地址中并在指定的地址处设置好给内核的启动参数,最后跳转到链接地址启动系统内核. 整个设计方案的文件分别为: start. S, U-Boot. c, Makefile, U-Boot. lds. 其中,第一阶段中的前 4 步在 start. S 中实现,剩余部分和第二阶段在 U-Boot. c 中实现.

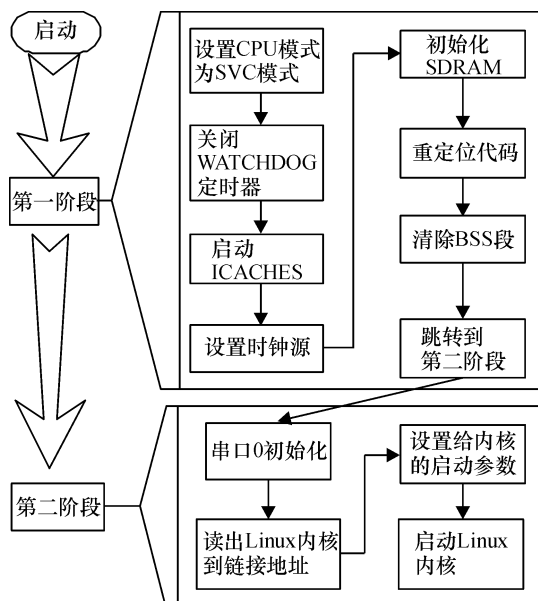


图1 整体流程框图

Fig.1 The overall flow diagram

1.3 U-Boot 第一阶段具体实现

1.3.1 初始化 CPU

如图 1 所示: 首先, S3C2440 的 CPU 核 ARM920T 有 7 种工作模式, 与其他 6 种模式相比, 在 SVC 模式下更有利于初始化硬件. 并且在做好初始化硬件等准备工作跳转到内核之前需要满足的条件之一就是 CPU 出于 SVC 模式下^[7].

然后, 关闭 WATCHDOG 定时器. 在 S3C2440 的 WATCHDOG 定时器有两种作用: 一是用于产生周期性的中断, 二是用于发出复位信号以重新启动出现问题而无法继续运行的系统^[8]. 在要制作的 U-

BOOT 中不需要用到 WATCHDOG 的功能, 可以通过往 WATCHDOG 控制寄存器 WTCN 写入零值来关闭 WATCHDOG 的功能.

第三, 启用 ICACHES. CACHE 是 CPU 通用寄存器和内存之间的一个高速的、容量相对较小的存储器. 基于程序的空间局部性原理和时间局部性原理, 达到较高的命中率从而避免 CPU 每次都一定要与相对慢速的内存交互数据来提高数据的访问速率^[9]. CACHE 可分为指令 CACHE; ICACHES 和数据 CACHE; DCACHES. 系统刚上电或复位时, ICACHE 中的内容是无效的, 并且 ICACHES 的功能是关闭的^[10]. 在程序中开启 ICACHES, 可以在一定程度上提高 U-Boot 的运行速度.

最后, 设置系统时钟源. S3C2440 的时钟源可以有两种方法获得: 一是外接晶振, 然后通过内部电路产生时钟源; 二是直接使用外部提供的时钟源. 该时钟模块给整个芯片提供 3 种时钟序列: FCLK 用于 CPU 核; HCLK 用于 AHB 总线上的设备; PCLK 用于 APB 总线上的设备. 对时钟模块的设置可以先通过时钟分频控制寄存器 CLKDIVN 分配好 FCLK、HCLK、PCLK 的比例, 再通过 MPLL 配置寄存器 MPLLCON 来设置 FCLK 的频率. MPLL 频率的计算公式为

$$M_{pll} = (2 * m * F_{in}) / (p * 2^s),$$

$$m = (MDIV + 8), p = (PDIV + 2), s = SDIV,$$

其中 F_{in} 为晶振频率 12 MHz, m 、 p 、 s 分别为配置参数^[11].

根据硬件资源的特性, 可取参数 $MDIV = 0x5c$, $PDIV = 0x01$, $SDIV = 0x01$, 按照以上公式即可得出 M_{pll} 的频率为 400 MHz.

1.3.2 初始化 SDRAM

U-Boot 的程序存放在实验平台的 nandflash 的开始部分. 当从 nandflash 启动 CPU 时, CPU 会通过内部的硬件自动把 nandflash 中开始的 4 kB 数据复制到称为“Steppingstone”的 4 kB 大小、起始地址为零的片内内部 RAM 中, 然后跳到该零地址开始执行程序^[12]. 如果 U-Boot 程序的大小大于 4 kB, 那在该内部 RAM 中就无法执行, 必须借助于外部的 SDRAM 来运行程序. 在本实验的硬件资源平台中, SDRAM 接在 S3C2440 的存储控制器的 BANK6 中, 起始地址为 0x30000000. 通过对存储控制器的位宽和等待控制寄存器 BWSCON 等共 13 个寄存器的设置即可初始化好 SDRAM, 使其正常工作.

1.3.3 U-Boot 程序的重定位

一般的程序都是存储在 nandflash 或 norflash 中. 在 norflash 中时, 由于 norflash 的特性是可以按地址读取数据, 所以程序可以直接读出并重定位到

SDRAM 中.但在 nandflash 中,由 nandflash 的特性决定了程序是不能直接按地址读出的,只能按页读取^[13].所以必须先初始化 nandflash 让其正常工作后才可以从 nandflash 中读取到正确的数据.当从 nandflash 启动时,需要在程序开始并且小于 4 kB 大小的范围内完成 nandflash 的初始化工作并把后面部分的程序从 nandflash 中重定位到外部 SDRAM 中的过程,再跳转到 SDRAM 中继续运行程序.由于 nandflash 的初始化过程和重定位的过程相对比较复杂,因而采用 C 语言来编写程序,使程序的可读性、可移植性等变得更强大,更有利于实现复杂的功能^[14].在汇编程序中设置好 SP 寄存器后,直接用 bl 跳转指令跳转的相应的 C 语言程序入口处即可实现调用 C 语言函数的功能.

初始化 nandflash 时需要对 nandflash 配置寄存器 NFNCONF 和 nandflash 控制寄存器 NFCONT 进行设置,如下:

```
#define NFCONT ( * (( volatile unsigned long * )0x4E000004) )
#define NFNCONF ( * (( volatile unsigned long * )0x4E000000) )
#define TACLS 0
#define TWRPH0 1
#define TWRPH1 0
NFNCONF = ( TACLS < < 12 ) | ( TWRPH0 < < 8 ) | ( TWRPH1 < < 4 );
NFCONT = ( 1 < < 4 ) | ( 1 < < 1 ) | ( 1 < < 0 );
```

初始化完成后,即可对 nandflash 进行读操作,并可以进行程序的重定位.过程如图 2 所示:首先用 boot 函数判断是从 nandflash 启动还是从 norflash 启动.判断的方法如下:如果从 nandflash 启动,零地址在片内 RAM 中;如果从 norflash 启动,零地址在 norflash 中.片内 RAM 是可以直接读写数据的,而 norflash 可直接读数据,但是在写之前必须先擦除成功才可写入数据.全部从零地址读取数据并保存到 tmp 变量中,再在零地址写入另一个值 tmp + 1,最后再读出零地址的值与变量 tmp 的值相比较.如果不相同即为从 norflash 启动,可以直接读出程序到外部 SDRAM 中;如果相同则为从 nandflash 启动,程序中用 nand_read 函数从 nandflash 中读出 U-Boot 程序并重定位到外部 SDRAM 中,程序如下:

```
#define NOR_FLASH_BOOT 1
#define NAND_FLASH_BOOT 2
int relocated ( unsigned char * src, unsigned char * dest, unsigned int len ) {
    if ( boot() == NOR_FLASH_BOOT ) {
```

```
flash_read( ( unsigned int ) src, dest, len );
    return 0;
}
else if ( boot() == NAND_FLASH_BOOT ) {
    nand_read( ( unsigned int ) src, dest, len );
    return 0;
}
else
    return -1;
}
```

在 flash_read 函数中,可以直接复制数据到目的地址:

```
void flash_read ( unsigned char * src, unsigned char * dest, unsigned int len ) {
    for ( int cyc = 0; cyc < len; cyc ++ ) {
        dest[ cyc ] = src[ cyc ];
    }
}
```

在 nand_read 函数中,分 5 步实现从 nandflash 中读出数据到目的地址中:

- (1) 发出选中 nandflash 芯片命令;
- (2) 发出读芯片命令和存放要读取的数据源的地址;
- (3) 发出读芯片命令并判断 nandflash 芯片的状态是否就绪状态;
- (4) 如果 nandflash 芯片是就绪状态就从源地址读取当前页的数据到目的地址;
- 重复(2) ~ (4)步骤,直到数据全部读取完毕;
- (5) 发出取消 nandflash 芯片片选状态的命令.

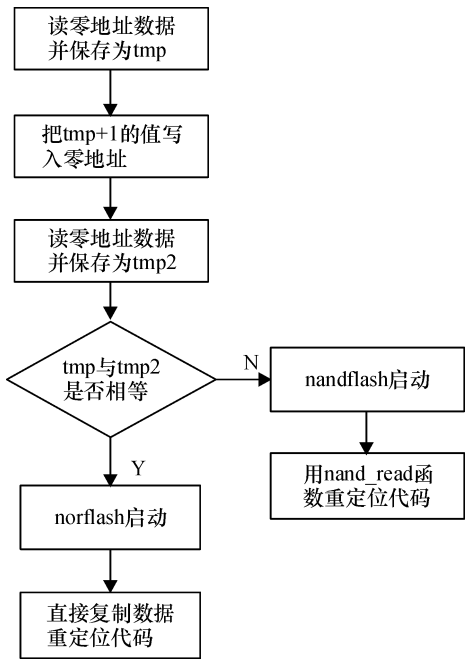


图2 U-Boot 重定位流程图

Fig.2 The flow diagram of U-Boot relocation

1.3.4 清除.bss 段数据

.bss 段是在链接脚本中定义的一段区域.bss: {*(.bss)*(COMMON)},用来存放程序中未初始化的全局变量和静态变量的一块内存区域.在程序编译完成之后,已经初始化的全局变量和静态变量保存在程序的.data 段中,未初始化的全局变量和静态变量保存在.bss 段中..data 段在可执行文件中,有系统从可执行文件中加载运行;.bss 段不在可执行文件中,在程序执行之前由系统初始化.bss 段数据^[15].

最后用把程序计数器 PC 指向 boot 函数,程序跳转到 boot 函数的链接地址处继续执行,完成第二阶段的启动工作.

```
ldr pc, =boot
```

1.4 U-Boot 第二阶段具体实现

1.4.1 初始化串口

硬件平台的 COM0 和 PC 的串口相连,通过串口就可以观察到程序在硬件平台上运行后的结果.并且内核启动时也需要一个初始化好的串口,并通过输出内核的启动信息.初始化硬件平台的串口后,在 PC 上可以通过串口工具 SecureCRT 观察到程序输出的信息.在 boot 函数中,用 uart_init 函数完成串口 0 初始化,初始化需要对与串口 0 相关的控制寄存器 UCON0 等 5 个寄存器进行设置:

```
UCON0 = 0x05;
ULCON0 = 0x03;
UFCON0 = 0x00;
UMCON0 = 0x00;
UBRDIV0 = (50000000 / ((115200 * 16)))-1
```

U-Boot 不依赖于任何函数库,所以要在串口中输出数据的功能必须 U-Boot 自身用函数实现.实现程序如下:

```
#define TXD0READY (1<2)
#define UTRSTAT0( *(volatile unsigned long *)
0x50000010)
void put_out(char *str) {
    for(int i = 0; str[i] != 0; i++) {
        while (! (UTRSTAT0 & TXD0READY));
        UTXH0 = str[i];
    }
}
```

1.4.2 读取 Linux 内核到链接地址

在 nandflash 中,存放内核镜像 zImage 的开始地址是 0x00240000,用 nand_read 函数从该地址开始读出内核到其指定的链接地址 0x30008000 中:

```
put_out ("Begin copy kernel from nand ..... \n \r");
nand_read(0x00240000, (unsigned char *)
0x30008000, 0x200000);
put_out (" copy kernel from nand complete! \n \r");
```

1.4.3 给内核设置启动参数

在 SDRAM 中地址为 0x30000100 处放置给内核启动参数的标记列表,需要设置 struct tag 结构体:

```
static struct tag *params;
put_out ("Begin set boot params..... \n \r");
setup_start_tag();
setup_memory_tags();
setup_commandline_tag("noinitrd theKernel root
=/dev/mtdblock2 init =/linuxrc
console = ttySAC0");
setup_end_tag();
put_out ("Set boot params complete ! \n \r");
```

1.4.4 启动内核

设置函数指针 theKernel 指向内核的运行地址 0x30008000,并跳转到该地址运行即可启动内核:

```
theKernel = (void (*)(int, int, unsigned int))
0x30008000;
put_out ("Skip to kernel ! \n \r");
theKernel(0, 168, 0x30000100);
```

其中,168 为机器 ID,0x30000100 为标记列表的开始地址.

2 实验结果与分析

以上程序文件复制到 Linux 服务器上编译后最终生成 U-Boot. bin 等文件,如图 3 所示,其中 U-Boot. bin 的大小为 2.32 kB.再把 U-Boot. bin 通过工具下载到 nandflash 的起始位置,并从 nandflash 启动实验平台即可在串口工具 SecureCRT 中看到如图 4 的输出信息.从中可以看出,本次制作的 U-Boot 能够成功地引导启动 2.6.30.4 版本的 Linux 系统内核.

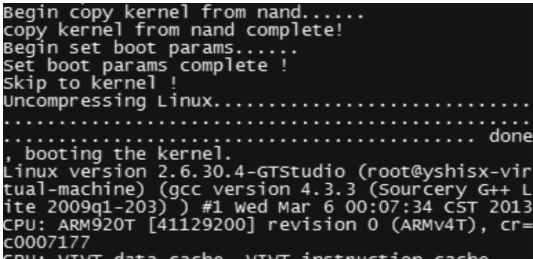


图 3 U-Boot 和内核启动的输出信息

Fig.3 Output information of U-Boot and kernel booting

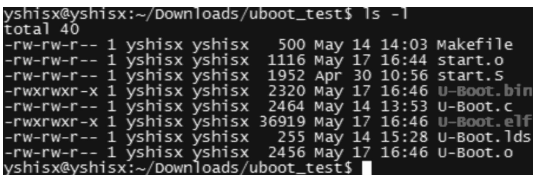


图 4 文件大小

Fig.4 Size of files

U-Boot 的发行版本经过编译生成的 U-Boot. bin 的大小一般至少为 200 kB 以上,之所以这么大主要有两个原因:第一,为了使得能够在各个硬件平台上的通用性最大化,方便移植,U-Boot 用了大量的代码来实现包含各个不同平台不同硬件的特性.这就使得在一个目标平台上运行的程序源代码中包含了大量其他平台不同硬件资源特性的无用代码.第二,在 U-Boot 的源代码中有一部分的代码用于实现调试功能的命令,如比较内存命令 cmp、复制内存命令 cp 等.而本文所论述的 U-Boot 设计方案中,紧密围绕本实验平台的硬件特性,所有代码都是跟本实验平台的硬件特性有关,没有包含本实验平台以外其他任何硬件资源特性的代码.而且,所有代码都围绕直接启动 Linux 内核这个最终目的来展开,去除了调试功能部分的代码,也没有实现其他多余的功能模块.所以,最终生成 U-Boot. bin 的大小就能够变得只有 2.32 kB.

3 结论

本文针对用到的实验平台的硬件资源特性,设计了一个资源占用量仅仅只有 2.32 kB 的 U-Boot,并成功引导 Linux 内核和挂接文件系统.目前,嵌入式系统的应用日益广泛,嵌入式系统中所实现的功能越来越多,这些功能都需要占用系统的资源,但在嵌入式系统中由于硬件条件和使用环境等条件的限制使得其资源量相对比较少.所以本实验设计的非常精悍的 U-Boot,可以为其他的功能让出更多的系统资源,使得系统可以拥有更多的功能,在用途越来越广泛的嵌入式系统应用中有非常大的实用价值.

参考文献:

- [1] 李翔,许迎娣. U-BOOT 的启动参数传递与移植分析[J]. 黑龙江科技信息,2008(34):83.
- [2] 朱博,孙运强. 嵌入式系统的 U-Boot 移植分析[J]. 中国测试技术,2008,34(2):57-60.
Zhu B, Sun Y Q. U-Boot transplantation analyses for embedded system[J]. China Measurement & Testing Technology, 2008,34(2):57-60.
- [3] 刘大伟. U-Boot 在 ARM 平台的移植与应用[J]. 中国传媒大学学报:自然科学版,2008,15(3):70-72,47.
Liu D W. Porting and Application of U-Boot on the ARM Platform[J]. Journal of Communication University of China:Science and Technology, 2008,15(3):70-72,47.
- [4] 何景波,孙运强,姚爱琴,等. U-Boot 在嵌入式系统上的移植分析与实现[J]. 机械工程与自动化,2009(4):50-52.
He J B, Sun Y Q, Yao A Q, et al. Transplantation of U-Boot in Embedded System. [J]. Mechanical Engineering & Automatio, 2009(4):50-52.
- [5] 袁磊,朱怡安,兰晴,等. 嵌入式系统 BootLoader 设计与实现[J]. 计算机测量与控制,2009,17(2):389-391.
Yuan L, Zhu Y A, Lan J, et al. Design and implementation of BootLoader in embedded system[J]. Computer Measurement & Control, 2009,17(2):389-391.
- [6] 师磊. U-Boot 在 S3C2440 上的分析与移植[J]. 计算机系统应用,2010,19(4):204-207.
Shi L. Analysis and transplantation of U-boot on the S3C2440[J]. Computer Systems & Applications, 2010,19(4):204-207.
- [7] Crifan Li. Uboot 中 start. S 源码的指令级的详尽解析[EB/OL]. [2011-05-01/2013-6-10]. http://www.cri-fan.com/files/doc/docbook/uboot_starts_analysis/release/html/uboot_starts_analysis.html.
- [8] 韦东山. 嵌入式 Linux 应用开发完全手册[M]. 北京:人民邮电出版社,2008:164-165.
- [9] 宋宝华. Linux 设备驱动开发详解[M]. 2 版. 北京:人民邮电出版社,2010:236-237.
- [10] 韦东山. 嵌入式 Linux 应用开发完全手册[M]. 北京:人民邮电出版社,2008:110-111.
- [11] 申爽. 基于 S3C2440 的 Uboot 分析与移植[J]. 计算机系统应用,2012,(5):222-225.
Shen S. Uboot analysis and transplantation based on S3C2440[J]. Computer Systems & Applications, 2012(5):222-225.
- [12] 何臻祥,陈波. 基于三星 S3C2440 Boot Loader 的研究[J]. 兵工自动化,2008,27(6):92-94.
He Z X, Chen B. Research of Boot Loader based on Samsung's S3C2440 [J]. Ordnance Industry Automation, 2008,27(6):92-94.
- [13] 冯林琳,耿恒山. 基于 S3C6410 的 Uboot 分析与移植[J]. 计算机与现代化,2013,(1):119-121.
Feng L L, Geng H S. Uboot analysis and transplantation based on S3C6410 [J]. Computer and Modernization, 2013(1):119-121.
- [14] 陈立定,陈登. 基于 NAND Flash 启动的 U-Boot 设计与实现[J]. 微处理机,2010,31(5):107-110.
Chen L D, Chen D. Design and realization of U-Boot booting from NAND flash[J]. Microprocessors, 2010,31(5):107-110.
- [15] 查婧. ARM9 嵌入式系统设计及 Uboot 的移植[D]. 陕西:中国科学院研究生院(西安光学精密机械研究所)信息科学与工程学院,2009.