嵌入式 BootLoader 的设计与实现

龙帆[1]，易波[2]，林能发[3]
（1. 湖南省交通科学研究院有限公司，湖南长沙，410000；2. 湖南大学信息科学与工程学院，湖南长沙，410000；3. 中联重科股份有限公司，湖南长沙，410000）

摘要：Bootloader是嵌入式系统的核心组成部分，它连接起了嵌入式操作系统和硬件架构平台，深刻影响着嵌入式设备的后续软件开发。UBoot是一款通用Bootloader，本文详细分析了u-boot的特点及启动流程，探讨利用Uboot实现支持NAND Flash和YAFFS文件系统的嵌入式Bootloader的方法。

关键词：Bootloader；U-boot；嵌入式系统；S3C2440；嵌入式开发板；硬件

DOI:10.16589/j.cnki.cn11-3571/tn.2020.19.003

0 引言

嵌入式系统作为一种专用计算机系统，它的软件和硬件都可根据需求进行裁剪，因此能满足于对功能、可靠性、性价比、体积、能耗都要求严格场景的应用[1][6]。嵌入式系统软件，可分为四个层次：Bootlader、板级支持包、嵌入式操作系统、用户应用程序。其中 Bootloader 和板级支持包严重依赖于系统硬件，是整个嵌入式系统开发中的难点和关键点。本文针对这个难点做了相应的研究，探讨了嵌入式 BootLoader 的实现方法。

1 嵌入式文件系统和 BootLoader

常用的嵌入式文件系统有 ROMFS、CRAMFS、UC/FS、RFS、JFFS/JFFS2、YAFFS 等。其中 YAFFS 可读写日志文件系统是专为 NANDFlash 设计的，其提供了直接访问文件系统的 API 自带 NAND 驱动，操作速度快，内存占用少，具备错误检测和坏块处理，采用多策略混合的垃圾回收算法，并提供跨平台支持。

Bootloader是嵌入式微处理器上电运行的第一段程序，主要完成硬件核心初始化、内存映射初始化、操作系统运行环境准备等，其位于整个存储区域最前端，如图 1 所示。通常 BootLoader 有两种模式：（1）启动模式：BootLoader 从非易失存储器件上将操作系统镜像加载到随机存储器 RAM 中解压运行；（2）下载模式：BootLoader 通过串口或网络从主机下载文件，此模式一般用于系统开发调试。由于 CPU 架构和板级结构种类庞杂，Bootloader 不可能实现完全的统一。[2][7][10] 但实现一个具有最大通用性，只需修改少量代码就可以在不同架构上运行的 Bootloader 则完全可行，U-Boot 就是这样的一个 Bootloader。

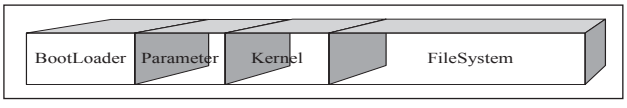

图 1　闪存存储示分配意图

U-Boot 源码中没有对 NAND Flash 启动和 YAFFS 文件系统的支持，本文将探讨移植 U-Boot，并增加对 NAND Flash 和 YAFFS 的支持。

2 Uboot 的特性及启动过程

U-boot 是目前嵌入式平台上具备最广泛通用性和应用最多的开源 Bootloader[3][8]。U-Boot 的启动可分成两个阶段：（1）用汇编实现的 CPU 体系结构初始化。（2）用 C 语言实现的板级初始化、用户交互功能和操作系统加载。其代码结构如图 2 所示[4][9]。

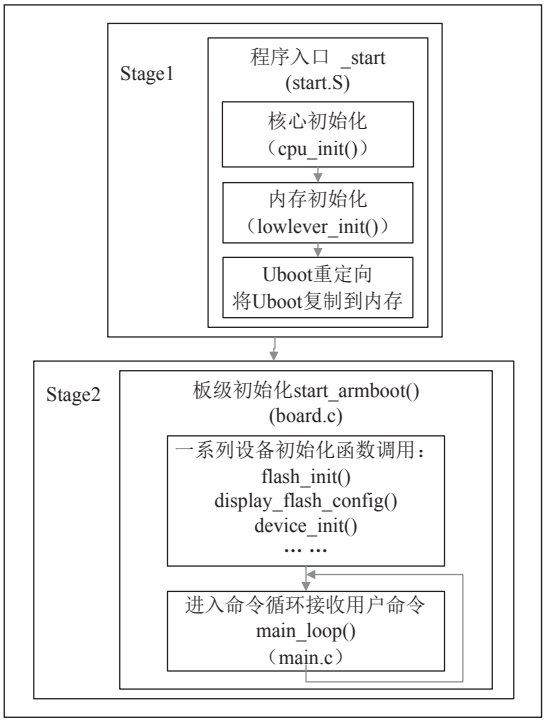

图 2　U-Boot 的执行顺序

3 NAND Flash 的特点和操作

NAND 和 NOR 是当前两种主流的非易失闪存技术。NAND 结构比 NOR 结构具备更高的储存密度，且 NAND 每个块的最大擦写次数能达一百万次，能大大提高产品寿命。但 NAND 在操作上较复杂，在执行其他操作前必须先写入

驱动程序，且 NAND 器件的写入操作也是相当有技巧的一项工作。

NAND 的数据是以 bit 形式保存在 memory cell 中的，一个 memory cell 只能储存一个 bit，每 8 个或 16 个 memory cell 连成 bit line，形成位宽。这些 Line 再组成 Page，然后每 32 个 Page 形成一个 Block，所以一个 Block 的大小是 16k，Block 是 NAND 器件中的最大操作单元，以 K9F1208B0C 为例，其结构如图 3 所示。

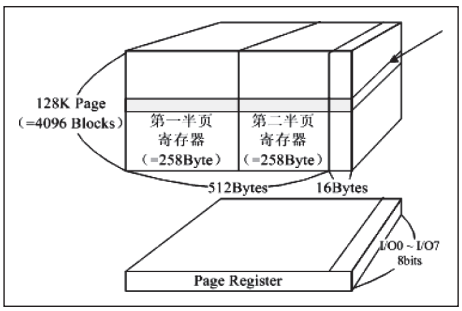

图 3　NAND 器件储存结构

NAND 器件储存单元的地址通过 Column Address 和 Page Address 指定，Column Address 指定了从 Page 上的某个 Byte 开始读取，Page Address 指定开始读取的 Page，只要指定了 Column Address，NAND 控制寄存器可以自动完成对整个 Page 的读取。当地址传递不能在一个周期内完成时，就须分多个周期来完成，就构成了 NAND 器件操作的多周期性，当得到 NAND 地址时我们通过如下分解得到每周期传递的地址：

```
NANDFLASHAD = addr & 0xFF;    // 计算得到列地址
NANDFLASHAD = (addr >> 9) & 0xFF;
NANDFLASHAD = (addr >> 17) & 0xFF;
NANDFLASHAD = (addr >> 25) & 0xFF;
```

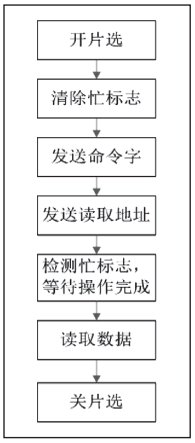

图 4　NAND Flah 读取操作命令队列

NAND 只能进行写 0 操作，在初始化后所有存储单元都为“1”，为可写状态。当某个已经为“0”的位需要复位为“1”时，只能借由清除整个区块来复位“1”的状态。NANDFlash 的地址，数据和命令端口是复用的，因此对 NAND 的操作要通过命令队列方式完成，如图 4 所示。

4 构建嵌入式 BootLoader

■ 4.1 硬件平台特性

目标板硬件资源如表 1 所示。

表1　硬件资源

硬　件	描　述
CPU	S3C2440
储存器	NANDFLASH 64M K9F1208B0C
	SDRAM 64M HY57V561620FTP-H
网卡芯片	DM9000AEP
晶振	外部12MHz

S3C2440 是三星公司生产的基于 ARM920T 内核的 RISC 处理器。通过对 S3C2440 的 NANDFalsh 控制器寄存器发送命令可实现对 NANDFalsh 芯片的读写。其 NAND Falsh 的启动过程如图 5 所示，上电后，NANDFlash 控制器将自动搬移 NANDFalsh 前 4KB 空间的数据到内部 RAM，并将内部 RAM 的起始地址设为 0 地址，CPU 从 0 地址开始执行 [5]。

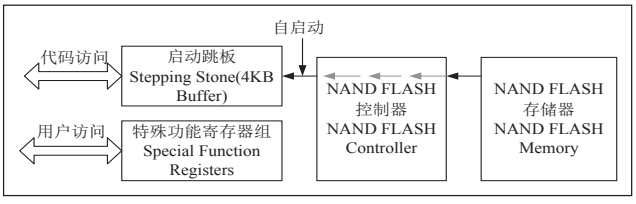

图 5　S3C2440 核心的 NAND Flash 启动

■ 4.2 修改底层驱动

首先配置系统运行时时钟。外部晶振为 12MHz，主频为 405MHz，时钟分频设为 1：4：8。修改 /cpu/arm920t/start.S，设置寄存器 CAMDIVN=0，CLKDIVN=5，并增加一个对时钟的缓冲操作：

```
mrc p15, 0, r1, c1, c0, 0
orr  r1, r1, #0xc0000000
mcr p15, 0, r1, c1, c0, 0
```

S3C2440 的核心只能从 NANDFlash 自动搬移小部分代码到内部 RAM（Steppingstone 区），因此 S3C2440 的核心配置以及将代码从 Flash 复制到 SDRAM 的工作必须在 Bootloader 的前 4KB 代码内完成。UBoot 并不支持 NANDFlash 启动，所以还须在底层增加 NANDFlash 驱动。在 /cpu/arm920t/start.S 中配置 NANDFlash 控制寄存器 NANDCONF 和 NANDFCONT，增加初始化操作：

```
ldr  r2，=((7<<12)|(7<<8)|(7<<4)|(0<<0))
```

```
str  r2，[#NANDCONF]          // 页大小和容量设置
ldr  r2，=( (1<<4)|(0<<1)|(1<<0) )
str  r2，[#NANDFCONT]    // 将 NAND 控制寄存器配置为使能
```

随后跳入 NANDFlash 操作函数 nand_read_ll 中完成 BootLoader 自身的拷贝。在 nand_read_ll 函数中关注下列代码：

```
{
int n，m；
NAND_ FLASH_ENABLE；          // 操作前打开片选
//#define NAND_CHIP_ENABLE （NANDFCONT &= ~(1<<1))
for(n= start_address; n < (start_address + length);) {
NAND_CLEAR_BUSY；          // 将忙标志清除掉
NANDCMD = 0；
NANDADDR = n & 0xFF；
// 根据芯片 Datasheet，地址写入需要 4 个时钟周期，第一周期为列地址，后面三个周期为页地址
NANDADDR = (n >> 9) & 0xFF；
NANDADDR = (n >> 17) & 0xFF；
NANDADDR = (n >> 25) & 0xFF；
NAND_DET_ BUSY；       // 判断操作是否完成
// #define NAND_DET_ BUSY
{ while(! (NANDSTAT&(1<<2)) )
for(m=0; m < NAND_SECTOR_SIZE; m++, n++){
* data = (NANDDATA & 0xFF)；
data ++；
}
NAND_FLASH_DISABLE；
return 0；
}
.....
}
```

■ 4.3 修改链接文件和配置

为了将 NAND Flash 的初始化和 U-Boot 重定向放到前 4K 中，需要修改链接文件 /BOARD/ myboard/u-boot.lds 中代码段的分配，将 CPU 初始化，NAND 初始化和读取放到最初的 Steppingstone 空间，将下列代码添加到链接文件：

```
BOARD/myboard /llevel_init.o(.text)
BOARD/myboard /nandflash_rd.o(.text)
```

在 U-Boot 板级配置文件 include/configs/ 中添加配置头文件。这个文件包括开发板的 CPU、时钟、RAM、Flash、网络、用户交互命令及其他相关配置。

```
#define     CONF_CMD_NAND          1
            //NAND 操作命令支持
#define     CONF_MTD_NAND_VERIFY_WRITE    1
            //NAND 写校验使能
#define     CONF_ NAND_DEVICE_MAX          1
            //NAND 器件数
#define     NAND_S3C2440           1
            //S3C2440 芯片支持
#define     CONF_DM9000    1
// 添加 NANDFlash 寄存器的映射地址
#define     NAND_CTL_BASE          0x4E000000
#define     NANDCONF               0
#define     NANDFCONT              4
#define     NANDCMD                8
#define     NANDADDR               12
#define     NANDDATA               16
#define     NANDSTAT               32
#define     NANDECC                44
```

■ 4.4 增加对 YAFFS 文件系统的支持

在 UBoot 中是从 do_nand() 函数进入 NAND Flash 操作的，在配置文件中加入 CONFIG_COMMANDS 和 CFG_CMD_NAND 宏定义后方可使用此函数。当进入 do_nand() 函数后会调用 /include/nand.h 中的 nand_read() 函数，然后通过访问本 NAND 芯片对应的 nand_info_t 结构完成对 nand_chip 结构中 cmdfunc 指针的调用，通过这个指针指向的函数完成向 NAND Flash 发送命令。要使 UBoot 支持 YAFFS 就要修改 do_nand() 函数，使其能够认识 YAFFS 文件。在 do_nand() 函数中增加下列代码：

```
if(f!= null && !strcmp(f,”.yaffs”)){
    if(yafffs_read){
        printf(“READ YAFFS IS NOT PROVIDE!”); }
    else{
        nand.writeoob=1；
        ret= nand_write_skip_bad(nand，off，&length，(unsigned char *)addrress)；
        nand.writeoob=0；
        }
```

在 UBoot 的交互命令中加入读写 YAFFS 的命令描述：

```
“nand write.yaffs - address off|partition size，” write
‘size’ starting at ‘off’ \n”
“to/from yaffs image in ‘addr’.\n”
```

修改顶层 Makefile 文件：

myboard_config:unconfig@$(makeconfig)$@(@:_config=)arm arm920t myboard mull s3c24x0

最后将工程中文件的（CONFIG_S3C2410）宏更改为（CONFIG_S3C2440）宏，再运行 make 命令编译生成 u-boot.bin 镜像，将镜像下载到目标板的 NAND Flash 中。将目标板重新启动后我们就可以利用 UBOOT 的 nand write.yaffs 命令完成将 YAFFS 文件写入 Flash。

5 结语

BootLoader 作为嵌入式系统的核心环节，对其设计与开发的研究就一直没有停止过。优秀的 Bootloader 不但要运行稳定、高效，还要具备强大的命令交互功能，这就对开发人员提出了极高的要求，他们不但要熟悉硬件体系架构、操作系统原理，还要具备软硬件协同开发能力，并且整个开发过程极其复杂，因此在成熟 Bootloader 上进行源码移植，使其满足新需求，能有效降低对开发人员的要求，缩短研发周期并提高产品稳定性，是工程开发的首选方式。本文设计的 BootLoader 提供 NAND 启动和 YAFFS 支持的同时保留了强大的用户交互能力，并且已在企业合作项目中取得了成功。

参考文献

* [1] 萨日那 . 嵌入式系统及模型化开发技术研究 [J]. 现代电子技术，2016,3:146-150.
* [2] 温暖 , 杨维明 , 彭菊红 , 等 . 基于 MCU 的嵌入式系统的 Bootloader 设计 [J]. 微电子学与计算机 ,2018(3):79-82.
* [3] The Denx U-Boot and Linux Guide[EB/OL]. Http://www.denx.de/twiki/bin/view/DULG/Manual，2020-08-12.
* [4] 税静 , 吴长水 . 发动机控制器在线升级系统的设计与实现 [J]. 农业装备与车辆工程 ,2020,58(6):44-48.
* [5] S3C2440A 32-BIT MICROCONTROLLER USER`S MANUAL. Revision 1[C]. 2016.
* [6] 廖勇，杨霞 . 嵌入式操作系统 [M]. 北京：高等教育出版社，2017：20-38.
* [7] 吴磊 , 皮智 , 袁宗胜 . 一种基于 S3C6410 的 BootLoader 的设计与实现 [J]. 计算机应用与软件 ,2016,33(9):238-244.
* [8] K C Wang.Embedded and Real-Time Operating Systems[M],Beijing:China Machine Press,2020:56-88.
* [9] 滕军 . 基于 ARM 的 BootLoader 系统加载设计 [J]. 信息与电脑 ,2019,31(17):82-83.
* [10] 龙帆 . 易波基于 ZigBee 的智能家居网关系统的研究与实现 [D]. 湖南 : 湖南大学 ,2010.

（上接第 28 页）

4 系统主要规格参数

（1）主控板电路是采用 STM32F407 嵌入式芯片作为控制核心，芯片采用工作频率为 168 MHz 的 Cortex ™ -M4 内核。

（2）步进电机型号为：J4218HB3401，控制采用 6 细分电流 1A，3200 脉冲步进电机旋转一圈；电机控制电源采用 12V。

（3）红外感应采用 940nm 波长红外 LED。

（4）步进电机驱动硬件系统采用 12V2A 的电源。

5 结束语

眼球自动对焦柔性调节方面，随时对相机传送的图像进行处理，确定待测瞳孔的坐标位置； PC 机和 SLO 相机交换瞳孔坐标位置，确定坐标定位 XYZ 步进电机移动参数，确保待测瞳孔的坐标和 GUI 界面图像显示区域坐标重合。在对焦过程中，采用梯形算法控制步进电机的运行，使得对焦更精准和平稳。

参考文献

* [1] Rodieck R W. The vertebrate retina: principles of structure and function[M]. Series of books in biology. Oxford: Freeman. 1973: 1044.
* [2] Hogg R E, Chakravarthy U. Visual function and dysfunction in early and late age-related maculopathy[J]. Progress in retinal and eye research,2006,25(3)：252
* [3] 陈蔚霖 , 常军 , 赵雪惠 , 金辉 . 广域眼底相机光学系统的设计与仿真分析 [J]. 中国光学 ,2020,13(04):814-821.
* [4] 张秀国 . 单片机 C 语言程序设计教程与实训 [M]. 北京大学出版社 ,2008：78-94.

（上接第 83 页）

* [5] 孔凡才，周良权 . 电子技术综合应用创新实训教程 [M]. 北京：高等教育出版社，2008.
* [6] 李鹏 . 数字电子技术及应用项目教程 [M]. 北京：电子工业出版社，2016.