Linux NAND Flash Guide

Revision: 0.3

Date: Mar 2009

北京君正集成电路有限公司

Ingenic Semiconductor Co. Ltd

Linux NAND Flash Guide

Release history

Date	Revision	Change
2009.04.17	0.3	修改了 APUS 板的 NAND 烧录参数
2009.03.24	0.2	添加在 PAVO 和 APUS 板烧录各种目标文件时烧录工具默认配置表
2009.02.17	0.1	Created

Disclaimer

This documentation is provided for use with Ingenic products. No license to Ingenic property rights is granted. Ingenic assumes no liability, provides no warranty either expressed or implied relating to the usage, or intellectual property right infringement except as provided for by Ingenic Terms and Conditions of Sale.

Ingenic products are not designed for and should not be used in any medical or life sustaining or supporting equipment.

All information in this document should be treated as preliminary. Ingenic may make changes to this document without notice. Anyone relying on this documentation should contact Ingenic for the current documentation and errata.

北京君正集成电路有限公司

北京市海淀区东北旺西路 8 号中关村软件园一号楼
信息中心 A 座 108 室，100193
Tel: 86-10-82826661
Fax: 86-10-82825845
Http: //www.ingenic.cn

内容

1 Linux 烧录方案

烧录方法：

1. 使用 usb boot 工具烧录，参考文档 USB_Boot_Tool_Manual_1.4_CN.pdf。(JZ4740 和 JZ4750 支持，JZ4730 不支持)
2. 使用 JDI 工具烧录，参考文档 JDI-Manual_EN_1.4.pdf,(烧录文件大小不能超过 9MB,且不支持 NAND multi-plane 操作)。
3. 在 Linux 运行环境下烧录（只能烧 uImage 和 NAND 文件系统，不能烧 u-boot）

烧录对象：

1. u-boot
2. uImage
3. 各种 NAND 文件系统映像:
   A. yaffs2 映像
   B. ubi 映像
   C. vfat 文件系统
   D. 其它文件系统

本文将举例介绍在君正开发板上使用 usb boot 工具和 JDI 工具以及在开发板上运行的 linux 下烧录各种目标文件的方法。关于两种工具的系统介绍请参考君正网站的 USB_Boot_Manual 和 JDI-Manual。

1.1 MTD 分区

在使用 NAND Flash 之前，首先要定义好 NAND Flash 的分区。MTD 分区的定义在 linux-2.6.24.3/drivers/mtd/nand/下的 jz4730_nand.c、jz4740_nand.c 或 jz4750_nand.c 文件里。用户需要根据自己的系统分别定义 NAND Flash 物理块的分区信息。

君正在标准分区信息的基础上增加了三个成员：use_planes，cpu_mode 和 mtdblock_jz_invalid。
其中 use_planes 用来决定该分区是否使用 multi-plane 读写操作，其值为 0 表示不使用 multi-plane，为 1 表示使用。如果所选用的 NAND Flash 不支持 multi-plane 操作，那么 use_planes 项的值无意义。目前，jz4730 暂不支持 multi-plane 操作，因此该项设置也不起作用。如果所选用的 NAND 不支持 multi-plane， 或所有分区都不使用 multi-plane，那么最好在配置 linux 编译选项时，使 CONFIG_MTD_NAND_MULTI_PLANE=n。
cpu_mode 用来指定该分区使用 cpu 模式还是 dma 模式。
mtdblock_jz_invalid 用来指示该分区是否使用君正提供的文件系统转换层 mtdblock_jz.c,比如 YAFFS2 分区不需要使用它，将 mtdblock_jz_invalid 设为 1，那么在 mount YAFFS2 分区时，就不会做一些不

必要的扫描,不会分配一些不必要的内存;VFAT 分区需要使用mtdblock_jz.c,要将mtdblock_jz_invalid 设为 0，否则使用该分区时会报异常。如果设 CONFIG_ALLOCATE_MTDBLOCK_JZ_EARLY=y，并且该分区使用 dma 模式，那么在 NAND 驱动初始化时，会给所有使用了 mtdblock_jz.c 的分区提前分配一个物理上连续的 NAND block 的缓存，这特别适用于会被当作 U 盘使用的 VFAT 分区，因为如果在加载 U 盘时才申请 1 个 block 的连续缓存，可能因为系统内存碎片太多而无法申请到。但这种提前申请的缓存无法释放。如果该分区使用 cpu 模式，就没有必要提前申请缓存，因为此时不需要连续的物理内存，一般都能申请到。

下面我们以在 jz4750 下使用 2GB 的 NAND Flash 为例来向读者介绍如何定义自己的分区：

参数：

Page size：4096 Bytes
Block size: 512 KB
Pages per block: 128
Row address cycles: 3
Total size: 2GB
Total blocks: 4096

NAND 分区表信息：

表 1　　2GB NAND Flash 的分区表示例

分区	起始物理块	结束物理块	分区大小	分区描述	是否使用 multi-plane	是否使用 cpu 方式	是否使用 mtdblock-jz
MTD BLOCK 0	0	7	4MB	MTD 分区 0 (bootloader)	不使用	不使用	不使用
MTD BLOCK 1	8	15	4MB	MTD 分区 1 (kernel)	不使用	不使用	不使用
MTD BLOCK 2	16	1023	504MB	MTD 分区 2 (rootfs)	不使用	不使用	不使用
MTD BLOCK 3	1024	2047	512MB	MTD 分区 3 (data)	使用	不使用	不使用
MTD BLOCK 4	2048	4095	1GB	MTD 分区 4 (vfat)	使用	不使用	使用

MTD 分区定义于文件 linux-2.6.24.3/drivers/mtd/nand/jz4750_nand.c 中，如下所示：

```
static struct mtd_partition partition_info[] = {
    {name:"NAND BOOT partition",
     offset:0 * 0x100000,
     size:4 * 0x100000,
     cpu_mode: 0,
     use_planes: 0,
     mtdblock_jz_invalid: 1},
    {name:"NAND KERNEL partition",
     offset:4 * 0x100000,
```

```
    size:4 * 0x100000,
    cpu_mode: 0,
    use_planes: 0,
    mtdblock_jz_invalid: 1},
   {name:"NAND ROOTFS partition",
    offset:8 * 0x100000,
    size:504 * 0x100000,
    cpu_mode: 0,
    use_planes: 0,
    mtdblock_jz_invalid: 1},
   {name:"NAND DATA partition",
    offset:512 * 0x100000,
    size:512 * 0x100000,
    cpu_mode: 0,
    use_planes: 1,
    mtdblock_jz_invalid: 1},
   {name:"NAND VFAT partition",
    offset:1024 * 0x100000,
    size:1024 * 0x100000,
    cpu_mode: 0,
    use_planes: 1,
    mtdblock_jz_invalid: 0},
  };
```

1.2 烧录 u-boot

配置和编译的具体过程请参考linux development guide 的 “3.2配置和编译U-Boot”。
烧录时必须注意的是，产品板或开发板对应的u-boot的配置文件的NAND相关参数要和烧录工具使用的NAND参数一致。而且烧录u-boot时只能用单plane。
下面烧录以JZ4750的APUS开发板使用K9GAG08U0M(2GB NAND)为例（具体分区信息见表1），u-boot-1.1.6/include/configs/apus.h是它的配置文件，其中NAND相关的默认参数设为：

```
#define CFG_NAND_BW8                    1
#define CFG_NAND_PAGE_SIZE              4096
#define CFG_NAND_BLOCK_SIZE             (512 << 10)
#define CFG_NAND_BADBLOCK_PAGE          127
#define CFG_NAND_BCH_BIT                8
#define CFG_NAND_ECC_POS                24
```

设置好后执行

```
$ make apus_nand_config
$ make
```

编译生成u-boot-nand.bin, 要把它烧录到NAND的MTD0分区。U-boot只支持usb boot和JDI工具烧录，不能在linux下烧录。

1.2.1 使用 usb boot 工具烧录 u-boot

APUS板使用USBBoot_APUS.cfg的配置，确保其中的NAND配置和上面编译u-boot时的NAND参数一致：

```
[NAND]
BUSWIDTH        8
ROWCYCLES        3
PAGESIZE        4096
PAGEPERBLOCK    128
FORCEERASE      0
OOBSIZE         128
ECCPOS          24
BADBLACKPOS     0
BADBLACKPAGE    127
PLANENUM        1
BCHBIT          8
```

执行命令：

```
USBBoot :> nprog 0 u-boot-nand.bin 0 0 -n
```

1.2.2 使用 JDI 工具烧录 u-boot

APUS板使用jz4750_apus_boot.cfg的配置，使NAND配置和上面编译u-boot时的NAND参数一致：

```
[NAND]
BUSWIDTH        8
ROWCYCLES       3
PAGESIZE        4096
PAGEPERBLOCK    128
OOB              0
OOBECC           0
ECCPOS          24
BCHBIT          8
BADBLOCKPOS      0
BADBLOCKPAGE    127
```

```
FORCEERASE      0
```

执行命令：

```
JDI> nerase 0 8
JDI> nprog 0 u-boot-nand.bin
```

1.3 烧录 uImage

同样以JZ4750的APUS开发板为例,把uImage烧录到mtd1分区（偏移量为4MB）。首先编译kernel:

```
$ make apus_defconfig
$ make xconfig
$ make uImage
```

使用usb boot和JDI烧录kernel与烧录u-boot类似，因为uImage是通过u-boot把uImage从NAND读到SDRAM中，所以也要保证板子对应的u-boot的配置文件的NAND相关参数和烧录工具使用的NAND参数一致。

1.3.1 使用 usb boot 工具烧录 uImage

APUS板使用USBBoot_APUS.cfg的配置，执行命令把uImage烧录到mtd1分区（偏移量4MB=4096*1024，按页烧录）：

```
USBBoot :> nprog 1024 uImage 0 0 -n
```

Usb boot 工具在执行nprog时自动进行了擦除工作，不需要另外进行擦除。

1.3.2 使用 JDI 工具烧录 uImage

先擦除mtd1分区(偏移量4MB = 512KB * 8，按块擦除)，然后烧录：

```
JDI> nerase 8 8
JDI> nprog 1024 uImage
```

1.3.3 在 linux 下烧录 uImage

具体的烧录方法和芯片类型相关。

1.3.3.1 JZ4730 在 linux 下烧录 uImage

对于 JZ4730, 因为 u-boot 读 uImage 时没有进行 ECC 校验，所以在开发板上执行：

```
# flash_eraseall /dev/mtd1
# nandwrite_mlc -a -p /dev/mtd1 uImage
```

1.3.3.2 JZ4740 在 linux 下烧录 uImage

对于 JZ4740, u-boot 读 uImage 时进行了 Reed-solomn ECC 校验, 为了使 linux 和 u-boot 对 NAND 的编解码方案一致，在 linux 烧录 uImage 时, 要确保满足下面两个条件：

1．编译 uImage 时对 CONFIG_MTD_BADBLOCK_FLAG_PAGE 的配置与编译 u-boot 时对 CFG_NAND_BADBLOCK_PAGE 的设置一致。本例中都为 127。
2.NAND 驱动中 chip->ecc.layout->eccpos[0]的值和 u-boot 的 NAND 参数 CFG_NAND_ECC_POS 的值相等。对于 jz4740，等于 28。

对于 JZ4740 的 PAVO 板使用 2KB 页 NAND 的情况，u-boot 的配置文件是 u-boot-1.1.6/include/configs/pavo.h，其中 NAND 相关的参数要设为：

```
#define CFG_NAND_BW8              1
#define CFG_NAND_PAGE_SIZE        2048
#define CFG_NAND_BLOCK_SIZE       (256 << 10)
#define CFG_NAND_BADBLOCK_PAGE    127
#define CFG_NAND_ECC_POS          28
```

在 linux/drviers/mtd/nand/nand_base.c 中，对于 PAVO 板上 2KB 页大小的 NAND,注意到以下代码：

```
static struct nand_ecclayout nand_oob_64 = {
  .eccbytes = 36,
  .eccpos = {
     28, 29, 30, 31,
     32, 33, 34, 35, 36, 37, 38, 39,
     40, 41, 42, 43, 44, 45, 46, 47,
     48, 49, 50, 51, 52, 53, 54, 55,
     56, 57, 58, 59, 60, 61, 62, 63},
  .oobfree = {
     {.offset = 2,
      . length = 26}}
}
```

```
chip->ecc.layout = &nand_oob_64;
```

可知 chip->ecc.layout->eccpos[0]=28，与 CFG_NAND_ECC_POS=28 一致。

确保满足以上条件后，就可在开发板上执行：

```
# flash_eraseall /dev/mtd1
# nandwrite_mlc -a -p /dev/mtd1 uImage
```

1.3.3.3 JZ4750 在 linux 下烧录 uImage

对于 JZ4750， u-boot 除了要满足上面 JZ4740 要满足的两个条件，另外，BCH 算法的设置也要一样：在编译 u-boot 时，对于 4KB 页的 NAND，要在 include/configs/apus.h 中

```
#define CFG_NAND_BADBLOCK_PAGE      127   /* 要和内核设置一致 */
#define CFG_NAND_ECC_POS            24     /* 要和内核设置一致 */
#define CFG_NAND_BCH_BIT             8     /* 要和内核设置一致 */
#define CFG_NAND_BCH_WITH_OOB         /* 只要 define 即可，不需要设定值 */
```

另外，uImage 也需要用君正修改过的 mkyaffs2image（源码在 linux-2.6/fs/yaffs2/utils/下）工具进行处理，在每个 NAND pagesize 内容之后，添加一个 oobsize 的区域（其中的内容可以任意），对于 4KB 页 NAND:

```
$ mkyaffs2image 2 uImage uImage.oob
```

然后在开发板上执行：

```
# flash_eraseall /dev/mtd1
# nandwrite_mlc -a -o /dev/mtd1 uImage.oob
```

注意 nandwrite_mlc 即可用于 MLC NAND 也可用于 SLC NAND。在 linux-2.6/drivers/mtd/mtd-utils/下执行 make,即可获得 nandwrite_mlc 和 flash_eraseall。

此时，在编译 u-boot 时定义了 CFG_NAND_BCH_WITH_OOB，那么用 usb boot 和 JDI 工具烧录 uImage 时，也需要烧录 uImage.oob, 否则 u-boot 读 uImage 会报 ECC 不可纠的错误提示。使用 usb boot 烧录 uImage.oob 时，相对于烧录 uImage, nprog 的参数由 -n 改为-o。

```
USBBoot :> nprog 1024 uImage.oob 0 0 -o
```

使用JDI烧录uImage.oob时，命令还是和烧录uImage一样，只是JDI配置文件中的OOB要设为1，然后

```
JDI> nerase 8 8
JDI> nprog 1024 uImage.oob
```

1.4 烧录各种 NAND 文件系统

Linux 使用 MTD（Memory Technology Devices）驱动来管理 NAND Flash 设备。YAFFS2 文件系统自己实现了 NFTL（NAND Flash Translation Layer）功能， 所以在 MTD NAND Flash 设备上可以直接构建 YAFFS2 文件系统。

君正还修改了 mtdblock 块设备驱动以支持 NAND Flash 的文件系统。修改后的 mtdblock 块设备驱动增加了逻辑块地址到物理块地址的映射、坏块管理、损耗均衡等功能。 这样，用户就可以在 NAND Flash 的 mtdblock 块设备上构建 EXT2、EXT3 和 FAT 等文件系统。

Linux 2.6 内核引入了 UBI (Unsorted Block Images)来对 Flash 进行管理。在 UBI 的基础上可以直接创建 UBIFS 文件系统。另外为了能在 UBI 上直接创建 EXT2 和 FAT 等文件系统，在 UBI 之上添加了 UBI BLOCK 设备层驱动（ubiblk.c 和 bdev.c）。

关于 NAND 文件系统的详细介绍请参考 linux development guide 的第 8 部分“NAND Flash 文件系统”。下面介绍各种 NAND 文件系统的烧录方法。

1.4.1 烧录 YAFFS2 映像

下面烧录以 JZ4750 的 APUS 开发板使用 K9GAG08U0M(2GB NAND)为例(具体分区信息见前面表 1)，把 YAFFS2 映像烧录到 mtd2 分区，本例中该分区不使用 multi-plane 特性。

1.4.1.1 创建 YAFFS2 映像

在 PC 上使用 mkyaffs2image 工具：

```
usage: mkyaffs2image layout# source image_file [convert]
    layout#     NAND OOB layout:
                    0 - nand_oob_raw, no used,
                    1 - nand_oob_64, for 2KB pagesize,
                    2 - nand_oob_128, for 2KB pagesize using multiple planes or 4KB
                        pagesize,
                    3 - nand_oob_256, for 4KB pagesize using multiple planes
    source          the directory tree or file to be converted
    image_file  the output file to hold the image
    'convert'   make a big-endian img on a little-endian machine. BROKEN !
```

烧录 image_file 要注意烧录工具 ECCPOS 的设置。目前最新驱动中 JZ4740 的 ECCPOS 设置为 28，JZ4750 的 ECCPOS 设置为 24，此值等于文件 linux-2.6/drivers/mtd/nand/nand_base.c 里的函数 nand_scan_tail()中 chip->ecc.layout->eccpos[0]的值。

比如使用 4KB pagesize NAND 并不使用 multiple planes 时，用下面命令生成 YAFFS2 映像,:

```
$ mkyaffs2image 2 /rootfs/ rootfs.yaffs2
```

mkyaffs2image 源码位于 linux/fs/yaffs2/utils/ 下。君正也对其做了修改以支持大于 2KB page size 和 multi-plane 的 MLC NAND。如果 mkyaffs2image 工具要处理的/rootfs/ 大小是 pagesize*N,，那么生成的 yaffs2 image 文件大小是 (pagesize + oobsize) * N,其中 oob 区域存放了每一个 page 的 yaffs2 文件系统信息(16 bytes)，以及这些信息的 ECC 校验码，ECC 校验可以采用 hamming （只能纠 1 bit 错误，用于 SLC NAND）或 reed-solomn ECC（能纠 2-10 bits 错误，用于 MLC NAND)。但 jz4750 不需要这种 ECC 校验，因为 yaffs2 文件系统调用 mtd->read_oob()读取一个 page 的 NAND 内容时， MTD 层中，BCH 算法已经对 oob 区域进行了校验，不需要再在 yaffs2 层做校验。使用 BCH 时， eccpos 固定设为 24，并且，为了便于 usb boot 烧录校验方便，mkyaffs2image 工具会使生成的映像文件中 oob 区域内 eccpos 以后的数据都设为 0xff。

所以，mkyaffs2image 工具会根据 linux 编译选项不同生成不同的 image。比如当内核 CONFIG_YAFFS_ECC_RS=y， 也即配置 yaffs2 使用 reed-solomn ECC 算法对 oob 区域进行校验时，mkyaffs2image 工具生成的 image 的 oob 区域就要存放相应的 reed-solomn ECC。当 CONFIG_YAFFS_ECC_HAMMING=y, image 的 oob 区域存放 Hamming ECC。当配置 CONFIG_MTD_HW_BCH_ECC=y, image 的 oob 区域内 eccpos（=24）以后的数据都设为 0xff。

让 mkyaffs2image 工具获得 linux 编译选项的方法是，每当编译 linux 改变 NAND ECC 种类或 yaffs2 中对 oob 使用哪种 ECC 的方法时，就进入 fs/yaffs2/utils/ 目录，重新 make 生成新的 mkyaffs2image 工具。

在使用 jz4750 的 BCH 算法时，为了使 u-boot 能正常读取在 linux 下烧录的 uImage，需要对 uImage 进行处理变为 uImage.oob, 即在每个 NAND pagesize 内容之后，添加一个 oobsize 的区域（其中的内容可以任意)。君正对 mkyaffs2image 工具进行了修改使其能对文件做这样的处理，对于 4KB 页 NAND:

```
$ mkyaffs2image 2 uImage uImage.oob
```

1.4.1.2 使用 usb boot 工具烧录 YAFFS2 映像

注意 YAFFS2 映像包含了 OOB 信息，但是 OOB 中没有 ECC 信息。设置 usb boot 工具的 NAND 相关配置信息，注意 mtd2 分区未使用 multi-plane 特性,所以要将 PLANENUM 设为 1：

```
[NAND]
BUSWIDTH        8
ROWCYCLES       3
PAGESIZE        4096
PAGEPERBLOCK    128
FORCEERASE      0
OOBSIZE         128
ECCPOS          24
BCHBIT          8
```

```
BADBLACKPOS      0
BADBLACKPAGE     127
PLANENUM         1
```

由分区信息可知，mtd2 的起始物理块是 16，即 16*128=2048 页。

```
USBBoot :> nprog 2048 rootfs.yaffs2 0 0 -o
```

另外，如果本例中 mtd2 分区使用了 mult-plane 特性，那么创建 YAFFS2 映像的命令为：

```
# mkyaffs2image 3 /rootfs/ rootfs.yaffs2
```

烧录时 usb boot 配置信息的 PLANENUM 设为 2，命令仍是

```
USBBoot :> nprog 2048 rootfs.yaffs2 0 0 -o
```

1.4.1.3 使用 JDI 工具烧录 YAFFS2 映像

本例中，JDI的NAND配置：

```
[NAND]
BUSWIDTH       8
ROWCYCLES      3
PAGESIZE       4096
PAGEPERBLOCK   128
OOB            1
OOBECC         0
ECCPOS         24
BCHBIT         8
BADBLOCKPOS    0
BADBLOCKPAGE   127
FORCEERASE     0
```

执行命令：

```
JDI> nerase 16 1008
JDI> nprog 2048 rootfs.yaffs2
```

JDI 目前只支持 NAND 单 plane 操作，而且烧录文件大小不能超过 9MB，就是说如果 mtd2 使用 multi-plane 特性或 rootfs.yaffs2 大于 9MB，就不能用 JDI 烧录。

1.4.1.4 在 linux 下烧录 YAFFS2 映像

用 nandwrite_mlc 命令把 rootfs.yaffs2 写到 mtd2 分区上，在目标板上运行下面命令(不论 mtd2 是否使用 multi-plane，命令都是一样，只是 rootfs.yaffs2 文件不同)：

```
# flash_eraseall /dev/mtd2
# nandwrite_mlc  -a  -o /dev/mtd2 rootfs.yaffs2
```

上面的 flash_eraseall 和 nandwrite_mlc 工具的源码位于 linux/drivers/mtd/mtd-utils/ 下，君正对源码进行了一些修改，以支持大于 2GB 的 MLC NAND。nandwrite_mlc 也可用于 SLC NAND。

1.4.2 烧录 UBI 映像

下面烧录以 JZ4740 的 PAVO 开发板使用 K9G8G08U0M(1GB NAND)为例（具体分区信息见表 1），把 UBI 映像烧录到 mtd3 分区，本例中该分区使用 multi-plane 特性。由于 UBI 映像一般会大于 9MB，所以就不介绍 JDI 工具的烧录方法了。

1.4.2.1 创建 UBI 映像

使用 mkfs.ubifs 工具可以将包括多种格式的卷生成映像文件，将映像文件写到 nand 上后，可直接使用包含的卷及文件。注意：编译和使用 mkfs.ubifs 需要安装 lzo 库（lzo-2.02.tar.gz），请到下述地址下载资源，具体操作请参考资源里的 mkfs.ubifs/README 文档。

下载地址：ftp://ftp.ingenic.cn/3sw/01linux/07utils/linux-nand-utils.tar.gz

假设制作这样一个映像文件，名字为 ubi.img，里面包括两个卷 ubifs 和 vfat，分别为 ubifs 格式和 vfat 格式。具体步骤如下：

首先用 mkfs.ubifs 工具在 PC 上制作 ubifs 卷映像 ubifs.img0：

```
$ export LD_LIBRARY_PATH=/opt/lzo/lib:$LD_LIBRARY_PATH
$ ./mkfs.ubifs -r /nfsroot/root26/ -m 2048 -e 258048 -c 813 -o ubifs.img0
```

-c 指定这个卷的最大有效容量为多少个逻辑块
-e 逻辑块容量（字节为单位）
可以开发板上用 ubinfo 工具得知这些数据。

接着在 Linux 系统制作 vfat 卷映像 vfat.img0， 此处映像大小为 60M：

```
$ dd if=/dev/zero of=vfat.img0 bs=1M count=60  //在PC上操作
#losetup /dev/loop0 vfat.img0          //在PAVO板上操作
#mkfs.vfat /dev/loop0
#mount -t vfat /dev/loop0 /mnt/udisk
#cp /mnt/mmc/* /mnt/udisk/       //拷贝需要包含在映像内的文件
#umount  /mnt/udisk
```

```
#losetup -d /dev/loop0
losetup: : No such device or address
```

用 ubirefimg: 将映像文件 ubifs.img0 和 vfat.img0 格式化成带 LEB 逻辑块号映像文件 ubifs.img 和 vfat.img：

```
$./ubirefimg ubifs.img0 ubifs.img
$./ubirefimg vfat.img0 vfat.img
```

然后用 ubinize 通过将 ubifs.img 和 vfat.img 生成一个映像 ubi.img。准备一个 cfg 配置文件，内容如下：

```
# cat ubinize.cfg
[ubifs]
mode=ubi
image=ubifs.img
vol_id=0
vol_size=200MiB
vol_type=dynamic
vol_name=ubifs
vol_alignment=1
vol_flag=autoresize

[vfat]
mode=ubi
image=vfat.img
vol_id=1
vol_size=298MiB
vol_type=dynamic
vol_name=vfat
vol_alignment=1
vol_flag=autoresize
```

然后运行：

```
# ./ubinize -o ubi.img ubinize.cfg -p 262144 -m 2048
```

至此，映像文件 ubi.img 已经制作完成。

1.4.2.2 使用 usb boot 工具烧录 UBI 映像

设置 usb boot 工具的 NAND 相关配置信息，注意 mtd3 使用了 multi-plane，所以要将 PLANENUM 设为 2：

```
[NAND]
```

```
BUSWIDTH        8
ROWCYCLES       3
PAGESIZE        4096
PAGEPERBLOCK    128
FORCEERASE      0
OOBSIZE         128
ECCPOS          24
BCHBIT          8
BADBLACKPOS     0
BADBLACKPAGE    127
PLANENUM        2
```

MTD3起始于2048块，每块128页，2048*128=262144，执行命令：

```
USBBoot :> nprog 262144 ubi.img  0 0 -n
```

1.4.2.3 在 linux 下烧录 UBI 映像

用 nandwrite_mlc 工具 将 ubi.img 写到 nand flash 的 mtd3 上，由于 UBI 映像不包含 oob，所以不需要 -o 选项。

```
# flash_eraseall /dev/mtd3
# nandwrite_mlc -a -q /dev/mtd3 ubi.img
```

加载 ubi 驱动后即可使用 ubi 设备，加载 ubi 驱动：

```
# modprobe ubi mtd=3
# modprobe ubifs
# modporbe ubiblk
# mount -t ubifs ubi0:ubifs /mnt/ubifs/
# mount -t vfat /dev/ubiblock1 /mnt/ubiblock1/
# ls /mnt/ ubiblock1/
```

你会发现预先包含进 vfat 卷映像的文件已经存在。

1.4.3 烧录 VFAT

目前没有专门的vfat映像制作工具，可以先把vfat数据在linux环境下写到指定分区，然后把数据从该分区完整读出来，生成母片文件rootfs.vfat，该文件包含OOB,而且OOB中包含了ECC。对于2KB页大小的NAND，该文件的结构是2KB数据和64字节OOB交替，对于4K页大小的Nand Flash，该文件的结构是4KB数

据和128字节包含了ECC信息的OOB交替；对于JZ4750，请将BCHBIT设置为4。然后使用usb boot工具烧录该母片文件。

目前最新驱动中ECCPOS设置为28，此值根据具体需要会有所变动。烧录时ECCPOS的设置请参考文件：

linux-2.6.24.3/drivers/mtd/nand/nand_base.c。

修改配置文件之后，请务必再执行Boot命令，来更新配置信息。通过以下命令烧录到mtd3分区，假设mtd3起始于2048块，每块128页，2048*128=262144，执行命令：

```
USBBoot :> nprog 262144 rootfs.vfat 0 0 -e
```

1.4.4 烧录其它文件系统

其他文件系统的烧录也要两个过程：制作镜像和烧录。

由于文件系统的特点不同，制作镜像的过程可能不一样。有的可以通过专门的工具，有的需要通过母片的方式。无论哪种方法，最后都生成的镜像文件都必须归类到三种文件类型，第一种是无OOB（如UBI映像），第二种是有OOB无ECC（如yaffs2映像），第三种是有OOB有ECC（如VFAT母片）。

前两种类型可以用三种方式烧录，即usb boot、JDI或在linux下烧录，只是JDI不能烧录大于9MB的文件，且不支持NAND multi-plane操作；最后一种类型只能用usb boot工具烧录。

2 Linux 升级参考方案

NAND 分区划分：

分区 1：NAND SPL
分区 2：Linux kernel 1 + ramdisk(包含升级程序)
分区 3：Linux kernel 2
分区 4：root 文件系统
分区 5：applications
分区 6：FAT 分区(U 盘)

正常流程：

上电 -> NAND SPL -> Linux kernel 2 -> root -> applications

升级流程：

上电 -> NAND SPL -> Linux kernel 1 -> ramdisk -> 升级程序

升级步骤如下：

1、按照正常流程启动系统，通过 usb 由主机传送升级文件到 U 盘分区。
2、按照升级流程启动，运行升级程序，升级程序从 U 盘分区读取升级文件进行系统升级。

注：

1、分区 1 和分区 2 永久保留，不需要升级
2、升级程序可以完成分区 3、分区 4 和分区 5 的升级
3、分区 6 的升级通过 USB 由主机直接拷贝的方式完成

3 附录

3.1 在 PAVO 和 APUS 板烧录各种目标文件时烧录工具的默认配置表

在君正 Jz4740 的 PAVO 开发板和 Jz4750 的 APUS 开发板上使用 K9G8G08U0B(2KB 页大小)或 K9GAG08U0M(4KB 页大小)的 NAND Flash 烧录各种目标文件时烧录工具的默认配置表如下：

其中，参数 PLANENUM 和 OOBSIZE 只在 usb boot 工具配置文件中存在；JDI 只支持单 plane 操作，oobsize 根据页大小自动算出。

参数 OOB 和 OOBECC 只在 JDI 工具中存在；在 usb boot 工具中，被烧录文件是否包含 OOB，以及 OOB 中是否包含 ECC 是由命令行参数指定（包含 OOB 使用参数-o，包含 OOB 和 ECC 使用参数-e，具体请参考文档 USB_Boot_Tool_Manual_1.4_CN.pdf）。

表中 uImage.oob 是为了在使用 jz4750 的 BCH 算法时，使 u-boot 能正常读取在 linux 下烧录的 uImage 而用 mkyaffs2image 工具特殊处理过的映像文件，具体的介绍请参见本文档 1.3.3.3 “JZ4750 在 linux 下烧录 uImage”.

Note：上表中 ECCPOS 均设置为统一固定值，这需要最新的烧录工具配合，请到网站上下载最新烧录工具，否则可能会烧录失败。

	PAVO 板								APUS 板									
	2KB 页 NAND				4KB 页 NAND				2KB 页 NAND					4KB 页 NAND				
	uboot	uImage	yaffs2	vfat	uboot	uImage	yaffs2	vfat	uboot	uImage	uImage.oob	yaffs2	vfat	uboot	uImage	uImage.oob	yaffs2	vfat
BUSWIDTH	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8
ROWCYCLES	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
PAGESIZE	2048	2048	2048	2048	4096	4096	4096	4096	2048	2048	2048	2048	2048	4096	4096	4096	4096	4096
PAGEPERBLOCK	128	128	128	128	128	128	128	128	128	128	128	128	128	128	128	128	128	128
FORCEERASE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ECCPOS	28	28	28	28	28	28	28	28	24	24	24	24	24	24	24	24	24	24
BADBLACKPOS	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
BADBLACKPAGE	127	127	127	127	127	127	127	127	127	127	127	127	127	127	127	127	127	127
BCHBIT	–	–	–	–	–	–	–	–	4	4	4	4	4	8	8	8	8	8
PLANENUM	1	1	1	2	1	1	1	2	1	1	1	1	2	1	1	1	1	2
OOBSIZE	64	64	64	64	128	128	128	128	64	64	64	64	64	128	128	128	128	128
OOB	0	0	1	1	0	0	1	1	0	0	1	1	1	0	0	1	1	1
OOBECC	0	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0	0	1

3.2 开发板 NAND FLASH 常用参数配置示例

- **K9F1G08U0M**

```
#define CFG_NAND_BW8            1
#define CFG_NAND_PAGE_SIZE      2048
#define CFG_NAND_ROW_CYCLE      2
#define CFG_NAND_BLOCK_SIZE     (256 << 10)
#define CFG_NAND_BADBLOCK_PAGE  0
```

- **K9G8G08U0M:**

```
#define CFG_NAND_BW8            1
#define CFG_NAND_PAGE_SIZE      2048
#define CFG_NAND_ROW_CYCLE      3
#define CFG_NAND_BLOCK_SIZE     (256 << 10)
#define CFG_NAND_BADBLOCK_PAGE  127
```

- **K9GAG08U0M:**

```
#define CFG_NAND_BW8            1
#define CFG_NAND_PAGE_SIZE      4096
#define CFG_NAND_ROW_CYCLE      3
#define CFG_NAND_BLOCK_SIZE     (512 << 10)
#define CFG_NAND_BADBLOCK_PAGE  127
```

- **K9LBG08U0M**

```
#define CFG_NAND_BW8            1
#define CFG_NAND_PAGE_SIZE      4096
#define CFG_NAND_ROW_CYCLE      3
#define CFG_NAND_BLOCK_SIZE     (512 << 10)
#define CFG_NAND_BADBLOCK_PAGE  127
```