Linux NAND Flash Guide

Revision: 0.4

Date: Aug 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
2009.08.25	0.4	Deleted the content of JDI
2009.04.17	0.3	Modified the parameters of nand of the apus board
2009.03.24	0.2	Added default configuration table when burning all kinds of files on the pavo and apus board
2009.02.17	0.1	Created

Disclaimer

Ingenic Semiconductor Inc.

Room 108, Information Center Block A
Zhongguancun Software Park
8 Dongbeiwang west Road, Haidian District
Beijing China, 100193

Tel: 86-10-82826661
Fax: 86-10-86865845
Http: //www.ingenic.cn

内容

1 Linux burning plan

Burning Method：

1. Use usb boot tool to burn, reference documentation USB_Boot_Tool_Manual_1.4_CN.pdf. (support JZ4740, JZ4750, and JZ4755, not support JZ4730)
2. Burn it in the Linux operating environment (only burn uImage and the NAND file system, does not burn u-boot)

Burning object：

1. u-boot
2. uImage
3. A variety of NAND file system image:

A. yaffs2 image
B. ubi image
C. vfat file system
D. Other File System

This article describes the method which is to burn a variety of file via usb boot tool and running linux on the developing a board. Please refer to USB_Boot_Manual on Ingenic's website.

1.1 MTD partitions

Prior to the use of NAND Flash, the first is to define NAND partition. MTD partition was defined in linux-2.6.24.3/drivers/mtd/nand/ jz4730_nand.c, jz4740_nand.c or jz4750_nand.c file. Users can define their own partition information of physical block NAND Flash.

Ingenic added three members based on the standard partition information: use_planes, cpu_mode and mtdblock_jz_invalid.

The use_planes which is used to decide whether to use multi-plane partition read and write operations or not, value is 0 means not to use multi-plane, that the use if value is 1. At present, jz4730 does not support multi-plane operation, so that the settings do not work. If the selected NAND does not support the multi-plane, or all of the partitions do not use multi-plane, it is best to configure the linux compile option, so that CONFIG_MTD_NAND_MULTI_PLANE = n.

The cpu_mode is used to specify the partition using the cpu mode or dma mode.

The mtdblock_jz_invalid is used to indicate whether to use the mtdblock_jz.c, such as YAFFS2 partition does not need to use it, so mtdblock_jz_invalid should be set to 1, then mount

YAFFS2 partition, it will not do something to scan, not be allocated a number of unnecessary memory; VFAT partition need to use mtdblock_jz.c, set mtdblock_jz_invalid to 0, otherwise this partition will report exception. If you set CONFIG_ALLOCATE_MTDBLOCK_JZ_EARLY = y, and that the partition will use dma mode, then a physically contiguous cache will be given when all partitions use mtdblock_jz.c in advance, which is particularly applicable to VFAT partition of U disk, because if loaded a U disk when the application for a continuous block cache, probably because too much system memory fragmentation can not be applied for. But this can not be released ahead of the application's cache. If the partition uses cpu mode, there is no need to apply for the cache in advance, because that does not require contiguous physical memory.

Here we use 2GB NAND Flash as an example with jz4750 to introduce how to define your own partition:

Parameters:

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

NAND partition table information:

Table 1 2GB NAND Flash partition table

partition	The beginning of physical block	The ending of physical block	The size of partition	Partition description	Whether to use multi-plane or not	Whether to use cpu mode or not	Whether to use mtdblock-jz or not
MTD BLOCK 0	0	7	4MB	MTD partiton 0 (bootloader)	Not use	Not use	Not use
MTD BLOCK 1	8	15	4MB	MTD partition 1 (kernel)	Not use	Not use	Not use
MTD BLOCK 2	16	1023	504MB	MTD partition 2 (rootfs)	not use	Not use	Not use
MTD BLOCK 3	1024	2047	512MB	MTD partition 3 (data)	use	Not use	Not use
MTD BLOCK 4	2048	4095	1GB	MTD partition 4 (vfat)	use	Not use	use

MTD partition was defined in linux-2.6.24.3/drivers/mtd/nand/jz4750_nand.c:

```
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 Burning u-boot

Please refer to the "3.2 configure and build U-Boot" of the linux development guide about the specific process of compiling and configuring.

Note that the configuration file of developing board must be same with NAND parameters of u-boot and burn tools. And only use single plane when burning u-boot.

The following is a example about the JZ4750 development board(APUS board) which uses the

K9GAG08U0M (2GB NAND) (specific partition information in Table 1).
u-boot-1.1.6/include/configs/apus.h is its configuration file, the default parameter are:

```
#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
```

Then implementation:

```
$ make apus_nand_config
$ make
```

u-boot-nand.bin will be come up. You should burn it to MTD partition 0 of NAND.

1.2.1 Using usb boot tool to burn u-boot

The NAND configuration of USBBoot_APUS.cfg which is APUS board's configuration should keep the same with u-boot's.

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

Then:

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

1.3 Burning uImage

Also take JZ4755(APUS board) as a example, burn uImage to mtd1 partition (the offset is 4MB). First, compile kernel:

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

Keep consistent NAND parameters, because uImage is burnt via u-boot from NAND to SDRAM.

1.3.1 Using usb boot tool to burn uImage

APUS board uses USBBoot_APUS.cfg, you should copy USBBoot_APUS.cfg to USBBoot.cfg, then implement command to burn uImage to mtd1 partition (the offset is 4MB=4096*1024, burn by page)

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

Usb boot tool will automatically erase the data when you execute the command nprog..

1.3.2 Burning uImage under the linux

The specific methods is related to chip's type.

1.3.2.1 Burning uImage under the linux for JZ4740

For JZ4740, there needs ECC checksum when u-boot read uImage, in order to make the same between codec of linux and u-boot, make sure that the following two conditions are met:

1. The value of CONFIG_MTD_BADBLOCK_FLAG_PAGE should be kept the same when compiling u-boot and uImage. In this case the value is 127.
2. In the NAND driver, the value of chip-> ecc.layout-> eccpos [0] equal to the value of CFG_NAND_ECC_POS of NAND parameters of u-boot. For jz4740, the value is 28.

For 2KB pagesize NAND of JZ4740 (PAVO board), u-boot configuration file is u-boot-1.1.6/include/configs/pavo.h，the NAND parameter are:

```
#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
```

Note code in the linux/drviers/mtd/nand/nand_base.c:

```
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;
```

From above code, you can know the chip->ecc.layout->eccpos[0]=28, this value is the same with CFG_NAND_ECC_POS=28.

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

1.3.2.2 Burning uImage under the Linux for JZ4750

For JZ4750, u-boot not only should meet the above two conditions of JZ4740, but also the same BCH algorithm settings:
For 4KB pagesize NAND when compiling u-boot, the nand parameters are in the include/configs/apus.h.

```
#define CFG_NAND_BADBLOCK_PAGE      127   /* the same with kernel
settings */
#define CFG_NAND_ECC_POS            24    /* the same with kernel
settings */
#define CFG_NAND_BCH_BIT             8  /* the same with kernel
settings */
#define CFG_NAND_BCH_WITH_OOB         /* it is ok to have only define ,
not have value*/
```

In addition, uImage also needs to use Ingenic's mkyaffs2image (source code is in the linux-2.6/fs/yaffs2/utils/under), add a oobsize area after each NAND pagesize (where the content can be arbitrary), for 4KB page NAND:

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

Then execute the following commands on the developing board:

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

nandwrite_mlc can be used for MLC NAND and SLC NAND. You can use command make to get nandwrite_mlc and flash_eraseall in the linux-2.6/drivers/mtd/mtd-utils/.

At this point, if u-boot defines CFG_NAND_BCH_WITH_OOB, uImage.oob is also needed to burn when burning uImage with the usb boot tool, or else a error information will be shown about ECC. When using the usb boot to burn uImage.oob, nprog parameter -o instead of -o.

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

1.4 Burning a variety of NAND file systems

Linux uses the MTD (Memory Technology Devices) driver to manage the NAND Flash device. Because the JFFS2 file system achieved NFTL (NAND Flash Translation Layer) function, so MTD NAND Flash devices can be directly build these two file systems.

Ingenic also modified mtdblock block device driver to support NAND Flash file system. The revised mtdblock block device driver increase the logical block addresses to physical block address mapping, bad block management, and loss of balance and other functions. In this way, users can build EXT2, EXT3 and FAT file systems on mtdblock block device of NAND Flash and so on.

Linux 2.6 kernel introduced the UBI (Unsorted Block Images) to manage Flash. Create UBIFS file system on the basis of the UBI. Also in order to be able to directly create EXT2, and FAT file system on the UBI, added UBI BLOCK device layer driver (ubiblk.c and bdev.c).

Refer to the eighth part of linux development guide “NAND Flash file system” about NAND file system.

1.4.1 Burning YAFFS2 image

The following is a example to use the K9GAG08U0M (2GB NAND) for JZ4750 (APUS board), burn the YAFFS2 image to mtd2 partition, in this case the partition does not use multi-plane properties.

1.4.1.1 Creating YAFFS2 image

Use mkyaffs2image on PC, the source locates in linux/fs/yaffs2/utils/.

```
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 !
```

Burning image_file should pay attention to ECCPOS setting of burn tool. In the latest driver of jz4740, the ECCPOS is set to 28, JZ4750's ECCPOS is set to 24, this value is equal to the chip-> ecc.layout-> eccpos [0] of the function nand_scan_tail () in the file which is linux-2.6/drivers/mtd/nand/nand_base.c.

Such as the 4KB pagesize NAND which does not use multiple planes, use the following command to generates YAFFS2 image:

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

The source of mkyaffs2image is located in linux/fs/yaffs2/utils/. Ingenic also made changes to support more than 2KB pagesize and the MLC NAND of multi-plane.

The tool mkyaffs2image will generate different images, which bases on a different linux compile options. For example when the kernel option CONFIG_YAFFS_ECC_RS = y, the oob area is stored in the corresponding reed-solomn ECC. When CONFIG_YAFFS_ECC_HAMMING = y, image of the oob area will store in the Hamming ECC.

You can get a new mkyaffs2image tool after re-making in the directory fs/yaffs2/utils/.

If using the BCH algorithm of a new mkyaffs2image tool, in order to make the u-boot read uImge rightly, a deal is needed to change uImage to uImage.oob, namely, after the contents of each of NAND pagesize, add a oobsize area (where the content can be arbitrary). For 4KB pagesize NAND:

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

1.4.1.2 Burning YAFFS2 using usb boot tool

Note that YAFFS2 image contains OOB information, but there is no ECC information in the OOB. Set NAND-related configuration information of usb boot tool, pay attention to no used multi-plane properties of mtd2 partition, so you should set PLANENUM to 1:

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

From the partition information we can see, the beginning of the physical block of mtd2 is 16, that is 16 * 128 = 2048.

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

In addition, if mtd2 partition uses the mult-plane properties, then the command to create YAFFS2 image as follows:

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

usb boot configuration information’s PLANENUM is 2 when burning, the command is as follows:

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

1.4.1.3 Burning YAFFS2 image under the linux

Write rootfs.yaffs2 to mtd2 partition with the command nandwrite_mlc (no matter that mtd2 use multi-plane or not, the command is the same, but different rootfs.yaffs2):

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

The tools flash_eraseall and nandwrite_mlc source are located in the linux/drivers/mtd/mtd-utils/.

1.4.2 Burning UBI image

The following is a example of JZ4740 (PAVO development board) which uses the K9G8G08U0M (1GB NAND), burn UBI image to mtd3 partition, in this case the partition uses the multi-plane properties.

1.4.2.1 Creating UBI image

NOTE: The compilation and use of mkfs.ubifs need to install the lzo library (lzo-2.02.tar.gz), then go to the following website to download resources, please refer to README document inside the mkfs.ubifs

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

Suppose that producing image file, name of ubi.img, which consists of two volumes ubifs and vfat, ubifs format and vfat format respectively. Concrete steps are as follows:

First, make ubifs.img0 using mkfs.ubifs tool on the PC:

```
$ 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 specifies the number of logical blocks of the maximum effective capacity of volume
-e logical block size (bytes)
You can get these data using ubinfo tool.

Next, make vfat.img0 under the linux, the image size is 60M:

```
$ dd if=/dev/zero of=vfat.img0 bs=1M count=60  //operat on the pc
# losetup /dev/loop0 vfat.img0         //operat on the pavo board
# mkfs.vfat /dev/loop0
# mount -t vfat /dev/loop0 /mnt/udisk
# cp /mnt/mmc/* /mnt/udisk/  //copy file which should be contained in the
                             image
#umount  /mnt/udisk
#losetup -d /dev/loop0
losetup: : No such device or address
```

With ubirefimg: format ubifs.img0 and vfat.img0 into ubifs.img and vfat.img with the LEB logical block numbers:

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

Then use ubinize to produce ubifs.img and vfat.img to ubi.img. Prepare a configuration file, as follows:

```
# 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
```

Then run:

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

Now image file ubi.img has been completed.

1.4.2.2 Using the usb boot tool to burn UBI image

Set NAND-related configuration information of usb boot tool, pay attention to mtd3 using multi-plane, so PLANENUM should be set 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 began at the 2048th block, each block had 128 pages, 2048 * 128 = 262144, execute the command:

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

1.4.2.3 Burning the UBI image under the linux

Write ubi.img to mtd2 of nand flash with nandwrite_mlc tool, because of no oob of UBI image, there is no option –o:

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

After loading ubi driver, ubi device can be used:

```
# 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/
```

You will find the image file which contains vfat has exited.

1.4.3 Burning VFAT

Now there is no specific tool to make vfat image, you can first write vfat data int to specific partition in the linux environment, and then read out the data from the partition, generate a master file rootfs.vfat which contains OOB, and the OOB contains the ECC. For 2KB pagesize

NAND, the file structure is 2KB of data with 64 byte OOB, for 4K page Nand Flash, the file structure is 4KB of data and 128 bytes which contains OOB about ECC information; for JZ4750, please set BCHBIT to 4. And then use the usb boot tool to burn the master file.
ECCPOS, in the latest driver, is set to 28; this value will vary according to specific needs. Settings when burning ECCPOS please refer to documents:
linux-2.6.24.3/drivers/mtd/nand/nand_base.c。

After modifying the configuration file, make sure to re-boot to update the configuration information. Use the following command to burn rootfs.vfat to mtd3 partition, assume mtd3 began in the 2048th block, each block has 128 pages, 2048 * 128 = 262144, and execute the command:

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

1.4.4 Burning other file systems

There are two processes to burn other file system: production of image and burn.

Because of the different characteristics of file system, the process of making image may be different. No matter using which methods to make image, image files must be classified into three kinds of file types, the first is no OOB (such as the UBI image), the second is OOB without ECC (such as the yaffs2 image), and the third OOB is an ECC (such as VFAT master). The first two types can be burned in two ways, usb boot or burning under the linux; the last type can only use usb boot tool to burn.

2 Reference plan about updating the Linux

NAND partition division:

Partition 1：NAND SPL
Partition 2：Linux kernel 1 + ramdisk (contains update program)
Partition 3：Linux kernel 2
Partition 4：Root file system
Partition 5：Applications
Partition 6：FAT partition (U disk)

Normal process:

Power up -> NAND SPL -> Linux kernel 2 -> root -> applications

Upgrade Process:

Power up -> NAND SPL -> Linux kernel 1 -> ramdisk -> Upgrade Procedure

Upgrade steps are as follows:

1, Start system according to normal proccess, sends the update file to U disk partition through the usb from the host.
2, Start according to normal update process, run the upgrade program; the upgrade program read the upgrade file from the U disk partition to upgrade system.

Note:
1, Division 1 and Division 2 should be keep, do not need to upgrade
2, the upgrade program can complete the update of Division 3, Division 4 and Division 5
3, the update of Division 6 can directly copy from the host via USB

Appendix

The default configuration table when burning a variety of target files on the PAVO and APUS board.

The configuration table is as follows:

Parameters OOBISIZE and PALNENUM only exist in usb boot tool's configuration file. Whether ECC exits in OOB which is decided by parameters (including OOB using the parameter -o, including OOB and ECC using parameters -e, please refer to documentation USB_Boot_Tool_Manual_1.4_CN.pdf).

In the table, uImage.oob is used to read uImage rightly when using jz4750's BCH algorithm, please see 1.3.3.3 of this document "JZ4750 under the linux burning uImage" to know more.

Note: the value of ECCPOS should be keep same, which depends on the latest burning tool, go to the website to download the latest Burning tools, or else burn will be failure.

	PAVO board								APUS board									
	2KB pagesize NAND				4KB pagesize NAND				2KB pagesize NAND					4KB pagesize 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

Configuration parameters of NAND FLASH

- **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
```

- **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  0
#define CFG_NAND_BCH_BIT        8
#define CFG_NAND_ECC_POS        24
#define CFG_NAND_IS_SHARE       1
```