BeagleBoardNAND

From eLinux.org

This page is about using (booting/running from) NAND (http://en.wikipedia.org/wiki/Flash_Memory#NAND_flash) memory on BeagleBoard. Parts of this page are inspired by Steve's flash procedure (http://www.sakoman.net/omap3/flash%20procedure.txt) document.

Contents

Hardware

OMAP3530 has 256MB NAND flash in PoP (PoP: Package-On-Package implementation for Memory Stacking) configuration.

- BeagleBoard has 256MB of Micron NAND
- EVM (http://focus.ti.com/docs/toolsw/folders/print/tmdxevm3503.html) 128MB of Samsung OneNAND or 128MB of Micron NAND
- Zoom MDK (http://www.logicpd.com/products/devkit/ti/zoom_mobile_development_kit) also uses the Micron NAND

Software

The following software parts can be stored and booted/run from NAND:

- X-Loader
- U-Boot (+ environment/configuration data)
- Linux kernel
- Linux file system

The memory partitioning for this as used on BeagleBoard:

```
0x00000000-0x00080000 : "X-Loader"
0x00080000-0x00260000 : "U-Boot"
0x00260000-0x00280000 : "U-Boot Env"
0x00280000-0x00680000 : "Kernel"
0x00680000-0x10000000 : "File System"
```

To be able to write something to (empty) NAND, you first need to boot from an other source, e.g. MMC/SD card boot. Besides the files you need for MMC/SD card boot (MLO & U-Boot), put the files you want to flash into first FAT partition of MMC/SD card, too. Then you can read them from there and write them to NAND.

Note: If something goes wrong writing the initial X-Loader, your board might not boot any more without pressing the user button. See BeagleBoard recovery article how to fix this, then.

X-Loader

Build (http://code.google.com/p/beagleboard/wiki/BeagleSoftCompile) or download binary (http://code.google.com/p/beagleboard/downloads/list) (x-load.bin.ift_for_NAND) X-Loader. Put it at first (boot) FAT partition of MMC/SD card and boot from card. Then start boot from card and use the following to write X-Loader to NAND:

```
...40T.......

Texas Instruments X-Loader 1.41
Starting on with MMC
Reading boot sector

147424 Bytes Read from MMC
Starting OS Bootloader from MMC...

U-Boot 1.3.3-00411-g76fe13c-dirty (Jul 12 2008 - 17:12:05)

OMAP3530-GP rev 2, CPU-OPP2 L3-165MHz
OMAP3 Beagle Board + LPDDR/NAND
DRAM:  128 MB
NAND:  256 MiB
In:    serial
Out:   serial
Err:   serial
Hit any key to stop autoboot:  0

OMAP3 beagleboard.org # mmcinit
OMAP3 beagleboard.org # fatload mmc 0:1 80000000 x-load.bin.ift_for_NAND
reading x-load.bin.ift_for_NAND

9808 bytes read
OMAP3 beagleboard.org # nand unlock
device 0 whole chip
nand_unlock: start: 00000000, length: 268435456!
NAND flash successfully unlocked
OMAP3 beagleboard.org # nandecc hw
OMAP3 beagleboard.org # nand erase 0 80000

NAND erase: device 0 offset 0x0, size 0x80000
Erasing at 0x60000 -- 100% complete.
OK
OMAP3 beagleboard.org # nand write 80000000 0 80000

NAND write: device 0 offset 0x0, size 0x80000
524288 bytes written: OK
OMAP3 beagleboard.org #
```

Note: The command nandecc hw is essential here! X-Loader is started by OMAP3 boot rom. This uses HW ECC while reading the NAND, so while writing, we have to use OMAP3 HW ECC, too. If you don't use HW ECC boot rom can't boot from NAND any more.

U-Boot

Build (http://elinux.org/BeagleBoard#U-Boot) or download binary (http://code.google.com/p/beagleboard/downloads/list) (flash-uboot.bin) U-Boot. Put it at first (boot) FAT partition of MMC/SD card and boot from card. Then start boot from card and use the following to write U-Boot to NAND:

```
OMAP3 beagleboard.org # mmcinit
OMAP3 beagleboard.org # fatload mmc 0:1 80000000 u-boot.bin
reading u-boot.bin

147424 bytes read
OMAP3 beagleboard.org # nand unlock
device 0 whole chip
nand_unlock: start: 00000000, length: 268435456!
NAND flash successfully unlocked
OMAP3 beagleboard.org # nandecc sw
OMAP3 beagleboard.org # nand erase 80000 160000

NAND erase: device 0 offset 0x80000, size 0x160000
Erasing at 0x1c0000 -- 100% complete.
OK
OMAP3 beagleboard.org # nand write 80000000 80000 160000

NAND write: device 0 offset 0x80000, size 0x160000
 1441792 bytes written: OK
OMAP3 beagleboard.org #
```

Note: You can use the same u-boot.bin you use to boot from MMC/SD card for NAND. There are no differences between U-Boot used for MMC/SD card boot and NAND boot.

Note: Here, you don't need the nandecc hw option. X-Loader which loads & starts U-Boot is able to understand SW ECC written by U-Boot.

Kernel

While X-Loader and U-Boot can be written only by U-Boot, for kernel and file system there are two ways to write them to NAND: Either by U-Boot (similar way as for X-Loader and U-Boot above) or from running kernel (e.g. booted from MMC card).

Note: X-Loader and U-Boot can't be written from already running kernel, too, because from kernel point of view X-loader and U-Boot NAND partitions are marked as write only. See omap3beagle_nand_partitions[] configuration structure in kernel's arch/arm/mach-omap2 directory.

Writing kernel with U-Boot

```
OMAP3 beagleboard.org # mmcinit
OMAP3 beagleboard.org # fatload mmc 0:1 80000000 uImage
reading uImage

OMAP3 beagleboard.org # nandecc sw
OMAP3 beagleboard.org # nand erase 280000 400000

NAND erase: device 0 offset 0x280000, size 0x400000
Erasing at 0x660000 -- 100% complete.
OK
OMAP3 beagleboard.org # nand write 80000000 280000 400000

NAND write: device 0 offset 0x280000, size 0x400000
 4194304 bytes written: OK
OMAP3 beagleboard.org #
```

Once you did this, use U-Boot commands to boot kernel (uImage) from NAND:

```
nand read 80000000 280000 400000 ; bootm 80000000
```

These, you can e.g. store as bootcmd and your board will automagically boot uImage from NAND.

Writing kernel with kernel

Once you have a kernel booted, e.g. from MMC card, you can use it to write himself (uImage) to NAND and then switch from MMC boot to kernel NAND boot. For this, observe kernel's boot messages. These should have something like

```
...
omap2-nand driver initializing
NAND device: Manufacturer ID: 0x2c, Chip ID: 0xba (Micron NAND 256MiB 1,8V 16-bit
cmdlinepart partition parsing not available
Creating 5 MTD partitions on "omap2-nand":
0x00000000-0x00080000 : "X-Loader"
0x00080000-0x00260000 : "U-Boot"
0x00260000-0x00280000 : "U-Boot Env"
0x00280000-0x00680000 : "Kernel"
0x00680000-0x10000000 : "File System"
...
```

At kernel's prompt command cat /proc/mtd will give you similar output:

```
root@beagleboard:~# cat /proc/mtd
dev:    size   erasesize  name
mtd0: 00080000 00020000 "X-Loader"
mtd1: 001e0000 00020000 "U-Boot"
mtd2: 00020000 00020000 "U-Boot Env"
mtd3: 00400000 00020000 "Kernel"
mtd4: 0f980000 00020000 "File System"
```

While the first three partitions (X-Loader, U-Boot and U-Boot Env) are read only from kernel point of view, Kernel and File System partition can be written by kernel itself. To do this, you need MTD User modules (http://www.linux-mtd.infradead.org/doc/general.html) in your kernel's

root file system.

In this example we mount boot (FAT) partition of MMC card (using a dual boot card) to read kernel image (uImage) from. If you have network connection in your kernel, you can use this, too. Or you put uImage in your root file system. Goal is to have access to uImage from running kernel to be able to write it to NAND.

```
root@beagleboard:~# mkdir -p /mnt/fat
root@beagleboard:~# mount /dev/mmcblk0p1 /mnt/fat/
root@beagleboard:~# ls -la /mnt/fat
-rwxr-xr-x    1 root     root         16740 Jul  7 17:28 mlo
-rwxr-xr-x    1 root     root        717116 Jul 24  2008 u-boot.bin
-rwxr-xr-x    1 root     root       2106940 Jul 26  2008 uImage
root@beagleboard:~# cp /mnt/fat/uImage .
root@beagleboard:~# ls -la
-rwxr-xr-x    1 root     root       2106940 Jul 22 00:30 uImage
root@beagleboard:~# flash_eraseall /dev/mtd3
Erasing 128 Kibyte @ 3e0000 -- 96 % complete.
root@beagleboard:~# nandwrite /dev/mtd3 uImage
Input file is not page aligned
Data did not fit into device, due to bad blocks
: Success
root@beagleboard:~#
```

File system

As with kernel, while X-Loader and U-Boot can be written only by U-Boot, for file system there are two ways to write them to NAND: Either by U-Boot (similar way as for X-Loader and U-Boot above) or from running kernel (e.g. booted from MMC card). A lot of users report they have issues with writing (root) file system with U-Boot. Main issue is that U-Boot has to write file system exactly in format kernel expects. If there are minor incompatibilities, kernel will later not be able to read file system written by U-Boot.

So, while we document here how to write file system with U-Boot, **recommended** way is to write (root) file system by kernel itself. With this, it is ensured that kernel writes a file system it will later be able to read.

Writing file system with U-Boot

This way is not recommended. See above.

```
OMAP3 beagleboard.org # mmcinit
OMAP3 beagleboard.org # fatload mmc 0:1 80000000 rootfs.jffs2
reading rootfs.jffs2

12976128 bytes read
OMAP3 beagleboard.org # nand unlock
device 0 whole chip
nand_unlock: start: 00000000, length: 268435456!
NAND flash successfully unlocked
OMAP3 beagleboard.org # nandecc sw
OMAP3 beagleboard.org # nand erase 680000 F980000

NAND erase: device 0 offset 0x680000, size 0xf980000
Erasing at 0xffe0000 -- 100% complete.
OK
OMAP3 beagleboard.org # nand write.jffs2 80000000 680000 ${file_size}

NAND write: device 0 offset 0x680000, size 0xc60000

Writing data at 0x12df800 -- 100% complete.
 12976128 bytes written: OK
OMAP3 beagleboard.org #
```

For more info see Steve's flash procedure (http://www.sakoman.net/omap3/flash%20procedure.txt) .

Writing file system with kernel

This is the recommended way. See above.

First, we boot kernel with (root) file system on SD card, write (root) file system using file system image at SD card to Beagle's NAND with running kernel. After this is done, we switch kernel's boot arguments to take root file system from NAND instead of SD card, then.

To be able to manipulate/erase/write NAND from kernel's user space, we need MTD (http://www.linux-mtd.infradead.org/) Utils (e.g. flash_eraseall).

If you don't have them already, you can get them

- using the angstrom demo, you can install via

```
opkg install mtd-utils
```

- cross compiling them your self (good cross compile exercise)

For file system in Beagle's NAND, we use JFFS2 (http://sourceware.org/jffs2/) . As JFFS2 is part of the standard git kernel, only thing is to configure kernel to be able to use JFFS2 is to enable in make menuconfig (check if already enabled):

```
CONFIG_JFFS2_FS=y
CONFIG_JFFS2_FS_DEBUG=0
CONFIG_JFFS2_FS_WRITEBUFFER=y
CONFIG_JFFS2_ZLIB=y
CONFIG_JFFS2_RTIME=y
```

Having kernel supporting JFFS2 and MTD Utils, we now first erase file system partition and

create JFFS2 into it:

```
root@beagleboard:~# cat /proc/mtd
dev:    size   erasesize  name
mtd0: 00080000 00020000 "X-Loader"
mtd1: 001e0000 00020000 "U-Boot"
mtd2: 00020000 00020000 "U-Boot Env"
mtd3: 00400000 00020000 "Kernel"
mtd4: 0f980000 00020000 "File System"
root@beagleboard:~# flash_eraseall -j /dev/mtd4
Erasing 128 Kibyte @ f960000 -- 99 % complete. Cleanmarker written at f960000.
```

Then, we can mount "File system" partition:

```
root@beagleboard:~# cd /mnt
root@beagleboard:~# mkdir nand
root@beagleboard:~# mount -t jffs2 /dev/mtdblock4 /mnt/nand
```

and extract the root file system image to it:

```
root@beagleboard:~# cd nand
root@beagleboard:~# tar xfz <where_ever_your_root_fs_image_is_at_sd_card>/rootfs.
... wait ...
root@beagleboard:~# cd ..
root@beagleboard:~# sync
root@beagleboard:~# umount nand
```

Now, you should reboot your board and edit bootargs in U-Boot to configure root fs in NAND:

```
... root=/dev/mtdblock4 rootfstype=jffs2 ...
```

Retrieved from "http://elinux.org/BeagleBoardNAND"

Categories: Linux | OMAP
