AN10860

LPC313x NAND flash data and bad block management

Rev. 01 — 11 August 2009 Application note

Document information

Info	Content
Keywords	LPC3130 LPC3131 LPC313x LPC313X LPC3153 LPC3154 LPC3141 LPC3142 LPC31XX LPC31xx Linux kernel Apex boot loader MTD NAND flash
Abstract	Provides an overview of how the LPC31XX handles storage, data and ECC information on NAND flash; how the LPC31XX boots from NAND flash, and the NAND flash data storage approach used by the Linux kernel and Apex boot loader

Revision history

Rev	Date	Description
01	20090811	Initial release

Contact information

For additional information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

1. Introduction

The NXP LPC31XX series ARM9 based microcontrollers provide a NAND flash controller with hardware ECC support as a feature of the device. Using the built-in hardware ECC offers NAND system bandwidth improvement and lower CPU cycle usage over software based ECC. However, the use of hardware ECC had some special requirements in the areas of page data layout and bad block management.

This application note explains how the NAND flash controller handles data and ECC layout when using the built-in hardware ECC. The impact to bad block management is also covered, with a wrap-up on how the Apex boot loader and the Linux kernel handle data and ECC layout and bad blocks for NAND flash devices.

2. LPC31XX NAND overview

The LPC31XX NAND flash controller provides various features to improve the performance of a LPC31XX based system when using one or more NAND flash devices. The main performance enhancing features on the LPC31XX NAND flash controller include two 528 byte data transfer buffers, automated hardware ECC generation and write, automatic ECC read and data correction, hardware data correction and buffer statuses, and automated buffer and ECC write to NAND.

These features, when used together, can increase NAND bandwidth considerably over a software only based solution.

2.1 NAND flash controller data buffers

The LPC31XX microcontroller has two 528 byte data transfer buffers used for NAND data transfer. When software has been fully optimized to use both buffers, one buffer will be used for hardware data transfer, while the other buffer will be used to read the previous transferred NAND data or write the next NAND data for transfer. These buffers can be used to double buffer data transfer operations that extend beyond 528 bytes. To use the hardware ECC capability, the data transfer buffers must be used.

2.1.1 LPC31XX NAND data and ECC layout

A single data transfer buffer holds up to 528 bytes of data organized as 516 payload data bytes and 12 ECC bytes. This is organized as shown in Table 1.

Table 1. LPC31XX NAND data transfer buffer layout

Payload data	Spare or payload data	Stored ECC data
512 or 516 bytes	4 or 0 bytes	12 bytes

The payload data can be configured as 512 or 516 bytes in the NAND controller config register.

When configured as 512 bytes, data in offsets 512 to 515 will not be written to flash[1] and the hardware ECC will only apply to the first 512 bytes of data.

1. The 4 spare bytes in offset locations 512 to 515 are sent as 0xFF values in 512 byte mode. Due to the nature of NAND flash, sending a 0xFF will prevent a change in data in the device.

When configured as 516 bytes, the hardware ECC applies to all 516 payload data bytes and all the payload data will be written to or read from the NAND device. These 4 extra bytes can be used for status information inside the NAND device such as clean or reserved flags.

2.1.1.1 NAND data layout and hardware ECC on small block flash

For small block flash with a page size of 512 bytes + 16 spare bytes, the data transfer buffer and page size of the device are the same size. For every data buffer transferred, 1 page in the NAND device will be filled or returned.

2.1.1.2 NAND data layout and hardware ECC on large and huge block flash

Large and huge block flash devices have page sizes of 2112 (2048 bytes + 64 spare bytes) bytes and 4224 (4096 bytes + 128 spare bytes) bytes, respectively. Because the NAND flash controller data transfer buffer works in 528 byte size chunks, the required layout pattern on these devices is a multiple of the data buffer transfer size. Each buffer transfer will be a quarter (large block) or an eighth (huge block) of the total NAND page size.

For large block devices, software will place up to four of the patterns shown in Table 1 linearly in the device page. The resulting page will actually contain four payload data blocks, each with its own ECC data section. Both NAND controller data transfer buffers can be used to double buffer up to four transfers to and from the device in this case.

Huge block devices use a similar approach to large block devices, but contain up to eight payload/ECC data sections.

2.2 Factory bad block markers and hardware ECC

NAND flash devices can degrade over time and develop bad blocks. Bad blocks may not be able to be fully erased, or may have stuck bits that won't correctly toggle. Flash software usually marks a block as bad to prevent its further use in a system. Even new devices may contain bad blocks from the manufacturer, with different devices containing different locations and a different number of bad blocks.

NAND device manufacturers pre-mark bad blocks on their devices. For small, large, and huge block devices, the factory supplied bad block markers are at offsets 512, 2048, and 4096 in the first page of block. A non-zero value at these locations is used to indicate a bad block.

When using the LPC31XX NAND controller's hardware ECC, the payload data or ECC bytes may overlap the factory supplied bad block marker. This isn't a concern for small block flash in the 512 byte transfer mode, as the payload data in offsets 0 to 511 and the ECC bytes in offsets 516 to 527 don't interfere with the factory bad block marker in offset 512. However, large and huge block flash data write patterns will overwrite the factory supplied bad block marker.

Since the bad block markers will be overwritten, they will need to be managed a different way. One possible method is to relocate all the bad block markers into a byte in the spare area of the device (i.e., those 4 unused bytes when using 512 byte mode). This would have to be done when the device is new before any other write operation occurs to the device. Another approach is to generate a bad block table somewhere on the flash device and use it instead of individual block markers.

Regardless of the approach used, careful management of the bad blocks is required to prevent accidental erasure of a device's bad block status.

2.3 LPC31XX NAND boot support

The LPC31XX microcontroller can boot from the first valid NAND device connected to any one of the four NAND chip selects. For the NAND flash device to be bootable, some preparations are needed on the NAND device. These preparations consist of setting up a data structure used by the LPC31XX boot ROM in block 0 – the structure contains information about the NAND flash device and a device bad block list.

A flow diagram of the LPC31XX NAND boot process can be found in the LPC31XX User manual.

2.3.1 Bootable image preparation

The bootable image to load and execute from the NAND device is stored starting in block 1. An image header appended to the bootable image is used by the boot ROM to determine the number of pages and blocks to load into ISRAM prior to execution.

The boot ROM uses a special bad block table defined in block 0 to determine if a block should be skipped during loading of the bootable image. If this bad block table isn't detected, all blocks used for the flash boot process are assumed to be good. *This block table is only needed for the boot process for the LPC31XX boot ROM. It doesn't need to indicate all the bad blocks that may exist on the device, but just the blocks that need to be skipped during NAND boot.*

2.3.2 Block 0 preparation

The bad block list on block 0 is used for the boot ROM's load of the bootable image. Although this list is optional for the flash boot process, it is highly recommended that the list be generated. For any flash devices that have bad blocks in the range of blocks used by the bootable image or have a non-contiguous range of blocks, this step is required so the boot ROM will skip those blocks.

It is recommended that this list be created when the flash device is still new from the factory bad block markers. This will also provide a backup copy of the original factory markers in the case something accidentally gets erased.

3. The LPC31XX boot ROM, Apex, and the Linux kernel MTD driver

The Apex boot loader and Linux kernel developed for the LPC31XX make use of the NAND controller's hardware ECC capability. Both Apex and the Linux kernel use bad block marker relocation to manage bad blocks. For both Apex and the Linux kernel, the individual bad block markers are merged into one or more bad block tables.

Using the NAND controller features is mostly automatic in Apex and Linux, but does require some understanding on how it works. Understanding how Apex and Linux handle bad blocks may help prevent accidental erasure of the device's bad block tables. *It is possible to accidentally erase your device and all its saved bad block markers, but not during normal operation and use of the device.*

This section details how Apex and Linux handle bad block management and the procedures necessary to prepare a new board for NAND flash operation. It is highly recommended this section be reviewed.

3.1 LPC31XX boot ROM, Apex, and Linux kernel NAND usage

3.1.1 Apex NAND flash support

Apex can read and write to NAND flash via the copy and setenv commands and the lpcnand sub-commands. Apex NAND operations will skip bad blocks[2] as defined by the device's bad block table stored in block 0. When Apex is first started, it will detect if the block 0 table has been generated and create it if needed (if that option has been enabled in Apex). Unless the table has been accidentally deleted, this table will only be generated once during the lifetime of the device.

For new devices from the factory, Apex will automatically scan the device for factory marked bad blocks and record all the marked bad blocks in block 0[2]. This saved table is used for the boot ROM, but may also be used to restore the factory bad block markers at a later time if ever needed or if the Linux MTD bad block tables get corrupted. Write and erase operations to block 0 are prohibited from Apex unless specifically enabled. If block 0 is accidentally erased, the factory supplied bad block markers will be unrecoverable.

3.1.2 The Linux MTD driver

The Linux MTD driver generates a bad block table and a mirrored version of the table that are stored on the NAND flash device. Multiple NAND flash devices will each get their own tables. These tables are generated the first time the Linux kernel is run. When this table is generated, it is generated from the factory supplied bad block markers. After the table has been generated, erase and write operations to the device will overwrite the factory bad block markers and new bad block markers will be stored in the bad block table.

The Apex bad block table in block 0 and the Linux MTD bad block table are two different and unique tables. Over the life of the device, bad block markers updated in the Linux bad block table will not be updated in the block 0 table. This is normally not an issue, as the block 0 table is used for the boot ROM only.

If for some reason the Linux kernel is executed prior to Apex and the kernel makes changes to the NAND device, Apex will scan those changed blocks as bad blocks due to

2. This option can be disabled in the Apex configuration menu.

the factory marker locations getting changed. Although this will increase the size of the block 0 table, it won't be a problem unless blocks used for the NAND boot process are changed.

3.2 Procedures for correct setup of NAND flash

On systems with new factory formatted flash, careful preparation of the flash device is required before using it in a system running when using hardware ECC. In most cases, preparation is handled entirely in software with Apex generating the bad block table on first execution and then Linux generating the MTD bad block table on the first Linux kernel execution after Apex loads and starts it.

To start Linux, Apex has to load and run first. If Apex erases or writes to any flash block used by the Linux MTD driver prior to the Linux kernel executing, then the Linux kernel will report the Apex modified blocks as bad on its first run. These blocks aren't bad, but have had their factory supplied bad block markers overwritten in Apex. Since the MTD driver in Linux needs to build the MTD bad block table on the first run, it needs to use the factory block markers for that table. Any easy fix for this would be to allow Linux to boot once prior to doing any NAND operations in Apex (with the exception of the block 0 table). In most cases, the Apex, kernel, and ramdisk storage areas in flash are separate from the MTD storage area for the kernel. So even though the blocks may be reported as bad in the MTD, they will probably never be modified by the kernel during normal use. *The board support files provided by NXP isolate the Apex storage areas from the Linux kernel by default.*

3.2.1 Preventing bad block issues between Apex and Linux

To prevent bad block issues between Apex and the Linux kernel, avoid writing to NAND regions that are used by the other application. Apex should avoid writing to areas of NAND used for the Linux MTD, while the Linux MTD should avoid areas used for Apex.

The Linux MTD driver will only allow operations to areas of NAND that have been designated as usable for MTD. Apex has no protection against writing to areas of NAND flash usable by anything else, so use Apex with care!

3.3 Shortcomings of the current Apex/Linux kernel MTD flash management approach

The Apex and Linux kernel MTD bad block support work well in most systems, but have several shortcomings that may appear over the life of the part or may occur if the setup procedure isn't strictly followed.

With two different bad block tables, it is possible over the life of the part that these two tables will start to differ in marked bad blocks. The block 0 table used by the LPC31XX boot ROM contains only the factory marked bad block locations, while the MTD bad block table(s) contain the devices bad block markers generated over the life of the part. As the table in block 0 is used for the LPC31XX boot process (or optionally factory marker restoration), these differences probably won't matter. However, if for some reason the device needs to be restored to factory status and new bad blocks have developed in the MTD bad block tables, then the MTD bad block tables will need to be recorded and then marked as bad manually after the part has been erased.

3.4 Apex support

When Apex boots, it will generate the block 0 tables from the factory supplied bad block markers on the NAND device. Until this table is generated, Apex will not allow NAND operations. If you plan on booting Linux, make sure this table has been generated.

Apex actually generates 2 devices used to interface to NAND: the *nand* device and the *lnand* device. The *nand* device is a raw interface device to NAND and is used for diagnostics only – do not use this device for normal NAND operations. The *lnand* device is used for normal operation.

3.4.1 Erase, write, and read commands

The Apex erase, read, and write commands to the *lnand* device will automatically skip bad blocks as marked in block 0. For example, if you try to write to block 1 and block 1 is bad, it will write it to block 2 instead (assuming block 2 isn't bad). This also applies to read functions. Erase functions will skip any block designated as bad.

Examples:

erase lnand:128K+128K

copy lnand:1M+1M 0x30000000

copy 0x31000000+1M lnand:128K

3.4.2 Bad block scan functions

There are 2 bad block scan functions: *scan* and *bblist*.

The *scan* function only identifies blocks with factory located bad block markers. If any blocks have been modified, they will also be identified as bad. This function is only usable on factory fresh parts that haven't been written to yet.

The *bblist* function identifies any blocks as stored in the block 0 list. These are the original factory marked blocks prior to any writes to the device.

Examples:

lpcnand scan

lpcnand bblist

3.4.3 Factory restore function

The *restore* function can be used to restore the device to a state similar to if it was just received from the factory. The device is completely erased and any blocks designated as bad in block 0 are marked as bad.

This function can be used if the device becomes corrupted during testing. *Any new bad blocks generated on the device since the block 0 list was created are not marked as bad.*

Example:

lpcnand restore

3.4.4 Format block 0 function

The *format* command creates the block 0 table. This command should not normally be used during the life of the part. The command will not execute if a table already exists with explicitly overriding it (with a '1' option). Avoid the use of this function.

3.4.5 Block 0 protection function

Block 0 is protected from erase or writes during normal operation. This can be disabled with the *protblk0* function. It is not recommended to disable this protection or write to block 0, as the bad block table is stored there and you could accidentally erase your factory marker list.

3.5 Setting up Apex to boot from NAND (EA3131 board with large block flash)

For the LPC31XX device to boot from NAND flash, block 0 must be formatted with the NAND boot data structure and bad block list, and the executable code to boot must be located in blocks 1 and on (skipping bad blocks).

Block 0 is automatically formatted by Apex when it starts. *If this option is disabled, use the lpcnand format command to build block 0.* See the LPC31XX User manual for information on the structures located in block 0.

When Apex is booted (either from an SD card or serial port), you will see a message similar to below:

apex => mem:0x11029000+0x14dc4 (85444 bytes)

This is where Apex has been located in memory and will need to be copied from into NAND flash.

Erase blocks 1, 2, and 3 with the *erase lnand:128K+384K* command. Then copy the Apex image from memory to nand with the copy command (*copy mem:0x11029000+0x14dc4 lnand:128K)*. Apex will then be copied to NAND, skipping any bad blocks.

Configure your board to boot FROM NAND (boot0, 1, and 2 all low) and power cycle the board. It should now be booting from NAND flash!

4. Legal information

4.1 Definitions

Draft — The document is a draft version only. The content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included herein and shall have no liability for the consequences of use of such information.

4.2 Disclaimers

General — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in medical, military, aircraft, space or life support equipment, nor in applications where failure or malfunction of a NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors accepts no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is for the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from national authorities.

4.3 Trademarks

Notice: All referenced brands, product names, service names and trademarks are property of their respective owners.

5. Contents

For more information, please visit: http://www.nxp.com
For sales office addresses, email to: salesaddresses@nxp.com

Date of release: 11 August 2009
Document identifier: AN10860_1