Root Cause Analysis   
UIFS Corruption in Empty Space

Table of Contents

[1 Scope 3](#_Toc56005146)

[2 Critical Issue Description 4](#_Toc56005147)

[2.1 The Kernel Panic 4](#_Toc56005148)

[2.2 System Description 4](#_Toc56005149)

[2.3 Root Cause Analysis 4](#_Toc56005150)

[2.4 Suggested fixes 5](#_Toc56005151)

[3 Analysis of Linux Kernel 3.10 6](#_Toc56005152)

[2.1 UBIFS Stack 6](#_Toc56005153)

[Back Trace of the Kernel Panic along the UBIFS Stack 7](#_Toc56005154)

[UBIFS / UBI Layers – Analysis of ubifs\_scan 7](#_Toc56005155)

[MTD / NAND Layers – Analysis of Lower layer mtd\_read() 8](#_Toc56005156)

[NAND - Analysis of nand\_do\_read\_ops() 9](#_Toc56005157)

[3.1 NFC Driver Layer 10](#_Toc56005158)

[2.2.1 Driver Data Structures and Initialization 10](#_Toc56005159)

[2.2.2 MTD structure 10](#_Toc56005160)

[2.2.3 NAND structure 11](#_Toc56005161)

[2.2.4 Marvell NFC Structure 12](#_Toc56005162)

[2.2.5 Analysis of the Driver **orion\_nfc\_cmdFunc** 13](#_Toc56005163)

[4 Marvell Patch for LK 3.10 14](#_Toc56005164)

[4.1 Analysis of the New marvell\_nand NFC Driver 14](#_Toc56005165)

[4.2 New driver marvell\_nfc\_hw\_ecc\_bch\_read\_page 15](#_Toc56005166)

[4.3 The Marvell Patch for LK 3.10 16](#_Toc56005167)

[4.4 Problems addressed in the Patch 17](#_Toc56005168)

[3.4.1 ERR\_DBERR is cleared anyway 17](#_Toc56005169)

[3.4.2 Correctable bitflips in empty chunks reported as error & not fixed. 18](#_Toc56005170)

[4.5 A Fixed Patch 19](#_Toc56005171)

[4.6 Attachments 19](#_Toc56005172)

[5 Summary 20](#_Toc56005173)

[6 Appendix 21](#_Toc56005174)

[6.1 nand\_read\_page\_hwecc 21](#_Toc56005175)

# Scope

The purpose of this document is to analyze the suggested patch by Marvell for the empty corruption UBIFS kernel panic and decide about the final solution.

In addition, for the purpose of documentation and for better understanding of the root cause, this document includes detailed descriptions of the relevant Linux Kernel 3.10 source code.

# Critical Issue Description

UBIFS is triggering a kernel panic upon root FS mount. The kernel initialization is stopped, not completed, which causes a system reboot because a watchdog timer expires. The UBIFS kernel panic causes an endless loop of unsuccessful reboots.

## The Kernel Panic

Here is the dump of the kernel errors:

UBIFS error (pid 1): ubifs\_scan: corrupt empty space at LEB 3:7571

UBIFS error (pid 1): ubifs\_scanned\_corruption: corruption at LEB 3:7571

UBIFS error (pid 1): ubifs\_scanned\_corruption: first 8192 bytes from LEB 3:7571

UBIFS error (pid 1): ubifs\_scan: LEB 3 scanning failed

VFS: Cannot open root device "ubi0:root" or unknown-block(0,0): error -117

Please append a correct "root=" boot option; here are the available partitions:

Kernel panic - not syncing: VFS: Unable to mount root fs on unknown-block(0,0)

## System Description

Our system is legacy, using Marvell Cetus SOC with a raw 1Gbit NAND of Micron. SW is based on Linux Kernel 3.10.70 + Marvell LSP. Marvell NAND Flash Controller is used, NAND ECC is 8 bit. The NFC driver is “armada-nand” by Marvell (mtd/nand/mvebu\_nfc/nand\_nfc.c) as part of Marvell LK 3.10 LSP.

## Root Cause Analysis

The issue is reproduced when we power cycle the devices in a loop. It is probably caused by **power cuts** which happen simultaneously upon NAND writes.

Then, once power is ON, the kernel panics for a corruption in empty space as UBIFS scans LEBs while mounting the root-fs NAND partition. Corruption happens when UBIFS expects an area to be empty, e.g. all 0xFF (erased), and finds that it is not empty.

UBIFS expects the lower layer NAND driver to report for ECC errors while reading a page. The issue here is that an error is not reported to UBIFS for bitflips in empty chunks and thus UBIFS expect any empty area to be totally erased, all 0xFF. If it finds a bitflips corruption (not all 0xFF) by itself where not expected, it panics the kernel mount for potential data corruption.

So, the reason for the panic is that any bitflips corruptions are not reported by the driver for empty chunks.

Here is the explanation why the bitflips are not reported:

After a page read is done with HW ECC enabled (READ0), **MV\_NFC\_UNCORR\_ERR\_INT is not set in the STATUS regisgter (0xd0014 bit[4])**, and ERR\_DBERR never reported by orion\_nfc\_error\_check, and thus is\_buf\_blank() emptiness test never triggered for empty chunks.

## Suggested fixes

The fix suggested by Marvell:

A patch suggested by Marvell, based on the new marvell\_nand.c driver introduced in the latest Linux Kernel 4.4.120.

According to Marvell (blue), after disabling the HW ECC with nfc\_disable\_hw\_ecc(info), MV\_NFC\_UNCORR\_ERR\_INT is triggered in the STATU register, and ERR\_DBERR is reported and the test for emptyness is\_buf\_blank() is triggered.

is\_buf\_blank() will trigger an error if any bitflip is found, e.g. not all bytes equals 0xFF.

So bitflips will be reported to UBIFS and the kernel panic will be resolved.

The fix suggested by Adva:

The patch suggested by Marvell fixes the kernel panic but it implies that an error will be reported by the driver for any number of bitflips, even for very easy and correctable bitflips (e.g. even for one bitflip). So, bitflips in empty chunk are not hidden anymore, but on the other hand any bitflip is now reported.

I think the fix should align to the new marvell\_nand driver and to the other patch applied to the u-boot - report an error in empty chunks only if the number of bitflips found exceeds the supported ECC strength / threshold. This way only real uncorrectable ECC errors will be reported up to UBIFS.

A workaround suggested for Adva:

sync() all UBIFS write-back buffers upon our dying gasp handler. This would minimize the risk for NAND writes during a power cut.

# Analysis of Linux Kernel 3.10

In this section, we will describe the relevant NAND source code in the legacy LK 3.10.

## 2.1 UBIFS Stack

Recall the following Kernel layers involved for UBIFS over NAND flash devices: UBIFS / UBI / MTD / NAND / specific HW NAND driver



## Back Trace of the Kernel Panic along the UBIFS Stack

The following call stack for the kernel panic was dumped in ubifs\_scan, when UBIFS tried to replay its journal upon root-fs mount:

[<c0015050>] (unwind\_backtrace+0x0/0xf8) from [<c00115f4>] (show\_stack+0x10/0x18)

[<c00115f4>] (show\_stack+0x10/0x18) from [<c0196634>] (ubifs\_scan+0x29c/0x378)

[<c0196634>] (**ubifs\_scan**+0x29c/0x378) from [<c0196aa4>] (ubifs\_replay\_journal+0x104/0x1380)

[<c0196aa4>] (ubifs\_replay\_journal+0x104/0x1380) from [<c018caf8>] (ubifs\_mount+0xe88/0x15c8)

[<c018caf8>] (ubifs\_mount+0xe88/0x15c8) from [<c00a0830>] (mount\_fs+0x14/0xc8)

[<c00a0830>] (mount\_fs+0x14/0xc8) from [<c00b7620>] (vfs\_kern\_mount+0x4c/0xc4)

[<c00b7620>] (vfs\_kern\_mount+0x4c/0xc4) from [<c00b992c>] (do\_mount+0x1ac/0x8e8)

[<c00b992c>] (do\_mount+0x1ac/0x8e8) from [<c00ba0ec>] (SyS\_mount+0x84/0xbc)

[<c00ba0ec>] (SyS\_mount+0x84/0xbc) from [<c0674ee0>] (mount\_block\_root+0x104/0x22c)

[<c0674ee0>] (mount\_block\_root+0x104/0x22c) from [<c06751a4>] (prepare\_namespace+0x90/0x194)

[<c06751a4>] (prepare\_namespace+0x90/0x194) from [<c0674bf0>] (kernel\_init\_freeable+0x180/0x1c8)

[<c0674bf0>] (kernel\_init\_freeable+0x180/0x1c8) from [<c04de5e8>] (kernel\_init+0x8/0x154)

[<c04de5e8>] (kernel\_init+0x8/0x154) from [<c000dfd8>] (ret\_from\_fork+0x14/0x3c)

It detects the corrupted empty space, dumps the corruption, and returns the -EUCLEAN error code that causes the kernel panic.

## UBIFS / UBI Layers – Analysis of ubifs\_scan

* + Calls **ubifs\_start\_scan** (fs/ubifs)
  + Which calls **ubifs\_leb\_read** (fs/ubifs)
  + Which calls **ubi\_read** (mtd/ubi.h)
  + Which calls **ubi\_leb\_read** (mtd/ubi)

**ubi\_leb\_read** calls lower layer NAND driver functions. It an ECC uncorrectable error is detected, it returns with -EBADMSG error code, indicating a data integrity problem.

## MTD / NAND Layers – Analysis of Lower layer mtd\_read()

Ubi\_leb\_read:

* + Calls mtd\_read (mtd)
  + Which calls mtd->\_read (mtd)
  + \_read is assigned to nand\_read (mtd) (mtd/nand/nand\_base)
  + Which calls nand\_do\_read\_ops (mtd)

The flow for lower layer driver functions is invoked by the generic mtd\_read().

Mtd\_read() expects nand\_do\_read\_ops() to return a non-negative integer representing the maximum number of bitflips that were corrected on any one ECC region, if applicable; otherwise zero is expected.

For errors including uncorrectable ECC errors, the drivers will return negative ERR codes.

An error is returned to the UBI/UBIFS layers in the event of an error, or if the corrected bitflips > supported ECC threshold.

## NAND - Analysis of nand\_do\_read\_ops()

nand\_do\_read\_ops() is used to read a whole data page with ECC.

Pseudo code:

/\* cmdFunc is a function pointer that was set to **orion\_nfc\_cmdFunc** \*/

nand->cmdFunc (info) -

if command is NAND\_CMD\_READ0 - read data into into data\_buff.

if error is returned, it is saved in info --> retcode. (no info about bitflips yet)

call read\_page()

/\* it is a function pointer that was set to **nand\_read\_page\_hwecc**.

it is a function to read a page according to the ECC generator requirements;

it returns the maximum number of bitflips corrected in any single ECC step, 0 if bitflips uncorrectable, -EIO hw error \*/

nand\_read\_page\_hwecc:

for each ecc chunk (ecc step) in the page:

call correct() to check the retcode saved.

/\* in our code, correct() just checks info->retcode \*/

if uncecc - mtd->ecc\_stats.failed++;

return maxbitflips = 0;

else

return maxbitflips.

/\*if new un-correctable error found, retrun error -EBADMSG, otherwise corrected max bitflips is returned \*/

if (mtd->ecc\_stats.failed - stats.failed)

return -EBADMSG;

return max\_bitflips;

## NFC Driver Layer

The NFC driver is mtd/nand/mvebu\_nfc/nand\_nfc.c.

### 2.2.1 Driver Data Structures and Initialization

Before going further with the driver functions analysis, it is worth introducing the driver data structure and how it is initialized.

The structures are aligned to the layering of MTD / NAND / Marvell NFC HW driver.

Upon kernel initialization “marvell, armada-nand” is found compatible in DTS and the driver probe function is triggered:

static int orion\_nfc\_probe(struct platform\_device \*pdev);

In this probe function the structures for layer MTD / NAND / Marvell NFC HW driver are allocated and initialized:

* struct orion\_nfc\_info \*info; //NFC struct
* struct nand\_chip \*nand; //NAND struct
* struct mtd\_info \*mtd; //MTD struct

These structures are organized into a corresponding hierarchy referencing:

mtd\_info -> nand\_chip -> orion\_nfc\_info . (done by nand\_scan\_tail() )

The NFC in 8bit mode with DMA Disabled using BCH 8bit ECC.

The following sections include some relevant data and callbacks assigned to these structures,

### 2.2.2 MTD structure

**Data**

erase size: 262144

write size: 4096

oobsize size: 128

bitflip threshold: 8

ecc strength : 8

**Callbacks -** This how MTD is linked to the NAND layer.

mtd->\_erase = nand\_erase;

mtd->\_point = NULL;

mtd->\_unpoint = NULL;

mtd->\_read = nand\_read;

mtd->\_write = nand\_write;

mtd->\_panic\_write = panic\_nand\_write;

mtd->\_read\_oob = nand\_read\_oob;

mtd->\_write\_oob = nand\_write\_oob;

mtd->\_sync = nand\_sync;

mtd->\_lock = NULL;

mtd->\_unlock = NULL;

mtd->\_suspend = nand\_suspend;

mtd->\_resume = nand\_resume;

mtd->\_block\_isbad = nand\_block\_isbad;

mtd->\_block\_markbad = nand\_block\_markbad;

mtd->writebufsize = mtd->writesize;

### 2.2.3 NAND structure

**Data**

nand ecc mode : NAND\_ECC\_HW // value is 2

nand ecc steps : 1 //it is 1 for our system. Steps means the number of basic ECC chunks per page. ECC is calculated per chunk.

nand ecc size : 4096

nand ecc strength : 8

**Callbacks** - This is how NAND is linked to Marvell NFC layer.

nand ->ecc.read\_page = nand\_read\_page\_hwecc;

nand ->ecc.write\_page = nand\_write\_page\_hwecc;

nand ->ecc.read\_page\_raw = nand\_read\_page\_raw;

nand ->ecc.write\_page\_raw = nand\_write\_page\_raw;

nand ->ecc.read\_oob = nand\_read\_oob\_std;

nand ->ecc.write\_oob = nand\_write\_oob\_std;

nand ->ecc.read\_subpage = nand\_read\_subpage;

nand ->ecc.write\_subpage = nand\_write\_subpage\_hwecc;

nand->waitfunc = orion\_nfc\_waitfunc;

nand->select\_chip= orion\_nfc\_select\_chip;

nand->dev\_ready= orion\_nfc\_dev\_ready;

nand->cmdfunc = orion\_nfc\_cmdfunc;

nand->read\_word= orion\_nfc\_read\_word;

nand->read\_byte= orion\_nfc\_read\_byte;

nand->read\_buf = orion\_nfc\_read\_buf;

nand->write\_buf = orion\_nfc\_write\_buf;

nand->block\_markbad= orion\_nfc\_markbad;

nand->ecc.mode= NAND\_ECC\_HW;

nand->ecc.hwctl= orion\_nfc\_ecc\_hwctl;

nand->ecc.calculate= orion\_nfc\_ecc\_calculate;

nand->ecc.correct= orion\_nfc\_ecc\_correct;

nand->ecc.size= pg\_sz[info->page\_size];

nand->ecc.layout= ECC\_LAYOUT;

### 2.2.4 Marvell NFC Structure

**Data**

info.ecc\_type: 2 // MV\_NFC\_ECC\_BCH\_1K

info.page\_size: 4096

nfc\_width : 8 bits

nfc\_mode: normal. // NAND mode - normal or ganged

num\_cs = 1 // Number of NAND devices

page\_per\_block: 64 //Pages per block, so a block is 256K

### 2.2.5 Analysis of the Driver **orion\_nfc\_cmdFunc**

orion\_nfc\_cmdFunc is the function that implements the actual lower layer chip commands.

In our case, the NAND\_CMD\_READ0 argument is used to read the data of a page chunk by the NFC.

* Data is read into info->data buffer by calling orion\_nfc\_do\_cmd\_pio(info), where info is a pointer to struct of type orion\_nfc\_info. orion\_nfc\_do\_cmd\_pio reads the NFC STATUS Register (0xD0014) and checks for errors when any lower NAND command is completed.

If any error found, like ECC uncorrectable data error (MV\_NFC\_UNCORR\_ERR\_INT), then the error return value is saved into info->retcode. For ECC uncorrectable error, retcode is set to ERR\_DBERR;

* If ECC Uncorrectable error is returned, then orion\_nfc\_cmdFunc checks for the reason.

It checks if this is a real ECC error in the data or a “fake” error that was calculated just because the page includes empty / erased chunks. It might be that for empty chunks, ECC was not calculated and an error is always returned. This case must be handled by the driver to avoid reporting a fake error for empty chunks.

* So, if ERR\_DBERR is returned in info->retcode, the page is tested for emptiness, to check if this is the case of empty chunks.

The emptiness test is done by a simple is\_buf\_blank() function which searches the data buffer to check it is all 0xFF. If any of the bytes of the buffer is not 0xFF, the page is considered as non-empty and the ERR\_DBERR is preserved, otherwise if all 0xFF, it is an empty page and the error is cleared and retcode is set to ERR\_NONE !

# Marvell Patch for LK 3.10

The patch is based on the new marvell\_nand driver mainlined into LK 4.4.120. It has a totally new implementation.

## Analysis of the New marvell\_nand NFC Driver

The relevant differences related to the legacy drivers are:

1. Before searching the chunks for emptiness, marvell\_nfc\_disable\_hw\_ecc() is called to disable ECC in HW.
2. The big difference is that when erased chunks are tested for emptiness upon read operation, it is not just that the error is cleared if any byte is NOT 0xFF (like in the legacy driver), bitflips in empty chunks are also corrected if the flips number is below the supported ECC correctable level / threshold.

So, if bitflips are below the ECC correctable level, they are fixed, and the page is reported as erased. Otherwise an error is returned only if uncorrectable bitflips are found. The new nand\_check\_erased\_ecc\_chunk(), introduced in kernel 4.10, is used instead of the legacy is\_buf\_blank(), it checks and corrects empty chunks.

Technically, an error mask is used to mark the errored chunks along the whole page read, and then only those chunks are tested for emptiness and “good” chunks are skipped.

## New driver marvell\_nfc\_hw\_ecc\_bch\_read\_page

Read of a page with BCH ECC is implemented in this new function.

Pseudo code analysis:

marvell\_nfc\_enable\_hw\_ecc(chip);

for each chunk in the page:

read the chunk: data + spare area

check for bitflips:

if uncorrectable ECC, ret is error and mark this chuck as errored in mask.

(might be an empty chunk, emptiness will be tested)

if bitflips is correctable, save max bitflips.

marvell\_nfc\_disable\_hw\_ecc(chip);

for each chunk in the page:

if not masked as errored - skip.

if marked as uncorrectable ECC:

check for emptiness under the ecc strength / threshold:

return error only if the number of bitflips > threshold.

## The Marvell Patch for LK 3.10

The patch is applied to **orion\_nfc\_cmdFunc** as described in **2.2.5.**

The major change for the legacy driver was the addition of the **marvell\_nfc\_disable\_hw\_ecc()** before the a chunk is tested for emptiness by is\_buf\_blank().

According to Marvell (blue), MV\_NFC\_UNCORR\_ERR\_INT bit was never set in the NFC STATUS register (0xD0014) and thus is\_buf\_blank() never triggered to check for emptiness. The result is that ECC errors upon reads were never reported even for easy & correctable bitflips.

According to Marvell, by disabling the HW ECC, MV\_NFC\_UNCORR\_ERR\_INT is set and is\_buf\_blank() is now triggered. The expected result from the patch is that empty chunks will now be tested for emptiness and an error should not be reported for them. Errors will be reported only if the chunks are not empty and any bitflip is found. As mentioned, please note that an error is reported even for correctable bitflips.

## Problems addressed in the Patch

### 3.4.1 ERR\_DBERR is cleared anyway

In orion\_nfc\_cmdFunc, where the patch is applied, I found that at the end of the function ERR\_DB\_ERR is cleared and ERR\_NONE is always set anyway.

This calls to question the effect of the patch if an error is always cleared.

Here is the function source code:

static void orion\_nfc\_cmdfunc(struct mtd\_info \*mtd, unsigned command,

int column, int page\_addr)

{

struct orion\_nfc\_info \*info =

(struct orion\_nfc\_info \*)((struct nand\_chip \*)mtd->priv)->priv;

….

switch (command) {

case NAND\_CMD\_READOOB:

…

break;

case NAND\_CMD\_READ0:

…

orion\_nfc\_do\_cmd\_pio(info);

nfc\_disable\_hw\_ecc(info);

if (info->retcode == ERR\_DBERR) {

/\* for blank page (all 0xff), HW will calculate its ECC as

\* 0, which is different from the ECC information within

\* OOB, ignore such double bit errors

\*/

if (is\_buf\_blank(info->data\_buff, mtd->writesize))

info->retcode = ERR\_NONE;

else

printk(PRINT\_LVL "%s: retCode == ERR\_DBERR\n", \_func\_\_);

}

break;

….….

default:

pr\_err("non-supported command.\n");

break;

}

**if (info->retcode == ERR\_DBERR)** {

pr\_err("double bit error @ page %08x (%d)\n",

page\_addr, info->cmd);

info->retcode = ERR\_NONE;

}

}

### 3.4.2 Correctable bitflips in empty chunks reported as error & not fixed.

In addition, as explained above, the legacy code will trigger is\_buf\_blank() for emptiness, and this function will report an error for any number of bitflips, even for correctable numbers. I suggest that we apply the code from the new driver that reports errors only if the bitflips number is above the ECC correctable level.

This change was introduced in the new driver and applied in the other patch by Marvell for the u-boot. I think it should be applied in this patch too.

## A Fixed Patch

1. Prior to Marvell patch (mainlined in LK 4.4.120), we applied commits that introduced emptiness checks, and correction of bitflips in chunks where the number of bitflips are less the ECC strength / threshold.

I prefer to use these functions because they are already mainlined into the kernel 4.10.

<https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=730a43fbc135e593cc3de3b1b895e49c05c8e2dc>

<https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=40cbe6eee97b706f27bcc4c6aa1018bbe4f1e577>

Attached our patch for LK 3.10 to include these commits too.

1. In the orion\_nfc\_cmdFunc, replace is\_blank\_buf() for emptiness check with the new nand\_check\_erased\_ecc\_chunk(). If bitflips are below the correctable level, report the page as erased and fix the flipped bits, otherwise report uncorrectable page error.

This logic was also applied by Marvell to the u-boot NFC driver and I think it should be applied for the kernel as well.

1. I didn’t fix the clearance of errors at the end of orion\_nfc\_cmdFunc(), because it might have an impact on any read operation and I prefer to double check with Marvell first.

## Attachments

* Marvell patch for LK 3.10
* Marvell patch for the u-boot
* Legacy LK 3.10 nand\_nfc.c source code
* ADVA patches:
  + A fixed version for Marvell LK 3.10 patch
  + A patch for the empty chunks functions introduced in LK 4.10.

# Summary

Not sure about the original patch because of the problems found in it. According to Marvell, it resolved the issue, so maybe the call to **marvell\_nfc\_disable\_hw\_ecc** has other impact that helps.

I applied both a fixed patch and the workaround for our dying gasp handler.

The fix in the patch only applies to the emptiness check of chunks and handle correctable number of bitflips.

# Appendix

## nand\_read\_page\_hwecc

Please note that orion\_nfc\_correct() always returns with 0 for success or -1 for error. Since correct() will always return 0 for successful reads, nand\_read\_page\_hwecc will never accumulate a positive max\_bitflips value as expected for those reads.

for (i = 0 ; eccsteps; eccsteps--, i += eccbytes, p += eccsize) {

int stat;

**/\* Barak: correct returns -1 or 0\*/**

stat = chip->ecc.correct(mtd, p, &ecc\_code[i], &ecc\_calc[i]);

if (stat < 0) {

mtd->ecc\_stats.failed++;

} else {

**/\* Barak - if stat is 0, max\_bitflips is not incremented ?\*/**

mtd->ecc\_stats.corrected += stat;

max\_bitflips = max\_t(unsigned int, max\_bitflips, stat);

}

}