NAND BBM in Linux File System

Author: Zhi Feng

1 Introduction

The NAND Flash is a popular storage media as it dominates the non-volatile Flash market. NAND Flash's storage capacity,power efficiency, cost effectiveness, and scalable design make it an ideal choice for a wide range of applications.However, one of the inherent challenges with NAND flash is the presence of bad blocks thateither already exist at manufacture or become unreliable over time due to wear and other factors. Managing these bad blocks efficiently is crucial to ensure data integrity and system stability.

Two common file systems designed for use with NAND flash on Linux systems are UBIFS (Unsorted Block Image FileSystem) and JFFS2 (Journaling Flash File System version 2). Both file systems have built-in mechanisms for handling bad blocks, but they work in different ways.

This application note explores how these file systems handle bad blocks and provides best practices for ensuring data integrity and performance when using NAND flash on Linux systems.

2 NAND Bad Blocks

Bad blocks in NAND flash memory can either be present from the factory (known as factory bad blocks) or develop overtime during the use of the device (known as runtime bad blocks). NAND flash memory is expected to have a certain number of bad blocks, and file systems designed for NAND must handle these bad blocks gracefully.

- **Factory Bad Blocks**: These are blocks that were marked as defective during the manufacturing process and are flagged before the device is shipped.
- **Runtime Bad Blocks**: During the lifetime of a NAND device, additional bad blocks may occur. This is completely normal, and flash manufacturers cannot avoid this in advance.

  Typically, flash manufacturers would guarantee that the total number of factory bad blocks and runtime bad blocks will not exceed certain numbers. In Skyhigh Memory NAND devices, the total number of bad blocks will not exceed 2% of NAND total blocks during the device lifetime.

2.1 Bad Block Marker

In order for the host software to recognize factory bad blocks, and manage bad blocks, all bad blocks should be marked in a pre-determined method. In Skyhigh Memory NAND devices, any block where the 1st byte in the spare area of the1st or 2nd or last page does not contain FFh is a Bad Block. That is, if the 1st byte of the first page has an FFh value, then the second page and the last page should also be checked. If any of these pages has a non-FFh value in its 1st byte of the spare area, that will indicate a bad block. Table 1 shows how to mark the block as bad and how to check if it is a badblock.

Furthermore, the bad block Information must be read before any erase is attempted, as the bad block marker could beerased, resulting in incorrect usage of such bad blocks.

Table 1. Bad Block Marker Marking and Checking Method

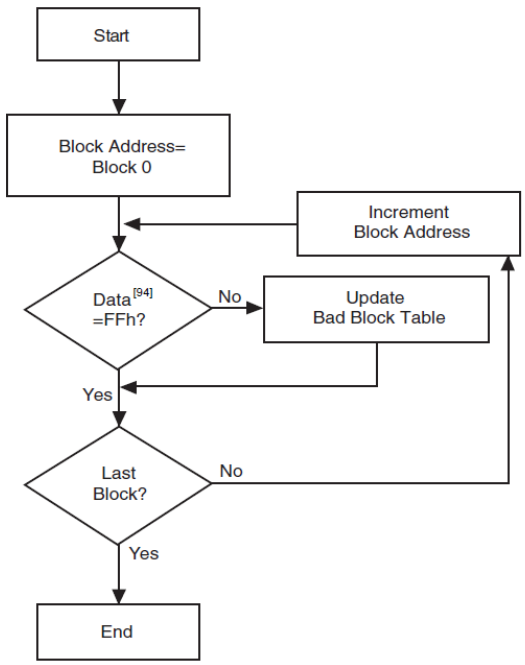
Bad Block Marker		Marking (execute on all 3 pages)	Checking (any of the pages)
Page 0	1st byte of the spare area	Write to 00h	Read if not FFh
Page 1	1st byte of the spare area	Write to 00h	Read if not FFh
Last page of the block	1st byte of the spare area	Write to 00h	Read if not FFh

3 General Best Practices for Handling NAND Bad Blocks

3.1 Bad Block Table (BBT) Management:

The BBT is a table maintained by the MTD subsystem in Linux to track bad blocks in NAND flash. Ensure that the BBTis updated regularly during system operation, and the MTD driver is properly configured to handle bad block detection and reporting. Figure 1 shows the general bad block management flowchart.

Figure 1. Bad Block Management Flowchart



3.2 Wear Leveling:

Most flash file systems use wear leveling techniques. By balancing out the erase counts on all blocks, wear levelingcan prevent certain blocks to become bad even if the user data are updated often in those blocks. On the logicallevel, some blocks may be updated much more often than others; however, with wear leveling, all blocks aremaintained about the same erase cycle count.

When using wear leveling, you should ensure to have static wear leveling enabled (in UBIFS/UBI). Because if you canboth static and dynamic data partitions, static wear leveling can greatly extend the life span of the flash byredistributing static data periodically.

3.3 ECC (Error Correction Codes):

Use the ECC algorithm specified by the NAND vendor for the specific NAND devices being used. Even when some biterrors happen in a NAND page, which is completely normal, the block can still be a good block. Only whenuncorrectable ECC errors happen consistently, a block may be marked as a bad block. ECC mechanism can correctmany of these errors before they result in bad blocks.

3.4 Regular Monitoring:

Periodically check the status of the NAND device for growing numbers of runtime bad blocks. Both UBIFS and JFFS2report errors in system logs, so it's essential to monitor them for signs of impending failure.

4 Handling Bad Blocks with JFFS2

JFFS2 is one of the oldest file systems for NAND flash, and it's designed with wear leveling and bad block management inmind. It treats flash as a log-structured file system, meaning that data is written sequentially, and old data is garbage-collected.

4.1 Key Features of JFFS2:

- **Log-structured design**: Data is written in a log format, allowing the file system to avoid writing to bad blocks.
- **Garbage collection**: Old data is erased and blocks are reclaimed, preventing frequent writes to the same blocks and enabling wear leveling.
- **Bad block management**: JFFS2 detects bad blocks during writes and ensures they are not used for storing valid data.

4.2 How JFFS2 Handles Bad Blocks:

4.2.1 Bad Block Detection:

JFFS2 works with the MTD in Linux, which interacts directly with the NAND hardware. The MTD driver automatically detects and marks bad blocks by reading the bad block markers provided by the NAND device.

During runtime, if JFFS2 encounters a bad block, it skips it and continues writing to the next available good block.

4.2.2 Wear-Leveling and Bad Block Avoidance:

JFFS2 performs wear leveling to distribute write and erase operations evenly across the flash memory, reducing the likelihood of premature bad block creation.

If a block fails during programming or erasing, JFFS2 marks it as bad and ensures it is no longer used for writing data.

4.2.3 Data Recovery:

In the event of a power failure or sudden shutdown, JFFS2 can recover data due to its journaling nature. Any incomplete or corrupted writes are detected during the next mount, and JFFS2 reconstructs the file system from thevalid logs.

4.2.4 Best Practices for JFFS2:

- **Frequent Backups**: Although JFFS2 handles bad blocks well, having a backup strategy is crucial to prevent dataloss due to unexpected hardware failures.
- **Avoid Frequent Mount/Unmount Cycles**: Each time JFFS2 mounts, it scans the entire file system, which can be time-consuming for large NAND devices. Keeping the file system mounted can improve performance.
- **Clean Marker and Number of Operations per page**: Some NAND devices can only allow certain number of writes per page, for example, NOP=4 means a page can only be written 4 times. Because JFFS2 writes a cleanmarker to the spare area after an erase operation, sometimes it may violate the number of writes and can lead to errors. In this case, one may disable the clean marker feature in the JFFS2 configuration.

5 Handling Bad Blocks with UBIFS

UBIFS is a newer file system designed specifically for flash memory. Unlike JFFS2, UBIFS does not directly interact with NAND hardware but works on top of UBI (Unsorted Block Images), a layer that abstracts the flash hardware details and manages bad blocks at a lower level.

5.1 Key Features of UBIFS:

- **UBI Layer**: UBIFS relies on UBI to manage the physical characteristics of NAND, such as bad block handling, wear leveling, and space management.
- **Scalability**: UBIFS is to scale well on large NAND devices, supporting partitions of several gigabytes efficiently.
- **Dynamic and Static Wear-Leveling**: UBI ensures even distribution of writes across all blocks, reducing wear and minimizing the creation of bad blocks.

5.2 How UBIFS Handles Bad Blocks:

5.2.1 Bad Block Management by UBI:

The UBI layer is responsible for handling bad blocks on the NAND flash. When UBI detects a bad block duringerase or write operations, it remaps the data to a good block and updates the internal metadata to reflect this.
UBIFS itself does not need to deal with bad blocks directly, as UBI hides the bad blocks from UBIFS.

The UBI marks a bad block if an error is detected during an erase or program operation. If a read error happens, it does not mark the block immediately, instead, if starts a “torturing” process, that erases and programs somedata to the block to verify if the operations are successful. It is important to ensure the ECC mechanism used bythe UBI is the same as what the NAND devices require. Otherwise, unnecessary “torturing” process may occur.

5.2.2 Automatic Remapping:

When a block is found to be bad during runtime, UBI automatically remaps it to a new block, making sure no data is written to bad blocks.
UBI also keeps track of how many bad blocks have been encountered and ensures that they are not used in future operations.

5.2.3 Wear-Leveling:

UBI handles both dynamic and static wear leveling. Dynamic wear leveling ensures that write operations are spread evenly across all blocks, while static wear leveling ensures that even blocks that store static data are occasionally moved to avoid uneven wear.

5.2.4 UBIFS Consistency:

UBIFS maintains consistency with the help of UBI. In case of power failure or unclean shutdowns, UBIFS can recover data and maintain file system integrity by referencing UBI's metadata, ensuring no data is lost even if bad blocks appear during runtime.

5.2.5 Best Practices for UBIFS:

- **ECC correction threshold**: Because different NAND devices may require different level of ECC correction capabilities, it is important to set up UBI low level driver to be consistent with the device requirement. This istypically done by changing the "`bitflip_threshold`" value in the MTD. This variable is defined in mth.h starting with Linux version 3.8. It is highly recommended to update to Linux version 3.8 or later when using UBIFS withNAND devices.

- **Monitor UBI Health**: Regularly check the UBI health reports to monitor the number of bad blocks and the wear level of the NAND device.

- **Minimize Erase Operations**: Avoid frequent data deletion and rewriting, as these operations can accelerate block wear and lead to the development of more bad blocks.

6 Conclusion

Handling bad blocks in NAND flash is critical for ensuring the reliability and longevity of embedded systems. JFFS2 and UBIFS, the two most common Linux file systems for NAND, both have robust bad block management mechanisms. JFFS2handles bad blocks directly, while UBIFS relies on the UBI layer for managing bad blocks and wear leveling.Understanding the nuances of each file system and following best practices can help prevent data loss and maximize the lifespan of NAND devices.

By implementing careful wear leveling, monitoring for bad blocks, and using appropriate error correction, embedded systems developers can ensure that their systems continue to function even in the face of inevitable NAND blockfailures.

7 Document History

Document Title: AN223240 - NAND BBM in Linux File System

Document Number: 002-23240

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**		ZFENG	10/02/2024	New Application Note