White Paper

Flash File Systems Overview

Table of Contents

1.0 Overview

Flash memory is a nonvolatile memory, which allows the user to electrically program (write) and erase information. The exponential growth of flash memory has made this technology an indispensable part of hundreds of millions of electronic devices.

Flash memory has several significant differences with volatile (RAM) memory and hard drive technologies which requires unique software drivers and file systems. This paper provides an overview of file systems for flash memory and focuses on the unique software requirements of flash memory devices.

1.1 Flash Architecture

1.1.1 Partitions

Flash devices are divided into one or more sections of memory called partitions. A multi-partition architecture allows a system processor to read from one partition while completing a write/erase in another partition. This permits executing code *and* programming data in the same flash device at the same time. In a device with only one partition similar multi-tasking may be done but it must be handled in software.

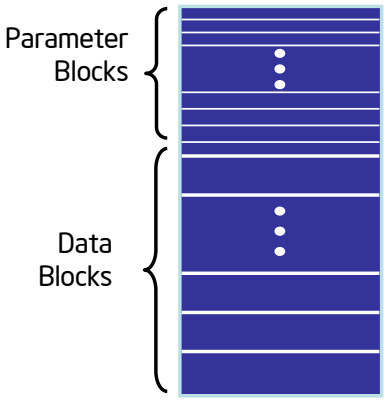

Figure 1 - Asymmetrical Blocking

1.1.2 Blocks

In addition to partitions, flash devices are further divided into sections of memory called blocks. Flash memory devices are available in symmetrical and asymmetrical blocking architectures as shown in Figure 1. Devices with all blocks the same size are called symmetrically-blocked. Devices that are asymmetrically-blocked typically have several blocks that are significantly smaller than the main array of flash blocks. Small blocks or parameter blocks are typically used for storing small data or boot code. Block sizes vary but typically range from 64Kb to 256Kb.

1.2 Programming Data

Flash devices allow programming values from a “1” to a “0”, but not from “0” to a “1” value. To program values back to “1”s requires erasing a full block. In most cases when data is edited it must be written to a new location in flash and the old data invalidated. Eventually invalid data needs to be reclaimed and this is usually done as a background process.

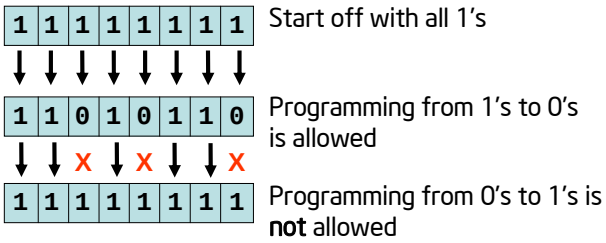

Figure 2 - Flash Programming Limitations

1.3 Data Integrity

The properties of flash memory make it ideal for applications that require high integrity while operating in challenging environments. On the hardware level, the integrity of data written to flash is generally maintained through ECC algorithms. In the case of NAND, bad block management is another data integrity issue. NAND flash is inherently less reliable than NOR flash and it is assumed that a certain percentage of blocks in a device will go “bad” during the device lifetime. Software is used to maintain a list of bad blocks which cannot be used.

Another data integrity issue is power loss. When power is lost *during* a write operation, ensuring data integrity is handled in a file system. Flash file systems must ensure that no data is corrupted regardless of when power-loss occurs.

2.0 Flash File System Functions

While flash file systems have many functions in common with file systems for other media there are many needs that are unique to file systems for flash devices. This section documents software features that address the distinctive requirements of flash memory.

2.1 Wear Leveling

Each block in a flash memory device has a finite number of erase-write cycles. To increase the longevity of a flash device, writes and erases should be spread as evenly as possible over all of the blocks on the device. This is called wear leveling. Wear leveling is generally done in software and while it is a relatively simple concept, care must be taken in the software to balance performance with even wear leveling of blocks.

2.2 Reclaim

As described in section 1.2, edits to files are usually not done "in place," rather data is written to a new location and the old data is invalidated. The invalid data must be cleaned up at regular intervals and this process is called garbage collection or reclaim. When a block is reclaimed the valid data is copied to a clean (erased) block of data called the spare block. When the reclaim process is completed the old block is erased and it becomes the new spare block as shown in Figure 3.

Generally, reclaim is done as a low priority background task, however, if the file system is critically low on free space, the file system will call a "state of emergency" and initiate a reclaim as a foreground (high priority) task. Some file systems also use the garbage collection process to perform other non-critical functions to make the file system stable or speed up future writes. Intelligent reclaim algorithms can reduce file system fragmentation and increase file system performance.

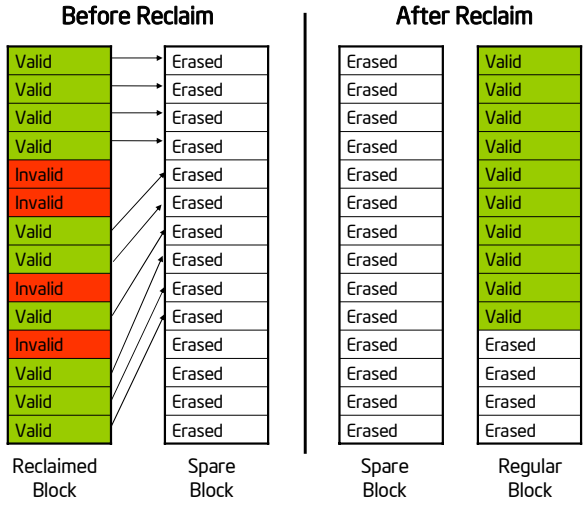

Figure 3 - Reclaiming Invalid Data

2.3 Read While Write

Many real-time applications require the capability to interrupt flash operations (write, erase) with a higher priority request. For example, while a flash device is writing data a high priority read request may need to be executed. Many flash devices provide the capability to suspend an operation and initiate a second operation and then return back to the first operation. In a multi-partition device (shown in Figure 4) each partition can execute read and write commands independent of each other. While mainly a hardware function, Hardware Read While Write requires software support.

In a single partition device, Read While Write is done entirely in software by suspending a write (or erase) and then initiating a read (as shown in Figure 5.) While software Read While Write provides excellent flexibility it usually comes with a performance tax for suspending / un-suspending flash operations.

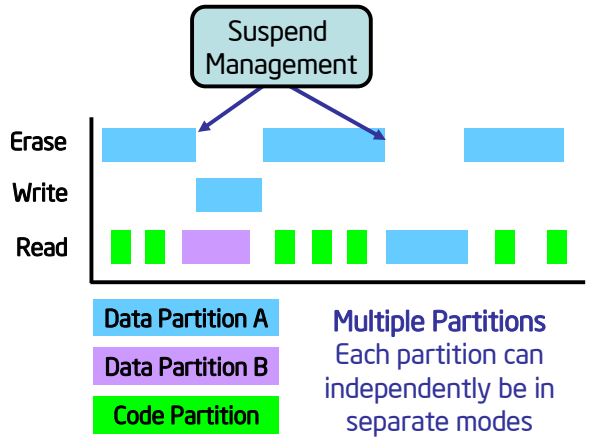

Figure 4 - Read While Write in a Multi-Partition Device

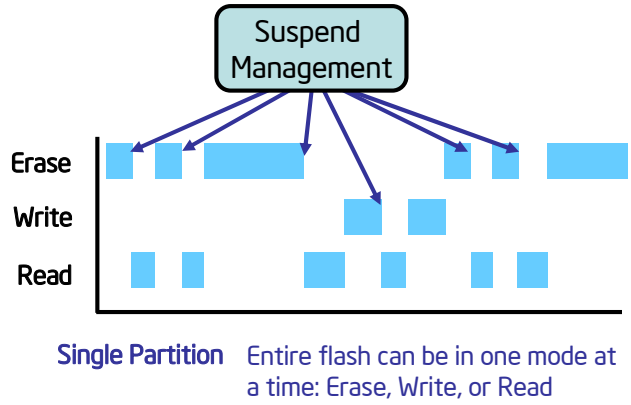

Figure 5 - Read While Write in a Single Partition Device

2.4 Memory Array Management

Flash devices come in a variety of memory array configurations including single and multiple partitions,

symmetrical and asymmetrical blocking, top and bottom boot configurations and more. In addition flash devices may be stacked together to provide even more configuration variations. In order to maximize the use of flash memory and support different memory configurations a flash file system should support the ability to map the memory in flash device to a configuration required by the application.

Gap support is an example of this. When two flash devices are stacked together applications may need the two flash devices to appear as one contiguous array of memory addresses. A boot block "in between" the two flash devices is seen as a gap in the flash memory array (as shown in Figure 6).

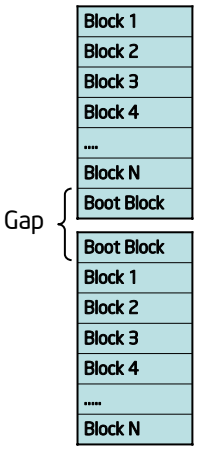

Figure 6 – Example of Memory Address Gap in Stacked Flash Devices

Another example of flash array management is virtual partition support. Applications generally write code and data into separate partitions on a flash device. In a device with a single partition (or very large partitions) a file system can provide a virtual partition capability to separate code and data along boundaries defined in software. Software partitions can be more efficient than hardware partitions because designers can arbitrarily specify the size of a code partition and designate the rest of the flash device for data instead of only being able to define partitions based on hardware boundaries.

2.5 Code Management

Flash devices are used to store both code (software executables) and data. There are two types of code management techniques available for embedded systems: Store and Download (SnD) and eXecute in Place (XiP).

In a SnD system, code is copied into RAM and is executed from RAM (as shown in Figure 7). SnD systems may load a complete executable into RAM or load parts of the application into RAM as needed (demand paging). Demand Paging reduces RAM utilization at the expense of performance.

XiP systems execute code directly from flash without having to copy the code into RAM (as shown in Figure 8). The XiP model reduces the amount of system RAM required and decreases system startup time.

XiP requires a random access memory device and so XiP can only be supported by NOR flash. NAND devices only support block addressing and hence can only employ a SnD method. Software support is required for XiP including writing code to flash in a contiguous, sequential address space and satisfying any operating system requirements such as page alignment. A file system may also support code compression such as the XiP file system cramfs used in Linux systems. Code compression reduces the amount of space allocated for code on a flash device.

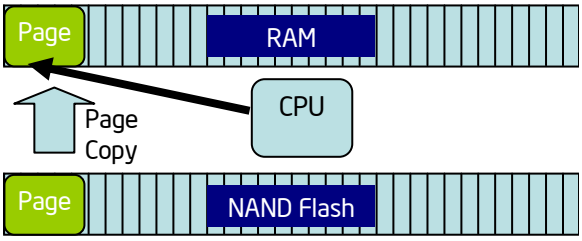

Figure 7 - Store and Download Code Model

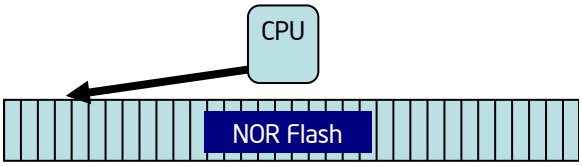

Figure 8 - XiP Code Model

3.0 File System Architecture

3.1 Architecture/Modular Design

Although flash file systems vary in their architecture most have the following components:

- API (Application Programming Interface) Layer
- File System Core
- Block Driver (for sector based file systems)
- Memory Technology Device (MTD) Layer

Dividing a file system into the layers described above provides a level of modularity that insulates the operating system and applications which use the file system from internal file system changes and minimizes the impacts of

device level changes on the file system. Figure 9 shows file system components for both sector and non-sector based file systems.

The API layer provides external applications with access to the functions of the file system. The API layer allows the internals of the file system to change without affecting applications that use the file system.

Sector based file systems (e.g. FAT) usually have a sector manager layer that provides an API for basic sector management functions such as reading, writing and deleting sectors.

The MTD provides specific information about the flash devices such as type of device, buffer sizes, block and partition sizes and erase regions to the flash file system. The ability to identify multiple devices is an important MTD feature as various types of flash (NOR & NAND) have to be supported by the file system. The MTD is the main repository of device specific code optimizations such as optimizations for block and page sizes, buffer sizes and read and write limitations.

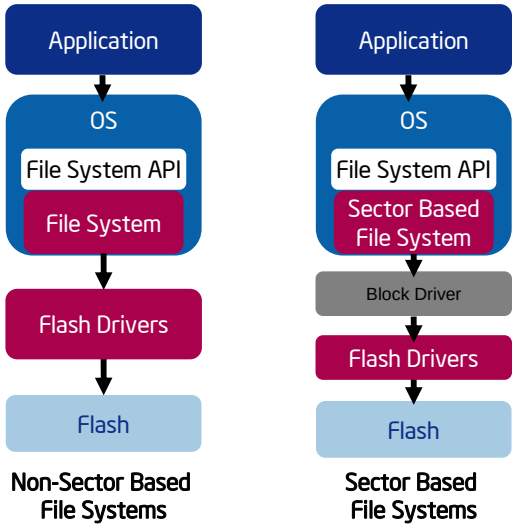

Figure 9 - Flash File System Environment

4.0 Reliability

This section describes the role of flash file system software in enhancing flash reliability. In this section we address power loss, flash reliability, and bad block management as well as recovery methods for these problems.

4.1 Power Loss Recovery

Power Loss Recovery is an essential element of a non-volatile memory file system and there are two power loss scenarios that need to be considered: power loss during program and power loss during erase. If power loss occurs during a read operation, it simply means that the read did not take place.

If power loss occurs during a program operation, the file system needs to return to the last known good state of the file system. Note that both the data that was written and the file system structures must be protected from corruption. Similarly, power loss can take place during an erase procedure. It is critical to have a recovery mechanism that prevents future program operations from occurring in the partially erased block and completes the initial erase attempt. Furthermore, a file system needs to be able to recover from single as well as double power loss. Double power loss means that the power is lost while recovering from the first power loss event.

Power loss recovery generally uses status bits or a Cyclic Redundancy Check (CRC) value. When using status bits, the application sets a status bit to indicate that a write operation completed successfully. During power loss recovery the application checks the status bit. If it is set, the data is considered valid. When using the CRC method, a CRC value is written during a write operation along with the data. During power loss recovery a new CRC is generated against the data written and compared with the existing CRC. If the two CRC numbers do not match, the data is considered invalid.

4.2 Error Code Correction (ECC)

Error Code Correction (ECC) is a way to identify and correct errors during read or write operations to flash. ECC is very common in NAND flash due to NAND reliability issues. Most ECC for flash can detect and correct single bit errors. However as NAND, and some NOR, devices trend toward Multi-Level (MLC) architecture and smaller lithographies, there is a need to perform multiple bit correction as error rates increase.

ECC is generally performed within a memory controller although software ECC is also possible. Several ECC algorithms are available, including Hamming Code, BCH (Bose, Chaudhuri, Hocquenghem), and Reed-Solomon - three that are among the most popular. ECC algorithms vary in complexity and their impact on design cost.

4.3 Bad Block Management (NAND only)

A flash device is divided into pre-defined blocks. In the case where multiple bit errors occur and are not correctable, the block is considered bad. These blocks are considered unusable and should not be programmed into any further. Due to reliability issues, NAND flash generally ships with existing bad blocks and in addition can develop bad blocks while in use. Most manufacturers indicate that 98% of the total blocks should be functional for a device to be considered utilizable. Bad blocks that exist in the shipped NAND flash, are usually marked bad in the flash at a location defined by the manufacturer in device specifications.

Figure 10 provides an example of how bad blocks are indicated in a shipped NAND device. A NAND device is split into Main and Spare Areas. Originally, in erased state, all the

bits are set to 1. Manufacturers use the spare area to indicate which blocks are bad.

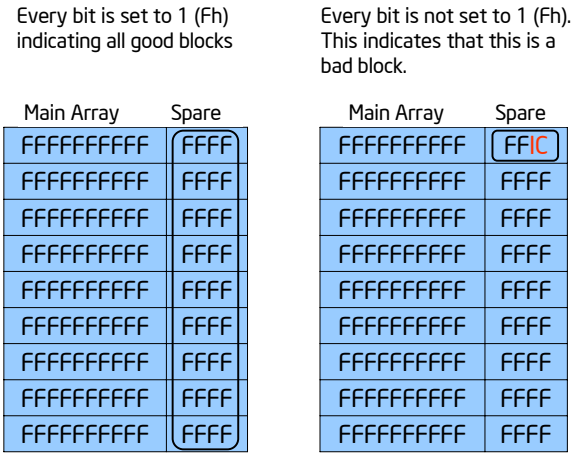

Figure 10 - Bad Block Indicators

Another aspect of bad block management is recovery. When a block goes bad while in use, it may contain previously programmed valid data and according to many device manufacturers may be recoverable.

A file system needs to recognize and account for bad blocks shipped with a device as well as blocks that go bad during use. In the latter case the current program operation should be restarted and should be completed in another "good" block. A file system also tracks the number of bad blocks for the device to ensure the bad block count is not exceeded. Additionally, the file system should have the ability to distinguish between the good and bad blocks despite power loss.

5.0 Flash File System Performance

5.1 The Importance of Flash File System Performance

Historically, flash file systems in embedded devices focused on reliability for critical user and system data. The focus from stability to performance came with the emergence of multi-media and converged devices, such as digital cameras, MP3 players, and smart phones. The user experience is now defined by response time, and reliability is assumed. With limited processor and bus speeds, a file system must be optimized to ensure acceptable performance.

A converged device such as a smart phone with multi-media capabilities is a good example of the challenging use case scenarios for file system performance. Files range in size from small critical system files to large multi-media files. The frequency of updates in these files also has a similar range in variability.

A flash file system needs to be optimized for all of these file types and use cases – providing multi-media read/write performance and at the same time maximizing the use of space for small system files. Some key file system functions that require optimal performance are:

- Read Speed
- Write Speed
- Reclaim
- Initialization Time
- File operations (create, open/close, rename, delete, find)

A flash file system also needs to ensure that performance does not degrade over time as a result of fragmentation.

Optimization Methods

While techniques vary, most high performing file systems use the following techniques among others for optimal performance:

- Caching schemes to minimize the number of flash vs. RAM accesses
- Intelligent reclaim algorithms to minimize the number of reclaims
- Algorithms to minimize system fragmentation
- Multi-threaded operations, including allowing file reads while writing to a file

6.0 Summary

This paper has addressed the various aspects of flash file system software and how it impacts the performance, longevity and data integrity of a flash device. A well designed flash device and a flash file system can ensure that a flash-based design utilizes all of the capabilities of the flash device in the most efficient manner possible.

Intel has a long history in flash memory software and has been designing flash file systems for over 15 years and has well over 50 flash software patents. This commitment has been recognized by flash system designers who have installed Intel's flash file systems on more than 300 million devices worldwide.