



## **Designing Fail-Safe Storage for Embedded Systems**

*By Dave Hughes, Chief Software Architect, HCC-Embedded*

### **Introduction**

In recent years it has become much easier to store large amounts of data even on small embedded systems, with help from the continuous advances in flash storage that consistently challenge Moore's law. For many developers of embedded systems the data they store and manipulate are extremely valuable. To design a system where data are stored in so that it is never lost or damaged, and is always maintained in a consistent state, is a complex task, and requires much thought and innovation.

HCC-Embedded has spent significant time and effort to analyze, design and test reliable file system solutions for embedded devices. This article investigates the design of complete storage solutions with particular reference to HDDs, SSDs, SD and CF cards, and raw flash.

### **File System Structure**

A standard model of media storage is shown in the diagram below. It is important to understand this layered approach when aiming for a reliable system. The reliability requirements reach from the top layer to the bottom layer: The application requires an adequate level of service from the file system; the file system requires an adequate level of service from the drivers; the drivers require an adequate level of service from the storage media. Without sufficient reliability in any of the three levels, failure becomes an option. And these levels of service need to be clearly defined, because it clearly makes no sense to claim that a file system is fail-safe if the storage devices are unreliable.

*Each layer of the system needs to understand what fail-safety is for that layer and what it requires of other layers.*

### **Application Layer Requirements**

Let's start by defining the requirements that an application developer requires for fail-safe storage. First, we need to know precisely what fail-safe means. At HCC-Embedded this boils down to two main features:

1. Any time a file is written to the system the action of closing or flushing the file will atomically switch its state from the previous consistent state to the new consistent state. Effectively, this leaves the decision as to what is consistent to the application developer.
2. In the event of unexpected reset of the system at any point while operating, when the system restarts all elements of the file system will be coherent and all files will be in their pre-restart consistent state.

This point of consistency needs to be emphasized, because in any file system you need to determine when the new added data are valid, and therefore it is time to update the related meta-data. All writes to the file system remain uncommitted to the storage media until a close or flush is issued to signal that the new state should be made valid. In this way the developer can determine what a consistent state for the data is. This does not mean that the data are not written to the target media; it does mean that the meta-data of the file still refer to the previously chosen consistent state until you decide otherwise. It also means that if you seek back in a file and overwrite something, then an original copy must be maintained. This ensures that, in case an unexpected event occurs, the original state of the file is valid and consistent.

*This rather straightforward description of fail-safety is the essence of a real, workable approach to building failsafe file systems that are truly reliable.*

### **File System Layer Requirements**

HCC-Embedded's file systems are designed to achieve the fail-safe requirements defined above, but only if the underlying drivers and physical media meet certain requirements:

1. The driver must write the data in the same sequence as it is received. This rule enforces the proper sequence even when there is a cache, because there is no possibility that a sector will be written when a previously written sector has not been written.
2. In the event of a physical reset or power loss, any sector of data must be either written, or the old sector contents must remain valid. There should be no intermediate state in which the contents of a sector are effectively random. The switch from the sector's old state to new state must be atomic.

It is possible to design systems with different requirements, but they must be clearly described and understood to ensure that the system will meet the failsafe design goal.

*These seemingly simple requirements are not easy to achieve because of the design of storage devices.*

### **Power Handling**

A stable power supply and a clean reset of power are mandatory, particularly when using systems incorporating flash memory. It is essential to have a brown-out detection that resets a flash card when power drops below a specified level. Without power at an acceptable level, reliable writing of data cannot be guaranteed. Various types of problems can occur. For example, a write may succeed when a previous one failed or a write or erase fails, thus creating bad blocks in the system.

Several available flash-based cards experience complete and permanent failure if you slowly drop the power to them while writing. Others include internal brown-out detection to avoid this.

An obvious solution for most of these issues is to provide reliable power to the device. This is not necessarily as simple as it first appears. Data on device behavior during power failure is often difficult to obtain from manufacturers. As a result, the worst-case time required to store all outstanding data may be difficult to ascertain. For some manufacturers' devices, such data are available, and thus it is possible to design to the published specifications. But, although there's a workable solution for these cases, the choices become somewhat limited.

*Unreliable power will undermine an otherwise failsafe file system.*

## **Storage Types**

Very few manufacturers, with only a few notable exceptions, state clearly the operating characteristics of the storage devices in terms that are useful to a designer of a genuinely failsafe system.

HCC has applied rigorous tests to many commercially available solid state memory cards. The vast majority have been shown to exhibit non-safe behavior. HCC classified only a small number of cards that are acceptable for use with failsafe file systems.

### Hard Disk Drives (HDDs)

These are uniquely difficult devices to use in reliable systems. The primary problem is that they all contain an integrated data cache that is generally difficult to control. If the cache were easy to control, doing so would decimate the disk performance. For all of the drives we have investigated, we have not found one that will clearly state worst case scenarios at power-off and how much time and power are required to ensure that the contents of the cache are safely written. If the safe flushing of the cache cannot be guaranteed, then you have worst case scenarios in which large amounts of data are lost, and data that are written later could be present.

### Flash Cards and SSDs

SD cards, Compact Flash cards and Solid State Drives all have similar design characteristics. They contain arrays of NAND flash that are managed by a flash translation layer (FTL), which provides a logical sector interface to the external system. The operation of FTLs is a closely guarded secret by most manufacturers, but it is clear that, to get maximum performance, many devices employ caching and parallel writing techniques. The result is that, in the event of unexpected reset, sectors can be written out of sequence.

### Flash

Integrated flash has a major advantage: The system design is then entirely under the developers' control. However, care must still be taken. Looking again at the diagram if flash is to be interfaced to the system then some kind of flash management layer is required between the file system and the flash. This is typically a flash translation layer, which is incorporated into SSD, SD card and Compact Flash card. It is imperative that the FTL be designed to be failsafe and also that power is properly controlled.

*All media have their own peculiarities and weaknesses. However, with proper understanding of a medium's behavior, which can be very complex and subtle, failsafe systems can be built.*

## Causes of Failure

What are the primary conditions that can cause a file system to be damaged?

Firstly it is useful to note that this can happen only when a write or erase operation is being performed on the target device. However, this could also be the result of a user operation or of a management function such as background erase or wear-leveling.

The two primary events which can cause file system failure are:

1. unexpected reset of the system because of power loss or other hardware condition
2. unexpected reset of the system because of a bug

For example a HDD typically contains a large internal cache (8 Mbytes or more), so if the disk loses power unexpectedly, then the file system may lose data in the cache, and much data could be written out of sequence. A file system claiming fail-safety really needs to specify what it can tolerate and still remain failsafe, and then the design needs to reflect that.

## Testing

To ensure the failsafe characteristics of a file system, and also to understand the issues that arise, HCC has carried out extensive testing in two main forms:

1. **Simulation.** The entire file system and the drivers can be tested in simulation on a PC. Millions of iterations of failure scenarios can be generated very quickly and it is possible to verify the design of the file system.
2. **On an Embedded System.** Simulation of course does not test the system with the target media, and as illustrated above it is crucially important to ensure that it provides the quality of service required. This is particularly true when device manufacturer do not provide detailed information about how their storage devices operate. This testing is done by using an external circuit to randomly reset the target. On each reboot a verification program checks to see if the contents of the system have been damaged.

Intensive testing of many SD cards has shown that very few meet the requirements of atomicity and sequential writing, and as such are not suitable when data are important.

***Conclusion.** With adequate understanding of storage devices and other hardware characteristics, and rigorous testing of all failure modes, it is possible to create truly failsafe, high-performance file systems that work well with all of the common storage media.*

Further information about designing fail-safe embedded storage solutions can be found at [www.hcc-embedded.com](http://www.hcc-embedded.com).

