Data Management Software (DMS) for AMD Simultaneous Read/Write Flash Memory Devices

Technology Background

July 2003

The following document refers to Spansion memory products that are now offered by both Advanced Micro Devices and Fujitsu. Although the document is marked with the name of the company that originally developed the specification, these products will be offered to customers of both AMD and Fujitsu.

Continuity of Specifications

There is no change to this document as a result of offering the device as a Spansion product. Any changes that have been made are the result of normal documentation improvements and are noted in the document revision summary, where supported. Future routine revisions will occur when appropriate, and changes will be noted in a revision summary.

Continuity of Ordering Part Numbers

AMD and Fujitsu continue to support existing part numbers beginning with "Am" and "MBM". To order these products, please use only the Ordering Part Numbers listed in this document.

For More Information

Please contact your local AMD or Fujitsu sales office for additional information about Spansion memory solutions.

Publication Number **22274** Revision **A** Amendment **0** Issue Date **November 1, 1998**

TECHNOLOGY BACKGROUND

Data Management Software (DMS) for AMD Simultaneous Read/Write Flash Memory Devices

AMD

Introduction

A wide variety of applications use Flash memory to store embedded control code. Most of these applications (including cellular phones, modems, and automobile engine control) have traditionally utilized an EEPROM to store factory, system, and/or user data. Replacing EEPROM with Flash memory simplifies hardware and software design, reduces board space requirements, and decreases system cost. Removing EEPROM can also increase system performance since Flash memory is both faster and more reliable than EEPROM.

Replacing an EEPROM with Flash memory presents a new set of problems, however. First, complex task management software will usually be needed, since most Flash devices can only perform one operation at a time. Secondly, some form of data management software must be developed to manipulate data within the sectors of Flash memory devices.

Task Managers

When storing code and data in a traditional flash device, the system cannot read operating code while data is being written or erased. This can cause unacceptable delays in system operation while waiting for the write to complete. To get around this limitation requires complex task management software. This software must constantly monitor priorities of operations, to suspend lower priority commands (such as erase and write) and activate higher priority ones (like read system code). While this software solution enables EEPROM replacement, it is very difficult to integrate the task management software with system software. This software solution also results in significant system overhead that impacts system performance, especially in real time applications.

The Am29DLxxx family of Simultaneous Read/Write Flash provides a unique performance advantage over other flash products. The device can read data while it is performing a write or erase operation. This simultaneous operation eliminates the need for complicated task managers, resulting in simpler software and increased performance. However, data management software is still required to manage data storage.

Data Management Software

AMD provides Data Management Software (DMS) to work together with Simultaneous Read-Write flash devices to make it easy for customers to store code and data in a single flash device[1]. DMS takes advantage of the Simultaneous Read/Write devices by storing the system software (including DMS) in one bank of the flash, and storing data in the other bank. Partitioning the memory this way allows code to be read out of one bank while the system updates and manages data in the other. Systems that use a traditional flash device must copy code to RAM to run the system while writing data to the flash memory.

DMS provides all the tools necessary to update data in a flash device. The system software only has to call one of seven functions to utilize DMS. Because the system only needs to access these few functions, DMS requires as few as 20 hours of integration time. Other solutions based on traditional flash devices require a minimum of 500 hours.

DMS stores and tracks data as virtual cells and blocks within the physical boundaries of the flash bank. Since a byte in flash may not be overwritten, an old occurrence of data is marked “dirty” when the data is updated. DMS continues to store data until there is not enough “clean” space in the bank to write new records. At this point, DMS initiates a cleanup process, saving the latest valid occurrence of each data in another sector, then erasing the old dirty sector. System software can be greatly simplified because it doesn’t have to perform this complex bookkeeping. DMS software even handles variable length parameter storage, which allows for data streaming applications like voice recording. With the introduction of DMS, AMD offers a simple and complete solution for data storage.

1. DMS also supports non-simultaneous devices if customers only want to utilize its data management capabilities. In this case, DMS must execute from an external memory device other than the flash device storing the data, and will require a user written task manager.

DMS Overview

DMS is broken up into four layers: the application layer, the cell layer, the block layer, and the device layer. These layers interact with each other, maintaining a minimum level of coupling and a maximum level of data abstraction. This simplifies the integration process by requiring the system software to only access one layer of DMS, hiding all complex data management tasks in the other layers of DMS. (See Figure 1.)

The **application interface layer** handles all communication between the DMS library and the user's application. From an object oriented perspective this is the public interface to the DMS object. A vendor or integrator can perform all necessary DMS operations by calling the routines in this layer.

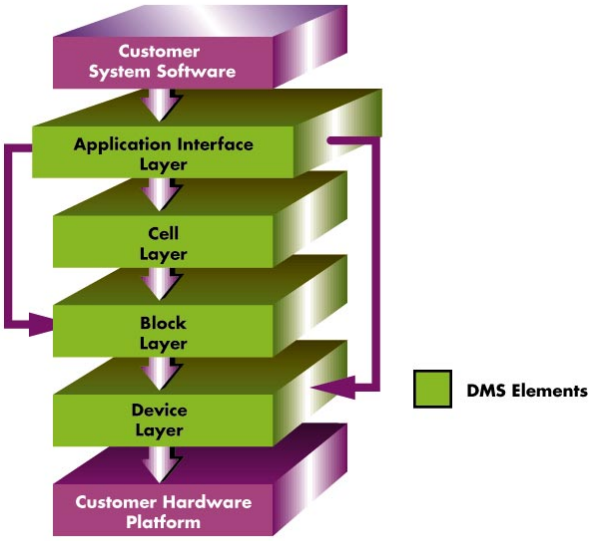

Figure 1. DMS Block Diagram

The **cell layer** provides an interface to DMS that is similar to an EEPROM. Each cell is composed of one or more data blocks. The data blocks are read from the data block layer and combined to form a cell. To prevent cell corruption, a compile time switch is provided that assures cell integrity at the expense of restrictions on cell size and free space. If the compile time cell integrity switch is not utilized, cell data may be corrupted during a power outage. For example, if power was lost during a cell update operation, the cell may be corrupted. Half of the blocks in the cell may contain new data, while the other half contain old data. While these blocks all contain valid data, this hybrid cell does not contain valid cell data. (See Figure 2). When re-initialized (cell integrity switch active), DMS detects the corrupted cell and reverts all data in the cell to old cell data.

The **data block layer** is responsible for most of the file structure functionality of DMS. An ID number uniquely identifies each data block. Cells consist of data blocks linked by ID numbers. To increase performance, data block information is sorted upon initialization and cached in an external memory table. If power is terminated while a data block is being written, the data block will revert to the previous version of itself. To provide wear leveling of the device, each data block is moved to a different location when written.

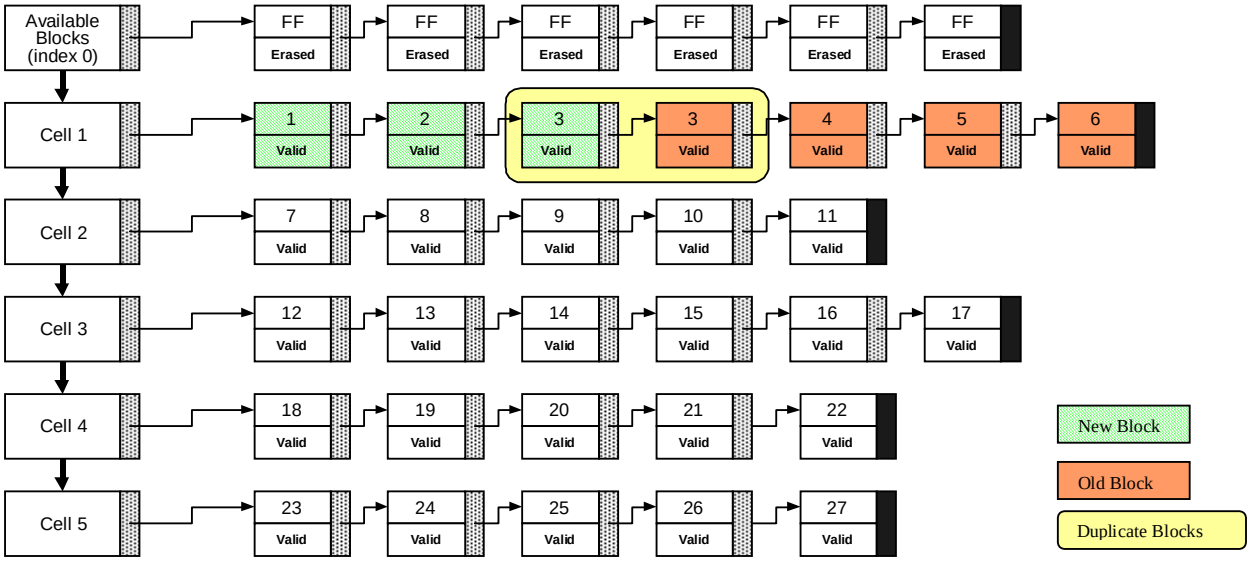

Figure 2. Corrupt Global Cell Table

The **device layer** of DMS encapsulates the device interface, providing a generic interface that does not depend on a specific processor or operating system. This approach allows other layers to remain unchanged from one environment to another. Platform specific information, such as memory mapping, is handled by a small sub-set of the device layer, the vendor specific module. The vendor specific module is unique to each DMS platform.

The functions that reside in the vendor specific module integrate DMS and the hardware platform. Therefore these are the only functions in DMS not necessarily written in ANSI C. Each time DMS is ported to a new platform these functions need to be modified. Because this module provides for rigid hardware abstraction, the remainder of DMS is not platform dependent and thus is written in ANSI C.

The seven functions in the top layer (Application Interface Layer) are the only functions that the system software needs to call. This layer abstracts the DMS into seven functions: Format, Initialize, Write, Cleanup, Read, IsBusy, and Shutdown.

Format

Format allows users to choose which sectors to manage with DMS. It then divides those allocated sectors into cells, and in turn partitions those cells into blocks. The Format function must be called before using DMS the first time, since it prepares the flash device for use by DMS. It formats the areas of the flash allocated for use by DMS into the device layout specified by the user. (See Figure 3.) First, Format erases all sectors allocated for DMS management. Then those sectors are partitioned into the cell lengths according to the cell structure the user specified. This is done on two levels, cell length and block length. In the header file, the user specifies the cell length, while the blocks are set at a default length (for example, 512 bytes). The cell is then broken up into an integer number of these blocks. When a sector is erased, Format must again be called to reformat that sector only.

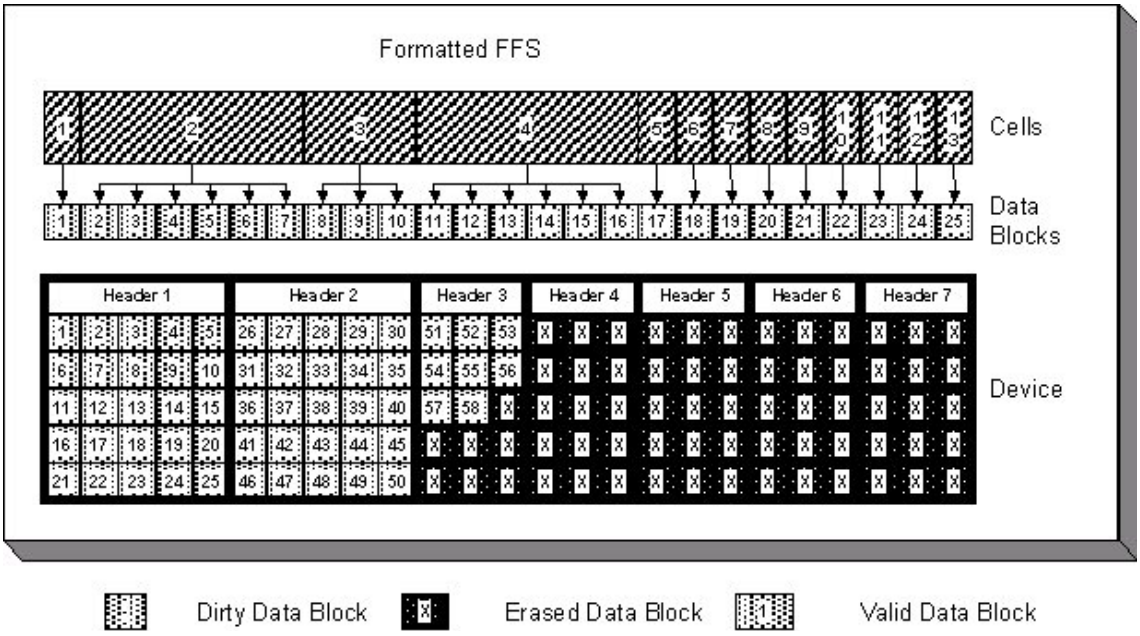

Figure 3. Formatted FFS

Cells and blocks are managed via two different means: Sector Erase header, and Global Cell Table. The Sector Erase header includes the following:

- ❑ sector index
- ❑ number of blocks that fit into the sector
- ❑ number of cells stored in the device
- ❑ number of blocks stored in the sector

Every time the global cell table is modified, the sector header is modified accordingly.

Initialize

Initialize creates the Global Cell Table based upon the cell structure defined by the user to manage data and track where data is physically located in the flash memory. Initialize should be called when booting up the device. Each cell stored in the flash has a corresponding entry in the Global Cell Table. Each entry consists of a number of nodes, which contain information about the blocks that constitute the cell. This information (unique block number, status of block (valid, invalid), and a pointer to the next block in the cell) is represented below in the global cell table, where each numbered block is a node. The nodes are added to the correct index (based on the Cell Number) in a sorted order based on the data block number within the node.

After all of the cells are defined and allocated in the flash, an integrity check is performed to ensure that the Global Cell table only points to valid data.

Write

The **Write** function writes data to a cell, after format and initialize are complete. Data is copied from the user buffer and stored in the flash device. This function will only return to the calling program after the write completes.

System software must send two parameters each time it calls the Write function. The first is the cell number to be written. The second parameter is a pointer to the current location of the data the user wants written to the device. Based upon the cell number sent, DMS will modify or add an entry in the Global Cell Table. DMS then calls the Cleanup function to ensure there is enough space available to write the entire cell without a cleanup having to occur during the write operation. Once this is ensured, DMS traces through the data to be copied, then writes this data block by block to the flash. If an existing cell is being updated, the status flag for each node in the Global Cell Table is updated with the new location of the data. If a new cell is being added, additional entries and nodes are added to the Global Cell Table.

Cleanup

As mentioned above, the **Cleanup** function defragments the sectors managed by DMS. First, it identifies the sector with the most dirty blocks. Next, all valid blocks in this sector are moved to another sector in the device. Finally, the entire “dirty” sector is erased and automatically re-formatted.

System code may call this function directly if it determines that it is in an idle state and would like to take care of the Cleanup and garbage collection process. This reduces the possibility of the user waiting for this cleanup to occur before data is written, causing an unacceptable delay.

Read

The **Read** operation reads the contents of a specific cell and stores the data in the user buffer. The cell number is used to index the Global Cell Table. DMS traces through that entry in the Table, node by node, and will output the data into the user buffer block by block, until the entire cell has been read.

IsBusy

IsBusy checks to see if the flash device is currently performing an operation. This is accomplished through the standard method of polling the DQ7 - Data Polling bit.

Shutdown

Shutdown deallocates static memory used by DMS. It should be called when shutting off the DMS functionality.

TECHNOLOGY BACKGROUND

One AMD Place
P.O. Box 3453
Sunnyvale,
California 94088-3453
(408) 732-2400
(800) 538-8450
TWX: 910-339-9280
TELEX: 34-6306

APPLICATIONS HOTLINE &
LITERATURE ORDERING
USA (408) 749-5703
JAPAN (03) 3346-7600
UK & EUROPE 44-(0)256-811101
TOLL FREE
USA (800) 222-9323
FRANCE 0590-8621
GERMANY 0130-8138575
ITALY 1678-77224

http://www.amd.com

22274A