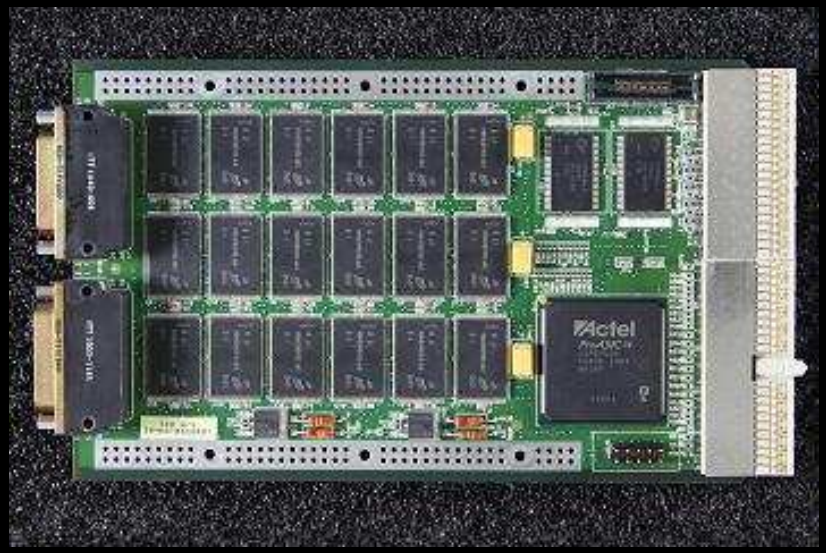
Integration of a Flash File System with VxWorks® and RTEMS

Robert Klar, Sue Baldor, and Allison Bertrand
Southwest Research Institute

Motivation: Mass Memory Board

- A high-capacity Mass Memory Module (MMM) was designed for the Magnetospheric Mutiscale (MMS) Mission
  - 128 Gigabytes of Flash
  - 1 MB of SRAM and EEPROM
  - 6U cPCI Form Factor
- The MMM was designed to support a hybrid CCSDS File-Delivery Protocol implementation
  - Does not use a File System
- Multi-Mission Mass Memory (M4) - smaller form factor board was designed to support multi-mission applications
  - 48 Gigabytes of Flash
  - 1 MB of C-RAM
  - 3U cPCI Form Factor
- An EM version w/o stacked memory was created
  - 6 Gigabytes of Flash
  - 128KB of C-RAM
- A lightweight file system is desirable to support a wider variety of storage applications



Software Goals/Requirements

- Make efficient use of the Mass Memory Board
- Provide wear-leveling of the Flash in order to maintain storage capacity for missions of a few years duration
- Provide support for file naming and directory structures
- Support common embedded operating systems such VxWorks® and RTEMS
- Require only a small amount of processor memory
- Provide for a responsive system
- Provide software which can be quickly made available to customers and put to use
  - Both TrueFFS and YAFFS have additional third party licensing requirements for use in commercial products

Reason for Wear-Leveling

- NAND Flash is a limited-life component
  - NAND Flash is rated at ~100,000 erase cycles
    - Flight-qualified Flash is derated to ~10,000 erase cycles
- Traditional File Allocation Table (FAT) File Systems were not designed for use with Flash devices
  - FAT is stored in a fixed physical location resulting in frequent updates to the same physical erase block
    - A conventional FAT file system would exceed limit very quickly without wear-leveling
    - For example, for an 8MB file, using a Cluster size of 8K, the erase limit would be exceeded on the FAT after only 10 writes of the file

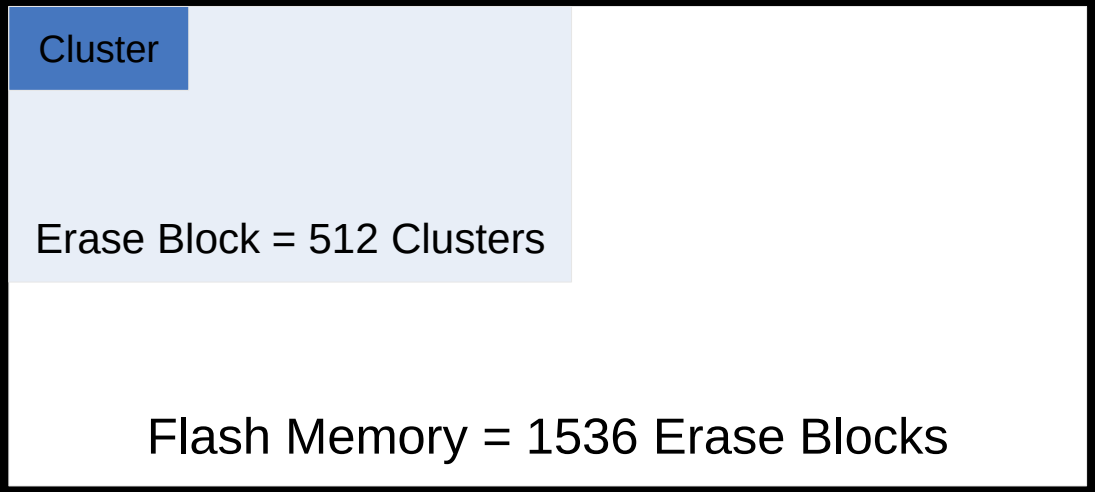
      8 MB / 8K = 1024 writes to the FAT per file write

      10 * 1024 = 10240 writes to FAT for 10 writes to file

Where to do Wear-Leveling

- Wear-leveling can be accomplished at several levels:
  - Hardware Controller
    - Off-the-shelf USB Flash Drives commonly use this approach. This allows the use of file systems not specifically designed for Flash.
  - Device Driver
    - This offers some of the same benefits as a hardware controller but adds a little complexity to the device driver. It has the advantage that we can make use of some of the features provided by the hardware design.
    - This is the alternative we chose!
  - File System
    - This also works well but may bring along some additional overhead. Configuring a file system to make use of your particular Flash device requires some work.

Data Block Terms and Sizes



- Sector = 512 bytes (smaller than smallest writable unit)
- Cluster = Flash Write Page = 16 Sectors = 8192 bytes
- Erase Block = 512 Clusters = 4 MB
- Total Flash = 1536 Erase Blocks = 6 GB

Approach to Wear-Leveling

- Use conventional FAT32 but relocate the FAT
- Do Logical to Physical Mapping
  - As FAT is updated, move portions of the table in order to evenly distribute erase cycles
- Take advantage of hardware features
  - M4 keeps some File Management Data in C-RAM for each Erase Block
    - Number of Erase Cycles
    - Bad Block Vector
    - Timestamp (managed by software)
    - Cluster State Counts (managed by software)
      - ERASED, USED, or INVALID
- Do Garbage Collection
  - Reclaim blocks when majority of pages are no longer used

VxWorks® and RTEMS File Systems

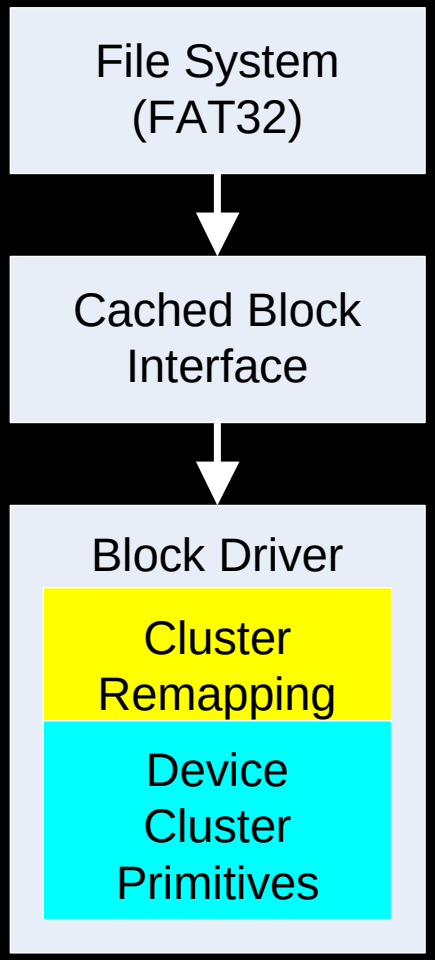
- VxWorks® has traditionally provided support for a variety of File Systems
  - RT-11 (deprecated)
  - DOSFS
  - Highly Reliable File System
  - TrueFFS (optional)
- RTEMS includes support for several simple File Systems
  - In-Memory File System (IMFS)
  - Mini-IMFS
  - MSDOS File System
  - RTEMS File System (RFS)

Structure of FAT File System

- Common FAT File Systems are FAT12, FAT16, and FAT32
  - FAT32 is the newest of these and includes support for the largest number of Clusters
  - FAT12 and FAT16 are not suitable for large capacities
- Smallest organizational unit is a Sector = 512 bytes
- First sector on media is the Boot Sector
  - This sector contains information about how the media is formatted
  - The Boot Sector includes the number of Sectors per Cluster
- Following the Boot Sector are the File Allocation Tables
  - Allocation unit is a Cluster
- Following the File Allocation Tables is the Root Directory
  - Root Directory can be located anywhere in data area in FAT32

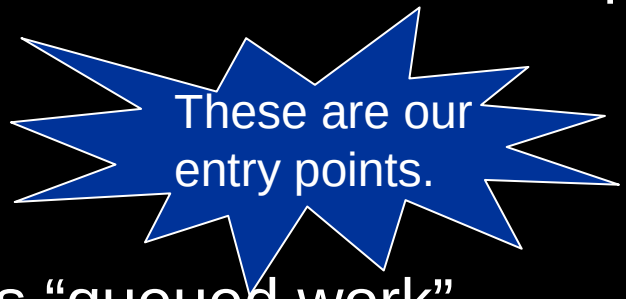
A Cached Block Driver

- VxWorks® 5.5 included a Cached Block I/O (CBIO) interface to dosFS
- VxWorks 6.x added features and changed the interface to use the eXtended Block Device (XBD) Framework
- RTEMS provides cached support for the MSDOS File System through the Block Device Library (libblock)



A Cached Block Driver

- Device Cluster Primitives
  - Invoked by Operating System File System
  - VxWorks 5.5 and RTEMS 4.10 have similar primitives
    - Read Block
    - Write Block
    - Verify Block (RTEMS)
    - I/O Control (VxWorks)
  - VxWorks 6.x introduces “queued work”
    - A device driver task operates on a queue of structures
  - VxWorks 6.x has different primitives
    - xf_ioctl - provide interface for miscellaneous control functions
    - xf_strategy - queue work and wakeup device driver task
    - xf_dump - provide fast way to write data in event of a fault

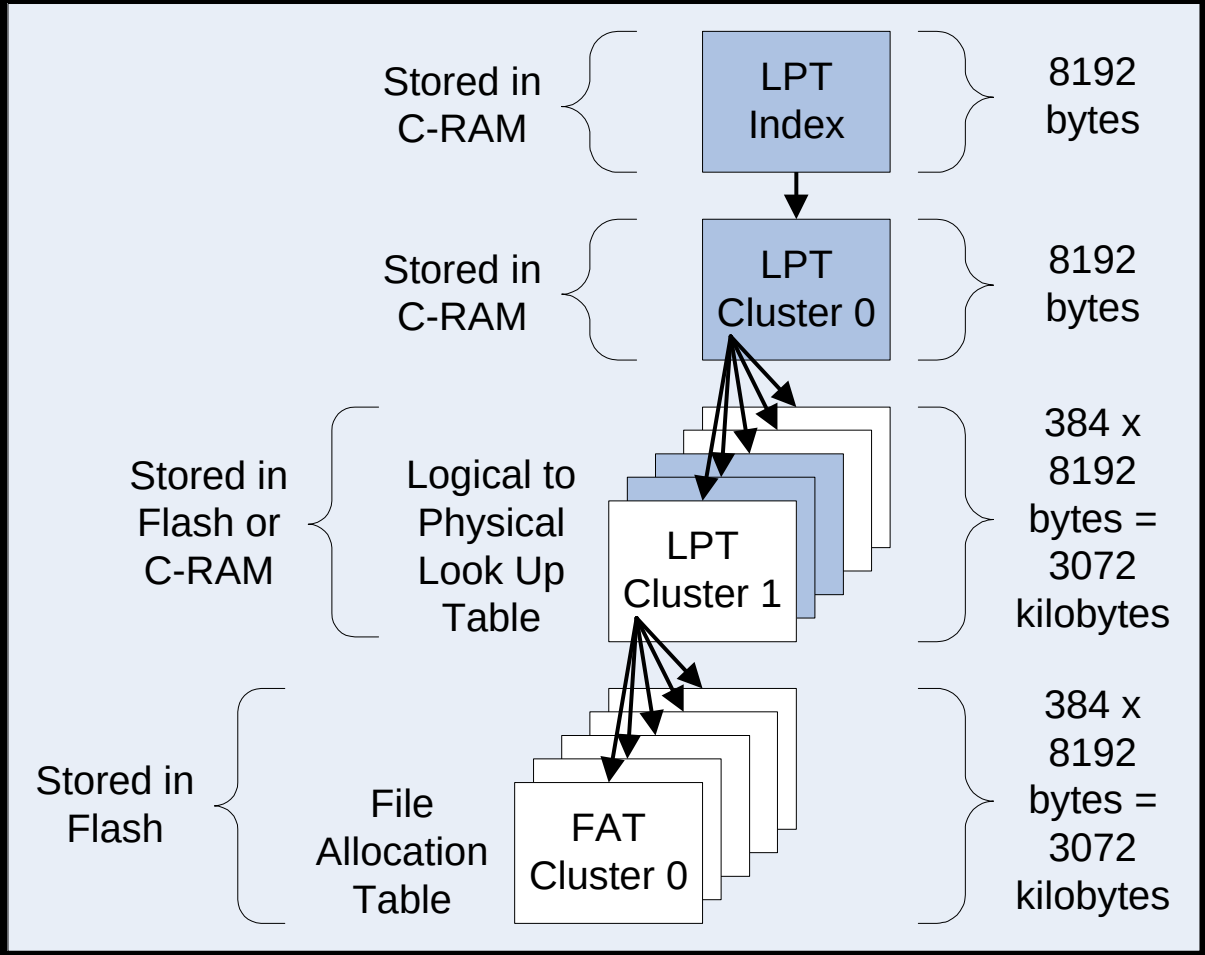


Initialization Steps

- VxWorks 5.5 Initialization
  - Create Device Descriptor
    - m4DevCreate()
  - Create Volume Configuration Structure
    - dosFsConfigInit()
  - Create Volume Descriptors
    - dosFsDevInit()
- RTEMS 4.10 Initialization
  - Setup Device Descriptor
  - Setup Configuration Structure
  - Setup Block Device I/O Ops Structure
  - Register the driver with the I/O system

Data Structures

- File Allocation Table
  - Stored in Flash
  - Relocated to Data Area Blocks
- Logical to Physical Look Up Table
  - Stored in Flash
    - Plan to eventually use C-RAM for caching
  - Map Logical Cluster to Physical Cluster



Data Structures / Cluster Selection Heaps

- Two small heaps are kept in processor RAM
  - Each is less than the total number of Erase Blocks (1536)
  - A *Free Cluster Heap* is used to prioritize Erase Blocks which contain some ERASED clusters by fewest number of Erase Cycles
    - Next physical cluster is at the top of the heap
    - High priority is a low number
  - A *Reclamation Cluster Heap* is used to prioritize Erase Blocks which do not contain ERASED clusters by fewest number of (Erase Cycles, Number of USED Clusters)
    - Number of Erase Cycles is periodically compared to top of Free Cluster Heap to determine need for Garbage Collection
    - High priority is a low number

Data Structures

- Cluster Map shares memory with LPT
  - Uses most-significant bits of each 32-bit word
  - 2 bits indicate Cluster State
    - ERASED, USED, INVALID
  - 2 spares – possibly could be used to add C-RAM caching
  - This is OK because FAT32 only uses 28 bits to represent a Cluster
  - Cluster Map represents Physical Cluster rather than Logical
- C-RAM is used for several purposes
  - Keep index to LPT 0 Cluster so that we can find it at initialization
    - Index is rotated through circular buffer in C-RAM to avoid wearing C-RAM word
    - Location of pointer is kept in RAM but must be discovered at initialization
  - Count number of writes to same LPT Cluster
    - Avoid moving Clusters if not necessary

Writing to Flash

- File System invokes Driver Write Block routine to write a Logical Cluster
  - May be a Data Cluster or a FAT Cluster
- The Driver looks up the Logical Cluster in the LPT. If a Physical Cluster is mapped already, then it is marked as INVALID in the Cluster Map. USED and INVALID counters are updated in the Erase Block.
- The Driver gets the next available ERASED Physical Cluster by looking at the Erase Block at top of the Free Cluster Heap. The Physical Cluster is marked USED in the Cluster Map. The corresponding USED and INVALID counters are updated in the Erase Block. If this was the last ERASED Physical Cluster in the Erase Block, then the Erase Block is removed and put on the Reclamation Cluster Heap.
  - The corresponding Cluster Map Cluster is updated (and moved if necessary, overwriting an ERASED block).
  - The corresponding LPT Cluster is updated (and moved if necessary, overwriting an ERASED block).

Garbage Collection

- Some Erase Blocks will contain data that seldom changes. Over time this results in a disparity in the number of erase cycles between these blocks and others.
  - To avoid this, Erase Blocks which contain only USED Clusters but that have few erase cycles will be periodically moved so that these become available to the Cluster Selection Algorithm
- Currently done inline in Write Block

Worst-Case Performance

- For each Data Cluster write there are additional writes to update data structures
  - Up to 5 Cluster writes to Flash
    - 2 writes to corresponding LPT Cluster (for Data Cluster) and Cluster Map
      - writes will depend on frequency of accessing same LPT Cluster
      - up to 4 writes if just writing 0's
    - 1 write to corresponding FAT Cluster
      - Cluster must be moved if Logical Cluster is replaced
      - up to 4 writes if just writing 0's
    - 2 writes to corresponding LPT Cluster (for FAT Cluster) and Cluster Map
      - writes will depend on frequency of accessing same LPT Cluster
      - up to 4 writes if just writing 0's
  - Up to 3 Cluster writes to C-RAM
    - 1 write to count writes to LPT
    - 1 write to new LPT pointer
      - same word accessed every 8192 times LPT 0 moves
    - 1 Write to set value of previous LPT pointer to INVALID
      - same word accessed every 8192 times LPT 0 moves

Next Steps

- Potential Optimizations
  - Use C-RAM to cache Clusters for LPT, Cluster Map and FAT
    - Reduce the number of Flash writes for moving frequently used Logical Clusters
  - Background Garbage Collection
    - Low priority task
  - Use block driver with a different File System
- Next Steps
  - Testing with VxWorks 5.5 and RTEMS 4.10 over next few months
    - Compute Average Performance with operational scenario
  - Testing with VxWorks 6.x

Questions?