A FAULT-TOLERANT RADIATION-ROBUST MASS STORAGE CONCEPT FOR HIGHLY SCALED FLASH MEMORY

Christian M. Fuchs\textsuperscript{1,2}, Carsten Trinitis\textsuperscript{1}, Nicolas Appel\textsuperscript{2}, and Martin Langer\textsuperscript{2}

\textsuperscript{1} Chair for Computer Architecture and Organization; \textsuperscript{2} Institute for Astronautics
Technical University Munich, Boltzmannstr. 15, 85748 Garching, Germany
\{christian.fuchs, carsten.trinitis, martin.langer, nicolas.appel\}@tum.de

ABSTRACT

Future space missions will require vast amounts of data to be stored and processed aboard spacecraft. While satisfying operational mission requirements, storage systems must guarantee data integrity and recover damaged data throughout the mission. NAND-flash memories have become popular for space-borne high performance mass memory scenarios, though future storage concepts will rely upon highly scaled flash or other memory technologies. With modern flash memory, single bit erasure coding and RAID based concepts are insufficient. Thus, a fully run-time configurable, high performance, dependable storage concept, requiring a minimal set of logic or software. The solution is based on composite erasure coding and can be adjusted for altered mission duration or changing environmental conditions.

1. INTRODUCTION

Scientific and future commercial space missions as well as miniaturized satellites impose increasing demands on their on-board computer (OBC) systems, especially data storage devices [1]. They may require vast amounts of data to be stored and processed aboard the spacecraft, high throughput, and the possibility for concurrent access of multiple threads, programs or devices. While satisfying these requirements, storage systems must guarantee data integrity and the recovery of degraded or damaged data (error detection and correction – EDAC) over a prolonged period of time in a hostile environment. Consistent data storage becomes even more crucial for long-term missions (e.g. JUICE [2] and Euclid [3]) or in cases where highly scaled memory is used.

Legacy memory technologies can not be scaled for modern storage applications due to mass and energy restrictions or result in high complex storage systems. Thus, single-level cell NAND-flash memories (SLC), have become popular for high performance mass memory scenarios as they offer reasonably high packing density, and can be manufactured sufficiently radiation hardened. The chip-industry has moved on from SLC to multi-level cell flash memories (MLC) due to economical reasons. Therefore, SLC will become unavailable and will force future spacecraft storage concepts to rely upon MLC or entirely different memory technologies. While there are promising candidates [4] to fill this role in the long run, technological evolution does not yet allow, for example, non-volatile magnetoresistive RAM (MRAM) to be used as mass storage [5]. Phase change [6] or charge-trap based memory both would at present be usable as mass storage, but are not yet widely available in high density versions [7].

Traditionally, single-bit error correction, shielding, specialized manufacturing techniques, coarse structure width and redundancy are combined to enable radiation tolerant flash [8]. However, the protective level offered by such solutions is static and fixed at design-time and can result in high cost and low overall efficiency. For miniaturized satellites, cost and efficiency are crucial, thus, countermeasures must be implemented at a different level. With modern MLC-flash single bit error correction is insufficient and all-in-one solutions, such as file systems, tend to become very complex and difficult to debug. For future prolonged missions and larger storage arrays, more sophisticated and efficient EDAC concepts are required. Thus, we present an advanced high performance dependable storage concept based on composite erasure coding. As MLC-flash is also widely used aboard miniaturized satellites, and the authors are involved in developing such a satellite, MOVE-II, development was originally driven by nanosatellite requirements. However, the approach can be applied more efficiently to commercial applications where miniaturization imposed limitations do not apply. The concept could be implemented even more efficiently with very large volumes common in commercial spaceflight applications. It can be implemented entirely in software, with or without hardware acceleration, but also partially or fully in hardware.

In the next section, the reasons for MLC and SLC flash requiring different fault detection, isolation, and recovery (FDIR) strategies will be discussed. We outline why traditional rad-hardening is insufficient for MLC and argue that voting and simple RAID-based solutions and block-
2. FLASH MEMORY

Each flash memory cell contains a single field effect transistor with an additional floating gate, see Fig. 3 or within a trapping layer like within EEPROMs. Voltage applied between source and drain generates an electric field with a conductive channel through which electrons can flow from source to drain. When applying voltage to the gate, electrons are pumped into the floating gate. The state of a cell is thus dependent on whether or not a specific threshold voltage $V_t$ is exceeded.

2.1. Single- and Multi-Level Cell Flash

Single Level Cell Flash (SLC) cells can store one bit per cell. If the voltage exceeds the threshold, a cell can be read as programmed (0), else as erased (1), see Fig. 1a. An MLC gate can be used to store multiple values using additional thresholds, see Fig. 1b. Assuming a four level cell, one can hold four states and represent two bits. The number of levels is not restricted to four, with $2^n$ states it is possible to encode $n$ bits, but electrical complexity grows and the required read sensitivity and write specificity increase with the number of bits represented. On the same piece of silicon, MLC thus allows a much higher packing density.

As the delta between levels decreases, increased precision is required for sensing and charge-placement on the floating gate. MLC memory is thus heavily dependent on its cells’ ability to retain charge. In contrast to SLC, a state machine is required for addressing MLC memory which in turn increases latency and adds considerable overhead logic. Addressing in MLC flash can thus take multiple cycles and the state machine may hang or introduce arbitrary delays.

Due to a shifting voltage threshold in floating gate cells caused by the total ionizing dose, MLC flash memories are more susceptible to bit errors than SLC [9]. Also, such highly scaled flash memories are rather prone to single event upsets (SEU) causing shifts in the thresh-

Figure 1. The voltage reference and thresholds of SLC- (a) and MLC cells with 4 (b) and 8 levels (c).

Figure 2. Example-Upsets encountered within 4- (a) and 8-level (b) cells.
NAND-flash memories are organized in blocks, in which cells are connected as NAND gates. The drawback here is that if a NAND-flash cell fails, the entire NAND gate will fail. In NOR-flash, cells form NOR gates, which allow more fine grained read access at the cost of strongly increased wiring and controller overhead. Therefore, in order to appropriately handle NAND-flash block corruption, an FS and the FTL must handle read and erase abstraction, as well as basic block FDIR. When data is written to flash memories, partial writes are impossible and the entire block’s previous content first has to be read and updated in RAM. Next, the block must be erased (by draining the block’s cells’ voltage) and may subsequently be programmed anew. Thus, read and write operations introduce different latency and make access to MLC flash much more complicated than to SLC due to the addressing state machine.

To access data and handle special properties of flash efficiently, an FS has to interact with the memory device directly or via the OS’s FTL. A flash FS must implement all functionality necessary to perform block wear leveling, read and erase block abstraction, bad-block relocation and garbage collection (depicted in blue in Fig. 4) to prevent premature degradation and failure of a bank. The FTL acts as an interface between hardware specific device drivers and the FS, and can provide part of this FDIR functionality instead of the FS.

Over time, a flash memory bank will accumulate defective blocks and utilize spare blocks to compensate. Traditionally, simple erasure coding (usually some form of cyclic block codes with large symbol sizes) is applied in software or by the controller to counter wear and charge leakage. Eventually, the pool of spares will be depleted, in which case the FTL or FS will begin recycling less defective blocks and compensate with erasure coding only, thereby sacrificing performance to a certain degree. For space use, the erasure codes’ symbol size is usually reduced to support one or two bit correcting erasure coding, as corruption will mostly result from radiation effects. However, if this solution is applied for MLC-NAND-flash, block EDAC becomes very inefficient due to the occurrence of both single bit- and grouped errors, the latter being induced by SEUs affecting multiple cells in highly scaled memory.

### 2.3. Majority Voting for Flash Memories

While voting is technically still possible for MLC-flash, it is severely constrained by the additional circuitry, logic and strongly varying timing behavior. Voting would have to be implemented for the addressing state machine as well, otherwise it could stall the entire voting circuit or permanently disable its memory bank. Due to the varying timing behavior of NAND-flash and the more complex logic, the resulting voter-circuit thus becomes more error prone. The added logic also requires more energy and reduces overall performance. Of course the slowest memory bank or block also dictates performance of the voting circuit.

### 3. THE MTD-MIRROR MIDDLEWARE LAYER

As outlined in the previous sections, error correction is crucial for current data storage based on NAND-flash. To enable future dependable MLC-NAND-flash based data storage solutions for space flight applications, existing EDAC functionality can be adapted and improvements added where necessary. Thus, we developed a storage system to satisfy the following requirements:

1. Efficient, fast data storage on MLC mass-memory.
2. Integrity protection and error correction with adjustable strength, to allow optimization according to mission duration, environment and type.
3. Efficient handling of direct and indirect radiation effects on the memory as well as the control logic.
4. Protection against device failure.
5. Low software- and hardware complexity: While a certain level of complexity is acceptable for commercial spaceflight applications, it is crucial in microsatellite design.
6. Universal FS support and interactivity.

We consider these requirements to be met best through enhanced EDAC functionality as FTL-middleware. At this level, RAID-like features and checksumming can be combined most effectively with a composite erasure coding system. As our use case includes a Linux based OBC, we implemented MTD-mirror on the memory technology device (MTD) FTL subsystem of the Linux Kernel. The solution is depicted in Fig. 4. Any unmodified flash-aware FS can be deployed on top of the MTD-mirror set. By utilizing mirroring (RAID1) and distributed parity (RAID5/6) we can therefore protect against device, bank and block failure. Within this abstract we focus
Figure 4. Memory access hierarchy for an MTD-Mirror set. Flash-memory specific logic is depicted in blue and partially resides within the FTL. Required modifications to enable the concept are depicted in yellow.

To safeguard against permanent block defects, single event functional interrupts, radiation induced programmatic errors and logic related problems, we apply coarse symbol level erasure coding. As this is insufficient to compensate for radiation effects, silent data corruption and bit flips are compensated using bit-wise error correction. The solution was implemented in the FTL, as the required logic can still be kept abstract and device independent while it can profit significantly from hardware acceleration. The FTL middleware also provides enhanced diagnostics, as no further abstraction is introduced.

3.1. Alternative Approaches

EDAC and device independence could also be provided by an FS directly as has already been shown for MRAM in [12]. A Flash FS such as UFFS could be extended to handle multiple memory devices and EDAC, or FTRFS [12] could be modified to handle flash memory. Even though possible to implement, such an all-in-one FS would be complex and error prone.

Device independence could also be added on top of an existing flash FS as a separate layer of software [13], see Fig. 5. Within a RAIF set, increased protective requirements could be satisfied with additional redundant copies of the FS content. The underlying individual FSs would then have to handle all EDAC functionality and escalate fatal errors and unrecoverable file issues to the set, as RAIF by itself does not offer any integrity guarantees beyond FS or file failure.

Since RAIF only reads from underlying FSs, it is prone to FS-metadata corruption which can result in single block errors failing entire FSs. Additionally, Flash-FSs usually rely upon parameter-fixed block based error correction and do not offer configurable protection for different FS structures, which is at best sub-optimal for space use.

A file damaged in different locations across the set’s FSs would become unrecoverable as RAIF would discard information regarding the location of damage to a file and in the best case would forward a defective copy to the application. It would therefore inhibit error correction and may even cripple recovery of larger files. While RAIF could be adapted to handle these issues, the resulting storage architecture would again become very complex, difficult to validate and debug. As RAIF implements FS redundancy, its storage efficiency will furthermore be inferior to distributed parity concepts such as the more advanced variants of the presented concept. As a pure software layer without the possibility to interact with the devices, hardware acceleration of RAIF would be impossible.

3.2. Device & Bank Failure Protection

In contrast to RAIF, RAID can be applied efficiently to storage architectures and has been used previously aboard spacecraft (e.g., in the GAIA mission) [14]. However, these were based on SLC (see Section 2) and only relied on RAID to achieve device fail-over through data mirroring (RAID1) and distributed parity (RAID5/6) [15, 14]. As RAID itself does not offer any integrity guarantees beyond protection against read device failure, designs usually rely upon the block level hardware error correction provided by the flash memory or controller or implement simple parity only.

The main issue encountered with plain RAID setups is the absence of validation for a block or group of blocks. RAID merely retains redundant copies of data – parity – which can be used to restore lost data. RAID foresees that a data block is either unrecoverably lost (signaled by

Figure 5. Memory access hierarchy for an enhanced RAIF based concept with added FS level error correction. Same color coding applies.
Figure 7. RAID prevents EDAC and wear leveling functionality within a flash-FS from being implemented. Affected elements are shown in red.

3.3. Block-Level Consistency

MTD-mirror’s block consistency protection is depicted in Fig. 6 and includes two checksums and error encoding layers. Thus, it implements a concatenated/composite erasure code system. The data checksum allows bypassing decoding of intact data, which will often be the optimistic default case. The second checksum can be used for error-scrubbing of erasure data and prevents symbol drift of the RS-layer. Even though CRC16 could be considered sufficient for most common page and block sizes, we utilize a 32-bit checksum to further minimize collision probability at a minimal compute overhead.

3.3.1. Protection against Multi-Bit Upsets

The first layer of erasure coding is based on relatively coarse symbols and protects against data corruption induced by stray writes, controller issues and multi-bit errors. As data on NAND-flash is stored in pages and blocks of fixed length and the coding layer should protect against corruption up to 8 byte length (int64_t), Reed-Solomon (RS) erasure coding [17] was selected. We chose to rely on the RS block code as the algorithm is
well analyzed, and widely used with NAND-flash memory and in various embedded scenarios, including spacecraft. Optimized software implementations, IP-cores and hardware acceleration are available.

Erasure coding with coarse symbols is efficient if symbols are largely or entirely corrupted, but shows weak performance when compensating radiation-induced bit-rot, to which MLC is comparably prone. SEUs will be evenly distributed across the memory and will thus equally degrade all data of a code word, corrupting multiple code symbols with comparably few bit errors. Therefore, RS is applied at the page level, instead of the block level to allow more efficient reads and avoid access to other pages within the same block to retrieve erasure coding parity. RS parity is therefore stored within each page, together with a checksum for the page and the parity. RS encoding and decoding can be should parallelized due to the small word sizes in hardware.

3.3.2. Bit-Level Erasure Coding

Previous radiation-tolerant OBC storage concepts often relied upon convolution codes as these allow efficient single-bit error correction. However, as error-models become more complex (2-bit errors as in MLC), codes complexity increases and efficiency diminishes. Therefore, a second level of erasure coding using Low-Density Parity Check Codes (LDPC) [18] was added to counter single or double bit-flips within individual code symbols of the first level RS code. LDPC was chosen as it is efficient with very small symbol sizes (1 or two bit), offers superior performance compared to convolution codes [19], and allows iterative decoding [20]. Only if RS decoding fails, the set resorts to LDPC. LCPC can then support recovery of slightly corrupted RS-symbols and parity. Thereby otherwise unrecoverable data can be repaired by salvaging damaged symbols which can drastically increase recovery rates on radiation-degraded memories.

Although LDPC codes benefit from longer code word lengths, Morita et al. [21] show that the gain from a 4KB code to a 32KB code can be negligible. For systems where buffer memory is scarce, it may therefore be of advantage to use comparably small codes and sacrifice a bit of LDPC performance. Thus, an LDPC word size between 3 and 4KB offers solid LDPC performance without requiring very large words and thereby enable fast iterative decoding.

3.3.3. Joint Iterative Decoding using Soft-Output

Shorter code words also enable joint iterative decoding [22] using soft-output for both LDPC and RS codes. LDPC can be adjusted to output not only plain copy of the expected original code word, but can also yield the decoder’s certainty about each bit’s value. Equally, an RS decoder could be extended to handle such soft-input and could even attempt decoding of multiple variants of the same word using different values for low-certainty values.

As depicted in Fig. 8, a closed feedback-loop that inputs the soft message output R(Y) of the LDPC decoder into RS can be constructed. The system iterates between RS and LDPC decoding until either decoder can reconstruct a valid code word. To tackle the issue of the thereby variable timing behavior, the number of iterations can be limited or a timeout can be defined.

3.3.4. Error Handling Runtime Behavior

In case the checksum does not match the plain block data, an MTD-mirror set will first attempt to retrieve an intact

![Figure 8. Joint iterative decoding using LDPC soft-output with added parallelization (triple-arrows). RS encoding can be parallelized to increase write-throughput. Speculative RS-decoding could be utilized to reduce LDPC iterations by performing multiple parallel RS-decoding attempts with different values for low-certainty bits.](image-url)
copy of the data from another memory of the RAID-set. If this fails, or all other blocks are invalid as well, erasure decoding for the damaged block is attempted. As multiple copies of the erasure code parity data and checksums are available, the set can also attempt repair using fields of different blocks in the hope of obtaining a consistent combination of block-data. This behavior can allow recovery even of strongly degraded data or permanently defective blocks.

As RS hardware-acceleration is readily available in our use-case, we apply the two FEC layers in order (Fig. 6). However, the sequence can be chosen based on the individual system design, the used algorithmic parameters, the available acceleration possibilities and phase of the mission. An important aspect for this decision is the expected level of degradation of the utilized flash memory due to radiation, thus the occurrence of single bit errors. If severe bit-rot is expected or higher order density MLC is used, the LDPC-layer should be applied prior to RS decoding. Thereby, the increased probability of the second FEC layer failing to recover data is accepted in the hope of achieving a sufficiently high amount of intact code symbols.

3.4. Optimization & Future Work

In this abstract, we focused on describing a storage solution based on RAID1 for simplicity reasons. While the logic required to implement this storage solution is relatively simple, more advanced distributed parity RAID concepts offer increased mass/cost/energy efficiency due to overhead reduction. Thus, we have been working to adapt and expand MTD-mirror to benefit from such more advanced architectures.

There has been prior research on adding checksumming support to RAID5 in [15, 14], though utilizing RAID5 directly would introduce certain problematic aspects. Error correction information in RAID5C can either be stored redundantly with each block, introducing unnecessary overhead, or as single copy within the parity-block. While this would increase the net storage capacity, a single point of failure would be introduced for each block group. If the parity block was lost, the integrity of data which was protected by this block could no longer be verified. Instead, RAID5 can be applied to data and error correction information independently, only requiring one extra checksum to be stored with each block.

RAID6, however, can be implemented almost as-is, with error correction data and checksums being stored directly on the two or more parity blocks associated with each group. There are also promising concepts for utilizing erasure coding for generating parity blocks by themselves, thereby obsoleting simple hamming-distance based parity coding [23, 24]. Further research on this topic is required and may enable optimization for flash memory and radiation aspects similar to the ones described in this paper.

4. CONCLUSIONS

We presented the MTD-mirror middleware, which by utilizing FEC enabled RAID1 and checksumming, allows a highly reliable MLC storage with a minimal set of logic or software. An MTD-mirror implementation is independent of the particular memory devices and can be entirely based on COTS hardware. In contrast to existing solutions and payload storage concepts, this approach does not require or enforce design-time fixed erasure coding parameters, as these can be adjusted later on.

Compared to existing block-level protection concepts, more advanced error reporting and diagnostics options are available, as MTD-mirror is implemented within the FTL. This allows more detailed reporting of data and flash health to the FS thereby enabling better resource utilization, ultimately reducing wear. If the utilized FS was to offer metadata-FEC itself, it could implement locally optimized (stronger) protection for critical FS structures whereas MTD-mirror would guarantee integrity baseline for other data. A majority of FS-internal software-only erasure coding could thus be offloaded to reduce energy consumption and increase performance.

FTLs like Linux-MTD exist for most modern OSs or are built into the base kernel for those supporting flash memory directly. Besides API adaptions to MTD-mirror, little additional work is required for porting it to other OSs.

The presented storage solution can be implemented either completely in software, or as a hardware accelerated hybrid. Hardware-interactivity is possible for both the FS and the FTL within which MTD-mirror is implemented. Thus the protective guarantees offered are fully run-time configurable and allow the storage solution to be adjusted for altered mission duration or changed of environmental conditions.

An MTD-mirror set could also include flash chips of multiple different brands, vendors, manufacturing techniques or structural width to further increase the storage’s dependability rate. Thereby, simplicity can be maintained, error sources minimized, testability can be increased and throughput maximized.

ACKNOWLEDGMENT

The authors would like to thank G. Furano and G. Magistrati of ESTEC for their encouragement and invaluable feedback on our payload storage research. We would also like to thank G. Williamson of Maxwell Technologies and P. Lombardi of Syderal SA for their insights offered and the interesting discussions on this topic at ADCSS2014.
REFERENCES