FTRFS: A Fault-Tolerant Radiation-Robust Filesystem for Space Use

Christian M. Fuchs[1,2], Martin Langer[2], and Carsten Trinitis[1]

[1] Technical University Munich, Chair for Computer Architecture and Organization
[2] Technical University Munich, Institute for Astronautics
christian.fuchs@tum.de, martin.langer@tum.de, carsten.trinitis@tum.de

Abstract. A satellite's on-board computer must guarantee integrity and recover degraded or damaged data over the entire duration of the spacecraft's mission in an extreme, radiated environment. While redundancy and hardware-side voting can protect Magnetoresistive RAM well from device failure, more sophisticated software-side storage concepts are required if advanced operating systems are used. A combination of hardware and filesystem measures can thus drastically increase system dependability, even for missions with a very long duration. We present a novel POSIX-compatible filesystem implementation offering memory protection, checksumming and forward error correction.

Keywords: Dependability, Data Storage, Filesystem, Spacecraft, Satellite, Radiation, MRAM, Memory Protection, Failure Tolerance, EDAC

1 Introduction

Recent small- and nanosatellite development has shown a rapid increase in available compute power and storage capacity, but also in system complexity. Cubesats [1], are currently the most popular nanosatellite form factor due to their cost efficiency and ever increased system performance. The authors are involved in developing such a satellite, MOVE-II, whose predecessor, First-MOVE, was launched into Low Earth Orbit (LEO) in 2013.

More challenging quality requirements, limitations in energy consumption, heat dissipation and the generally extreme environmental conditions result in spaceflight software and hardware evolution being considerably more time consuming and slower paced. Ultimately, nanosatellite computing will evolve away from federated clusters of specialized microcontrollers [2], a development that could also be observed with larger spacecraft over the past decades. Instead, more powerful, hardened, centralized general purpose computers will cover a wider range of responsibilities [3, 4]. Thereby, overall spacecraft complexity can be reduced and efficiency improved, while each individual computer's complexity increases [2]. Certainly, an increased compute burden also requires more sophisticated operating system (OS) software, which in turn results in increased code complexity and size [5].

For very simple computers, custom tailored OSs offer an excellent balance of size and functionality. However, development of proprietary OSs for unique custom computers has been abandoned in most of the IT industry, in favor of standard soft- and hardware reuse. This is still an ongoing process in spaceflight, though already producing a focus on a few types of radiation hardened processor platforms (i.e. LEON3, PPC750, RAD6000, see [6]) running common OSs [7, 8]. The same evolution has begun in nanosatellite computing, albeit much faster.

OSs popular in spaceflight such as RTEMS can consume less than 256KB of non-volatile (nv) memory [9], whereas Linux requires at least 2MB. If such a larger OS is used aboard a satellite, more sophisticated storage concepts are needed. Data must be stored permanently and consistently throughout the mission lifetime. Space missions often last between 5 and 10 years [10], but can reach 25 years or longer like with the Voyager probes. Thus a satellite's command and data handling (CDH), the on-board computer, must guarantee integrity and recover degraded or damaged data (error detection and correction – EDAC) over a prolonged period of time in a hostile environment. We consider a filesystem (FS) the most resource conserving and efficient approach, which also allows dynamically adjustable protection for the individual data structures. As Magnetoresistive Random-Access Memory (MRAM) [11] is widely used for radiation resistant data storage in nanosatellites, FTRFS is applied FS to this technology.

This paper is organized as follows: Section 2 introduces the specific requirements and hazards to computing in orbit and deep-space, as well as the properties of different memories. Section 3 analyzes existing FSs and related research, to avoid implementation from scratch. Section 4 presents our FS, offering memory protection, forward error correction (FEC) and checksumming for both data and metadata. First results of our FS implementation are provided in Section 4.4 and its limitations are elaborated in Section 4.5. Finally, Section 5 contains potential solutions and our next steps in development.

2 Impact of the Spaceflight Use Case

Besides extreme temperature variations and the absence of atmosphere for heat dissipation, the impact of the near-Earth radiation environment must be considered in space computing. About 20% of all anomalies [12] aboard satellites can be attributed to high-energy particles from the sources depicted in Figure 1.

Particles originating from Earth's radiation belts, the Van Allen belts, consist mostly of trapped protons and electrons. Galactic Cosmic Rays from beyond our solar system are mostly protons [13, 14], whereas various other high-energy particles are ejected by the Sun during Solar Particle Events (SPEs).

Therefore, depending on the orbit of the spacecraft and the occurrence of SPEs, an on-board computer will be penetrated by a mixture of high-energy protons, electrons and heavy ions. Physical shielding using aluminum or other material can reduce certain radiation effects. However, sufficient protection would require a spacecraft to dedicate unreasonable additional mass to shielding.

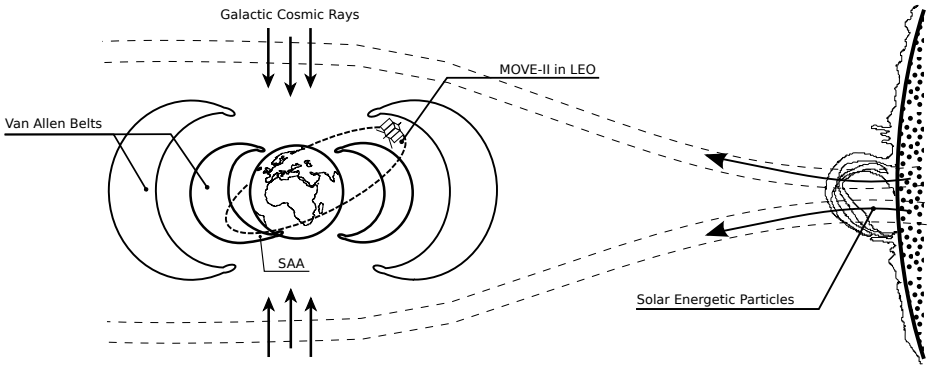

Fig. 1. The sources of radiation effecting a satellite. Figure is not to scale.

Furthermore, in LEO, the radiation bombardment will be increased while transiting the South Atlantic Anomaly (SAA). Earth's magnetic field experiences a local, height-dependent dip within the SAA, due to an offset of the spin axis from the magnetic axis. In this zone, a satellite and its electronics will experience an increase of proton flux of up to 10^4 times (energies > 30 MeV) [14]. This flux increase results in a rapid growth of bit errors and other upsets in a satellite's CDH. In case of MOVE-II, the full functionality of CDH-subsystem is required at all time due to scientific measurements being conducted from one of the satellite's possible payloads, even though brief outages (e.g. reboots) are acceptable. This scientific payload should measure the anti-proton flux within the SAA, as its physical properties are subject of scientific debate.

Different storage technologies vary regarding the energy-threshold necessary to induce an effect and the type of effect caused. The most important radiation induced phenomena on memory are:

- Single Event Effects (SEE), local ionization from protons or heavy ions
- Total Ionizing Dose (TID), the cumulative effect of charge trapping in the oxide of electronic devices
- Displacement Damage due to structural displacement in crystalline components of electronic hardware.

Other types of SEEs, the destructive ones being the most relevant, are well described in [15]. Some novel memory technologies (e.g. MRAM [11], PCRAM [16]) have shown inherent radiation tolerance against bit-flips, Single Event Upsets (SEUs), due to their data storage mechanism [17, 18].

Due to a shifting voltage threshold in floating gate cells caused by TID, commercial flash memories are more susceptible to bit errors. Highly scaled flash memories are also prone to SEUs causing shifts in the threshold voltage profile of one or more storage cells, referred to as Multiple Bit Upset [19].

All these memory technologies are sensitive to Single Event Functional Interrupts (SEFIs) [20], which can affect blocks, banks or entire circuits due to particle strikes in the peripheral circuitry.

3 Related Work and Preexisting File Systems

Filesystems often include performance optimizations like disk head tracking, utilization of data locality and caching. However, most of these enhancements do not apply to storage technologies used in spaceflight. In fact, such optimizations add significant code overhead, possibly resulting in a more error prone FS and may even reduce performance.

Next-generation FSs, e.g. BTRFS and ZFS, are designed to handle many-terabyte sized devices and RAID-pools. Silent data corruption has become a practical issue with such large volumes [21]. Thus, these FSs can maintain checksums for data blocks and metadata. Due to their intended use in large disk pools, they do also offer integrated multi-device functionality.

Multi-device functionality would certainly be advantageous, but neither ZFS nor BTRFS scale to small storage volumes. Minimum volume sizes are far beyond what current nanosatellite CDHs can offer. Also, future development of these FSs will eventually result in design decisions not in favor of spaceflight application.

FSs for flash devices, like the memory technology itself, have evolved considerably over the past decade [22, 23]. Upcoming FSs already handle challenges like potentially negative compression rates [24] or erase/write-block abstraction, offer proper wear leveling and interact with device EDAC functionality (checksumming, spare handling and recovery). UFFS even offers integrity protection for data and metadata using erasure codes.

Most new flash-FSs interact directly with memory [1], thereby are incompatible with other memory technologies unless flash properties are emulated. This introduces further IO and may result in unnecessary data loss, as flash memory is of course block oriented.

RAM filesystems are usually optimized for throughput or simplicity, often resulting in a relatively slim codebase. If designed for volatile RAM, these FS are optimized for simplicity and do not necessarily require a nondestructive unmount procedure. Non-volatile RAM FSs perform direct memory access to optimize for throughput, other utilize compression to increase storage capacity [25].

Except for *PRAMFS* [26], none of these FSs consider memory protection to increase dependability. *PRAMFS* offers execute-in-place (XIP) support [27] and is POSIX-compatible, but offers no data integrity protection.

In contrast to flash memories RAM filesystems are not block based, but benefit from the ability to access data arbitrarily. Thereby, no intermediate block management is required and read-erase-update cycles are unnecessary. While simple block-layer EDAC would certainly be possible, structures within a RAM filesystem can be protected individually allowing for stronger protection.

Open source space engineering and CDH research is directed mainly towards testing radiation related properties of memory technologies [20, 28] and on NAND-flash in particular [29, 30]. At the time of this writing, we are unaware of advanced software-side non-flash driven storage concepts for space use.

[1] in the case of Linux through the memory technology device subsystem (MTD)

4 FTRFS

FTRFS (fault-tolerant radiation-robust filesystem for space use) operates efficiently with small volumes($\leq$4MB), but also scales to larger volumes and is bootable.

Regarding the FS's threat model, ECC is applied to all CPU-caches and volatile SRAM, thus faults in these deceives are considered detectable and possibly correctable at runtime. A CPU running FTRFS must be equipped with a memory management unit with its page-table residing in volatile memory. All other elements (e.g. periphery and ALUs), other memories (e.g. registers and buffers) and in-transit data are considered potential error sources, see Section 2.

Memory protection has been largely ignored in RAM-FS design. In part, this can be attributed to a misconception of memory protection as a pure security-measure against malware. However, for directly mapped nv-memory, memory protection introduces the memory management unit as a safeguard against data corruption due to upsets in the system [32]. Thus, only in-use memory pages will be writable even from Kernel space, whereas the vast majority of memory is kept read-only, protected from misdirected write access i.e. due to SEUs in a register used for addressing during a store operation.

While data compression has been popular in size constrained FSs, it would offer low compression rates, as well-compressible data, e.g. log data, will not be kept in the same memory as the OS core components. Thus, it would offer little capacity gains but entail severe code overhead.

After a detailed OS evaluation, we chose the Linux kernel as the base for our FS due to its adaptability, extensive soft/hardware support and vast community. We decided against utilizing RTEMS mainly due to our limited software development manpower.

A loss of components has to be compensated at the software- or hardware level through voting or simple redundancy. Multi-device capability was considered for this FS, however it should rather be implemented below the FS level (e.g. via majority voting in hardware [33]) or as an overlay, e.g. RAIF [34].

The capability to detect and correct metadata and data errors was considered crucial during development. Based on the mission duration, destination or the orbit a spacecraft operates in, different levels of protection will be necessary. The protective guarantees offered can be adjusted at format time or later through the use of additional tools.

Our satellite's CDH offers 32MB of ECC-SRAM and is driven by an ARM Cortex-A5 CPU, however it could be upgraded to a Cortex-A7-MP. Due to the relatively restricted system resources aboard a nanosatellite, cryptographic checksums do not offer a significant benefit. Instead, CRC32 is utilized for performance reasons in tandem with Reed-Solomon encoding (RS) [31].

4.1 Metadata Integrity Protection

For proper protection at the FS level, in addition to the stored filesystem objects (inodes) and their data, all other metadata must be protected. Figure 2 depicts

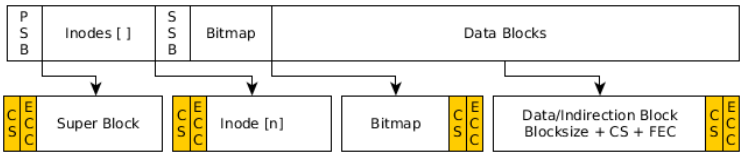

Fig. 2. The basic layout of the presented FS. EDAC data is appended or prepended to each FS structure. PSB and SSB refer to the primary and secondary super blocks.

the basic layout. Although similar to *ext2* and *PRAMFS* [26], data addressing and bad block handling work fundamentally different. We borrow memory protection from the *wprotect* component of *PRAMFS*, as well as the superblock and inode layout. *PRAMFS* is licensed under GPLv2 and based upon *ext2*.

The Super Block (SB) is kept redundantly, as depicted in Figure 2. An update to the SB always implies a refresh of the secondary SB, hence, hereafter no explicit reference of the secondary SB will be made. The SB also contains EDAC parameters for blocks, inodes and the bitmap.

The SB is the most critical structure within our FS, and is static after volume creation. Its content is copied to system memory at mount time, thus it is sufficient to assure SB consistency the first time it is accessed.

As the SB contains critical FS information, we avoid accumulating errors over time through scrubbing. Thereby, the CRC checksum is re-evaluated each time certain filesystem API functions (e.g. directory traversal) are performed.

A block-usage bitmap is dynamically allocated based on the overhead subtracted data-block count and is appended to the secondary SB. The bitmap EDAC is also dynamically sized and must be stored beyond the compile-time static SB, even though placing it there would be convenient. Thus, the protection data is located in the first block after the end of the bitmap, see Figure 2. In case the bitmap is extended, the new part of the bitmap is initialized and then the error correction data is recomputed at its new location. We refrain from recomputing and re-checking the EDAC data upon each access, instead FEC data is checked before and updated after each relevant operation has been concluded.

Inodes are kept as an array. Their consistency is of paramount importance as they define the logical structure of the filesystem. The array's length is determined upon FS initialization and can change only if the volume is resized. As each inode is an independent entity, an inode-table wide EDAC is unnecessary. Instead, we extend and protect each inode individually.

4.2 Data Consistency and Organization

To optimize the FS towards both larger (e.g. a kernel image, a database) and very small (e.g. scripts) files, direct and double indirect data addressing are supported, as depicted in Figure 3. The FS selects automatically which method is used. Data protection requirements vary depending on block size, and use case. Thus FTRFS allows the user to adjust the protection strength for data blocks, as will be described in the next section.

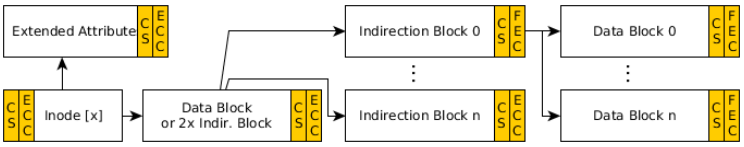

Fig. 3. Each inode can either utilize direct addressing or double indirection. Extended attributes are always addressed directly.

Data block size cannot be arbitrarily decreased, as some Linux kernel subsystems assume them to be sized to a power of two. Instead, the FS internally utilizes larger blocks to include EDAC data, see Figure 4.

Extended attributes (*xattr*) are deduplicated and referenced by one or more inodes, as depicted in Figure 3. Like in *PRAMFS*, *xattrs* are stored as data blocks, thereby we can treat these identically to regular data.

Nanosatellites, at least the non-classified ones, are not yet considered security critical devices. However, the application area of nanosatellites will expand considerably in the future [3]. An increasing professionalization will introduce enhanced requirements regarding dependability and security. Shared-satellite usage scenarios as well as technology testing satellites will certainly also require stronger security measures, which can be implemented using *xattrs*.

An *xattr* block's integrity is verified once its reference is resolved. Once all write access (in bulk) has been concluded, the EDAC data is updated.

4.3 Algorithm Details and Performance

Our primary goal was to develop an FS which could be used to store a full size-optimized Linux root FS including a kernel image safely over a long period of time within an 8MB volume. There are numerous erasure codes available that could be used to protect our FS. After careful consideration, RS was chosen mainly due to the following reasons:

- The algorithm is well analyzed, and widely used in various embedded scenarios, including spacecraft. Optimized software implementations, IP-cores and hardware accelerations are available.
- MRAM, while being SEU immune, is still prone to stray-writes, controller errors and in-transit data corruption. RS relies upon symbol level error correction, which is precisely the kind of corruption the FS must correct. Misdirected access within a page evades memory protection and corrupts the FS, thus corrupted single-byte, 2, 4 and 8B runs will occur.

RS decoding is computationally expensive, thus the protected data is subdevided into sub-blocks sized to 128B plus the user specified error number of correction-roots simplifying addressing and guaranteeing data alignment for power-of-two correction-root counts. Inodes and SBs can be fit into one single RS-code, while data block length does not result in extreme checking times. To

skip the expensive RS decoding step during regular operation, a CRC32 checksum allows high-performance checking. The RS-code is only read in case the checksum is invalid.

Data blocks are divided into subblocks so the FS can make optimal use of the RS code length. For common block-sizes and error correction strengths, 5 to 19 RS codes are necessary, see Table 1 for information on expected overhead. The correction data is accumulated at the end of the data block. Checksums across the entire block's data, each subblock and the error correction data are also retained. The resulting data format is depicted in Figure 4. Protection can be enhanced further by performing symbol interleaving for the RS codes and the block data, at the cost of performance.

FS traversal and data access will eventually slow down for strongly degraded storage volumes. As we immediately commit corrected data to memory, performance degradation is only temporary, assuming soft-faults.

4.4 Results and Current Status

FTRFS is currently undergoing testing and has been implemented for the Linux kernel. Due to its POSIX-compliance, it could easily be ported to other platforms. The memory protection functionality has been inherited from *PRAMFS*, the FS structure from *ext2*. We utilize the RS implementation of the Linux kernel, as its API also supports hardware acceleration.

Several components of the FS should undergo an optimization process, which will result in a drastic performance increase. Even though we have not yet conducted long-term benchmarking and performance analysis, the throughput degradation during regular operations is minimal. Modern CPUs can compute CRC32 within a few cycles due to hardware acceleration. We intend to publish additional performance and energy consumption metrics, once testing has been concluded and basic optimizations have been applied and the OBC computer has been finalized.

Data is read and written once per access. It is good practice in critical scenarios and especially spaceflight to read and write data multiple times, or deploy

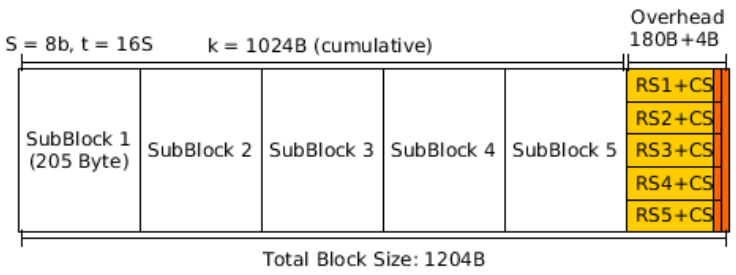

Fig. 4. A data block subdivided into 5 subblocks. Separate checksums for the entire data block, EDAC data and each subblock are depicted in red.

more advanced consistency checking techniques [35]. These changes could be applied in bulk, through a macro, or compiler side.

The level of protection offered by the FS can be adjusted at format-time, or later by using a proprietary FS-tuning tool. RS has a long record of space use in CDH and communications. Thus, we know the algorithm offers efficient protection regarding our threat scenario. Once testing has been concluded, we will perform long-term performance analysis in a degraded environment. To benchmark the FS, data degradation can be introduced using fault injection and we will be performed these tests after optimizations have been applied. However, artificial fault injection is usually not considered sufficient to prove the efficiency of a fault-tolerance concept for space-use. Our satellite's CDH computer including the FS will – and in general a satellite has to – undergo testing using various radiation sources before launch. Results will be made available once these tests have been carried out.

4.5 Conceptional Limitations and Restrictions

It is debatable whether journaling would increase FTRFS's reliability, as it usually helps safeguard FS consistency with slow storage media [36] due to power loss or disconnect. However, all access in our FS happens synchronously, and MRAM is only slightly slower than regular DRAM. Thus, journaling is currently not implemented.

Loss of power can also happen in our spaceflight use case, but depending on the event it can be handled differently. Spacecraft are battery backed and will utilize on-PCB components providing relatively abundant hold-back time after the electrical power subsystem (EPS) and the battery are disconnected due to latch-up protection. The FS can thus either conclude a pending write operation within the remaining active time, or the OS will have sufficient time to cancel pending writes in case the system has sufficient warning time.

The FS can not protect itself from device or memory bank failure. However, as MRAM access is deterministic, majority voting can be implemented in hardware to compensate for device failure [33]. This would also further increase protection against SEFIs, as upsets within one chip would be compensated by voting.

Data Structure	Size (B)	Correction Symbols/Code	# Codes	Correction Total (B)	Overhead (B)	Overhead (%)
Super Block	128	32	1	32	68	53.13%
Inode	160	32	1	32	68	42.50%
Data Blocks	1024	4	5	20	68	5.86%
	1024	16	5	80	188	17.58%
	4096	4	17	68	212	4.98%
	4096	16	19	304	692	16.70%
Bitmap	1773	32	10	320	688	38.80%

Table 1. EDAC overhead for FS structures. Bitmap: 16MB FS, 5% inodes, 1024B BS

If data is stored with RS-symbol interleaving, a XIP mapping would technically be impossible. XIP could still perform mappings for non-interleaved data though, but thereby only the clear-text part of each RS code would be mapped and read. Via this memory mapping, integrity protection for stored file data would be ignored, unless we accept that a potential XIP mapping would allow program code to be loaded/executed without any integrity checking. Thereby, the integrity assumptions upon which FTRFS's concept is based would be violated and integrity could not be guaranteed for any executed program stored on the FS. Theoretically, data integrity could also be checked each time a mapping is established for a block. To perform these checks however, this data would have to be read in full, obsoleting the performance advantage and RAM conserving properties of XIP. XIP and FS-level data integrity protection can thus be considered mutually exclusive.

5 Outlook and Future Work

Permanent defects will require FEC upon every access to an object. If such a hard fault occurred in a frequently accessed object (e.g. the root inode or a populated directory), we would want to avoid future re-checks. In the current FS implementation, there is no functionality to avoid this behavior, however it could be added later on.

Bad-block relocation is already implemented within the FS, but only used during write, truncate and allocation operations, not during other access. The only exception hereby is the root inode, which currently is assumed to be in a fixed location, like in *PRAMFS*. This feature as well could be implemented in a future version and would certainly increase storage reliability, performance and reduce data degradation.

FTRFS could theoretically also operate on different memory technologies, however, most of its advantages are enabled through RAM properties. Protection at the FS layer would be rather complex, unwieldy and could still not offer proper protection against device failure. Thus, the authors are working on a different protective concept for flash memory.

In contrast to RAM, flash access times will vary depending on block integrity. Thus, full voting based majority decisions would require very complex control logic. If voting was conducted utilizing hardware-side flash controllers, a delayed response from one controller would stall access to the entire voting circuit. Even if the result has already been determined, the circuit would still be busy.

A transparent protective layer utilizing RAID1, FEC and checksuming could however be implemented as an MTD middleware layer. MTD-striping [37] has been proposed as a middleware function in the past, but has never been included in the Linux Kernel. However, the existence of the MTD-striping code proofs the feasibility of a mirroring and protection MTD-layer.

6 Conclusions

We presented a novel filesystem implementation enabling a software-side protective scheme against data degradation due to environmental effects introduced by the space environment as described in Section 2. We have shown the feasibility of a bootable, POSIX-compatible FS which can efficiently protect an OS image from device failure and software flaws according to the threat model outlined at the beginning of Section 4.

With respect to our use case in spaceflight, neither component level, nor hardware- or software-side measures individually can guarantee sufficient system consistency. Traditionally, radiation effects in space systems are compensated for with stronger hardware-EDAC and component-redundancy, which do not scale for complex systems and result in increased energy consumption. While redundancy and hardware-side voting can protect well from device failure, data integrity protection is difficult at this level. A combination of hardware and software measures can thus drastically increase system dependability, even for missions with a very long duration.

References

1. H Heidt et al. Cubesat: A new Generation of Picosatellite for Education and Industry Low-Cost Space Experimentation. In *14. AIAA/USU Conference on Small Satellites, Proc.*, 2000.
2. S Busch and K Schilling. UWE-3: A Modular System Design for the Next Generation of Very Small Satellites. In *Proceedings of Small Satellites Systems and Services–The 4S Symposium, Slovenia*, 2012.
3. D Evans and M Merri. OPS-SAT: An ESA Nanosatellite for Accelerating Innovation in Satellite Control. Spaceops, 2014.
4. C Bridges et al. Smartphone Qualification & Linux-based Tools for Cubesat Computing Payloads. In *Aerospace Conference, 2013 IEEE*, pages 1–10. IEEE, 2013.
5. M Stringfellow, N Leveson, and B Owens. Safety-Driven Design for Software-Intensive Aerospace and Automotive Systems. *IEEE, Proc.*, 98(4):515–525, 2010.
6. K Ryu, E Shin, and V Mooney. A Comparison of Five Different Multiprocessor SoC Bus Architectures. In *Digital Systems Design, 2001. Proceedings. Euromicro Symposium on*, pages 202–209. IEEE, 2001.
7. D McComas. NASA/GSFC's Flight Software Core Flight System. 2012.
8. J Williams and N Bergmann. Reconfigurable Linux for Spaceflight Applications. *Proc. Military and Aerospace Programmable Logic Devices (MAPLD 04)*, 2004.
9. D Atienza et al. Systematic Dynamic Memory Management Design Methodology for Reduced Memory Footprint. *ACM-TODAES*, 11(2):465–489, 2006.
10. J Saleh, D Hastings, and D Newman. Weaving Time into System Architecture: Satellite Cost per Operational Day and Optimal Design Lifetime. *Acta Astronautica*, 54(6):413–431, 2004.
11. R Katti, H Stadler, and J Wu. High Speed Magneto-resistive Random Access Memory, December 22 1992. US Patent 5,173,873.
12. S Bourdarie and M Xapsos. The Near-Earth Space Radiation Environment. *IEEE Trans. on Nuclear Science*, 55:1810–1832, 2008.

13. M Xapsos, P O'Neill, and T O'Brien. Near-Earth Space Radiation Models. *IEEE Transactions on Nuclear Science*, 60:1691–1705, June 2013.
14. J Schwank, M Shaneyfelt, and P Dodd. Radiation Hardness Assurance Testing of Microelectronic Devices and Integrated Circuits. *IEEE Transactions on Nuclear Science*, 60:2074–2100, June 2013.
15. ESA/ESTEC Requirements and Standards Division ECSS: Calculation of Radiation and its Effects and Margin Policy Handbook. ECSS-E-HB-10-12A, 2010.
16. F Chen. Phase-Change Memory, February 26 2014. US Patent App. 14/191,016.
17. G Tsiligiannis et al. Testing a Commercial MRAM Under Neutron and Alpha Radiation in Dynamic Mode. *IEEE Trans. on Nuclear Science*, 60, 2013.
18. J Maimon et al. Results of Radiation Effects on a Chalcogenide Non-Volatile Memory array. In *Aerospace Conference, 2004. Proceedings. 2004 IEEE*, volume 4, pages 2306–2315. IEEE, 2004.
19. S Gerardin et al. Radiation Effects in Flash Memories. *IEEE Transactions on Nuclear Science*, 60:1953–1969, June 2013.
20. D Nguyen and F Irom. Radiation Effects on MRAM. In *Radiation and Its Effects on Components and Systems*, pages 1–4. IEEE, 2007.
21. M Baker et al. A Fresh Look at the Reliability of Long-Term Digital Storage. In *ACM SIGOPS Operating Systems Review*, volume 40, pages 221–234. ACM, 2006.
22. J Engel and R Mertens. LogFS – Finally a Scalable Flash File System. In *12th International Linux System Technology Conference*, 2005.
23. S Qiu and N Reddy. NVMFS: A Hybrid File System for Improving Random Write in NAND-Flash SSD. In *Mass Storage Systems and Technologies (MSST), 2013 IEEE 29th Symposium on*, pages 1–5. IEEE, 2013.
24. W Liangzhu. The Investigation of JFFS2 Storage. In *Microcomputer Information 8*, 030, 2008.
25. N Edel et al. MRAMFS: a compressing file system for non-volatile RAM In *MASCOTS 2004. Proceedings. The IEEE Computer Society's 12th Annual International Symposium on.*, IEEE, 2004.
26. M Stornelli. Protected and Persistent RAM Filesystem. pramfs.sourceforge.net.
27. J Hulbert. The Advanced XIP File System. In *Linux Symposium*, page 211, 2008.
28. M Elghefari et al. Radiation Effects Assessment of MRAM Devices. 2008.
29. M Cassel et al. NAND-Flash Memory Technology in Mass Memory Systems for Space Applications. In *DASIA 2008*, volume 665, page 25, 2008.
30. H Herpel et al. Next Generation Mass Memory Architecture. In *DASIA 2010*.
31. SB Wicker et al. *Reed-Solomon Codes and their Applications*. Wiley & Sons, 1999.
32. S Suzuki and K Shin. On Memory Protection in Real-Time OS for Small Embedded Systems. In *Real-Time Computing Systems and Applications, 1997. Proceedings., Fourth International Workshop on*, pages 51–58. IEEE, 1997.
33. S Su et al. A Hardware Redundancy Reconfiguration Scheme for Tolerating Multiple Module Failures. *Computers, IEEE Transactions on*, 100(3):254–258, 1980.
34. N Joukov et al. Raif: Redundant Array of Independent Filesystems. In *Mass Storage Systems and Technologies, 2007. MSST 2007. 24th IEEE*, pages 199–214.
35. B Cagno et al. Verifying Data Integrity of a Non-Volatile Memory System during Data Caching Process. US Patent 8,037,380.
36. V Prabhakaran, A Arpaci-Dusseau, and R Arpaci-Dusseau. Analysis and Evolution of Journaling File Systems. In *USENIX Annual Technical Conference, General Track*, pages 105–120, 2005.
37. A Belyakov. Linux-MTD Striping Middle Layer. Linux-MTD mailing list, 03 2006.