FLARE: a Design Environment for FLASH-based Space Applications

Maurizio CARAMIA(*), Stefano DI CARLO (+), Michele FABIANO(+), Paolo PRINETTO(+)

(*)
Thales Alenia Space Italia
Torino, Italy

{Maurizio.Caramia}@thalesaleniaspace.com

(+)
Politecnico di Torino
Dipartimento di Automatica e Informatica
Torino, Italy

{Stefano.Dicarlo, Michele.Fabiano, Paolo.Prinetto}@polito.it

***Abstract* – Designing a mass-memory device (i.e., a solid-state recorder) is one of the typical issues of mission-critical space system applications. Flash-memories could be used for this goal: a huge number of parameters and trade-offs need to be explored. Flash-memories are nonvolatile, shock-resistant and power-economic, but in turn have different drawback: e.g., their cost is higher than normal hard disk and the number of erasure cycles is bounded. Moreover space environment presents various issues especially because of radiations: different and quite often contrasting dimensions need to be explored during the design of a flash-memory based solid-state recorder. No systematic approach has so far been proposed to consider them all as a whole: as a consequence a novel design environment currently under development is aimed at supporting the design of flash-based mass-memory device for space applications.**

I. INTRODUCTION

Nowadays processing power available for embedded technology and boards is absolutely overcoming the one available just a few years ago. However, in space applications the very strict requirements have often driven the design choices toward older and/or lower-performing radiation-tolerant electronics. Although each new space application has its own story and requires increasing intelligence and autonomy [1], a typical mission-critical space system application includes solid state recorder(s), redundant mission computers, flight guidance and navigation systems, health monitoring computers and robotic manipulator controllers.

The issue of solid-state recorder(s) for space applications is addressed in this paper. In particular, we shall present a design environment currently under development, to support the design of flash-based hard disks (HDs) for space applications.

Flash-memory based systems are gaining acceptance and usage not only in the consumer market but in space applications, as well, where they mainly play the role of high-capacity storage devices: in fact flash-memory guarantees both the non-volatility in case of power loss and a highest storage density [2]. Moreover they are shock-resistant and power-economic: power consumption is always a critical issue in space applications. However designing flash-based systems for space application requires both exploring a huge number of design dimensions and evaluating a huge amount of trade-offs among all such dimensions. The most relevant dimensions include, e.g., flash-memory technology, flash-memory architecture, file management system, dependability enhancement strategies, power consumption, weight, physical size and so on. A complex and powerful design environment is thus needed to properly evaluate the impact of the choices in each dimension and the related trade-offs.

Unfortunately no such a design environment is today available: in fact in the literature each paper is typically tackling just one specific issue in just one design dimensions. No systematic approach has so far been proposed to consider them all as a whole. Such a concurrent exploration capability is mandatory to provide the designers a powerful design environment, capable of supporting them through all the steps of the design cycle, including Architectural Exploration, Design Validation & Verification, (Automatic) Test insertion, Dependability evaluation and so on.

This paper presents the general architecture of a novel design environment currently under development to support the design of flash-based mass memories, especially for space applications. As pointed out before, the project is mainly pushed by the unavailability, at our best knowledge, of a commercial tool capable of supporting a systematic analysis and exploration of the different possible alternatives.

The rest of the paper is organized as follows: Section 2 addresses the main flash-memory peculiarities, Section 3 explores the dimensions of the issues of designing flash-based mass memory devices, focusing the attention also on the space applications, Section 4 deals with the modeling of flash-based HDs, while Section 5 proposes a possible architecture for a design environment to support the design of flash-based hard disks for space applications.

II. FLASH-MEMORY PECULIARITIES

Flash-memories present several interesting features that properly fit with the requirements of mass-memories for space applications, while possible alternatives need to be evaluated.

Our solid-state recorder would need a relative high capacity: the first most suitable solution could be DRAMs. On the one hand DRAMs are very fast, reliable and provide a very high data rate, but on the other hand they need a battery pack-up to not lose data and this issue generate an intricate balance between battery mass and data retention time: data retention over years is not feasible and count of battery charge cycles are limited. DRAMs are not discussed anymore in this paper.

The second and more attractive solution is the use of flash-memories. There are two major types of flash-memory in the current market: NOR and NAND flash-memory. NOR flash-memory is for EEPROM replacement and is more suitable for program execution, while NAND flash-memory is more suitable for storage systems [3], [4]: Table 1 briefly sums up the main characteristic of these types of flash-memory.

This paper addresses only NAND flash-memories: in fact they are the most suitable choice for HD replacement.

On the one hand, flash-memories are nonvolatile, shock-resistant, and power-economic: a pure flash-memory based solution could guarantee unlimited data retention time and no need of battery backup. On the other hand flash-memories are still much more expensive than hard disk drive memory. Only rather large data structures could be accessed and, in addition, DDR2 SDRAMs provide higher write/read rate (e.g., 6 Gbit/s) compared to the moderate one accomplished by flash-memories (e.g., up to 80/200 Mbit/s) [2]. Moreover one of the main challenging aspects of flash-memories is that the space already written (i.e., programmed) with data usually cannot be overwritten unless it is erased from the flash-memory device.

A NAND flash-memory is usually partitioned into *blocks*: each block has a fixed number of *pages* and each page has a fixed size. A block is the smallest unit for erase operations, while read and write operations are done in terms of pages, i.e., a page can be erased only if its whole corresponding block is erased, whereas a page can be read/written independently.

In addition flash-memory wears out after a certain number of erasure cycles (i.e., actually 10^6 for NAND flash-memory): if the erasure cycles of a block exceed this number, it becomes a "bad block" and is not reliable for storing data anymore.

	NAND	**NOR**
Standby Power	Low/Med	Low
Active Power	Low	Med/High
Cost per bit	Low	High
Read Speed	Med/High	High
Write Speed	High	Low
Erase Speed	High	Medium
Capacity	High	Low
Erase Cycles	10^6	10^5
File Storage Use	Easy	Hard
Code Execution	Hard	Easy
Interface	I/O-like	SRAM-like

TABLE 1 – NAND VS NOR FLASH-MEMORY

Finally another peculiar aspect of flash-memories is that technology provides the possibility of storing more than one bit of information per cell: in fact traditional flash-memory devices (Single-Level Cell or SLC) store only one bit per cell, while newer devices (Multi-Level Cell or MLC) are able to store typically two bits per cell.

III. ISSUES IN DESIGNING FLASH-BASED HARD-DISKS FOR SPACE APPLICATIONS

When a flash-based system for space application has to be designed, the investigation of a vast quantity of design parameters needs to be defined. A possible partial taxonomy could involve *Flash-memory Technology*, *Flash-memory Architecture*, *Flash-memory Wearing*, *Testing and Dependability* and, finally, *Using flash-memory as Hard-Disk*.

Designer should discriminate among which technology to choose (*Flash-memory Technology*) and they may also have to select the most appropriate flash-memory chipset (*Flash-memory Architecture*). [18]

In the problem of *Wearing*, a significant role is played by the so called wear leveling techniques, which are aiming at distributing data evenly across each memory block of the entire flash-memory to avoid single block to wear out. Several interesting wear leveling techniques have been proposed [13] – [16] and could be considered or higher capacity flash-memory devices could be used, then especially taking care of the resulting drawbacks in terms of weight and volume [2]. An effective comparative analysis of some wear leveling algorithms could be found in [14].

Flash-memory Testing is quite different from testing other kinds of memory: disturbances or faults not conforming to any of the traditionally known fault models used in testing RAMs could be experienced and specific fault models are needed to properly represent the most frequent physical defects are needed. Then efficient test algorithms are needed and Built-In Self Test (BIST) and Built-In Self Diagnosis (BISD) circuits have to been taken in consideration. [18]

However wearing, testing and dependability are strictly linked among each other and designers should evaluate the right trade-off among them.

Several challenging aspects need to be addressed when *using a flash-memory as a mass-memory device*. A possible incomplete taxonomy of these aspects could involve:

- looking for proper solutions in order to let OS successfully communicate with NAND flash-memory devices (i.e., *Operating System Management*) [3] – [8]
- implementing an efficient logical to physical address translation process for fast operations (i.e., *Address Translation*) [18]
- proper techniques to handle blocks exceeding the maximum number of erase cycles (i.e., *Bad Block Management*) [19]
- proper strategies for reclaiming invalidated space to be erased in order to free some space (i.e., *Garbage Collection*) [18]

Finally designers should address Error Detection And Correction (EDAC) techniques, evaluating the most proper choice for their design. [18]

[18] is a more detailed investigation about these main flash-memory issues discussed above.

A. *Flash-based Hard-Disks in the Space Environment*

A solid state recorder for critical space missions needs to satisfy many different constraints, including, among the others, no loss of mass memory data and the guaranteed availability of storage capability at End-Of-Life (EOL). A well-designed flash-based memory system can meet the requirements of interplanetary missions, but its design must compensate for flash's shortcomings in speed, radiation tolerance, noise, and read/write cycle life and this compensation leverage the costs.

On the one hand vendors should absolutely provide flash-memories physically qualified to survive in the space environment with the help of proper strategies [9] – [12], while on the other hand data integrity, reliability, simplicity, modularity, and autonomy are just some of the key features to fulfill (e.g., reliable storage of context data, also during spacecraft power outage).

Moreover designers should evaluate the most proper choice for accomplishing the level of dependability requested by their design, with the help of ECC algorithms for error checking and correction of NAND flash-memory. A more detailed survey about the most peculiar flash-memory design dimensions and trade-offs to tackle during the design of flash-based hard disks for space applications could be found in [18].

IV. MODELING OF FLASH-BASED HARD-DISKS

Several issues and aspects need to be addressed during the modeling of a flash-based HD. First of all, Figure 1 shows a first possible high view of a flash-based mass-memory device.

There are three main functional blocks: a *Non-Volatile Memory* is needed to provide integrity of data, a *Volatile Memory* is used for performance reasons, while a *Memory Controller* is managing and controlling the overall system, providing several features. The mass-memory device is interacting with the requests of the external world, e.g., the operations of the OS in use.

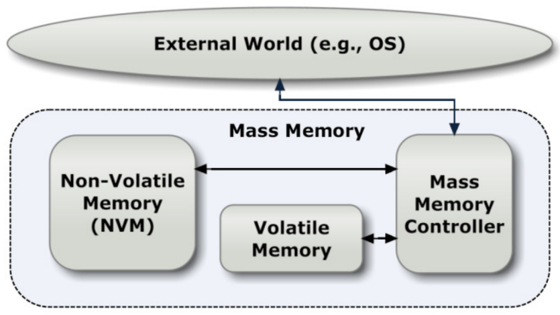
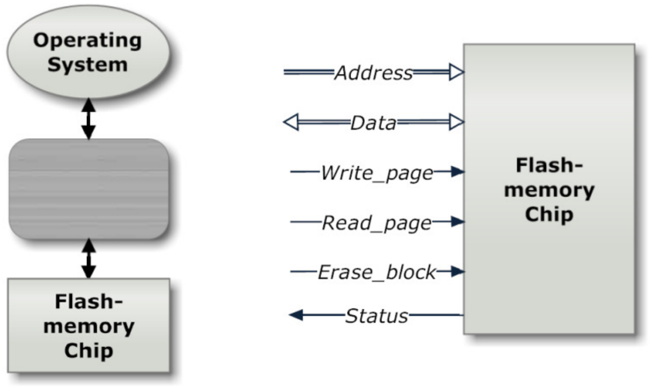


Figure 1 – A Flash-based Hard Disk Architecture



(a) (b)

Figure 2 – (a) High-level and (b) low-level view of flash architecture

It is possible to refine this first architectural view: in fact, during the design of a flash-based HD, the OS and the applications want to successfully communicate with the bare flash-memory chip. Figure 2 (a) shows that an intermediate block is needed to accomplish this task. On the one hand OS usually would like to exploit its typical system calls (e.g., *open, read, write*) to work with the mass-memory device, without taking care of anything else. On the other hand, the flash-memory chip would like to receive the most proper commands to accomplish the operations previously requested from the OS: Figure 2 (b) shows a possible view of the commands a flash-memory chip usually would like to get. E.g., [16] briefly presents a typical layered system architecture of popular flash-memory-based file systems.

So designers should develop a sort of *managing part* to tackle all the typical issues of flash-memories, presented also in the previous paragraphs. This is the most important and challenging part of designing a flash-based HD: many often contrasting issues, parameters and dimensions are involved in this part, which has to address them in the most proper way.

This *managing part* could be named "flash-memory manager" and could be split into its composing functional blocks as Figure 3 shows: these blocks represent the main issues a flash-memory based system has to tackle and to solve in the more possible efficient way. Designers have to manage how the logical to physical *Address Translation* is accomplished: reliability and efficiency are only two of the parameters of quality of this aspect. They need also to focus on *Wear Leveling* and *Garbage Collection* techniques and strategies. At the same time, designers need to distinguish among several *EDAC* strategies: a trade-off between needed reliability and related costs leads their choice over a particular code rather than another one. In addition designers have to manage *Bad Blocks*.

V. THE FLARE DESIGN EVALUATION ENVIRONMENT

In this section the proposed FLash ARchitecture Evaluation (FLARE) design environment to support the design of flash-based hard disks for space applications is introduced. FLARE is currently under development.

Actually there is no systematic support for the development of a flash-based hard disk qualified for space applications. Designers have always to think about the most suitable choice for the specific space applications they are dealing with: the huge number of variables and parameters could easily lead to unverified scenarios and to delayed product release.

In fact the level of confidence with these parameters is directly linked with the designers' skill, cleverness and experience. As a result, a systematic tool to support the design of flash-based hard disks for space applications is needed.

A. *Evaluation parameters*

FLARE tool is intended to evaluate several aspects of the design of a flash-based system. Designers have to tackle many critical issues: FLARE could help them to distinguish and identify the peculiar features of these aspects and to evaluate the most suitable solution for them.

The *capacity* of the flash-memory is the first fundamental parameter to set: designers should discuss about the physical quantity of flash-memory required by the design. This is a typical issue of space applications: in fact space critical missions require minimizing all the costs as much as possible and the dimension of the flash-memory is the first significant parameter that designers and their companies have to face. Designers could have to discriminate among different flash-memories of different capacities during the design of their system: FLARE could provide them with an overall evaluation of which *capacity* is more suitable for their design.

Designing a flash-based HD means dealing with NAND flash-memories, which are always partitioned in blocks and in turn each block is divided in pages: once *capacity* is set, designers have to address the *dimension* of each block and of each block or, in the same way, the *number* of blocks and pages for each block.

Obviously, designers could decide the *capacity* from the dimension and the number of blocks and pages, but the issue is practically the same. FLARE could help the designers to do this decision, in order to understand which level of granularity would more properly fit with the current design and to decide the most suitable flash-memory chipset.

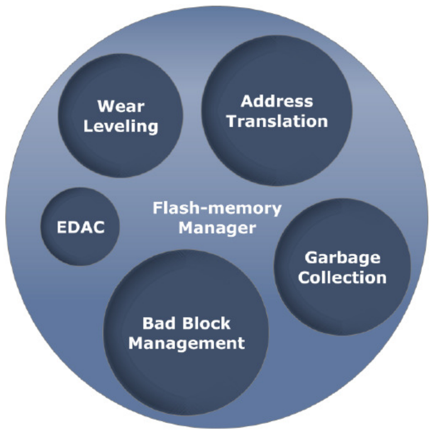


Figure 3 – The main functional block of *Flash-memory manager*

An essential parameter to evaluate is the *percentage of wearing of each block*: especially in mission-critical space applications, resources are always a key-point of the mission and it is desirable or, usually, mandatory that the percentage of wasted resources is as low as possible. E.g., on the one hand it could be enough 2GByte NAND flash-memory with some kind of wear-leveling techniques or on the other hand a bigger NAND flash-memory device could be requested in order to accomplish mission requirements. At the same time, designers could need to explore several kinds of solutions, in order to find that one with the most fitting percentage of wearing. As a consequence, this parameter is strictly linked to the adopted wear-leveling strategies: with the help of FLARE, designers could evaluate how this percentage varies as the wear-leveling techniques change.

As a consequence designers need to calculate the percentage of flash-memory which is not "dead", i.e., the percentage of blocks which did not become *bad blocks* at the End-Of-Life (EOL). Mission-critical space applications sometimes could explicitly require a fixed amount of flash-memory still alive at the EOL: designers have to evaluate the possible alternatives and to find the most affordable solution at the minimum possible cost for their design.

Designers have to provide a well-defined level of dependability according to their specific design. A fundamental role could be played by the so called *Out-Of-Bound (OOB)* data [3]: they can be exploited also to store some kind of ECC/EDAC codes, in order to accomplish the required dependability. The smart reader could get the unavoidable trade-off between spare data and user data: in fact it is true that a bigger *OOB* area could provide higher level of dependability, but at the same time would provide poor service in term of user data storage. Designers have to tackle this issue and find the most suitable solution for their particular design. Moreover, designers have to evaluate among the *Built-In Self Test* (BIST) functionalities, evaluating at the same time the *percentage of errors detected/corrected*.

This is only a possible incomplete taxonomy of what is needed to be evaluated during the design of a flash-based mass-memory device for space applications. Moreover all these parameters are strictly linked together and they affect each other in a complex way: so an exploration of these different and quite often contrasting dimensions is needed and no systematic approach has so far been proposed to consider them all as a whole.

B. *FLARE Architecture*

The proposed FLash ARchitecture Evaluation (FLARE) design environment is aimed at supporting designers through all the steps of the design cycle flash-based hard disk for space applications, including Architectural Exploration, Design Validation & Verification, (Automatic) Test insertion, Dependability evaluation and so on. FLARE is currently under development.

Figure 4 shows the architecture of the system.

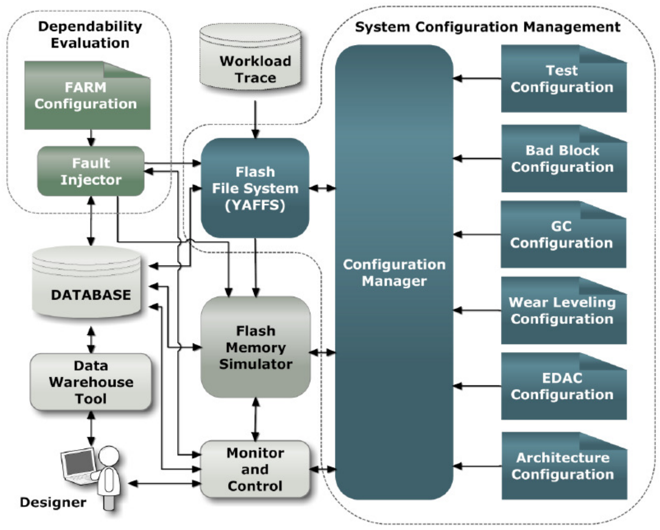


Figure 4 – A detailed view of FLARE architecture

1) System Configuration Management

The *System Configuration Management* allows setting and exploring the possible alternatives and design dimensions: designers are able to easily modify the memory configuration block (*Architecture configuration*), the Test infrastructure (*Test configuration*), and all the architectural solutions aimed at tackling Flash aging (*Bad block, Garbage Collection, Wear Leveling Configuration*).

The *Architecture Configuration* block is intended to contain all the details about the architecture of the flash-memory to emulate: capacity, number of blocks and number of pages are only some of the main architectural parameters that the designers are able to set.

With the *Test Configuration* block, the designer can set all the parameters for correctly testing the proposed flash-memory: all the issues addressed in the previous paragraphs are taken into account and the proper fault-models and the specific testing strategies can be specified, always according to the particular application flash-memory is used for.

As clearly showed previously, some wear-leveling strategies are needed to spread writes over the flash-memory: designers are capable to exploit the *Wear Leveling Configuration* block to specify all the details about the wear-leveling strategies to adopt during the emulation campaign. The range of these details can be variable: designers could choose a “simple” less/more aggressive wear-leveling strategy among the ones just provided with FLARE tool or developing their own wear-leveling algorithm could be a valid alternative, in order to evaluate it.

If wear leveling strategies aim to spread write operations over the flash-memory device, at a certain point invalidated space should be reclaimed: in the *Garbage Collection* module, designers are able to specify the strategies to identify a block, to collect its good pages and to erase it. It is usually strictly connected with wear-leveling strategies: it could even be considered that GC preferences are managed by wear-leveling strategies, but these two issues are kept separated now.

However, blocks exceeding the maximum number of erasure cycles are marked as bad: in *Bad Block Configuration* module designers can set the proper parameters to mark, identify and exclude bad blocks from active space memory. Simple well-known strategies could be used (e.g., *Skip Block Method*) as well as new approaches can be experimented and evaluated by designers.

Dependability of flash-memory need to be guaranteed: designers are able to specify in the *EDAC Configuration* block all the parameters needed to accomplish the required level of data integrity and reliability. E.g., a reasonable question could be if a CRC code is enough to accomplish the requested level of reliability or something more is needed. Maybe some ECC would be absolutely necessary to accomplish the required level, e.g., Orthogonal Reed-Solomon Error Correction Code might be the EDAC strategy designers were looking for.

In *EDAC Configuration* block, designers are capable of defining, exploring and evaluating all possible EDAC strategies for their particular design.

The designers’ configuration choices feed the so called *Configuration Manager* block: this layer is thought to take care of managing the “static” data coming from the various dimensions of the design of a flash-memory device (i.e., the “note” modules on the right) and to get it across the core functional blocks. This layer is essential for dispatching the updated configuration modules discussed above to the appropriate managing blocks. The architectural choice of having this kind of layer is strictly linked to flexibility: in fact on the one hand if some changes to parameters and algorithms are needed, designers can simply modify the proper module(s) not interesting in the rest, because the *Configuration Manager* layer will take care of dispatching the updated configuration(s) to the appropriate blocks. On the other hand, designers are capable of developing new (compatible) configuration modules, in case they felt like the existing modules were not enough for their needs: adding new configuration modules to the whole architecture would turn in very few efforts, thanks to this way of partitioning, and would result in high modularity and flexibility.

2) Flash Memory Simulator

The system kernel is the newly developed *Flash Memory Simulator*, charged of providing the designer the possibility of simulating and evaluating all the parameters of interest.

The *Flash-memory Simulator* block is one of the most important functional blocks of the FLARE tool: in fact it is thought to emulate the behavior of the configured flash-memory. The desired architecture is specified in the *Architecture Configuration* module discussed before: a “customized” architectural configuration for the flash-memory device could be identified or a ready for use configuration could be chosen from a developed library. Then the *Configuration Manager* takes this information and advertises the *Flash-memory Simulator* block about the architectural details of the flash-memory to emulate.

3) *Dependability Evaluation*

In addition to the overall architecture, some fault injection techniques could be considered: a *Fault Injector* functional block is added for this purpose. It is fed by a *Fault Activation Readout Measure (FARM) Configuration* block [17]: it sets all the needed parameters for the fault injector to make it work as requested. In this way a fault can be injected in the system to evaluate its effect in the emulated flash-memory. Fault injection is an additional function of the FLARE tool: in fact it is represented surrounded by a dotted rounded rectangle in order to highlight this point, i.e., it is not essential to the correctness of the FLARE tool, but at the same time it could be very useful for experimenting various fault injection techniques and configurations.

A *Fault injection* environment provides the designer to assess the target system dependability via a powerful manager of fault injection campaigns in all the part of the system itself. [17]

4) *Utilities*

As the name intuitively suggests, the *Monitor and Control* block is monitoring and controlling the output of the previous *Flash-memory Emulator* block. Designers can have under control all the events of the core blocks in order to get a more comprehensive knowledge about the countermeasure to take in some specific cases. The *Monitor and Control* block is peculiarly different from the *Data Warehouse Tool* block: in fact the last one is a mean with which the user can extract information about the emulated flash-memory at the EOL timeline, whereas the first one is a sort of automatic tool informing the user about the most significant events of the actual emulation campaign.

The use of a *Database* is fundamental to gather all the information needed at the EOL timeline: its role is simply to store data. The user is able to access the data with the help of a *Data Warehouse Tool*: data and metadata can be extracted, transformed and loaded, to easily accomplish all the designers' requests.

VI. Conclusions and Future Works

This paper has proposed FLARE, a design environment to support the designers during the design of flash-based hard disks for space applications. The composing blocks of the proposed architecture highlight the high-level of modularity and flexibility that this tool will be able to provide to designers: in fact each block is intended to be a sort of plug-in block, which can simply be plugged-out and replaced by another block when necessary, without taking care of the rest. As a result, designers are provided with a powerful and flexible environment, able to clearly identify the best choices for the current design.

FLARE tool is currently under development and refinement: the first implementation data of the tool are intended to be provided soon.

VII. References

[1] Anthony Lai: "Space-ready, radiation-tolerant processor modules: A COTS technology strategy", *Military Embedded Systems Resource Guide, May 2005*

[2] Cassel M., Walter D., Schmidt H., Gliem F., Michalik H., Stähle M., Vögele K., Roos P. Casel.: "NAND-Flash-memory Technology in Mass Memory Systems for Space Applications", *Proceedings Data Systems In Aerospace (DASIA) 2008, Palma de Mallorca, Spain, 2008*

[3] Chang L. P., Kuo T. W.: "An efficient management scheme for large-scale flash-memory storage systems", *Proceedings of the 2004 ACM Symposium on Applied Computing , Nicosia, Cyprus, 862-868, 2004*

[4] Hsieh Jen-Wei, Tsai Yi-Lin, Kuo Tei-Wei, Lee Tzao-Lin: "Configurable Flash-Memory Management: Performance versus Overheads" *IEEE Transactions on Computers, Vol. 57, no. 11, 2008*

[5] Intel Corporation, Technical Report: "Understanding the Flash Translation Layer (FTL) Specification", *December 1998*

[6] Woodhouse D., Red Hat, Inc.: "JFFS : The Journalling Flash File System", *http://sources.redhat.com/jffs2/jffs2.pdf , 2001*

[7] JFFS2, http://sourceware.org/jffs2/

[8] Aleph One Company, Cambridge, UK: "Yet Another Flash File System", *http://www.aleph1.co.uk/yaffs/index.html, 2002*

[9] Brüggemann M., Schmidt H., Walter D., Gliem F., Michalik H.: "Further Heavy Ion and Proton SEE Evaluation of High Capacity NAND-Flash-memory Devices for Safeguard Data Recorder", *8th ESA/ESTEC D/TEC-QCA Final Presentation Day, February 2007*

[10] Schmidt H., Walter D., Brüggemann M., Gliem F., Harboe-Sørensen R., Virtanen A.: "Heavy Ion SEE Studies on 4-Gbit NAND-Flash-memories", *Radiation Effects on Components and Systems (RADECS) 2007, DWL-14, September 2007*

[11] Schmidt H., Walter D., Gliem F., Nickson B., Harboe-Sorensen R., Virtanen A.: "TID and SEE Tests of an Advanced 8 Gbit NAND-Flash-memory", *Proc. IEEE Radiation Effects Data Workshop, 2008, 38-41*

[12] Brüggemann M., Schmidt H., Walter D., Gliem F., Harboe-Sørensen R., Roos P., Stähle M.: "SEE Tests of NAND Flash-memory Devices for Use in a Safeguard Data Recorder", *Radiation Effects on Components and Systems (RADECS) 2006, A-3, Volume A-3, 2006*

[13] SanDisk Corporation, White Paper: "SanDisk Flash-memory Cards Wear Leveling", *Doc. No. 80-36-00278, October 2003*

[14] Chang Li-Pin: "On Efficient Wear Leveling for Large-Scale Flash-MemoryStorage Systems", *Proceedings of the 22nd ACM Symposium on Applied Computing, 2007*

[15] M. L. Chiang, Paul C. H. Lee, R. C. Chang: "Using Data Clustering To Improve Cleaning Performance For Flash Memory", *Software - Practice and Experience, 1999*

[16] Chang Y.-H., Hsieh J.-W., Kuo T.-W.: "Endurance Enhancement of Flash-Memory Storage, Systems: An Efficient Static Wear Leveling Design" *Proc. 44th ACM/IEEE Design Automation Conference (DAC) '07, 212-217, 2007*

[17] Benso A., Prinetto P.: "Fault Injection Techniques and Tools for Embedded Systems Reliability Evaluation" – Kluver Academic Publishers, ISBN: 1-4020-7589-8, 2003

[18] Caramia M., Di Carlo S., Fabiano M., Prinetto P.: "Flash-memories in Space Applications: Trends and Challenges", *East-West Design & Test Symposium (EWDTS) 2009, Moscow, Russia, September 18-21, to appear*

[19] Samsung, Application Note: "XSR1.5 Bad Block Management", *May 2007*