A Fault-Tolerant and High-Speed Memory Controller Targeting 3D Flash Memory Cubes for Space Applications

Anthony Agnesina[1], Da Eun Shim[1], James Yamaguchi[2], Christian Krutzik[2],
John Carson[2], Dan Nakamura[3], and Sung Kyu Lim[1]
[1]School of ECE, Georgia Institute of Technology, Atlanta, Georgia, USA
[2]Irvine Sensors Corporation, Costa Mesa, California, USA
[3]NASA Jet Propulsion Laboratory, California Institute of Technology, Pasadena, California, USA
{agnesina,daeun,limsk}@gatech.edu
{jyamaguchi,ckrutzik,jcarson}@irvine-sensors.com
daniel.i.nakamura@jpl.nasa.gov

***Abstract*—In this work, we develop a new 3D flash memory cube architecture that integrates multiple flash dies and their logic controller in a unique and optimized fashion for space applications. In our Loaf-of-Bread (LOB) configuration, flash dies are standing up and bonded laterally instead of the conventional pancake-style vertical die stacking. Our LOB allows the flash dies to be bonded without the use of through-silicon-vias and micro-bumps. Instead, we insert a redistribution layer in between two adjacent dies to bring the IO signals to the bottom, where a logic controller die collects the IOs from all flash dies and coordinates the communication with an off-cube host processor. Thus, our LOB configuration allows the users to form flash memory cubes using off-the-shelf 2D flash dies and complete the integration at a packaging house instead of a fab. A key element in our LOB flash cube is the logic controller architecture that supports fault-tolerant and energy-efficient operation of the cube. We develop the controller architecture and validate the functionality using C++ emulation and FPGA prototyping. Compared with a state-of-the-art space-grade flash memory module, our system shows a 20X bandwidth improvement in a smaller form factor along with a 25X better ratio of density per volume.**

***Index Terms*—Aerospace and electronic systems, Computer architecture, Disk drives**

I. Introduction

Advances in non-volatile memory (NVM) have led to the ability of creating high density flash-based storage systems retaining information despite power loss. These systems have been readily accepted by the commercial market in the form of solid-state drives (SSDs). SSDs provide high memory density per unit volume, low power and weight without the downsides of their hard disk drives counterparts (e.g. mechanical parts). These attributes make them compelling for use in the space arena where more NVM applications for solid state recorders are being pursued. In fact, SSDs have already been used in a few space missions, including the Mars Global Surveyor (1996), Cassini (1997) and New Horizons (2006). However, they have not yet been widely adopted inside embedded systems for space, as the performance of space-qualified SSD controllers does not measure up to those of the enterprise server and consumer markets.

3D memory cubes have emerged over the last few years, presenting solutions to the end of Moore's law in terms of size, performance and power efficiency. Typical solutions in commercial applications such as Samsung's V-NAND [14] and IBM's High Bandwidth Memory [11] use vertical channels built into each die to connect chips together. However, the behavior and resilience of such channels in a space environment is yet unclear. The authors of [17] propose a space-qualified NAND flash memory cube, by stacking packaged chips connected through flip chip bonding at the bottom of the stack. In [1], the authors explore a novel way of stacking, arranging the dies in a vertical fashion to allow direct individual access to the bottom edge of each die. While these new developments make NVM much more attractive for use in space, the physical properties of NAND flash require a complex extra layer of processing and control, making their system integration difficult.

However, to the best of our knowledge, no specific controller architecture has been proposed to take advantage of the new cube characteristics as well as solve reliability and performance issues of outdated space SSD controllers. The contributions of this paper are as follows:

- We propose a new space-qualified SSD controller micro-architecture targeting 3D memory cubes that renders the memory system reliable while retaining state-of-the-art performance characteristics.
- Our novel architecture combines a flash translation layer (FTL) controller with RTL accelerators and MRAM caching to offload critical processor tasks. It also includes several error mitigation enhancements to address NAND flash deficiencies in terms of radiation tolerance.
- We design software and FPGA-based emulators to validate combined operation of the memory cube and RTL controller.

This research is funded by the NASA SBIR Grant under the contract number NNX17CP47P. A portion of the research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

2576-6996/20/$31.00 ©2020 IEEE
DOI 10.1109/ICCD50377.2020.00078

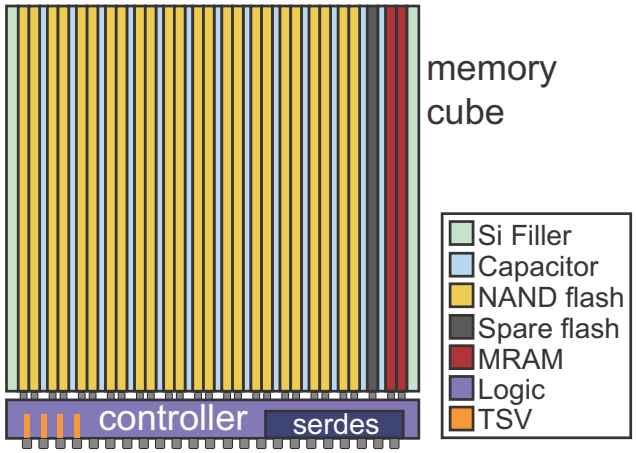

Fig. 1: Schematic view of our 3D NAND flash cube. The Loaf-of-Bread configuration allows heterogeneous stacking of COTS NAND and COTS MRAM dies into a single cube structure and enables individual die access for improved bandwidth and reliability.

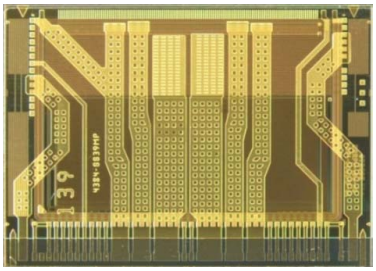

Fig. 2: Completed 2-metal redistribution layer, bringing IOs to the bottom of the die for electrically edge-connected attachment to the logic controller.

The rest of this paper is organized as follows. Section II reviews the overall cube and controller architectures as well as our novel solutions to classical SSD/FTL limitations. Section III presents RTL enhancements for built-in safety, scalability and power-up performance. Section IV shows how we integrate advanced memory management techniques to make the whole system more notably efficient and reliable. In Section V, we design a custom software emulator to confirm our architectural choices, we validate our RTL controller through simulation as well as verify the combined operation of our custom firmware and RTL controller on a FPGA platform. Finally, conclusions are given in Section VI.

II. Overview of our SSD Architecture

A. Overall Cube Structure

Our logic controller targets the memory cubes built using the technique developed by [1], where bare memory dies are stacked in a Loaf-of-Bread fashion. This configuration enables heterogeneous integration of different memory types such as DDR, NAND, MRAM, ReRAM or SRAM, as well as offers an individual, short and direct point-to-point interface to each die. This allows in particular parallelism in the execution of operations and individual die access for improved fault-tolerance.

The architecture of our 3D NAND flash memory system is depicted in Figure 1. Individual Commercial-Off-The-Shelf (COTS) single-level cell (SLC) NAND flash and MRAM dies are stacked at wafer level in a vertical configuration. Die IOs are routed to the bottom of the stack on a redistribution layer applied on each die as shown in Figure 2. High-frequency bypass capacitors are incorporated in-between the dies. The stack is then attached to a controller chip with integrated through-silicon vias (TSV) structure to allow bumping to the 3D stack (top side). A typical ball grid array interface (bottom side) allows the cube to be further packaged (e.g. in ceramic package) and connect to the underlying PCB and external host system.

B. Overall Controller Architecture

Figure 3 shows an overall view of our state-of-the-art SSD controller architecture centered around a logic driven implementation and integrated in the chip sitting underneath the cube. The controller is composed of the following components:

- A host interface connects a host computer to the SSD and allows multiple stacks to be chained together for scalability and achieve higher memory capacity.
- A system processor implements FTL mechanisms, performing I/O requests from the host and flash management procedures. Many of the processor tasks are accelerated by offload to RTL, an easy means to improve performance, reliability and energy consumption without resorting to complex software optimizations.
- A memory manager handles the data path, using external MRAM to cache host data and FTL data structures.
- A flash multiplexer/demultiplexer connects the FTL and MRAM cache decoupled command and data paths to each of the low-level NAND flash controllers.
- The low-level controllers communicate with the NAND flash chips in parallel for improved bandwidth and fault-tolerance.

A processor-driven implementation as found in consumer SSDs requires a very high-performance processor to handle high bandwidth in-line data processing, which complicates the design and increases power requirements. For this reason, we develop an all-logic data path to allow a low power/low resource processor to handle necessary functions. The logic driven implementation offers lowest power (minimize data movement), lowest gate-count and optimal performance, at the expense of flexibility and upgradeability.

C. Our Answers to traditional SSD Design Tradeoffs

The SSD architecture involves many design tradeoffs. Its central piece, the FTL, handles all the host commands and manages the NAND flash to maximize performance, reliability and endurance. It fills many duties such as the address mapping of the logical block addresses (LBA) of the data from the host to the physical page addresses (PPA) of the data in the flash, garbage collection (GC) that frees invalid memory space, wear-leveling that spreads flash writes across the entire die to prevent premature burn-out of blocks, and bad-block management that maps out corrupt blocks. A software-only

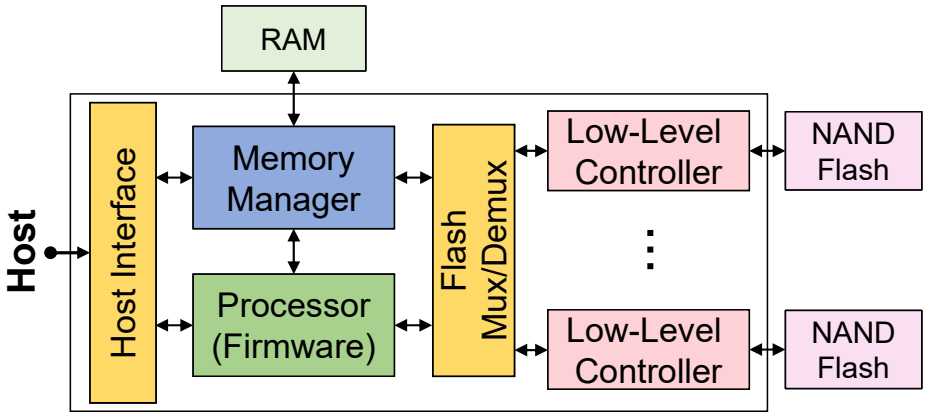

Fig. 3: Our flash-cube SSD controller architecture aiming to solve the limitations presented in Table I.

TABLE I: Our methods to increase performance, reliability and lifetime of space SSD systems, relying notably on the combination of LOB cube structure, RTL accelerators, and MRAM caching.

Aspect	Metrics of Interest/Limitations	Our Solutions
Performance	Bandwidth	• Parallel interface of LOB structure supported by individual low-level controllers • RapidIO cube network scalability
	Density	• LOB structure enabling much taller stacks • RapidIO cube network scalability
	Address mapping latency	• Accelerated hardware memory manager including caching
	CPU processing overhead	• Multi-level caching • RTL Block Merger inside each low-level controller
	Power-up wakeup time	• RTL bootloader • MRAM as main memory
Reliability	COTS NAND radiation weaknesses	• Advanced error mitigation techniques given in Table II
	Unsafe shutdown	• MRAM as cache and main memory
Lifespan	Normal wearing	• Hybrid-FTL offering efficient wear-leveling • Data compression and scrambling
	Write amplification	• Caching • Hot-cold data identification

implementation typically suffers many limitations in terms of performance and safety features, as described in [2]. Table I summarizes the main solutions that we present in the next sections.

D. Our Reliability Management for Space

SLC NAND flash shows improved radiation tolerance to total ionizing dose (20-150 krads (Si)) and single-event upset with the reducing of feature size, due to thinner oxide required for the device fabrication [9]. However, heavy ion and proton irradiations still induce upsets and NAND flash is susceptible to single-event effects (SEE) on the order of 10^{-11} bit/day. With advanced technology nodes NAND flash reliability is indeed at risk due to fewer electrons in the flash memory cell floating gate and larger cell-to-cell interference and disturbance effects [5]. Table II shows how our 3D memory module provides robust protection against all forms of failures of COTS NAND flash devices, including radiation induced in case of long term storage in high dose environments.

E. Comparison

Table III compares our solution with the Radiation Tolerant and Intelligent Memory Stack (RTIMS) from 3D Plus [4], which was integrated on the NASA Curiosity rover for Mars exploration. Compared with the RTIMS, our solution offers a significantly higher Density$\times$BandWidth/Volume ratio. Table IV estimates worst case power consumption of our memory solution, expected to remain under 9W. Note that high-performance SSDs from the consumer market of similar densities exhibit typical peak power consumption between 4-8W. In our case, the 9W value is highly pessimistic. Indeed, for example, MRAM current is heavily dependent on bank activation rate. The worst case scenario of 100% non-repeating bank access is mitigated with out-of-order execution of our MRAM memory controller. Moreover, the MRAM is mainly used for translation table lookup. Assuming sequential write access with input data rate of 800MB/s would require about 102,400 lookups per second and another 102,400 writes (assuming all misses) which requires less than 8% active mode.

TABLE II: Our mitigation of flash error sources.

Possible Error	Our Mitigations
Raw bit error (read and write disturb, charge loss, stuck bit, radiation event)	Bose–Chaudhuri–Hocquenghem (BCH) error-correcting codes (ECC)
Bit error accumulation	Scrubbing (read, correct and rewrite data)
Grown bad blocks	Block replacement
Die failure	Enable cold spare & rebuild using redundant array of independent disks (RAID)
Logical block table error	Reconstruct table using metadata
SEE causing bit upset during erase or write	Partial read-back and block/die recovery using RAID in case of failure
SEE during read	Embed address with CRC into spare area and validate on read-back
Controller SEE	Radiation hardened process & checksum for address/data path

TABLE III: Comparison of our solution with the RTIMS, which offers significantly higher bandwidth in a smaller form factor.

Metric	Our cube	RTIMS [4]
Package Size (mm)	25×25×15	28×28×10
Density	96GB	3GB
#dies	24	3
Peak Bandwidth	800MB/s	∼40MB/s
Density×BW/Volume (arb.)	8.2	0.015

TABLE IV: Worst case peak active power consumption based on values obtained from the datasheets.

Component	Power
DDR MRAM (write mode)	2×300mW [6]
NAND flash (write mode)	24×180mW [13]
Cortex-M3	20mW [3]
SerDes (4)	1.6W [8]
Controller	2W
Total	8.5W

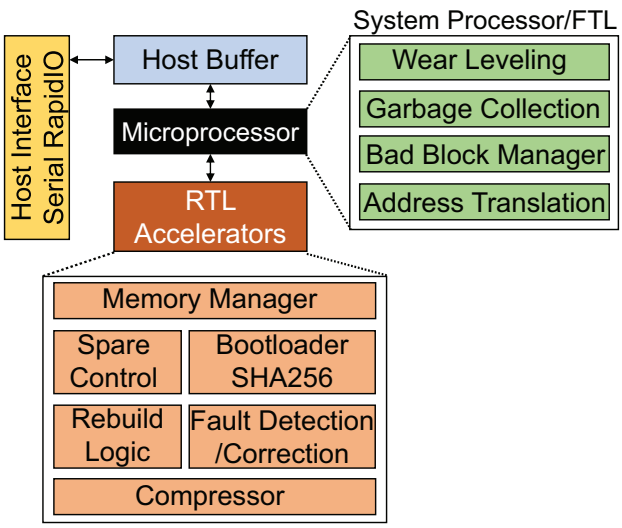

Fig. 4: FTL co-software/hardware architecture. RTL accelerator modules connect as custom Advanced High-performance Bus (AHB) peripherals to the CPU.

III. Data Management Improvements

Figure 4 shows the main pieces of our FTL co-software/hardware architecture integrating a low-power microprocessor (ARM Cortex-M3). While we implement the more complex tasks in software (e.g. external cache replacement policy, FTL algorithms), the entire movement (made through simple custom parallel interfaces with reduced signaling), protection and processing of the data is done in hardware. This requires a proper consideration of error-correcting codes, metadata storage, host interface, and data hashing and compression.

A. Data & Data Path Protection

Data path connections are controlled in hardware and necessary buffering is made through FIFOs to allow for fast, low-latency and non-stalling communications. The data is stored with strong BCH ECC on each page of the RAM cache and NAND flash, and passed through between modules with cyclic redundancy checksums (CRC).

B. Optimized Metadata

A spare area is dedicated in each page to store information necessary to rebuild the FTL logical block map and related data structures at startup. It is also used to store health metrics such as Program/Erase (P/E) cycles. A summary of our advanced metadata layout is shown in Figure 5. It holds ECC to recover bit errors and determine which pages are valid and which blocks are failure-free. The address metadata is used by the controller to validate page reads and the address translation table. The timestamp is useful for scrubbing validation and identification, and a data hash serves as additional verification of boot data.

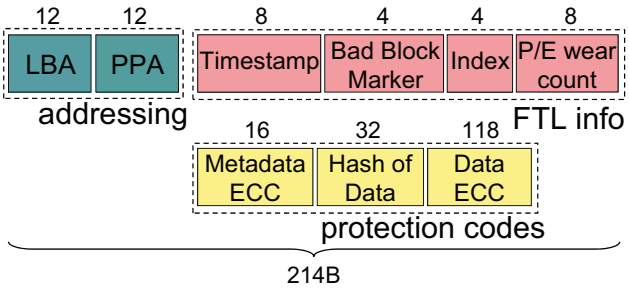

Fig. 5: Layout of metadata stored within each page, assuming 8KB page size SLC with 448B spare area and strong BCH(9130, 8192, 67) encoding of user data.

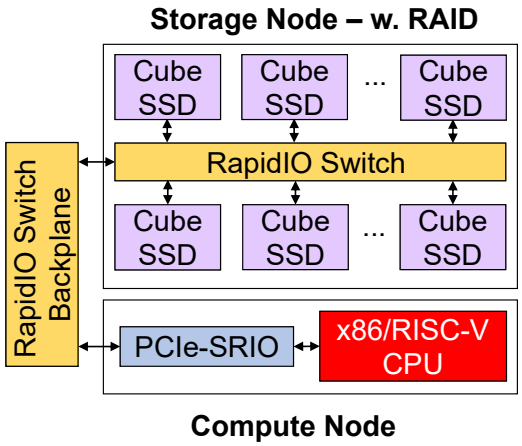

Fig. 6: Our scalable and reliable SSD cube network architecture, based on the space-qualified Serial RapidIO protocol. Each cube stores 96GB of data (24-layers).

C. A Host Interface for Reliability and Scalability

The host interface logic is based on the high-speed and fault-tolerant Serial RapidIO (SRIO) protocol. SRIO serves as a standard RTL-based interface, serving similar functions as SATA or NVMe, such as request scheduling. One principal advantage of SRIO is its various fault-tolerance features. In particular, it allows inter-module redundancy with no additional overhead required by the host processor, as the FTL is handled mostly by the controller of each memory module. A scalable and reliable network of cubes is then built as shown in Figure 6. Technology of RAID is implemented among the cubes, to enhance bandwidth and reliability, by spreading data and parity across multiple disks. This way, we protect from data loss in the event that one or more disks in the array fails completely. This is a valuable feature to enhance mission life since it allows for backup modules to take-over failed modules.

D. RTL Bootloader

A bootloader is stored as a read-only memory and preloads the CPU instruction and data caches. For reliability and fault prevention, the configuration data is stored in the lower block of each NAND flash with built-in ECC SHA encoding. To ensure that a valid copy was decoded, a hash table is stored at the end of the data and instruction sections. The hash table stores a SHA256 checksum for each page of the code section. We accelerate the booting process by implementing the SHA256 hash function in RTL. The system data is such that the P/E cycle limit of the flash is ensured by design, with no heavy writing on those pages (typically not a factory bad block). Each block is 1MB in size, which allows for system usage beyond the process code data.

E. Compression

We use a hardware GZIP compression engine, based on a systolic architecture and LZ77 algorithm presented in [15]. Compressing the data saves space and has the benefit of reducing write amplification and increasing write bandwidth. Note that not all data is compressed, and this feature is only used for radar imaging or audio in solid state recorders for example.

IV. A Hardware-only Memory Management

Our hardware accelerated memory system is shown in Figure 7. It includes components generally found in modern state-of-the-art microprocessor designs that are rarely put together in a SSD controller chip. A memory management unit (MMU) handles the orchestration of data movement between the FTL, the caches and the memory cube. Two levels of caching are used for better performance. First-level is a non-blocking 4-way associative SRAM cache with pseudo-tree least recently used (LRU) policy. The SRAM content is protected by a Hamming code single-error correction double-error detection on each of the two ports. The last-level cache is the MRAM which also serves as main memory for the FTL processor. Caching speeds up the FTL tasks where frequent mapping table lookups and updates are necessary (address translation, bad block, free block list, metadata for health metrics, etc.) and increase response time to the host for NAND bare data read requests (e.g. hot pages), since fetching from the flash requires more time.

A. Our Address Translation Scheme

To emulate a logical disk from the host point-of-view, the FTL performs a mapping from the LBA of the host to the PPA inside the NAND.

1) Hybrid Mapping: For mapping, we use a static scheme provided at a page-granularity in page order by a block sequencing from die-to-die. On each die, we use the hybrid log-structured mapping of [10] where writes are appended to the next free spot in the block that is currently being written to. This way, writes are spreaded across all pages, performing wear leveling and increasing the lifetime of the devices in the cube.

2) Mapping Acceleration: A Translation-Lookaside Buffer (TLB) caches the last LBA-PPA mappings. In case of a miss, a hardware Page-Table Walker (PTW) "walks" the multi-level Page Table Entry (PTE) stored in the L1 cache or the MRAM to look for the mapping. If the mapping exists, it automatically loads the translation into the TLB. Otherwise,

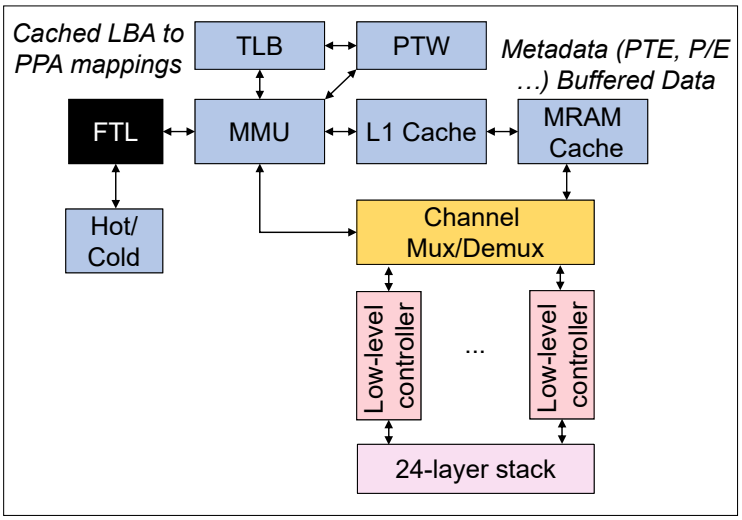

Fig. 7: Memory management in our SSD architecture, based on high-performance CPU architecture techniques.

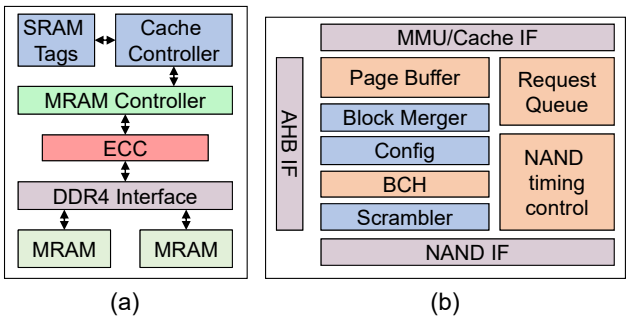

Fig. 8: (a) MRAM cache architecture, (b) low-level controller architecture.

the FTL is notified which in turn provides a mapping to the never-allocated page. For robustness against SEEs, parity bits are added to the TLB contents. The TLB is also cached in MRAM so that the CPU can restore the corrupted state of the TLB from the MRAM in case an error is detected.

B. MRAM Caching & Buffering

We use the MRAM to store various controller metadata (e.g. translation table, block map table) and frequently accessed/hot pages. The organization of the MRAM cache is shown in Figure 8(a). State-of-the-art DRAM caches store tags in the DRAM itself [16]. Instead, we store tags on-chip for low latency of tag lookups, which in turn incurs the need of a large on-chip SRAM. Two MRAM dies are used in parallel to protect against single event effects (the data is stored with ECC and mirrored on writes and down-selected on reads), and connected to the cache controller through a DDR4 interface. The non-volatile nature of MRAM optimizes power-up and power-down sequencing and protects from sudden power failures. These MRAM features optimize system energy consumption, reduce the overhead of the controller (less frequent journaling/checkpointing) and increase the lifetime of the device: block table read/write operations from/to the NAND flash array are done only when necessary, which reduces write amplification. The MRAM device exhibits small radiation effects at the storage cell level, as well as unlimited endurance and extreme retention characteristics (20+ year) [7].

C. Hot-Cold Data Identification

We implement an efficient hot page identification scheme presented in [12] in RTL, originally designed for software implementation. It consists of two fixed-length LRU lists of LBAs (hot list and candidate hot list), updated based on write requests LBAs. This scheme is used to improve garbage collection necessary to enumerate previously-used blocks that must be erased or recycled. Our GC policy avoids copying hot-live data, as pages that store hot data usually have a high chance of becoming invalid in the near future.

D. Low-Level Controllers

To obtain high bandwidth and low latency, we implement one channel controller per NAND device (= flash die), taking advantage of the LOB cube structure offering individual die connections. We operate dies in parallel, i.e. they are accessed concurrently and carry out different operations independently. This way, GC of multiple blocks are done in parallel when needed. The block diagram of each low-level controller is shown in Figure 8(b). When data is sent from the MMU/last-level cache to the controller, it is buffered in its own internal queue, rather than being stalled in the cache. This provides increased performance and reduces cache accesses. Our low-level controller architecture includes in particular the following features.

1) Data Scrambling: We scramble the data so that stored zeroes and ones are equally distributed to optimize binary data distribution. This reduces cells interference with adjacent bits (read/write disturb). This also aids in an even distribution of cell voltages for each bit over the lifetime of the flash, reducing

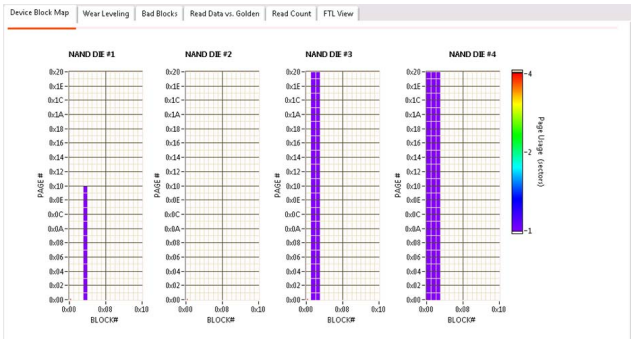

Fig. 9: Custom interface C++ software for SSD architecture testing.

```
Init processor core on module #1
  build free blocks...
Adding free block, addr=0
Adding free block, addr=20
Adding free block, addr=40
Adding free block, addr=60
  build bmt...
[processor::task_gc_build_bmt] Repairing inconsistent NAND bmt index: 0
  done init.
Init processor core on module #2
  build free blocks...
Adding free block, addr=0
Adding free block, addr=20
Adding free block, addr=40
Adding free block, addr=60
  build bmt...
  done init.
erase command: addr=0 len=16384
```

Fig. 10: Simulation of the repair of an inconsistent block map table at powerup.

the stress on the tunnel oxide. This in turn maximizes flash life. The scrambling and descrambling is based on a linear feedback shift register, which uses the LPA as a seed value so that descrambling is possible even after a page has been moved (e.g. using internal copy-back operation).

2) Block Merger: Upon exhaustion of log blocks where updated data is stored, the hybrid mapping requires expensive merging operations in order to allocate new log blocks. Merging will reunite pages from many blocks through page copies and block erases and is a critical performance bottleneck. To limit the CPU processing during merging that would involve passing around large pages through the Channel Mux/Demux from a NAND device to an other, we perform these expensive merge operations during GC inside each low-level controller in hardware.

V. Verification of the Proposed Architecture

A. Software Emulator

In order to test the correct functioning of the FTL firmware and help in the design of the RTL controller, we develop a custom C++ trace-driven emulator of the flash controller, NAND stack with MRAM caching and SRIO daisy-chainable interface. While executing the trace, we observe the status of each die in the stack using the monitoring GUI shown in Figure 9. This helps demonstrate functionality of our various flash management features (firmware with GC, wear leveling, etc.), the capability to recover flash corruptions (e.g. rebuild table in case of power-loss scenario as shown in Figure 10), and optimizes memory tables and processor interaction. Furthermore, the code stores a "golden" copy of all data transactions to assist debugging and perform runtime validation of operations on the cube.

TABLE V: Design metrics of the FPGA SSD controller. The ARM processor and host interface are mapped onto a Cyclone V while the rest of the design is placed onto a Virtex Ultrascale.

Module	LUTs	FFs	Memory bits
on Intel Cyclone V			
Cortex-M3	22k	16.6k	3.9M
Serial RapidIO	9.6k	10k	150K
on Xilinx Virtex UltraScale			
Memory Manager	14.9K	14.7K	5.5M
Compressor	28K	46K	33K
SHA256	2.1K	1.8K	0
24 NAND controllers	82K	39K	860K
Total	158K	128K	10.4M

B. RTL Simulation & FPGA Implementation

We perform extensive Verilog simulations of most hardware functions by building advanced testbenches for each module. Figure 11 shows correct parallel operations of the low-level controllers connected to NAND Verilog models from Micron. We verify the operation of the firmware with the bare-metal code exercising the AHB RTL peripherals correctly.

We implement the Cortex-M3 and Serial RapidIO interface on Intel Cyclone V due to licensing reasons. We build the prototype board shown in Figure 12 to test the processor. It comprises a single socketed flash package connected to an expansion header on a ARM MPS2+ board. The rest of the SSD controller including the 24 low-level controllers and RTL accelerators is mapped onto a Xilinx XCVU440 FPGA. The fabric utilization of the FPGA resources is shown in Table V. Figure 13 shows the two aforementioned mappings.

VI. Conclusion

SSD performance is impacted by many design tradeoffs due to the circuit limitations of NAND flash. NAND flash also suffers from reliability issues such as cell wear-out, charge leakage, etc. These issues are intensified in the space radiation environment. In this work, we propose a mostly hardware space SSD controller architecture for a new 3D NAND flash memory cube, where dies can be operated in parallel. The novelty of our work is to combine a FTL controller with RTL acceleration and MRAM cache in a single architecture. The enhancements we propose in terms of reliability and high-speed features help SSDs stand out as the most sensible option for data storage in modern interplanetary spacecraft.

References

[1] A. Agnesina et al. A Novel 3D DRAM Memory Cube Architecture for Space Applications. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, June 2018.

[2] N. Agrawal et al. Design Tradeoffs for SSD Performance. In *USENIX 2008 Annual Technical Conference*, ATC'08, pages 57–70, Berkeley, CA, USA, 2008. USENIX Association.

[3] ARM. ARM Cortex Series Documentation.

[4] M. Bagatin et al. SEE Tests of the NAND Flash Radiation Tolerant Intelligent Memory Stack. In *2015 IEEE Radiation Effects Data Workshop (REDW)*, July 2015.

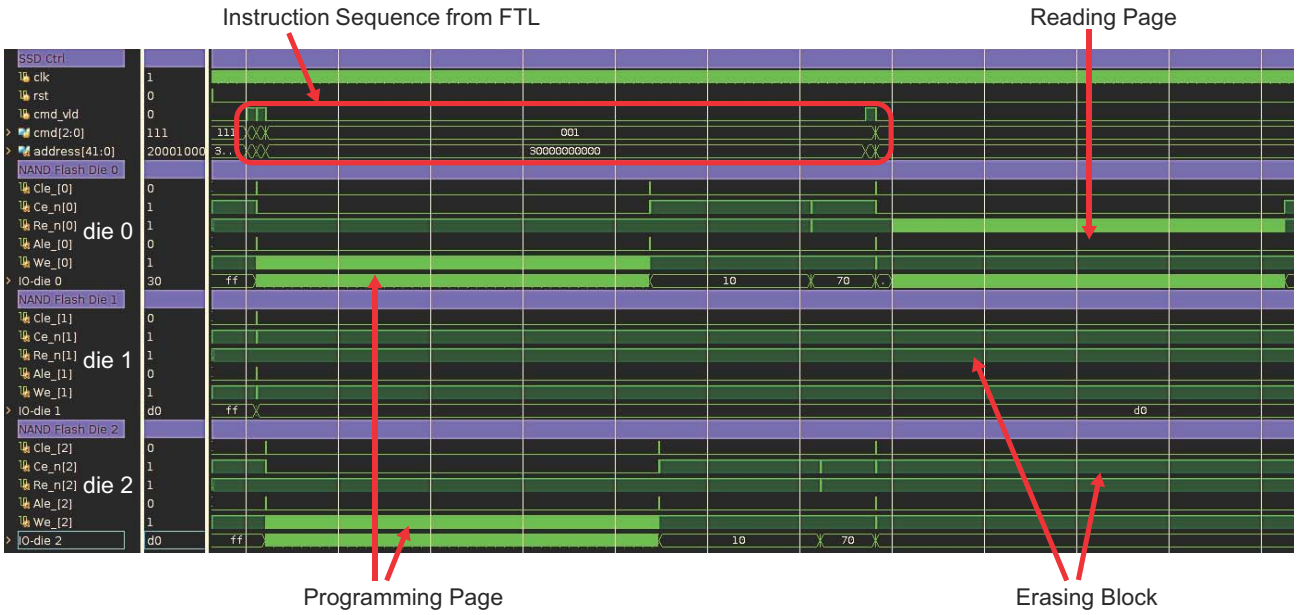

Fig. 11: Simulation of the parallelism of low-level controllers, where dies are operated concurrently.

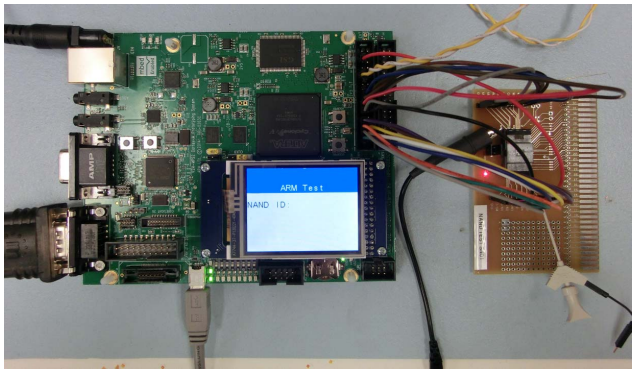

Fig. 12: Cortex-M3 based prototyping system with Cyclone-V FPGA.

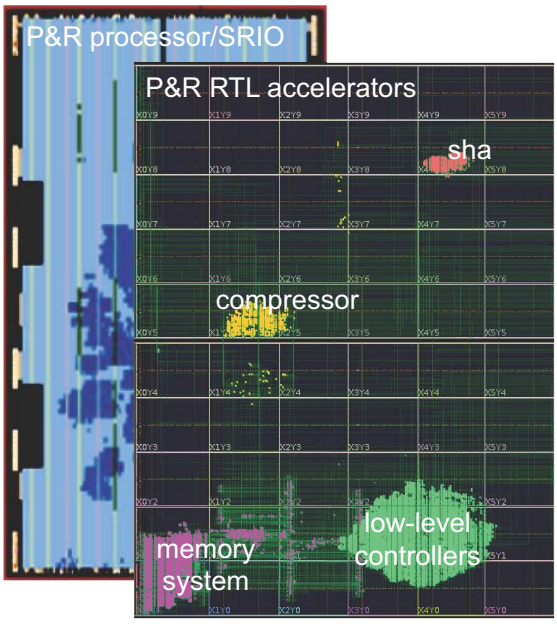

Fig. 13: FPGA layouts of SSD controller for memory cube.

[5] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu. Errors in Flash-Memory-Based Solid-State Drives: Analysis, Mitigation, and Recovery. *ArXiv*, abs/1711.11427, 2017.

[6] Everspin. Using MRAM to Optimize System Energy Consumption.

[7] J. Heidecker. MRAM Technology Status, 2013.

[8] Honeywell. HXSRD02 Slider 1x/4x sRIO PHY and SERDES Quad Transceiver Radiation Hardened.

[9] F. Irom et al. Single Event Effect and Total Ionizing Dose Results of Highly Scaled Flash Memories. In *2013 IEEE Radiation Effects Data Workshop (REDW)*, pages 1–4, July 2013.

[10] J. Kim et al. A space-efficient flash translation layer for CompactFlash systems. *Consumer Electronics, IEEE Transactions on*, 2002.

[11] D. U. Lee et al. 25.2 A 1.2V 8Gb 8-channel 128GB/s high-bandwidth memory (HBM) stacked DRAM with effective microbump I/O test methods using 29nm process and TSV. In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, Feb 2014.

[12] Li-Pin Chang et al. An adaptive striping architecture for flash memory storage systems of embedded systems. In *Eighth IEEE Real-Time and Embedded Technology and Applications Symposium*, Sep. 2002.

[13] Micron. *NAND Flash Die 32Gb Die: x8 300mm SLC, MT29F32G08AB*.

[14] K. Park et al. Three-Dimensional 128 Gb MLC Vertical nand Flash Memory With 24-WL Stacked Layers and 50 MB/s High-Speed Programming. *IEEE Journal of Solid-State Circuits*, Jan 2015.

[15] O. Plugariu et al. FPGA systolic array GZIP compressor. In *2017 9th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*, June 2017.

[16] M. K. Qureshi et al. Fundamental Latency Trade-off in Architecting DRAM Caches: Outperforming Impractical SRAM-Tags with a Simple and Practical Design. In *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 235–246, Dec 2012.

[17] Tak-kwong Ng et al. Radiation tolerant intelligent memory stack (RTIMS). In *2nd IEEE International Conference on Space Mission Challenges for Information Technology (SMC-IT'06)*, July 2006.