Enhancing eMMC with Multi-Stream

Sushma Vishwakarma
Memory Division
Samsung Semiconductor India R&D center
Bangalore, India
sushma.v@samsung.com

Anantha Sharma
Memory Division
Samsung Semiconductor India R&D center
Bangalore, India
anantha.sk@samsung.com

Sharath Kumar Kodase
Memory Division
Samsung Semiconductor India R&D center
Bangalore, India
sharath@samsung.com

***Abstract*— Multi-stream for SSD is a concept where the host writes data with similar expected life times to contiguous blocks of NAND memory. Consequently this data has higher chances of being invalidated together. During garbage collection (GC), there will be minimal valid pages to copy, resulting in improved endurance and decreased write-amplification factor.**

Implementing Multi-stream in eMMC devices presents challenges due to lower DRAM, lower computational resource available. Maintenance of stream related information requires increase in DRAM usage, transfer buffer usage as well as addition computation for GC of stream related memory area. In this paper, we examine implementation of Multi-Stream concept in eMMC despite its low resource constraints. We have experimented with an implementation that supports up to 4 streams which uses additional ~108 bytes of DRAM and ~104 bytes of code section, and improves WAF by ~50% on FIO (File Input/Output) benchmarking setup where lifetimes are simulated by multiple instances of FIO.

Keywords—component, formatting, style, styling, insert (key words**)**

I. INTRODUCTION

In this paper we are suggesting how to use multi-stream with eMMC to improve endurance of device. eMMC (Embedded Multimedia Card) is an embedded storage solution; with an MMC interface, flash memory & controller, all in a BGA package. This NAND flash based eMMC is widely used for high performance applications such as mobile phone, smart phone, tablet computer, notebook computer, and automotive segment etc. eMMC provides fast scalable performance with interface speeds of up to 400Mbps.

II. BACKGROUND

eMMC chip is made up of NAND flash-memory, which is architecturally partitioned into blocks, and each block contains a fixed number of pages. A page, which consists of a data area to store user data & a spare area to store housekeeping data such as error correction code, status flags, etc. Due to Hardware characteristics of flash memory, data cannot be directly updated to their residing pages unless their belonging blocks are erased first. As a result invalid pages (page having out-of-date data) should not be read & should be recycled when the number of free pages are low. Because of this out-of-place update policy, there is a need to map the logical block address (LBA) to its corresponding physical block address (PBA) over flash memory. This mapping is done by FTL (Flash Translation Layer) that runs on a controller. Each block can survive over a limited number of program/erase cycles (P/E).

The flash memory space of eMMC consists of Blocks and each block consists of a fixed number of Pages. A Block is the smallest unit for erase operations, while a Page is the basic unit of reads and writes. A basic size of a page is 4KB, and the size of a block is 48MB, which is, dependent on NAND. The current single level cell NAND (SLC) generally supports 100,000 P/E cycles (depending upon the used flash technology), and multiple-level cell NAND (MLC) supports approximately one tenth of what SLC can achieve.

NAND flash memory constraints can be summarized as: (1) Write/erase granularity asymmetry: writes are performed on pages while erase operations are executed on blocks. (2) Erase-before-write rule: one of the most important constraints as one cannot modify data in-place. A costly erase operation must be achieved before data can be modified in case one needs to update data on the same location. (3) Limited number of Write/Erase (W/E) cycles. After the maximum number of erase cycles is achieved, a given memory cell becomes unusable.

III. DESIGN IDEA

A stream is an abstraction of eMMC capacity allocation that stores a set of data with the same lifetime expectancy. Multi-stream is a concept where we can group data stream into multiple streams, where all the data belonging to a particular stream will have almost same lifetime. Below example will describe how this stream can address the eMMC aging problem. Below figure gives the example where two NAND block (4 pages per block) have been filled up and new data are written to fill Block 2.

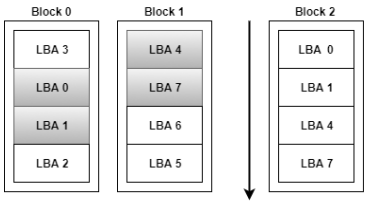

Figure 1.1 Write pattern (0-1-4-7) is applied, and as the result, some data become invalid in Block 0 and Block 1

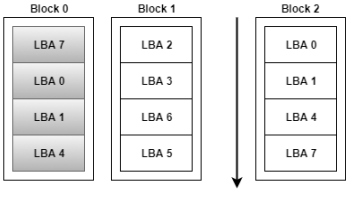

Figure 1.2 Write pattern (0-1-4-7) is applied, invalidating all data in Block 0 but none in Block 1

Clearly, from figure 1.2, future GC will proceed more efficiently because an empty NAND flash block (Block 1) can be reclaimed quickly without copying data around. These

examples demonstrate that an eMMC's GC overheads depend not only on the current write pattern, but also on how data have been already placed in the eMMC.

An eMMC that implements the proposed multi-stream interface allows the host system to specify the lifetime of data in the form of stream-id. A multi-streamed eMMC allocates physical capacity carefully to place data in a stream together and not to mix data from different streams.

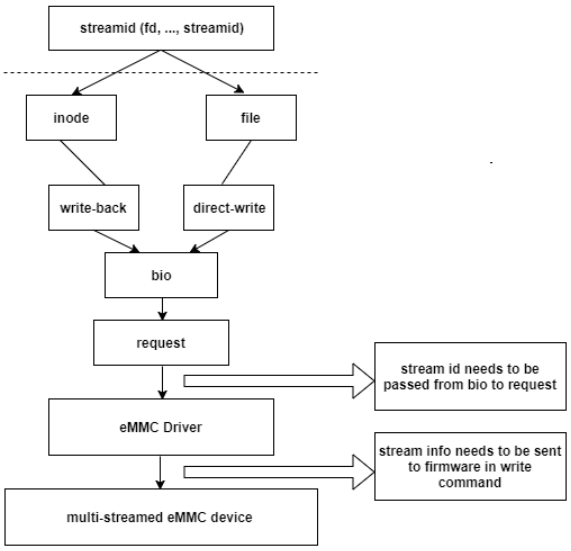

Figure 2 Application is passing stream id for eMMC, through system stack

All the data associated with a stream is expected to be invalidated at the same time, e.g., updated, trimmed, unmapped and de-allocated.

IV. IMPLEMENTATION

The challenge for implementing multi-stream in eMMC is due to its limited computational resource & DRAM and transfer buffers. Transfer buffers are an area of memory that hold data in transit, either from the host or from the NAND. Data is moved to transfer buffers by way of DMA.

In any implementation of multi-stream, we need to maintain stream information for each block. This stream information needs to be maintained alongside the data for as long as it lasts, until it is unmapped or overwritten. The stream information determines which blocks in NAND that data associated with it will be written to. Both for host-writes as well as card initiated writes such as GC. Maintenance of this information for each block, however, increases on-disk meta size as well as increasing complexity of GC.

Our approach to address these requirements is by limiting maintenance of the stream information only during host write operations. In this, we allocate 'Active Block' for each streams separately and assign stream-ids for them respectively. Active Block is a NAND block with "write operation in progress" state. Incoming data is written into these active blocks according to their stream-id. A transfer buffer will be associated with each Active block to hold the data in transit. Once active block is completely written, we add it into list of data blocks. Data structure for Active block is part of global context of FTL and it contains block index, page offset, program level etc. total of 27 bytes, which are needed for writing data into NAND pages. Global context information will be written into NAND over period, so that in case of Sudden Power off scenario we can recover to previous state.

As number of stream-ids increases, there will be corresponding increase in DRAM usage size by same number of active block data structure as well as the number of transfer buffers required. This limits the number of stream-ids supported by the FTL. Currently based on the number of Active blocks we can have in memory, our implementation sets the number of streams supported to be 4.

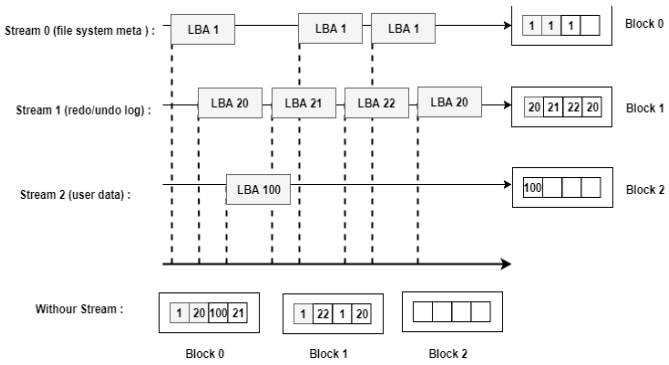

Figure 3 Multi-streamed eMMC writes data into a related NAND block according to stream ID regardless of LBA. Three stream are introduced to store different types of host system data

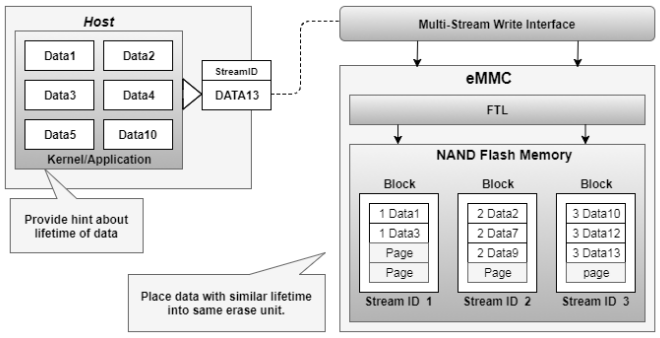

Figure 4 Mapping Data with same lifetime to corresponding stream-id blocks

During GC, stream-id information is ignored. Data belongs to all stream-ids are considered as a single pool of data blocks and GC victim block is chosen as if there is no change because of multi-stream. Same mechanism is used for free block list as well, hence irrespective of difference in amount of total data belonging to any of particular stream, we can allocate blocks as per existing policy. Since data with similar lifetime is residing in a NAND block, it is expected that all the data are going to be erased almost at same time and hence GC operation will have minimal overhead for

Identify applicable funding agency here. If none, delete this text box.

copying of valid pages to a new NAND block. However, this is not ideal. It is possible in this implementation that during GC, data from different streams will get written to same GC destination block, decreasing the efficiency of multi-stream implementation.

System configuration	Fio configuration
Hardware setup: V310 SMDK eMMC sample: 64 GB	I/O workload : 0% read, 100% write
Software : Linux kernel : Linaro 3.18.57 Preconditioning : Secure erase	3 parallel jobs with different data lifetime; 2X, 101X & 709X

To improve efficiency, the application or host needs to judiciously use stream-ids, either by applying application specific intelligence as to know what data will have similar life-times or by applying machine learning to classify data to different stream-ids.

To associate data with a stream-id, JEDEC eMMC SPEC 5.1 has no support. Hence we are using context-id field (4 bits: B25-B28) of CMD23 for passing stream-id information from host to device while write operation. To distinguish between context-id and stream-id, we are using reserved field in EXT_CSD register (byte 133). With this interface, host will send stream-id to the device, and can this filed can support up to 15 streams from interface point of view where id "0" is considered as normal data which has not assigned any stream-id number.

V. LIMITATIONS IN EMMC

Performance of eMMC also depends on fill factor of device, again it is dependent on firmware design policies and implementation. GC operations are high when the device is almost full with data. This is mainly because all the data blocks will have many valid data pages and few invalid pages and number of overprovisioned blocks affects this ratio. Active blocks uses these overprovisioned blocks and hence increasing number of streams degrades worst-case Iglobal context and when the fill factor is above 90% then we can de-activate multi-stream feature through which worst-case performance is kept intact.

The eMMC device uses Transfer buffers to cache the data written by host, which accumulates random pages and does an interleaved program to NAND to boost the performance. In case of multi-stream, we have to accumulate the data in separate buffers attached with each stream-id since each transfer buffer is attached to active blocks. Having more stream-id hinders performance by creating buffer crunch for cache operation. With our experiments, we could observe gain in performance up to 4 stream-ids and after which buffers are insufficient for read-write operations.

VI. BENCHMARKING ENVIRONMENT

Benchmarking numbers taken by running FIO for both multi-streamed eMMC and without multi-streamed eMMC. For multi-stream support FIO ran for three different stream-ids.

Table 1 Benchmarking Environment

VII. RESULT

FIO benchmarking analysis after ~ 1400GB writes.

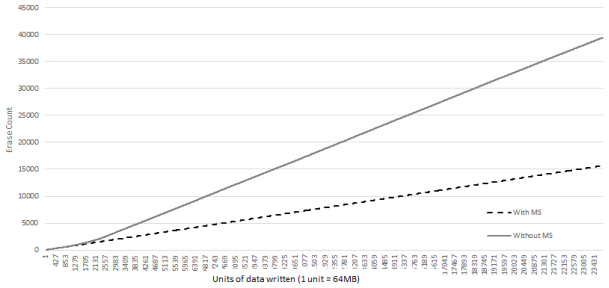

Figure 5.1 Erase Count MLC

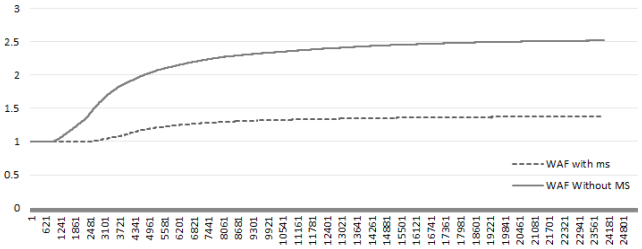

Figure 5.2 Write Amplification Factor

VIII. REAL TIME USECASES

Benchmarking results shows that an intuitive data to stream mapping can lead to consistent latency and better NAND flash lifetime on the multi-streamed eMMC. We further believe that many applications and use cases can get similarly large benefits using multi-streamed eMMC. Consider log bases Database management system like Cassandra, SQLite4, RocksDB and many more. Similarly, some multi-head log-structured file system like F2FS maintains data separation based on their update frequencies. These are applications that explicitly manages data streams & orient their IO to be sequential.

A. Flash Friendly File System

Flash Friendly File System (F2FS) supports hot and cold data separation in File System. At runtime, F2FS manages six active logs inside the "Main Area:" Hot/Warm/cold for node and data; depends on their update frequencies. The detailed definition of hotness in F2FS is listed below:

Table 2 Block Allocation policy

Type	Update Frequency	Contained Objects	Stream id
Node	Hot	Directory's inode block or direct node block	1
	Warm	Regular file's inode block or direct node block	2

	Cold	Indirect node block	3
Data	Hot	Directory's data block	1
	Warm	Updated data of regular files	2
	Cold	Appended data of regular files moved data by cleaning, multimedia file's data	3

B. SQLite Data Base

Database are more commonly used in embedded system applications & especially in consumer electronics. SQLite is one of the most used database application. It uses nine types of temporary files & each file has different lifetime. That can be used for assigning stream id.

Table 3 Temporary files used by SQLite

Sq	SQLite File	Life Span	Stream id
1	Rollback Journal	Created and destroy at each transaction	3
2	Master Journal	Created only for commit operation	3
3	Write-ahead Log (WAL) Files	Created at first connection of DB and remains till the last connection	2
4	Shared Memory Files	Lifetime same as WAL	2
5	Statement Journals	Creates for update or insert statement	3
6	Temp database	Created using "create temp table" deleted when DB connection close	2
7	Materializations of views and subqueries	Temporary tables created by materialization are each stored in their own separate temporary file, which is automatically deleted at the conclusion of the query	2
8	Transient Indices	Temporary file for a transient index; automatically deleted at the end of the statement that uses it.	3
9	Transient DB used by VACCUME	Temporary file created by the VACUUM command exists only for the duration of the command itself.	3

From the Table 3 it is clearly visible that these temporary files can be nicely map into stream for multi-streamed.

IX. FUTURE WORK

Effectiveness of Multi-stream on endurance of eMMC mainly depends on how good the data is mapped to streams based on lifetime. Some of the applications like database engines file system page cache, etc., where lifetime defined clearly, can identify stream-id for data. Challenge comes when multiple applications has to adhere mutually acceptable stream-id mappings since the performance is highly depends upon similarity of data lifetimes in single stream.

Machine Learning can be applied to determine streams, where clustering of Logical Block Addresses are made based on frequency of overwrites on range of LBA compared to average frequencies on complete device. Since eMMC resource constraint system, these algorithms can run in host driver as well and pass on the information.

X. CONCLUSION

Benchmarking results shows that intuitive data to stream mapping can lead to large benefits in throughput, consistent latency and NAND flash lifetime on the multi-streamed eMMC. There is significant improvement in Endurance as WAF for multi-streamed eMMC reduced by ~50%. We believe that many applications and use cases will get similar large gains from multi-streamed eMMC if the host provides appropriate lifetime information of the incoming data.

REFERENCES

[1] Feng Chen, David A. Kaufaty, and Xiaodong Zhang, "Understanding Intrinsic Charactristics and System Implications of Flash Memory based Solid State Drives".

[2] Jeong-Uk Kang, Jeeseok Hyun, Hyunjo Maeng, and Sangyeum cho "The Multi-streamed Solid-state Drive"

[3] eMMC Flash Programming User's Guide "http://www2.lauterbach.com/pdf/emmcflash.pdf", November 2018

[4] JEDEC eMMC Electrical Standard 5.1 "https://www.jedec.org/sites/default/files/docs/JESD84-B51.pdf" February 2015