Improving MLC flash performance and endurance with Extended P/E Cycles

**Fabio Margaglia**, André Brinkmann
Johannes Gutenberg University, Mainz, Germany
Motivation

• Flash wear out is dependent on the number of erase operations

• Many efforts to reduce the erase operations
  – Reuse pages through special encodings (WOM codes) and data structures that exploit the flash properties
  – Proven for SLC or in simulation
  [Yagmohan et. al, MSST 2010, Odeh et al., MSST 2014, Yadgar et al., FAST 2015]

• Non of these techniques have ever been implemented within MLC flash environment
  – Is this possible?
  – What would be the constraints?
Normal P/E Cycles

Page: read/program granularity
Block: erase granularity
Must erase before program (arbitrary data)

P/E Cycle:
1. Program all pages in order, one time
2. Erase entire block

Problems:
1. Erase is slow
2. Need to copy valid pages
3. Wear the flash
Extended P/E Cycles

Extend the P/E cycles:

• Reprogram pages multiple times per cycle

(+) Reduce the number of erase operations
(+) Reduce internal copying: improve performance
(+) Reduce flash wear out: improve endurance

(~) Reprogram pages is not trivial in SLC
(--) Even more difficult in MLC

Our contribution:
Enable Extended P/E Cycles on MLC
Every SLC is characterized by $V_{th}$

- program operations increase $V_{th}$
- erase operations reset $V_{th}$
- single read point $R_1$

Distinguish between 2 states $\Rightarrow$ 1 bit of information
- Low $V_{th}$: ‘1’
- High $V_{th}$: ‘0’

Condition for page reprograms: **never 0 to 1**

**Write Once Memory Constraint**
Write Once Memory Codes

WOM Codes:
• encode $n$ bits with $m$ wits, where $m > n$
• encoding organized into WOM compatible generations

Example:

<table>
<thead>
<tr>
<th>Plain bits</th>
<th>1&lt;sup&gt;st&lt;/sup&gt; gen</th>
<th>2&lt;sup&gt;nd&lt;/sup&gt; gen</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>111</td>
<td>000</td>
</tr>
<tr>
<td>01</td>
<td>110</td>
<td>001</td>
</tr>
<tr>
<td>10</td>
<td>101</td>
<td>010</td>
</tr>
<tr>
<td>11</td>
<td>011</td>
<td>100</td>
</tr>
</tbody>
</table>

[Rivest and Shamir, 1982] (originally expressed with 0 to 1 constraint)
WOM compatible data structures

**B-Tree:**
- Initialize nodes with **all 1s**
- **Append** new keys, values, pointers
  [Kaiser et. al, SYSTOR 2013]

```
key1, key2 111111111111111
Reprogram
key1, key2, key3 1111111
```

**Bloom Filters:**
- Based on bitmap WOM compatible by construction
  [Bloom, 1970]
What about MLC?
Every MLC is characterized by $V_{th}$

- program operations increase $V_{th}$
- erase operations reset $V_{th}$
- 3 read points $R_1$, $R_2$, $R_3$

Distinguish between 4 states $\Rightarrow$ 2 bits of information
Called Low Bit and High Bit

Condition for page reprograms: WOM constraint is not enough
• Program disturbance between Low Bit and High Bit

[Grupp et. al, MICRO 2009]
MLC – mapping bits to pages

The bits of a single MLC are mapped to 2 independent pages

Program disturbance **across** pages

$V_{th}$ diagram is **not the best tool** to understand program disturbance
State Diagram

Extracted from Samsung 35nm MLC chip

• **Bubbles** = flash states with bit values
• **Transitions** = program operations on single bit
  e.g. P 1 LB = program ‘1’ on the Low Bit
State Diagram

5 States, 2 with the same bit values

- **Bubbles** = flash states with bit values
- **Transitions** = program operations on single bit
e.g. P 1 LB = program ‘1’ on the Low Bit

Diagram:

- **Erased State**
  - Low Bit: ‘1’
  - High Bit: ‘1’

- **State 1**
  - Low Bit: ‘0’
  - High Bit: ‘1’

- **State 2**
  - Low Bit: ‘1’
  - High Bit: ‘0’

- **State 3**
  - Low Bit: ‘0’
  - High Bit: ‘0’

- **State 4**
  - Low Bit: ‘0’
  - High Bit: ‘1’

- **State 5**
  - Low Bit: ‘1’
  - High Bit: ‘1’

**Transitions**

- (0) P 1 LB
- (1) P 0 LB
- (2) P 0 HB
- (3) P 1 LB
- (4) P 0 HB
- (5) P 1 HB
- (6) P 0 LB
- (7) P 0 HB
- (8) P 1 HB
- (9) P 0 LB / P 1 HB
- (10) P 0 HB / P 1 LB
- (11) P 1 HB / P 0 HB
State Diagram: Program the Low Page

Erased State
Low Bit: ‘1’
High Bit: ‘1’

State 1
Low Bit: ‘0’
High Bit: ‘1’

State 2
Low Bit: ‘1’
High Bit: ‘0’

State 3
Low Bit: ‘0’
High Bit: ‘0’

State 4
Low Bit: ‘0’
High Bit: ‘1’
State Diagram: Faulty Transitions

Erased State
Low Bit: ‘1’  
High Bit: ‘1’

State 1
Low Bit: ‘0’  
High Bit: ‘1’

State 2
Low Bit: ‘1’  
High Bit: ‘0’

State 3
Low Bit: ‘0’  
High Bit: ‘0’

State 4
Low Bit: ‘0’  
High Bit: ‘1’

Transitions:
1. P 0 HB → P 0 HB
2. P 1 HB → P 1 HB
3. P 1 LB → P 1 LB
4. P 0 LB → P 0 LB
5. P 1 LB / P 0 HB
6. P 0 LB → P 0 HB
7. P 0 HB → P 0 HB
8. P 1 HB → P 1 HB
9. P 0 LB / P 1 HB
10. P 1 LB / P 0 HB
11. P 0 LB / P 1 HB

States:
- Erased State
- State 1
- State 2
- State 3
- State 4
WOM constraint alone does not avoid all these transitions
Our solution

Bimodal usage that allows:

• Page reprograms (only the low pages)
• Uses both high and low page
• Compatible with WOM constraint
Bimodal Usage

1\textsuperscript{st} mode: reprogram all low pages with WOM constraint

2\textsuperscript{nd} mode: - stop reprogramming the low pages (still readable) 
- program the high pages once (any value)
Prototype

Jasmine OpenSSD board
www.openssd-project.org

- ARM7TDMI single core
- SATA interface
- 64MB DRAM
- Samsung 35nm MLC
Interface augmented:
- read
- write
- overwrite

The application generates WOM compatible data

Other approaches do not modify the interface and use WOM codes internally (too heavy for the Jasmine controller) [Yadgar et al., FAST 2015]
Baseline Flash Translation Layer

- Log structured
- Page Mapped
- Garbage Collection with greedy strategy

- Overwrite commands are considered as writes
Implementing the Bimodal Usage

Distinguish between 2 types of blocks:

Write Block

Next write page

Overwrite Block

Next overwrite page

- : Reprogrammable page
- : Not reprogrammable page
- : Empty page

1st Mode

Overwrite bit in the mapping table:

<table>
<thead>
<tr>
<th>31</th>
<th>18</th>
<th>13</th>
<th>6</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>x</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

overwrite bit

bank

block

page
Seal Operation

1. Read the **inverse mapping** table from the last low page
2. Flip the **overwrite bits** in the main mapping table
3. Set the **next write page pointer** to use the high pages
Putting it all together..
GC strategies

Need new write block:
• Seal
• GC (1)

Victim selection: greedy

Need new overwrite block:
• GC (1)
• GC (2)

Victim selection: greedy
• Flash pages have a **spare region**. Traditionally ECC is stored there.
• Jasmine limitation: **fixed ECC location** in the spare region
• Spare region is FLASH as well.

Solution:
- **WOM codes** with correction capabilities
  [Gad et al., “Polar codes for noisy channels”, ISIT 2014]
- **Append ECCs** in the spare region
- Manage ECC in **software**

In the evaluation **8 page reprograms** are used
- not much gain beyond that [Kaiser et al., SYSTOR2013]
Evaluation

• What is the reliability of the extended P/E cycles?

• How many erase operations can we save?

• How much data is copied internally by GC?

• How is performance affected?
Reliability

BER test loop:
- Reprogram 8 times all Low Pages in random order
- Program 1 time all High Pages in page order
- Erase entire block
- WOM compatible data artificially generated (50% bit flipped)

Comparison with previous works:
MLC 35nm BER > 10^{-4} @ 10k P/E cycles
Evaluation (1): Benchmark

Micro Benchmark:

SSD user addressable space (8GB) Overprovisioning (1GB)

Dataset size (6GB)

Overwrite region Write region

Parameters:
- Overwrite region percentage (5% to 50%)
- Overwrite skewness (40% to 80%)
- Uniform distribution inside W/OW region
- 32KB write operations
- WOM compatible data generated artificially

Benchmark run:
1. **Warmup**: write all write region and all overwrite region
2. **Run**: write/overwrite twice the dataset (12GB) according to parameters
Evaluation (2): # Erase Operations

- Corresponds to the number of GC operations

85%

Overwrite skewness = 40%

Overwrite skewness = 60%

Overwrite skewness = 80%

Overwritable region percentage

Baseline
Seal FTL

![Graph showing overwrite skewness at 40%, 60%, and 80% overwritable region percentage with Baseline and Seal FTL]

- Overwrite skewness = 40%
- Overwrite skewness = 60%
- Overwrite skewness = 80%

Baseline
Seal FTL
Evaluation (4): Total data copied

Overwrite skewness = 40%

Overwrite skewness = 60%

Overwrite skewness = 80%

Overwritable region percentage

80%

Baseline
Seal FTL
Evaluation (5): Latency

![Graph showing latency measurements for different overwrite skewness and overwritable region percentages.](image)

- **Baseline**
- **Seal FTL**
Evaluation (6): Reuse Distance

Distances of overwrites on the same LBA (in GB) for 80% overwrite skewness

* histogram plotted with 1024 bins of size 8 MB each
Evaluation (7): Percentage of reprogram ops.

The graph shows the percentage of reprogram operations over the percentage of overwritable space for different skewness values. The skewness values are 40, 60, and 80, represented by different line styles and markers.

- **Red circles** with line: ow skewness = 40
- **Green triangles** with line: ow skewness = 60
- **Blue triangles** with line: ow skewness = 80

The x-axis represents the percentage of overwritable space, ranging from 5% to 50%, and the y-axis represents the percentage of reprogram operations, ranging from 0% to 100%.
Conclusion

• The extended P/E cycles can be implemented in MLC

• In contrast to SLC, MLCs pose more challenges

• This technique is very effective for hot data limited in size
Thanks for Your Attention

Questions?

All code can be found at
https://github.com/zdvresearch

We are hiring!
https://research.zdv.uni-mainz.de

Contact:
margagl@uni-mainz.de
brinkmann@uni-mainz.de
Related Work

- Previous flash studies confirm our findings
  - [Grupp. et al, MICRO 2009]

- WOM codes are actively developed
  - Polar WOM [Burshtein and Strugatski, IT 2013]

- WOM compatible data structures are under study
  - B-Tree [Kaiser et al, SYSTOR 2013]

- FTL designs that do not change the SSD interface
  - [Yadgar et al, FAST 2015]