Write Once Memory Codes and Lattices for Flash Memories

Brian M. Kurkoski
Japan Advanced Institute of Science and Technology

September 19, 2012
Seoul National University
### Hard Drives vs. Flash-Based Storage

<table>
<thead>
<tr>
<th></th>
<th>Hard Disk Drive</th>
<th>Flash Memory Drive (SSD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spinning disk—the only moving part inside your computer</td>
<td>Semiconductor memory — Mechanically durable</td>
<td></td>
</tr>
<tr>
<td>Fast sequential access</td>
<td>Very fast random reading and writing (~100 times faster than disk)</td>
<td></td>
</tr>
<tr>
<td>But, future increases limited by rotational speed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>High power</td>
<td>Low power</td>
<td></td>
</tr>
<tr>
<td>must buy a whole drive</td>
<td>small memories possible</td>
<td></td>
</tr>
<tr>
<td>• cannot buy drive for ¥550</td>
<td>• 2 GB for ¥550</td>
<td></td>
</tr>
<tr>
<td>Lowest cost per GB</td>
<td>High cost per GB</td>
<td></td>
</tr>
<tr>
<td>Long-term reliability: mechanical failure</td>
<td>Long-term reliability: semiconductor failure</td>
<td></td>
</tr>
</tbody>
</table>
SSD Price Drop and Capacity Increases

April 2010

November 2010

Not just SSDs:
- smartphones
- tablet computers
- cameras and video
- music players
- enterprise storage
- embedded systems
NAND Flash is arranged:

- one package has e.g. 8128 blocks (K9K8G08U0M)
- one block consists of 32-128 pages
- one page consists of 512-4096 bytes
- Data is stored in transistors, called “cells”
- cell stores 1-3 bits using charge/voltage

Flash transistor = cell
Charge is Easy to Add, Hard to Remove

Flash transistor = cell

In flash memories, charge is stored on a “floating gate”, and read as a voltage. Charge can easily be increased, but can only be decreased by an erasure operation. Only whole blocks of ~512 KB can be erased.
Flash Memory “Wears Out”

Like clothes you wear often, flash memories “wear out” as they are used.

- SLC flash (1 bit): 100,000 program/erase cycles
- TLC flash (3 bit): ~1,000 program/erase cycles
- MLC flash (2 bit): 10,000 program/erase cycles

Grupp, et al. 5 different MLC flash
Outline

WOM Codes are “Write Once Memory” codes.
- They allow re-writing a memory without erasing
- Promising application in extending the longevity of flash memories
- Theoretically interesting

Part 1: Binary WOM Codes [Tutorial]
- Simple examples
- Capacity
- Good code constructions

Part 2: non-binary WOM Codes
- Capacity
- Lattice-based code constructions [new results]
An Interesting Problem: Rewriting for Flash

Since 0→1 is easy, want a code which allows re-writing without 1→0 change

This problem is not new!
Motivated by CD-R, Rivest and Shamir proposed “WOM Codes” in 1982

- An “uncoded scheme” writes memory two times with rate 1/2
- Rivest and Shamir showed rate → 0.77

Rivest and Shamir, Inf. and Contr

1. Wolf et al, Bell Labs TR
2. Fiat and Shamir, IT Trans

Pre-2007 history of papers on “WOM Codes” and capacity
**Write-Once Memory**

In a “Write-Once Memory,”

- Memory begins in 0 state
- $0 \rightarrow 1$ is allowed
- $1 \rightarrow 0$ is not allowed (or expensive)

![Diagram of memory states](image)

**How to write two (or more) times?**

“Uncoded” approach has Rate = 1/2:

```
| 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
```

- first write
- second write

```
| 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
```

- first write
- second write
“How to Reuse a ‘Write Once’ Memory”
Rivest and Shamir 1982

Toy example rate 2/3 code:
- 3 storage “cells”
- Store 2 bits of information
- Can write data 2 times
- Example: store 10, then store 00

Initial State: [0 0 0]

Write 10: [ ]

Write 00: [ ]
“How to Reuse a ‘Write Once’ Memory”
Rivest and Shamir 1982

Toy example rate 2/3 code:
- 3 storage “cells”
- Store 2 bits of information
- Can write data 2 times
- Example: store 10, then store 00

Initial State: 0 0 0
Write 10: 1 0 0
Write 00: 0 0 0
“How to Reuse a ‘Write Once’ Memory”
Rivest and Shamir 1982

Toy example rate 2/3 code:

- 3 storage “cells”
- Store 2 bits of information
- Can write data 2 times
- Example: store 10, then store 00

Initial State: 0 0 0
Write 10: 1 0 0
Write 00: 1 1 1
Some Definitions for WOM Codes

- number of cells $n$, number of levels per cell $q$
  - $q = 2$ binary (this Part 1)
  - $q > 2$ non-binary (in Part 2)
- number of writes $t$, $t \geq 2$ is of interest
- number of messages on write $i$ is $M_i$ and $i = 1, 2, \ldots, t$; message = \{1,2,\ldots, M_i\}
- Rate on write $i$ is: $R_i = \frac{1}{n} \log_2 M_i$ bits/cell
- **Sum rate** $R$ is: $R = \sum_{i=1}^{t} R_i$
Sum Rate for Toy Example Code

\[ q = 2 \text{ binary} \]
\[ n = 3 \text{ cells} \]
\[ t = 2 \text{ writes} \]
\[ M_1 = 4, \quad R_1 = \frac{\log_2 4}{3} = \frac{2}{3} \]
\[ M_2 = 4, \quad R_2 = \frac{\log_2 4}{3} = \frac{2}{3} \]
\[ \text{sum rate} \quad R = R_1 + R_2 = \frac{4}{3} \]

Questions:

- Can we do better?
- What are the fundamental limits on the sum rate?
Achievable Rate Region [Heegard 1985]

For a binary WOM the t-write achievable rate region \( R^{(t)} \) is given by:

\[
R^{(t)} = \{ (R_1, \ldots, R_t) \mid R_1 \leq h(p_1), \\
R_2 \leq (1-p_1)h(p_2), \\
\vdots \\
R_{t-1} \leq (1 - p_1) \ldots (1 - p_{t-2})h(p_{t-1}) \\
R_t \leq (1 - p_1) \ldots (1 - p_{t-1}) \}
\]

where \( h(.) \) is the binary entropy function.

- Constructed a “partition code”
- Also gave a recursive formula to calculate the capacity \( C^{(t)} \), i.e. maximum achievable sum-rate
This talk concentrates on $t = 2$

- Simplest example
- Highest rate — Most practical

Heegard’s result for $t = 2$ writes:

$$R^{(2)} = \left\{ (R_1, R_2) \mid R_1 \leq h(p), \hspace{1cm} R_2 \leq (1 - p) \right\}$$

for $0 \leq p \leq 0.5$
Two Important Cases:
Capacity and Equal-Rate Capacity

**Capacity** ("unrestricted rate capacity")
- The capacity $C(t)$ is the maximum of the achievable sum rates for $t$ writes
- Heegard also showed:
  $$C(t) = \log_2(t+1)$$
- The rates are **not equal in general**, $R_1 \neq R_2 \neq ...$

**Equal-Rate Capacity** ("fixed rate capacity")
- Equal rates $R_1 = R_2 = ...$ are of practical importance
- Heegard gave a recursive method to compute capacity $C_0(t)$ of a $t$-write binary WOM code
Binary WOM codes for t=2 writes

\[ R^{(2)} = \left\{ (R_1, R_2) \mid R_1 \leq h(p), \quad R_2 \leq (1 - p) \right\} \]

The (unrestricted-rate) capacity \( C^{(2)} \) of 2-write binary WOM is:

\[
C^{(2)} = \max_{R_1, R_2 \in R^{(2)}} (R_1 + R_2)
\]

\[
C^{(2)} = \max_{p \in [0, \frac{1}{2}]} (h(p) + (1 - p))
\]

The sum is maximized with \( p^* = \frac{1}{3} \), so:

\[
R_1 = 0.918
\]

\[
R_2 = \frac{2}{3}
\]

\[
C^{(2)} = \log_2 3 \approx 1.58
\]
The equal-rate (fixed rate) capacity is:

$$C_0^{(2)} = h(p^*) + (1 - p^*)$$

and the solution to

$$h(p^*) = (1 - p^*)$$

is $p^* = 0.227$, which gives:

$$R_1 = R_2 \approx 0.773$$

$$C_0^{(2)} = 1.546$$
Some code constructions

Coset-based constructions, based on linear error-correcting codes:


There are many many other code constructions in the literature

- “heuristical”
- Coset-type are efficient constructions (some with $t = 2$)
An Early Coset-Based WOM Code


Use coset coding to encode information. Pick a linear code. Encoding:

1. Information is encoded as the syndrome of a sequence
2. From the coset of that syndrome, select the coset codeword with the minimum weight.
3. Write that coset codeword to memory.

Decoding:

1. Compute the syndrome of the recorded sequence.

Example: Use Hamming (7,4) code to encode information with $T = 3$ writes:

$$(0, 0, 0) \rightarrow (1, 0, 1) \rightarrow (1, 1, 0) \rightarrow (0, 0, 0)$$

$\text{①} \text{②} \text{③}$$
Hamming Codewords

<table>
<thead>
<tr>
<th>000000</th>
<th>000001</th>
<th>0000010</th>
<th>0000100</th>
<th>0001000</th>
<th>001000</th>
<th>0100000</th>
</tr>
</thead>
<tbody>
<tr>
<td>001111</td>
<td></td>
<td></td>
<td></td>
<td>000111</td>
<td>010111</td>
<td>1001111</td>
</tr>
<tr>
<td>001001</td>
<td></td>
<td></td>
<td></td>
<td>001101</td>
<td>011001</td>
<td>1010011</td>
</tr>
<tr>
<td>001100</td>
<td>$\mathbf{1}$</td>
<td>$\mathbf{2}$</td>
<td>$\mathbf{3}$</td>
<td></td>
<td>011100</td>
<td>1011100</td>
</tr>
<tr>
<td>010010</td>
<td></td>
<td></td>
<td></td>
<td>010001</td>
<td>001001</td>
<td>1101010</td>
</tr>
<tr>
<td>010100</td>
<td></td>
<td></td>
<td></td>
<td>010100</td>
<td>011110</td>
<td>1110110</td>
</tr>
<tr>
<td>011010</td>
<td></td>
<td></td>
<td></td>
<td>011001</td>
<td>001101</td>
<td>1111011</td>
</tr>
<tr>
<td>011101</td>
<td></td>
<td></td>
<td></td>
<td>011101</td>
<td>001101</td>
<td>1111011</td>
</tr>
<tr>
<td>100100</td>
<td></td>
<td></td>
<td></td>
<td>100100</td>
<td>110100</td>
<td>0010011</td>
</tr>
<tr>
<td>100101</td>
<td></td>
<td></td>
<td></td>
<td>100101</td>
<td>111011</td>
<td>0010101</td>
</tr>
<tr>
<td>101010</td>
<td></td>
<td></td>
<td></td>
<td>101010</td>
<td>111010</td>
<td>0011010</td>
</tr>
<tr>
<td>101100</td>
<td></td>
<td></td>
<td></td>
<td>101100</td>
<td>111101</td>
<td>0011010</td>
</tr>
<tr>
<td>110001</td>
<td></td>
<td></td>
<td></td>
<td>110001</td>
<td>100001</td>
<td>1101001</td>
</tr>
<tr>
<td>110110</td>
<td></td>
<td></td>
<td></td>
<td>110110</td>
<td>100100</td>
<td>0101100</td>
</tr>
<tr>
<td>111000</td>
<td></td>
<td></td>
<td></td>
<td>111000</td>
<td>101000</td>
<td>0110001</td>
</tr>
<tr>
<td>111111</td>
<td>$\mathbf{3}$</td>
<td></td>
<td></td>
<td>111011</td>
<td>101111</td>
<td>0111111</td>
</tr>
</tbody>
</table>

Syndromes = Information

\[
\begin{align*}
(0, 0, 0) & \rightarrow (1, 0, 1) \rightarrow (1, 1, 0) \rightarrow (0, 0, 0) \\
\end{align*}
\]
YKSVW Two-Write WOM Codes

E. Yaakobi, S. Kayser, P.H. Siegel, A. Vardy, J.K. Wolf “Efficient two-write WOM-codes,”

1. Use a linear code with \((n - k)\) by \(n\) parity check matrix \(H\).
2. Form the first codebook: \(v\) is a codeword if \(H\) remains full rank after columns of \(H\) corresponding to \(v_i = 1\) are set to zero.
3. For the second codebook (write \(s\) of \(n - k\) bits): \(v_1\) is current state of memory. Solve \(H' v_2 = H v_2 + s\). Write \(v_1 + v_2\) to memory.

This construction guarantees 2 writes.
YKSVW Two-Write WOM Codes

\[ \triangledown \text{construction based on well-known codes} \]

\[ \text{construction based on computer-based searches with } n = 33 \]
Part 1 Summary: Binary WOM Codes

Motivation are codes that can improve the longevity of flash memories

Binary WOM codes
- Allow increasing 0->1, but 1->0 is not allowed
- Can write t times

Review of results
- Concentrated on $t = 2$ writes — simplest and most practical case
- Capacity and rate regions are known, but rates are not equal
- More practical: equal rate codes. “Equal-Rate capacity” can be found
- Some very promising code constructions

Comments on WOM encoding/decoding
- Encoding tends to be complicated (like source coding)
- Decoding tends to be simpler
- Small $n$ codes tend to have good performance