Constrained Flash Memory Programming

Amit Berman and Yitzhak Birk

{bermanam@tx, birk@ee}.technion.ac.il
Technion – Israel Institute of Technology

International Symposium on Information Theory, August, 2011
Contributions

• Characterization and formulation of Inter-Cell Interference in Flash Memory

• Proposing constrained coding for ICI mitigation

• Examples: Capacity and codes for “Breadth-First” and “Even-Odd” ICI channels
Flash Memory

• Non-Volatile Solid-State Memory Device

• Widely used in consumer and industrial electronics
Floating-Gate (FG) Cell

- Conductive FG layer surrounded by dielectrics
- Write: Inject charge from substrate to FG layer
- Erase: Release charge from FG layer to substrate
$V_t$ Distributions

- Stored data = amount of charge in FG
  - The amount of charge determines $V_t$, the minimum gate voltage that causes the transistor to conduct
- Observed $V_t$ is inaccurate
  - Cell programming with limited resolution + ....
  - Interference from other cells (~40% and growing!)
- Result: must allocate a $V_t$ range to each stored value
Inter-Cell Coupling

• FG-FG inter-cell coupling causes the charge in one cell to affect a neighboring cell’s threshold voltage.
**$V_t$ Distribution Widening**

- When considering each cell in isolation, the observed phenomenon is a “widening” of the threshold voltage distributions.

![Graph showing the effect of widening in $V_t$ distributions](image)

Width multiplied by up to 4x
ICI experiment (program pulse) [Lee et-al.]

• Neglecting $C_{FGXY}$ and assuming $Q_{FG}=0$, the floating gate voltage due to ICC is:

$$V_{FG} = \frac{C_{ONO}V_{CG} + C_{FGX}(V_1 + V_2) + C_{FGY}(V_3 + V_4) + V_{FGCG}(V_5 + V_6)}{C_{TUN} + C_{ONO} + 2C_{FGX} + 2C_{FGY} + 2C_{FGCG}}$$

$C_{TUN}$, $C_{ONO}$ are the FG dialectics cap., $V_5$, $V_6$ are gate voltages in neighbor vertical cells (do not appear in the Figure)
Insights
ICI with Program & Verify

• Program & Verify:
  – Charge is added to a cell in small increments
  – $V_t$ is checked after each addition
  – Programming ceases upon reaching the desired $V_t$

• Therefore, $V_t$ of any given cell is affected only by charge added to its neighbors after its own charging has been completed.

The effect of inter-cell interference depends on the coupling, data and the programming scheme
Existing Interference-Mitigation Schemes

• Proportional programming [Trinh et-al., USP 6,996,004]
  ▪ Concurrent, “proportional” programming of same-row cells → near-simultaneous completion.
  ▪ Pros: insensitive to coupling parameters, simple read.
  ▪ Cons: complicated, possibly slow programming, can’t account for next line.

• Intelligent read decoding [Li et-al., USP 7,301,839]
  ▪ Based on programming order, decode w. successive interference cancellation.
  ▪ Pros: simple programming.
  ▪ Cons: Must know coupling parameters, no variation allowed, requires fine-resolution read → complex and slow
Our Approach: Constrained Coding

• Forbid certain adjacent-cell level combinations:
  – Criterion depends on programming order
  – Threshold is a design trade-off
• Programming: use only permissible combinations (legal code words)
• Decoding: use inverse mapping
Constrained Coding – Main Features

• Pros:
  – Limits the effect of inter-cell coupling → narrow distributions → many levels or higher reliability
  – Fairly simply encoding and decoding
  – Only need to know an upper bound on coupling coefficients

• Cons:
  – Code rate <1 → some loss of capacity relative to ideal with narrow distributions.
Constrained Coding - Remarks

- Can be combined with ECC

- Complementary to the previous schemes and can be combined with them:
  - Semi-accurate programming + minimal restrictions
  - Some restrictions with simpler intelligent read decoding
Flash Constrained Coding Scheme Synopsis

1) Assign ICI severity function $D(c)$ to data sequence, depends on programming order
   • $D: \{\text{data sequence } c\} \rightarrow \text{severity value}$
   • More interference $\rightarrow$ larger value

2) Choose $T \rightarrow$ require $D(c)<T \rightarrow$ determine code rate

3) Construct Encoder and Decoder
1) Severity Function
Example: 1-D, “Breadth 1st” Coding

- 1-D: a single row of cells is considered
- Programming (charge & verify)
  - All >0 cells programmed to level 1
  - All >1 cells programmed to level 2
  - …
- Sequence eligibility criterion:

\[
D(C) = \max \{ N_L - C, 0 \} + \max \{ N_R - C, 0 \} < T
\]

- T represents a trade-off:
  - Large T: efficient coding, but wider distributions and fewer levels
  - Small T: opposite pros and cons

\[ N_L, C, N_R: \text{respective target levels} \]
Breadth 1$^{st}$ programming order - demonstration

- $\{0,...,3\}$ levels per cell
Breadth 1\textsuperscript{st}” programming order - demonstration
Breadth 1\textsuperscript{st}” programming order - demonstration

Target values: \begin{array}{ccc} 3 & 2 & 1 \end{array}

```
\begin{array}{c|c|c}
3 & 3 & 3 \\
2 & 2 & 2 \\
1 & 1 & 1 \\
\end{array}
```
Breadth 1st” programming order - demonstration

Target values: 3 2 1

Program pulse
- Applied to all
- Charge injection
- Inter-Cell Interference
Breadth 1\textsuperscript{st}” programming order - demonstration

Target values: \hspace{1cm} 3 \hspace{1cm} 2 \hspace{1cm} 1

Verify

- Non reached desired value
Breadth 1st” programming order - demonstration

Target values: 3 2 1

Program pulse
• Applied to all
• - Charge injection
• - Inter-Cell Interference
• - Accounted-for ICI
Breadth 1st” programming order - demonstration

Target values: 3 2 1

Verify

- Right cell reached target
  (no more program pulses for it)
Program pulse

- Applied left and center
- Charge injection
- Inter-Cell Interference
- Accounted-for ICI
- Post-programming ICI

Target values: 3 2 1
Breadth 1\textsuperscript{st}” programming order - demonstration

Target values: 3 2 1

Verify
- Centered cell reached target
Breadth 1st" programming order - demonstration

Target values: 3 2 1

Program pulse
• Applied left and center
• - Charge injection
• - Inter-Cell Interference
• - Accounted-for ICI
• - ICI after verified
Breadth 1st” programming order - demonstration

Target values: 3 2 1

Verify
• Non verified
Breadth 1st” programming order - demonstration

Target values: 3 2 1

Program pulse
- Applied to left cell
- Charge injection
- Inter-Cell Interference
- Accounted-for ICI
- ICI after verified
Breadth 1st” programming order - demonstration

Target values: 3 2 1

Verify
- Left cell verified

Program Ends

Breadth-first programming order - demonstration

Verify
• Left cell verified

Program Ends
Breadth 1st” programming order - demonstration

Target values: 3 2 1

Centered cell is erroneous

\[ D(C) = \max \{ N_L - C, 0 \} + \max \{ N_R - C, 0 \} < T \]
2) Set $T$ and determine code rate
Example: T=2, L={0,1} two levels per cell

- D(c)<2: forbid the combination of 101
- Corresponding language graph and adjunct matrix:

\[ A_G = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \end{pmatrix} \]

- Capacity: \( Cap(S) = \lim_{l \to \infty} \frac{\log_2 N(l;S)}{l} = \log_2 \lambda(A_G) = 0.8115 \)

- Determine code rate to be \( 4/5 = 0.8 \)
  - Encoder: uncoded 4 bits input, encoded 5 bits output
  - Decoder: vice versa.
3) Encoder and Decoder Construction
Example: $T=2$, one program level per cell

- For simplicity, the numbers mark groups of 5-bit sequences

\[
A_g^5 = \begin{pmatrix} 7 & 9 & 5 \\ 5 & 7 & 4 \\ 4 & 5 & 3 \end{pmatrix}
\]
Example: T=2, one program level per cell

- State-splitting algorithm

```
131
00010
00110

132
01110
10010
11110
```
Example: T=2, one program level per cell

- State-splitting algorithm

```
  211  212
  00000 10000
        11000
        11100
        00100
```
Example: $T=2$, one program level per cell

- State-splitting algorithm
Example: T=2, one program level per cell

- State-splitting algorithm
• Final
• Nodes are marked with letters
- **xy** - edges going from state x to state y
- **input/output** - allocation of inputs and outputs

<table>
<thead>
<tr>
<th>State</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>aa</td>
<td>0000/00100</td>
<td>0111/00100</td>
</tr>
<tr>
<td>ab</td>
<td>0000/00100</td>
<td>1000/01000</td>
</tr>
<tr>
<td>ah</td>
<td>0000/00100</td>
<td>1001/01100</td>
</tr>
<tr>
<td>bd</td>
<td>0000/00100</td>
<td>1010/00000</td>
</tr>
<tr>
<td>bc</td>
<td>0000/00100</td>
<td>1011/01000</td>
</tr>
<tr>
<td>bi</td>
<td>0000/00100</td>
<td>1100/11000</td>
</tr>
<tr>
<td>ce</td>
<td>0000/00011</td>
<td>1000/00011</td>
</tr>
<tr>
<td>cf</td>
<td>0000/00011</td>
<td>1100/11001</td>
</tr>
<tr>
<td>dg</td>
<td>0000/00011</td>
<td>1010/00110</td>
</tr>
<tr>
<td>dh</td>
<td>0000/00011</td>
<td>1011/11110</td>
</tr>
<tr>
<td>di</td>
<td>0000/00011</td>
<td>1010/10010</td>
</tr>
<tr>
<td>df</td>
<td>0000/00011</td>
<td>1011/11110</td>
</tr>
<tr>
<td>ee</td>
<td>0000/00111</td>
<td>0111/00111</td>
</tr>
<tr>
<td>ef</td>
<td>0000/00111</td>
<td>1001/00001</td>
</tr>
<tr>
<td>ec</td>
<td>0000/00111</td>
<td>1001/11111</td>
</tr>
<tr>
<td>ed</td>
<td>0000/00111</td>
<td>1010/10011</td>
</tr>
<tr>
<td>fa</td>
<td>0000/00000</td>
<td>0101/00100</td>
</tr>
<tr>
<td>fb</td>
<td>0000/00000</td>
<td>0110/01000</td>
</tr>
<tr>
<td>fg</td>
<td>0000/00000</td>
<td>0111/01100</td>
</tr>
<tr>
<td>gg</td>
<td>0000/00000</td>
<td>1000/01000</td>
</tr>
<tr>
<td>gg</td>
<td>0000/00000</td>
<td>1001/01100</td>
</tr>
<tr>
<td>gg</td>
<td>0000/00000</td>
<td>1010/10000</td>
</tr>
<tr>
<td>gg</td>
<td>0000/00000</td>
<td>1011/11000</td>
</tr>
<tr>
<td>gg</td>
<td>0000/00000</td>
<td>1100/11000</td>
</tr>
<tr>
<td>gg</td>
<td>0000/00000</td>
<td>1101/10000</td>
</tr>
<tr>
<td>gg</td>
<td>0000/00000</td>
<td>1110/10011</td>
</tr>
<tr>
<td>gg</td>
<td>0000/00000</td>
<td>1111/00100</td>
</tr>
<tr>
<td>gc</td>
<td>gd</td>
<td>gh</td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
<td>---------</td>
</tr>
<tr>
<td>0000/1000</td>
<td>0100/1000</td>
<td>1000/11110</td>
</tr>
<tr>
<td>0001/1100</td>
<td>0101/1100</td>
<td>1001/00010</td>
</tr>
<tr>
<td>0010/1110</td>
<td>0110/1110</td>
<td>1010/00110</td>
</tr>
<tr>
<td>0011/00100</td>
<td>0111/00100</td>
<td>1011/10010</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ha</th>
<th>hb</th>
<th>hc</th>
<th>hd</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000/0100</td>
<td>0100/0100</td>
<td>1000/01000</td>
<td>1100/01000</td>
</tr>
<tr>
<td>0001/00000</td>
<td>0101/00000</td>
<td>1001/00000</td>
<td>1101/00000</td>
</tr>
<tr>
<td>0010/01100</td>
<td>0110/01100</td>
<td>1010/01100</td>
<td>1110/01100</td>
</tr>
<tr>
<td>0011/00100</td>
<td>0111/00100</td>
<td>1011/00100</td>
<td>1111/00100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ie</th>
<th>if</th>
<th>ig</th>
<th>ii</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000/00001</td>
<td>0101/00001</td>
<td>1010/00001</td>
<td>1111/00010</td>
</tr>
<tr>
<td>0001/00011</td>
<td>0110/00011</td>
<td>1011/00011</td>
<td>1111/00011</td>
</tr>
<tr>
<td>0010/00111</td>
<td>0111/00111</td>
<td>1100/00111</td>
<td>1110/00111</td>
</tr>
<tr>
<td>0011/01111</td>
<td>1000/01111</td>
<td>1101/01111</td>
<td>1110/01111</td>
</tr>
<tr>
<td>0100/01001</td>
<td>1001/01001</td>
<td>1110/01001</td>
<td>1111/01001</td>
</tr>
</tbody>
</table>
Implementation with look-up table

- Rate penalty, e.g. $R = \frac{5}{8} = 0.625$
- Error propagation-free

$A_G^8 = \begin{pmatrix} 37 & 49 & 28 \\ 28 & 37 & 21 \\ 21 & 28 & 16 \end{pmatrix}$

<table>
<thead>
<tr>
<th>#</th>
<th>Data</th>
<th>Enc</th>
<th>#</th>
<th>Data</th>
<th>Enc</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00000</td>
<td>00000000</td>
<td>16</td>
<td>10000</td>
<td>00010000</td>
</tr>
<tr>
<td>1</td>
<td>00001</td>
<td>10000000</td>
<td>17</td>
<td>10001</td>
<td>00011000</td>
</tr>
<tr>
<td>2</td>
<td>00010</td>
<td>11000000</td>
<td>18</td>
<td>10010</td>
<td>00011100</td>
</tr>
<tr>
<td>3</td>
<td>00011</td>
<td>11100000</td>
<td>19</td>
<td>10011</td>
<td>00001000</td>
</tr>
<tr>
<td>4</td>
<td>00100</td>
<td>11110000</td>
<td>20</td>
<td>10100</td>
<td>00001100</td>
</tr>
<tr>
<td>5</td>
<td>00101</td>
<td>11111000</td>
<td>21</td>
<td>10101</td>
<td>00000100</td>
</tr>
<tr>
<td>6</td>
<td>00110</td>
<td>11111100</td>
<td>22</td>
<td>10110</td>
<td>10010000</td>
</tr>
<tr>
<td>7</td>
<td>00111</td>
<td>01000000</td>
<td>23</td>
<td>10111</td>
<td>10011000</td>
</tr>
<tr>
<td>8</td>
<td>01000</td>
<td>01100000</td>
<td>24</td>
<td>11000</td>
<td>10011100</td>
</tr>
<tr>
<td>9</td>
<td>01001</td>
<td>01110000</td>
<td>25</td>
<td>11001</td>
<td>10001000</td>
</tr>
<tr>
<td>10</td>
<td>01010</td>
<td>01111000</td>
<td>26</td>
<td>11010</td>
<td>10001100</td>
</tr>
<tr>
<td>11</td>
<td>01011</td>
<td>01111100</td>
<td>27</td>
<td>11011</td>
<td>10000100</td>
</tr>
<tr>
<td>12</td>
<td>01100</td>
<td>00100000</td>
<td>28</td>
<td>11100</td>
<td>01001000</td>
</tr>
<tr>
<td>13</td>
<td>01101</td>
<td>00110000</td>
<td>29</td>
<td>11101</td>
<td>01001100</td>
</tr>
<tr>
<td>14</td>
<td>01110</td>
<td>00111000</td>
<td>30</td>
<td>11110</td>
<td>00100000</td>
</tr>
<tr>
<td>15</td>
<td>01111</td>
<td>00111100</td>
<td>31</td>
<td>11111</td>
<td>00100100</td>
</tr>
</tbody>
</table>
General Language Graphs
Breadth 1\textsuperscript{st} program order

- Given: constraint \( T, \{0,\ldots,L\} \) levels per cell
- Language graph is:
Breadth 1\textsuperscript{st} program order

- Additional edges if labels are unique to nodes
Even-Odd program order

• 1-D: a single row of cells is considered
• Even cells are programmed first, Odd cell are second
• Sequence eligibility criterion:

\[
D(c) = \begin{cases} 
0, & c \text{ odd} \\
Level(c-1) + Level(c+1), & c \text{ even}
\end{cases}
\]
Even-Odd program order

- Given: constraint $T$, $\{0,...,L\}$ levels per cell
- Language graph is:
Note

• Graphs are Shannon covers.
• Proof – second graph in the paper, first in future paper.
Capacity Results
Language Capacity

• Normalized, calculated as:

\[
 Cap_{\text{Norm}}(S) = \lim_{{l \to \infty}} \frac{\log_2 N(l; S)}{\log_2 (L + 1)} = \frac{\log_2 \lambda (A_G)}{\log_2 (L + 1)}
\]
Capacity Implication Example

• T=5, breadth-1st programming, L=4, code rate=0.95.
• Assumption: constrained coding permitted an increase in the number of levels from 4 to 5:
  • Baseline: \( 1.0 \cdot \log_2(4) = 2 \)
  • Constrained coding: \( 0.95 \cdot \log_2(5) = 2.2 > 2 \)
• A 10% increase in capacity
Conclusions

• Constrained coding can be used to chop off the tail of $V_t$ distributions with only a minor reduction in rate

• Can be used beneficially to increase capacity or to increase reliability

• Can replace proportional programming and intelligent decoding or complement them
Contributions

• Characterization and formulation of Inter-Cell Interference in Flash Memory

• Proposing constrained coding for ICI mitigation

• Examples: Capacity and codes for “Breadth-First” and “Even-Odd” ICI channels