Error Recovery Flows in NAND Flash SSDs

Viet-Dzung Nguyen
Marvell Semiconductor, Inc.
Outline

- Data Reliability in NAND Flash Memories
- Concept of an Error Recovery Flow (ERF)
- Components of an ERF
- Optimization of an ERF in an LDPC-Based Controller
- Comparison with a BCH-Based ERF
Data Reliability in NAND Flash

- NAND flash stores information in different charge levels of NAND flash cells
- In a fresh NAND device, the distributions of threshold voltages around their mean values are very sharp
  - Hence, the number of bits detected erroneously is very small
- In a NAND device stressed by program/erase cycle, retention or read disturb, the distributions become wider
  - Hence, the number of bits detected erroneously is larger
Error correction codes are employed to correct errors introduced by NAND flash.

- Decoder classification:
  - Hard decoders use the hard decision generated by one read from NAND.
  - Soft decoders use soft information generated by many reads from NAND.
Issue a read to NAND with a $V_{th}$ setting, decode with a hard decoder

Is decoding successful?

Data sent to host

Set $i=0$

Run error recovery procedure $i$

Is decoding successful?

Exit ERF with success

Set $i:=i+1$

Any more ERPs left?

Exit ERF with failure

On-the-fly Decode

Error Recovery Flow
Components of an ERF

Hard Read Retry

- Retrying hard input decoder by performing a read with new $V_{th}$
- Useful when sub-optimal $V_{th}$ were used for original read. The performance is not sensitive to quality of soft information

Soft Read Retry

- Running soft input decoder using soft information collected from multiple reads
- Useful to recover pages with many error bits. Has higher average latency but very good performance

$V_{th}$ Calibration

- Inferring optimal $V_{th}$ using histogram of charge levels
- Useful when sub-optimal $V_{th}$ settings are reason for failure and a good setting could not be found easily

LLR Calibration

- Adjusting LLRs assignment based on collected charge level histogram
- Useful when soft information is collected with sub-optimal $V_{th}$ and LLRs were assigned assuming that the $V_{th}$ were optimal
Components of an ERF (cont.)

- **Inter-Cell Interference Cancellation**
  - Cancelling the interference caused by adjacent cells by assigning LLRs based on states of adjacent cells
  - Useful to recover pages with very bad quality

- **Hard Error Mitigation**
  - Adjusting LLRs taking hard errors into account, detecting bad cells and bad bit-lines
  - Useful to recover pages with disproportionate number of hard errors

- **RAID**
  - Recovering one failed component in a RAID stripe by XOR-ing the remaining successful components in the stripe
  - Useful to recover from die failures or specific failure mechanisms of 3D NANDs
ERF Optimization Goal

- Recover data with highest probability of success and lowest latency:
  - Some system has specific requirements on host time out
  - A good ERF must produce a good shape of the latency distribution
  - A good ERF must not exhaust hardware and NAND resources that are needed to sustain other traffic
## ERF Classification

<table>
<thead>
<tr>
<th>Description</th>
<th>Simple/Typical/Static</th>
<th>Complex/Customized/Adaptive</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1. Hard decode with $V_{th}$ settings from the NAND vendor’s read retry table</td>
<td>1. Start with a hard decode that uses a customized $V_{th}$ setting (derived by the firmware’s media management algorithm)</td>
</tr>
<tr>
<td></td>
<td>2. Soft decode with a few hard-coded $V_{th}$ settings</td>
<td>2. The subsequent steps can be either a hard decode step or soft decode step. Each read in the subsequent steps use a $V_{th}$ setting that is adaptively derived.</td>
</tr>
<tr>
<td></td>
<td>3. RAID</td>
<td>3. Early detection of defective NANDs and trigger RAID as soon as possible if RAID is deemed necessary</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$V_{th}$ Settings</th>
<th>Pre-determined</th>
<th>Derived on-the-fly</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Reliability</th>
<th>Typically meets low-end requirements if stay within or below NAND vendor’s NAND stressing boundary</th>
<th>Meet high-end requirements even if NAND is stressed beyond NAND vendor’s stressing boundary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latency</td>
<td>Acceptable average latency, very bad worst case latency</td>
<td>Optimal latency distribution</td>
</tr>
<tr>
<td>Optimization complexity</td>
<td>Simple, requires little NAND characterization</td>
<td>Complex, requires extensive NAND characterization</td>
</tr>
</tbody>
</table>
ERF Optimization Strategy

- NAND characterization
- ERF Optimization

NAND error bit distributions at various NAND stressing condition and \(V_{th}\) settings

Error Recovery Flow

NAND error bit distributions at various NAND stressing condition and \(V_{th}\) settings

DPS Techniques
Statistical Analysis Tools

HARDWARE SIMULATOR
(MARVELL INTERNAL, ALSO AVAILABLE FOR SELECTED CUSTOMERS)

Refine

Performance estimations:
- Retry rate at different drive conditions
- Throughput at different drive conditions
Comparison with BCH-Based ERF

- In BCH-based controllers, the aim of ERF is to find a $V_{th}$ setting in which the number of raw bit errors is less than the correction capability of the BCH code.

- Hence, ERF in BCH controllers primarily involves read retries in which the failed page is retried by reading with a different $V_{th}$ setting for next read retry.
  - If a particular read retry results in a failure, it does not provide any feedback which can help in choosing the $V_{th}$ for next read retry.
  - This means that the long tail in the distribution of retry latency cannot be cut short.

- Don’t use an LDPC-based ERF that looks like a BCH-based ERF 😊

- Advanced tools exists (at least in Marvell’s controllers): We collaborate with customers on optimizing ERFs as well!
Questions?