Understanding Error Correction Mandates for Flash Memory

Charles Camp

IBM Flash Systems Development

Special thanks to Tom Parnell (IBM Research - Zurich); Gary Tressler and Tom Griffin (IBM Technology Development)

Agenda

- Why are We Here?
- Error Correction Fundamentals
  - A Simple Channel/Storage Model
  - Extension to the Real World
  - RBER, UBER, and the Magic of Correction
  - Tradeoffs and Numerical Examples
- The Quest for Deeper Knowledge
  - Characterization
  - Analysis
- Questions?... Comments?

Why are We Here?

- Why are We Here?
- Error Correction Fundamentals
  - A Simple Channel/Storage Model
  - Extension to the Real World
  - RBER, UBER, and the Magic of Correction
  - Tradeoffs and Numerical Examples
- The Quest for Deeper Knowledge
  - Characterization
  - Analysis
- Questions?... Comments?

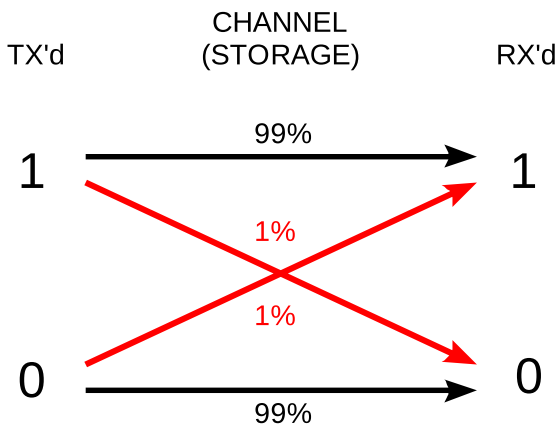
Why are We Here?

- Flash memory is a lossy storage medium.
- Device manufacturers issue error correction mandates that must be met in order to guarantee data sheet specifications, e.g. write endurance.
- In some cases, a manufacturer will recommend a particular error correction scheme or algorithm.
- What if we can live with relaxed specifications? Can we get away with less error correction?
- What if we need performance beyond the data sheet specifications? Can we improve performance with increased error correction?
- How do we know how well our codes perform?

Error Correction Fundamentals

- Why are We Here?
- Error Correction Fundamentals
  - A Simple Channel/Storage Model
  - Extension to the Real World
  - RBER, UBER, and the Magic of Correction
  - Tradeoffs and Numerical Examples
- The Quest for Deeper Knowledge
  - Characterization
  - Analysis
- Questions?... Comments?

A Simple Channel/Storage Model



- Example : binary symmetric channel with equal error probability for transmission (storage) of either 0 or 1 .
- While highly simplistic, the BSC serves as a reasonable first-order approximation of Flash.
- In this example $P_e = 0.01$, Pr(success) = $1 – P_e = 0.99$.
- The probability of error for any single bit transmitted across the channel is the raw bit error rate, or RBER. In this example, RBER = 0.01 .

A Simple Channel/Storage Model

- Given n transmitted (or stored) bits, instead of simply one bit, what is the probability of having exactly k errors within those n bits?

$$\Pr(k) = \binom{n}{k} \times P_e^{\,k} \times (1 - P_e)^{n-k} = \frac{n!}{k!(n-k)!} P_e^{\,k} \times (1 - P_e)^{n-k}$$

A Simple Channel/Storage Model

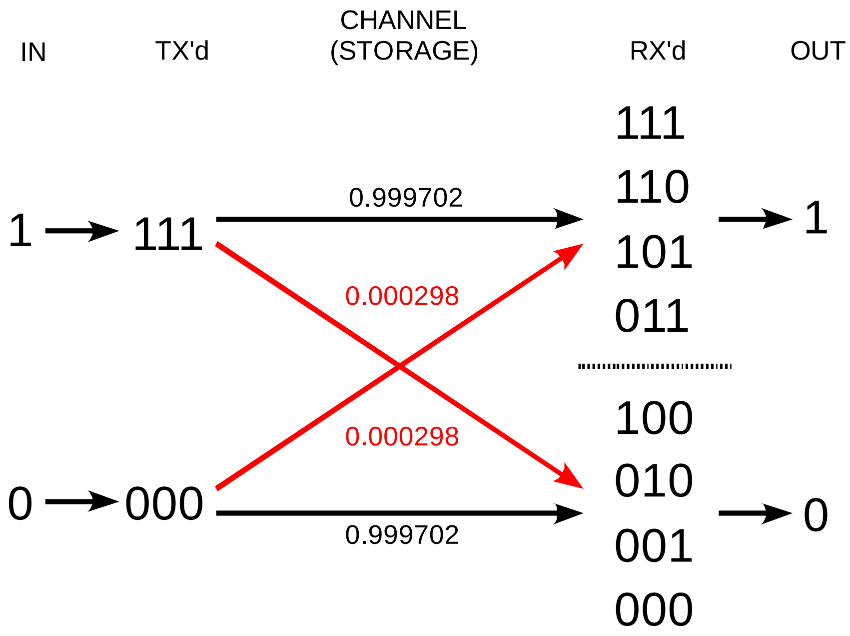
- Consider three stored bits (n = 3), using an RBER of 0.01 from the previous example…

$$\Pr(\text{exactly 0 errors}) = (3! / (0! \times 3!)) \times .01^0 \times 0.99^3 = 0.970299$$
$$\Pr(\text{exactly 1 error}) = (3! / (1! \times 2!)) \times .01^1 \times 0.99^2 = 0.029403$$
$$\Pr(\text{exactly 2 errors}) = (3! / (2! \times 1!)) \times .01^2 \times 0.99^1 = 0.000297$$
$$\Pr(\text{exactly 3 errors}) = (3! / (3! \times 0!)) \times .01^3 \times 0.99^0 = 0.000001$$

- Probability of having x or less errors is the sum of the individual probabilities for $k \leq x$…

$$\Pr(\text{1 or less errors}) = 0.029403 + 0.970299 + = 0.999702$$

A Simple Channel/Storage Model



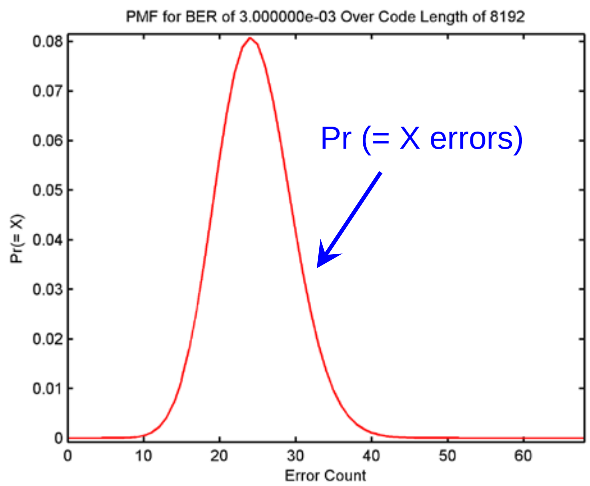
- Example : BSC with simple 3x majority logic encoding. Single data bits are sent as 3-bit code words. A single-bit error within any code word is guaranteed to be “fixed”.
- Raw bit error rate through the channel (RBER) remains 0.01. Code rate = 0.333 (impractical for most storage applications). Post-decoding error rate, however, drops to 0.000298 – an improvement of more than 33x!

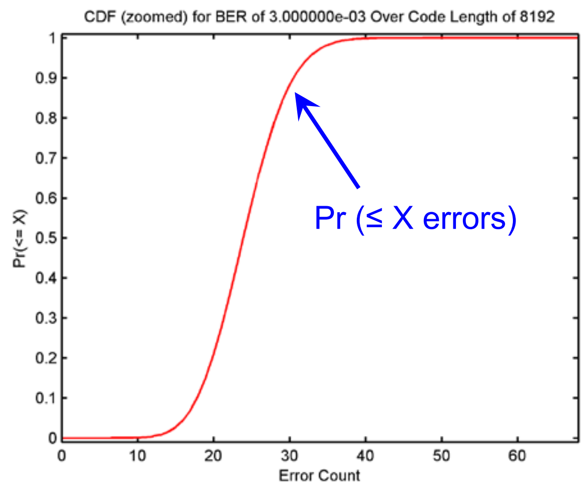
Extension to the Real World

- Previous example is interesting, but not practical. Very short code words are inefficient, majority logic particularly so.
- Recent error correction schemes for Flash memory have relied heavily upon BCH codes.
- BCH codes are algebraic codes. Algebraic codes provide deterministic performance - they *guarantee* that a particular number of errors within a single code word can always be corrected.
- Flash device manufacturers typically mandate that users correct X errors within Y bits. BCH codes are a good fit for this task, since they can be designed to meet the manufacturer's requirement *deterministically* – no guessing!
- To really understand the performance of these codes, however, we first need to extend the mathematics we just covered.
- Nothing we need, however, is outside the scope of a good freshman-level or sophomore-level course in probability.

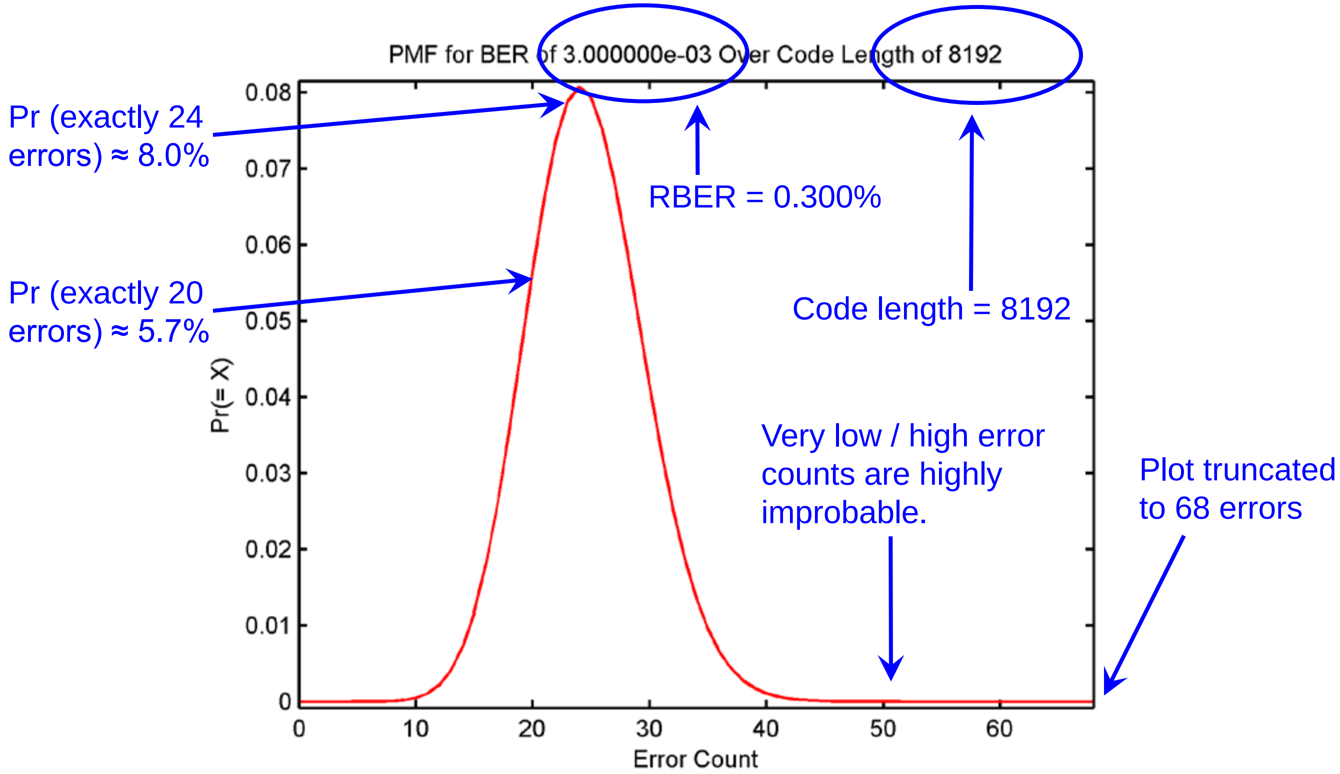
Extension to the Real World

- Given a channel (or storage medium) of the type we discussed earlier, and an RBER for the channel, the error count within a group of n bits is a random variable.
- The distribution of error counts can be seen in the random variable's probability mass function (pmf) and cumulative distribution function (cdf).



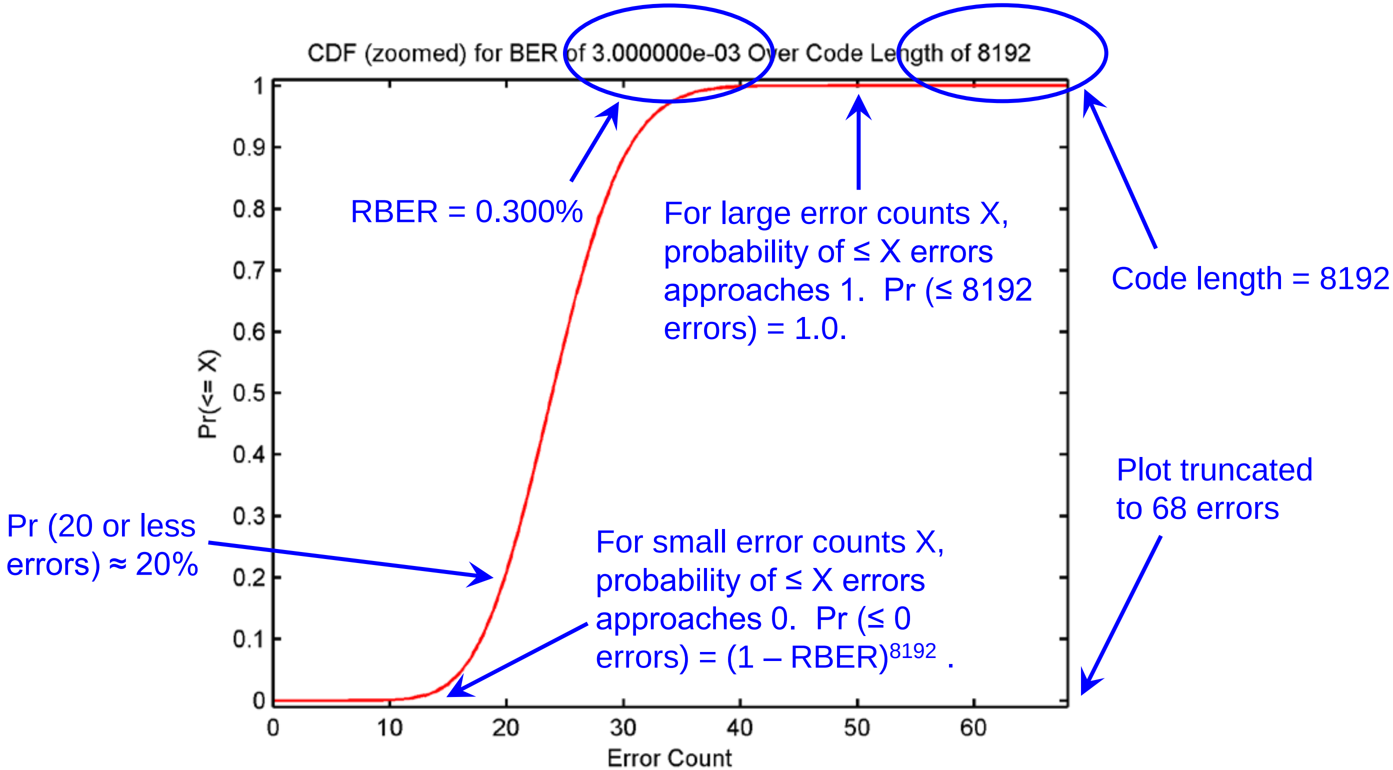


Extension to the Real World



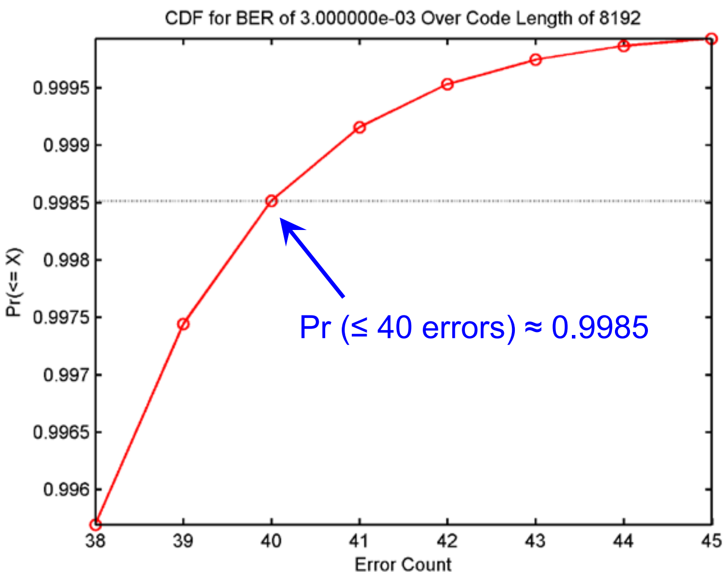
- Consider a collection (codeword) of 8192 bits, written to and then retrieved from a memory storage device, with RBER = 3.0e-3.
- The pmf illustrates the probability of occurrence for each possible error count within a code word.

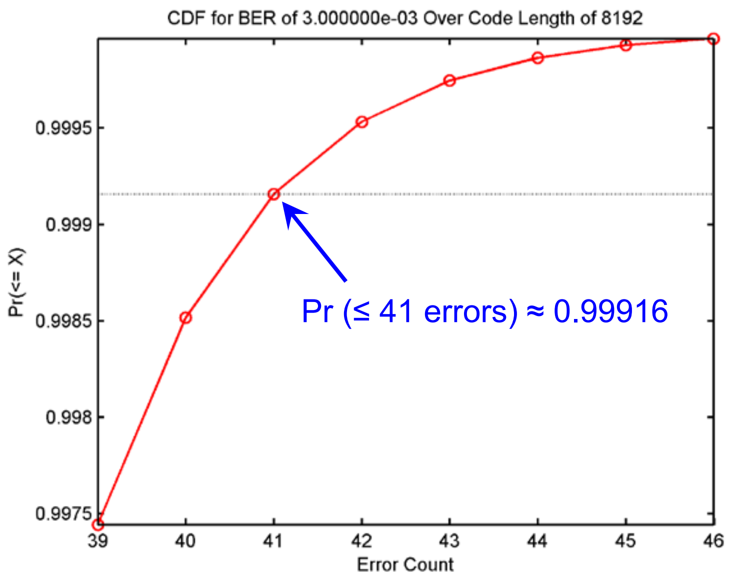
Extension to the Real World



- The cumulative distribution function (cdf) is the summation (integral) of the probability mass function.
- Given a specific number of errors, the cdf illustrates the probability of having less than or equal to that number of errors within a code word.

Extension to the Real World





- Assume that we can correct 40 errors within a code word. Probability of not successfully correcting = Pr (> 40 errors) = 1 - Pr (≤ 40 errors) ≈ 0.0015. This is called the frame error rate, or FER.
- Correcting 41 errors drops frame error rate to ≈ 1 – 0.99916 = 0.00084. A 2.5% increase in correction strength yields a 44% reduction in frame error rate!

RBER, UBER, and the Magic of Correction

- To understand the error characteristics of corrected code words, we need to understand how error correction changes the previous distribution.

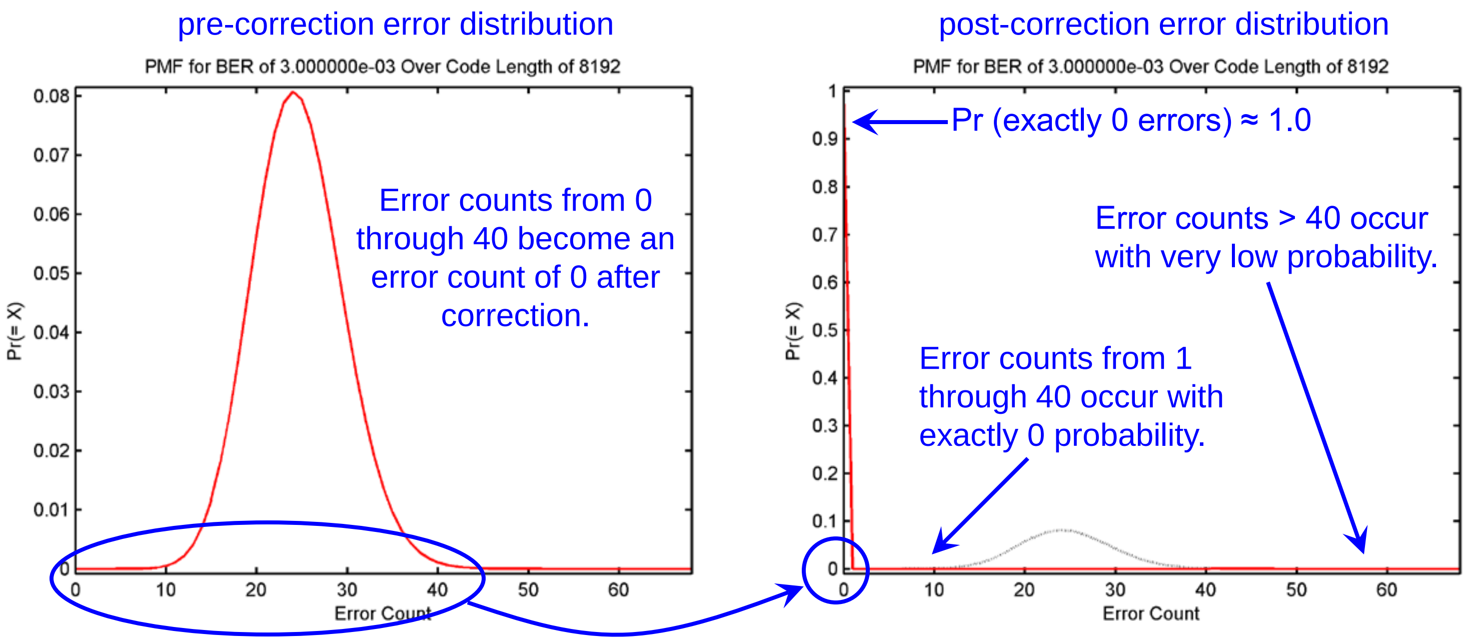
- Assume that we can correct exactly t errors in each code word.

- After correction, there will be NO code words with error counts ranging from 1 through t. Corrected code words will have either 0 errors or > t errors.

- In the corrected code word, Pr (0 errors) = the probability of having from 1 through t errors in the original code word.

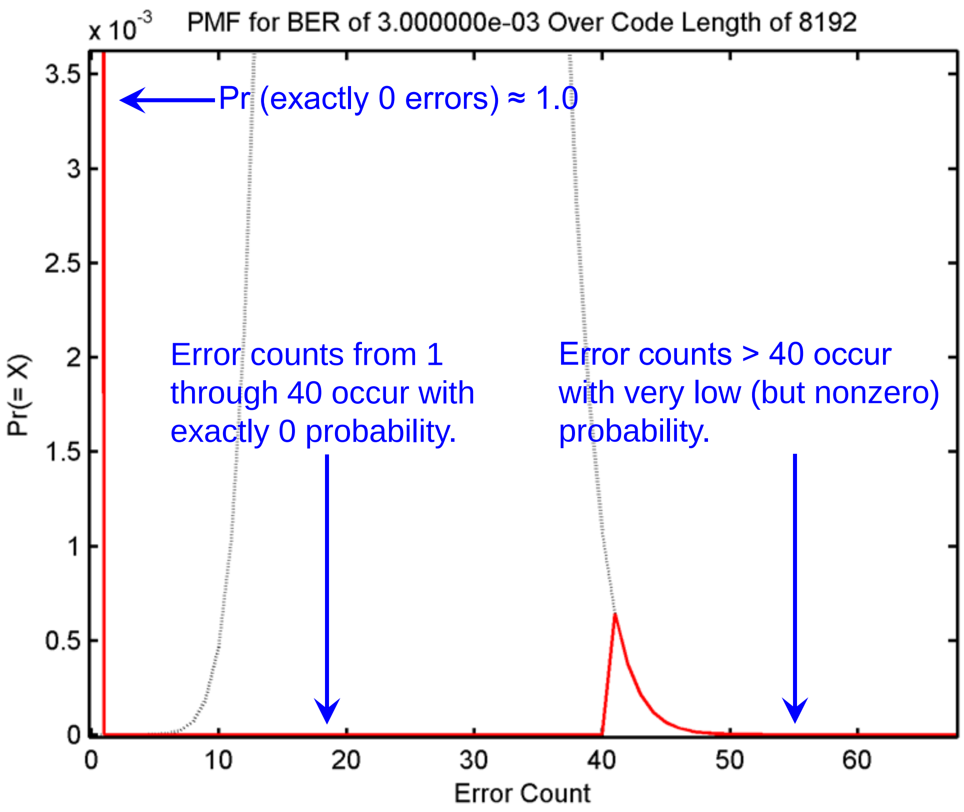
- The distribution of error counts in the corrected code word is *heavily* biased towards 0 errors.

RBER, UBER, and the Magic of Correction



- Assume that we can correct 40 errors in each codeword.
- Error correction modifies the original distribution by "piling up" pre-correction error counts from 0 through 40 into the post-correction "0-error" bin.
- Error counts greater than 40 occur with exactly the same probability as before. Average error count, however, is dramatically reduced.

RBER, UBER, and the Magic of Correction



- Looking closely at the pmf of corrected code word errors illustrates the fact that error count probabilities have “piled up” at 0.
- How do we use this distribution to calculate the bit error rate for corrected data?

RBER, UBER, and the Magic of Correction

- Post-correction bit error rate is called UBER, short for uncorrected bit error rate. UBER is the industry-standard metric for evaluating error correction performance in Flash memory.
- If we know the distribution of possible errors within a code word, i.e. the pmf, then calculating the uncorrected bit error rate is very straightforward.

$$UBER = \frac{\sum_{k=0}^{l} k \times \Pr(k)}{l} = \frac{\sum_{k=t+1}^{l} k \times \Pr(k)}{l}$$

- Note that the summation can start at t+1, since all other summation terms below t+1 are 0 for an error correction scheme with strength t.

Tradeoffs and Numerical Examples

Correction strength has a significant impact on UBER. As correction strength varies from 37 to 43, at an RBER of 2.00e-3, UBER decreases by a factor of more than 250.

Code Length	RBER	Strength (t)	Code Rate	UBER
8192	2.00e-3	37	0.937	1.612e-08
8192	2.00e-3	38	0.935	6.808e-09
8192	2.00e-3	39	0.933	2.805e-09
8192	2.00e-3	40	0.932	1.128e-09
8192	2.00e-3	41	0.930	4.426e-10
8192	2.00e-3	42	0.928	1.697e-10
8192	2.00e-3	43	0.927	6.362e-11

↓ 250x

Tradeoffs and Numerical Examples

RBER also has a significant impact on UBER. As correction strength varies from 37 to 43, at an RBER of 1.25e-3, UBER decreases by a factor of more than 4000!

Code Length	RBER	Strength (t)	Code Rate	UBER
8192	1.25e-3	37	0.937	1.016e-13
8192	1.25e-3	38	0.935	2.705e-14
8192	1.25e-3	39	0.933	7.012e-15
8192	1.25e-3	40	0.932	1.775e-15
8192	1.25e-3	41	0.930	4.383e-16
8192	1.25e-3	42	0.928	1.057e-16
8192	1.25e-3	43	0.927	2.489e-17

↓ 4000x

Tradeoffs and Numerical Examples

If we fix correction strength at 40, and vary RBER from 2.75e-3 to 1.25e-3, UBER decreases by a factor of more than 840,000,000!

Code Length	RBER	Strength (t)	Code Rate	UBER
8192	2.75e-3	40	0.932	1.503e-06
8192	2.50e-3	40	0.932	2.116e-07
8192	2.25e-3	40	0.932	1.987e-08
8192	2.00e-3	40	0.932	1.128e-09
8192	1.75e-3	40	0.932	3.373e-11
8192	1.50e-3	40	0.932	4.350e-13
8192	1.25e-3	40	0.932	1.775e-15

↓ 840,000,000x

Tradeoffs and Numerical Examples

- So far, we have focused on evaluation of a correction scheme using a fixed code word size.
- What if wish to change the length of the code word?
- Shorter code words are generally less efficient, but require less processing resources and deliver lower read latency in an *absolute* sense.
- Longer code words are generally more efficient, but require more processing resources and deliver higher latency in an *absolute* sense.
- The key is to choose a correction strength that delivers the same or lower UBER.

Tradeoffs and Numerical Examples

Correction strength for a shorter or longer codeword must be chosen to meet required UBER. Given a fixed correction strength of 40 over 8192 bits, what strength is required over 4K or 16K to achieve the same UBER?

RBER	length = 8192 strength:UBER	length = 4096 strength:UBER	length = 16384 strength:UBER
1.25e-3	40 : 1.775e-15	29 : 3.503e-16	60 : 1.308e-15
1.50e-3	40 : 4.350e-13	28 : 1.499e-13	62 : 2.567e-13
1.75e-3	40 : 3.373e-11	27 : 1.964e-11	64 : 1.571e-11
2.00e-3	40 : 1.128e-09	26 : 1.052e-09	65 : 8.621e-10
2.25e-3	40 : 1.987e-08	26 : 9.624e-09	67 : 1.151e-08
2.50e-3	40 : 2.116e-07	25 : 1.645e-07	68 : 1.676e-07
2.75e-3	40 : 1.503e-06	25 : 7.519e-07	69 : 1.480e-06

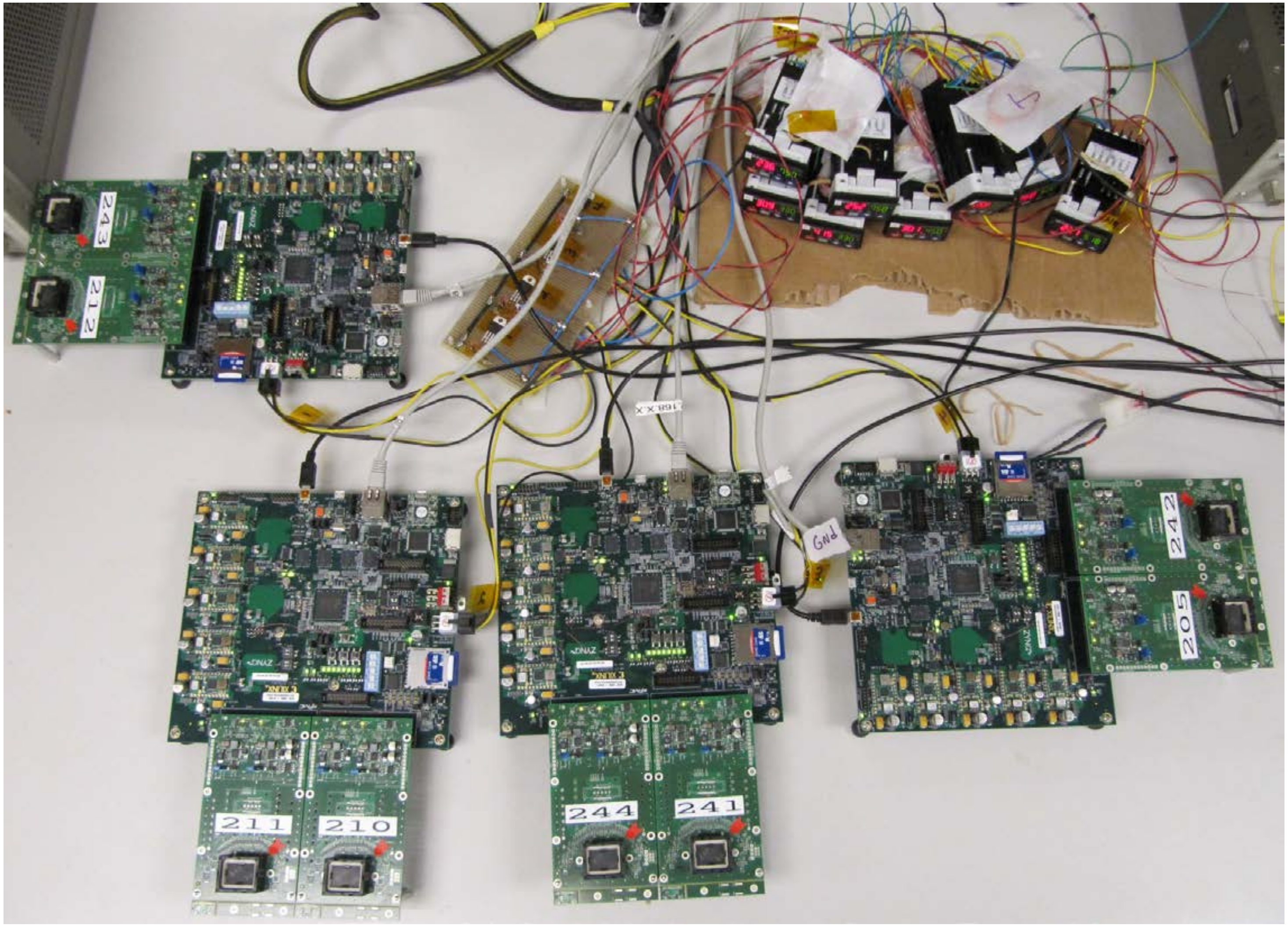
Tradeoffs and Numerical Examples

- Understanding the performance of a particular codeword length and correction strength requires us to calculate UBER. This requires knowledge of RBER.
- Unfortunately, Flash device manufacturers do not generally specify RBER!
- More importantly, RBER varies with Flash wear, temperature, and a variety of other factors that are often difficult to control, let alone predict.
- For these reasons, as well as others, it is far easier to simply do what the manufacturer recommends.
- Unfortunately, this is not going to satisfy enterprise customers, who demand to know the performance and expected lifetimes of their storage systems.
- It is also not going to work in a competitive industry characterized by “pushing the envelope”.
- We simply need to know more… We need to dig deeper!

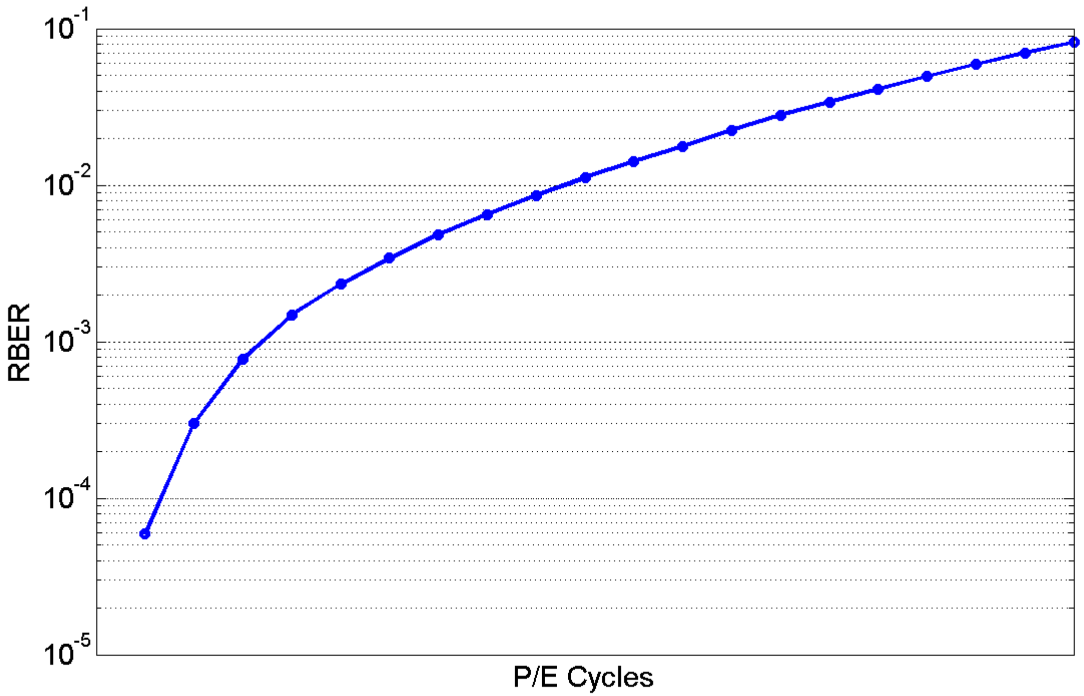
The Quest for Deeper Knowledge

- Why are We Here?
- Error Correction Fundamentals
  - A Simple Channel/Storage Model
  - Extension to the Real World
  - RBER, UBER, and the Magic of Correction
  - Tradeoffs and Numerical Examples
- The Quest for Deeper Knowledge
  - Characterization
  - Analysis
- Questions?... Comments?

IBM Flash Characterization Platform



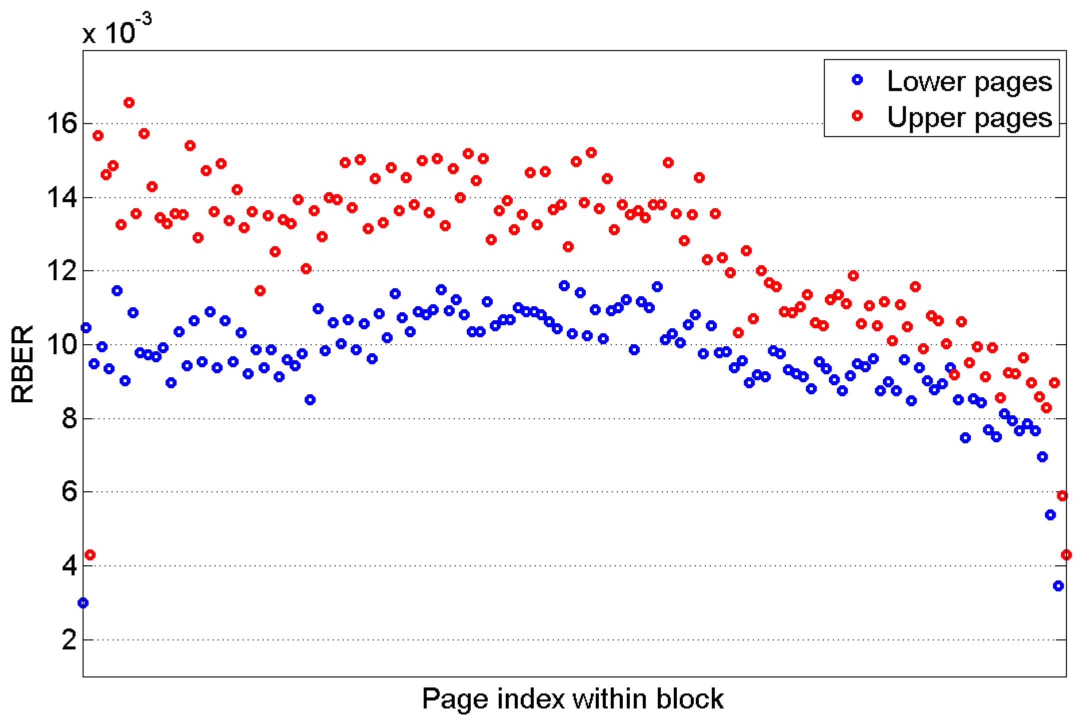
Characterization / Analysis



- As one might expect, as Flash cells are used (e.g. programmed and erased) their reliability worsens and the probability of reading a bit incorrectly (RBER) increases
- Extreme P/E cycle conditions lead to an RBER that exceed 1e-2
- If we really want to *"push the envelope"* then we must be prepared to deal with reading 1 in every 100 bits incorrectly!

Characterization / Analysis

- RBER is not completely determined by P/E cycles
- It has been established in the literature that RBER can vary across blocks (two blocks subjected to the same number of P/E cycles may have completely different RBER)
- RBER can even vary within a block (from page to page) as shown below:



Characterization / Analysis

- How can we achieve operational UBER < 1e-15 given RBER = 1e-2 using BCH codes?
- For a code length of 8192 bits we would need correction strength t=157. This corresponds to a code rate of 0.73 which is very low for storage applications
- To become more efficient we can try increasing the BCH code length:
    - Higher code rate is achieved (0.73 → 0.78)
    - BUT: the implementation complexity does not scale well (t=1585 ?!)
- A different approach is required

Code length	RBER	Strength (t)	Code Rate	UBER
8192	0.01	157	0.732	8.210e-16
16384	0.01	267	0.756	6.627e-16
32768	0.01	469	0.771	9.614e-16
65536	0.01	852	0.779	8.691e-16
131072	0.01	1585	0.782	8.955e-16

Questions / Comments

- Why are We Here?
- Error Correction Fundamentals
  - A Simple Channel/Storage Model
  - Extension to the Real World
  - RBER, UBER, and the Magic of Correction
  - Tradeoffs and Numerical Examples
- The Quest for Deeper Knowledge
  - Characterization
  - Analysis
- Questions?... Comments?