Bit Error Rate in NAND Flash Memories

Neal Mielke[1], Todd Marquart[2], Ning Wu[1], Jeff Kessenich[2], Hanmant Belgal[1], Eric Schares[1], Falgun Trivedi[2], Evan Goodness[1], and Leland R. Nevill[2]

[1]Intel Corporation, 2200 Mission College Blvd, Santa Clara, CA 95054
Tel 408-765-0300, Fax 408.765-9630, Neal.R.Mielke@intel.com

[2]Micron Technology, Inc., 8000 S. Federal Way, Boise, ID 83707
Tel 208-368-2435, Fax 208-363-2919, TMarquart@micron.com

ABSTRACT

NAND Flash memories have bit errors that are corrected by error-correction codes (ECC). We present raw error data from multi-level-cell devices from four manufacturers, identify the root-cause mechanisms, and estimate the resulting uncorrectable bit error rates (UBER). Write, retention, and read-disturb errors all contribute. Accurately estimating the UBER requires care in characterization to include all write errors, which are highly erratic, and guardbanding for variation in raw bit error rate. NAND UBER values can be much better than 10^{-15}, but UBER is a strong function of program/erase cycling and subsequent retention time, so UBER specifications must be coupled with maximum specifications for these quantities.

I. INTRODUCTION

NAND Flash memories [1][2] are widely used for data storage because of their compactness, low power, low cost, high data throughput, and reliability. Scaling and multi-level-cell (MLC) technology [3] have enabled NAND to replace hard disk drives (HDDs) in portable devices and in some computers. Like HDDs, NAND memories are not intrinsically error-free but rely on error correction coding (ECC) [4] to correct raw bit errors. (Appendix 1 reviews NAND's basic structure and operation.)

Prior work has identified several mechanisms that can lead to bit errors in Flash memories, including program disturb from tunneling and hot-electron injection [5][6][7][8][9], quantum-level noise effects [10][11], erratic tunneling [12][13], SILC-related data retention and read disturb [14][15][16], and detrapping-induced retention [17][18][19][20][21][22][23]. The prior work investigated shifts in memory-cell device characteristics such as the threshold voltage (V_T); the shifts were physically explained and mathematically modeled. Little has been written on the data error rate caused by these shifts. This paper begins with observed raw bit errors, identifies the physical causes, and estimates the rate of failure with ECC applied.

The fraction of bits that contain incorrect data before applying ECC is called the raw bit error rate (RBER). The error rate after applying ECC is called the uncorrectable bit error rate (UBER). HDD manufacturers and NAND-based storage manufacturers often quote UBER values on their datasheets, typically 10^{-13} to 10^{-16} [24]. But little is written about how UBER is defined or measured. This paper presents workable definitions and measurement methods applicable to NAND.

UBER is a useful reliability metric for mass-storage devices such as HDDs because a bit error that damages one file out of many is not equivalent to a functional failure that destroys the drive. UBER is used to specify the data-corruption rate, whereas metrics such as mean time between failure (MTBF) [25] specify the functional failure rate. This distinction is often not made for electronic components; instead, a device is often counted simply as a failure whether it has a single-bit error or a catastrophic functional failure [26]. For NAND components used in mass-storage applications, separating the two makes sense.

ECC schemes supplement user data with parity bits which store enough extra information for the data to be reconstructed if one or more bits are corrupted. The user and parity bits together are called an ECC codeword (CW), typically one 512-byte *sector* of user data plus the parity bits. ECC is implemented externally using a memory microcontroller. MLC NAND datasheets require ECC schemes capable of correcting several errors per CW.

If an ECC scheme can correct "E" failing bits per codeword, then the codeword will have an uncorrectable error if (E+1) or more bits fail. The probability that a codeword will fail is:

$$P_{CW} = \sum_{n=E+1}^{N} \binom{N}{n} \cdot \mathrm{RBER}^{n} \cdot (1-\mathrm{RBER})^{N-n} \qquad (1)$$

Here N is the number of bits per codeword. The term in the sum is the binomial probability that *n* bits will be in error; the sum from E+1 to N is the probability that there are more than E errors. Usually RBER is small. Then (1-RBER) is about 1 and the first term in the sum is dominates, so P_{CW} is proportional to RBER^{E+1}.

As is common practice, we will define the UBER at any instant to be P_{CW} divided by the number of user bits in the codeword:

$$\mathbf{UBER(instantaneous)} = \frac{\mathbf{P_{CW}}}{\mathbf{User\ data\ bits\ per\ CW}} \qquad (2)$$

Figure 1 shows the instantaneous UBER vs. RBER for ECC correctability ranging from 0 (none) to 8 bits per sector. ECC greatly reduces the bit error rate. The curves become steeper with increasing E, consistent with the RBER^{E+1} relationship.

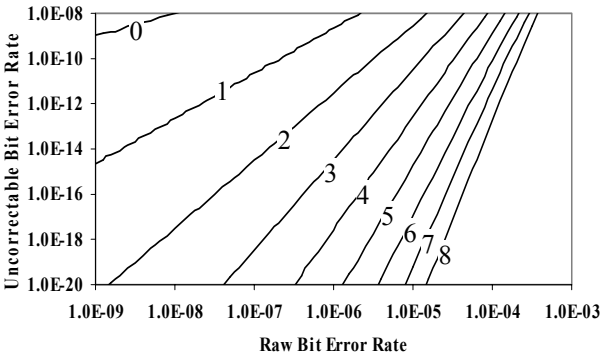

Figure 1: Dependence of the UBER on the RBER for different levels of ECC correctability

II. EXPERIMENTAL DETAILS

Experiments were performed on 8Gb MLC NAND devices manufactured on 63nm to 72nm technology nodes from four manufacturers. Included is a 72nm device manufactured by the

978-1-4244-2050-6/08/$25.00 ©2008 IEEE

joint venture between Intel and Micron (IMFT). The devices were specified for use with 4-bit ECC. Data retention was specified as 10 years for two devices and unspecified for the others. Maximum program/erase (P/E) cycle count was specified at 10K for two devices, at 5K for one device, and was not specified for the fourth. P/E cycling was done up to 10K cycles, data retention for as long as a year, and read disturb to 10K reads per page. Stresses were at room temperature with random data patterns. Raw errors were logged without ECC; RBER and UBER values were determined by analyzing the logs. Sample sizes were sufficient to measure the raw bit error rate but usually not sufficient for 4-bit-ECC failures to appear. The behavior of ECC failures was studied either by applying 1-bit ECC or by using mathematical extrapolation for 4-bit ECC. The non-IMFT devices had smaller sample sizes and more statistical noise.

III. RESULTS

A. Write Errors

Figure 2 shows the RBER measured after writing data to the four devices, as a function of prior P/E cycles. Data were verified and errors logged only at the cycle points marked by the symbols; to speed up the stress, the cycling in between was without error logging. This method is called the sparse-verify method. For all devices, the RBER increases with cycles but stays below the level at which the recommended 4-bit ECC scheme would fail.

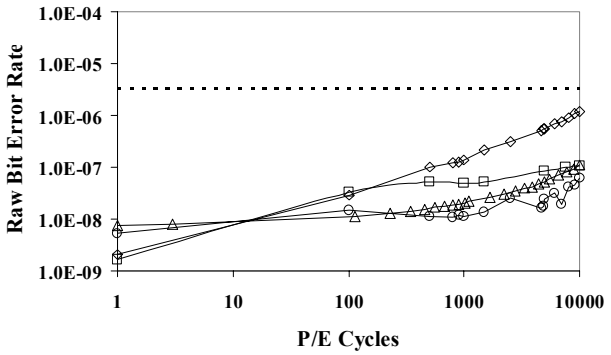

Figure 2: RBER after writing data, as a function of prior program/erase cycles. Errors were logged at the cycle points noted by the symbols. Each symbol represents a different product; these same symbols are used in other figures. The IMFT-manufactured product is Δ. The devices marked by ◊ and O were not specified for 10K cycles. The dashed line is the RBER where the instantaneous UBER reaches 10^{-15} according to Equations (1) and (2). Non-monotonic curves result from small sample sizes and erratic RBER behavior.

Because the data pattern is known, it is possible to identify what V_T level (of the four MLC levels) a failing cell was supposed to have and what V_T level it actually had. Almost all the errors were due to cells with higher V_T's than intended. This behavior is sketched in Figure 3, which shows tails on the upper edges of the level distributions. From bit-error data, only the number of cells exceeding the read points can be deduced, not the V_T's themselves. The figure shows that the percentage weighting of the three tails varied from device to device. Only one device (O) had significant errors from cells with V_T's that were too low.

Cells moving up from L0 were erased cells that became programmed when other cells were being programmed; this is called program disturb. Several mechanisms causing program disturb have been discussed in the literature. During programming, the failing cells were biased with an inhibit bias scheme (Figure 24). Cells on selected strings but deselected wordlines have $V_{INHIBIT}$ on their control gates (CGs) and grounded string potentials. This can lead to some tunneling, called $V_{INHIBIT}$ disturb. Inhibited cells on selected wordlines and deselected bitlines have the full programming voltage on their CGs and rely on a boosted string potential to suppress tunneling; but some tunneling, called V_{PGM} disturb, can still occur. Setting $V_{INHIBIT}$ too low risks V_{PGM} disturb, and setting it too high risks $V_{INHIBIT}$ disturb; minimizing disturb requires a careful balance [5][6]. The underlying physics is Fowler-Nordheim tunneling, but as is known from NOR erase-distribution studies such tunneling is nonuniform [28] and erratic [12][13] in small-dimension devices because of cathode non-planarity and unstable positive charges in the tunnel oxide. As a result, some cells tunnel more easily than others, causing bit errors. Finally, mechanisms related to junction leakage can discharge the boosted string [7] or result in hot-electron injection [8][9].

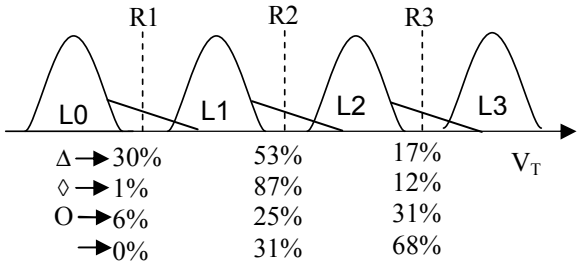

Figure 3: Schematic illustration of the dominant types of write errors and the percentage weighting of each type for each product. The RBER was dominated by cells with higher V_T's than intended; exceptions exist to some degree and result in percentages sometimes adding to less than 100%. The levels and read points R1–R3 are discussed in Appendix 1.

Cells moving up from L1 and L2 were intentionally programmed to one level but ended up too high. These *overprogramming* failures are due to the failure of the MLC programming algorithm (Figure 23). This algorithm tests to see whether a cell has been programmed high enough, but not whether it has been programmed too high. Several mechanisms can cause overprogramming. First, random telegraph noise [10] can fool a verify step in Figure 23, causing a cell to receive an additional pulse which it does not need. Second, erratic tunneling caused by unstable positive charges in the oxide can cause the V_T jump from one pulse to the next to be too large. Third, even in the absence of unstable oxide charges the V_T jump in one pulse can be too large because of Poisson statistics related to the small number of electrons that are injected per pulse [11]. Finally, even if the cell is placed to the proper level it can move higher when adjacent cells are later programmed, because of the coupling between floating gates (FGs) [27]. A given cell can fail because of the combined effects of more than one mechanism.

The RBER increases with cycle count. This is primarily attributed to the buildup of traps in the tunnel oxide. Individual failing bits are highly erratic. For example, for the IMFT device only 26% of the bits that failed at 5K cycles also failed at 10K cycles, and similar behavior was seen for the other devices. So, as RBER increases new bits fail and many old failures recover. The erratic behavior stems from the root-cause physics of the mechanisms as described above, which involve unstable oxide charges and quantum noise. The erratic behavior is very important in the measurement of the UBER, as will be discussed below.

To study the erratic behavior, 4000 blocks were cycled to 10K cycles, with a random pattern that changed every cycle. Failing bits were logged after each write; this method is called the every-cycle-verify method.

Figure 4 shows the behavior of a representative bit. This bit first failed at cycle 4731 and then failed only 10 more times. This bit was more repeatable than most: the average failing bit failed only four times. These errors may seem more erratic than implied by the earlier statement that 26% of bits that failed at 5K cycles still failed at 10K cycles. Indeed, in the every-cycle-verify experiment only 0.01% of the bits that failed before cycle 10K also failed at cycle 10K. There are two reasons for the seemingly huge discrepancy between 26% and 0.01%, discussed next.

First, in the every-cycle-verify experiment a different data pattern was written every cycle, whereas in the sparse-verify experiment the same pattern was written at each verify step (changing patterns were used only for the cycling). A bit failing at one verify is less likely to fail at a later one if the data pattern is different. This is because the failing cell may be written to a V_T level that is less likely to fail and because the adjacent-cell V_T levels will be different, affecting FG-to-FG coupling [27], gate induced drain leakage [7], and other effects. As a result, in the every-cycle-verify experiment 8% (not 26%) of bits failing at 5K cycles also failed at 10K. But this is still a far cry from 0.01%.

The second and dominant effect is best explained by example. Suppose that a device has one firm-failing bit that fails at every cycle, and that at every cycle there is a second erratic failing bit that is different each cycle. There would then be 10,000 failing bits in the first 9,999 cycles (one firm-failing bit and 9,999 totally erratic ones), of which only one (0.01%) would fail at 10K cycles. On the other hand, there would be 2 failing bits at 5K cycles, and one of them (50%) would fail at 10K. The first observation, resulting in the 0.01% value, is possible only with an every-cycle-verify experiment. Both 0.01% and 50% are mathematically correct, but the quantity counted is different. The every-cycle-verify calculation (0.01%) is the percentage of failing bits *with each failing bit included only once.* The sparse-verify estimate (50%) is the percentage of failing bits *with each failing bit included as many times as it fails.* Including the firm error only once would mean including it only when it failed in the first cycle, not at 5K as was done for the 50% calculation. The sparse-verify calculation gives the impression that the typical failing bit is far more repeatable than it really is because it over-counts those rare bits that repeat quite often.

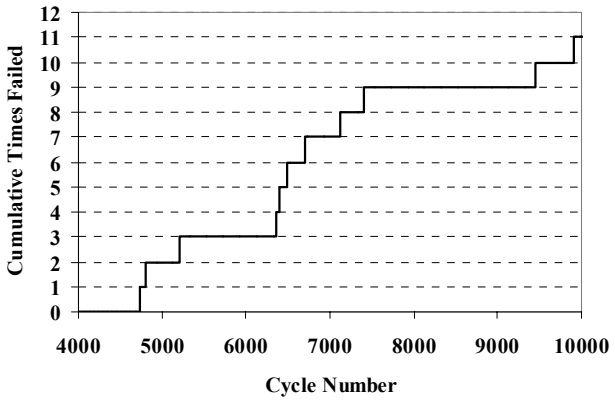

Figure 4: Pass/Fail behavior of a representative bit.

ECC errors, not raw bit errors, are what matter. With the datasheet level of 4-bit ECC there were no ECC errors. But we can study the characteristics of ECC errors by applying only *single-bit* ECC, in which case a number of failures were observed.

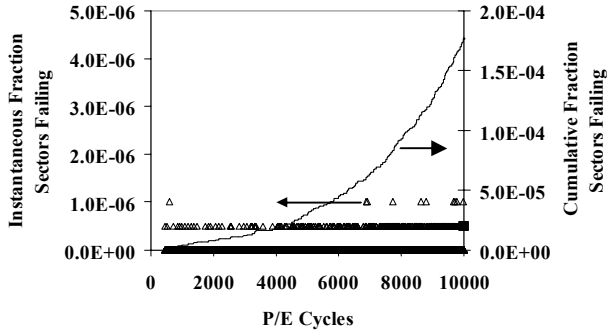

Figure 5: Sectors failing as a function of P/E cycles (IMFT device). Δ: Instantaneous fraction of sectors failing at each cycle. Smooth curve: Fraction of sectors that failed up to and including the indicated cycle number. Each failing sector is counted only the first time that it fails. The instantaneous data are quantized because there were ≤2 errors at any instant.

For the every-cycle-verify experiment, Figure 5 shows the fraction of sectors failing single-bit ECC as a function of cycles. The cumulative curve counts all sectors failing up to a particular cycle number and is simply the integral of the instantaneous data. The cumulative errors were two orders of magnitude larger than the instantaneous errors: 354 sectors failed but never more than 2 at the same cycle. This is a manifestation of the erratic nature of most failing bits. In fact, the ECC failures are even more erratic than the underlying bits: the average failing bit failed 4 times but the average failing sector failed less than twice. It is expected, and observed by us in other experiments, that failures repeat fewer and fewer times as one progresses from no ECC (raw bit errors) to single-bit ECC to multiple-bit ECC. This is because it is less likely for a group of failing bits to all repeat than for any one of them to repeat. For the datasheet level of 4-bit ECC, it is expected that the average ECC failure would occur in only one cycle and then never fail again.

Because of the erratic nature of ECC failures, it is important that NAND write-error reliability not be measured simply by counting the errors from sparse-verify experiments. This would result in the vast majority of failures not being detected. Ideally, one would always have data from every-cycle-verify experiments. Yet it is common practice for non-volatile memory characterizations to involve full data-integrity testing only at specific verify points. In particular we do not have every-cycle-verify data for the data sets shown in Figure 2. We next discuss how the ECC failure rate be deduced from sparse-verify data.

Figure 6 overlays the data from the sparse-verify and every-cycle-verify experiments. The instantaneous fraction fail in the sparse-verify experiment is about three orders of magnitude lower than the cumulative fraction in the every-cycle-verify experiment, confirming the gross under-counting that can result from using sparse-verify data "as is". The upper curve (▲) is the result of *integrating* the sparse-verify curve (Δ), interpolating values between the data points. Integrating the curve is equivalent to counting the failures that occurred between verifies. The integration result matches the every-cycle-verify data reasonably well, about a factor of two greater. This overestimation is a general consequence of the integration method which can be understood and compensated for. In a sparse-verify experiment, the instantaneous data points (Δ in Figure 6) include all failing sectors at each verify point, including any that may have failed earlier. Integrating the curve therefore estimates the total failing sectors *with a given sector counted more than once if it fails more than once.* By contrast, in the every-cycle-verify experiment we

counted a sector as failing only the first time that it failed, because in field use a sector will be retired if it fails. As a result, the integration method overestimates the failure rate by a factor equal to the average number of times that a failing sector fails. With single-bit ECC, the average sector failed 1.8 times, resulting in that degree of overestimation. (The results also differ slightly because the RBERs of the two experimental samples were somewhat different). As already mentioned our expectation with multiple-bit ECC is that this factor approaches unity so that the overestimation will be negligible.

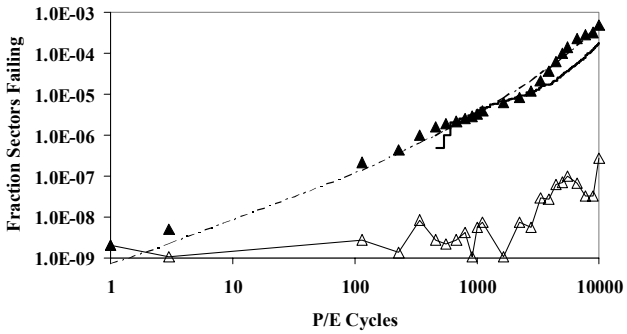

Figure 6: Sectors failing single-bit ECC as a function of cycles for the IMFT product for both the every-cycle-verify and sparse-verify experiments. Δ: Instantaneous fraction from the sparse-verify experiment. ▲: Integral of the instantaneous curve. Solid line: Cumulative data from every-cycle-verify experiment (from Figure 5). Dashed line: Prediction from binomial statistics (Equation 1) with a small correction for RBER variability.

Another issue that arises is that because of insufficient sample size an experiment fails to detect any ECC failures at all. The most straightforward way is to replace the fraction of sectors failing (Δ symbols in Figure 6) with binomial estimates (Equation 1) in the integration. In principle, one merely needs to measure RBER, which can be done with a small sample size. A difficulty is that Equation (1) assumes that RBER is uniform, whereas it actually varies from device to device and from sector to sector. This variability increases the failure rate and must be guardbanded for. How this can be done is covered in the Discussion section. The dashed line in Figure 6 is the result of using this approach. This dashed line matches the result (▲) from integrating the actual data (Δ).

Figure 7 shows the estimated cumulative fraction failing for all four devices. The devices have similar qualitative behavior, although one device has a much higher error rate.

B. Retention Errors

Nonvolatile memory reliability characterization methods typically consist of P/E cycling followed either by an unbiased bake or by a biased read-disturb stress [26]. These post-cycling stresses check whether the device is able to retain data over time.

Figure 8 shows RBER during a room-temperature bake following 10K P/E cycles. RBER at time=0 is due to write errors induced by cycling (Figure 2). RBER climbs with time because of data-retention errors.

Retention errors are mostly due to charge loss: cells losing charge and thus moving from one V_T level to the one below. Two dominant mechanisms cause this: 1) loss of FG charge via stress-induced leakage current (SILC) through the tunnel oxide [29][14][15][16]; and 2) detrapping of tunnel-oxide charge that had been trapped during cycling [17][18][19][20][21][22]. The effect on the V_T distributions is sketched in Figure 9. Detrapping causes the distributions to intrinsically broaden and shift lower. SILC causes a subset of cells to lose charge, forming a tail in the distribution. Because SILC is more strongly field-dependent than detrapping is [16][21], SILC tends to dominate the RBER of L3 cells, which have the largest electric field in the tunnel oxide because they have the most stored electrons. Detrapping tends to dominate the RBER of L1 and L2 cells. Interestingly, the same detrapping that generates retention errors also causes some write errors (from the final cycle) to recover with time because some of the tail cells that were above their intended read level in Figure 3 drop below that read level due to charge loss. Over the retention period of Figure 8, about a third of the write errors recovered.

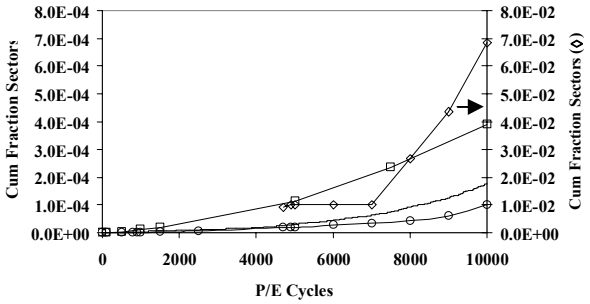

Figure 7: Cumulative fraction of sectors failing vs. cycles for several devices. Smooth line: IMFT device (every-cycle-verify experiment). ◊: Integration of observed failing sectors, plotted against the right-hand axis. and O: Integration of binomial estimates. Results from the sparse-verify experiments were divided by 1.8 to correct for double-counting.

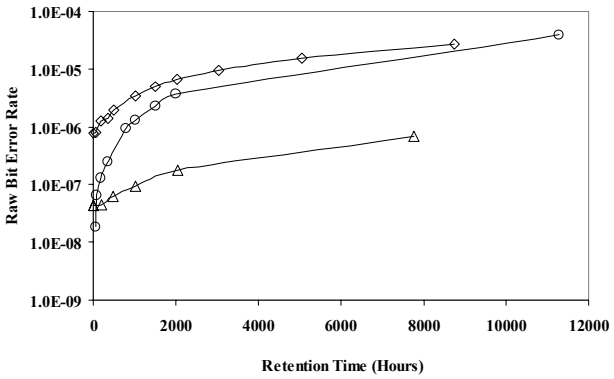

Figure 8: RBER vs. room-temperature retention time following 10K P/E cycles. Symbols have the same meaning as in Figure 2. Extended retention data are not available for . Devices ◊ and O were not specified for 10K cycles. Differences between the initial RBER here and the final RBER in Figure 2 are due to variation from sample to sample and (for curve O) small-sample statistical noise in the early data points.

Both retention mechanisms are represented in the products studied, but their relative contributions vary. The errors were mostly L3→L2 type in the IMFT (Δ) and one other (◊) products, L2→L1 type in the third product (O), and both these types in the data from shorter retention times available for product ( ). Based on Figure 9, the L3→L2 errors would be attributed to SILC and the L2→L1 errors to detrapping. The characteristics of the retention errors can be more clearly seen by plotting the charge-loss errors alone, with write errors removed, as shown in Figure 10. Curve Δ in the lower figure shows that RBER scales as a power law in cycles with exponent somewhat less than unity, which is consistent with what is known for SILC [16]. Our direct measurements of the V_T distribution (not shown) confirm the presence of a SILC-like tail as sketched in Figure 9. Curve (O)

has a much steeper dependence on cycling count, which is what is seen for the detrapping mechanism because of its intrinsic nature. Although curve (◊) is dominated by L3→L2, the increasing cycling slope suggests that the physical mechanism may be a mix of both SILC and detrapping.

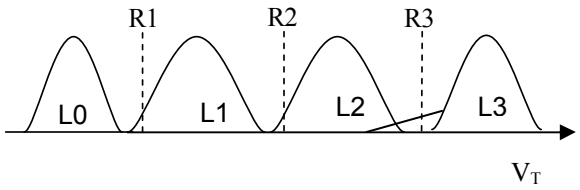

Figure 9: illustration of the sources of retention RBER.

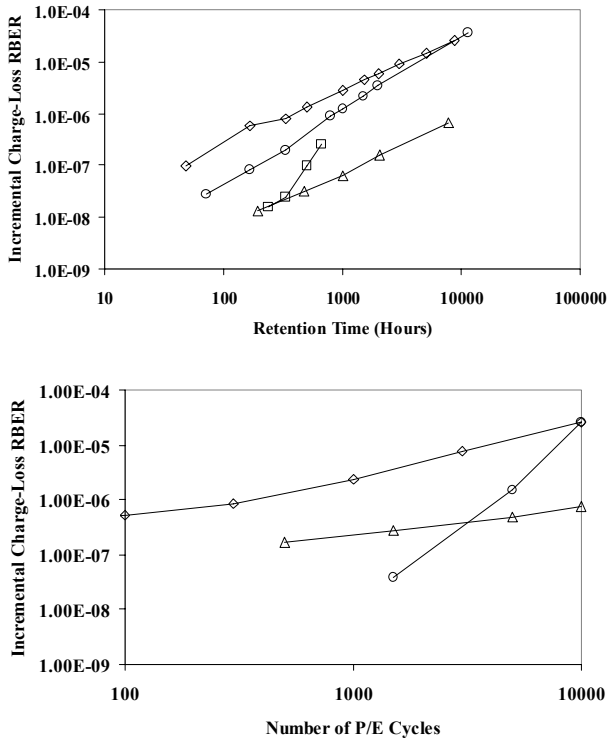

Figure 10: Charge-loss RBER as a function of (top) retention bake time after 10k P/E cycles and (bottom) number of P/E cycles followed by 1 year of bake. The bake was at room temperature. Data are available over only a short retention time (672 hours) for product .

The relative contributions of the mechanisms also depend on the cycling and bake conditions. These devices were cycled at room temperature over several days and then baked at room temperature. If the cycling had been done at higher temperature or over a longer time then the detrapping contribution would have been reduced because some traps would have annealed in the delays between cycles [21][22]. On the other hand, if the retention bake had been done at high temperature then the detrapping would have been larger and the SILC smaller. This is because detrapping is strongly temperature accelerated [21][22] whereas SILC anneals at high temperature [16][30]. In fact, it is often thought that the detrapping mechanism is significant only at high temperature, but this discussion shows that some products under some conditions may be dominated by detrapping even at room temperature. The products dominated by detrapping charge-loss might have had substantially better retention had they been cycled over a more realistic time, such as a year.

It is reasonable to assume that any given verify step records all errors that occurred up to that point in time, because once a cell shifts from one level to the one below it is not likely to shift back. It is therefore not necessary to integrate failure data as we did with write errors. This is an approximation, because telegraph noise [10] and the recovery of write errors can lead to errors that subsequently recover, but these effects should be small.

Figure 11 shows the cumulative fraction of sectors failing single-bit ECC as a function of total stress. For the IMFT device, the retention errors contribute little to the ECC failure rate, which is dominated by write errors. This may seem surprising because Figure 8 shows that retention errors dominate the RBER. The explanation is that the write errors are erratic whereas retention errors are not. The total number of write errors is far greater than suggested by the RBER at the end of cycling because errors from all earlier cycles must be counted. For retention errors, this is not true, because all errors appear in the final data point.

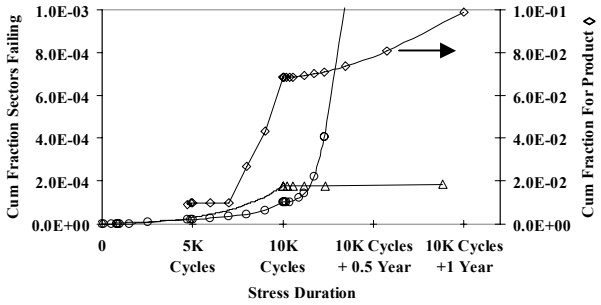

Figure 11: Cumulative sectors failing with single-bit ECC as a function of stress duration in the sequential stress of 10K cycles followed a long retention period. The write errors are from Figure 7. ◊ is plotted against the right-hand axis. Retention errors are by direct counting of actual failing sectors, except for the first few data points for curve O which rely on binomial extrapolation.

C. Read-Disturb Errors

Figure 12 shows RBER as a function of the number of reads per page performed on devices cycled 10K times. The generation of errors with reads is called read disturb.

Whenever a NAND cell is read (Figure 13), a voltage V_{PASS} is applied to all deselected wordlines in the block. V_{PASS} must be higher than the highest V_T of the programmed cells so that the deselected cells do not block the current from the cell being read. The V_{PASS} bias tends to disturb bits up in V_T either though SILC [14][23], which allows electrons to reach the floating gate, or through the filling of traps in the tunnel oxide [17].

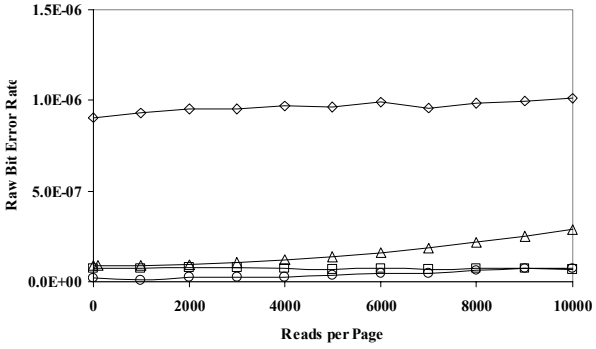

Figure 12: RBER vs. number of reads following 10K cycles.

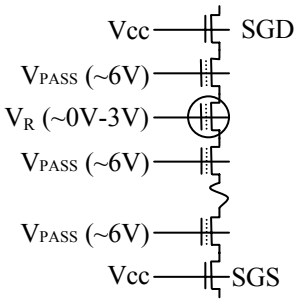

Figure 13: Read bias. The circled cell is the one being read.

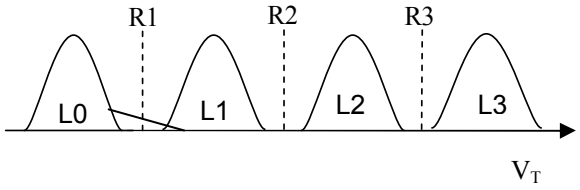

Figure 14: Illustration of the sources of read-disturb RBER

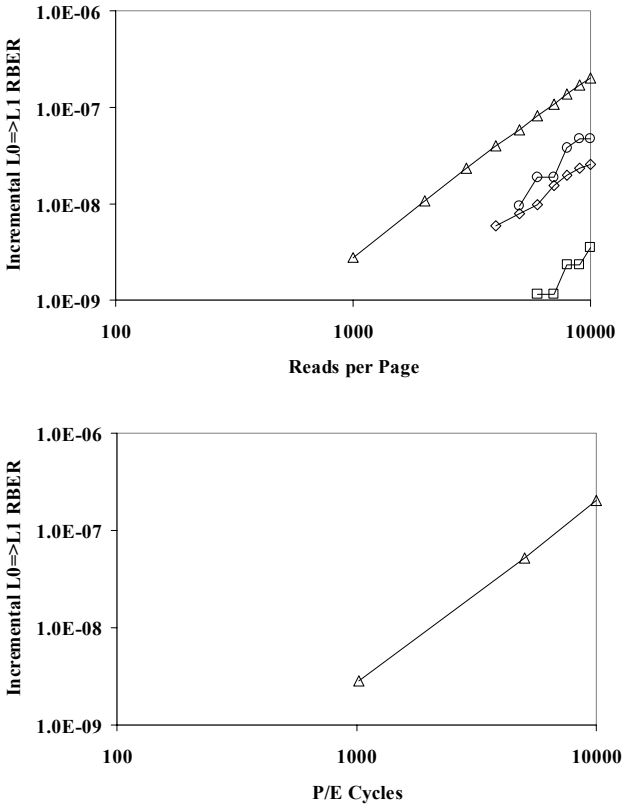

Figure 15: Incremental RBER from read disturb as measured by L0 bit failures not present in the first read, as a function of (top) number of reads per page for 10K-cycled blocks and (bottom) number of cycles before the 10K reads per page.

The failures in Figure 12 are overwhelmingly from L0 cells moving higher, as illustrated in Figure 14. This is as expected for the SILC mechanism, which is strongly field dependent, because the lowest V_T state has the highest electric field in the tunnel oxide under read bias. The characteristics of the read-disturb failures are best studied by excluding the write errors and plotting only the incremental read-disturb errors, as shown in Figure 15. The RBER grows as a power-law in reads (top figure) and in P/E cycles (bottom), consistent again with SILC [16].

As with retention, it is a reasonable approximation that any verify step contains all prior errors so that integration of errors is unnecessary. The lack of erratic behavior is evident in the data: 95% of the failing bits at 5K reads also failed at 10K reads. Figure 16 shows the cumulative fraction of sectors failing single-bit ECC as a function of total stress. Read disturb has a negligible contribution to the ECC failure rate, which is dominated by write errors. As with retention, the dominance of write errors is due to the cumulative impact of write errors, which must be integrated over all cycles.

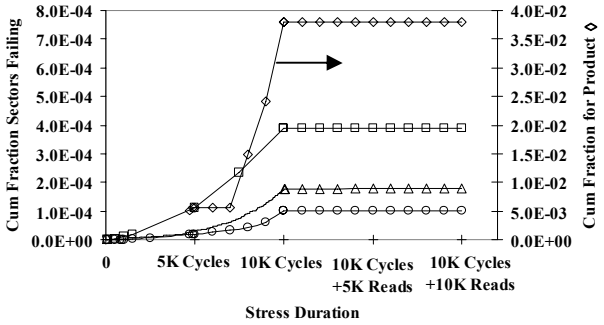

Figure 16: Cumulative sectors failing as a function of stress in a sequential stress of 10K cycles + 10K reads/page. Write errors are from Figure 7. Curve ◊ is plotted against the right-hand axis.

IV. DISCUSSION

For single-bit ECC, the post-ECC failure rate is well summarized by Figures 11 and 16, which show cumulative sectors failing as a function of total usage. It is common in reliability work, however, to seek to summarize reliability through a single parameter, such as the MTBF or the UBER.

How could a UBER value be extracted from the figures mentioned? The typical HDD datasheet statement is that UBER is less than some value *per bit read.* Because NAND failures are not uniform in time (see Figs. 11 and 16), what makes sense is to calculate the UBER based on the cumulative results through any point of time of interest:

$$\mathrm{UBER}(t) = \frac{\text{Cum fraction sectors failing up to time t}}{(\text{bits per sector}) \cdot (\text{number of reads per sector in time t})} \qquad (3)$$

The numerator can be determined given a particular use condition based on experimental data such as was done earlier in this paper. One must then determine the denominator. A general usage model for nonvolatile memory devices is a period of extensive P/E cycling, such as Ncyc cycles, followed by a data-retention (or disturb) period with Nret reads per sector. The UBER can then be re-stated as follows:

$$\mathrm{UBER}(t) = \frac{\text{Cum fraction sectors failing up to time t}}{(\text{bits per sector}) \cdot (\text{Ncyc} \cdot \text{Reads/Cycle} + \text{Nret})} \qquad (4)$$

where Reads/Cycle is the number of times, on average, that each sector is read between cycles, i.e., the read-to-write ratio.

For a known application, one could use the expected values for Reads/Cycle and Nret. For general reliability specification, one can look at practice in the HDD industry. One such practice [24] is to evaluate UBER by writing a drive and reading it back, over and over until (for example) 10^{15} bits have been read, and then calculate the UBER by dividing observed errors by the number of

bits read. Such an evaluation has a read-to-write ratio of unity. For NAND one could therefore assume that Reads/Cycle=1. This is clearly the most conservative possible choice of the number of reads between cycles to be used in a UBER calculation for write and retention errors. That is because NAND write and retention errors are caused by the writes and the retention time, not the reads. So the numerator does not increase significantly if more than one read is performed between writes; adding such reads would simply increase the denominator. One read between cycles is also worst case for read disturb, because adding reads between cycles would not increase the read-disturb failures (in the numerator) by as much as the number of reads in the denominator (this is demonstrated in Appendix 2).

For applications involving a read-disturb retention period, Nret should simply be set to the number of reads in the read-disturb stress, such as 10K in our examples here.

For unbiased retention, one could conservatively assume Nret=0. In our example of 10K cycles plus 1 year of retention, it is plausible that Nret would be small compared to Ncyc. But there are also specifications for data retention for uncycled or lightly-cycled devices, and then it is not reasonable for Nret to be small compared to Ncyc. Thus one should assume some non-zero Nret. During cycling, we assume that the device is read once per cycle, or 10K times over its lifetime in this example. One could assume this read rate even during a retention period. In practice, UBER values are normally specified with only order-of-magnitude levels (10^{-13} vs. 10^{-14} vs 10^{-15}, for example), so it may be acceptable to use a rough estimate for Nret.

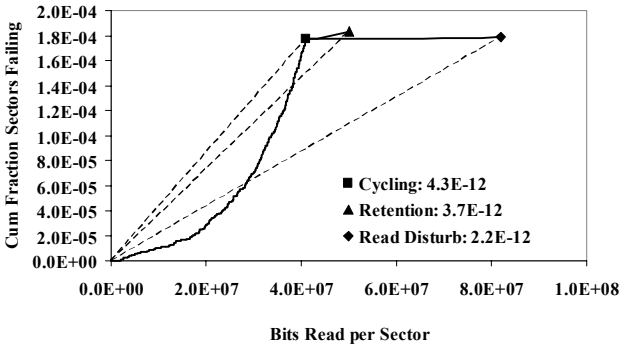

Figure 17: Cumulative fraction of sectors failing with 1-bit ECC for the IMFT device vs. number of reads per sector. Devices are cycled 10K times and then receive either a 1-year data-retention period or a period with 10K reads per sector. The symbols mark the points at which one might evaluate the UBER: at the end of cycling, retention, and read disturb. The legend indicates the UBER values, which are the slopes of the dashed lines.

Figure 17 re-plots the IMFT data from Figures 11 and 16 against the number of bits read per sector. Each sector is assumed to be read once per P/E cycle, at a rate of 2.5K cycles per year during the retention period, and once for each read during the read-disturb stress. The UBER ranges from 2.2E-12 to 4.3E-12 depending on which point in time one chose for the calculation. The worst-case value is obtained for cycling itself due to write errors; the UBER goes down with retention and read-disturb time. These values are greatly exaggerated by the use of single-bit ECC for devices specified for use with 4-bit ECC.

For a NAND device to meet some UBER requirement, it would be necessary to state the use-condition envelope over which that UBER would be met and then to ensure that the worst-case UBER within that envelope met the requirement. For example, from Figure 17 the use-condition envelope would extend to 10K cycles and to at least 1 year of retention and 10K reads per page, and the worst-case UBER would be 4.3E-12.

HDDs are commonly assumed to have constant UBER. Using the example of Figure 17, an HDD specified with a UBER of 4.3E-12 would follow the uppermost dashed line. A NAND device specified for 10K cycles with a UBER of 4.3E-12 would have fewer failures than the HDD for any condition except at precisely the maximum specification. For example, if a typical application had only half the cycles of the worst-case specification, then the UBER would be lower by a factor of 3. This effect becomes larger for multi-bit ECC and is discussed further at the end of this paper.

What is desired, of course, is an estimate of the UBER with the datasheet 4-bit ECC scheme. There were no failures with 4-bit ECC, so we now turn to how binomial estimates can be made, guardbanded for variation in RBER.

Table 1 works out a simple example. The cells on a single wordline are divided between odd and even pages. The even page is written before the odd page, which results in different RBER values (because of FG-to-FG coupling [27] and other effects). The table begins by showing the write-error RBERs at 10K cycles for odd and even pages, normalized to the average of the two. The UBER rows show the calculated UBER of the odd and even pages combined, relative to the UBER that would result from uniform RBER. For example, consider 1-bit ECC, where UBER scales as $RBER^2$. For the IMFT device, the odd-pages RBER is 1.25 times the average of odd and even. This means that the odd-page UBER is $1.25^2 = 1.56$ times the value that would have occurred had the RBER been uniform. The same calculation is $0.75^2 = 0.56$ for the even pages, and the overall UBER is increased by a factor of (1.56+0.56)/2 = 1.06. In the jargon of yield modeling, this 1.06 factor can be considered a cluster factor (CF). If one uses an unguardbanded RBER in Equation (1), one needs to multiply the P_{CW} value by the UBER CF to obtain the correct value. One could equivalently guardband the RBER instead. Guardbanding P_{CW} by a factor of some UBER CF is equivalent to guardbanding the RBER by $(\text{UBER CF})^{1/(E+1)}$. The table also shows these RBER CFs. The RBER CFs are different for different NAND devices, which is the result of differences in process, circuit design, and internal algorithms. Indeed, sometimes even pages are worse than odd pages and sometimes vice versa.

		IMFT	◊	O	
RBER	**Even**	0.75	0.51	1.13	1.39
	Odd	1.25	1.49	0.87	0.61
UBER CF	**1-bit ECC**	1.06	1.24	1.02	1.16
	4-bit ECC	1.65	3.74	1.17	2.68
RBER CF	**1-bit ECC**	1.03	1.12	1.01	1.07
	4-bit ECC	1.11	1.30	1.03	1.22

Table 1: Odd/even page effect on ECC failure rate for write errors. Symbols have the same meaning as in the figures.

The odd/even page difference is only one of several sources of variation in NAND RBER. For example, RBER can vary significantly by position within the string and from one unit to another. As a result, the effect of variability is greater than Table 1 indicates. It is beyond the scope of this paper to review all sources of variation in detail. But they can all be addressed using essentially the same method described in Table 1: Analyze the RBER variation and find its impact on UBER by using the power-law relationship between RBER and UBER.

	IMFT	◊	O	
1-bit ECC	1.2	1.4	1.4	1.2
4-bit ECC	2.1	3.0	3.2	2.3

Table 2: Estimated write-error RBER CFs for all sources of variation combined. The values were obtained by multiplying the CFs obtained for each individual source of variation.

Table 2 summarizes the RBER CFs for all sources of variation combined. Note that the CFs increase with increasing ECC. This is a consequence of the increasing steepness of the UBER-vs-RBER relationship with higher levels of ECC (Figure 1). Similar analysis was performed to arrive at CFs for the errors in retention and read-disturb stresses.

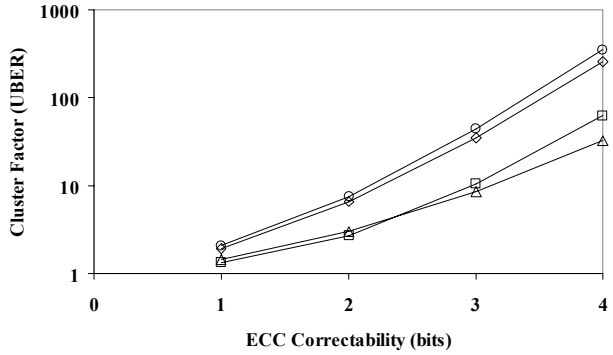

Figure 18: UBER cluster factor vs. ECC correctability, for write errors. Symbols are as defined in the earlier figures.

Although RBER needs to be adjusted by only a factor of about 3 or less, Figure 18 shows that the impact on UBER can be far larger. For this reason, UBER should not be calculated using the binomial Equation (1) with the directly-measured RBER. The RBER must be guardbanded to account for variation. The CF approach sketched here is only one approach. For NAND UBER, modeling of RBER variability can be as important and complex as modeling of the underlying mechanisms.

Figure 19 extends Figure 17 to 4-bit ECC, using the guardbanded-binomial approach. Using datasheet ECC lowers the UBER by more than nine orders of magnitude so that it becomes negligible. Perhaps surprisingly, the dominant mechanism switches from write errors with 1-bit ECC to retention errors with 4-bit ECC, at least for the example of a 1-year retention period. The reason for the dominance of write errors with 1-bit ECC is that the higher RBER after the retention period did not increase UBER by enough to exceed the write errors summed up over all cycles. Increasing ECC greatly magnifies the impact of the higher RBER from retention, because UBER scales as $RBER^{E+1}$. But increasing ECC does not increase the multiplication factor for the integration of write errors.

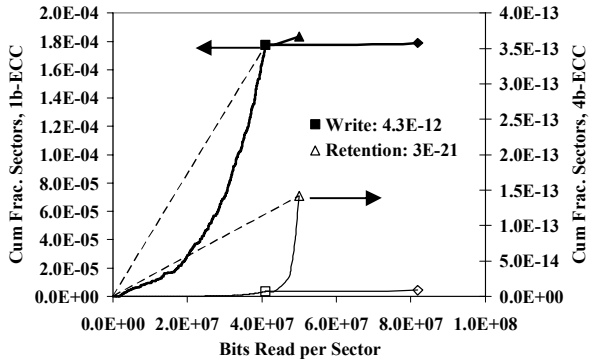

Figure 19: Cumulative fraction of sectors failing vs. bits read for the IMFT device, for both 1-bit and 4-bit ECC. The 1-bit curves are from Figure 17, with the same symbols. The 4-bit curves use binomial estimation from Equation (1) guardbanded for variation. The legend shows the worst-case UBER values (slopes of the dashed lines).

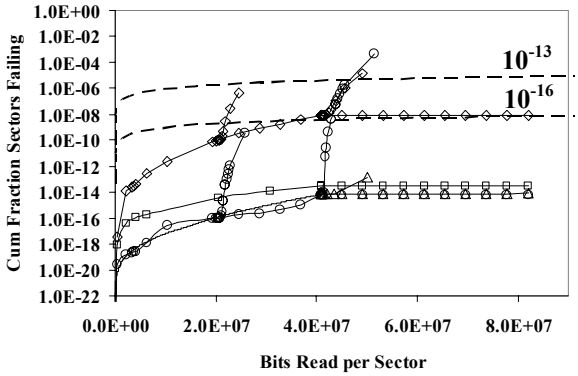

Figure 20: Sectors failing vs. # reads for 4-bit ECC. The dashed lines correspond to constant UBERs of 10^{-13} and 10^{-16}. Symbols are as defined in earlier figures. For each curve, the initial segment to about 4E7 reads is for write errors, the nearly-flat segment to 8E7 reads is for read disturb, and the steeper segments starting at 4E7 are for retention. Extended retention data are unavailable for device . Devices ◊ and O were not specified to 10K cycles, and these have retention curves shown for 5K cycles starting at about 2E7 reads, in addition to the post-10K curves.

How general are these results? Figure 20 shows the results with 4-bit ECC for all four devices. Retention errors dominate the UBER for all devices if the 1-year retention period is included (1-year retention data are unavailable for device). Read disturb is negligible and write errors have an intermediate effect.

Figure 20 shows that NAND devices can have extremely low UBER values. All devices are orders of magnitude below the HDD range of 10^{-13} to 10^{-16} in early life. Only the two devices not specified for 10K-cycle operation reach (and eventually exceed) the HDD range at the end of life. The figure shows that both would be within the HDD range or better with a reduced specification of 5K cycles. On the other hand, one of these devices is specified for 10-year retention, which might not seem warranted by the data in the figure. This should reinforce the point that defining UBER requires defining the usage range for which that UBER is achieved. Conversely, defining a usage range such as a retention lifetime is meaningful only if the UBER is defined.

Because NAND UBER increases with cycle count, retention time, and reads per page, specifying a UBER value requires specifying the limits for those quantities. If those quantities are specified, real applications with less than the absolute maximum-specification usage conditions will have much better UBER than the specified value. HDDs, on the other hand, are generally assumed to have constant UBER, so that all usage conditions will have that level of failure and no better. Another difference between NAND UBER and HDD UBER is that NAND UBER can be determined over the full product lifetime by performing a lifetime's worth of P/E cycles. On the other hand, for HDDs UBER is often measured over a few weeks or months of simulated use [24], and one must assume that UBER does not degrade over the subsequent years

Although we have shown data for 1 year of data retention following 10K cycles, this is best considered a worst-case use condition. If 10K cycles are performed over 5 years, the device is rewritten on average every 4 hours, so a 1-year retention period would require quite exceptional circumstances.

IV. CONCLUSION

This paper has studied bit errors in NAND memories from four manufacturers. The devices have the same error mechanisms but vary in terms of which dominates. The erratic nature of write errors requires that reliability characterizations verify the data after every cycle or use data integration to account for errors that went undetected. Variation in RBER must be guardbanded for in performing binomial calculations of the ECC error rate. We provide workable definitions for uncorrectable bit error rate and estimate its values. ECC causes the failure rates to steepen with respect to P/E cycle count and retention time so that specifying a UBER value is meaningful only if the range of use is specified. Within that range, the UBER can be quite low, orders of magnitude below a typical specification of 10^{-15}.

V. ACKNOWLEDGEMENTS

We would like to thank Chun Fung (Kitter) Man for the every-cycle-verify results, Kevin Wohlschlegel for the high-volume stress infrastructure, and Darshit Vashi for exchanges of data and discussion.

APPENDIX 1: Review of NAND Structure and Operation

NAND structure and operation are briefly reviewed here; see reference [2] for more information.

Figure 21 shows a *string* of Flash memory cells. Each cell is an n-channel MOS transistor with a *floating gate* (FG) between the control gate (CG) and the channel. The floating gate is electrically isolated by an interpoly dielectric (IPD) at the top and a tunnel oxide (TOX) at the bottom. A string typically consists of 32 cells, connected through a select gate at the drain end (SGD) to a bitline and through a select gate at the source end (SGS) to a source diffusion. A memory block consists of many such strings, side-by-side, with common source, separate bitlines, and CGs that are laid out as wordlines perpendicular to the bitlines. The cells on one wordline are divided into two to four *pages*.

Programming the cell means injecting electrons onto the floating gate and is accomplished by applying positive voltage to the control gate which capacitively couples to the FG and induces electrons to tunnel up from the channel. *Erase* means removing electrons by reversing the bias. Adding electrons to the FG raises the V_T of the device, and removing them lowers it.

Current MLC memories store two bits of information per cell by placing cells into four different V_T levels, identified as L0 – L3 in Figure 22. During read, the CG for the selected cell is biased to chosen read points (R1–R3 in Figure 22), and if the cell conducts current then the V_T is determined to be below that value. The CGs for the deselected cells in the string are biased at a voltage V_{PASS} that is higher than the highest programming V_T level so that string current is determined only by the V_T of the selected cell (Figure 13). An entire page is read at once in a single parallel operation (though data are serially clocked out to the external system).

The erase operation places all cells into the lowest level (L0), typically below zero volts. A cell is placed into one of the three positive V_T levels using the programming algorithm shown in Figure 23. A sequence of increasing CG pulses is applied, each raising the V_T. Between pulses, a verify step is performed, like the read operation described above, with the CG set to the minimum V_T acceptable for that level. When the V_T crosses this read level, the verify passes and programming in subsequent CG steps is inhibited [5] using a bias scheme such as that shown in Figure 24. During programming, a cell being programmed has a grounded bitline, grounding the string silicon potential and allowing tunneling of electrons from that ground potential to the positive FG. A cell being inhibited has its bitline biased high enough for the SGD device to be off and the string's silicon potential to be floating. Deselected wordlines are biased to a voltage $V_{INHIBIT}$ which couples positive voltage to the floating string potential, boosting that potential and thereby reducing the voltage across the tunnel oxide and suppressing the tunneling. Variations on this self-boost scheme are often used [6].

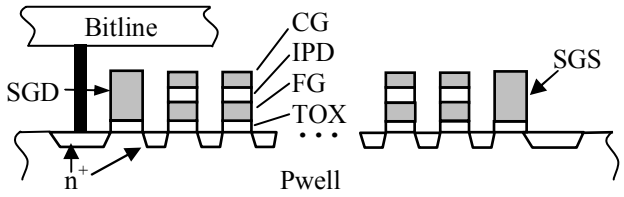

Figure 21: Cross-section of a NAND string

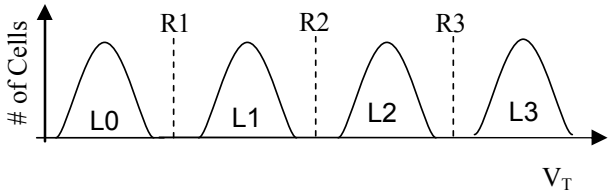

Figure 22: Distribution of cells in four levels for MLC Flash. Bell-shaped curves represent the probability distributions of the thousands of cells in a block placed to each of the four V_T levels.

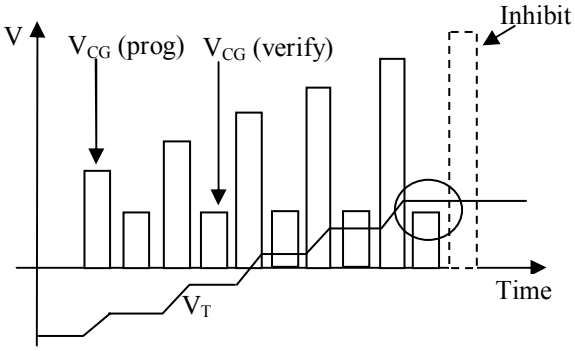

Figure 23: MLC programming placement algorithm. The cell passes at the circled verify step. The final programming pulse (dashed lines) is inhibited.

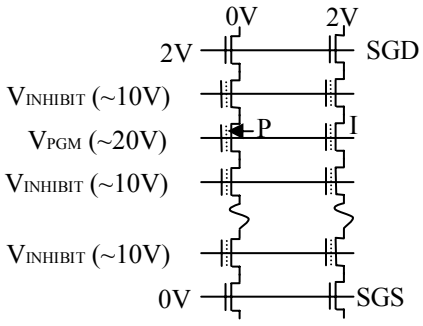

Figure 24: Program inhibit bias. Cell labeled P programs because of the high field induced by the high CG voltage and the string potential at ground due to the grounded bitline. Cell labeled I is inhibited because devices SGD and SGS are off, floating the string silicon potential which is boosted by coupling from the $V_{INHIBIT}$ bias placed on the deselected wordlines.

APPENDIX 2: Worst-Case Assumption for Read Disturb

This section demonstrates that one read between cycles is worst-case for the read-disturb UBER

If read-disturb errors were not erratic from one P/E cycle to another, then the conclusion would follow naturally. In that case, adding a read to an early cycle point could at most generate an error that would have occurred after cycling anyway. Adding a read to the total performed after cycling, on the other hand, would increase the total number of errors because of the power-law relationship between errors and read count (Figure 15). Therefore, to maximize the number of failures reads should be shifted to the end of cycling.

If read disturb errors are erratic with cycling, the conclusion is less obvious because reads at some earlier cycle might generate failures that would not have occurred otherwise. It is not practical to collect read-disturb data after every cycle because the time required would be excessive. It is possible to calculate what the result of such an experiment would be, making the worst-case assumption that all read disturb errors are thoroughly erratic from cycle to cycle so that the failures between different cycle counts are fully additive. As shown in Figure 15, read-disturb RBER increases as a power-law in cycle count and in number of reads. Therefore, the RBER from read disturb can be calculated given the cycle count and subsequent reads. Assuming binomial statistics, the fraction of CWs failing (P_{CW}) follows from (1). With the assumption of fully-erratic errors, the cumulative fraction of codewords failing would be the integral of (1) over cycles, and this can be inserted into the numerator of (4) to obtain the UBER.

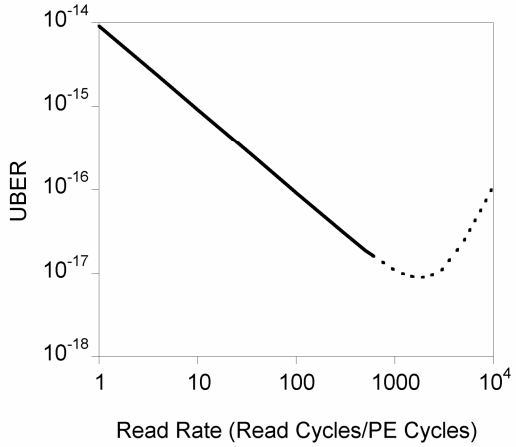

Figure 25: Read-disturb UBER up to 10K program/erase cycles with various reads between each cycle. This curve is estimated from assumed models of cycling degradation and a uniform RBER. The dotted line represents stress points that would require greater than 10 years to achieve in the field.

We considered a hypothetical device with UBER=10^{-14} with 1-bit ECC, when measured with 10K cycles followed by 10K reads, and we simulated what the UBER would have been had the reads between earlier cycles been greater than unity. Figure 25 shows the result. As the number of reads between cycles is increased, the UBER at first drops because a few reads are not enough to generate significant additional errors in the numerator, to offset the increased reads in the denominator. This behavior continues up to the point (dotted section) where performing the stress would take more than 10 years to complete for a NAND device with typical specifications. Only beyond this 10-year limit does read disturb between cycles become large enough for UBER to begin increasing, and even then it does not rise above the value corresponding to a read-write ratio of unity. One can understand why by considering the final point, at 10K reads per cycle. In the UBER Equation (4), adding 10K reads between cycles increases the denominator by a factor of 5000: 10K cycles * 10K reads/cycle, vs. 10K cycles + 10K reads. Therefore, performing 10K reads per cycle would need to increase the fraction of failing CWs by 5000 just for the UBER to stay the same. If every instance of 10K reads caused the same number of CWs to fail, then indeed the number would increase by a factor of 10K, doubling the UBER. But P_{CW} is such a steep function of cycles that the CW error rate is far lower in earlier cycles. As a result, even though there are 10K read-disturb stresses in this extreme case, the number of failing sectors increases by far less than the denominator does, resulting in a lower UBER.

REFERENCES

[1] Y. Itoh, M. Momodomi., R. Shirota, Y. Iwata, R. Nakayama, R.Kirisawa, T. Tanaka, K. Toita, S. Inoue, and F. Masuoka, "An experimental 4 Mb CMOS EEPROM with a NAND structured cell," IEEE ISSCC, pp 134-135, 1989

[2] J. Brewer and M. Gill, eds., Nonvolatile Memory Technologies with Emphasis on Flash: A Comprehensive Guide to Understanding and Using Flash Memory Devices, Wiley-IEEE Press, 2008

[3] M. Bauer, R. Alexis, G. Atwood, B. Baltar, A. Fazio, K. Frary, M. Hensel, M. Ishac, J. Javanifard, M. Landgraf, D. Leak, K. Loe, D. Mills, P. Ruby, R. Rozman, S. Sweha, S. Talreja, K. Wojciechowski, "A multilevel-cell 32 Mb flash memory," IEEE ISSCC, pp 132-133, 1995

[4] S. Lin and D. Costello, Error Control Coding, 2nd Edition, Prentice Hall, 2004

[5] K. Suh, B. Suh, Y. Lim, J. Kim, Y. Choi, Y. Koh, S. Lee, S. Kwon, B. Choi, J. Yum, J. Choi, J. Kim and H. Lim, "A 3.3 V 32 Mb NAND Flash Memory with Incremental Step Pulse Programming Scheme," IEEE J. Sol. St. Circuits 30(11), Nov. 1995, pp 1149-1156

[6] T. Jung, Y. Choi, K. Suh, B. Suh, J. Kim, Y. Lim, Y. Koh, J. Park, K. Lee, "A 3.3V 128Mb Multi-Level NAND Flash Memory for Mass Storage Applications," IEEE ISSCC, pp 32-33, 1996

[7] S. Hur, J. Lee, M. Park, J. Choi, K. Park, K. Kim, and K. Kim, "Effective Program Inhibition Beyond 90nm NAND Flash Memories," Proc. 2004 NVSM, pp 44-45

[8] J. Lee, C. Lee, M. Lee, H. Kim, K. Park, and W. Lee, "A New Program Disturbance Phenomenon in NAND Flash Memory by Source/Drain Hot-Electrons Generated by GIDL Current,"IEEE NVSM 2006, pp 31-33

[9] S. Joo, H. Yang, K.Noh, H. Lee, W. Woo, J. Lee, M. Lee,W. Choi, K. Hwangh, Y. Kim, S. Sim, S. Kim, H. Chang, G. Bae, "Abnormal Disturbance Mechanism of Sub-100 nm NAND Flash Memory," Japanese J. Applied Physics, 45(8A), 2006, pp 6210-6215

[10] H. Kurata, K. Otsuga, A. Kotabe, S. Kajiyama, T. Osabe, Y. Sasago, S. Narumi*, K. Tokami, S. Kamohara, O. Tsuchiya, "The Impact of Random Telegraph Signals on the Scaling of Multilevel Flash Memories," IEEE Symp. VLSI Circuits, 2006, pp 112-113

[11] C. Compagnoni, A. Spinelli, R. Gusmeroli, A. Lacaita, S. Beltrami, A. Ghetti and A. Visconti, "First evidence for injection statistics accuracy limitations in NAND Flash constant-current Fowler-Nordheim programming," IEDM Tech Dig. 2007, pp 165-168

[12] T. Ong, A. Fazio, N. Mielke, S. Pan, N. Righos, G. Atwood, and S. Lai, "Erratic Erase in ETOX™ Flash Memory Array," 1993 VLSI Tech. Symp., pp 83-84

[13] A. Chimenton and P. Olivo, "Erratic Erase in Flash Memories—Part I: Basic Experimental and Statistical Characterization," IEEE Trans. Elect. Dev. 50(4), April 2003, pp 1009-1014

[14] A. Brand, K. Wu, S. Pan and D. Chin, "Novel Read Disturb Failure Mechanism Induced By FLASH Cycling," in Proc. 2003 IRPS, pp 127-132, (1993).
[15] R. Degraeve, F. Schuler, B. Kaczer, M. Lorenzini, D. Wellekens, P. Hendrickx, M. van Duuren, G. Dormans, J. Van Houdt, L. Haspeslagh, G. Groeseneken, G. Tempel, "Analytical percolation model for predicting anomalous charge loss in flash memories," IEEE Trans. Elect. Dev., 51(9), Sept. 2004, pp 1392-1400
[16] H. Belgal, N. Righos, I. Kalastirsky, J. Peterson, R. Shiner, and N. Mielke, "A new reliability model for post-cycling charge retention of flash memories," Proc. 2002 IRPS, pp 7-20
[17] M. Kato; N. Miyamoto; H. Kume; A. Satoh; T. Adachi; M. Ushiyama; K. Kimura, " Read-disturb degradation mechanism due to electron trapping in the tunnel oxide for low-voltage flash memories," 1994 IEDM Tech. Dig., pp 45-48 (1994)
[18] R. Yamada; Y. Mori; Y. Okuyama; J. Yugami; T. Nishimoto; H. Kume, "Analysis of detrap current due to oxide traps to improve flash memory retention," Proc. 2000 IRPS, pp 200-204 (2000)
[19] R. Yamada, T. Sekiguchi, Y. Okuyama, J. Yugami, H. Kume, "A novel analysis method of threshold voltage shift due to detrap in a multi-level flash memory," Tech. Dig. 2001 VLSI Tech. Symp., pp 115-116
[20] J. Lee, J. Choi, D. Park, and K. Kim, "Degradation of Tunnel Oxide by FN Current Stress and Its Effects on Data Retention Characteristics of 90-nm NAND Flash Memory," 2003 IRPS, pp. 497, 2003
[21] N. Mielke, H. Belgal, I. Kalastirsky, P. Kalavade, A. Kurtz, Q. Meng, N. Righos, and J. Wu, "Flash EEPROM Threshold Instabilities due to Charge Trapping During Program/Erase Cycling," IEEE trans. Dev. and Mat. Reliability, vol. 2, No. 3, pp 335-244, 2004
[22] N. Mielke, H. Belgal, A. Fazio, Q. Meng, and N. Righos, "Recovery Effects in the Distributed Cycling of Flash Memories," Proc. 2006, pp 29-35
[23] K. Takeuchi, s. Satoh, T. Tanaka, K. Imamiya, K. Sakui, "A negative Vth cell architecture for highly scalable, excellently noise-immune, and highly reliable NAND flash memories," IEEE J. Sol. St. Circuits, 34(5), pp 675-684, May 1999
[24] J. Gray and C. van Ingen, "Empirical Measurements of Disk Failure Rates and Error Rates," Microsoft Research Technical Report MSR-TR-2005-166, Dec. 2005
[25] IDEMA Standard R2-98, "Specification of Hard Disk Drive Reliability"
[26] JEDEC Standard JESD47E, "Stress-Test-Driven Qualification of Integrated Circuits," JEDEC Solid State Technology Association, January 2007
[27] J. Lee; S. Hur; J. Choi, "Effects of floating-gate interference on NAND flash memory cell operation," IEEE Electron Device Letters 23(5), May 2002, pp 264-266
[28] S. Muramatsu , T. Kubota , N. Nishio , H. Shirai , M. Matsuo , N. Kodama , M. Horikawa , S. Ssaito , K. A rai , and T. O kazawa , " The Solution of Over - Erase Problem Controlling Poly -Si Grain Size - Modifi ed Scaling Principles for Flash Memory , " IEEE IEDM Tech. Dig. , pp. 847 – 850 , 1994
[29] K. Naruke, S. Taguchi and M. Wada, "Stress Induced Leakage Current Limiting To Scale Down EEPROM Tunnel Oxide Thickness," in *Proc. IEDM*, pp. 424-427, (1988).
[30] A. Modelli, F. Gilardoni, D. Ielmini, and A.S. Spinelli, "A New Conduction Mechanism for the Anomalous Cells in Thin Oxide Flash EEPROMs," Proc. IRPS 2001, pp. 61-66