Trends in NAND Flash Memory Error Correction

June 2009

Eric Deal
Cyclic Design

Introduction

NAND Flash is a popular storage media because of its ruggedness, power efficiency, and storage capacity. Flash densities have roughly doubled every year and manufacturers are pushing flash into smaller geometries to further decrease the cost per gigabyte. At the same time, flash devices are moving to store more bits per cell to further increase the storage density.

After providing a brief background on NAND flash, this paper examines the underlying trends in the flash market and the corresponding changes that must be made in the error correction algorithms to maintain the integrity of data stored in next-generation NAND flash devices. Designers must anticipate these changes to ensure that devices designed today remain competitive by taking advantage of the latest flash technology, rather than allowing it to render them obsolete.

NAND Flash Primer

Each page in a current generation NAND flash device consists of 4KB of data plus an additional spare area (usually 128-256 bytes) used for error correction and metadata, additional information used by the file system to track information on flash usage. Next generation NAND flash device pages will likely consist of 8KB of data plus an undetermined amount of spare area. It is important to realize the amount of spare area on a given flash part is fixed by the manufacturer. **For this reason, a NAND flash controller must balance the requirements of metadata and error correction in order to support a wide variety of flash parts that have both varying error correction requirements and spare area sizes.**

SLC versus MLC Flash

NAND flash memory uses an array of floating-gate transistors to store information. In traditional single-level cell (SLC) devices, each bit cell stores only one bit of information. Multi-level cell (MLC) devices can store more than one bit per cell by partially charging the bit cells to encode multiple bit states, as shown in illustration 1.

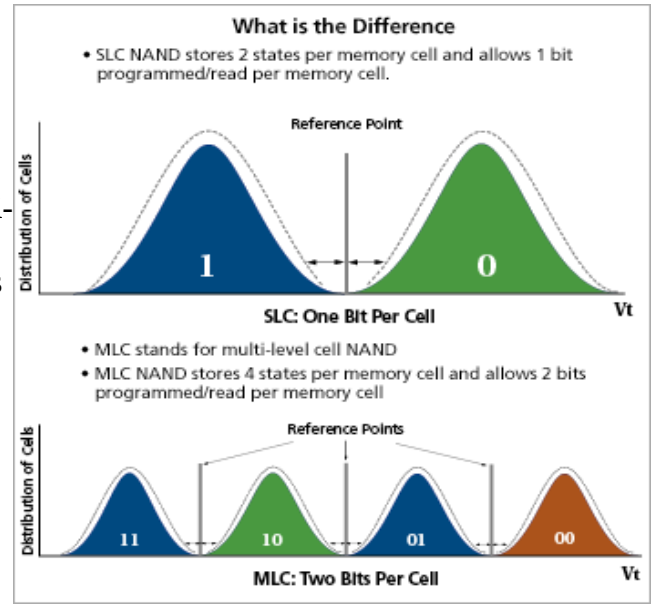


Illustration 1: SLC versus MLC bit cell charge levels (source: http://micron.com/nandcom [1])

As the number of bits stored per cell increases, bit values are represented by smaller voltage ranges, creating more uncertainty in the value stored in the bit cell. This uncertainty increases the likelihood for data to be stored or read incorrectly, requiring higher levels of error correction for MLC flash than for SLC flash. SLC flash has typically required single-bit correction over 512 byte sectors on the flash since the individual bit error rate (BER) is extremely low. MLC flash traditionally has required more powerful correction algorithms capable of correcting four to eight bits to accommodate the higher bit error rates arising from the higher uncertainty of charging and detecting the multiple voltage ranges in a single bit cell.

Trends in Flash Memory

Since the underlying bit cells used to construct both SLC and MLC are similar, SLC flash will always

be more expensive than MLC flash on a per gigabyte basis. Currently, SLC is about twice as expensive as MLC for low-density flash parts (1Gb devices), and the price premium increases by more than 4x for newer high-density parts (4Gb+ devices) since MLC is typically the first to be produced in a new process node. For this reason, MLC NAND flash has become the dominant form of NAND flash and constitutes about 90% of the flash parts shipped. **In order to take advantage of these low-cost, high density parts, NAND controllers must be designed to take advantage of not only flash parts currently on the market, but also those that will be introduced during the device's expected lifetime.**

Next generation flash parts will continue to evolve to:

- Smaller geometries (45nm to 32nm and beyond),
- More bits per cell (2-level MLC to 3-level and 4-level MLC), and
- Larger page sizes (8KB versus 4KB).

As the bit cells get smaller, fewer electrons can be trapped in the floating gates, leading to more uncertainty in the amount of charge present. At the same time, manufacturers are moving past two bit MLC to create designs with three or four bits per cell, further increasing storage density. The effect is to narrow the valid voltage ranges for a given value, again increasing the chances for program and read disturbances to corrupt data.

As flash geometries get smaller, manufacturers are also moving to larger page sizes in order to minimize the wiring overhead required to erase, program, and read the bit cells. Current flash devices typically employ a 4KB page size, but manufacturers are likely to move to an 8KB page size in the next generation of devices that will arrive in 2010.

All of these factors require NAND flash controllers to implement more powerful ECC algorithms, not only for MLC flash, but also for SLC flash. In addition, controllers currently designed around a 4KB page may not be able to handle the addressing and buffering requirements associated with the larger 8K page size, rendering the controller obsolete.

Error Correction

Error correction is an integral part of using NAND flash that ensures data integrity. This section discusses the strategies used for NAND flash error correction codes (ECC) and provides a basis for making decisions on how to best support next generation NAND devices.

Bit and Block Error Rates

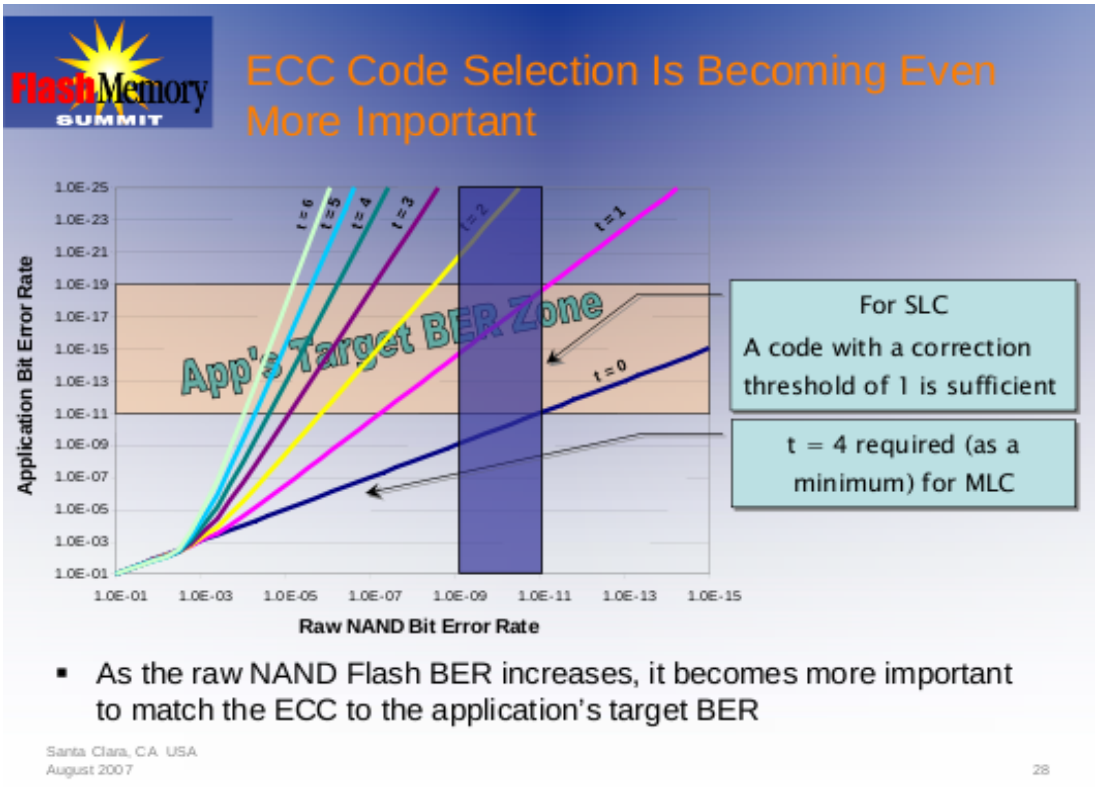
Any discussion on the strengths of error correction algorithms ultimately comes down to the raw bit error rate of the underlying medium and the allowable block error rate that is acceptable to an application. System architects must determine an acceptable level based on the customer's sensitivity to losing a portion of their data. For instance, occasionally losing a block in a digital audio/video player may be acceptable whereas losing a block of data on a solid-state disk may not.

The raw bit error rate (BER) refers to the probability of a bit error occurring in an individual bit cell on a flash device. Bit errors are a natural result of uncertainty when implementing any data storage and must be managed by hardware or software so that the integrity of the underlying information is not compromised. For NAND flash, this is done by protecting groups of bits with a higher-level error correction algorithm. Ideally, all errors in the storage would be corrected by the ECC algorithm. In reality the algorithm protects against a range of errors that are likely to occur. Errors beyond that range

may be unrecoverable. An example of such a failure is a large scratch on a DVD, which may result in a DVD player producing a pause in the video for a movie as it tries to recover after losing bytes from the video stream.

A failure that results in data being lost is called a block error and the rate of these failures is called the block error rate (BLER). When designing a storage application, a designer must decide on an error rate that is acceptable for the application and then provide enough redundancy in the error correction to provide this level of reliability. As mentioned above, different applications may have different levels of acceptable block error rates, and the underlying correction algorithms must be designed to meet these needs.

The following figure comes from a Micron presentation in 2007 at MEMCON [3], showing the relationship between the bit error rate, an application's acceptable block error rate, and the error correction required. (Note that Micron has chosen to work with the corrected bit error rate instead of block error rate on the vertical axis. These cannot be compared directly, but the idea is the same. Micron also uses the variable *t* to denote the error correction strength). The diagram indicates an acceptable application error rate (the brown horizontal box from 1E-11 to 1E-19) and corresponding error correction required for SLC (vertical blue box corresponding to ECC1). The diagram shows that ECC4 would be required for an MLC application, implying an underlying MLC bit error rate of about 1E-7.



Computing Block Error Rate

The application's block error rate can be computed from the bit error rate using the following equation [4]:

$$BLER = \binom{N}{E} * p^{E} * (1-p)^{(N-E)}$$

Where:

N is the number of bits in a block,
E is the number of errors in a block
and p is the probability of a bit error (bit error rate)

This equation basically states that the block error rate is dependent on three factors:

- The number of statistical combinations of failing bit patterns (E combinations of N),
- The probability of E errors occurring (*p* raised to the E power), and
- The probability of N-E correct data bits ((1-*p*) raised to the N-E power).

Since we are only interested in the failing cases, E will be assigned a value that it is just beyond the power of the error correction code under consideration, thus the equation will tell us the probability of encountering our first uncorrectable error. Technically the equation should be a summation of errors greater than the ECC capability, but the subsequent terms tend to be insignificant and can be ignored for simplicity.

To decrease the block error rate, the primary variable a designer has to work with is E, the ECC level. Ideally, a designer could simply keep increasing the correction level to continue to improve the block error rate, but unfortunately, additional correction requires more parity data which are limited by the spare area size. Since flash manufacturers are not necessarily providing more spare area in a flash page, designers must be creative in increasing the error protection while minimizing the generated parity data.

Which Algorithm?

Initial designs implementing SLC NAND used either no error correction or minimally correcting Hamming codes which provide single error correct and double error detect capabilities. Given the low bit error rates of early flash, this was adequate to correct the occasional bit error that occurred. As bit error rates increased with each successive generation of both SLC and MLC flash, designers moved to more complicated cyclic codes such as Reed-Solomon (R/S) or Bose-Chaudhuri-Hocquenghem (BCH) algorithms to increase the correction capability.

While both of the algorithms are similar, R/S codes perform correction over multi-bit symbols while BCH performs correction over single-bit symbols. The selection of the most effective correction code is a tradeoff between the number of symbol errors that need to be corrected and the additional storage requirements for the generated parity data. Typically R/S codes are used when errors are expected to occur in bursts (since correcting a single symbol may correct multiple bit errors) and BCH is used when the bit errors are non-correlated, or randomly distributed. Flash has been shown to have non-correlated bit errors in both manufacturer and independent studies[2], thus BCH is better suited for NAND flash since it allows the coding for higher numbers of bit errors with fewer parity bytes.

Effective NAND Flash ECC for Next Generation Devices

Devices utilizing NAND flash must incorporate very high levels of error correction in order to guarantee support for next generation flash devices. The trend to smaller geometries and more bits per cell will require more correction than today's devices, but manufacturers have not released detailed

design information on page sizes or recommended ECC requirements. To solve these issues, designs must proactively incorporate higher levels of ECC and yet be flexible enough to support configurations using both current and next generation flash parts. Unfortunately, waiting for the specifications is not an option unless time-to-market is not a concern.

Stronger ECC

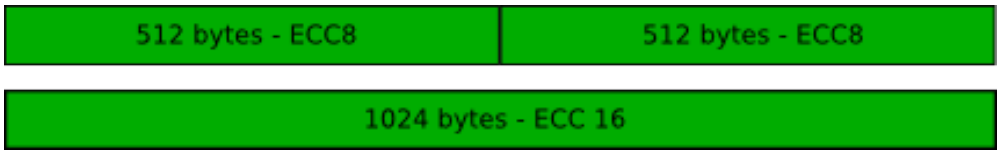
In addition to writing data to the flash, a NAND controller also writes parity data to the flash that enables it to perform error correction. The parity data is generated from a single long polynomial that is typically implemented as a linear-feedback shift register (LFSR). When the controller reads the data back from the flash, it uses the parity information to reconstruct a set of simultaneous equations from which it can determine the locations of the errors within the data set.

The most obvious method of improving the BLER is to increase the ECC level of the underlying correction algorithm (effectively raising E in the BLER equation). Higher levels of ECC are achieved by implementing longer LFSR polynomials that effectively increase the number of simultaneous equations possible and thus make it possible to locate more error locations. The downsides of the longer polynomials are more complexity in the equation solver that computes the locations of the errors as well as more storage required on the flash for parity information. Unfortunately, the current flash page size limits the amount of ECC that can be applied to 8-16 bits of correction when using 512 byte blocks (depending on the flash manufacturer). This limitation can only be overcome by considering correction over other longer block sizes.

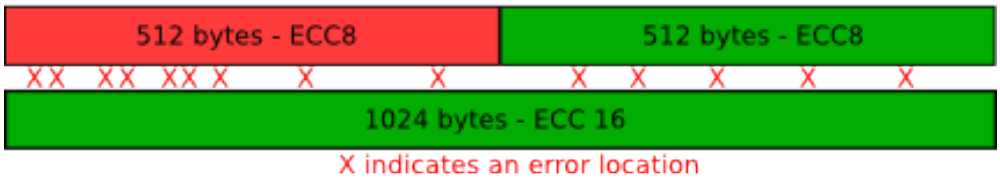
Large Block Error Correcting Codes

The other variable in the BLER computation is the block size over which corrections are performed (N in the BLER equation). Typical controllers correct data in 512 byte blocks which is convenient since it matches the sector size traditionally used in storage applications. There are tradeoffs, however, in selecting larger or smaller block sizes since the block size determines the size of each parity symbol and the ECC level determines the number of parity symbols required. The selection of correction block sizes can lead to some interesting results as shown in the following diagrams.

At first glance, correcting 8 bit errors over 512 bytes for two consecutive blocks appears equivalent to correcting 16 bit errors over the same 1024 byte block since both cases correct sixteen bit errors over each 1KB of data.



In actuality, the two scenarios are not equivalent. Consider the following distribution of 14 errors (9 in the first 512 bytes, five in the second 512 bytes):



In this scenario, the ECC16 correction over 1024 bytes can correct all the errors while the ECC8 solution fails to correct the first block (shown in red). Performing the correction over the longer block provides more protection since it can better handle higher concentrations of errors. The issue then becomes determining the optimal point at which correction over a 1KB block provides more capability than correction over two 512 byte blocks.

Using the BLER equation, it is possible to graph the relative strengths of the various ECC schemes to determine statistically which ones are optimal. The graph below shows the bit error rate on the horizontal axis and the corresponding block error rate on the vertical axis for various ECC schemes over a wide range of bit error rates.

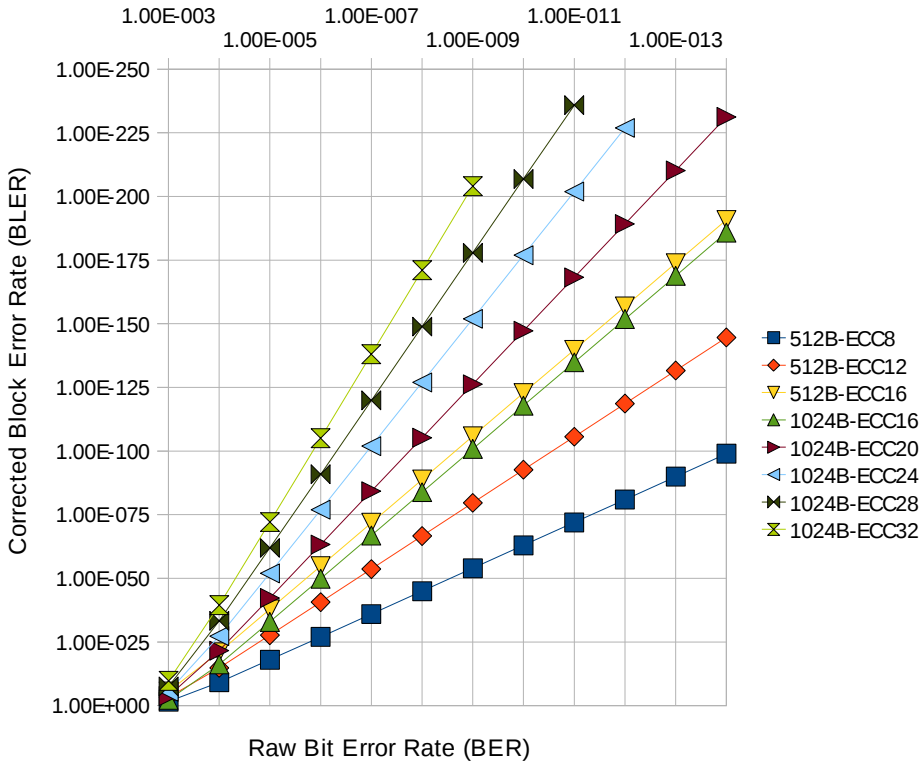


Illustration 2: Overview of Relative ECC Strengths

At low bit error rates (the right side of the graph), the ECC16 over 1KB (dark green) is only slightly less effective than ECC16 over 512 bytes (yellow). This makes sense given the probability even a single bit error is somewhat low.

For MLC flash (and future generations of SLC flash), it is more interesting to focus on the high bit error rate portion of the graph (lower left corner).

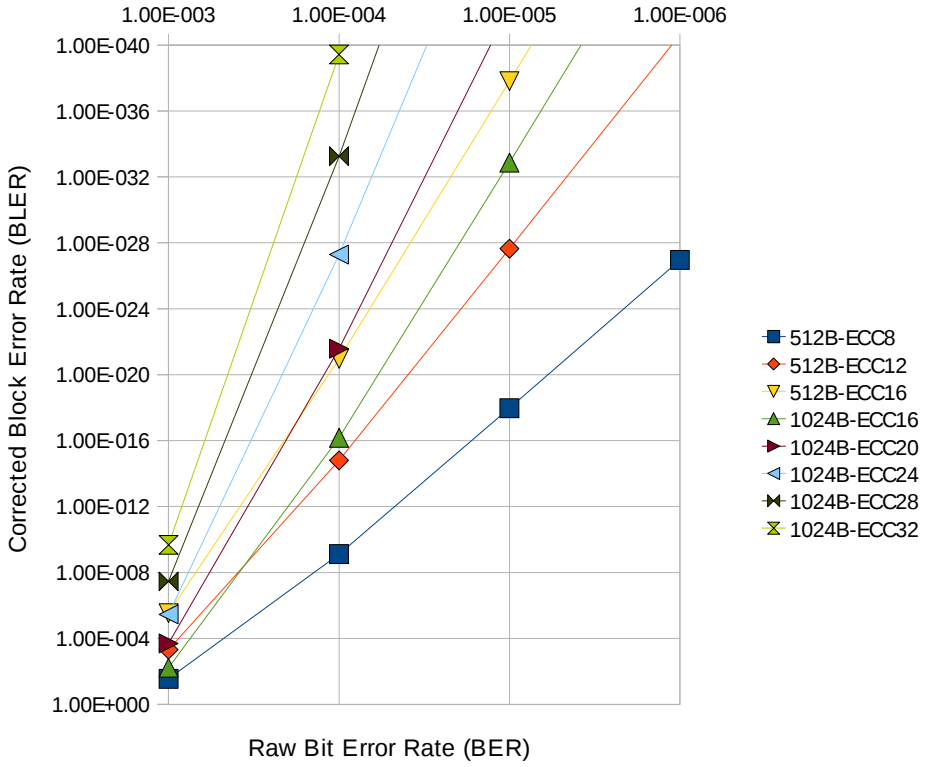


Illustration 3: ECC Strengths at High Bit Error Rates

At these higher bit error rates, it becomes obvious that ECC20 over 1KB blocks (magenta) is roughly equivalent to ECC16 over 512 bytes (yellow), but is far stronger as the bit error rate drops. At a BER of 1E-4, moving to from ECC20 to ECC24 provides roughly 6 orders of magnitude greater protection (1E-28 versus 1E-22) over either of these schemes and using ECC28 increases protection by a further six orders of magnitude.

To achieve the reliability required for most applications with next generation flash, controllers must be upgraded to correct data over 1KB blocks and implement much higher levels of error correction than they currently implement.

Parity Byte Requirements

As previously mentioned, the level of ECC protection is limited by the number of spare bytes available in a flash page. The following table summarizes the total number of bytes required to protect a 4KB page for each of the algorithms:

ECC Algorithm	Parity bytes/block	Parity bytes/page
512B-ECC8	13	104
512B-ECC12	19.5	156
512B-ECC16	26	**208**
1024B-ECC16	28	112
1024B-ECC20	35	140
1024B-ECC24	42	168
1024B-ECC28	49	**196**
1024B-ECC32	56	224

Table 1: Parity bytes required for various ECC algorithms.

From the table, it is interesting to note that supporting ECC28 over 1KB blocks not only provides better correction capabilities, but also consumes fewer overall bytes in the spare area (values in bold in the table: 208 for ECC16 over 512B, 196 for ECC28 over 1024B).

Far better ECC protection can be attained by performing error correction over larger blocks while actually using fewer parity bytes. The current generation of flash parts can still be used with the 1KB block algorithms, and the controller would also be capable of supporting next generation flash devices.

ECC Complexity

Extending an existing controller to support error correction over a larger block size is more complicated than it appears. The underlying correction algorithm maps the data and parity into a mathematical codeword that limits the number of bytes placed into it. Typically the correction algorithm is developed around the desired block size which determines the most efficient codeword and Galois field to be used. For 512 byte blocks, an 8191-bit codeword is optimal and thus a Galois field over GF(2^{13}) is used resulting in 13-bit parity symbols. The selection of the Galois field also determines the underlying polynomials that govern the mathematics behind the correction.

When correcting over a 1KB block size, the GF(2^{13}) is not large enough, so the correction logic must be rebuilt from scratch using a new Galois field GF(2^{14}) and a different set of polynomials. This increases the parity symbol size to 14 bits and changes the construction of the LFSRs, which must expand in length to accommodate the higher ECC requirements. In addition, the equation solver logic must be expanded to support the additional computations, which increases the algorithm run time and computational resources required to implement the correction logic.

Correction logic must also be carefully optimized to balance silicon resources with the performance required to correct data coming from the NAND device. The correction algorithm should be capable of correcting data at the fastest rate at which the NAND can operate, but care must be taken not to over-

design the correction logic and end up with a bloated ECC controller. There are several optimizations that can be made to a design to minimize area while still enabling it to perform corrections at a specified correction rate.

It is important to note that the time involved in developing these new algorithms is significant, especially for an engineer who has never been exposed to the math behind the correction algorithms. Additionally, debugging and verifying an ECC design can be difficult given the complexity of the math and algorithms involved in computing the error locations.

Conclusion

NAND Flash devices require sophisticated error correction algorithms to minimize errors that occur during the programming and read operations. Next generation flash devices will move to smaller geometries and increased number of bits per cell, features that will increase the underlying bit error rate. To accommodate these trends, designers must make tradeoffs in the error correction to balance correction capability with the amount of space available in a flash page for ECC parity bytes. Moving to larger ECC correction blocks provides much more powerful correction capabilities while consuming a similar number of bytes for parity to that in a smaller block.

Next generation devices will also likely move to 8KB page sizes and will likely require error correction over 1KB data blocks in order to minimize the number of spare bytes in a flash page. Controllers need to be updated to handle this larger page size and must be flexible enough to accommodate the vagaries in the ECC requirements and spare area size of next generation flash parts.

As next generation flash comes to market in 2010, companies risk obsolete products or long delays in product development schedules unless they are prepared for these parts. The data presented shows the strategies required to evaluate NAND flash error correction codes and provides a basis for making decisions on how to best support next generation NAND devices.

About Cyclic Design

Cyclic Design supplies BCH Error Correction IP for NAND flash applications. Visit http://cyclicdesign.com for more information about Cyclic Design's portfolio of error correction IP for semiconductor applications.

References:

[1] http://www.micron.com/nandcom/

[2] Eitan Yaakobi, Jing Ma, Adrian Caulfield, Laura Grupp, Steven Swanson, Paul H. Siegel, and Jack K. Wolf, University of California, San Diego (Winter 2009) “ERROR CORRECTION CODING FOR FLASH MEMORIES”.

[3] Jim Cooke (August 2007) “The Inconvenient Truths about NAND Flash Memory” Micron MEMCON '07 presentation.

[4] http://www.fcet.staffs.ac.uk/alg1/2004_5/Semester_1/Communications,%20COMMS%20(CE00038-2)/word%20notes/Block%20Error%20Rate.doc