What Types of ECC Should Be Used on Flash Memory?

AN99200 discusses the possible ECC schemes and algorithms that can be implemented in systems using SkyHigh SLC NAND flash products.

1 ECC Usage with NAND Flash

SkyHigh has introduced SLC NAND flash products with endurance features comparable with or better than other existing solutions in the market. The SkyHigh NAND flash product offers up to 100,000 Write/Erase cycles with1-bit ECC for 4x nm products and 4 bits ECC for 3x nm products.

It is important to note that since the SkyHigh NAND flash supports the Copy back function, some important considerations have to be taken into account by the host in order to avoid any possible accumulation of single bit errors. The host should make sure to implement either of the following scenarios:

- Readout of data to compute ECC (and modify data if needed) before writing it back. The data from the source page can be read out by the host in order to compute error detection. Before copying back data to an alternate device page, the host may perform data correction if needed. Since the data to be written is still in the page register, the host is only required to upload the corrected bytes using the “Change Write Column”/ ”Random Data Input” command, making the sequence faster.
- Implement an ECC scheme that exceeds the minimum required ECC.

On the other hand, in order to ensure that the data is stored properly over the life of the NAND flash device, it is highly recommended that the following additional precautions be taken:

- Always check status after executing Write, Erase and Copyback operations.
- Implement bad-block management, garbage collection and wear-leveling algorithms.

2 Most Commonly Used ECC Algorithms

This section provides details of the three most commonly used ECC algorithms.

- Hamming Algorithm
- Reed-Solomon Algorithm
- Bose-Chaudhuri-Hocquenghem (BCH) algorithm

2.1 The Hamming Algorithm

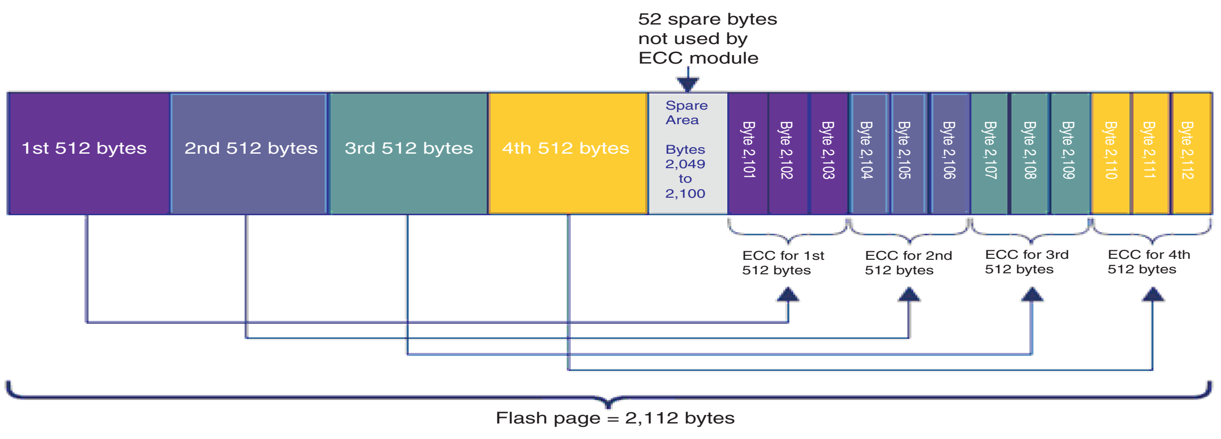
The Hamming algorithm is relatively straightforward and easy to be implemented in software or hardware. The limitation of Hamming algorithm is its limited error correction abilities. Hamming code is able to correct single bit errors and detect two bits errors. For NAND flash, the Hamming algorithm for ECC computation can be used in the case where the host system performs a Read for Copy Back function and makes all necessary bit corrections prior to writing it back, as explained in the previous section. A Hamming code is usually defined as (2^n-1, 2^n-n-1), where:

- n = the number of overhead bits
- 2^n-1 = the block size
- 2^n-n-1 = the number of data bits in the block

All Hamming codes can detect two errors and correct one error. Common Hamming code sizes are (7, 4), (15, 11), and (31, 26). Meaning in a 7-bit block only 4 bits are data, the other 3 bits are correction code; the same goes for (15,11) and (31,26).

For example, a NAND flash with 2 kB pages that uses a Hamming code algorithm may look like Figure 1.

Figure 1. NAND Storage of Hamming Code



In this configuration the Hamming code requires 3 bytes of ECC information for each 512-bytes sector, or 12 bytes (96 bits) of ECC encoding information is required per 2 kB page.

2.2 The Reed-Solomon Algorithm

The Reed-Solomon algorithm is often used on outer encoding while convolutional code is used on inner encoding (The inner code takes the result of the pre-coding operation and generates a sequence of encoding symbols. Each encoding symbol is the XOR of a randomly chosen set of symbols from the pre-code output). The convolutional code allows the correction of widely scattered errors but is not able to correct highly concentrated errors. The Reed-Solomon algorithm is often used in NAND flash memory interfaces. Reed-Solomon codes are often used to handle NAND flash bit-flipping phenomenon. The Reed-Solomon Algorithm is therefore widely used for NAND and in other mass data storage devices. The Reed-Solomon encoder uses Galois Field arithmetic operations to add parity symbols. Parameters are listed below:

- n = the number of code symbols
- s = gives the size of symbols (s-bit symbols). $n=2^s-1$
- t = number of correctable errors, 2*t is the number of parity check symbols
- k = number of message symbols (k=n-2t)

A Reed-Solomon code is specified as RS(n,k) with S-bit symbols.

n= k+2t = 2^s-1	
Data (k)	Parity (2t)

The decode process takes several stages to get error location and correct the error. The first decoding stage is syndrome computation; this stage transfers symbols to data syndrome. The decoder tells if errors are detected at this stage. After that, the algorithm computes error polynomial based on syndromes in the finite Galois field. Following stages find the roots of error polynomial to locate errors and correct them.

Example: A popular Reed-Solomon code is RS (255, 223) with 8-bit symbols. Each code word contains 255 code word bytes, of which 223 bytes are data and 32 bytes are parity. For this code:

n=255, k =223, s=8

2t=32, t=16

The decoder can correct any 16 symbol errors in the code word: i.e. errors up to 16 bits anywhere in the code word can be automatically corrected.

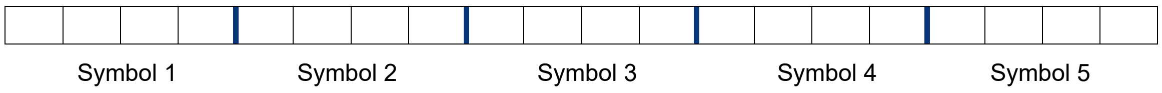
2.3 The BCH Algorithm

Hamming code provides the easiest hardware or software implementation; but it only corrects single bit errors. Reed-Solomon algorithm provides more robust error correction ability; but requires a large amount of system resources (CPU cycles or logic cells) to implement. Bose-Chaudhuri-Hocquenghem (BCH) algorithm is becoming popular because of its improved efficiency over Reed-Solomon algorithm. BCH code is a large class of multiple errors correcting codes. One advantage of BCH is that both highly concentrated and widely scattered errors are detected. Another advantage is that the encoding and decoding techniques are relatively simple compared to Reed-Solomon code. BCH codes belong to the class of linear block codes, to be more specific, the subclass of cyclic codes.

A block code consists of a set of vectors with N elements, where the vectors are called code word and N is called the length of the code word; q symbols are the elements of a code word. If the elements consist of the two symbols 1 and 0, the code is a binary code. A block code maps k information bit into a code word with length of N, and the ratio r = k/N is defined to be the rate of the code. As stated before, the elements of a code word are selected from an alphabet of q symbols. Codes are constructed from fields with q elements. In coding, q is usually a finite number, so the field is a finite field or a so called Galois field.

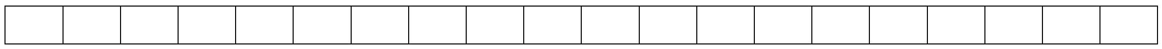
The main difference between Reed-Solomon and binary BCH is the underlying structures. Reed-Solomon algorithm is symbol-based and BCH algorithm is binary.

Reed-Solomon algorithm (Symbol base code)



Symbol length = 4; code length = 5 symbols

BCH algorithm (binary base code)



Symbol length = 1; code length = 20 symbols

3 Conclusion

Hamming based block codes are the most commonly used ECC for SLC NAND. Hamming codes are relatively straightforward and simple to be implemented in either software or hardware. The disadvantage of Hamming codes is its limited error correction capabilities, with two bit errors detection and only one bit error correction. Therefore, it is mainly used in SLC NAND flash applications.

Reed-Solomon and BCH are able to handle multiple errors. Both codes are powerful and able to handle both random and burst errors. Reed-Solomon code is a subset of the BCH. However, BCH is simpler than Reed-Solomon to decode and implement. On the other hand, Reed-Solomon code is more suitable for concatenated codes.

Document History Page

Document Title: AN99200 - What Types of ECC Should Be Used on Flash Memory? Document Number: 001-99200				
Rev.	**ECN No.**	**Orig. of Change**	**Submission Date**	**Description of Change**
**	–	–	11/27/2007	Initial version
*A	–	–	03/20/2011	Updated with references to MS NAND flash
*B	–	–	08/30/2011	Updated to refer to NAND products in general
*C	4978573	MSWI	10/21/2015	Updated in Cypress template
*D	5219601	MNAD	08/05/2016	Updated sections 2.2 and 2.3 Updated template
*E	5807182	AESATMP8	07/10/2017	Updated logo and Copyright.
*F	6295927	MNAD	08/30/2018	Updated template
*G	6311826	YOQI	09/17/2018	Fixed alignment for labels in Section 2.3: “The BCH Algorithm,” on page 3
*H		MNAD	05/29/2019	Updated to SkyHigh format