What Types of ECC Should Be Used on Flash Memory?

Application Note

by Scott Chen

1. Abstract

NOR Flash normally does not need ECC (Error-Correcting Code). On the other hand, NAND requires ECC to ensure data integrity. NAND Flash includes extra storage on each page to store ECC code as well as other information for wear-leveling, logical to physical block mapping, and other software overhead functions. The size of extra storage (spare area) is normally 16 byte per 512 byte sector but other sizes are also used. ECC algorithm correction strength (number of bit errors that can be corrected) depends on the ECC algorithm used to correct the errors (these algorithms may be implemented in either hardware or software). Simple Hamming codes can only correct single bit errors. Reed-Solomon code can correct more errors and is used on many of the current controllers. BCH (Bose, Ray-Chaudhuri, Hocquenghem) codes can also correct multiple bit errors and are becoming popular because of their improved efficiency over Reed-Solomon.

1.1 The Hamming Algorithm

There are two different memory cell technologies used in NAND Flash devices. The first one is the traditional implementation, where each memory cell stores a single bit of data. This approach is categorized as single level cell (SLC). The second approach uses multi level cell (MLC) to store two bits of data in one cell. The data integrity in MLC requires a more sophisticated error correction than SLC NAND Flash devices.

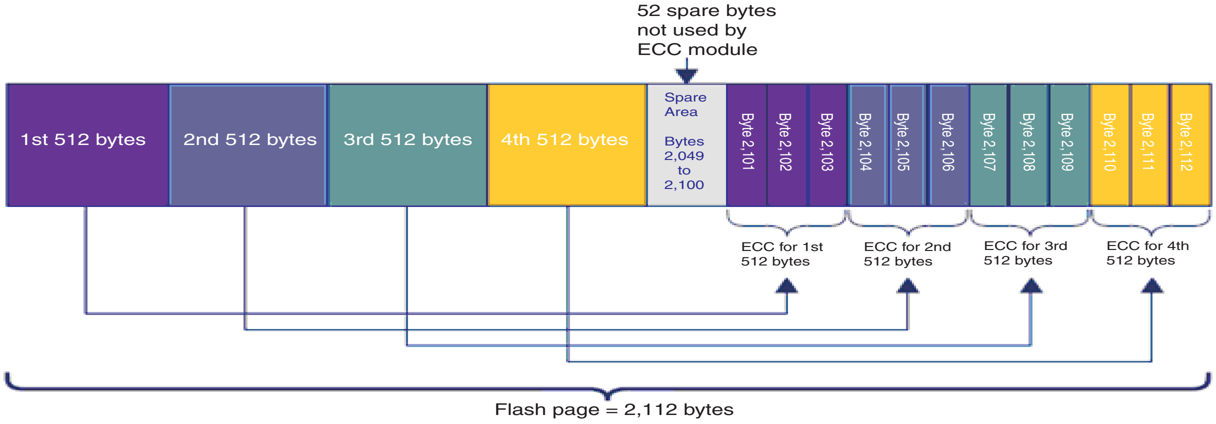
The Hamming algorithm is relatively straightforward and easy to be implemented in software or hardware. The limitation of Hamming algorithm is its limited error correction abilities. Hamming code is able to correct single bit errors and detect two bits errors. For example, many SLC, small page and large page NAND Flash can detect 2 bit errors and correct 1 bit error per 256 or 512 bytes. 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 2KB pages that uses a Hamming code algorithm may look like Figure 1.1.

Figure 1.1 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 2KB page.

1.2 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 concentration errors. Reed-Solomon algorithm is often used in NAND Flash memory interfaces. Reed-Solomon codes are often used to handle NAND Flash bit-flipping phenomenon. All NAND vendors recommend using Error Correcting Codes algorithms, especially Reed-Solomon. 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.

<table>
<tr><th colspan="2">n= k+2t = 2^s-1</th></tr>
<tr><td>Data (k)</td><td>Parity (2t)</td></tr>
</table>

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.

Currently, there are only a few micro-controllers that implement Reed-Solomon encoding, most of which are targeted to the USB or memory card applications, using a 4-byte-on-512-byte-page encoding specifically for MLC (Multi-Level Cell) NAND Flash devices.

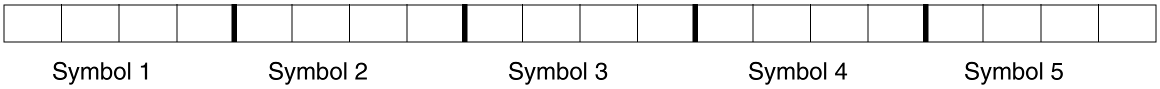
1.3 The BCH Algorithm

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.

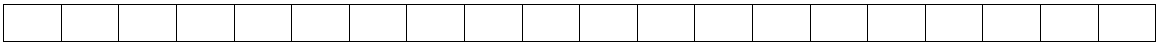
The main difference between Reed-Solomon and binary BCH is the underlying structures. Reed-Solomon algorithm is symbol-based and BCH algorithm is binary.

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

Both Reed-Solomon algorithm and BCH algorithm are common ECC choices for MLC NAND Flash. Micro-controllers specially designed for SD Cards, SPI, eMMC and embedded NAND are becoming more common that use built-in hardware 6/9/12-bit BCH ECC circuits. Moreover, some MLC NAND Flash devices have an internal BCH ECC (Error Correction Code) generation engine, which can speed up data integrity checking. Extra command functions concerning embedded ECC are also provided in software drivers.

2. Conclusion

Hamming based block codes are the most commonly used ECC for SLC. 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 on SLC NAND flash application.

Both Reed-Solomon and BCH are able to handle multiple errors and are widely used on MLC flash. 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.

3. Revision History

Section	Description
Revision 01 (November 27, 2007)	
	Initial release

Colophon

The products described in this document are designed, developed and manufactured as contemplated for general use, including without limitation, ordinary industrial use, general office use, personal use, and household use, but are not designed, developed and manufactured as contemplated (1) for any use that includes fatal risks or dangers that, unless extremely high safety is secured, could have a serious effect to the public, and could lead directly to death, personal injury, severe physical damage or other loss (i.e., nuclear reaction control in nuclear facility, aircraft flight control, air traffic control, mass transport control, medical life support system, missile launch control in weapon system), or (2) for any use where chance of failure is intolerable (i.e., submersible repeater and artificial satellite). Please note that Spansion will not be liable to you and/or any third party for any claims or damages arising in connection with above-mentioned uses of the products. Any semiconductor devices have an inherent chance of failure. You must protect against injury, damage or loss from such failures by incorporating safety design measures into your facility and equipment such as redundancy, fire protection, and prevention of over-current levels and other abnormal operating conditions. If any products described in this document represent goods or technologies subject to certain restrictions on export under the Foreign Exchange and Foreign Trade Law of Japan, the US Export Administration Regulations or the applicable laws of any other country, the prior authorization by the respective government entity will be required for export of those products.

Trademarks and Notice

The contents of this document are subject to change without notice. This document may contain information on a Spansion product under development by Spansion. Spansion reserves the right to change or discontinue work on any product without notice. The information in this document is provided as is without warranty or guarantee of any kind as to its accuracy, completeness, operability, fitness for particular purpose, merchantability, non-infringement of third-party rights, or any other warranty, express, implied, or statutory. Spansion assumes no liability for any damages of any kind arising out of the use of the information in this document.

Copyright © 2007 Spansion Inc. All rights reserved. Spansion®, the Spansion Logo, MirrorBit®, MirrorBit® Eclipse™, ORNAND™, HD-SIM™ and combinations thereof, are trademarks of Spansion LLC in the US and other countries. Other names used are for informational purposes only and may be trademarks of their respective owners.