A Novel Approach to Design an Error Detection and Data Recovery Architecture in MLC NAND Flash Memories

M.V. AJAY KUMAR¹
PG Scholar, Guntur Engineering College, Guntur, A.P-INDIA,
Email:mvajykumar@yahoo.com.

KURAGANTI SYAM BABU²
Assoc Prof, Guntur Engineering College, Guntur, A.P-INDIA,
Email:kuraganti.vemu@gmail.com.

G. SREENIVAS RAO³
Asst Prof, Guntur Engineering College, Guntur, AP-INDIA,
Email:sree.cnu2012@gmail.com.

Abstract: Error detection and correction or error control is techniques that enable reliable delivery of digital data over unreliable communication channels or storage medium. Error detection is the detection of errors caused by noise or other impairments during transmission from the transmitter to the receiver. Error correction is the detection of errors and reconstruction of the original, error-free data. Error control coding (ECC) is essential for correcting soft errors in Flash memories. In this paper we propose use of product code based schemes to support higher error correction capability. Specifically, we propose product codes which use Reed-Solomon (RS) codes along rows and Hamming codes along columns and have reduced hardware overhead. We propose a flexible product code based ECC scheme that migrates to a stronger ECC scheme when the numbers of errors due to increased program/erase cycles increases by maintaining the latency and memory under control.

Keywords: Hamming Codes, Reed-Solomon codes Error correction codes (ECCs), flash memories.

I. INTRODUCTION

A. Definition and memory types

Electronic space provided by silicon chips (semiconductor memory chips) or magnetic/optical media as temporary or permanent storage for data and/or instructions to control a computer or execute one or more programs. Two main types of computer memory are: (1) Read only memory (ROM), smaller part of a computer's silicon (solid state) memory that is fixed in size and permanently stores manufacturer's instructions to run the computer when it is switched on. (2) Random access memory (RAM), larger part of a computer's memory comprising of hard disk, CD, DVD, floppies etc., (together called secondary storage) and employed in running programs and in archiving of data. Memory chips provide access to stored data or instructions that is hundreds of times faster than that provided by secondary storage.

Memory errors: Memory errors are of two types, namely hard and soft.

- Hard errors are caused due to fabrication defects in the memory chip and cannot be corrected once they start appearing.
- Soft errors on the other hand are caused predominantly by electrical disturbances.

B. Error Control Coding

The goal of error control coding is to encode information in such a way that even if the channel (or storage medium) introduces errors, the receiver can correct the errors and recover the original transmitted information.

i) Parity Checking:

One simple way to detect errors is:

1. Count the number of ones in the binary message.
2. Append one more bit, called the parity bit, to the message.
3. Set the parity bit to either 0 or 1, so that the number of ones in the result is even. For example, if the original message contained 17 ones, the parity bit would be a one; if there had been 16 ones, the parity bit would be a zero.
4. Count the number of ones in the received message, including the parity bit. The result will always be even if no errors were encountered. (This approach also works if the parity bit is set to make the count come out odd, as long as the receiver checks for an odd count).

This simple check does have two limitations: it only detects errors, without being able to correct them; and it can’t detect errors that invert an even number of bits.

ii) Hamming Codes

Hamming codes are an extension of this simple method that can used to detect and correct a larger set of errors [14]. Hamming’s development [Ham][12],[13] is a very
direct construction of a code that permits correcting single-bit errors \[7\]. He assumes that the data to be transmitted consists of a certain number of information bits \( u \), and he adds to these a number of check bits \( p \) such that if a block is received that has at most one bit in error, then \( p \) identifies the bit that is in error (which may be one of the check bits). Specifically, in Hamming's code \( p \) is interpreted as an integer which is \( 0 \) if no error occurred, and otherwise is the \( 1 \)-origined index of the bit that is in error. Let \( k \) be the number of information bits, and \( m \) the number of check bits used. Because the \( m \) check bits must check themselves as well as the information bits, the value of \( p \), interpreted as an integer, must range from \( 0 \) to which are distinct values. Because \( m \) bits can distinguish cases, we must have

\[ 2^m \geq m + k + 1 \]  

\[ (1) \]

ii) Reed-Solomon codes

Reed-Solomon codes were developed in 1960 by Irving S. Reed and G. Stavne Solomon, who were then members of MIT Lincoln Laboratory. Their seminal article was entitled "Polynomial Codes Over Certain Finite Fields." (Reed & Solomon 1960) When the article was written, an efficient decoding algorithm was not known \[17,18\]. A solution for the latter was found in 1969 by Elwyn Berlekamp and James Massey, and is since known as the Berlekamp-Massey decoding algorithm. In 1977, RS codes were notably implemented in the Voyager program in the form of concatenated codes. The first commercial application in mass-produced consumer products appeared in 1982 with the compact disc, where two interleaved RS codes are used.

The parameters of a Reed-Solomon code are:

- \( m \) = the number of bits per symbol
- \( n \) = the block length in symbols
- \( k \) = the uncoded message length in symbols
- \( n-k \) = the parity check symbols (check bytes)
- \( t \) = the number of correctable symbol errors
- \( n-k=2t \) (for \( n-k \) even)
- \( n-k-1 = 2t \) (for \( n-k \) odd)

Therefore, an RS code may be described as an \( (n,k) \) code for any RS code where, \( n \leq 2^m -1 \), and \( n-k \geq 2t \).

II. ECCS WITH FAULT SECURE DETECTOR

2.1 Error-Correcting Code Reviews:

Let \( c = (c_0,c_1,\ldots,c_k) \) be the \( k \)-bit information vector that will be encoded into an \( n \)-bit codeword, \( e = (e_0,e_1,\ldots,e_n) \). For linear codes, the encoding operation essentially performs the following vector-matrix multiplication:

\[ c = iG \]  

Where \( G \) is a \( k \times n \) generator matrix. The validity of a received encoded vector can be checked with the Parity-Check matrix, which is a \( (n-k) \times n \) binary matrix named \( H \).

The checking or detecting operation is basically summarized as the following vector-matrix multiplication:

\[ s = cH^T \]  

The minimum distance of an ECC, \( d \), is the minimum number of code bits that are different between any two code words. The maximum number of errors that an ECC can detect is \( d-1 \), and the maximum number that it corrects is \( d/2 \). Any ECC is represented with a triple \( (n,k,d) \), representing code length, information bit length, and minimum distance, respectively.

B. FSD-ECC Definition

The restricted ECC definition which guarantees a FSD-ECC is as follows.

Definition I: Let be an ECC with minimum distance is FSD-ECC if it can detect any combination of overall or fewer errors in the received codeword and in the detector circuitry.

Theorem: Let \( c \) be an ECC, with minimum distance \( d \). \( c \) is FSD-ECC if any error vector of weight \( 0 < e < d -1 \), has syndrome vector of weight at least \( d - e \).

Note: The following proof depends on the fact that any single error in the detector circuitry can corrupt at most one output (one syndrome bit). This can be easily satisfied for any circuit by implementing the circuit in such a way that no logic element is shared among multiple output bits; therefore, any single error in the circuit corrupts at most one output (one syndrome bit).

C. Euclidean Geometry Code Review

The construction of Euclidean Geometry codes based on the lines and points of the corresponding finite geometries[27]. Euclidean Geometry codes are also called
EG-LDPC codes based on the fact that they are low-density parity-check (LDPC) codes [14]. LDPC codes have a limited number of 1’s in each row and column of the matrix; this limit guarantees limited complexity in their associated detectors and correctors making them fast and light weight [9].

Let EG be a Euclidean Geometry with \( n \) points and \( J \) lines. EG is a finite geometry that is shown to have the following fundamental structural properties:
1) Every line consists of \( \rho \) points;
2) Any two points are connected by exactly one line;
3) Every point is intersected by \( \gamma \) lines;

D. Efficiency of EG-LDPC

It is important to compare the rate of the EG-LDPC code with other codes to understand if the interesting properties of low-density and FSD-ECC come at the expense of lower code rates. We compare the code rates of the EG-LDPC codes that we use here with an achievable code rate upper bound (Gilbert-Varshamov bound) and a lower bound (Hamming bound). Table I shows the upper and lower bounds on the code overhead, for each of the used EG-LDPC. The EG-LDPC codes are no larger than the achievable Gilbert bound for the same \( k \) and \( d \) value and they are not much larger than the Hamming bounds. Consequently, we see that we achieve the FSD property without sacrificing code compactness.

III. FAULT-TOLERANT MEMORY SYSTEM OVERVIEW

We outline our memory system design that can tolerate errors in any part of the system, including the storage unit and encoder and corrector circuits using the fault-secure detector. For a particular ECC used for memory protection, let \( E \) be the maximum number of error bits that the code can correct and \( \rho \) be the maximum number of error bits that it can detect, and in one error combination that strikes the system, let \( e_c, e_m, \) and \( e_c \) be the number of errors in encoder, a memory word, and corrector, and let \( e_{de} \) and \( e_{dc} \) be the number of errors in the two separate detectors monitoring the encoder and corrector units.

1) Any single error in the encoder or corrector circuitry can at most corrupt a single codeword bit (i.e., no single error can propagate to multiple codeword bits);

2) There is a fault secure detector that can detect any combination of errors in the received codeword along with errors in the detector circuit. This fault-secure detector can verify the correctness of the encoder and corrector operation.

The first property is easily satisfied by preventing logic sharing between the circuits producing each codeword bit or information bit in the encoder and the corrector respectively. We define the requirements for a code to satisfy the second property. An overview of our proposed reliable memory system is shown in Fig. 1 and is described in the following. The information bits are fed into the encoder to encode the information vector, and the fault secure detector of the encoder verifies the validity of the encoded vector. If the detector detects any error, the encoding operation must be redone to generate the correct codeword. The codeword is then stored in the memory. During memory access operation, the stored code words will be accessed from the memory unit. Code words are susceptible to transient faults while they are stored in the memory.

Therefore a corrector unit is designed to correct potential errors in the retrieved code words. In our design (see Fig. 1) all the memory words pass through the corrector and any potential error in the memory words will be corrected. Similar to the encoder unit, a fault-secure detector monitors the operation of the corrector unit. All the units shown in Fig. 1 are implemented in fault-prone the only component which must be implemented in reliable circuitry are two OR gates that accumulate the syndrome bits for the detectors shown in Fig. 4.

![Fig1: Fault-tolerant memory architecture, with pipelined corrector.](image)

**Detector, Encoder, and Corrector Circuit Area in the Number of 2-Input Gates**

<table>
<thead>
<tr>
<th>Code</th>
<th>(15,7,5)</th>
<th>(63,37,9)</th>
<th>(255,175,17)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Detector</td>
<td>54</td>
<td>575</td>
<td>3976</td>
</tr>
</tbody>
</table>

TABLE-II
A. ENCODER

An bit codeword $c$, which encodes a $k$-bit information vector $i$ is generated by multiplying the $k$-bit information vector with a $k \times n$ bit generator matrix $G$; i.e., $c = i \cdot G$. EG-LDPC codes are not systematic and the information bits must be decoded from the encoded vector, which is not desirable for our fault-tolerant approach due to the further complication and delay that it adds to the operation. However, these codes are cyclic codes [1]. We used the procedure presented in [1] and [4] to convert the cyclic generator matrices to systematic generator matrices for all the EG-LDPC codes under consideration.

Fig. 2 shows the systematic generator matrix to generate the $(15, 7, 5)$ EG-LDPC code. The encoded vector consists of information bits followed by parity bits, where each parity bit is simply an inner product of information vector and a column of $X$, from $G = [I : X]$.

![Generator matrix for the (15, 7, 5) EG-LDPC in systematic format.](image)

Note: The identity matrix in the left columns.

Fig. 3 shows the encoder circuit to compute the parity bits of the $(15, 7, 5)$ EG-LDPC code. In this figure $i = (i_0, i_1, i_2, ..., i_6)$ is the information vector and will be copied to $(c_0, ..., c_8)$ bits of the encoded vector, $c$, and the rest of encoded vector, the parity bits, are linear sums (XOR) of the information bits. If the building block is two-input gates then the encoder circuitry takes 22 two-input XOR gates. Table II shows the area of the encoder circuits for each EG-LDPC codes under consideration based on their generator matrices. Once the XOR functions are known, the encoder structure is very similar to the detector structure shown in Fig. 3.

IV. RESULTS

Fig. 4. Waveform of Nand Flash Memory Design.

Fig. 5. RTL Schematic Design.
DEVICE UTILIZATION SUMMARY

<table>
<thead>
<tr>
<th>Device Utilization Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic Utilization</td>
</tr>
<tr>
<td>Number of Slice Flip Flops</td>
</tr>
<tr>
<td>Number of 4 input LUTs</td>
</tr>
</tbody>
</table>

| Logic Distribution          |       |           |            |         |
| Number of occupied Slices   | 640   | 1,920     | 33%        |         |
| Number of Slices containing only related logic | 640 | 640 | 100% | |
| Number of Slices containing unrelated logic | 0 | 640 | 0% | |

| Total Number 4 input LUTs   | 956   | 3,840     | 24%        |         |

| Number used as logic        | 920   |           |            |         |
| Number used as a route-thru | 36    |           |            |         |

V. CONCLUSION

Simulation results show that product codes can achieve better performance with less memory and low latency. The number of flipflops consumed is only 280. So the hardware required to implement is also very much reduced. With the enhancement of the comprehensive mission management system the proposed design method will also take minimum amount of cycles to find out the error. In future there is a chance to develop the error correcting scheme for analog circuits also by using sophisticated EDA tools.

VI. REFERENCES


