ISSN: 2278 – 909X
International Journal of Advanced Research in Electronics and Communication Engineering (IJARECE)
Volume 2, Issue 12, December- 2013

LOSSLESS IMPLEMENTATION OF NAND FLASH MEMORY ARCHITECTURE USING PRODUCT CODE SCHEMES

B.Alekhya[1] , B.Teena prasoona[2], A.Sairamya[3]

[1]*Student (M.Tech), VLSI (SD), Department of ECE, Avanthi Institute of Engineering &Technology ,*
[2]*Asst.Professor, Department of ECE, Avanthi Institute of Engineering &Technology,*
[3]*Student (M.Tech), VLSI (SD), Department of ECE, Avanthi Institute of Engineering &Technology ,*

Abstract

Flash memory has become the dominant technology for non-volatile memory. We focus on NAND Flash memories since they have lower erase times, less chip area per cell which allows greater storage density, and lower cost per bit than NOR Flash memories [2]. Specifically, we focus on multi-level cell (MLC) Flash memories which store two or more bits per cell by supporting four or more voltage states. These have even greater storage density and are the dominant Flash memory technology. 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. We also 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. The results in the paper which is analyzed using Xilinx software with NAND flash design.

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

1. INTRODUCTION

There are some inherent limitations of NAND Flash memories. These include write/read disturbs, data retention errors, bad block accumulation, limited number of writes [3]–[5], and stress-induced leakage current [6]. In recent years, due to cell size scaling, these issues have become critical. In particular, re- liability of MLC memory significantly degrades due to reduced gap between adjacent threshold levels.

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.

2. FAULT-TOLERENT 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 D be the maximum number of error bits that it can detect, and in one error combination that strikes the system, let e_e, 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 [6]. 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 [7]. 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. 1.

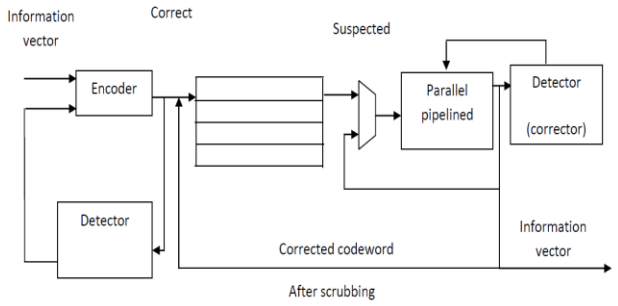

FIG1: Fault-tolerant memory architecture, with pipelined corrector.

2.1. 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., $c = i.G$. 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.

$$\begin{pmatrix}
1&0&0&0&0&0&0&1&0&0&1&1&1&0&1\\
0&1&0&0&0&0&0&1&1&0&0&1&1&1&0\\
0&0&1&0&0&0&0&0&1&1&1&0&0&0&1\\
0&0&0&1&0&0&0&1&0&1&1&1&0&0&0\\
0&0&0&0&1&0&0&0&1&0&1&1&1&0&0\\
0&0&0&0&0&1&0&0&0&1&0&1&1&1&0\\
0&0&0&0&0&0&1&0&0&0&1&0&1&1&1
\end{pmatrix}$$

Fig2: Generator matrix for the (15, 7, 5) EG-LDPC in systematic format; Note the identity matrix in the left columns.

Fig. 2 shows the systematic generator matrix to generate (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 $G = [I : X]$. Fig. 3 shows the encoder circuit to compute the parity bits of the (15, 7, 5) EG-LDPC code. In this figure $i = (i_0, i_1, i_2, \ldots\ldots i_6)$ is the information vector and will be copied to $(c_0, \ldots\ldots c_6)$ 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.

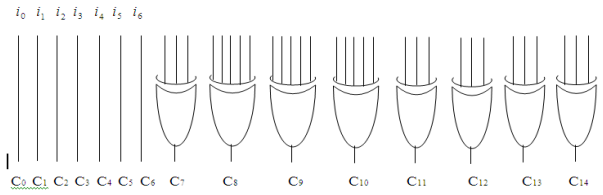

FIG 3: Structure of an encoder circuit for the (15, 7, 5) EG-LDPC code; i_0 to i_6 are 7-bit information vector. Each of the XOR gates generates one parity bit of the encoded vector. The codeword consists of seven information bits followed by eight parity bits.

3. Memory Errors

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

1. Hard errors are caused due to fabrication defects in the memory chip and cannot be corrected once they start appearing.

2. Soft errors on the other hand are caused predominantly by electrical disturbances.

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. There is a parity checking sequence method is calculation:

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 method can only detect the errors but doesn't rectify the errors. The memory system consists of fault tolerant analysis of design and represents the correction in the memory storage with different Error correcting codes (ECC) with different Fault secure detector.

3.1 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 [8] 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 is a distinct value. We have m bits distinguished data.

$$2^m \geq m + k + 1 \qquad (1)$$

3.2 ECC with Fault Secure Detector

3.2.1 Error-Correcting Code:

Let $i = (i_0, i_1, i_2, \ldots\ldots i_{k-1})$ be the k -bit information vector that will be encoded into an n -bit codeword, $c = (c_0, c_1, \ldots\ldots c_{n-1})$. For linear codes, the encoding operation essentially performs the following vector-matrix multiplication:

$$c = i.G \qquad \ldots\ldots (1)$$

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:

$$s = c.H^T \qquad \ldots\ldots (2)$$

The $(n-k)$ -bit vector s is called the syndrome vector. A syndrome vector is zero if c is a valid codeword, and nonzero if c is an erroneous codeword. Each code is uniquely specified by its generator matrix or parity-check matrix.

A code is a systematic code if every codeword consists of the original k -bit information vector followed by $n-k$ parity bits. With this definition, the generator matrix of a systematic code must have the following structure:

$$G = [I : X]. \qquad \ldots\ldots\ldots (3)$$

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.

3.2.2 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.

4. SYNTHESIS RESULTS

The result explains about the NAND flash memory waveform design for multiple input data.

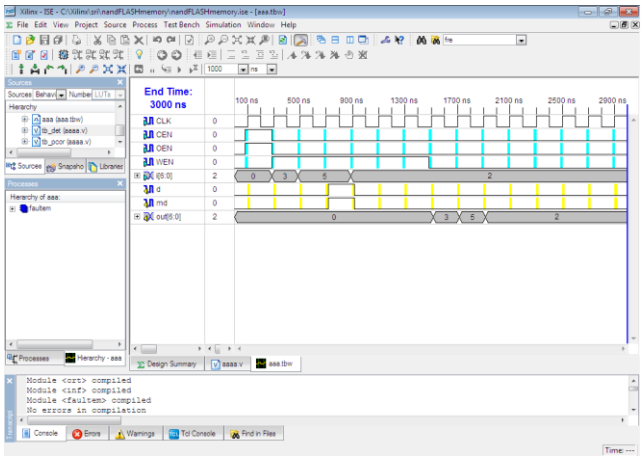

Fig-4 Waveform for NAND Flash memory Design

This result explains about the internal RTL schematic design of architecture.

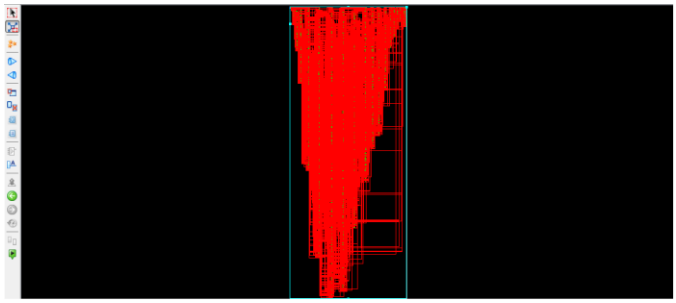

Fig-5 RTL Schematic Design of NAND flash memories

This result explains about the number of Look up tables (LUT) and flip-flops are designed in the circuit.

Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	280	3,840	7%	
Number of 4 input LUTs	920	3,840	23%	
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			

Fig-6 device utilization summary

6. CONCLUSION

This paper explains about the NAND Flash design memories in the Xilinx and the architecture structure in RTL schematic and specification details about the flip-flops, Look-up-tables (LUT) used. The product code schemes are used for the correction of any error in the design and recover the design status in the circuit. We can do the future work by using different codes schemes about the better result in the design of the flash memories in Non-volatile circuits.

REFRENCES

[1] S. Gregori, A. Cabrini, O. Khouri, and G. Torelli, "On-chip error correcting techniques for new-generation flash memories," *Proc. IEEE*, vol. 91, no. 4, pp. 602–616, Apr. 2003.

[2]T. Chen, Y. Hsiao, Y. Hsing, and C.Wu, "An adaptive-rate error correction scheme for NAND flash memory," in *Proc. 27th IEEE VLSI Test Symp.*, 2009, pp. 53–58.

[3] H. Choi, W. Liu, and W. Sung, "VLSI implementation of BCH error correction for multilevel cell NAND flash memory," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 18, no. 5, pp. 843–847, May 2010.

[4] T. Jianzhong, Q. Qinglin, B. Feng, R. Jinye, and Z. Yongxiang, "Study and design on high reliability mass capacity memory," in *Proc. IEEE Int. Conf. Softw. Eng. Service Sci.*, 2010, pp. 701–704.

[5] P. Desnoyers, "Empirical evaluation of NAND flash memory performance," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 1, pp. 50–54, 2010.

[6] D. Rossi and C. Metra, "Error correcting strategy for high speed and high density reliable flash memories," *J. Electron. Test.: Theory Appl.*, vol. 19, no. 5, pp. 511–521, Oct. 2003.

[7] N. Agrawal, V. Prabhakaran, T. Wobber, J.Davis,M.Manasse, and R. Panigrahy, "Design tradeoffs for SSD performance," in *Proc. USENIX Annu. Techn. Conf. Annu. Techn. Conf.*, 2008, pp. 57–70.

[8] L. Pantisano and K. Cheung, "Stress-induced leakage current (SILC) and oxide breakdown: Are they from the same oxide traps?," *IEEE Trans. Device Mater. Reliab.*, vol. 1, no. 2, pp. 109–112, Jun. 2001.

BIOGRAPHIES

B.Alekhya pursuing M.Tech in specialization of VLSI at Avanthi Institute of Engineering Technology in Department of Electronics and Communication Engineering.

B.Teena prasoona completed M.Tech and working as an Assistant Professor in Avanthi InstituteofEngineering&Technology in Department of Electronics and communication Engineering.