IOSR Journal of Electronics and Communication Engineering (IOSR-JECE)
e-ISSN: 2278-2834,p- ISSN: 2278-8735.Volume 8, Issue 2 (Nov. - Dec. 2013), PP 71-78
www.iosrjournals.org

Efficient Majority Logic Fault Detection with Difference-Set Codes for Memory Applications

[1]B. Arun Kumar, [2]M. R. N. Tagore, [3]Dr. Giri Babu Kande
[1]*PG Student (M.Tech VLSI), Dept. Of ECE, Vasireddy Venkatadri Ins. Tech., Nambur, Guntur, AP, India*
[2]*Associate Professor, Dept. Of ECE, Vasireddy Venkatadri Ins. Tech., Nambur, Guntur, AP, India*
[3] *Professor & Head, Dept. Of ECE, Vasireddy Venkatadri Ins. Tech., Nambur, Guntur, AP, India*

Abstract: *In this Paper, we focus on a class of LDPC codes known as Euclidean Geometric (EG) LDPC codes, which are constructed deterministically using the points and lines of a Euclidean geometry. Minimum distances for EG codes are also reasonably good and can be derived analytically. memory error correction code has been implemented using pipelined cyclic corrector where majority logic gate determined the error .LDPC soft error decoding is also implemented for the same memory error detection and correction comparison of the results are done .as the majority gate can detect only upto 2 error the extending majority gate with ldpc soft decoding can decrease the bit error rate.*

Keywords: *Error correction code (ECC), memory fault tolerance, soft-error rate (SER), Block codes, low-density parity check (LDPC), Memory.*

I. Introduction

Memory cells have been protected from soft errors for more than a decade; due to the increase in soft error rate in logic circuits, the encoder and decoder circuitry around the memory blocks have become susceptible to soft errors as well and must also be protected. We introduce a new approach to design fault-secure encoder and decoder circuitry for memory designs.

Hamming codes are often used in today's memory systems to correct single error and detect double errors in any memory word. In these memory architectures, only errors in the memory words are tolerated and there is no preparation to tolerate errors in the supporting logic (i.e. encoder and corrector).

However combinational logic has already started showing susceptibility to soft errors, and therefore the encoder and decoder (corrector) units will no longer be immune from the transient faults. Therefore, protecting the memory system support logic implementation is more important. Here we proposed a fault tolerant memory system that tolerates multiple errors in each memory word as well as multiple errors in the encoder and corrector units.

We illustrate using Euclidean Geometry codes and Projective Geometry codes to design the fault-tolerant memory system, due to their well-suited characteristics for this application.

Low density parity-check (LDPC) codes were first discovered by Gallager in the early 1960s [2] and have recently been rediscovered and generalized .It has been shown that these codes achieve a remarkable performance with iterative decoding that is very close to the Shannon limit[3]. Consequently, these codes have become strong competitors to turbo codes for error control in many communication and digital storage systems where high reliability is required. LDPC codes can be constructed using random or deterministic approaches.

In this paper, we focus on a class of LDPC codes known as Euclidean Geometric (EG) LDPC codes, which are constructed deterministically using the points and lines of a Euclidean geometry [1, 6].

The paper consists of an efficient VLSI implementation of fault secure encoder and decoder for memory applications. A novel and efficient VLSI architecture is proposed and implemented for the fault secure memory. The VLSI architecture has been authored in Verilog code for fault secure encoder and decoder for memory and its synthesis was done with Xilinx XST. Xilinx ISE Foundation 9.1i has been used for performing mapping, placing and routing. For behavioral simulation and place and route simulation ISE simulator has been used. The Synthesis tool was configured to optimize for area and high effort considerations. The interest of the project work is an attempt to obtain fault secure memory architecture. This fault secure memory is used in computer systems mainly servers and in memory applications and also used in military applications.

II. Some Error Correcting Codes

In Particular, we identify a class of error-correcting codes (ECCs) that guarantees the existence of a simple fault-tolerant detector design. This class satisfies a new, restricted definition for ECCs which guarantees that the ECC codeword has an appropriate redundancy structure such that it can detect multiple errors occurring in both the stored codeword in memory and the surrounding circuitries.

In this part of the paper, we present our novel, restricted ECC definition for our fault-secure detector capable codes. Before starting the details of our new definition we briefly review basic linear ECCs.

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

$$c=i.G \qquad \text{.....(1)}$$

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:

$$s=c.H^T \qquad \text{……(2)}$$

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:

$$G=\left[I:X\right]. \qquad \text{……..(3)}$$

Where I is a $k\times k$ identity matrix and X is a $k\times(n-k)$ matrix that generates the parity-bits. The advantage of using systematic codes is that there is no need for a decoder circuit to extract the information bits. The information bits are simply available in the first k bits of any encoded vector. A code is said to be a cyclic code if for any codeword c , all the cyclic shifts of the codeword are still valid code words. A code is cyclic if the rows of its parity-check matrix and generator matrix are the cyclic shifts of their first rows.

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.

Table 1 EG-LDPC UPPER AND LOWER BOUNDS ON CODE LENGTH:

Hamming bound	EG-LDPC	Gilbert-Varshamov bound
(14,7,5)	(15,7,5)	(17,7,5)
(58,37,9)	(63,37,9)	(67,37,9)
(222,175,17)	(255,175,17)	(255,175,17)

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 and 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. Implementation of The Proposed System

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 be the maximum number of error bits that the code can correct and be the maximum number of error bits that it can detect, and in one error combination that strikes the system, let , , and be the number of errors in encoder, a memory word, and corrector, and let and be the number of errors in the two separate detectors monitoring the encoder and corrector units. In conventional designs, the system would guarantee error correction as long as and . In contrast, here we guarantee that the system can correct any error combination as long as, and This design is feasible when the following two fundamental properties are satisfied:

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.

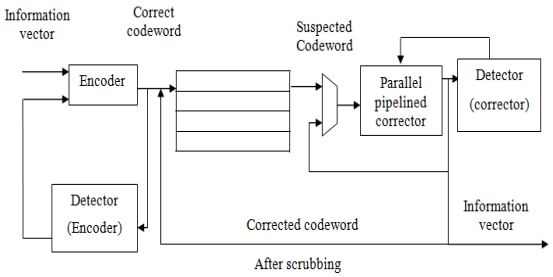

Figure 1 Fault-tolerant memory architecture, with pipelined corrector.

Table 2 Detector, Encoder, and Corrector Circuit Area In The

Code	(15,7,5)	(63,37,9)	(255,175,17)
Detector	45	501	3825
Encoder	22	355	6577
Serial corrector	19	83	331
Parallel corrector	285	5229	84405

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.

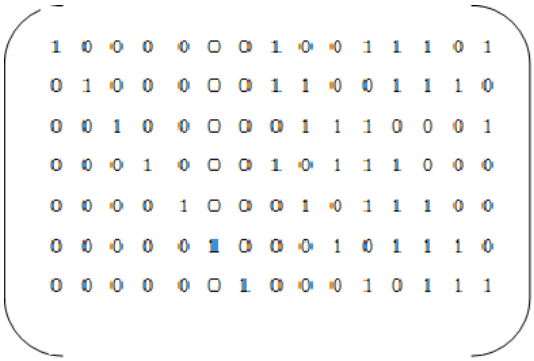

Figure 2 Generator matrix for the (15, 7, 5) EG-LDPC in systematic format;

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 $i = (i_0, i_1, i_2, ...i_6)$ is the information vector and will be copied to $(c_0, ...c_6)$ 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-in puts 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.

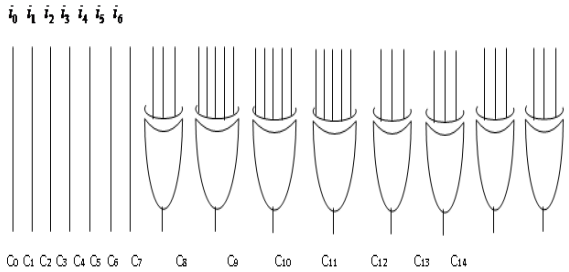

Figure 3 Structure of an encoder circuit for the (15, 7, 5) EG-LDPC code; to are 7-bit information vector.

Fault Secure Detector:

The core of the detector operation is to generate the syndrome vector, which is basically implementing the following vector-matrix multiplication on the received encoded vector C and parity-check matrix H :

$$S = c.H^T \quad \text{..... (1).}$$

Therefore each bit of the syndrome vector is the product of C with one row of the parity-check matrix. This product is a linear binary sum over digits of C where the corresponding digit in the matrix row is 1. This binary sum is implemented with an XOR gate. Fig. 4 shows the detector circuit for the (15, 7, 5) EG-LDPC code. Since the row weight of the parity-check matrix is ρ , to generate one digit of the syndrome vector we need a ρ -input XOR gate, or $(\rho - 1)$ 2-input XOR gates. For the whole detector, it takes $n(\rho - 1)$ 2-input XOR gates. Table II illustrates this quantity for some of the smaller EG-LDPC codes. Note that implementing each syndrome bit with a separate XOR gate satisfies the assumption of no logic sharing in detector circuit implementation.

An error is detected if any of the syndrome bits has a nonzero value. The final error detection signal is implemented by an OR function of all the syndrome bits. The output of this -input OR gate is the error detector signal.

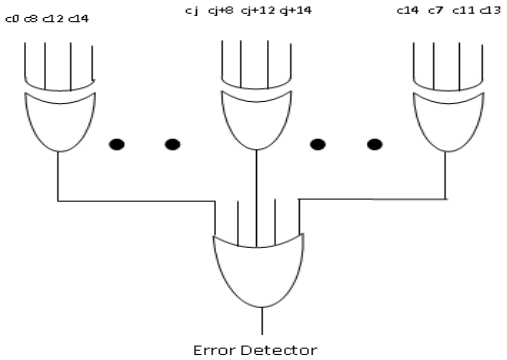

Figure 4 Fault-secure detector for (15, 7, 5) EG-LDPC code.

Corrector:

One-step majority-logic correction is a fast and relatively compact error-correcting technique [1]. There is a limited class of ECCs that are one-step-majority correctable which include type-I two-dimensional EG-

LDPC. In this section, we present a brief review of this correcting technique. Then we show the one-step majority-logic corrector for EG-LDPC codes.

One-Step Majority-Logic Corrector:

One-step majority logic correction is the procedure that identifies the correct value of a each bit in the codeword directly from the received codeword; this is in contrast to the general message-passing error correction strategy (e.g., [5]) which may demand multiple iterations of error diagnosis and trial correction. Avoiding iteration makes the correction latency both small and deterministic. This technique can be implemented serially to provide a compact implementation or in parallel to minimize correction latency.

This method consists of two parts:
1) Generating a specific set of linear sums of the received vector bits and
2) Finding the majority value of the computed linear sums.

The majority value indicates the correctness of the code-bit under consideration; if the majority value is 1, the bit is inverted, otherwise it is kept unchanged. The theory behind the one-step majority corrector and the proof that EG-LDPC codes have this property are available in [1]. Here we overview the structure of such correctors for EG-LDPC codes.

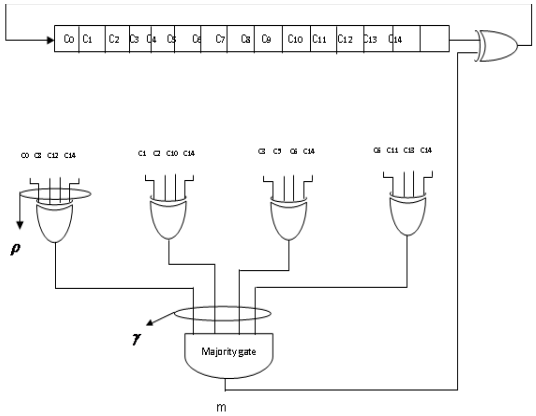

Figure 5 Serial one-step majority logic corrector structure to correct last bit (bit14th) of 15-bit (15,7,5) EG-LDPC code

Majority Circuit Implementation:

Here we present a compact implementation for the majority gate using Sorting Networks [6]. The majority gate has application in many other error-correcting codes, and this compact implementation can improve many other applications.

A majority function of γ binary digits is simply the median of the digits (where we define the median of an even number of digits as the $\gamma/2+1$ st smallest digit).

To find the median of the γ inputs, we do the following:

1) Divide the γ inputs into two halves with size $\gamma/2$;

2) Sort each of the halves;

3) The median is 1 if for $i=1,2,.....,\gamma/2$ the i th element of one half and the $(\gamma/2+1-i)$ th element of the other half are both 1.

We use binary Sorting Networks [6] to do the sort operation of the second step efficiently. An -input sorting network is the structure that sorts a set of bits, using 2-bit sorter building blocks. Fig. 6(a) shows a 4-input sorting network. Each of the vertical lines represents one comparator which compares two bits and assigns the larger one to the top output and the smaller one to the bottom [see Fig. 6(b)]. The four-input sorting network, has five comparator blocks, where each block consists of two two-input gates; overall the four-input sorting network consists of ten two-input gates in total.

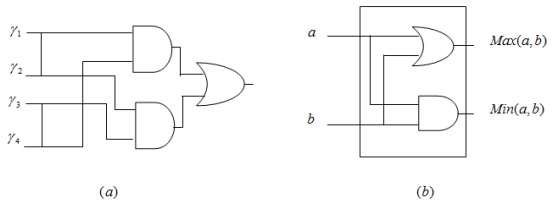

Figure 6.(a) Four-input sorting network; (b).each vertical line shows a one-input

The algorithm will be explained on the basis of the example code already introduced in equation 1 and figure 1. An error free received codeword would be e.g. c = [1 0 0 1 0 1 0 1]. Let's suppose that we have a BHC channel and the received the codeword with one error – bit c1 flipped to 1.

1. In the first step all v-nodes ci send a "message" to their (always ci → fj 2 in our example) c-nodes fj containing the bit they believe to be the correct one for them. At this stage the only information a v-node ci has, is the corresponding received i-th bit of c, yi .

That means for example, that c0 sends a message containing 1 to f1 and f3, node c1 sends messages containing y1 (1) to f0 and f1, and so on.

c-node	received/sent			
f_0	received:	$c_1 \to 1$	$c_3 \to 1$	$c_4 \to 0$	$c_7 \to 1$
	sent:	$0 \to c_1$	$0 \to c_3$	$1 \to c_4$	$0 \to c_7$
f_1	received:	$c_0 \to 1$	$c_1 \to 1$	$c_2 \to 0$	$c_5 \to 1$
	sent:	$0 \to c_0$	$0 \to c_1$	$1 \to c_2$	$0 \to c_5$
f_2	received:	$c_2 \to 0$	$c_5 \to 1$	$c_6 \to 0$	$c_7 \to 1$
	sent:	$0 \to c_2$	$1 \to c_5$	$0 \to c_6$	$1 \to c_7$
f_3	received:	$c_0 \to 1$	$c_3 \to 1$	$c_4 \to 0$	$c_6 \to 0$
	sent:	$1 \to c_0$	$1 \to c_3$	$0 \to c_4$	$0 \to c_6$

Figure 7 Message Sending Pattern

In the second step every check nodes fj calculate a response to fj → ci every connected variable node. The response message contains the bit that fj believes to be the correct one for this v-node ci assuming that the other v-nodes connected to fj are correct. In other words: If you look at the example, every c-node fj is connected to 4 v-nodes. So a c-node fj looks at the message received from three v-nodes and calculates the bit that the fourth v-node should have in order to fulfill the parity check equation. Table 3 gives an overview about this step.

Important is, that this might also be the point at which the decoding algorithm terminates. This will be the case if all check equations are fulfilled. We will later see that the whole algorithm contains a loop, so another possibility to stop would be a threshold for the amount of loops.

Next phase: the v-nodes receive the messages from the check ci → fj nodes and use this additional information to decide if their originally received bit is OK. A simple way to do this is a majority vote. When coming back to our example that means, that each v-node has three sources of information concerning its bit. The original bit received and two suggestions from the check nodes. Table 3 illustrates this step. Now the v-nodes can send another message with their (hard) decision for the correct value to the check nodes.

In our example, the second execution of step 2 would terminate the decoding process since c1 has voted for 0 in the last step. This corrects

Table 3 Step 3 of the described decoding algorithm

v-node	y_i received	messages from check nodes		decision
c_0	1	$f_1 \to 0$	$f_3 \to 1$	1
c_1	1	$f_0 \to 0$	$f_1 \to 0$	0
c_2	0	$f_1 \to 1$	$f_2 \to 0$	0
c_3	1	$f_0 \to 0$	$f_3 \to 1$	1
c_4	0	$f_0 \to 1$	$f_3 \to 0$	0
c_5	1	$f_1 \to 0$	$f_2 \to 1$	1
c_6	0	$f_2 \to 0$	$f_3 \to 0$	0
c_7	1	$f_0 \to 1$	$f_2 \to 1$	1

IV. Results and Discussion

The behavioral simulation and post rout simulations waveforms for the fault secure encoder is shown in figure 8 and figure 9. In the figure 8,the input is information vector and output is the detector output d which detects the errors in the encoder. First information vector is given to encoder it gives encoded vector as an output which is n-bit length. This encoded vector is given as input to the detector. Any error is present in encoded vector the detector output is '1'. If it is '0' encoded codeword is correct.

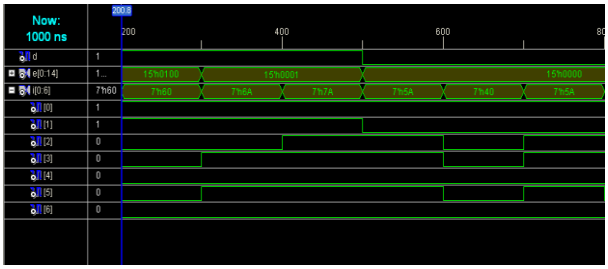

Figure 8 Behavioral simulation waveform for the fault secure encoder

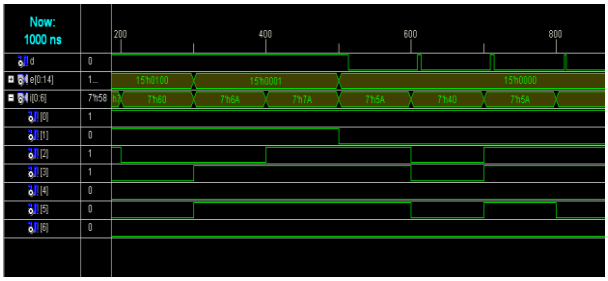

Figure 9 Post route simulation waveform for the fault secure encoder

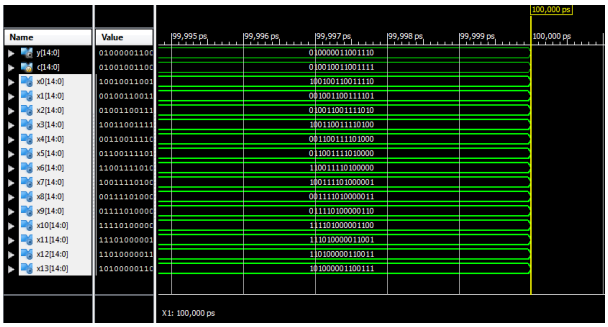

Fig 10: majority logic gate corrector

V. Conclusions

Using this architecture tolerates transient faults both in the storage unit and in the supporting logic (i.e., encoder, decoder (corrector), and detector circuitries). The main advantage of the proposed architecture is using this detect-and-repeat technique we can correct potential transient errors in the encoder or corrector output and provide fault-tolerant memory system with fault-tolerant supporting circuitry. ldpc soft decoding takes less area compared to majority logic gate techniques and in this architecture there is no need of decoder because we use systematic generated matrix.

Fault secure encoder and decoder for memory applications is to protect the memory and supporting logic from soft errors. The proposed architecture tolerates transient faults both in the storage unit and in the supporting logic. Scope for further work is instead of memory we use nano memory which provides smaller, faster, and lower energy devices which allow more powerful and compact circuitry.

Acknowledgements

The authors would like to thank the anonymous reviewers for their comments which were very helpful in improving the quality and presentation of this paper.

References

[1] Shu Lin and Daniel J. Costello. Error Control Coding. Prentice Hall, second edition, 2004.
[2] R. G. Gallager, "Low-density parity-check codes", *IRE Trans. Information Theory*, vol. IT-8, no. 1, pp. 21–28, January 1962.
[3] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes", *Electronics Letters*, vol. 32, no. 18, pp.1645–1646, March 1997.
[4] R. J. McEliece, *The Theory of Information and Coding*. Cambridge, U.K.: Cambridge University Press, 2002.
[5] M. Sipser and D. Spielman, "Expander codes," *IEEE Trans. Inf. Theory*, vol. 42, no. 6, pp. 1710–1722, Nov. 1996.
[6] D. E. Knuth, *The Art of Computer Programming*, 2nd ed. Reading, MA: Addison Wesley, 2000.
[7] Allen D. Holliday, Hamming Error-Correction Codes, February 17, 1994 (revised June 15, 2002; March 1, 2004).
[8] H. Tang, J. Xu, S. Lin, and K. A. S. Abdel-Ghaffar, "Codes on finite geometries," *IEEE Trans. Inf. Theory*, vol. 51, no. 2, pp. 572–596, Feb. 2005.
[9] H. Naeimi and A. DeHon, "Fault-tolerant nano-memory with fault secure encoder and decoder," presented at the Int. Conf. Nano-Netw., Catania, Sicily, Italy, Sep. 2007.
[10] S. J. Piestrak, A. Dandache, and F. Monteiro, "Designing fault-secure parallel encoders for systematic linear error correcting codes," *IEEE Trans. Reliab.*, vol. 52, no. 4, pp. 492–500, Jul. 2003.