Protecting Flash Memories with a High Reliability and Low Cost ECC[1]

Lake Bu
Reliable Computing Laboratory
Electrical and Computer Engineering
Boston University
bulake@bu.edu

Mark Karpovsky
IEEE Life Fellow
Reliable Computing Laboratory
Electrical and Computer Engineering
Boston University
markkar@bu.edu

Abstract: Error correcting codes (ECC) are widely used in flash memories. Most popular ECCs nowadays are single-bit or double-bit binary error correction. In this paper we propose a new class of ECC which provides higher reliability for memory: single-byte and double-byte error correction, where the size of a byte can be customized without increasing the decoding algorithm's complexity. This new class of ECC is called Group Testing Based Codes (GTB codes). The decoding cost of GTB codes is also much lower than that of the popular non-binary ECCs in terms of hardware redundancy and time latency. The advantage of this new code is that it does not involve finite field computations over *GF*(*Q*), *Q* is the number of possible different values in a symbol of a codeword, in calculating the syndromes, which makes it possible to have simple decoding algorithms.

Keywords: flash memory, reliability, error control codes, group testing, superimposed codes, GTB codes, encoding, decoding, redundancy.

I. INTRODUCTION

In the manufacture of flash memories, usually NOR Flash does not need ECC codes. However NAND Flash will need ECCs to protect data from being distorted [1]. Popular ECCs are single error correction double error detection binary Hamming codes, non-binary Hamming codes, double error correction triple error detection Bose-Chaudhuri- Hocquenghem (BCH) codes [2], and Reed-Solomon codes. Besides the conventional ECC codes, interleaved codes are also utilized to convert bit-error correction to byte-error correction while keeping the simple decoding algorithm of binary error correction. Among them are interleaved Hamming codes and interleaved Orthogonal Latin Square Codes (OLSCs).

The binary codes have simple decoding algorithms because all computations are carried out over *GF*(2) binary field. In hardware implementation, it only required XORs for binary additions and ANDs for binary multiplications in the finite field. However, such codes (single-bit error correcting binary Hamming codes, double-bit error correcting binary BCH codes) provide limited protection for the Flash memories. They can either maintain the fault tolerance of the memories under singe-bit our double-bit distortions, or detect, not necessarily correct, errors under double-bit or triple-bit distortions.

The non-binary codes enable the flash memories with much stronger reliability. The protection is over byte distortions. For instance, if a memory is allocated by *b*-bit bytes, than a non-binary ECC such as Reed-Solomon code can protect the memory from an error on *n* bytes with a redundancy of 2*n* bytes. However decoding of such a code demands computations over *GF*(*Q*) finite field ($Q = 2^b$). The hardware and time cost is much more expensive than that of the binary ECCs. What's more, the bigger the finite field is, the more complicated the computation will be. Although Reed-Solomon codes are widely used in military or state facilities such as satellites, they are not very cost-efficient in regular

[1] This work is sponsored by the NSF grant CNS 1012910.

industry products.

Therefore, this paper will introduce a new class of codes (GTB codes) which has the best of the two sides: a strong protection for memories, and a simple decoding algorithm which results in low decoding complexity in hardware and time. This new class of codes is non-binary in every symbol of a codeword. However, the codewords are defined/encoded by a binary matrix, and decoded by another binary matrix, which only requires bitwise XORs in field additions despite how many bits a byte has. In this way the computation of syndromes are merely binary XORs. The analysis of syndromes is also just binary operations. Thus the GTB codes protects memories on byte level while having binary decoding algorithms. The only drawback of GTB codes is that it costs more redundant bytes in storage than non-binary ECCs such as Reed-Solomon codes.

The following article is organized as: section II is about the introduction of group testing technique and superimposed codes; section III is about the construction and properties of GTB codes using superimposed codes' matrices; section VI is about how to encode the GTB codes; section V is about the error locating and correction of GTB codes; section VI is to compare GTB codes with other popular ECCs.

II. SUPERIMPOSED CODES

A. Definitions of the superimposed codes

Group testing plays an important role in biology, chemistry, computer diagnosis. Using group testing technique the number of testings can be dramatically reduced. Superimposed codes are very widely used to generate test patterns for group testing.

We have two definitions of superimposed codes.

Definition 2.1: Let $1 \le m \le N$, $N > 1$ be integers and *M* is a binary $A \times N$ matrix, where each of the *N* columns is a codeword. The set of columns of *M* is called a (m, N, l) superimposed code if the Boolean sums of all up to any *m* subset of columns are different, and the maximum weight of the rows of *M* is l [3].

Definition 2.2: Let $M_{i,j} \in \{0,1\}$ be the element in row i and column j in the matrix *M* of size $A \times N$. The set of columns of *M* is called an *m*-superimposed code, if for any set of up to *m* columns $(m \subset [N])$ and any single column $h \notin m$, there exists a row $i \in [A]$, for which $M_{i,h} = 1$ for column *h*, and $M_{i,j} = 0$ for all $j \in m$ [4] [5].

The latter is stricter than the former. We will use both properties for the sections below.

B. Construction of superimposed codes

Firstly we will define a few notations:

q: Number different elements in the *GF*(*q*) finite field for the Reed-Solomon code C_q.

n_q: Length of Reed-Solomon codeword C_q.

k_q: Number of codewords' information digits in C_q.

d_q: The distance of Reed-Solomon code C_q.

A: Length of a superimposed code C_{SI}.

N: Size of a superimposed code C_{SI}.

d_{SI}: The distance of superimposed code C_{SI}.

m: The maximum number of the subsets whose Boolean sums are different. In group testing it is often referred to as the multiplicity of errors.

l: The maximum weight of each row in *M*.

Construction 2.1: Let C be a $(n_q, k_q, d_q)_q$ *q*-ary linear error correcting code, $q \neq 2^s$, s is an integer. Suppose we represent each element in *GF*(*q*) by a q-bit binary vector with Hamming weight 1. Construct *C*' by substituting every *q*-ary element in the codewords *C* by its corresponding binary vector. The binary linear code *C*' has the following parameters [6]:

$$A = qn_q$$
$$N = q^{k_q}$$
$$l = q^{k_q - 1}$$
$$d_{SI} = 2d_q$$
$$m = \left\lfloor \frac{n_q - 1}{n_q - d_q} \right\rfloor = \left\lfloor \frac{n_q - 1}{k_q - 1} \right\rfloor$$

There are other ways to construct superimposed codes. When $m = 1$ the columns of a binary Hamming check matrix can be selected as codewords of a 1-superimposed code [7]. These methods will generate binary matrices for superimposed codes, which will be used as decoding matrices for GTB codes.

C. Properties of superimposed codes

Due to certain properties of the superimposed codes, they can be used for error locating in group testing. Firstly some definitions are to be made for later convenience.

Definition 2.3: A binary matrix is *m*-separable if the bitwise OR of up to *m* columns are mutually different.

Definition 2.4: For any two *A*-bit binary vectors *u* and *v*, we say that *u* covers *v* if $u \cdot v = u$, where $\cdot$ denotes the dot product of the two vectors. An ($A \times N$) matrix *M* is *m*-disjunct if the bitwise OR of any set of no more than *m* columns does not cover any other single column that is not in the set. The columns of an *m*-disjunct matrix compose an *m*-superimposed code.

It has been proved that the matrix *M* constructed from superimposed codes is not only *m*-separable but also *m*-disjunct [5].

As an ($A \times N$) *m*-superimposed code matrix is used in group testing, *A* is called the number of total tests needed to find *m* errors, and *N* the total number of elements participating in the tests.

Example 2.1: A $(n_q, k_q, d_q)_q = (7,4,3)_2$ Hamming code has a checking matrix where all columns are mutually different and not equal to 0. This makes the matrix a 1-separable matrix and 1-disjunct matrix. Therefore it is also a 1-superimposed code:

$$M = \begin{vmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{vmatrix}$$

According to its properties, this matrix can be used in group testing to identify single errors. Say, there are 7 people out of which 1 person is a virus carrier. These 7 people are numbered with 1-7. According to the test pattern above, there are 3 groups of mingled blood tests with participants: {7, 6, 5, 4}, {7, 6, 3, 2} and {7, 5, 3, 1}. Each test will result in a syndrome of either 1 (the mingle blood has the virus) or 0 (the mingled blood does not have the virus). The 3 syndromes will identify the virus carrier. For example, if the 3 syndromes are (100), it is the number 4 participant who carries the virus.

D. Bounds of superimposed codes

According to D'yachkov and Singleton, the lower and upper bounds on a minimum length *A* of superimposed codes are [4] [8]:

$$\begin{cases} m=1, & A=\lceil \log_2 N \rceil \\ m\geq 2, & \Omega(\dfrac{m^2 \log_2 N}{\log_2 m}) \leq A \leq \mathrm{O}(m^2 \log_2 N) \end{cases}$$

In the above equations, $f(n) = \mathrm{O}(g(n))$ means that there exists a constant *c* and integer *T* such that $f(n) \leq cg(n)$ for all $n > T$. $f(n) = \Omega(g(n))$ means that $g(n) = \mathrm{O}(f(n))$

This bound can be very tight because the lower and upper bounds are just different by: $1/\log_2 m$.

III. GTB CODES

After superimposed codes are introduced, we can use their binary matrices as the check matrices for GTB codes, so that GTB codes will be encoded and decoded under mere binary operations.

Definition 3.1: Let an $A \times N$ *m*-superimposed code matrix be constructed by *Construction 2.1*, and a *Q*-ary linear code *v* satisfies $M \cdot v = 0$ for any $v \in V$, $V \subseteq GF(Q)$. This new $(N, K, D)_Q$ linear code *V* has length *N*, information digit length K, and distance $D = 2m+1$ which is able to correct *m* digits of errors if there is no error masking [7].

Definition 3.2: By $M \cdot v = S$, there will be *A* syndromes and for any syndrome $S_i \in \{S_1, S_2 \cdots S_A\}$ there is a support of the syndrome $S_{\sup}(i)$ and it is defined as:

$$S_{\sup}(i) = \begin{cases} 0, & S_i = 0 \\ 1, & S_i \neq 0 \end{cases}$$

To help understanding, we will firstly examine a simple example when *m* = 1 on how to decode a non-binary GTB codeword over *GF*(*Q*) with a binary matrix.

Example 3.1: A superimposed code matrix constructed from $(n_q, k_q, d_q)_q = (7,4,3)_2$ Hamming code that serves as the checking matrix for code *V* to correct single errors $(m = 1)$. If the *Q-ary* incoming messages are all 3-bit digits, then their *Q* will be 2^3. Digit 1, 2, and 4 will be redundant digits and the rest digits 3, 5, 6, 7 are information digits.

$$M = \begin{array}{c} \begin{array}{ccccccc} R_1 & R_2 & K_3 & R_4 & K_5 & K_6 & K_7 \end{array} \\ \left|\begin{array}{ccccccc} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{array}\right| \end{array}$$

It is easy to check that the last three digits are generated from the product between the checking matrix and the message ($\oplus$ is for XOR):

$$\begin{cases} R_1 = K_3 \oplus K_5 \oplus K_7 \\ R_2 = K_3 \oplus K_6 \oplus K_7 \\ R_4 = K_5 \oplus K_6 \oplus K_7 \end{cases}$$

According to the checking matrix a legal message could be $v = (010,001,000,000,001,010,011)_2$ and each digit is a 3-bit number in *GF*(8) filed. If this message is distorted by an error $e = (0,0,0,0,111,0,0)$ to $\tilde{v} = (010,001,000,000,110,010,011)_2$, when it is applied to the checking matrix we have the syndrome as:

$$M \cdot v = \begin{vmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{vmatrix} \cdot \begin{vmatrix} 010 \\ 001 \\ 000 \\ 000 \\ 110 \\ 010 \\ 011 \end{vmatrix} = S = \begin{vmatrix} 111 \\ 000 \\ 111 \end{vmatrix} \Rightarrow S_{\text{sup}} = \begin{vmatrix} 1 \\ 0 \\ 1 \end{vmatrix}$$

According to this syndrome pattern it is easy to get the error location as S_{sup} which is at K_5, and magnitude $(111)_2$. By XORing this magnitude with K_5 the codeword can be corrected to: $v = (010,001,000,000,001,010,011)_2$.

Remark 3.1: In this example, since the checking matrix only has 0's and 1's, no matter how large the *Q* is, the syndrome computation is always as simple as bitwise XOR of each digit of the codeword. No multiplication or inversion in finite filed GF(*Q*) is involved. This property makes it possible to have dramatic reduce on hardware and time cost in decoding of *V* [7].

Remark 3.2: During decoding, S_{sup} is used for error locating, and *S* is used for error correction. This is always the case when $m \geq 1$.

IV. Encoding of GTB Codes

Encoding of the new code *V* is simple by using the checking matrix *M* generated from superimposed codes. For simplifying explanation, we will introduce the concept of blocks first.

Definition 4.1: For an $(A \times N)$ *m*-superimposed code constructed by *Construction 2.1*, it can be equally segmented into n_q sub-matrices. Each sub-matrix has exactly *q* rows and *N* columns. We call each of these sub-matrices a block, which will be of convenience for later proofs and computations. Each row of a block has *l* 1's and each column of a block has just one 1.

Since the redundancy is $R = A - m$, and according to *Definition 4.1*, the linear combination of rows in each block is always the same, we can just take off one row from any *m* blocks, and all the rows left are linear independent and can be used for encoding the code *V*.

Corollary 4.1: To encode *V* is just to convert the $A - m$ linear independent rows of *M* into standard form, while keeping all the column numbers as original during column permutation.

Example 4.1: A GTB code *V* in *GF*(*Q*)has *K* = 6 information digits and is able to correct double errors. According to (2) it can be constructed by the Reed-Solomon code of *q* = 4 and parameters $(n_q, k_q, d_q)_q = (3,2,2)_4$. Then the linear code *V* has parameters of *N* = 16, *A* = 12, *R* = 10, *K* = 6, *m* = 2, *l* = 4. The code rate of this code is *K*/*N*=0.375 and it is capable of locating and correcting double errors.

$$M = \begin{vmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \end{vmatrix}$$

After removing $m = 2$ rows and transform the new matrix into standard form. Note that during the transformation, columns 13 and 10 are permuted.

$$M' = \begin{array}{c|cccccccccc|cccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 13 & 11 & 12 & 10 & 14 & 15 & 16 \\ \hline 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \end{array}$$

And so the 6 information digits are: $K_{11}, K_{12}, K_{10}, K_{14}, K_{15}, K_{16}$; and the 10 redundant digits are: $R_1, R_2, R_3, R_4, R_5, R_6, R_7, R_8, R_9, R_{13}$. Here $K_i, R_i \in GF(Q) = GF(2^b)$.

Therefore by the identity submatrix, this code is encoded as ($\oplus$ is for XOR):

$$\begin{cases} R_1 = K_{12} \oplus K_{15} \oplus K_{16} \\ R_2 = K_{10} \oplus K_{11} \oplus K_{12} \oplus K_{14} \oplus K_{15} \\ R_3 = K_{10} \oplus K_{15} \oplus K_{16} \\ R_4 = K_{11} \oplus K_{14} \oplus K_{15} \\ R_5 = K_{10} \oplus K_{11} \oplus K_{14} \\ R_6 = K_{11} \oplus K_{12} \oplus K_{15} \\ R_7 = K_{10} \oplus K_{11} \oplus K_{16} \\ R_8 = K_{11} \oplus K_{12} \oplus K_{14} \oplus K_{15} \oplus K_{16} \\ R_9 = K_{10} \oplus K_{11} \oplus K_{12} \\ R_{13} = K_{14} \oplus K_{15} \oplus K_{16} \end{cases}$$

After encoding, this specific code V has K=6 and is able to correct double errors.

V. DECODING OF GTB CODES

A. Error locating

From the definitions of GTB codes we know that the matrix M constructed from superimposed codes is not only m-separable but also m-disjunct. No single column h will be covered by the bit-wise OR of up to m-columns other than h.

Moreover, the number of 1's in each column of M is exactly n_q. This property results in a simple error locating algorithm as shown in the next theorem.

Theorem 5.1: Let $S_{\sup}$ be the A-bit binary vector representing the support of syndromes. Then $u = S_{\sup} \times M$ is a vector of length N, where $\times$ is the arithmetic multiplication. Suppose all errors are successfully detected. We have $u_i \le n_{q,}$, $1 \le i \le N$. Moreover, there are at most m indexes i such that $u_i = n_q$. These indexes correspond to the error locations.

Algorithm 3.1: According to *Theorem 5.1*, let u_i be the value of the i^{th} element (N elements total) of $u = S_{\sup} \times M$. The generalized error locating expression is:

$$u = S_{\sup} \times M;$$
$$\text{for}(i = 1; i \le N; i++)$$
$$(u_i == n_q) \quad ? \quad \text{error : not error};$$

From the above algorithm it is easy to find that the error locating complexity is $A \cdot N$ for the worst case.

Example 5.4: In *Example 4.1*, when the second and the third digits are distorted out of $N = 16$ digits, the A bit binary support syndrome is: $S_{\text{sup}} = (100000110000011)$. Then by arithmetic multiplication $u = S_{\text{sup}} \times M = (133111210011101010111101210)$. Thereby it can be found that $u_2 = u_3 = n_q = 3$, which successfully identifies the 2 distorted digits. At the same time, all the other elements of u are less than $n_q = 3$.

B. Error correction

The finding of errors' magnitudes and error correction are then developed based on the properties introduced in the following theorems.

Theorem 5.2: In an m-superimposed code's matrix M, there always exists $m+1$ rows with which any given $m+1$ columns can form an $(m+1)\times(m+1)$ identity sub-matrix [7].

Theorem 5.2: Let v be an N-digit codeword of code V with a checking matrix constructed from an $A \times N$ matrix of a superimposed code. Let code v be distorted to $\tilde{v}$ by an N-digit error vector $E = \{0,0,e_1 \cdots, e_m, \cdots 0\}$ which has up to m non-zero magnitudes, such that $\tilde{v} = v + E = (\tilde{v}_1, \tilde{v}_2, \cdots, \tilde{v}_N)$. Also let S be the A-digit syndrome where $S = M \cdot \tilde{v}$ and $S = \{S_1, S_2, \cdots S_A\}$. Then for any $e_h \in E$, there always exists a row i in M, where $S_i = e_h$ and so $v_h = \tilde{v}_h \oplus e_h = \tilde{v}_h \oplus S_i$. Here operator $\oplus$ is the bit-wised XOR.

Note that *Theorem 5.2* is actually deducted *Definition 2.2*.

Algorithm 5.1: Based on *Theorem 5.1 and 5.2*, let $E = \{0,0,e_i \cdots, e_m, \cdots 0\}$ be the error vector which distorts an N-digit codeword v to $\tilde{v} = \{\tilde{v}_1, \tilde{v}_2, \cdots, \tilde{v}_N\}$, $x_h \in \{x_1, x_2, \cdots, x_m\}$ the locations/column numbers of these m non-zero magnitudes in E, $M_{i,j}$ the binary value of the element on row i and column j of superimposed code matrix M in size $A \times N$, and S_i the syndrome corresponding to row i such that $S_i = M_{i,*} \cdot \tilde{v}$, the generalized error correction expression is:

```
for (i = 1; i ≤ A; i++) {
  if (S_i ≠ 0) {
    if ((M_{i,x_1} == 1) && (M_{i,x_2} ··· M_{i,x_m} == 0))
      v_{x_1} = ṽ_{x_1} ⊕ S_i;
    else if ( (M_{i,x_2} == 1) && (M_{i,x_1} ··· M_{i,x_m} == 0))
      v_{x_2} = ṽ_{x_2} ⊕ S_i;
          ⋮
    else if (M_{i,x_m} == 1 ) && (M_{i,x_1} ··· M_{i,x_m} == 0))
      v_{x_m} = ṽ_{x_m} ⊕ S_i;
  }
};
```

It is easy to find that the error correction complexity is $A \cdot m$ for the worst case.

VI. COMPARISON WITH OTHER ECCS

The GTB codes work well in terms of decoding complexity comparing to other ECCs. Here for m = 1 and m = 2 we both have experimental results to show the advantages of GTB codes over other popular codes such as Hamming and Reed-Solomon codes.

To protect a page in flash memory, we choose the page size of 512 bits as the codeword's size. When m = 1, the comparison is made between Hamming and GTB code V:

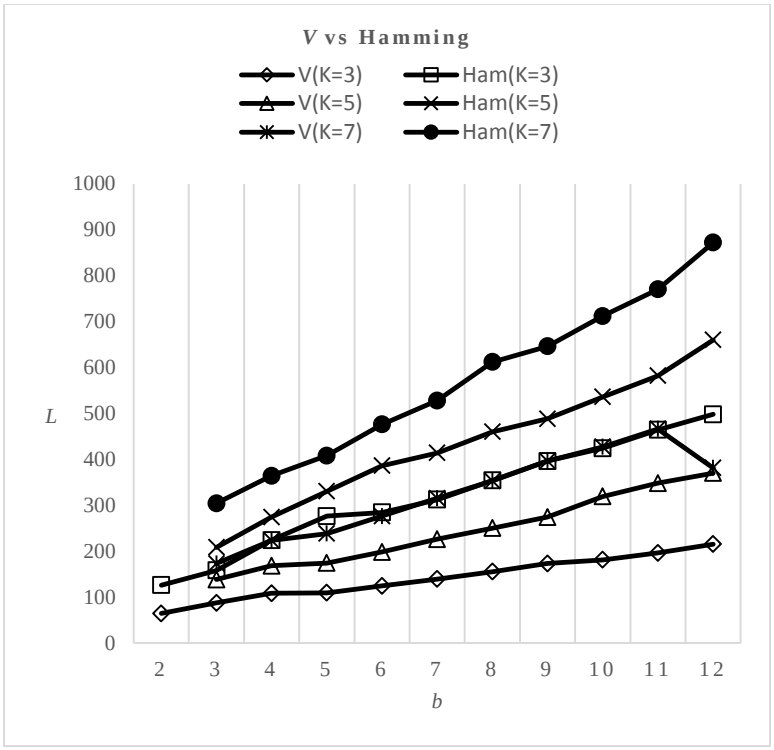

Fig 6.1

And the hardware savings of GTB codes over Hamming codes are:

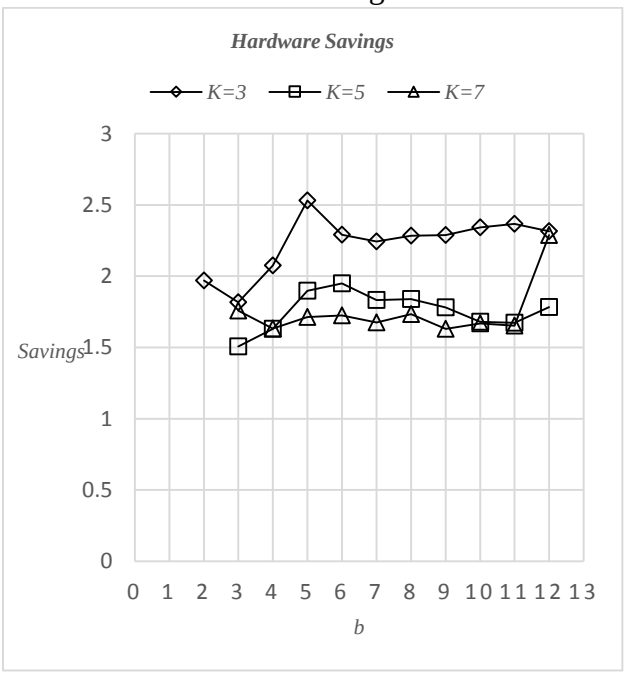

Fig 6.2

It can be seen that when $m = 1$, GTB codes save at least 1.5 times, at most 2.5 times than Hamming codes in protecting a page in flash memory.

When $m = 2$, the comparison is made between Reed-Solomon and GTB code V:

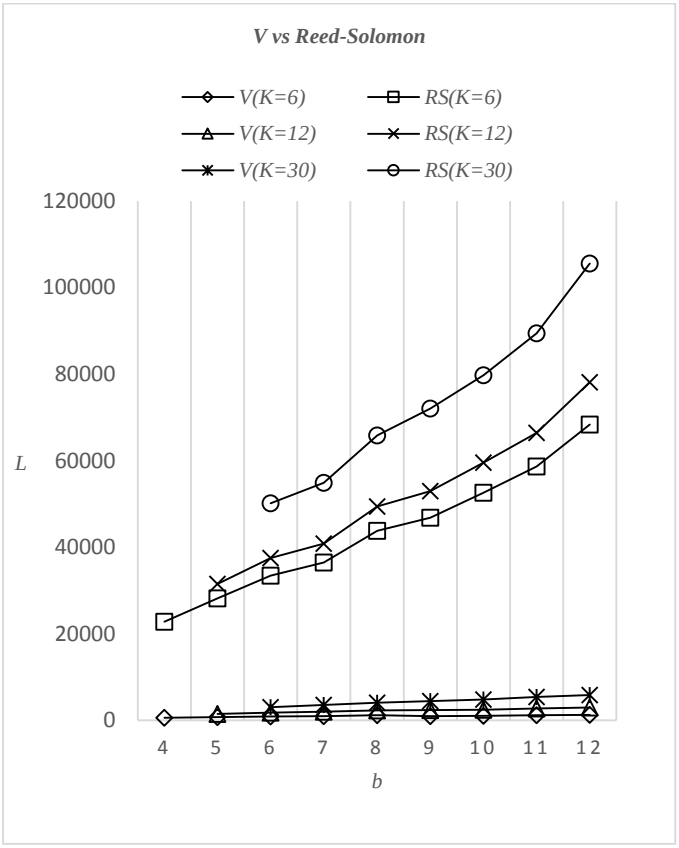

Fig 6.3

And the hardware savings of GTB codes over Reed-Solomon codes are:

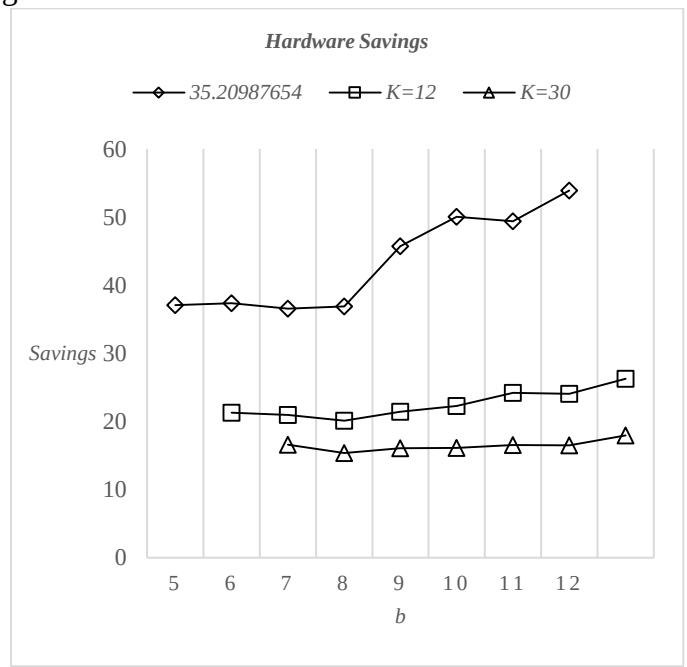

Fig 6.4

It can be seen that when $m = 2$, GTB codes save at least 15 times, at most 55 times than Hamming codes in protecting a page in flash memory.

VII. CONCLUSION

We have introduced this new construction of group testing based error correcting codes (GTB codes) and its application on caches. For codewords with digits in Galois field $GF(Q)$, the presented new codes' computation complexity will not involve any computation in Galois fields. Instead the computation complexity is just $O(\log Q)$. Although it achieves the low decoding complexity at the cost of the increase of redundant digits, through theoretical and experimental prove it is found that it costs much less hardware and time than the popular codes such as Hamming and Reed-Solomon codes etc.

Based on the GTB codes' fast and low complexity decoding which outweighs Hamming and Reed-Solomon codes, we suggest this code as an alternate or replacement of the current popular ECC in flash memory designs, which require small latency and low decoding complexity [9].

REFERENCES

[1] Spansion Inc., "What Types of ECC Should Be Used on Flash Memory?", Application Notes, March, 2011.
[2] Micron Inc., "Error Correction Code (ECC) in Micron Single-Level Cell (SLC) NAND", Application Notes, 2011.
[3] Arkady G. D'yachkov, "On Optimal Parameters of a Class of Superimposed Codes and Designs", 1998.
[4] A. D'yachkov, A.Macula, D.Torney, P. Vilenkin, S.Yekhanin, "New Results in the Theory of Superimposed Codes", 2000.
[5] W. H. Kautz and R. C. Singleton, "Nonrandom binary superimposed codes, IEEE Transaction on Information Theory", Vol. 4, No. 10, October 1964, pp. 363-377.
[6] P. Luo, Pei, A.Y.L. Lin, W. Zhen, and M. Karpovsky, "Hardware implementation of secure Shamir's secret sharing scheme," High-Assurance Systems Engineering (HASE), 2014 IEEE 15th International Symposium on. IEEE, 2014.
[7] Lake Bu, Mark Karpovsky, Zhen Wang, "New Byte Error Correcting Codes with Simple Decoding for Reliable Cache Design", IEEE IOLTS, July 2015.
[8] A. G. D'yachkov and V. V. Rykov, "Bounds for the length of disjunctive codes", Problems of Information Transmission, vol. 18, no.3, pp. 7-13, 1982.
[9] S. Ge, Z. Wang, P. Luo, M. Karpovsky, "Secure memories resistant to both random errors and fault injection attacks using nonlinear error correction codes," Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy. ACM, 2013.