

# A Simple and Efficient Burst Error Correcting Code Based on an Array Code\*

Dan Raphaeli, *Senior Member IEEE*

## Abstract—

In this work, we show that a widely used array code, known as the Even-Odd code, which is targeted at phased burst-errors, may also be useful for non-phased burst errors. A new decoder is proposed for this code which effectively converts it into a more general burst-error-correcting code. The proposed scheme is shown to be capable of correcting almost all bursts up to a certain length, such that its performance is attractive for many communication applications. Since the failure rate is sufficiently low, the code can be practically classified as a burst-error-correcting code. The redundancy in this code is equal to twice the maximal burst length, which is the same redundancy as the lower bound of conventional burst error correcting codes (the Reiger bound). Both the encoder and the decoder have very low complexity, both in terms of number of operations and in terms of computer code size. We analyze the probability of failure, provide tight upper and lower bounds, and show that asymptotically this probability is approaching zero for large blocks.

## I. INTRODUCTION

Burst error correcting codes are needed in virtually uncountable applications. Error correcting codes designed to correct a burst of length  $l$  bits can correct any error pattern that spans not more than  $l$  bits. Such codes will be called complete burst error correcting codes. There are quite a few constructions for complete burst error correcting codes like the Fire codes and other [8]. In this paper we present an error correcting code that is not complete, since it can correct most burst pattern of length  $\leq l$  but not all of them. However, if the number of uncorrectable patterns is sufficiently small, this code can be used in practice as a burst error correcting code. For example, for  $l = 29$  the probability that a burst of length  $\leq l$  is not corrected is about  $10^{-5}$ . Since the frame error rate (FER) in communication systems would be the product of this probability and that of such a burst occurring, for most applications sufficiently low values of FER are achieved despite the limited correction capability. There are other constructions for non complete error correcting codes which achieve lower redundancy e.g. [11] and references therein. However, all these constructions are based on product codes and are optimized for phased errors.

The advantages of the non-complete burst error correcting code presented in this paper are the very efficient and simple decoding algorithm, the low redundancy, and the fact that it is systematic. The redundancy of a complete burst error correcting code has to satisfy the Reiger bound [8]. According to this bound, the number of parity check bits in a code, should satisfy

Dr. Dan Raphaeli is with the Electrical Engineering-Systems Department, Faculty of Engineering, Tel-Aviv University, Tel-Aviv 69978, Israel

\*A LONGER VERSION OF THIS PAPER IS ACCEPTED TO IEEE TRANSACTIONS ON INFORMATION THEORY

$n - k \geq 2l$ , where  $l$  is the burst length in bits,  $k$  is the number of data bits and  $n$  is the number of coded bits. The code presented here satisfies  $n - k = 2l$ . However, it is not a complete code. Therefore, the bound does not apply and can serve for the purpose of comparison only.

A comparison between the correction capability of the scheme proposed here and that of existing efficient<sup>1</sup> complete burst error correcting codes with similar  $k$  and  $n$ , is presented in Table I. The first two were taken from [8, Table 9.3], and the third from [9] ( $n_1 = 31, n_2 = 35$ ), being the best codes we have found in the literature. The multiplication by a small constant in the optimal code parameters refers to the interleaving degree applied. The parameter  $m$  used in the construction of the code will be defined shortly. It generates a code with  $k = m(m-1), n = (m+2)(m-1)$ . It is important to note that the ability of the complete codes to decode successfully bursts longer than their nominal length, deteriorates very rapidly. Simulations show that the fraction of burst patterns leading to failure in the decoding for the (151,136) code is 0.045, 0.145, 0.23 for  $l = 7, 8, 9$  respectively, and for the (217,202) code the fractions are 0.032, 0.105, 0.295 for  $l = 7, 8, 9$  respectively. Note that examining the bound developed by Gallager [13], there is a wide gap between the upper bound and the lower bound, so there is a possibility to make good incomplete cyclic codes, but it is still an open problem how to design one.

TABLE I

A COMPARISON BETWEEN THE CORRECTION CAPABILITIES OF EVEN-ODD CODES AND OPTIMAL BURST ERROR CORRECTING CODES

Case	Our Code	Optimal Code
1	$m=17$ , the code is (304,272), can correct 16 bits bursts	(151×2=302,136×2=272) can correct only 12 bits
2	$m=25$ , (648,600) can correct 24 bits	(217×3=651,202×3=606) can correct only 18 bits
3	$m=33$ , (1120,1056) can correct 32 bits	(1085,1020) can correct only 25 bits

This original code corrects all errors if they span one column only in a block. However, when used as a binary non-phased burst error correcting code, the burst may spread over two consecutive columns, while maintaining a length of up to one column. In this paper we will show that this code is useful also

<sup>1</sup>Within close proximity to the Reiger bound

for handling errors of this type, show a new decoding algorithm and analyze the probability of success in correcting non-phased bursts.

Another advantage of the algorithm is its simplicity. Both the encoder and the decoder mainly need word XOR operations, which can be performed within one cycle of most microprocessors, and cyclic shift operations that are also one cycle operations per word size. The most efficient choice for  $m$  is one more than the CPU word size. Using this value for  $m$ , one cycle ROR (in assembly – ROTate Right through carry) accomplishes the cyclic shift. The number of calculations in the encoder or in the decoder is in order of  $m^2$ . The number of calculations for a Reed-Solomon code of comparable size is in the order of  $m^3$ . Furthermore, the Reed Solomon algorithm is much more complex and requires far more processing power when implemented in software. The complexity of the new algorithm is comparable to or even lower than that of cyclic codes using the error trapping decoder, depending on the order of the generator polynomial. There is about  $m^2$  polynomial divisions needed ( $m^2$  is the code length for the same block size), where the complexity for each polynomial division is proportional to its order.

## II. THE EVEN-ODD SCHEME

The encoding scheme is taken from the original paper [2] and need not be changed for the new application. Assume there are  $m + 2$  disks, the first  $m$  disks contain the data and the other two are redundant. In the new application, a disk represents a column of bits in the array. If  $m$  is a prime number, the correction of the phased burst is guaranteed. However, even for non-prime  $m$  the correction failure probability (in short ‘fail probability’) is not larger than the typical fail probability of the new algorithm, so non-prime values of  $m$  are also considered. To simplify the presentation, we assume that each of the  $m$  disks has  $m - 1$  symbols (sectors) of information on it. The symbol’s size can vary from one bit to any size whatsoever. However, for the new decoding algorithm the symbol is required to be binary.

Let  $\{a_{i,j}\}, i = 0, \dots, m-2, j = 0, \dots, m-1$ , be an array of  $m(m-1)$  information symbols. Now we define the encoding method. The first step is to append a row of zeros to the end of the array, so now data symbols are indexed 0 to  $m-2$ , and  $a_{m-1,i} = 0$ . These zero bits are not transmitted, but are defined for convenience. The parity columns indexed  $m$  and  $m+1$  are defined as follows:

$$s = \bigoplus_{i=0}^{m-1} a_{m-i-1,i} \quad (1)$$

$$a_{j,m} = \bigoplus_{i=0}^{m-1} a_{j,i} \quad j = 0, \dots, m-1 \quad (2)$$

$$a_{j,m+1} = s \oplus \left( \bigoplus_{i=0}^{m-1} a_{\langle j-i \rangle_m, i} \right) \quad j = 0, \dots, m-1, \quad (3)$$

where  $\langle i \rangle_m = i \bmod m$ , and  $\oplus$  denotes XOR. Equations (1)–(3) define the two redundant disks, where disk  $m$  is a simple XOR disk, in which each sector is a XOR of all corresponding sectors in the data disk. Disk  $m+1$  is calculated using a diagonal XOR of the data disks’ sectors. The temporary variable  $s$  is the parity bit of the diagonal and is not transmitted.

## III. THE NOVEL SCHEME

In this paper we suggest a change in the scheme that modifies it for use in a communication channel with burst noise. Let  $P0$  denote the  $m^{\text{th}}$  column of  $\{a_{i,j}\}$  and  $P1$  shall denote the  $m+1^{\text{st}}$  column of  $\{a_{i,j}\}$ , i.e. the parity columns. We shall rearrange the columns to be transmitted in the order  $P0, 0, 1, \dots, m-1, P1$ . The bits in each column are transmitted from  $m-2$  to 0. We are interested in correcting bursts of errors with a length of not more than one column (i.e.  $m-1$  bits). The burst can start at column  $p$  row  $q-1$ , and end (at most) at column  $p+1$  row  $q$ . In this case the burst spans column  $p$  rows  $0, \dots, q-1$  and column  $p+1$  rows  $q, \dots, m-2$ . The parity  $P0$  was placed first so that one burst will not affect both parities, resulting in high failure rates. In fact,  $P0$  can be placed anywhere in the block as long as it is not adjacent to  $P1$ , without changing the fail probability. The information that we have to retrieve from the syndromes of the received array is as follows:

- 1) The rows which contain an error. This information is found simply in the syndrome vector  $S0$ .
- 2) The number of the column in which the errors start, denoted by  $p$ .
- 3) The portion of the burst that is in the first of the two columns, denoted by  $q$ . For example, if the length of the column is  $m = 11$ , and the error burst span bits 0–5 of column  $p$  and bits 6–9 in column  $p+1$ , then  $q = 6$ . The phased error case will be the special case of  $q = 0$ , where the whole burst is within the column  $p+1$ .

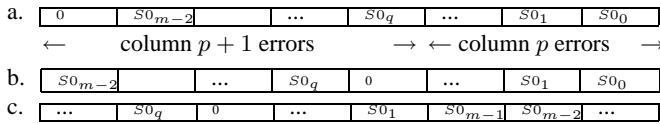
In the original phased error decoder a zero bit is added to all the data columns and a cyclic rotation is performed on the syndrome  $S0$  until it is equal or complementary to syndrome  $S1$ . The number of the column in which the errors start is determined by the number of rotations. Similarly, in the new algorithm we use a cyclic rotation operation in order to locate the column number. However, in order to match  $S0$  and  $S1$ , the cyclic rotation must be preceded by a correction in  $S1$ .

If columns  $p$  and  $p+1$  contain errors, it is easy to show that the  $S1$  syndrome is the union of the error pattern of column  $p$  shifted left  $p$  times<sup>2</sup>, and the error pattern of column  $p+1$  shifted left  $p+1$  times. The error pattern found in  $S1$ , which may be inverted (depending on the value of the  $s$  bit) is obtained by two steps. In the first step we insert a ‘0’ at position  $q$  of  $S0$ , and left-shift the bits starting from this position. We define this operation as  $T_q(x_0, x_1, \dots, x_{m-2}, 0) = (x_0, x_1, \dots, x_{q-1}, 0, x_q, \dots, x_{m-2})$ . In the second step the pattern is rotated  $p$  cyclic left shifts. These two steps are shown in Fig. 1, wherein each line in the figure shows the syndrome column as it is modified through the operations above. Fig. 1a shows the  $S0$  syndrome in which bits of value ‘1’ represent rows in error. The vector after the zero insertion is shown in Fig. 1b, and after rotation in Fig. 1c.

The first step of the decoding algorithm is to distinguish between three possible cases:

- 1) The number of ‘1’s in  $S0$  is equal to that of  $S1$ .
- 2) The sum of the number of ‘1’s in  $S0$  and the number of ‘1’s in  $S1$  is  $m$ .

<sup>2</sup>Before shifting left or right one has to transpose the column vector to be a row, and this row is placed in a shift register msb-left, lsb-right


Fig. 1. Generation of  $S1$  from  $S0$ 

3) Neither of the above.

In the first case we proceed to the matching process. The second case indicates that  $S1$  is inverted ( $s = 1$ ) and therefore needs to be inverted before the matching process. In the third case we conclude that the parity columns themselves were corrupted which requires special treatment as follows. We test for the case of a corrupted  $S1$  by rotating it once to the left, and checking if the bits of  $S0$  and  $(S1$  or  $\bar{1} \oplus S1)$  match. This match is performed from the first '1' in  $S0$  (going from bit  $m - 1$  to bit 0). We test for the case of a corrupted  $S0$  by similarly checking if the bits of  $S0$  and  $(S1$  or  $\bar{1} \oplus S1)$  match, starting from the first bit that  $S1$  is '1' going in the opposite direction (from bit 0 to bit  $m - 2$ ), where the value of bit  $m - 1$  of  $S1$  indicates whether  $S1$  needs to be inverted prior to the comparison. The algorithm may be summarized as follows:

- 1) Compute the syndromes and check if there is no error by checking if  $S0 = \bar{0}$  or  $S1 = \bar{0}$  or  $S1 = \bar{1}$ . Otherwise, continue.
- 2) Count the number of '1's in  $S0$  and  $S1$ , and determine whether they are equal or add up to  $m$ . If one of these conditions is satisfied then skip to step 4. Otherwise either the burst corrupted  $S0$  and the first data column, or it corrupted the last column and part of  $S1$ .
- 3) Correct the data according to the corrupted parity case (as explained above) and quit.
- 4) If the sum of the number of '1's in  $S0$  and the number of '1's in  $S1$  is  $m$  then invert  $S1$ .
- 5) Scan all the possibilities of  $p$  and  $q$ , apply  $T_q(x)$  and  $p$  left cyclic rotations on  $S0$  until  $S1$  is equal or complementary to  $S0$ .
- 6) correct the data using  $p, q$  and  $S0$ .

The algorithm's main task, step 5, is summarized in Fig. 2. Note that various optimizations for complexity reduction can be applied according to implementation method resulting in a very efficient and compact implementation. For example, the inner loop can be eliminated: In order to compare two words where part of one word is shifted once, we look for the first difference between the two words and then shift the word and compare again, now start from the bit there was a difference in the first comparison. The number of operations is now in the order of  $m^2$  (i.e.  $O(m)$  word comparisons).

#### IV. AN UPPER BOUND ON THE FAILURE RATE

The decoding failure is when the decoder finds incorrect  $(p, q)$  pair, which satisfies the matching conditions. Each such failure leads to erroneous correction and therefore corrupted block. The failure probability depends on the bursts statistics. We assume that burst position is randomly chosen within the array, burst length is  $m - 1$  and 50% bit error probability within

```

for p = 0 to m - 1
  for q = 0 to m - 2
    S0_NEW = R_p(T_q(S0))
    if {S0_NEW = S1 or S0_NEW = (\bar{1} \oplus S1)} stop
  end
end
end

```

Fig. 2. The core of the new decoding algorithm

the burst. In a more realistic situation the burst length is exponentially distributed. We will later modify the analysis and give results for shorter bursts, and then any distribution can be evaluated.

We assume that only data columns are corrupted. We will add the case of errors in the parity columns later. For simplifying the analysis, we will treat the case of burst starting in the last data column as continuing cyclically to the first one. Therefore all possible values for  $(p, q)$  are allowed. If  $m$  is chosen to be odd, there cannot be confusion between a pattern and its inverse (the inverse occurs when  $s = 1$ ). For even  $m$  this case should also be added to the analysis.

The code is a linear binary block code. The length of  $S0$  is  $m$  bits and its first term is always 0. Since the possibility of  $S0$  being all zeros is irrelevant (in this case, there are no errors), then there are  $2^{m-1} - 1$  relevant vectors that define the vector space of  $S0$ . If  $(p, q)$  be the location of an error burst, the condition  $T_q(R_p(S0)) = (S1$  or  $\bar{1} \oplus S1)$  will be called a match. In the following Lemma we will prove that the probability that there is more than one match is not dependent on the burst start location. If we assume a random search order when looking for a match, then the decoder has uniform fail probability in terms of burst location.

Let us call the location  $(p = m - 1, q = 0)$  the natural burst. We can assume without loss of generality [14] that the burst occurred in the natural place. For each  $(p, q)$  pair we like to find the burst patterns  $S0$  that if placed in the natural place or if places in burst start location  $(p, q)$  will generate the same  $S1$ . We can write a set of binary equations to find the number of possible solutions.

We take the vector  $S0 = (w_0, w_1, \dots, w_q, w_{q+1}, \dots, w_{m-1})$  and get the set of  $m$  equations with  $m$  variables

$$R_p(T_q(S0)) = S0. \quad (4)$$

In addition to this equation we also have  $w_{m-1} = 0$ . The rank of this equation set, denoted by  $\text{RANK}_{p,q}$ , and the rank of the vector space of the solutions to this set are summed up to  $m$ . If the rank is full, then there is only one solution: the zero vector. This means that there is no pattern that can be confused between the natural location and the burst start location  $(p, q)$ . For a different rank, the number of such patterns is

$$2^{m - \text{RANK}_{p,q}} - 1. \quad (5)$$

For example, in the array shown in Table II,  $m = 5, q = 1$  and  $p = 3$ .

TABLE II

DATA	DATA	DATA	$w_0$	DATA
DATA	DATA	DATA	DATA	$w_1$
DATA	DATA	DATA	DATA	$w_2$
DATA	DATA	DATA	DATA	$w_3$
0	0	0	0	0

The original vectors, before applying  $R_p$  and  $T_q$  are

$$S0 = (w_0, w_1, w_2, w_3, w_4), \quad (6)$$

$$S1 = (w_0, w_1, w_2, w_3, w_4). \quad (7)$$

The effect of  $q$  (which determines where the bits start in the second column) is shifting  $w_{m-1}$  to the  $q^{\text{th}}$  place and shifting every bit to its left, one space to the left<sup>3</sup>, giving us the next set

$$S0 = (w_0, w_1, w_2, w_3, w_4), \quad (8)$$

$$S1 = (w_0, w_4, w_1, w_2, w_3). \quad (9)$$

The effect of  $p$  is adding  $p$  cyclic left rotations, leaving us with the next set

$$S0 = (w_0, w_1, w_2, w_3, w_4), \quad (10)$$

$$S1 = (w_1, w_2, w_3, w_0, w_4). \quad (11)$$

This gives us the following set of equations:

$$w_0 = w_1; w_1 = w_2; w_2 = w_3; w_3 = w_0; w_4 = w_4. \quad (12)$$

Since the number of all the possibilities of  $S0$  is  $2^{m-1} - 1$ , and there is a probability of  $1/2$  that the true location will be encountered when the decoder search is independent of the burst location, then the error probability for a specific  $(p, q)$  pair is

$$P_{\text{fail}}(p, q) = \frac{1}{2} \frac{2^{m-\text{RANK}_{p,q}} - 1}{2^{m-1} - 1}. \quad (13)$$

Now we repeat the calculation for all pairs of  $(p, q)$  and use the union bound to upper bound the fail probability

$$P_{\text{fail}} \leq \sum_{p=1}^{m-2} \sum_{q=0}^{m-2} P_{\text{fail}}(p, q). \quad (14)$$

Note that the cases  $p = 0$  and  $p = m - 1$  were excluded since they represent images that are identical to the correct burst. A tighter bound will be obtained later in this section.

The next problem is to find an easy and fast way to calculate the rank of the equation sets. Since we are considering sets of  $m$  equations with  $m$  variables, the complexity of the solution is of order  $m^2$ . We present here a scheme that allows the calculation of the rank with a complexity of order  $m$ . The algorithm is described as follows.

We start with the equation that contains  $w_0$  in the right hand side. We take the variable from the left side of this equation and

<sup>3</sup>The direction left is defined as the direction of increasing indexes as common in binary representation of a number.

find the equation that has this variable in its right side. We take the variable from the left side and find again the equation that has it in the right side. We repeat this procedure until completing a cycle. For our example, the cycles are

$$w_0 \rightarrow w_3 \rightarrow w_2 \rightarrow w_1 \rightarrow w_0, \quad (15)$$

$$w_4 \rightarrow w_4. \quad (16)$$

If we did not traverse all the variables, we start another cycle at the first variable that we did not traverse. The rank is given by

$$\text{RANK}_{p,q} = m - L_{p,q} + 1, \quad (17)$$

where  $L_{p,q}$  is the number of cycles. The explanation is simple. Each cycle group variables that are equal and present one free variable in the solution. Therefore the number of cycles is the rank of the solution space. However, the cycle that contain the  $w_{m-1}$  is not free since  $w_{m-1} = 0$ , and so adding this limitation the rank of the solution space is (number of cycles) - 1. The rank of the equations set plus the rank of the solution space is the number of variables  $m$ .

Using the following Lemma we will now obtain a (much) tighter upper bound by deleting solutions that are completely included in other equation sets.

*Lemma 1:* If  $L_{p,q} > L_{p,q \pm 1}$  then any  $S0$  solving the equations for  $(p, q \pm 1)$  will solve also the set for  $(p, q)$ .

*Proof of Lemma 1:* One can verify that increasing or decreasing  $q$  by 1 switches the location of  $w_{m-1}$  with the adjacent variable. Now, if that variable belongs to a different loop than that containing  $w_{m-1}$ , switching the places links these two loops together to one loop, thus the rank is increased. Otherwise, the switching breaks the loop that contains  $w_{m-1}$  and that variable into two loops, thus the rank is decreased. In the first case, uniting the two loops causes the loop containing the  $w_{m-1}$  to increase, which means more variables are forced to be 0. As a result, any solution for the system with the united loop is also solution of the case where the loop is broken into two loops.  $\square$

Using this Lemma the local maxima of the rank in a column cover all solutions contained in its neighbors and these can be excluded from the union bound. The updated bound is

$$P_{\text{fail}} \leq \sum_{p=1}^{m-2} \sum_{q=0, L_{p,q} > L_{p,q+1} \& L_{p,q} > L_{p,q-1}}^{m-2} P_{\text{fail}}(p, q), \quad (18)$$

where the condition  $L_{p,q} > L_{p,q-1}$  is taken as true if  $q = 0$ .

#### A. Calculating A Lower Bound For The Scheme's Performance

The bound we present in this paragraph is a lower bound to the algorithm performance. The results show that the bound becomes tighter as  $m$  increases. Going through the cases in which the scheme cannot correct the error burst shows that in the majority of these cases there are special patterns for the  $S0$  vector, cases in which  $S0$  is divided to recurring patterns as shown in the following example.

Let's assume  $m = 11$  and  $S0 = (S0_0, \dots, S0_{m-1}) = (1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0)$ . We can see that this is the pattern

(1, 0, 1, 1, 1) is repeating itself, followed by the additional 0. If we choose  $q = 5$ , we move the zero after the first repetition and we get (1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1). Rotation to bring the zero to position 10 will cause the patterns to switch positions, but since they are equal, we get the same original vector. There are also patterns that are not cyclic and have images, so the equation is a lower bound. However, the results show that the ratio between the number of such patterns to the cyclic ones is diminishing for large  $m$ , so the lower bound tightens as  $m$  increases.

Since the  $m^{\text{th}}$  symbol of  $S_0$  is always 0, and that is the symbol that is moved to the  $q^{\text{th}}$  location in the vector, we can find two kinds of patterns:

- 1) Patterns that are periodic for the  $m - 1$  bits of  $S_0$ , requiring  $q > 1$  for creating images.
- 2) Patterns that are periodic for all the  $m$  bits of  $S_0$  (including the preceding zero), and then the symmetry is obtained using only cyclic rotations.

First, we find all the prime dividers of  $m - 1$ . All the non-prime divider cases are included in the prime dividers case with slightly higher probability. Using the prime dividers only is therefore a lower bound. Listing all the possibilities we obtain the lower bound

$$P_{\text{fail}} \geq \frac{\sum_j \frac{t_j - 1}{t_j} \left( 2^{\frac{m-1}{t_j}} - 1 \right)}{2^{m-1} - 1} + \frac{\sum_j \frac{N'_j - 1}{N'_j} \left( 2^{\frac{m}{N'_j}} - 1 \right)}{2^{m-1} - 1}, \quad (19)$$

where  $N_j$  are the prime dividers of  $m - 1$  and  $N'_j$  are the prime dividers of  $m$ . The value of  $N_j = 2$  will always exist in one of the summations, and lead to the highest term. We can bound the expression by

$$P_{\text{fail}} \geq 2^{\frac{-1-m}{2}}. \quad (20)$$

For large  $m$  this bound is tightening and becomes a good approximation since the ratio between the term resulting from  $N_j = 2$  to all the other terms is increasing. The results for all the bounds compared to simulation are shown in Fig. 3. Note that the results in which errors in the parity columns are included are lower since the fail probability when the parities are affected is lower than if the data only is affected as discussed shortly. Since the errors are placed uniformly less of the bursts ( $m/(m+2)$ ) are affecting the data, leading to overall (slight) decrease in the fail probability.

## V. CONCLUSION

We developed a simple decoding algorithm to correct non phased burst errors for this code. The code has low redundancy ( $l = (n - k)/2$ ), it is very simple, it is systematic, and it needs only  $O(m^2)$  operations to decode in a very easy to implement software or hardware. The scheme performance was given both with tight bounds and approximately in a simple closed form. This code provides a practical alternative to the well known burst error correcting codes. There are few cases for the error burst that are not covered by the derivations above, for example The scheme performance for short bursts, and require small modifications in the formula. These are treated in [14].

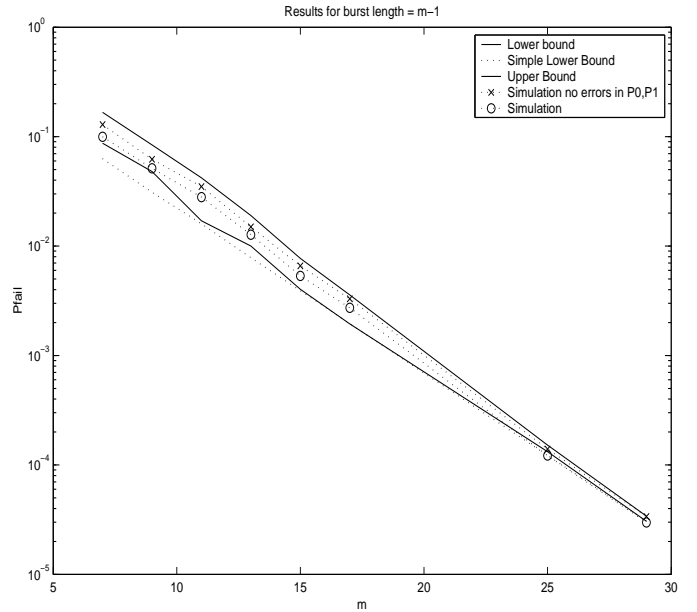


Fig. 3. Simulation with and without errors in the parity, lower bound (19) and simple lower bound (20) and upper bound (18)

## REFERENCES

- [1] J.G. Proakis : "DIGITAL COMMUNICATIONS", McGraw - Hill
- [2] M. Blaum, J. Brady, J. Bruck, J. Menon- "EVENODD : An Efficient Scheme for Tolerating Double Disk Failures in RAID Architectures", IEEE Trans. on Computers, vol. 44, February 1995, pp. 192 - 201.
- [3] F.J. MacWilliams and N.J.A. Sloane, "The Theory of Error - Correcting Codes", Amsterdam, The Netherlands: North Holland, 1977.
- [4] M. Blaum, "A Class of Byte - Correcting Array Codes", IBM Research Report, RJ 5652 (57151), May 1987.
- [5] M. Blaum and R. Roth, "New Array Codes for Multiple Phased Error Correction", IEEE Trans. inform. Theory, pp 66-77, Jan 1993.
- [6] R. Goodman and M. Sayano , "Size Limits on Phased Burst Error Correcting Array Codes", Electron. Lett., vol. 26, pp 55-56, 1990.
- [7] R. Goodman , R.J. McEliece and M. Sayano , "Phased Burst Correcting Array Codes", IEEE Trans. Inform. Theory, pp. 684-693, Mar. 1993.
- [8] S.Lin, D.J.Costello, Jr. , "Error Control coding: Fundamentals and Applications" , Prentice Hall, PTR.
- [9] W. Zhang, J. K. Wolf, "A Class of Binary Burst Error-Correcting Quasi-Cyclic Codes", IEEE Transactions on Information Theory, vol. 34, No. 3, May 1988, pp. 463-479
- [10] O. Keren, S. Litsyn, "Codes Correcting Phased Burst Errors", IEEE Trans. on Info. Theory , vol. 44, Jan. 1998, pp.416-420.
- [11] R. Roth, G. Seroussi, "Reduced-Redundancy Product Codes for Burst Error Correction," IEEE trans. Info. Theory, vol. 44, Jul. 1998, pp. 1395-1406.
- [12] M. Blaum, "A Family of Efficient Burst-Correcting Array Codes", IEEE Trans. on Info. Theory, vol. 36, no. 3, May 1990, pp. 671-675
- [13] R. G. Gallager: "Information Theory and Reliable Communication", Wiley, New York, 1968
- [14] D. Raphaeli, "The Burst Error Correcting Capabilities of a Simple Array Code", to appear in IEEE Trans. On Info. Theory