$e_s = c_s/\psi(x_s, y_s)$, and here $\psi(x_s, y_s) \neq 0$ by the choice of the points $P_1, \cdots, P_k$. Remark, that $\deg[\psi(x,y)] = p_2 < h$, by the assumption $j \geq (3/2)m - (3/2)$. If $t > t_0$ we simply add points with errorvalue zero to the previously stated construction. This concludes the proof of the theorem. □

We have used the Hermitian curve because the rational points on this are so easy to handle, but this is probably also the case for many other curves.

For a code $C^*(j)$ from a Hermitian curve, we have however more information in the decoding situation than the syndromes S_{ab}, $a + b \leq j$, and this can be used to get a minor improvement. This fact has no influence on the general results for the algorithm as previously described, but since the extra information is readily available in this specific situation we will make some comments about it.

From the curve equation $y^{r+1} - x^r - x = 0$ follows, in general, that

$$S_{a\,b+r+1} = S_{a+r\,b} + S_{a+1\,b}.$$

Therefore, when we are decoding a code $C^*(j)$, we know the syndromes $S_{a\,b}$, $a + b \leq j$ and $S_{0\,j+1}$, $S_{1\,j}, \cdots, S_{j-r\,r+1}$. Using all these syndromes as input to the algorithm one can realize, either by theoretical arguments or by experiments in concrete situations, that an error pattern as the one in Theorem 1 will be correctly decoded. To construct examples where the algorithm breaks down also with this extended input, one must change things a little.

We choose the error points in the same way as before, but such that the smallest degree h of an error locator satisfies

$$h = p_1 + p_2,\ p_2 = m - 3 - (j - 2h) - 1. \tag{3.11}$$

Let us now imagine, that we run the algorithm with all syndromes $S_{a\,b}$, $a+b \leq j+1$, as input. Then, with notation as above, because of (3.11) the rank of the matrix $\underline{F}_{j+1-h}$ is smaller that t (cf. Lemma 3). One can then, as before, find an error pattern for which the algorithm fails, and therefore of course the algorithm also fails if the input is the syndromes $S_{a\,b}$, $a + b \leq j$, and $S_{0\,j+1}, \cdots, S_{j-r\,r+1}$. To find the smallest number of points for which this construction is possible, we shall minimize an expression corresponding to (3.6). Carrying out the calculations one obtains

$$t_1 = \frac{d^*}{2} - \frac{m^2}{8} + \frac{3m}{4} - \frac{1}{8}, \tag{3.12}$$

which is a somewhat greater bound than (3.8). But the difference is not significant compared to the bound itself, and we will not discuss this problem further.

This situation can only occur if $m \geq 6$, so the smallest case in characteristic 2 is $r = 8$, which gives codes of length 504 over GF(64).

The bound (3.9) is the same as the bound (1.3), and hence the results in this correspondence shows, that the bound obtained in [2] in the general case is the optimal one for the method considered.

References

[1] J. Justesen, K.J. Larsen, A. Havemose, H.E. Jensen and T. Høholdt, "Construction and decoding of a class of algebraic geometry codes," *IEEE Trans. Inform. Theory*, vol. 35, pp. 811–821, July 1989.

[2] S. Sakata, "Extension of the Berlekamp–Massey algorithm to N dimensions," *Inform. Computat.*, vol. 84, no. 2, pp. 207–239, Feb. 1990.

[3] J. Justesen, K.J. Larsen, H. Jensen, and T. Høholdt, "Fast decoding of codes from algebraic plane curves," *IEEE Trans. Inform. Theory*, vol. 38, pp. 111–120, Jan. 1992.

[4] A.N. Skorobogatov and S.G. Vlădut, "On the decoding of algebraic-geometric codes," *IEEE Trans. Inform. Theory*, vol. 36, pp. 1051–1061, Sept. 1990.

[5] I.M. Duursma, "Algebraic decoding using special divisors," preprint, Eindhoven Univ. of Technol., The Netherlands, 1991.

Phased Burst Error-Correcting Array Codes

Rodney M. Goodman, Robert J. McEliece, and Masahiro Sayano

***Abstract*—Various aspects of single phased burst error-correcting array codes are explored. These codes are composed of two-dimensional arrays with row and column parities with a diagonally cyclic readout order; they are capable of correcting a single burst error along one diagonal. Optimal codeword sizes are found to have dimensions $n_1 \times n_2$ such that n_2 is the smallest prime number larger than n_1. These codes are capable of reaching the Singleton bound. A new type of error, approximate errors is defined; in q-ary applications, these errors cause data to be slightly corrupted and thererfore still close to the true data level. Phased burst array codes can be tailored to correct these codes with even higher rates than before.**

***Index Terms*—Error-correcting codes, array codes, phased burst correction, approximate errors.**

I. Introduction

In computer memory and communications applications, information can be corrupted by bursts of noise which occur within one of many predetermined sectors or time intervals. These noise patterns will be called phased burst errors [1] because although the noise pattern may be random at each burst, its duration and starting points are restricted to certain intervals. Noise sources which can generate these errors include line noise, synchronization errors in demodulation, timing errors in multivalued memories, and backscatter radar signals. These errors are often periodic in time (or, in the case of memories, in position) and can be long in duration. (See Fig. 1).

A motivation for studying this problem is the encoding of multilevel random access memories, where each cell contains more than one bit of data. These memories use dynamic RAM cells to store one of several discrete voltages. An experimental 4-Mbit chip with 16 possible voltage levels (4 bits worth of data) stored in each cell was reported in [2]. Voltage levels in each cell are stored and sensed by ramping the voltages on pertinent row and column select lines.

Manuscript received November 4, 1991. R.M. Goodman and M. Sayano are supported in part by NSF Grant MIP-8711568. R.J. McElliece was supported by AFOSR Grant 91-0037 and a grant from Pacific Bell. This work was presented in part at the IEEE/CAM Information Theory Workshop, Cornell University, Ithaca, NY, June 25–29, 1989, in part at the International Symposium on Information Theory, San Diego, CA, January 14–19, 1990, and in part at the IEEE International Symposium on Information Theory, Budapest, Hungary, June 24–28, 1991.

R.M. Goodman and R.J. McEliece are with The Department of Electrical Engineering, California Institute of Technology, Mail Code 116–81, Pasadena, CA 91125.

M. Sayano was with The Department of Electrical Engineering, California Institute of Technology, Mail Code 116–81, Pasadena, CA 91125. He is now with The Aerospace Corporation, El Segundo, CA 90009–2957.

IEEE Log Number 9204213.

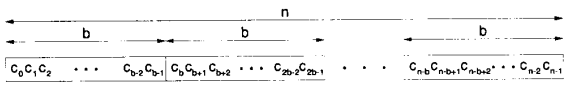

Fig. 1. Phased codeword. Errors can occur only within each b-symbol section.

These memories, unlike CCD type memories, are not serial, but have random access times which are much longer (~ 200 μsec) than binary RAM's and are therefore not optimal for working memory. However, since an entire row can be read at each access, blocks of data are available quickly; this, coupled with their high density, makes them ideal for mass storage devices.

The advantages of multilevel memories are evident when they are used as mass storage devices. Compared to magnetic hard disks, multilevel RAM's have faster block access times (~ 200 μs vs. ~ 20 ms), comparable throughput (20 Mb/s), and low power requirements (standby power of ~ 100 μw and active power of ~ 10 mW vs. standby and active power of ~ 1 W). Magnetic disks require mechanical drives that cannot be scaled downward with disk capacity; that is, for smaller disks, there is a significant "volume overhead" which decreases the storage density with respect to the entire disk package. Multilevel RAM's have no such penalty. Therefore, these memories are useful for replacing medium to small capacity magnetic hard disks where power is restricted, as in laptop computers.

The main drawback of multilevel RAMs, aside from the need for constant power to maintain memory contents, is the following. Since the cells contain charges separated by small discrete steps and since access timing accuracy is important, timing errors can cause errors in entire rows of memory. Error-correcting codes used at the chip level typically encode a single row as a codeword; these types of codes are thus useless in this case, where the entire row can be erroneous. In practice, large random access memory chips are broken up into a number of blocks. A code can be placed across these block lines to break up the long burst at the cost of increased access circuitry and encoder/decoders. The resulting codewords experience long phased bursts of errors in relatively short codewords. (See Fig. 2) A code which can correct long phased bursts with high rate and short codeword length is desired for this application.

Most well-known (or standard) burst error-correcting codes which can correct long bursts have extremely long codewords which must be shortened—and therefore result in a lower rate code—when used in applications with small block sizes. Fire codes, for example, have high rate at the expense of large codewords, and they do not take advantage of the phased nature of the error. Thus, standard burst error correcting codes such as the fire code may not be best for correcting phased bursts in practice, especially where a high-rate, high-speed, short codeword length code is needed. However, some codes, such as Reed–Solomon codes over serially arranged bit blocks, have high rate, can correct phased bursts, and are optimal. Here, array codes are presented as an alternative to the Reed–Solomon codes to correct single phased burst errors.

Array codes offer the advantages of block structure and easy encoding and decoding. The concept was first introduced by Elias [3]; the first array codes were Gilbert codes, developed in 1960 [4]. Gilbert codes are constructed from a two-dimensional array of cells with row and column parities, and their size can be varied greatly. Given a diagonal (helical) readout order on a rectangular array of size n_1 by n_2, with s being the diagonal skipping value, burst error correction is possible. An example is shown in Fig. 3.

Bounds on correctable burst lengths of Gilbert codes were studied by Neumann in 1965 [5]. Bahl and Chien in 1969 [6] made corrections to Neumann's work, and generalized the result for higher

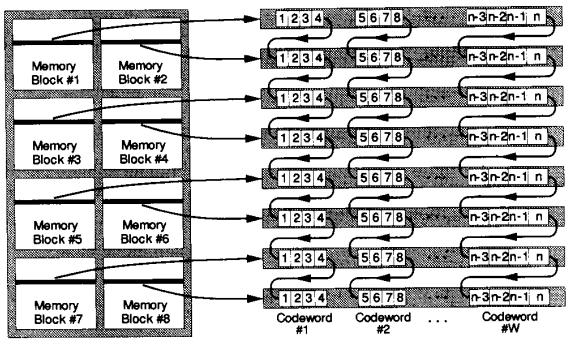

Fig. 2. Coding across multiple rows. This figure shows an example with eight blocks in a chip, N cells per row, $W = N/4$ codewords of size 8×4 with possible phased bursts of $b = 4$. Any one row in a block can be in error, and the code can correct this error.

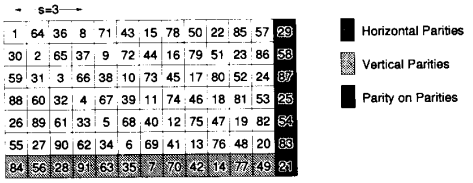

Fig. 3. Array codeword. Example is shown for $n_1 = 7$, $n_2 = 13$, and $s = 3$. Readout order as numbered.

dimensional Gilbert codes in 1971 [7]. Burton and Weldon also studied bounds on burst length [8]. Generalization of the Gilbert code to form burst error correcting array codes, along with the study of the special case for $s = 1$, was done by Farrell and Hopkins in 1982 [9]. In 1986, Blaum, *et al.* showed that for $b = n_1 - 1$, $n_2 \geq 2n_1 - 3$ is required [10]. Zhang and Wolf generalized the bound to find b for any n_1, n_2, and s in 1988 [11]. Further work by Blaum has shown that $b = n_1$ is attainable for $s = -1$ and $s = -2$ [12], [13]; these results were generalized and codes with efficiencies approaching unity were found by Zhang [14] and Sivarajan, McEliece, and van Tilborg [15].

Previous work concentrated mainly on general bursts, where burst starting location is not restricted. Under the limitation that all bursts occur only within one of many preselected blocks, array codes can be made extremely powerful and efficient without much difficulty, resulting in short codewords with long burst correcting powers. Phased burst error-correcting array codes are constructed, as are general burst error-correcting array codes, from two-dimensional arrays of cells with row and column simple parity checks and diagonally cyclic readout order. Note that the parity on parities (check on checks, in the rightmost bottom corner) can be considered to be both a vertical and a horizontal parity check. Errors are now restricted to occur only in a single diagonal; therefore, the readout parameter s [11], which denotes how many columns to skip before reading the next diagonal, is irrelevant here. Previous work on general burst error-correcting array codes made mention of s: The length of a correctable burst depends on s [11], [12]. For phased bursts, s can be any value so long as s and n_2 are mutually prime (which ensures that the entire array is filled).

Various properties of single phased burst error correcting array codes will be explored in this work. Section II contains the encoding and two possible decoding algorithms. Section III contains theorems on the allowed codeword sizes for phased burst correction. In Sections IV and V, optimal codeword sizes for this code are discussed. Section IV will cover array codes for the correction of

approximate errors, those errors in q-ary codes in which the erroneous value is restricted to be no more than Δ away from the true value. The Appendix contains proofs of all theorems.

II. Encoding and Decoding Strategies

Encoding information bits entails taking the parities of the rows and the columns as needed. For q-ary codes, these parities are taken modulo q, and is therefore an easy and fast operation. This can be implemented quickly and in parallel using XOR gates for $q = 2$ or their nonbinary equivalent, addition modulo q, for $q > 2$.

For decoding, the parities are first recomputed from the received codeword; from these the positions with parity violations (and in the case of nonbinary fields, the amount of the violation) are known. Consider the parity on parities to be both a horizontal as well as a vertical parity value. Becasue bursts are restricted to one diagonal, the hoizontal parities give the burst pattern. The vertical parities, given the burst pattern, provide information on the burst position. The horizontal parities are "stuffed" with $n_2 - n_1$ zeros so that both the horizontal and vertical parities are of length n_2. The vertical and horizontal parities can be represented as

$$V = [v_0 \quad v_1 \quad v_2 \quad \cdots \quad v_{n_2-1}]$$

and

$$H = [h_0 \quad h_1 \quad h_2 \cdots \quad h_{n_2-1}],$$

respectively. The problem now is finding the error position; that is, finding a value of e such that $0 \leq e < n_2$ and

$$[v_{e \bmod n_2} \quad v_{(e+1) \bmod n_2} \quad v_{(e+2) \bmod n_2} \quad \cdots \quad v_{(e+n_2-1) \bmod n_2}] = [h_0 \quad h_1 \quad h_2 \quad \cdots \quad h_{n_2-1}].$$

Once the error location is known, error correction is accomplished by XORing this erroneous row with the horizontal parities. (For q-ary applications, the horizontal parities are subtracted modulo n_2 from the erroneous row.)

A. Cyclic Convolution

One method for finding the error position e is cyclic convolution. This algorithm can be parallelized for extremely fast implementation. By cyclically convolving the two parity sets V and H, the error position can be found: The error location e occurs where the convolution yields a maximum. In both the binary and nonbinary case, this amounts to finding the number of cyclic shifts of the horizontal parities needed to match the pattern given by the vertical parities. The decoding algorithm can be implemented in parallel with n_2 convolution circuits, thus quickly yielding the position of the burst in two steps: First, calculate the cyclic convolutions in parallel; second, find the maximum value. This, however, requires a large amount of circuitry or computing nodes (n_2^2 computations). Serially implemented, this algorithm requires n_2^2 multiplications, $n_2^2 - n_2$ additions, and n_2 comparisons. In parallel, using n_2^2 computing nodes, the algorithm requires the time for one multiplication, $\log_2 n_2$ additions, and $\log_2 n_2$ comparisons in sequence.

B. Shiloach's Algorithm

One other method for finding e is using Shiloach's Algorithm [16], [17]. This is a serial implementation which requires, at most, $3(n_2 - 1)$ comparisons. Shiloach's Algorithm is much faster and requires less circuitry than a serially implemented cyclic convolution algorithm, but it is still slower than a parallel implementation of the cyclic convolution algorithm and cannot be effectively parallelized. An outline of Shiloach's algorithm follows. For two vectors A and B of length n defined as

$$A = [a_0 \quad a_1 \quad a_2 \quad \cdots \quad a_{n-1}],$$
$$B = [b_0 \quad b_1 \quad b_2 \quad \cdots \quad b_{n-1}],$$

two integers $i, j < n$ can be found such that

$$[a_{i \bmod n} \quad a_{(i+1) \bmod n} \quad a_{(i+2) \bmod n} \quad \cdots \quad a_{(i+n-1) \bmod n}] = [b_{j \bmod n} \quad b_{(j+1) \bmod n} \quad b_{(j+2) \bmod n} \quad \cdots \quad b_{(j+n-1) \bmod n}].$$

The algorithm returns $k = 0$ if no such match exists and $k = n$ if there is a valid match.

```
i = 0;
j = 0;
k = 0;
WHILE (i < n AND j < n AND k < n)
{   IF      (a_(i+k) mod n = b_(j+k) mod n)   k = k + 1;
    ELSE IF (a_(i+k) mod n < b_(j+k) mod n)   { i = i + k + 1;
                                                k = 0; }
    ELSE                                      { j = j + k + 1;
                                                k = 0; }
}
```

Implemented serially, this algorithm requires at most $3(n_2 - 1)$ comparisions and $3(n_2 - 1)$ additions.

III. Allowed Codeword Sizes

We have shown in [18] that such codes can always correct one phased burst of length n_1, the length of the vertical dimension, if $n_2 \geq 2n_1$, where n_2 is the length of the horizontal dimension, and that such codes can never correct one phased burst of length n_1 if $n_2 \leq n_1$. Furthermore, we have proven that such codes can correct one phased burst along a diagonal for $n_1 < n_2 \leq 2n_1$ with cyclic readout order, if and only if

$$n_2 \neq \frac{\alpha + 1}{\alpha}(n_1 - \beta), \tag{1}$$

where $\alpha \geq 1$ and $\beta \geq 1$. From these results, we have found codewords of short length and high rate which can correct single phased bursts. The theorem concentrated on filtering out those values of n_2 not allowed for a given value of n_1. Here another approach is used to obtain an equivalent result; this theorem regarding codeword sizes yields the largest possible n_1 for a given n_2.

Theorem 1: An array code with diagonal cyclic readout order can correct a single phased burst along a diagonal, if and only if

$$n_1 \leq n_2\left(1 - \frac{1}{\kappa}\right), \tag{2}$$

where κ is the smallest prime divisor of n_2.

Proof: See Appendix. □

Neumann presented a result very similar to (2) in [5] as the maximum general burst length correctable by an array. Specifically, the claim was made that given an array of size $n_1 \times n_2$, the maximum burst length correctable in a Gilbert code is

$$b = \min\left[n_1\left(1 - \frac{1}{\kappa_1}\right), \quad n_2\left(1 - \frac{1}{\kappa_2}\right)\right], \tag{3}$$

where κ_1 is the smallest prime divisor of n_1 and κ_2 is the smallest prime divisor of n_2. This was shown to be incorrect by Bahl and

TABLE I
ALLOWED CODEWORD SIZES FOR $n_1 < n_2 < 2n_2$

n_1	Allowed $n_2, n_2 < 2n_1$	n_1	Allowed $n_2, n_2 < 2n_1$
2	**3**	**22**	**23**,29,31,33,35,37,39,41,43
3	5	23	29,31,35,37,39,41,43,45
4	**5**,7	24	29,31,35,37,39,41,43,45,47
5	7,9	25	29,31,35,37,39,41,43,45,47,49
6	**7**,9,11	26	29,31,35,37,39,41,43,45,47,49,51
7	11,13	27	29,31,35,37,41,43,45,47,49,51,53
8	11,13,15	**28**	**29**,31,35,37,41,43,45,47,49,51,53,55
9	11,13,15,17	29	31,37,41,43,45,47,49,51,53,55,57
10	**11**,13,15,17,19	**30**	**31**,37,41,43,45,47,49,51,53,55,57,59
11	13,17,19,21	31	37,41,43,47,49,51,53,55,57,59,61
12	**13**,17,19,21,23	32	37,41,43,47,49,51,53,55,57,59,61,63
13	17,19,21,23,25	33	37,41,43,47,49,51,53,55,57,59,61,63,65
14	17,19,21,23,25,27	34	37,41,43,47,49,51,53,55,57,59,61,63,65,67
15	17,19,23,25,27,29	35	37,41,43,47,49,53,55,57,59,61,63,65,67,69
16	**17**,19,23,25,27,29,31	**36**	**37**,41,43,47,49,53,55,57,59,61,63,65,67,69,71
17	19,23,25,27,29,31,33	37	41,43,47,49,53,55,57,59,61,63,65,67,69,71,73
18	**19**,23,25,27,29,31,33,35	38	41,43,47,49,53,55,57,59,61,63,65,67,69,71,73,75
19	23,25,29,31,33,35,37	39	41,43,47,49,53,55,59,61,63,65,67,69,71,73,75,77
20	23,25,29,31,33,35,37,39	**40**	**41**,43,47,49,53,55,59,61,63,65,67,69,71,73,75,77,79
21	23,29,31,33,35,37,39,41	41	43,47,49,53,55,59,61,63,65,67,69,71,73,75,77,79,81

Optimal codes are in bold type, for which the smallest n_2 is prime.

Chien [6]. For phased bursts, Neumann's results are not applicable; therefore, this is a new result.

Theorem 1 is equivalent to the result of [18]. Since an array code is phased burst correcting, if and only if (1) holds, solving for n_1 yields

$$n_1 \neq n_2 \frac{\alpha}{\alpha+1} + \beta, \qquad \text{where } \alpha \geq 1 \text{ and } \beta \geq 1. \tag{4}$$

as a necessary and sufficient condition equivalent to (1). Since $\beta \geq 1$, the right side of (4) can increase without limit; therefore, (4) is equivalent to

$$n_1 \leq n_2 \frac{\alpha}{\alpha+1}, \qquad \text{where } \alpha \geq 1. \tag{5}$$

Since the right side of (5) must be an integer, $\alpha + 1$ must be a factor of n_2. The right side is small when $\alpha + 1$ is small; therefore, the right side is smallest when $\alpha + 1$ is the smallest factor of n_2, which is equivalent to the smallest prime factor of n_2, which was defined earlier as κ. Substitution of κ for $\alpha + 1$ in (5) results in (2); therefore, (1) and (2) are equivalent.

Special cases of Theorem 1 present properties of allowable n_2 for any n_1. The first two were mentioned in [18] but were not obvious from (1).

Corollary 1: If $n_2 \geq 2n_1$, then the burst error-correcting array code can always correct a phased burst of errors of length n_1.

Proof: See Appendix. □

Corollary 2: If $n_1 \geq n_2$ the burst error-correcting array code cannot correct a phased burst of errors of length n_1.

Proof: See Appendix. □

Corollary 3: For any n_1, the set allowed values of n_2 for the single phased burst error-correcting code includes all prime numbers greater than n_1.

Proof: See Appendix. □

IV. MEETING THE SINGLETON BOUND

Burst error-correcting codes are measured in optimality by efficiency rather than by rate. However, single phased burst error-correcting codes can be considered not as being burst error-correcting, since these phased bursts are effectively symbols taken in a serial bit stream, but as being single error-correcting codes. Therefore, the rate is the measure of optimality.

Linear error-correcting codes can reach but not exceed the Singleton bound, which states that to correct t errors, a code must have at least $2t$ parity symbols:

$$2t \leq n - k.$$

The code is capable of correcting a single phased burst, so $t = 1$. When n_2 is prime, using Corollary 3,

$$n_2 \geq n_1 + 1.$$

Take $n_2 = n_1 + 1$, so that the array size which is as square as possible. This should provide a codeword with as high rate as possible [19]. Then, taking each symbol to be a group of bits n_1 long,

$$\begin{aligned} n - k &= \left\lceil \frac{n_1 n_2}{n_1} \right\rceil - \left\lceil \frac{(n_1 - 1)(n_2 - 1)}{n_1} \right\rceil \\ &= [n_2] - \left\lceil \frac{n_1 n_2 - n_1 - n_2 + 1}{n_1} \right\rceil \\ &= n_2 - n_2 + \frac{n_1 + (n_1 + 1) - 1}{n_1} \\ &= 2. \end{aligned}$$

Therefore, single phased burst array codes can be optimal in the sense that they reach the Singleton bound; this occurs when n_2 is prime and $n_1 = n_2 - 1$. This was known previously [20] and is confirmed as a special case of Theorem 1.

Table I lists some allowed codeword sizes for single phased burst error-correcting array codes; optimal codeword sizes are in bold face type. Additionally, note that the smallest n_2 allowed for a given n_1 is the smallest prime number larger than n_1. This claim will be shown to be true for all $n_2 < 10^7$ in the next section; to prove the claim true for all n_2 is an unsolved number theory problem.

V. OPTIMAL CODEWORD SIZES

Table I hints at the claim made here, that for any n_1, the smallest allowed n_2 is the smallest prime number which is still greater than n_1.

Though this claim cannot be proven outright, it can be given strong supporting evidence. In this section, a theorem which, coupled with examination of all prime numbers less than 10^7, proves the claim for all $n_2 < 10^7$, will be presented. Since practical block lengths are much smaller than 10^7, the claim is proven for all practical values of n_2 and n_1.

The claim that the smallest n_2 for any given n_1 is the smallest prime number greater than n_1 will be supported in two steps. First, a theorem will be proven showing that if the difference between the two dimensions $n_2 - n_1$ is less than the square root of the larger number, the original claim is true. In the worst case, this difference between the two dimensions will be the difference between two consecutive prime numbers. Second, this theorem will be applied to all prime numbers below 10^7; the claim will be shown to hold true for these numbers. These two will help show that in the finite range $0 < n_2 < 10^7$ the claim is true, and that the claim in general relies on an unsolved problem in number theory.

Theorem 2: The smallest allowed n_2 for a given value of n_1 is the smallest prime number larger than n_1 if

$$n_2 - n_1 < \sqrt{n_2}. \tag{6}$$

Proof: See Appendix. □

Since the worst case (largest) difference between n_1 and n_2 occurs when both n_1 and n_2 are primes, all prime numbers below 10^7 were checked; this revealed that (6) holds for all pairs of consecutive primes except the pair 113 and 127. Here, $n_2 - n_1 < 14$ is possible while $\sqrt{n_2} = \sqrt{127} = 11.27$. Because Theorem 2 covers only cases where $n_2 - n_1 < \sqrt{n_2}$, n_2 is covered only in the range $113 < n_2 \leq 124$. Thus, the cases $n_2 = 125$ and $n_2 = 126$ were checked manually using (2),

$$n_1 \leq n_2\left(1 - \frac{1}{\kappa}\right).$$

When $n_2 = 125$, $n_1 \leq 100$ is required, and when $n_2 = 126$, $n_1 \leq 63$ is required for the code to be phased burst correcting. Since $n_1 = 113$, these two are not valid codewords. Therefore, even in the special case where (6) fails, the claim holds true for all prime numbers less than 10^7. Since most codewords are smaller than 10^7, the analysis thus far is usually sufficient for practical codeword sizes.

Theorem 2 suggests that if all consecutive primes p_n and p_{n+1} are separated by a distance $d_n = p_{n+1} - p_n$ which is less than $\sqrt{p_{n+1}}$, then the claim regarding phased burst array code sizes, namely, that the smallest n_2 allowed is the smallest prime number greater than n_1, is true. This is an unsolved problem in number theory. The best results obtained so far are close but insufficient [21],

$$d_n = O\left(p_n^{\theta}\right), \qquad \text{where } \theta = \frac{11}{20} - \frac{1}{384} \approx 0.5474.$$

However, since the theorem has been proven for all n_2 less than 10^7, this claim can be assumed true for all practical purposes: Almost all applications have codeword sizes of less than 10^7.

VI. Correcting Approximate Errors

In some analog memory and communications applications, the codeword contains q-ary symbols, that is, more than one bit of information is stored in a single memory cell (16-level RAMs [2], for example) or more than one bit is sent per use of the channel through the use of many discrete analog values to represent these q-ary symbols (amplitude and phase modulation in high-speed modems, for example). If these codewords are subject to errors which only change the received/retrieved symbol slightly from the transmitted/stored symbol, the information which remains in these cells can be used to help correct errors. For example, in multi-level memories, as mentioned in Section II-A, if there are timing errors in reading or writing the row of cells, the contents of that entire row will possibly be close to but in error from the true value. This information can be used to help correct the codeword.

Another example is in PSK (phase shift keying) modulation. Here, the symbols are a ring of size q, and the most common errors will be those closest to the correct value. Drifting from the correct sync value will cause errors which are close to the correct value. This condition can be corrected the next time the signal is synchronized, and synchronization is typically done either continuously or at regular intervals. The second case, used more often for high-speed applications, can result in time intervals with incorrect sync which are corrected by the next interval, resulting in bursts which are phased. FSK (frequency shift keying) has a similar problem in that the correct frequency may be mistaken for one close to that frequency; therefore, a code which is tailored to correct these most common errors with as high rate as possible will be useful.

In this section, an approach using the remnant information in corrupted symbols to correct these errors—and therefore use less redundant symbols in the encoding—will be explored. This type of error will be referred to as approximate errors, or those with erroneous values which are always approximately equal to the actual values. (This situation can be considered to be one with a skewed error distribution, since the errors are restricted to a certain class. Bitwise errors of this type and codes to handle them were studied in [22]; here a different approach is used.) If the actual value stored in a given symbol were x and if the symbol were in error, the resulting received/retrieved symbol will be between $x - \Delta$ and $x + \Delta$, where $\Delta < q/2$ and q is the number of values each symbol can take. The symbols can be considered to be a ring of size q or can be terminated at the mimimum and maximum.

A. Parity Values for Approximate Errors

The errors addressed in this section have magnitudes within Δ of the correct values. Since the errors are of limited severity, the parities need not be taken modulo q, as they were before. To correct errors of magnitude Δ, it is necessary to take parities modulo $2\Delta + 1$ to be able to distinguish between the $2\Delta + 1$ values which the error can take. (This includes one value for zero, the no error case.) However, it is not necessary to take two orthogonal sets of parities modulo $2\Delta + 1$. The two sets of parities in the phased burst error correcting array code contribute in a distinct manner, with the horizontal set of parities determining error pattern and vertical parities determining error position with the help of the horizontal parities. The horizontal parities, therefore, must be taken modulo $2\Delta+1$ to uniquely represent the error pattern. Determining error position, however, can be done by knowing only the existence or nonexistence of an error at each horizontal and vertical parity position. This can be accomplished by taking the parity modulo $\Delta + 1$, to distinguish between the Δ levels (including the no error case) of error possible.

Therefore, it is sufficient to record only the parities taken modulo $\Delta+1$ for the vertical parities to determine error location and modulo $2\Delta+1$ for the hoizontal parities to determine error value. The vertical parities, because there are more of them, are taken with $\Delta+1$ so that less space is taken with them. The parity on parities is taken modulo $2\Delta + 1$ so that it may be used as a horizontal parity value.

Initially only the approximate error correcting case will be explored; later, parity cell compression will also be included in the analysis. The encoding format is the same as before, but with the parities taken with differing modulo sums. Decoding is also as before, with the following exceptions: First, the parities which were taken

modulo $2\Delta + 1$ need to be recomputed for modulo $\Delta + 1$. These parities are then used to locate the error burst in the manner described above. Second, the burst is then corrected using the modulo $2\Delta + 1$ parities.

Error correction is accomplished as follows. The magnitude of the error is limited, $|e| < \Delta$. Define $\delta = 2\Delta + 1$. The horizontal parities were originally computed so that for each row i,

$$\left[\sum_{j=0}^{n_2-1} c_{ij} + h_i\right] \bmod \delta = 0.$$

where c_{ij} represents the contents of the jth cell in row i, and h_i is the horizontal parity of row i. Suppose an error of e occurs in cell c_{ik}. If r_{ij} is the received (and possibly erroneous) version of c_{ij}, then

$$\left[\sum_{j=0}^{n_2-1} r_{ij} + h_i\right] \bmod \delta = e \bmod \delta$$
$$= z.$$

Since values of $e < 0$ are mapped onto the region $\Delta + 1$ to 2Δ, the following correction rules apply:

$$c_{ik} = \begin{cases} r_{ik} - z, & \text{if } 0 < z \le \Delta. \\ r_{ik} - z + \delta, & \text{if } \Delta < z \le 2\Delta. \end{cases}$$

Thus, correction of approximate errors is possible with vertical parities taken modulo $\Delta + 1$ and horizontal parities taken modulo $2\Delta + 1$.

B. Parity Cell Compression

Parity cell compression for phased burst error correcting array codes is achieved as follows. The vertical parities are taken modulo $\Delta + 1$, and the horizontal parities, modulo δ. Since only a few bits of information are needed to record both sets of parities, there is some unused capacity for storing information in these parity symbols. These can be taken advantage of by storing more than one parity value in each symbol. There are two ways this can be accomplished: The parities can be compressed and represented as a column of length $n_1 - 1$, in which case this acts like another column added to the array of information symbols (without any parities), or they can be represented as a row of length $n_2 - 1$, in which case this acts like another row added to the array of information symbols.

In both cases, the parity on parities values is unused and is therefore not coded. Effectively, the code becomes a linear sum code [23]. This code can still correct a single phased burst using the information that the parity on parities is immune to error, a condition that occurs because the parity on parities is always assumed to be zero.

Note that this parity compression, therefore, is equivalent to removing a row or a column of parity values from the code and placing them in other cells. The error correction capacity of the code should remain unchanged; therefore, the code must be capable of correcting errors along any single diagonal, though the error magnitude must be within Δ for all values. This implies that the remaining column or row of parities can contain an error as well. However, because there is more than one parity value recorded into each symbol, the values must be coded in such a way that the additional parity values stored there are not affected, since these additional values record parity from rows or columns which differ from the location at which they are stored. Therefore, the requirements for parity compression are: Vertical parities are taken modulo $\Delta + 1$; horizontal parities are taken moludo $2\Delta + 1$; parities must be stored into $n_1 - 1$ or $n_2 - 1$ symbols; parity on parities is unneeded; and those additional parity values stored in a symbol must remain unaffected by errors within magnitude Δ.

The original parity value can be stored without coding since if it is erroneous, it can be corrected. This error can alter the value of the entire symbol by up to $\pm\Delta$. However, if a simple means of storing two parity values in the same symbol is used, a carry or borrow error can occur. That is, by storing the two parity values as

$$P = p_{\text{original}} + (\Delta + 1)p_{\text{additional}},$$

the two values can be found by

$$p_{\text{additional}} = \left\lfloor \frac{P}{\Delta + 1} \right\rfloor.$$
$$p_{\text{original}} = P \bmod(\Delta + 1).$$

However, if an error occurs in P, a carry or a borrow may occur and alter the values stored, causing an error to propagate into the value for $p_{\text{additional}}$:

$$\left\lfloor \frac{P}{\Delta + 1} \right\rfloor - 1 \le \left\lfloor \frac{P + e}{\Delta + 1} \right\rfloor \le \left\lfloor \frac{P}{\Delta + 1} \right\rfloor + 1.$$

The effect of a carry, borrow, or neither alters the resulting calculated $p_{\text{additional}}$ by $+1$, -1, or 0, respectively.

Thus, a buffer is needed between p_{original} and $p_{\text{additional}}$ that is large enough to distinguish between the carry, borrow, or neither carry nor borrow cases. The two cases, either having horizontal parities included into existing vertical parities or having vertical parities included into existing horizontal parities, will be addressed separately. The strategy for both are similar.

For horizontal parities included into existing vertical parities, consider the following. A horizontal parity value ($p_2 = h_i$) is included into an existing vertical parity value ($p_1 = v_j$) by

$$P = p_1 + \gamma p_2.$$

where γ is some constant. Let $Q = \gamma p_2$. Assume p_2 is known.

$$Q \le P \le Q + \Delta.$$

It is obvious that $\gamma > \Delta$. Now let an error of up to Δ in magnitude occur in P.

$$Q + e \le P + e \le Q + e + \Delta.$$

which, in the worst case (maximum range the values can span), is

$$Q - \Delta \le P + e \le Q + \Delta + \Delta. \tag{7}$$

To be uniquely able to distinguish between the case where a carry, a borrow, or neither occurs, there must be an unambiguous mapping given the worst case occurs. From (7), the maximum range which $P + e$ can span given p_2 is

$$\gamma - 1 = (Q + 2\Delta) - (Q - \Delta).$$
$$\gamma = 3\Delta + 1. \tag{8}$$

Therefore, to include a horizontal parity into an existing vertical parity,

$$P = v_j + (3\Delta + 1)h_i. \tag{9}$$

Fig. 4 outlines this analysis.

Decomposing these two parities into their original forms in the presence of errors is straightforward. The vertical parity in this case should reflect the error imposed upon the column.

$$p_1 = [P \bmod(3\Delta + 1)] \bmod(\Delta + 1). \tag{10}$$

The horizontal parity should remain unaffected by errors. Using.

$$p_2 = P \operatorname{div}(3\Delta + 1).$$

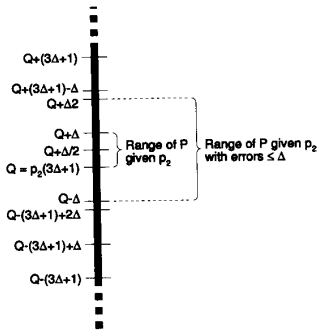

Fig. 4. Including horizontal parities into existing vertical parities. Range of possible values for P given p_2 is shown, along with range possible with errors of magnitude $\leq \Delta$. To avoid overlap, $\gamma = 3\Delta + 1$.

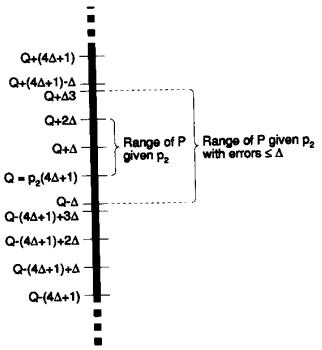

Fig. 5. Including vertical parities into existing horizontal parities. Range of possible values for P given p_2 is shown, along with range possible with errors of magnitude $\leq \Delta$. To avoid overlap, $\gamma = 4\Delta + 1$.

where $(a \text{ div } b)$ is the closest integer to a/b, can still be in error due to a possible carry: In general, $P + e \leq Q + 2\Delta$. If $P + e$ is larger than $Q + \gamma/2$, a carry error can occur. Since

$$Q + 2\Delta > Q + \frac{3\Delta + 1}{2} = Q + \frac{\gamma}{2},$$

an error can result.

Therefore, an offset μ must be used:

$$\mu + Q - \Delta > Q - \frac{3\Delta + 1}{2} \quad \text{and} \quad \mu + Q + 2\Delta < Q + \frac{3\Delta + 1}{2},$$

which becomes

$$\mu > -\frac{\Delta + 1}{2} \quad \text{and} \quad \mu < -\frac{\Delta - 1}{2}.$$

Therefore, if μ is even, then

$$p_2 = \left[P - \frac{\Delta}{2}\right] \text{div}(3\Delta + 1). \tag{11}$$

If μ is odd, then

$$p_2 = \left[P - \frac{\Delta + 1}{2}\right] \text{div}(3\Delta + 1). \tag{12}$$

Equation (12) assures that, using the usual technique for rounding off fractional values (i.e., $1.4 \rightarrow 1$ and $1.5 \rightarrow 2$), p_2 is found without error:

$$\left[Q - \Delta - \frac{\Delta + 1}{2}\right] = Q - \frac{3\Delta + 1}{2}$$

and

$$\left[Q + 2\Delta - \frac{\Delta + 1}{2}\right] = Q + \frac{3\Delta + 1}{2} - 1.$$

For vertical parities included into axisting horizontal parities, the strategy is very similar. A vertical parity value ($p_2 = v_j$) is included into an existing horizontal parity value ($p_1 = h_i$) by

$$P = p_1 + \gamma p_2,$$

where γ is some constant. Let $Q = \gamma p_2$. Assume p_2 is known.

$$Q \leq P < Q + \delta$$

or, equivalently,

$$Q \leq P \leq Q + 2\Delta.$$

Now let an error of up to Δ in magnitude occur in P.

$$Q + e \leq P + e \leq Q + e + 2\Delta,$$

which, in the worst case (maximum range the values can span), is

$$Q - \Delta \leq P + e \leq Q + \Delta + 2\Delta. \tag{13}$$

Equation (13) is similar to (7); the former differs from the latter only by a constant Δ on the right far side. To be uniquely able to distinguish between the case where a carry, a borrow, or neither occurs, there must be an unambiguous mapping given the worst case occurs. From (13), the maximum range which $P + e$ can span given p_2 is

$$\begin{aligned} Q - \Delta + \gamma - 1 &= Q + 3\Delta, \\ \gamma &= 4\Delta + 1, \end{aligned} \tag{14}$$

which is similar in form to (8). Therefore, to include a horizontal parity into an existing vertical parity,

$$P = h_i + (4\Delta + 1)v_j. \tag{15}$$

Fig. 5 outlines this analysis.

Decomposing these two parities into their original forms in the presence of errors is done as before. The horizontal parity in this case should reflect the error imposed upon the row.

$$p_1 = [P \bmod (4\Delta + 1)] \bmod (2\Delta + 1). \tag{16}$$

The vertical parity should remain unaffected by errors; again, an offset μ must be used:

$$\mu + Q - \Delta > Q - \frac{4\Delta + 1}{2} \quad \text{and} \quad \mu + Q + 3\Delta < Q + \frac{4\Delta + 1}{2},$$

which becomes

$$\mu > -\Delta + \frac{1}{2} \quad \text{and} \quad \mu < -\Delta - \frac{1}{2}.$$

Therefore, $\mu = \Delta$ and

$$p_2 = [P - \Delta] \text{ div}(4\Delta + 1). \tag{17}$$

C. Actual Compression Technique

When this codeword is actually implemented, it may not be efficient to attempt integer division and remainder (modulo) calculations. High-speed applications require the code to have as few operations as possible for encoding the decoding, and division operations are time intensive. Therefore, placing parity values into bit fields within the parity symbol may be advantageous: In this format, examining the bit fields is sufficient to obtain the necessary parity values.

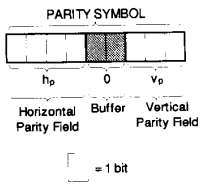

Fig. 6. Parity cell compression into bit fields, with horizontal parity included into vertical parity symbol. One parity symbol is shown, with parity values encoded into bits. Blank space of two bits is safety buffer to avoid carry-over errors into horizontal parity field.

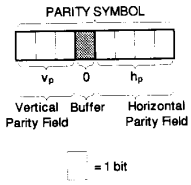

Fig. 7. Parity cell compression into bit fields, with vertical parity included into horizontal parity symbol. One parity symbol is shown, with parity values encoded into bits. Blank space of one bit is safety buffer to avoid carry-over errors into vertical parity field.

The best choice for Δ is $2^m - 1$; the vertical parities are taken modulo 2^m and the horizontal parities are taken modulo 2^{m+1}. The range of the horizontal parities is larger than necessary, but 2^{m+1} is used since it allows easy derivation of the modulo $\Delta + 1$ parity value for determining error location: The m least significant bits of the horizontal parity value provide this parity information.

To store more than one parity value into each symbol of redundancy, the strategy used in the previous subsection are applied. If horizontal parities are being stored in vertical parity symbols, then the minimum γ is $3\Delta + 1$; this translates into at least $m + 2$ bits, or a buffer of two blank unused bits between the vertical and horizontal parity value bit fields. (See Fig. 6.) These bit fields serve as flags as to when a carry or a borrow had occurred. Correction for carries and borrows are as follows:

buffer bitfield = 00 : no adjustments needed

buffer bitfield = 01 : no adjustments needed

buffer bitfield = 10 : this case will never occur

buffer bitfield = 11 : add 1 to horizontal parity field.

If vertical parities are being stored in horizontal parity symbols, then the minimum γ is $4\Delta + 1$; this translates into at least $m + 2$ bits, or a buffer of one blank unused bit between the horizontal and vertical parity value bit fields. (See Fig. 7). This one extra bit with additional information obtained from the value in the horizontal parity bit field provides carry or borrow information. Because the horizontal parity is between 0 and $2^{m+1} - 1$, inclusive, an error can only increase this value to $2^{m+1} + 2^m - 2$, which is represented by a carry of 1 and a residue of $2^m - 2$. Likewise, an error can only decrease this value to $-\Delta$, which is represented by a borrow of 1 and a residue of $2^m + 1$. Therefore, if a carry occurs and the residue is greater than or equal to $2^m + 1$, or $\Delta + 2$, then a borrow has occurred; if a carry occurs and the residue is less than or equal to $2^m - 2$, or $\Delta - 1$, then a carry has occurred. Therefore, when the buffer is nonzero, if the most significant bit of the horizontal bit field is 1, then a borrow has occurred; otherwise, a carry has occurred.

buffer bitfield = 0 : no adjustments needed

buffer bitfield = 1 and MSB of horizontal parity field

= 0 : no adjustments needed

buffer bitfield = 1 and MSB of horizontal parity field

= 1 : add 1 to vertical parity field

Note that for parity values which are inserted into existing parity symbols, there is no need for contiguous bit fields to be used. Therefore, these inserted parity values may be placed in any order to fit them in the available space. However, even if noncontiguous bit field packing is used, the magnitude of approximate errors correctable using this implementation is less than that possible by using the algorithm outlined in the previous subsection. The tradeoff made here is ease in encoding and decoding in favor of optimality of parity symbol usage. Thus, the horizontal and vertical parities are recorded in fewer symbols but in a manner which allows easy determination of parity values without integer division or remainder (modulo) calculations, resulting in an easily encodable and decodable code which has very high rate and can correct a phased burst of approximate errors.

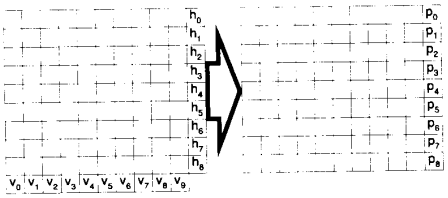

Fig. 8. Parity cell compression. Codeword on left is the standard 10×11 phased burst correcting array code; codeword on right is the 9×11 phased burst correcting array code correcting approximate errors of $\Delta \leq 7$ with parity cell compression. Parity values are compressed into fewer symbols.

D. Example

Consider two codewords, both capable of correcting a single phased burst of length 9 and approximate errors of size $\Delta = 7$ from symbols of size $q = 512$. The first is the standard phased burst array codeword of dimensions $n_1 = 9$, $n_2 = 11$; the second, the codeword with parity compression derived from the standard codeword of dimensions $n_1 = 10$, $n_2 = 11$. The latter is effectively a 9×11 codeword with 90 information symbols capable of correcting a burst of length 9 and of size $\Delta = 7$. The first has a rate of 0.81, with 80 information symbols, and is capable of correcting a single phased burst of length 9 and of any error magnitude. The second has a rate of 0.91 with 90 information symbols and is capable of correcting a single phased burst of length 9 so long as the error magnitude of each symbol does not exceed 7. (See Fig. 8).

Vertical parity values are placed in each horizontal parity symbol using every possible space; therefore, the 4 least significant bits are used to store the horizontal parities, while the most significant 4 bits are used to store the vertical parity values. The fifth most significant bit, in between the two bit fields, is used as a buffer. The vertical parity values are stored across the parity symbols such that four bits each from nine symbols are used to store ten three bit numbers. One possible arrangement is shown in Fig. 9; a total of six bits remain unused. (Note that a more optimal utilization of available parity symbol space would have resulted in $\Delta = 8$, but at greatly increased computational complexity.)

VII. Conclusion

Various aspects of the single phased burst error-correcting array code have been explored. The general solution for valid codeword

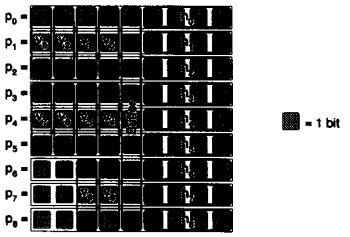

Fig. 9. Parity symbols shown bitwise. Each parity symbol is composed of 9 bits (for a total of 512 possible values per symbol). Bit mapping for each parity value is shown.

sizes to correct phased bursts of length n_1 was presented. The code can have dimensions such that the Singleton bound is met with equality and are therefore optimal. The minimum size for n_2, given n_1, was shown to be the next highest prime larger than n_1 for $n_2 < 10^7$; it is an unsolved number theory problem to prove this true for all n_1 and n_2. Two decoding algorithms were covered. Also, encoding and decoding strategies for approximate errors were presented, with means of compressing parities to further increase the rate of these codes.

Appendix

Theorem 1: An array code with diagonal cyclic readout order can correct a single phased burst along a diagonal, if and only if

$$n_1 \leq n_2\left(1 - \frac{1}{\kappa}\right), \tag{2}$$

where κ is the smallest prime divisor of n_2.

Proof: Consider the parity values to be represented by polynomials. The horizontal parity values $h_2, h_1, h_2, \cdots, h_{n_1-1}$ can be represented by

$$h(x) = \sum_{i=0}^{n_1-1} h_i x^i$$

and the vertical parity values $v_0, v_1, v_2, \cdots, v_{n_2-1}$ can be represented by

$$v(x) = \sum_{i=0}^{n_2-1} v_i x^i.$$

Since the error pattern appears in the horizontal parities, an error in the eth diagonal will result in

$$v(x) = x^e h(x) \qquad (\text{mod } x^{n_2} - 1).$$

The proof will be by contradiction. Given that (2) holds true, assume that erroneous decoding occurs. Erroneous decoding can occur if there exists another value d, which yields the same vertical syndrome. That is, for the code to *not* work,

$$x^e h(x) \equiv x^d h(x) \qquad (\text{mod } x^{n_2} - 1), \tag{18}$$

where $d \neq e$. Rearranging (18) yields

$$\left(x^k - 1\right)h(x) \equiv 0 \qquad (\text{mod } x^{n_2} - 1), \tag{19}$$

where $k = (e - d) \bmod n_2$, $k \neq 0$, as the condition for the code to not work.

Theorem 2.3 in [24] states that, for positive n and m,

$$\gcd(t^n - 1, t^m - 1) = t^{\gcd(n,m)} - 1.$$

Apply this theorem to (19): Call $\gcd(k, n_2) = \gamma$. Then, by factoring out the common term, the condition for the code to not work becomes

$$h(x) \equiv 0 \qquad \left(\text{mod } \frac{x^{n_2} - 1}{x^\gamma - 1}\right),$$

where $\gamma \neq 0$. Note that since $\gamma | n_2$, then $\gamma \leq n_2/\kappa$, where κ is the smallest prime divisor of n_2. Therefore,

$$\deg\left(\frac{x^{n_2} - 1}{x^\gamma - 1}\right) \geq n_2\left(1 - \frac{1}{\kappa}\right),$$

which then requires

$$\deg(h(x)) \geq n_2\left(1 - \frac{1}{\kappa}\right).$$

But since

$$\deg(h(x)) \leq n_1 - 1 \tag{20}$$

and from (2)

$$\deg(h(x)) \leq n_1 - 1 \leq n_2\left(1 - \frac{1}{\kappa}\right) - 1,$$

there is a contradiction, so erroneous decoding cannot occur when (2) holds.

The converse is also proven by contradiction. Given that erroneous decoding cannot occur, assume that (2) is not true. Consider the case where

$$n_1 \geq n_2\left(1 - \frac{1}{\kappa}\right) + 1,$$

the converse of (2). Then from (20) the condition for erroneous decoding can be met, and the error pattern

$$h(x) = \left(\frac{x^{n_2} - 1}{x^{n_2/\kappa} - 1}\right)$$

yields the same parity pattern for errors $h(x)$ and $x^\kappa h(x)$. Erroneous decoding can then occur, so when erroneous decoding does not occur, (2) must hold. □

Corollary 1: If $n_2 \geq 2n_1$, then the burst error-correcting array code can always correct a phased burst of errors of length n_1.

Proof: Consider the result of Theorem 1. To be phased burst correcting, an array must have the dimensions dictated by (2)

$$n_1 \leq n_2\left(1 - \frac{1}{\kappa}\right),$$

where κ is the smallest prime divisor of n_2. The smallest possible value for κ is $\kappa = 2$; therefore, the worst case for all possible values of n_2 will require that

$$n_1 \leq n_2\left(1 - \frac{1}{2}\right).$$

Thus,

$$n_2 \geq 2n_1. \qquad \square$$

Corollary 2: If $n_1 \geq n_2$ the burst error-correcting array code cannot correct a phased burst of errors of length n_1.

Proof: Consider the result of Theorem 1. To be phased burst correcting, an array must have the dimensions dictated by (2)

$$n_1 \leq n_2\left(1 - \frac{1}{\kappa}\right),$$

where κ is the smallest prime divisor of n_2. The largest possible value for κ is $\kappa = n_2$, or the case when n_2 is prime. Therefore, in the best case (when the array is as square as possible)

$$n_1 \leq n_2\left(1 - \frac{1}{n_2}\right)$$

is required. This simplifies to

$$n_2 \geq n_1 + 1.$$

Thus, the smallest that n_2 is allowed to be is $n_1 + 1$, and if n_2 is smaller than that, the array code cannot correct phased bursts of length n_1. □

Corollary 3: For any n_1, the set of allowed values of n_2 for the single phased burst error correcting code includes all prime numbers greater than n_1.

Proof: From (2),

$$n_1 \leq n_2\left(1 - \frac{1}{\kappa}\right).$$

If n_2 is prime, then because $\kappa = n_2$, (2) becomes $n_1 \leq n_2 - 1$. Thus, all prime numbers greater than n_1 are acceptable values of n_2. □

Theorem 2: The smallest allowed n_2 for a given value of n_1 is the smallest prime number larger than n_1 so long as $n_2 - n_2 < \sqrt{n_2}$.

Proof: From Corollary 2, $n_2 > n_1$; from Corollary 1, $n_2 < 2n_1$ is the region of interest. Therefore, the region of interest is $n_1 < n_2 < 2n_1$, because if $n_2 \leq n_1$ the code is guaranteed not to work, and if $n_2 \geq 2n_1$ the code is guaranteed to work. Also, from Corollary 3, all primes n_2 in the range $n_1 < n_2 < 2n_1$ generate acceptable codewords.

Choose $n_{2,p}$ to be the smallest prime number greater than n_1 and $n_{2,c}$ to be the composite (nonprime) numbers greater than n_1 and less than $n_{2,p}$. If (2) is to be satisfied,

$$n_1 \leq n_{2,c}\left(1 - \frac{1}{\kappa_c}\right), \tag{21}$$

where κ_c is the smallest prime factor of $n_{2,c}$. The smallest prime factor of any composite number is less than the square root of that number; therefore, for an acceptable $n_{2,c}$—where (21) holds—to exist,

$$\begin{aligned} n_1 &\leq n_{2,c}\left(1 - \frac{1}{\sqrt{n_{2,c}}}\right) \\ &\leq n_{2,c} - \sqrt{n_{2,c}}. \end{aligned} \tag{22}$$

For any $n_{2,c}$ such that $n_1 < n_{2,c} < n_{2,p}$, if (22) is true, then $n_{2,c}$ is a valid codeword dimension. Since the opposite is desired, that is, that there are no valid composite (nonprime) $n_{2,c}$ in the range $n_1 < n_{2,c} < n_{2,p}$, and since (21) is a direct consequence of Theorem 1, which gives a necessary and sufficient condition for acceptable values of n_1 and n_2,

$$n_1 > n_{2,c} - \sqrt{n_{2,c}},$$

which can be rearranged to obtain

$$n_{2,c} - n_1 < \sqrt{n_{2,c}} \tag{23}$$

as the condition required for no n_2 less than the smallest prime number larger than n_1 to be an acceptable codeword size. Since the largest that $n_{2,c}$ can be is $n_{2,p} - 1$, (23) becomes

$$(n_{2,p} - 1) - n_1 < \sqrt{n_{2,p} - 1},$$

which implies

$$n_{2,p} - n_1 < \sqrt{n_{2,p}}.$$ □

ACKNOWLEDGMENT

The authors would like to thank Dr. M. Blaum of IBM and Prof. P. G. Farrell of the University of Manchester for comments and suggestions.

REFERENCES

[1] S. Lin and D. J. Costello, *Error Control Coding: Fundamentals and Applications.* Englewood Cliffs, NJ: Prentice Hall, 1983.
[2] M. Horiguchi *et al.*, "An experimental large capacity semiconductor file memory using 16 levels cell storage," *IEEE J. Solid-State Circuits*, vol. SC-23, pp. 27–33, 1988.
[3] P. Elias, "Error-free coding," *Trans. IRE Professional Group Inform. Theory*, vol. IT-4, pp. 29–37, 1954.
[4] E. N. Gilbert, "A problem in binary encoding," *Proc. Symp. Appl. Math.*, vol. 10, pp. 291–297, 1960.
[5] P. G. Neumann, "A note on Gilbert burst-correcting codes," *IEEE Trans. Inform. Theory*, vol. IT-11, pp. 377–384, 1965.
[6] L. R. Bahl and R. T. Chien, "On Gilbert burst-error-correcting codes," *IEEE Trans. Inform. Theory*, vol. IT-15, pp. 431–433, 1969.
[7] ——, "Single- and multiple-burst-correcting properties of a class of cyclic product codes," *IEEE Trans. Inform. Theory*, vol. IT-17, pp. 594–600, 1971.
[8] H. O. Burton and E. J. Weldon, "Cyclic product codes," *IEEE Trans. Inform. Theory*, vol. IT-11, pp. 433–439, 1965.
[9] P. G. Farrell and S. J. Hopkins, "Burst-error-correcting array codes," *The Ratio and Electronic Engineer*, vol. 52, pp. 188–192, 1982.
[10] M. Blaum, P. G. Farrell, and H. C. A. van Tilborg, "A class of burst error-correcting array codes," *IEEE Trans. Inform. Theory*, vol. IT-32, pp. 836–839, Nov. 1986.
[11] W. Zhang and J. K. Wolf, "A class of binary burst error-correcting quasi-cyclic product codes," *IEEE Trans. Inform. Theory*, vol. 34, pp. 463–479, May 1988.
[12] M. Blaum, P. G. Farrell, and H. C. A. van Tilborg, "Multiple burst error-correcting array codes," *IEEE Trans. Inform. Theory*, vol. 34, pt. I, pp. 1061–1066, Sept. 1988.
[13] M. Blaum, "A family of efficient burst error-correcting array codes," IBM Internal Res. Rep. RJ6732, 1989.
[14] Z. Zhang, "Limiting efficiencies of burst-correcting array codes," *IEEE Trans. Inform. Theory*, vol. 37, pp. 976–981, July 1991.
[15] K. N. Sivarajan, R. J. McEliece, and H. C. A. van Tilborg, "Burst-error-correcting and detecting codes," *IEEE Int. Symp. Inform. Theory*, San Diego, CA, Jan. 14–19, 1990.
[16] Y. Shiloach, "A fast equivalence-checking algorithm for circular lists," *Inform. Processing Lett.*, vol. 8, pp. 236–238, 1979.
[17] M. Blaum and R. M. Roth, "New array codes for multiple phased burst correction," IBM Internal Res. Rep. RJ8303, 1991.
[18] R. M. Goodman and M. Sayano, "Size limits on phased burst error correcting array codes," *Electron. Lett.*, vol. 26, pp. 55–56, 1990.
[19] P. Calingaert, "Two-dimensional parity checking," *J. ACM*, vol. 8, pp. 186–200, 1961.
[20] M. Blaum, "A class of byte-correcting array codes," IBM Internal Res. Rep. RJ5652, 1987.
[21] P. Ribenboim, *The Book of Prime Number Records,* 2nd ed. New York: Springer-Verlag, 1989.
[22] T. Fuja and C. Heegard, "Focused codes for channels with skewed errors," *IEEE Trans. Inform. Theory*, vol. 36, pp. 773–783, 1991.
[23] T. Fuja, C. Heegard, and R. M. Goodman, "Linear sum codes for random access memories," *IEEE Trans. Comput.*, vol. 37, pp. 1030–1042, 1988.
[24] R. J. McEliece, *Finite Fields for Computer Scientists and Engineers.* Norwell, MA: Kluwer Academic, 1987.