PROTECTING CONVOLUTION-TYPE ARITMETIC ARRAY CALCULATIONS WITH GENERALIZED CYCLIC CODES

G. Robert Redinbo
Department of Electrical
and Computer Engineering
University of California
Davis, California 95616 USA

ABSTRACT

Fault-tolerance in dense high-speed arithmetic units that calculate convolutions between arrays of data is introduced through cyclic codes which are defined over the rings and fields commonly employed by such units. New systematic encoding and data manipulation techniques make the application of these generalized cyclic codes to error detection straightforward and efficient. The necessary overhead parity computations have complexity proportional to the number of parity symbols squared, whereas the error-detecting capability for both random and burst errors is directly related to this parity number.

INTRODUCTION

As digital electronic implementations of arithmetic units become more dense through shrinking VLSI technology and as their speed of operation increases, internal fault-tolerance will be needed within arithmetic systems. In particular, error protection against soft errors is a critical requirement. Erroneous calculations must be detected as close to their source and as soon as possible to avoid propagating their effects beyond certain hardware and data boundaries.

High-speed convolution of data arrays is one common and important class of arithmetic processing which needs adequate protection. Properly defined linear codes can protect the addition and scaling of data arrays. However, the convolution of such arrays, while constructed from basic simple operations, requires more structured cyclic error-correcting codes. This paper demonstrates how generalized cyclic codes, defined over the rings and fields usually employed in such processing, may be incorporated directly and quite naturally within the implementations of such arithmetic systems. Furthermore, new encoding and parity manipulation methods are developed which permit straightforward and efficient mechanizations. Errors are detected immediately at the conclusion of the processing pass, allowing appropriate error control actions may be initiated. Typical responses to detected errors may be to retry the calculation or to enter a subsystem testing mode.

The central operation in signal processing or digital filtering is the convolution of data sample sequences. These samples' arithmetic values may be viewed as algebraic elements in finite rings such as the integers modulo an integer q, denoted by Z_q, or as real or complex numbers, labeled respectively by R and C. Typical ring representations include two's complement, where $q = 2^m$ or one's complement with $q = 2^m-1$. The machine format can be either fixed or floating point.

The fundamental convolution operation may be displayed using mathematical symbols, starting from the two data sequences.

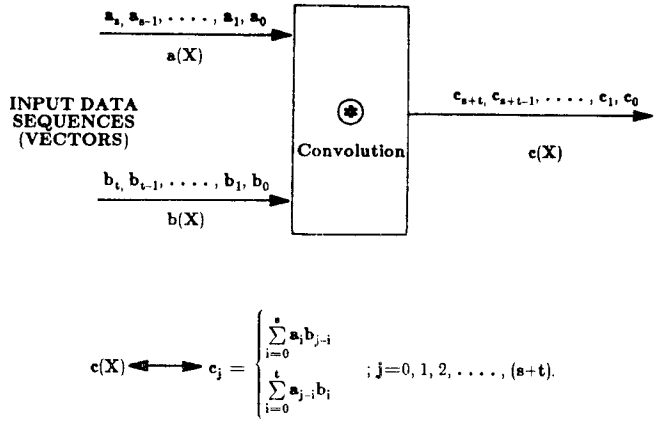
$$\text{DATA SEQUENCES} \begin{cases} a_0, a_1, a_2, \ldots, a_s \\ b_0, b_1, b_2, \ldots, b_t \end{cases} ; \; a_i, b_j \in Z_q \text{ or R or C}$$

Their convolution involves the well-defined operations in the underlying ring or field

$$c_j = \sum_{i=0}^{s} a_i b_{j-i} \qquad j = 0, 1, 2, \ldots, (s+t). \quad (1)$$

$$c_j = \sum_{i=0}^{t} a_{j-i} b_i$$

In these defining equations, any sample with index outside the prescribed range is considered to be zero. See Figure 1 for a schematic representation of the basic operation to be protected.



$$c(X) \longleftrightarrow c_j = \begin{cases} \sum_{i=0}^{s} a_i b_{j-i} \\ \sum_{i=0}^{t} a_{j-i} b_i \end{cases} ; \; j=0, 1, 2, \ldots, (s+t).$$

CONVOLUTION OF ARRAYS
FIGURE 1

This work was supported in part by the Office of Naval Research through Grant N00014-86-K-0518.

A modern view and the main point of departure for several recent texts [1-3] considers these sequences as polynomials in an indeterminant X. Then convolution is intrinsic in the normal definition of polynomial products.

$$c(X) = a(X)b(X) \qquad (2)$$

where

$$a(X) = a_0+a_1X+a_2X^2+\ldots+a_sX^s \leftrightarrow \{a_0, a_1, \ldots, a_s\}$$

$$b(X) = b_0+b_1X+b_2X^2+\ldots+b_tX^t \leftrightarrow \{b_0, b_1, \ldots, b_t\}$$

$$c(X) = c_0+c_1X+c_2X^2+\ldots+c_{s+t}X^{s+t}$$

$$\leftrightarrow \{c_0, c_1, \ldots, c_{s+t}\}$$

The polynomials in these cases belong to an algebraic structure, the commutative ring of polynomials, usually given the respective symbols $Z_q[X]$, $R[X]$ and $C[X]$.

Many fast signal processing algorithms rely upon the mathematical properties arising from this polynomial view. For example, when an exponent k is chosen sufficiently large, an equivalent form of equation (2) makes the algebraic structure even richer by introducing residue class rings modulo (X^k-1), [1].

$$c(X) \equiv a(X)b(X) \text{ modulo } (X^k-1) \; ; \quad k > s+t+1 \qquad (3)$$

The potential for protecting such operations with cyclic error-correcting codes is obvious in light of this equation. Cyclic codes are defined and manipulated as polynomial residue algebras, and their common fundamental processing operation involves polynomial products [4]. Thus it is natural to investigate cyclic codes as a powerful means of detecting errors in these types of operations.

In order to apply cyclic codes to the arithmetic setting being considered here, two hurdles must be overcome. Firstly, most cyclic codes are defined over finite fields, primarily because their design depends upon roots of polynomials in extension fields. Secondly, and equally as important, no previously known data encoding format exists which leaves the data symbols in their unaltered form while appending the proper parity symbols when passed through the polynomial product operation. The data and parity positions become intermixed when processed in this way. The first difficulty is resolved by generalized cyclic codes which have just recently been presented in the literature [5,6]. These types of codes will be reviewed here. The second problem is solved by a new encoding approach to be introduced here.

GENERALIZED CYCLIC CODES

Cyclic codes represent a powerful and wide class of codes with easily determined guaranteed distance properties that can be used for detecting both random and burst errors. They are naturally defined in a residue class ring of a polynomial algebra using the modulus (X^n-1), where n is the code length [4]. Their defining feature states that every cyclic end-around shift of the elements comprising a code polynomial is also a code polynomial. That is the purpose of the modulo (X^n-1) reduction.

The salient features of generalized cyclic codes will be explained using a generic algebraic structure Γ which is at least a commutative ring with identity. Such a structure also covers the fields of real numbers R and complex numbers C. The ring of polynomials $\Gamma[X]$, the set of polynomials in indeterminant X, can be reduced to a residue class ring using (X^n-1). This new structure is written symbolically as $\Gamma[X]/(X^n-1)$.

A generalized cyclic code is defined by a single generator polynomial, g(X), whose leading term is a unit in Γ. The code is a principal ideal generated by g(X), denoted by $((g(X)))$, and formally defined as

$$((g(X))) =$$

$$\{p(X) \equiv q(X)g(X) \text{ modulo } (X^n-1) : q(X)\varepsilon\Gamma[X]\} \qquad (4)$$

The degree of g(x) is (n-k), the number of parity positions supported by the code. Construction techniques for the generator polynomial depend on the exact nature of Γ. Nevertheless, the Euclidean Algorithm is one common underlying principle. It guarantees unique quotients and remainders for division, provided that the highest indexed oefficient in the divisor g(X) is a unit (invertible) element of Γ. Since this general result will be cited later, it will be stated here [8]. For any $f(X)\varepsilon\Gamma[X]$ the exist polynomials q(X), the quotient, and r(X), the remainder, such that

$$f(X) = q(X)g(X) + r(X)$$

$$; \text{ degree } \{r(X)\} < \text{degree } \{g(X)\} \qquad (5)$$

The Euclidean Algorithm also shows the burst detecting capabilities of a cyclic code. A burst is a consecutive segment of a code word's elements which have the beginning and ending elements of the segment in error, permitting any number of erroneous position in between [7]. Since the code is cyclic, a burst can also occur in an end-around sense. A burst can be modeled by adding an error polynomial of the form $X^ue(X)$ to the code word. However a disruption like this can always be detected as long as g(X) does not divide e(X). (Remember every code word by construction (4) is a multiple of g(X), and thus a burst divisible by the generator polynomial is undetectable.) When the polynomial part, e(X), of the burst error polynomial has degree less than that of g(X), this division is impossible, providing the burst detecting ability of (n-k) positions.

Cyclic codes over the real or complex fields are defined by using consecutively indexed powers of the n^{th} complex root of unity, e.g., $[\exp(j\,2\pi/n)p]$, where $j = \sqrt{-1}$ and p is any integer modulo n. The funda mental construction techniques are given by Marshall [6], and use the discrete Fourier transform domain in which contiguously indexed transform coefficients determine the generator polynomial. By requiring conjugate roots be included, real generator polynomials are constructed. In the more general case of complex numbers, maximum distance separable codes [4] (analagous to powerful Reed-Solomon codes) are easily established. Such codes can detect erroneous positions equal in number to the degree of g(X), the maximum permitted by the Singleton bound [4].

Generalized cyclic codes over finite integer rings, Z_{p^m}, can be defined by first examining roots in some extension field of the finite field Z_p, p a prime number [5]. Primitive elements in this extension field are studied as members of a multiplicative cyclic group in Z_{p^m}. Again, consecutively indexed roots and their conjugates are used in constructing the generator polynomial with the desired error-detecting parameters. The most general situation for Z_q involves fields and rings associated with the prime factors in the integer q. The Chinese Remainder Theorem [1] allows components over Z_{p^m} to be reassembled, defining a generator polynomial over Z_q. The lengths of segments of adjacently indexed roots guarantee the detecting performance of the final code.

These techniques along with others [9-12] insure the availability of a variety of generalized cyclic codes that can provide a wide choice of random and burst error-detecting abilities. With the existence of good cyclic codes over the proper rings and fields established, the second problem of efficient systematic encoding and parity manipulation of the code word symbols will be addressed.

SYSTEMATIC DATA ENCODING AND PARITY MANIPULATION

One important requirement of any protection scheme is the location and manipulation of the original data and associated parity symbols without additional processing. There are several methods for encoding data code words with this feature, generally referred to as systematic encoding [7]. However when two such encoded code words are producted (to implement the convolution), the corresponding parity symbols are not easily distinguished except with complicated processing. This section will first demonstrate a standard systematic encoding technique from which the respective data and parity may be extracted easily. Then a new approach will be given for processing and combining the respective parity parts to yield the new parity values corresponding to the convolved data segments.

A common systematic encoding scheme relies on the Euclidean Algorithm for uniquely defining the parity positions [7]. The data portion, say a(X), is placed in the higher indexed positions, by multiplying by X^{n-k}, effectively shifting to data to inclusively indexed positions, (n-k), (n-k+1), ..., (n-1). The uniquely related parity symbols are represented by the polynomial $r_a(X)$, derived from equation (5) with g(X) as the divisor.

$$\{X^{n-k}a(X)\} = q_a(X)g(X) + r_a(X)$$

$$; \quad \deg r_a(X) < \deg g(X) \qquad (6)$$

The code word affiliated with data a(X) is given by

$$a(X) \xrightarrow{\text{ENCODE}} [X^{n-k}a(X) - r_a(X)]$$

A simple transposition in equation (6) shows that this is indeed a multiple of g(X), the defining property of a cyclic code word (see equation (4)). Furthermore the parity values represented by $r_a(X)$ do not interfere with the original data, now shown as $X^{n-k}a(X)$, in their shifted positions. A similar encoding also applies to data b(X), where $r_b(X)$ is the unique remainder analogous to equation (6).

$$b(X) \xrightarrow{\text{ENCODE}} [X^{n-k}b(X) - r_b(X)]$$

The protection of the convolution of two code words is considered. If the two respective code words for a(X) and b(X) are producted, it is easy to see the intermingling and overlapping of parity and data parts. However the data portions are easily extracted and producted. Then the question is: how can the parity parts $r_a(X)$ and $r_b(X)$ be processed to yield the correct parity? In symbols, what relationship exists between $r_a(X)$ and $r_b(X)$, and the new parity $r_{ab}(X)$ related to the product a(X)b(X)?

$$[a(X)b(X)] \xrightarrow{\text{ENCODE}} \{X^{n-k}[a(X)b(X)] - r_{ab}(X)\}$$

This new parity part is the remainder in the division by g(X).

$$r_{ab}(X) \equiv X^{n-k}[a(X)b(X)] \bmod g(X) \qquad (7)$$

The answer, to be demonstrated conclusively in the next paragraph, is computationally straightforward.

$$r_{ab}(X) \equiv f(X)r_a(X)r_b(X) \bmod g(X) \qquad (8a)$$

where

$$f(X) \equiv X^k \bmod g(X)\ . \qquad (8b)$$

The validity of the above claim revolves round showing that the expression $\{X^{n-k}[a(X)b(X)] - r_{ab}(X)\}$ is a multiple of g(X), modulo (X^n-1). In this regard two identities need to be compiled. The first comes

from equations (8) which imply that there is a quotient $q_{ab}(X)$ satisfying

$$r_{ab}(X) = X^k r_a(X) r_b(X) - q_{ab}(X) g(X) \qquad (9)$$

The second needed expression follows from the coded form of b(X), similar to equation (6), this time implying another quotient $q_b(X)$.

$$b(X) \equiv x^n b(X) \equiv X^k \{X^{n-k} b(X)\} \bmod (X^n - 1)$$

$$\equiv X^k q_b(X) g(X) + X^k r_b(X) \bmod (X^n - 1) \qquad (10)$$

These identities when combined with equation (6) permit the following series of equalities.

$$\{X^{n-k}[a(X)b(X)] - r_{ab}(X)\}$$

$$\equiv [q_a(X)g(X) + r_a(X)][X^k q_b(X)g(X) + X^k r_b(X)]$$

$$-[X^k r_a(X) r_b(X) - q_{ab}(X)g(X)] \bmod (X^n - 1)$$

$$\equiv g(X)\{X\, q_a^k(X) q_b(X) + X\, q_b^k(X) r_a(X)$$

$$+ X^k q_a(X) r_b(X) + q_{ab}(X)\} \bmod (X^n - 1) \qquad (11)$$

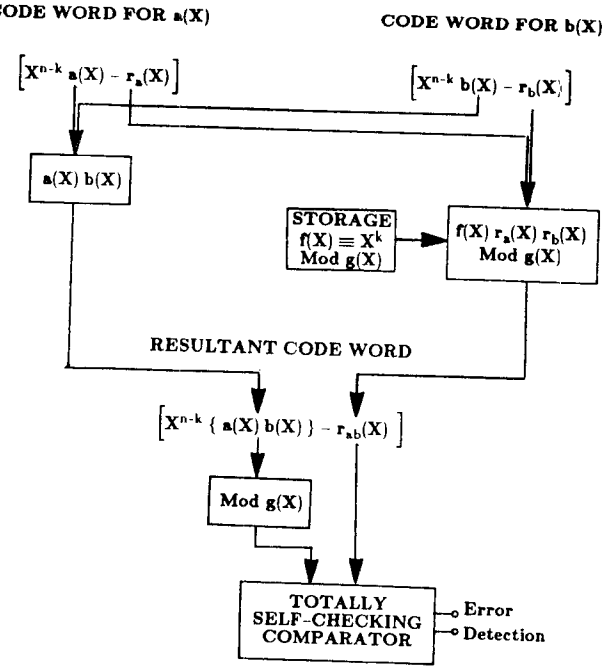
The very construction of $r_{ab}(X)$, equation (8a), guarantees that

$$\deg r_{ab}(X) < \deg g(X) .$$

On the other hand, the Euclidean Algorithm asserts a unique remainder polynomial associated with the encoding of [a(X)b(X)]; equations (8) provide that polynomial and it has degree less than (n-k) also.

The use of this systematic encoding in a fault-tolerant realization for convolving sequences represented by a(X) and b(X) is shown schematically in Figure 2. The steps in forming the new systematically encoded code word are easily identified with straightforward manipulations. The protection overhead is governed by the modulo g(X) operations, which in turn are proportional to the degree of g(X), the number of parity positions employed by the code. Even the regeneration of the parity symbols, needed in the totally-self checking comparator [13], depends on the code generating polynomial g(X). The complexity of the modulo reductions will be discussed in the next section.

There are situations where one of the sequences to be convolved is fixed and known in advance. For example, b(X) could be the impulse response of a digital filter [1,2]. The previous technique can be used to store the known sequence and its precomputed parity positions. For a predetermined sequence represented by b(X), the stored positions correspond to the code word



USE OF SYSTEMATIC ENCODING FORMAT IN PROTECTING CONVOLUTION {a(X)b(X)}

FIGURE 2

$\{X^{n-k} b(X) - r_b(X)\}$. One simplification allows the parity positions $r_b(X)$ to be combined with be combined with f(X), equation (8b), reducing the number of operations required for $r_{ab}(X)$. However, there is an alternate, equally effective method for handling this special circumstance.

The known sequence, say b(X), is also stored in its reduced form modulo g(X). Then the systematically encoded code word related to [a(X)b(X)] is given by

$$X^{n-k}[a(X)b(X)] - r_0(X) ,$$

where the parity positions are defined uniquely as

$$r_0(X) \equiv \beta(X) r_a(X) \bmod g(X) \qquad (12a)$$

$$\beta(X) \equiv b(X) \bmod g(X) \qquad (12b)$$

The correctness of this approach is easily demonstrated by noting that equations (12) insure the existence of a quotient $q_0(X)$ and remainder $r_0(X)$ giving

$$\beta(X) r_a(X) = q_0(X) g(X) + r_0(X) \qquad (13)$$

After cancellation of terms, the code representation for [a(X)b(X)] is clearly a code word.

$$X^{n-k}[a(X)b(X)] - r_0(X) \equiv$$

$$g(X)[q_a(X) + q_0(X)] \bmod (X^n - 1)$$

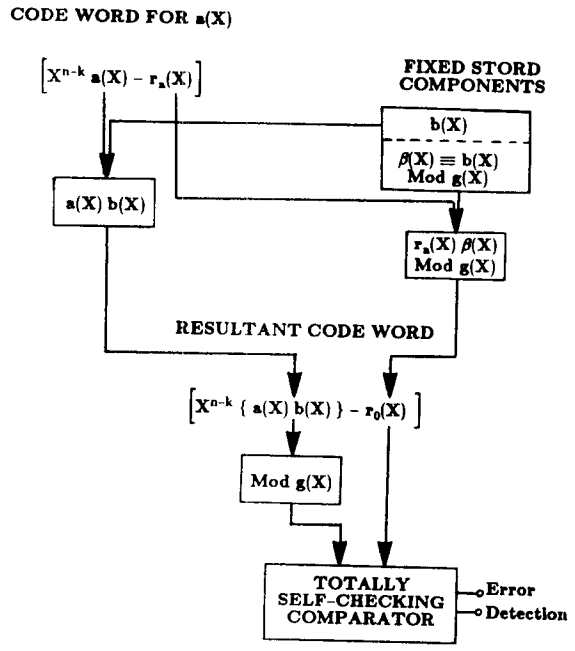
Thus this abbreviated method still produces a code word, with all the error detecting potential of the code, but with a simpler formula for the parity portion.

This reduced special case occurs because only one of the parity parts $r_a(X)$ emanates from $X^{n-k}a(X)$, allowing ease in separating the effects of data and parity. The slightly less complex implementation of this method is depicted in Figure 3. The residue of the fixed sequence $\beta(X)$ is stored as well to expedite the parity formation.

COMPLEXITY OF PARITY GENERATION

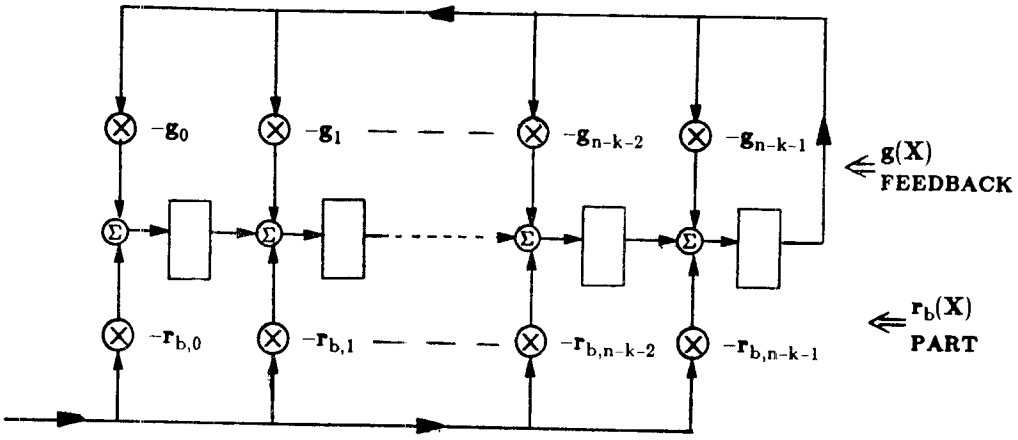
There are numerous ways to realize the convolution and modulo reductions prescribed in the previously described methods. They range from distributed arithmetic processors to time-multiplexing a high-speed ALU resource. In order to study the general complexity, a realization as shown in Figure 4 is considered. The basic principles are easily adapted to many other configurations including one which uses general addressable memory and a microprogrammed controller. There are (n-k) storage locations and the leading coefficient of the code generating polynomial g(X) is taken as 1, without loss of generality. The lower portion of this figure implements the product $r_a(X)r_b(X)$, while the feedback path in the upper part performs the mod g(X) reduction simultaneously.

One complexity measure may be a count of the number of multiplications and additions required. The product part needs multiplications and additions on the order of the following estimates.



PROTECTING CONVOLUTION WHEN ONE SEQUENCE IS KNOWN
FIGURE 3

PRODUCT PART { MULTIPLICATIONS: $(n-k)(n-k+1)$; ADDITIONS: $(n-k)(n-k-1)$



PART OF PARITY GENERATION
$-\{r_a(X)\,r_b(X)\}$ Mod g(X)
FIGURE 4

On the other hand, the modulo reduction is heavily influenced not only by the degree of $g(X)$, but also by the number of nonzero coefficients. The notation $|g|$ will denote the number of nonzero terms in $g(X)$, including the leading coefficient presently assumed to be 1. Then the additional number of multiplications and additions for the feedback part may be estimated.

MODULO REDUCTION PART	MULTIPLICATIONS:	$(n-k-1)(\|g\|-1)$
	ADDITIONS:	$(n-k-1)(\|g\|-1)$

These estimates will be helpful in projecting the overall complexities. There are two different approaches that may be taken in realizing the parity calculations dictated by equation (8a). One approach would first form the triple product $\{f(X)r_a(X)r_b(X)\}$, yielding a polynomial with possible highest degree $3(n-k-1)$, followed by the modulo reduction. This approach uses more interim memory locations. An alternate method would interconnect two series of operations as depicted in Figure 4. With this realization equation (8a) is implemented in two stages, say first producing

$$d(X) \equiv r_a(X)r_b(X) \bmod g(X) ,$$

and then completing the calculations with

$$r_{ab}(X) \equiv d(X)f(X) \bmod g(X) .$$

Straightforward estimates show that either approach employs on the same respective orders of multiplications and additions.

ORDER OF PARITY GENERATION COMPLEXITY

MULTIPLICATIONS:	$2[(n-k)^2 + (n-k-1)(\|g\|-1)$
ADDITIONS:	$2(n-k-1)[(n-k) + (\|g\|-1)]$

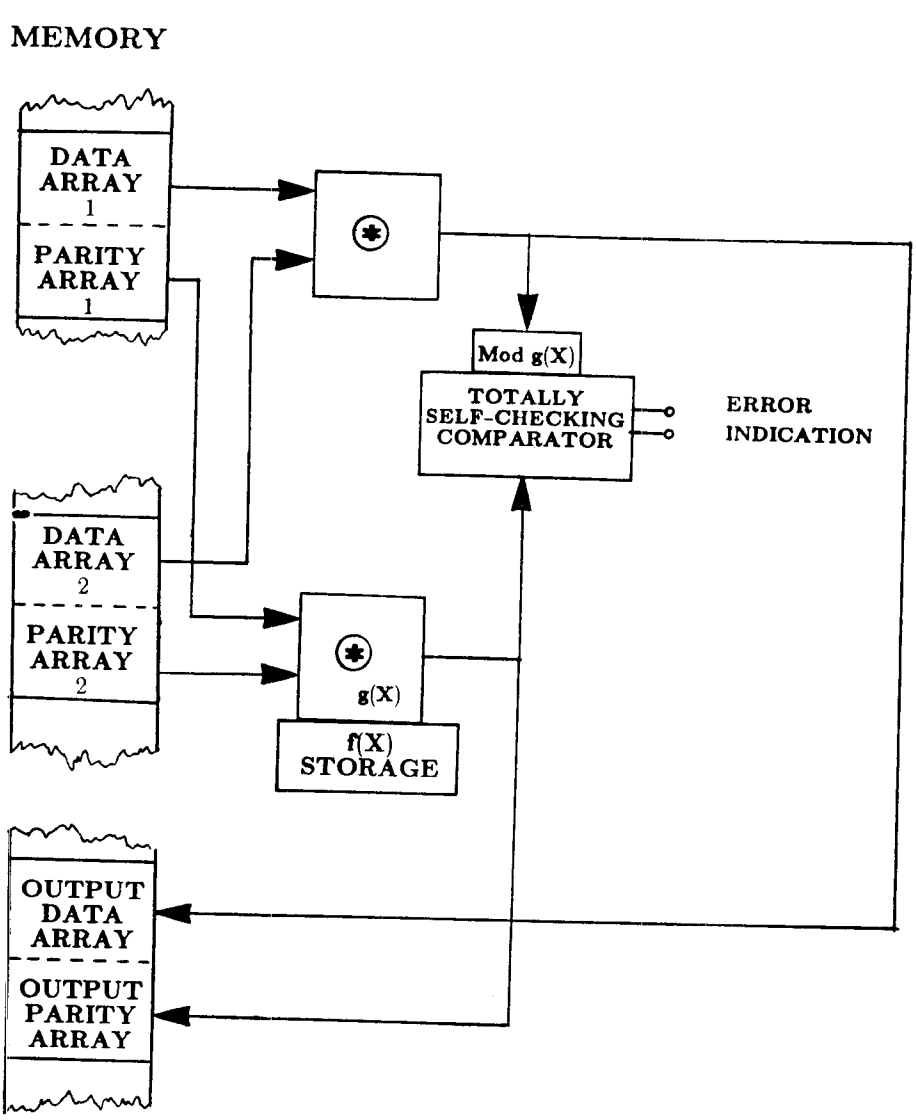
The dominant factor in both items is $(n-k)^2$, a quantity related to the number of parity positions in the code.

CONCLUSIONS

Recent coding theory results, which permit generalized cyclic codes over rings and fields commonly employed in arithmetic formats, may be applied in protecting convolution-type array calculations. Of course, any ancillary array additions or scalings are also protected because cyclic codes possess the usual linearity properties. New systematic encoding and processing techniques make their application straightforward and efficient. The methods may be combined in numerous ways offering a wide variety of powerful techniques for introducing error detection directly in the realizations of such operations. One such configuration for employing these techniques is shown in Figure 5.

REFERENCES

1. J. H. McClellan and C. M. Rader, Number Theory in Digital Signal Processing. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1979.
2. H. J. Nusbaumer, Fast Fourier Transform and Convolution Algorithms. New York: Springer-Verlag, 1981.
3. R. E. Blahut, Fast Algorithms for Digital Signal Processing. Reading, Massachusetts: Addison-Wesley Publishing Co., 1985.
4. W. W. Peterson and E. J. Weldon, Jr., Error-Correcting Codes, Second Edition. Cambridge, Massachusetts: The MIT Press, 1972.
5. P. Shankar, "On BCH Codes Over Arbitrary Integer Rings," IEEE Transactions on Information Theory, Vol. IT-25, pp. 480-483, July 1979.
6. T. G. Marshall, Jr., "Coding of Real-Number Sequences for Error Correction: A Digital Signal Processing Problem," IEEE Journal of Selected Areas in Communications, Vol. SAC-2, pp. 381-392, March 1984.
7. S. Lin and D. J. Costello, Jr., Error Control Coding: Fundamentals and Applications. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1983.
8. N. Jacobson, Basic Algebra I. San Francisco: W. H. Freeman and Co., 1974.
9. I. F. Blake, "Codes Over Certain Rings," Information and Control, Vol. 20, pp. 396-404, 1972.
10. I. F. Blake, "Codes Over Integer Residue Rings," Information and Control, Vol. 29, pp. 295-300, 1975.
11. E. Spiegel, "Codes Over Z_m," Information and Control, Vol. 35, pp. 48-51, 1977.
12. E. Spiegel, "Codes Over Z_m, Revisited," Information and Control, Vol. 37, pp. 100-104, 1978.
13. J. Wakerly, Error Detecting Codes, Self-Checking Circuits and Applications. New York: North-Holland, 1978.



$\circledast_{g(X)}$ **Denotes Convolution, Modulo g(X).**

ONE POSSIBLE SYSTEM CONFIGURATION
FIGURE 5