Fast Fault-Tolerant Digital Convolution Using a Polynomial Residue Number System

Paul E. Beckmann, *Member, IEEE*, and Bruce R. Musicus, *Member, IEEE*

***Abstract*—We describe a fault-tolerant convolution algorithm which is an extension of residue number system fault-tolerance schemes applied to polynomial rings. The algorithm is suitable for implementation on multiprocessor systems and is able to concurrently mask processor failures. We develop a fast algorithm based on long division for detecting and correcting multiple processor failures. We then select moduli polynomials that yield an efficient and robust FFT-based algorithm. For this implementation, we study single fault detection and correction, and apply a generalized likelihood ratio test to optimally detect system failures in the presence of computational noise.**

The coding scheme which is presented is capable of protecting over 90% of the computation involved in convolution. Parts not covered by our scheme are assumed to be protected via triple modular redundancy. This hybrid approach is able to detect and correct any single system failure with as little as 70% overhead, compared with 200% needed for a system fully protected via modular redundancy.

I. Introduction

Convolution is an important operation used in many digital signal processing applications. It is the basis of digital filtering and correlation, and performing convolution is often the most computationally intensive step of an algorithm. Extremely high throughput convolution systems use multiprocessor configurations [1]. Computation is distributed over several processors, and each computes a portion of the result. Unfortunately, the large amount of hardware in multiprocessor systems increases the likelihood that a soft or hard failure will occur and corrupt the result. Thus some degree of fault tolerance is desirable in these systems.

The traditional approach to fault tolerance is modular redundancy (MR) [2], [3]. In this technique, several identical copies of the system operate in parallel using the same data, and their outputs are compared with voter circuitry. If no errors have occurred, all outputs will agree exactly. Otherwise, if an error has occurred, the faulty module can be easily identified and the correct output determined. MR is a very general technique and can be applied to any computational task. Unfortunately, it does not take advantage of the specific structure of a problem and requires a large amount of hardware overhead (200% for single error correction). Also, when compared to binary error-correcting codes, MR provides less protection relative to the overhead needed.

Recently, an alternative method of protecting signal processing operations called algorithm-based fault-tolerance (ABFT) has emerged [4]. High level arithmetic codes are used to encode entire sequences or arrays of data, rather than individual samples. Algorithms are modified to operate on this encoded data. ABFT has been successfully applied to matrix operations [5], FFT's [6], and has been generalized to include all linear signal processing operations [7], [8]. In ABFT schemes, a high level arithmetic code protects the majority of computation, while MR protects parts not covered by the code. This hybrid scheme yields robust systems at a significantly lower cost than fully redundant implementations.

Protection of convolution operations has been previously studied by Redinbo [9]. He applies generalized cyclic error-correcting codes with known minimum distance properties [10]. These codes are basically BCH codes defined over the fields of real or complex numbers. Redundancy is incorporated by premultiplying sequences by a generator polynomial. The resulting code is systematic and separate, with parity symbols being processed in an independent channel. The main limitation of this work is that the error detecting and correcting properties of generalized cyclic codes dictate that direct evaluation of convolution be used. Fast algorithms, such as the FFT, may not be employed.

This paper presents a novel, fault-tolerant convolution algorithm which is based on a polynomial residue number system (RNS). As in integer RNS schemes, computation is decomposed into independent residue channels and redundancy incorporated by adding extra residue channels. This decomposition yields an algorithm which is computationally equivalent to the Winograd convolution algorithm (WCA), and its parallel structure makes it ideally suited for multiprocessor implementations. Furthermore, and most importantly, our fault-tolerant algorithm has the same underlying structure as fast convolution algorithms which are based on polynomial residue number systems, and our algorithm is fast and efficient as well. We also

Manuscript received March 26, 1991; revised July 31, 1992. The associate editor coordinating the review of this paper and approving it for publication was Dr. Pierre Duhamel. This work was supported in part by a Rockwell Doctoral Fellowship, in part by Draper Laboratory, Inc., under Grant DL-H-418472, and in part by the Advanced Research Projects Agency monitored by ONR under Grant N00014-89-J-1489.

P. E. Beckmann was with the Research Laboratory of Electronics, Massachusetts Institute of Technology, Cambridge, MA 02139. He is now with the Bose Corporation, Framingham, MA 01701.

B. R. Musicus is with the Research Laboratory of Electronics, Massachusetts Institute of Technology, Cambridge, MA 02139 and Bolt, Beranek, & Newman, Cambridge, MA 02138.

IEEE Log Number 9208844.

optimally handle the effects of computational noise by detecting and correcting errors with a generalized likelihood ratio test.

The derivation of the fast fault-tolerant algorithm is done in several steps, and its practicality becomes evident only at the very end. In Section II we present background material in polynomial rings and summarize the WCA. Then in Section III we add redundancy to the WCA and present a fault detection and correction scheme which can handle multiple processor failures. The derivation is very general, and yields a wide variety of implementations. Section IV focuses on single error correction, and we choose moduli such that the algorithm may be computed efficiently with FFT's. In Section V we apply a generalized likelihood ratio test to deal with computational noise inaccuracies, and in Section VI discuss the efficiency of our algorithm. We conclude with Section VII, where we summarize this paper's contributions.

II. Winograd Convolution Algorithm

In this section we describe the fundamentals of the Winograd convolution algorithm which forms the basis of our fault-tolerant algorithm. The operation we are interested in protecting is the linear convolution of two finite length data sequences. Let $a[n]$ and $b[n]$ be P-point sequences which are nonzero for $0 \le n \le P - 1$. The linear convolution, denoted by $a[n] * b[n]$, results in a $Q = 2P - 1$ point sequence $c[n]$ as follows:

$$c[n] = \sum_{i=0}^{n} a[i]b[n-i] \quad \text{for } n = 0, \cdots, Q-1. \tag{1}$$

A key idea we exploit is to represent individual samples as elements of a field and entire sequences as elements of a polynomial ring [11]. Let F be a field and denote by $F[x]$ the set of polynomials with coefficients in F. $F[x]$ is a ring under the operations of polynomial addition and multiplication, and is called the ring of polynomials in x over F. We represent the sequence $a[n]$ by the polynomial

$$a(x) = \sum_{i=0}^{P-1} a[i]x^i \tag{2}$$

and represent $b[n]$ and $c[n]$ by $b(x)$ and $c(x)$ in a similar fashion. The degree of polynomial $a(x)$, denoted by deg $a(x)$, refers to the highest power of x in $a(x)$.

Let $M(x)$ be an element of $F[x]$, and denote by $F[x]/M(x)$ the set of polynomials in $F[x]$ with degree less than deg $M(x)$. $F[x]/M(x)$ is a ring under normal polynomial addition and multiplication modulo $M(x)$. The division algorithm for polynomials guarantees the uniqueness of the modulo operation [12]. We use the notation $r(x) = \langle a(x) \rangle_{M(x)}$ to represent the remainder when $a(x)$ is divided by $M(x)$. $M(x)$ is called the modulus and $r(x)$ is called the residue.

It is well known [13] that the polynomial product $c(x) = a(x)b(x)$ is equivalent to the linear convolution in (1). Furthermore, the product can be computed in the finite degree polynomial ring $F[x]/M(x)$ by choosing an $M(x)$ such that deg $M(x) >$ deg $c(x)$. The modulo operation will not affect the result and a linear convolution will still be computed.

A polynomial residue number system (RNS) [13] is an isomorphic representation of the finite degree polynomial ring $F[x]/M(x)$. It decomposes a large ring into a direct sum of several smaller rings. To define a polynomial RNS isomorphic to $F[x]/M(x)$, we first factor $M(x)$ into N relatively prime polynomials,

$$M(x) = m_1(x)m_2(x) \cdots m_N(x) \tag{3}$$

where each $m_k(x)$ is a member of $F[x]/M(x)$. Then, it can be shown that the direct sum of rings $F[x]/m_1(x) \times \cdots \times F[x]/m_N(x)$ is isomorphic to $F[x]/M(x)$. The mapping from $a(x) \in F[x]/M(x)$ to its direct sum representation is accomplished by computing the N residues

$$a_k(x) = \langle a(x) \rangle_{m_k(x)} \quad \text{for } k = 1, \cdots, N \tag{4}$$

where $a_k(x) \in F[x]/m_k(x)$. We denote this isomorphism by

$$a(x) \approx \{a_1(x), a_2(x), \cdots, a_N(x)\}. \tag{5}$$

The inverse mapping from a set of residues $\{a_1(x), a_2(x), \cdots, a_N(x)\}$ to $a(x) \in F[x]/M(x)$ is computed by the Chinese remainder theorem (CRT) for polynomials:

$$a(x) = \sum_{k=1}^{N} \langle a_k(x)D_k(x) \rangle_{m_k(x)} M_k(x) \tag{6}$$

where

$$M_k(x) = \frac{M(x)}{m_k(x)} \tag{7}$$

and $D_k(x)$ is chosen such that

$$\langle M_k(x)D_k(x) \rangle_{m_k(x)} = 1. \tag{8}$$

The isomorphism between $F[x]/M(x)$ and its direct sum allows us to perform computation in $F[x]/M(x)$ by operations in each of the smaller rings $F[x]/m_1(x), \cdots, F[x]/m_N(x)$. The isomorphism holds for both ring operations. Let $a(x)$ and $b(x)$ be members of $F[x]/M(x)$ with residue representations,

$$a(x) \approx \{a_1(x), \cdots, a_N(x)\}$$
$$\text{and} \quad b(x) \approx \{b_1(x), \cdots, b_N(x)\}. \tag{9}$$

The residue representation of the sum $\langle a(x) \pm b(x) \rangle_{M(x)}$ or product $\langle a(x)b(x) \rangle_{M(x)}$ can be computed by N independent residue additions or multiplications:

$$\langle a(x) \pm b(x) \rangle_{M(x)} \approx \{a_1(x) \pm b_1(x), \cdots, a_N(x) \pm b_N(x)\} \tag{10}$$

$$\langle a(x)b(x) \rangle_{M(x)} \approx \{\langle a_1(x)b_1(x) \rangle_{m_1(x)}, \cdots, \langle a_N(x)b_N(x) \rangle_{m_N(x)}\}. \tag{11}$$

A polynomial RNS is the basis of the Winograd convolution algorithm (WCA). We have already discussed the main steps involved, and we summarize them here for clarity. To compute the linear convolution $c(x) = a(x)b(x)$, choose an $M(x)$ such that $\deg M(x) > \deg c(x)$. Define an RNS by factoring $M(x)$ into N coprime polynomials. (Both of these steps are done off-line.) Then, given the sequences $a(x)$ and $b(x)$, compute the two sets of residues (9). Perform the residue multiplications in each of the smaller rings (11), and then reconstruct using the CRT (6). The WCA thus computes a long convoution using N simpler polynomial products in independent residue channels.

Convolving two sequences of length P using direct convolution (1) requires $\mathcal{O}(P^2)$ operations. If we use the WCA instead and choose the moduli polynomials carefully, then convolution can be computed with as little as $\mathcal{O}(P \log_2 P)$ operations. The savings over direct convolution can be substantial.

III. Fault-Tolerant System

In this section we incorporate redundancy in the WCA to yield a robust algorithm. The WCA is mapped onto a multiprocessor architecture and a suitable error model is derived. Then an algorithm for detecting and correcting multiple processor failures is presented.

We add redundancy to the WCA by adding extra residue channels and constraining the length of the convolution. We start with a Q-point linear convolution computed using N moduli, and add C extra moduli $M_{N+1}(x), \cdots, m_{N+C}(x)$. These moduli must be coprime to each other and to the original N moduli. Since $N + C$ residues are used, computation is isomorphic to the larger ring $F[x]/M^+(x)$ where $M^+(x) = \Pi_{k=1}^{N+C} m_k(x)$. In $F[x]/M^+(x)$ a convolution of length $Q^+ = \Sigma_{k=1}^{N+C} \deg m_k(x)$ could be computed. However, we restrict the lengths of the input sequences such that the result has length $Q \leq \Sigma_{k=1}^{N} \deg m_k(x)$. Only a portion of the allowable output length is used and the $Q^+ - Q$ high order coefficients of the result should be zero.

A. Multiprocessor Architecture

We distribute computation of the WCA in a multiprocessor system such that one residue is corrupted per processor failure. We accomplish this by performing the computation needed for each residue channel on a separate, independent processor. Thus, we assume that $N + C$ processors are available, and assign to the kth processor the computation of the kth input residues

$$a_k(x) = \langle a(x) \rangle_{m_k(x)} \tag{12}$$

$$b_k(x) = \langle b(x) \rangle_{m_k(x)} \tag{13}$$

as well as the kth residue product

$$c_k(x) = \langle a_k(x) b_k(x) \rangle_{m_k(x)}. \tag{14}$$

Our approach can only protect against errors in the first two steps of the WCA. To ensure proper computation of the CRT reconstruction and error detection and correction, we assume that triple modular redundancy (TMR) is used in these steps. This hybrid approach is practical since the bulk of computation occurs during the first two steps of the WCA. A diagram of the overall system architecture is shown in Fig. 1.

Our chief concern is guarding against failures in the $N + C$ residue processors, and so we assume that data I/O and interprocessor communication are reliable. These functions can be protected using standard techniques such as binary error-correcting codes or triplicated buses. We declare that a processor has failed when it does not compute the correct result given its input. This model covers a wide range of possible processor failures including transient single bit arithmetic errors as well as complete processor failure. We also assume that when a processor fails, it corrupts only the computation assigned to it, and does not affect the communication network or any other processor. With this model, one residue will be corrupted per processor failure.

We denote the outputs of the processors by $z_k(x)$, and assume that λ failures occur in processors $\{k_1, \cdots, k_\lambda\}$. The processor output will have value

$$z_k(x) = \begin{cases} c_k(x) + \phi_k(x) & \text{for } k = k_1, \cdots, k_\lambda \\ c_k(x) & \text{else} \end{cases} \tag{15}$$

where $\phi_{k_i}(x)$ is the net effect of the failure in channel k_i, and $\deg \phi_{k_i}(x) < \deg m_{k_i}(x)$. Using this set of residues, the outputs of the reliable CRT step is

$$z(x) = \sum_{k=1}^{N+C} \langle z_k(x) D_k^+(x) \rangle_{m_k(x)} M_k^+(x) \tag{16}$$

where now

$$M_k^+(x) = \frac{M^+(x)}{m_k(x)} \tag{17}$$

and $D_k^+(x)$ is chosen such that

$$\langle D_k^+(x) M_k^+(x) \rangle_{m_k(x)} = 1. \tag{18}$$

Substituting (15) into (16) gives

$$z(x) = c(x) + \sum_{i=1}^{\lambda} \langle \phi_{k_i}(x) D_{k_i}^+(x) \rangle_{m_{k_i}(x)} M_{k_i}^+(x) \tag{19}$$

and we see that processor failures affect the result in an additive manner.

B. Fault Detection and Correction

We now develop an algorithm that detects and corrects multiple processor failures by examining the result $z(x)$. The algorithm is complicated by the fact that we must determine the exact number and location of failed processors. Let D be the largest nonnegative integer satisfying

$$\deg \left[\frac{M^+(x)}{m_{l_1}(x) \cdots m_{l_D}(x)} \right] \geq Q \quad \text{for every set of } D \text{ unique moduli } \{m_{l_1}(x), \cdots, m_{l_D}(x)\}. \tag{20}$$

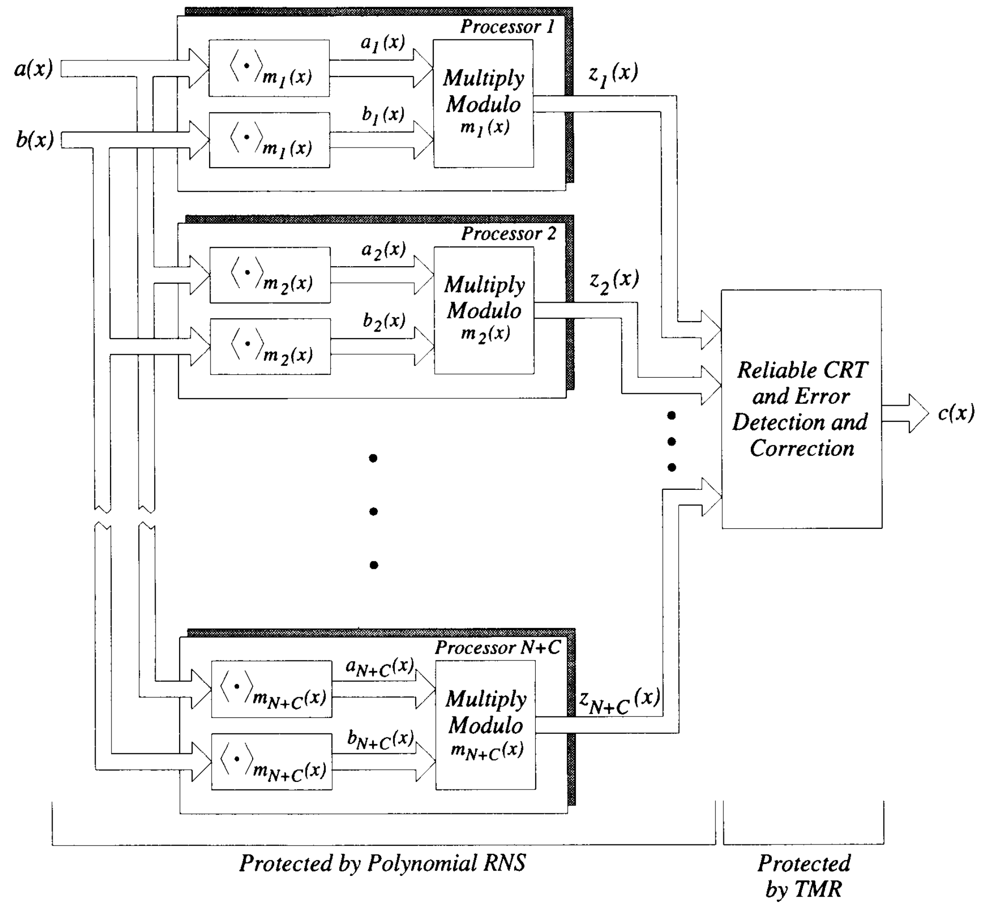

Processor 1
a(x)
b(x)
$\langle\cdot\rangle_{m_1(x)}$
$a_1(x)$
$b_1(x)$
Multiply Modulo $m_1(x)$
$z_1(x)$
Processor 2
$\langle\cdot\rangle_{m_2(x)}$
$a_2(x)$
$b_2(x)$
Multiply Modulo $m_2(x)$
$z_2(x)$
Reliable CRT and Error Detection and Correction
c(x)
Processor N+C
$\langle\cdot\rangle_{m_{N+C}(x)}$
$a_{N+C}(x)$
$b_{N+C}(x)$
Multiply Modulo $m_{N+C}(x)$
$z_{N+C}(x)$
Protected by Polynomial RNS
Protected by TMR

Fig. 1. Robust multiprocessor architecture used to compute convolutions. Computation is first divided into $N + C$ residue operations, each of which is computed by an independent processor. The outputs of these processors are fed to a reliable system which computes the CRT reconstruction and performs error detection and correction.

D measures the amount of redundancy present in the polynomial RNS. It serves the same purpose as the minimum distance of a binary error-correcting code. To change a valid set of residues into another valid set, at least $D + 1$ residues must be modified. The specific value of D depends on the redundant moduli $m_{N+1}(x), \cdots, m_{N+C}(x)$. If the degrees of the redundant moduli are all greater than or equal to the degrees of the original N moduli,

$$\deg m_{N+i}(x) \geq \deg m_j(x) \quad \text{for} \quad \begin{cases} i = 1, \cdots, C \\ j = 1, \cdots, N \end{cases} \tag{21}$$

then $D = C$.

The properties of a polynomial RNS with a given value of D are described in the following two theorems which are proved in Appendix A.

Theorem 1: Fault Detection

Let D satisfy (20). Then

a) If no failures occur ($\lambda = 0$), then $\deg z(x) < Q$ and the correct convolution is $c(x) = z(x)$.

b) If between 1 and D failures occur ($1 \leq \lambda \leq D$), then $\deg z(x) \geq Q$.

Theorem 2: Fault Correction

Decompose D as $D = \alpha + \beta$ for some integers $\alpha \geq \beta \geq 0$. Assume that no more than α processors can fail at any time. Then:

a) We can reliably distinguish when β or fewer errors occur from the case when between $\beta + 1$ and α failures occur.

b) If β or fewer failures occur, we can correct them by examining $z(x)$.

The decomposition $D = \alpha + \beta$ presented in Theorem 2 is not arbitrary, but determines the error detecting and correcting ability of the code. α is the maximum number of simultaneous processor failures which we will attempt to detect, while β is the maximum number we will attempt to correct. Since a failure must be detected in order to be corrected, α must be greater than or equal to β. A key issue is that we must make an assumption about the maximum number of simultaneous processor failures which can occur. Then we add sufficient redundancy to attain a desired level of protection. For example, suppose that we anticipate at most two processors failing at one time. Then we could do any of the following:

1) Detect up to two processor failures, but correct none of them ($D = 2$, $\alpha = 2$, $\beta = 0$).

2) Detect up to two processor failures. If one processor failed, we can determine this and correct the result ($D = 3$, $\alpha = 2$, $\beta = 1$).

3) Detect and correct up to two processor failures ($D = 4$, $\alpha = 2$, $\beta = 2$).

The minimum redundancy needed for single fault detection is $D = 1$, and this can be accomplished by $C = 1$ extra modulus which satisfies

$$\deg m_{N+1}(x) \geq \deg m_i(x) \qquad \text{for } i = 1, \cdots, N. \tag{22}$$

For single fault detection and correction we need $D = 2$, and this can be satisfied by $C = 2$ extra moduli satisfying

$$\deg m_{N+1}(x),\ \deg m_{N+2}(x) \geq \deg m_i(x) \qquad \text{for } i = 1, \cdots, N. \tag{23}$$

In general, with C extra moduli satisfying (21), we can simultaneously detect and correct at most $\lfloor C/2 \rfloor$ faults, where $\lfloor C/2 \rfloor$ is the largest integer not greater than $C/2$. This result is equivalent to the error detecting and correcting ability of a distance $C + 1$ binary error-correcting code [14].

Theorem 2 ensures that errors may be corrected but does not state how to perform the actual error correction. This is given in the following theorem which determines the exact number and location of the faulty processors, and during this process, corrects the output. The proof of this theorem is also given in Appendix A.

Theorem 3: Fault Correction Algorithm

Suppose the moduli satisfy the conditions (20) and assume that $\lambda \leq \alpha$ failures occurred in processors $\{k_1, \cdots, k_\lambda\}$. Then:

```
1) Given the (possibly faulty) residues $z_k(x)$, use a reliably implemented CRT to reconstruct the corresponding sequence $z(k)$.
2) If $\deg z(x) < Q$ then no fault has occurred, so $c(x) = z(x)$. STOP.
3) Otherwise a fault has occurred.
   For $p = 1, \cdots, \beta$
      For all possible sets of $p$ processors
         $1 \leq j_1 < j_2 \cdots < j_p \leq N + C$
         Compute $r_{j_1,\cdots,j_p}(x) = \langle z(x) \rangle_{M^+_{j_1,\cdots,j_p}(x)}$
         where $M^+_{j_1,\cdots,j_p}(x) = \frac{M^+(x)}{m_{j_1}(x) \cdots m_{j_p}(x)}$.
         If $\deg r_{j_1,\cdots,j_p}(x) < Q$, then the $p$ processors $\{j_1, \cdots, j_p\}$ have failed, and $c(x) = r_{j_1,\cdots,j_p}(x)$ is the correct convolution. STOP.
4) If all the polynomials $r_{j_1,\cdots,j_\beta}(x)$ have degree $\geq Q$, then $\beta + 1$ to $\alpha$ faults occurred, and this cannot be corrected. STOP.
```

This algorithm essentially does an exhaustive search of all possible combinations of failed processors. It begins by checking if no processors failed by testing if $\deg z(x) < Q$. If so, then $c(x) = z(x)$ is the correct result. Otherwise, it begins with $p = 1$ and checks if the error was caused by a single processor failure. If not, then all possible two processor failures are checked ($p = 2$). This continues until $p = \beta$. If no set of residues which explains the fault can be found, then by Theorem 2 we know that between $\beta + 1$ and α faults occurred, and this cannot be corrected. Testing for multiple failures in this manner guarantees that only faulty processors will be corrected.

The algorithm can be implemented quickly if we recognize that only the high order coefficients of the remainder (those of degree $\geq Q$) must be computed before testing if $\deg r_{j_1,\cdots,j_p}(x) < Q$. If any of these coefficients are nonzero, then we abort the division and test the next set of processors. Once we find $\{j_1, \cdots, j_p\}$ such that the high order coefficients are all zero, we know that processors $\{j_1, \cdots, j_p\}$ have failed. To obtain the correct result, we complete the division. This procedure requires roughly $(Q^+ - Q)/Q$ as much computation as performing all divisions completely.

Even with this fast fault correction algorithm, checking for multiple processor failures can be computationally expensive. The procedure in Theorem 3 essentially does an exhaustive search of all possible combinations of failed processors. Checking for β or fewer failures requires a total of

$$\sum_{i=1}^{\beta} \frac{(N + C)!}{i!(N + C - i)!} \tag{24}$$

separate polynomial divisions. This is reasonable only for small values of β.

If exact arithmetic is used, then the above procedure is sufficient. However, if any rounding or truncation occurs during computation, the high order coefficients of $r_{j_1,\cdots,j_p}(x)$ will never be exactly zero, and our fault test needs to be modified. This is done in Section V.

The error detection and correction techniques that are described in this section are similar to those used in integer RNS [15]. However, we perform arithmetic in polynomial rings, rather than in integer rings. Encoding entire sequences with a polynomial RNS has several advantages over low-level single sample encoding using an integer RNS. First, off-the-shelf fixed or floating point arithmetic units may be used since computation is performed in the complex field. Integer residue number systems, on the other hand, require nonstandard finite ring arithmetic units. They have great difficulty with rounding operations, and have limited dynamic range. Second, and most importantly, in polynomial rings the choice of moduli constrains the length of convolution that can be performed, but not the dynamic range of the sample values. Since the length is specified in advance, overflow can be avoided.

IV. Fast FFT-Based Algorithm

In this section we present a specific set of moduli which allows each step of the algorithm to be implemented efficiently. We show that when mapped onto a 2-D array, our algorithm is computationally equivalent to computing the convolution using a Cooley–Tukey FFT with two additional rows.

A. FFT Moduli

Computation is reduced if we use sparse polynomials as moduli. This simplifies computing the residues and performing the residue multiplications. Also, $M^+_{j_1,\cdots,j_p}(x)$ will be sparse, simplifying error detection and correction.

Suppose the samples to be convolved are elements of the field of complex numbers. Also assume that $a[n]$ and $b[n]$ have lengths such that the maximum length of the convolution is a composite $Q = NR$ for integers N and R. Then a particularly elegant choice for the moduli is as follows:

$$m_k(x) = x^R - W_{N+C}^{-(k-1)} \quad \text{for } k = 1, \cdots, N+C \tag{25}$$

where $W_{N+C} = \exp[j(2\pi/N + C)]$ is the $(N + C)$th root of unity. Note that $m_k(x)$ can be written as the product of R first-order factors:

$$m_k(x) = \prod_{i=0}^{R-1} (x - W_S^{-(k-1)-i(N+C)}) \tag{26}$$

where $S = (N + C)R$. From this expansion it can be seen that the $m_k(x)$ have no roots in common, and are thus coprime. Also note that

$$M^+(x) = \prod_{k=1}^{N+C} m_k(x) = x^S - 1. \tag{27}$$

Our method computes $c(x)$ modulo $M^+(x)$ which corresponds to S-point circular convolution. Hence, to achieve fault-tolerance, we have embedded a $Q = NR$ point linear convolution in an $S = (N + C)R$ point circular convolution.

We will assume that at most a single processor can fail at any one time, and add redundancy such that this failure can be reliably detected and corrected. We thus require $\beta \geq 1$ and $\alpha \geq \beta$. To minimize redundancy, we choose $\alpha = \beta = 1$ and therefore $D = 2$. Since our moduli are all of the same degree, we need two extra moduli to achieve this level of redundancy. We will assume that $C = 2$ throughout the rest of this section.

B. Algorithm Description

We now describe in detail the steps involved in a fault-tolerant convolution algorithm which uses the moduli shown in (25). We describe the steps in terms of polynomial operations and as operations on two-dimensional arrays. The latter description reveals the relationship between our fault-tolerant algorithm and standard convolution algorithms. This relationship will be discussed in Section IV-C.

We map the sequence $a[n]$ onto a 2-D array as follows:

$$a[n_1, n_2] = a[n_1R + n_2] \quad \text{for} \quad \begin{array}{l} 0 \leq n_1 \leq N+1 \\ 0 \leq n_2 \leq R-1 \end{array} \tag{28}$$

where n_1 is the row index and n_2 is the column index. Note that since $a[n]$ does not occupy the entire array, we zero pad it to length S. We compute the residues $a_k[n]$ via operations on $a[n_1, n_2]$ and place the residues in the 2-D array

$$A[k, n] = a_k[n] \quad \text{for} \quad \begin{array}{l} 1 \leq k \leq N+2 \\ 0 \leq n \leq R-1. \end{array} \tag{29}$$

The arrays $b[n_1, n_2]$, $c[n_1, n_2]$, $z[n_1, n_2]$ and $B[k, n]$, $C[k, n]$, $Z[k, n]$ are similarly defined. We also use the following array representation of the remainder $r_j(x)$ which is used during error detection and correction:

$$r_j[n_1, n_2] = r_j[n_1R + n_2] \quad \text{for} \quad \begin{array}{l} 0 \leq n_1 \leq N \\ 0 \leq n_2 \leq R-1 \\ 1 \leq j \leq N+2. \end{array} \tag{30}$$

Note that $r_j[n_1, n_2]$ has $N + 1$ rows while the other arrays have $N + 2$ rows.

1) Computation of Residues: The first step of the algorithm is to compute the residues $a_k(x)$. Since the moduli $m_k(x)$ are sparse polynomials, each coefficient of a residue polynomial is the sum of only $N + 2$ terms,

$$a_k(x) = \sum_{n=0}^{R-1} x^n \sum_{l=0}^{N+1} W_{N+2}^{-(k-1)l} a[n + lR] \quad \text{for } 1 \leq k \leq N+2 \tag{31}$$

or in array notation,

$$A[k, n] = \sum_{l=0}^{N+1} W_{N+2}^{-(k-1)l} a[l, n] \quad \text{for } 1 \leq k \leq N+2. \tag{32}$$

For each n, this equation is recognized as an $N + 2$ point DFT along the column $a[\cdot, n]$. Similar operations are used to compute the residue array $B[k, n]$.

2) Residue Multiplications: The second step of the algorithm is to perform the residue multiplications $z_k(x) = \langle a_k(x)b_k(x)\rangle_{m_k(x)}$ for $k = 1, 2, \cdots, N + 2$. In general, it is difficult to efficiently compute convolution modulo an arbitrary polynomial. However, the special moduli (25) allow the products to be computed by circular convolutions through a simple transformation as follows. First premultiply $a_k[n]$ and $b_k[n]$ by a phase shift,

$$\tilde{a}_k[n] = a_k[n]\, W_S^{-(k-1)n} \Leftrightarrow \tilde{A}[k, n] = A[k, n]\, W_S^{-(k-1)n}$$
$$\tilde{b}_k[n] = b_k[n]\, W_S^{-(k-1)n} \Leftrightarrow \tilde{B}[k, n] = B[k, n]\, W_S^{-(k-1)n}. \tag{33}$$

Then convolve these new residues using an R-point circular convolution,

$$\tilde{z}_k(x) = [\tilde{a}_k(x)\tilde{b}_k(x)\rangle_{(x^R-1)}$$
$$\Updownarrow$$
$$\tilde{Z}[k, n] = \sum_{i=0}^{R-1} \tilde{A}[k, i]\tilde{B}[k, \langle n - i\rangle_R]. \tag{34}$$

Finally postmultiply to obtain the desired product residues

$$z_k[n] = \tilde{z}_k[n]\, W_S^{(k-1)n} \Leftrightarrow Z[k, n] = \tilde{Z}[k, n]\, W_S^{(k-1)n}. \tag{35}$$

Many efficient circular convolution algorithms exist [16] and any one could be used to compute (34). A logical choice would be to use an FFT-based algorithm by taking R-point FFT's of each row, multiplying point by point, and then taking inverse R-point FFT's,

$$\tilde{Z}[k, \cdot] = \mathrm{FFT}_R^{-1}\, [\mathrm{FFT}_R[\tilde{A}[k, \cdot]] \cdot \mathrm{FFT}_R\, [\tilde{B}[k, \cdot]]]. \tag{36}$$

3) CRT Reconstruction: The next step in our algorithm is to reconstruct the polynomial $z(x)$ using all $N + 2$ residues. By assumption, this operation is computed reliably. It can be shown that for moduli (25), each polynomial $M_k^+(x)$ is sparse, with only one out of every R coefficients being nonzero,

$$M_k^+(x) = \frac{M^+(x)}{m_k(x)} = \sum_{i=0}^{N+1} W_{N+2}^{(k-1)(i+1)} x^{iR}. \tag{37}$$

Also, the $D_k^+(x)$ are constants,

$$D_k^+(x) = \frac{1}{N+2} W_{N+2}^{-(k-1)}. \tag{38}$$

Since deg $z_k(x) < R$, the CRT reconstruction is formed from nonoverlapped, shifted, and scaled combinations of the $z_k(x)$. We find that

$$z(x) = \frac{1}{N+2} \sum_{k=1}^{N+2} \sum_{i=0}^{N+1} z_k(x) x^{iR}\, W_{N+2}^{i(k-1)}$$
$$\Updownarrow$$
$$z[n_2, n_2] = \frac{1}{N+2} \sum_{k=1}^{N+2} Z[k, n_2]\, W_{N+2}^{(k-1)n_1}. \tag{39}$$

This is similar to (32) and is recognized as $N + 2$ point inverse DFT's along each column $Z[\cdot, n_2]$.

4) Fast Fault Detection and Correction: If deg $z(x) < Q$ then no fault has occurred, and we are done, $c(x) = z(x)$. This corresponds to the last two rows of $z[n_1, n_2]$ being zero. If there are nonzero entries in the last two rows, then a fault $\phi_q(x)$ has occurred in some processor q. To locate the fault we must divide $z(x)$ by each $M_j^+(x)$ in turn, and check the leading R coefficients of the remainder $r_j(x) = \langle z(x)\rangle_{M_j^+(x)}$.

For this special choice of moduli, there is an even faster fault detection technique. It can be shown that an error $\phi_q[n_2]$ linearly perturbs each row of the output,

$$z[n_1, n_2] = \begin{cases} c[n_1, n_2] + \dfrac{W_{N+2}^{(q-1)n_1}}{N+2}\, \phi_q[n_2] \\ \qquad \text{for } n_1 = 0, \cdots, N-1 \\ \dfrac{W_{N+2}^{(q-1)n_1}}{N+2}\, \phi_q[n_2] \\ \qquad \text{for } n_1 = N, N+1. \end{cases} \tag{40}$$

This reveals an alternative, simpler method of locating the fault. Instead of computing and testing $N + C$ residues, we may use the correlation between nonzero samples of $z[N, n_2]$ and $z[N + 1, n_2]$ to quickly identify q,

$$\hat{q} = \left\langle \text{round}\left[\frac{N+2}{2\pi} \text{ARG}\, (z[N+1, n_2] \cdot z^*[N, n_2])\right]\right\rangle_{N+2} + 1 \tag{41}$$

where ARG (x) refers to the principal value, or angle, of the complex quantity x. Note that this approach is sensitive to computational noise which may corrupt the correlation and lead to incorrect fault diagnoses. This problem becomes more severe as $|\phi_q[n_2]|$ decreases since small perturbations greatly affect ARG $(z[N + 1, n_2]z^*[N, n_2])$. A more intelligent approach to estimating $\hat{q}$ is to use all the samples in rows $z[N, \cdot]$ and $z[N + 1, \cdot]$ rather than just two samples. This approach is pursued in detail in Section V.

C. Algorithm Summary

The computational steps involved in the FFT-based algorithm are summarized in Fig. 2. Close examination reveals that this procedure is similar to computing convolution using Cooley-Tukey FFT's [16] of length $(N + 2)R$. If Cooley-Tukey FFT's were used, we would first arrange the data into rows of 2-D matrices. Then compute the FFT of each column, multiply the array by twiddle factors, take an FFT of each row, and multiply these row FFT's. Then inverse FFT each row, multiply by twiddle factors, and inverse FFT each column. The only difference between a Cooley-Tukey FFT and our algorithm is that the initial column FFT's must be replaced with DFT's which compute each sample independently. This is necessary because the computation in each row must be done independently by a separate processor, and column FFT's would violate this partitioning of computation. Thus each processor must be loaded with the input sequences $a[n]$ and $b[n]$, and will evaluate only the single sample of each

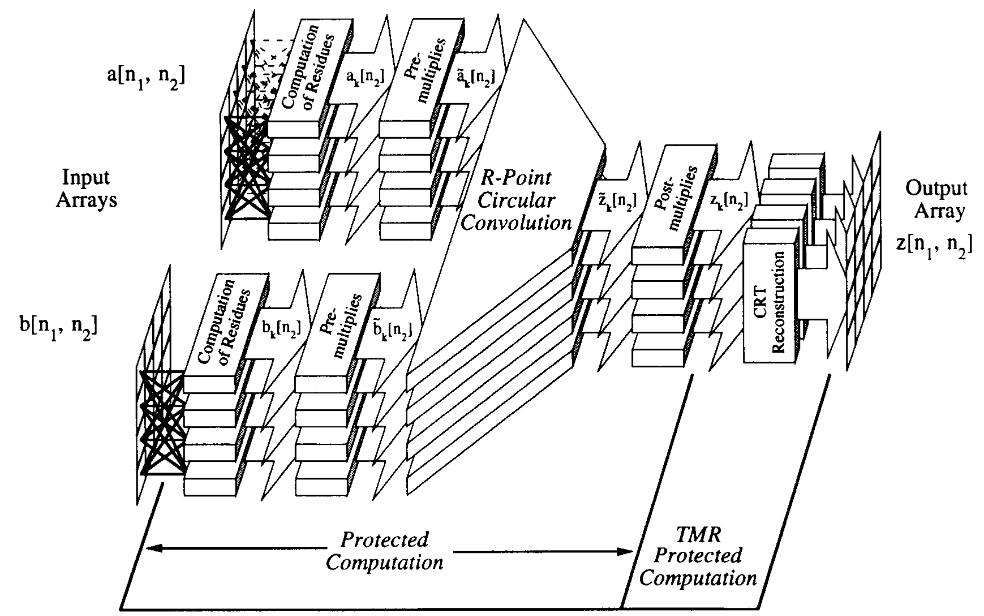


Fig. 2. Computational steps involved in fault-tolerant FFT-based algorithm. The example shown is of an 8-point convolution computed by four 4-point convolutions.

column DFT that it needs. FFT's are not efficient under these circumstances. In the CRT, however, the distribution of computation is not critical to the functioning of the algorithm since the CRT is computed reliably using TMR. Thus the $N + 2$ residue processors output all their data to the CRT processors, which then use column FFT's to compute $z(x)$.

Added insight can be gained by studying two extreme cases. First, when $N = 1$, operation is analogous to TMR. We compute an R-point convolution using three independent R-point convolutions. Another interesting case occurs when $R = 1$. Then $m_k(x) = (x - W_{N+2}^{-(k-1)})$ and $a_k(x)$ and $b_k(x)$ are constants equal to the value of the Fourier transforms of $a[n]$ and $b[n]$ at frequency $2\pi(k - 1)/(N + 2)$. This is equivalent to computing convolution by having each processor multiply a single DFT component. Adding two samples to the length of the convolution allows an error in any DFT component to be detected and corrected. This is similar to a method proposed by Wolf to protect communication channels from impulse noise [17]. He encodes sequences using DFT's and adds extra DFT coefficients for redundancy.

V. Generalized Likelihood Ratio Test

If infinite precision arithmetic is used with no rounding, then all computation will be exact and the fault detection/correction procedure discussed in Section IV-B4 would be sufficient. Unfortunately, processors must use fixed or floating point approximations to the complex numbers, and rounding and truncation errors occur. In this section we discuss how to distinguish between these small deviations and actual processor failures in an optimal manner.

A similar problem is addressed in [18], [19] in which A/D converter failures are diagnosed in the presence of quantization noise. A stochastic model of system behavior is developed, and converter failures detected and corrected using a generalized likelihood ratio test (GLRT). The GLRT determines the most likely converter failure and estimates the output using all available data. We apply a similar technique to the problem of diagnosing processor failures in our FFT-based algorithm.

We begin with (40) and add an extra term $\epsilon[n_1, n_2]$ to model the net effect of computational noise on each sample of the result,

$$z[n_1, n_2] = \begin{cases} c[n_1, n_2] + \dfrac{W_{N+2}^{(q-1)n_1}}{N+2}\phi_q[n_2] + \epsilon[n_1, n_2] \\ \quad \text{for } n_1 = 0, \cdots, N-1 \\ \dfrac{W_{N+2}^{(q-1)n_1}}{N+2}\phi_q[n_2] + \epsilon[n_1, n_2] \\ \quad \text{for } n_1 = N, N+1. \end{cases} \tag{42}$$

Let z, c, and ϕ_q represent the arrays $z[n_1, n_2]$, $c[n_1, n_2]$, and $\phi_q[n_2]$, respectively. We choose a set of hypotheses to model the behavior of our system. Let H_* represent the hypothesis that all processors are functioning properly, and let H_q represent the hypothesis of a failure in the qth processor, where $q = 1, \cdots, N + 2$. Also define P_* and P_q as the *a priori* probabilities of these events and assume that they are independent of data and fault.

Our basic approach is to compute the likelihood (probability) of observing the output z assuming that each hypothesis is true. The hypothesis with largest likelihood is most probable, and serves as our fault diagnosis. Unfortunately, the likelihoods depend on the correct output $\boldsymbol{c}$ and on the fault ϕ_q, if any, which are unknown. We therefore use a GLRT to jointly estimate the likelihoods and unknown parameters.

Define L_* to be the log likelihood of z and H_* conditioned on $\boldsymbol{c}$, maximized over all possible correct outputs $\boldsymbol{c}$,

$$\begin{aligned} L_* &= \max_{c} \log p(z, H_* \mid c) \\ &= \max_{c} \log p(z \mid H_*, c) + \log P_*. \end{aligned} \tag{43}$$

We employ log likelihoods rather than probabilities since they can be solved more easily. Both methods are equivalent since the logarithm is a monotonically increasing function. Similarly, define L_q as the log likelihood of z and H_q conditioned on $\boldsymbol{c}$ and ϕ_q, with $\boldsymbol{c}$ and ϕ_q set to their most likely values,

$$\begin{aligned} L_q &= \max_{c,\phi_q} \log p(z, H_q \mid c, \phi_q) \\ &= \max_{c,\phi_q} \log p(z \mid H_q, c, \phi_q) + \log P_q. \end{aligned} \tag{44}$$

We will compute L_* and L_q for $q = 1, \cdots, N + C$ and pick the largest. The largest likelihood corresponds to the most likely failure hypothesis given the observed data z. Also, the values $\hat{c}$ and $\hat{q}$ which maximize the likelihoods serve as estimates of the correct output and error.

The solution of the likelihood equations depends heavily on the probability distribution of the computational noise and cannot be solved in general. However, since $\epsilon[n_1, n_2]$ is the accumulated effect of many rounding operations, it is reasonable to model it as white, zero-mean Gaussian noise with variance σ_ϵ^2. We will also assume that $\epsilon[n_1, n_2]$ is independent of signal and fault,

$$p(\epsilon[n_1, n_2] \mid c, \phi_q) = p(\epsilon[n_1, n_2]) = N(0, \sigma_\epsilon^2). \tag{45}$$

We assume that the processors are uniform and have equal probability of failure. The GLRT is solved in Appendix B and it reduces to the following simple form. First compute the constant

$$L'_* = 4\sigma_\epsilon^2 \log \left(\frac{P_*}{P_q} \right). \tag{46}$$

This serves as a threshold which decides between H_* and all other hypotheses. Next, estimate the most likely failed processor given the observed data,

$$\hat{q} = \left\langle \text{round} \left[\frac{N+2}{2\pi} \text{ARG} \, (\rho_{N+1,N}) \right] \right\rangle_{N+2} + 1 \tag{47}$$

where $\rho_{i,j}$ is the total correlation between the ith and jth rows of $z[n_1, n_2]$,

$$\rho_{i,j} = \sum_{n_2=0}^{R-1} z[i, n_2] z^*[j, n_2]. \tag{48}$$

Then compute the log likelihood of a failure in processor $\hat{q}$,

$$L'_{\hat{q}} = \rho_{N,N} + \rho_{N+1,N+1} + 2 \, \text{Re} \, (\rho_{N,N+1} W_{N+2}^{(\hat{q}-1)}). \tag{49}$$

Compare $L'_{\hat{q}}$ with the threshold L'_*. If $L'_{\hat{q}}$ is smaller, we declare that no processor has failed and ascribe the deviation to computational noise. The estimate of the output, $\hat{c}[n_1, n_2]$, then equals

$$\hat{c}[n_1, n_2] = \begin{cases} z[n_1, n_2] & \text{for } n_1 = 0, \cdots, N-1 \\ 0 & \text{for } n_1 = N, N+1. \end{cases} \tag{50}$$

Otherwise, if $L'_{\hat{q}}$ is greater than L'_*, we declare that processor $\hat{q}$ has failed. We correct the output by first estimating the error

$$\begin{aligned} \hat{\phi}_{\hat{q}}[n_2] = \frac{N+2}{2} \Big[& z[N, n_2] W_{N+2}^{-(\hat{q}-1)N} \\ & + z[N+1, n_2] W_{N+2}^{-(\hat{q}-1)(N+1)} \Big] \end{aligned} \tag{51}$$

and then subtracting a phase shifted copy of this estimate from $z[n_1, n_2]$,

$$\hat{c}[n_1, n_2] = \begin{cases} z[n_1, n_2] - \dfrac{W_{N+2}^{(q-1)n_1}}{N+2} \hat{\phi}_{\hat{q}}[n_2] \\ \qquad \text{for } n_1 = 0, \cdots, N-1 \\ 0 \quad \text{for } n_1 = N, N+1. \end{cases} \tag{52}$$

This method improves performance over the simple method (41) because a total of $2R$ samples are used to identify the faulty processor instead of only two samples. It yields a more accurate estimate of $\phi_q[n_2]$ by reducing computational noise through averaging. The GLRT requires little additional computation; roughly $3R$ multiplications are needed to compute the correlations and roughly $3R + NR$ to correct the fault.

With a GLRT, error detection depends on the value of random computational noise, and therefore errors may not always be reliably detected. High noise levels can cause false alarms and small errors can go undetected. In practice, however, this is not a serious problem since all large errors are properly detected, and during a false alarm, the error estimate $\hat{\phi}_{\hat{q}}[n_2]$ is generally quite small, and the improperly applied error correction does not corrupt the output significantly.

The numerical value of the decision threshold L'_* depends on the statistics of the computational noise which we modeled as Gaussian random variables. In practice, the distribution of the computational noise depends heavily on details of the implementation (arithmetic precision, values of N and R, FFT routines used) and a Gaussian model may be inappropriate. Also note, if floating point arithmetic is used, then the actual computational noise may be correlated with the data, and it may be necessary to scale the threshold according to the magnitude of the input sequences [20]. It is best to use computer simulations to choose a threshold that achieves the desired false alarm probability.

When a faulty processor is detected, the fault is corrected and then several alternative courses of action may be taken. The processor may be monitored to see if the error disappears. If so, then the fault was only transient and normal operation can continue. If the error persists, the faulty processor can be shutdown and the system reconfigured as an $N + 1$ processor error detecting system. Alternatively, a standby processor might be switched in to replace the faulty processor.

VI. Fault-Tolerance Overhead

In this section we discuss the efficiency and fault coverage of the single error-correcting FFT-based system discussed in the previous section. We compare an unprotected NR-point convolution computed by a standard FFT-based algorithm, with a fault-tolerant NR-point convolution which is embedded in an $(N + 2)R$-point circular convolution. Our analysis focuses on how the polynomial RNS protects the computation involved in convolution and does not take into account the additional hardware and processing required for reliable interprocessor communication and I/O. We examine two quantities: overhead and coverage. Overhead is defined as the percentage of extra computation needed for fault tolerance relative to the unprotected algorithm. Coverage is the percentage of total computation protected by the polynomial RNS (the remaining computation is protected via TMR).

We divide the FFT-based algorithm into four steps. During step 1, we compute the residue arrays $A[k, n_2]$ and $B[k, n_2]$. Processor k computes row k of these arrays using DFT's as discussed in Section IV-C. In step 2, the residues are convolved using R-point FFT's. Step 3 is the CRT reconstruction using $N + 2$ point column FFT's. During step 4, error detection and correction are performed. Note that steps 3 and 4 are protected via TMR, and we will weight the computation required for these steps accordingly.

For simplicity, when considering the computational complexity of an algorithm, we count only the number of multiplications involved. This is a reasonable approximation, since for most FFT algorithms, the total number of operations is proportional to the number of multiplications. Let M_1, M_2, M_3, and M_4 be the number of multiplications in each step. Also, let M_u be the total number of multiplications in a standard unprotected NR-point convolution computed with FFT's. Using these definitions, our performance measures may be written as

$$\text{overhead} = \frac{M_1 + M_2 + 3(M_3 + M_4)}{M_u} - 1 \qquad (53)$$

$$\text{coverage} = \frac{M_1 + M_2}{M_1 + M_2 + M_3 + M_4}. \qquad (54)$$

As part of our calculations we must compare the number of multiplications in N and $N + 2$ point FFT's. This is difficult to do for arbitrary values of N and so we make a rough approximation. We assume that an L-point FFT requires $2L \log_2 L$ multiplications [16], even though L may not be a power of 2. The majority of multiplications occur in the FFT's, and we ignore twiddle and transform coefficient multiplications. With these assumptions we obtain

$$M_1 \approx 2R(N + 2)^2 \qquad (55)$$

$$M_2 \approx 6R(N + 2) \log_2 R \qquad (56)$$

$$M_3 \approx 2R(N + 2) \log_2 (N + 2) \qquad (57)$$

$$M_4 \approx 6R + RN \qquad (58)$$

$$M_u \approx 6RN \log_2 (RN). \qquad (59)$$

TABLE I
Overhead of FFT-Based Algorithm

		R		
		64	256	1024
$N + 2$	4	195%	174%	161%
	8	93%	82%	74%
	16	86%	74%	65%
	32	117%	100%	88%
	64	192%	165%	145%

Using these approximations, we evaluated our measures for several values of N and R and the results are shown in Tables I and II. A good reference with which to compare the calculated overheads is that required by a MR system offering a similar level of fault protection. A single error-correcting MR system requires triplication, that is, 200% overhead. Our method, on the other hand, achieves this level of fault tolerance with as little as 70% overhead, a substantial savings. We find that overhead varies strongly with N and reaches a minimum at $N + 2 = 16$. Two separate factors contribute to this behavior. First, if N is small, the two additional row convolutions make up a sizable portion of the total computation, and thus the overhead is high. Second, when N is large, the DFT's in step 1 require significant amounts of computation since the number of operations involved grows as $(N + 2)^2$. Efficient operation occurs between these two extremes.

We also find that the polynomial RNS protects the majority of computation, as demonstrated by the high coverage values in Table II. In most instances, over 90% of the computation occurs in the residue channels. Thus, the bulk of computation is covered by the low cost arithmetic code, while only the remaining 10% need be protected by more expensive MR.

VII. Conclusion

In this paper we presented a new approach to protecting linear convolution which was considerably cheaper than traditional methods based on modular redundancy. Our algorithm used a polynomial residue number system (RNS), which is the underlying structure of the Winograd convolution algorithm. Computation is decomposed into independent, parallel residue channels, and redundancy incorporated by adding extra residue channels. Analogous to integer RNS fault-tolerance schemes, single errors can

TABLE II
FAULT COVERAGE OF FFT-BASED ALGORITHM

		R		
		64	256	1024
$N+2$	4	88%	90%	92%
	8	87%	90%	91%
	16	88%	90%	91%
	32	90%	91%	92%
	64	93%	93%	94%

be detected by adding one extra modulus, and corrected using two extra moduli. However, we do not encounter many of the problems associated with integer RNS.

We derived conditions on the redundant moduli such that a desired level of fault-tolerance may be achieved, and presented an algorithm for detecting and correcting multiple processor failures. Importantly, we presented a specific set of moduli polynomials which yielded an efficient FFT-based algorithm. The effects of computational noise were handled using a generalized likelihood ratio test, and the resulting fault detection/correction algorithm was both fast and accurate.

The parallel nature of our algorithm makes it ideal for implementation on a multiprocessor system. We distribute computation such that each residue channel is computed by a separate processor. The low cost polynomial RNS coding scheme is able to protect the bulk of computation, roughly 90%, while the remaining 10% is protected via more expensive triple modular redundancy. This hybrid approach yields a system fully protected against any single failure at a cost significantly lower than a fully redundant implementation. We are able to achieve this level of reliability with only 70% overhead, compared with 200% needed for triple modular redundancy.

Variations of our scheme are possible which allow a wider range of operations to be protected. Addition and subtraction of polynomials can also be protected, and we are not limited to protecting individual ring operations, but can protect several with a single residue encoding and a single error test. The ony requirement is that the length of the final result be constrained. Errors may be detected and corrected in the same manner.

APPENDIX A
PROOF OF THEOREMS

This Appendix contains proofs of the theorems which were presented in Section III-B. We assume throughout that λ failures occur in processors $\{k_1, \cdots, k_\lambda\}$ and that the outputs of the residue processors have value (15). The result of the reliable CRT reconstruction will then be (19). We also assume that the moduli satisfy (20) for some value of D, and that $D = \alpha + \beta$ for $\alpha \geq \beta \geq 0$.

If no errors have occurred, $\lambda = 0$, then $z(x) = c(x)$ and thus deg $z(x) < Q$. This proves Theorem 1a. To prove 1b, we use the following lemma:

Lemma 1: Let $\phi_{k_1}(x) \neq 0, \cdots, \phi_{k_\lambda}(x) \neq 0$ be any set of $1 \leq \lambda \leq D$ polynomials with deg $\phi_{k_i}(x) <$ deg $m_{k_i}(x)$. Then:

$$\deg\left[\sum_{i=1}^{\lambda} \langle \phi_{k_i}(x) D_{k_i}^+(x)\rangle_{m_{k_i}(x)} M_{k_i}^+(x)\right] \geq Q. \qquad (60)$$

Proof of Lemma 1: Let

$$\tilde{\phi}_{k_i}(x) = \langle \phi_{k_i}(x) D_{k_i}^+(x)\rangle_{m_{k_i}(x)}.$$

Since deg $\phi_{k_i}(x) <$ deg $m_{k_i}(x)$ and $\phi_{k_i}(x) \neq 0$, and since $D_{k_i}^+(x)$ and $m_{k_i}(x)$ are coprime, we know that $\tilde{\phi}_{k_i}(x) \neq 0$. Then

$$\sum_{i=1}^{\lambda} \tilde{\phi}_{k_i}(x) M_{k_i}^+(x)$$
$$= \left[\frac{M^+(x)}{m_{k_1}(x) \cdots m_{k_\lambda}(x)}\right] [\tilde{\phi}_{k_1}(x) m_{k_2}(x) \cdots m_{k_\lambda}(x)$$
$$+ \tilde{\phi}_{k_2}(x) m_{k_1}(x) m_{k_3}(x) \cdots m_{k_\lambda}(x) + \cdots$$
$$+ \tilde{\phi}_{k_\lambda}(x) m_{k_1}(x) \cdots m_{k_{\lambda-1}}(x)]. \qquad (61)$$

The first term on the right-hand side has degree $\geq Q$ by (20). The second term cannot be zero because it cannot be evenly divided by any of the polynomials $m_{k_1}(x), \cdots, m_{k_\lambda}(x)$. Thus the degree of the right-hand side is $\geq Q$ and Lemma 1 is true. □

Since Lemma 1 is true, we know that the error term in (19) will always have degree $\geq Q$. Since deg $c(x) < Q$, $z(x)$ must have degree $\geq Q$ if between 1 and D failures occur. This proves Theorem 1. □

Theorem 2 is contained in Theorem 3, and thus proving Theorem 3 is sufficient. We rely upon the following lemma:

Lemma 2: Assume that $\lambda \leq \alpha$ errors occur in processors $\{k_1, \cdots, k_\lambda\}$. Let $\{j_1, \cdots, j_p\}$ be any set of p processors with $p \leq \beta$. Then

$$\deg [\langle z(x)\rangle_{M^+_{j_1,\cdots,j_p}(x)}] < Q \quad \text{if and only if}$$

$$\{k_1, \cdots, k_\lambda\} \subset \{j_1, \cdots, j_p\}$$

where

$$M^+_{j_1,\cdots,j_p}(x) = \frac{M^+(x)}{m_{j_1}(x) \cdots m_{j_p}(x)}.$$

Proof of Lemma 2: Let $r_{j_1,\cdots,j_p}(x) = \langle z(x)\rangle_{M^+_{j_1,\cdots,j_p}(x)}$. We prove this lemma in two parts. First, we show that deg $r_{j_1,\cdots,j_p}(x) \geq Q$ if $\{k_1, \cdots, k_\lambda\} \not\subset \{j_1, \cdots, j_p\}$. Then, we show that deg $r_{j_1,\cdots,j_p}(x) < Q$ if $\{k_1, \cdots, k_\lambda\} \subset \{j_1, \cdots, j_p\}$.

Using (61), write the result of the reliable CRT reconstruction in the form

$$z(x) = c(x) + \frac{M^+(x)}{m_{k_1}(x) \cdots m_{k_\lambda}(x)} \Phi(x) \qquad (62)$$

where $\Phi(x) \neq 0$ and $\Phi(x)$ is coprime to $m_{k_1}(x), \cdots, m_{k_\lambda}(x)$. Then, using the division algorithm for polynomials, write this as

$$z(x) = r_{j_1, \cdots, j_p}(x) + \frac{M^+(x)}{m_{j_1}(x) \cdots m_{j_p}(x)} Q(x) \quad (63)$$

where $Q(x)$ and $r_{j_1, \cdots, j_p}(x)$ are the quotient and remainder when $z(x)$ is divided by $M^+_{j_1, \cdots, j_p}(x)$. Suppose $\{j_1, \cdots, j_p\} \cap \{k_1, \cdots, k_\lambda\} = \{m_1, \cdots, m_r\}$ where $r \leq \min(\lambda, p)$, and suppose $\{j_1, \cdots, j_p\} \cup \{k_1, \cdots, k_\lambda\} = \{n_1, \cdots, n_s\}$ where $s \leq \lambda + p$. Let $\{\tilde{j}_1, \cdots, \tilde{j}_{p-r}\} = \{j_1, \cdots, j_p\} - \{m_1, \cdots, m_r\}$ represent the indices of $\{j_1, \cdots, j_p\}$ which do not appear in $\{k_1, \cdots, k_\lambda\}$. Similarly, let $\{\tilde{k}_1, \cdots, \tilde{k}_{\lambda-r}\} = \{k_1, \cdots, k_\lambda\} - \{m_1, \cdots, m_r\}$. Equating (62) and (63) gives

$$\begin{aligned} c(x) &- r_{j_1, \cdots, j_p}(x) \\ &= \left[\frac{M^+(x)}{m_{n_1}(x) \cdots m_{n_s}(x)}\right] [Q(x) m_{\tilde{k}_1}(x) \cdots m_{\tilde{k}_{\lambda-r}}(x) \\ &\quad - \Phi(x) m_{\tilde{j}_1}(x) \cdots m_{\tilde{j}_{p-r}}(x)]. \end{aligned} \quad (64)$$

Since $\lambda \leq \alpha$ and $p \leq \beta$, we know that $s \leq D$. Therefore by (20), the first term on the right-hand side has degree $\geq Q$. When $\{k_1, \cdots, k_\lambda\} \not\subset \{j_1, \cdots, j_p\}$, then $\{\tilde{k}_1, \cdots, \tilde{k}_{\lambda-r}\}$ will be nonempty. Then, since $\Phi(x)$ is coprime to $m_{k_1}(x), \cdots, m_{k_\lambda}(x)$, the second term on the right-hand side cannot be zero. Thus the right-hand side has degree $\geq Q$ and deg $r_{j_1, \cdots, j_p}(x) \geq Q$ if $\{k_1, \cdots, k_\lambda\} \not\subset \{j_1, \cdots, j_p\}$.

Now assume that $\{k_1, \cdots, k_\lambda\} \subset \{j_1, \cdots, j_p\}$. Begin with (62) and expand the second term

$$z(x) = c(x) + \sum_{i=1}^{\lambda} \tilde{\phi}_{k_i}(x) M^+_{k_i}(x). \quad (65)$$

Now take the residue when this is divided by $M^+_{j_1, \cdots, j_p}(x)$,

$$\begin{aligned} r_{j_1, \cdots, j_p}(x) &= \langle c(x) \rangle_{M^+_{j_1, \cdots, j_p}(x)} \\ &\quad + \sum_{i=1}^{\lambda} \langle \tilde{\phi}_{k_i}(x) M^+_{k_i}(x) \rangle_{M^+_{j_1, \cdots, j_p}(x)}. \end{aligned} \quad (66)$$

Since deg $M^+_{j_1, \cdots, j_p}(x) >$ deg $c(x)$, $\langle c(x) \rangle_{M^+_{j_1, \cdots, j_p}(x)} = c(x)$. Also, since $M^+_{j_1, \cdots, j_p}(x)$ divides $M^+_{k_i}(x)$,

$$\langle \tilde{\phi}_{k_i}(x) M^+_{k_i}(x) \rangle_{M^+_{j_1, \cdots, j_p}(x)} = 0 \qquad \text{for } i = 1, \cdots, \lambda.$$

The above equation then reduces to

$$r_{j_1, \cdots, j_p}(x) = c(x). \quad (67)$$

Thus $r_{j_1, \cdots, j_p}(x) = c(x)$ and deg $r_{j_1, \cdots, j_p}(x) < Q$ when $\{k_1, \cdots, k_\lambda\} \subset \{j_1, \cdots, j_p\}$. This proves Lemma 2. □

To show that the procedure in Theorem 3 works properly, we consider two cases. First assume that no failures occurred, $\lambda = 0$. Then by Theorem 1, $z(x) = c(x)$ and deg $z(x) < Q$, and the procedure would stop in step 1. Second assume that $\lambda \leq \alpha$ errors occurred. Lemma 2 guarantees that deg $r_{j_1, \cdots, j_p}(x) \geq Q$ if $\{k_1, \cdots, k_\lambda\} \not\subset \{j_1, \cdots, j_p\}$. Since we are checking all possible combinations of p processors, starting with $p = 1$ and continuing until $p = \beta$, deg $r_{j_1, \cdots, j_p}(x)$ will be less than Q if and only if $\lambda \leq \beta$, $p = \lambda$, and $\{j_1, \cdots, j_p\} = \{k_1, \cdots, k_\lambda\}$. Then by Lemma 2, $c(x) = r_{j_1, \cdots, j_p}(x)$ is the correct solution. Otherwise, if $\beta + 1 \leq \lambda \leq \alpha$ failures occurred, then $\{k_1, \cdots, k_\lambda\} \not\subset \{j_1, \cdots, j_p\}$ and we continue to step 4. This set of faults is uncorrectable. □

Appendix B
Solution of Likelihood Equations

In this Appendix we solve the likelihood equations discussed in Section V. We begin with (43) and (44) and model computational noise as a zero mean Gaussian random process (45). With this assumption, the likelihoods become

$$L_* = \max_{c} \left[\eta_* - \frac{1}{2\sigma_\epsilon^2} \sum_{n_1=0}^{N+1} \sum_{n_2=0}^{R-1} |z[n_1, n_2] - c[n_1, n_2]|^2 \right] \quad (68)$$

$$L_q = \max_{c, \phi_q} \left[\eta_q - \frac{1}{2\sigma_\epsilon^2} \sum_{n_1=0}^{N+1} \sum_{n_2=0}^{R-1} \left| z[n_1, n_2] - c[n_1, n_2] - \frac{W_{N+2}^{(q-1)n_1}}{N+2} \phi_q[n_2] \right|^2 \right] \quad (69)$$

where η_* and η_q are constants,

$$\eta_* = \log P_* - \tfrac{1}{2} S \log(2\pi\sigma_\epsilon^2) \quad (70)$$

$$\eta_q = \log P_q - \tfrac{1}{2} S \log(2\pi\sigma_\epsilon^2). \quad (71)$$

We start by maximizing L_* over $\mathbf{c}$ to obtain $\hat{c}[n_1, n_2]$, an estimate of the output given that H_* is true. Since $c[n_1, n_2]$ is zero for rows N and $N + 1$, (68) can be rewritten as

$$L_* = \max_{c} \left[\eta_* - \frac{1}{2\sigma_\epsilon^2} \left\{ \sum_{n_1=0}^{N-1} \sum_{n_2=0}^{R-1} |z[n_1, n_2] - c[n_1, n_2]|^2 + \sum_{n_1=N}^{N+1} \sum_{n_2=0}^{R-1} |z[n_1, n_2]|^2 \right\} \right]. \quad (72)$$

Maximizing over $\mathbf{c}$ we obtain

$$\hat{c}[n_1, n_2] = \begin{cases} z[n_1, n_2] & \text{for } n_1 = 0, \cdots, N-1 \\ 0 & \text{for } n_1 = N, N+1. \end{cases} \quad (73)$$

Substituting $\hat{c}[n_1, n_2]$ into (72) yields the likelihood that hypothesis H_* is true,

$$L_* = \eta_* - \frac{1}{2\sigma_\epsilon^2} \sum_{n_1=N}^{N+1} \sum_{n_2=0}^{R-1} |z[n_1, n_2]|^2. \quad (74)$$

Computing L_q is more difficult since we must estimate both $\boldsymbol{c}$ and ϕ_q. Maximizing (69) over $\boldsymbol{c}$ we obtain

$$\hat{c}_I[n_1, n_2] = \begin{cases} z[n_1, n_2] - \dfrac{W_{N+2}^{(q-1)n_1}}{N+2}\phi_q[n_2] \\ \qquad \text{for } n_1 = 1, \cdots, N-1 \\ 0 \qquad \text{for } n_1 = N, N+1. \end{cases} \tag{75}$$

(Note that we call this $\hat{c}_I[n_1, n_2]$ to emphasize that it is an intermediate step in the maximization process, and not the final estimate of the correct output.) Substituting $\hat{c}_I[n_1, n_2]$ into (69), the log likelihood function becomes

$$L_q = \max_{\phi_q}\left[\eta_q - \frac{1}{2\sigma_\epsilon^2}\sum_{n_1=N}^{N+1}\sum_{n_2=0}^{R-1}\left| \cdot z[n_1, n_2] - \frac{W_{N+2}^{(q-1)n_1}}{N+2}\phi_q[n_2]\right|^2\right]. \tag{76}$$

Maximizing over ϕ_q and keeping in mind that complex quantities are involved, we obtain

$$\hat{\phi}_q[n_2] = \frac{N+2}{2}\left[z[N, n_2]W_{N+2}^{-(q-1)N} + z[N+1, n_2]W_{N+2}^{-(q-1)(N+1)}\right]. \tag{77}$$

The error estimate $\hat{\phi}_q[n_2]$ is formed by averaging the last two rows of $z[n_1, n_2]$ with appropriate phase shifts and a scale factor of $N + 2$. Substituting $\hat{\phi}_q[n_2]$ for $\phi_q[n_2]$ in (75) gives the estimate of the correct output,

$$\hat{c}[n_1, n_2] = \begin{cases} z[n_1, n_2] - \dfrac{W_{N+2}^{(q-1)n_1}}{N+2}\hat{\phi}_q[n_2] \\ \qquad \text{for } n_1 = 1, \cdots, N-1 \\ 0 \qquad \text{for } n_1 = N, N+1. \end{cases} \tag{78}$$

Finally, substituting (78) and (77) into (69), and after some algebra, we obtain the likelihood of hypothesis H_q,

$$L_q = \eta_q - \frac{1}{2\sigma_\epsilon^2}\sum_{n_1=N}^{N+1}\sum_{n_2=0}^{R-1}|z[n_1, n_2]|^2 + \frac{1}{2\sigma_\epsilon^2}\frac{2}{(N+2)^2}\sum_{n_2=0}^{R-1}|\hat{\phi}_q[n_2]|^2. \tag{79}$$

The likelihood equations can be further simplified by assuming that the failure probabilities P_q are the same for $q = 1, \cdots, N + 2$, and by using relative likelihoods defined by

$$L_k' = 4\sigma_\epsilon^2\left[L_k - \eta + \frac{1}{2\sigma_\epsilon^2}\sum_{n_1=N}^{N+1}\sum_{n_2=0}^{R-1}|z[n_1, n_2]|^2\right]. \tag{80}$$

The likelihoods then reduce to

$$L_*' = 4\sigma_\epsilon^2 \log\frac{P_*}{P_q} \tag{81}$$

and

$$L_q' = \rho_{N,N} + \rho_{N+1,N+1} + 2\,\mathrm{Re}\,(\rho_{N,N+1}W_{N+2}^{(q-1)}) \tag{82}$$

where $\rho_{i,j}$ is the total correlation between the ith and jth rows of $z[n_1, n_2]$ and is defined in (48).

We can simplify the hypothesis testing procedure by solving directly for the value of q which maximizes (82) rather than computing each L_q'. This yields

$$\hat{q} = \left\langle \text{round}\left[\frac{N+2}{2\pi}\,\mathrm{ARG}\,(\rho_{N+1,N})\right]\right\rangle_{N+2} + 1. \tag{83}$$

Our fault test then proceeds as follows. First, compute the constant L_*'. Then compute $\hat{q}$ and $L_{\hat{q}}'$ using (8.) and (82). If $L_*' > L_{\hat{q}}'$, we declare that no processor has failed and ascribe the nonzero samples in rows $z[N, \cdot]$ and $z[N + 1, \cdot]$ to computational noise. Otherwise, we declare that processor $\hat{q}$ has failed and correct the fault using (77) and (78).

References

[1] D. Chin, J. Passe, F. Bernard, H. Taylor, and S. Knight, "The Princeton engine: A real-time video system simulator," *IEEE Trans. Consumer Electron.*, vol. 34, pp. 285-297, May 1988.

[2] D. A. Rennels, "Fault-tolerant computing—concepts and examples," *IEEE Trans. Comput.*, vol. C-33, pp. 1116-1129, Dec. 1984.

[3] J. von Neuman, *Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components.* Princeton University Press, 1956, pp. 43-98.

[4] J. A. Abraham, "Fault-tolerance techniques for highly parallel signal processing architectures," *Proc. SPIE Int. Soc. Opt. Eng.*, vol. 614, pp. 49-65, 1986.

[5] J.-Y. Jou and J. A. Abraham, "Fault-tolerant matrix arithmetic and signal processing on highly concurrent computing structures," *Proc. IEEE*, vol. 74, pp. 732-741, May 1986.

[6] J.-Y. Jou and J. A. Abraham, "Fault-tolerant FFT networks," *IEEE Trans. Comput.*, vol. 37, pp. 548-561, May 1988.

[7] W. S. Song and B. R. Musicus, "Fault-tolerant architecture for a parallel digital signal processing machine," in *Proc. 1987 IEEE Int. Conf. Comput. Design: VLSI in Comput. and Processors*, Oct. 1987, pp. 385-390.

[8] W. S. Song, "A fault-tolerant multiprocessor architecture for digital signal processing applications," Ph.D. dissertation, M.I.T., Jan. 1989.

[9] G. R. Redinbo, "System level reliability in convolution computations," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. . 7, pp. 1241-1252, Aug. 1989.

[10] T. G. Marshall, Jr., "Coding of real number sequences for error correction: A digital signal processing problem," *IEEE J. Select. Areas Commun.*, vol. SAC-2, pp. . 81-. 92, Mar. 1984.

[11] J. H. McClellan and C. M. Rader, *Number Theory in Digital Signal Processing.* Englewood-Cliffs, NJ: Prentice-Hall, 1979.

[12] I. N. Herstein, *Topics in Algebra.* New York: Wiley, 1975.

[1.] R. E. Blahut, *Fast Algorithms for Digital Signal Processing.* Reading, MA: Addison-Wesley, 1985.

[14] R. E. Blahut, *Theory and Practice of Error Control Codes.* Reading, MA: Addison-Wesley, 1984.

[15] M. A. Soderstrand, W. K. Jenkins, G. A. Jullien, and F. J. Taylor, *Residue Number System Arithmetic: Modern Applications in Digital Signal Processing.* New York: IEEE Press, 1986.

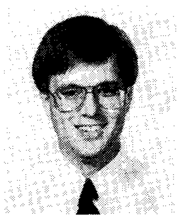
[16] C. S. Burrus, *DFT/FFT and Convolution Algorithms.* New York: Wiley Interscience, 1985.

[17] J. K. Wolf, "Redundancy, the discrete Fourier transform, and impulse noise cancellation," *IEEE Trans. Commun.*, vol. COM-. 1, pp. 458-461, Mar. 198. .

[18] P. E. Beckmann and B. R. Musicus, "Fault-tolerant round-robin A/D converter systems," *IEEE Trans. Circuits Syst.*, vol. . 8, pp. 1420-1429, Dec. 1991.

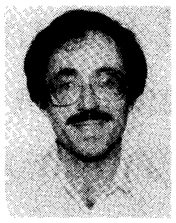
[19] P. E. Beckmann and B. R. Musicus, "Fault-tolerant round-robin A/D converter systems," RLE-Tech Rep. 561, M.I.T., Cambridge, MA, Dec. 1990.

[20] A. V. Oppenheim and C. J. Weinstein, "Effects of finite register length in digital filtering and the fast Fourier transform," *Proc. IEEE*, vol. 8, pp. 957-976, Aug. 1972.



Paul E. Beckmann (S'90–M'92) received the S.B. and S.M. degrees in electrical engineering in 1989, and the Ph.D. degree in 1992, all from the Massachusetts Institute of Technology, Cambridge.

He held a Rockwell Doctoral Fellowship from 1989 to 1992 while working in the M.I.T. Digital Signal Processing Group. Currently, he is a Research Engineer with Bose Corporation, Framingham, MA. His main area of research is in signal processing algorithms and architectures for consumer and professional audio products.



Bruce R. Musicus (S'77–M'78) received the S.B. degree from Harvard in 1975, the M.S. and E.E. degrees from M.I.T. in 1979, and the Ph.D. degree from M.I.T. in 1982.

He was an Associate Professor of Electrical Engineering and Computer Science at M.I.T., working in the areas of signal processing algorithms and architectures. He has taught courses both at M.I.T. and in industry on topics such as real-time computer systems, digital computer design, digital signal processing, and recursive filtering. Currently he is at Bolt, Beranek & Newman, working on sonar system design, signal processing, and speech recognition.