Fast Convolution Using Fermat Number Transforms with Applications to Digital Filtering

RAMESH C. AGARWAL AND CHARLES S. BURRUS, MEMBER, IEEE

Abstract—**The structure of transforms having the convolution property is developed. A particular transform is proposed that is defined on a finite ring of integers with arithmetic carried out modulo Fermat numbers. This Fermat number transform (FNT) is ideally suited to digital computation, requiring on the order of $N \log N$ additions, subtractions and bit shifts, but no multiplications. In addition to being efficient, the Fermat number transform implementation of convolution is exact, i.e., there is no roundoff error. There is a restriction on sequence length imposed by word length but multidimensional techniques are discussed which overcome this limitation. Results of an implementation on the IBM 370/155 are presented and compared with the fast Fourier transform (FFT) showing a substantial improvement in efficiency and accuracy.**

I. INTRODUCTION

THE convolution property of certain transforms can be used to efficiently compute the cyclic convolution of two long discrete sequences. One such transform is the discrete Fourier transform (DFT) with the fast Fourier transform (FFT) implementation [1] which operates on sequences in the complex number field. To compute the DFT of a sequence of length N requires on the order of $(N/2) \log_2 (N/2)$ complex multiplications, which often take most of the computation time. In addition, the FFT implementation of cyclic convolution introduces significant amounts of roundoff error [2], thus deteriorating the signal-to-noise ratio. When working with digital machines, the data are available only with some finite precision and, therefore, without loss of generality, the data can be considered to be integers with some upper bound. In this digital domain, the complex number field of the continuous domain can be replaced with a finite field or, more generally, a finite ring of integers with addition and multiplication modulo some integer F. In this ring, using transforms similar to the DFT, the cyclic convolution can be performed very efficiently and without any roundoff error. One such transform presented here is a number theoretic transform which operates in the ring of integers modulo a Fermat number. Computation of this transform of length N, analogous to an FFT, requires on the order of $N \log_2 N$ additions, bit shifts, and subtractions, but no multiplication. This is considerably simpler than the FFT.

Knuth [3] has proposed the use of transforms in finite fields. Pollard [4] discussed transforms having the cyclic convolution property in a finite field and also gives the conditions for having transforms with the cyclic convolution property in a finite ring of integers. Good [5] also mentioned the use of transforms in a finite ring of integers. Schonhage and Strassen [6] defined transforms having the cyclic convolution property modulo a Fermat number and discussed their application to fast multiplication of very large integers. Knuth [7] elaborated on the work of Schonhage and Strassen. Nicholson [8] presented an algebraic theory of finite Fourier transforms in any ring and established fast FFT-type algorithms to compute these transforms. Rader [9], [10] proposed finite transforms in rings of integers modulo both Mersenne and Fermat numbers. He first proposed the application to digital signal processing, showed that the transforms could be calculated using only additions and bit shifting, showed the word-length constraint, and suggested two-dimensional transforms as a possible relaxation of that constraint. Agarwal and Burrus [11] defined Fermat transforms and also proposed their application for fast digital convolution.

The purpose of this paper is to develop the general structure of transforms with the convolution property and then present the number theoretic transform modulo a Fermat number as a practical scheme for use with digital computation. The properties are then discussed for the implementation of digital filters and the results of a software implementation on an IBM 370/155 are given. It is believed that the presentation here is different from previous works and gives more insight into these transforms. One result of this approach allows the maximum length of sequences that can be convolved to be increased by a factor of 2 over what had previously been reported in [10].

The choice of terminology in a new area is always difficult. In this paper the general name, number-theoretic transform, is used for any transform using number-theoretic concepts in its definition. The two particular transforms with arithmetic modulo Mersenne and Fermat numbers are called Mersenne number transforms (MNT's) and Fermat number transforms (FNT's). The particular MNT's and FNT's with a basis function $\alpha = 2$ are called Rader transforms (RT's).

These transforms are truly digital transforms, taking into account the quantization in amplitude and the finite precision of digital signals. They bear the same relation to digital signal as the DFT does to discrete-time or sampled data signals and the Fourier or Laplace transforms do to continuous-time signals. In the same manner that the relation of discrete-time signals to continuous-

Manuscript received July 16, 1973; revised November 19, 1973. This work was supported by NSF Grant GK-23697.

The authors are with the Department of Electrical Engineering, Rice University, Houston, Tex. 77001.

time signals through sampling involves a possible folding or aliasing in the frequency domain, the relation of calculations with the DFT to calculations with the number theoretic transforms involves a possible folding of the amplitude that must be taken into account.

In Section II, the structures of transforms having the cyclic and the noncyclic convolution properties are developed. It is shown that what we call the DFT structure is the only structure which supports the cyclic convolution property and any transform having the DFT structure will have the cyclic convolution property. In Section III, the necessary and sufficient conditions for the existence of transforms in a ring of integers modulo of an integer F are established. In Section IV, FNT's are defined which, computationally, appear to be the best transforms for implementing convolution. In Section V, reference is given to a two-dimensional transform implementation for long one-dimensional convolution. In Section VI, a hardware implementation to compute the FNT is suggested. Comparison of the FNT with the FFT implementation of the DFT, to implement convolution, is made in Section VII. In Section VIII, a software implementation of the FNT on an IBM 370/155 is presented and timing comparisons are made with the FFT. For cyclic convolution lengths up to 256, FNT implementations are faster by a factor of 3 to 5 over the best FFT implementations which make use of the symmetry of the DFT for real data. For computers with slower multiplication, the advantages look even better. Finally, several generalizations and applications are presented.

II. THE STRUCTURE OF TRANSFORMS HAVING THE CONVOLUTION PROPERTY

In this section the structure of transforms having the cyclic convolution property, i.e., the transform of cyclic convolution of two sequences is equal to the product of their transforms, is established. The class of transforms considered is the set of linear nonsingular (invertible) transforms which map a sequence of length N to another sequence of length N. It is shown that what we call the DFT structure is the only structure which supports the cyclic convolution property and, moreover, any transform having the DFT structure will have the cyclic convolution property.

Let x_n and h_n, $n = 0,1,\cdots,N-1$, be two sequences which are to be cyclically convolved as given by

$$y_n = x_n * h_n = \sum_{k=0}^{N-1} x_k h_{n-k} \qquad n = 0,1,\cdots N-1. \quad (1)$$

This definition assumes the sequences are periodically extended with period N or the indices are evaluated modulo N. Here we will describe sequences as vectors of length N. Since only linear invertible transforms are considered, they can be represented by an $N \times N$ nonsingular matrix T whose elements are $t_{k,m}$, $k,m = 0,1,\cdots, N-1$. Let capital letters denote the transformed sequences, i.e.,

$$\boldsymbol{X} = T\boldsymbol{x}$$

$$\boldsymbol{H} = T\boldsymbol{h}$$

$$\boldsymbol{Y} = T\boldsymbol{y}.$$

Consider the conditions on $t_{k,m}$'s so that

$$\boldsymbol{Y} = \boldsymbol{X} \otimes \boldsymbol{H} \quad (3)$$

where $\otimes$ denotes term by term multiplication of the vectors.

Equations (2) and (1) are combined to give

$$Y_k = \sum_{n=0}^{N-1} t_{k,n} y_n = \sum_{n=0}^{N-1} t_{k,n} \sum_{m=0}^{N-1} x_m h_{n-m}$$

$$= \sum_{m=0}^{N-1} \sum_{l=0}^{N-1} x_m h_m t_{k,m+l}$$

$$X_k = \sum_{m=0}^{N-1} t_{k,m} x_m$$

$$H_k = \sum_{l=0}^{N-1} t_{k,l} h_l \qquad k = 0,1,\cdots,N-1. \quad (4)$$

The individual terms in (3) can be rewritten as

$$Y_k = X_k H_k$$

which, using (4), becomes

$$\sum_{m=0}^{N-1} \sum_{l=0}^{N-1} x_m h_l t_{k,m+l} = \sum_{m=0}^{N-1} \sum_{l=0}^{N-1} x_m h_l t_{k,m} t_{k,l}. \quad (5)$$

Matching the multiples of $x_m h_l$ on both sides of (5) gives

$$t_{k,m+l} = t_{k,m} t_{k,l} \qquad k,l,m = 0,1,\cdots,N-1. \quad (6)$$

Repeated applications of (6) gives

$$t_{k,m} = t_{k,1}{}^m \qquad k,m = 0,1,\cdots,N-1. \quad (7)$$

And, since the convolution is cyclic, the indices in (6) are added modulo N. This gives

$$t_{k,m}{}^N = 1 \qquad k,m = 0,1,\cdots,N-1. \quad (8)$$

Therefore, the $t_{k,m}$'s are Nth roots of unity. For T to be nonsingular, all the $t_{k,1}$'s should be distinct and, since there are only N distinct Nth roots of unity, the $t_{k,1}$'s must be these N distinct roots. Without loss of generality, $t_{1,1}$ can be taken as a root of order N, i.e., N is the least positive integer such that $t_{1,1}{}^N = 1$. Therefore, all $t_{k,1}$'s can be written as some powers of $t_{1,1}$. Again without loss of generality, the rows of T can be arranged so that

$$t_{k,1} = t_{1,1}{}^k. \quad (9)$$

Combining (7) and (9) and denoting $t_{1,1}$ by α gives the following

$$t_{k,m} = \alpha^{km} \qquad k,m = 0,1,\cdots,N-1. \quad (10)$$

The structure thus established causes the transform to be orthogonal. The elements $\tilde{t}_{k,m}$ of T^{-1} are given by

$$\tilde{t}_{k,m} = N^{-1}\alpha^{-km} \qquad k,m = 0,1,\cdots,N-1. \quad (11)$$

To prove this, consider

$$TT^{-1} = I$$

or

$$N^{-1} \sum_{n=0}^{N-1} \alpha^{kn}\alpha^{-ln} = \delta_{k,l} \tag{12}$$

where $\delta_{k,l}$ is the Kronecker delta function. Taking $k - l = p$, (12) can be rewritten as

$$N^{-1} \sum_{n=0}^{N-1} \alpha^{pn} = \begin{cases} 1 & \text{if } p = 0 (\text{mod } N) \\ 0 & \text{otherwise.} \end{cases} \tag{13}$$

To prove the orthogonality, we have to prove (13). If $p = 0(\text{mod } N)$, then $\alpha^p = 1$ and the first part is thus established. If $p \neq 0(\text{mod } N)$, $\alpha^p \neq 1$ or $(\alpha^p - 1) \neq 0$. Multiplying (13) by $(\alpha^p - 1)$, we obtain

$$N^{-1}(\alpha^p - 1) \sum_{n=0}^{N-1} \alpha^{pn} = N^{-1}(\alpha^{pN} - 1) = 0.$$

Thus (13) is established.

An examination of the preceding development reveals that the existence of an $N \times N$ transform having the cyclic convolution property depends only on the existence of an α that is a root of unity of order N, and the existence of N^{-1}. The structures of the transform matrix T and its inverse are given by (10) and (11) respectively, and these are the only structures which support the cyclic convolution property. This structure is called the DFT structure. Because of the DFT structure, any transform having the cyclic convolution property also has a fast computational algorithm similar to the FFT if N is highly composite [8]. In the complex number field the DFT, with $\alpha = e^{-j2\pi/N}$, is the only transform having the cyclic convolution property. But, in a finite field or, more generally, a ring we might also have transforms with the cyclic convolution property if there exists a root of unity of order N and if N^{-1} exists. In the next section we establish results for the special case of a finite ring of integers modulo of an integer, which seem to be most appropriate for practical digital computation. Appendix A lists some of the properties of these transforms.

Thus far we have discussed only the transforms having the cyclic convolution property. Noncyclic convolution can be implemented using cyclic convolution by padding the sequences with zeros [1]. If only noncyclic convolution is desired, a more general class of transforms is possible. Consider two sequences of lengths L and M, respectively, that are noncyclically convolved giving an output sequence of length $N = L + M - 1$. First we pad the sequences with zeros to make them of length N each. For the $N \times N$ transform T, the desired restrictions on $t_{k,m}$'s are still given by (6), but now the indices are not added modulo N; this modifies (6) to be

$$\begin{aligned} t_{k,m+l} = t_{k,m}t_{k,l} \qquad & k = 0,1,\cdots,N-1 \\ & m = 0,1,\cdots,M-1 \\ & l = 0,1,\cdots,L-1. \end{aligned} \tag{14}$$

Repeated applications of (14) gives

$$t_{k,m} = k_{k,1}{}^m \qquad k,m = 0,1,\cdots,N-1. \tag{15}$$

This is the same as (7), but now we do not have (8) which implies cyclic convolution. Because of the absence of (8), $t_{k,1}$'s are not restricted to Nth roots of unity. The only restriction is the nonsignularity constraint which requires all $t_{k,1}$'s to be distinct. A similar procedure is discussed by Knuth [3] in connection with the multiplication of large integers which is almost analogous to noncyclic convolution. When the $t_{k,1}$'s are not restricted to Nth roots of unity, in general there is no FFT-type algorithm to compute the transform or the inverse transform. As a matter of fact, the inverse transform does not have a simple structure. Because of these limitations we restrict ourselves to transforms having the cyclic convolution property.

III. TRANSFORMS IN THE RING OF INTEGERS MODULO AN INTEGER

In any practical situation, or when working with digital machines, the data are available only with some finite precision, and therefore, without loss of generality, the data can be considered to be integers with some upper bound. To compute convolution in this digital domain, operations in the complex number field of the continuous domain can be imitated with a finite field or, more generally, a finite ring of integers under additions and multiplications modulo some integer F; and an integer α of order N replaces $e^{-j2\pi/N}$ used in a DFT. In this ring, when two integer sequences $x(n)$ and $h(n)$ are convolved, the output integer sequence $y(n)$ is congruent to the convolution of $x(n)$ and $h(n)$ modulo F. In the ring of integers modulo F, conventional integers can be unambiguously represented if their absolute value is less than $F/2$. If the input integer sequences $x(n)$ and $h(n)$ are so scaled that $| y(n) |$ never exceeds $F/2$, we would get the same results by implementing convolution in the ring of integers modulo F as that obtained with normal arithmetic. This is similar to the overflow constraint in fixed-point digital machines.

In most digital filtering applications, $h(n)$ represents the impulse response and is known a priori; also the maximum magnitude of the input signal is usually known. In this situation, we can bound the peak output magnitude by,

$$| y(n) | \leq | x(n) |_{\max} \sum_{n=0}^{N-1} | h(n) |.$$

When we relate continuous-time signals to discrete-time signals through sampling, we obtain an aliasing effect in the frequency domain; all the frequencies are obtained modulo the sampling frequency. A similar effect occurs when the amplitude is discretized and convolution is performed in the integer domain modulo an integer F. We obtain aliasing in amplitude which can be avoided as noted before.

In order to have a computational advantage in implementing cyclic convolution, we will derive a transform having the cyclic convolution property in this ring. In addition, we would like this transform to have a fast computational algorithm. First, we will present results about the existence of such transforms which were shown to depend on the existence of an α of order N and existence N^{-1} in the ring. The number theory necessary to understand the derivations is very basic and can be found in any elementary book on number theory [14].

Let

$$F = p_1^{r_1} p_2^{r_2} \cdots p_l^{r_l} \tag{16}$$

be the prime factorization of F. Since α is of order N, we have

$$\alpha^N = 1 (\text{mod } F) \tag{17}$$

which implies

$$\alpha^N = 1 (\text{mod } p_i^{r_i}) \qquad i = 1,2,\cdots,l. \tag{18}$$

If the transform matrix T is to be nonsingular, no two rows of T can be the same, which implies

$$\alpha^p \not\equiv \alpha^p (\text{mod } p_i^{r_i}) \qquad i = 1,\cdots,l, p \neq q$$
$$0 \leq p,q \leq N-1. \tag{19}$$

Equations (18) and (19) imply that α should be of order N with respect to each prime power factor $p_i^{r_i}$ of F, i.e., N is the least positive integer such that

$$\alpha^N = 1 (\text{mod } p_i^{r_i}) \qquad i = 1,2,\cdots,l. \tag{20}$$

If α is of order less than N with respect to any moduli $p_i^{r_i}$, then T would be singular with respect to that modulus. Also, for N^{-1} to exist, N should be relatively prime to F, i.e., p_i should not be a factor of N. These two considerations give Theorem 1, which is proven in Appendix B.

Theorem 1

An $N \times N$ transform T having the cyclic convolution property in the ring of integers modulo an integer F exists if and only if N divides $0(F) \triangleq gcd(p_1 - 1, p_2 - 1, \cdots, p_l - 1)$.

IV. FERMAT NUMBER TRANSFORMS

For number theoretic transforms to be more attractive in comparison to the FFT for implementing convolution, they should be computationally efficient. There are three requirements that will be considered. First, N should be highly composite (preferably a power of 2) for a fast FFT type algorithm to exist. Second, since complex multiplications take most of the computational effort in calculating the FFT, it is important that the multiplication by powers of α be a simple operation. This is possible if the powers of α have very few bit binary representations; preferably also a power of two, where multiplication by a power of α reduces to a word shift. Third, in order to facilitate arithmetic modulo F, F should also have a very few bit binary representation.

Now we shall make a systematic investigation of good choices for F, for which the maximum transform length N is not too small. As noted above, we would like F to have a very few bit binary representation. The first possibility is 2^b; it has a prime factor 2 and therefore according to Theorem 1 of Section III, the maximum possible transform length is 1. For $2^b - 1$, let k be a composite PQ, where P is prime. Then $2^P - 1$ divides $2^{PQ} - 1$ and the maximum possible length of the transform will be governed by the length possible for $2^P - 1$. Therefore, only the primes P need to be considered interesting. Numbers of this form are known as Mersenne numbers and Rader [10] has discussed convolution using Mersenne numbers in detail. For MNT's [10], it can be shown that transforms of length at least $2P$ exist and the corresponding $\alpha = -2$. Mersenne number transforms are not of as much interest because $2P$ is not highly composite and, therefore, we do not have fast FFT-type computational algorithms to compute the transforms. For $2^k + 1$, say k is odd. Then 3 divides $2^k + 1$ and the largest possible transform length is 2. Thus k is even. Let k be $s2^t$, where s is an odd integer. Then $2^{2^t} + 1$ divides $2^{s2^t} + 1$ and the length of the possible transform will be governed by the length possible for $2^{2^t} + 1$. Therefore, integers of the form $2^{2^t} + 1$ are of interest. These numbers are known as Fermat numbers and will be discussed in detail in this paper. Fermat numbers seem to be optimum in the sense of having transforms whose length is interesting while the word size is moderate. Numbers of the form $2^{s2^t} + 1$ are also of limited interest and are discussed in Section IX. A systematic investigation of those F which require more than two bit representation is difficult. Our preliminary investigation in that direction does not seem to be very encouraging.

In the light of the above discussion, it is proposed to use $F_t \triangleq 2^b + 1, b = 2^t$, t being a positive integer, as good choices for F. F_t is known as the tth Fermat number. Arithmetic modulo F_t can be performed using b-bit arithmetic, as will be discussed in Section VI. The number of bits used for the signal and the filter coefficients decide the value of b to be used so as not to cause error in the output due to overflow. The value of b required is slightly more than double the number of bits used for the signal representation. One simple bound on the output was presented in the last section. Computationally, Fermat numbers seem to be the best choices for F, therefore, as far as is practicable, they should be used. If the value of b obtained from the overflow consideration is not a power os 2, it should be increased to the next power of 2 in order that Fermat numbers can be used. Some other possibilities are discussed in Section IX. Since the arithmetic is done modulo a Fermat number, we call the associated transforms FNT's. Below we define FNT's and their inverse transforms for sequences of length $N = 2^m$:

$$X(k) = \sum_{n=0}^{N-1} x(n) \alpha^{nk} (\text{mod } F_t) \quad k = 0,1,\cdots,N-1 \tag{21}$$

$$x(n) = (2^{-m}) \sum_{k=0}^{N-1} X(k) \alpha^{-nk} (\text{mod } F_t)$$

$$F_t = 2^b + 1, \qquad b = 2^t \tag{22}$$

α is an integer of order N, i.e., N is the least positive integer such that

$$\alpha^N = 1 (\text{mod } F_t).$$

Note that $\alpha^{-1} = \alpha^{N-1}$ and $2^{-m} = -2^{b-m} (\text{mod } F_t)$.

As discussed in Sections II and III, we can perform cyclic convolution of two integer sequences of length N, using the FNT, if certain conditions are satisfied. Now we discuss the possible values of N for which an FNT exists, given a choice of F_t. As implied by Theorem 1 of Section III, for transforms (mod F_t) of length N to exist, N should divide $0(F_t)$.

Fermat numbers up to F_4 are all primes, therefore for these cases $0(F_t) = 2^b$, and we can have an FNT for any length $N = 2^m$, $m \leq b$. For these Fermat primes the integer 3 is an α of order $N = 2^b$, giving the largest possible transform length. Of course there are 2^{b-1} other integers also which are of order 2^b. They can be obtained by taking odd powers of 3. If an integer α is of order N, then $\alpha^p (\text{mod } F_t)$ will be of order N/p, if N/p is an integer. Therefore, using Fermat primes, an integer α of order 2^m $m \leq b$, is given by $3^{2^{b-m}} (\text{mod } F_t)$. The integer 2 is of order $2b = 2^{t+1}$. If α is taken as 2 or a power of 2, all the powers of α would be some powers of 2, and, for these cases, as discussed in the beginning of this section, the FNT can be computed very efficiently and is called the RT.

There are no other known Fermat primes. For digital filtering applications, $F_5(b = 32)$ and $F_6(b = 64)$ seem to be most practical. Lucas [13] has proven that every prime factor of composite F_t is of the form $K \cdot 2^{t+2} + 1$. Therefore, 2^{t+2} divides $0(F_t)$, for $t > 4$. In particular it can be verified that for F_5 and F_6, $0(F_t) = 2^{t+2}$. Therefore, for these choices of Fermat numbers, the maximum possible transform length is $2^{t+2} = 4b$. Also, we assert that α_p given by (23) is of order 2^{t+2} mod F_t, $t \geq 2$.

$$\sqrt{2} \triangleq \alpha_p = 2^{2^{t-2}}(2^{2^{t-1}} - 1). \tag{23}$$

We denote this α_p as $\sqrt{2}$ because

$$\alpha_p^2 = 2 \bmod F_t.$$

The proof that α_p given by (23) is of order 2^{t+2} with respect to any factor of F_t is given in Appendix C. Any odd power of $\sqrt{2}$ will also be of order 2^{t+2}. By raising $\sqrt{2}$ to 2^{t+2-m}th power, we obtain an integer α of order 2^m, $m \leq t + 2$.

Table I gives values of N for the two most important values of α and also gives the maximum possible N for the most practical values of b. The Fermat number transform has also been defined by Rader [10], using $\alpha = 2$ but his development does not indicate the possibility of using $\sqrt{2}$. Using the results of Section III, we have found the maximum possible length N for which an FNT exists, for the most practical values of b, and have found the corresponding integer α. Using $\alpha = \sqrt{2}$, the maximum possible lengths for these transforms have increased by a factor of 2, which makes them more useful. In the next section we discuss a two-dimensional convolution scheme whereby very long one-dimensional sequences can also be convolved using a two-dimensional FNT.

TABLE I
PARAMETERS FOR SEVERAL POSSIBLE IMPLICATIONS FOR THE FNT'S

t	b	F_t	N[a] $\alpha = 2$	N $\alpha = \sqrt{2}$	N_{max}	α for N_{max}
3	8	$2^8 + 1$	16	32	256	3
4	16	$2^{16} + 1$	32	64	65536	3
5	32	$2^{32} + 1$	64	128	128	$\sqrt{2}$
6	64	$2^{64} + 1$	128	256	256	$\sqrt{2}$

[a] This case corresponds to the RT.

The basis functions for the FNT are integer exponentials mod F_t, in comparison to the complex exponentials for the DFT. These integer exponentials fold over after they cross $F_t/2$. Because of the nature of modulo arithmetic, the FNT coefficients do not seem to have any physical meaning. Although the signal for which the FNT is being taken may be very small, its FNT coefficients may lie anywhere between 0 and $F_t - 1$. As a matter of fact, during various stages of the computation each accumulation of the signal "overflows" many times. But still the end result of the convolution would be exact if the input signals are properly bounded. FNT's follow all the properties mentioned in Appendix A.

Example

To make the ideas of this section more clear, we now present an example. This example will illustrate several points, treatment of negative values in the data, the structure of the transform and the inverse transform matrix, negative powers of α, frequent "overflow" during computation, meaningless of the transform values, and exactness of the final answer. This example will not demonstrate the efficient implementation of the RT using the binary arithmatic. That will be illustrated in Section VI on implementation of the RT.

Consider two sequences $x = (2, -2, 1, 0)$ and $h = (1, 2, 0, 0)$, whose convolution is desired. From the overflow consideration, it is sufficient if we work modulo $F_2 = 17$. We want $N = 4$, for F_2 the integer 2 is of order 8, therefore $2^2 = 4$ is an α of order 4. The transformation matrix T is given by

$$T = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 4 & 4^2 & 4^3 \\ 1 & 4^2 & 4^4 & 4^6 \\ 1 & 4^3 & 4^6 & 4^9 \end{bmatrix} \equiv \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 4 & -1 & -4 \\ 1 & -1 & 1 & -1 \\ 1 & -4 & -1 & 4 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 4 & 16 & 13 \\ 1 & 16 & 1 & 16 \\ 1 & 13 & 16 & 4 \end{bmatrix} (\text{mod } 17).$$

Since $4^{-1} = -4 \pmod{17}$, the inverse transformation matrix T^{-1} is given by

$$T^{-1} = 4^{-1}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 4^{-1} & 4^{-2} & 4^{-3} \\ 1 & 4^{-2} & 4^{-4} & 4^{-6} \\ 1 & 4^{-3} & 4^{-6} & 4^{-9} \end{bmatrix} = -4\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -4 & -1 & 4 \\ 1 & -1 & 1 & -1 \\ 1 & 4 & -1 & -4 \end{bmatrix}$$

$$= 13\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 13 & 16 & 4 \\ 1 & 16 & 1 & 16 \\ 1 & 4 & 16 & 13 \end{bmatrix} \pmod{17}.$$

The transforms of x and h are given by

$$X = Tx = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 4 & 16 & 13 \\ 1 & 16 & 1 & 16 \\ 1 & 13 & 16 & 4 \end{bmatrix}\begin{bmatrix} 2 \\ 15 \\ 1 \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} 18 \\ 78 \\ 243 \\ 213 \end{bmatrix} = \begin{bmatrix} 1 \\ 10 \\ 5 \\ 9 \end{bmatrix} \pmod{17}.$$

Note that in x, -2 was represented by $-2 + 17 = 15$. Similarly,

$$H = (3,9,16,10) \quad \text{and} \quad Y = X \cdot H = (3,90,80,90)$$
$$= (3,5,12,5) \pmod{17}.$$

Taking the inverse transform of Y,

$$y = (2,2,14,2) \pmod{17}.$$

According to our assumption, integers are supposed to lie between -8 and 8. Therefore, 14 must be represented as $14 - 17 = -3$. This gives $y = (2,2,-3,2)$, which is the correct answer. Also, note that y is a symmetric sequence, therefore Y is also a symmetric sequence. Other than this, the transform values have no significance.

V. TWO-DIMENSIONAL CONVOLUTION FOR CONVOLVING LONG SEQUENCES

Arithmetic modulo F_t can be implemented using a $b = 2^t$ bit representation of integers, as will be discussed later in Section VI, with some provision for representing 2^b. The maximum length of sequences which can be cyclically convolved using the FNT is $4b$, using $\alpha = \sqrt{2}$, as discussed above. Therefore, the length of sequences which can be convolved is proportional to the word length in bits. Thus, for long sequences, word length requirement may be excessive. Rader [10] suggested using a two-dimensional convolution scheme to convolve long one-dimensional sequences. Agarwal and Burrus [11], [12] presented such a two-dimensional convolution scheme. Using this scheme, cyclic convolution of length $N = LM$ is implemented as a two-dimensional cyclic convolution of length $(2L - 1)$ by M (or $2L$ by M). This two-dimensional cyclic convolution can be implemented using a two-dimensional FNT [11], [12] defined similar to the one-dimensional transform. Using this two-dimensional scheme, the word length required is proportional to the square root of the length of the sequences to be convolved which would give for a maximum sequence length, $8b^2$, rather than $4b$. If M is taken as the maximum possible length $4b$, and $2L - 1$ is a small integer, then either direct convolution or another high-speed algorithm could be employed [12], [15] to compute convolution along the short dimension and the one-dimensional FNT could be used along the long dimension. Computationally this combination can be very efficient as will be shown in the implementation in Section VIII. Table II lists the maximum lengths for two-dimensional transforms.

TABLE II
MAXIMUM ONE-DIMENSIONAL CYCLIC CONVOLUTION LENGTHS USING TWO-DIMENSIONAL FNT OR RT

Word length b	N for $\alpha = 2$	N for $\alpha = \sqrt{2}$
16	512	2048
32	2048	8192
64	8192	32768

VI. A SUGGESTED HARDWARE IMPLEMENTATION

Since the structure of the FNT is similar to that of the DFT, and N is a power of 2, a fast implementation of the FNT's similar to the FFT with radix 2 exists. As a matter of fact, all we have to do is replace $W = e^{-j2\pi/N}$ by α in any FFT algorithm [1].

In computing the FNT, arithmetic is done modulo $2^b + 1$. In this arithmetic the only allowed integers are $0,1,\cdots,2^b$ and all integers whose absolute values do not exceed 2^{b-1} can be represented unambiguously. Negative integers are represented by adding $2^b + 1$ to them; this is similar to twos complement and ones complement representation of negative integers. Using a b-bit register, all integers from 0 to $2^b - 1$ can be represented. The problem remains to represent 2^b. If the data are uncorrelated, the probability that this number will appear after an arithmetical operation is approximately 2^{-b}. For digital filtering applications, b would typically be 32 or 64; in these cases the probability of occurrence of 2^b is extremely small. If occasional error in convolution is permissible, for these cases, probably there is no need of any extra

hardware to represent 2^b. If the need exists, an extra bit could be used to represent 2^b at the expense of a more complicated hardware. The following discussion is based on the b-bit representation of the integers.

Now, we discuss how various basic arithmetical operations can be performed modulo F_t.

A. Negation

Let

$$A = \sum_{i=0}^{b-1} a_i 2^i, a_i = 0 \text{ or } 1.$$

Then

$$\begin{aligned} -A &= -\sum_{i=0}^{b-1} a_i 2^i \\ &= \sum_{i=0}^{b-1} \bar{a}_i 2^i - (2^b - 1) \\ &= \sum_{i=0}^{b-1} \bar{a}_i 2^i - (2^b - 1) + 2^b + 1 \qquad \bmod F_t \\ &= \sum_{i=0}^{b-1} \bar{a}_i 2^i + 2 \qquad \bmod F_t. \end{aligned}$$

Thus to negate a number, we have to complement each bit and add 2 to the result. For example,

$$(\text{mod } 17): 4 = 0100; -4 = \left\{ \begin{array}{r} 1011 \\ +\quad 10 \\ \hline 1101 \end{array} \right\} = 1101 = 13;$$

$$13 + 4 = 17.$$

B. Addition

When we add two b-bit integers, we obtain a b-bit integer and possibly a carry bit. The carry bit represents $2^b = -1 \bmod F_t$. To implement arithmetic modulo $2^b + 1$, we simply subtract the carry bit. Thus the hardware should be of the carry subtract type. For example,

$$\begin{array}{l} (\text{mod } 17): 10 + 9 = 17 = 2 (\text{mod } 17) \\ \end{array} \begin{array}{r} 1010 \\ +1001 \\ \hline 10011 \\ \hookrightarrow\ 1 \\ \hline 0010 = 2 \end{array}$$

C. Subtraction

Subtraction is implemented as an addition by first negating the subtrahend and then adding them. Addition must be done according to B.

D. General Multiplication

When we multiply two b-bit integers, we get a $2b$-bit product. Let C_L be the b-bit low-order part of it and C_H be the b-bit high-order part of it, then

$$A \times B = C_L + C_H 2^b = C_L - C_H (\text{mod } F_t).$$

Thus, all we have to do is subtract the higher order register from the lower order register. The subtraction needs to be done according to C. For example,

$$(\text{mod } 17): 13 \times 9 = 117 = 15 (\text{mod } 17); 117 = \frac{0111}{C_H}\ \frac{0101}{C_L}$$

$$\begin{array}{rl} C_L = & 0101 \\ (-C_H) = & 1010 \\ \hline & 1111 = 15. \end{array}$$

E. Multiplication by a Power of 2

If α is taken as 2 or a power of 2, RT's are obtained and the only multiplications involved in computing them are those by some powers of 2. These multiplications are particularly simple to implement in arithmetic modulo F_t. Suppose we need to multiply the contents of a register by 2^k, $0 < k < b$, all we need to do is left-shift the contents of the register by k bits and subtract the k overflow bits. A convenient way to do this is to append the data register on the left with a register of zeros, left-shift the double register by k positions, dropping the leading zeros, and then subtract the higher order register from the lower order register, as in D. If k is outside the range $0 < k < b$, we make use of the fact that $2^b = -1 \bmod F_t$. Computation of the inverse transform required multiplications by negative powers of 2 which can be converted to positive powers by the following relationship, $2^{-k} = -2^{b-k} \bmod F_t$. An alternate method to multiply by 2^{-k} is to load the data in the higher order register of a double register, filling the lower order register with zeros, then right-shift the double register by k positions and subtract the low-order register from the high-order register mod F_t. For fast implementation of the bit shift operation, shift amount k should be expressed in a binary form and at a clock pulse shifting should be done by a power of 2. For example,

$$(\text{mod } 17): 11 \times 2^3 = 88 = 3 \bmod 17$$

$$11 = 0000\ 1011.$$

Shift left 3 positions $\dfrac{0101}{C_H}\ \dfrac{1000}{C_L}$

$$\begin{array}{rl} C_L = & 1000 \\ (-C_H) = & +1100 \\ \hline & 1\ 0100 \\ & \hookrightarrow 1 \\ \hline & 0011 = 3 \end{array}$$

$$11 \times 2^{-3} = 11 \times (-2^{4-3}) = -22 = 12 \bmod 17$$

$$11 = 1011\ 0000.$$

$$\text{Shift right 3 positions } \frac{0001}{C_H}\ \frac{0110}{C_L}$$

$$\begin{aligned} C_H &= \quad 0001 \\ (-C_L) &= +1011 \\ &\hline \\ &\quad 1100 = 12. \end{aligned}$$

For implementation of the fast RT, unlike the FFT, we do not need to store the powers of α. For serial arithmetic, we could have a register which stores the shift amount k, and as we go along the fast RT flow chart, we continually update the shift amount. All this is particularly simple to do in binary arithmetic. If α is taken as $\sqrt{2}$, only the odd powers of α require a two-bit representation. In the fast FNT algorithm, only one stage requires multiplications by odd powers of α. Multiplication by an odd power of $\sqrt{2}$ would probably be best done by first multiplying by the just lower even power of $\sqrt{2}$, which is a power of 2, then multiplying by $\sqrt{2}$, which is a two-bit number. The resulting increase in computation effort is very small.

VII. COMPARISON WITH THE FFT

As noted in the previous section, computing the RT is a very simple operation on a binary machine. Now let us compare the complexity of various basic operations involved in computing the RT vis-a-vis the FFT. If the two sequences $x(n)$ and $h(n)$ have b_1 and b_2 bit representations, respectively, and are of length N, then the output $y(n)$ would need no more than a $(b_1 + b_2 + \log_2 N)$ bit representation. To obtain the correct result $b \geq b_1 + b_2 + \log_2 N$. In Section III we have given a better bound on the output. Roughly speaking, we need twice the number of bits to carry out the convolution using the RT as compared to the fixed-point FFT implementation of the convolution. But in the DFT, every data point is treated as a complex number and therefore requires two words, one for the real part and one for the imaginary part. Thus, in effect, the hardware requirement for two transforms is about the same. Although for real data it is possible to make use of the symmetry properties of the DFT's they require extra computation and for the purpose of comparison it will be ignored, even though we have taken this into account for our IBM 370/155 implementation to be discussed in the next section. Therefore we shall assume, in the FFT implementation, each data point is represented by a $b/2$ bit real part and a $b/2$ bit imaginary part.

One $b/2$ bit complex addition is equivalent to two $b/2$ bit real additions, which are comparable to a b-bit addition modulo F_t. Thus, the complexity of addition/subtraction is the same in both the transforms. Similarly, it can be shown that a $b/2$ bit complex multiplication is comparable to a b-bit multiplication modulo F_t. Computation of the RT requires multiplications by powers of 2, which implemented as bit shifts and subtractions become much simpler operations compared to complex multiplications required in the FFT implementation.

To compute a length N fast RT, $N \log_2 N$ additions/subtractions, and $(N/2) \log_2 N/2$ multiplications by some powers of 2 are required which are implemented as bit shifts and subtractions. To compute the convolution using the FFT, most of the time is taken in computing the complex multiplications required to compute the transforms. A comparison with the RT reveals that these complex multiplications are replaced by bit shifts and subtractions which are much faster operations. This results in considerable computational savings in the implementation of convolution. This fact has been verified on a general purpose machine (IBM 370/155). The computation required to multiply the two transforms is about the same for both the implementations. To convolve long sequences using the two-dimensional FNT or RT, the computational effort increases by, at the most, a factor of 2. Still, the RT implementation of convolution is much faster as compared to the FFT implementation.

RT's have some additional advantages over the FFT. First, the FFT implementation requires storing all the powers of W requiring a significant amount of storage which may be an important factor for a small minicomputer or a special purpose hardware implementation. Second, fixed-point FFT implementation introduces a significant amount of the roundoff noise at the output, 6–8 bits depending on the data [2]. This degrades the signal-to-noise ratio during the filtering operations. The FNT or RT implementation is error free, the only source of error is input A/D quantization.

VIII. IMPLEMENTATION ON THE IBM 370/155

The word length of the IBM 360–370 series is 32 bits and, therefore, is well suited for the implementation of convolution using the FNT or RT modulo $F_5 = 2^{32} + 1$. It has two's complement representation of negative integers, i.e., $-x$ is represented as $2^{32} - x$. We want negative integers to be represented as the complement modulo $2^{32} + 1$, i.e., $-x$ is to be represented by $2^{32} + 1 - x$, and to accomplish this 1 is added whenever a negative integer is encountered in the data. As noted before, on this machine, we cannot represent -1. If a -1 is encountered in the data, it is rounded either to 0 or to -2. This is equivalent to introducing some initial quantization noise. If the data are uncorrelated, the probability that -1 will appear after an arithmetical operation during computation of the transform is roughly $2^{-32} \simeq 10^{-10}$ per operation. This error introduced during computation is rather serious and probably the corresponding output block will be meaningless. Assuming $N = 64$, the probability that a particular block is erroneous is roughly $10^{-6} \sim 10^{-7}$. For most filtering operations, this may be permissible.

Logical add (ALR) and subtract instructions (SLR) are used to add and subtract two integers modulo F_5. If a carry is detected after add, 1 is subtracted from the result, and if no carry is detected after subtract, 1 is added to the result. Multiplication by 2^k is done using the logical left shift operation (SLDL) and multiplication by 2^{-k} is

TABLE III
CYCLIC CONVOLUTION TIMINGS FOR LENGTH N REAL SEQUENCES

N	FFT ms	FNT or RT ms
32	16	3.3
64	31	7.4
128	60	16.6[a]
256	123	40.0[b]
256	123	80.0[c]
512	245	166.0[c]
1024	530	340.0[c]
2048	1260	720.0[c]

[a] Using $\alpha = \sqrt{2}$.
[b] Using 2 by 128 convolution.
[c] Using two-dimensional RT.

done using logical right shift operation (SRDL), $0 < k < 32$, as explained in Section VI. Similarly, multiplication of two integers modulo F_5 can also be done.

Two assembler subprograms were written to compute the fast RT and the inverse for any sequence length from 2^1 to 2^6, taking α as a power of 2. Two more subprograms were written to compute the fast FNT and its inverse for length 128 sequences, using $\alpha = \sqrt{2}$, given by (23). Out of the 7 stages of the fast algorithm, only one stage requires multiplication by odd powers of $\sqrt{2}$, which are implemented as suggested in Section VI.

To cyclically convolve sequences longer than 128, two-dimensional convolution schemes were used [12]. One such scheme was 2 by 128 convolution for length 256 sequences discussed in Section X of [12]. Another program was written to convolve long one-dimensional sequences using two-dimensional RT's, as discussed in Section III of [12]. Timings for various implementations of convolution and their comparison with the FFT implementations are shown in Table III. To compute these timings it is assumed that transforms of the h sequences have been precalculated. Thus, timings are for computing the x transforms, multiplications of the transforms, and their inverse transforms. For the FFT implementation, a very efficient algorithm [17] was used, which employed both radix 4 and radix 2 steps and took into account the fact that the data are real. For sequences up to length 256, improvements of the order of 3 to 5 were obtained over the FFT implementations. Since the hardware of IBM 360/370 series is not designed to do arithmetic modulo Fermat numbers, extra branch instructions were necessary after add or subtract operations to take into account the carry bit. If the hardware is designed to do arithmetic modulo Fermat numbers, these instructions could be avoided, resulting in a significant reduction in the computation time. Copies of the assembler subprograms are available from the authors.

IX. GENERALIZATIONS AND OTHER APPLICATIONS

In this paper we have discussed convolution in the rings of integers modulo of Fermat numbers. These numbers seem to be the best choice for implementation on digital computers. Nevertheless, any integer F, for which $O(F) > 1$ can be used, was discussed in Section IV. Rader [10] proposed the use of Mersenne numbers M_p ($2^p - 1$, p a prime). MNT's have $\alpha = -2$ and $N = 2p$, and therefore do not have an FFT-type fast computational algorithm.

In many situations the quantization and overflow considerations may require arithmetic to be done using 10 to 20-bit arithmetic. For this situation, working in the integer ring modulo $F_4(2^{16} + 1)$ would be insufficient and at the same time, computing the FNT modulo $F_5(2^{32} + 1)$ would require excessive number of bits. In this situation, we may perform convolution modulo $2^{24} + 1$. Also, many computers have 24-bit word length. Transforms similar to FNT's exist for this situation. We could take $F = 2^{24} + 1$ and then $\alpha = 8$ gives $N = 16$, and $\alpha = 8^{1/2} = 2^7(2^{12} - 1)$ gives $N = 32$ as the maximum length of the one-dimensional transform.

In general one could take $F = 2^{s2^t} + 1$, s is an odd integer. In that case, $\alpha = 2^s$ would give $N = 2^{t+1}$, and $\alpha = (2^s)^{1/2} = 2^{[(s-1)/2+s2^{t-2}]}(2^{s2^{t-1}} - 1)$ would give $N = 2^{t+2}$. In many situations it may be possible to have transforms of greater length than 2^{t+2}. For example, taking $F = 2^{40} + 1$, the maximum possible transform length is 256, and, taking $F = 2^{80} + 1$, the maximum possible transform length is 1024, but the corresponding α's may not be simple.

As discussed earlier, the FNT implementation requires roughly twice the number of bits for the arithmetic. Most minicomputers have short word lengths and in that situation it may not be possible to efficiently use FNT's, but there is a solution to this problem. We can split the data bits into two halves and then convolve them.

$$x(n) = x_2(n) + x_1(n)2^k, \qquad |x_2(n)| < 2^k$$
$$h(n) = h_2(n) + h_1(n)2^k, \qquad |h_2(n)| < 2^k \tag{24}$$

$$y = x*h = x_1*h_1 \cdot 2^{2k} + (x_1*h_2 + x_2*h_1)2^k + x_2*h_2. \tag{25}$$

Now, since x_1, h_1, x_2, and h_2 have roughly half the number of bits, it should be possible to convolve them using approximately half the number of bits. If necessary, a more precise analysis of the above situation could be easily performed. In (25), the last term, in comparison to the first term, is very small and can be neglected. We need to take two transforms for x and two transforms for h, the summation shown within the parentheses can be performed in the transform domain. Finally, we need to take two inverse transforms, one for x_1*h_1 and the other for $(x_1*h_2 + x_2*h_1)$.

FNT's can also be used for convolutions required in block recursive realizations of recursive digital filters [16]. For these realizations, convolutions form the main computational task. Since convolution with the FNT's can be performed with both efficiency and with perfect accuracy, it should be very attractive for this application. For very high-order recursive filters this may reduce roundoff noise problems associated with these filters. Other appli-

cations of number theoretic transforms are discussed by Pollard [4] and Knuth [3], [7]. They include fast multiplication of large integers and multiplication of polynomials.

APPENDIX A

PROPERTIES OF THE TRANSFORMS HAVING THE CYCLIC CONVOLUTION PROPERTY

Below we summarize the definition and some important properties of the transforms having the cyclic convolution property. It is assumed that all arithmetical operations are performed in a ring. Most of these properties are similar to the DFT properties [1].

A. Definition

Transform:

$$F(k) = \sum_{n=0}^{N-1} f(n)\alpha^{nk} \qquad k = 0,1,\cdots,N-1. \quad \alpha^N = 1 \tag{A1}$$

Inverse Transform:

$$f(n) = N^{-1} \sum_{k=0}^{N-1} F(k)\alpha^{-nk} \qquad n = 0,1,\cdots,N-1. \tag{A2}$$

B. Basis Functions

$$\varphi_n(k) = \alpha^{-nk} \qquad n,k = 0,1,\cdots,N-1. \tag{A3}$$

C. Orthogonality

The inner product of two basis functions are given by [see (12) and (13)]

$$\langle \varphi_n, \varphi_m \rangle = \sum_{k=0}^{N-1} \varphi_n(k)\varphi_m^{-1}(k) = \sum_{k=0}^{N-1} \alpha^{(m-n)k} = \begin{cases} N & \text{if } m = n \bmod N \\ 0 & \text{otherwise} \end{cases} \tag{A4}$$

This implies the orthogonality of the basis functions.

D. Periodicity

$F(k)$ can also be periodically extended, similar to the extension of $f(n)$,

$$f(n+N) = f(n)$$
$$F(K+N) = F(K). \tag{A5}$$

E. Symmetry Property

If the signal is symmetric, i.e.,

$$f(n) = f(-n) = f(N-n)$$

the transformed sequence is also symmetric, i.e.,

$$F(K) = F(-K) = F(N-K). \tag{A6}$$

If the signal is antisymmetric, i.e.,

$$f(n) = -f(-n) = -f(N-n),$$

the transformed sequence is also antisymmetric, i.e.,

$$F(K) = -F(-K) = -F(N-K). \tag{A7}$$

F. Symmetry of the Transformed Pair

$$T\{T\{f(n)\}\} = Nf(-n). \tag{A8}$$

G. Shift Theorem

If

$$T\{f(n)\} = F(K)$$
$$T\{f(n+m)\} = F(K)\alpha^{-mK}. \tag{A9}$$

H. Fast Computational Algorithm

If N can be factored as

$$N = r_1 \cdot r_2 \cdots r_m$$

then a fast computational algorithm similar to the FFT can be used to compute the transform which requires on the order of $N(r_1 + r_2 + \cdots + r_m)$ arithmetical operations.

I. Convolution Property

Transform of cyclic convolution of two sequences in the product of their transforms, i.e., if

$$y = x * h$$

then

$$T\{y\} = T\{x\} \cdot T\{h\}. \tag{A10}$$

J. Parseval's Theorem

Let

$$X(k) = T\{x(n)\}$$
$$H(k) = T\{h(n)\}.$$

Then

$$N \sum_{n=0}^{N-1} x(n)h(n) = \sum_{k=0}^{N-1} X(k)H(-k) \tag{A11}$$

and

$$N \sum_{n=0}^{N-1} x(n)h(-n) = \sum_{k=0}^{N-1} X(k)H(k). \tag{A12}$$

For the particular case of $x(n) = h(n)$

$$N \sum_{n=0}^{N-1} x^2(n) = \sum_{k=0}^{N-1} X(k)X(-k) \tag{A13}$$

and

$$N \sum_{n=0}^{N-1} x(n)x(-n) = \sum_{k=0}^{N-1} X(k)^2. \tag{A14}$$

Note that in this case, the conventional Parseval's theorem

$$N \sum_{n=0}^{N-1} |x(n)|^2 = \sum_{k=0}^{N-1} |X(k)|^2$$

does not hold, because in a ring magnitude is undefined.

K. Stretch Property

The stretch property exists if the transform for the longer length sequence exists in that ring.

L. Sampling Property

The sampling property exists.

APPENDIX B

PROOF OF THEOREM I

As shown before, the existence of such a transform depends on the existence of an α of order N with respect to each prime power factor moduli, $p_i^{r_i}$. According to Euler's Theorem [14], N, order of α, must divide $\varphi(p_i^{r_i})$, where $\varphi(p_i^{r_i})$ is the Euler's φ function given by

$$\varphi(p_i^{r_i}) = p_i^{r_i-1}(p_i - 1). \tag{B1}$$

Thus, $N \mid \varphi(p_i^{r_i})$, and since p_i is not a factor of N, $N \mid (p_i - 1)$, $i = 1,2,\cdots,l$ or

$$N \mid gcd\{p_1 - 1, p_2 - 1, \cdots, p_l - 1\}$$

or

$$N \mid 0(F). \tag{B2}$$

This proves the necessary part of the theorem. To prove sufficiency we have to show the existence of an α of order N if $N \mid 0(F)$. Note that (B2) rules out F being an even integer because in that case $0(F) = 1$. If $N \mid 0(F)$, $N \mid (p_i - 1)$ or $N \mid \varphi(p_i)$, therefore, one can find integers β_i of order N modulo $p_i^{r_i}$ [14], $i = 1,2,\cdots,l$. Then, by the Chinese Remainder Theorem, one can find an α of order N modulo $F = p_1^{r_1}\cdots p_l^{r_l}$, such that

$$\alpha = \beta_i (\text{mod}\, p_i^{r_i}), \qquad i = 1,2,\cdots,l. \tag{B3}$$

This is our desired α of order N. Since N and p_i are relatively prime, N^{-1} exists in this ring. This completes the proof of the theorem.

APPENDIX C

To prove that α_p of (23) is of order $2^{t+2} \bmod F_t$, we have to prove that it is of order 2^{t+2} with respect to each prime power factor of F_t (20). Let $p_i^{r_i}$ be a prime power factor of F_t, then

$$\begin{aligned} {\alpha_p}^{2^{t+2}} = 2^{2^{t+1}} &= 1 \bmod F_t \\ &= 1 \bmod p_i^{r_i}. \end{aligned} \tag{C1}$$

Let N_i be the order of α_p with respect to $p_i^{r_i}$, then by Euler's theorem,

$$N_i \mid 2^{t+2} \tag{C2}$$

therefore N_i must be a power of 2. Also,

$$\begin{aligned} {\alpha_p}^{2^{t+1}} = 2^{2^t} &= -1 \bmod F_t \\ &= -1 \bmod p_i^{r_i}. \end{aligned} \tag{C3}$$

From (C1) and (C3), it is obvious that

$$N_i = 2^{t+2}$$

completing the proof.

ACKNOWLEDGMENT

The authors wish to thank C. M. Rader of Lincoln Laboratories, Cambridge, Mass., for his comments and discussions.

REFERENCES

[1] B. Gold and C. M. Rader, *Digital Processing of Signals*. New York: McGraw-Hill, 1969.
[2] A. V. Oppenheim and C. Weinstein, "Effects of finite register length in digital filtering and the fast Fourier transform," *Proc. IEEE*, vol. 60, pp. 957–976, Aug. 1972.
[3] D. E. Knuth, *The Art of Computer Programming*, vol. 2, *Seminumerical Algorithms*. Reading, Mass.: Addison-Wesley, 1969, p. 555 and pp. 259–262.
[4] J. M. Pollard, "The fast Fourier transform in a finite field," *Math. Comput.*, vol. 25, pp. 365–374, Apr. 1971.
[5] I. J. Good, "The relation between two fast Fourier transforms," *IEEE Trans. Comput.*, vol. C-20, pp. 310–317, Mar. 1971.
[6] A. Schonhage and V. Strassen, "Fast multiplication of large numbers," (in German) *Comput.*, vol. 7, pp. 281–292, 1971.
[7] D. E. Knuth, "The art of computing programming—errata et addenda," Comput. Sci. Dep., Stanford University, Stanford, Calif., Rep. STAN-CS-71-194, pp. 21–26, Jan. 1971.
[8] P. J. Nicholson, "Algebraic theory of finite Fourier transforms," *J. Comput. Syst. Sci.*, vol. 5, pp. 524–547, 1971.
[9] C. M. Rader, "The number theoretic DFT and exact discrete convolution," IEEE Arden House Workshop on Digital Signal Processing, Harriman, N. Y., Jan. 11, 1972.
[10] ——, "Discrete convolution via Mersenne transforms," *IEEE Trans. Comput.*, vol. C-21, pp. 1269–1273, Dec. 1972.
[11] R. C. Agarwal and C. S. Burrus, "Fast digital convolution using Fermat transforms," in *Southwest IEEE Conf. Rec.*, Houston, Tex., Apr. 1973, pp. 538–543.
[12] ——, "Fast one-dimensional digital convolution by multidimensional techniques," *IEEE Trans. Acoust., Speech, and Signal Processing*, vol. ASSP-22, pp. 1–10, Feb. 1974.
[13] L. E. Dickson, *History of the Theory of Numbers*, vol. I. Washington, D. C.: Carnegie Institute, 1919, p. 376.
[14] O. Ore, *Number Theory and Its History*. New York: McGraw-Hill, 1948.
[15] D. A. Pitassi, "Fast convolution using Walsh functions," in *Proc. Conf. Applications of Walsh Functions*, Washington, D. C., April 1971, pp. 130–133.
[16] C. S. Burrus, "Block realization of digital filters," *IEEE Trans. Audio Electroacoust.*, vol. AU-20, pp. 230–235, Oct. 1972.
[17] R. C. Singleton, "An algorithm for computing the mixed radix fast Fourier transform," *IEEE Trans. Audio Electroacoust.*, vol. AU-17, pp. 93–103, June 1969.