H. J. Nussbaumer

Linear Filtering Technique for Computing Mersenne and Fermat Number Transforms

Abstract: In this paper, the implementation of pseudo-Mersenne and Fermat Number Transforms is discussed. It is shown that some pseudo-Mersenne Transforms can be computed efficiently by a linear filtering approach. This approach is extended to cover the case of Fermat and pseudo-Fermat Number Transforms by using a special coding scheme for implementing arithmetic operations in a Fermat number system.

Introduction

Mersenne and Fermat Number Transforms have been introduced by Rader [1], Agarwal, and Burrus [2, 3]. These transforms are potentially attractive for digital filtering applications because they have the convolution property and can be computed without multiplications.

In principle, such Number Theoretic Transforms (NTT) could be implemented in the same way as Discrete Fourier Transforms but with multiplications by trigonometric functions replaced by multiplications by powers of two, all operations being performed modulo a Mersenne or Fermat number. When the transforms have a composite number of terms, as is the case with Fermat Number Transforms (FNT) or some pseudo-Mersenne Transforms [4], various pipeline computing techniques can be used [5-7].

In practice, however, direct transposition of Fast Fourier Transform (FFT) architectures does not necessarily lead to optimum implementations and the development of special configurations for computing NTT seems worth exploring. Along these lines, McClellan [8] has proposed a new coding technique for simplifying the implementation of Fermat Number Transforms. We have also shown [9] that additional savings in the number of multiplications required to compute complex convolutions by way of FNT can be achieved by taking advantage of the real number representation of $\sqrt{-1}$ in a Fermat number system.

In this paper, we consider the implementation of pseudo-Mersenne Transforms. We show that when the number of terms is a perfect square, all but one of the variable shift circuits normally required to compute the transform are eliminated. We then extend these results to cover the case of FNT and pseudo-Fermat Number Transforms [10] implemented with modified carry-add circuits and discuss the case of transforms having a number of terms equal to a multiple of a perfect square.

Linear filtering computation of pseudo-Mersenne Transforms

In a preceding paper [4], we have shown that pseudo-Mersenne Transforms could be defined in a ring modulo p_2 by

$$A_k = \left(\sum_{n=0}^{N-1} a_n W^{nk}\right)_{\text{mod } p_2} \qquad k = 0, 1, \cdots, N-1, \qquad (1)$$

with $\{a_n\}$ being the input integer sequence, and

$$p = p_1 \cdot p_2 = 2^q - 1; \qquad (2)$$

$$W = 2^r. \qquad (3)$$

Under certain conditions, this transform has the convolution property and therefore can be used to compute convolutions. It has the advantage over the conventional Mersenne Transform that q does not have to be a prime and therefore that the number of transform terms $N = q/r$ can be composite. As $p = p_1 \cdot p_2$, the pseudo-Mersenne Transform can be computed modulo p in one's complement arithmetic with only a last operation modulo p_2 with

$$A_k = \left(\left(\sum_{n=0}^{N-1} a_n W^{nk}\right)_{\text{mod } p}\right)_{\text{mod } p_2}. \qquad (4)$$

We have shown that many transforms having a highly factorizable number of terms can be defined for which p_1 is small compared to p_2 so that the penalty in word length increase incurred when operating modulo p instead of modulo p_2 is only of the order of 20 percent.

Under these conditions, the use of these transforms is attractive for digital filtering applications because the number of operations required for evaluating the transforms can be reduced by means of FFT-type algorithms and the various operations consist only in additions and multiplications by powers of two. One's complement

additions can be performed very simply with conventional adders by folding the most significant carry output into the least significant carry input. One's complement multiplications by fixed powers of two reduce to fixed circular shifts and therefore correspond merely to permuting connecting wires in a hardware implementation. By contrast, multiplications by variable powers of two require a multistage shift circuit which, although much less expensive than a general multiplier, is still relatively costly.

Under these conditions, rather than using conventional FFT-type pipeline implementations [5–7] which require variable shift circuits, it would be desirable to devise an approach using only fixed shift circuits. This is partly possible by computing the transform as a convolution.

Using the conventional chirp Z-transform algorithm [11] with the substitution

$$nk = \frac{n^2}{2} + \frac{k^2}{2} - \frac{(k-n)^2}{2}, \tag{5}$$

we have

$$A_k = \left(W^{k^2/2} \sum_{n=0}^{n-1} a_n W^{n^2/2} W^{-(k-n)^2/2} \right)_{\text{mod } p_2}, \tag{6}$$

which, except for premultiplication of the input sequence $\{a_n\}$ by $W^{n^2/2}$ and postmultiplication by $W^{k^2/2}$, reduces the transform computation to a convolution of $\{b_n = a_n W^{n^2/2}\}$ with the impulse response $h_l = W^{-l^2/2}$.

If $W^{-\frac{1}{2}}$ is chosen to be a power of two, a practical circuit for calculating the transform reduces to one multiplier by variable powers of two, $W^{n^2/2}$, followed by a linear filter with tap values $W^{-l^2/2}$, which are fixed powers of two and a final multiplier by variable powers of two, $W^{k^2/2}$. It can be seen therefore that all but two of the multipliers by powers of two normally required to compute the transform are eliminated.

When the number of transform terms N is a perfect square, with

$$N = M^2, \tag{7}$$

it is possible to reduce the number of additions required to compute the transform without increasing the number of multiplications by variable powers of two by using the technique proposed by Bluestein for discrete Fourier Transforms [12], [13]. The Z transform $H(Z)$ of the impulse response h_l is

$$H(Z) = \sum_{l=0}^{2N-1} W^{-l^2/2} Z^{-l}, \tag{8}$$

and with

$$\begin{aligned} 1 &= u + Mv \qquad u = 0, 1, \cdots, M-1 \\ &\qquad\quad v = 0, 1, \cdots, 2M-1, \end{aligned} \tag{9}$$

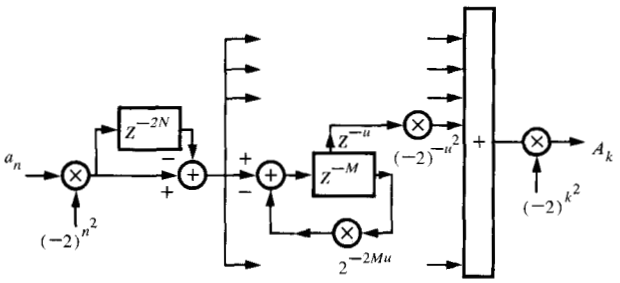

Figure 1 Recursive computation of a pseudo-Mersenne Transform.

we have

$$H(Z) = \sum_{u=0}^{M-1} W^{-u^2/2} Z^{-u} \sum_{v=0}^{2M-1} W^{-Muv} W^{-v^2M^2/2} Z^{-Mv} \tag{10}$$

or

$$H(Z) = (1 - Z^{-2M^2}) \sum_{u=0}^{M-1} \frac{W^{-u^2/2} Z^{-u}}{1 + W^{-Mu} Z^{-M}}, \tag{11}$$

which shows that $H(Z)$ may be realized by a bank of M recursive filters. This approach is particularly interesting in the case of the 25-point and 49-point real transforms and 100-point and 196-point complex transforms computed respectively modulo $(2^{25} - 1)/31$ and $(2^{49} - 1)/127$. For the real transforms, we can take $W^{\frac{1}{2}} = -2$, $W = 4$, and the transform circuit can be realized as shown in Fig. 1. Because the multipliers by fixed powers of two reduce to a simple permutation of connecting wires, the hardware count is limited to $2M$ one's complement adders, M shift registers of M words, one shift register of $2N$ words and two variable shift circuits.

In practice, pseudo-Mersenne Transforms are used for implementing digital filters. For time-invariant filters, the transform of the tap values will be precomputed and stored in a memory and the postmultiplication in the direct transform is redundant with the premultiplication in the inverse transform, so that only one variable shift circuit is required per transform [14].

We note also that with this configuration the duty cycle is only 50 percent, as one block time is required to load the recursive filters and one block time is required for output. Continuous operation would normally require two identical circuits, each processing alternate input blocks, with output available for successive time intervals at alternate circuit outputs.

A much more attractive approach consists in taking advantage of the fact that, during output of the circuit, the recursive filter operation is reduced to recirculation with multiplication by $-W^{-Mu}$. Under these conditions, the transform circuit can be modified as shown in Fig. 2, by adding a bank of circuits for recirculation and multiplication by $-W^{-Mu}$. Transfer from the recursive filters

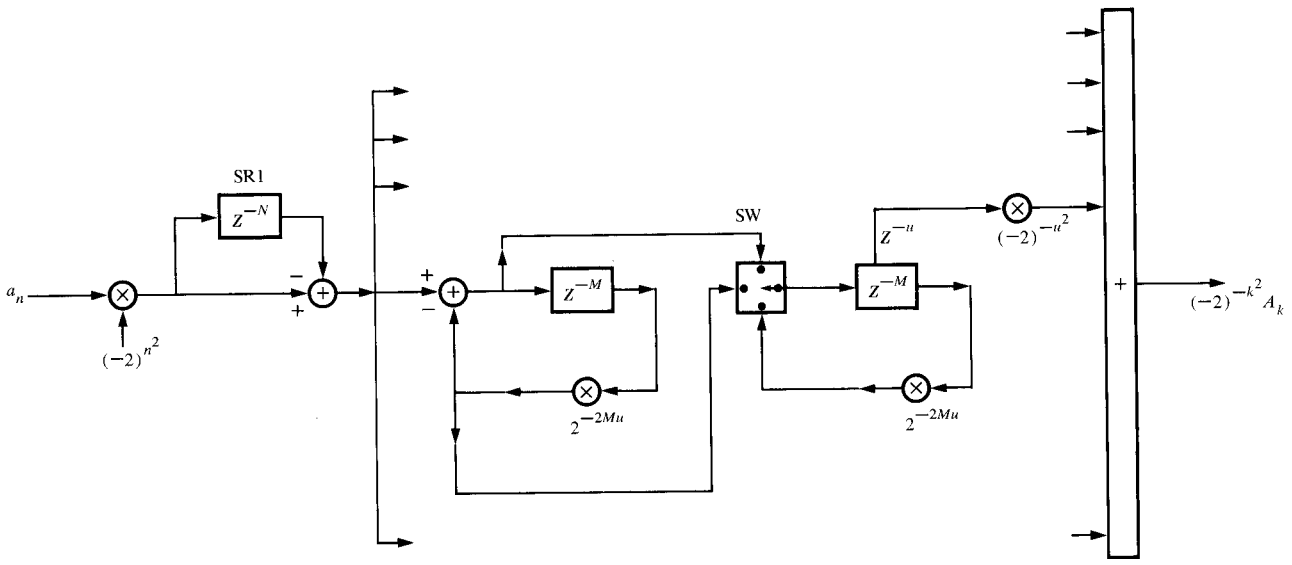

Figure 2 High speed circuit for the recursive computation of pseudo-Mersenne Transforms.

to the recirculation circuits takes place at the end of each block time, without any dead-band interval under control of a switch SW. This circuit operates in a continuous mode, successive input and output blocks being adjacent.

The input shift register, SR1, which serves to reset the recursive filters, has a length reduced to N words instead of $2N$ words in the conventional circuit. This shift register could be eliminated and replaced by auxiliary switches in the recursive filter loops, but in most cases, this last approach will lead to a more complex implementation.

It can be seen that the implementation of an N-term pseudo-Mersenne Transform with the Fig. 2 configuration will require $2M$ adders, one variable shift circuit and $3N$ words of storage. Under the same conditions, a conventional pipeline approach would require $2M - 2$ adders and between M and $2M$ variable shift circuits depending on the particular implementation [15].

Linear filtering computation of Fermat Number Transforms

We have so far seen that for pseudo-Mersenne Transforms having a number of terms which is a perfect square, it is possible to devise an efficient implementation which requires about as many additions as with an FFT-type algorithm but allows a nearly complete elimination of variable shift circuits. The efficiency of the proposed implementation is essentially due to the fact that, in one's complement arithmetic, the cost of multiplying by fixed powers of two is negligible.

Fermat Number Transforms (FNT) are evaluated modulo $2^q + 1$. In such number systems, arithmetic circuits are not simple so that the proposed approach would not seem attractive for FNT. McClellan, however, has shown [8] that if the input signal words are coded in such a way that individual bits correspond to ± 1 instead of 0 or 1, arithmetic operations in a Fermat number system can be implemented in a manner that is similar to one's complement arithmetic. A similar technique has been proposed very recently by Leibowitz [16] and brought to the attention of the author by one of the referees. We have derived independently a different and slightly simpler coding scheme that achieves a similar result and which we describe here. In a Fermat number system modulo $p = 2^q + 1$, the various input words a_n can be represented by

$$a_n = \sum_{i=0}^{q-1} a_{n,i} 2^i + a_{n,q} 2^q \qquad a_{n,i} = 0 \text{ or } 1 \qquad a_{n,i} a_{n,q} = 0. \tag{12}$$

With this representation, a number x_n can be represented unambiguously modulo $2^q + 1$ provided $-2^{q-1} \leq x \leq 2^{q-1}$.
Then

$$a_n = x_n \qquad \text{if } x_n \geq 0;$$

$$a_n = x_n + 2^q + 1 \qquad \text{if } x_n < 0.$$

Instead of computing the FNT modulo p directly on the input sequence $\{a_n\}$, we first convert this sequence into a new sequence $\{b_n\}$ with

$$b_n = 2^q - a_n \tag{13}$$

or with

$$\bar{a}_n = \sum_{i=0}^{q-1} \bar{a}_{n,i} 2^i \qquad \bar{a}_{n,i} = 0 \text{ if } a_{n,i} = 1 \qquad \bar{a}_{n,i} = 1 \text{ if } a_{n,i} = 0, \tag{14}$$

$$b_n = \bar{a}_n + 1 + a_{n,q} 2^q. \tag{15}$$

The coded samples can thus be obtained very simply by taking the complement of the q less significant bits of a_n and adding 1 in a $q+1$-bit binary adder to the word $\bar{a}_n + a_{n,q}2^q$.

The transform or the convolution is computed on the sequence $\{b_n\}$, and the final result is obtained by a decoding operation according to Eqs. (13) or (15). The condition $a_n = 0$ corresponds to $b_n = 2^q$. This condition will be depicted by a flag bit $b_{n,q}$ in the $q+1$-bit words representing b_n. In all other cases, b_n can be represented by

$$b_n = \sum_{i=0}^{q-1} b_{n,i}2^i. \tag{16}$$

To obtain the representation of the negative of a_n, one can do as follows. If $b_{n,q} = 1$, then $a_n = 0$ and one can skip the operation. Otherwise, taking the complement $\bar{b}_n$ of b_n yields $\bar{b}_n = 2^q - 1 - b_n$, and with Eq. (13) and $2^q \equiv -1$,

$$\bar{b}_n = 2^q + a_n, \tag{17}$$

which shows that the negative of a number is obtained as in one's complement arithmetic by inverting the q low-order bits except when $b_{n,q} = 1$.

Addition of two numbers a_n and a_l can also be performed very easily in the transposed system. If b_n and b_l are the coded values of a_n and a_l and if c_n is the sum of b_n and b_l, we have

$$c_n = \sum_{i=0}^{q-1} c_{n,i}2^i + c_{n,q}2^q \qquad c_{n,i} = 0 \text{ or } 1 \tag{18}$$

and

$$c_n = 2^{q+1} - a_n - a_l. \tag{19}$$

The coded value d_n corresponding to $a_n + a_l$ is defined by

$$d_n = 2^q - a_n - a_l. \tag{20}$$

We have, therefore,

$$d_n = c_n - 2^q \equiv \sum_{i=0}^{q-1} c_{n,i}2^i + 1 - c_{n,q}, \tag{21}$$

which shows that, in the transposed system, addition can be performed with ordinary adders in a way similar to one's complement arithmetic but with higher order carry fed back after complementation into the less significant carry input line of the adder. If either one or both of the operands are zero, as depicted by $b_{n,q} = 1$ or $b_{l,q} = 1$, propagation of the complemented fold over carry is inhibited. An inhibition circuit must also be implemented to prevent spurious oscillations caused by the feedback of complemented carries when $d_n = 2^q - 1$.

It can be seen easily, from the rules of addition, that multiplications by 2^r correspond, in the transposed system, to an r-bit rotation around the q-bit word with complementation of the overflow bits. When the flag bit is "one," depicting the condition $a_n = 0$, inversion of the overflow bits is inhibited. Under these conditions, multiplications by fixed powers of two can be implemented very simply, as shown in Fig. 3 in the case of a multiplication-by-2 circuit.

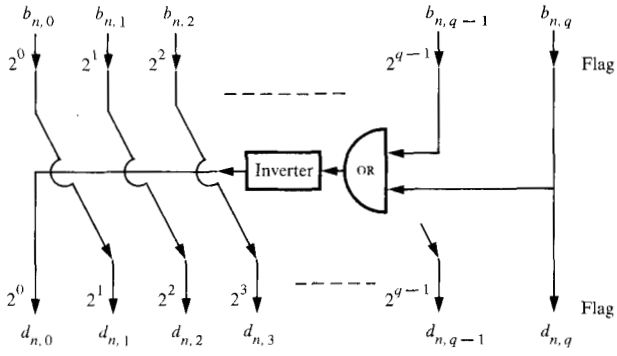

Figure 3 Multiply-by-two circuit in transposed Fermat number system.

The approach discussed in the preceding section can therefore be applied to FNT with almost the same efficiency as for pseudo-Mersenne Transforms. A particularly interesting case corresponds to the 64-point transform evaluated modulo $2^{32} + 1$. When the number of terms is a multiple of four, as is the case with FNT, such that $N = M^2$ and with some complex pseudo-Mersenne Transforms, additional savings can be achieved in the transform implementation. This relates to the fact that, when a digital filter is evaluated via NTT's, the input sequence must be divided into blocks and each block must be padded with a suitable number of zeros in order to prevent folding and aliasing [17]. In the case of Mersenne and Fermat Number Transforms where the maximum transform length is limited, the blocks are usually such that about half the samples are zero.

In the following, we consider the case corresponding to blocks half filled with zeros, with $a_n \neq 0$ for $0 \leq n < N/2$ and $a_n = 0$ for $N/2 \leq n < N-1$. If the transform is evaluated by means of a conventional pipeline technique, the fact that half the input terms are zero results only in the elimination of the additions in the first transform stage and therefore does not bring significant computational savings. By contrast, if $N = M^2$ and the transform is calculated with the recursive filtering circuit of Fig. 2, a factor of about two improvement in computing efficiency can be obtained. This is due to the fact that actual computation proceeds in two steps: during the first step, for $0 \leq n < N/2$, the $N/2$ nonzero input samples a_n are entered into the recursive filters while recursive computation proceeds. During the second step, for $N/2 \leq n < N-1$, the input samples are zero and the recursive

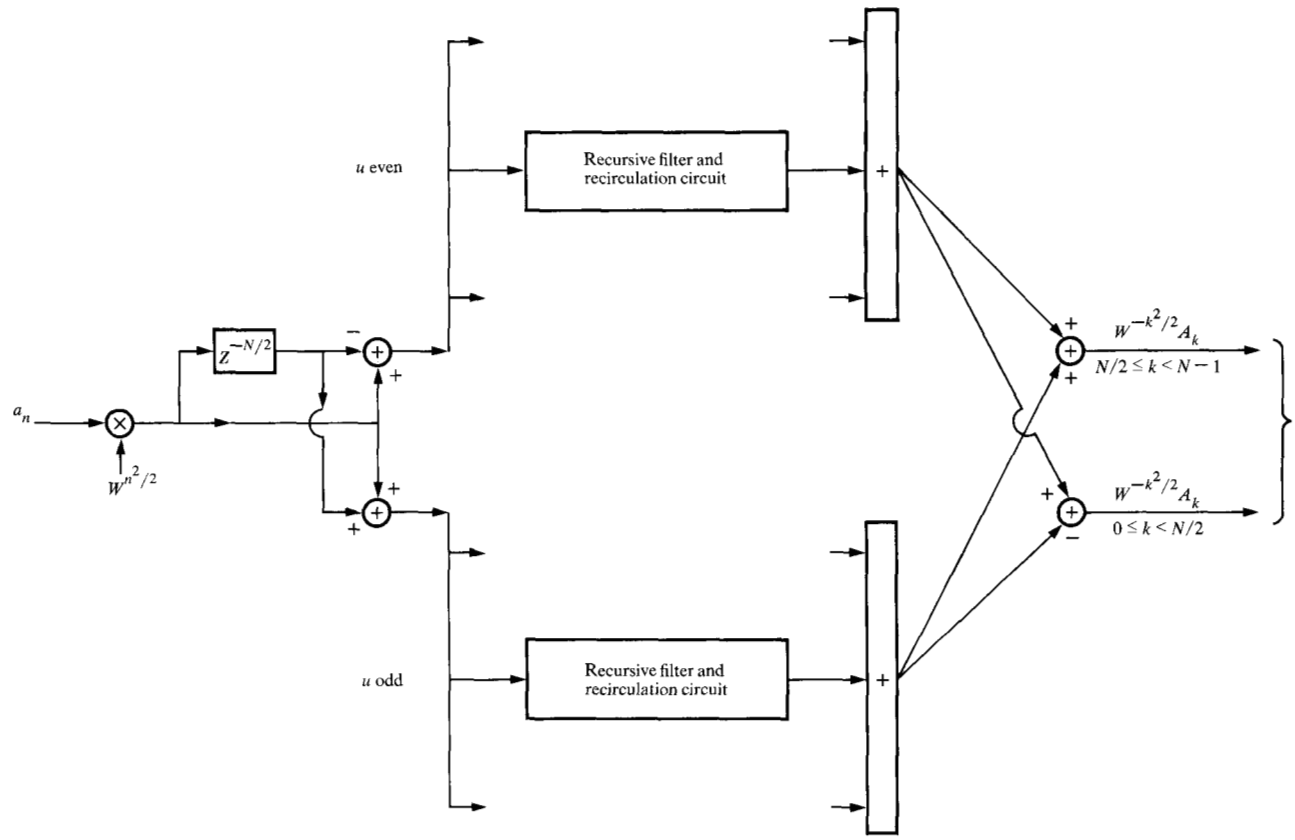

Figure 4 Fast direct transform circuit for M even.

computation reduces to a recirculation with fixed multiplication by W^{-uM}. The duration of these recirculations is $N/2$, corresponding to $M/2$ multiplications by W^{-uM}.

Under these conditions, the final output of the recursive filters at time N is the same as that available at time $N/2$, but multiplied by $W^{-uM^2/2}$ or by $(-1)^u$, as $W^{-M^2/2} \equiv -1$. This means that there is no need for actually carrying computations from time $N/2$ to N, provided that proper sign corrections are made on the filter outputs and that, for about the same amount of hardware, the speed of computation can be doubled.

A practical circuit for computing direct transforms can be realized as shown in Fig. 4. This circuit is very similar to that corresponding to Fig. 2, except that the recursive filters and recirculation circuits are split into two groups, one corresponding to u even and one corresponding to u odd. Such a circuit can be implemented with $2(M + 1)$ adders, one variable-shift circuit and $3N/2$ words of storage. As the actual computation time corresponds only to $N/2$ input samples, the number of additions per output sample is reduced to $M + 1$ as against $2M$ in the general case. This approach is applicable only to direct transforms. For the inverse transform, all the data values are nonzero and the computation must be performed with the circuit described in Fig. 2.

Alternate configurations

The approaches discussed in the preceding sections place some restrictions on transform length. One limitation is that we must have $N = M^2$. Another and less obvious limitation stems from the fact that computing an N-term transform with root W by means of a chirp Z-transform algorithm requires multiplications by powers of $W^{\frac{1}{2}}$. This reduces by a factor of two the maximum length of NTT's that can be computed without multiplications.

These limitations can be partly relieved by combining the chirp Z-transform technique with a partial FFT-type decomposition.

Assuming for instance a one-stage, radix 2 decomposition with decimation in frequency and an NTT,

$$A_k = \left(\sum_{n=0}^{N-1} a_n W^{nk}\right)_{\text{mod } p} \qquad k = 0, 1, \cdots, N-1, \quad (22)$$

we have

$$A_{2k} = \left(\sum_{n=0}^{N/2-1} (a_n + a_{n+N/2}) W^{2nk} \right)_{\text{mod } p}, \tag{23}$$

$$A_{2k+1} = \left(\sum_{n=0}^{N/2-1} (a_n - a_{n+N/2}) W^n W^{2nk} \right)_{\text{mod } p}$$

$$k = 0, 1, \cdots, \frac{N}{2} - 1. \tag{24}$$

If $N/2$ is a perfect square, with $N/2 = M^2$, the transform can be computed by a first stage with additions $a_n \pm a_{n+N/2}$, followed by a variable shift circuit for multiplication by W^{n^2} in the case of A_{2k} by W^{n^2+n} in the case of A_{2k+1} and followed by a bank of recursive circuits. It can be seen therefore that the multiplication required for one-stage FFT decomposition has been combined with that required for the chirp Z-transform and that the number of operations has been reduced to $2M + 1$ additions and one variable shift per output sample. In the case of complex transforms, it may be advantageous to use a one-stage, radix 4 decomposition with transforms of length $N = 4M^2$.

The applicability of the recursive filtering technique for computing NTT can therefore be extended to transform lengths multiple of a perfect square and the number of variable shift circuits reduced to only one when $N = M^2$, $2M^2$ or $4M^2$. This covers many practical cases corresponding to pseudo-Mersenne Transforms defined modulo $2^{25} - 1$, $2^{27} - 1$, $2^{49} - 1$, pseudo-Fermat Number Transforms defined modulo $2^{25} + 1$, $2^{27} + 1$, $2^{49} + 1$ and Fermat Number Transforms.

Concluding remarks

In this paper we have considered possible architectures for computing Number Theoretic Transforms. We have capitalized on the low cost of multiplications by fixed powers of two to devise configurations that are well adapted for pseudo-Mersenne Transforms having a number of terms which is a multiple of a perfect square. We have also shown that, thanks to a special coding scheme, this approach could be extended to cover the case of Fermat and pseudo-Fermat Number Transforms. For Fermat Number Transforms, the proposed technique, when used with FFT-type decompositions, covers a wide range of possible transform lengths. In the case of FNT, however, the proposed technique is somewhat suboptimal because the computation is not fully factorized, and the number of multiplications by variable powers of two is reduced at the expense of an increased number of additions.

References

1. C. M. Rader, "Discrete Convolutions via Mersenne Transforms," *IEEE Trans. Comput.* **C-21,** 1269 (1972).
2. R. C. Agarwal and C. S. Burrus, "Fast Convolution Using Fermat Number Transforms with Applications to Digital Filtering," *IEEE Trans. Acoust. Speech Signal Process.* **ASSP-22,** 87 (1974).
3. R. C. Agarwal and C. S. Burrus, "Number Theoretic Transforms to Implement Fast Digital Convolution," *Proc. IEEE* **63,** 550 (1975).
4. H. J. Nussbaumer, "Digital Filtering Using Complex Mersenne Transforms," *IBM J. Res. Develop.* **20,** 498 (1976).
5. G. D. Bergland and H. W. Hale, "Digital Real Time Spectral Analysis," *IEEE Trans. Electron. Comput.* **EC-16,** 180 (1967).
6. H. L. Groginsky and G. A. Works, "A Pipeline Fast Fourier Transform," *IEEE Trans. Comput.* **C-19,** 1015 (1970).
7. G. C. O'Leary, "Nonrecursive Digital Filtering Using Cascade Fast Fourier Transformers," *IEEE Trans. Audio Electroacoust.* **AU-18,** 177 (1970).
8. J. H. McClellan, "Hardware for the Fermat Number Transform," *Report ESD-TR-75-149,* Lincoln Laboratory, Massachusetts Institute of Technology, Cambridge, MA, 1975.
9. H. J. Nussbaumer, "Complex Convolutions via Fermat Number Transforms," *IBM J. Res. Develop.* **20,** 282 (1976).
10. H. J. Nussbaumer, "Digital Filtering Using Pseudo-Fermat Number Transforms," *IEEE Trans. Acoust. Speech Signal Process.*, to be published.
11. L. R. Rabiner, R. W. Schafer, and C. M. Rader, "The Chirp Z-Transform Algorithm and its Application," *Bell Syst. Tech. J.* **48,** 1249 (1969).
12. L. I. Bluestein, "A Linear Filtering Approach to the Computation of the Discrete Fourier Transform," Northeast Electronics Research and Engineering Meeting, paper no. 10-218, 1968. IEEE, New York.
13. L. I. Bluestein, "A Linear Filtering Approach to the Computation of Discrete Fourier Transform," *IEEE Trans. Audio Electroacoust.* **AU-18,** 451 (1970).
14. G. R. Nudd and O. W. Otto, "Chirp Signal Processing using Acoustic Surface Wave Filters," *1975 Ultrasonics Symposium Proceedings,* p. 350. IEEE, New York.
15. A. V. Oppenheim and R. W. Schafer, *Digital Signal Processing,* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1975, Ch. 6, p. 307.
16. L. M. Leibowitz, "A Simplified Binary Arithmetic for the Fermat Number Transform," *IEEE Trans. Acoust. Speech Signal Process.* **ASSP-24,** 356 (1976).
17. B. Gold, C. M. Rader, A. V. Oppenheim, and T. G. Stockham, *Digital Processing of Signals,* McGraw-Hill Book Company, Inc., New York, 1969, Ch. 7, p. 203.

Received June 11, 1976; revised November 23, 1976

The author is located at the Compagnie IBM France, Centre d'Etudes et Recherches, 06610 La Gaude, France.