Winograd's Short DFT Algorithms*

C. Sidney Burrus
Ivan Selesnick

In 1976, S. Winograd [20] presented a new DFT algorithm which had significantly fewer multiplications than the Cooley-Tukey FFT which had been published eleven years earlier. This new Winograd Fourier Transform Algorithm (WFTA) is based on the type- one index map from Multidimensional Index Mapping[1] with each of the relatively prime length short DFT's calculated by very efficient special algorithms. It is these short algorithms that this section will develop. They use the index permutation of Rader described in the another module to convert the prime length short DFT's into cyclic convolutions. Winograd developed a method for calculating digital convolution with the minimum number of multiplications. These optimal algorithms are based on the polynomial residue reduction techniques of Polynomial Description of Signals: Equation 1[2] to break the convolution into multiple small ones [2], [12], [14], [23], [21], [9].

The operation of discrete convolution defined by

$$y(n) = \sum_k h(n-k)\ x(k) \tag{1}$$

is called a **bilinear** operation because, for a fixed $h(n)$, $y(n)$ is a linear function of $x(n)$ and for a fixed $x(n)$ it is a linear function of $h(n)$. The operation of cyclic convolution is the same but with all indices evaluated modulo N.

Recall from Polynomial Description of Signals: Equation 3[3] that length-N cyclic convolution of $x(n)$ and $h(n)$ can be represented by polynomial multiplication

$$Y(s) = X(s)\ H(s) \mod \left(s^N - 1\right) \tag{2}$$

This bilinear operation of (1) and (2) can also be expressed in terms of linear matrix operators and a simpler bilinear operator denoted by o which may be only a simple element-by-element multiplication of the two vectors [12], [9], [10]. This matrix formulation is

$$Y = C\left[AXoBH\right] \tag{3}$$

where X, H and Y are length-N vectors with elements of $x(n)$, $h(n)$ and $y(n)$ respectively. The matrices A and B have dimension M x N , and C is N x M with $M \geq N$. The elements of A, B, and C are constrained to be simple; typically small integers or rational numbers. It will be these matrix operators that do the equivalent of the residue reduction on the polynomials in (2).

*Version 1.14: Apr 10, 2010 1:32 pm -0500
†http://creativecommons.org/licenses/by/2.0/
[1]"Multidimensional Index Mapping" <http://cnx.org/content/m16326/latest/>
[2]"Polynomial Description of Signals", (1) <http://cnx.org/content/m16327/latest/#uid1>
[3]"Polynomial Description of Signals", (3) <http://cnx.org/content/m16327/latest/#uid3>

In order to derive a useful algorithm of the form (3) to calculate (1), consider the polynomial formulation (2) again. To use the residue reduction scheme, the modulus is factored into relatively prime factors. Fortunately the factoring of this particular polynomial, $s^N - 1$, has been extensively studied and it has considerable structure. When factored over the rationals, which means that the only coefficients allowed are rational numbers, the factors are called cyclotomic polynomials [2], [12], [14]. The most interesting property for our purposes is that most of the coefficients of cyclotomic polynomials are zero and the others are plus or minus unity for degrees up to over one hundred. This means the residue reduction will generally require no multiplications.

The operations of reducing $X(s)$ and $H(s)$ in (2) are carried out by the matrices A and B in (3). The convolution of the residue polynomials is carried out by the o operator and the recombination by the CRT is done by the C matrix. More details are in [2], [12], [14], [9], [10] but the important fact is the A and B matrices usually contain only zero and plus or minus unity entries and the C matrix only contains rational numbers. The only general multiplications are those represented by o. Indeed, in the theoretical results from computational complexity theory, these real or complex multiplications are usually the only ones counted. In practical algorithms, the rational multiplications represented by C could be a limiting factor.

The $h(n)$ terms are fixed for a digital filter, or they represent the W terms from Multidimensional Index Mapping: Equation 1[4] if the convolution is being used to calculate a DFT. Because of this, $d = BH$ in (3) can be precalculated and only the A and C operators represent the mathematics done at execution of the algorithm. In order to exploit this feature, it was shown [23], [9] that the properties of (3) allow the exchange of the more complicated operator C with the simpler operator B. Specifically this is given by

$$Y = C\left[AXoBH\right] \tag{4}$$

$$Y' = B^T\left[AXoC^TH'\right] \tag{5}$$

where H' has the same elements as H, but in a permuted order, and likewise Y' and Y. This very important property allows precomputing the more complicated C^TH' in (5) rather than BH as in (3).

Because BH or C^TH' can be precomputed, the bilinear form of (3) and (5) can be written as a linear form. If an M x M diagonal matrix D is formed from $d = C^TH$, or in the case of (3), $d = BH$, assuming a commutative property for o, (5) becomes

$$Y' = B^TDAX \tag{6}$$

and (3) becomes

$$Y = CDAX \tag{7}$$

In most cases there is no reason not to use the same reduction operations on X and H, therefore, B can be the same as A and (6) then becomes

$$Y' = A^TDAX \tag{8}$$

In order to illustrate how the residue reduction is carried out and how the A matrix is obtained, the length-5 DFT algorithm started in The DFT as Convolution or Filtering: Matrix 1[5] will be continued. The DFT is first converted to a length-4 cyclic convolution by the index permutation from The DFT as Convolution or Filtering: Equation 3[6] to give the cyclic convolution in The DFT as Convolution or Filtering[7]. To avoid

[4] "Multidimensional Index Mapping", (1) <http://cnx.org/content/m16326/latest/#uid1>
[5] "The DFT as Convolution or Filtering", (12) <http://cnx.org/content/m16328/latest/#uid9>
[6] "The DFT as Convolution or Filtering", (3) <http://cnx.org/content/m16328/latest/#uid4>
[7] "The DFT as Convolution or Filtering" <http://cnx.org/content/m16328/latest/>

confusion from the permuted order of the data $x(n)$ in The DFT as Convolution or Filtering[8], the cyclic convolution will first be developed without the permutation, using the polynomial $U(s)$

$$U(s) = x(1) + x(3)\,s + x(4)\,s^2 + x(2)\,s^3 \tag{9}$$

$$U(s) = u(0) + u(1)\,s + u(2)\,s^2 + u(3)\,s^3 \tag{10}$$

and then the results will be converted back to the permuted $x(n)$. The length-4 cyclic convolution in terms of polynomials is

$$Y(s) = U(s)\ \ H(s) \mod \left(s^4 - 1\right) \tag{11}$$

and the modulus factors into three cyclotomic polynomials

$$s4 - 1 = \left(s^2 - 1\right)\left(s^2 + 1\right) \tag{12}$$

$$= (s-1)(s+1)\left(s^2 + 1\right) \tag{13}$$

$$= P_1\ P_2\ P_3 \tag{14}$$

Both $U(s)$ and $H(s)$ are reduced modulo these three polynomials. The reduction modulo P_1 and P_2 is done in two stages. First it is done modulo $\left(s^2 - 1\right)$, then that residue is further reduced modulo $(s-1)$ and $(s+1)$.

$$U(s) = u0 + u1s + u_2s^2 + u_3s^3 \tag{15}$$

$$U'(s) = ((U(s)))_{(s^2-1)} = (u_0 + u_2) + (u_1 + u_3)\,s \tag{16}$$

$$U1(s) = \left(\left(U'(s)\right)\right)_{P_1} = (u_0 + u_1 + u_2 + u_3) \tag{17}$$

$$U2(s) = \left(\left(U'(s)\right)\right)_{P_2} = (u_0 - u_1 + u_2 - u_3) \tag{18}$$

$$U3(s) = ((U(s)))_{P_3} = (u_0 - u_2) + (u_1 - u_3)\,s \tag{19}$$

The reduction in (16) of the data polynomial (15) can be denoted by a matrix operation on a vector which has the data as entries.

$$\begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} u_0 + u_2 \\ u_1 + u_3 \end{bmatrix} \tag{20}$$

and the reduction in (19) is

$$\begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} u_0 - u_2 \\ u_1 - u_3 \end{bmatrix} \tag{21}$$

[8] "The DFT as Convolution or Filtering" <http://cnx.org/content/m16328/latest/>

Combining (20) and (21) gives one operator

$$\begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} u_0 + u_2 \\ u_1 + u_3 \\ u_0 - u_2 \\ u_1 - u_3 \end{bmatrix} = \begin{bmatrix} u_0 + u_2 \\ u_1 + u_3 \\ u_0 - u_2 \\ u_1 - u_3 \end{bmatrix} = \begin{bmatrix} w_0 \\ w_1 \\ v_0 \\ v_1 \end{bmatrix} \tag{22}$$

Further reduction of $v_0 + v_1 s$ is not possible because $P_3 = s^2 + 1$ cannot be factored over the rationals. However $s^2 - 1$ can be factored into $P_1 P_2 = (s-1)(s+1)$ and, therefore, $w_0 + w_1 s$ can be further reduced as was done in (17) and (18) by

$$\begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} = w_0 + w_1 = u_0 + u_2 + u_1 + u_3 \tag{23}$$

$$\begin{bmatrix} 1 & -1 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} = w_0 - w_1 = u_0 + u_2 - u_1 - u_3 \tag{24}$$

Combining (22), (23) and (24) gives

$$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} r_0 \\ r_1 \\ v_0 \\ v_1 \end{bmatrix} \tag{25}$$

The same reduction is done to $H(s)$ and then the convolution of (11) is done by multiplying each residue polynomial of $X(s)$ and $H(s)$ modulo each corresponding cyclotomic factor of $P(s)$ and finally a recombination using the polynomial Chinese Remainder Theorem (CRT) as in Polynomial Description of Signals: Equation 9[9] and Polynomial Description of Signals: Equation 13[10].

$$Y(s) = K_1(s) U_1(s) H_1(s) + K_2(s) U_2(s) H_2(s) + K_3(s) U_3(s) H_3(s) \tag{26}$$

mod $\left(s^4 - 1\right)$

where $U_1(s) = r_1$ and $U_2(s) = r_2$ are constants and $U_3(s) = v_0 + v_1 s$ is a first degree polynomial. U_1 times H_1 and U_2 times H_2 are easy, but multiplying U_3 time H_3 modulo $\left(s^2 + 1\right)$ is more difficult.

The multiplication of $U_3(s)$ times $H_3(s)$ can be done by the Toom-Cook algorithm [2], [12], [14] which can be viewed as Lagrange interpolation or polynomial multiplication modulo a special polynomial with three arbitrary coefficients. To simplify the arithmetic, the constants are chosen to be plus and minus one and zero. The details of this can be found in [2], [12], [14]. For this example it can be verified that

$$((v0 + v1s)(h0 + h1s))_{s^2+1} = (v_0 h_0 - v_1 h_1) + (v_0 h_1 + v_1 h_0) s \tag{27}$$

which by the Toom-Cook algorithm or inspection is

$$\begin{bmatrix} 1 & -1 & 0 \\ -1 & -1 & 1 \end{bmatrix} \left[\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} v_0 \\ v_1 \end{bmatrix} o \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} h_0 \\ h_1 \end{bmatrix} \right] = \begin{bmatrix} y_0 \\ y_1 \end{bmatrix} \tag{28}$$

[9] "Polynomial Description of Signals", (9) <http://cnx.org/content/m16327/latest/#uid10>
[10] "Polynomial Description of Signals", (13) <http://cnx.org/content/m16327/latest/#uid14>

where o signifies point-by-point multiplication. The total A matrix in (3) is a combination of (25) and (28) giving

$$AX = A_1 A_2 A_3 X \tag{29}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} r_0 \\ r_1 \\ v_0 \\ v_1 \end{bmatrix} \tag{30}$$

where the matrix A_3 gives the residue reduction $s^2 - 1$ and $s^2 + 1$, the upper left-hand part of A_2 gives the reduction modulo $s - 1$ and $s + 1$, and the lower right-hand part of A1 carries out the Toom-Cook algorithm modulo $s^2 + 1$ with the multiplication in (5). Notice that by calculating (30) in the three stages, seven additions are required. Also notice that A_1 is not square. It is this "expansion" that causes more than N multiplications to be required in o in (5) or D in (6). This staged reduction will derive the A operator for (5)

The method described above is very straight-forward for the shorter DFT lengths. For $N = 3$, both of the residue polynomials are constants and the multiplication given by o in (3) is trivial. For $N = 5$, which is the example used here, there is one first degree polynomial multiplication required but the Toom-Cook algorithm uses simple constants and, therefore, works well as indicated in (28). For $N = 7$, there are two first degree residue polynomials which can each be multiplied by the same techniques used in the $N = 5$ example. Unfortunately, for any longer lengths, the residue polynomials have an order of three or greater which causes the Toom-Cook algorithm to require constants of plus and minus two and worse. For that reason, the Toom-Cook method is not used, and other techniques such as index mapping are used that require more than the minimum number of multiplications, but do not require an excessive number of additions. The resulting algorithms still have the structure of (8). Blahut [2] and Nussbaumer [14] have a good collection of algorithms for polynomial multiplication that can be used with the techniques discussed here to construct a wide variety of DFT algorithms.

The constants in the diagonal matrix D can be found from the CRT matrix C in (5) using $d = C^T H$' for the diagonal terms in D. As mentioned above, for the smaller prime lengths of 3, 5, and 7 this works well but for longer lengths the CRT becomes very complicated. An alternate method for finding D uses the fact that since the linear form (6) or (8) calculates the DFT, it is possible to calculate a known DFT of a given $x(n)$ from the definition of the DFT in Multidimensional Index Mapping: Equation 1[11] and, given the A matrix in (8), solve for D by solving a set of simultaneous equations. The details of this procedure are described in [9].

A modification of this approach also works for a length which is an odd prime raised to some power: $N = P^M$. This is a bit more complicated [12], [23] but has been done for lengths of 9 and 25. For longer lengths, the conventional Cooley-Tukey type- two index map algorithm seems to be more efficient. For powers of two, there is no primitive root, and therefore, no simple conversion of the DFT into convolution. It is possible to use two generators [12], [14], [21] to make the conversion and there exists a set of length 4, 8, and 16 DFT algorithms of the form in (8) in [12].

In Table 1 an operation count of several short DFT algorithms is presented. These are practical algorithms that can be used alone or in conjunction with the index mapping to give longer DFT's as shown in The Prime Factor and Winograd Fourier Transform Algorithms[12]. Most are optimized in having either the theoretical minimum number of multiplications or the minimum number of multiplications without requiring a very large number of additions. Some allow other reasonable trade-offs between numbers of multiplications and

[11] "Multidimensional Index Mapping", (1) <http://cnx.org/content/m16326/latest/#uid1>
[12] "The Prime Factor and Winograd Fourier Transform Algorithms" <http://cnx.org/content/m16335/latest/>

additions. There are two lists of the number of multiplications. The first is the number of actual floating point multiplications that must be done for that length DFT. Some of these (one or two in most cases) will be by rational constants and the others will be by irrational constants. The second list is the total number of multiplications given in the diagonal matrix D in (8). At least one of these will be unity (the one associated with $X(0)$) and in some cases several will be unity (for $N = 2^M$). The second list is important in programming the WFTA in The Prime Factor and Winograd Fourier Transform Algorithm: The Winograd Fourier Transform Algorithm[13].

Length N	Mult Non-one	Mult Total	Adds
2	0	4	4
3	4	6	12
4	0	8	16
5	10	12	34
7	16	18	72
8	4	16	52
9	20	22	84
11	40	42	168
13	40	42	188
16	20	36	148
17	70	72	314
19	76	78	372
25	132	134	420
32	68	-	388

Table 1: Number of Real Multiplications and Additions for a Length-N DFT of Complex Data

Because of the structure of the short DFTs, the number of real multiplications required for the DFT of real data is exactly half that required for complex data. The number of real additions required is slightly less than half that required for complex data because $(N-1)$ of the additions needed when N is prime add a real to an imaginary, and that is not actually performed. When $N = 2m$, there are $(N-2)$ of these pseudo additions. The special case for real data is discussed in [4], [7], [19].

The structure of these algorithms are in the form of $X = CDAX$ or B^TDAX or A^TDAX from (5) and (8). The A and B matrices are generally M by N with $M \geq N$ and have elements that are integers, generally 0 or ± 1. A pictorial description is given in Figure 1.

[13] "The Prime Factor and Winograd Fourier Transform Algorithms": Section The Winograd Fourier Transform Algorithm <http://cnx.org/content/m16335/latest/#uid12>

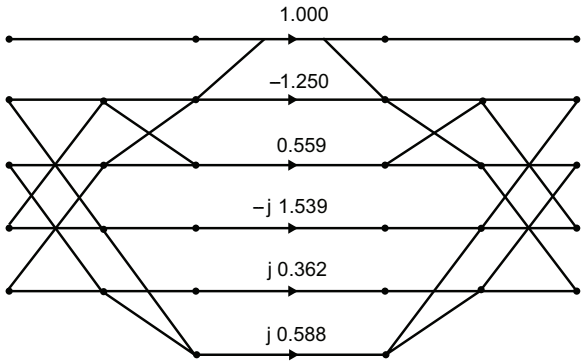

Figure 1: Flow Graph for the Length-5 DFT

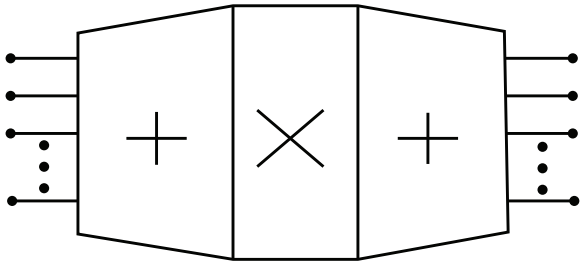

Figure 2: Block Diagram of a Winograd Short DFT

The flow graph in Figure 1 should be compared with the matrix description of (8) and (30), and with the programs in [2], [12], [3], [14] and the appendices. The shape in Figure 2 illustrates the expansion of the data by A. That is to say, AX has more entries than X because $M > N$. The A operator consists of additions, the D operator gives the M multiplications (some by one) and A^T contracts the data back to N values with additions only. M is one half the second list of multiplies in Table 1.

An important characteristic of the D operator in the calculation of the DFT is its entries are either purely real or imaginary. The reduction of the W vector by $\left(s^{(N-1)/2} - 1\right)$ and $\left(s^{(N-1)/2} + 1\right)$ separates the real and the imaginary constants. This is discussed in [23], [9]. The number of multiplications for complex data is only twice those necessary for real data, not four times.

Although this discussion has been on the calculation of the DFT, very similar results are true for the calculation of convolution and correlation, and these will be further developed in Algorithms for Data with Restrictions[14]. The $A^T DA$ structure and the picture in Figure 2 are the same for convolution. Algorithms and operation counts can be found in [2], [14], [1].

[14] "Algorithms for Data with Restrictions" <http://cnx.org/content/m16338/latest/>

1 The Bilinear Structure

The bilinear form introduced in (3) and the related linear form in (6) are very powerful descriptions of both the DFT and convolution.

$$\text{Bilinear: } Y = C\left[AX \ o \ BH\right] \tag{31}$$

$$\text{Linear: } Y = CDA \ X \tag{32}$$

Since (31) is a bilinear operation defined in terms of a second bilinear operator o , this formulation can be nested. For example if o is itself defined in terms of a second bilinear operator @, by

$$X \ o \ H = C^{'}\left[A^{'}X \ @ \ B^{'}H\right] \tag{33}$$

then (31) becomes

$$Y = CC^{'}\left[A^{'}AX \ @ \ B^{'}BH\right] \tag{34}$$

For convolution, if A represents the polynomial residue reduction modulo the cyclotomic polynomials, then A is square (e.g. (25) and o represents multiplication of the residue polynomials modulo the cyclotomic polynomials. If A represents the reduction modulo the cyclotomic polynomials plus the Toom-Cook reduction as was the case in the example of (30), then A is NxM and o is term-by- term simple scalar multiplication. In this case AX can be thought of as a transform of X and C is the inverse transform. This is called a rectangular transform [1] because A is rectangular. The transform requires only additions and convolution is done with M multiplications. The other extreme is when A represents reduction over the N complex roots of $s^N - 1$. In this case A is the DFT itself, as in the example of (43), and o is point by point complex multiplication and C is the inverse DFT. A trivial case is where A, B and C are identity operators and o is the cyclic convolution.

This very general and flexible bilinear formulation coupled with the idea of nesting in (34) gives a description of most forms of convolution.

2 Winograd's Complexity Theorems

Because Winograd's work [2], [12], [23], [21], [22], [24] has been the foundation of the modern results in efficient convolution and DFT algorithms, it is worthwhile to look at his theoretical conclusions on optimal algorithms. Most of his results are stated in terms of polynomial multiplication as Polynomial Description of Signals: Equation 3[15] or (11). The measure of computational complexity is usually the number of multiplications, and only certain multiplications are counted. This must be understood in order not to misinterpret the results.

This section will simply give a statement of the pertinent results and will not attempt to derive or prove anything. A short interpretation of each theorem will be given to relate the result to the algorithms developed in this chapter. The indicated references should be consulted for background and detail.

Theorem 1[23] Given two polynomials, $x(s)$ and $h(s)$, of degree N and M respectively, each with indeterminate coefficients that are elements of a field H, $N+M+1$ multiplications are necessary to compute the coefficients of the product polynomial $x(s)h(s)$. Multiplication by elements of the field G (the field of constants), which is contained in H, are not counted and G contains at least $N+M$ distinct elements.

The upper bound in this theorem can be realized by choosing an arbitrary modulus polynomial $P(s)$ of degree $N+M+1$ composed of $N+M+1$ distinct linear polynomial factors with coefficients in G which, since its degree is greater than the product $x(s)h(s)$, has no effect on the product, and by reducing $x(s)$ and $h(s)$ to $N+M+1$ residues modulo the $N+M+1$ factors of $P(s)$. These residues are multiplied by each other, requiring $N+M+1$ multiplications, and the results recombined using the Chinese remainder

[15] "Polynomial Description of Signals", (3) <http://cnx.org/content/m16327/latest/#uid3>

theorem (CRT). The operations required in the reduction and recombination are not counted, while the residue multiplications are. Since the modulus $P(s)$ is arbitrary, its factors are chosen to be simple so as to make the reduction and CRT simple. Factors of zero, plus and minus unity, and infinity are the simplest. Plus and minus two and other factors complicate the actual calculations considerably, but the theorem does not take that into account. This algorithm is a form of the Toom-Cook algorithm and of Lagrange interpolation [2], [12], [14], [23]. For our applications, H is the field of reals and G the field of rationals.

Theorem 2[23] If an algorithm exists which computes $x(s)h(s)$ in $N+M+1$ multiplications, all but one of its multiplication steps must necessarily be of the form

$$mk = \left(gk' + x(gk)\right)\left(gk'' + h(gk)\right) \quad \text{for } k = 0, 1, ..., N+M \tag{35}$$

where g_k are distinct elements of G; and g_k' and g_k'' are arbitrary elements of G

This theorem states that the structure of an optimal algorithm is essentially unique although the factors of $P(s)$ may be chosen arbitrarily.

Theorem 3[23] Let $P(s)$ be a polynomial of degree N and be of the form $P(s) = Q(s)k$, where $Q(s)$ is an irreducible polynomial with coefficients in G and k is a positive integer. Let $x(s)$ and $h(s)$ be two polynomials of degree at least $N-1$ with coefficients from H, then $2N-1$ multiplications are required to compute the product $x(s)h(s)$ modulo $P(s)$.

This theorem is similar to Theorem 1 (p. 8) with the operations of the reduction of the product modulo $P(s)$ not being counted.

Theorem 4[23] Any algorithm that computes the product $x(s)h(s)$ modulo $P(s)$ according to the conditions stated in Theorem 3 and requires $2N-1$ multiplications will necessarily be of one of three structures, each of which has the form of Theorem 2 internally.

As in Theorem 2 (p. 9), this theorem states that only a limited number of possible structures exist for optimal algorithms.

Theorem 5[23] If the modulus polynomial $P(s)$ has degree N and is not irreducible, it can be written in a unique factored form $P(s) = P_1^{m_1}(s) P_2^{m_2}(s) ... P_k^{m_k}(s)$ where each of the $P_i(s)$ are irreducible over the allowed coefficient field G. $2N-k$ multiplications are necessary to compute the product $x(s)h(s)$ modulo $P(s)$ where $x(s)$ and $h(s)$ have coefficients in H and are of degree at least $N-1$. All algorithms that calculate this product in $2N-k$ multiplications must be of a form where each of the k residue polynomials of $x(s)$ and $h(s)$ are separately multiplied modulo the factors of $P(s)$ via the CRT.

Corollary: If the modulus polynomial is $P(s) = s^N - 1$, then $2N - t(N)$ multiplications are necessary to compute $x(s)h(s)$ modulo $P(s)$, where $t(N)$ is the number of positive divisors of N.

Theorem 5 (p. 9) is very general since it allows a general modulus polynomial. The proof of the upper bound involves reducing $x(s)$ and $h(s)$ modulo the k factors of $P(s)$. Each of the k irreducible residue polynomials is then multiplied using the method of Theorem 4 (p. 9) requiring $2Ni-1$ multiplies and the products are combined using the CRT. The total number of multiplies from the k parts is $2N-k$. The theorem also states the structure of these optimal algorithms is essentially unique. The special case of $P(s) = s^N - 1$ is interesting since it corresponds to cyclic convolution and, as stated in the corollary, k is easily determined. The factors of $s^N - 1$ are called cyclotomic polynomials and have interesting properties [2], [12], [14].

Theorem 6[23], [21] Consider calculating the DFT of a prime length real-valued number sequence. If G is chosen as the field of rational numbers, the number of real multiplications necessary to calculate a length-P DFT is $u(DFT(N)) = 2P - 3 - t(P-1)$ where $t(P-1)$ is the number of divisors of $P-1$.

This theorem not only gives a lower limit on any practical prime length DFT algorithm, it also gives practical algorithms for $N = 3, 5$, and 7. Consider the operation counts in Table 1 to understand this theorem. In addition to the real multiplications counted by complexity theory, each optimal prime-length algorithm will have one multiplication by a rational constant. That constant corresponds to the residue modulo (s-1) which always exists for the modulus $P(s) = s^{N-1} - 1$. In a practical algorithm, this multiplication must be carried out, and that accounts for the difference in the prediction of Theorem 6 (p. 9) and count in Table 1. In addition, there is another operation that for certain applications must be counted as a multiplication.

That is the calculation of the zero frequency term $X(0)$ in the first row of the example in The DFT as Convolution or Filtering: Matrix 1[16]. For applications to the WFTA discussed in The Prime Factor and Winograd Fourier Transform Algorithms: The Winograd Fourier Transform Algorithm[17], that operation must be counted as a multiply. For lengths longer than 7, optimal algorithms require too many additions, so compromise structures are used.

Theorem 7[24], [6] If G is chosen as the field of rational numbers, the number of real multiplications necessary to calculate a length-N DFT where N is a prime number raised to an integer power: $N = Pm$, is given by

$$u(DFT(N)) = 2N - ((m2 + m)/2)\, t(P-1) - m - 1 \tag{36}$$

where $t(P-1)$ is the number of divisors of $(P-1)$.

This result seems to be practically achievable only for $N = 9$, or perhaps 25. In the case of $N = 9$, there are two rational multiplies that must be carried out and are counted in Table 1 but are not predicted by Theorem 7 (p. 10). Experience [8] indicates that even for $N = 25$, an algorithm based on a Cooley-Tukey FFT using a type 2 index map gives an over-all more balanced result.

Theorem 8[6] If G is chosen as the field of rational numbers, the number of real multiplications necessary to calculate a length-N DFT where $N = 2m$ is given by

$$u(DFT(N)) = 2N - m2 - m - 2 \tag{37}$$

This result is not practically useful because the number of additions necessary to realize this minimum of multiplications becomes very large for lengths greater than 16. Nevertheless, it proves the minimum number of multiplications required of an optimal algorithm is a linear function of N rather than of $NlogN$ which is that required of practical algorithms. The best practical power-of-two algorithm seems to the Split-Radix [5] FFT discussed in The Cooley-Tukey Fast Fourier Transform Algorithm: The Split-Radix FFT Algorithm[18].

All of these theorems use ideas based on residue reduction, multiplication of the residues, and then combination by the CRT. It is remarkable that this approach finds the minimum number of required multiplications by a constructive proof which generates an algorithm that achieves this minimum; and the structure of the optimal algorithm is, within certain variations, unique. For shorter lengths, the optimal algorithms give practical programs. For longer lengths the uncounted operations involved with the multiplication of the higher degree residue polynomials become very large and impractical. In those cases, efficient suboptimal algorithms can be generated by using the same residue reduction as for the optimal case, but by using methods other than the Toom-Cook algorithm of Theorem 1 (p. 8) to multiply the residue polynomials.

Practical long DFT algorithms are produced by combining short prime length optimal DFT's with the Type 1 index map from Multidimensional Index Mapping[19] to give the Prime Factor Algorithm (PFA) and the Winograd Fourier Transform Algorithm (WFTA) discussed in The Prime Factor and Winograd Fourier Transform Algorithms[20]. It is interesting to note that the index mapping technique is useful inside the short DFT algorithms to replace the Toom-Cook algorithm and outside to combine the short DFT's to calculate long DFT's.

3 The Automatic Generation of Winograd's Short DFTs

by Ivan Selesnick, Polytechnic Institute of New York University

[16] "The DFT as Convolution or Filtering", (12) <http://cnx.org/content/m16328/latest/#uid9>

[17] "The Prime Factor and Winograd Fourier Transform Algorithms": Section The Winograd Fourier Transform Algorithm <http://cnx.org/content/m16335/latest/#uid12>

[18] "The Cooley-Tukey Fast Fourier Transform Algorithm": Section The Split-Radix FFT Algorithm <http://cnx.org/content/m16334/latest/#uid16>

[19] "Multidimensional Index Mapping" <http://cnx.org/content/m16326/latest/>

[20] "The Prime Factor and Winograd Fourier Transform Algorithms" <http://cnx.org/content/m16335/latest/>

3.1 Introduction

Efficient prime length DFTs are important for two reasons. A particular application may require a prime length DFT and secondly, the maximum length and the variety of lengths of a PFA or WFTA algorithm depend upon the availability of prime length modules.

This [15], [18], [16], [17] discusses automation of the process Winograd used for constructing prime length FFTs [2], [8] for $N < 7$ and that Johnson and Burrus [9] extended to $N < 19$. It also describes a program that will design any prime length FFT in principle, and will also automatically generate the algorithm as a C program and draw the corresponding flow graph.

Winograd's approach uses Rader's method to convert a prime length DFT into a $P - 1$ length cyclic convolution, polynomial residue reduction to decompose the problem into smaller convolutions [2], [14], and the Toom-Cook algorithm [2], [13]. The Chinese Remainder Theorem (CRT) for polynomials is then used to recombine the shorter convolutions. Unfortunately, the design procedure derived directly from Winograd's theory becomes cumbersome for longer length DFTs, and this has often prevented the design of DFT programs for lengths greater than 19.

Here we use three methods to facilitate the construction of prime length FFT modules. First, the matrix exchange property [2], [9], [11] is used so that the transpose of the reduction operator can be used rather than the more complicated CRT reconstruction operator. This is then combined with the numerical method [9] for obtaining the multiplication coefficients rather than the direct use of the CRT. We also deviate from the Toom-Cook algorithm, because it requires too many additions for the lengths in which we are interested. Instead we use an iterated polynomial multiplication algorithm [2]. We have incorporated these three ideas into a single structural procedure that automates the design of prime length FFTs.

3.2 Matrix Description

It is important that each step in the Winograd FFT can be described using matrices. By expressing cyclic convolution as a bilinear form, a compact form of prime length DFTs can be obtained.

If y is the cyclic convolution of h and x, then y can be expressed as

$$y = C\left[Ax. * Bh\right] \tag{38}$$

where, using the Matlab convention, $.*$ represents point by point multiplication. When A,B, and C are allowed to be complex, A and B are seen to be the DFT operator and C, the inverse DFT. When only real numbers are allowed, A, B, and C will be rectangular. This form of convolution is presented with many examples in [2]. Using the matrix exchange property explained in [2] and [9] this form can be written as

$$y = RB^T\left[C^T Rh. * Ax\right] \tag{39}$$

where R is the permutation matrix that reverses order.

When h is fixed, as it is when considering prime length DFTs, the term $C^T Rh$ can be precomputed and a diagonal matrix D formed by $D = diag\{C^T Rh\}$. This is advantageous because in general, C is more complicated than B, so the ability to "hide" C saves computation. Now $y = RB^T DAx$ or $y = RA^T DAx$ since A and B can be the same; they implement a polynomial reduction. The form $y = R^T DAxT$ can also be used for the prime length DFTs, it is only necessary to permute the entries of x and to ensure that the DC term is computed correctly. The computation of the DC term is simple, for the residue of a polynomial modulo $a - 1$ is always the sum of the coefficients. After adding the x_0 term of the original input sequence, to the $s - l$ residue, the DC term is obtained. Now $DFT\{x\} = RA^T DAx$. In [9] Johnson observes that by permuting the elements on the diagonal of D, the output can be permuted, so that the R matrix can be hidden in D, and $DFT\{x\} = A^T DAx$. From the knowledge of this form, once A is found, D can be found numerically [9].

3.3 Programming the Design Procedure

Because each of the above steps can be described by matrices, the development of a prime length FFTs is made convenient with the use of a matrix oriented programming language such as Matlab. After specifying the appropriate matrices that describe the desired FFT algorithm, generating code involves compiling the matrices into the desired code for execution.

Each matrix is a section of one stage of the flow graph that corresponds to the DFT program. The four stages are:

1. Permutation Stage: Permutes input and output sequence.
2. Reduction Stage: Reduces the cyclic convolution to smaller polynomial products.
3. Polynomial Product Stage: Performs the polynomial multiplications.
4. Multiplication Stage: Implements the point-by-point multiplication in the bilinear form.

Each of the stages can be clearly seen in the flow graphs for the DFTs. Figure 3 shows the flow graph for a length 17 DFT algorithm that was automatically drawn by the program.

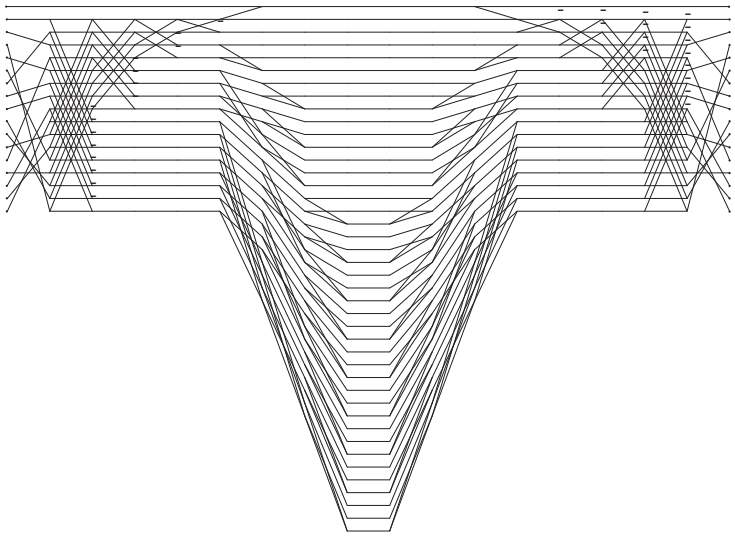

Figure 3: Flowgraph of length-17 DFT

The programs that accomplish this process are written in Matlab and C. Those that compute the appropriate matrices are written in Matlab. These matrices are then stored as two ASCII files, with the dimensions in one and the matrix elements in the second. A C program then reads the flies and compiles them to produce the final FFT program in C [18]

3.4 The Reduction Stage

The reduction of an N^{th} degree polynomial, $X(s)$, modulo the cyclotomic polynomial factors of $\left(s^N - 1\right)$ requires only additions for many N, however, the actual number of additions depends upon the way in which the reduction proceeds. The reduction is most efficiently performed in steps. For example, if $N = 4$ and

$\left((X(s))_{s-1}\right.$,$\left((X(s))_{s+1}\right.$ and $\left((X(s))_{s^2+1}\right.$ where the double parenthesis denote polynomial reduction modulo $(s-1)$, $s+1$, and s^2+1, then in the first step $((X(s)))_{s^2-1}$, and $((Xs))_{s^2+1}$ should be computed. In the second step, $((Xs))_{s-1}$ and $((Xs))_{s+1}$ can be found by reducing $((X(s)))_{s^2-1}$ This process is described by the diagram in Figure 4.

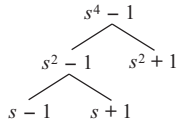

Figure 4: Factorization of s^4-1 in steps

When N is even, the appropriate first factorization is $\left(S^{N/2}-1\right)\left(s^{N/2}+1\right)$, however, the next appropriate factorization is frequently less obvious. The following procedure has been found to generate a factorization in steps that coincides with the factorization that minimizes the cumulative number of additions incurred by the steps. The prime factors of N are the basis of this procedure and their importance is clear from the useful well-known equation $s^N-1=\prod_{n|N}C_n(s)$ where $C_n(s)$ is the n^{th} cyclotomic polynomial.

We first introduce the following two functions defined on the positive integers,

$$\psi(N)=\text{the smallest prime factor of } N \text{ for} N>1 \tag{40}$$

and $\psi(1)=1$.

Suppose $P(s)$ is equal to either $\left(s^N-1\right)$ or an intermediate noncyclotomic polynomial appearing in the factorization process, for example, $\left(a^2-1\right)$, above. Write $P(s)$ in terms of its cyclotomic factors,

$$P(s)=C_{k_1}(s)\ C_{k_2}(s)\ \cdots C_{k_L} \tag{41}$$

define the two sets, G and G , by

$$G=\{k_1,\cdots,k_L\}\quad\text{and}\quad G\ =\{k/gcd(G):\ k\in G\} \tag{42}$$

and define the two integers, t and T, by

$$t=min\{\psi(k):k\in G\ ,k>1\}\ \text{and}\quad T=maxnu(k,t):k\in G\} \tag{43}$$

Then form two new sets,

$$A=\{k\in G:T\mid k\}\quad\text{and}\quad B=\{k\in G:T|k\} \tag{44}$$

The factorization of $P(s)$,

$$P(s)=\left(\prod_{k\in A}C_k(s)\right)\left(\prod_{k\in B}C_k(s)\right) \tag{45}$$

has been found useful in the procedure for factoring $\left(s^N-1\right)$. This is best illustrated with an example.

Example: $N=36$

Step 1. Let $P(s)=s^{36}-1$. Since $P=C_1C_2C_3C_4C_6C_9C_{12}C_{18}C_{36}$

$$G=G\ =\{1,2,3,4,6,9,12,18,36\} \tag{46}$$

$$t = min\{2,3\} = 2 \tag{47}$$

$$A = \{k \in G : 4|k\} = \{1,2,3,6,9,18\} \tag{48}$$

$$B = \{k \in G : 4|k\} = \{4,12,36\} \tag{49}$$

Hence the factorization of $s^{36}-1$ into two intermediate polynomials is as expected,

$$\prod_{k\in A} C_k(s) = s^{18}-1, \quad \prod_{k\in B} C_k(s) = s^{18}+1 \tag{50}$$

If a 36th degree polynomial, $X(s)$, is represented by a vector of coefficients, $X = (x_{35},\cdots,x_0)^{'}$, then $\left((X(s))_{s^{18}-1}\right.$ (represented by X') and $\left((X(s))_{s^{18}+1}\right.$ (represented by X") is given by

$$test \tag{51}$$

which entails 36 additions.

Step 2. This procedure is repeated with $P(s) = s^{18}-1$ and $P(s) = s^{18}+1$. We will just show it for the later. Let $P(s) = s^{18}+1$. Since $P = C_4C_{12}C_{36}$

$$G = \{4,12,36\}, \qquad G^{'} = \{l,3,9\} \tag{52}$$

$$t = min3 = 3 \tag{53}$$

$$T = max\nu(k,3) : k \in G = maxl,3,9 = 9 \tag{54}$$

$$A = k \in G : 9|k\} = \{4,12\} \tag{55}$$

$$B = k \in G : 9|k\} = \{36\} \tag{56}$$

This yields the two intermediate polynomials,

$$s^6+1, \quad \text{and} \quad s^{12}-s^6+1 \tag{57}$$

In the notation used above,

$$\begin{bmatrix} X^{'} \\ X^{''} \end{bmatrix} = \begin{bmatrix} I_6 & -I_6 & I_6 \\ I_6 & I_6 & \\ -I_6 & & I_6 \end{bmatrix} X \tag{58}$$

entailing 24 additions. Continuing this process results in a factorization in steps

In order to see the number of additions this scheme uses for numbers of the form $N = P-1$ (which is relevant to prime length FFT algorithms) figure 4 shows the number of additions the reduction process uses when the polynomial X(s) is real.

Figure 4: Number of Additions for Reduction Stage

3.5 The Polynomial Product Stage

The iterated convolution algorithm can be used to construct an N point linear convolution algorithm from shorter linear convolution algorithms [2]. Suppose the linear convolution y, of the n point vectors x and h (h known) is described by

$$y = E_n^T \ D \ E_n \ x \tag{59}$$

where E_n is an "expansion" matrix the elements of which are $\pm l$'s and 0's and D is an appropriate diagonal matrix. Because the only multiplications in this expression are by the elements of D, the number of multiplications required, $M(n)$, is equal to the number of rows of E_n. The number of additions is denoted by $A(n)$.

Given a matrix E_{n_1} and a matrix E_{n_2}, the iterated algorithm gives a method for combining E_{n_1} and E_{n_2} to construct a valid expansion matrix, E_n, for $N \leq n_1 n_2$. Specifically,

$$E_{n_1,n_2} = \left(I_{M(n_2)} \otimes E_{n_1}\right)\left(E_{n_2} \times I_{n_1}\right) \tag{60}$$

The product $n_1 n_2$ may be greater than N, for zeros can be (conceptually) appended to x. The operation count associated with E_{n_1,n_2} is

$$A(n_1, n_2) = n! A(n_2) + A(n_1) M n_2 \tag{61}$$

$$M(n_1, n_2) = M(n_1) \ M(n_2) \tag{62}$$

Although they are both valid expansion matrices, $E_{n_1,n_2} \neq E_{n_2,n_1}$ and $A_{n_1,n_2} \neq A_{n_2,n_1}$ Because $M_{n_1,n_2} \neq M_{n_2,n_1}$ it is desirable to chose an ordering of factors to minimize the additions incurred by the expansion matrix. The following [1], [14] follows from above.

3.5.1 Multiple Factors

Note that a valid expansion matrix, E_N, can be constructed from E_{n_1,n_2} and E_{n_3}, for $N \leq n_1 n_2 n_3$. In general, any number of factors can be used to create larger expansion matrices. The operation count associated with E_{n_1,n_2,n_3} is

$$A(n_1, n_2, n_3) = n_1 n_2 A(n_3) + n_1 A(n_2) M(n_3) + A(n_1) M(n_2) M(n_3) \tag{63}$$

$$M(n_1, n_2, n_3) = M(n_1) M(n_2) M(n_3) \tag{64}$$

These equations generalize in the predicted way when more factors are considered. Because the ordering of the factors is relevant in the equation for $A(.)$ but not for $M(.)$, it is again desirable to order the factors to minimize the number of additions. By exploiting the following property of the expressions for $A(.)$ and $M(.)$, the optimal ordering can be found [1].

reservation of Optimal Ordering. Suppose $A(n_1, n_2, n_3) \leq min\{A(n_{k_1}, n_{k_2}, n_{k_3}) : k_1, k_2, k_3 \in \{1, 2, 3\}$ and distinct$\}$, then

1.
$$A(n_1, n_2) \leq A(n_2, n_1) \tag{65}$$

2.
$$A(n_2, n_3) \leq A(n_3, n_2) \tag{66}$$

3.

$$A(n_1, n_3) \leq A(n_3, n_1) \tag{67}$$

The generalization of this property to more than two factors reveals that an optimal ordering of $\{n_1, \cdots, n_{L-i}\}$ is preserved in an optimal ordering of $\{n_1, \cdots n_L\}$. Therefore, if $(n_1, \cdots n_L)$ is an optimal ordering of $\{n_1, \cdots n_L\}$, then (n_k, n_{k+1}) is an optimal ordering of $\{n_k, n_{k+1}\}$ and consequently

$$\frac{A(n_k)}{M(n_k) - n_k} \leq \frac{A(n_{k+1})}{M(n_{k+1}) - n_{k+1}} \tag{68}$$

for all $k = 1, 2, \cdots, L-1$.

This immediately suggests that an optimal ordering of $\{n_1, \cdots n_L\}$ is one for which

$$\frac{A(n_1)}{M(n_1) - n_1} \cdots \frac{A(n_L)}{M(n_L) - n_L} \tag{69}$$

is nondecreasing. Hence, ordering the factors, $\{n_1, \cdots n_L\}$, to minimize the number of additions incurred by $E_{n_1, \cdots, n_L}$ simply involves computing the appropriate ratios.

3.6 Discussion and Conclusion

We have designed prime length FFTs up to length 53 that are as good as the previous designs that only went up to 19. Table 1 gives the operation counts for the new and previously designed modules, assuming complex inputs.

It is interesting to note that the operation counts depend on the factorability of $P - 1$. The primes 11, 23, and 47 are all of the form $1 + 2P_1$ making the design of efficient FFTs for these lengths more difficult.

Further deviations from the original Winograd approach than we have made could prove useful for longer lengths. We investigated, for example, the use of twiddle factors at appropriate points in the decomposition stage; these can sometimes be used to divide the cyclic convolution into smaller convolutions. Their use means, however, that the 'center* multiplications would no longer be by purely real or imaginary numbers.

N	Mult	Adds
7	16	72
11	40	168
13	40	188
17	82	274
19	88	360
23	174	672
29	190	766
31	160	984
37	220	920
41	282	1140
43	304	1416
47	640	2088
53	556	2038

Table 2: Operation counts for prime length DFTs

The approach in writing a program that writes another program is a valuable one for several reasons. Programming the design process for the design of prime length FFTs has the advantages of being practical, error-free, and flexible. The flexibility is important because it allows for modification and experimentation with different algorithmic ideas. Above all, it has allowed longer DFTs to be reliably designed.

More details on the generation of programs for prime length FFTs can be found in the 1993 Technical Report.

References

[1] R. C. Agarwal and J. W. Cooley. New algorithms for digital convolution. *IEEE Trans. on ASSP*, 25(2):3928211;410, October 1977.

[2] Richard E. Blahut. *Fast Algorithms for Digital Signal Processing.* Addison-Wesley, Reading, Mass., 1985.

[3] C. S. Burrus and T. W. Parks. *DFT/FFT and Convolution Algorithms.* John Wiley & Sons, New York, 1985.

[4] P. Duhamel. Implementation of 'split-radix' fft algorithms for complex, real, and real-symmetric data. *IEEE Trans. on ASSP*, 34:2858211;295, April 1986. A shorter version appeared in the ICASSP-85 Proceedings, p. 20.6, March 1985.

[5] P. Duhamel and H. Hollmann. Split radix fft algorithm. *Electronic Letters*, 20(1):148211;16, January 5 1984.

[6] M. T. Heideman and C. S. Burrus. On the number of multiplications necessary to compute a length-dft. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 34(1):918211;95, February 1986.

[7] M. T. Heideman, H. W. Johnson, and C. S. Burrus. Prime factor fft algorithms for real8211;valued series. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, page 28A.7.18211;4, San Diego, CA, March 1984.

[8] H. W. Johnson and C. S. Burrus. Large dft modules: N = 11, 13, 17, 19, and 25. Technical report 8105, Department of Electrical Engineering, Rice University, Houston, TX 772518211;1892, 1981.

[9] Howard W. Johnson and C. S. Burrus. On the structure of efficient dft algorithms. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 33(1):2488211;254, February 1985.

[10] D. P. Kolba and T. W. Parks. A prime factor fft algorithm using high speed convolution. *IEEE Trans. on ASSP*, 25:2818211;294, August 1977. also in.

[11] J. S. Lim and A. V. Oppenheim. *Advanced Topics in Signal Processing.* Prentice-Hall, Englewood Cliffs, NJ, 1988.

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

[13] Douglas G. Myers. *Digital Signal Processing, Efficient Convolution and Fourier Transform Techniques.* Prentice-Hall, Sydney, Australia, 1990.

[14] H. J. Nussbaumer. *Fast Fourier Transform and Convolution Algorithms.* Springer-Verlag, Heidelberg, Germany, second edition, 1981, 1982.

[15] I. W. Selesnick and C. S. Burrus. Automating the design of prime length fft programs. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, volume 1, page 1338211;136, ISCAS-92, San Diego, CA, May 1992.

[16] I. W. Selesnick and C. S. Burrus. Multidimensional mapping techniques for convolution. In *Proceedings of the IEEE International Conference on Signal Processing*, volume III, pages III–2888211;291, IEEE ICASSP-93, Minneapolis, April 1993.

[17] I. W. Selesnick and C. S. Burrus. Extending winograd's small convolution algorithm to longer lengths. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, volume 2, page 2.4498211;452, IEEE ISCAS-94, London, June 30 1994.

[18] Ivan W. Selesnick and C. Sidney Burrus. Automatic generation of prime length fft programs. *IEEE Transactions on Signal Processing*, 44(1):148211;24, January 1996.

[19] H. V. Sorensen, D. L. Jones, M. T. Heideman, and C. S. Burrus. Real valued fast fourier transform algorithms. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 35(6):8498211;863, June 1987.

[20] S. Winograd. On computing the discrete fourier transform. *Proc. National Academy of Sciences, USA*, 73:10068211;1006, April 1976.

[21] S. Winograd. On computing the discrete fourier transform. *Mathematics of Computation*, 32:1758211;199, January 1978.

[22] S. Winograd. On the multiplicative complexity of the discrete fourier transform. *Advances in Mathematics*, 32(2):838211;117, May 1979. also in.

[23] S. Winograd. *Arithmetic Complexity of Computation*. SIAM CBMS-NSF Series, No. 33. SIAM, Philadelphia, 1980.

[24] S. Winograd. Signal processing and complexity of computation. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, page 13818211;1384, ICASSP-80, Denver, April 1980.