Real-Valued Fast Fourier Transform Algorithms

HENRIK V. SORENSEN, STUDENT MEMBER, IEEE, DOUGLAS L. JONES, STUDENT MEMBER, IEEE, MICHAEL T. HEIDEMAN, STUDENT MEMBER, IEEE, AND C. SIDNEY BURRUS, FELLOW, IEEE

Abstract—**This tutorial paper describes the methods for constructing fast algorithms for the computation of the discrete Fourier transform (DFT) of a real-valued series. The application of these ideas to all the major fast Fourier transform (FFT) algorithms is discussed, and the various algorithms are compared. We present a new implementation of the real-valued split-radix FFT, an algorithm that uses fewer operations than any other real-valued power-of-2-length FFT. We also compare the performance of inherently real-valued transform algorithms such as the fast Hartley transform (FHT) and the fast cosine transform (FCT) to real-valued FFT algorithms for the computation of power spectra and cyclic convolutions. Comparisons of these techniques reveal that the alternative techniques always require more additions than a method based on a real-valued FFT algorithm and result in computer code of equal or greater length and complexity.**

I. Introduction

ONE of the most important tools in modern digital signal processing applications is the fast Fourier transform (FFT). The FFT efficiently computes the discrete Fourier transform (DFT), a mapping of a length-N complex sequence to its length-N complex spectrum. Although most FFT algorithms are designed to compute the DFT of a complex sequence, in many applications the sequence to be transformed is real-valued. It is widely known that a length-$(N/2)$ complex FFT algorithm can be used to compute the DFT of a length-N real-valued sequence. It is less well known that algorithms of greater efficiency can be developed by exploiting symmetries within the complex-valued algorithms. Several such modified algorithms have been published, but a general discussion of these techniques is not available in the literature. For this reason, less efficient algorithms are generally used.

This paper reviews the general methods for constructing fast algorithms for the computation of the DFT of a real-valued series (RFFT). These methods are applied to all major FFT algorithms, and the resulting algorithms are compared. Inherently real-valued transforms, such as the discrete Hartley transform (DHT), can be used instead of the DFT for power spectrum computation or for fast computation of cyclic convolutions. We compare the performance of inherently real-valued transform techniques and RFFT algorithms for the computation of power spectra and cyclic convolutions. Section II of the paper describes the traditional methods, in which the real-valued DFT is computed using a complex-valued FFT. The third section shows in detail how a complex-valued radix-2 FFT can be converted into an RFFT. The same principles are applied in the subsequent sections in describing the higher-radix RFFT's, the split-radix RFFT, the prime factor RFFT, and the Winograd RFFT. Finally, the various algorithms are compared to each other and also to inherently real-valued transforms like the Hartley and cosine/sine transform.

Manuscript received March 26, 1986; revised October 20, 1986. This work was supported in part by the NSF under Grant ECS 83-14006, in part by an NSF graduate fellowship, and in part by NASA under Grant NGT 44-006-804.

H. V. Sorensen, D. L. Jones, and C. S. Burrus are with the Department of Electrical and Computer Engineering, Rice University, Houston, TX 77251.

M. T. Heideman was with the Department of Electrical and Computer Engineering, Rice University, Houston, TX. He is now with the Digital Image Processing Lab, Lockheed Missiles and Space Company, Palo Alto, CA 94304.

IEEE Log Number 8714125.

II. Real-Valued FFT's Using Complex-Valued FFT's

The DFT of a sequence $x(n)$ is defined to be

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk} \qquad k = 0, 1, \cdots, N-1 \quad (1)$$

where $W_N = e^{-j(2\pi/N)}$. In this definition, both $x(n)$ and $X(k)$ are assumed to be complex. However, if $x(n)$ is real, then half of the inputs to the algorithm (all the imaginary parts) are zero. Simply removing those parts of the CFFT algorithm that originate in the imaginary part of the input does not significantly reduce the complexity or storage requirements of an RFFT algorithm. This is apparent from the structure of FFT algorithms, as shown in Fig. 1. Assuming the input to a length-N ($=2^M$) FFT is real, there are fewer real (marked X) and more complex (marked O) variables after each stage of the algorithm. However, certain symmetries exist [10], [11] that are helpful in developing a real-valued FFT.

$$x(n) \text{ is real if and only if } X(k) = X^*(-k) = X^*(N-k) \quad (2)$$

$$X(k) \text{ is real if and only if } x(n) = x^*(-n) = x^*(N-n) \quad (3)$$

$x(n)$ is real and even if and only if $X(k)$ is real and even (4)

$x(n)$ is real and odd if and only if $X(k)$ is purely imaginary and odd. (5)

There are three fundamentally different ways to use an FFT algorithm for complex-valued data to compute the DFT of a real-valued sequence. The simplest approach is

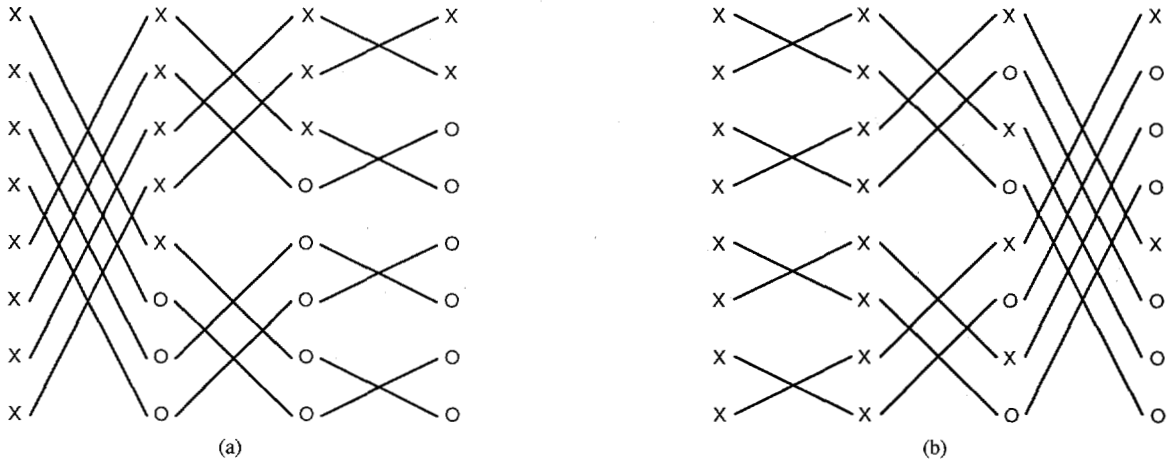


Fig. 1. (a) Propagation of realness in DIF FFT. (b) Propagation of realness in DIT FFT. The symbol X indicates a real value, and O indicates a complex value.

TABLE I
COMPLEXITY OF CFFT'S [36]

		Radix 2	Split Radix
3 multiplications and 3 additions per complex multiplications	Multiplications	$(3/2)MN - 5N + 8$	$MN - 3N + 4$
	Additions	$(7/2)MN - 5N + 8$	$3MN - 3N + 4$
4 multiplications and 2 additions per complex multiplications	Multiplications	$2MN - 7N + 12$	$(4/3)MN - (38/9)N + 6 + (2/9)(-1)^M$
	Additions	$3MN - 3N + 4$	$(8/3)MN - (16/9)N + 2 - (2/9)(-1)^M$

to expand the sequence with a zero imaginary part to obtain a complex sequence and then apply the CFFT, but this is also the most expensive approach in terms of operational complexity. Exactly the same number of operations and the same amount of storage are required as for the DFT of a complex-valued sequence of the same length.

The number of operations required to compute two different CFFT algorithms is shown in Table I. Note that the complexity depends on whether the complex multiplications are done with three real multiplications and three real additions (a 3/3 algorithm) (see [8]) or with four real multiplications and two real additions (a 4/2 algorithm).

If the CFFT is done, using a split-radix FFT with 3 butterflies (i.e., all trivial operations have been removed by introducing 2 extra butterflies) [36] and a 3/3 algorithm, the corresponding RFFT also requires $MN - 3N + 4$ multiplications, $3MN - 3N + 4$ additions, and $2N$ storage elements.

A second technique uses the symmetries of the DFT to transform two real-valued sequences simultaneously by computing one CFFT. This method has been discussed extensively in the literature [5], [11], [32]. If a sequence $x(n)$ is real, then its transform $X(k)$ has an even real part and an odd imaginary part. The DFT is a linear transform, so the DFT of $z(n) = x(n) + jy(n)$ is

$$\begin{aligned}\text{DFT}\left[z(n)\right] &= Z(k) = Z_r(k) + jZ_i(k)\\ &= \text{DFT}\left[x(n) + jy(n)\right]\\ &= \left\{X_r(k) - Y_i(k)\right\} + j\left\{X_i(k) + Y_r(k)\right\}\end{aligned} \tag{6}$$

where subscripts r and i denote real and imaginary parts, respectively. Since

$$\begin{aligned}Z(N-k) = &\left\{X_r(k) + Y_i(k)\right\}\\ &- j\left\{X_i(k) - Y_r(k)\right\},\end{aligned} \tag{7}$$

it can be found that

$$\begin{aligned}\text{DFT}\left[x(n)\right] &= X_r(k) + jX_i(k)\\ &= \frac{1}{2}\left\{Z_r(k) + Z_r(N-k)\right\}\\ &\quad + j\frac{1}{2}\left\{Z_i(k) - Z_i(N-k)\right\}\\ &\qquad k = 0, 1, \cdots, \frac{N}{2}\end{aligned}$$

$$\begin{aligned}\text{DFT}\left[y(n)\right] &= Y_r(k) + jY_i(k)\\ &= \frac{1}{2}\left\{Z_i(N-k) + Z_i(k)\right\}\end{aligned}$$

$$+ j\frac{1}{2}\{Z_r(N-k) - Z_r(k)\}$$

$$k = 0, 1, \cdots, \frac{N}{2}. \quad (8)$$

Note that because $x(n)$ is real, $X(k)$ has complex conjugate (or Hermitian) symmetry, so it is only necessary to compute $X(k)$ for $0 \le k \le (N/2)$. It is also apparent that for $k = 0$ and $k = (N/2)$, (8) can be simplified since $k \equiv N - k \pmod N$, yielding $X_r(k) = Z_r(k)$, $Y_r(k) = Z_i(k)$, and $X_i(k) = Y_i(k) = 0$ when $k = 0$ or $k = (N/2)$. Thus, the cost for two length-N RFFT's is one length-N CFFT and $2N - 4$ extra additions to separate the two real-valued transforms. If two transforms are needed, as would be the case when convolving two arbitrary sequences or doing block processing, the operation count per real-valued transform is half of that described above, and thus one transform requires $(1/2)\,MN - (3/2)\,N + 2$ multiplications and $(3/2)\,MN - (1/2)\,N$ additions; again assuming the CFFT is done by a split-radix 3/3 FFT algorithm.

The third technique for computing the DFT of a real-valued sequence using a CFFT is useful when only one real-valued transform is needed. This technique follows from the observation that the final stage of a decimation-in-time FFT algorithm combines two independent transforms of length $(N/2)$ to compute a length-N transform [11].

$$X(k) = \sum_{n=0}^{N-1} x(n)\,W_N^{nk}$$

$$= \sum_{n=0}^{N/2-1} x(2n)\,W_{N/2}^{nk} + W_N^k \sum_{n=0}^{N/2-1} x(2n+1)\,W_{N/2}^{nk}$$

$$k = 0, 1, \cdots, N-1. \quad (9)$$

The DFT now consists of two parts—a DFT of the even-indexed inputs and a DFT of the odd-indexed inputs that are combined to produce the full-length DFT. If the data are real-valued, the two half-length DFT's are also transforms of real-valued data, and they can be computed as described in (8) by combining the two sequences into a length-$(N/2)$ complex sequence $z(n) = x(2n) + jx(2n+1)$. This sequence is transformed with a half-length CFFT, and the two half-length DFT's are separated as shown in (8) and recombined according to (9) to produce the full-length DFT as the real-valued sequence. The separation of the two half-length transforms and the computation of the last stage of the DFT requires $N - 6$ multiplications and $(5/2)\,N - 6$ additions (assuming 4 real multiplications and 2 real additions per complex multiplication), so if the half-length CFFT is done using a radix-2 4/2 algorithm, the total number of operations is $MN - (7/2)\,N - 6$ multiplications and $(3/2)\,MN - (N/2) - 2$ additions. Since the last stage is done separately as a radix-2 stage, it is not possible to compute the total algorithm as a split-radix or radix-4 algorithm even though the two half-length transforms can be computed using these techniques. This slightly reduces the efficiency of these techniques relative to full-length algorithms. For the split-radix 3/3 algorithm, the complexity comes out to be $(1/2)\,MN - (5/4)\,N$ multiplications and $(3/2)\,MN - (1/4)\,N - 4$ additions when it is further assumed that the last stage is also implemented using a 3/3 complex multiplication algorithm.

III. The Radix-2 Decimation-in-Time RFFT

The techniques applied in the development of FFT algorithms specialized for real-valued series (instead of using CFFT's) are essentially the same for all of the well-known algorithms. The fundamental idea, as described by Bergland [1] and Sande [31], is that symmetries due to the real-valued nature of the data can be used at every stage of the FFT algorithm to remove redundant operations. In this section, the application of this idea to the simplest and most widely known FFT algorithm, the radix-2 decimation-in-time FFT, is thoroughly discussed.

In the standard derivation of the radix-2 decimation-in-time FFT algorithm [25], a length-$N(=2^M)$ transform is rewritten as the sum of two summations of $(N/2)$ elements each as in (9). The first sum is a length-$(N/2)$ DFT of the even-indexed samples $x(0), x(2), \cdots, x(N-2)$, and the second sum is a length-$(N/2)$ DFT of the odd-indexed samples $x(1), x(3), \cdots, x(N-1)$ multiplied by a term W_N^k called a twiddle factor. If the time series $x(n)$ is real-valued, the real part of its Fourier transform is even symmetric, and the imaginary part is odd symmetric. Thus, the Fourier transform coefficients satisfy

$$X(0) \text{ and } X\left(\frac{N}{2}\right) \text{ are real,} \quad (10)$$

$$X(k) = X^*(N-k), \qquad 1 \le k \le \frac{N}{2} - 1. \quad (11)$$

These symmetries allow us to reduce the memory and computational requirements by about a factor of 2 since the symmetries imply that the coefficients $X(k)$, $(N/2) + 1 \le k \le N - 1$, and the imaginary parts of $X(0)$ and $X(N/2)$ need not be computed or stored. These symmetries are illustrated in Fig. 2 for a length-16 RDFT. Real-valued terms are denoted by an X, and complex numbers are denoted by an O. Dashed arcs connect the numbers that are complex conjugates.

The reduction in computation is achieved by noting that only half of the complex values need be computed since the others are then known by the complex conjugate symmetry. One possibility is to compute $X(0)$ through $X(N/2)$, but a better choice is to compute $X(0)$ through $X(N/4)$ and $X(N/2)$ through $X(3N/4)$. The latter choice is preferred because these are the values computed by the first $(N/4) + 1$ complex butterflies in the standard complex-valued algorithm. Thus, we construct a real-valued algorithm simply by skipping the last $(N/4) - 1$ butterflies since we already know their output. The but-

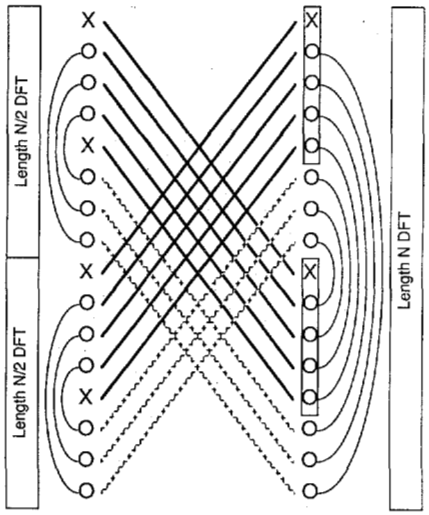


Fig. 2. Last stage of a length-16 DFT. The symbols X and O indicate real and complex values, respectively. The arcs indicate values that are complex conjugates of each other.

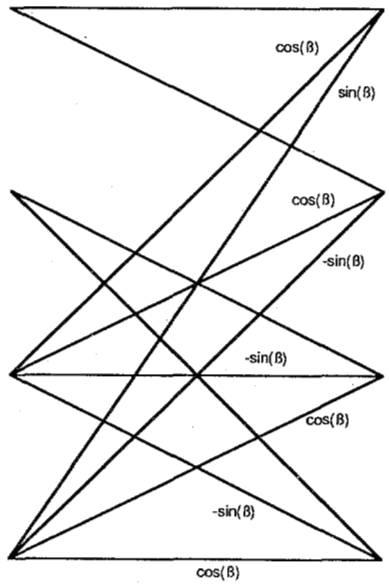


Fig. 3. Real-valued butterfly; β is the rotation angle.

terflies and the outputs that are actually computed are boxed in Fig. 2, and the others are bypassed.

To compute the RFFT in place, the redundancies must be utilized to halve the storage requirement compared to the corresponding CFFT. This is possible if the real part of the kth complex-valued coefficient is placed in the kth location in the real array, and the imaginary part is stored in the redundant $(N - k)$th location. Since the imaginary part of the kth complex output $(1 \leq k \leq (N/2) - 1)$ is stored in the $(N - k)$th memory location, the value in that location from the previous stage must be used before it is overwritten. This is not a problem since the current content of this memory location is needed only by the current butterfly. The butterfly operations for the RFFT and the CFFT are identical; only the memory locations from which the data are fetched are different. The resulting butterfly is shown in Fig. 3. It closely resembles the butterfly for the FHT presented in [34].

A special butterfly in the final stage computes $X(0)$ and $X(N/2)$ separately because they are real, and since it requires no multiplications, this is also reduces the computational complexity. A further savings of four additions and a complex multiplication in the last stage is achieved by noting that the butterfly computing $X(N/4)$ and $X(3N/4)$ requires no multiplications or additions (only a sign change) for real-valued data. These four values are computed separately in a "special butterfly." This scheme exactly halves the number of multiplications and storage locations and reduces the additions to two fewer than half that required to compute one stage of the complex-valued transform.

This approach can be applied at every stage by recognizing that the half-length transforms of the even- and odd-indexed data samples are again DFT's of real-valued data, so the same approach can be recursively applied to shorter DFT's to complete the algorithm. The result is a radix-2 decimation-in-time FFT algorithm for real-valued data that requires exactly half the multiplications and storage, and $N - 2$ fewer than half the additions of the algorithm for complex data; hence, the complexity is $(3/4)\,MN - (5/2)\,N + 4$ multiplications and $(7/4)\,MN - (7/2)\,N + 6$ additions using a 3/3 complex multiplication algorithm. Fig. 4 illustrates the algorithm for a length-16 transform. At every stage, the values that are real are indicated by an X, and the complex values are indicated by an O. Locations that are complex conjugate of another location (and where the imaginary part of the complex number is stored) are connected by a dashed arc. Only the butterflies that are actually computed are shown. A program implementing this approach is given in the Appendix.

It is important to note that the decimation-in-time algorithm is natural for the fast calculation of the spectrum of a real-valued series because the shorter sequences are also real-valued, and the symmetry of their transforms can be exploited at every stage to reduce operations and storage. Although a decimation-in-frequency algorithm (for complex inputs) at the end yields a spectrum with the proper symmetry, the redundancies are not apparent within the algorithm and cannot easily be exploited to reduce the operation count and storage from that required for complex data. As can be seen in the length-8 decimation-in-frequency algorithm represented in Fig. 1(a), after the first stage there are five real-valued and three complex-valued numbers, for a total of eleven real numbers to be stored.

For the inverse discrete Fourier transform (IDFT) that transforms the complex conjugate symmetric spectrum to a real-valued time series, the decimation-in-frequency algorithm is the natural choice. If the even-indexed and odd-indexed frequency samples are separated into two half-length series, symmetry of both series is maintained. At every stage of the decimation-in-frequency IDFT algorithm, the redundancies are explicit and easily removed. This corresponds to a flow graph reversal [29] of the de-

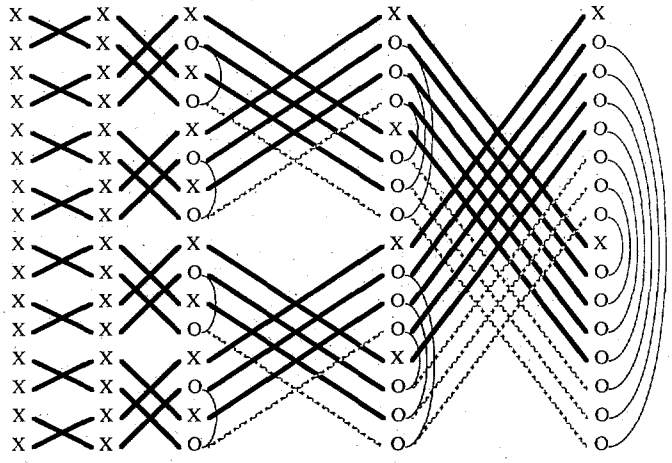


Fig. 4. A length-16 radix-2 RFFT. The symbol X indicates a real value, and O indicates a complex value. Arcs show complex values that are conjugates of each other. Only the butterflies drawn in solid lines need be computed.

cimation-in-time algorithm for real-valued series just described. Decimation-in-time algorithms are appropriate for computing the DFT or IDFT of a real-valued series, and decimation-in-frequency algorithms are applicable for computing the DFT or IDFT of a complex conjugate symmetric series.

The Appendix contains a simple program for computation of the DFT of a real-valued series. The program implements the techniques described above and is purposely simple for maximum clarity. An inverse RFFT program can easily be derived by modifying a decimation-in-frequency inverse CFFT program in a similar way. Special techniques for saving operations that have been applied to the FFT of a complex series can similarly be applied to these programs. For example, the complex multiplications can be implemented with three real multiplications and three additions, and sine/cosine lookup tables can be added. These and other features can easily be implemented by comparison to a corresponding FFT program such as those found in [8].

IV. Higher-Radix Algorithms

The most well-known efficient FFT algorithm for real-valued sequences is probably Bergland's radix-8 RFFT [2], [33]. Like the radix-4 and higher-radix algorithms, it is developed in the same manner as the radix-2 RFFT. The symmetry of the transform implies that only half of the butterflies need be computed and also results in simplication of the butterflies with simple coefficients. This reduction saves half of the multiplications and storage and slightly more than the half of the additions over the algorithm for complex data. As before, a decimation-in-time decomposition yields shorter-length transforms that are themselves transforms of real-valued data, and the same savings are obtained at every stage in the algorithm. The complete algorithms require exactly half the multiplications and storage and $N-2$ less than half of the additions required by the corresponding algorithm for complex data. Since the split-radix algorithm in the next section is both shorter and more efficient than the higher-radix algorithms, no programs are presented here. Several efficient programs based on this approach are available in the literature [33]. As before, higher-radix IDFT algorithms use a decimation-in-frequency approach in which the redundancies are explicit and easily removed.

V. Real-Valued Split-Radix FFT's

It has been shown [13], [38] that a DFT can be computed by breaking it up into a length $(N/2)$ DFT over the even-indexed part of the input sequence and two length-$(N/4)$ DFT's over the odd-indexed part of the input sequence. Iterating this scheme for the whole transform (all stages) leads to a very efficient decimation-in-frequency algorithm called the split-radix FFT algorithm, which requires fewer operations than a radix-2 FFT, a radix-4 FFT, or any higher-radix FFT. As described in [14], [36], this argument also holds for the decimation-in-time algorithm, and the decomposition is

$$X(k) = \sum_{n=0}^{N/2-1} x(2n)\, W_{N/2}^{nk} + W_N^k \sum_{n=0}^{N/4-1} x(4n+1)\, W_{N/4}^{nk} + W_N^{3k} \sum_{n=0}^{N/4-1} x(4n+3)\, W_{N/4}^{nk}$$

$$k = 0, 1, \cdots, N-1. \qquad (12)$$

A decimation-in-time split-radix algorithm has complex conjugate symmetry at each stage. By using this symmetry as described for the radix-2 and higher-radix algorithms, a real-valued split-radix algorithm [35] can be derived along the same lines as the complex-valued algorithm in [36]. The resulting program is shown in the Appendix. The complexity for this program is $(2/3)\,MN - (19/9)\,N + 3 + (1/9)(-1)^M$ multiplications and $(4/3)\,MN - (17/9)\,N + 3 - (1/9)(-1)^M$ additions if the complex multiplications are done using the 4/2 strategy. If instead a 3/3 strategy is used, the complexity is $(1/2)\,MN - (3/2)\,N + 2$ multiplications and $(3/2)\,MN - (5/2)\,N + 4$ additions. In both cases, the number of multiplications is exactly half that of the corresponding complex split-radix FFT, while the number of additions is $N-2$ fewer than half, as was the case for the radix-2 algorithm.

For the forward transform, the input is real and the output is complex conjugate symmetric, whereas for the inverse transform the input is complex conjugate symmetric and the output is real. Therefore, two algorithms are needed if the intended application needs both a forward and inverse transform. The decimation-in-time forward transform should be used with a decimation-in-frequency inverse transform if unscrambling is required. An interesting feature of the split-radix algorithm is that, while it is not possible to compute a real-valued radix-2 decimation-in-frequency FFT in place, the split-radix decimation-in-frequency algorithm has been computed in place [15], [21], [38]. By noting the symmetries within the al-

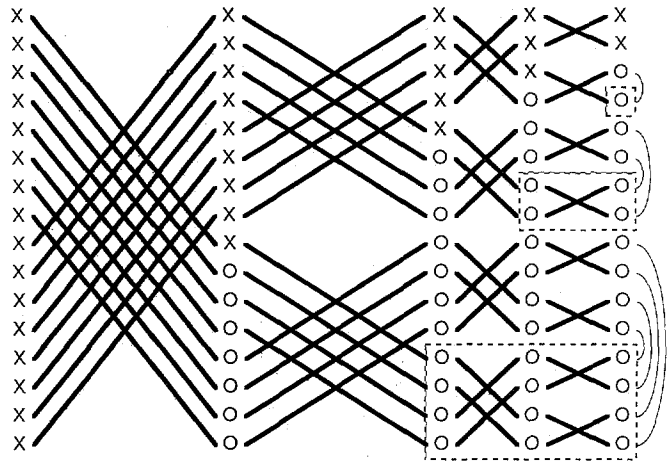


Fig. 5. Split-radix length-16 DIF CFFT of real-valued data. The symbol X indicates a real value, and O indicates a complex value. The arcs indicate complex values that are conjugates of each other. The boxed values are redundant and need not be computed.

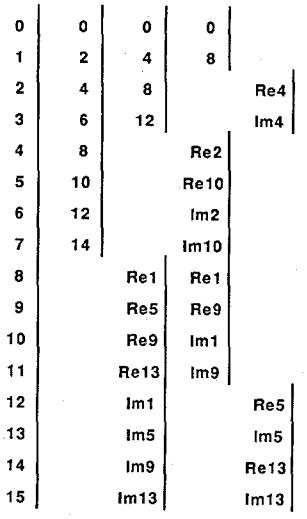


Fig. 6. Storage scheme for a length-16 DIF split-radix FFT from [14]. As the values become complex, there is an "empty" location for the imaginary part.

gorithm, it is apparent that the boxed branches in Fig. 5 need not be computed. The even-indexed part remains real and can thus be stored in place. The odd-indexed part becomes complex, but because of the complex conjugate symmetry on the output, only half of these complex values need be stored. Fig. 5 (from [15]) shows a way of storing the values of a real-valued decimation-in-frequency split-radix FFT. A drawback of this method is that the transform produces the output in a strange order, requiring a complicated unscrambler. But if the unscrambling is not needed, the decimation-in-frequency split-radix RFFT is a good choice when coupled with a decimation-in-time inverse transform.

VI. Prime Factor Algorithms

If the transform length $N = KL$ is a product of relatively prime factors, the one-dimensional DFT can be mapped to a two-dimensional DFT for appropriate choices of A, B, C, and D [6]:

$$X(k) = X(Ak_1 + Bk_2) = X(k_1, k_2)$$
$$= \sum_{n_2=0}^{K-1} \left[\sum_{n_1=0}^{L-1} x(n_1, n_2)\, e^{-j2\pi n_1 k_1/L} \right] \cdot e^{-j2\pi n_2 k_2/K} \qquad 0 \le k_1 \le L-1,\quad 0 \le k_2 \le K-1 \tag{13}$$

where $x(n) = x(Cn_1 + Dn_2) = x(n_1, n_2)$ and K and L are the lengths of the "short transforms" over n_1 and n_2, respectively. This expression is merely a DFT of each of the columns of $x(n_1, n_2)$ (length-L DFT's) followed by a transform of each of the rows (length-K DFT's) as described in [8]. This decomposition can be recursively applied to factors of K and L to generalize the prime factor algorithm (PFA) to higher dimensions. The most efficient programs implementing the PFA rely on a set of extremely efficient "DFT modules" for small prime (or power of prime) lengths to compute the transform in each dimension [7], [18].

The redundancies due to real-valued data in the PFA can be removed as follows. The first set of short (length-L) transforms (over the columns) are of real-valued data, so half of the multiplications and $L - 1$ ($L - 2$ if L is even) more than half of the additions in the short-length DFT's can be saved by using specialized DFT modules for real-valued data. When computing the second dimension, the row corresponding to the zero frequency index of the first stage of transforms is real-valued (as is the $(L/2)$th row if L is even) and requires half the multiplications and $K - 1$ ($K - 2$ if K is even) fewer than half the additions. A second-stage real-valued module computes the DFT of this (or these) row(s). The other $L - 1$ or $L - 2$ rows are complex, but since the input data are real-valued, the $(L - k_1)$th row is the complex conjugate of the k_1th row after the DFT over the first dimension. Only half of the length-K complex transforms need be computed or stored since the DFT of the complex conjugate of a sequence is the time-reversed complex conjugate of the DFT of the sequence. The real part can be stored in the k_1th row, and the imaginary part can be stored in the $(L - k_1)$th row. If the imaginary parts of the output are stored in reverse $(K - k_2)$ order, the output order of the transform is given by the mapping in (13) where the real part is in $X(k_1, k_2)$ and the corresponding imaginary part is in $X(L - k_1, K - k_2)$. Hence, using the symmetries of the complex PFA's reduces the storage and multiplication to half and the additions to $N - 1$ ($N - 2$ if N is even) fewer than half that of a complex-valued PFA [17].

The scheme described above requires both a real-valued and a complex-valued short DFT module for each stage. As an alternative, the complex transforms can be computed by applying a real-valued module to the real and to the imaginary parts of the data and adding the results appropriately to get a complex transform. This requires ex-

actly the same number of operations as a complex module. Another alternative is to commute the combining additions to the end of the algorithm [17]. The floating-point operation counts are the same, but recalculation of the indexes at the final combination stage makes this technique somewhat slower [17].

The inverse PFA takes a complex conjugate spectrum and computes the real-valued sequence. The indexing and storage locations are exactly the same as in the forward real-valued transform, except that the order of operations is reversed. In essence, the real-valued PFA is run backwards. Two types of modules are required for each short length: an inverse (complex-to-complex) module and a complex-conjugate-symmetric-to-real inverse module.

Another recent scheme [19] uses a prime factor map, but computes the modules using the Hartley transform [3], [34]. This algorithm is much less efficient than using the real-valued DFT to compute the modules because it requires many more additions [34].

VII. Real-Valued Winograd Fourier Transform Algorithms

One way of calculating the short-length DFT's used by the PFA is with Winograd DFT modules [24], [39]. These modules typically use the minimum number of multiplications required for prime length transforms. The modules were originally derived for real-valued inputs, and for complex data the real-valued transforms of the real and imaginary parts are combined in a final stage. Real-valued DFT modules can therefore be directly derived, or the computation on the imaginary input can simply be deleted from a complex-valued module, saving half the multiplications and slightly more than half the additions. As shown in [39], the short DFT modules can be nested to give the Winograd Fourier transform algorithm (WFTA). After nesting, the operations associated with the real and imaginary parts of the input remain separate until a final combination stage, so real-valued algorithms can again be constructed directly or simply exercised from a complex-valued program. Parsons develops these ideas in more detail in [26]. Programs are easily obtained by modifying the short modules in [8] or the general program in [23]. Operation counts are given in [8].

VIII. Comparison

Table II compares the different algorithms for computing the real-valued FFT described in this paper. It shows the minimum counts for each case and assumes that the complex multiplications (for the length-2^M algorithms) are done using three real multiplications and three real additions. The operation counts for the three methods for computing the RFFT using a CFFT are based on the CFFT being an optimum split-radix CFFT. Also for comparison purposes, the complexity of Bergland's radix-4 (FAST) and radix-8 programs (FFA) from [33] are supplied (the numbers are changed to conform with the 3/3 complex multiplication algorithm). The PFA counts are from [17], and the WFTA counts are derived from the complex WFTA counts in [8]. The lowest counts for a power-of-2 algorithm are obtained by the split-radix FFT. They were first achieved by Yavne [40], but because his derivation was very tedious, it went rather unnoticed. In 1984, three papers appeared almost simultaneously [13], [22], [38], all describing the split-radix algorithm. A subsequent paper [36] demonstrates the existence of both a DIF and a DIT algorithm and provides short programs that implement the split-radix algorithm. Another recently published algorithm for real-valued data is the VFFT by Preuss [28]. His algorithm has the same multiplicative complexity as the split-radix, but requires more additions. Since the VFFT requires excessive processor time for indexing and data swapping, it is not included in the table.

All the algorithms except the "direct CFFT" are computed in place and hence require only N storage locations (direct CFFT requires $2N$). The table reflects the same relative relationships in terms of operation counts that are familiar from CFFT's. In terms of indexing (overhead), the power-of-2 algorithms require only slightly more than half that of the corresponding CFFT, while the PFA and WFTA require more than that. The PFA indexing becomes complicated, and since the program also gets long (both real and complex modules needed) the PFA is normally not a good choice unless an intermediate length between powers of 2 is needed or the RFFT is coded in-line (to reduce indexing). The WFTA has the lowest multiplication count, but achieves this by increasing the additions and data storage, and since the indexing is also rather complicated, the WFTA is rarely used.

The programs in the Appendix have not been optimized in terms of run-time, but rather to obtain maximum readability of the code. These programs are derived from the complex programs in [8] and are similar to the Hartley programs in [34]. The split-radix RFFT in the Appendix is probably slower than Bergland's FFA, but with proper optimization, the split-radix should be faster on most machines.

If the RFFT is used to calculate a cyclic convolution, inherently real-valued transforms exist that could be used instead. The Hartley transform [3], [4] is shown in [34] to have properties similar to those of the DFT. It has been demonstrated that the Hartley transform and the RFFT are very strongly related and that for every RFFT algorithm there exists an equivalent Hartley algorithm, and vice versa [34]. The Hartley transform algorithms require about N more additions than the corresponding RFFT algorithms, and a convolution takes about $3N$ more additions using the Hartley transform. The resulting programs are nearly the same, due to the fundamental similarity between the Hartley and RFFT algorithms, so the indexing and other overhead costs are essentially the same [34]. The only advantage of the Hartley transform is that it is its own inverse, which means that a separate inverse program is not needed. The Hartley transform may therefore be useful on limited memory machines.

It has been shown [16] that a split-radix hybrid Hartley/

TABLE II
OPERATION COUNTS FOR DIFFERENT ALGORITHMS. IT IS ASSUMED THAT THE COMPLEX MULTIPLICATIONS ARE DONE USING THREE MULTIPLICATIONS AND THREE ADDITIONS. THE PROGRAMS "FAST" AND "FFA" HAVE BEEN CHANGED ACCORDINGLY

Multiplicative/Additive Complexity for Computing Real-Valued FFT's

	CFFT Direct		CFFT Double		CFFT Packing		Radix-2 RFFT		FAST [33] Radix-4		FFA [33] Radix-8		Real Split-Radix		Real PFA		Real WFTA	
Length	Multiplications	Additions	Multiplications	Additions	Multiplications	Additions	Multiplications	Additions	Multiplications	Additions	Multiplications	Additions	Multiplications	Additions	Multiplications	Additions	Multiplications	Additions
8	4	52	2	32	2	30	2	20	2	20	2	20	2	20	2	20	2	20
12															8	38	8	38
15															25	67	17	67
16	20	148	10	88	12	88	12	62	11	62	12	62	10	60	10	60	10	60
30															50	164	34	164
32	68	388	34	224	40	228	44	174	37	170	37	167	34	164				
35															75	265	53	299
60															100	386	68	386
64	196	964	98	544	112	556	132	454	109	442	103	425	98	420				
72															98	500	82	508
112															198	984	154	1056
120															230	920	138	920
128	516	2308	258	1280	288	1308	356	1126	285	1082	283	1053	258	1028				
140															300	1338	212	1474
252															568	2726	392	3042
256	1284	5380	642	2944	704	3004	900	2694	717	2586	683	2477	642	2436				
280															670	3024	426	3296
504															1262	6080	786	6712
512	3076	12 292	1538	6656	1664	6780	2180	6278	1709	5978	1611	5709	1538	5636				
560															1550	6816	964	8026
1008															2902	13 544	1774	17 208
1024	7172	27 652	3586	14 848	3840	15 100	5124	14 342	4013	13 658	3851	13 069	3586	12 804				

RFFT algorithm can compute the Hartley transform in the same number of multiplications and two more additions than it takes to compute a DFT using a split-radix RFFT, and takes four more additions to compute a cyclic convolution using the hybrid Hartley/RFFT transform than when using the RFFT. However, the hybrid Hartley/RFFT requires both Hartley and RFFT code segments, so the program is about twice the size of an RFFT program, or an IF statement must be introduced in the innermost loop.

Another alternative for fast convolution is use of the cosine/sine transform. A split-radix cosine transform has been developed [20], [38], but it is less efficient than an RFFT due to the relationship between the DFT and DCT [15], [38]. As an example, 80 multiplications and 209 additions are required to compute a length-32 cosine transform, while only 34 multiplications and 164 additions are needed to compute a length-32 RFFT. Furthermore, the cosine transform does not have as simple a convolution property [9] as the DFT and Hartley transform, so it should only be used in cases, such as certain image processing applications, in which the properties of the cosine transform are preferred.

More specialized RFFT algorithms have been published. Martens [21] describes an algorithm for length 2^s3^t and achieves a complexity comparable to a radix-2 (exactly equal if the length equals a power 2). Radix-3, -6, and -12 [12], [27], [37] algorithms have also been presented, but their complexity is greater than, and the code is not as regular and compact as, a radix-4 FFT, so they will be of interest only in special cases.

Further savings in the computations can be obtained if the real-valued input sequence is also symmetric. The algorithms presented in the literature [30], [41] are not computed in place. Preuss [28] has a version that computes the real-valued symmetric FFT in place, but again, the overhead is excessive. Neither the decimation-in-time nor the decimation-in-frequency algorithm is well suited for removing all the redundancies, and they both result in a complicated unscrambler. However, if it is not necessary to unscramble, the real-valued split-radix decimation-in-frequency program [15] should be used.

IX. CONCLUSION

Although most FFT algorithms are developed for complex-valued sequences, redundancies and symmetries in all of these algorithms can be exploited to reduce the multiplications and storage by exactly a factor of 2 and the addition by slightly more than a factor of 2. The two commonly used techniques of using complex FFT's, the "doubling" and the "packing" techniques, take several N more additions than a specialized algorithm for real input data.

Inherently real-valued transforms such as the Hartley transform have been suggested as alternatives to the DFT in applications such as power spectrum computation and fast convolution. The performance of these alternatives is often compared to a nonoptimal or less efficient CFFT-based implementation. Careful comparisons of these techniques reveal that they always require more operations than a method based on an efficient real-valued FFT algorithm. Furthermore, these other algorithms require as

long or often longer computer code than the real-valued FFT algorithms discussed here. They also never achieve less data shuffling and indexing than the real-valued FFT. Direct implementations of real-valued FFT algorithms require fewer operations and shorter, simpler computer code than other techniques, and they should be used.

APPENDIX

```
C-----------------------------------------------------------------------------C
C                                                                             C
C     Real-valued, in-place, Cooley-Tukey radix-2 FFT program                 C
C     Real input and output data in array X                                   C
C     Length is N = 2 ** M                                                    C
C     Decimation-in-time, cos/sin in innermost loop                           C
C     Output in order:                                                        C
C          [Re(0), Re(1), ..., Re(N/2), Im(N/2-1), ..., Im(1)]                C
C                                                                             C
C     H. V. Sorensen,  Rice University,  Jan. 1985                            C
C                                                                             C
C-----------------------------------------------------------------------------C
      SUBROUTINE RVFFT (X,N,M)
      REAL X(1)
C -------Digit reverse counter-------------------------------------------------C
100   J = 1
      N1 = N - 1
      DO  104 I = 1,N1
          IF (I.GE.J) GOTO 101
          XT   = X(J)
          X(J) = X(I)
          X(I) = XT
101       K = N/2
102       IF   (K.GE.J) GOTO 103
              J = J - K
              K = K/2
              GOTO 102
103       J = J + K
104   CONTINUE
C -------Length two butterflies------------------------------------------------C
      DO  60 I = 1,N, 2
          XT     = X(I)
          X(I)   = XT + X(I+1)
          X(I+1) = XT - X(I+1)
60    CONTINUE
C -------Other butterflies-----------------------------------------------------C
      N2 = 1
      DO  10 K = 2, M
          N4 = N2
          N2 = 2*N4
          N1 = 2*N2
          E  = 6.283185307179586/N1
          DO  20 I = 1, N, N1
              XT       = X(I)
              X(I)     = XT + X(I+N2)
              X(I+N2)  = XT - X(I+N2)
              X(I+N4+N2) = -X(I+N4+N2)
              A = E
C Note that in the first run through, N4=1, so the next statement
C is 'DO 30 J=1,0', which might cause problems on some compilers
              DO  30 J=1,N4 - 1
                  I1 = I + J
```

```
                    I2  = I - J + N2
                    I3  = I + J + N2
                    I4  = I - J + N1
                    CC = COS (A)
                    SS = SIN (A)
                    A = A + E
                    T1 = X(I3) * CC+X(I4) * SS
                    T2 = X(I3) * SS-X(I4) * CC
                    X(I4) =  X(I2) - T2
                    X(I3) = -X(I2) - T2
                    X(I2) =  X(I1) - T1
                    X(I1) =  X(I1) + T1
30              CONTINUE
20          CONTINUE
10      CONTINUE
        RETURN
        END

C-------------------------------------------------------------------------------------------------------------------------------C
C                                                                                                                               C
C       A real-valued, in-place, split-radix FFT program                                                                        C
C       Real input and output data in array X                                                                                   C
C       Length is N=2**M                                                                                                        C
C       Decimation-in-time, cos/sin in second loop                                                                              C
C       Output in order:                                                                                                        C
C                [ Re(0), Re(1), · · · , Re(N/2), Im(N/2-1), · · · , Im(1) ]                                                    C
C                                                                                                                               C
C       H. V. Sorensen,  Rice University,  Oct. 1985                                                                            C
C                                                                                                                               C
C-------------------------------------------------------------------------------------------------------------------------------C
        SUBROUTINE RVFFT(X,N,M)
        REAL X(N)
C -------Digit reverse counter-------------------------------------------------------------------------------------------------C
100     J = 1
        N1 = N - 1
        DO  104 I = 1,N1
            IF (I.GE.J) GOTO 101
            XT  = X(J)
            X(J) = X(I)
            X(I) = XT
101         K = N/2
102         IF  (K.GE.J) GOTO 103
                J = J - K
                K = K/2
                GOTO 102
103         J = J + K
104     CONTINUE
C -------Length two butterflies------------------------------------------------------------------------------------------------C
        IS = 1
        ID = 4
70      DO  60 I0 = IS,N,ID
            I1   = I0 + 1
            R1   = X(I0)
            X(I0) = R1 + X(I1)
            X(I1) = R1 - X(I1)
60      CONTINUE
            IS = 2 * ID - 1
            ID = 4 * ID
```

```
      IF (IS.LT.N) GOTO 70
C -------L shaped butterflies ---------------------------------------------------------------- C
      N2 = 2
      DO  10 K = 2, M
          N2 = N2 * 2
          N4 = N2/4
          N8 = N2/8
          E  = 6.2831853071719586/N2
          IS = 0
          ID = N2 * 2
40        DO  38 I = IS, N - 1, ID
              I1 = I + 1
              I2 = I1 + N4
              I3 = I2 + N4
              I4 = I3 + N4
              T1 = X(I4) + X(I3)
              X(I4) = X(I4) - X(I3)
              X(I3) = X(I1) - T1
              X(I1) = X(I1) + T1
              IF (N4 .EQ. 1) GOTO 38
              I1 = I1 + N8
              I2 = I2 + N8
              I3 = I3 + N8
              I4 = I4 + N8
              T1  = (X(I3) + X(I4))/SQRT(2.0)
              T2  = (X(I3) - X(I4))/SQRT(2.0)
              X(I4) = X(I2) - T1
              X(I3) = -X(I2) - T1
              X(I2) = X(I1) - T2
              X(I1) = X(I1) + T2
38        CONTINUE
              IS = 2 * ID - N2
              ID = 4 * ID
          IF (IS.LT.N ) GOTO 40
          A = E
          DO  32 J = 2, N8
              A3 = 3 * A
              CC1 = COS (A)
              SS1 = SIN (A3)
              CC3 = COS (3)
              SS3 = SIN (A3)
              A = J * E
              IS = 0
              ID = 2 * N2
36            DO  30 I = IS, N - 1, ID
                  I1 = I + J
                  I2 = I1 + N4
                  I3 = I2 + N4
                  I4 = I3 + N4
                  I5 = I + N4 - J + 2
                  I6 = I5 + N4
                  I7 = I6 + N4
                  I8 = I7 + N4
                  T1 = X(I3) * CC1 + X(I7) * SS1
                  T2 = X(I7) * CC1 - X(I3) * SS1
                  T3 = X(I4) * CC3 + X(I8) * SS3
                  T4 = X(I8) * CC3 - X(I4) * SS3
                  T5 = T1 + T3
```

```
                T6 = T2 + T4
                T3 = T1 - T3
                T4 = T2 - T4
                T2    = X(I6) + T6
                X(I3) = T6    - X(I6)
                X(I8) = T2
                T2    = X(I2) - T3
                X(I7) = -X(I2) - T3
                X(I4) = T2
                T1    = X(I1) + T5
                X(I6) = X(I1) - T5
                X(I1) = T1
                T1    = X(I5) + T4
                X(I5) = X(I5) - T4
                X(I2) = T1
30          CONTINUE
                IS = 2 * ID - N2
                ID = 4 * ID
            IF (IS.LT.N) GOTO 36
32      CONTINUE
10  CONTINUE
    RETURN
    END
C-------------------------------------------------------------------------------------------------------------------------------C
C                                                                                                                               C
C   A real-valued, in-place, split-radix IFFT program                                                                           C
C   Hermitian symmetric input and real output in array X                                                                        C
C   Length is N=2**M                                                                                                            C
C   Decimation-in-frequency, cos/sin in second loop                                                                             C
C   Input order:                                                                                                                C
C            [ Re(0),Re(1), · · · , Re(N/2),Im(N/2-1), · · · , Im(1) ]                                                          C
C                                                                                                                               C
C   H.V. Sorensen,  Rice University,  Nov. 1985                                                                                 C
C                                                                                                                               C
C-------------------------------------------------------------------------------------------------------------------------------C
    SUBROUTINE IRVFFT(X,N,M)
    REAL X(N)
C -------L shaped butterflies ----------------------------------------------------------------------------------------------------C
    N2 = 2*N
    DO 10 K = 1, M-1
        IS = 0
        ID = N2
        N2 = N2/2
        N4 = N2/4
        N8 = N4/2
        E  = 6.283185307179586/N2
17      DO 15 I = IS,N-1,ID
            I1 = I + 1
            I2 = I1 + N4
            I3 = I2 + N4
            I4 = I3 + N4
            T1    = X(I1) - X(I3)
            X(I1) = X(I1) + X(I3)
            X(I2) = 2*X(I2)
            X(I3) = T1    - 2 * X(I4)
            X(I4) = T1    + 2 * X(I4)
            IF (N4 .EQ. 1) GOTO 15
            I1 = I1 + N8
```

```
                I2 = I2 + N8
                I3 = I3 + N8
                I4 = I4 + N8
                T1    = (X(I2) - X(I1))/SQRT(2.0)
                T2    = (X(I4) + X(I3))/SQRT(2.0)
                X(I1) = X(I1) + X(I2)
                X(I2) = X(I4) - X(I3)
                X(I3) = 2 * (-T2-T1)
                X(I4) = 2 * (-T2+T1)
15          CONTINUE
                IS = 2 * ID - N2
                ID = 4 * ID
            IF (IS .LT. N - 1) GOTO 17
            A = E
            DO 20 J = 2, N8
                A3 = 3 * A
                CC1 = COS (A)
                SS1 = SIN (A)
                CC3 = COS (A3)
                SS3 = SIN (A3)
                A = J * E
                IS = 0
                ID = 2 * N2
40              DO 30 I = IS, N - 1, ID
                    I1 = I + J
                    I2 = I1 + N4
                    I3 = I2 + N4
                    I4 = I3 + N4
                    I5 = I + N4 - J + 2
                    I6 = I5 + N4
                    I7 = I6 + N4
                    I8 = I7 + N4
                    T1    = X(I1) - X(I6)
                    X(I1) = X(I1) + X(I6)
                    T2    = X(I5) - X(I2)
                    X(I5) = X(I2) + X(I5)
                    T3    = X(I8) + X(I3)
                    X(I6) = X(I8) - X(I3)
                    T4    = X(I4) + X(I7)
                    X(I2) = X(I4) - X(I7)
                    T5 = T1 - T4
                    T1 = T1 + T4
                    T4 = T2 - T3
                    T2 = T2 + T3
                    X(I3) = T5 * CC1 + T4 * SS1
                    X(I7) = -T4 * CC1 + T5 * SS1
                    X(I4) = T1 * CC3 - T2 * SS3
                    X(I8) = T2 * CC3 + T1 * SS3
30              CONTINUE
                    IS = 2 * ID - N2
                    ID = 4 * ID
                IF (IS .LT. N - 1) GOTO 40
20          CONTINUE
10      CONTINUE
C -------Length two butterflies---------------------------------------------------------------------------------- C
        IS = 1
        ID = 4
70      DO 60 I0 = IS, N, ID
```

```
            I1    = I0 + 1
            R1    = X(I0)
            X(I0) = R1 + X(I1)
            X(I1) = R1 - X(I1)
 60     CONTINUE
            IS = 2 * ID - 1
            ID = 4 * ID
        IF (IS.LT.N) GOTO 70
C -------Digit reverse counter---------------------------------------------------------------------- C
100     J = 1
        N1 = N - 1
        DO  104 I = 1, N1
            IF (I.GE.J) GOTO 101
            XT    = X(J)
            X(J)  = X(I)
            X(I)  = XT
101         K = N/2
102         IF  (K.GE.J) GOTO 103
                J = J - K
                K = K/2
                GOTO 102
103         J = J + K
104     CONTINUE
        DO  99 I = 1, N
            X(I) = X(I)/N
99      CONTINUE
        RETURN
        END
```

ACKNOWLEDGMENT

The authors would like to thank the reviewers for their suggestions and for bringing [14] and [37] to their attention.

REFERENCES

[1] G. D. Bergland, "A fast Fourier transform algorithm for real-valued series," *Commun. ACM*, vol. 11, no. 10, pp. 703–710, Oct. 1968.

[2] ——, "A radix-eight fast Fourier transform subroutine for real-valued Series," *IEEE Trans. Audio Electroacoust.*, vol. AU-17, pp. 138–144, June 1969.

[3] R. N. Bracewell, "The fast Hartley transform," *Proc. IEEE*, vol. 72, pp. 1832–1835, Dec. 1983.

[4] ——, *The Hartley Transform.* New York: Oxford Univ. Press, 1986.

[5] E. O. Brigham, *The Fast Fourier Transform.* Englewood Cliffs, NJ: Prentice-Hall, 1974.

[6] C. S. Burrus, "Index mappings for multidimensional formulation of the DFT and convolution," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-25, pp. 239–242, June 1977.

[7] C. S. Burrus and P. W. Eschenbacher, "An in-place, in-order prime factor FFT algorithm," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-29, pp. 806–817, Aug. 1981.

[8] C. S. Burrus and T. W. Parks, *DFT/FFT and Convolution Algorithms.* New York: Wiley, 1985.

[9] W. H. Chen and S. C. Fralick, "Image enhancement using cosine transform filtering," in *Proc. Symp. Image Science Mathematics*, Monterey, CA, Nov. 10–12, 1976, pp. 186–192.

[10] J. W. Cooley, P. A. W. Lewis, and P. D. Welch, "The finite Fourier transform," *IEEE Trans. Audio Electroacoust.*, vol. AU-17, pp. 77–85, June 1969; reprinted in L. R. Rabiner and C. M. Rader, Eds., *Digital Signal Processing.* New York: IEEE Press, 1972, pp. 251–259.

[11] ——, "The fast Fourier transform algorithm: Programming considerations in the calculation of sine, cosine, and Laplace transforms," *J. Sound Vib.*, vol. 12, no. 3, pp. 315–337, July 1970; reprinted in L. R. Rabiner and C. M. Rader, Eds., *Digital Signal Processing.* New York: IEEE Press, 1972, pp. 271–293.

[12] E. Dubois and A. Venetsanopoulos, "A new algorithm for the radix-3 FFT," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-26, pp. 222–225, June 1978.

[13] P. Duhamel and H. Hollmann, "'Split radix' FFT algorithm," *Electron. Lett.*, vol. 20, no. 1, pp. 14–16, Jan. 5, 1984.

[14] P. Duhamel, "A 'split-radix' fast Fourier transform," *Ann. Télécommun.* (in French), vol. 40, no. 9–10, pp. 418–494, Sept.-Oct. 1985.

[15] ——, "Implementation of 'split radix' FFT algorithms for complex, real and real-symmetric data," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-34, pp. 285–295, Apr. 1986.

[16] P. Duhamel and M. Vetterli, "Cyclic convolution of real sequences: Hartley versus Fourier and new schemes," in *Proc. 1986 IEEE Int. Conf. Acoust., Speech, Signal Processing*, Tokyo, Japan, Apr. 8–11, 1986, pp. 6.5.1–6.5.4.

[17] M. T. Heideman, C. S. Burrus, and H. W. Johnson, "Prime factor FFT algorithms for real-valued series," in *Proc. 1984 IEEE Int. Conf. Acoust., Speech, Signal Processing*, San Diego, CA, Mar. 19–21, 1984, pp. 28A.7.1–28A.7.4.

[18] D. P. Kolba and T. W. Parks, "A prime factor FFT algorithm using high-speed convolution," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-25, pp. 281–294, Aug. 1977.

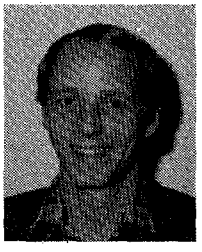
[19] R. Kumaresan and P. K. Gupta, "A prime factor FFT algorithm with real valued arithmetic," *Proc. IEEE*, vol. 73, pp. 1241–1243, July 1985.

[20] B. G. Lee, "A new algorithm to compute the discrete cosine transform," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-32, pp. 1243–1245, Dec. 1984.

[21] J. B. Martens, "Discrete Fourier transform algorithms for real valued sequences," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-32, pp. 390–396, Apr. 1984.

[22] ——, "Recursive cyclotomic factorization—A new algorithm for calculating the discrete Fourier transform," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-32, pp. 750–761, Aug. 1984.

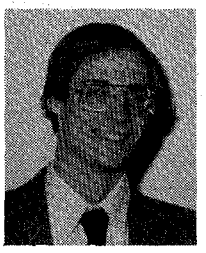
[23] J. H. McClellan and C. M. Rader, *Number Theory in Digital Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1979.
[24] H. J. Nussbaumer, *Fast Fourier Transform and Convolution Algorithms*. New York: Springer-Verlag, 1982.
[25] A. V. Oppenheim and R. W. Schafer, *Digital Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1975.
[26] T. W. Parsons, "A Winograd-Fourier transform algorithm for real-valued data," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-27, pp. 398–402, Aug. 1979.
[27] S. Prakash and V. V. Rao, "A new radix-6 FFT algorithm," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-29, pp. 939–941, Aug. 1981.
[28] R. D. Preuss, "Very fast computation of the radix-2 discrete Fourier transform," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-30, pp. 595–607, Aug. 1982.
[29] L. R. Rabiner and B. Gold, *Theory and Application of Digital Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1975.
[30] L. R. Rabiner, "On the use of symmetry in FFT computations," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-27, pp. 233–239, June 1979.
[31] G. Sande, "Fast Fourier transform—A globally complex algorithm with locally real implementations," in *Proc. 4th Annu. Princeton Conf. Inform. Sci. Syst.*, Princeton, NJ, Mar. 26–27, 1970, pp. 136–142.
[32] R. C. Singleton, "On computing the fast Fourier transform," *Commun. ACM*, vol. 10, no. 10, pp. 647–654, Oct. 1967.
[33] IEEE Acoust., Speech, Signal Processing Society, *Programs for Digital Signal Processing*. New York: IEEE Press, 1979.
[34] H. V. Sorensen, D. L. Jones, C. S. Burrus, and M. T. Heideman, "On computing the discrete Hartley transform," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-33, pp. 1231–1238, Oct. 1985.
[35] H. V. Sorensen, D. L. Jones, M. T. Heideman, and C. S. Burrus, "A split-radix real-valued fast Fourier transform," in *Proc. 3rd Eur. Signal Processing Conf.*, The Hague, The Netherlands, Sept. 2–5, 1986, pp. 287–290.
[36] H. V. Sorensen, M. T. Heideman, and C. S. Burrus, "On computing the split-radix FFT," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-34, pp. 152–156, Feb. 1986.
[37] Y. Suzuki, T. Sone, and K. Kido, "A new FFT algorithm of radix 3, 6, and 12," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-34, pp. 380–383, Apr. 1986.
[38] M. Vetterli and H. J. Nussbaumer, "Simple FFT and DCT algorithms with reduced number of operations," *Signal Processing*, vol. 6, no. 4, pp. 267–278, Aug. 1984.
[39] S. Winograd, "On computing the discrete Fourier transform," *Math. Comput.*, vol. 32, no. 141, pp. 175–199, Jan. 1978.
[40] R. Yavne, "An economical method for calculating the discrete Fourier transform, *Fall Joint Comput. Conf. AFIPS Conf. Proc.*, vol. 33, p. 1, Spartan, Washington, DC, 1968, pp. 115–125.
[41] H. Ziegler, "A fast Fourier transform algorithm for symmetric real-valued series," *IEEE Trans. Audio Electroacoust.*, vol. AU-20, pp. 353–356, Dec. 1972.



Henrik V. Sorensen (S'84) was born in Skanderborg, Denmark, on January 17, 1959. He received the B.S. and M.S. degrees in electrical engineering from Aalborg University Center, Aalborg, Denmark, in 1981 and 1983, respectively.

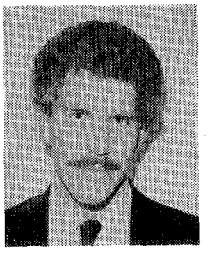
He is presently working as a doctoral student in electrical engineering at Rice University, Houston, TX. His professional interests are in the area of digital signal processing.

Mr. Sorensen is a member of DIF, the Danish Engineering Society.



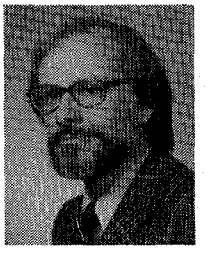
Douglas L. Jones (S'81) was born in Dallas, TX, on January 17, 1961. He received the B.S.E.E. degree (summa cum laude) in 1983 and the M.S.E.E. degree in 1986 from Rice University, Houston, TX.

He is currently engaged in graduate study at Rice University under a National Science Foundation Graduate Fellowship. His research interests are in digital signal processing and time-varying signal analysis.



Michael T. Heideman (S'82) was born in Medford, OR, on November 13, 1956. He received the B.S. and M.S. degrees in electrical engineering from Stanford University, Stanford, CA, in 1978 and 1980, respectively. He received the Ph.D. degree from Rice University, Houston, TX, in 1986.

From 1978 to 1981 he was a member of the Technical Staff of Lockheed Missiles and Space Company, Palo Alto, CA, conducting research in image processing. From 1981 to 1986 he was with Lockheed Engineering and Management Services Company, Houston, TX, where he was involved in the development of radar cross-section models for the Space Shuttle program. He is now back with Lockheed in Palo Alto.



C. Sidney Burrus (S'55–M'61–SM'75–F'81) was born in Abilene, TX, on October 9, 1934. He received the B.A., B.S.E.E., and M.S. degrees from Rice University, Houston, TX, in 1957, 1958, and 1960, respectively, and the Ph.D. degree from Stanford University, Stanford, CA, in 1965.

From 1960 to 1962 he served in the Navy Reserve by teaching at the Nuclear Power School in New London, CT, and during the summers of 1964 and 1965 he was a Lecturer in Electrical Engineering at Stanford University. In 1965 he joined the Faculty at Rice University where he is now Professor and Chairman of Electrical and Computer Engineering. From 1968 to 1975 he was a Visiting Faculty Member at Baylor College of Medicine and a Consultant at the VA Hospital Research Center in Houston. From 1972 to 1978 he was Master of Lovett College at Rice University. In 1975 and again in 1979 he was a Visiting Professor at the Institut für Nachrichtentechnik, Universität Erlangen-Nürnberg, West Germany. During the summer of 1984 he was a Visiting Fellow at Trinity College, Cambridge University, Cambridge, England.

Dr. Burrus is a member of Tau Beta Pi and Sigma Xi. He has received teaching awards from Rice University in 1969, 1974, 1975, 1976, and 1980, an IEEE S-ASSP Senior Award in 1974, a Senior Alexander von Humboldt Award in 1975, a Senior Fulbright Fellowship in 1979, and the IEEE S-ASSP Technical Achievement Award in 1985. He is coauthor (with T. W. Parks) of two books: *DFT/FFT and Convolution Algorithms* and *Digital Filter Design* (New York: Wiley, 1985 and 1987).