On Computing the Split-Radix FFT

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

Abstract—**This paper presents an efficient Fortran program that computes the Duhamel–Hollmann split-radix FFT. An indexing scheme is used that gives a three-loop structure for the split-radix FFT that is very similar to the conventional Cooley–Tukey FFT. Both a decimation-in-frequency and a decimation-in-time program are presented. An arithmetic analysis is made to compare the operation count of the Cooley–Tukey FFT for several different radixes to that of the split-radix FFT. The split-radix FFT seems to require the least total arithmetic of any power-of-two DFT algorithm.**

Introduction

SINCE the discovery of the fast Fourier transform (FFT) [1], [2], considerable effort has gone into the development of FFT algorithms for sequence lengths that are powers of two. Very early, a nested-loop indexing structure was developed which was more efficient than the first recursive programs. Next, the use of higher radixes was employed to reduce the required arithmetic [3]. In 1968, Yavne [4] published an algorithm that used the lowest number of arithmetic operations known at that time, but the paper went relatively unnoticed. Winograd applied the theory of computational complexity to the calculation of the DFT [5]–[7] and derived a lower bound on the number of multiplications required to compute a length-2^M DFT and gave a constructive approach to the generation of these algorithms. These ideas were used to find the minimum number of real multiplications [8], [9] and the minimum number of complex multiplications [10] required to calculate the length-2^M DFT.

Recently, attention has returned to the power-of-two FFT algorithms. Duhamel and Hollmann [11], [12] derived an algorithm called the split-radix FFT (SRFFT) which has the same arithmetic complexity as Yavne's algorithm but has a much simpler structure and a clearer theoretical basis. Martens derived a similar algorithm [13] with essentially the same arithmetic complexity using an approach based on polynomial reduction. Vetterli and Nussbaumer [14] also derived an algorithm with the same arithmetic requirements but using still another approach. These algorithms have the optimal number of multiplications and the minimum known number of additions for lengths up to and including 16, and have what seem to be the best compromise operation count for all longer power-of-two lengths [9].

Manuscript January 25, 1985; revised May 31, 1985. This work was supported by the National Science Foundation under Grant ECS 83-14006.

The authors are with the Department of Electrical and Computer Engineering, Rice University, Houston, TX 77251-1892.

IEEE Log Number 8406026.

This paper presents an indexing scheme which efficiently implements the Duhamel–Hollmann SRFFT. Because the algorithm is in some sense a mixture of a radix-2 and a radix-4 FFT, the indexing is not straightforward. The approach presented here results in a program that has a form very similar to the classical three-loop Cooley–Tukey FFT and allows the same modifications and improvements that are possible with it [13]. The result is an efficient, flexible, and compact program.

The Basic Split-Radix Algorithm

The basic idea behind the SRFFT as derived by Duhamel and Hollmann [11], [12] is the application of a radix-2 index map to the even-indexed terms and a radix-4 map to the odd-indexed terms. The basic definition of the DFT

$$C_k = \sum_{n=0}^{N-1} x_n W^{nk} \tag{1}$$

with $W = e^{-j2\pi/N}$ gives

$$C_{2k} = \sum_{n=0}^{N/2-1} [x_n + x_{n+N/2}] W^{2nk} \tag{2}$$

for the even index terms, and

$$C_{4k+1} = \sum_{n=0}^{N/4-1} [(x_n - x_{n+N/2}) - j(x_{n+N/4} - x_{n+3N/4})] W^n W^{4nk} \tag{3}$$

and

$$C_{4k+3} = \sum_{n=0}^{N/4-1} [(x_n - x_{n+N/2}) + j(x_{n+N/4} - x_{n+3N/4})] W^{3n} W^{4nk} \tag{4}$$

for the odd index terms. This results in an L-shaped "butterfly" which relates a length-N DFT to one length-$N/2$ DFT and two length-$N/4$ DFT's with twiddle factors. Repeating this process for the half- and quarter-length DFT's, until scalars result, gives the SRFFT algorithm in much the same way as the decimation-in-frequency radix-2 Cooley–Tukey FFT is derived [16], [17]. The resulting flow graph for the algorithm calculated in place looks like a radix-2 FFT except for the location of the twiddle factors. Indeed, it is the location of the twiddle factors that makes this algorithm essentially different. The L-shaped SRFFT butterfly advances the calculation of the top half by one of the M stages while the lower half, much as a

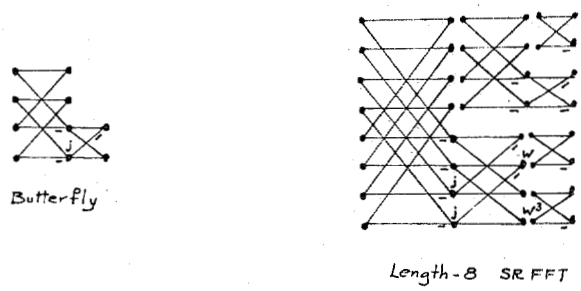


Fig. 1. The split-radix FFT structure.

radix-4 butterfly, calculates two stages at once. This can be seen in Fig. 1.

Unlike the fixed radix, mixed radix, or variable radix [18] Cooley–Tukey FFT or even the prime factor algorithm (PFA) [19] or Winograd Fourier transform algorithm (WFTA), the split-radix FFT does not progress stage by stage or, in terms of indexes, does not complete each nested sum in order. This is perhaps best seen from the polynomial reduction strategy of Martens [13].

The Split-Radix FFT Program

An indexing scheme has been worked out which allows a program to calculate the SRFFT stage by stage. This is not simple because of the mixture of radix-2 and radix-4 indexing. For example, the first stage will always calculate $N/4$ butterflies and associated twiddle factors. The next stage, however, only operates on $N/2$ data points since the L-shaped butterflies of the first stage have already calculated half of that stage. Keeping track of which part of a particular stage has already been calculated is the indexing problem to be solved. This is done by generating the indexes with a mixture of steps of two and four. A three-loop Fortran program was first written which stepped through the M stages in the outer loop, stepped through the even and odd divisions in the second loop, and calculated the adjacent butterflies and their twiddle factors in the innermost loop. In order to minimize the number of sine and cosine evaluations or lookups, the order of the second and third loop indexing was reversed. This resulted in a program structure very similar to the standard Cooley–Tukey FFT. The basic one-butterfly program is given in the Appendix using a notation parallel with that in [19] and [15].

The DO 10 loop in the Fortran program steps sequentially through the M stages of the length-2^M FFT setting up the necessary index increments. The DO 20 and DO 30 loops calculate the butterflies and twiddle factors from (2), (3), and (4). The DO 30 loop steps through to find each uncalculated block of $N4$ butterflies. The first set of passes through the DO 30 loop finds most of the blocks. The IF–GOTO 40 branch resets the DO 30 starting point and step increment and then repeats until all the blocks are found. It is this repeated loop that allows the efficient generation of the necessary indexes. Note that the blocks are not found in sequential order. The DO 20 loop moves the starting point of the DO 30 loop ahead by one, calculates the next values for the sine and cosine terms, and then restarts the DO 30 loop until all the butterflies for that stage are calculated.

Because the final stage of the split-radix FFT consists of length-2 DFT's, the length-4, L-shaped butterfly is not proper and a special final stage is programmed in the DO 60 loop. The same indexing scheme is used to find the location of the length-2 DFT's.

The calculations are done in place with the four outputs of each L-butterfly stored in the input locations in bit-reversed order. This results in the total program giving the output in bit-reversed order, hence, the bit-reversed counter at the end of the program.

Modifications to the Program

The same modifications that have been made to the Cooley–Tukey FFT can also be made to the SRFFT. The basic derivation of Duhamel and Hollmann gives what is called a decimation-in-frequency algorithm. In many cases, this is the preferred form since the bit-reversed unscrambler is located after the FFT itself. It is possible to develop a decimation-in-time version of the SRFFT which also has a very similar structure to the decimation-in-time Cooley–Tukey FFT. That program is also included in the Appendix of this paper. The decimation-in-time structure has a symmetry that allows a simpler modification to a real-data algorithm [12]. Although not done in the program in the Appendix, it also allows a modification of the butterfly to reduce temporary storage variables [20].

The basic program given in the Appendix does not remove extraneous multiplications and additions. The first pass through the DO 20 loop uses twiddle factors of W raised to the zero power and, therefore, requires no multiplications. A special butterfly can be inserted just before the DO 30 statement that uses no multiplications and fewer additions. The DO 30 loop should then start at $J = 2$. This program is called a two-butterfly algorithm and uses significantly fewer operations than the basic one-butterfly algorithm. A three-butterfly program can be written that reduces the number of multiplications from four to two for the case where the twiddle factor is $W^{N/8}$ or $W^{3N/8}$. This will result in a program with the least arithmetic possible for the SRFFT, but the third butterfly does not offer as much improvement as the second butterfly (it gives no reduction in additions) and is more difficult to incorporate in the program.

If the computer or hardware used to implement the algorithm has a significantly slower multiplication than addition, the three-multiplication three-addition algorithm for the complex twiddle factor multiplication can be used rather than the usual four-multiplication two-addition version given in the Appendix. The use of a table lookup or recursive calculation of the sine and cosine values can easily be used. Special care in programming and the use of in-line code as discussed by Morris [20] gives the same improvements as for the Cooley–Tukey FFT. These modifications are explained for the Cooley–Tukey FFT in [15].

Approximately one-half of the reduction in arithmetic obtained by adding the third-butterfly to the two-butterfly

program can be obtained by directly programming the last three stages. Special length 4 and 8 butterflies are written which are optimal and which are used for the last three stages. In addition to a reduction in arithmetic, there is also a reduction in data transfers and index calculations. This program is called a two-butterfly-plus-two-stages algorithm. A copy of a program with two butterflies and two special last stages, and with a table lookup for twiddle factor values can be obtained directly from the authors.

The ideas behind application of the split-radix decomposition to the calculation of the DFT also apply to other transforms. The authors have written a split-radix fast Hartley transform [21], Vetterli and Nussbaumer [14] have applied it to cosine transforms, and it can be applied to number theoretic transforms and others.

Evaluation of the Programs

An analysis of the number of arithmetic operations required by the various forms of the split-radix program, and a comparison to efficient Cooley–Tukey programs, is necessary to evaluate this new approach to calculating the FFT. Table I gives the number of real multiplications and real additions required by the SRFFT to calculate a length-N DFT of complex data using four real multiplications and two real additions for each complex multiplication by a twiddle factor. Table II contains the same values when using three real multiplications and three real additions for each complex multiplication.

Table III compares the operation count of a radix-2, radix-4, and split-radix FFT. The three butterfly versions of the radix-2 and the radix-4 FFT's are practical, efficient examples of Cooley–Tukey FFT's. Values for both the two-butterfly SRFFT with explicit final stages and the three-butterfly SRFFT are given for comparison.

A table for the same programs using the 3-multiply 3-add complex multiply would show the same relative results. For further comparisons to the PFA and WFTA, consult the tables in [15].

The following formulas can be used to generate counts for longer lengths that use the 4-multiply 2-add scheme for $N = 2^M$.

One-Butterfly

$$\text{Multiplies} = (4/3)MN - (8/9)N + (-1)^M(8/9)$$
$$\text{Adds} = (8/3)MN - (4/9)N + (-1)^M(4/9)$$

Two-Butterfly

$$\text{Multiplies} = (4/3)MN - (32/9)N + 4 - (-1)^M(4/9)$$
$$\text{Adds} = (8/3)MN - (16/9)N + 2 - (-1)^M(2/9)$$

Two-Butterfly plus Two Stages

$$\text{Multiplies} = (4/3)MN - (35/9)N + 4 + (-1)^M(8/9)$$
$$\text{Adds} = (8/3)MN - (16/9)N + 2 - (-1)^M(2/9)$$

Three-Butterfly

$$\text{Multiplies} = (4/3)MN - (38/9)N + 6 + (-1)^M(2/9)$$
$$\text{Adds} = (8/3)MN - (16/9)N + 2 - (-1)^M(2/9)$$

The following give the same counts for the 3-mult-3-add scheme.

TABLE I
The Number of Real Multiplications and Additions for Four SRFFT Programs Using Four Real Multiplications and Two Real Additions per Complex Multiplication for N Complex Data Points

Length	One-BF		Two-BF		Two-BF+		Three-BF	
N	M1	A1	M2	A2	M2+	A2+	M3	A3
8	24	60	8	52	4	52	4	52
16	72	164	32	144	28	144	24	144
32	184	412	104	372	92	372	84	372
64	456	996	288	912	268	912	248	912
128	1080	2332	744	2164	700	2164	660	2164
256	2504	5348	1824	5008	1740	5008	1656	5008
512	5688	12060	4328	11380	4156	11380	3988	11380
1024	12744	26852	10016	25488	9676	25488	9336	25488
2048	28216	59164	22760	56436	22076	56436	21396	56436
4096	61896	129252	50976	123792	49612	123792	48248	123792
8192	134712	280348	112872	269428	110140	269428	107412	269428
16384	291272	604388	247584	582544	242124	582544	236664	582544

TABLE II
The Number of Real Multiplications and Additions for Four SRFFT Programs Using Three Real Multiplications and Three Real Additions per Complex Multiplication for N Complex Data Points

Length	One-BF		Two-BF		Two-BF+		Three-BF	
N	M1	A1	M2	A2	M2+	A2+	M3	A3
8	18	66	6	54	4	52	4	52
16	54	182	24	152	22	150	20	148
32	138	458	78	398	72	392	68	388
64	342	1110	216	984	206	974	196	964
128	810	2602	558	2350	536	2328	516	2308
256	1878	5974	1368	5464	1326	5422	1284	5380
512	4266	13482	3246	12462	3160	12376	3076	12292
1024	9558	30038	7512	27992	7342	27822	7172	27652
2048	21162	66218	17070	62126	16728	61784	16388	61444
4096	46422	144726	38232	136536	37550	135854	36868	135172
8192	101034	314026	84654	297646	83288	296280	81924	294916
16384	218454	677206	185688	644440	182958	641710	180228	638980

TABLE III
The Number of Real Multiplications and Additions for Four FFT Programs Using Four Real Multiplications and Two Real Additions per Complex Multiplication for N Complex Data Points

Length	Radix-2 Three-BF		Radix-4 Three-BF		Split-Radix Two-BF +		Split-Radix Three-BF	
N	M1	A1	M1	A1	M2+	A2+	M3	A3
8	8	52			4	52	4	52
16	40	148	28	144	28	144	24	144
32	136	388			92	372	84	372
64	392	964	284	920	268	912	248	912
128	1032	2308			700	2164	660	2164
256	2568	5380	1884	5080	1740	5008	1656	5008
512	6152	12292			4156	11380	3988	11380
1024	14344	27652	10588	25944	9676	25488	9336	25488
2048	32776	61444			22076	56436	21396	56436
4096	73736	135172	54620	126296	49612	123792	48248	123792
8192	163848	294916			110140	269428	107412	269428
16384	360456	638980	267612	595288	242124	582544	236664	582544

One-Butterfly

$$\text{Multiplies} = MN - (2/3)N + (-1)^M(2/3)$$
$$\text{Adds} = 3MN - (2/3)N + (-1)^M(2/3)$$

Two-Butterfly

$$\text{Multiplies} = MN - (8/3)N + 3 - (-1)^M(1/3)$$
$$\text{Adds} = 3MN - (8/3)N + 3 - (-1)^M(1/3)$$

Two-Butterfly plus Two Stages

$$\text{Multiplies} = MN - (17/6)N + 3 + (-1)^M(1/3)$$
$$\text{Adds} = 3MN - (17/6)N + 3 + (-1)^M(1/3)$$

Three-Butterfly

$$\text{Multiplies} = MN - 3N + 4$$
$$\text{Adds} = 3MN - 3N + 4$$

Note that the addition of the second butterfly gives a significant improvement. The third butterfly is probably not worth the programming trouble, but adding the last two special stages probably is worthwhile. The actual execution times are both hardware and compiler dependent and, therefore, the final evaluation would require timings on the particular system to be used.

Conclusions

This paper has developed an efficient, flexible, compact program which implements the split-radix FFT algorithm of Duhamel and Hollmann. The structure of the program is very similar to that of a conventional Cooley–Tukey FFT and is simple to program. Most of the various improvements that can be applied to the Cooley–Tukey FFT can also be applied to the SRFFT. An analysis of the arithmetic complexity showed that the SRFFT is a significant improvement over the Cooley–Tukey FFT and is probably optimal in the sense of minimum floating point arithmetic. It has an efficiency exceeding a radix-8 FFT, a size comparable to a radix-4 FFT, and the flexibility of a radix-2 FFT.

Example Fortran programs for the decimation-in-frequency and the decimation-in-time algorithms are included in the Appendix. A two-butterfly program with special last stages and table lookup of the twiddle factors is available directly from the authors.

Appendix

```
C-----------------------------------------------------------------C
C       A DUHAMEL-HOLLMAN SPLIT-RADIX DIF FFT                     C
C       REF: ELECTRONICS LETTERS, JAN. 5, 1984                    C
C       COMPLEX INPUT AND OUTPUT DATA IN ARRAYS X AND Y           C
C       LENGTH IS N = 2 ** M                                      C
C       C.S. BURRUS,       RICE UNIV.       DEC. 1984             C
C-----------------------------------------------------------------C
C
        SUBROUTINE FFT(X,Y,N,M)
        REAL X(1),Y(1)
C
        N2 = 2*N
        DO  10 K = 1, M-1
            N2 = N2/2
            N4 = N2/4
            E  = 6.283185307179586/N2
            A = 0
            DO  20 J = 1, N4
                A3  = 3*A
                CC1 = COS(A)
                SS1 = SIN(A)
                CC3 = COS(A3)
                SS3 = SIN(A3)
                A   = J*E
                IS  = J
                ID  = 2*N2
 40             DO 30 I0 = IS, N-1, ID
                    I1 = I0 + N4
                    I2 = I1 + N4
                    I3 = I2 + N4
C
                    R1    = X(I0) - X(I2)
                    X(I0) = X(I0) + X(I2)
                    R2    = X(I1) - X(I3)
                    X(I1) = X(I1) + X(I3)
                    S1    = Y(I0) - Y(I2)
                    Y(I0) = Y(I0) + Y(I2)
                    S2    = Y(I1) - Y(I3)
                    Y(I1) = Y(I1) + Y(I3)
C
                    S3    = R1 - S2
                    R1    = R1 + S2
                    S2    = R2 - S1
                    R2    = R2 + S1
                    X(I2) = R1*CC1 - S2*SS1
                    Y(I2) =-S2*CC1 - R1*SS1
                    X(I3) = S3*CC3 + R2*SS3
                    Y(I3) = R2*CC3 - S3*SS3
 30             CONTINUE
                IS = 2*ID - N2 + J
                ID = 4*ID
                IF (IS.LT.N) GOTO 40
 20         CONTINUE
 10     CONTINUE
C
C---------------LAST STAGE, LENGTH-2 BUTTERFLY--------------C
C
        IS = 1
        ID = 4
 50     DO 60 I0 = IS, N, ID
            I1    = I0 + 1
            R1    = X(I0)
            X(I0) = R1 + X(I1)
            X(I1) = R1 - X(I1)
            R1    = Y(I0)
            Y(I0) = R1 + Y(I1)
            Y(I1) = R1 - Y(I1)
   60   CONTINUE
            IS = 2*ID - 1
            ID = 4*ID
        IF (IS.LT.N) GOTO 50
C
C---------------BIT REVERSE COUNTER-------------------------C
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
            XT   = Y(J)
            Y(J) = Y(I)
            Y(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
        RETURN
        END
```

```
C-----------------------------------------------------------------C
C                                                                 C
C       A DUHAMEL-HOLLMAN SPLIT-RADIX DIT FFT                     C
C       REF: Electronics Letters, Jan. 5, 1984                    C
C       Complex input and output data in arrays X and Y           C
C       Length is N = 2 ** M                                      C
C                                                                 C
C       H.V. Sorensen,    Rice University,    Jan. 4  1985       C
C                                                                 C
C-----------------------------------------------------------------C
C
        SUBROUTINE FFT(X,Y,N,M)
        REAL X(1),Y(1)
C
C-------------------Digit reverse counter---------------------C
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
            XT   = Y(J)
            Y(J) = Y(I)
            Y(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
C-------------------Length two transforms---------------------C
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)
            R1    = Y(I0)
            Y(I0) = R1 + Y(I1)
            Y(I1) = R1 - Y(I1)
  60    CONTINUE
        IS = 2*ID - 1
        ID = 4*ID
        IF (IS.LT.N) GOTO 70
C
C
C-------------------L SHAPED BUTTERFLIES----------------------C
C
        N2 = 2
        DO  10 K = 2, M
            N2 = N2*2
            N4 = N2/4
            E  = 6.283185307179586/N2
            A = 0
            DO  20 J = 1, N4
                A3 = 3*A
                CC1 = COS(A)
                SS1 = SIN(A)
                CC3 = COS(A3)
```

```
            SS3 = SIN(A3)
            A   = J*E
            IS  = J
            ID  = 2*N2
 40         DO  30 I0 = IS, N-1, ID
                I1 = I0 + N4
                I2 = I1 + N4
                I3 = I2 + N4
C
                R1     = X(I2)*CC1 + Y(I2)*SS1
                S1     = Y(I2)*CC1 - X(I2)*SS1
                R2     = X(I3)*CC3 + Y(I3)*SS3
                S2     = Y(I3)*CC3 - X(I3)*SS3
                R3     = R1 + R2
                R2     = R1 - R2
                R1     = S1 + S2
                S2     = S1 - S2
C
                X(I2) = X(I0) - R3
                X(I0) = X(I0) + R3
                X(I3) = X(I1) - S2
                X(I1) = X(I1) + S2
                Y(I2) = Y(I0) - R1
                Y(I0) = Y(I0) + R1
                Y(I3) = Y(I1) + R2
                Y(I1) = Y(I1) - R2
C
 30         CONTINUE
             IS = 2*ID - N2 + J
             ID = 4*ID
             IF (IS.LT.N) GOTO 40
 20       CONTINUE
 10    CONTINUE
C
       RETURN
       END
```

Acknowledgment

The authors would like to thank P. Duhamel for preprints of his papers, copies of his programs, and other private communications with us. Thanks are also due to D. L. Jones for his contributions.

References

[1] M. T. Heideman, D. H. Johnson, and C. S. Burrus, "Gauss and the history of the FFT," *IEEE Acoust., Speech, Signal Processing Mag.*, vol. 1, pp. 14–21, Oct. 1984.

[2] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Math. Comput.*, vol. 19, pp. 297–301, Apr. 1965.

[3] L. R. Rabiner and C. M. Rader, Eds., *Digital Signal Processing, Selected Reprints*. New York: IEEE Press, 1972.

[4] R. Yavne, "An economical method for calculating the discrete Fourier transform," in *Proc. Fall Joint Comput. Conf.*, 1968, pp. 115–125.

[5] S. Winograd, "On computing the discrete Fourier transform," *Math. Comput.*, vol. 32, pp. 175–199, 1978.

[6] —, "On the multiplicative complexity of the discrete Fourier transform," *Adv. Math.*, vol. 32, pp. 83–117, May 1979.

[7] —, *Arithmetic Complexity of Computation*, SIAM CBMS-NSF Series, no. 33. Philadelphia PA: SIAM, 1980.

[8] M. T. Heideman and C.S. Burrus, "Multiply/add tradeoffs in length-2^n FFT algorithms," in *Proc. IEEE Int. Conf.* ASSP, Tampa, FL, Apr. 1985.

[9] —, "On the number of multiplications necessary to compute a length-2^n DFT," *IEEE Trans. Acoust., Speech, Signal Processing*, pp. 91–95, this issue.

[10] P. Duhamel and H. Hollmann, "Existence of a 2^n FFT algorithm with a number of multiplications lower than 2^{n+1}," *Electron. Lett.*, vol. 20, pp. 690–692, Aug. 16, 1984.

[11] —,"Split radix FFT algorithm," *Electron. Lett.*, vol. 20, pp. 14–16, Jan. 5, 1984.

[12] P. Duhamel, "Implementation of 'split-radix' FFT algorithms for complex, real, and real-symmetric data," *IEEE Trans. Acoust., Speech, Signal Processing*, to appear.

[13] J. B. Martens, "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.

[14] M. Vetterli and H. J. Nussbaumer, "Simple FFT and DCT algorithms with reduced number of operations," *Signal Processing*, vol. 6, pp. 267–278, Aug. 1984.

[15] C. S. Burrus and T. W. Parks, *DFT/FFT and Convolution Algorithms*. New York: Wiley, 1984.

[16] A. V. Oppenheim and R. W. Schafer, *Digital Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1975.

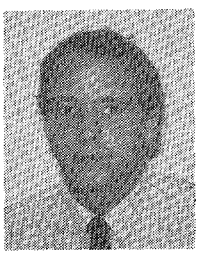
[17] L. R. Rabiner and B. Gold, *Theory and Application of Digital Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1975.

[18] H. W. Johnson and C. S. Burrus, "Twiddle factors in the radix-2 FFT," in *Proc. 1982 Asilomar Conf. Circuits, Syst., Comput.*, Nov. 1982, pp. 413–416.

[19] 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.

[20] L. R. Morris, *Digital Signal Processing Software*. Ottawa, Canada: DSPS Inc., 1982, 1983.

[21] 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.



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.

Michael T. Heideman (S'82), for a photograph and biography, see this issue, p. 95.

C. Sidney Burrus (S'55–M'61–SM'75–F'81), for a photograph and biography, see this issue, p. 95.