An approach to DFT calculations using standard microprocessors

by James T. Rayfield
Harvey F. Silverman

The use of the DFT as an everyday tool is now commonplace, principally due to advances in hardware technology. Special-purpose VLSI chips for signal processing are available. In this paper, we describe an approach which marries the Winograd Fourier Transform Algorithm (WFTA) with a state-of-the-art 16-bit general-purpose microprocessor for the purpose of DFT calculation. The heart of the approach is the real-input 240-point WFTA, which has been carefully optimized for time and space. In particular, an implementation for the 10-MHz M68000 executes in 10.8 ms. A simple hardware module is described which implements the optimized software. The use of the module for the inverse transform and for the complex case is also discussed. Advantages to the approach taken in this paper are the low cost/performance ratio and the general-purpose nature of the system that allows many non-signal-processing functions to be performed by the microprocessor.

1. Introduction

The DFT has become an everyday tool for a broad spectrum of applications, in large part due to advances in hardware technology. In signal analysis [1], speech processing [2, 3], image processing [4], sonar signal processing [5], seismic processing [6], and numerous other areas ranging from mechanical vibration analysis to the analysis of I/O referencing for computer programs [7], DFT techniques are being adopted as the costs go down.

The price paid for using the DFT in an everyday environment has until recently been measured in terms of thousands of dollars. This level has been achieved in several pieces of test equipment such as the Nicolet Oscilloscope [8]. VLSI is currently causing the latest revolution in the use of the DFT, in that the use of special-purpose digital signal processing chips is becoming widespread. The most widely used of these is the Texas Instruments TMS-320 [9], upon which a fixed-point, real-valued DFT of 256 points can be computed in 4.86 ms [10]. (This figure does not include the double-buffered I/O or windowing that is needed in most applications.) Because the TMS-320 currently costs $150, a substantial reduction in systems cost to the end user is easy to predict.

This paper describes an approach to the calculation of the DFT which potentially offers many end users even lower cost than does a special-purpose VLSI implementation. The ideas here are not profound; the WFTA and the real/complex symmetry algorithms are well known to DFT experts; multi-microprocessor structures are well known to hardware and systems experts. However, the combination of using 1) WFTA, 2) a standard general-purpose microprocessor, 3) a tightly optimized assembler-language version of the WFTA for a particularly "good" selection of N and for some useful applications, and 4) algorithmic, hardware, and software extensions based on using a real WFTA as a building module is rather unique. In fact, there are strong advantages to this combination, although at first a

cursory reading of [11] might lead one to believe that the WFTA is ill-suited to the general-purpose computer environment. The WFTA fits the integer/general-purpose microprocessor environment very well.

Although the WFTA in combination with a standard microprocessor has been discussed by Gibson and McCabe [12], the approach taken is to discuss the more traditional complex implementation and a general-N system as suggested in [13, 14]. In sharp contrast, we advocate the use of a *single-N, real-input*, highly optimized implementation as a modular building block for a DFT calculation system. This is reasonable for our application—real-time speech spectrography—and for several others. Most of all, however, it is a most suitable scheme to allow the highest level of time optimization for a standard microprocessor. This is because 1) reasonable microprogrammed multiplication is an instruction set member, although its calculation time is substantially larger than that for addition, 2) sophisticated modes of addressing are available, 3) the number of general-purpose registers has grown, and 4) register/register operations are quite fast. As a result, for the WFTA which was optimized, the percentages of time taken by multiplication, addition, memory reads, and memory writes were evenly balanced.

One of the important aspects of this paper is the comparison of DFT calculation systems. Many real implementations suffer due to the additional processing required for windowing, or due to I/O operations. Thus, we shall compare performance in three ways. First, when only operations for the DFT are considered (i.e., one assumes that no windowing is done, the input data are already in memory, and the output data are written to memory). Second, the cost of using an N-point time window is added to the first measure. Third, the cost of obtaining N points of data and of outputting N transform points [$(N/2) - 1$ complex plus two real values] in an interrupt-driven fashion is considered. Here, it is assumed that two input and two output buffers are used so that the process can proceed in real time; one buffer is used to accumulate data while the other is used for the transform process. The method for comparison is cited in each case.

In the next section, the 240-point real WFTA is introduced and its implementation on a microprocessor is discussed. A short description of general-purpose microprocessor properties affecting the choice of the WFTA is given next, followed by a detailed breakdown of the timing of various parts of a specific implementation. In the following section, software extensions are discussed, such as complex calculation using the real implementation, and inverse transform. Several tradeoffs are possible. The construction of a module which can perform the basic transform is introduced in the fourth section, along with some indication as to how to apply the module as a building element for faster operation or larger transform calculation.

2. Implementation

The principal class of applications for which this WFTA routine was programmed is real-time speech processing. This imposed a number of constraints on the implementation. Most speech processing requires a sampling rate on the order of 6–12 kHz and a transform size which corresponds to a duration of 20–40 ms. This length window is short enough to include only a relatively stationary portion of the speech signal but large enough to include several pitch periods. In addition, it may be desirable in some applications to have some overlap between adjacent windows in the time domain. Together, these factors determine the minimum throughput required for a real-time implementation.

Finally, it is often desirable to have sufficient idle time to allow the processor to handle tasks other than the WFTA, such as premultiplication of the input signal by a time-domain window function, postprocessing of the transform data, and miscellaneous control functions.

• *The 240-point WFTA algorithm*

Window sizes ranging from 210 to 420 points were considered on the basis of the above constraints on sampling rate and window duration. Of the possible sizes in this range, the 240-point algorithm was found to have the lowest number of multiplications per point, $324/240 = 1.35$, and was chosen for that reason; also, 240 is a three-factor algorithm (3, 16, 5) and three-factor algorithms are slightly easier to optimize than are four-factor algorithms. However, the additional cost per point of several other sizes in this range is not that great. For example, the 336-point algorithm has 1.45 multiplications per point, and the 360-point algorithm has 1.47 multiplications per point.

A flow diagram for the 240-point WFTA is shown in **Figure 1.** The first step is a reordering of the input data into three 80-point vectors. These vectors are processed by the input additions of the three-point small-N algorithm and are then formed into three groups of 16 five-vectors. Each of these groups is processed by the input additions of the 16-point small-N algorithm and emerges as 18 five-vectors. Each of these 54 five-vectors is processed by the complete five-point small-N algorithm, and each subgroup of 18 five-vectors is recombined into a group of 16 five-vectors via the output additions for the 16-point algorithm. These are recombined into three 80-vectors, processed by the output additions of the three-point algorithm, and then combined into a 240-vector. This vector is reordered into the desired output vector. The time taken for each phase of the coded 240-point algorithm and the size of that portion of the code are given in **Table 1.**

• *Pertinent features of general-purpose microprocessors*

Some timings for representatives of two important families of 16-bit microprocessors are given in **Table 2.** Among the significant points are that instructions exist for fixed-point

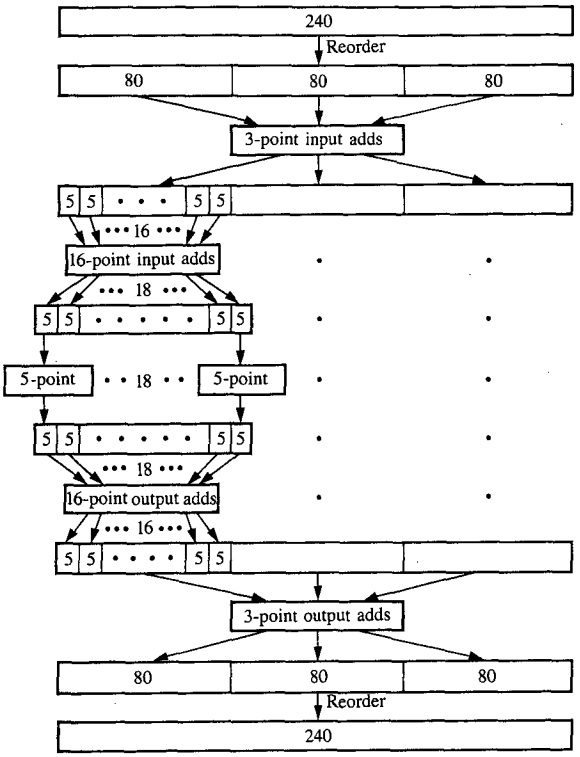

Figure 1

240-Point WFTA flow diagram.

Table 1 Timing summary by WFTA phase for 10-MHz M68000.

Phase	*Time* (ms)	*Code size* (bytes)
Reordering/3-point input additions	1.2076	88
16-point input additions	1.3760	378
5-point complete algorithm	4.5212	186
16-point output additions	1.8608	516
Reordering/3-point output additions	1.8180	170
Subtotal	10.7836	1338
Windowing	2.2572	26
Input/output	2.3040	10
Total	15.3448	1374

Table 2 Basic microprocessor timing (16-bit operands, times in microseconds).

Operation	*68000*	*80186*
Memory cycle	0.4	0.500
Move reg to reg	0.4	0.375
Move mem[base] → reg	0.8	1.125
Move mem[base + displ] → reg	1.2	1.125
Move mem[base + displ + index] → reg	1.4	1.125
Add/subtract reg to reg	0.4	0.375
Multiply reg to reg (avg)	5.4	4.375
Multiply reg to reg (max)	7.0	4.625

multiplication, but, as they are in microcode rather than being implemented in hardware, multiplication requires an order of magnitude more time than a simple addition. This would imply that most of the time required by a standard FFT implementation would be spent performing multiplications. As the WFTA is the algorithm with the minimum number of multiplications, it appears to be very suitable for implementation on such processors.

In addition, standard microprocessors have a large selection of addressing modes, including indirect, indirect plus displacement, auto-increment/decrement, and indirect plus index register plus displacement. These constituted significant aids to the implementation of the WFTA and allowed us to achieve real-time performance for speech spectrography.

• *Details of optimization*

A number of optimizations contributed to the overall speed of the implementation. For example, the reordering, windowing, and first input additions were combined into the same loop. This was done by fetching the data in a reordered fashion, rather than having a separate reordering phase, and by reordering the window coefficients so that they corresponded to the reordered data. In addition to reducing loop overhead, this considerably reduces memory references since intermediate results can be kept in registers. The reordering offsets and window coefficients were combined (alternatingly) into a single vector to allow access to them via a single address register in auto-increment mode. This saves many cycles, as auto-increment mode is free on the M68000.

Fixed integer scaling was used throughout the implementation, using a word length of 16 bits. The program accepts an array of 12-bit integers, typically from an A/D converter, in the range −2048 to +2047 as input, leaving a factor of 16 of headroom. The intermediate results are scaled down by a factor of 4 before the 16-point input additions and by another factor of 4 before the five-point algorithm, thus guaranteeing that no overflow can occur by the end of the inner loop multiplications. Furthermore, if overflow occurs during the five-point, 16-point, or three-point output additions, the final results are guaranteed to be correct. This is because all word-length additions and subtractions are effectively done in modular arithmetic. Thus, the output will be correct even if overflow occurs in intermediate steps as long as the correct results are not truncated at their final destination [15].

We have implemented another version of the program by using a simple variable-scaling method. For the 240-point algorithm, after scaling by 4 before the 16-point input additions, as in the fixed scaling, the program determines the maximum data value after the 16-point input additions, and shifts all the data at this stage up or down by the appropriate number in an allowable shift range (say 3 down to 7 up). Results can be renormalized at the completion of the WFTA. If, for example, a log magnitude is the only result required, as in spectrography, all that is needed is to add the appropriate constant to the final result.

All input additions in our implementation are real (non-complex) additions as we have assumed real data. Also, all multiplications are pure; that is, they involve a real number times a real number or an imaginary number times a real number. For the 240-point implementation, each group of six coefficients for the five-point algorithm is always of the form (R, R, R, I, I, I) or (I, I, I, R, R, R), where R = real and I = imaginary. (A WFTA implementation for another window size results in two different complementary patterns.) The pattern for each group of multiplications is specified by a binary flag stored with each group of six coefficients. This allows two different inner loops to be coded, each of which handles one of the patterns. The code to be executed is chosen by the single test for the six coefficients, which is significantly faster than testing each coefficient separately.

Many of the output additions for the five-point algorithm are pseudo-additions; that is, they involve a real number plus an imaginary number. Thus, only data movement need be done, not actual arithmetic operations in these cases. The binary flags control where the results of the five-point algorithm output additions are placed; this is, indeed, the only code in the inner loop which is controlled by the flags. After these output additions, all data are treated as complex-valued. The real and imaginary components are handled separately by virtually identical sections of code.

The final output additions are combined in a loop with the output reordering. Only 121 frequency points are calculated, since the real-valued nature of the input data yields a symmetric result; that is, there are real-valued terms for $r = 0$ and $r = 120$, and 119 complex terms for $r = [1, 119]$.

- *Packaging*

The current 240-point WFTA is packed as a subroutine for the C language. It is not set up as a double-buffered module. It accepts three parameters: the address of the input array (240 12-bit integers stored in 16-bit words), the address of the output array (120 complex integers of four bytes each), and the address of a scratch area (1520 bytes). The input and output arrays may both be placed at the beginning of the scratch area to save memory. The size of the object code is 1338 bytes, and 1716 bytes of constant data are needed. If buffered accumulation is needed, as in most applications, an additional 1920 bytes of RAM are required over the 1520-byte scratch area. In this case, data may be gathered and disseminated simultaneously with the signal processing, but for a cost of an additional 2.3 ms.

Table 3 Timing summary (no windowing).

Operation	*Time* (ms)	*Percent*
Transfer from memory	2.4690	23
Transfer reg-reg	0.5988	5
Transfer to memory	2.1890	20
Add/subtract	1.5728	15
Multiply	2.2680	21
Shift	1.0932	10
Test/branch	0.0964	1
Loops	0.4964	5
Total	10.7836	100

Table 4 Timing summary (with windowing).

Operation	*Time* (ms)	*Percent*
Transfer from memory	2.5650	20
Transfer reg-reg	0.5988	5
Transfer to memory	2.1890	17
Add/subtract	1.7648	13
Multiply	3.9480	30
Shift	1.3812	10
Test/branch	0.0964	1
Loops	0.4976	4
Total	13.0408	100

- *Timing*

The following tables summarize the execution time required by the WFTA implementation. **Table 3** shows the execution time breakdown for the standard WFTA, while **Table 4** shows the execution time breakdown for the WFTA preceded by a time-domain windowing function.

The execution times are broken down into transfer instructions, arithmetic instructions, and control instructions. Note that the time spent on addition and multiplication is approximately balanced for the non-windowing version. The windowing significantly increases the time spent on multiplication. Also note that little time is wasted on register-to-register transfers and control instructions.

3. Software extensions

- *Inverse transform*

The utility of the approach would be quite limited if the heavily optimized forward algorithm could not be used for

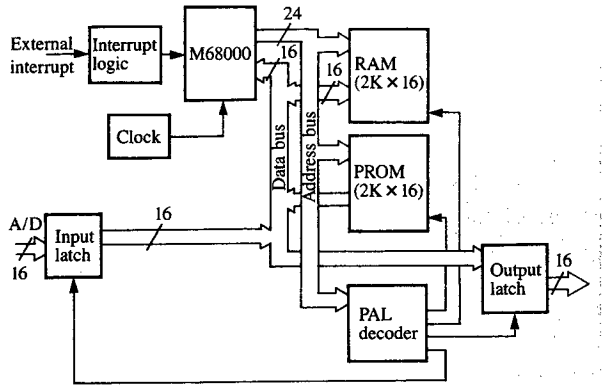

Figure 2

Hardware module.

an inverse. In this section we show that the inverse calculation may be achieved with this routine with very small additional computation and program storage.

Consider the real DFT transform pair

$$x(n) \leftrightarrow X(r) \qquad 0 \le n, r \le N-1, \tag{1}$$

where

$$X(N-r) = X^*(r) \qquad 0 \le r \le \frac{N}{2} \tag{2}$$

and * implies the complex conjugate.

We define A and B as *real* sequences such that

$$X(r) \equiv A(r) + jB(r) \qquad 0 \le r \le N-1. \tag{3}$$

We also define an operator (not a DFT)

$$s(n) \equiv \sum_{r=0}^{\frac{N}{2}-1} S(r) W_N^{nr} \qquad 0 \le n \le N-1, \tag{4}$$

where $S(r)$ is a *real* sequence. If we use Eq. (4) to define $a(n)$ and $b(n)$ from $A(r)$ and $B(r)$ of Eq. (3), the following may be derived from the definition of the inverse DFT:

$$x(n) = \frac{1}{N}\{2\mathrm{Re}[a(n)] + 2\mathrm{Im}[b(n)] - X(0) - X(N/2)\}$$
$$0 \le n \le N-1. \tag{5}$$

Suppose we next form the *real* N-point sequence $z(i)$:

$$z(i) = \begin{cases} A(i) + B(i) & 0 \le i \le \frac{N}{2} - 1, \\ A(N-i) - B(N-i) & \frac{N}{2} \le i \le N-1. \end{cases} \tag{6}$$

We then take the forward real transform of $z(i)$ and attempt to write the resulting data in terms of a and b. By using the DFT definition, substituting Eq. (5), changing the independent variable so that all the summations are in the range $[0, (N/2)-1]$, and using the definition of the complex conjugate of the operator defined in Eq. (4), we get

$$\mathrm{Re}[Z(r)] = 2\mathrm{Re}[a(r)] - A(0) - A(N/2) \qquad 0 \le r \le \frac{N}{2},$$

$$\mathrm{Im}[Z(r) = 2\mathrm{Im}[b(r)] \qquad 0 \le r \le \frac{N}{2}. \tag{7}$$

Thus, we may combine the results of Eqs. (5) and (7) to obtain

$$x(n) = \frac{1}{N}\{\mathrm{Re}[Z(n)] + \mathrm{Im}[Z(n)]\} \qquad 0 \le n \le \frac{N}{2} - 1,$$

$$x(n) = \frac{1}{N}\{\mathrm{Re}[Z(N-n)] - \mathrm{Im}[Z(N-n)]\} \qquad \frac{N}{2} \le n \le N-1. \tag{8}$$

Therefore, the algorithm for the inverse transform using the optimized code is 1) form $z(i)$ from the $(N/2)+1$ frequency components; 2) take the forward transform (apply the routine) to $z(i)$; and 3) apply Eq. (8) to get the time-domain data $x(n)$. If we neglect the divisions by N, only 480 extra additions need be performed over the forward transform, 240 to form $z(i)$ and 240 to obtain $x(n)$. Available address registers imply that only 24 cycles per operation need be done, or an additional 1.152 ms is added to the time taken for the forward transform.

• *Complex transform*

The complex transform may be calculated from the real transform by the most obvious means. If we define a complex $x(n)$ as

$$x(n) = g(n) + jh(n), \tag{9}$$

where $g(n)$ and $h(n)$ are respectively the real and imaginary parts of the complex data, the complex transform may be easily constructed from the real transforms of $g(n)$ and $h(n)$, i.e., $G(r)$ and $H(r)$, by

$$\mathrm{Re}[X(r)] = \begin{cases} \mathrm{Re}[G(r)] - \mathrm{Im}[H(r)] \\ \qquad 0 \le r \le \frac{N}{2} - 1, \\ \mathrm{Re}[G(N-r)] + \mathrm{Im}[H(N-r)] \\ \qquad \frac{N}{2} \le r \le N-1, \end{cases}$$

$$\mathrm{Im}[X(r)] = \begin{cases} \mathrm{Im}[G(r)] + \mathrm{Re}[H(r)] \\ \qquad 0 \le r \le \frac{N}{2} - 1, \\ \mathrm{Re}[H(N-r)] - \mathrm{Im}[G(N-r)] \\ \qquad \frac{N}{2} \le r \le N-1. \end{cases} \tag{10}$$

Thus, two real transforms plus the shuffling of Eq. (10) are required to do the complex transform. The shuffling is *in-place* and 0.24 ms must be added to the cost of a complex 240-point transform. When the calculation time from Table 3 is used, the complex transform takes 21.8 ms, with I/O and windowing costs neglected.

It should be noted that there is only a slight increase in cost in using this technique as compared to writing a complex WFTA program. In the real program, full advantage has been taken of the fact that the data are real all the way through until the final two stages of output additions. Therefore, the complex program would give no advantage in any of these code sections. On the other hand, there is duplication in the last two stages of output additions; it may be inferred from Table 1 that 3.66 ms are *wasted* here. Furthermore, this waste could be eliminated if in-line code were implemented, but this is a very tedious process which implies the use of extensive program memory and offers only a little payoff.

4. Hardware extensions

An example of a hardware module capable of performing real-time DFTs is shown in **Figure 2.** The module includes a 10- or 12.5-MHz M68000, a small amount of very inexpensive static RAM, two PROM or EPROM chips, each of at least 4K × 8, a PAL (or even simpler, upper-address-line assignment) for PROM and I/O decoding, one or two TTL chips for interrupt handling, two latches for I/O and a clock. The 240-point DFT program and all its coefficients are put into the PROM or EPROM, and, in the double-buffered implementation, the module takes a stream of real data, currently from a 12-bit A/D converter. It also outputs transform data at the input sample rate, which can be up to 18 400 Hz assuming that no overlap is required for the output transforms.

This module can also be used to add flexibility to a DFT application. For instance, a number of modules can be used in parallel to improve throughput, perform larger transforms, or perform multidimensional transforms. The improvement in throughput can be easily achieved using several of these modules as slave processors off a single master, providing the master is able to tend, in a temporal fashion, to the care and feeding of the slaves. This same idea holds for multidimensional transforms, although the need to reorganize the data becomes an overriding factor.

For larger transforms, say reasonable numbers such as 480 and 960, one can use a standard butterfly technique of the FFT to combine outputs from 2 or 4, respectively, 240-point real transforms. It is better to apply the existing code in a decimation-in-time combination algorithm, in that the input data to the 240-point transforms are real, and all the advantages of the program apply. It appears somewhat simpler to include additional code and memory in each processor module so that the larger algorithm can be done locally. As is shown in **Table 5,** the additional memory requirement is almost negligible in view of the common sizes for memory chips for the module. Our claim is that the gain in using a direct 504- or 1008-point WFTA and in reoptimizing the code for these sizes is not worthwhile, especially when the 240-point code is in hand. As can be seen in Table 5, a group of eight modules, each with a minimum of 12K bytes of PROM and RAM could perform 960-point real transforms in under 10 ms with no overlap. Thus, a sample rate of approximately 100 kHz could be handled.

Table 5 Modular extension data (double-buffered I/O, no windowing).

Algorithm	*Time* (ms)	*Program PROM*	*Data PROM*	*RAM*
240-point real	13.04	1338	1716	3440
240-point complex	26.32	1388	1716	5360
480-point real	34.72	1500	3258	5360
960-point real	77.16	1700	7098	9200

Some data are extrapolated.

Summary

We have shown that the marriage of a conventional 16-bit microprocessor with a highly optimized WFTA DFT implementation can provide real-time streamed DFT output for surprisingly high data rates. In fact, the 240-point algorithm runs only about a factor of 2 slower than does the 256-point FFT on the TMS-320 signal processing chip. The ability of the general-purpose microprocessor to perform tasks other than signal processing makes the use of this marriage attractive for many implementations.

We have also indicated the simplicity of the hardware, and a few ways in which this highly optimized code may be used for more extensive purposes, such as the calculation of the inverse 240-point transform, complex transforms, and larger real transforms. We expect that the relationship in performance between this general-purpose microprocessor implementation and that of special-purpose signal processing VLSI will remain about the same in the future; technology appears to be improving throughput at about the same rate. Therefore, we predict that the current approach will continue to be a viable one for the immediate future.

Acknowledgment

This work was principally supported by National Science Foundation Grant ECS-8113494.

References

1. *Model 5820A Cross Channel Spectrum Analyzer, Technical Information,* Wavetek Rockland Inc., Rockleigh, NJ 07647, 1983.
2. H. F. Silverman and N. R. Dixon, "A Parametrically Controlled Spectrum Analysis System for Speech," *IEEE Trans. Acoust., Speech, & Signal Proc.* **ASSP-22,** 362–381 (October 1974).

3. B. A. Dautrich, L. R. Rabiner, and T. B. Martin, "On the Effects of Varying Filter Bank Parameters on Isolated Word Recognition," *IEEE Trans. Acoust., Speech, & Signal Proc.* **ASSP-31,** 793–807 (August 1983).
4. P. E. Dudgeon and R. M. Mersereau, *Multi-Dimensional Digital Signal Processing*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1984.
5. C. H. Knapp and G. C. Carter, "The Generalized Correlation Method for Estimation of Time Delay," *IEEE Trans. Acoust., Speech, & Signal Proc.* **ASSP-24,** 320–327 (August 1976).
6. A. V. Oppenheim, Ed., *Applications of Digital Signal Processing*, Englewood Cliffs, NJ, 1978, Ch. 7.
7. P. A. W. Lewis, and P. C. Yue, "Statistical Analysis of Program Reference Patterns in a Paging Environment," *Digest of Papers, IEEE International Computer Society Conference (5th)*, New York, 1971, pp. 133–134.
8. Nicolet Oscilloscope Division, *Nicolet 2090 Digital Oscilloscope Instruction Manual*, Nicolet Instrument Corporation, North Vale, NJ 07647.
9. *TMS32010 User's Guide*, Texas Instruments Corporation, Dallas, TX, 1983.
10. L. R. Morris, oral presentation at the Brown/ASSP Workshop on Fast Algorithms, Brown University, Providence, RI, October 1983.
11. L. R. Morris, "A Comparative Study of Time-Efficient FFT and WFTA Programs for General Purpose Computers," *IEEE Trans. Acoust., Speech, & Signal Proc.* **ASSP-26,** 141–150 (April 1978).
12. R. M. Gibson and D. P. McCabe, "Fourier Transform Algorithm Implementations on a General Purpose Microprocessor," *Proceedings, IEEE International Conference on Acoustics, Speech, and Signal Processing*, Atlanta, GA, pp. 670–672.
13. H. F. Silverman, "An Introduction to Programming the Winograd Fourier Transform Algorithm (WFTA)," *IEEE Trans. Acoust., Speech, & Signal Proc.* **ASSP-25,** 152–165 (April 1977).
14. H. F. Silverman, "A Method for Programming the Complex, General-*N* Winograd Fourier Transform Algorithm," *Proceedings, IEEE International Conference on Acoustics, Speech, and Signal Processing*, Hartford, CT, pp. 369–372.
15. R. C. Agarwal, IBM Research Division, Yorktown Heights, NY, private communication.

Received July 5, 1984; revised October 1, 1984

James T. Rayfield *Laboratory for Engineering Man/Machine Systems (LEMS), Division of Engineering, Brown University, Providence, Rhode Island 02912.* Mr. Rayfield is a graduate student in the Division of Engineering at Brown University. His research interests include parallel processing, distributed systems, and signal processing. He worked at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, in the summer of 1983 in the area of image processing support software. Mr. Rayfield currently has an IBM graduate fellowship. He received his Sc.B. degree from Brown University in 1983.

Harvey F. Silverman *Laboratory for Engineering Man/Machine Systems (LEMS), Division of Engineering, Brown University, Providence, Rhode Island 02912.* Dr. Silverman has been a Professor of Engineering at Brown University since 1980. His research interests include speech recognition, digital signal processing, and computer architecture. Prior to 1980, he was a research staff member at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, working in the areas of speech recognition, computer performance analysis, and image processing. Dr. Silverman received his B.S. and B.S.E.E. degrees from Trinity College in 1965 and 1966, respectively, and his Sc.M. and Ph.D. degrees from Brown University in 1968 and 1971, respectively.