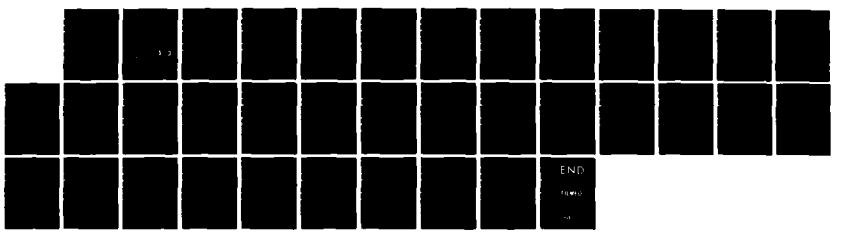
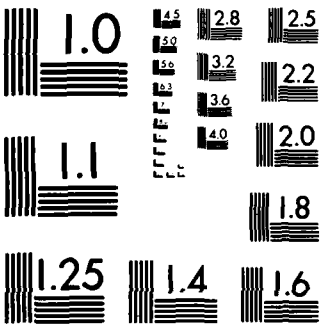
AD-A156 625 AN IMPROVED FFT (FAST FOURIER TRANSFORM) FOR A FOUR PARALLEL PIPE SIMD AR..(U) NAVAL UNDERWATER SYSTEMS CENTER NEW LONDON CT NEW LONDON LAB.. 1/1

UNCLASSIFIED T T TYLASKA ET AL. 01 MAY 85 NUSC-TR-7363 F/G 12/1 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963 A

NUSC Technical Report 7363
1 May 1985

An Improved FFT for a Four Parallel Pipe SIMD Arithmetic Processor

Theodore T. Tylaska
Thomas C. Choinski
Submarine Sonar Department

Naval Underwater Systems Center
Newport, Rhode Island / New London, Connecticut

Preface

This report was prepared under Project No. A75044, "Very High Speed Integrated Circuit Insertions Into the Enhanced Modular Signal Processor," Principal Investigator, T. T. Tylaska (Code 325). The Sponsoring Activity is the Naval Sea Systems Command (Code PMS 412).

The Technical Reviewer for this report was Dr. G. C. Carter (Code 3314).

Reviewed and Approved: 1 May 1985

F. J. Kingsbury
Head, Submarine Sonar Department

The authors of this report are located at the
New London Laboratory, Naval Underwater Systems Center,
New London, Connecticut 06320.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) TR 7363	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Underwater Systems Center 6b OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) New London Laboratory New London, CT 06320	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING / SPONSORING ORGANIZATION Naval Sea Systems Command 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code) Washington, DC 20362	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. 64507N; PROJECT NO. A75044; TASK NO S1440; WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification)
AN IMPROVED FFT FOR A FOUR PARALLEL PIPE SIMD ARITHMETIC PROCESSOR

12 PERSONAL AUTHOR(S)
Theodore T. Tylaska and Thomas C. Choinski

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Progress	FROM _____ TO _____	1 May 1985	34

16 SUPPLEMENTARY NOTATION

17 COSATI CODES		
FIELD	GROUP	SUB-GROUP
09	02,03	

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Discrete Fourier Transforms
Fast Fourier Transforms
Parallel Processors
Signal Processor Architectures

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

An algorithm is derived that reduces the execution time of a fast Fourier transform (FFT) by a factor of 4 utilizing a loosely coupled four parallel pipe processor that has a single instruction, multiple data (SIMD) stream architecture. This algorithm distributes the FFT computations among the four parallel pipes for the radix-4 and the mixed radix-2/-4 cases. The algorithm is demonstrated specifically for both a 16- and a 32-point FFT. In addition, the radix-4 FFT algorithm is derived in detail, along with formulas for the execution times and a method that computes the inverse FFT. The necessity for prescrambling the input data is explained and a simple hardware implementation is given.

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION
☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☒ DTIC USERS	UNCLASSIFIED
22a NAME OF RESPONSIBLE INDIVIDUAL Thomas C. Choinski	22b TELEPHONE (Include Area Code) (203) 440-5391 22c. OFFICE SYMBOL

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

AN IMPROVED FFT FOR A FOUR PARALLEL PIPE SIMD ARITHMETIC PROCESSOR

INTRODUCTION

The effectiveness of sonar and radar is being improved by increased processing of the received signals. The fast Fourier transform (FFT) is a basic building block for various signal processing enhancement algorithms. The computational load imposed by signal bandwidths and number of beams requires parallel processors.

A militarized signal processor, the AN/UYS-2, that utilizes parallelism and data flow constructs is being constructed to satisfy the need for increased computational capability. Size constraints, power and reliability requirements, etc., along with given internal data transfer rates, dictated a single-instruction, multiple-data (SIMD), four parallel pipe arithmetic processor. Connector pin limitations on the boards that are used to package the processor necessitated use of a common coefficient memory feeding the four parallel data pipes. This common coefficient memory also is the only cross-connection between the four pipes.

The problem is to utilize the four parallel data pipes in the given architecture to perform the FFT. One method would transfer data in four independent sets and then perform four parallel FFT's in lock step. However, this is not compatible with the data flow architecture of the AN/UYS-2. Another method would perform one FFT on one data set four times faster using the four parallel pipes simultaneously.

This report describes the second method, whereby an FFT computation on a SIMD four parallel pipe arithmetic processor can proceed approximately four times faster than on a single pipe arithmetic processor with the same instruction cycle rate. The basic concept uses four parallel data processing pipes to compute one radix-4 FFT. Only one pipe would be employed to perform the last stage of the FFT. The first step in explaining the concept is to derive a suitable version of the radix-4 FFT. Next, formulas are derived to show the addition and multiplication time needed to compute an N-point FFT. Then, the FFT algorithm for using mixed radixes is given. (This method works almost as well when computing radix-2 or mixed radix-2 and -4 FFT's.) An efficient scheme for prescrambling the mixed radixes is also derived. This method can be utilized to compute the inverse discrete Fourier transform.

AVAILABLE ARCHITECTURE

The external architecture for the four parallel pipe arithmetic processor is shown in figure 1, and the internal architecture of a pipe is shown in figure 2. Each pipe has its own operand memory that can be simultaneously read and written to. A limitation of this architecture is that the common coefficient memory is shared by all pipes, and crosscoupling between pipes is accomplished by one pipe writing into memory and another pipe reading from that same memory.

With the above parallel processing architecture in mind, the question is how to utilize the hardware to compute the FFT. The radix-4 FFT is shown to be an efficient way to compute the FFT, irrespective of what hardware is used, because four data points are multiplied by the vector [1, j, -1, -j], which reduces the number of complex multiples required by a factor of about 4.[1,2]

CONCEPT AND DERIVATION

In partitioning the FFT into four pipes, i.e., placing one fourth of the data points into each of the separate pipes, a decimation-in-time on the incoming data set can be performed. One formulation of the radix-4 decimation-in-time FFT does not require cross-connection between the four data sets until the very last stage of the FFT. For this last stage, the independent results of the four separate pipes are transferred into the operand memory of only one pipe. This transferral operation is necessary because this stage of the FFT computation requires access to all points to form the resulting frequency sample.

DERIVATION OF A RADIX-4 FFT

The derivation of the radix-4 FFT is based on a decimation-in-time of the input sequence, $x(n)$, into smaller subsequences: Assume that the number of input data points is a power of 4; i.e., $N = 4^v$, and the object will be to compute the discrete Fourier transform (DFT):

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk}, \quad W_N = e^{\frac{-2\pi j}{N}}, \quad k = 0, 1, \ldots, N-1; \; j = \sqrt{-1} \qquad (1)$$

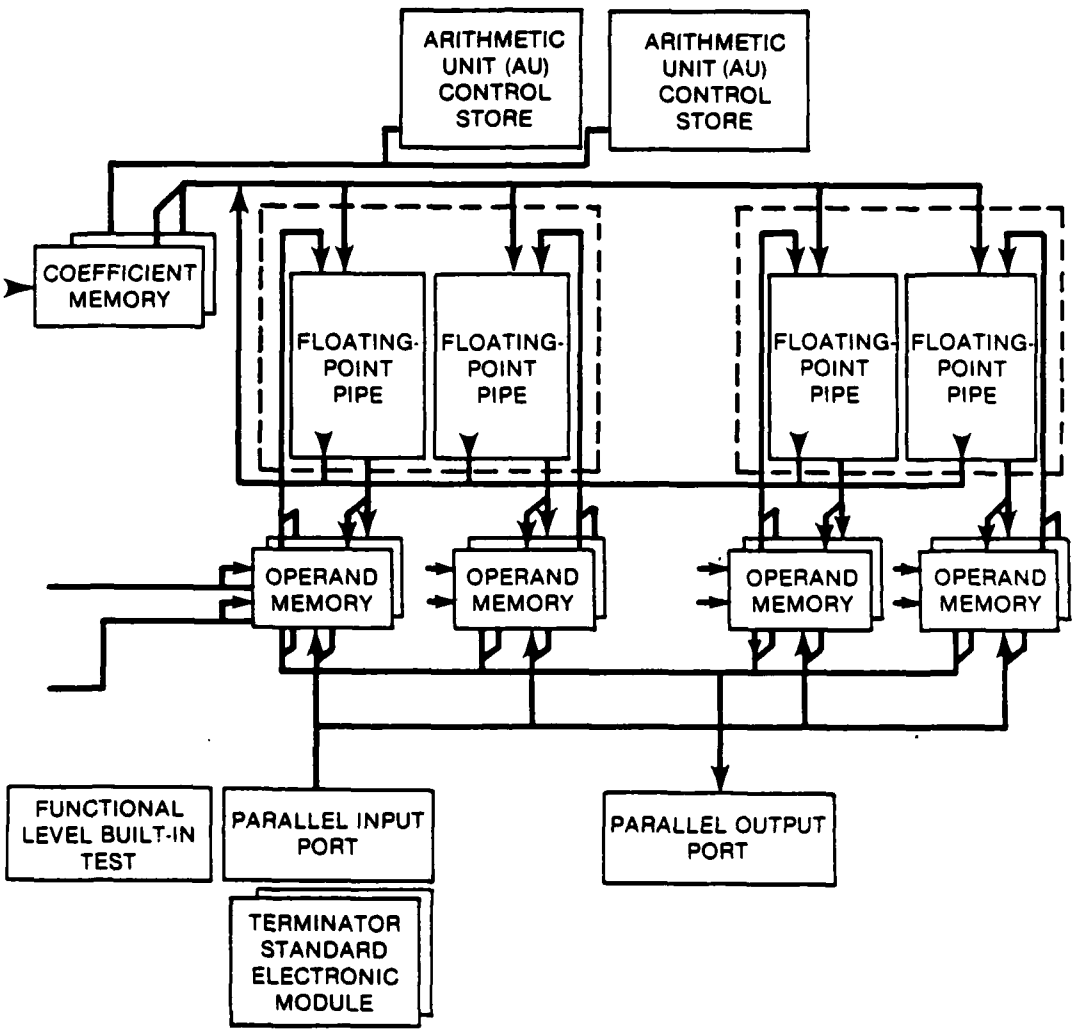


Figure 1. Architecture of the Four Parallel Pipe Arithmetic Processor

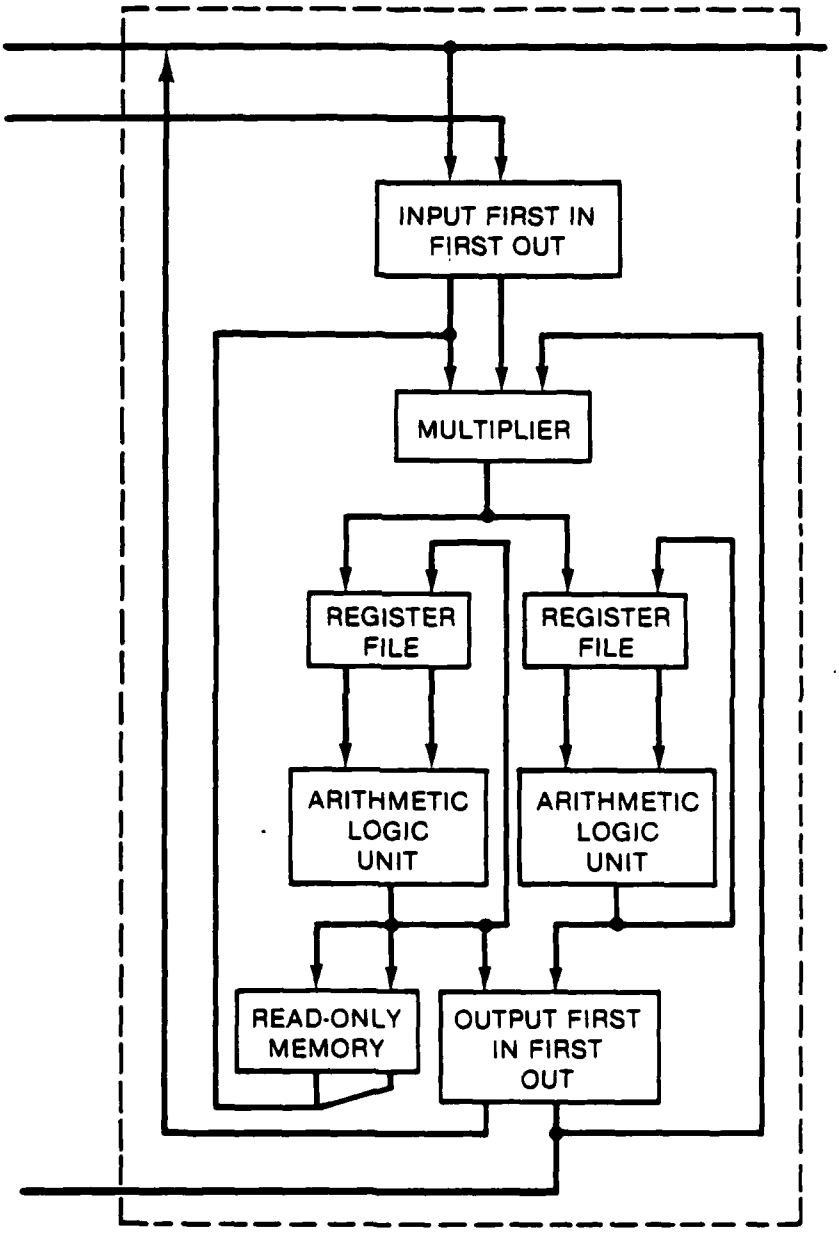


Figure 2. Internal Architecture of One Pipe

Since N is a power of 4, separate the input sequence, x(n), into four N/4 data point subsequences as follows:

$$4r,\ 4r + 1,\ 4r + 2,\ 4r + 3,$$

where

$$r = 0,\ 1,\ \ldots\ ,\ (N/4) - 1.$$

Now write the DFT as

$$X(k) = \sum_{r=0}^{(N/4)-1} x(4r)W_N^{4rk} + \sum_{r=0}^{(N/4)-1} x(4r+1)W_N^{(4r+1)k}$$

$$+ \sum_{r=0}^{(N/4)-1} x(4r+2)W_N^{(4r+2)k} + \sum_{r=0}^{(N/4)-1} x(4r+3)W_N^{(4r+3)k}, \quad (2)$$

$$k = 0,\ 1,\ \ldots,\ N - 1.$$

Next, write the DFT in terms of the four subsequences:

$$X(k) = \sum_{r=0}^{(N/4)-1} x(4r)W_N^{4rk} + W_N^k \sum_{r=0}^{(N/4)-1} x(4r+1)W_N^{4rk}$$

$$+ W_N^{2k} \sum_{r=0}^{(N/4)-1} x(4r+2)W_N^{4rk} + W_N^{3k} \sum_{r=0}^{(N/4)-1} x(4r+3)W_N^{4rk}. \quad (3)$$

But

$$W_N^{4r} = W_{\frac{N}{4}} \tag{4}$$

because $W_N^{4r} = e^{\frac{-2\pi j 4r}{N}} = e^{\frac{-2\pi j\, r}{(N/4)}} = W_{(N/4)}^{r}$.

So

$$X(k) = \sum_{r=0}^{(N/4)-1} x(4r) W_{(N/4)}^{rk} + W_N^{k} \sum_{r=0}^{(N/4)-1} x(4r+1) W_{(N/4)}^{rk}$$

$$+ W_N^{2k} \sum_{r=0}^{(N/4)-1} x(4r+2) W_{(N/4)}^{rk} + W^{3k} \sum_{r=0}^{(N/4)-1} x(4r+3) W_{(N/4)}^{rk}. \tag{5}$$

Although the index k ranges over N values, k = 0, 1, ... , N - 1, each of the sums, $G_0(k)$, $G_1(k)$, $G_2(k)$, and $G_3(k)$, need be computed for k = 0, 1, 2, ... , (N/4) - 1 because each is periodic in k with a period of N/4, where

$$G_0(k) = \sum_{r=0}^{(N/4)-1} x(4r) W_{(N/4)}^{rk}, \tag{6}$$

$$G_1(k) = \sum_{r=0}^{(N/4)-1} x(4r+1) W_{(N/4)}^{rk}, \tag{7}$$

$$G_2(k) = \sum_{r=0}^{(N/4)-1} x(4r + 2)W_{(N/4)}^{rk}, \quad (8)$$

and

$$G_3(k) = \sum_{r=0}^{(N/4)-1} x(4r + 3)W_{(N/4)}^{rk}. \quad (9)$$

Note that each of the sums in the above expressions is an N/4 data point transform. Also, as k ranges over 0, 1, ... , N - 1, each of the sums produce only N/4 different values of the N/4 point transforms.

Now rewrite equation (5) as

$$X(k) = G_0(k) + W_N^k G_1(k) + W_N^{2k} G_2(k) + W_N^{3k} G_3(k), \; k = 0, 1, \ldots, N - 1. \quad (10)$$

But, as noted above, each $G_i(k)$, i = 0,1,2,3, is periodic in k of period N/4. Since k ranges from 0, 1, ... , N - 1 values, partition k into four sequences, which are given by

$$m, \; m + (N/4), \; m + 2(N/4), \; m + 3(N/4); \; m = 0, \ldots, (N/4) - 1.$$

Substitute these m values for k in equation (10) to obtain the basic radix-4 kernal:

$$X(m) = G_0(m) + W_N^m G_1(m) + W_N^{2m} G_2(m) + W_N^{3m} G_3(m), \; m = 0, \ldots, (N/4) - 1 \quad (11)$$

$$X[m + (N/4)] = G_0(m) + W_N^{[m+(N/4)]}G_1(m) + W_N^{2[m+(N/4)]}\ G_2(m)$$

$$+ W_N^{3[m+(N/4)]}G_3(m), \tag{12}$$

$$X[m + 2(N/4)] = G_0(m) + W_N^{[m+2(N/4)]}G_1(m) + W_N^{2[m+2(N/4)]}G_2(m)$$

$$+ W_N^{3[m+2(N/4)]}G_3(m), \tag{13}$$

and

$$X[m + 3(N/4)] = G_0(m) + W_N^{[m+3(N/4)]}G_1(m) + W_N^{2[m+3(N/4)]}G_2(m)$$

$$+ W_N^{3[m+3(N/4)]}G_3(m). \tag{14}$$

Separating the k frequency terms into four sets, as shown above, and factoring common powers of W_N reduces the number of multiplications in equations (11) through (14) from 12 to 3. This reduction is seen by noting that

$$W_n^{[m+(N/4)]} = W_N^m\ W_N^{(N/4)} = (-j)W_N^m \tag{15}$$

$$W_N^{2[m+(N/4)]} = W_N^{2m}\ W_N^{2(N/4)} = (-1)W_N^{2m}, \tag{16}$$

$$W_N^{3[m+(N/4)]} = W_N^{3m}\ W_N^{3(N/4)} = (+j)W_N^{3m}, \tag{17}$$

$$W_N^{[m+2(N/4)]} = (-1) \bullet W_N^m, \tag{18}$$

$$W_N^{2[m+2(N/4)]} = (1) \bullet W_4^{2m}, \tag{19}$$

$$W_N^{3[m+2(N/4)]} = (-1) \bullet W_N^{3m}, \tag{20}$$

$$W_N^{[m+3(N/4)]} = (+j) \bullet W_N^m, \tag{21}$$

$$W_N^{2[m+3(N/4)]} = (-1) \bullet W_N^{2m}, \tag{22}$$

and

$$W_N^{3[m+3(N/4)]} = (j) \bullet W_N^{3m} . \tag{23}$$

Now, where appropriate, substitue equations (15) through (23) into (11) through (14) to simplify the basic radix-4 kernal in equations (24) through (27):

$$X(m) = G_0(m) + W_N^m G_1(m) + W_N^{2m} G_2(m) + W_N^{3m} G_3(m) \tag{24}$$
$$m = 0, 1, \ldots, (N/4) - 1,$$

$$X[m + (N/4)] = G_0(m) + (-j) W_N^m G_1(m) + (-1) W_N^{2m} G_2(m) + (+j) W_N^{3m} G_3(m), \tag{25}$$

$$X[m + 2(N/4)] = G_0(m) + (-1) W_N^m G_1(m) + (1) W_N^{2m} G_2(m) + (-1) W_N^{3m} G_3(m), \tag{26}$$

and

$$X[m + 3(N/4)] = G_0(m) + (+j)W_N^m G_1(m) + (-1)W_N^{2m} G_2(m) + (-j)W_N^{3m} G_3(m). \quad (27)$$

The flowchart for the computation in equations (24) through (27) can be drawn as shown in figure 3.

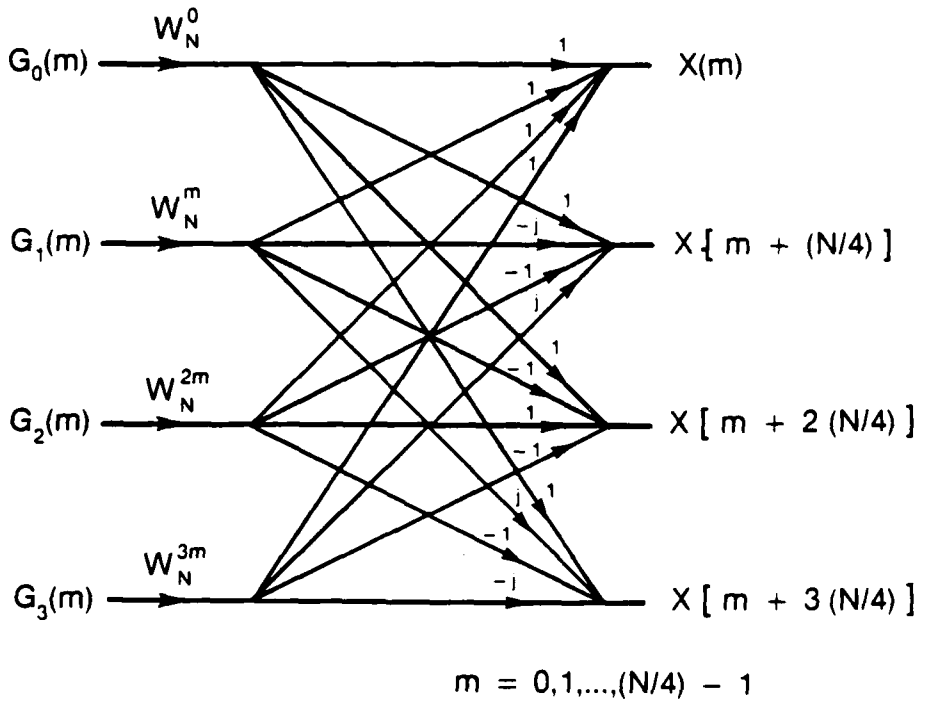


Figure 3. Basic Radix-4 Kernal Computation of a Radix-4 FFT

Each of the $G_i(m)$ are $(N/4)$-point transforms and since $N = 4^v$, each of the transforms can be broken down to four $(N/4)$-point transforms to create another stage in the computation. This staging can be continued until one of the transforms contains only four points; for example, when $m = 0$ and $N = 4$ one obtains

$$X(0) = G_0(0) + W_4^0 G_1(0) + W_4^0 G_2(0) + W_4^0 G_3(0), \tag{28}$$

$$X(0 + 1) = G_0(0) - jW_4^0 G_1(0) - W_4^0 G_2(0) + jW_4^0 G_3(0), \tag{29}$$

$$X(0 + 2) = G_0(0) - 1W_4^0 G_1(0) + W_4^0 G_2(0) - W_4^0 G_3(0), \tag{30}$$

and

$$X(0 + 3) = G_0(0) + jW_4^0 G_1(0) - W_4^0 G_2(0) - jW_4^0 G_3(0). \tag{31}$$

But

$$G_0(0) = x(0), \tag{32}$$

$$G_1(0) = x(1), \tag{33}$$

$$G_2(0) = x(2), \tag{34}$$

and

$$G_3(0) = x(3). \tag{35}$$

Equations (28) through (31) can be condensed to

$$X(k) = x(0)W_4^0 + W_4^k\, x(1) + W_4^{2k} x(2) + W_4^{3k} x(3),\ k = 0,1,2,3. \tag{36}$$

Thus, equations (24) through (27) with $m = 0$ give tne same result as taking a four-point DFT according to equation (1), as shown below:

$$X(k) = \sum_{n=0}^{3} x(n)W_4^{nk} \quad , k = 0,1,2,3, \tag{37}$$

$$X(0) = x(0) + x(1) + x(2) + x(3), \tag{38}$$

$$X(1) = x(0) + W_4^1 x(1) + W_4^2 x(2) + W_4^3 x(3), \tag{39}$$

$$X(2) = x(0) + W_4^2 x(1) + W_4^4 x(2) + W_4^6 x(3), \tag{40}$$

and

$$X(3) = x(0) + W_4^3 x(1) + W_4^6 x(2) + W_4^9 x(3), \tag{41}$$

which reduce to

$$X(0) = x(0) + x(1) + x(2) + x(3), \tag{42}$$

$$X(1) = x(0) - (j)x(1) - x(2) + jx(3), \tag{43}$$

$$X(2) = x(0) - x(1) + x(2) - x(3), \tag{44}$$

and

$$X(3) = x(0) + (j)x(1) - x(2) - jx(4). \tag{45}$$

The result is an algorithm that repeatedly uses the basic computation in figure 3 can be derived.

The advantages of deriving equations (24) through (27) is that the number of multiplications in the basic radix-4 kernal is reduced from 12 to 3 because $G_1(m) = (a + jb)$ multiplied by a power of j reduces to

$$(a + jb)(j) = -b + ja \tag{46}$$

and

$$(a + jb)(-j) = b - ja. \tag{47}$$

Also

$$(a + jb)(-1) = -a - jb. \tag{48}$$

Thus, by interchanging the real and imaginary components or by negating, one is able to perform the required multiplication.

The radix-4 decimation-in-time FFT was derived, and a reduction in multiplicatons by a factor of 4 was shown. Next will be shown how the radix-4 FFT is computed on a four pipe arithmetic processor similar to the one in figures 1 and 2.

ADAPTING RADIX-4 FFT TO A FOUR PIPE ARITHMETIC PROCESSOR

The object here is to divide the input sequence evenly among the four pipes and let each pipe perform a radix-4 kernal computation on its input data points independently of other points in other pipes. See figure 4 for the flowchart for a 16-point radix-4 FFT. Since the four parallel computations use the same coefficients and use the points in exactly the same manner, the computation can be carried out using a SIMD architecture with a common coefficient memory.

The important point is that in stage 1 of the FFT each radix-4 kernal computation in a given pipe "calls" on the data points only within that pipe, and no cross-connection between operand memories is required.

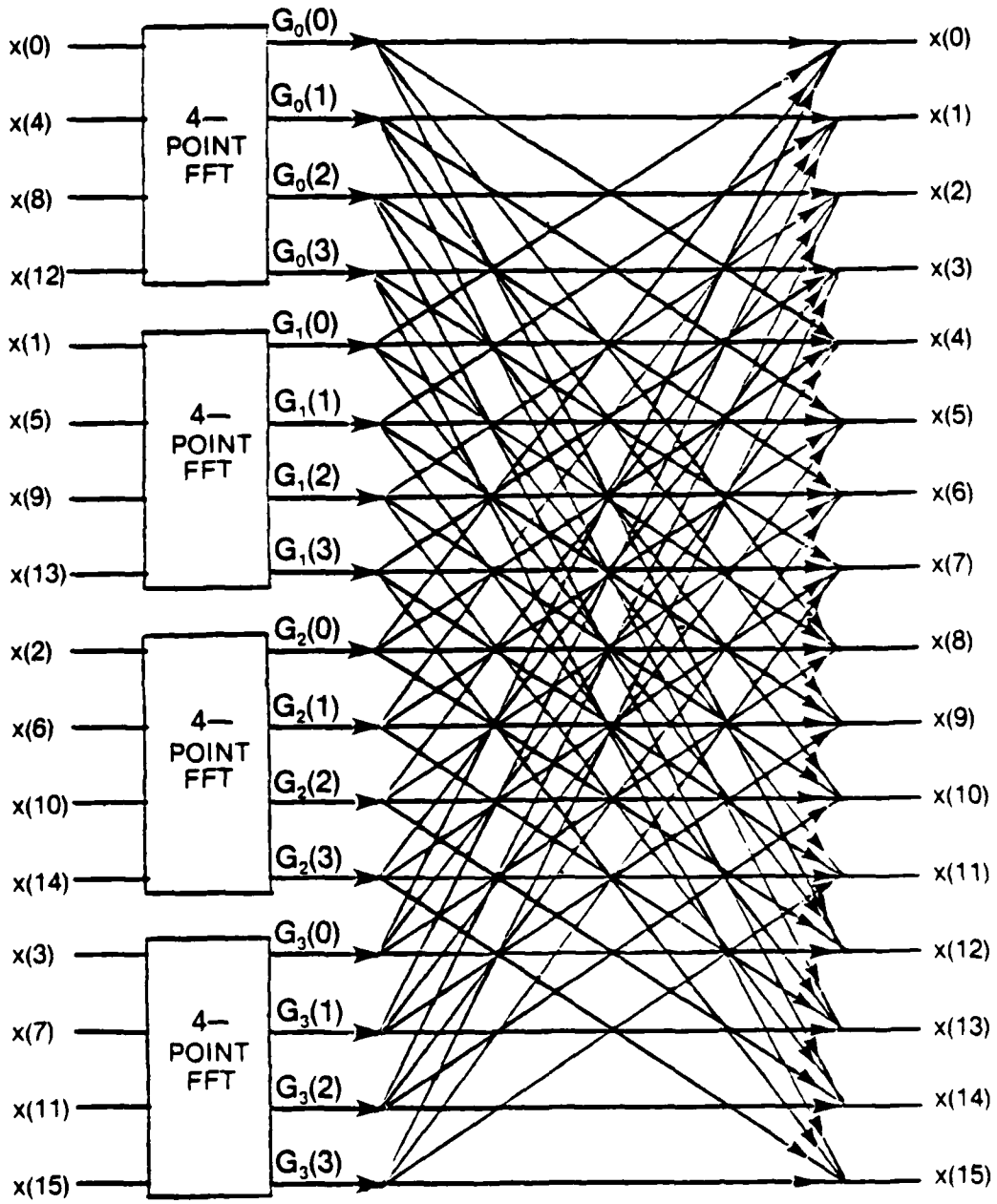


Figure 4. Flowchart for a 16-Data-Point Radix-4 FFT

However, stage 2 (the last stage in general) requires access to all the points of the intermediate results from the previous stage. The architecture restriction on crosscoupling dictates that a single pipe must be used for the last stage of the computation. The architecture allows points to be selectively read from the individual operand memories and routed to a given pipe, say the first pipe in figure 1.

In the last stage of the FFT the appropriate field in the microword of the microsequencer controlling the various data paths is set to read $G_0(0)$ from operand memory 1, $G_1(0)$ from operand memory 2, $G_2(0)$ from operand memory 3, and finally $G_3(0)$ from operand memory 4 to compute $X(0)$. Likewise the remaining intermediate results are selected from the individual pipe operand memories and combined in pipe 1 to produce $X(2), X(3),\ldots,X(N-1)$.

In general, if $N = 4^v$, then $v - 1$ stages of the radix-4 FFT can be computed in the four parallel pipes. The last stage, the v^{th}, would then be computed using only one pipe.

A measure of the time required for an N point radix-4 FFT can be arrived at as follows: Let $N = 4^v$; thus, the FFT requires $\log_4(N)$ stages for the FFT computation. By looking at the basic radix-4 kernal, given in equations (11) through (14), and its flowchart in figure 3, it is seen that 3 complex multiplies and 12 complex adds are required per kernal. Also, note that (N/4)-kernals must be computed for each stage. If a unit of time is required for one complex multiply, then the multiply time for an $N = 4^v$ point radix-4 FFT is given by

$$\frac{3\,[\log_4(N) - 1](N/4)}{4} + 3(N/4) = (3N/4)\left[1 + \frac{\log_4(N) - 1}{4}\right], \qquad (49)$$

where

$$\frac{3\,[\log_4(N) - 1]}{4}\left(\frac{N}{4}\right)$$

is the time required to compute the first $v - 1$ stages in parallel and $3(N/4)$ is the multiplication time to compute the last stage. Similarly, the time for computing the complex adds is given by

$$12(N/4)\left[\frac{\log_4 (N) - 1}{4}\right] + 12(N/4) = 3N\left[1 + \frac{\log_4 (N) - 1}{4}\right] . \qquad (50)$$

By judicial use of the architecture within a given pipe, as shown in figure 2, one is able to reduce the addition time by "pipelining in time" the FFT computations and performing two parallel additions for each multiplication.[1] (The phrase "pipelining-in-time" means that three sets of latches are interposed in the path through a given arithmetic processor (one of four parallel pipes) so that after three clock periods the pipe is full, and addition and multiplication is overlapped.) In other words, the time for computing a given FFT is proportional to the time for multiplication, which is given by

$$\text{Time} = \frac{3N}{4}\left[1 + \frac{\log_4 (N) - 1}{4}\right], \qquad (51)$$

where one can see that as N becomes larger the multiplication time decreases to approximately one fourth of that when using a single pipe. Thereby, one FFT can be computed approximately four times faster by using the four parallel pipes.

Next, some practical considerations in implementing this radix-4 algorithm on a four-parallel-pipe architecture are given; i.e., the size of the FFT, size of the operand memory, mixed radix FFT's, speed formulas, bit reversal required to re-order input data, and inverse FFT's are discussed.

PRACTICAL CONSIDERATIONS

FFT SIZE

The size of the FFT determines how efficiently the four parallel pipes can be used. Generally, the more data points there are to transform, the less the effect of using only one pipe for the last stage has on overall computation time. The time required to transform several FFT sample sizes, using both a single- and a parallel-pipe FFT, are given in table 1, where it can be seen that the scheme approaches the theoretical reduction in speed of a factor of 4.

Table 1. Efficiency of a Four Parallel Pipe FFT
As a Function of Transform Size

Size of FFT	Time for Single-pipe FFT	Time for Parallel-Pipe FFT	Ratio Of Single- to Parallel-Pipe Scheme
256	768	336	2.3
1,024	3,840	1,536	2.5
4,096	18,432	6,912	2.7
16,384	86,016	27,648	3.1

Another characteristic of FFT size is that it must be a power of 4. Efficiency is increased by this restriction because the powers of W_N are located at the 90-deg quadrant points of the unit circle, and most multiplications reduce to multiplying by +1, +j, -1, or -j. However, this parallel processing method can be adapted to work on data sizes that are a power of 2.

MIXED-RADIX FFT's AND SPEED FORMULAS

Acoustic signal processing generally uses time-bandwidth products where 512, 1024, 2048, or 4096 points are to be transformed. Since 512 and 2048 are powers of 2 and not 4, a mixed-radix FFT can be employed to retain most of the FFT efficiency by writing 512 as $2 \cdot 4^4$ and 2048 as $2 \cdot 4^5$. For the 512 sample FFT, one radix-2 stage and four radix-4 stages must be performed to complete the computation. Similarly, for the 2048 computation, one radix-2 stage and five radix-4 stages are required.

When performing the mixed radix FFT on the SIMD architecture in figure 1, the radix-2 stage can be performed first or last. A slight increase in speed can be achieved when the radix-2 kernals are used in the last stage. The reason for this can be explained using a $32 = 2 \cdot 4^2$ data point FFT as an example.

The example FFT is performed by computing a radix-4 first stage, a radix-4 second stage, and finally a radix-2 third (last) stage (figure 5). First, the 32 original data points are divided equally among the 4 pipes so that each pipe has 8 points. Two radix-4 kernals are computed in each of

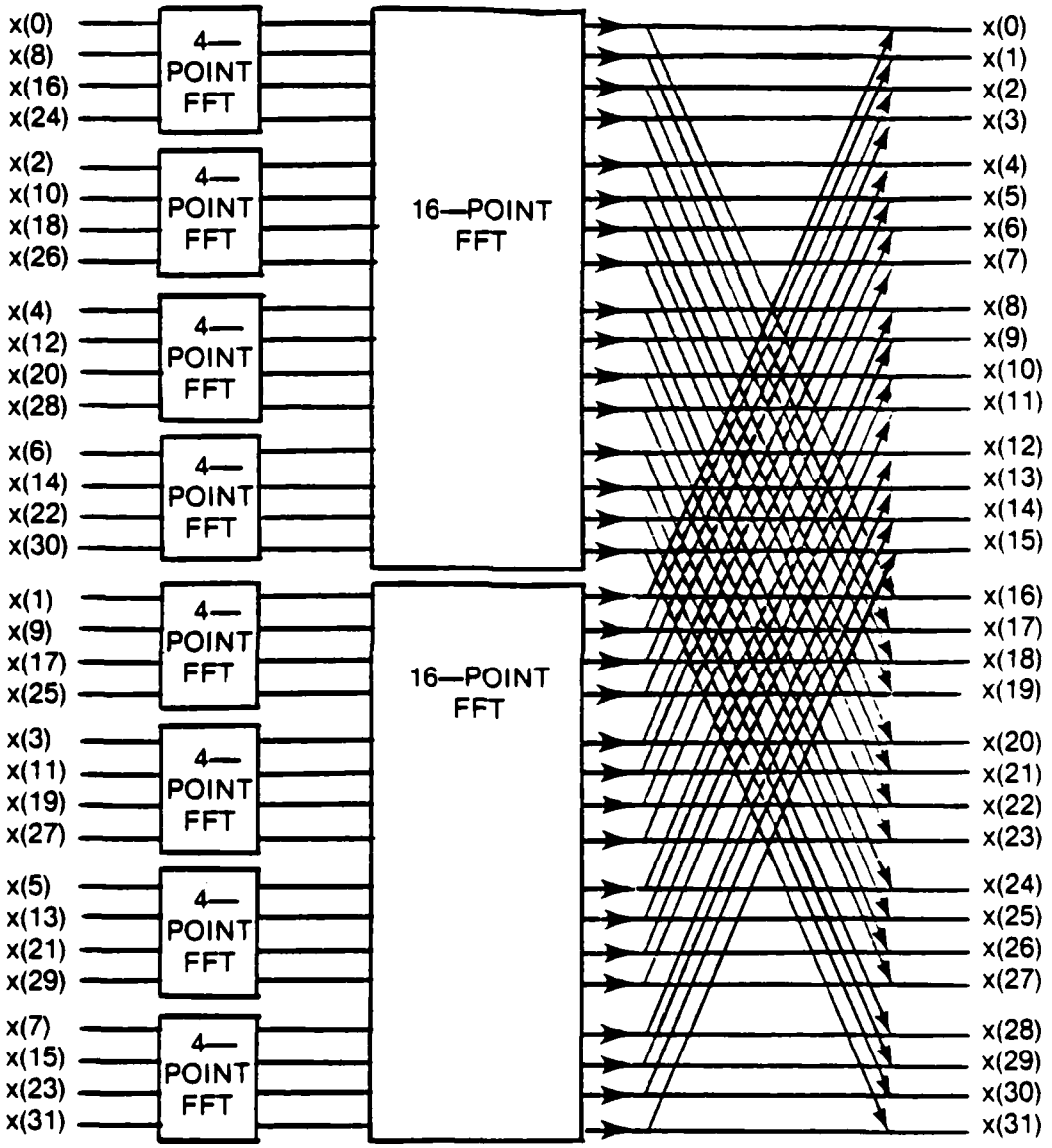


Figure 5. Partial Flowchart for a 32-Data-Point FFT Using Two Radix-4 Stages Followed by One Radix-2 Stage

the four parallel pipes in the first stage. Now, the second stage radix-4 computations are performed, but for this stage points from pipes one and two are crosscoupled. Because of the crosscoupling restriction, half of the second stage computations must be performed in only one pipe, say pipe one. Similarly, the points in pipes three and four must be combined into one pipe, say pipe three. Therefore, only two of the possible four pipes can be used in the second stage FFT computation.

The third stage computation presents the same problem as the second stage because data points from pipes 1 and 3 must be combined in only one pipe, say pipe one. Re-ordering the staging as 4, 4, and 2 prevents the four pipes from being used in parallel for the last two stages. Also, in the last stage, the radix-2 FFT will require multiplication by $32/2 = 16$ different powers of W_{32}. However, the first stage multiplication coefficients reduce to powers of J, which speeds up the computation time.

Alternatively, we can reorder the 32 data-point FFT stages, as shown in figure 6, so the radix-2 stage is performed first, followed by two radix-4 stages. The first stage radix-2 kernal computations are performed in the four parallel pipes, with each pipe computing four 2-point FFT's. The second stage radix-4 kernal computations can again be performed in the four parallel pipes because each computation requires only those points already in its own operand memory. The result is that four 8-point FFT's are performed in parallel in the second stage.

The third stage is the only computation that needs data points from the other pipes. Therefore, the third stage radix-4 kernal computations must be performed in only one pipe, say pipe one. This arrangement permits two stages of the FFT to be done using the four parallel pipes, and only the last stage must be done using one pipe. Also, the radix-2 coefficients needed in the first stage are +1 or -1, thereby eliminating multiplication by powers of W_{32}. In general, whenever radix-2 and -4 stages are required, the radix-2 stage should be performed last.

Formulas for the number of multiplications involved for mixed radixes are given next.

When N is equal to two times some power of four (i.e., $N = 2 \cdot 4^v$), the complex multiply time can be determined by allowing for $3(N/4)$ multiplies for eacn radix-4 stage and $N/2$ multiplies for each radix-2 stage. If any stage is performed in four parallel pipes the time to compute the complex multiplies should be divided by four. The only exception applies to any radix-2 or -4 first stage. The first stage doesn't contain any multiplies because the coefficients will be ± 1 for radix-2 and ± 1 or $\pm j$ for radix-4.

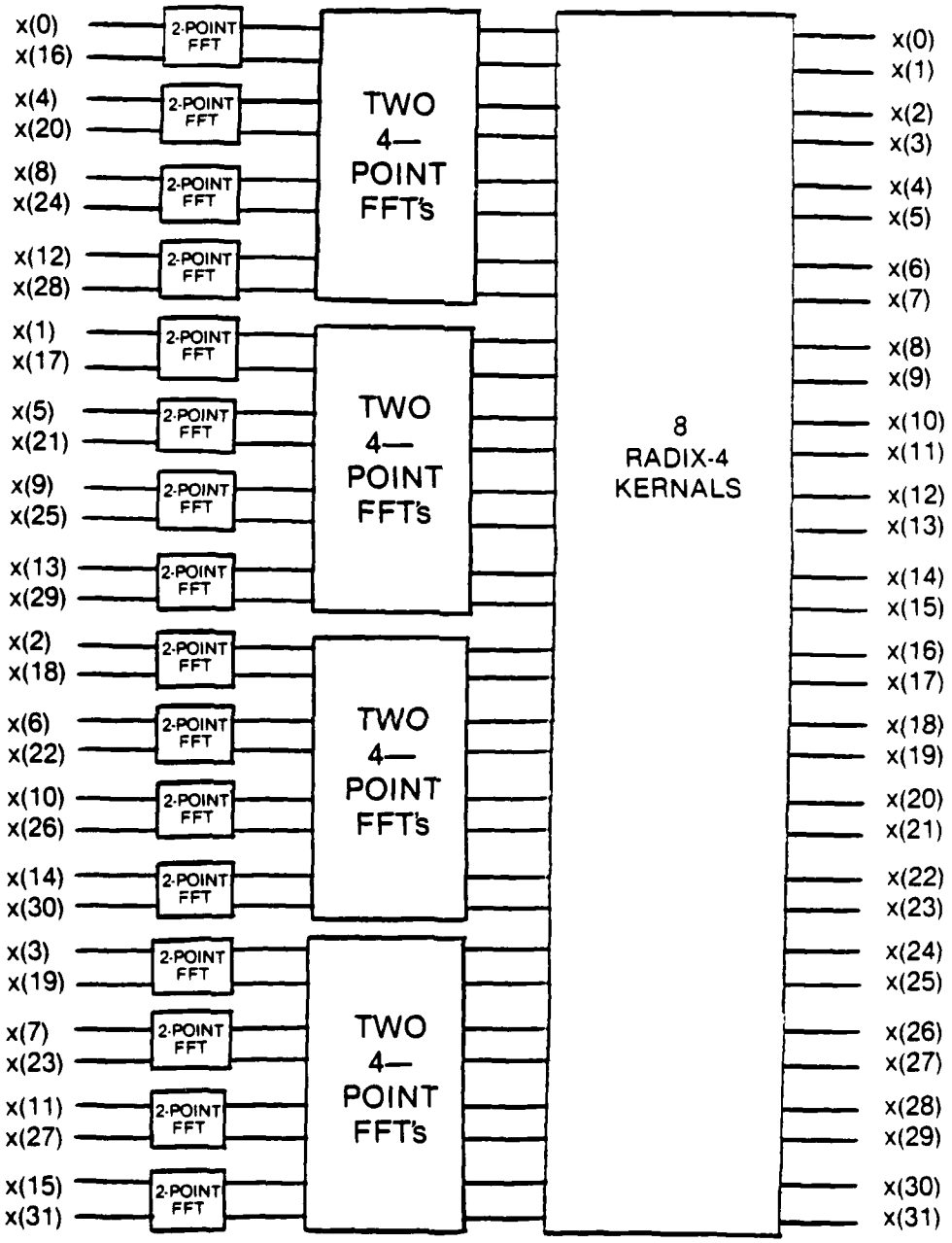


Figure 6. Partial Flowchart for a 32-Data-Point FFT Using One Radix-2 Stage Followed by Two Radix-4 Stages

A formula will now be derived for a mixed-radix FFT when radix-4 stages are performed first and the radix-2 stage is performed last. In this case, there are (V - 2) radix-4 stages with multiplying coefficients done in four pipes, one radix-4 stage done in two pipes, and one radix-2 stage done in one pipe. The total number of complex multiplies is determined by multiplying the number of stages by the number of parallel multiplies per stage. Thus, the

$$\text{NCM (number of complex multiplies)} = \frac{1}{4}\left[(V - 2)\frac{3N}{4}\right] + \frac{3N}{(2)4} + \frac{N}{2} , \quad (52)$$

which reduces to

$$NCM = \frac{N}{16}(3V + 8). \quad (53)$$

Similarly, when the radix-2 stage is performed first, there are (V - 1) radix-4 stages done in four parallel pipes, one radix-4 stage done in a single pipe, and a radix-2 stage done in four pipes, which has no multiplying coefficients other than ± 1. The total number of serial complex multiplies, considering the parallelism of the four pipes, is

$$NCM = \frac{1}{4}\left[(V - 1)\frac{3N}{4}\right] + \frac{3N}{4} , \quad (54)$$

which reduces to

$$NCM = \frac{N}{16}(3V + 9). \quad (55)$$

The difference between the NCM for the two formulas, above, is N/16. This difference is not a significant percentage of the number of complex multiplies. The stage order is really driven by the complexity involved in shuffling data between pipes since the first case requires shuffling for two stages.

OPERAND MEMORY

Another advantage of the radix-4 parallel pipe arrangement when computing the FFT is that three of the operand memories need hold only one quarter of the total sample size. However, the operand memory, which serves to combine the four individual results in the last stage, must be large enough to hold all N points. The computation scheme described is essentially an in-place scheme. Four temporary locations must be allocated for the four values, $G_0(m)$, $G_1(m)$, $G_2(m)$, and $G_3(m)$, and their values must be saved each time m is varied from 0 to (N/4) - 1 in each stage.

BIT REVERSAL TO RE-ORDER INPUT DATA

This section discusses why and how the input data sequence should be re-ordered before proceeding with the FFT computations. By re-ordering the input sequence,

- the FFT computations require less temporary storage,
- the resulting output sequence is in its natural order, and
- the address generation required to index to the proper data points and coefficients is simplified.

The discussion on re-ordering begins with the simple radix-2, then the radix-4, and finally the mixed radix, radix-2 and -4.

In the derivation of the radix-2 decimation-in-time algorithm,[2] the original input sequence is first sorted into even and then odd numbered data points. Next, the even numbered points are sorted into an even and an odd group. Similarly, the odd points from the first sort are also sorted into an even and an odd group. The sorting process, breaking each group into new even and odd groups, continues in each stage until there are only two points left in each group, and they are already sorted into even and odd because there are only two points in a set.

An example of this process is shown in table 2, where the original binary ordering, the ordering after the first sort, and the ordering after the second sort are listed. A third sort is not necessary.

Table 2. Successive Sorts of Input Data Points for a Radix-2 FFT

Original Binary Ordering $J_2J_1J_0$	First Sort $J_1J_0J_2$		Second Sort $J_0J_1J_2$	
000	000		000	Even
001	010		100	
010	100	Even	010	Odd
011	110		110	
100	001		001	
101	011		101	Even
110	101	Odd	011	
111	111		111	Odd

After the first sort all the even data points are in the first half of the sorted sequence, and the odd points are in the second half of the sequence. The same result is obtained by considering the least significant bit, J_0, in the original binary ordering to be the most significant bit, and the most significant two bits, J_2J_1, to be the least significant ones, i.e., $(J_0J_2J_1)_2$. The second sort in table 2 separates the even points from the first sort into even and odd points, and the odd points from the first sort are also separated into even and odd points. Observe that the same result is obtained by making the most significant bit, J_1, of the J_2J_1 bit pair the most significant bit to obtain the pair J_1J_2. As there are only two points in the resulting subsequence and they are already in even and then odd order. A third sort is not necessary.

After sorting, the location of an original data point can be determined by performing the well-known bit reversal procedure for radix-2 FFT's. That is, if $(J_2J_1J_0)_2$ is the binary representation of an original point, that point ends up in location $(J_0J_1J_2)_2$ following the sorting in the various stages required to compute the FFT.

The above process is generalized for radix-4 by sorting the original N-point sequence into four subsequences composed of the data points 4r, 4r + 1, 4r + 2, and 4r + 3, r = 0, 1, ..., (N/4) - 1. Then, each of the four resulting subsequences is independently sorted the same way; i.e., subsequence 4r is sorted into 4q, 4q + 1, 4q + 2, and 4q + 3, q = 0, 1, ..., (N/16) - 1.

The sorting process continues for $\log_4$ (N) stages. The original data can be re-ordered by bit reversing the base four digits or by pair-wise reversing the binary bit representations of the sample index. For example,

if $(J_3J_2J_1J_0)_2$ is the binary representation of the original sequence, the sorted sequence will be ordered as $(J_1J_0J_3J_2)_2$.

If the original set of data points contains $N = 4^3$ points, then three sorts are required, although the last sort need not be performed as there would be only 4 points in the resulting 16 subsequences to sort, and they are already automatically sorted. If the binary representation of the original sequence were $(J_5J_4J_3J_2J_1J_0)_2$, the points would end up in the location represented by $(J_1J_0J_3J_2J_5J_4)_2$ from which the radix-4 FFT would be performed.

For the mixed-radix FFT, the sorted ordering of the data can be obtained by reversing either bits or pairs of bits. As an example, suppose $32 = 2.4^2$ data points are to be transformed by performing a decimation-in-time FFT having a radix-2 stage first, followed by two radix-4 stages. Note that in the derivation of a radix-2/-4/-4 FFT, a radix-4 stage would be derived followed by another radix-4 stage and then a radix-2 stage. However, the radix-2 stage would be computed first, followed by the two radix-4 stages. Let $(J_4J_3J_2J_1J_0)_2$ be the binary representation of the original data sequence. The input data would be sorted by 4's, then by 4's again, and finally by 2's into even and odd points. The reversal procedure would be $(J_2J_1J_0J_4J_3)$, then $(J_0J_2J_1J_4J_3)_2$, where the entities J_4J_3 and J_2J_1 are treated as pairs that remain in fixed positions relative to each other.

The above re-ordering is necessary prior to any computation so that the FFT can be carried out "in place." If the flow diagram is drawn for each stage of the FFT, the data points on the same horizontal level transform into points on that same horizontal level. No temporary memory need be allocated except for two locations in the radix-2 kernal and four in the radix-4 kernal.

The net effect of performing the radix-2 kernal (butterfly) or radix-4 kernal (dragonfly) on re-ordered data is that the resulting frequency data points of the FFT end up in their correct order and the intermediate memory locations can be overwritten during each stage of the FFT computation.

The digit-reversal procedure can take place in hardware simply by building a binary counter that counts from the left (most significant bit) or by transposing the wires from a normal counter, as shown in figure 7. This counter is used to index into the original data set to obtain the properly re-ordered data points.

The digit reversal counter can be generalized to any radix (not necessarily powers of 2) by using an adder circuit that causes a register to step through a bit reversed sequence of any radix, as shown in figure 8. Here the adder is binary, but the adder increments are chosen to simulate a

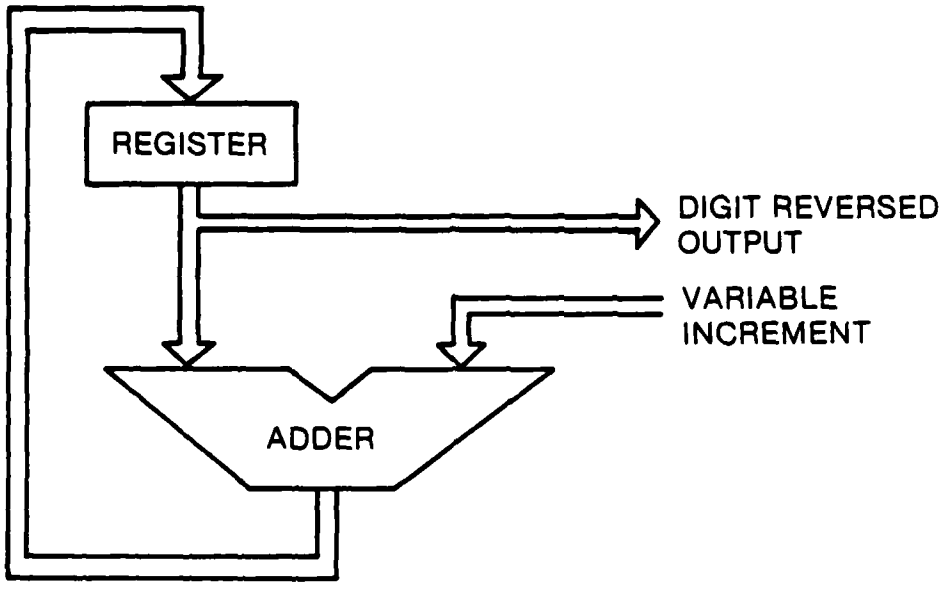


Figure 7. Digit Reversal for a Radix 2/-4/-4 FFT

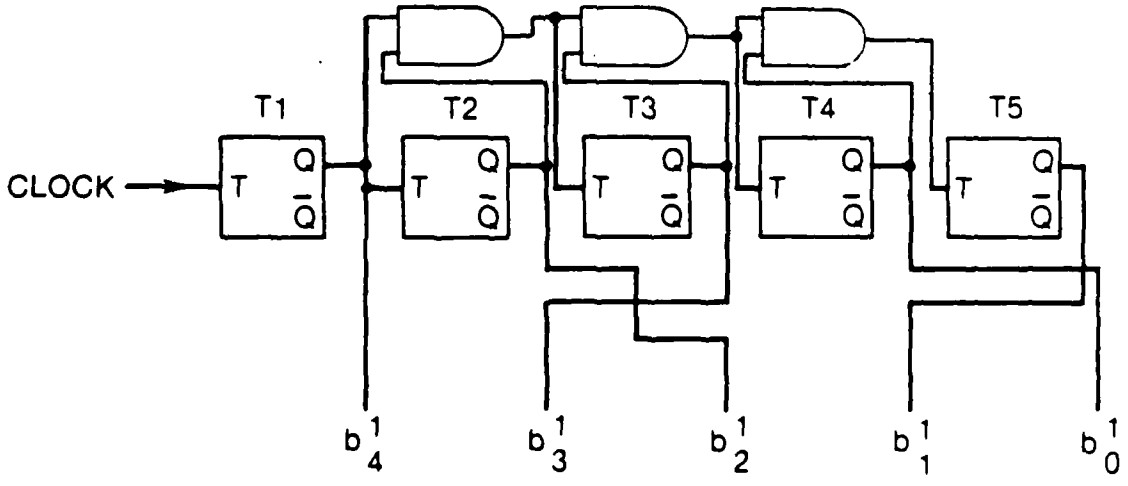


Figure 8. Generalized Bit Reversal Hardware

counter of any mixed radix desired, like 3, 5, 4, where the number of data points is factorable as N = (3)(5)(4).

Sample sizes that are not powers of 2 are generally not necessary[3] and create awkward hardware.

INVERSE DFT's

The exact same parallel processing FFT algorithm and hardware discussed here can be used to compute the inverse DFT, which is given by

$$x(k) = \frac{1}{N} \sum_{i=0}^{N-1} X(i) e^{\frac{2\pi j}{N} ik} . \tag{56}$$

The inverse DFT can be obtained using the forward transform coefficients and by

1. conjugating X(i) to obtain $\hat{X}(i)$,
2. performing the forward transform to obtain

$$\hat{x}(k) = \sum_{i=0}^{N-1} \overline{X(i)} e^{\frac{-2\pi j}{N} ik}, \tag{57}$$

and

3. dividing $\hat{x}(k)$ by N and conjugating the result to obtain

$$x(k) = \frac{\overline{\hat{x}(k)}}{N}. \tag{58}$$

In other words, to obtain the inverse DFT simply conjugate the input data points, take forward transform, divide by N, and conjugate the result.

CONCLUSION

A method was devised to efficiently utilize a loosely interconnected, SIMD four parallel pipe arithmetic processor with a single common coefficient memory to compute the FFT. Previously users of this architecture employed the four pipes to execute four independent FFT's on four independent data sets, thereby limiting its use.

The method described here uses the four parallel pipes to compute a single FFT approximately four times faster than that possible in a single pipe arithmetic processor. A decimation-in-time radix-4 FFT algorithm that is adapted to allow the computation to proceed on the four parallel pipes is derived. Basically the original N-point data set is separated into four N/4 point data sets, and each of these points is partially transformed in each of the four pipes. The four pipes operate in parallel until the last stage, at which time one pipe must be used to finish the computation.

Formulas showing the addition and multiplication time needed to compute an N-point radix-4 FFT are derived and described here. A comparison of computing an FFT in a single pipe versus four parallel pipes is made to show that a fourfold decrease in execution time is possible with the four parallel pipes. Also, a method is shown for computing an FFT whose length is not a power of 4. An example of mixing radix-2 and -4 stages shows that the radix-2 stage can be performed first or last. Formulas for the execution time of the mixed radix FFT computation are given.

It is advantageous to re-order the input data in the described FFT scheme because "in-place-computations" can save memory and also the output appears in the correct order. In addition, for an N-point transform, three operand memories need be N/4 words, and one operand memory must be N words in size.

A generalized method for re-ordering the data is presented that allows data to be prescrambled for radix-2, -4, or mixed-radixes. This re-ordering method uses an arrangement of a binary counter that counts from the most significant rather than from the least significant bit. The output of this counter can be used to index naturally ordered input data points and locate the appropriate scrambled data points for the radix-2 (butterfly) or the radix-4 (dragonfly) FFT computations.

Finally a method is shown that allows the same computations to produce the inverse DFT. The computation method described here can be generalized to any radix FFT and to any number of parallel processing data pipes in an arithmetic processor.

REFERENCES

1. R. R. Shively, "Architecture of a Programmable Digital Signal Processor," IEEE Transactions on Computers, vol. C-31, no. 1, January 1982, pp. 16-22.

2. A. V. Oppenheim and Ronald W. Schafer, Digital Signal Processing, Prentice-Hall Inc., Englewood Cliffs, NJ 1975.

3. G. D. Bergland, "A Guided Tour of the Fast Fourier Transform," IEEE Spectrum, vol. 6, July 1969, pp. 41-52.

BIBLIOGRAPHY

1. Bergland, G. D., and D. E. Wilson, "A Fast Algorithm for a Global, Highly Parallel Processor," IEEE Transactions on Audio and Electroacoustics, vol. Au-17, no. 2, June 1969, pp. 125-127.

2. Cooley J. W. and J. W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series," Mathematics of Computation, 1965, vol. 19, no. 90, 1965, pp. 297-301

3. Cochran, W. T., J. W. Cooley, D. L. Favin, H. D. Helms, R. A. Kaenel, W. W. Lang, G. C. Maling, D. E. Nelson, C. H. Radar, and P. D. Welch, "What is the Fast Fourier Transform," IEEE Transactions on Audio and Electroacoustics, vol. Au-15, June 1967, pp 44-45.

4. Morris, L. R., "Time Efficient Radix-4 Fast Fourier Transform," in Programs For Digital Signal Processing, edited by the Digital Signal Processing Committee for the IEEE Acoustics, Speech and Signal Processing Society, IEEE Press, 1979, pp. 1.8-1 through 1.8-11

5. Pease, M. C., "An Adaptation of the Fast Fourier Transform for Parallel Processing," Journal of the Association for Computing Machinery, vol. 15, no. 2, April 1968, pp. 305-316.

6. Singleton, R. C., "An Algorithm for Computing the Mixed Radix Fast Fourier Transform," IEEE Transactions on Audio and Electroacoustics, vol. Au-17, no. 2, June 1969, pp. 93-103.

INITIAL DISTRIBUTION LIST

Addressee		No. of Copies
NRL	(Codes 5150 (R. Stevens), 5155 (G. Ross))	2
NAVSEASYSCOM	(PMS-412 (G. Melcher, CMDR W. Hatcher, H. Taylor, I. Hall, CAPT Goodman))	5
NAVAIRDEVCEN	(Codes 5013 (A. Gropp), 5021 (L. Hart, R. Peck (2), 3032 (T. Stover))	5
NOSC	Codes 551 (C. Morrin), 552 (J. Anderson), 723 (J. Hall, A. White), 724 (D. Gurwell, G. Ottinger), 7205 (J. E. Watring, C. Whitson), 951 (J. Dickinson))	9
NAVAL WEAPONS SUPPORT CENTER (Code 6334 (S. Oliphant))		1
DTIC, Alexandria		12
ANALYTIC DISCIPLINES, INC. (ADI) (F. Bloch)		1
TRW, INC. (TRW)	(A. Toutsi, (TRW-WI 5410))	1
	(J. Ling, (TRW-WI 4667))	1
AT&T TECHNOLOGIES, INC. Burlington, NC 27215	(E. Rutherford, EMSP Engr. Manager)	1
	(C. Farlow, EMSP Firmware Development)	1
AT&T TECHNOLOGIES, INC. Whippany, NJ 07981	(L. MacDougal, EMS Development Environment Software)	1
	(R. Shively, EMSP Processor Design Group)	1

END

FILMED

8-85

DTIC