

COMPUTATION OF THE DISCRETE FOURIER TRANSFORM - PART 3

1. Lecture 20 - 45 minutes

Computational Considerations

① Inverse DFT

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

$$W_N = e^{-j \frac{2\pi}{N}}$$

DFT

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

② Bit Reversal

a.

$N = P_1 \cdot P_2 \cdots P_V$
 $q_1 = P_1$

$N = P_1 \cdot q_1$

Subsequence: every P_1^{th} point

P_1 sequences of length q_1

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

$$= \sum_{r=0}^{q_1-1} X(P_1 r) W_N^{P_1 r k}$$

$$+ \sum_{r=0}^{q_1-1} X(P_1 r + 1) W_N^{(P_1 r + 1) k}$$

$$+ \dots$$

$$X(k) = \sum_{l=0}^{P_1-1} W_N^{lk} \sum_{r=0}^{q_1-1} X(P_1 r + l) W_N^{P_1 r k}$$

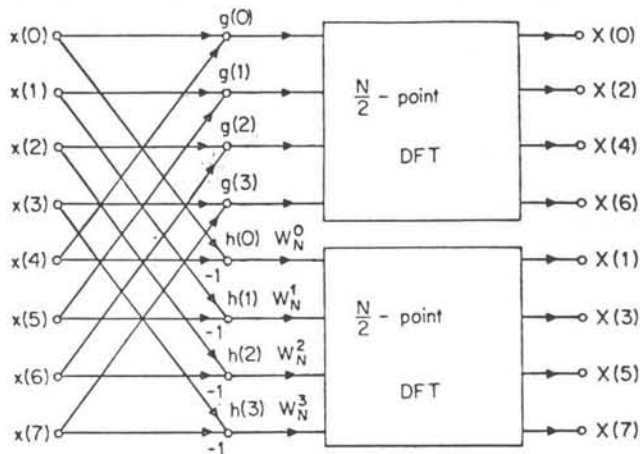
$W_N^{P_1 r k} = e^{-j \frac{2\pi}{N} P_1 r k} = e^{-j \frac{2\pi}{P_1 \cdot q_1} P_1 r k} = e^{-j \frac{2\pi}{q_1} r k} = W_{q_1}^{rk}$

$$X(k) = \sum_{l=0}^{P_1-1} W_N^{lk} \underbrace{\sum_{r=0}^{q_1-1} X(P_1 r + l) W_{q_1}^{rk}}_{q_1\text{-point DFT}}$$

$q_1 = P_2 \cdot P_3 \cdots P_V$

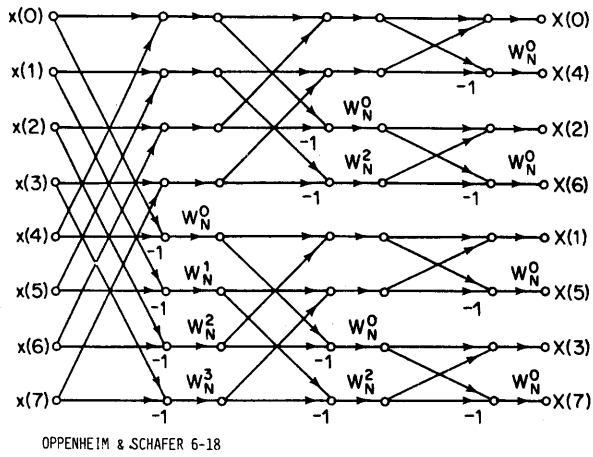
MADS \Rightarrow
 $N [P_1 + P_2 + \dots + P_V - V]$

b.



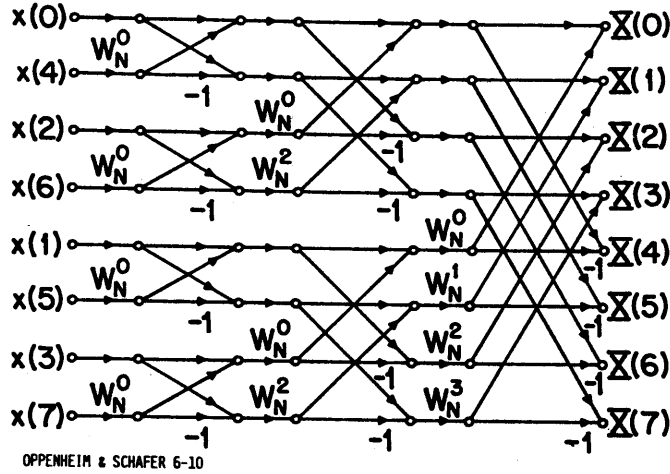
Computation of even and odd numbered DFT values.

c.



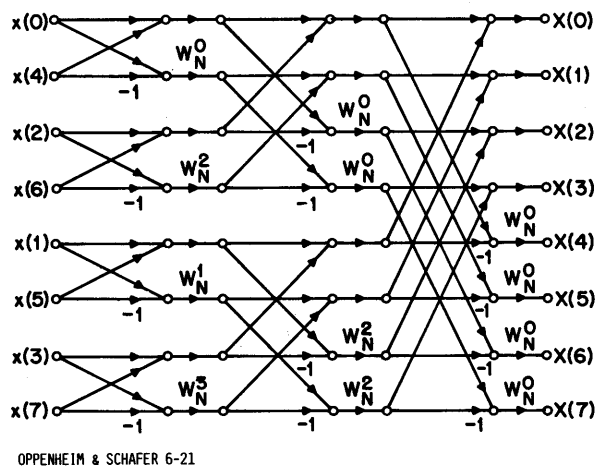
Flow-graph of complete decimation-in-frequency decomposition of an eight point DFT computation.

d.



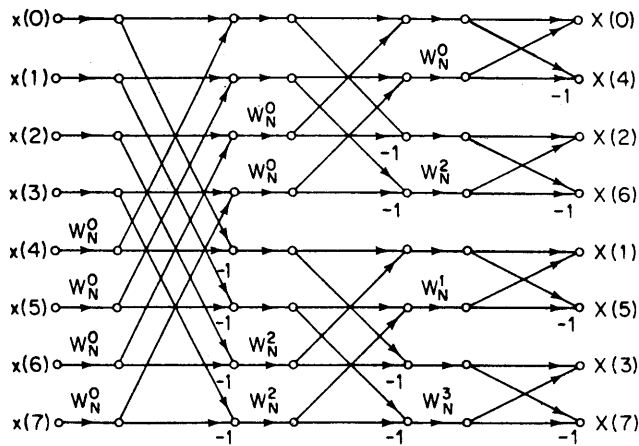
FFT flow-graph for decimation-in-time algorithm.

e.



Rearrangement of the decimation-in-frequency flow-graph d. The input is now in bit-reversed order and the output is in normal order.

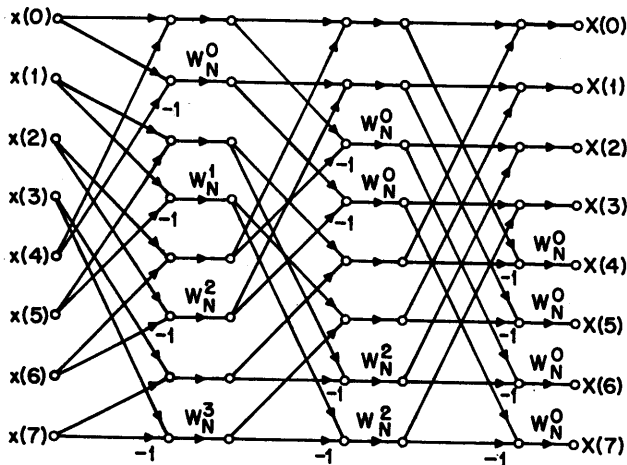
f.



OPPENHEIM & SCHAFER 6-12

Rearrangement of e so that the input is in normal order and output in bit-reversed order.

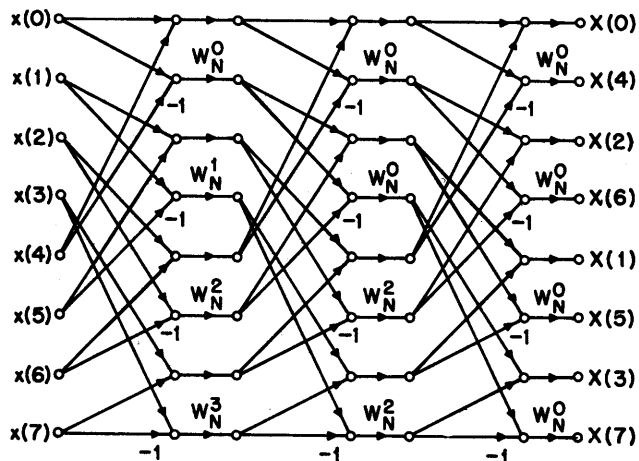
g.



OPPENHEIM & SCHAFER 6-22

Rearrangement of d so that both input and output are in normal order.

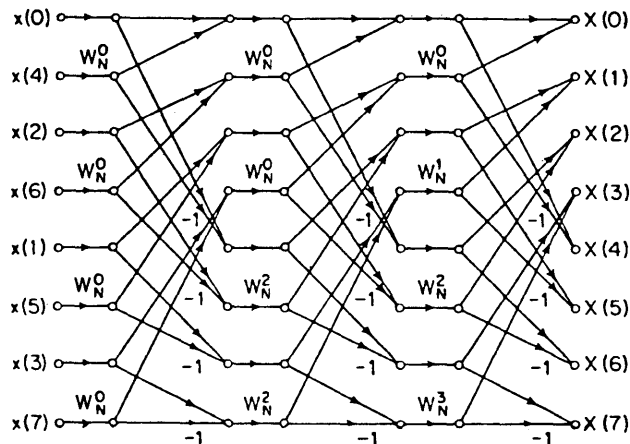
h.



OPPENHEIM & SCHAFER 6-23

Rearrangement of d so that geometry is identical in each stage.

i.



OPPENHEIM & SCHAFFER 6-14

Decimation-in-time flow-graph for which the geometry is identical for each stage.

j.

Correction: On (Fig. a.) W_N is written as $e^{j\frac{2\pi}{N}}$ it should be $e^{-j\frac{2\pi}{N}}$.

2. Comments

In the previous lecture we introduced a class of FFT algorithms referred to as "decimation-in-frequency." This class of algorithms is developed on the basis of successive subdivisions of the output, as compared with the "decimation-in-time" algorithms which are based on successive subdivisions of the input. While the flowgraph for decimation-in-frequency as initially derived represents an in-place computation with the input in normal order and the output in bit-reversed order, there are a variety of rearrangements of the flowgraph just as there were for the "decimation-in-time" algorithms. The various forms of the "decimation-in-frequency" flowgraphs are related to the decimation-in-time flowgraph through the transposition theorem. The choice between the various forms of the FFT algorithm is generally based on such considerations as the importance of in-place computation, whether it is more convenient to require bit-reversal at the input or output, order in which coefficients are stored, etc. For example if we intend to follow a DFT by an inverse DFT it is generally preferable to begin with an algorithm for which the input is in normal order and the output is in bit-reversed order. If for the inverse transform a form of the algorithm is used which requires bit-reversed input data and generates the output in normal order, it is never necessary to rearrange the order of the data.

Throughout the discussion of the FFT algorithms we have concentrated on "radix-2" algorithms, i.e. we have assumed that N is a power of two. More generally, efficient algorithms for the computation of the DFT can be utilized when N is decomposable as a product of factors. We

conclude this lecture with a brief introduction to these more general classes of FFT algorithms.

3. Reading

Text: Section 9.4 (page 599), and section 9.5.

4. Problems

Problem 20.1

The basic radix-2 FFT algorithms based on decimation-in-time are indicated in the text, Figures 9.10, 9.14, 9.15, 9.16. The basic radix-2 FFT algorithms based on decimation-in-frequency are indicated in the text Figures 9.20, 9.22, 9.23, 9.24. For each of these eight flow-graphs indicate whether or not each of the following following properties is true or not:

- (1) Represents an in-place computation
- (2) Input is in normal order
- (3) Output is in normal order
- (4) Coefficients should be stored in bit-reversed order
- (5) Accessing of the data is identical for every array

Problem 20.2

We wish to implement a filter on a small computer by evaluating the DFT of input data, multiplying by the DFT of the unit-sample response and then computing the inverse DFT. The length of input data is a power of two but is sufficiently large that it cannot be stored in random-access memory. Consequently we wish to choose an algorithm which permits the data to be stored in and accessed from disk memory.

- (a) How would you modify any one of the radix-2 algorithms discussed in the text so that it computes the inverse DFT rather than the DFT?
- (b) Which of the eight radix-2 algorithms listed in problem 20.1 would it be most convenient to use for the computation of the DFT?
- (c) Which of the eight radix-2 algorithms listed in problem 20.1 would you modify according to part (a) and use for the inverse DFT?
- (d) In implementing the transform and inverse transform, let us assume that the disk is divided into four tracks which we'll refer to as A, B, C, and D. The input data will initially be stored on

tracks C and D. With the algorithm which you chose in (b), how should the input data be divided between tracks A and B?

(e) After completing the computation of the DFT, on what disk tracks and in what order is the result stored?

(f) Assume that the DFT values have been multiplied by the DFT of the unity sample response and the product is stored in the same order as were the DFT values in (c). Do these values have to be rearranged in any way before utilizing the algorithm chosen in (c) for the inverse DFT?

Problem 20.3

In rearranging data from normal order to bit-reversed order, a common procedure is to program a counter which counts sequentially in normal order and a second counter which counts sequentially in bit-reversed order. On most computers, of course, a normal counter usually corresponds to an index register which is incremented by unity. Most computers do not have a bit reversed counter but they are easy to implement. One possible flow-chart to implement a bit-reversed counter is show in Figure P20.3-1.

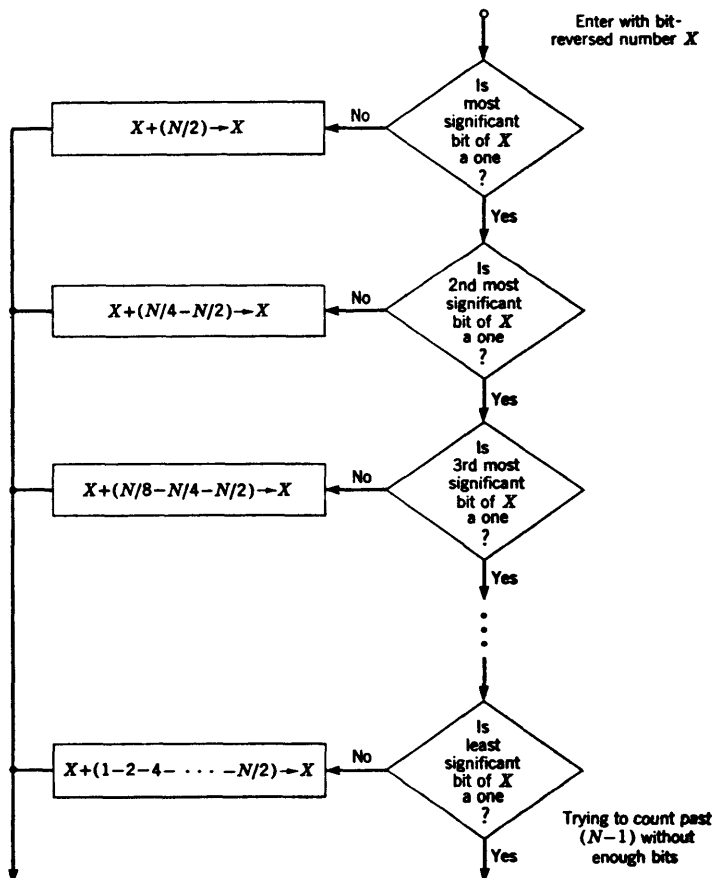


Figure P20.3-1

Let us assume that we have a normal counter (counter A) and a bit-reversed counter (counter B). In Figure P20.3-2 is shown a flow-chart intended to sort data from normal order to bit-reversed order. Determine whether a program implementing this flow-chart will sort the data as desired. If not, insert the necessary corrections into the flow-chart.

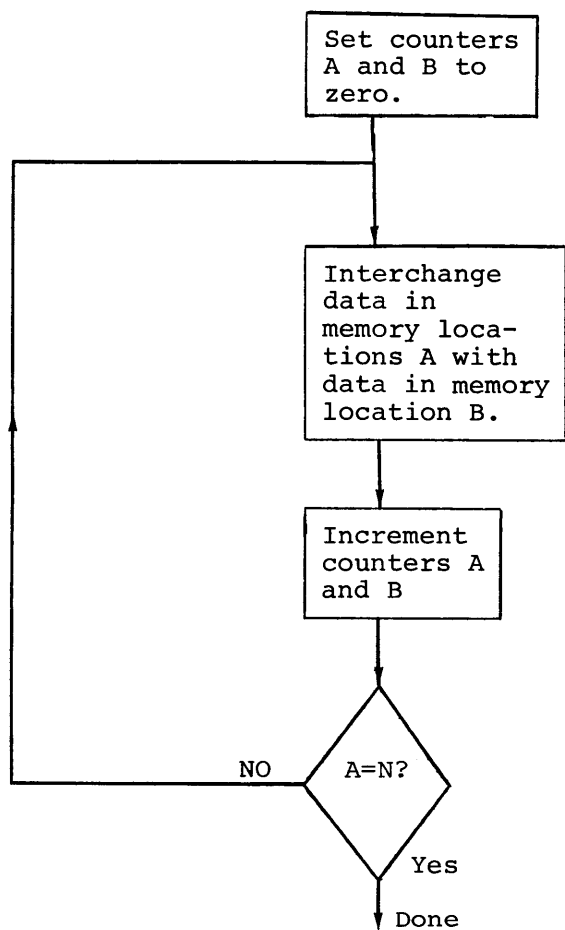


Figure P20.3-2

Problem 20.4

Draw the flow diagram for a nine-point (i.e., 3 x 3) decimation-in-time FFT algorithm.

Problem 20.5*

The FORTRAN program shown in Figure P20.5-1 is an implementation of the decimation-in-frequency algorithm depicted in Figure 6.18 of the text. The program evaluates the DFT:

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-j(2\pi/N)kn}, \quad k = 0, 1, \dots, N-1$$

```

SUBROUTINE FFT(X,M)                                0001
COMPLEX X(1024), U,W,T                            0002
N = 2**M                                           0003
PI = 3.14159265358979                             0004
DO 20 L = 1,M                                     0005
LE = 2**(M + 1 - L)                               0006
LE1 = LE/2                                         0007
U = (1.0,0.0)                                      0008
W = CMPLX(COS(PI/FLOAT(LE1)), -SIN(PI/FLOAT(LE1))) 0009
DO 20 J = 1, LE1                                  0010
DO 10 I = J,N,LE                                  0011
IP = I+LE1                                         0012
T = X(I) + X(IP)                                  0013
X(IP) = (X(I) - X(IP))*U                          0014
10 X(I) = T                                        0015
20 U = U*W                                         0016
NV2 = N/2                                          0017
NM1 = N - 1                                       0018
J = 1                                              0019
DO 30 I = 1,NM1                                    0020
IF(I.GE.J) GO TO 25                               0021
T = X(J)                                           0022
X(J) = X(I)                                        0023
X(I) = T                                           0024
25 K = NV2                                         0025
26 IF(K.GE.J) GO TO 30                             0026
J = J - K                                          0027
K = K/2                                            0028
GO TO 26                                           0029
30 J = J + K                                        0030
RETURN                                             0031
END                                                0032
```

Figure P20.5-1

In the subroutine FFT(X,M) X is a complex array of dimension N that contains initially the input sequence $x(n)$ and finally contains the transform $X(k)$. The quantity M is an integer, $M = \log_2 N$.

This program is a straightforward implementation of the flow-graph of Fig. 9.20 of the text. The program is very elegant but not as efficient as it could be. Greater efficiency can be obtained at the cost of a more complex program.

A significant increase in efficiency is suggested by noting that in the last stage of the flow graph in Fig. 6.18, the complex multipliers are all unity. Thus if the last stage is implemented separately, we can eliminate $N/2$ complex multiplications

- (a) What is the percentage reduction in multiplications that results?
- (b) Modify the program to implement this saving in multiplications.
- (c) Many small computers have FORTRAN compilers without the capability of complex arithmetic. Modify the given program so that only real operations are involved. That is, using the present subroutine as a guide, write a subroutine

FFT(XR,XI,M)

where XR and XI are real arrays of dimension N which initially contain the real part and the imaginary part of the input and finally the real and imaginary parts of the transform.

SOME CONCLUDING REMARKS

With lesson 20 we conclude this introductory set of lessons on digital signal processing. It has often been said that the purpose of a course is to uncover rather than to cover a subject and that description applies particularly in this case. Throughout the lectures I have tried to concentrate on the basic fundamentals of digital signal processing to provide a firm background for proceeding to applications and advanced topics. As you know, we have only covered the first six chapters in the text and omitted the more advanced topics from some of those. I would like to encourage you to take the time to look over the parts of those six chapters which were not assigned reading. I would also like to encourage you to continue on through the text; in chapters 10 and 11, in particular, you will find frequent reference to applications.

With the first six chapters as background, I feel that you will find much of the technical literature in the field of digital signal processing to be interesting and understandable. As I mentioned in the introductory lecture, this material has important applications in a wide variety of areas, and I would like to encourage you to explore some of these applications and also consider whether some of the techniques that we have discussed have application to your own area of interest. While we have only been able to present the fundamentals, I hope that this set of lectures will serve to open the door for you to an exciting, dynamic, and important field.

MIT OpenCourseWare
<http://ocw.mit.edu>

Resource: Digital Signal Processing
Prof. Alan V. Oppenheim

The following may not correspond to a particular course on MIT OpenCourseWare, but has been provided by the author as an individual learning resource.

For information about citing these materials or our Terms of Use, visit: <http://ocw.mit.edu/terms>.