Web Appendix M - The Fast Fourier Transform

M.1 Introduction

The forward DFT is defined by

$$X[k] = \sum_{n=0}^{N_F-1} x[n] e^{-j2\pi nk/N_F} .$$

A straightforward way of computing the DFT would be by the following algorithm (written in MATLAB) which directly implements the operations indicated above.

```
.
.
.
%      (Acquire the input data in an array, x, with NF elements.)
.
.
.
%
%      Initialize the DFT array to a column vector of zeros.
%
X = zeros(NF,1) ;
%
%      Compute the Xn's in a nested, double for loop.
%
for k = 0:NF-1
       for n = 0:NF-1
              X(k+1) = X(k+1)+x(n+1)*exp(-j*2*pi*n*k/NF) ;
       end
end
.
.
.
```

(By the way, one should never actually write this program in MATLAB because the DFT is already built in to MATLAB as an intrinsic function called `fft`.)

The computation of a DFT using this algorithm requires N^2 complex multiply-add operations. Therefore the number of computations increases as the square of the number of elements in the input vector which is being transformed. In 1965 James Cooley and John Tukey popularized an algorithm which is much more efficient in computing time for large input arrays whose length is an integer power of 2. This algorithm for computing the DFT is the so-called *fast Fourier transform* or just FFT.

James W. Cooley (1926-)

John Wilder Tukey (1915-2000)

James Cooley received his Ph.D. in applied mathematics from Columbia University in 1961. Cooley was a pioneer in the digital signal processing field, having developed, with John Tukey, the fast Fourier transform. He developed the FFT through mathematical theory and applications, and has helped make it more widely available by devising algorithms for scientific and engineering applications.

John Tukey received his Ph.D. from Princeton in mathematics in 1939. He worked at Bell Labs from 1945 to 1970. He developed new techniques in data analysis. He developed graphing and plotting methods that now appear in standard statistics texts. He wrote many publications on time series analysis and other aspects of digital signal processing that are now very important in engineering and science. He developed, along with James Cooley the fast Fourier transform algorithm. He is credited with having coined , as a contraction of "binary digit", the word "bit", the smallest unit of information used by a computer.

M.2 The Decimation-in-Frequency FFT

There are two commonly-used FFT algorithms called the *decimation-in-time* (*DIT)* algorithm and the *decimation-in-frequency (DIF)* algorithm. We will consider first the DIF algorithm. We will see in this development that the FFT algorithm is a good example of the problem-solving motto "divide and conquer" which is so effective in LTI system analysis. At every stage of calculation in the FFT algorithm we defer the actual DFT calculation by splitting the data into two halves and finding the DFT of the halves separately. We do this until we get down to multiple 2-point DFT's which cannot be further subdivided. The DIF FFT algorithm can also be considered an example of "top-down" problem solving in which we attack the overall problem and, as necessary, assign parts of the analysis as sub-problems until we get to sub-problems that are simple enough to solve directly.

The DFT formula for an N -point transform is

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j2\pi nk/N} . \qquad \text{(M.1)}$$

Similarly to what we did in the development of the DTFS in Chapter 4, it is convenient to use the notation $W_N = e^{-j2\pi/N}$. (In chapter 4 it was defined as $W_N = e^{j2\pi/N}$. Because of the minus sign in the exponent of *e* in (M.1) it is more convenient here to define it with the sign of the exponent changed.) Then

$$\mathrm{X}\left[k\right] = \sum_{n=0}^{N-1} \mathrm{x}\left[n\right] W_N^{nk} \quad . \tag{M.2}$$

We will restrict this development to $N = 2^\gamma$, with γ being a positive integer because only in that case is the FFT algorithm optimally efficient. To compute all *N* values of $\mathrm{X}\left[k\right]$, $0 \le k < N$ directly from (M.2) requires $N\left(N-1\right)$ complex additions and N^2 complex multiplications. (These numbers are based on a simple calculation treating every calculation the same. Some of the multiplications are by W_N^0 which is obviously 1. In those cases we could reduce the number of multiplications but, for large *N* which is the most practical case, the number of multiplications would not change much, would be more complicated to calculate and writing a computer program to take advantage of the elimination of these multiplications would complicate the program. So we will stick with this, admittedly oversimplified, estimate.)

The FFT algorithm is very intricate and recursive so it helps to start with a small example and then to expand the concepts to the general case. We will start with the smallest possible DFT $N = 2$, γ=1, $W_N = e^{-j2\pi/2} = -1$.

$$\mathrm{X}\left[k\right] = \sum_{n_0=0}^{1} \mathrm{x}\left[n\right] W_2^{nk}$$

and

$$\begin{bmatrix} \mathrm{X}\left[0\right] \\ \mathrm{X}\left[1\right] \end{bmatrix} = \begin{bmatrix} W_2^0 & W_2^0 \\ W_2^0 & W_2^1 \end{bmatrix} \begin{bmatrix} \mathrm{x}\left[0\right] \\ \mathrm{x}\left[1\right] \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} \mathrm{x}\left[0\right] \\ \mathrm{x}\left[1\right] \end{bmatrix}$$

We can realize a 2-point DFT as a 2-input-2-output system (Figure M-1).

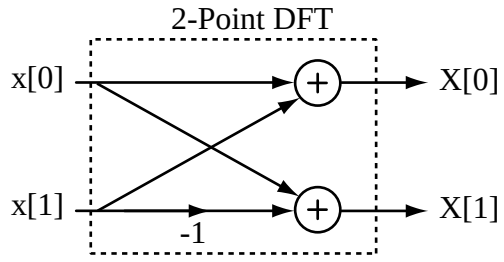

Figure M-1 A 2-point DFT

Now we will step up to the next level, a 4-point DFT.

$$X[k] = \sum_{n=0}^{3} x[n] W_4^{nk} = \sum_{n=0}^{1} x[n] W_4^{nk} + \sum_{n=2}^{3} x[n] W_4^{nk}$$

$$X[k] = \sum_{n=0}^{1} x[n] W_4^{nk} + \sum_{n=0}^{1} x[n+2] W_4^{(n+2)k}$$

$$X[k] = \sum_{n=0}^{1} \left(x[n] + x[n+2] W_4^{2k} \right) W_4^{nk}$$

$$W_4^{2k} = e^{-j2\pi(2k)/4} = e^{-j\pi k} = (-1)^k$$

$$X[k] = \sum_{n=0}^{1} \left(x[n] + (-1)^k x[n+2] \right) W_4^{nk}$$

Let q be any integer. Then

$$X[2q] = \sum_{n=0}^{1} \left(x[n] + \underbrace{(-1)^{2q}}_{=1} x[n+2] \right) W_4^{2nq} = \sum_{n=0}^{1} \left(x[n] + x[n+2] \right) W_4^{2nq}$$

$$W_4^{2nq} = {}^{-j2\pi(2nq)/4} = e^{-j2\pi nq/2} = W_2^{nq}$$

$$X[2q] = \sum_{n=0}^{1} \left(x[n] + x[n+2] \right) W_2^{nq} = \text{2-point DFT of } \left(x[n] + x[n+2] \right)$$

Similarly,

$$X[2q+1] = \text{2-point DFT of } \left(x[n] - x[n+2] \right) W_4^{n}$$

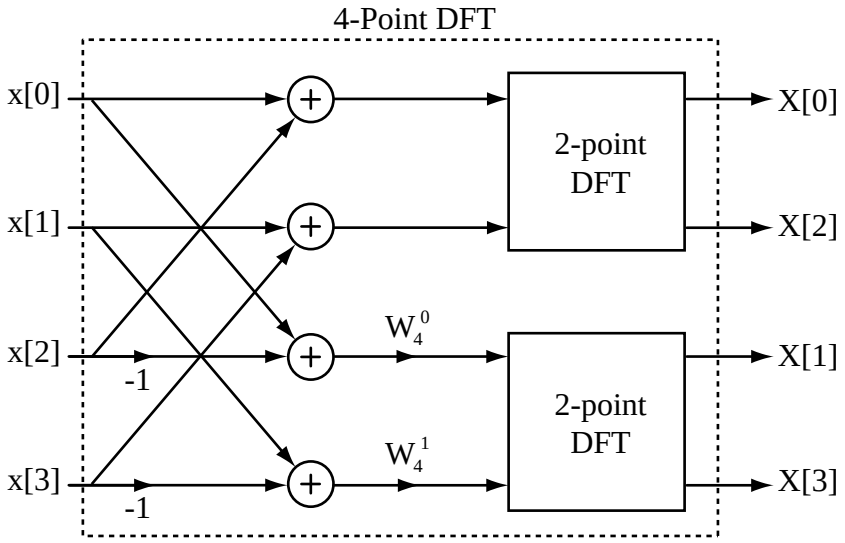

Figure M-2 A 4-point DIF DFT computed using two 2-point DFT's

Now consider the general case, an *N*-point DFT.

$$X[k]=\sum_{n=0}^{N-1}x[n]W_N^{nk}=\sum_{n=0}^{N/2-1}x[n]W_N^{nk}+\sum_{n=N/2}^{N-1}x[n]W_N^{nk}$$

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

$$X[k]=\sum_{n=0}^{N/2-1}\left(x[n]+x[n+N/2]W_N^{Nk/2}\right)W_N^{nk}$$

$$W_N^{Nk/2}=e^{-j2\pi(Nk/2)/N}=e^{-j\pi k}=(-1)^k$$

$$X[k]=\sum_{n=0}^{N/2-1}\left(x[n]+(-1)^k x[n+N/2]\right)W_N^{nk}$$

Let *q* be any integer. Then

$$X[2q]=\sum_{n=0}^{N/2-1}\left(x[n]+\underbrace{(-1)^{2q}}_{=1}x[n+N/2]\right)W_N^{2nq}$$

$$=\sum_{n=0}^{N/2-1}\left(x[n]+x[n+N/2]\right)W_N^{2nq}$$

$$W_N^{2nq} = {}^{-j2\pi(2nq)/N} = e^{-j2\pi nq/(N/2)} = W_{N/2}^{nq}$$

$$\mathrm{X}[2q] = \sum_{n=0}^{N/2-1} \left(\mathrm{x}[n] + \mathrm{x}[n+N/2]\right) W_{N/2}^{nq}$$
$$= (N/2)\text{-point DFT of } \left(\mathrm{x}[n] + \mathrm{x}[n+N/2]\right)$$

Similarly,

$$\mathrm{X}[2q+1] = (N/2)\text{-point DFT of } \left(\mathrm{x}[n] - \mathrm{x}[n+N/2]\right) W_N^n$$

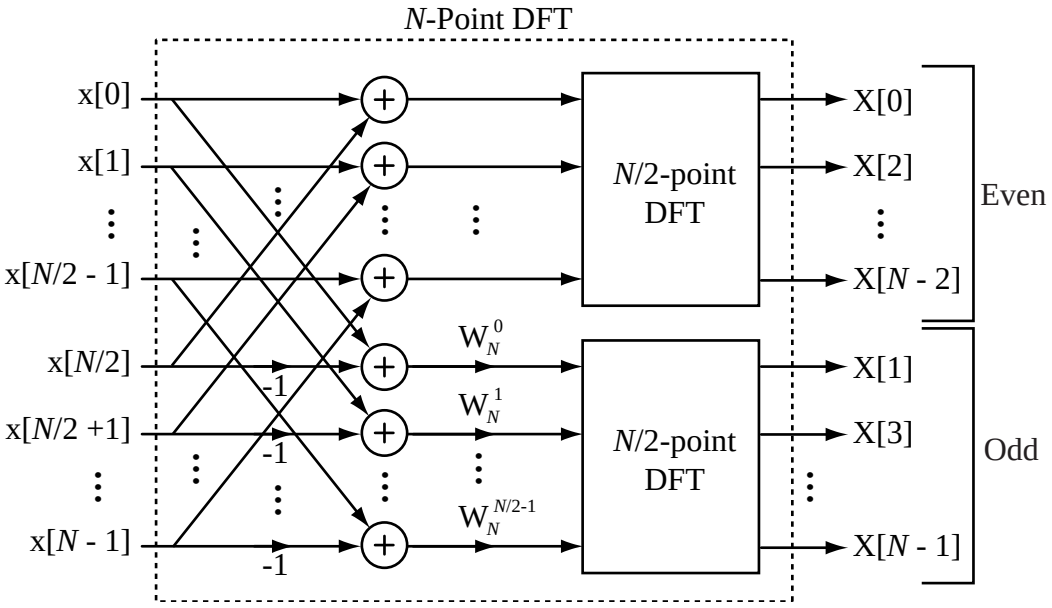

Figure M-3 An *N*-point DIF DFT computed using two *N*/2-point DFT's

The system in Figure M-3 illustrates the recursive nature of the FFT algorithm. Instead of actually computing an *N*-point DFT directly, split the data into two halves, form two linear combinations of those *N*/2-point data sets and the compute two *N*/2-point DFT's. The even-indexed DFT data points $\mathrm{X}[0], \mathrm{X}[2], \cdots, \mathrm{X}[N-2]$ are computed in the top half and the odd-indexed data points $\mathrm{X}[1], \mathrm{X}[3], \cdots, \mathrm{X}[N-1]$ are computed in the bottom half. *N* complex additions and *N*/2 complex multiplications are needed to prepare the *N* input data for the two *N*/2-point DFT's. Then each of the two *N*/2-point DFT's will require *N*/2 complex additions and *N*/4 complex multiplications to prepare for the four *N*/4-point DFT's. This subdivision continues until we get to *N*/2 2-point DFT's, each of which requires two complex additions and one complex multiplication. Suppose *N* is eight. Then the numbers of complex additions and complex multiplications are

$$\underbrace{8A + 4M}_{\text{8-point level}} + \underbrace{2[4A + 2M]}_{\text{4-point level}} + \underbrace{4[2A + M]}_{\text{2-point level}} = 24A + 12M$$

where A is the number of complex additions and M is the number of complex multiplications. Generalizing to N points, at each level there are N complex additions and $N/2$ complex multiplications and there are γ levels. Therefore, in general, $A = N\gamma$ and $M = N\gamma / 2$. Table M-1 summarizes the number of arithmetic operations for several powers of two for both the original DFT formula, (M.2), and the DIF FFT algorithm.

Table M-1 Numbers of additions and multiplications and ratios for several N's

γ	N	A_{DFT}	M_{DFT}	A_{FFT}	M_{FFT}	A_{DFT} / A_{FFT}	M_{DFT} / M_{FFT}
1	2	2	4	2	1	1	4
2	4	12	16	8	4	1.5	4
3	8	56	64	24	12	2.33	5.33
4	16	240	256	64	32	3.75	8
5	32	992	1024	160	80	6.2	12.8
6	64	4032	4096	384	192	10.5	21.3
7	128	16256	16384	896	448	18.1	36.6
8	256	65280	65536	2048	1024	31.9	64
9	512	261632	262144	4608	2304	56.8	113.8
10	1024	1047552	1048576	10240	5120	102.3	204.8

So, by using this technique we significantly reduce the number of complex additions and multiplications, especially for large N. It is important to emphasize that these speed improvement factors do not apply if γ is not an integer. For this reason, practically all actual DFT analysis is done with the FFT using data vector lengths which are an integer power of 2. (In MATLAB if the input vector is an integer power of 2 in length, the algorithm used in the MATLAB function `fft` is the FFT algorithm just discussed. If it is not an integer power of 2 in length, the DFT is still computed but the speed suffers because a less efficient algorithm must be used.)

The output data from the FFT algorithm don't come out in natural order but re-ordering them is simple and fast. The scrambling is illustrated for $N = 8$ in Figure M-4 with the natural order on the left and the scrambled order on the right.

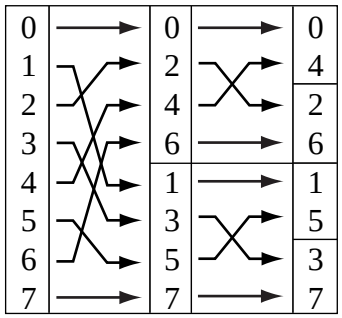

Figure M-4 Scrambling of the output data order for $N = 8$

The process of unscrambling the output data order is not as complicated as it may look. It can be done very quickly and efficiently by expressing the N indices as γ-bit binary numbers, and then simply reversing the bit order (Figure M-5).

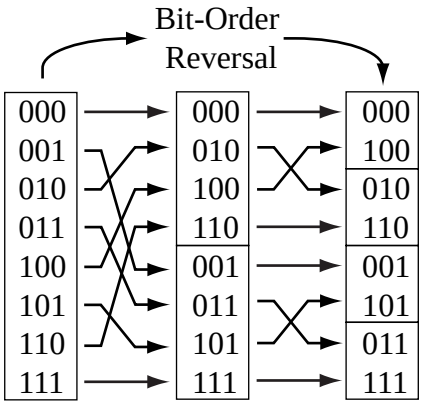

Figure M-5 Bit-order reversal to correctly identify the indices of the output data

So, for example, the second data point in an 8-point output array is actually $X[4]$ not $X[1]$.

M.3 The Decimation-in-Time FFT

Probably the easiest way to introduce the DIT FFT algorithm is to take the 4-point DIF FFT and re-order the input and output data while keeping all the connections the same (Figure M-6).

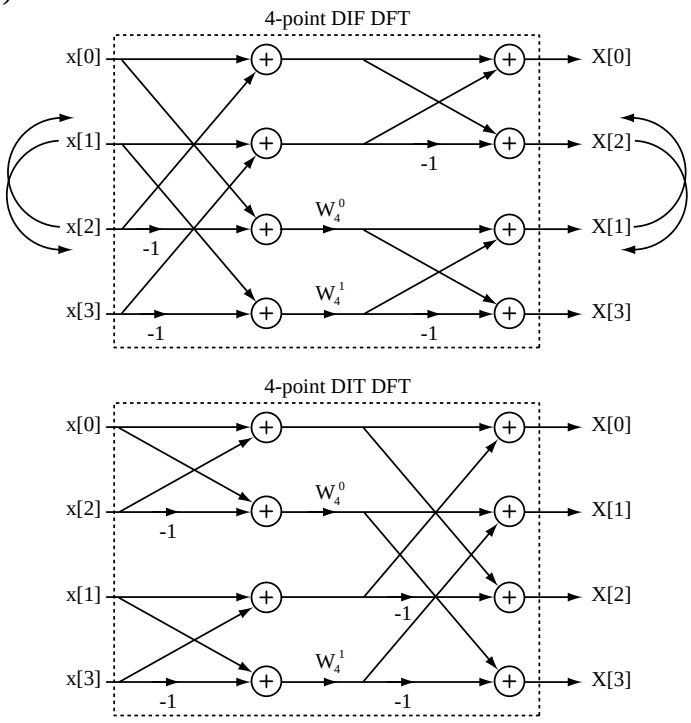

Figure M-6 Converting from a 4-point DIF FFT to a 4-point DIT FFT

The DIT FFT is the dual of the DIF FFT. It begins by reordering the input data by bit-reversing the indices and then builds up from 2-point DFT's to the final *N*-point DFT instead of decomposing an *N*-point FFT into multiple 2-point DFT's. If the DIF FFT is an example of "top-down" problem solving, then the DIT FFT must be an example of "bottom-up" problem solving. We begin the development of the DIT FFT by separating the input data into its even-indexed and odd-indexed values.

$$\mathrm{X}\left[k\right] = \sum_{n=0}^{N-1} \mathrm{x}\left[n\right] W_N^{nk} = \sum_{n \text{ even}} \mathrm{x}\left[n\right] W_N^{nk} + \sum_{n \text{ odd}} \mathrm{x}\left[n\right] W_N^{nk}$$

We can make the transformations $n \to 2q$ in the even summation and $n \to 2q+1$ in the odd summation yielding

$$\mathrm{X}\left[k\right] = \sum_{q=0}^{N/2-1} \mathrm{x}\left[2q\right] W_N^{2qk} + \sum_{q=0}^{N/2-1} \mathrm{x}\left[2q+1\right] W_N^{2qk} W_N^{k} .$$

Using the previously derived $W_N^{2nq} = W_{N/2}^{nq}$,

$$\mathrm{X}\left[k\right] = \sum_{q=0}^{N/2-1} \mathrm{x}\left[2q\right] W_{N/2}^{qk} + W_N^{k} \sum_{q=0}^{N/2-1} \mathrm{x}\left[2q+1\right] W_{N/2}^{qk} .$$

Let the even-indexed points in x be $\mathrm{x}_e\left[n\right]$ and let the odd-indexed points in x be $\mathrm{x}_o\left[n\right]$. Then

$$\mathrm{X}\left[k\right] = N/2 - \text{point DFT of } \mathrm{x}_e\left[n\right] + W_N^{k} \left(N/2 - \text{point DFT of } \mathrm{x}_o\left[n\right] \right).$$

In computing the values of $\mathrm{X}\left[k\right]$ for $N/2 \le k < N$ we need values of the two *N*/2-point DFT's which have been computed in the range $0 \le k < N/2$. Since the DFT is periodic the *N*/2-point DFT at any index *k* is the same as it is at $k \pm mN/2$ where *m* is any integer. Therefore the needed values at indices $N/2 \le k < N$ can come directly from indices $0 \le k < N/2$ which have already been computed. The form of a general *N*-point DIT DFT is illustrated in Figure M-7.

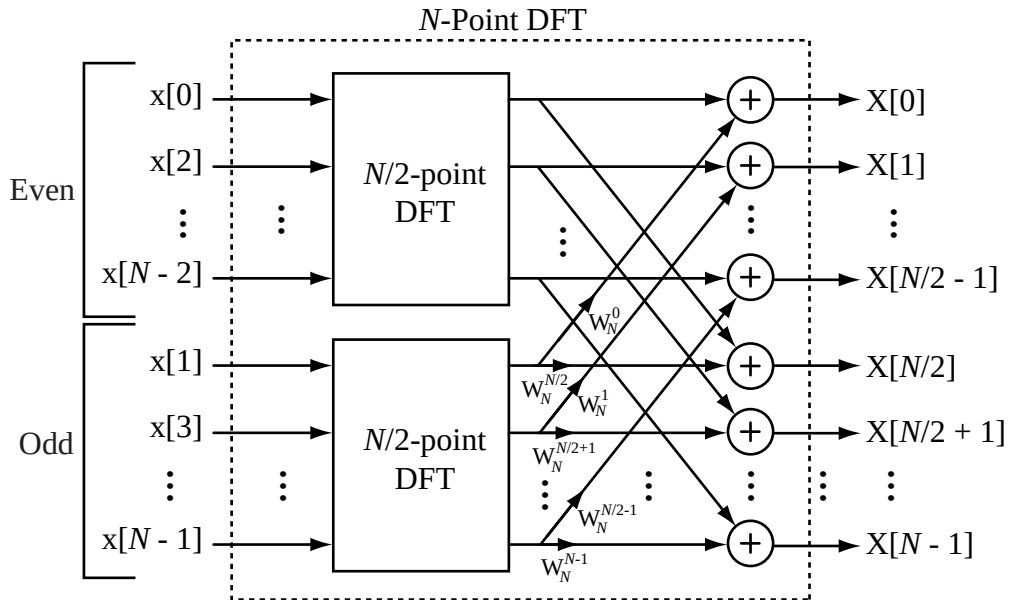

Figure M-7 An *N*-point DIT DFT computed using two *N*/2-point DFT's

In this form of the FFT algorithm the input data are scrambled but the output data are in natural order, just the opposite of the DIF FFT. The computational efficiencies of these two algorithms are basically the same. The only reason to choose between them might be that one form or the other makes system design easier given available hardware and/or software.

Example M-1 Comparison of three methods of finding a DFT

Find the DFT of $\left\{x_0[0], x_0[1], x_0[2], x_0[3]\right\} = \left\{-2, 4, 3, 5\right\}$

(a) directly using the DFT formula

$$X_0[k] = \sum_{n=0}^{N-1} x_0[n] W_N^{nk}$$

(b) using the DIF FFT system diagram,

and (c) using the DIT FFT system diagram.

$$W_4^1 = e^{-j\pi/2}$$

$$X_0[0] = \sum_{n=0}^{3} x_0[n] = -2 + 4 + 3 + 5 = 10$$

$$X_0[1] = \sum_{n=0}^{3} x_0[n] e^{-j\pi n/2} = -2 - j4 - 3 + j5 = -5 + j$$

$$X_0[2] = \sum_{n=0}^{3} x_0[n] e^{-j\pi n} = -2 - 4 + 3 - 5 = -8$$

$$X_0[3] = \sum_{n=0}^{3} x_0[n] e^{-j3\pi n/2} = -2 + j4 - 3 - j5 = -5 - j$$

4-point DIF DFT

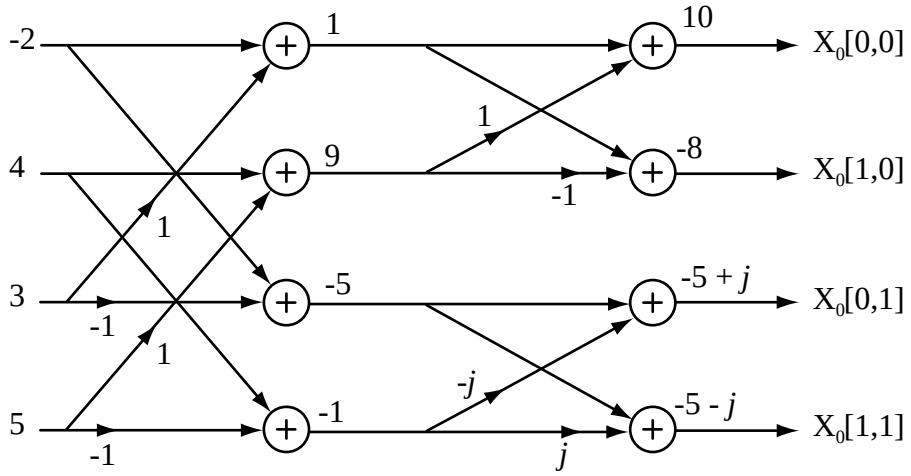

Figure M-8 DIF FFT system

4-point DIT DFT

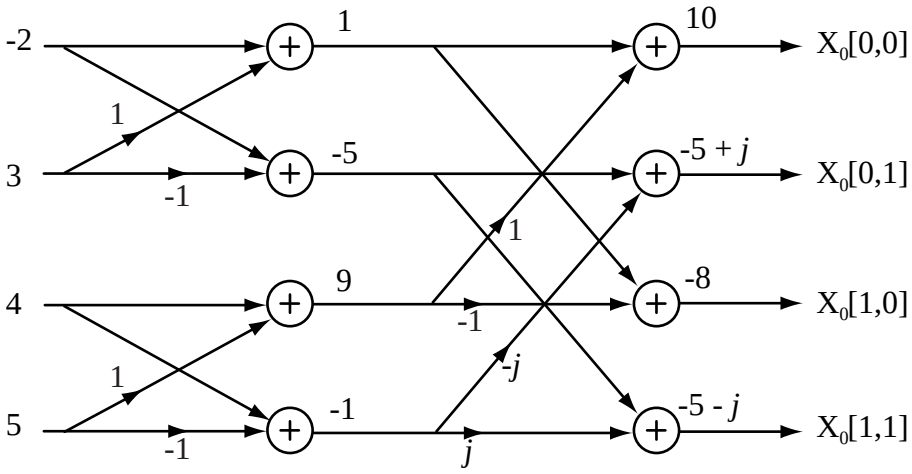

Figure M-9 DIT FFT System