Computational Arrays for the Discrete Fourier Transform

Lennart Johnsson
and
Danny Cohen

Computer Science
California Institute of Technology

4168:TR:81

COMPUTATIONAL ARRAYS FOR THE DISCRETE FOURIER TRANSFORM

by

Lennart Johnsson and Danny Cohen

TR:4168:81

Computer Science Department
California Institute of Technology
Pasadena, California 91125

Presented at
The 22nd Computer Society International Conference,
CompCon 81, San Francisco, February 23-26, 1981

The research described in this paper was sponsored by the
Defense Advanced Research Projects Agency, ARPA Order Number 3771,
and monitored by the
Office of Naval Research
under contract number N00014-79-C-0597

COMPUTATIONAL ARRAYS FOR THE DISCRETE FOURIER TRANSFORM

Lennart Johnsson
Computer Science
California Institute of Technology
Pasadena, Ca. 91125

Danny Cohen
Information Sciences Institute
University of Southern California
Marina Del Rey, Ca. 90291

Abstract. A mathematical approach towards the development of computational arrays for the Discrete Fourier Transform (DFT) is persued in this paper. Mathematical expressions for the DFT are given a direct hardware interpretation. Different implementations are developed by formal manipulation of the equations defining the DFT. Properties of the implementations can be told directly from the corresponding equations. Special consideration is given to the performance of implementations and corresponding hardware requirements. The standard equations defining the DFT on N values corresponds if the equations are given a direct hardware interpretation to an implementation requiring N^2 modules. By formal manipulation of the equations defining the DFT we develop implementations requiring N and $\log_2 N$ modules respectively.

INTRODUCTION

The Discrete Fourier Transform (DFT) has gained widespread use in signal processing and numerical analysis during the last decade, largely due to the efficient way of computing the DFT that was published by Cooley and Tukey[1] in 1965. Their algorithm is refered to as the Fast Fourier Transform (FFT). The FFT algorithm by Cooley and Tukey[1] was indeed a rediscovery of similar ways of computing the DFT, Cooley,Lewis and Welch[2]. The idea used in the FFT is to explore the properties of the coeffiecients to reduce the computations. The computational complexity that for the DFT of N data items is $O(N^2)$ is $O(N\log_2 N)$ for the FFT. Today there exist a variety of FFT algorithms. The algorithms differ in their storage requirements and addressing schemes, see for instance Oppenheim and Schafer[3]. Winograd[8,9] has published algorithms for the computation of the DFT that requires fewer multiplications than the FFT algorithms but more additions.

Most FFT algorithms have been devised for machines equipped with random access memory, that is each access of memory is considered equally costly. This assumption is not realistic if special harware is considered, especially in VLSI technology. Communication is a precious entity at feature sizes of about one micron or less, Mead and Sutherland[4]. It is highly desirable to minimize the number as well as the length of wires required on a chip. Wires largely determines the size of the chip. Long wires are likely to adversely affect the

performance.

Several solutions for implementing FFT algorithms in hardware have been proposed. Some of them have also been realised. Recently computational arrays and networks for the implementation of the DFT and FFT in VLSI technology have been proposed. Kung and Leiserson[5] describes an implementation that requires N modules. Their array accepts a new value every second cycle. We will also present an array that uses N modules. However our array accepts a new value every cycle. It also differs in the way coefficients are generated. In the arrays we propose all modules are identical. We also present a solution making use of $\log_2 N$ modules.

In this paper a mathematical approach towards computational networks for the DFT is persued. By manipulating the equations defining the DFT implementations with different properties can be derived. The properties of the implementations can be told directly from the equations defining the hardware. By adhering to a direct hardware interpretation of the equations correctness of the implementation is assured.

Depending upon the environment in which the FFT is used different criteria can be applied in determining what is a good solution. A basic assumption is that of the organisation and availability of the input data. In signal processing applications it is reasonable to assume the data to be available serially and in the order data is being recorded. Any other order require some form of intermediate storage. In applications where data is stored before the computation of the DFT is made, there is an option as to which input order to use. It may even be possible to access the data in parallell. Here we will assume serial input.

Various criteria are usually applied in evaluating the qualities of algorithms and architectures for the computation of a function. We will discuss a few criteria of importance to the computation of the DFT before we introduce a notation suitable for the derivation of different implementations. By formal manipulations of the equations defining the DFT we proceed from a direct interpretation of the equations requiring N^2 modules to implementations requiring N modules and finally to implementations requiring $\log_2 N$ modules.

CRITERIA FOR EVALUATION OF SOLUTIONS

Factors that need to be evaluated in most applications are correctness, accuracy, computational rate, delay, performance, area requirements or number of parts, modularity, fault tolerance, flexibility, power requirements and cost. In a computational network the intent is that several operations be performed concurrently. Algorithms generally introduce sequence requirements. The computational rate is inversely proportional to the maximum time, T, required by the network to perform the computations initiated in a sequential step in such a fashion that the result will be correctly interpreted in the next step.

The delay of a computational network is a measure of the time elapsed after the input is applied until the corresponding output is available. The delay is a function of the computational rate, T, and the number of steps required to compute the result from the input. We will measure the delay in units of T. The computational rate and the delay will then measure different properties of the network.

In computing the FFT the input and output are vectors that we assume enters and leaves respectively the network in serial form. This assumption leaves some alternatives for defining the delay. We will measure the delay from the time all components of the input vector becomes available to the network. A consistent view would be to define the delay to last until all the components of the DFT are available at the output. In some applications this may however be too conservative a measure. If not all components of the DFT are required or the interpretation of the DFT can start when the first component is available it is also of interest to know when the first component is available. We will define the minimum delay to be the time measured in units of T elapsed between the instant when all inputs are available and the availability of the first component of the DFT at the output. maximum delay is the time measured in units of T until all components of the DFT becomes available.

Figure 1 gives a graphic illustration to the definitions of minimum and maximum delay.

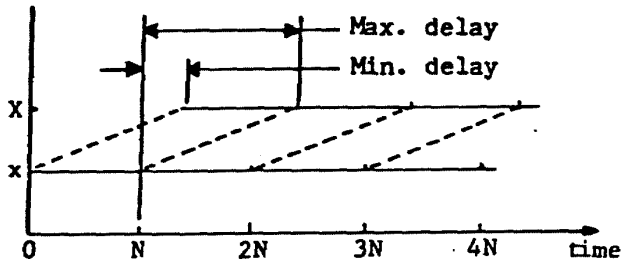

Figure 1.

With performance of a computational network we refer to the number of transforms that can be computed in a given amount of time. The performance is a function of the computational rate and the number of steps required to compute the output from the input. We measure the performance as the inverse of the number of steps required for the computation of a transform. Measured in this way we can in a much simplified way consider performance as a measure of higher level architectural or algorithmic properties while the computational rate is more closely related to lower level circuit qualities. The delay may or may not affect the performance.

Typically there is a trade off between hardware and performance. Suitable measures for the amount of hardware is needed. Considering implementation in the form of custom circuitry the ultimate measure of the amount of hardware is area. As an intermediate measure of hardware requirements the number of logic units (e.g. adders, multipliers) may be used. Sometimes sets of logic units naturally forms modules that appears in patterns in a computational network. We will use such modules as a measure of the hardware resources required. We will also define the content of the modules in terms of logic units.

We are not defining any particular measures of modularity and fault tolerance. Our discussion of these properties will be in common terms that need no particular explanation.

THE DISCRETE FOURIER TRANSFORM

The Discrete Fourier Transform can be defined as

$$X(k) = \sum_{j=0}^{N-1} a^{jk} x(j) \qquad k = 0,1,2,\ldots,N-1 \qquad (1)$$

In (1) x denotes the data and X the Fourier Transform. The coefficients $a = \exp(-2\pi i/N)$. The inverse transform can be expressed as

$$x(j) = 1/N \sum_{k=0}^{N-1} a^{-jk} X(k) \qquad j = 0,1,2,\ldots,N-1 \qquad (2)$$

For our treatment of the DFT and associated computational networks we introduce a delay operator Z defined as follows

$$Zx(j) = x(j-1) \qquad j>0$$

Using the notation $Z^k = ZZ^{k-1}$ the following result is apparent

$$Z^k x(j) = x(j-k) \qquad k \leq j$$

We also define an inverse Z^{-1} of the delay operator.

$$Z^{-1}x(j) = x(j+1)$$

The inverse has the following properties.

$ZZ^{-1} = Z^{-1}Z = I$ and $Z^kZ^{-k} = Z^{-k}Z^k = I$ for all k.

$Z^0 = I$, I = the identity operator

A constant C has the same value at all times. Hence applying the delay operator to a constant C has the following property.

$Z^kC = C$ for all k.

We notice that a constant C and the delay operator Z commute,

$Z^k(Cx(j)) = Cx(j-k) = CZ^kx(j)$,

and in general

$Z^kF(x,y) = F(Z^kx,Z^ky)$

For convenience we have assumed $j>0$ above.

Using the notation introduced above we have

$$x(j) = Z^{N-1-j}x(N-1)$$

and equation (1) can be written as

$$X(k) = \sum_{j=0}^{N-1} a^{jk} Z^{N-1-j} x(N-1) \quad k = 0,1,2,...,N-1 \tag{3}$$

With an apparent change in summation index and observing that x does not contain the summation index, equation (3) can be rewritten as

$$X(k) = \sum_{j=0}^{N-1} a^{(N-1-j)k} Z^{j} x(N-1) \quad k = 0,1,2,...,N-1 \tag{4}$$

In a signal processing environment values of the variable x can often be considered to be sampled from a continuous signal at equidistant instances of time. We have in equations (1) - (4) considered the time interval 0 - N-1.

Let the variable n denote time. Subsequently we will sometimes use the following interpretation of the DFT

$$X(n,k) = \sum_{j=0}^{N-1} a^{(N-1-j)k} Z^{j} x(n) \quad k = 0,1,2,...,N-1 \tag{5}$$

For convenience we may sometimes write

$$X(k) = \sum_{j=0}^{N-1} a^{(N-1-j)k} Z^{j} x \qquad k = 0,1,2,...,N-1$$

The intent of equation (5) is to separate the notation of time at which data is collected from the set of data on which the transform is computed. In equation (5) index n refers to time associated with data. Index j refers to the set of data for which the transform is to be computed, while k is the index of the components of the DFT. We assume that a new value of x occurs at certain intervals of time and that the computational network has to accept or receive a new value of x when available. In the computational network time can be associated also with the indicies j and k in various ways. In this paper we will present a few alternatives. Several other alternatives are described in Johnsson[7].

From a mathematical viewpoint it does not matter in which order we write the terms of the summation or if we interchange the order of the delay operator and the coefficients. There is a number of ways to write the right hand side of the equations which are all correct mathematically.

In the hardware interpretation of the equations we will assume that there is a unit corresponding to each mathematical operation and that the units are ordered with the unit corresponding to the rightmost operator in a term first. Each term is associated with a piece of hardware. All terms can be evaluated concurrently. If a sequence of parenthesis are used to express priority in the order of evaluation, the hardware interpretation starts from the outermost parenthesis.

COMPUTATIONAL NETWORKS OF N MODULES

According to equation (1) the computation of each component of X consists of evaluating a product sum. The computation of the response from a finite impulse response filter (FIR) also consists in the evaluation of a product sum. Various networks for FIRs have been investigated by Cohen[6]. The networks exhibits different properties with respect to computational rate, delay and logic units required. A straight forward use of the arrays discussed by Cohen[6] results in a network that has N^2 modules. There exist networks that can compute a DFT with no delay and in O(1) time. Hence a DFT is computed for every unit of time T. The DFTs are computed on the N most recent values of x. The N^2 modules perform N^2 operations (multiplications and additions) for every unit of time T.

Computing the DFT is equivalent to multiplying a matrix by a vector. The matrix contains the coefficients used to compute the DFT while the vector contains the data for which the DFT is to be computed. The computational requirements for the DFT can be reduced by exploiting the properties of the coefficients as done in FFT algorithms. In this section we will present solutions using N modules. We defer the discussion of solutions using $\log_2 N$ modules to the next section.

Equation (5) corresponds to N finite impulse response filters, one for every k. Each filter can be envisioned as consisting of N modules, one for every term. In a direct hardware interpretation the modules would all be different containing from zero to N-1 delays. The modules would operate in parallell and the outputs from the modules forming a finite impulse response filter has to be summed in order for the computation of a component of the DFT to be complete. This kind of hardware implementation will have a computational rate that can be improved in many ways, Cohen[6].

Since the operator Z commutes with constants equation (5) can be rewritten as

$$X(n,k) = (a^{(N-1)k}+Z(a^{(N-2)k}+Z(a^{(N-3)k}+....+Za^{0})...)x(n) \qquad k=0,1,2,...,N-1 \qquad (6)$$

Equation (6) contains a total of N-1 delays, a considerable savings compared to the direct interpretation of (5). We notice further that there is no delay between the arrival of x(n) and the availability of X(n,k). Equation (6) consists of N-1 operators of the form constant + delay. Equation (6) corresponds to a pipelined implementation. Products and partial sums are formed as the data become available. The variable x(n) is distributed or broadcasted to all N-1 operators. Applying the operator to its inputs produces one of the inputs to the next operator. Figure 2 shows an array of modules corresponding to equation (6).

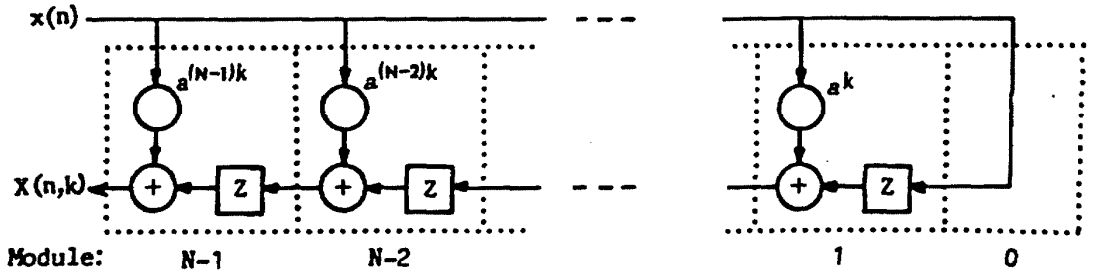

Figure 2.

To compute the DFT N arrays are required. One array does not need any multipliers since all coefficients are equal to 1. Hence the number of identical units required are at least $(N-1)^2$ plus N-1 units containing an adder and a delay.

In equations (5) and (6) the interrelationsship between the coefficients was disregarded. The coefficients in a row and/or a column of the matrix of coefficients defining the DFT can however easily be generated from each other. For the coefficients in column j it is true that

$$a^{jk} = a^j a^{j(k-1)}$$

In general consider a recursion as follows:

$$\text{Let } v_j(k) = a^j v_j(k-1) = a^j Z v_j(k) = a^{jk} Z^k v_j(k) = a^{jk} v_j(0)$$

$$\text{Let } v_j(0) = x(j)$$

A circuit implementing this recursion is shown in Figure 3.

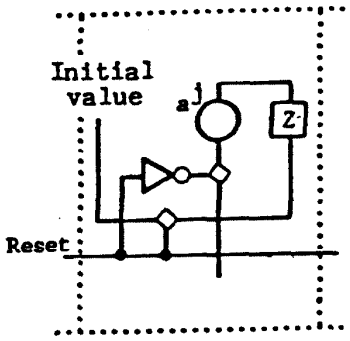

Figure 3.

With the initial values assumed for the variables v_j equation (1) can be rewritten as

$$X(k) = \sum_{k=0}^{N-1} v_j(k) \qquad k=0,1,2,...,N-1 \qquad (7)$$

Considering the variables v_j as generated by modules as given in Figure 3, equation (7) corresponds to an array of N modules the outputs of which are summed. The modules are all initiated at k = 0 but with different values. Module j is initiated to x(j) and has one of the multiplier inputs always set to a^j. The module corresponding to $2.v_0(k)$ can be reduced to a storage cell. The summed output of the array equals X(k) for k = 0.1.2.....N-1. An array corresponding to equation (7) can be drawn as in Figure 4. In Figure 4 we have used the relation $x(j) = Z^{N-1-j}x(N-1)$. It should be noticed that the modules need N steps to compute all components of the DFT. The array can then be reinitiated to compute a new DFT. Hence one array as in Figure 4 suffice to compute all components of the DFT in N steps. The DFT is computed on contiguos sets of N distinct data points.

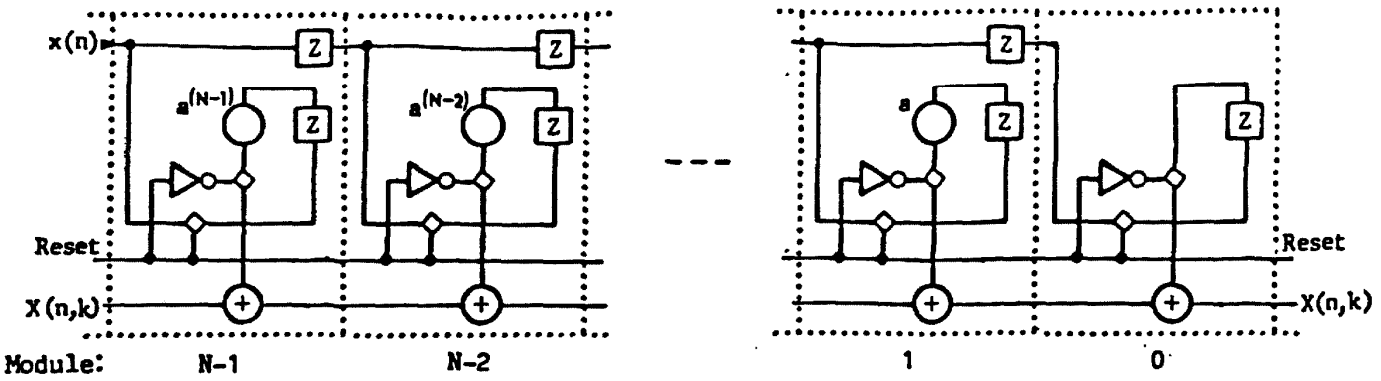

Figure 4.

The reset signal is defined as

$$\text{reset}(n) = 1 \quad \text{when} \quad n = 0(\text{mod}N)$$

$$= 0 \quad \text{otherwise}$$

The implementation corresponding to equation (7) will not have a high computational rate since there are N-1 adders in series. The minimum delay is zero and the maximum is N-1. Pipelining can be used to improve the computational rate.

Let u(j,n) be the output of module j at time n. Let the modules be connected into an array so that the output S(n) of the array can be written as follows

$$S(n) = u(N-1,n) + Z(u(N-2,n) + Z(....Z(u(0,n))...))$$

Further, initiate each module so that

$$u(j,n) = x(n) \quad \text{for} \quad n=j(\text{mod}N), \quad j=0,1,2,...,N-1 \quad m=0,1,2,3,....$$

$$= a^j u(j,n-1) \quad \text{otherwise}$$

Notice that u(j,n) are only defined for n>j-1.

Using the expressions for u, S can be expressed as

$$S(mN+j) = u(N-1,mN+j)+Z(u(N-2,mN+j) + Z(....Z(u(0,mN+j))...))$$

$$S(mN+j) = a^{(N-1)(j+1-N)}x(mN+N-1)+a^{(N-2)(j+1-N)}x(mN+N-2)+$$

$$+a^{(N-3)(j+1-N)}x(mN+N-3) + ... + a^0 x(mN+0) \qquad (8)$$

where mN+J>N-2.

Hence S(N-1) = X(0), S(N) = X(1), ..., S(2(N-1)) = X(N-1)

$$X(mN+N-1,k) = S(mN+N-1+k)$$

$$= a^{(N-1)k}x_{(mN+N-1)} + a^{(N-2)k}x_{(mN+N-2)} +$$

$$+ a^{(N-3)k}x_{(mN+N-3)} + \ldots + a^{0}x_{(mN+0)} \quad k = 0,1,2,\ldots,N-1 \quad m = 0,1,2,3,\ldots. \quad (9)$$

An array corresponding to equations (8) and (9) can be drawn as in Figure 5.

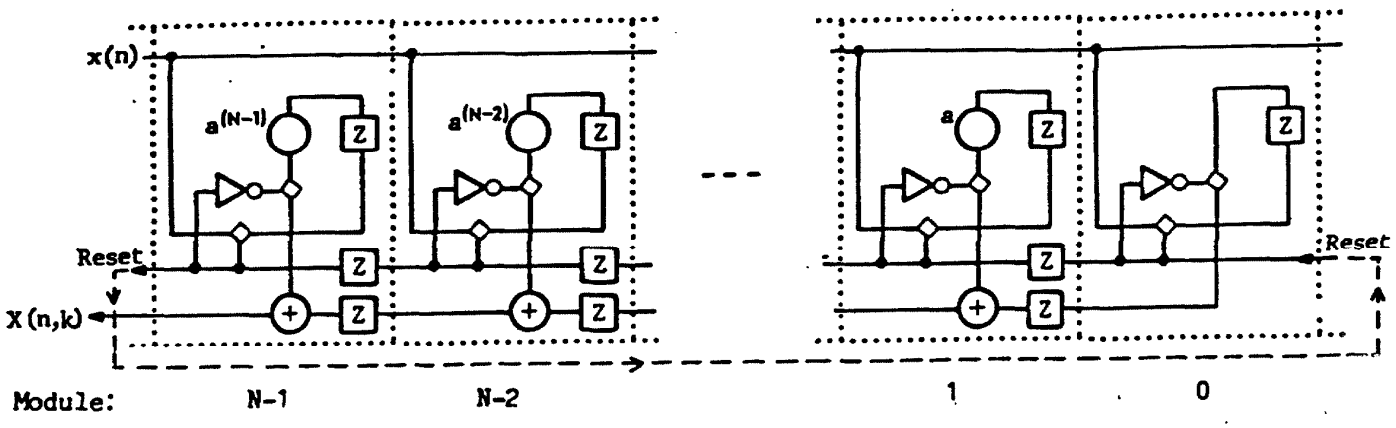

Figure 5.

The implementation of equations (8) and (9) requires slightly larger area than the previous implementation. A delay of the one-bit reset signal is required. The delay units for the input signal is removed and delays of the partial product sums introduced. The computational rate should be much improved. The components of the vector x are broadcasted to every module. Only one module at a time is receiving. The data is multiplexed to the modules.

If the array is layed out in a plane as a square, the length of the broadcasting wire will be $O(\sqrt{N})$. If the broadcasting wire is limiting the computational rate it is also possible to improve it by introducing delays between modules. A trade off may have to be made between delay and the computational rate. If delays are introduced between every pair of modules the total delay will be N-1. Broadcasting the values of the vector x in a tree like fashion will introduce a delay of order $\log_2 N$ and requires additional area.

The characteristics of the arrays defined by equation (7) and equations (8) and (9) respectively can be summarized as follows:

a) Serial input, serial output. The DFT is computed on distinct, contigous sets of N data.

b) The precision is determined by the multiplication and addition of N-1 numbers.

c) The computational rate of the array corresponding to equation (7) is limited by N-1 additions in series. The computational rate of the array corresponding to equations (8) and (9) is determined by one multiplication and one addition. Hence it should be much improved over the array corresponding to equation (7).

d) The minimum delay is zero. The maximum delay is N-1.

e) A DFT can be computed once every N cycles.

f) There are N identical modules. The module corresponding to the variable x(0) can be simplified. The multiplier is unnecessary. Without this simplification we have

One adder, one multiplier and two delays per module, for the array of equation (7).

One adder, one multiplier, two delays for data and a 1 bit delay for the reset signal per module, for the array of equations (8) and (9).

g) Modularity: all parts identical, except possibly for the x(0) module.

h) Every module participates in the computation of every component of the DFT. There is no degree of fault tolerance.

i) The arrays are designed for matrix vector multiplication with the assumption that the quotient between two consecutive elements in a column is the same for all elements in the column. The quotient may vary from column to column. If the vector and matrix size is less than the number of modules the implementation of equation (7) will still work correctly. In the implementation corresponding to equations (8) and (9) the vector x has to be appended with zeroes to form a vector of length N or means be provided to tap the arrays after the proper number of modules and shorten the length of the sampling "loop".

If the order of the input data is not the same as the sampling order the coefficients used by the multipliers have to be interchanged. The leftmost module corresponds to the data item arriving as number N-1 and shall have a coefficient corresponding to the sampling instant of that data item. The output will appear in normal order.

Wave fronts can be used to illustrate the progression of the computations. We define the progression of the computation in each module in terms of the output of its multiplier. Hence in the implementation corresponding to equation (7) all modules have computed equally many coefficients at any given time. Illustrated in terms of the matrix of coefficients defining the DFT the computations proceed from top to bottom concurrently in each column and the computations are equally far along in all columns, see Figure 6. In the implementation according to equations (8) and (9) all coefficients are computed in the module corresponding to x(0) when the computations of coefficients start in the module corresponding to x(N-1). Hence the progression of the computations can be drawn as a

diagonal from the lower left hand corner of the matrix to its upper right hand corner at time N-1. The wave front proceeds from the upper left hand corner to the lower right hand corner. Most of the time there are two wave fronts at distance N passing over the matrix.

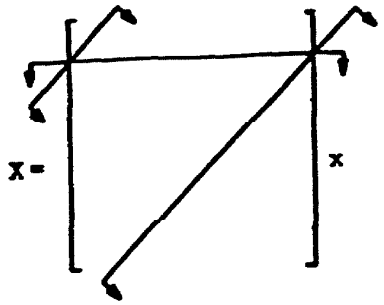

Figure 6.

COMPUTATIONAL NETWORKS FOR THE FFT

The computational networks investigated above have a module/performance ratio of $O(N^2)$. The complexity of FFT algorithms is $O(N\log_2 N)$. It takes O(N) time to receive the data, assuming serial input. If a sequence of DFTs are to be computed it is apparently not possible to compute a DFT at a higher rate than once every N cycles. Still there is however a choice to be made whether to minimize delay or to reduce the amount of hardware. Here we will present solutions that require $\log_2 N$ modules and has a maximum delay of N.

Assume $N = 2^m$, m = integer

Then $a^{(j+N/2)k} = a^{jk}$ if k is even, j=0,1,2,...,N/2-1

$=-a^{jk}$ if k is odd, j=0,1,2,...,N/2-1

Hence we get

$$X(2k) = \sum_{j=0}^{N/2-1} a^{j2k}(x(j)+x(j+N/2)) \quad k=0,1,..,N/2-1 \qquad (10)$$

$$X(2k+1) = \sum_{j=0}^{N/2-1} a^{j(2k+1)}(x(j)-x(j+N/2)) \quad k=0,1,..,N/2-1 \quad \text{or}$$

$$X(2k+1) = \sum_{j=0}^{N/2-1} a^{j2k}(a^{j}(x(j)-x(j+N/2))) \quad k=0,1,..,N/2-1 \qquad (11)$$

The even and odd components of the DFT can be obtained by performing a product sum over N/2 components of the input vector. The coefficients can be the same if we compute two new vectors of the length N/2. The coefficient matrix is of the size N/2 by N/2. It is applied twice, once for the even components of the DFT, once for the odd components.

Using the Z operator equation (11) can be written as

$$X(2k) = \sum_{j=N/2}^{N-1} a^{(j-N/2)2k}(Z^{N/2}+1)x(j) \qquad (12)$$

$$X(2k+1) = \sum_{j=N}^{3N/2-1} a^{j2k}a^{j}(Z^{N}-Z^{N/2})x(j) \qquad (13)$$

Let $u(j-N/2) = (Z^{N/2}+1)x(j)$ $\quad j=N/2,N/2+1,\ldots,N-1$

and $u(j-N/2) = a^{j}(Z^{N}-Z^{N/2})x(j)$ $\quad j=N,N+1,\ldots,3N/2-1$

Then the DFT can be written in matrix form as follows

$$\begin{array}{l} X(0) \\ X(2) \\ \\ \\ X(N-2) \\ X(1) \\ X(3) \\ \\ \\ X(N-1) \end{array} = \left|\begin{array}{cccccc|c} 1 & 1 & 1 & 0 & 0 & 0 & \\ 1 & a^2 & a^4 & 0 & 0 & 0 & \\ & & & & & & \\ & & & & & & \\ 1 & & & 0 & 0 & 0 & u \\ 0 & 0 & 0 & 1 & 1 & 1 & \\ 0 & & 0 & 1 & a^2 & & \\ & & & & & & \\ & & & & & & \\ 0 & & 0 & 1 & & & \end{array}\right|$$

Hence the matrix representation has changed from a full matrix to a block diagonal matrix where the diagonal blocks are identical. A module computing the vector u is shown in Figure 7. The output of the module is the components of the vector u exactly in the order used in equation (14).

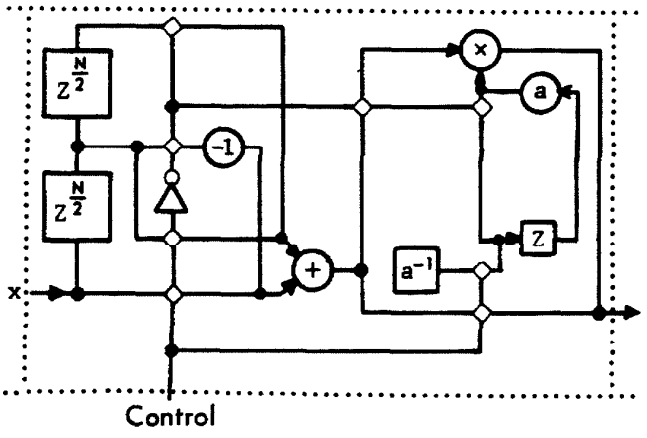

Figure 7.

The observation that was used to arrive at equations (10) - (14) can obviously be applied repetitively until the original matrix is transformed into a block diagonal matrix where the blocks are of dimension 2 by 2. The first row of the diagonal block is (1 1) and the second

is (1 $a^{N/2}$) or (1 -1). Hence no multiplier is needed in the final step. It should be noticed that for each module used the number of vectors double and their length is divided by two. The last module is supplied by N/2 vectors of length two. The number of values supplied to a module is constant. The interpretation of the input stream varies from module to module. An array corresponding to a complete diagonalization of the matrix consists of $\log_2 N$ modules as shown in Figure 8.

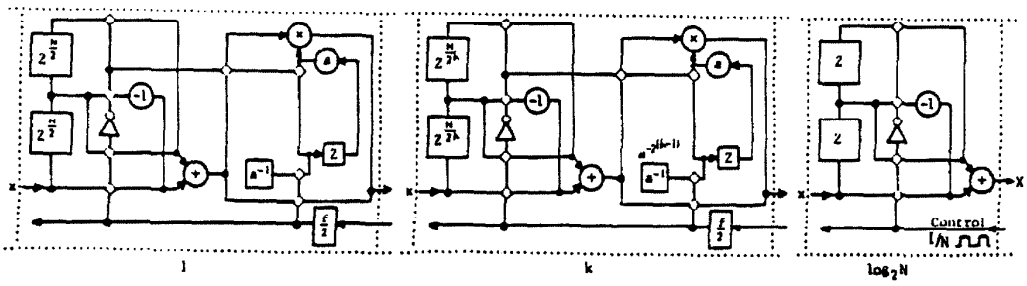

Figure 8.

In the design shown in Figure 8 the input is assumed to be supplied in normal order. The components of the DFT is computed in bit reversed order.

The computational rate is determined by $\log_2 N$ adders and multipliers in series. A delay can be introduced between each pair of modules to increase the computational rate. The output will be delayed $\log_2 N-1$ steps.

It should be noticed that the multiplexing of the inputs to the adder and the multiplier depends on the location of the module in the array. For the first or input module the input and output to the adder shall be switched every N/2 steps. For the second module the switching shall be made every N/4 steps and so on.

Each module contains two multipliers used half of the time. Unfortunately they are used at the same time. It certainly would be desirable to have one multiplier used 100% of the time. One obvious alternative would be to use the idle half of the multiplier receiving input data to compute coefficients. This would however require a shift register to store the coefficients. The register would be of length N/2 for the first or input module, N/4 for the next etc. Using shift registers the coefficients could be read from a table instead of being generated in the modules. Reading coefficients from outside may improve accuracy. A way of avoiding the shift registers would be to arrange the computations of additions and subtractions so that they alternated. A coefficient could then be generated while an addition is made and used in the following step.

If the order of the input signal would have been x(0), x(N/2), x(1), x(1+N/2), x(2), x(2+N/2),....then only two delays would have been necessary in the input module. The output sequence would consist of x(0)+x(N/2), x(0)-x(N/2), x(1)+x(1+N/2), a(x(1)-x(1+N/2)),....Hence this order of the input signal not only reduces the number of delays in the first module but also provides a good order with respect to coefficient generation. One multiplier will suffice. The multiplier will compute new coefficients every other cycle. A coefficient is used in the DFT computation in the cycle that follows the one in which it was computed. The sequence of coefficients to be generated is 1, a, a^2, a^3,..., a^{N-1}. A module as shown in Figure 9 can serve as input module.

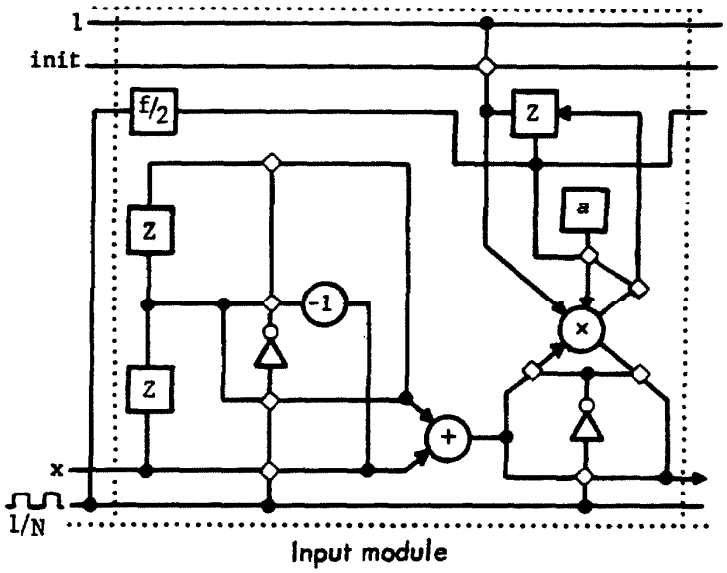

Figure 9.

The module next to the input module in an array for the computation of the DFT need to be slightly different from the input module. The first module is supplied with one vector to be transformed into two according to equations (10) and (11). Its output consists of those two vectors interleaved. The second module shall generate two new vectors for each of the incomming vectors and has to recognize the interleaving to make the proper computations. A module as in Figure 10 will do that. Each module has N delays and taps between every pair of delays. The same module can be used as input module if the input is supplied in normal order. For further details and proof of correctness see Johnsson[7].

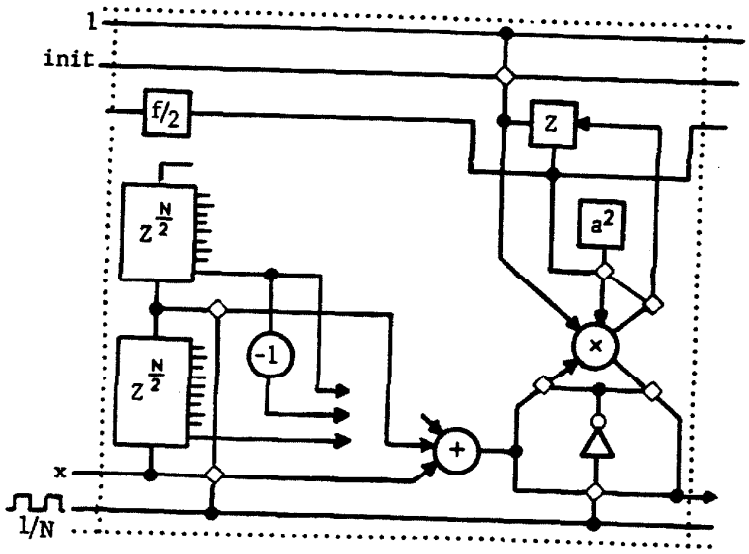

Figure 10.

The multiplexing of the input to the adder shall supply it with values a distance N/2 apart. The multiplexing is made every second cycle in such a way that after two numbers have been added the taps one step ahead in the shift direction are connected to the adder the next step and the same two numbers subtracted. The next step two new numbers are added by not changeing taps. The following step two new taps are connected to the adder and the numbers subtracted and so on. Instead of using a shift register between every pair of cells a random access memory can obviously be used.

The sequence of coefficients to be generated in module number two is 1, 1, a^2, a^2, a^4, a^4, a^6, a^6,..., $a^{N/2-2}$, $a^{N/2-2}$. Again those coefficients are used every second cycle. Hence a new coefficient need not be generated in every second cycle in module number two.

In general the sequence of coefficients in module number k is 1, 1,..., 1, $a^{1*2^{k-1}}$, $a^{1*2^{k-1}}$, ..., $a^{1*2^{k-1}}$, ..., $a^{1*(2^{k-1}(N/(2^k)-1))}$, $a^{1*(2^{k-1}(N/(2^k)-1))}$, ..., $a^{1*(2^{k-1}(N/(2^k)-1))}$. There are 2^{k-1} coefficients of the same value in sequence. For the final or output module k equals $\log_2 N$. The coefficients are then all 1 and no multiplier is needed. The number of coefficients in each module totals N/2. One cofficient is used every second cycle.

This alternative with input in normal order will allow all modules to be identical and only use one multiplier. The price paid is that every module contains N delays and taps between every pair. If the delays are layed out in form of a square it should be possible to limit wire length to the square root of N. The components of the DFT will appear in bit reversed order.

The characteristics of the implementations proposed in this section can be summarized as follows:

a) Serial input, serial output. If a continous stream of data is supplied to the array the DFT will be computed on sets of N distinct, contigous input values.

b) Coefficients are generated in the modules through multiplication. N/2-1 succesive multiplications are performed in the first module to compute the last coefficien. A total of $\log_2 N$ additions an multiplications are performed on the input data in the computation of each component of the DFT.

c) The computational rate may be limited by the signal propagation through $\log_2 N$ adders and multipliers in series. If the shift registers are implemented on chip it is possible to avoid wires of length N by folding the shift register array.

d) The minimum delay is N-1 and the maximum delay hence 2N-1.

e) One DFT can be computed every N cycles.

f) Each module in the array corresponding to Figure 8 contains one adder, two multipliers, a delay and a storage cell for coefficients and a number of delay units that varies from module to module. The number of delays is N for the input module and 2 for the output module.

In the implementation corresponding to Figure 10 each module contains one adder, one multiplier and N delays. For a complete array the number of delays has increased from 2(N-1) to $N\log_2 N$ and the number of multipliers decreased from 2(N-1) to N-1 compared to the previous alternative.

g) The modules in the implementation corresponding to Figure 8 differ with respect to the number of delays. This difference should not be of a significant disadvantage if several modules can be placed on one chip. With one module per chip $\log_2 N$ different chips are needed.

In the implementation corresponding to Figure 10 all modules can be made identical.

h) Every module participates in the computation of every component of the DFT. Error in an adder will cause an error in every component of the DFT while error in any one multiplier will cause half of the output to be erronous. Which components will be erronous depends upon which multiplier malfunctions. The analysis can be carried out in a straight forward fashion using equation (14) and the equations defining the vector u. For instance if any of the two multipliers (or both) in the input module fails all odd components in the DFT will be erronous.

i) Each module corresponding to Figure 8 can only be used in the computation of a DFT on at least as many points as it contains delays. A module containing N delays can serve as the input module in computing the DFT on N points, as the second module in computing the DFT on 2N points and so on.

In the implementation corresponding to Figure 10 the modules can not be used in the computation of a DFT on more than N points. If there are fewer than N points, say M then the modules can be used if the control is changed. The registers shall be tapped at a distance M/2 apart and the scanning of the registers and the coefficient generation reset every M steps.

Instead of supplying the input in normal order bit reversed order is often prefered. With bit reversed input two delays suffice in the input module. The next module requires 4 delays with taps between every pair and the output module requires N delays with taps between every pair. The adder will use taps a distance 2 apart in the second module and a distance N/2 apart in the output module. In the input module new coefficients are needed every other cycle. The sequence of coefficient values is however not easy to generate. The sequence would be 1, $a^{N/4}$, $a^{N/8}$, $a^{3N/8}$,...., a, $a^{N/4+1}$, $a^{N/8+1}$,.... The power of a equals the index of every second input. Hencee bit reverse input does not seem to be attractive if the coefficients are to be generated in the modules.

Every module in this alternative depends on N. The modules are not easily adaptable to a different N, larger or smaller. In the previous alternative a set of modules computing the DFT for a certain N always form the set of modules closest to the output for a larger N.

CONCLUSIONS

We have taken a mathematical approach towards the development of computational networks for the Discrete Fourier Transform. We assumed data to be supplied serially and searched for algorithms that computes a DFT in an amount of time that equals the time it takes to supply the input data. By formal manipulation of the equations defining the DFT recognizing the properties of the coefficients we derived a few computational arrays for the DFT. Special consideration was given to the communication structure of the algorithms. The properties of the arrays are in general apparent from the equations the arrays are implementing. By exploring the properties of the coeffiecients we derived equations which nicely corresponds to arrays of N modules, each containing one adder, one multiplier and one to three delay elements. Further exploration of the properties of coefficients rendered arrays of logN modules, each of which contained one adder, one multiplier and at most N delay elements. Establishing an efficient flow of data and preserving a high degree of modularity are the main difficulties in solutions using $\log_2 N$ modules.

ACKNOWLEDGEMENT

The research reported here has been supported by the Defense Advanced Research Project Agency under contract MDA-80-C-0523. Views and conclusions contained in this paper are the authors and shall not be interpreted as representing the offical opinion or policy of DARPA, the US Government or any person or agency connected with them.

REFERENCES

1. Cooley J W and Tukey J W, "An Algorithm for Machine Calculation of Complex Fourier Series", Math. Computation, Vol. 19, pp. 297-301, 1965

2. Cooley J W, Lewis P A W and Welch P D, "Historical Notes on the Fast Fourier Transform", IEEE Trans. Audio Electroacoust., Vol. AU-15, pp. 76-79, June 1967

3. Oppenheim A V and Schafer R W, "Digital Signal Processing", Prentice-Hall, 1976.

4. Mead C A and Sutherland I, "Microelectronics and Computer Science", Scientific American, Vol. 237, no. 3, pp. 210-229, 1977.

5. Kung H T and Leiserson C E, "Algorithms for VLSI Processor Arrays", in Mead C A and Conway L, "Introduction to VLSI Systems", Addison-Wesley, 1980.

6 Cohen D, "Mathematical Approach to Computational Networks", Report ISI/RR-78-73, Information Sciences Institute, 1978.

7 Johnsson S L, "A Mathematical Approach Computational Networks for the Discrete Fourier Transform", in preparation.

8 Winograd S, "On Computing the Discrete Fourier Transform", Mathematics of Computation, Vol. 32, no. 141, 1978, pp. 175 - 199.

9 Winograd S, "Arithmetic Complexity of Computations", CBMS-NSF Regional Conference Series in Applied Mathematics, no. 33, SIAM 1980.