EE 274 / CoE 121

1st Semester 2009-2010

Rowena Cristina L. Guevara

Example 1

7.1 The first five points of the eight-point DFT of a real-valued sequence are

$\{0.25, 0.125-j0.3018, 0, 0.125-j0.0518, 0\}$

Determine the remaining three points.

Since x(n) is real, the real part of the DFT is even, imaginary part odd. Thus, the remaining points are

$\{0.125+j0.0518, 0, 0.125+j0.3018\}$

Example 2

7.2 Compute the eight-point circular convolution for the following sequences.

(a)

$$x_1(n) = \{1, 1, 1, 1, 0, 0, 0, 0\}$$

$$x_2(n) = \sin \frac{3\pi}{8} n, \qquad 0 \le n \le 7$$

$$\begin{aligned} \tilde{x}_2(l) &= x_2(l), \quad 0 \le l \le N-1 \\ &= x_2(l+N), \quad -(N-1) \le l \le -1 \\ \tilde{x}_2(l) &= sin(\frac{3\pi}{8} l), \quad 0 \le l \le 7 \\ &= sin(\frac{3\pi}{8}(l+8)), \quad -7 \le l \le -1 \\ &= sin(\frac{3\pi}{8}|l|), \quad |l| \le 7 \end{aligned}$$

Example 2

Therefore, $x_1(n) \textcircled{8} x_2(n)$

$$
\begin{aligned}
&= \sum_{m=0}^{3} \tilde{x}_2(n-m) \\
&= sin(\frac{3\pi}{8}|n|) + sin(\frac{3\pi}{8}|n-1|) + \ldots + sin(\frac{3\pi}{8}|n-3|) \\
&= \{1.25, 2.55, 2.55, 1.25, 0.25, -1.06, -1.06, 0.25\}
\end{aligned}
$$

Example 2

Use DFT to compute the circular convolution

$$\begin{aligned} X_1(k) &= \sum_{n=0}^{7} x_1(n) e^{-j\frac{\pi}{4}kn} \\ &= \{4, 1-j2.4142, 0, 1-j0.4142, 0, 1+j0.4142, 0, 1+j2.4142\} \end{aligned}$$

$$\begin{aligned} X_2(k) &= \{1.4966, 2.8478, -2.4142, -0.8478, -0.6682, -0.8478, \\ &\quad -2.4142, 2.8478\} \end{aligned}$$

$$\begin{aligned} \text{DFT of } x_1(n) \textcircled{8} x_2(n) &= X_1(k)X_2(k) \\ &= \{5.9864, 2.8478-j6.8751, 0, -0.8478+j0.3512, 0, \\ &\quad -0.8478-j0.3512, 0, 2.8478+j6.8751\} \end{aligned}$$

$x_1(n) \textcircled{8} x_2(n)$ = {1.25, 2.55, 2.55, 1.25, 0.25,−1.06,−1.06, 0.25}

Example 3

7.10 $x_1(n) = \{0, 1, 2, 3, 4\}$

$s(n) = \{1, 0, 0, 0, 0\}$

Is there a sequence $x_3(n)$ such that $S(k)=X_1(k)X_2(k)$

Let $x_3(n) = \{x_0, x_1, \ldots, x_4\}$. Then,

$$\begin{bmatrix} 0 & 4 & 3 & 2 & 1 \\ 1 & 0 & 4 & 3 & 2 \\ 2 & 1 & 0 & 4 & 3 \\ 3 & 2 & 1 & 0 & 4 \\ 4 & 3 & 2 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$x_3(n) = \left\{ \underset{\uparrow}{-0.18}, 0.22, 0.02, 0.02, 0.02 \right\}$$

Example 4

7.17 a. Determine the Fourier Transform $X(\omega)$ of the signal

$x(n) = \{1, 2, 3, 2, 1, 0\}$

b. Compute the six-point DFT $V(k)$ of the signal

$v(n) = \{3, 2, 1, 0, 1, 2\}$

c. Is there any relation between $X(\omega)$ and $V(k)$?

Example 4

(a)
$$\begin{aligned} X(w) &= \sum_{n=-\infty}^{\infty} x(n)e^{-jwn} \\ &= e^{j2w} + 2e^{jw} + 3 + 2e^{-jw} + e^{-j2w} \\ &= 3 + 2cos(2w) + 4cos(4w) \end{aligned}$$

(b)
$$\begin{aligned} V(k) &= \sum_{n=0}^{5} v(n)e^{-j\frac{2\pi}{6}nk} \\ &= 3 + 2e^{-j\frac{2\pi}{6}k} + e^{-j\frac{2\pi}{6}2k} + 0 + e^{-j\frac{2\pi}{6}4k} + e^{-j\frac{2\pi}{6}5k} \\ &= 3 + 4cos(\frac{\pi}{3}k) + 2cos(\frac{2\pi}{3}k) \end{aligned}$$

(c) $V(k) = X(w)|_{w=\frac{2\pi k}{6}=\frac{\pi k}{3}}$

$v(n)$ is one period $(0 \leq n \leq 7)$ of a periodic sequence obtained by repeating $x(n)$.

The Fast Fourier Transform (FFT)

The DFT has many applications in DSP, including linear filtering, spectrum and correlation analysis.

There are more efficient means of computing the DFT.

The direct computation of the DFT involves the evaluation of the sum:

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

$$W_N = e^{-j2\pi/N}$$

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

The Fast Fourier Transform (FFT)

$$X_R(k)=\sum_{n=0}^{N-1}\left[x_R(n)\cos\frac{2\pi kn}{N}+x_I(n)\sin\frac{2\pi kn}{N}\right]$$

$$X_I(k)=-\sum_{n=0}^{N-1}\left[x_R(n)\sin\frac{2\pi kn}{N}-x_I(n)\cos\frac{2\pi kn}{N}\right]$$

This requires

a. $2N^2$ evaluations of trigonometric functions

b. $4N^2$ real multiplications.

c. $4N(N-1)$ real additions

d. A number of indexing and addressing operations

The complexity increases in the $O(N^2)$.

The Fast Fourier Transform (FFT)

A common approach in reducing the computational complexity of the DFT involves a *divide-and-conquer* method.

This approach decomposes the *N*-point DFT into *successively smaller* DFTs.

This leads to a family of computationally-efficient algorithms called the *Fast Fourier Transform* (FFT) algorithms.

Symmetry Property: $W_N^{k+N/2} = -W_N^k$

Periodicity Property: $W_N^{k+N} = W_N^k$

The Fast Fourier Transform (FFT)

An FFT algorithm would involve the following steps:

- Store the signal $x(n)$ column-wise
- Compute the M-point DFT of each row
- Multiply the resulting array by the phase factors
  $W_N^{lq}=e^{-j2\pi lq/N}$
- Compute the L-point DFT of each column
- Read the resulting array row-wise

The Fast Fourier Transform (FFT)

A divide and conquer approach is applicable to cases where the number of points *N* in the DFT is not a prime number.

An important case is when the length *N* of the DFT can be expressed as $N = r^{v}$.

In this case, each of the smaller DFTs have size *r*, this results in a more regular pattern for the DFT computation using the FFT algorithm. The number *r* is called the *radix* of the FFT algorithm. The most common FFT algorithm is the *radix*-2 FFT.

The Fast Fourier Transform (FFT)

The following table lists the comparison of the computational complexity for the direct DFT computation and the FFT algorithm

No. of points	Complex Multiplications	Complex Multiplications	Speed Improvement Factor
8	64	12	5.3
128	16384	448	36.6
1024	1048576	5120	204.8

The Fast Fourier Transform (FFT)

Consider the discrete Fourier Transform (DFT)

$$(1)\quad x(n) = \tfrac{1}{N}\sum_{n=0}^{N-1} X(k)e^{j2nk/N}$$

n=0,1,…,N-1

$$(2)\quad X(k) = \tfrac{1}{N}\sum_{n=0}^{N-1} x(n)e^{-j2nk/N}$$

k=0,1,…,N-1

The Fast Fourier Transform (FFT)

Let N=4, & let $W = e^{-j2\pi/N}$. Then, we can write

$$X(0) = x(0)W^0 + x(1)W^0 + x(2)W^0 + x(3)W^0$$

$$X(1) = x(0)W^0 + x(1)W^1 + x(2)W^2 + x(3)W^3$$

$$X(2) = x(0)W^0 + x(1)W^2 + x(2)W^4 + x(3)W^6$$

$$X(3) = x(0)W^0 + x(1)W^3 + x(2)W^6 + x(3)W^9$$

$2N^2$ evaluation of trigonometric fcns & $4N^2$ real multiplies

OR N^2 complex multiplies & (N-1)N complex additions

The Fast Fourier Transform (FFT)

In Matrix form,

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix} = \begin{bmatrix} W^0 & W^0 & W^0 & W^0 \\ W^0 & W^1 & W^2 & W^3 \\ W^0 & W^2 & W^4 & W^6 \\ W^0 & W^3 & W^6 & W^9 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix}`$$

$$X(n) = W^{nk} x(k) \quad \rightarrow \text{notation}$$

The Fast Fourier Transform (FFT)

Now,

$$W^{nk} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & W^1 & W^2 & W^3 \\ 1 & W^2 & W^0 & W^2 \\ 1 & W^3 & W^2 & W^1 \end{bmatrix}$$

The Fast Fourier Transform (FFT)

Since $W^{nk} = W^{nk \bmod (N)}$ $\rightarrow$ remainder upon division of nk by N

$$W^6 = e^{\frac{-j2\pi}{6}(6)} = e^{-j3\pi}$$

$$= e^{-j\pi} = e^{-j\frac{2\pi}{4}(2)} = W^2$$

The Fast Fourier Transform (FFT)

Simplifying,

$$\begin{bmatrix} X(0) \\ X(2) \\ X(1) \\ X(3) \end{bmatrix} = \begin{bmatrix} 1 & W^0 & 0 & 0 \\ 1 & W^2 & 0 & 0 \\ 0 & 0 & 1 & W^1 \\ 0 & 0 & 1 & W^3 \end{bmatrix} \begin{bmatrix} 1 & 0 & W^0 & 0 \\ 0 & 1 & 0 & W^0 \\ 1 & 0 & W^2 & 0 \\ 0 & 1 & 0 & W^2 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix}$$

The Fast Fourier Transform (FFT)

Let

$$X_1(n) = \begin{bmatrix} X_1(0) \\ X_1(1) \\ X_1(2) \\ X_1(3) \end{bmatrix} = \begin{bmatrix} 1 & 0 & W^0 & 0 \\ 0 & 1 & 0 & W^0 \\ 1 & 0 & W^2 & 0 \\ 0 & 1 & 0 & W^2 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix}$$

The Fast Fourier Transform (FFT

1. X(0) is computed by one complex mult. & one complex add.

2. X(1) is also one complex mult. & one complex add.

3. X(2) is computed by one complex add since $W^2 x(2) = -W^0 x(0)$ and $W^0 x(2)$ was already computed in 1.

4. X(3) is computed by one complex add.

$$\text{TOT} = \text{2 complex multi.} + \text{4 complex add.}$$

The Fast Fourier Transform (FFT)

Now,

$$\begin{bmatrix} X(0) \\ X(2) \\ X(1) \\ X(3) \end{bmatrix} = \begin{bmatrix} 1 & W^0 & 0 & 0 \\ 1 & W^2 & 0 & 0 \\ 0 & 0 & 1 & W^1 \\ 0 & 0 & 1 & W^3 \end{bmatrix} \begin{bmatrix} X_1(0) \\ X_1(1) \\ X_1(2) \\ X_1(3) \end{bmatrix}$$

The Fast Fourier Transform (FFT)

Now,

$$\begin{bmatrix} X(0) \\ X(2) \\ X(1) \\ X(3) \end{bmatrix} \Leftarrow$$

Note that the elements are not in order.

Reverse the bits in the binary representation of the index of each element to determine the unscrambled locations.

The Fast Fourier Transform (FFT)

1. X(0) needs 1 complex mult & 1 complex add.
2. X(2) needs 1 complex add since
3. X(1) needs 1 complex mult. & 1 complex add.
4. X(3) needs 1 complex add.

TOT= 2 complex mult + 4 complex add

The Fast Fourier Transform (FFT)

All in all, need 4 complex mult & 8 complex add in the FFT algorithm.

Comparing DFT & FFT,

DFT: complex mult. and N(N-1) complex add.

FFT: $N\gamma/2$ complex mult and $N\gamma$ complex add.

$2^{\gamma} = N$

The Fast Fourier Transform (FFT)

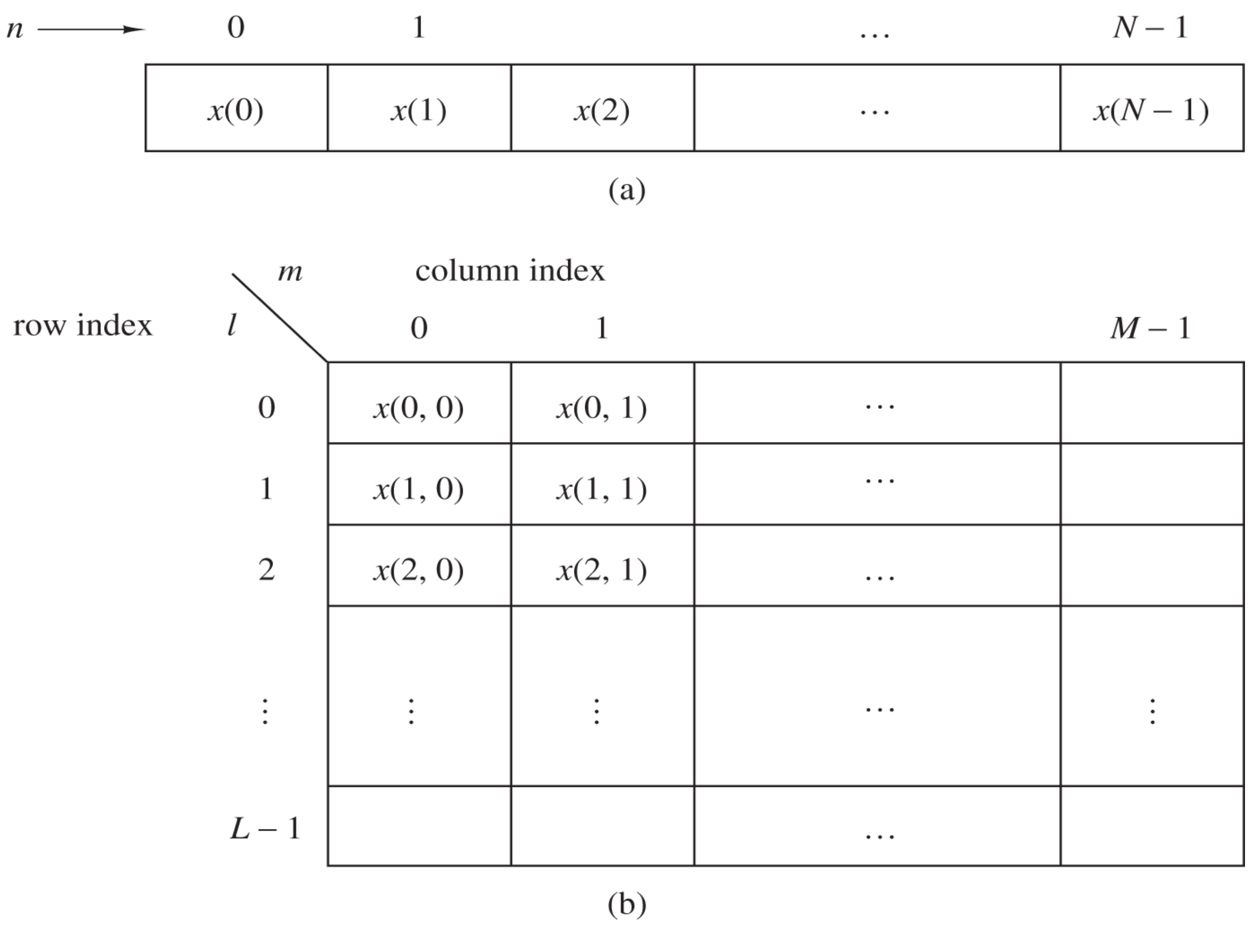
In short, the FFT is a procedure for factoring an NxN matrix into γ matrices (each NxN), s.t. these matrices minimize the number of multiplications and additions.

For the inverse FFT, let

$$W = e^{j2\pi/N}.$$

and proceed as before.

Divide and Conquer Approach to DFT Computation



$$N = LM$$

Figure 8.1.1 Two dimensional data array for storing the sequence $x(n)$, $0 \le n \le N - 1$.

Divide and Conquer Approach to DFT Computation

Row-wise $n = Ml + m$

$l \backslash m$	0	1	2		$M-1$
0	$x(0)$	$x(1)$	$x(2)$	$\cdots$	$x(M-1)$
1	$x(M)$	$x(M+1)$	$x(M+2)$	$\cdots$	$x(2M-1)$
2	$x(2M)$	$x(2M+1)$	$x(2M+2)$	$\cdots$	$x(3M-1)$
	$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$
$L-1$	$x((L-1)M)$	$x((L-1)M+1)$	$x((L-1)M+2)$	$\cdots$	$x(LM-1)$

(a)

$$n = Ml + m$$

Column-wise $n = l + mL$

$l \backslash m$	0	1	2		$M-1$
0	$x(0)$	$x(L)$	$x(2L)$	$\cdots$	$x((M-1)L)$
1	$x(1)$	$x(L+1)$	$x(2L+1)$	$\cdots$	$x((M-1)L+1)$
2	$x(2)$	$x(L+2)$	$x(2L+2)$	$\cdots$	$x((M-1)L+2)$
	$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$
$L-1$	$x(L-1)$	$x(2L-1)$	$x(3L-1)$	$\cdots$	$x(LM-1)$

(b)

$$n = l + mL$$

Figure 8.1.2 Two arrangements for the data arrays.

Divide and Conquer Approach to DFT Computation

$$k = Mp + q$$

$$k = qL + p$$

$$X(p,q) = \sum_{m=0}^{M-1} \sum_{l=0}^{L-1} x(l,m) W_N^{(Mp+q)(mL+l)}$$

$$W_N^{(Mp+q)(mL+l)} = W_N^{MLmp} W_N^{mLq} W_N^{Mpl} W_N^{lq}$$

$$X(p,q) = \boxed{\sum_{l=0}^{L-1} \left\{ \boxed{W_N^{lq} \left[\boxed{\sum_{m=0}^{M-1} x(l,m) W_M^{mq}} \right]} \right\} W_L^{lp}}$$

Compute M-point DFTs

Create rectangular array

Compute L-point DFTs

Divide and Conquer Approach to DFT Computation

From straight DFT requires N^2 complex multiplications and $N(N-1)$ complex additions

to

Complex multiplication: $N = (M + L + 1)$

Complex additions: $N = (M + L - 2)$

For N=1000, L=2, M=500, from 10^6 complex multiplications to 503,000 complex multiplications; complex additions is also reduced by about a factor of 2.

Divide and Conquer Approach to DFT Computation

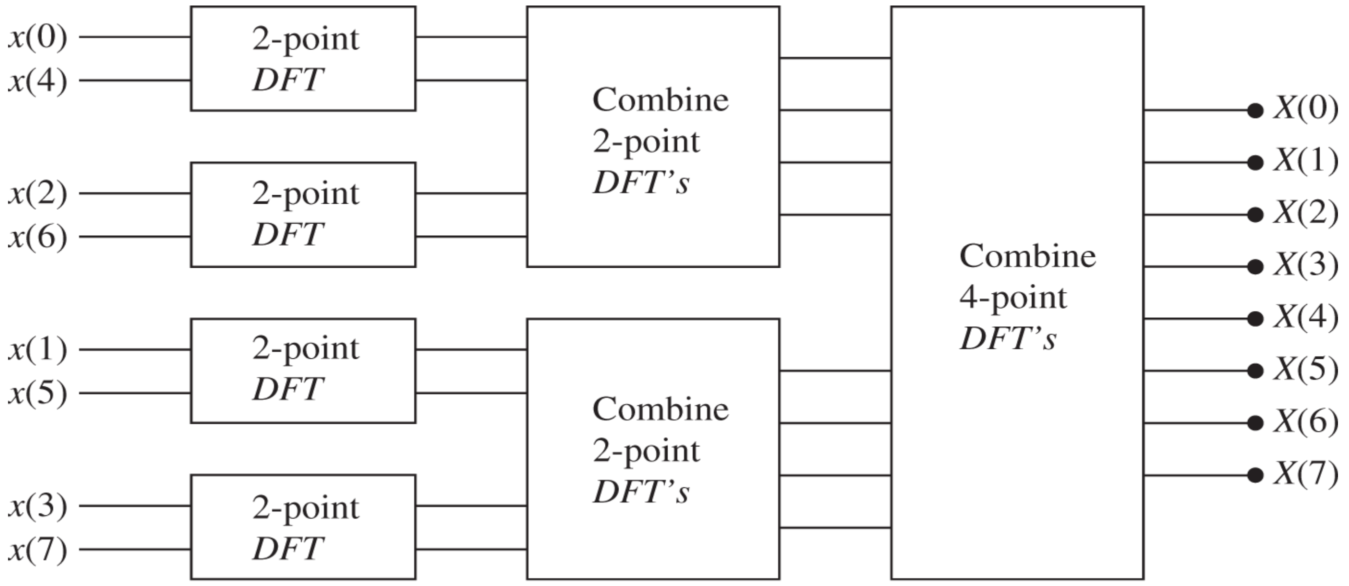


Figure 8.1.5 Three stages in the computation of an $N = 8$-point DFT.

Divide and Conquer Approach to DFT Computation

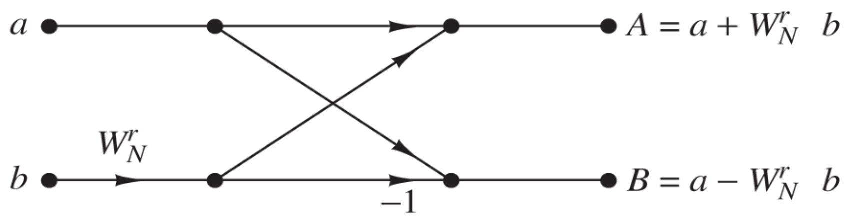
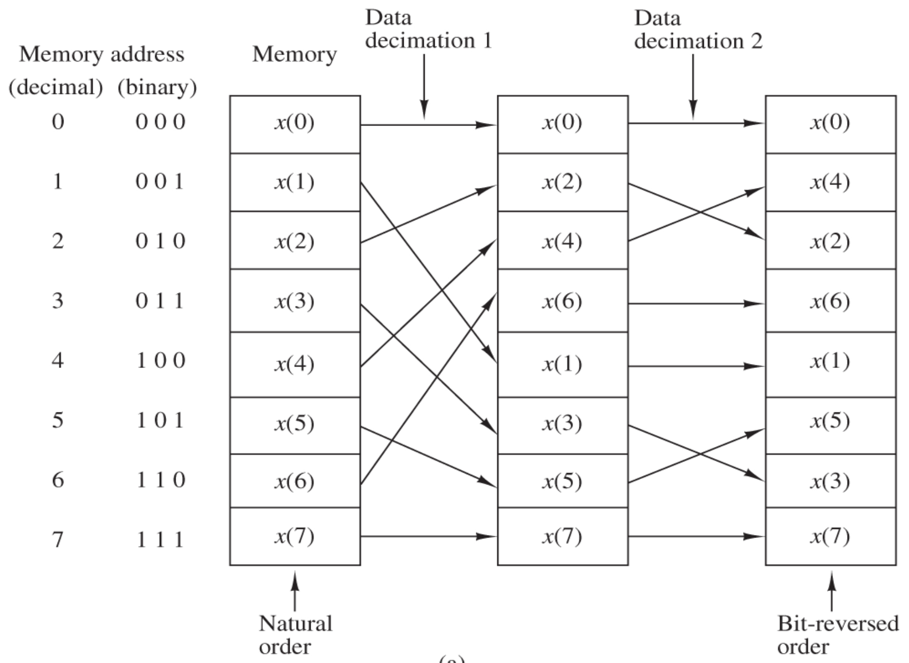


Figure 8.1.7 Basic butterfly computation in the decimation-in-time FFT algorithm.

Divide and Conquer Approach to DFT Computation



(a)

$(n_2 n_1 n_0)$	$\rightarrow$	$(n_1 n_0 n_2)$	$\rightarrow$	$(n_0 n_1 n_2)$
(0 0 0)	$\rightarrow$	(0 0 0)	$\rightarrow$	(0 0 0)
(0 0 1)	$\rightarrow$	(0 1 0)	$\rightarrow$	(1 0 0)
(0 1 0)	$\rightarrow$	(1 0 0)	$\rightarrow$	(0 1 0)
(0 1 1)	$\rightarrow$	(1 1 0)	$\rightarrow$	(1 1 0)
(1 0 0)	$\rightarrow$	(0 0 1)	$\rightarrow$	(0 0 1)
(1 0 1)	$\rightarrow$	(0 1 1)	$\rightarrow$	(1 0 1)
(1 1 0)	$\rightarrow$	(1 0 1)	$\rightarrow$	(0 1 1)
(1 1 1)	$\rightarrow$	(1 1 1)	$\rightarrow$	(1 1 1)

(b)

Figure 8.1.8 Shuffling of the data and bit reversal.

Simple and Efficient FFT Implementation in C++

```
FFT(x) {
   n=length(x);
   if (n==1) return x;
   m = n/2;
   X = (x_{2j})_{j=0}^{m-1};
  Y = (x_{2j+1})_{j=0}^{m-1};
   X = FFT(X);
  Y = FFT(Y);
   U = (X_{k mod m})_{k=0}^{n-1};
  V = (g^{-k}Y_{k mod m})_{k=0}^{n-1};
   return U+V;
}
```

From: http://www.drdobbs.com/cpp/a-simple-and-efficient-fft-implementatio/199500857

Efficient Computation of the DFT of Two Real Sequences

$$x(n) = x_1(n) + jx_2(n), \qquad 0 \le n \le N-1$$

$$X(k) = X_1(k) + jX_2(k)$$

$$x_1(n) = \frac{x(n) + x^*(n)}{2} \qquad x_2(n) = \frac{x(n) - x^*(n)}{2j}$$

$$X_1(k) = \frac{1}{2}\left\{\mathrm{DFT}\left[x(n)\right] + \mathrm{DFT}\left[x^*(n)\right]\right\}$$

$$X_2(k) = \frac{1}{2j}\left\{\mathrm{DFT}\left[x(n)\right] - \mathrm{DFT}\left[x^*(n)\right]\right\}$$

Efficient Computation of the DFT of Two Real Sequences

$$x(n) = x_1(n) + jx_2(n), \qquad 0 \le n \le N-1$$

$$X_1(k) = \frac{1}{2}\left[X(k) + X^*(N-k)\right]$$

$$X_2(k) = \frac{1}{j2}\left[X(k) - X^*(N-k)\right]$$

Efficient Computation of the DFT of a *2N*-Point Real Sequence

$$x_1(n) = g(2n)$$
$$x_2(n) = g(2n+1)$$

$$x(n) = x_1(n) + jx_2(n) \quad \Longrightarrow \quad \begin{aligned} X_1(k) &= \frac{1}{2}\left[X(k) + X^*(N-k)\right] \\ X_2(k) &= \frac{1}{2j}\left[X(k) - X^*(N-k)\right] \end{aligned}$$

Decimation-in-time FFT produces

$$G(k) = X_1(k) + W_2^k N X_2(k), \qquad k = 0,1,\ldots,N-1$$
$$G(k+N) = X_1(k) - W_2^k N X_2(k), \qquad k = 0,1,\ldots,N-1$$

The Goertzel Algorithm

$$X(k) = W_N^{-kN} \sum_{m=0}^{N-1} x(m) W_N^{km} = \sum_{m=0}^{N-1} x(m) W_N^{-k(N-m)}$$

$$y_k(n) = \sum_{m=0}^{N-1} x(m) W_N^{-k(n-m)}$$

$$h_k(n) = W_N^{-kn} u(n)$$

$$X(k) = y_k(n)\big|_{n=N}$$

$$H_k(z) = \frac{1}{1 - W_N^{-k} z^{-1}}$$

$$y_k(n) = W_N^{-k} y_k(n-1) + x(n), \qquad y_k(-1) = 0$$

The Goertzel Algorithm

$$H_k(z) = \frac{1 - W_N^k z^{-1}}{1 - 2\cos(2\pi k/N) z^{-1} + z^{-2}}$$

$$\upsilon_k(n) = 2\cos\frac{2\pi k}{N}\upsilon_k(n-1) - \upsilon_k(n-2) + x(n)$$

$$y_k(n) = \upsilon_k(n) - W_N^k \upsilon_k(n-1)$$

The Goertzel Algorithm

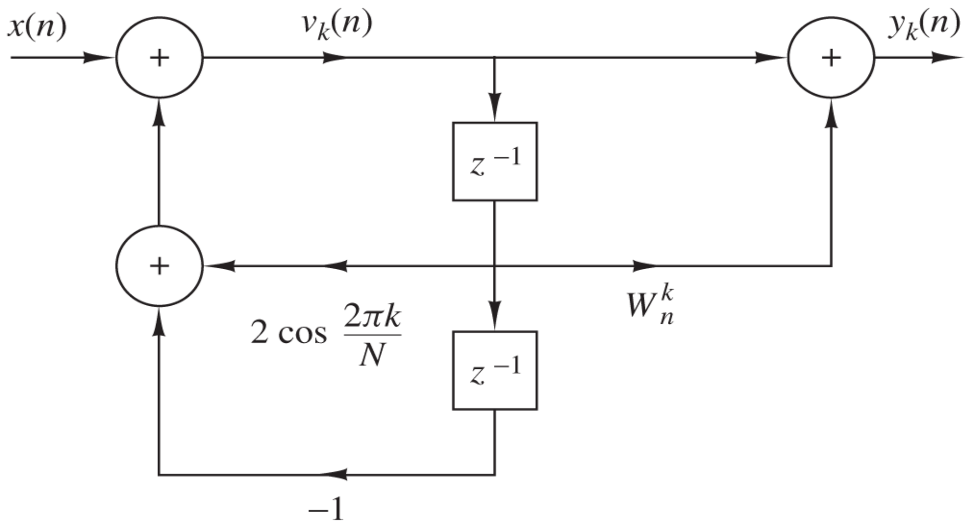


Figure 8.3.1 Direct form II realization of two-pole resonator for computing the DFT.

The Chirp-*z* Transform Algorithm

$$X(z_k) = \sum_{n=0}^{N-1} x(n) z_k^{-n}, \qquad k = 0,1,\ldots,L\text{-}1 \qquad 8.3.10$$

$$z_k = re^{j2\pi kn/N}, \qquad k = 0,1,2,\ldots,N-1$$

$$X(z_k) = \sum_{n=0}^{N-1} \left[x(n) r^{-n} \right] e^{-j2\pi kn/N}, \qquad k = 0,1,2,\ldots,N-1 \qquad 8.3.11$$

$$z_k = r_0 e^{j\theta_0} \left(R_0 e^{j\phi_0} \right)^k, \qquad k = 0,1,\ldots,L-1 \qquad 8.3.12$$

The Chirp-z Transform Algorithm

$$X(z_k) = \sum_{n=0}^{N-1} x(n) z_k^{-n} \qquad 8.3.13$$

$$= \sum_{n=0}^{N-1} x(n) \left(r_0 e^{j\theta_0}\right)^{-n} V^{-nk}$$

$$V = R_0 e^{j\phi_0} \qquad 8.3.14$$

$$nk = \frac{1}{2}\left[n^2 + k^2 - (k-n)^2\right] \qquad 8.3.15$$

$$X(z_k) = V^{-k^2/2} \sum_{n=0}^{N-1} \left[x(n) \left(r_0 e^{j\theta_0}\right)^{-n} V^{-n^2/2}\right] V^{(k-n)^2/2} \qquad 8.3.16$$

The Chirp-*z* Transform Algorithm

$$g(n) = x(n)\left(r_0 e^{j\theta_0}\right)^{-n} V^{-n^2/2} \qquad 8.3.17$$

$$X(z_k) = V^{-k^2/2}\sum_{n=0}^{N-1} g(n) V^{(k-n)^2/2} \qquad 8.3.18$$

$$h(n) = V^{n^2/2} \qquad 8.3.19$$

$$\begin{aligned} X(z_k) &= V^{-k^2/2} y(k) \\ &= \frac{y(k)}{h(k)}, \qquad k = 0,1,\ldots,L-1 \end{aligned} \qquad 8.3.20$$

The Chirp-*z* Transform Algorithm

$$y(k)=\sum_{n=0}^{N-1} g(n)h(k-n), \qquad k=0,1,\ldots,L-1 \qquad 8.3.21$$

$$h_1(n)=h(n-N+1), \qquad n=0,1,\ldots,M-1 \qquad 8.3.22$$

$$y(n)=y_1(n+N-1), \qquad n=0,1,\ldots,L-1 \qquad 8.3.23$$

$$h_2(n)=\begin{cases} h(n), & 0\le n\le L-1 \\ h(n-N-L+1), & L\le n\le M-1 \end{cases} \qquad 8.3.24$$

The Chirp-z Transform Algorithm

$$y(n) = y_2(n), \qquad n = 0,1,\ldots,L-1 \qquad 8.3.25$$

$$V^{-n^2/2} = e^{-j\pi n^2/N}$$

$$= \cos\frac{\pi n^2}{N} - j\sin\frac{\pi n^2}{N} \qquad 8.3.26$$

$$h(n) = V^{n^2/2}$$

$$= \cos\frac{\pi n^2}{N} + j\sin\frac{\pi n^2}{N} \qquad 8.3.27$$

$$= h_r(n) + jh_i(n)$$

$$|X(z_k)| = |y(k)|, \qquad k = 0,1,\ldots,N-1 \qquad 8.3.28$$

The Chirp-*z* Transform Algorithm

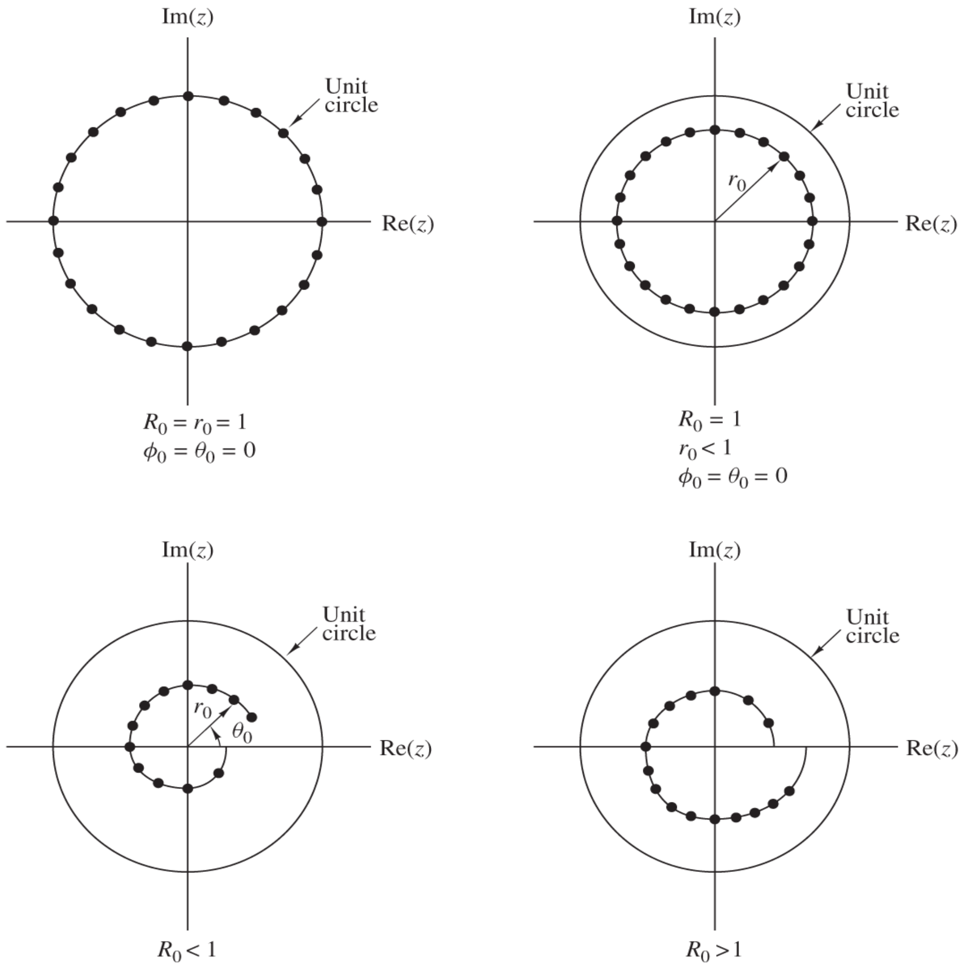


Figure 8.3.2 Some examples of contours on which we may evaluate the z-transform.

The Chirp-*z* Transform Algorithm

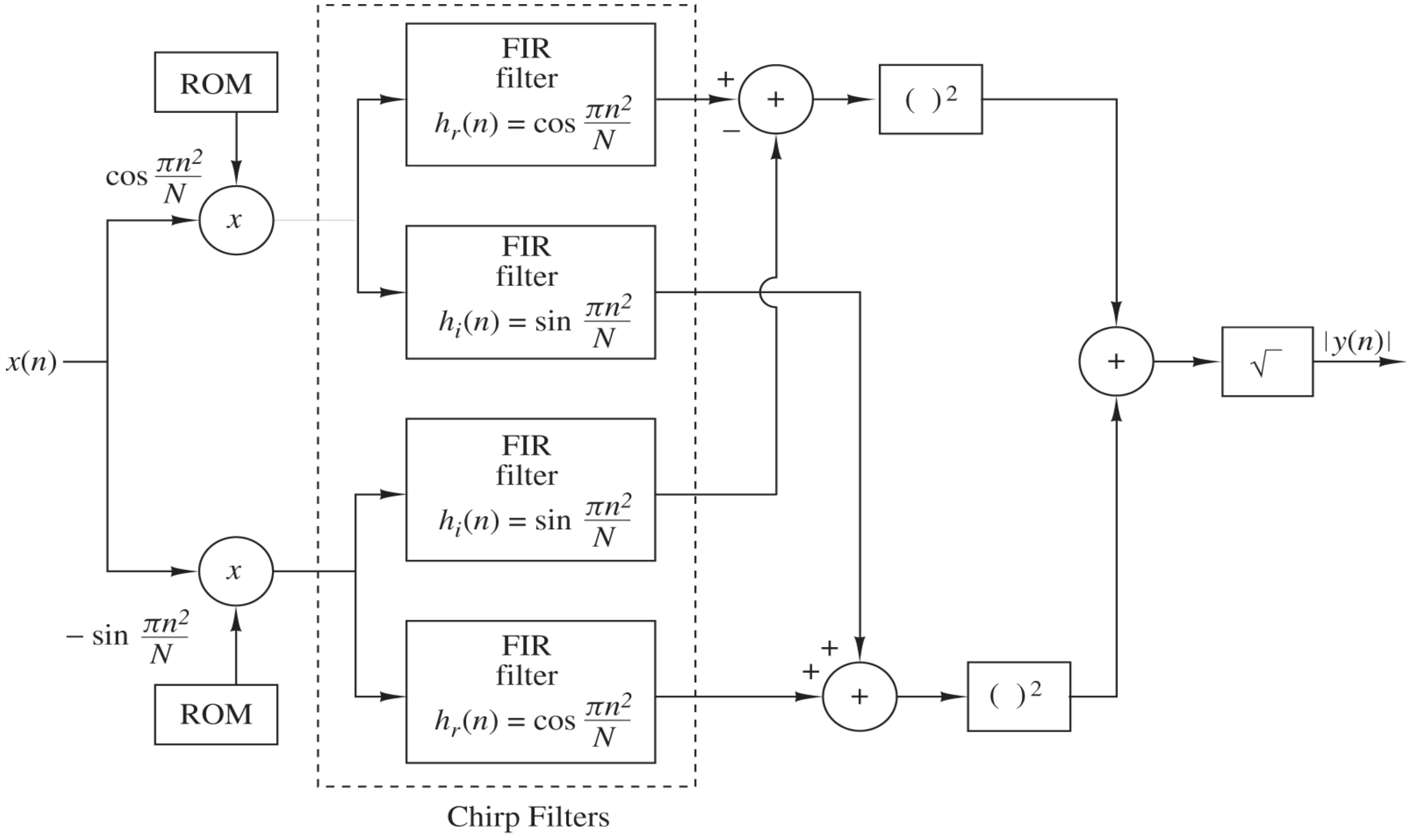

ROM
$\cos \frac{\pi n^2}{N}$
x
$x(n)$
x
$-\sin \frac{\pi n^2}{N}$
ROM
FIR filter $h_r(n) = \cos \frac{\pi n^2}{N}$
FIR filter $h_i(n) = \sin \frac{\pi n^2}{N}$
FIR filter $h_i(n) = \sin \frac{\pi n^2}{N}$
FIR filter $h_r(n) = \cos \frac{\pi n^2}{N}$
Chirp Filters
+
−
+
$(\ )^2$
+
+
+
$(\ )^2$
+
$\sqrt{\ }$
$|y(n)|$

Figure 8.3.3 Block diagram illustrating the implementation of the chirp-*z* transform for computing the DFT (magnitude only).

Example 1

8.4 The z-transform of the sequence $x(n) = u(n) - u(n\text{-}7)$ is sampled at five points on the unit circle as follows

$$x(k) = X(z)\Big|_{z=e^{j2\pi k/5}}, \qquad k = 0,1,2,3,4$$

Determine the inverse DFT $x'(n)$ of $X(k)$. Compare it with $x(n)$ and explain the results.

Example 1

$$
\begin{aligned}
X(z) &= 1 + z^{-1} + \ldots + z^{-6} \\
X(k) &= X(z)\big|_{z=e^{j\frac{2\pi}{5}}} \\
&= 1 + e^{-j\frac{2\pi}{5}} + e^{-j\frac{4\pi}{5}} + \ldots + e^{-j\frac{12\pi}{5}} \\
&= 2 + 2e^{-j\frac{2\pi}{5}} + e^{-j\frac{4\pi}{5}} + \ldots + e^{-j\frac{8\pi}{5}} \\
x'(n) &= \{2, 2, 1, 1, 1\} \\
x'(n) &= \sum_m x(n + 7m), \qquad n = 0, 1, \ldots, 4
\end{aligned}
$$

Temporal aliasing occurs in first two points of $x'(n)$ because $X(z)$ is not sampled at sufficiently small spacing on the unit circle.

Example 2

8.11 Show that the product of two complex numbers *(a+jb)* and *(c+jd)* can be performed with three real multiplications and five additions using the algorithm

$$x_R = (a-b)d + (c-d)a$$
$$x_I = (a-b)d + (c+d)b$$

Where $x = x_R + jx_I = (a+jb)(c+jd)$

Example 2

$$\begin{aligned}
x &= x_R + jx_I \\
&= (a+jb)(c+jd) \\
e &= (a-b)d \qquad \text{1 add , 1 mult} \\
x_R &= e + (c-d)a \qquad \text{2 adds 1 mult} \\
x_I &= e + (c+d)b \qquad \text{2 adds 1 mult} \\
\text{Total} & \qquad\qquad \text{5 adds 3 mult}
\end{aligned}$$