第4章 快速傅里叶变换(FFT)

4.1 引　言

DFT是数字信号分析与处理中的一种重要变换。但直接计算DFT的计算量与变换区间长度N的平方成正比，当N较大时，计算量太大，所以在快速傅里叶变换FFT(Fast Fourier Transform)出现以前，直接用DFT算法进行谱分析和信号的实时处理是不切实际的。

$$X(k)=\sum_{n=0}^{N-1}x(n)e^{-j2\pi nk/N} \qquad k=0,1,\cdots N-1$$

$$x(n)=\frac{1}{N}\sum_{k=0}^{N-1}X(k)e^{j2\pi nk/N} \qquad n=0,1,\cdots N-1$$

若 $x(n)$ 是 N点序列， 实现 $x(n)$ 的DFT，

即求出N个$X(k)$

需要：N^2 次复数乘法，$N(N-1)$ 次复数加法！

解决耗时的乘法问题是将数字信号处理理论用于实际的关键问题。特别是30年前，计算机的速度相当慢。因此，很多学者对解决DFT的快速计算问题产生了极大的兴趣。

Cooley J W, Tukey J W. An algorithm for the machine computation of complex Fourier series.

Mathematics of Computation, 1965, pp297~301

DSP的正式开端！

4.2 基2FFT算法

4.2.1 直接计算DFT的特点及减少运算量的基本途径

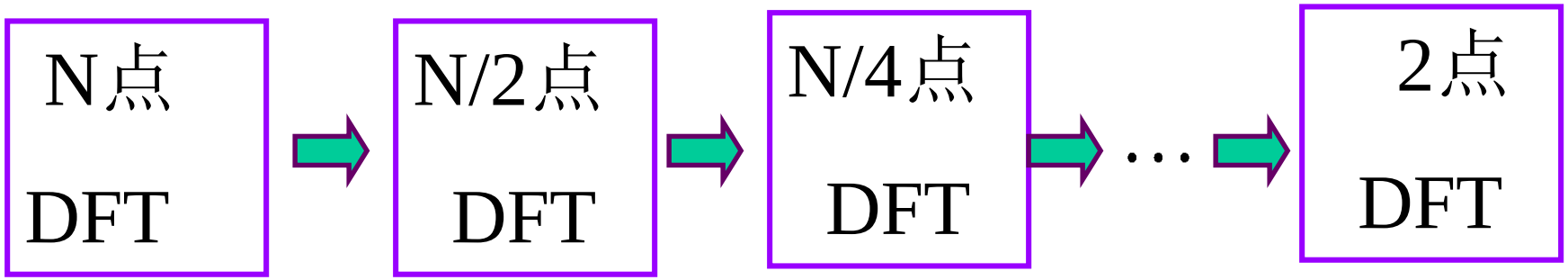
FFT 的思路：$X(k)=\sum_{n=0}^{N-1}x(n)W_N^{nk}\qquad k=0,1,\cdots N-1$

如何充分利用这些关系？

1. $W_N^{0}=1$
2. $W_N^{N}=1,\quad W_N^{mN}=1$
3. $W_N^{N+r}=W_N^{r}$
4. $W^{\frac{N}{2}}=-1$
5. $W_N^{\frac{N}{2}+r}=-W_N^{r}$

4.2.2 时域抽取法基2FFT基本原理

FFT的核心思想是：



1个　　2个　　4个　　N/2个

问题是如何分最有效？可以对时间变量分（DIT-FFT)，也可对频率变量分(DIF-FFT）

设序列$x(n)$的长度为N，且满足$N=2^M$，M为自然数。按n的奇偶把$x(n)$分解为

两个$N/2$点的子序列

$$x_1(r)=x(2r),\qquad r=0,\ 1,\ \cdots,\ \frac{N}{2}-1$$

$$x_2(r)=x(2r+1),\qquad r=0,\ 1,\ \cdots,\ \frac{N}{2}-1$$

则$x(n)$的DFT为

$$X(k)=\sum_{n=\text{偶数}}x(n)W_N^{kn}+\sum_{n=\text{奇数}}x(n)W_N^{kn}$$

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

$$=\sum_{r=0}^{N/2-1}x_1(r)W_N^{2kr}+W_N^k\sum_{r=0}^{N/2-1}x_2(r)W_N^{2kr}$$

因为 $W_N^{2kr} = \mathrm{e}^{-\mathrm{j}\frac{2\pi}{N}2kr} = \mathrm{e}^{-\mathrm{j}\frac{2\pi}{N/2}kr} = W_{N/2}^{kr}$

所以
$$X(k) = \sum_{r=0}^{N/2-1} x_1(r)W_{N/2}^{kr} + W_N^k \sum_{r=0}^{N/2-1} x_2(r)W_{N/2}^{kr}$$
$$= X_1(k) + W_N^k X_2(k) \qquad k = 0,1,2,\cdots,N-1$$

其中 $X_1(k)$和$X_2(k)$分别为$x_1(r)$和$x_2(r)$的$N/2$点DFT, 即
$$X_1(k) = \sum_{r=0}^{N/2-1} x_1(r)W_{N/2}^{kr} = \mathrm{DFT}[x_1(r)]_{N/2}$$
$$X_2(k) = \sum_{r=0}^{N/2-1} x_2(r)W_{N/2}^{kr} = \mathrm{DFT}[x_2(r)]_{N/2}$$

由于$X_1(k)$和$X_2(k)$均以$N/2$为周期，且$W_N^{k+\frac{N}{2}} = -W_N^k$ ，因此$X(k)$又可表示为

$$X(k) = X_1(k) + W_N^k X_2(k), \qquad k = 0,\ 1,\ \cdots,\ \frac{N}{2} - 1$$

$$X(k+\frac{N}{2}) = X_1(k) - W_N^k X_2(k), \qquad k = 0,\ 1,\ \cdots,\ \frac{N}{2} - 1$$

这样，就将N点DFT分解为两个$N/2$点DFT和上面两个式子的运算。

上式的运算可用图4.2.1所示的流图符号表示，称为蝶形运算符号。

可见，要完成一个蝶形运算，需要一次复数乘法和两次复数加法运算。

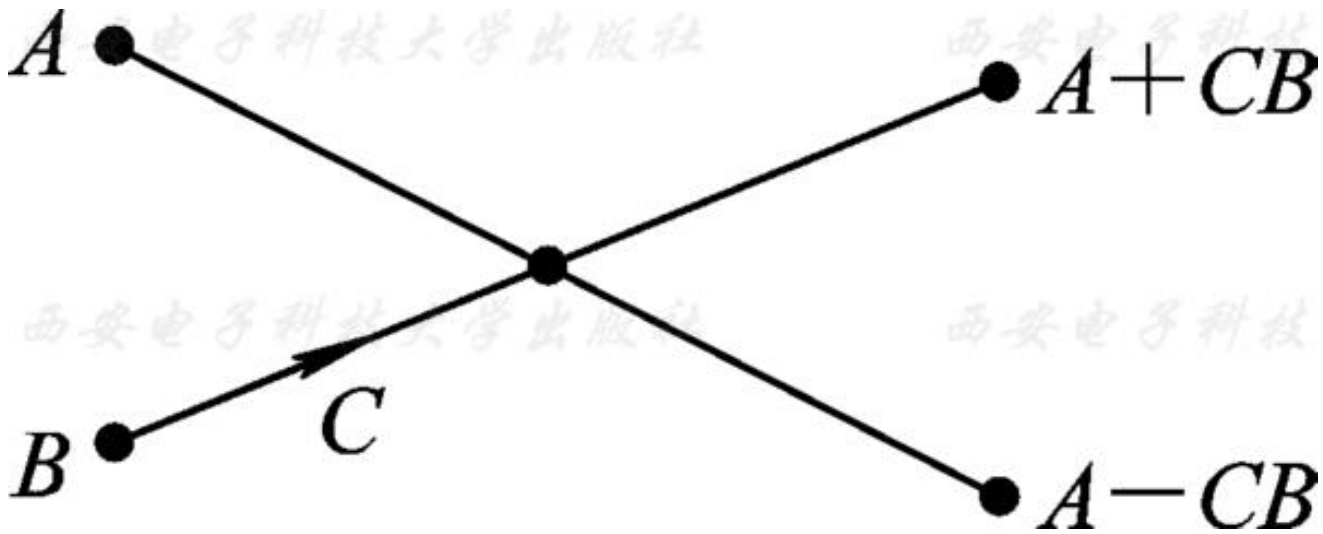


图4.2.1 蝶形运算符号

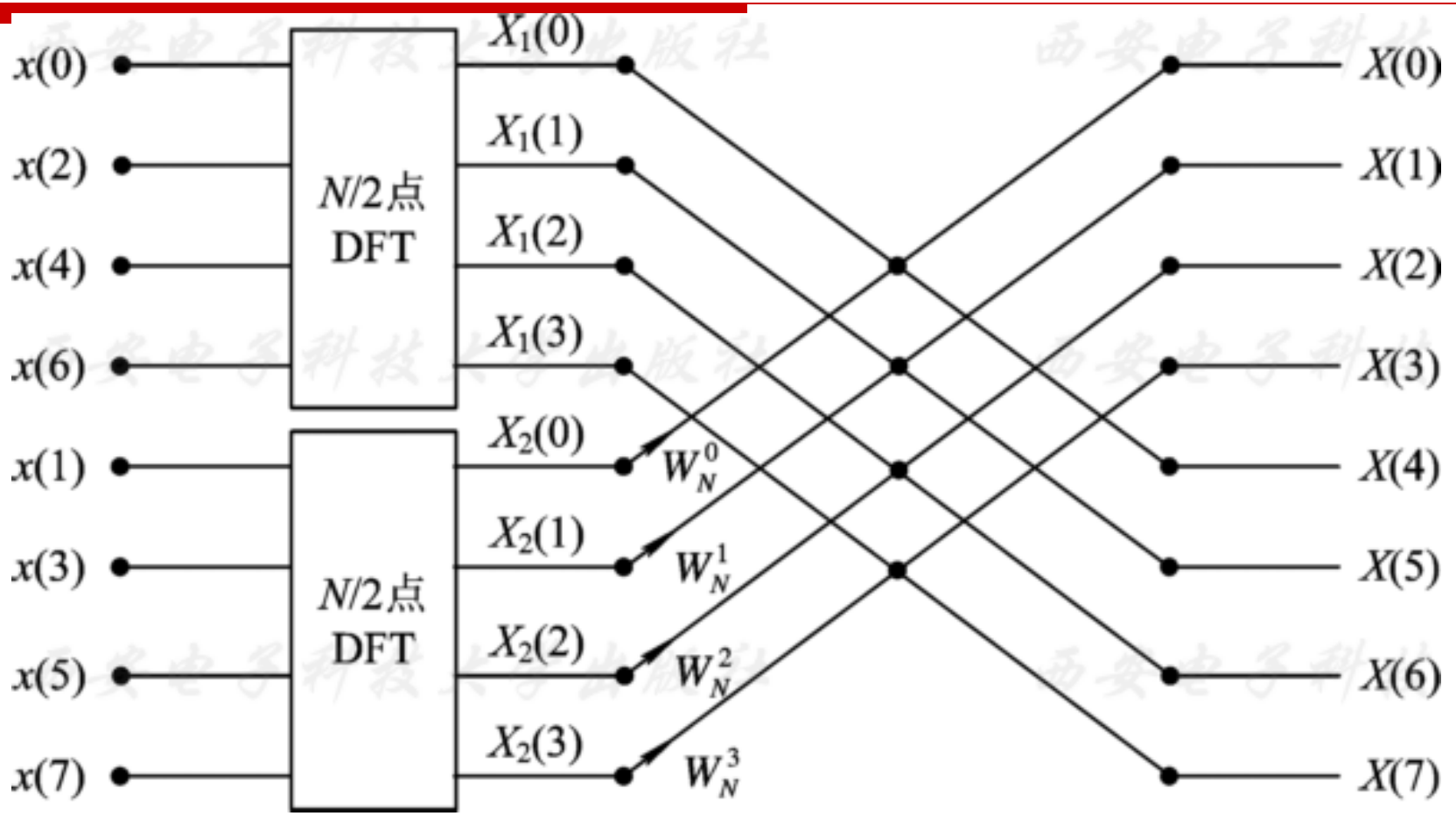


图4.2.2 8点DFT一次时域抽取分解运算流图

由图4.2.2容易看出，经过一次分解后，计算1个N点DFT共需要计算两个$N/2$点DFT和$N/2$个蝶形运算。而计算一个$N/2$点DFT需要$(N/2)^2$次复数乘法和$N/2(N/2-1)$次复数加法。所以，按图4.2.2计算N点DFT时，总共需要的复数乘法次数为

$$2\left(\frac{N}{2}\right)^2+\frac{N}{2}=\left.\frac{N(N+1)}{2}\right|_{N>>1}\approx\frac{N^2}{2}$$

复数加法次数为

$$N\left(\frac{N}{2}-1\right)+\frac{2N}{2}=\frac{N^2}{2}$$

由此可见，仅仅经过一次分解，就使运算量减少近一半。既然这样分解对减少DFT的运算量是有效的，且$N=2^M$, $N/2$仍然是偶数，故可以对$N/2$点DFT再作进一步分解。

与第一次分解相同，将$x_1(r)$按奇偶分解成两个$N/4$点的子序列$x_3(l)$和$x_4(l)$，即

$$\left.\begin{aligned}x_3(l)&=x_2(2l)\\x_4(l)&=x_1(2l+1)\end{aligned}\right\}\qquad l=0,\quad 1,\quad \cdots,\quad \frac{N}{4}-1$$

$X_1(k)$又可表示为

$$X_1(k)=\sum_{l=0}^{N/4-1}x_1(2l)W_{N/2}^{2kl}+\sum_{l=0}^{N/4-1}x_1(2l+1)W_{N/2}^{k(2l+1)}$$

$$=\sum_{l=0}^{N/4-1}x_3(l)W_{N/4}^{kl}+W_{N/2}^{k}\sum_{l=0}^{N/4-1}x_4(l)W_{N/4}^{kl}$$

$$=X_3(k)+W_{N/2}^{k}X_4(k)\qquad k=0,\quad 1,\quad \cdots,\quad \frac{N}{2}-1$$

式中

$$X_3(k)=\sum_{l=0}^{N/4-1}x_3(l)W_{N/4}^{kl}=\mathrm{DFT}[x_3(l)]_{N/4}$$

$$X_4(k)=\sum_{l=0}^{N/4-1}x_4(l)W_{N/4}^{kl}=\mathrm{DFT}[x_4(l)]_{N/4}$$

同理，由$X_3(k)$和$X_4(k)$的周期性和 $W_{N/2}^{m}$ 的对称性 $\left(W_{N/2}^{k+N/4}=-W_{N/2}^{k}\right)$

最后得到：

$$\left.\begin{aligned}X_1(k)&=X_3(k)+W_{N/2}^{k}X_4(k)\\X_1(k+N/4)&=X_3(k)-W_{N/2}^{k}X_4(k)\end{aligned}\right\},\quad k=0,\quad 1,\quad \cdots,\quad N/4-1$$

用同样的方法可计算出

$$\left.\begin{aligned} X_2(k) &= X_5(k) + W_{N/2}^{k} X_6(k) \\ X_2\left(k+\frac{N}{4}\right) &= X_5(k) - W_{N/2}^{k} X_6(k) \end{aligned}\right\} \quad k = 0,\quad 1,\quad \cdots,\quad \frac{N}{4}-1$$

其中

$$X_5(k) = \sum_{l=0}^{N/4-1} x_5(l) W_{N/4}^{kl} = \mathrm{DFT}[x_5(l)]_{N/4}$$

$$X_6(k) = \sum_{l=0}^{N/4-1} x_6(l) W_{N/4}^{kl} = \mathrm{DFT}[x_6(l)]_{N/4}$$

$$\left.\begin{aligned} x_5(l) &= x_2(2l) \\ x_6(l) &= x_2(2l+1) \end{aligned}\right\}, \quad l = 0,\quad 1,\quad \cdots,\quad N/4-1$$

这样，经过第二次分解，又将*N*/2点DFT分解为2个*N*/4点DFT和*N*/4个蝶形运算，如图4.2.3所示。

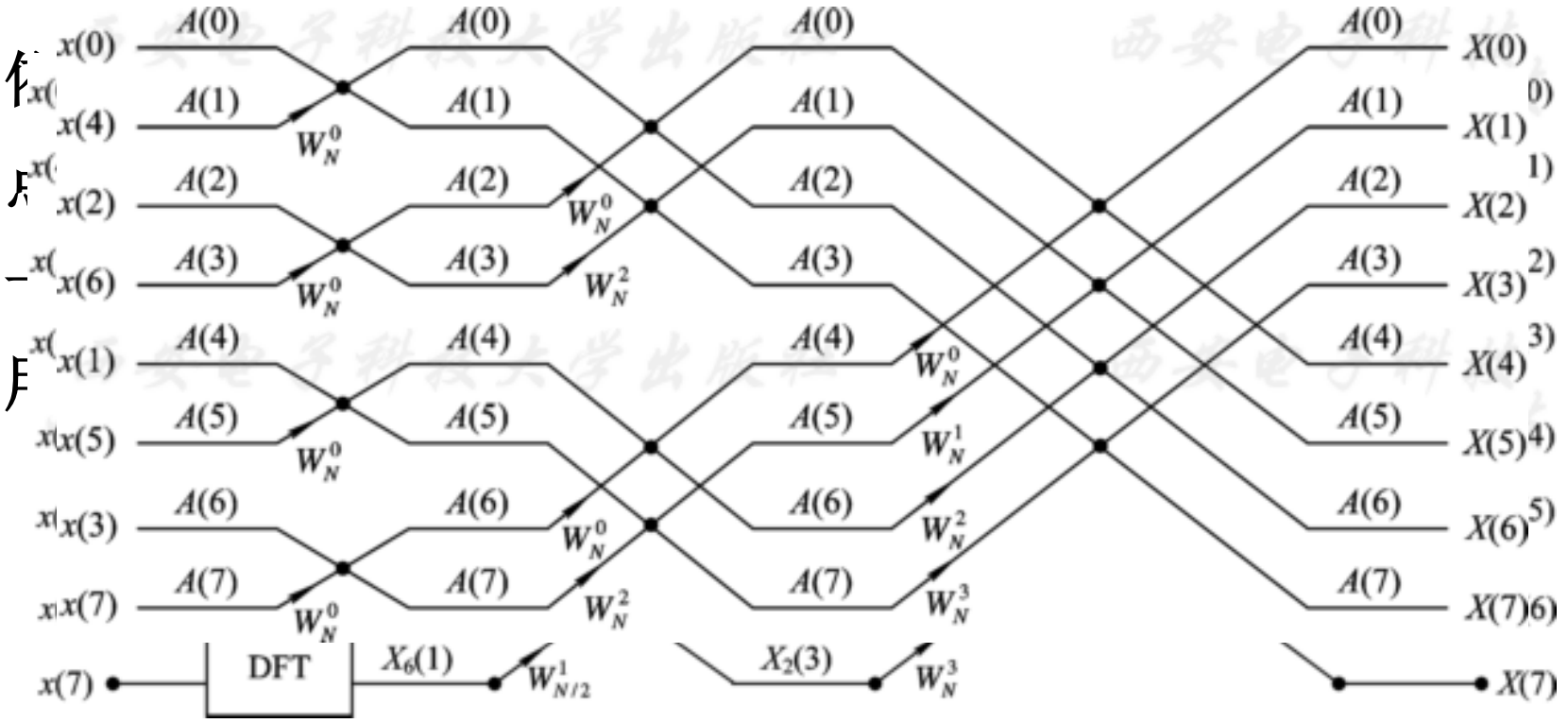


图4.2.3 8点DIT-FFT运算流图

4.2.3 DIT-FFT算法与直接计算DFT运算量的比较

由DIT-FFT算法的分解过程及图4.2.4可见，$N=2^M$ 时，其运算流图应有M级蝶形，每一级都由$N/2$个蝶形运算构成。因此，每一级运算都需要$N/2$次复数乘和N次复数加(每个蝶形需要两次复数加法)。所以，M级运算总共需要的复数乘次数为

$$C_M = \frac{N}{2} \cdot M = \frac{N}{2} \mathrm{lb} N$$

复数加次数为

$$C_A = N \cdot M = N \mathrm{lb} N$$

而直接计算DFT的复数乘为N^2次，复数加为$N(N-1)$次。当$N>>1$时，N^2(*N*/2) lb*N*，所以，DIT-FFT算法比直接计算DFT的运算次数大大减少。例如，$N=2^{10}=1024$时，

$$\frac{N^2}{\frac{N}{2}\mathrm{lb}N}=\frac{1048576}{5120}=204.8$$

这样，就使运算效率提高200多倍。图4.2.5为FFT算法和直接计算DFT所需复数乘法次数C_M与变换点数*N*的关系曲线。由此图更加直观地看出FFT算法的优越性，显然，N越大时，优越性就越明显。

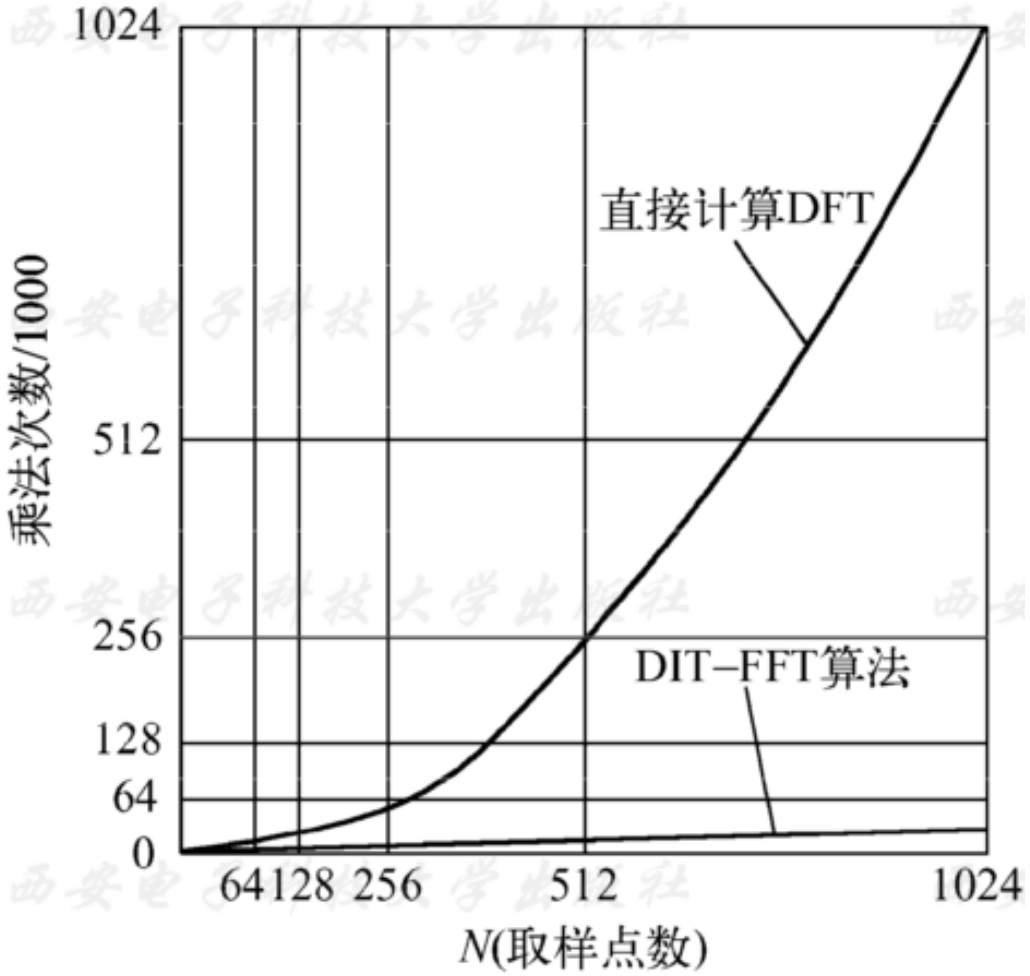


图4.2.5 DIT-FFT算法与直接计算DFT所需复数乘法次数的比较曲线

4.2.4 DIT-FFT的运算规律及编程思想

1. 原位计算

DIT-FFT的运算过程很有规律。$N=2^M$点的FFT共进行M级运算，每级由$N/2$个蝶形运算组成。

同一级中，每个蝶形的两个输入数据只对计算本蝶形有用，这就意味着计算完一个蝶形后，所得输出数据可立即存入原输入数据所占用的存储单元(数组元素)。

经过M级运算后，原来存放输入序列数据的N个存储单元(数组A)中便依次存放$X(k)$的N个值。

这种利用同一存储单元存储蝶形计算输入、输出数据的方法称为原位(址)计算。

原位计算可节省大量内存，从而使设备成本降低。

2．旋转因子的变化规律

***N*点DIT-FFT运算流图中，每级都有*N*/2个蝶形。**

每个蝶形都要乘以因子 W_N^p ，称其为旋转因子，***p*为旋转因子的指数。但各级的旋转因子和循环方式都有所不同。**

为了编写计算程序，应先找出旋转因子W_N^p与运算级数的关系。用***L***表示从左到右的运算级数(***L*=1，2，…，*M*)。观察图4.2.4不难发现，第*L*级共有2^{L-1}个不同的旋转因子。

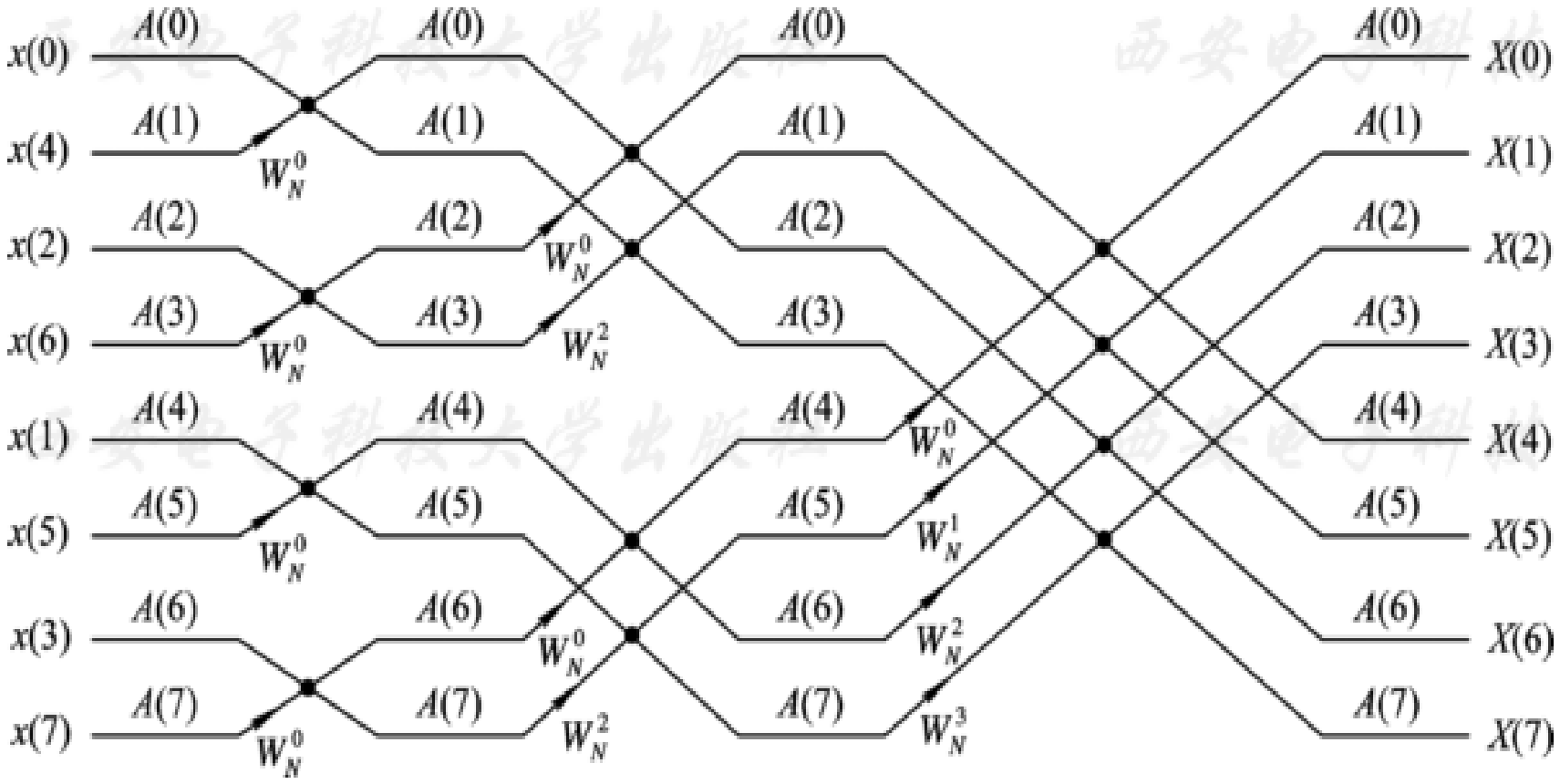


图4.2.4 8点DIT-FFT运算流图

$N=2^3=8$时的各级旋转因子表示如下：

$L=1$时

$$W_N^p = W_{N/4}^J = W_{2^L}^J \qquad J=0$$

$L=2$时

$$W_N^p = W_{N/2}^J = W_{2^L}^J \qquad J=0,\quad 1$$

$L=3$时

$$W_N^p = W_N^J = W_{2^L}^J \qquad J=0,\quad 1,\quad 2,\quad 3$$

对$N=2^M$的一般情况，第L级的旋转因子为

$$W_N^p = W_{2^L}^J, \qquad J = 0,\quad 1,\quad 2,\cdots,\quad 2^{L-1}-1$$

因为 $$2^L = 2^M \times 2^{L-M} = N \cdot 2^{L-M}$$

所以

$$W_N^p = W_{N\cdot 2^{L-M}}^J = W_N^{J\cdot 2^{M-L}} \qquad J = 0,\quad 1,\quad 2,\quad 2^{L-1}-1$$

$$p = J \cdot 2^{M-L}$$

这样，就可按(4.2.12)和(4.2.13)式确定第L级运算的旋转因子(实际编程序时，L为最外层循环变量)。

3. 蝶形运算规律

设序列$x(n)$经时域抽选(倒序)后，按图4.2.4所示的次序(倒序)存入数组A中。如果蝶形运算的两个输入数据相距B个点，应用原位计算，则蝶形运算可表示成如下形式：

$$A_L(J) \Leftarrow A_{L-1}(J) + A_{L-1}(J+B)W_N^p$$

$$A_L(J+B) \Leftarrow A_{L-1}(J) - A_{L-1}(J+B)W_N^p$$

式中

$$p = J \times 2^{M-L} \quad J = 0,1,\cdots,2^{L-1}-1; \quad L = 1,2,\cdots,M$$

下标L表示第L级运算，$A_L(J)$则表示第L级运算后的数组元素$A(J)$的值（即第L级蝶形的输出数据）。而$A_{L-1}(J)$表示第L级运算前$A(J)$的值（即第L级蝶形的输入数据）。

4. 编程思想及程序框图

第L级中，每个蝶形的两个输入数据相距$B=2^{L-1}$个点；每级有B个不同的旋转因子；同一旋转因子对应着间隔为2^L点的2^{M-L}个蝶形。

总结上述运算规律，便可采用下述运算方法。先从输入端(第1级)开始，逐级进行，共进行M级运算。在进行第L级运算时，依次求出B个不同的旋转因子，每求出一个旋转因子，就计算完它对应的所有2^{M-L}个蝶形。这样，我们可用三重循环程序实现DIT-FFT运算，程序框图如图4.2.6所示。

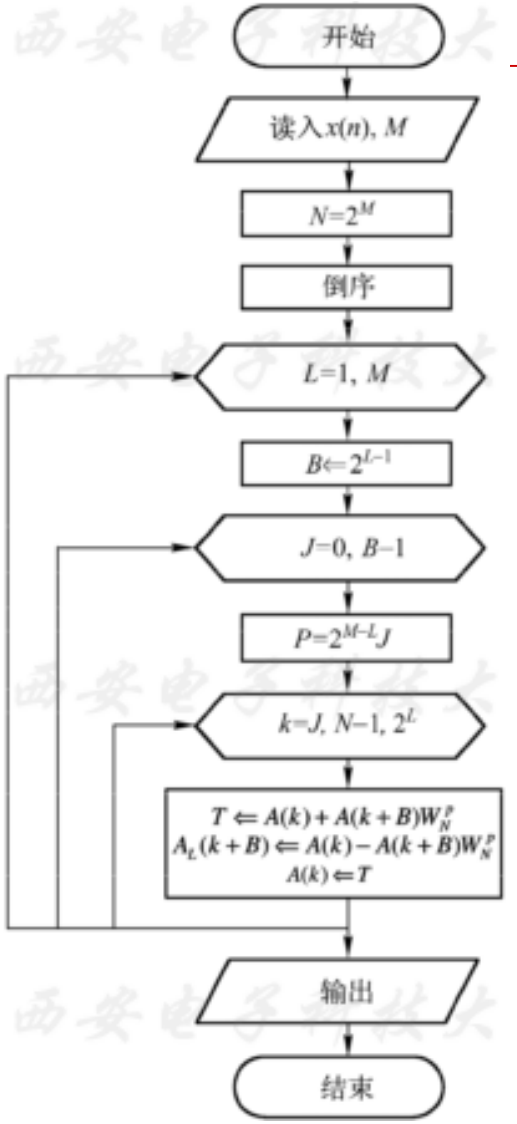
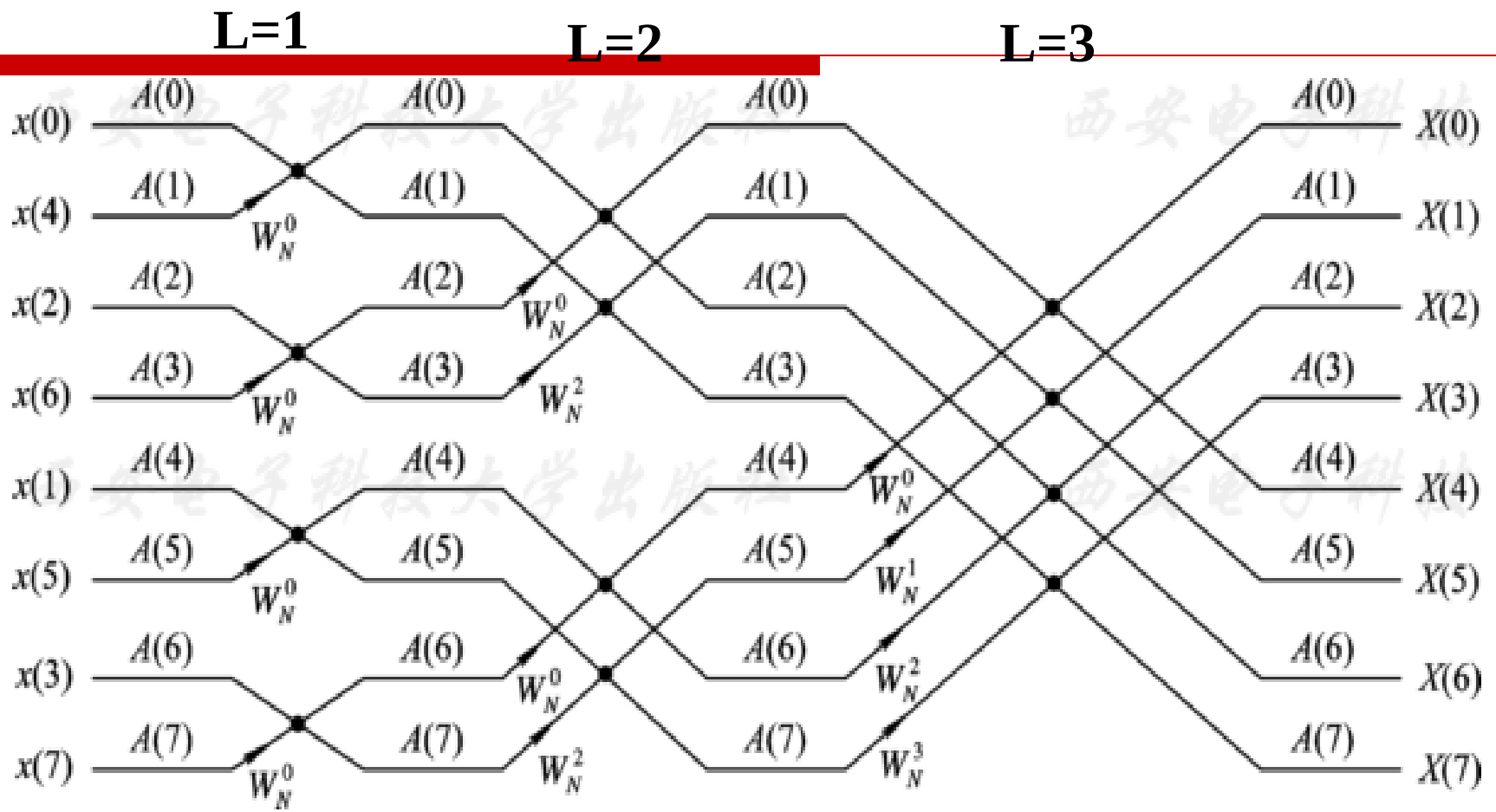


图4.2.6 DIT-FFT运算和程序框图

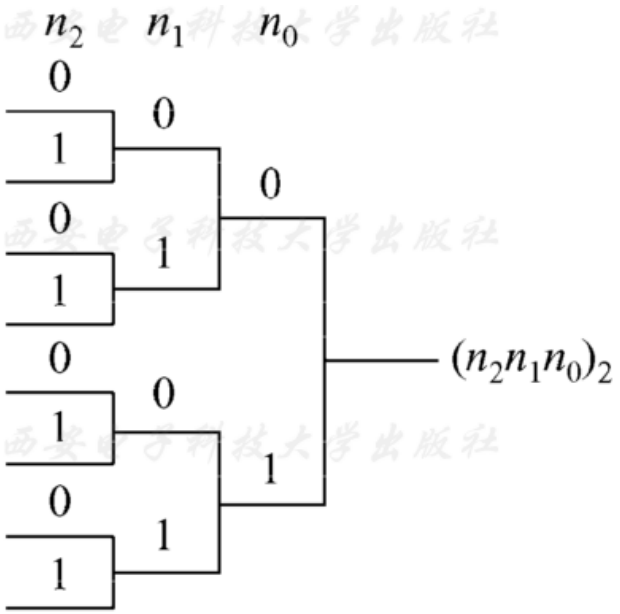


具有相同的旋转因子的蝶形运算单元间隔为2^L

同一个蝶形运算两个元素间隔为2^{L-1}

5. 序列的倒序

DIT-FFT算法的输入序列的排序看起来似乎很乱，但仔细分析就会发现这种倒序是很有规律的。由于$N=2^M$，因此顺序数可用M位二进制数$(n_{M-1}n_{M-2}\cdots n_1n_0)$表示。$M$次偶奇时域抽选过程如图所示。



000	0
100	4
010	2
110	6
001	1
101	5
011	3
111	7

表4.2.1 顺序和倒序二进制数对照表

顺　　序		倒　　序	
十进制数 I	二进制数	二进制数	十进制数 J
0	0 0 0	0 0 0	0
1	0 0 1	1 0 0	4
2	0 1 0	0 1 0	2
3	0 1 1	1 1 0	6
4	1 0 0	0 0 1	1
5	1 0 1	1 0 1	5
6	1 1 0	0 1 1	3
7	1 1 1	1 1 1	7

4.2.5 频域抽取法FFT(DIF-FFT)

在基2FFT算法中，频域抽取法FFT也是一种常用的快速算法，简称DTF- FFT。

设序列$x(n)$长度为$N=2^M$，首先将$x(n)$前后对半分开，得到两个子序列，其DFT可表示为如下形式：

$$
\begin{aligned}
X(k) &= \mathrm{DFT}[x(n)] = \sum_{n=0}^{N-1} x(n) W_N^{kn} \\
&= \sum_{n=0}^{N/2-1} x(n) W_N^{kn} + \sum_{n=N/2}^{N-1} x(n) W_N^{kn} \\
&= \sum_{n=0}^{N/2-1} x(n) W_N^{kn} + \sum_{n=0}^{N/2-1} x\left(n+\frac{N}{2}\right) W_N^{k(n+N/2)} \\
&= \sum_{n=0}^{N/2-1} \left[x(n) + W_N^{kN/2} x\left(n+\frac{N}{2}\right) \right] W_N^{kn}
\end{aligned}
$$

式中

$$W_N^{kN/2}=(-1)^k=\begin{cases}1, & k=\text{偶数}\\ -1, & k=\text{奇数}\end{cases}$$

将$X(k)$分解成偶数组与奇数组，当k取偶数($k=2m$, $m=0, 1, \cdots, N/2-1$)时

$$\begin{aligned}X(2m)&=\sum_{n=0}^{N/2-1}\left[x(n)+x\left(n+\frac{N}{2}\right)\right]W_N^{2mn}\\&=\sum_{n=0}^{N/2-1}\left[x(n)+x\left(n+\frac{N}{2}\right)\right]W_{N/2}^{mn}\end{aligned}\qquad(4.2.14)$$

当k取奇数($k=2m+1$, $m=0, 1, \cdots, N/2-1$)时，

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

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

(4.2.15)

令

$$\left.\begin{aligned}x_1(n)&=x(n)+x\left(n+\frac{N}{2}\right)\\ x_2(n)&=\left[x(n)-x\left(n+\frac{N}{2}\right)\right]W_N^n\end{aligned}\right\},\quad n=0,\quad 1,\quad 2,\quad \cdots,\quad \frac{N}{2}-1$$

将$x_1(n)$和$x_2(n)$分别代入(4.2.14)和(4.2.15)式，可得

$$\begin{cases} X(2m)=\sum_{n=0}^{N/2-1}x_1(n)W_{N/2}^{mn} \\ X(2m+1)=\sum_{n=0}^{N/2-1}x_2(n)W_{N/2}^{mn} \end{cases} \qquad (4.2.16)$$

表明，$X(k)$按奇偶k值分为两组，其偶数组是$x_1(n)$的$N/2$点DFT，奇数组则是$x_2(n)$的$N/2$点DFT。$x_1(n)$、$x_2(n)$和$x(n)$之间的关系也可用图4.2.10所示的蝶形运算流图符号表示。

$$\left.\begin{aligned} x_1(n) &= x(n)+x\left(n+\frac{N}{2}\right) \\ x_2(n) &= \left[x(n)-x\left(n+\frac{N}{2}\right)\right]W_N^n \end{aligned}\right\},\quad n=0,\ 1,\ 2,\ \cdots,\ \frac{N}{2}-1$$

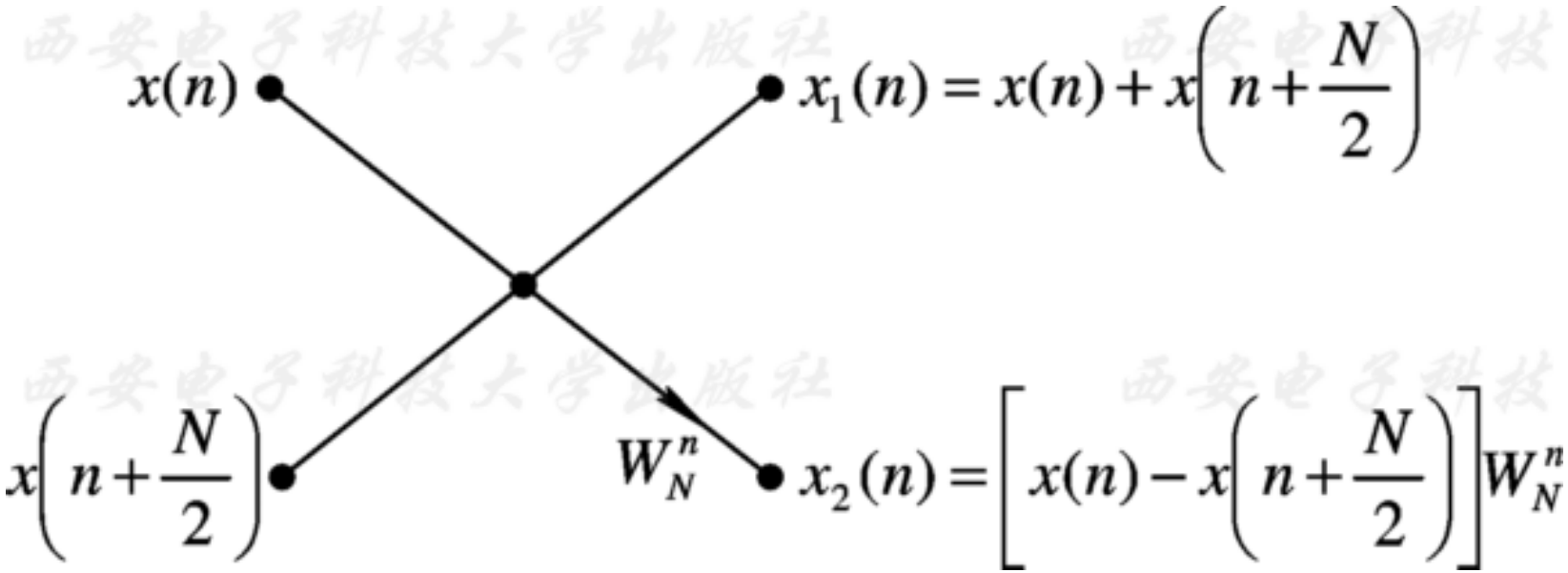


图4.2.10　　DTF－FFT蝶形运算流图符号

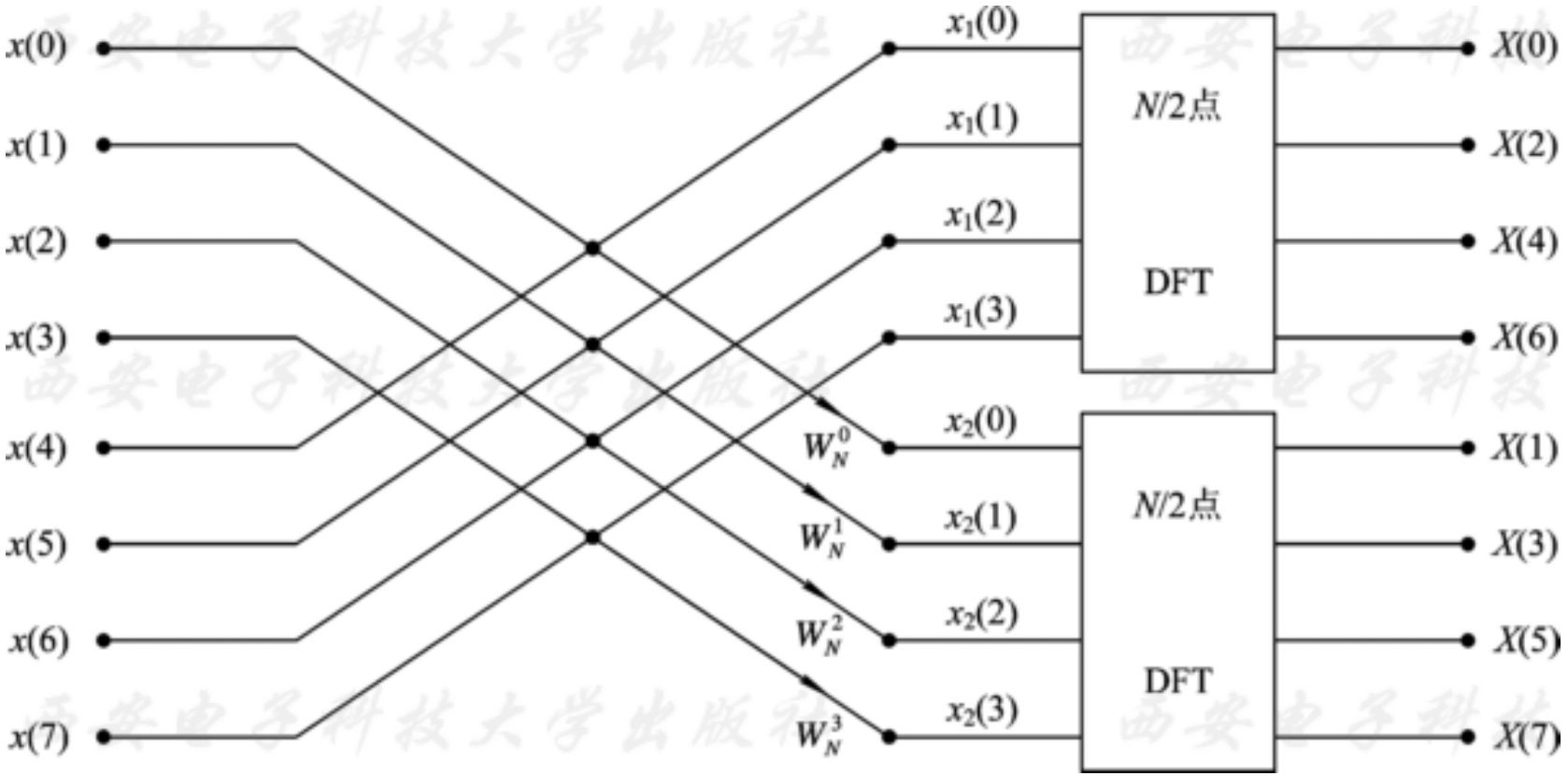


图4.2.11 DIF-FFT 第一次分解运算流图（N=8）

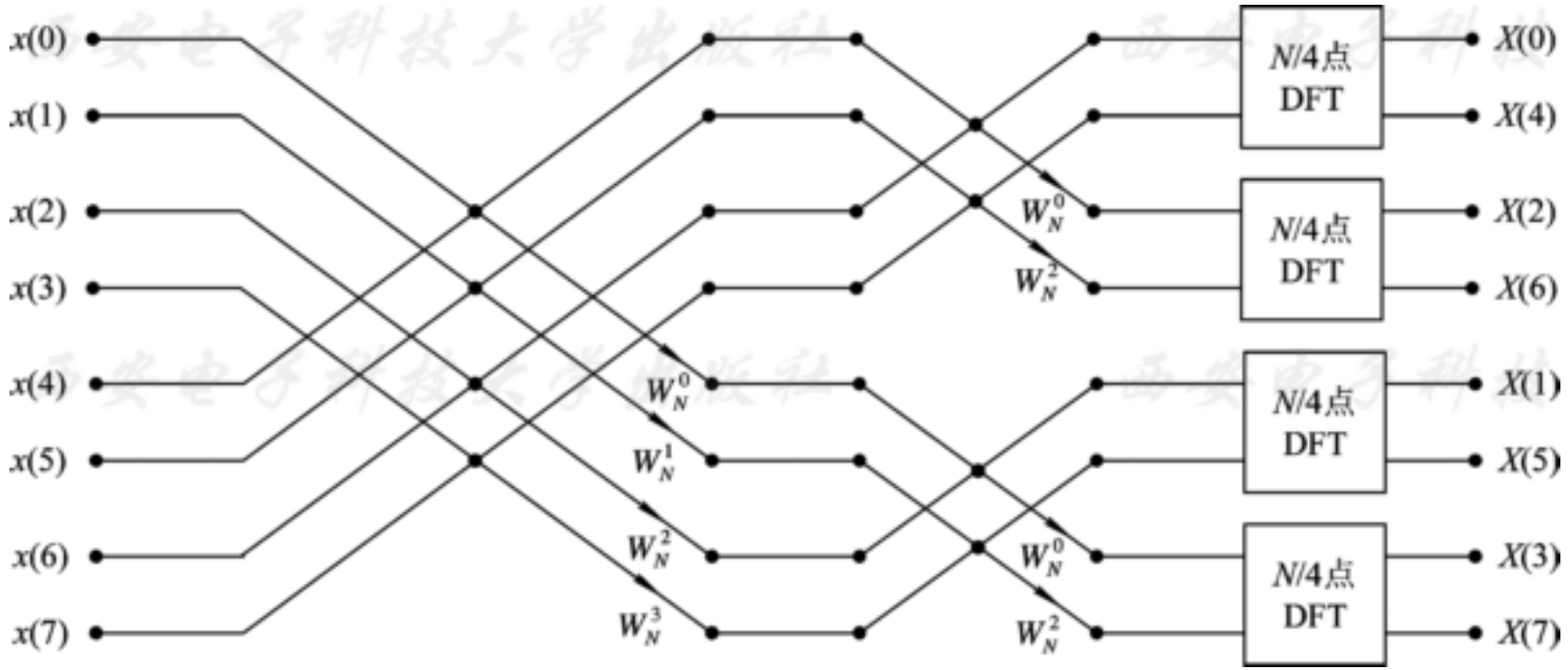


图4.2.12 DIF-FFT 第二次分解运算流图（$N = 8$）

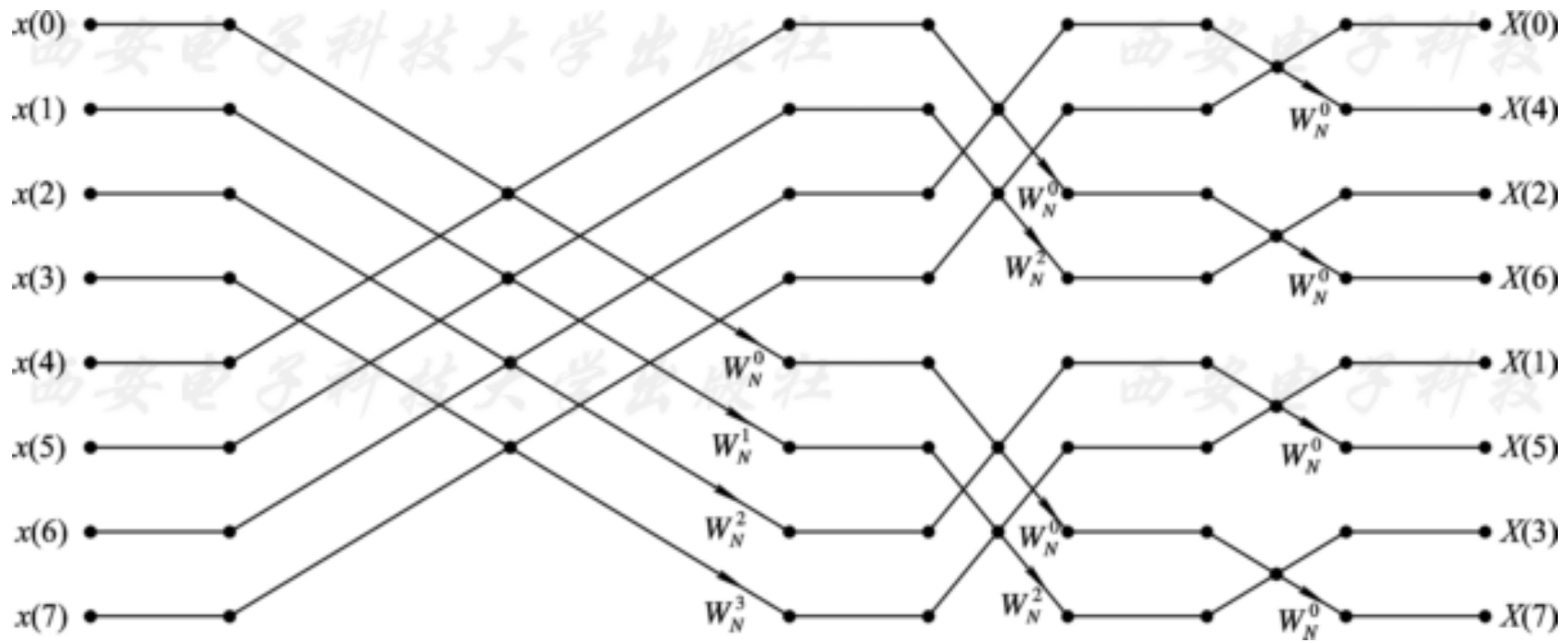


图4.2.13 DIF-FFT运算流图（N =8）

DIT-FFT和DIF-FFT算法的区别。

（1）蝶形运算的组成不同

DIT—FFT：旋转因子乘在蝶形运算的输入端

DIF—FFT：旋转因子乘在蝶形运算的输出端

（2）输入输出的顺序不同

DIT—FFT：输入倒序，输出顺序

DIF—FFT：输入顺序，输出倒序

4.2.6 IDFT的高效算法

上述FFT算法流图也可以用于计算IDFT。比较DFT和IDFT的运算公式：

$$X(k)=\mathrm{DFT}[x(n)]=\sum_{n=0}^{N-1}x(n)W_N^{kn}$$

$$x(n)=\mathrm{IDFT}[x(n)]=\frac{1}{N}\sum_{k=0}^{N-1}X(k)W_N^{-kn}$$

只要将DFT运算式中的系数 W_N^{kn} 改变为 W_N^{-kn}，最后乘以1/N，就是IDFT运算公式。

所以，只要将上述的DIT-FFT与DIF-FFT算法中的旋转因子 W_N^p 改为 W_N^{-p}，最后的输出再乘以1/N 就可以用来计算IDFT。只是现在流图的输入是$X(k)$，输出就是$x(n)$。

因此，原来的DIT-FFT改为IFFT后，称为DIF-IFFT更合适；DIF-FFT改为IFFT后, 应称为DIT-IFFT。

如果希望直接调用FFT子程序计算IFFT，则可用下面的方法：

由于

$$x(n)=\frac{1}{N}\left[\sum_{k=0}^{N-1}X^*(k)W_N^{kn}\right]^*=\frac{1}{N}\{DFT[X^*(k)]\}^*$$

所以，可以先将$X(k)$取复共轭，然后直接调用FFT子程序，或者送入FFT专用硬件设备进行DFT运算，最后取复共轭并乘以$1/N$得到序列$x(n)$。这种方法虽然用了两次取共轭运算，但可以与FFT共用同一子程序，因而用起来很方便。

第4章 快速傅里叶变换(FFT)

1.FFT的基本原理，DIT-FFT,DIF-FFT的图会画。

2.DFT和FFT的计算量。

3.DIT-FFT和DIF-FFT算法的区别。