第 5 章

快速傅里叶变换

5.1 概　　述

离散傅里叶变换(DFT)和卷积是信号处理中两个最基本也是最常用的运算,它们涉及信号与系统的分析与综合这一广泛的信号处理领域。由第 3 章可知,卷积可化为 DFT 来实现,实际上其他许多算法,如相关、滤波、谱估计等也都可化为 DFT 来实现。当然,DFT 也可化为卷积来实现。由后面的讨论可知,它们之间有着互通的关系。

对 N 点序列 $x(n)$,其 DFT 变换对定义为

$$\begin{cases} X(k)=\sum_{n=0}^{N-1}x(n)W_N^{nk}, & k=0,1,\cdots,N-1, \quad W_N=\mathrm{e}^{-\mathrm{j}\frac{2\pi}{N}} \\ x(n)=\frac{1}{N}\sum_{k=0}^{N-1}X(k)W_N^{-nk}, & n=0,1,\cdots,N-1 \end{cases} \tag{5.1.1}$$

显然,求出 N 点 $X(k)$ 需要 N^2 次复数乘法及 $N(N-1)$ 次复数加法。众所周知,实现一次复数乘需要四次实数乘两次实数加,实现一次复数加则需要两次实数加。当 N 很大时,其计算量是相当可观的。例如,若 $N=1024$,则需要 1 048 576 次复数乘法,即4 194 304 次实数乘法。所需时间过长,难于“实时”实现。对于 2-D 图像处理,所需计算量更是大得惊人。

其实,在 DFT 运算中包含大量的重复运算。观察式(3.5.12) 的 $\boldsymbol{W}$ 矩阵,虽然其中有 N^2 个元素,但由于 W_N 的周期性,其中只有 N 个独立的值,即 W_N^0, W_N^1,…, W_N^{N-1},且在这 N 个值中有一部分取极简单的值。简而言之,W_N 因子的取值有如下特点:

① $W^0=1, W^{N/2}=-1$;

② $W_N^{N+r}=W_N^r, W^{N/2+r}=-W^r$。

例如,对 4 点 DFT,按式(5.1.1) 直接计算需 $4^2=16$ 次复数乘,按上述周期性及对称性,可写成如下矩阵形式:

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

将该矩阵的第二列和第三列交换，得

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

由此得出

$$\left.\begin{aligned} X(0) &= [x(0)+x(2)]+[x(1)+x(3)] \\ X(1) &= [x(0)-x(2)]+[x(1)-x(3)]W^1 \\ X(2) &= [x(0)+x(2)]-[x(1)+x(3)] \\ X(3) &= [x(0)-x(2)]-[x(1)-x(3)]W^1 \end{aligned}\right\} \tag{5.1.2}$$

这样，求出 4 点 DFT 实际上只需要一次复数乘法。问题的关键是如何巧妙地利用 W 因子的周期性及对称性，导出一个高效的快速算法。这一算法最早由 J.W. Cooley 和 J.W.Tukey 于1965 年提出[Coo65]。

Cooley 和 Tukey 提出的快速傅里叶变换算法(fast Fourier transform, FFT)使 N 点 DFT 的乘法计算量由 N^2 次降为$\frac{N}{2}\log_2 N$ 次。仍以 $N=1024$ 为例，计算量降为 5120 次，仅为原来的 4.88%。因此人们公认这一重要发现是数字信号处理发展史上的一个转折点，也可以称为一个里程碑。以此为契机，加之超大规模集成电路(VLSI)和计算机的飞速发展，使得数字信号处理的理论在过去的近 60 年中获得了飞速的发展，并广泛应用于众多的技术领域，显示了这一学科的巨大生命力。

自 Cooley-Tukey 算法提出之后，新的算法不断涌现，总的来说，快速傅里叶变换的发展方向有两个，一是针对 N 等于 2 的整数次幂的算法，如基 2 算法、基 4 算法、实因子算法和分裂基算法等，另一个是 N 不等于 2 的整数次幂的算法，它是以 Winograd 为代表的一类算法(素因子算法、Winograd 算法)。

可以证明，式(5.1.2)的 4 点 DFT 可以不用乘法而只用加法来实现，因此基 4 算法比基 2 算法更有效。1984 年提出的分裂基(split-radix)算法[Duh86]同时使用基 2 和基 4 算法，被认为是目前对 N 等于 2 的整数次幂中各类算法中最为理想的一种。

Winograd 算法(WFTA)和上述算法在理论上有着根本的差别，它是建立在下标映射和数论上的一套完全新颖的算法[Win76]。在实际应用上，所需乘法次数比 Cooley-Tukey 算法有了明显的减少，因此被认为是对 FFT 算法的一大贡献。但 WFTA 理论上比较复

杂,编程也比较困难,数据的长度受到较大的限制,在程序中,数据所占的内存及数据的传递次数也比 Cooley-Tukey 算法增加很多。随着计算机技术的发展,当执行一个乘法指令和执行一个加法指令所需的时间不是相差很多,而且数据的传递时间相对于运算时间也不能忽略不计时,WFTA 是否还具有突出的优点已经受到人们的质疑。但是,WFTA 的思路及理论价值,特别是与之有关的一套计算复杂性理论[Aus84],对研究和学习 FFT 这一课题的读者来说,都是应该了解的。

以上对自 1965 年后 FFT 的发展做了简要的概述,有兴趣的读者可参看文献[Duh90a],该文较好地讨论了这一问题。在这近 50 年发展的过程中,人们对 FFT 及 DFT 的历史渊源也发生了兴趣。这些年的研究成果发现,FFT 算法的历史可追溯到 200 多年之前德国数学家高斯的工作。在这 200 多年的历程中,DFT 及 FFT 经历了一个漫长而又有趣的过程,与之相关联的论文就有 2400 余篇[Hei84]。

本章重点讨论 DFT 的基 2 算法、分裂基算法和使频域细化的 CZT 算法,并简要讨论频域只取少数点的 Goertzel 算法。限于篇幅,在输入端或输出端仅取少数点的另一种算法,即 Pruning 算法和 Winograd 算法,本书不再讨论,其简要介绍可参看文献[Hgs03]。

5.2 时间抽取(DIT)基 2 FFT 算法

5.2.1 算法的推导

对式(5.1.1),令 $N=2^M$,M 为正整数,可将 $x(n)$ 按奇、偶分成两组,即令 $n=2r$ 及 $n=2r+1$,而 $r=0,1,\cdots,N/2-1$,于是

$$\begin{aligned} X(k) &= \sum_{r=0}^{N/2-1} x(2r)W_N^{2rk} + \sum_{r=0}^{N/2-1} x(2r+1)W_N^{(2r+1)k} \\ &= \sum_{r=0}^{N/2-1} x(2r)W_{N/2}^{rk} + W_N^k \sum_{r=0}^{N/2-1} x(2r+1)W_{N/2}^{rk} \end{aligned} \tag{5.2.1}$$

式中

$$W_{N/2} = \mathrm{e}^{-\mathrm{j}\frac{2\pi}{(N/2)}} = \mathrm{e}^{-\mathrm{j}4\pi/N}$$

令

$$A(k) = \sum_{r=0}^{N/2-1} x(2r)W_{N/2}^{rk}, \qquad k=0,1,\cdots,N/2-1 \tag{5.2.2a}$$

$$B(k) = \sum_{r=0}^{N/2-1} x(2r+1)W_{N/2}^{rk}, \qquad k=0,1,\cdots,N/2-1 \tag{5.2.2b}$$

那么

$$X(k)=A(k)+W_N^kB(k),\qquad k=0,1,\cdots,N/2-1 \tag{5.2.3a}$$

$A(k)$、$B(k)$ 都是 $N/2$ 点的 DFT，$X(k)$ 是 N 点 DFT，因此单用式(5.2.3a)表示 $X(k)$ 并不完全。但因

$$X(k+N/2)=A(k)-W_N^kB(k),\quad k=0,1,\cdots,N/2-1 \tag{5.2.3b}$$

所以用 $A(k)$、$B(k)$ 可完整地表示 $X(k)$。$N=8$ 时，$A(k)$、$B(k)$ 及 $X(k)$ 的关系如图 5.2.1 所示。

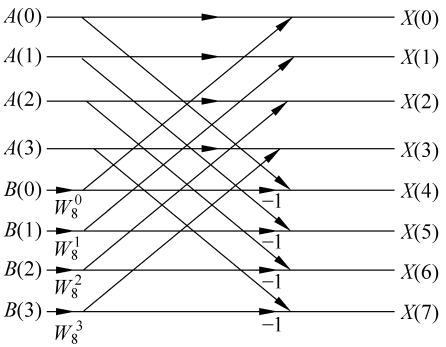

图 5.2.1　$N=8$ 时 $A(k)$、$B(k)$ 及 $X(k)$ 之间的关系

$A(k)$、$B(k)$ 仍是高复合数$(N/2)$ 的 DFT，可按上述方法继续给以分解。分别令 $r=2l$，$r=2l+1$，而 $l=0,1,\cdots,N/4-1$，则 $A(k)$ 和 $B(k)$ 可表示为

$$\begin{aligned}A(k)&=\sum_{l=0}^{N/4-1}x(4l)W_{N/2}^{2lk}+\sum_{l=0}^{N/4-1}x(4l+2)W_{N/2}^{(2l+1)k}\\&=\sum_{l=0}^{N/4-1}x(4l)W_{N/4}^{lk}+W_{N/2}^k\sum_{l=0}^{N/4-1}x(4l+2)W_{N/4}^{lk}\end{aligned}$$

令

$$C(k)=\sum_{l=0}^{N/4-1}x(4l)W_{N/4}^{lk},\qquad k=0,1,\cdots,N/4-1 \tag{5.2.4a}$$

$$D(k)=\sum_{l=0}^{N/4-1}x(4l+2)W_{N/4}^{lk},\qquad k=0,1,\cdots,N/4-1 \tag{5.2.4b}$$

那么

$$A(k)=C(k)+W_{N/2}^kD(k),\qquad k=0,1,\cdots,N/4-1 \tag{5.2.5a}$$

$$A\left(k+\frac{N}{4}\right)=C(k)-W_{N/2}^kD(k),\quad k=0,1,\cdots,N/4-1 \tag{5.2.5b}$$

同理，令

$$E(k)=\sum_{l=0}^{N/4-1}x(4l+1)W_{N/4}^{lk},\qquad k=0,1,\cdots,N/4-1 \tag{5.2.6a}$$

$$F(k)=\sum_{l=0}^{N/4-1}x(4l+3)W_{N/4}^{lk},\qquad k=0,1,\cdots,N/4-1 \tag{5.2.6b}$$

则

$$B(k)=E(k)+W_{N/2}^{k}F(k),\qquad k=0,1,\cdots,N/4-1 \tag{5.2.7a}$$

$$B\left(k+\frac{N}{4}\right)=E(k)-W_{N/2}^{k}F(k),\quad k=0,1,\cdots,N/4-1 \tag{5.2.7b}$$

若 $N=8$,这时 $C(k)$、$D(k)$、$E(k)$、$F(k)$都是二点的 DFT,无须再分,即

$$\begin{aligned}C(0)&=x(0)+x(4), & E(0)&=x(1)+x(5)\\ C(1)&=x(0)-x(4), & E(1)&=x(1)-x(5)\\ D(0)&=x(2)+x(6), & F(0)&=x(3)+x(7)\\ D(1)&=x(2)-x(6), & F(1)&=x(3)-x(7)\end{aligned}$$

若 $N=16,32$ 或 2 的更高的幂,可按上述方法继续分下去,直到两点的 DFT 为止。以上算法是将时间 n 按奇、偶分开,故称时间抽取算法(decimation in time,DIT)。现将上述过程示于图 5.2.2,其基本运算单元示于图 5.2.3。

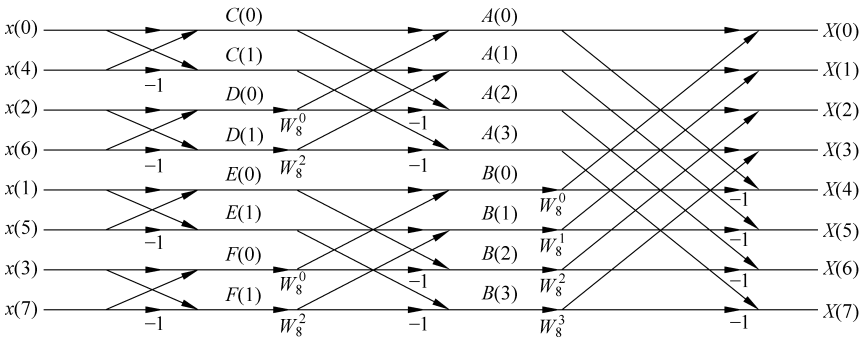

图 5.2.2 8 点 FFT 时间抽取算法信号流图

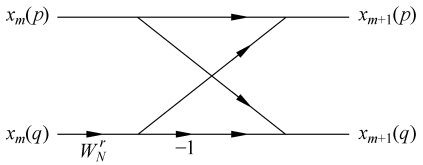

图 5.2.3 第 m 级蝶形单元

5.2.2 算法的讨论

现将 5.2.1 节所述的推导过程做一详细讨论,以期找到 FFT 算法的一般规律。

1. “级”的概念

上述推导过程，将 N 点 DFT 先分成两个 $N/2$ 点 DFT，再是 4 个 $N/4$ 点 DFT，进而是 8 个 $N/8$ 点 DFT，直至 $N/2$ 个两点 DFT。每分一次，称为一“级”运算。因为 $M=\log_2 N$，所以 N 点 DFT 可分成 M 级，如图 5.2.4 所示。图中 $N=8$，因此 $M=3$，从左至右，依次为 $m=0$ 级，$m=1$ 级，$m=2$ 级。

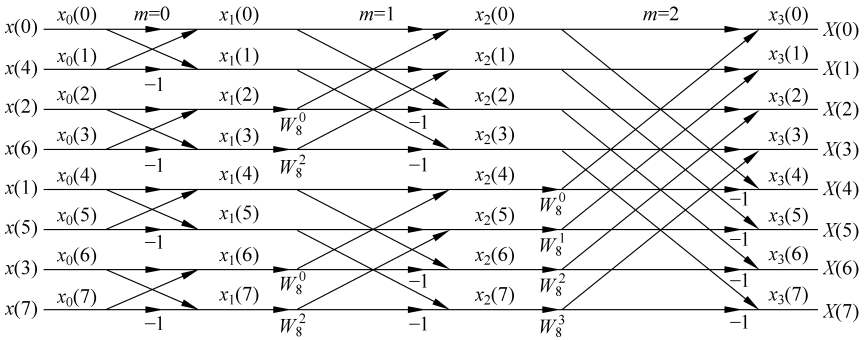

图 5.2.4　8 点 FFT 时间抽取算法信号流图

2. 蝶形单元

在图 5.2.2 中有大量如图 5.2.3 的运算结构，由于该运算结构的几何形状像蝴蝶，故称“蝶形运算单元”，在第 m 级，有

$$\left.\begin{aligned} x_{m+1}(p) &= x_m(p) + W_N^r x_m(q) \\ x_{m+1}(q) &= x_m(p) - W_N^r x_m(q) \end{aligned}\right\} \tag{5.2.8}$$

p、q 是参与本蝶形单元运算的上、下节点的序号。很明显，第 m 级序号为 p、q 的两点只参与这一个蝶形单元的运算，其输出在第 $m+1$ 级，且这一蝶形单元也不再涉及别的点。由于这一特点，在计算机编程时，我们可将蝶形单元的输出仍放在输入数组中，故称为“同址运算”。

由于每一级都含有 $N/2$ 个蝶形单元，每一个蝶形单元又只需要一次复数乘、两次复数加，因此，完成 $M=\log_2 N$ 级共需要的复数乘法数 M_c 和复数加法数 M_a 分别是

$$M_c = \frac{N}{2}\log_2 N = MN/2 \tag{5.2.9a}$$

$$M_a = N\log_2 N = MN \tag{5.2.9b}$$

将图 5.2.2 改画成图 5.2.4，让每一级的数据自上而下都按自然顺序排列，那么在第 m 级，上、下节点 p、q 之间的距离为

$$q - p = 2^m \tag{5.2.10}$$

3. “组”的概念

由图 5.2.4 可以看出,每一级的 $N/2$ 个蝶形单元可以分成若干组,每一组有着相同的结构及 W^r 因子分布。如 $m=0$ 级分成了四组,$m=1$ 级分成了两组,$m=M-1$ 级分成了一组。因此,第 m 级的组数是 $N/2^{m+1}$,$m=0,1,\cdots,M-1$。

4. W^r 因子的分布

再次考查式(5.2.1)～式(5.2.7)可以发现,第一次将 N 点 DFT 分成两个 $N/2$ 点 DFT 时,相当于图 5.2.4 的最右边一级,这时出现的 W^r 因子是 W_N^r,而 $r=0,1,\cdots,N/2-1$。再往下分时,依次是 $W_{N/2}^r,W_{N/4}^r,\cdots$,故每一级 W^r 因子分布的规律如下:

$$\begin{aligned}
&m=0\text{ 级},W_2^r, && r=0\\
&m=1\text{ 级},W_4^r, && r=0,1\\
&m=2\text{ 级},W_8^r, && r=0,1,2,3\\
&\quad\vdots && \quad\vdots\\
&m=M-1\text{ 级},W_N^r, && r=0,1,\cdots,N/2-1
\end{aligned}$$

因此,不难总结出 W^r 因子分布的一般规律:

$$\text{第 } m \text{ 级},W_{2^{m+1}}^r,\quad r=0,1,\cdots,2^m-1 \tag{5.2.11}$$

5. 码位倒置

由图 5.2.4 可以看出,变换后的输出序列 $X(k)$ 依照正序排列,但输入序列 $x(n)$ 的次序不再是原来的自然顺序,这正是由于将 $x(n)$ 按奇、偶分开所产生的。对 $N=8$,其自然序号是 0,1,2,3,4,5,6,7。第一次按奇、偶分开,得到两组 $N/2$ 点 DFT,$x(n)$ 的序号是

$$0,\ 2,\ 4,\ 6\quad|\quad 1,\ 3,\ 5,\ 7$$

对每一组再按奇、偶分开,这时应将每一组仍按自然顺序排列,故抽取后得 4 组,每组的序号是

$$0,\ 4\quad|\quad 2,\ 6\quad|\quad 1,\ 5\quad|\quad 3,\ 7$$

这一顺序正是图 5.2.4 输入端序列 $x(n)$ 的排列次序。掌握这一规律,对 N 为 2 的更高次幂,我们都可得到正确的抽取次序。

如果我们将 $x(n)$ 的序号 $n=0,1,\cdots,N-1$ 写成二进制,如 $N=8$,那么 $x(0),\cdots,x(7)$ 对应为

$$x(000),x(001),x(010),x(011),x(100),x(101),x(110),x(111)$$

将二进制数码翻转,得

$$x(000),x(100),x(010),x(110),x(001),x(101),x(011),x(111)$$

它们对应的十进制序号分别是

$$x(0),x(4),x(2),x(6),x(1),x(5),x(3),x(7)$$

这也正是按奇、偶抽取所得到的顺序。掌握了这一规律,我们就可以正确编程,FFT 的软

件已是通用程序，读者只要了解排序的规律即可。

5.3　频率抽取(DIF)基 2 FFT 算法

和 DIT 相对应，DIF 算法是将频域 $X(k)$ 的序号 k 按奇、偶分开。对式(5.1.1) 的 DFT，先将 $x(n)$ 按序号分成上、下两部分，得

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

式中，$W_N^{Nk/2} = (-1)^k$，分别令 $k = 2r$，$k = 2r+1$，而 $r = 0,1,\cdots,N/2-1$，于是得

$$X(2r) = \sum_{n=0}^{N/2-1} \left[x(n) + x\left(n+\frac{N}{2}\right) \right] W_{N/2}^{nr} \tag{5.3.1a}$$

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

$$r = 0,1,\cdots,\frac{N}{2}-1 \tag{5.3.1b}$$

令

$$g(n) = x(n) + x\left(n+\frac{N}{2}\right) \tag{5.3.2a}$$

$$h(n) = \left[x(n) - x\left(n+\frac{N}{2}\right) \right] W_N^n \tag{5.3.2b}$$

则

$$X(2r) = \sum_{n=0}^{N/2-1} g(n)W_{N/2}^{nr} \tag{5.3.3a}$$

$$X(2r+1) = \sum_{n=0}^{N/2-1} h(n)W_{N/2}^{nr} \tag{5.3.3b}$$

这样，就将一个 N 点 DFT 分成了两个 $N/2$ 点的 DFT，分的办法是将 $X(k)$ 按序号 k 的奇、偶分开。感兴趣的读者可以仿照时间抽取的办法继续分下去，直到得到两点的 DFT。图 5.3.1 给出了一个 16 点 DIF 算法流图，以备和其他的算法相比较。由该图可以看出，输入是正序，输出是按奇、偶分开的倒序。

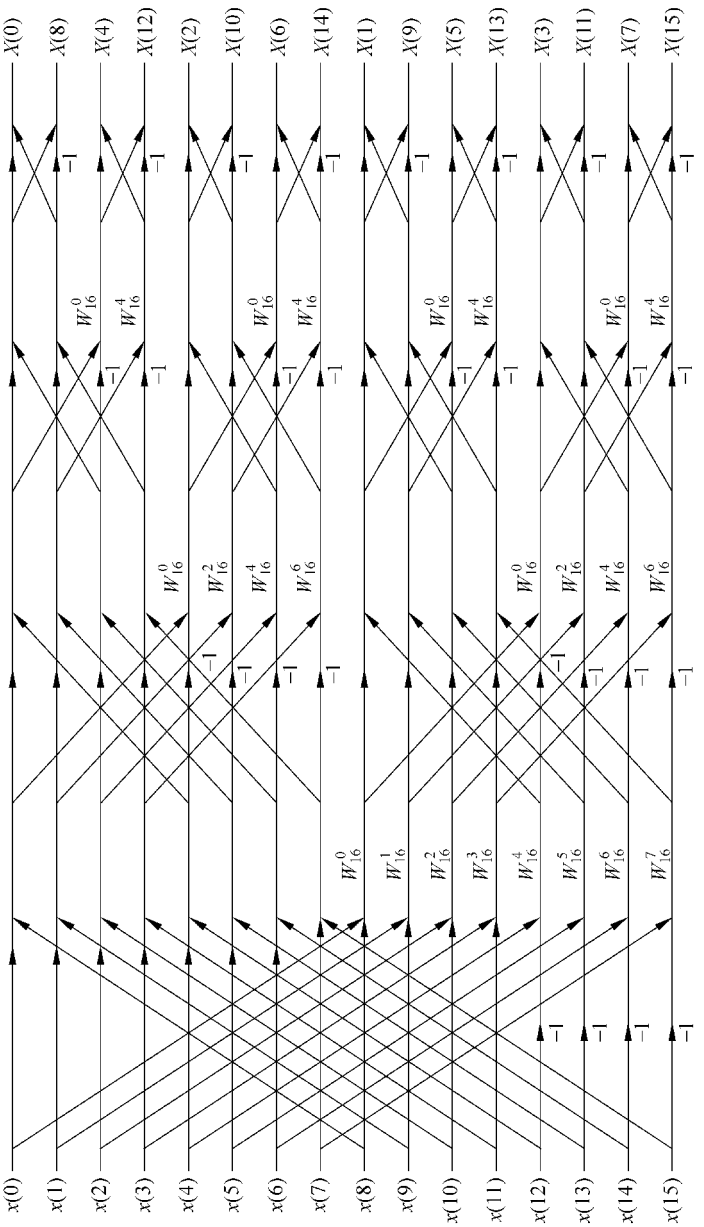

图 5.3.1 16 点 DIF 基 2 FFT 算法

不论是DIT算法还是DIF算法，都可以看作是将 $N\times N$ 的 $\boldsymbol{W}$ 矩阵作分解来实现的。将此留作练习，由读者自行完成。

5.4　进一步减少运算量的措施

在讨论其他算法之前，我们先研究一下，是否可以采取一些措施进一步减少计算量。

5.4.1　多类蝶形单元运算

对 $N=2^M$，共需进行 M 级运算，每级有 $N/2$ 个蝶形单元，而每个蝶形单元需要一次复数乘法，所以总共需要 $MN/2$ 次复数乘法。由式(5.2.11)可知，当 $m=0$ 时，即对第零级，所有的 W 因子的指数全为零，所以 $W^r=1$，这一级不需要乘法。对 $m=1$ 级，$W^r=1$ 或 $W^r=-\mathrm{j}$。我们知道，两个复数相乘时，若一个为纯虚数，则也不需要做乘法。在 DFT 中，W^r 又称旋转因子(twiddle factor)，像 W^0、W_2^1、W_4^1 这样的旋转因子又称无关紧要的旋转因子(trivial twiddle factor)。去掉前两级后，所需的复数乘法次数应是

$$M_c=\frac{N}{2}(M-2) \tag{5.4.1}$$

进一步分析，在 $m=2$ 级，每一组含有 W_8^0，W_8^2 这两个无关紧要的旋转因子，这一级共有 $(N/2^{2+1})=N/8$ 组，故这一级无关紧要的旋转因子数为 $N/4$ 个。以此类推，$m=3$ 时有 $N/8$ 个，最后一级，即 $m=M-1$ 时，有 $N/2^{M-1}$ 个。这样，从 $m=2$ 至最后一级共有

$$\frac{N}{4}+\frac{N}{8}+\cdots+\frac{N}{N/2}=\frac{N}{2}-2 \tag{5.4.2}$$

个无关紧要的旋转因子。这样，式(5.4.1)应改写为

$$M_c=\frac{N}{2}(M-3)+2 \tag{5.4.3}$$

读者可自己验证，所需要的复数加法量是

$$A_c=\frac{3}{2}N(M-1)+2 \tag{5.4.4}$$

前已述及，实现一次复数乘需要四次实数乘、两次实数加，但对 $W_8^1=(1-\mathrm{j})\sqrt{2}/2$ 这样特殊的复数，因为

$$(c+\mathrm{j}c)(x+\mathrm{j}y)=R+\mathrm{j}I$$

其中