快速付里叶变换

硬件实现、误差分析及通信应用译文选

FFT

清华大学无线电系快速付里叶变换组编译

快速付里叶变换

（硬件实现、误差分析及通信应用译文选）

清华大学无线电系快速付里叶变换组编译

人民邮电出版社

内 容 提 要

本书分三大部分。第一部分叙述用硬件实现快速付里叶变换（FFT）的各种方法，特别是讨论了提高速率的各种并行化处理方法。第二部分为FFT的误差分析，讨论了定点运算与浮点运算的舍入误差、量化误差及有限字长对运算的影响。第三部分是在通信中应用FFT的一些典型实例。可供对FFT和电子计算机的基本原理和语言有所了解的读者，在研究与使用FFT时作参考。

快 速 付 里 叶 变 换

（硬件实现、误差分析及通信应用译文选）

清华大学无线电系快速付里叶变换组编译

人民邮电出版社出版
北京东长安街27号
河北省邮电印刷厂印刷
新华书店北京发行所发行
各地新华书店经售

开本：850×1168 1/32　1980年1月河北第一版
印张：8　页数：128　1980年1月河北第一次印刷
字数：212千字　印数：1—8,700册

统一书号：15045·总2334—无683

定价：1.00元

编译前言

近几年来，随着计算技术和数字电路工艺的迅速发展，在信号处理中使用数字计算机和专用数字电路，已经成为一种明显的趋势。快速付里叶变换（简写为 *FFT*）是数字信号处理的主要工具之一。自1965年库利-图基（*Cooley—Tukey*）发明这一算法至今，时间虽然不长，它的发展速度却十分惊人。七十年代初，国外市场已开始有*FFT*的专用硬件出售。现在，*FFT* 处理机已经成为通用计算机的外围设备。

*FFT*所以得到如此迅速的发展，主要有以下几个原因：

首先，由于 *FFT* 把离散付里叶变换（*DFT*）的计算速度提高了N/log_2N倍（N是需要处理的数据样点数），使很多信号的处理工作能够与整个系统的运行速度协调，因此，它的应用就不再局限于某些数据的事后处理或某些系统的模拟研究，而被扩展到数据的实时处理，并开始用于控制系统。

其次，在发现 *FFT* 之前，虽然在语言通信、雷达和其它领域，早就认为可用数字的方法实现信号处理，但它主要用于极其复杂的信号处理工作，而且在速度、成本等方面都赶不上模拟系统。*FFT* 的发现，第一次说明了用数字系统分析频谱比模拟系统优越，这就为广泛应用数字方法处理信号打开了局面。

最后，*FFT* 算法之所以快速，其根本原因在于利用了原始变换矩阵的多余性。这一特性也适用于付里叶变换以外的其它一些变换。在*FFT*的影响下，人们对广义的快速变换产生了强烈的兴趣并进行了深入的研究，结果使各种快速正交变换在今天的数字信号处理中，占据了极其重要的地位。因此，*FFT*对数字信号处理的发展起

着变革性的作用，它改变了模拟信号处理中原有的一些概念和方法，使数字信号处理成为一门具有广泛用途的新技术。

近年来，FFT在国内也受到了普遍的重视。为此，我们选编了国外若干资料，供FFT研究和使用者参考。本书的读者应对FFT、电子计算机的基本原理和语言有所了解注(1)。全书主要分为三个部分，着重在FFT的实现及其在通信中的应用。兹介绍其内容如下：

第一部分是FFT的硬件实现。一般来说，用通用计算机可以实现FFT。但是，由于通用计算机只有单个运算单元，所以在计算FFT时的速度较低；又由于计算FFT没有充分发挥通用机的灵活性和多功能的特点，因而也造成了浪费。因此，现在大都是用专用硬件来实现FFT。这些专用硬件可以是专用的，也可以是通用计算机的外围设备。在后一种情况下，它可以是只包含一个与通用机联用的高速运算单元，也可以有自己的存贮器。这一部分叙述了用硬件实现FFT的各种方法，特别是讨论了提高速率的各种并行化处理方法。

第二部分为FFT的误差分析。由于数字计算机或专用处理机只能使用有限字长，因而所有数值仅能用有限位精度来实现。同时，所有算术运算的结果还必须舍入，这在计算付里叶系数时，必然会带来误差，其影响不容忽视。

这一部分收集了有关FFT误差分析的四篇文章，分别对：(1)定点运算时的舍入误差，(2)浮点运算时的舍入误差，(3)有限字长的影响及(4)量化误差作了分析。

第三部分为在通信中应用FFT的一些实例。

除上述三部分外，还有两个附录。附录一中给出了两种计算DFT的新方法，使读者了解在数字信号处理方面，也象其它科学技术领域一样，正在日新月异地继续发展。附录二是用130机汇编语言编写的FFT程序。只要把包含FFT语句的$BASIC$解释程序输入到

注(1)：关于FFT的基本原理，请参看*E.Oran Brigham*：“*Fast Fourier Trausform*”，1974。上海科技出版社已翻译出版。以后简称此书为“书A”。关于计算机语言的书，国内已有多种出版，请读者自行参考。

130机，那么，用极简单程序，就能实现FFT。这为使用者提供了一定的方便。

本书由王文渊、朱雪龙、尹达衡、王祜民、邱盛藩等同志、翻译，由常迥、相行峻、程佩青等同志校阅。附录二由王祜民同志提供。

清华大学无线电系
快速付里叶变换组

目　录

一、专用硬件

摘译自*Lawrence R.Rabiner, Bernard Gold, "Theory and Application of Digital Signal Processing"* 注(2)第十章，*pp* 573—626.

1—1 引言

对于作为数字谱分析技术必要前提的快速付里叶变换(*FFT*)算法，作者已在他处作了相当详细的介绍注(3)，这里仅讨论实现各种*FFT*算法的方法。我们应当了解各种不同的*FFT*结构，以便保证经济地实现，并使所选的结构能与特定的问题相适应。为此，我们将比较完整地论述不同的*FFT*流图。

为了特殊的目的，基2算法可能优于其他算法；但是基4算法已经受到较多的注意，因为它可以比基2节省硬件。我们将比较详细地研究基4系统，并给出基2、基4之间的一般比较。

对于某些极高速系统来说，需要一种流水线*FFT*结构。我们将叙述这样的系统。此外，我们还要讨论*FFT*硬件并行化的其它形式。

1—2 对FFT算法的进一步讨论注(4)

*FFT*可分为“位内算法”(*in-place*)和“非位内算法”(*not-in-place*)。位内算法规定：构成*FFT*的所有*DFT*，全要返回到它们原先占有的寄存器。例如，在图1—1的16点*FFT*中，寄

注(2)：以后简称此书为“书*B*”。

注(3)：指书*B*中第六章。

注(4)：指在注(3)所列的第六章的基础上再作深入讨论。

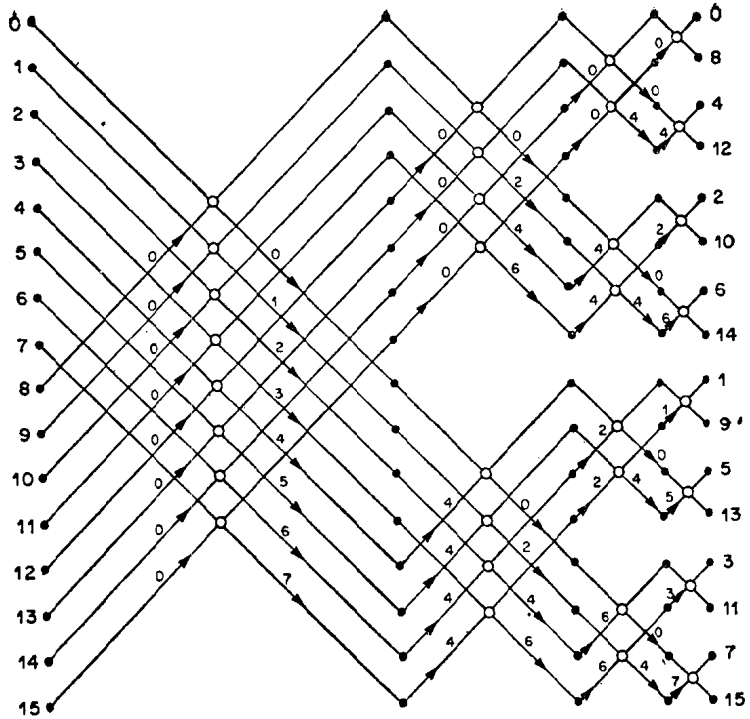

图 1—1　自然次序输入和颠倒位序输出的位内16点FFT

存器 0 和 8 的内容进入到第一级最上面的“开环”（*open circle*）所表示的两点DFT。我们把输入规定为f_0〔$=x(0)$〕和f_1〔$=x(8)$〕，则两点输出是F_0和F_1，F_0代替了f_0，而F_1代替了f_1。

位内算法会发生结果的打乱，对基 2 FFT来说，打乱相应于频率指数的颠倒位序，如图1—1所示。我们将在第三节讨论高基数时的打乱。

此外，FFT结构可以是时间抽取(DIT)和频率抽取(DIF)注(5)。DIT是在两点 DFT 的节点之前有旋转因子，而 DIF 则是在两点 DFT 之后有旋转因子。两组旋转因子均示于图1—1，根据是使用DIT还是DIF而选择相应的一组。箭头旁的数字相应于乘上W^k，

注(5)：关于DIT和DIF，请参看书A及书B。

其中，K就是图示的数字。即使输入指数是颠倒位序的，图1—1的结构也仍然适用，这时，最后的输出将是自然次序。

图1—2给出了另一种排列，它能给出同样有效的算法。*DIT*和*DIF*两种算法也画在同一张图上。

除非另有说明，一般的习惯都是把寄存器从上到下进行编号，而这些号码通常并不标出来。图1—1和图1—2中在输入和输出处的数字，是抽样和变换结果的指数。

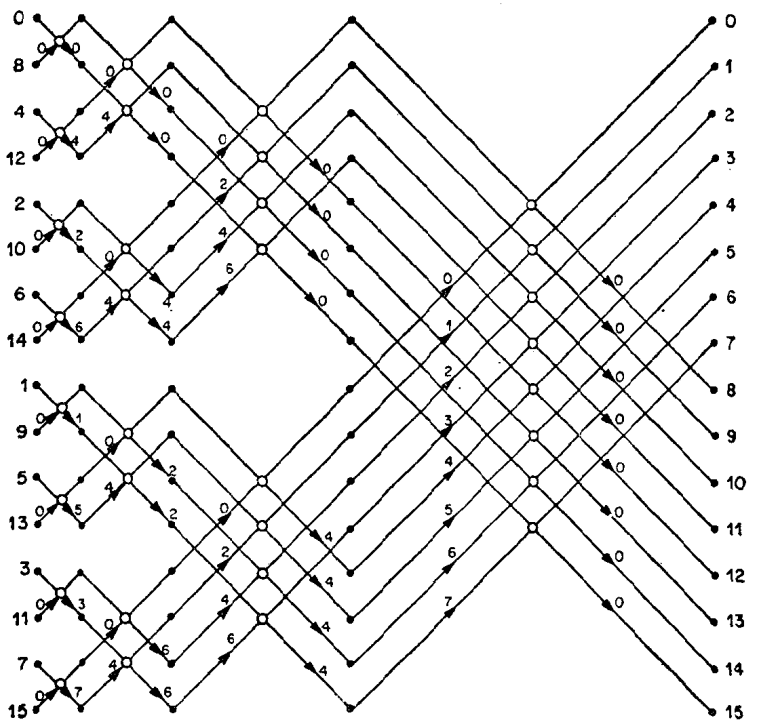

图 1—2　颠倒位序输入和自然次序输出的位内*FFT*（*DIT*系数在节点左面，*DIF*系数在节点右面）

一个 16 点、基 2、具有固定几何结构（*Constant geometry*）的算法示于图1—3。这里，蝶式运算的输出并没有送回到原来的输入处，所以，它是非位内算法。从一个级到下一级时，指数保持不

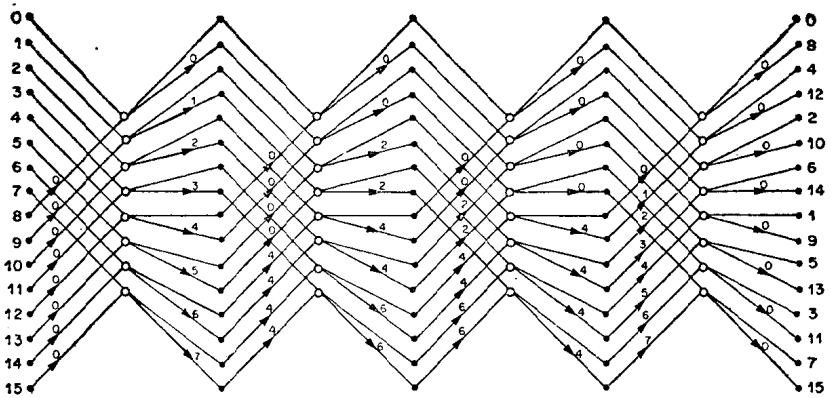

图 1—3　固定几何结构算法：基 2，16点，非位内，自然次序输入，颠倒位序输出（同时注明了DIT和DIF乘子）。

变，这使得一些程序或硬件设计得到简化。输入是自然次序，而输出是颠倒次序的。作为一个一般规则，位内算法需要N个复数寄存器，而非位内算法时则需要$2N$个复数寄存器。图1—4表示一个颠倒位序输入的固定几何结构算法，它给出自然次序的输出。从图1—3中节点两边的箭头，我们就能找到DIT和DIF的旋转因子。图1—5给出了一个避免颠倒位序的算法，其代价是要作一个非位内算法。

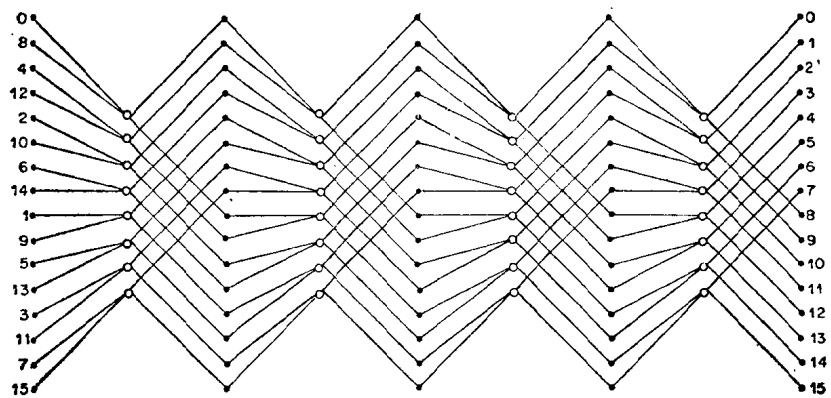

图 1—4　固定几何结构算法：基 2，16点，非位内，颠倒位序输入，自然次序输出

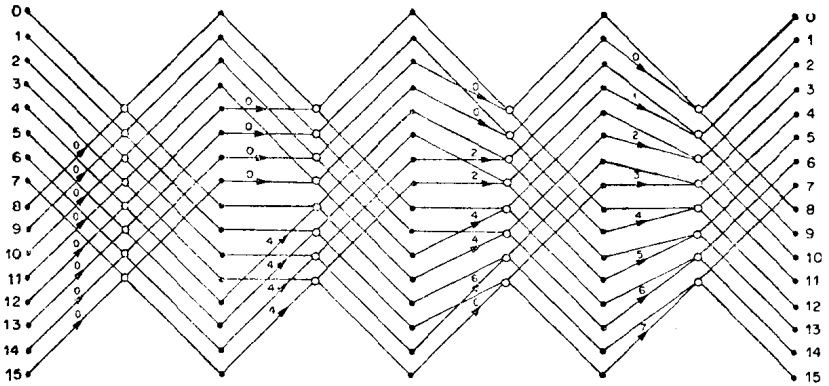

图 1—5　非顺倒位序的输入和输出：DIT；非位内，16点，基 2 FFT

图中仅示出了DIT部分，但是，只要在节点之后配置旋转因子，作成DIF也很容易。

图1—6说明怎样设计一个利用高速暂存存贮器的FFT算法。我们将在后面讨论这种结构的硬件实现。这里，我们仅指出：节点能够配对，并且每两个蝶式运算是对四个输入抽样进行的，因此这些抽样在下四个抽样被处理之前通过了两级FFT。例如，我们假设把

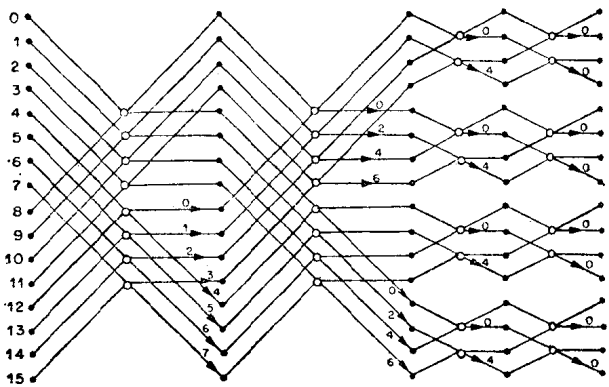

图 1—6　适用于对四个输入样点连续进行两级处理，而在两个蝶式运算期间无需“访问”贮存器的16点、基2FFT硬件。输入为自然次序，而输出被打乱（不是位序颠倒），DIF。

抽样 0 和 8 作为一对送入第 0 级的节点 0，而把抽样 4 和12送入第 0 级的节点 4 。在作了这两个蝶式运算之后，我们用存于相同的四个寄存器、也就是 0 、 4 ， 8，和12里的结果，继续进到第一级的节点 0 和第一级的节点 4 。用同样的方法，我们能送入寄存器 1 ，5 ，9 和13，并也再进两级。这样，只要运算单元不是象一般方法那样，只处理两个抽样，而是能处理 4 个抽样，那么存贮周期就只需要一半。在这种特殊的形式中，因为算法能看成两级一次的位内运算，所以位序要发生颠倒。

图1—7说明怎样能避免位序颠倒。在这个16点算法中，前两级象图1—1那样作，也就是进行简单的位内运算。然后，我们把 4 个抽样送入运算单元，作两次蝶式运算，并交换其结果，如图中最后两级所示。应当注意，我们在这里没有遵守不给寄存器以号码的规则。这样做的目的是为了一直追踪指数。由于我们的寄存器编号出现颠倒位序，所以输出抽样必然是自然次序（在一个完全的位内算法中，毕竟没有涉及寄存器号码，所以结果是颠倒位序的）。

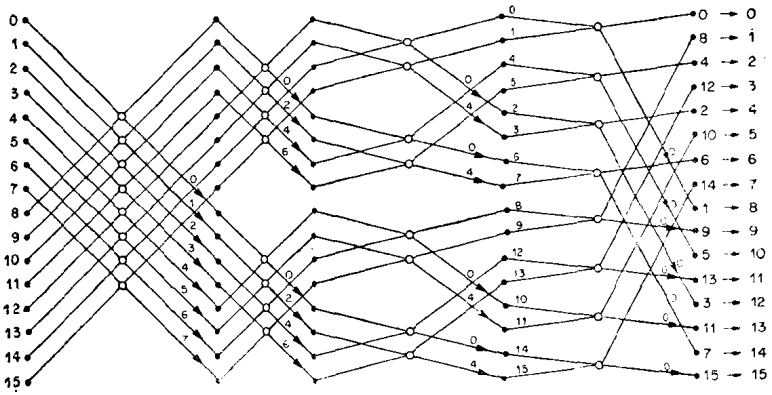

图 1—7　用一次两个蝶式运算算法避免位序颠倒，DIF

图1—8是一次进行两个蝶式运算的另一种算法，然而，它与图1—7的目的不同。在图1—8中，我们把存贮寄存器进行排列，以便一次能读（或写）两个复数字。这样，例如若有抽样 0 和 8 并行地送入蝶式运算，就会节省一个存贮周期。图中右下角的表给出了

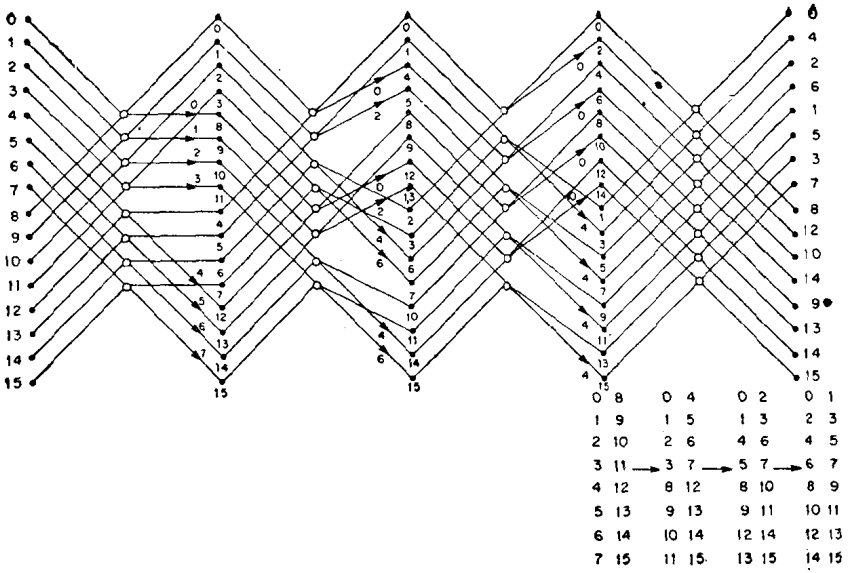

图 1—8　适用于双重并行的一次两个蝶式运算算法

FFT 进行过程中对抽样所希望的配合。为了达到这种配合以便保持并行化，就必须一次实现四个输出点的交换。例如，抽样 0,8和4,12被交换后作为 0，4 和 8，12送入。这里，我们再次不符合习惯地给寄存器以号码。最后结果，即最右一列数值，表明了输出点打乱的情况。它实际上是颠倒位序和一个（8 × 2）阵列的行-列互换的组合。

1—3　FFT变址——固定基时的颠倒位序和颠倒数字位序

我们从一个非常简单的例子着手来说明在位内 *FFT* 中，输出点是怎样被打乱的。也就是说，尽管$x(K)$位于寄存器k中，$X(K)$一般不在寄存器k里。作为例子，我们研究一个四点序列 $x(n)$。$x(n)$

的DFT能写成：

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

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

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

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

下面，我们用四点FFT，按照图 1—9 的流图来完成上述计算。应当注意，$X(1)$和$X(2)$的位置已经颠倒，也就是说，结果已经打乱。我们知道，如果一个FFT表达成连续的“抽取”（用两维表示一维数据）注(6)，那就会产生打乱。这打乱能用一连串套接的行-列互换表示，如图 1—10 中对一个16点FFT所表示的那样。

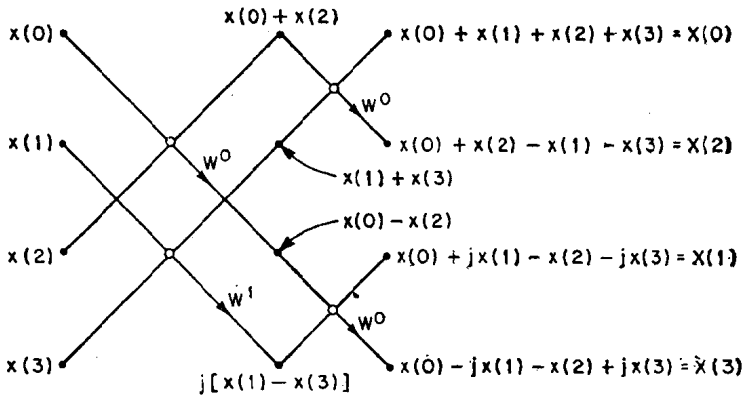

图 1—9　基 2、四点FFT、DIF、位内算法的流图

对于固定基数的FFT来说，输出数据的次序能够非常简洁地表示成输入数据次序的函数。在基 2 系统中，可把结果叫做颠倒位序（*bit reversal*）。更一般地，在基r系统中，则要叫做颠倒数字

注(6)：参看书B中的第六章。

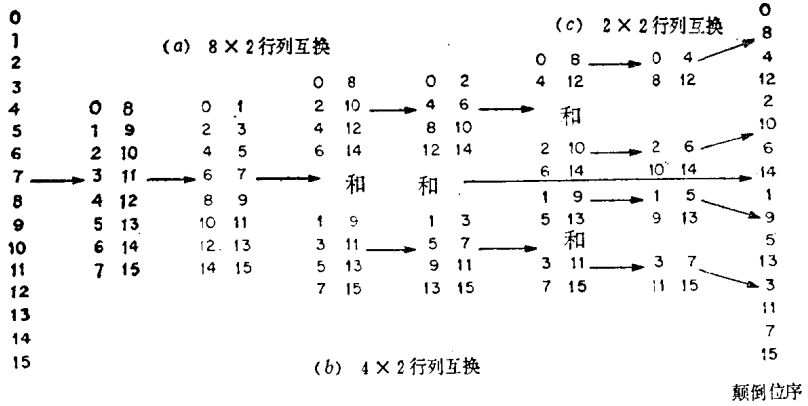

图 1—10　用连续的行-列互换说明16点FFT中的颠倒位序

位序（*digit reversal*），而这数字是基 r 计数系统中的数。这一结果，我们以后再证明。因此，我们若从寄存器n中的$f(n)$着手，而且用二进数$x_5x_4x_3x_2x_1x_0$（设为六位地址，其中，每个x_i是0或1.）表示n，则变换输出就寄存在地址为 $x_0x_1x_2x_3x_4x_5$ 的寄存器中。也就是说，二进数中每位的次序是颠倒的。在基4 FFT 中，如果x_i是基4数，它取的数值只能是0，1，2，3，那就可得到相同的结果；对于基8数而言，x_i将是一个八进数。

为了证明颠倒数字位序，我们利用一个简单的、存在于任意两维阵列的原始次序和同一阵列交换后的次序之间的数值关系。设矩阵：

$$\begin{array}{ccccc} 0 & 1 & 2 & \cdots & m-1 \\ m & m+1 & m+2 & \cdots & 2m-1 \\ 2m & & & & \cdot \\ \vdots & & & & \cdot \\ \vdots & & & & \cdot \\ (p-1)m & \cdot & \cdot & \cdot & N-1 \end{array}$$

有 p 行 和m列，$N=mp$。如果用 p 去乘矩阵中的每个元素，模以 $N-1$，并取其结果，那就得到了前面讨论的行-列互换。这样，变换