第九章、滤波器的算法强度缩减

学过数字信号处理，就不可能不知道滤波器。经典的滤波器分两类：FIR 和 IIR。FIR 是有限长冲击响应滤波器，硬件电路是非递归的；而 IIR 是无限长冲击响应滤波器，硬件电路存在递归环路，值得注意的是，IIR 可以看成是一个 FIR 和递归环路的级联。本章的目的在于如何并行实现 FIR 滤波器，在获得高吞吐率的同时尽可能节约面积和功耗。

回想第八章、快速卷积，前面讨论过 FIR 其实也是一个卷积的过程，只是这个卷积不是短卷积，而是长卷积。根据数字信号处理理论（参见——胡广书《数字信号处理：理论、算法与实现》第二版 3.6.2 节），长卷积可以用短卷积实现，短卷积正是第八章所解决的问题，如此一来 FIR 就能高效实现。OK！没错，第八章的知识已经为 FIR 并行实现提供了一个途径，但是这里要讨论 FIR 的另一种并行结构，当然了，这种新的 FIR 并行结构仍然和快速卷积密不可分，稍后便见分晓。

在进入正题之前，要先擦亮眼睛！记住一点，卷积其实是一种多项式乘法，快速卷积给出了“短”多项式乘法的有效实现，只要是多项式乘法，就能用快速卷积知识搞定。废话那么多，无非想告诉大家，**接下来讨论的 FIR 并行实现其实可看成是“短”多项式乘法。**

本章的任务：

弄清楚以下 4 个问题，

1） FIR 的多相分解。

2） 多相 FIR 的并行结构，以及**如何使用快速卷积构造高效的多相 FIR 并行结构。**

3） 大尺寸并行 FIR 算法。

4） 转置的并行滤波器。

很多数字信号算法，比如 FFT 和现在讨论的并行 FIR，都让我联想到“分而治之”这个道理。分治就是把一个复杂的问题分成两个或更多的相同或相似的子问题，再把子问题分成更小的子问题……直到最后子问题可以简单的直接求解，原问题的解即子问题的解的合并。

FIR 多相分解也是一种“分而治之”处理，其做法就是将“待卷积”的两个序列$x(n)$和$h(n)$按交叉数据分配的原则分为“相同数量的”若干个子序列，比如 2 相分解如下示，

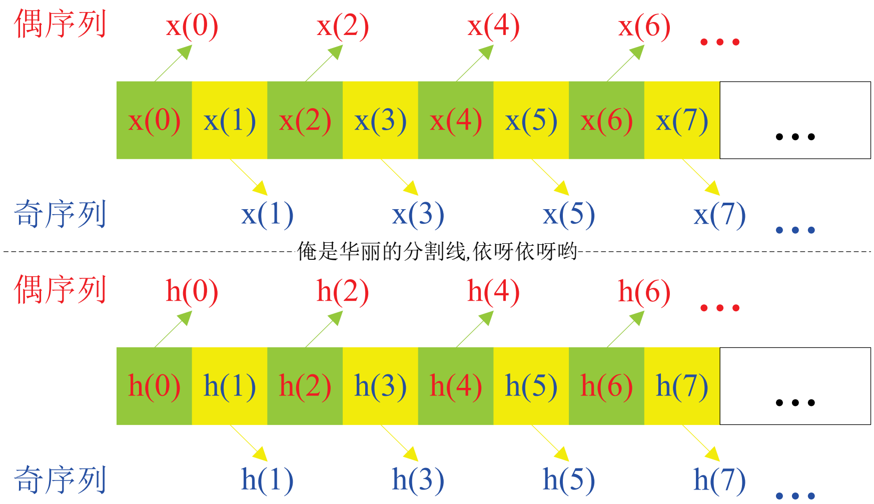

图 1　　2 相分解

在频域，可表示为

$$X(z) = X_0(z^2) + z^{-1}X_1(z^2) \tag{1}$$

其中，

$$X_0(z) = \sum_{n=0}^{+\infty} x(2n)\,z^{-n} \wedge X_1(z) = \sum_{n=0}^{+\infty} x(2n+1)\,z^{-n} \tag{2}$$

同理，有

$$H(z) = H_0(z^2) + z^{-1}H_1(z^2) \tag{3}$$

其中，当N为偶数时，

$$H_0(z) = \sum_{n=0}^{N/2-1} h(2n)\,z^{-n} \wedge H_1(z) = \sum_{n=0}^{N/2-1} h(2n+1)\,z^{-n} \tag{4}$$

若N为奇数，只需修正一下公式(4)的求和上标即可。

如此滤波结果的频域表示为，

$$\begin{aligned}
&Y(z)\\
=\;& X(z)\,H(z)\\
=\;& \left(X_0(z^2) + z^{-1}X_1(z^2)\right)\left(H_0(z^2) + z^{-1}H_1(z^2)\right)\\
=\;& X_0(z^2)\,H_0(z^2) + z^{-1}X_0(z^2)\,H_1(z^2) + z^{-1}X_1(z^2)\,H_0(z^2) + z^{-2}X_1(z^2)\,H_1(z^2)\\
=\;& \left(X_0(z^2)\,H_0(z^2) + z^{-2}X_1(z^2)\,H_1(z^2)\right) + z^{-1}\left(X_0(z^2)\,H_1(z^2) + X_1(z^2)\,H_0(z^2)\right)
\end{aligned}$$

(5)

如果$Y(z)$也做两相分解，则正好有

$$Y(z) = Y_0(z^2) + z^{-1}Y_1(z^2) \tag{6}$$

其中，

$$\begin{aligned} Y_0\left(z^2\right) &= X_0\left(z^2\right)H_0\left(z^2\right)+z^{-2}X_1\left(z^2\right)H_1\left(z^2\right) \\ Y_1\left(z^2\right) &= X_0\left(z^2\right)H_1\left(z^2\right)+X_1\left(z^2\right)H_0\left(z^2\right) \end{aligned} \tag{7}$$

将z^2替换为z，则公式(7)就是

$$\begin{aligned} Y_0\left(z\right) &= X_0\left(z\right)H_0\left(z\right)+z^{-1}X_1\left(z\right)H_1\left(z\right) \\ Y_1\left(z\right) &= X_0\left(z\right)H_1\left(z\right)+X_1\left(z\right)H_0\left(z\right) \end{aligned} \tag{8}$$

也就是说$y\left(n\right)$的奇偶序列分别对应两个子序列序列的和，即

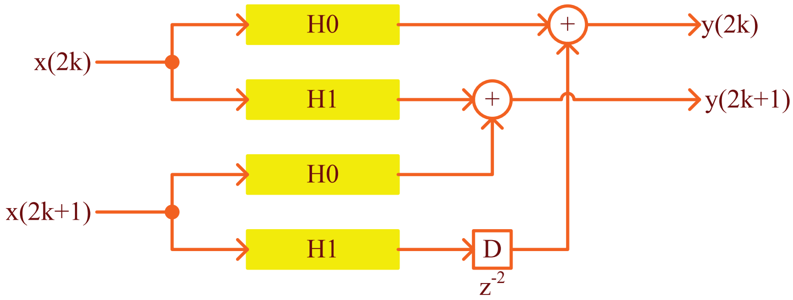

图 2　　2 相分解并行结构，没有经过优化，占用资源多

很多朋友看到这可能会很迷惑，或者说有点乱！！其原因是什么呢？这里给出几点关键的提示，大家开动脑筋仔细的推导一遍，相信会有所收获：

- 为什么序列的奇偶分解，在频域会表示成如公式(1)、公式(3)和公式(6)的形式，其中公式右边的自变量是z^2，而不是“理所当然”的z？在公式右边第二项中出现的z^{-1}又是为何？

以$X\left(z\right)$为例，“深入”推导一下，

$$\begin{aligned} X\left(z\right) = \sum_{n=0}^{+\infty}x\left(n\right)z^{-n} &= \left(\sum_{n=0}^{+\infty}x\left(2n\right)z^{-2n}\right)+\left(\sum_{n=0}^{+\infty}x\left(2n+1\right)z^{-(2n+1)}\right) \\ &= \left(\sum_{n=0}^{+\infty}x\left(2n\right)z^{-2n}\right)+z^{-1}\left(\sum_{n=0}^{+\infty}x\left(2n+1\right)z^{-2n}\right) \end{aligned} \tag{9}$$

记$x\left(n\right)$的奇偶序列分别为$x_0\left(n\right)$和$x_1\left(n\right)$，则公式(9)可表示如下

$$\begin{aligned} X\left(z\right) &= \left(\sum_{n=0}^{+\infty}x_0\left(n\right)z^{-2n}\right)+z^{-1}\left(\sum_{n=0}^{+\infty}x_1\left(n\right)z^{-2n}\right) \\ &= X_0\left(z^2\right)+z^{-1}X_1\left(z^2\right) \end{aligned} \tag{10}$$

因为

$$X_0\left(z^2\right)=\sum_{n=0}^{+\infty}x_0\left(n\right)z^{-2n} \quad , \quad X_1\left(z^2\right)=\sum_{n=0}^{+\infty}x_1\left(n\right)z^{-2n} \tag{11}$$

显然，有

$$X_0\left(z\right)=\sum_{n=0}^{+\infty}x_0\left(n\right)z^{-n} \quad , \quad X_1\left(z\right)=\sum_{n=0}^{+\infty}x_1\left(n\right)z^{-n} \tag{12}$$

，，，注意，$x\left(n\right)$的奇偶序列的 z 变换分别是$X_0\left(z\right)$和$X_1\left(z\right)$，而不是$X_0\left(z^2\right)$和$X_1\left(z^2\right)$。我们不能简单的说，$X\left(z\right)$是$x\left(n\right)$奇偶序列的 z 变换之和，关系没那么简单，详细思考公式(10)，看看到底意味着什么就清楚了。

- 为什么会有公式(7)，也就是说，怎么知道$z^{-2}X_1\left(z^2\right)H_1\left(z^2\right)$属于$Y_0\left(z^2\right)$，而不是

$Y_1\left(z^2\right)$?

如果仔细观察/“深入去想象”$Y_0\left(z^2\right)$和$Y_1\left(z^2\right)$每一项的z因子的幂次到底是什么就能明白。

其实，$Y_0\left(z^2\right)$每一项z因子的幂次必为偶数，而$Y_1\left(z^2\right)$每一项z因子的幂次必为奇数。

$z^{-2}X_1\left(z^2\right)H_1\left(z^2\right)$的每一项$z$因子的幂次是偶数，所以属于$Y_0\left(z^2\right)$。大家可以摸清其中的规律，这样在更多相分解（3 相或 4 相等等）时，就不会搞错。

- 公式(5)的$z^{-2}X_1\left(z^2\right)H_1\left(z^2\right)$中的$z^{-2}$意味着 2 个延时，可是为什么图 2 中只有一个 D？课本上说到“2 并行电路的z^{-2}属于快延时，在实际电路中就是一个延时”，此话怎讲？还有公式(5)的$z^{-1}\left(X_0\left(z^2\right)H_1\left(z^2\right)+X_1\left(z^2\right)H_0\left(z^2\right)\right)$中的$z^{-1}$又表示什么呢，难道是 1/2 个延时吗？

实际上，图 2 是根据公式(8)画出的，而非公式(5)。注意公式(5)(6)(7)(8)，公式(8)也是从公式(5)推导而来。请思考公式(5)和公式(8)的关系，见图 3，顶端子图相当于公式(8)所表示的奇偶序列，往下即为公式(7)，再往下的奇序列右移一位就是公式(6)中的z^{-1}，最后是序列相加，结果正好是完整序列$y\left(n\right)$。

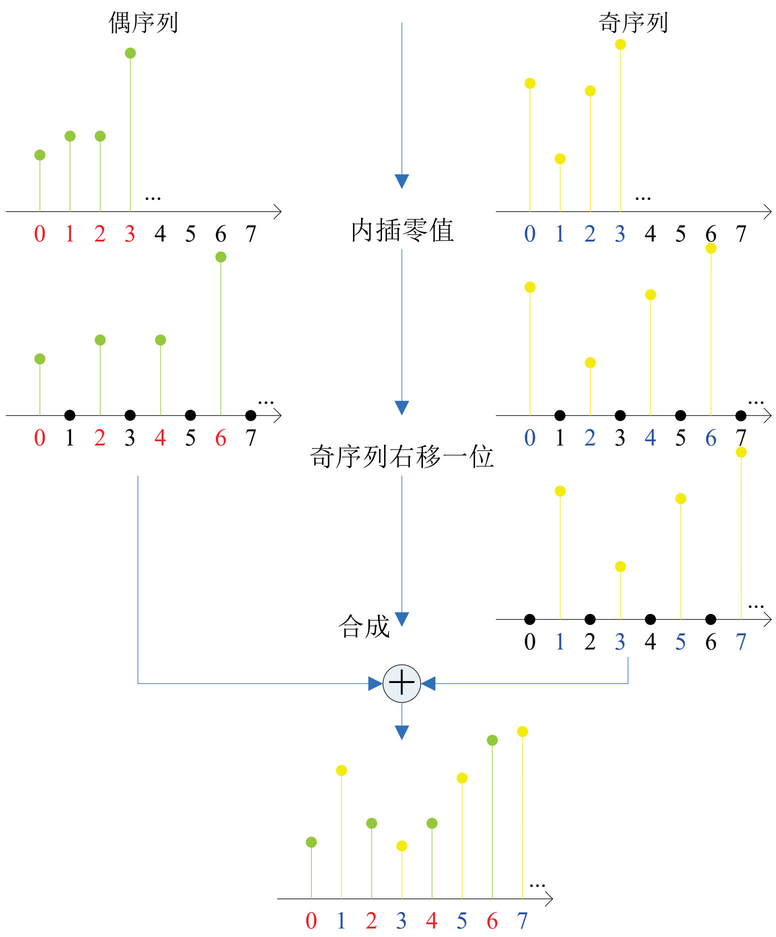

图 3　　序列的合成

题外话：序列的多相分解初看感觉很容易，但细想又没那么简单。能深入理解每个公式的物理意义是最好不过的，如果不能，至少应该记住多相分解的公式!!

推而广之，FIR 的 3 相分解如下，

$$Y(z)=\sum_{i=0}^{2} Y_i\left(z^3\right) z^{-i}=X(z)\,H(z)=\left(\sum_{i=0}^{2} X_i\left(z^3\right) z^{-i}\right)\left(\sum_{i=0}^{2} H_i\left(z^3\right) z^{-i}\right) \tag{13}$$

简化后，有

$$\begin{aligned}
&\sum_{i=0}^{2} Y_i z^{-i}\\
=\;& \left(\sum_{i=0}^{2} X_i z^{-i}\right)\left(\sum_{i=0}^{2} H_i z^{-i}\right)\\
=\;& X_0H_0+\left(X_0H_1+X_1H_0\right) z^{-1}+\left(X_2H_0+X_1H_1+X_0H_2\right) z^{-2}\\
& +\left(X_1H_2+X_2H_1\right) z^{-3}+X_2H_2z^{-4}\\
=\;& \left(X_0H_0+\left(X_1H_2+X_2H_1\right) z^{-3}\right)+\left(\left(X_0H_1+X_1H_0\right)+X_2H_2z^{-3}\right) z^{-1}\\
& +\left(X_2H_0+X_1H_1+X_0H_2\right) z^{-2}
\end{aligned} \tag{14}$$

三路输出，分别为

$$\begin{array}{rcl} Y_0 & = & X_0H_0 + (X_1H_2 + X_2H_1)\, z^{-3} \\ Y_1 & = & (X_0H_1 + X_1H_0) + X_2H_2z^{-3} \\ Y_2 & = & X_2H_0 + X_1H_1 + X_0H_2 \end{array} \tag{15}$$

写成矩阵形式，有

$$\begin{pmatrix} Y_0 \\ Y_1 \\ Y_2 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & z^{-3} & 0 \\ 0 & 1 & 0 & 0 & z^{-3} \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} X_0H_0 \\ X_0H_1 + X_1H_0 \\ X_2H_0 + X_1H_1 + X_0H_2 \\ X_1H_2 + X_2H_1 \\ X_2H_2 \end{pmatrix} \tag{16}$$

公式(15)和(16)对应于 2 相分解中的公式(6)和(7)，注意，公式(15)和(16)的自变量是z^3，只需进行变量代换，将z^3替换为z，就能得到 FIR 3 并行电路的构造式子，如下

$$\begin{array}{rcl} Y_0\,(z) & = & X_0\,(z)\, H_0\,(z) + (X_1\,(z)\, H_2\,(z) + X_2\,(z)\, H_1\,(z))\, z^{-1} \\ Y_1\,(z) & = & (X_0\,(z)\, H_1\,(z) + X_1\,(z)\, H_0\,(z)) + X_2\,(z)\, H_2\,(z)\, z^{-1} \\ Y_2\,(z) & = & X_2\,(z)\, H_0\,(z) + X_1\,(z)\, H_1\,(z) + X_0\,(z)\, H_2\,(z) \end{array} \tag{17}$$

根据公式(17)很容易画出最终电路图，见课本图 9-2。

值得注意的 3 点：

1. 公式(16)右边的向量是多项式相乘$\left(\sum_{i=0}^{2} X_i z^{-i}\right)\left(\sum_{i=0}^{2} H_i z^{-i}\right)$的结果的各项系数，见公式(14)的第二步。
2. 对这些系数进行“合并同类项”，就能得到输出Y_i，也就是公式(14)的最后一步，写成矩阵形式如公式(16)。
3. 多项式乘法可以用快速卷积搞定。

如此一来，就把高效 FIR 多相分解并行实现转化为快速卷积问题来解决。下面代码用于构造 N 相分解的“合并同类项”矩阵，对应于公式(16)，其中$p = z^{-1}$，

代码 1 求 FIR 多相分解的“前前”置矩阵，用于从快速卷积推导多相分解高效实现

```
> restart :
> N := 3;
                                N := 3
> Y := collect(sum(H[i]·p^i, i = 0..N − 1)·sum(X[i]·p^i, i = 0..N − 1), p);
```

$$Y := H_2\, X_2\, p^4 + (H_1\, X_2 + H_2\, X_1)\, p^3 + (H_0\, X_2 + H_1\, X_1 + H_2\, X_0)\, p^2 + (H_0\, X_1 + H_1\, X_0)\, p + H_0\, X_0$$

```
> YY := array(0..N-1, [seq(0, i=0..N-1)]) :
  for i from 0 to 2·(N-1) do
    j := i mod N;
    YY[j] := YY[j] + y[i]·p^(iquo(i, N)·N);
  od:
  for i from 0 to N-1 do
    print(YY[i]);
  od:
```

$$y_0 + y_3\ p^3$$

$$y_1 + y_4\ p^3$$

$$y_2$$

```
> A := Matrix(N, 2·N-1) :
  for i from 1 to N do
    for j from 1 to 2·N-1 do
      A[i, j] := coeff(YY[i-1], y[j-1]);
    od:
  od:
  print(A);
```

$$\begin{bmatrix} 1 & 0 & 0 & p^3 & 0 \\ 0 & 1 & 0 & 0 & p^3 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

为了验证用快速卷积来解决 FIR 多相分解问题的正确性，使用 8.6 节 3x3 快速卷积的结果来得出 FIR 3 相分解的计算矩阵，如下

代码 2　由 3x3 快速卷积导出 3 相分解的实现，并进行验证

```
> simplify(evalm(
```

$$simplify\left(evalm\left(\begin{bmatrix} 1 & 0 & 0 & p^3 & 0 \\ 0 & 1 & 0 & 0 & p^3 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \&* \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & 0 & 1 & 0 & 0 \\ -1 & 1 & -1 & 0 & 1 & 0 \\ 0 & -1 & -1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \&* \begin{bmatrix} H_0 & 0 & 0 & 0 & 0 & 0 \\ 0 & H_1 & 0 & 0 & 0 & 0 \\ 0 & 0 & H_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & H_0+H_1 & 0 & 0 \\ 0 & 0 & 0 & 0 & H_0+H_2 & 0 \\ 0 & 0 & 0 & 0 & 0 & H_1+H_2 \end{bmatrix} \&* \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}\right)\right);$$

$$\begin{bmatrix} H_0 & p^3 H_2 & p^3 H_1 \\ H_1 & H_0 & p^3 H_2 \\ H_2 & H_1 & H_0 \end{bmatrix}$$

计算结果为上述代码的蓝色矩阵所示，正好就是 FIR 3 相分解的原始矩阵，见课本公式(9-17)。其实这不难理解，FIR 多相分解本质就是多项式相乘，从公式(14)可以明确看出来。

传统的 3 相分解一共需要 9 个子滤波器，而上述基于快速卷积的解决方案只需 6 个子滤波器。下面是使用改进 winograd 快速卷积算法设计的另一个实现，其中只需 5 个子滤波器，

代码 3　另外一个 3 相分解的例子，只使用 5 个子滤波器，更节省资源（？？）

$$> \; simplify\left(evalm\left(\begin{bmatrix} 1 & 0 & 0 & p^3 & 0 \\ 0 & 1 & 0 & 0 & p^3 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \&* \begin{bmatrix} 2 & 0 & 0 & 0 & 0 \\ 1 & 2 & -2 & 1 & -2 \\ -2 & 3 & 1 & 0 & -1 \\ -1 & 1 & 1 & -1 & 2 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \&* \begin{bmatrix} \frac{H_0}{2} & 0 & 0 & 0 & 0 \\ 0 & \frac{H_0+H_1+H_2}{6} & 0 & 0 & 0 \\ 0 & 0 & \frac{H_0-H_1+H_2}{2} & 0 & 0 \\ 0 & 0 & 0 & \frac{H_0-2\cdot H_1+4\cdot H_2}{6} & 0 \\ 0 & 0 & 0 & 0 & H_2 \end{bmatrix} \&* \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & -2 & 4 \\ 0 & 0 & 1 \end{bmatrix}\right)\right);$$

$$\begin{bmatrix} H_0 & p^3\,H_2 & p^3\,H_1 \\ H_1 & H_0 & p^3\,H_2 \\ H_2 & H_1 & H_0 \end{bmatrix}$$

大家可以根据这个思路，动手试试!!

对于小规模的多相分解实现，只使用快速卷积就能满意解决！但是对于大规模多相分解，比如 16 相分解，就需要设计 16x16 的快速卷积高效算法，这个做起来有点困难。应该怎么办呢？“车到山前必有路，船到桥头自然直” 幸运的是，我们仍然可以使用“分而

治之”的方法来搞定，基本思路就是将 16 相分解划分为 4 个 4 相分解问题，然后再将 4 相分解划分为 2 个 2 相分解问题，直接使用快速卷积算法搞定 2 相分解，然后在合并为 4 相分解，最后再合并为 16 相分解。当然了，也可以直接用快速卷积解决 4 相分解，而不一定非要划分到 2 相。

课本上偏好于使用 2 相分解，所以下面先来讨论 2 相分解的几种典型解法：

1. 2 相分解相对简单，可以通过观察来构造快速算法，如课本上的 9.2.2.1 节的结果

$$\begin{pmatrix} Y_0 \\ Y_1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & z^{-2} \\ -1 & 1 & -1 \end{pmatrix} \begin{pmatrix} H_0 & 0 & 0 \\ 0 & H_0+H_1 & 0 \\ 0 & 0 & H_1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} X_0 \\ X_1 \end{pmatrix} \quad (18)$$

2. 直接利用第八章，例 8.2.1 和例 8.2.3 的 2x2 Cook-Toom/改进 Cook-Toom 快速卷积结果，有

代码 4　直接由快速卷积导出的 FIR 2 相分解实现

$$> simplify\left(MTM[transpose]\left(Matrix\left(evalm\left(\begin{bmatrix} 1 & 0 & p^2 \\ 0 & 1 & 0 \end{bmatrix} \&* \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -1 \\ -1 & 1 & 1 \end{bmatrix} \&* \begin{bmatrix} H_0 & 0 & 0 \\ 0 & \frac{H_0+H_1}{2} & 0 \\ 0 & 0 & \frac{H_0-H_1}{2} \end{bmatrix} \&* \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & -1 \end{bmatrix} \&* \begin{bmatrix} X_0 \\ X_1 \end{bmatrix}\right)\right)\right)\right);$$

$$\begin{bmatrix} H_0\ X_0 + p^2\ H_1\ X_1 \\ H_1\ X_0 + H_0\ X_1 \end{bmatrix}$$

$$> \mathit{simplify}\left(\mathit{MTM}[\mathit{transpose}]\left(\mathit{Matrix}\left(\mathit{evalm}\left(\begin{bmatrix} 1 & 0 & p^2 \\ 0 & 1 & 0 \end{bmatrix} \&* \begin{bmatrix} 1 & 0 & 0 \\ 1 & -1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \&* \begin{bmatrix} H_0 & 0 & 0 \\ 0 & H_0-H_1 & 0 \\ 0 & 0 & H_1 \end{bmatrix} \&* \begin{bmatrix} 1 & 0 \\ 1 & -1 \\ 0 & 1 \end{bmatrix} \&* \begin{bmatrix} X_0 \\ X_1 \end{bmatrix}\right)\right)\right)\right);$$

$$\begin{bmatrix} H_0\ X_0 + p^2\ H_1\ X_1 \\ H_1\ X_0 + H_0\ X_1 \end{bmatrix}$$

根据上述 3 个 2 相分解的解乘开，但需注意：保持子滤波器的结构，即

$$\begin{aligned} Y_0 &= H_0X_0 + z^{-2}H_1X_1 \\ Y_1 &= (H_0+H_1)(X_0+X_1) - H_0X_0 - H_1X_1 \end{aligned} \tag{19}$$

$$\begin{aligned} Y_0 &= \left(1-z^{-2}\right)H_0X_0 + z^{-2}\frac{H_0+H_1}{2}(X_0+X_1) + z^{-2}\frac{H_0-H_1}{2}(X_0-X_1) \\ Y_1 &= \frac{H_0+H_1}{2}(X_0+X_1) - \frac{H_0-H_1}{2}(X_0-X_1) \end{aligned} \tag{20}$$

$$\begin{aligned} Y_0 &= H_0X_0 + z^{-2}H_1X_1 \\ Y_1 &= H_0X_0 - (H_0-H_1)(X_0-X_1) + H_1X_1 \end{aligned} \tag{21}$$

这里我们将公式(19)和(20)对应的结构图画出，公式(21)的结构图留给大家练习，

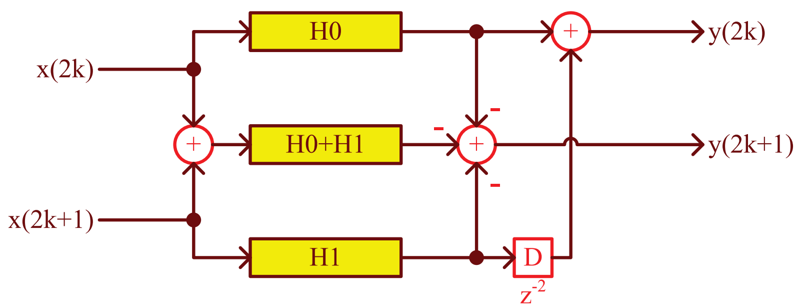

图 4　　公式(19)的 2 相并行 FIR 结构

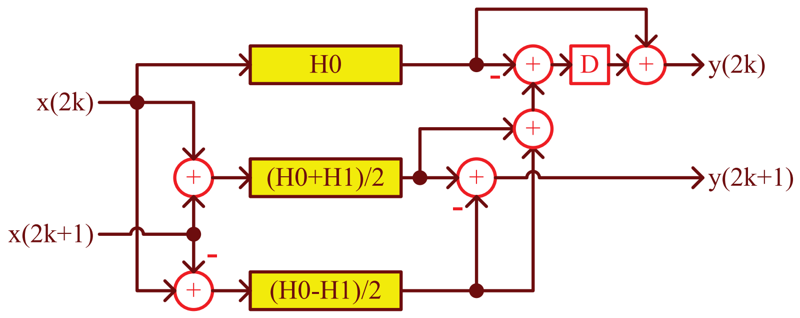

图 5　　公式(20)的 2 相并行 FIR 结构

相比起来，公式(19)的结构要简单一些。有了 2 相分解的公式(19)，就能用分治的方法来处理更大规模的多相分解问题，详细内容请阅读课本的 9.2.2.1 节的后半段。分解的过程本质是简单的，但手工操作却非常烦人，，幸运的是，用 Maple 可以轻松搞定，，其实就是编写一个递归函数，一层层的进行 2 相分解，代码如下，

代码 5　根据公式(19)编制的大规模多相分解问题求解方法

```
> restart :
> Hset := { };
                    Hset := { }
>

FFA2 :=proc(S)
  local HX, H, X, T, K, N, R;
  global Hset;
  HX := op(S);
  K := degree(HX[1], p);
  if K=0 then
    R := B[sort(HX[1])];
    Hset := Hset union {R};
    RETURN(R);
  else
    N := floor((K-1)/2);
    H[0] := sum(coeff(HX[1], p, i)*p^i, i=0..N);
    H[1] := sum(coeff(HX[1], p, i)*p^(i-N-1), i=N+1..K);
    X[0] := sum(coeff(HX[2], p, i)*p^i, i=0..N);
    X[1] := sum(coeff(HX[2], p, i)*p^(i-N-1), i=N+1..K);
    T[0] := FFA2(H[0]*X[0]);
    T[1] := FFA2(H[1]*X[1]);
    T[2] := FFA2((H[0]+H[1])*(X[0]+X[1]));
    RETURN(T[0]+T[1]*p^(2*(N+1))+p^(N+1)*(T[2]-T[0]-T[1]));
  fi;
end:
```

```
> 
  printY := proc(S, N)
    local i, j, Y;
    Y := Array(0..N-1, [seq(0, i=0..N-1)]) :
    for i from 0 to 2·(N-1) do
      j := i mod N;
      Y[j] := Y[j] + coeff(S, p, i)·p^(iquo(i, N)·N);
    od;
    RETURN(Y);
   end:

> 
  printQ := proc(Y, Hset, N)
    local i, j, K, Q;
    K := nops(Hset);
    Q := Matrix(N, K);
    for i from 0 to N-1 do
      for j from 1 to K do
        Q[i+1, j] := coeff(Y[i], Hset[j]);
      od;
    od;
    RETURN(Q);
   end:

> 
> N := 4;
```

$$N := 4$$

```
> 

> S := FFA2(sum(H[i]·p^i, i=0..N-1)·sum(X[i]·p^i, i=0..N
    -1)) :

> Y := printY(S, N) :
> 
> Q := printQ(Y, Hset, N);
```

$$Q := \begin{bmatrix} 1 & -p^4 & p^4 & -p^4 & 0 & 0 & p^4 & 0 & 0 \\ -1 & -1 & -p^4 & -p^4 & 1 & 0 & 0 & p^4 & 0 \\ -1 & 1 & -1 & p^4 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & 1 \end{bmatrix}$$

```
> nops(Hset); print(Hset);
```

$$9$$

$$\left\{B_{H_0}, B_{H_1}, B_{H_2}, B_{H_3}, B_{H_0+H_1}, B_{H_0+H_2}, B_{H_1+H_3}, B_{H_2+H_3}, B_{H_0+H_1+H_2+H_3}\right\}$$

```
> simplify(evalm(Q
```

$$
\&*\begin{bmatrix}
H_0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & H_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & H_2 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & H_3 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & H_0 + H_1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & H_0 + H_2 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & H_1 + H_3 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & H_2 + H_3 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & H_0 + H_1 + H_2 + H_3
\end{bmatrix}
$$

$$
\&*\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
1 & 1 & 0 & 0 \\
1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 \\
0 & 0 & 1 & 1 \\
1 & 1 & 1 & 1
\end{bmatrix}\Bigg)\Bigg);
$$

$$\begin{bmatrix} H_0 & p^4 H_3 & p^4 H_2 & p^4 H_1 \\ H_1 & H_0 & p^4 H_3 & p^4 H_2 \\ H_2 & H_1 & H_0 & p^4 H_3 \\ H_3 & H_2 & H_1 & H_0 \end{bmatrix}$$

代码中进行了 4 相分解求解（N:=4;），验证结果正确。

这里，再给出基于公式(20)的递归 2 相分解程序如下，

代码 6　根据公式(20)编制的大规模多相分解问题求解方法

```
> restart :
> Hset := { };
                    Hset := { }
>
FFA2 := proc(S)
  local HX, H, X, T, K, N, R;
  global Hset;
  HX := op(S);
  K := degree(HX[1], p);
  if K = 0 then
    R := B[sort(HX[1])];
    Hset := Hset union {R};
    RETURN(R);
  else
    N := floor((K - 1)/2);
    H[0] := sum(coeff(HX[1], p, i)·p^i, i = 0..N);
    H[1] := sum(coeff(HX[1], p, i)·p^(i - N - 1), i = N + 1..K);
    X[0] := sum(coeff(HX[2], p, i)·p^i, i = 0..N);
    X[1] := sum(coeff(HX[2], p, i)·p^(i - N - 1), i = N + 1..K);
    T[0] := FFA2(H[0]·X[0]);
    T[1] := FFA2((H[0] + H[1])/2·(X[0] + X[1]));
    T[2] := FFA2((H[0] - H[1])/2·(X[0] - X[1]));
    RETURN((1 - p^(2·(N + 1)))·T[0] + p^(2·(N + 1))·(T[1] + T[2]) + p^(N + 1)·(T[1]
    - T[2]));
  fi;
 end:
>
```

```
>
printY :=proc(S, N)
  local i, j, Y;
  Y := Array(0..N-1, [seq(0, i=0..N-1)]) :
  for i from 0 to 2·(N-1) do
    j := i mod N;
    Y[j] := Y[j] + coeff(S, p, i)·p^(iquo(i, N)·N);
  od;
  RETURN(Y);
 end:

>
>
printQ :=proc(Y, Hset, N)
  local i, j, K, Q;
  K := nops(Hset);
  Q := Matrix(N, K);
  for i from 0 to N-1 do
    for j from 1 to K do
      Q[i+1, j] := coeff(Y[i], Hset[j]);
    od;
  od;
  RETURN(Q);
 end:

>
>
> N := 4;
```

$$N := 4$$

```
>
> S := FFA2(sum(H[i]·p^i, i=0..N-1)·sum(X[i]·p^i, i=0..N
     -1)) :

> Y := printY(S, N) :
>
>
> Q := printQ(Y, Hset, N);
```

$$Q := \begin{bmatrix} 1-p^4 & 0 & 0 & 2\ p^4 & 0 & -p^4 & -p^4 & p^4 & p^4 \\ 0 & -1+p^4 & 1-p^4 & 0 & 0 & -p^4 & p^4 & -p^4 & p^4 \\ -1+p^4 & 1-p^4 & 1-p^4 & -1-p^4 & 1-p^4 & p^4 & p^4 & p^4 & p^4 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & -1 & 1 \end{bmatrix}$$

```
> nops(Hset); print(Hset);
```

$$9$$

$$\left\{B_{H_0}, B_{\frac{1}{2}H_0-\frac{1}{2}H_1}, B_{\frac{1}{2}H_0+\frac{1}{2}H_1}, B_{\frac{1}{2}H_0-\frac{1}{2}H_2}, B_{\frac{1}{2}H_0+\frac{1}{2}H_2}, B_{\frac{1}{4}H_0-\frac{1}{4}H_1-\frac{1}{4}H_2+\frac{1}{4}H_3}, B_{\frac{1}{4}H_0+\frac{1}{4}H_1-\frac{1}{4}H_2-\frac{1}{4}H_3}, B_{\frac{1}{4}H_0-\frac{1}{4}H_1+\frac{1}{4}H_2-\frac{1}{4}H_3}, B_{\frac{1}{4}H_0+\frac{1}{4}H_1+\frac{1}{4}H_2+\frac{1}{4}H_3}\right\}$$

>

>

$$\mathit{simplify}\left(\mathit{evalm}\left(Q\&*\mathit{MTM}[\mathit{diag}](\mathit{map}(\mathit{op},[\mathit{op}(\mathit{Hset})]))\&*\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & -1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & -1 & -1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & 1 & 1 \end{bmatrix}\right)\right);$$

$$\begin{bmatrix} H_0 & p^4 H_3 & p^4 H_2 & p^4 H_1 \\ H_1 & H_0 & p^4 H_3 & p^4 H_2 \\ H_2 & H_1 & H_0 & p^4 H_3 \\ H_3 & H_2 & H_1 & H_0 \end{bmatrix}$$

，注意，在程序中 Q 矩阵就是课本上所谓的前置矩阵，而对角矩阵 H 可根据 Hset 的每个元素下标写出，，其实 Hset 的元素的下标就是对角矩阵的对角线元素，后置矩阵 P 与 2 相分解公式和 Hset 元素相关，比如公式(19)，H_0就有X_0，有H_1就有X_1，有(H_0+H_1)就有(X_0+X_1)，请思考一下，对角矩阵和后置矩阵相乘，结果应该是什么形式，，不就是为了得到H_0X_0、H_1X_1和$(H_0+H_1)(X_0+X_1)$项吗！！所以只需根据 Hset 即可写出后置矩阵。对于公式(20)的情况也类似，大家可以结合代码的中例子的结果思考。。

一般的，FIR 多相分解结果可写为矩阵形式

$$Y = Q \cdot H \cdot P \cdot X \tag{22}$$

其中Q为前置矩阵，H为子滤波器的对角阵，P为后置矩阵，将这三个矩阵乘开，记为H，注意此H非公式(22)中的彼H，，则

$$Y = \mathrm{H} \cdot X \tag{23}$$

对公式(23)所表示的结构进行转置，包含两个步骤：

1. 对X进行上下反转，也就是 flipud(X)，要维持公式(23)的正确性，则必须对 H 进行左右反转，即 fliplr(H)。
2. 对Y进行上下反转，也就是 flipud(Y)，要维持公式(23)的正确性，则必须对 H 进行上下反转，即 flipud(H)。

所以，如果同时反转X和Y，那么必须进行 flipud(fliplr(H))操作，当 H 为方阵时 flipud(fliplr(H))等价于矩阵的转置，将该过程表示为公式如下

$$Y_F = \mathrm{H}^T \cdot X_F \tag{24}$$

系统的转置结构有时会改善其舍入噪声。此外系统的转置也可以对信号流图进行操作而得，此处不再详述。

根据公式(24)和公式(22)，有

$$\begin{aligned} Y_F \;=\; \mathrm{H}^T \cdot X_F \;&=\; (Q \cdot H \cdot P)^T \cdot X_F \\ &=\; P^T \cdot H^T \cdot Q^T \cdot X_F \end{aligned} \tag{25}$$

以课本上的 2 相分解为例，原滤波器结构为

$$\begin{pmatrix} Y_0 \\ Y_1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & z^{-2} \\ -1 & 1 & -1 \end{pmatrix} \begin{pmatrix} H_0 & 0 & 0 \\ 0 & H_0 + H_1 & 0 \\ 0 & 0 & H_1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} X_0 \\ X_1 \end{pmatrix} \tag{26}$$

其转置结构为

$$\begin{pmatrix} Y_1 \\ Y_0 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} H_0 & 0 & 0 \\ 0 & H_0 + H_1 & 0 \\ 0 & 0 & H_1 \end{pmatrix} \begin{pmatrix} 1 & -1 \\ 0 & 1 \\ z^{-2} & -1 \end{pmatrix} \begin{pmatrix} X_1 \\ X_0 \end{pmatrix} \tag{27}$$

最终电路结构如下图

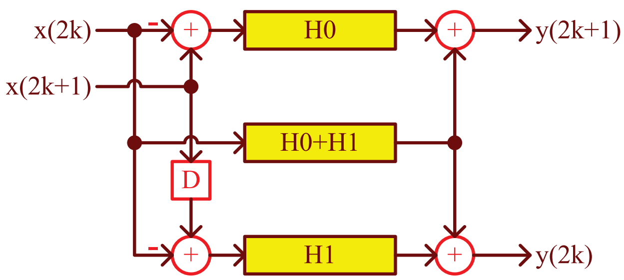

图 6　　FIR 2 相分解的转置结构

附注：滤波器的多相分解高效实现包含两个问题，

1. 由快速卷积推导出小规模多相实现矩阵式，详细的例子请查看公式(13)-(17)。
2. 如何利用分治的方法解决大规模多相分解问题，这个请先阅读课本上的 9.2.2.1 节后半段，将其编制为 Maple 程序如代码 5-6 所示。

3. 思考：处理 9x9 的多相分解，虽然可以用代码 5 或 6 搞定，但也可以用 3 个 3 相分解搞定；如果使用 3 个 3 相分解，首先要求出 3 相分解的高效实现，比如代码 3 所示结构，如何将代码 5 改写成 3 相分解处理的程序呢？

关于第九章还有 DCT 变换和佚阶滤波器的算法强度缩减，在此就先跳过。大家可以自学，请记住以下几点：

- 一个问题是否能进行分治处理？
- 是否存在可以共享的子结构？如果存在，共享这些结构将会带来资源的节省。

佚阶滤波器的功能就是大小排序，《计算机算法导论》中给出了多种排序算法，，本书要实现的算法看起来属于归并排序，大家可以先看看相关的算法介绍，再看课本！