数字信号处理

邓振淼

中山大学电子与通信工程学院

2019-8-28

快速傅里叶变换(FFT)

- 引言
- 基2 FFT算法
- 进一步减小运算量的措施
- 其他快速算法简介

引言

- 1965年库利(J. W. Cooley)和图基(J .W. Tukey)发表了著名的“An algorithm for the machine calculatI/On of complex Fourier series. Math. Comp., Vol. 19 (April 1965), pp. 297–301” →FFT
- 1984年，法国的P. Dohamel和H. Hollmann提出分裂基快速算法，运算效率进一步提升。

快速傅里叶变换(FFT)

- 引言
- 基2 FFT算法
- 进一步减小运算量的措施
- 其他快速算法简介

降低运算量的途径

- 把N点DFT分解为几个较短的DFT，可使乘法次数大大减少，另外，利用旋转因子W_N^{kn}的周期性、对称性和可约性来减少DFT的运算次数。

$$W_N^{m+lN}=e^{-\frac{2\pi}{N}(m+lN)}=e^{-j\frac{2\pi}{N}m}=W_N^m \qquad \rightarrow \text{周期性}$$

$$W_N^{-m}=W_N^{N-m}\text{或}\left[W_N^{N-m}\right]^*=W_N^m,\ W_N^{m+\frac{N}{2}}=-W_N^m \qquad \rightarrow \text{对称性}$$

$W_N^m=W_{N/n}^{m/n}$，N/n和m/n为整数。 →可约性

- 利用这些性质提出：基2-FFT、基4-FFT、分裂基FFT、DHT等快速傅立叶变换算法
- **基2-FFT分为时域抽取法FFT(DIT-FFT)和频域抽取法(DIF-FFT)。**

基2 DIT-FFT

- 设序列$x(n)$的长度为N，且满足$N=2^M$，M为自然数。按n的奇偶把分解为两个$N/2$点的子序列

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

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

- 收其DFT为

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

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

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

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

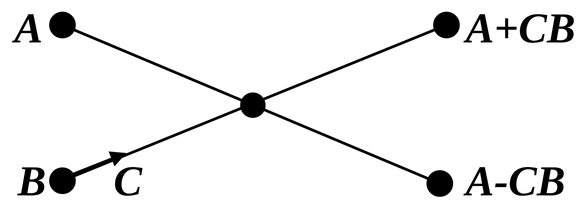
$$=X_1(k)+W_N^kX_2(k)，\ k=0,1,\cdots,N-1$$

基2 DIT-FFT

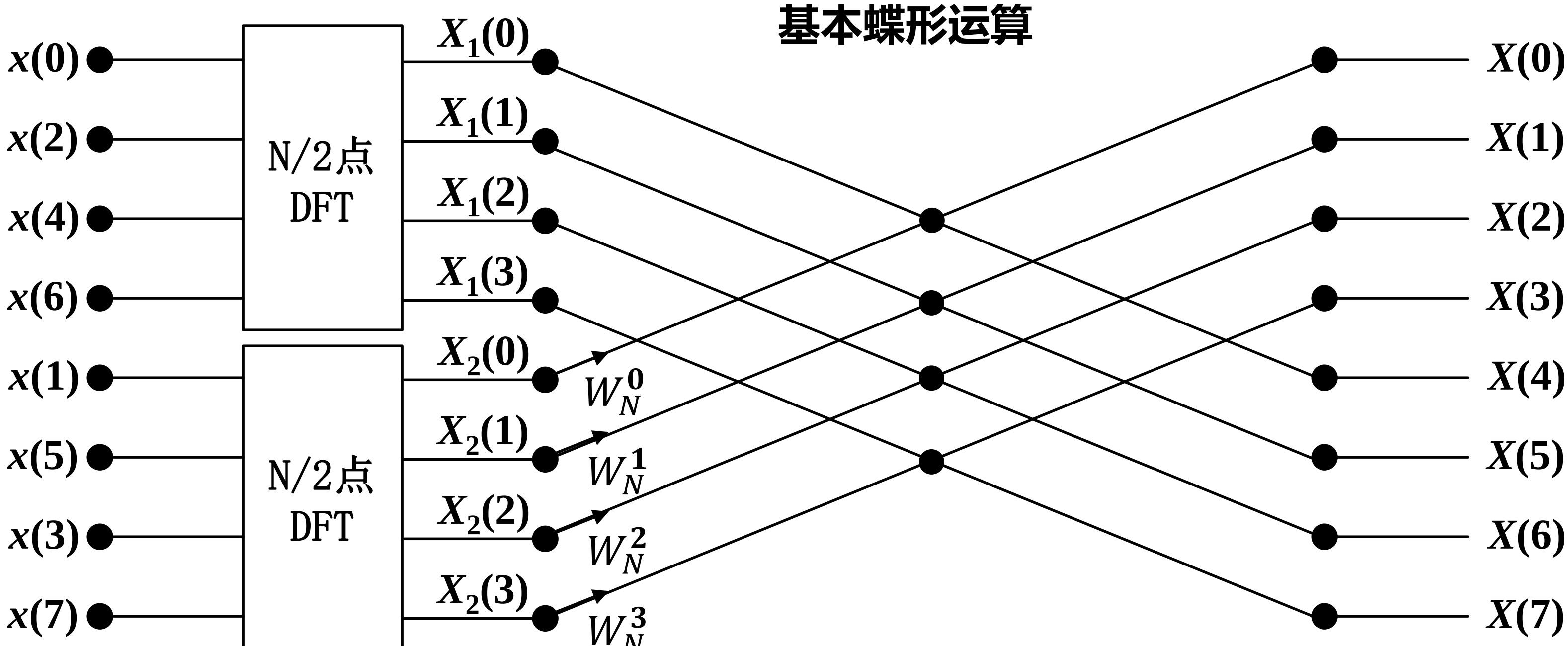
- 由于$X_1(k)$和$X_2(k)$以$N/2$为周期，且$W_N^{k+N/2}=-W_N^k$，因此

$$X(k)=X_1(k)+W_N^kX_2(k),k=0,1,\cdots,N/2-1$$

$$X(k+N/2)=X_1(k)-W_N^kX_2(k),k=0,1,\cdots,N/2-1$$



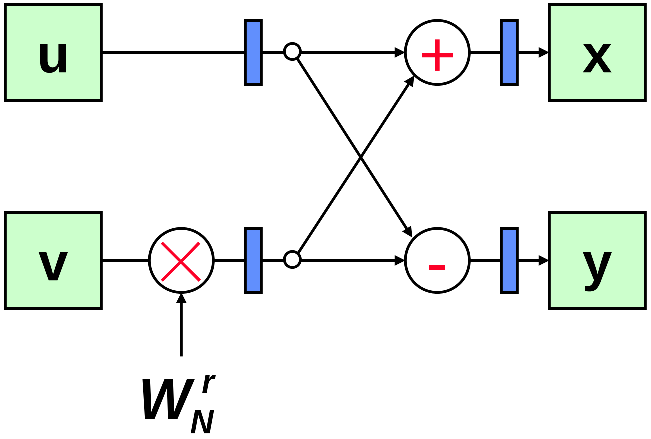
基本蝶形运算



计算量分析-复蝶形运算

大小	16	8192	Δ
I/O	448	229K	
蝶形	32	53K	
乘法			
加法			
移位	0	0	

- 一个蝶形包含：
  - 1个复加
  - 1个复减
  - 1个复常数乘法



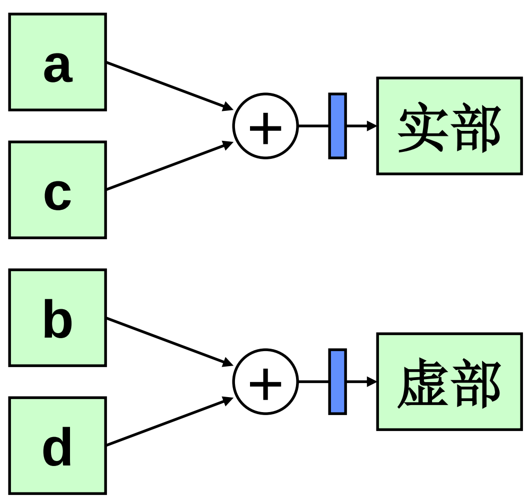
计算量分析- 复数加法

➢ 复数加法把实部和虚部分析相加：

$$(a+jb)+(c+jd)=(a+c)+j(b+d)$$

2个加法

大小	16	8192	Δ
I/O	448	229K	
蝶形	32	53K	
乘法			
加法	128	213K	
移位	0	0	



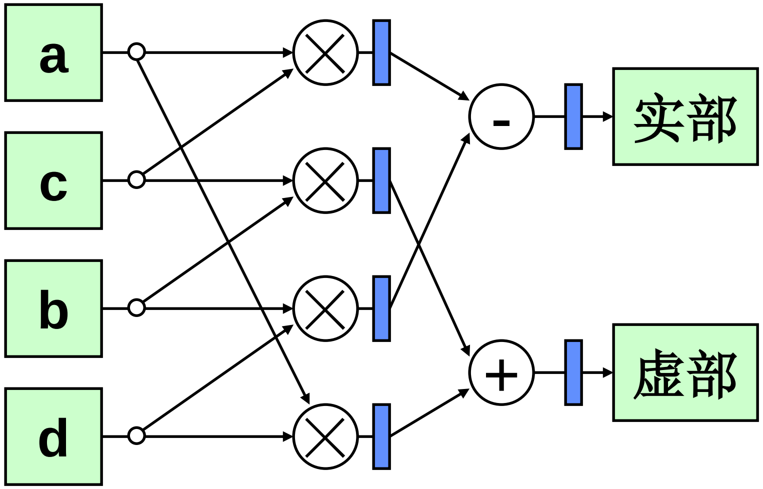
计算量分析- 复数乘法

大小	16	8192	Δ
I/O	448	229K	
蝶形	32	53K	
乘法	128	213K	
加法	192	320K	
移位	0	0	

- 复数乘法的FOIL方法:

$$(a+jb)(c+jd)=(ac-bd)+j(ad+bc)$$

4次乘法和两次加法



计算量分析- 高效的复数乘法

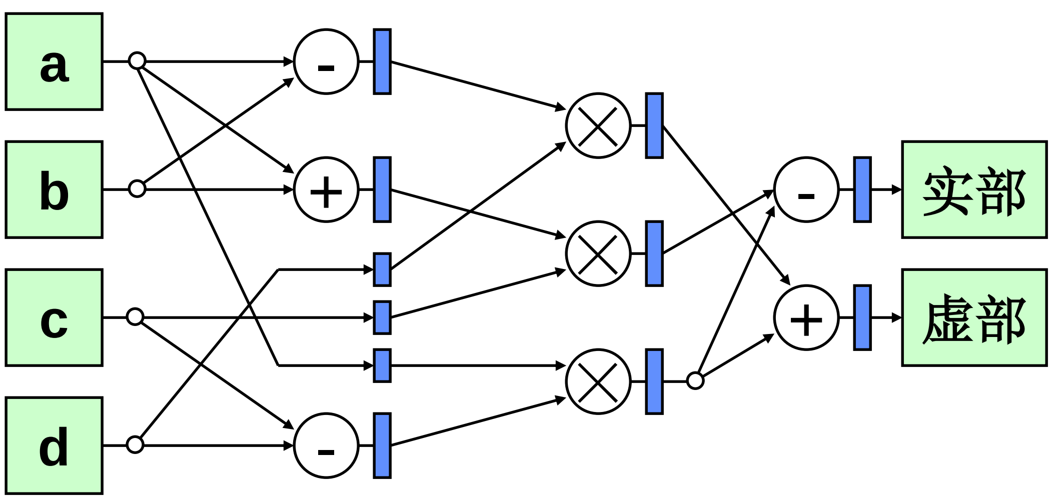
大小	16	8192	Δ
I/O	448	229K	
蝶形	32	53K	
乘法	96	159K	75%
加法	288	480K	150%
移位	0	0	

- 另一种复乘需要更少的乘法：

$$(ad+bc)=c(a+b)-a(c-d)$$

$$(ac-bd)=d(a-b)+a(c-d)$$

3次乘法和5次加法



基2 DIT-FFT

- 一个蝶形运算，包括一个复数乘法和两次复数加法。
- 经过一次分解，N点DFT需要计算两个$N/2$点DFT和$N/2$个蝶形运算。
- 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。所以总的复数乘法次数为

$$2\overset{\text{DFT}}{\boxed{\left(\frac{N}{2}\right)^2}}+\overset{\text{蝶形}}{\boxed{\frac{N}{2}}}=\left.\frac{N(N+1)}{2}\right|_{N\gg 1}\approx\frac{N^2}{2}$$

- 总的复数加法次数为

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

- 经过一次分解，运算量减小近一半！

基2 DIT-FFT

- 分别将$x_1(r)$和$x_2(r)$分解成两个点子序列，即

$$x_3(\boldsymbol{l}) = x_1(\mathbf{2}\boldsymbol{l}) \qquad x_5(\boldsymbol{l}) = x_2(\mathbf{2}\boldsymbol{l})$$

$$x_4(\boldsymbol{l}) = x_1(\mathbf{2}\boldsymbol{l}+\mathbf{1}) \qquad x_6(\boldsymbol{l}) = x_2(\mathbf{2}\boldsymbol{l}+\mathbf{1})$$

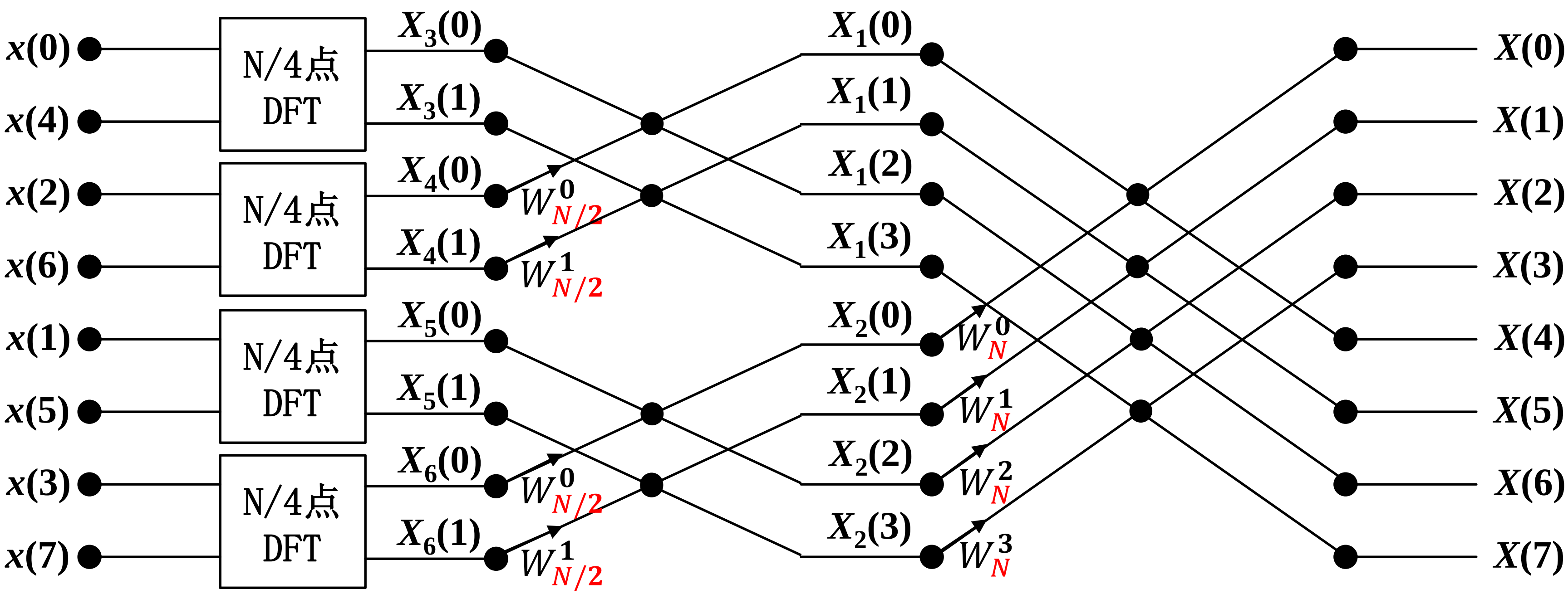
- 其中$\boldsymbol{l} = 0,1,\cdots,\frac{\boldsymbol{N}}{\mathbf{4}} - 1$。从而可得

$$X_1(k) = X_3(k) + W_{N/2}^{k} X_4(k)$$
$$X_1\left(k+\frac{N}{4}\right) = X_3(k) - W_{N/2}^{k} X_4(k)$$

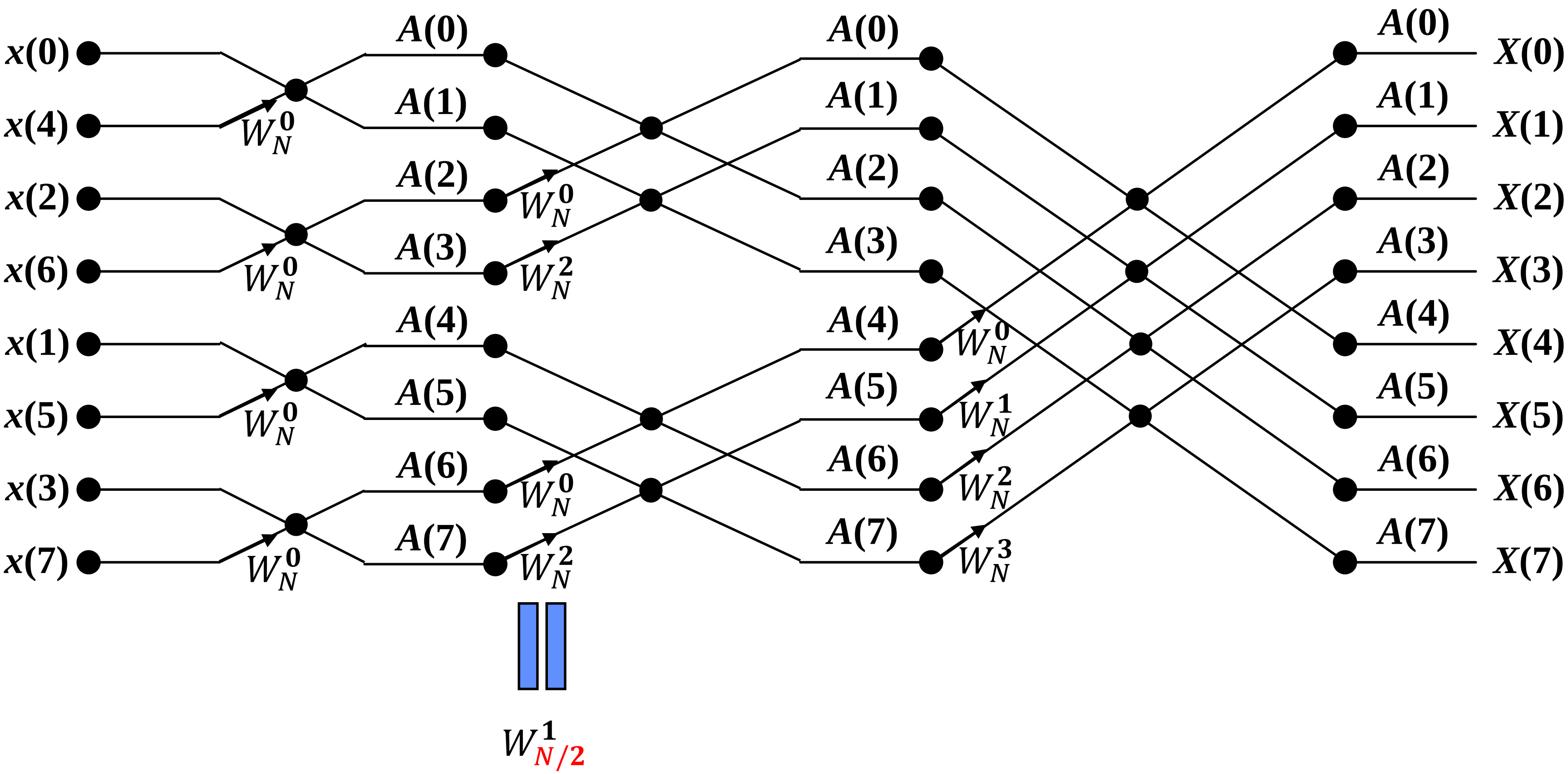
$$X_2(k) = X_5(k) + W_{N/2}^{k} X_6(k)$$
$$X_2\left(k+\frac{N}{4}\right) = X_5(k) - W_{N/2}^{k} X_6(k)$$

- $X_3(k) = \sum_{l=0}^{N/4-1} x_3(\boldsymbol{l})\ W_{N/4}^{kl} = \mathbf{DFT}[x_3(\boldsymbol{l})]_{N/4}$
- $X_4(k) = \mathbf{DFT}[x_4(\boldsymbol{l})]_{N/4}$，$X_5(k) = \mathbf{DFT}[x_5(\boldsymbol{l})]_{N/4}$，$X_6(k) = \mathbf{DFT}[x_6(\boldsymbol{l})]_{N/4}$

8点DFT二次抽取分解运算流图



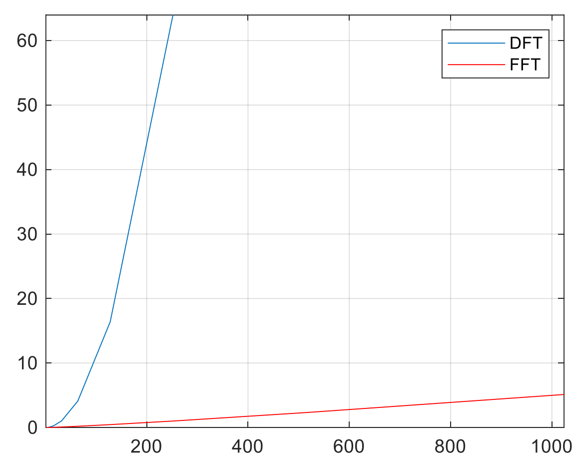
8点DFT运算流图

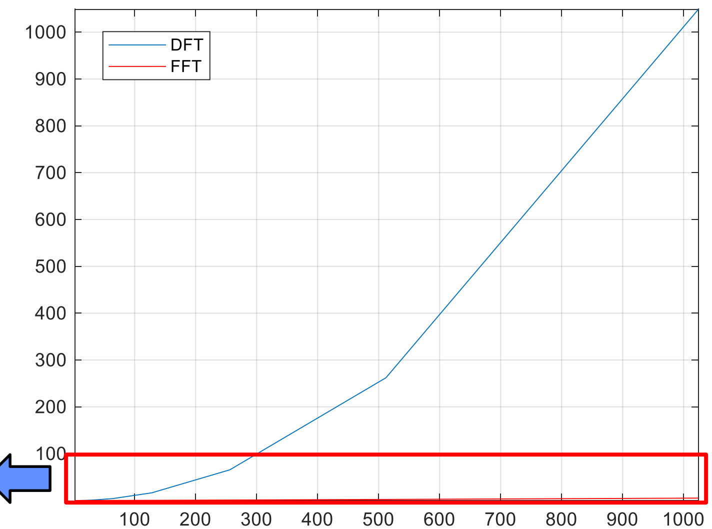

x(0)
x(4)
x(2)
x(6)
x(1)
x(5)
x(3)
x(7)
A(0)
A(1)
A(2)
A(3)
A(4)
A(5)
A(6)
A(7)
X(0)
X(1)
X(2)
X(3)
X(4)
X(5)
X(6)
X(7)
W_N^0
W_N^1
W_N^2
W_N^3
$W_{N/2}^1$

DIT-FFT与DFT运算量比较

- DIT-FFT的复数乘法次数为$C_M = \frac{N}{2}M = \frac{N}{2}\log_2 N$，复数加法次数为$C_A = NM = N\log_2 N$。其中$M = \log_2 N$。
- 直接计算DFT的复乘次数为N^2，复数加法次数为$N(N-1)$。
- 加速比：$\boldsymbol{R} = \frac{N^2}{\frac{N}{2}\log_2 N}$
- $N = \mathbf{1024}$时，$\boldsymbol{R} = 204.8$倍。

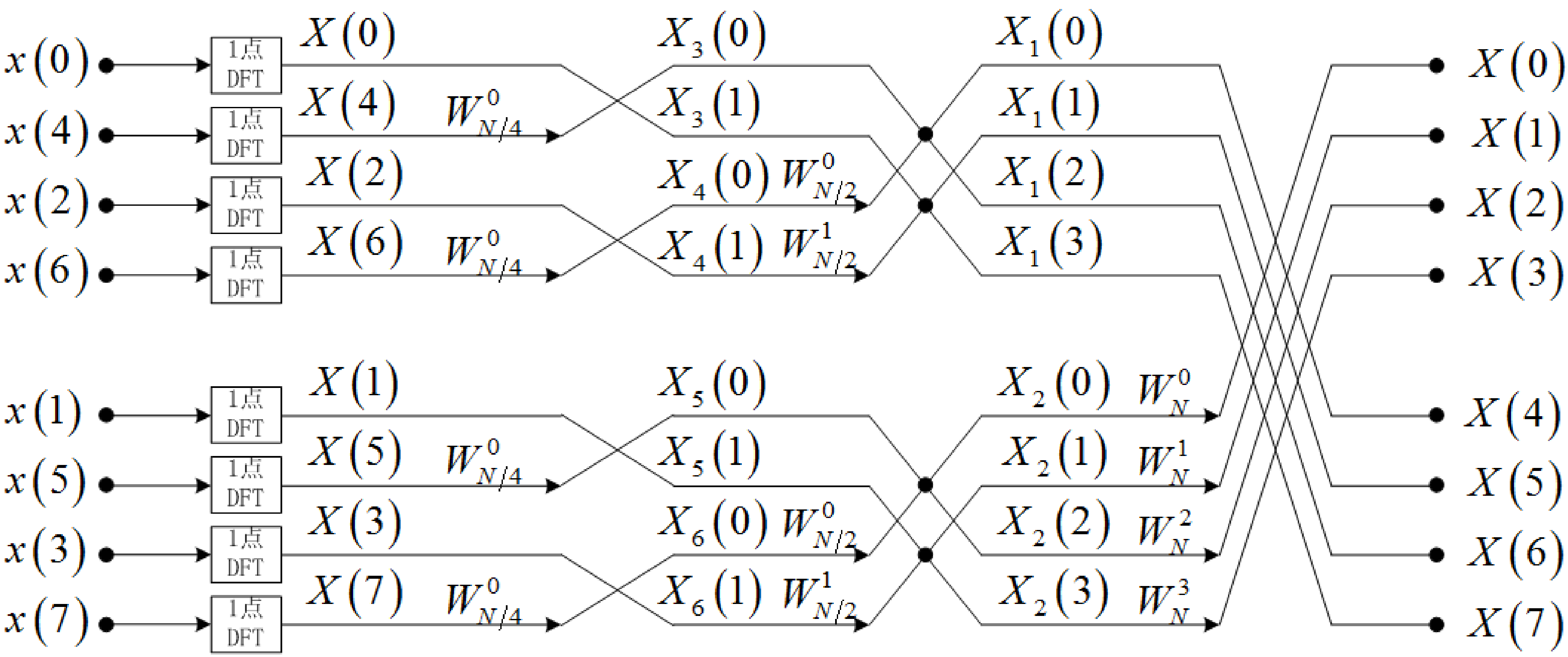
书上的图貌似不准确？！





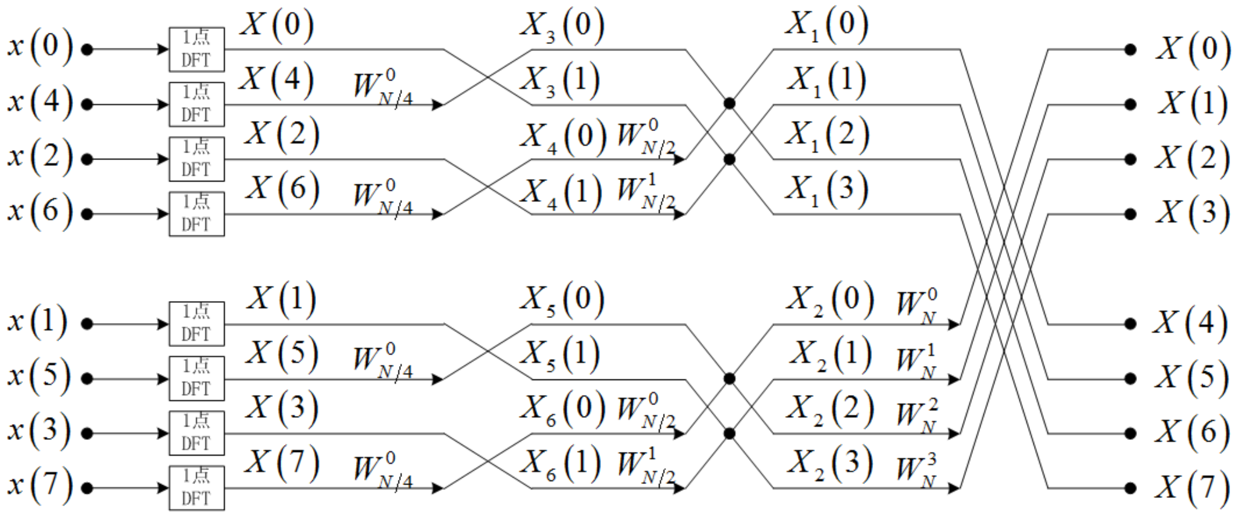
DIT-FFT运算规律

➤ **原位计算**：运算过程无需增加新的存储单元



DIT-FFT运算规律

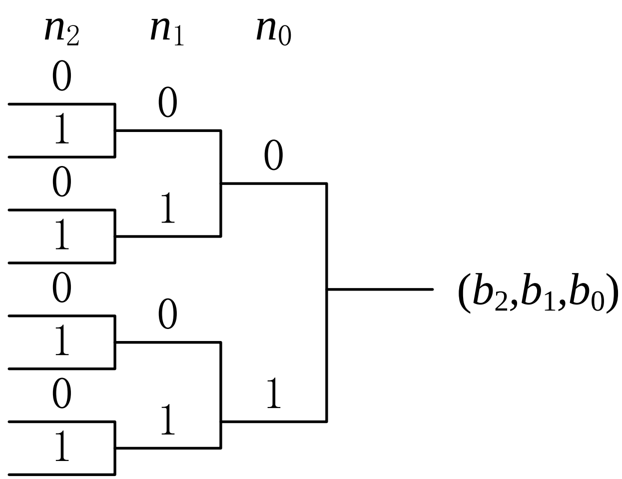
- 旋转因子的变化规律和蝶形节点间距



- 第1级蝶形运算旋转因子为W_N^0，蝶形节点间距为1
- 第2级蝶形运算旋转因子为W_N^0、$W_N^{N/4}$，蝶形节点间距为2
- 第3级蝶形运算旋转因子为W_N^0、$W_N^{N/8}$、$W_N^{2N/8}$、$W_N^{3N/8}$，蝶形节点间距为4
- 第M级蝶形运算旋转因子为W_N^0，W_N^1，…，$W_N^{N/2-1}$，蝶形节点间距为$N/2$

序列的倒序

形成倒序树状图

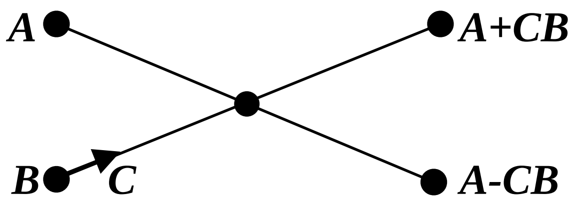


$b_2b_1b_0$	
000	0
100	4
010	2
110	6
001	1
101	5
011	3
111	7

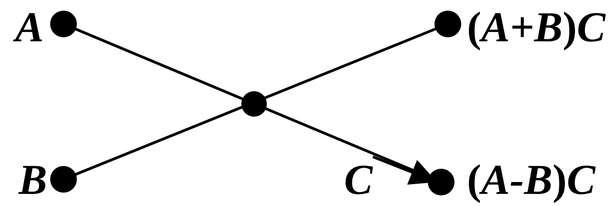
倒序和顺序对照表

顺序		倒序	
十进制数I	二进制数	二进制数	十进制数J
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

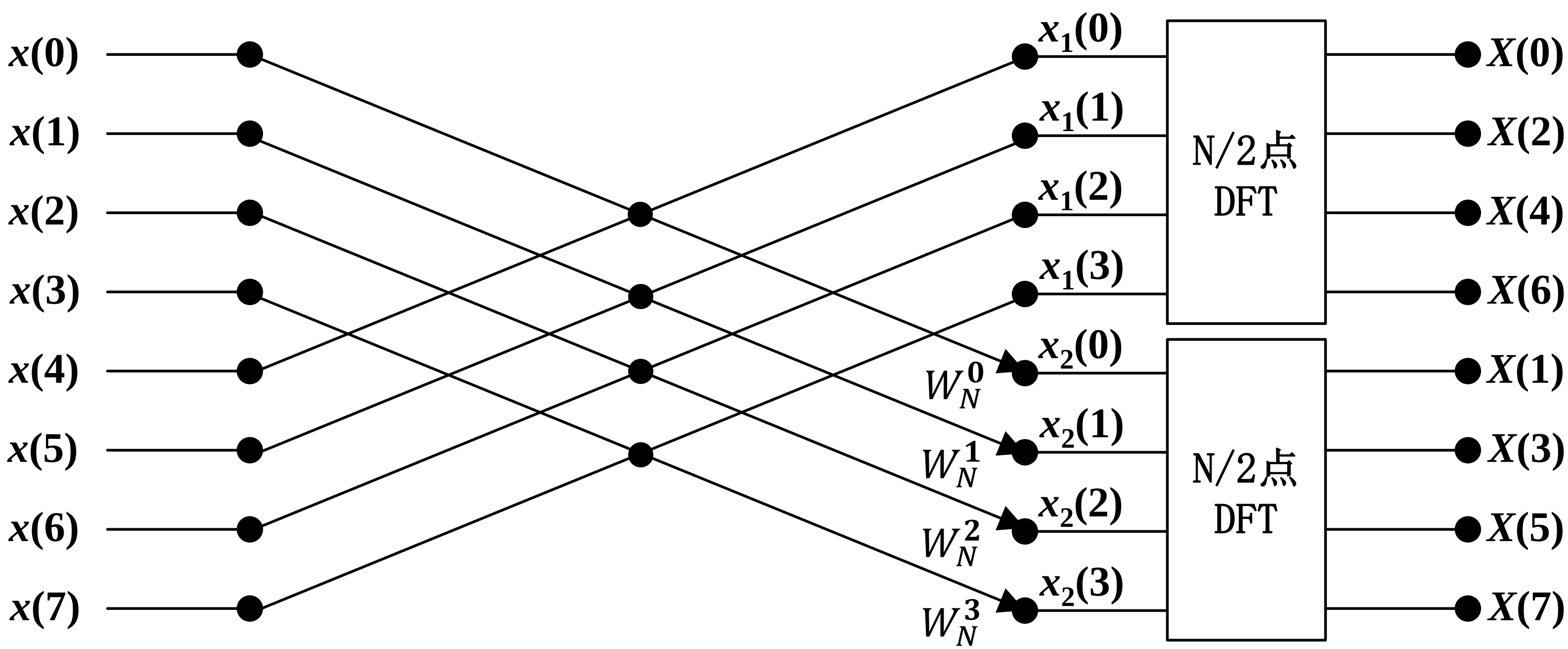
频域抽取DIF-FFT



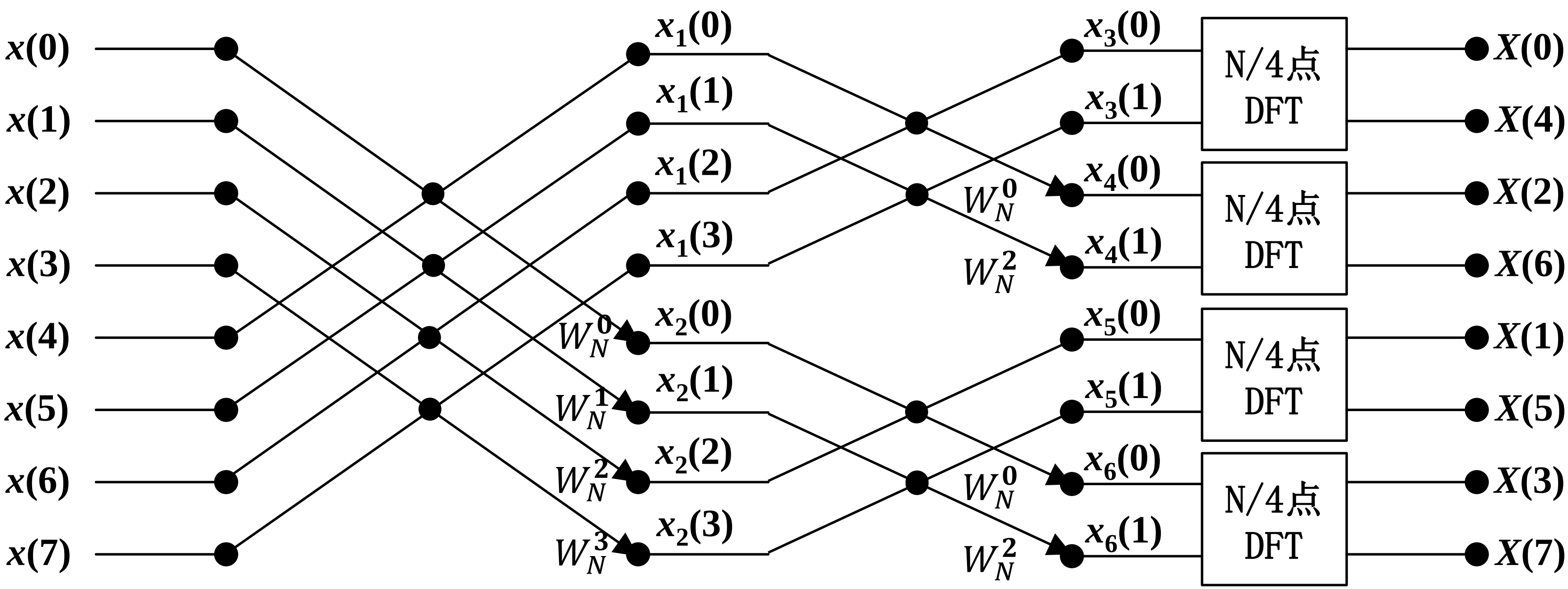
时域抽取基本蝶形运算



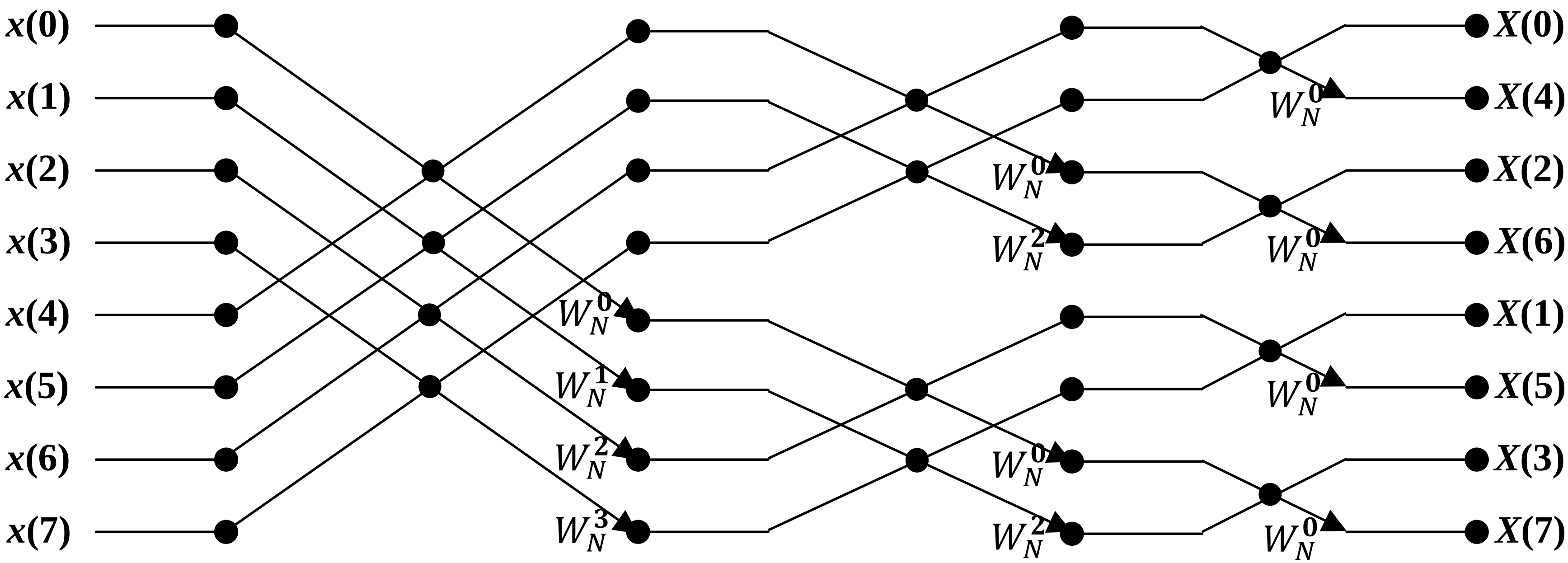
频域抽取基本蝶形运算



DIF-DFT二次抽取分解运算流图



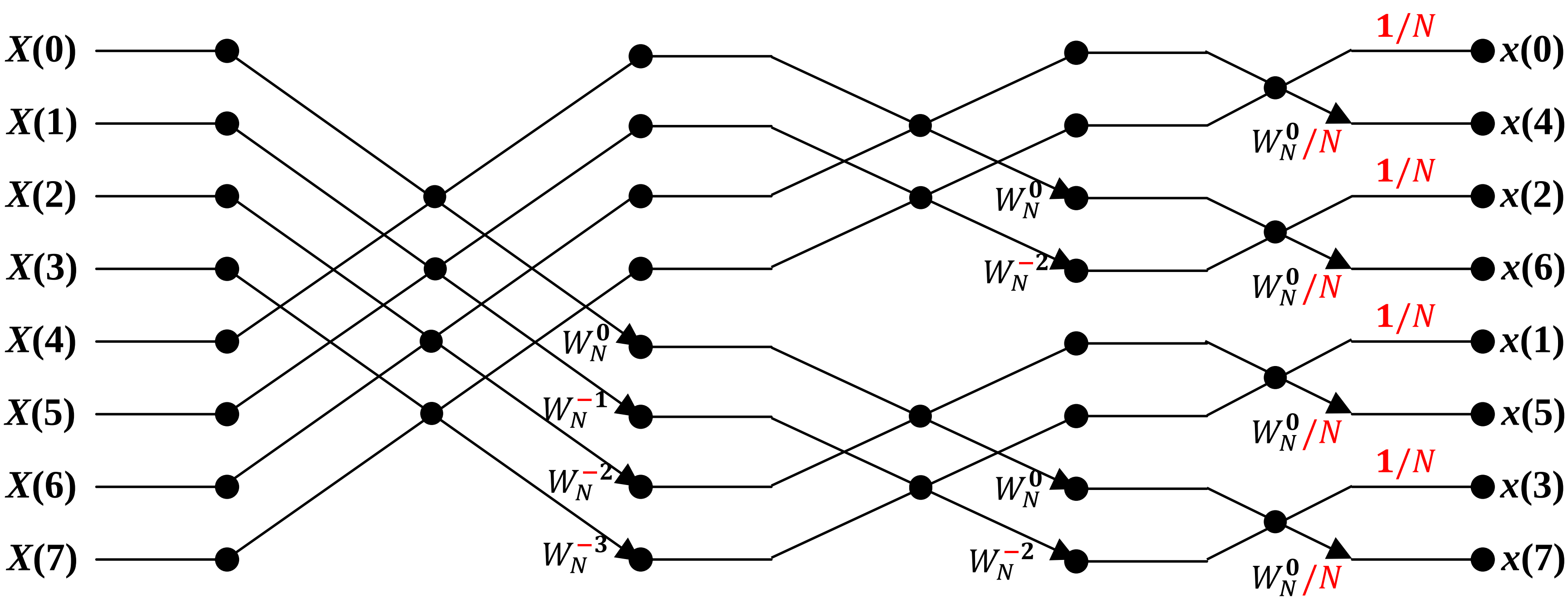
8点DIF-DFT运算流图



- 仔细观察基2DIF-FFT运算流图和基2DIT-FFT运算流图会发现，将频域抽取法的运算流图反转，并将输入变输出，输出变输入，正好得到时域抽取法的运算流图。(实际上我做PPT时，就是直接在PPT里反转一下就得到了☺)
- 按频域抽取算法与按时域抽取算法是两种等价的FFT算法，此外，在基2FFT的基础上，还有变形基2FFT运算流图，原理类似。

IDFT高效算法

➢旋转因子指数变极性法：W_N^{kn}变成W_N^{-kn}，再乘以系数$1/N$。



➢直接调用FFT子程序：

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

用MATLAB编写自己的FFT代码

myfft.m

```
function [A] = myfft(A,M)
N=2^M; LH=N/2; J=LH; N1=N-2;
for I=1:1:N1  %完成输入的倒序
   if I<J     %I=1时，A(2)和A(5)对换
       T=A(I+1);
       A(I+1)=A(J+1);
       A(J+1)=T;
    end
    K=LH;
    while J>=K
        J=J-K;
        K=K/2;
    end
    J=J+K;
end
```

用MATLAB编写自己的FFT代码

```
for L=1:1:M    %第L级蝶形
    B=2^(L-1);
    for J=0:B-1
        p=J*2^(M-L); %旋转因子系数
        for k=J:2^L:N-1      %计算蝶形
            T=A(k+1)+A(k+B+1)*exp(-1i*2*pi*p/N);
            A(k+B+1)=A(k+1)-A(k+B+1)*exp(-1i*2*pi*p/N);
            A(k+1)=T;
        end
    end
end
end
```

用MATLAB编写自己的FFT代码

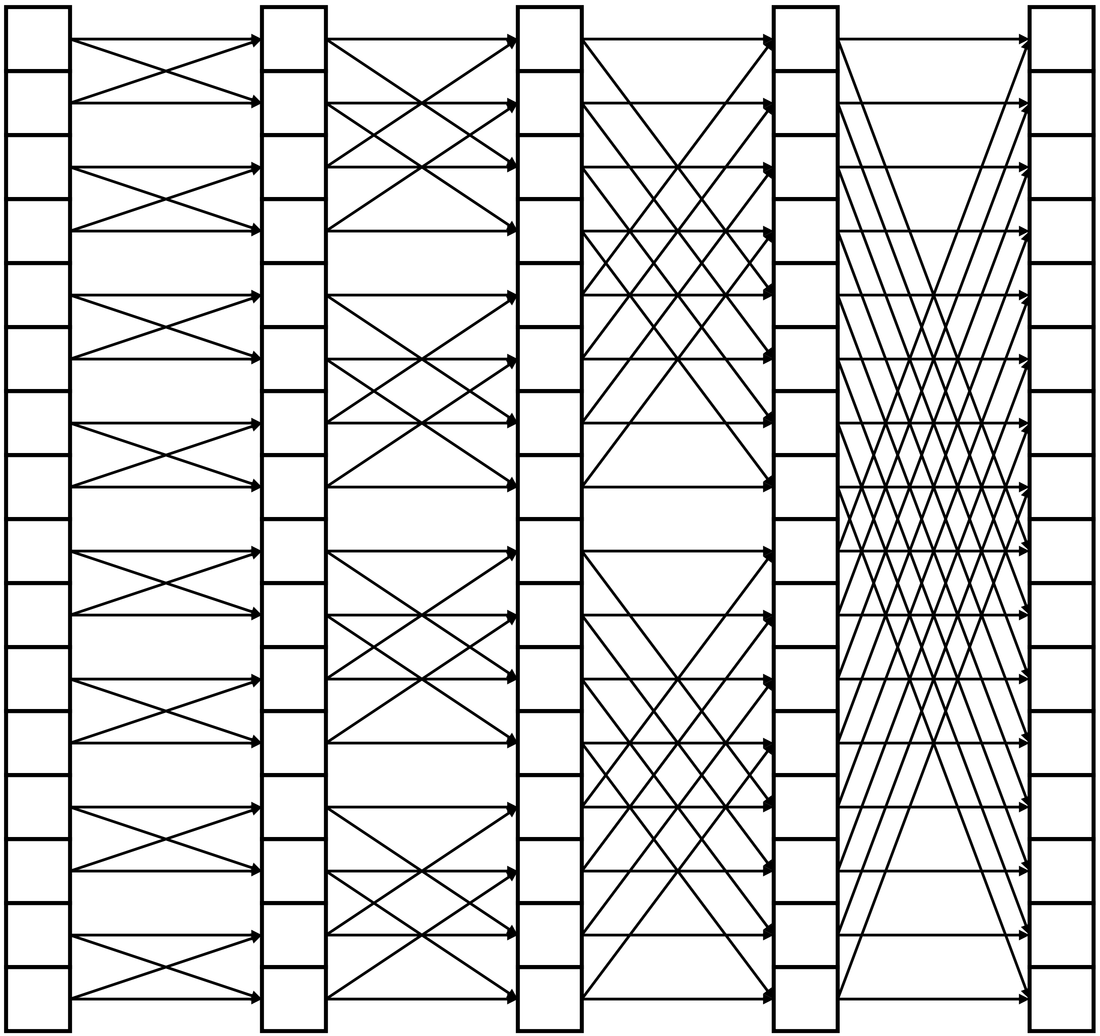
```
N=1024;
sig=randn(1,N)+1i*randn(1,N);
tic
for i=1:1000
    fft(sig);
end
toc

tic
for i=1:1000
    myfft(sig,10);
end
toc
```

FFT_Test.m

运行结果：
FFT_Test
时间已过 0.013525 秒。
时间已过 3.143415 秒。

FPGA里FFT的基本并行结构

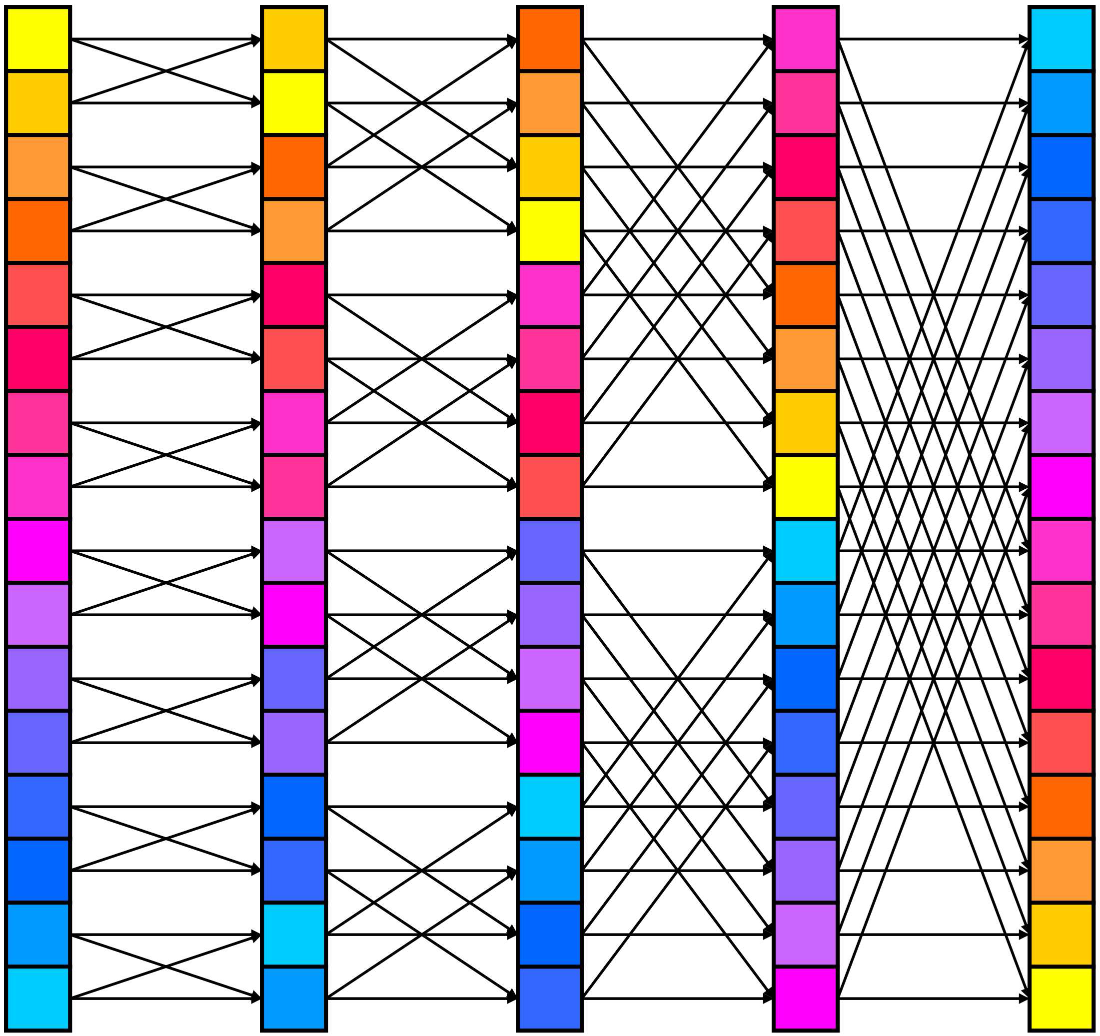


大小	16	8192	Δ
I/O	448	229K	
蝶形	32	53K	
乘法			
加法			
移位	0	0	

并行FFT

- 蝶形结构
- 与直接计算DFT相比，移除了冗余计算

FPGA里FFT的并行流水线结构

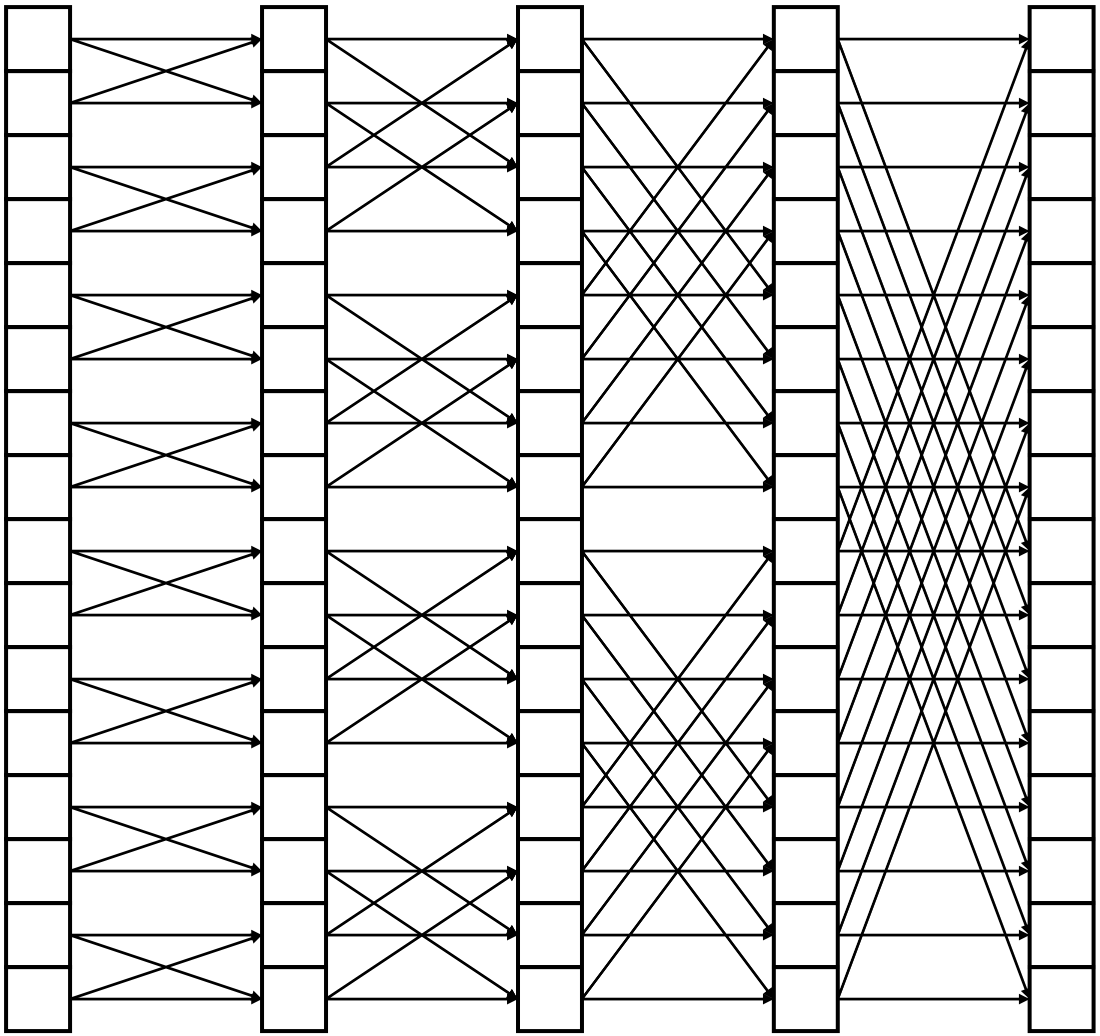


大小	16	8192	Δ
I/O	448	229K	
蝶形	32	53K	
乘法	96	159K	
加法	288	480K	
移位	0	0	

Pipelined版本

- 受到I/O的限制，实际上无法实现。8192点输入管脚需要65606个！
- 100%有效性

FPGA里FFT的串行输入



大小	16	8192	Δ
I/O	28	28	.01%
蝶形	32	53K	
乘法	96	159K	
加法	288	480K	
移位	0	0	

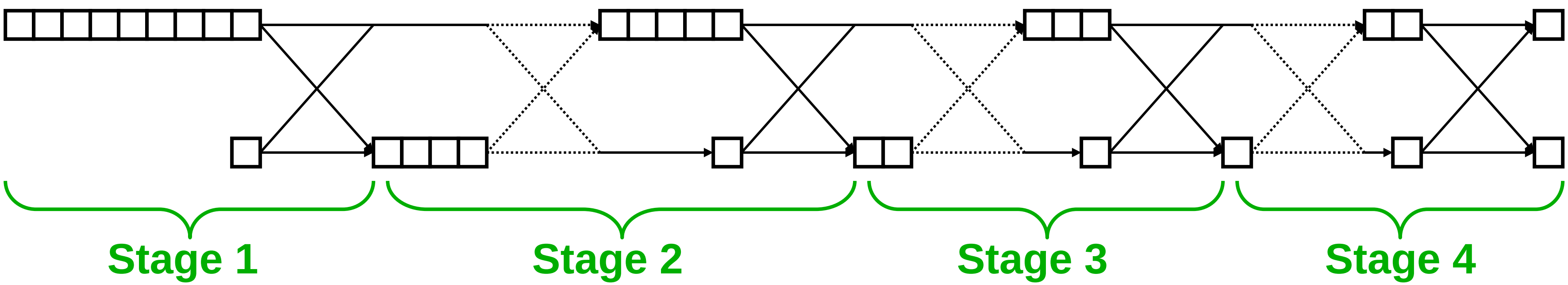
A serial versI/On

- **I/O速率与A/D匹配**
- **6.25%的有效性**

FPGA里FFT的串行架构

大小	16	8192	Δ
I/O	28	28	
蝶形	4	13	.03%
乘法	12	39	.03%
加法	36	117	.03%
移位	22	12K	

- 并行结构可以分解为
  – 每一层一个蝶形
  – 每个样本只占用一个时钟周期
  – 同样的延时和速率
  – 设计更有效

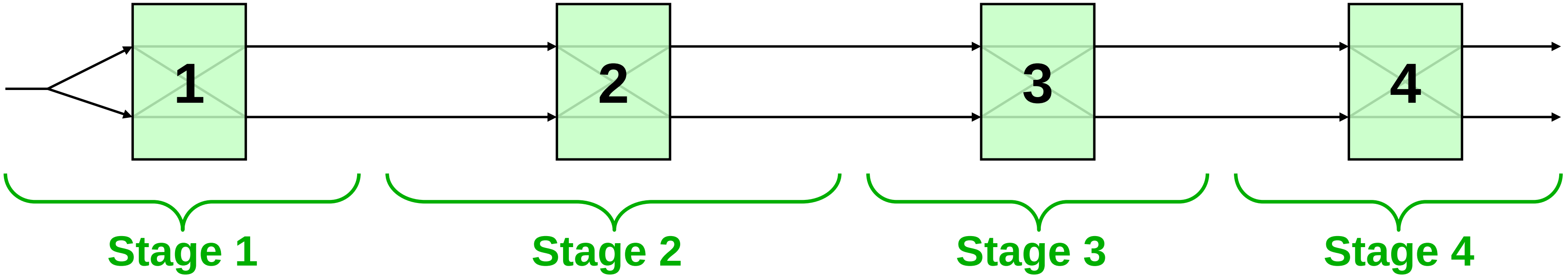


50%的有效性

高层视角

大小	16	8192	Δ
I/O	28	28	
蝶形	4	13	
乘法	12	39	
加法	36	117	
移位	22	12K	

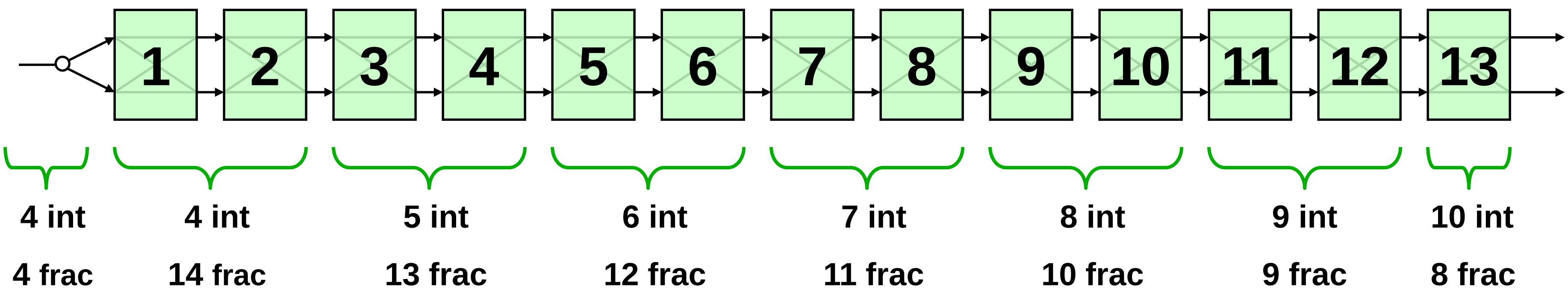
- 用一个包含下述组件的单元代替复杂的结构：
  - FIFOs
  - 蝶形
  - 交换网络



8192点FFT结构

- 需要13层
- 定点计算
- 通过高速动态范围以提高精度
- 溢出替换为饱和值

大小	16	8192	Δ
I/O	28	28	
蝶形	4	13	
乘法	12	39	
加法	36	117	
移位	22	12K	



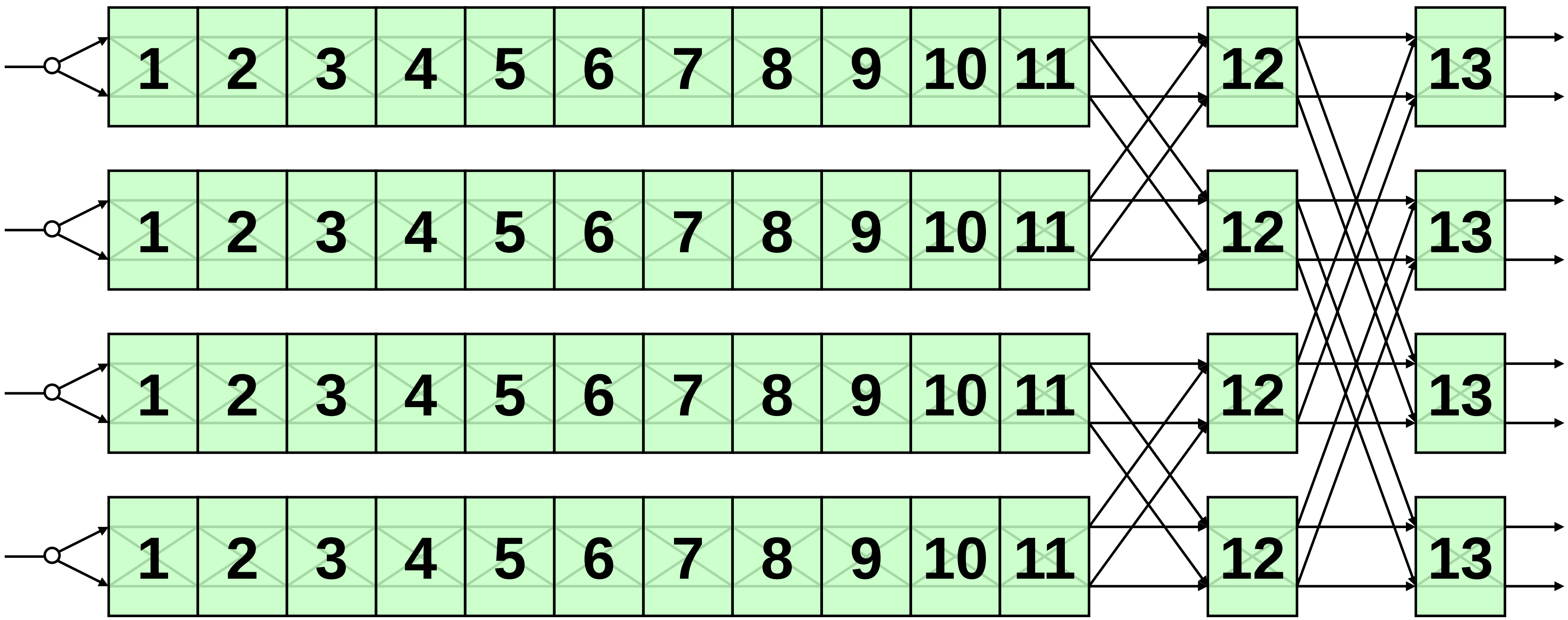
$$\underbrace{0110}.\underbrace{0101}$$

$$6 + \frac{5}{16}$$

提高并行度

增加更多的并行通道
- 乘法器速率150 MHz
- I/Q组件产生600 MSPS的数据
- 通过并行实现实时处理

大小	16	8192	Δ
I/O	112	112	400%
蝶形	16	52	400%
乘法	48	156	400%
加法	144	468	400%
移位	16	12K	100%



快速傅里叶变换(FFT)

- 引言
- 基2 FFT算法
- 进一步减小运算量的措施
- 其他快速算法简介

快速傅里叶变换(FFT)

- 引言
- 基2 FFT算法
- 进一步减小运算量的措施
- 其他快速算法简介

作业

1. 课后习题P116-121：1，5

谢谢！