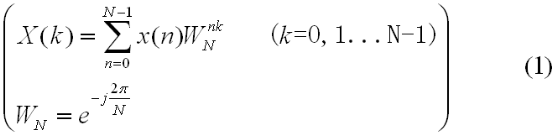
## FFT的计算

在数字信号处理中常常需要用到离散傅立叶变换(DFT)，以获取信号的频域特征。尽管传统的DFT算法能够获取信号频域特征，但是算法计算量大，耗时长，不利于计算机实时对信号进行处理。因此至DFT被发现以来，在很长的一段时间内都不能被应用到实际的工程项目中，直到一种快速的离散傅立叶计算方法——FFT，被发现，离散傅立叶变换才在实际的工程中得到广泛应用。需要强调的是，FFT并不是一种新的频域特征获取方式，而是DFT的一种快速实现算法。本文就FFT的原理以及具体实现过程进行详尽讲解。

### DFT计算公式

**本文不加推导地直接给出DFT的计算公式：**

[](http://img.ph.126.net/7VqiMvu-GcAbGmw6Olr0FA==/3844103756937664543)

其中x(n)表示输入的离散数字信号序列，WN为旋转因子，X(k)为输入序列x(n)对应的N个离散频率点的相对幅度。一般情况下，假设x(n)来自于低通采样，采样频率为fs，那么X(k)表示了从-fs/2率开始，频率间隔为fs/N,到fs/2-fs/N截至的N个频率点的相对幅度。因为DFT计算得到的一组离散频率幅度值实际上是在频率轴上从成周期变化的，即X(k+N)=X(k)。因此任意取连续的N个点均可以表示DFT的计算效果，负频率成分比较抽象，难于理解，根据X(k)的周期特性，于是我们又可以认为X(k)表示了从零频率开始，频率间隔为fs/N,到fs-fs/N截至的N个频率点的相对幅度。

### N点DFT的计算量

根据(1)式给出的DFT计算公式，我们可以知道每计算一个频率点X(k)均需要进行N次复数乘法和N-1次复数加法，计算N各点的X(k)共需要N^2次复数乘法和N\*(N-1)次复数加法。当x(n)为实数的情况下，计算N点的DFT需要2\*N^2次实数乘法，2\*N\*(N-1)次实数加法。

### 旋转因子WN的特性

  1.WN的对称性

[WN对称性](http://img.ph.126.net/UqC2jvMUWcbrZhPcKU4uaA==/3845792606797927785)        2.WN的周期性

[WN周期性](http://img.ph.126.net/jppxRsJ7cUV2q19_K-tQdg==/2380715353019292390)

  3.WN的可约性

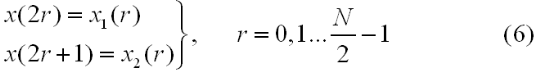
WN可约性

根据以上这些性质，我们可以得到式(5)的一系列有用结果

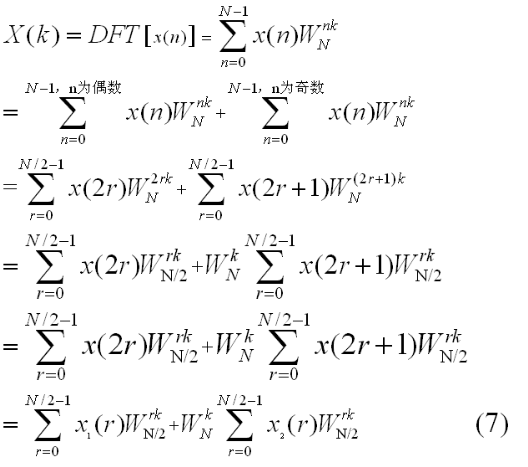
WN性质

### 基-2 FFT算法推导

假设采样序列点数为N=2^L，L为整数，如果不满足这个条件可以人为地添加若干个0以使采样序列点数满足这一要求。首先我们将序列x(n)按照奇偶分为两组如下：

[](http://img.ph.126.net/pQB61KN6N8-meqAzmjVZgw==/2380715353019292393)

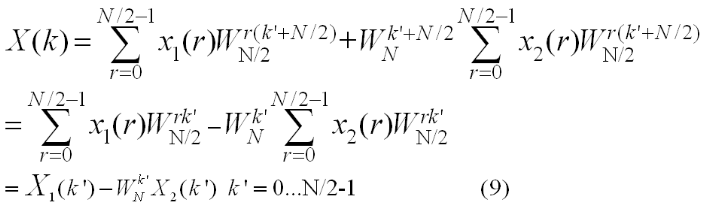
于是根据DFT计算公式(1)有：



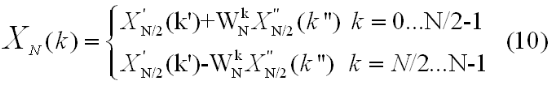
至此，我们将一个N点的DFT转化为了式(7)的形式，此时k的取值为0到N-1,现在分为两段来讨论，当k为0～N/2-1的时候，因为x1(r)，x2(r)为N/2点的序列，因此，此时式(7)可以写为：

FFT偶数部分

而当 k取值为N/2~N-1时，k用k’+N/2取代，k’取值为0~N/2-1。对式(7)化简可得：



综合以上推导我们可以得到如下结论：一个N点的DFT变换过程可以用两个N/2点的DFT变换过程来表示，其具体公式如式(10)所示DFT快速算法的迭代公式：



上式中X'(k’)为偶数项分支的离散傅立叶变换，X''(k’’)为奇数项分支的离散傅立叶变换。 式(10)的计算过程可以用图1的蝶形算法流图直观地表示出来。

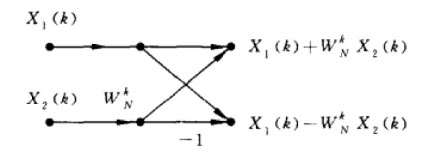


图1 时间抽取法蝶形运算流图

在图1中，输入为两个N/2点的DFT输出为一个N点的DFT结果，输入输出点数一致。运用这种表示方法，8点的DFT可以用图2来表示：

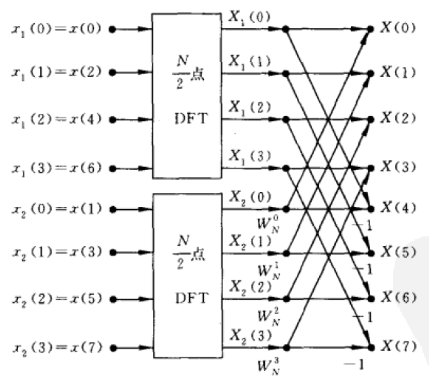


图2 8点DFT的4点分解

根据公式(10)，一个N点的DFT可以由两个N/2点的DFT运算构成，再结合图1的蝶形信号流图可以得到图2的8点DFT的第一次分解。该分解可以用以下几个步骤来描述：

  1.将N点的输入序列按奇偶分为2组分别为N/2点的序列

  2.分别对1中的每组序列进行DFT变换得到两组点数为N/2的DFT变换值X1和X2

  3.按照蝶形信号流图将2的结果组合为一个N点的DFT变换结果

根据式(10)我们可以对图2中的4点DFT进一步分解，得到图3的结果，分解步骤和前面一致。

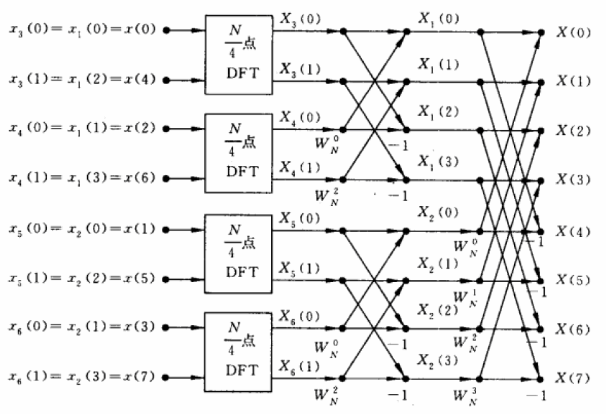
[](http://img.ph.126.net/u_rEpZZHW70oBWWqoRK65A==/3845792606797927790)

图3 8点DFT的2点分解

最后对2点DFT进一步分解得到最终的8点FFT信号计算流图：

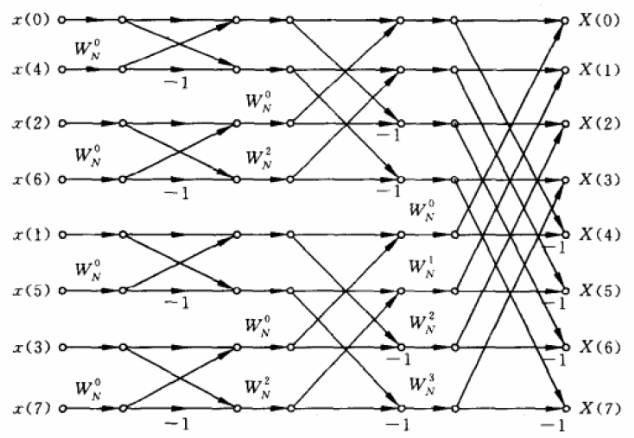


图4 8点DFT的全分解

从图2到图4的过程中关于旋转系数的变化规律需要说明一下。看起来似乎向前推一级，在奇数分组部分的旋转系数因子增量似乎就要变大，其实不是这样。事实上奇数分组部分的旋转因子指数每次增量固定为1，只是因为每向前推进一次，该分组序列的数据个数变少了，为了统一使用以原数据N为基的旋转因子就进行了变换导致的。每一次分组奇数部分的系数WN,这里的N均为本次分组前的序列点数。以上边的8点DFT为例，第一次分组N=8,第二次分组N为4，为了统一根据式(4)进行了变换将N变为了8，但指数相应的需要乘以2。

### N点基-2 FFT算法的计算量

从图4可以看到N点DFT的FFT变换可以转为log2(N)级级联的蝶形运算，每一级均包含有N/2次蝶形计算。而每一个蝶形运算包含了1次复数乘法，2次复数加法。因此N点FFT计算的总计算量为：

复数乘法——N/2×log2(N)

复数加法——N×log2(N)。

假设被采样的序列为实数序列，那么也只有第一级的计算为实数与复数的混合计算，经过一次迭代后来的计算均变为复数计算，在这一点上和直接的DFT计算不一致。因此对于输入序列是复数还是实数对FFT算法的效率影响较小。一次复数乘法包含了4次实数乘法，2次实数加法，一次复数加法包含了2次复数加法。因此对于N点的FFT计算需要总共的

实数乘法数量为：2×N×log2(N)；

总的复数加法次数为：2xNxlog2(N)。

### N点基-2 FFT算法的实现方法

从图4我们可以总结出对于点数为N=2^L的DFT快速计算方法的流程：

**1.对于输入数据序列进行倒位序变换。**

该变换的目的是使输出能够得到X(0)~X(N-1)的顺序序列，同样以8点DFT为例，该变换将顺序输入序列x(0)~x(7)变为如图4的x(0),x(4),x(2),x(6),x(1),x(5),x(3),x(7)序列。其实现方法是：假设顺序输入序列一次村在A(0)~A(N-1)的数组元素中，首先我们将数组下标进行二进制化(例：对于点数为8的序列只需要LOG2(8) = 3位二进制序列表示，序号6就表示为110)。二进制化以后就是将二进制序列进行倒位，倒位的过程就是将原序列从右到左书写一次构成新的序列，例如序号为6的二进制表示为110，倒位后变为了011，即使十进制的3。第三步就是将倒位前和倒位后的序号对应的数据互换。依然以序号6为例，其互换过程如下：

temp = A(6); A(6) = A(3); A(3) = A(6);

实际上考虑到执行效率，如果对于每一次输入的数据都需要这个处理过程是非常浪费时间的。我们可以采用指向指针的指针来实现该过程，或者是采用指针数组来实现该过程。  
   
**2.蝶形运算的循环结构。**  
 从图4中我们可以看到对于点数为N = 2^L的fft运算，可以分解为L阶蝶形图级联，每一阶蝶形图内又分为M个蝶形组，每个蝶形组内包含K个蝶形。根据这一点我们就可以构造三阶循环来实现蝶形运算。编程过程需要注意旋转因子与蝶形阶数和蝶形分组内的蝶形个数存在关联。   
  
**3.浮点到定点转换需要注意的关键问题**   
上边的分析都是基于浮点运算来得到的结论，事实上大多数嵌入式系统对浮点运算支持甚微，因此在嵌入式系统中进行离散傅里叶变换一般都应该采用定点方式。对于简单的DFT运算从浮点到定点显得非常容易。根据式(1)，假设输入x(n)是经过AD采样的数字序列，AD位数为12位，则输入信号范围为0~4096。为了进行定点运算我们将旋转因子实部虚部同时扩大2^12倍，取整数部分代表旋转因子。之后，我们可以按照(1)式计算，得到的结果与原结果成比例关系，新的结果比原结果的2^12倍。但是，对于使用蝶形运算的fft我们不能采用这种简单的放大旋转因子转为整数计算的方式。因为fft是一个非对称迭代过程，假设我们对旋转因子进行了放大，根据蝶形流图我们可以发现其最终的结果是，不同的输入被放大了不同的倍数，对于第一个输入x(0)永远也不会放大。举一个更加形象的例子，还是以图4为例。从图中可以看出右侧的X(0)可以直接用下式表示： 

FFT原理与实现 - kanku - kanku的博客

从上式我们可以看到不同输入项所乘的旋转因子**个数**(注意这里是个数，就算是wn^0，也被考虑进去了，因为在没有放大时wn^0等于1，放大后所有旋转因子指数模均不为1，因此需要考虑)。这就导致输入不平衡，运算结果不正确。经查阅相关资料，比较妥善的做法是，首先对所有旋转因子都放大2^Q倍，Q必须要大于等于L，以保证不同旋转因子的差异化。旋转因子放大，为了保证其模为1，在每一次蝶形运算的乘积运算中我们需要将结果右移Q位来抵消这个放大，从而得到正确的结果。之所以采用放大倍数必须是2的整数次幂的原因也在于此，我们之后可以通过简单的右移位运算将之前的放大抵消，而右移位又代替了除法运算，大大节省了时间。

**4.计算过程中的溢出问题**

最后需要注意的一个问题就是计算过程中的溢出问题。在实际应用中，AD虽然有12位的位宽，但是采样得到的信号可能较小，例如可能在0~8之间波动，也就是说实际可能只有3位的情况。这种情况下为了在计算过程中不丢失信息，一般都需要先将输入数据左移P位进行放大处理，数据放大可能会导致溢出，从而使计算错误，而溢出的极限情况是这样：假设我们数据位宽为D位(不包括符号位)，AD采样位数B位，数字放大倍数P位，旋转因此放大倍数Q位，FFT级联运算带来的最大累加倍数L位。我们得到:

FFT原理与实现 - kanku - kanku的博客

假设AD位宽12，数据位宽32，符号位1位，因此有效位宽31位，采样点数N，那么我们可以得到log2(N)+P+Q<=19,假设点数128,又Q>=L可以得到放大倍数P<=5

## FFT的Mtalab实现和分析

FFT是离散傅立叶变换的快速算法，可以将一个信号变换到频域。有些信号在时域上是很难看出什么特征的，但是如果变换到频域之后，就很容易看出特征了。这就是很多信号分析采用FFT变换的原因。另外，FFT可以将一个信号的频谱提取出来，这在频谱分析方面也是经常用的。

虽然很多人都知道FFT是什么，可以用来做什么，怎么去做，但是却不知道FFT之后的结果是什意思、如何决定要使用多少点来做FFT。

现在就根据实际经验来说说FFT结果的具体物理意义。一个模拟信号，经过ADC采样之后，就变成了数字信号。采样定理告诉我们，采样频率要大于信号频率的两倍，这些我就不在此罗嗦了。

采样得到的数字信号，就可以做FFT变换了。N个采样点，经过FFT之后，就可以得到N个点的FFT结果。为了方便进行FFT运算，**通常N取2的整数次方。**  
    **假设采样频率为Fs，信号频率F，采样点数为N。那么FFT之后结果就是一个为N点的复数**。每一个点就对应着一个频率点。这个点的模值，就是该频率值下的幅度特性。具体跟原始信号的幅度有什么关系呢？假设原始信号的峰值为A，那么FFT的结果的每个点（除了第一个点直流分量之外）的模值就是A的N/2倍。而第一个点就是直流分量，它的模值就是直流分量的N倍。而每个点的相位呢，就是在该频率下的信号的相位。第一个点表示直流分量（即0Hz），而最后一个点N的再下一个点（实际上这个点是不存在的，这里是假设的第N+1个点，也可以看做是将第一个点分做两半分，另一半移到最后）则表示采样频率Fs，这中间被N-1个点平均分成N等份，每个点的频率依次增加。

例如某点n所表示的频率为：Fn=(n-1)\*Fs/N。由上面的公式可以看出，Fn所能分辨到频率为为Fs/N，如果采样频率Fs为1024Hz，采样点数为1024点，则可以分辨到1Hz。1024Hz的采样率采样1024点，刚好是1秒，也就是说，采样1秒时间的信号并做FFT，则结果可以分析到1Hz，如果采样2秒时间的信号并做FFT，则结果可以分析到0.5Hz。**如果要提高频率分辨力，则必须增加采样点数**，也即采样时间。频率分辨率和采样时间是倒数关系。  假设**FFT之后某点n用复数a+bi表示，那么这个复数的模就是An=根号a\*a+b\*b，相位就是Pn=atan2(b,a)**。

根据以上的结果，就可以计算出n点（n≠1，且n<=N/2）对应的信号的表达式为：An/(N/2)\*cos(2\*pi\*Fn\*t+Pn)，即2\*An/N\*cos(2\*pi\*Fn\*t+Pn)。对于n=1点的信号，是直流分量，幅度即为A1/N。

由于FFT结果的对称性，通常我们只使用前半部分的结果，即小于采样频率一半的结果。

假设我们有一个信号，它含有2V的直流分量，频率为50Hz、相位为-30度、幅度为3V的交流信号，以及一个频率为75Hz、相位为90度、幅度为1.5V的交流信号。用数学表达式就是如下：

S=2+3\*cos(2\*pi\*50\*t-pi\*30/180)+1.5\*cos(2\*pi\*75\*t+pi\*90/180)

式中cos参数为弧度，所以-30度和90度要分别换算成弧度。我们以256Hz的采样率对这个信号进行采样，总共采样256点。按照我们上面的分析，Fn=(n-1)\*Fs/N，我们可以知道，每两个点之间的间距就是1Hz，第n个点的频率就是n-1。我们的信号有3个频率：0Hz、50Hz、75Hz，应该分别在第1个点、第51个点、第76个点上出现峰值，其它各点应该接近0。实际情况如何呢？我们来看看FFT的结果的模值如图所示。



从图中我们可以看到，在第1点、第51点、和第76点附近有比较大的值。我们分别将这三个点附近的数据拿上来细看：  
1点： 512+0i  
2点： -2.6195E-14 - 1.4162E-13i   
3点： -2.8586E-14 - 1.1898E-13i  
  
50点：-6.2076E-13 - 2.1713E-12i  
51点：332.55 - 192i  
52点：-1.6707E-12 - 1.5241E-12i  
  
75点：-2.2199E-13 -1.0076E-12i  
76点：3.4315E-12 + 192i  
77点：-3.0263E-14 +7.5609E-13i  
     
    很明显，1点、51点、76点的值都比较大，它附近的点值都很小，可以认为是0，即在那些频率点上的信号幅度为0。接着，我们来计算各点的幅度值。分别计算这三个点的模值，  
结果如下：  
1点： 512  
51点：384  
76点：192  
    **按照公式，可以计算出直流分量为：512/N=512/256=2；50Hz信号的幅度为：384/(N/2)=384/(256/2)=3；75Hz信号的幅度为192/(N/2)=192/(256/2)=1.5。可见，从频谱分析出来的幅度是正确的。**

然后再来计算相位信息。直流信号没有相位可言，不用管它。**先计算50Hz信号的相位，atan2(-192, 332.55)=-0.5236,结果是弧度，换算为角度就是180\*(-0.5236)/pi=-30.0001。再计算75Hz信号的相位，atan2(192, 3.4315E-12)=1.5708弧度，换算成角度就是180\*1.5708/pi=90.0002。**可见，相位也是对的。根据FFT结果以及上面的分析计算，我们就可以写出信号的表达式了，它就是我们开始提供的信号。  
    总结：假设采样频率为Fs，采样点数为N，做FFT之后，某一点n（n从1开始）表示的频率为：Fn=(n-1)\*Fs/N；**该点的模值除以N/2就是对应该频率下的信号的幅度**（**对于直流信号是除以N**）；该点的相位即是对应该频率下的信号的相位。相位的计算可用函数atan2(b,a)计算。atan2(b,a)是求坐标为(a,b)点的角度值，范围从-pi到pi。要精确到xHz，则需要采样长度为1/x秒的信号，并做FFT。要提高频率分辨率，就需要增加采样点数，这在一些实际的应用中是不现实的，需要在较短的时间内完成分析。**解决这个问题的方法有频率细分法，比较简单的方法是采样比较短时间的信号，然后在后面补充一定数量的0，使其长度达到需要的点数，再做FFT，这在一定程度上能够提高频率分辨力。具体的频率细分法可参考相关文献。**

## FFT的FPGA实现

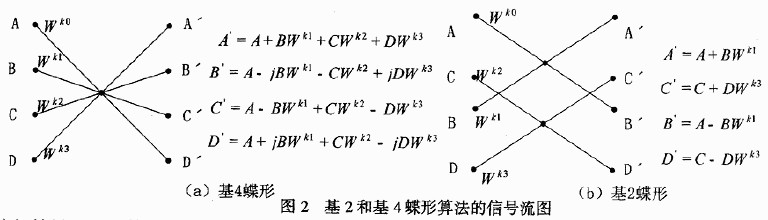
对于变换长度为N的序列x(n)其傅立叶变换可以表示如下：

当点数N较大时，必须对式(1)进行基4/基2分解，以短点数实现长点数的变换。而IDFT的实现在DFT的基础上就显得较为简单了：

由式(2)可以看出，在FFT运算模块的基础上，只需将输进序列进行取共轭后再进行FFT运算，输出结果再取一次共轭便实现了对输进序列的IDFT运算，因子1/N对于不同的数据表示格式具体实现时的处理方式是不一样的。IDFT在FFT的基础上输进和输出均有一次共轭操纵，但它们共用一个内核，仍然是十分方便的。

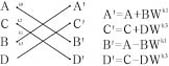
#### 基4和基2

基4和基2运算流图及信号之间的运算关系如图1所示：





1. 基４蝶形算法



（b）基２蝶形算法

以基4为例，令a=r0+j×i0；b=r1+j×i1；c=r2+j×i2；d=r3+j×i3；wk0=c0+j×s0：wk1=c1+j×s1；wk2=c2+j×s2；wk3=c3+j×s3。

分别代入图1中的基4运算的四个等式中有：

a'=[r0+(r1×c1-i1×s1)+(r2×c2-i2×s2)+(r3×c3-i3×s3)]+j[i0+(i1×c1+r1×s1)+(i2×c2+r2×s2)+(i3×c3+r3×s3)] 式(3)

b'=[r0+(i1×c1+r1×s1)-(r2×c2-i2×s2)-(i3×c3+r3×s3)]+j[i0-(r1×c1-i1×s1)-(i2×c2+r2×s2)+(r3×c3-i3×s3)] 式(4)

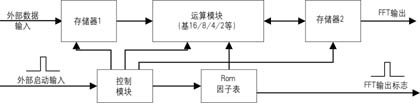
c'=[r0-(r1×c1-i1×s1)+(r2×c2-i2×s2)-(r3×c3-i3×s3)]+j[i0-(i1×c1+r1×s1)+(i2×c2+r2×s2)-(i3×c3+r3×s3)] 式(5)

d'=[r0-(i1×c1+r1×s1)-(r2×c2-i2×s2)+(i3×c3+r3×s3)]+j[i0+(r1×c1-i1×s1)-(i2×c2+r2×s2)-(r3×c3-i3×s3)] 式(6)

可以看出，式(3)至式(6)有多个公共项和类似项，这一点得到充分利用之后可以大大缩减基4和基2运算模块中的乘法器的个数，如上面a'至d'的四个等式中的这三对类似项：(r1×c1-i1×s1)与(i1×c1+r1×s1)、(r2×c2-i2×s2)与(i2×c2+r2×s2)、(r3×c3-i3×s3)与(i3×c3+r3×s3)以高于输入数据率的时钟进行时分复用，最终可以做到只需要3个甚至1个复数乘法器便可以实现。基2运算之所以采用图1-(b)中的形式进行基2运算，是为了将基本模块做成基4/2复用模块，它对于n有着更大的适用性和可借鉴性。在基4、基2和基4/2模块的基础上，构建基16、基8和基16/8模块有着非常大的意义。

算法实现

傅立叶变换实现时首先进行基2、基4分解，一般来说，如果算法使用基4实现，虽然使用的资源多了一些，但速度上的好处足以弥补。如果资源充足，使用基16、基8或基16/8复用模块，速度可以大大提高。一般fft实现简单框图如图2所示。



在图2中，运算模块即为基2/4/8/16模块或它们的复用模块，rom表中存储的是n点旋转因子表。控制模块产生所有的控制信号，存储器1和2的读写地址、写使能、运算模块的启动信号及因子表的读地址等信号。当然对于运算模块为基16/8复用模块时，控制模块就需要产生模式选择信号，如对于运算模块是基4/2模块时，该信号就决定了内部运算模块是进行基4运算还是基2运算。存储器1作为当前输入标志对应输入n点数据的缓冲器，存储器2作为中间结果存储器，用于存储运算模块计算出的各pass的结果。在图中的各种地址、使能和数据的紧密配合下，经过一定延时后输出计算结果及其对应指示标志。图2只是一定点或浮点的fft实现模块，如果是块浮点运算，则必须加入一个数据因子控制器，控制每遍运算过程中的数据大小，并根据各个pass的乘性因子之和的大小，对最终输出进行大小控制，以保证每段fft运算输出增益一致。

外部输入为n点数据段流和启动信号(n点之间如无间隔，则每n数据点输入一脉冲信号)，一方面，外部数据存入存储器1中，同时通过控制模块的控制，读出存储器1中的前段n点数据和rom表中的因子及相关控制信号送入运算核心模块进行各个pass的运算，每个pass的输出都存入存储器2中，最后一个pass的计算结果存入存储器2中，并在下一个启动头到来后，输出计算结果。对图2的实现，除去运算模块，关键是各个pass数据因子读写地址及控制信号的配合。

速度、资源和精度

假定输入数据的速率为fin，则每数据的持续时间t=1/fin，运算模块的计算时钟频率为fa，对于n(n=2p，p即为pass数目)点fft计算时延与pass数目直接相关。如果使用基2运算不考虑控制开销，纯粹的计算时延为td=p×n×t×fin/fa。显然在fa>p× fin时，在n点内可完成fft运算。否则不能完成，即不能实现流型的变换。这在n很大且输入数据速率较高时以fpga实现几乎是不可能的，而且内部计算时钟过高容易导致电路的工作不稳定。设基2时的最小可流型工作运算频率为fa0，则使用基4实现流型的变换，计算时钟fa= fa0就可以。而使用基8时计算时钟fa= fa0便可完成，基16时为fa0的1/4。上面所讨论的是纯基运算，当n不为4的幂次方时(如n=2048=16×16×8，运算模块为基16/8复用模块)，而又希望使用较低倍的时钟完成运算时，图2中的运算模块必然包括基4/2复用模块(即基16/8复用模块)，这也就是前面提到复用模块的主要用意。由上面的分析可以得出结论，如果计算使用的基越大，完成速度越快。

但是，使用基16/8模块所使用的逻辑资源要比基4/2模块多将近一倍，这是因为基16/8复用模块是以基4模块和基4/2复用模块构建而成。当然，可以直接实现基16/8复用模块，但用fpga很难解决复杂度和成本问题。另外，如果流型运算间隔比n点数据长度长一倍以上，可以考虑在较低的计算时钟下使用基2运算模块实现流型fft。

运算结果的精度直接与计算过程中数据和因子位数(浮点算法)相关，如果中间计算的位数、存储数据位数和rom表中的位数越大，输出精度就越大。当然，位数增大后逻辑运算资源和存储资源都会直线上升。

浮点、块浮点和定点fft

根据运算过程中对数据位数取位和表示形式的不同，可以将fft分为浮点fft、块浮点fft和定点fft。它们在实现时对于系统资源的要求是不同的，而且有着不同的适用范围。

浮点fft是基于数据表示为浮点的基础之上的，即数据是由一纯小数和一因子组成，输入要转成纯小数和因子的浮点表示形式，所有计算过程中保存应得计算结果大小，而输出要变成所需大小的定点表示形式。只要因子位数足够大，浮点fft计算是不会溢出的。而定点则是所有计算过程中都是定点运算，如果各个pass的截位规则不适当，很容易出现溢出，必须要有溢出控制。块浮点是介于它们之间的一种运算机制，它是根据本pass的输入数据的大小，在计算之前进行控制(数据上移一比特或下移一比特或乘以一特定因子)，可以保证不溢出，但一般也需要溢出控制。

浮点运算没有溢出，信号平均信噪比高，但由于因子的运算必然导致电路复杂，实现困难。定点运算实现简单，难以保证不溢出，需要统计得出合适的截位规则，否则溢出严重导致输出结果错误。块浮点由于每个pass(包括最后输出前)结束后有一统计控制过程，延时较大，但是可以保证不溢出而且电路又相对浮点来说简单得多。

应根据具体应用的具体要求，选择合适的fft。如果要求精度，并且要解决频域很高的单频干扰，就必须使用浮点的fft，使用数据位数很大的定点和块浮点也能解决这个问题，但位数的确定十分困难。如果不要求高精度，逻辑资源和rom比较紧张，可考虑定点运算。如果输入在频域集中于几个点上或者对精度要求一般，可以慢速处理，可以采用块浮点运算，就能够保证这几点的信噪比，而忽略其他点处的信噪比。

非常详细的参考网站

<http://www.reynwar.net/ben/docs/fft_dit/>