**FIR滤波器以及吉布斯效应**

最近要录一些环境声音数据做实验，录音笔上有一个选项——[**Low Cut Filter**](http://helpguide.sony.net/icd/sx3/v1/cs2/contents/TP0000020730.html)，即低频切除滤波器，就是一个高通滤波器，其作用是“减轻投影机声音或因风产生的啸叫声等低频噪音，以便更清晰地录制文件“，截止频率可调。   
于是，考虑自己设计一个滤波器以实现这一功能，打算使用FIR滤波器。

**滤波器设计指标**

关于滤波器的设计指标，[xiahouzuoxin的这篇文章](http://blog.csdn.net/xiahouzuoxin/article/details/21251561)介绍得很详细。不过里面有一小点讲错：衰减指标的分贝值应定义为10*logP* *out* *P* *in*   10logPoutPin 或者20*logV* *out* *V* *in*   20logVoutVin ，注意其中使用功率比和幅值比时常系数的差别。

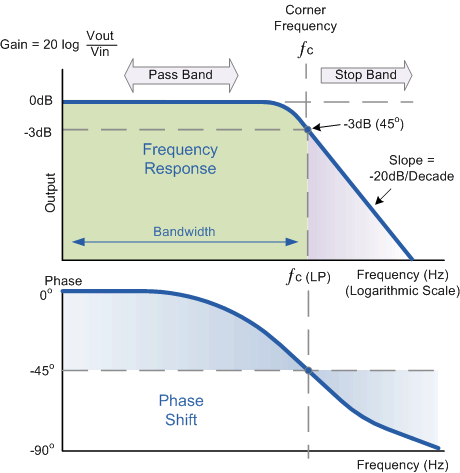
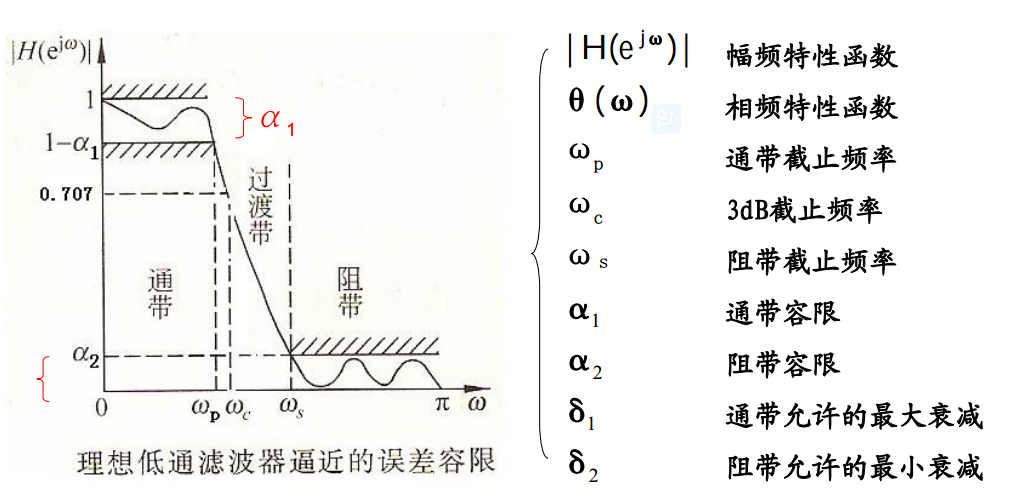
图1 低通滤波器   


图2 滤波器设计指标（对应低通滤波器）   


关于图1和图2的说明：

* 图1的幅频特性单位为dB，是按通带增益为1归一化后的结果。图1中的3dB衰减即对应图2中的0.707（由20*logV* −3*dB* *V* 0*dB*  =−3 20logV−3dBV0dB=−3 得*V* −3*dB* =0.707 V−3dB=0.707 ），即在此点处信号幅值衰减至原来的0.707，而功率衰减至原来的一半。工程上，通常以3dB截止频率来计算带宽。
* 几个特征频率
  + 通带截止频率*ω* *p*  ωp ：通带与过渡带边界点的频率
  + 阻带截止频率*ω* *s*  ωs ：阻带与过渡带边界点的频率
  + 3dB截止频率*ω* *c*  ωc ：信号功率衰减至一半的频率
* 增益与衰减
  + 通带增益：对于低通为*ω*=0 ω=0 时的增益，对于高通为*ω*→∞ ω→∞ 时的增益，对于带通为中心频率处的增益
  + 通带容限*α* 1  α1 ：通带内，1−*α* 1 ≤|*H*(*e* *jω* )|≤1+*α* 1 ,|*ω*|≤*ω* *p*  1−α1≤|H(ejω)|≤1+α1,|ω|≤ωp
  + 通带衰减*δ* 1  δ1 ：相对于通带增益，通带内波纹允许的最大衰减，其分贝值为*δ* 1 =20*logH*(*e* *j*0 )*H*(*e* *jω* *p*  ) =−20*log*(1−*α* 1 ) δ1=20logH(ej0)H(ejωp)=−20log(1−α1)
  + 阻带容限*α* 2  α2 ：阻带内，|*H*(*e* *jω* )|≤*α* 2 ,*ω* *s* ≤*ω*≤*π* |H(ejω)|≤α2,ωs≤ω≤π
  + 阻带衰减*δ* 2  δ2 ：相对于通带增益，阻带内波纹允许的最小衰减，其分贝值为*δ* 2 =20*logH*(*e* *j*0 )*H*(*e* *jω* *s*  ) =−20*log*(*α* 2 ) δ2=20logH(ej0)H(ejωs)=−20log(α2)
* 物理频率与角频率的关系：*ω*=2*πf* ω=2πf 。同时，对于采样频率为*f* *s*  fs 的信号，根据Nyquist采样定律，信号的最高频率为*f* *s* 2  fs2 ，所以频谱反映的是0~*f* *s* 2  fs2 的谱信息，对应角频率0~*π* π （离散时间实信号的频谱共轭对称且以2*π* 2π 为周期）。

**FIR滤波器的设计**

FIR滤波器，即有限长单位脉冲响应滤波器，具有两个突出的特点：**线性相位和稳定性**。

* **稳定性**：FIR滤波器的单位脉冲响应为有限长，其*z* z 变换收敛
* **线性相位**：如果FIR滤波器的单位脉冲响应是实数，且满足偶对称或奇对称的条件，则滤波器就具有严格的线性相位特性。
  + 偶对称：*h*[*n*]=*h*[*N*−1−*n*]→*H*(*e* *jω* )=*e* −*jω*(*N*−12 ) ∑ *N*−1 *n*=0 *h*[*n*]cos[*ω*(*n*−*N*−12 )]*ϕ*(*ω*)=*e* −*ωN*−12   h[n]=h[N−1−n]→H(ejω)=e−jω(N−12)∑n=0N−1h[n]cos⁡[ω(n−N−12)]ϕ(ω)=e−ωN−12
  + 奇对称：*h*[*n*]=−*h*[*N*−1−*n*]→*H*(*e* *jω* )=*e* −*j*[*ω*(*N*−12 )+*π*2 ] ∑ *N*−1 *n*=0 *h*[*n*]sin{*ω*(*n*−*N*−12 )}*ϕ*(*ω*)=*e* −*ωN*−12 −*π*2   h[n]=−h[N−1−n]→H(ejω)=e−j[ω(N−12)+π2]∑n=0N−1h[n]sin⁡{ω(n−N−12)}ϕ(ω)=e−ωN−12−π2

**理解线性相位**

线性相位特性是指系统的相频特性是一条直线，其意义在于**系统对所有频率的信号均产生相同的时延，不会发生相位失真**。

线性系统引起的信号失真由两方面因素造成——   
1. **幅度失真**：系统对信号中各频率分量的幅度产生不同程度的衰减   
2. **相位失真**：系统对信号中各频率分量产生的相移不与频率成正比，使响应的各频率分量在时间轴上产生不同的时延，其相对位置产生变化（频域相移对应于时域时移）

设滤波器频响为

*H*(*ω*)=*A*(*ω*)*e* *jϕ*(*ω*)  H(ω)=A(ω)ejϕ(ω)

对于线性相位滤波器，有*ϕ*(*ω*)=*αω*+*β* ϕ(ω)=αω+β ，*α*、*β* α、β 为常数。   
考虑频率为*ω* 0  ω0 的复正弦信号，

*x*(*t*)=*e* *jω* 0 *t* →*X*(*ω*)=2*πδ*(*ω*−*ω* 0 ) x(t)=ejω0t→X(ω)=2πδ(ω−ω0)

通过系统*H*(*ω*) H(ω) 后，有

*Y*(*ω*)=*X*(*ω*)*H*(*ω*)=2*πδ*(*ω*−*ω* 0 )*A*(*ω*)*e* *jϕ*(*ω*) =2*πδ*(*ω*−*ω* 0 )*A*(*ω* 0 )*e* *jϕ*(*ω* 0 )  Y(ω)=X(ω)H(ω)=2πδ(ω−ω0)A(ω)ejϕ(ω)=2πδ(ω−ω0)A(ω0)ejϕ(ω0)

→*y*(*t*)=*A*(*ω* 0 )*e* *jω* 0 (*t*+*ϕ*(*ω* 0 )*ω* 0  )  →y(t)=A(ω0)ejω0(t+ϕ(ω0)ω0)

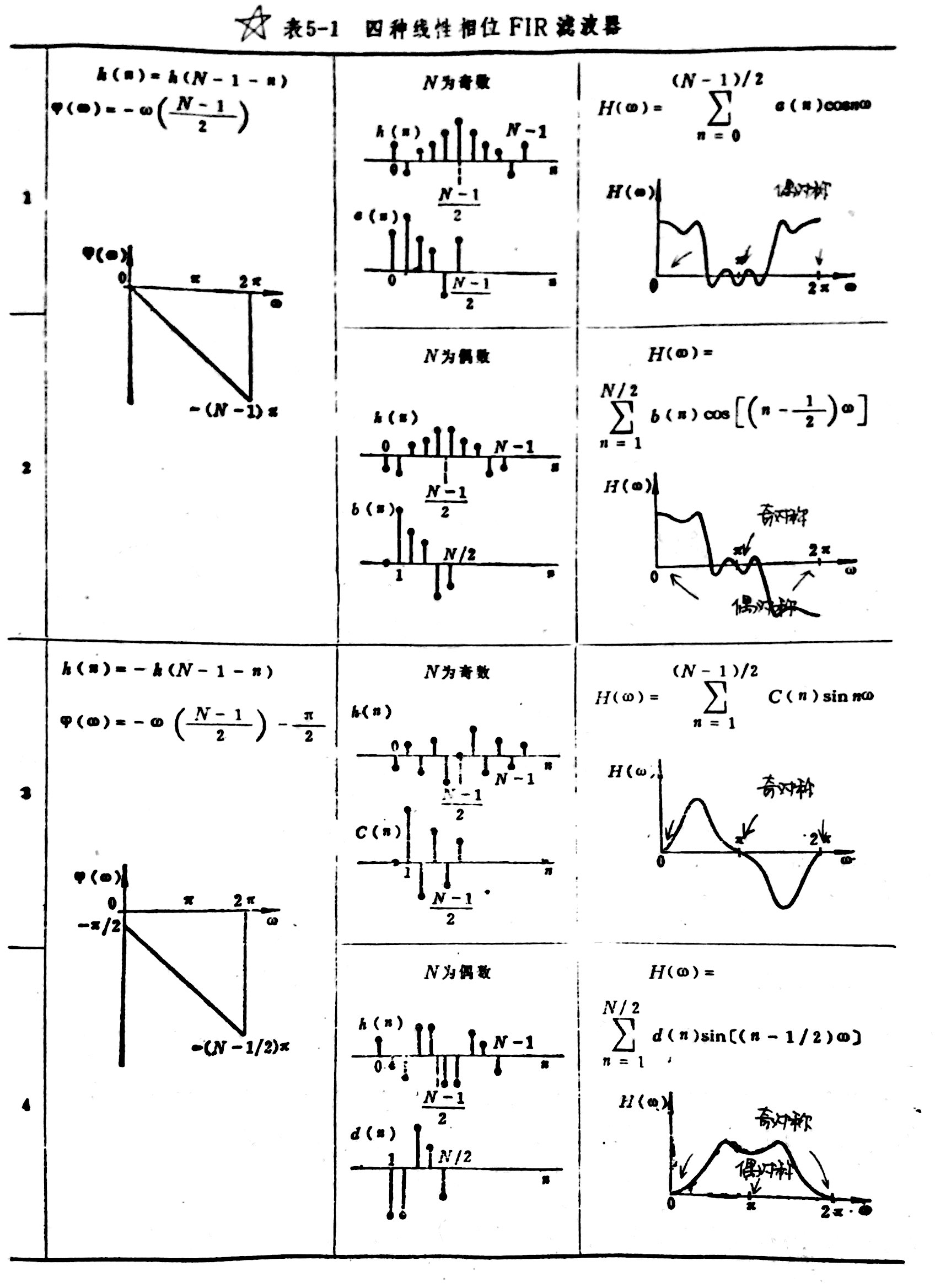
对于线性相位系统，有

*y* *lp* (*t*)=*A*(*ω* 0 )*e* *j*(*ω* 0 (*t*+*α*)+*β*)  ylp(t)=A(ω0)ej(ω0(t+α)+β)

即**信号***x*(*t*) x(t) **通过系统后产生***α* α **的时延，而且当***β*≠0 β≠0 **时，将产生相移***β* β。而对于非线性相位系统，这个时延与*ω* 0  ω0 有关。进一步，任意信号均可分解成一系列复正弦信号的加权和。对于线性相位系统，不同频率的复正弦信号通过系统后均产生相同的时延，而非线性相位系统可能会对信号造成失真或变形。

**四种线性相位FIR滤波器**

根据*N* N **的奇偶、***h*[*n*] h[n] **奇对称还是偶对称**的情况，可分别对应四种滤波器。为设计出线性相位的FIR滤波器，需根据滤波器的设计要求（低通、高通、带通、带阻）确定对应的*h*[*n*] h[n] 应满足的条件。

表1 四种线性相位FIR滤波器[3]   


上表中最右列为滤波器的幅频特性*H*(*ω*) H(ω) ，即*H*(*e* *jω* )=*H*(*ω*)*e* *jϕ*(*ω*)  H(ejω)=H(ω)ejϕ(ω) 。这里要注意虽然*H* ( *e* *jω* ) H(ejω) 是以2*π* 2π 为周期的，但不等价于*H*(*ω*) H(ω) 也是以2*π* 2π 为周期。以*ϕ*(*ω*)=−*αω*−*β* ϕ(ω)=−αω−β 为例，

*H*(*e* *j*(*ω*+2*π*) )=*H*(*ω*+2*π*)*e* −*j*(*α*(*ω*+2*π*)+*β*) =*H*(*ω*+2*π*)*e* −*j*(*αω*+*β*) *e* −*j*2*πα*  H(ej(ω+2π))=H(ω+2π)e−j(α(ω+2π)+β)=H(ω+2π)e−j(αω+β)e−j2πα

因此有

*H*(*ω*+2*π*)*e* −*j*2*πα* =*H*(*ω*) H(ω+2π)e−j2πα=H(ω)

当*α*∈Z α∈Z 时，*H*(*ω*+2*π*)=*H*(*ω*) H(ω+2π)=H(ω) ，对应表中第1、3种情况；   
当*α*=*m*+12 ,*m*∈Z α=m+12,m∈Z 时，*H*(*ω*+2*π*)=−*H*(*ω*) H(ω+2π)=−H(ω) ，对应表中第2、4种情况。

**窗口法设计FIR滤波器**

窗口法设计FIR滤波器的基本思路是**依据希望得到的理想滤波器的频响***H* *d* (*e* *jω* ) Hd(ejω) **，计算其单位脉冲响应***h* *d* [*n*] hd[n] **，从中直接截取一段***h*[*n*] h[n] **以逼近理想值**。同时，为保证线性相位特性，*h*[*n*] h[n] 需满足对应的偶对称或奇对称特性。   
**例1：截止频率为***ω* *c*  ωc **的理想低通滤波器**   
根据表1，所设计的滤波器为I、II类。设时延为*α* α ，理想滤波器频响为

*H* *d* (*e* *jω* )={ *e* −*jωα* , |*ω*|≤*ω* *c*  0,  *ω* *c* <|*ω*|<*π*  Hd(ejω)={0,  ωc<|ω|<πe−jωα, |ω|≤ωc

计算其单位脉冲响应，为

*h* *d* [*n*]=12*π* ∫ *ω* *c*  −*ω* *c*  *e* −*jωα* *e* *jωn* *dω*=*e* *jω*(*n*−*α*) 2*πj*(*n*−*α*) | *ω* *c*  −*ω* *c*  =sin[*ω* *c* (*n*−*α*)]*π*(*n*−*α*)  hd[n]=12π∫−ωcωce−jωαejωndω=ejω(n−α)2πj(n−α)|−ωcωc=sin⁡[ωc(n−α)]π(n−α)

这是一个以*α* α 为中心的偶对称的无限长非因果序列。所截取的*h*[*n*] h[n] 需满足偶对称，*N* N 可奇可偶，因此

*h*[*n*]={ *h* *d* [*n*], 0≤*n*≤*N*−1 0,   *else*  h[n]={0,   elsehd[n], 0≤n≤N−1

*α*=*N*−12  α=N−12

**例2：截止频率为***ω* *c*  ωc **的理想高通滤波器**   
根据表1，所设计的滤波器为IV类。设时延为*α* α ，理想滤波器频响为

*H* *d* (*e* *jω* )={ *e* −*j*(*ωα*+*π*2 ) , *ω* *c* ≤*ω*≤2*π*−*ω* *c*  0,     0≤*ω*<*ω* *c*  *and* 2*π*−*ω* *c* <*ω*≤2*π*  Hd(ejω)={0,     0≤ω<ωc and 2π−ωc<ω≤2πe−j(ωα+π2), ωc≤ω≤2π−ωc

其单位脉冲响应为

*h* *d* [*n*]=12*π* ∫ 2*π*−*ω* *c*  *ω* *c*  *e* −*j*(*ωα*+*π*2 ) *e* *jωn* *dω*=*e* −*jπ*2  *e* *jω*(*n*−*α*) 2*πj*(*n*−*α*) | 2*π*−*ω* *c*  *ω* *c*  =*e* *jω* *c* (*n*−*α*) −*e* *j*2*π*(*n*−*α*) *e* −*jω* *c* (*n*−*α*) 2*π*(*n*−*α*) − → − − − − − − −  *α*=*m*+12 ,*m*∈Z  =cos[*ω* *c* (*n*−*α*)]*π*(*n*−*α*)  hd[n]=12π∫ωc2π−ωce−j(ωα+π2)ejωndω=e−jπ2ejω(n−α)2πj(n−α)|ωc2π−ωc=ejωc(n−α)−ej2π(n−α)e−jωc(n−α)2π(n−α)→α=m+12,m∈Z=cos⁡[ωc(n−α)]π(n−α)

这是一个以*α* α 为中心的奇对称的无限长非因果序列。截取的*h*[*n*] h[n] 需满足奇对称，且*N* N 为偶数，因此

*h*[*n*]={ *h* *d* [*n*], 0≤*n*≤*N*−1 0,   *else*  h[n]={0,   elsehd[n], 0≤n≤N−1

*α*=*N*−12  α=N−12

在这里，*h*[*n*] h[n] 可看做是*h* *d* [*n*] hd[n] 与一个窗口函数的乘积，上例中使用的是矩形窗函数*R* *N* [*n*] RN[n] 。对于一般的*N* N 点窗函数*w*[*n*] w[n] ，有

*h*[*n*]=*h* *d* [*n*]*w*[*n*] h[n]=hd[n]w[n]

这样，只要给定截止频率，选择一个窗函数和点数，我要设计的LCF就可以搞定啦。

**窗函数的影响**

这种对理想单位脉冲响应加窗的处理对频响将产生什么影响？以及逼近质量如何？

*h*[*n*]=*h* *d* [*n*]*w*[*n*]→*H*(*e* *jω* )=12*π* *H* *d* (*e* *jω* )∗*W*(*e* *jω* ) h[n]=hd[n]w[n]→H(ejω)=12πHd(ejω)∗W(ejω)

以**矩形窗**为例，其频谱为

*R* *N* [*n*]=1, *n*=0,⋯,*N*−1→*W* *R* (*e* *jω* )=*e* −*jωN*−12  sin(*ωN*/2)sin(*ω*/2)  RN[n]=1, n=0,⋯,N−1→WR(ejω)=e−jωN−12sin⁡(ωN/2)sin⁡(ω/2)

其中线性相位部分*e* −*jωN*−12   e−jωN−12 表示窗函数时延一半长度。不考虑相位特性，只关注其幅度谱，为

*W* *R* (*ω*)=sin(*ωN*/2)sin(*ω*/2)  WR(ω)=sin⁡(ωN/2)sin⁡(ω/2)

对照表1可知，*R* *N* [*n*] RN[n] 偶对称，其幅度谱的特性根据*N* N 的奇偶符合第I、II类的情况。其他特性：

* 在*ω*=±2*πN*  ω=±2πN 内有一主瓣，然后向两侧呈衰减震荡展开，主瓣宽度4*πN*  4πN ，主瓣峰值*W* *R* (0)=*N* WR(0)=N
* 在0∼*π* 0∼π 内的零点为*ω*=2*kπN* ,*k*=1,2,⋯,⌊*N*2 ⌋ ω=2kπN,k=1,2,⋯,⌊N2⌋ ，对应旁瓣个数约为⌊*N*2 ⌋−1 ⌊N2⌋−1 ，且旁瓣谱峰是逐渐衰减的
* *N* N 越大，主瓣越窄，旁瓣越多，震荡越密集

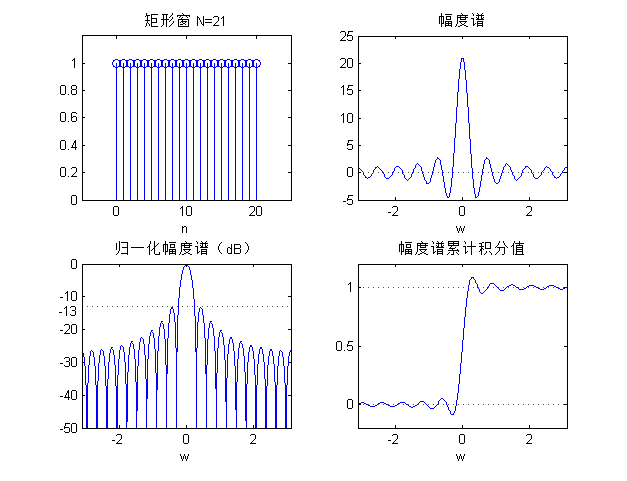
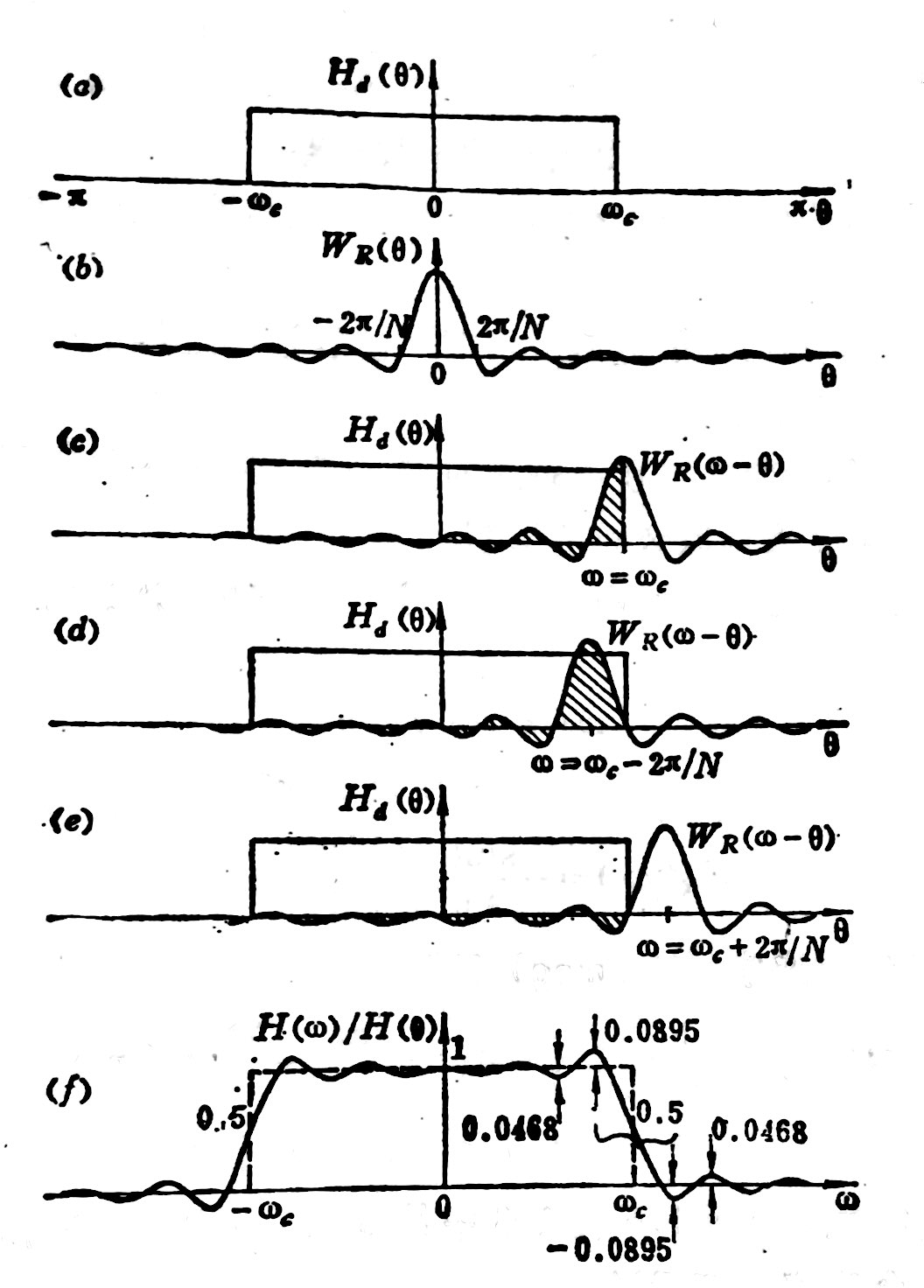
图3 矩形窗幅度谱   


图4 矩形窗函数效果   


分析使用矩形窗时的FIR滤波器的幅度谱，

*H*(*ω*)=12*π* ∫ *π* −*π* *H* *d* (*θ*)*W* *R* (*ω*−*θ*)*dθ*=12*π* ∫ *ω* *c*  −*ω* *c*  *W* *R* (*ω*−*θ*)*dθ*=12*π* ∫ *ω*+*ω* *c*  *ω*−*ω* *c*  *W* *R* (*θ*)*dθ* H(ω)=12π∫−ππHd(θ)WR(ω−θ)dθ=12π∫−ωcωcWR(ω−θ)dθ=12π∫ω−ωcω+ωcWR(θ)dθ

* *ω*=0 ω=0 时，*H*(0) H(0) 等于*W* *R* (*θ*) WR(θ) 在−*ω* *c*  −ωc 到*ω* *c*  ωc 上的积分，当*ω* *c* ≫2*πN*  ωc≫2πN 时（一般都满足），*H*(0) H(0) 近似于*W* *R* (*θ*) WR(θ) 在−*π* −π 到*π* π 上的全部积分面积，即

*H*(0)≈12*π* ∫ *π* −*π* *W* *R* (*θ*)*dθ*=*R* *N* [0]=1 H(0)≈12π∫−ππWR(θ)dθ=RN[0]=1

* *ω*=*ω* *c*  ω=ωc 时，*H*(0) H(0) 等于*W* *R* (*θ*) WR(θ) 在0到2*ω* *c*  2ωc 上的积分，近似于*W* *R* (*θ*) WR(θ) 的一半积分面积，即*H*(*ω* *c* )=12 *H*(0) H(ωc)=12H(0)
* *ω*=*ω* *c* −2*πN*  ω=ωc−2πN 时，*H*(0) H(0) 等于*W* *R* (*θ*) WR(θ) 在−2*πN*  −2πN 到2*ω* *c* −2*πN*  2ωc−2πN 上的积分，近似于*W* *R* (*θ*) WR(θ) 的一半积分面积再加上一半主瓣面积。由于此时主瓣全部计算在内，*H*(*ω*) H(ω) 达到最大值，出现正肩峰；相反的，*ω*=*ω* *c* +2*πN*  ω=ωc+2πN 时，主瓣全部未计算在内，此时*H*(*ω*) H(ω) 达到最小值，出现负肩峰

因此，可总结出*H*(*ω*) H(ω) **的特性**：

* **肩峰**：*H*(*ω*) H(ω) 在*ω* *c*  ωc 两旁距离2*πN*  2πN 处分别出现正负肩峰，肩峰值的大小与主瓣和旁瓣的相对大小有关，取决于窗函数的形状；肩峰值的大小直接影响滤波器通带的平稳和阻带的衰减（通带、阻带容限），应越小越好
* **过渡带**：两肩峰之间形成一个过渡带，其宽度为4*πN*  4πN ，等于主瓣宽度
* **通带和阻带内波动**：由于旁瓣的影响，当*ω* ω 由两肩峰处继续朝通带和阻带变化时，形成长长的余震

在选用窗函数设计FIR滤波器时，应注意：

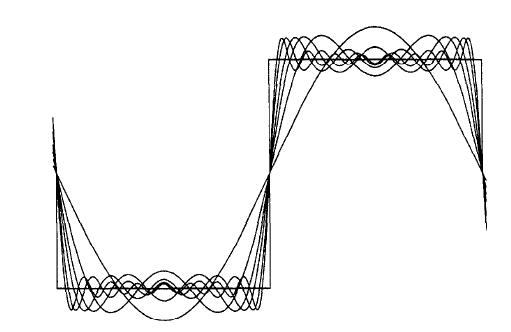
* 主瓣宽度尽量窄，以获得较陡的过渡带→ → 增加窗函数点数
* 尽量减小旁瓣，使能量集中在主瓣，以减小肩峰，提高阻带衰减→ → 寻找合适的窗函数

这里注意一个很奇特的特性：

* 矩形窗情况下，**最大肩峰值始终为0.0895**（此时阻带最小衰减为-21dB），与*N* N 无关——**吉布斯（Gibbs）效应**，实际上是由于时域截断造成的

**吉布斯（Gibbs）效应**

**吉布斯效应**：将**具有不连续点**的周期函数（如矩形脉冲)进行傅立叶级数展开后，**选取有限项**进行合成。当选取的项数越多，在所合成的波形中出现的峰起越靠近原信号的不连续点。当选取的项数很大时，该峰起值趋于一个常数，大约等于总跳变值的9%。这种现象称为吉布斯效应。

[图5 吉布斯效应](http://blog.csdn.net/deepdsp/article/details/7406251)   


吉布斯效应原始的说法是为了说明，当用傅里叶级数的有限项（有限个频率分量）去逼近信号时，在信号的间断点处会出现一个过渡带以及起伏和超量（肩峰），也就是说**频域的截断带来时域的弥散**。而且，当所取项数*N* N 为有限值时，随着*N* N 的增大，起伏不断向间断点处压缩，但起伏的峰值大小保持不变。同时，*N* N 增大还有一个好处是，起伏具有的总能量会变小——逼近误差越小。因此，在实际做这样的近似时，应该选择足够大的*N* N ，以保证这些起伏具有的总能量可以忽略。

在学傅里叶变换时有一个经典问题：**傅里叶变换的基函数——复正弦函数都是平滑的，用复正弦函数的加权和怎么能表示那些有间断点的时间波形如方波？**——这也是拉普拉斯最初在审查傅里叶的论文时的一个质疑点。回答是：[5-6]

* 一方面，可以说拉普拉斯是对的，用一系列正弦波的线性组合无法精确描述那些具有间断点的信号。根据吉布斯现象，当正弦波数量增多时，超量始终存在，且幅值保持稳定约为9%，只是超量的宽度变小并向间断点压缩。
* 另一方面，用一系列正弦波的线性组合可以无限逼近那些具有间断点的信号，以至**逼近误差的能量为0**。当正弦波数量无限多时，超量依然存在，但其宽度为0，因此在能量意义上为0。更精确地，间断点处的值收敛于信号在该跳跃点两侧值的均值。可以证明，傅里叶级数表示是收敛的，其逼近误差的能量为0。基于此，傅立叶是对的。

回到本文，在用窗函数法设计FIR滤波器时，实际遇到的是与上面相反的吉布斯现象——**时域的截断带来频域的弥散，即用窗函数去截断理想滤波器的单位脉冲响应时，对应的FIR滤波器的幅度谱在截止频率处会出现过渡带以及起伏和肩峰**。

一开始纠结了好半天这个0.0895或是这个9%是怎么算出来的，直接手算推导失效，最后只好借助于强大的MatLab了。这里可借助于矩形窗幅度谱*W* *R* (*ω*) WR(ω) 的特性来解释吉布斯效应。上面讲到，*N* N **的增大可以使主瓣变窄**→ → **过渡带变窄，同时旁瓣增多，震荡变密集**→ → **通带和阻带内震动加快，但并不能改变肩峰值和波动的相对大小（主瓣与旁瓣的相对比例）**。

首先从幅值上简单分析，主瓣峰值*W* *R* (0)=*N* WR(0)=N ，第一旁瓣峰值大约在3*πN*  3πN 处取得，有

|*W* *R* (3*πN* )|=|sin3*π*2 sin3*π*2*N*  |≈2*N*3*π* ≈0.212*N* |WR(3πN)|=|sin⁡3π2sin⁡3π2N|≈2N3π≈0.212N

关于*N* N 做归一化后，第一旁瓣峰值对应于13dB衰减处，如图3。也就是说主瓣与旁瓣的相对大小是固定的。

做出不同*N* N 值下矩形窗幅度谱的累计积分曲线，可知其肩峰值稳定在0.09附近，如图6、7。同时，计算波动所具有的总能量，这里计算图6中左半边波形所具有的总能量，在*N*=11、21、51、101 N=11、21、51、101 时分别为28.85、15.11、6.22、3.12，确实随着*N* N 的增大而减小。

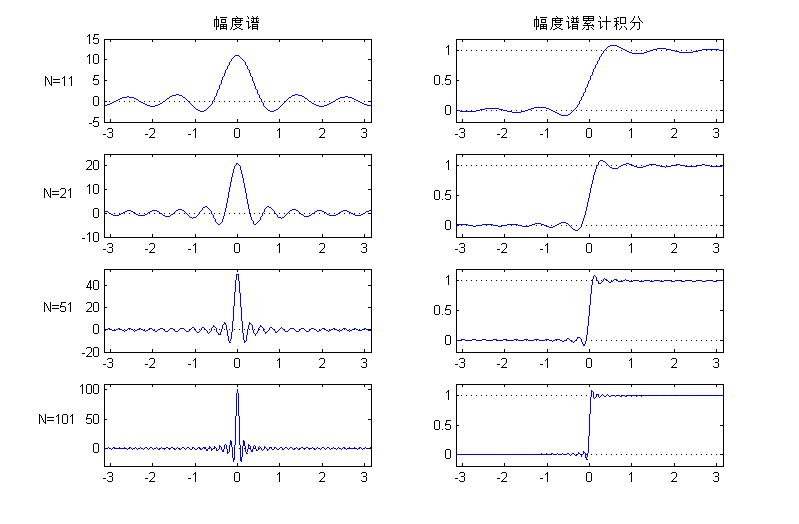
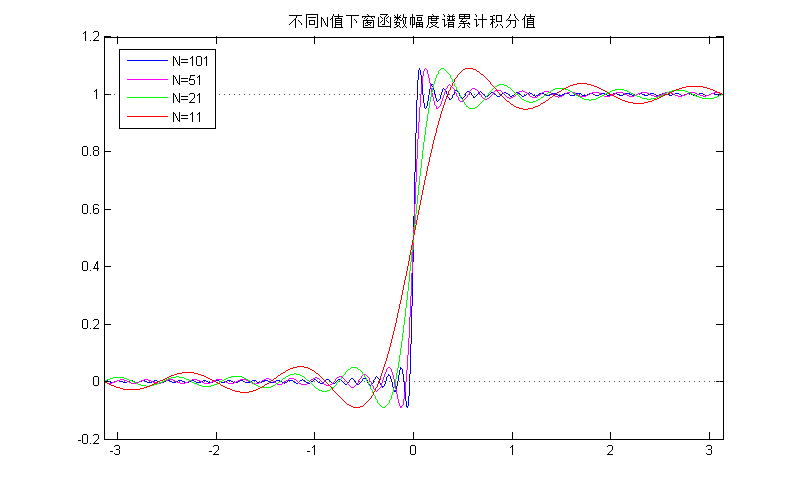
图6 不同长度矩形窗的幅度谱   


图7 不同长度矩形窗的幅度谱累计积分   


以上。下篇讨论窗函数。

**FIR滤波器（1）- 基础知识**

  FIR滤波器广泛应用于数字信号处理中，主要功能就是将不感兴趣的信号滤除，留下有用信号。FIR滤波器是全零点结构，系统永远稳定；并且具有线性相位的特征，在有效频率范围内所有信号相位上不失真。在无线通信收发机中的DDC/DUC模块，抽取和内插都需要加入滤波器以防止信号在频谱上混叠，最典型的是采用FIR滤波器实现半带滤波器。

         FIR滤波处理如下式所示，其中x(n)为输入信号，h(n)为FIR滤波系数，y(n)为经过滤波后的信号；N表示FIR滤波器的抽头数，滤波器阶数为N-1。

1.gif

         由上式可得到FIR滤波器在FPGA中的实现结构，如图1所示，主要由延迟单元Z-1、乘法器和累加器组成。此结构为直接型FIR滤波器结构，也称横向结构(transverse)。

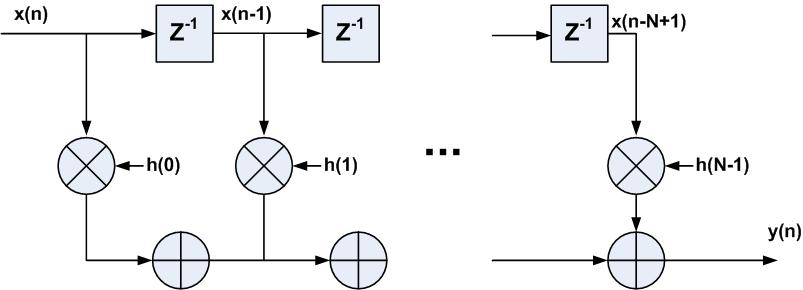


图1

         设计FIR滤波器的方法有多种，其中Matlab软件提供了很多关于滤波器设计的工具箱，FDATool就是一个很好的工具，如图2所示就是FDATool的界面，可以在Matlab的Command窗口中直接输入FDATool命令来调用。

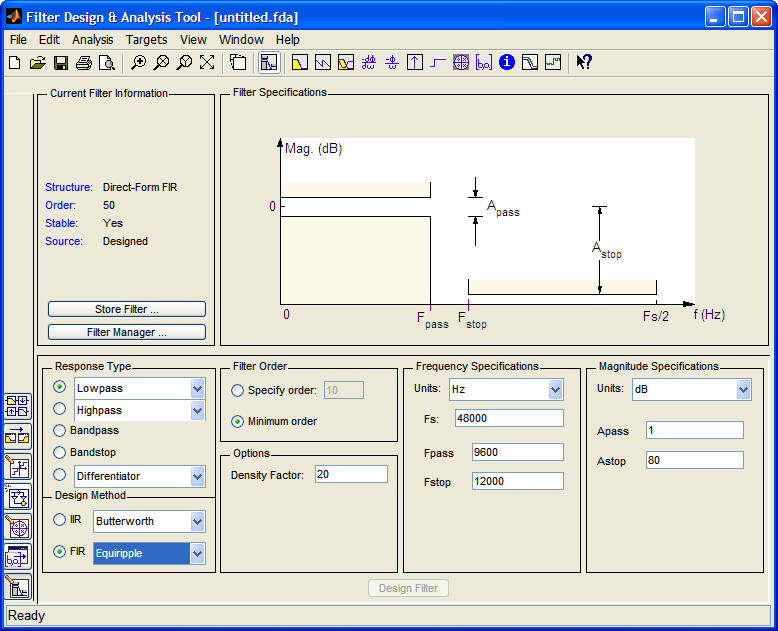


图2

滤波器的设计首先需要设置的参数：

         (1) Response Type：选择FIR滤波器的类型：低通、高通、带通和带阻等。如图3所示为Lowpass中的下拉选项，在DDC/DUC模块设计中，抽取和内插需要使用Halfband Lowpass类型，而channel filter需要使用Raised-cosine类型。

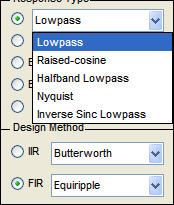


图3

         (2) Design Method：FIR滤波器设计方法有多种，如图4所示，最常用的是窗函数设计法(Window)、等波纹设计法(Equiripple)和最小二乘法(Least-Squares)等。其中窗函数设计法在学校课堂中是重点讲解的，提到FIR滤波器肯定会想到hamming、kaiser窗，但是实际应用中却很少使用，因为如果采用窗函数设计法，达到所期望的频率响应，与其它方法相比往往阶数会更多；而且窗函数设计法一般只参照通频带wp、抑制频带ws和理想增益来设计滤波器，但是实际应用中通频带和抑制带的波纹也是需要考虑的，那在这种情况下，采用等波纹设计法就非常适用了。

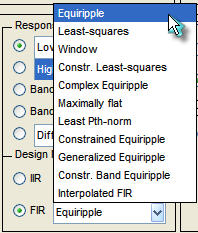


图4

         (3) Filter Order：设置滤波器的阶数，这个选项直接影响滤波器的性能，阶数越高，性能越好，但是相应在FPGA实现耗用的资源需要增多。在这个设置中提供2个选项：Specify order和Minimum order，Specify order是工程师自己确定滤波器的阶数，Minimum order是让工具自动确定达到期望的频率相应所需要的最小阶数，因此具体选择哪个选项得视实际情况而定了。

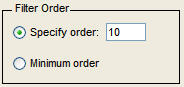


图5

         (4) Frequency Specification：设置频率响应的参数，包括采样频率Fs、通带频率Fpass和阻带频率Fstop。

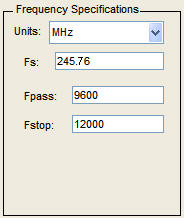


图6

         参数设置完成后，FDATool就会分析并且生成滤波系数，如图7所示，可以得到滤波器的频率相应曲线，并且可以通过File->Export导出滤波系数，如图8所示。

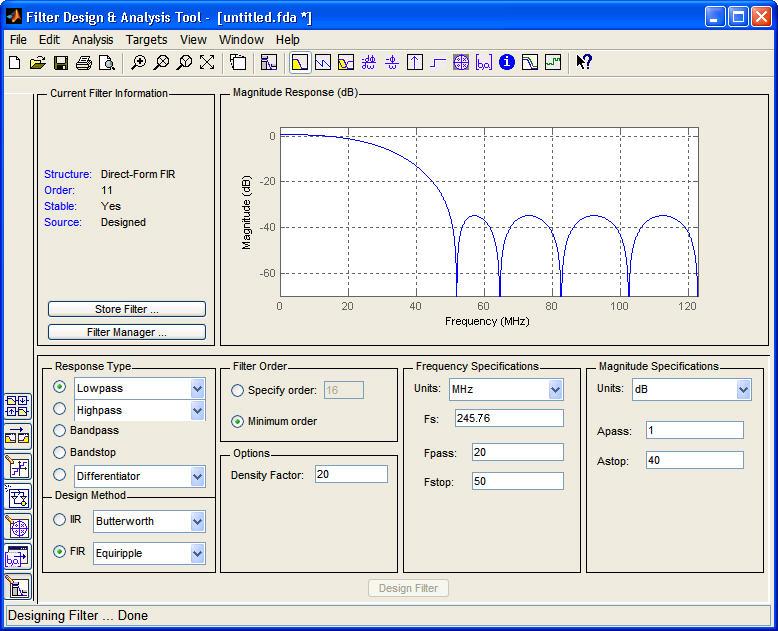


图7

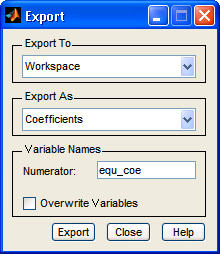


图8

         为了快速验证FIR滤波器的FPGA实现，使用Xilinx的System Generator工具，如图9所示为FIR滤波器的验证模型，其中通过Gateway In和Gateway out模块分隔matlab simulink模块和Xilinx FPGA模块，matlab simulink模块用于产生测试源，接收并显示滤波后波形。还有System Generator Token用于生成Xilinx FPGA模块的HDL代码。

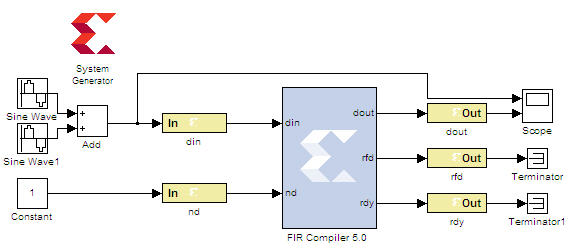


图9

         其中FIR Compiler 5.0模块的参数设置如图10所示，滤波系数直接调用FDATool生成的滤波系数equ\_coe，输出为全精度数据。

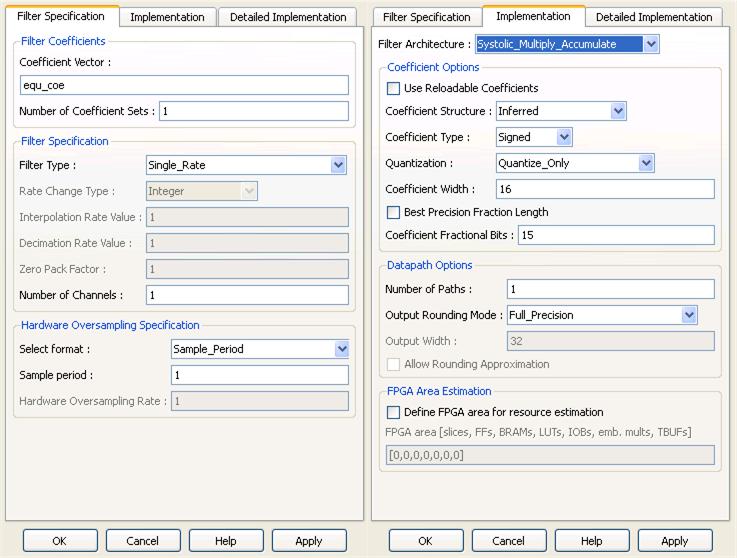


图10

         得到输出结果如图11所示，上边图为输入原波形，由两个频率分量的正弦波叠加而成，频率分别为2MHz和100MHz，经过FIR滤波之后，100MHz频率分量被滤除。

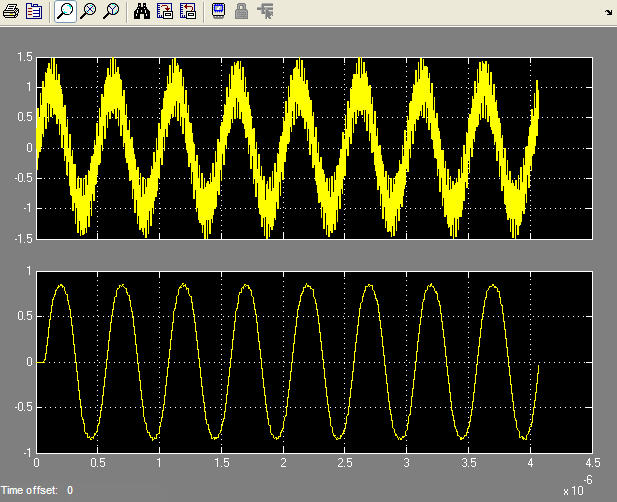


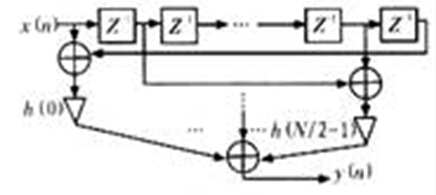
图11

参考：[http://bbs.ednchina.com/BLOG\_ARTICLE\_3003476.HTM#](http://bbs.ednchina.com/BLOG_ARTICLE_3003476.HTM)

**FIR滤波器的FPGA实现**

2015年09月06日 15:52:53

阅读数：7405

1.FIR滤波器简介   
FIR(Finite Impulse Response)滤波器：有限长单位冲激响应滤波器，又称为非递归型滤波器，是数字信号处理系统中最基本的元件，它可以在保证任意幅频特性的同时具有严格的线性相频特性，同时其单位抽样响应是有限长的，因而滤波器是稳定的系统。因此，FIR滤波器在通信、图像处理、模式识别等领域都有着广泛的应用。   
2.并行FIR滤波器   
根据传递函数H(Z)和FIR滤波器系数的对称性，可得FIR滤波器的一般实现结构，串型结构和并型结构。本文介绍并行结构的FIR，其结构框图如图1，并行滤波器的滤波速度快，一个时钟周期内完成一次滤波，但消耗大量的FPGA资源，如乘累加器，且器件的延迟较大，工作频率不宜太高。   
  
图 1   
3.FIR滤波器Matlab的实现   
设计一个15阶（长度为16）的低通线性相位FIR滤波器，采用布莱克曼窗函数设计，截止频率为500Hz,采样频率为2000Hz,输入数据12比特量化，系数12比特量化，输出数据位宽为29比特。   
文件：fir.m(产生滤波器的系数并量化)   
function hn = fir   
fs = 2000;%采样频率   
fd = 500;%截止频率   
Qb = 12;   
N = 16;   
% [n,wn,beta,ftype]=kaiserord(fd,mag,dev,Fs);   
% b=fir1(n,wn,ftype,kaiser(n+1,beta));   
w\_kais=blackman(N)’;   
b=fir1(N-1,fd\*2/fs,w\_kais);   
Q\_b = round(b/max(abs(b))\*(2^(Qb-1)-1));   
hn = Q\_b;   
文件：NoiseAndCarrier.m（产生200Hz和800Hz的正弦合成信号以及噪声，将数据量化后写入txt文件，作为信号源）   
f1 = 200;%信号1200Hz   
f2 = 800;%信号2800Hz   
fs = 2000;%采样频率   
Qb = 12;   
t = 0:1/fs:1;   
s1 = sin(2\*pi\*f1\*t);%信号1   
s2 = sin(2\*pi\*f2\*t);%信号2   
s = s1 + s2;%信号合成   
Noise = randn(1,length(t));%噪声

%归一化   
s = s/max(abs(s));   
Noise = Noise/max(abs(Noise));

%量化   
Q\_s = round(s\*(2^(Qb-1)-1));   
Q\_Noise = round(Noise\*(2^(Qb-1)-1));

%通过滤波器   
hn = fir   
filter\_s = filter(hn,1,s);   
filter\_noise = filter(hn,1,Noise);   
%信号的幅频特性   
m\_s = 20\*log(abs(fft(s,1024)))/log(10);m\_s = m\_s - max(m\_s)   
m\_noise = 20\*log(abs(fft(Noise,1024)))/log(10);m\_noise = m\_noise - max(m\_noise);   
%滤波后的幅频特性   
Fm\_s = 20\*log(abs(fft(filter\_s,1024)))/log(10);Fm\_s = Fm\_s - max(Fm\_s);   
Fm\_noise = 20\*log(abs(fft(filter\_noise,1024)))/log(10);Fm\_noise = Fm\_noise - max(Fm\_noise);   
%滤波器的幅频特性   
hn = 20\*log(abs(fft(hn,1024)))/log(10);hn = hn - max(hn);

%画图   
x\_f =1:fs/length(m\_s):fs/2;%横坐标，Hz   
mf\_s =m\_s(1:length(x\_f));   
mf\_noise = m\_noise(1:length(x\_f));   
Fmf\_s = Fm\_s(1:length(x\_f));   
Fmf\_noise =Fm\_noise(1:length(x\_f));   
hn\_f = hn(1:length(x\_f));   
subplot(211)   
plot(x\_f,mf\_s,’-‘,x\_f,Fmf\_s,’-.’,x\_f,hn\_f,’–’);   
xlabel(‘频率（Hz）’);ylabel(‘幅度（dB）’);   
legend(‘输入信号’,’输出信号’,’滤波器’);grid on;   
subplot(212)   
plot(x\_f,mf\_noise,’-‘,x\_f,Fmf\_noise,’-.’,x\_f,hn\_f,’–’);   
xlabel(‘频率（Hz）’);ylabel(‘幅度（dB）’);   
legend(‘输入信号’,’输出信号’,’滤波器’);grid on;

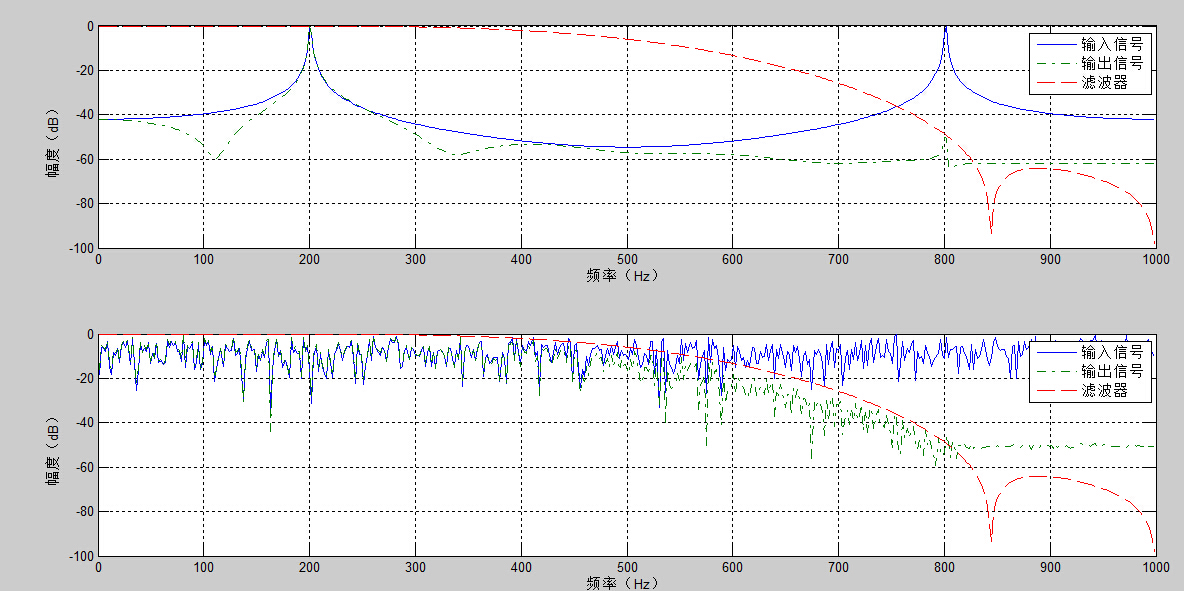
%将数据以十进制的格式写入txt文件   
fid =fopen(‘E:\Matlab\FPGA\fir\_filter\int\_s.txt’,’w’);   
fprintf(fid,’%8d\r\n’,Q\_s);   
fprintf(fid,’;’);   
fclose(fid);

fid =fopen(‘E:\Matlab\FPGA\fir\_filter\int\_noise.txt’,’w’);   
fprintf(fid,’%8d\r\n’,Q\_Noise);   
fprintf(fid,’;’);   
fclose(fid);

%将数据以二进制的格式写入txt文件   
fid =fopen(‘E:\Matlab\FPGA\fir\_filter\bin\_s.txt’,’w’);   
for i=1:length(Q\_s)   
s = dec2bin(Q\_s(i)+(Q\_s(i)<0)\*2^Qb,Qb);   
for j = 1:Qb   
if(s(j) == ‘1’ )   
tb = 1;   
else   
tb = 0;   
end   
fprintf(fid,’%d’,tb);   
end   
fprintf(fid,’\r\n’);   
end   
fprintf(fid,’;’);   
fclose(fid);

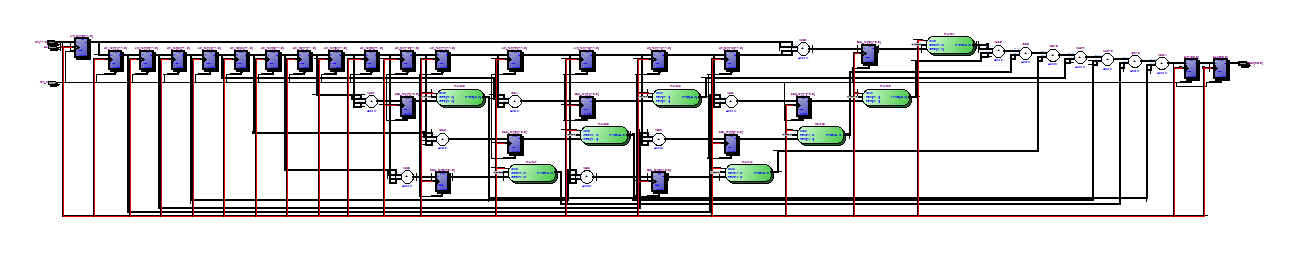
fid =fopen(‘E:\Matlab\FPGA\fir\_filter\bin\_noise.txt’,’w’);   
for i=1:length(Q\_Noise)   
s = dec2bin(Q\_Noise(i)+(Q\_Noise(i)<0)\*2^Qb,Qb);   
for j = 1:Qb   
if(s(j) == ‘1’ )   
tb = 1;   
else   
tb = 0;   
end   
fprintf(fid,’%d’,tb);   
end   
fprintf(fid,’\r\n’);   
end   
fprintf(fid,’;’);   
fclose(fid);

执行结果如图2：

图 2   


从图2（上）可以看出在200Hz和800Hz 有个尖峰，为正弦波信号，经过滤波器后800Hz的信号幅度被衰减了50dB,达到了滤波效果。

4.FPGA的实现

按照图1直接进行设计，设计后的RTL视图如图3，和图1差不多。   
  
图 3   
文件：FirParallel.v   
/***\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\****   
模块名：FirParallel   
功能：15阶低通线性相位FIR滤波器，采用布莱克曼窗函数设计，截止频率为500Hz,采样频率为2000Hz，系数量化为12bit   
作者：冬瓜   
时间：2015.9.6   
Email:lidong10280528@163.com   
***\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\****/   
module FirParallel   
(   
clk,rst\_n,   
xin,yout   
);   
input clk,rst\_n;//时钟和复位信号，低电平复位   
input signed [ 11:0 ]xin;//输入数据   
output signed [ 28:0 ]yout;//滤波后的数据

//将数据存入移位寄存器中   
reg signed [11:0]xin\_reg[15:0];   
reg [ 3:0 ]i,j;   
always @( posedge clk or negedge rst\_n )   
if( !rst\_n )   
begin   
for( i=0;i<15;i=i+1 )   
xin\_reg[ i+1 ] <= ‘d0;   
end   
else   
begin   
for( j=0;j<15;j=j+1 )   
xin\_reg[ j+1 ] <= xin\_reg[ j ];

xin\_reg[ 0 ] <= xin;   
end

//将对称系数的输入数据相加   
reg signed [ 12:0 ]add\_reg[7:0];   
always @( posedge clk or negedge rst\_n )   
if( !rst\_n )   
begin   
for(i=0;i<8;i=i+1)   
add\_reg[ i ] <= ‘d0;   
end   
else   
begin   
for( i=0;i<8;i=i+1 )   
add\_reg[ i ] = { xin\_reg[ i ][ 11 ],xin\_reg[ i ] } + { xin\_reg[ 15-i ][ 11 ],xin\_reg[ 15-i ] };   
end

wire signed [ 11:0 ]coe[ 7:0 ];//滤波器系数   
wire signed [ 24:0 ]mout[ 7:0 ];//乘法器输出   
assign coe[ 0 ] = 12’h0;   
assign coe[ 1 ] = 12’hffd;   
assign coe[ 2 ] = 12’h00f;   
assign coe[ 3 ] = 12’h02e;   
assign coe[ 4 ] = 12’hf8b;   
assign coe[ 5 ] = 12’hef9;   
assign coe[ 6 ] = 12’h24e;   
assign coe[ 7 ] = 12’h7ff;

//实例化乘法器   
//乘法器核设置如下：   
//dataa:12bit   
//datab:13bit   
//有符号，流水线1级   
mult u1 (   
.clock ( clk ),   
.dataa ( coe[ 0 ] ),   
.datab ( add\_reg[ 0 ] ),   
.result ( mout[ 0 ] )   
);   
mult u2 (   
.clock ( clk ),   
.dataa ( coe[ 1 ] ),   
.datab ( add\_reg[ 1 ] ),   
.result ( mout[ 1 ] )   
);   
mult u3 (   
.clock ( clk ),   
.dataa ( coe[ 2 ] ),   
.datab ( add\_reg[ 2 ] ),   
.result ( mout[ 2 ] )   
);   
mult u4 (   
.clock ( clk ),   
.dataa ( coe[ 3 ] ),   
.datab ( add\_reg[ 3 ] ),   
.result ( mout[ 3 ] )   
);   
mult u5 (   
.clock ( clk ),   
.dataa ( coe[ 4 ] ),   
.datab ( add\_reg[ 4 ] ),   
.result ( mout[ 4 ] )   
);   
mult u6 (   
.clock ( clk ),   
.dataa ( coe[ 5 ] ),   
.datab ( add\_reg[ 5 ] ),   
.result ( mout[ 5 ] )   
);   
mult u7 (   
.clock ( clk ),   
.dataa ( coe[ 6 ] ),   
.datab ( add\_reg[ 6 ] ),   
.result ( mout[ 6 ] )   
);   
mult u8 (   
.clock ( clk ),   
.dataa ( coe[ 7 ] ),   
.datab ( add\_reg[ 7 ] ),   
.result ( mout[ 7 ] )   
);

//对乘法器输出的结果进行累加，并输出滤波后的数据   
reg signed[ 28:0 ]sum;   
reg signed[ 28:0 ]dout;   
reg [ 3:0 ]k;   
always @( posedge clk or negedge rst\_n )   
if( !rst\_n )   
begin   
sum <= ‘d0;   
dout <= ‘d0;   
end   
else   
begin   
dout <= sum;   
sum = ‘d0;   
for( k=0;k<8;k=k+1 )   
sum = sum + mout[ k ];   
end   
assign yout = dout;   
endmodule

激励文件：FirParalle.vt   
`timescale 1 ps/ 1 ps   
module FirParallel\_vlg\_tst();   
// constants   
// general purpose registers

reg clk;   
reg rst\_n;   
reg [11:0] xin;   
// wires   
wire [28:0] yout;

// assign statements (if any)   
FirParallel i1 (   
// port map - connection between master ports and signals/registers   
.clk(clk),   
.rst\_n(rst\_n),   
.xin(xin),   
.yout(yout)   
);   
parameter clk\_period =20;   
parameter half\_clk\_period = clk\_period/2;   
parameter data\_period = clk\_period;   
parameter half\_data\_period = data\_period/2;   
parameter data\_num = 1000;   
parameter sim\_time = data\_num\*data\_period;

//产生时钟和复位信号   
initial   
begin   
clk = 1;   
rst\_n = 0;

**400;**

rst\_n = 1;

**sim\_time $finish;**

end   
always #half\_clk\_period clk = ~clk;

//读取txt文件   
reg [ 11:0 ]stimulus[ 1:data\_num ];   
integer i;   
initial   
begin   
$readmemb(“bin\_s.txt”,stimulus);   
i = 0;   
repeat(data\_num)   
begin   
i = i + 1;   
xin = stimulus[ i ];

**data\_period;**

end   
end

//将滤波后的数据写入txt文件   
wire write\_clk = clk & rst\_n;//写时钟，复位不写数据   
integer fid;   
initial   
begin   
fid = *fopen*(“*filter* *s* ”,”*w*”);*if*(!*fid*)*begin* fopen(“filters”,”w”);if(!fid)begin display(“cannot open the file!”);   
*finish*;*endendwiresigned*[28:0]*s* *y* *out*=*yout*;//将无符号转换为有符号数*always*@(*posedgewrite* *c* *lk*) finish;endendwiresigned[28:0]syout=yout;//将无符号转换为有符号数always@(posedgewriteclk) fdisplay(fid,”%d”,s\_yout);   
endmodule

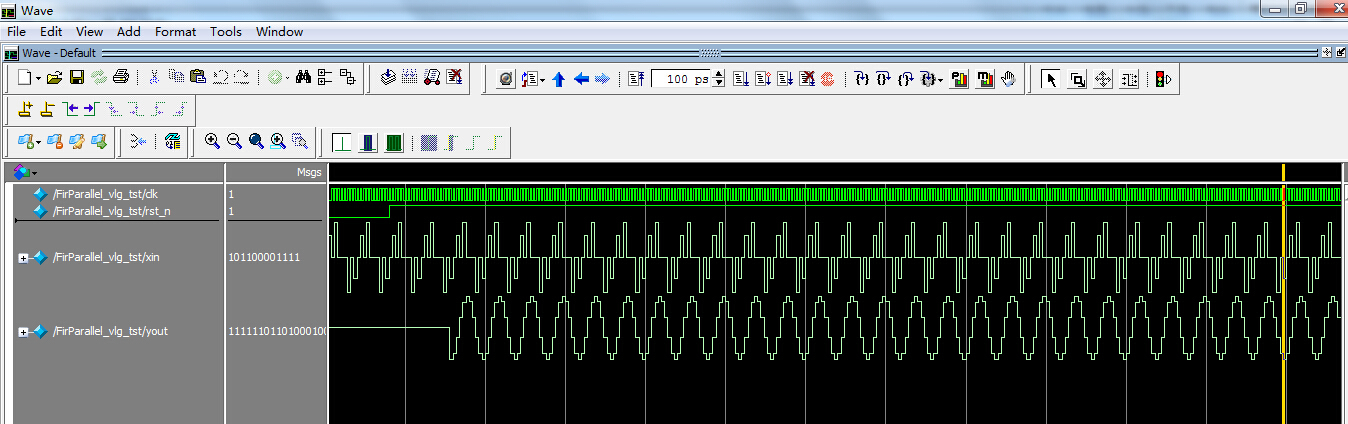
仿真结果如图4   


图 4

* 1
* 2

5.Matlab对结果分析   
文件：FIRAnalysis.m   
fs = 2000;

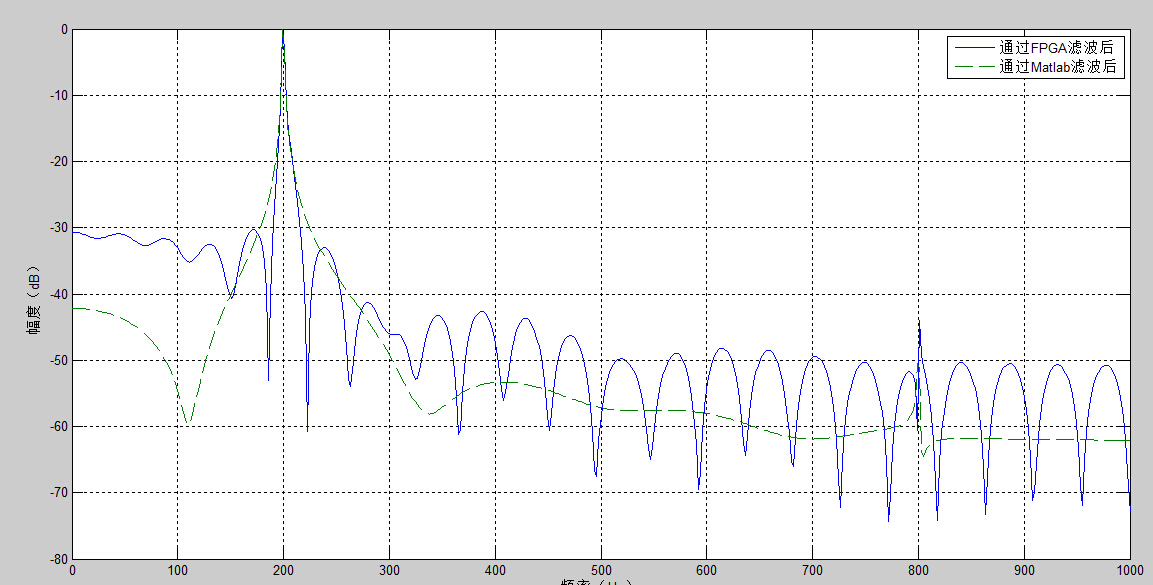
fid = fopen(‘E:\FPGA\_Project\Modelsim\FirParallel\simulation\modelsim\filter\_s.txt’,’r’);%读取通过Verilog编写的FIR滤波器后的数据   
[FPGA\_s\_out N\_FPGA\_s]=fscanf(fid,’%lg’,inf);   
fclose(fid);   
fid = fopen(‘E:\FPGA\_Project\Modelsim\FirParallel\simulation\modelsim\int\_filter\_s.txt’,’r’);%读取通过Matlba编写的FIR滤波器后的数据   
[Matlab\_s\_out N\_Matlab\_s]=fscanf(fid,’%lg’,inf);   
fclose(fid);

%归一化处理   
FPGA\_s\_out = FPGA\_s\_out/max(abs(FPGA\_s\_out));   
Matlab\_s\_out = Matlab\_s\_out/max(abs(Matlab\_s\_out));

FPGA\_s = 20\*log(abs(fft(FPGA\_s\_out,1024)))/log(10);   
FPGA\_s = FPGA\_s - max( FPGA\_s );

Matlab\_s = 20\*log(abs(fft(Matlab\_s\_out,1024)))/log(10);   
Matlab\_s = Matlab\_s - max(Matlab\_s);

x\_f = 0:fs/length(FPGA\_s):fs/2;   
FPGA\_s =FPGA\_s(1:length(x\_f));   
Matlab\_s =Matlab\_s(1:length(x\_f));

plot(x\_f,FPGA\_s,’-‘,x\_f,Matlab\_s,’–’);   
xlabel(‘频率（Hz）’);ylabel(‘幅度（dB）’);   
legend(‘通过FPGA滤波后’,’通过Matlab滤波后’);   
grid on;   
  
图 5   
从图5看出，对比s(t)信号（200Hz和800Hz的正弦合成信号）通过FPGA的滤波以及通过Matlab的滤波，两者在200Hz和800Hz的幅度一样，虽然在其他频率点有误差，但在误差范围内。

版权声明：本文为博主原创文章，未经博主允许不得转载。 https://blog.csdn.net/u013056038/article/details/48245293

文章标签： [FPGA](http://so.csdn.net/so/search/s.do?q=FPGA&t=blog) [Verilog](http://so.csdn.net/so/search/s.do?q=Verilog&t=blog) [FIR](http://so.csdn.net/so/search/s.do?q=FIR&t=blog)

个人分类： [FPGA](https://blog.csdn.net/u013056038/article/category/5662175) [Verilog](https://blog.csdn.net/u013056038/article/category/5685303)

**FIR滤波器相关解释**

2017年12月13日 08:50:23

阅读数：706



LTI（Linear Time-Invariant）

线性时不变：

线性时不变系统是根据系统输入和输出是否具有线性关系来定义的。满足叠加原理的系统具有线性特性。线性满足y=kx函数。

根据系统的输入和输出关系是否具有线性来定义 满足叠加原理的系统具有线性特性。即若对两个激励x1(n)和x2(n)，有T[ax1(n)+bx2(n)]=aT[x1(n)]+bT[x2(n)]，式中a、b为任意常数。

时不变系统

时不变系统：就是系统的参数不随时间而变化，即不管输入信号作用的时间先后，输出信号响应的形状均相同，仅是输入信号出现的时间不同。用数学表示为T[x(n)]=y[n]则 T[x(n-n0)]=y[n-n0]，这说明序列x(n)先移位后进行变换与它先进行变换后再移位是等效的。

线性时不变系统

线性时不变系统：既满足叠加原理又具有时不变特性，它可以用单位脉冲响应来表示。单位脉冲响应是输入端为单位脉冲序列时的系统输出，一般表示为h(n)，即h(n)=T[δ(n)]。

任一输入序列x(n)的响应y(n)=T[x(n)]=T[ δ(n-k)]；

由于系统是线性的，所以上式可以写成y(n)=T[δ(n-k)]；

又由于系统是时不变的，即有T[δ(n-k)]=h(n-k)；

从而得y(n)=h(n-k)=x(n)\*h(n)；

这个公式称为线性卷积，用“\*”表示。

齐次性

若激励f(t)产生的响应为y(t)，则激励Af(t)产生的响应即为Ay(t)，此性质即为齐次性。其中A为任意常数。

f(t)系统y(t)，Af(t)系统Ay(t)

叠加性

若激励f1(t)与f2(t)产生的响应分别为y1(t)， y2(t)，则激励f1(t)+f2(t)产生的响

应即为y1(t)+y2(t)，此性质称为叠加性。

线性

若激励f1(t)与f2(t)产生的响应分别为y1(t)， y2(t)，则激励A1f1(t)+A2f2(t)产生的响应即为A1y1(t)+A2y2(t)，此性质称为线性。

时不变性

若激励f(t)产生的响应为y(t)，则激励f(t-t0)产生的响应即为y(t-t0)，此性质称为不变性，也称定常性或延迟性。它说明，当激励f(t)延迟时间t0时，其响应y(t)也延迟时间t0，且波形不变。

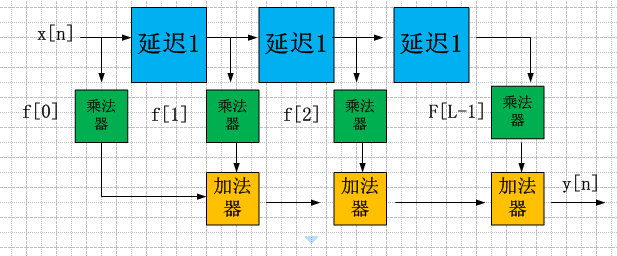
微分性

若激励f(t)产生的响应为y(t)，则激励f'(t)产生的响应即y’(t),此性质即为微分性。

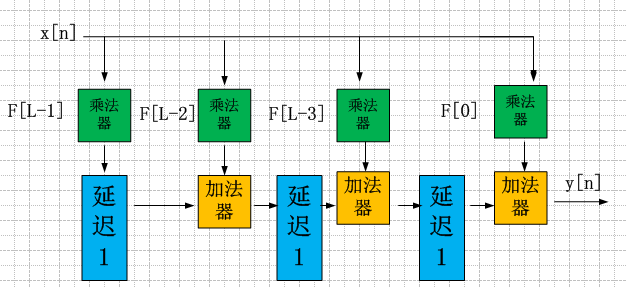
积分性

若激励f(t)产生的响应为y(t)，则激励f(t)的积分产生的响应即为y(t)的积分。此性质称为积分性。

FIR滤波器结构：



直接形式的FIR滤波器



转置结构的FIR滤波器

FIR滤波器特点：

* + FIR滤波器的最主要的特点是没有反馈回路，稳定性强，故不存在不稳定的问题；
  + FIR具有严格的线性相位，幅度特性随意设置的同时，保证精确的线性相位;
  + 设计方式是线性的，硬件容易实现；
  + 滤波器过度工程有有限区间；
  + FIR滤波器设计需要更多的参数，增加计算量

**1---FIR滤波器简介**

2017年09月08日 14:56:47

阅读数：7102

**1. FIR滤波器特性**

[Markdown中数学公式整理](http://blog.csdn.net/zdk930519/article/details/54137476)   
FIR（Finite Impulse Response）Filter：有限冲激响应滤波器，又称为非递归线性滤波器，与IIR对应。

FIR滤波器，顾名思义，其脉冲响应由有限个采样值构成。长度（抽头数）为*N* N 、阶数为*N*−1 N−1 的FIR系统的转移函数、差分方程和单位冲激响应分别如下列三式所示。

*H*(*z*)=∑ *n*=0 *N*−1 *h*(*n*)*z* −*n*  H(z)=∑n=0N−1h(n)z−n

*y*(*n*)=∑ *k*=0 *N*−1 $*h*(*k*)*x*(*n*−*k*) y(n)=∑k=0N−1$h(k)x(n−k)

*h*(*n*)=*h*(0)∗*δ*(*n*)+*h*(1)∗*δ*(*n*−1)+...*h*(*N*−1)∗*δ*[*n*−(*N*−1)] h(n)=h(0)∗δ(n)+h(1)∗δ(n−1)+...h(N−1)∗δ[n−(N−1)]

IIR（Infinite Impulse Response）Filter：无限冲激响应滤波器，又称为递归线性滤波器。   
FIR与IIR相对的   
**优点：**

* 可以具备线性相位特性
* 易于设计

**缺点：**

* 同样指标的滤波器，FIR需要更多的参数，即实现时消耗更多的计算单元，产生更大的延迟。

详细可参考：[IIR滤波器和FIR滤波器的区别与联系](http://blog.csdn.net/sky247391475/article/details/51659881)   
**线性相位的概念：**   
如果滤波器的N个实值系数为对称或者反对称结构，该滤波器具有线性相位。

*W*(*n*)=±*W*(*N*−1−*n*) W(n)=±W(N−1−n)

**特性：**通过线性相位滤波器的信号的所有频率部分具有相同的延迟量。

**2. FIR滤波器是怎么滤波的**

信号通过一个FIR滤波器其实就是信号与FIR滤波器的系数进行卷积（即乘累加）的过程。   
**简单信号模型**   
低频信号：即在时域上变化慢的信号，如1 1 1 1 2 2 2 2，还有直流信号1 1 1 1 1 1 1   
高频信号：即在时域上变化快的信号，如1 2 1 2 1 2 1 2   
**简单的滤波器模型**   
低通滤波器：1 1   
高通滤波器：1 -1

想象一下上面的信号与下面的信号做卷积的过程。

**3. FIR滤波器的设计**

可以通过下面两句话理解FIR滤波器的设计

* FIR滤波器的设计比较简单，就是要设计一个数字滤波器去逼近一个理想的低通滤波器。通常这个理想的低通滤波器在频域上是一个矩形窗。
* 在数字信号处理中，傅里叶变换用来求取时域信号的频率成分。因此，通过对特定频率幅度和相位的响应，求取傅里叶变换的逆变换（IDFT），就可以求出满足要求的数字滤波器。

逼近一个理想的滤波器 —-> 获得的序列是无限长的 —-> 需要截断 —-> 加窗

[使用窗函数设计FIR滤波器](http://blog.csdn.net/zhoufan900428/article/details/8969470)

**设计FIR滤波器需要提供的参数：**   
1）滤波器的类型：低通、高通、带通、带阻   
2）滤波器的采样频率   
3）滤波器的系数个数   
4）阻带衰减（dB）   
5）通带纹波（dB）   
6）过渡带带宽（Hz）

**4. FIR滤波器的实现**

**作者高亚军《基于FPGA的数字信号处理》中将FIR滤波器的实现讲的非常详细**   
FIR滤波器的实现很简单，根据其结构，其实就是延迟乘累加。   
[实现FIR滤波器这么简单，为什么各种数字信号处理教材堆砌那么多公式、引入那么多复杂概念，让人望而却步？](https://www.zhihu.com/question/29945169?sort=created)

但在实现时可以对延迟乘累加结构进行变形，以获得不同的资源消耗效果。

* **转置结构**

标准的FIR结构是先延迟，再乘累加，而转置的结构是先乘，在延迟累加。转置结构相当于改变了延时单元的位置，这就是所谓的**重定时（Retiming）**技术。   
转置结构的好处是：每两个加法器之间都插入了之前的延时器，缩短了关键路径，电路可以跑到更高的速度。

* **MAC实现（串行结构FIR）**

Multiply Accumulation Cell，乘累加模块   
**设计思想：**只用一个乘法器和一个累加器按时间顺序依次完成一次滤波运算所需的N次乘法和N-1次加法。   
关于累加器**位宽**的考虑，怎样保证累加运算不溢出，一般我们从乘法和加法的角度来考虑，但为了节省硬件资源，还可以从另一个角度来考虑：   
滤波运算输出的最大值为

|*y*| *max* =2 *B*−1 ∑ *n*=0 *N*−1 |*h*(*n*)| |y|max=2B−1∑n=0N−1|h(n)|

所以加法器位宽只要满足|*y*| *max*  |y|max 的需求即可   
[基于FPGA MAC FIR滤波器的实现](http://xueshu.baidu.com/s?wd=paperuri%3A%28d7139365176a0e359285c1e3421c4bf5%29&filter=sc_long_sign&tn=SE_xueshusource_2kduw22v&sc_vurl=http%3A%2F%2Fwww.docin.com%2Fp-1245246035.html&ie=utf-8&sc_us=15611184283661112979)   
**优劣势：**   
串行结构占用极少的资源（只用了一个乘加器），这是它的优势，而其吞吐量不会太高，这是它的不足。它正是“以速度换资源”思想的体现，在某些场合灵活使用会体现其应有的价值。如在滤波器的采样率较低时，可以采用这样结构，使用高频时钟处理，在不增加处理延时的情况下节省硬件资源。

* **DA实现**

分布式算数（Distribute Arithmetic, DA）是实现滤波器的一种方法，其基本思想是将数字滤波器内的乘法和加法运算用查找表和一个移位累加器来实现。   
DA的结构挺复杂的，但简单来讲可以说是利用了乘法运算可以使用移位相加来实现这么一个特性。

4∗2=8 4∗2=8

0*b*100左移*log* 2 2位=0*b*1000 0b100左移log22位=0b1000

使用DSP资源实现乘法运算可以达到更高的运行速度，如果不是DSP不够用，现代滤波器实现应该不会再使用古老的DA实现了吧？？

**FIR滤波器**

2012年11月17日 21:53:13

阅读数：17934

From： http://zaazbb.blog.163.com/blog/static/16897855920095205945813/

**FIR滤波器设计中常用的窗函数**

**:  矩形窗、三角窗，汉宁窗，汉明(Ham-ming)窗.**

FIR滤波器

**Part 1: Basics**

1.1 什么是FIR滤波器?

FIR 滤波器是在数字信号处理(DSP)中经常使用的两种基本的滤波器之一,另一个为IIR滤波器.

1.2 FIR代表什么?

FIR是有限冲激响应(Finite Impulse Response)的简称.

1.3 FIR(有限冲激响应)中的有限该如何理解?

冲激响应是有限的意味着在滤波器中没有发反馈.

1.4 FIR 怎么发音?

有些人直接读字母音 F-I-R; 也有人发做fir的音[:], fir是冷杉树.

1.5 FIR 滤波器外有什么其他选择?

DSP滤波器还有一类: IIR(无限冲激响应,Infinite Impulse Response).IIR滤波器使用反馈,因此当信号输入后,输出是根据算法循环的.

1.6 FIR滤波器与IIR滤波器比较?

每一种都有优缺点.但总得来说, FIR滤波器的优点远大于缺点,因此在实际运用中,FIR滤波器比IIR滤波器使用的比较多.

1.6.1 相对于IIR滤波器, FIR滤波器有什么优点?

相较于IIR滤波器, FIR滤波器有以下的优点:

　　\* 可以很容易地设计线性相位的滤波器. 线性相位滤波器延时输入信号,却并不扭曲其相位.

　　\* 实现简单. 在大多数DSP处理器, 只需要对一个指令积习循环就可以完成FIR计算.

　　\* 适合于多采样率转换,它包括抽取(降低采样率), 插值(增加采样率)操作. 无论是抽取或者插值, 运用FIR滤波器可以省去一些计算, 提高计算效率. 相反,如果使用IIR滤波器,每个输出都要逐一计算,不能省略,即使输出要丢弃.

　　\* 具有理想的数字特性. 在实际中，所有的DSP滤波器必须用有限精度（有限bit数目）实现，而在IIR滤波器中使用有限精度会产生很大的问题，由于采用的是反馈电路，因此IIR通常用非常少的bit实现，设计者就能解决更少的与非理想算术有关的问题。

　　\* 可以用小数实现. 不像IIR滤波器，FIR滤波器通常可能用小于1的系数来实现。（如果需要，FIR滤波器的总的增益可以在输出调整）。当使用定点DSP的时候，这也是一个考虑因素，它能使得实现更加地简单。

1.6.2 相较于IIR滤波器, FIR滤波器的缺点是什么?

相比较于IIR滤波器, 有时FIR滤波器为了得到一个给定的滤波响应特性,需要花费更多的存储器或者计算. 当然,用FIR滤波器去实现某些响应也是不实际的.

1.7 在描述FIR滤波器的时候,都要提到什么术语?

\* 冲激响应 - FIR滤波器的冲激响应实际上是FIR的系数.

　　\* 抽头(Tap) - FIR的抽头是系数或者延时对.FIR抽头的个数(通常用N来表示)意味着:1)实现滤波器所需要的存储空间, 2) 需要计算的数目, 3) 滤波器能滤掉的数量, 实际上,越多的抽头意味着有更多的阻带衰减, 更少的波纹,更窄的滤波等等.

　　\* 乘累加 (MAC) - 在FIR方面考虑,MAC是指把延时的数据采样与相应的系数相乘，然后累加结果。通常，FIR每一个抽头都需要一个MAC。大多数DSP微处理器实现MAC操作都是单指令周期。

　　\* 跃迁带（Transition Band） -在通带和阻带边沿之间的频带。跃迁带越窄，需要更多的抽头去实现滤波器。也有说，小的跃迁带就是一个sharp滤波器。

　　\* 延时线- 一组存储器单元，实现在FIR计算中的Z^-1延时。

　　\* 环形缓存 - 一个特殊的缓存，是首尾相连的。通常由DSP微处理器实现。

**Part 2: Properties**

2.1 线性相位

2.1.1 FIR滤波器和线性相位之间有什么关系?

大多数的FIR滤波器是线性相位滤波器. 当需要设计线性相位滤波器时, 通常使用FIR滤波器.

2.1.2 什么是线性相位滤波器?

线性相位是指滤波器的相位响应是频率的线性函数（在+/-180度）。因此滤波器的延时后，所有的频率相位相同。因而滤波器不会产生相位和延迟扭曲。在某些领域，比如数字解调器，没有相位或者延迟扭曲是FIR滤波器相对于其他IIR和模拟滤波器的一个关键优点

2.1.3 线性滤波器的条件是什么?

FIR滤波器经常被设计成为线性相位的，当然不是必须要这么做。如果滤波器的系数是关于中心系数对称的，也就是说第一个系数和最后一个系数相同，第二个系数和倒数第二个相同，那么FIR滤波器就是线性的。有奇数个系数的FIR滤波器，中心单独的系数没有对应的。

2.1.4 什么是线性相位FIR滤波器的延时?

非常简单的公式: 给定FIR滤波器有N个抽头,那么延时是(N - 1) / (2 \* Fs), 这里Fs是采样频率. 比如, 21抽头的线性相位滤波器运行在1kHz, 那么延时就是(21 - 1) / (2 \* 1 kHz)=10 微秒.

2.1.4 除了线性相位,还可以选择什么?

当然是非线性的了。实际上，最流行的选择是最小相位滤波器。最小相位滤波器，也叫最小延时滤波器，比线性相位滤波器具有更少的延时，当两者的幅度响应相同时以非线性相位特性。低通滤波器在它的冲击响应中心有最大的系数。而最小相位滤波器的最大系数在开始部分。

2.2 频率响应

2.2.1 什么是FIR滤波器的Z变换r?

对于N抽头的滤波器, 系数为h(k), 那么输出由:

　　y(n)=h(0)x(n) + h(1)x(n-1) + h(2)x(n-2) + ... h(N-1)x(n-N-1),

滤波器的z变换就是:

　　H(z)=h(0)z-0 + h(1)z-1 + h(2)z-2 + ... h(N-1)z-(N-1) , or

2.2.2 FIR滤波器的频率响应公式是什么?

H(z)中的变量z为连续的复数变量，可以描述为 z="r"·ejw，这里r是幅度，w是z的角度。如果令r=1，H(z)就变成了滤波器频率响应H(jw)。这也就意味着替代z为ejw，得到了滤波器频率响应H(w)。

　　H(jw)=h(0)e-j0w + h(1)e-j1w + h(2)e-j2w + ... h(N-1)e-j(N-1)w , or

使用欧拉公式, e-ja=cos(a) - jsin(a), 我们可以把H(jw)写成矩形表示:

　　H(jw)=h(0)[cos(0w) - jsin(0w)] + h(1)[cos(1w) - jsin(1w)] + ... h(N-1)[cos((N-1)w) - jsin((N-1)w)] , or

2.2.3 能用离散傅立叶变换(DFT)来计算FIR的频率响应么?

可以。对于N抽头的FIR，可以得到N evenly-spaced points of the frequency response by doing a DFT on the filter coefficients.但是，为了得到任意频率的频率响应，需要使用上边的公式。

2.2.4 FIR滤波器的DC增益指的是什么?

DC(0 Hz)输入信号包含每个采样都为1.0。通过延时线后，输出是所有系数的和。因而，在DC处滤波器的增益就是所有系数之和。

　　可以通过上边的公式进行验证。问我们设w为0， cos项就一直为1，而sin项则一直为0。因此频率响应就变成了：

2.2.5 如何调整FIR滤波器的增益?

简单地在系数上乘上因子.

**2.3 数字性质**

2.3.1 FIR滤波器是固有稳定的?

是的，因为没有反馈，任何有限的输入产生有限的输出。

2.3.2 什么使FIR滤波器的数字性质变好？

缺少反馈是关键。在计算机中实现FIR滤波器时，每个计算都产生数字错误。由于FIR滤波器没有反馈，因此不能够记住以前的错误。相反，IIR滤波器的反馈可能导致错误的积累。这个实际的影响就是，可以用更少的bit去实现与IIR滤波器相同精度的滤波器。比如，FIR滤波器通常用16位来实现的话，IIR滤波器就通常需要32位，或者更多。

2.4 为什么通常在多采样率系统中采用FIR滤波器而不采用IIR滤波器?

因为只有一小部分的计算需要用减采样或者插值滤波器来实现。

由于FIR滤波器不使用反馈，因而只有那些实际需要使用的输出才需要计算。比如，在减采样的时候（N个输出中只有一个有效），那么其他的N-1输出就不会进行计算。类似的，对于插值滤波器（在采样点中插入0来提高采样率），你不必实际地用FIR滤波器乘以系数，求和得到，你只需要忽略和这些值有关的乘加（因为它们不会改变结果）。

　　相反，因为IIR滤波器使用反馈，每个输入都必须使用，每个输入必须计算，因为所有的输入和输出对滤波器的反馈都有影响。

2.5 有哪些特殊的FIR滤波器?

Aside from "regular" and "extra crispy" there are:

　　\* 矩形 -矩形 FIR 滤波器是每个系数都是1.0的简单的滤波器。因而对于N个抽头的矩形滤波器，它的输出仅仅是过去N个采样之和。由于矩形FIR只能实现加法，因此当乘法器实现比较昂贵时，在硬件实现中会考虑。

　　\* 希尔伯特变换（Hilbert Transformer） - 希尔伯特变换是把信号相移90度。它们经常被用在给定实数部分，产生虚数部分。

　　\* 差分（Differentiator） -差分器的幅度响应是频率的线性函数。现在已经不流行了，但是以前曾经在FM解调器上使用过。

　　\* Lth-Band - 也叫做“Nyquist"滤波器，这些滤波器是在多速率应用中特殊的一类滤波器。主要的卖点是，每L个系数有一个为0，那么就将减少乘累加操作的实现（著名的半带滤波器就是这一种）。

　　\* Raised-Cosine - 这是一种特殊类型的滤波器，有时会用在数字数据应用方面。（通带上的频率响应是被上移一个常数的cos形状）。

**Part 3: Design**

3.1 有哪些设计FIR滤波器的方法?

三种最流行的设计方法:

* Parks-McClellan: Parks-McClellan 方法( MATLAB里用Remez)是设计FIR滤波器中可能是使用最光的.method (inaccurately called "Remez" by Matlab) is probably the most widely used FIR filter design method. It is an iteration algorithm that accepts filter specifications in terms of passband and stopband frequencies, passband ripple, and stopband attenuation. The fact that you can directly specify all the important filter parameters is what makes this method so popular. The PM method can design not only FIR "filters" but also FIR "differentiators" and FIR "Hilbert transformers".
* Windowing:. In the windowing method, an initial impulse response is derived by taking the Inverse Discrete Fourier Transform (IDFT) of the desired frequency response. Then, the impulse response is refined by applying a data window to it.
* Direct Calculation: The impulse responses of certain types of FIR filters (e.g. Raised Cosine and Windowed Sinc) can be calculated directly from formulas.

3.2 如何实际地设计FIR滤波器?

当然是用FIR设计程序呀. 虽然可以使用手工亲自的方法进行设计滤波器,但是使用FIR滤波器程序比较简单.

**Part 4: Implementation**

4.1 实现FIR滤波器基本的方法是什么?

FIR滤波器的结构上包含两个东西:一个是采样点延迟线,一个是系数. 可以由以下方法实现FIR滤波器:

　　1. 把输入的采样点放入到延迟线中.

　　2. 把延迟线中的数与相应的系数相乘并累加.

　　3. 移位, 使下一个输入采样能进入延迟线.

4.2 用C语言如何实现FIR滤波器?

为了展示众多的方法和技巧，这里提供用C语言实现的FIR滤波器算法。

[fir\_algs\_1-0.c](http://www.dzkf.cn/upimg/soft/1_061231135622.rar) C 源码

[fir\_algs\_1-0.zip](http://www.dzkf.cn/upimg/soft/1_061231135728.zip) C 源码（ MS Visual C++ 6.0 工程文件）

包括以下功能模块：

　　1. fir\_basic: 实现基本的FIR滤波器

　　2. fir\_circular: 说明环行buffer是如何实现FIR的。

　　3. fir\_shuffle: 一些TI的处理器上使用的shuffle down技巧

　　4. fir\_split: 把FIR滤波器展开为两块，避免使用环行缓存。

　　5. fir\_double\_z: 使用双精度的延迟线，使可以使用一个flat buffer。

　　6. fir\_double\_h: 使用双精度的系数，使可以使用一个flat buffer。

4.3 用汇编如何实现FIR滤波器?

FIR滤波器的汇编算法是跟处理器对应的，但是大多数普通的系统会使用一个DSP处理器提供的环行缓存。

　　1. 配置环行缓存。加载系数和延迟线指针。然后对每个采样点执行以下操作：

　　2. Store the incoming data in the delay line; increment the delay-line pointer.Digital

　　3. Clear the multiplier-accumulator.

　　4. Loop over all coefficients/delays; accumulate the values obtained by multiplying the coefficients by the delayed samples.

　　5. Round or truncate the result as the FIR output.

　　Alternatively, a "shuffle down" method is used in Texas Instruments' older fixed-point processors to implement circular buffers. The processor literally moves each sample delay values by one slot during each multiply-accumulate (via the "MACD" instruction).

　　Each DSP microprocessor manufacturer provides example FIR assembly code in its data books or its application handbooks, so be sure to look at those before you "reinvent the circular buffer".

4.4 如何测试以及实现的FIR滤波器?

　　Here are a few methods:

* Impulse Test: A very simple and effective test is to put an impulse into it (which is just a "1" sample followed by at lest N - 1 zeroes.) You can also put in an "impulse train", with the "1" samples spaced at least N samples apart. If all the coefficients of the filter come out in the proper order, there is a good chance your filter is working correctly. (You might want to test with non-linear phase coefficients so you can see the order they come out.) We recommend you do this test whenever you write a new FIR filter routine.
* Step Test: Input N or more "1" samples. The output after N samples, should be the sum (DC gain) of the FIR filter.
* Sine Test: Input a sine wave at one or more frequencies and see if the output sine has the expected amplitude.
* Swept FM Test: From Eric Jacobsen: "My favorite test after an impulse train is to take two identical instances of the filter under test, use them as I and Q filters and put a complex FM linear sweep through them from DC to Fs/2. You can do an FFT on the result and see the complete frequency response of the filter, make sure the phase is nice and continuous everywhere, and match the response to what you'd expect from the coefficient set, the precision, etc."

4.5 在实现FIR滤波器的过程中有什么有用的技巧?

　　FIR tricks center on two things 1) not calculating things that don't need to be calculated, and 2) "faking" circular buffers in software.

4.5.1 如何跳过不必要的计算?

　　First, if your filter has zero-valued coefficients, you don't actually have to calculate those taps; you can leave them out. A common case of this is "half-band" filter, which have the property that every-other coefficient is zero.

　　Second, if your filter is "symmetric" (linear phase), you can "pre-add" the samples which will be multiplied by the same coefficient value, prior to doing the multiply. Since this technique essentially trades an add for a multiply, it isn't really useful in DSP microprocessors which can do a multiply in a single instruction cycle. However, it is useful in ASIC implementations (in which addition is usually much less expensive than multiplication); also, some newer DSP processors now offer special hardware and instructions to make use of this trick.

4.5.2 How do I fake circular buffers in software?

　　When hardware support for circular buffers isn't available, you have to "fake" them. Also, since ANSI C has no construct to describe circular buffers, most C compilers can't generate code to use them, even if the target processor has them.

　　You can always implement a circular buffer by duplicating the logic of a circular buffer in software (and many have), but the overhead can be prohibitive; the circular-fake might take several instructions to implement, compared to just a single instruction to do the multiply-accumulate operation. Therefore you need to fake it.

　　Here are several basic techniques to fake circular buffers:

　　1. **Split the calculation**: You can split any FIR calculation into its "pre-wrap" and "post-wrap" parts. By splitting the calculation into these two parts, you essentially can do the circular logic only once, rather than once per tap. (See fir\_double\_z in FirAlgs.c above.)

　　2. **Duplicate the delay line**: For a FIR with N taps, use a delay line of size 2N. Copy each sample to its proper location, as well as at location-plus-N. Therefore, the FIR calculation's MAC loop can be done on a flat buffer of N points, starting anywhere within the first set of N points. The second set of N delayed samples provides the "wrap around" comparable to a true circular buffer. (See fir\_double\_z in FirAlgs.c above.)

　　3. **Duplicate the coefficients**: This is similar to the above, except that the duplication occurs in terms of the coefficients, not the delay line. Compared to the previous method, this has a calculation advantage of not having to store each incoming sample twice, and it also has a memory advantage when the same coefficient set will be used on multiple delay lines. (See fir\_double\_h in FirAlgs.c above.)

　　4. **Use block processing**: In block processing, you use a delay line which is a multiple of the number of taps. You therefore only have to move the data once per block to implement the delay-line mechanism. When the block size becomes "large", the overhead of a moving the delay line once per block becomes negligible.