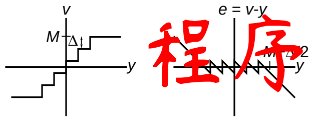
Oversampling Delta-Sigma Data Converters
The One-Page Story

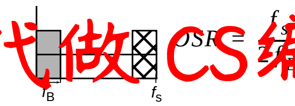
Quantization



As long as the quantizer does not overload $|e| \le \Delta/2$.

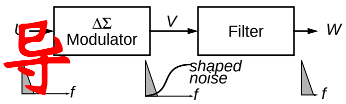
If i) the quantizer does not overload, ii) the input to the quantizer is busy and iii) the number of quantization levels is large, then the ... a power $\sigma_e^2 = \Delta^2/12 =$ ...

Oversampling



For white noise, the power in the band-of-interest is the power of the signal divided by *OSR*.
⇒ Oversampling reduces noise.
The first alias is approximately 2*OSR* times higher in frequency than the upper passband edge.
⇒ Anti-aliasing requirements are relaxed by oversampling.

Basic ΔΣ Architecture



For an ADC system, the modulator is analog and the (decimation) filter is digital. The anti-alias filter which precedes the modulator is not shown.

For a DAC system, the modulator is digital and the filter is analog. The interpolation filter which precedes the modulator is not shown.

MOD1

$V(z) = U(z) + \dots$

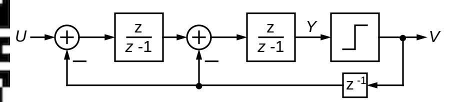
```
double u, v= ...
while( cin > ...
    x += u - v;
    v = x<0?-1:1;
    cout << v << endl;
    }
```

$$IQNP = \frac{\pi^2\sigma_e^2}{3(OSR)^3} \begin{cases} = 2\sigma_e^2(OSR)^{-1} & \text{for sinc}^1 \\ = 2\sigma_e^2(OSR)^{-3} & \text{for sinc}^2 \end{cases}$$

$$(|u| \le 1) \Rightarrow (|x| \le 2)$$

Quantization noise is not white– susceptible to "limit cycles," esp. with small DC inputs. For DC inputs, the output spectrum is discrete.

MOD2



$$V(z) = U(z) + (1 - z^{-1})^2 E(z)$$

```
double u, v=0, x1=0, x2=0;
while( cin >> u ){
    x1 +=  u - v;
    x2 += x1 - v;
    v = x2<0?-1:1;
    cout << v << endl;
    }
```

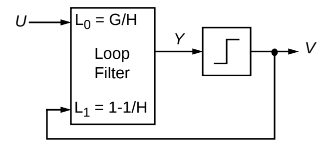
$$IQNP = \frac{\pi^4\sigma_e^2}{5(OSR)^5} \begin{cases} = 6\sigma_e^2(OSR)^{-4} & \text{for sinc}^2 \\ = 6\sigma_e^2(OSR)^{-5} & \text{for sinc}^3 \end{cases}$$

(u constant, $|u| \le 1$) ⇒

$$\left(|x_1| \le |u| + 2, \quad |x_2| \le \frac{(5 - |u|)^2}{8(1 - |u|)}\right)$$

Less susceptible to limit cycles. Frequent quantizer overload (which leads to excess quantization noise) can be avoided by the use of a less aggressive NTF.

MODn



$$V(z) = G(z)U(z) + H(z)E(z)$$

```
H = synthesizeNTF(n=3,OSR=64, ...
     opt=0,Hinf=1.5,f0=0);

[v xn xmax y] = simulateDSM(u,H,nlev=2,x0=0);
```

$$IQNP = \frac{\sigma_H^2\sigma_e^2}{OSR}$$

`sigma_H = rmsGain(H,0,0.5/OSR);`

Lee's Rule (with margin):

$\|H\|_\infty < 1.5 \Rightarrow$ probably stable

Can trade-off stability (in terms of the frequency of quantizer overload or in terms of the maximum stable input) for increased noise suppression by increasing the bound on $\|H\|_\infty$.

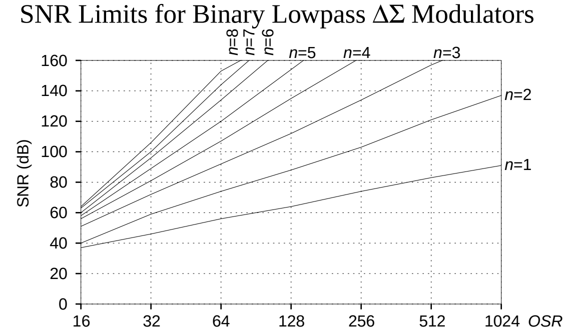
ΔΣ Toolbox Quick Reference

```
MAIN FUNCTIONS
ntf = synthesizeNTF(order=3,OSR=64,opt=0,H_inf=1.5,f0=0)
ntf = clans(order=4,OSR=64,Q=5,rmax=0.95,opt=0)
[snr,amp,k0,k1,sigma_e2] = predictSNR(ntf,OSR=64,amp=...,f0=0)
[v,xn,xmax,y] = simulateDSM(u,ABCD,nlev=2,x0=0) or
[v,xn,xmax,y] = simulateDSM(u,ntf,nlev=2,x0=0)
[snr,amp] = simulateSNR(ntf,OSR,amp=...,f0=0,nlev=2,f=1/(4*OSR),k=13)
[a,g,b,c] = realizeNTF(ntf,form='CRFB',stf=1)
ABCD = stuffABCD(a,g,b,c,form='CRFB')
[a,g,b,c] = mapABCD(ABCD,form='CRFB')
[ABCDs,umax]=scaleABCD(ABCD,nlev=2,f=0,xlim=1,ymax=nlev+5,umax,N=1e5)
[ntf,stf] = calculateTF(ABCD,k=1)
[gu,gv,H,L0,L0k] = designLCBP(n=3,f0=1/16,fb=1/128,Hinf=1.6,t1=0,…)
[sv,sx,sigma_se,max_sx,max_sy] = simulateESL(v,mtf,M=16,dw=[1…],sx0=[0…])
[f1,f2,info] = designHBF(fp=0.2,delta=1e-5,debug=0)
[s,e,n,o,Sc] = findPIS(u,ABCD,nlev=2,options)

AUXILIARY FUNCTIONS
window = ds_hann(N)
snr = calculateSNR(hwfft,f)
sigma_H = rmsGain(H,f1,f2)
H_inf = infnorm(H)
[A B C D] = partitionABCD(ABCD, m)
tf_z = evalTF(tf,z)
figureMagic( xRange, dx, xLab, yRange, dy, yLab, size )
printmif(file,size,font,fig)

The toolbox is available from http://www.mathworks.com/matlabcentral/
fileexchange.
```

SNR Limits for Binary Lowpass ΔΣ Modulators



CRFB Topology for a 3rd-Order Modulator