Virtual Analog (VA) Filter Implementation and Comparisons

Will Pirkle

I have had several requests from readers to do a Virtual Analog (VA) Filter Implementation plug-in. The source of these designs is a book electronically published in June 2012 named *The Art of VA Filter Design* by Vadim Zavalishin. This awesome piece of work is free and available from many sources including my own site www.willpirkle.com. This short book is an excellent introduction to basic analog filtering theory as well as digital transformations. It is so concise in this respect that I am considering using it as part of the text materials for an Advanced Analog Circuits class I teach. I highly recommend this excellent book - you will need to understand its content to use this App Note; for the most part I use the same variable names as the book so you will want to use it as a reference. Zavalishin's derivations and descriptions are so well thought out and so well written that it makes no sense for me to repeat them here and I don't think it can be simplified any more that it already is in his book.

Another reason that I enjoyed this book is that the author followed a similar derivation to reach the bilinear transform as my DSP professor (Claude Lindquist) did when I was in graduate school; in fact I use part of that same derivation in my classes and book. Also similar was the use of an integrator as a prototype filter to generate the discrete time transforms. Since integrators have simple transfer functions, analysis easily leads to various transforms. Zavalishin introduces two digital integrators which he names *naive* and *trapezoidal*. These integrators lead to two transforms. The naive integrator leads to the Reverse Euler Transform while the trapezoidal integrator leads to the Bilinear Transform.

Before proceeding to the plug-in implementations I'd like to share another take on the analysis. As noted in my book and in various other sources, the bilinear transform can be found by sampling an analog transfer function H(s) to produce $H_s(s)$. The idea is to find some function g(z) such that:

$$H(s) \xrightarrow{s=g(z)} H(z) \xrightarrow{z=e^{j\omega T}} H_s(s)$$

After using s = g(z) to replace the s with z terms, the resulting discretized filter is evaluated at:

$$z = e^{j\omega T}$$

$$T = \text{ one sample period}$$

to produce a sampled version of the analog transfer function. In the s-plane, the sampled version $H_s(s)$ has replicated sets of poles and zeros that repeat at the sample rate. The Nyquist Frequency limits us to just one set of them. For an analog integrator:

$$H(s) = \frac{1}{s}$$

which has a real and pole at a finite value (position) on the left hand plane and a zero at infinity. The zero at infinity is shown in Figure 4.1 on the left of the two hash marks. The ideal integrator produces a lowpass filter with a cutoff frequency of 0Hz.

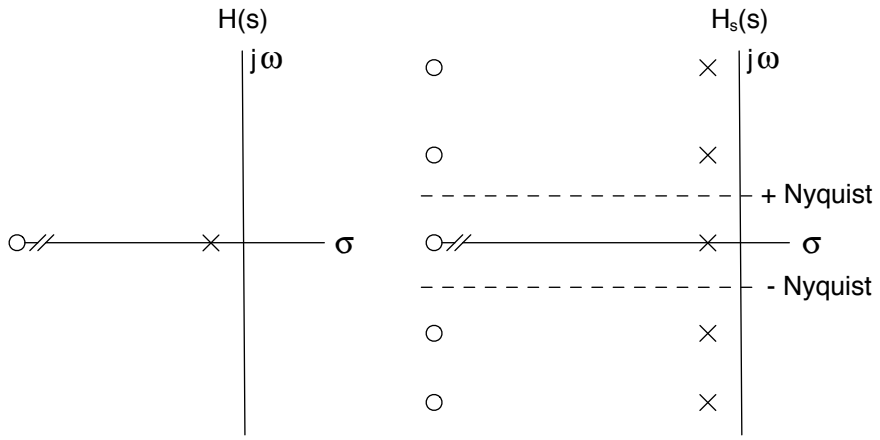


Figure 4.1: The sampled version of H(s) has repeating sets of poles and zeros (to infinity in each direction).

The key function that does the mapping from s to z is

$$z = e^{j\omega T}$$

$$s = j\omega$$

then

$$z = e^{sT}$$

$$\ln(z) = sT$$

or

$$sT = \ln(z)$$

The problem here of course is the log function. This equation could be solved with a Taylor Expansion:

$$sT = 2\left[\frac{z-1}{z+1} + \frac{1}{3}\left(\frac{z-1}{z+1}\right)^3 - \frac{1}{5}\left(\frac{z-1}{z+1}\right)^3 + ...\right] \qquad \text{Re}(z) \geq 0$$

Taking the first term leads to the Bilinear Transform:

$$s \simeq \frac{2}{T}\frac{z-1}{z+1}$$

This is only one answer, however. There are a few more that can be used, with other limitations on z, for example:

$$sT = \left[(z-1) - \frac{1}{2}(z-1)^2 + \frac{1}{3}(z-1)^3 - ...\right] \qquad 0 < |z| \leq 2$$

Taking the first term only leads to the Forward Euler Transform:

$$s \simeq \frac{1}{T}(z-1)$$

Another solution is:

$$sT = \left[\frac{z-1}{z} + \frac{1}{2}\left(\frac{z-1}{z}\right)^2 + \frac{1}{3}\left(\frac{z-1}{z}\right)^3 + ...\right] \qquad \text{Re}(z) > 0.5$$

Taking the first term yields the Reverse Euler Transform:

$$s \simeq \frac{1}{T}\frac{z-1}{z}$$

You can design digital integrators by substituting the transform terms for the s-terms. Since an integrator has the transfer function:

$$H(s) = \frac{1}{s}$$

this has the effect of flipping the transform upside down to produce the transfer function. It should be noted that taking more than the first term of any of the series that produce these transform results above will result in a digital realization that does not have the same filter order; in the above cases, the polynomial order of the analog filter is preserved in the digital version.

Some consider using the series expansions above to be a “clumsy” approach to finding digital transformations and instead prefer the reverse direction of operation - the method is to approximate continuous integrators by discrete integrators, then flip their transfer functions. There are actually a whole slew of digital transforms (including the three above) which can be found with the integrator -> transform method. A few more include:

The Modified Bilinear Transform

$$s = \frac{2}{T}\frac{z(z-1)}{z+1}$$

The Lossless Transform

$$s = \frac{1}{T}\frac{z-1}{z^{1/2}}$$

The Optimum Transform

$$s = \frac{2}{T}\frac{z-1}{z^{1/4}+z^{3/4}}$$

Even more transforms - as well as a method for generating transforms from any analog prototype filter with all real poles/zeros - can be found in Lindquist’s book *Adaptive and Digital Signal Processing*.

Since an integrator calculates the area under a curve, it might be useful to examine this approximation for a few of these integrators:

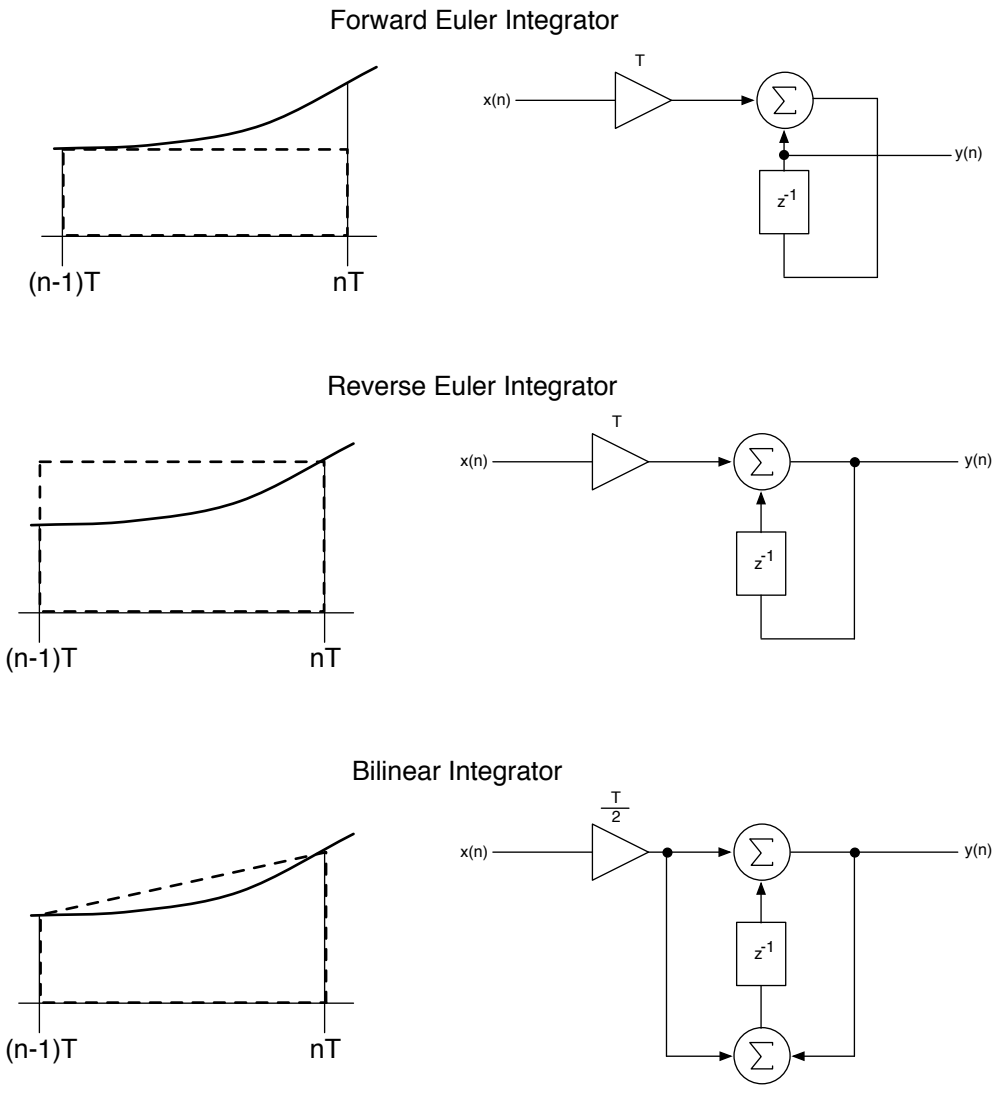


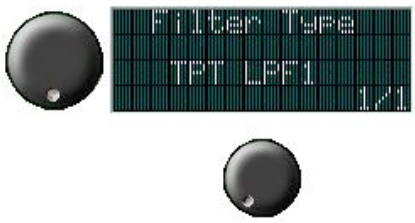
Figure 4.2: Several digital integrators with area of integration shown in dashed lines

Figure 4.2 reveals why the bilinear transform makes a good choice. The Forward Euler Integrator is also called a “zero order hold” integrator. It does a poor job of approximating the area under the curve simply holding the x(n-1) value. The Reverse Euler Integrator is what Zavalishin refers to as the *naive* integrator. Its results are also poor as it holds the current value backwards. The Bilinear Integrator is what Zavalishin refers to as the *trapezoidal* integrator. Of the three it obviously produces the closest approximation and you can clearly see the trapezoidal shape and the linear part of bilinear. Each of these integrators produces a transform. Only the Optimum Integrator produces a better result than the Bilinear Integrator however it contains fractional sample terms, $z^{-1/2}$ and $z^{-3/4}$ making it impractical for realization.

Of most important note is the way the bilinear transform maps analog poles and zeros to digital ones - it maps the infinite area on the left hand side of the s-plane to a finite area inside the unit circle in the z-plane. The first problem is frequency warping but that can be solved by pre-warping the analog cutoff

frequency. A bigger problem (for trying to exactly replicate analog frequency responses) is that the mapping takes zeros at infinity in the s-plane and maps them to zeros at z = -1 in the z-plane. This produces an error in the high frequency response of bilinear transformed lowpass filters; the analog and digital responses diverge for any analog transfer functions that have zeros at infinity (not all of them do, for example the first order high pass filter has its zero at s = 0 which becomes the correct zero at z = +1 after the transform). See my section on Massberg filters in my book for an interesting solution to this issue.

The filters in *The Art of VA Filter Design* are based off the trapezoidal integrator which produces a bilinear transformation. These filters also have the same problem with zeros at infinity mapped to z = -1 which sometimes produces errors in the responses compared with the analog counterparts. The reason these filters are so interesting (and titled Virtual Analog) is that they mathematically preserve the original analog filter topology, including (in most cases) zero-delay loops. The filters are called ***Topology Preserving Transform*** filters or ***TPT*** filters. Also, they do not use the direct form filter structures and are less susceptible to problems that result from using them. They are more efficient in processing as well as in ease of coefficient calculation. See *The Art of VA Filter Design* for more details on the problems with traditional direct form structures.



The VAFilters are all coded in a RackAFX Project named *VAFilters* available with this App Note. You control the filter type with the LCD Control's value knob. There are many filters to play with as I also included the normal BZT-biquad versions and the Massberg Analog Matched versions for the LPF cases; on the left is what the 1st Order TPT filter would look like chosen in the control; the value knob is the one on the bottom.

1st Order Filters

Figure 4.3 shows the block diagram for the one-pole TPT filter providing both low pass (LP) and high pass (HP) outputs. The HP output is simply $y_{HP}(n) = x(n) - y_{LP}(n)$. This block diagram is fully derived and explained in the *The Art of VA Filter Design.*

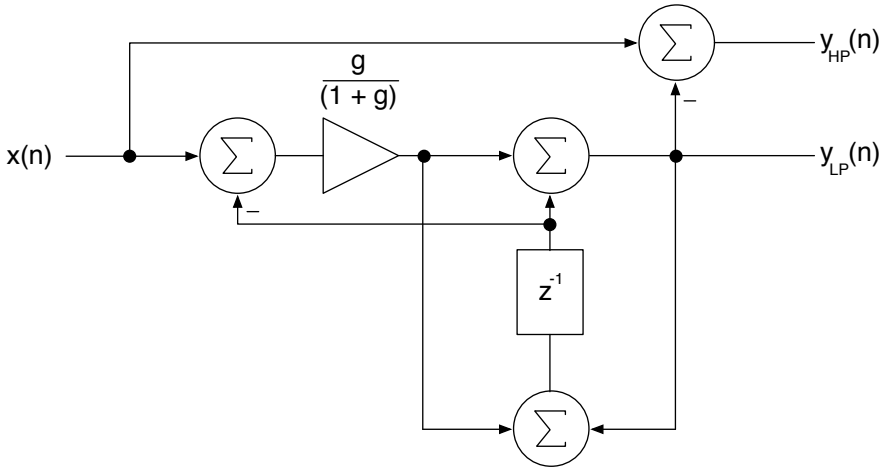


Figure 4.3 the First Order TPT Filter Topology

The C++ code for this filter is simple. In RackAFX I defined a 2-channel array to use for the z^{-1} registers (one each for left and right). The array is called m_fZ1[2]. Because the trapezoidal integration warps the frequency response, pre-warping via the tan() function is required. Notice the rule of "read before write" is followed (see my book for details on that).

```
// pre-warp the cutoff- these are bilinear-transform filters
float wd = 2*pi*m_fFc;
float T  = 1/(float)m_nSampleRate;
float wa = (2/T)*tan(wd*T/2);
float g  = wa*T/2;

// big combined value
float G = g/(1.0 + g);

// 2-channel output buffer for LPF
float LP[2];

// do the filter, see VA book p. 46
//
// form sub-node value v(n)
float v = (pInput[0] - m_fZ1[0])*G;

// form output of node + register
LP[0] = v + m_fZ1[0];

// setup for next time through
m_fZ1[0] = LP[0] + v;
```

The HPF output is formed simply doing the subtraction in the block diagram; that code is trivial. For stereo signals, the right channel (m_fZ1[1] and LP[1]) is processed in the same way.

In the sample project that goes with this App Note, I also implement:

- the (old fashioned) BZT (bilinear z-transform) version of LPF and HPF using the biquad structure.
- the Massberg Analog Modeling filter (see my book) which also uses a biquad structure

You can use the RackAFX analyzer to verify that the frequency responses of the TPT and BZT-biquad filters are for all practical purposes, identical. You can compare with the Massberg version to see the error in the frequency response at high cutoff values. The Massberg filter's response is closer to the analog version however it's coefficient calculations are far more detailed and it will suffer from direct-form implementation issues. You can also run audio through the filters - this is the whole point behind RackAFX - and compare the audible results of the three filters.

> The RackAFX Analyzer works by shooting impulses into your filter (each time you move a control) and taking the FFT of the IR. You are looking at the "true" performance of your plug-in and not a mathematical evaluation of a transfer function. The FFT's poor resolution at low frequencies is an issue here.

I refer to the (old fashioned) BZT-biquad version simply as "BZT" elsewhere in the text.

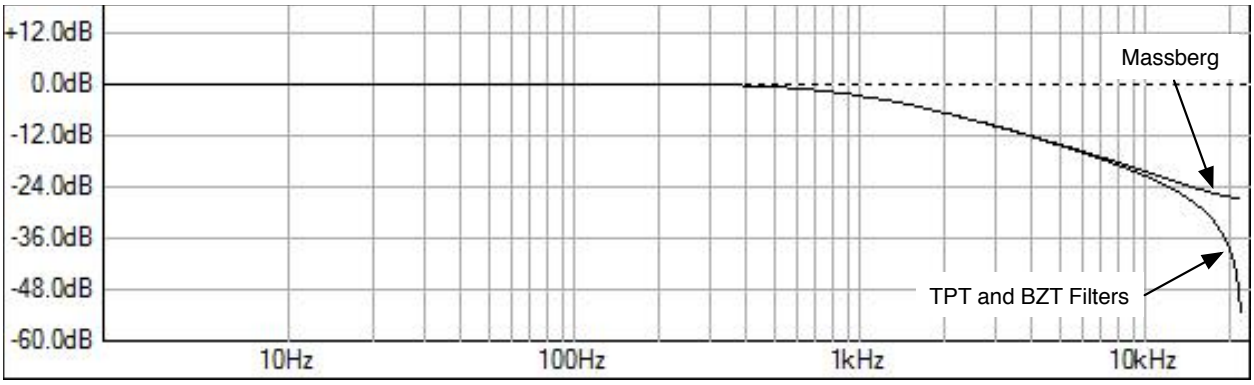


Figure 4.4: Comparison of TPT/BZT and Massberg filters

The screenshot above is directly from RackAFX; the latest version includes the ability to take snap-shots so you can compare responses or make family-of-curve plots. The TPT and BZT curves are right on top of one another and are identical. The error at high frequencies is apparent; the zero at Nyquist pulls the responses of the TPT and BZT filters down at high frequencies. The Massberg nearly perfectly matches the analog filter's response at high frequencies and Nyquist. As we will see later, this HF error will cause issues with the Moog Ladder filter response as well.

Oversampled 1st Order Filters

We see that the TPT filters produce an essentially identical frequency response as their BZT-biquad counterparts and that there is an error at high frequencies for the 1st Order LPF. Oversampling will mitigate this issue. I implemented 4X oversampling with a 512-tap FIR filter designed with RackAFX's FIR Designer Plug-In. The passband frequency is 20kHz and the stop-band frequency is 22kHz. The ripple is < 0.1dB and the stop-band attenuation is well below the 16-bit noise floor, at around -125dB. See **AN-1** for details on the oversampling operations and example plug-in.

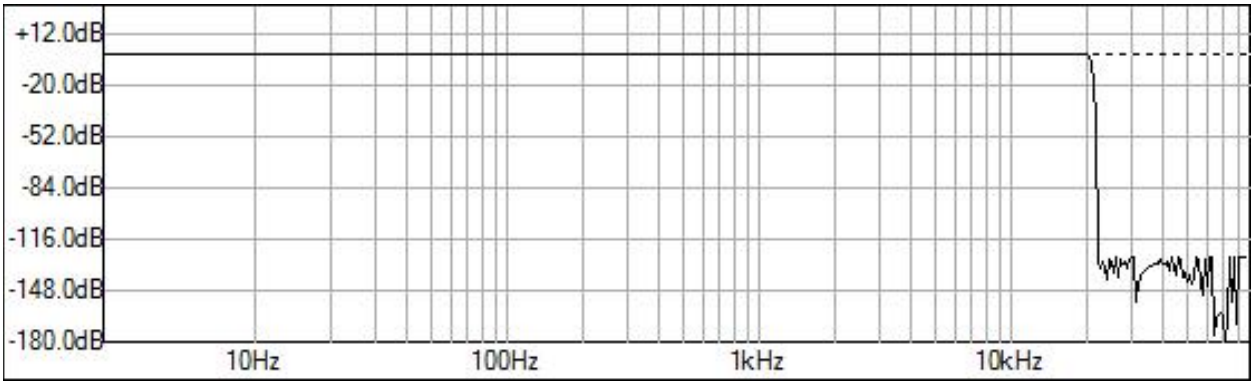


Figure 4.4a: the 512-tap FIR filter used in both the Interpolator and Decimator; note the filter is designed at 176.4kHz

By using 4X oversampling, the new Nyquist frequency is 88.2kHz so at our (normal) Nyquist frequency of 22.050kHz, the zeros at the *oversampled* Nyquist have no effect and the rolloff is linear across frequency in dB. Figure 4.4b shows the results for the first order LPF. Notice that the anti-imaging filter's band edge

(from the decimator) is visible, but the frequency responses just up to that point now look “analog” with no HF losses. Oversampling to the rescue!

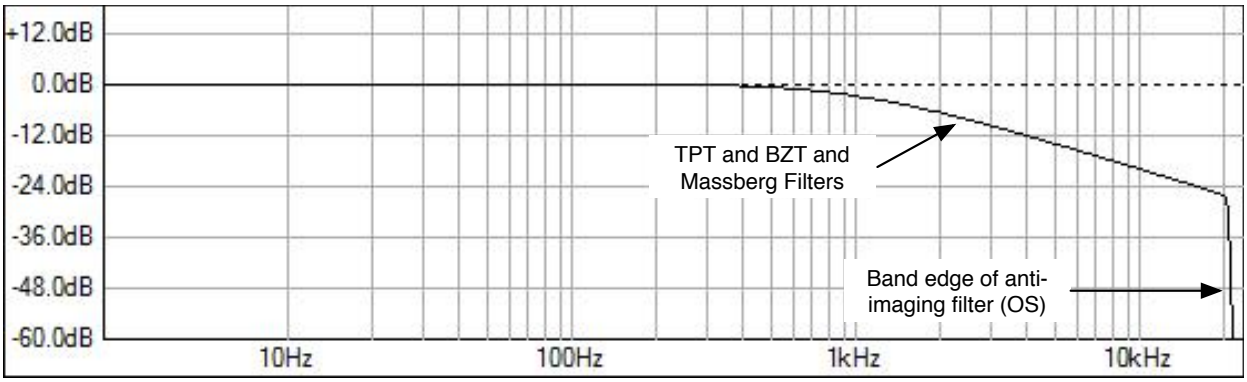


Figure 4.4b: with 4X oversampling, all three filters lie on top of one another; their responses are essentially identical.

2nd Order Filters

I also implemented all of the 2nd Order filters from the VA book. They are all based on the Stave Variable Filter structure. The analog circuit is:

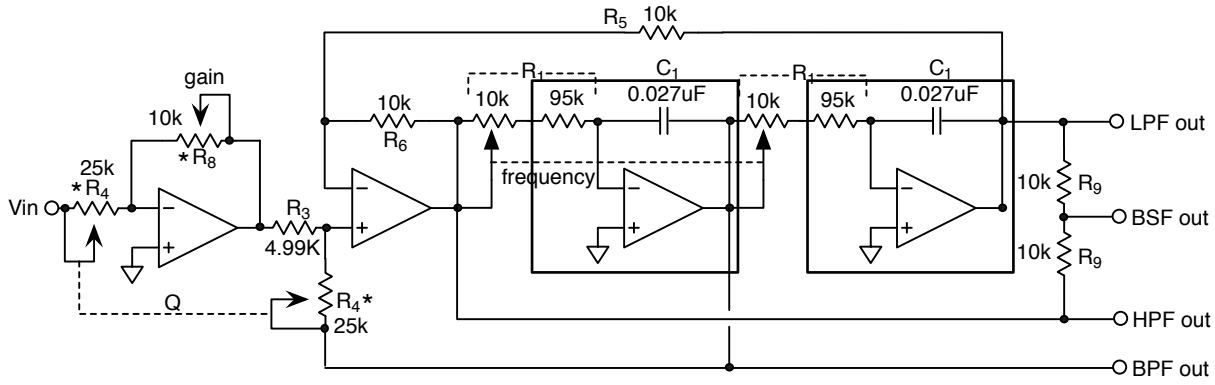


Figure 4.5: the analog Stave Variable Filter (SVF)

The pair of op-amp integrators are in the superimposed rectangles. You can see the outputs from the different locations within the structure. You can also see the global feedback path through R_5 and the second feedback path from the first integrator (that connects to BPF out). Compare this with the TPT filter in the VA Book (pp. 77, 81) and you can see how the topology is preserved.

Figure 4.6 shows the block diagram for the TPT version of the filter.

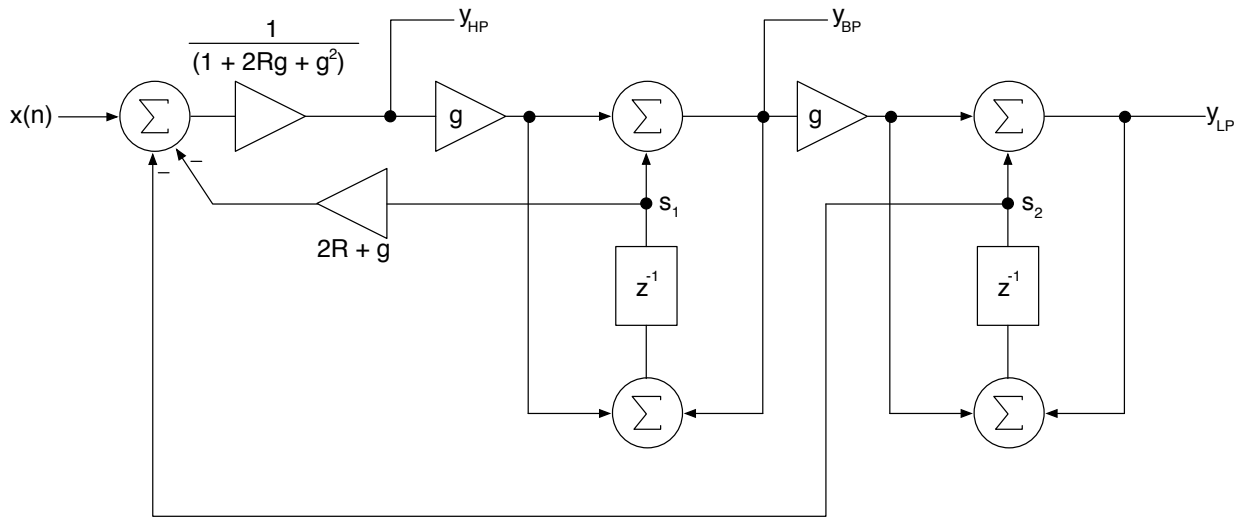


Figure 4.6: the TPT version of the SVF filter, the zero-delay loop has been resolved.

If you analyze my block diagram, you will be able to verify the y_{HP} output's difference equation (p. 80 in the VA book):

$$y_{HP}(n) = \frac{x(n) - 2Rs_1(n) - gs_1(n) - s_2(n)}{1 + 2Rg + g^2}$$

The y_{HP} is calculated first, then applied to the input of the first integrator. The read-before-write is used (i.e. s1 and s2 are read first and updated last). The difference equations for all the other filters are:

$$y_{BP}(n) = gy_{HP}(n) + s_1(n)$$
$$y_{LP}(n) = gy_{BP}(n) + s_2(n)$$
$$y_{UBP}(n) = 2Ry_{BP}(n)$$
$$y_{BSHELF}(n) = x(n) + 2KRy_{BP}(n)$$
$$y_{NOTCH}(n) = x(n) - 2Ry_{BP}(n)$$
$$y_{APF}(n) = x(n) - 4Ry_{BP}(n)$$
$$y_{PEAK}(n) = y_{LP}(n) - y_{HP}(n)$$

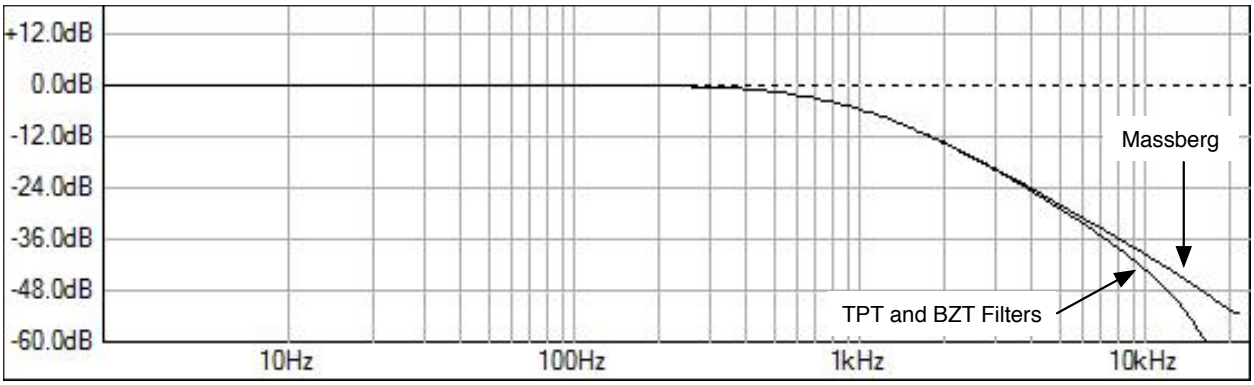


Figure 4.7: comparison of TPT, BZT and Massberg 2nd order responses

The only real detail with the 2nd Order Filters is that we usually specify them with a Q value whereas Zavalishin uses R; what we call “critically damped” or Q = 0.5 in analog corresponds to R = 1 for Zavalishin. So:

$$R = 1/2Q$$

Once again for the LPF we observe an error at very high frequencies in Figure 4.7 as the Massberg filter matches the analog response much closer (it does have an error but it is slight and we will see that error later) and that the TPT and BZT outputs are for all practical purposes identical.

The only other analog filter (from the filters implemented in the above difference equations) that has zeros at infinity is the bandpass filter (BPF). Its analog zeros are at both s = 0 as well as s = +/-j(infinity) (on the imaginary axis); these zeros get mapped to z = +1 (correct) and z = -1 (error). There is no Massberg equivalent for the BPF. The Plug-in implementations all check out correctly and are verified with the RackAFX Analyzer.

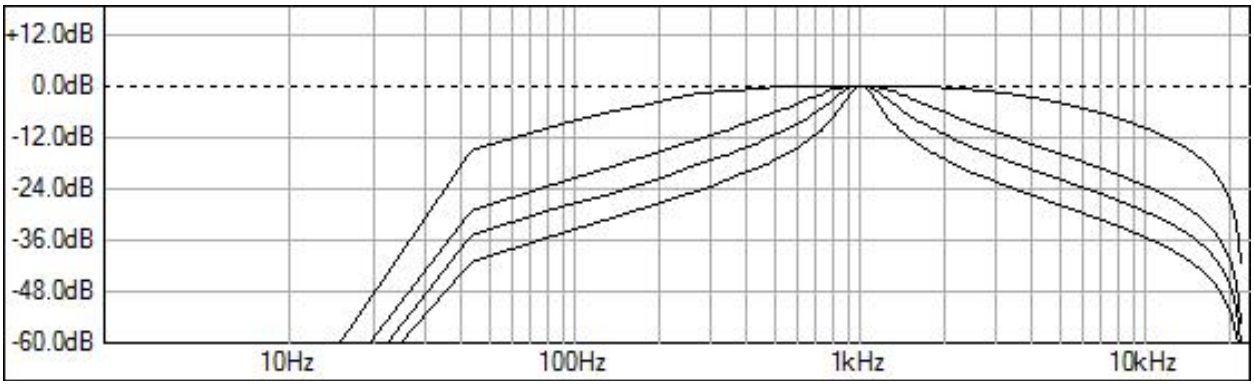


Figure 4.8: Unity-gain normalized BPF responses for Q = 0.5, 1.0, 2.5 and 10

You can see the effect of the zero at Nyquist in Figure 4.8 as the responses all get pulled down near Nyquist. You can also see the problem with using the FFT of an IR, for this 1024 point FFT the bins are spaced by 43Hz so only one data point exists between 0 and 43Hz, two between 43 and 86Hz, etc…

Oversampled 2nd Order Filters

We can cure the HF problems with these two filters the same way as before, by using oversampling techniques. Using 4X oversampling with the same setup as before (same anti-imaging filter) we can get a closer match to the analog responses. In Figure 4.7a I analyze a 2nd Order LPF fc = 1kHz Q = 5 and we observe much better results. In Figure 4.8a we have the unity gain BPF with fc = 1kHz Q = 0.5 and again we observe the benefits of oversampling for very high frequencies.

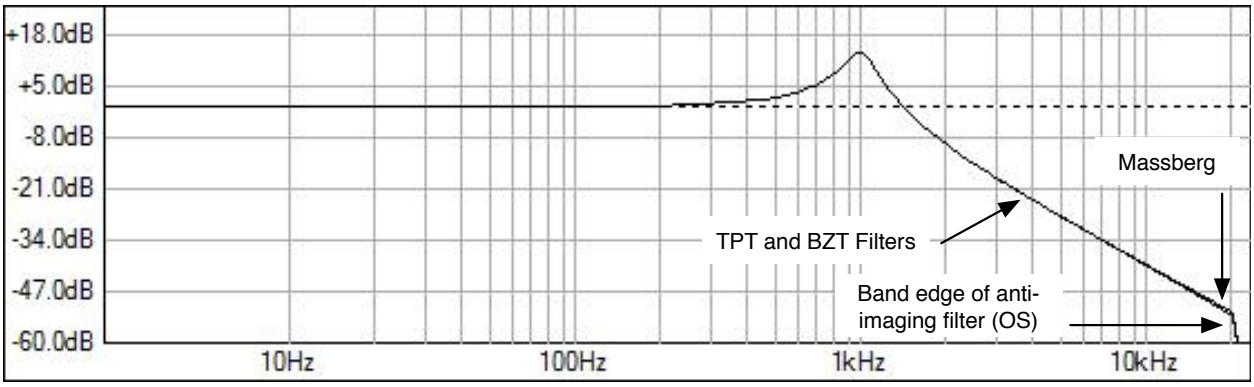


Figure 4.7a: With 4X Oversampling the HF deviations are removed for the LPF. You can see the Massberg filter has a slight deviation at the highest frequencies; I believe this is actually the error in the Massberg filter (it is based on a shelving filter)

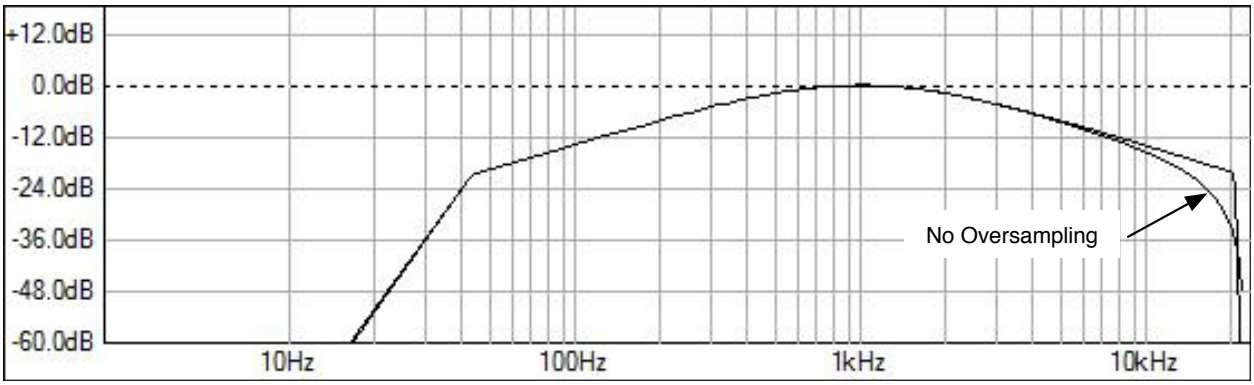


Figure 4.8a: Oversampling 4X with the unity gain BPF (as before the LF error is due to the FFT)

Figures 4.9 - 4.13 show the results for the other filters (no oversampling really needed for them for HF resolution). The results match the VA Book plots.

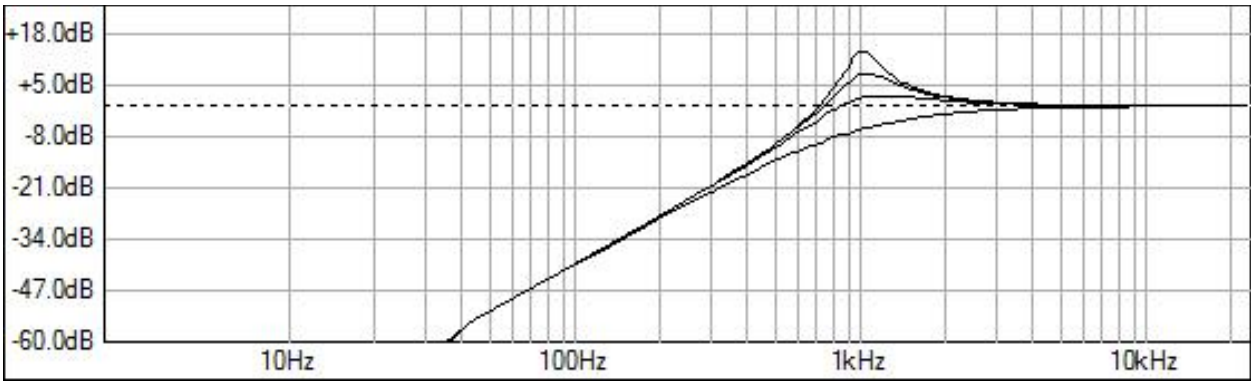


Figure 4.9: HPF responses for Q = 0.5, 1.0, 2.5 and 10

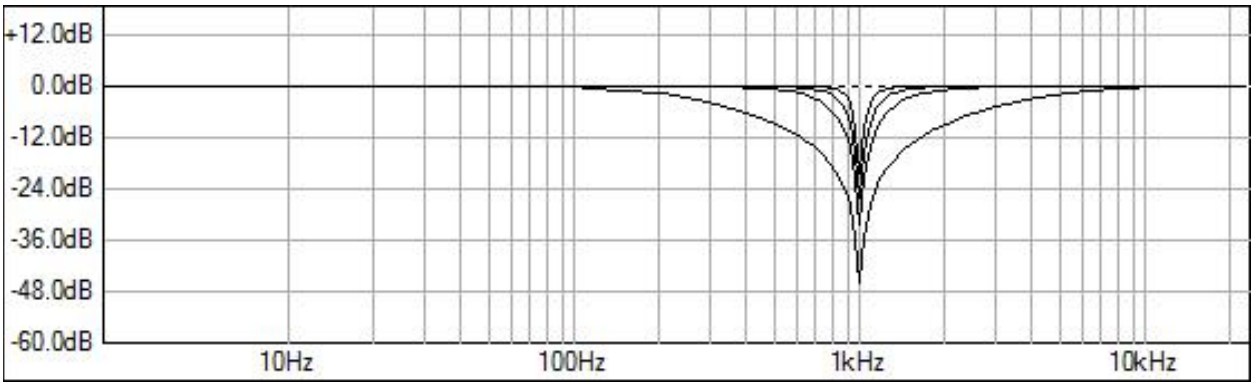


Figure 4.10: Notch responses for Q = 0.5, 1.0, 2.5 and 10

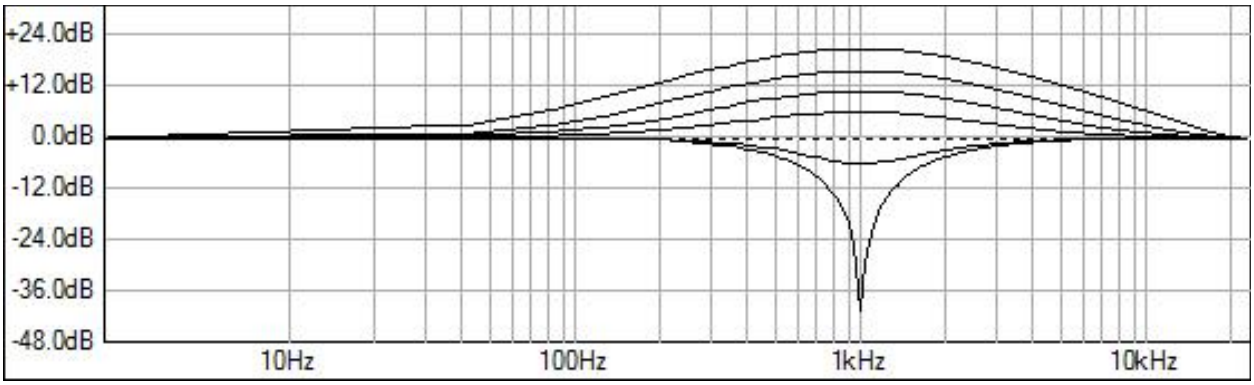


Figure 4.11: The Band-shelving responses for K= -1.0, -0.5, 1.0, 2.5 and 10 with Q held at 1.0

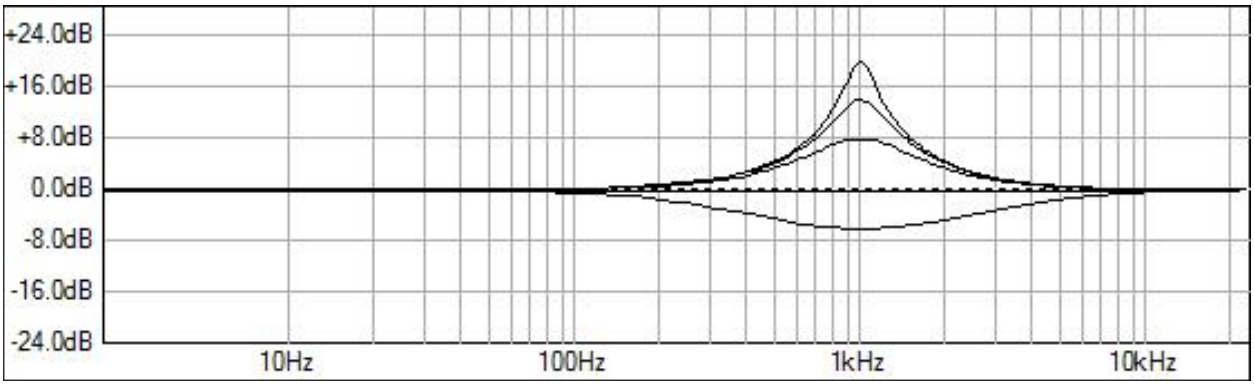


Figure 4.12: Peaking filter with Q = 0.5, 1.0, 2.5, 5 and 10

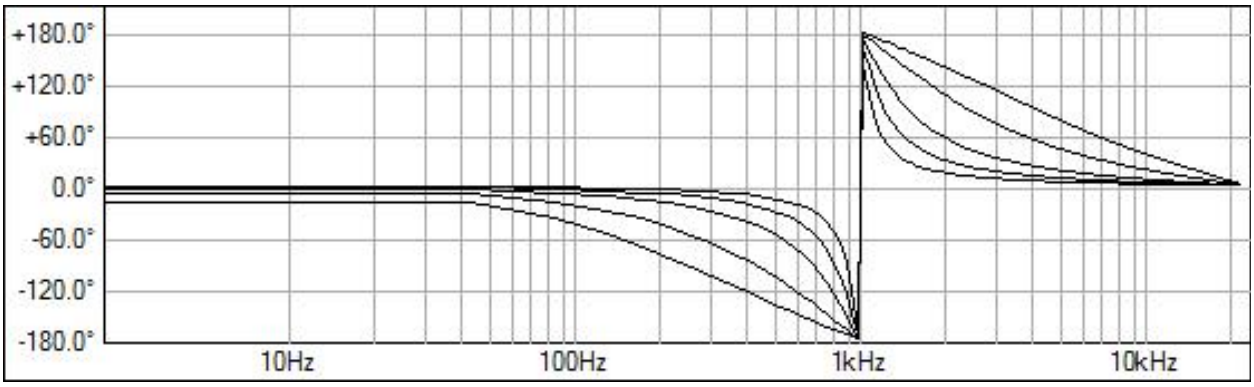


Figure 4.13: APF Phase Responses for Q = 0.5, 1.0, 2.5, 5 and 10

Here is the code listing for the TPT SVF, left channel only. The highpass output is calculated first and placed into HP[0]; the BPF is the output of the first integrator while the LPF is the output of the second. After that the storage registers are updated. Then, all the other filter types are calculated using the simple equations above.

```
// pre-warp the cutoff- these are bilinear-transform filters
float wd = 2*pi*m_fFc;
float T  = 1/(float)m_nSampleRate;
float wa = (2/T)*tan(wd*T/2);
float g  = wa*T/2;

float xn = pInput[0];

// Zavalishin R = 1/2Q in his VA book
float R = 1.0/(2.0*m_fQ);

// two channel output buffers
float HP[2];
float BP[2];
float LP[2];
```

```
// do left channel
HP[0] = (xn - (2.0*R + g)*m_fZ1[0] - m_fZ2[0])/(1.0 + 2.0*R*g + g*g);
BP[0] = g*HP[0] + m_fZ1[0];
LP[0] = g*BP[0] +  m_fZ2[0];

// z1 register update
m_fZ1[0] = g*HP[0] + BP[0];
m_fZ2[0] = g*BP[0] + LP[0];
```

Moog Ladder Filter

The Moog Ladder Filter consists of four 1st order LPFs in series all set with the same cutoff frequency (called *synchronously tuned* or *sync-tuned*). It is an ingenious design; the resonance is created by subtracting the output from the input. It works because a 4th order LPF has a phase shift of 180 degrees at the cutoff frequency. Inverting this and adding it back to the input swings this frequency back in phase creating resonance at that frequency. It also drops the gain at low frequencies below the cutoff where the phase is not re-inverted. The higher the Q, the more the low frequencies are attenuated. You can see from Figure 4.14 that there is indeed a delay-less feedback loop making this a nice candidate for the TPT filter implementation.

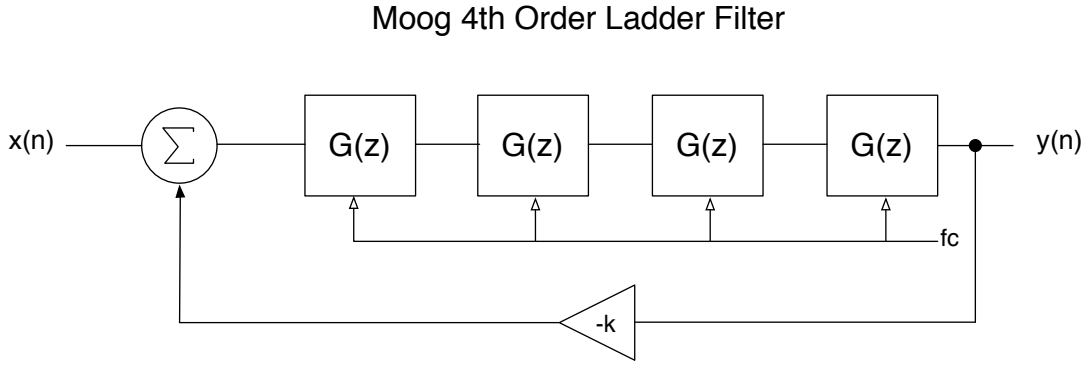


Figure 4.14: the Moog Ladder Structure

Stilson-Smith Ladder Filter

You can find the code and details for a Bonus Project on my website that implements the Stilson-Smith Moog Ladder Filter in Figure 4.15 (hereafter referred to as the *Stilson Ladder Filter* or simply *Stilson*). It has certainly been criticized in the past because it uses a delay element in the feedback path which a) doesn't preserve the delay-less topology and b) results in a 5-pole filter.

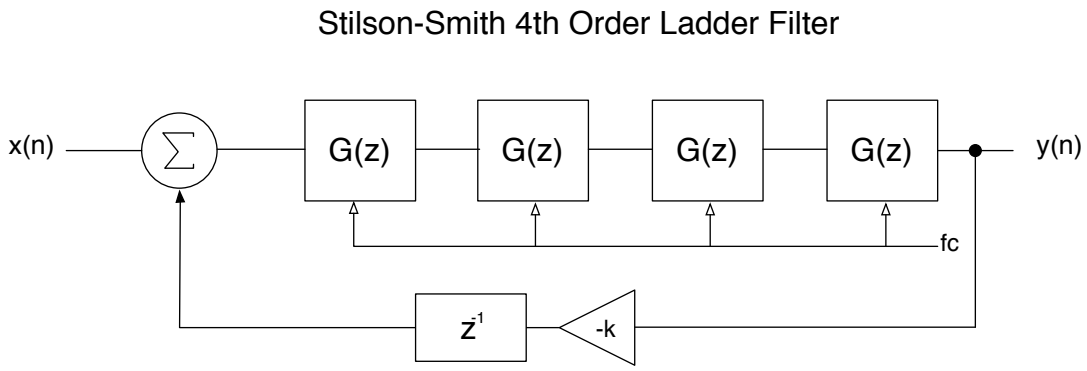


Figure 4.15: the Stilson-Smith implementation uses a delay in the feedback path

The Stilson version also has problems mapping the cut-off frequency correctly (linearly with the control). It is designed with the bilinear transform, but then multiple corrections are applied to try to get it to match the original's frequency response. It obtains a sort of hybrid response which while not perfect does operate nearly properly at very high frequencies. See the original paper for more details.

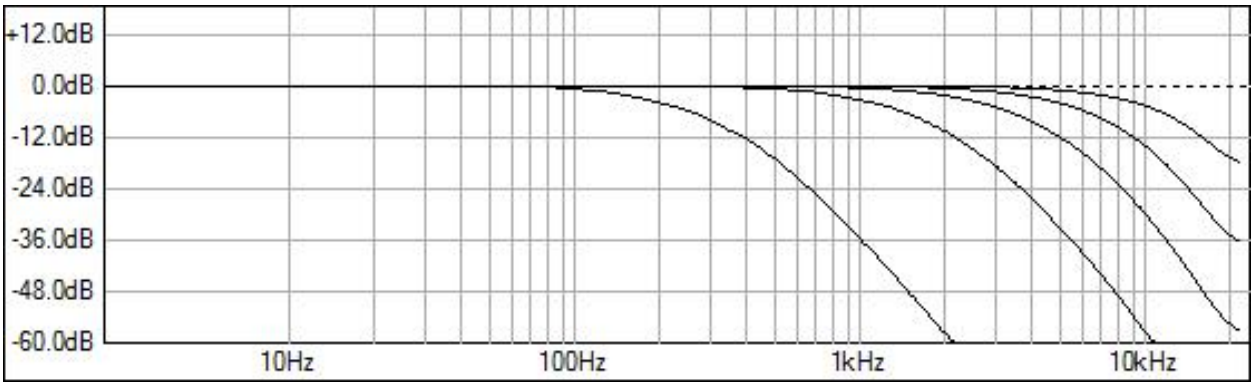


Figure 4.16: The Stilson filter's responses with Q = 0.707 and a variety of fc settings: 1kHz, 5kHz, 10kHz, 15kHz and 20kHz

Figure 4.16 shows the Stilson filter with Q = 0.707 and a variety of fc settings and we can make several observations; the cutoff frequency doesn't match up linearly with the control, *however* we also observe a very analog-like response for very high frequencies. The gains at Nyquist do not fall to zero but rather have finite values.

Cascaded and synchronously tuned filters have a problem called Bandwidth Shrinkage (see Lindquist's *Active Network Design with Signal Filtering Applications* book, page 100 for discussion and shrinkage factor table). When you cascade four analog first order filters together and tune them to the same cutoff fc, the resulting 4th order filter will have a shrunken bandwidth and the cutoff will theoretically be 0.435fc and the Stilson filter suffers from this but the whole thing is compounded by the correction factors and 5th pole so that we don't attempt to get a perfect match with the control (it can be argued that this is justified in a musical filter application; rarely do performers actually care about the exact frequencies as they are twisting the knobs *but* they may value a linear control range).

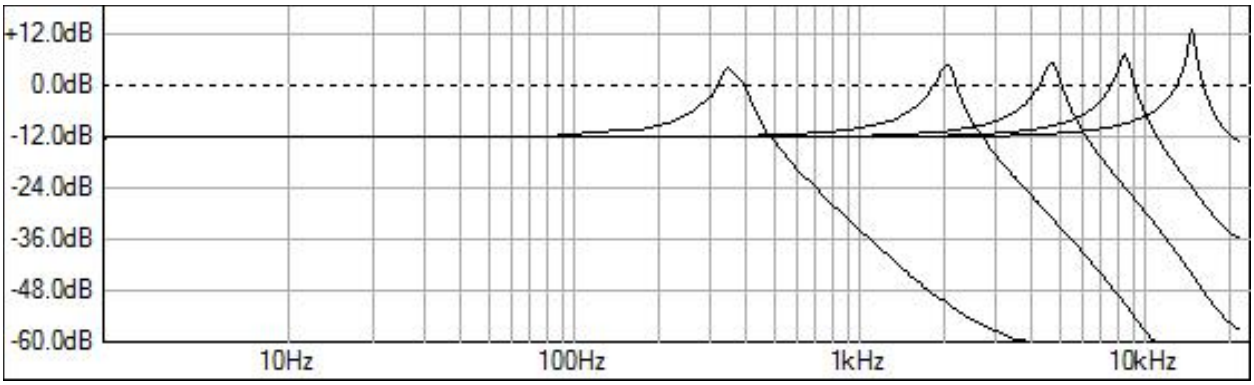


Figure 4.17: the Stilson filter with Q = 20 and a variety of fc settings: 1kHz, 5kHz, 10kHz, 15kHz and 20kHz

Figure 4.17 shows the same set of cutoffs as Figure 4.15 but with Q = 20 and we observe the resonant peaking as well as the reduction in LF gain of about -12dB. We also observe the non-zero gains at Nyquist as well as a slight rise in the resonant peaking as the fc is increased. See the original paper for more plots and discussions errors in the filter. Finally, Figure 4.18 shows the expected change in LF gain with changes in resonance (Q).

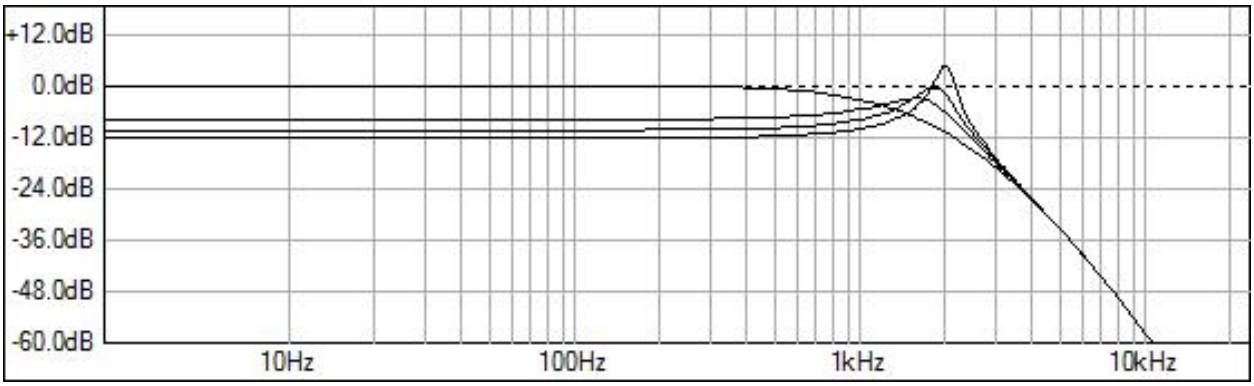


Figure 4.18: the LF gain drops as the Q increases

TPT Ladder Filter

Zavalishin's TPT Ladder Filter is implemented using four cascaded TPT 1-pole LPF filters as previously designed above. I made a C++ object that implements a TPT 1-pole LPF and then sync tuned and cascaded four of them together for my RackAFX implementation. These are going to have the same sync-tuning issue with bandwidth shrinkage. You also can not easily apply a correction factor of 0.435 because of the bilinear warping so no effort is made to get the frequency control to linearly match the actual response. Figure 4.19 shows the filter topology.

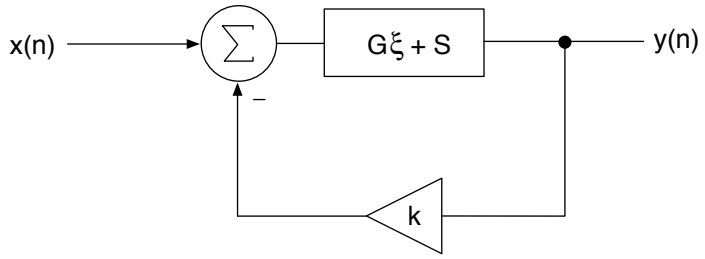


Figure 4.19: the TPT Ladder Filter topology; the details are found in the VA book, p. 61; u(n) is the output of the summer

Zavalishin's analysis of the zero-delay loop and four cascaded LPFs provides us with the input to the first LPF, u(n).

$$u(n) = \frac{x(n) - kS}{1 + kG}$$

$$G = g^4$$

$$S = g^3 s_1(n) + g^2 s_2(n) + g s_3(n) + s_4(n)$$

After calculation, u(n) is fed into the first TPT object and passed through the line of four of them. Figure 4.20 shows the block diagram.

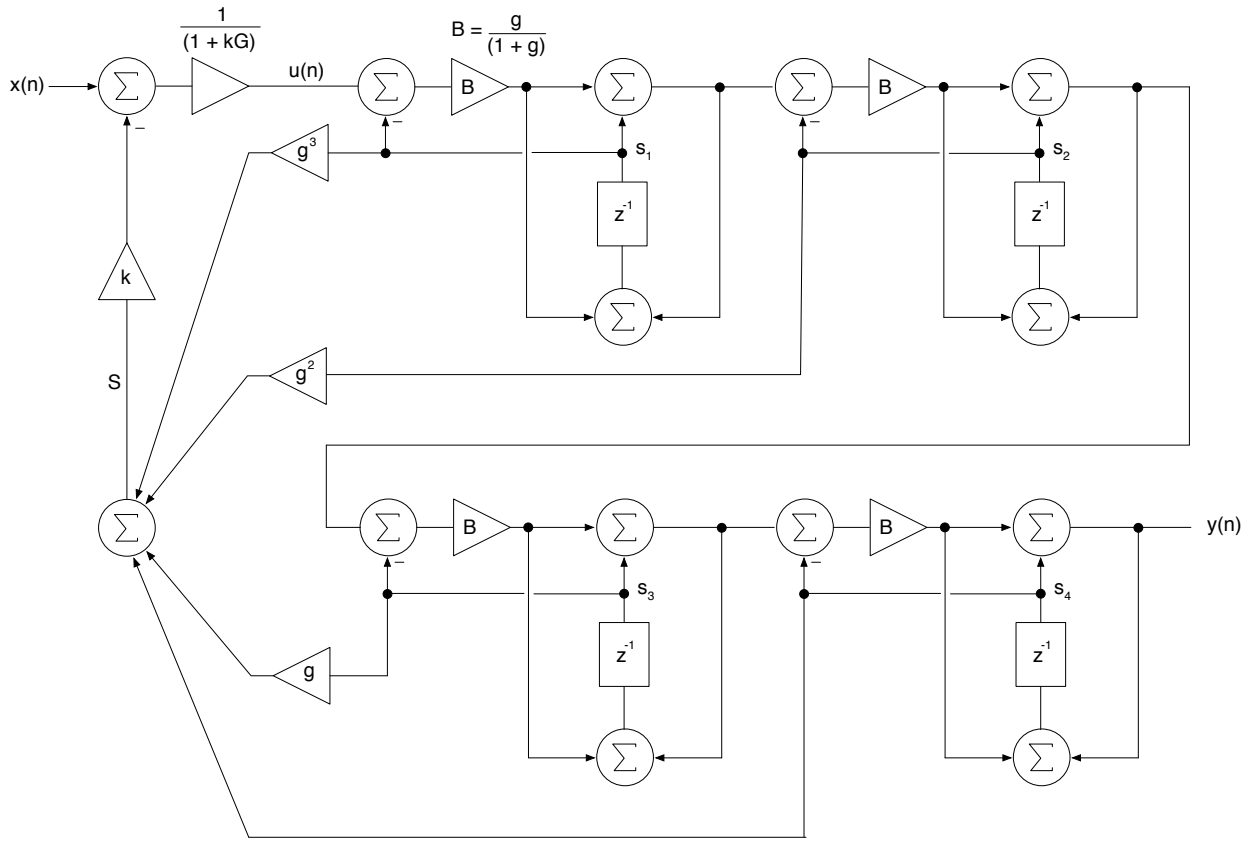


Figure 4.20: the TPT Ladder Filter block diagram; note the location of u(n) which feeds the first LPF. The location of S is also shown (the output of the summer-of-s's)

Figure 4.21 shows the TPT Ladder Filter (now referred to as just *TPT Filter*) with the Q held at 0.707 and varying the cutoff frequency.

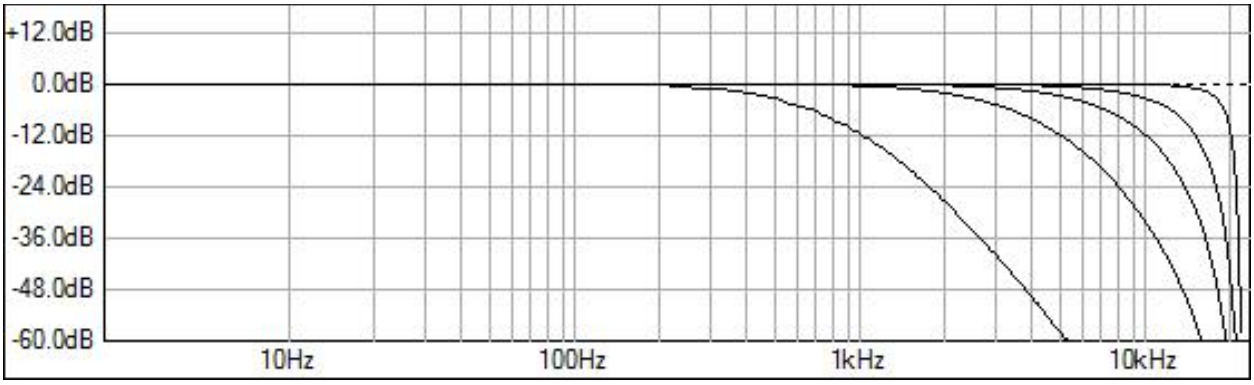


Figure 4.21: the TPT Ladder filter's responses with Q = 0.707 and a variety of fc settings: 1kHz, 5kHz, 10kHz, 15kHz and 20kHz

The effect of the bilinear transform is apparent; the zero at Nyquist pulls all the responses downward at very high frequencies. We also observe bandwidth shrinkage, however it is much better behaved compared to the Stilson filter (the fc values lie right on the -12dB line across the board). At low cutoff frequencies the zeros at Nyquist pose no real problems and the responses look good. However, as the

cutoff frequency rises and becomes very high, the zeros do have an effect and squash the response, even with the inverted feedback trying to pull it back up at the resonant frequency.

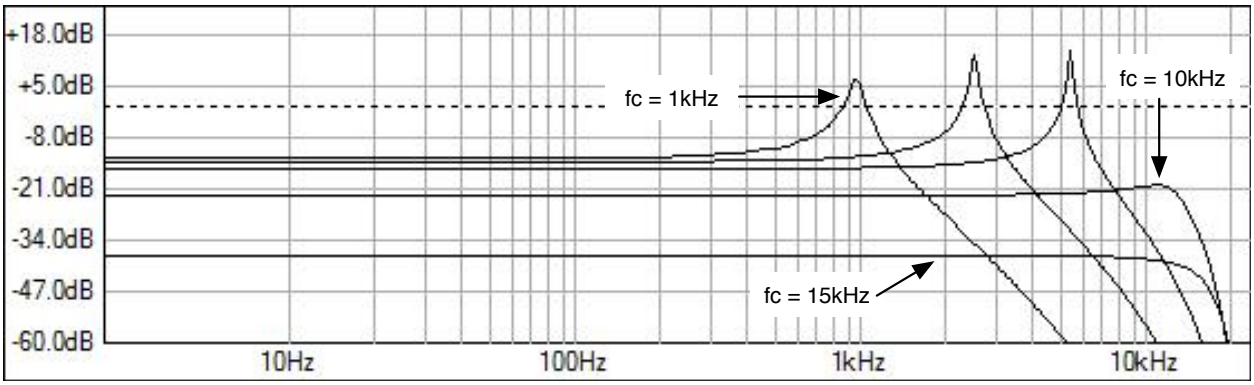


Figure 4.22: the responses for Q = 20 and fc = 1kHz, 2.5kHz, 5kHz, 10kHz, 15kHz and (not visible because it is below -60dB) 20kHz

In Figure 4.22 we observe some interesting details. The resonant peak frequency is close for low frequencies and only a small error is introduced as fc increases. We also observe the slight increase in resonant peaking as fc increases. Most obvious however is the failure of the filter as fc goes above about 9kHz or so. Those zeros at Nyquist are in effect in addition to throwing the low frequencies out of phase with the input. On the other hand, it could certainly be argued that for musical applications we might not need to have fc get much higher than 10kHz or so.

Trying to compare the Stilson and TPT Ladder filters is tricky because of the lack of linear control of cutoff frequency. However, I did adjust the two filters to try to match the resonant peaking and resonant frequency as closely as possible (this is by no means an all inclusive comparison). The results are shown in Figure 4.23.

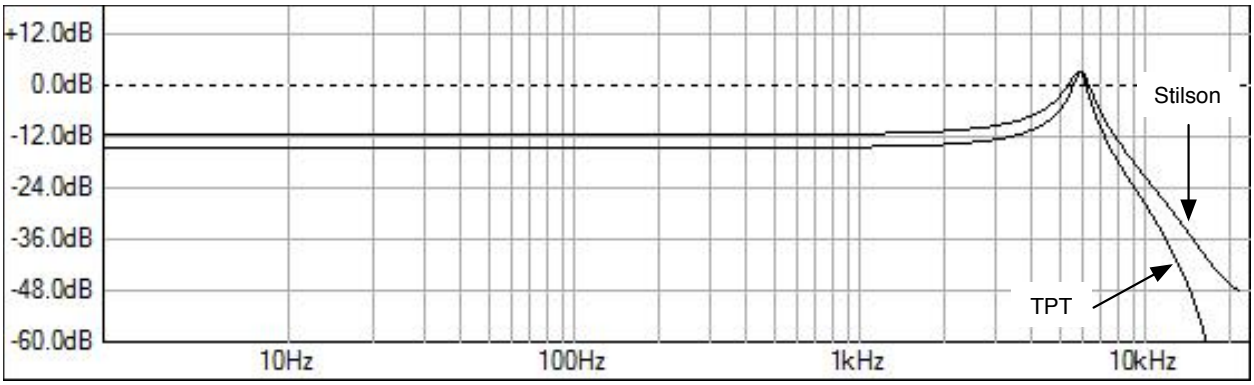


Figure 4.23: comparing the TPT and Stilson Ladder filters while holding resonant peaking and frequency constant shows differences in both LF gain reduction (TPT has slightly more) and HF response (Stilton has finite gain at Nyquist).

Oversampled TPT Ladder Filter

Once again it's oversampling to the rescue here as we can resolve the HF problems that way. The same 4X Oversampling is used as before with the same anti-imaging filters. Figure 4.21a shows the improvements in HF responses for the case of Q = 0.707 and varying the cutoff. Again we observe the increase in HF gain (except of course the anti-imaging band edge). Figure 4.22a shows the same set of curves with Q = 20. Everything looks excellent save for the fc = 20kHz case; it is being brought down by the oversampling anti-imaging filter's cutoff (the band edge is 20kHz-22kHz).

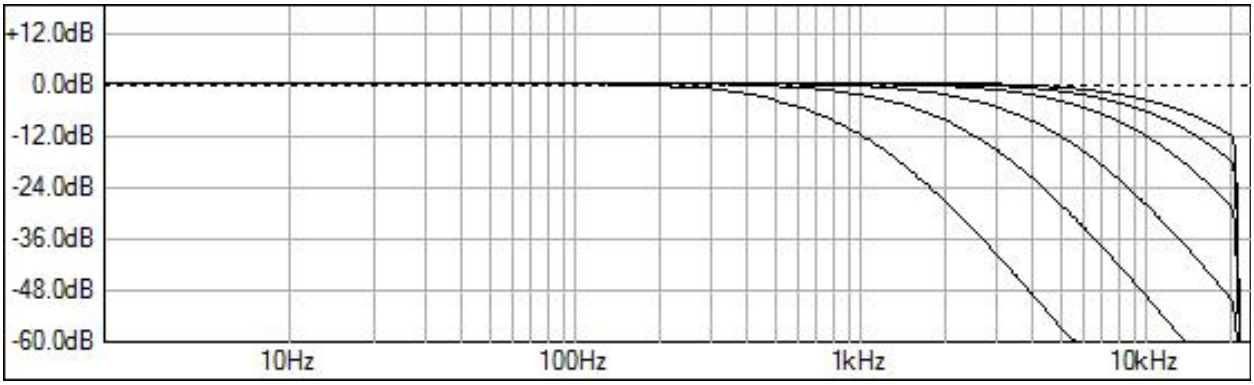


Figure 4.21a: Oversampling 4X with Q = 0.707 and fc = 1kHz, 2.5kHz, 5kHz, 10kHz, 15kHz and 20kHz

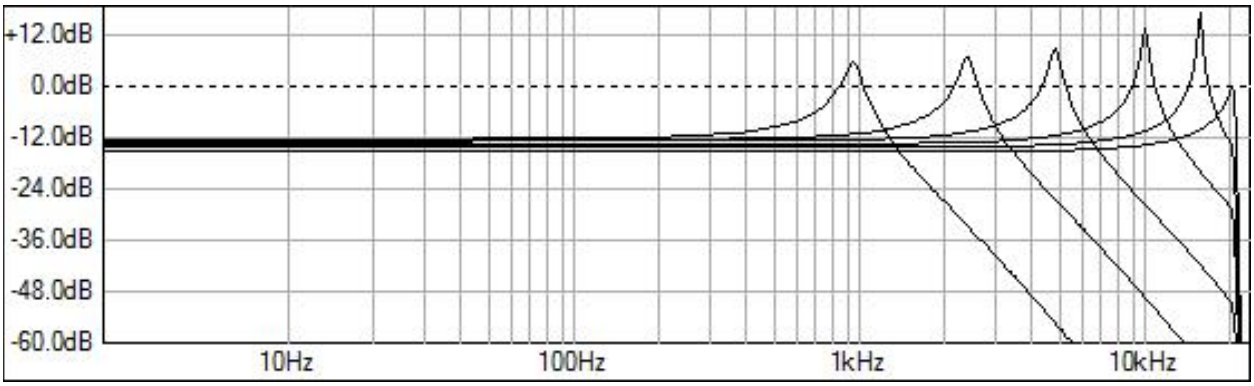


Figure 4.22a: Oversampling with Q = 20 and fc = 1kHz, 2.5kHz, 5kHz, 10kHz, 15kHz and 20kHz

Each TPT 1-pole LPF is implemented in a C++ class CTPTMoogFilterStage:

```
class CTPTMoogFilterStage
{
public:
	CTPTMoogFilterStage(void);
	~CTPTMoogFilterStage(void);

protected:
	// controls
	float G;				// cutoff
	float scalar;			// scalar value
	float sampleRate;		// fs
	float z1;				// z-1 storage location

public:
	inline void initialize(float newSampleRate)
	{
		// save
		sampleRate = newSampleRate;
		z1 = 0;
	}

	void setFc(float fc);
	float doFilterStage(float xn);
	float getSampleRate(){return sampleRate;}
	float getStorageRegisterValue(){return z1;}
};
```

Notice I provide a function to read the Storage Register value; this is needed to create our sum-of-s's in the implementation. The setFc() pre-warps the cutoff and calculates the big G value - this is identical to the implementation of the single LPF code above. The doFilterStage() method is simple and straight out of the book:

```
void CTPTMoogFilterStage::setFc(float fc)
{
	// prewarp the cutoff- these are bilinear-transform filters
	float wd = 2*pi*fc;
	float T  = 1/sampleRate;
	float wa = (2/T)*tan(wd*T/2);
	float g  = wa*T/2;

	// calculate big G value; see Zavalishin p46 The Art of VA Design
	G = g/(1.0 + g);
}
// TPT 1 pole filter; see Zavalishin p46 The Art of VA Design
float CTPTMoogFilterStage::doFilterStage(float xn)
{
	float v = (xn - z1)*G;
	float out = v + z1;
	z1 = out + v;

	return out;
}
```

The four first order filters are cascaded and implemented by another C++ object CTPTLadderFilter:

```
class CTPTMoogLadderFilter
{
public:
	CTPTMoogLadderFilter(void);
	~CTPTMoogLadderFilter(void);

protected:
	CTPTMoogFilterStage filter1;
	CTPTMoogFilterStage filter2;
	CTPTMoogFilterStage filter3;
	CTPTMoogFilterStage filter4;

	float k; // Q control
	float fc; // fc control

public:
	inline void initialize(float newSampleRate)
	{
		filter1.initialize(newSampleRate);
		filter2.initialize(newSampleRate);
		filter3.initialize(newSampleRate);
		filter4.initialize(newSampleRate);
	}

	inline void calculateTPTCoeffs(float cutoff, float Q)
	{
		// 4 sync-tuned filters
		filter1.setFc(cutoff);
		filter2.setFc(cutoff);
		filter3.setFc(cutoff);
		filter4.setFc(cutoff);

		// NOTE: Q is limited to 20 on the UI to prevent blowing up
		// Q = 0.707 -> 25 ==> k = 0->4
		k = 4.0*(Q - 0.707)/(25.0 - 0.707);

		// ours
		fc = cutoff;
	}

	float doTPTMoogLPF(float xn);
};
```

The doTPTMoogLPF function finishes the implementation:

```
float CTPTMoogLadderFilter::doTPTMoogLPF(float xn)
{
	// calculate g
	float wd = 2*pi*fc;
	float T  = 1/(float)filter1.getSampleRate();
	float wa = (2/T)*tan(wd*T/2);
```

```
	float g  = wa*T/2;

	float G = g*g*g*g;
	float S = g*g*g*filter1.getStorageRegisterValue() +
	          g*g*filter2.getStorageRegisterValue() +
	          g*filter3.getStorageRegisterValue() +
	          filter4.getStorageRegisterValue();

	// u is input to filters, straight from book
	float u = (xn - k*S)/(1 + k*G);

	// four cascades using nested functions
	float filterOut = filter4.doFilterStage(filter3.doFilterStage
	                  (filter2.doFilterStage(filter1.doFilterStage(u))));

	// output
	return filterOut;
}
```

NOTE: both ladder filters can become unstable at high Q values. With Q = 25 on the RackAFX plug-in, k = 4 and the filter will be on the verge of blowing up. High cutoff frequencies in combination with high Q can also cause failure. In the Plug-In example, the Stilson parameters are bounded to prevent blowing up (this is from the original Bonus Project), however I left the TPT filter unbounded, so you can blow it up if you want to. If you do this in RackAFX, you can reset the filter (after lowering the fc and/or Q controls) with the Reset toolbar button or menu item. It will call prepareForPlay() which is usually where you reset everything.

Coda

I designed and implemented many of the filters from the VA book and analyzed their frequency responses (you can also analyze their impulse, phase and step responses with the RackAFX Analyzer). We observe that the frequency responses are for all practical purposes identical to the BZT-biquad versions and therefore suffer from incorrectly mapped zeros at z = Nyquist. However:

- TPT Filters preserve the analog topology (see the analog SVF circuit) including delay-less loops.
- TPT Filters are less computationally expensive in processing the audio, updating the storage registers and calculating the filter coefficients.
- TPT Filters do not suffer from the same issues with direct-form implementations; whether these issues are audible or pose problems in different applications is not addressed here.
- TPT Filters still have the bilinear frequency mapping with HF errors from the analog versions when the analog filters have zeros at infinity.
- The HF errors can be mitigated by oversampling techniques.
- The oversampling will add overhead to the CPU usage, however.
- The Massberg Filters approximate the HF analog responses well without oversampling but their coefficient calculations are not trivial and they use the direct form implementation
- The Stilson Ladder Filter also approximates the HF analog response well without oversampling but it has a delay in the feedback path making it really a 5-Pole Filter and uses a direct form structure for its sub-filters.
- Both the TPT and Stilson Ladder filters can become unstable at high cutoffs and Q values.

References

Lindquist, Claude. 1989. *Adaptive and Digital Signal Processing with Digital Signal Filtering Applications*, Chap. 1,5. Miami: Steward and Sons.

Lindquist, Claude. 1977. *Active Network Design with Signal Filtering Applications*, Chap. 2. Long Beach: Steward and Sons.

Pirkle, Will. 2012. *Designing Audio Effect Plug-Ins in C++*, Burlington: Focal Press.

Stilson, Tim., Smith, Julius O. *Analyzing the Moog VCF with Considerations for Digital Implementation*, Proceedings of the International Computer Music Conference, Computer Music Association, 1996, Hong Kong.

Zavalishin, Vadim. 2012. *The Art of VA Filter Design*, published electronically