Background
This brief App Note accompanies the projects for the paper *Novel Hybrid Virtual Analog Filters Based on the Sallen-Key Architecture*, Paper 9195 presented on October 12, 2014. The full background theory for all the filters is fully documented in the paper; this App Note shows the filter models and code.

The following filters are included:
- resonant first order LPF and HPF
- resonant first order low and high shelving filters (LSF, HSF)
- second order filters with resonant frequency and cutoff frequency decoupled and independently modulate-able and controllable

You can now synthesize multiply-resonant filters:
- second order doubly resonant lowpass filter (LPF) with two resonant peaks that are independently modulate-able and controllable

Conceptual Block Diagram
The paper’s filters are all based on the conceptual block diagram for the Sallen-Key filter, shown in Figure 10.1. In this case, the filter is a second order lowpass filter.

![Conceptual Block Diagram](image)

Figure 10.1: the conceptual block diagram for the Sallen-Key lowpass filter and it’s signal flow graph

In these filters, the second order bandpass filter in the feedback path always stays the same. Its center frequency controls the resonant frequency of the final filter. The amplifier K controls the Q of the filter, from none to self-oscillation at K = 2. See the AES paper for all the details. Using this model and replacing the input filter (LPF2) with various other filter types, we can directly synthesize interesting filters with unique qualities. Table 10.1 lists the filter combinations. Notice that since the loop bandpass filter controls the resonant frequency and the input filter controls the cutoff frequency, this model de-couples the controls making them independently adjustable.
Table 10.1: the input/loop filter combinations and the resulting filters

<table>
<thead>
<tr>
<th>INPUT FILTER</th>
<th>LOOP FILTER</th>
<th>RESULTING FILTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st order LPF</td>
<td>2nd Order Bandpass</td>
<td>Resonant 1st order LPF</td>
</tr>
<tr>
<td>1st order HPF</td>
<td>2nd Order Bandpass</td>
<td>Resonant 1st order HPF</td>
</tr>
<tr>
<td>1st order LSF</td>
<td>2nd Order Bandpass</td>
<td>Resonant 1st order LSF</td>
</tr>
<tr>
<td>1st order HSF</td>
<td>2nd Order Bandpass</td>
<td>Resonant 1st order HSF</td>
</tr>
<tr>
<td>2nd order LPF</td>
<td>2nd Order Bandpass</td>
<td>Resonant 2nd order LPF</td>
</tr>
<tr>
<td>2nd order HPF</td>
<td>2nd Order Bandpass</td>
<td>Resonant 2nd order HPF</td>
</tr>
</tbody>
</table>

Filter Structures

As noted in the paper, any filter structures that may be fashioned into a linear combination of the current input sample (scaled or not) plus the sum of the past samples (scaled or not) may be used. Using Zavalishin’s notation, this is \( y(n) = Gx(n) + S(n) \) where \( S(n) \) represents the storage component of past samples. You may use DF1, DF2, Transformed DF, Trapezoidal or any other structure that adheres to the linear equation. For simplicity’s sake, we used the old-fashioned DF1 structure that all students are familiar with. We encourage you to use other structures and perform listening tests on the resulting filters. Figure 10.2 shows the DF1 form (aka biquad) structure modified to produce the required input + past samples. The simplified equivalent diagram is on the right; place whatever filter structure you like inside it.

![Diagram](image)

Figure 10.2: (a) the biquad structure with an added output port for the \( S(n) \) value and (b) the simplified block diagram equivalent; the \( S \)-port may be relocated to the top of the filter block diagram if needed

Using the simplified block form of the filter, and using either the Modified Härmä method (AES Paper 9194) or Zavalishin’s algebraic method to resolve the delay-free loop, the Sallen-Key structure in Figure 10.1 can be redrawn as in Figure 10.3.
Figure 10.3: the generalized Sallen-Key topology; the loop filter BPF2 is always a 2nd order BPF with a midband gain $H_0 = 0.5$ while the Input Filter varies in type

The modified DF1 structure is implemented in a C++ class called CBiquadEx, a modified version of CBiQuad that ships with RackAFX. The code simply separates the previous/current sample values and is modified for the shelving filters with the same structure as in *Designing Audio Effects Plug-ins in C++* page 189, Fig 6.27.

```cpp
class CBiquadEx : public CBiQuad
{
public:
    CBiquadEx(void);
    ~CBiquadEx(void);

    float m_f_c0;
    float m_f_d0;

    inline float doBiQuad(float f_xn)
    {
        // just do the difference equation: $y(n) = a0x(n) + a1x(n-1) +$
        // $a2x(n-2) - b1y(n-1) - b2y(n-2)$
        float yn = m_f_a0*f_xn + m_f_a1*m_f_Xz_1 + m_f_a2*m_f_Xz_2 -
                    m_f_b1*m_f_Yz_1 - m_f_b2*m_f_Yz_2;

        // underflow check
        if(yn > 0.0 && yn < FLT_MIN_PLUS) yn = 0;
        if(yn < 0.0 && yn > FLT_MIN_MINUS) yn = 0;

        // shuffle delays
        // Y delays
        m_f_Yz_2 = m_f_Yz_1;
        m_f_Yz_1 = yn;

        // X delays
        m_f_Xz_2 = m_f_Xz_1;
        m_f_Xz_1 = f_xn;

        return yn*m_f_c0 + f_xn*m_f_d0;
    }

    inline float getS_Sum()
    {
        // ...
    }
};
```
The processing code, which is the same for all filters except the doubly-resonant one, is shown below. It simply follows the block diagram of Figure 10.3. The processing contains a nonlinear tanh() block in case you want to experiment with this - it eases the harshness at high-gains before oscillation. Notice the loop filter’s midband gain is 0.5, which is also explained in the AES paper.

```c
{   
    return m_f_Xz_1*m_f_a1 + m_f_Xz_2*m_f_a2 -
    m_f_Yz_1*m_f_b1 - m_f_Yz_2*m_f_b2;
}
```

Results
Figures 10.4, 10.5 and 10.6 show some plots of the resulting filter responses taken from the RackAFX audio FFT analyzer. These figure also show the ability to decouple the resonant and cutoff frequencies.
Figure 10.4: first order hybrid lowpass filter with $f_c = 1$kHz, $K = 1.99$ and (a) resonant frequency equals cutoff frequency, (b) $f_r = 100$Hz and (c) $f_r = 30$Hz; the resonant frequency is lower than the cutoff frequency, (d) $f_r = 10$kkHz; the resonant frequency is higher than the cutoff frequency.

Figure 10.5: first order hybrid highpass filter with $f_c = 1$kHz, $K = 1.99$ and (a) resonant frequency equals cutoff frequency, (b) $f_r = 10$kkHz; the resonant frequency is higher than the cutoff frequency, (c) $f_r = 500$Hz and (d) $f_r = 100$Hz; the resonant frequency is lower than the cutoff frequency.
Doubly-Resonant Filters

By adding a second bandpass filter into the structure, we can obtain a doubly-resonant filter with a response like that in Figure 10.7. The structure is shown in Figure 10.8. As described in the paper, this filter has two frequency controls: one for the cutoff/normal resonant frequency, and another for the second resonant peak (aka the twin-peak). You could apply modulators to each peak and move them back and forth across one another. Other variations also exist. You can also modify the project code so that you can “park” the twin-peak at a certain distance (musically, perhaps octaves or octaves+fifth) and then apply one modulation that moves both peaks together. How does it sound? Well, it sound just like a doubly-resonant lowpass filter. At self-oscillation, both resonant frequencies can be heard. If one of the peaks is on the roll-off portion of the curve, it’s oscillation is similarly reduced in amplitude by the same attenuation amount.
Figure 10.7: actual frequency response of the doubly-resonant lowpass filter with coupled cutoff and main resonant frequency and variable second resonant frequency

Notice that a 5th order correction curve is applied when the two peaks are near each other. This is fully explained in the AES paper, along with notes on implementing more complex versions of this same filter.

Figure 10.8: the doubly resonant LPF block diagram
The code for the doubly-resonant filter is:

```cpp
// process input through LPF1
double y1 = m_InputFilter.doBiQuad(xn);

// get S2, S3
double S2 = m_BPF1BQ.getS_Sum();
double S3 = m_BPF2BQ.getS_Sum();

// --- form input to loop
double u = m_dAlpha*(y1 + S2 + S3);

// --- apply DR correction
float k = m_dK;

// --- DR correction with 5th order polynomial
if(m_dResonantFrequency > 0.4*m_dFc &&
   m_dResonantFrequency <= 1.8*m_dFc)
{
    double x = m_dResonantFrequency/m_dFc;
    double mult = 3.707479358*x*x*x*x - 23.53633376*x*x*x +
                  56.27733306*x*x - 61.71480829*x + 30.0208912*x -
                  4.261994245;
    k *= mult;
}

// --- form output
y = (k)*u;

// NAIVE NLP
if(m_uNLP)
  y = tanh(1.4*y);

// --- process through first loop filter, gain = 0.5
m_BPF1BQ.doBiQuad(0.5*y);

// --- process through second loop filter, gain = 0.5
m_BPF2BQ.doBiQuad(0.5*y);

// --- output scaling
if(k > 0)
  y *= 1/k;
```
Filter Equations:
For simplicity's sake, the very familiar bilinear-Z-transform DF1 versions of the filters are implemented. The equations for the shelving filters may be found in *Designing Audio Effects Plugins in C++*.

\[
\begin{align*}
\text{LPF1: } & \quad \theta_c = 2\pi f_c / f_s \\
& \quad \gamma = \frac{\cos \theta_c}{1 + \sin \theta_c} \\
& \quad a_0 = \frac{1 - \gamma}{2} \\
& \quad a_1 = \frac{1 + \gamma}{2} \\
\text{HPF1: } & \quad \theta_c = 2\pi f_c / f_s \\
& \quad \gamma = \frac{\cos \theta_c}{1 + \sin \theta_c} \\
& \quad a_0 = \frac{1 + \gamma}{2} \\
& \quad a_1 = \frac{1 - \gamma}{2} \\
\text{LPF2: } & \quad \theta_c = 2\pi f_c / f_s \\
& \quad \beta = 0.5 \cdot \frac{1 - \tan \left( \theta_c / 2Q \right)}{1 + \tan \left( \theta_c / 2Q \right)} \\
& \quad \gamma = (0.5 + \beta) \cos \theta_c \\
& \quad a_0 = 0.5 - \beta \\
& \quad a_1 = -0.5 \cdot \beta \\
\text{HPF2: } & \quad \theta_c = 2\pi f_c / f_s \\
& \quad \beta = 0.5 \cdot \frac{1 - \tan \left( \theta_c / 2Q \right)}{1 + \tan \left( \theta_c / 2Q \right)} \\
& \quad \gamma = (0.5 + \beta) \cos \theta_c \\
& \quad a_0 = 0.5 + \beta \\
& \quad a_1 = -0.5 + \beta \\
\text{BPF2: } & \quad \theta_c = 2\pi f_c / f_s \\
& \quad \beta = 0.5 \cdot \frac{1 - \tan \left( \theta_c / 2Q \right)}{1 + \tan \left( \theta_c / 2Q \right)} \\
& \quad \gamma = (0.5 + \beta) \cos \theta_c \\
& \quad a_0 = 0.5 + \beta + \gamma \\
& \quad a_1 = -0.5 + \beta + \gamma \\
& \quad a_2 = -0.5 - \beta \\
\end{align*}
\]

Revision History:
1.0: Initial Release, *December 8, 2014*

References:

Pirkle, Will, “Novel Hybrid Virtual Analog Filters Based on the Sallen-Key Architecture,” presented at the AES137th Convention, Los Angeles, USA, 2014 October 9-12. Paper 9194