

Multieffect effector

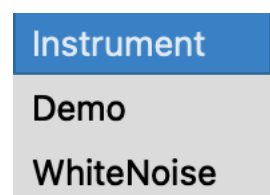
Our main goal is developing a wah-wah, phaser and flanger that work together. The instruction is shown in this link by video: <https://youtu.be/AH2qjShTcqW>

We decided to connect filters consistently, one by one. The original sound comes into Wah-Wah. Afterwards, the processed signal goes to the next filter - Phaser. The last filter that modifies the signal is Flanger.

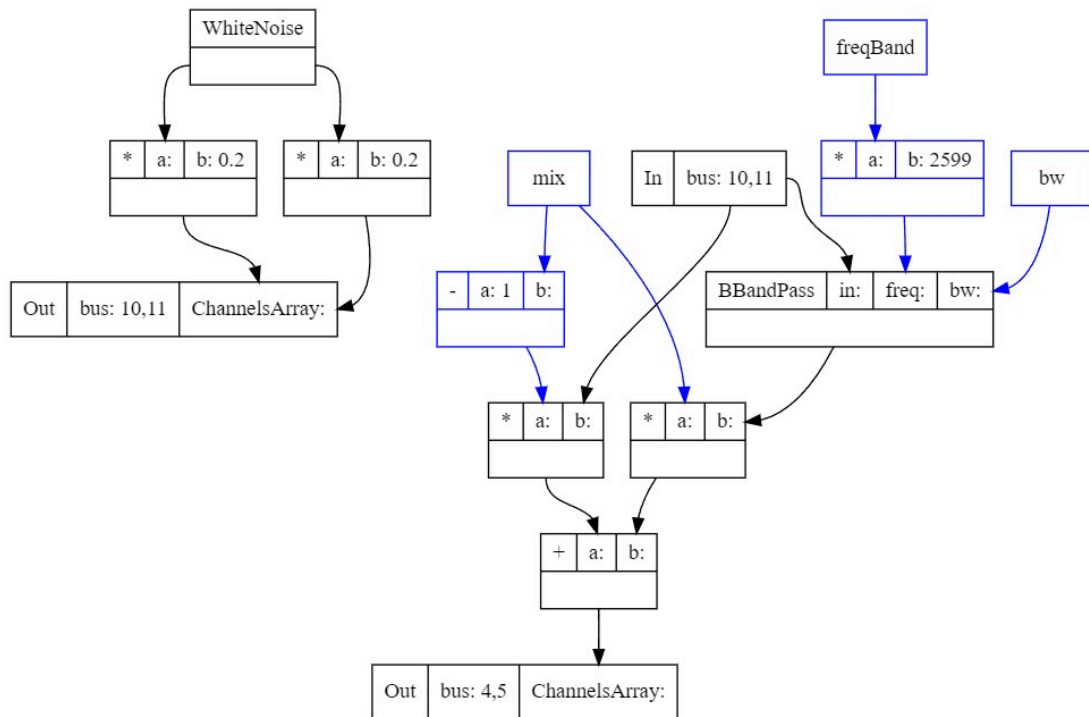
The application looks like this:



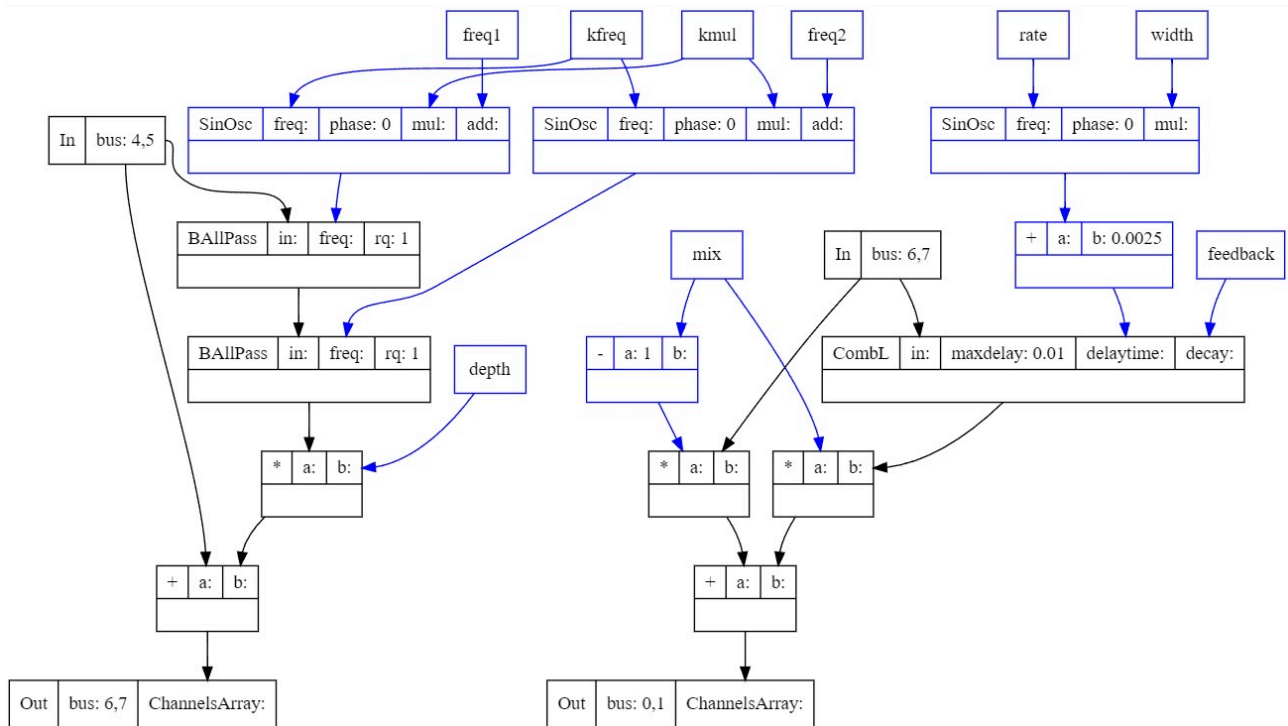
Our application has three regimes of working. First, the real-time filtering, users can plug in their instruments (e.g. guitar) and play with filters. The audio input is by default input channel 1 of the server. Secondly, the demo mode works with audio files loaded into the buffer. And the last one, WhiteNoise, helps to understand how filters work in a more efficient way.



Block schemes



WhiteNoise and Wah-Wah



Phaser and Flanger

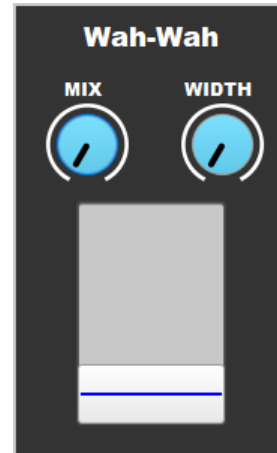
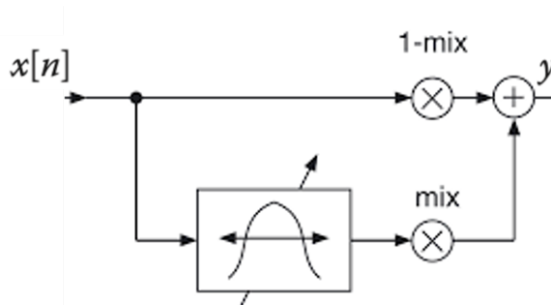
Filters

Wah-Wah

The wah-wah effect is produced mostly by foot-controlled signal processors containing a bandpass filter with variable center frequency and a small bandwidth. Moving the pedal back and forth changes the bandpass center frequency.

In our project, we created two knobs that control the mixing and multiplication effects, and one slider that controls frequency bandwidth.

```
sgn = BBandPass.ar(x_n, 2599*freqBand+300, 3*bw+0.05, mix);
```



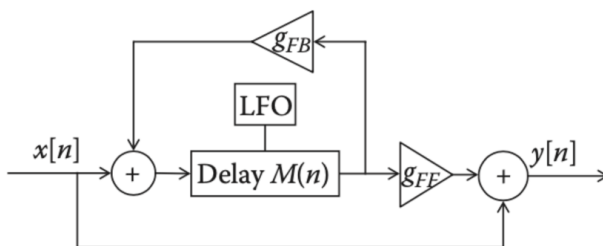
Flanger

Flanger mixes two identical signals together, one signal delayed by a small and (usually) gradually changing period, usually smaller than 20 milliseconds. This produces a swept comb filter effect: peaks and notches are produced in the resulting frequency spectrum, related to each other in a linear harmonic series. Varying time delay causes these to sweep up and down the frequency spectrum

Four knobs control: the mixing effect, rate (it controls LFO), width (it controls multiplication) and regen (or feedback that controls delay time).

```
lfo = SinOsc.kr(rate * 2, 0, 0.001 * width * 2);
```

```
delay = CombL.ar(inSig, 0.01, 0.0025 + lfo, 0.05 * feedback, 1, 0);
```



Phaser

Phasor will first create a duplicate of the dry signal. Then the duplicated signal is processed with a cascade of a series of All-pass filters. In the end, the phasor will add the processed signal back with the original one with a weight called depth.

- Algorithm

All-pass filters only change the phase response of the signal but not the amplitude, thus the phase is reversed at specific frequencies. Adding the phase-shifted signal back to the dry signal creates cancellations at frequencies (notched) where the phase of the two signals are 90-degree reversed. N All-pass filters (also called n stages) will create $n/2$ notches. The notch frequencies are modulated through LFO.

- Code

BAIIPass is a second-order all-pass filter. Note: AllpassL can not achieve the task, AllpassL is a Schroeder all-pass filter according to the documentation, this filter acts like a delay. It creates a Comb effect in the frequency domain, and its superclass is CombL which makes sense.

SelectX allows the user to use the if condition in a SynthDef. The output is selected from an array of inputs, plus it performs an equal power crossfade between two adjacent channels.

