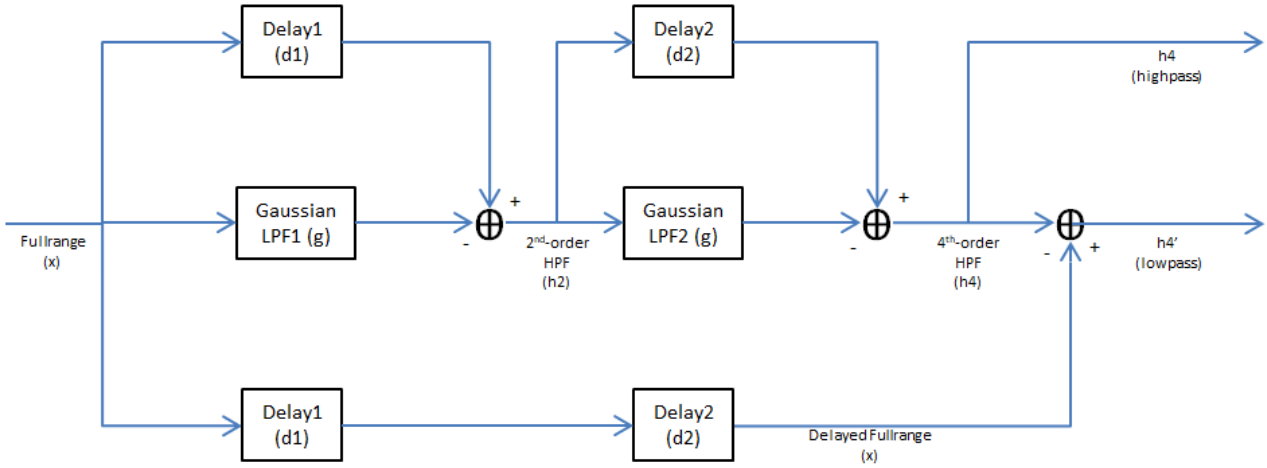
Matched-Delay Subtractive Crossover Pair With 4th-Order Highpass Slope

This is an implementation of the crossover filter pair described graphically in the figure below. That method starts with a standard Matched-Delay Subtractive highpass filter, passes it through another identical Matched-Delay Subtractive highpass filter, resulting in a 4th-order highpass filter, then derives the complementary lowpass filter from that highpass filter and the delayed fullrange input signal.



The lowpass filters used for LPF1 and LPF2 are typically, but not necessarily, Gaussian or Bessel.

In the nomenclature of the figure above:

$$h_4(t) = h_2(t - d_2) - h_2(t) * g$$
$$h_2(t) = x(t - d_1) - x(t) * g$$

where

$h_4(t)$ is the response of the final 4th-order HPF
$h'_4(t)$ is the response of the final complementary LPF (which will have rolloff characteristics similar to g)
g is the impulse response of the prototype Gaussian (or Bessel approximation) LPF
$x(t)$ is the input signal
$d = d_1 = d_2$ are pure delays equal to the delay through LPF g (empirically determined)
* denotes convolution in the time domain

To implement as shown requires:

- delay of $x(t)$ by d seconds (or equivalent number of samples)
- delay of $x(t)$ by $2d$ seconds (or equivalent number of samples)
- delay of $h_2(t)$ by d seconds (or equivalent number of samples)
- convolution of $x(t)$ with g
- convolution of $h_2(t)$ with g
- 3 subtractions

Totals:

- 1 tapped delay of $x(t)$, length $2d$
- 1 delay of $h_2(t)$, length d
- 3 subtractions
- 2 convolutions

Optimization:

$$\begin{aligned} h_4(t) &= [x(t-d_1-d_2) - x(t-d_2)*g] - [x(t-d_1) - x(t)*g]*g \\ &= x(t-d_1-d_2) - x(t-d_2)*g - x(t-d_1)*g + x(t)*g*g \\ &= x(t-2d) - 2x(t-d)*g + x(t)*g*g \\ &= x(t-2d) - [2x(t-d) - x(t)*g]*g \end{aligned} \qquad \text{(HPF)}$$

and

$$\begin{aligned} h'_4(t) &= x(t-2d) - h_4(t) \\ &= [x(t-2d)] - \{x(t-2d) - [2x(t-d) - x(t)*g]*g\} \\ &= [2x(t-d) - x(t)*g]*g \end{aligned} \qquad \text{(complementary LPF)}$$

It is actually easier to implement

$$h'_4(t) = [2x(t-d) - x(t)*g]*g$$

first, and then derive

$$h_4(t) = x(t-2d) - [2x(t-d) - x(t)*g]*g = x(t-2d) - h'_4(t)$$

from it.

To implement this requires:

- delay of $x(t)$ by d seconds (or equivalent number of samples)
- multiplication of delayed $x(t-d)$ by 2
- convolution of $x(t)$ with g

- subtraction of $x(t)*g$ from $2x(t-d)$
- convolution of $[2x(t-d)-x(t)*g]$ with g
- delay of $x(t)$ by $2d$ seconds (or equivalent number of samples)
- subtraction of $h'_4(t)$ from $x(t-2d)$

Totals:

- 1 tapped delay of $x(t)$, length $2d$
- 1 multiplication or addition
- 2 subtractions
- 2 convolutions

Net benefit is that the delay buffer for $h_2(t)$ is eliminated.

Procedure:

1. Compute $x(t)*g$ by passing the input signal $x(t)$ through the lowpass filter g.
2. Compute $2x(t-d)$ by delaying the input signal $x(t)$ by d seconds (or equivalent number of samples) and then applying a gain of 2.
3. Compute $[2x(t-d)-x(t)*g]$ by subtracting the results of Step 1, $x(t)*g$, from the results of Step 2, $2x(t-d)$.
4. Compute $h'_4(t)=[2x(t-d)-x(t)*g]*g$ by passing the result of Step 3, $[2x(t-d)-x(t)*g]$, through the lowpass filter g. This is the lowpass output.
5. Compute $x(t-2d)$ by delaying the input signal $x(t)$ by $2*d$ seconds (or equivalent number of samples).
6. Compute $h_4(t)=x(t-2d)-h'_4(t)$ by subtracting the results of Step 4, $h'_4(t)$, from the results of Step 5, $x(t-2d)$. This is the highpass output.