The Filter Wizard
issue 27: Prediction and Negative-Delay Filters: Five Things You Should Know
Kendall Castor-Perry

All systems, including filters, are causal. That means they can't produce a response to an (unpredictable) stimulus before that stimulus arrives. So, how the heck can you build a filter that 'predicts' something? Well, it all depends on how high you set your sights for the quality and the relevance of that prediction.

So, riffing on the "Five Things You Should Know" format that was very popular last time, let's ask five central questions whose answers can help us navigate through this filtery quagmire.

How do filters delay signals?

Information can be impressed on a signal in many ways, and it always takes a finite amount of time to pass through a processing system. You'll be very familiar with the concept of the propagation delay of a digital block. It's simply the time elapsed between some state change at the input to the corresponding state change at the output of that block. The digital-minded reader's first thought might be of a stream of '1's and '0's, expressed physically as detectably different voltage or current levels. Propagation delay is fine for such signals, but not meaningful when we consider analogue signals that don't really have defining features associated with particular points in time.

We often lowpass-filter signals and data sequences to get rid of 'noise' – high-frequency variations that we've decided have no meaning and are getting in the way of observing a more important underlying feature. The filtering process can lend our observation a rather 'heavy touch', though; it's definitely a case of the observer affecting the observation. The most obvious consequence of conventional filtering, when we view the response graphically, is that there's clearly a time delay between variations in the input signal and corresponding variations in the filtered output. We'll see this clearly on a test signal in a moment when we look at some examples.

How do we quantify this form of delay?

This 'lag', between the input signal to the filter (or any other linear signal processing block) and the resulting output, is closely linked to the **group delay**, which is equal to (minus) the derivative of the phase response with frequency. Use sensible units for this; if you measure the phase in radians, and express the frequency in its angular form of radians per second, then (radians) divided by (radians per second) gives you a handy answer in seconds. Or you could use 'cycles' – a cycle is one full rotation, or 360 degrees. Phase difference measured in cycles, divided by the difference in regular frequency measured in Hertz, (the same as cycles per second), also gives you an answer in seconds.

It's tempting to ask, then, why we don't just design a filter that doesn't have any group delay, if we want to avoid this lag. If you've read my columns before, you'll probably recognize the 'jeopardy setup' in that sentence. Because, you guessed it – it's not as easy as that. If you look up or calculate the behaviour of the 'standard' flavours of lowpass filter response – by and large, the ones named after dead mathematicians – you'll find that their group delay is stubbornly positive, right down to zero frequency. We need to go a little off-piste here.

Can we eliminate (or more than eliminate) such delay?

The strict answer is 'no', if you want it to be zero at **every** frequency. But there's an exact technique for developing a compensating filter that, when cascaded with the original, can give you zero or even negative group delay **at DC**. This can be very useful, as we'll see. You don't need to engage in any trial and error – the Million Monkeys can stay in their cage today!

Let's say you have some lowpass transfer function **H** that has unity gain at DC. It's straightforward to show that a new transfer function **H' = 2-H** is also unity gain at DC, and has a group delay at DC that has the same magnitude as that of **H**, but **negative**. If you cascade **H** and **H'** (i.e. connect them in series), you'll get an overall transfer function, let's call it **H1**, which has unity gain at DC and **zero** group delay at DC. For any linear transfer function in the s- or z-domain, **H1** is simply equal to **HH'**, i.e. **H1 = H(2-H)**. This is **always** realizable if **H** is realizable, whatever the type of filter.

This might appear to be a bizarre thing to do. Because the function **H'** has the same order as **H** (whether we are using analogue or digital filters), you can see that combining them doubles the 'size' of the filter and therefore the resources needed to implement it. Perhaps less easy to visualize is that it's likely to significantly degrade the attenuation performance of your filter. If **H** is a lowpass function that has unity gain at DC, and a gain of unity or below at all other frequencies, then the function **2-H** has a value that can oscillate between 1 and 3, i.e. it could introduce a 'bump' of up to 9.5 dB in the response. If this falls in the stopband of the overall filter, then all that happens is a degradation of attenuation. If the bump is in the passband, then the overall passband response of the cascade will be very different from that of **H** alone.

Here's a simple example. Let's start with an n=2 Butterworth filter at 10 kHz for **H**, implemented digitally at a sample rate of 100 ksps. To design the filters and get the plots, I used a new release (available from February 2012) of the Filter tool for PSoC Creator, which gave the following coefficients for **H,** and the amplitude and group delay plot in figure 1:

Final coefficients for Biquad filter :
Coefficients are in the order A0, A1, A2, B1 and B2

0.0674552917480469
0.134910583496094
0.0674552917480469
-1.14298057556152
0.412801742553711

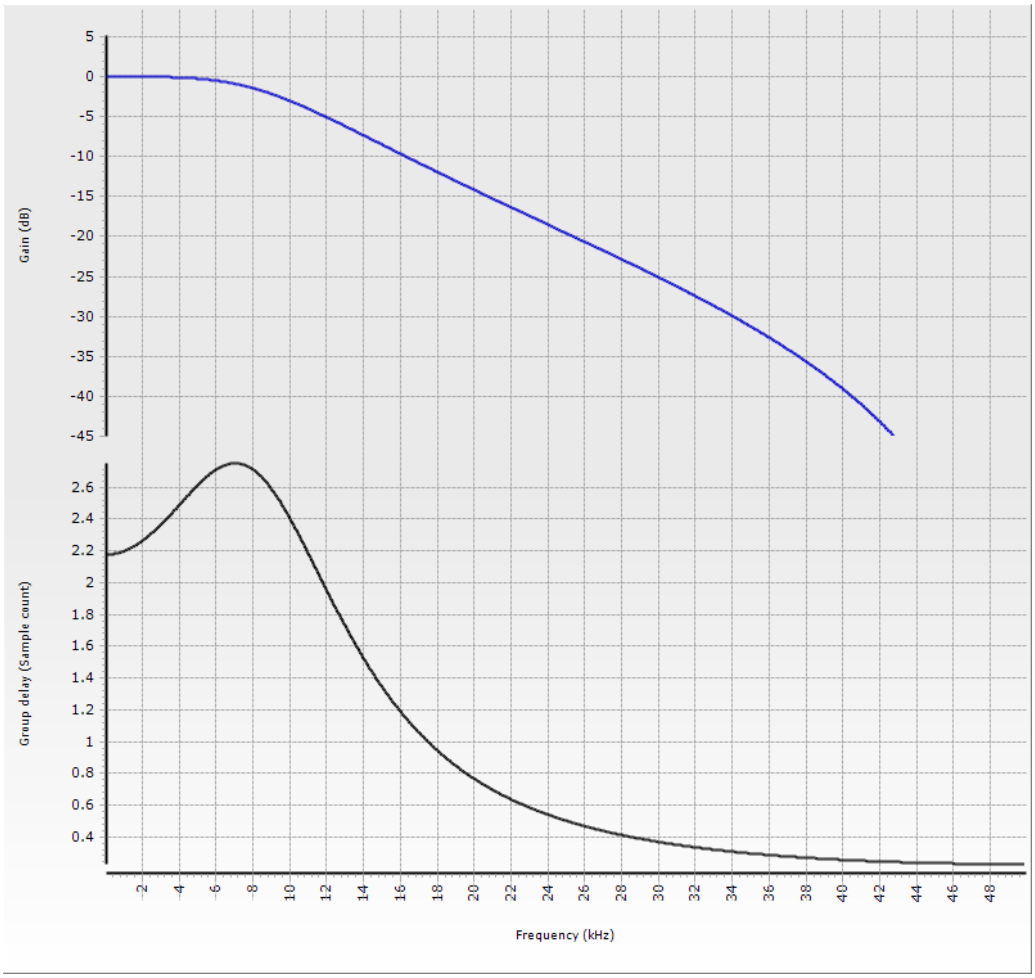

Figure 1: n=2 Butterworth filter at 0.01 Fs, magnitude and group delay.

The compensating filter **H'** has the same denominator as **H**, and a numerator that's equal to two minus (the numerator of **H**). I calculated that in a quick spreadsheet and pasted it back into the PSoC Creator Filter tool. The tool checks out the best ordering and gain for the two biquad sections; it took out 4 dB of gain to ensure that the bumpy response went no higher than 0 dB, see figure 2:

Final coefficients for Biquad filter :
Coefficients are in the order A0, A1, A2, B1 and B2

0.216065168380737
-0.2706618309021
0.0847635269165039
-1.14298057556152
0.412801742553711

0.372884273529053
0.745768547058105
0.372884273529053
-1.14298057556152
0.412801742553711

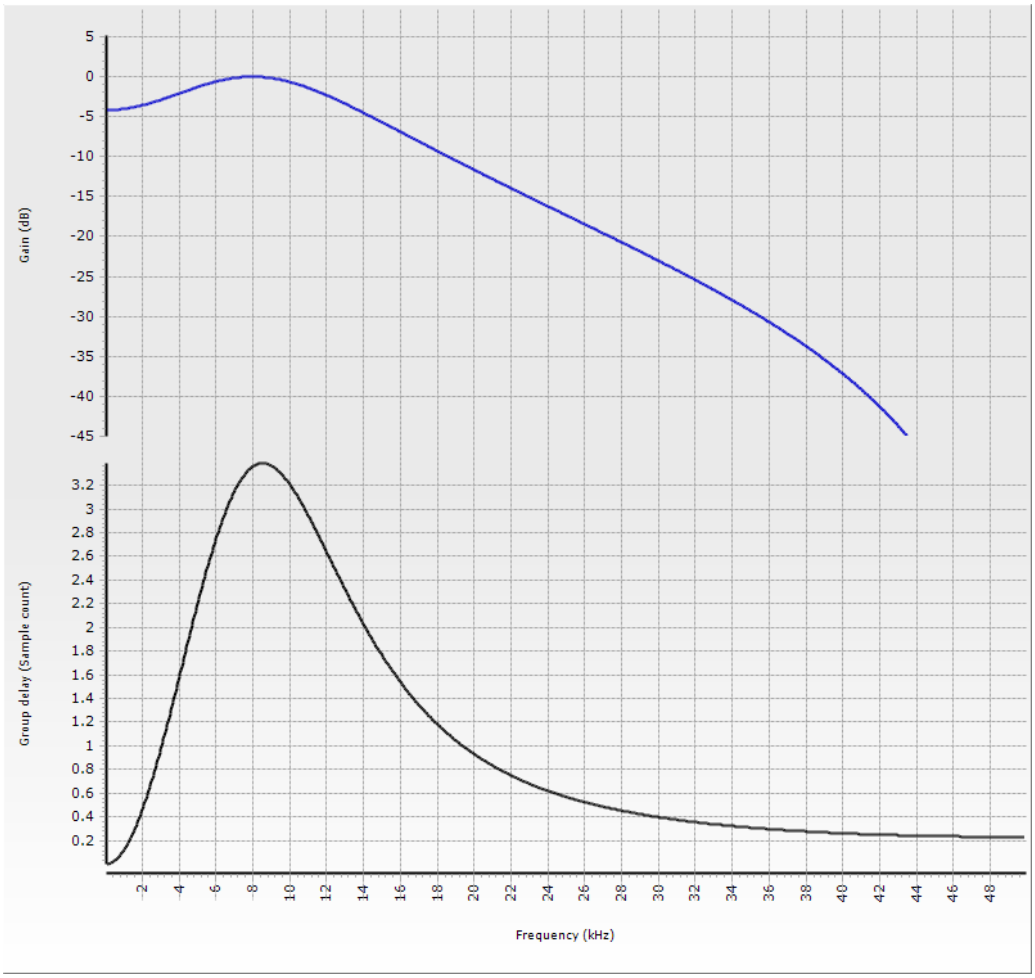

Figure 2: n=2 Butterworth cascaded with its compensating filter; zero DC group delay.

The frequency response is noticeably non-flat (there's that bump) in the passband, and some relative stopband rejection has been given up. If you're familiar with control systems theory, you'll see straight away that what we achieved was the addition of transfer function zeroes, whose group delay contribution exactly cancels out that of the original filter poles (and of the new poles that came along for the ride). But it's not such a bad response – it's still going to be useful for getting rid of high frequency noise of a data sequence – such as illustrated in figure 3, on some mystery (oooh!) data:

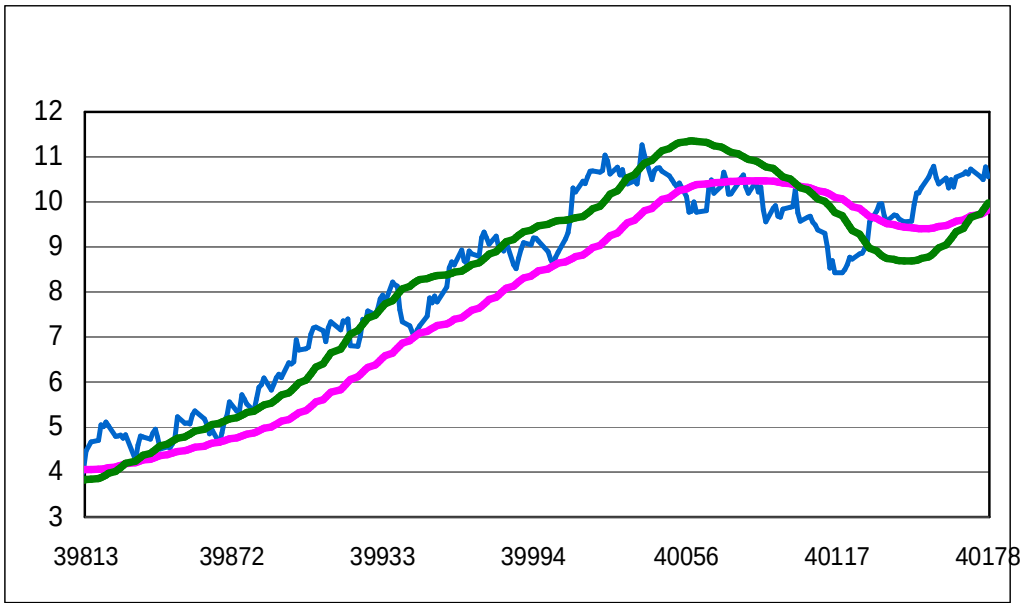

Figure 3: Some data (blue), Butterworth response (pink) and compensated (green).

We don't have to use the same function **H** to construct the compensating filter. If two transfer functions **HA** and **HB** each have both unity DC gain and the same value of DC group delay, then **H1 = HA(2-HB)** also has unity DC gain and zero DC group delay.

In particular, if **HB** is a pure time delay of value **T**, equal to **HA**'s DC group delay, we can get a lovely simplification in an FIR realization. For transfer functions where **T** just happens to be equal to **N** sample periods, we get **H1 = HA(2-z^-N)**, which can be implemented very easily in almost any digital filter structure, because these negative powers of z map straight onto unit sample delays. This condition is always met for a symmetrical FIR filter with 2N+1 taps; with a bit more work, it can be adapted to asymmetrical cases where N isn't an integer.

So, whether we choose to work in the s-domain or the z-domain, we can construct lowpass transfer functions with zero group delay at DC. But we don't have to stop at zero group delay, though; we can easily make it **negative**, and this is when we get into prediction territory. In a sampled system, it would be handy to have a filter whose output is a prediction of what the input signal is **going to be** at the next sample instant. In other words, a filter whose DC group delay is **minus one** sample periods. In the FIR case mentioned above, that's almost embarrassingly simple. Instead of using a compensation function of **2-z^-N** we just use **2-z^-(N+1)**.

Now causality would be violated if some energy actually came out of the filter before any went in. So any signal that contains information can't possibly appear at the output in a negatively-delayed form. But some signals convey no information – whatever some observers might believe, if they are psychologically invested in them – and so there's no causality to violate when the group delay is negative.

How do such filters behave?

These functions have a useful property. Obviously, for constant (i.e. DC) inputs, the output voltage equals the input, just like for a regular lowpass filter. But now equality is also met for an input **ramp** of constant rate. Unlike a 'standard' lowpass transfer function, there's no 'lag' between the output of the filter and the input signal under ramp excitation. Let's build another example and examine its properties more closely.

This time we'll do an FIR example. The starting filter for our **HA** is a symmetrical 9-tap FIR filter, (which therefore has a constant group delay of 4 samples). This was designed to notch out AC line frequencies with quite a wide tolerance around 60 Hz. I'll explain the rationale and how to design such "dial-a-null" filters explicitly in a future Filter Wizard, but let's just take it as read for the moment. For our **HB**, to get a zero-delay filter, we'll just use a simple delay of 4 samples. This makes **2-HB** look like a 5-tap FIR filter with coefficients (2,0,0,0,-1). The cascade **HAHB** is implemented as a single FIR filter by convolving the two z-plane sequences together to get a 13-tap result filter. The magnitude and group delay of **HA** and **HAHB** are shown in figure 4, this time in LTspice. The rather odd frequencies and times are because this filter was designed to operate at a sample rate of 220 samples per second. Again we've got a lumpy passband and have lost some of the stopband response.

Now. we can stray into prediction territory. If we set **HB'** to be a 5 sample delay instead of 4, and then recalculate the (now 14-tap) cascade, we get the **HAHB'** that's also shown in figure 4 (green traces). You can see that the DC group delay is now **negative** by around 4.5ms, as desired.

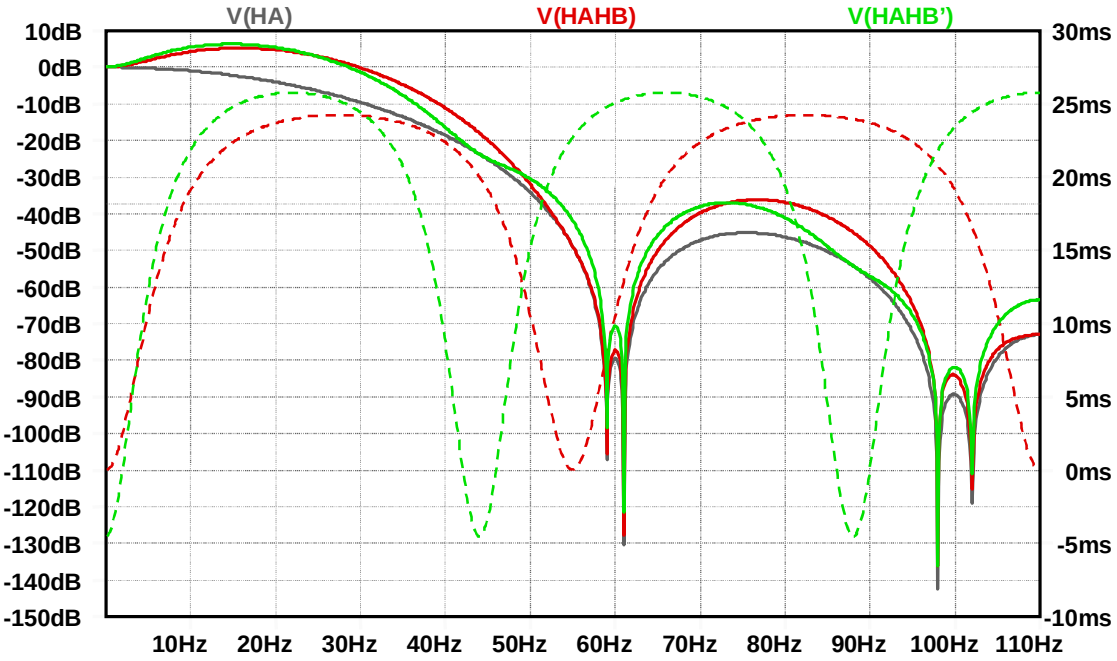

Figure 4: FIR example, plus zero delay and minus-one-sample versions.

What's the payback here? Well, let's look at the time domain behaviour. Each of the three filters is excited by a triangular stimulus that ramps up and back down again. The excitation and responses are shown in figure 5.

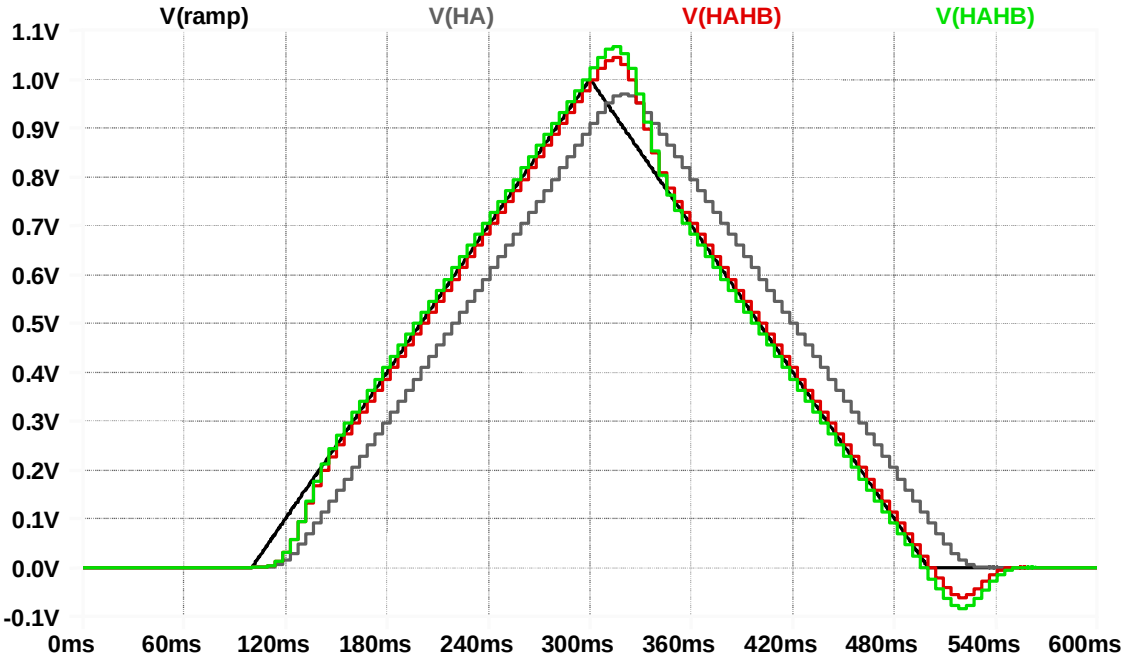

Figure 5: Response of uncompensated and compensated FIR filters to a ramp.

The 'lag' caused by the initial lowpass filter **HA** is plain to see. If you were trying to detect the point at which the signal passes some limit point, you'd clearly experience a delay in the detection response. The **HAHB** trace shows the output of our zero DC group delay filter – and it has zero lag! This highlights a hugely important point that is true in general of lowpass filters: the ramp lag between output and input of such a filter is numerically equal to the value of the group delay at DC. So, if we compensate the filter design so that the DC group delay is zero, we get rid of the lag. Of course, there's always a price to pay, and we can see that such filters are not so happy just after sudden changes in the slope of the input waveform.

If you zoom in on the **HAHB** trace, you'll see that each new sample hits the input ramp trace. The value at the output of the prediction version **HAHB'** moves to the value that the ramp will have at the beginning of the **next** sample period, as we... predicted, I suppose!

Where are such filters useful?

There are many industrial monitoring applications where 'normal' behaviour means that signals are stable (but noisy, representing temperatures, pressures, stress in a physical

structure and so on). The 'abnormal' behaviour is that some measured system parameter becomes uncontrollable and starts to ramp away.

In control systems whose feedback paths need to be filtered, this zero-delay type of filter is useful. Eliminating the group delay down at very low frequencies can significantly increase the effectiveness of the control loop at suppressing some sensed behaviour at these frequencies. Control engineers are used to manipulating the zeroes of a system transfer function to coerce the desired loop behaviour, and that's just what we did here, in a more analytical way. Our transfer function arithmetic created zeroes that cancel out the DC group delay characteristic of the poles. I already said that, didn't I!

Such zero- or negative-delay filters are often used to process non-electronic signals too. For instance, if the price of a financial instrument (such as a stock) is thought to be ramping linearly, but that ramp is corrupted by short-term trading noise, a zero-delay filter can be a useful way of extracting the underlying behaviour. As you can infer from the filter's behaviour in figure 5 though, when the triangle changes direction, such a filter will give you a wildly inaccurate result for a while until the price behaviour is smoothly ramping again. Figure 3's mystery data is in fact a sequence of share price values.

Traders of such financial instruments actually use some pretty sophisticated filtering processes on their sequences of price data. I've often been told that if the bottom falls out of the electronics market, there'll always be a job for a Filter Wizard in the financial sector, sniffing out interesting signals from huge sets of price data. But let's sail on past that particular nest of sirens, tied securely to a sturdy engineering mast, and get back on track!

An engineering application where delay manipulation like this is valuable is in the compensation for a digital class D amplifier's supply voltage variations. For a given mark:space ratio at the output switches, the mean output voltage of such an amplifier is proportional to the supply voltage – I.e. it has no power supply rejection. That's a bad thing when people seem to want to spend so little money on the power supplies in consumer audio equipment.

We can measure the instantaneous supply voltage and feed it back into the switch control algorithm. But due to the filter delays involved in measuring the supply voltage of such an amplifier, and injecting that data back into the control system, you end up correcting not for the supply voltage that's there now but for what was there some time ago. This discrepancy limits the power supply rejection we can achieve with the loop. If we filter the measured power supply voltage using a lowpass filter (that doesn't unduly emphasize some of the high frequency rubbish that is present) that has the appropriate **negative** group delay at DC, we can compensate for this effect, at least at very low relative frequencies such as at the harmonics of the AC line frequency. This technique can make a significant difference to the AC line ripple rejection of a digital power amplifier, and the approach is employed in some commercial digital amplifier designs.

So, quite a wide-ranging subject to cram into a single Filter Wizard piece; I hope you've found it stimulating. In this case, my parting puns are fairly predictable... and you should try out some of these techniques **without delay**! Happy Filtering in 2012 / Kendall