The Filter Wizard
issue 8: Countdown to S-to-Z
Kendall Castor-Perry

This column is a "Part 2" and it'll make much more sense if you read "an Excelent Fit, Sir!" first. Well, I *hope* it will, anyway. In that column, we used Excel's 'solver' functionality to bend a filter transfer function to our will. Or, rather, our *customer's* will, since he is *much* more important than we are.

Recap: we computed a filter for use in flattening out the frequency response of a sensor with a frequency response looking like figure 1:

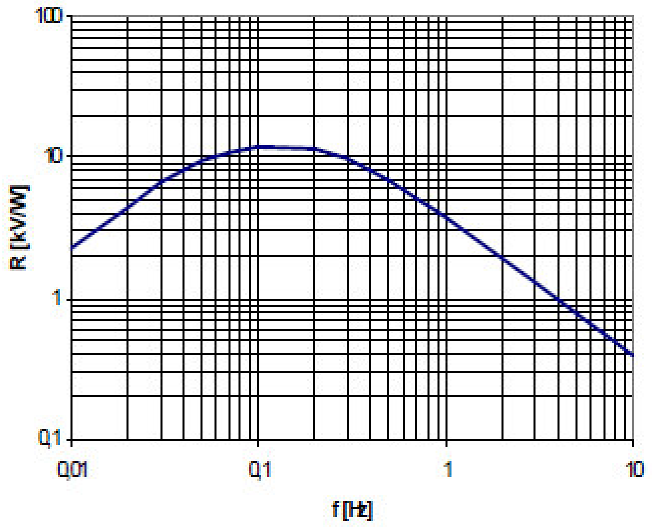

figure1: what we started with

We 'turned the response upside down'; extended it (with orders to 'come back downwards' again); guessed some filter coefficients, and fine-tuned them with Excel's solver. The light blue curve in figure 2 shows what we got:

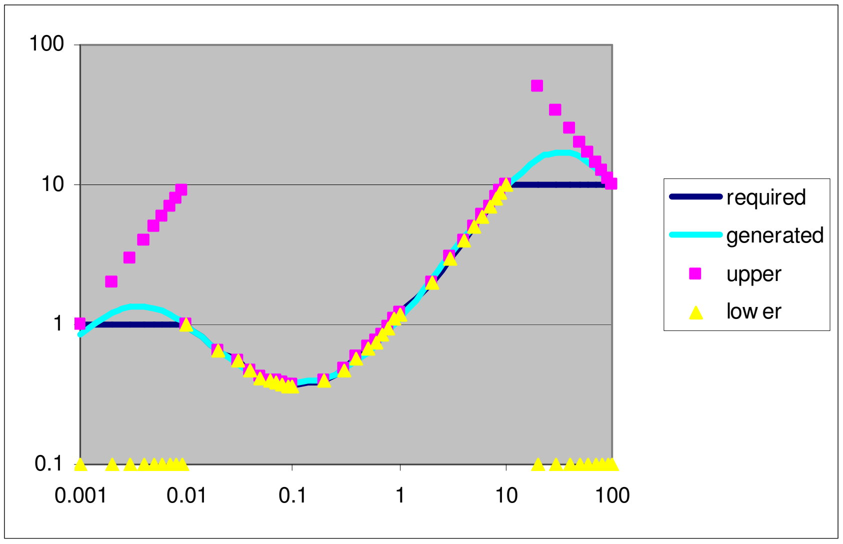

figure 2: what we ended up with

The 'accuracy' of our fit is shown in figure 3. It's rather wiggly; that just reflects the rather approximate way in which we eyeballed the required frequency response off figure 1 in the first place.

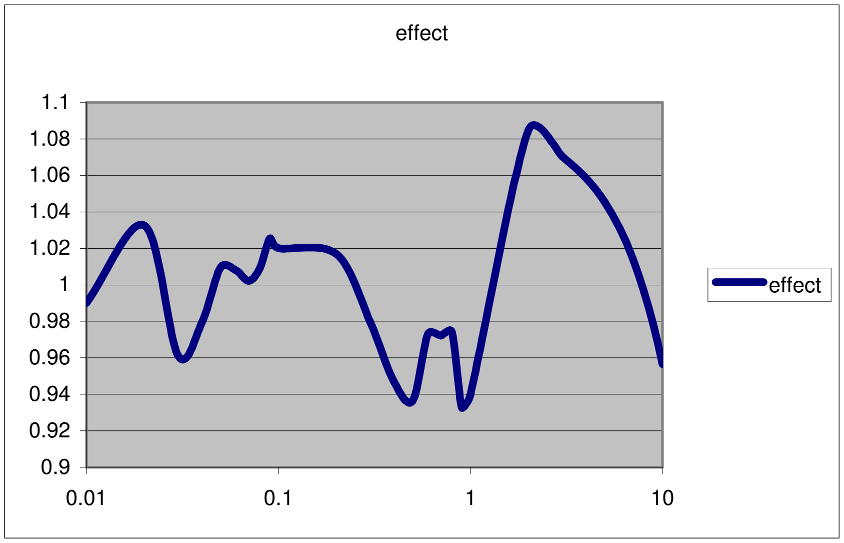

figure 3: 8% either way, that's roughly 0.7dB in real money

Now, our customer doesn't want to buy big, expensive, high quality capacitors to build an active filter with this response (good, all the more of these rare beasts for me, buwahahah). So, having established the principle of optimizing in the analogue domain, let's look at whether need to make any changes in order to use it to create a *digital* filter.

Well, the first thing to do is to choose a sample rate. The noise bandwidth of our system will be dominated by this equalizing filter, whose response falls at 6dB per octave above about 30Hz. This means that the output noise will be essentially independent of the amount of decimation that we dial into PSoC3's highly configurable low-noise delta-sigma ADC. If we set the intrinsic sample rate too high, we'll use power unnecessarily, and may have to watch for arithmetic issues in the digital filter. If we set the sample rate too low, then the inherent response of the decimation filter may have to be accounted for. This response gives both a slight response fall-off in the passband, and can also result in some residual aliasing; this is typical of all instrumentation-grade delta-sigma ADCs.

Well, we need not worry; if we pick a sample rate of 1ksps, the droop in the passband is completely negligible, and the falling response of the sensor itself ensures that aliasing isn't going to be an issue. Another more advanced digital filter design issue called 'frequency warping' can also be completely ignored (something for a future column, methinks).

Even-order harmonics of the AC line frequency will cause a modulation of the incident light level (dang! I've given the game away, yes, it *is* a PIR sensor). We can address this potential interferer with some extra filtering. So, just because we can, I've put in another lowpass filter section designed to be flat up to 10Hz (our equalization limit), and I've also sneakily dropped in a notch (or 'zero', as we Filter Wizards tend to call them) at 100Hz, which is where most of the trouble is going to be, in Europe anyway. I used a Butterworth section, which has a dissipation of sqrt(2); positioned the -3dB point out of the way at 20Hz, and instead of leaving the a2 coefficient at 0. I made it equal to 1/25, which gives a zero at sqrt(25) times the 20Hz normalization frequency, i.e. 100Hz. After a final 'solve', Excel gives:

$$H_1(s) = \frac{9.751405 \cdot s^2 + 11.5677 \cdot s + 1.290113}{0.118385 \cdot s^2 + 30.27418 \cdot s + 0.502194} \qquad @\, 0.3Hz$$

$$H_1(s) = \frac{0 \cdot s^2 + 29.92619 \cdot s +}{0.320265 \cdot s^2 + 30.09666 \cdot s + 0.648624} \qquad @\, 0.3Hz$$

$$H_3(s) = \frac{0.04 \cdot s^2 + 0 \cdot s + 1}{s^2 + 1.4142 \cdot s + 1} \qquad @\, 20Hz \qquad [5]$$

We can't put it off any longer; let's make that move from the s-domain to the z-domain. A spreadsheet is certainly a nice warm place for the s-to-z transform to hide. If you feel like doing the algebra yourself, just substitute the *bilinear z transform* expression for s:

$$s = 2 \cdot f_s \cdot \frac{1 - z^{-1}}{1 + z^{-1}} \qquad [6]$$

into equation [1] in the first installment of this article, and do the manipulation. Or, you can go and look it up in Lyons, time-poor engineer-styly (Understanding Digital Signal Processing, 2nd edition, pp266-267). This gives the coefficients of powers of z in H(z), which translate more or less directly to the actual multiplication factors used in the digital filter sections, depending on the particular topology you use. The Digital Filter Block in PSoC3 can be programmed to implement many different forms of digital filter; we'll save that detail for another day. What now remains in the last hundred words is to work out the transfer function, and then run it through a simulator to show that it does indeed do what it “says on the tin”.

So that you can check your own work, here are the coefficients in standard equation form that I ended up with for the three filter sections in table 1 (your optimizations may be slightly different:.

y(n)=sum of:	66.4472639140263x(n)	0.0809015462156547x(n)	0.0402204786802926x(n)
	-132.746083290296x(n-1)	0x(n-1)	-0.065954219990818x(n-1)
	66.2988505762885x(n-2)	-0.0809015462156547x(n-2)	0.0402204786802926x(n-2)
and			
	1.61157086751063y(n-1)	1.83726859118721y(n-1)	1.82269634820766y(n-1)
	-0.611583012532556y(n-2)	-0.837275201613798y(n-2)	-0.837183085577423y(n-2)

table 1: the three filter sections in equation form

I’ll wheel out trusty LTSpice to analyze the filter. The block I use is a z-domain biquad model that is actually built using transmission lines as delay elements. It runs cleanly and quickly in both time and frequency domains. I embed the whole circuit in ASCII schematic form in another worksheet in the Excel file; it picks up the design values directly from the s-to-z transform sheet. Just paste the worksheet back to a text file with .asc extension and it opens straight up with LTSpice (figure 7)

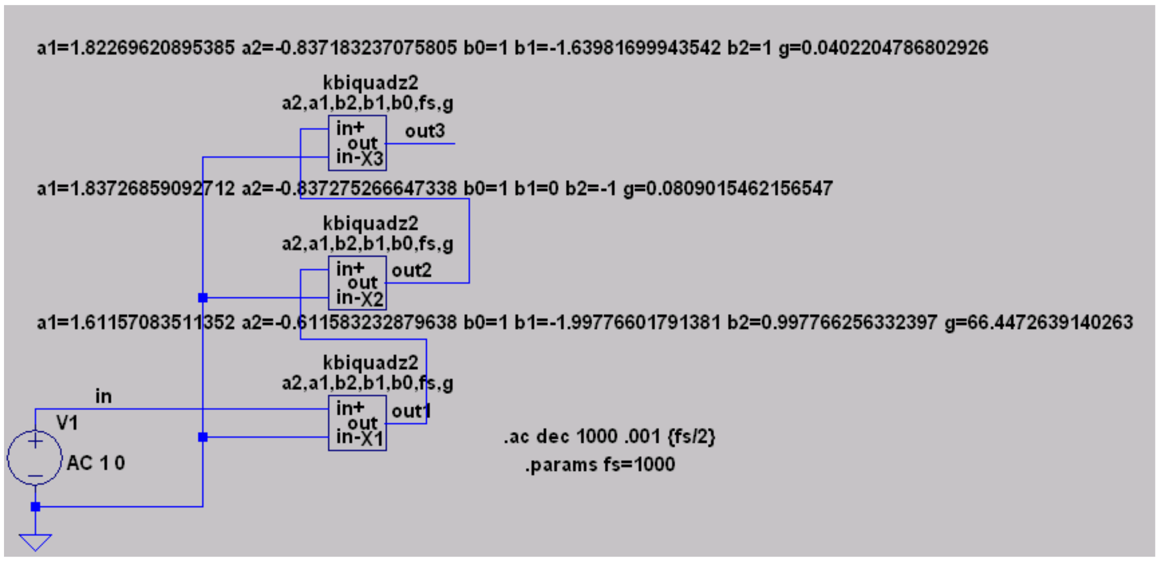
a1=1.82269620895385 a2=-0.837183237075805 b0=1 b1=-1.63981699943542 b2=1 g=0.0402204786802926
kbiquadz2
a2,a1,b2,b1,b0,fs,g
in+
out
in-X3
out3
a1=1.83726859092712 a2=-0.837275266647338 b0=1 b1=0 b2=-1 g=0.0809015462156547
kbiquadz2
a2,a1,b2,b1,b0,fs,g
in+
out
in-X2
out2
a1=1.61157083511352 a2=-0.611583232879638 b0=1 b1=-1.99776601791381 b2=0.997766256332397 g=66.4472639140263
kbiquadz2
a2,a1,b2,b1,b0,fs,g
in+
out
in-X1
out1
in
V1
AC 1 0
.ac dec 1000 .001 {fs/2}
.params fs=1000

Figure 7: simple analysis of the digital biquad

And, finally the suspense is over: here’s the response of the final filter (figure 8). The notch at 100Hz can clearly be seen, as can the fall to another notch at 500Hz – half the sample rate – which is characteristic of a digital filter designed with the bilinear transform method:

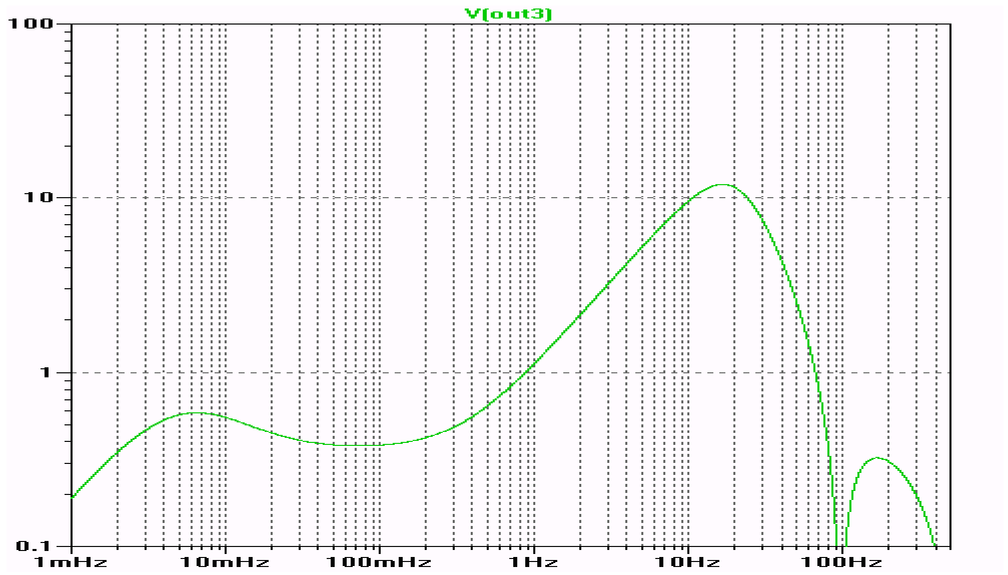

figure 8: the final realized response

Space limitations mean this could only be a whistle-stop tour through this design approach. I'll enthuse more about uses for my new friend the PSoC3 from time to time. What filter problems do you wish *you* could solve in a single chip? – Kendall.