The Filter Wizard
issue 29: Now synthesize your FIR filters using high-school algebra
Kendall Castor-Perry

Part 1 of this pair of articles broke down (i.e. factorized) the polynomial representation of an FIR filter into a product of quadratic and linear factors. This follow-up piece turns the process on its head and builds up complete FIR filters from small, easy-to-understand pieces.

The story so far

In Part 1 we took a regular FIR filter design and wrote down the filter coefficients in polynomial form to get equation [1]:

$$\begin{aligned}F(z) &= 0.000427485z^{14} + 0.001181126z^{13} - 0.005110383z^{12} - 0.023124099z^{11}\\ &- 0.01358223z^{10} + 0.091848493z^{9} + 0.268171549z^{8} + 0.359999657z^{7}\\ &+ 0.268171549z^{6} + 0.091848493z^{5} - 0.01358223z^{4} - 0.023124099z^{3}\\ &- 0.005110383z^{2} + 0.001181126z^{1} + 0.000427485z^{0}\end{aligned} \qquad [1]$$

Then we found all the roots of this polynomial, and used them to write down the factorized form of the polynomial:

$$\begin{aligned}F(z) &= \left(z^2 - 0.606685z + 0.0941454\right)\cdot\left(z^2 + 1.34888z + 1\right)\\ &\cdot\left(z^2 + 1.783416z + 1\right)\cdot\left(z^2 + 1.589812z + 1\right)\cdot\left(z^2 - 6.444124z + 10.62187\right)\\ &\cdot\left(z + 0.367424\right)\cdot\left(z + 0.950358\right)\cdot\left(z + 1.052235\right)\cdot\left(z + 2.721648\right)\end{aligned} \qquad [2]$$

As a parting shot, I pointed out that there are three quadratic terms there with unity coefficients of z^0 – and there are three deep nulls in the gain response of the filter, as was shown in the figures from part 1. Let’s take a deep breath and examine the responses of all these individual linear and quadratic factors, to see if there are some clues there.

Figure 1 shows the individual responses, treated as two- or three-tap FIR filters, of each of the factors in parentheses in equation [2]; the five quadratic factors marked as q1 to q5 and the four linear factors and L1 to L4. It’s quite a jumble of a graph, but you don’t have to be very awake to see the major salient detail: three of those quadratic factors have deep notches in the frequency response. These are indeed the three factors whose constant (z^0) coefficient is unity!

So, here’s the first takeaway. In an FIR filter whose stopband contains a number of sharp nulls, each one of them comes into being because the response of one of the polynomial’s quadratic factors falls to zero at one frequency. Just so that we don’t jump to conclusions about the particular form the factor needs to have, let’s do some more algebra to make sure. Are we having fun yet?

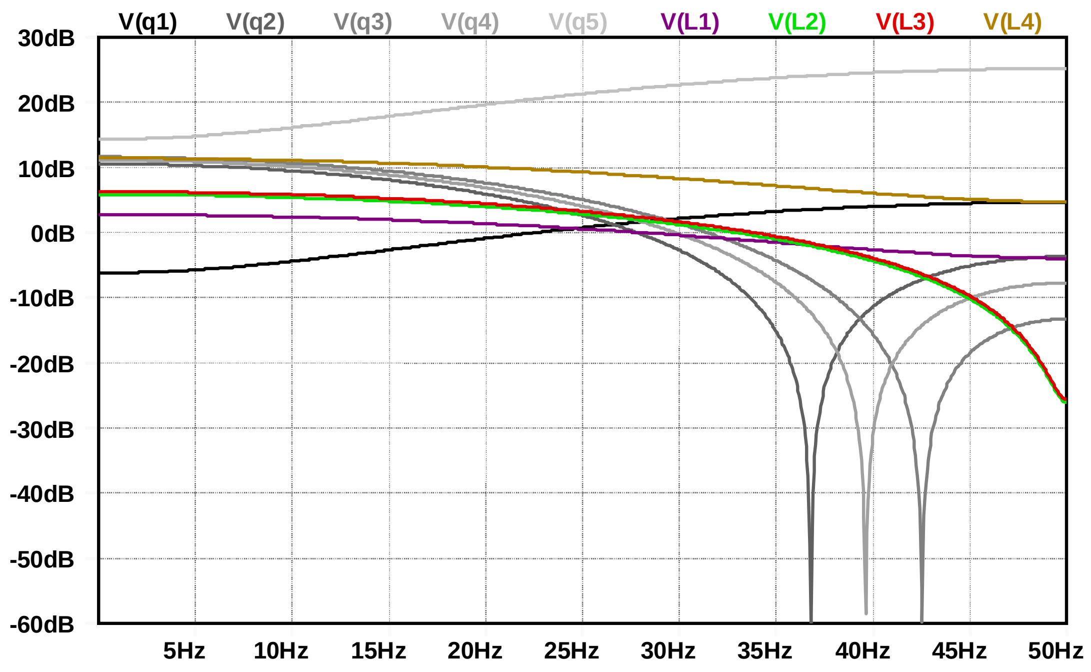


Figure 1: The frequency response of all the quadratic and linear factors of equation [2].

The frequency response of a quadratic factor used as a filter

Here's where we make an important substitution. Until now, our z has been a mystery variable with no obvious relationship to the behaviour of a sinewave. Let's introduce the expression that actually defines the z-transform. The key relationships between z , z^-1 and the frequency f of a sinewave input can be expressed in either an exponential or a trig format:

$$z = e^{j \cdot 2\pi \frac{f}{fs}};\ z^{-1} = e^{-j \cdot 2\pi \frac{f}{fs}} \qquad [3]$$

$$z = \cos\left(2\pi \frac{f}{fs}\right) + j \cdot \sin\left(2\pi \frac{f}{fs}\right);\ z^{-1} = \cos\left(2\pi \frac{f}{fs}\right) - j \cdot \sin\left(2\pi \frac{f}{fs}\right) \qquad [4]$$

where fs is the sample rate. You can make a mental connection to the effect of a small time delay (equal to the sampling interval) on the phase of a sinewave of frequency f. z is a complex variable that 'rotates round' the complex plane between purely real and purely imaginary, as the frequency f affects the argument of the trig functions.

You'll often encounter the exponential form [3] in filter books, but the complex trig form of [4] somehow seems more relevant to the engineer's habit of stuffing a sinewave into something and seeing what happens. So, let's substitute [4] into part 1's equation [3] and see if we can develop an expression for the frequency response.

$$Q(z) = z^2 - 2\cdot x\cdot z + \left(x^2 + y^2\right)$$

$$\textit{with } z = \cos\left(2\pi\frac{f}{f_S}\right) + j\cdot\sin\left(2\pi\frac{f}{f_S}\right)$$

$$\textit{gives } Q(f)$$

$$= \left(\cos\left(2\pi\frac{f}{f_S}\right) + j\cdot\sin\left(2\pi\frac{f}{f_S}\right)\right)^2 - 2\cdot x\cdot\cos\left(2\pi\frac{f}{f_S}\right) + j\cdot\sin\left(2\pi\frac{f}{f_S}\right) + \left(x^2 + y^2\right)$$

$$\boxed{\begin{aligned} Q(f) &= \cos^2\left(2\pi\frac{f}{f_S}\right) - \sin^2\left(2\pi\frac{f}{f_S}\right) - 2\cdot x\cdot\cos\left(2\pi\frac{f}{f_S}\right) + \left(x^2 + y^2\right) \\ &+ j\cdot\left(2\cdot\cos\left(2\pi\frac{f}{f_S}\right)\cdot\sin\left(2\pi\frac{f}{f_S}\right) - 2\cdot x\cdot\sin\left(2\pi\frac{f}{f_S}\right)\right)\end{aligned}} \qquad [5]$$

The first line of the boxed portion of [5] shows the real part, and the second line shows the imaginary part. Now, for a quadratic section to give a null in the frequency response, there must be a frequency fn for which Q(fn)=0. This means that both the real and the imaginary parts of equation [5] must be identically zero at that frequency. It should be easy to see that for the imaginary term in [5] to vanish for a frequency fn, we simply need to have

$$x = \cos\left(2\pi\frac{f_N}{f_S}\right) \qquad [6]$$

and if this is the case, we can substitute it back into the real part of [5] and set that to equal zero, from which we get

$$\cos^2\left(2\pi\frac{f_N}{f_S}\right) - \sin^2\left(2\pi\frac{f_N}{f_S}\right) - 2\cdot\cos^2\left(2\pi\frac{f_N}{f_S}\right) + \cos^2\left(2\pi\frac{f_N}{f_S}\right) + y^2 = 0$$

$$\textit{i.e.}$$

$$y = \pm\sin\left(2\pi\frac{f_N}{f_S}\right) \qquad [7]$$

$$\textit{and so, because } \cos^2 + \sin^2 = 1,$$

$$\boxed{Q(z) = z^2 - 2\cdot\cos\left(2\pi\frac{f_N}{f_S}\right)\cdot z + 1} \qquad [8]$$

Bingo! This corroborates our earlier observation, and what's more, it gives us a direct expression for the quadratic factor that's needed in order to locate a frequency response null at any fn that we want. Figure 2 shows the frequency response of quadratic factors (three-tap filters) built using equation [8], for several notch frequencies. Equations [6] and [7] give the real and imaginary parts, and we can see that there are some special cases. For zero frequency (z term = -2) and half the sample rate (z term = +2), the factor Q(z) is the product of two equal real factors, either (z-1)*(z-1) or (z+1)*(z+1). For a frequency of Fs/4, the roots are purely imaginary, and Q(z)=(z+j)*(z-j).

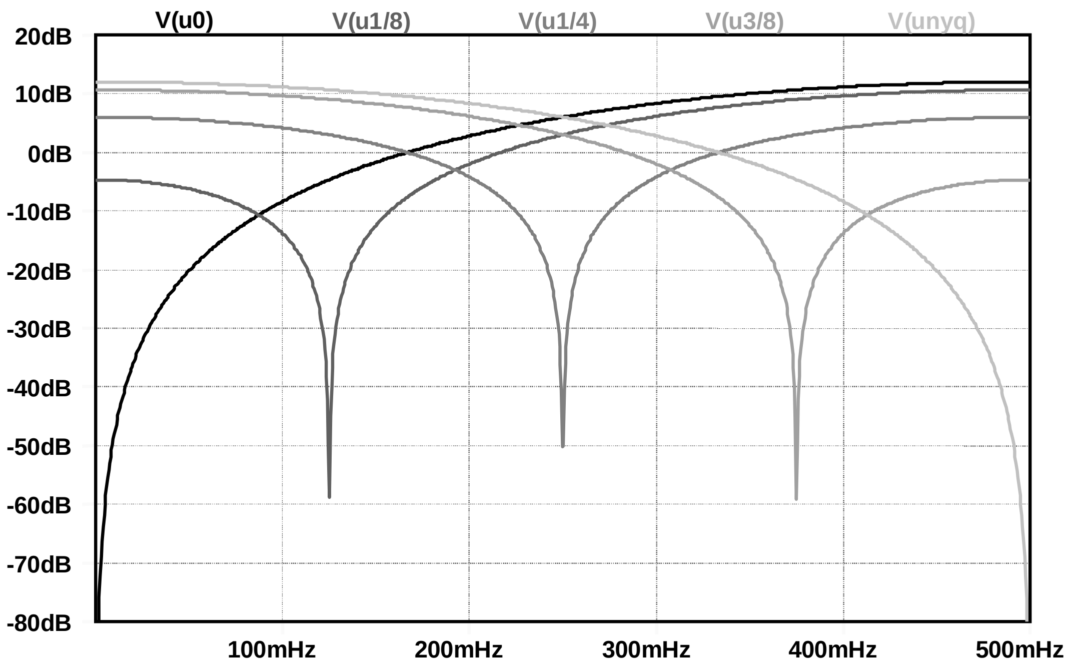


Figure 2: Factors of the form of equation [8] for various values of null frequency.

Now for a real example. Let's say that I want very high attenuation between around 49 Hz and 51 Hz, again between 59 Hz and 61 Hz (i.e. the system must be able to reject both commonly-used AC line frequencies without change of coefficients or sample rate), and finally also between 98 Hz and 102 Hz. I'll specify 220 Hz as the sample rate to ensure that the 2nd harmonic of 60 Hz aliases round and falls right into that same hole that the 50 Hz second harmonic does (homework, if you can't see why that happens). So, what happens if I just build a filter that has nulls at these six frequencies? It's simple to make up a spreadsheet that calculates the z coefficient shown in equation [8] for a set of frequencies, and multiplies them up into a single polynomial. The six frequencies create six quadratic factors, shown in table 1.

sample rate	220					
null frequency	49	51	59	61	98	102
z^2 term	1	1	1	1	1	1
z^1 term	-0.341044385	-0.22798282	0.227983	0.341044	1.883689	1.948024
z^0 term	1	1	1	1	1	1

Table 1: Six quadratic factors, calculated for specific null frequencies.

When multiplied together (equation [9]), these six quadratic factors produce a twelfth-order polynomial (equation [10]).

$$Ex(z) = (z^2 - 0.341044z + 1)\cdot(z^2 - 0.227983z + 1)\cdot(z^2 + 0.227983z + 1)$$
$$\cdot(z^2 + 0.341044z + 1)\cdot(z^2 + 1.883689z + 1)\cdot(z^2 + 1.948024z + 1) \qquad [9]$$

$$Ex(z) = 0.004253551z^{12} + 0.016298385z^{11} + 0.040413766z^{10}$$
$$+ 0.07874911z^{9} + 0.120772144z^{8} + 0.154853936z^{7} + 0.169318216z^{6}$$
$$+ 0.154853936z^{5} + 0.120772144z^{4} + 0.07874911z^{3} + 0.040413766z^{2}$$
$$+ 0.016298385z^{1} + 0.004253551z^{0} \qquad [10]$$

The frequency response of this filter is shown in figure 3, straight out of Excel. For the intended application, it's actually already a pretty good filter, "straight out of the box", and it has no wasted coefficients at all!

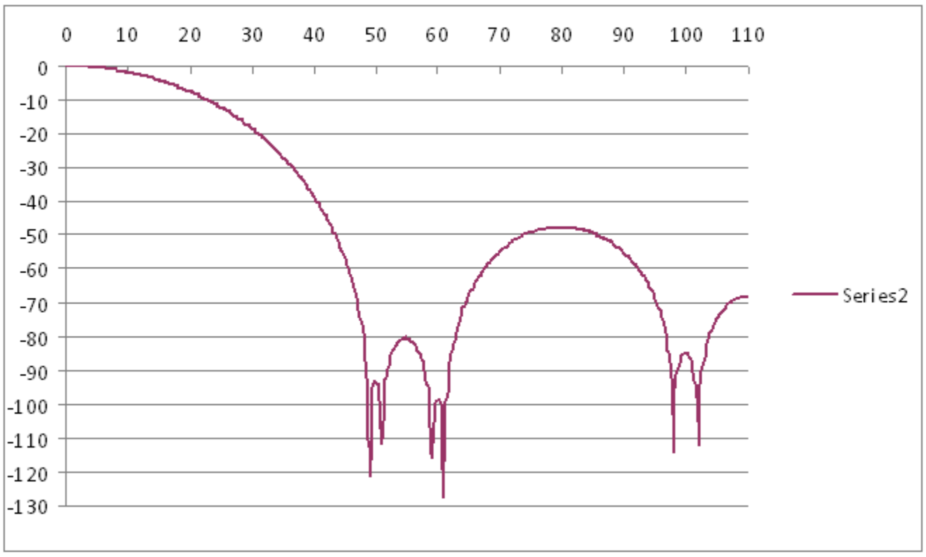


Figure 3: The frequency response of the filter defined by equation [10].

Let's take a closer look at the region around 50 Hz and 60 Hz in LTSpice. Yep, those nulls are indeed where we wanted them to be – see figure 4. Now, it's easy to play around with this approach using the spreadsheet, adding and moving nulls until you get exactly what you want. Not all combinations of null frequencies will necessarily produce a response that you're happy with first time round. You'll find that as you try to push the stopband of a lowpass filter well below fs/4, for example, you'll need to add extra nulls above this frequency to 'pin the stopband down'. It's an empirical way to proceed, for sure – but you certainly get a feeling for what's going on with these nulls.

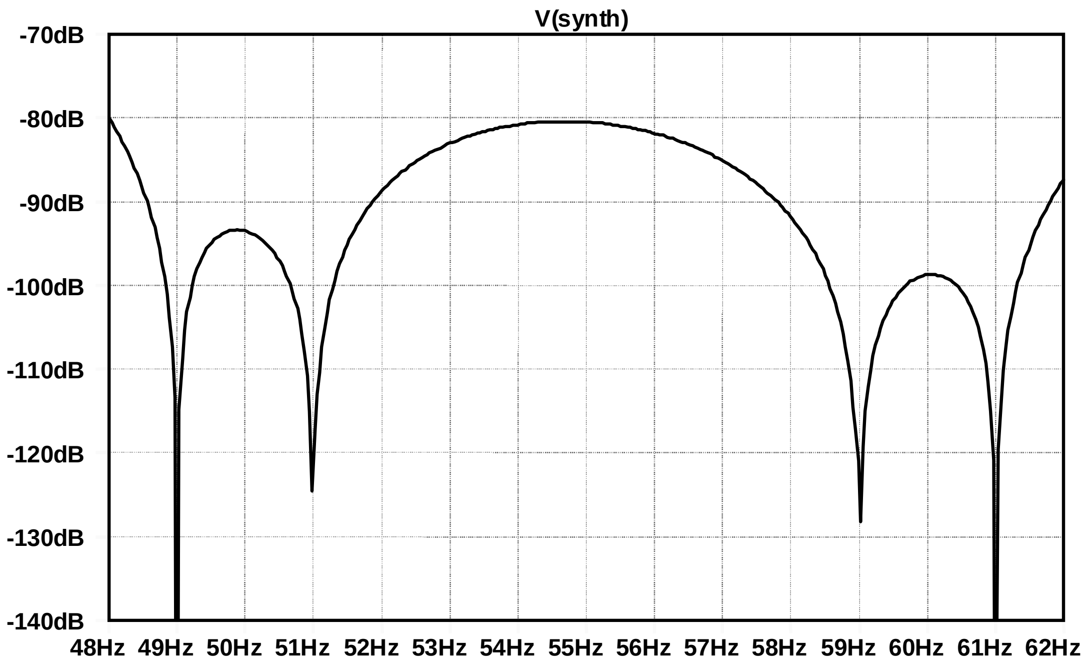


Figure 4: Up close and polynomial; nice notches just where we asked for them.

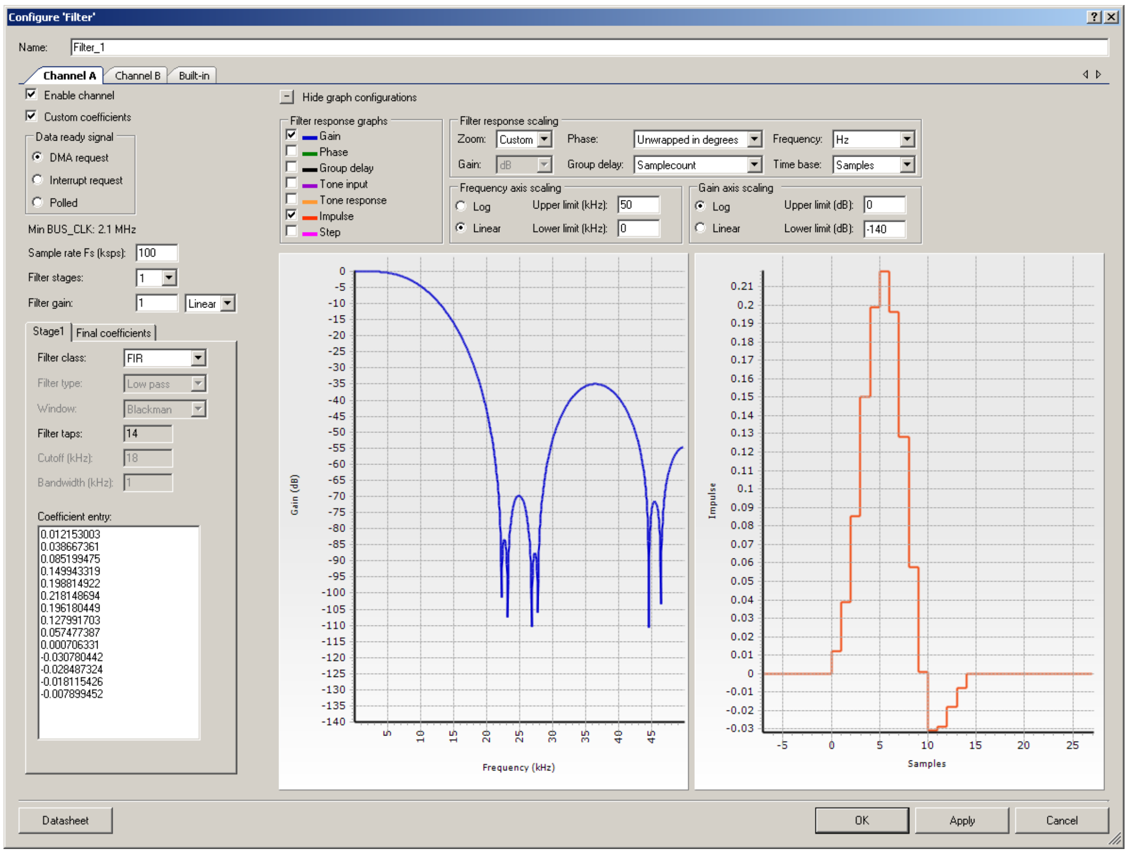
If you want a filter with an even number of taps, you can multiply in an extra linear term of (z+1), which has the effect of putting a null at fs/2 (which gives us some extra stopband rejection). That's something we can deduce from one of those earlier special cases; if a function Q(z)=(z+1)*(z+1) has a null at fs/2, so must its square root.

Flattening the passband

Our filter has got useful stopband behaviour, but the passband response isn't flat anywhere, it rolls off in the passband right from low frequencies. The response is actually close to Gaussian; you can see from the coefficients that the filter isn't going to overshoot at all on a step input signal. How to tell? Just look at the sum of progressively more terms in the impulse response; it's monotonically increasing because all the filter coefficients are positive, so the time response to a step never falls as each extra coefficient is 'uncovered'. That behaviour can be useful, but what do you do if you'd rather have a flatter frequency response over at least part of the passband?

We can add more factors to our polynomial to flatten out that droop, choosing those that contribute a gently rising response as frequency increases from DC. Look back at figure 1; some of those polynomial terms didn't give us a null, but just a gentle slope up or down in response. These acted together to give a flatter overall frequency response to our original example, plotted in part 1. We can construct our own terms to flatten out that gentle low frequency droop. Each quadratic factor you add will increase the coefficient count by two; multiplying in a linear term just adds one to the tap count.

The general analytic expression for the frequency response of an arbitrary root value (or pair of values) is quite clunky and I won't write it down here. It's quite possible to build an optimization script (think Million Monkeys) that can tweak the coefficients of extra linear and quadratic terms to flatten out the passband to the desired level. Here's an example done entirely empirically; I've taken our synthesized filter and multiplied the polynomial by an additional (z-0.65). The rising response of this extra term has the effect of flattening out the droopy passband at lower frequencies – and also degrading the stopband rejection a little. Figure 5 shows what those coefficients look like when entered back into PSoC Creator's filter tool. The impulse response has become non-symmetrical, meaning it's no longer a linear phase filter. We can rectify that with yet more terms – but not today, I'm out of space and time! "Swings and roundabouts" works for filter design too.


Configure 'Filter'

Name: Filter_1

Channel A | Channel B | Built-in

[✓] Enable channel
[✓] Custom coefficients

Data ready signal
[✓ DMA request] Interrupt request Polled

Min BUS_CLK: 2.1 MHz
Sample rate Fs (ksps): 100
Filter stages: 1
Filter gain: 1 Linear

Stage1 | Final coefficients

Filter class: FIR
Filter type: Low pass
Window: Blackman
Filter taps: 14
Cutoff (kHz): 18
Bandwidth (kHz): 1

Coefficient entry:

```
0.012153003
0.038667361
0.085199475
0.149943319
0.198814922
0.218148694
0.196180449
0.127991703
0.057477387
0.000706331
-0.030780442
-0.028487324
-0.018115426
-0.007899452
```

Hide graph configurations

Filter response graphs
[✓] Gain
[ ] Phase
[ ] Group delay
[ ] Tone input
[ ] Tone response
[✓] Impulse
[ ] Step

Filter response scaling
Zoom: Custom
Phase: Unwrapped in degrees
Frequency: Hz
Gain: dB
Group delay: Samplecount
Time base: Samples

Frequency axis scaling
Log [✓ Linear]
Upper limit (kHz): 50
Lower limit (kHz): 0

Gain axis scaling
[✓ Log] Linear
Upper limit (dB): 0
Lower limit (dB): -140

Figure 5: Throw in another root (z-0.65) and we can flatten that passband a bit.

Let's be honest; this isn't intended as a replacement for more conventional filter design techniques, especially if you're in a hurry!. But it's a really good way of understanding the effect that those roots – filter designers call them 'zeroes' – of the filter polynomial are getting up to under the hood. For our example hum rejection filter, we got a useful result (we'll be using it commercially), with the added satisfaction that we built the coefficient set up right from first principles, and we understand exactly why it has its nulls where they are.

I hope this shows you that it’s sometimes useful to break down the apparently monolithic, single-stage nature of an FIR filter into simpler pieces that can be separately manipulated. Try dialing some of these nulls into your next FIR filter design. It’s the ‘notchural’ way!
best / Kendall