Chapter 8: The Design and Implementation of Finite Impulse Response Filters

8.1 Introduction

Design of FIR filters is predicated on a single question: given a desired frequency response, how do we obtain the (discrete) coefficients of its impulse response? There are several ways in which the coefficients may be obtained, and each method has its own merits and limitations. Here, we shall study two of the most widely applied techniques, these being the *window method* and the *frequency sampling method*. It is fair to say that in practice the overwhelming majority of filters are computed using these two methods, since they are simple to implement, the algorithms are inherently stable, efficient and rapid and the resulting impulse response has excellent phase and frequency response characteristics.

8.2 The window method

The window method provides a very straightforward, computationally efficient and effective means of calculating the FIR coefficients of simple low-pass, high-pass, band-pass and band-stop filters. The *ideal* frequency responses of these filter types are shown in Figures 8.1a to 8.1d.

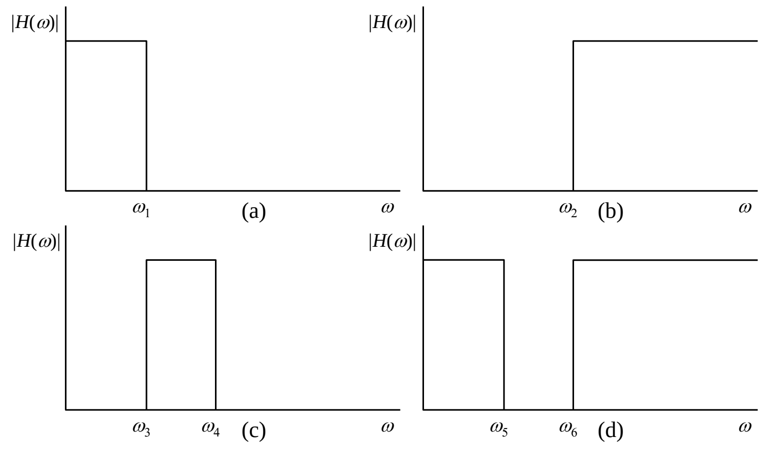


Figure 8.1 Ideal frequency responses of (a) low-pass, (b) high-pass, (c) band-pass and (d) band-stop filters.

The first stage in this process is based upon a discrete form of the inverse Fourier transform of a rectangular function; the solution is given by

$$x(t) == \frac{\omega_1}{\pi}\left[\frac{\sin\omega_1 t}{\omega_1 t}\right] \tag{8.1}$$

Note that this is the continuous time representation. The impulse response of a discrete low pass FIR filter is therefore given by

$$h[n] = \frac{2\pi f_1}{\pi}\left[\frac{\sin(2\pi f_1 n / f_s)}{2\pi f_1 n}\right] = \frac{\sin(2\pi f_1 n / f_s)}{n\pi}, \quad n \neq 0 \tag{8.2}$$

where f_1 represents the cut-off point of the filter and f_s represents the sample rate in hertz. When n = 0, both the numerator and the denominator of Equation 8.2 are zero; in this case they equate to 1, and so $h[0] = 2f_1/f_s$. Using a similar line of reasoning, we may obtain the impulse responses for the three other remaining simple filter types. These are given in Table 8.1.

The equations listed in Table 8.1 allow us to calculate an indefinite number of coefficients for the impulse response. The more coefficients we use, the more closely the performance of the filter approximates the ideal frequency response; however, the trade-off for long filters is increased storage requirement and longer processing times. Typically, modern FIR filters use anything between a few tens and several thousand coefficients. The second stage is to apply a suitable window function to the impulse response. If the impulse response is abruptly truncated, i.e. if we use a rectangular window function, then the frequency response of the filter will contain undesirable ripples in the pass and stop band region. The use of other window functions ameliorates this phenomenon, but it also adversely affects the width of the transition zone, denoted by Δf.

Filter Type	***h*[*n*] (*n* ≠ 0)**	***h*[*n*] (n = 0)**
Low-pass	$\frac{\sin(2\pi f_1 n/f_s)}{n\pi}$	$\frac{2f_1}{f_s}$
High-pass	$-\frac{\sin(2\pi f_2 n/f_s)}{n\pi}$	$1-\frac{2f_2}{f_s}$
Band-pass	$\frac{\sin(2\pi f_4 n/f_s)}{n\pi}-\frac{\sin(2\pi f_3 n/f_s)}{n\pi}$	$\frac{2}{f_s}(f_4-f_3)$
Band-stop	$\frac{\sin(2\pi f_5 n/f_s)}{n\pi}-\frac{\sin(2\pi f_6 n/f_s)}{n\pi}$	$1-\frac{2}{f_s}(f_5-f_6)$

Table 8.1. Impulse response equations for simple filter types using the window method.

The transition zone width is governed by the choice of window function and the number of coefficients or taps in the design. Once the window type has been selected, the final stage in the design process is to decide upon a transition zone width and calculate the number of coefficients required. The final column of Table 7.1 is useful in this regard. For a given transition zone, which we here define as the region between the 90% and 10% amplitude points of the normalised frequency response (with a Nyquist of 1 Hz), the number of coefficients, *N*, is given by

$$N=\frac{k}{\Delta f} \tag{8.3}$$

Figure 8.2. shows some of the most important features of the frequency response of a typical filter. Clearly this is a low-pass type, but the various characteristics have their equivalents in the other types depicted in Figure 8.1.

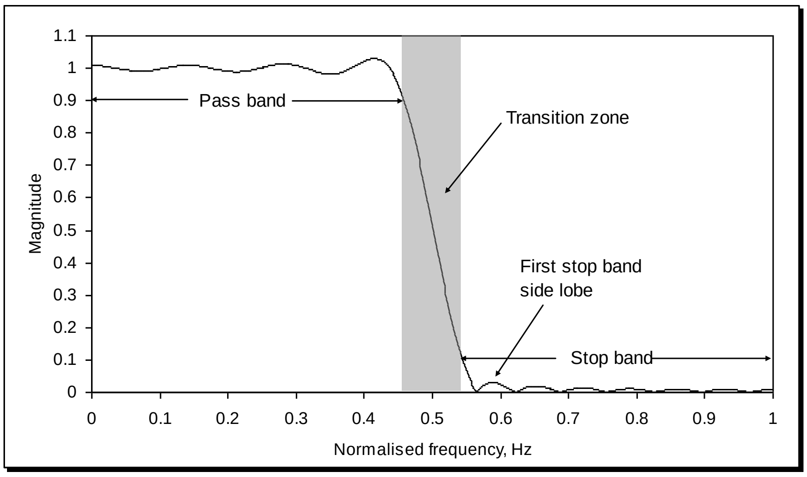
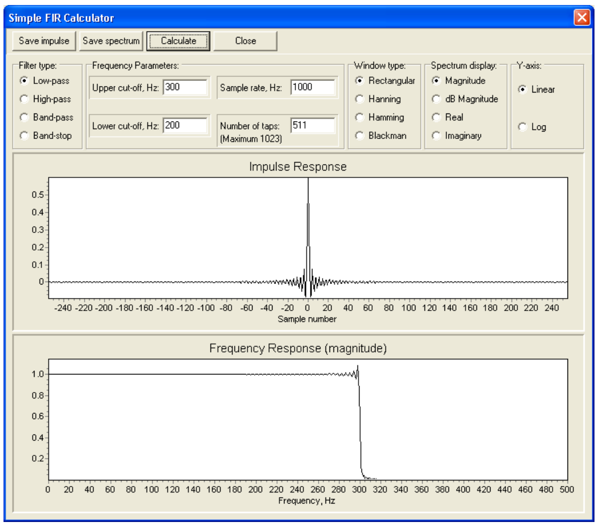


Figure 8.2. Main features of a typical low-pass filter frequency response.

In summary therefore, the procedure for using the window method is as follows.

1. Using the appropriate formulae listed, calculate the coefficients for the specified filter type.
2. Apply a suitable window function, taking into account the stop band attenuation.
3. Using Equation 8.3, determine the number of coefficients required for a given window type, according to the desired transition width.


Simple FIR Calculator

Filter type: [✓ Low-pass] High-pass Band-pass Band-stop

Frequency Parameters: Upper cut-off, Hz: 300 Sample rate, Hz: 1000 Lower cut-off, Hz: 200 Number of taps: (Maximum 1023) 511

Window type: [✓ Rectangular] Hanning Hamming Blackman

Spectrum display: [✓ Magnitude] dB Magnitude Real Imaginary

Y-axis: [✓ Linear] Log

Figure 8.3. Screenshot of *Fir.exe*.

Incidentally, step three is often not computed in a formal way; instead, the designer may instead select an arbitrary number of taps, altering this until a satisfactory filter response results. A screenshot of a program is shown in Figure 8.3 that exploits the window method to generate the coefficients of the four standard filter types we have discussed so far, for FIR filters comprising up to 1023 coefficients. The program will also calculate the frequency response; Listing 8.1 shows a critical extract from the program within which the impulse response is calculated.

```
fh   :=strtofloat(edit1.text);
fl   :=strtofloat(edit2.text);
srate:=strtofloat(edit3.text);
taps :=strtoint(edit4.text);
frq:=srate/N1;
k:=taps div 2;
{Next part determines kind of filter}
case f_type.itemindex of
  0: for n:=-k to k do if (n<>0) then h[n]:=sin(2*pi*fh*n/srate)/(pi*n)
     else h[n]:=fh*2/(srate);
  1: for n:=-k to k do if (n<>0) then h[n]:=-sin(2*pi*fl*n/srate)/(pi*n)
     else h[n]:=1-fl*2/(srate);
  2: for n:=-k to k do if (n<>0)
     then
       h[n]:=sin(2*pi*fh*n/srate)/(pi*n)-sin(2*pi*fl*n/srate)/(pi*n)
       else
       h[n]:=(2/srate)*(fh-fl);
  3: for n:=-k to k do if (n<>0)
     then
       h[n]:=sin(2*pi*fl*n/srate)/(pi*n)-sin(2*pi*fh*n/srate)/(pi*n)
       else
         h[n]:=1-(2/srate)*(fh-fl);
end;
{Now apply selected window}
case w_type.itemindex of
  1: for n:=-k to k do h[n]:=h[n]*(0.5+0.5*cos(2*pi*n/taps));
  2: for n:=-k to k do h[n]:=h[n]*(0.54+0.46*cos(2*pi*n/taps));
  3: for n:=-k to k do h[n]:=h[n]*(0.42+0.5*cos(2*pi*n/(taps-
     1))+0.08*cos(4*pi*n/(taps-1)));
end;
```

Listing 8.1.

The program first allocates the various input parameters to the variables `fh` (upper cut-off point), `fl` (lower cut-off point), `srate` (sample rate) and `taps` (number of taps). Note that the impulse response will be held in the array h[n], so this is initially set to zero. Depending on the choices the user has made, the program generates the coefficients of one of the four filter types, employing directly the equations given in Table 8.1. After this, the program applies one of the standard windows. Finally, the program computes the frequency response (not shown in Listing 8.1).

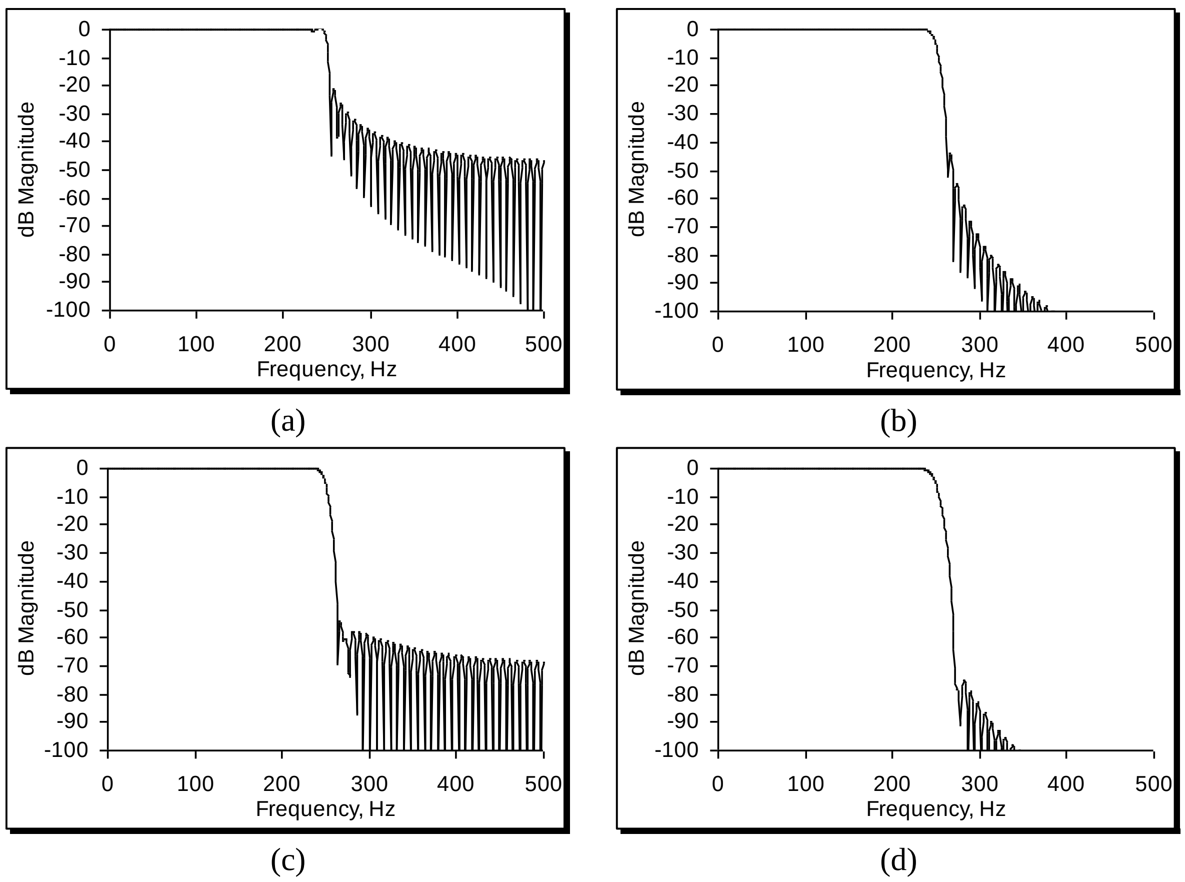


Figure 8.4. Magnitude frequency responses, shown in dBs, of a low-pass FIR filter with a cut-off of 250 Hz and a sample rate of 1 kHz using (a) a rectangular, (b) a Hanning, (c) a Hamming and (d) a Blackman window.

Figure 8.4 shows some examples of the output, in which a 127-tap low-pass filter with a cut-off of 250 Hz and a sample rate of 1 kHz is designed using (a) a rectangular, (b) a Hanning, (c) a Hamming and (d) a Blackman window. The plots suggest that the stop-band performance of the filter designed using the rectangular window, as far as audio purposes are concerned, is rather poor.

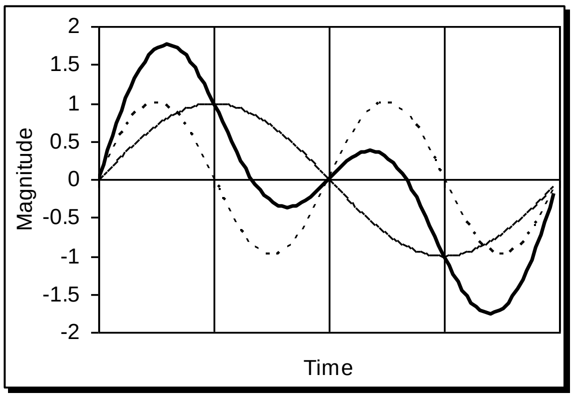
8.3 Phase linearity

One of the stated advantages of the FIR filter is that it can be designed to have perfectly linear phase. Each harmonic is therefore delayed by the same time interval, so the shape of the signal is preserved. A non-linear phase system that distorts the shape of a signal is undesirable in many circumstances. The phase delay of a filter, T_p, is the time delay each frequency component of the output signal experiences after passing through the filter. It is given by

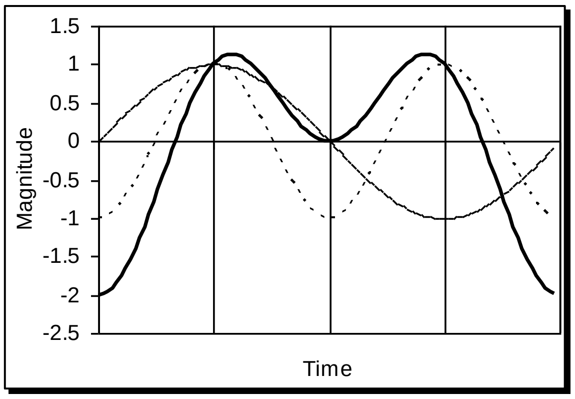
$$T_p = -\frac{\Phi_d(\omega)}{\omega} \tag{8.4}$$

Figure 8.5a shows three sine waves. The solid one is the fundamental (or first harmonic), and has a frequency of exactly 1Hz. The second curve drawn in the broken line has a frequency of 2Hz. Both have a phase offset of zero. The signal shown in bold is the sum of these two sinusoids, i.e. a notional signal we might present to a filter. Now: think about a filter that delayed the onset of the fundamental by 250 ms. If we wished to preserve the shape of the summed signal, then the second harmonic would also have to be delayed by 250 ms. Expressed in radians, if in this case the first harmonic was delayed by $\pi/2$, then the second harmonic would need to be delayed by π. Thus we reach an important conclusion: in order to preserve the phase characteristics of the signal, each harmonic must be delayed by a negative phase angle, whose magnitude is linearly proportional to its frequency, i.e.

$$\Phi_d(\omega) = -T_p \omega \qquad (8.5)$$



(a)



(b)

Figure 8.5. Effects of linear and non-linear phase on signals. In both parts (a) and (b), the bold trace denotes the sum of a 1 Hz and a 2Hz sine wave. However, in part (b), the 2 Hz sine wave has also been phase delayed by $\pi/2$, ie. 90°.

If a filter adheres to Equation 8.5, we know that each harmonic is delayed by the same time interval. In Figure 8.5b an example is given of an output where this has not happened. The 2nd harmonic has been delayed by $\pi/2$ (125 ms), whilst the fundamental has been left unaltered. Observe the change in shape of the summed signal. It is worth mentioning that the negative sign appears in Equation 8.5 because in practice, the FIR filter, like any other filter, is causal. This means that the signal passing through it is always delayed in time, since the filter cannot anticipate the input signal. There are exactly four kinds of FIR arrangements that deliver this linear phase characteristic; the four kinds are listed as follows:

1. Even symmetry, with odd-number of taps.
2. Even symmetry, with even number of taps.
3. Odd symmetry, with odd number of taps.
4. Odd symmetry, with even number of taps.

By far the easiest to design is the first kind. Since the window method delivers a symmetrical filter, by choosing an odd number of coefficients or taps we obey the relationship:

$$h(n) = h(N\text{-}n\text{-}1), \qquad n = 0,1...(N\text{-}1)/2 \qquad (N \text{ odd}) \qquad (8.6)$$

(Incidentally, if the filter were symmetrical at $t = 0$, then the filter would have no imaginary terms in its Fourier transform and hence no time delay of the signal at all.) To consolidate the matter of phase linearity, let's look at the effects some notional linear phase filter could have on some signal we will device. Table 8.2 indicates a composite signal $x[n]$ with 4 harmonics, with the first at 1 Hz. The harmonics have arbitrarily been assigned phase angles, Φ_x and delays, T_{px} (relative to time zero). A linear phase filter delays each harmonic by a phase angle proportional to the negative of the frequency. This generates an output signal $y[n]$ with phase angles Φ_y and delays T_{py}. In the final column, the phase differences between the input and output signals are shown for each harmonic. It is a constant value of -0.05s, indicating that all harmonics have been delayed in time by a fixed amount and therefore the filter has a truly linear phase response.

ω	Φ_x	T_{px}	$\Phi_h = -\alpha\omega$	Φ_y	T_{py}	$T_{px} - T_{py}$
2π	0.3π	-0.1500	-0.1π	0.2π	-0.1000	-0.0500
4π	0.8π	-0.2000	-0.2π	0.6π	-0.1500	-0.0500
6π	0.4π	-0.0667	-0.3π	0.1π	-0.0167	-0.0500
8π	0.7π	-0.0875	-0.4π	0.3π	-0.0375	-0.0500

Table 8.2. Effects of a linear-phase filter on the first harmonics of a composite signal.

8.4 The frequency sampling method

The window method is excellent for designing the four basic filter types, but what if we need a filter with a rather more unusual response? The window method will not suffice under these circumstances, yet its underlying principle suggests how a general-purpose algorithm may be devised to enable the design of FIR filters with *any* conceivable frequency response, respecting both magnitude and phase. The window method equations represent the inverse Fourier transforms of their respective ideal frequency responses. It follows that to obtain the impulse response of an arbitrary filter, we start in the frequency domain by specifying the values of its real and imaginary terms and then take the inverse Fourier transform to realise the impulse response.

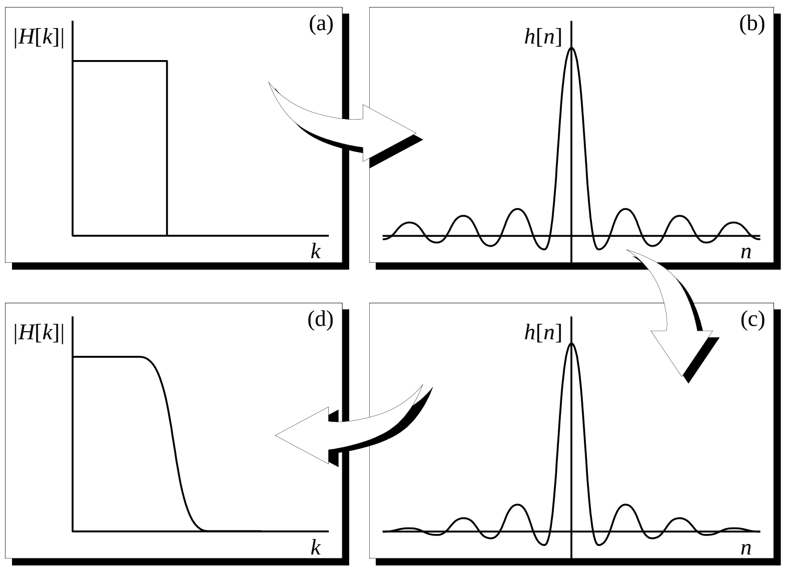


Figure 8.6. Design steps using the frequency sampling method.

We highlight the three stages in the design of FIR filters using this technique. These are:

1. The frequency response of the filter is specified in the Fourier-domain, as shown in Figure 8.6a. This may be an ideal brick-wall type, but it need not be so. It could equally be any arbitrary shape. For a linear phase filter, the frequency response is determined by setting the real terms to their intended values, and leaving the imaginary terms as zero.
2. The inverse Fourier transform is taken of the designated frequency response. This results in a time domain signal which is *not centred*, ie the right-hand part of the signal starts at $t = 0$, and the left-hand part extends backwards from the final value. Hence manipulation is necessary to centre the impulse response. Once centred, it will appear typically as in Figure 8.6b.
3. The ends of the impulse response must now be tapered to zero using a suitable window function. An impulse response thus modified is shown in Figure 8.6c. Application of the window minimises ripples in the pass and stop bands but it also increases the width of the transition zone, resulting in a frequency response shown in Figure 116.d.

For a simple linear phase pass-band filter, the frequency sampling method is encapsulated by the expression

$$h[n] = f_w[n]F^{-1}\{H[k]\} \qquad \begin{cases} H_r[k] = 1, & f_l < k < f_h \\ H_r[k] = 0, & \text{elsewhere} \\ H_i[k] = 0 \end{cases} \tag{8.7}$$

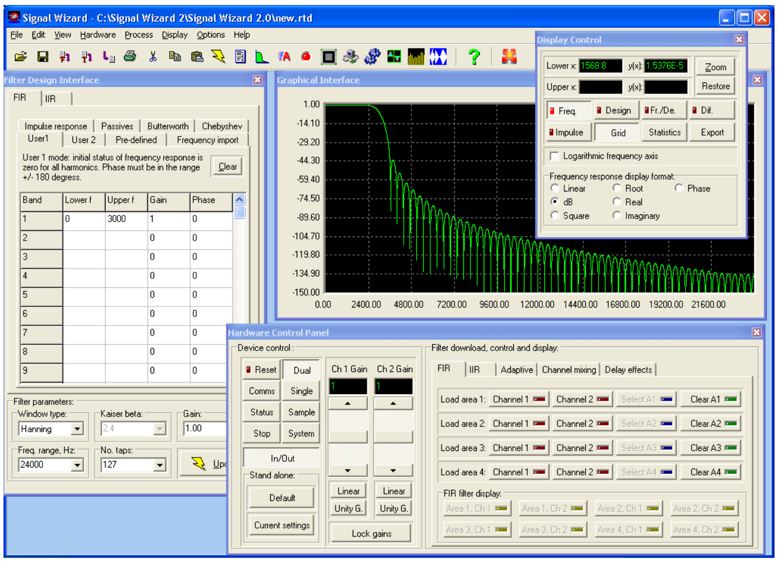
where $f_w[n]$ denotes the window function. For a filter with an arbitrary frequency response, the method is adapted so that

$$h[n] = f_w[n]F^{-1}\{H[k]\} \qquad \begin{cases} H_r[k] = a_k \\ H_i[k] = b_k \end{cases} \tag{8.8}$$

The filter design must take into account the sample rate of the signal, and the FFT length must be able to accommodate the tap requirement of the final impulse response. For example, if we needed a 711-tap FIR filter to process an audio signal sampled at 30 kHz, then the FFT would need to be 1024 points in length (at a minimum), and we would design the filter based on a Nyquist point of 15 kHz. Once the 1024-point impulse response had been obtained, it would be reduced to 711 taps by a windowing function, again designed with the appropriate sample rate in mind. The application of an appropriate window function is particularly important with the frequency sampling method, because in addition to minimising ripples as a result of truncation, it ensures that the frequency response changes smoothly as we move from one discrete harmonic to the next. To clarify this matter, imagine a filter designed using a 1024-point Fourier record, in which the Nyquist frequency was set at 10.24 kHz (ie the digital signal had been sampled at 20.48 kHz). Since 512 harmonics represent the positive frequencies, the spectral resolution of this system is 10240 / 512 = 20 Hz. When we design a filter using the frequency sampling method, we can only be sure of its conformity to our design with signal frequencies that are exact multiples of 20 Hz. In between this the response is, strictly speaking, indeterminate, *if* a rectangular window is used to truncate the impulse response. This ambiguity is ameliorated when we apply a smoothly-changing window function.

8.5 Software for arbitrary FIR design: *Signal Wizard*

The *Signal Wizard* system (Figure 8.7) comprises a software system and associated hardware module. It can calculate FIR, IIR, and adaptive filters, and run them in real time or off-line using the DSP hardware. It uses the frequency sampling method to calculate the coefficients for low-pass, high-bass, multiple band-stop, band-pass filters or arbitrary filters. Phase distortion can be completely eliminated in the filtered signal, no matter how sharp the filter is. Alternatively, arbitrary phase distortion can be introduced if this is desirable, something that we will look at in a short while. It is even possible to design deconvolution (also known as inverse filters). Again, this is something we will look at in detail later. The design package indicates the response that will be produced and the deviation from that specified, using both graphical and statistical outputs. We have already learnt that a FIR filter differs from its ideal frequency representation once we limit the number of taps and we use a window function to minimise ripple, so it is useful to be able to see these differences to optimise filter design.



Signal Wizard - C:\Signal Wizard 2\Signal Wizard 2.0\new.rtd

User 1 mode: initial status of frequency response is zero for all harmonics. Phase must be in the range +/- 180 degrees.

Band	Lower f	Upper f	Gain	Phase
1	0	3000	1	0
2			0	0
3			0	0
4			0	0
5			0	0
6			0	0
7			0	0
8			0	0
9			0	0

Filter parameters: Window type: Hanning; Kaiser beta: 2.4; Gain: 1.00; Freq. range, Hz: 24000; No. taps: 127

Lower x: 1568.8 y(x): 1.5376E-5

Frequency response display format: dB

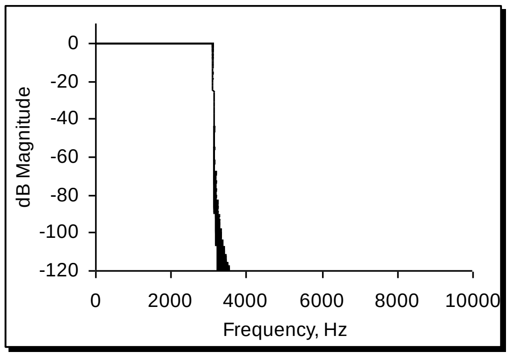
Ch 1 Gain: 1 Ch 2 Gain: 1

File Edit View Hardware Process Display Options Help
Filter Design Interface
FIR IIR
Impulse response Passives Butterworth Chebyshev
User1 User 2 Pre-defined Frequency import
Clear
Graphical Interface
1.00
-14.10
-29.20
-44.30
-59.40
-74.50
-89.60
-104.70
-119.80
-134.90
-150.00
0.00 2400.00 4800.00 7200.00 9600.00 12000.00 14400.00 16800.00 19200.00 21600.00
Display Control
Upper x: y(x):
Zoom
Restore
Freq. Design Fr./De. Dif.
Impulse Grid Statistics Export
Logarithmic frequency axis
Linear Root Phase
dB Real
Square Imaginary
Hardware Control Panel
Device control:
Reset Dual
Comms Single
Status Sample
Stop System
In/Out
Stand alone:
Default
Current settings
Linear Linear
Unity G. Unity G.
Lock gains
Filter download, control and display:
FIR IIR Adaptive Channel mixing Delay effects
Load area 1: Channel 1 Channel 2 Select A1 Clear A1
Load area 2: Channel 1 Channel 2 Select A2 Clear A2
Load area 3: Channel 1 Channel 2 Select A3 Clear A3
Load area 4: Channel 1 Channel 2 Select A4 Clear A4
FIR filter display:
Area 1, Ch 1 Area 1, Ch 2 Area 2, Ch 1 Area 2, Ch 2
Area 3, Ch 1 Area 3, Ch 2 Area 4, Ch 1 Area 4, Ch 2

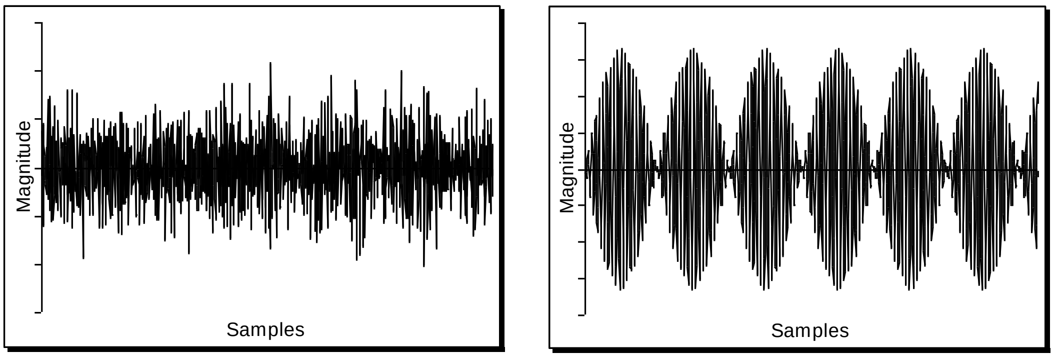
Figure 8.7. Screenshot of *Signal Wizard 2*.

If you use *Signal Wizard* to design a 3.1 kHz low-pass brick-wall filter with 2047 taps and a frequency range of 12 kHz, then its dB magnitude frequency response would appear as shown in

Figure 8.8a; the magnitude drops from 0 dB to -60 dB within approximately 60 Hz, ie this represents 0.5% of the frequency range of the system. This kind of performance is almost impossible with analogue filters. What is more, the filter is unconditionally stable and has pure linear phase. Of what use is a filter with such a sharp cut-off? Observe Figure 8.8b. Buried in this seemingly random data is a tone-burst signal; in this case, the noise was broadband, starting at 3.2 kHz. Using the brick-wall filter, the tone burst trace is recovered, shown in Figure 8.8c. This comprises a mix of two sinusoids at 2.85 kHz and 3 kHz. The reconstruction of the signal is near-perfect, with no discernable remaining effect of the broadband noise.



(a)



(b) (c)

Figure 8.8. (a) Frequency response of a 2047-tap low-pass 3.1 kHz brick wall filter; (b) noisy dual tone-burst signal, (c) recovered signal after filtering.

8.5.1 Arbitrary frequency response design

The extraordinary power of the frequency sampling method comes into its own when we wish to design filters with unusual or arbitrary shapes. Figure 8.9, for example, shows the kind of frequency response that can be designed and executed using *Signal Wizard*.

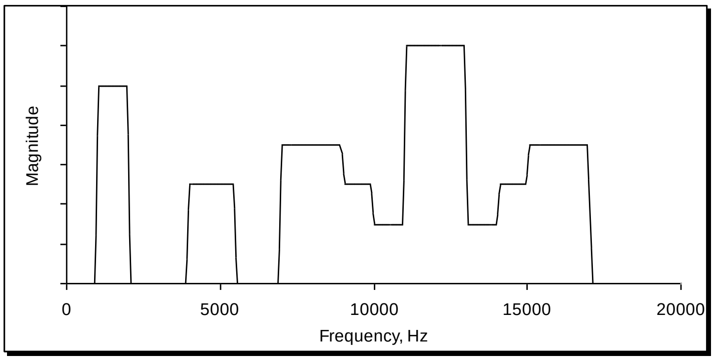
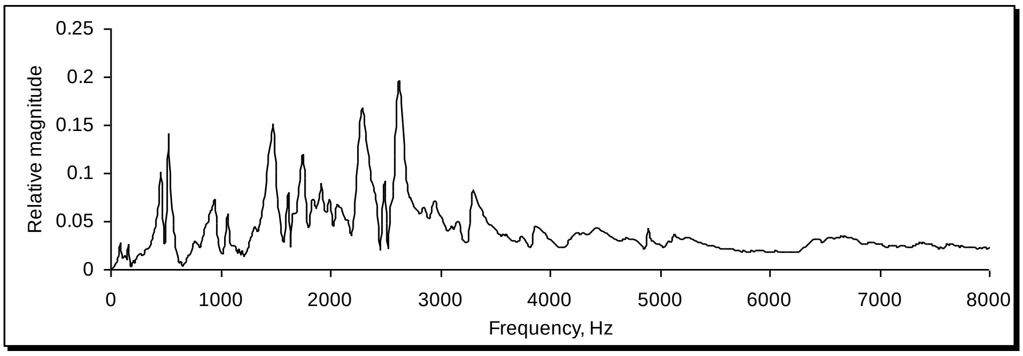


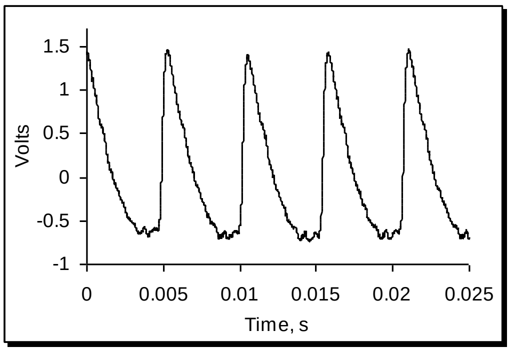
Figure 8.9. Arbitrary frequency of a filter comprising 1023-taps.

You might argue that such an arbitrary response would never be needed in practice, but this is very far from the truth. The bridge and body of a violin, for example, are essentially linear, any non-linearities being player-dependent and confined to manner in which the bow moves over the string.

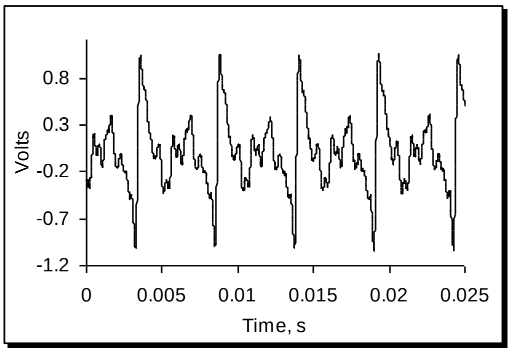
Regardless of the quality of the violin, when the bow is drawn across a string, it sticks to the hairs on the bow and is displaced. A single kink in the string travels towards the neck, is reflected, and as it passes under the bow on its return to the bridge it causes the string to slip back. The string action was first described by Helmholtz and is therefore called a Helmholtz wave. As a result of this slip-stick action, the force acting on the bridge is a simple saw tooth, which, when heard in isolation, has an unpleasant nasal quality. However, this waveform and its associated frequency response are mediated by the impulse / frequency responses of the bridge and body before being transmitted through the air; it is the contribution of these components that is responsible for bestowing to the violin its characteristic voice.



(a)



(b)



(c)

Figure 8.10. (a) Violin bridge/body frequency response; (b) pure string force signal; (c) force signal after filtering using a digital version of the bridge/body impulse response (with thanks to Jim woodhouse).

An approximate violin bridge/body impulse response can be obtained by tapping the bridge with a small instrumented hammer and using a microphone and an ADC to record the signal. This impulse response is then processed with an FFT algorithm to obtain the frequency response. A typical response is shown in Figure 8.10a. The force signal from *just* a string may be obtained using miniature accelerometers mounted on the bridge; Figure 8.10b. This is then processed digitally with the bridge/body filter to synthesise a convincing violin sound; the resulting waveform appears in Figure 8.10c. This has important implications for electronic music and instruments. Electric violins, for example, generally have no body, so in simple systems the sound produced is from the amplified raw saw tooth. By feeding this into a DSP system programmed with a sophisticated FIR equivalent of an acoustic bridge/body impulse response, we can get the electric violin to emulate the sound of an acoustic (and very expensive) counterpart, at minimal cost.

We are not restricted to violins; since most acoustic instruments are essentially linear systems, they also lend themselves to this kind of emulation. In principle, the arbitrary frequency response technique can be used to mimic *any* linear system, a few examples being analogue networks, phase shift systems, ultrasonic transducers, microphones, loudspeakers, and acoustic room responses.

8.5.2 Linear-phase analogue equivalents using FIR methods

One of the criticisms that is sometimes levelled at FIR designs is that they cannot replicate the responses of electronic analogue filters (unlike IIR types). In fact, this is quite wrong; to show how this can be achieved, we need to study the frequency response of a given analogue filter. For example, the low-pass Butterworth has a magnitude frequency response given by

$$|H(f)| = \frac{1}{\sqrt{1+\left(\dfrac{f}{f_c}\right)^{2n}}} \tag{8.9}$$

Where f_c represents the cut-off frequency, i.e. this is the -3 dB point, and n is the order of the filter. As we stated, a low-pass Butterworth has a roll-off of 6n dBs per octave. To realise this as a linear phase FIR structure, all we have to do is to employ the frequency sampling method, in which we establish the magnitudes of the real (cosine) harmonics by simply applying Equation 8.9. Next, we take the inverse transform which yields the linear-phase impulse response. Figure 8.11, for example, shows a 40th order design with a cut-off of 4 kHz, i.e. this has a roll-off of 240 dBs per octave!

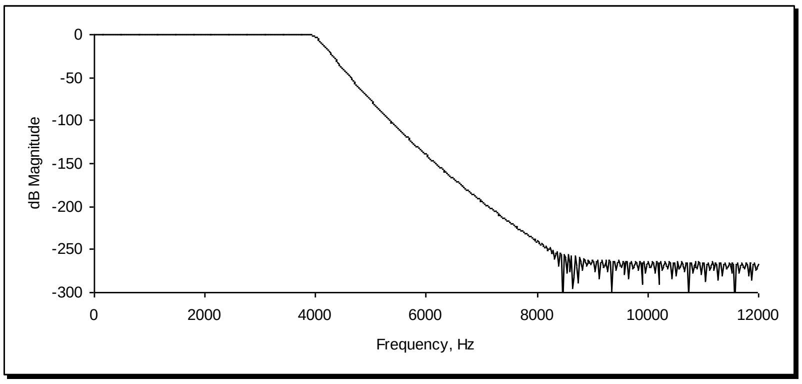


Figure 8.11. A 40th order Butterworth, 4 kHz cut-off. Since it is an FIR filter it has pure linear phase.

Observe that the filter's performance degrades after 8 kHz, not because there is anything wrong with the method, but because we have reached the quantisation noise limit of the algorithm – remember that -240 dB is equal to 10^{-12}. It would be extremely difficult, if not impossible to get this kind of transition zone performance with an analogue filter. The component tolerances required would be tiny indeed, and careful circuit layout would be critical to offset the very real risk of instability. However well we designed the circuit, there would be nothing we could do about the phase distortion introduced within the transition zone. In contrast, a digital off-line or real-time system would execute this filter with complete ease, with the added bonus of complete phase linearity.

If we were being fastidious, we might argue that we have not *quite* reproduced the behaviour of an analogue filter, because although it has the same frequency response, the impulse response has a different shape (here it is symmetrical). This is undeniably true, even though it is usually the case that the frequency response, and not the impulse response, is uppermost in the minds of filter designers and users. If we really want an FIR filter to have the same frequency and impulse response as its analogue counterpart, we employ a technique called IIR to FIR translation. This technique is much simpler than it sounds, and we will explore it in detail when we have covered the design of IIR filters.

8.5.3 Phase change filters

We will now look at how to design FIR filters with defined phase responses at specific frequencies. This is actually very simple, and we still make use of the frequency sampling method. The difference now is that, if we want to have a phase other than zero at some frequency, then the imaginary terms for those harmonics are no longer left at zero, since the phase is determined by the arctangent of the imaginary terms divided by the real. In other words, we implement Equation 8.8. As an example, take a look at Figure 8.12. This represents a filter with a pass band between 0 and

511 Hz. However, at 256 Hz there is a sharp notch. This is because the phase is zero for all frequencies below 256 Hz, but has been inverted (180°) for frequencies between 256 and 511 Hz.

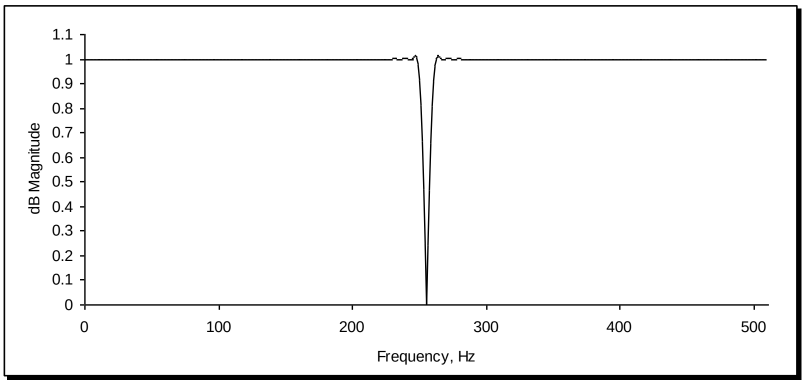


Figure 8.12. All-pass filter, with phase inversion beyond 256 kHz.

This is not an artefact of the design, but a consequence, again, of limiting a theoretically infinite response with a finite number of coefficients. The smaller their number, the wider the gap becomes. The drops will always be there, because we are completely inverting the signal of the upper band band. If you were using this filter in practice, feeding it with a swept sine wave, as the frequency approached and then exceeded 256 kHz, the filter would at some point flip the polarity of the sinusoid, as demanded by the design, ie it would necessarily pass through a null point. The longer the FIR filter, the narrower this “flip region” becomes.

We can design filters with any phase response we like using this method. Filters that adjust the phase and not the magnitude of a signal are sometimes termed *all-pass filters*, and are widely used for creating audio effects, for modulation and demodulation purposes. One of the most important all-pass filters in this regard is the Hilbert transform. This is a filter that shifts every harmonic by 90° or $\pi/2$, and as a consequence, the shape of the filtered signal is no longer preserved. Figure 8.13 shows a typical impulse and frequency response of a Hilbert transform filter.

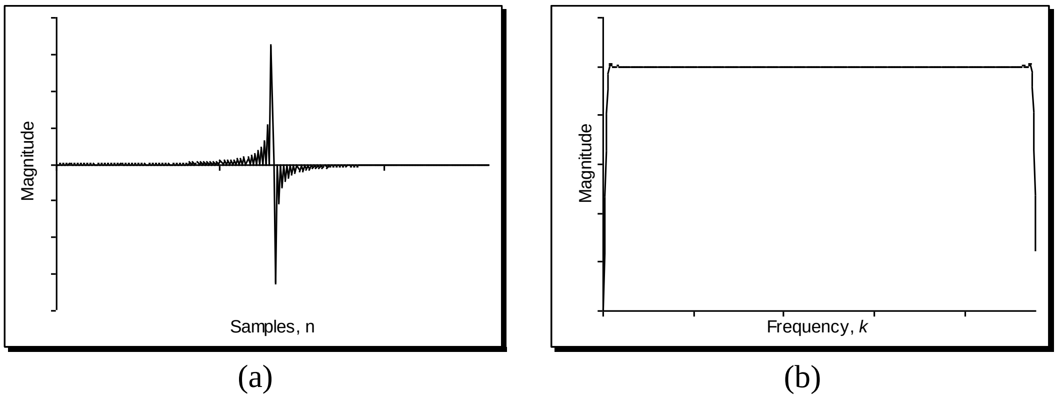


Figure 8.13. (a) Impulse and (b) frequency response of a Hilbert transform.

Because the filter shifts everything by 90°, it must have odd symmetry, as indicated by Figure 8.13a. The *mean* value of a signal filtered by an FIR structure is equal to the sum of the coefficients. In this case, that sum is zero. As a consequence, the gain of a Hilbert transform *must* be zero at the DC frequency, or its zeroth harmonic. This is confirmed by the frequency response shown in Figure 8.13b. Moreover, the magnitude of the response also falls to zero as it approaches the Nyquist point. It is often said that the human hearing system is insensitive to the phase of a signal. This is only partially true; for example, if we listen to a sequence of two signals, the second being phase shifted with respect to the first, then it is difficult or impossible to detect any change in timbre. This is even the case where the phase changes are extreme. However, if the two signals are played simultaneously, then the auditory effect can be quite dramatic. Phase shifting of signals in this manner (together with time shifting), forms the basis of surround sound and virtual surround sound systems.

8.6. Inverse filtering and signal reconstruction

Inverse filtering, or deconvolution as it is also known, is a very important branch of DSP and signal processing generally. There are various ways of realising inverse filters, and although the mathematics of some of these are rather labyrinthine, the basic idea of deconvolution is straightforward enough. Imagine a signal that started as a nice crisp waveform, but during transmission along a cable, it arrived at its destination in a smeared form, due to the impedance properties of the transmission medium. The cable can therefore be viewed as a filter, with a given frequency response. If we know this, we can in principle process the smeared signal with the *inverse* of the cable's frequency response, thereby reconstructing the original signal. If we say that $x(t)$ is the original signal, $y(t)$ is the degraded signal, and $h(t)$ is the impulse response of the degradation system (the cable in the above example), then as we have already seen, the output is the convolution of the input signal with the impulse response, i.e.

$$y(t) = x(t) * h(t) \tag{8.10}$$

Convolution in the one domain is equivalent to multiplication in the other, so:

$$y(t) = x(t) * h(t) = F^{-1}\{Y(\omega)\} = F^{-1}\{X(\omega)H(\omega)\} \tag{8.11}$$

A naïve approach to recover $x(t)$ would involve the Fourier transform of both $y(t)$ and $h(t)$, a division and an inverse Fourier transform of this division to yield the reconstructed expression in the time domain, ie

$$x(t) = F^{-1}\left[\frac{Y(\omega)}{H(\omega)}\right] \tag{8.12}$$

Such a simple scheme is, however, rarely successful in practice. The reason is that convolution usually takes the form of a low-pass filter, reducing the magnitude of the high frequency components. Deconvolution in this case attempts to reverse the process by boosting their magnitudes in the division process. However, if they have fallen to levels below the signal-to-noise of the system, then at some frequency ω_0 the division of $Y(\omega_0)$ by a very small value generates a large magnitude component at that frequency which is purely noise. In extreme cases, the value of $H(\omega_0)$ may be zero, in which case an ill-conditioned division results. The presence of noise is a significant factor in determining the efficacy of the inverse filtering procedure; if, as is often the case, the noise is independent of the signal, then Equation 8.10 becomes

$$y(t) = x(t) * h(t) + s(t) \tag{8.13}$$

where $s(t)$ is a noise term. The deconvolution expression now becomes

$$x(t) = F^{-1}\left[\frac{Y(\omega)}{H(\omega)} - \frac{S(\omega)}{H(\omega)}\right] \tag{8.14}$$

Many sophisticated methods exist for deconvolution, which is often performed in the frequency domain; a time-windowed sample of the output function is transformed to the frequency domain, divided by the modified frequency response of the system, and inverse transformed back to the time domain. However, such an approach is clearly not suitable for real-time operation, where all the processing must be conducted in the time domain. Inspection of Equation 8.12 shows that to conduct deconvolution in the time domain, the impulse response of the inverse filter must be pre-computed, and then applied to the output signal using the normal time convolution equation. Typically,

$$\tilde{h}(t) = F^{-1}\left[\frac{1}{H(\omega) + s}\right] \tag{8.15}$$

hence

$$x(t) = y(t) * \tilde{h}(t) \tag{8.16}$$

Signal Wizard has a facility to perform real-time inverse deconvolution or inverse filtering based on this simple approach. The time-domain version of this inverted frequency response is computed automatically and can be used to process signals in the normal way.

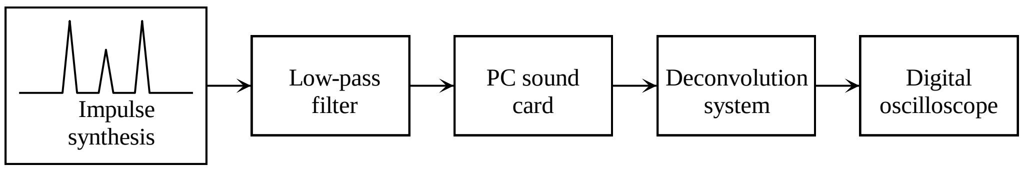


Figure 8.14. Inverse filter (deconvolution) evaluation system.

Figure 8.14 depicts how such a simple inverse filtering system may be evaluated in practice. First, an arbitrary waveform is generated; here, the signal comprises three impulsive values. Next, the signal is degraded using a suitable low-pass FIR filter. The inverse of this filter is now calculated, and used to process the degraded signal. Figure 8.15 shows some typical results obtained using real-time processing. In part (a), we see a signal comprising three main impulsive values, synthesised digitally and played through a computer soundcard. The ripples present are not a function of the system, but of the impulse generation software and the frequency response of the sound card. A new file was then generated by convolving this signal with a digital low pass filter. This produced a smeared signal, shown in part (b), which was again transmitted to the system. The plot confirms significant loss of detail, with only two very broad peaks present in the signal. The frequency response of the low-pass filter was then inverted with a suitable offset *s*. The degraded signal was then played through the soundcard, and processed in real-time using the inverse filter. The reconstructed signal appears in part (c). Not only has the temporal relationship between the three values been precisely restored, so too in large measure have their amplitudes. The simple technique outlined will not always be so successful, often because of subtleties associated with noise variation and non-linear effects of the so-called "blur function". Hence, many other approaches exist, some of which rely on mathematical estimates of the noise.

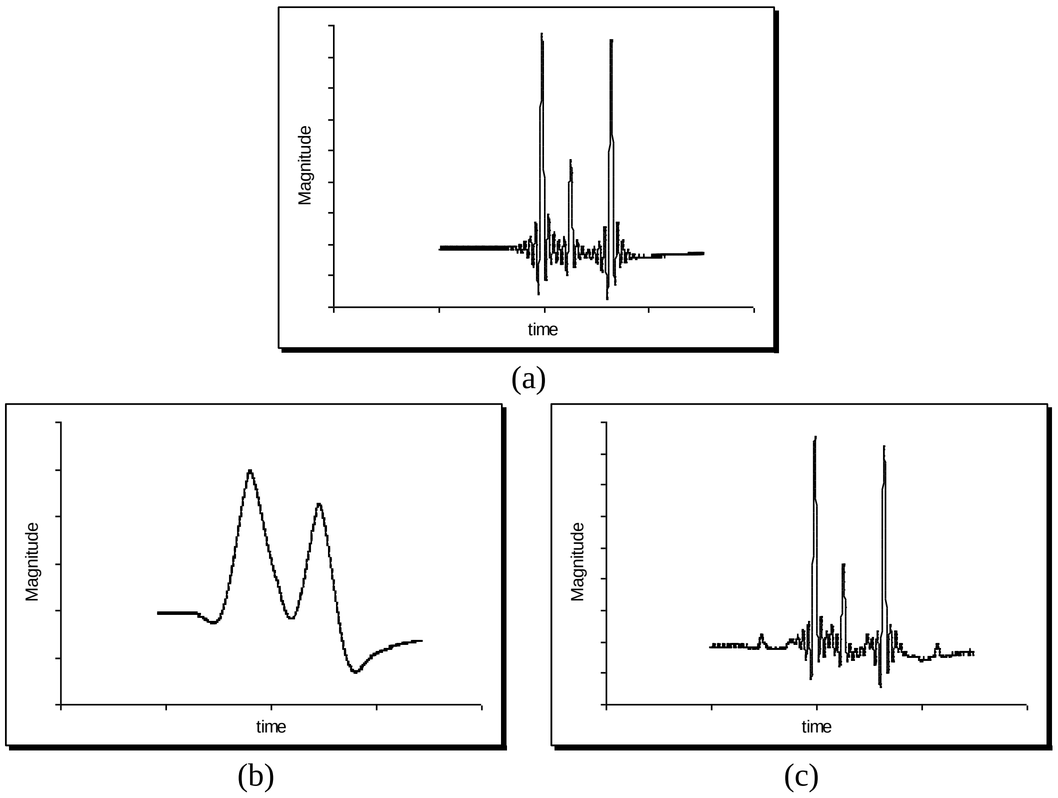


Figure 8.15. Results from a real-time deconvolution system. (a) original signal, containing three major peaks; (b) distorted signal after low-pass filtering; (c) reconstructed signal after processing with an inverse filter.