P. Starič, E. Margan:

Wideband Amplifiers

Part 7:

Algorithm Application Examples

Any computer program can be reduced by at least one command line.

Any computer program has at least one command line with an error.

...

Any computer program can eventually be reduced
to a single command line, having at least one error.

(Conservative extrapolation of Murphy's Law to computer programming)

(blank page)

Contents:

List of Figures:

List of Routines:

7.0 Introduction

In Part 6 we have developed a few numerical algorithms that will serve us as the basis of system analysis and synthesis. We have shown how simple it is to implement the analytical expressions related to the various aspects of system performance into compact, fast executing computer code which reduces the tedious mathematics to pure routine. Of course, a major contribution to this easiness was provided by the programming environment, a high level, maths–oriented language called Matlab™ (Ref. [7.1]).

As wideband amplifier designers, we want to be able to accurately predict amplifier performance, particularly in the time-domain. With the algorithms developed, we now have the essential tools to revisit some of the circuits presented in previous parts, possibly gaining a better insight into how to put them to use in our new designs eventually.

But the main purpose of Part 7 is to put the algorithms in a wider perspective. Here we intentionally use the term ‘system’, in order to emphasize the high degree of integration present in modern electronics design, which forces us to abandon the old paradigm of adding up separately optimized subsystems into the final product; instead, the design process should be conceived to optimize the total system performance from the start. As more and more digital processing power is being built in to modern products, the analog interface with the real world needs to be given adequate treatment on the system level, so that the final product eventually becomes a successful integration of both the analog and the digital world.

Now, we hear some of you analog circuit designers asking in a low voice “why do we need to learn any of this digital stuff?” The answer is that digital engineers would have a hard time learning the analog stuff, so there would be no one to understand the requirements and implications of a decent AD or DA interface. On the other hand, for an analog engineer learning the digital stuff is so simple, almost trivial, and it pays back well with better designs, and it acquires for you the respect due from fellow digital engineers.

(blank page)

7.1 Using Convolution: Response to Arbitrary Input Waveforms

7.1.1 From Infinitesimal to Discrete Time Integration

The time-domain algorithms that we have developed in Part 6 gave us the system response to two special cases of input signal: the unit-area impulse and the unit-amplitude step. Here we will consider the response to any type of input signal, provided that its application will not exceed neither the input nor the output system capabilities. In technical literature this is known as the BIBO-condition[1]. And, of course, we are still within the constraints of our initial LTIC-conditions[2].

As we have seen in Part 1, Sec. 1.14, the system's time domain response to an arbitrary input signal can be calculated in two ways:

a) by transforming the input signal to complex frequency domain, multiplying it by the system transfer function and transforming the result back to the time domain;

b) directly in time domain by the convolution integral.

A short reminder of the convolution integral definition and the transcription from differential to difference form is in order here. Let $x(t)$ be the time domain signal, presented to the input of a system being characterized by its impulse response $f(t)$. The system output can then be calculated by convolving $x(t)$ with $f(t)$:

$$y(t) = \int_{t_0}^{t_1} f(\tau - t)\, x(t)\, dt \tag{7.1.1}$$

where τ is a fixed time constant, its value chosen so that $f(t)$ is time reversed. Usually, it is sufficient to make τ large enough to allow the system's impulse response $f(t)$ to completely relax and reach the steady state again (not just the first zero-crossing point!) with a tolerance of some 0.01 % or less.

If $x(t)$ was applied to the system at t_0, then this can be the lower limit of integration. Of course, the time scale can always be renormalized so that $t_0 = 0$. The upper integration limit, labeled t_1, can be wherever needed, depending on how much of the input and output signal we are interested in.

Now, in Eq. 7.1.1 dt is implicitly approaching zero, so there would be an infinite number of samples between t_0 and t_1. Since our computers have a limited amount of memory (and we have a limited amount of time!) we must make a compromise between the sampling rate and the available memory length and adjust them so that we cover the signal of interest with enough resolution in both time and

[1] Bounded input → bounded output. This property is a consequence of our choice of basic mathematical assumptions; since our math tools were designed to handle an infinite amount of infinitesimal quantities, BIBO is the necessary condition for convergence. However, in the real analog world, we are often faced with UBIBO requirements (unbounded input), i.e., our instrumentation inputs must be protected from overdrive. Interestingly, the inverse of BIBO is in widespread use in the computer world, in fact, any digital computer is a GIGO type of device (garbage in → garbage out; unbounded!).

[2] Linearity, Time Invariance, Causality. Although some engineers consider oscillators to be 'acausal', there is always a perfectly reasonable cause why an amplifier oscillates, even if we fail to recognize it at first.

amplitude. So if M is the number of memory bytes reserved for $x(t)$, the required sampling time interval is:

$$\Delta t = \frac{t_1 - t_0}{M} \tag{7.1.2}$$

Then, if Δt replaces dt, the integral in Eq. 7.1.1 transforms into a sum of M elements, $x(t)$ and $y(t)$ become vectors `x(n)` and `y(n)`, where `n` is the index of a signal sample location in memory, and $f(\tau - t)$ becomes `f(m-n)`, with `m=length(f)`, resulting in:

$$\texttt{y(n)} = \sum_{\texttt{n=1}}^{\texttt{M}} \texttt{f(m-n)*x(n)} \tag{7.1.3}$$

Here Δt is implicitly set to 1, since the difference between two adjacent memory locations is a unit integer. Good book-keeping practice, however, recommends the construction of a separate time scale vector, with values from t_0 to t_1, in increments of Δt between adjacent values. All other vectors are then plotted against it, as we have seen it done in Part 6.

7.1.2 Numerical Convolution Algorithm

In Part 1 we have seen that solving the convolution integral analytically can be a time consuming task, even for a skilled mathematician. Sometimes, even if $x(t)$ and $f(t)$ are analytic functions, their product need not be elementarily integrable in the general case. In such cases we prefer to take the $\mathcal{L}$ transform route; but this route can sometimes be equally difficult. Fortunately numerical computation of the convolution integral, following Eq. 7.1.3, can be programmed easily:

```
function y=vcon(f,x)
%VCON   Convolution, step-by-step example. See also CONV and FILTER.
%
%      Call :       y=vcon(f,x);
%      where:       x(t) --> the input signal
%                   f(t) --> the system impulse response
%                   y(t) --> the system response to x(t) by convolving
%                            f(t) with x(t).
%      If  length(x)=nx  and  length(f)=nf, then length(y)=nx+nf-1.

%      Erik Margan, 861019, Last editing 890416; Free of copyright!

                          % force f to be the shorter vector :
if length(f) > length(x)
      xx=x; x=f; f=xx;    % exchange x and f via xx
      clear xx
end
nf=length(f);             % get the number of elements in x and f
nx=length(x);
f=f(:).';                 % organize x and f as single-row vectors
x=x(:).';
y=zeros(2,nx+nf-1);       % form a (2)-by-(nx+nf-1) matrix y, all zeros
y(1,1:nx)=f(1)*x;         % first row of y: multiply x by f(1)
for k=2:nf                % second row: multiply and shift (insert 0)
      y(2, k-1:nx+k-1)=[0, f(k)*x];
                          % sum the two rows column-wise and
      y(1,:)=sum(y);      % put result back into first row
end                       % repeat for all remaining elements of f;
y=y(1,:);                 % the result is the first row only.
```

To get a clearer view of what the VCON routine is doing, let us write a short numerical example, using a 6-sample input signal and a 3-sample system impulse response, and display every intermediate result of the matrix y in VCON:

```
x=[0 1 3 5 6 6];    f=[1 3 -1];    y=vcon(x,f);

% initialization - all zeros, 2 rows, 6+3-1 columns:
    0     0     0     0     0     0     0     0
    0     0     0     0     0     0     0     0
% step 1: multiply x by the first sample of f, f(1)=1 and
% insert it into the first row:
    0     1     3     5     6     6     0     0
    0     0     0     0     0     0     0     0
% step 2: multiply x by the second sample of f, f(2)=3,
% shift it one place to the right by adding a leading zero and
% insert it into the second row:
    0     1     3     5     6     6     0     0
    0     0     3     9    15    18    18     0
% step 3: sum both rows vertically, put the result in the first row
    0     1     6    14    21    24    18     0
    0     0     3     9    15    18    18     0
% iterate steps 2 and 3, each iteration using the next sample of f:
    0     1     6    14    21    24    18     0
    0     0     0    -1    -3    -5    -6    -6

    0     1     6    13    18    19    12    -6
    0     0     0    -1    -3    -5    -6    -6
% after 2 iterations (because f is only 3 samples long)
% the result is the first row of y:
    0     1     6    13    18    19    12    -6
% actually, the result is only the first 6 elements:
    0     1     6    13    18    19
% since there are only 6 elements in x, the process assumes the rest
% to be zeros. So the remaining two elements of the result represent
% the relaxation from the last value (19) to zero by the integration
% of the system impulse response f.

% Basically, the process above does the following:
% (note the reversed sequence of f)

                0  1  3  5  6  6
          -1  3  1
(↓*) ==>        0                 (→+) ==> 0  --> y(1)

                0  1  3  5  6  6
             -1  3  1
(↓*) ==>        0  1              (→+) ==> 1  --> y(2)

                0  1  3  5  6  6
               -1  3  1
(↓*) ==>        0  3  3           (→+) ==> 6  --> y(3)

                0  1  3  5  6  6
                  -1  3  1
(↓*) ==>        0 -1  9  5        (→+) ==> 13  --> y(4)

% ......... etc.
```

For convolution Matlab has a function named CONV, which uses a built in FILTER command to run substantially faster, but then the process remains hidden from the user; however, the final result is the same as with VCON. Another property of Matlab is the matrix indexing, which starts with 1 (see the lower limit of the sum

symbol in Eq. 7.1.3), in contrast to most programming languages which use memory 'pointers' (base address + offset, the offset of the array's first element being 0).

7.1.3 Numerical Convolution Examples

Let us now use the VCON routine in a real life example. Suppose we have a gated sine wave generator connected to the same 5th-order Butterworth system which we inspected in detail in Part 6. Also, let the Butterworth system's half power bandwidth be 1 kHz, the generator frequency 1.5 kHz, and we turn on the gate in the instant the signal crosses the zero level. From the frequency response calculations, we know that the forced response amplitude (long after the transient) will be:

```
[z,p]=buttap(5);                        % Butterworth 5-pole system
Aout=Ain*abs(freqw(z,p,1500/1000));  % output steady-state amplitude
```

Here `p` are the poles of the normalized 5th-order Butterworth system, `z` is an empty matrix (no zeros); the signal frequency is normalized to the system's cut off.

But how will the system respond to the signal's 'turn on' transient? We can simulate this using the algorithms we have developed in Part 6 and VCON:

```
fh=1000;                % system half-power bandwidth, 1kHz
fs=1500;                % input signal frequency, 1.5kHz
t=(0:1:300)/(50*fh);    % time vector, 20us delta-t, 6ms range
nt=length(t);

[z,p]=buttap(5);        % 5th-order Butterworth system
p=2*pi*fh*p;            % denormalized system poles
Ir=atdr(z,p,t,'n');     % system impulse-response

d=25;                   % switch-on delay, 25 time-samples
                        % make the input signal :
x=[zeros(1,d), sin(2*pi*fs*t(1:nt-d))];

y=vcon(Ir,x);           % convolve x with Ir ;

A=nt/(2*pi*fh*max(t)); % denormalize amplitude of Ir for plot

                        % plot the input, the system impulse response
                        % and the convolution result :
plot( t*fh, x, '-g', ...
      t*fh, [zeros(1,d), Ir(1:nt-d)*A],'-r', ...
      t*fh, y(1:nt), '-b')
xlabel('t [ms]')
```

The convolution result, compared to the input signal and the system impulse response, is shown in Fig. 7.1.1 .

Note that we have plotted only the first `nt` samples of the convolution result; however, the total length of `y` is `length(x)+length(Ir)-1`, or one sample less that the sum of the input signal and the system response lengths. The first `length(x)=nt` samples of `y` represent the system's response to `x`, whilst the remaining `length(Ir)-1` samples are the consequence of the system relaxation: since there are no more signal samples in `x` after the last point `x(nt)`, the convolution assumes that the input signal is zero and calculates the system relaxation from the last signal value.

This is equivalent to a response caused by an input step from `x(nt)` to zero. So if we are interested only in the system response to the input signal, we simply limit the response vector to the same length as was the input signal. Also, in the general case the length of the system's impulse response vector, `nr=length(Ir)` , does not have to be equal to the input signal vector length, `nx=nt`. In practice, we often make `nr<<nx` , but `Ir` should be made long enough to include the system relaxation to a level very close to zero, as only then will the sum of all elements of `Ir` not differ much from the system gain.

There is, however, an important difference in the plot and the calculation, that must be explained. The impulse response which we obtained from Butterworth system poles was normalized to represent a unity gain system, since we want to see the frequency dependence on the output amplitude by comparing the input and output signals. Thus our system should either have a gain of unity, or the output should be normalized to the input in some other way (e.g., if the gain is known, we could either divide the output signal by the gain, or multiply the input signal). But the unity gain normalized impulse response would be too small in amplitude, compared to the input signal, so we have plotted the ideal impulse response.

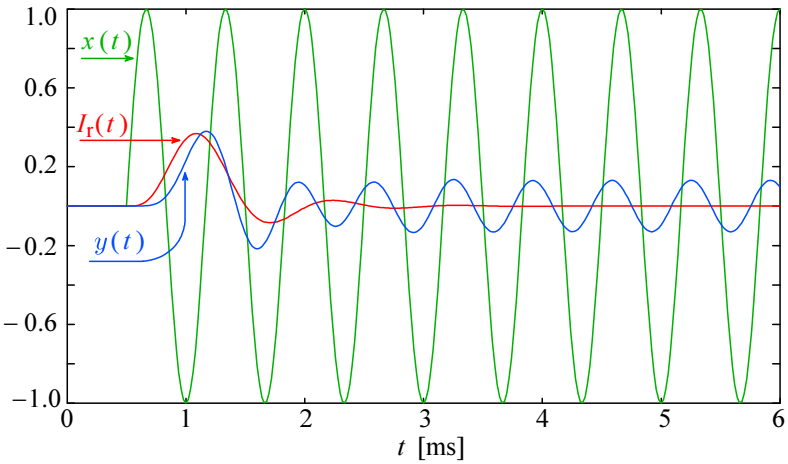

Fig. 7.1.1: Convolution example: response $y(t)$ to a sine wave $x(t)$ switched-on into a 5th-order Butterworth system, whose impulse-response is $I_r(t)$, shown here as the ideal response (instead of unity gain); both are delayed by the same switch–on time (0.5 ms). The system responds by phase shifting and amplitude modulating the first few wave periods, reaching finally the forced ('steady state') response.

Can we check whether our routine works correctly?

Apart from entering some simple number sequences as before, we can do this by entering an input signal for which we have already calculated the result in a different way, say, the unit step (see Fig. 6.5.1, Part 6). By convolving the impulse response with the unit step, instead of the sine wave, we should obtain the now known step response:

```
% continuing from above:
h=[zeros(1:d), ones(1:nt-d)];       % h(t) is the unit step function
y=vcon(Ir,h);                       % convolve h with Ir
plot( t*fh, h, '-g', ...
      t*fh, [zeros(1,d), Ir(1:nt-d)*A],'-r', ...
      t*fh, y(1:nt), '-b')
xlabel('t [ms]')
```

The resulting step response, shown in Fig. 7.1.2, should be identical to that of Fig. 6.5.1, Part 6, neglecting the initial 0.5 ms (25 samples) time delay and the different time scale:

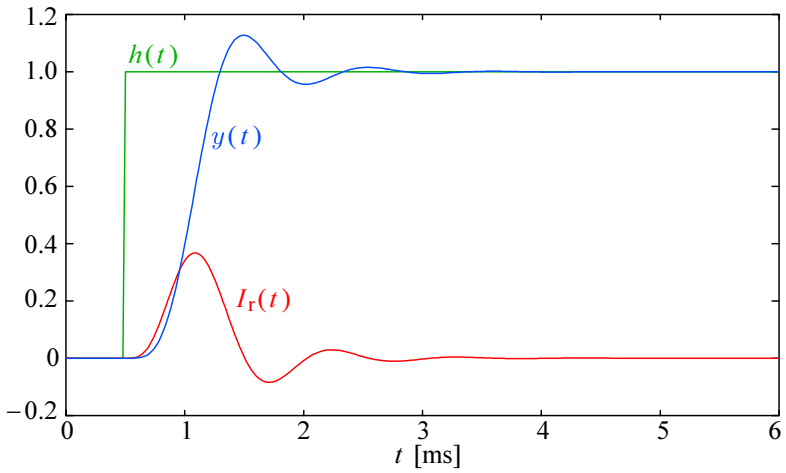

Fig. 7.1.2: Checking convolution: response $y(t)$ of the 5th-order Butterworth system to the unit step $h(t)$. The system's impulse response $I_r(t)$ is also shown, but with its ideal size (not unity gain). Apart from the 0.5 ms (25-samples) time delay and the time scale, the step response is identical to the one shown in Part 6, Fig.6.5.1.

We can now revisit the convolution integral example of Part 1, Sec. 1.15, where we had a unit-step input signal, fed to a two-pole Bessel-Thomson system, whose output was in turn fed to a two-pole Butterworth system. The commands in the following window simulate the process and the final result of Fig. 1.15.1 . But this time, let us use the TRESP (Part 6) routine for the frequency to time domain transform. See the result in Fig. 7.1.3 and compare it to Fig. 1.15.1g .

```
[z1,p1]=bestap(2,'t'); % Bessel-Thomson 2nd-order system poles
[z2,p2]=buttap(2);     % Butterworth 2nd-order system poles
N=256;                 % number of samples
m=4;                   % set the bandwidth factor
w=(0:1:N-1)/m;         % frequency vector, w(m+1)=1 ;
F1=freqw(p1,w);        % Bessel-Thomson system frequency response
F2=freqw(p2,w);        % Butterworth system frequency response

[S1,t]=tresp(F1,w,'s');% step-response of the Bessel-Thomson system
I2=tresp(F2,w,'u');    % unity-gain Butterworth impulse response ;
                       % both have the same normalized time vector
d=max(find(t<=15));    % limit the plot to first 15 time units
I2=I2(1:d);            % limit the I2 vector length to d

     % convolution of Bessel-Thomson system step-response with
     % the first d points of the Butterworth impulse response :
y=vcon(I2,S1);

A=N/(2*pi*m*max(t));   % amplitude denormalization for I2
     % plot first d points of all three responses vs. time :
plot( t(1:d), S1(1:d), '-r',...
      t(1:d), I2(1:d)*A, '-g',...
      t(1:d),  y(1:d), '-b' )
xlabel('Time [s]')
```

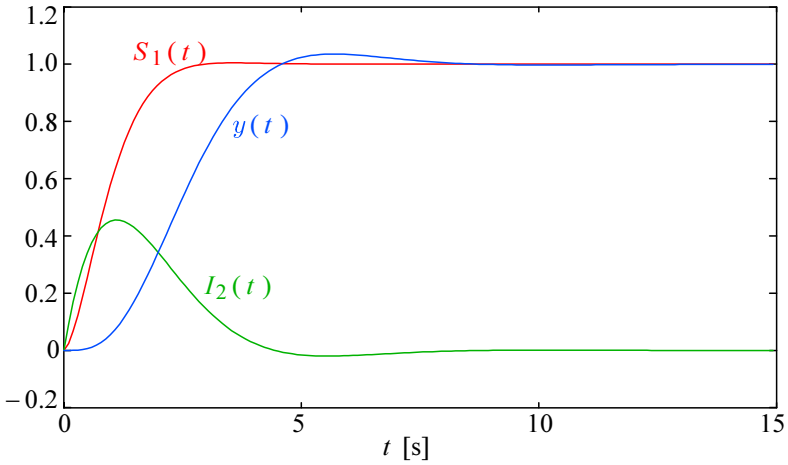

Fig. 7.1.3: Convolution example of Part 1, Sec. 1.15. A Bessel–Thomson 2-pole system step response $S_1(t)$ has been fed to the 2-pole Butterworth system and convolved with its impulse response $I_2(t)$, resulting in the output step response $y(t)$. Compare it with Fig. 1.15.1g.

The VCON function is a lengthy process. On a 12 MHz AT-286 PC, which was used for the first experiments back in 1986–7, it took more than 40 s to complete the example shown above, but even with today's fast computers there is still a noticeable delay. The Matlab CONV routine is much faster.

The reader might question the relevance of the total calculation time, since, for the purpose of a circuit design aid, anything below 1 s should be acceptable (this is comparable with the user's reaction time, when making a simple go/no go assessment of the result). However, imagine an automated optimization program loop, adjusting the values of some 10–20 circuit components. Such a loop might take hundreds or even thousands of executions before reaching satisfactory performance criteria, so a low routine time would be welcome. Moreover, if the routine will eventually be implemented in hardware to acquire and process the signal in real time, a low routine time is of vital importance. For example, in order to continuously process a 16 bit stereo audio stream, divided into 1024 sample chunks, using a 32 word long filter impulse response, the total routine calculation time must be less than 250 µs.

In some cases, particularly with long signal sequences (`N > 1000`), it could be interesting to take the Fourier transform route, numerically.

Here is an example using a signal recorded by a nuclear magnetic resonance imaging (MRI) system. The MRI RF signal is very weak (< 1 mV), so the detection is noisy and there is some interference from another source. We shall try to clean it by using a 5th-order Bessel-Thomson digital filter with a unity gain and a 1 MHz cut off:

```
load R.dat % load the recorded signal from a file "R.dat"
N=length(R);            % total vector length, N=2048 samples
Tr=102.4e-6;            % total record time 102.4 us
dt=Tr/N;                % sampling time interval, 50 ns
t=dt*(0:1:N-1);         % time vector reconstruction

% plot the first 1200 samples of the recorded signal
plot(t(1:1200,R(1:1200),'-g')
xlabel('Time [\mus]')  % input signal R, fist 60us, see Fig.7.1.4
G=fft(R);               % G is the FFT spectrum of R
G=G(1:N/2);             % use only up to the Nyquist freq. ( 10 MHz )
f=(1:1:N/2)/dt;         % frequency vector reconstructed
```

```
[z,p]=bestap(5,'n');    % 5th-order Bessel filter poles
p=p*2*pi*1e+6;          % half-power bandwidth is 1 MHz
F=freqw(z,p,2*pi*f);    % filter frequency response

% multiplication in frequency is equal to convolution in time:
Y=F.*G;                 % output spectrum

x=max(find(f<=8e+6));   % plot spectrum up to 8 MHz
M=max(abs(G));          % normalize the spectrum to its peak value
plot( f(1:x), abs(F(1:x)),    '-r', ...
      f(1:x), abs(G(1:x))/M, '-g', ...
      f(1:x), abs(Y(1:x))/M, '-b' )
xlabel('Frequency [MHz]'), ylabel('Normalized Magnitude')
                        % see Fig.7.1.5

y=2*(real(fft(conj(Y)))-1)/(N/2);  % return to time domain

a=max(find(t<=5e-5));
b=min(find(t>=20e-6));
plot( t(a:b), g(a:b), '-g', t(a:b), y(a:b), '-b' )
xlabel('Time [\mus]')   % see Fig.7.1.6
```

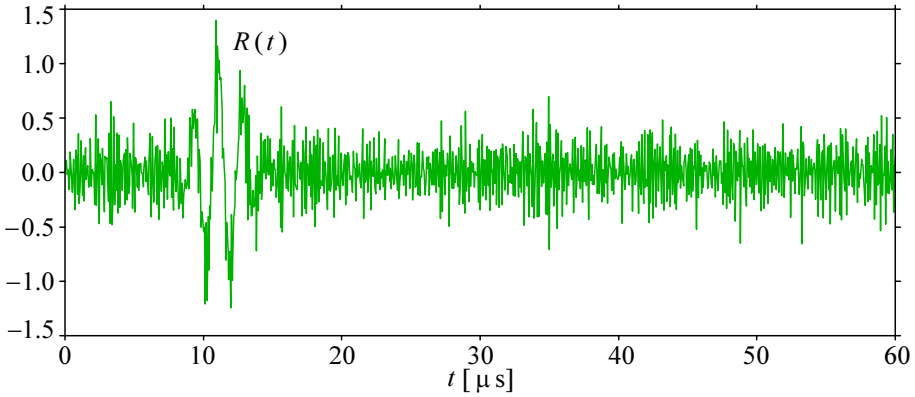

Fig. 7.1.4: Input signal example used for the spectral-domain convolution example (first 1200 samples of the 2048 total record length)

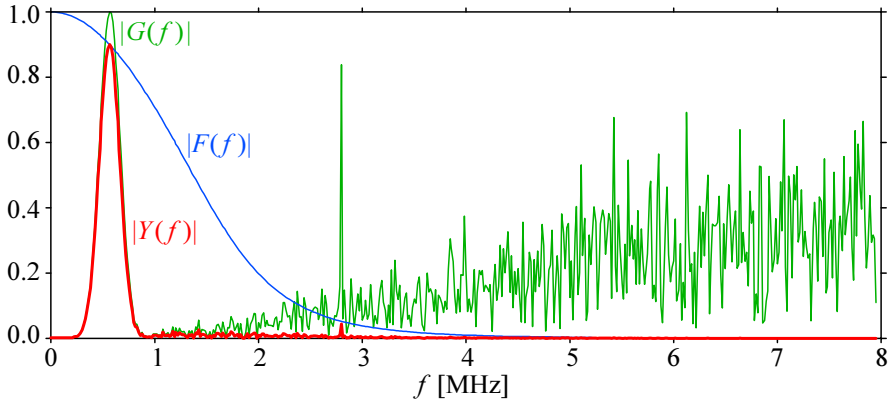

Fig. 7.1.5: The spectrum $G(f)$ of the signal in Fig. 7.1.4a is multiplied by the system's frequency response $F(f)$ to produce the output spectrum $Y(f)$. Along with the modulated signal centered at 560 kHz, there is a strong 2.8 MHz interference from another source and a high level of white noise (rising with frequency), both being substantially reduced by the filter.

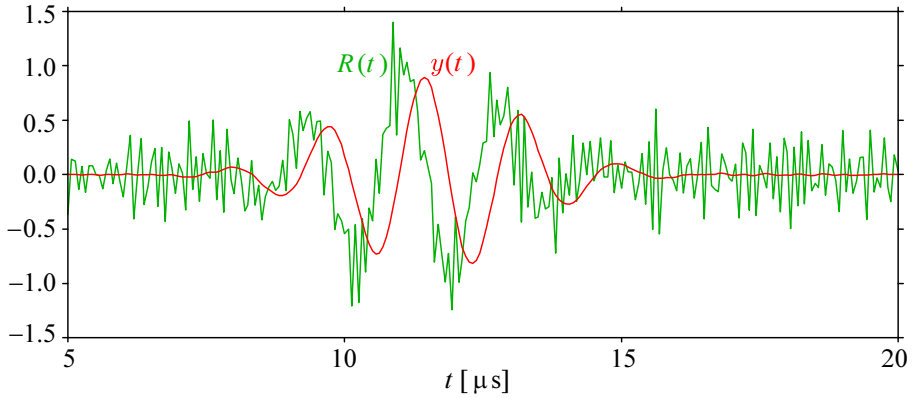

Fig. 7.1.6: The output spectrum is returned to time domain as $y(t)$ and is compared with the input signal $R(t)$, in expanded time scale. Note the small change in amplitude, the reduced noise level and the envelope delay (approx. $1/4$ period time shift), with little change in phase. The time shift is equal to $1/2$ the number of samples of the filter impulse response.

Fig. 7.1.6 illustrates the dramatic improvement in signal quality that can be achieved by using Bessel filters.

In MRI systems the test object is put in a strong static magnetic field. This causes the nucleons of the atoms in the test object to align their magnetic spin to the external field. Then a short RF burst, having a well defined frequency and duration, is applied, tilting the nucleon spin orientation perpendicular to the static field (this happens only to those nucleons whose resonant frequency coincides with that of the RF burst).

After the RF burst has ceased, the nucleons gradually regain their original spin orientation in a top-like precession motion, radiating away the excess electromagnetic energy. This EM radiation is picked up by the sensing coils and detected by an RF receiver; the detected signal has the same frequency as the excitation frequency, both being the function of the static magnetic field and the type of nucleons. Obviously the intensity of the detected radiation is proportional to the number of nucleons having the same resonant frequency[3].

In addition, since the frequency is field dependent a small field gradient can be added to the static magnetic field, in order to split the response into a broad spectrum. The shape of the response spectral envelope then represents the spatial density of the specific nucleons in the test object. By rotating the gradient around the object the recorded spectra would represent the ‘sliced view’ of the object from different angles. A computer can be used to reconstruct the volumetric distribution of particular atoms through a process called ‘back-projection’ (in effect, a type of spatial convolution).

From this short description of the MRI technique it is clear that the most vital parameter of the filter, applied to smooth the recorded signal, is its group delay flatness. Only a filter with a group delay being flat well into the stop band will be able to faithfully deliver the filtered signal, preserving its shape both in the time and the frequency domain, and Bessel–Thomson filters are ideal in this sense. Consequently a sharper image of the test object is obtained.

[3]The 1952 Nobel prize for physics was awarded to *Felix Bloch* and *Edward Mills Purcell* for their work on nuclear magnetic resonance; more info at <http://nobelprize.org/physics/laureates/1952/>.

(blank page)

7.2 System Front–End Design Consideration

7.2.1 General Remarks

The trend in modern instrumentation has been definitely going digital from the 1970s, benefiting from cheap microprocessor technology, being implemented as early as possible in the signal processing chain. Likewise, reverting back to analog occurs only if absolutely necessary and as late as possible. The key features are precision and repeatability of measurements, and those properties are often called upon to justify the sacrificing of other system properties (of those, the system bandwidth is no exception, in fact, it is the first victim in most cases!). In spite of this digital 'tyranny', analog engineers were (for now, at least) able to cope quite successfully with it. Actually, over the years they have managed to stay well in front of both demands and expectations.

Another key word of modern technology is miniaturization, and that in connection with ever-lowering power consumption. So the current technological front is concentrated on the integration of analog and digital functions on the same IC chip, using the lowest possible supply voltage and applying clever power management schemes in both hardware and software.

Of course, digitalization is causing many restrictions as well. In contrast with analog continuous time systems, digital systems operate in discrete time, on the transitions of the system clock. And, since there is only a limited amount of memory, which also has a finite access time, the sampling window is ever shrinking. As if this were not enough (and in spite of promoting precision!), digital systems are not very flexible to upgrade ; for example, to change from an 8 bit to a 12 bit system, an analog circuit would have to be improved by 2^4 or 16 times (if it was not that good already!); in contrast, the digital part would have to be redesigned completely, not just changed. This is because an increased number of gates and flip-flops change state at each clock transition, increasing the supply current peaks; also the circuit area is increased, increasing the length of the interconnections. Both facts increase the RF interference and the possibility of noise injection back into the delicate analog input stage.

In Part 6, Sec. 6.5 we have already learned a few basic facts about the effects of signal sampling. We know that a finite sampling density means only that the signal repeats in time, so the length of the sampling window should be chosen in accordance with the signal length and the sampling frequency. More difficult to handle is the problem of finite word length, since it sets the effective system resolution and the conversion noise level.

7.2.2 Aliasing Phenomena in Sampling Systems

To illustrate further the use of the algorithms developed in Part 6, let us consider the design requirements of a front–end amplifier driving an analog to digital converter (ADC). In theory the minimum required amplifier bandwidth should be equal to the Nyquist frequency, which is one half of the ADC's sampling clock frequency, $f_N = f_c/2$. In practice, however, undistorted reconstruction of a periodic waveform can be achieved only if the signal content above the Nyquist frequency has been attenuated to levels lower than the ADC resolution. This is known in literature as the ***Shannon*'s sampling theorem** [Ref. 7.2].

The purpose of filtering the signal above the Nyquist frequency is to avoid **'aliasing'**. Fig. 7.2.1 shows a typical situation resulting in a signal frequency alias in relation to the sampling clock frequency.

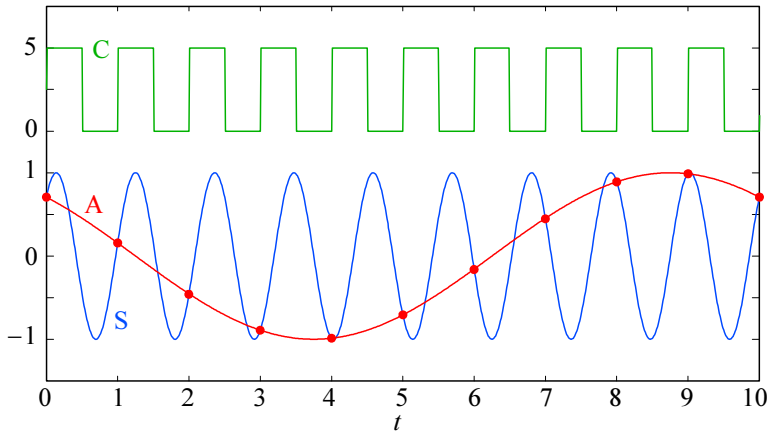

Fig. 7.2.1: Aliasing (frequency mirroring). A high frequency signal S, sampled by an ADC at each rising edge of the clock C of a comparably high frequency, can not be distinguished from its low frequency alias A, which is equal to the difference between the clock and signal frequency, $f_a = f_s - f_c$. In this figure, $f_s = (9/10)\, f_c$, therefore $f_a = -(1/10)\, f_c$ (Yes, a **negative** frequency! This can be verified by increasing the clock frequency very slightly and watch the aliased signal apparently moving backwards in time).

The alias frequency f_a is simply a difference between the signal frequency f_s and the sampling clock's frequency f_c:

$$f_a = f_s - f_c \tag{7.2.1}$$

Aliasing can be best understood if we recall a common scene in Western movies, where the wheels of the stage coach seem to be rotating backwards, while the horses are being whipped to run wild to escape from the desperados behind. The perceived frequency of rotation of the wheel is equal to the **difference** between the actual rotation frequency and the frequency at which the pictures were taken.

A wheel, rotating at the cycle frequency f_w equal to the picture rate f_p (or its integer multiple or sub-multiple, $f_w = n\, f_p/m$, where m is the number of wheel arms), would be perceived as stationary. Likewise, if an ADC's sampling frequency is equal to the signal frequency (see Fig. 7.2.2), the apparent result is a DC level.

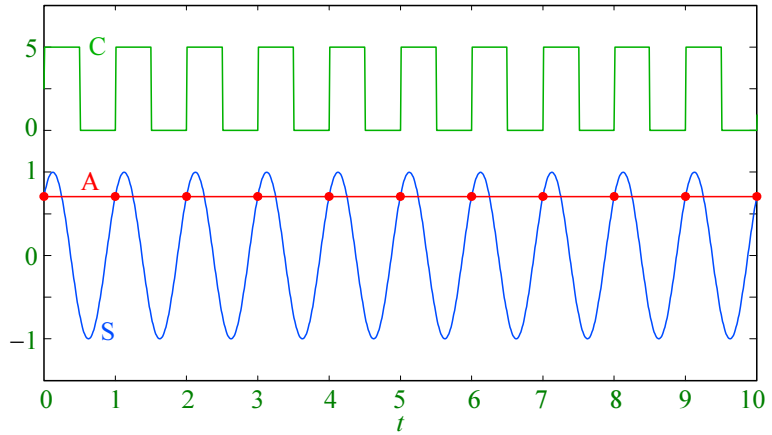

Fig. 7.2.2: Alias of a signal equal in frequency to the sampling clock looks like a DC.

Furthermore, a signal with a frequency slightly higher than the sampling frequency could not be distinguished from a low frequency equal to the difference of the two, as in Fig. 7.2.3. Experienced Hi-Fi enthusiasts and car mechanics will surely remember seeing this, if we remind them of the 'stroboscope effect'.

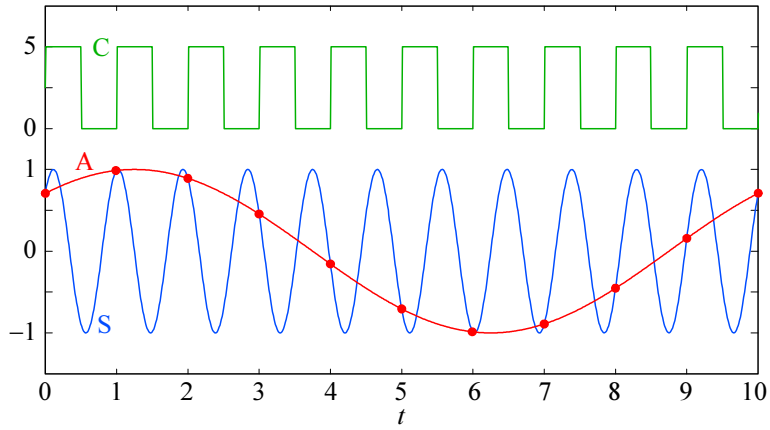

Fig. 7.2.3: A signal frequency slightly higher than the sampling frequency aliases into a low frequency, equal to the difference of the two (but now positive).

Here is the ALIAS routine for Matlab, by which we can calculate the aliasing for any clock and signal frequency desired (assuming infinitely short aperture time).

```
function fa=alias(fs,fc,phi)
% ALIAS calculates the alias frequency of a sampled sinewave signal.
%       Call :   fa = alias( fs, fc, phi ) ;
%       where:   fs is the signal frequency
%                fc is the sampling clock frequency
%               phi is the initial signal phase shift

% Erik Margan, 920807, Free of copyright!

if nargin < 3
     phi = pi/3 ;   % signal phase shift re. clk, arbitrary value
end
ofs = 2 ;           % clock offset
A = 1/ofs ;         % clock amplitude
m = 100 ;           % signal reconstruction factor is equal to
                    % the number of dots within a clock period
N = 1 + 10 * m ;    % total number of dots
dt = 1 / ( m * fc ) ;   % delta-t for time reconstruction
t = dt * ( 0 : 1 : N ) ;    % time vector

fa = fs - fc ;      % alias frequency (can be negative!)
clk = ofs + A * sign( sin( 2 * pi * fc *t ) ) ; % clock
sig = sin( 2 * pi * fs * t + phi ) ;     % sampled signal
sal = sin( 2 * pi * fa * t + phi ) ;     % alias signal

plot( t, clk, '-g',...
      t, sig, '-b',...
      t, sal, '-r',...
      t(1:m:N), sig(1:m:N), 'or')
xlabel( 't' )
```

Of course, the sampled signal is more often than not a spectrum, either discrete or continuous, and aliasing applies to a spectrum in the same way as to discrete frequency signals. In fact, the superposition theorem applies here, too.

We have noted that a sampled spectrum is symmetrical about the sampling frequency, because a signal, sampled by a clock with exactly the same frequency, aliases as a DC level, which in turn depends on the initial signal phase shift relative to the clock. However, something odd happens already at the Nyquist frequency, as can be seen in Fig. 7.2.4 and Fig. 7.2.5. In both figures the signal frequency is equal to the Nyquist frequency ($1/2$ the sampling frequency), but differs in phase. Although the correct alias signal is equal in amplitude to the original signal, we perceive an amplitude which varies with phase.

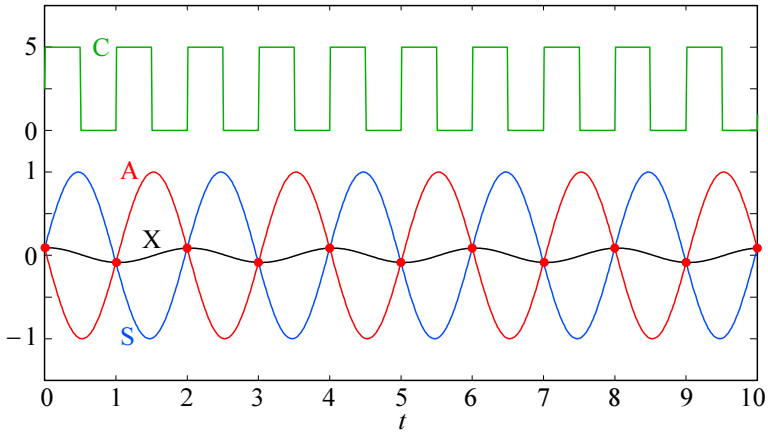

Fig. 7.2.4: When the signal frequency is equal to the Nyquist frequency, there are two samples per period and the correct alias signal is of the same amplitude as the original signal. However, the perceived alias amplitude is a function of the phase difference between the signal and the clock. A $10°$ phase shift results in a low apparent amplitude, as shown by the X waveform.

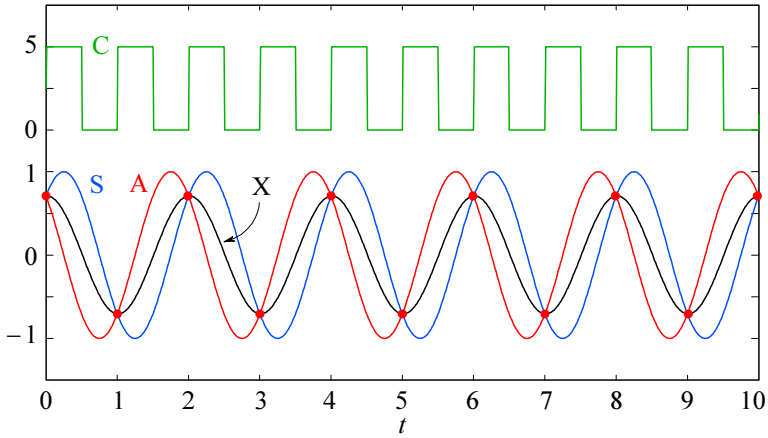

Fig. 7.2.5: Same as in Fig. 7.2.4, but with a 45° phase shift. The apparent amplitude of X is now higher.

In fact, if our ADC were to be sampling a slowly sweeping sinusoidal signal, its spectral envelope would follow the $(\sin \omega T_s)/\omega T_s$ function, shown in Fig. 7.2.6, with the first zero at the Nyquist frequency, the second zero at the sampling frequency and so on, a zero at every harmonic of the Nyquist frequency.

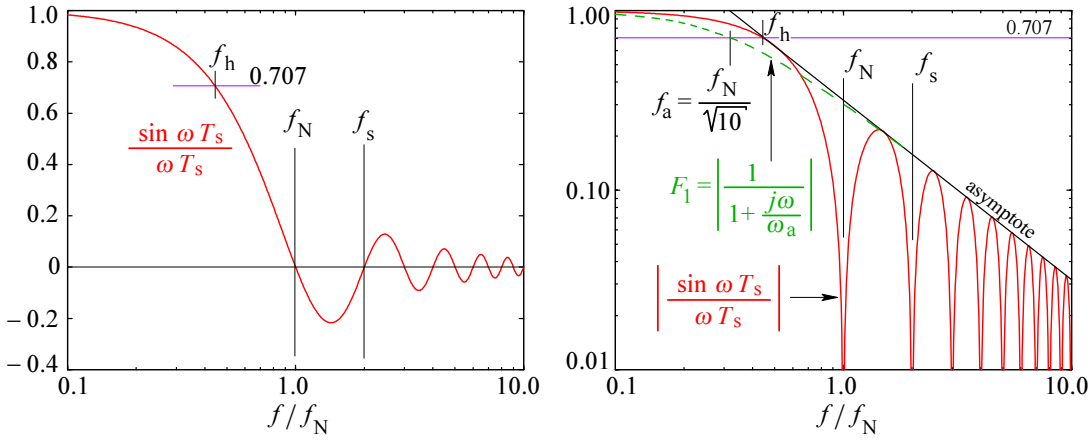

Fig. 7.2.6: The spectrum resulting from sampling a constant amplitude sinusoidal signal varying in frequency from $0.1 f_N$ to $10 f_N$ follows the $(\sin \omega T_s)/\omega T_s$ function, where $T_s = 1/f_s$. The function is shown in the linear vertical scale on the left and in the log of the absolute value on the right. The first zero occurs at the Nyquist frequency, the second at the sampling frequency and so on, at every Nyquist harmonic. Note that the effective sampling bandwidth f_h is reduced to about 0.43 f_N. The asymptote is the same as for a simple RC low pass filter, $-20\,\text{dB}/10f$ with a cut off at $f_a = f_N/\sqrt{10}$.

The aliasing amplitude follows this same $(\sin \omega T_s)/\omega T_s$ function, from the Nyquist frequency up. An important side effect is that the bandwidth is reduced to about 0.43 f_N. This can be taken into account when designing an input anti-aliasing filter and partially compensate the function pattern below the Nyquist frequency by an adequate peaking.

7.2.3 Better Anti-Aliasing With Mixed Mode Filters

By the term ‘mixed mode filter’ we mean a combination of analog and digital filtering which gives the same result as a single filter having the same total number of poles. The simplest way to understand the design requirements and optimization, as well as the advantages of such an approach, is by following an example.

Let us imagine a sampling system using an ADC with a 12 bit amplitude resolution and a 50 ns time resolution (sampling frequency $f_s = 20\,\text{MHz}$). The number of discrete levels resolved by 12 bits is $A = 2^{12} = 4096$; the ADC relative resolution level is simply $1/A$, or in dB, $a = 20 \log_{10}(1/A) = -72\,\text{dB}$. According to the Shannon sampling theorem the frequencies above the Nyquist frequency ($f_N = f_s/2 = 10\,\text{MHz}$) must be attenuated by at least 2^{-12} to reduce the alias of the high frequency spectral content (signal or noise) below the ADC resolution.

As we have just learned, the $(\sin \omega T_s)/\omega T_s$ function of the alias spectrum allows us to relax the filter requirements by some 4 bits (a factor of 4.5 or 13 dB) at the frequency 0.7 f_s; for a while, we are going to neglect this, leaving it for the end of our analysis.

Let us also assume a 4 V peak to peak ADC input signal range and let the maximum required vertical amplifier sensitivity be 5 mV/division. Since oscilloscope displays usually have 8 vertical divisions, this means 40 mV of input for a full scale display, or a gain of 100. We would like to achieve the required gain–bandwidth product with either a two- or a three-stage amplifier. We shall assume a 5-pole filter

for the two-stage amplifier (a 3-pole and a 2-pole stage), and a 7-pole filter for the three-stage amplifier (one 3-pole stage and two 2-pole stages). We shall also inspect the performance of a 9-pole (four-stage) filter to see if the higher bandwidth (achieved as a result of a steeper cut off) justifies the cost and circuit complexity of one additional amplifier stage.

Now, if our input signal was of a square wave or pulse form, our main requirement would be a shortest possible ADC 'aperture' time and an analog bandwidth as high as possible. Then we would be able to recognize the sampled waveform shape even with only 5 samples per period. But suppose we would like to record a transient event having the form of an exponentially decaying oscillating wave, along with lots of broad band noise, something like the signal in Fig. 7.1.4. To do this properly we require both aliasing suppression of the spectrum beyond the Nyquist frequency and preserving the waveform shape; the later requirement limits our choice of filter systems to the Bessel–Thomson family.

Finally, we shall investigate the possibility of improving the system bandwidth by filtering the recorded data digitally.

We start our calculations from the requirement that any anti-alias filter must have the attenuation at the Nyquist frequency f_N equal to the ADC resolution level. Since we know that the asymptote attenuation slope depends on the system order n (number of poles) as $n \times 20\,\mathrm{dB}/10f$, we can follow those asymptotes from f_N back to the maximum signal level; the crossing point then defines the system cut off frequency f_{hn} for each of the three filter systems.

Since we do not have an explicit relation between the Bessel–Thomson filter cut off and its asymptote, we shall use Eq. 6.3.10 for Butterworth systems to find the frequency f_a at which the 5-, 7-, and 9-pole Butterworth filter would exhibit the $A = 2^{12}$ attenuation required. By using $f_h = f_N^2/f_a$ we can then find the Butterworth cut off frequencies. Then by using the modified Bessel–Thomson poles (those that have the same asymptote as the Butterworth system of comparable order) we can find the Bessel–Thomson cut off frequencies which satisfy the no-aliasing requirement.

```
A=2^12;     % ADC resolution limit sets the required attenuation
fs=2e+7;    % ADC sampling frequency, 20MHz
fN=fs/2;    % Nyquist frequency, 10MHz
M=1e+6;     % megahertz scale-factor

% the normalized 5-, 7- and 9-pole system asymptotes, all crossing
% the ADC resolution limit, 1/A, at fN, after Eq.6.3.10, will have
% the following cutoff frequencies :

fh5=fN/10^(log10(A^2-1)/(2*5));
fh7=fN/10^(log10(A^2-1)/(2*7));
fh9=fN/10^(log10(A^2-1)/(2*9));
disp(['fh5 = ', num2str(fa5/M), ' MHz'])
disp(['fh7 = ', num2str(fa7/M), ' MHz'])
disp(['fh9 = ', num2str(fa9/M), ' MHz'])
% the disp commands return the following values :

» fh5 = 1.8946 MHz
» fh7 = 3.0475 MHz
» fh9 = 3.9685 MHz
```

We now find the poles and the system bandwidth of the 5-, 7-, and 9-pole Bessel–Thomson systems, which have their responses normalized to the same asymptotes as the above Butterworth systems of equal order:

```
N=601;                    % number of frequency samples
f=fN*logspace(-2,0,N); % length-N frequency vector, from 2 decades
                          % below fN to fN, in log-scale
w=2*pi*f;                 % angular frequency

[z5,p5]=bestap(5,'a'); % Bessel-Thomson asymptote-normalized systems
[z7,p7]=bestap(7,'a');
[z9,p9]=bestap(9,'a');

p5=p5*2*pi*fa5;           % Scaling-up the poles by the previously
p7=p7*2*pi*fa7;           % calculated cutoff frequencies, so that all
p9=p9*2*pi*fa9;           % three responses have 1/A attenuation at fN

M5=20*log10(abs(freqw(p5,w))); % Calculate magnitudes in dB ;
M7=20*log10(abs(freqw(p7,w)));
M9=20*log10(abs(freqw(p9,w)));
                          % plot magnitudes in dB vs. log frequency
db3=-3.0103;              % the -3dB reference level
                          % and the ADC resolution limit
semilogx( f/M, M5, '-r',...
          f/M, M7, '-g',...
          f/M, M9, '-b',...
          fN*[0.05, 0.35]/M, [db3, db3], '-k'...
          [f(1), f(N)]/M, [1/A, 1/A], '-k' )
xlabel( 'f [MHz]' ) ;
ylabel( 'Attenuation [dB]' )
```

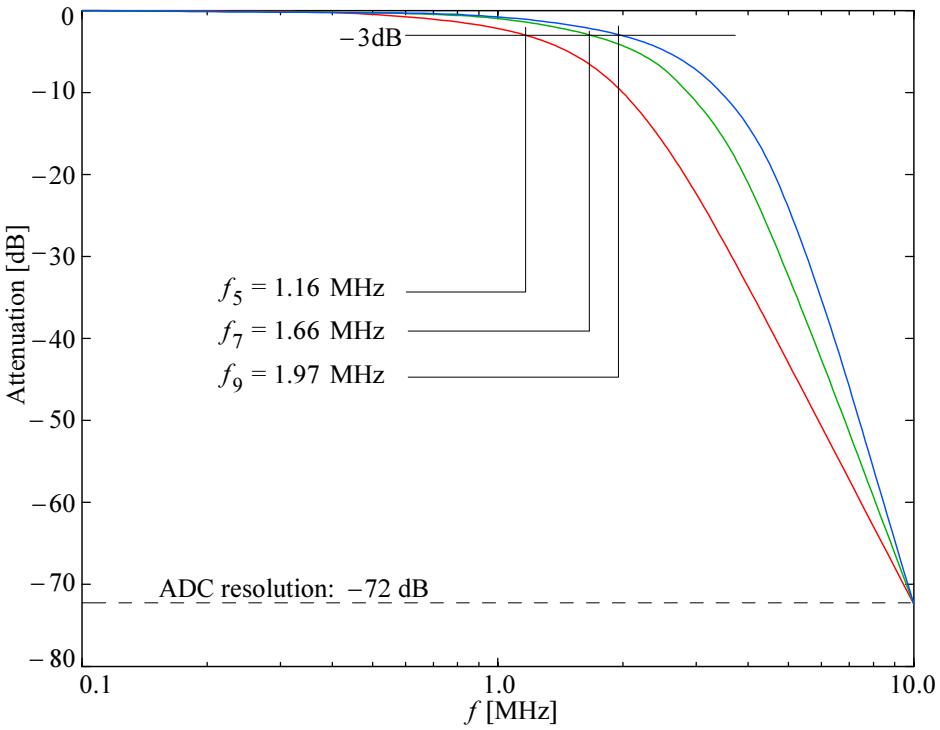

Fig. 7.2.7: Magnitude vs. frequency of Bessel–Thomson 5-, 7-, and 9-pole systems, normalized to the attenuation of 2^{-12} (−72 dB) at f_N (10 MHz).

Fig. 7.2.7 shows the frequency responses, calculated to have the same attenuation, equal to the relative ADC resolution level of −72 dB, at the Nyquist frequency. We now need their approximate −3 dB cut off frequencies:

```
m=abs(M5-3.0103);       % compare the magnitudes with the -3dB level
x5=find(m==min(m));     % and find the index of each frequency limit

m=abs(M7-3.0103);
x7=find(m==min(m));

m=abs(M9-3.0103);
x9=find(m==min(m));

[f5, f7, f9]=f([x5, x7, x9]);       % find the cutoff frequencies

% display cutoff frequencies of the Bessel-Thomson systems :
disp(['f5 = ', num2str(f5/M), ' MHz'])
disp(['f7 = ', num2str(f7/M), ' MHz'])
disp(['f9 = ', num2str(f9/M), ' MHz'])
» f5 = 1.166 MHz
» f7 = 1.660 MHz
» f9 = 1.965 MHz
```

Note that these values are much lower then the cutoff frequencies of the asymptotes, owing to the more gradual roll-off of Bessel–Thomson systems. Also, note that a greater improvement in performance is achieved by increasing the system order from 5 to 7 then from 7 to 9. We would like to have a confirmation of this fact from the step responses (later, we shall also see how these step responses would look when sampled at the actual sampling time intervals).

```
fs=2e+7;          % sampling frequency
t=(0:1:500)/fs;   % time vector (to calculate the rise times we need
                  % a much finer sampling then the actual 50 ns

S5=atdr(z5,p5,t,'s');         % Step responses
S7=atdr(z7,p7,t,'s');
S9=atdr(z9,p9,t,'s');

% plot the step responses
% and the 0.1 and 0.9 reference levels to compare the rise times :
x10=t(50,130), x90=t(150,300), y10=[0.1,0.1], y90=[0.9,0.9];

plot(t,S5,'-r', t,S7,'-g', t,S9,'-b', x10,y10,'-k', x90,y90,'-k' )
xlabel( 't [us]' )

% calculate the rise times :
x5a=find( abs(S5-y10) == min( abs(S5-y10) ) );
x5b=find( abs(S5-y90) == min( abs(S5-y90) ) );
x7a=find( abs(S7-y10) == min( abs(S7-y10) ) );
x7b=find( abs(S7-y90) == min( abs(S7-y90) ) );
x9a=find( abs(S9-y10) == min( abs(S9-y10) ) );
x9b=find( abs(S9-y90) == min( abs(S9-y90) ) );
Tr5=t(x5b)-t(x5a);
Tr7=t(x7b)-t(x7a);
Tr9=t(x9b)-t(x9a);
```

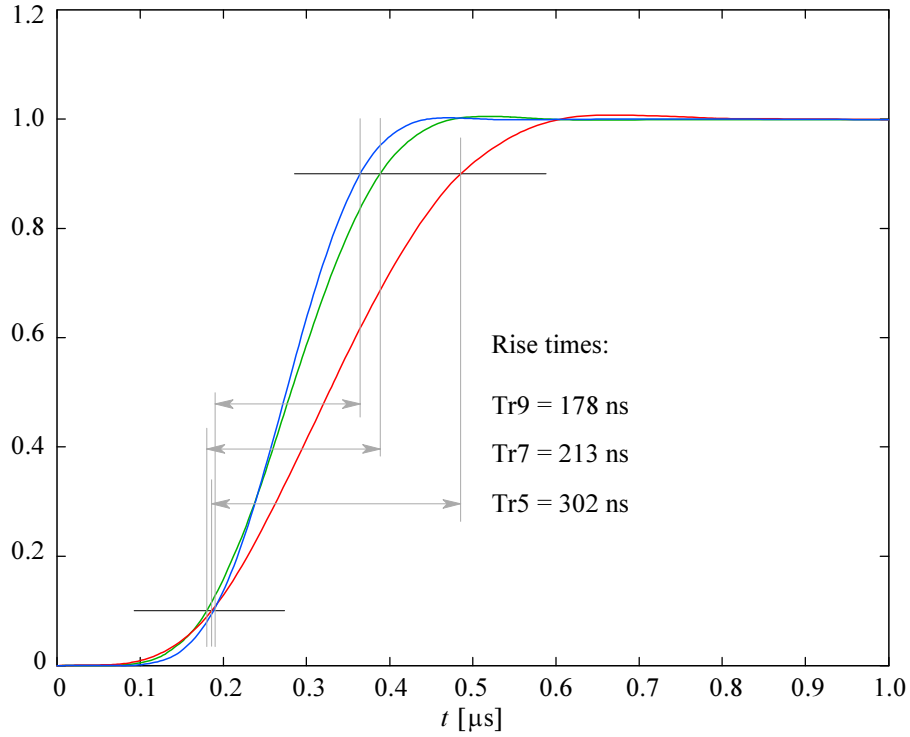

Fig. 7.2.8 : Step responses of the 5-, 7-, and 9-pole Bessel–Thomson systems, having equal attenuation at the Nyquist frequency. The rise times are calculated from the number of samples between the 10% and 90% of the final value of the normalized amplitude.

We see that in this case the improvement from order 7 to order 9 is not so high as to justify the added circuit complexity and cost of one more amplifying stage.

So let us say we are temporarily satisfied with the 7-pole filter system. However, its 1.66 MHz bandwidth for a 12 bit ADC, sampling at 20 MHz, is simply not good enough. Even the 1.96 MHz bandwidth of the 9-pole system is rather low. The question is whether we can find a way around the limitations imposed by the anti-aliasing requirements?

Most ADC recording systems do not have to show the sampled signal in real time. To the human eye a screen refreshing rate of 10 to 20 times per second is fast enough for most purposes. Also many systems are intentionally made to record and accumulate large amounts of data to be reviewed later. So on most occasions there is plenty of time available to implement some sort of signal post–processing.

We are going to show how a digital filter can be combined with the analog anti-aliasing filter to expand the system bandwidth beyond the aliasing limit without increasing the sampling frequency.

Suppose we could implement some form of digital filtering which would suppress the alias spectrum below the ADC resolution and then we ask ourselves what would be the minimum required pass band attenuation of such a filter. The answer is simple: the filter attenuation must follow the inverse of the alias spectrum envelope. But if we were to allow the spectrum around the sampling frequency to alias, our digital filter would need to extend its attenuation characteristic back to DC. Certainly this is neither practical nor desirable. Therefore since $f_s = 2f_N$, our bandwidth improvement factor, let us call it B, must be lower than 2.

So let us increase the filter cut off by $B = 1.73$; the input spectrum would then contain frequencies only up to Bf_N, which would alias back down to $f_s - Bf_N$, in this case $2 - 1.73 = 0.27 f_N$. This frequency is high enough to allow the realization of a not too demanding digital filter.

Let us now study the shape of the alias spectrum which would result from taking our original 7-pole analog filter, denoted by F_{7o}, and pushing it up by the chosen factor B to F_{7b}, as shown in Fig. 7.2.9. The spectrum S_A between f_N and Bf_N is going to be aliased below the Nyquist frequency into S_B.

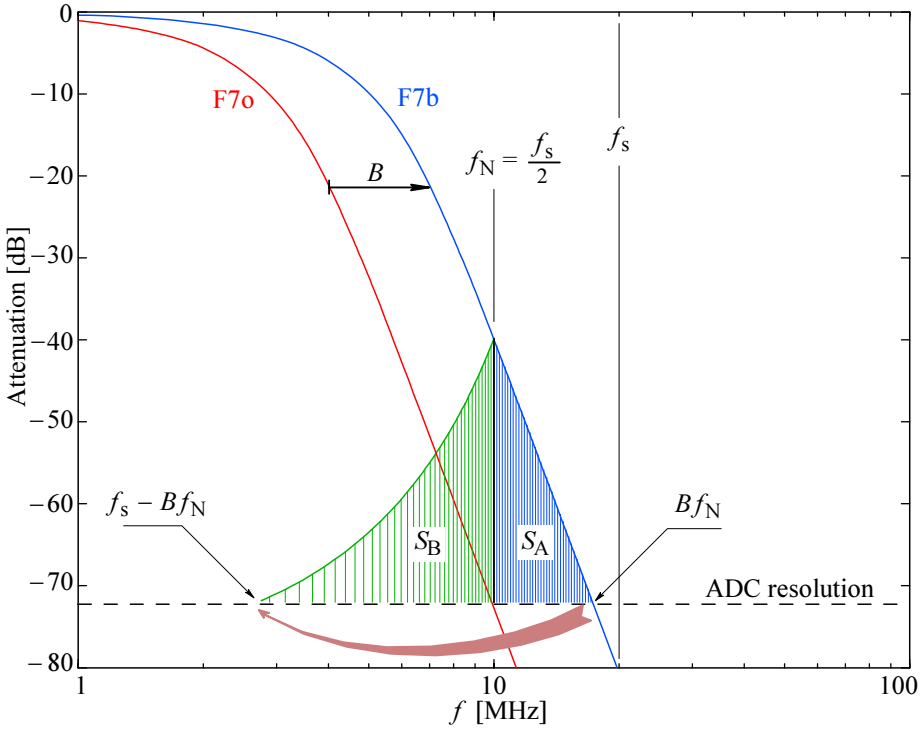

Fig. 7.2.9: Alias spectrum of a 7-pole filter with a higher cut off frequency. F_{7o} is our original 7-pole Bessel-Thomson analog filter, which crosses the 12-bit ADC resolution level of -72 dB at exactly the Nyquist frequency, $f_N = f_S/2 = 10$ MHz. This guaranties freedom from aliasing, but the bandwidth is rather low. If we move it upwards by a factor $B = 1.73$ to F_{7b}, the spectrum S_A will alias into S_B. Note the alias spectrum inversion: f_N remains in its place, whilst Bf_N is aliased to $f_s - Bf_N$. Note also that the alias spectral envelope has changed in comparison with the original: in the log-log scale plot a linearly falling spectral envelope becomes curved when aliased. This change of the spectral envelope is important, since it will allow us to use a relatively simple filter response shape to suppress the aliased spectrum below the ADC resolution.

Note that in the log frequency scale the aliased spectrum envelope is not linear, even if the original one is (as defined by the attenuation characteristic of the analog filter).

If we flip the spectrum S_B up, as in Fig. 7.2.10, the resulting spectral envelope, denoted by F_{rq}, represents the minimal attenuation requirement of a digital filter, which would restore freedom from aliasing.

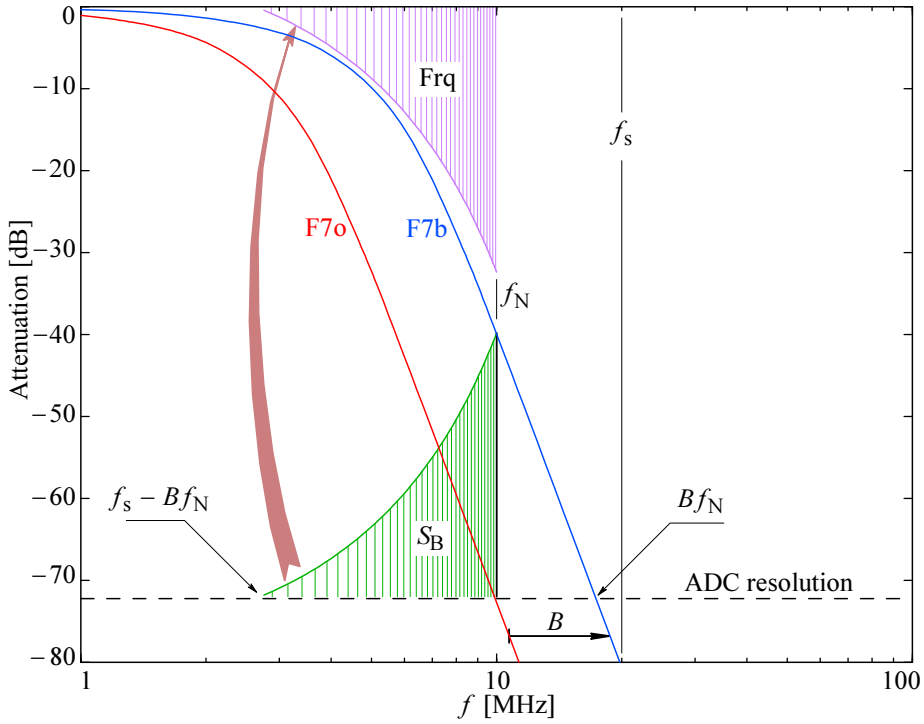

Fig. 7.2.10: If we invert the alias spectrum S_B the envelope of the resulting spectrum F_{rq} represents the minimum attenuation requirement that a digital filter should have in order to suppress the aliased spectrum below the ADC resolution.

The following procedure shows how to calculate and plot the various elements of Fig. 7.2.9 and Fig. 7.2.10, and F_{rq} in particular, starting from the previously calculated 7-pole Bessel–Thomson system magnitude F_{7o}:

```
     % the following calculation assumes a log frequency scale and
     % a linear in dB attenuation.
A=2^12;                 % number of levels resolved by a 12-bit ADC
a=20*log10(1/A);        % ADC resolution, -72dB
Nf=601;                 % number of frequency samples
f=logspace(6,8,Nf);     % frequency vector, 1 to 100 MHz
B=1.73;                 % chosen bandwidth increase (max. 1.8)
     % the original 7-pole filter magnitude crosses a at fN :
F7o=20*log10(abs(freqw(p7, 2*pi*f)));
     % F7o shifted up by B to F7b :
F7b=20*log10(abs(freqw(p7*B, 2*pi*f)));

fA=B*fN;                % F7b crosses ADC resolution (a) at fA
xn=min(find(f>=fN));    % index of fN in f
xa=min(find(f>=fA));    % index of fA in f
Sa=F7a(xn:xa);          % source of the alias spectrum
fa=f(xn:xa);            % frequency band of Sa
Sb=F7a(xa:-1:xn);       % the alias spectrum, from fs-fA to fN
fb=fs-f(xa:-1;xn);      % frequency band of Sb
Frq=a-Sa;               % min. required dig.filt. magnitude in dB
fr=fb;                  % in the same freq. range: fs-fa to fN
M=1e+6;                 % MHz scale factor
semilogx( f/M,F7o,'-r', f/M,'-b', fa/M,Sa,'-y', fb/M,Sb,'-c',...
          fr/M,Frq,'-m', [f(1),f(Nf)]/M,[a,a],'--k',...
          [fN,fN],[-72,-5],':k', [fs,fs],[-80,0],':b' )
xlabel('f [MHz]')
```

As can be seen in Fig. 7.2.10, the required minimum attenuation F_{rq} is broad and smooth, so we can assume that it can be approximated by a digital filter of a relatively low order; e.g., if the analog filter has 7 poles, the digital one could have only 6 poles. The combined system would then be effectively a 13-pole. Of course, the digital filter reduces the combined system's cut off frequency, but it would still be higher than in the original, non-shifted, analog only version. However, the main problem is that the cascade of two separately optimized filters has a non-optimal response and the shape of the transient suffers most. This can be solved by simply starting from a higher order system, say, a 13-pole Bessel–Thomson. Then we assign 7 of the 13 poles to the analog filter and 6 poles to the digital one.

The 6 poles of the digital filter must be transformed into appropriate sampling time delays and amplitude coefficients. This can be done either with 'z-transform' mapping, or simply by calculating its impulse response and use it for convolution with the sampled input signal, as we shall do here.

But note that since now the 7 poles of the analog filter will be taken from a 13-pole system, they will be different from the 7-pole system discussed so far (see a comparison of the poles in Fig. 7.2.11). Although the frequency response will be different, the shape of the alias band will be similar, since the final slope is the same in both cases. Nevertheless, we must repeat the calculations with the new poles.

The question is by which criterion do we choose the 7 poles from the 13. From F_{rq} in Fig. 7.2.9 we can see that the digital filter should not cut sharply, but rather gradually. Such a response could be achieved if we reserve the poles with the lower imaginary part for the digital filter and assign those with high imaginary part to the analog filter. But then the analog filter step response would overshoot and ring, compromising the dynamic range. Thus, the correct design strategy is to assign the real and every other complex conjugate pole pair to the analog filter, as shown below.

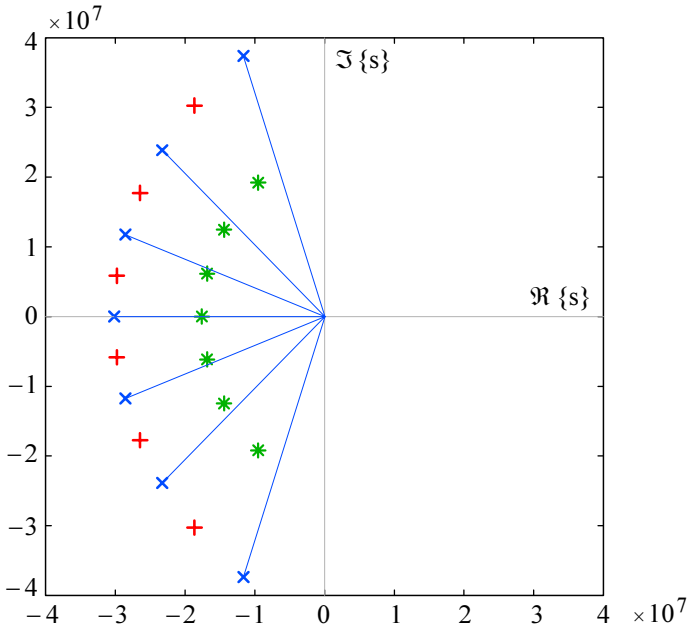

Fig. 7.2.11: The 13-pole mixed mode system, the analog part marked by '×', the digital by '+'; compared with the original 7-pole analog only filter, marked by '*'. Note the difference in pattern size (proportional to bandwidth!).

The values of the mixed mode filter poles for the analog and digital part are:

```
» p7a :  1e+7 *  -3.0181
                 -2.8572 - 1.1751i
                 -2.8572 + 1.1751i
                 -2.3275 - 2.3850i
                 -2.3275 + 2.3850i
                 -1.1637 - 3.7353i
                 -1.1637 + 3.7353i

» p6d :  1e+7 *  -2.9785 - 0.58579i
                 -2.9785 + 0.58579i
                 -2.6460 - 1.77250i
                 -2.6460 + 1.77250i
                 -1.8655 - 3.02640i
                 -1.8655 + 3.02640i
```

Here is a calculation of aliasing suppression using a 13-pole mixed mode filter by using the above poles:

```
fN=1e+7;                % Nyquist frequency
M=1e+6;                 % MHz-us conversion factor
fs=2*fn;                % sampling frequency
A=2^12;                 % ADC dynamic range
a=20*log10(1/A);        % ADC resolution limit in dB
Nf=601;                 % number of frequency samples
f=logspace(6,8,Nf);     % log-scaled frequency vector, 1 - 100 MHz

     % find the frequency normalization factor :
f7=fN/10^(log10(A^2-1)/(2*7));
     % the 7-pole analog-only filter, used as a reference :
[z7,p7]=bestap(7,'a');
p7=2*pi*f7*p7;          % denormalized poles of the reference
F7o=20*log10(abs(freqw(p7,2*pi*f)));  % magnitude of the reference

[z13,p13]=bestap(13,'a');    % the 13th-order Bessel-Thomson system
B=1.73;                 % chosen bandwidth increase
p13=B*2*pi*f7*p13;      % denormalize the 13 poles (f7 same as ref.)
p13=sort(p13);          % sort the poles in ascending abs. value
p7a=p13([1,4,5,8,9,12,13]);  % analog filter pole selection
p6d=p13([2,3,6,7,10,11]);    % digital-equivalent filter poles

F7a=20*log10(abs(freqw(p7a,2*pi*f))); % analog system magnitude
F6d=20*log10(abs(freqw(p6d,2*pi*f))); % digital-equivalent magnitude
F13=20*log10(abs(freqw(p13,2*pi*f))); % total a+d system magnitude

xa=max(find(F7a>=a));  % freq. index of F7a crossing at a
xn=max(find(f<=fN));   % Nyquist frequency index
fa=f(xn:xa);           % frequency band above Nyquist
Fa=F7a(xn:xa);         % response spectrum above Nyquist
fb=fs-f(xa:-1:xn);     % alias band
Fb=F7a(xa:-1:xn);      % alias spectrum
Frq=a-Fb;              % alias suppression requirement
semilogx( f/M,  F7o, '-k', ...
          f/M,  F7a, '-r', ...
          f/M,  F6d, '-b', ...
          f/M,  F13, '-g', ...
          fa/M, Fa,  '-y', ...
          fb/M, Fb,  '-c', ...
          fb/M, Frq, '-m', ...
          [f(1),f(Nf)]/M, [a,a],'--k' )
axis([1, 100, -80, 0]);
```

The result of the above plot operation (`semilogx`) can be seen in Fig. 7.2.12, where we have highlighted the spectrum under the analog filter F_{7a} beyond the Nyquist frequency, its alias, and the inverted alias, which represents the minimum required attenuation F_{rq} of the digital filter. Note how the 6-pole digital filter's response F_{6d} just touches the F_{rq}. Probably it is just a coincidence that the bandwidth increase factor B chosen for the analog filter is equal to $\sqrt{3}$ (we have arrived at this value by repeating the above calculation several times, adjusting B on each run). This process can be easily automated by iteratively comparing the samples of F_{rq} and F_{6d}, and increase or decrease the factor B accordingly.

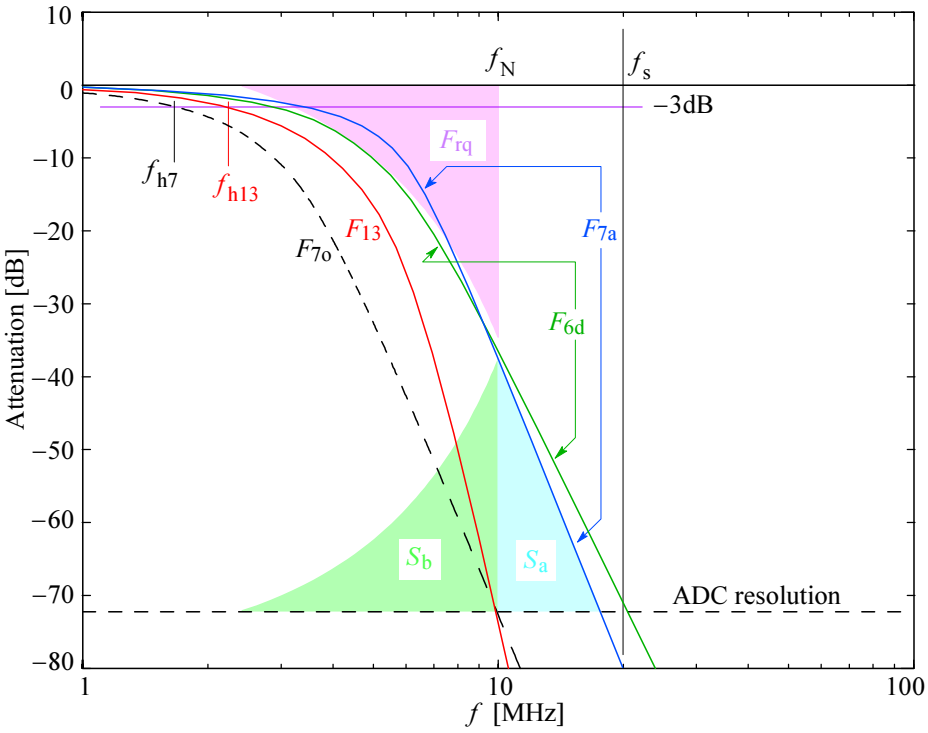

Fig. 7.2.12: The bandwidth improvement achieved with the 13-pole mixed mode filter. The F_{7o} is the original 7-pole analog only filter response, as in Fig. 7.2.9. The new analog filter response F_{7a}, which uses 7 of the 13 poles as shown in Fig. 7.2.11, was first calculated to have the same -72 dB attenuation at the Nyquist frequency f_N (as F_{7o}), and then all the 13 poles were increased by the same factor $B = 1.73$ as before. The resulting F_{7a} response generates the alias spectrum S_b from its source S_a. The envelope of the inverted alias spectrum F_{rq} sets the upper limit for the digital filter's response, F_{6d}, required for effective alias suppression. The response F_{13} is the total analog + digital 13-pole system, which crosses the ADC resolution limit at f_N, and suppresses the alias band below the ADC resolution level, which was the main goal. In this way the system's cut off frequency has increased from 1.66 MHz for F_{7o} to 2.2 MHz for the F_{13}. Thus, a bandwidth improvement of about 1.32 is achieved (not very much, but still significant — note that there are ways of improving this further!).

As expected, the digital filter has reduced the system bandwidth below that of analog filter; however, the analog + digital system's response F_{13} has a cut off frequency f_{h13} well above the f_{h7} of the original analog only 7-pole response, the F_{7o}:

$$f_{7o} = 1.66\,\text{MHz} \qquad f_{13} = 2.2\,\text{MHz}$$

Therefore the total bandwidth improvement factor is $f_{13}/f_{7o} = 1.32$.

The digital filtering process which we are going to use is actually a convolution of discrete signal samples with the filtering coefficients which represent the filter impulse response (sampled) in the time domain. Therefore, if we are going to implement the digital filter in software, we should pay attention to the number of samples of the digital filter impulse response. A high number of samples means a longer time to calculate the convolution. Luckily, the pole selection used will have a nearly Gaussian, non-oscillating, impulse response with only a small undershoot, so a suitable digital filter can be made with only 11 samples (see Fig. 7.2.15).

Let us now calculate and compare the two step responses to examine the effect of bandwidth improvement in time domain:

```
t=2e-9*(0:1:500);              % time vector, 2 ns resolution
M=1e+6;
g7o=atdr(z70,p70,t,'s');       % reference 7-pole system step-response
g13=atdr(z13,p13,t,'s');       % 13-pole step-response

% plot the step responses with the 0.1 and 0.9 reference levels :
plot( t*M, g7o, '-k',...
      t*M, g13, '-r',...
      t([5, 25])*M, [0.1, 0.1], '-k',...
      t([15, 45])*M, [0.9, 0.9], '-k' )
xlabel('t [us]')
```

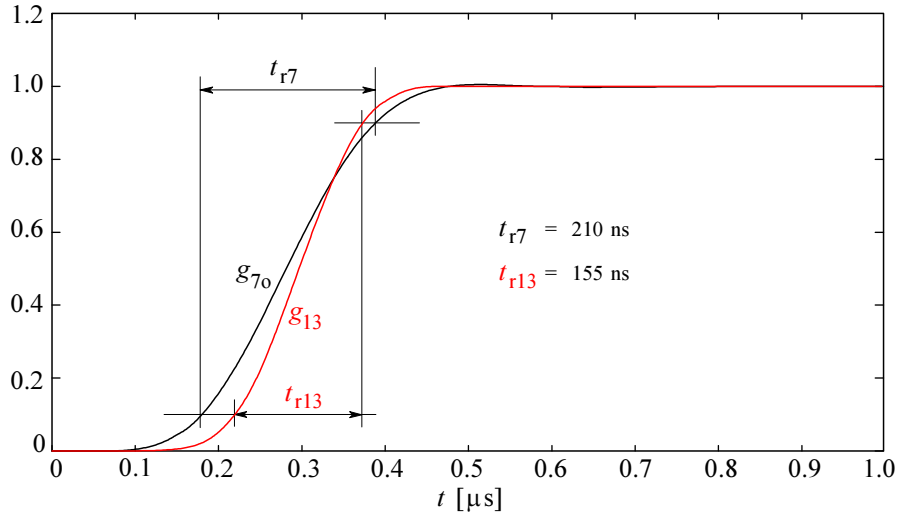

Fig. 7.2.13: Step response comparison of the original 7-pole analog only filter g_{7o} and the improved 13-pole mixed mode (7-pole analog + 6-pole digital) Bessel–Thomson filter, g_{13}. Note the shorter rise time and longer time delay of g_{13}. The resulting rise time is also better than that of the 9-pole analog only filter (see Fig. 7.2.8).

The greatest improvement, however, can be noted in the group delay; as shown in Fig. 7.2.14, the mixed mode system more than doubles the linear phase bandwidth, thus putting a lower constraint on spectrum analysis.

```
                              % continuing from previous calculations:
gd7o=gdly(z7,p7,2*pi*f);      % group-delay for the original 7-pole and
gd13=gdly(z13,p13,2*pi*f); % the mixed-mode 13-pole system
semilogx( f/M, gd7o*M, '-k', f/M, gd13*M, '-r')
```

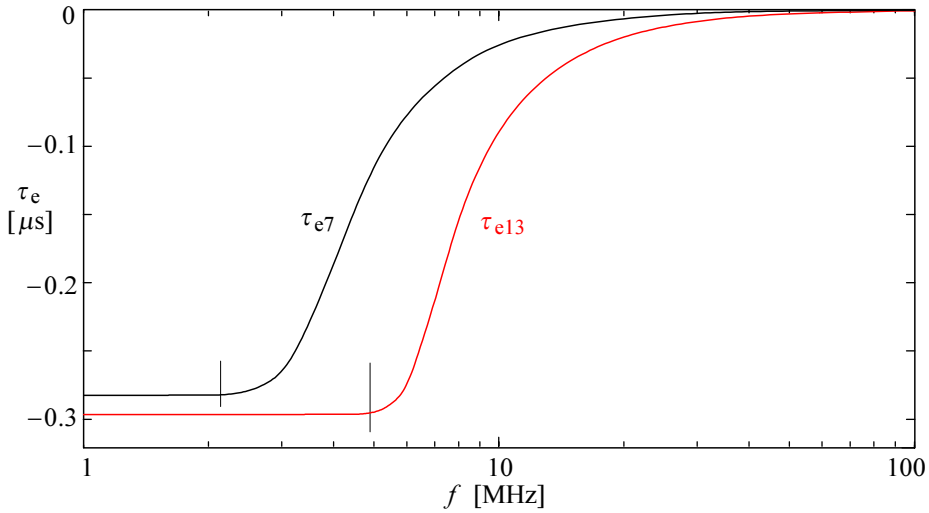

Fig. 7.2.14: Envelope delay comparison of the original 7-pole analog only filter τ_{e7} and the mixed mode 13-pole Bessel–Thomson filter τ_{e13}. The 7-pole analog only filter is linear to a little over 2 MHz, while the 13-pole mixed mode filter is linear well into the stop band, up to almost 5 MHz, more than doubling the useful linear phase bandwidth.

The reader is encouraged to repeat all the above calculations also for the 5- and 9-pole case to examine the dependence of the bandwidth improvement factor on the analog filter's slope.

As we have seen, the bandwidth improvement factor is very sensitive to the steepness of the attenuation slope, so the 9-pole analog filter assisted by an 8-pole equivalent digital filter may be found to be attractive now, extending the bandwidth further.

7.2.4 Gain Optimization

We have said that we need a total gain of 100. Since the analog 7-pole filter can be realized with a three-stage amplifier (one 3-pole stage and two 2-pole stages, see Fig. 7.2.22), the gain of each stage can be optimized by taking the third root of the total gain, $100^{1/3} = 4.6416$ or 13.3 dB (compare this to a two-stage 5-pole filter, where each stage would need $100^{1/2} = 10$ or 20 dB). The lower gain should allow us to use opamps with lower f_T and still have a good phase margin to prevent pole drifting from the optimum (because of the decreasing feedback factor at high frequencies). This is important, since the required 12 bit precision is difficult to achieve without local feedback, and the cost of fast precision amplifiers can be high.

As calculated before, for the sampling frequency of 20 MHz the bandwidths are 1.660 MHz for the 7-pole analog only filter and 2.188 MHz for the 13-pole mixed mode system. In addition to the 13.3 dB of gain, each amplifier stage will need at least 20 dB of feedback at these frequencies, to prevent the response peaking which would result from the finite amplifier bandwidth (if it were too low). By accounting for the amplifier gain roll–off of 20 dB/decade we conclude that we need amplifiers with a unity gain bandwidth of at least 70 MHz for a 9-pole filter and 120 MHz for a 7-pole filter. Note also that amplifiers with a unity gain bandwidth of 160 MHz would be required for the two stages of the 5-pole filter with 20 dB of gain per stage and a system cutoff frequency of only 1.160 MHz!

7.2.5 Digital Filtering Using Convolution

Before we set off to design the analog filter part, let us check the digital filtering process. If we take the output of the analog filter and convolve it with the impulse response of the 6-pole equivalent digital filter, we obtain the response of the composite 13-pole filter. We would also like to see how the step response would look when sampled at the actual ADC sampling frequency of 20 MHz:

```
dt=2e-9;                        % 2 ns time resolution for plotting
Nt=501;                         % length of time vector
t=dt*(0:1:Nt-1);                % time vector
M=1e+6;                         % microsecond normalization factor
fs=2e+7;                        % actual ADC sampling frequency (20MHz)
r=1/(fs*dt);                    % num. of dt's in the sampling period

g7a=atdr(z7,p7a,t,'s');         % step-response of the analog part
h6d=atdr(z7,p6d,t,'n');         % normalized impulse resp. digital part
g13=conv(g7a,h6d);              % digital filtering = convolution
g13=g13(1:max(size(t)));        % take only length(t) samples

plot( t*M, g7a, '-r',...
      t*M(1:r:Nt), g7a(1:r:Nt), 'or',...
      t*M, Nt*h6d, '-g',...
      t*M(1:r:Nt), Nt*h6d(1:r:Nt), 'og',...
      t*M, g13, '-b',...
      t*M(1:r:Nt), g13(1:r:Nt), 'ob' )
xlabel('t [us]')
```

The plot can be seen in Fig. 7.2.15. The dot markers indicate the first 15 time samples of the analog filter step response, the digital filter impulse response and the composite mixed mode filter step response.

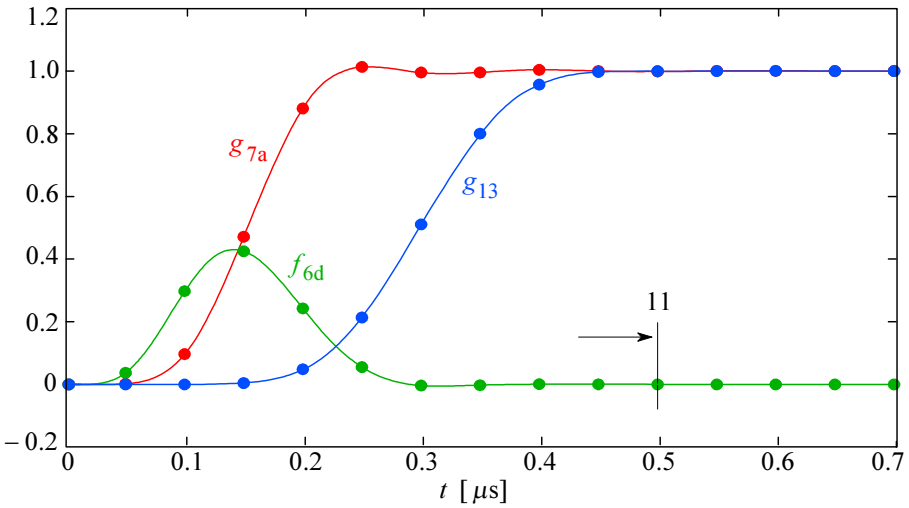

Fig. 7.2.15 : Convolution as digital filtering for a unit step input. The 7-pole analog filter step response g_{a7} is sent to the 6-pole equivalent digital filter, having the impulse response of f_{d6}. Their convolution results in the step response g_{13} of the effectively 13-pole mixed mode composite filter. The marker dots represent the actual time quantization as would result from the ADC sampling at 20 MHz. The impulse response of the digital filter is almost zero after only 11 samples, so the digital filter needs only the first 11 samples as its coefficients for convolution. Note that even if the value of the first coefficient is zero, it must nevertheless be used in order to have the correct response.

Of course, to simulate the actual digitalization, we should have multiplied the normalized signal by 2^{12} and then rounded it to integers to obtain a vertical range of 4096 discrete levels. However, the best computer monitors have a vertical resolution of only $1/4$ of that. If viewed on screen the signal would be seen as if it were digitized with a 10 bit resolution at best (but with lower quantization noise). On paper, with a printer resolution of 600 dots per inch, the vertical size of this figure would have to be 6.5 inches (16.5 cm) in order to appreciate the full ADC resolution.

Nevertheless, measurement precision is always of prime importance, particularly if additional operations are required to extract the information from the recorded signal. In such a case the digital filtering can help, since it will additionally suppress the quantization noise, as well as the amplifier's noise, by a factor equal to the square root of the number of filter coefficients. Disregarding the analog input noise for the moment, if the quantization noise of the 12 bit ADC were ± 2 LSBs at the maximum sampling frequency, its precision would be effectively 10 bits. So, if the impulse response of our digital filter was 11 samples long the quantization noise would be averaged by an additional $\sqrt{11}$, or about 3.3 times, resulting in an almost 2 bit improvement in precision. Effectively, the internal signal precision in memory would be restored to nearly 12 bits.

A final note on the step response: we have started the design of the mixed mode filter because we have assumed that aliasing would be a problem. However, aliasing is a problem only for periodic signals, not for the step! Nevertheless, we are interested in the step response for two reasons:

- one is that even if we were not to care for it, our customers and the people using our instrument would want to know the rise time, the settling time, etc., and, also very important, our competitors would certainly not spare their means of checking it if they can show they are better, or just very close but cheaper!
- the other reason is that the step response will give us a direct evidence of the system phase linearity, a parameter of fundamental importance in spectrum analysis.

7.2.6 Analog Filters With Zeros

By using a 13-pole mixed mode filtering we have so far achieved a bandwidth improvement of about 1.37 compared to a 7-pole analog only filter. But even greater improvements are possible if we use analog filters with zeros. Combining a Bessel low pass response with zeros in the stop band is difficult but not impossible to achieve. The zeros must be in pairs and purely imaginary, and they must be placed at the Nyquist frequency and its first harmonic, the sampling frequency (beyond f_s the filter attenuation is high, so any aliasing from there will already be below the ADC resolution). In this way the zeros effectively prevent aliasing down to DC, and they also modify the filter slope to become very steep near them, allowing us to increase the bandwidth further still: a factor of up to 1.87 can be achieved.

One such example, calculated from the same initial assumptions as before, is shown in the following figures. Fig. 7.2.x5 shows the poles and zeros, Fig. 7.2.x6 shows the frequency responses, and Fig. 7.2.x7 shows the alias spectrum in relation to the filter responses and the resulting alias suppression by the digital filter.

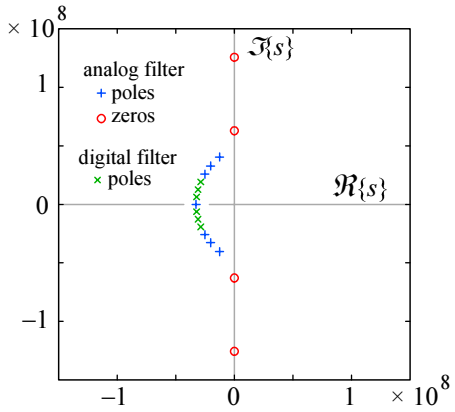

Fig. 7.2.16: An example of a similar mixed mode filter, but with the analog filter using two pairs of imaginary zeros, one pair at the sampling frequency and the other pair at the Nyquist frequency.

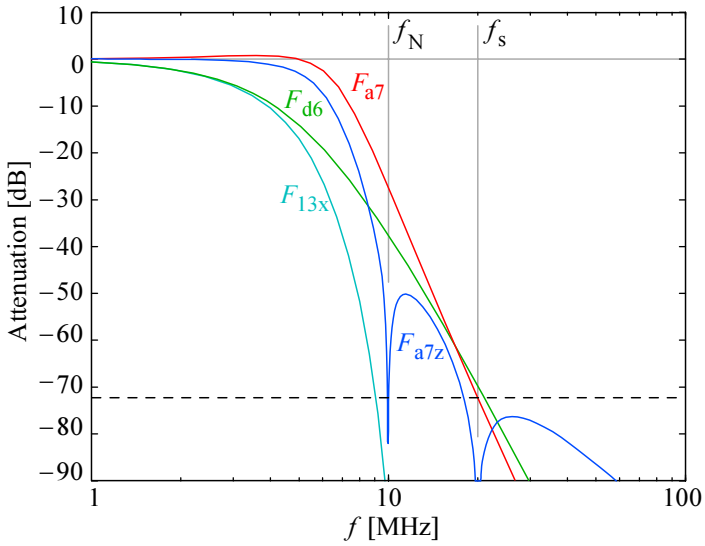

Fig. 7.2.17: By adding the zeros the analog filter frequency response in modified from F_{a7} to F_{a7z}. F_{d6} is the digital filter response and F_{13x} is the composite mixed mode response.

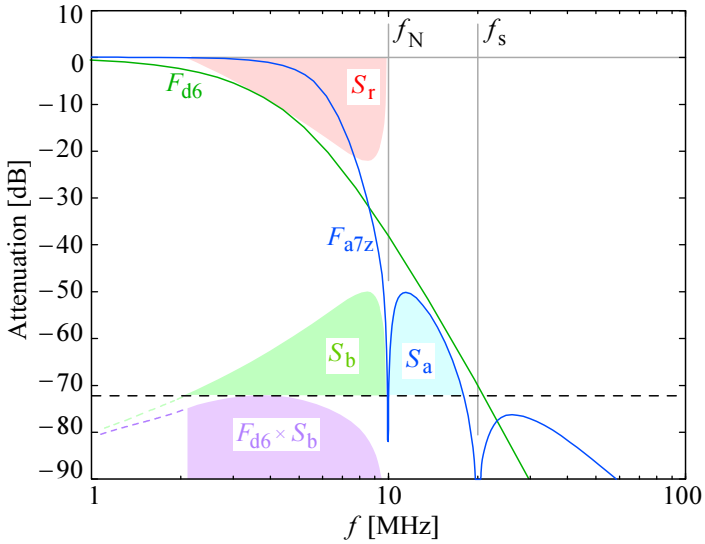

Fig. 7.2.18: The spectrum S_a is aliased into S_b. The difference between the ADC resolution and S_b (both in dB) gives S_r, the envelope of which determines the minimum attenuation required by F_{d6} to suppress S_b below the ADC resolution.

Fig. 7.2.19 shows the time domain responses. Note the influence of zeros on the analog filter response.

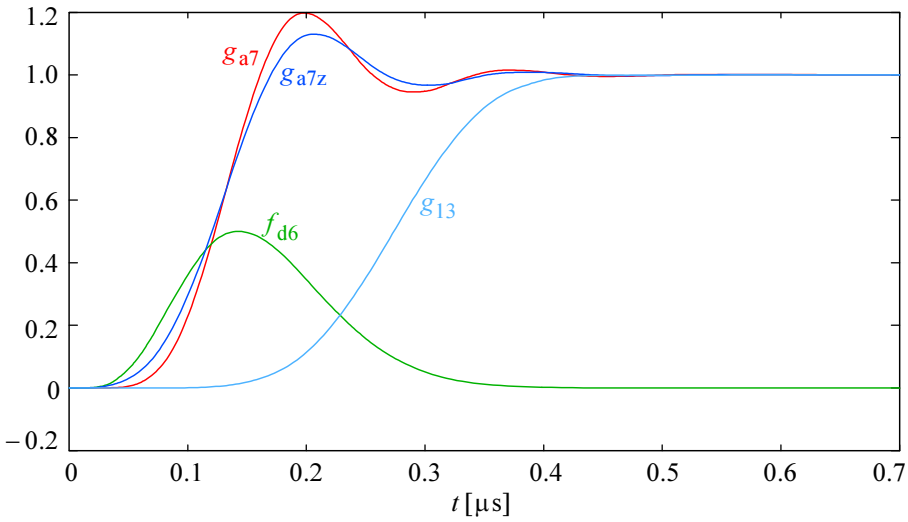

Fig. 7.2.19: The step response of the 7-pole analog filter g_{a7} is modified by the 4 zeros into g_{a7z}. Because the alias spectrum is narrower the bandwidth can be increased. The mixed mode system step response g_{13} has a rise time of less than 150 ns (in contrast with the 180 ns for the case with no zeros).

7.2.7 Analog Filter Configuration

So far we have inspected the system performance in detail, without knowing the circuit's actual configuration. Now it is time to select the analog filter configuration and calculate the component values needed to comply with the required performance. To achieve this we have to consider both the system's accuracy (which has to be equal or better than the 12 bit ADC resolution) and the gain–bandwidth product of each stage, which we have already accounted for in part.

In general, any filter topology could be used if only the transfer function is considered. However for high frequency applications the filter topology which suits us best is the 'Multiple–Feedback' type (MFB), built around operational amplifiers. MFB uses a phase inverting, virtual ground amplifier with the filtering components in its feedback loop, as shown in Fig. 7.2.16 and 7.2.18. Without feedback, the 12 bit precision, offered by the ADC, would be impossible to obtain and preserve in the analog circuit. As a bonus, this topology also offers lowest distortion.

In fact, the non-inverting configurations, such as the '*Sallen–Key*' (SK) filter type or the '*Frequency Dependent Negative Resistance*' (FDNR) type, must support a common mode signal as high as the input signal, whilst the filtering is done on the small differential signal. The mismatch between the inverting and non-inverting opamp input impedance is inevitable in filters, so the finite and frequency dependent common mode rejection plays an important role regarding signal fidelity. As a 'rule of thumb' for high speed opamps: **invert if you can; if you can not, invert twice!**

Another factor, which becomes important in high frequency active filters, is the impedance of the filtering components; the MFB configuration allows us to use low resistance and moderate capacitance values, thus minimizing the influence of

strays. In the figures below are the schematic diagrams of a 3-pole and a 2-pole stage. We shall use the 3+2+2 cascade to implement the 7-pole filter required.

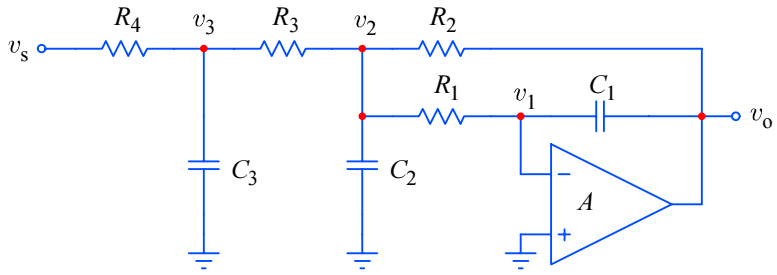

Fig. 7.2.20. The 'Multiple–Feedback' 3-pole (MFB-3) low pass filter configuration

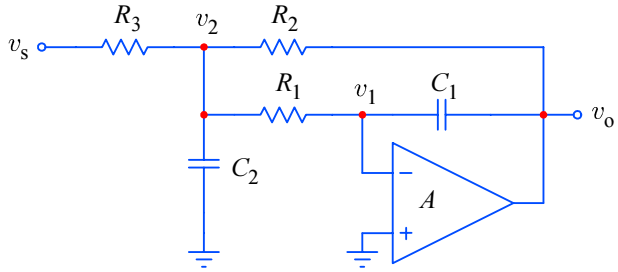

Fig. 7.2.21: The 'Multiple–Feedback' 2-pole (MFB-2) low pass filter configurations

7.2.8 Transfer Function Analysis of the MFB-3 Filter

Our task is to find the transfer functions of these two circuits and then relate the time constants to the poles so that the required component values can be found.

The 3rd-order stage will be the first in the cascade, so let us calculate its components first. For the complete analysis please refer to Appendix 7.1; here we write only the resulting transfer function, which in the general case is:

$$F(s) = A_0 \frac{-s_1 s_2 s_3}{(s-s_1)(s-s_2)(s-s_3)}$$

$$= A_0 \frac{-s_1 s_2 s_3}{s^3 - s^2(s_1+s_2+s_3) + s\,(s_1 s_2 + s_1 s_3 + s_2 s_3) - s_1 s_2 s_3} \tag{7.2.2}$$

For the MFB-3 circuit the coefficients at each power of s are:

$$-(s_1+s_2+s_3) = \frac{1}{C_3 R_4}\left(1+\frac{R_4}{R_3}\right) + \frac{1}{C_2 R_3}\left(1+\frac{R_3}{R_2}+\frac{R_3}{R_1}\right) \tag{7.2.3}$$

$$s_1 s_2 + s_1 s_3 + s_2 s_3 = \frac{1+(R_3+R_4)\left(\frac{1}{R_2}+\frac{1}{R_1}\right)}{C_2 C_3 R_3 R_4} + \frac{\frac{R_3}{R_2}}{C_1 C_2 R_1 R_3} \tag{7.2.4}$$

$$-s_1 s_2 s_3 = \frac{R_3}{R_2}\left(1+\frac{R_4}{R_3}\right)\frac{1}{C_1 C_2 C_3 R_1 R_3 R_4} \tag{7.2.5}$$

and the DC gain A_0 is:

$$A_0 = -\frac{R_2}{R_3 + R_4} \tag{7.2.6}$$

By examining the coefficients and the gain we note that we can optimize the component values by making the resistors R_1, R_3, and R_4 equal:

$$R_1 = R_3 = R_4 = R \tag{7.2.7}$$

The coefficients and the gain equations now take the following form:

$$-(s_1 + s_2 + s_3) = \frac{2}{C_3 R} + \frac{1}{C_2 R}\left(2 + \frac{R}{R_2}\right) \tag{7.2.8}$$

$$s_1 s_2 + s_1 s_3 + s_2 s_3 = \frac{1}{C_2 C_3 R^2}\left(3 + \frac{2R}{R_2}\right) + \frac{1}{C_1 C_2 R^2} \cdot \frac{R}{R_2} \tag{7.2.9}$$

$$-s_1 s_2 s_3 = \frac{2R}{R_2} \cdot \frac{1}{C_1 C_2 C_3 R^3} \tag{7.2.10}$$

$$A_0 = -\frac{R_2}{2R} \tag{7.2.11}$$

To simplify our expressions we shall substitute the term R/R_2 by:

$$M = \frac{R}{R_2} = -\frac{1}{2A_0} \tag{7.2.12}$$

so the coefficients can be written as:

$$K_2 = -(s_1 + s_2 + s_3) = \frac{2}{C_3 R} + \frac{2 + M}{C_2 R} \tag{7.2.13}$$

$$K_1 = s_1 s_2 + s_1 s_3 + s_2 s_3 = \frac{3 + 2M}{C_2 C_3 R^2} + \frac{M}{C_1 C_2 R^2} \tag{7.2.14}$$

$$K_0 = -s_1 s_2 s_3 = \frac{2M}{C_1 C_2 C_3 R^3} \tag{7.2.15}$$

Next we can normalize the resistance ratios and the RC time constants in order to eliminate the unnecessary variables. After the equations are solved and the component ratios are found we shall denormalize the component values to the actual cut off frequency, as required by the poles. We can thus set the normalization factor:

$$N = \frac{1}{R} \tag{7.2.16}$$

so instead of R we have the normalized resistance R_N:

$$R_N = NR = 1 \tag{7.2.17}$$

Accordingly, the capacitance values are now also normalized, and to distinguish the new values from the original ones we shall label the capacitors as:

$$C_\mathrm{a} = \frac{C_1}{N} \qquad C_\mathrm{b} = \frac{C_2}{N} \qquad C_\mathrm{c} = \frac{C_3}{N} \tag{7.2.18}$$

With these changes we obtain:

$$K_2 = \frac{2}{C_\mathrm{c}} + \frac{2+M}{C_\mathrm{b}} \tag{7.2.19}$$

$$K_1 = \frac{3+2M}{C_\mathrm{b}C_\mathrm{c}} + \frac{M}{C_\mathrm{a}C_\mathrm{b}} \tag{7.2.20}$$

$$K_0 = \frac{2M}{C_\mathrm{a}C_\mathrm{b}C_\mathrm{c}} \tag{7.2.21}$$

We can now express, say, C_a from Eq. 7.2.21:

$$C_\mathrm{a} = \frac{2M}{K_0C_\mathrm{b}C_\mathrm{c}} \tag{7.2.22}$$

which we insert into Eq. 7.2.20:

$$K_1 = \frac{3+2M}{C_\mathrm{b}C_\mathrm{c}} + \frac{K_0C_\mathrm{c}}{2} \tag{7.2.23}$$

From this we express C_b:

$$C_\mathrm{b} = \frac{2(3+2M)}{2K_1C_\mathrm{c} - K_0C_\mathrm{c}^2} \tag{7.2.24}$$

Now we can eliminate C_b from Eq. 7.2.19:

$$K_2 = \frac{2}{C_\mathrm{c}} + \frac{2+M}{\dfrac{2(3+2M)}{2K_1C_\mathrm{c} - K_0C_\mathrm{c}^2}} \tag{7.2.25}$$

and we remain with only C_c, for which we have a third-order equation:

$$C_\mathrm{c}^3 - 2\frac{K_1}{K_0}\,C_\mathrm{c}^2 + \frac{2(3+2M)K_2}{(2+M)K_0}\,C_\mathrm{c} - \frac{4(3+2M)}{(2+M)K_0} = 0 \tag{7.2.26}$$

By substituting the coefficients of this equation with p, q, and r:

$$p = -2\frac{K_1}{K_0} \tag{7.2.27}$$

$$q = \frac{2(3+2M)K_2}{(2+M)K_0} \tag{7.2.28}$$

$$r = -\frac{4(3+2M)}{(2+M)K_0} \tag{7.2.29}$$

we can rewrite Eq. 7.2.23 as:

$$C_c^3 + p\,C_c^2 + q\,C_c + r = 0 \tag{7.2.30}$$

The real general solution for this third-order equation (see Appendix 2.1) is:

$$C_c = -\frac{2}{3}\sqrt{p^2 - 3q}\,\sin\left\{\frac{1}{3}\arctan\left[\frac{-\sqrt{-3}\,(2\,p^3 - 9\,p\,q + 27\,r)}{9\sqrt{4\,r\,p^3 - p^2q^2 - 18\,p\,q\,r + 4\,q^3 + 27\,r^2}}\right]\right\} - \frac{p}{3} \tag{7.2.31}$$

By inserting the poles s_1, s_2, and s_3 into the expressions for K_0, K_1, K_3, and the expression for gain in M, and then using it all in the expressions for p, q, and r, we finally obtain the value of C_c. The explicit expression would be too long to write here, and, anyway, it is only a matter of simple substitution, which in a numerical algorithm would not be necessary. With the value of C_c known we can go back to Eq. 7.2.24 to find the value of C_b:

$$C_b = \frac{2\left(3 - \dfrac{1}{A_0}\right)}{\left[2(s_1s_2 + s_1s_3 + s_2s_3)C_c + s_1s_2s_3C_c^2\right]} \tag{7.2.32}$$

and from Eq. 7.2.20 we express C_a:

$$C_a = -\frac{1}{A_0}\cdot\frac{1}{-s_1s_2s_3\,C_bC_c} \tag{7.2.33}$$

Now that the normalized capacitor values are known, the denormalization process makes use of the circuit's cut off frequency, ω_{3h} (which, to remind you, is different from the cut off frequency ω_{7h} of the 7-pole filter, and also different from the total system cut off, ω_{13h}); ω_{3h} relates to K_0 and the poles as:

$$K_0 = \omega_{3h}^3 = -s_1s_2s_3 \tag{7.2.34}$$

From ω_{3h} we can denormalize the values of R and the capacitors to acquire some suitable values, such that the opamp of our choice can easily drive its own feedback impedance and the input impedance of the following stage. For high system bandwidth, it is advisable to choose R as low as possible, usually in the range between 150 and 750 Ω. Let us say that we can set $R = 220\,\Omega$. We use the inverse:

$$N = \frac{1}{R} = \frac{1}{220} \tag{7.2.35}$$

to denormalize the capacitors accordingly:

$$C_1 = NC_a \tag{7.2.36}$$

$$C_2 = NC_b \tag{7.2.37}$$

$$C_3 = NC_c \tag{7.2.38}$$

The cut off frequency is:

$$f_{3h} = \frac{\omega_{3h}}{2\pi} = \frac{1}{2\pi R}\sqrt[3]{\frac{1}{-A_0}\cdot\frac{1}{C_1C_2C_3}} \tag{7.2.39}$$

From the system gain we obtain the value of R_2:

$$R_2 = -2\,R\,A_0 \tag{7.2.40}$$

By inserting the first three poles from `p7a` for s_1, s_2, and s_3, we obtain the following component values:

```
% The poles of the 7th-order analog filter:

p7a:     1e+7 *  -3.0181
                 -2.8572 - 1.1751i
                 -2.8572 + 1.1751i
                 -2.3275 - 2.3850i
                 -2.3275 + 2.3850i
                 -1.1637 - 3.7353i
                 -1.1637 + 3.7353i      [rad/s]

% The first three poles of p7a are assigned to the MFB-3 circuit:

s1 = -3.0181 * 1e+7                     [rad/s]
s2 = ( -2.8572 - 1.1751i ) * 1e+7       [rad/s]
s3 = ( -2.8572 + 1.1751i ) * 1e+7       [rad/s]

% The single stage gain is:

Ao = 100^(1/3) = 4.642

% ------ MFB-3 components: ------------------------------

                R=R4=R3=R1 = 220 Ohm      R2 = 2042 Ohm

  C3 = 102.8 pF         C2 = 274.5 pF     C1 = 24.9 pF

% The cut off frequency is:

  f3h = 4.879 MHz
```

7.2.9 Transfer Function Analysis of the MFB-2 Filter

The derivation for the two second-order stages, which are both of the form shown in Fig. 7.2.21, is also reported in detail in Appendix 7.2. Again, here we give only the resulting transfer function:

$$\frac{v_o}{v_s} = -\frac{R_2}{R_3}\cdot\frac{\dfrac{R_3}{R_2}\cdot\dfrac{1}{R_1R_3C_1C_2}}{s^2 + s\left(\dfrac{R_3}{R_1}+\dfrac{R_3}{R_2}+1\right)\dfrac{1}{R_3C_2} + \dfrac{R_3}{R_2}\cdot\dfrac{1}{R_1R_3C_1C_2}} \tag{7.2.41}$$

By comparing this with the general form:

$$H(s) = A_0 \frac{s_1 s_2}{(s - s_1)(s - s_2)} = A_0 \frac{s_1 s_2}{s^2 - s\,(s_1 + s_2) + s_1 s_2} \tag{7.2.42}$$

we find the system gain:

$$A_0 = -\frac{R_2}{R_3} \tag{7.2.43}$$

and the component values are found from the denominator polynomial coefficients, in which, in order to optimize the component values, we again make $R_1 = R_3 = R$:

$$s_1 s_2 = \frac{1}{-A_0\, R^2\, C_1 C_2} \tag{7.2.44}$$

$$-(s_1 + s_2) = \frac{1}{R\, C_2} \left(2 + \frac{1}{-A_0}\right) \tag{7.2.45}$$

Solving for C_2 and C_1, respectively, we have:

$$C_2 = \frac{1}{-R\,(s_1 + s_2)} \left(2 + \frac{1}{A_0}\right) \tag{7.2.46}$$

$$C_1 = \frac{-(s_1 + s_2)}{s_1 s_2} \frac{1}{R(2A_0 + 1)} \tag{7.2.47}$$

Again we introduce the normalization factor $N = 1/R$, so that $R_N = NR = 1$ and we accordingly normalize the capacitor values:

$$C_\mathrm{a} = \frac{C_1}{N} \qquad\qquad C_\mathrm{b} = \frac{C_2}{N} \tag{7.2.48}$$

Then:

$$C_\mathrm{b} = \frac{1}{-(s_1 + s_2)} \left(2 + \frac{1}{A_0}\right) \tag{7.2.49}$$

$$C_\mathrm{a} = \frac{-(s_1 + s_2)}{s_1 s_2} \frac{1}{2A_0 + 1} \tag{7.2.50}$$

Let us say that here, too, we use the same values for R and N, as before ($R = 220\,\Omega$, $N = 1/200$; note however that in general we can use a different value if for whatever reason we find it more suitable — one such reason can be the preferred values of capacitors). Thus:

$$C_1 = NC_\mathrm{a} \qquad\qquad C_2 = NC_\mathrm{b} \tag{7.2.51}$$

The cut off frequency of the MFB-2 circuit is simply:

$$f_\mathrm{2h} = \frac{\omega_\mathrm{2h}}{2\pi} = \frac{\sqrt{s_1 s_2}}{2\pi} = \frac{1}{R\sqrt{-A_0\, C_1 C_2}} \tag{7.2.52}$$

From `p7a` we use the 4th and the 5th pole for the first MFB-2 stage and the 6th and 7th pole for the second MFB-2 stage. With those we obtain the following component values:

```
% The first MFB-2 circuit:
 s1 = ( -2.3275 - 2.3850i ) * 1e+7     [rad/s]
 s2 = ( -2.3275 + 2.3850i ) * 1e+7     [rad/s]
f2h = 5.304 MHz

------ component values: -----------------------------
 Ao = 4.642      R=R3=R1 = 220 ohm      R2 = 1021 ohm
                      C2 = 216.3 pF     C1 = 18.5 pF

% The second MFB-2 circuit:
 s1 = ( -1.1637 - 3.7353i ) * 1e+7     [rad/s]
 s2 = ( -1.1637 + 3.7353i ) * 1e+7     [rad/s]
f2h = 6.227 MHz

------ component values: -----------------------------
 Ao = 4.642      R=R3=R1 = 220 ohm      R2 = 1021 ohm
                      C2 = 432.7 pF     C1 = 6.7 pF
```

We can now finally draw the complete circuit schematic diagram with component values:

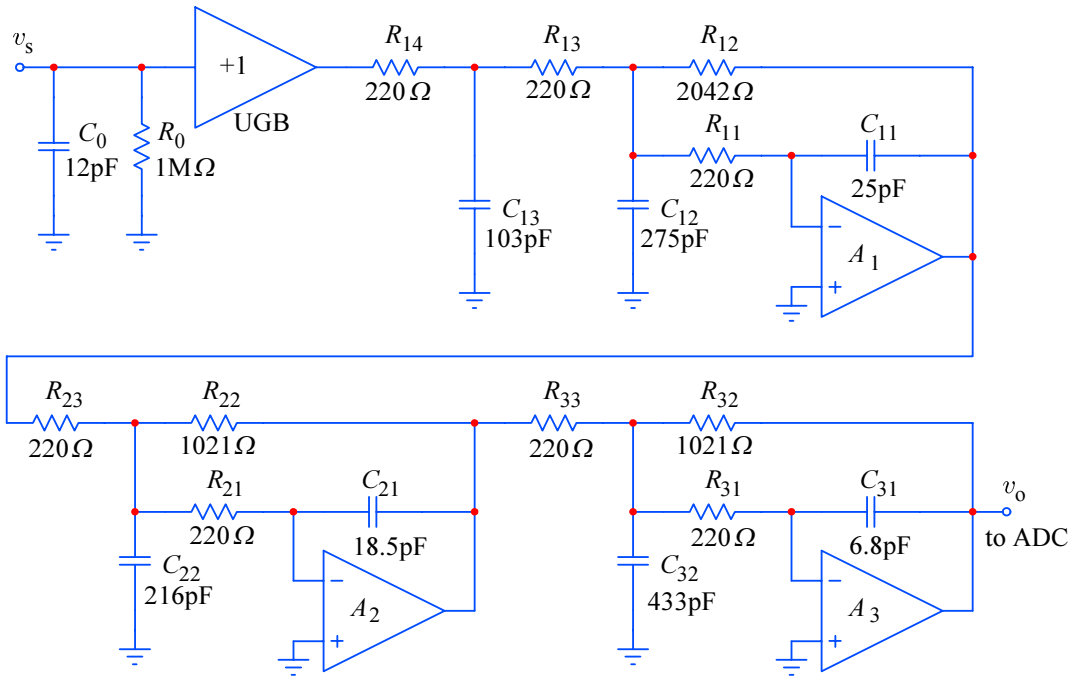

Fig. 7.2.22: The realization of the 7-pole analog filter for the ADC discussed in the aliasing suppression example. The input signal is separated from the filter stages by a high impedance (1 MΩ, 12 pF) unity gain buffer (UGB). The first amplifier stage A_1 with a gain of 4.64 is combined with the third-order filter, the components are calculated from the first three poles taken from the 7-pole analog part of the 13-pole mixed-mode system. The following two second-order filter stages A_2 and A_3 have the same gain as the first stage, whilst their components are calculated from the next two complex conjugate pairs of poles from the same bunch of 7. All three amplifying stages are inverting, so the final inversion must be done either at the signal display, the digital filter routine, or simply by taking the 2's complement of the ADC's digital word.

7.2.10 Standardization of Component Values

More often than not, multi-stage filters will have awkward component values, far from either of the closest preferred standard E12 or E24 values. In addition the greater the number of stages, the larger will be the ratio of maximal to minimal values, forcing the use of components with very tight tolerance.

Fortunately we are not obliged to use exactly the calculated values; indeed, we are free to adjust them, paying attention that each stage keeps its time constants as calculated. I.e., the capacitors will be probably difficult to obtain in E24 values, but resistors are easily available in E48 and even E96 values, so we might select the closest E12 values for the capacitors and then select the resistors from, say, E48.

Considering for example the first 2-pole stage (A_2) we could use 18 pF for C_{21} (instead of 18.5); then C_{22} would be 210 pF (say, 180 and 30 pF in parallel), the resistors R_{21} and R_{23} can be set to 226 Ω and R_{22} can be set to 1050 Ω.

The other two stages can be changed in a similar way. It is advisable not to depart from the initial values too much, in order to keep the impedances close to the driving capability of the amplifiers (remember that each amplifier has to drive both the input impedance of the following stage and the impedance of its own feedback network) and also to remain well above the influence of stays (low value capacitances and the amplifier inverting inputs are the most sensitive in this respect).

7.2.11 Concluding Remarks

The initial design requirement for the procedure described is probably an overkill, since we shall very rarely have the noise level or some other interfering signal as high as the full ADC amplitude range at the Nyquist frequency limit or above. If we are confident that the maximum disturbance level at the Nyquist frequency will always be some 30 dB lower than the full range amplitude, we can relax the filter specifications accordingly, either by having a less steep and less complicated filter, or by raising the filter's cut off frequency so that the attenuation at the Nyquist frequency intersects the level 30 dB above the ADC resolution.

For the example above, the maximum input signal was 40 mV, so, in the case of an interfering signal of, say, 1.3 mV (−30 dB), our filter would need an attenuation of about −42 dB at the Nyquist frequency. For the 7th-order analog part of the filter, having the attenuation slope of $-140\,\mathrm{dB}/10f$, this would result in a nearly doubled bandwidth (the digital part remains the same), but note that the alias spectrum can now extend down to DC, since its source spectrum includes the sampling frequency. Also, as we have seen at the beginning of this section, the $(\sin \omega T_\mathrm{s})/\omega T_\mathrm{s}$ spectral envelope allows us a further 12–13 dB at about $0.7 f_\mathrm{s}$. In such a case an additional filter stage, with zeros at f_N and f_s and the third harmonic of f_N (all on the imaginary axis), such as an inverted (type II) Chebyshev filter, could be used to improve the alias rejection at low frequencies without spoiling the upper roll off slope of the frequency response by much, thus also preserving the smooth step response.

Résumé and Conclusion

We have shown only a small part of the vast possibilities of use offered by the application of numerical routines, either for the purpose of system design or for implementing them within the system's digital signal processing.

Some additional information and a few examples can be found on the disk included with the book, in form of the '*.M' files to be run within Matlab. Many of those files were used to produce the various figures in the book and can be used as a starting point for further routine development.

One of the problems of writing a book about a fast developing subject is that by the time the writers have finished the editing, several years might have passed and the book is no longer at the forefront of the technology's development.

We have tried to prevent the book from becoming outdated too soon by including all the necessary theory and covering the fundamental design principles which are independent of technology, and thus of lasting value. We have also tried to cover some of the most important steps in the development from a historical point of view, to show how those theoretical concepts and design principles have been applied in the past and how they have evolved to today's forms.

Although the importance of staying alert and following the new developments and ideas is as high as ever, the knowledge of the basic theory and its past applications helps us to identify more quickly the correct paths to follow and the aims to pursue.

(blank page)

References:

[7.1] *J.N. Little* and *C.B. Moller*, The MathWorks, Inc.:
PC-MATLAB For Students (containing disks with Matlab program)
Prentice-Hall, 1989

MATLAB-V For Students (containing a disk with Matlab program)
Prentice-Hall, 1998

Web link : <http://www.mathworks.com/>

See also the books on electronics + Matlab at:
<http://www.mathworks.com/matlabcentral/link_exchange/MATLAB/Books/Electronics/>

[7.2] *C.E. Shannon*, Collected Papers,
IEEE Press, Cat. No.: PC 0331

See also:
<http://en.wikipedia.org/wiki/Nyquist-Shannon_interpolation_formula>

[7.3] *A.V. Oppenheim* and *R. W. Schafer*, Digital Signal Processing,
Prentice-Hall, 1975

[7.4] *K. Azadet*, Linear-phase, continuous-time video filters based on mixed A/D structure,
ECCTD 1993 - Circuit Theory and Design, pp. 73–78, Elsevier Scientific Publishing

[7.5] *E. Margan*, Anti-aliasing with mixed-mode filters,
Electronics World and Wireless World, June, 1995, pp. 513–518
<http://www.softcopy.co.uk/electronicsworld/>