Digital Filtering With SigLib

Numerix Ltd.

March 2004

Numerix Ltd.
7 Dauphine Close, Coalville, Leics, LE67 4QQ, UK
Phone : +44 (0)208 020 0046, Fax : +44 (0)208 020 0047
Internet : http://www.numerix-dsp.com
Email : support@numerix-dsp.com

Introduction

This applications note answers some common FAQs on the implementation of digital filters and it provides advice about which SigLib functions can be used to achieve the desired functionality. All filter coefficients used in this applications note were generated using the Digital Filter Plus design package, which is available from : http://www.numerix-dsp.com/dfplus.

FIR Filters

One common method for modifying the characteristics of a digital FIR filter is to use modulation techniques to adjust the coefficients to move the centre / cut-off frequencies. For example a low-pass filter with a cut-off frequency (Fc) of 1KHz can be modified to a band-pass filter with centre frequency 6KHz and bandwidth 2KHz by multiplying the coefficients of the LPF with a sinusoid of frequency 6KHz. This shifts the centre frequency from 0Hz to 6 KHz.

If we wish to convert a low-pass filter into a high pass one then we can use this technique and the sinusoid has the same frequency as the Nyquist frequency. If we look at a digital pure sine wave at the Nyquist frequency then all the coefficients are 0 but if we look at a cosine then the values are +1, -1, +1, ….. etc. So multiplying the filter coefficients by this cosine is equivalent to negating every alternate coefficient.

SigLib Functions To Use

FIR Filtering :

SIF_Fir	- Initialise FIR filter functionality
SDS_Fir	- Perform FIR filter on a data sample
SDA_Fir	- Perform FIR filter on a array of data

IIR Filters

One common method for designing digital Infinite Impulse Response (IIR) filters is to design a generic low-pass filter using a standard methodology such as Butterworth, Elliptic etc, with normalised frequency characteristics, and then shift the pass / stop bands to the desired locations.

The trick to manipulating the characteristics of a digital filter is to use normalised frequencies i.e. we map the cut-off frequencies to a normalised sample rate of 1.0 Hz. We can now design a normalised set of filter coefficients and then move the cut-off frequencies to meet any particular requirements. We can also do this at design time or run time because modifying the coefficients is just a matter of calling one simple function.

SigLib includes two functions for shifting the cut-off frequencies : SDA_IirLpLpShift to shift the cut-off frequency space of a low-pass filter and SDA_IirLpHpShift to convert the low-pass filter into a high-pass type and also shift the cut-off frequency. Of course band-pass filters can be constructed from a low-pass filter in series with a high-pass filter and band-stop filters can be constructed from a low-pass filter in parallel with a high-pass filter so with these two simple functions we can generate filters to meet any specific requirements.

One common problem with modifying the frequency characteristics of an IIR filter is that the filter pass-band gain (D.C. for a low-pass filter and the Nyquist frequency for a high pass filter) will usually be modified during the process. If the resulting gain is lower than the original then this can cause numerical underflow within the filter and a loss of precision. Alternatively, if the new gain is larger then this can result in numerical overflow. Both of these situations should be avoided and are more likely to occur when using fixed point numbers. To help alleviate this problem both the SDA_IirLpLpShift and SDA_SDA_IirLpHpShift function returns the gain scaling factor at the centre frequency of the filter so that the application can account for the gain change. There are two options for handling this :

1/ Modify the magnitude of the input or output data by the scaling factor to maintain the required pass-band gain. This can be performed using the SDA_Multiply function in SigLib.

2/ Use the function SDA_ModifyIirFilterGain to adjust coefficients of the filter such that the gain of the filter at the centre frequency of the filter has the desired value.

Option 2 is usually the preferred method because it maintains the maximum dynamic range of the signal. This is especially true when using fixed point numbers.

SigLib Functions To Use

IIR Filtering :

SIF_Iir	- Initialise IIR filter functionality - uses the biquad filter structure
SDS_Iir	- Perform IIR filter on a data sample - uses the biquad filter structure
SDA_Iir	- Perform IIR filter on a array of data - uses the biquad filter structure
SIF_OrderNIir	- Initialise the order N IIR filter functionality - uses a single order N structure
SDS_OrderNIir	- Perform an order N IIR filter on a data sample - uses a single order N structure
SDA_OrderNIir	- Perform an order N IIR filter on a array of data - uses a single order N structure
SDA_IirLpLpShift	- Shift the cut-off frequency of an IIR digital filter
SDA_SDA_IirLpHpShift	- Transform a LPF IIR filter into a HPF and shift the cut-off frequency

SigLib Example

As an example, a low-pass filter for voice band telecommunications might require a 6th order Butterworth characteristic, a sample rate of 9600 Hz and a cut-off frequency of 1200 Hz. If we normalise the sample rate to 1.0 Hz then the normalised cut-off frequency is 9600 / 1200 Hz or 0.125.

If we design a normalised set of coefficients then we could choose any Fc however a convenient choice would be Fc = 0.1 Hz and the normalised coefficient array for a 6th order Butterworth filter would look as follows :

```
SFLOAT NormalisedCoefficientArray [ ] =
{
        6.090963e-02, 1.218192e-01, 6.090963e-02,
        -1.032069e+00, 2.757079e-01,

        6.745527e-02, 1.349105e-01, 6.745527e-02,
        -1.142980e+00, 4.128015e-01,

        8.288257e-02, 1.657651e-01, 8.288257e-02,
        -1.404384e+00, 7.359151e-01
};
```

We can now use the function SDA_IirLpLpShift to move the cut-off frequency to our desired value (0.125), as follows :

```
      Scale = SDA_IirLpLpShift (NormalisedCoefficientArray,
ModifedCoefficientArray, 0.1, 0.125, 1.0, 3);
```

The function prototype and parameter description is are as follows :

```
SFLOAT SDA_IirLpLpShift (const SFLOAT *,      /* Source coefficients */
                         SFLOAT *,            /* Destination coefficients */
                         const SFLOAT,        /* Original cut-off frequency */
                         const SFLOAT,        /* Required cut-off frequency */
                         const SFLOAT,        /* Sample rate */
                         const SFIX);         /* Number of biquads */
```

The following array shows the modified filter coefficients :

```
SFLOAT ModifedCoefficientArray [ ] =
{
        8.701455e-02, 1.740291e-01, 8.701455e-02,
        -8.402869e-01, 1.883451e-01,

        9.763107e-02, 1.952621e-01, 9.763107e-02,
        -9.428090e-01, 3.333333e-01,

        1.237912e-01, 2.475824e-01, 1.237912e-01,
        -1.195433e+00, 6.905989e-01
};
```

We could of course now take our original normalised filter coefficients and use them in any desired application with any sample rate and just shift the cut-off frequency as required.

A complete example of this technique is provided in the SigLib library with the testifs2 programs in the Example and DSPExample directories.

Phase Shift

Fundamentally, all filters shift the phase of a signal. This is due to the nature of the filter structure, which includes delay elements combined with the Multiply-Accumulate operations.

Some aspects of filter delay and hence phase shift can be controlled, for example a linear phase shift – where the group delay is constant and hence so is the relative phase shift – can be created by using a filter with a symmetrical impulse response. For causal filters this means that the IIR structure can not provide a linear phase response.

There is nothing that any of us can do to prevent phase or time delays in digital filters but there are ways to work around it. You will need to work out what the phase shift is within the pass-band of the filter and adjust for this by delaying the signal a suitable number of samples to adjust it accordingly. If you have a specific frequency that you are filtering then this is usually not a problem because the delay can be calculated. The problem comes if you have a broadband signal then this can be problematical because the delay at each frequency will be different so you can only account for one frequency.

For linear phase FIR filters the delay – referred to as Group Delay – is a constant value that is typically measured in number of samples or milliseconds. For a filter implementation with C/C++ indexing (from zero) this delay is equivalent to (filter length – 1) / 2 samples. So, for example, a filter length of 50 coefficients will give a group delay of 24.5 samples, which is difficult to compensate for. A filter length of 51 coefficients however will give a group delay of 25 samples, which is an integer number and hence much easier to compensate for. This is the reason why most linear phase FIR filters are designed with an integer number of coefficients.

Filter delays are covered in more detail in chapter 19 of the “The Scientist and Engineer's Guide to Digital Signal Processing” [1].

SigLib Functions To Use

SDS_Delay	- Delay the data by a number of samples
SIF_BlockDelay	- Initialise a block delay
SDA_Delay	- Delay the data in an array by N samples

References

[1] “The Scientist and Engineer's Guide to Digital Signal Processing”, Steven W. Smith, Ph.D., California Technical Publishing, ISBN 0-9660176-3-3 (1997). This is available for download from : http://www.dspguide.com.

[2] Lyons R. G., (1999), Understanding Digital Signal Processing, Addison-Wesley, USA.

[3] Oppenheim, A. V. and Schafer, R. W., (1989), Discrete Time Signal Processing, Prentice-Hall Inc., USA.