ME 333 Introduction to Mechatronics
Assignment 6: Digital Finite Impulse Response (FIR) Filters
Electronic submission due before 12:30 PM Tuesday February 22

The PIC32 is a powerful tool for digital signal processing (DSP), for example for filtering a signal, computing a Fourier transform (FFT) to find the frequency content of a signal, or video decoding. Some efficient code written for these functions for the PIC32 are summarized in Chapter 3 of the Microchip 32-Bit Language Tools Libraries manual, included in your Microchip distribution. It is not the purpose of ME 333 to discuss DSP in any detail, but its importance is such that we at least need to recognize the possibility of digitally filtering a signal. In this homework we will focus on using a digital Finite Impulse Response (FIR) filter to low-pass filter a signal, something we did previously with an RC filter in circuitry.

RC Low-Pass Filter (LPF)

As we've discussed before, an RC LPF has a cutoff frequency of $1/(2\pi RC)$ when expressed in Hz, or $1/RC$ when expressed in radians/s. The response of the filter as a function of frequency, i.e., the *frequency response*, can be visualized in a Bode plot, as shown in Figure 1. (In this assignment we focus on the magnitude response, not the phase response.) The Bode magnitude plot indicates the gain of a signal as it passes through the filter. For example, if we put a sinusoid of a frequency ω and amplitude A into the input, the plot tells us that the amplitude of the output will be GA, where G is the "gain" at the frequency ω. (The output will also be a sinusoid of frequency ω, due to the linearity of the filter. We will assume all our filters are linear.)

Usually a Bode plot will plot the frequency on a log scale, as shown in the figure. Also, the magnitude will be plotted as 20 log G, in units of decibels (dB). So a gain G = 1 (the output amplitude is equal to the input amplitude) is 20 log 1 = 0 dB. A gain G = 10 is 20 dB, and a gain G = 0.1 is -20 dB. Every factor of 10 in G corresponds to 20 dB.

The cutoff frequency of a low-pass filter is typically defined as the frequency at which the magnitude is -3 dB, or $G = 10^{(-3\ dB/20)} = 0.71$. This is where the power of the signal (proportional to the amplitude squared) is half of the input power. For our RC filter, with magnitude response as shown in the figure, the cutoff frequency is 1 rad/s, or about 0.16 Hz. Thus an input sinusoid at 1 rad/s would pass to the output at 0.71 times the amplitude. At lower frequencies (e.g., 0.1 rad/s and less), the gain is approximately 1. At higher frequencies, the gain drops by a factor of 10 (i.e., 20 dB) for each factor of 10 increase in the input frequency.

If we are using our LPF on a noisy sensor signal, we usually assume that the signal we care about is at low frequencies, and high frequency content is noise. Thus we would design the LPF to preserve the signals we care about and cut off higher frequencies. If we took the FFT of the original sensor signal, and the FFT of the filtered sensor signal, we should see that the filtered version has little power content at high frequencies.

A First Digital LPF: The Moving Average Filter (MAF)

If we don't want to use external components, or if the signal is not a simple analog voltage, a circuitry LPF is no longer an option. In this case, the first thing to try would be to take a running average of the sensor signals. For example, we could set the output of the filter to be equal to the average of the previous k samples. This is called a Moving Average Filter (MAF). If we choose k=1, we just get the original (sampled) signal. As we increase k, the filtered output gets smoother as higher frequencies get "averaged" out. We should choose k large enough to get rid of the frequencies we don't want, but small enough that we can still see the signal frequencies we care about.

If we choose k=4, for example, our output is the average of the four previous inputs:

out[n] = 0.25*in[n-3] + 0.25*in[n-2] + 0.25*in[n-1] + 0.25*in[n].

We can plot the gain of our MAF as a function of frequency, just like we did with the RC LPF. One fundamental difference is that the RC filter is an analog filter working on continuous time signals, while our MAF is a digital filter working on sampled versions of continuous time signals. As an example, imagine you are sampling at a frequency f, and the signal you are sampling is a pure sinusoid with frequency f. Then each sample you take will have the same value, and the signal of frequency f will be indistinguishable from a DC (constant) signal.

More generally, assume that we are sampling at frequency f, and the signal we are sampling is a pure sinusoid at frequency $f/2 + f_{extra}$, where $f_{extra} < f/2$. Then the sampled signal will "appear" to be a signal of frequency $f/2 - f_{extra}$. When a high frequency signal appears to be a lower frequency signal due to the sampling process, this is called *aliasing*. Thus f/2 is an important frequency, known as the Nyquist frequency. If our sampling frequency is f = 2000 Hz, then $f_{Nyquist}$ = 1000 Hz.

So now when we plot our frequency response, we will only plot it from 0 to $f_{Nyquist}$. On the horizontal axis, we will normalize frequencies by dividing by $f_{Nyquist}$, so our frequencies run from 0 to 1.

Now let's use Matlab to plot the frequency magnitude responses of two MAFs: one that averages three samples and one that averages 11 samples. See Figure 2. This figure was generated by the Matlab command

b=ones(3,1)/3; freqz(b); hold on; b=ones(11,1)/11; freqz(b)

where b is the vector of coefficients that multiply the samples and the Matlab command freqz plots the frequency response corresponding to the coefficients. We can see that the cutoff frequency is approximately 0.3 times the Nyquist frequency for the three-sample MAF, and approximately 0.1 times the Nyquist frequency for the 11-sample MAF. This agrees with our intuition: the average that uses more samples should cut off more frequencies than the average that uses fewer samples.

Some things to note about the magnitude response plots: (1) The frequencies only run from 0 to 1 (the Nyquist frequency), as we noted before, and (2) there is significant "ripple," where the response switches from decreasing to increasing and vice-versa.

1. If we have input frequencies in the range [1, 2], i.e., [Nyquist frequency, sampling frequency], the magnitude response is simply the mirror image about a vertical line through the Nyquist frequency. (This is apparent by our reasoning above, where a signal of frequency $f/2 + f_{extra}$,

appears to be a signal of frequency f/2 - f_{extra}.) Then the magnitude response repeats for higher frequencies, since a frequency F + f is indistinguishable from a frequency F.

2. The ripple is not something we saw with our simple RC LPF, but is a common feature of digital filters. Note, for example that the three-sample MAF response drops to zero (negative infinity dB) exactly at 2/3 the Nyquist frequency. This is easy to understand: if we are sampling at frequency f, and the signal we are sampling is a pure sinusoid at frequency (f/2)*(2/3) = f/3, then we are getting exactly three evenly-spaced-in-time samples per waveform, and they will always sum (average) to zero. This is not true at slightly different frequencies, hence the ripple.

So are these MAFs good low-pass filters? An ideal low-pass filter would pass the desired frequencies and then drop immediately to zero gain at the cutoff frequency (a "brick wall" filter). By contrast, even our 11-sample MAF is passing some high frequencies with a gain of up to 0.1 (-20 dB). MAFs are easy to understand, but we can do better.

Finite Impulse Response (FIR) Filters

In our examples above, each sample received the same weight. Instead, we could give them different weights. (The weights should still sum to one, though.) We could write our filter more generally as

out[n] = b[k-1]*in[n-k+1] + b[k-1]*in[n-k+2] + ... + b[0]*in[n].

This is called a Finite Impulse Response (FIR) filter. The "finite" in "FIR" refers to the *impulse response* of the filter---the time response you get when all inputs are zero, then a single input has value 1 (the impulse), then all future inputs are zero. The response of the filter will become zero again after k time steps from the impulse, i.e., in finite time. This is in contrast to an Infinite Impulse Response (IIR) filter, which may have a nonzero impulse response for all time after the impulse. (Though the impulse response of a stable IIR does tend to zero as time goes to infinity. We will describe IIRs briefly later, but they are not the focus of this assignment.)

The *order* of the FIR filter is equal to the number of samples k minus 1. Thus a first-order FIR filter uses only two samples. The higher the order of the filter, the better the behavior you can get, at the expense of more computation. (In the analog domain, our simple RC filter is a first-order filter.)

Matlab provides a number of options for designing digital filters. One simple example is fir1. (The last character is a one, not an ell.) Try "doc fir1" to learn more about it. To design a tenth-order FIR LPF with cutoff frequency of 0.25 times the Nyquist frequency, we can type

b = fir1(10,0.25)

which yields the eleven coefficients

b = [-0.0039 0.0000 0.0321 0.1167 0.2207 0.2687 0.2207 0.1167 0.0321 0.0000 -0.0039]

plotted in Figure 3.

Using

b = fir1(10,0.25); freqz(b); hold on; b = fir1(30,0.25); freqz(b)

we can plot the frequency response of a tenth-order and a 30th-order FIR LPF designed by Matlab. See Figure 4. The 30th-order LPF is apparently better than the tenth-order LPF by any measure: flatter response at the pass-band frequencies, faster drop-off at the cutoff frequency, less ripple and greater attenuation at higher frequencies.

You can use fir1 to design high-pass, band-pass, and band-stop filters, too. To move beyond fir1 (which uses a simple window design method), try Matlab's fdatool for optimal filter design. We won't go into that here.

Infinite Impulse Response (IIR) Filters, Briefly

An IIR filter uses past outputs, as well as the current and past inputs, to calculate the current output. It can be written

a[j-1]*out[n-j+1] + ... + a[0]*out[n] = b[k-1]*in[n-k+1] + ... + b[0]*in[n]

where k is the number of inputs and j is the number of outputs in the equation. out[n] is easily calculated by moving the out[n-...] terms to the right-hand side of the equation and dividing by a[0]. IIR filters open up a number of new possibilities for design, including popular Butterworth and Chebyshev filters.

Differencing Filters

If you want to calculate the derivative of a signal, you can use the simple differencing filter

out[n] = (in[n] - in[n-1])/dt,

where dt is the sample time. If your original signal was noisy, the high-frequency noise will be amplified by this differencing filter. You should either low-pass filter your original signal, then use the differencing filter, or use the differencing filter and then low-pass filter the result. If your filters are linear (sums of coefficients multiplied by samples) and there are no nonlinearities introduced by your computation (e.g., saturation), then the result will be the same whether you LPF and then difference, or difference and then LPF.

Causal and Acausal FIR Filters

A *causal* (or real-time) filter is one that calculates its output based only on current and previous inputs. An *acausal* filter is one that uses future inputs too. For example, a causal filter might be

out[n] = b[2]*in[n-2] + b[1]*in[n-1] + b[0]*in[n]

whereas an acausal filter might be

out[n] = b[2]*in[n-1] + b[1]*in[n] + b[0]*in[n+1].

The first formulation (causal) is used for real-time applications, where you need the current filtered value now, for real-time control. The second formulation (acausal) is used when you have collected a bunch of data and then are post-processing the data (not real-time control).

Because causal filters use only past data to calculate the current value, there is an inherent delay in the (smoothed) output signal relative to the input signal, and this delay gets larger as you use more past samples (higher order filters). This is an issue for real-time control, as control to delayed information can lead to instabilities. So you should balance your desire for a smooth sensor signal with delays due to high-order filters.

If you are post-processing data, you should always use an acausal filter. This gets rid of the delay in high-order filters.

Implementing an FIR Filter

The process of calculating the output as a function of the input signal is one of "convolving" the filter coefficients with the signal. If the filter and signal are each stored in an array, it is a simple process of implementing a for loop to run over the elements of the filter array, multiply by the appropriate elements of the signal array, and sum, to get an element of the output array. You may be able to think of efficient ways to implement this, but any simple implementation is fine for this assignment.

Questions

1. Plot the magnitude response plot of a fifth-order (six-sample) MAF in Matlab. What is the cutoff frequency? What fraction of a sinusoid at 40% of the sampling frequency makes it through the filter? If the signal is a sinusoid at 1.8 times the Nyquist frequency, what is the filter's gain for this signal?

2. Imagine you are sampling a signal at 100 kHz. Design an FIR LPF (using fir1) that passes all frequencies below 20 kHz with a gain of greater than 0.7, and all frequencies at 35 kHz and above (up to the Nyquist frequency) with a gain of less than 0.01. What is the Nyquist frequency? What dB should the 35 kHz and above portion of the output be under? What is the lowest-order FIR filter (designed with fir1) that will accomplish this? Print the magnitude response of this filter and label the cutoff frequency and 0.01 gain line.

3. Plot the coefficients of a 48th-order FIR LPF designed by Matlab for a cutoff frequency of 0.2 times the Nyquist frequency. Use `stem`, not `plot`, to show the coefficients.

4. You can use fir1 to create high-pass, band-stop (notch), and band-pass filters, too. Consult Matlab documentation and design a 50th-order band-stop (notch) filter to remove 60 Hz noise. Your stop band will be 50 Hz to 70 Hz, and your sampling frequency is 400 Hz. Turn in the command you used to create the filter coefficients, a stem plot of the coefficients, and the frequency response showing the stop band.

5. Let's say you want to low-pass filter a signal with a four-sample MAF, then difference it to get the derivative of the signal. There are two filters involved here, one with coefficients [0.25, 0.25, 0.25, 0.25] and the other with coefficients b[0]=1 and b[1]=-1 (since we'll assume dt=1). These two filters can be combined to make a single filter with five coefficients, b[0]...b[4]. This filter can be obtained by convolving the two filters (or you can do it by brute force, with an example input). Keep in mind that the filters implicitly have zeros at either end of the coefficient list. Give the coefficients b[0]...b[4] of the new filter and plot the frequency response.

Programming

For this part of the assignment, you are given a prerecorded signal called SIGNAL in assignment6_signal.h. SIGNAL was recorded at 1 kHz for 1 second. It has frequency components of 2 Hz, 20 Hz, and 400Hz.

Write a program that can implement a general FIR filter. Then use it to low-pass filter SIGNAL in the following ways:

1. With an acausal MAF using 13 samples (six before and six after) each data point.
2. With the same filter as above, but implemented causally.
3. With a 12^{th} order causal FIR filter with a cutoff frequency of 100 Hz with coefficients designed in Matlab.

Submit to Blackboard, in a zipped file:

- A plot of SIGNAL with the results of your three filters

- Your assignment6.c file and any code used to apply the filters
- A pdf of your answers to the 5 questions listed above

Note:
A bare-bones code template has been provided in Assignment6.zip. It contains some minimal code and hints in assignment6.c, our usual NU32v2 board functions in NU32v2.c and NU32v2.h, SIGNAL in assignment6_signal.h, and the usual procdefs.ld.

Also:

Cut 8" of black, red, white and blue wire. Strip each end and bring for class on Tuesday 2/22.