ECE 4680 DSP Laboratory 4: 
FIR Digital Filters

Due 12:15 PM Friday October 31, 2014

Introduction

Finite Impulse Response Basics

Chapter 3 of the course text deals with FIR digital filters. In ECE 2610 considerable time was spent with this class of filter. Recall that the difference equation for filtering input \( x[n] \) with filter coefficient set \( h[n] \), \( n = 0, \ldots, N \), is

\[
y[n] = \sum_{k=0}^{N} h[k] x[n-k].
\] (1)

The number of filter coefficients (taps) is \( N + 1 \) and the filter order is \( N \). The coefficients are typically obtained using MATLAB’s filter design function, \texttt{fdatool()}\texttt{.} The filter impulse response is

\[
h[n] = \sum_{k=0}^{N} h[k] \delta[n-k]
\] (2)

the filter frequency response is

\[
H(e^{j\omega}) = \sum_{k=0}^{N} h[k] e^{-jk\omega}
\] (3)

and the filter system function (\( z \)-domain representation) is

\[
H(z) = \sum_{k=0}^{N} h[k]z^{-1} = h[0] + h[1]z^{-1} + \cdots + h[N]z^{-N}.
\] (4)

From the system function it is clear why the filter order is \( N \). The highest negative power of \( z \) is \( N \).

Once a set of filter coefficients is available an FIR filter can make use of them. In MATLAB code this is easy since we have the \texttt{filter()} function available. In MATLAB suppose \( x \) is a vector input signal values that we want to filter and vector \( h \) contains the FIR coefficients. We can obtain the filtered output vector \( y \) via

\[
>> y = \text{filter}(h,1,x);
\]

In this lab we move beyond the use of MATLAB for filtering signals, and consider the real-time
implementation of a sample-by-sample filter algorithm in C/C++. In text Section 3.4 a series of refinements is considered.

**Real-Time FIR in C**

At the core of FIR filtering is the following code that implements (1) for sample-by-sample processing. By sample-by-sample I mean that each time the ISR fires a new sample has arrived at the ADC and a new filtered sample must returned to the DAC. The first for loop is in fact the sum-of-products represented by (1). The array \( x_{\text{buffer}} \) holds \( x[n-k] \) for \( k = 0, \ldots, N \) (here \( N_{\text{FIR}} \) is equivalent to \( N \) in (1)). The second for loop updates the filter history by discarding the oldest input, \( x[n-N] \), and sliding all the remaining samples to the left one position. The most recent input, \( x[n] \), ends up in \( x_{\text{buffer}}[1] \) to make room for the new input being placed into \( x_{\text{buffer}}[0] \) on the next call of the ISR.

```c
//Work with Left ADC sample
x_buffer[0] = 0.25 * CodecDataIn.Channel[ LEFT];
//Use the next line to noise test the filter
//x_buffer[0] = 0.125* ((short) rand_int()); //scale input by 1/8

//Filtering using a 32-bit accumulator
for (i=0; i< N_FIR; i++)
{
    result += x_buffer[i] * h[i];
}

//Update filter history
for (i=N_FIR-1; i>0; i--)
{
    x_buffer[i] = x_buffer[i-1];
}

//Return 16-bit sample to DAC
CodecDataOut.Channel[ LEFT] = (short) result;
```

**Expectations**

When completed, submit lab report which documents code you have written and a summary of your results. Screen shots from the scope and any other instruments and software tools should be included as well. I expect lab demos of certain experiments to confirm that you are obtaining the expected results and knowledge of the tools and instruments.

**Problems**

**Measuring Filter Frequency Response Using the Network Analyzer**

1. The Comm/DSP lab has test equipment that be used to measure the frequency response of
an analog filter. In particular, we can use this equipment to characterize the end-to-end response of a digital filter that sits inside of an A/D-H(z)-D/A processor, such as the OMAP-L138. The instructor will demonstrate how to properly use the Agilent 4395A vector/spectrum analyzer for taking frequency response measurements of the WinDSK8 equalizer app.

In text Section 3.2 you find a description of a real-time graphic equalizer application. Up to this point you have not played with any of the winDSK (now winDSK8) applications. The first order of business is learning how to run winDSK8 apps on the PC that communicate with the OMAP-L138 via RS232. The key is setting up a serial port connection (RS232) between the host PC and the serial port connector on the OMAP board. DIP switch S2 on the board needs to be configured as shown in Figure 1, depending upon the capability of your PC serial port. Note switch S2-1 must be ON in order to enable the winDSK8 kernel.

You would like to run the serial connection at the highest possible rate. Experience however has shown that when using serial cables found in the lab and the current lab computers, the maximum baud rate is likely just 115200 bits/s. When I use a USB-to-serial dongle, as shown in Figure 1, I am able to get the full rate of 921600 bits/s as supported by the board.

<table>
<thead>
<tr>
<th>S2-2</th>
<th>S2-3</th>
<th>Baud rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFF</td>
<td>OFF</td>
<td>115200</td>
</tr>
<tr>
<td>ON</td>
<td>OFF</td>
<td>230400</td>
</tr>
<tr>
<td>OFF</td>
<td>ON</td>
<td>460800</td>
</tr>
<tr>
<td>ON</td>
<td>ON</td>
<td>921600</td>
</tr>
</tbody>
</table>

Note: Using the PC serial port on the current lab PCs 115200 seems to be the best that can be done. With a USB-to-serial dongle I can get 921600.

Figure 1: Configuring the OMAP-L138 s2 DIP switches serial port communications.
Turning to the vector network analyzer consider the block diagram of Figure 2. You use the vector network analyzer to measure the frequency response magnitude in dB as the ratio of the analog output over the analog input (say using analyzer ports B/R). The phase difference formed as the output phase minus the input phase, the phase response, can also be measured by the instrument, although that is not of interest presently. The frequency range you sweep over should be consistent with the sampling frequency. If you are sampling at 48 ksps what should the maximum frequency of interest be?

Plots from the 4395A can be obtained in several ways. One approach is to bring the captured data directly into MATLAB using the HP-IB data bus. Helper files can be found at http://www.eas.uccs.edu/wickert/ece4670/. A second approach is to use the instruments capability to save an Excel data file or a tiff image file to the 3.5in floppy drive on the Agilent 4395A. Note portable floppy drives can be found in the lab for connecting to the PC’s, and hence provides a means to transfer data from the network analyzer to the PC via sneaker-net. You will likely takes this option. Thirdly, there is a 4395A PC screen capture utility described on the ECE4670 Web Site.

When you first launch winDSK8 you are presented with the dialog shown in Figure 3. Under DSP-Board configuration you need to set DSP to Zoom and the Sample Rate to 48 kHz. Next set the Host Interface Configuration to the appropriate COM port. Use the PC Device Manager app to see which port number is assigned to the serial port on the back of the lab PC. Choose this port as the COM Port (likely only one COM port is found for the lab PCs, so choose this port). Set the Baud rate to the highest possible rate that works. This requires some experimentation. Set the rate, set the S2 DIP switches accordingly, and then test by clicking Get Board Version to see if you have a valid connection. Likely you will
have to settle for the lowest rate.

Figure 3: winDSK8 start-up dialog and app chooser.

Now it's time to launch the Graphic Equalizer app. Clicking the button shown in Figure 3 will push an OMAP-L138 executable down to the board and start the equalizer as shown in Figure 4. In measuring the frequency response of the equalizer you will only test one channel. In Figure 4 you can see that only the right sliders for each frequency band have been manipulated above and below the nominal 0 dB (center position) slider positions.

a) Obtain the frequency response in dB versus frequency for the two approximate slider settings of Figure 4 (a & b). Since the filter can have greater than unity gain over selected frequency bands, be sure that the input signal from the network analyzer does not overload the system. The concern is that the filter output in the digital domain will overdrive the DAC inputs. Compare your results to the network analyzer results I obtained using the Analog Discovery, as shown in Figure 5a.

b) Since all of the filters are in parallel (see Figure 3.4 of the text), it is hard to discern the frequency response of just one filter. To get a better idea of the 4.32 kHz band filter frequency response, push the gain on this filter to the maximum and push the gain on the remaining filters each to the minimum. Capture and record the frequency response, measuring the lower and upper 3 dB frequencies of the bandpass response.
For a really accurate comparison of measured versus theory, the filter coefficients are available on the book’s CD under the Chapter 11 CCS code. You can use MATLAB’s `freqz()` function to obtain the frequency response scaled to a sampling rate of 48 kHz. This is an optional exercise for part (b).

c) Using an audio music source, such as your cell phone music player, connect the source to the OMAP-L138 inputs and connect the powered PC speakers to the output. Listen to the impact of the equalizer on shaping the music spectrum. Demonstrate this to your instructor.

---

Figure 4: The winDSK8 equalizer app running with two different filter gain settings.
Figure 5: Network analyzer sweeps of the equalizer from the Analog Discovery.
2. Now it's time to design and implement your own FIR filter using MATLAB's `fdatool` (see Figure 5) and implement it using the filter algorithm described in the introduction to this lab.

To introduce the design procedure using MATLAB's `fdatool`, consider an equiripple design using a sampling frequency of $f_s = 8$ kHz with passband and stop critical frequencies of $f_p = 1600$ Hz and $f_s = 2000$ Hz respectively, and a passband ripple of $\pm 0.5$ dB (1 dB total), and a stopband attenuation of 60 dB. The `fdatool` design is shown in Figure 6.

![Figure 6: An $f_s = 8$ kHz lowpass design example using MATLAB's `fdatool`.](image)

A complete CCS project for this filter, including a MATLAB file for creating C header files from and FDA Tool design, can be found in the ZIP package Lab4.zip. The project name in CCS is `fir_float_AIC3106` and the ISR code is in `ISRs_fir_float.c`. Using FDA tool you export your design to the MATLAB workspace by choosing Export... from the
file menu. Then choose export as Objects as shown in Figure 7. Once the object is in the

![Figure 7: Exporting an FDA tool design to the workspace as object Hd.](image)

MATLAB workspace you can then use the function `filter_C_header.m` (found in the Lab 4 zip package) to create the header file similar to that already found in the project as shown in Figure 8. From the MATLAB command window this function creates the .h file as follows:

```
>> filter_C_header(Hd,'float','fir_fltcoeff.h',3)
```

For the filter design shown in Figure 6 and having the coefficient header file as shown in Figure 8, the swept frequency response, as obtained using the Analog Discovery, is shown in Figure 9.

![Figure 8: Sample filter coefficient header file found in the ZIP package already.](image)
Your assignment is complete a design using a sampling rate of 48 kHz having an equiripple FIR lowpass response with 1dB cutoff frequency at 5 kHz, a passband ripple of 1dB, and stopband attenuation of 60 dB starting at 6.5 kHz. See Figure 10 for a graphical depiction of these amplitude response requirements.

When the times comes to get your filter running you just need to move your h-file into the project and then include it in the ISRs_fir_float.c file (comment out or remove #include "fir_fltcoeff.h") and re-build the project. Make special note of the fact that the algorithm in the ISR only passes the left channel codec signal through your FIR filter design. The right channel goes straight through from input to output. If you should sweep this channel by accident it will result in a lowpass response, but the cutoff frequency is fixed at $f_s/2$ Hz (in this case 24 kHz).

**Figure 9:** Measured response of the filter design shown in Figure 5.

**Figure 10:** Equiripple lowpass filter amplitude response design requirements.
// A portion of ISRs_fir_float.c
CodecDataIn.UINT = ReadCodecData(); // get input data samples

// Work with Left ADC sample
x_buffer[0] = 0.25 * CodecDataIn.Channel[LEFT];
// Use the next line to noise test the filter
// x_buffer[0] = 0.125*((short) rand_int()); // scale input by 1/8

// Filtering using a 32-bit accumulator 'result'
for(i=0; i < N_FIR; i++)
{
    result += x_buffer[i] * h[i];
}
// Update filter history
for(i=N_FIR-1; i > 0; i--)
{
    x_buffer[i] = x_buffer[i-1];
}

// Return 16-bit sample to DAC
CodecDataOut.Channel[LEFT] = (short) result;
// Copy Right input directly to Right output with no filtering
CodecDataOut.Channel[RIGHT] = CodecDataIn.Channel[RIGHT];
/* end your code here */
WriteCodecData(CodecDataOut.UINT); // send output data to port

a) Using the network analyzer obtain the analog frequency response of your filter design and compare it with your theoretical expectations from FDA tool. Check the filter gain at the passband and stopband critical frequencies to see how well they match the theoretical design results.

b) Measure the time spent in the ISR when running the FIR filter of part (a) when using -o3 optimization. Recall your experiences with the digital output pin in Lab 3. Note that digital I/O is enabled in this project. How much time do you have to spend in the ISR when sampling at 48 kHz? What is the maximum sampling rate you can operate your filter at and still meet real-time operation?

c) Suppose the total time spent in the ISR, call it \( T_{\text{ISR}} \), is composed of a fixed time component \( T_{\text{fixed}} \) plus the time required to run the actual DSP algorithm, call it \( T_{\text{alg}} \). Experimentally measure \( T_{\text{fixed}} \) by bypassing the two for loops of the FIR filter routine. Then infer \( T_{\text{alg}} \) from the two ISR time measurement you have made. Determine the maximum number of FIR coefficients the present algorithm can support with \( f_s = 48 \text{ kHz} \) and still meet real-time. Show your work. You can assume that \( T_{\text{alg}} \) grows linearly with the number of coefficients, \( N_{\text{FIR}} \).
Measuring Frequency Response Using White Noise Excitation

3. Rather than using the network analyzer as in Problems 1–2, this time you will use the PC digital audio system to capture a finite record of DAC output as shown in Figure 10. Use the same FIR filter as used in Problem 2. The software capture tool that is useful here is the shareware program *GoldWave* (*goldwave.zip* on Web site). A 30 second capture at 44.1 kHz seems to work well. Since the sampling rate is only 32 ksps.

To get this setup you first need to add a function to your code so that you can digitally generate a noise source as the input to your filtering algorithm. Add the following uniform random number generator function to the ISR code module:

```c
//White noise generator for filter noise testing
long int rand_int(void)
{
    static long int a = 100001;
    a = (a*125) % 2796203;
    return a;
}
```

Note the code is already in place in the project you extract from the Lab 4 zip file. Now you will drive your filter algorithm with white noise generated via the function `rand_int()`. In your filter code you will replace the read from the audio code with something like the following:

```c
//Work with Left ADC sample
//x_buffer[0] = 0.25 * CodecDataIn.Channel[ LEFT];
//Use the next line to noise test the filter
x_buffer[0] = 0.125*((short) rand_int()); //scale input by 1/8
```

Once a record is captured in GoldWave it can be saved as a *.wav* file. The *.wav* file can then be loaded into MATLAB using the `wavread()` function. MATLAB has the ability to import data files directly from the *Import Data...* option under the *file* menu. Note, GoldWave can directly display waveforms and their corresponding spectra, but higher quality spectral analysis can be performed using the MATLAB signal processing toolbox. The spectral analysis function to be used in MATLAB is `psd()`, which implements Welch’s method of averaged periodograms. Suppose that the *.wav* file is saved as *test1.wav*, then a plot of the frequency response would be created as follows:

```matlab
>> [x,Fs] = wavread('test1.wav'); %Get x and sampling rate
>> % Use 2048 pt. FFT and plot with proper fs value
>> simpleSA(detrend(x),2048,48,-80,10); %detrend() removes DC offsets
>> % Rescale plot as needed and overlay other plots if desired.
```

Note the spectral estimation function `simpleSA.m`, which wraps `psd()`, can be found in the zip package. One condition to watch out for is overloading of the PC sound card line
input. Goldwave has level indicators to help with this however. Use the sound card line input found on the back of the PC Workstation. The software mixer application has a separate set of mixer gain settings just for recording. You will need to adjust the Line In volume (similar to Figure 11) and watch the actual input levels on Goldwave.

In summary, using the procedure described above, obtain a MATLAB plot of the frequency response magnitude in dB versus frequency, of the OMAP-L138 DAC output chan-
nel. Normalize the filter gain so that it is unity at its peak frequency response magnitude.

**Using a Circular Buffer**

4. Implement the circular buffer as described in text Listing 3.6 into the filter design of Problem 2. You will have to read through Section 3.4.3 of the text to get an understanding of the circular buffer concept. The basic idea is trying to eliminate the update buffer history for loop. Using the real-time IRQ timing pulse see if you achieve any speed improvement under -o3 optimization. What is maximum number of filter coefficients you can handle at 48 kHz?
Appendix: Importing an Eclipse CCS Project

This appendix describes how to import an existing eclipse CCS project, such as the one contained in the ZIP package of Lab 4, into your CCS workspace. As an example suppose you want to import the project fir_float_AIC3106.

You begin by placing the project folder in some convenient location, not inside your existing CCS workspace, say on the desktop. In CCS go to the Project menu and select Import Existing.

![Figure 12: Importing an existing CCS project.](image)

---

Appendix: Importing an Eclipse CCS Project
CCS Eclipse Project, as shown in Figure 12. From the dialog that opens select a search directory that takes you to where the project to be imported is located. CCS will discover projects in the directory where the project folder is located automatically. Be sure to choose Copy projects into workspace, and then click Finish. The project will now be listed in the CCS Project Explorer.

Figure 13: Importing an existing CCS project and making a copy in your workspace.

---

**Appendix: Importing an Eclipse CCS Project**