FIR Filter Design and Implementation

In this lab you will do the following:
Part 1. Design and implement a bandpass or low pass FIR filter on a noise signal.
Part 2. Design and implement two notch filters to remove unwanted sinusoids in a corrupted THEFORCE.

PRE-LAB

Answer the following:
1. What operating system are you using?
2. How much time does it take to complete the lab (do not include the time to write your program)?

In the pre-lab you will do the preliminary design of all the filters using Matlab SPTOOL.

Part I

For part I, you need to design either a bandpass or low pass filter. This filter will then be applied to an incoming noise signal and the performance evaluated. If you design the bandpass filter, it should have a center frequency of 1kHz with a passband of 200Hz. In other words, design for a lower cutoff of 900 Hz and an upper cutoff of 1100Hz. The lowpass filter should be a 1kHz filter.

Using SPTOOL for filter design:

Go into Matlab and at the prompt, type sptool. A new window should open.

Under filters, hit New. Since our CODEC samples at 8kHz, set the frequency to 8000Hz.
Select an algorithm.
Uncheck minimum order.
Set the type of filter you are designing.
Set the frequency parameters for the given filter type.
The parameters you play with in the design are the Order and the Beta or the weights. The order is the number of filter coefficients less 1. We discussed Beta in class. This parameter is specific for the Kaiser window. The weights are the importance of minimizing ripple in the passband and the stopband.
The system can produce various error messages and reset the frequency and beta parameters. If that happens, plug the parameters in again and try it again until it no longer changes them. To avoid the error messages, you may have to play around with the order and the beta. I have gotten an error message before which went away when I increased the order. Then I started decreasing the order to the original value, and it worked without an error message. So the system is sensitive. Also, sometimes if you hit okay to acknowledge the error message, you can proceed by hitting Apply without problems. You may need to click within the gray area of the graph window before you hit Apply.
I suggest starting out with an order above 50 and a beta of 2 or 4 if you are using the Kaiser window. Changing beta by just 1 can make a significant difference.
Once you have the filter which you like, go back to the main sptool gui and highlight your filter. Underneath select View. This will open Filter Viewer. Using the Marker menu with Tracker selected, you can make various measurements. Be sure that the scale is set for dB.

Answer the following:
3. What algorithm did you use?
4. What is the order of your filter?
5. What is the 3db BW of your designed filter?
6. What is the stopband attenuation in db for the first side-lobe? If the peak of your filter is not at 0db (ex. -5db), you will have to make adjustments to report the right attenuation. Also, what is the frequency/ies associated with the stopband?

Exporting the filter:

Once you have a filter you are satisfied with, you need to export the data to the Matlab workspace. File > Export. An export window comes up. Click on your filter, filt1 for example. Click Export to Workspace.

In Matlab, the coefficients are in the workspace as filt1.tf.num or as (name of your filter).tf.num. Save the coefficients with a comma after each coefficient in a file entitled something like bp1000.dat. Hint: use a command like fprintf(fid,'%f, ',filt1.tf.num);

Modifying the coefficient file:

Open the *.dat file using Microsoft Word. It should be changed to look like the following:

#define N 51 (51 is an example. It is the number of filter coefficients. Order + 1 from sptool).

float h[N]={0.0 -0.1 0.2 … the filter coefficients .. };

Be sure to hit return after the semi-colon at the end. Then save this file as something like bp1000.cof, but do not use bp1000.dat since you will load this file back into Matlab later.

Part II

THEFORCE was corrupted by adding two sinusoids to it. We want to remove these sinusoids in order to approximately recover the original signal.

Load the corruptvoice.wav file into Goldwave. Save it as a *.mat file and then load it into Matlab. Import the signal into sptool and get the PSD of this signal.

Answer the following:
7. What were the frequencies of the two sinusoids which were added to the original signal?

Now design two FIR notch (bandstop) filters to remove the unwanted sinusoids. The order does NOT have to be the same for the two filters. Export the filters to the Matlab workspace.

Answer the following:
8. What is the order and BW of the two filters?

In this lab, we are going to input the coefficients as short integers. In order to do that, we need to convert the coefficients from floating point. At the Matlab prompt, type: cof =

round(filtlf.tf.num*2^15); in order to make the conversion for the bandstop filter centered at the lower frequency. Filtlf is just the name of the filter which I exported to Matlab. Yours may be different. Repeat for the other filter.

Save the coefficients to two different files. Edit the two files with Microsoft Word such that they look like the following:

#define N1 81 (81 should be replaced by the number of coefficients in your filter)
(N1 is the number of coefficients associated with the filter at the lower frequency. For the higher frequency filter, replace N1 with N2).

short hlf[N1] = {3, -10, 20 … }; (hit the return key after the semi-colon)
(for the high frequency coefficients replace hlf[N1] with hhf[N2])

Save these edited files.

I. FILTER APPLIED TO NOISE

Setting up the project:

Create a folder entitled firbuf. All of the files needed for processing interrupts (refer to notes or previous lab or below) should be placed into this folder along with the file entitled Firbuf.c and your filter coefficient file.

We will generate noise using Goldwave and send that to the DSK which will process the noise using our filter. Initially, the output will be sent to the speakers.

Open Goldwave and a new sound file with a frequency of 8kHz. Create a sound using the expression evaluator under tools. Use the expression: 0.5-rand(1). This approximates white noise. The magnitude of the FFT or the spectrum of white noise is a constant. You can do a check of this. Go to Effects > Filter > Noise Reduction or Effects > Filter > Paramter EQ. The spectrum should be relatively flat. In order to do analysis in Matlab, we will save this noise in a *.mat file. File > Save As. Save the file in the firbuf folder as a Matlab file entitled noise. When you are asked if you want to update the sound window with the new format, choose No.

Set up the Device Controls window for loop play.

Connect the SOUND CARD to the speakers and listen to the output.

Now connect the output of the sound card to the input of the DSK and the output of the DSK to the speakers.

Build and Create the Project:

The following files are the interrupt ***support*** files:

C6xdsk.cmd, c6x11dsk.h, C6xdsk.h, c6xdskinit.c, c6xdskinit.h, c6xinterrupts.h, vectors_11.asm

You will need to create a project and add the following files:

Firbuf.c and the relevant support files (*.h files are not added – Scan All Dependencies instead)

Firbuf.c implements your filter using the filter coefficients found in your cof file. It creates an input buffer, in_buffer, which saves the measured input and an output buffer, yn_buffer, which saves the signal sent out. It also has some statements which have been initially commented out which save the in_buffer to a file.

Open the Firbuf.c file. There is an include line for the file containing the filter coefficients. Change the name of the file to your file name.

Now build the project. You will get a warning which you can ignore. (You must incrementally build the project anytime you change the name of the file to be included, i.e. when you want to use the low pass filter.)

Quick Check and Analysis

Make sure Goldwave is running the noise input. Run the project. You should hear a difference. Let's do a quick look at the signal we are sending out by looking at a graph of yn_buffer. We will look at the FFT Magnitude of the signal. The acquisition buffer size we will use is 1024. Set the FFT framesize to 1024 as well. Set the FFT order, DSP data type, and the sampling rate. You can update the graph display by right-clicking in the graph, going to properties, and hitting OK. You can also change between a linear and a log scale (resembles decibels since db=20*log(x)) by right-clicking and selecting Log Scale.

Optional: You can look at the actual output being sent by opening up a graph window and choosing Single Time.

Optional: You can look at the input after sampling by the DSK by doing as you did for yn_buffer but with in_buffer.

Optional: You can look at the filter coefficients by looking at a graph in Single Time of h. The FFT magnitude of h is the frequency spectrum of your filter, your design.

Answer the following:
9. Is the FFT magnitude of the output as you expected? Why? In order to answer this question properly, you need to consider the FFT or spectrum of white noise.
10. What did you set the following to when looking at the FFT magnitude of yn_buffer: FFT order, DSP data type, and sampling rate?

Detailed Analysis

For the detailed analysis, we will send the output of the DSK to another Goldwave. Running two Goldwaves can sometimes introduce some additional noise. But this should work fine for our purposes.

Connect the output to the microphone input of the sound card.

Calibrating Goldwave

We need to make sure that the volume for the microphone input is not set so high that we are saturating the sound card input.

Open another Goldwave. Open New with a frequency of 8000Hz. Open the Device Controls window. Right click and go to Properties. Choose Record. Tick the following: Monitor, Ctrl Key Safety. Monitor allows you to watch the input from the microphone without actually recording. Ctrl Key Safety means that in order to record, you have to hit the Ctrl key. Go to the Graph tab and choose Colour Amplitude for the graph display. This display will be in the bottom of the Device Controls window. Go to the Volume tab. Move the Volume window so that you can see the Device Controls window.

Now make sure the other Goldwave is sending the noise input to the DSK. Go to CCS and Debug > Restart. Run. Now go to the Goldwave we are using for output from the DSK. Adjust the volume for the microphone. You will see the amplitude vary in the amplitude graph in the Device Controls window. Adjust the volume so that the amplitude does not get clipped by the sound card, i.e. that it does not reach the +/-1 boundaries.

Generating Data Files for Analysis

We are going to generate a number of additional files for use with matlab for analysis.

One file we want is the output of the filter, specifically out of the DSK. We will acquire this through Goldwave. Make sure the input to the DSK board is being sent. Go to CCS, Debug > Restart, and Run. Now go to the Goldwave which is being used for the output. Hit Ctrl-Record in the Device Controls window. Once you have at least 15 seconds of data, stop recording. Halt the project in CCS.

Let's select 15 seconds of data and save it to a mat file. Edit > Marker > Set. Under Finish marker, set the time to 15.000. OK. Edit > Trim. Now 15 seconds of data fills the window. File > Save As. Save the file with name out in Matlab format. When asked if you want to update the sound window, choose No.

We will also want the actual input to the filter, i.e. after the codec digitizes the input signal. To get this file, we will modify the Firbuf.c file. Go to CCS and open this file. Comment out the line: inbufcnt=0;. Remove the comments from the section

```
//          {
//       inbufcnt=0;
.
to
.
//            puts("done");
//          }
```

Incrementally build the project and load the program.

Make sure the noise input is being sent to the DSK. There is no need to worry about recording the output for this. Run. As soon as the word *done* appears in the output, Halt the program. The *done* tells us that 10000 data points from the codec were saved into the buffer and written to the file *infilt.dat* which is probably in your Debug folder. We do not want the

program to reopen the file and start writing to the file again as indicated by *start*, so it is important to halt the process as soon as the *done* appears.

Matlab Analysis

We are ready to do analysis now. We have four files:

noise.mat – the noise input to the DSK
infilt.dat – the noise signal after the codec before being processed by the filter
bp1000.dat (or your file name) – the filter coefficients
out.mat – the output of the DSK

Go to Matlab.
Load noise.mat. Save wavedata as noise, i.e. noise = wavedata;.
Clear wavedata.
Load infilt.dat from possibly the Debug folder.
Load out.mat. Save wavedata as out, i.e. out=wavedata;.

We will do some processing of data using sptool. So open sptool.
Let's import our data.
File > Import. Choose From Workspace. Import as Signal. Select samplingrate and arrow pointing to Sampling Frequency. Select noise and arrow pointing to Data. Type noise for the Name. OK.
File > Import. Choose From Workspace. Import as Signal. Select samplingrate and arrow pointing to Sampling Frequency. Select infilt and arrow pointing to Data. Type infilt for the Name. OK.
File > Import. Choose From Workspace. Import as **Filter. Form is Transfer Function. Select samplingrate and arrow pointing to Sampling Frequency. Select bp1000 and arrow pointing to Numerator. Type in 1 for the Denominator. Type bp1000 for the Name. OK.
File > Import. Choose From Workspace. Import as Signal. Select samplingrate and arrow pointing to Sampling Frequency. Select out and arrow pointing to Data. Type out for the Name. OK.

Now we will generate a number of spectrums in order to make comparisons.

Highlight noise. Hit create under the Spectra section. Check in the upper corner that Signal is noise. Options > Magnitude Scale > Linear. With the following settings Welch, 1024, 256, hanning, 0, hit Apply. This is the power spectral density of the input noise. Look at this in decibels by Options > Magnitude Scale > Decibels. Close the window. Edit > Name > spect1. Set the name to noisesp.

Repeat for the infilt signal. Set the name to infiltsp.

Answer the following:
11. Did you notice a difference between the PSD of the noise and infilt? There is an anti-aliasing filter in the codec. Can you provide an estimate of the cutoff frequency of the filter?

Repeat for the out signal. Set the name to outsp.

Now we will provide some estimates of what our PSDs should have been based on our filter.

Highlight noise in the signal section. Highlight bp1000 in the filter section. Hit Apply in the Filter section. Leave algorithm alone. For output signal, type noiseout. OK.
Highlight noiseout and hit Create in the Spectra section. Do as before and change the name to noiseoutsp.

Repeat but this time use infilt as the input to the filter. The final name should be infiltoutsp.

Now it is time to export the data and make some plots in Matlab.

File > Export. Click noisesp. Export to Workspace.
Repeat for infiltsp, outsp, noiseoutsp, and infiltoutsp.

Go to the Matlab workspace. The frequency vector is the same for all of the signals. Type the following at the prompt: f = noisesp.f;.
The sptool program uses an FFT algorithm to create the PSDs. This produces an effect called wrap around which will be confusing for us. By looking at the frequencies of 0 to 4000 (the Nyquist frequency), we will avoid this problem and have understandable graphs. Let us find the indices for the frequencies from 0 to 4000. At the prompt, type: i = find(f = = 0). Then type: i = find(f < 4000). The last number is the index we need. For me, the numbers were 513 and 1025. Let's truncate f to these indices, i.e. type: f = f(513:1025);.

Our power spectral density is in linear form and is not normalized. Also, we must truncate to the frequencies of interest. To do this, type the following for each signal:
nsp = log (noisesp.P(513:1025))/max(log(noisesp.P));
Let the arrays of interest be: nsp, insp, osp, nosp, and inosp.

Answer the following:
12. Provide a plot of nsp and insp as a function of frequency. Note the differences.
13. Provide a plot of osp, nosp, and inosp as a function of frequency. Osp represents our actual output signal, after the codec. Nosp represents our expected output (to the codec – in a perfect world of perfect DSPs, codecs and sound cards, this is what we would expect Goldwave to record) based on our noise signal provided to the DSK. Inosp represents our expected output (to the codec) based on our digitized input noise signal. Note differences and make comments.

II. REMOVING UNWANTED SINUSOIDS

Create a folder entitled Notch. All of the files needed for processing interrupts should be placed into this folder along with the file entitled Notch2.c, Notch2.gel, and your two coefficient files.

Create the project.

Open the Notch2.c file. There are two include lines for the files containing the filter coefficients. Replace bslf.cof with the name of the file containing the coefficients for your low frequency notch filter. Replace bslf.cof with that for your high frequency notch filter.

Build the project. Load the GEL file. It provides two filter states. State 1 causes the filters to be bypassed so that the input to the DSK is sent directly to the output. In this state, you

can listen to the corrupted voice. If you switch to state 2, then the corrupted voice will be filtered before being sent to the output.

Load the corruptvoice.wav file into Goldwave. Connect the output of the sound card to the input of the DSK. Connect the output of the DSK to the speakers.

Run Goldwave in loop mode.

Run your project in state 1 and listen to the corrupted voice. Switch to state 2 and listen to the filtered corrupted voice.

Let's save the output of the filter to a file.
Edit the Notch2.c file. Comment out the j = 0; line after the line 'if (j == Bufsize - 1)' and remove the comments associated with printing output to a file named out.dat. Incrementally build and load the program again. Run the program while in state 2. Halt the program as soon as the *done* appears.

Load the out.dat file from the Debug folder into Matlab. Get the spectra of this output signal. As we did before, normalize and truncate outsp along with corruptsp.

Answer the following:
14. Provide a plot of the normalized spectra of the corrupted input signal and the filtered output signal as a function of frequency. Note the differences and make comments.