EE477 Digital Signal Processing
Laboratory Exercise #13
Real time FIR filtering

Spring 2004

The object of this lab is to implement a C language FIR filter on the SHARC evaluation board. We will filter a signal from the computer output and from a random noise generator. The filtered signal will be measured to determine the filter's frequency response.

Board Set Up

SHARC evaluation board on gray PVC box. **Note:** The signal flow chart for the board is on the class web site.

1. Plug in the box and turn it on. (The LED's for Flags 2 & 3 should start blinking).
2. Plug the audio output ("audio *to* computer") miniplug into the line-level audio input jack on the computer sound card (blue colored jack). Double-click the "speaker" icon in the Windows tray and select the *line-in* source using the recording panel, and unmute *line-in* using the playback panel. Adjust the speaker volume and you should hear a pizzicato rendition of Mancini's "Theme from Peter Gunn."
3. Plug the serial cable between the SHARC evaluation board and the computer's serial port #1 (com1). Make sure the port switch box is set to the proper cable (A or B).
4. Launch the VisualDSP++ program. If the software is configured to use the evaluation board, a window will pop up saying "Hit the reset button" for the evaluation board. If the window does not pop up, go into the Sessions menu and switch to the session (or create a new session) for the **EZ-KIT 21061** and the **21061** hardware.

MATLAB Setup:

1. Open Matlab
2. Run the data acquisition demo: `demoai_fft`.
3. You should see a display of the waveform as well as the spectrum.

Programming Setup:

1. Create the folder `c:\EEclasses\EE477\LAB13`
2. Copy the files `lab13.c`, `lab13.ldf`, and `061__ezkit_hdr.asm` from the course web site http://www.coe.montana.edu/ee/rmaher/ee477/notes.htm to this directory.
3. Create a new VisualDSP++ project called `lab13`. The project file should also be kept in your `LAB13` directory.

4. Under Project->options select:
    a. Processor : ADSP-21061
    b. Type : DSP executable
    c. When it asks to add support for the VisualDSP++ kernel select NO.
    d. Add source file : `061__ezkit_hdr.asm`
    e. Add source file : `lab13.c`
    f. Add linker file: `lab13.ldf`
    g. You will also need to add a header file named `fir.h` that will be created below.

5. You should already have a session for the evaluation board. If not, create a new session: Session->New session->EZ-KIT 21061.

Examine the C program file `lab13.c` . The `main()` routine is at the end of the file, around line 291. It calls several initialization routines and then goes into an idle (do nothing) loop.

The processing happens in an interrupt routine, `spr0_asserted()`, around line number 125. This routine is called once for each stereo input sample pair. Note that the original function just copies the left input sample to both the left and right output samples. You will later need to insert instructions to compute the FIR filter.

Build and run the pass-through program:

From the VisualDSP workspace, select "rebuild all" to compile and link the programs. The system will automatically download the executable code to the EVB and run to a breakpoint in the `main()` function. Now press the "run" button. The second LED should start flashing and VDSP++ should flash "running" in the status bar.

Adjust the switches so that the noise generator is connected to the DSP input and the board output is sent to the computer. Do you hear the sound?

Now launch Matlab and run the display program `demoai_fft` . The Matlab window should show the input signal and the spectrum. Make note of the signal amplitude.

Observe the Matlab display for both white noise and pink noise. How do they differ? Also observe the DSP input and the DSP output (output switch). Can you see the effects of the sampling and reconstruction processes?

To stop the program use the VDSP++ software reset button. *Never press the hardware reset button, unless the software instructs you to press it.* The reset process takes about 15 seconds, after which you will get control of VDSP++ again.

Edit the `lab13.c` program so that the output is caused to be 0.1 times the input. All you need to do is edit the line that copies the filter input to the filter output. Save and rebuild the program, run it, and observe the output. Is it $1/10^{th}$ as big as before?
[*Instructor Verification*]

Finally, re-edit the program so that it just passes the signal again. Soft reset, build, and run the program to verify that it is once again just passing the signal unaltered.

Filter Design:

Using Matlab, design a bandpass filter using the `fir1()` function from the signal processing toolbox. Design a 10-tap bandpass filter centered at 10kHz (the hardware sampling frequency is set to 48kHz). Note what order `fir1` requires in order to get 10 taps. Choose your own cut-off or "edge" frequencies and record them for your lab report.

Use `[H,W]=freqz(…)` to generate a linear plot of magnitude vs. frequency for the filter you designed. You will need to include this theoretical plot (with labels) in your report, as described below.

You will need to copy the filter coefficients produced by Matlab into a file named `fir.h`, where the coefficients form a single column and are comma delimited. These coefficients will then be used in your C program (see the `#include fir.h` line). You can dump the coefficients from Matlab by using the following Matlab code:

```
fid = fopen('c:\eeclasses\ee477\lab13\fir.h','wt'); %'wt' means write text-format file
fprintf(fid,'%0.18f,\n',b);                         % vector b contains the filter coefs
fclose(fid);                                               %close the file
```

You could also cut-and-paste from Matlab and a text editor using:

```
fprintf(1,'%0.18f,\n',b);            % fid=1 means standard output (the screen)
```

Programming:

1. Make sure header file `fir.h` is identified in the project file group.
2. If necessary, edit the line near the beginning of `lab13.c` to change `NUM_TAPS` to the number of coefficients you actually wrote to `fir.h`. Also make sure that the path to `fir.h` is correct in the `#include` section.

Task 1:

Edit the program to *filter* the input samples using the coefficients you created in Matlab and stored in `fir.h`. You need to write your own FIR filter instructions in `spr0_asserted()`. Use the arrays `coeffs[]` and `state[]` for the coefficients and the filter delay line, respectively. Keep in mind that the filter routine is called to process one sample at a time, so you need to adjust the `state[]` array so that the delay line is ready for the next call.

Note that you might be well advised to test your FIR code first with a single coefficient set to 1.0 and all the other coefficients set to zero. This should implement a “pass through” filter.

When ready, use the bandpass filter coefficients in your program. Soft reset, build, and run your FIR program. Does the spectrum displayed by `demoai_fft` resemble your filter?

Task 2:

Measure the filter frequency response using a sequence of sinusoidal inputs. You can attach an external signal generator and observe the output using an oscilloscope. Take and tabulate sufficient measurements to cover the frequency range of interest—especially in the transition bands around the bandpass region.

You should enter these measured data points via Matlab and plot the results on top of the theoretical filter response you obtained via `freqz`. Consider any discrepancies and unexpected behavior.

> **Note:** Instead of the external function generator and scope, you can automate the data collection process by using the data acquisition toolbox in Matlab to generate and record signals. Connect the "signal *from* computer" cable to the soundcard output (green) jack, or to the headphone jack on the front of the computer. Modify the `DAQ.m` file to automatically generate tones, sending these tones through the DSP and recording the DSP output. You can then have Matlab compare the input and output signals automatically as you step through frequencies. Give it a try!

Task 3:

Increasing the length of the filter would allow a more selective frequency response, but at some point the amount of time required to compute the filter will exceed the intersample interval. In other words, if the filter is too long, the DSP can't finish computing the current result before the next input sample arrives, and the processor starts missing samples. Investigate what number of `NUM_TAPS` causes the filter to malfunction. How did you tell? Use `fir1` to design a maximum length filter and verify its behavior.

Lab write up:

Include your FIR filter code and a figure showing the magnitude response of the FIR filter, both theoretical and measured. You will need to normalize the data points that you took so they will fit on the plot. How well do your points match the filter you designed? Explain. Comment on Tasks 1& 3 as well, including all pertinent results and observations.

EE477
Digital Signal Processing
Spring 2004

Instructor Verification
Lab #13

Demonstrate “pass” program with 0.1 gain factor _____