EE 445S Real Time Digital Signal Processing Laboratory

Prof. Brian L. Evans

Lab 3 Handout

a) This lab deals with designing and implementing digital filters. The design will be done using Matlab (fdatool) and implementation will be done in the DSP.

b) We will be designing both FIR (Finite Impulse Response) as well as IIR (Infinite impulse response) filters.

**WEEK 1:**

**Task 1: FIR filter design**

* Design a Remez exchange 25th order band – pass filter. fs = sampling frequency = 48kHz. flow = low cut off frequency = 9.6kHz. fhigh = 12kHz. Use fdatool in Matlab. This is a FIR filter.
* Obtain and store the filter coefficients.
* We will be plotting a comparison of the theoretical and experimental magnitude response for the lab report.
* Obtain the gain in dB for the theoretical plot from Matlab at different frequencies. (Say, 1, 2, 3, .., 12 kHz)

**Task 2: DSK Magnitude Response**

* The project files are the same as used in task 3 (following).
* Set the sampling frequency to 48kHz on the audio codec.
* Measure the amplitude response of the DSK board by applying a sine wave from the signal generator to the line input and loop samples internally in the DSP back to the line output. This means you will read both the channels and write back to both the channels. You can then measure the voltage of the output and divide it by the voltage of the signal you sent in. This is the gain. You can convert it to decibels (dB).
* Obtain the gain in dB at different frequencies. (Say, 1, 2, 3, .., 12 kHz)

**Task 3: FIR filter implementation**

* Write a circular convolution function. The function will be called convolve. The following will be the function prototype to be used. Please do not change this as we will be comparing this C function with the assembly convolve function (This is already available for use).

Prototype: float **convolve** (float \*x, const float \*h, const unsigned hlen, const unsigned xlen, const unsigned newest)

Here x is the circular buffer, h is the filter coefficient array, hlen is the length of the filter coefficient array (in words) = number of taps = filter order + 1, xlen is the length of the circular buffer array (in words), which is a power of 2, newest is the newest index. (A handy tip: When you multiply x with h, multiply h by 1000 and typecast the result to int16\_t prior to multiplication)

1. Create a new folder called “Lab\_3” under “Lab\_445S”. Create a folder called “Lab\_3\_Workspace” under “Lab\_3”. In this workspace, create a new project called “FIR\_Filter” using the same files provided in Lab 1, and replace the main.c and main.h file with the one provided in the folder “Lab\_3\_FIR” present in “Lab\_3\_DSP\_Files.zip”. Modify israudio.c. Take the general program flow from Slides 3-7 and 3-8.
   * The code should initialize the filter coefficients.
   * The code should read from the signal generator, update the circular buffer (using only the left channel data) and update the “newest” index. (Use readsample and writesample functions)
   * The code should make a function call to the convolve function you wrote earlier.
   * The code should output the result from the convolve function on the right channel and output the unfiltered input to the left channel. This way you can see both the input and output on the oscilloscope.
2. In other words, all you need to modify is the isr file. We use interrupt, so in the isrAudio function, you need to implement your code like:

*readsample;*

*store the sample in circular buffer;*

*do convolution and get output result*

*writesample(the output)*

And to make the code clearer, convolution should be implemented in a separate subfunction.

1. To test the functionality of your code, feed different frequencies from the function generator to both the channels of your Line In. Feed the stereo output coming from the Line Out to the oscilloscope. Use the same frequencies from Task 1. Obtain Vout/Vin and convert it into decibels (dB). This is the second part of point #1 of lab report.
2. Now, plot both the graphs on top of each other (Task 1 and Task 2). This compares the theoretical and the experimental magnitude plots.

**Task 4: Code Optimization**

1. Create a new folder called “Lab\_3” under C:\Lab\_445S\. Copy the contents of the folder “Lab\_3\_Block\_FIR” present in “Lab\_3\_DSP\_Files.zip” downloaded from the website into C:\Lab\_445S\Lab\_3\Lab\_3\_Block\_FIR\.
2. Open Code Composer Studio. Select the workspace as: C:\Lab\_445S\Lab\_3\Lab\_3\_Workspace. Create a new project in Code Composer called “fir\_block”. Follow steps mentioned in Lab 1 handout to create a project till step 7. In step 8, browse to C:\Lab\_445S\Lab\_3\Lab\_3\_Block\_FIR and add all files apart from the “bsl”, “gel”, “tests”, and “dsplib64x+” folder.
3. Right click on the project -> Build properties -> Include options. Add the path C:\Lab\_445S\Lab\_3\Lab\_3\_Block\_FIR\dsplib64x+\include.
4. Right click on the project -> Link files -> Browse to C:\Lab\_445S\Lab\_3\Lab\_3\_Block\_FIR\dsplib64x+\lib and add “dsplib64plus.lib”.
5. Turn on Pin 8 of the Dip switches to the right (S2).
6. Right click on the project -> Active build configuration -> Debug. Run the program. Go to Tools -> RTA -> Statistics Data. Look at the variable labeled “benchmark”. Look at the average value. Note this down. This gives the number of machine cycles required to run the cfir filter.
7. Right click on the project -> Active build configuration -> Change to Release. Right click on the project -> Build properties -> Basic options -> Choose the optimization level to 2.Note down the average value of “benchmark”.
8. Right click on the project -> Build properties -> Basic options -> Change the optimization level to 3.. Note down the average value of “benchmark”.
9. Open the file “fir.c”. Look at line 55, 56. Comment out these two calls to cfir and uncomment the lines 57, 58. This calls the library function for FIR filter which has been coded using assembly. Repeat for the lines 81, 82, 83, and 84.
10. Repeat steps 6 to 8. Store the machine cycles in a table.

**Task 5: IIR filter design**

* We will now move on to the IIR filter design and implementation.
* We will implement the same IIR filter using 2 different architectures: Type 1 Direct Form (DFI) and Type 2 Direct Form (DFII).
* First, design a 6th order IIR Butterworth band – pass filter using fdatool.
* fs = sampling frequency = 8kHz. flow = low cut off frequency = 1kHz. fhigh = 2.5kHz
* Obtain and store the filter coefficients.
* We will be plotting a comparison of the theoretical and experimental magnitude responses (both DFI and DFII) for the lab report.
* Obtain the gain in dB for the theoretical plot from Matlab at these following frequencies:

400 Hz, 500 Hz, 600 Hz, 700 Hz, 1 kHz, 1.5 kHz, 2 kHz, 2.5 kHz, 3 kHz, 3.5 kHz and 4 kHz.

**Task 6: IIR filter DFI implementation**

* Modify isr.c like you did for the FIR filter.
* We will break up the entire filter into a cascade of **biquads** (2nd order sections). In our case, we will have three biquads in series.
* Write a generic biquad “convolution” function. This takes **one** biquad’s filter coefficients and the input and computes the output for that particular biquad. Remember to scale (gain) your biquad output appropriately.
* The generic flow of the program will be similar to the FIR.
  + The code should initialize the filter coefficients for **each** biquad.
  + The code should read from both the left channel and right channel.
  + The code should make a function call to a biquad convolve function. This will be done in a cascade (in a series fashion) for three biquads.
  + The code should output the result from the last convolve function on the right channel and output the unfiltered input to the left channel. This way you can see both the input and output on the oscilloscope.
* To test the functionality of your code, feed different frequencies from the function generator to both the channels of your Line In. Feed the stereo output coming from the Line Out to the oscilloscope. Use the same frequencies from Task 3. Obtain Vout/Vin and convert it into decibels (dB).
* Though it is not mandatory to implement the coefficients and states of the biquad this way, it is **strongly recommended** to use a structure which stores the biquad coefficients and states as its data members. This will help passing of parameters back and forth function and make it easier to debug your code.

**Task 7: Profile the DFI biquad convolution**

* Here, you will profile the DFI biquad convolution routine you just wrote for Debug, Release (both for optimization level 2 and 3). Insert breakpoints just before and after your biquad filter function and count the number of machine cycles required. Store the result in a tabular form.

**Task 8: IIR filter DFII implementation**

* Repeat Task 6 for DFII implementation.
* Only the biquad “convolution” would change from what you had for Task 6. The rest of the code would be the same.
* To test the functionality of your code, feed different frequencies from the function generator to both the channels of your Line In. Feed the stereo output coming from the Line Out to the oscilloscope. Use the same frequencies from Task 3. Obtain Vout/Vin and convert it into decibels (dB).
* Though it is not mandatory to implement the coefficients and states of the biquad this way, it is **strongly recommended** to use a structure which stores the biquad coefficients and states as its data members. This will help passing of parameters back and forth function and make it easier to debug your code.

**Task 9: Profile the DFII biquad convolution**

* Here, you will profile the DFII biquad convolution routine you just wrote forDebug, Release (both for optimization level 2 and 3). Present the data in the following format:

|  |  |  |
| --- | --- | --- |
| **Optimization level** | **Biquad conv, DFI** | **Biquad conv, DFII** |
| **Debug** |  |  |
| **Release (Optimization Level 2)** |  |  |
| **Release (Optimization Level 3)** |  |  |