Utah State University
Real-Time Digital Signal Processing Laboratory
ECE 5640

Lab 2: FIR Filtering

Objective

The purpose of this lab is to compare the implementation of a finite impulse-response (FIR) filter using different structures.

References

1. *Digital Signal Processing Principles, Algorithms, and Applications* by Proakis and Manolakis, Section 9.2.
2. *OMAP-L138 Datasheet*, Texas Instruments.
3. *OMAP L138 Technical Reference Manual*, Texas Instruments.
4. *TMS320C674x DSP CPU and Instruction Set Reference Guide*, Texas Instruments.
5. On-line documentation for *Code Composer Studio*.

Required Equipment

The following equipment is necessary to complete the lab:

1. The Texas Instruments OMAP-L138 Experimenter Kit.
2. An Analog Discovery soft instrumentation board.
3. Cables for connecting the system to the Analog Discovery board.

Required Software

The following software is necessary to complete the lab:

1. The Texas Instruments *Code Composer Studio*
2. The OMAP-L138 Experimenter board support library (BSL): `evmomapl138_bsl.lib`.
3. The `.zip` archive file `lab-files.zip`.
4. The float filter weight file `firfilt32.dat`.

Background

FIR filtering can be performed using various filtering structures. Among these are the direct form I and lattice structures. Each has advantages and disadvantages when considering number of arithmetic operations, memory requirements, and round-off error.

(Remember to include all of your C code in the write-up.)

Procedure

1. A 32 weight FIR filter using floating-point weights is given in the file:

   `firfilt32.dat`

   The sampling frequency is 48 kHz. Check the frequency response of the filter using `Matlab` and graph it in dB using `Matlab` or another graphing program and print out for future reference.

2. Write a C program that implements a direct form I FIR filter. Your program should be able to filter with the filter given in procedure 1. For example, the following code could be included in the file:

```
.
.
.
/* FIR filter coefficients */
float coeffs[] = {coeff1, coeff2, ... , coeff32};
.
.
.
main()
{
.
.
.
```

   The files to create a project in *Code Composer Studio* (CCS) are provided in the `.zip` archive `lab-files.zip` containing `template1.c`, `template1.h`, `aic3106.c`, `mcasp.c`, and `BoardConfig.ccxml`. The file `template1.c` chould be modified to uncomment the statments which read the samples from the `line in` port.

   To make sure you optimize the computations necessary between samples, try to implement a circular buffer in software *without* using the "C" modulo operator. (Hint: try masking the indexing variable.) Do *not* unroll the loops of the filter code but optimize the "C" code enough to allow the filter to run as fast as possible without moving the input samples in the delay line each time a new sample is received.

After you have debugged for correct functionality, you may need to turn on the compiler optimizer to get your code to run fast enough. This is easy to do; just select the `release` build configuration in CCS and be sure to configure like you did in the "Initial Code Composer Studio Setup" section of Lab 1.

Save and print out the assembly code created by the compiler.

3. 3. Download and boot the FIR filter. Check the frequency response of the filter by using the white noise output of the Digilent Analog Discovery as input to the filter. Observe the spectrum of the output using the Digilent Waveforms software:

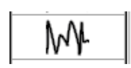
Generating White Noise

(a) Connect the 3.5mm output of the Analog Discovery to the 3.5mm line input of the OMAP kit.

(b) Open the Digilent Waveforms software.

(c) Select the arbitrary waveform generator function.

(d) Select the noise waveform option on the left side of the window.



(e) Select a frequency of at least 100kHz.

(f) Select amplitude of 500mV.

(g) Select `Run AWG 2` to begin white noise generation.

(h) White noise output can be verified by feeding the 3.5mm output back into the oscilloscope input of the Analog Discovery (Orange leads 1+ and 1-), and using the FFT display of the scope function.

Viewing the Spectrum

(a) Feed the 3.5mm output of your filter into a splitter box, then use a BNC to Alligator clip cable to connect the signal to the input of the Analog Discovery scope. Connect the red alligator clip to the 1+ input (orange) and the black alligator clip to the 1- input (orange/white) of the Analog Discovery.

(b) Open the Digilent Waveforms software.

(c) Select the scope function.

(d) Select the FFT display button near the top of the window.

(e) Right-click on in the FFT display window to adjust the horizontal axis zoom to achieve an appropriate bandwidth view.

(f) Right-click on in the FFT display window, and select averaging.

Plot the frequency response. Is it what you expected?

4. Design a lattice filter using reflection coefficients computed from the filter weights from procedure 1. The computation of the reflection coefficients can be done in a program on the host (do *not* use `Matlab` to compute the reflection coefficients) or in the 'C67 program itself. You can check that the coefficients are correct by comparing your computed values to coefficients computed by `Matlab`. Implement the lattice filter on the 'C67. (Note that a circular buffer is *not* needed for the lattice filter.) Print the frequency response.

   Remember that the derivation of the lattice filter in your book assumes the first filter coefficient, α_0, is 1. Since your filter (probably) does not meet this condition, you can add 1.0 to the beginning of the filter coefficients, and subtract the input from the output and produce a 1-sample delayed FIR output.

   Again, save and print out the assembly code created by the compiler.

Questions

1. What are the memory requirements and computational requirements of the two implementations in number of locations and number of computations?
2. How do the two frequency responses compare?
3. Which do you think is the "best" implementation. Why?
4. Look at the assembly code. Do you think you could optimize it to run more efficiently? How?