ECE 4020

Lab Part 4 Report

Ryan Colon

03.11.22

Introduction

Contained in this report is the documentation of the implementation and evaluation of a filter program created in [ii] on a dedicated DSP board. To implement this program, the MATLAB filter design code suggested in [i] was adjusted and the c program in [ii] was slightly modified. First the changes to both programs will be discussed, followed by the testing process and subsequent conclusions.

Matlab and C Program Adjustments

The MATLAB program discussed in [i] was created in such a way that the coefficients were read from a text file on the PC. However, this can’t be done on a dedicated DSP board. So instead, the code was adjusted to output the coefficients to a text file in the format of a constant signed fract array declaration in C, so it could be copied and pasted directly into the programs source code. This adjustment is shown below:

Text

Description automatically generated

Figure 1. MATLAB Filter File Output Code Snippet

In the original C program, the function “process\_samples” was used to implement the filter. The “process\_samples” function was marked for JACK to call back to every time a set of samples was ready to be processed. The same method can be used on the DSP board, but instead of a function being the callback, we create an entire source code file for the board to run whenever a set of samples is ready to be processed. This source code file was called “Process Samples.c” and is available in appendix A. For this program to run, a few short modifications had to be made: the summation result was made to be of the ‘accumulator’ data type; the math involving the delay line, coefficients, and convolution were converted to fixed point arithmetic, and a clipping feature was added to the program to manually handle over/underflow. These changes are shown below:

Text

Description automatically generated

Figure 2. Process Samples.c Code Snippet

Test Procedure

List of equipment used:

* Arbitrary Function Generator
* Oscilloscope
* Blackfin BF561 Board
* Spectrum Analyzer

The evaluation of the filter began with a simple test to check to make sure everything was operating as it should be. For this test, the function generator output was connected to the right channel of Audio In 1 of the BF561 and the right channel of the Audio Out 1 of the BF561 was hooked up to the channel 1 input of the oscilloscope. A block diagram of the connections is show on the next page:

Diagram

Description automatically generated

Figure 3. Basic Test Connection Setup

To test basic operation, a sinusoidal signal of varying frequency was provided to the blackfin and the output monitored on the oscilloscope. The filter appeared to be operating as intended, so a more thorough test involving a spectrum analyzer was done.

For this test, a network analyzer was connected through an attenuator as an input to the BF561 and the blackfin output was plugged into T1 on the spectrum analyzer. A block diagram of this configuration is shown on the next page:

Diagram

Description automatically generated

Figure 4. Network Analyzer Test Connection Diagram

The purpose of this test is to check and make sure that the filter continues to the meet the specs outlined in [i]. For this, the magnitude and group delay of the filter and passthrough were measured over the operating range of the filter (10 Hz to 24 kHz). To make it easy to visualize, the vertical divisions for the magnitude were set to 10 dB/div and the group delay range was set from 0 to 5ms. Shown below are the measurements for the filter:

A screen shot of a computer

Description automatically generated with medium confidence

Figure 5. Overall Filter Magnitude Response

A screen shot of a computer

Description automatically generated with medium confidence

Figure 6. Overall Filter Group Delay

As can be seen in all the above images, the filter has provided well over 5 divisions suppression of the stopband, thus meeting the specified criteria for that range. To check and make sure the passband meets spec, the range was adjusted to 10 Hz to 4 kHz and the dB/div were set to 1 dB/div. This measurement is shown below:

A screenshot of a computer

Description automatically generated with low confidence

Figure 7. Passband Filter Magnitude Response

A screen shot of a computer

Description automatically generated with low confidence

Figure 8. Passband Filter Group Delay

As can be seen by the magnitude response, the ripple is well within spec, and given the overall and passband group delay we can see that the filter is behaving as intended. To check for any effects resulting from aliasing, measurements of the passthrough signal were taken as well, and they are shown below:

A screen shot of a computer

Description automatically generated with low confidence

Figure 9. Overall Passthrough Magnitude Response

A picture containing chart

Description automatically generated

Figure 10. Overall Passthrough Group Delay

The passthrough behavior appears as expected, and the filter measurements show that it is operating well within spec so the goal of this experiment has been achieved.

Concluding Remarks

In this report, a filter that has already been designed in [i] was implemented on a DSP board and shown to meet specs.

Citations

1. Colon, R., 2022. *ECE 4020 Lab Part 1 Report*.
2. Colon, R., 2022. *ECE 4020 Lab Part 2 Report*.

Appendix

1. Process Samples C Program

#include "Talkthrough.h"

#include <stdfix.h>

#define NUM\_COEFF 162

//--------------------------------------------------------------------------//

// Function:    Process\_Data()                                              //

//                                                                          //

// Description: This function is called from inside the SPORT0 ISR every    //

//              time a complete audio frame has been received. The new      //

//              input samples can be found in the variables iChannel0LeftIn,//

//              iChannel0RightIn, iChannel1LeftIn and iChannel1RightIn      //

//              respectively. The processed data should be stored in        //

//              iChannel0LeftOut, iChannel0RightOut, iChannel1LeftOut,      //

//              iChannel1RightOut, iChannel2LeftOut and iChannel2RightOut   //

//              respectively.                                               //

//--------------------------------------------------------------------------//

void Process\_Data(void)

{

    /\* Old Program, merely passthrough. Holding onto it for debugging purposes.

    iChannel0LeftOut = iChannel0LeftIn;

    iChannel0RightOut = iChannel0RightIn;

    iChannel1LeftOut = iChannel1LeftIn;

    iChannel1RightOut = iChannel1RightIn;

    \*/

    //I don't know if this will compile but it's technically one line lol

    static const fract filt[NUM\_COEFF] = {0.00235455956003r, 0.00220551156530r, 0.00280095641082r, 0.00312185742445r, 0.00304308982707r, 0.00250259141645r, 0.00154822993674r, 0.00032057526390r, -0.00096053984159r, -0.00203387405835r, -0.00266654613723r, -0.00270420457038r, -0.00212937028755r, -0.00106606013151r, 0.00022879346710r, 0.00142940872042r, 0.00221076153526r, 0.00233934605887r, 0.00173604977464r, 0.00051267461054r, -0.00104555253200r, -0.00254161186331r, -0.00356483473056r, -0.00380070104505r, -0.00312186805347r, -0.00163874289972r, 0.00031306655324r, 0.00224343083100r, 0.00362940046085r, 0.00405250903572r, 0.00332186469666r, 0.00154334422047r, -0.00088459223929r, -0.00335557718266r, -0.00520300720528r, -0.00587518697477r, -0.00509450588016r, -0.00295366488212r, 0.00007837133138r, 0.00325372415844r, 0.00572390883855r, 0.00676211494130r, 0.00596876195462r, 0.00340738254642r, -0.00037481545715r, -0.00445400910506r, -0.00774785837969r, -0.00929405686331r, -0.00851991048257r, -0.00543048039086r, -0.00065723215096r, 0.00465608060108r, 0.00911559260442r, 0.01142864405910r, 0.01076070454551r, 0.00700194057542r, 0.00086647212063r, -0.00621821655046r, -0.01242104041674r, -0.01595400317452r, -0.01555243781989r, -0.01086689486446r, -0.00265805060304r, 0.00727608383057r, 0.01644740604028r, 0.02224154609225r, 0.02257822099166r, 0.01651732888375r, 0.00465700768827r, -0.01079044272069r, -0.02629403109346r, -0.03763571359643r, -0.04076339122648r, -0.03269191039328r, -0.01225575975269r, 0.01946020379678r, 0.05912605102477r, 0.10163023630168r, 0.14092768308092r, 0.17111141309793r, 0.18749662633764r, 0.18749662633764r, 0.17111141309793r, 0.14092768308092r, 0.10163023630168r, 0.05912605102477r, 0.01946020379678r, -0.01225575975269r, -0.03269191039328r, -0.04076339122648r, -0.03763571359643r, -0.02629403109346r, -0.01079044272069r, 0.00465700768827r, 0.01651732888375r, 0.02257822099166r, 0.02224154609225r, 0.01644740604028r, 0.00727608383057r, -0.00265805060304r,

    -0.01086689486446r, -0.01555243781989r, -0.01595400317452r, -0.01242104041674r, -0.00621821655046r, 0.00086647212063r, 0.00700194057542r, 0.01076070454551r, 0.01142864405910r, 0.00911559260442r, 0.00465608060108r, -0.00065723215096r, -0.00543048039086r, -0.00851991048257r, -0.00929405686331r, -0.00774785837969r, -0.00445400910506r, -0.00037481545715r, 0.00340738254642r, 0.00596876195462r, 0.00676211494130r, 0.00572390883855r, 0.00325372415844r, 0.00007837133138r, -0.00295366488212r, -0.00509450588016r, -0.00587518697477r, -0.00520300720528r, -0.00335557718266r, -0.00088459223929r, 0.00154334422047r, 0.00332186469666r, 0.00405250903572r, 0.00362940046085r, 0.00224343083100r, 0.00031306655324r, -0.00163874289972r, -0.00312186805347r, -0.00380070104505r, -0.00356483473056r, -0.00254161186331r, -0.00104555253200r, 0.00051267461054r, 0.00173604977464r, 0.00233934605887r, 0.00221076153526r, 0.00142940872042r, 0.00022879346710r, -0.00106606013151r, -0.00212937028755r, -0.00270420457038r, -0.00266654613723r, -0.00203387405835r, -0.00096053984159r, 0.00032057526390r, 0.00154822993674r, 0.00250259141645r, 0.00304308982707r, 0.00312185742445r, 0.00280095641082r, 0.00220551156530r, 0.00235455956003r};

    static fract delay[NUM\_COEFF] = {0.0r};

    accum sum;

    delay[0] = rbits((short)(iChannel0LeftIn>>16));

    //Apply the filter

    sum = 0.0k;

    int i;

    for(i = 0; i < NUM\_COEFF; i++){

        sum += delay[i]\*filt[i];

    }

    //Shift the delay line

    int j;

    for(j = NUM\_COEFF - 1; j > 0; j--){

            delay[j] = delay[j-1];

    }

    //Clip the signal

    sum = (sum > ACCUM\_MAX) ? ACCUM\_MIN : sum;

    sum = (sum < ACCUM\_MIN) ? ACCUM\_MAX : sum;

    iChannel0LeftOut = bitslr((long fract)sum);

    iChannel0RightOut = iChannel0RightIn;

}