Lab 5

Filter Design Structures

ECE 406: Real-Time Digital Signal Processing

April 7, 2015

Christopher Daffron

cdaffron@utk.edu

There was only one objective for this lab. The objective was to explore different filter design structures, including the Second Order Sections / Direct Form II structure and the FIR lattice structure. The SOS structure was implemented for both FIR and IIR filters and all of the structures were implemented using circular buffers.

Task 1

Step 1

The first part of Task 1 was to take some reference measurements on an implementation of an FIR Direct Form I filter. For this task, I used the circular buffer implementation from Lab 3 that also takes advantage of the symmetry of the coefficients to further improve the performance of the filter. I chose to use this implementation because it had the best performance in cycles of all of the FIR implementations from Lab 3, but the frequency response would be identical for any of those implementations. The FIR filter that I am using was designed using the Parks McClellan method and therefore is an equiripple filter. A plot showing both the expected and actual frequency responses is shown below.

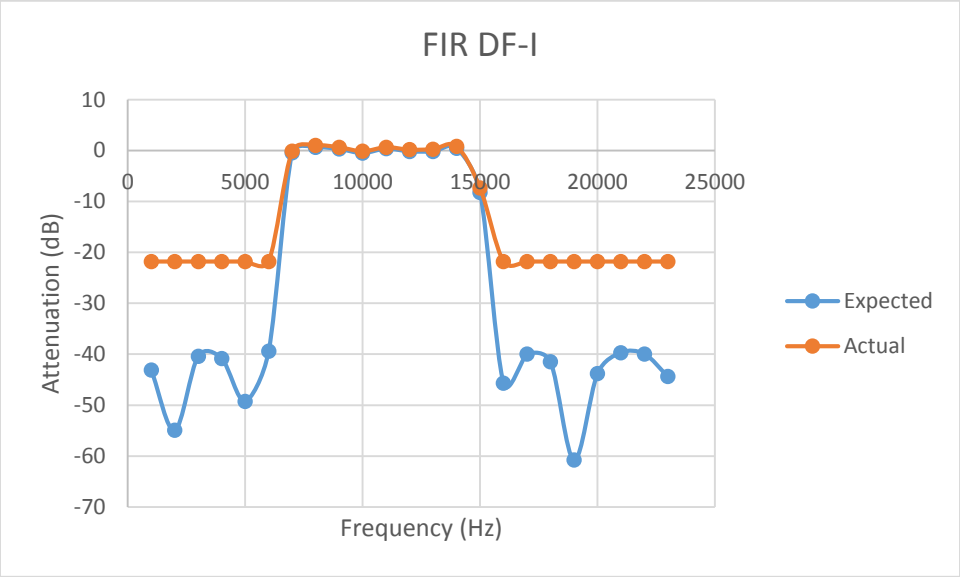

Figure 1: FIR DF-I Frequency Response

I also measured the number of cycles taken to calculate a single output value and found it to take 3,167 cycles.

Step 2

After taking reference measurements on the Direct Form I structure filter, I used the tf2sos function in MATLAB to create the Second Order Sections matrix and a gain coefficient from the DF-I filter coefficients. The MATLAB code that was used to do this can be found in the Appendix. Next, I wrote an implementation in C that uses circular buffers to implement the filter. This code can also be found in the Appendix. After completing the code, I ran it and tested it on the DSK board. I took measurements at selected frequencies to create a chart comparing the expected and actual frequency response of the filter. That chart is shown below.

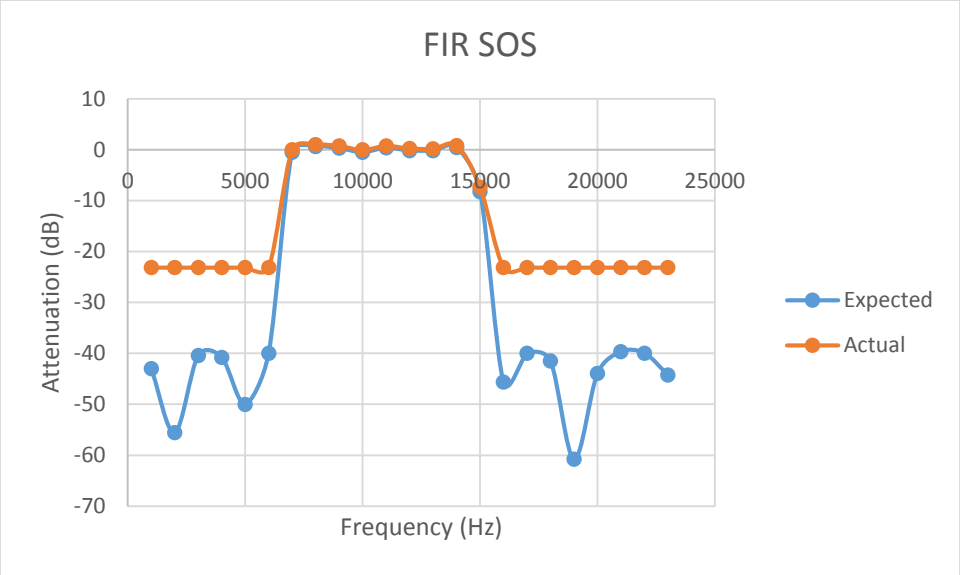

Figure 2: FIR SOS Frequency Response

The frequency response of the FIR SOS implementation lines up nicely with the expected frequency response. The only noticeable difference is in the areas where there is a high level of attenuation. In these areas, the noise going into the oscilloscope becomes larger than the signal coming from the DSK and the peak-to-peak voltage of that noise is shown. I did have one issue when running my code on the DSK board. The number of cycles required to calculate each output value is greater than the number of cycles available to service a single interrupt. I measure the number of cycles required with no optimization enabled to be 5,111 cycles, longer than the maximum allowed number of cycles at 48 kHz of 4,692. I fixed this in two ways. First, I ran the code with a lower sampling rate of 8 kHz and found that it worked correctly. I then recompiled the code with optimization enabled and found that it now only took 2,693 cycles to service an interrupt and could therefore be run at 48 kHz.

Step 3

Next, I needed to use the tf2latc function in MATLAB to create the necessary coefficients for a lattice structure implementation. MATLAB encountered a problem with this due to some of the zeros being on the unit circle, so I made my code find those zeros and move them slightly off the unit circle. The MATLAB code that I used to do this can be found in the Appendix. Once I computed the coefficients, I created an implementation in C and ran it on the DSK. It initially did not work for the same reason as the previous filter, taking too many clock cycles to service an interrupt. I measured the number of clock cycles and found it to take 9,396 cycles with no optimization, way over the limit of 4,692 cycles. Initially, I fixed this by running the board with a sampling rate of 8 kHz and found it to work correctly. I then enabled optimization and found that it now took 4,495 cycles to service an interrupt, slightly below the limit. Because of this, I was able to take the necessary measurements to compare the actual frequency response to the expected one. The graph of this is shown below.

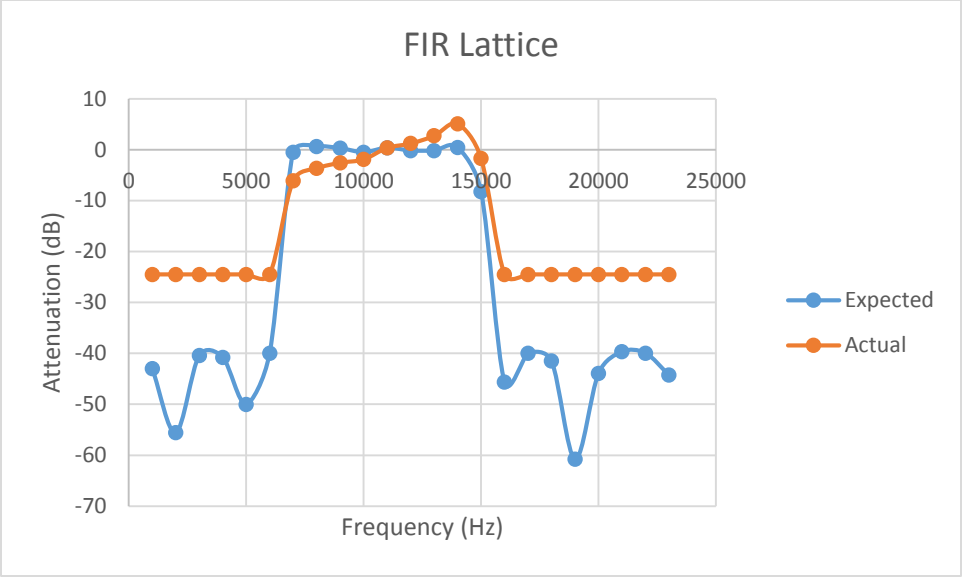

Figure 3: FIR Lattice Frequency Response

As can be seen in the figure, the frequency response of the filter is slightly different than the expected frequency response. This is likely because of the adjustments that had to be made to the zeros to get the MATLAB function to work correctly.

Task 2

Step 1

For step 1, I created a Direct Form II implementation of the IIR filter from Lab 4. To get the coefficients that were used with this, I first opened the FDA Tool back up and generated the DF-II coefficients. These coefficients can be found in the appendix. Then, using the block diagram of the filter structure as a reference, I created a C implementation of the filter, which can also be found in the Appendix. I then ran the code and found that it worked correctly. Upon testing the design, I found that it takes 1,360 clock cycles to service an interrupt without optimization and 863 cycles without optimization. I then took measurements to compare the expected and actual frequency responses and found that they line up nicely. That chart can be found below.

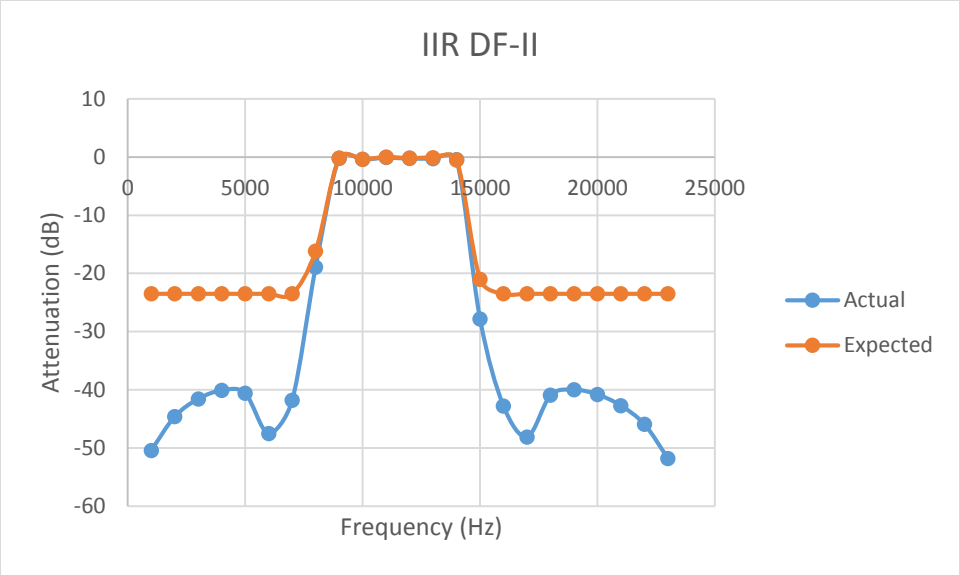

Figure 4: IIR DF-II Frequency Response

Step 2

For step 2, I created a Second Order Sections implementation of the IIR filter from Lab 4. To do this, I first used MATLAB to create the SOS matrix. The code that was used to do this can be found in the Appendix. I then created a C implementation of the SOS filter which can also be found in the Appendix. Upon running this code, I found that it did work correctly without optimization at a 48 kHz sampling rate. This is likely because I used the elliptic filter from Lab 4 which creates a SOS matrix that is only 5 x 6. Upon testing the design, I found that it takes 1,576 cycles to service an interrupt without optimization and 910 cycles with optimization. I then took measurements to compare the expected and actual frequency responses and found that the line up nicely. That chart can be found below.

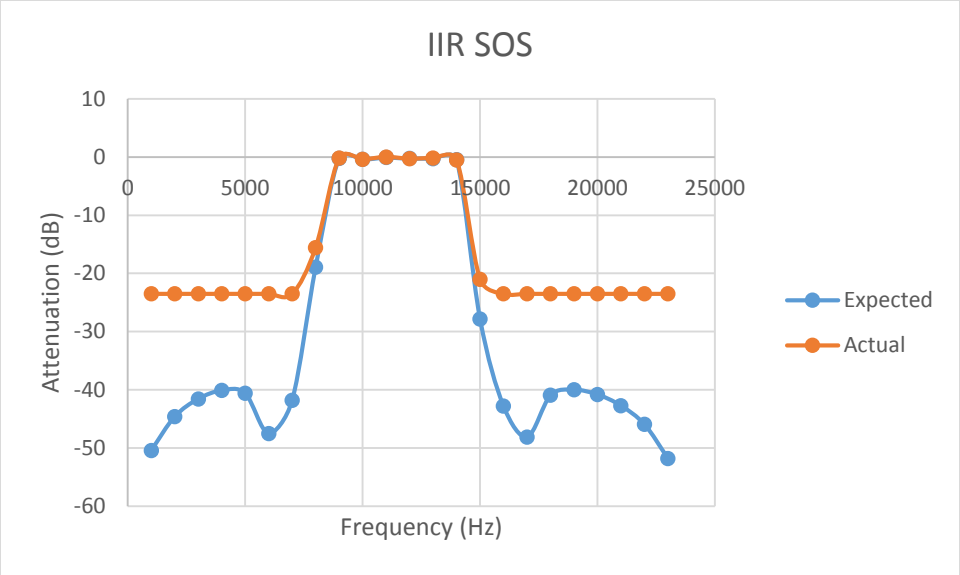

Figure 5: IIR SOS Frequency Response

Conclusion

In this lab, I got hands-on experience with different filter structures, specifically FIR DF-II/SOS, FIR Lattice, and IIR DF-II/SOS. I also learned more about the processes necessary to convert between them and the appropriate MATLAB functions to use. Finally, I feel that I have a better understanding of circular buffering and using the DSK board.

Appendix

fir_to_sos.m

```
clear all;
B = [
-0.004912325099208618
-0.014619079916495251
 0.0060757570503416243
 0.01952535804916462
 0.0024331187695853683
-0.009874609910231035 8
-0.0037963320758576859
-0.012010346542919192
-0.01035742425347112
 0.01992326384135619
 0.021487608632075595
-0.0036831076567773926
-0.0025285456210165994
-0.010532837642683505
-0.034386903509944826
-0.0011850529678652522
 0.039757652237396482
 0.014026994706631228
 0.0043585313963652086
```

```
 0.014292489095961029
-0.04562980713703095
-0.062041754213086692
 0.02997771785352504
 0.041487703139491801
 0.0066936154158850807
 0.087583480119140356
 0.030241608546776121
-0.23544107929841201
-0.15849062460484048
 0.26627195921759944
 0.26627195921759944
-0.15849062460484048
-0.23544107929841201
 0.030241608546776121
 0.087583480119140356
 0.0066936154158850807
 0.041487703139491801
 0.02997771785352504
-0.062041754213086692
-0.04562980713703095
 0.014292489095961029
 0.0043585313963652086
 0.014026994706631228
 0.039757652237396482
-0.0011850529678652522
-0.034386903509944826
-0.010532837642683505
-0.0025285456210165994
-0.0036831076567773926
 0.021487608632075595
 0.01992326384135619
-0.01035742425347112
-0.012010346542919192
-0.0037963320758576859
-0.0098746099102310358
 0.0024331187695853683
 0.01952535804916462
 0.0060757570503416243
-0.014619079916495251
-0.0049123250992086183 ];

A = 1;

[SOS, G] = tf2sos(B, A);

sampling_freq = 48000;
signal_freq = 12000;
time = (0:1/sampling_freq:0.05)';
audio_in = sin(2 * pi * signal_freq * time);

audio_out = sosfilt(SOS,audio_in) * G;
plot(audio_out);
```

fir_to_latc.m

```
clear all;
B = [
-0.004912325099208618
-0.014619079916495251
 0.0060757570503416243
 0.01952535804916462
 0.0024331187695853683
-0.0098746099102310358
-0.0037963320758576859
-0.012010346542919192
-0.01035742425347112
 0.01992326384135619
 0.021487608632075595
-0.0036831076567773926
-0.0025285456210165994
-0.010532837642683505
-0.034386903509944826
-0.0011850529678652522
 0.039757652237396482
 0.014026994706631228
 0.0043585313963652086
 0.014292489095961029
-0.04562980713703095
-0.062041754213086692
 0.02997771785352504
 0.041487703139491801
 0.0066936154158850807
 0.087583480119140356
 0.030241608546776121
-0.23544107929841201
-0.15849062460484048
 0.26627195921759944
 0.26627195921759944
-0.15849062460484048
-0.23544107929841201
 0.030241608546776121
 0.087583480119140356
 0.0066936154158850807
 0.041487703139491801
 0.02997771785352504
-0.062041754213086692
-0.04562980713703095
 0.014292489095961029
 0.0043585313963652086
 0.014026994706631228
 0.039757652237396482
-0.0011850529678652522
-0.034386903509944826
-0.010532837642683505
-0.0025285456210165994
-0.0036831076567773926
 0.021487608632075595
 0.01992326384135619
-0.01035742425347112
```

```
-0.012010346542919192
-0.0037963320758576859
-0.0098746099102310358
 0.0024331187695853683
 0.01952535804916462
 0.0060757570503416243
-0.014619079916495251
-0.0049123250992086183 ];

[SOS, G] = tf2sos(B,1);
[Z,P,K] = sos2zp(SOS,G);

for i=1:length(Z)
    if abs(Z(i)) == 1
        Z(i) = Z(i) + 0.01;
    end
end

[SOS, G] = zp2sos(Z,P,K);
[Bn,An] = sos2tf(SOS,G);
K = tf2latc(Bn);

sampling_freq = 48000;
signal_freq = 12000;
time = (0:1/sampling_freq:0.05)';
audio_in = sin(2 * pi * signal_freq * time);

[audio_out, temp] = latcfilt(K,audio_in);
audio_out = audio_out * G;
plot(audio_out);
```

iir_to_sos.m

```
clear all;
B = [
     0.022446109482960157
    -0.010573379073474012
     0.014402415206826613
    -0.0080366144026941533
     0.022544111354959838
     0.00000000000000000034694469519536142
    -0.022544111354959838
     0.0080366144026941498
    -0.014402415206826613
     0.010573379073474012
    -0.022446109482960157 ];

A = [
     1
    -0.70635031270725446
     3.5219499200390536
    -2.0200628628750592
     5.5383108441644708
    -2.4294345269044149
```

```
     4.6510805728204474
    -1.4124566248726653
     2.0783744842441458
    -0.3332994829583692
     0.39240786067834416 ];

[SOS, G] = tf2sos(B,A);

sampling_freq = 48000;
signal_freq = 12000;
time = (0:1/sampling_freq:0.05)';
audio_in = sin(2 * pi * signal_freq * time);

audio_out = sosfilt(SOS,audio_in) * G;
plot(audio_out);
```

FIRmono_ISRs.c (FIR SOS)

```
// Welch, Wright, & Morrow,
// Real-time Digital Signal Processing, 2011

///////////////////////////////////////////////////////////////////////
// Filename: ISRs.c
//
// Synopsis: Interrupt service routine for codec data transmit/receive
//
///////////////////////////////////////////////////////////////////////

#include "DSP_Config.h"
#include "coeff.h"   // load the filter coefficients, B[n] ... extern

// Data is received as 2 16-bit words (left/right) packed into one
// 32-bit word.  The union allows the data to be accessed as a single
// entity when transferring to and from the serial port, but still be
// able to manipulate the left and right channels independently.

#define LEFT  0
#define RIGHT 1

volatile union {
       Uint32 UINT;
       Int16 Channel[2];
} CodecDataIn, CodecDataOut;


/* add any global variables here */
float workingData[3] = {0.0f, 0.0f, 0.0f};
float section[nSections][3] = { {0.0, 0.0, 0.0 }, {0.0, 0.0, 0.0 },  {0.0, 0.0, 0.0 },
{0.0, 0.0, 0.0 }, {0.0, 0.0, 0.0 }, {0.0, 0.0, 0.0 }, {0.0, 0.0, 0.0 }, {0.0, 0.0, 0.0 },
{0.0, 0.0, 0.0 }, {0.0, 0.0, 0.0 }, {0.0, 0.0, 0.0 }, {0.0, 0.0, 0.0 }, {0.0, 0.0, 0.0 },
{0.0, 0.0, 0.0 }, {0.0, 0.0, 0.0 }, {0.0, 0.0, 0.0 }, {0.0, 0.0, 0.0 }, {0.0, 0.0, 0.0 },
{0.0, 0.0, 0.0 }, {0.0, 0.0, 0.0 }, {0.0, 0.0, 0.0 }, {0.0, 0.0, 0.0 }, {0.0, 0.0, 0.0 },
{0.0, 0.0, 0.0 }, {0.0, 0.0, 0.0 }, {0.0, 0.0, 0.0 }, {0.0, 0.0, 0.0 }, {0.0, 0.0, 0.0 },
{0.0, 0.0, 0.0 }, {0.0, 0.0, 0.0 }, };
```

```
int wDataBase = 0;
int wDataPtr = 0;
int sDataBase = 0;
int sDataPtr = 0;
int sTemp;
Int32 i;


interrupt void Codec_ISR()
////////////////////////////////////////////////////////////////////////
// Purpose:   Codec interface interrupt service routine
//
// Input:     None
//
// Returns:   Nothing
//
// Calls:     CheckForOverrun, ReadCodecData, WriteCodecData
//
// Notes:     None
////////////////////////////////////////////////////////////////////////
{

       if(CheckForOverrun())                                  // overrun error occurred
(i.e. halted DSP)
              return;                                                          // so serial
port is reset to recover

       CodecDataIn.UINT = ReadCodecData();             // get input data samples

    workingData[wDataBase] = CodecDataIn.Channel[LEFT];
    wDataPtr = wDataBase;
    sDataPtr = sDataBase;

    section[0][sDataPtr] = SOS[0][0] * workingData[wDataPtr];
    wDataPtr = wDataPtr - 1;
    if( wDataPtr < 0 )
        wDataPtr = 2;

    section[0][sDataPtr] += SOS[0][1] * workingData[wDataPtr];
    wDataPtr--;
    if( wDataPtr < 0 )
        wDataPtr = 2;

    section[0][sDataPtr] += SOS[0][2] * workingData[wDataPtr];

    for( i = 1; i < nSections; i++ )
    {
        sTemp = sDataPtr;
        section[i][sDataPtr] = SOS[i][0] * section[i - 1][sDataPtr];
        sTemp--;
        if( sTemp < 0 )
            sTemp = 2;

        section[i][sDataPtr] += SOS[i][1] * section[i - 1][sTemp];
        sTemp--;
        if( sTemp < 0 )
            sTemp = 2;
```

```
        section[i][sDataPtr] += SOS[i][2] * section[i - 1][sTemp];
    }

       CodecDataOut.Channel[LEFT]  = section[nSections - 1][sDataPtr] * G; // store
filtered value

    sDataBase++;
    if( sDataBase > 2 )
        sDataBase = 0;

    wDataBase++;
    if( wDataBase > 2 )
        wDataBase = 0;

       WriteCodecData(CodecDataOut.UINT);       // send output data to  port
}
```

Coeff.c (FIR SOS)

```
// Welch, Wright, & Morrow,
// Real-time Digital Signal Processing, 2011

/* coeff.c                            */
/* FIR filter coefficients            */
/* exported by MATLAB using FIR_DUMP2C */

#include "coeff.h"

float SOS[nSections][6] = {
             { 1,  2.98724420483449,      0,                   1, 0, 0 },
             { 1,  1.33475669594793,      0.334756695947931, 1, 0, 0 },
             { 1, -0.0970008952457194,    1.28633325500842,  1, 0, 0 },
             { 1, -0.585357662041507,     1.28200740861843,  1, 0, 0 },
             { 1,  0.396578667982785,     1.27514245948497,  1, 0, 0 },
             { 1, -1.02674058332933,      1.24928690379015,  1, 0, 0 },
             { 1,  1.81090943110374,      1.00000000000001,  1, 0, 0 },
             { 1,  1.95171678731647,      1.00000000000000,  1, 0, 0 },
             { 1, -1.93476861042596,      1.00000000000000,  1, 0, 0 },
             { 1,  1.01895862045961,      1.00000000000000,  1, 0, 0 },
             { 1, -1.87359018808507,      1.00000000000000,  1, 0, 0 },
             { 1, -1.42483975827141,      1.00000000000000,  1, 0, 0 },
             { 1, -1.70071711149304,      1.00000000000000,  1, 0, 0 },
             { 1,  1.30886697498458,      1.00000000000000,  1, 0, 0 },
             { 1,  0.922379913196053,     1.00000000000000,  1, 0, 0 },
             { 1,  1.15900002833138,      1.00000000000000,  1, 0, 0 },
             { 1,  1.98786243702094,      0.999999999999999, 1, 0, 0 },
             { 1, -1.79463772269755,      0.999999999999999, 1, 0, 0 },
             { 1,  1.89230204823063,      0.999999999999998, 1, 0, 0 },
             { 1, -1.97633195641700,      0.999999999999998, 1, 0, 0 },
             { 1, -1.59742151666035,      0.999999999999996, 1, 0, 0 },
             { 1, -1.49646926446452,      0.999999999999996, 1, 0, 0 },
             { 1,  1.58955264504626,      0.999999999999996, 1, 0, 0 },
             { 1, -1.99736496139526,      0.999999999999996, 1, 0, 0 },
             { 1,  1.70932250199171,      0.999999999999996, 1, 0, 0 },
             { 1,  1.45464677570174,      0.999999999999992, 1, 0, 0 },
```

```
            { 1, -0.821861319617099,     0.800456642078092, 1, 0, 0 },
            { 1,  0.311007342774050,     0.784226101610566, 1, 0, 0 },
            { 1, -0.456594601643000,     0.780026693510033, 1, 0, 0 },
            { 1, -0.0754088373817896,    0.777403519738324, 1, 0, 0 },
};

float G = -0.00491232509920862;
```

coeff.h (FIR SOS)

```
// Welch, Wright, & Morrow,
// Real-time Digital Signal Processing, 2011

/* coeff.h                              */
/* FIR filter coefficients              */
/* exported by MATLAB using FIR_DUMP2C */

#define nSections 30

extern float SOS[30][6];
extern float G;
```

FIRmono_ISRs_LATC.c

```
// Welch, Wright, & Morrow,
// Real-time Digital Signal Processing, 2011

////////////////////////////////////////////////////////////////////////
// Filename: ISRs.c
//
// Synopsis: Interrupt service routine for codec data transmit/receive
//
////////////////////////////////////////////////////////////////////////

#include "DSP_Config.h"
#include "coeff.h"   // load the filter coefficients, B[n] ... extern

// Data is received as 2 16-bit words (left/right) packed into one
// 32-bit word.  The union allows the data to be accessed as a single
// entity when transferring to and from the serial port, but still be
// able to manipulate the left and right channels independently.

#define LEFT  0
#define RIGHT 1

volatile union {
       Uint32 UINT;
       Int16 Channel[2];
} CodecDataIn, CodecDataOut;


/* add any global variables here */
float workingDataA[N] = {0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f,
0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f,
0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f,
```

```
0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f,
0.0f, 0.0f, 0.0f, 0.0f};
float workingDataB[N][2] = { {0.0f, 0.0f}, {0.0f, 0.0f}, {0.0f, 0.0f}, {0.0f, 0.0f},
{0.0f, 0.0f}, {0.0f, 0.0f}, {0.0f, 0.0f}, {0.0f, 0.0f}, {0.0f, 0.0f}, {0.0f, 0.0f},
{0.0f, 0.0f}, {0.0f, 0.0f}, {0.0f, 0.0f}, {0.0f, 0.0f}, {0.0f, 0.0f}, {0.0f, 0.0f},
{0.0f, 0.0f}, {0.0f, 0.0f}, {0.0f, 0.0f}, {0.0f, 0.0f}, {0.0f, 0.0f}, {0.0f, 0.0f},
{0.0f, 0.0f}, {0.0f, 0.0f}, {0.0f, 0.0f}, {0.0f, 0.0f}, {0.0f, 0.0f}, {0.0f, 0.0f},
{0.0f, 0.0f}, {0.0f, 0.0f}, {0.0f, 0.0f}, {0.0f, 0.0f}, {0.0f, 0.0f}, {0.0f, 0.0f},
{0.0f, 0.0f}, {0.0f, 0.0f}, {0.0f, 0.0f}, {0.0f, 0.0f}, {0.0f, 0.0f}, {0.0f, 0.0f},
{0.0f, 0.0f}, {0.0f, 0.0f}, {0.0f, 0.0f}, {0.0f, 0.0f}, {0.0f, 0.0f}, {0.0f, 0.0f},
{0.0f, 0.0f}, {0.0f, 0.0f}, {0.0f, 0.0f}, {0.0f, 0.0f}, {0.0f, 0.0f}, {0.0f, 0.0f},
{0.0f, 0.0f}, {0.0f, 0.0f}, {0.0f, 0.0f}, {0.0f, 0.0f}, {0.0f, 0.0f}, };

short audioIn = 0;
short audioInPrev = 0;
int wDataBase = 0;
int wDataPtr = 0;
Int32 i;


interrupt void Codec_ISR()
///////////////////////////////////////////////////////////////////////
// Purpose:   Codec interface interrupt service routine
//
// Input:     None
//
// Returns:   Nothing
//
// Calls:     CheckForOverrun, ReadCodecData, WriteCodecData
//
// Notes:     None
///////////////////////////////////////////////////////////////////////
{
       /* add any local variables here */
  int wTemp;

       if(CheckForOverrun())                                    // overrun error occurred
(i.e. halted DSP)
              return;                                                  // so serial
port is reset to recover

       CodecDataIn.UINT = ReadCodecData();             // get input data samples

       wDataPtr = wDataBase;
       audioIn = CodecDataIn.Channel[LEFT];

    workingDataA[0] = audioIn + (-K[0])*audioInPrev;
    workingDataB[0][wDataPtr] = (-K[0])*audioIn + audioInPrev;

    for(i = 1; i < N; i++)
    {
        wTemp = wDataPtr - 1;
        if( wTemp < 0 )
            wTemp = 1;
        workingDataA[i] = workingDataA[i - 1] + (-K[i])*workingDataB[i-1][wTemp];
        workingDataB[i][wDataPtr] = (-K[i])*workingDataA[i-1] + workingDataB[i-1][wTemp];
    }
```

```
       CodecDataOut.Channel[LEFT]  = workingDataA[N-1] * G;

    audioInPrev = audioIn;

    wDataBase++;
    if( wDataBase > 1 )
        wDataBase = 0;

       WriteCodecData(CodecDataOut.UINT);       // send output data to  port
}
```

Coeff.c (FIR LATC)

```
// Welch, Wright, & Morrow,
// Real-time Digital Signal Processing, 2011

/* coeff.c                              */
/* FIR filter coefficients              */
/* exported by MATLAB using FIR_DUMP2C */

#include "coeff.h"

float K[N] =
{ 0.708347286579795,
0.866280471499831,
-0.847743892850342,
-0.171273000542422,
0.838158969398302,
0.300314030074267,
-1.66833218603506,
-0.360846115054084,
-1.94829783837347,
0.495627485056741,
0.909550497084756,
2.69852593750170,
-0.803218561265403,
0.901525743859919,
-3.58299548433647,
-0.903274270974640,
-0.541367131573569,
-1.03118694145163,
-113.541808392627,
1.05490872136030,
0.652223466958557,
1.04617059992913,
-7.54832747249398,
-1.03196858987343,
0.911519514689635,
-5.67864062772742,
-0.819486856740405,
-1.01562783896913,
-0.964395903931580,
-80.7347319767867,
0.965869732345182,
0.842886495164584,
-1.56320394344333,
-1.03944622999894,
```

```
17.4429567789071,
1.00149607131183,
0.623298852074261,
-15.4440356455750,
-0.647094762276179,
-1.55614152075418,
0.996622013184018,
161.240901531064,
-0.999589884888182,
0.470039435784371,
-1.67797926459951,
-0.354104089650592,
0.555408894948825,
-1.44997011607623,
-5.93136401326020,
1.10513118596252,
0.849737166965821,
4.74192920256734,
-1.93912514157582,
-1.33838651792281,
-1.51143764368395,
-0.603654987777907,
0.999999999995231,
0.453257672577323,
0.990876200868026, };

float G = -0.00491232509920862;
```

coeff.h (FIR LATC)

```
// Welch, Wright, & Morrow,
// Real-time Digital Signal Processing, 2011

/* coeff.h                              */
/* FIR filter coefficients              */
/* exported by MATLAB using FIR_DUMP2C */

#define N 59

extern float K[N];
extern float G;
```

IIRmono_ISRs_DFii.c

```
// Welch, Wright, & Morrow,
// Real-time Digital Signal Processing, 2011

////////////////////////////////////////////////////////////////////////
// Filename: ISRs.c
//
// Synopsis: Interrupt service routine for codec data transmit/receive
//
////////////////////////////////////////////////////////////////////////

#include "DSP_Config.h"
#include "coeff.h"
```

```
// Data is received as 2 16-bit words (left/right) packed into one
// 32-bit word.  The union allows the data to be accessed as a single
// entity when transferring to and from the serial port, but still be
// able to manipulate the left and right channels independently.

#define LEFT  0
#define RIGHT 1

volatile union {
       Uint32 UINT;
       Int16 Channel[2];
} CodecDataIn, CodecDataOut;

//float section[nSections][3] = { {0.0f, 0.0f, 0.0f}, {0.0f, 0.0f, 0.0f}, {0.0f, 0.0f,
0.0f}, {0.0f, 0.0f, 0.0f}, {0.0f, 0.0f, 0.0f}};

//float w[nSections][3] = { {0.0f, 0.0f, 0.0f}, {0.0f, 0.0f, 0.0f}, {0.0f, 0.0f, 0.0f},
{0.0f, 0.0f, 0.0f}, {0.0f, 0.0f, 0.0f}};

float delayed[N + 1] = {0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f,
0.0f};

//int sDataBase = 0;
//float workingData = 0.0f;

int i;


interrupt void Codec_ISR()
////////////////////////////////////////////////////////////////////////
// Purpose:   Codec interface interrupt service routine
//
// Input:     None
//
// Returns:   Nothing
//
// Calls:     CheckForOverrun, ReadCodecData, WriteCodecData
//
// Notes:     None
////////////////////////////////////////////////////////////////////////
{
       /* add any local variables here */
    int sTemp;
    float left = 0.0f;
    float right = 0.0f;
    float out = 0.0f;
    float currentIn = 0.0f;

       if(CheckForOverrun())                                  // overrun error occurred
(i.e. halted DSP)
              return;                                                    // so serial
port is reset to recover

       CodecDataIn.UINT = ReadCodecData();              // get input data samples
       currentIn = CodecDataIn.Channel[LEFT];
       delayed[0] = currentIn;
```

```
	left = delayed[0];

	for(i = 1; i < N; i++)
	{
		left -= (A[i] * delayed[i]);
	}

	delayed[0] = left;

	for(i = 0; i < N; i++)
	{
		right += ( B[i] * delayed[i] );
	}

	for(i = N; i > 0; i--)
		delayed[i] = delayed[i-1];

	CodecDataOut.Channel[LEFT] = right;

	WriteCodecData(CodecDataOut.UINT);	// send output data to  port

	return;
}
```

Coeff.c (IIR DF-II)

```
// Welch, Wright, & Morrow,
// Real-time Digital Signal Processing, 2011

/* coeff.c                              */
/* FIR filter coefficients              */
/* exported by MATLAB using FIR_DUMP2C */


/* Equiripple FIR LPF with passband to */
/* 5 kHz assuming Fs=48 kHz            */


#include "coeff.h"

float B[N] = {
               0.022446109482960157,
              -0.010573379073474012,
               0.014402415206826613,
              -0.0080366144026941533,
               0.022544111354959838,
               0.00000000000000000034694469519536142,
              -0.022544111354959838,
               0.0080366144026941498,
              -0.014402415206826613,
               0.010573379073474012,
              -0.022446109482960157
};

float A[N] = {
               1,
              -0.70635031270725446,
```

```
               3.5219499200390536,
              -2.0200628628750592,
               5.5383108441644708,
              -2.4294345269044149,
               4.6510805728204474,
              -1.4124566248726653,
               2.0783744842441458,
              -0.3332994829583692,
               0.39240786067834416
};
```

IIRmono_ISRs_SOS.c

```
// Welch, Wright, & Morrow,
// Real-time Digital Signal Processing, 2011

/////////////////////////////////////////////////////////////////////////
// Filename: ISRs.c
//
// Synopsis: Interrupt service routine for codec data transmit/receive
//
/////////////////////////////////////////////////////////////////////////

#include "DSP_Config.h"
#include "coeff.h"

// Data is received as 2 16-bit words (left/right) packed into one
// 32-bit word.  The union allows the data to be accessed as a single
// entity when transferring to and from the serial port, but still be
// able to manipulate the left and right channels independently.

#define LEFT  0
#define RIGHT 1

volatile union {
       Uint32 UINT;
       Int16 Channel[2];
} CodecDataIn, CodecDataOut;

float section[nSections][3] = { {0.0f, 0.0f, 0.0f}, {0.0f, 0.0f, 0.0f}, {0.0f, 0.0f,
0.0f}, {0.0f, 0.0f, 0.0f}, {0.0f, 0.0f, 0.0f}};

float w[nSections][3] = { {0.0f, 0.0f, 0.0f}, {0.0f, 0.0f, 0.0f}, {0.0f, 0.0f, 0.0f},
{0.0f, 0.0f, 0.0f}, {0.0f, 0.0f, 0.0f}};

int sDataBase = 0;
float workingData = 0.0f;

int i;

interrupt void Codec_ISR()
/////////////////////////////////////////////////////////////////////////
// Purpose:   Codec interface interrupt service routine
//
// Input:     None
//
```

```
// Returns:   Nothing
//
// Calls:     CheckForOverrun, ReadCodecData, WriteCodecData
//
// Notes:     None
////////////////////////////////////////////////////////////////////////
{
       /* add any local variables here */
  int sTemp;

       if(CheckForOverrun())                                  // overrun error occurred
(i.e. halted DSP)
             return;                                                  // so serial
port is reset to recover

       CodecDataIn.UINT = ReadCodecData();             // get input data samples

  workingData = CodecDataIn.Channel[LEFT] * G;

  sTemp = sDataBase - 1;
  if( sTemp < 0 )
    sTemp = 2;
  w[0][sDataBase] = workingData - SOS[0][4] * w[0][sTemp];

  sTemp--;
  if( sTemp < 0 )
    sTemp = 2;
  w[0][sDataBase] -= SOS[0][5] * w[0][sTemp];

  sTemp = sDataBase - 1;
  if( sTemp < 0 )
    sTemp = 2;
  section[0][sDataBase] = SOS[0][0] * w[0][sDataBase] + SOS[0][1] * w[0][sTemp];

  sTemp--;
  if( sTemp < 0 )
    sTemp = 2;
  section[0][sDataBase] += SOS[0][2] * w[0][sTemp];

  for( i = 1; i < nSections; i++)
  {
    sTemp = sDataBase - 1;
    if( sTemp < 0 )
      sTemp = 2;
    w[i][sDataBase] = section[i-1][sDataBase] - SOS[i][4] * w[i][sTemp];

    sTemp--;
    if( sTemp < 0 )
      sTemp = 2;
    w[i][sDataBase] -= SOS[i][5] * w[i][sTemp];

    sTemp = sDataBase - 1;
    if( sTemp < 0 )
      sTemp = 2;
    section[i][sDataBase] = SOS[i][0] * w[i][sDataBase] + SOS[i][1] * w[i][sTemp];

    sTemp--;
    if( sTemp < 0 )
```

```
      sTemp = 2;
    section[i][sDataBase] += SOS[i][2] * w[i][sTemp];
  }

       CodecDataOut.Channel[LEFT] = section[nSections-1][sDataBase];

  sDataBase++;
  if( sDataBase > 2 )
    sDataBase = 0;

       WriteCodecData(CodecDataOut.UINT);        // send output data to  port

       return;
}
```

Coeff.c (IIR SOS)

```
// Welch, Wright, & Morrow,
// Real-time Digital Signal Processing, 2011

/* coeff.c                              */
/* FIR filter coefficients              */
/* exported by MATLAB using FIR_DUMP2C */

#include "coeff.h"

float SOS[nSections][6] = {
              { 1, 4.44089209850063e-16, -0.999999999999999, 1, -0.141475422543136,
0.665915800181715 },
              { 1, 1.12909702920455, 1.00000000000000, 1, 0.398220503701660,
0.801314006827437 },
              { 1, -1.33889549467908, 0.999999999999999, 1, -0.680109746360016,
0.810260129420495 },
              { 1, 0.827749074875837, 1.00000000000000, 1, 0.613337721231956,
0.951152114219966 },
              { 1, -1.08900681098941, 1.00000000000000, 1, -0.896323368737722,
0.954203923997317 },
};

float G = 0.0224461094829602;
```

coeff.h (IIR SOS)

```
// Welch, Wright, & Morrow,
// Real-time Digital Signal Processing, 2011

/* coeff.h                              */
/* FIR filter coefficients              */
/* exported by MATLAB using FIR_DUMP2C */

#define nSections 5

extern float SOS[nSections][6];
extern float G;
```