Real-Time Digital Signal Processing : Lab 5 - Real-time Implementation of IIR Filters

Jiyu Fang CID: 01054797 Deland Liu CID: 01066080

March 2018

1 Objectives

- Learn to design IIR filters using Matlab
- Implement the IIR filters using the C6713 DSK system in real-time
- Measure the filter characteristics using a netowkr or spectrum analyzer

2 Equipment

- Spectrum Digital DSK-EVM-eZdsp onboard USB Emulator(DSK6713)
- Oscilloscope
- Software signal generator
- APX520 audio analyzer

3 IIR Filter

IIR filter has a difference equation shown below.Current output is a weighted sum of current and previous M inputs and of previous N outputs. This is different from FIR filters such that IIR filters can not only have zeroes anywhere (determined by coefficients b) but also poles anywhere (determined by coefficients a). As a result, IIR filters are not inherently stable with the feedback present and become unstable when any pole is outside the unit circle on z plane.

$$y(n) = \sum_{k=0}^{M} b(k)x(n-k) - \sum_{k=1}^{N} a(k)y(n-k)$$

Taking Z-transform of difference equation would produce the transfer function below. Intuitively, for a given order, IIR filter can have sharper roll off and narrower transition band compared to FIR filter by placing poles adjacent to zeroes that are closer to the unit circle at either side of the transition band. This is because poles close to the unit circle can pull gain magnitude up to infinity while zeroes close to the unit circle drag it down close to zero. If the poles and zeros are close enough, the gain is kept at approximately one before reaching passband edge frequencies, because the product of distance from those zeros is approximately equal to the product of distance from those poles, hence a sharp narrower transition band or wider pass band could be achieved. However, it is impossible for IIR filters to have a linear phase response.

$$H(z) = \frac{b_0 + b_1 z^{-1} + \ldots + b_M z^{-M}}{1 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_N z^{-N}}$$

4 Single-Pole Filter

To implement **digital IIR filter design**, it is better to first design the filter in continuous time, then convert it to discrete time. There are standard analogue filters, such as Butterworth and Elliptic filters, of which coefficients can be found through a look-up table for a given performance. Note the transfer function of a continuous-time filter is described in Laplace domain s, where multiplying e^{-sT_s} delays the signal by T_s. The equivalent in discrete time is multiplying z^{-1} in z domain, therefore we can write:

$$z = e^{sT_s} \quad \Longrightarrow \quad s = \frac{1}{T_s} ln(z)$$

An approximation of the series exppansion of the logarithm leads to:

$$
\begin{aligned}
s &= \frac{1}{T_s} ln(z) \\
&= \frac{2}{T_s}[\frac{z-1}{z+1} + \frac{1}{3}(\frac{z-1}{z+1})^3 + \frac{1}{5}(\frac{z-1}{z+1})^5 + \frac{1}{7}(\frac{z-1}{z+1})^7 + \ldots] \\
&\approx \frac{2}{T_s}\frac{z-1}{z+1}
\end{aligned}
$$

The mapping between the analogue and digital frequencies by the bilinear transform is not linear. In other words, the relationship between analogue and digital frequencies should be:

$$w_a = w_d \qquad \textit{both in radians per second}$$

However, in the bilinear transform:

$$
\begin{aligned}
jw_a &= \frac{2}{T_s}\frac{e^j w_d T_s - 1}{e^j w_d T_s + 1} \\
&= \frac{2}{T_s}\frac{e^{j\frac{w_d T_s}{2}} - e^{-j\frac{w_d T_s}{2}}}{e^{j\frac{w_d T_s}{2}} + e^{-j\frac{w_d T_s}{2}}} \\
&= j\frac{2}{T_s}\frac{(e^{j\frac{w_d T_s}{2}} - e^{-j\frac{w_d T_s}{2}})/2j}{(e^{j\frac{w_d T_s}{2}} + e^{-j\frac{w_d T_s}{2}})/2} \\
&= j\frac{2}{T_s}\frac{\sin(w_d T_s/2)}{\cos(w_d T_s/2)} \\
&= j\frac{2}{T_s}\tan(w_d T_s/2) \\
\implies w_a &= \frac{2}{T_s}\tan(w_d T_s/2)
\end{aligned}
$$

The two frequencies are almost equal for small $w_d T_s$ since $\lim_{w_d T_s \to 0} \tan(w_d T_s/2) \approx \frac{w_d T_s}{2}$. Therefore, the higher the sampling frequency(smaller the T_s), the better the approximation.

4.1 Coefficients Calculation

Given an analogue low-pass filter:

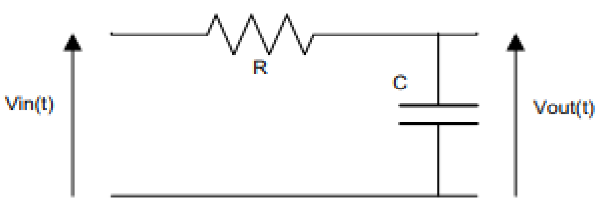

Figure 1: RC low pass filter with $R = 1k\Omega$ and $C = 1\mu F$

Since in this case the cut-off frequency:

$$f_c = \frac{1}{2\pi RC} = \frac{1}{2\pi 10^{-3}} \approx 159.15Hz \ll f_s = 8kHz$$

a good approximation can be achieved by converting it to discrete time using bilinear transform:

$$
\begin{aligned}
H_a(s) &= \frac{1/sC}{R + 1/sC} = \frac{1}{sRC + 1} \\
H_d(z) &= H_a(s)|_{s=\frac{2}{T_s}\frac{z+1}{z-1}} \\
&= \frac{1}{RC\frac{2}{T_s+1}\frac{z+1}{z-1}} \\
&= \frac{z+1}{(1-2RCf_s) + (1+2RCf_s)z} \\
&= \frac{1+z^{-1}}{(1-2RCf_s) + (1+2RCf_s)z^{-1}}
\end{aligned}
$$

$$R = 1k\Omega \quad C = 1\mu F \quad f_s = 8kHz \quad \Longrightarrow H(z) = \frac{1/17 + 1/17z^{-1}}{1 - 15/17z^{-1}}$$

Coefficients $b[0] = \frac{1}{17}$, $b[1] = \frac{1}{17}$, $a[0] = 1$, $a[1] = \frac{-15}{17}$ are checked with Matlab and are put into text file called *single_pole_coef.txt* with double precision and included in the C program.

```
R = 1000; %Resistor value
C = 10^-6;%Capacitor value
fs = 8000; %Sampling frequency
H = tf([1],[R*C,1]); %Find transfer function of RC filter in s domain
Hd = c2d(H,1/fs,'tustin') %Complete Tustin transform
[b,a] = tfdata(Hd); %Find coefficients in z domain
b = cell2mat(b);
a = cell2mat(a);
%save to single_pole_coef.txt
fileID = fopen('single_pole_coef.txt','w');
formSpec1 = 'double b[]={%.16e';
fprintf(fileID,formSpec1,b(1));
formSpec2 = ', %.16e';
fprintf(fileID,formSpec2,b(2:end));
fprintf(fileID,'};\n');
formSpec3 = 'double a[]={%.16e';
fprintf(fileID,formSpec3,a(1));
fprintf(fileID,formSpec2,a(2:end));
fprintf(fileID,'};\n');
fclose(fileID);
```

Figure 2: Matlab code for bilinear transformation

4.2 C Implementation

The implementation of IIR filter in Direct Form I requires two delay lines and two separate adders to perform convolution sum computations. We need two arrays $v[]$ and $w[]$ to maintain history of input and output samples in the delay line:

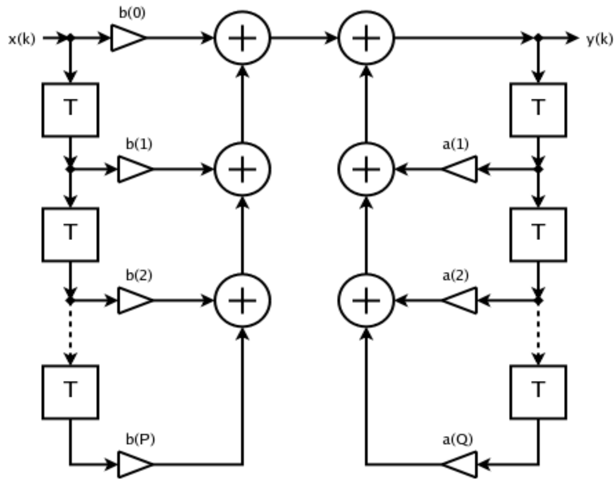

Figure 3: Direct form I

We can dynamically allocate the array sizes based on the length of numerator and denominator coefficient arrays using the function *calloc*. Hence, when filter order alters, we could always create the memory we need without manually changing array length definitions. v and w are first defined as pointers to doubles. Assume that $a[]$(numerator coefficients, including 1) and $b[]$(denominator coefficients) are of the same length.

```
//dynamically allocat array sizes to store input and output samples
order = sizeof(a)/sizeof(a[0]) -1;
w = (double *) calloc(order+1, sizeof(double));//dynamic memory, order+1 elements, each of size double
v = (double *) calloc(order+1, sizeof(double));
```

Figure 4: Dynamic allocation of memory

v and w are now the pointers to the first element of an array of doubles and thus the array can be accessed with normal way of $v[]$ and $w[]$.

The algorithm needs to complete two separate steps when a new input sample arrives: calculate output sample through convolution sum and updating values in 2 delay lines for next iteration. Note that the case of i=0 is handled separately and $v[0]$ and $w[0]$ are not used, such that b[k] can be directly multiplied with v[k] and a[k] directly multiplied with w[k] within the *for* loop. Shifting of samples to represent delay elements are also done within the *for* loop at each iteration, hence avoiding a second *for* loop. $v[1]$ and $w[1]$ are updated at the end of the function.

```
void single_pole_IIR(void){
    int k;
    samp = b[0]*sampin;                 //calculation with b[0] done individually
                                        //so within for loop can multiply v[k]with b[k]
                                        //and w[k] with a[k]
    for(k = order; k>0; k--){
        samp += v[k]*b[k] - w[k]*a[k];//perform convolution sum following
                                        //IIR difference equation
        w[k] = w[k-1];                  //perform delay for input values stored in v
        v[k] = v[k-1];                  //perform delay for output values stored in w
    }
    v[1] = sampin;                      //update first delayed element v[1], or x(n-1) for next iteration
    w[1] = samp;                        //update w[1], or y(n-1) for next iteration
}
```

Figure 5: Direct form I Implementation

4.3 Results

By driving the input with a square wave of $140Hz$ with $V_{pp} < 5.66V$ from the signal generator, we could find the time constant of the filter. Time constant is time taken for the system's step response to reach $1 - 1/e \approx 63.2\%$ of its final asymptotic value say from a step increase. Frequency of 140Hz is chosen so that it is greater than the cut-off frequency of the output high-pass filter($7Hz$) and it is low enough for the output to reach the final value(steady state) for a single step increase.

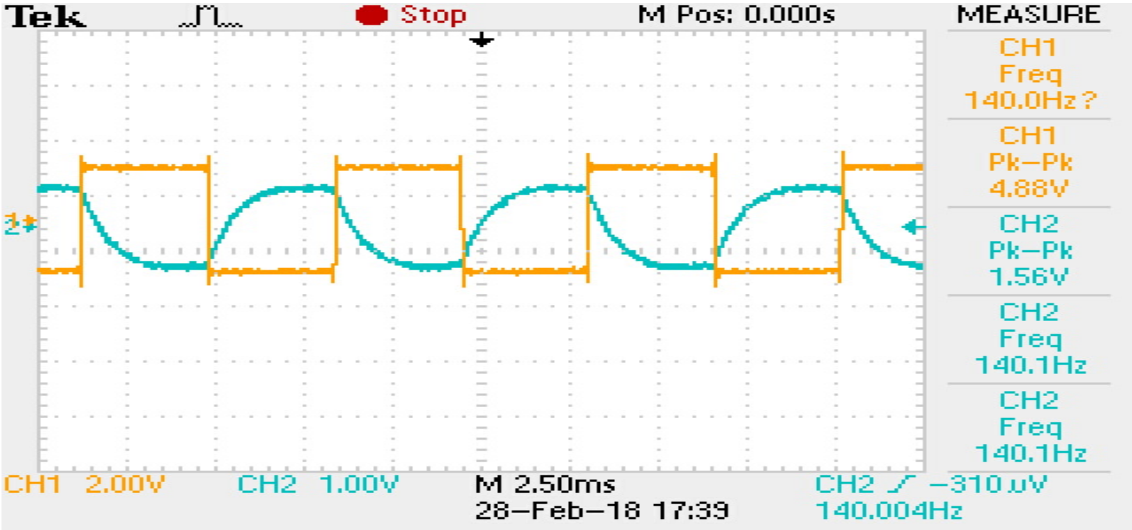

Figure 6: Square wave of $140Hz$ is applied, output could reach steady state within half of the period

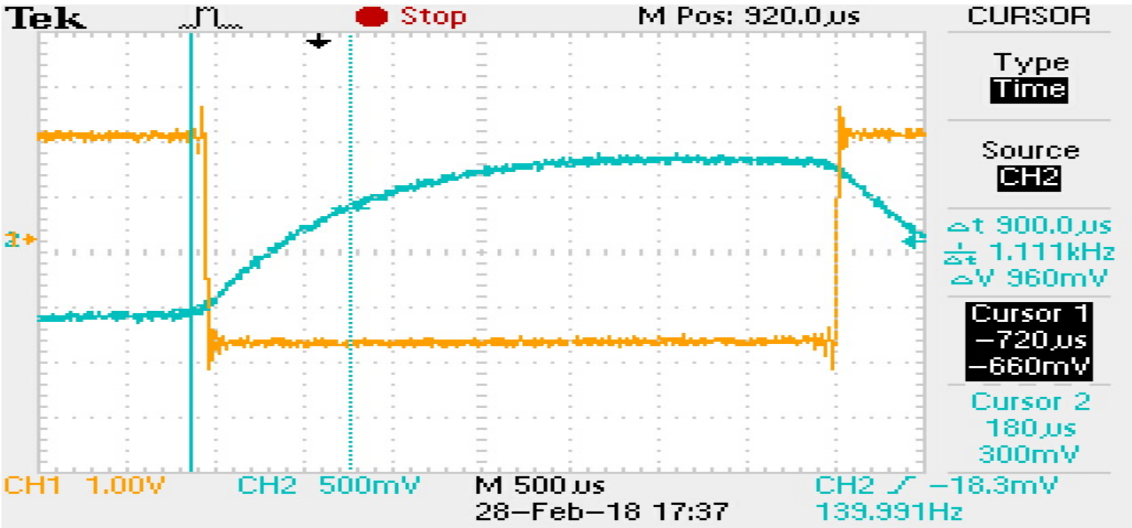

Figure 7: Step response

The time constant measured using oscilloscope is $\tau_{square\,wave} = 900\mu s$ (measured with cursors). ΔV = 1.56V, cursor1 should be situated at -660mV + ΔV ×(1-1/e) = 325.92mV, and time constant can be measured. However, this time constant is not accurate since the frequency response is also affected by other filters inside the AIC23 audio chip. In addition, as shown, cursor1 cannot be placed accurately at 325.92mV, hence accuracy is reduced. In the following section, the ideal discrete IIR response is found in MATLAB and time constant extracted using *damp* function.

```
R = 1000; %Resistor value
C = 10^-6;%Capacitor value
fs = 8000; %Sampling frequency
H = tf([1],[R*C,1]); %Find transfer function of RC filter
Hd = c2d(H,1/fs,'tustin') %Complete Tustin transform
[num,den] = tfdata(Hd); %Find coefficients
num = cell2mat(num)
[den] = cell2mat(den)
damp(Hd);
```

Figure 8: Matlab for Plotting Ideal Digital IIR Filter

$$\tau_{IIR,ideal} = 999\mu s$$

The frequency response of the discrete IIR filter and all-pass response are obtained with Audio Precision APX520. To obtain all-pass response, in the ISR the samples are simply read and written out directly, with no filtering applied:

```
void ISR_AIC (void)
{
        // samp = 0;//reset output
         //sampin = mono_read_16Bit();
         //single_pole_IIR();
         //bandpass_direct2_IIR();
         //bandpass_direct2_transposed_IIR();
         //samp = (short)samp; //rounding purpose
         mono_write_16Bit(mono_read_16Bit());
}
```

Figure 9: ISR to obtain all-pass response

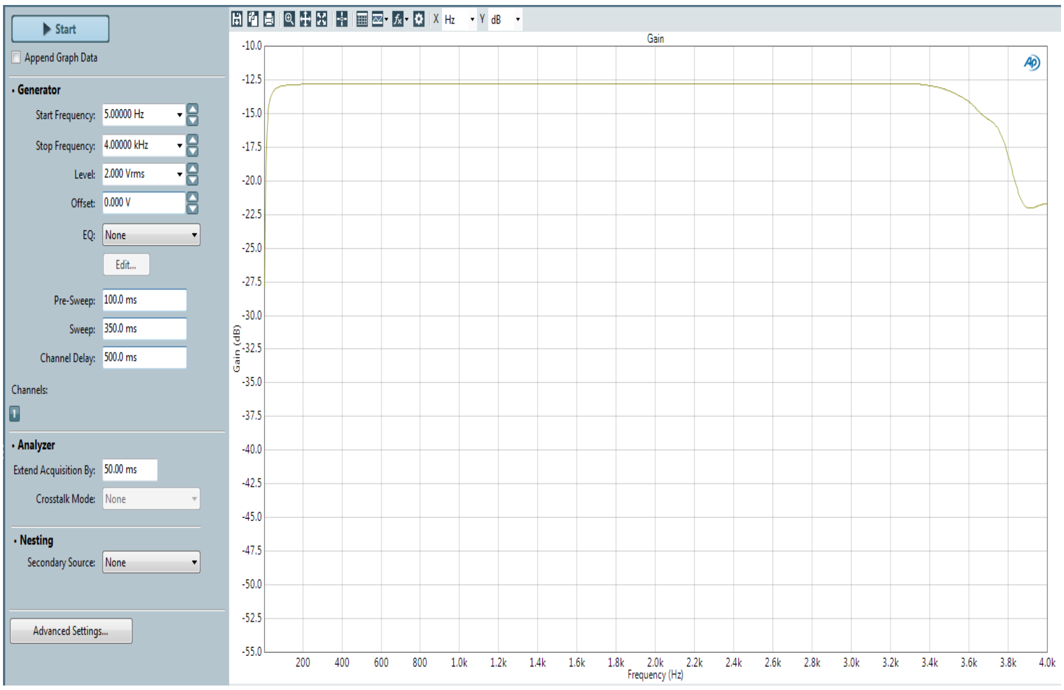

Figure 10: All Pass Magnitude Response

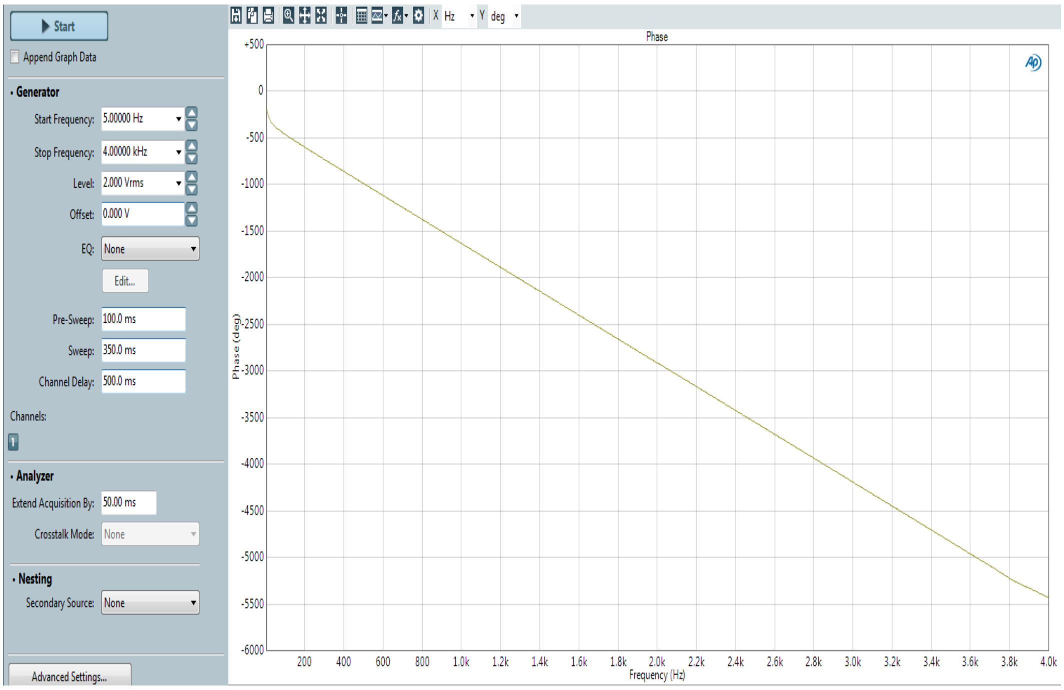

Figure 11: All Pass Phase Response

As mentioned in Lab 4, the gain of $\frac{1}{4}$ at the input of DSK unit plus some other attenuation introduces an overall attenuation of about -12.9dB. Roll-off at roughly 4kHz can be attributed to anti-aliasing filter and reconstruction filter at the input and output of the AIC23 audio chip respectively, which have cutoff at Nyquist Frequency (f_N=4kHz). Roll-off at small frequencies can be explained by high-pass filter with cut-off at 7Hz at output of Audio Chip. Note that final magnitude response in dBs equals sum of that of series filters.

IIR is applied and frequency response measured again using the same equipment:

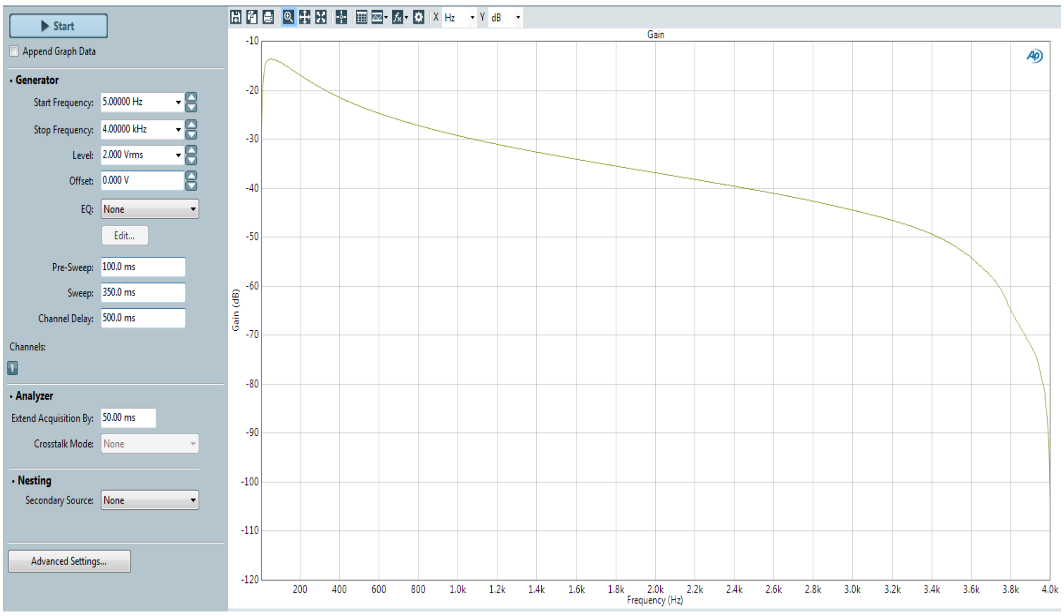

Figure 12: Measured Magnitude Response(IIR Filter Applied)

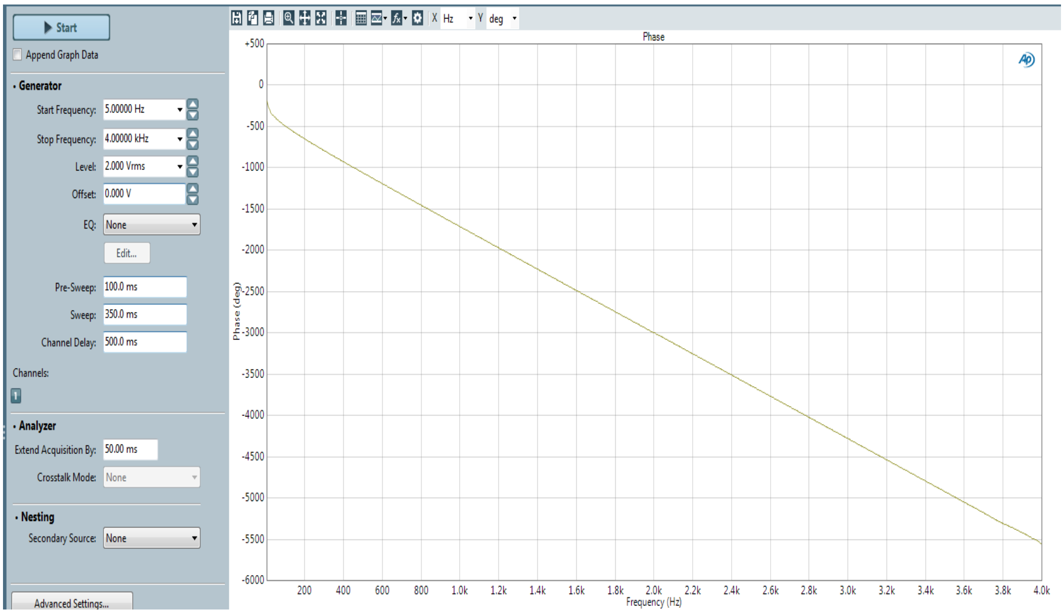

Figure 13: Measured Phase Response(IIR Filter Applied)

By exporting the graph data from APX500 to Matlab, actual discrete IIR filter frequency response is extracted by subtracting all pass response from measured IIR filter response. This is shown below. The time constant of this filter is found:

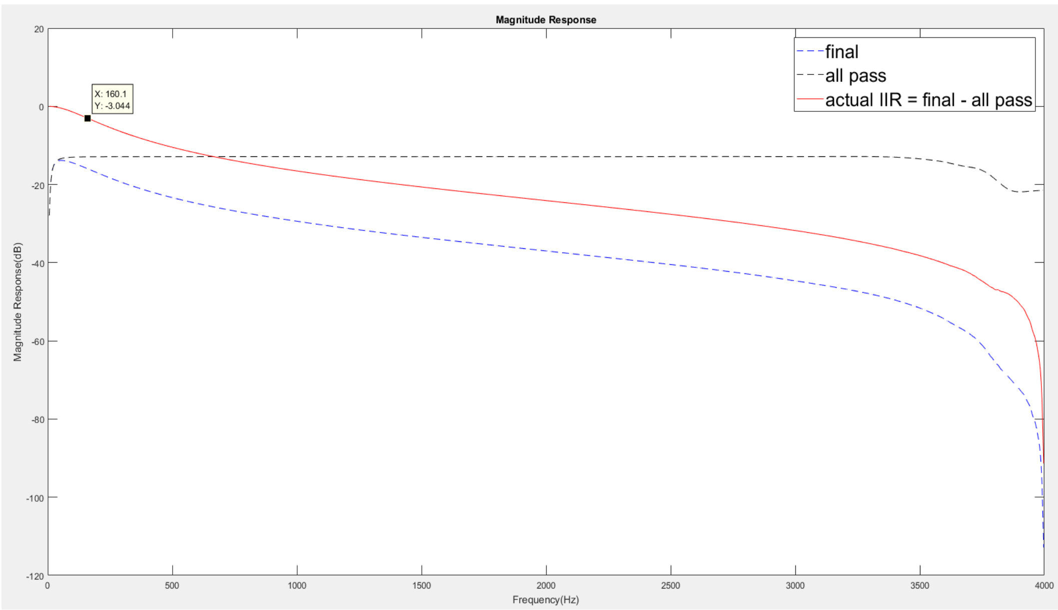

Figure 14: Final response here refers to measured response. Measured response minus all pass response to obtain actual IIR discrete filter response

The extracted actual digital IIR filter frequency response is very close to the ideal digital IIR frequency response given by the Matlab:

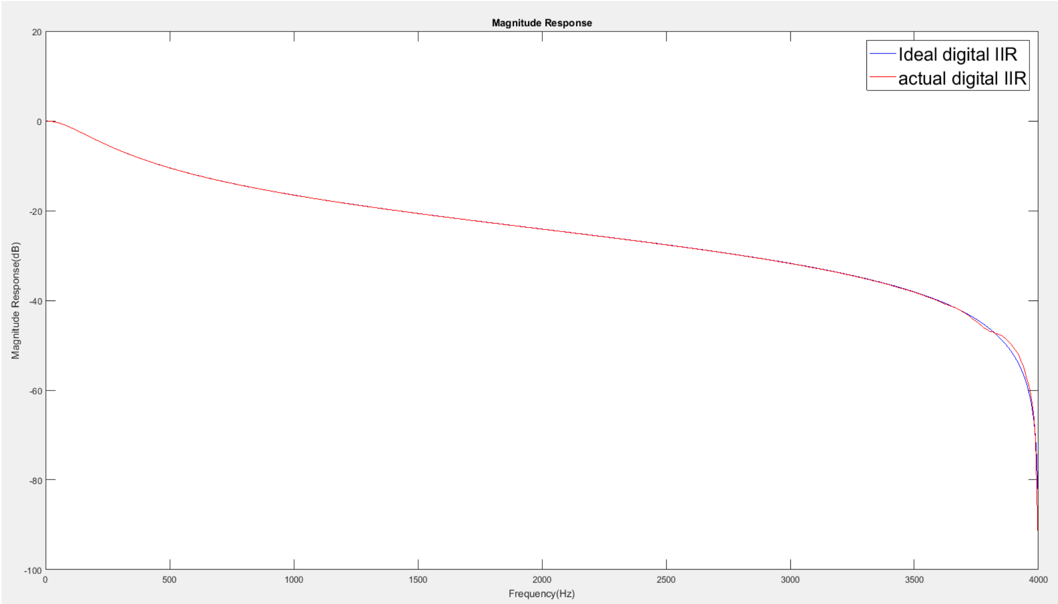

Figure 15: Final Magnitude Response

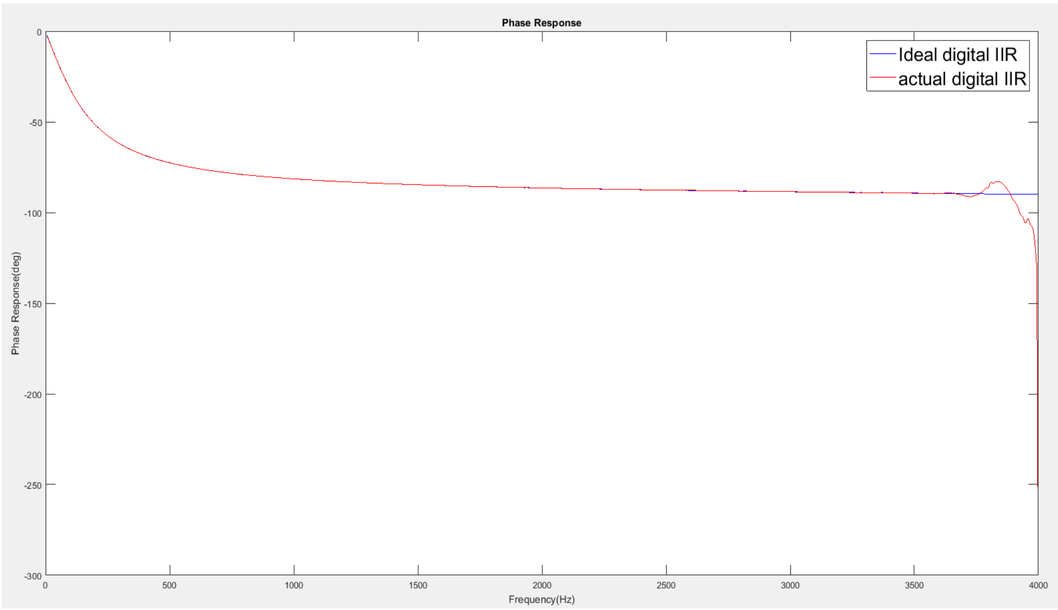

Figure 16: Final Phase Response

Both magnitude and phase response are almost identical to ideal response and only deviates when approaching to the Nyquist Frequency($f_N = 4kHz$). This might be due to non-ideal anti-aliasing filter inside AIC23 audio chip, where the frequency response above the Nyquist Frequency is non-zero, hence there is small portion of overlap in frequency domain of the reconstructed signal. This explains the deviation, which can be resulted from the magnitude and phase response of aliasing components from the image around $f_s = 8kHz$ when measuring response using APX500.

Since time constant for a first-order time-invariant system is also equal to $\tau = \frac{1}{2\pi f_c}$, where f_c is the cut-off frequency, or the frequency for which the gain is 3 dB smaller than the nominal pass band value(0 dB). Use the cursor to find the 3dB drop point on the plot, $f_c = 160.1Hz$. The actual time constant of the IIR filter implemented on DSK is found to be $\tau_{IIR,actual} = \frac{1}{2\pi 160.1} = 994\mu s$.
Note that for the RC circuit shown above, analogue measurements are taken: $f_c = \frac{1}{2\pi RC} = 159.15\,Hz$

and $\tau = RC = 1\,ms$.

-	Time Constant(μs)	Corner Frequency(Hz)
Analogue IIR response	1000	159.15
Ideal Digital IIR response (from Matlab)	999	
Actual Digital IIR response (Measured - All Pass)	994	160.1
Measured Digital IIR response (from Audio Precision)	900	

As shown in table, the time constant and the corner frequency of the actual "processed" digital IIR filter are very close to that of the analogue filter, hence in this case we can say that the bilinear transformation gives a good approximation of the analogue filter frequency response. The time constant $\tau_{square\,wave} = 900\mu s$ measured from oscilloscope deviates a little from that of continuous filter and actual discrete IIR filter, since attenuation within the board and influence from reconstruction/anti-aliasing filters are taken into account.
Comparing the frequency response of analogue and actual discrete IIR filters in Matlab gives us:

```
close all;
R = 1000; %Resistor value
C = 10^-6;%Capacitor value
H = tf([1],[R*C,1]); %Find transfer function of RC filter in s domain
w = f(3:1249).*2.*pi;
[mag,phase,wout] = bode(H,w);%return the value at frequcies specified by vector w(rad/s)
mag = 20.*log10(squeeze(mag(1,1,:)));%absolute value of gain(not in dBs)
phase = squeeze(phase(1,1,:));%in degrees
```

Figure 17: Matlab code for generating analog filter response

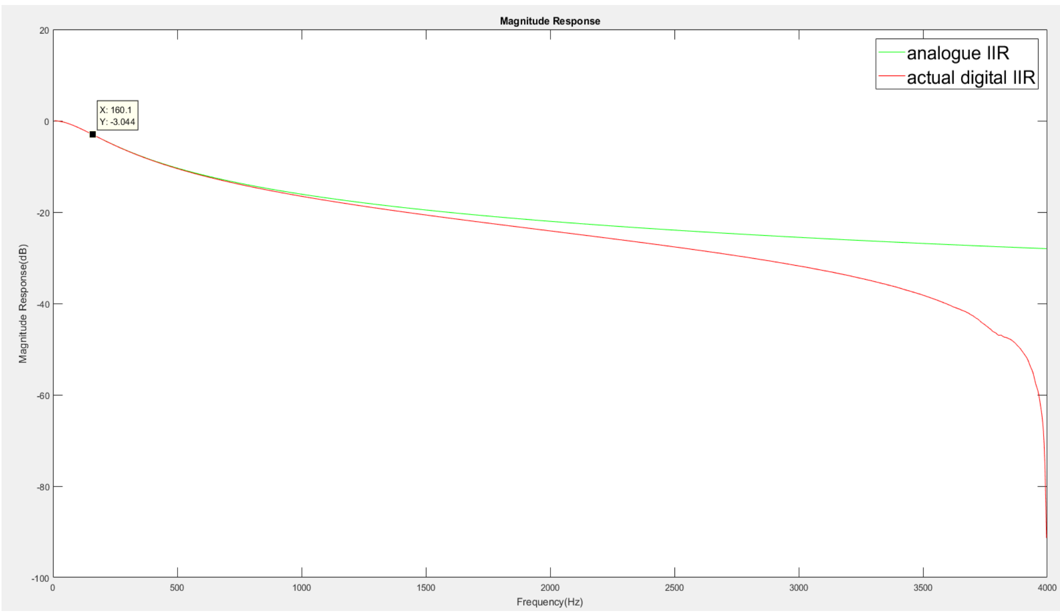

Figure 18: Comparison of Magnitude Response of Analogue and Digital IIR Filters

The gain response of actual discrete IIR filter closely follows that of the analogue IIR filter at low frequency and deviates quite significantly as approaching Nyquist frequency f_N. Intuitively, one has to recognize that Bilinear transform performs non-linear warping of frequency scales, because an approximation is used during series expansion. In this case of first order low pass filter, in the z domain, an extra zero at z = -1 (corresponding to f_N) is introduced in addition to a pole very close to cut-off frequency.

This is straight forward by looking at the bilinear equation, $s = k\frac{z-1}{z+1}$, every pole in s domain will give an extra zero at Nyquist Frequency in z domain, since in the process of rearranging we need to multiply both numerator and denominator by $(z+1)$. The larger the difference between cut-off frequency to sampling frequency, the better the approximation of transform. As a result of this extra zero introduced, discrete response rolls off faster when approaching f_N.

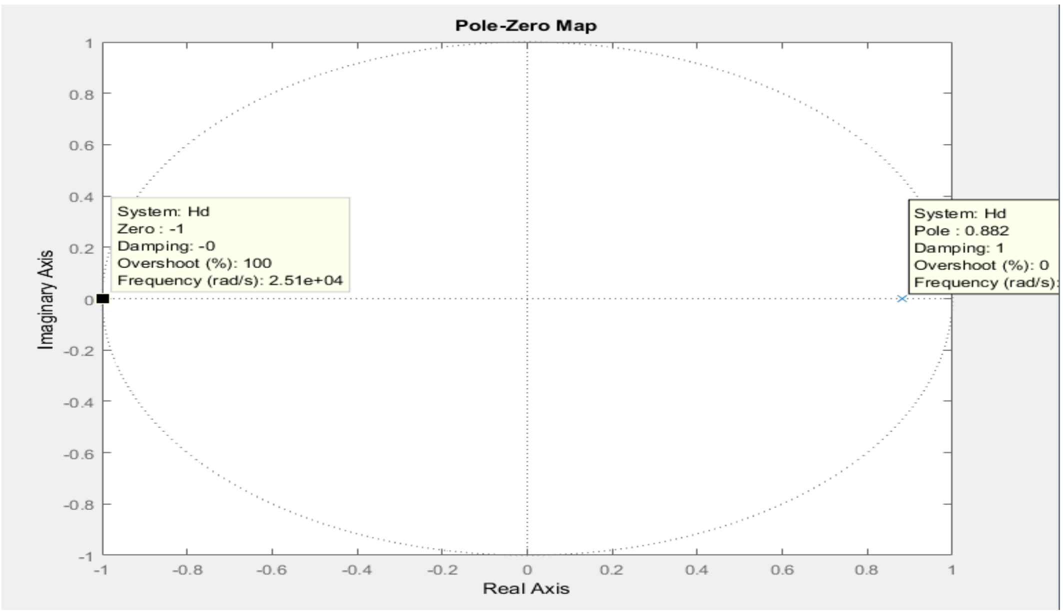

Figure 19: Pole and zero location in z-domain after applying Bilinear Transform

The non-linear mapping is apparent in $w_a = \frac{2}{T_s}\tan(w_d T_s/2)$ as w_d approaches $2\pi f_N$. $\lim_{w_d \to 2\pi f_N} \frac{2}{T_s}\tan(w_d T_s/2) = \infty$. Since gain of analogue filter approaches -∞ dB as w_a approaches ∞. This hence explains the deviation.

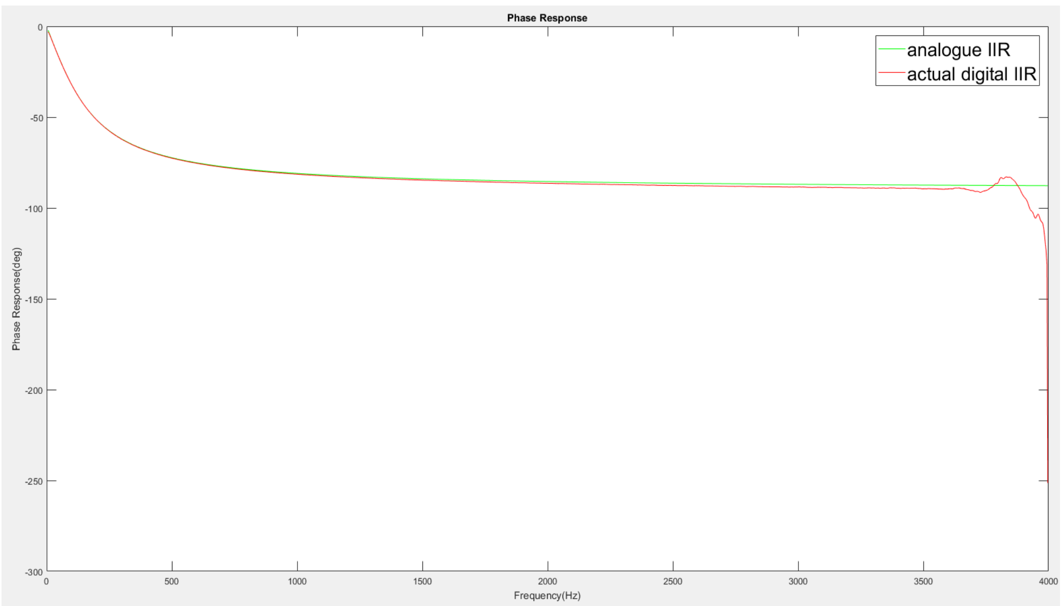

Figure 20: Comparison of Phase Response of Analogue and Digital IIR Filters

The same explanation above can be used to explain the deviation between phase response as f_N is reached.

5 Bandpass Filter: Direct Form II

In this exercise, we will first design a bandpass filter in Matlab to satisfy given specification, then implement it on the DSK using Direct Form II. Measure the frequency response of the digital IIR bandpass filter with APX500 and verify it with the Matlab plot. Finally, we need to find the relationship between number of instruction cycles per sample needed in C implementation and the filter order.

5.1 Filter Specification

Order: 4th
Passband edge frequencies = [270, 450] Hz
Peak-to-Peak Passband Ripple = 0.3dB
Stopband Attenuation: 20dB

5.2 Filter Design

In this case we are designing an elliptic digital bandpass filter using Matlab function *ellip*. It takes input arguments of filter order(n), peak-to-peak passband ripple(Rn), stopband attenuation(Rs) and a vector of passband edge frequencies(w), and returns the numerator and denominator coefficients of transfer function of the filter with order 2n. The coefficients are then automatically saved into declarations of arrays with double precision into the file $bandpass_coef.txt$ and included in the C program. b and a contain the numerator and denominator coefficients(including 1) respectively.

```
close all;
fs = 8000;%sampling frequency
Rn = 0.3;%peak to peak pass band ripple in dB
n = 2;%bandpass filter order = 2n = 4
Rs = 20;%stopband attenuation
w = [270,450].*2./fs;%passband edge frequencies normalized to Nyquist Frequency
[b, a] = ellip (n,Rn,Rs,w ,'bandpass');%return filter coefficients
freqz(b,a,1249,fs);%plot the frequency response
%save filter coefficients into array declarations in bandpass_coef.txt
%with double precision
fileID = fopen('bandpass_coef.txt','w');
formSpec1 = 'double b[]={%.16e';
fprintf(fileID,formSpec1,b(1));
formSpec2 = ', %.16e';
fprintf(fileID,formSpec2,b(2:end));
fprintf(fileID,'};\n');
formSpec3 = 'double a[]={%.16e';
fprintf(fileID,formSpec3,a(1));
fprintf(fileID,formSpec2,a(2:end));
fprintf(fileID,'};\n');
fclose(fileID);
```

Figure 21: Matlab Code for Bandpass Filter Design

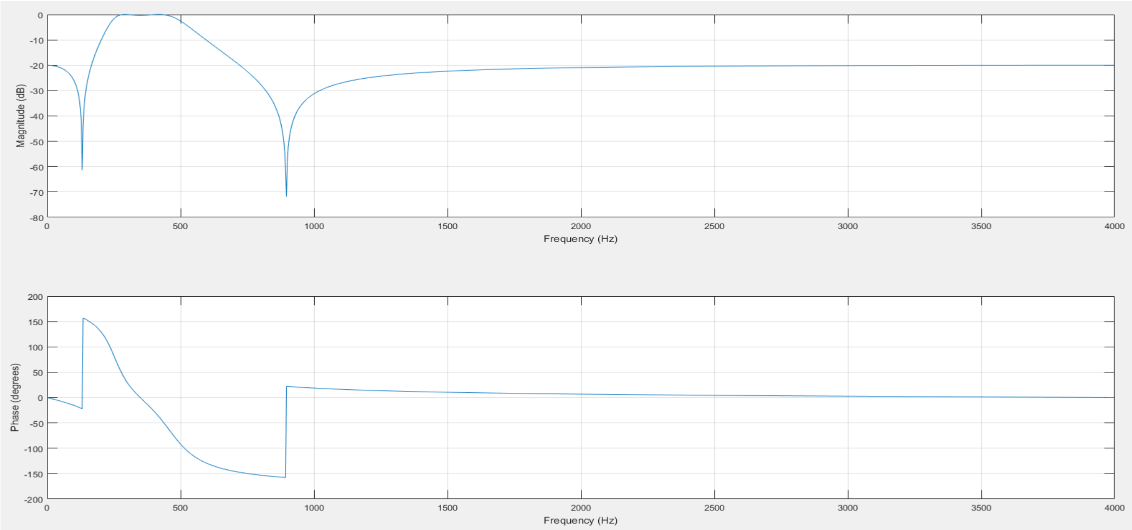

Figure 22: Matlab plot of the Frequency Response

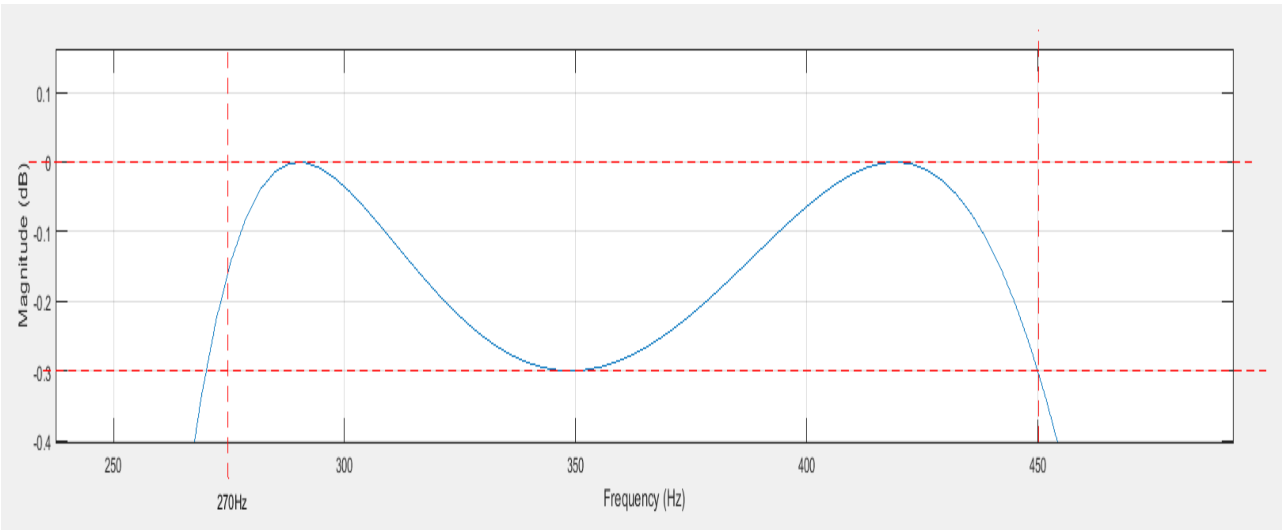

Figure 23: Passband Edge Frequencies and Passband Ripple specifications are satisfied

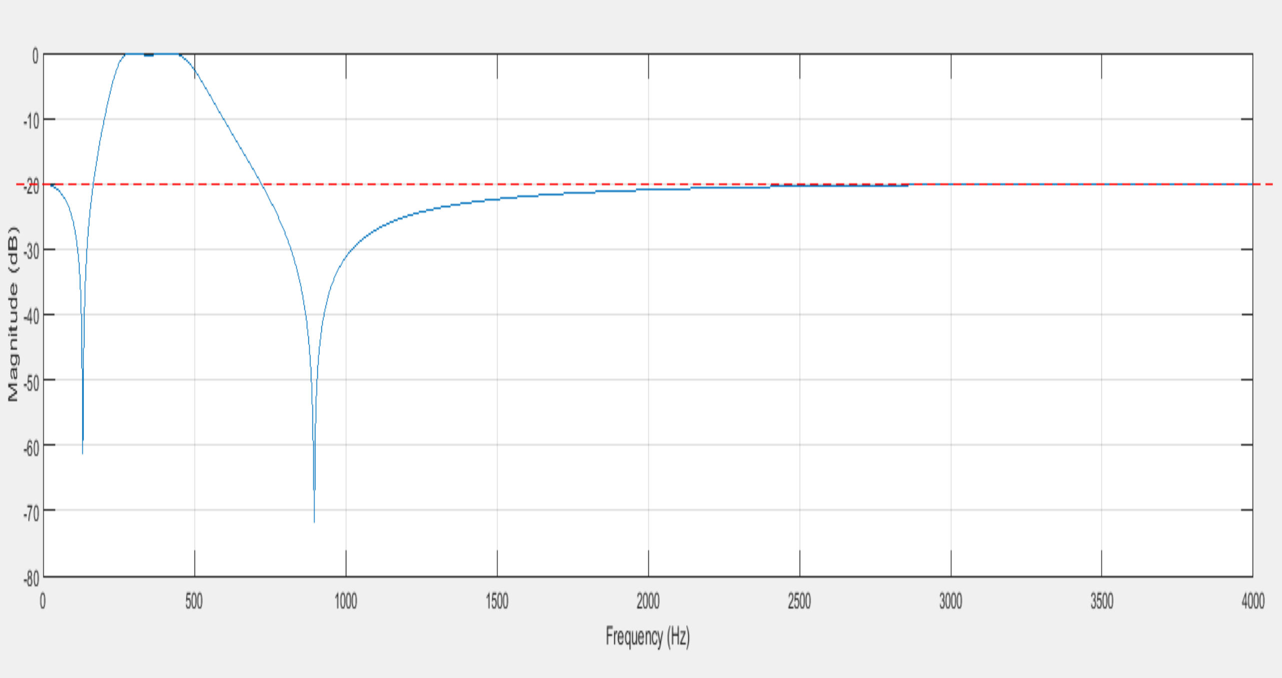

Figure 24: Stopband Attenuation specification is satisfied

An Elliptic filter has ripple in both the pass and stop bands but has the sharpest cut off of all the common filter types. In s domain, similar to Chebyshev filters, an elliptical filter has poles placed on a ellipse, but in addition it also has zeros placed on the imaginary axis adjacent to poles, which contributes a faster roll-off than Chebyshev filters. Nonetheless,introducing zeros add ripple in the stopband and placing poles on an ellipse close to the axis introduces passband ripple. Since the imaginary axis in s plane corresponds to the unit circle in z plane, as we can see from the pole-zero map below, there are four poles in ellipse shape close to unit circle and four zeros on the unit circle outside the pass band pulling the gain to zero.

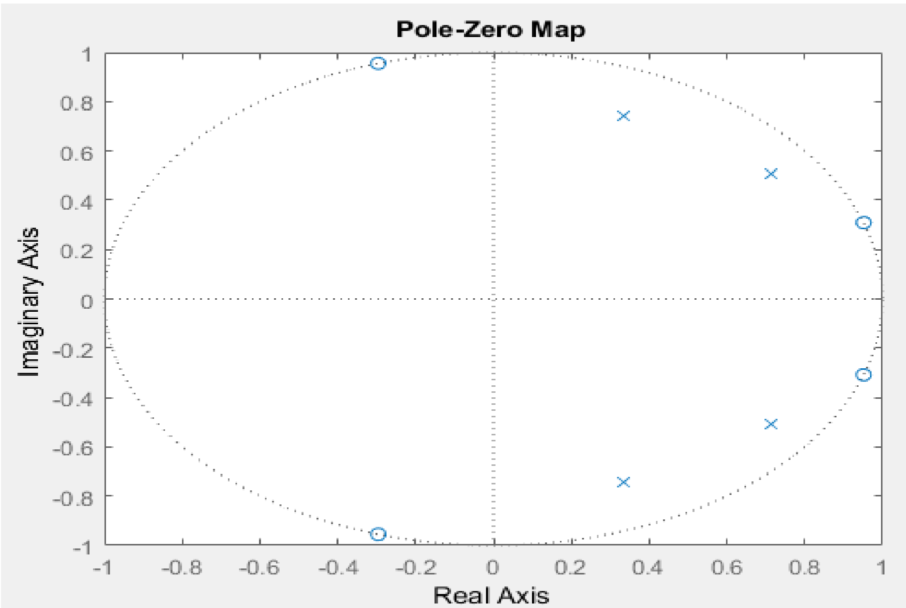

Figure 25: Pole-Zero Pole of the Digital Elliptic Filter

5.3 C Implementation

Now we would implement this filter in the Direct Form II. This form is obtained by swapping around the order of the two parts of IIR transfer function (the polynomial in numerator and denominator) and combining the delay line. In terms of implementation, compared to Direct form I, delay units are reduced by half and we only need one array in our C implementation to store delay line values, which is more efficient.

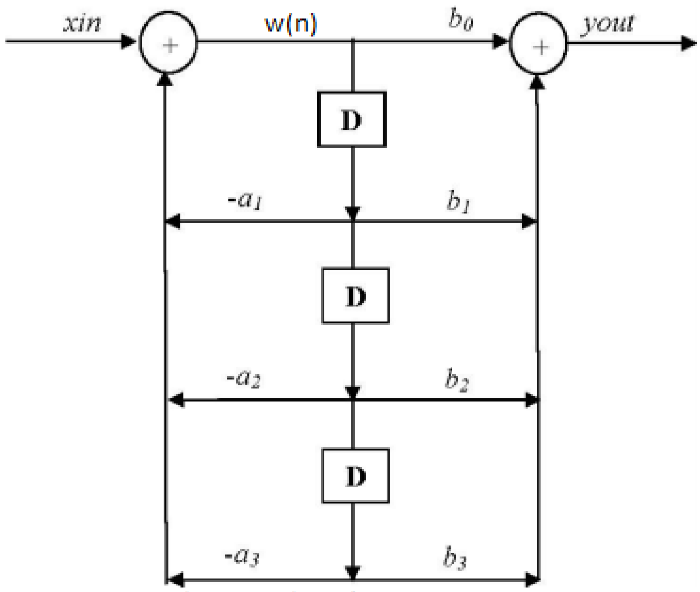

Figure 26: Direct Form II

Define an intermediate state w, we can say that:

$$w(n) = x(n) - a_1 w(n-1) - a_2 w(n-2) - \cdots - a_N w(n-N)$$
$$y(n) = b_0 w(n) + b_1 w(n-1) + b_2 w(n-2) + \cdots + b_M w(n-M)$$

In this case $N = M = 4$ since the filter order is 4. Length of coefficient arrays equals to order + 1. We use *calloc* function again to dynamically allocate the array size as shown in section 4. The algorithm calculates $w(n)$ and $y(n)$ step by step inside the for loop. i iterates from *order* to 1 since the current value of y and w depend on previous values of w. At each iteration $w[i]$ data is shifted to next higher index to represent delay elements. After the for loop is finished, the correct value of $w(n)$ is computed and its product with $b[0]$ is added to the current output. Note that w[0] of code below refers to w(n) of diagram above.

```
void bandpass_direct2_IIR(void){
    int i;
    w[0] = sampin; //write input sample to w[0](w(n))
    for(i=order; i>0; i--){ //for loop to carry out convolution sum
                            //we are not using w[0] yet, so can multiply w[i] with a[i]
                            //and b[i]
        w[0] -= a[i]*w[i]; // w[0](w(n)) =x(n)-a[order]*w[order]-a[order-1]*w[order-1]...
        samp += b[i]*w[i]; //yout = b[order]*w[order]+b[order-1]*w[order-1]...
        w[i] = w[i-1]; //perform delay after the value is used in the calculation
    }
    samp += b[0]*w[0]; //finish output computation since the correct w[0](w(n)) is ready
}
```

Figure 27: Direct Form II

5.4 Results

Measure the frequency response using APX500 audio analyzer. Note that the audio preci

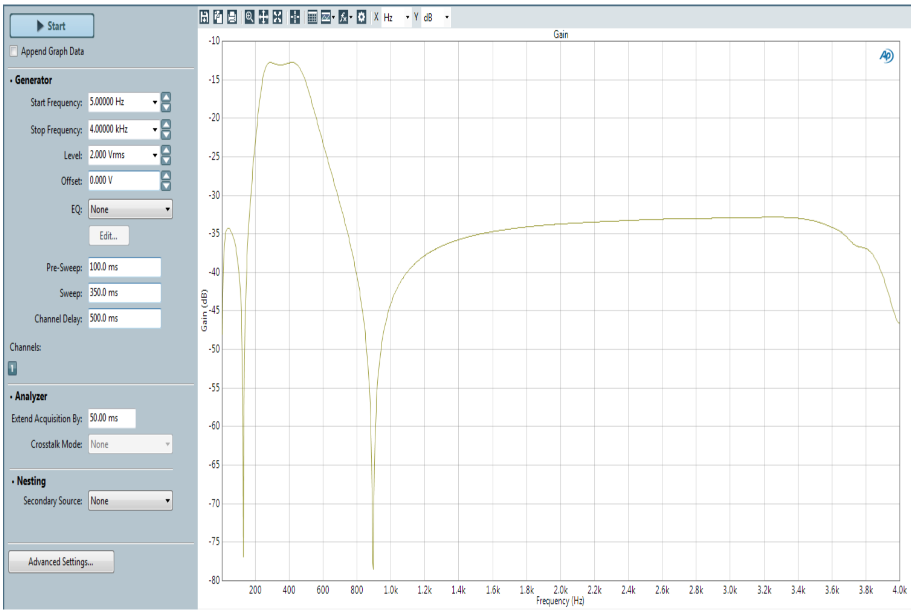

Figure 28: Measured Magnitude Response

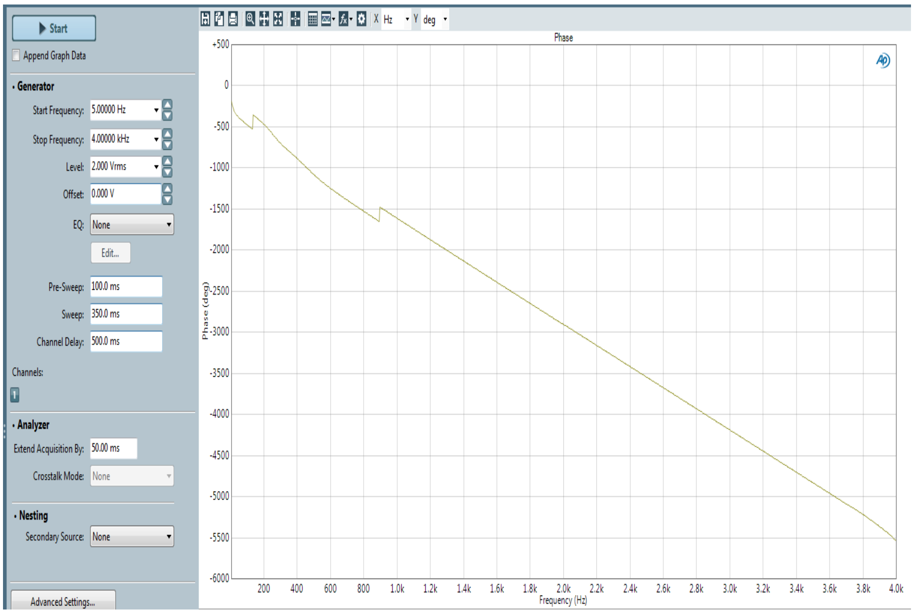

Figure 29: Measured Phase Response

Now we will repeat the steps similar to section 4.3, that is exporting the graph data of the measured response and subtract it by the all pass response for both magnitude and phase to extract the actual frequency response of the elliptic filter alone.

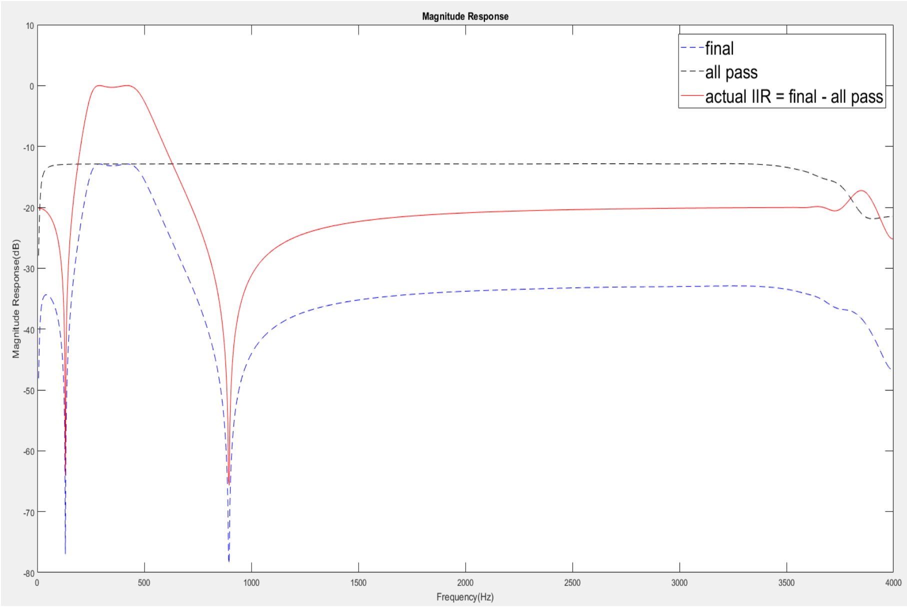

Figure 30: Actual Digital Bandpass Filter Magnitude Response. This is calculated with measured gain response minus all pass response.

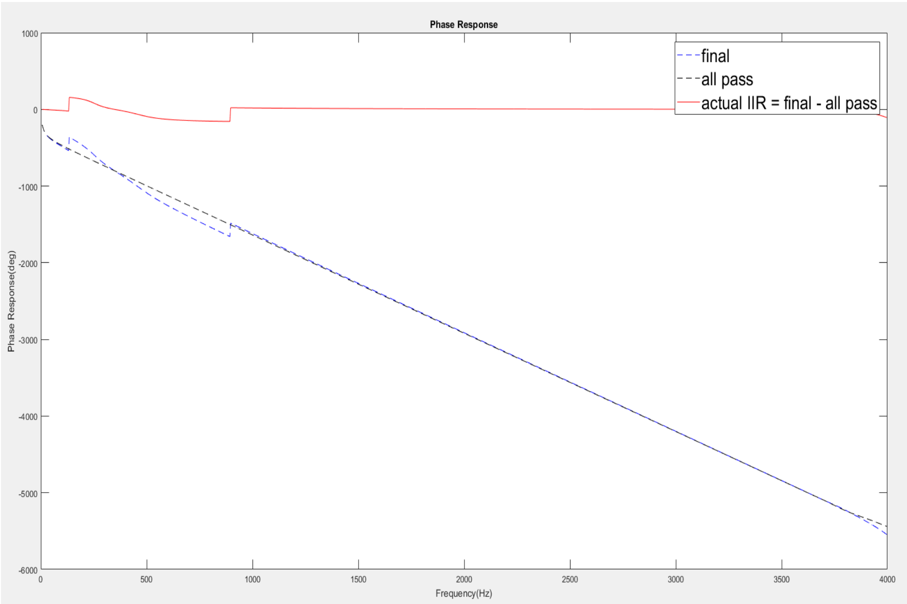

Figure 31: Actual Digital Bandpass Phase Response. This is calculated with measured phase response minus all pass response.

Compare the actual digital bandpass filter with the plot in Matlab:

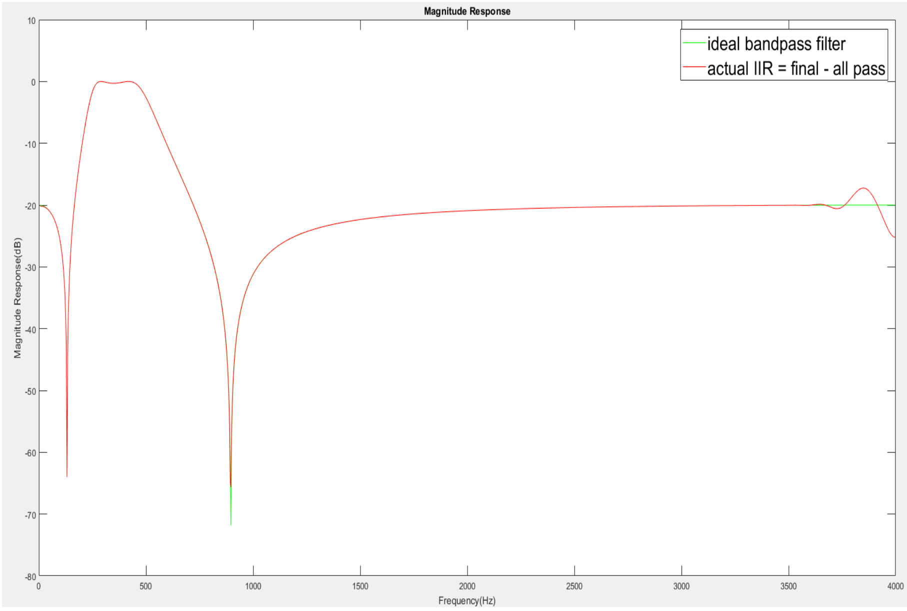

Figure 32: Comparison of Ideal bandpass filter and actual implemented filter

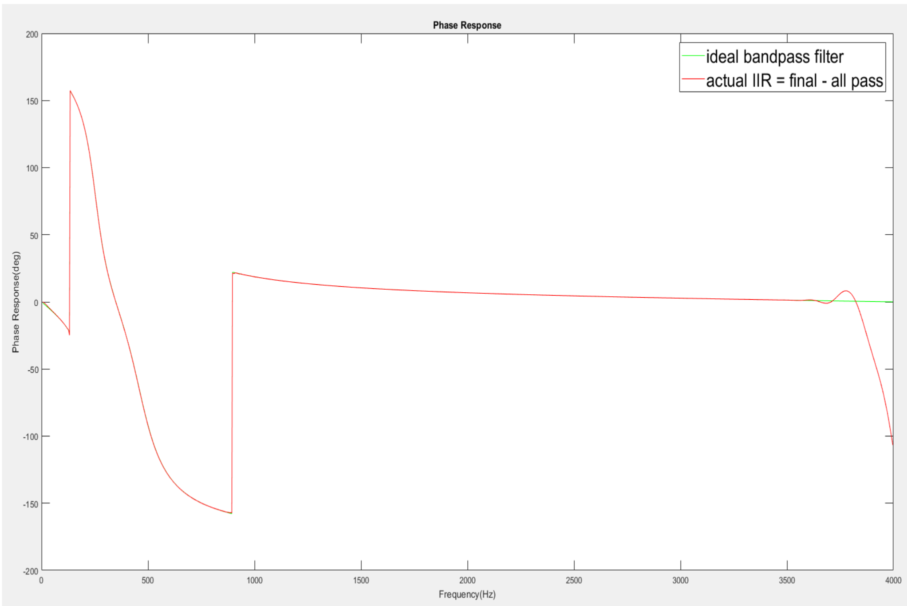

Figure 33: Actual Digital Bandpass Phase Response

The actual bandpass elliptical filter response of order 4 closely tracks the ideal response given by Matlab for except the little peak occurring at high frequencies.This deviation is due to filtering operations, which is explained above.

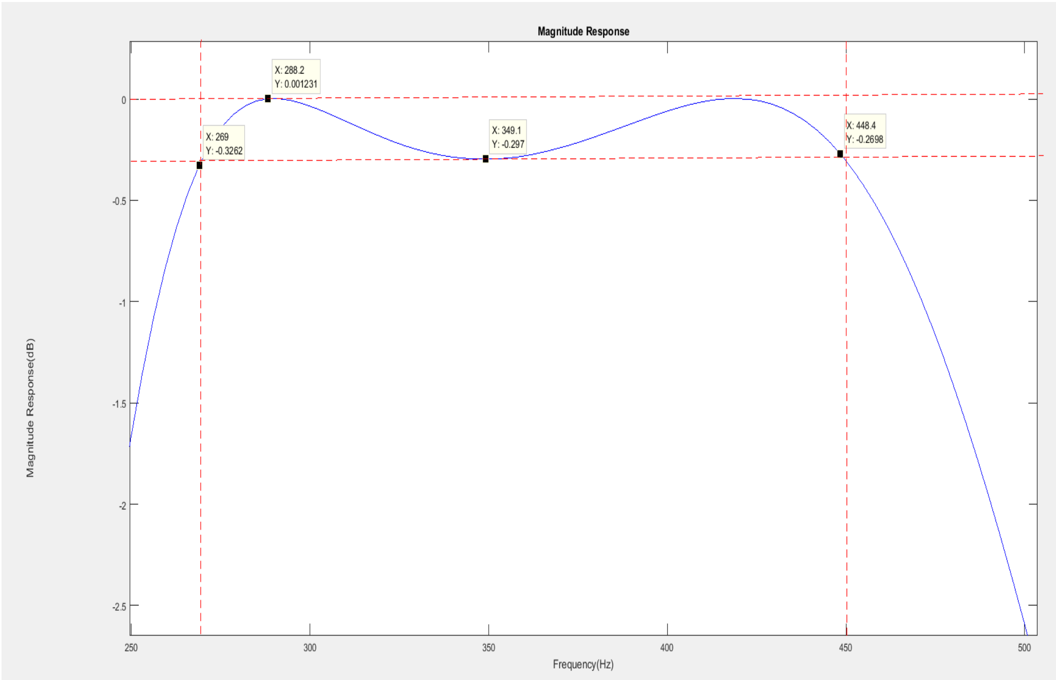

Figure 34: Graph of Actual Bandpass Response

As shown, all specifications as mentioned in the **Filter Specification** are satisfied.

5.5 C Implementation for Direct Form II Transposed Form

The Direct II Transposed form is unchanged in behavior comparing to the traditional Direct II form. The direction for each branch is reversed, branch divisions and branch summations are interchanged and input is swapped for output. This form avoids specific shifting of samples, because shifting is inherent in the calculations of output. This would become more obvious in the following example. In the following diagram, x[n] represents operation of delay elements.

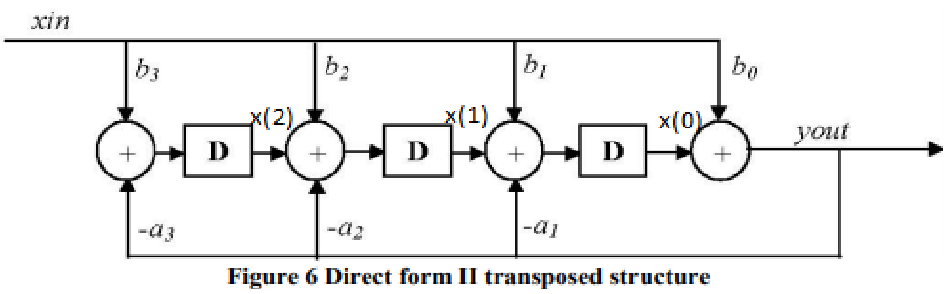

Figure 35: Direct II transposed form

It can be deduced that:

$$
\begin{aligned}
y_{out} &= b_0 X_{in} + x_0 \\
x_0 &= b_1 X_{in} - a_1 y_{out} + x1 \\
x_1 &= b_2 X_{in} - a_2 y_{out} + x2 \\
x_2 &= b_3 X_{in} - a_3 y_{out}
\end{aligned}
$$

Hence in C implementation, at each iteration of *for* loop a sample within buffer x[m-1] is updated with previous buffer value x[m] and coefficients b[m], a[m] following formula:

$$x[m-1] = X_{in} * b[m] - y_{out} * a[m] + x[m]$$

Outside the loop, the x[order-1] would be calculated individually, because it is the first value in the buffer that is not weighted by previous buffer values.

```
void bandpass_direct2_transposed_IIR(void){
    int m;
    samp = b[0]*sampin + x[0];//calculate current sample output
    for(m =1; m < order; m++){ ;//no circular buffer or shifting needed
        x[m-1] = sampin*b[m] - samp*a[m] + x[m];//Update each buffer value, except x[order-1]
                                            //with previous buffer values and coefficients
                                            //a and b
    }
    x[order-1] = sampin*b[order] - samp*a[order];//x[order-1] is not influenced by other values in x
                                                //hence computed individually
}
```

Figure 36: Direct II transposed form Implementation

5.6 Results for Direct Form II Transposed Form

Measure the frequency response using APX500 audio analyzer:

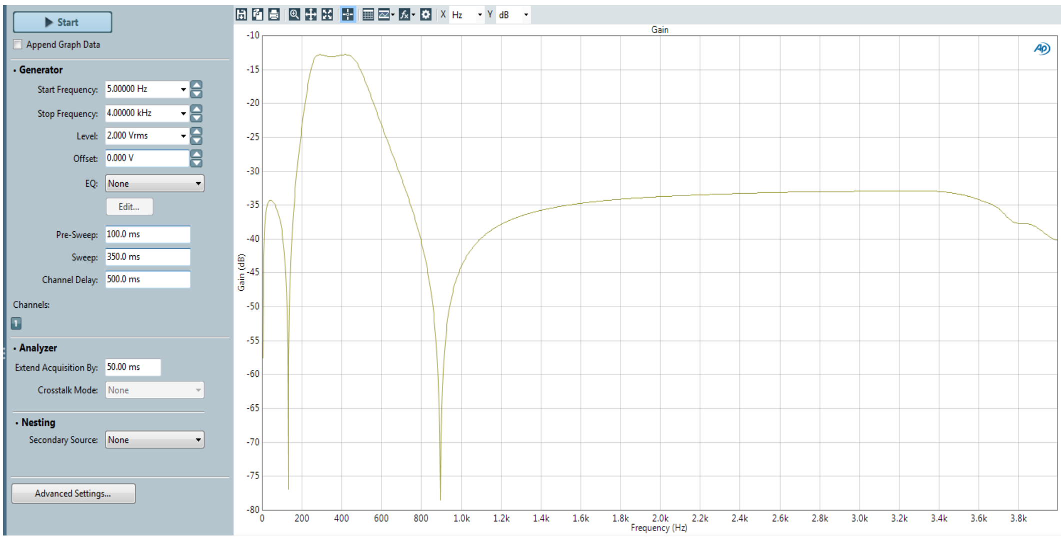

Figure 37: Direct II transposed Magnitude response

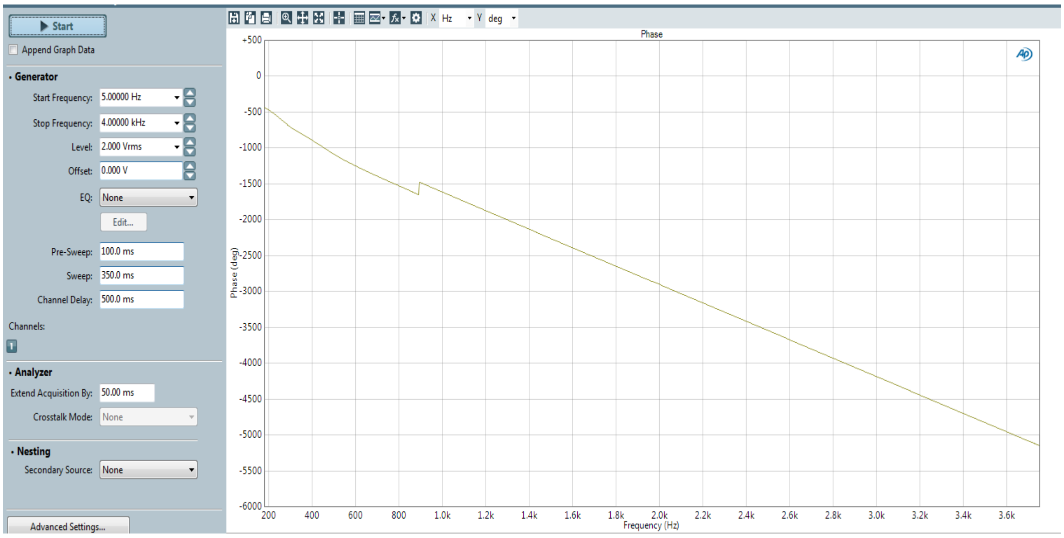

Figure 38: Direct II transposed Phase response

Same as before, measured data from Audio Precision are extracted and subtract all pass response to obtain actual discrete IIR filter response. The actual response is then compared with ideal response from Matlab.

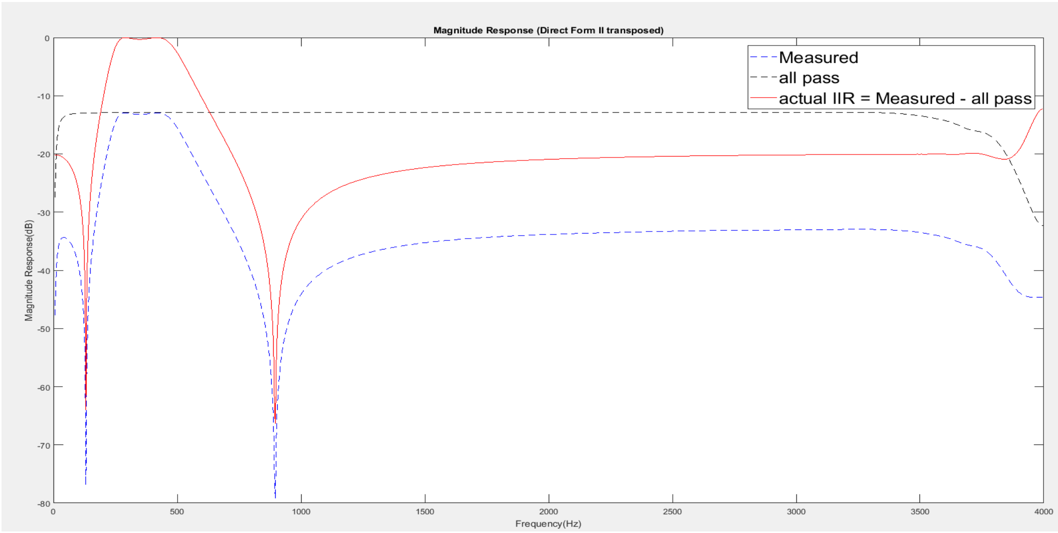

Figure 39: Actual discrete IIR Bandpass filter Magnitude response. Direct form II Transposed Implementation

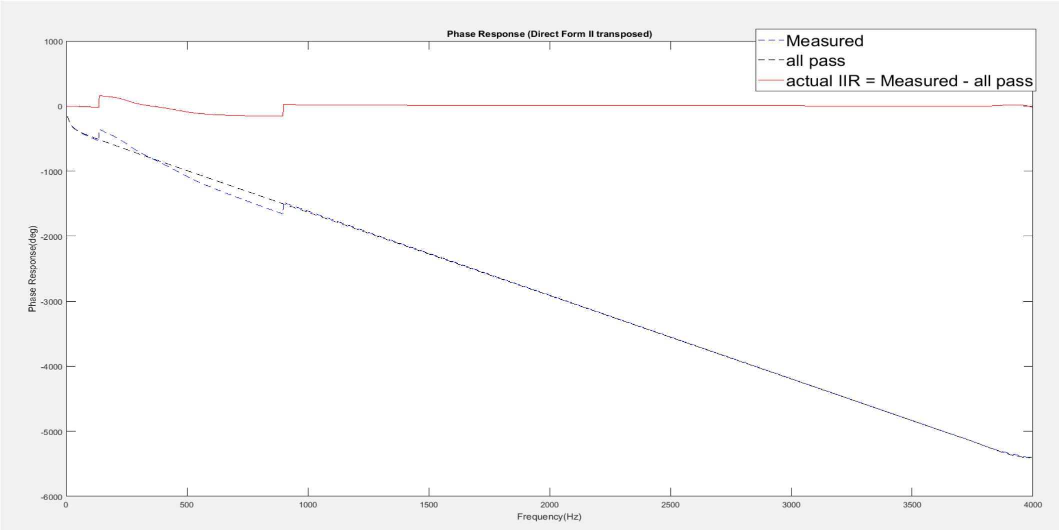

Figure 40: Actual discrete IIR Bandpass filter Phase response. Direct form II Transposed Implementation

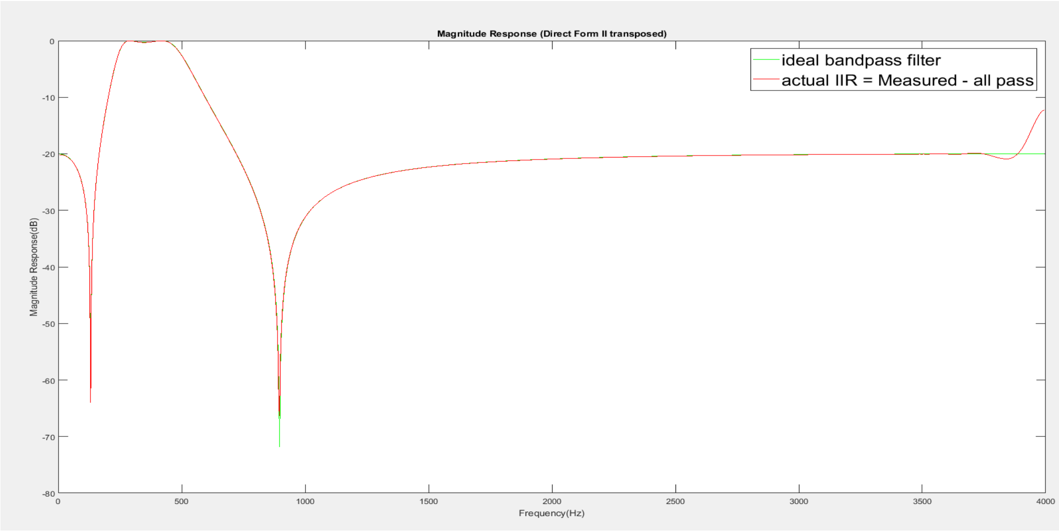

Figure 41: Comparison of Ideal bandpass filter and actual implemented filter

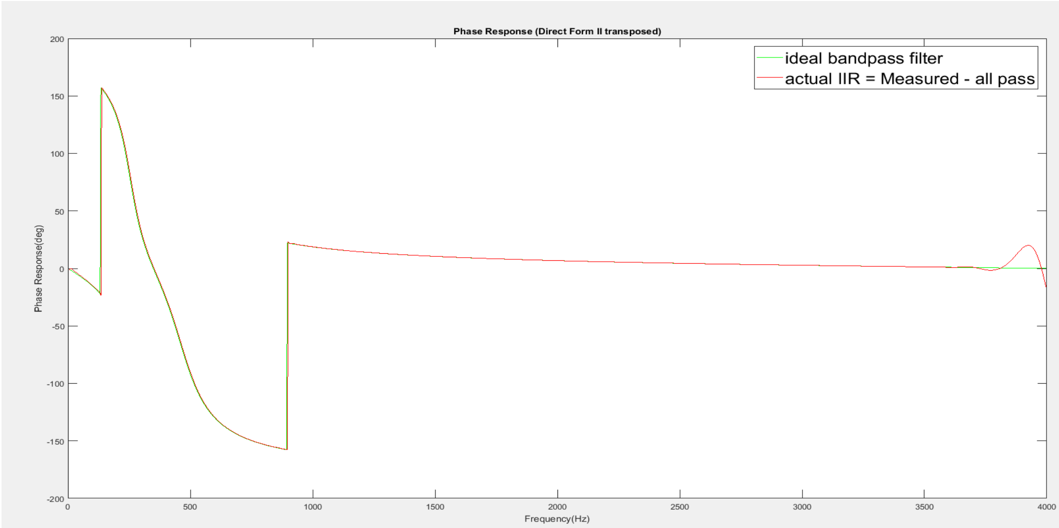

Figure 42: Comparison of Ideal bandpass filter and actual implemented filter

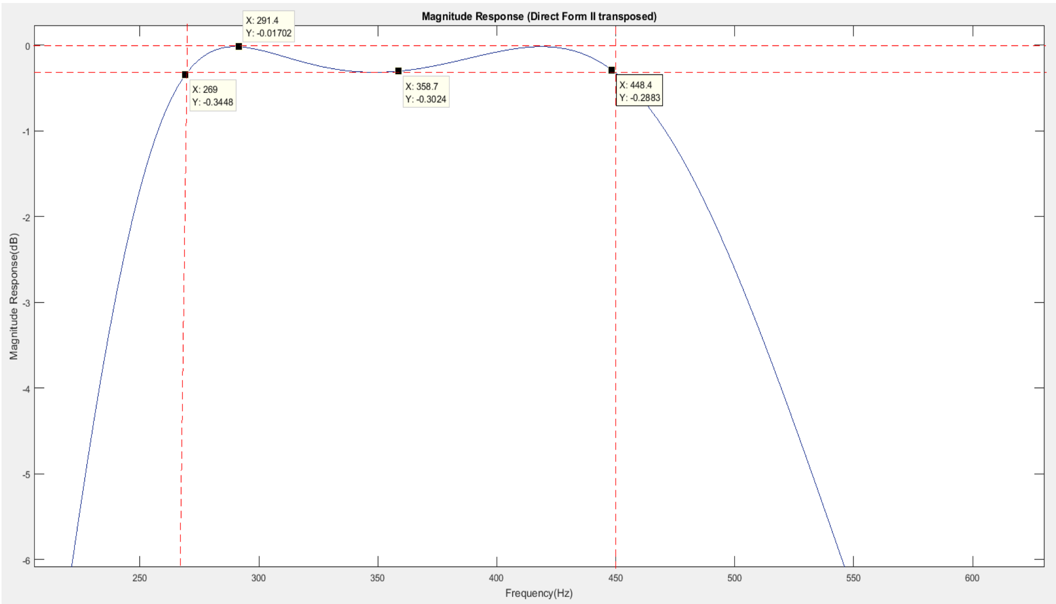

Figure 43: All specifications are satisfied

As shown, the gain and phase response for Direct Form II Transposed Form are almost identical to the results for traditional Direct II form, hence proving the unchanged behavior. The next section would explore the algorithm performance and efficiency difference of different implementations.

6 Performance Comparison of 2 Direct Form Filters

In this section we use the profiling clock method to measure the relationship between instruction cycles and the filter order. Two breakpoints are set at calls to $mono\_read\_16Bit()$ and $mono\_write\_16Bit()$ respectively so that the count only includes the filtering operations. Measurements are done for both non-transposed and transposed Direct Form II filters and for both no optimization and -o2 optimization level. Measurements are taken several times to find the average cycles needed for a given order filter under certain optimization level.

-	No optimization		-o2 optimization	
Order	Direct form 2	Direct form 2 Transposed	Direct form 2	Direct form 2 Transposed
2	321.5	229.5	286.25	240.5
4	491.75	349.25	357.25	252.25
6	661.5	469.5	433.75	262.5
8	831.25	589.5	509.5	272.25
10	1001.75	709.75	585.75	282.5
12	1171.5	829.25	661.75	293
14	1341.25	949.5	737.5	302.25
16	1511.5	1069.5	813	312.5
18	1681.75	1189.5	889.5	322.25
20	1851.5	1309.75	965	332.75
40	3551.5	2509.75	1725.75	432.25
50	4401.5	3109.5	2125.5	482.75
60	5259.5	3709.25	2529.5	3348.42
64	5627.25	3949	2677.25	1116.75
70	6150	4309.25	2883	582.75
80	7000.5	4909.5	3254.5	632.75

The results are then plotted in Matlab as shown below:

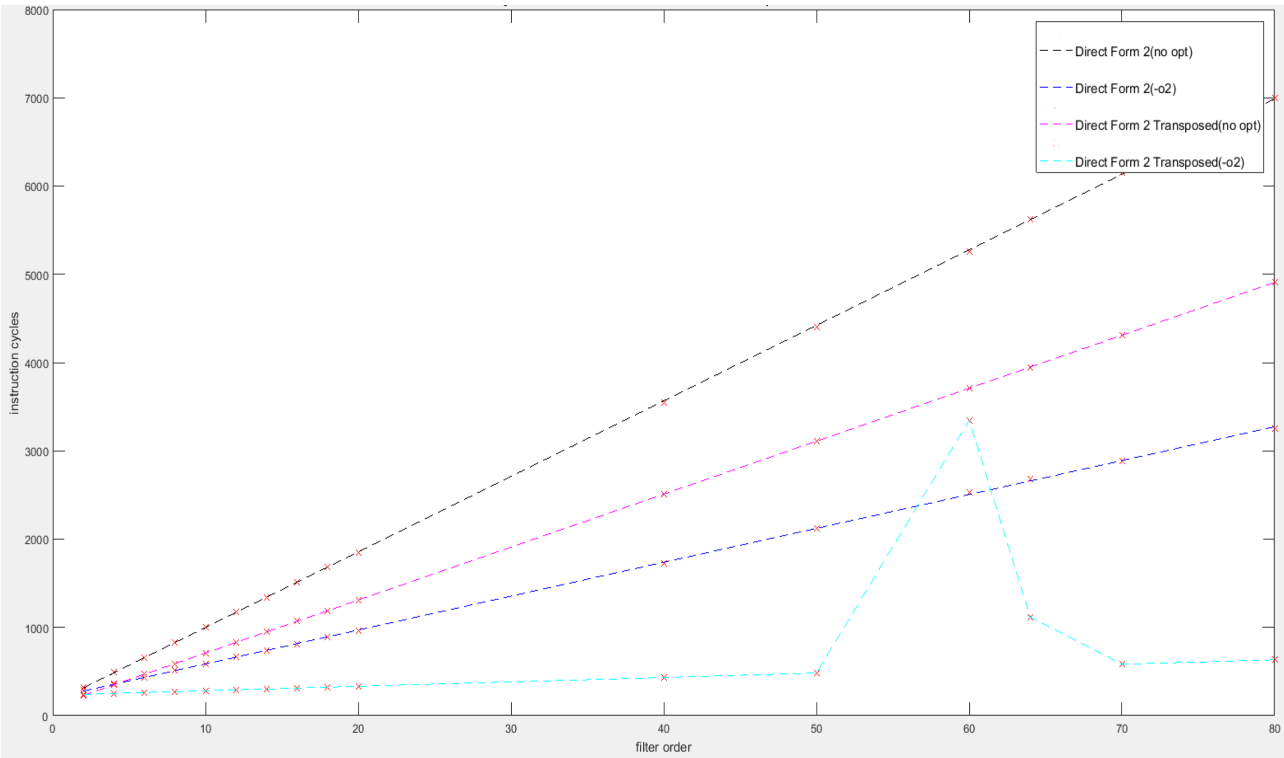

Figure 44: Plot of Instruction Cycles against Filter Order

As we can see above, all the curves have linear trend except for direct form 2 transposed filter under the -o2 optimization.The cycles needed for the transposed form under -o2 of order 60 and 64 jump out the linear trend of the other data points. This might be due to the way the compiler optimizes the code. Let's ignore the exception points on the 4th curve for the moment and only look at overall performance. At lower orders there are only small differences between 4 curves, cycles needed are almost the same for filter with order 2. Differences between performances increase as order increases. Overall, direct form 2 transposed is faster than the non-transposed form for both optimisations since in transposed form the shifting of the samples is inherent in the calculations for the output. It is also straight forward from

the prospective of the C implementation, where the non-transposed code contains a for loop with three instructions whereas the transposed contains only one. However, the transposed form implementation takes greater advantage of the optimization since difference between *noopt* and $-o2$ lines are greater that of the non-transposed form given the same order.
The expressions of 4 best-fit straight lines are given below:

-	Expressions
Direct form 2(no opt)	$85.5563n + 143.8626$
Direct form 2(-o2)	$38.3912n + 202.3319$
Direct form 2 Transposed(no opt)	$59.9973n + 109.5336$
Direct form 2 Transposed(-o2)	$5.0097n + 232.1316$

Where the intercepts represent code overhead and the gradients represent the number of extra cycles raised by an extra coefficient from an increase in the filter order. Optimized codes have lower gradients and higher overheads in each form, while transposed form codes have lower gradients under each optimization level. The decrease in gradients from no optimization to -o2 optimization is about 55% for non-transposed form while it is 91% for transposed form.
The difference in the degree of improvement due to optimization between two implementations draws our attention to how the code is optimised under o2 optimization. o2 optimization takes great advantage of software pipelining and parallel instructions as it perform loop optimizations and loop unrolling. Parallel instructions that have no data or control dependencies with each other, which can be executed at the same time by utilizing all processing units. The implementation of direct form II transposed contains more parallel instruction. This is because for each stage of x(the value after each delay element), the operations with coefficients and current input/output can be calculated first. For example, $a_i x_{in}$ and $b_i y_{out}$ can be calculated first for x_{i-1}(the value after i-th delay element) before the correct value of x_i is available.
Also, the overhead of optimised code is larger than the original code for each implementation. The reason might be that the optimizer is creating a more efficient set-up in order to reduce the number of loop iterations. We could see the optimisation result by comparing the assembly code for the loop section in the Appendix.

7 Appendix

- C Implementation

```c
/*************************************************************************************
                               DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
                                                                IMPERIAL COLLEGE LONDON

                                       EE 3.19: Real Time Digital Signal Processing
                                               Dr Paul Mitcheson and Daniel Harvey

                                                          LAB 5: Real-Time Implementation of IIR Filters

                                             ********* I N T I O. C **********

  Demonstrates inputing and outputing data from the DSK's audio port using interrupts.

 *************************************************************************************
                                 Updated for use on 6713 DSK by Danny Harvey: May-Aug 2006
                                Updated for CCS V4 Sept 10
 ************************************************************************************/
/*
 *        You should modify the code so that interrupts are used to service the
 *  audio port.
 */
/***************************** Pre-processor statements ******************************/

#include <stdlib.h>
//  Included so program can make use of DSP/BIOS configuration tool.
#include "dsp_bios_cfg.h"

/* The file dsk6713.h must be included in every program that uses the BSL.  This
   example also includes dsk6713_aic23.h because it uses the
   AIC23 codec module (audio interface). */
#include "dsk6713.h"
#include "dsk6713_aic23.h"

// math library (trig functions)
#include <math.h>

// Some functions to help with writing/reading the audio ports when using interrupts.
#include <helper_functions_ISR.h>

//FIR filter coefficients
#include "single_pole_coef.txt"
//#include "bandpass_coef.txt"
/******************************** Global declarations ********************************/

/* Audio port configuration settings: these values set registers in the AIC23 audio
   interface to configure it. See TI doc SLWS106D 3-3 to 3-10 for more info. */
DSK6713_AIC23_Config Config = { \
                         /**********************************************************************/
                         /*   REGISTER                    FUNCTION                                SETTINGS         */
                         /**********************************************************************/\
    0x0017,  /* 0 LEFTINVOL  Left line input channel volume  0dB                   */\
    0x0017,  /* 1 RIGHTINVOL Right line input channel volume 0dB                   */\
    0x01f9,  /* 2 LEFTHPVOL  Left channel headphone volume   0dB                   */\
    0x01f9,  /* 3 RIGHTHPVOL Right channel headphone volume  0dB                   */\
    0x0011,  /* 4 ANAPATH    Analog audio path control       DAC on, Mic boost 20dB*/\
    0x0000,  /* 5 DIGPATH    Digital audio path control      All Filters off       */\
    0x0000,  /* 6 DPOWERDOWN Power down control              All Hardware on       */\
    0x0043,  /* 7 DIGIF      Digital audio interface format  16 bit                */\
    0x008d,  /* 8 SAMPLERATE Sample rate control             8 KHZ                 */\
    0x0001   /* 9 DIGACT     Digital interface activation    On                    */\
                         /**********************************************************************/
};


// Codec handle:- a variable used to identify audio interface
DSK6713_AIC23_CodecHandle H_Codec;

/******************************** Global Variables *******************************/



short sampin;
double samp;
double *w,*v,*x;//delcare pointers of type double
int order;//filter order
/* Audio channel gain values, calculated to be less than maximum acceptable value. */



 /******************************** Function prototypes ******************************/
void init_hardware(void);
void init_HWI(void);
```

```
void ISR_AIC (void);
void single_pole_IIR(void);
void bandpass_direct2_IIR(void);
void bandpass_direct2_transposed_IIR(void);
/********************************** Main routine ************************************/
void main(){


        // initialize board and the audio port
  init_hardware();
   // initialize the look-up table

  /* initialize hardware interrupts */
  init_HWI();

  //dynamically allocat array sizes to store input and output samples
  order = sizeof(a)/sizeof(a[0]) -1;
  w = (double *) calloc(order+1, sizeof(double));//dynamic memory, order+1 elements, each of size double
  v = (double *) calloc(order+1, sizeof(double));
  x = (double *) calloc(order, sizeof(double));

  /* loop indefinitely, waiting for interrupts */
  while(1)
  {};

}

/********************************** init_hardware() *********************************/
void init_hardware()
{
    // Initialize the board support library, must be called first
    DSK6713_init();

    // Start the AIC23 codec using the settings defined above in config
    H_Codec = DSK6713_AIC23_openCodec(0, &Config);

        /* Function below sets the number of bits in word used by MSBSP (serial port) for
        receives from AIC23 (audio port). We are using a 32 bit packet containing two
        16 bit numbers hence 32BIT is set for  receive */
        MCBSP_FSETS(RCR1, RWDLEN1, 32BIT);

        /* Configures interrupt to activate on each consecutive available 32 bits
        from Audio port hence an interrupt is generated for each L & R sample pair */
        MCBSP_FSETS(SPCR1, RINTM, FRM);

        /* These commands do the same thing as above but applied to data transfers to
        the audio port */
        MCBSP_FSETS(XCR1, XWDLEN1, 32BIT);
        MCBSP_FSETS(SPCR1, XINTM, FRM);


}

/********************************** init_HWI() **************************************/
void init_HWI(void)
{
        IRQ_globalDisable();                        // Globally disables interrupts
        IRQ_nmiEnable();                                // Enables the NMI interrupt (used by the debugger)
        IRQ_map(IRQ_EVT_RINT1,4);                // Maps an event to a physical interrupt
        IRQ_enable(IRQ_EVT_RINT1);                // Enables the event
        IRQ_globalEnable();                                // Globally enables interrupts

}

/******************** WRITE YOUR INTERRUPT SERVICE ROUTINE HERE***********************/

void ISR_AIC (void)
{
            samp = 0;//reset output
                sampin = mono_read_16Bit();
                single_pole_IIR();
                //bandpass_direct2_IIR();
                //bandpass_direct2_transposed_IIR();
                //mono_write_16Bit(mono_read_16Bit());
                mono_write_16Bit(samp);

}


void single_pole_IIR(void){
        int k;
        samp = b[0]*sampin;                                  //calculation with b[0] done individually
                                                                                   //so within for loop can multiply v[k]with b[k]
                                                                                   //and w[k] with a[k]
        for(k = order; k>0; k--){
                samp += v[k]*b[k] - w[k]*a[k];//perform convolution sum following
                                                                                   //IIR difference equation
```

```c
                w[k] = w[k-1];                                          //perform delay for input values stored in v
                v[k] = v[k-1];                                          //perform delay for output values stored in w
        }
        v[1] = sampin;                                                  //update first delayed element v[1], or x(n-1) for next iteration
        w[1] = samp;                                                   //update w[1], or y(n-1) for next iteration
}

void bandpass_direct2_IIR(void){
        int i;
        w[0] = sampin; //write input sample to w[0](w(n))
        for(i=order; i>0; i--){ //for loop to carry out convolution sum
                                                        //we are not using w[0] yet, so can multiply w[i] with a[i]
                                                        //and b[i]
                w[0] -= a[i]*w[i]; // w[0](w(n)) =x(n)-a[order]*w[order]-a[order-1]*w[order-1]...
                samp += b[i]*w[i]; //yout = b[order]*w[order]+b[order-1]*w[order-1]...
                w[i] = w[i-1]; //perform delay after the value is used in the calculation
        }
        samp += b[0]*w[0]; //finish output computation since the correct w[0](w(n)) is ready
}


void bandpass_direct2_transposed_IIR(void){
        int m;
        samp = b[0]*sampin + x[0];//calculate current sample output
        for(m =1; m < order; m++){ //no circular buffer or shifting needed
                x[m-1] = sampin*b[m] - samp*a[m] + x[m];//Update each buffer value, except x[order-1]
                                                                                //with previous buffer values and coefficients
                                                                                //a and b
        }
        x[order-1] = sampin*b[order] - samp*a[order];//x[order-1] is not influenced by other values in x
                                                                                        //hence computed individually
}
```

- Single-pole Filter Design and Graph Plotting

```
close all;
R = 1000; %Resistor value
C = 10^-6;%Capacitor value
fs = 8000; %Sampling frequency
H = tf([1],[R*C,1]); %Find transfer function of RC filter
Hd = c2d(H,1/fs,'tustin') %Complete Tustin transform

[num,den] = tfdata(Hd); %Find coefficients
num = cell2mat(num)
[den] = cell2mat(den)
%{
damp(Hd);
%save iir_coef.txt num den -ascii -double -tabs
figure;
step(H,'-',Hd,'--');
damp(Hd);
%}
figure;
pzmap(Hd)

[h,f] = freqz(num,den,1249,fs);
%freqz(num,den,1249,fs)

gaindirect = cell2mat(directgain(4,2));
phasedirect = cell2mat(directphase(4,2));
gainy = cell2mat(Gain(4,2));
phasey = cell2mat(Phase(4,2));

figure;
plot(f(3:1249),gainy,'--b');
hold on;
plot(f(3:1249),gaindirect,'--k');
hold on;
plot(f(3:1249),gainy-gaindirect,'-r');
%hold on;
%plot(f(3:1250),gainy,'-r');
legend({'final','all pass','actual IIR = final - all pass'},'FontSize',20);
title('Magnitude Response');
xlabel('Frequency(Hz)');
ylabel('Magnitude Response(dB)');
hold off;
figure;
plot(f(3:1249),phasey,'--b');
hold on;
plot(f(3:1249),phasedirect,'--k');
hold on;
plot(f(3:1249),phasey-phasedirect,'-r');
%hold on;
%plot(f(3:1250),phasey,'-r');
title('Phase Response');
xlabel('Frequency(Hz)');
ylabel('Phase Response(deg)');
hold off;
Hd
%{
%save fir_coef.txt b -ascii  -double -tabs; %save to fir_coef.txt
fileID = fopen('fir_coef.txt','w');
```

```
formSpec1 = 'double b[]={%.16e';
fprintf(fileID,formSpec1,b(1));
formSpec2 = ', %.16e';
fprintf(fileID,formSpec2,b(2:end));
fprintf(fileID,'};\n');
fclose(fileID);
%}

w = f(3:1249).*2.*pi;
[mag,phase,wout] = bode(H,w);
mag = squeeze(mag(1,1,:));
phase = squeeze(phase(1,1,:));
figure;
plot(f(3:1249),20.*log10(mag),'-g');
hold on;
plot(f(3:1249),gainy-gaindirect,'-r');
legend({'analogue IIR','actual digital IIR'},'FontSize',20);
title('Magnitude Response');
xlabel('Frequency(Hz)');
ylabel('Magnitude Response(dB)');
hold off;
figure;
plot(f(3:1249),phase,'-g');
hold on;
plot(f(3:1249),phasey-phasedirect,'-r');
legend({'analogue IIR','actual digital IIR'},'FontSize',20);
title('Phase Response');
xlabel('Frequency(Hz)');
ylabel('Phase Response(deg)');
hold off;
figure;
bode(H);


figure;
plot(f(3:1249),20.*log10(mag),'-g');
hold on;
plot(f(3:1249),20.*log10(abs(h(3:1249))),'-r');
legend({'analogue IIR','ideal digital IIR'},'FontSize',20);
title('Magnitude Response');
xlabel('Frequency(Hz)');
ylabel('Magnitude Response(dB)');
hold off;

figure;
plot(f(3:1249),phase,'-g');
hold on;
plot(f(3:1249),unwrap(angle(h(3:1249))).*180./pi,'-r');
legend({'analogue filter','ideal digital IIR'},'FontSize',20);
title('Phase Response');
xlabel('Frequency(Hz)');
ylabel('Phase Response(deg)');
hold off;
```

- Bandpass Filter Design and Graph Plotting

```
close all;
fs = 8000;
Rn = 0.3;
n = 10;
Rs = 20;
w = [270,450].*2./fs;
[b, a] = ellip(n,Rn,Rs,w,'bandpass');

%{
C = reshape([b,a],length(b),2)'
%dlmwrite('coef.txt' ,C, ',')
figure;
freqz(b,a,1029,fs);
figure;
H = tf(b,a,1/fs);
damp(H)

pzmap(H);
figure;
bode (H);
%}
[h,f]=freqz(b,a,1249,fs);
freqz(b,a,1249,fs);
fileID = fopen('bandpass_coef.txt','w');
formSpec1 = 'double b[]={%.16e';
fprintf(fileID,formSpec1,b(1));
formSpec2 = ', %.16e';
fprintf(fileID,formSpec2,b(2:end));
fprintf(fileID,'};\n');
formSpec3 = 'double a[]={%.16e';
fprintf(fileID,formSpec3,a(1));
fprintf(fileID,formSpec2,a(2:end));
fprintf(fileID,'};\n');
fclose(fileID);

gaindirect = cell2mat(directgain(4,2));
phasedirect = cell2mat(directphase(4,2));
gainy = cell2mat(Gain(4,2));
phasey = cell2mat(Phase(4,2));

figure;
plot(f(3:1249),gainy,'--b');
hold on;
plot(f(3:1249),gaindirect,'--k');
hold on;
plot(f(3:1249),gainy-gaindirect,'-r');
legend({'final','all pass','actual IIR = final - all pass'},'FontSize',20);
title('Magnitude Response');
xlabel('Frequency(Hz)');
ylabel('Magnitude Response(dB)');
hold off;

figure;
plot(f(3:1249),phasey,'--b');
hold on;
plot(f(3:1249),phasedirect,'--k');
hold on;
```

```
plot(f(3:1249),phasey-phasedirect,'-r');
legend({'final','all pass','actual IIR = final - all pass'},'FontSize',20);
title('Phase Response');
xlabel('Frequency(Hz)');
ylabel('Phase Response(deg)');
hold off;

figure;
plot(f(3:1249),20.*log10(abs(h(3:1249))),'-g');
hold on;
plot(f(3:1249),gainy-gaindirect,'-r');
legend({'ideal bandpass filter','actual IIR = final - all pass'},'FontSize',20);
title('Magnitude Response');
xlabel('Frequency(Hz)');
ylabel('Magnitude Response(dB)');
hold off;

figure;
plot(f(3:1249),unwrap(angle(h(3:1249))).*180./pi,'-g');
hold on;
plot(f(3:1249),phasey-phasedirect,'-r');
legend({'ideal bandpass filter','actual IIR = final - all pass'},'FontSize',20);
title('Phase Response');
xlabel('Frequency(Hz)');
ylabel('Phase Response(deg)');
hold off;
```

- Find the Expressions of Cycles vs Filter Order and Plot out the Trends

```
close all;
n = [1:10].*2;
cycles = [321.5,491.75,661.5,831.25,1001.75,1171.5,1341.25,1511.5,1681.75,1851.5];

p = polyfit(n,cycles,1)
f = polyval(p,n);
a = p(1)
b = p(2)
figure;
plot(n,cycles, 'xr');
hold on;
plot(n,f,'--b')
title(horzcat('y = ',num2str(a),'n + ',num2str(b),', Direct Form II'));
xlabel('filter order');
ylabel('instruction cycles');
hold off;



cycles1 = [286.25,357.25,433.75,509.5,585.75,661.75,737.5,813,889.5,965];
p = polyfit(n,cycles1,1)
f = polyval(p,n);
a = p(1)
b = p(2)
figure;
plot(n,cycles1, 'xr');
hold on;
plot(n,f,'--b')
title(horzcat('y = ',num2str(a),'n + ',num2str(b),', Direct Form II -o2'));
xlabel('filter order');
ylabel('instruction cycles');
hold off;



cycles = [229.5,349.25,469.5,589.5,709.75,829.25,949.5,1069.5,1189.5,1309.75];
p = polyfit(n,cycles,1)
f = polyval(p,n);
a = p(1)
b = p(2)
figure;
plot(n,cycles, 'xr');
hold on;
plot(n,f,'--b')
title(horzcat('y = ',num2str(a),'n + ',num2str(b),', Direct Form II Transposed'));
xlabel('filter order');
ylabel('instruction cycles');
hold off;

cycles1 = [240.5,252.25,262.5,272.25,282.5,293,302.25,312.5,322.25,332.75];
p = polyfit(n,cycles1,1)
f = polyval(p,n);
a = p(1)
b = p(2)
figure;
plot(n,cycles1, 'xr');
hold on;
```

```
plot(n,f,'--b')
title(horzcat('y = ',num2str(a),'n + ',num2str(b),', Direct Form II Transposed -o2'));
xlabel('filter order');
ylabel('instruction cycles');
hold off;
```

- Bandpass Filter Design and Graph Plotting

```
close all;
fs = 8000;
Rn = 0.3;
n = 10;
Rs = 20;
w = [270,450].*2./fs;
[b, a] = ellip(n,Rn,Rs,w,'bandpass');

%{
C = reshape([b,a],length(b),2)'
%dlmwrite('coef.txt' ,C, ',')
figure;
freqz(b,a,1029,fs);
figure;
H = tf(b,a,1/fs);
damp(H)

pzmap(H);
figure;
bode (H);
%}
[h,f]=freqz(b,a,1249,fs);
freqz(b,a,1249,fs);
fileID = fopen('bandpass_coef.txt','w');
formSpec1 = 'double b[]={%.16e';
fprintf(fileID,formSpec1,b(1));
formSpec2 = ', %.16e';
fprintf(fileID,formSpec2,b(2:end));
fprintf(fileID,'};\n');
formSpec3 = 'double a[]={%.16e';
fprintf(fileID,formSpec3,a(1));
fprintf(fileID,formSpec2,a(2:end));
fprintf(fileID,'};\n');
fclose(fileID);

gaindirect = cell2mat(directgain(4,2));
phasedirect = cell2mat(directphase(4,2));
gainy = cell2mat(Gain(4,2));
phasey = cell2mat(Phase(4,2));

figure;
plot(f(3:1249),gainy,'--b');
hold on;
plot(f(3:1249),gaindirect,'--k');
hold on;
plot(f(3:1249),gainy-gaindirect,'-r');
legend({'final','all pass','actual IIR = final - all pass'},'FontSize',20);
title('Magnitude Response');
xlabel('Frequency(Hz)');
ylabel('Magnitude Response(dB)');
hold off;

figure;
plot(f(3:1249),phasey,'--b');
hold on;
plot(f(3:1249),phasedirect,'--k');
hold on;
```

```
plot(f(3:1249),phasey-phasedirect,'-r');
legend({'final','all pass','actual IIR = final - all pass'},'FontSize',20);
title('Phase Response');
xlabel('Frequency(Hz)');
ylabel('Phase Response(deg)');
hold off;

figure;
plot(f(3:1249),20.*log10(abs(h(3:1249))),'-g');
hold on;
plot(f(3:1249),gainy-gaindirect,'-r');
legend({'ideal bandpass filter','actual IIR = final - all pass'},'FontSize',20);
title('Magnitude Response');
xlabel('Frequency(Hz)');
ylabel('Magnitude Response(dB)');
hold off;

figure;
plot(f(3:1249),unwrap(angle(h(3:1249))).*180./pi,'-g');
hold on;
plot(f(3:1249),phasey-phasedirect,'-r');
legend({'ideal bandpass filter','actual IIR = final - all pass'},'FontSize',20);
title('Phase Response');
xlabel('Frequency(Hz)');
ylabel('Phase Response(deg)');
hold off;
```

- Bandpass Filter Design and Graph Plotting

```
close all;
fs = 8000;
Rn = 0.3;
n = 10;
Rs = 20;
w = [270,450].*2./fs;
[b, a] = ellip(n,Rn,Rs,w,'bandpass');

%{
C = reshape([b,a],length(b),2)'
%dlmwrite('coef.txt' ,C, ',')
figure;
freqz(b,a,1029,fs);
figure;
H = tf(b,a,1/fs);
damp(H)

pzmap(H);
figure;
bode (H);
%}
[h,f]=freqz(b,a,1249,fs);
freqz(b,a,1249,fs);
fileID = fopen('bandpass_coef.txt','w');
formSpec1 = 'double b[]={%.16e';
fprintf(fileID,formSpec1,b(1));
formSpec2 = ', %.16e';
fprintf(fileID,formSpec2,b(2:end));
fprintf(fileID,'};\n');
formSpec3 = 'double a[]={%.16e';
fprintf(fileID,formSpec3,a(1));
fprintf(fileID,formSpec2,a(2:end));
fprintf(fileID,'};\n');
fclose(fileID);

gaindirect = cell2mat(directgain(4,2));
phasedirect = cell2mat(directphase(4,2));
gainy = cell2mat(Gain(4,2));
phasey = cell2mat(Phase(4,2));

figure;
plot(f(3:1249),gainy,'--b');
hold on;
plot(f(3:1249),gaindirect,'--k');
hold on;
plot(f(3:1249),gainy-gaindirect,'-r');
legend({'final','all pass','actual IIR = final - all pass'},'FontSize',20);
title('Magnitude Response');
xlabel('Frequency(Hz)');
ylabel('Magnitude Response(dB)');
hold off;

figure;
plot(f(3:1249),phasey,'--b');
hold on;
plot(f(3:1249),phasedirect,'--k');
hold on;
```

```
plot(f(3:1249),phasey-phasedirect,'-r');
legend({'final','all pass','actual IIR = final - all pass'},'FontSize',20);
title('Phase Response');
xlabel('Frequency(Hz)');
ylabel('Phase Response(deg)');
hold off;

figure;
plot(f(3:1249),20.*log10(abs(h(3:1249))),'-g');
hold on;
plot(f(3:1249),gainy-gaindirect,'-r');
legend({'ideal bandpass filter','actual IIR = final - all pass'},'FontSize',20);
title('Magnitude Response');
xlabel('Frequency(Hz)');
ylabel('Magnitude Response(dB)');
hold off;

figure;
plot(f(3:1249),unwrap(angle(h(3:1249))).*180./pi,'-g');
hold on;
plot(f(3:1249),phasey-phasedirect,'-r');
legend({'ideal bandpass filter','actual IIR = final - all pass'},'FontSize',20);
title('Phase Response');
xlabel('Frequency(Hz)');
ylabel('Phase Response(deg)');
hold off;
```

- Bandpass Filter Design and Graph Plotting

```
close all;
fs = 8000;
Rn = 0.3;
n = 10;
Rs = 20;
w = [270,450].*2./fs;
[b, a] = ellip(n,Rn,Rs,w,'bandpass');

%{
C = reshape([b,a],length(b),2)'
%dlmwrite('coef.txt' ,C, ',')
figure;
freqz(b,a,1029,fs);
figure;
H = tf(b,a,1/fs);
damp(H)

pzmap(H);
figure;
bode (H);
%}
[h,f]=freqz(b,a,1249,fs);
freqz(b,a,1249,fs);
fileID = fopen('bandpass_coef.txt','w');
formSpec1 = 'double b[]={%.16e';
fprintf(fileID,formSpec1,b(1));
formSpec2 = ', %.16e';
fprintf(fileID,formSpec2,b(2:end));
fprintf(fileID,'};\n');
formSpec3 = 'double a[]={%.16e';
fprintf(fileID,formSpec3,a(1));
fprintf(fileID,formSpec2,a(2:end));
fprintf(fileID,'};\n');
fclose(fileID);

gaindirect = cell2mat(directgain(4,2));
phasedirect = cell2mat(directphase(4,2));
gainy = cell2mat(Gain(4,2));
phasey = cell2mat(Phase(4,2));

figure;
plot(f(3:1249),gainy,'--b');
hold on;
plot(f(3:1249),gaindirect,'--k');
hold on;
plot(f(3:1249),gainy-gaindirect,'-r');
legend({'final','all pass','actual IIR = final - all pass'},'FontSize',20);
title('Magnitude Response');
xlabel('Frequency(Hz)');
ylabel('Magnitude Response(dB)');
hold off;

figure;
plot(f(3:1249),phasey,'--b');
hold on;
plot(f(3:1249),phasedirect,'--k');
hold on;
```

```
plot(f(3:1249),phasey-phasedirect,'-r');
legend({'final','all pass','actual IIR = final - all pass'},'FontSize',20);
title('Phase Response');
xlabel('Frequency(Hz)');
ylabel('Phase Response(deg)');
hold off;

figure;
plot(f(3:1249),20.*log10(abs(h(3:1249))),'-g');
hold on;
plot(f(3:1249),gainy-gaindirect,'-r');
legend({'ideal bandpass filter','actual IIR = final - all pass'},'FontSize',20);
title('Magnitude Response');
xlabel('Frequency(Hz)');
ylabel('Magnitude Response(dB)');
hold off;

figure;
plot(f(3:1249),unwrap(angle(h(3:1249))).*180./pi,'-g');
hold on;
plot(f(3:1249),phasey-phasedirect,'-r');
legend({'ideal bandpass filter','actual IIR = final - all pass'},'FontSize',20);
title('Phase Response');
xlabel('Frequency(Hz)');
ylabel('Phase Response(deg)');
hold off;
```

- Bandpass Filter Design and Graph Plotting

```
close all;
fs = 8000;
Rn = 0.3;
n = 10;
Rs = 20;
w = [270,450].*2./fs;
[b, a] = ellip(n,Rn,Rs,w,'bandpass');

%{
C = reshape([b,a],length(b),2)'
%dlmwrite('coef.txt' ,C, ',')
figure;
freqz(b,a,1029,fs);
figure;
H = tf(b,a,1/fs);
damp(H)

pzmap(H);
figure;
bode (H);
%}
[h,f]=freqz(b,a,1249,fs);
freqz(b,a,1249,fs);
fileID = fopen('bandpass_coef.txt','w');
formSpec1 = 'double b[]={%.16e';
fprintf(fileID,formSpec1,b(1));
formSpec2 = ', %.16e';
fprintf(fileID,formSpec2,b(2:end));
fprintf(fileID,'};\n');
formSpec3 = 'double a[]={%.16e';
fprintf(fileID,formSpec3,a(1));
fprintf(fileID,formSpec2,a(2:end));
fprintf(fileID,'};\n');
fclose(fileID);

gaindirect = cell2mat(directgain(4,2));
phasedirect = cell2mat(directphase(4,2));
gainy = cell2mat(Gain(4,2));
phasey = cell2mat(Phase(4,2));

figure;
plot(f(3:1249),gainy,'--b');
hold on;
plot(f(3:1249),gaindirect,'--k');
hold on;
plot(f(3:1249),gainy-gaindirect,'-r');
legend({'final','all pass','actual IIR = final - all pass'},'FontSize',20);
title('Magnitude Response');
xlabel('Frequency(Hz)');
ylabel('Magnitude Response(dB)');
hold off;

figure;
plot(f(3:1249),phasey,'--b');
hold on;
plot(f(3:1249),phasedirect,'--k');
hold on;
```

```
plot(f(3:1249),phasey-phasedirect,'-r');
legend({'final','all pass','actual IIR = final - all pass'},'FontSize',20);
title('Phase Response');
xlabel('Frequency(Hz)');
ylabel('Phase Response(deg)');
hold off;

figure;
plot(f(3:1249),20.*log10(abs(h(3:1249))),'-g');
hold on;
plot(f(3:1249),gainy-gaindirect,'-r');
legend({'ideal bandpass filter','actual IIR = final - all pass'},'FontSize',20);
title('Magnitude Response');
xlabel('Frequency(Hz)');
ylabel('Magnitude Response(dB)');
hold off;

figure;
plot(f(3:1249),unwrap(angle(h(3:1249))).*180./pi,'-g');
hold on;
plot(f(3:1249),phasey-phasedirect,'-r');
legend({'ideal bandpass filter','actual IIR = final - all pass'},'FontSize',20);
title('Phase Response');
xlabel('Frequency(Hz)');
ylabel('Phase Response(deg)');
hold off;
```

- Assembly for Direct Form II Implementation with no Optimisation

```
                  bandpass_direct2_IIR:
0x0000D070:   07BF005A            SUB.L2      SP,8,SP
0x0000D074:   02016E4E            LDH.D2T2    *+B14[366],B4
0x0000D078:   00006001            NOP         4
0x0000D07C:   00000000 ||         NOP
0x0000D080:   0310073B            INTDP.L2    B4,B7:B6
0x0000D084:   0200BA6E ||         LDW.D2T2    *+B14[186],B4
0x0000D088:   00006000            NOP         4
0x0000D08C:   031002F6            STW.D2T2    B6,*+B4[0]
0x0000D090:   039022F6            STW.D2T2    B7,*+B4[1]
0x0000D094:   0200BD6E            LDW.D2T2    *+B14[189],B4
0x0000D098:   00006000            NOP         4
0x0000D09C:   023C22F6            STW.D2T2    B4,*+SP[1]
0x0000D0A0:   00100ADA            CMPLT.L2    0,B4,B0
0x0000D0A4:   30001A90      [!B0] B.S1        C$DW$L$_bandpass_direct2_IIR$2$E (PC+212 = 0x0000d174)
0x0000D0A8:   00008000            NOP         5
                  C$DW$L$_bandpass_direct2_IIR$2$B, C$L4:
0x0000D0AC:   0300BA6C            LDW.D2T1    *+B14[186],A6
0x0000D0B0:   02881428            MVK.S1      0x1028,A5
0x0000D0B4:   028000E8            MVKH.S1     0x10000,A5
0x0000D0B8:   02101059            MV.L1X      B4,A4
0x0000D0BC:   00000000 ||         NOP
0x0000D0C0:   01901059            MV.L1X      B4,A3
0x0000D0C4:   02148B64 ||         LDDW.D1T1   *+A5[A4],A5:A4
0x0000D0C8:   03186B64            LDDW.D1T1   *+A6[A3],A7:A6
0x0000D0CC:   0480BA6C            LDW.D2T1    *+B14[186],A9
0x0000D0D0:   00004000            NOP         3
0x0000D0D4:   0310C700            MPYDP.M1    A7:A6,A5:A4,A7:A6
0x0000D0D8:   02240364            LDDW.D1T1   *+A9[0],A5:A4
0x0000D0DC:   0000E000            NOP         8
0x0000D0E0:   02188338            SUBDP.L1    A5:A4,A7:A6,A5:A4
0x0000D0E4:   0000A000            NOP         6
0x0000D0E8:   02A42274            STW.D1T1    A5,*+A9[1]
0x0000D0EC:   02240274            STW.D1T1    A4,*+A9[0]
0x0000D0F0:   023C22E6            LDW.D2T2    *+SP[1],B4
0x0000D0F4:   0200BA6C            LDW.D2T1    *+B14[186],A4
0x0000D0F8:   02880C2A            MVK.S2      0x1018,B5
0x0000D0FC:   028000EA            MVKH.S2     0x10000,B5
0x0000D100:   00000000            NOP
0x0000D104:   01901058            MV.L1X      B4,A3
0x0000D108:   02106B65            LDDW.D1T1   *+A4[A3],A5:A4
0x0000D10C:   02148BE6 ||         LDDW.D2T2   *+B5[B4],B5:B4
0x0000D110:   00006000            NOP         4
0x0000D114:   02109700            MPYDP.M1X   A5:A4,B5:B4,A5:A4
0x0000D118:   0280B96E            LDW.D2T2    *+B14[185],B5
0x0000D11C:   0200B86E            LDW.D2T2    *+B14[184],B4
0x0000D120:   0000C000            NOP         7
0x0000D124:   0210931A            ADDDP.L2X   B5:B4,A5:A4,B5:B4
0x0000D128:   0000A000            NOP         6
0x0000D12C:   0280B97E            STW.D2T2    B5,*+B14[185]
0x0000D130:   0200B87E            STW.D2T2    B4,*+B14[184]
0x0000D134:   0300BA6C            LDW.D2T1    *+B14[186],A6
0x0000D138:   028FE058            SUB.L1      A3,1,A5
0x0000D13C:   00004000            NOP         3
0x0000D140:   0218AB66            LDDW.D1T2   *+A6[A5],B5:B4
0x0000D144:   02180058            MV.L1       A6,A4
0x0000D148:   01907E40            ADDAD.D1    A4,A3,A3
0x0000D14C:   00002000            NOP         2
0x0000D150:   028C2276            STW.D1T2    B5,*+A3[1]
```

```
0x0000D154:   020C0276          STW.D1T2     B4,*+A3[0]
0x0000D158:   023C22E6          LDW.D2T2     *+SP[1],B4
0x0000D15C:   00006000          NOP          4
0x0000D160:   0213E05A          SUB.L2       B4,1,B4
0x0000D164:   023C22F6          STW.D2T2     B4,*+SP[1]
0x0000D168:   00100ADA          CMPLT.L2     0,B4,B0
0x0000D16C:   2FFFE990    [ B0] B.S1         C$L4  (PC-180 = 0x0000d0ac)
0x0000D170:   00008000          NOP          5
            C$L5, C$DW$L$_bandpass_direct2_IIR$2$E:
0x0000D174:   0200BA6E          LDW.D2T2     *+B14[186],B4
0x0000D178:   01880C28          MVK.S1       0x1018,A3
0x0000D17C:   018000E8          MVKH.S1      0x10000,A3
0x0000D180:   020C0364          LDDW.D1T1    *+A3[0],A5:A4
0x0000D184:   0380B96E          LDW.D2T2     *+B14[185],B7
0x0000D188:   021003E6          LDDW.D2T2    *+B4[0],B5:B4
0x0000D18C:   0300B86E          LDW.D2T2     *+B14[184],B6
0x0000D190:   00004000          NOP          3
0x0000D194:   02109702          MPYDP.M2X    B5:B4,A5:A4,B5:B4
0x0000D198:   00010000          NOP          9
0x0000D19C:   0218831A          ADDDP.L2     B5:B4,B7:B6,B5:B4
0x0000D1A0:   0000A000          NOP          6
0x0000D1A4:   0200B87E          STW.D2T2     B4,*+B14[184]
0x0000D1A8:   0280B97E          STW.D2T2     B5,*+B14[185]
0x0000D1AC:   07BD005A          ADD.L2       8,SP,SP
0x0000D1B0:   000C0362          B.S2         B3
0x0000D1B4:   00008000          NOP          5
```

- Assembly for Direct Form II Implementation with o2 Optimisation

```
                    bandpass_direct2_IIR:
0x0000D2BC:   02016E4E                LDH.D2T2     *+B14[366],B4
0x0000D2C0:   0480BA6C                LDW.D2T1     *+B14[186],A9
0x0000D2C4:   00004000                NOP          3
0x0000D2C8:   0210073A                INTDP.L2     B4,B5:B4
0x0000D2CC:   00006000                NOP          4
0x0000D2D0:   02240276                STW.D1T2     B4,*+A9[0]
0x0000D2D4:   02A42276                STW.D1T2     B5,*+A9[1]
0x0000D2D8:   00088C29                MVK.S1       0x1118,A0
0x0000D2DC:   0200BD6E ||             LDW.D2T2     *+B14[189],B4
0x0000D2E0:   0288942A                MVK.S2       0x1128,B5
0x0000D2E4:   0380B96C                LDW.D2T1     *+B14[185],A7
0x0000D2E8:   028000EA                MVKH.S2      0x10000,B5
0x0000D2EC:   000000E8                MVKH.S1      0x10000,A0
0x0000D2F0:   00100ADA                CMPLT.L2     0,B4,B0
0x0000D2F4:   30002610        [!B0]   B.S1         C$L18 (PC+304 = 0x0000d410)
0x0000D2F8:   0300B86C                LDW.D2T1     *+B14[184],A6
0x0000D2FC:   02949E42                ADDAD.D2     B5,B4,B5
0x0000D300:   01901058                MV.L1X       B4,A3
0x0000D304:   02007E40                ADDAD.D1     A0,A3,A4
0x0000D308:   01A47E40                ADDAD.D1     A9,A3,A3
0x0000D30C:   008403E3                MVC.S2       CSR,B1
0x0000D310:   00901059 ||             MV.L1X       B4,A1
0x0000D314:   010000A9 ||             MVK.S1       0x0001,A2
0x0000D318:   0010105B ||             MV.L2X       A4,B0
0x0000D31C:   00000000 ||             NOP
0x0000D320:   0207CF5B                AND.L2       -2,B1,B4
0x0000D324:   04141059 ||             MV.L1X       B5,A8
0x0000D328:   028C11A2 ||             MV.S2X       A3,B5
0x0000D32C:   009003A2                MVC.S2       B4,CSR
                    C$DW$L$_bandpass_direct2_IIR$4$B, C$L16, C$L15:
0x0000D330:   03240364                LDDW.D1T1    *+A9[0],A7:A6
0x0000D334:   00006000                NOP          4
0x0000D338:   0210C338                SUBDP.L1     A7:A6,A5:A4,A5:A4
0x0000D33C:   0000A000                NOP          6
0x0000D340:   B2240274        [!A2]   STW.D1T1     A4,*+A9[0]
0x0000D344:   B2A42274        [!A2]   STW.D1T1     A5,*+A9[1]
0x0000D348:   B40035E7        [!A2]   LDDW.D2T2    *B0--[1],B9:B8
0x0000D34C:   B30C2364 ||     [!A2]   LDDW.D1T1    *+A3[1],A7:A6
0x0000D350:   0380B96F                LDW.D2T2     *+B14[185],B7
0x0000D354:   B20C0364 ||     [!A2]   LDDW.D1T1    *+A3[0],A5:A4
0x0000D358:   0300B86E                LDW.D2T2     *+B14[184],B6
0x0000D35C:   00002000                NOP          2
0x0000D360:   04191702                MPYDP.M2X    B9:B8,A7:A6,B9:B8
0x0000D364:   00004000                NOP          3
0x0000D368:   0410105A                MV.L2X       A4,B8
0x0000D36C:   0214105A                MV.L2X       A5,B4
0x0000D370:   B41454F6        [!A2]   STW.D2T2     B8,*B5--[2]
0x0000D374:   B21462F6        [!A2]   STW.D2T2     B4,*+B5[3]
0x0000D378:   02203564                LDDW.D1T1    *A8--[1],A5:A4
0x0000D37C:   030C3564                LDDW.D1T1    *A3--[1],A7:A6
0x0000D380:   0319031A                ADDDP.L2     B9:B8,B7:B6,B7:B6
0x0000D384:   00000000                NOP
0x0000D388:   8087E058        [ A1]   SUB.L1       A1,1,A1
0x0000D38C:   8FFFF610        [ A1]   B.S1         C$L15 (PC-80 = 0x0000d330)
0x0000D390:   0210C700                MPYDP.M1     A7:A6,A5:A4,A5:A4
0x0000D394:   00002000                NOP          2
```

```
0x0000D398:   B300B87F     [!A2]  STW.D2T2     B6,*+B14[184]
0x0000D39C:   00000000  ||        NOP
0x0000D3A0:   A10BE059     [ A2]  SUB.L1       A2,1,A2
0x0000D3A4:   B380B97E  || [!A2]  STW.D2T2     B7,*+B14[185]
             C$L17, C$DW$L$_bandpass_direct2_IIR$4$E:
0x0000D3A8:   03240364            LDDW.D1T1    *+A9[0],A7:A6
0x0000D3AC:   00006000            NOP          4
0x0000D3B0:   0210C338            SUBDP.L1     A7:A6,A5:A4,A5:A4
0x0000D3B4:   0000A000            NOP          6
0x0000D3B8:   02240274            STW.D1T1     A4,*+A9[0]
0x0000D3BC:   02A42274            STW.D1T1     A5,*+A9[1]
0x0000D3C0:   040035E7            LDDW.D2T2    *B0--[1],B9:B8
0x0000D3C4:   030C2364  ||        LDDW.D1T1    *+A3[1],A7:A6
0x0000D3C8:   0380B96F            LDW.D2T2     *+B14[185],B7
0x0000D3CC:   020C0364  ||        LDDW.D1T1    *+A3[0],A5:A4
0x0000D3D0:   0300B86E            LDW.D2T2     *+B14[184],B6
0x0000D3D4:   00002000            NOP          2
0x0000D3D8:   04191702            MPYDP.M2X    B9:B8,A7:A6,B9:B8
0x0000D3DC:   00004000            NOP          3
0x0000D3E0:   0410105A            MV.L2X       A4,B8
0x0000D3E4:   0214105A            MV.L2X       A5,B4
0x0000D3E8:   041454F6            STW.D2T2     B8,*B5--[2]
0x0000D3EC:   021462F6            STW.D2T2     B4,*+B5[3]
0x0000D3F0:   00002000            NOP          2
0x0000D3F4:   0319031A            ADDDP.L2     B9:B8,B7:B6,B7:B6
0x0000D3F8:   008403A2            MVC.S2       B1,CSR
0x0000D3FC:   00008000            NOP          5
0x0000D400:   0380B97F            STW.D2T2     B7,*+B14[185]
0x0000D404:   039C1058  ||        MV.L1X       B7,A7
0x0000D408:   03181059            MV.L1X       B6,A6
0x0000D40C:   0300B87E  ||        STW.D2T2     B6,*+B14[184]
             C$L18:
0x0000D410:   02000364            LDDW.D1T1    *+A0[0],A5:A4
0x0000D414:   04240364            LDDW.D1T1    *+A9[0],A9:A8
0x0000D418:   00006000            NOP          4
0x0000D41C:   02110700            MPYDP.M1     A9:A8,A5:A4,A5:A4
0x0000D420:   00010000            NOP          9
0x0000D424:   02188318            ADDDP.L1     A5:A4,A7:A6,A5:A4
0x0000D428:   0000A000            NOP          6
0x0000D42C:   0200B87C            STW.D2T1     A4,*+B14[184]
0x0000D430:   0280B97C            STW.D2T1     A5,*+B14[185]
0x0000D434:   000C0362            B.S2         B3
0x0000D438:   00008000            NOP          5
```

- Assembly for Direct Form II Transposed Implementation with no Optimisation

```
             bandpass_direct2_transposed_IIR:
0x0000D1B8:   07BF005A           SUB.L2       SP,8,SP
0x0000D1BC:   02016E4E           LDH.D2T2     *+B14[366],B4
0x0000D1C0:   01880C28           MVK.S1       0x1018,A3
0x0000D1C4:   018000E8           MVKH.S1      0x10000,A3
0x0000D1C8:   020C0364           LDDW.D1T1    *+A3[0],A5:A4
0x0000D1CC:   0300BC6E           LDW.D2T2     *+B14[188],B6
0x0000D1D0:   0210073A           INTDP.L2     B4,B5:B4
0x0000D1D4:   00006000           NOP          4
0x0000D1D8:   02109702           MPYDP.M2X    B5:B4,A5:A4,B5:B4
0x0000D1DC:   031803E6           LDDW.D2T2    *+B6[0],B7:B6
0x0000D1E0:   0000E000           NOP          8
0x0000D1E4:   0210C31A           ADDDP.L2     B7:B6,B5:B4,B5:B4
0x0000D1E8:   0000A000           NOP          6
0x0000D1EC:   0200B87E           STW.D2T2     B4,*+B14[184]
0x0000D1F0:   0280B97E           STW.D2T2     B5,*+B14[185]
0x0000D1F4:   020000AA           MVK.S2       0x0001,B4
0x0000D1F8:   023C22F6           STW.D2T2     B4,*+SP[1]
0x0000D1FC:   0200BD6E           LDW.D2T2     *+B14[189],B4
0x0000D200:   02BC22E6           LDW.D2T2     *+SP[1],B5
0x0000D204:   00006000           NOP          4
0x0000D208:   0010AAFA           CMPLT.L2     B5,B4,B0
0x0000D20C:   30001410     [!B0] B.S1         C$DW$L$_bandpass_direct2_transposed_IIR$2$E (PC+160 = 0x0000d2a0)
0x0000D210:   00008000           NOP          5
             C$DW$L$_bandpass_direct2_transposed_IIR$2$B, C$L6:
0x0000D214:   02016E4E           LDH.D2T2     *+B14[366],B4
0x0000D218:   0408142B           MVK.S2       0x1028,B8
0x0000D21C:   0380B96E ||        LDW.D2T2     *+B14[185],B7
0x0000D220:   0300B86E           LDW.D2T2     *+B14[184],B6
0x0000D224:   040000EA           MVKH.S2      0x10000,B8
0x0000D228:   0420ABE6           LDDW.D2T2    *+B8[B5],B9:B8
0x0000D22C:   02080C28           MVK.S1       0x1018,A4
0x0000D230:   01941058           MV.L1X       B5,A3
0x0000D234:   020000E8           MVKH.S1      0x10000,A4
0x0000D238:   02106B64           LDDW.D1T1    *+A4[A3],A5:A4
0x0000D23C:   0210073A           INTDP.L2     B4,B5:B4
0x0000D240:   03190702           MPYDP.M2     B9:B8,B7:B6,B7:B6
0x0000D244:   0480BC6C           LDW.D2T1     *+B14[188],A9
0x0000D248:   0300BC6C           LDW.D2T1     *+B14[188],A6
0x0000D24C:   040FE058           SUB.L1       A3,1,A8
0x0000D250:   02109702           MPYDP.M2X    B5:B4,A5:A4,B5:B4
0x0000D254:   00010000           NOP          9
0x0000D258:   0218833A           SUBDP.L2     B5:B4,B7:B6,B5:B4
0x0000D25C:   02246B64           LDDW.D1T1    *+A9[A3],A5:A4
0x0000D260:   01991E40           ADDAD.D1     A6,A8,A3
0x0000D264:   00006000           NOP          4
0x0000D268:   02109318           ADDDP.L1X    A5:A4,B5:B4,A5:A4
0x0000D26C:   0000A000           NOP          6
0x0000D270:   028C2274           STW.D1T1     A5,*+A3[1]
0x0000D274:   020C0274           STW.D1T1     A4,*+A3[0]
0x0000D278:   023C22E6           LDW.D2T2     *+SP[1],B4
0x0000D27C:   00006000           NOP          4
0x0000D280:   0210205A           ADD.L2       1,B4,B4
0x0000D284:   023C22F6           STW.D2T2     B4,*+SP[1]
0x0000D288:   0200BD6E           LDW.D2T2     *+B14[189],B4
0x0000D28C:   02BC22E6           LDW.D2T2     *+SP[1],B5
0x0000D290:   00006000           NOP          4
0x0000D294:   0010AAFA           CMPLT.L2     B5,B4,B0
```

```
0x0000D298:   2FFFF290     [ B0]  B.S1          C$L6 (PC-108 = 0x0000d214)
0x0000D29C:   00008000            NOP           5
            C$L7, C$DW$L$_bandpass_direct2_transposed_IIR$2$E:
0x0000D2A0:   02816E4E            LDH.D2T2      *+B14[366],B5
0x0000D2A4:   0308142B            MVK.S2        0x1028,B6
0x0000D2A8:   0480B96E ||         LDW.D2T2      *+B14[185],B9
0x0000D2AC:   0400B86E            LDW.D2T2      *+B14[184],B8
0x0000D2B0:   030000EA            MVKH.S2       0x10000,B6
0x0000D2B4:   03188BE6            LDDW.D2T2     *+B6[B4],B7:B6
0x0000D2B8:   02080C28            MVK.S1        0x1018,A4
0x0000D2BC:   01901058            MV.L1X        B4,A3
0x0000D2C0:   020000E8            MVKH.S1       0x10000,A4
0x0000D2C4:   02106B64            LDDW.D1T1     *+A4[A3],A5:A4
0x0000D2C8:   0214073A            INTDP.L2      B5,B5:B4
0x0000D2CC:   0320C702            MPYDP.M2      B7:B6,B9:B8,B7:B6
0x0000D2D0:   00004000            NOP           3
0x0000D2D4:   02109702            MPYDP.M2X     B5:B4,A5:A4,B5:B4
0x0000D2D8:   00010000            NOP           9
0x0000D2DC:   0218833A            SUBDP.L2      B5:B4,B7:B6,B5:B4
0x0000D2E0:   0380BC6E            LDW.D2T2      *+B14[188],B7
0x0000D2E4:   030FF05A            SUB.L2X       A3,1,B6
0x0000D2E8:   00004000            NOP           3
0x0000D2EC:   031CDE42            ADDAD.D2      B7,B6,B6
0x0000D2F0:   021802F6            STW.D2T2      B4,*+B6[0]
0x0000D2F4:   029822F6            STW.D2T2      B5,*+B6[1]
0x0000D2F8:   07BD005A            ADD.L2        8,SP,SP
0x0000D2FC:   000C0362            B.S2          B3
0x0000D300:   00008000            NOP           5
0x0000D304:   00000000            NOP
0x0000D308:   00000000            NOP
0x0000D30C:   00000000            NOP
0x0000D310:   00000000            NOP
0x0000D314:   00000000            NOP
0x0000D318:   00000000            NOP
0x0000D31C:   00000000            NOP
```

- Assembly for Direct Form II Transposed Implementation with o2 Optimisation

```
                  C$DW$L$_main$3$E, bandpass_direct2_transposed_IIR:
0x0000D018:   04BC1059              MV.L1X        SP,A9
0x0000D01C:   07BD54F4 ||           STW.D2T1      FP,*SP--[10]
0x0000D020:   05BD22F6              STW.D2T2      B11,*+SP[9]
0x0000D024:   053D02F6              STW.D2T2      B10,*+SP[8]
0x0000D028:   01BCE2F6              STW.D2T2      B3,*+SP[7]
0x0000D02C:   07248074              STW.D1T1      A14,*-A9[4]
0x0000D030:   06A4A074              STW.D1T1      A13,*-A9[5]
0x0000D034:   0624C074              STW.D1T1      A12,*-A9[6]
0x0000D038:   05A4E074              STW.D1T1      A11,*-A9[7]
0x0000D03C:   05250074              STW.D1T1      A10,*-A9[8]
0x0000D040:   05016E4E              LDH.D2T2      *+B14[366],B10
0x0000D044:   07888C28              MVK.S1        0x1118,FP
0x0000D048:   078000E8              MVKH.S1       0x10000,FP
0x0000D04C:   023C0364              LDDW.D1T1     *+FP[0],A5:A4
0x0000D050:   0680BC6C              LDW.D2T1      *+B14[188],A13
0x0000D054:   0328073A              INTDP.L2      B10,B7:B6
0x0000D058:   00006000              NOP           4
0x0000D05C:   03189700              MPYDP.M1X     A5:A4,B7:B6,A7:A6
0x0000D060:   02340364              LDDW.D1T1     *+A13[0],A5:A4
0x0000D064:   0000E000              NOP           8
0x0000D068:   03188318              ADDDP.L1      A5:A4,A7:A6,A7:A6
0x0000D06C:   0000A000              NOP           6
0x0000D070:   0300B87C              STW.D2T1      A6,*+B14[184]
0x0000D074:   0380B97C              STW.D2T1      A7,*+B14[185]
0x0000D078:   0700BD6C              LDW.D2T1      *+B14[189],A14
0x0000D07C:   00006000              NOP           4
0x0000D080:   00B848D8              CMPGT.L1      2,A14,A1
0x0000D084:   80003C10        [ A1] B.S1          C$L14 (PC+480 = 0x0000d260)
0x0000D088:   00008000              NOP           5
0x0000D08C:   04889429              MVK.S1        0x1128,A9
0x0000D090:   0235E0F9 ||           SUB.L1        FP,A13,A4
0x0000D094:   02803C2B ||           MVK.S2        0x0078,B5
0x0000D098:   01B5E8C1 ||           SUB.D1        A13,FP,A3
0x0000D09C:   003BF05A ||           SUB.L2X       A14,1,B0
0x0000D0A0:   048000E9              MVKH.S1       0x10000,A9
0x0000D0A4:   0200442B ||           MVK.S2        0x0088,B4
0x0000D0A8:   01BD105B ||           ADD.L2X       8,FP,B3
0x0000D0AC:   06340059 ||           MV.L1         A13,A12
0x0000D0B0:   00340940 ||           MV.D1         A13,A0
0x0000D0B4:   02803C29              MVK.S1        0x0078,A5
0x0000D0B8:   0480442B ||           MVK.S2        0x0088,B9
0x0000D0BC:   00000000 ||           NOP
0x0000D0C0:   04148AF9              CMPLT.L1      A4,A5,A8
0x0000D0C4:   0225A5E1 ||           SUB.S1        A13,A9,A4
0x0000D0C8:   02A5A8C0 ||           SUB.D1        A9,A13,A5
0x0000D0CC:   02109AF9              CMPLT.L1X     A4,B4,A4
0x0000D0D0:   0214B8FA ||           CMPGT.L2X     B5,A5,B4
0x0000D0D4:   02109F7B              AND.L2X       B4,A4,B4
0x0000D0D8:   02247AF9 ||           CMPLT.L1X     A3,B9,A4
0x0000D0DC:   00000000 ||           NOP
0x0000D0E0:   02208F79              AND.L1        A4,A8,A4
0x0000D0E4:   02102DDA ||           XOR.L2        1,B4,B4
0x0000D0E8:   02102DD9              XOR.L1        1,A4,A4
0x0000D0EC:   0428073A ||           INTDP.L2      B10,B9:B8
0x0000D0F0:   00909F7A              AND.L2X       B4,A4,B1
0x0000D0F4:   40001290        [ B1] B.S1          C$L8 (PC+148 = 0x0000d174)
0x0000D0F8:   02BBE058              SUB.L1        A14,1,A5
0x0000D0FC:   01A50058              ADD.L1        8,A9,A3
0x0000D100:   023D0058              ADD.L1        8,FP,A4
```

```
0x0000D104:  0225105A            ADD.L2X     8,A9,B4
0x0000D108:  40802ADA    [ B1]   CMPLT.L2    1,B0,B1
0x0000D10C:  048403E3            MVC.S2      CSR,B9
0x0000D110:  01901059 ||         MV.L1X      B4,A3
0x0000D114:  0014105A ||         MV.L2X      A5,B0
0x0000D118:  0227CF5B            AND.L2      -2,B9,B4
0x0000D11C:  041011A2 ||         MV.S2X      A4,B8
0x0000D120:  009003A2            MVC.S2      B4,CSR
            C$DW$L$_bandpass_direct2_transposed_IIR$5$B, C$L6, C$L5:
0x0000D124:  020C3765            LDDW.D1T1   *A3++[1],A5:A4
0x0000D128:  022037E6 ||         LDDW.D2T2   *B8++[1],B5:B4
0x0000D12C:  00006000            NOP         4
0x0000D130:  0210C701            MPYDP.M1    A7:A6,A5:A4,A5:A4
0x0000D134:  0210C702 ||         MPYDP.M2    B7:B6,B5:B4,B5:B4
0x0000D138:  00010000            NOP         9
0x0000D13C:  021093B8            SUBDP.L1X   B5:B4,A5:A4,A5:A4
0x0000D140:  00000000            NOP
0x0000D144:  04003364            LDDW.D1T1   *++A0[1],A9:A8
0x0000D148:  00006000            NOP         4
0x0000D14C:  02110318            ADDDP.L1    A9:A8,A5:A4,A5:A4
0x0000D150:  00000000            NOP
0x0000D154:  2003E05A    [ B0]   SUB.L2      B0,1,B0
0x0000D158:  2FFFFC92    [ B0]   B.S2        C$L5 (PC-28 = 0x0000d124)
0x0000D15C:  00004000            NOP         3
0x0000D160:  02004074            STW.D1T1    A4,*-A0[2]
0x0000D164:  02802074            STW.D1T1    A5,*-A0[1]
            C$L7, C$DW$L$_bandpass_direct2_transposed_IIR$5$E:
0x0000D168:  00002010            B.S1        C$L14 (PC+256 = 0x0000d260)
0x0000D16C:  00A403A2            MVC.S2      B9,CSR
0x0000D170:  00006000            NOP         4
            C$L8:
0x0000D174:  40000D10    [ B1]   B.S1        C$L10 (PC+104 = 0x0000d1c8)
0x0000D178:  00008001            NOP         5
0x0000D17C:  00000000 ||         NOP
            C$DW$L$_bandpass_direct2_transposed_IIR$9$B, C$L9:
0x0000D180:  020C3765            LDDW.D1T1   *A3++[1],A5:A4
0x0000D184:  020C37E6 ||         LDDW.D2T2   *B3++[1],B5:B4
0x0000D188:  00006000            NOP         4
            C$DW$L$_bandpass_direct2_transposed_IIR$10$B, C$DW$L$_bandpass_direct2_transposed_IIR$9$E:
0x0000D18C:  0210C701            MPYDP.M1    A7:A6,A5:A4,A5:A4
0x0000D190:  02110702 ||         MPYDP.M2    B9:B8,B5:B4,B5:B4
0x0000D194:  04302364            LDDW.D1T1   *++A12[1],A9:A8
0x0000D198:  0003E05A            SUB.L2      B0,1,B0
0x0000D19C:  0000C000            NOP         7
0x0000D1A0:  021093B8            SUBDP.L1X   B5:B4,A5:A4,A5:A4
0x0000D1A4:  0000A000            NOP         6
0x0000D1A8:  02110318            ADDDP.L1    A9:A8,A5:A4,A5:A4
0x0000D1AC:  00002000            NOP         2
0x0000D1B0:  2FFFFC10    [ B0]   B.S1        C$L9 (PC-32 = 0x0000d180)
0x0000D1B4:  30001810    [!B0]   B.S1        C$L14 (PC+192 = 0x0000d260)
0x0000D1B8:  00002000            NOP         2
0x0000D1BC:  02303674            STW.D1T1    A4,*A12++[1]
0x0000D1C0:  02B03674            STW.D1T1    A5,*A12++[1]
            C$DW$L$_bandpass_direct2_transposed_IIR$10$E:
0x0000D1C4:  00000000            NOP
            C$L10:
0x0000D1C8:  058403E3            MVC.S2      CSR,B11
0x0000D1CC:  0000405B ||         ADD.L2      2,B0,B0
0x0000D1D0:  030C37E7 ||         LDDW.D2T2   *B3++[1],B7:B6
0x0000D1D4:  040C3764 ||         LDDW.D1T1   *A3++[1],A9:A8
```

```
0x0000D1D8:    022FCF5B             AND.L2       -2,B11,B4
0x0000D1DC:    008001AA ||          MVK.S2       0x0003,B1
0x0000D1E0:    009003A2             MVC.S2       B4,CSR
0x0000D1E4:    010002AA             MVK.S2       0x0005,B2
              C$L11:
0x0000D1E8:    051801A1             MV.S1        A6,A10
0x0000D1EC:    059C0941 ||          MV.D1        A7,A11
0x0000D1F0:    0103B059 ||          SUB.L1X      B0,3,A2
0x0000D1F4:    00000312 ||          B.S2         C$L12 (PC+24 = 0x0000d1f8)
              C$DW$L$_bandpass_direct2_transposed_IIR$14$B, C$L12:
0x0000D1F8:    0010C319             ADDDP.L1     A7:A6,A5:A4,A1:A0
0x0000D1FC:    53303364 ||   [!B1]  LDDW.D1T1    *++A12[1],A7:A6
0x0000D200:    00000000             NOP
0x0000D204:    A40C3764      [ A2]  LDDW.D1T1    *A3++[1],A9:A8
0x0000D208:    7030C075      [!B2]  STW.D1T1     A0,*-A12[6]
0x0000D20C:    2003E05B ||   [ B0]  SUB.L2       B0,1,B0
0x0000D210:    021893B9 ||          SUBDP.L1X    B5:B4,A7:A6,A5:A4
0x0000D214:    03214701 ||          MPYDP.M1     A11:A10,A9:A8,A7:A6
0x0000D218:    02190703 ||          MPYDP.M2     B9:B8,B7:B6,B5:B4
0x0000D21C:    A30C37E6 ||   [ A2]  LDDW.D2T2    *B3++[1],B7:B6
0x0000D220:    4087E05B      [ B1]  SUB.L2       B1,1,B1
0x0000D224:    610829C3 ||   [ B2]  SUB.D2       B2,0x1,B2
0x0000D228:    A10BE1A1 ||   [ A2]  SUB.S1       A2,1,A2
0x0000D22C:    70B0A075 ||   [!B2]  STW.D1T1     A1,*-A12[5]
0x0000D230:    2FFFFB12 ||   [ B0]  B.S2         C$L12 (PC-40 = 0x0000d1f8)
              C$L13, C$DW$L$_bandpass_direct2_transposed_IIR$14$E:
0x0000D234:    0010C318             ADDDP.L1     A7:A6,A5:A4,A1:A0
0x0000D238:    00002000             NOP          2
0x0000D23C:    00308074             STW.D1T1     A0,*-A12[4]
0x0000D240:    00B06074             STW.D1T1     A1,*-A12[3]
0x0000D244:    00AC03A2             MVC.S2       B11,CSR
0x0000D248:    00000000             NOP
0x0000D24C:    00B02074             STW.D1T1     A1,*-A12[1]
0x0000D250:    03AC0059             MV.L1        A11,A7
0x0000D254:    032801A1 ||          MV.S1        A10,A6
0x0000D258:    00304075 ||          STW.D1T1     A0,*-A12[2]
0x0000D25C:    00000000 ||          NOP
              C$L14:
0x0000D260:    01889429             MVK.S1       0x1128,A3
0x0000D264:    0228073B ||          INTDP.L2     B10,B5:B4
0x0000D268:    043DCB64 ||          LDDW.D1T1    *+FP[A14],A9:A8
0x0000D26C:    018000E8             MVKH.S1      0x10000,A3
0x0000D270:    020DCB64             LDDW.D1T1    *+A3[A14],A5:A4
0x0000D274:    01B5DE40             ADDAD.D1     A13,A14,A3
0x0000D278:    00002000             NOP          2
0x0000D27C:    02209702             MPYDP.M2X    B5:B4,A9:A8,B5:B4
0x0000D280:    0210C700             MPYDP.M1     A7:A6,A5:A4,A5:A4
0x0000D284:    00010000             NOP          9
0x0000D288:    0210933A             SUBDP.L2X    B5:B4,A5:A4,B5:B4
0x0000D28C:    0000A000             NOP          6
0x0000D290:    020C4076             STW.D1T2     B4,*-A3[2]
0x0000D294:    028C2076             STW.D1T2     B5,*-A3[1]
0x0000D298:    053C83E7             LDDW.D2T2    *+SP[4],B11:B10
0x0000D29C:    04BC1058 ||          MV.L1X       SP,A9
0x0000D2A0:    01BCE2E6             LDW.D2T2     *+SP[7],B3
0x0000D2A4:    0724C264             LDW.D1T1     *+A9[6],A14
0x0000D2A8:    06244364             LDDW.D1T1    *+A9[2],A13:A12
0x0000D2AC:    05242364             LDDW.D1T1    *+A9[1],A11:A10
0x0000D2B0:    07BD52E4             LDW.D2T1     *++SP[10],FP
0x0000D2B4:    000C0362             B.S2         B3
```