Report: Lab 5

EE3-19 Real-time Digital Signal Processing

Tayyab Bhandari {01334547}, **Timo Thans** {01668411},
Department of Electrical and Electronic Engineering, Imperial College London

I. Introduction

In this lab we learnt how to design IIR filters using Matlab and implement this on the C6713 DSK board to test with an input signal in real-time. We have looked into three different implementation styles: direct form I, direct form II and direct form II transposed. We also measured our designed filter characteristics using a spectrum analyzer and compared this to the design specification from Matlab.

II. Tustin Transform Filter Design

We had to map an analogue filter into a discrete time version using the Tustin transform and the filter in question is shown in figure 1. The resistor value is 1kΩ, capacitor value is 1μF, and we use the fact that $s = j\omega$. Firstly, to find the transfer

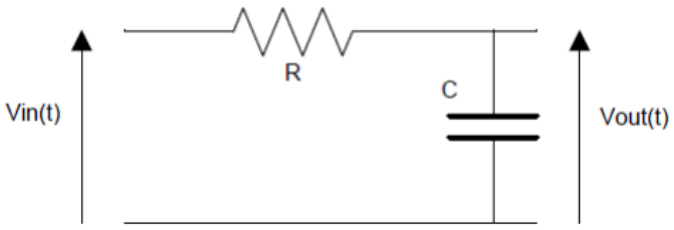

Fig. 1. First Order RC Analogue Circuit

function of the circuit as a Laplace transform, we used a simple potential divider technique:

$$\begin{aligned}\frac{Vout}{Vin}(s) &= \frac{\frac{1}{sC}}{R+\frac{1}{sC}} \\ &= \frac{1}{1+sRC}\end{aligned} \tag{1}$$

An approximation of the series expansion of the logarithm of the Tustin transform gives:

$$\begin{aligned}s &= \frac{2}{T_s}\frac{z-1}{z+1} \\ T_s &= \frac{1}{F_s}\end{aligned} \tag{2}$$

Since the cut-off frequency in this circuit is low in comparison to the sampling frequency which is set to 8kHz, a good approximation can be achieved without applying frequency warping. We used this transform to convert the Laplace domain transfer function into a Z-domain transfer function. The transfer function of an IIR filter is shown in figure 2 and we had to manipulate our equation until we had it in this form. After the substitution and further manipulation, our equation

$$\boldsymbol{H(z) = \frac{b_0 + b_1 Z^{-1} + b_2 Z^{-2} + \cdots + b_M Z^{-M}}{1 + a_1 Z^{-1} + a_2 Z^{-2} + \cdots + a_N Z^{-N}}}$$

Fig. 2. Transfer Function of IIR Filter

is of the form as shown in equation 3.

$$\begin{aligned}H(z) &= \frac{T_s z + T_s}{(T_s + 2RC)z + (T_s - 2RC)} \\ &= \frac{\frac{T_s}{T_s+2RC} + \frac{T_s}{T_s+2RC}z^{-1}}{1 + \frac{T_s-2RC}{T_s+2RC}z^{-1}} \\ &= \frac{\frac{1}{17} + \frac{1}{17}z^{-1}}{1 + \frac{-15}{17}z^{-1}}\end{aligned} \tag{3}$$

As we can see, this is in the form which matches figure 2 where $b_0 = \frac{1}{17}, b_1 = \frac{1}{17}, a_1 = \frac{-15}{17}$. The cut off frequency of the filter is given by $\frac{1}{2\pi RC}$ = 159Hz.

III. Implementation and Measurement of Response

To implement this design, we used the direct form I structure as shown in figure 3, D is a unit delay. In terms of implementation, this requires two delay lines done via circular buffers. The code is shown in listing 1.

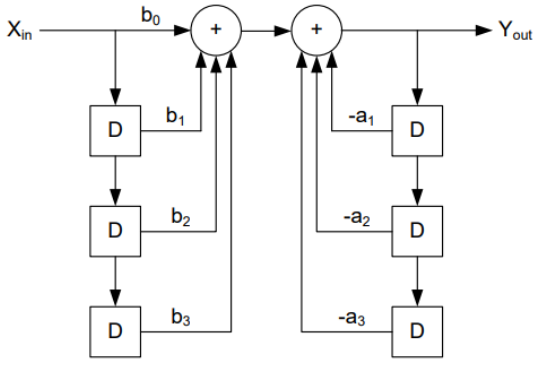

Fig. 3. Block Diagram of Direct Form I

```
void low_pass_filter(void);
{
    //Read sample from the codec
    xin = mono_read_16Bit();
    //Perform the difference equation
```

```
        yout = b[0]*xin + v[1]*b[1] - w[1]*a[1];
        //shift the values in a buffer of 2
        w[1] = w[0];
        v[1] = v[0];
        }
        //Store values in buffer
        v[1] = xin;
        w[1] = yout;
        }
    }
```

Listing 1. Low Pass Filter Implementation

We evaluated the design with the spectrum analyzer, the results are shown in figure 4. Due to the potential dividers in the hardware, it was very important to compare this with the all pass filter to check if our filter operates as expected. The cut off frequency obtained from the analogue filter was 159Hz and we see from this plot that the digital filter has a roll off around this frequency, therefore this verifies that the frequency response is as expected.

We also drove the input with a low frequency square wave

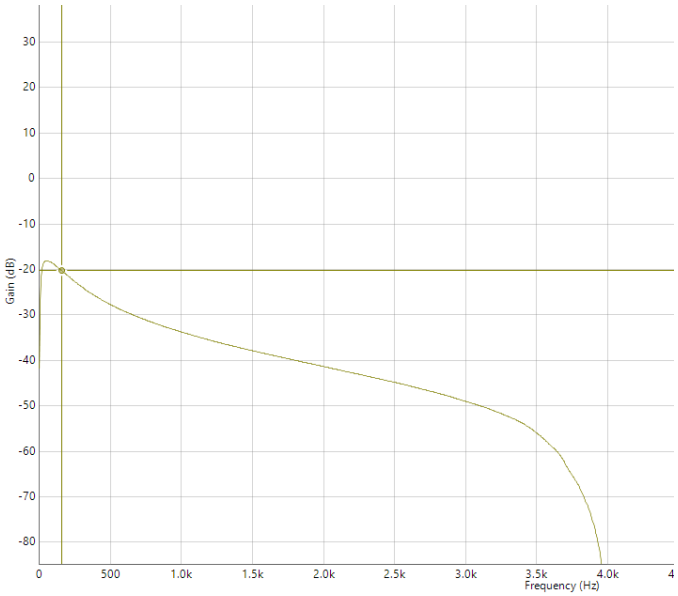

Fig. 4. Result of Low Pass Filter Implementation

to find the time constant of the filter. There is a high pass filter (HPF) on the input to the board so we had to choose the frequency of the square wave carefully so this HPF has little effect on the shape of the waveform. The time constant is the time required to charge the capacitor to 63% of the difference between the initial voltage value and the final one. The time constant of the analogue filter is given by $\tau = RC$ which gives 1ms and we measured the time constant of our digital filter by calculating the time decay to 63% of the final value and this gave us 0.8ms. Therefore, we can conclude that our designed digital filter has many similarities to the analogue filter counterpart.

IV. Comparison Analogue filter design

For a comparison to our IIR digital filter, we used the Tustin coefficients to design the filter in Matlab using the *freqz* function, the code for this is shown in listing 2. This code gives the frequency response of the first order low pass analogue filter.

```
fc = 159 % cut-off frequency
fs = 8000 % sampling frequency
% coefficients from Tustin transform filter design
b = [0.0588235 0.0588235]
a = [1 -0.8823529]
% frequency response
[h, f] = freqz(b, a, 5000, fs)
```

Listing 2. Analogue Filter in Matlab

Figure 5 displays a comparison between our designed digital filter, Matlab's designed first order analogue low pass filter as well as an all pass filter, this gives us an indication of the differences in the respective frequency responses.

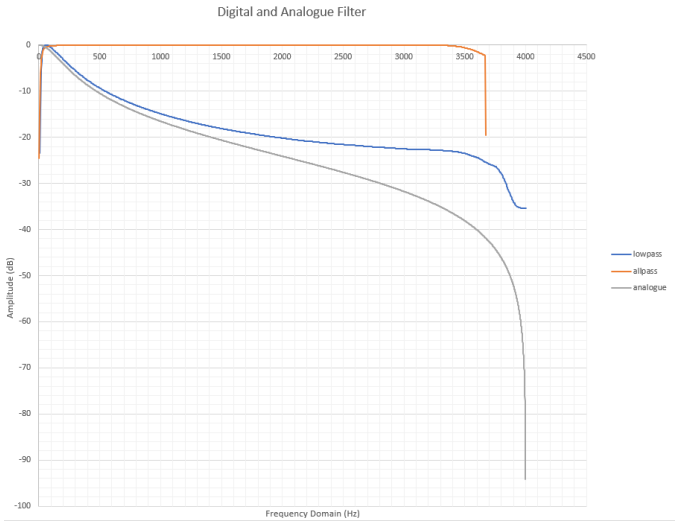

Fig. 5. Comparison Digital, Analogue and All-pass Filter

V. Direct form II Filter Design

We wanted to program an elliptic bandpass filter where order = 4th, passband = 270-450 Hz, passband ripple = 0.3db, stopband atenuation: 20db and this was done using the Matlab function *ellip*. Firstly, we calculated the coefficients needed for this design in Matlab and used them in a way which is suitable for inclusion in a C program. These coefficients are stored in two arrays, ***b[]*** and ***a[]***. We implemented this in real-time using a direct form II structure and a diagram of this setup is shown in figure 6.

Implementation of direct form II transpose will be discussed in the next section. The order of the two parts of the transfer function is swapped compared to direct form I but without changing the transfer from input to output. This diagram has been made more efficient by using only one delay line so fewer computations are required.

The elliptic filter used has ripple in both the pass and stop bands but has the sharpest cut off of all the common filter types, this roll off is caused by additional zeros on the imaginary axis in the Z-domain. Listing 3 shows the Matlab

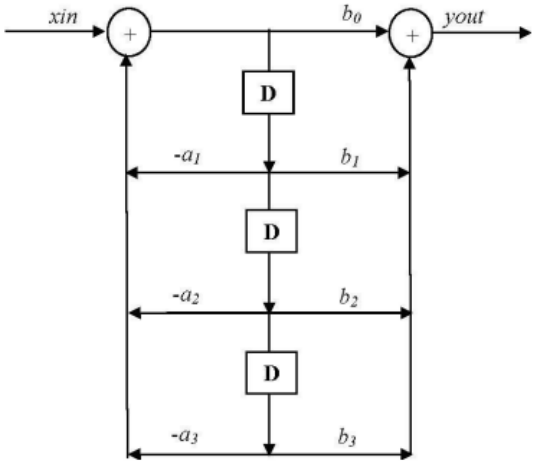

Fig. 6. Block Diagram of Direct Form II

code used to generate the filter coefficients according to the design specification.

```
% use elliptic filter design
[b, a] = ellip(6, .3, 20, [270/4000 450/4000])
% frequency response
[h,f] = freqz(b,a, 1204, 8000)
```

Listing 3. Elliptic Filter Design

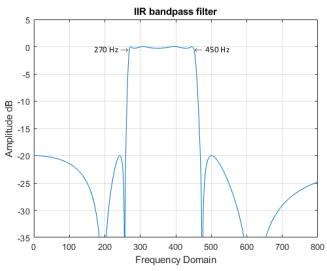

Fig. 7. Design of IIR filter in Matlab

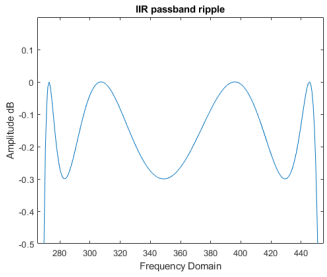

Fig. 8. Pass band ripple of IIR design

Figures 7 and 8 show the specification of the designed Matlab filter including passband and stopband frequencies and ripples. For implementation in real-time, we created two arrays, ***v[]*** and ***w[]*** which we initialised to zero in the main shown in listing 4.

```
for (i=0; i < order; i++){
    v[i] = 0;
    w[i] = 0;
}
```

Listing 4. Initialise new arrays

Listing 5 displays the code for our *bandpass* function which is where we implemented the code in direct form II.

```
double bandpass(void)
{
    //first coefficient multiplication
    yout = b[0]*v[0];
    //multiply each v value and w value by
        corresponding coefficients
    for (k = order; k > 0; k--){
        //implement the IIR difference equation
        yout += v[k]*b[k] - w[k]*a[k];
        //delay operator for w
        w[k] = w[k-1];
        //delay element for v
        v[k] = v[k-1];
    }
    //first delayed v element stores input read
    v[1] = v[0];
    //output of difference equation stored in
        first delayed w element
    w[1] = yout;
    return yout;
}
```

Listing 5. Matlab filter design

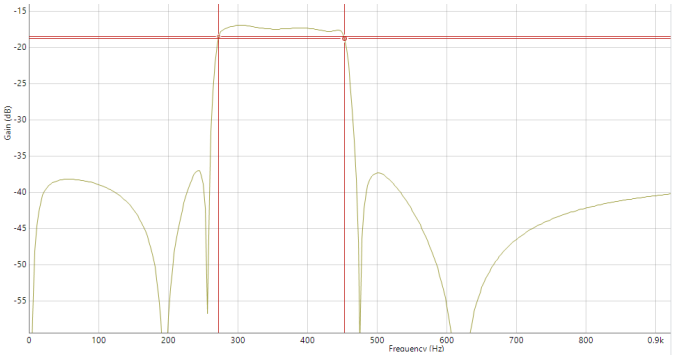

Fig. 9. Performance of the IIR Filter

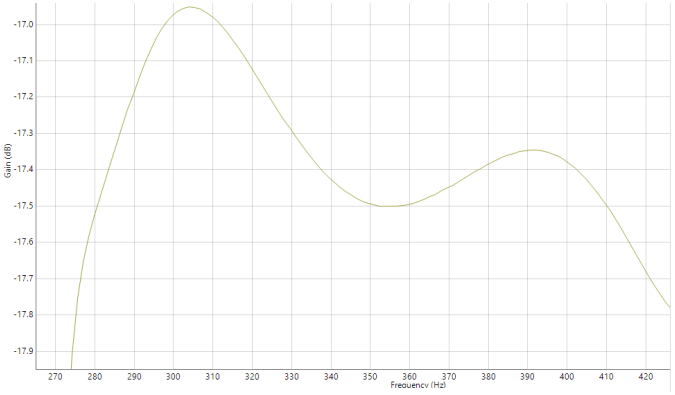

Fig. 10. Result of the passband ripple

Figures 9 and 10 show the frequency response plot to our designed filter and as we compare this to the Matlab plot, the results closely match the expected response in terms of passband frequency and ripple. The major difference is an offset in gain and this is due to internal potential dividers at each input port of the DSK board which has an effect of attenuating the gain.

VI. Direct Form II Transposed Filter Design

Performance is a major concern in R-DSP so it is important to look into more efficient algorithms for filter designs. The direct form II transposed filter design is based on the rules

that a network is unchanged in behaviour if you reverse the direction of each branch, interchange branch divisions and branch summations, swap input for output. The structure of this type of implementation is shown in figure 11.

The coding for a transposed filter is slightly more efficient

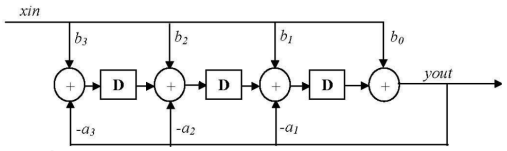

Fig. 11. Block Diagram of Direct Form II Transposed

than the untransformed direct forms because shifting of the samples is inherent in the calculations for the output. From this we derived the code in listing 6.

Firstly we had to add the first coefficient multiplied by the current sample. Then the difference equation is calculated whilst updating the buffers. This is where we should see an increase in performance compared to the non transpose algorithm. Finally, both input and output samples are stored in the buffer.

```
    void dir2Tfilter(void)
{
    xin = mono_read_16Bit(); //Read the sample
    //first coefficient multiplication
    yout = b[0]*xin;
    for (k=order; k>0;k--){
        yout += v[k]*b[k]- w[k]*a[k];
        //delay operator for w
        w[k] = w[k-1];
        //delay operator for v
        v[k] = v[k-1];
    }
    //store in the buffers
    v[1] = xin;
    w[1] = yout;
}
```

Listing 6. Implementation of direct form II transpose structure

The frequency response of the filter with transpose structure was then measured with a spectrum analyser. The results are shown in figures 12 and 13. This verifies the filter frequency response agrees with the Matlab prediction.

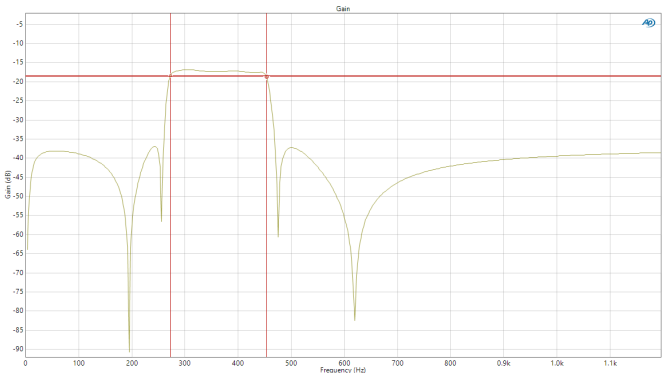

Fig. 12. Performance of the Filter

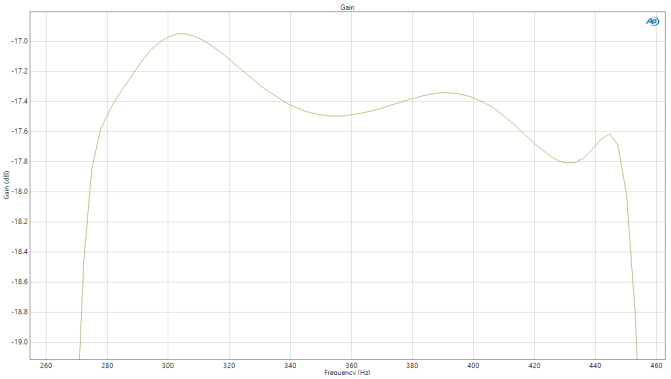

Fig. 13. Result of the Passband Ripple

VII. Performance Comparison

After implementing the different algorithms the performances were evaluated by measuring the amount of instruction clock cycles. This was done using the profiling clock method to determine how many instruction cycles per sample are needed for a filter of order n in the form $A + Bn$. The count includes only the instructions between the calls to *mono_read_16Bit* and *mono_write_16Bit*.

The compiler optimizations have a different effect on

	No optimization		o0		o2	
	Transpose	Direct	Transpose	Direct	Transpose	Direct
1	814	905	818	905	289	498
2	742	845	758	844	289	497
3	742	845	758	844	288	497

TABLE I
Instruction Cycles Count with Different Optimization Options of Each Implementation

the performance for different filter orders. It would be interesting to see which optimization level performs best for which filter order. Therefore we have evaluated the performance of a 2, 4 and 6 order filter with no, o0 and o2 compiler optimizations. Also we have differentiated between the normal direct form and the transposed direct form algorithm. The results are shown in figure 14.

Fig. 14. Optimization for Different Filter Orders