ECE444: HW6

Thomas Smallarz

November 15, 2020

Contents

Design

Steps

From Chapter 2 and Chapter 8 the general flow of designing this filter will be:

1. Compute Poles/Zeros for CT Lowpass Inverse Chebyshev Prototype filter (based on Ex. 2.17)
2. Compute pre-warped frequencies using simplified method (based on Ex. 8.7)
3. Compute Poles/Zeros for DT Bandpass Inverse Chebyshev filter H(z) (based on Ex. 8.7)
4. Sort DT Poles/Zeros so that Poles/Zeros are can be matched with their conjugate, and are ordered so pairs of Poles/Zeros that are nearest each other can be created into 2^{nd} order systems
5. Combine conjugate pairs to create second order systems
6. Output coefficients to header file

Computed Poles/Zeros for CT Lowpass Inverse Chebyshev Prototype filter

pk:	zk:
-0.0118 + 1.0050i	0.0000 - 1.0164i
-0.0367 + 1.0292i	0.0000 - 1.0421i
-0.0665 + 1.0805i	0.0000 - 1.0968i
-0.1061 + 1.1656i	0.0000 - 1.1884i
-0.1645 + 1.2969i	0.0000 - 1.3326i
-0.2605 + 1.4976i	0.0000 - 1.5602i
-0.4391 + 1.8114i	0.0000 - 1.9393i
-0.8321 + 2.3201i	0.0000 - 2.6479i
-1.9207 + 3.1040i	0.0000 - 4.3406i
-5.2833 + 2.7990i	0.0000 -12.9149i
-5.2833 - 2.7990i	0.0000 +12.9149i
-1.9207 - 3.1040i	0.0000 + 4.3406i
-0.8321 - 2.3201i	0.0000 + 2.6479i
-0.4391 - 1.8114i	0.0000 + 1.9393i
-0.2605 - 1.4976i	0.0000 + 1.5602i
-0.1645 - 1.2969i	0.0000 + 1.3326i
-0.1061 - 1.1656i	0.0000 + 1.1884i
-0.0665 - 1.0805i	0.0000 + 1.0968i
-0.0367 - 1.0292i	0.0000 + 1.0421i
-0.0118 - 1.0050i	0.0000 + 1.0164i

```
%% Design Low-Pass Inverse Chebyshev Prototype Filter with normalized freq
% Based on Ex. 2.17
alphap = 2; % pass-band alpha of 2dB
alphas = 20; % stop-band alpha of 20dB
omegap = 1; % prototype filter cutoff freq of 1 rad/sec
% calculate stop-band frequency based on alpha's and omega
omegas = omegap*cosh(acosh(sqrt((10^(alphas/10)-1)/(10^(alphap/10)-1)))/K);
epsilon = 1/sqrt(10^(alphas/10)-1);
k = 1:K;
% calculate poles
pk = -omegap*sinh(asinh(1/epsilon)/K)*sin(pi*(2*k-1)/(2*K))+...
    1j*omegap*cosh(asinh(1/epsilon)/K)*cos(pi*(2*k-1)/(2*K));
pk = omegap*omegas./pk;
% calculate zeros
zk = 1j*omegas.*sec(pi*(2*k-1)/(2*K));
% calculate coefficients of expanded form based on poles/zeros
B = prod(pk./zk)*poly(zk); A = poly(pk);
```

Compute pre-warped frequencies using simplified method

```
fp1 = F(k);
fp2 = Fs / 2 - fp1; % upper pass band frequencies (Hz)

wp1 = 2*pi.*fp1; wp2 = 2*pi.*fp2; % convert from Hz to Rad/Sec
wp1_w = tan(wp1.*T/2); % SIMPLIFIED procedure for pre-warped lower passband omegas
wp2_w = tan(wp2.*T/2); % SIMPLIFIED procedure for pre-warped upper passband omegas

c1 = (wp1_w*wp2_w - 1) / (wp1_w*wp2_w + 1);
c2 = (wp2_w - wp1_w) / (wp1_w*wp2_w + 1);
```

Compute Poles/Zeros for DT Bandpass Inverse Chebyshev filter H(z)

Zdig:	Pdig:
0.3136 - 0.9496i	-0.3094 - 0.9473i
-0.3136 - 0.9496i	0.3094 - 0.9473i
-0.3207 - 0.9472i	0.3138 - 0.9382i
0.3207 - 0.9472i	-0.3138 - 0.9382i
-0.3357 - 0.9420i	0.3251 - 0.9255i
0.3357 - 0.9420i	-0.3251 - 0.9255i
-0.3602 - 0.9329i	0.3440 - 0.9073i
0.3602 - 0.9329i	-0.3440 - 0.9073i
-0.3973 - 0.9177i	-0.3718 - 0.8803i
0.3973 - 0.9177i	0.3718 - 0.8803i
-0.4522 - 0.8919i	0.4105 - 0.8388i
0.4522 - 0.8919i	-0.4105 - 0.8388i
0.5331 - 0.8460i	-0.4614 - 0.7721i

-0.5331 - 0.8460i	0.4614 - 0.7721i
0.6522 - 0.7581i	-0.5224 - 0.6613i
-0.6522 - 0.7581i	0.5224 - 0.6613i
0.8158 - 0.5784i	-0.5806 - 0.4753i
-0.8158 - 0.5784i	0.5806 - 0.4753i
0.9728 - 0.2318i	-0.6092 - 0.1819i
-0.9728 - 0.2318i	0.6092 - 0.1819i
0.9728 + 0.2318i	-0.6092 + 0.1819i
-0.9728 + 0.2318i	0.6092 + 0.1819i
0.8158 + 0.5784i	0.5806 + 0.4753i
-0.8158 + 0.5784i	-0.5806 + 0.4753i
0.6522 + 0.7581i	-0.5224 + 0.6613i
-0.6522 + 0.7581i	0.5224 + 0.6613i
0.5331 + 0.8460i	0.4614 + 0.7721i
-0.5331 + 0.8460i	-0.4614 + 0.7721i
-0.4522 + 0.8919i	0.4105 + 0.8388i
0.4522 + 0.8919i	-0.4105 + 0.8388i
0.3973 + 0.9177i	-0.3718 + 0.8803i
-0.3973 + 0.9177i	0.3718 + 0.8803i
0.3602 + 0.9329i	-0.3440 + 0.9073i
-0.3602 + 0.9329i	0.3440 + 0.9073i
0.3357 + 0.9420i	-0.3251 + 0.9255i
-0.3357 + 0.9420i	0.3251 + 0.9255i
0.3207 + 0.9472i	0.3138 + 0.9382i
-0.3207 + 0.9472i	-0.3138 + 0.9382i
0.3136 + 0.9496i	0.3094 + 0.9473i
-0.3136 + 0.9496i	-0.3094 + 0.9473i

```
% computing zeros/poles for H(z)
for i = 1:length(zk)
   Zdig(i,:) = roots([1, 2*c1./(1-c2*zk(i)), (1+c2*zk(i))./(1-c2*zk(i))]);
end

for i = 1:length(pk)
   Pdig(i,:) = roots([1, 2*c1./(1-c2*pk(i)), (1+c2*pk(i))./(1-c2*pk(i))]);
end
```

Sort DT Poles/Zeros so that Poles/Zeros are can be matched with their conjugate, and are ordered so pairs of Poles/Zeros that are nearest each other can be created into 2nd order systems

```
% appending columns of zeros, then sorting by imag value
temp = -j.*Zdig;
temp2 = sort([temp(:,1);temp(:,2)],'ComparisonMethod','real');
Zdig_sort = j.*temp2;
clear temp temp2;

temp = -j.*Pdig;
temp3 = sort([temp(:,1);temp(:,2)],'ComparisonMethod','real');
Pdig_sort = j.*temp3;
clear temp temp2;
```

Combine conjugate pairs to create second order systems

```
% create 2nd order real systems to cascade
I = length(Zdig_sort);
for i = 1:I/2
    Zdig2(i,:) = poly([Zdig_sort(i,1),conj(Zdig_sort(i,1))]);
end

I = length(Pdig_sort);
for i = 1:I/2
    Pdig2(i,:) = poly([Pdig_sort(i,1),conj(Pdig_sort(i,1))]);
end
```

Output coefficients to header file

```
B_coef = B(:,:,2);
A_coef = A(:,:,[2 3]);
```

```
%% Create Header File
GenerateHeader(B_coef,A_coef,Gain);
```

Plots

Compute H(z)

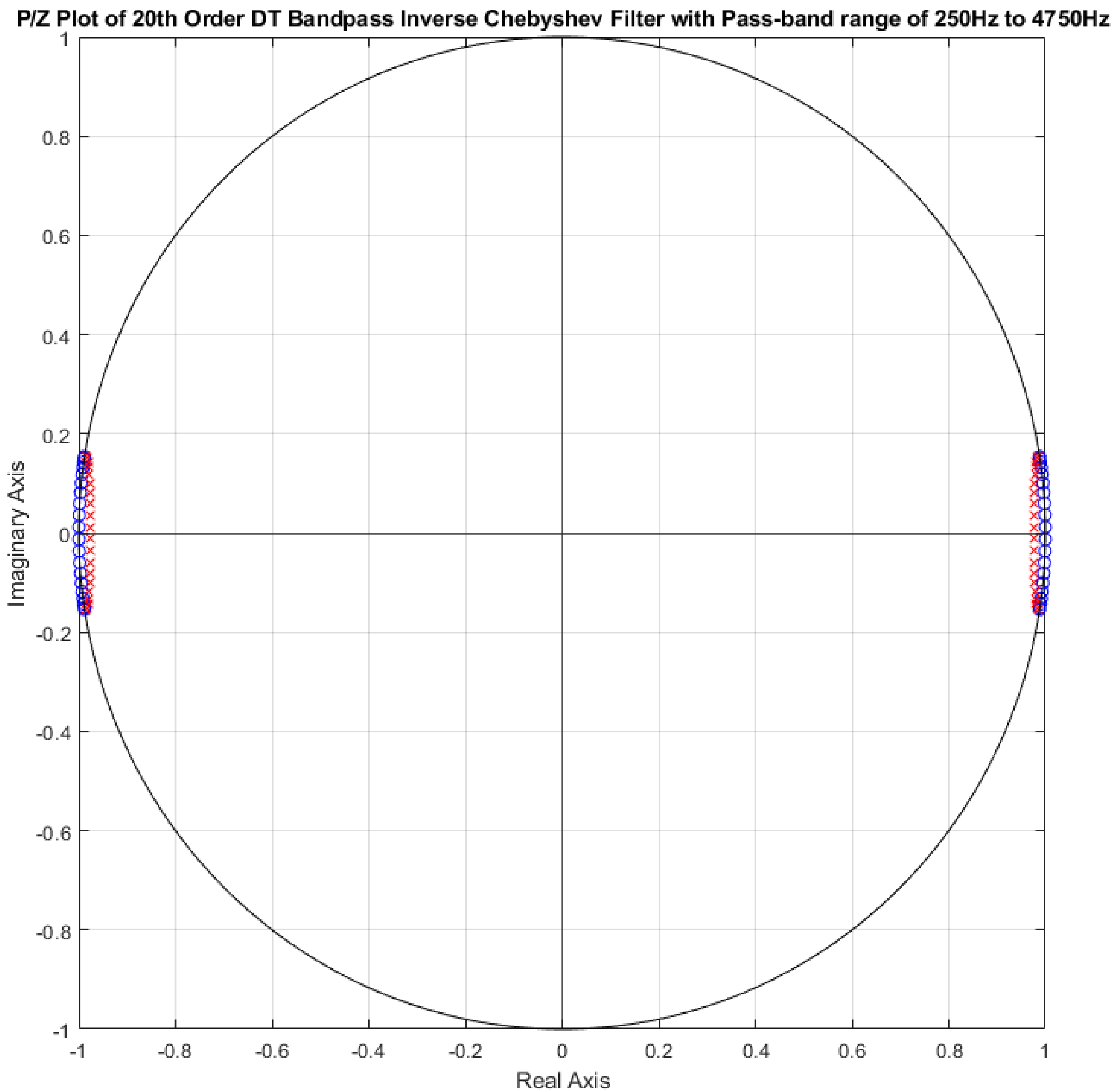
```
Omega = linspace(0,pi,10001); H = 1;
% evaluate all 2nd order Zero polynomials over 0->Pi
for i = 1:length(Zdig2)
    H = H .* polyval(Zdig2(i,:),exp(1j*Omega));
end
% evaluate all 2nd order Pole polynomials over 0->Pi
for i = 1:length(Pdig2)
    H = H ./ polyval(Pdig2(i,:),exp(1j*Omega));
end
% multiply by gain factor
G = B(1)/A(1)*prod(1/c2-zk)/prod(1/c2-pk);
Gain = [Gain;G];
H = H.*G;
```

Plotting

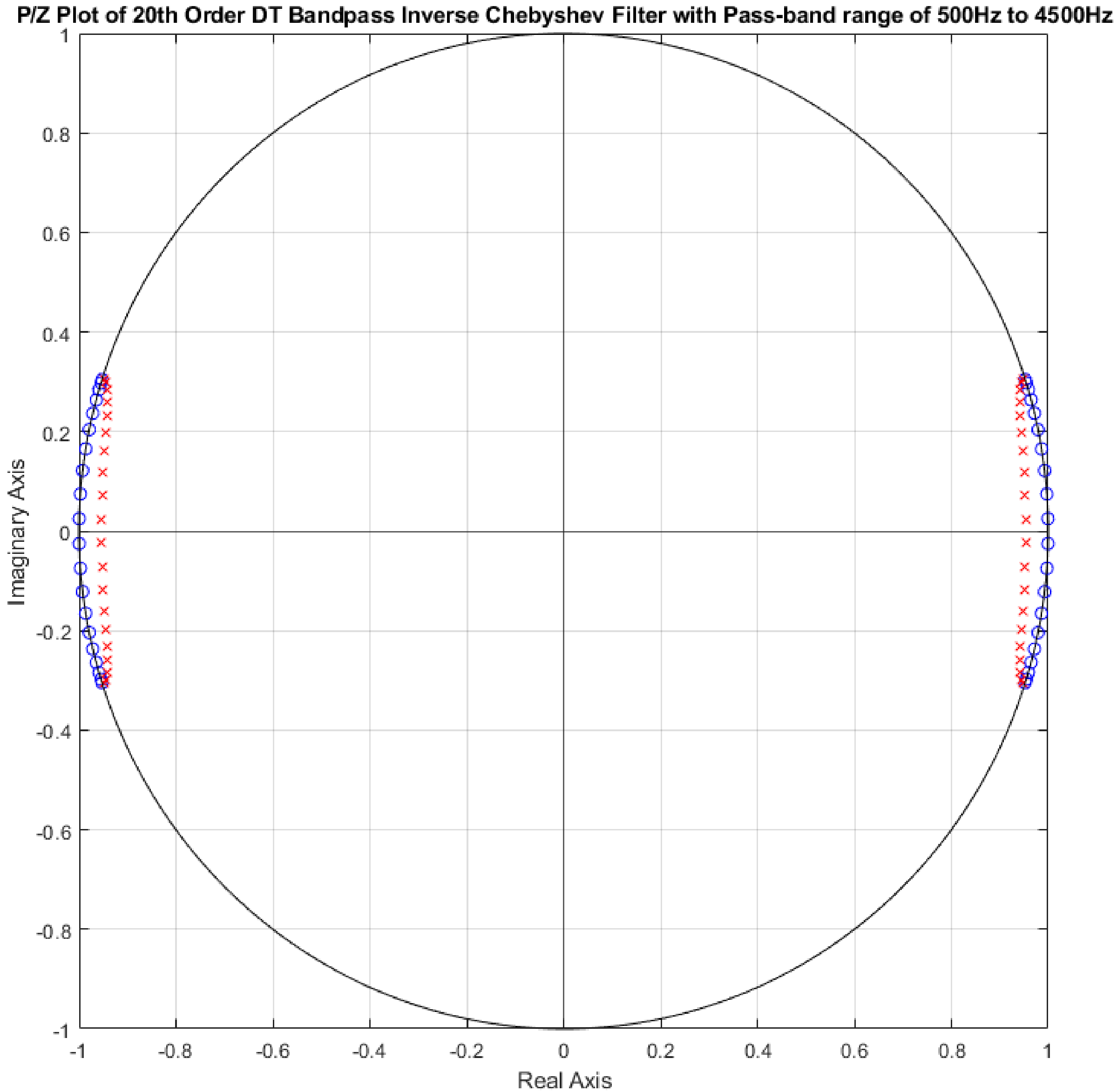
```
% |H(z)|
figure(2*k-1); set(gcf,'Position',[970+20*k,200-30*k,820,800]);
subplot(2,1,1);
plot(Omega/T,abs(H),'k-'); axis([0 pi/T -0.05 1.05]);
xlabel("\omega"); ylabel("|H(z)|");
title("Frequency Response of " + K +"th Order DT BP Inv. Ch. with passband

% 20log10(H(z))
subplot(2,1,2);
plot(Omega/T,20*log10(abs(H)),'k-');
xlabel("\omega"); ylabel("20log_1_0|H(z)| (dB)");
title("Frequency Response of " + K +"th Order DT BP Inv. Ch. with passband
figure(2*k); set(gcf,'Position',[20+20*k,200-30*k,820,800]);
plot(real(Zdig(:)),imag(Zdig(:)),'bo'); hold on;
plot(real(Pdig(:)),imag(Pdig(:)),'rx'); hold on;

plot(real(exp(j.*[0:0.001:2*pi])),imag(exp(j.*[0:0.001:2*pi])),'k');

title("P/Z Plot of " + K + "th Order DT Bandpass Inverse Chebyshev Filter '
grid on; axis([-1 1 -1 1]); xlabel("Real Axis"); ylabel("Imaginary Axis");
xline(0); yline(0);
```

With $f_{lower\ passband} = [250\ \ 500\ \ 1000\ \ 2000]$ and $K = 20$

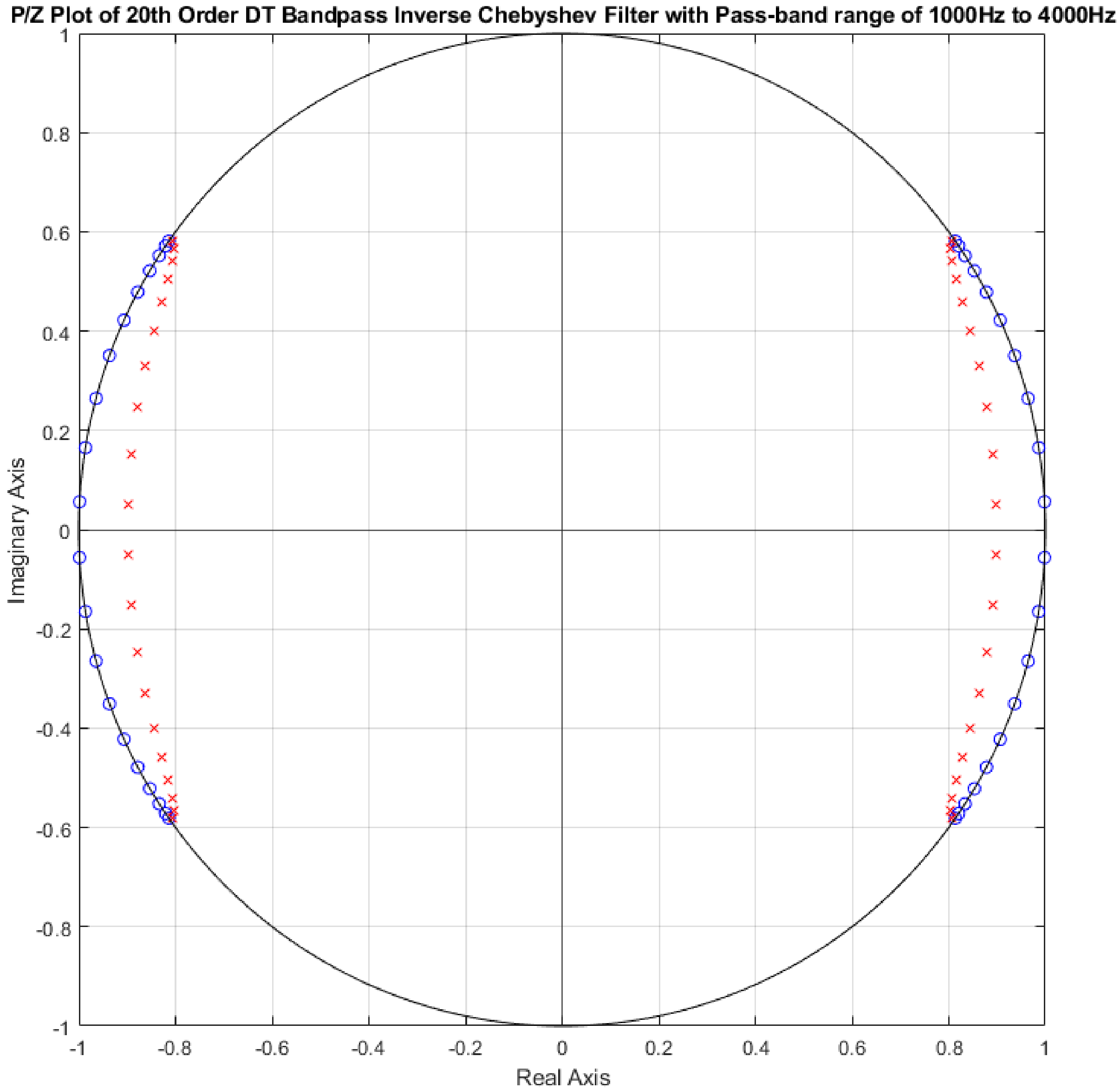
P/Z Plots:

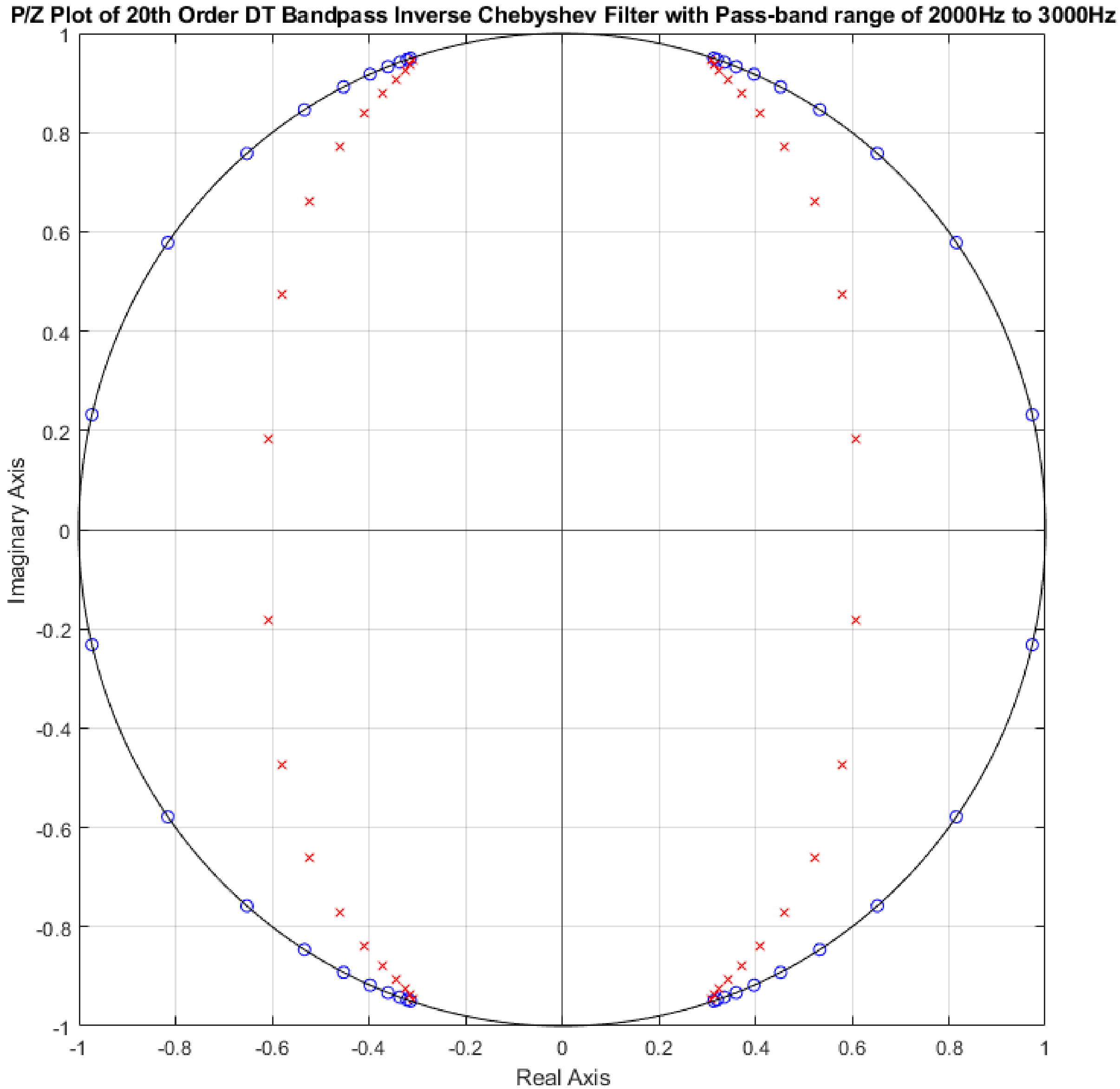


P/Z Plot of 20th Order DT Bandpass Inverse Chebyshev Filter with Pass-band range of 250Hz to 4750Hz
Imaginary Axis
Real Axis
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

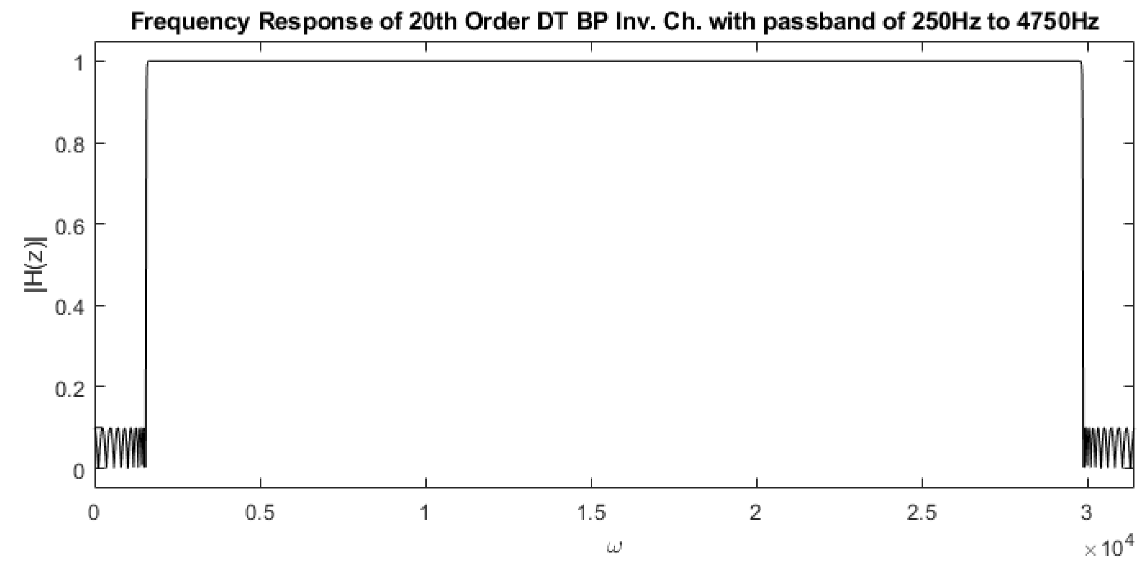


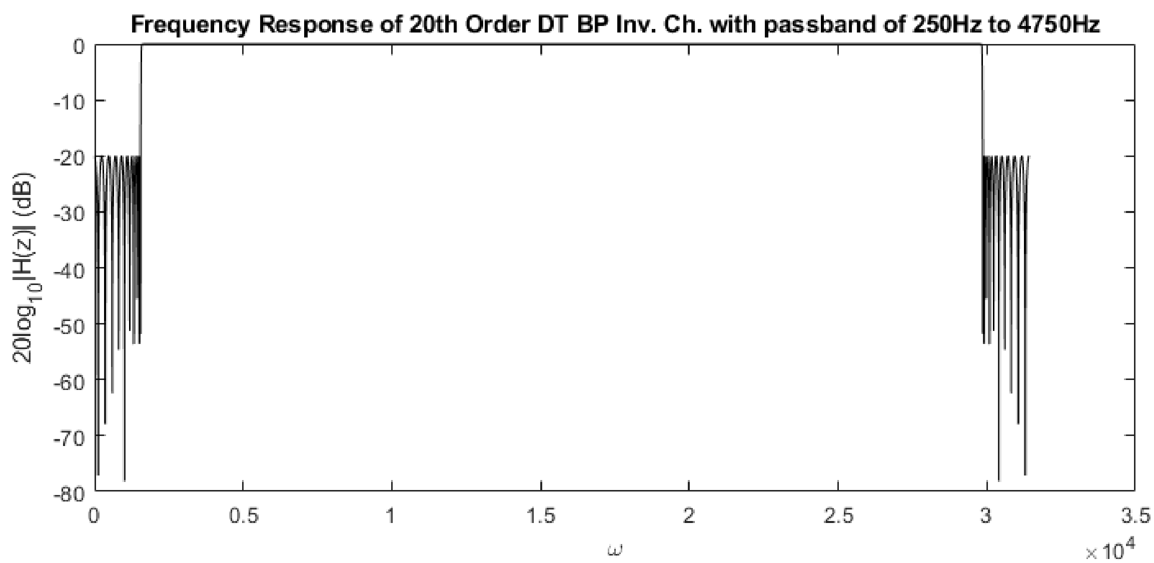
P/Z Plot of 20th Order DT Bandpass Inverse Chebyshev Filter with Pass-band range of 500Hz to 4500Hz
Imaginary Axis
Real Axis
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

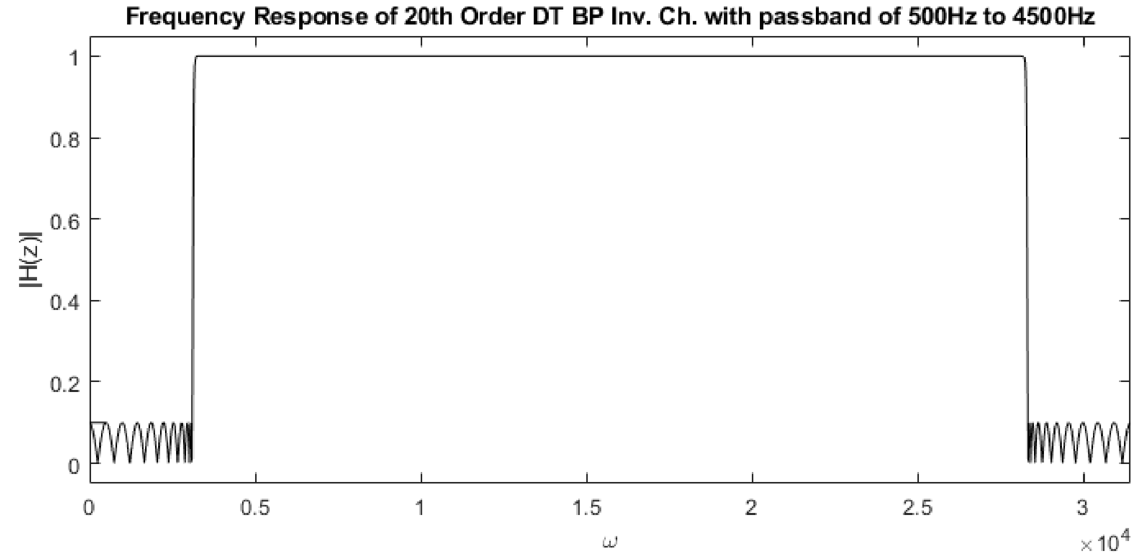

P/Z Plot of 20th Order DT Bandpass Inverse Chebyshev Filter with Pass-band range of 1000Hz to 4000Hz
Imaginary Axis
Real Axis
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

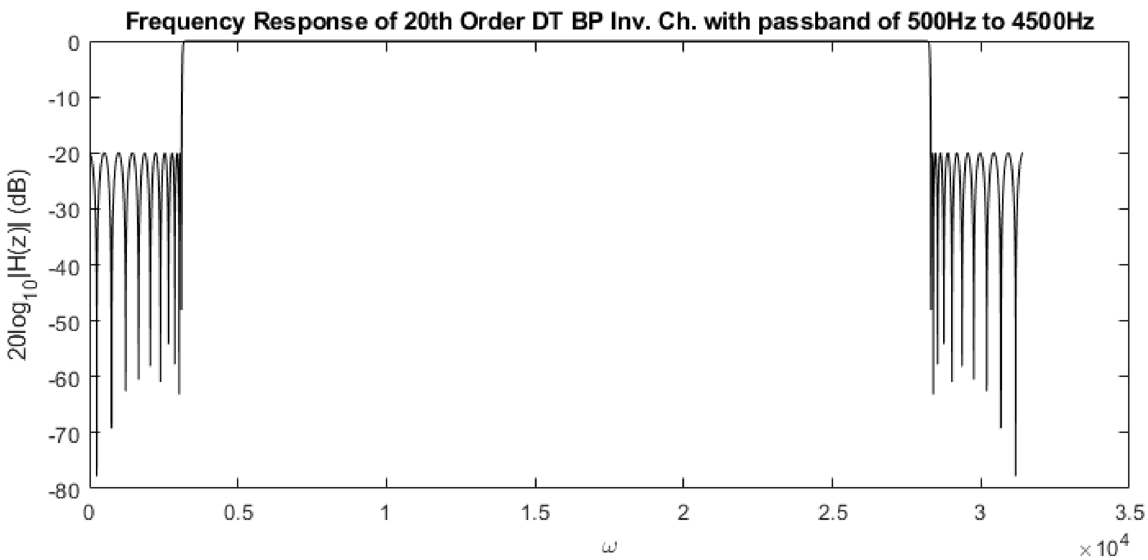


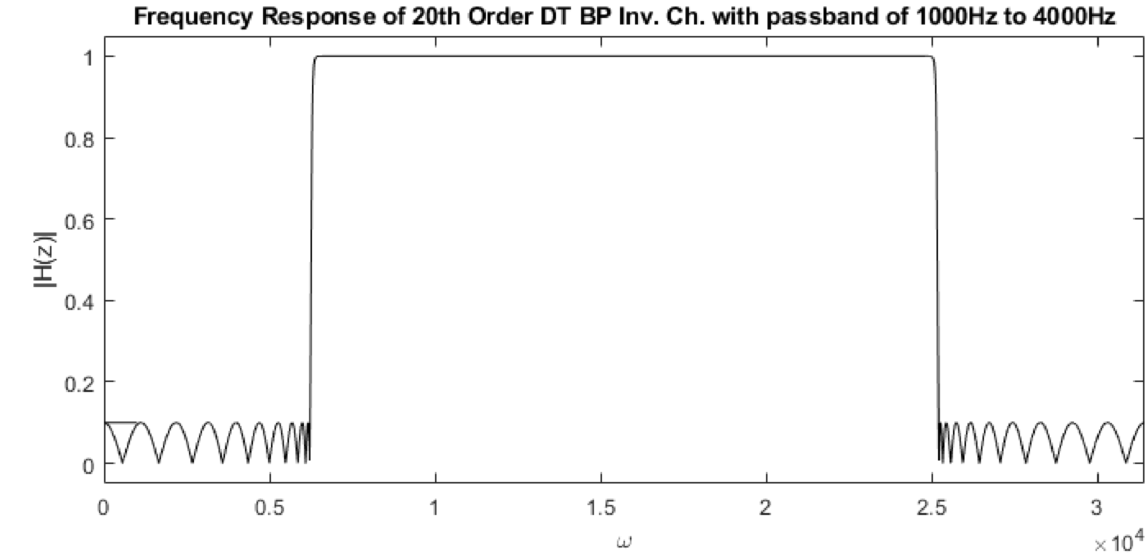
Magnitude Response Plots

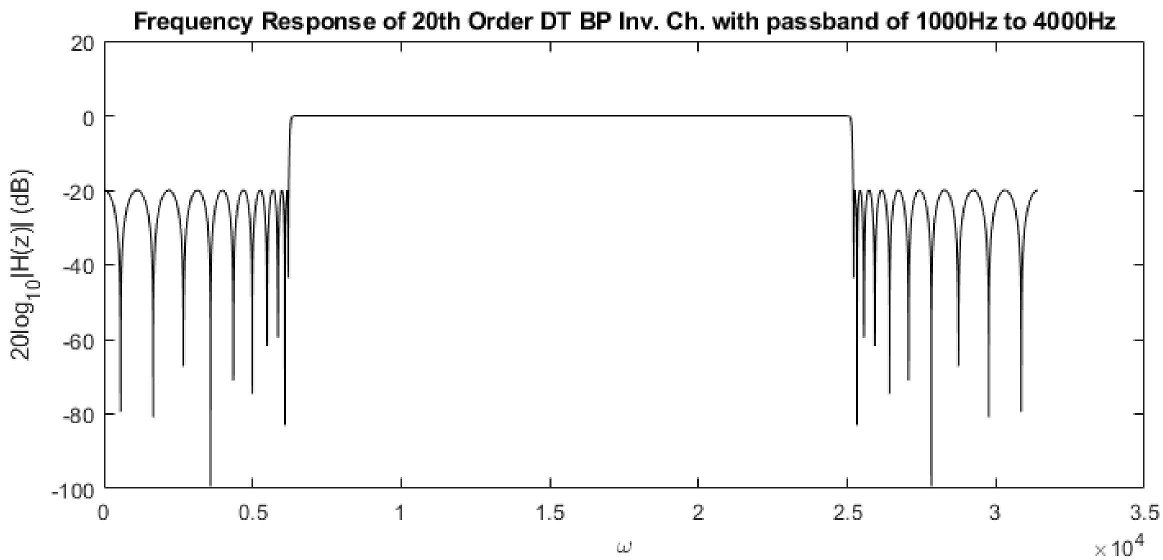

Frequency Response of 20th Order DT BP Inv. Ch. with passband of 250Hz to 4750Hz
|H(z)|
1
0.8
0.6
0.4
0.2
0
0.5
1
1.5
2
2.5
3
ω
×10^4

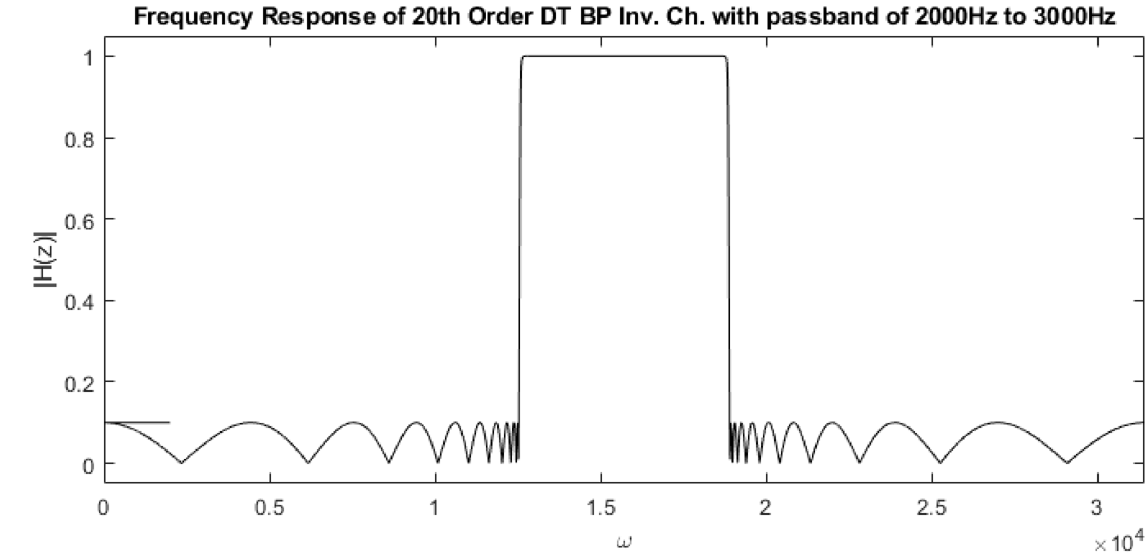

Frequency Response of 20th Order DT BP Inv. Ch. with passband of 250Hz to 4750Hz
20log10|H(z)| (dB)
0
-10
-20
-30
-40
-50
-60
-70
-80
0.5
1
1.5
2
2.5
3
3.5
ω
×10^4

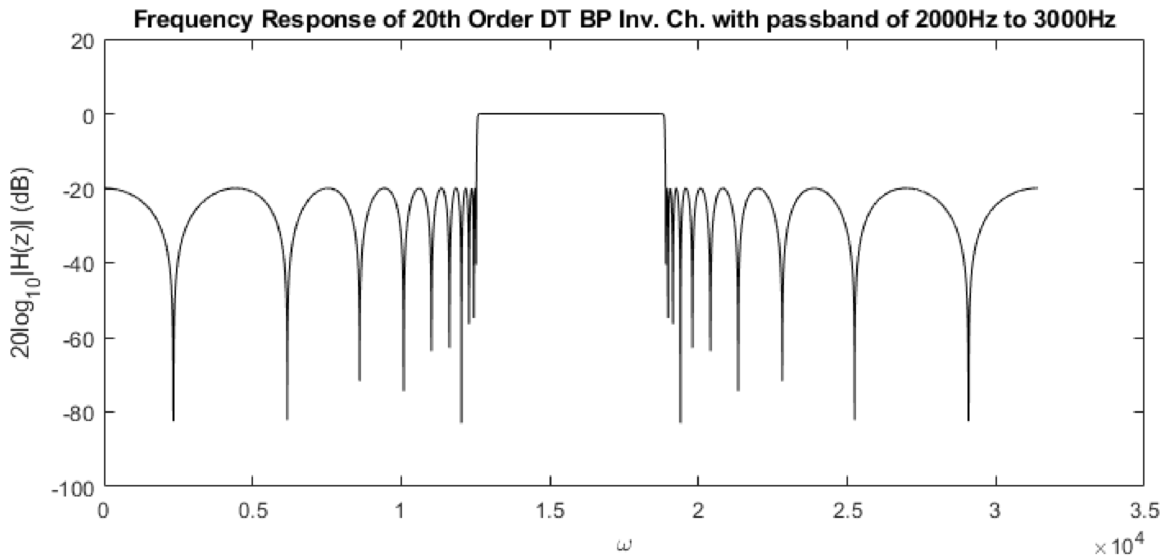

Frequency Response of 20th Order DT BP Inv. Ch. with passband of 500Hz to 4500Hz
|H(z)|
ω
×10^4


Frequency Response of 20th Order DT BP Inv. Ch. with passband of 500Hz to 4500Hz
20log10|H(z)| (dB)
ω
×10^4


Frequency Response of 20th Order DT BP Inv. Ch. with passband of 1000Hz to 4000Hz
|H(z)|
ω
×10^4
0
0.2
0.4
0.6
0.8
1
0.5
1.5
2
2.5
3


Frequency Response of 20th Order DT BP Inv. Ch. with passband of 1000Hz to 4000Hz
20log10|H(z)| (dB)
ω
×10^4
20
0
-20
-40
-60
-80
-100
0.5
1
1.5
2
2.5
3
3.5





Filter Coefficients

Outputted coef.h header file:

```
#define K 20
float G[4] = {0.743159, 0.552154, 0.304332, 0.091647};
float B[4][K] = {
  {  1.976152,  -1.976152,  1.977292,  -1.977292,
     1.979467,  -1.979467,  1.982471,  -1.982471,  1.986021,
     -1.986021,  1.989774,  -1.989774,  1.993363,  -1.993363,
     1.996432,  -1.996432,  -1.998670,  1.998670,  -1.999850,
     1.999850   },
  {  -1.905031,  1.905031,  -1.909341,  1.909341,
     -1.917621,  1.917621,  -1.929195,  1.929195,  -1.943073,
     1.943073,  -1.957993,  1.957993,  -1.972507,  1.972507,
```

```
    -1.985110,  1.985110,   -1.994420,  1.994420,   -1.999367,
    1.999367    },
  {   1.627070,   -1.627070,  1.640619,   -1.640619,
    1.667352,   -1.667352,  -1.706380,  1.706380,   -1.755964,
    1.755964,   1.813064,   -1.813064,  -1.872881,  1.872881,
    -1.928712,  1.928712,   1.972558,   -1.972558,  -1.996843,
    1.996843    },
  {   -0.627195,  0.627195,   0.641418,   -0.641418,
    0.671374,   -0.671374,  0.720436,   -0.720436,  0.794665,
    -0.794665,  0.904334,   -0.904334,  -1.066225,  1.066225,
    -1.304377,  1.304377,   -1.631500,  1.631500,   -1.945521,
    1.945521    }
};

float A[4][K][2] = {
  {
    {1.972075,  0.996411},
    {-1.972075, 0.996411},
    {1.966230,  0.989347},
    {-1.966230, 0.989347},
    {1.961672,  0.982558},
    {-1.961672, 0.982558},
    {1.958335,  0.976188},
    {-1.958335, 0.976188},
    {1.956074,  0.970381},
    {-1.956074, 0.970381},
    {1.954692,  0.965277},
    {-1.954692, 0.965277},
    {1.953961,  0.961006},
    {-1.953961, 0.961006},
    {1.953655,  0.957687},
    {-1.953655, 0.957687},
    {1.953576,  0.955415},
    {-1.953576, 0.955415},
    {1.953576,  0.954262},
    {-1.953576, 0.954262}
  },
  {
    {-1.896539, 0.993180},
    {1.896539,  0.993180},
    {-1.888005, 0.979751},
    {1.888005,  0.979751},
    {-1.883692, 0.966733},
    {1.883692,  0.966733},
    {-1.883049, 0.954308},
    {1.883049,  0.954308},
    {-1.885301, 0.942711},
    {1.885301,  0.942711},
    {-1.889498, 0.932235},
    {1.889498,  0.932235},
    {-1.894598, 0.923219},
    {1.894598,  0.923219},
    {-1.899561, 0.916027},
```

```
        {1.899561,  0.916027},
        {-1.903460, 0.911003},
        {1.903460,  0.911003},
        {1.905598,  0.908416},
        {-1.905598, 0.908416}
    },
    {
        {1.611899,  0.988960},
        {-1.611899, 0.988960},
        {1.607148,  0.966968},
        {-1.607148, 0.966968},
        {1.614450,  0.944690},
        {-1.614450, 0.944690},
        {1.632260,  0.921800},
        {-1.632260, 0.921800},
        {-1.658497, 0.898257},
        {1.658497,  0.898257},
        {1.690352,  0.874464},
        {-1.690352, 0.874464},
        {1.724173,  0.851446},
        {-1.724173, 0.851446},
        {1.755577,  0.830938},
        {-1.755577, 0.830938},
        {1.779945,  0.815234},
        {-1.779945, 0.815234},
        {1.793296,  0.806640},
        {-1.793296, 0.806640}
    },
    {
        {0.618702,  0.993124},
        {-0.618702, 0.993124},
        {-0.627601, 0.978774},
        {0.627601,  0.978774},
        {-0.650139, 0.962262},
        {0.650139,  0.962262},
        {-0.687922, 0.941469},
        {0.687922,  0.941469},
        {0.743690,  0.913169},
        {-0.743690, 0.913169},
        {-0.821035, 0.872033},
        {0.821035,  0.872033},
        {0.922759,  0.809050},
        {-0.922759, 0.809050},
        {1.044852,  0.710215},
        {-1.044852, 0.710215},
        {1.161269,  0.563015},
        {-1.161269, 0.563015},
        {1.218333,  0.404174},
        {-1.218333, 0.404174}
    }
};

float buf[K][2] = {
```

{0,0}, {0,0}, {0,0}, {0,0}, {0,0}, {0,0}, {0,0}, {0,0},
{0,0}, {0,0}, {0,0}, {0,0}, {0,0}, {0,0}, {0,0}, {0,0},
{0,0}, {0,0}, {0,0}, {0,0}
};

Implementation

Initial C-code

Initially in MATLAB:

```
x = X(n*T); % adc input
x = real(Gain(filter_select))*x;
for k = 1:K % for each stage
    v = x - A_coef(filter_select,k,1).*buf(k,1) - A_coef(filter_select,k,2).*buf(k,2);
    y = v + B_coef(filter_select,k,1).*buf(k,1) + buf(k,2);
    buf(k,2) = buf(k,1);
    buf(k,1) = v;
    x = y;
end

Y = [Y;y];
```

In Keil uVision IDE:

Main Interrupt

```
void PIT0_IRQHandler(void){ //This function is called when the timer interrupt expires
  GPIOA->PSOR |= GPIO_PSOR_PTSO(0x1u << 1); // Turn on Red LED
  ADC0->SC1[0] &= 0xE0;    //Start conversion of channel 1
  adc_meas = ADC0->R[0];   //Read channel 1 after conversion

  if(fs!=0x4){ //if filter select is not set to digital wire
    x = (float)adc_meas;

    for(k=0;k<K;k++){ //for each stage of filter
      v = x - A[fs][k][0]*buf[k][0] - A[fs][k][1]*buf[k][1]; //compute 'middle-man' variable 'v'
      y = v + B[fs][k]*buf[k][0] + buf[k][1]; //compute output of filter stage y
      buf[k][1] = buf[k][0]; //cycle buffer (n-1) to (n-2)
      buf[k][0] = v; //input 'v' into buffer (n-1)
      x = y; //set output of stage to new input
    }

    adc_meas = (uint16_t)(y+2048); // mask y for output
  } //end of if statement

  DAC0->DAT[0].DATL = DAC_DATL_DATA0(adc_meas & 0xFFu);   //Set Lower 8 bits of Output
  DAC0->DAT[0].DATH = DAC_DATH_DATA1((adc_meas >> 0x8)&0xFFu);    //Set Higher 8 bits of Output

  NVIC_ClearPendingIRQ(PIT0_IRQn);              //Clears interrupt flag in NVIC Register
  PIT->CHANNEL[0].TFLG  = PIT_TFLG_TIF_MASK;    //Clears interrupt flag in PIT Register

  GPIOA->PCOR |= GPIO_PCOR_PTCO(0x1u << 1); // Red LED = 0
}
```

Button Interrupt

```c
// K++ BUTTON
void PORTB_IRQHandler(void){ //This function might be called when the SW3 is pushed
  if(fs==0x4){ //if filter select is equal to 4 (digital wire)
    fs = 0x0; //set equal to zero (loop back around to lowest starting passband freq)
  }else{fs++;}

  for(k=0;k<K;k++){
    buf[k][0]=0;
    buf[k][1]=0;
  }
  NVIC_ClearPendingIRQ(PORTB_IRQn);    //CMSIS Function to clear pending interrupts on PORTB
  PORTB->ISFR = (0x1u << 17);          //clears PORTB ISFR flag
}
```

Optimizations/Revisions made

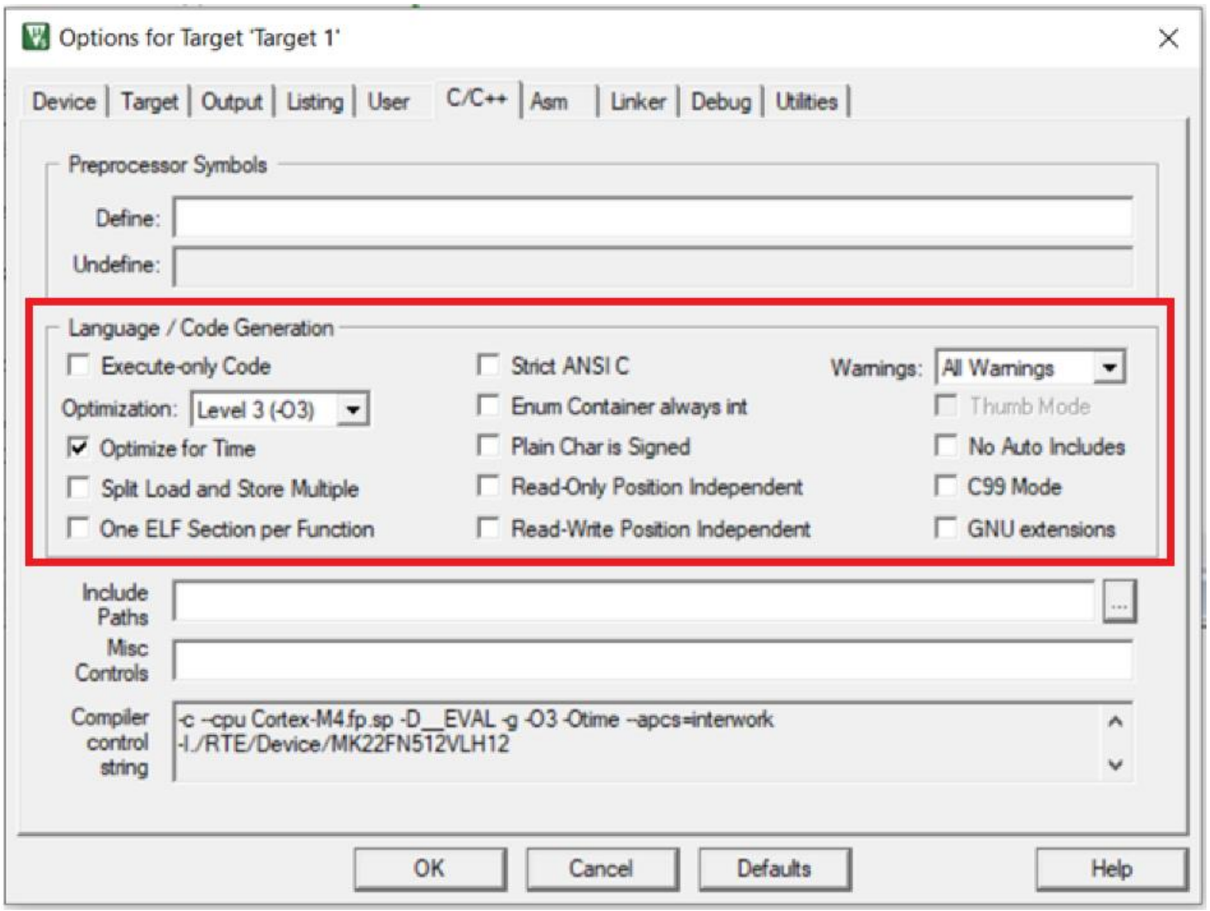
Pulling out first stage

Most of my time in the main interrupt was from the large for-loop that calculates the next output value. I saw that if I took out the first stage of the for-loop, and nested together some of the calculations, I saved unnecessary assignment operations

```c
void PIT0_IRQHandler(void){ //This function is called when the timer interrupt expires
  GPIOA->PSOR |= GPIO_PSOR_PTSO(0x1u << 1); // Turn on Red LED
  ADC0->SC1[0] &= 0xE0;    //Start conversion of channel 1
  adc_meas = ADC0->R[0];  //Read channel 1 after conversion

  if(fs!=0x4){ //if filter select is not set to digital wire

    // First Stage
    v = (G[fs]*(((float)adc_meas)-((float)2048))) - A[fs][0][0]*buf[0][0] - A[fs][0][1]*buf[0][1];
    y = v + B[fs][0]*buf[0][0] + buf[0][1]; //compute output of filter stage y
    buf[0][1] = buf[0][0]; //cycle buffer (n-1) to (n-2)
    buf[0][0] = v; //input 'v' into buffer (n-1)

    // Rest of Stages
    for(k=1;k<K;k++){ //for each stage of filter
      v = y - A[fs][k][0]*buf[k][0] - A[fs][k][1]*buf[k][1]; //compute 'middle-man' variable 'v'
      y = v + B[fs][k]*buf[k][0] + buf[k][1]; //compute output of filter stage y
      buf[k][1] = buf[k][0]; //cycle buffer (n-1) to (n-2)
      buf[k][0] = v; //input 'v' into buffer (n-1)
    }

    adc_meas = (uint16_t)(y+2048); // mask y for output
  } //end of if statement

  DAC0->DAT[0].DATL = DAC_DATL_DATA0(adc_meas & 0xFFu);   //Set Lower 8 bits of Output
  DAC0->DAT[0].DATH = DAC_DATH_DATA1((adc_meas >> 0x8)&0xFFu);    //Set Higher 8 bits of Output

  NVIC_ClearPendingIRQ(PIT0_IRQn);              //Clears interrupt flag in NVIC Register
  PIT->CHANNEL[0].TFLG  = PIT_TFLG_TIF_MASK;    //Clears interrupt flag in PIT Register

  GPIOA->PCOR |= GPIO_PCOR_PTCO(0x1u << 1); // Red LED = 0
}
```

Debug Optimizations

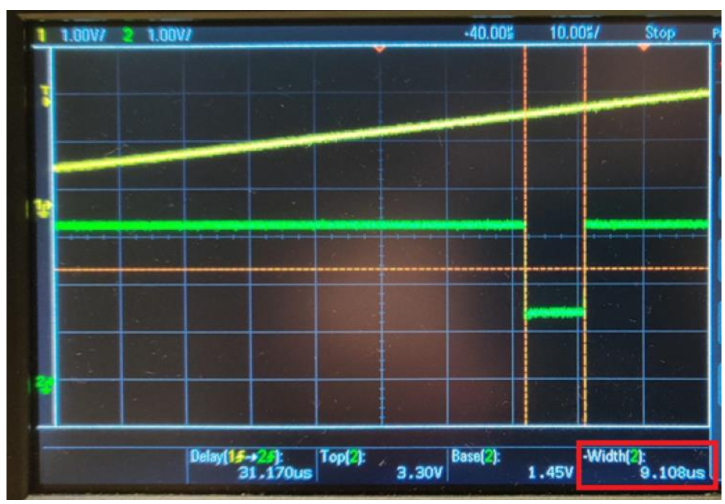
The second optimization I made was to increase the Keil uVision IDE’s Code Generation Optimization level



Testing Initial vs. Optimized C-code

By Probing the Red LED light that is programmed to turn on at the beginning of the ISR, and turn off at the end:

Before pulling out the first stage:

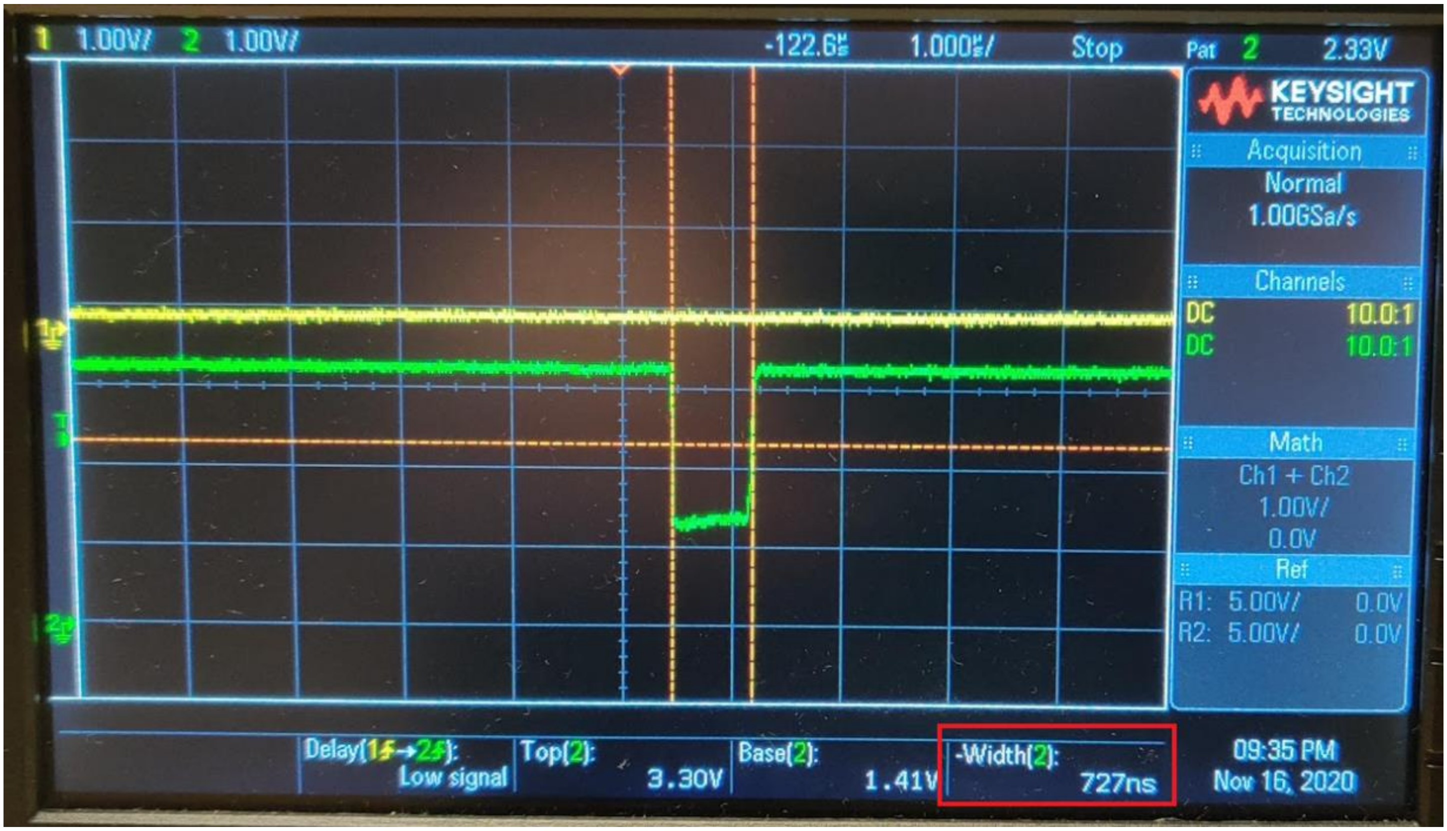


After pulling out the first stage:

This was with K = 256. This gave about 6us more time before I would miss timing

Finding Highest Order Possible

At K = 256 I had 9.108us left. Next was K = 280.



There may be more debugger settings that could increase the amount of time in the interrupt before missing, but without that this is the highest order I can go.