EECS 152B / CSE 135B

DSP Design & Laboratory

Assignment 3

Implementation FIR and IIR Filter using TI c6713 DSK

Winter 2017

Diego Bravo

Hector Solano

Paul Dao

Lab Session:

Lab Dates: 02/8/2017, 02/15/2017, and 02/22/2017

Introduction

In this experiment, Finite Impulse Response (FIR) and Infinite Impulse Response (IIR) filters are designed using Matlab and C to realize lowpass, highpass, and bandpass filters on the TI c6173 DSK DSP board . FIR filters are longer in length compared to IIR filters; IIR filters utilize recursion.

An FIR filter is finite in length with length N and is expressed as a summation of delays described by Equation 1.

$$y[n] = \sum_{i=0}^{N} b[i]x[n-i] \quad \text{Equation 1}$$

where the array, b[i], are the coefficients of the delayed inputs obtained using MATLAB. Problems 1-3 implement lowpass, highpass, and bandpass filters, respectively, using FIR filters.

An IIR filter is described as a recursive delay by Equation 2.

$$y[n] = \frac{1}{a_0}\left(\sum_{i=0}^{N} b[i]x[n-i] - \sum_{i=1}^{M} a[i]y[n-i]\right) \quad \text{Equation 2}$$

where b[i] is an array containing the coefficients of the numerator and a[i] of the denominator of the transfer function H(z). Problems 4-6 realize lowpass, highpass, and bandpass filters, with IIR instead of FIR to examine the decrease in length of the filter.

The DSP board's sampling rate is set to 16kHz in order for the highpass and bandpass filters to satisfy the Nyquist Criterion. Thus, the nyquist frequency is 8kHz. In the MATLAB scripts, the normalized cutoff frequencies are wnorm=0.22 (LPF) and wnorm2=0.57 (HPF).

Problem 1

OBJECTIVE & PROBLEM

Implement an FIR filter for low pass filtering with cutoff frequency 1760 Hz using MATLAB to obtain the coefficients.

BRIEF EXPLANATION OF THE CODE

Before starting the polling process, we store the lowpass FIR coefficients generated in Matlab in array "lpf_fir". We produce this coefficient vector by calling the fir1 function and giving it order M=50 and cutoff frequency wnorm = 1760 / 8000 Hz. The following C code starts the polling process by calling the comm_poll() function from the Vectors_poll.asm assembly code file. After initializing the polling process, we store 51 input samples to use for convolution. Next, the program enters the infinite while-loop where we implement a lowpass FIR filter by convolving the input samples with the lowpass FIR filter coefficients. Once the output has been calculated, we shift all the inputs in the "buffer" array to the left by 1 index and store the latest input sample in the last cell of the array. Finally, the the result of the convolution is output.

CODE

C source code implements Equation 1 for a FIR lowpass filter

RESULTS & CONCLUSION

The code was tested by connecting the line-out port of the DSP board to an oscilloscope via positive and ground wires. To test our low pass filter, we used a tone generator and connected the generator's output to the DSP board's audio input. Upon execution of the program, the oscilloscope displayed a clear sinusoid for tones with a frequency $\leq$ 1760 Hz. When the tone reached about 1760+ Hz, the sinusoid became distorted, showing that only signals with frequency $\leq$ 1760 Hz were allowed. The test tone is set at approximately 900Hz and after passing the tone through the lowpass filter, the sine wave in Figure 1 is generated.

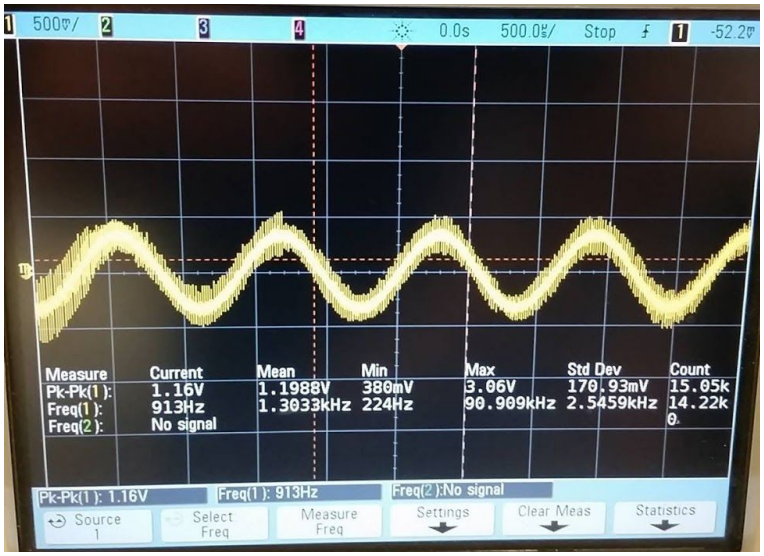

Figure 1: Resulting sine wave post filtering

Amplitude (Peak-to-peak V)	Frequency (Hz)
1.16	913

Problem 2

OBJECTIVE & PROBLEM

Implement an FIR filter for high pass filtering with cutoff frequency 4560 Hz.

BRIEF EXPLANATION OF THE CODE

Before starting the polling process, we store the highpass FIR coefficients generated in Matlab in array "hpf_fir". We produce this coefficient vector by calling the fir1 function and giving it order M=50 and cutoff frequency wnorm2= 4560 / 8000 Hz. The following C code starts the polling process by calling the comm_poll() function from the Vectors_poll.asm assembly code file. After initializing the polling process, we store 51 input samples to use for convolution. Next, the program enters the infinite while-loop where we implement a highpass FIR filter by convolving the input samples with the highpass FIR filter coefficients. Once the output has been calculated, we shift all the inputs in the "buffer" array to the left by 1 index and store the latest input sample in the last cell of the array. Finally, we output the result of the convolution.

CODE

```
#include "dsk6713_aic23.h"
#include "dsk6713_led.h"
#define DSK6713_AIC23_INPUT_MIC 0x0015
#define DSK6713_AIC23_INPUT_LINEIN 0x0011
Uint32 fs = DSK6713_AIC23_FREQ_16KHZ; // 1
Uint16 inputsource = DSK6713_AIC23_INPUT_LINEIN; // 0x011

unsigned int i = 0;
const int SIZE = 51;
// Problem2
float hpf_fir [51] =
{-0.000719150312854400,0.000935044661893439,0.000442110331584869,-0.00160958733
367831,0.000194981511618248,0.00253681043743606,-0.00174280100983270,-0.0031764
3317418914,0.00453858851009697,0.00251649791536997,-0.00832597506970928,0.00064
6873892044080,0.0119995565631141,-0.00725858432270368,-0.0135714877449711,0.017
5396463194574,0.0103202148713842,-0.0306917093286511,0.00118828929073415,0.0449
096811805753,-0.0263224722820524,-0.0577381744168883,0.0810056320484234,0.06668
33063922820,-0.309043289229996,0.429341650509185,-0.309043289229996,0.066683306
3922820,0.0810056320484234,-0.0577381744168883,-0.0263224722820524,0.0449096811
805753,0.00118828929073415,-0.0306917093286511,0.0103202148713842,0.01753964631
94574,-0.0135714877449711,-0.00725858432270368,0.0119995565631141,0.00064687389
2044080,-0.00832597506970928,0.00251649791536997,0.00453858851009697,-0.0031764
3317418914,-0.00174280100983270,0.00253681043743606,0.000194981511618248,-0.001
60958733367831,0.000442110331584869,0.000935044661893439,-0.000719150312854400
};

void main() {
   comm_poll();
   float y;
   short buffer[SIZE];

   for(i = 0; i < SIZE; i++) {
      buffer[i] = input_left_sample();
   }

   while(1) {
     y = 0;
     for (i = 0; i < SIZE; i++)
         y += hpf_fir[i] * buffer[SIZE - i - 1];

     for (i = 0; i < SIZE - 1; i++)
```

```
        buffer[i] = buffer[i + 1];

    buffer[SIZE - 1] = input_left_sample();

    output_left_sample((short)y);
  }
}
```

C source code implements Equation 1 for a FIR highpass filter

RESULTS & CONCLUSION

The code was tested by connecting the line-out port of the DSP board to an oscilloscope via positive and ground wires. To test our high pass filter, we used a tone generator and connected the generator's output to the DSP board's audio input. Upon execution of the program, the oscilloscope displayed a clear sinusoid for tones with a frequency $\geq$ 4560 Hz. When the tone reached about 4560- Hz, the sinusoid became distorted, showing that only signals with frequency $\geq$ 4560 Hz were allowed. The test tone is set at approximately 6040 Hz and the output from passing the tone through the highpass filter is seen below in Figure 2.

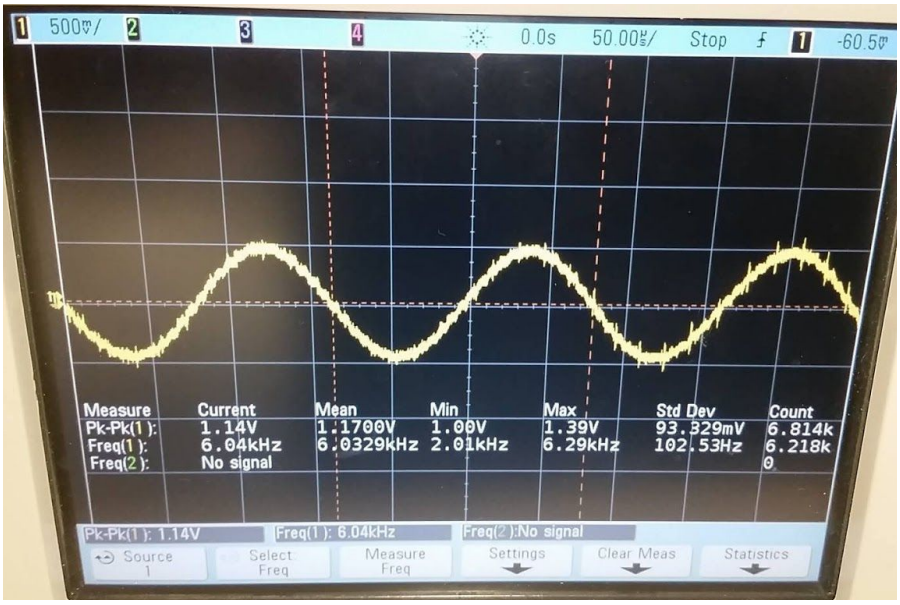

Figure 2: Resulting sine wave post highpass filtering

Amplitude (V peak-to-peak)	Frequency (Hz)
1.14	6040

Problem 3

OBJECTIVE & PROBLEM

Implement an FIR filter for band pass filtering with cutoff frequencies 1760 Hz and 4560 Hz.

BRIEF EXPLANATION OF THE CODE

Before starting the polling process, we store the bandpass FIR coefficients generated in Matlab in array "bpf_fir". We produce this coefficient vector by calling the fir1 function and giving it order M=50 and cutoff frequencies 1760 / 8000 Hz and 4560 / 8000 Hz. The following C code starts the polling process by calling the comm_poll() function from the Vectors_poll.asm assembly code file. After initializing the polling process, we store 51 input samples to use for convolution. Next, the program enters the infinite while-loop where we implement a bandpass FIR filter by convolving the input samples with the bandpass FIR filter coefficients. Once the output has been calculated, we shift all the inputs in the "buffer" array to the left by 1 index and store the latest input sample in the last cell of the array. Finally, we output the result of the convolution.

CODE

```
#include "dsk6713_aic23.h"
#include "dsk6713_led.h"
#define DSK6713_AIC23_INPUT_MIC 0x0015
#define DSK6713_AIC23_INPUT_LINEIN 0x0011
Uint32 fs = DSK6713_AIC23_FREQ_16KHZ; // 1
Uint16 inputsource = DSK6713_AIC23_INPUT_LINEIN; // 0x011

unsigned int i = 0;
const int SIZE = 51;

float bpf_fir[SIZE ] =
{0.00173804038433836,-0.000081832998238869,-0.00019775739430279 6,0.00082888573
9262472,-0.00212364071389786,-0.00507905032265195,-0.000091803024303 5165,0.0037
2654819024800,-0.000538963539585851,0.00431061666171047,0.0151620016893806,0.00
191720239813089,-0.0173385080693537,-0.00638126960350276,-0.00395696633047738,-
0.0303153520677700,-0.00869784943549450,0.0521362171617645,0.0363829432014739,-
0.00576742902759117,0.0442866897719188,0.0301851411190233,-0.171026242646072,-0.
220759943321729,0.107305119874599,0.349838107144798,0.107305119874599,-0.220759
943321729,-0.171026242646072,0.0301851411190233,0.0442866897719188,-0.005767429
02759117,0.0363829432014739,0.0521362171617645,-0.00869784943549450,-0.03031535
20677700,-0.00395696633047738,-0.00638126960350276,-0.0173385080693537,0.001917
20239813089,0.0151620016893806,0.00431061666171047,-0.000538963539585851,0.0037
```

```
2654819024800,-0.0000918030243035165,-0.00507905032265195,-0.00212364071389786,
0.000828885739262472,-0.000197757394302796,-0.0000818329982388869,0.00173804038
433836};

void main() {
  comm_poll();
  float y;
  short buffer[SIZE];

  for(i = 0; i < SIZE; i++){
    buffer[i] = input_left_sample();
  }

  while(1){
    y = 0;
    for (i = 0; i < SIZE; i++)
        y += bpf_fir[i] * buffer[SIZE - i - 1];

    for (i = 0; i < SIZE - 1; i++)
       buffer[i] = buffer[i + 1];

    buffer[SIZE - 1] = input_left_sample();
    output_left_sample((short)y);
  }
}
```

C source code implements Equation 1 for a FIR bandpass filter

RESULTS & CONCLUSION

The code was tested by connecting the line-out port of the DSP board to an oscilloscope via positive and ground wires. To test our band pass filter, we used a tone generator and connected the generator's output to the DSP board's audio input. Upon execution of the program, the oscilloscope displayed a clear sinusoid for tones with a frequency $\geq$ 1760 Hz and $\leq$ 4560 Hz. When the tone reached about 1760+ Hz and 4560- Hz, the sinusoid became distorted, showing that only signals with frequency $\geq$ 1760 Hz and 4560- Hz were allowed. The test tone is set at approximately 3000 Hz and the output from passing the tone through the bandpass filter is seen below in Figure 3.

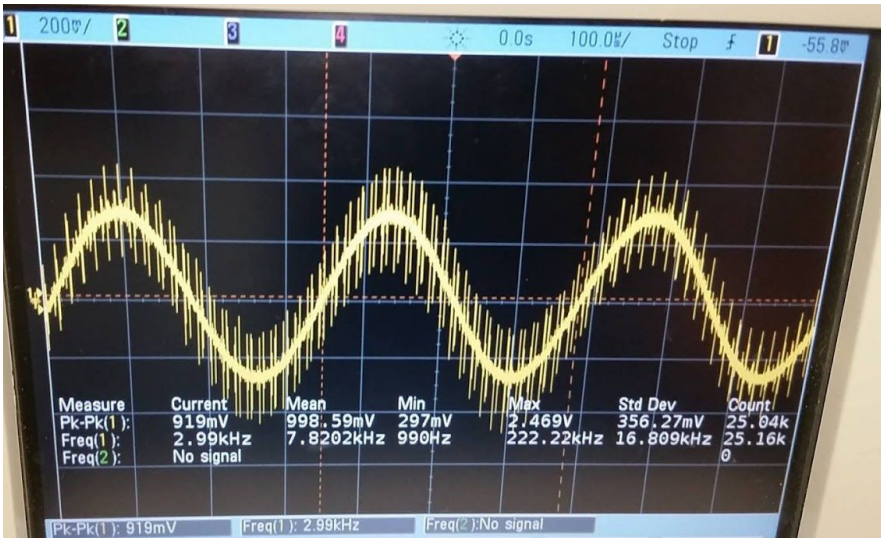

Figure 3: Resulting sine wave post bandpass filtering

Amplitude (V peak-to-peak)	Frequency (Hz)
0.919	2990

Problem 4

OBJECTIVE & PROBLEM

Implement an IIR filter for low pass filtering with cutoff frequency 1760 Hz to observe a decrease in the overall filter's length relative to the FIR filter.

BRIEF EXPLANATION OF THE CODE

Before starting the polling process, we store the numerator and denominator coefficients generated in Matlab in 2 different arrays, "lpf_iir_b" and "lpf_iir_a" respectively. We produce these 2 coefficient vectors by calling the butter function and giving it order M=10 and cutoff frequency 1760 / 8000 Hz. The following C code starts the polling process by calling the comm_poll() function from the Vectors_poll.asm assembly code file. After initializing the polling process, we store 11 input samples to an array that stores input samples and 11 zeros to an array that stores output samples to use for convolution. Next, the program enters the infinite while-loop where we first shift all the inputs in array "x" to the right by 1 and shift all the output samples in array "y" to the right by 1. After that, we store a new input sample in the 1st index of array "x" and implement a lowpass IIR filter. We implement the lowpass IIR filter by first convolving the input samples with the lowpass IIR filter numerator coefficients in array "lpf_iir_b". Next, we convolve the values in the output buffer "y" with all the denominator

coefficients in array “lpf_iir_a” except the 1st denominator coefficient. We then take the result of the 2nd convolution and subtract it from the result of the 1st convolution. The resulting difference is finally multiplied by the reciprocal of the 1st denominator coefficient, or (1 / lpf_iir_a[0]), stored in the first index of array “y”, and outputted.

CODE

```
#include "dsk6713_aic23.h"
#include "dsk6713_led.h"
#define DSK6713_AIC23_INPUT_MIC 0x0015
#define DSK6713_AIC23_INPUT_LINEIN 0x0011

Uint32 fs = DSK6713_AIC23_FREQ_16KHZ; // 1
Uint16 inputsource = DSK6713_AIC23_INPUT_LINEIN; // 0x011
unsigned int i = 0;
const int FILTER_LEN = 11;

float lpf_iir_b[FILTER_LEN] =
{0.0152550224105746,-0.0493796754291947,0.0961854954970261,-0.118723890386639,0.
122312552889201,-0.114107264166309,0.122312552889201,-0.118723890386639,0.09618
54954970260,-0.0493796754291946,0.0152550224105746};
float lpf_iir_a[11] =
{1,-5.29478690795677,13.3830271120091,-20.8604390248056,22.0409073555607,-16.3997
573159628,8.67332019492232,-3.20904788995105,0.794102467286104,-0.1183998580049
64,0.00826561269860652};

void main() {
   comm_poll();
   char i;
   short x[FILTER_LEN];
   double y[FILTER_LEN];
   double temp;

   // Initialize buffer to 0
   for (i = 0; i < FILTER_LEN; i++) {
      x[i] = 0;
   }
   for (i = 0; i < FILTER_LEN; i++) {
      y[i] = 0.0;
   }

   while(1) {
```

```
      for (i = FILTER_LEN-1; i > 0; i--) {
          x[i] = x[i-1];
      }

      for (i=FILTER_LEN-1; i > 0; i--) {
          y[i] = y[i-1];
      }
     temp = 0;
     x[0] = input_left_sample();
     for (i = 0; i < FILTER_LEN; i++) {
         temp += lpf_iir_b[i] * x[i];
     }
      y[0] = temp; //* lpf_iir_a[0];
      for (i = 1; i < FILTER_LEN; i++) {
           y[0] -= lpf_iir_a[i] * y[i];
      }
      y[0] = y[0] * (1 / lpf_iir_a[0]);
      output_left_sample((short) y[0]);
   }
}
```

C Code Implements Equation 2 to create IIR lowpass filter

RESULTS & CONCLUSION

The code was tested by connecting the line-out port of the DSP board to an oscilloscope via positive and ground wires. To test our low pass filter, we used a tone generator and connected the generator's output to the DSP board's audio input. Upon execution of the program, the oscilloscope displayed a clear sinusoid for tones with a frequency $\leq$ 1760 Hz. When the tone reached about 1760+ Hz, the sinusoid became distorted, showing that only signals with frequency $\leq$ 1760 Hz were allowed. With less coefficients and thus less delays, the lowpass IIR filter showcased the same functionality as the lowpass FIR filter, which has more coefficients. It was noted that going below 10 samples resulted in a LPF where signals with frequencies greater than 1760 Hz were still being passed, thus order of M=10 was utilized. The test tone is set at approximately 1500 Hz and the output from passing the tone through the IIR lowpass filter is seen below in Figure 4.

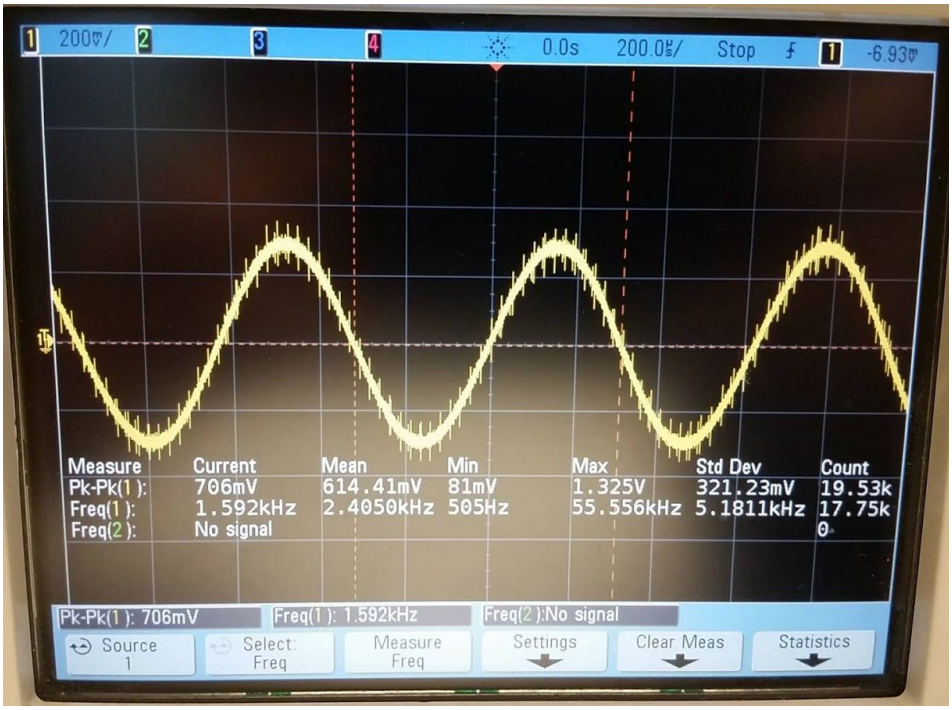

Figure 4: Resulting sine wave post lowpass filtering

Amplitude (V peak-to-peak)	Frequency (Hz)
0.706	1592

Problem 5

OBJECTIVE & PROBLEM

Implement an IIR filter for high pass filtering with cutoff frequency 4560 Hz.

BRIEF EXPLANATION OF THE CODE

Before starting the polling process, we store the numerator and denominator coefficients generated in Matlab in 2 different arrays, “hpf_iir_b” and “hpf_iir_a” respectively. These two coefficient vectors are created using MATLAB by calling the butter() function and giving it order 10 and cutoff frequency 4560 / 8000 Hz. The following C code starts the polling process by calling the comm_poll() function from the Vectors_poll.asm assembly code file. After initializing the polling process, we store 11 input samples to an array that stores input samples and 11 zeros to an array that stores output samples to use for convolution. Next, the program enters the infinite while-loop where we first shift all the inputs in array “x” to the right by 1 and shift all the output samples in array “y” to the right by 1. After that, we store a new input sample in the 1st index of array “x” and implement a highpass IIR filter. We implement the

highpass IIR filter by first convolving the input samples with the highpass IIR filter numerator coefficients in array “hpf_iir_b”. Next, we convolve the values in the output buffer “y” with all the denominator coefficients in array “hpf_iir_a,” except the 1st denominator coefficient. We then take the result of the 2nd convolution and subtract it from the result of the 1st convolution. The resulting difference is finally multiplied by the reciprocal of the 1st denominator coefficient, or (1 / hpf_iir_a[0]), stored in the first index of array “y”, and outputted.

CODE

```
#include "dsk6713_aic23.h"
#include "dsk6713_led.h"
#define DSK6713_AIC23_INPUT_MIC 0x0015
#define DSK6713_AIC23_INPUT_LINEIN 0x0011

Uint32 fs = DSK6713_AIC23_FREQ_16KHZ; // 1
Uint16 inputsource = DSK6713_AIC23_INPUT_LINEIN; // 0x011
const int FILTER_LEN = 11;

double hpf_iir_b[FILTER_LEN] =
{0.000884577085863563,-0.00884577085863563,0.0398059688638604,-0.10614925030362
8,0.185761188031348,-0.222913425637618,0.185761188031348,-0.106149250303628,0.03
98059688638604,-0.00884577085863563,0.000884577085863563};
double hpf_iir_a[FILTER_LEN] =
{1,1.39450010493605,2.16234423909640,1.68974273396699,1.21840264081444,0.5629925
42767413,0.219023203137815,0.0564306293019384,0.0108463384415639,0.00121020224
486195,.0000667276513233946};

void main() {
   comm_poll();
   char i;
   short x[FILTER_LEN];
   double y[FILTER_LEN];
   double temp;

   // Initialize buffer to 0
   for (i = 0; i < FILTER_LEN; i++) {
      x[i] = 0;
   }

   for (i = 0; i < FILTER_LEN; i++) {
      y[i] = 0.0;
```

```
  }

  while(1) {
    for (i = FILTER_LEN-1; i > 0; i--) {
        x[i] = x[i-1];
    }

    for (i=FILTER_LEN-1; i > 0; i--) {
        y[i] = y[i-1];
    }
    temp = 0;
    x[0] = input_left_sample();
    for (i = 0; i < FILTER_LEN; i++) {
       temp += hpf_iir_b[i] * x[i];
    }
     y[0] = temp;;
     for (i = 1; i < FILTER_LEN; i++) {
          y[0] -= hpf_iir_a[i] * y[i];
     }
     y[0] = y[0] * (1 / hpf_iir_a[0]);
     output_left_sample((short) y[0]);
  }
}
```

C Code Implements Equation 2 to create IIR highpass filter

RESULTS & CONCLUSION

The code was tested by connecting the line-out port of the DSP board to an oscilloscope via positive and ground wires. To test our high pass filter, we used a tone generator and connected the generator's output to the DSP board's audio input. Upon execution of the program, the oscilloscope displayed a clear sinusoid for tones with a frequency $\geq$ 4560 Hz. When the tone reached about 4560- Hz, the sinusoid became distorted, showing that only signals with frequency $\geq$ 4560 Hz were allowed. With less coefficients and thus less delays, the highpass IIR filter showcased the same functionality as the highpass FIR filter, but with a shorter length. The test tone is set at approximately 1500 Hz and the output from passing the tone through the IIR highpass filter is seen below in Figure 5.

Figure 5: Resulting sine wave post highpass filtering

Amplitude (V peak-to-peak)	Frequency (Hz)
0.522	5950

Problem 6

OBJECTIVE & PROBLEM

Implement an IIR filter for band pass filtering with cutoff frequencies 1760 Hz and 4560 Hz.

BRIEF EXPLANATION OF THE CODE

Before starting the polling process, we store the numerator and denominator coefficients generated in Matlab in 2 different arrays, “bpf_iir_b” and “bpf_iir_a” respectively. We produce these 2 coefficient vectors by calling the butter() function and giving it order M=10 and passband frequencies 1760 / 8000 Hz and 4560 / 8000 Hz. The following code starts the polling process by calling the comm_poll() function from the Vectors_poll.asm assembly code file. After initializing the polling process, we store 11 input samples to an array that stores input samples and 11 zeros to an array that stores output samples to use for convolution. Next, the program enters the infinite while-loop where we first shift all the inputs in array “x” to the right by 1 and shift all the output samples in array “y” to the right by 1. After that, we store a

new input sample in the 1st index of array “x” and implement a bandpass IIR filter. We implement the bandpass IIR filter by first convolving the input samples with the bandpass IIR filter numerator coefficients in array “bpf_iir_b”. Next, we convolve the values in the output buffer “y” with all the denominator coefficients in array “bpf_iir_a,” except the 1st denominator coefficient. We then take the result of the 2nd convolution and subtract it from the result of the 1st convolution. The resulting difference is finally multiplied by the reciprocal of the 1st denominator coefficient, or (1 / bpf_iir_a[0]), stored in the first index of array “y”, and outputted.

CODE

```
#include "dsk6713_aic23.h"
#include "dsk6713_led.h"
#define DSK6713_AIC23_INPUT_MIC 0x0015
#define DSK6713_AIC23_INPUT_LINEIN 0x0011

Uint32 fs = DSK6713_AIC23_FREQ_16KHZ; // 1
Uint16 inputsource = DSK6713_AIC23_INPUT_LINEIN; // 0x011

unsigned int i = 0;
const int FILTER_LEN = 21;

float bpf_iir_b[FILTER_LEN ] =
{0.00017238374019171 6,0,-0.00172383740191716,0,0.00775726830862722,0,-0.02068604
88230059,0,0.0362005854402604,0,-0.043440702528312 5,0,0.0362005854402604,0,-0.020
6860488230059,0,0.00775726830862722,0,-0.00172383740191716,0,0.0001723837401917
16};
float bpf_iir_a[FILTER_LEN ] =
{1,-4.93471074974774,14.1071880704920,-29.1603297837713,48.5049851195987,-67.6571
897465564,81.3524732995103,-85.4971760627105,79.4338163377995,-65.5479269717378,
48.2158693435008,-31.5906305154249,18.4154149449824,-9.49463155136709,4.30164217
159637,-1.68906061357564,0.566167259814615,-0.156857675539605,0.034604323689757
8,-0.00546758351163364,0.000532881341674807};

void main() {
   comm_poll();
   char i;
   short x[FILTER_LEN];
   double y[FILTER_LEN];
   double temp;

   // Initialize buffer to 0
```

```
    for (i = 0; i < FILTER_LEN; i++) {
      x[i] = 0;
    }

    for (i = 0; i < FILTER_LEN; i++) {
      y[i] = 0.0;
    }

    while(1) {
      for (i = FILTER_LEN-1; i > 0; i--) {
          x[i] = x[i-1];
      }

      for (i=FILTER_LEN-1; i > 0; i--) {
          y[i] = y[i-1];
      }
     temp = 0;
     x[0] = input_left_sample();
     for (i = 0; i < FILTER_LEN; i++) {
        temp += bpf_iir_b[i] * x[i];
     }
      y[0] = temp;
      for (i = 1; i < FILTER_LEN; i++) {
           y[0] -= bpf_iir_a[i] * y[i];
      }
      y[0] = y[0] * (1 / bpf_iir_a[0]);
      output_left_sample((short) y[0]);
   }
}
```

C Code Implements Equation 2 to create IIR bandpass filter

RESULTS & CONCLUSION

The code was tested by connecting the line-out port of the DSP board to an oscilloscope via positive and ground wires. To test our band pass filter, we used a tone generator and connected the generator's output to the DSP board's audio input. Upon execution of the program, the oscilloscope displayed a clear sinusoid for tones with a frequency >= 1760 Hz and <= 4560. When the tone reached about 1760+ Hz and 4560- Hz, the sinusoid became distorted, showing that only signals with frequency >= 1760 Hz and 4560- Hz were allowed. With less coefficients and thus less delays, the bandpass IIR filter showcased the same functionality as the bandpass FIR filter, which has more coefficients. The test tone is set at approximately 1500 Hz and the output from passing the tone through the IIR bandpass filter is seen below in Figure 6.

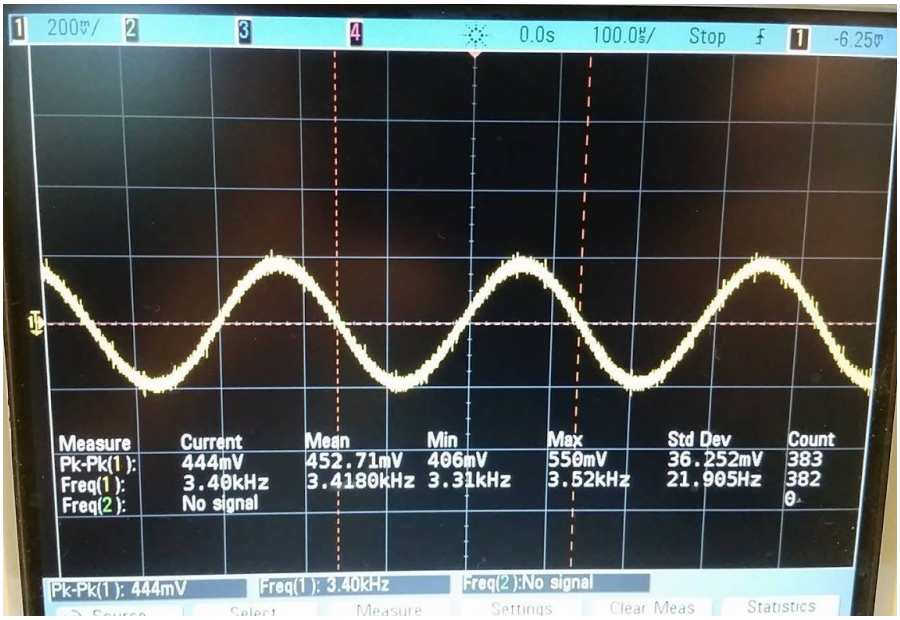

Figure 6: Resulting sine wave post bandpass filtering

Amplitude (V peak-to-peak)	Frequency (Hz)
0.444	3400

QUESTIONS:

A: The gain of every filter in the linear scale is equal to 1Vpp since the gain in decibels is 0 dB, thus no amplification should be present. In comparison to the gain of the output for the FIR filters in Figures 1-3, the amplitude of the output waveform is near 1Vpp so there is negligible error. However, the output from the IIR filters seen in Figure 4-6 are near 0.5Vpp so there was some distortion possibly caused by the length of the IIR filters.

1. FIR Lowpass Filter

The Matlab script attached below generates coefficients for a 50th order FIR lowpass filter with a normalized cutoff frequency, wnorm=0.22pi radians/sample. The fir1() function returns a vector of length M+1 with the coefficients of the desired lowpass filter, which is to be used as an array in C.

MATLAB SCRIPT

```
M   = 50; % filter order
fs = 16e3; % sampling freq = 16kHz
nyquist = fs / 2;
Flow  = 1760; % cutoff freq for LPF
wnorm = Flow/nyquist; % normalized frequency = 0.22pi
b = fir1(M,wnorm,'low');
```

Matlab script generates coefficients for lowpass filter frequency response

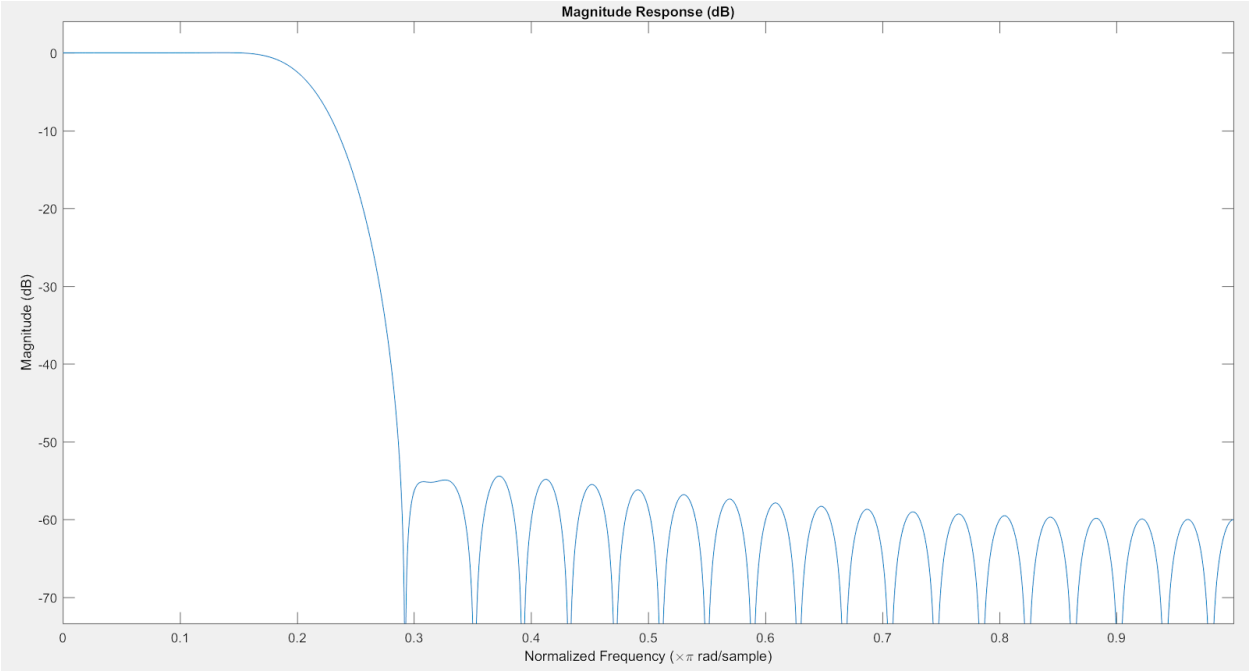

Figure 7: Magnitude Response of FIR lowpass filter

The stopband of the FIR is well below -40dB with an attenuation of approximately 55 dB. The transition frequencies are $0.22 \leq \omega \leq 0.29\,\pi$ radians/sample. The gain of the LPF is 0 dB in log scale and is of magnitude 1V in the linear scale.

FIR Highpass Filter

The Matlab code attached below obtains coefficients of the FIR. The fir1() function returns the coefficients of the desired highpass filter with length M+1. The normalized frequency, wnorm2, is 0.57pi radians/sample.

MATLAB SCRIPT

```
M   = 50;
fs = 16e3; % sampling freq = 16kHz
nyquist = fs / 2;
Fhi=4560; % Hz - cutoff freq for HPF
wnorm2=Fhi/nyquist; % normalized frequency
b2 = fir1(M,wnorm2,'high');
```

Matlab script generates coefficients for highpass filter frequency response

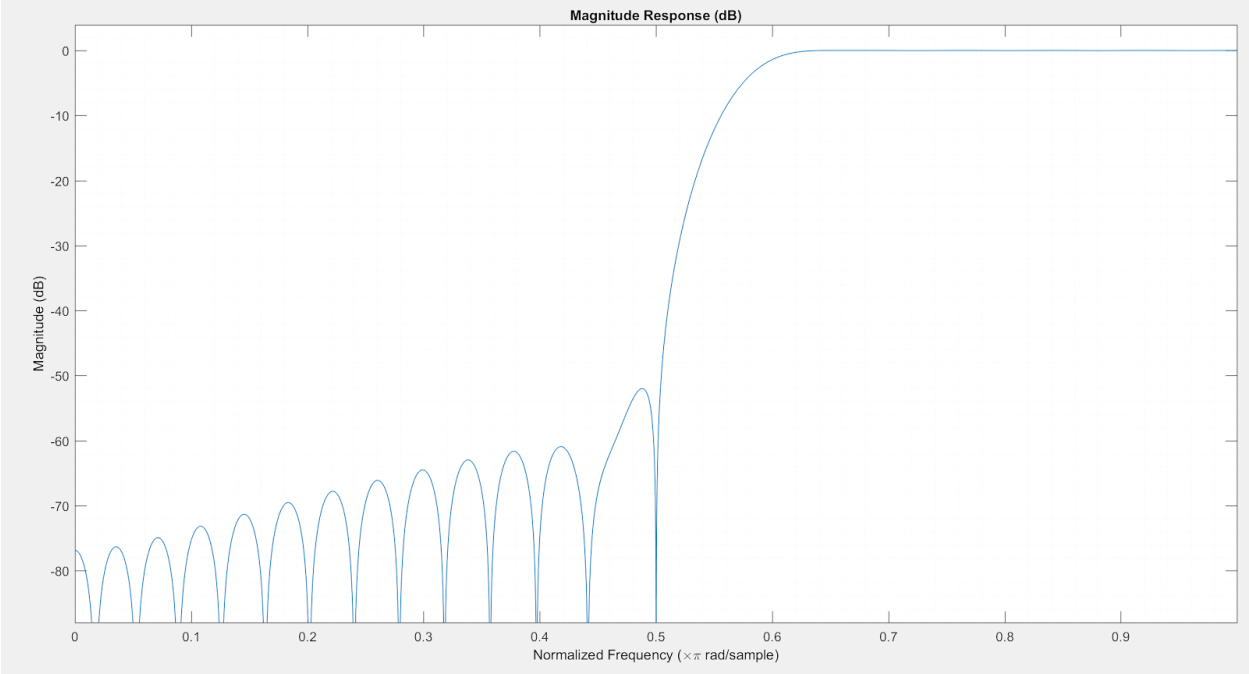

Figure 8: Magnitude Response of FIR highpass filter

The stopband of the FIR is well below -40dB with an attenuation of approximately 50 dB. The transition frequencies are $0.5 \leq \omega \leq 0.6\,\pi$ radians/sample.

FIR Bandpass Filter

The Matlab code attached below demonstrates how the coefficients of the FIR bandpass filter are acquired. The bandpass is designed with M=50 samples. The fir1() function returns the coefficients of the desired bandpass filter with length M+1. The passband frequencies of the BPF are wnorm=0.22pi radians/sample and wnorm2=0.57pi radians/sample.

MATLAB SCRIPT

```
M = 50;
b3 = fir1(M,[wnorm wnorm2]);
fvtool(b3);
```

Matlab script generates coefficients for FIR bandpass filter

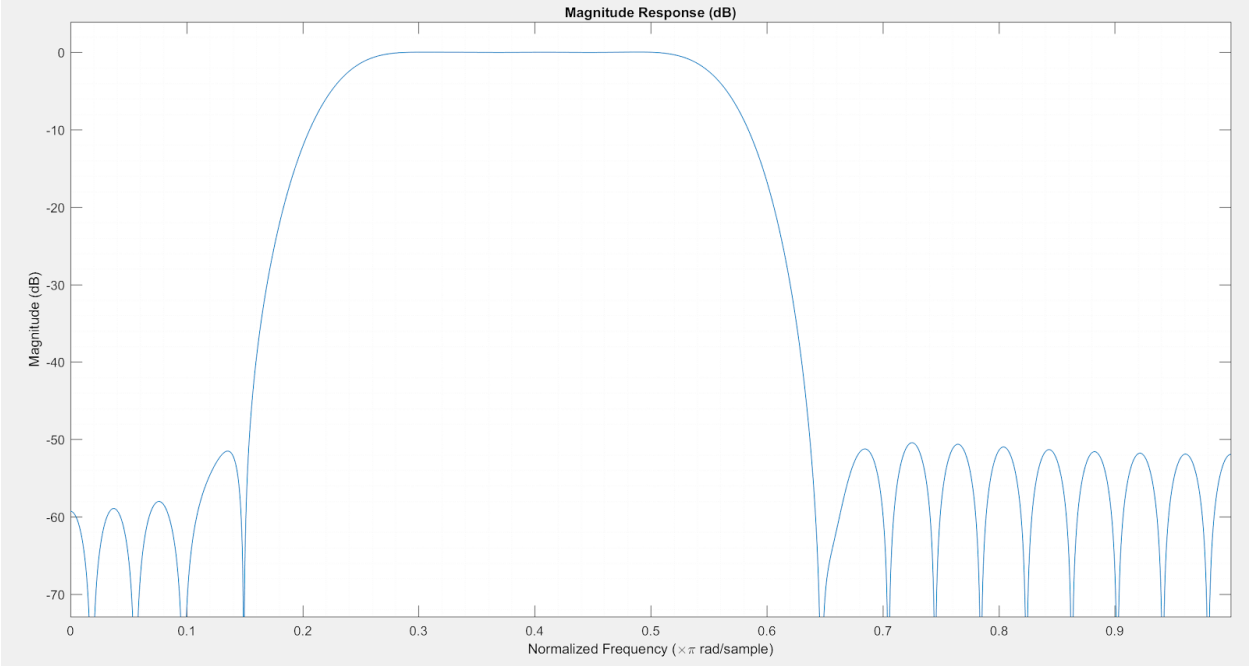

Figure 3: Magnitude Response of FIR bandpass filter

The stopbands of the FIR are well below -40dB with an attenuation of approximately 50 dB. The transition frequencies are $0.15 \leq \omega_1 \leq 0.2\,\pi$ radians/sample and $0.6 \leq \omega_2 \leq 0.65\,\pi$ radians/sample.

IIR Lowpass Filter

The following MATLAB script utilizes the butter() function to create an analog lowpass Butterworth filter that has order M=10 and the same cutoff frequency from Problem 1.

MATLAB SCRIPT

```
M=10;
[Blp,Alp]=butter(M,wnorm,'low'); % cutoff freq = wnorm = 0.22pi
fvtool(Blp,Alp); % Magnitude Response
```

Matlab script generates numerator, Blp_n, and denominator, Alp_n, coefficients for discrete LPF

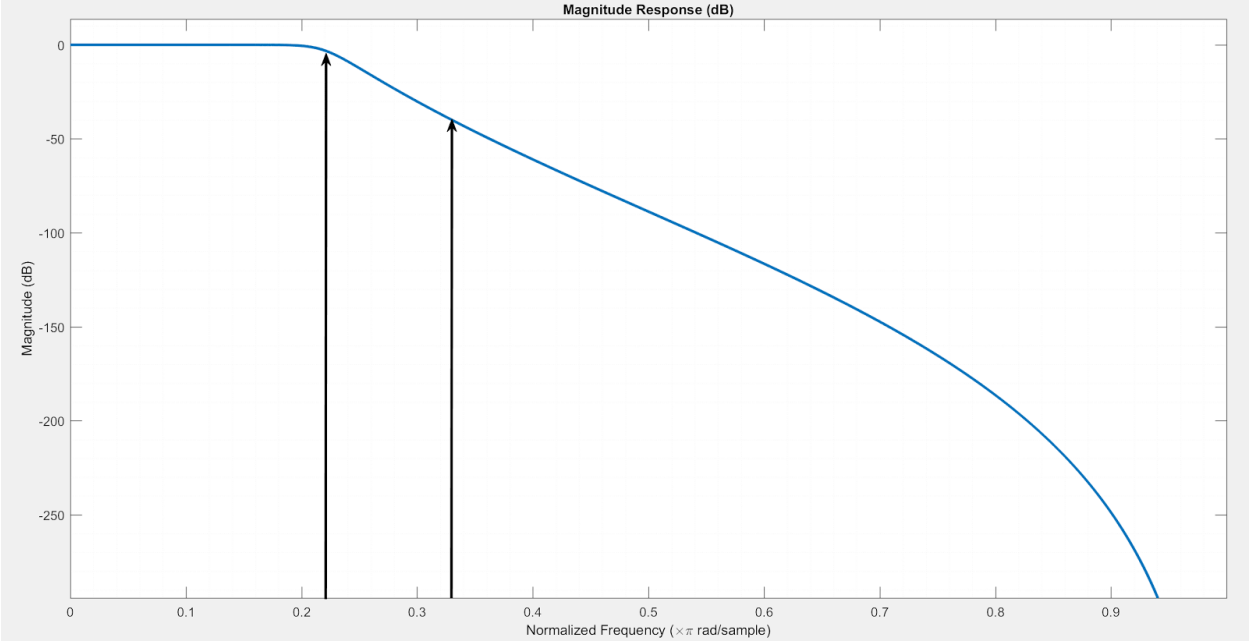

Figure 4: Magnitude Response of IIR Lowpass Filter

The attenuation of the stopband is well below 40 dB at the cutoff frequency wnorm=0.22. The transition band ranges from $0.22 \leq \omega \leq 0.32\,\pi$ radians/sample.

IIR Highpass Filter

The MATLAB script below generates the denominator and numerator coefficients of the transfer function for a highpass filter of order M=10 by using the butter() function.

MATLAB SCRIPT

```
M=10;
[Bhp,Ahp]=butter(M,wnorm2,'high'); % wnorm2=0.57pi
fvtool(Bhp,Ahp);
```

Matlab script generates IIR highpass filter coefficients

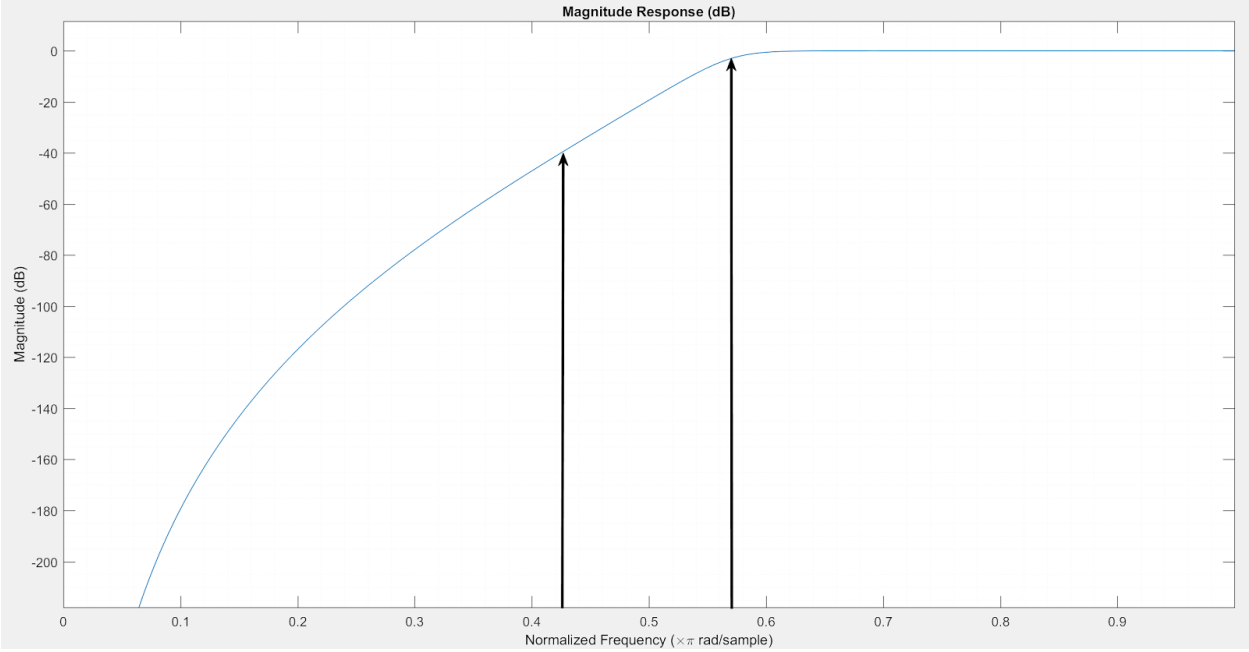

Figure 5: Magnitude Response of IIR Highpass Filter

The attenuation of the stopband is well below 40 dB at the cutoff frequency wnorm=0.22. The transition band ranges from $0.42 \leq \omega \leq 0.57\,\pi$ radians/sample.

IIR Bandpass Filter

The Matlab script below generates the coefficients for a bandpass filter using the butter() function. The passband frequencies of the bandpass filter are the same cutoff frequencies of the LPF and HPF.

```
M=10;
[Bbp,Abp]=butter(M,[wnorm wnorm2]); % wnorm=0.22pi , wnorm2=0.57pi
fvtool(Bbp,Abp);
```

Matlab script generates coefficients for an IIR bandpass filter

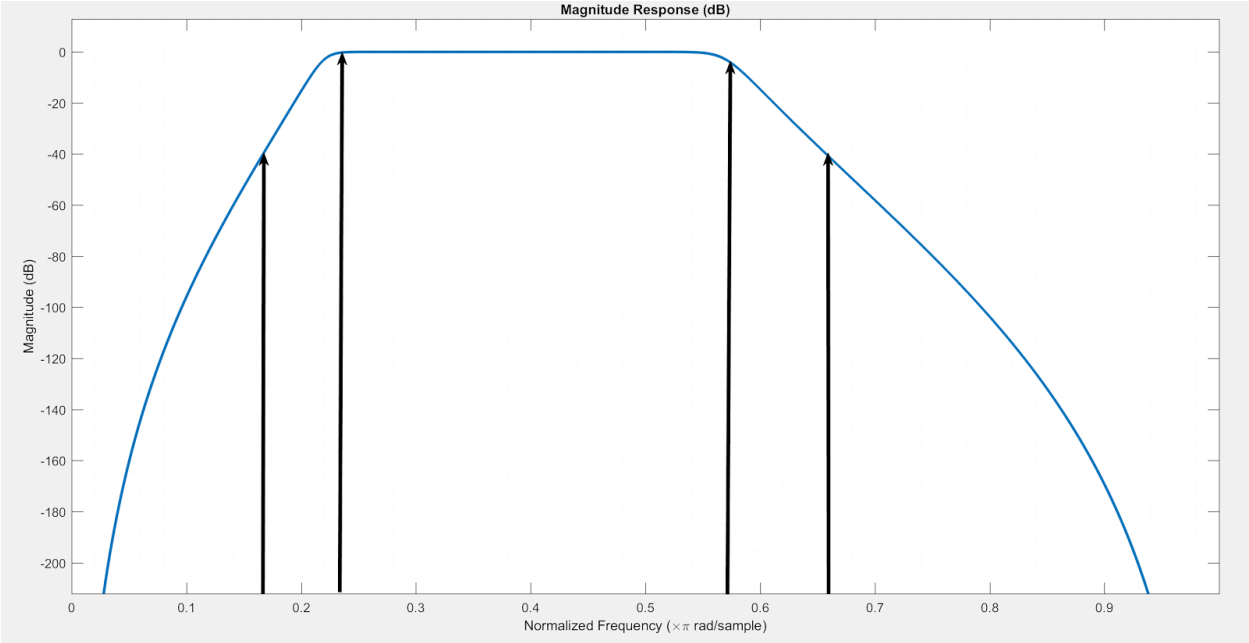

Figure 6: Magnitude Response of IIR Bandpass filter

The stopbands of the IIR bandpass filter are well below -40dB. The transition frequencies are $0.16 \leq \omega_1 \leq 0.227\,\pi$ radians/sample and $0.565 \leq \omega_2 \leq 0.66\,\pi$ radians/sample.