Experiment 9

AIM: Realize an Low pass FIR filter with cutoff 800 Hz , sampling frequency 8000Hz and filter length of 50 using Hamming Window using DSP Board.

Apparatus: DSP Board (DSP6416) , CRO , Function generator

Theory:

In signal processing, a finite impulse response (FIR) filter is a filter whose impulse response (or response to any finite length input) is of finite duration, because it settles to zero in finite time. This is in contrast to infinite impulse response (IIR) filters, which may have internal feedback and may continue to respond indefinitely (usually decaying). The impulse response of an Nth-order discrete-time FIR filter (i.e., with a Kronecker delta impulse input) lasts for N + 1 samples, and then settles to zero. FIR filters can be discrete-time or continuous-time, and digital or analog. For a discrete-time FIR filter, the output is a weighted sum of the current and a finite number of previous values of the input. The operation is described by the following equation, which defines the output sequence y[n] in terms of its input sequence x[n]:

Y[n] = box[n] + b1x[n-1] + b2x[n-2] .....

To design a filter means to select the coefficients such that the system has specific characteristics. The required characteristics are stated in filter specifications. Most of the time filter specifications refer to the frequency response of the filter. There are different methods to find the coefficients from frequency specifications:

1. Window design method

2. Frequency Sampling method

3. Weighted least squares design

Window design method:

In the Window Design Method, one designs an ideal IIR filter, then applies a window function to in the time domain, multiplying the infinite impulse by the window function. This results in the frequency response of the IIR being convolved with the frequency response of the window function. If the ideal response is sufficiently simple, such as rectangular, the result of the convolution can be relatively easy to determine. In fact one usually specifies the desired result first and works backward to determine the appropriate window function parameter(s). Kaiser windows are particularly well-suited for this method because of their closed form specifications.

IIR Vs FIR Filters:

(i) IIR filters are difficult to control and have no particular phase, whereas FIR filters

make a linear phase always possible.

(ii) IIR can be unstable, whereas FIR is always stable.

(iii) IIR is derived from analog, whereas FIR has no analog history. IIR filters make

polyphase implementation possible, whereas FIR can always be made casual..

(iv) FIR filters are dependent upon linear-phase characteristics, whereas IIR filters are used for applications which are not linear.

(v) FIR's delay characteristics is much better, but they require more memory. On the other hand, IIR filters are dependent on both i/p and o/p, but FIR is dependent upon i/p only

(vi) IIR filters consist of zeros and poles, and require less memory than FIR filters, whereas FIR only consists of zeros.

(vii) IIR filters can become difficult to implement, and also delay and distort adjustments can alter the poles & zeroes, which make the filters unstable, whereas FIR filters remain stable.

FIR filters are used for tapping of a higher-order, and IIR filters are better for tapping of lower-orders, since IIR filters may become unstable with tapping higher-orders. FIR stands for Finite IR filters, whereas IIR stands for Infinite IR filters. IIR and FIR filters are utilized for filtration in digital systems. FIR filters are more widely in use, because they differ in response. FIR filters have only numerators when compared to IIR filters, which have both numerators and denominators. Where the system response is infinite, we use IIR filters, and where the system response is zero, we use FIR filters. FIR filters are also preferred over IIR filters because they have a linear phase response and are non recursive, whereas IIR filters are recursive, and feedback is also involved. FIR cannot simulate analog filter responses, but IIR is designed to do that accurately. IIR's impulse response when compared to FIR is infinite.

Program:

```
#include "dsk6713_aic23.h" //support file for codec, DSK

Uint32 fs=DSK6713_AIC23_FREQ_8KHZ; //set sampling rate

#define DSK6713_AIC23_INPUT_MIC 0x0015

#define DSK6713_AIC23_INPUT_LINE 0x0011

Uint16 inputsource=DSK6713_AIC23_INPUT_LINE; // select input

#include <math.h> //for performing modulation operation

static short in_buffer[100];

Uint32 sample_data;

short k=0;

//float filter_Coeff[] = { -0.0017,-0.0020,-0.0024,-0.0027,-0.0021,

//0.0000,0.0044 ,0.0117,0.0221,0.0351,0.0500,0.0655,0.0799,0.0917,

//0.0994,0.1021,0.0994,0.0917,0.0799,0.0655,0.0500, 0.0351,0.0221,

//0.0117,0.0044,0.0000,-0.0021,-0.0027, -0.0024,-0.0020, -0.0017};

// for fc=400Hz

float filter_Coeff[] = { -0.0017, 0.0000,0.0029,-0.0000,-0.0067, 0.0000, 0.0141,-0.0000,-

0.0268, 0.0000, 0.0491,-0.0000,-0.0969,0.0000,0.3156, 0.5008,0.3156, 0.0000,-0.0969,-

0.0000, 0.0491,0.0000,-0.0268,-0.0000,0.0141, 0.0000,-0.0067,-0.0000, 0.0029, 0.0000,-

0.0017};

// for fc=2KHz

short l_input, r_input,l_output, r_output;

void comm_intr();
```

```
void output_left_sample(short);
short input_left_sample();
signed int FIR_FILTER(float *h, signed int);
interrupt void c_int11() //interrupt service routine
{
l_input = input_left_sample(); //inputs data
l_output=(Int16)FIR_FILTER(filter_Coeff ,l_input);
output_left_sample(l_output);
return;
} // end of interrupt routine
signed int FIR_FILTER(float * h, signed int x)
{
int i=0;
signed long output=0;
in_buffer[0] = x; /* new input at buffer[0] */
for(i=31;i>0;i--)
in_buffer[i] = in_buffer[i-1]; /* shuffle the buffer */
for(i=0;i<31;i++)
output = output + h[i] * in_buffer[i];
return(output);
}
void main()
{
comm_intr(); //init DSK, codec, McBSP
while(1);
}
```

Note:

Use Matlab to generate the filter coefficients.

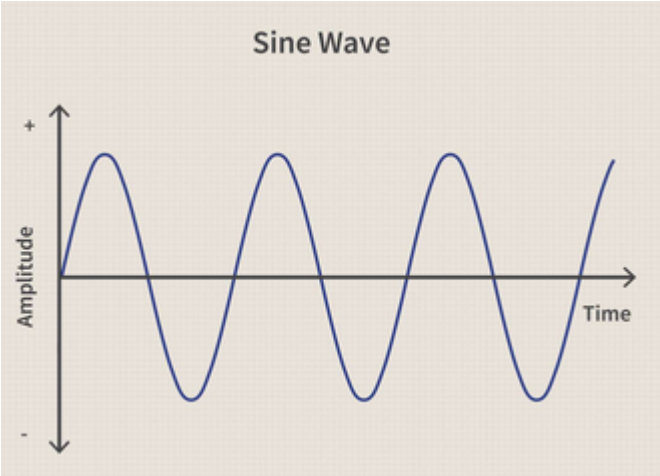
The syntax is fir1(N,2*fc/Fs,'high', hamming(N+1) here fc is cutoff frequency, Fs is sampling rate. fir1 is low pass filter by default.

E.g: 1.To design LPF with cutoff frequency 400Hz and sampling frequency 8000Hz, with order

30, Execute the following command in matlab.

fir1(30,2*400/8000,hamming(31)): This will generate 30 coefficients of LPF

Observation:



Experiment 10

AIM:Design and implementations of IIR Elliptic Band stop filter using Direct form-II structure for the given specification on the DSP board.

Apparatus: DSP Board (DSP6416) , CRO , Function generator

Theory: The infinite impulse response (IIR) filter is a recursive filter in that the output from the filter is computed by using the current and previous inputs and previous outputs. Because the filter uses previous values of the output, there is feedback of the output in the filter structure.

IIR filter can be realize using ditect form , direct form 2 , lattice ladder, parallel and cascade structure.The most basic structure of IIR is Direct firm 2 structure.
For instance, analog electronic filters composed of resistors, capacitors, and/or inductors (and perhaps linear amplifiers) are generally IIR filters. On the other hand, discrete-time filters (usually digital filters) based on a tapped delay line employing no feedback are necessarily FIR filters. The capacitors (or inductors) in the analog filter have a "memory" and their internal state never completely relaxes following an impulse (assuming the classical model of capacitors and inductors where quantum effects are ignored). But in the latter case, after an impulse has reached the end of the tapped delay line, the system has no further memory of that impulse and has returned to its initial state; its impulse response beyond that point is exactly zero

Program:

Create the CCS Project as mentioned in the procedure

```
// iir.c generic iir filter using cascaded second order sections
// 16-bit integer coefficients read from .cof file
#include "DSK6713_AIC23.h" //codec-DSK interface support
Uint32 fs=DSK6713_AIC23_FREQ_8KHZ;//set sampling rate
#define DSK6713_AIC23_INPUT_MIC 0x0015
#define DSK6713_AIC23_INPUT_LINE 0x0011
Uint16 inputsource=DSK6713_AIC23_INPUT_LINE;
#include "bs1800int.cof"
short input_left_sample();
void output_left_sample(short);
void comm_intr();
short w[NUM_SECTIONS][2] = {0};
interrupt void c_int11() //interrupt service routine
```

```
{
short section; // index for section number
short input; // input to each section
int wn,yn; // intermediate and output values in each stage
input = input_left_sample();
for (section=0 ; section< NUM_SECTIONS ; section++)
{
wn = input - ((a[section][0]*w[section][0])>>15) - ((a[section][1]*w[section][1])>>15);
yn = ((b[section][0]*wn)>>15) + ((b[section][1]*w[section][0])>>15) +
((b[section][2]*w[section][1])>>15);
w[section][1] = w[section][0];
w[section][0] = wn;
input = yn; // output of current section will be input to next
}
output_left_sample((short)(yn)); // before writing to codec
return; //return from ISR
}
void main()
{
comm_intr(); //init DSK, codec, McBSP
while(1); //infinite loop
}
```

Note: IIR Bandstop Filter Design Using fdatool

1. open the matlab and change the default location(For e.g C:\Users\student\Documents\MATLAB)to the currently working CCS project(For e.g D:\Proj_name) location.
2. type fdatool in command window

```
>> fdatool
```

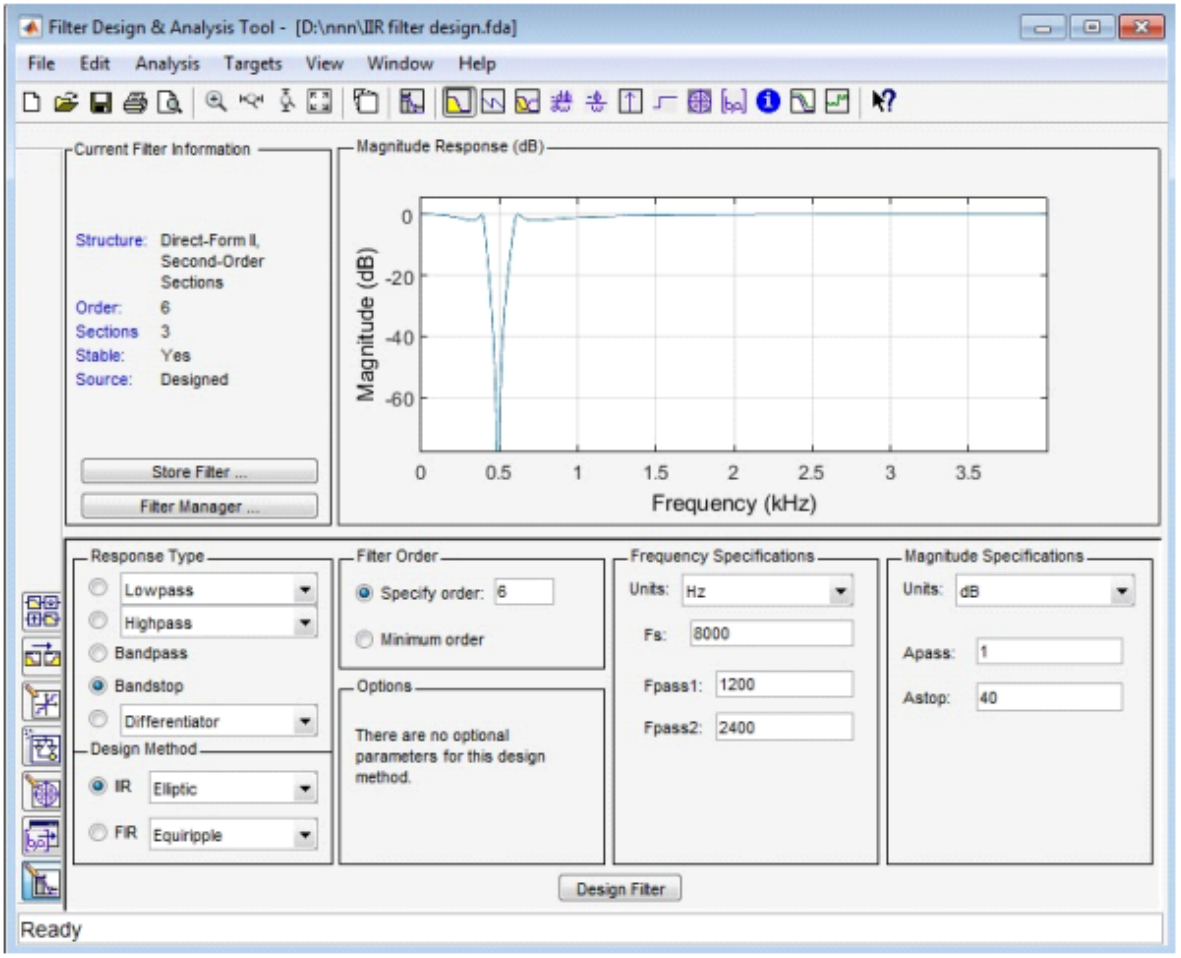


Fig 10 Filter specification in Fdatool

Figure 10 shows the fdatool window corresponding to the design of a sixth order IIR bandstop filter centered at 1800 Hz. The filter coefficients can be exported to the MATLAB workspace by selecting File ’ Export and then setting the parameters Export to, Export as , and Variable Names SOS Matrix , and Scale Values to Workspace, Coefficients, SOS , and G , respectively. Click on Export

3. Copy the file name dsk_sos_iir67int.m from support (C:\dsk6713\support) folder to the

currently working CCS project.

4. Type dsk_sos_iir67int(SOS,G) command in the matlab command window .

>> dsk_sos_iir67int(SOS,G)

Enter the filename bs1800int.cof.

5. The filter coefficient designed by using fdatool is saved into bs1800int.cof.

The following is the example of filter coefficient file,

// bs1800int.cof

// this file was generated automatically using function dsk_sos_iir67int.m

#define NUM_SECTIONS 3

int b[NUM_SECTIONS][3] = {

{11538, -4052, 11538},

{32768, 599, 32768},

{32768, -22852, 32768} };

int a[NUM_SECTIONS][2] = {

{-5832, 450},

{17837, 26830},

{-35076, 27634} };

Observation: