DSP Project
LAB Emulation on TMS320C5505 Processor: FIR, IIR, FFT Implementation
EE 561

Joao Dias 17391

January 2014

Instructor: Ahmet Onat

1 Introduction & Objectives

This project is the emulation of this term labs objectives for DSP course with TMS320C5505 16-bit fixed-point processor.
The achieved objectives are the follow:

Input/Output Signals from the processor

FIR Filter Implementation

IIR Filter Implementation

FFT Implementation

These were the lab objectives during this term. First lab objective was achieved before and the report was already delivered.
Source code and some results will be delivered aside this report since there is a project for each objective and there are some audio results that i want to show ("Audio Files" folder). Graphs and images will be in the report.
The code was developed using 'C' and assembly. Some code was taken from TI repositories, some taken from labs and some written for me.
Due to a problem with the eZDSP board (input jack was damaged) I had to generate digital signals inside the processor emulating the ADC output. This signals are sine waves with different frequencies and they are generated using TI's "55xdsph.lib" library.
Since I've put together some code to work with the audio input and it is working as expected I'll deliver it too. However on this report i will only talk about the code that uses the internal signal generators.
These sine waves will be referred as input from now on.

2 Input/Output Signals From the Processor

This is the base used for all the other implementations since all of them depend on the board input and output.
The board is equipped with a TLV320AIC3204 stereo codec. This device is responsible for the analog to digital signals conversion and vice-versa. The converted signals are then exchanged with the processor using I^2S (special protocol designed for audio communications) and I^2C.
At this stage I just looped the input and output in order to get the input signal on the output. We now that the signal will be almost the same with a little bit off delay introduced by the processing time and with some distortion introduced by the sampling and quantization operations.
I tested the code with two different sampling frequencies and the audio results are annexed to the report as "$IO48000.wav$" and "$IO6857.wav$". The spectrum for 48kHz sampling frequency is shown below.

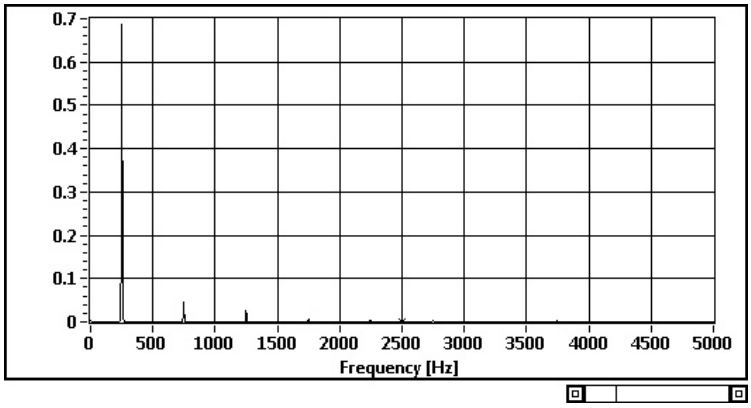

Figure 1: Frequency response for a sinusoidal input with 250Hz frequency.

As we can listen in the provided audio files aliasing effect is obvious in one of the files. With 48kHz the signal is identical to the original sine but with 6857Hz the signal is completely different. It was completely distorted and higher frequencies became reflected in the lower spectrum which is a big problem.

3 FIR Filter Implementation

In the FIR filter implementation I designed two filters with MATLAB. One Low-pass and the other one High-Pass. To test the filters I used a variable sine function that keeps going from 50Hz to 4kHz.
To show the results I recorded again some audio showing the filter effects on the signal. The files are "$50 - 4000_hp.wav$" (high pass) and "$50 - 4000_lp.wav$" (low pass). The new part in the code implementation is just the inclusion of an assembly code from Texas Instruments that implements an FIR filter. A file to hold filter coefficients was created as well. This file is responsible for holding filter coefficients that would be called on "$main$" in order to change input signal value (filtering). The

filter is implemented in assembly due to performance issues.
Filters frequency spectrum in response to the variable sine are shown below.

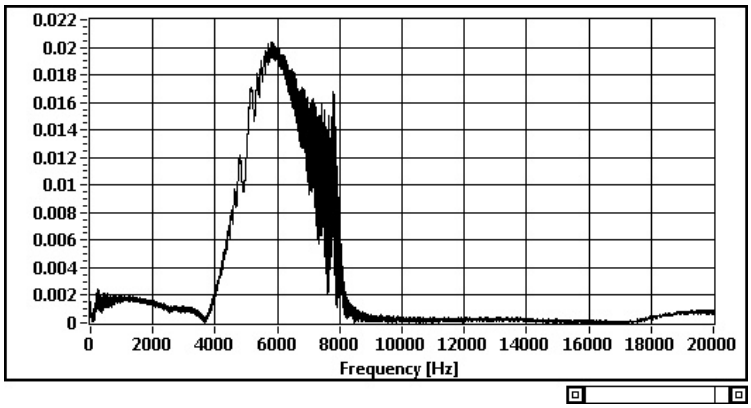

Figure 2: Filter Output spectrum in response to a variable sine function from 50Hz-8000Hz(High Pass filter with 6kHz Cutoff frequency).

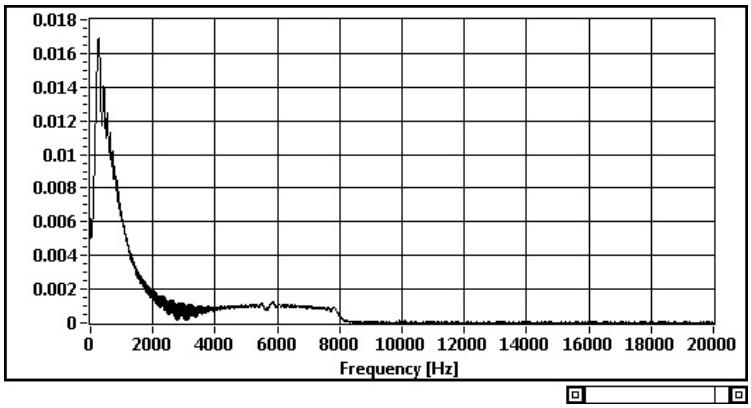

Figure 3: Output spectrum in response to a variable sine function from 50Hz-8000Hz (Low Pass filter with 300Hz Cutoff frequency).

To compare the filters real frequency response with the theoretical one I took the magnitude values (100Hz steps) for some frequencies and made a graph. The graphs that I made can be compared with the ones generated from MATLAB when I designed the filter. The magnitude units are different. The ones from MATLAB are in dB and the ones that I measure are in magnitude units. However it's easy to understand that when the signal is more attenuated it's value will decrease and when it's less attenuated it's value will stay the same. Keeping that in mind we can compare both frequency responses.

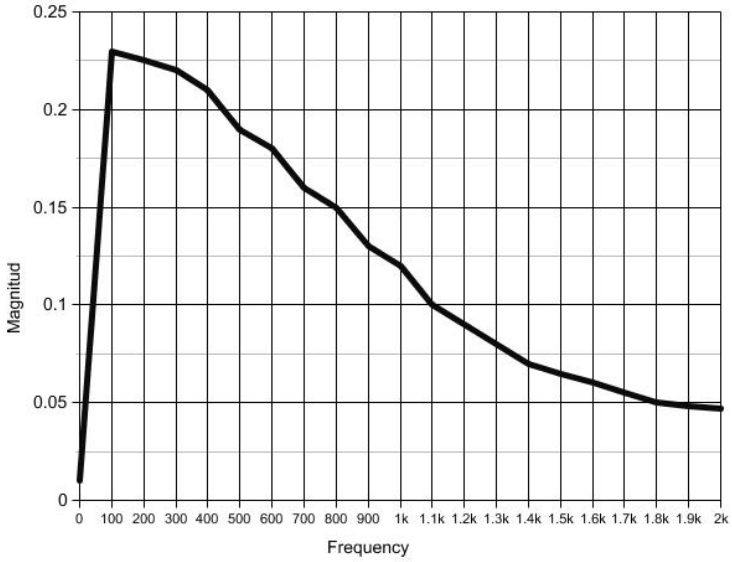

Figure 4: Measured frequency response for the low pass filter with 300Hz cutoff frequency.

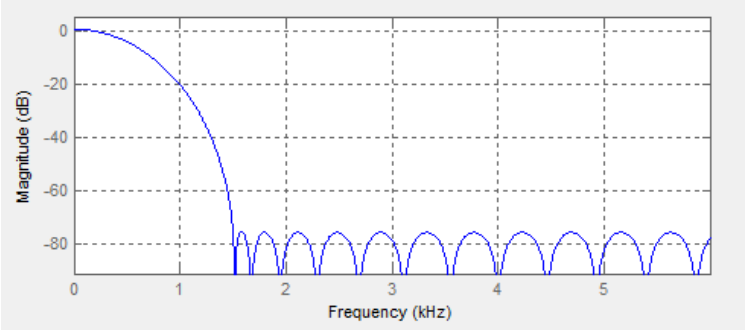

Figure 5: Theoretical frequency response for the low pass filter with 300Hz cutoff frequency.

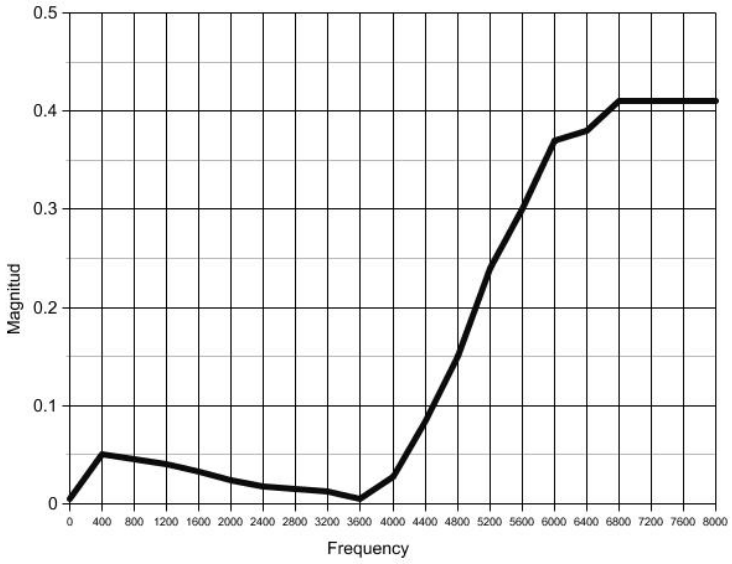

Figure 6: Measured frequency response for the high pass filter with 6kHz cutoff frequency.

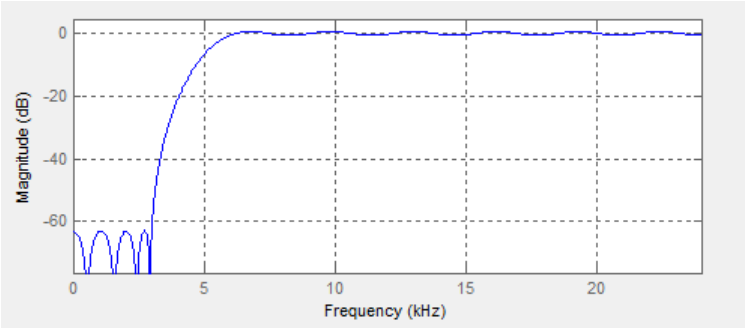

Figure 7: Theoretical frequency response for the high pass filter with 6kHz cutoff frequency.

4 IIR Filter Implementation

For the IIR implementation I made the same as for the FIR filters. I implemented two IIR filters, a low pass and a high pass and I inputted again the same variable sine function from 50Hz to 8000Hz. The audio proved to be really clean with these filters as you can ear with "$lp300.wav$" and "$hp300.wav$". This make sense because IIR filters will attenuate more the frequencies outside its pass band.
In terms of code I'm using a '.c" file to hold the filters related code. This code uses one header to hold coefficients values (numerator and denominator). Both values are read into the ".c" file and used in a two stage direct form II implementation from Texas Instruments. I send my input to the filtering function with the desired filter coefficients and I receive the filter output ready to put in codec's line out.
I performed the same measurements for IIR as for FIR. I measured the output in response to the variable sine signal and frequency response of the filter.

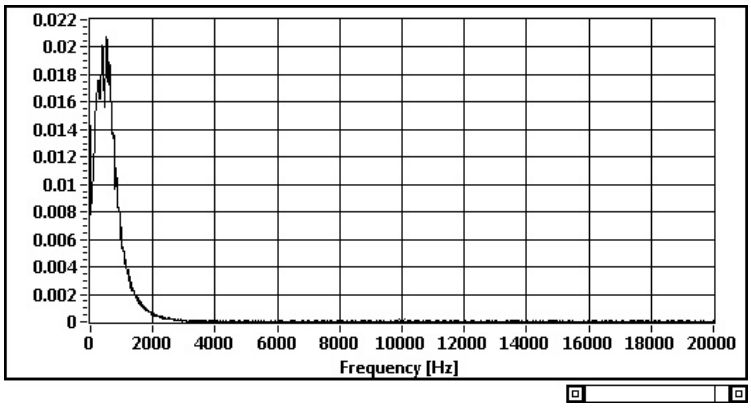

Figure 8: Spectrum of low pass IIR filter (Cutoff frequency 300Hz) with a variable sine input.

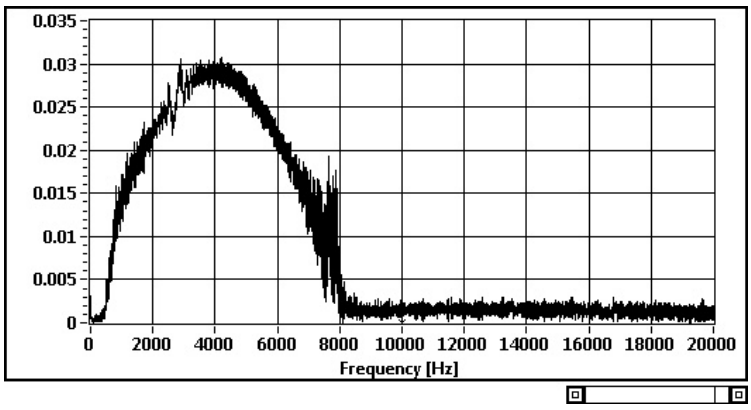

Figure 9: Spectrum of high pass IIR filter (Cutoff frequency 300Hz) with a variable sine input.

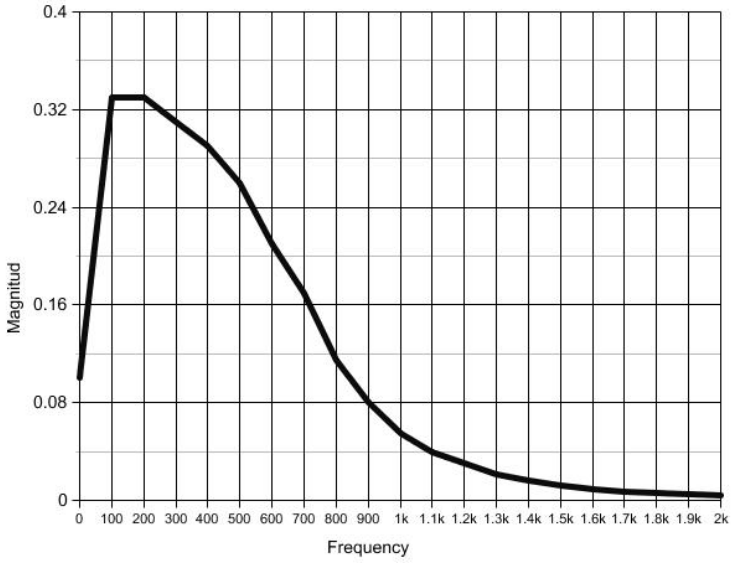

Figure 10: Measured frequency response for a low pass IIR filter (Cutoff frequency 300Hz).

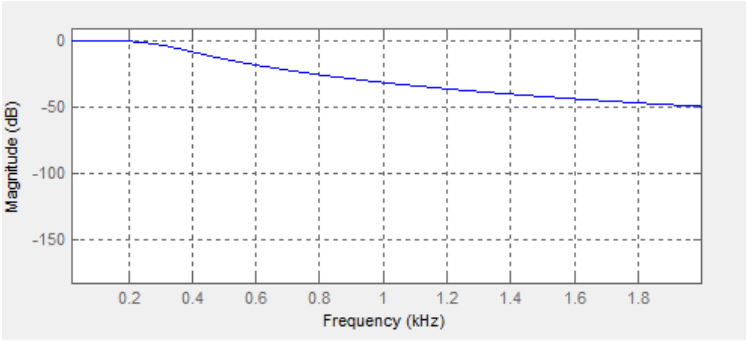

Figure 11: Theoretical frequency response for a low pass IIR filter (Cutoff frequency 300Hz).

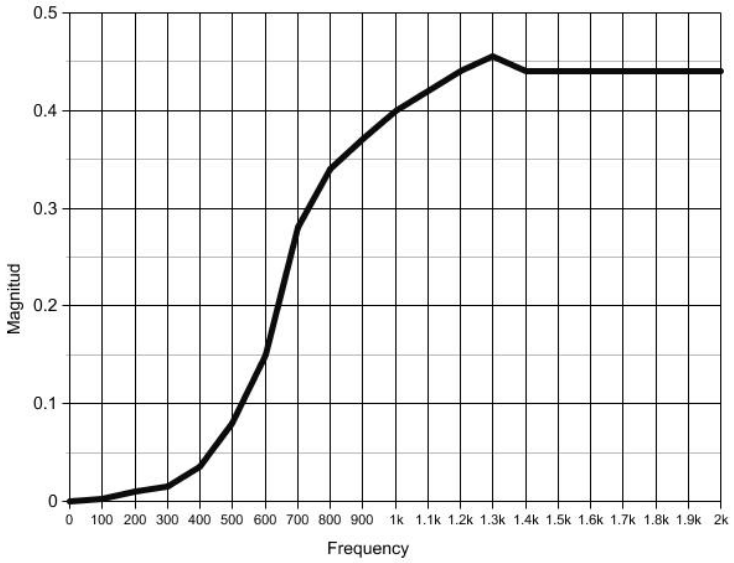

Figure 12: Measured frequency response for a high pass IIR filter (Cutoff frequency 300Hz).

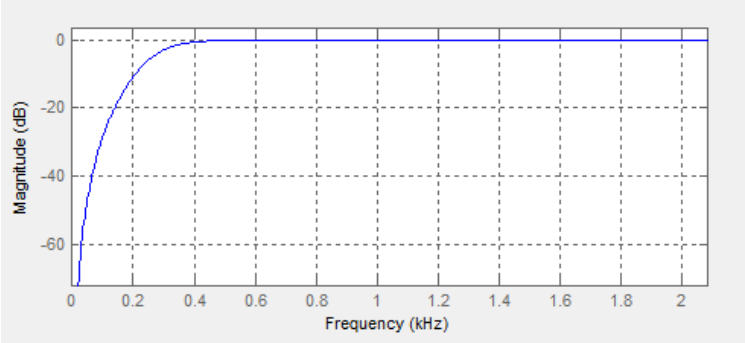

Figure 13: Theoretical frequency response for a high pass IIR filter (Cutoff frequency 300Hz).

5 FFT Implementation

In order to implement the FFT I used the code provided in the lab and mixed it with mine. I followed the steps to make the implementation and I created all the missing functions.
I read data from my sine generator input and I send it to the input buffer, waiting till it fills up with samples. When this operation is completed I perform a rotation between the 3 buffers (I used "memcopy" function to do that since we can't assign a vector to another vector directly.
After the rotation the FFT is performed over the intermediate buffer and the output buffer is sent to the board output line.
I was not able to get any interesting output results from CCS. I think that the code is well implemented but I just didn't manage to get any output. Regarding that I decided to implement a FFT on MATLAB. The code is this one:

```
Fs = 2500; % Sampling frequency
t = 0:1/Fs:1; % Time vector of 1 second
f = 1000; % frequency
x = sin(2*pi*t*f);
N = 512; % Length of FFT
X = fft(x,N);
X = X(1:N/2);
mx = abs(X);
f = (0:N/2-1)*Fs/N;
figure(1);
plot(t,x);
title('Sine Wave Signal');
xlabel('Time (s)');
ylabel('Amplitude');
figure(2);
plot(f, mx);
xlabel('f (Hz)');
ylabel('Power');
```

Figure 14: MATLAB code to implement a FFT for a sinusoidal input of 1kHz with 2.5kHz sampling frequency.

The result is the expected as shown in Figure 15. We got a peak at 1kHz frequency representing the sine function frequency. If we change the sampling frequency obviously we'll have different results. I choose 2.5kHz because is more than two times the sine frequency, but the value could be another one.

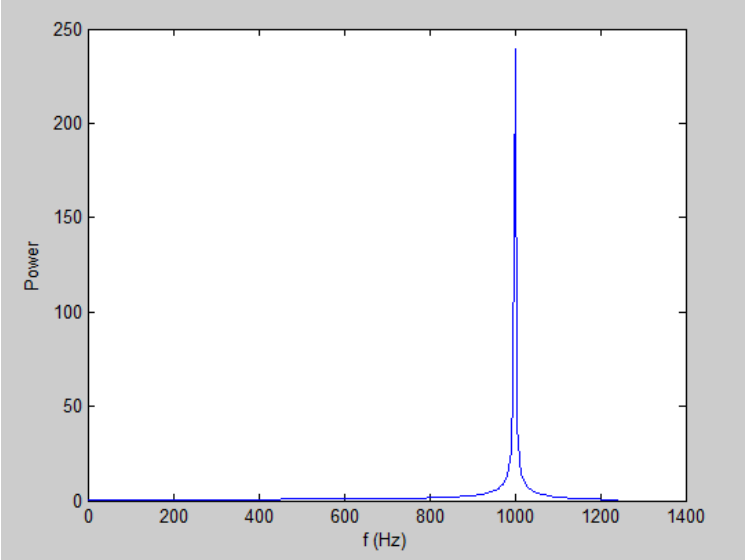

Figure 15: MATLAB spectrum for a FFT with sinusoidal input.

6 Conclusions & Achievements

I felt like going into the class's theory and I even understood better some concepts after this project.
Before this project I was a little bit confused regarding IIR filters. I didn't know what were the characteristics that made them so different from FIR filters. Now I understand that they are not just harder to implement but they are not always better too. For a lower order filter as in this project they proved to be better and we can hear that in the audio files. However for bigger order filters we can have instability and a lot of confusion with this kind of filter.
Another good thing was to work with CCS and TMS320C5505 since I never worked with either one of these.
Unfortunately my board's input jacket became damaged and I was not able to get the audio results that I want and I couldn't some DSP functions that I wanted to use but I think that the solution that I found was enough to show the required concepts on this project.