**FFT Algorithm developed using Split Radix Algorithm**

**[Software Report [21/09/2018]]**

**Description:**

* This code is developed as a modified version of the previous code developed using Cooley- Tukey Algorithm
* The major difference between split radix and Cooley-Tukey algorithm is that split radix algorithm uses more number of splittings (4 times) of the input matrix (whose FFT needs to be calculated) whereas in Cooley Tukey algorithm we used only two splits of the input vector.
* In this code we have given a double tone sinusoidal signal with frequencies 280 and 750 to test the Split Radix FFT algorithm we developed.
* In the Split Radix algorithm we have used the butterfly structure to compute the FFT using 4-point DFT method. The larger the number of splits in the input vector the faster the computation of the output.
* In this case the N values must be a power of 4 to ensure that splitting of the input vector occurs in a proper way.
* With this code we could get output as desired with impulses at the two input frequencies of the sinusoid in the output frequency response plot.
* While Split radix code ensures faster calculation of output the problem with it is that it consumes more space when compared to the Cooley Tukey algorithm.

**MATLAB CODE:**

**MAIN PROGRAM :**

clc;

clear all;

close all;

t = 0:1/1000:1;

x = sin(2\*pi\*280\*t)+sin(2\*pi\*750\*t);

N=input('Enter the value of N');

%N indicates the number of times the dft is split to compute fft (it

%decides accuracy of fft generated)

z = fft\_cal(x,N);

subplot(2,1,1);

plot(t,x);

xlabel('time');

ylabel('amplitude');

title('Input Signal');

subplot(2,1,2);

plot(abs(z));

xlabel('frequency');

ylabel('magnitude');

title('Magnitude Plot');

**FFT FUNCTION (SPLIT RADIX CODE ):**

function p = fft\_cal1(x,M)

N = 4^M;

L=length(x);

if(N<L)

error('N should be greater than L ');

end

%to ensure that the subsequent splittings occur without any problem

xn = x;

xn=[xn zeros(1,N-L)];

xe=xn(2:2:end);

xo1=xn(1:4:end);

xo3=xn(3:4:end);

for k=0:N-1

for r=0:N/2-1

Wne=exp(-1j\*pi\*k\*r\*4/N);

Xe(k+1,r+1) = Wne;

end

end

for k=0:N-1

for r=0:N/4-1

Wno=exp(-1j\*pi\*k\*r\*8/N);

Xo(k+1,r+1) = Wno;

end

end

for k=0:N-1

Wn1=exp(-1j\*2\*pi\*k/N);

Wn3=exp(-1j\*6\*pi\*k/N);

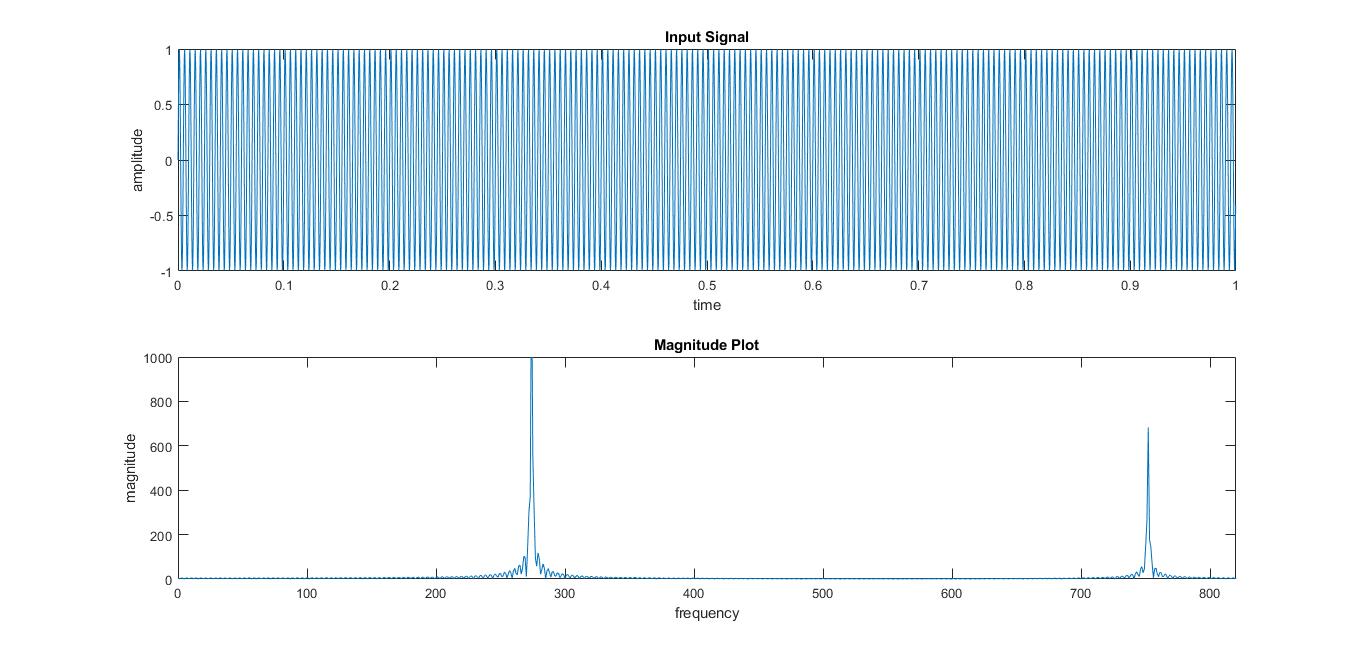
end

p = Xe\*xe' + Wn1\*(Xo\*xo1')+Wn3\*(Xo\*xo3');

end

In this code since we went for further splitings this one was more efficient than the previous one based on the time consumed.

OUTPUT PLOTS:



Thank you,

Akash & Ritheesh.