**DEPARTMENT**

***of***

**ELECTRONICS AND COMMUNICATION ENGINEERING**



**Academic Year: 2022-2023**

**DIGITAL SIGNAL PROCESSING**

**LAB INSTRUCTOR MANUAL**

# VISION

* To initiate high quality technical education and to nurture young minds towards creative thinking that inspires them to undertake innovations in the field of Electronics and Communication Engineering (ECE) and be competent in the global arena..
* To Emphasize on the student body to carry out research for the service of our Nation and to the Society at large.

# MISSION

* Constantly upgrade engineering pedagogy that caters to the growing challenges of the Industry.
* Develop conceptual learning that leads towards critical and innovative thinking.
* Establish good harmony with industry that fills the gap between academia and the outside world enabling the students to prepare for diverse and competitive career paths.
* To endorse higher studies and pursue research in the ECE discipline with sensitivity towards societal requirements.

# PROGRAMME EDUCATIONAL OBJECTIVES

**PEO I**

To enable graduates to pursue research, or have a successful career in academia or industries associated with Electronics and Communication Engineering, or as entrepreneurs.

**PEO II**

To provide students with strong foundational concepts and also advanced techniques and tools in order to enable them to build solutions or systems of varying complexity.

**PEO III**

To prepare students to critically analyze existing literature in an area of specialization and ethically develop innovative and research oriented methodologies to solve the problems identified.

## PROGRAM SPECIFIC OBJECTIVES (PSOs)

1. To analyze, design and develop solutions by applying foundational concepts of electronics and communication engineering.
2. To apply design principles and best practices for developing quality products for scientific and business applications.
3. To adapt to emerging information and communication technologies (ICT) to innovate ideas and solutions to existing/novel problems.

## PROGRAM OUTCOMES

**Engineering Graduates will be able to:**

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one‟s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

# List of Experiments

|  |  |  |
| --- | --- | --- |
| **S. No** | **Name of the Experiment** | **Page No** |
|  | **SCILAB/MATLAB Programs** |  |
| **1** | Generation of Various Waveforms |  |
| **2** | Computation of Linear Convolution and Circular Convolution |  |
| **3** | Computation of Auto Correlation and Cross Correlation |  |
| **4** | Spectrum Analysis using DFT and FFT |  |
| **5** | Design of IIR-Butterworth Filters and Chebyshev Filters |  |
| **6** | Design of FIR Filters using Windows |  |
| **7** | Computation of Up Sampling and Down Sampling |  |
|  | **DSP Processor Experiments** |  |
| **8** | Study of addressing modes of DSP Processor using CCS debugging Software |  |
| **9** | Generation of Various Waveform using TMS 320C 6713 |  |
| **10** | Implementation of IIR Filters using TMS320C6713 |  |
| **11** | Implementation of FIR Filters using TMS320C6713 |  |
| **12** | Performing Up sampling and Down sampling using TMS320C6713 |  |
| **13** | Implementation of Linear Convolution and Circular Convolution using TMS 320C6713 |  |
| **14** | Implementation of basic Arithmetic and Logical operations Using TMS320C5416 |  |
|  | **Additional Experiments** |  |
| **15** | 1-D Signal Analysis – ECG and Speech Signal |  |
| **16** | 2-D signal Analysis - Image Enhancement and  Segmentation |  |
|  | **APPENDIX** |  |
| **A** | TMS3206713 Floating Point Digital Signal Processor |  |
| **B** | TMS320C54X Fixed Point Digital Signal Processor |  |

# SCILAB PROGRAMS

**Expt.No.1 Generation of Various Waveforms**

**AIM:**

To generate the given set of basic signals such as unit step, impulse, ramp, rectangular, triangular, sine and cosine signals.

**REQUIREMENTS:**

SCILAB, Personal computer.

**THEORY:**

**UNIT IMPULSE SIGNAL:**

Unit impulse signal is represented by an unit amplitude when t=0 and zero elsewhere.A δ(t) = 0, t ≠ 0 and ∫δ(t)dt = 1

In discrete time domain, the unit impulse signal is called a unit sample signal. It is defined as δ(n) = 1, n = 0

0, n ≠ 0

**UNIT STEP SIGNAL:**

Unit step signal is causal in nature and is unity for positive time period axis.

The discrete-time unit step signal is defined as

U(n) = 0, n < 0

1, n ≥ 0

**UNIT RAMP SIGNAL :**

Unit ramp signal is an increasing function of slope =1 and is represented by line x = y. In discrete-time domain, the unit-ramp signal is defined as r(n) = 0, n < 0

n, n ≥ 0

**EXPONENTIAL SIGNAL:**

It may be either growing or decaying in nature. They are also known as rising exponential or decaying exponential.

**SINE/COSINE WAVEFORM:**

Based on the frequency, Sine and Cosine waveforms are generated.

**ALGORITHM:**

**SINE SEQUENCE:**

Step 1: Initialize the time period.

Step 2: Design the function of sine wave.

Step 3: Plot the sine sequence.

Step 4: Display the output

**COSINE SEQUENCE:**

Step 1: Initialize the time period.

Step 2: Design the function of cosine wave.

Step 3: Plot the cosine sequence.

Step 4: Display the output

**RAMP SEQUENCE:**

Step 1: Read the length of the ramp sequence.

Step 2: Define the length of the ramp sequence.

Step 3: Plot the ramp sequence using stem function.

Step 4: Display the output.

**EXPONENTIAL SEQUENCE:**

Step 1: Read the length of the exponential sequence.

Step 2: Read its amplitude.

Step 3: Plot the exponential sequence using stem function.

Step 4: Display the output.

**UNIT-STEP SEQUENCE:**

Step 1: Read the length of the unit-step sequence.

Step 2: Plot the unit-step sequence.

Step 3: Display the output.

**UNIT-IMPULSE SEQUENCE:**

Step 1: Read the length of the unit-impulse sequence.

Step 2: Plot the unit-impulse sequence.

Step 3: Display the output.

## a. Generation of Signals(Continuous)

**PROGRAM:**

clc clear close

*//Continuous time signal generation*

*// Impulse Signal* t=0:0.01:1 [r,c]=size(t)

delta=[zeros(1,(c-1)/2), ones(1,1), zeros(1,(c-1)/2)]

subplot(421) plot(t,delta) xlabel('Time (s)') ylabel('Amplitude') title('Impulse Signal')

*// Step Signal with step size A*

*// For unit step signal, keep A=1*

A=2; t1=0:0.01:10 [r1,c1]=size(t1) u=A\*ones(1,c1) subplot(422) plot(t1,u) xlabel('Time (s)') ylabel('Amplitude') title('Step Signal')

*// Step Signal with step size A*

*//For unit Ramp signal keep A=1*

A=2; t2=0:0.01:10 r=A\*t2 subplot(423) plot(t2,r) xlabel('Time (s)') ylabel('Amplitude') title('Ramp Signal')

*//Parabolic Signal* A=3 t3=0:0.01:2 p=(A\*t^2)/2 subplot(424) plot(t,p) xlabel('Time (s)') ylabel('Amplitude') title('Parabolic Signal')

*//Sinusoidal Signal* t4=0:0.001:1 f=2; *// 2 Hz signal* Sin\_Sig=A\*sin(2\*%pi\*f\*t4) subplot(425) plot(t4,Sin\_Sig) xlabel('Time (s)') ylabel('Amplitude') title('Sinusoidal Signal')

*// Cosine Signal* t5=0:0.01:1 f1=3

Cos\_Sig=A\*cos(2\*%pi\*f1\*t5) subplot(426) plot(t5,Cos\_Sig) xlabel('Time (s)') ylabel('Amplitude') title('Cosinudoidal Signal')

*// Eponential Signal*

t6=0:0.01:1 A2=2 b=2

Exp\_Sig=A2\*%e^(b\*t6) subplot(427) plot(t6,Exp\_Sig) xlabel('Time (s)') ylabel('Amplitude') title('Exponential Signal')

**b. Generation of Signals(Discrete)**

// Generation of Unit sample Sequence n=-2:1:2

del\_discrete=[zeros(1,2), ones(1,1), zeros(1,2)]

figure subplot(231) plot2d3(n,del\_discrete) xlabel('Time (s)') ylabel('Amplitude') title('Unit Impulse Sequence') // Unit Step Sequence

n1=0:1:5 N=length(n1) Unit\_Step=ones(1,N) subplot(232) plot2d3(n1,Unit\_Step) xlabel('Discrete Time n') ylabel('Amplitude') title('Unit Step Sequence') //Ramp sequence

n2=0:1:10 Unit\_ramp=n2 subplot(233) plot2d3(n2,Unit\_ramp) xlabel('Discrete Time n') ylabel('Amplitude')

title('Unit ramp Sequence')

// Exponential Sequence a=0.6 n3=0:0.5:10 Exp\_seq=a^n3 subplot(234) plot2d3(n3,Exp\_seq) xlabel('Discrete Time n') ylabel('Amplitude') title('Exponenetial Sequence')

//Sinusoidal Sequence

n4=-1:0.1:1 A6=2 f2=1

Sin\_Dis=A\*sin(2\*%pi\*f2\*n4) subplot(235) plot2d3(n4,Sin\_Dis) xlabel('Discrete Time n') ylabel('Amplitude')

title('Discrete Time Sinusoidal Sequence')

//Cosinusoidal Sequence

n4=-1:0.1:1 A6=2 f2=1

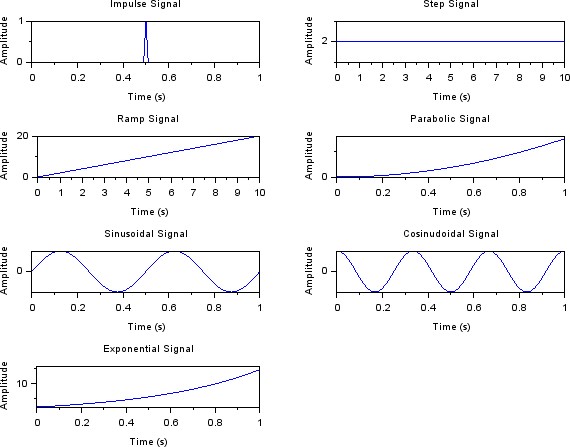
Sin\_Dis=A\*cos(2\*%pi\*f2\*n4)

subplot(236) plot2d3(n4,Sin\_Dis) xlabel('Discrete Time n') ylabel('Amplitude')

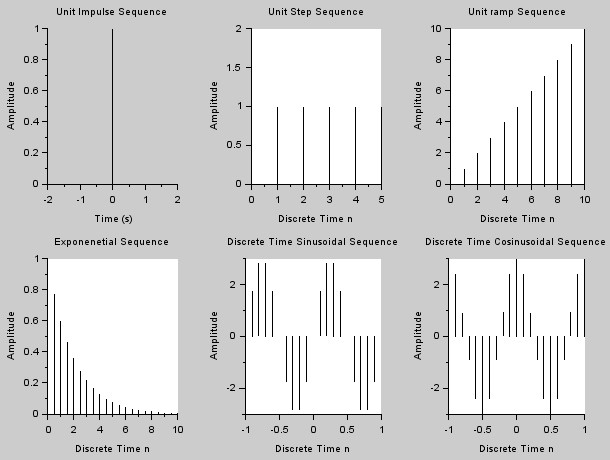
title('Discrete Time Cosinusoidal Sequence')

**OUTPUT:**

**Continuous Time Signals:**



**Discrete Time Signal**



**Viva Questions:**

What is a Signal?

A signal is a function of one or more independent variables which contain some information.

Eg: Radio signal, TV signal, Telephone signal etc.

Define unit step, ramp and delta functions for CT.

Unit step function is defined as U(t)= 1 for t >= 0

0 otherwise

Unit ramp function is defined as r(t)= t for t>=0

0 for t<0

Unit delta function is defined as

δ(t)= 1 for t=0

0 otherwise Define CT signals.

Continuous time signals are defined for all values of time. It is also called as an analog signal and is represented by x(t). Eg: AC waveform, ECG etc.

Define DT signal.

Discrete time signals are defined at discrete instances of time. It is represented by x(n). Eg: Amount deposited in a bank per month.

**CONCLUSION:**

Thus, various signals and waveforms are generated using Scilab and are plotted successfully.

**Expt.No.2 Computation of Linear Convolution and Circular Convolution**

**AIM:**

To perform linear convolution of two given sequences.

**REQUIREMENTS:**

SCILAB, Personal computer.

**THEORY:**

Convolution is a mathematical operation equivalent to finite impulse response (FIR) filtering. *Convolution is important in digital signal processing because convolving two sequences in the time domain is equivalent to multiplying the sequences in the frequency-domain.* Convolution finds its application in processing signals especially analyzing the output of a system. Consider the signals x1(n) and x2(n). Convolution of these two signals is given by x3(n) = x1(n) \* x2(n)

**ALGORITHM:**

1. Read the input sequences x(n), h(n).
2. Calculate the length of the sequences.
3. Find the length of convolved sequence.
4. Plot the input sequences.
5. Perform linear convolution between x(n) and h(n).
6. Display the convolved sequence.
7. Plot the convolved sequence.

**A. Linear Convolution**

**PROGRAM:**

clc clear close

x=input('enter the sequence :') h=input('enter the sequence:') lx=length(x) lh=length(h) x=[x,zeros(1,lh)] h=[h,zeros(1,lx)] y=zeros(1,lx+lh-1) for n=1:lx+lh-1 y(n)=0 for m=1:lx+lh-1 if(m<n+1) y(n)=y(n)+x(m)\*h(n-m+1)

end end end

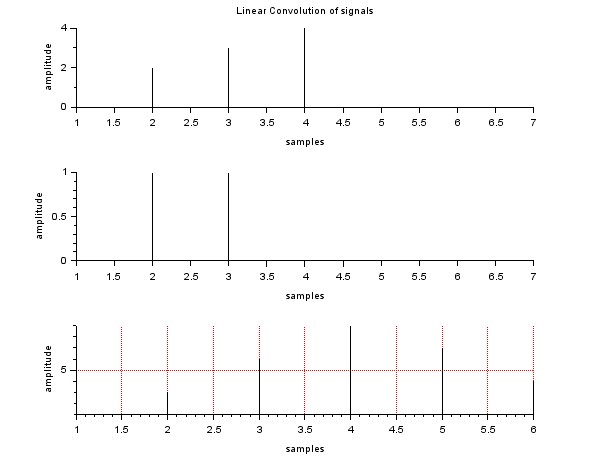
*//y=conv(x,h)*

subplot(3,1,1); plot2d3(x); xlabel('samples'); ylabel('amplitude'); title('Linear Convolution of signals');

subplot(3,1,2); plot2d3(h); xlabel('samples');

ylabel('amplitude');

subplot(3,1,3); plot2d3(y); xgrid(5, 1, 7) xlabel('samples'); ylabel('amplitude'); disp('x=',x) disp('h=',h) disp('Y=',y)

**OUTPUT:**

**Viva Questions:**

1. What is the length of the linearly convolved signal?

Length of output=(length of input1+length of input2) -1

1. What is the difference between linear convolution and circular convolution?

Linear convolution is the basic operation to calculate the output for any linear time invariant system given its input and its impulse response. Circular convolution is the same thing but considering that the support of the signal is periodic (as in a circle, hence the name).

1. Multiplication of two signals in time domain is equal to Convolution of the two signals in frequency domain.
2. What are the different methods of linear convolution?

Tabular method

Graphical method

**CONCLUSION:**

Thus, linear convolution is performed between two sequences successfully.

b. **Circular Convolution**

**AIM:**

To perform circular convolution of two given sequences.

**REQUIREMENTS :**

SCILAB, Personal computer.

**THEORY:**

The convolution property of DFT says that, the multiplication of the DFT‟s of the two sequence is equivalent to the DFT of the circular convolution of the two sequences. X1(K) .X2(K) = DFT{ x1(n) \* x2(n)}

**ALGORITHM:**

1. Read the input sequences x1(n), x2(n).
2. Calculate the length of the sequences.
3. If the lengths are not equal, do zero padding.
4. Plot the input sequences.
5. Perform circular convolution between x1(n) and x2(n).
6. Display the convolved sequence.
7. Plot the convolved sequence.

**PROGRAM:**

clc clear close

x=[1 2 3 4 5]*//input('enter the elements of x:');* h=[1 1 1 1]*//input('enter the elements of h:');* l=length(x); m=length(h) N=max(l,m); if l<N

x=[x,zeros(1,N-l)]

end if m<N h=[h,zeros(1,N-m)] end

*////x=[x,zeros(1,N-l)]; ////h=[h,zeros(1,N-1)];* y=zeros(1,N); for i=1:N y(i)=0 for j=1:N k=i-j+1

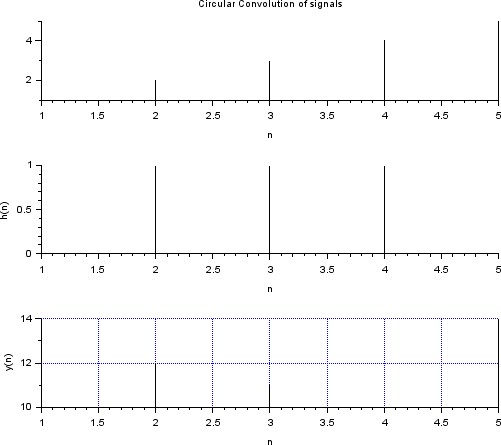
if ( k<=0) k=i-j+1+N end y(i)=y(i)+x(j)\*h(k); end end subplot(3,1,1); plot2d3(x);

title('Circular Convolution of signals');

xlabel('n'); subplot(3,1,2); plot2d3(h); xlabel('n'); ylabel('h(n)') subplot(3,1,3); plot2d3(y); xgrid(2) xlabel('n'); ylabel('y(n)')

disp('output',y)

**OUTPUT:**



**Viva Questions:**

1. What is Zero Padding?

**Zero padding** is a simple concept; it simply refers to adding zeros to end of a time-domain signal to increase its length

1. What is the length of the circularly convolved signal?

In the case of circular convolution, length of output= length of input1=length of input2

1. What is the difference between overlap add method and overlap save method?

**Overlap-Save:**

The overlap-save procedure cuts the signal up into equal length segments with some overlap. Then it takes the DFT of the segments and saves the parts of the convolution that correspond to the circular convolution. Because there are overlapping sections, it is like the input is copied therefore there is not lost information in throwing away parts of the linear convolution. **Overlap-Add**

The overlap-add procedure cuts the signal up into equal length segments with no overlap. Then it zero-pads the segments and takes the DFT of the segments. Part of the convolution result corresponds to the circular convolution. The tails that do not correspond to the circular convolution are added to the adjoining tail of the previous and subsequent sequence. This addition results in the aliasing that occurs in circular convolution.

**CONCLUSION:**

Thus, circular convolution is performed between two sequences successfully.

**Ex. No. 03 Computation of Auto Correlation and Cross Correlation**

**AIM:**

To perform Auto Correlation and Cross Correlation for the discrete sequences.

**REQUIREMENTS :**

SCILAB, Personal computer.

**THEORY:**

Autocorrelation, sometimes known as serial correlation in the discrete time case, is the correlation of a signal with a delayed copy of itself as a function of delay. It is often used in signal processing for analyzing functions or series of values, such as time domain signals.

Cross-correlation is a measurement that tracks the movements of two or more sets of time series data relative to one another. It is used to compare multiple time series and objectively determine how well they match up with each other and, in particular, at what point the best match occurs.

**ALGORITHM:**

1. Read the input sequences.
2. Enter the starting index.
3. Perform the correlation.
4. Plot the input sequences.
5. Plot the Correlated result.

**PROGRAM:**

**//a. Auto Correlation** clc; clear;

x =input( 'Enter the input Sequence: ' );

m =length(x);

lx =input( ' Enter the Starting time index of the input Sequence: ' ) hx = lx+m -1; n = lx :1: hx; x\_fold = x($ : -1:1) ; nx = 2\*lx; nh = 2\*m+nx-2; r = nx:nh;

Rxx =convol(x, x\_fold );

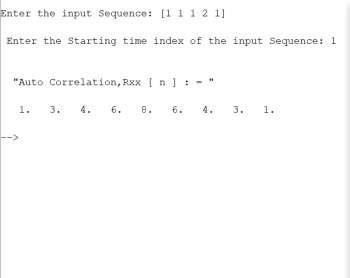
disp('Auto Correlation,Rxx [ n ] : = ',Rxx); colordef("white"); plot2d3(r,Rxx);

**//b. Cross Correlation** clc; clear;

x =input( 'Enter the First Input Sequence:' ); y =input( 'Enter the Second Input Sequence:' ) mx =length(x); my =length(y); lx =input( 'Enter the lower time index of the First Input Sequence:' ) ly =input( 'Enter the lower time index of the Second Input Sequence:' ); hx = lx+mx -1; n = lx :1: hx; x\_fold = x($ : -1:1) ; y\_fold = y($ : -1:1) ; nx = lx+ly;

ny = nx+mx+my -2; r = nx:ny;

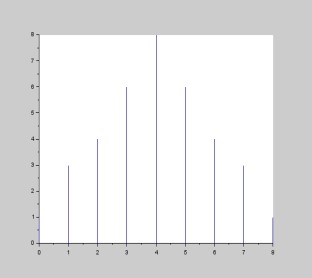
Rxy =convol(x, y\_fold ); Ryx =convol( x\_fold ,y); disp('Cross Correlation Rxy [ n ] : = ',Rxy ); disp('Cross Correlation Ryx [ n ] : = ',Ryx ); colordef("white"); plot2d3(r,Rxy);



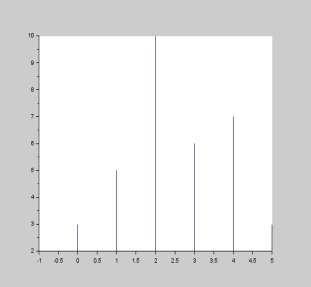
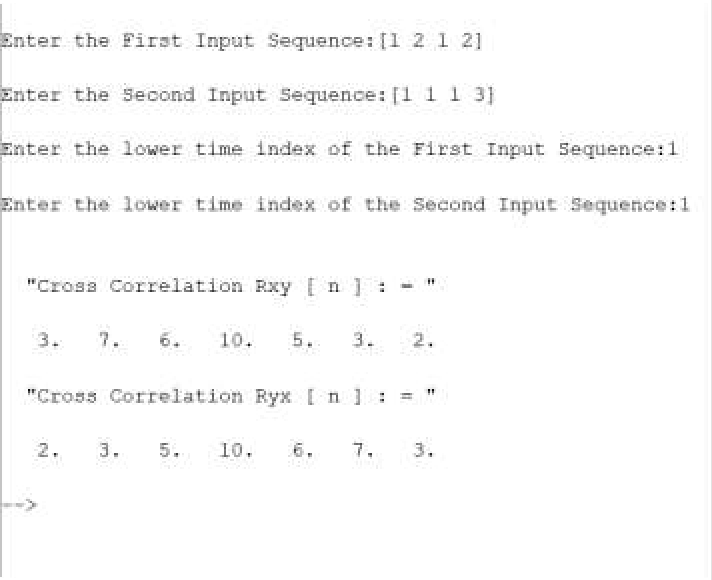
**Output**

**of Auto**

**Correlation:**



**Output of Cross Correlation**



**Viva Questions:**

**List some of the fields where correlation concepts used**

Autocorrelation and cross correlation have been widely used in many engineering and basic science fields, including electrical, acoustic, and geophysical applications, to name only a few, where noise isolation, spatial similarity between signals, and feature detection is of interest.

**What are the applications of correlation in DSP?**

It is commonly used for searching a long signal for a shorter, known feature. It has applications in **pattern recognition, single particle analysis, electron tomography, averaging, cryptanalysis, and neurophysiology**. The cross-correlation is similar in nature to the convolution of two functions.

**How do you find the autocorrelation of a signal?**

Autocorrelation (for example sound signals)

* Finding the value of the signal at a time t,
* Finding the value of the signal at a time t + τ,
* Multiplying those two values together,
* Repeating the process for all possible times, t, and then.
* Computing the average of all those products.

**CONCLUSION:**

Thus, the auto and cross-correlation on the discrete sequence was performed and the result was plotted.

**Expt.No:4 Spectrum Analysis using DFT and FFT**

**AIM:**

To write a SCILAB program to get the single sided amplitude spectrum using FFT.

**REQUIREMENTS:**

Personal Computer with SCILAB

**PROGRAM**

**a. Program for calculation of DFT of a signal** clc ; clf ; clear ;

x= input ( 'Enter the input sequence' );

N=length(x);

//N= input ( 'Enter the value length of input sequence' ); for k =1: N y(k)=0; for n =1: N

y(k)=y(k)+x(n).\* exp(-%i \*2\* %pi \*(k -1) \*(n -1)/N);

A= real (y); B= imag (y); end ; end ; mag = abs (y); x1= atan ( imag (y),real (y)); phase =x1 \*(180/ %pi ); disp ( 'The Resultant DFT Sequence is'); disp (y); disp ( 'The resultant real value is' ); disp (A); disp ( 'the resultnat Imaginary value is' ); disp (B); disp ( 'The Manitude response is' );

disp (mag); disp ( 'The phase response is' ); disp ( phase ); for n=1: N y(n)=0; for k =1: N

y(n)=y(n) +(1/ N)\*(y(k).\* exp (%i \*2\* %pi \*(k -1) \*(n-1) /N)); C= real (x); end ; end ; disp ( 'The resultant IDFT sequence is' ); disp (C); subplot (4 ,2 ,1); plot2d3 (x); title ( 'The input sequence is' ); subplot (4 ,2 ,2); plot2d3 (A); title ( 'The resultatn DFT sequence' ); subplot (4 ,2 ,3); plot2d3 (A); title ( 'Real value' ); subplot (4 ,2 ,4); plot2d3 (B); title ( 'Imaginary value' ); subplot (4 ,2 ,5); plot2d3 (mag); title ( 'Magnitude response' ); subplot (4 ,2 ,6); plot2d3 ( phase ); title ( 'Phase response' ); subplot (4 ,2 ,7);

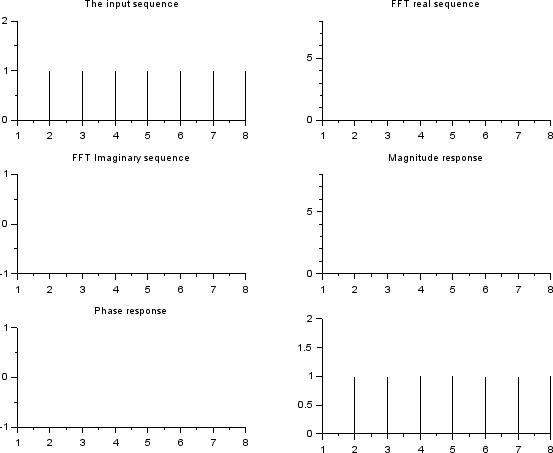
**b. Program for FFT of the signal**

clc ; clf ; clear all; x=[1 1 1 1 1 1 1 1]

//x= input ( 'Enter the input sequence' );

//N= input ( 'Enter the value of N' ); N=length(x) y= fft (x); A= real (y); B= imag (y); mag = abs (y); x1= atan ( imag (y),real (y)); phase =x1 \*(180/ %pi ); disp ( 'The resultant FFT sequence is' ); disp (y); disp ( 'The manidtude response is' ); disp (mag); disp ( 'The phase response is' ); disp ( phase ); z= ifft (y); disp ( 'The resultant IFFT sequence is' ); disp (z); subplot (3 ,2 ,1); plot2d3 (x); title ( 'The input sequence' ); subplot (3 ,2 ,2); plot2d3 (A); title ( 'FFT real sequence' ); subplot (3 ,2 ,3); plot2d3 (B); title ( 'FFT Imaginary sequence' ); subplot (3 ,2 ,4); plot2d3 (mag); title ( 'Magnitude response' ); subplot (3 ,2 ,5); plot2d3 ( phase ); title ( 'Phase response' ); subplot (3 ,2 ,6);

plot2d3 (x);



**Viva Questions:**

1. Why FFT is needed?

The direct evaluation DFT requires N2 complex multiplications and N2 –N complex additions. Thus, for large values of N direct evaluation of the DFT is difficult. By using FFT algorithm the number of complex computations can be reduced. So we use FFT.

1. **What is FFT? (University)**

The Fast Fourier Transform is an algorithm used to compute the DFT. It makes use of the symmetry and periodicity properties of twiddle factor to effectively reduce the DFT computation time. It is based on the fundamental principle of decomposing the computation of DFT of a sequence of length N into successively smaller DFTs.

1. **How many multiplications and additions are required to compute N point DFT using radix-2 FFT? (University)**

The number of multiplications and additions required to compute N point DFT using radix-2 FFT are N log2 N and N/2 log2 N respectively.

1. **What are the applications of FFT algorithm?**

The applications of FFT algorithm includes

* 1. Linear filtering
  2. Correlation
  3. Spectrum analysis

1. **What is the drawback in DTFT?**

The drawback in discrete time Fourier transform is that it is continuous function of 𝜔 and cannot be processed by digital systems.

**CONCLUSION:**

Thus, the spectrum of analog signal using FFT is computed using Scilab.

**Expt. No. 5 Design of IIR-Butterworth Filters and Chebyshev Filters**

**a. Design of IIR Filters (Butterworth Filter)**

**AIM**

To design and implement IIR (LPF, HPF BPF and BSF) Filters.

**REQUIREMENTS:**

PC with Scilab

**THEORY:**

The IIR filter can realize both the poles and zeroes of a system because it has a rational transfer function, described by polynomials in z in both the numerator and the denominator:

*M bk z* *k*

*H* (*z*) *k* 0

*N* *k*

 *ak Z*

*k* 1 (2)

The difference equation for such a system is described by the following:

*M*  *N* *y*(*n*)  *bk x*(*n*  *k*)   *ak y*(*n*  *k*)

*k* 0 *k* 1 (3)

M and N are order of the two polynomials bk and ak are the filter coefficients. These filter coefficients are generated using FDS (Filter Design software or Digital Filter design package).

IIR filters can be expanded as infinite impulse response filters. In designing IIR filters, cutoff frequencies of the filters should be mentioned. The order of the filter can be estimated using butter worth polynomial. That‟s why the filters are named as butter worth filters. Filter coefficients can be found and the response can be plotted.

**PROGRAM:**

**//Butterworth IIR LOW PASS Filter**

clc; clear all;

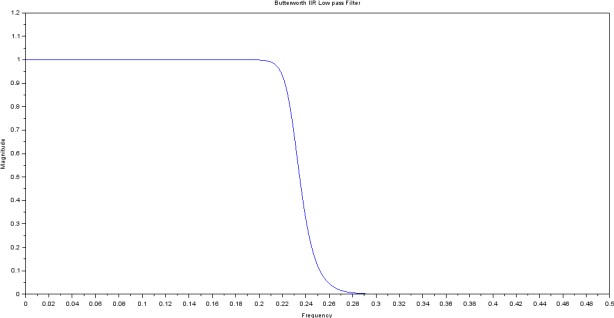
Fs=2000;

Fp=1000; Fsamp=9000; ap=2; as=90; wp=(2\*Fp)/Fsamp ws=(2\*Fs)/Fsamp;

N=(log(sqrt(((10^(0.1\*as))-1)/((10^(0.1\*ap))-1)))/log(ws/wp)); n=ceil(N); e=sqrt((10^(0.1\*ap))-1); l=sqrt((10^(0.1\*as))-1); omc=0.5\*((wp/e^(1/n))+(ws/l^(1/n)));

hz=iir(n,'lp','butt',omc,[0 0]); [mag phase]=frmag(hz,256); plot(phase,mag);

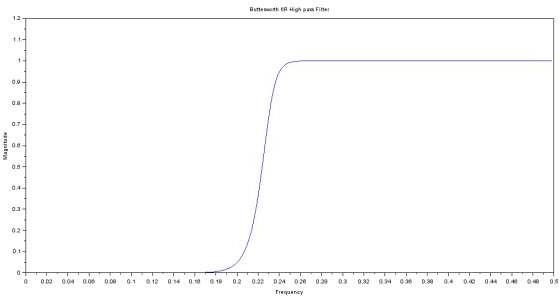
title("Butterworth IIR Low pass Filter") xlabel("Frequency"); ylabel("Magnitude");



### HIGH PASS Filter

Fs=2000;

Fp=1000; Fsamp=9000; ap=2; as=90; wp=(2\*Fp)/Fsamp ws=(2\*Fs)/Fsamp;

N=(log(sqrt(((10^(0.1\*as))-1)/((10^(0.1\*ap))-1)))/log(ws/wp)); n=ceil(N); e=sqrt((10^(0.1\*ap))-1); l=sqrt((10^(0.1\*as))-1); omc=0.5\*((wp/e^(1/n))+(ws/l^(1/n)));

hz=iir(n,'hp','butt',omc,[0 0]); [mag phase]=frmag(hz,256); plot(phase,mag);

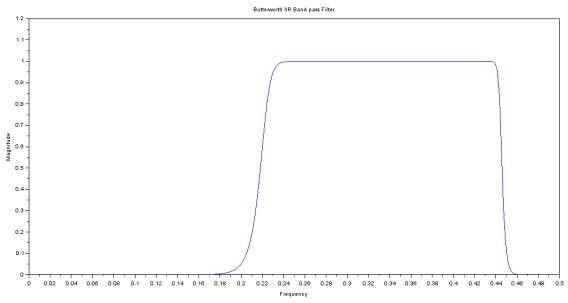
title("Butterworth IIR High pass Filter") xlabel("Frequency"); ylabel("Magnitude");

**Communication Engineering**

**Band Pass Filter**

Fs=2000;

Fp=1000; Fsamp=9000; ap=2; as=90; wp=(2\*Fp)/Fsamp ws=(2\*Fs)/Fsamp;

N=(log(sqrt(((10^(0.1\*as))-1)/((10^(0.1\*ap))-1)))/log(ws/wp)); n=ceil(N);

e=sqrt((10^(0.1\*ap))-1); l=sqrt((10^(0.1\*as))-1); omc=0.5\*((wp/e^(1/n))+(ws/l^(1/n)));

hz=iir(n,'bp','butt',[wp ws],[0 0]); [mag phase]=frmag(hz,256); plot(phase,mag);

title("Butterworth IIR Band pass Filter") xlabel("Frequency"); ylabel("Magnitude");

### Band Stop Filter

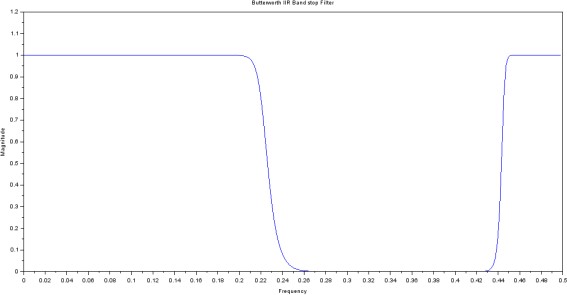
Fs=2000;

Fp=1000; Fsamp=9000; ap=2; as=90; wp=(2\*Fp)/Fsamp ws=(2\*Fs)/Fsamp;

N=(log(sqrt(((10^(0.1\*as))-1)/((10^(0.1\*ap))-1)))/log(ws/wp)); n=ceil(N); e=sqrt((10^(0.1\*ap))-1); l=sqrt((10^(0.1\*as))-1); omc=0.5\*((wp/e^(1/n))+(ws/l^(1/n)));

hz=iir(n,'sb','butt',[wp ws],[0 0]); [mag phase]=frmag(hz,256); plot(phase,mag);

title("Butterworth IIR Band stop Filter") xlabel("Frequency"); ylabel("Magnitude");



**VIVA QUESTIONS:**

1. What are the different types of filters based on frequency response?

Based on frequency response the filters can be classified as

1. Lowpass filter
2. Highpass filter
3. Band pass filter
4. Band reject filter

2. Mention the procedures for digitizing the transfer function of an analog filter? The two important procedures for digitizing the transfer function of an analog filter are • Impulse invariance method.

• Bilinear transformation method.

1. What is Warping Effect? (University)

In Bilinear Transformation,

Ω = 2/T tan ω/2

For Small values of ω, Ω = 2/T ω/2 = ω/T

For low frequency, the relationship between Ω and ω are linear. Therefore digital filtres have the same amplitude as the analog filter.

1. What are the advantages & disadvantages of bilinear transformation? Advantages:

* The bilinear transformation provides one-to-one mapping.
* Stable continuous systems can be mapped into realizable, stable digital systems.
* There is no aliasing.

Disadvantage:

* The mapping is highly non-linear producing frequency, compression at high frequencies.
* Neither the impulse response nor the phase response of the analog filter is preserved in a digital filter obtained by bilinear transformation.

**CONCLUSION:**

Thus, the IIR filter is designed and the magnitude and phase responses are verified successfully.

**B. Design of IIR filters (Chebyshev filter)**

**AIM**

To design and implement IIR (LPF/HPF/SPF) using Chebyshev filters.

**REQUIREMENTS :**

SCILAB and PC

**THEORY:**

The IIR filter can realize both the poles and zeroes of a system because it has a rational transfer function, described by polynomials in z in both the numerator and the denominator:

*M bk z* *k*

*H* (*z*) *k* 0

*N* *k*

 *ak Z*

*k* 1 (2)

The difference equation for such a system is described by the following:

*M*  *N* *y*(*n*)  *bk x*(*n*  *k*)   *ak y*(*n*  *k*)

*k* 0 *k* 1 (3)

M and N are order of the two polynomials bk and ak are the filter coefficients. These filter coefficients are generated using FDS (Filter Design software or Digital Filter design package).

IIR filters can be expanded as infinite impulse response filters. In designing IIR filters, cutoff frequencies of the filters should be mentioned. The order of the filter can be estimated using butter worth polynomial. That‟s why the filters are named as butter worth filters. Filter coefficients can be found and the response can be plotted.

**ALGORITHM:**

1. Get the band Pass and Band stop frequencies.
2. Get the band Pass and Band stop ripples.
3. Read the sampling frequencies.
4. Calculate the order of the filter 5. Draw magnitude and phase.

**PROGRAM:**

**//Chebyshev Type I IIR Low Pass Filter** clc; clear all;

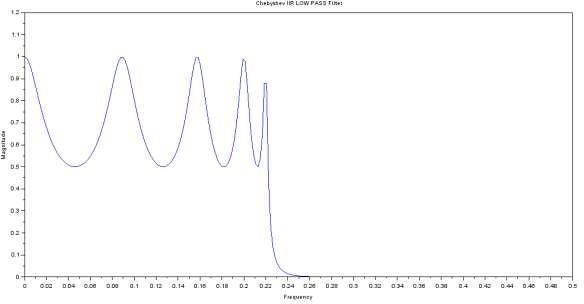
Fs=2000;

Fp=1000; Fsamp=9000; ap=2;

as=90; wp=(2\*Fp)/Fsamp ws=(2\*Fs)/Fsamp;

N=(acosh(sqrt(((10^(0.1\*as))-1)/((10^(0.1\*ap))-1)))/acosh(ws/wp)); n=ceil(N); hz=iir(n,'lp','cheb1',wp,[0.5 1]); [mag phase]=frmag(hz,256); plot(phase,mag);

title("Chebyshev IIR LOW PASS Filter") xlabel("Frequency"); ylabel("Magnitude");



**//Chebyshev Type II IIR High Pass Filter** clc; clear all;

Fs=2000;

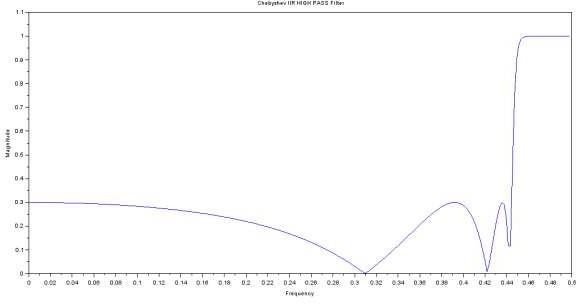
Fp=1000; Fsamp=9000; ap=2; as=60;

wp=(2\*Fp)/Fsamp ws=(2\*Fs)/Fsamp;

N=(acosh(sqrt(((10^(0.1\*as))-1)/((10^(0.1\*ap))-1)))/acosh(ws/wp)); n=ceil(N); hz=iir(n,'hp','cheb2',ws,[0 0.3]); [mag phase]=frmag(hz,256); plot(phase,mag);

title("Chebyshev IIR HIGH PASS Filter")

xlabel("Frequency"); ylabel("Magnitude");



**//Chebyshev Type II IIR Band Pass Filter** clc; clear all;

Fs=2000;

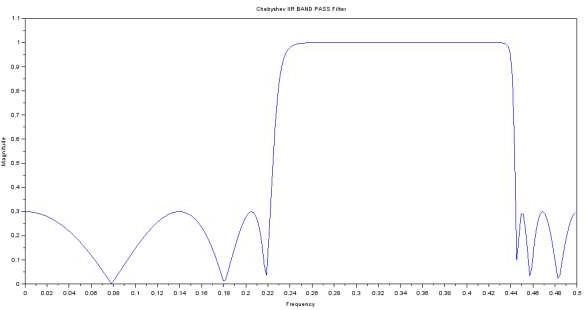
Fp=1000; Fsamp=9000; ap=2; as=60; wp=(2\*Fp)/Fsamp ws=(2\*Fs)/Fsamp;

N=(acosh(sqrt(((10^(0.1\*as))-1)/((10^(0.1\*ap))-1)))/acosh(ws/wp)); n=ceil(N);

hz=iir(n,'bp','cheb2',[wp ws],[0 0.3]);

[mag phase]=frmag(hz,256); plot(phase,mag);

title("Chebyshev IIR BAND PASS Filter") xlabel("Frequency"); ylabel("Magnitude");



**//Chebyshev Type I IIR BAND STOP Filter** clc;

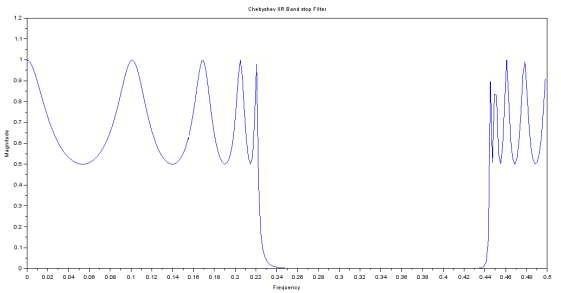
clear all;

Fs=2000;

Fp=1000; Fsamp=9000; ap=2; as=90; wp=(2\*Fp)/Fsamp ws=(2\*Fs)/Fsamp;

N=(acosh(sqrt(((10^(0.1\*as))-1)/((10^(0.1\*ap))-1)))/acosh(ws/wp)); n=ceil(N); hz=iir(n,'sb','cheb1',[wp ws],[0.5 1]); [mag phase]=frmag(hz,256); plot(phase,mag);

title("Chebyshev IIR Band stop Filter") xlabel("Frequency"); ylabel("Magnitude");



**CONCLUSION:**

Thus the IIR ( Chebyshev ) filters are designed and frequency response has been plotted.

**Expt. No. 6 Design of FIR Filters using Windows**

**AIM:**

To design a FIR filter (lowpass/ High pass/ bandpass/ Band stop) using window techniques.

**REQUIREMENTS :**

SCILAB, Personal computer.

**THEORY:**

A discrete time filter produces a discrete-time output sequence y(n0 for the discrete-time input sequence x(n). A filter may be required to have a given frequency response, or a specific response to an impulse, step, or ramp, or simulate an analog system. Digital filters are classified as Finite Impulse Response (FIR) filters and Infinite Impulse Response(IIR) filters. In the FIR system, the impulse response sequence is of finite duration, i.e. it has a finite number of non-zero terms. FIR filters are designed using following techniques:

1. Fourier series method
2. Frequency sampling method
3. Window techniques

**ALGORITHM:**

1. Get the passband and stopband ripples.
2. Get the passband and stopband edge frequencies.
3. Read the sampling frequency.
4. Calculate the order of the filter.
5. Find the window coefficients of the specified window.
6. Draw the magnitude and phase responses.

**// Program to design FIR Low Pass Filter**

clc ; close ;

M =11// input ( 'Enter the odd filter length =' ); // Filter length

Wc = %pi/2//input ( ' Enter the digital cutoff frequency=' ); // Digital Cut off Frequency Tuo = (M -1)/2 // Center value for n = 1:M

if (n == Tuo +1)

hd(n) = Wc/ %pi ;

else

hd(n) = sin(Wc \*((n -1) -Tuo)) /(((n -1) -Tuo )\*%pi);

end

end

// Rectangular Window for n = 1:M

W(n) = 1; end

// Windowing filter Coefficients h = hd.\*W;

disp (h, ' Filter Coefficients are ' ) [hzm ,fr ]= frmag (h ,256) ; hzm\_dB = 20\* log10 (hzm)./ max ( hzm );

subplot (2 ,1 ,1) plot (2\*fr , hzm ) xlabel ( ' Normalized Digital Frequecny W' ); ylabel ( 'Magnitude ' ); title ( ' Frequency Response of FIR LPF using Rectangular window' )

xgrid (1) subplot (2,1,2) plot (2\*fr , hzm\_dB ) xlabel ( 'Normalized Digital Frequecy W' ); ylabel ( 'Magnitude in dB ' ); title ( ' Frequency Response of FIR LPF using Rectangular window' ) xgrid (1)

**// Program to Design FIR High Pass Filter**

clear ; clc ; close ;

M =21// input ( 'Enter the odd filter length =' ); // Filter length

Wc =%pi/2// input ( 'Enter the digital cut off frequency=' ); // Digital Cut off frequency

Tuo = (M -1) /2 // Center value

for n = 1:M

if (n == Tuo+1) hd(n) = 1-Wc/ %pi ;

else hd(n) = ( sin ( %pi \*((n -1) -Tuo)) -sin(Wc \*((n -1) -Tuo ))) /(((n -1) -Tuo )\* %pi );

end end // Rectangular window for n = 1:M W(n) = 1; end

//Windowing Filter coefficients h = hd .\*W; disp (h, 'Filter Coefficients are' ) [hzm ,fr ]= frmag (h ,256) ;

hzm\_dB = 20\* log10 (hzm)./ max (hzm);

subplot (2 ,1 ,1) plot (2\*fr , hzm ) xlabel ( 'Normalized Digital Frequency W' ); ylabel ( 'Magnitude ' ); title ( 'Frequency response of FIR HPF using rectangular window' )

xgrid (1) subplot (2 ,1 ,2) plot (2\*fr , hzm\_dB ) xlabel ( 'Normalized digital frequency W' ); ylabel ( 'Magnitude in dB' ); title ( 'Frequency response of FIR HPF using rectangular window')

xgrid (1)

**//Program to Design FIR Band Pass Filter**

clear; clc ; close ;

M =11// input ( 'Enter the odd filter length=' ); //Filter length

// Digital cut of frequency [Lower Cutoff, Upper cut off]

Wc =[%pi/4 3\*%pi/4]// input ( 'Enter the digital cut off frequency=' );

Wc2 = Wc(2)

Wc1 = Wc(1)

Tuo = (M -1) /2 // Center value hd = zeros (1,M); W = zeros (1,M); for n = 1:11 if (n == Tuo +1) hd(n) = (Wc2 - Wc1 )/ %pi ; else n hd(n) = ( sin ( Wc2 \*((n -1) -Tuo)) -sin( Wc1 \*((n -1) -Tuo ))) /(((n -1) -Tuo )\* %pi ); end if(abs(hd(n)) <(0.00001) ) hd(n) =0; end end hd;

//Rectangular window for n = 1:M W(n) = 1; end

//Windowing Filter Coefficients h = hd .\*W;

disp (h, ' Filter Coefficients are' ) [hzm ,fr ]= frmag (h ,256) ; hzm\_dB = 20\* log10 (hzm)./ max ( hzm );

subplot (2 ,1 ,1) plot (2\*fr , hzm ) xlabel ( 'Normalized digital frequency' ); ylabel ( 'Magni tude ' ); title ( 'Frequency response of FIR BPF using rectangular window')

xgrid (1) subplot (2 ,1 ,2) plot (2\*fr , hzm\_dB ) xlabel ( 'Normalized Digital Frequency W'); ylabel ( 'Magnitude in dB'); title ( 'Frequency response of FIR BPF using rectangular window'); xgrid (1)

**// Program to Design FIR Band Reject Filter**

clear ; clc ; close ;

M = 11//input ( 'Enter the odd filter length=' ); // Filter length

//Digital Cut off frequency [ Lower cut off, Upper cut off]

Wc =[%pi/3 2\*%pi/3]// input ( 'Enter the digital cut off frequecy=');

Wc2 = Wc(2)

Wc1 = Wc(1)

Tuo = (M -1) /2 //Center value hd = zeros (1,M); W = zeros (1,M); for n = 1:M if (n == Tuo +1)

hd(n) = 1 -(( Wc2 -Wc1)/ %pi ); else hd(n)=( sin ( %pi \*((n -1) -Tuo))-sin ( Wc2 \*((n -1) -Tuo))+sin ( Wc1 \*((n -1) -Tuo ))) /(((n -1) -Tuo )\* %pi );

end if(abs(hd(n)) <(0.00001) ) hd(n) =0; end

end

//Rectangular Window

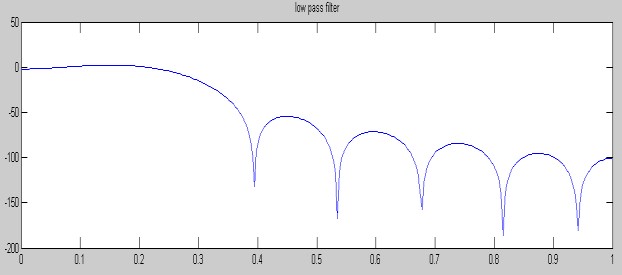
for n = 1:M W(n) = 1; end

//Windowing filter coefficients h = hd .\*W; disp (h, 'Filter coeffiecitents are' ) [hzm ,fr ]= frmag (h ,256) ; hzm\_dB = 20\* log10 (hzm)./ max ( hzm);

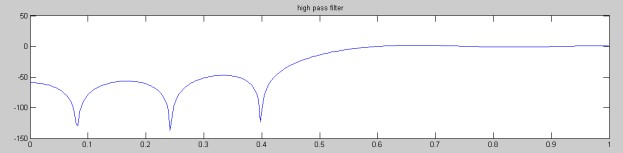
subplot (2 ,1 ,1) plot (2\*fr , hzm ) xlabel ( 'Normalized digital frequency W'); ylabel ( 'Magnitude ' ); title ( 'Frequency response of FIR BSF using rectangular window'); xgrid (1) subplot (2 ,1 ,2) plot (2\*fr , hzm\_dB ) xlabel ( 'Normalized digital frequency W'); ylabel ( 'Magnitude in dB'); title ( ' Frequency response of FIR BSF using rectangular window') xgrid (1)

**OUTPUT:**

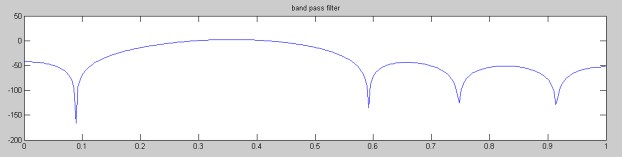
### LOWPASS USING RECTANGULAR



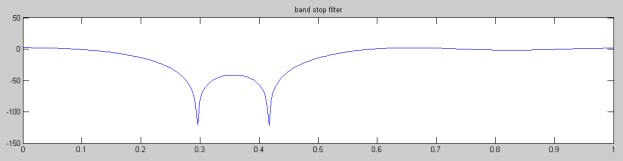
**HIGH PASS FILTER USING BLACKMANN WINDOW**



**BAND BASS FILTER USING HAMMING WINDOW**



### BANDSTOP USING HANNING WINDOW



**REVIEW QUESTIONS:**

1. **What are FIR filters?**

The filter designed by selecting finite number of samples of impulse response ℎ(𝑛) obtained from inverse Fourier transform of desired frequency response 𝐻(𝜔) are called FIR filters

1. **Write the steps involved in FIR filter design?** 
   1. Choose the desired frequency response Hd(w)
   2. Take the inverse Fourier transform and obtain Hd(n)
   3. Convert the infinite duration sequence Hd(n) to h(n)
   4. Take Z transform of h(n) to get H(Z)

1. **What are the disadvantages of FIR Filter? (University)**

The duration of impulse response should be large to realize sharp cut off filters. The non-integral delay can lead to problems in some signal processing applications.

1. **List the well-known design technique for linear phase FIR filter design?** 1. Fourier series method and window method 2. Frequency sampling method.

3. Optimal filter design method.

**CONCLUSION:**

Thus, the FIR filter is designed using window technique and the magnitude and phase responses are verified successfully.

**Expt. No.7 Computation of Up Sampling and Down Sampling**

**AIM:**

To design an Upsampling and dowunsampling of signals using matlab.

**ALGORITHM:**

* Assign the variable for length of the sinusoidal sequence and sampling factor.
* Generate a sinusoidal waveform
* Increase the sampling rate by interpolating zeros according to sampling factor.
* Plot the graph.

**PROGRAM: //UPSAMPLING:**

clc; clear all; x=[1 2 3 4] l=5; k=length(x); y=zeros(1,k\*l); t=1:l:k\*l; y(t)=x; subplot(2,1,1) plot2d3(x);

title('input');

xlabel('samples') ylabel('amp') subplot(2,1,2) title('output'); xlabel('samples') ylabel('amp') plot2d3(y)

**//DOWNSAMPLING**

clc; clear all; x=[1 2 3 4 6 7 8 9 6]

l=2; k=length(x); t=1:l:k; y=x(t); subplot(2,1,1) plot2d3(x);

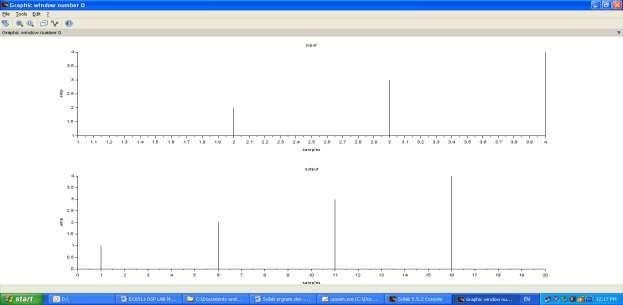
title('input');

xlabel('samples')

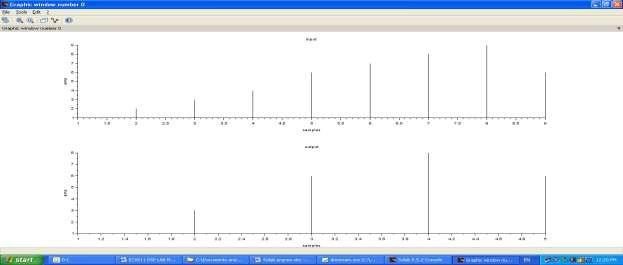
ylabel('amp') subplot(2,1,2) title('output'); xlabel('samples') ylabel('amp') plot2d3(y)

**OUTPUT:**

**UPSAMPLE:**



**DOWNSAMPLE:**



**CONCLUSION:**

Thus, the up-sampling and down-sampling of signals were performed using SCILAB.

**DSP PROCESSOR**

**TMS 320C6713/C5416**

**EXPERIMENTS**

## INTRODUCTION TO DSP PROCESSORS

A signal can be defined as a function that conveys information, generally about the state or behavior of a physical system. There are two basic types of signals viz Analog (continuous time signals which are defined along a continuum of times) and Digital (discrete-time).

Remarkably, under reasonable constraints, a continuous time signal can be adequately represented by samples, obtaining discrete time signals. Thus digital signal processing is an ideal choice for anyone who needs the performance advantage of digital manipulation along with today‟s analog reality.

Hence a processor which is designed to perform the special operations(digital manipulations) on the digital signal within very less time can be called as a Digital signal processor. The difference between a DSP processor, conventional microprocessor and a microcontroller are listed below.

**Microprocessor** or General Purpose Processor such as Intel xx86 or Motorola 680xx family Contains - only CPU

-No RAM

-No ROM

-No I/O ports

-No Timer

**Microcontroller** such as 8051 family Contains - CPU

* RAM
* ROM

-I/O ports

* Timer &
* Interrupt circuitry

Some Micro Controllers also contain A/D, D/A and Flash Memory

**DSP Processors** such as Texas instruments and Analog Devices

Contains - CPU

* RAM -ROM
* I/O ports
* Timer

Optimized for – fast arithmetic

* Extended precision - Dual operand fetch
* Zero overhead loop
* Circular buffering

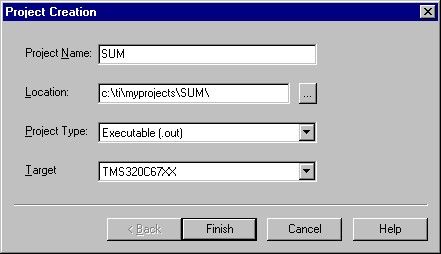
The basic features of a DSP Processor are

|  |  |
| --- | --- |
| **Feature** | **Use** |
| Fast-Multiply accumulate | Most DSP algorithms, including filtering, transforms, etc. are  multiplication- intensive |
| Multiple – access memory  architecture | Many data-intensive DSP operations require reading a program instruction and multiple data items during each instruction cycle for best performance |
| Specialized addressing modes | Efficient handling of data arrays and first-in, first-out buffers in memory |
| Specialized program control | Efficient control of loops for many iterative DSP algorithms. Fast interrupt handling for frequent I/O operations. |
| On-chip peripherals and I/O interfaces | On-chip peripherals like A/D converters allow for small low cost system designs. Similarly I/O interfaces tailored for common peripherals allow clean interfaces to off-chip I/O devices. |

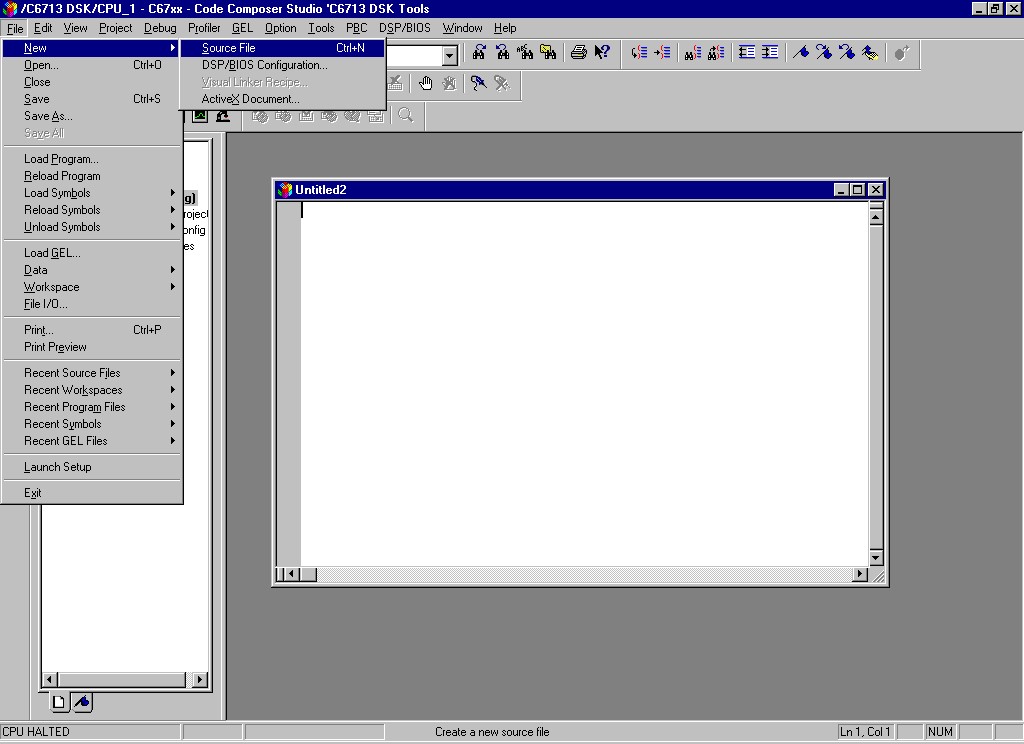
## Procedure to work on Code Composer Studio

1. **To create a New Project**

*Project*  *New* ***(SUM.pjt)***

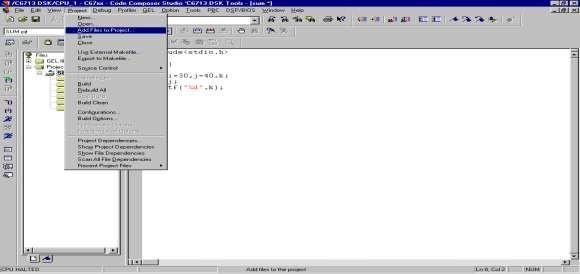


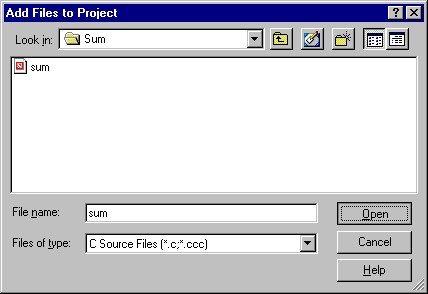
1. **To Create a Source file** *File*  *New*



*Type the code (Save & give a name to file, Eg:* ***sum.c****).*

1. **To Add Source files to Project** *Project*  *Add files to Project*  *sum.c*





1. **To Add rts6700.lib file & hello.cmd:**

***Project***  ***Add files to Project*** ***rts6700.lib***

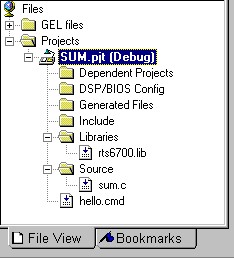
***Path: c:\CCStudio\c6000\cgtools\lib\rts6700.lib***

***Note: Select Object & Library in(\*.o,\*.l) in Type of files***

***Project***  ***Add files to Project*** ***hello.cmd***

***Path: c:\ti\tutorial\dsk6713\hello1\hello.cmd***

***Note: Select Linker Command file(\*.cmd) in Type of files***



1. **To Compile:**

*Project*  *Compile File*

1. **To build or Link:**

*Project*  *build,*

*Which will create the final executable* ***(.out)*** *file.(Eg. sum.out).*

1. **Procedure to Load and Run program: *Load program to DSK:***

*File*  *Load program*  *sum. out*

1. **To execute project:**

*Debug*  *Run.*

### Expt. No. 8 Study Of Addressing Modes Of DSP Processor Using CCS Debugging Software

**AIM:**

To study about direct, indirect and immediate addressing modes in TMS320C50 debugger.

**EQUIPMENTS REQUIRED:**

1.System with TMS 320C50 debugger software

2.TMS 320C50 Kit.

**ALGORITHM:**

**IMMEDIATE ADDRESSING MODE:**

1. Initialize data pointer with 100H data.
2. Load the accumulator with first data.
3. Add the second data with accumulator content.
4. Store the accumulator content in specified address location.

**DIRECT ADDRESSING MODE:**

1. Initialize data pointer with 100H data.
2. Load the accumulator with first data, whose address is specified in the instruction.
3. Add the accumulator content with second data, whose address is specified in the instruction.
4. Store the accumulator content in specified address location.

**IN-DIRECT ADDRESSING MODE:**

1. Load the auxiliary register with address location of first data.
2. The auxiliary register (AR0) is modified indirectly as # symbol.
3. Load the second data into accumulator and perform addition operation.
4. Store the result.

**PROGRAM:**

**//Program for immediate addressing mode**

.mmregs

.text

START:

LDP #100H EC 56-Digital Signal Processing Lab LACC #1241H

ADD #1200H

SACL 2H

H: B H

**//Program for direct addressing mode**

.mmregs

.text

START:

LDP #100H

LACC 0H

ADD 1H

SACL 2H

H: B H

**//Program for adding two numbers with indirect addressing mode.**

.mmregs

.text

START:

LAR AR0,#8000H

MAR \*,AR0

LACC \*+,0 ;WITH ZERO SHIFT

ADD \*+

SACL \*+

H: B H

**OUTPUT:**

**CONCLUSION:**

Thus the Program which illustrates various addressing modes were executed with TMS 320C50 debugger software.

### Expt. No. 9 Generation of Various Waveform using TMS 320C6713

**AIM:**

To generation of different waveforms for Sine, Cosine, Exponential and Random noise signals using DSP processor C6713.

**EQUIPMENTS:**

CCS studio,PC,C6713 Kit

**PROCEDURE:**

1. switch on the DSP board
2. Open the Code Composer Studio
3. Create a new project from the pull down menu „Project‟. From this select „New‟ and save the project in :c\ti\myprojects.
4. Create a new source file from the pull down menu „File‟. From this select „New‟ and save
5. Add the source file to the project by pull down menu „Project‟ and select „Add files to project‟.
6. Add library files to the project by pull down menu „Project‟ and select „Add files to project‟ in the path :c\ti\c6000\cgtools\lib\rts6700.
7. Add linker command file to the project by pull down menu „Project‟ and select „Add files to project‟ in the path :c\ti\tutorialdsk5416\hello\hello.cmd.
8. Now Compile the file & Rebuild All the file by using „Project‟ menu by selecting

„Compile‟ & „Rebuild all‟.

1. Load the program to the processor by selecting the menu „File‟ and „load program‟.

Now execute the program by selecting „Debug‟ and „Run‟.

10.10.

**ALGORITHM:**

1. Get the frequency Value
2. Generating the sine wave using corresponding formula
3. Plot the graph.

**PROGRAM:**

**a) SINE WAVE GENERATION**

#include <stdio.h> #include <math.h>

int t; float pi=3.14; float x[256]; void main() { f=8000; for(t=0; t<=256; t++)

{ x[t]=sin(2\*pi\*f\*t); printf("%f \n",x[t]);

}

}

**b) COSINE WAVE GENERATION**

#include <stdio.h> #include <math.h>

int t; float pi=3.14; float x[256]; void main() { f=8000; for(t=0; t<=256; t++)

{ x[t]=cos(2\*pi\*f\*t); printf("%f \n",x[t]);

}

}

**C) EXPONENTIAL WAVEFORM:**

#include<stdio.h> #include<math.h>

int i; float a=0.2; float ex[200]; void main()

{ for(i=0;i<=150;i++)

{ ex[i]= exp(a\*i); printf("The exponential wave :%f\n ",ex[i]); }

}

**D) RANDOM NOISE WAVEFORM:**

#include<stdio.h> #include<math.h>

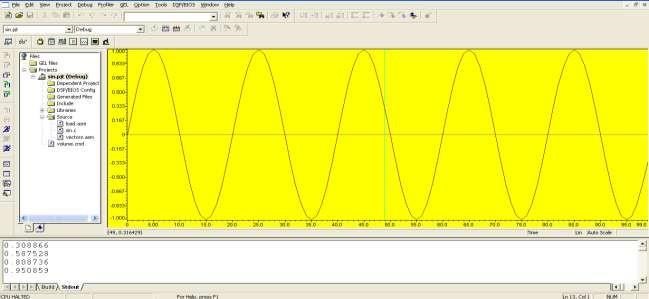
int i;

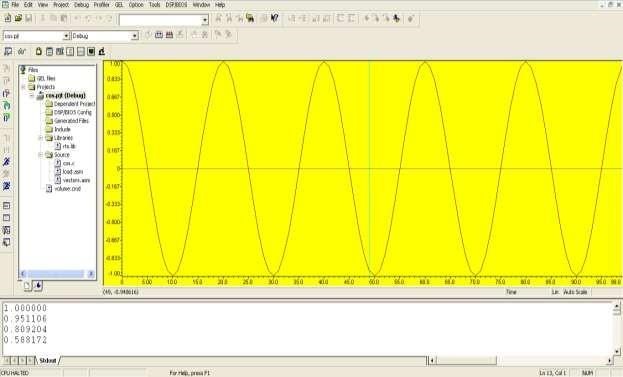
float a=0.2; float r[256]; void main()

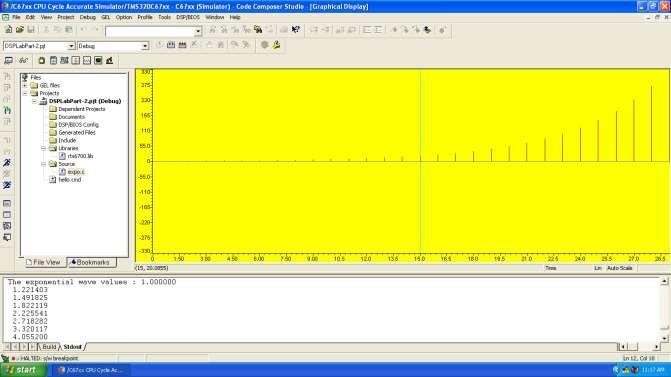
{ for(i=0;i<=256;i++)

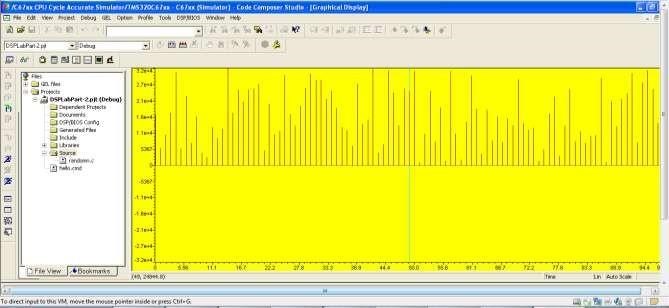
{ r[i]= 0.5\* sin(2\*3.14\*1000\*i)+0.02\*cos(2\*3.14\*5000\*i)+exp(a\*i); printf("The exponential wave :%f\n ",r[i]); }

}









**CONCLUSION:**

Thus the generation of different waveform were implemented by using TMS320C6713.

### Expt. No. 10 Implementation of IIR Filter using TMS3206713 Kit

**AIM**

To verify IIR filter design parameters using C6713 Kit.

**EQUIPMENTS:**

Operating System – Windows XP

Constructor **-** Simulator and TMS 6713 kit

Software - CCStudio v.3.1

**PROCEDURE:**

1. switch on the DSP board
2. Open the Code Composer Studio
3. Create a new project from the pull down menu „Project‟. From this select „New‟ and save the project in :c\ti\myprojects.
4. Create a new source file from the pull down menu „File‟. From this select „New‟ and save
5. Add the source file to the project by pull down menu „Project‟ and select „Add files to project‟.
6. Add library files to the project by pull down menu „Project‟ and select „Add files to project‟ in the path :c\ti\c6000\cgtools\lib\rts6700.
7. Add linker command fileto the project by pull down menu „Project‟ and select „Add files to project‟ in the path :c\ti\tutorial\dsk67xx\hello\hello.cmd.
8. Now Compile the file & Rebuild All the file by using „Project‟ menu by selecting „Compile‟

& „Rebuild all‟.

1. Load the program to the processor by selecting the menu „File‟ and „load program‟. Now execute the program by selecting „Debug‟ and „Run‟.

**ALGORITHM:**

1. Get the order of the filter
2. Get the cut off frequency of the filter.
3. Calculate transfer function of the system
4. Plot the graph

**PROGRAM:**

//iirfilters

#include<stdio.h> #include<math.h> inti,w,wc,c,N; float H[100]; float mul(float, int); void main()

{

printf("\n enter order of filter ");

scanf("%d",&N);

printf("\n enter the cutoff freq ");

scanf("%d",&wc);

printf("\n enter the choice for IIR filter 1. LPF 2.HPF "); scanf("%d",&c); switch(c)

{ case 1:

for(w=0;w<100;w++)

{

H[w]=1/sqrt(1+mul((w/(float)wc),2\*N)); printf("H[%d]=%f\n",w,H[w]);

} break; case 2:

for(w=0;w<=100;w++)

{

H[w]=1/sqrt(1+mul((float)wc/w,2\*N)); printf("H[%d]=%f\n",w,H[w]);

} break;

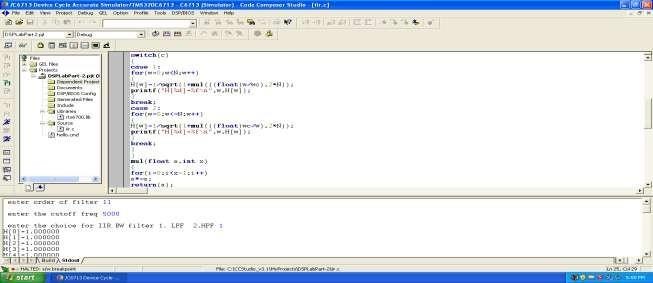
} }

float mul(float a,int x)

{

for(i=0;i<x-1;i++) a\*=a; return(a);

}



**CONCLUSION:**

Thus the IIR filters were designed by using TMS320C6713.

### Expt. No. 11 Implementation of FIR Filters using TMS320C6713

**AIM**

To verify FIR filter design parameters using C6713 Kit.

**EQUIPMENTS:**

Operating System – Windows XP

Constructor **-** Simulator and TMS kit

Software - CCStudio 3

**THEORY:**

A Finite Impulse Response (FIR) filter is a discrete linear time-invariant system whose output is based on the weighted summation of a finite number of past inputs. An FIR transversal filter structure can be obtained directly from the equation for discrete-time convolution.

*N* 1

*y*(*n*)   *x*(*k*)*h*(*n*  *k*) 0  *n*  *N* 1**(1)**

*k* 0

In this equation, x(k) and y(n) represent the input to and output from the filter at time n. h(n-k) is the transversal filter coefficients at time n. These coefficients are generated by using FDS (Filter Design Software or Digital filter design package).

FIR – filter is a finite impulse response filter. Order of the filter should be specified. Infinite response is truncated to get finite impulse response. placing a window of finite length does this. Types of windows available are Rectangular, Barlett, Hamming, Hanning, Blackmann window etc. This FIR filter is an all zero filter.

**PROCEDURE:**

1. switch on the DSP board
2. Open the Code Composer Studio
3. Create a new project from the pull down menu „Project‟. From this select „New‟ and save the project in :c\ti\myprojects.
4. Create a new source file from the pull down menu „File‟. From this select „New‟ and save
5. Add the source file to the project by pull down menu „Project‟ and select „Add files to project‟.
6. Add library files to the project by pull down menu „Project‟ and select „Add files to project‟ in the path :c\ti\c6000\cgtools\lib\rts6700.
7. Add linker command fileto the project by pull down menu „Project‟ and select „Add files to project‟ in the path :c\ti\tutorialdsk6711\hello\hello.cmd.
8. Now Compile the file & Rebuild All the file by using „Project‟ menu by selecting „Compile‟

& „Rebuild all‟.

1. Load the program to the processor by selecting the menu „File‟ and „load program‟. Now execute the program by selecting „Debug‟ and „Run‟.

**PROGRAM:**

**(a). For Rectangular and Triangular windows:**

#include<stdio.h>

#include<math.h> #define pi 3.1415 int n,N,c; float wr[64],wt[64]; void main()

{

printf("\n enter no. of samples,N= :"); scanf("%d",&N);

printf("\n enter choice of window function\n 1.rect \n 2. triang \n c= :"); scanf("%d",&c);

printf("\n elements of window function are:"); switch(c) { case 1:

for(n=0;n<=N-1;n++)

{ wr[n]=1;

printf(" \n wr[%d]=%f",n,wr[n]);

} break; case 2:

for(n=0;n<=N-1;n++)

{ wt[n]=1-(2\*(float)n/(N-1)); printf("\n wt[%d]=%f",n,wt[n]);

}

break;

}

}

#### (b). FIR Filter Design using Hamming Window

#include<stdio.h>

#include<math.h> #define pi 3.1415 int N,n;

float H[50],h[50],WH[50]; main()

{

printf("enter the length of the filter coefficients for h(n) and hamming window:"); scanf("%d",&N);

printf("\n calculated filter coefficients are:\n");

for(n=-(N-1)/2;n<=(N-1)/2;n++)

{ h[n]=0.0; if(n==0) h[n]=(float)1/3;

else h[n]=sin(n\*180/3)/(n\*pi); printf("h[%d]=%f\n",n,h[n]);

}

printf("\n calculated hamming window coefficient are :\n");

for(n=-(N-1)/2;n<=(N-1)/2;n++)

{

WH[n]=0.0;

WH[n]=0.54+0.46\*cos(2\*pi\*n/(N-1));

printf("WH[%d]=%f\n",n,WH[n]);

}

printf("\n the final FIR filter coefficient after window:\n");

for(n=-(N-1)/2;n<=(N-1)/2;n++)

{

H[n]=0.0; H[n]=h[n]\*WH[n]; printf("H[%d]=%f\n",n,H[n]);

}

}

#### (c). FIR Filter Design using Hannming Window

#include<stdio.h>

#include<math.h> #define pi 3.1415

int N,n; float H[50],h[50],wh[50]; main() {

printf("enter the length of filter coefficient for h[n] and a hamming window:"); scanf("%d",&N);

printf("\n calculted filter coefficient are:\n");

for(n=-(N-1)/2;n<=(N-1)/2;n++)

{ h[n]=0.0; if(n==0) h[n]=(float)1/3;

else

h[n]=sin(n\*180/3)/(n\*pi);

printf("h[%d]=%f\n",n,h[n]);

}

printf("\n calculated hanning window coeffcient are:\n");

for(n=-(N-1)/2;n<=(N-1)/2;n++)

{ wh[n]=0.0; wh[n]=0.5+0.5\*cos(2\*pi\*n/(N-1)); printf("wh[%d]=%f\n",n,wh[n]);

}

printf("\n final filter coefficient after windowing:\n");

for(n=-(N-1)/2;n<=(N-1)/2;n++)

{

H[n]=0.0; H[n]=h[n]\*wh[n];

printf("H[%d]=%f\n",n,H[n]);

}

}

#### (d). FIR Filter Design using Blackman Window

#include<stdio.h>

#include<math.h> #define pi 3.1415

int N,n;

float H[50],h[50],wb[50]; main() {

printf("enter the length of Fir filter coefficient h[n] and a black man window:\n"); scanf("%d",&N);

printf("\n calculated filter coefficient are:\n");

for(n=-(N-1)/2;n<=(N-1)/2;n++)

{ h[n]=0.0; if(n==0) h[n]=(float)1/3;

else

h[n]=sin(n\*180/3)/(n\*pi); printf("h[%d]=%f\n",n,h[n]);

}

printf("calculated blackman window coefficient are:\n"); for(n=-(N-1)/2;n<=(N-1)/2;n++)

{ wb[n]=0.0;

wb[n]=0.42+0.5\*cos(2\*pi\*n/(N-1))+0.08\*cos(4\*pi\*n/(N-1));

printf("wb[%d]=%f\n",n,wb[n]); }

printf("\nThe final fir filter coefficient after windowing:\n"); for(n=-(N-1)/2;n<=(N-1)/2;n++)

{

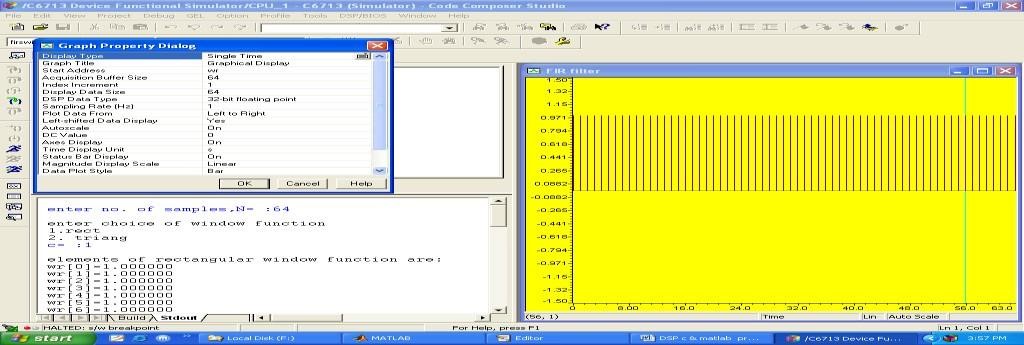
H[n]=0.0; H[n]=h[n]\*wb[n];

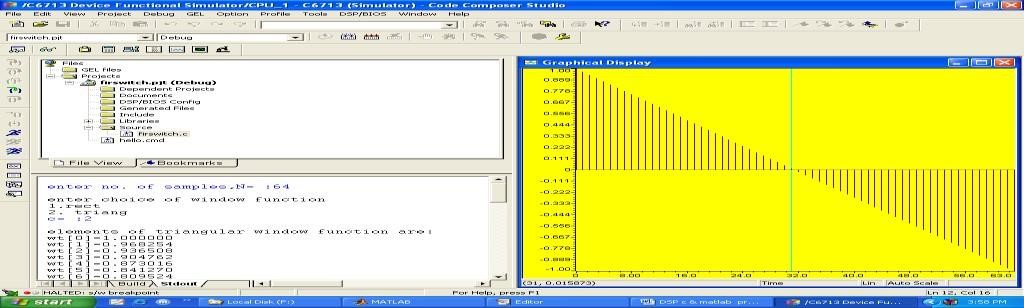
printf("H[%d]=%f\n",n,H[n]);

}

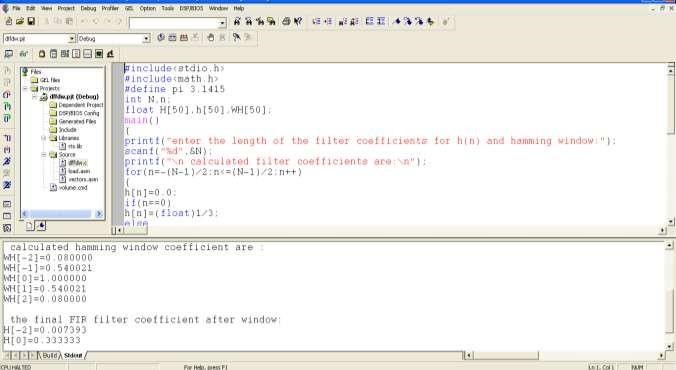
}

#### Output: (a) Rectangular and Triangular windows

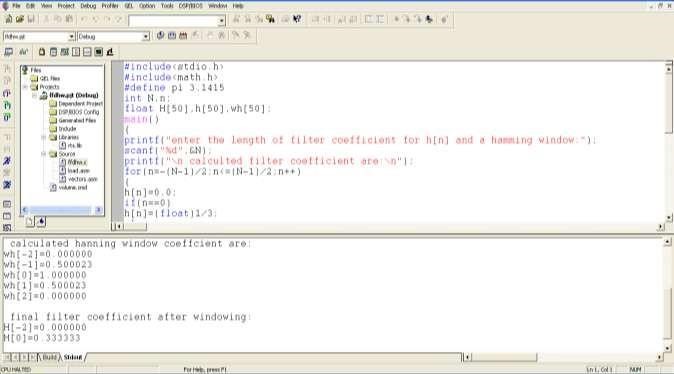




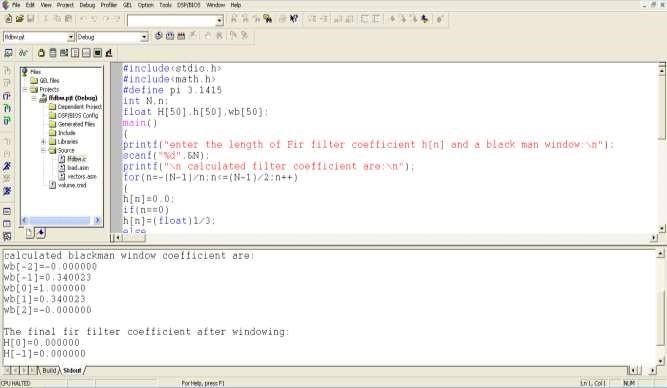
**(b). FIR Filter Design using Hamming Window**



### (c). FIR Filter Design using Hanning Window



#### (d). FIR Filter Design using Blackman Window



**CONCLUSION:**

Thus the FIR filters were designed by using TMS320C6713.

#### Expt. No. 12 Performing Up-Sampling and Down-Sampling using TMS3206713 Kit

**AIM**

To verify the performance of Up-Sampling and Down-Sampling process using C6713 Kit.

**EQUIPMENTS:**

Operating System – Windows XP

Constructor **-** Simulator and TMS 6713 kit

Software - CCStudio v.3.1

**PROCEDURE:**

1. switch on the DSP board
2. Open the Code Composer Studio
3. Create a new project from the pull down menu „Project‟. From this select „New‟ and save the project in :c\ti\myprojects.
4. Create a new source file from the pull down menu „File‟. From this select „New‟ and save
5. Add the source file to the project by pull down menu „Project‟ and select „Add files to project‟.
6. Add library files to the project by pull down menu „Project‟ and select „Add files to project‟ in the path :c\ti\c6000\cgtools\lib\rts6700.
7. Add linker command fileto the project by pull down menu „Project‟ and select „Add files to project‟ in the path :c\ti\tutorial\dsk67xx\hello\hello.cmd.
8. Now Compile the file & Rebuild All the file by using „Project‟ menu by selecting „Compile‟

& „Rebuild all‟.

1. Load the program to the processor by selecting the menu „File‟ and „load program‟. Now execute the program by selecting „Debug‟ and „Run‟.

**ALGORITHM:**

1. Generate a signal with specific frequency
2. Assign the up-sampling rate and down-sampling rate
3. Calculate the sampled values for increased and decreased sampled rates
4. Plot the graph

**Up-Sampling:**

#include<stdio.h>

#include<math.h>

float out1[200], out2[200]; float amp=5,freq=50; float fs=5000, t=0.0;

int i,j,k,L; void main()

{

L=100; k=2; for(i=0;i<=L;i++) { out1[i]=amp\*sin(2\*3.14\*freq\*t);

t=t+1/fs; } for(i=0;i<=(k\*L);i++) { out2[i]=0; } i=0; for(j=0;j<=(k\*L);j=k+j)

{ out2[j]=out1[i]; i++;

}

for(j=0;j<=(k\*L);j++) { printf("\n %f",out2[j]);

}

}

**Down-Sampling:**

#include<stdio.h> #include<math.h> float s1[100],s2[50]; float amp=50,freq=50; float fs=1200; float t=0.0; int i,j,L,k=2; void main() { L=100; for(i=0;i<=100;i++) { s1[i]=amp\*sin(2\*3.14\*freq\*t); t=t+(1/fs); } for(i=0;i<=(L/k);i++) { s2[i]=0; } i=0; for(j=0;j<=(L/k);j=j+k)

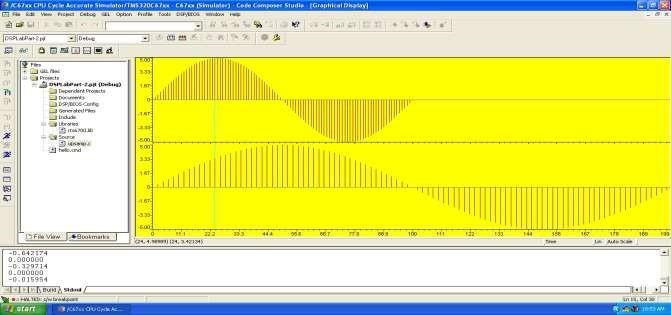
{ s2[i]=s1[j]; i++;

printf("down Sampled Length: %f\n",s2[i]);

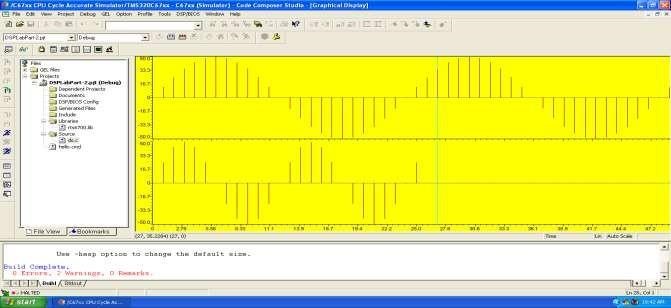
}

}

**Output: a) Up-Sampling:**



**b) Down-Sampling:**



**CONCLUSION:**

Thus the performance of Up-sampling and Down-sampling is verified by using TMS320C67xx.

#### Expt. No. 13 Implementation of Linear and Circular Convolution using

**TMS320C6713**

**AIM:**

To verify Linear Convolution and Circular Convolution using DSP processor TMS320C6713.

**THEORY:**

**(a).** Linear Convolution involves the following operations.

1. Folding
2. Multiplication
3. Addition
4. Shifting

These operations can be represented by a Mathematical Expression as follows:

x[k]= Input signal Samples h[k]= Impulse response co-efficient. y[n]= Convolution output. n = No. of Input samples h = No. of Impulse response co-efficient.

**PROCEDURE:**

* Open Code Composer Studio, make sure the DSP kit is turned on.

* Start a new project using „Project-new „ pull down menu, save it in a separate directory(c:\ti\myprojects) with name **lconv.pjt.**

* Add the source files **conv.c**
* to the project using „Projectadd files to project‟ pull down menu.

* Add the linker command file **hello.cmd** .

(Path: c:\ti\tutorial\dsk6713\hello1\hello.cmd)

* Add the run time support library file **rts6700.lib**

(Path: c:\ti\c6000\cgtools\lib\rts6700.lib)

* Compile the program using the „Project-compile‟ pull down menu or by clicking the shortcut icon on the left side of program window.

* Build the program using the „Project-Build‟ pull down menu or by clicking the shortcut icon on the left side of program window.

* Load the program(lconv.out) in program memory of DSP chip using the

„File-load program‟ pull down menu.

* To View output graphically

Select view  graph  time and frequency.

**PROGRAM**

##### (a). / \* Program to Implement Linear Convolution \* /

#include<stdio.h> int x[30] = {0},y[30] = {0}, h[30] = {0}, m, n, i=0,j; main()

{ for( i=0; i<30; i++ )

{ x[i] = 0; y[i] = 0; h[i] = 0;

}

printf(“Enter the Length of the input & impulse sequence:\n”); scanf(“%d”,&n); scanf(“%d”,&m); printf( “Enter the input sequence:\n”); for(i=0;i<n;i++) scanf(“%d”,&x[i]); printf( “Enter the impulse sequence:\n”); for(j=0;j<m;j++) scanf(“%d”,&h[j]); for(i=0;i<m+n-1;i++)

{ y[i]=0; for(j=0;j<=i;j++) y[i]+ = x[j]\*h[i-j];

} for( i=0;i<m+n-1;i++ )

printf(“The Output Response is y[i]: %d\n”,y[i]);

**}**

**OUTPUT:**

enter value for m4

enter value for n4

Enter values for i/p

1 2 3 4

Enter Values for n

1 2 3 4

The Value of output y[0]=1

The Value of output y[1]=4

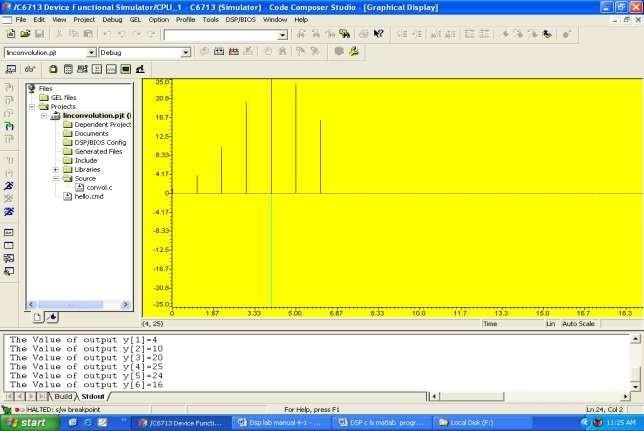
The Value of output y[2]=10

The Value of output y[3]=20

The Value of output y[4]=25

The Value of output y[5]=24

The Value of output y[6]=16



**(b). /\* Program to implement circular convolution\*/**

#include<stdio.h>

int m,n,x[30],h[30],y[30],i,j,temp[30],k,x2[30],a[30]; void main()

{ printf(" enter the length of the first sequence\n"); scanf("%d",&m);

printf(" enter the length of the second sequence\n"); scanf("%d",&n); printf(" enter the first sequence\n");

for(i=0;i<m;i++) scanf("%d",&x[i]);

printf(" enter the second sequence\n"); for(j=0;j<n;j++)

scanf("%d",&h[j]);

if(m-n!=0) /\*If length of both sequences are not equal\*/

{

if(m>n) /\* Pad the smaller sequence with zero\*/

{

for(i=n;i<m;i++) h[i]=0; n=m;

} for(i=m;i<n;i++) x[i]=0;

m=n;

} y[0]=0; a[0]=h[0];

for(j=1;j<n;j++) /\*folding h(n) to h(-n)\*/ a[j]=h[n-j];

/\*Circular convolution\*/ for(i=0;i<n;i++) y[0]+=x[i]\*a[i]; for(k=1;k<n;k++)

{ y[k]=0;

/\*circular shift\*/ for(j=1;j<n;j++) x2[j]=a[j-1]; x2[0]=a[n-1];

for(i=0;i<n;i++)

{ a[i]=x2[i]; y[k]+=x[i]\*x2[i];

}

}

/\*displaying the result\*/

printf(" the circular convolution is\n"); for(i=0;i<n;i++) printf("%d \t",y[i]);

}

**OUTPUT:**

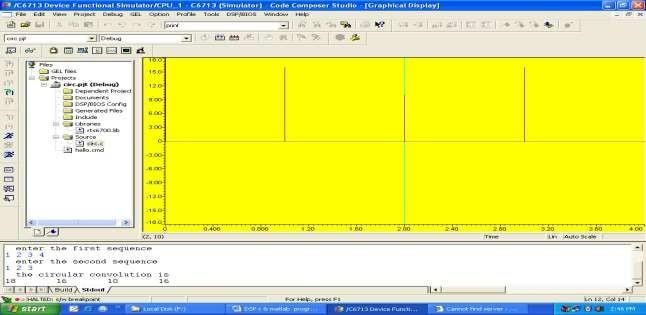
Enter the length of the first sequence 4

Enter the length of the second sequence 3

Enter the first sequence 1 2 3 4

Enter the second sequence 1 2 3

The circular convolution is 18 16 10 16



**CONCLUSION:**

Thus the linear convolution and circular convolution of the given two signals were obtained using TMS320C6713.

#### Expt. No. 14 Implementation of basic arithmetic and Logical operations Using TMS320C5416 Kit

**AIM:**

To implement basic arithmetic operations using DSP Processor TMS 5416.

**EQUIPMENTS:**

CS studio,PC,TMS5416 Kit, USB Cable.

##### ADDITION

;starting address: 1000

;input address: 1500 ;output address

.include "5416\_iv.asm"

.def start

.data

.word 0003h,0006h

.text

start STM #1500h,AR1 ;FIRST INPUT ADDRESS STM #1501h,AR2 ;SECOND INPUT ADDRESS

STM #1600h,AR3 ;OUTPUT ADDRESS

LD \*AR1,A

LD \*AR2,B

ADD A,0,B

STL B,\*AR3 ;STORING THE OUTPUT AT THE ADDRESS POINTED BY AR3, WHICH IS 1600 HERE.

WAIT B WAIT

##### SUBTRACTION

.include "5416\_iv.asm"

.def start

.data

.word 0008h,0004h

.text

start STM #1500h,AR1

STM #1501h,AR2

STM #1600h,AR3 LD \*AR1,A

LD \*AR2,B

SUB A,0,B

STL B,\*AR3 wait nop nop b wait

**MULTIPLICATION**

.include "5416\_IV.asm"

.def start

.data

.word 0010h,0002h

.text

start

STM #1500h,AR3

STM #1501h,AR5 STM #2000h,AR2

MPY \*AR3,\*AR5,B STL B,\*AR2

WAIT B WAIT

SHIFTING

.include "5416\_IV.asm"

.def start

.data

.word 0004h,0002h

.text

|  |  |
| --- | --- |
| start STM #1500h,AR3 STM #1501h,AR5 | |
|  | STM #3000h,AR4  STM #3001h,AR6  LD \*AR3,0,A  SFTA A,-1  STL A,\*AR4  LD \*AR5,0,B  SFTA B,1  STL B,\*AR6 |
| WAIT | B WAIT  **CIRCULAR BUFFER**  .include "5416\_IV.asm"  .def start  .data  .word 1,2,3,4,5  .text |
| start STM #1500h,AR5  STM #2000h,AR6  STM #4h,BK  STM #20h,BRC RPTB L1 | |

LD \*AR5+%,A

STL A,\*AR6+

L1 NOP

WAIT B WAIT

**BIT REVERSAL**

.include "5416\_iv.asm"

.def start

.data

.word 0,1,2,3,4,5,6,7 .text

|  |  |
| --- | --- |
|  |  |
| start | LD #0004h,A STLM A,AR0  STM #1500h,AR1  NOP  NOP  RPT #07h  MVDK \*AR1+0B,#3000h  NOP |
| wait | b wait |

**OUTPUT:**

**Arithmetic Operations:**

|  |  |  |
| --- | --- | --- |
| **Addition:** |  |  |
| **Address** |  | **Data** |
| **Input**: 1500 |  | 0003 |
| **Input:** 1501 |  | 0006 |
| **Output:** 1600 |  | 0009 |
| **Subtraction: Address** |  | **Data** |
| **Input**: 1500 |  | 0004 |
| **Input:** 1501 |  | 0008 |
| **Output:** 1600 |  | 0004 |
| **Multiplication: Address** |  | **Data** |
| **Input**: 1500 |  | 0010 |
| **Input:** 1501 |  | 0002 |
| **Output:** 1600 |  | 0020 |

**Logical Operations:**

**Bit Reversal:**

**Input:** 0 1 2 3 4 5 6 7

0000 0001 0010 0011 0100 0101 0110 0111

**Output:**

**15000:** 0000 0100 0010 0110 0001 0101 0011 0111

0 4 2 6 1 5 3 7

**Circular Buffer:**

**1500:** 0001 0002 0003 0004 0005

**2000:** 0001 0002 0003 0004 0001 0002 0003 0004

**Shifting:**

**Input**: 1500 0004

**Input:** 1501 0002

**Output:** 3000 0002

**Conclusion:**

Thus the basic arithmetic and Logical operations were verified using TMS320C5416 Kit.

**ADDITIONAL EXPERIMENTS**

## Expt. No. 15 1-D Signal Analysis – Speech and ECG Signal

**Aim :**

To develop an algorithm to process the real time speech and ECG signals.

**Requirements**

PC having SCILAB /MATLAB software.

**Algorithm:**

**(a). For ECG Signal Filtering**

1. Read the ECG signal in to SCILAB/MATLAB.
2. Generate a rectangular mask and Gaussian mask.
3. Convolve the noisy ECG signal with rectangular and Gaussian mask individually.
4. Compare both the outputs qualitatively.

**(b). For Speech Signal**

1. Read the real time speech signal.
2. Plot the speech visualization.
3. Amplify the speech signal. 4. Plot the amplified version of the signal.

**PROGRAM**

**(a). ECG signal (Noise Removal)**

clc; clear all; close all; a=load('ecg.dat'); figure(1),plot(a) h=[1 1 1 1 1]; z=conv(a,h,'same'); figure(2),plot(z); h1=[1 2 4 2 1]; z1=conv(a,h1,'same');

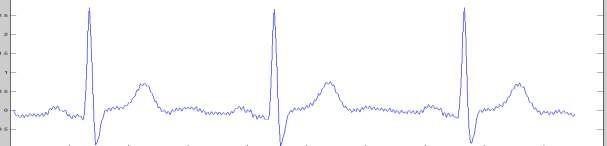
figure(3),plot(z1);

**(b). Speech signal Amplification** clc; clear all; close all;

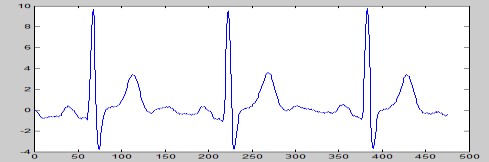
[a,fs]=wavread('female\_speech.wav'); subplot(2,1,1) plot(a); b=3\*a; subplot(2,1,2) plot(b);

**OUPUT:**

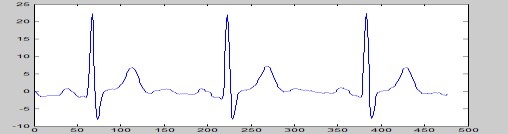
**(a). Noisy ECG Signal**



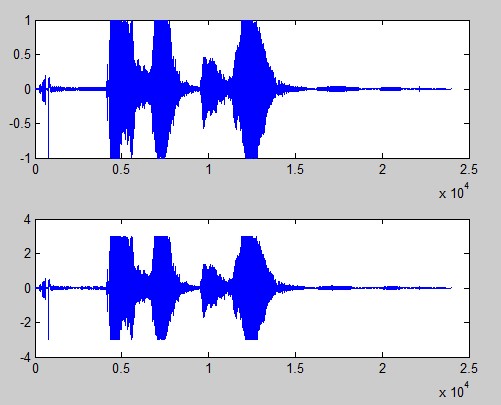
**ECG signal filtering using low pass filter**



**ECG signal filtering using Gaussian function**



**(b). Original Speech Signal and Amplified Speech Signal**



**Result:**

Thus the algorithms for filtering an ECG signal and amplifying the Speech signal were successfully implemented.

### Expt. No. 16 2-D Signal Analysis - Image Enhancement and Segmentation

**Aim :**

(a) To develop an algorithm to enhance the image quality by increase, reduce or invert the image. (b) To develop a algorithm to segment the region of interest from an Image.

**Apparatus :**

PC having SCILAB /MATLAB software.

**ALGORITHM:**

**(a). Image Enhancement**

1. Read an Image from hard drive in to SCILAB/MATLAB.
2. Increase or Decrease the brightness of an Image through linear straight line operator with unit positive slope (with positive and negative constants )
3. Invert the image using negative slope straight line transfer function

**(b). Image Segmentation**

1. Read an Image in to MATLAB/SCILAB
2. Select the threshold of region of interest using imtool or histogram method.
3. Make the pixels which satisfying the threshold condition as „1‟ and assign „0‟ to other pixel positions.
4. Perform morphological operations to remove unwanted clusters.
5. AND the segmented ROI with the original image.

**PROGRAM**

**(a). Image Enhancement**

clear all close all

r=imread('cameraman.tif');

[m n]=size(r); for i=1:m for j=1:n s(i,j)=r(i,j)-80; s1(i,j)=r(i,j)+100;

s2(i,j)=255-r(i,j);

end

end subplot(2,2,1);imshow(r);title('Original image') subplot(2,2,2);imshow(s);title('Dark image') subplot(2,2,3);imshow(s1);title('Bright image') subplot(2,2,4);imshow(s2);title('Image Negative')

**(b). Image Segmentation**

clear all; close all;

clc;

a=imread('onion.png');

[m n p]=size(a);

for i=1:m

for j=1:n for k=1:p if ((a(i,j,1)>=128 & a(i,j,1)<=190)&... (a(i,j,2)>=19 & a(i,j,2)<=75)&...

(a(i,j,3)>=20 & a(i,j,3)<=70)); b(i,j)=1;

else b(i,j)=0; end end

end

end %c=rgb2gray(b); d=bwareaopen(b,1500); d=imfill(d,'holes'); %h1=zeros(m,n,p)

h1(:,:,1)=a(:,:,1).\*(uint8(d)); h1(:,:,2)=a(:,:,2).\*(uint8(d)); h1(:,:,3)=a(:,:,3).\*(uint8(d));

figure(1),imshow(a) figure(2),imshow(b) figure(3),imshow(d) figure(4),imshow(h1) **OUTPUT:**

**(a). Image Enhancement**



**(b). Image Segmentation**

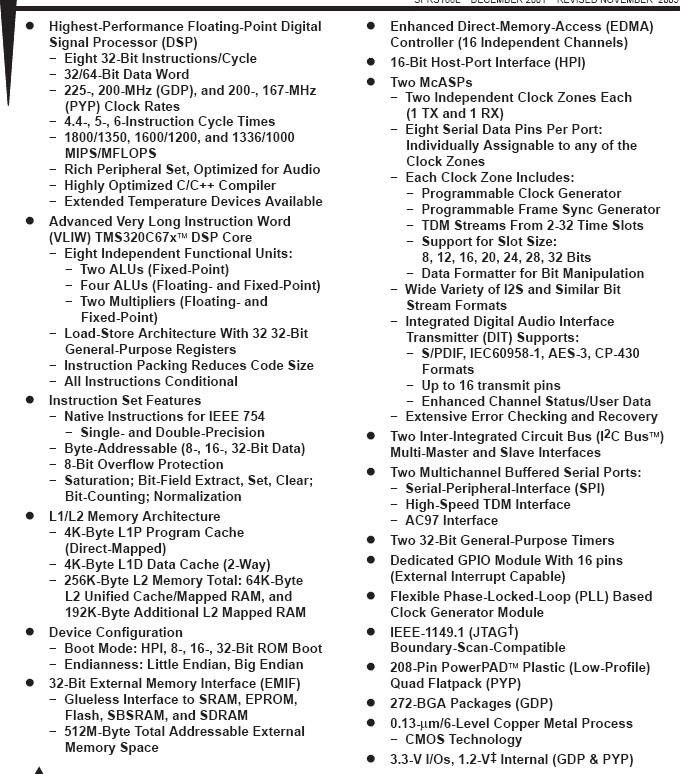


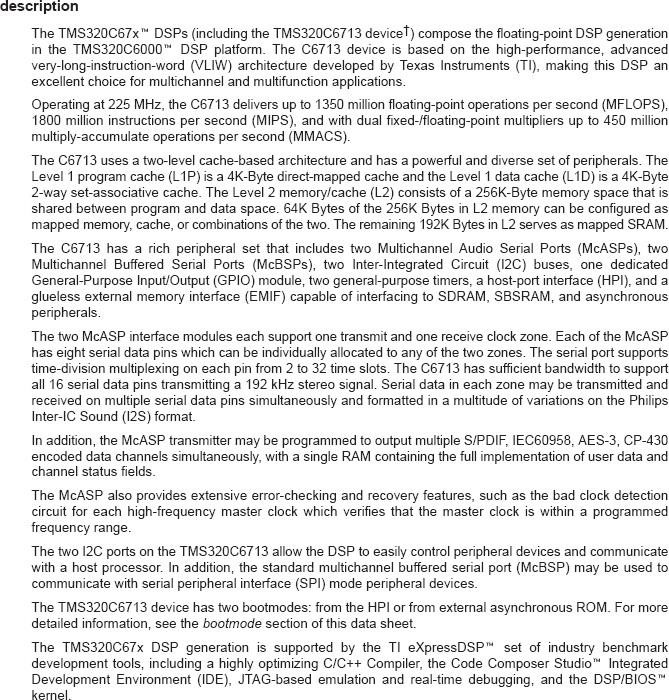
**CONCLUSION:**

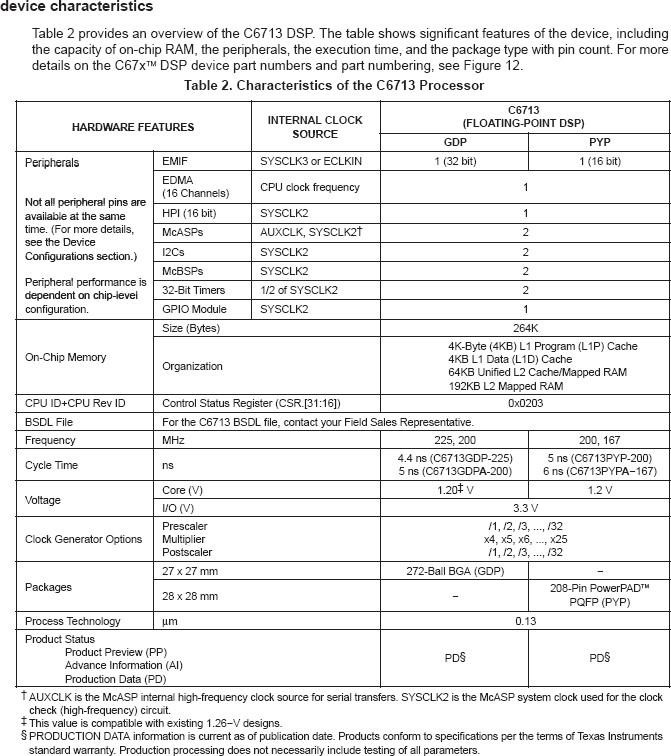
Thus the algorithm for image enhancement and segmentation were developed and implemented successfully.

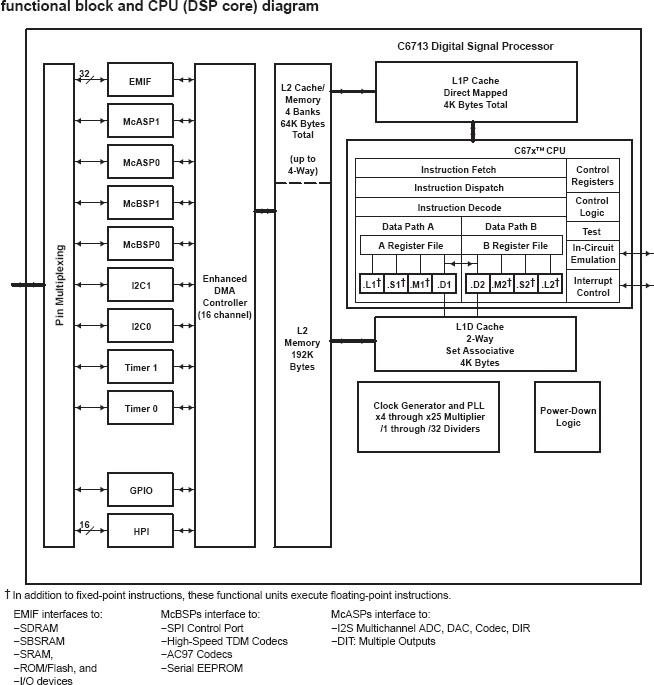
## APENDIX

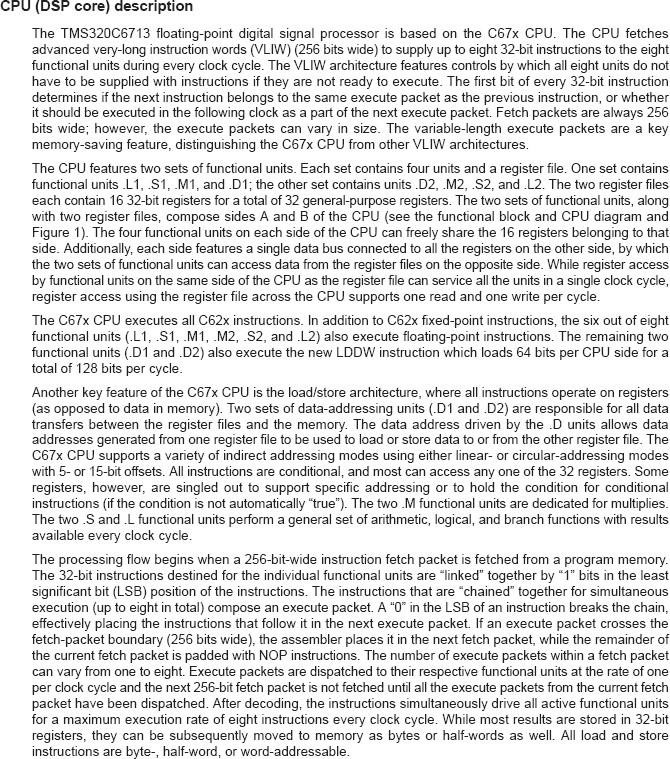
1. **TMS3206713 Floating Point Digital Signal Processor**

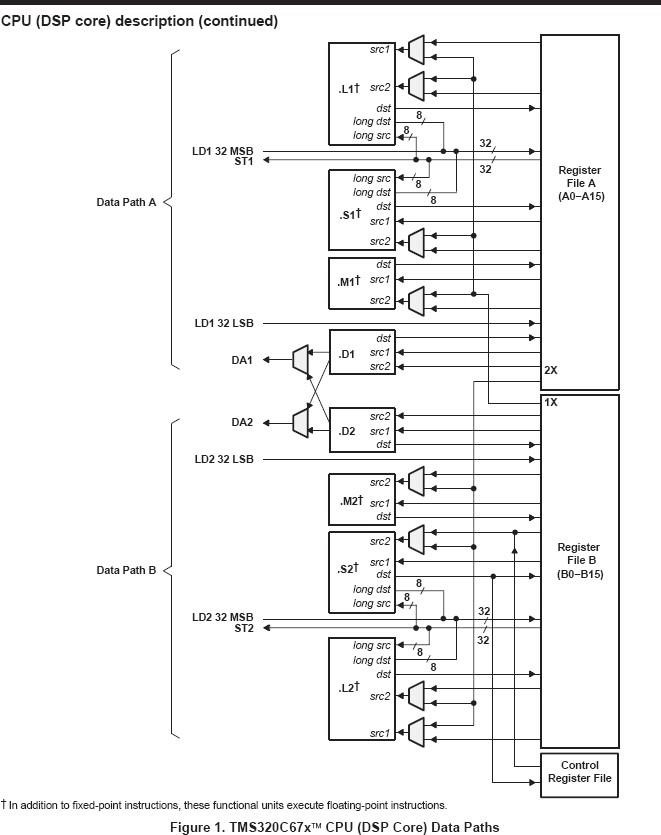




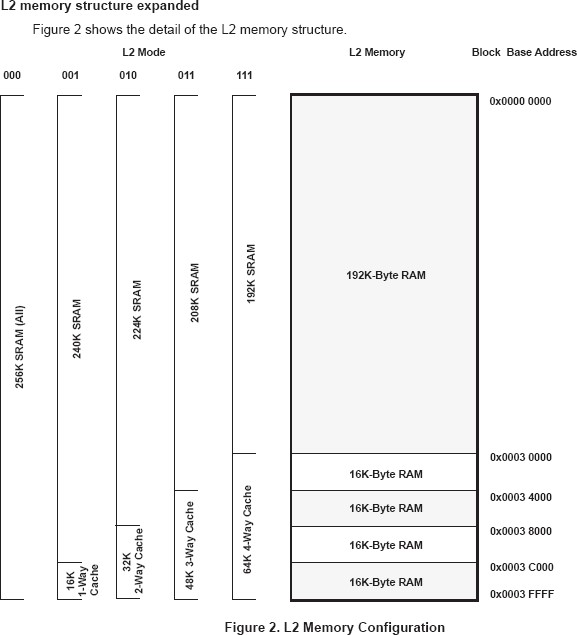












1. **TMS320CX5 Digital Signal Processor**

