S.S.M COLLEGE OF ENGINEERING
KOMARAPALAYAM

Department of _____

_____Laboratory Record

NAME _____COURSE _____

REGISTER No. _____YEAR _____

Certified that this is bonafide record of work done by the above student of the

_____

Laboratory During the year 20 - 20 .

Signature of Lab In charge

Signature of Head of the Department

Submitted for the Practical Examination held on _____

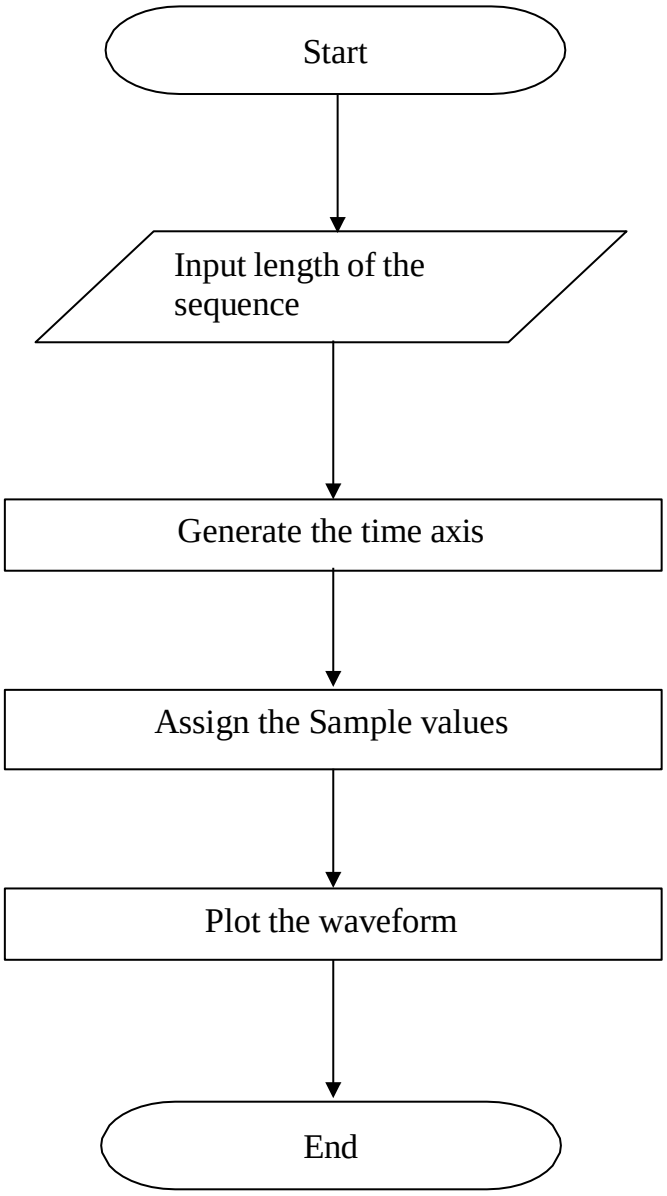
Internal Examiner

External Examiner

INDEX

S.No	Date	Name of the Experiment	Page No	Marks Awarded	Sign	Remarks

FLOW CHART:


Start
Input length of the sequence
Generate the time axis
Assign the Sample values
Plot the waveform
End

EXP.NO:1

DATE:

GENERATION OF ELEMENTARY DISCRETE TIME SEQUENCES

AIM:

To generate the following standard input signals

1. Unit impulse Signal.
2. Unit step Signal.
3. Unit ramp Signal.
4. Exponential Signal.
5. Sine Wave.
6. Cosine Wave

APPARATUS REQUIRED:

PC having MATLAB software.

ALGORITHM:

1. Clear all the variables.
2. Input the length of the sequence.
3. Generate the time axis.
4. Assign the sample values.
5. Plot the waveform.

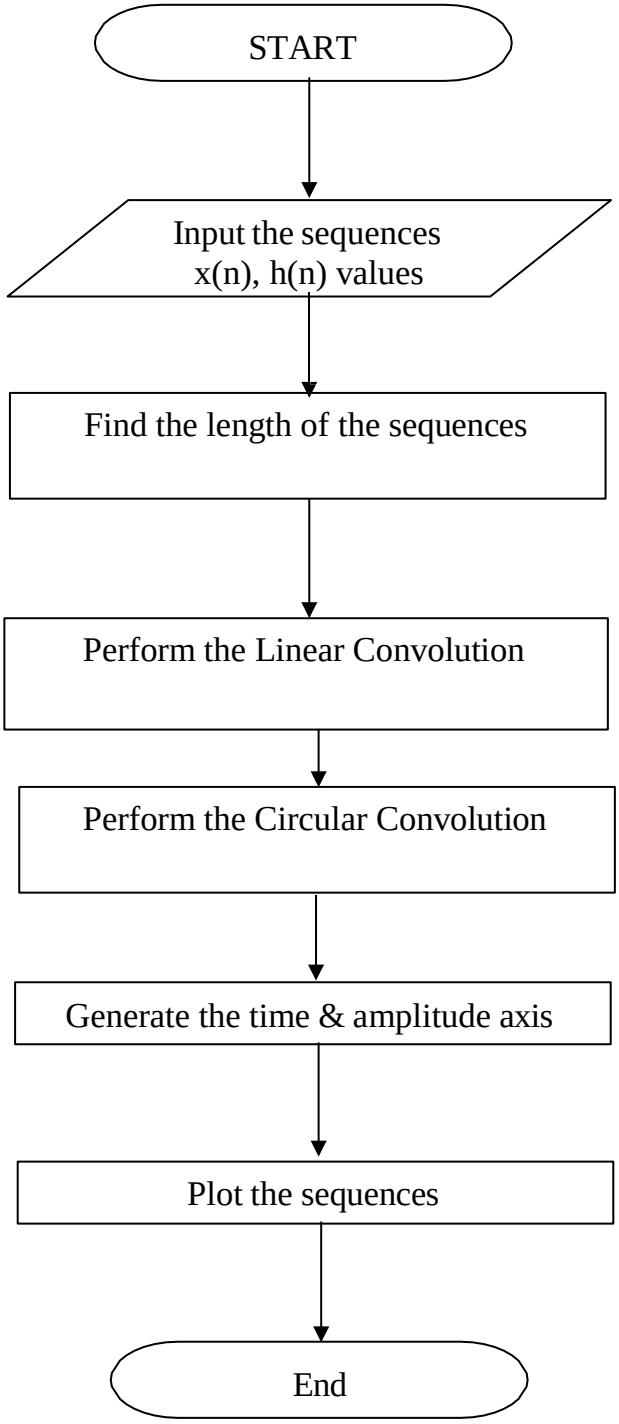
PROGRAM:

```
%unit impulse
clc;
N=25;
n=-3:1:3;
x4=(N==0);
subplot(3,2,1);
stem(n==0);
title('unit impulse signal');
xlabel('time');
ylabel('amp');
%cosine wave
clc;
N=20;
n=(0:.5:N);
x1=((exp(1i*n)+exp(-1i*n))/2);
subplot(3,2,3);
stem(x1);
xlabel('time');
ylabel('amp');
title('cosine wave');

%unit step
clc;
N=20;
n=0:3;N;
x2=(n>=0);
subplot(3,2,5);
stem(x2);
xlabel('time');
ylabel('amp');
title('unit step');
```

```
%sine wave
clc;
n=(1:.4:20);
x=((exp(i*n)-exp(-1*n))/(2*i));
subplot(3,2,2);
stem(x);
xlabel('time');
ylabel('amp');
title('sine wave');

%exponential
clc;
N=5;
a=2;
n=0:1:N;
x5=a.^n;
subplot(3,2,4);
stem(n,x5);
xlabel('time');
ylabel('amp');
title('exponential');
%unit ramp
clc;
n=20;
n=(0:2:n);
x3=n;
subplot(3,2,6);
stem(n,x3);
xlabel('time');
ylabel('amp');
title('unit ramp');
```

RESULT: Thus the above elementary signal has been generated by using the Mat lab.

FLOWCHART:


START
Input the sequences
x(n), h(n) values
Find the length of the sequences
Perform the Linear Convolution
Perform the Circular Convolution
Generate the time & amplitude axis
Plot the sequences
End

EXP.NO:2

DATE:

LINEAR AND CIRCULAR CONVOLUTION OF TWO SEQUENCES

AIM:

To write a program to find the Linear and circular convolution of two sequences.

APPARATUS REQUIRED:

PC having MATLAB software.

ALGORITHM:

1. Input the required sequences.
2. Perform the convolution using the corresponding Matlab function
3. Find the length of the sequences.
4. Generate the time axis.
5. Plot the waveforms

PROGRAM:

Linear Convolution

```
clc;clear all;close all;
x=input('Enter X[n]:');
h=input('enter h[n]:');
y= conv(x,h);
subplot(3,2,1);
stem(x);
xlabel('time');
ylabel('amplitude');
title('input sequence x[n]');
subplot(3,2,2);
stem(h);
xlabel('time');
ylabel('amplitude');
title('impluse resonse of the system h[n]');
subplot(3,2,3);
stem(y);
xlabel('time');
ylabel('amplitude');
title('linear convolution');
disp(y);
```

Circular Convolution

```
clc;clear all;close all;
x=input('Enter X[n]:');
h=input('enter h[n]:');
y1 =cconv(x,h,4);
subplot(3,2,1);
stem(x);
xlabel('time');
ylabel('amplitude');
title('input sequence x[n]');
subplot(3,2,2);
stem(h);
xlabel('time');
ylabel('amplitude');
title('impluse resonse of the system h[n]');
subplot(3,2,3);
stem(y1);
xlabel('time');
ylabel('amplitude');
title('circular convolution');
disp(y1);
```

Output :

Linear Convolution :

Enter the 1st sequence :
Enter the 2nd sequence :
The resultant Signal is : y =

Circular Convolution :

Enter the 1st sequence :
Enter the 2nd sequence :
The resultant Signal is : y =

RESULT:

Thus the MATLAB Program for linear and circular convolution of two sequence has been executed and the output is verified successfully.

PROGRAM:

% Program for computing crosscorrelation

```
clc;clear all;close all;
x=input('Enter X[n]:');
h=input('enter h[n]:');
y= xcorr(x,h);
subplot(2,2,1);
stem(x);
xlabel('time');
ylabel('amplitude');
title('input sequence x[n]');
subplot(2,2,2);
stem(h);
xlabel('time');
ylabel('amplitude');
title('impluse resonse of the system h[n]');
subplot(2,2,3);
stem(y);
xlabel('time');
ylabel('amplitude');
title('crosscorrelation ');
disp(y);
```

% Program for computing autocorrelation

```
clc;clear all;close all;
x=input('Enter X[n]:');
y= xcorr(x,x);
subplot(2,2,1);
stem(x);
xlabel('time');
ylabel('amplitude');
title('input sequence x[n]');
subplot(2,2,2);
stem(y);
xlabel('time');
ylabel('amplitude');
title('auto  correlation ');
disp(y);
```

OUTPUT:

Cross Correlation

Enter the 1st sequence :
Enter the 2nd sequence :
The resultant Signal is : y =

Auto Correlation

Enter the sequence :
The resultant Signal is : y =

EXP.NO:3

DATE:

CROSS CORRELATION AND AUTO CORRELATION

AIM : To write a program in MATLAB to perform Cros Correlation and auto correlation of an input sequence.

APPARATUS REQUIRED:

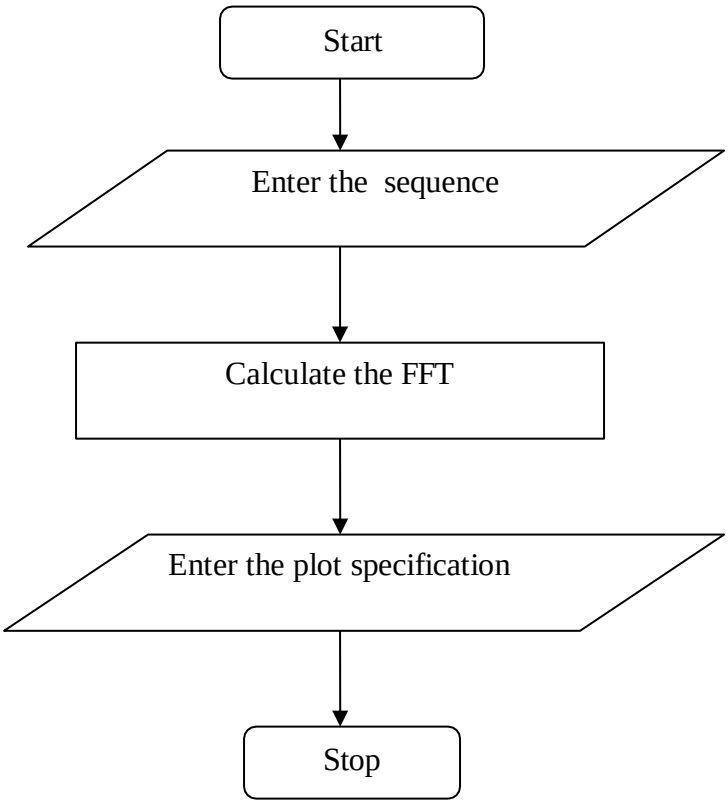
PC having MATLAB software.

ALGORITHM :

1. Start
2. Read the first input sequence
3. Read thesecond input sequence
4. Perform cross correlation and auto correlation using correlation function
5. Plot the sequences
6. Display the input and output sequences
7. Stop

Result : Thus the correlation of two signals has been executed and the output is verified successfully. using MATLAB.

FLOWCHART:


Start
Enter the sequence
Calculate the FFT
Enter the plot specification
Stop

EXP.NO:4

DATE:

FREQUENCY ANALYSIS USING DFT

AIM:

To write a program to calculate Frequency Analysis using DFT for a given sequence.

APPARATUS REQUIRED:

PC having MATLAB software.

ALGORITHM:

- Start the program.
- Enter the sequence.
- Using FFT algorithm find the Fourier transform of the given numbers.
- Enter the plot specifications.
- Stop the program.

PROGRAM:

```
clear all;
x=input ('enter the sequence:');
N=input('enter the no of samples:');
X=fft(x,N);
disp(X);
subplot(2,2,1);
stem(X);
xlabel('X(k)');
ylabel('amplitude');
title('FFT');
%computation of magnitude response of X(K);
subplot(2,2,2);
z=abs(X);
disp('magnitude response is');
disp(z);
stem(z);
title('plot of magnitude of x(k)');
xlabel('no index k');
ylabel('magnitude');
%computation of the phase response
subplot(2,2,3);
t=angle(X);
disp('phase response is');
disp(t);
stem(t);
grid;
title('plot of phase of (x(k)');
xlabel('index k');
ylabel('angle in radians');
%computation of IFFT
x=ifft(X,N);
disp(x);
subplot(2,2,4);
stem(x);
xlabel('x(n)');
ylabel('amplitude');
title('IFFT');
```

Output :

DFT :

Enter the sequence =
Enter the length of Fourier Transform =
DFT is X(K) =
magnitude response of X(K)=
Phase response of X(K) =

IDFT :

Enter the sequence :
Enter length of DFT =
IDFT x(n)=

RESULT:

Thus the MATLAB Program to calculate the Frequency Analysis using DFT for the given sequence has been executed and the output is verified successfully.

PROGRAM:

```
%FIR filter design window techniques
clc;
clear all;
close all;
 c=input('enter your choice of window function 1.Rectangular 2.Kaiser 3.bartlett 4.blackman
5.hamming 6.hanning:/n');
rp=input('enter passband ripple');
rs=input('enter the stopband ripple');
fp=input('enter passband freq');
fs=input('enter stopband freq');
f=input('enter sampling freq');
beta=input('enter the beta value');
wp=2*fp/f;
ws=2*fs/f;
num=-20*log10(sqrt(rp*rs))-13;
dem=14.6*(fs-fp)/f;
n=ceil(num/dem);
n1=n+1;
if(rem(n,2)~=0);
    n1=n
    n=n-1
end
if(c==1)
  y=rectwin(n1);
  disp('Rectangular window filter response');
end
if(c==2)
  y=kaiser(n1,beta);
  disp('Kaiser window filter response');
End
if(c==3)
 y=bartlett(n1);
  disp('Bartlett window filter response');
end
if(c==4)
 y=blackman(n1);
disp('Blackman window filter response');
end
if(c==5)
y=hamming(n1);
disp('Hamming window filter response');
end
```

EXP.NO:5

DATE:

FIR FILTER DESIGN

AIM:

To Design the FIR filter using window techniques

APPARATUS REQUIRED:

PC having MATLAB software.

ALGORITHM:

- Start the program.
- Get the pass band and stop band ripples
- Get the pass band and stop band frequencies
- Get the sampling frequency
- Calculate the order of the filter
- Find the window coefficients
- Drraw the magnitude and phase response
- Stop the program

```
if(c==6)
y=hanning(n1);
disp('Hamming window filter response');
end
%LPF
b=fir1(n,wp,y);
[h,o]=freqz(b,1,256);
m=20*log10(abs(h));
subplot(2,2,1);
plot(o/pi,m);
title LPF
ylabel('Gain in db');
xlabel('(a) normalized freq')
%HPF
b=fir1(n,wp,'high',y);
[h,o]=freqz(b,1,256);
m=20*log(abs(h));
subplot(2,2,2);
plot(o/pi,m);
title('HPF');
ylabel('Gain in db');
xlabel('(b)normalized freqz');
%BPF
wn=[wp ws];
b=fir1(n,wn,y);
[h,o]=freqz(b,1,256);
m=20*log10(abs(h));
subplot(2,2,3);plot(o/pi,m);
title('BPF');
ylabel('Gain in db');
xlabel('(c)normalized freqz');
%BSF
b=fir1(n,wn,'stop',y);
[h,o]=freqz(b,1,256);
m=20*log(abs(h));
subplot(2,2,4);
plot(o/pi,m);
title('BSF');
ylabel('Gain in db');
xlabel('(d)normalized freqz');
```

RESULT:

Thus the MATLAB Program to design the Fir Filter Using Window Techniques was done successfully & verified.

PROGRAM:

```
clc;
close all;
format long;
display('Butterworth filter');
a2=input('enter the choice of the filter in number\n 1. analog LPF\n 2.analog HPF\n 3. analog BPF\n
4.analog BSF\n 5. digital LPF\n 6.digital HPF\n 7.digital BPF\n 8.digital BSF\n Choice:');
rp=input('enter the passband ripple:');
rs=input('enter the stopband ripple:');
wp=input('enter the passband freq:');
ws=input('enter the stopband freq:');
fs=input('enter the sampled freq:');
w1=2*wp/fs;
w2=2*ws/fs;
w=0:0.01:pi;
if(a2==1)
[n,wn]=buttord(w1,w2,rp,rs,'s');
[b,a]=butter(n,wn,'s');
[h,om]=freqs(b,a,w);
end
if(a2==2)
[n,wn]=buttord(w1,w2,rp,rs);
[b,a]=butter(n,wn,'high','s');
[h,om]=freqs(b,a,w);
end
if(a2==3)
[n]=buttord(w1,w2,rp,rs);
wn=[w1 w2];
[b,a]=butter(n,wn,'bandpass','s');
[h,om]=freqs(b,a,w);
end
if(a2==4)
[n]=buttord(w1,w2,rp,rs,'s');
wn=[w1 w2];
[b,a]=butter(n,wn,'stop','s');
[h,om]=freqs(b,a,w);
end
if(a2==5)
[n,wn]=buttord(w1,w2,rp,rs,'s');
[z,p,k]=butter(n,wn);
[b,a]=zp2tf(z,p,k);
[b,a]=butter(n,wn,'s');
[h,om]=freqz(b,a,w);
end
if(a2==6)
[n,wn]=buttord(w1,w2,rp,rs);
[b,a]=butter(n,wn,'high');
```

EXP.NO :6a

DATE:

IIR FILTER DESIGN- *BUTTERWORTH FILTER*

AIM:

To Design the (I) Butterworth analog filters of low pass ,high pass ,band pass and band stop filter (II) Butterworth digital filters of low pass and high pass band pass and band stop filter using MATLAB

APPARATUS REQUIRED:

PC having MATLAB software.

ALGORITHM:

- Start the program.
- Get the pass band and stop band ripples
- Get the pass band and stop band frequencies
- Get the sampling frequency
- Calculate the order of the filter
- Find the filter coefficient
- Draw the magnitude and phase response
- Stop the program

```
[h,om]=freqz(b,a,w);
end
if(a2==7)
[n]=buttord(w1,w2,rp,rs);
wn=[w1 w2];
[b,a]=butter(n,wn,'bandpass');
[h,om]=freqz(b,a,w);
end
if(a2==8)
[n]=buttord(w1,w2,rp,rs);
wn=[w1 w2];
[b,a]=butter(n,wn,'stop');
[h,om]=freqz(b,a,w);
end
m=20*log(abs(h));
an=angle(h);
subplot(2,1,1);
plot(om/pi,m);
ylabel('gain in db');
xlabel('(a)normalised frequency');
title('amplitude response');
subplot(2,1,2);
plot(om/pi,an);
ylabel('phase in radians');
xlabel('(b)normalised frequency');
title('phase response');
```

RESULT:

Thus the MATLAB Program to design the (I) butterworth analog filters of lowpass and high pass ,band pass and band stop filter (II) butterworth digital filters of lowpass and high pass filter ,band pass and band stop filter was done successfully & verified.

PROGRAM

```
clc;
close all;
format long;
display('chebyshev filter');
a2=input('enter the choice of the filter in number\n 1. analog LPF\n 2.analog HPF\n 3. analog BPF\n
4.analog BSF\n 5. digital LPF\n 6.digital HPF\n 7.digital BPF\n 8.digital BSF\n Choice:');
rp=input('enter the passband ripple:');
rs=input('enter the stopband ripple:');
wp=input('enter the passband freq:');
ws=input('enter the stopband freq:');
fs=input('enter the sampled freq:');
w1=2*wp/fs;
w2=2*ws/fs;
w=0:0.01:pi;
if(a2==1)
[n,wn]=cheb1ord(w1,w2,rp,rs,'s');
[b,a]=cheby1(n,rp,wn,'s');
[h,om]=freqs(b,a,w);
end
if(a2==2)
[n,wn]=cheb1ord(w1,w2,rp,rs,'s');
[b,a]=cheby1(n,rp,wn,'high','s');
[h,om]=freqs(b,a,w)
end
if(a2==3)
[n]=cheb1ord(w1,w2,rp,rs,'s');
wn=[w1 w2];
[b,a]=cheby1(n,rp,wn,'bandpass','s');
[h,om]=freqs(b,a,w);
end
if(a2==4)
[n]=cheb1ord(w1,w2,rp,rs,'s');
wn=[w1 w2];
[b,a]=cheby1(n,rp,wn,'stop','s');
[h,om]=freqs(b,a,w);
end
if(a2==5)
[n,wn]=cheb1ord(w1,w2,rp,rs);
[b,a]=cheby1(n,rp,wn);
[h,om]=freqz(b,a,w);
end
if(a2==6)
 [n,wn]=cheb1ord(w1,w2,rp,rs);
[b,a]=cheby1(n,rp,wn,'high');
[h,om]=freqz(b,a,w);
end
```

EXP.NO:6b

DATE:

IIR FILTER DESIGN -CHEBYSHEV FILTER

AIM:

To Design the (I) chebyshev analog filters of low pass ,high pass ,band pass and band stop filter (II) chebyshev digital filters of low pass and high pass band pass and band stop filter using MATLAB

APPARATUS REQUIRED:

PC having MATLAB software.

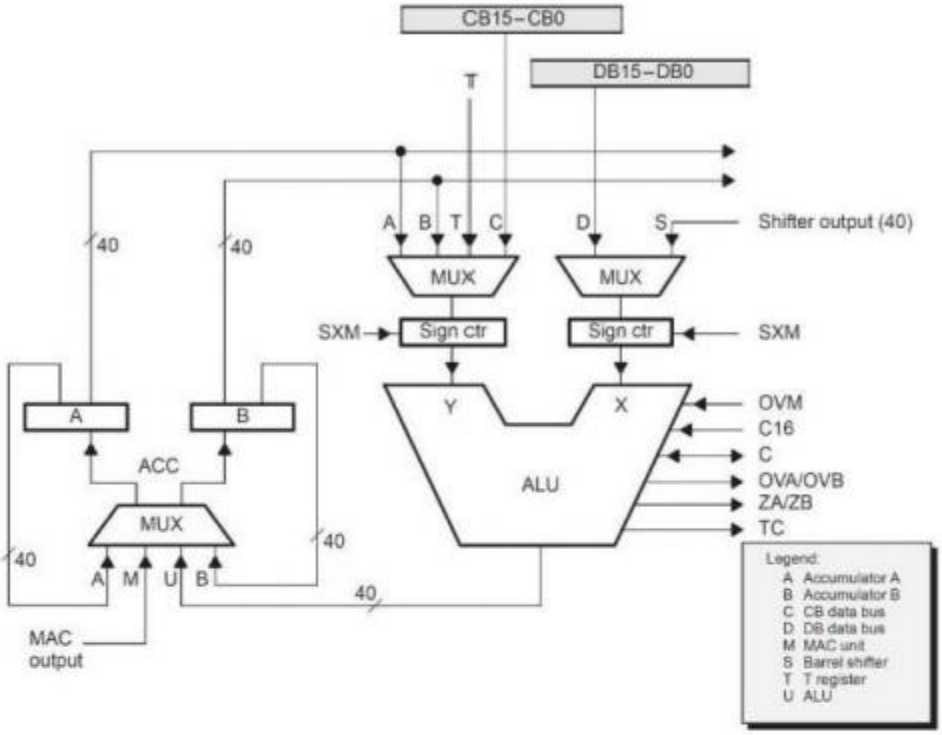
ALGORITHM:

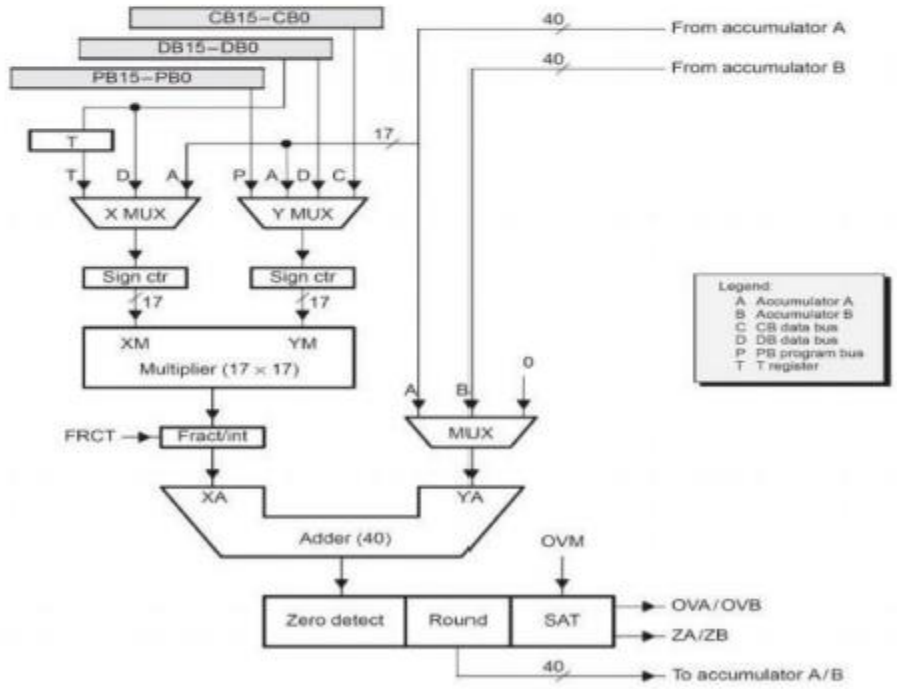
- Start the program.
- Get the pass band and stop band ripples
- Get the pass band and stop band frequencies
- Get the sampling frequency
- Calculate the order of the filter
- Find the filter coefficient
- Draw the magnitude and phase response
- Stop the program

```
if(a2==7)
[n]=cheb1ord(w1,w2,rp,rs);
wn=[w1 w2];
[b,a]=cheby1(n,rp,wn,'bandpass');
[h,om]=freqz(b,a,w);
end
if(a2==8)
[n]=cheb1ord(w1,w2,rp,rs);
wn=[w1 w2];
[b,a]=cheby1(n,rp,wn,'stop');
[h,om]=freqz(b,a,w);
end
m=20*log10(abs(h));
an=angle(h);
subplot(2,1,1);
plot(om/pi,m);
ylabel('gain in db');
xlabel('(a)normalised frequency');
title('amplitude response');
subplot(2,1,2);
plot(om/pi,an);
ylabel('phase in radians');
xlabel('(b)normalised frequency');
title('phase response');
```

RESULT: Thus, the MATLAB Program to design the (I) chebyshev analog filters of lowpass and high pass, band pass and band stop filter (II) chebyshev digital filters of lowpass and high pass filter ,band pass and band stop filter was done successfully & verified.

Architecture of Digital Signal Processor





EX.NO:7

DATE:

STUDY OF ARCHITECTURE OF DIGITAL SIGNAL PROCESSOR

AIM: To study of architecture of digital signal processor

ARCHITECTURE

The 54XX DSPs use an advanced, modified Harvard architecture that maximizes processing power by maintaining one program memory bus and three data memory buses. These processors also provide an arithmetic logic unit (ALU) that has a high degree of parallelism, application-specific hardware logic, on-chip memory, and additional on-chip peripherals. These DSP families also provide a highly specialized instruction set, which is the basis of the operational flexibility and speed of these DSPs. Separate program and data spaces allow simultaneous access to program instructions and data, providing the high degree of parallelism. Two reads and one write operation can be performed in a single cycle. Instructions with parallel store and applicationspecific instructions can fully utilize this architecture. In addition, data can be transferred between data and program spaces. Such parallelism supports a powerful set of arithmetic, logic, and bit-manipulation operations that can all be performed in a single machine cycle. Also included are the control mechanisms to manage interrupts, repeated operations, and function calls

1.Central Processing Unit (CPU)

The CPU of the 54XX devices contains:

- A 40-bit arithmetic logic unit (ALU)
- Two 40-bit accumulators
- A barrel shifter
- A 17 -bit multiplier/adder
- A compare, select, and store unit (CSSU)

2. Arithmetic Logic Unit (ALU)

The ‟67XX devices perform 2s-complement arithmetic using a 40-bit ALU and two 40-bit accumulators (ACCA and ACCB). The ALU also can perform Boolean operations. The ALU can function as two 16-bit ALUs and perform two 16-bit operations simultaneously when the C16 bit in status register 1 (ST1) is set.

3. Accumulators The accumulators,

ACCA and ACCB, store the output from the ALU or the multiplier / adder block; the accumulators can also provide a second input to the ALU or the multiplier / adder. The bits in each accumulator are grouped as follows: • Guard bits (bits 32–39) • A high-order word (bits 16–31) • A low-order word (bits 0–15) Instructions are provided for storing the guard bits, the high-order and the loworder accumulator words in data memory, and for manipulating 32-bit accumulator words in or out of data memory. Also, any of the accumulators can be used as temporary storage for the other.

4. Barrel Shifter The ‟67XX‟s barrel shifter has a 40-bit input connected to the accumulator or data memory (CB, DB) and a 40-bit output connected to the ALU or data memory (EB). The barrel shifter produces a left shift of 0 to 31 bits and a right shift of 0 to 16 bits on the input data. The shift requirements are defined in the shift-count field (ASM) of ST1 or defined in the temporary register (TREG), which is designated as a shift-count register. This shifter and the exponent detector normalize the values in an accumulator in a single cycle. The least significant bits (LSBs) of the output are filled with 0s and the most significant bits (MSBs) can be either zero-filled or signextended, depending on the state of the sign-extended mode bit (SXM) of ST1Additional shift capabilities enable the processor to perform numerical scaling, bit extraction, extended arithmetic, and overflow prevention operations

5. Multiplier/Adder The multiplier / adder perform 17-bit 2s-complement multiplication with a 40- bit accumulation in a single instruction cycle. The multiplier / adder block consists of several elements: a multiplier, adder, signed/unsigned input control, fractional control, a zero detector, a rounder (2s-complement), overflow/saturation logic, and TREG.

The multiplier has two inputs: one input is selected from the TREG, a data memory operand, or an accumulator; the other is selected from the program memory, the data memory, an accumulator, or an immediate value. The fast on-chip multiplier allows the C67XX to perform operations such as convolution, correlation, and filtering efficiently. In addition, the multiplier and ALU together execute multiply/accumulate (MAC) computations and ALU operations in parallel in a single instruction cycle. This function is used in determining the Euclid distance, and in implementing symmetrical and least mean square (LMS) filters, which are required for complex DSP algorithms.

6. Compare, Select, and Store Unit (CSSU)

The compare, select, and store unit (CSSU) performs maximum comparisons between the accumulator's high and low words, allows the test/control (TC) flag bit of status register 0 (ST0) and the transition (TRN) register to keep their transition histories, and selects the larger word in the accumulator to be stored in data memory. The CSSU also accelerates Viterbi-type butterfly computation with optimized on-chip hardware.

7. Program Control

Program control is provided by several hardware and software mechanisms: The program controller decodes instructions, manages the pipeline, stores the status of operations, and decodes conditional operations. Some of the hardware elements included in the program controller are the program counter, the status and control register, the stack, and the address-generation logic. Some of the software mechanisms used for program control includes branches, calls, and conditional instructions, are peat instruction, reset, and interrupt. The C67XX supports both the use of hardware and software interrupts for program control. Interrupt service routines are vectored through a re-locatable interrupt vector table. Interrupts can be globally enabled / disabled and can be individually masked through the interrupt mask register (IMR). Pending interrupts are indicated in the interrupt flag register (IFR). For detailed information on the structure of the interrupt vector table, the IMR and the IFR, see the device-specific data sheets

8. Status Registers (ST0, ST1)

The status registers, ST0 and ST1, contain the status of the various conditions and modes for the "67XX devices. ST0 contains the flags (OV, C, and TC) produced by arithmetic operations and bit manipulations in addition to the data page pointer (DP) and the auxiliary register pointer (ARP) fields. ST1 contains the various modes and instructions that the processor operates on and executes.

9. Auxiliary Registers (AR0–AR7) The eight 16-bit auxiliary registers (AR0–AR7) can be accessed by the central arithmetic logic unit (CALU) and modified by the auxiliary register arithmetic units (ARAUs). The primary function of the auxiliary registers is generating 16-bit addresses for data space. However, these registers also can act as general-purpose registers or counters.

10. Temporary Register (TREG) The TREG is used to hold one of the multiplicands for multiply and multiply/accumulate instructions. It can hold a dynamic (execution-time programmable) shift count for instructions with a shift operation such as ADD, LD, and SUB. It also can hold a dynamic bit address for the BITT instruction. The EXP instruction stores the exponent value computed into the TREG, while the NORM instruction uses the TREG value to normalize the number. For ACS operation of Viterbi decoding, TREG holds branch metrics used by the DADST and DSADT instructions.

11. Transition Register (TRN) The TRN is a 16-bit register that is used to hold the transition decision for the path to new metrics to perform the Viterbi algorithm. The CMPS (compare, select, max, and store) instruction updates the contents of the TRN based on the comparison between the accumulator high word and the accumulator low word.

12. Stack-Pointer Register (SP) The SP is a 16-bit register that contains the address at the top of the system stack. The SP always points to the last element pushed onto the stack. The stack is manipulated by interrupts, traps, calls, returns, and the PUSHD, PSHM, POPD, and POPM instructions. Pushes and pops of the stack pre decrement and post increment, respectively, all 16 bits of the SP. " "

13. Circular-Buffer-Size Register (BK) The 16-bit BK is used by the ARAUs in circular addressing to specify the data block size.

14. Block-Repeat Registers (BRC, RSA, REA) The block-repeat counter (BRC) is a 16-bit register used to specify the number of times a block of code is to be repeated when performing a block repeat. The blockrepeat start address (RSA) is a 16-bit register containing the starting address of the block of program memory to be repeated when operating in the repeat mode. The 16- bit block-repeat end address (REA) contains the ending address if the block of program memory is to be repeated when operating in the repeat mode.

15. Interrupt Registers (IMR, IFR) The interrupt-mask register (IMR) is used to mask off specific interrupts individually at required times. The interrupt-flag register (IFR) indicates the current status of the interrupts.

16. Processor-Mode Status Register (PMST) The processor-mode status registers (PMST) controls memory configurations of the 67XX devices.

17. Power-Down Modes There are three power-down modes, activated by the IDLE1, IDLE2, and IDLE3 instructions. In these modes, the 67XX devices enter a dormant state and dissipate considerably less power than in normal operation. The IDLE1instruction is used to shut down the CPU. The IDLE2 instruction is used to shut down the CPU and on-chip peripherals. The IDLE3 instruction is used to shut down the 67XX processor completely. This instruction stops the PLL circuitry as well as the CPU and peripherals.

Result:

Thus the architecture of digital signal processor was studied.

PROGRAM:

```
#include<stdio.h>  //header file
int main()
{
  int a, b , add, sub, mul; // integer type variables
  float divide;            // float type variable

  printf("Enter first number:");
  scanf("%d", &a);

  printf("\nEnter second number:");
  scanf("%d", &b);

  add=(a+b);
  printf("\nAddition of first and second number is= %d",add);

  sub=(a-b);
  printf("\nSubtraction of second  from first = %d",sub);

  mul=(a*b);
  printf("\nMultiplication of first and second number=%d",mul);

  divide=a/(float)b;                // typecasting as division may result in decimal point values

  printf("\nDivision of first number by second number =%.2f",divide);
  return 0;   // program ended with zero errors
}
```

PROGRAM:

Program for immediate addressing mode . MMREGS .TEXT START: LDP #100H LACC #1241H ADD #1200H SACL 2H HTL: END	**Program for direct addressing mode** .MMREGS . TEXT START: LDP #100H LACC 0H ADD 1H SACL 2H HTL: END
Program for adding two numbers with indirect addressing mode. .MMREGS . TEXT START: LAR AR0,#8000H MAR *,AR0 LACC *+,0 ;WITH ZERO SHIFT ADD *+	SACL *+ HTL: END

EX.NO:8

DATE:

MAC OPERATION USING VARIOUS ADDRESSING MODES

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

APPARATUS REQUIRED:

1. System with TMS 320C50 debugger software
2. TMS 320C50 Kit.
3. RS232 cable.

PROGRAM LOGIC:

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.

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.

OUTPUT:

RESULT: Thus, the mac operation using various addressing modes using TMS320 kit was performed successfully.

PROGRAM:

SINE WAVEFORM

```
#include <math.h>
#include "zzzcfg.h"
#include <c:\ccstudio_v3.1\c5400\dsk5416\include\dsk5416.h>
#include <c:\ccstudio_v3.1\c5400\dsk5416\include\dsk5416_pcm3002.h>
#define BUFSIZE 0x64
#define PI 3.1416
float ar0, ar1, ar2;
float step=(2*PI*10000)/48000;
short left_input,right_input,left_output,right_output;
  DSK5416_PCM3002_Config setup = {

    0x1ff,      // Set-Up Reg 0 - Left channel DAC attenuation
    0x1ff,      // Set-Up Reg 1 - Right channel DAC attenuation
    0x0,        // Set-Up Reg 2 - Various ctl e.g. power-down modes
    0x0         // Set-Up Reg 3 - Codec data format control
};
void main ()
{
          int size=BUFSIZE;
    short data,i;
    float value;
    DSK5416_PCM3002_CodecHandle hCodec;
       // Initialize the board support library
    DSK5416_init();
    // Start the codec
    hCodec = DSK5416_PCM3002_openCodec(0, &setup);
    // Set codec frequency
    DSK5416_PCM3002_setFreq(hCodec, 48000);
    ar0 = cos(step);
          ar1 = sin(-step);
ar2 = sin(-(2*step));
                    // Endless loop IO audio codec
          while(1){

                    /*******  processing *********/
             value = (2*ar0*ar1) - ar2;
      data = value*16000;
      ar2 = ar1;
      ar1 = value;
      left_output=data;
      right_output=data;0078
             /*****************************/
                    while(!DSK5416_PCM3002_write16(hCodec, left_output));
                    while(!DSK5416_PCM3002_write16(hCodec, right_output));
          }  }
```

EX.NO. 9

DATE:

Generation of various signals and random noise

AIM:

To Genarate the signals using TMS320C5X Processor

APPARATUS REQUIRED:

System with TMS 320C50 software

TMS320C50 Processor kit

USB Cable

CRO

Function Generator

PROCEDURE :

1. Connect a Signal Generator/audio input to the **LINE IN** Socket or connect a microphone to the **MIC IN** Socket.

2. Connect CRO/Desktop Speakers to the Socket Provided for **LINE OUT or** connect a headphone to the **Headphone out** Socket.

3. Now Switch on the DSK and Bring Up Code Composer Studio on the PC.

4. Use the **Debug → Connect** menu option to open a debug connection to the DSK board

5. Create a new project with name **Fir.pjt.**

6. Open the **File** Menu ➔ **new** ➔ **DSP/BIOS Configuration** ➔select "**dsk5416.cdb**" and save it as "**xyz.cdb**"

TRIANGULAR WAVEFORM

```
#include<math.h>
#include "tri_wav_configcfg.h"
#include <dsk5416.h>
#include <dsk5416_pcm3002.h>
#define STEP 300
int temp=0;
short left_output,right_output;

DSK5416_PCM3002_Config setup = {
   0x1ff,      // Set-Up Reg 0 - Left channel DAC attenuation
   0x1ff,      // Set-Up Reg 1 - Right channel DAC attenuation
   0x0,        // Set-Up Reg 2 - Various ctl e.g. power-down modes


 0x0         // Set-Up Reg 3 - Codec data format control
};

void main ()
{
int count=0,sign=1,i;
   DSK5416_PCM3002_CodecHandle hCodec;
      // Initialize the board support library
   DSK5416_init();
   // Start the codec
   hCodec = DSK5416_PCM3002_openCodec(0, &setup);
   // Set codec frequency
   DSK5416_PCM3002_setFreq(hCodec, 48000);
         // Endless loop IO audio codec
         while(1){
                  count=count + sign;
     temp=count*200;
           if(count==160||count==0)
                  sign = -sign;
           left_output=temp;
     right_output=temp;
                  // Write 16 bits to the codec, loop to retry if data port is busy
                  while(!DSK5416_PCM3002_write16(hCodec,   left_output));
                  while(!DSK5416_PCM3002_write16(hCodec, right_output));
         }
}
```

7. Add "**xyz.cdb**" to the current project.

Project→ Add files to project →xyz.cdb

8. Automatically three files are added in the **Generated file** folder in project pane
   xyzcfg.cmd→ Command and linking file
   xyzcfg.s62 → optimized assembly code for configuration
   xyzcfg_c.c → Chip support initialization

9. Open the **File** Menu ➔ **new** ➔ **Source file**
10. Type the code in editor window. Save the file in project folder. (Eg: **Codec.c**).

    Important note: Save your source code with preferred language extension. For ASM codes save the file as code(.asm) For C and C++ codes code (*.c,*.cpp) respectively.

11. Add the saved "**Fir.c**" file to the current project which has the main function and calls all the other necessary routines.
    Project → Add files to Project → Codec.c

12. Add the library file "**dsk5416f.lib**" to the current project

    Path → "**C:\CCStudio_v3.1\C5400\dsk5416\lib\dsk5416f.lib**"
    Files of type → Object and library files (*.o*, *.l*)

13. Copy header files **"dsk5416.h"** and "**dsk5416_pcm3002.h"** from and paste it in current project folder.

    C:\CCStudio_v3.1\C5400\dsk5416\include.

14. Add the header file generated within **xyzcfg_c.c** to **Fir.c**

15. Compile the program using the '**Project-compile**' pull down menu or by
Clicking the shortcut icon on the left side of program window.

SQUARE WAVE:

```
#include "hemucfg.h"
#include <c:\ccstudio_v3.1\c5400\dsk5416\include\dsk5416.h>
#include <c:\ccstudio_v3.1\c5400\dsk5416\include\dsk5416_pcm3002.h>
#include<math.h>
short left_output,right_output;
DSK5416_PCM3002_Config setup = {
   0x1ff,      // Set-Up Reg 0 - Left channel DAC attenuation
   0x1ff,      // Set-Up Reg 1 - Right channel DAC attenuation
   0x0,        // Set-Up Reg 2 - Various ctl e.g. power-down modes
   0x0         // Set-Up Reg 3 - Codec data format control
};


void main ()
{
          int count=0,step=32700,value=0;
   DSK5416_PCM3002_CodecHandle hCodec;
      // Initialize the board support library
   DSK5416_init();
   // Start the codec
   hCodec = DSK5416_PCM3002_openCodec(0, &setup);
   // Set codec frequency
   DSK5416_PCM3002_setFreq(hCodec, 48000);
          // Endless loop IO audio codec
          while(1){
                    count++;
     if(count==150)
     {
          count=0;
                    step = -step;
                    value= step;
     }
            left_output=value;
     right_output=value;
                    // Write 16 bits to the codec, loop to retry if data port is busy
                    while(!DSK5416_PCM3002_write16(hCodec,   left_output));
                    while(!DSK5416_PCM3002_write16(hCodec, right_output));
          }
}
```

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

17. Load the program (Codec.out) in program memory of DSP chip using the '**File-load program**' pull down menu.

18. Debug→ Run

Result:

Thus the signal using TMS320C5X has been generated.

PROGRAM:

```
#include "filtercfg.h"
#include <C:\CCStudio_v3.1\c5400\include\dsk5416.h>
#include <C:\CCStudio_v3.1\c5400\include\dsk5416_pcm3002.h>
Int16 left_input;
Int16   left_output;
Int16   right_input;
Int16 right_output;
static short in_buffer[100];
float filter_Coeff[] ={-0.000019,-0.000170,-0.000609,-0.001451,-0.002593,-0.003511,-
0.003150,0.000000,0.007551,0.020655,0.039383,0.062306,0.086494,0.108031,0.122944,0.128279,0.1
22944,0.108031,0.086494,0.062306,0.039383,0.020655,0.007551,0.000000,-0.003150,-0.003511,-
0.002593,-0.001451,-0.000609,-0.000170,-0.000019};
/*Select particular set of co-efficients different Cut-off frequencies from given Tables */
DSK5416_PCM3002_Config setup = {
 0x1ff,      // Set-Up Reg 0 - Left channel DAC attenuation
   0x1ff,      // Set-Up Reg 1 - Right channel DAC attenuation
   0x2,        // Set-Up Reg 2 - Various ctl e.g. power-down modes
   0x0         // Set-Up Reg 3 - Codec data format control
};

void main ()
{

   DSK5416_PCM3002_CodecHandle hCodec;
      // Initialize the board support library
   DSK5416_init();
   // Start the codec
   hCodec = DSK5416_PCM3002_openCodec(0, &setup);
   // Set codec frequency
   DSK5416_PCM3002_setFreq(hCodec,8000);

         // Endless loop IO audio codec
         while(1)
         {
                  // Read 16 bits of codec data, loop to retry if data port is busy
                  while(!DSK5416_PCM3002_read16(hCodec, &left_input));
                  while(!DSK5416_PCM3002_read16(hCodec, &right_input));
                           left_output=FIR_FILTER(&filter_Coeff ,left_input);
                          // right_output=left_output;
                  // Write 16 bits to the codec, loop to retry if data port is busy

while(!DSK5416_PCM3002_write16(hCodec, left_output));
                  //while(!DSK5416_PCM3002_write16(hCodec, left_output));
         }
}
signed int  FIR_FILTER(float * h, signed int x)
```

EXP.NO:10

DATE:

FIR FILTER DESIGN

AIM:

To design a FIR filter using TMS320C5X

APPARATUS REQUIRED:

System with TMS 320C50 software

TMS320C50 Processor kit

USB Cable

CRO

Function Generator

PROCEDURE:

1. Connect a Signal Generator/audio input to the **LINE IN** Socket or connect a microphone to the **MIC IN** Socket.
2. Connect CRO/Desktop Speakers to the Socket Provided for **LINE OUT or** connect a headphone to the **Headphone out** Socket.
3. Now Switch on the DSK and Bring Up Code Composer Studio on the PC.
4. Use the **Debug → Connect** menu option to open a debug connection to the DSK board
5. Create a new project with name **Fir.pjt.**
6. Open the **File** Menu **➔ new ➔ DSP/BIOS Configuration ➔**select "**dsk5416.cdb**" and save it as "**xyz.cdb**"
7. Add "**xyz.cdb**" to the current project.
   Project→ Add files to project →xyz.cdb
8. Automatically three files are added in the **Generated file** folder in project pane
   xyzcfg.cmd→ Command and linking file
   xyzcfg.s62 → optimized assembly code for configuration
   xyzcfg_c.c → Chip support initialization

```
{
int i=0;
signed long output=0;
in_buffer[0] = x;  /* new input at buffer[0]  */
for(i=30;i>0;i--)
in_buffer[i] = in_buffer[i-1]; /* shuffle the buffer  */
for(i=0;i<31;i++)
output = output + h[i] * in_buffer[i];
return(output);

}
```

9. Open the **File** Menu ➔ **new** ➔ **Source file**
10. Type the code in editor window. Save the file in project folder. (Eg: **Codec.c**).

    Important note: Save your source code with preferred language extension. For ASM codes save the file as code(.asm) For C and C++ codes code (*.c,*.cpp) respectively.
11. Add the saved "**Fir.c**" file to the current project which has the main function and calls all the other necessary routines.

    Project → Add files to Project → Codec.c

12. Add the library file "**dsk5416f.lib**" to the current project

    Path → "**C:\CCStudio_v3.1\C5400\dsk5416\lib\dsk5416f.lib**"

    Files of type → Object and library files (*.o*, *.l*)

13. Copy header files **"dsk5416.h"** and "**dsk5416_pcm3002.h"** from and paste it in current project folder.

    C:\CCStudio_v3.1\C5400\dsk5416\include.
14. Add the header file generated within **xyzcfg_c.c** to **Fir.c**
15. Compile the program using the '**Project-compile**' pull down menu or by Clicking the shortcut icon on the left side of program window.
16. Build the program using the '**Project-Build**' pull down menu or by clicking the shortcut icon on the left side of program window.
17. Load the program (Codec.out) in program memory of DSP chip using the '**File-load program**' pull down menu.

18. Debug→ Run

Result: Thus the FIR filter has been designed by using TMS320C50

PROGRAM

```
#include "filtercfg.h"
#include <dsk5416.h>
#include <dsk5416_pcm3002.h>
Int16 left_input;
Int16 left_output;

Int16 right_input;
Int16 right_output;
const signed int filter_Coeff[] =
{
    //48,48,48, 32767, -30949, 29322
    //96,96,96,32767,-30918,29614  /*fc=600*/
    //188,259,188,32767,-29498,27695/*fc=1000*/
    //11617,11617,11617,32767,8683,15505/*fc=8000*/
    //1455,1455,1455,32767,-23140,21735/*fc=2500*/
    5058,-5058,5058,32767,9995,15803/*fc=7000 -HP */
    //22586,-22586,22586,32767,-20964,15649
    //20539,-20539,20539,32767,-18173,13046
    //15241,-15241,15241,32767,-10161,7877

};

/*Select particular set of co-efficients different Cut-off frequencies from given Tables */

DSK5416_PCM3002_Config setup = {

   0x1ff,      // Set-Up Reg 0 - Left channel DAC attenuation
   0x1ff,      // Set-Up Reg 1 - Right channel DAC attenuation
   0x2,        // Set-Up Reg 2 - Various ctl e.g. power-down modes
   0x0         // Set-Up Reg 3 - Codec data format control
};
void main ()
{
   DSK5416_PCM3002_CodecHandle hCodec;
      // Initialize the board support library
   DSK5416_init();
   // Start the codec
   hCodec = DSK5416_PCM3002_openCodec(0, &setup);
   // Set codec frequency
   DSK5416_PCM3002_setFreq(hCodec,24000);
         // Endless loop IO audio codec
         while(1)
         {
                  // Read 16 bits of codec data, loop to retry if data port is busy
```

Ex.No: 11
Date :

IIR FILTER DESIGN

AIM:

To design a IIR filter using TMS320C5X

APPARATUS REQUIRED:

Code Composer Studio V3.1

TMS320C50 Processor kit

USB Cable

CRO

Function Generator

PROCEDURE:

1. Connect a Signal Generator/audio input to the **LINE IN** Socket or connect a microphone to the **MIC IN** Socket.
2. Connect CRO/Desktop Speakers to the Socket Provided for **LINE OUT or** connect a headphone to the **Headphone out** Socket.
3. Now Switch on the DSK and Bring Up Code Composer Studio on the PC.
4. Use the **Debug → Connect** menu option to open a debug connection to the DSK board
5. Create a new project with name **Fir.pjt.**
6. Open the **File** Menu **➔ new ➔ DSP/BIOS Configuration ➔**select "**dsk5416.cdb**" and save it as "**xyz.cdb**"
7. Add "**xyz.cdb**" to the current project.

Project→ Add files to project →xyz.cdb

```
while(!DSK5416_PCM3002_read16(hCodec,                    &left_input));
                while(!DSK5416_PCM3002_read16(hCodec, &right_input));
                        left_output=IIR_FILTER(&filter_Coeff ,left_input);
                        right_output=left_output;
                // Write 16 bits to the codec, loop to retry if data port is busy
                while(!DSK5416_PCM3002_write16(hCodec,   left_output));
                while(!DSK5416_PCM3002_write16(hCodec, right_output));
        }
}

signed int IIR_FILTER(const signed int * h, signed int x1)
{
static signed int x[6] = { 0, 0, 0, 0, 0, 0 };  /* x(n), x(n-1), x(n-2). Must be static */
        static signed int y[6] = { 0, 0, 0, 0, 0, 0 };  /* y(n), y(n-1), y(n-2). Must be static */
   long temp=0;

  temp = x1; /* Copy input to temp */

  x[0] = (signed int) temp; /* Copy input to x[stages][0] */

  temp =  ( (long)h[0] * x[0]) ;   /* B0 * x(n)   */

    temp += ( (long)h[1] * x[1]);    /* B1/2 * x(n-1) */

    temp += ( (long)h[1] * x[1]);    /* B1/2 * x(n-1) */

    temp += ( (long)h[2] * x[2]);    /* B2 * x(n-2) */

    temp -= ( (long)h[4] * y[1]);    /* A1/2 * y(n-1) */

    temp -= ( (long)h[4] * y[1]);   /* A1/2 * y(n-1) */

    temp -= ( (long)h[5] * y[2]);   /* A2 * y(n-2) */

    /* Divide temp by coefficients[A0] */

    temp >>= 15;

        if ( temp > 32767 )
     {
  temp = 32767;
     }
    else if ( temp < -32767)
     {
       temp = -32767;
     }
```

8. Automatically three files are added in the **Generated file** folder in project pane

   xyzcfg.cmd→ Command and linking file

   xyzcfg.s62 → optimized assembly code for configuration

   xyzcfg_c.c → Chip support initialization

9. Open the **File** Menu ➔ **new** ➔ **Source file**
10. Type the code in editor window. Save the file in project folder. (Eg: **Codec.c**).

    Important note: Save your source code with preferred language extension. For ASM codes save the file as code(.asm) For C and C++ codes code (*.c,*.cpp) respectively.

11. Add the saved "**Fir.c**" file to the current project which has the main function and calls all the other necessary routines.

    Project → Add files to Project → Codec.c

12. Add the library file "**dsk5416f.lib**" to the current project

    Path → "**C:\CCStudio_v3.1\C5400\dsk5416\lib\dsk5416f.lib**"

    Files of type → Object and library files (*.o*, *.l*)

13. Copy header files **"dsk5416.h"** and "**dsk5416_pcm3002.h"** from and paste it in current project folder.

    C:\CCStudio_v3.1\C5400\dsk5416\include.

14. Add the header file generated within **xyzcfg_c.c** to **Fir.c**

15. Compile the program using the '**Project-compile**' pull down menu or by Clicking the shortcut icon on the left side of program window.

```
 y[0] = (short int) ( temp );

    /* Shuffle values along one place for next time */

    y[2] = y[1];  /* y(n-2) = y(n-1) */
    y[1] = y[0];  /* y(n-1) = y(n)   */

    x[2] = x[1];  /* x(n-2) = x(n-1) */
    x[1] = x[0];  /* x(n-1) = x(n)   */

    /* temp is used as input next time through */

  return ((short int)temp*1);
}
```

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

17. Load the program (Codec.out) in program memory of DSP chip using the '**File-load program**' pull down menu.

18. **Debug→ Run**

Result:

Thus the IIR filter has been designed successfully by using TMS320C50X

PROGRAM:

a) Illustration of up-sampling by an integer factor

```
function[y]=up(x)
x=input('Enter X[n]:');
k = input('Enter upsampling factor = ')
y=zeros(1,k*length(x));
y([1:k:length(y)])=x;
end
```

b) Illustration of down-sampling by an integer factor

```
function[y]= dns(x)
x=input('Enter X[n]:');
k = input('Enter downsampling factor = ')
y= x(1:k:length(x));end
```

EXP.NO:12

DATE:

IMPLEMENT AN UP-SAMPLING AND DOWN SAMPLING OPERATION

AIM: To write a program to implement an up-sampling and down sampling operation

PROGRAM LOGIC:

Get the input sequence.

Get the filter coefficients and also the decimation and interpolation factor.

Plot the graph.

c) Multirate Filters using Decimation:

```
Fs = 4.8e4;
t =0:1/Fs:1-(1/Fs);
x = cos(2*pi*4000*t);
Hm=mfilt.firdecim(2);
% Note cutoff frequency of 1/2 normalized frequency
fvtool(Hm);
% Note the group delay of 34 samples
fvtool(Hm,'analysis','grpdelay');
y = filter(Hm,x);
% Note delay in output is consistent with 36/2
stem(y(1:48),'markerfacecolor',[0 0 1]);
```

d) Multirate Filters using interpolation

```
Fs = 4.8e4;
t =0:1/Fs:1-(1/Fs);
x = cos(2*pi*4000*t);
Hm=mfilt.firinterp(2);
% Note cutoff frequency of 1/2 normalized frequency
fvtool(Hm);
% Note the group delay of 34 samples
fvtool(Hm,'analysis','grpdelay');
y = filter(Hm,x);
% Note delay in output is consistent with 36/2
stem(y(1:48),'markerfacecolor',[0 0 1]);
```

OUTPUT :

Input

Upsampling factor

Output=

Down sampling factor

Output=

RESULT :

Thus we implented multirate filters using upsampling and down sampling successfully.

PROGRAM:

```
/* prg to implement linear convolution */
#include<stdio.h>
int y[20];
main()
{       int m=6;                                        /*Lenght of i/p samples sequence*/
        int n=6;                                        /*Lenght of impulse response Co-efficients
*/
        int i=0,j;
        int x[15]={1,1,1,2,1,0,0,0,0,0,0};      /*Input Signal Samples*/
        int h[15]={1,1,2,1,0,0,0,0,0,0,0};      /*Impulse Response Co-efficients*/
        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("%d\n",y[i]);
}
```

EXP.NO:13a

DATE:

IMPLEMENTATION OF LINEAR CONVOLUTION

AIM:

To write a program to find the Convolution of two sequences using TMS320C50

APPARATUS REQUIRED:

System with TMS 320C50 software

PROCEDURE:

- Open Code Composer Studio, make sure the DSP kit is turned on.
- Use the **Debug → Connect** menu option to open a debug connection to the DSK board
- Start a new project using 'Project-new ' pull down menu, save it in a separate directory(C:\CCStudio_v3.1\myprojects) with name
  Add the source files to the project using 'Project→add files to project' pull down menu.
- Add the linker command file **hello.cmd** .
  (Path: C:\CCStudio_v3.1\tutorial\sim54xx\hello1\hello.cmd)
- Add the run time support library file **rts_EXT.lib**
  (Path: C:\CCStudio_v3.1\c5400\cgtools\lib\rts_EXT.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**.

Output :

Linear Convolution :

Enter the 1st sequence :

Enter the 2nd sequence :

The resultant Signal is : y =

RESULT: Thus the linear convolution of two sequences is done successfully by using TMS320C50.

PROGRAM:

```
/* prg to implement Circular Convolution %
 #include<stdio.h>
 #include<math.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];
```

EXP.NO:13b

DATE:

IMPLEMENTATION OF CIRCULAR CONVOLUTION

AIM:

To write a program to find the Circular Convolution of two sequences using TMS320C50

APPARATUS REQUIRED:

System with TMS 320C50 software

PROCEDURE:

- Open Code Composer Studio, make sure the DSP kit is turned on.
- Use the **Debug → Connect** menu option to open a debug connection to the DSK board
- Start a new project using 'Project-new ' pull down menu, save it in a separate directory(C:\CCStudio_v3.1\myprojects) with name
  Add the source files to the project using 'Project→add files to project' pull down menu.
- Add the linker command file **hello.cmd** .
  (Path: C:\CCStudio_v3.1\tutorial\sim54xx\hello1\hello.cmd)
- Add the run time support library file **rts_EXT.lib**
  (Path: C:\CCStudio_v3.1\c5400\cgtools\lib\rts_EXT.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**.

```
 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:

Circular Convolution :

Enter the 1st sequence :

Enter the 2nd sequence :

The resultant Signal is : y =

RESULT: Thus the circular convolution of two sequences is done successfully by using TMS320C50.