

Lab manual for

Digital Signal Processing Lab

IV B. Tech I Semester



Prepared by

J. Sunil Kumar
&
P. Saritha

Department of Electronics & Communication Engineering

Turbomachinery Institute of Technology & Sciences

(Approved by AICTE & Affiliated to JNTUH)

Indresam(v), Patancheru(M), Medak(Dist). Pin: 502 319

Turbomachinery Institute of Technology & Sciences

Certificate

*This is to certify that Mr. / Ms. RollNo..... of I/II/III/IV
B.Tech I / II Semester ofbranch has completed the laboratory work
satisfactorily in Lab for the academic year 20 ... to 20 ...as prescribed in the
curriculum.*

Place:

Date:

Lab In charge

Head of the Department

Principal

LIST OF EXPERIMENTS:

1. To study the architecture of DSP chips – TMS 320 6713.32-bit floating point processor.
2. Introduction to MATLAB: generation of basic signals
3. To verify linear convolution
4. To verify the circular convolution
5. To design FIR filter (LP/HP) using windowing technique a. Using rectangular window
b. Using triangular window c. Using Kaiser window
6. To implement IIR filter (LP/HP) & frequency response of analog LP/HP and butter worth filters & cheby shev filters.
7. N-point FFT algorithm
8. MATLAB program to generate sum of sinusoidal signals
9. To compute power density spectrum of a sequence
10. To find the FFT of given 1-D signal and plot.

EXPERIMENTS OTHER THAN JNTU SYLLABUS:

11. Auto correlation and cross correlation of given sequences.
12. Impulse response.

1. 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.

ARCHITECTURE OF 6713 DSP PROCESSOR

This chapter provides an overview of the architectural structure of the TMS320C67xx DSP, which comprises the central processing unit (CPU), memory, and on-chip peripherals. The C67xE DSPs use an advanced modified Harvard architecture that maximizes processing power with eight buses. Separate program and data spaces allow simultaneous access to program instructions and data, providing a high degree of parallelism. For example, three reads and one write can be performed in a single cycle. Instructions with parallel store and application-specific instructions 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, the C67xx DSP includes the control mechanisms to manage interrupts, repeated operations, and function calling.

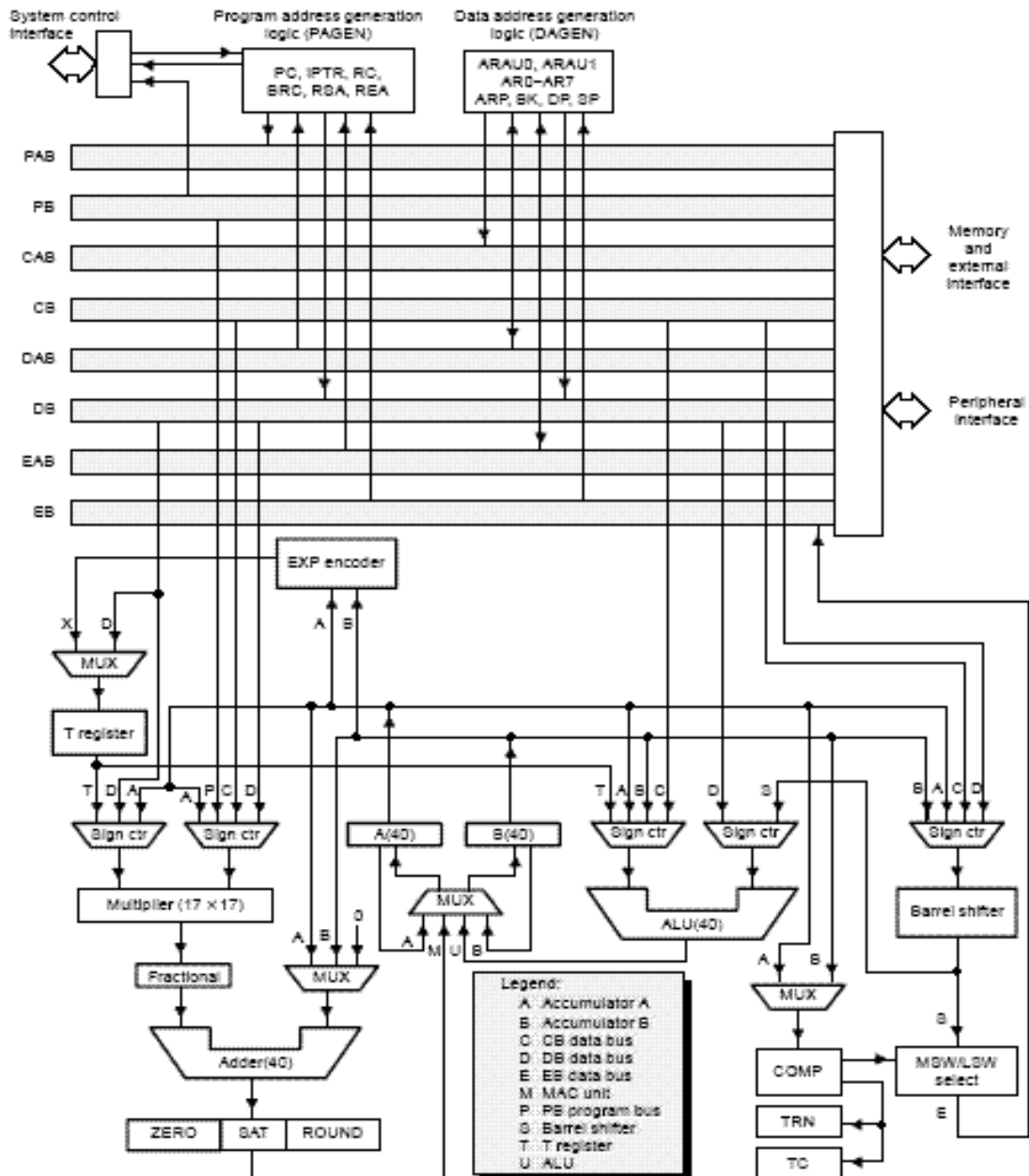


Fig 2 – 1 BLOCK DIAGRAM OF TMS 320VC 6713

Bus Structure

The C67xx DSP architecture is built around eight major 16-bit buses (four program/data buses and four address buses):

- _ The program bus (PB) carries the instruction code and immediate operands from program

memory.

- _ Three data buses (CB, DB, and EB) interconnect to various elements, such as the CPU, data address generation logic, program address generation logic, on-chip peripherals, and data memory.
- _ The CB and DB carry the operands that are read from data memory.
- _ The EB carries the data to be written to memory.
- _ Four address buses (PAB, CAB, DAB, and EAB) carry the addresses needed for instruction execution.

The C67xx DSP can generate up to two data-memory addresses per cycle using the two auxiliary register arithmetic units (ARAU0 and ARAU1). The PB can carry data operands stored in program space (for instance, a coefficient table) to the multiplier and adder for multiply/accumulate operations or to a destination in data space for data move instructions (MVPD and READA). This capability, in conjunction with the feature of dual-operand read, supports the execution of single-cycle, 3-operand instructions such as the FIRS instruction. The C67xx DSP also has an on-chip bidirectional bus for accessing on-chip peripherals. This bus is connected to DB and EB through the bus exchanger in the CPU interface. Accesses that use this bus can require two or more cycles for reads and writes, depending on the peripheral's structure.

Central Processing Unit (CPU)

The CPU is common to all C67xE devices. The C67x CPU contains:

- _ 40-bit arithmetic logic unit (ALU)
- _ Two 40-bit accumulators
- _ Barrel shifter
- _ 17×17 -bit multiplier
- _ 40-bit adder
- _ Compare, select, and store unit (CSSU)
- _ Data address generation unit
- _ Program address generation unit

Arithmetic Logic Unit (ALU)

The C67x DSP performs 2s-complement arithmetic with a 40-bit arithmetic logic unit (ALU) and two 40-bit accumulators (accumulators A and B). The ALU can also perform Boolean operations. The ALU uses these inputs:

- _ 16-bit immediate value
- _ 16-bit word from data memory
- _ 16-bit value in the temporary register,
- _ Two 16-bit words from data memory
- _ 32-bit word from data memory
- _ 40-bit word from either accumulator

The ALU can also function as two 16-bit ALUs and perform two 16-bit operations simultaneously.

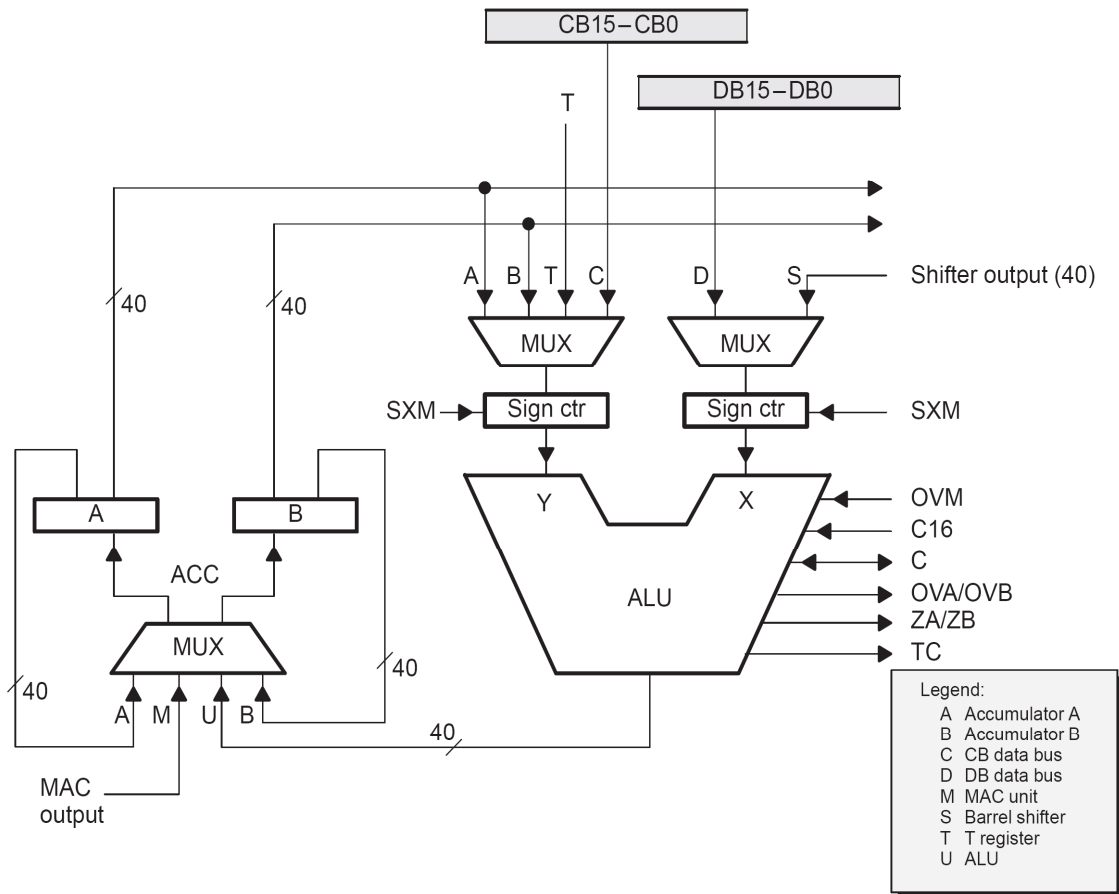


Fig 2 – 2 ALU UNIT

Accumulators

Accumulators A and B store the output from the ALU or the multiplier/adder block. They can also provide a second input to the ALU; accumulator A can be an input to the multiplier/adder. Each accumulator is divided into three parts:

- Guard bits (bits 39–32)
- High-order word (bits 31–16)
- Low-order word (bits 15–0)

Instructions are provided for storing the guard bits, for storing the high- and the low- order accumulator words in data memory, and for transferring 32-bit accumulator storage words in or out of data memory. Also, either of the accumulators can be used as temporary storage for the other.

Barrel Shifter

The C67x DSP barrel shifter has a 40-bit input connected to the accumulators or to data memory (using CB or DB), and a 40-bit output connected to the ALU or to data memory (using EB). The barrel shifter can produce 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 of the instruction, the shift count field (ASM) of status register ST1, or in temporary register T (when it is designated as a shift count register).The barrel shifter and the exponent encoder normalize the values in an accumulator in a single cycle. The LSBs of the output are filled with 0s, and the MSBs can be either zero filled or sign extended, depending on the state of the sign-extension mode bit (SXM) in ST1. Additional shift capabilities enable the processor to perform numerical scaling, bit extraction, extended arithmetic, and overflow prevention operations.

Multiplier/Adder Unit

The multiplier/adder unit performs 17 _ 17-bit 2s-complement multiplication with a 40- bit addition in a single instruction cycle. The multiplier/adder block consists of several elements: a multiplier, an adder, signed/unsigned input control logic, fractional control logic, a zero detector, a rounder (2s complement), overflow/saturation logic, and a 16-bit temporary storage register (T).

The multiplier has two inputs: one input is selected from T, a data-memory operand, or accumulator A; the other is selected from program memory, data memory, accumulator A, or an immediate value. The fast, on-chip multiplier allows the C54x DSP to perform operations efficiently such as convolution, correlation, and filtering. 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 Euclidian distance and in implementing symmetrical and LMS filters, which are required for complex DSP algorithms. See section 4.5, Multiplier/Adder Unit, on page 4-19, for more details about the multiplier/adder unit.

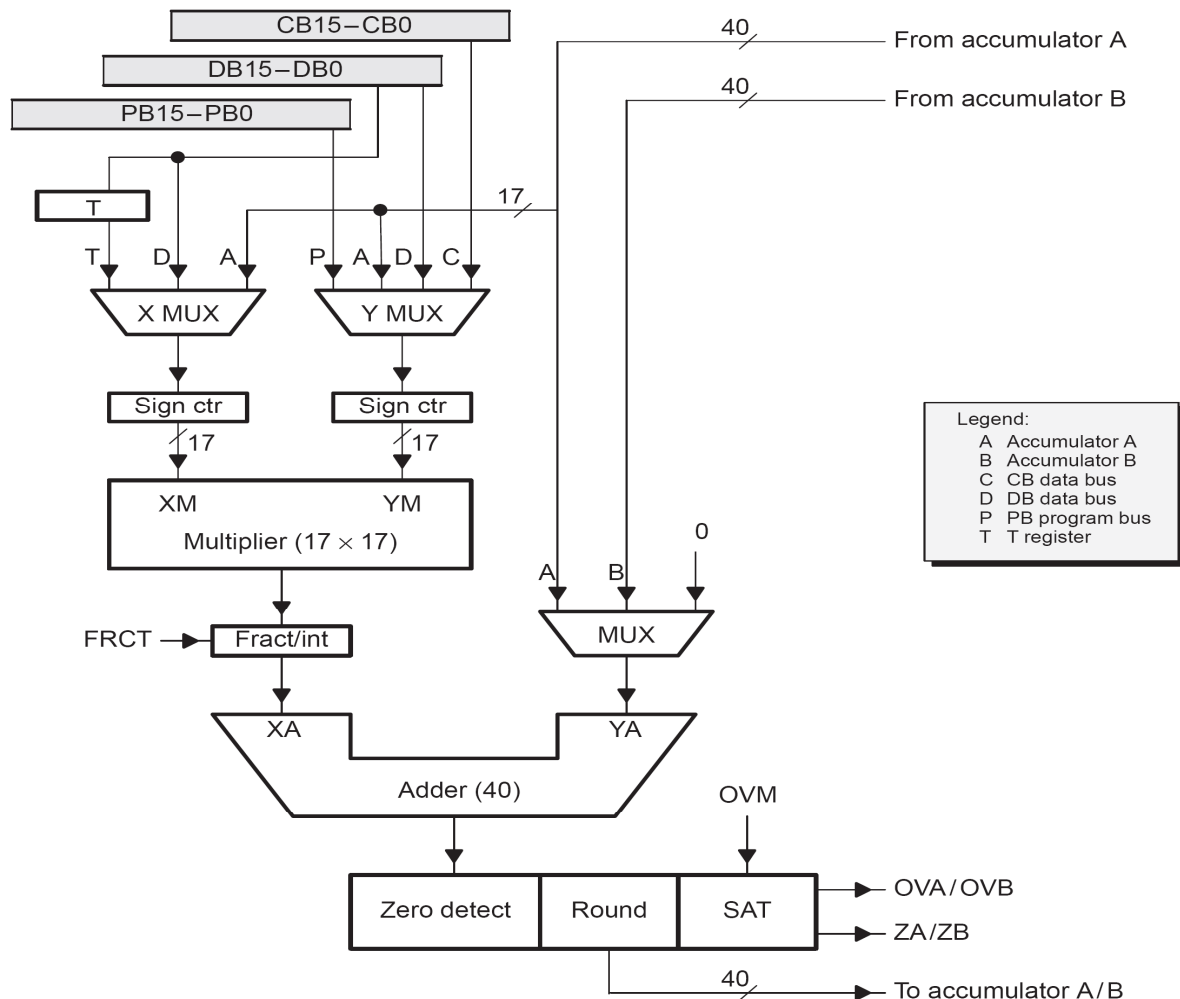


Fig 2 – 3 MULTIPLIER/ADDER UNIT

These are the some of the important parts of the processor and you are instructed to go through the detailed architecture once which helps you in developing the optimized code for the required application.

2. VERIFY LINEAR CONVOLUTION

LINEAR CONVOLUTION USING CCS

AIM: To perform linear convolution of two sequences

EQUIPMENT REQUIRED:

Operating System	– Windows XP
Constructor	- Simulator
Software	- CCStudio 3.3 & MATLAB 7.5

THEORY:

Convolution is a formal mathematical operation, just as multiplication, addition, and integration. Addition takes two *numbers* and produces a third *number*, while convolution takes two *signals* and produces a third *signal*. Convolution is used in the mathematics of many fields, such as probability and statistics. In linear systems, convolution is used to describe the relationship between three signals of interest: the input signal, the impulse response, and the output signal.

$$y(n) = \sum_{k=0}^{N-1} x_1(k)x_2(n-k) \quad 0 < n < N-1 \quad (1)$$

In this equation, $x_1(k)$, $x_2(n-k)$ and $y(n)$ represent the input to and output from the system at time n . Here we could see that one of the input is shifted in time by a value every time it is multiplied with the other input signal. Linear Convolution is quite often used as a method of implementing filters of various types.

PROCEDURE:

- 1) Generate the first input sequence 'x'.
- 2) Plot the sequence in discrete form. [Make use of stem()]
- 3) Give some relevant names to x-axis and y-axis.
- 4) Generate second input sequence 'y'.
- 5) Repeat steps (2) and (3) for second sequence 'y'.
- 6) Use the inbuilt function 'conv()' to compute linear convolution of 'x' and 'y'.

$$z = \text{conv}(x,y)$$
- 7) Plot the output sequence 'z'. [Repeat steps 2 and 3]
- 8) Make use of subplot () to plot the inputs and output sequences in a single window.

PROGRAM:

// Linear convolution program in c language using CCStudio

```
#include<stdio.h>
int x[15],h[15],y[15];
main()
{
int i,j,m,n;
printf("\n enter value for m");
scanf("%d",&m);
printf("\n enter value for n");
scanf("%d",&n);
printf("Enter values for i/p x(n):\n");
```

```

for(i=0;i<m;i++)
scanf("%d",&x[i]);
printf("Enter Values for i/p h(n) \n");
for(i=0;i<n; i++)
scanf("%d",&h[i]);

// padding of zeros
for(i=m;i<=m+n-1;i++) x[i]=0;
for(i=n;i<=m+n-1;i++)
h[i]=0;
/* convolution operation */
for(i=0;i<m+n-1;i++)
{
y[i]=0;
for(j=0;j<=i;j++)
{
y[i]=y[i]+(x[j]*h[i-j]);
}
}
//displaying the o/p
for(i=0;i<m+n-1;i++)
printf("\n The Value of output y[%d]=%d",i,y[i]);
}

```

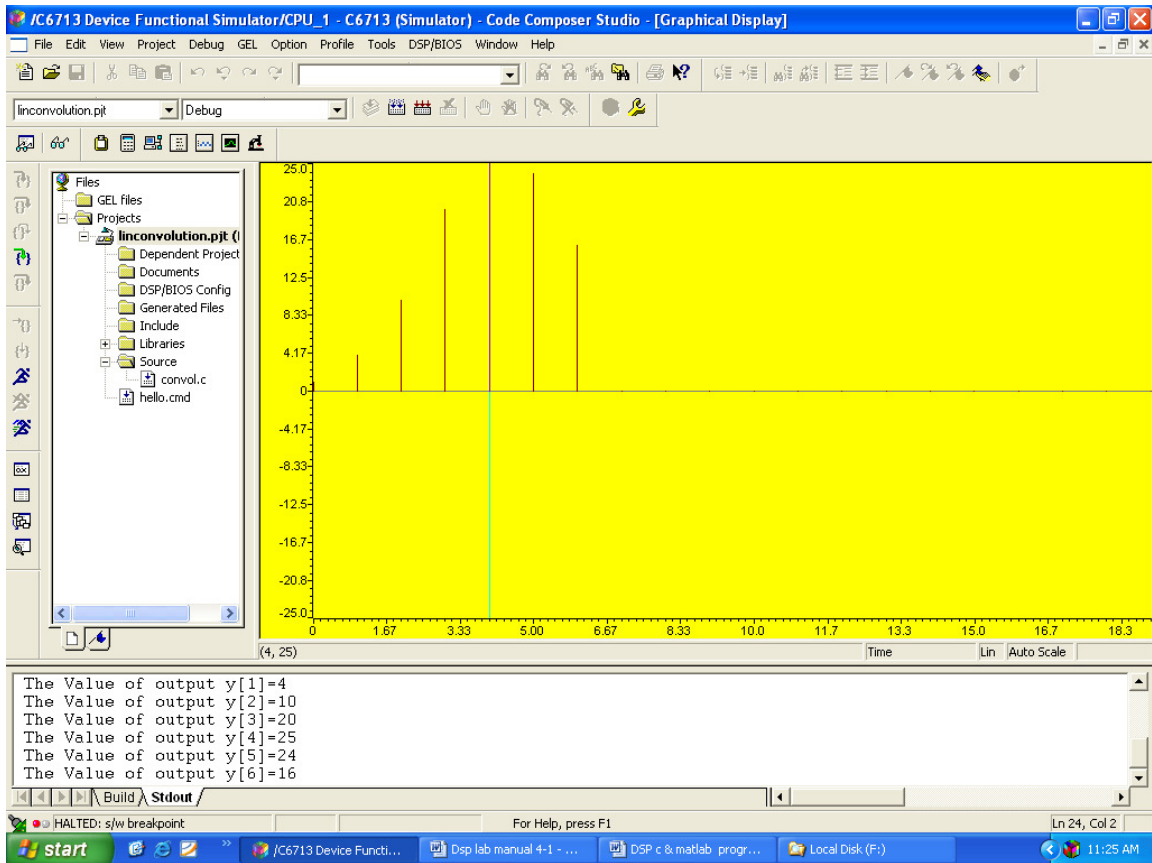
Result:

```

Enter value for m      4
Enter value for n      4
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

```



Result: linear convolution of two sequences using CCStudio 3.3 is obtained.

Viva Questions:

1. What is the purpose of using convolution?
2. Give the formula for calculating linear convolution.

Exercise Questions:

1. Write a program to find Discrete Fourier Transform of any Two Sequences
2. Write a program to find Inverse Discrete Fourier Transform of any Two Sequences
3. Write a program to find FFT of a signal

PROGRAM USING CODE COMPOSER STUDIO 3.3

```

/* program to implement circular convolution */
#include<stdio.h>
int m,n,x[30],h[30],y[30],i,j, 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]);
}

```

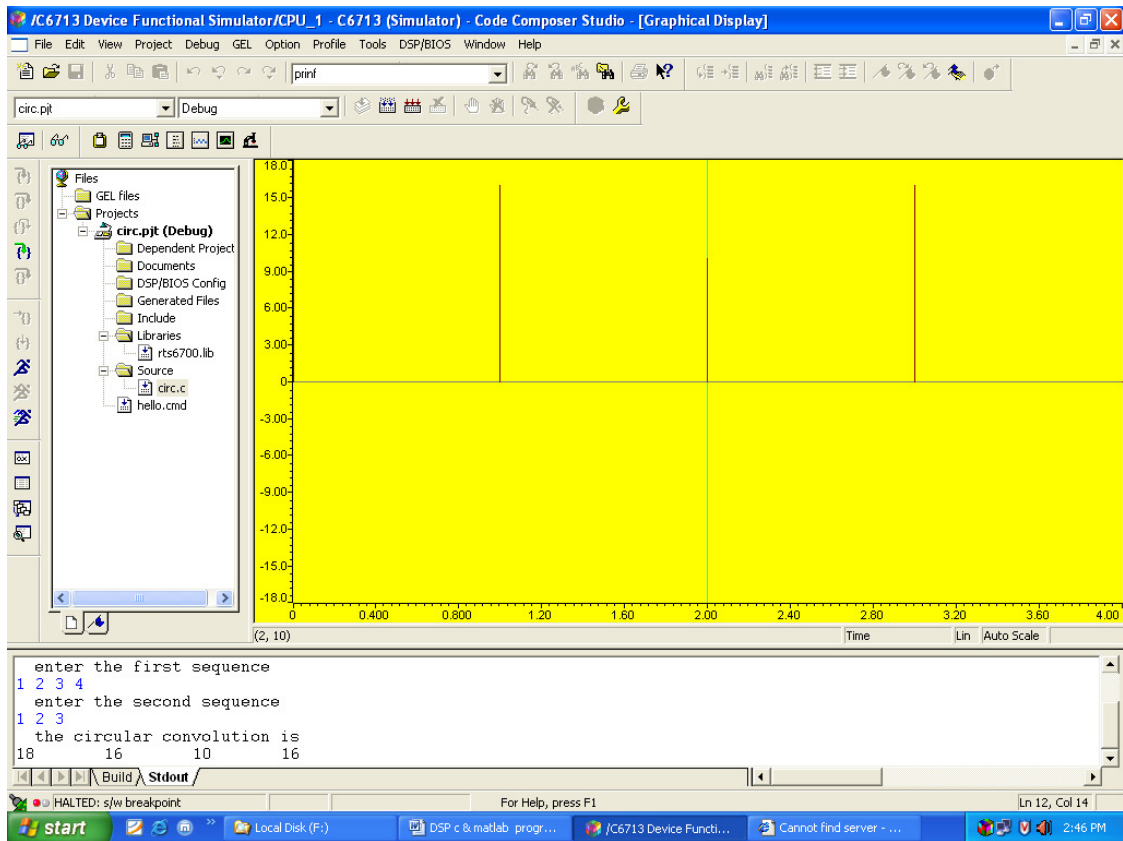
OUTPUT:-

```

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

```

MODEL GRAPH:-



RESULT: Circular Convolution of two sequences using CC Studio3.3 is obtained.

VIVA QUESTIONS:

1. What is the different between Circular and Linear convolution?
2. What is the function in MATLAB used for padding zeros to a sequence? If your sequence is, $x = [1\ 2\ 3\ 4]$ and you want to pad zeros to it. How can you do that in MATLAB?
3. What is the use of following functions in MATLAB:
 - i. length()
 - ii. max()
 - iii. min()
4. Give the steps to get the result of linear convolution from the method of circular convolution
5. What is the circular convolution
6. Explain how cyclic (circular) convolution of two periodic sequences can be obtained using DFT techniques

EXERCISE QUESTIONS:

1. Write a program to sampling
2. Write a program to Up sampling & Down sampling

4. DESIGN FIR FILTER (LP/HP) USING WINDOWING TECHNIQUE

FIR FILTERS (LP/HP) USING WINDOWING TECHNIQUE

a) Using Rectangular Window b) Using Bartlett Window c) Using Kaiser Window

AIM: To design of FIR(LP/HP) filters using rectangular window.

EQUIPMENT REQUIRED:

Operating System – Windows XP Constructor
 – Simulator
 Software – CCStudio 3 & MATLAB 7.5

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.

$$y(n) = \sum_{k=0}^{N-1} x(k)h(n-k) \quad 0 < n < N-1 \tag{1}$$

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. Enter the passband ripple (rp) and stopband ripple (rs).
2. Enter the passband frequency (fp) and stopband frequency (fs).
3. Enter the sampling frequency (f).
4. Calculate the analog passband edge frequency (wp) and stop band edge frequency (ws)
 $wp=2*fp/f$ $ws=2*fs/f$
5. Calculate the order of the filter using the following formula,
 $(-20\log_{10} (rp.rs) -13)$
 $n= \frac{\hspace{10em}}{(14.6 (fs-fp)/f)}$
 [Use 'ceil()' for rounding off the value of 'n' to the nearest integer] if 'n' is an odd number, then reduce its value by '1'.
6. Generate (n+1)th point window coefficients. For example boxcar(n+1) generates a rectangular window. $y=boxcar(n+1)$
7. Design an nth order FIR filter using the previously generated (n+1) length window function.
 $b=fir1(n,wp,y)$
8. Find the frequency response of the filter by using 'freqz()' function. $[h,o]=freqz(b,a,k)$ This function returns k-point complex frequency response vector 'h' and k-point frequency vector 'o' in radians/samples of the filter.

$$H(e^{jw})= \underline{B(e^{jw})} = \underline{b(1)+b(2)e^{-jw}+.....+b(m+1)e^{-jmw}}$$

$$A(e^{j\omega}) = a(1) + a(2)e^{-j\omega} + \dots + a(n+1)e^{-jn\omega}$$

Where a, b are vectors containing the denominator and numerator coefficients.

Here a=1.

9. Calculate the magnitude of the frequency response in decibels (dB). $m = 20 \cdot \log_{10}(\text{abs}(h))$
10. Plot the magnitude response [magnitude in dB Vs normalized frequency (ω/π)]
11. Give relevant names to x- and y- axes and give an appropriate title for the plot.
12. Repeat the procedure for high pass, band pass, and band stop filters.
13. Plot all the responses in a single figure window.[Make use of Subplot].
14. Repeat the program for different types of windows available:
 - a. Rectangular window
 - b. Triangular window
 - c. Kaiser window

PROGRAM USING CODE COMPOSER STUDIO 3.3:

```
#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;
}
}
```

RESULT: By using windowing techniques (Bartlett, Rectangular, Blackman) the filters are designed

VIVA QUESTIONS:

1. What are the uses of function ceil and for?
2. Define boxcar
3. Define Kaiser
4. Define Bartlett
5. What is an FIR system? Compare FIR and IIR system?

$$a(s) = \frac{a(1)S^{na-1} + a(2)S^{na-2} + \dots + a(na)}{b(s)}$$

Where a,b are the vectors containing the denominator and numerator co-efficients.

8. Calculate the magnitude of the frequency response in decibels (dB)
 $m = 20 * \log_{10}(\text{abs}(h))$
9. Plot the magnitude response [magnitude in dB Vs normalized frequency (om/pi)]
10. Give relevant names to x and y axes and give an appropriate title for the plot.
11. Repeat the procedure for highpass, bandpass, bandstop filters.
12. Plot all the responses in a single figure window.[Make use of subplot()].
13. Repeat the program for Chebyshev type-I and Chebyshev type-II filters.

PROGRAM USING CODE COMPOSER STUDIO 3.3:

```
//iirfilters
#include<stdio.h>
#include<math.h>
int i,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);
}

```

RESULT :

