Chettinad College of Engineering & Technology

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERNG

III YEAR / VI SEM

CEC337 – DSP ARCHITECTURE AND PROGRAMMING LAB MANUAL

CHETTINAD COLLEGE OF ENGINEERING & TECHNOLOGY, KARUR

Department of Electronics and Communication Engineering

Vision and Mission of the Institution and Department

Vision of the Institution:

1. To holistically develop competent and responsible Engineers and Managers as future leaders by providing an enriching, safe and joyful learning environment where students feel empowered.

Mission of the Institution:

1. To impart knowledge and the skills through active learning, industrial exposure and innovative project development.
2. To develop leaders through effective mentoring, SMART goal setting and providing a joyful and safe learning environment.
3. To facilitate research in Engineering and Technology and encourage independent learning.

Vision of the Department:

1. To provide the quality education in the field of Electronics and Communication Engineering which caters the needs of the society in line with the technological revolution.

Mission of the Department:

1. To upgrade the technical knowledge of the students continuously by providing industrial exposure and innovative projects.
2. To establish a creative learning environment for the students by active learning of the techniques in the electronics and communication engineering field.
3. To nurture career improvement by facilitating skill development and training in the recent technologies.

I. PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

Graduates can

- Gain adequate knowledge to become good professional in electronic and communication engineering associated industries, higher education and research.
- Develop attitude in lifelong learning, applying and adapting new ideas and technologies as their field evolves.
- 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.

II. PROGRAM OUTCOMES (POs)

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.

III. PROGRAM SPECIFIC OUTCOMES (PSOs)

The Students will be able to

- Design, develop and analyze electronic systems through application of relevant electronics, mathematics and engineering principles
- Design, develop and analyze communication systems through application of fundamentals from communication principles, signal processing, and RF System Design & Electromagnetics.
- Adapt to emerging electronics and communication technologies and develop innovative solutions for existing and newer problems

CEC337 DSP ARCHITECTURE AND PROGRAMMING L T P C
2 0 2 3

COURSE OBJECTIVES:

- Study the architecture of programmable DSP processors
- Learn to implement various standard DSP algorithms in DSP Processors
- Use the Programmable DSP Processors to build real-time DSP systems

UNIT I ARCHITECTURES FOR PROGRAMMABLE DSP PROCESSORS 6

Basic Architectural features, DSP Computational building blocks, Bus architecture and memory, Data addressing capabilities, Address generation Unit, Programmability and program execution, Speed issues, Features for external interfacing.

UNIT II TMS320C5X PROGRAMMABLE DSP PROCESSOR 6

Architecture of TMS320C54xx DSP processors, Addressing modes – Assembly language Instructions -Memory space, interrupts, and pipeline operation of TMS320C54xx DSP Processor, On-Chip peripherals, Block Diagram of TMS320C54xx DSP starter kit

UNIT III TMS320C6X PROGRAMMABLE DSP PROCESSOR 6

Commercial TI DSP processors, Architecture of TMS320C6x DSP Processor, Linear and Circular addressing modes, TMS320C6x Instruction Set, Assembler directives, Linear Assembly, Interrupts, Multichannel buffered serial ports, Block diagram of TMS320C67xx DSP Starter Kit and Support Tools

UNIT IV IMPLEMENTATION OF DSP ALGORITHMS 6

DSP Development system, On-chip, and On-board peripherals of C54xx and C67xx DSP development boards, Code Composer Studio (CCS) and support files, Implementation of Conventional FIR, IIR, and Adaptive filters in TMS320C54xx/TMS320C67xx DSP processors for real-time DSP applications, Implementation of FFT algorithm for frequency analysis in real-time.

UNIT V APPLICATIONS OF DSP PROCESSORS 6

Voice scrambling using filtering and modulation, Voice detection and reverse playback, Audio effects, Graphic Equalizer, Adaptive noise cancellation, DTMF signal detection, Speech thesis using LPC, Automatic speaker recognition

30 PERIODS

PRACTICAL EXERCISES: 30 PERIODS

1. Real-Time Sine Wave Generation
2. Programming examples using C, Assembly and linear assembly
3. Implementation of moving average filter
4. FIR implementation with a Pseudorandom noise sequence as input to a filter
5. Fixed point implementation of IIR filter
6. FFT of Real-Time input signal

HARDWARE & SOFTWARE SUPPORT TOOLS

- TMS320C54xx/TMS320C67xx DSP Development board
- Code Composer Studio (CCS)
- Function Generator and Digital Storage Oscilloscope
- Microphone and speaker

TOTAL: 60 PERIODS

COURSE OUTCOMES:

After the course, the student should have

CO	Course Outcomes	POs	PSOs
C311.1	Understand the architectural features of DSP Processors.	1,2,3,4,5,6,10,12	1,2,3
C311.2	Comprehend the organization of TMS320C54xx DSP processors	1,2,3,4,5,6,10,12	1,2,3
C311.3	Build solutions using TMS320C6x DSP Processor	1,2,3,4,5,6,10,12	1,2,3
C311.4	Implement DSP Algorithms	1,2,3,4,5,6,10,12	1,2,3
C311.5	Study the applications of DSP Processors.	1,2,3,4,5,6,10,12	1,2,3

CO's-PO's & PSO's MAPPING

Course	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
C311.1	3	3	3	2	2	2	-	-	-	1	-	3	3	3	3
C311.2	3	3	2	2	2	2	-	-	-	1	-	2	3	3	3
C311.3	3	3	3	2	2	2	-	-	-	1	-	2	2	2	2
C311.4	3	3	3	3	2	2	-	-	-	1	-	2	2	3	2
C311.5	3	3	3	2	2	2	-	-	-	1	-	2	2	3	2
C311	3	3	2.8	2.2	2	2	-	-	-	1	-	2.2	2.4	2.8	2.4

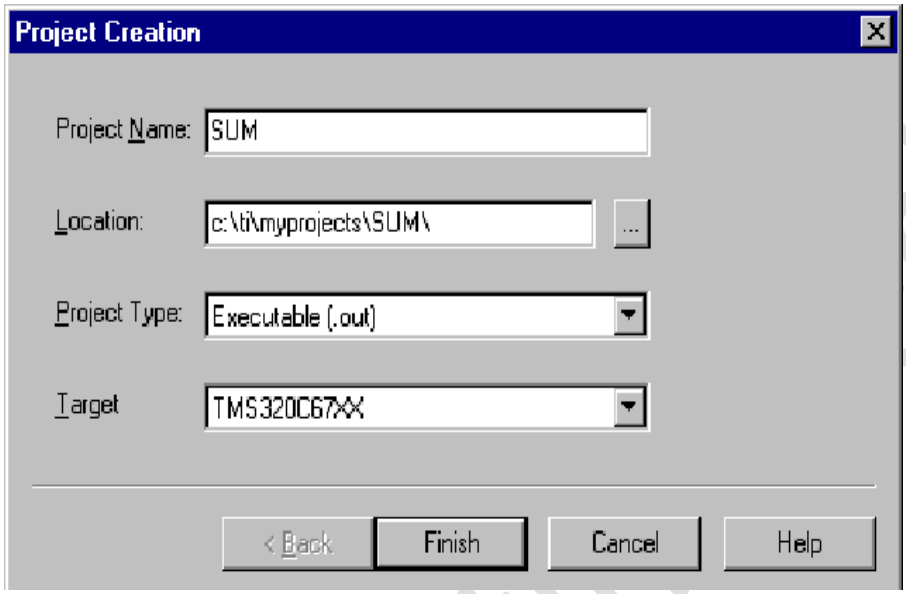
1 - Low, 2 - Medium, 3 - High, '-' -No Correlation

TABLE OF CONTENTS

PROCEDURE TO WORK ON CODE COMPOSER STUDIO

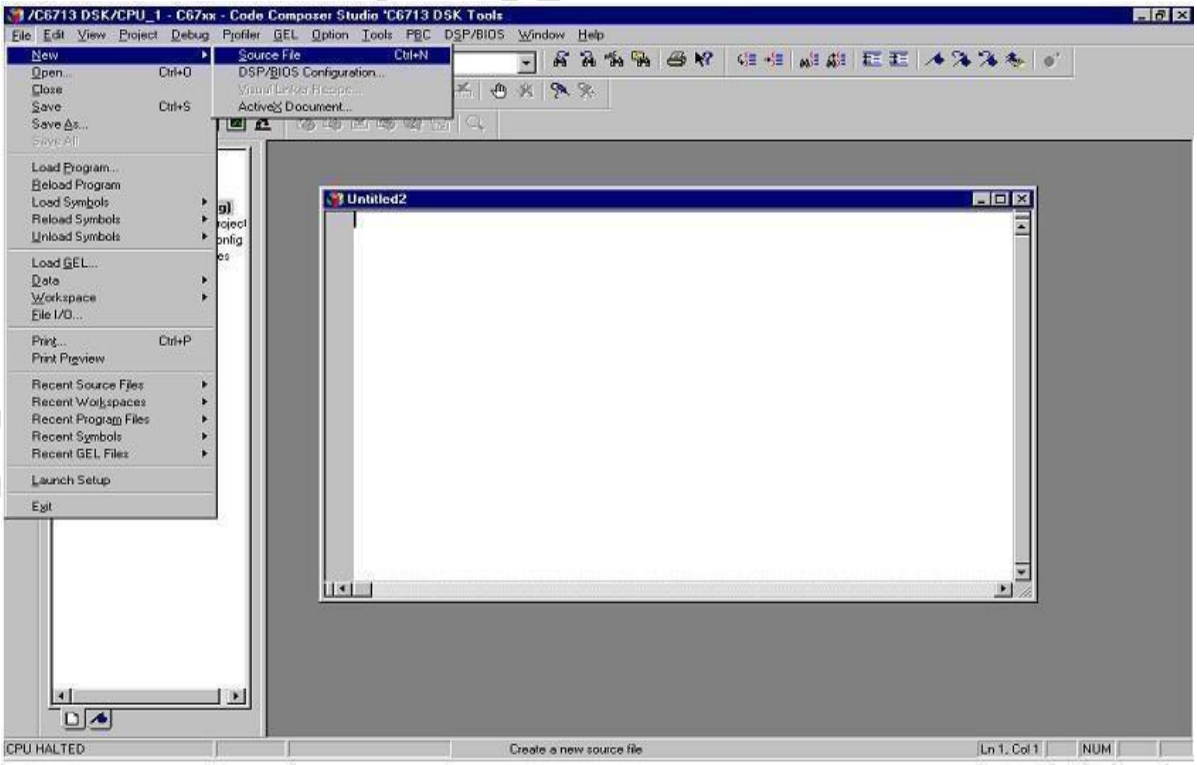
1. To create a New Project

Project → New (SUM.pjt)



2. To Create a Source file

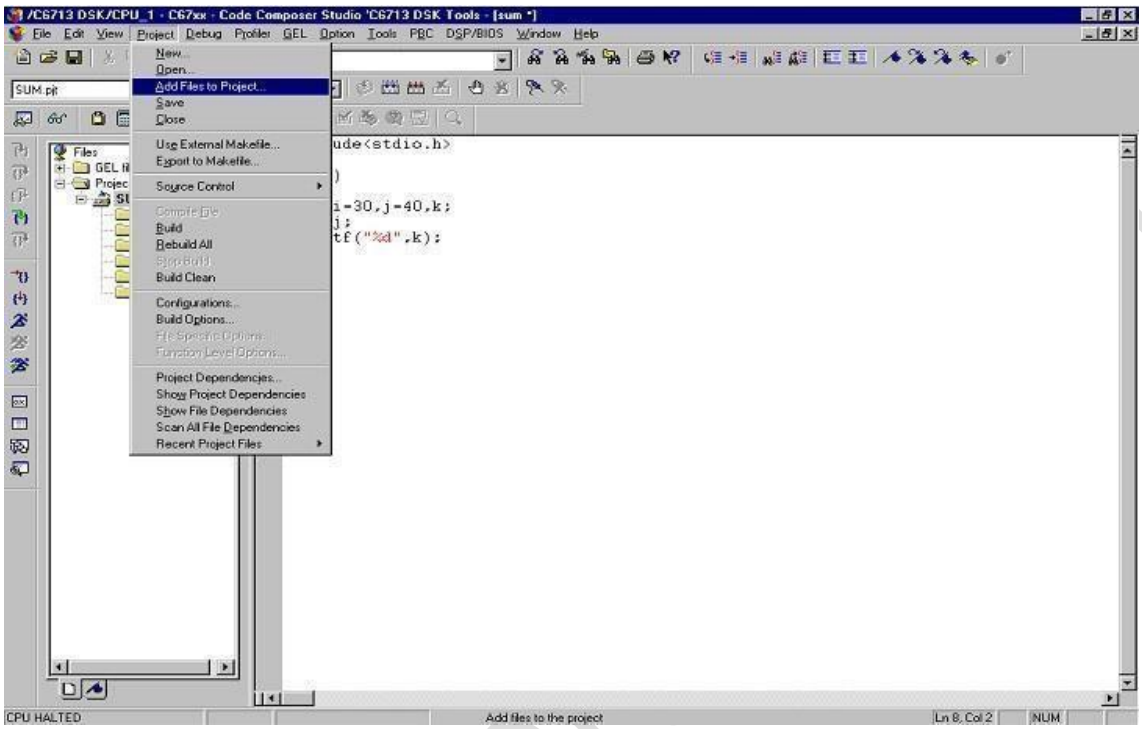
File → New

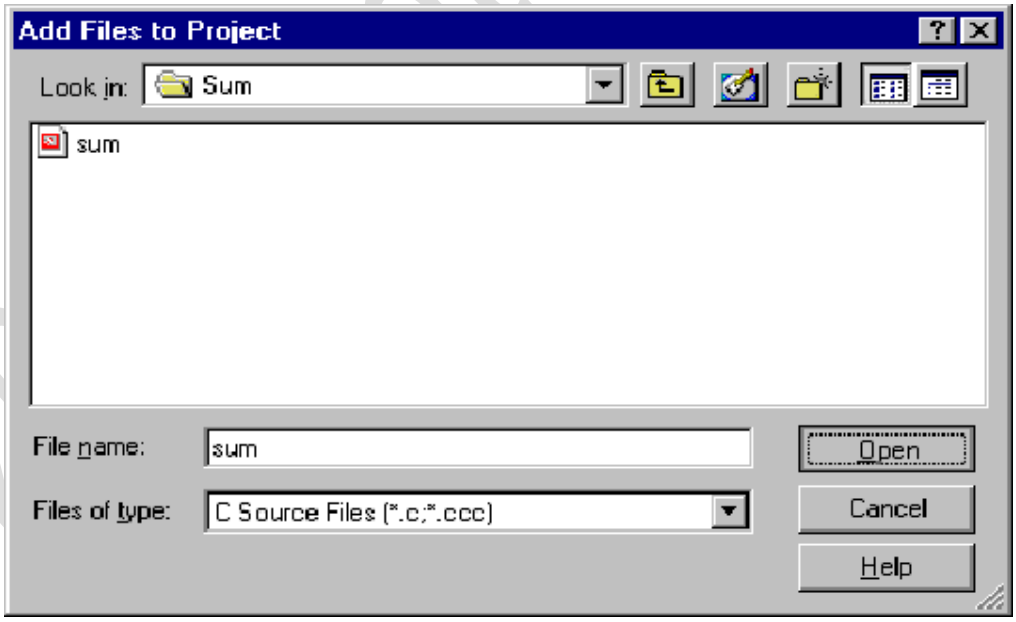


Type the code (Save &give a name to file, Eg: sum.c)

3. To Add Source files to Project

Project → Add files to Project → sum.c





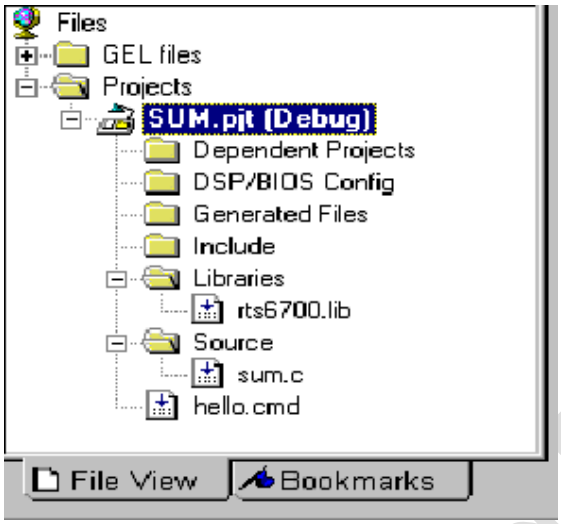
4. To Addr ts6700.lib file & hello.cmd:

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



5. To Compile:

Project → Compile File

6. To build or Link:

Project → build,

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

7. Procedure to Load and Run program:

Load program to DSK:

File → Load program → sum.Out

8. To execute project:

Debug → Run.

Ex.No : 1 **Real-Time Sine Wave Generation**
Date :

Aim

To generate a real time sine wave using TMS320C5416XX DSP kit.

Apparatus Required

Hardware : Personal Computer, TMS320C67XX kit
Software : Code Compeser Studio 6.0

Theory

A sine-wave input generates a sine-wave output at the same frequency; the only differences possible between input and output are the gain and the phase. In other words, the response of an LTI system to any one frequency can be characterized completely, knowing only phase and gain.

Procedure

1. Open Code Composer Studio v6.

2. To create the New Project
   Project→ New (File Name. pjt, Eg: vvits.pjt)

3. To create a Source file
   File →New→ Type the code (Save & give file name, Eg: sum.c).

4. To Add Source files to Project

5. To Add rts.lib file & Hello.cmd:
   Project→ Add files to Project→ rts6700.lib
   Library files: rts6700.lib (Path: c:\ti\c6000\cgtools\lib\ rts6700.lib)

6. Project→ Add files to Project →hello.cmd
   CMD file - Which is common for all non real time programs. (Path: c:\ti \ tutorial\dsk6713\hello1\hello.cmd)
   Note: Select Linker Command file (*.cmd) in Type of files

Compile

1. To Compile: Project→ Compile

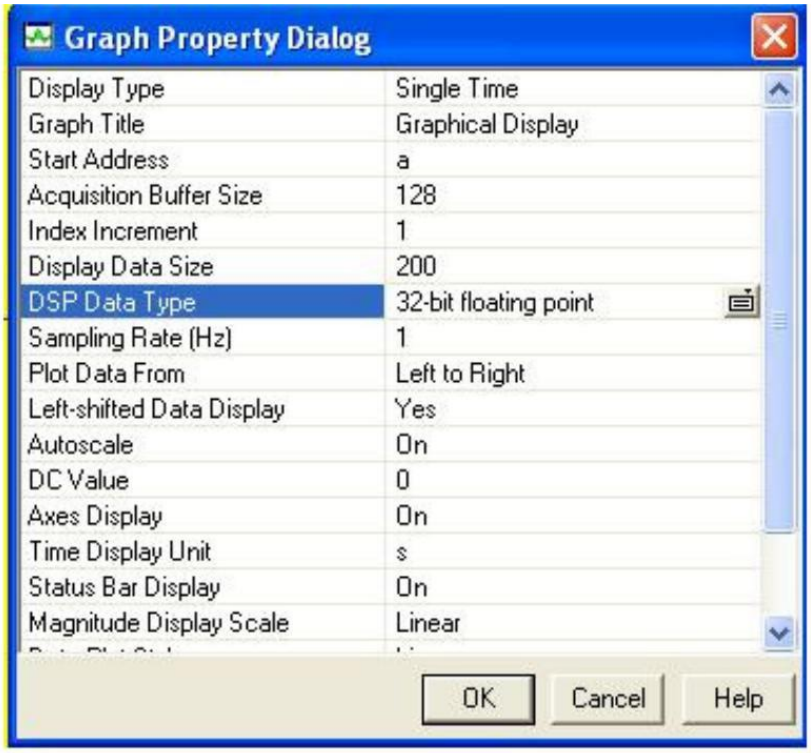
2. To Rebuild: project → rebuild,
   Which will create the final .out executable file. (Eg. vvit.out).

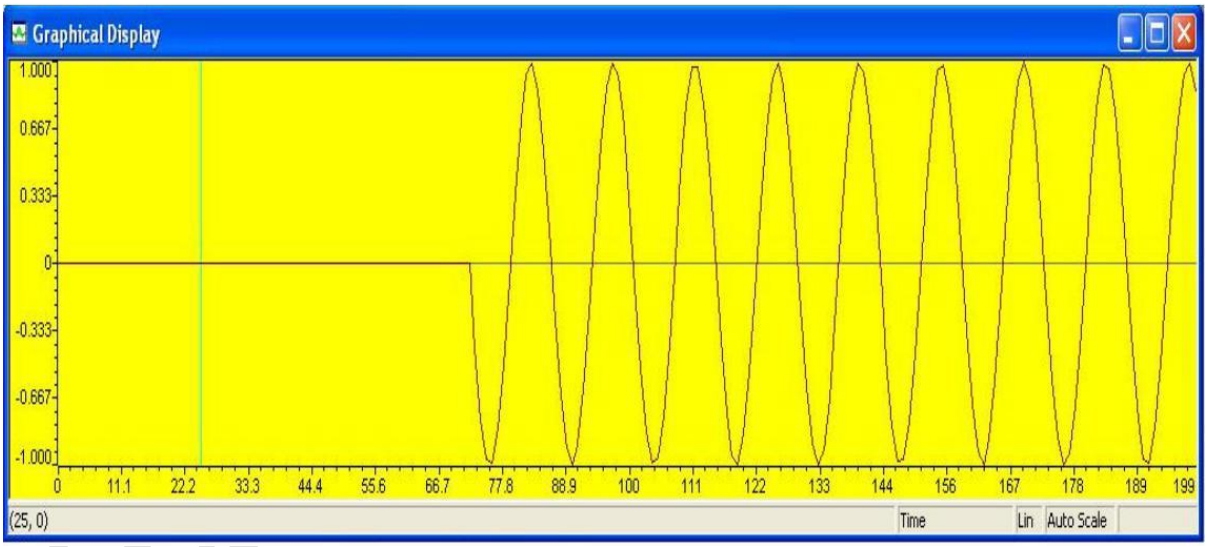
3. Procedure to Lode and Run program:
   Load the Program to DSK: File→ Load program →vvit.out
   To execute project: Debug → Run

Program

```
#include <stdio.h>
#include <math.h>
float a[500];
void main()
{
int i=0;
for(i=0;i<500;i++)
{
a[i]=sin(2*3.14*10000*i);
}
}
```

Output





Results

Thus the sine wave generated successfully using TMS320C6748 DSP kit.

Ex.No : 2.a **Programming examples using C, Assembly and linear assembly**
Date :

Aim

To write C program for calculate the efficient dot product using TMS320C6748 DSP kit.

Apparatus Required

Hardware : Personal Computer, TMS320C67XX kit
Software : Code Compeser Studio 6.0

Procedure

1. Open Code Composer Studio v6.

2. To create the New Project
   Project→ New (File Name. pjt, Eg: vvits.pjt)

3. To create a Source file
   File →New→ Type the code (Save & give file name, Eg: sum.c).

4. To Add Source files to Project

5. To Add rts.lib file & Hello.cmd:
   Project→ Add files to Project→ rts6700.lib
   Library files: rts6700.lib (Path: c:\ti\c6000\cgtools\lib\ rts6700.lib)

6. Project→ Add files to Project →hello.cmd
   CMD file - Which is common for all non real time programs. (Path: c:\ti \ tutorial\dsk6713\hello1\hello.cmd)
   Note: Select Linker Command file (*.cmd) in Type of files

Compile

1. To Compile: Project→ Compile

2. To Rebuild: project → rebuild,
   Which will create the final .out executable file. (Eg. vvit.out).

3. Procedure to Lode and Run program:
   Load the Program to DSK: File→ Load program →vvit.out
   To execute project: Debug → Run

Program

Efficient Dot Product

```
//dotpopt.c Optimized dot product of two arrays
#include <stdio.h>
#include <dotp4.h> //header file with data
#define count 4
short x[count] = {x_array}; //declare 1st array
short y[count] = {y_array}; //declare 2nd array
volatile int result = 0; //result
main()
{
result = dotpfunc(x,y,count); //call optimized function
printf("result = %d decimal \n", result);//print result
}
//dotpfunc.c Optimized dot product function
int dotpfunc(const short *a, const short *b, int ncount)
{
int sum = 0;
int i;
_nassert((int)(a)%4 == 0);
_nassert((int)(b)%4 == 0);
_nassert((int)(ncount)%4 == 0);
for ( i = 0; i < ncount; i++)
{
sum += (a[i] * b[i]); //sum of products
}
return (sum); //return sum as result
}
```

Output

```
    1   2   3          1   1   1
X= 4   5   6       Y=1   1   1
    7   8   9          1   1   1

Result = [6 15 24]
```

Result

Thus the efficient calculation dot product is implemented successfully using TMS320C6748 DSP kit.

Ex.No : 2.b **Programming examples using C, Assembly and linear assembly**
Date :

Aim

To find the factorial of a number using C calling an assembly function using TMS320C6748 DSP kit.

Apparatus Required

Hardware : Personal Computer, TMS320C67XX kit
Software : Code Compeser Studio 6.0

Procedure

1. Open Code Composer Studio v6.

2. To create the New Project
   Project→ New (File Name. pjt, Eg: vvits.pjt)

3. To create a Source file
   File →New→ Type the code (Save & give file name, Eg: sum.c).

4. To Add Source files to Project

5. To Add rts.lib file & Hello.cmd:
   Project→ Add files to Project→ rts6700.lib
   Library files: rts6700.lib (Path: c:\ti\c6000\cgtools\lib\ rts6700.lib)

6. Project→ Add files to Project →hello.cmd
   CMD file - Which is common for all non real time programs. (Path: c:\ti \ tutorial\dsk6713\hello1\hello.cmd)
   Note: Select Linker Command file (*.cmd) in Type of files

Compile

1. To Compile: Project→ Compile

2. To Rebuild: project → rebuild,
   Which will create the final .out executable file. (Eg. vvit.out).

3. Procedure to Lode and Run program:
   Load the Program to DSK: File→ Load program →vvit.out
   To execute project: Debug → Run

Program

```
//Factorial.c Finds factorial of n. Calls function factfunc
#include <stdio.h> //for print statement
void main()
{
short n = 7; //set value
short result; //result from asm function
result = factfunc(n); //call ASM function factfunc
printf("factorial = %d", result);//print result from asm function
}
```

```
;Factfunc.asm Assembly function called from C to find factorial
.def _factfunc ;ASM function called from C
_factfunc: MV A4,A1 ;setup loop count in A1
SUB A1,1,A1 ;decrement loop count
LOOP: MPY A4,A1,A4 ;accumulate in A4
NOP ;for 1 delay slot with MPY
SUB A1,1,A1 ;decrement for next multiply
[A1] B LOOP ;branch to LOOP if A1 # 0
NOP 5 ;five NOPs for delay slots
B B3 ;return to calling routine
NOP 5 ;five NOPs for delay slots
.end
```

Output

Factorial of 7 is 5040

Result

Thus the factorial of a number using C calling an assembly function is successfully executed using TMS320C6748 DSP kit.

Ex.No : 2.c **Programming examples using C, Assembly and linear assembly**
Date :

Aim

To find the factorial of a number using C calling an linear assembly function using TMS320C6748 DSP kit.

Apparatus Required

Hardware : Personal Computer, TMS320C67XX kit
Software : Code Compeser Studio 6.0

Procedure

1. Open Code Composer Studio v6.

2. To create the New Project
   Project→ New (File Name. pjt, Eg: vvits.pjt)

3. To create a Source file
   File →New→ Type the code (Save & give file name, Eg: sum.c).

4. To Add Source files to Project

5. To Add rts.lib file & Hello.cmd:
   Project→ Add files to Project→ rts6700.lib
   Library files: rts6700.lib (Path: c:\ti\c6000\cgtools\lib\ rts6700.lib)

6. Project→ Add files to Project →hello.cmd
   CMD file - Which is common for all non real time programs. (Path: c:\ti \ tutorial\dsk6713\hello1\hello.cmd)
   Note: Select Linker Command file (*.cmd) in Type of files

Compile

1. To Compile: Project→ Compile

2. To Rebuild: project → rebuild,
   Which will create the final .out executable file. (Eg. vvit.out).

3. Procedure to Lode and Run program:
   Load the Program to DSK: File→ Load program →vvit.out
   To execute project: Debug → Run

Program

Factorial Using C Calling a Linear Assembly Function (factclasm)

```
//Factclasm.c Factorial of number. Calls linear ASM function
#include <stdio.h> //for print statement
void main()
{
short number = 7; //set value
short result; //result of factorial
result = factclasmfunc(number); //call ASM function factlasmfunc
printf("factorial = %d", result); //result from linear ASM function
}
```

```
;Factclasmfunc.sa Linear ASM function called from C to find factorial
.ref _factclasmfunc ;Linear ASM func called from C
_factclasmfunc: .cproc number ;start of linear ASM function
.reg a,b ;asm optimizer directive
mv number,b ;setup loop count in b
mv number,a ;move number to a
sub b,1,b ;decrement loop counter
loop: mpy a,b,a ;n(n-1)
sub b,1,b ;decrement loop counter
[b] b loop ;loop back to loop if count #0
.return a ;result to calling function
.endproc ;end of linear ASM function
```

Output

Factorial of 7 is 5040

Result

Thus the factorial of a number using C calling a linear assembly function is successfully executed using TMS320C6748 DSP kit.

Ex.No : 3 **Implementation of Moving Average filter**
Date :

Aim

To write the C program for implement the moving average filter using TMS320C6748 DSP kit.

Apparatus Required

Hardware : Personal Computer, TMS320C67XX kit
Software : Code Compeser Studio 6.0

Theory

The moving average filter is widely used in DSP and arguably is the easiest of all digital filters to understand. It is particularly effective at removing (high frequency) random noise from a signal or at smoothing a signal. The moving average filter operates by taking the arithmetic mean of a number of past input samples in order to produce each output sample. This may be represented by the equation

$$y(n) = \frac{1}{N} \sum_{i=0}^{N-1} x(n-i)$$

where x(n) represents the nth sample of an input signal and y (n) the nth sample of the filter output. The moving average filter is an example of convolution using a very simple filter kernel or impulse response comprising N coefficients each of value 1 /N .

Procedure

1. Open Code Composer Studio v4.

2. To create the New Project
   Project→ New (File Name. pjt, Eg: vvits.pjt)

3. To create a Source file
   File →New→ Type the code (Save & give file name, Eg: sum.c).

4. To Add Source files to Project

5. To Add rts.lib file & Hello.cmd:
   Project→ Add files to Project→ rts6700.lib
   Library files: rts6700.lib (Path: c:\ti\c6000\cgtools\lib\ rts6700.lib)

6. Project→ Add files to Project →hello.cmd
   CMD file - Which is common for all non real time programs. (Path: c:\ti \ tutorial\dsk6713\hello1\hello.cmd)
   Note: Select Linker Command file (*.cmd) in Type of files

Compile

1. To Compile: Project→ Compile

2. To Rebuild: project → rebuild,

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

3. Procedure to Lode and Run program:

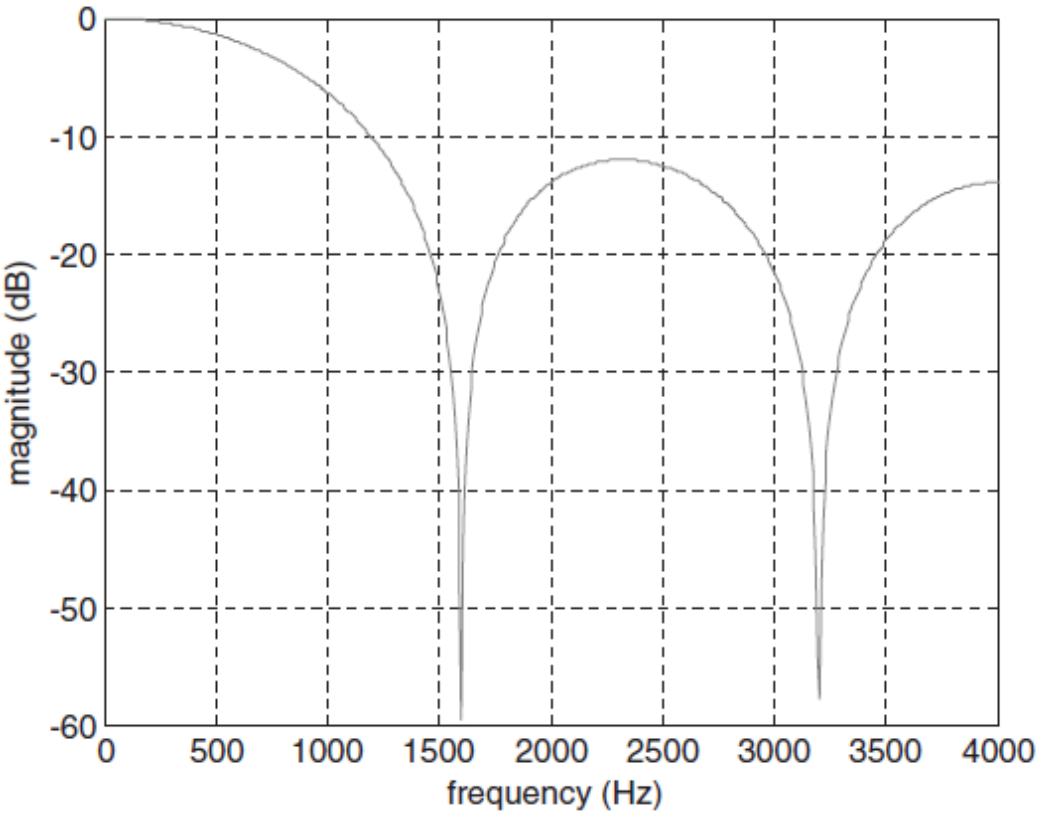
Load the Program to DSK: File→ Load program →vvit.out

To execute project: Debug → Run

Program

```
#include "DSK6713_AIC23.h" //codec support
Uint32 fs=DSK6713_AIC23_FREQ_8KHZ; //set sampling rate
#define DSK6713_AIC23_INPUT_MIC 0x0015
#define DSK6713_AIC23_INPUT_LINE 0x0011
Uint16 inputsource=DSK6713_AIC23_INPUT_LINE; //select input
#define N 5 //no of points averaged
float x[N]; //filter input delay line
float h[N]; //filter coefficients
interrupt void c_int11() //interrupt service routine
{
short i;
float yn = 0.0;
x[0]=(float)(input_left_sample()); //get new input sample
for (i=0 ; i<N ; i++) //calculate filter output
yn += h[i]*x[i];
for (i=(N-1) ; i>0 ; i--) //shift delay line contents
x[i] = x[i-1];
output_left_sample((short)(yn)); //output to codec
return;
}
void main()
{
short i; //index variable
for (i=0 ; i<N ; i++) //initialise coefficients
h[i] = 1.0/N;
comm_intr(); //initialise DSK
while(1); //infinite loop
}
```

Output



Result

Thus the C program for implementing moving average filter is executed successfully using TMS320C6748 DSP kit.

Ex.No : 4 FIR implementation with a Pseudorandom noise sequence as input to a filter
Date :

Aim

To write a C program for implementation of FIR filter with a pseudorandom noise sequence as input to a filter using TMS320C6748 DSP kit.

Apparatus Required

Hardware : Personal Computer, TMS320C67XX kit
Software : Code Compeser Studio 6.0

Theory

An FIR filter is a filter with no feedback in its equation. This can be an advantage because it makes an FIR filter inherently stable. Another advantage of FIR filters is the fact that they can produce linear phases. So, if an application requires linear phases, the decision is simple, an FIR filter must be used. The main drawback of a digital FIR filter is the time that it takes to execute. Since the filter has no feedback, many more coefficients are needed in the system equation to meet the same requirements that would be needed in an IIR filter. For every extra coefficient, there is an extra multiply and extra memory requirements for the DSP. For a demanding system, the speed and memory requirements to implement an FIR system can make the system unfeasible.

Procedure

1. Open Code Composer Studio v6.

2. To create the New Project
   Project→ New (File Name. pjt, Eg: vvits.pjt)

3. To create a Source file
   File →New→ Type the code (Save & give file name, Eg: sum.c).

4. To Add Source files to Project

5. To Add rts.lib file & Hello.cmd:
   Project→ Add files to Project→ rts6700.lib
   Library files: rts6700.lib (Path: c:\ti\c6000\cgtools\lib\ rts6700.lib)

6. Project→ Add files to Project →hello.cmd
   CMD file - Which is common for all non real time programs. (Path: c:\ti \ tutorial\dsk6713\hello1\hello.cmd)
   Note: Select Linker Command file (*.cmd) in Type of files

Compile

1. To Compile: Project→ Compile

2. To Rebuild: project → rebuild,
   Which will create the final .out executable file. (Eg. vvit.out).

3. Procedure to Lode and Run program:

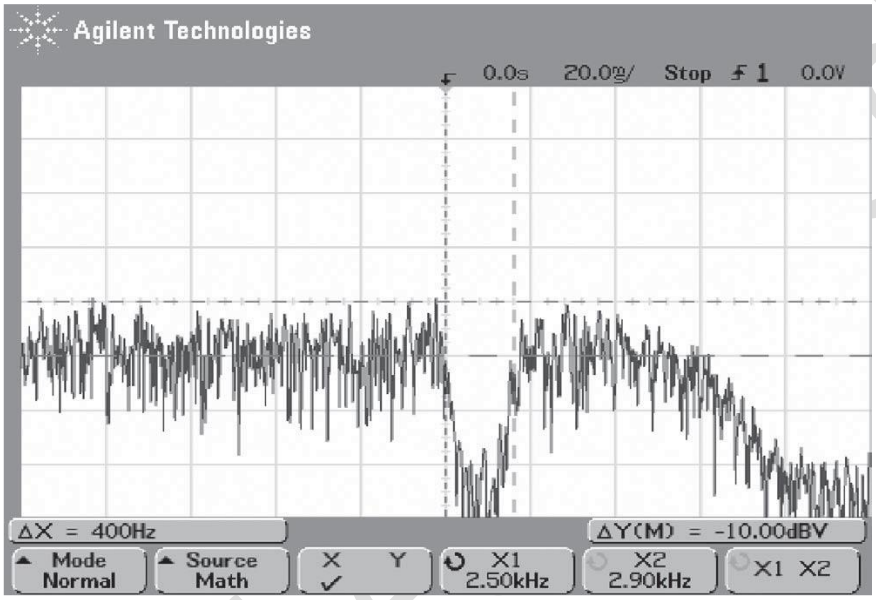
Load the Program to DSK: File→ Load program →vvit.out
To execute project: Debug → Run

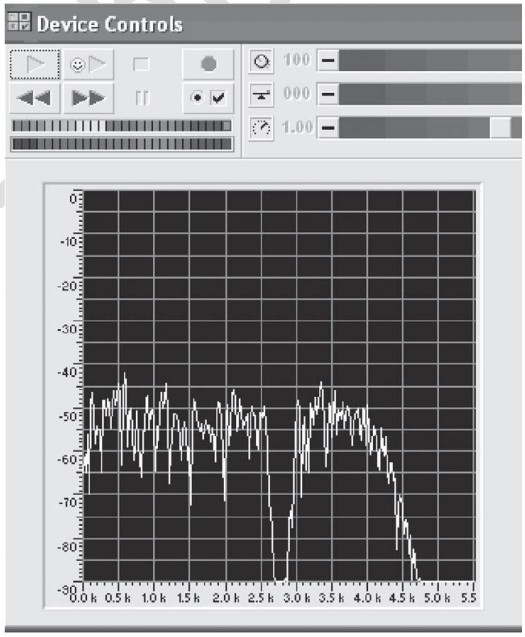
Program

```
//firprn.c FIR with internally generated input noise sequence
#include "DSK6713_AIC23.h" //codec support
Uint32 fs=DSK6713_AIC23_FREQ_8KHZ; //set sampling rate
#define DSK6713_AIC23_INPUT_MIC 0x0015
#define DSK6713_AIC23_INPUT_LINE 0x0011
Uint16 inputsource=DSK6713_AIC23_INPUT_LINE; //select line in
#include "bs2700f.cof" //filter coefficient file
#include "noise_gen.h" //support file for noise
int fb; //feedback variable
shift_reg sreg; //shift register
#define NOISELEVEL 8000 //scale factor for noise
float x[N]; //filter delay line
int prand(void) //pseudo-random noise
{
int prnseq;
if(sreg.bt.b0)
prnseq = -NOISELEVEL; //scaled -ve noise level
else
prnseq = NOISELEVEL; //scaled +ve noise level
fb =(sreg.bt.b0)^(sreg.bt.b1); //XOR bits 0,1
fb^=(sreg.bt.b11)^(sreg.bt.b13); //with bits 11,13 -> fb
sreg.regval<<=1; //shift register 1 bit left
sreg.bt.b0=fb; //close feedback path
return prnseq;
}
void resetreg(void) //reset shift register
{
sreg.regval=0xFFFF; //initial seed value
fb = 1; //initial feedback value
}
interrupt void c_int11() //interrupt service routine
{
short i; //declare index variable
float yn = 0.0;
x[0] = (float)(prand()); //get new input sample
for (i=0 ; i<N ; i++) //calculate filter output
yn += h[i]*x[i];
for (i=(N-1) ; i>0 ; i--) //shift delay line contents
x[i] = x[i-1];
output_left_sample((short)(yn)); //output to codec
return; //return from interrupt
}
void main()
```

```
{
resetreg(); //reset shift register
comm_intr(); //initialise DSK
while (1); //infinite loop
}
```

Output





Result

Thus the C program for implementation of FIR filter with a pseudorandom noise sequence as input to a filter using TMS320C6748 DSP kit is executed successfully.

Ex.No : 5 **Fixed point implementation of IIR filter**
Date :

Aim

To write a C program for fixed point implementation of IIR filter using TMS320C6748 DSP kit.

Apparatus Required

Hardware : Personal Computer, TMS320C67XX kit
Software : Code Compeser Studio 6.0

Theory

The infinite impulse response (IIR) filter is a recursive filter in that the output from the filter is computed by using the current and previous inputs and previous outputs. Because the filter uses previous values of the output, there is feedback of the output in the filter structure. The design of the IIR filter is based on identifying the pulse transfer function G(z) that satisfies the requirements of the filter specification. This can be undertaken either by developing an analogue prototype and then transforming it to the pulse transfer function, or by designing directly in digital.

Procedure

1. Open Code Composer Studio v6.

2. To create the New Project
   Project→ New (File Name. pjt, Eg: vvits.pjt)

3. To create a Source file
   File →New→ Type the code (Save & give file name, Eg: sum.c).

4. To Add Source files to Project

5. To Add rts.lib file & Hello.cmd:
   Project→ Add files to Project→ rts6700.lib
   Library files: rts6700.lib (Path: c:\ti\c6000\cgtools\lib\ rts6700.lib)

6. Project→ Add files to Project →hello.cmd
   CMD file - Which is common for all non real time programs. (Path: c:\ti \ tutorial\dsk6713\hello1\hello.cmd)
   Note: Select Linker Command file (*.cmd) in Type of files

Compile

1. To Compile: Project→ Compile

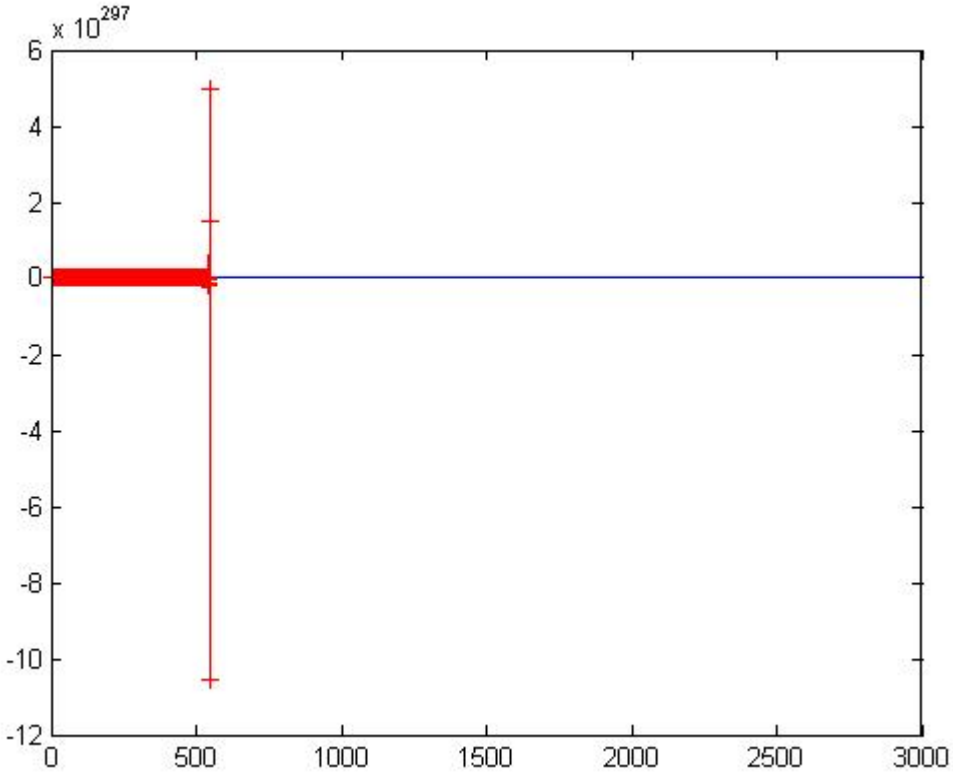
2. To Rebuild: project → rebuild,
   Which will create the final .out executable file. (Eg. vvit.out).

3. Procedure to Lode and Run program:
   Load the Program to DSK: File→ Load program →vvit.out
   To execute project: Debug → Run

Program

```
// iir.c filter using cascaded second order sections
// 16-bit integer coefficients read from .cof file
#include "DSK6713_AIC23.h" //codec support
Uint32 fs=DSK6713_AIC23_FREQ_8KHZ; //set sampling rate
#define DSK6713_AIC23_INPUT_MIC 0x0015
#define DSK6713_AIC23_INPUT_LINE 0x0011
Uint16 inputsource=DSK6713_AIC23_INPUT_LINE;
#include "bs1800int.cof"
short w[NUM_SECTIONS][2] = {0};
interrupt void c_int11() //interrupt service routine
{
short section; //index for section number
short input; //input to each section
int wn,yn; //intermediate and output
//values in each stage
input = input_left_sample();
for (section=0 ; section< NUM_SECTIONS ; section++)
{
wn = input - ((a[section][0]*w[section][0])>>15)
- ((a[section][1]*w[section][1])>>15);
yn = ((b[section][0]*wn)>>15)
+ ((b[section][1]*w[section][0])>>15)
+ ((b[section][2]*w[section][1])>>15);
w[section][1] = w[section][0];
w[section][0] = wn;
input = yn; //output of current section
//will be input to next
}
output_left_sample((short)(yn)); //before writing to codec
return; //return from ISR
}
void main()
{
comm_intr(); //init DSK, codec, McBSP
while(1); //infinite loop
}
```

Output



Result

Thus the C program for implementation of fixed point IIR filters using TMS320C6748 DSP kit is executed successfully.

Ex.No : 6 **FFT of Real-Time input signal**
Date :

Aim

To implement the FFT (Fast Fourier Transform) of a real-time input signal using the TMS320C6748 DSP kit

Apparatus Required

Hardware : Personal Computer, TMS320C67XX kit
Software : Code Compeser Studio 6.0

Theory

The DFT converts a time - domain sequence into an equivalent frequency – domain sequence. The inverse DFT performs the reverse operation and converts a frequency - domain sequence into an equivalent time - domain sequence. The FFT is a very efficient algorithm technique based on the DFT but with fewer computations required. The FFT is one of the most commonly used operations in digital signal processing to provide a frequency spectrum analysis. Two different procedures are introduced to compute an FFT: the decimation - in - frequency and the decimation - in - time.

Procedure

1. Open Code Composer Studio v6.

2. To create the New Project
   Project→ New (File Name. pjt, Eg: vvits.pjt)

3. To create a Source file
   File →New→ Type the code (Save & give file name, Eg: sum.c).

4. To Add Source files to Project

5. To Add rts.lib file & Hello.cmd:
   Project→ Add files to Project→ rts6700.lib
   Library files: rts6700.lib (Path: c:\ti\c6000\cgtools\lib\ rts6700.lib)

6. Project→ Add files to Project →hello.cmd
   CMD file - Which is common for all non real time programs. (Path: c:\ti \ tutorial\dsk6713\hello1\hello.cmd)
   Note: Select Linker Command file (*.cmd) in Type of files

Compile

1. To Compile: Project→ Compile

2. To Rebuild: project → rebuild,
   Which will create the final .out executable file. (Eg. vvit.out).

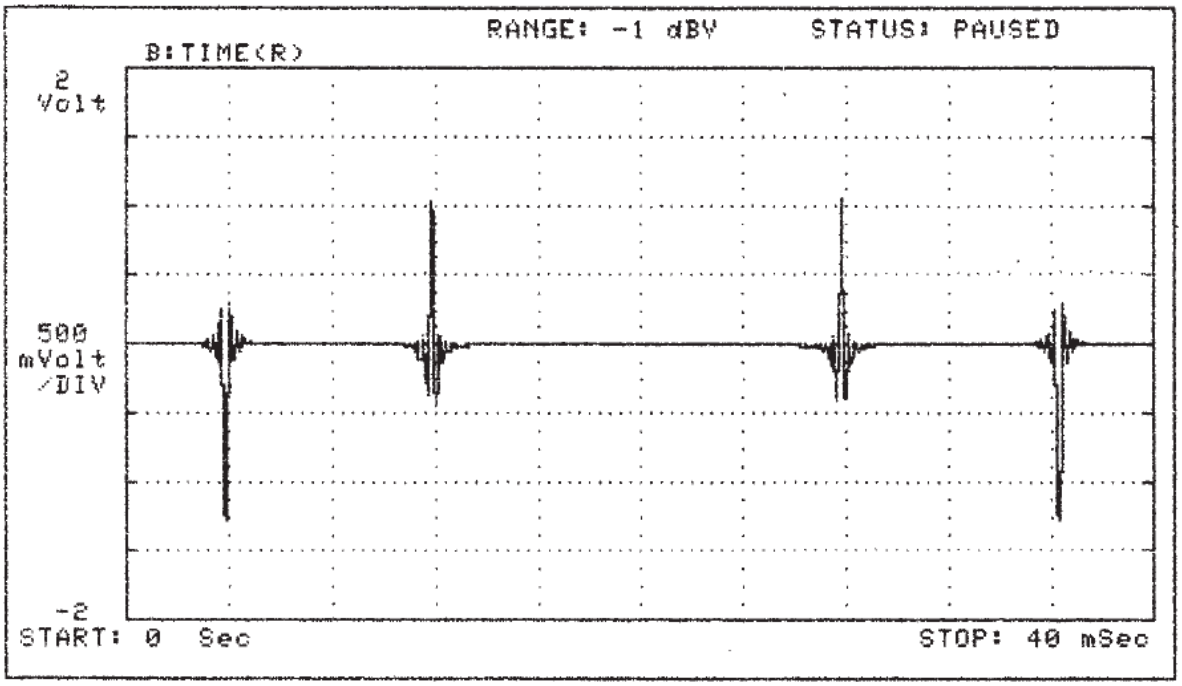
3. Procedure to Lode and Run program:
   Load the Program to DSK: File→ Load program →vvit.out
   To execute project: Debug → Run

Program

```
#include "dsk6713_aic23.h"
Uint32 fs=DSK6713_AIC23_FREQ_8KHZ;
#include <math.h>
#define PTS 256 //# of points for FFT
#define PI 3.14159265358979
typedef struct {float real,imag;} COMPLEX;
void FFT(COMPLEX *Y, int n); //FFT prototype
float iobuffer[PTS]; //as input and output buffer
float x1[PTS]; //intermediate buffer
short i; //general purpose index variable
short buffercount = 0; //number of new samples in iobuffer
short flag = 0; //set to 1 by ISR when iobuffer full
COMPLEX w[PTS]; //twiddle constants stored in w
COMPLEX samples[PTS]; //primary working buffer
main()
{
for (i = 0 ; i<PTS ; i++) // set up twiddle constants in w
{
w[i].real = cos(2*PI*i/512.0); //Re component of twiddle constants
w[i].imag =-sin(2*PI*i/512.0); //Im component of twiddle constants
}
comm_intr(); //init DSK, codec, McBSP
while(1) //infinite loop
{
while (flag == 0) ; //wait until iobuffer is full
flag = 0; //reset flag
for (i = 0 ; i < PTS ; i++) //swap buffers
{
samples[i].real=iobuffer[i]; //buffer with new data
iobuffer[i] = x1[i]; //processed frame to iobuffer
}
for (i = 0 ; i < PTS ; i++)
samples[i].imag = 0.0; //imag components = 0
FFT(samples,PTS); //call function FFT.c
for (i = 0 ; i < PTS ; i++) //compute magnitude
{
x1[i] = sqrt(samples[i].real*samples[i].real
+ samples[i].imag*samples[i].imag)/16;
}
x1[0] = 32000.0; //negative spike for reference
} //end of infinite loop
} //end of main
interrupt void c_int11() //ISR
{
output_sample((short)(iobuffer[buffercount]));//output from iobuffer
iobuffer[buffercount++]=(float)((short)input_sample());//input>iobuffer
if (buffercount >= PTS) //if iobuffer full
```

```
{
buffercount = 0; //reinit buffercount
flag = 1; //set flag
}
}
```

Output



Result

Thus the implementation of FFT of a real time input signal using the TMS320C6748 is successfully executed.