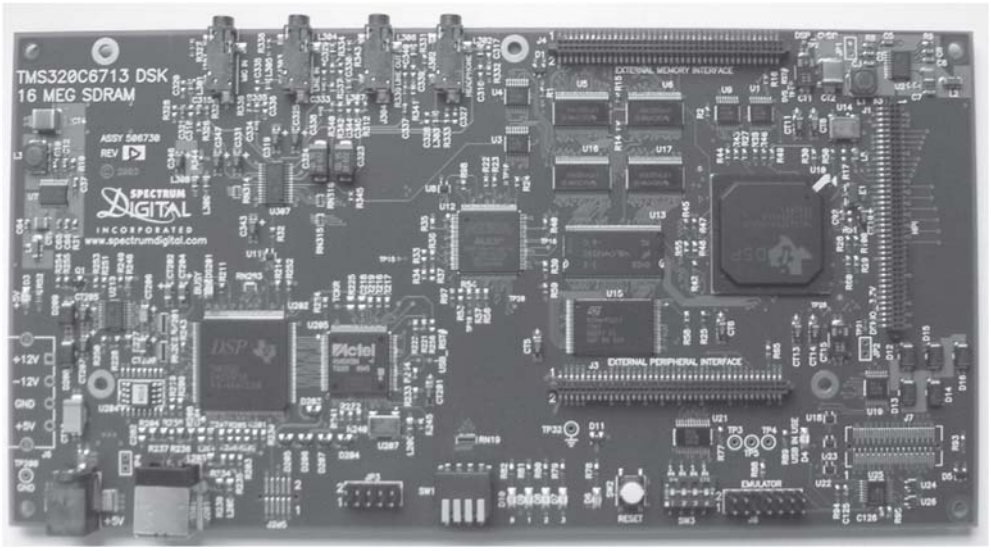
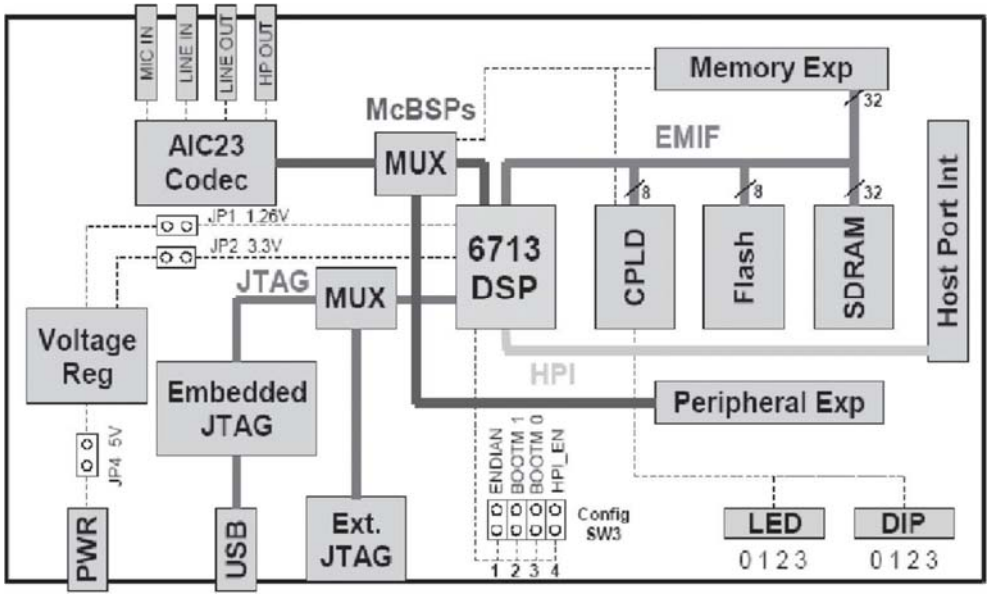
CONTENTS

STUDY OF TMS320C6713 DIGITAL SIGNAL PROCESSOR :





TMS320C6713 DIGITAL SIGNAL PROCESSOR :

The TMS320C6713 (C6713) is based on the very - long - instruction - word (VLIW) architecture, which is very well suited for numerically intensive algorithms. The internal program memory is structured so that a total of eight instructions can be fetched every cycle. For example, with a clock rate of 225 MHz, the C6713 is capable of fetching eight 32 - bit instructions every 1/(225 MHz) or 4.44 ns.
Features of the C6713 include 264 kB of internal memory (8 kB as L1P and L1D Cache and 256 kB as L2 memory shared between program and data space), eight functional or execution units composed of six ALUs and two multiplier units, a 32 - bit address bus to address 4 GB (gigabytes), and two sets of 32 - bit general - purpose registers.

The C67xx processors (such as the C6701, C6711, and C6713) belong to the family of the C6x floating - point processors; whereas the C62xx and C64xx belong to the family of the C6x fi xed - point processors. The C6713 is capable of both fi xed - and floating - point processing. The architecture and instruction set of the C6713 .

TMS 320C6416 DIGITAL SIGNAL PROCESSOR :

The TMS320C6416 (C6416) is based on the VELOCITI advanced very - long - instruction - word (VLIW) architecture, which is very well suited for numerically intensive algorithms. The internal program memory is structured so that a total of eight instructions can be fetched every cycle. For example, with a clock rate of 1 GHz, the C6416 is capable of fetching eight 32 - bit instructions every 1/(1 GHz) or 1.0 ns.

Features of the C6416 include 1056 kB of internal memory (32 kB as L1P and L1D cache and 1024 kB as L2 memory shared between program and data space), eight functional or execution units composed of six ALUs and two multiplier units, a 32 - bit address bus to address 4 GB (gigabytes), and two sets of 32 - bit general - purpose registers.

CODE COMPOSER STUDIO :

Code Composer Studio (CCS) provides an integrated development environment (IDE) for real - time digital signal processing applications based on the C programming language. It incorporates a C compiler, an assembler, and a linker. It has graphical capabilities and supports real - time debugging.

The C compiler compiles a C source program with extension .c to produce an assembly source fi le with extension *.asm* . The assembler assembles an *.asm* source fi le to produce a machine language object fi le with extension *.obj* . The linker combines object fi les and object libraries as input to produce an executable file with extension *.out* . This executable fi le represents a linked common object file format (COFF), popular in Unix - based systems and adopted by several makers of digital signal processors [44] . This executable fi le can be loaded and run directly on the digital signal processor. Chapter 3 introduces the linear assembly source fi le with extension .sa , which is a " cross " between C and assembly code. A linear optimizer optimizes this source fi le to create an assembly fi le with extension .asm (similar to the task of the C compiler).

A Code Composer Studio project comprises all of the fi les (or links to all of the fi les) required in order to generate an executable fi le. A variety of options enabling fi les of different types to be added to or removed from a project are provided. In addition, a Code Composer Studio project contains information about exactly how fi les are to be used in order to generate an executable fi le. Compiler/linker options can be specifi ed. A number of debugging features are available, including setting breakpoints and watching variables, viewing memory, registers, and mixed C and assembly code, graphing results, and monitoring execution time. One can step through a program in different ways (step into, or over, or out).

Real - time analysis can be performed using CCS ' s real - time data exchange (RTDX) facility. This allows for data exchange between the host PC and the target DSK as well as analysis in real - time without halting the target.

FILE TYPES :

You will be working with a number of fi les with different extensions. They include:

1. file.pjt : to create and build a project named file.

2. file.c : C source program.

3. file.asm : assembly source program created by the user, by the C compiler, or by the linear optimizer.

4. file.sa : linear assembly source program. The linear optimizer uses *file.sa* as input to produce an assembly program *file.asm* .

5. file.h : header support file.

6. file.lib : library fi le, such as the run - time support library file rts6700.lib .

7. file.cmd : linker command fi le that maps sections to memory.

8. file.obj : object fi le created by the assembler.

9. file.out : executable fi le created by the linker to be loaded and run on the C6713 or C6416 processor.

10. file.cdb : confi guration fi le when using DSP/BIOS.

EXP NO : 01	REAL-TIME SINE WAVE GENERATION.
DATE :	

AIM :

To generate the waveform for the sine wave signal using MATLAB.

APPARATUS REQUIRED :

S.NO	NAME OF THE APPARATUS	SPECIFICATIONS	QUANTITY
01.	Personal Computer Setup	Windows 10	01
02.	Softwares Used	Matlab, Code Composer Studio (CCS) .	01
03.	Hardwares Used	DSP Trainer Kit, TMS320C54xx/TMS320C67xx DSP Development board, Function Generator and Digital Storage Oscilloscope , Microphone and speaker.	01

ALGORITHM :

1) Enter the number of cycles, period and amplitude for respective waves.

2) Generate the signals using corresponding general formula.

3) Plot the graph.

4)Verify the output.

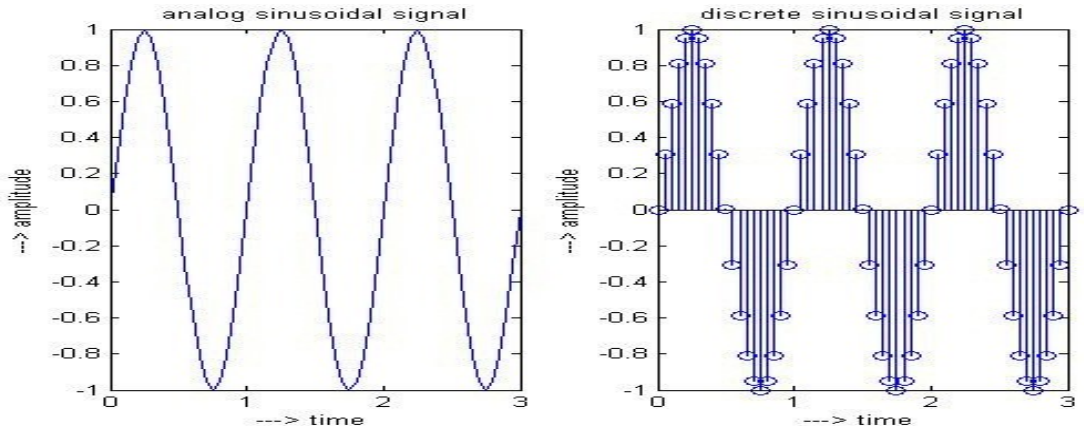
5)Verify the results.

PROGRAM :

```
clear all;

close all;clc;

N = input('enter the number of cycles....');

t = 0:0.05:N;

x = sin(2*pi*t);

subplot(121);

plot(t,x);

xlabel('---> time');

ylabel('---> amplitude');

title('analog sinusoidal signal');

subplot(122);

stem(t,x);

xlabel('---> time');

ylabel('---> amplitude');

title('discrete sinusoidal signal');
```

OUTPUT :

```
Enter the number of cycles : 3
```

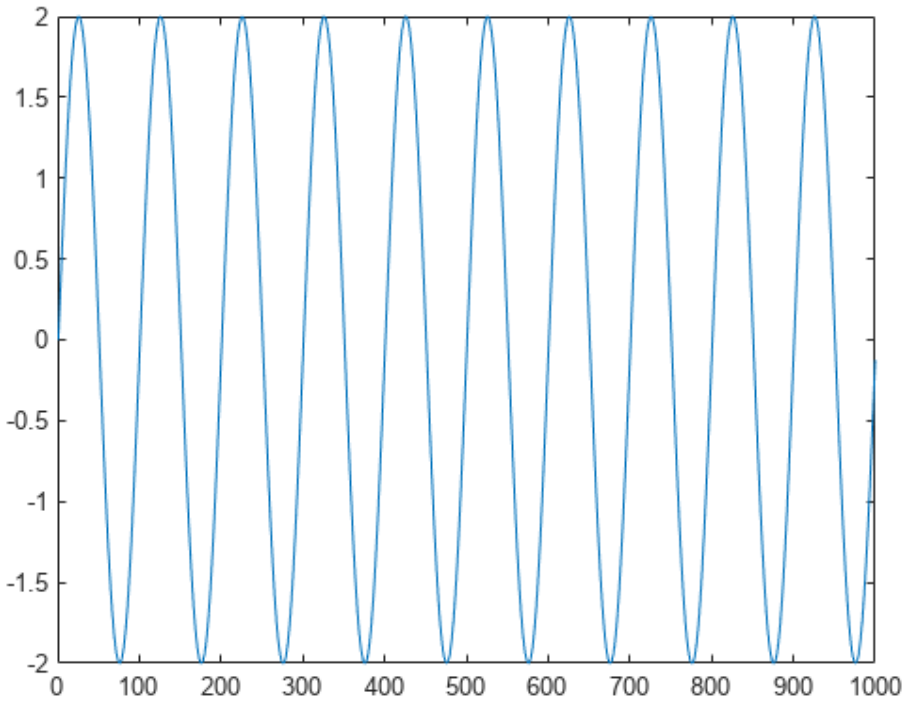


1.(B) : SINE WAVE GENERATION PROGRAM USING EIGHT POINTS WITH DIP SWITCH CONTROL (SINE8_LED.C) :

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_MIC; //select input
#define LOOPLENGTH 8
short loopindex = 0; //table index
short gain = 10; //gain factor
short sine_table[LOOPLENGTH]=
{0,707,1000,707,0,-707,-1000,-707}; //sine values
void main()
{
comm_poll(); //init DSK,codec,McBSP
DSK6713_LED_init(); //init LED from BSL
DSK6713_DIP_init(); //init DIP from BSL
while(1) //infinite loop
{
if(DSK6713_DIP_get(0)==0) //if DIP #0 pressed
{
DSK6713_LED_on(); //turn LED #0 ON
output_left_sample(sine_table[loopindex++]*gain); //output
if (loopindex >= LOOPLENGTH) loopindex = 0; //reset index
}
else DSK6713_LED_off(0); //else turn LED #0 OFF
} //end of while(1)
} //end of main
```

OUTPUT :



RESULT :

Thus the real time sine wave signal was generated .

EXP NO : 02	PROGRAMMING EXAMPLES USING C, ASSEMBLY AND LINEAR ASSEMBLY.
DATE :	

AIM :

To implement the programming examples using C for assembly and linear assembly.

APPARATUS REQUIRED :

S.NO	NAME OF THE APPARATUS	SPECIFICATIONS	QUANTITY
01.	Personal Computer Setup	Windows 10	01
02.	Softwares Used	Matlab, Code Composer Studio (CCS) .	01
03.	Hardwares Used	DSP Trainer Kit, TMS320C54xx/TMS320C67xx DSP Development board, Function Generator and Digital Storage Oscilloscope , Microphone and speaker.	01

ALGORITHM :

1) Enter the number of cycles, period and amplitude for respective waves.

2) Generate the signals using corresponding general formula.

3) Plot the graph.

4)Verify the output.

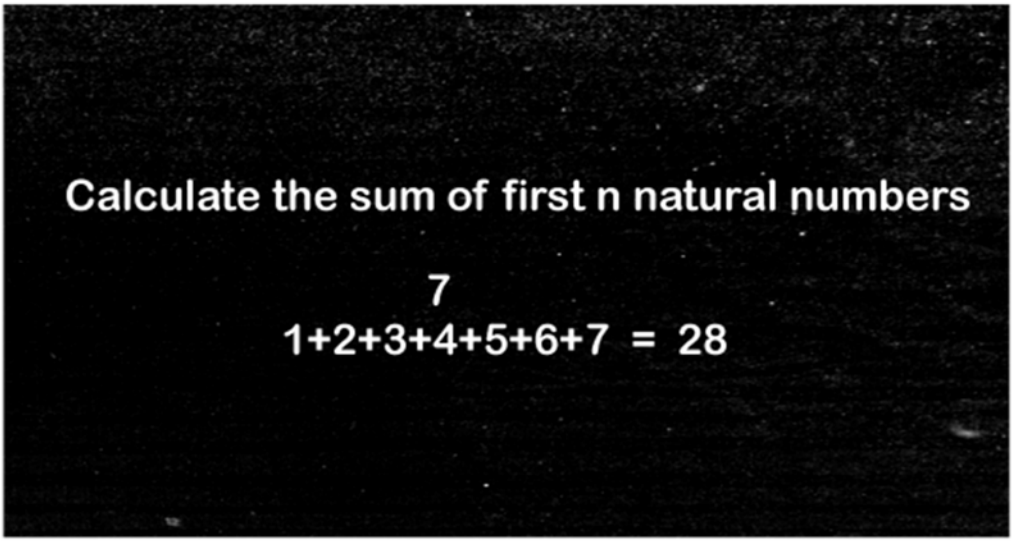
5)Verify the results.

PROGRAM :

2.(A) : SUM OF N + (N - 1) + (N - 2) + · · · + 1,USING C CALLING AN ASSEMBLY FUNCTION (SUM).

```
#include <stdio.h>
main()
{
short n=6; //set value
short result; //result from asm function
result = sumfunc(n); //call ASM function sumfunc
printf("sum = %d", result); //print result from asm function
}

;Sumfunc.asm Assembly function to find n + (n-1) + ... + 1

.def _sumfunc ;function called from C
_sumfunc: MV .L1 A4,A1 ;setup n as loop counter
SUB .S1 A1,1,A1 ;decrement n
LOOP: ADD .L1 A4,A1,A4 ;accumulate in A4
SUB .S1 A1,1,A1 ;decrement loop counter
[A1] B .S2 LOOP ;branch to LOOP if A1#0
NOP 5 ;five NOPs for delay slots
B .S2 B3 ;return to calling routine
NOP 5 ;five NOPs for delay slots
.end
```

OUTPUT :



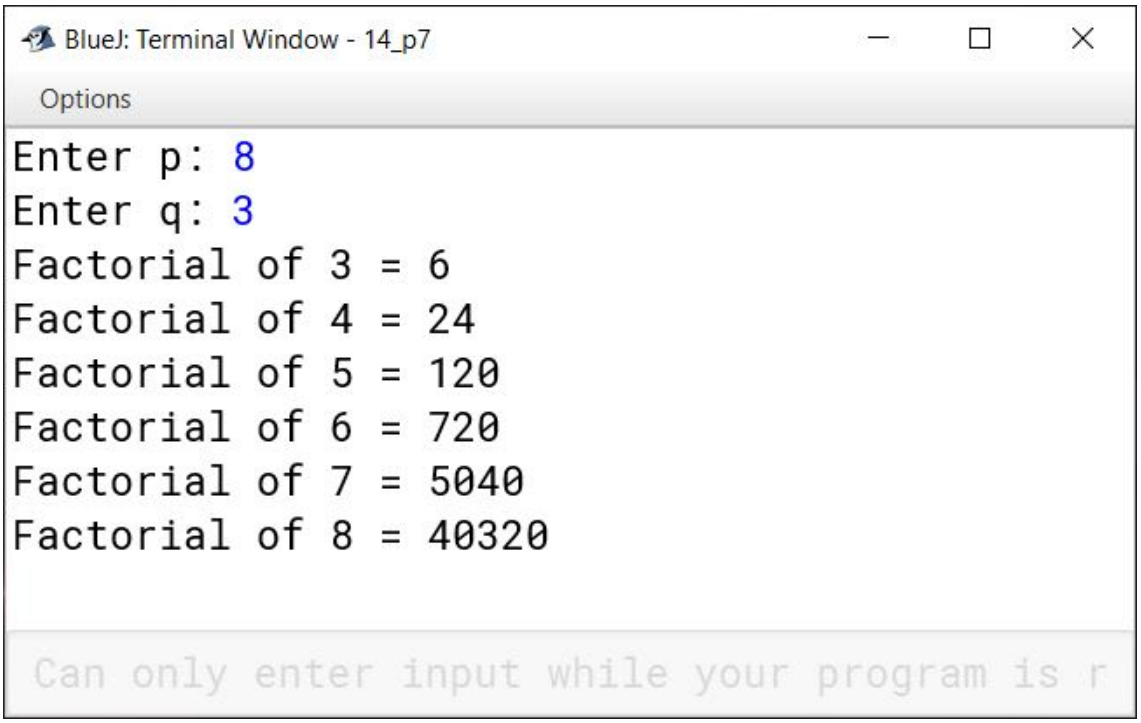
2.(B) : FACTORIAL OF A NUMBER USING C CALLING AN ASSEMBLY FUNCTION (FACTORIAL).

```
#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 :


BlueJ: Terminal Window - 14_p7
Options
Enter p: 8
Enter q: 3
Factorial of 3 = 6
Factorial of 4 = 24
Factorial of 5 = 120
Factorial of 6 = 720
Factorial of 7 = 5040
Factorial of 8 = 40320
Can only enter input while your program is r

2.(C) : 32 - BIT PSEUDORANDOM NOISE GENERATION USING C CALLING AN ASSEMBLY FUNCTION (NOISEGEN_CASM).

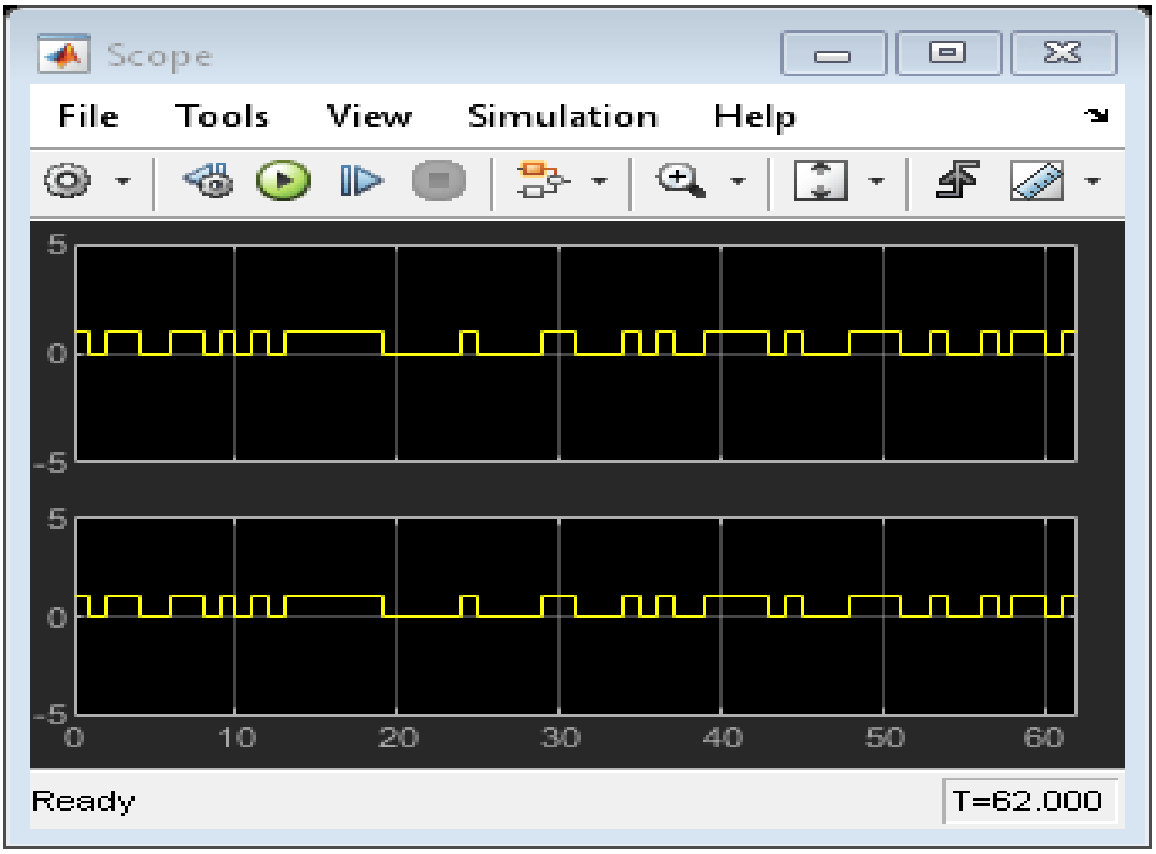
```
#include "dsk6713_aic23.h" //codec-DSK support file
Uint32 fs=DSK6713_AIC23_FREQ_48KHZ; //set sampling rate
int previous_seed;
short pos = 16000, neg = -16000; //scaling noise level
interrupt void c_int11()
{
previous_seed = noisefunc(previous_seed); //call ASM function
if(previous_seed & 0x01) output_left_sample(pos);//positive
scaling
else output_left_sample(neg);//negative scaling
}
void main ()
{
comm_intr(); //init DSK,codec,McBSP
previous_seed = noisefunc(0x7E521603); //call ASM function
while (1); //infinite loop
}
```

;**Noisegen_casmfunc.asm** Noise generation C-called function

```
.def _noisefunc ;ASM function called from C
_noisefunc ZERO A2 ;init A2 for seed manipulation
MV A4,A1 ;seed in A1
SHR A1,17,A1 ;shift right 17->bit 17 to LSB
ADD A1,A2,A2 ;add A1 to A2 => A2
SHR A1,11,A1 ;shift right 11->bit 28 to LSB
ADD A1,A2,A2 ;add again
SHR A1,2,A1 ;shift right 2->bit 30 to LSB
```

```
ADD A1,A2,A2 ;
SHR A1,1,A1 ;shift right 1->bit 31 to LSB
ADD A1,A2,A2 ;
AND A2,1,A2 ;Mask LSB of A2
SHL A4,1,A4 ;shift seed left 1
OR A2,A4,A4 ;Put A2 into LSB of A4
B B3 ;return to calling function
NOP 5 ;5 delays for branch
```

OUTPUT :



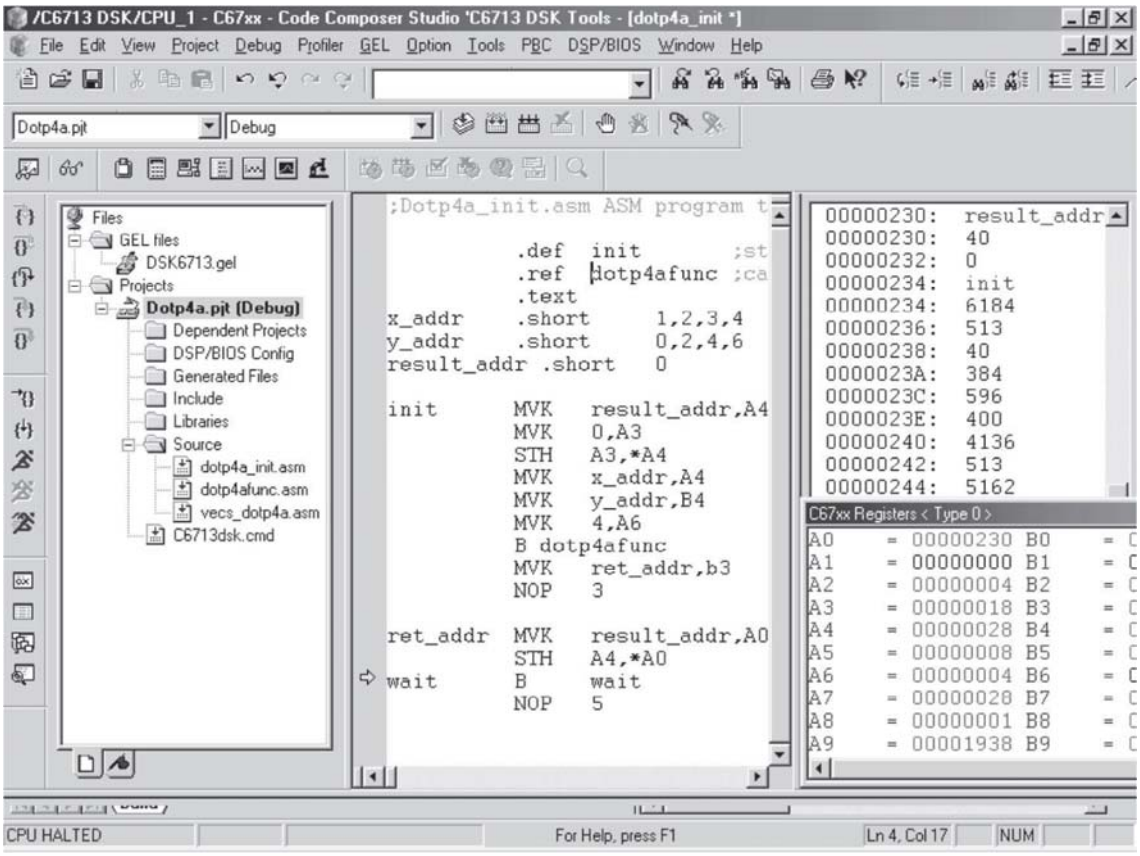
2.(D) : DOT PRODUCT USING C FUNCTION CALLING A LINEAR ASSEMBLY FUNCTION (DOTP4CLASM) :

```
#include <stdio.h> //for printing statement
#include "dotp4.h" //arrays of data values
#define count 4 //number of data values
short x[count] = {x_array}; //declare 1st array
short y[count] = {y_array}; //declare 2nd array
volatile int result = 0; //result
main()
{
result = dotp4clasmfunc(x,y,count); //call linear ASM func
printf("result = %d decimal \n", result); //print result
}
```

;**Dotp4clasmfunc.sa** Linear assembly function to multiply two arrays

```
.ref _dotp4clasmfunc ;ASM func called from C
_dotp4clasmfunc: .cproc ap,bp,count ;start section linear ASM
.reg a,b,prod,sum ;asm optimizer directive
zero sum ;init sum of products
loop: ldh *ap++,a ;pointer to 1st array->a
ldh *bp++,b ;pointer to 2nd array->b
mpy a,b,prod ;product = a*b
add prod,sum,sum ;sum of products -->sum
sub count,1,count ;decrement counter
[count] b loop ;loop back if count # 0
.return sum ;return sum as result
.endproc
```

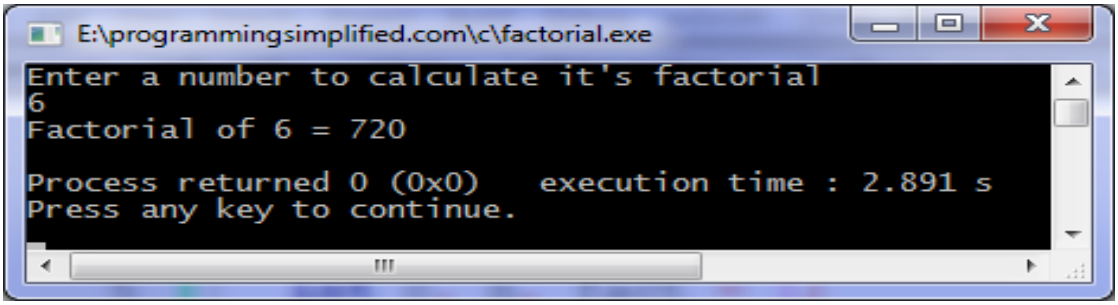
OUTPUT :


/C6713 DSK/CPU_1 - C67xx - Code Composer Studio 'C6713 DSK Tools - [dotp4a_init *]
File Edit View Project Debug Profiler GEL Option Tools PBC DSP/BIOS Window Help
Dotp4a.pjt
Debug
Files
GEL files
DSK6713.gel
Projects
Dotp4a.pjt (Debug)
Dependent Projects
DSP/BIOS Config
Generated Files
Include
Libraries
Source
dotp4a_init.asm
dotp4afunc.asm
vecs_dotp4a.asm
C6713dsk.cmd
;Dotp4a_init.asm ASM program t
.def init ;st
.ref dotp4afunc ;ca
.text
x_addr .short 1,2,3,4
y_addr .short 0,2,4,6
result_addr .short 0
init MVK result_addr,A4
MVK 0,A3
STH A3,*A4
MVK x_addr,A4
MVK y_addr,B4
MVK 4,A6
B dotp4afunc
MVK ret_addr,b3
NOP 3
ret_addr MVK result_addr,A0
STH A4,*A0
wait B wait
NOP 5
00000230: result_addr
00000230: 40
00000232: 0
00000234: init
00000234: 6184
00000236: 513
00000238: 40
0000023A: 384
0000023C: 596
0000023E: 400
00000240: 4136
00000242: 513
00000244: 5162
C67xx Registers < Type 0 >
A0 = 00000230 B0 =
A1 = 00000000 B1 =
A2 = 00000004 B2 =
A3 = 00000018 B3 =
A4 = 00000028 B4 =
A5 = 00000008 B5 =
A6 = 00000004 B6 =
A7 = 00000028 B7 =
A8 = 00000001 B8 =
A9 = 00001938 B9 =
CPU HALTED
For Help, press F1
Ln 4, Col 17
NUM

2.(E) : FACTORIAL USING C CALLING A LINEAR ASSEMBLY FUNCTION (FACTCLASM).

```
#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 :


E:\programmingsimplified.com\c\factorial.exe
Enter a number to calculate it's factorial
6
Factorial of 6 = 720
Process returned 0 (0x0)   execution time : 2.891 s
Press any key to continue.

RESULT :

Thus the programming examples using C for assembly and linear assembly wasimplemented.

EXP NO : 03	IMPLEMENTATION OF MOVING AVERAGE FILTER
DATE :	

AIM :

To implement the moving average filter .

APPARATUS REQUIRED :

S.NO	NAME OF THE APPARATUS	SPECIFICATIONS	QUANTITY
01.	Personal Computer Setup	Windows 10	01
02.	Softwares Used	Matlab, Code Composer Studio (CCS) .	01
03.	Hardwares Used	DSP Trainer Kit, TMS320C54xx/TMS320C67xx DSP Development board, Function Generator and Digital Storage Oscilloscope , Microphone and speaker.	01

ALGORITHM :

1) Enter the number of cycles, period and amplitude for respective waves.

2) Generate the signals using corresponding general formula.

3) Plot the graph.

4)Verify the output.

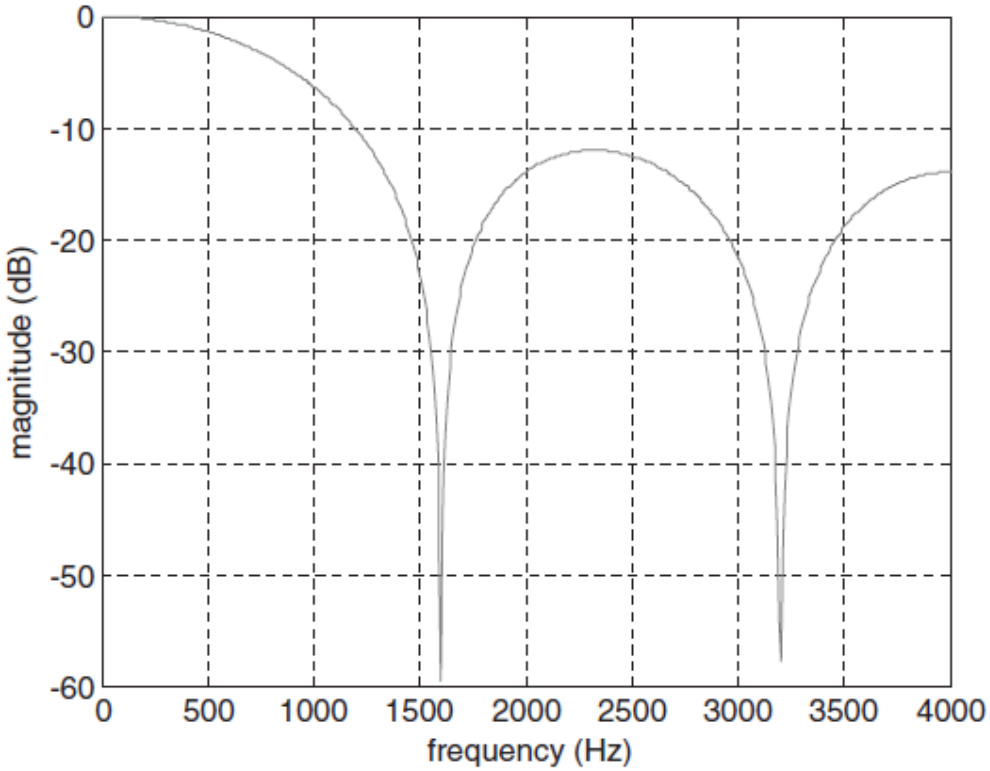
5)Verify the result.

PROGRAM :

//**average.c**

```
#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 :


0
-10
-20
-30
-40
-50
-60
magnitude (dB)
0
500
1000
1500
2000
2500
3000
3500
4000
frequency (Hz)

RESULT :

Thus the moving average filter was implemented.

EXP NO : 04	FIR IMPLEMENTATION WITH A PSEUDORANDOM NOISE SEQUENCE AS INPUT TO A FILTER.
DATE :	

AIM :

To implement the FIR with a pseudorandom noise sequence as input to a filter .

APPARATUS REQUIRED :

S.NO	NAME OF THE APPARATUS	SPECIFICATIONS	QUANTITY
01.	Personal Computer Setup	Windows 10	01
02.	Softwares Used	Matlab, Code Composer Studio (CCS) .	01
03.	Hardwares Used	DSP Trainer Kit, TMS320C54xx/TMS320C67xx DSP Development board, Function Generator and Digital Storage Oscilloscope , Microphone and speaker.	01

ALGORITHM :

1) Enter the number of cycles, period and amplitude for respective waves.

2) Generate the signals using corresponding general formula.

3) Plot the graph.

4)Verify the output.

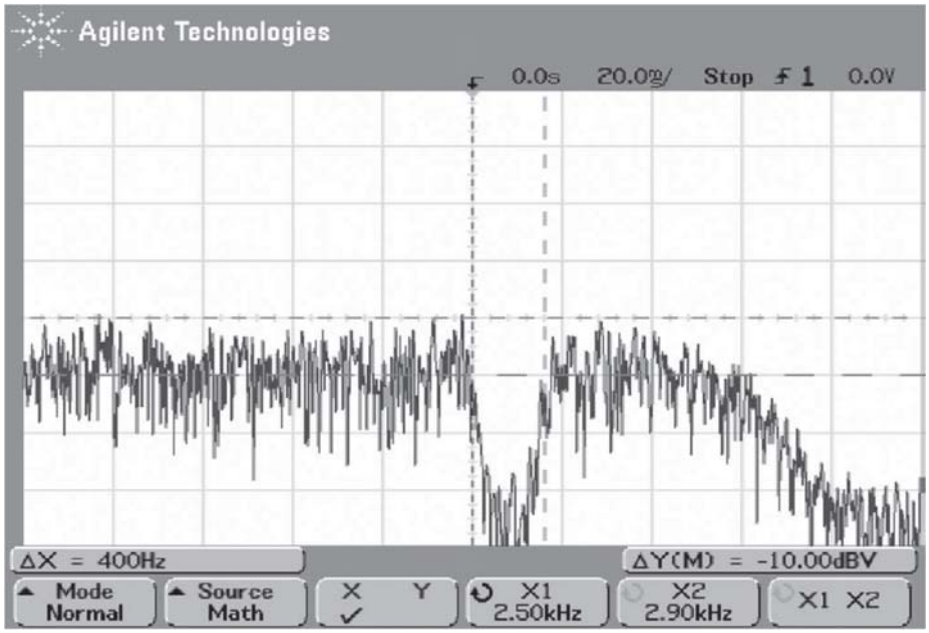
5)Verify the result.

PROGRAM :

DSK_FIR67.M

```
% MATLAB function to write FIR filter coefficients
% in format suitable for use in C6713 DSK programs
% firnc.c and firprn.c
% written by Donald Reay
%
function dsk_fir67(coeff)
%
coefflen=length(coeff);
fname = input('enter filename for coefficients ','s');
fid = fopen(fname,'wt');
fprintf(fid,'// %s\n',fname);
fprintf(fid,'// this file was generated automatically using
function
dsk_fir67.m\n',fname);
fprintf(fid,'\n#define N %d\n',coefflen);
fprintf(fid,'\nfloat h[N] = { \n');
% j is used to count coefficients written to current line
% in output file
j=0;
% i is used to count through coefficients
for i=1:coefflen
% if six coeffs have been written to current line
% then start new line
if j>5
j=0;
fprintf(fid,'\n');
end
% if this is the last coefficient then simply write
% its value to the current line
% else write coefficient value, followed by comma
if i==coefflen
fprintf(fid,'%2.4E',coeff(i));
else
fprintf(fid,'%2.4E,',coeff(i));
j=j+1;
end
end
fprintf(fid,'\n};\n');
fclose(fid);
```

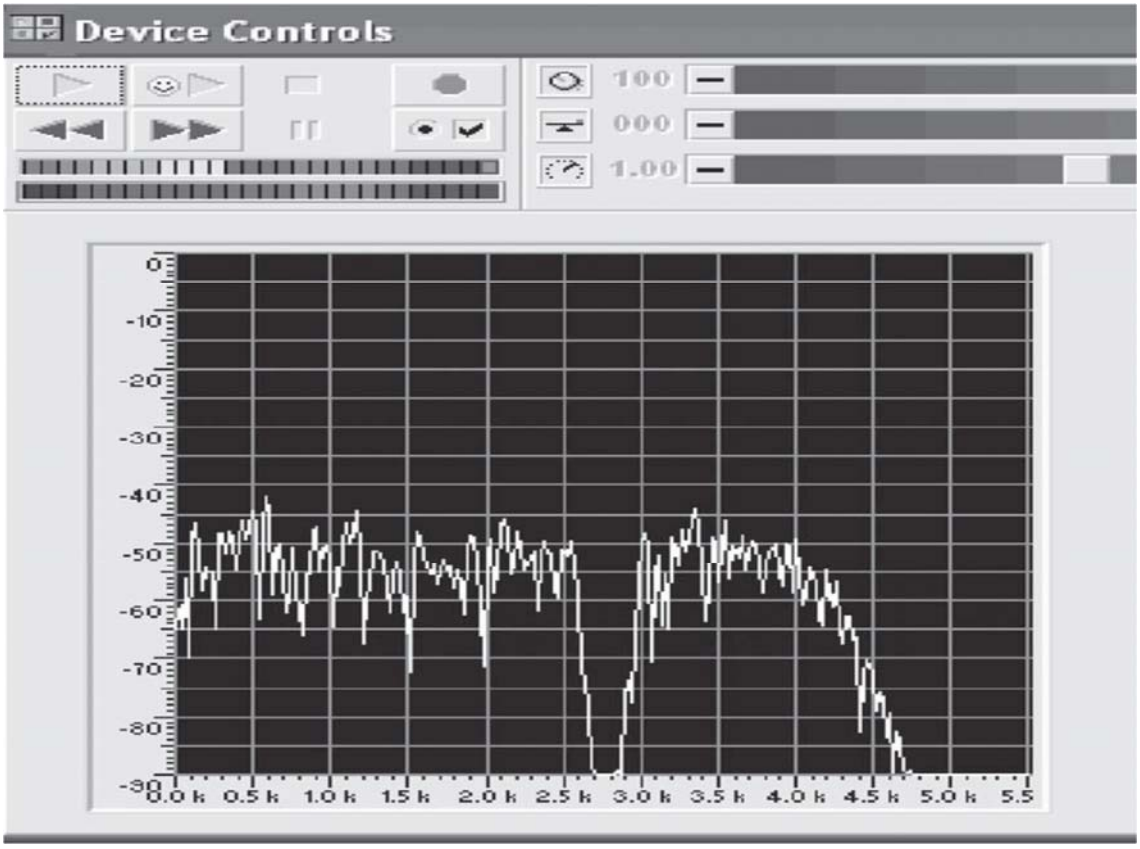
OUTPUT :


Agilent Technologies
0.0s
20.0µ/
Stop
1
0.0V
ΔX = 400Hz
ΔY(M) = -10.00dBV
Mode
Normal
Source
Math
X
Y
X1
2.50kHz
X2
2.90kHz
X1 X2

//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 :


Device Controls
100
000
1.00
0
-10
-20
-30
-40
-50
-60
-70
-80
-90
0.0 k
0.5 k
1.0 k
1.5 k
2.0 k
2.5 k
3.0 k
3.5 k
4.0 k
4.5 k
5.0 k
5.5

RESULT :

Thus the FIR with a pseudorandom noise sequence as input to a filter was implemented.

EXP NO : 05	FIXED POINT IMPLEMENTATION OF IIR FILTER
DATE :	

AIM :

To implement the fixed point implementation of IIR filter.

APPARATUS REQUIRED :

S.NO	NAME OF THE APPARATUS	SPECIFICATIONS	QUANTITY
01.	Personal Computer Setup	Windows 10	01
02.	Softwares Used	Matlab, Code Composer Studio (CCS) .	01
03.	Hardwares Used	DSP Trainer Kit, TMS320C54xx/TMS320C67xx DSP Development board, Function Generator and Digital Storage Oscilloscope , Microphone and speaker.	01

ALGORITHM :

1) Enter the number of cycles, period and amplitude for respective waves.

2) Generate the signals using corresponding general formula.

3) Plot the graph.

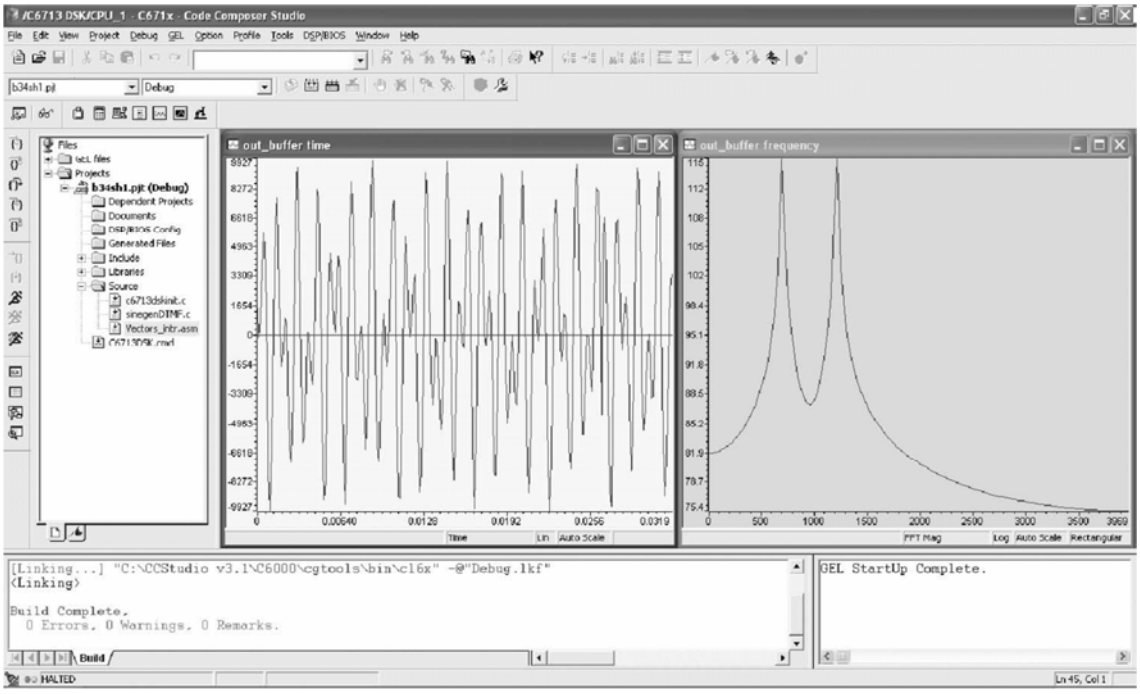
4)Verify the output.

5)Verify the result.

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;
#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 :


C6713 DSK/CPU_1 - C671x - Code Composer Studio
File Edit View Project Debug GEL Option Profile Tools DSP/BIOS Window Help
b34sh1.pjt
Debug
Files
GEL files
Projects
b34sh1.pjt (Debug)
Dependent Projects
Documents
DSP/BIOS Config
Generated Files
Include
Libraries
Source
c6713dskinit.c
sinegenDTMF.c
Vectors_intr.asm
out_buffer time
out_buffer frequency
Time
Lin
Auto Scale
FFT Mag
Log
Auto Scale
Rectangular
[Linking...] "C:\CCStudio v3.1\C6000\cgtools\bin\cl6x" -@"Debug.lkf"
<Linking>
Build Complete,
0 Errors, 0 Warnings, 0 Remarks.
GEL StartUp Complete.
Build
HALTED
Ln 45, Col 1

RESULT :

Thus the fixed point implementation of IIR filter was implemented.

EXP NO : 06	FFT OF REAL-TIME INPUT SIGNAL
DATE :	

AIM :

To implement the FFT of real time signal.

APPARATUS REQUIRED :

S.NO	NAME OF THE APPARATUS	SPECIFICATIONS	QUANTITY
01.	Personal Computer Setup	Windows 10	01
02.	Softwares Used	Matlab, Code Composer Studio (CCS) .	01
03.	Hardwares Used	DSP Trainer Kit, TMS320C54xx/TMS320C67xx DSP Development board, Function Generator and Digital Storage Oscilloscope , Microphone and speaker.	01

ALGORITHM :

1) Enter the number of cycles, period and amplitude for respective waves.

2) Generate the signals using corresponding general formula.

3) Plot the graph.

4)Verify the output.

5)Verify the result.

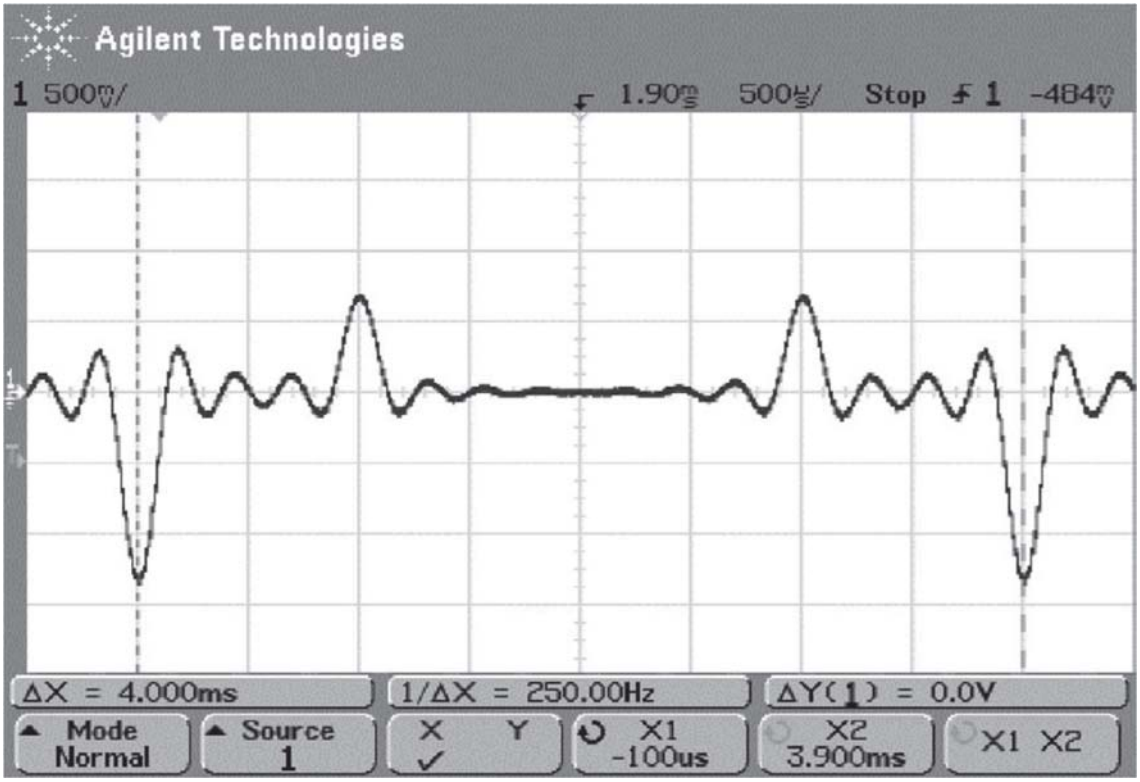
PROGRAM :

```
//fft128c.c
#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 <math.h>
#include "fft.h"
#define PI 3.14159265358979
#define TRIGGER 32000
#define N 128
#include "hamm128.h"
short buffercount = 0; //no of samples in iobuffer
short bufferfull = 0; //indicates buffer full
COMPLEX A[N], B[N], C[N];
COMPLEX *input_ptr, *output_ptr, *process_ptr, *temp_ptr;
COMPLEX twiddle[N];
short outbuffer[N];
interrupt void c_int11(void) //ISR
{
output_left_sample((short)((output_ptr + buffercount)->real));
outbuffer[buffercount] =
-(short)((output_ptr + buffercount)->real);
(input_ptr + buffercount)->real =
(float)(input_left_sample());
(input_ptr + buffercount++)->imag = 0.0;
if (buffercount >= N)
{
buffercount = 0;
bufferfull = 1;
}
```

```
}
```

```
main()
{
int n;
for (n=0 ; n<N ; n++) //set up twiddle factors
{
twiddle[n].real = cos(PI*n/N);
twiddle[n].imag = -sin(PI*n/N);
}
input_ptr = A;
output_ptr = B;
process_ptr = C;
comm_intr(); //initialise DSK
while(1) //frame processing loop

{
while(bufferfull==0); //wait for new frame
bufferfull = 0; //of input samples
temp_ptr = process_ptr; //rotate frame pointers
process_ptr = input_ptr;
input_ptr = output_ptr;
output_ptr = temp_ptr;
fft(process_ptr,N,twiddle); //process contents of buffer
for (n=0 ; n<N ; n++) // compute magnitude
{ // and place in real part
(process_ptr+n)->real =
-sqrt((process_ptr+n)->real*(process_ptr+n)->real
+ (process_ptr+n)->imag*(process_ptr+n)->imag)/16.0;
}
(process_ptr)->real = TRIGGER; // add oscilloscope trigger
} //end of while(1)
}
```

OUTPUT :


Agilent Technologies
1 500mV/
1.90ms
500Hz/
Stop
1
-484mV
ΔX = 4.000ms
1/ΔX = 250.00Hz
ΔY(1) = 0.0V
Mode
Normal
Source
1
X
Y
X1
-100us
X2
3.900ms
X1 X2

RESULT :

Thus the FFT of real time signal was implemented.