R.V. College of Engineering
Bengaluru - 59

(Autonomous Institution affiliated to VTU, Belagavi)

Department of Electronics and Communication Engineering

Communication Systems-I
16EC52

Laboratory Manual

(Autonomous Scheme 2016)

R.V. College of Engineering, Bengaluru - 59

(Autonomous Institution affiliated to VTU, Belagavi)

Department of Electronics and Communication Engineering

Laboratory Certificate

This is to certify that Mr. / Ms ______________________________ ______________________________has satisfactorily completed the course of Experiments in Practical ______________________________ prescribed by the Department during the year ____________________

Name of the Candidate: ____________________________________

USN No.: __________________ Semester:___________________

Marks	
Max. Marks	Obtained
50	

Marks in Words	

Signature of the staff in-charge

Head of the Department

Date:

SCHEME OF CONDUCT AND EVALUATION

CLASS: V SEMESTER CIE MARKS: (Max.) : 50

YEAR: 2018-19 SEE MARKS: (Max.) : 50

Expt. No.	**TITLE**	**Duration in Hrs**	**Max. Marks**	**Marks Obtained**
1.	Frequency Modulation and Demodulation	2.30	10	
2.	Verification of Sampling theorem	2.30	10	
3.	Implementation of Convolution and DFT	2.30	10	
4.	Realization of FIR filter to meet given specifications	2.30	10	
5.	Realization of IIR filter to meet given specifications	2.30	10	
6.	Linear block code and Huffman code	2.30	10	
7.	Time Division Multiplexing	2.30	10	
8.	Pulse Code Modulation& Delta Modulation	2.30	10	
9.	Generation of Noise and study of its properties	2.30	10	
10.	Line codes generation and Pe and PSD calculation	2.30	10	
	TEST	2.30	**50**	
	TOTAL MARKS			

Rubrics for Evaluation

Sl. No	Criteria	Excellent	Good	Average	Max Score
Data sheet					
A	Problem statement	9-10	6-8	1-5	**10**
B	Design & specifications	9-10	6-8	1-5	**10**
C	Expected output	9-10	6-8	1-5	**10**
Record					
D	Simulation/ Conduction of the experiment	14-15	11-13	1-10	**15**
E	Analysis of the result	14-15	11-13	1-10	**15**
Viva					**40**
Total					**100**
Scale down to 10 marks					

Department Vision

Imparting quality technical education through interdisciplinary research, innovation and teamwork for developing inclusive & sustainable technology in the area of Electronics and Communication Engineering.

Department Mission

- To impart quality technical education to produce industry-ready engineers with a research outlook.
- To train the Electronics & Communication Engineering graduates to meet future global challenges by inculcating a quest for modern technologies in the emerging areas.
- To create centres of excellence in the field of Electronics & Communication Engineering with industrial and university collaborations.
- To develop entrepreneurial skills among the graduates to create new employment opportunities.

COMMUNICATION SYSTEMS-I LAB (12EC53)

List of Experiments:

1) Frequency Modulation and Demodulation (Matlab)
2) Verification of Sampling theorem (Matlab & Circuit)
3) Implementation of Convolution and DFT (DSP kit)
4) Realization of FIR filter to meet given specifications (DSP kit)
5) Realization of FIR filter to meet given specifications (DSP kit)
6) Linear block code and Huffman code (Matlab)
7) Time Division Multiplexing (Matlab & Circuit)
8) PCM & DM(Matlab & Simulink)
9) Generation of Noise and study of its properties (Matlab)
10) Line codes generation and P_e and PSD calculation

Course Outcomes: **The students will be able to**
1. Associate and apply the concepts of digital formatting, reconstruction to digital transmitters and receivers used in cellular and other communication devices.
2. Analyze and compute performance of continuous wave modulation, digital formatting schemes.
3. Test and validate digital formatting schemes and block codes under noisy channel conditions to estimate the performance in practical communication systems.
4. Design/Demonstrate by way of simulation or emulation of different functional blocks of formatting and block error correction.

CO-PO Mapping

CO/PO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	M	L	M	L	M	L	-	-	M	L	-	M
CO2	M	L	M	L	M	L	-	-	M	L	-	M
CO3	M	L	M	L	M	L	L	L	M	L	-	M
CO4	M	L	M	L	M	L	L	L	M	L	-	M

EVALUATION SCHEME:

Particulars of the Experiments Performed

CONTENTS

Sl. No.	Date	Experiments	Page No.	Marks Obtained	Signature of Staff
1		Frequency Modulation and Demodulation			
2		Verification of Sampling theorem			
3		Implementation of Convolution and DFT			
4		Realization of FIR filter to meet given specifications			
5		Realization of IIR filter to meet given specifications			
6		Linear block code and Huffman code			
7		Time Division Multiplexing			
8		Pulse Code Modulation& Delta Modulation			
9		Generation of Noise and study of its properties			
10		Line codes generation and Pe and PSD calculation			
Total					
TEST (50)					

The list of steps to implement the c code on DSK 6713 kit using CC studio version 5.5.

DSP Starter Kit (DSK) 6713 in Code Composer Studio V5.5

1. Open Code composer Studio (CCS) using short cut of CCS 5.5.0

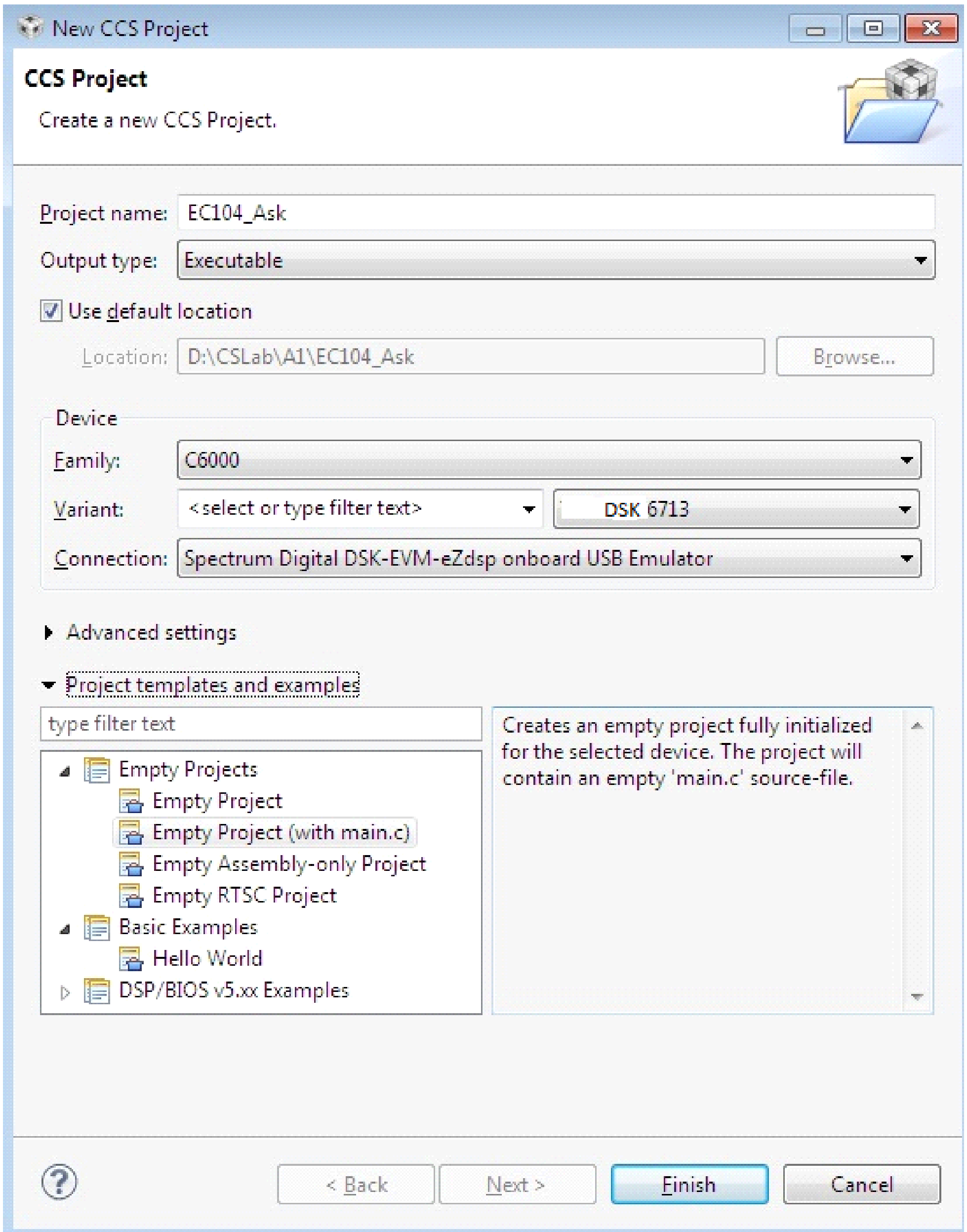
2. Choose the Workspace location (Preferably on D Drive like D:\CSLab\A1, D:\CSLab\A2....)

3. Create new project by menu ↵Project↵New CCS Project

4. Enter Project name USN_Exp_name (e.g. EC104_ conv. Do not start the project name with a number).

Output type: Executable(default)
Device Family:C6000
Variant:C671x Floating-point DSP, DSK6713
Connection: Spectrum Digital DSK-EVM-eZdsp on board Emulator
Project template and examples: Empty Project(with main.c)

Advanced Setting:
Linker Command file: C6713dsk.cmd (Browse C:\dsk6713\support)
Click finish↵ (Fig.1)



New CCS Project

CCS Project

Create a new CCS Project.

Project name: EC104_Ask

Output type: Executable

☑ Use default location

Location: D:\CSLab\A1\EC104_Ask

Device

Family: C6000

Variant: <select or type filter text> DSK 6713

Connection: Spectrum Digital DSK-EVM-eZdsp onboard USB Emulator

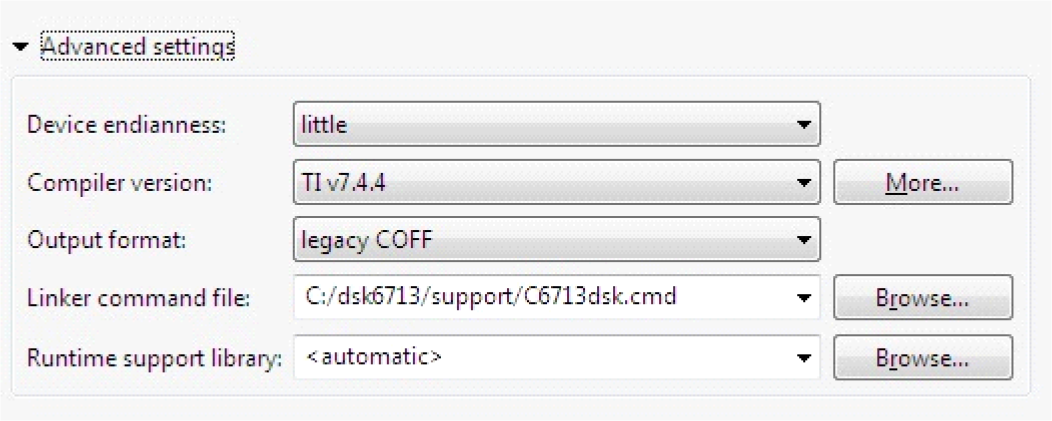
Advanced settings

Project templates and examples

- Empty Projects
  - Empty Project
  - Empty Project (with main.c)
  - Empty Assembly-only Project
  - Empty RTSC Project
- Basic Examples
  - Hello World
- DSP/BIOS v5.xx Examples

Creates an empty project fully initialized for the selected device. The project will contain an empty 'main.c' source-file.

Advance Setting,



Advanced settings

Device endianness: little

Compiler version: TI v7.4.4

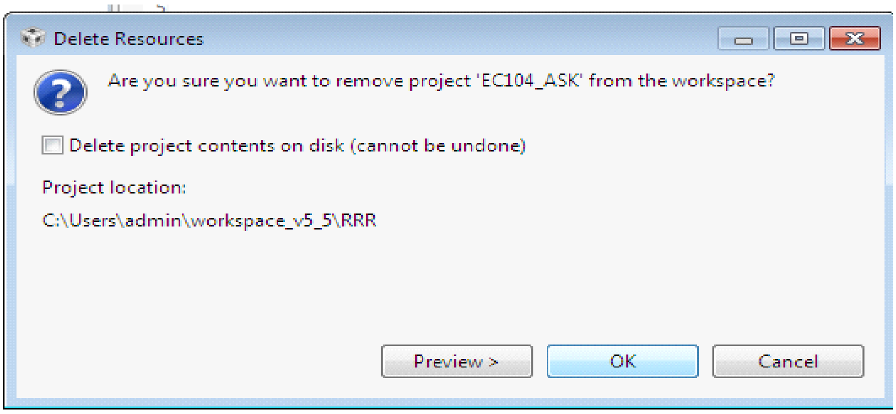
Output format: legacy COFF

Linker command file: C:/dsk6713/support/C6713dsk.cmd

Runtime support library: <automatic>

Fig.1 Creating New Project

5.If any previous project exist in project explorer, it can be deleted by right click ↵ on the previous project & click on delete↵, then give "Ok" on pop-up menu (**Note:** Do not select the option **Delete the Project contents on disk**.(Fig.2) It will delete someone project files)



Delete Resources

Are you sure you want to remove project 'EC104_ASK' from the workspace?

☐ Delete project contents on disk (cannot be undone)

Project location:

C:\Users\admin\workspace_v5_5\RRR

Fig.2. Deletion of previous project.

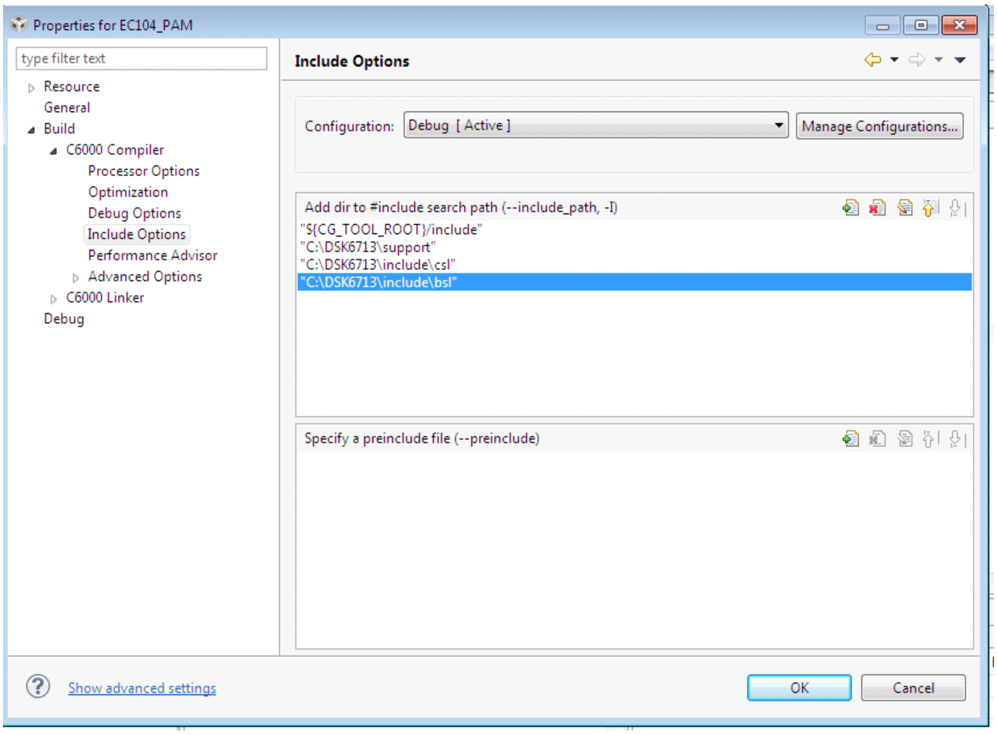
6. Start to write the source program in main.c & save by File↵Save↵

7. **C6000 Compiler setting**, Go Project↵Properties↵ Build↵C6000 Compiler↵ Include Options↵

a) Include Support files, Chip select (CSL) & Board support (BSL) files, (Fig.3)

↵Add path (by option , In Add dir to #include search path (--include_path,-I))

- C:\DSK6713\support
- C:\DSK6713\include\csl
- C:\DSK6713\include\bsl



Configuration: Debug [Active] Manage Configurations...

Add dir to #include search path (--include_path, -I)

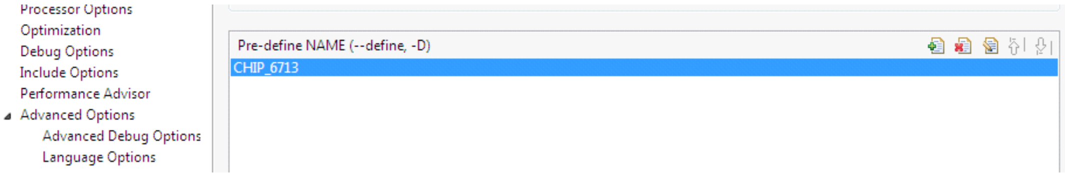
"${CG_TOOL_ROOT}/include"
"C:\DSK6713\support"
"C:\DSK6713\include\csl"
"C:\DSK6713\include\bsl"

Specify a preinclude file (--preinclude)

Fig.3. Inclusion of chip select, board support files in C6700 Compiler.

b) In Advanced Options,

Predefined Symbols:CHIP_6713 (Remove any previous symbols by delete ✖))(Fig.4)



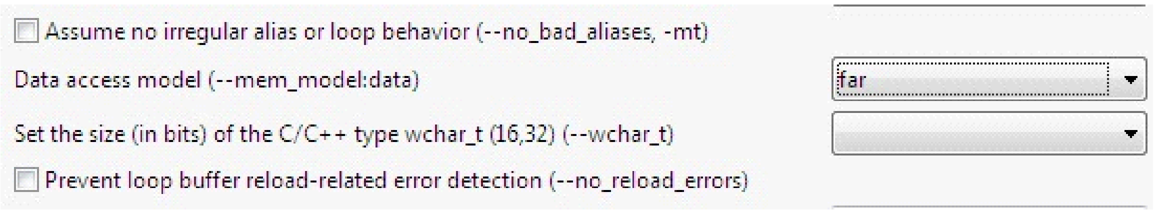
Pre-define NAME (--define, -D)

CHIP_6713

Fig.4. Define predefined symbol in C6700 Compiler.

Runtime Model Options:

Data Access model: far (Fig.5)



□ Assume no irregular alias or loop behavior (--no_bad_aliases, -mt)

Data access model (--mem_model:data) far

Set the size (in bits) of the C/C++ type wchar_t (16,32) (--wchar_t)

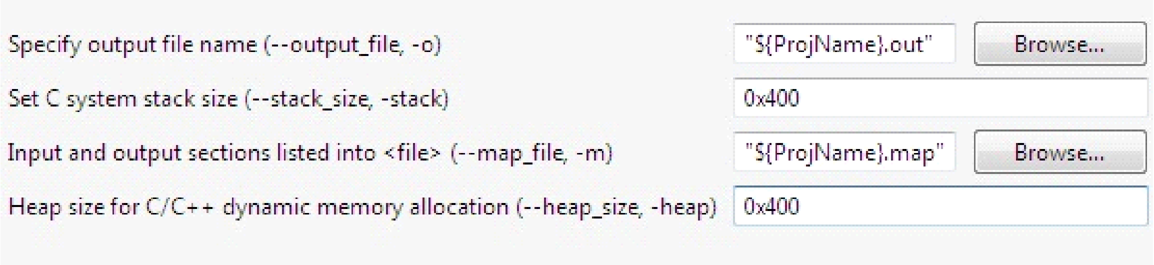
□ Prevent loop buffer reload-related error detection (--no_reload_errors)

Fig.5. Define Runtime Memory Model in C6700 Compiler.

8. C6000 Linker Settings:

a) Under Basic option:(Fig.6)

- Set C system stack size: 0x400
- Set C system stack size: 0x400



Specify output file name (--output_file, -o) "${ProjName}.out"

Set C system stack size (--stack_size, -stack) 0x400

Input and output sections listed into <file> (--map_file, -m) "${ProjName}.map"

Heap size for C/C++ dynamic memory allocation (--heap_size, -heap) 0x400

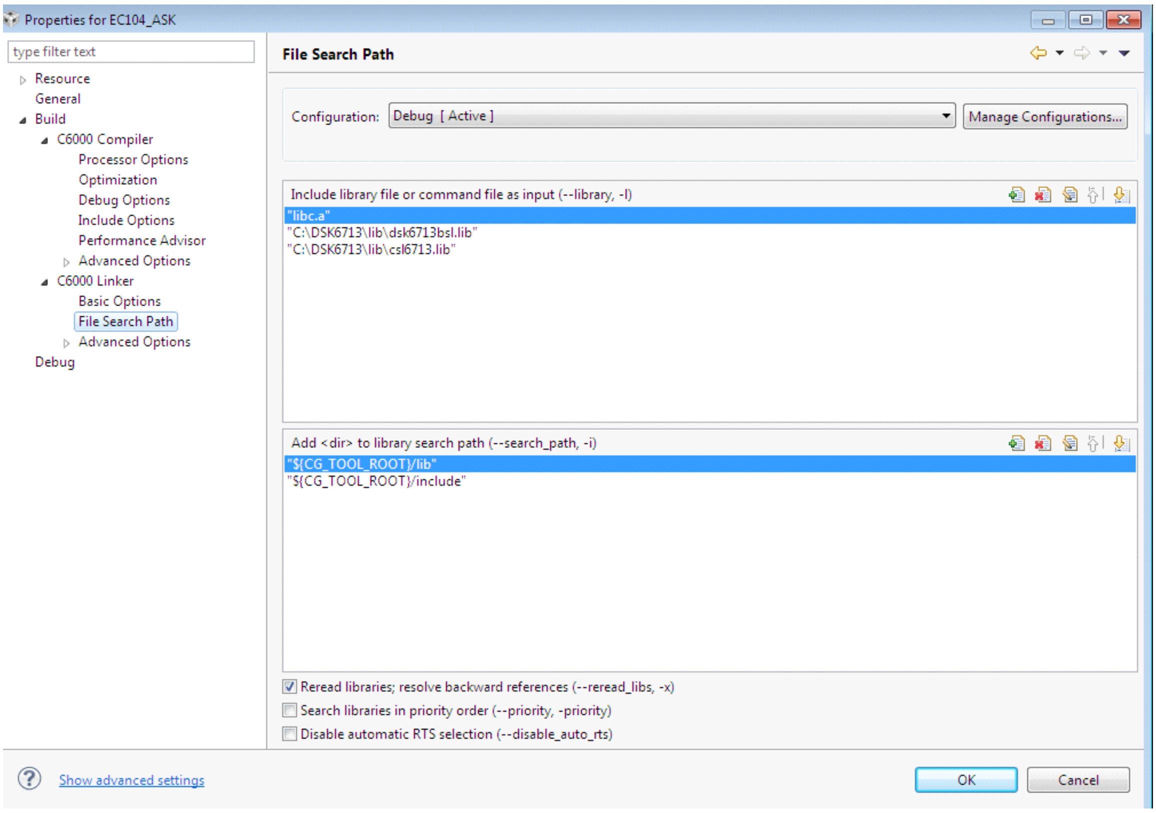
Fig.6. Define stack & heap size in C6700 Compiler.

b) In File Search Path: (Fig.7)

Under Include file or command file as input(--library, -I)

Add following Library Paths:((by option)

- C:\DSK6713\lib\dsk6713bsl.lib
- C:\DSK6713\lib\csl6713.lib



Properties for EC104_ASK

File Search Path

Configuration: Debug [Active]

Include library file or command file as input (--library, -l)

"libc.a"
"C:\DSK6713\lib\dsk6713bsl.lib"
"C:\DSK6713\lib\csl6713.lib"

Add <dir> to library search path (--search_path, -i)

"${CG_TOOL_ROOT}/lib"
"${CG_TOOL_ROOT}/include"

☑ Reread libraries; resolve backward references (--reread_libs, -x)

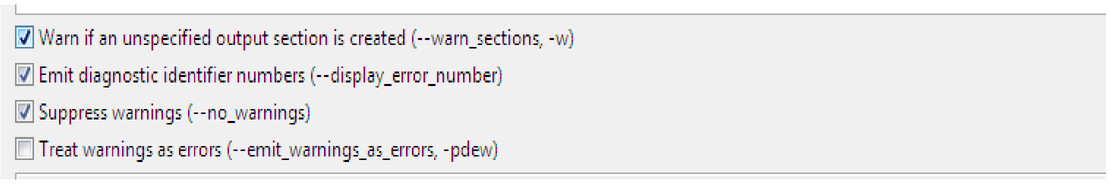
□ Search libraries in priority order (--priority, -priority)

□ Disable automatic RTS selection (--disable_auto_rts)

Fig.7. Inclusion of Board & chip support libraries in C6700 Linker

c) In Advanced Options:

In Diagnostics: **Choose Suppress warning (--no warning)** (Fig.8)



☑ Warn if an unspecified output section is created (--warn_sections, -w)
☑ Emit diagnostic identifier numbers (--display_error_number)
☑ Suppress warnings (--no_warnings)
☐ Treat warnings as errors (--emit_warnings_as_errors, -pdew)

Fig.8. C6700 Linker Diagnostic option

Then Click **Ok**↵

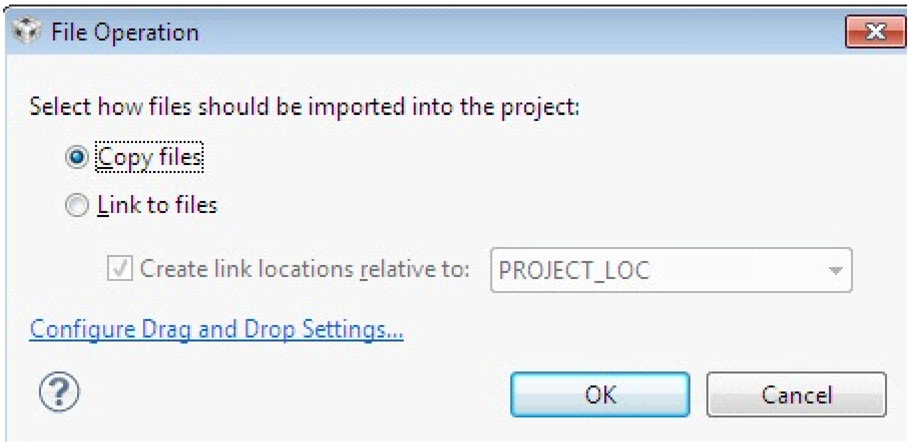
9. Add Support files to Project

Project↵Add Files..↵ Then browse to C:\DSK6713\support

add following files(c file & assembly file)

- C6713dskinit.c
- C6713dskinit.h
- Vectors_intr.asm

& choose File operation as Copy files then enter Ok↵ (Fig.9)



File Operation

Select how files should be imported into the project:

● Copy files
○ Link to files
☑ Create link locations relative to: PROJECT_LOC

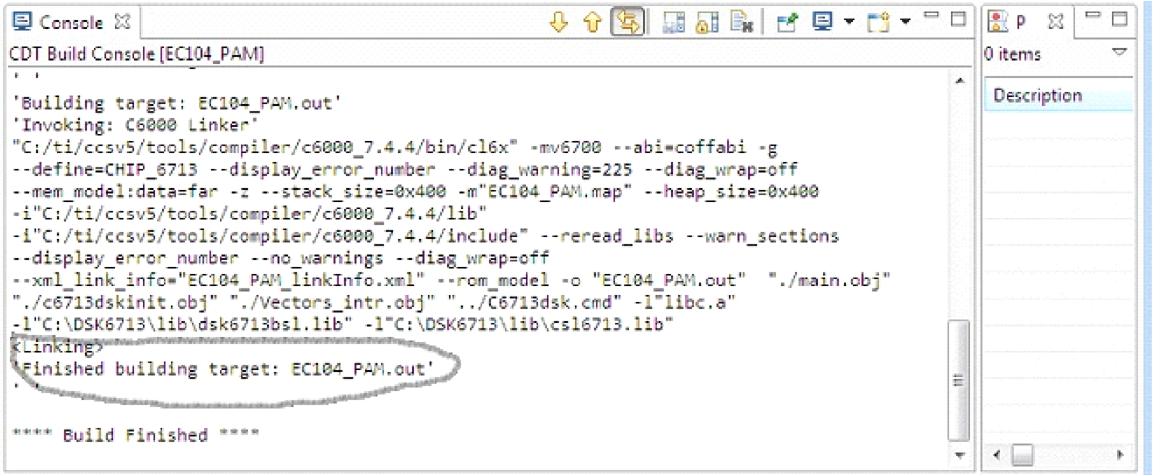
Fig.9 Copy Support Files to project

10. Build the Project:

Manual Build: Project ↵Build Automatically (Unclick)

Project ↵Build Project↵ (or mouse-click 🔨)

Project is build successfully & output file(.out) is generated & can be seen in console,(Fig. 10)



```
' '
'Building target: EC104_PAM.out'
'Invoking: C6000 Linker'
"C:/ti/ccsv5/tools/compiler/c6000_7.4.4/bin/cl6x" -mv6700 --abi=coffabi -g
--define=CHIP_6713 --display_error_number --diag_warning=225 --diag_wrap=off
--mem_model:data=far -z --stack_size=0x400 -m"EC104_PAM.map" --heap_size=0x400
-i"C:/ti/ccsv5/tools/compiler/c6000_7.4.4/lib"
-i"C:/ti/ccsv5/tools/compiler/c6000_7.4.4/include" --reread_libs --warn_sections
--display_error_number --no_warnings --diag_wrap=off
--xml_link_info="EC104_PAM_linkInfo.xml" --rom_model -o "EC104_PAM.out"  "./main.obj"
"./c6713dskinit.obj" "./Vectors_intr.obj" "../C6713dsk.cmd" -l"libc.a"
-l"C:\DSK6713\lib\dsk6713bsl.lib" -l"C:\DSK6713\lib\csl6713.lib"
<Linking>
'Finished building target: EC104_PAM.out'
' '

**** Build Finished ****
```

Fig. 10 After successful Build Console Window

11. Connection of DSK Hardware & Diagnostic Test

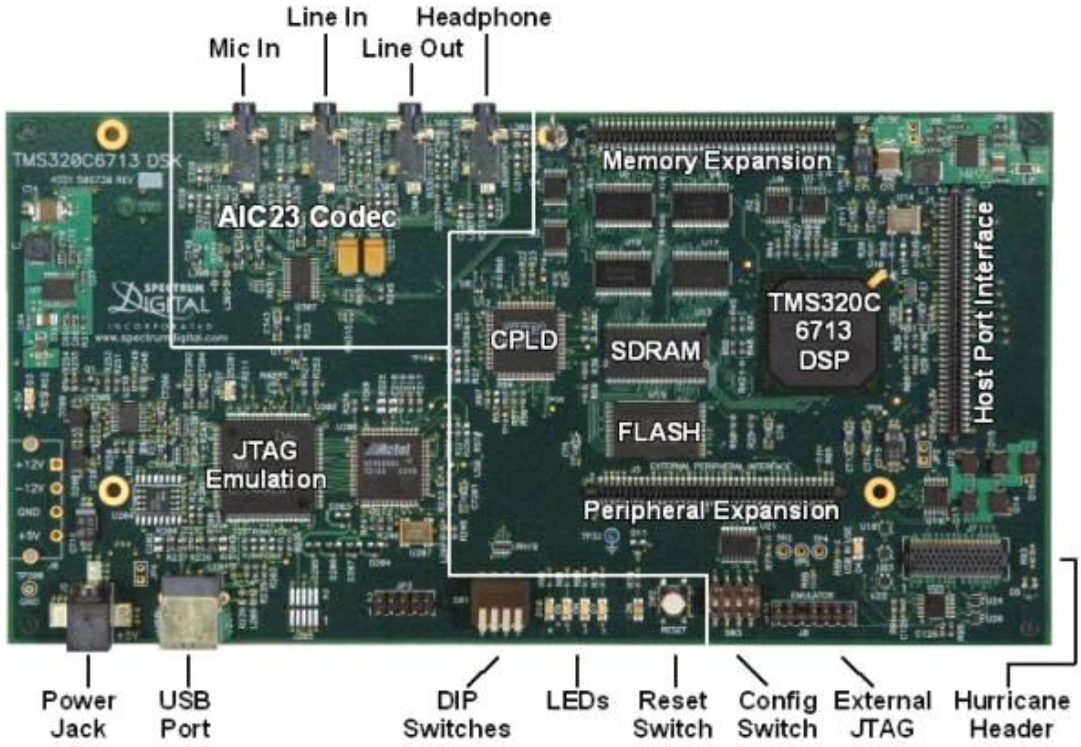
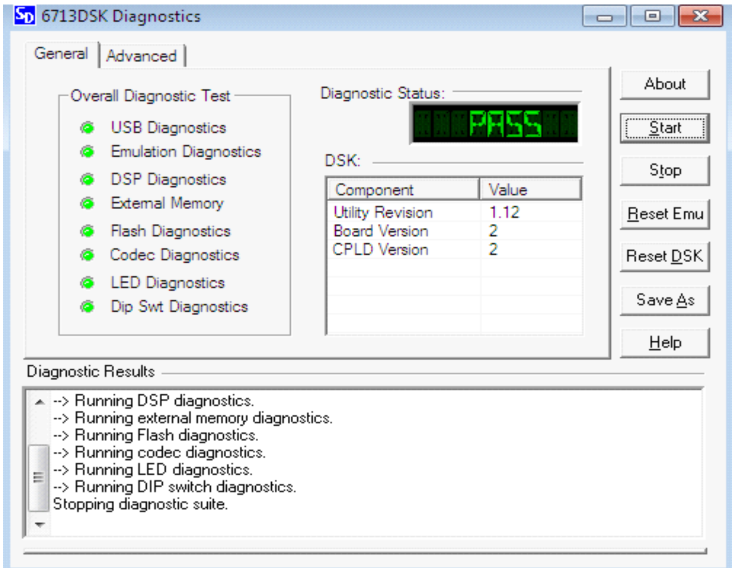


Fig. 11 Block diagram of DSK 6713 Board

- ❖ Connect Power supply
- ❖ Connect the USB cable with PC.
- ❖ Run the Diagnostic Test :(From Desktop icon)

Pop-up menu User Access control: Yes &

choose Start↵After successful test ,close the window (Fig. 12)



Overall Diagnostic Test

- ● USB Diagnostics
- ● Emulation Diagnostics
- ● DSP Diagnostics
- ● External Memory
- ● Flash Diagnostics
- ● Codec Diagnostics
- ● LED Diagnostics
- ● Dip Swt Diagnostics

Diagnostic Status: PASS

DSK:

Component	Value
Utility Revision	1.12
Board Version	2
CPLD Version	2

Diagnostic Results

```
--> Running DSP diagnostics.
--> Running external memory diagnostics.
--> Running Flash diagnostics.
--> Running codec diagnostics.
--> Running LED diagnostics.
--> Running DIP switch diagnostics.
Stopping diagnostic suite.
```

Fig. 12 Diganostic Window of C6713

12. Connection of Input & Output signals

- ❖ Connect the Function generator with CRO through stereo cable and choose the Sine signal & verify that input signal amplitude <1.5 V & Frequency less than Sampling frequency/2 (Sampling frequency can be found in the program)

(Imp Note: the input voltage to the DSK Board should not exceed more than 1.5 V). Verify the input voltage in the CRO before connecting to DSP Kit

- ❖ Connect the Function generator to **LINE IN** in the DSP Kit through Stereo cable (Fig. 13)
- ❖ Connect the CRO to **LINE OUT** in the DSP Kit through stereo cable (Fig. 13)

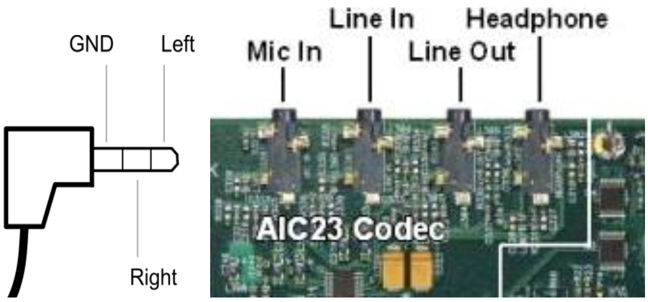


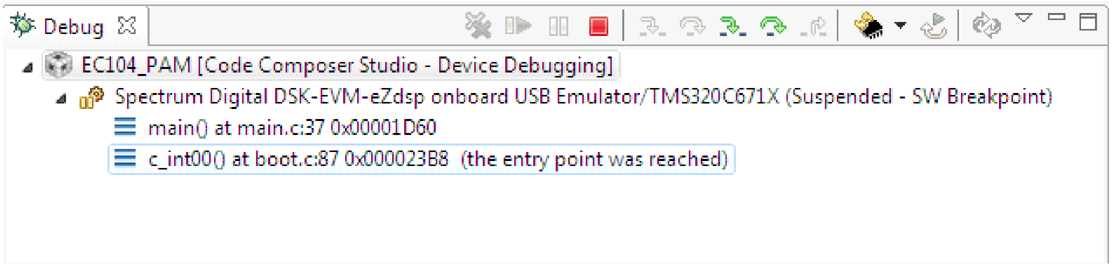
Fig. 13 Stereo pin configuration & AIC23 Codec interface

13. Debug the Program

Go CCS V5.5, Select Run↵Debug↵ (or mouse-click)

The Spectrum Digital DSK-EVM-eZdsp onboard USB Emulator is connected to DSK 6713 Target board successfully. (If the target is not connected check whether a Diagnostics Window (In Step 11) is closed or unplug & reconnect the power supply)

Screenshot On successful connection of Target (Fig. 14)



EC104_PAM [Code Composer Studio - Device Debugging]
Spectrum Digital DSK-EVM-eZdsp onboard USB Emulator/TMS320C671X (Suspended - SW Breakpoint)
main() at main.c:37 0x00001D60
c_int00() at boot.c:87 0x000023B8 (the entry point was reached)

Fig. 14 Successful Connection of DSK 6713 Target board

14. Run the Program,

Go to Run↵Resume↵

15. Verify the waveform in the CRO.

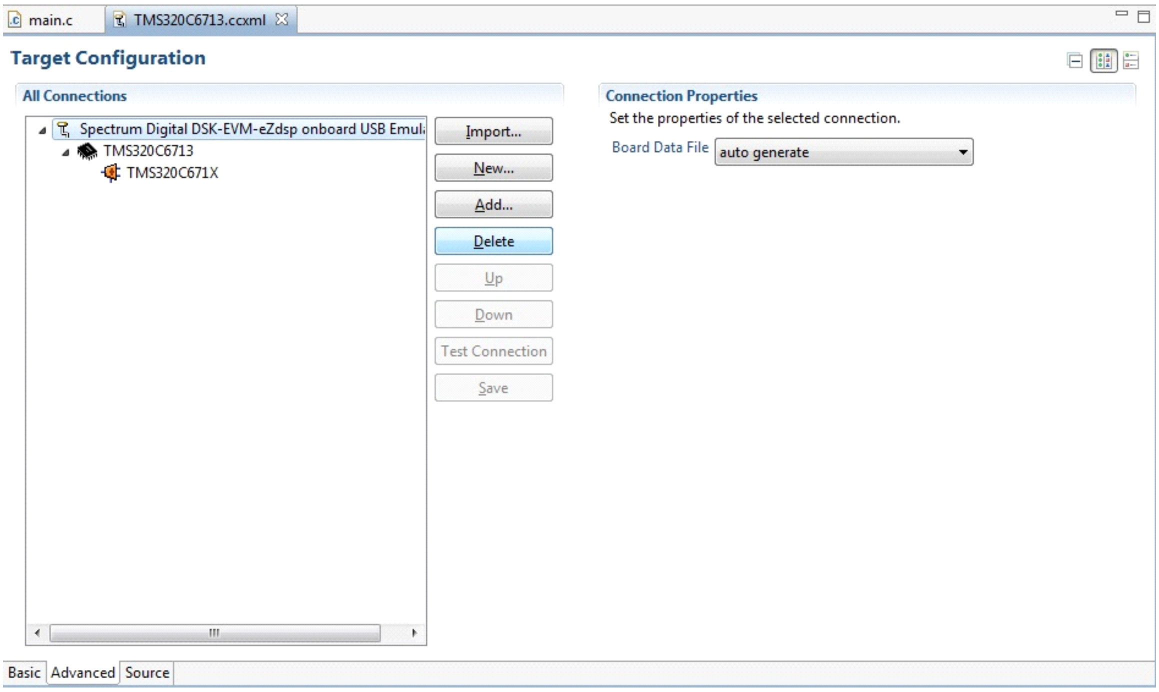
DSP Starter Kit (DSK) 6713 in Code Composer Studio V5.5 (Simulator Mode)

Procedure:

After Step 9 of DSP Starter kit procedure given above, follow the below instructions

↵Under targetConfigs↵TMS320C6713.ccxml

Under Advanced Tab



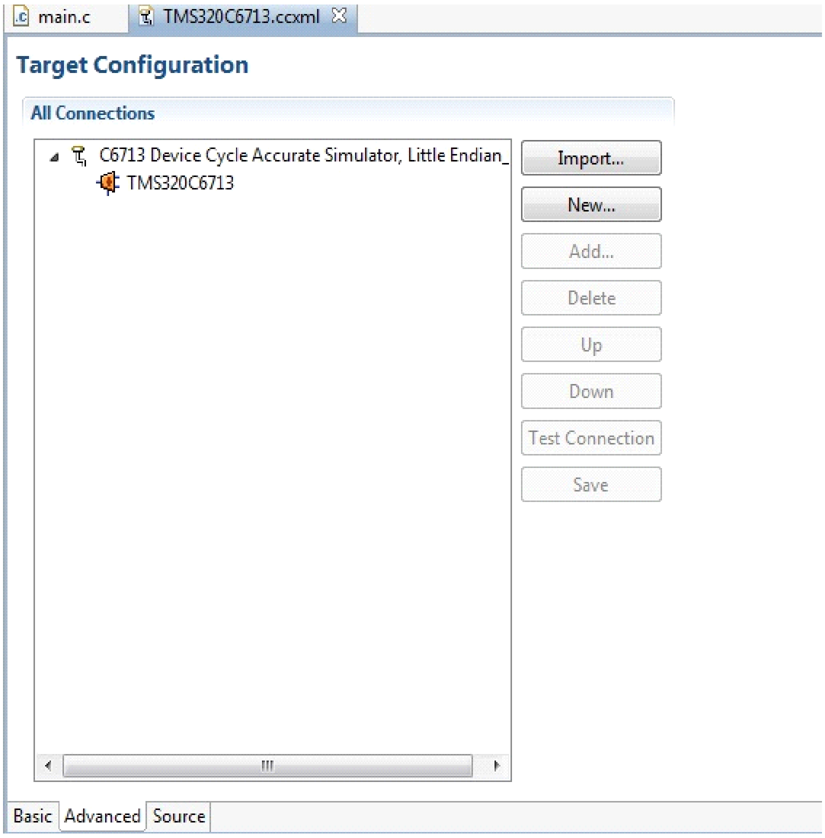
Set the properties of the selected connection.

Board Data File: auto generate

Fig.1. Existing target configuration of C6713

Delete existing target configuration, ,(Fig.1)

↵Import & add C6713 Device Cycle Accurate Simulator, Little Endian,(Fig.2)



main.c TMS320C6713.ccxml

Target Configuration

All Connections

C6713 Device Cycle Accurate Simulator, Little Endian_
TMS320C6713

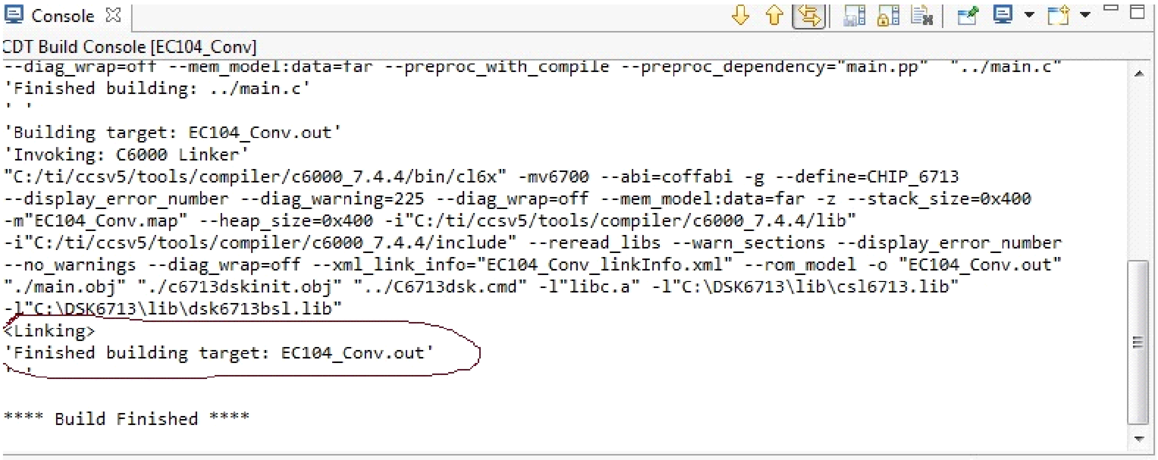
Fig.2. Configuration for Simulator Mode of C6713 Device Cycle Accurate Simulator

Build the Project:

Manual Build: Project ↵Build Automatically (Unclick)

Project ↵Build Project↵ (or mouse-click)

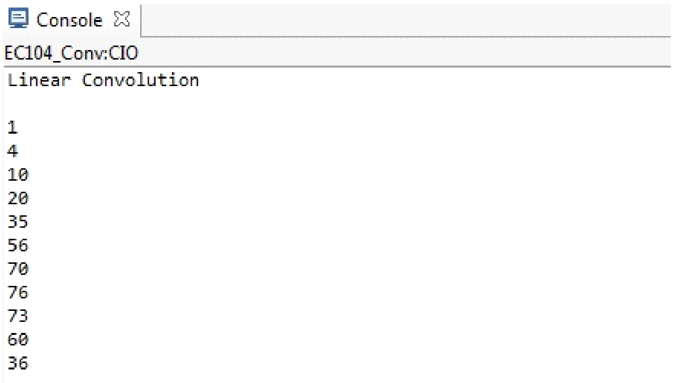
Project is build successfully & output file(.out) is generated & can be seen in console,(Fig. 3)



```
CDT Build Console [EC104_Conv]
--diag_wrap=off --mem_model:data=far --preproc_with_compile --preproc_dependency="main.pp"  "../main.c"
'Finished building: ../main.c'
' '
'Building target: EC104_Conv.out'
'Invoking: C6000 Linker'
"C:/ti/ccsv5/tools/compiler/c6000_7.4.4/bin/cl6x" -mv6700 --abi=coffabi -g --define=CHIP_6713
--display_error_number --diag_warning=225 --diag_wrap=off --mem_model:data=far -z --stack_size=0x400
-m"EC104_Conv.map" --heap_size=0x400 -i"C:/ti/ccsv5/tools/compiler/c6000_7.4.4/lib"
-i"C:/ti/ccsv5/tools/compiler/c6000_7.4.4/include" --reread_libs --warn_sections --display_error_number
--no_warnings --diag_wrap=off --xml_link_info="EC104_Conv_linkInfo.xml" --rom_model -o "EC104_Conv.out"
"./main.obj" "./c6713dskinit.obj" "../C6713dsk.cmd" -l"libc.a" -l"C:\DSK6713\lib\csl6713.lib"
-l"C:\DSK6713\lib\dsk6713bsl.lib"
<Linking>
'Finished building target: EC104_Conv.out'
' '

**** Build Finished ****
```

Fig.3. After successful Build Console Window

Output can be verifed in Console Window,(Fig.4)



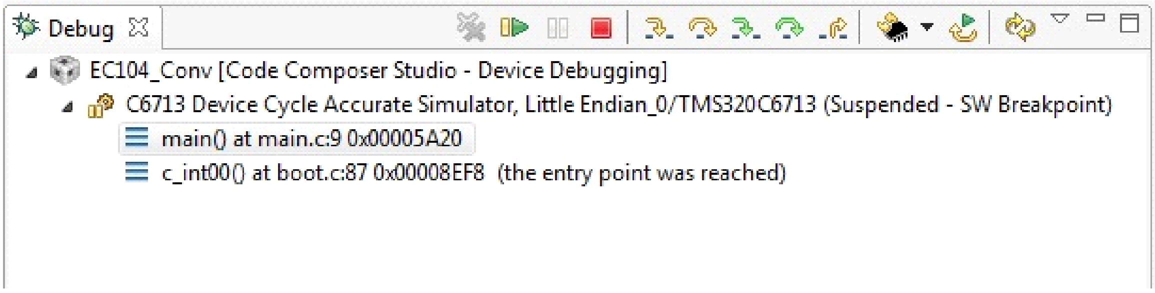
```
EC104_Conv:CIO
Linear Convolution

1
4
10
20
35
56
70
76
73
60
36
```

Fig.4. Output in Console Window

Debug the Program

Go CCS V5.5, Select Run↵Debug↵ (or mouse-click)

Screenshot On successful connection of Target



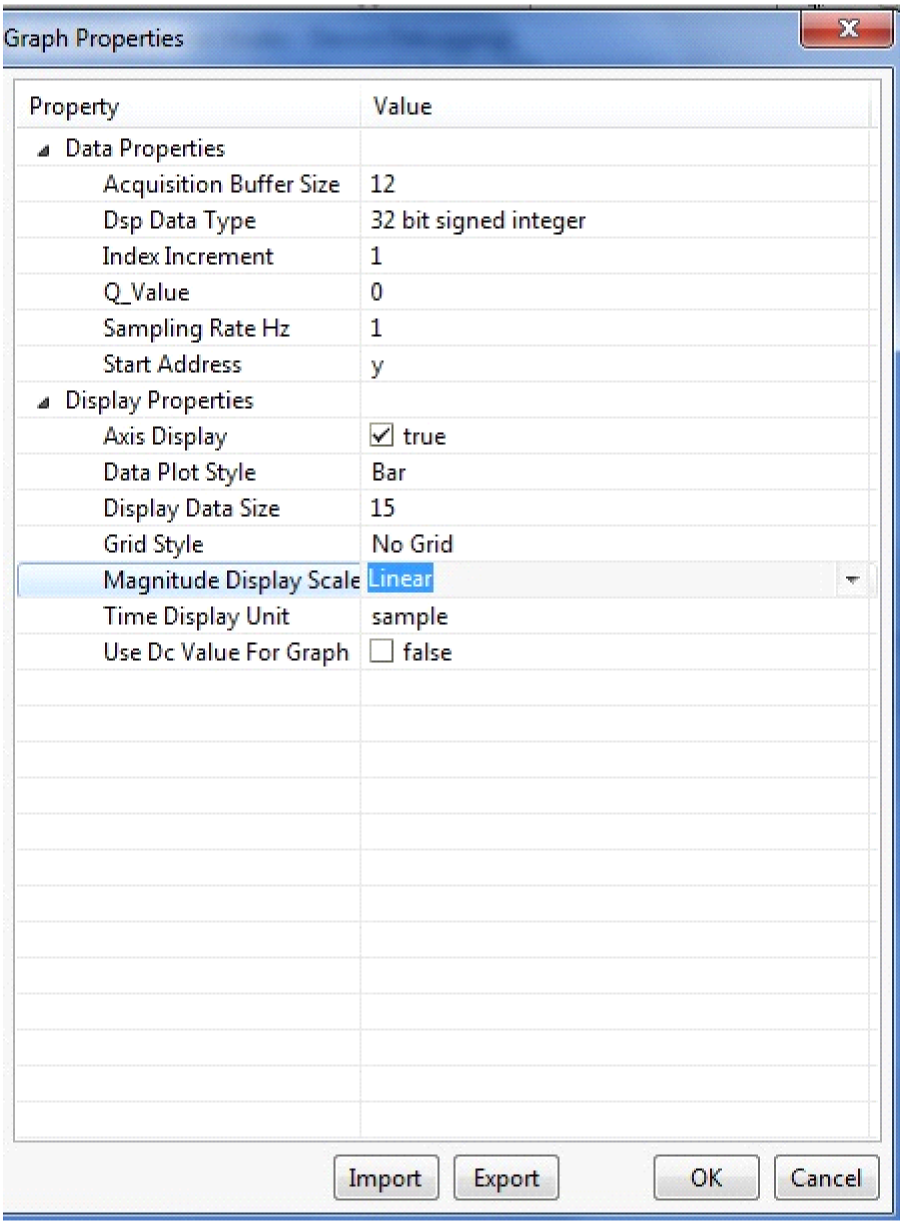
```
EC104_Conv [Code Composer Studio - Device Debugging]
  C6713 Device Cycle Accurate Simulator, Little Endian_0/TMS320C6713 (Suspended - SW Breakpoint)
    main() at main.c:9 0x00005A20
    c_int00() at boot.c:87 0x00008EF8  (the entry point was reached)
```

Fig.5. Successful Connection of DSK 6713 Target board

To view graph

↵Tools↵Graph ↵Single Time

Then set the values according to input values (Example as shown in following figure)



Property	Value
◢ Data Properties	
Acquisition Buffer Size	12
Dsp Data Type	32 bit signed integer
Index Increment	1
Q_Value	0
Sampling Rate Hz	1
Start Address	y
◢ Display Properties	
Axis Display	☑ true
Data Plot Style	Bar
Display Data Size	15
Grid Style	No Grid
Magnitude Display Scale	Linear
Time Display Unit	sample
Use Dc Value For Graph	□ false

Fig.6. Graph Single Time Setting

Experiment 1

Frequency Modulation and Demodulation

Aim:

Simulate and interpret the Spectrum of Message and Frequency modulated signal.

Theory:

Angle modulation:

This is one of the methods of modulating a sinusoidal carrier wave, namely, angle modulation in which either the phase or frequency of the carrier wave is varied according to the message signal. In this method of modulation the amplitude of the carrier wave is maintained constant.

In general the angle modulated wave is given by

$$s(t)=A_c \operatorname{Cos}(\theta(t))$$

Where $A_c$is the amplitude of the carrier which is constant and a angular argument θ(t)) is varied by a message signal m(t).

Frequency Modulation:

Frequency Modulation (FM) is the form of angle modulation in which the instantaneous frequency $f_i(t)$ is varied linearly with a message signal m(t). The instantaneous frequency is given by

$$f_i(t) = f_c + \frac{k_f}{2\pi} m(t)$$

The term fc represents the frequency of the unmodulated carrier and constant $K_f$represents the frequency sensitivity of the modulator. We have

$$f_i(t) = \frac{1}{2\pi} \frac{d\theta(t)}{dt}$$

Therefore

$$\theta(t) = 2\pi fct + Kf \int_0^t m(t)\, dt$$

Frequency modulated wave in the time domain is given by

$$S(t) = A_c \operatorname{Cos} [2\pi fct + Kf \int_0^t m(t)\, dt]$$

Spectral Analysis of Tone FM:

Considering a sinusoidal modulating signal m(t)=A$_m$ cos(2πfmt), we get

$$s(t) = A_c \sum_{n=-\infty}^{\infty} J_n(\beta)\cos[2\pi(f_c + nf_m)t]$$

Where $\beta = \dfrac{k_f A_m}{2\pi f_m}$ is the modulation index or the deviation ratio of FM

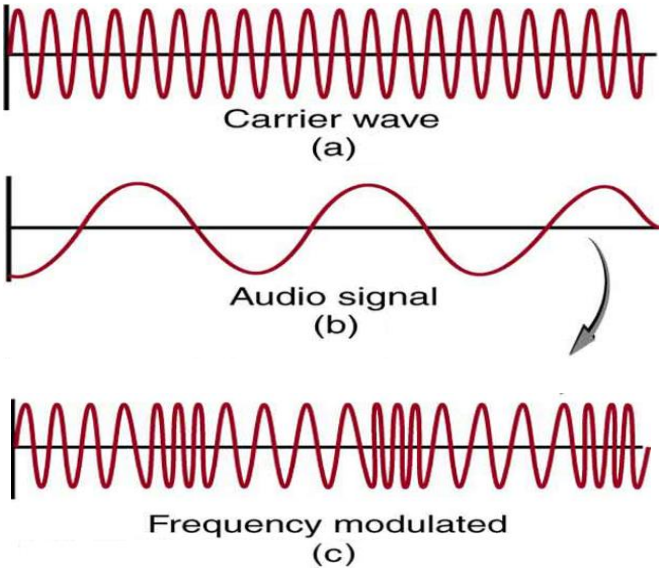
Also, $$S(f) = \frac{A_c}{2} \sum_{n=-\infty}^{\infty} J_n(\beta)[\delta(f - f_c - nf_m) + \delta(f - f_c + nf_m)]$$

Where $J_n(\beta)$ is the Bessel Function.

The Number of significant side frequencies varies with the modulation index (β). The below table shows the number of significant side frequencies (including both lower and upper side frequencies) for different value of modulation index (β)

Modulation index (β)	Number of Significant Side Frequencies (2n$_{max}$)
0.1	2
0.3	4
0.5	4
1	6
2	8
5	16
10	28
20	50
30	70

Time domain representation of frequency modulated signal

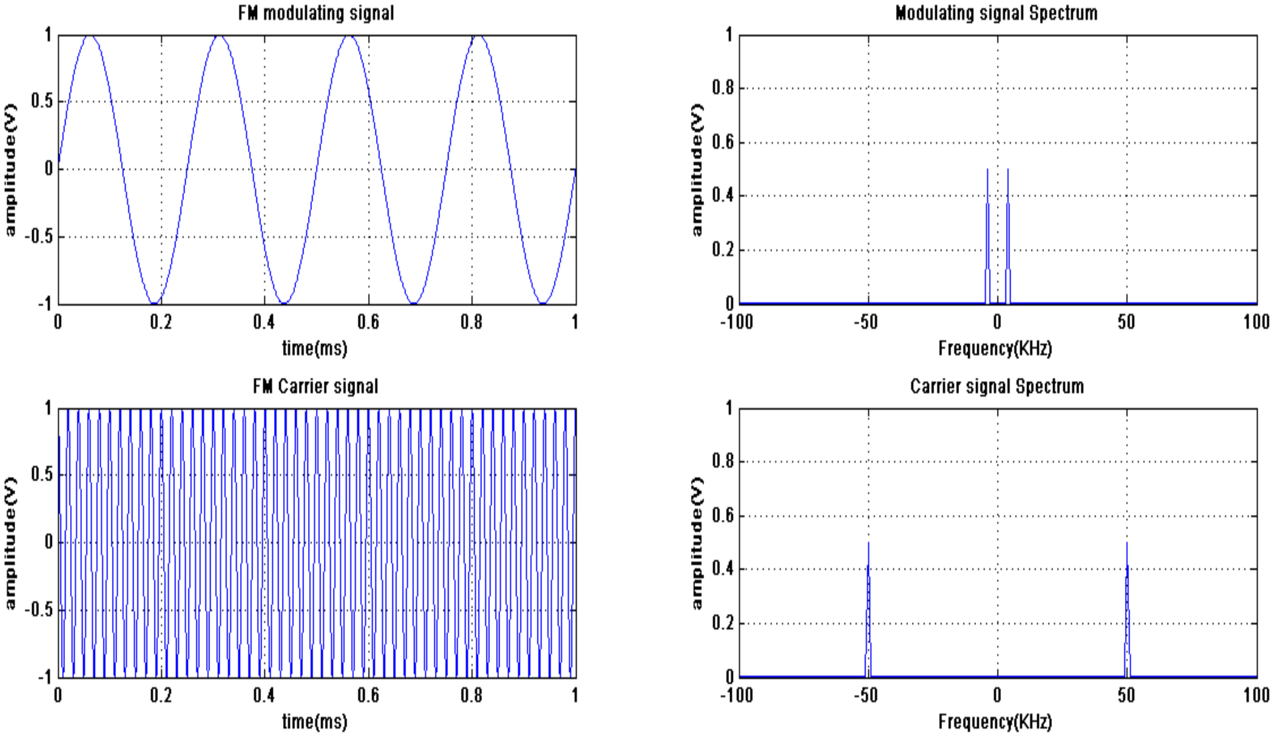


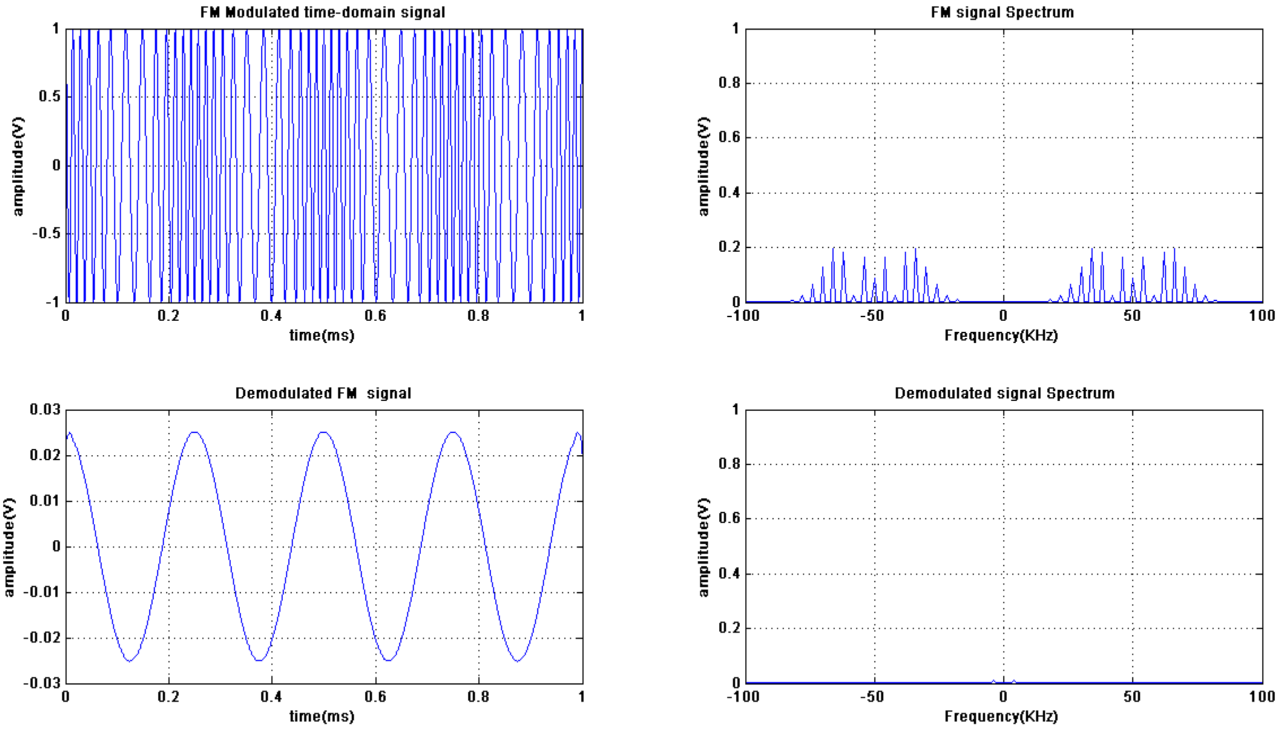
MATLAB Program:

```
% Frequency Modulation %
close all; clear all; clc;

%**************Variables***********************%

fm=4;        % Modulating Signal Frequency (KHz)
fc=50;      % Carrier Frequency (KHz)
 Am=1;        % Modulating Signal Amplitude(V)
 Ac=1;         % Carrier Signal Amplitude(V)
 m=5;         % index of FM; Vary 'm' and see the effect on the number of significant side frequencies

 %******************Signals*******************%

fs=5000; % Sampling frequency (Ksamples/sec)
Wc=2*pi*fc;
 Wm=2*pi*fm;
 t=0:1/(fs):1;%length = 5001

 choice1 = questdlg('Choose input for FM','Input','Sine','cosine' );

switch choice1 %modulating signal
case 'Sine'
        Modulating_Signal=Am*sin(Wm*t);
case 'cosine'
        Modulating_Signal=Am*cos(Wm*t);
end

Carrier_Signal_fm = Ac*cos(Wc*t);%carrier signal

 %**********Plot the signals in time domain and frequency domain************%

figure(1);
subplot(221);plot(t, Modulating_Signal);
title('FM modulating signal');xlabel('time(ms)'), ylabel('amplitude(V)');grid on;

 mod_Sig_freq=fftshift(abs(fft(Modulating_Signal)));
axiss=-fs/2:fs/(length(mod_Sig_freq)-1):fs/2;

subplot(222);plot(axiss,mod_Sig_freq*(1/fs));axis([-100 100 0 1])
title('Modulating signal Spectrum');xlabel('Frequency(KHz)'), ylabel('amplitude(V)');grid on;

subplot(223);plot(t, Carrier_Signal_fm);
title('FM Carrier signal');xlabel('time(ms)'), ylabel('amplitude(V)');grid on;

 Carrier_Signal_freq=fftshift(abs(fft( Carrier_Signal_fm)));
subplot(224);plot(axiss,Carrier_Signal_freq*(1/fs));axis([-100 100 0 1])
title('Carrier signal Spectrum');xlabel('Frequency(KHz)'), ylabel('amplitude(V)');grid on;

%*****************Modulation***************%

 FM_time= Ac*cos((Wc*t)+(m*Modulating_Signal));
 FM_freq=fftshift(abs(fft(FM_time)));
```

```
%*****************Demodulation******************%

FM_DEM_T= demod(FM_time,fc,fs,'fm');
 FM_DEM_F=fftshift(abs(fft(FM_DEM_T)));

 %*****************Plot Modulated and demodulated signals******************%

figure(2);
subplot(221);plot(t,FM_time);
xlabel('time(ms)'),ylabel('amplitude(V)');title('FM Modulated time-domain signal');grid on;

subplot(222);plot(axiss, FM_freq*(1/fs));axis([-100 100 0 1])
title('FM signal Spectrum');xlabel('Frequency(KHz)'), ylabel('amplitude(V)');grid on;

subplot(223);plot(t,FM_DEM_T);
title('Demodulated FM  signal'); xlabel('time(ms)'), ylabel('amplitude(V)');grid on;

subplot(224);plot(axiss,FM_DEM_F*(1/fs));axis([-100 100 0 1])
title(' Demodulated signal Spectrum'); xlabel('Frequency(KHz)'), ylabel('amplitude(V)');grid on;
```

Expected Output Waveform:





Observation:

Sl. No	Criteria	Max Marks	Marks obtained
Data sheet			
A	Problem statement	10	
B	Design & specifications	10	
C	Expected output	10	
Record			
D	Simulation/ Conduction of the experiment	15	
E	Analysis of the result	15	
	Viva	40	
	Total	100	
Scale down to 10 marks			

Experiment 2

Verification of Sampling theorem

Aim:

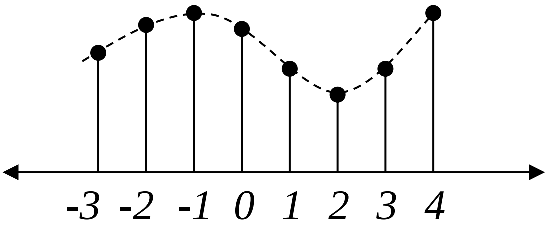
a. Simulate and interpret the sampling theorem on a continuous wave for all three cases

(i) Under -sampling
(ii) Over -sampling
(iii) Critical sampling and comment on the result.

b. To design the sampling circuit and verify the Sampling Theorem

Theory:

Sampling is a conversion of continuous-time signal into a discrete-time signal by taking the "samples" of the continuous-time signal at discrete time instants.



Let $x_c(t)$ is the continuous time signal. The sampled signal x[n] is

$$x[n] = x_c(nT) \qquad -\infty < n < \infty$$

T$_s$ is the Sampling period in Seconds. $fs = \frac{1}{T}$ is the sampling frequency in Hz. Sampling frequency in radian-per-second Ωs=2πfs rad/sec.

Sampling is done by multiplying continuous-time signal $x_c(t)$ with impulse train δ$_T$(t). The sampled signal $x_s(t)$ is given by

$$x_s(t) = x_c(t)\delta_T(t) = \sum_{n=-\infty}^{\infty} x_c(t)\delta(t - nT)$$

By using Multiplication in time domain property of Fourier transform,

$$X_s(j\Omega) = \frac{1}{2\pi} X_c(j\Omega) * S(j\Omega)$$

Fourier transform of an impulse train $\delta_T(t)$ is again an impulse train denoted by S(jΩ) and is given by

$$S(j\Omega) = \frac{2\pi}{T} \sum_{k=-\infty}^{\infty} \delta(\Omega - k\Omega_s)$$

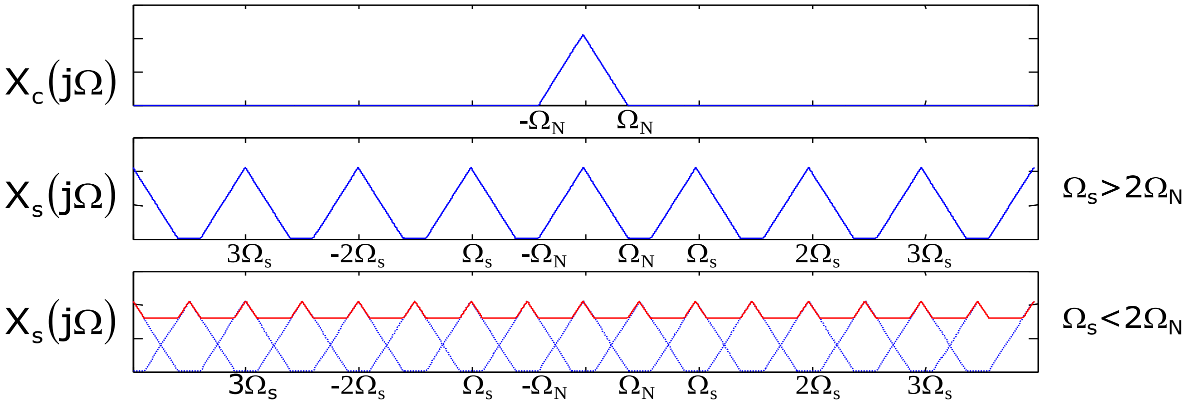
$$X_s(j\Omega) = \frac{1}{T} \sum_{k=-\infty}^{\infty} X_c(j(\Omega - k\Omega_s))$$

This tells us that the impulse train modulator

- Creates images of the Fourier transform of the input signal
- Images are periodic with sampling frequency
- If $\Omega s < \Omega_N$ sampling maybe irreversible due to aliasing of images

Sampling Theorem:
A band limited signal can be reconstructed exactly if it is sampled at a rate at least twice the maximum frequency component in it.

The maximum frequency component of $x_c(t)$ is f_m. To recover the signal $x_c(t)$ exactly from its samples it has to be sampled at a rate $f_s \geq 2f_m$. The minimum required sampling rate $f_s = 2f_m$



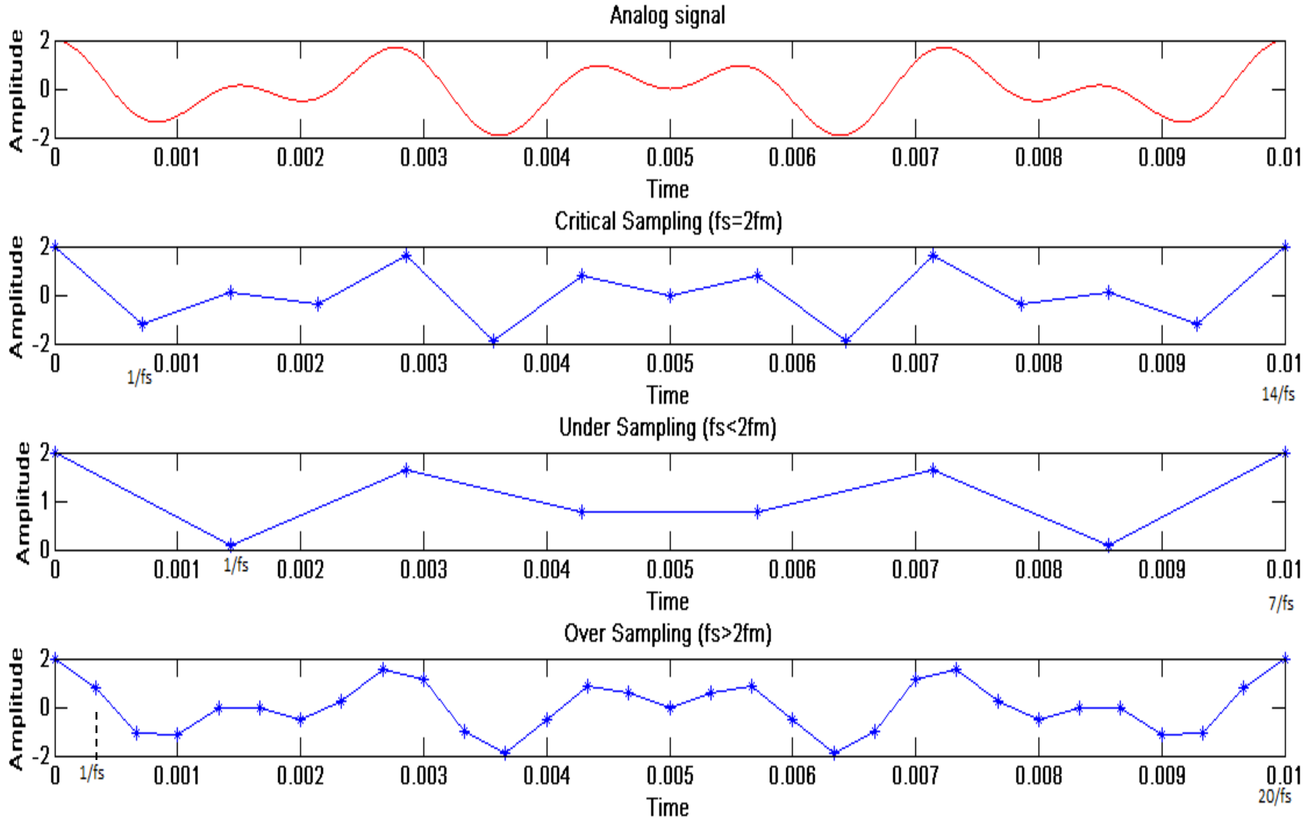
Program:

a) Sampling theorem in time domain

```
%scope:To verify the sampling theorem in time domain
%definiton of variables
%xanalog      :Given analog signal
%xsampled     :Sampled version of xanalog
%tsamp        :sampling interval(spacing between any two  samples)
%fs           :sampling frequency(samples/second)
%t            :time duration in which the analog signalexist(0 to 0.01(tfinal) in this program)
```

```
%fm          :highest frequency(700Hz)contained in the analog signal
%Techniques for reconstruction:1)Zero Order Hold(ZOH) %interpolation results in a
staircase waveform, is implemented %by MATLAB plotting function stairs(), 2)First Order
Hold(FOH) %where the adjacent samples are joined by straight lines is %implemented by
MATLAB plotting function plot()

tfinal=0.01;
t=0:0.00001:tfinal;
xanalog=cos(2*pi*400*t)+cos(2*pi*700*t);
subplot(4,1,1);
plot(t,xanalog,'r-');
xlabel('Time');ylabel('Amplitude');title('Analog signal');

%critical sampling  fs=2fm(fm=700)
fs=1400;
tsamp=0:1/fs:tfinal;
xsampled=cos(2*pi*400*tsamp)+cos(2*pi*700*tsamp);
subplot(4,1,2);
plot(tsamp,xsampled,'b*-');

%stairs(tsamp,xsampled,'b*-');
xlabel('Time');ylabel('Amplitude');title('Critical Sampling (fs=2fm)');

%Under Sampling(fs<2fm)
fs=700;
tsamp=0:1/fs:tfinal;
xsampled=cos(2*pi*400*tsamp)+cos(2*pi*700*tsamp);
subplot(4,1,3);
plot(tsamp,xsampled,'b*-');

%stairs(tsamp,xsampled,'b*-');
xlabel('Time');ylabel('Amplitude');title('Under Sampling (fs<2fm)');

%over Sampling(fs>2fm)
fs=2000;
tsamp=0:1/fs:tfinal;
xsampled=cos(2*pi*400*tsamp)+cos(2*pi*700*tsamp);
subplot(4,1,4);
plot(tsamp,xsampled,'b*-');

%stairs(tsamp,xsampled,'b*-');
xlabel('Time');ylabel('Amplitude');title('Over Sampling (fs>2fm)');
%To implement ZOH, uncomment stairs(.......) function and comment plot(.....) function
```

Expected Output Waveform:



Observation:

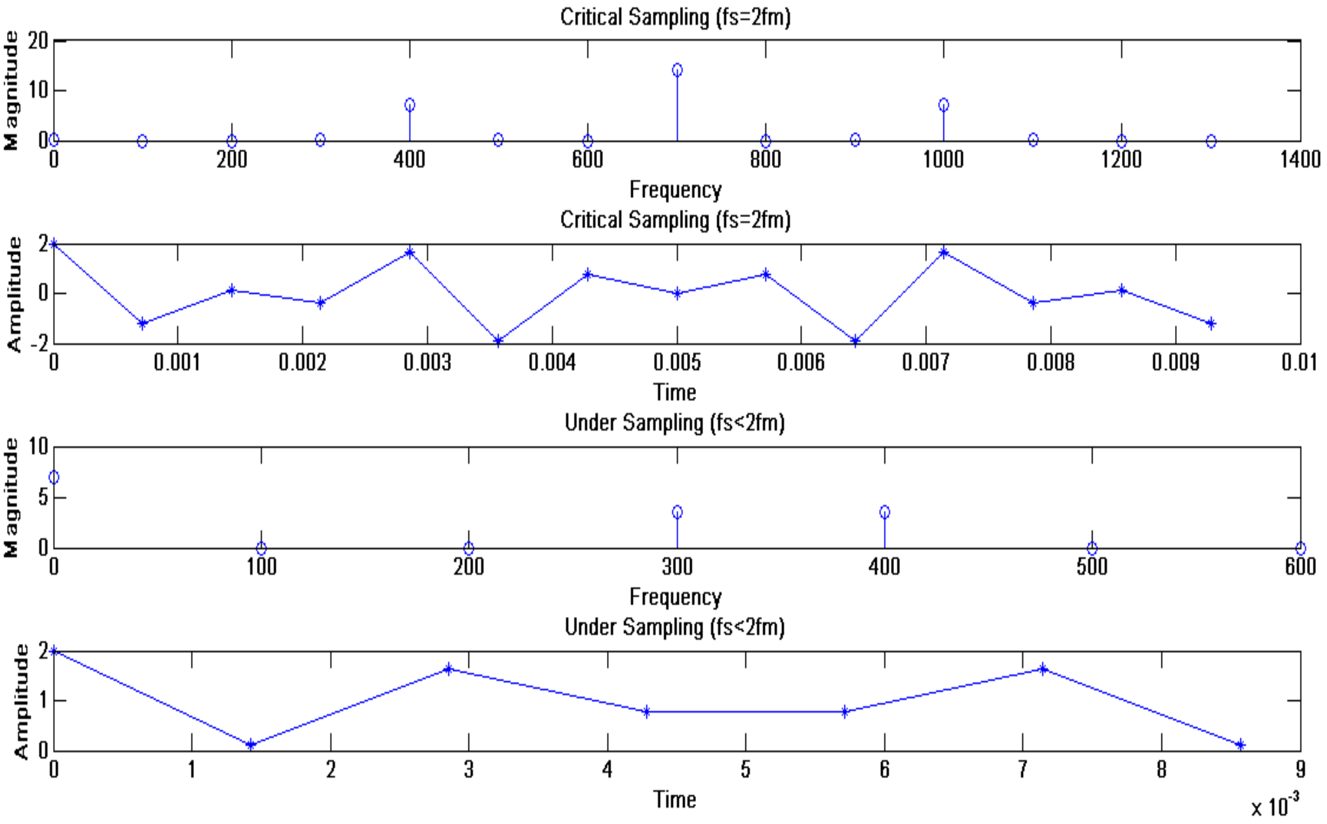
Sampling theorem in frequency domain

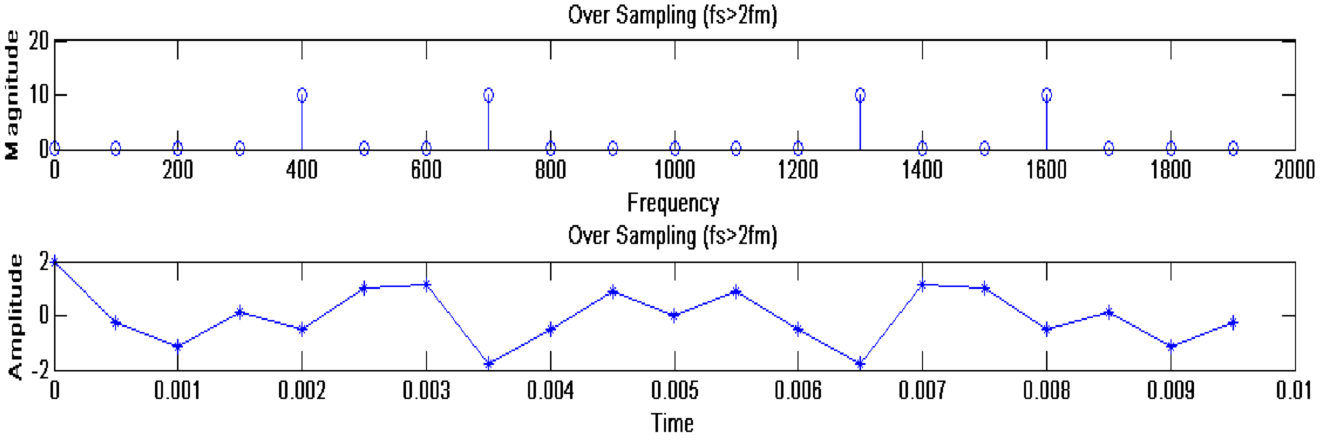
Program:

```
%  To verify the sampling theorem in frequency domain
%definiton of variables
%xanalog          :Given analog signal%xsampled         :Sampled version of xanalog
%xsampled_DFT       :DFT of sampled signal    %tsamp              :sampling interval(spacing
between any two samples)%xsampled_length   :no. of samples of analog signal collected
%xreconstructed    :sampled sequence in time domain
%fs:samplimg frequency(samples/second)
%t               :time duration in which the analog signal exist (0 to 0.01(tfinal) in this program)
%fm              :The highest frequency(700 Hz) contained in the analog signalsampling in time
domain results in periodic components infrequency domain centered at nfs, where n=.... -3,-2,-
1,1,2 %3,.....sampling in frequency domain can also be used to %analyze aliasing effect

tfinal=0.01;
t=0:0.00001:tfinal;
xanalog=cos(2*pi*400*t)+cos(2*pi*700*t);
%critical sampling  fs=2fm(fm=700)
fs=1400;%Note that samples will repeat after 13/fs,hence its %sufficient if we consider first
14 samples(0 to 13) of analog %signal (observe the figures given previously)

tsamp=0:1/fs:13/fs;
xsampled=cos(2*pi*400*tsamp)+cos(2*pi*700*tsamp);%calculate 14 point DFT of sampled
signal. abs(...) is %required because plot function displays only real values

xsampled_DFT=abs(fft(xsampled));
xsampled_length=0:length(xsampled_DFT)-1;

figure(1);subplot(4,1,1);stem(100*xsampled_length,xsampled_DFT);
xlabel('Frequency');ylabel('Magnitude');title('Critical Sampling (fs=2fm)');

xreconstructed=ifft(fft(xsampled)); %To reconstruct the signal, ifft(...) can be  used.
subplot(4,1,2);plot(tsamp,xreconstructed,'b*-');
xlabel('Time');ylabel('Amplitude');title('Critical Sampling (fs=2fm)');

 %Under Sampling(fs<2fm)
fs=700;%Note that samples will repeat after 6/fs,hence its sufficientif we consider first 7
samples(0 to 6) of analog signal
tsamp=0:1/fs:6/fs;
xsampled=cos(2*pi*400*tsamp)+cos(2*pi*700*tsamp);
xsampled_DFT=abs(fft(xsampled)); %calculate 7 point DFT of sampled signal.
xsampled_length=0:length(xsampled_DFT)-1;
```

```
subplot(4,1,3);stem(100*xsampled_length,xsampled_DFT)
xlabel('Frequency');ylabel('Magnitude');title('Under Sampling (fs<2fm)');

xreconstructed=ifft(fft(xsampled)); %To reconstruct the signal, ifft(...) can be  used.
subplot(4,1,4);plot(tsamp,xreconstructed,'b*-');
xlabel('Time');ylabel('Amplitude');title('Under Sampling (fs<2fm)');

%over Sampling(fs>2fm)
fs=2000;%Note that samples will repeat after 19/fs,hence its %sufficient if we consider first
20 %samples(0 to 19) of analog %signal
tsamp=0:1/fs:19/fs;
xsampled=cos(2*pi*400*tsamp)+cos(2*pi*700*tsamp);%calculate 20 point DFT of sampled
%signal.
xsampled_DFT=abs(fft(xsampled));
xsampled_length=0:length(xsampled_DFT)-1;

figure(2);subplot(4,1,1);stem(100*xsampled_length,xsampled_DFT)
xlabel('Frequency');ylabel('Magnitude');title('Over Sampling (fs>2fm)');

xreconstructed=ifft(fft(xsampled));
subplot(4,1,2);plot(tsamp,xreconstructed,'b*-');
xlabel('Time');ylabel('Amplitude');title('Over Sampling (fs>2fm)');
```

Expected Output Waveform:





Observation:

b)Verification of sampling theorem

Aim:

To design and verify Sampling theorem using flat top samples

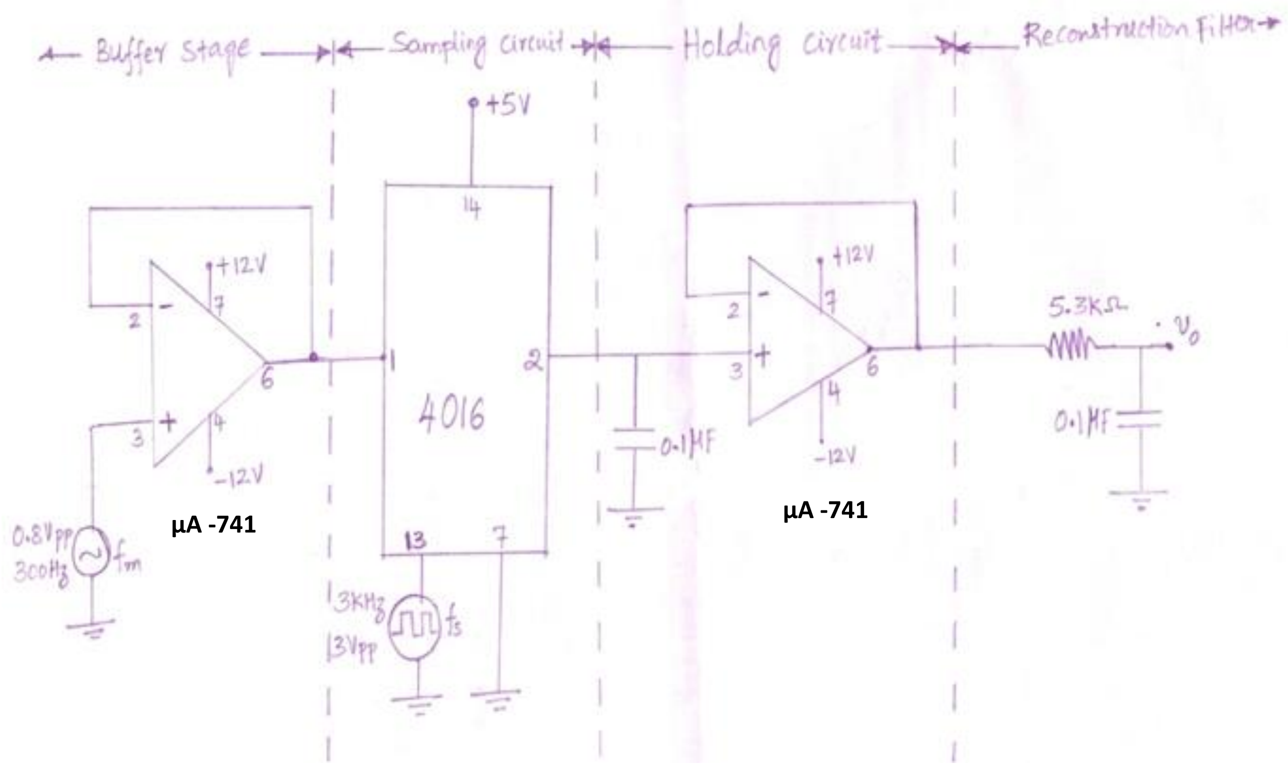
Apparatus Required:

Function Generator, CRO, Connecting wires, BNC Probes, dc power supply, CD-4016 (CMOS bilateral Switch), µA-741 (Op-amp), Resistor, Capacitor

Theory:

The analog signal can be converted into a discrete time signal by a process called sampling. The sampling theorem for a band limited signal of a finite energy can be statedas“A band limited signal of finite energy which has no frequency component higherthan W Hz is completely described by specifying the values of the signal at instants oftime separated by 1/2W seconds”.It can be recovered from the knowledge of the samples taken at the rate of 2W persecond.

Circuit Diagram:



Design of Reconstruction filter:

$f_m = \frac{1}{2\pi RC}$ As f_m=300Hz, choose C=0.1µF. Therefore, R=5.3KΩ

Procedure:

1. Connections are made as per the Circuit diagram.
2. Apply the input signal with a frequency of 300Hz (0.8V P-P) using a function generator through a buffer amplifier to pin No. 1 of IC 4016.
3. Sampling clock frequency of 3KHz of higher amplitude (say, 3V P-P) is applied as a control signal to pin No. 13 of IC 4016.
4. Observe the sample and hold output at pin No. 6 of second opamp.
5. Reconstructed signal can be obtained across 0.1µF capacitor.
6. Vary the sampling frequency and study the change in reconstructed signal.

Observation:

Sl. No	Criteria	Max Marks	Marks obtained
Data sheet			
A	Problem statement	10	
B	Design & specifications	10	
C	Expected output	10	
Record			
D	Simulation/ Conduction of the experiment	15	
E	Analysis of the result	15	
	Viva	40	
	Total	100	
Scale down to 10 marks			

Experiment3

Implementation of Convolution and DFT on DSP kit

Aim:

To implement the Linear convolution, Circular convolution and DFT on the DSP kit.

Theory:

Linear Convolution:

For a system with the impulse response $h[n]$, the output for any arbitrary input $x[n]$ is given by convolution defined as,

$$y[n] = x[n] * h[n]$$

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k]$$

Circular convolution:

It is periodic convolution. $x_1[n]$ and $x_2[n]$ are the two finite duration sequences of length N, the circular convolution between though sequence is given by

$$x_3[n] = x_1[n] \otimes x_2[n] = x_2[n] \otimes x_1[n]$$

$$x3[n] = \sum_{m=0}^{N-1} x_2[m] x_1[((n-m))_N]$$

There is a practical difficulty which often arises, when circular convolution is performed digitally by fast convolution. For circular convolution it is assumed that x(n) and h(n) contain roughly the same number of samples, otherwise they are zero-filled up to the same length of transform. However, in practice the input sequence will be relatively longer than the impulse response. In such cases, it is uneconomical in terms of computing time (and storage) to use the same length of transform for both x(n) and h(n). In addition, in real-time applications the use of very long transform lengths for x(n) may give rise to an unacceptable processing delay.

This problemis addressed by segmenting the input signal into sections of manageable length and then performing fast convolution on each section and finally combining the outputs. Two well-known techniques are commonly used: The overlap-add method and the overlap-save method.

Difference between linear and circular convolution:

Circular convolution along with zero padding is known as linear convolution. Circular convolution is used for periodic and finite signals while linear convolution is used for periodic and infinite signals.

Linear and circular convolution are fundamentally different operations. However, there are conditions under which linear and circular convolution are equivalent. Establishing this equivalence has important implications. For two vectors, x and y, the circular convolution is equal to the inverse discrete Fourier transform (DFT) of the product of the vectors' DFTs. Knowing the conditions under which linear and circular convolution are equivalent allows you to use the DFT to efficiently compute linear convolutions.

The linear convolution of an N-point vector, x, and a L-point vector, y, has length N+L-1.For the circular convolution of x and y to be equivalent, you must pad the vectors with zeros to length at least N+L-1 before you take the DFT. After you invert the product of the DFTs, retain only the first N+L-1 elements.

Discrete Time Fourier Transform:

Discrete Fourier transform is defined for sequences with finite length.

For a sequence x[n] with length N (x[n] for n=0,1, 2, ..., N-1), the discrete-time Fourier transform is

$$X(\omega) = \sum_{n=-\infty}^{\infty} x[n]e^{-j\omega n} = \sum_{n=0}^{N-1} x[n]e^{-j\omega n}$$

$X(\omega)$ periodic with period 2π. The usually considered frequency interval is (-π , π). There are infinitely many points in the interval. If x[n] has N points, we compute N equally spaced ω in the interval (-π , π). That is, we sample $X(\omega)$ using the frequencies

$$\omega = \omega_k = \frac{2\pi k}{N}, \qquad 0 \le k \le N-1$$

These N points are

$$X(k) \quad = X(\frac{k2\pi}{N}) \quad = \sum_{n=0}^{N-1} x[n]e^{-j\frac{k2\pi}{N}n} \qquad\qquad \omega = \frac{k2\pi}{N}, K = 0,1,...N-1$$

The above equation is known as N-point DFT Analysis equation.

Inverse DFT:

The DFT values (X(K), 0 ≤ k ≤ N – 1), uniquely define the sequence x[n] through the inverse DFT formula (IDFT)

$$\text{x(n) = IDFT \{X(k)\}} = \frac{1}{N}\sum_{k=0}^{N-1} \text{X(k)w}_\text{n}{}^{-kn}, \quad 0 \le \text{N} \le \text{n} - 1$$

The above equation is known as Synthesis equation

<u>Procedure:</u>

Program For Linear Convolution:

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

        int y[20];
main()
        {
int m=6; /*Length of i/p samples sequence*/
        int n=6; /*Length of impulse response Co-efficient */
int i,j;
        int x[15]={1,2,3,4,5,6,0,0,0,0,0,0};/*Input Signal Samples*/
        int h[15]={1,2,3,4,5,6,0,0,0,0,0,0};/*Impulse Response Co-effs*/

                for(i=0;i<m+n-1;i++)
                {
                y[i]=0;
                for(j=0;j<=i;j++)
                        y[i]+=x[j]*h[i-j];
                }

                printf("Linear Convolution\n");
                for(i=0;i<m+n-1;i++)
                printf("%d\n",y[i]);

while(1);//Infinite Loop will run the program infinitely to view graph

        }
```

Expected Output:

```
Linear Convolution
1  4  10   20  35  56  70  76  73   60  36
```

Observation:

Program for Circular Convolution:

```
#include<stdio.h>
 #include<math.h>
int y[30];
void main()
 {
   int m,n,x[30],h[30],i,j,temp_conv,temp,k;
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;
            }
            else{
            for(i=m;i<n;i++)
            x[i]=0;
            m=n;
            }
   } //end of if(m-n!=0)

for(j=0;j<m;j++)
            {
            y[j]=0;
            temp_conv=0;
            for(k=0;k<m;k++)
            {
            if((j-k)<0)
            temp=j-k+m;
            else
            temp=j-k;
   temp_conv=temp_conv+x[k]*h[temp];

   }
y[j]=temp_conv;

 }
/*displaying the result*/
printf("The circular convolution is\n");
for(i=0;i<m;i++)
printf("%d \t",y[i]);

while(1);//Infinite Loop will run the program infinitely to view graph

}
```

Expected Output:

```
Enter the length of the first sequence
3
Enter the length of the second sequence
5
Enter the first sequence
1 2 3
Enter the second sequence
4 5 6 7 8
The circular convolution is
41      37      28      34      40
```

Observation:

Program for computing N point DFT of a given sequence:

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

main()
{
int i,N,k,n;
float x[10];
float real,imaginary,w,magnitude[10],phase[10];
float real_array[10],imaginary_array[10];
printf("Enter the length of DFT:");
scanf("%d",&N);
printf("Enter the value of x[n]:");

for(i=0;i<N;i++)
scanf("%f",&x[i]);
for(k=0;k<N;k++)
{
real_array[k]=0;
imaginary_array[k]=0;
real=0;
imaginary=0;
for(n=0;n<N;n++)
{
w=(2*3.1416*k*n)/N;
real=real+(x[n]*cos(w));
imaginary=imaginary+(x[n]*sin(w));
}
real_array[k]=real;
imaginary_array[k]=-imaginary;
}
printf("The output sequence:\n");
for(k=0;k<N;k++)
{
printf("%f + i%f ",real_array[k],imaginary_array[k]);
magnitude[k]=sqrt((real_array[k]*real_array[k])+(imaginary_array[k]*imaginary_array[k]));
//atan2(y,x) computes inverse tan(y/x), the value in the range %[-p, p]
phase[k]=atan2(imaginary_array[k],real_array[k]);
printf("\t\tMagnitude:%f",magnitude[k]);
printf("\t\tPhase(rad):%f\n",phase[k]);
}
while(1);//Infinite Loop will run the program infinitely to view graph
}
```

Expected Output:

```
Enter the length of DFT:4
Enter the value of x[n]:1 2 3 4
The output sequence:
10.000000 + i0.000000          Magnitude:10.000000      Phase(rad):0.000000
-1.999963 + i2.000022          Magnitude:2.828417       Phase(rad):2.356180
-2.000000 + i0.000059          Magnitude:2.000000       Phase(rad):3.141563
-2.000112 + i-1.999934         Magnitude:2.828460       Phase(rad):-2.356239
```

Observation:

Sl. No	Criteria	Max Marks	Marks obtained
Data sheet			
A	Problem statement	10	
B	Design & specifications	10	
C	Expected output	10	
Record			
D	Simulation/ Conduction of the experiment	15	
E	Analysis of the result	15	
	Viva	40	
	Total	100	
Scale down to 10 marks			

Experiment 4

Realization of FIR filter to meet given specifications using DSP STARTER KIT

Aim:

Implementations of FIR filter for the given specification on DSP board.

Theory:

In signal processing, a finite impulse response (FIR) filter is a filter whose impulse response (or response to any finite length input) is of finite duration, because it settles to zero in finite time. This is in contrast to infinite impulse response (IIR) filters, which may have internal feedback and may continue to respond indefinitely (usually decaying).

The impulse response of an Nth-order discrete-time FIR filter (i.e., with a Kronecker delta impulse input) lasts for N + 1 samples, and then settles to zero.

FIR filters can be discrete-time or continuous-time, and digital or analog.

For a discrete-time FIR filter, the output is a weighted sum of the current and a finite number of previous values of the input. The operation is described by the following equation, which defines the output sequence y[n] in terms of its input sequence x[n]:

$Y[n] = b_o x[n] + b_1 x[n-1] + b_2 x[n-2]$ **……….**

To design a filter means to select the coefficients such that the system has specific characteristics. The required characteristics are stated in filter specifications. Most of the time filter specifications refer to the frequency response of the filter. There are different methods to find the coefficients from frequency specifications:

1. Window design method
2. Frequency Sampling method
3. Weighted least squares design

Window design method:

In the Window Design Method, one designs an ideal IIR filter, then applies a window function to in the time domain, multiplying the infinite impulse by the window function. This results in the frequency response of the IIR being convolved with the frequency response of the window function. If the ideal response is sufficiently simple, such as rectangular, the result of the convolution can be relatively easy to determine. In fact one usually specifies the desired result first and works backward to determine the appropriate window function parameter(s). Kaiser windows are particularly well-suited for this method because of their closed form specifications.

IIR Vs FIR Filters:

(i) IIR filters are difficult to control and have no particular phase, whereas FIR filters make a linear phase always possible.

(ii) IIR can be unstable, whereas FIR is always stable.

(iii) IIR is derived from analog, whereas FIR has no analog history. IIR filters make polyphase implementation possible, whereas FIR can always be made casual..

(iv) FIR filters are dependent upon linear-phase characteristics, whereas IIR filters are used for applications which are not linear.

(v) FIR's delay characteristics is much better, but they require more memory. On the other hand, IIR filters are dependent on both i/p and o/p, but FIR is dependent upon i/p only

(vi) IIR filters consist of zeros and poles, and require less memory than FIR filters, whereas FIR only consists of zeros.

(vii) IIR filters can become difficult to implement, and also delay and distort adjustments can alter the poles & zeroes, which make the filters unstable, whereas FIR filters remain stable. FIR filters are used for tapping of a higher-order, and IIR filters are better for tapping of lower-orders, since IIR filters may become unstable with tapping higher-orders.

FIR stands for Finite IR filters, whereas IIR stands for Infinite IR filters. IIR and FIR filters are utilized for filtration in digital systems. FIR filters are more widely in use, because they differ in response. FIR filters have only numerators when compared to IIR filters, which have both numerators and denominators.

Where the system response is infinite, we use IIR filters, and where the system response is zero, we use FIR filters. FIR filters are also preferred over IIR filters because they have a linear phase response and are non recursive, whereas IIR filters are recursive, and feedback is also involved. FIR cannot simulate analog filter responses, but IIR is designed to do that accurately. IIR's impulse response when compared to FIR is infinite.

Program:

```
#include "dsk6713_aic23.h" //support file for codec, DSK

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

#include <math.h> //for performing modulation operation
```

```
static short in_buffer[100];

Uint32 sample_data;

short k=0;

//float filter_Coeff[] = { -0.0017,-0.0020,-0.0024,-0.0027,-0.0021,

                    //0.0000,0.0044 ,0.0117,0.0221,0.0351,0.0500,0.0655,0.0799,0.0917,

                    //0.0994,0.1021,0.0994,0.0917,0.0799,0.0655,0.0500, 0.0351,0.0221,

                    //0.0117,0.0044,0.0000,-0.0021,-0.0027, -0.0024,-0.0020, -0.0017}; // for fc=400Hz

float filter_Coeff[] = { -0.0017, 0.0000,0.0029,-0.0000,-0.0067, 0.0000, 0.0141,-0.0000,-0.0268, 0.0000,
0.0491,-0.0000,-0.0969,0.0000,0.3156, 0.5008,0.3156, 0.0000,-0.0969,-0.0000, 0.0491,0.0000,-0.0268,-
0.0000,0.0141, 0.0000,-0.0067,-0.0000, 0.0029, 0.0000,-0.0017};

// for fc=2KHz

short l_input, r_input,l_output, r_output;

void comm_intr();

void output_left_sample(short);

short input_left_sample();

signed int FIR_FILTER(float *h, signed int);

interrupt void c_int11() //interrupt service routine

{

l_input = input_left_sample(); //inputs data

l_output=(Int16)FIR_FILTER(filter_Coeff ,l_input);

output_left_sample(l_output);

return;

} // end of interrupt routine

signed int FIR_FILTER(float * h, signed int x)

{

int i=0;

signed long output=0;

in_buffer[0] = x; /* new input at buffer[0] */

for(i=31;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);

}



void main()

{

comm_intr(); //init DSK, codec, McBSP

while(1);

}
```

Note:

Use Matlab to generate the filter coefficients.

The syntax is fir1(N,2*fc/Fs,’high’, hamming(N+1) here fc is cutoff frequency, Fs is sampling rate. fir1 is low pass filter by default.

E.g: 1.To design LPF with cutoff frequency 400Hz and sampling frequency 8000Hz, with order 30, Execute the following command in matlab.
fir1(30,2*400/8000,hamming(31)): This will generate 30 coefficients of LPF

Observation:

Sl. No	Criteria	Max Marks	Marks obtained
Data sheet			
A	Problem statement	10	
B	Design & specifications	10	
C	Expected output	10	
Record			
D	Simulation/ Conduction of the experiment	15	
E	Analysis of the result	15	
	Viva	40	
	Total	100	
Scale down to 10 marks			

Experiment 5

Realization of IIR filter to meet given specifications using DSP STARTER KIT

Aim:

Implementations of IIR filter for the given specification on DSP board.

Sl. No	**Criteria**	**Max Marks**	**Marks obtained**
Data sheet			
A	Problem statement	10	
B	Design & specifications	10	
C	Expected output	10	
Record			
D	Simulation/ Conduction of the experiment	15	
E	Analysis of the result	15	
	Viva	40	
	Total	100	
Scale down to 10 marks			

Experiment 6

Channel coding

Aim: a)To Encode and Decode the linear block code words for a given generator matrix or parity matrix

b) To Encode the using Huffman coding for a given probability of occurrence.

a) *Linear Block code:*

Theory: Communication through noisy channels is subject to errors. In order to decrease the effect of errors and achieve reliable communication, it is necessary to transmit sequences that are as different as possible so that the channel noise will not change one sequence into another. This means some redundancy has to be introduced to increase the reliability of communication. The introduction of redundancy results in transmission of extra bits and a reduction of transmission rate. Channel coding schemes can be generally divided into two classes, block codes and convolution codes. In block coding, binary source output sequence of length k are mapped into binary channel input sequence of length n; therefore the rate of the resulting code is k/n bits per transmission. Such a code is called (n,k) block code and consists of 2^k code words of length *n*.

Linear block codes are the most important and widely used class of block codes. A block code is linear if any linear combination of two code words is a codeword. In the binary case, this means that the sum of any two codeword is a code word. In linear block code the code words form a $k$dimensional subspace of an n-dimensional space. Linear block codes are generated by their generator matrix G which is a $k \times n$. Binary matrix such that each code word *c*can be written as

$$c = uG$$

Where u is the binary data sequence of length k (encoder input).

An important parameter in a linear block code, which determines its error correcting capabilities, is the minimum distance of the code, which is defined as the minimum Hamming distance between any two distinct code words.

Decoding of Linear Block Code

Each k×n generating matrix G= [Ik | P] is associated with a (n−k) ×n parity check matrix given by

$$H = [P^T \mid In-k].$$

Basic property of codeword: c is a codeword in the (n, k) block code generated by G, if and only if $cH^T = 0$.

Received row vector r can be written as r=c+e

All the elements are binary valued, e.g. if the transmitted $c_i = 1$ and is received in error:

$r_i = 0$, then $e_i = 1$

syndromeis defined by $S=rH^T= (c+e)H^T=cH^T+eH^T=eH^T$

S is related to the error vector e, and can be used to detect and correct errors

Error Detection and Correction Capabilities

•Weight of a codeword c is the number of nonzero elements in c

•Hamming distance between two codewords c_1 and c_2 is the number of elements in which they differ

•Minimum distance of a codebook,dmin, is the smallest Hamming distance between any pair of codewords in the codebook

•The minimum distance dmin of a linear block code is equal to the minimum weight of any nonzero codeword in the code

•Code with dmin can detect up to dmin−1errors and correct up to (dmin−1)/2 errors in each codeword

Program:

```
% Variables and matrices used:
% G : generated matrix ( K * N)
% P : Parity matrix (K * N-K)
% Msg : Message vector (K)
% R : Received vector (N)
% H_t : Transpose of parity check matrix (N * N-K)

N=input('Enter the length of the code vector n :');
K=input('Enter the length of the message vector k :');
choice = input('What would you like to enter ? \n\n 1.Generateed matrix or \n 2.parity matrix
\n\n Enter your choice :');

if(choice == 1)
        G = input('\n \n Enter the Generated matrix :');
display('Entered Generated matrix : ');
display(G);

        P = G([1:K],[K+1:N]);
display('The parity matrix :');
display(P);

else

P = input('\n \n Enter the Parity matrix :');
display('Entered Parity matrix : ');
display(P);
end
```

% Generate all possible code vectors

```
%display('Message vector :');

for i=0:2^(K)-1
    n=i;
for j=1:K
Msg2(j) = mod(n,2);
        n = fix(n/2);
end
for j=1:length(Msg2)
Msg(length(Msg2)-j+1) = Msg2(j);
end

   % display(Msg);

check = Msg * P;
    [r,c]=size(check);
for x=1:r
for y=1:c
if(mod(check(x,y),2)==0)
check(x,y)=0;
else
check(x,y)=1;
end
end
end
code = [Msg,check];
display(code);
end

R = input('Enter the received matrix :');
```

% Find H_t

```
H = [P',eye(N-K)];
H_t = H';
display(H_t);
% Calculate syndrome bits
Synd = R * H_t;
 [r,c]=size(Synd);
for x=1:r
for y=1:c
if(mod(Synd(x,y),2)==0)
                Synd(x,y)=0;
else
                Synd(x,y)=1;
end
end
end
```

```
display('Syndrome bits :');
display(Synd);

% Detect and correct the error

 [r,c]=size(H_t);
for x=1:r
flag = 1;
for y=1:c
if(H_t(x,y)~=Synd(y))
flag =0;
break;
end
end
if(flag==1)
disp 'Error in bit :';
display(x);

for i=1:N
if(i==x)
E(i)=1;
if(R(i)==1)
R(i)=0;
else
R(i)=1;
end
else
E(i)=0;
end
end

disp 'Error vector : ';
display(E);

disp 'Correctd Vector :';
display(R);
end
end
```

Expected Output:

```
Enter the length of the code vector n :6

Enter the length of the message vector k :3

What would you like to enter ?
1.Generateed matrix or
2.parity matrix

 Enter your choice :2
Enter the Parity matrix :[1 1 1;1 1 1;1 0 1]


Entered Parity matrix :

P = 1     1     1
1     1     1
1     0     1

Code words:

0     0      0     0     0     0
0     0     1     1     0     1
0     1      0     1     1     1
0     1     1      0     1     0
1     0     0      1     1     1
1     0      1     0     1     0
1     1      0     0     0     0
1     1      1     1     0     1



Enter the received matrix :[1 0 0 1 0 1]
H_t =     1     1     1
          1     1     1
          1     0     1
          1     0     0
          0     1     0
          0     0     1

Synd =     0     1     0
Error in bit :x =     5
Error vector : E =     0     0     0     0     1     0
Corrected Vector :R =1     0     0     1     1     1
```

Observation:

b) HUFFMAN CODING:

Theory:

This coding technique is developed by David Huffman. Huffman codes are

- Prefix codes
- Optimum for a given model (set of probabilities)

Based on two observations regarding optimum prefix codes:

1. In an optimum code, symbols that occur more frequently (have a higher probability of occurrence) will have short codewords than symbols that occur less frequently
2. In an optimum code, the two symbols that occur least frequently will have the same length

Redundancy is the difference between the entropy and average length and efficiency is given by

$$\eta = H(s)/Laverage*\log_2 r$$

Program:

```
clear all;close all;clc;
prob=0;
% disp('The set of probabilities');
% p=input('enter the model(set of probabilities)');
prob=[0.2 0.15 0.13 0.12 0.1 0.09 0.08 0.07 0.06];

[huffcode,n]=huffmancode(prob);
entropy=sum(-log(prob)*prob'/log(2));
disp(['sym','-->','   ','codeword','   ','prob']);
for i=1:n
    codeword_length(i)=n-length(find(abs(huffcode(i,:))==32));
    disp(['x',num2str(i),'---->',huffcode(i,:),'    ',num2str(prob(i))]);
end

codeword_length;
avg_length=codeword_length*prob';
disp(['entropy= ',num2str(entropy)]);
disp(['aver cw len=  ',num2str(avg_length)]);
redundancy=abs(entropy-avg_length);
disp('redundancy');
disp(redundancy);

function[huffcode,n]=huffmancode(prob);
if min(prob)<0
error('negative prob');
else if (sum(prob))>1,
error('sum is not equal to 1')
end
[probsort,probord]=sort(prob)
n=length(prob);
q=prob;
for i=1:n-1
   [q,l]=sort(q);
   m(i,:)=[l(1:n-i+1),zeros(1,i-1)];
   q=[q(1)+q(2),q(3:end),1];
end
cword=blanks(n^2)
cword(n)='0';
cword(2*n)='1';
for i1=1:n-2
ctemp=cword;
   idx0=find(m(n-i1,:)==1)*n;
cword(1:n)=[ctemp(idx0-n+2:idx0) '0'];
cword(n+1:2*n)=[cword(1:n-1) '1'];
for i2=2:i1+1
      idx2=find(m(n-i1,:)==i2);
cword(i2*n+1:(i2+1)*n)=ctemp(n*(idx2-1)+1:n*idx2);
end
end
for i=1:n
      idx1=find(m(1,:)==i);
huffcode(i,1:n)=cword(n*(idx1-1)+1:idx1*n);
end
end
```

Expected Output:

```
Probability set=[0.08 0.3 0.03 0.1 0.05 0.02 0.22 0.2]
probsort =    0.0600    0.0700    0.0800    0.0900    0.1000    0.1200    0.1300    0.1500
0.2000
probord =     9     8     7     6     5     4     3     2     1
cword =
sym-->   codeword   prob
x1---->      00    0.2
x2---->     110    0.15
x3---->     101    0.13
x4---->     011    0.12
x5---->     010    0.1
x6---->    1111    0.09
x7---->    1110    0.08
x8---->    1001    0.07
x9---->    1000    0.06
entropy= 3.0731 bits/m-symbols
aver cw len=  3.1
redundancy    0.0269
```

Observation:

Sl. No	Criteria	Max Marks	Marks obtained
Data sheet			
A	Problem statement	10	
B	Design & specifications	10	
C	Expected output	10	
Record			
D	Simulation/ Conduction of the experiment	15	
E	Analysis of the result	15	
	Viva	40	
	Total	100	
Scale down to 10 marks			

Experiment 7

Time-Division multiplexing

Aim:

a. Simulate and interpret theTime-Division multiplexing
b. To design and test the Time-Division multiplexing

Theory:

Time-division multiplexing (TDM) is a type of digital or (rarely) analog multiplexing in which two or more signals or bit streams are transferred apparently simultaneously as sub-channels in one communication channel, but are physically taking turns on the channel.

The time domain is divided into several recurrent timeslots of fixed length, one for each sub-channel. A sample byte or data block of sub-channel 1 is transmitted during timeslot 1, sub-channel 2 during timeslot 2, etc. One TDM frame consists of one timeslot per sub-channel. After the last sub-channel the cycle starts all over again with a new frame, starting with the second sample, byte or data block from sub-channel 1.

Application examples:

- The plesiochronous digital hierarchy (PDH) system, also known as the PCM system, for digital transmission of several telephone calls over the same four-wire copper cable (T-carrier or E-carrier) or fiber cable in the circuit switched digital telephone network

- The SDH and synchronous optical networking (SONET) network transmission standards that have surpassed PDH.

- The RIFF (WAV) audio standard interleaves left and right stereo signals on a per-sample basis

- The left-right channel splitting in use for stereoscopic liquid crystal shutter glasses

TDM can be further extended into the time division multiple access (TDMA) scheme, where severalstations connected to the same physical medium, for example sharing the same frequency channel,can communicate. Application examples include:

- The GSM telephone system

Program:

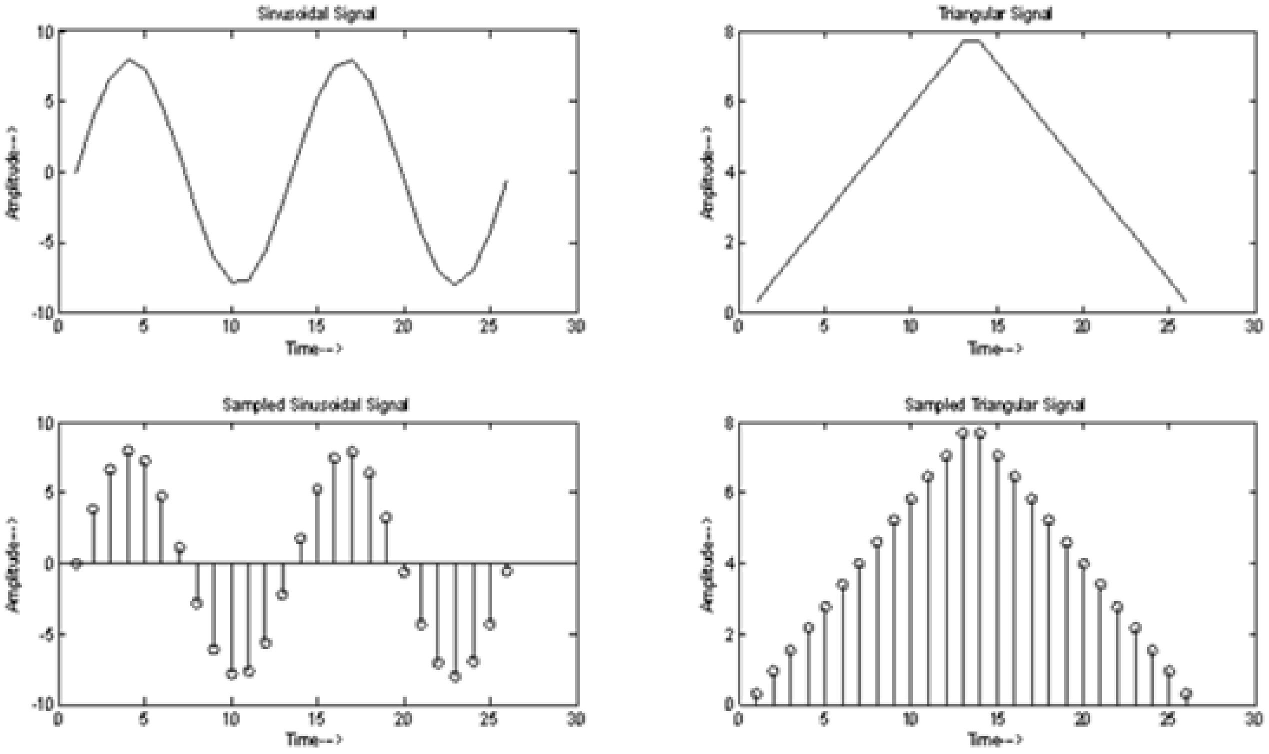
```
clc;
```

```
closeall;
clearall;
% Signal generation
x=0:.5:4*pi;                    % siganal taken upto 4pi
sig1=8*sin(x);                   % generate 1st sinusoidal signal
l=length(sig1);
sig2=8*triang(l);                 % Generate 2nd traingularSigal

% Display of Both Signal
subplot(2,2,1);
plot(sig1);
title('Sinusoidal Signal');
ylabel('Amplitude--->');xlabel('Time--->');
subplot(2,2,2);
plot(sig2);
title('Triangular Signal');
ylabel('Amplitude--->');xlabel('Time--->');

% Display of Both Sampled Signal
subplot(2,2,3);
stem(sig1);
title('Sampled Sinusoidal Signal');
ylabel('Amplitude--->');xlabel('Time--->');
subplot(2,2,4);
stem(sig2);
title('Sampled Triangular Signal');
ylabel('Amplitude--->');xlabel('Time--->');
l1=length(sig1);
l2=length(sig2);

for i=1:l1
sig(1,i)=sig1(i);    % Making Both row vector to a matrix
sig(2,i)=sig2(i);
end

% TDM of both quantize signal
tdmsig=reshape(sig,1,2*l1);

% Display of TDM Signal
figure
stem(tdmsig);
title('TDM Signal');
ylabel('Amplitude--->');xlabel('Time--->');
```

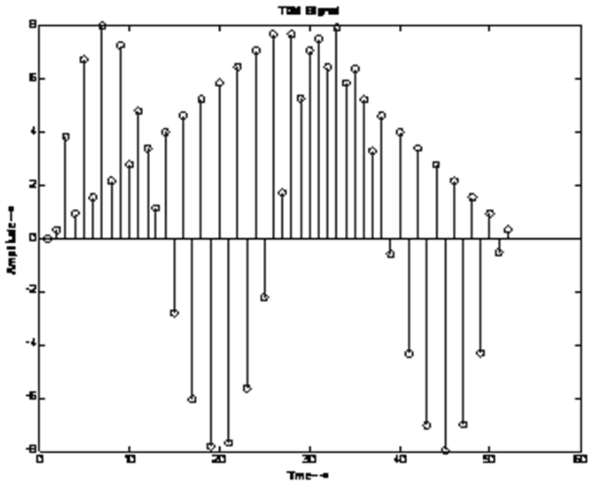
```
% Demultiplexing of TDM Signal
demux=reshape(tdmsig,2,l1);
for i=1:l1
sig3(i)=demux(1,i);        % Converting The matrix into row vectors
sig4(i)=demux(2,i);
end

% display of demultiplexed signal
figure
subplot(2,1,1)
plot(sig3);
title('Recovered Sinusoidal Signal');
ylabel('Amplitude--->');xlabel('Time--->');
subplot(2,1,2)
plot(sig4);
title('Recovered Triangular Signal');
ylabel('Amplitude--->');xlabel('Time--->');
```

Expected Output Waveform:

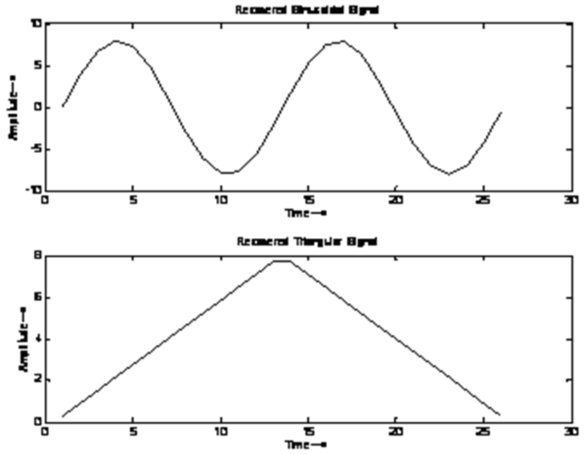
Input signal and the sampled version:



Multiplexed signal



Reconstructed signal



Observation:

b) Circuit realization of TDM

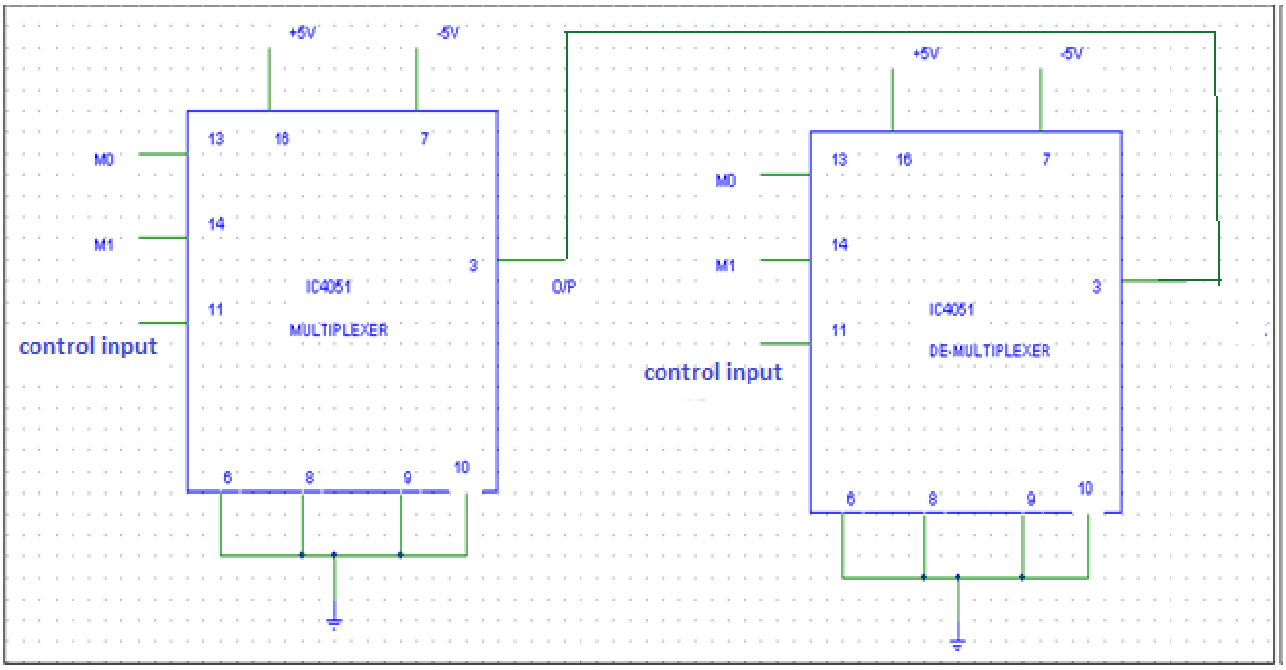
Aim:

To design and demonstrate the working of time division multiplexing for Pulse Amplitude ModulatedSignals using discrete components.

Apparatus Required:

Function Generator, CRO, Connecting wires, BNC Probes, dc power supply, IC 4051-2(Mux / De-Mux)

Circuit Diagram:



Procedure:

1) Connections are made as per the circuit diagram.A control input of Square wave signal with a frequency of 500Hz (5V) is applied to pin. No. 11 of both IC’s.
2) M0 - Sine wave signal with a frequency of 1KHz (5V)
3) M1 - Triangular wave signal with a frequency of 1KHz (5V)
4) Signals, M0 & M1 is provided as inputs to pin 13 and pin 14 of IC - 4051 Mux-circuit.
5) TDM output is fed as input to Pin 3 of Demux-circuit. The signals M0 and M1 are retrieved back at Pin 13 and Pin 14.

Observation:

Sl. No	**Criteria**	**Max Marks**	**Marks obtained**
Data sheet			
A	Problem statement	10	
B	Design & specifications	10	
C	Expected output	10	
Record			
D	Simulation/ Conduction of the experiment	15	
E	Analysis of the result	15	
	Viva	40	
	Total	100	
Scale down to 10 marks			

Experiment 8

Pulse code Modulation and Delta Modulation

Aim:

a) Generation of PCM signals with various quantization levels using Matlab &SIMULINK

b) Demonstrate Delta-Sigma modulator in Matlab & SIMULINK

a) Pulse code modulation

Theory:

Analog-to-digital conversion:

A digital signal is superior to an analog signal because it is more robust to noise and can easily be recovered, corrected and amplified. For this reason, the tendency today is to change an analog signal to digital data. In this section we describe two techniques, pulse code modulation and delta modulation

Pulse code Modulation (PCM):

Pulse-code modulation (PCM) is a method used to digitally represent sampled analog signals, which was invented by Alec Reeves in 1937. It is the standard form for digital audio in computers and various Blu-ray, Compact Disc and DVD formats, as well as other uses such as digital telephone systems. A PCM stream is a digital representation of an analog signal, in which the magnitude of the analogue signal is sampled regularly at uniform intervals, with each sample being quantized to the nearest value within a range of digital steps.

PCM consists of three steps to digitize an analog signal:

1. Sampling
2. Quantization
3. Binary encoding

- Before we sample, we have to filter the signal to limit the maximum frequency of the signal as it affects the sampling rate.
- Filtering should ensure that we do not distort the signal, ie remove high frequency components that affect the signal shape.

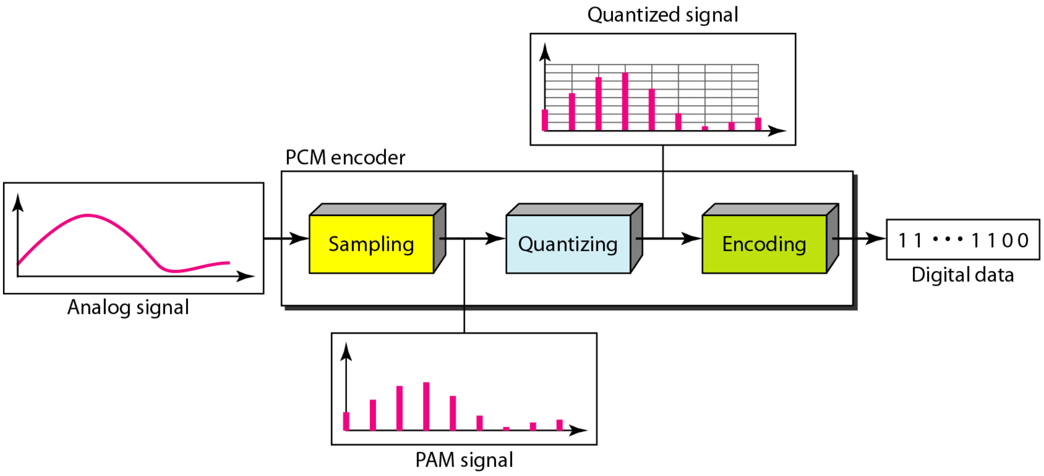


Figure 5.1: Components of PCM encoder

PCM Decoder:

- To recover an analog signal from a digitized signal we follow the following steps:
  - We use a hold circuit that holds the amplitude value of a pulse till the next pulse arrives.
  - We pass this signal through a low pass filter with a cutoff frequency that is equal to the highest frequency in the pre-sampled signal.
- The higher the value of L, the less distorted a signal is recovered.

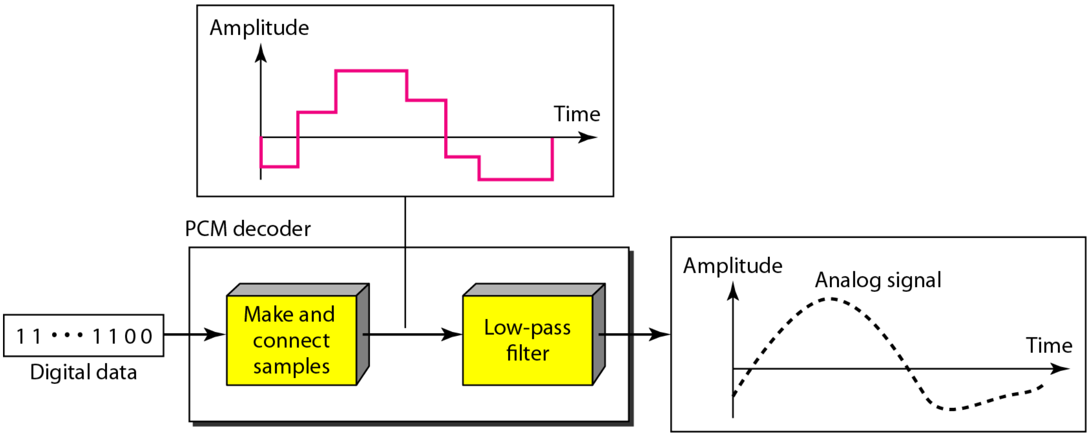


Figure 5.2: Components of PCM decoder

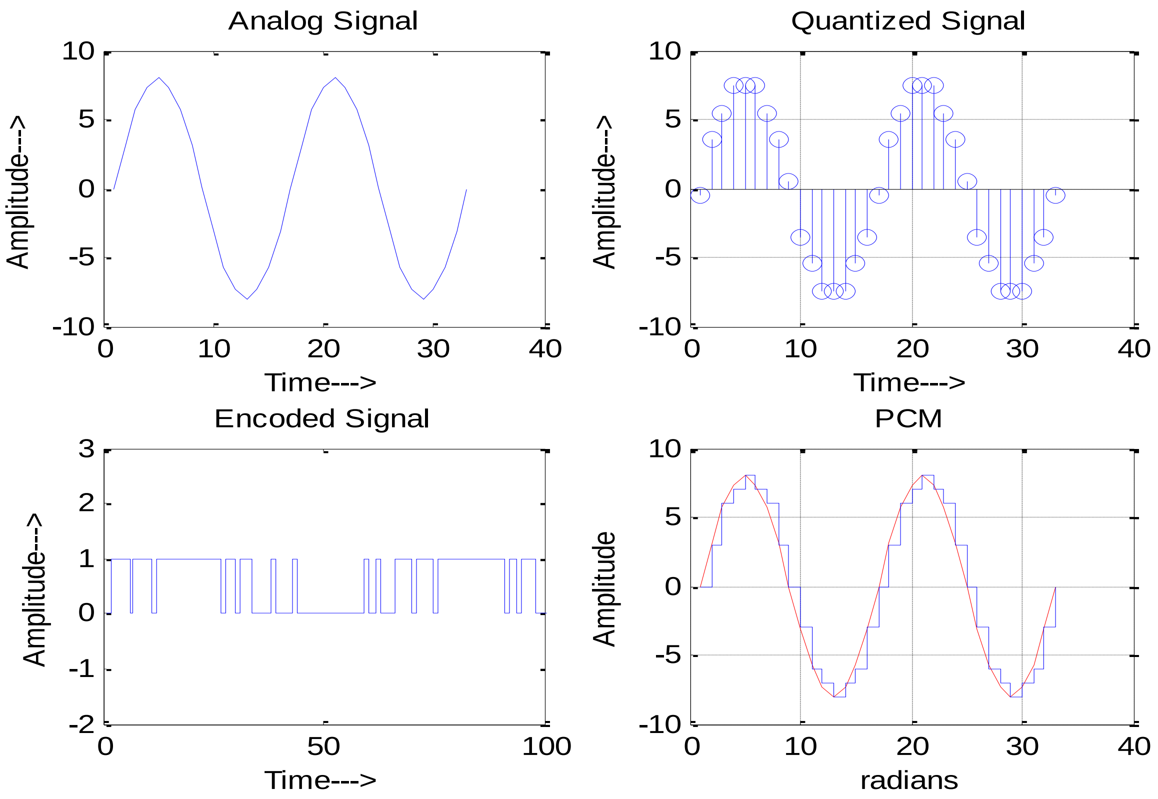
Program:

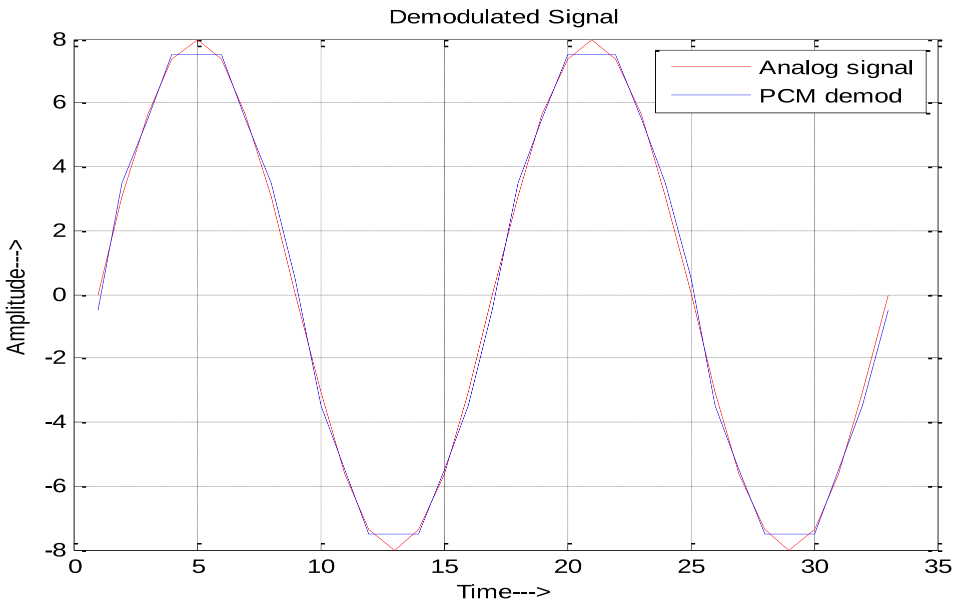
Main Program:

```
close all;
% Main program

clear all;
% n1=input('enter number of samples in a period :');%n1=16
n1=16;
t=0:2*pi/n1:4*pi;
a=8*sin(t);
[sqnr,a_quan]=u_pcm(a,8);%n-bit pcm (8bits or 16 bits)

%function u_pcm

function[sqnr,a_quan]=u_pcm(a,n);
% n=input('enter n value for n-bit pcm system :');
% n1=input('enter number of samples in a period :');
l=2^n;
subplot(2,2,1);
plot(a);
title('analog signal');
ylabel('amplitude--->'); xlabel('time--->');

vmax=8;
vmin=-vmax;
del=(vmax-vmin)/l;
part=vmin:del:vmax;
code=vmin-(del/2):del:vmax+(del/2);
[ind,q]=quantiz(a,part,code);

l1=length(ind);
l2=length(q);

for i=1:l1
if(ind(i)~=0)
ind(i)=ind(i)-1;
end
i=i+1;
end
for i=1:l2
   if(q(i)==vmin-(del/2))
      q(i)=vmin+(del/2);
   end
end
subplot(2,2,2);
stem(q);grid on;
title('quantized signal');
ylabel('amplitude--->'); xlabel('time--->');
```

```
code=de2bi(ind,'left-msb');
k=1;
for i=1:l1
    for j=1:n
        coded(k)=code(i,j);
        j=j+1;
        k=k+1;
    end
    i=i+1;
end
subplot(2,2,3);
grid on;
stairs(coded);
axis([0 100 -2 3]);
title('encoded signal');
ylabel('amplitude--->'); xlabel('time--->');

for i=1:length(a)
    a_quan(i)=round(a(i)/del)*del;
end
subplot(2,2,4)
plot(a,'r');
hold on;
stairs(a_quan);
title('PCM');
xlabel('radians');
ylabel('amplitude');
grid on;

figure(2);
qunt=reshape(coded,n,length(coded)/n);
index=bi2de(qunt','left-msb');
q=del*index+vmin+(del/2);
plot(a,'r');
hold on;
grid on;
plot(q);
legend('analog signal','PCM demod')
title('PCM DEMODULATION')
ylabel('amplitude--->');
xlabel('time--->');
sqnr=20*log10(norm(a)/norm(a-a_quan));
```

Expected Output Waveform:





Observation:

Non uniform PCM (µ law)

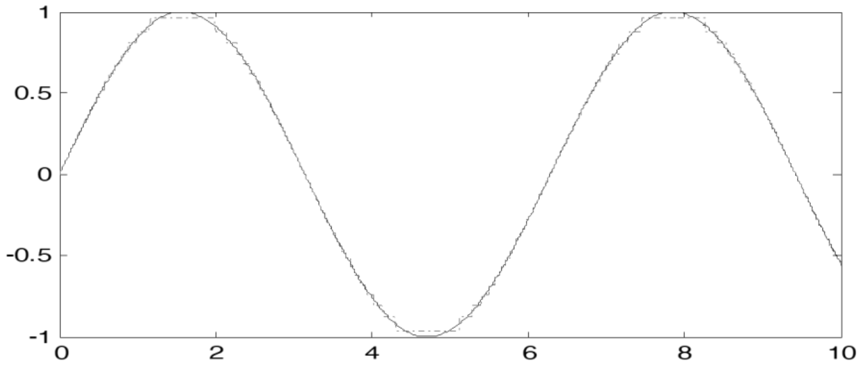
Program:

Main Program:

```
%non uniform
%main program
clear all;
clc;
% n1=input('enter number of samples in a period :');
n1=16;
t=0:2*pi/n1:4*pi;
a=8*sin(t);

[sqnr,aquan]=mula_pcm(a,16,255);%8 bit or 16 bit
figure(3);
plot(t,a,'-',t,aquan,'-');

%function mula_pcm
function [sqnr,a_quan]=mula_pcm(a,n,mu)
[y,maximum]=mulaw(a,mu);
[sqnr,y_q]=u_pcm(y,n);
a_quan=invmulaw(y_q,mu);
a_quan=maximum*a_quan;
sqnr=20*log10(norm(a)/norm(a-a_quan));
%function mulaw
function[y,a]=mulaw(x,mu)
a=max(abs(x));
y=(log(1+mu*abs(x/a))./log(1+mu)).*sign(x);

%function invmulaw


function x=invmulaw(y,mu)
```

x=(((1+mu).^(abs(y))-1)./mu).*sign(y);

Expected Output Waveform:



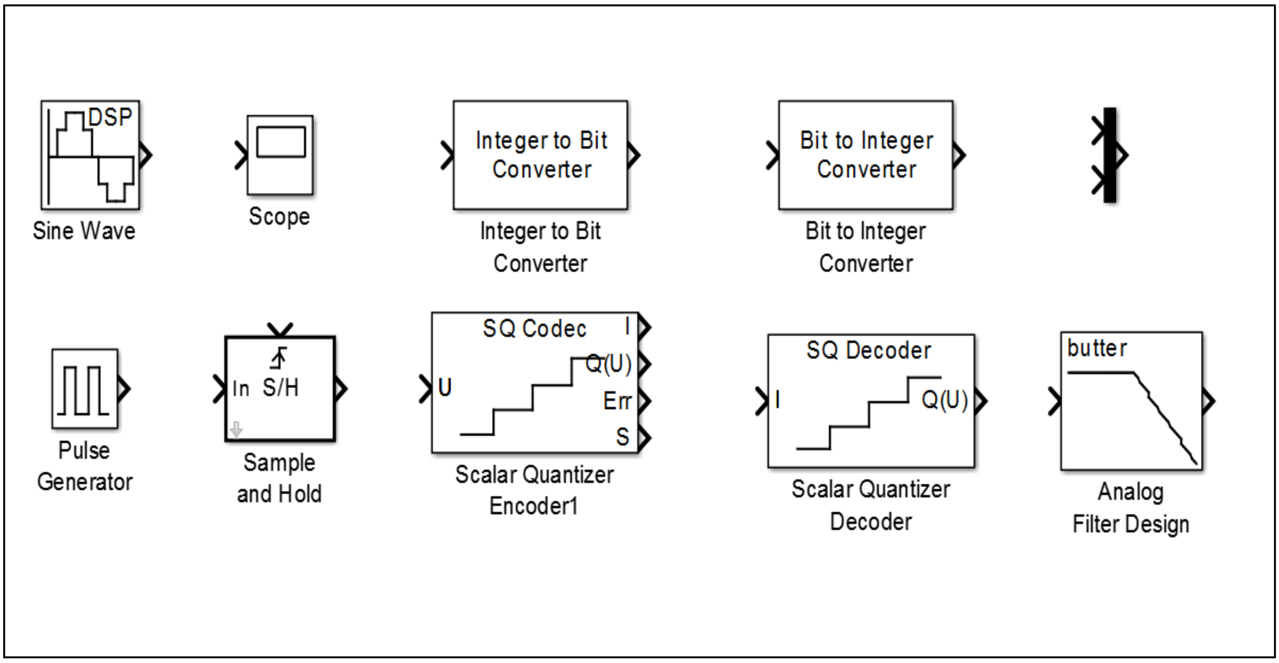
Simulink:

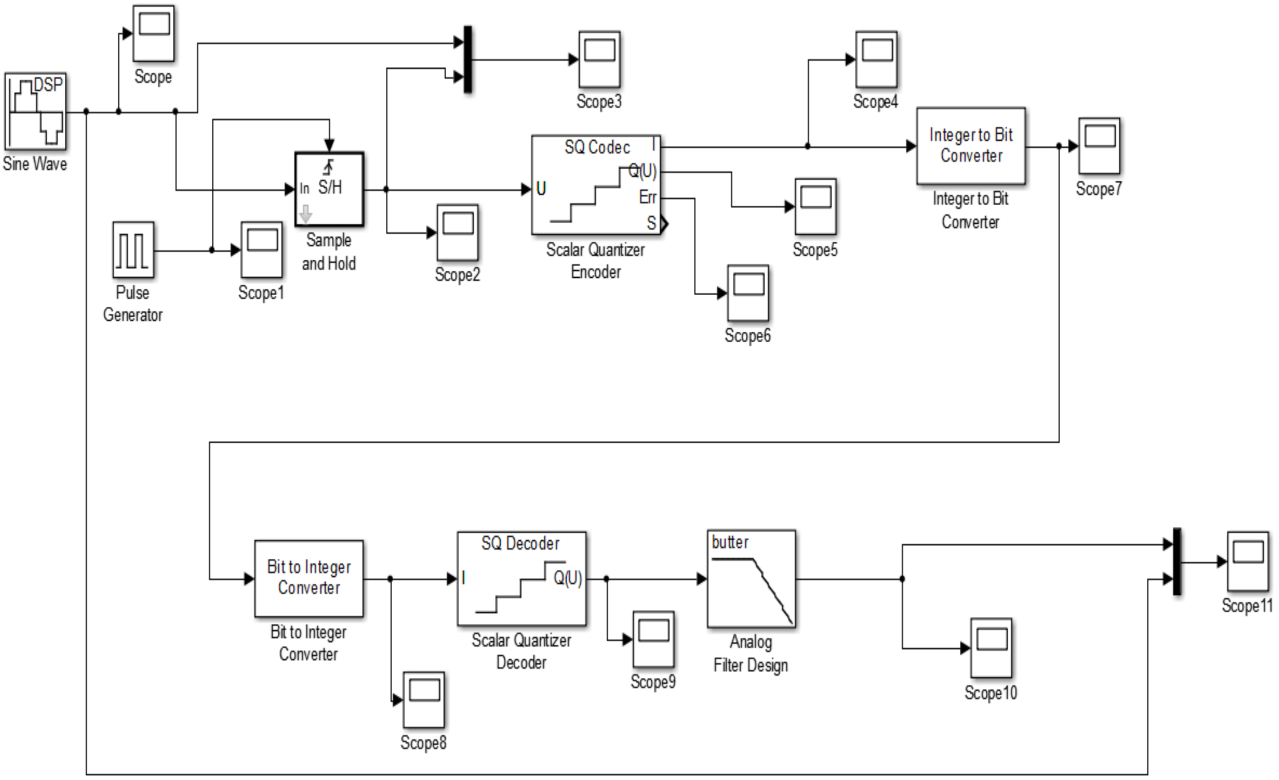
Procedure:

- Type **Simulink** in the Matlab Command Window to open a Simulink Library Browser
- Go to Signal Processing Blockset -> Signal Processing Sources
- Choose Sine wave as source. Right click on it and select add to untitled.

File -> save as -> give file name and save it with .mdl extension For ex: PCM_codec.mdl

In this experiment simulink is used to generate a whole PCM waveform (Transmitter and receiver). The below shown blocks could be found in Simulink Library Browser:





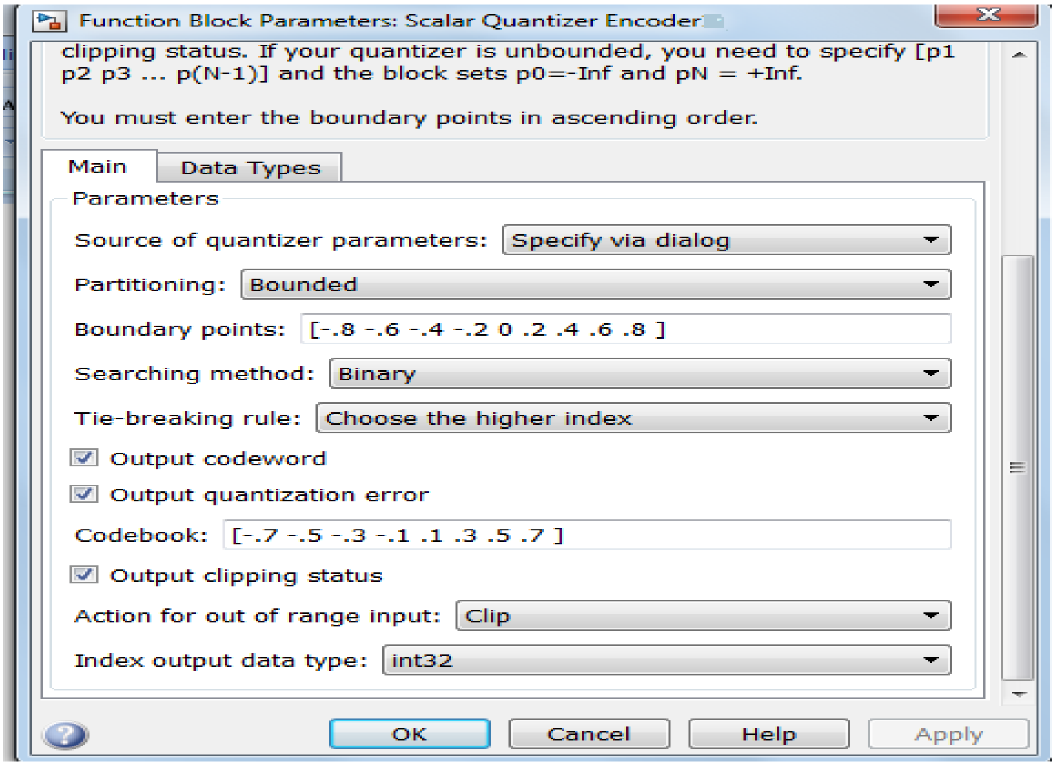
Input parameters are:

(a) Sine wave
Amplitude: 0.8 V
Frequency: 5 Hz
Phase offset: 0 radians
Sample mode: Discrete
Output complexity: Real
Computation method: Trigonometric fcn
Sample time: 1/1000
Samples per frame: 1

(b) Pulse Generator
Pulse type: Time based
Time: Use simulation time
Amplitude (V): 1
Period (secs): 0.01
Pulse width (%of period): 1
Phase delay (secs): 0
Choose the bottom check box.

(c) Sample and Hold
Trigger type: raising edge
Initial condition:0

(d) Scalar Quantizer Encoder



Function Block Parameters: Scalar Quantizer Encoder

clipping status. If your quantizer is unbounded, you need to specify [p1 p2 p3 ... p(N-1)] and the block sets p0=-Inf and pN = +Inf.

You must enter the boundary points in ascending order.

Source of quantizer parameters: Specify via dialog

Partitioning: Bounded

Boundary points: [-.8 -.6 -.4 -.2 0 .2 .4 .6 .8]

Searching method: Binary

Tie-breaking rule: Choose the higher index

☑ Output codeword

☑ Output quantization error

Codebook: [-.7 -.5 -.3 -.1 .1 .3 .5 .7]

☑ Output clipping status

Action for out of range input: Clip

Index output data type: int32

(e) Integer to Bit Converter
Number of bits per integer: 3

(f) Bit to integer Converter
Number of bits per integer: 3

(g) Scalar Quantizer decoder
Source of codebook: Specify via dialog
Quantization codebook: [-.7 -.5 -.3 -.1 .1 .3 .5 .7]

(h)Analog Filter Design
Design method: Butterworth
Filter type: Low pass
Filter order: 20
Passband edge frequency (rad/s):2*3.142*10

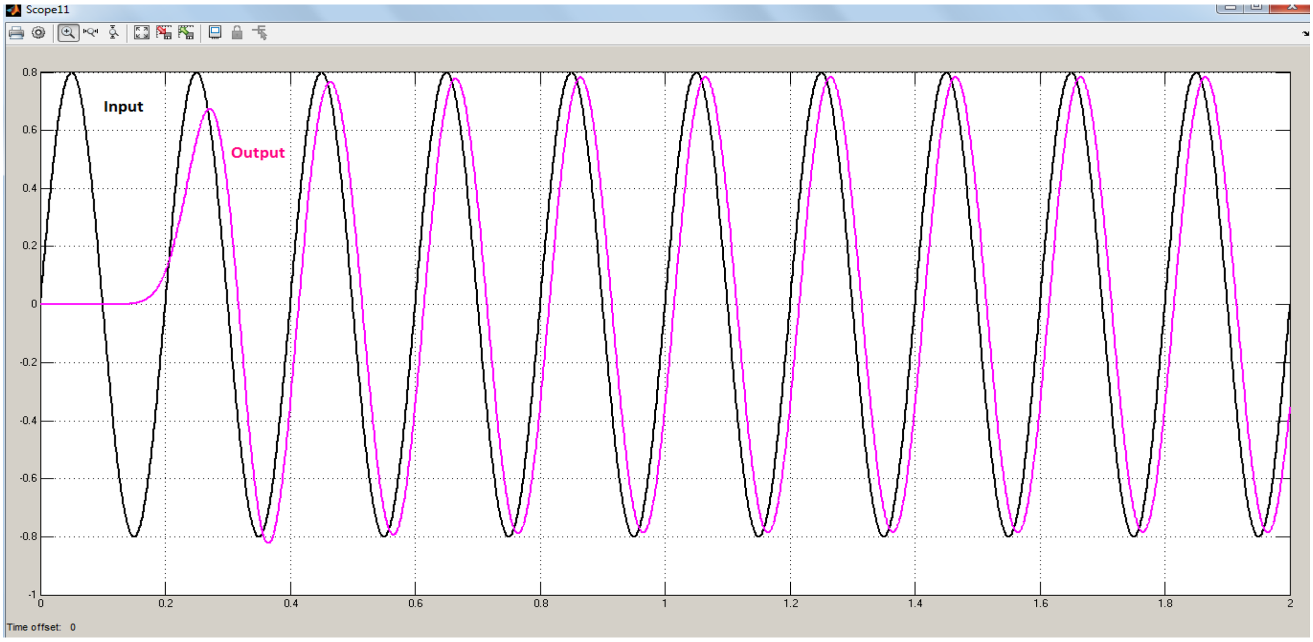
If the filtered signal (scope 10) is not the same as the original signal, change fc to get the desiredsignal (try and error method).

* Keep in mind that the filtered signal is affected by the following factors: a) Run time b) Frequency of original signal c) Cutoff frequency fc

- Set the simulation time to 2s, Click on Start Simulation/Run icon at the top of the window

- Check Autoscale in the Axes to see the scope output. The signal at various stages can be seen using multiple scopes at various points.

Expected Output Waveform:



Observation:

b) Delta-Sigma modulator

Aim:

To DemonstrateDelta-Sigma modulator in Matlab &SIMULINK

Theory:

Delta Modulation:

Delta modulation (DM or Δ-modulation) is an analog-to-digital and digital-to-analog signal conversion technique used for transmission of voice information where quality is not of primary importance. DM is the simplest form of differential pulse-code modulation (DPCM) where the difference between successive samples is encoded into n-bit data streams. In delta modulation, the transmitted data is reduced to a 1-bit data stream.

- This scheme sends only the difference between pulses, if the pulse at time t_{n+1} is higher in amplitude value than the pulse at time t_n, then a single bit, say a “1”, is used to indicate the positive value.
- If the pulse is lower in value, resulting in a negative value, a “0” is used.
- This scheme works well for small changes in signal values between samples.
- If changes in amplitude are large, this will result in large errors.

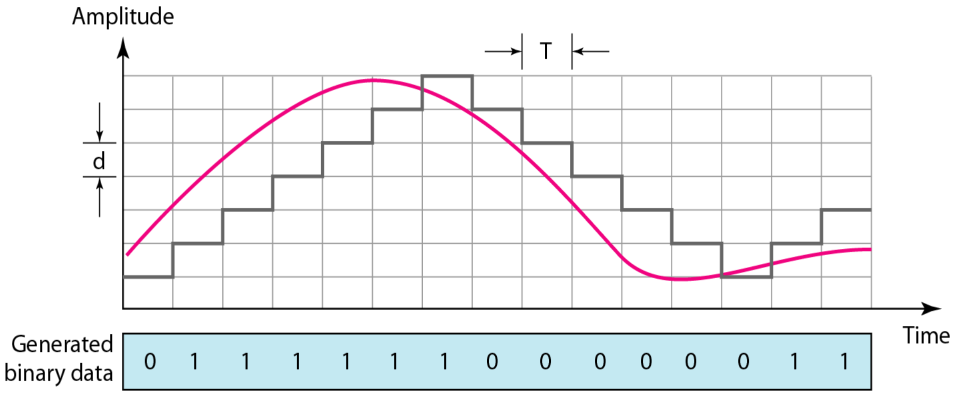


Figure 6.1:The process of delta modulation

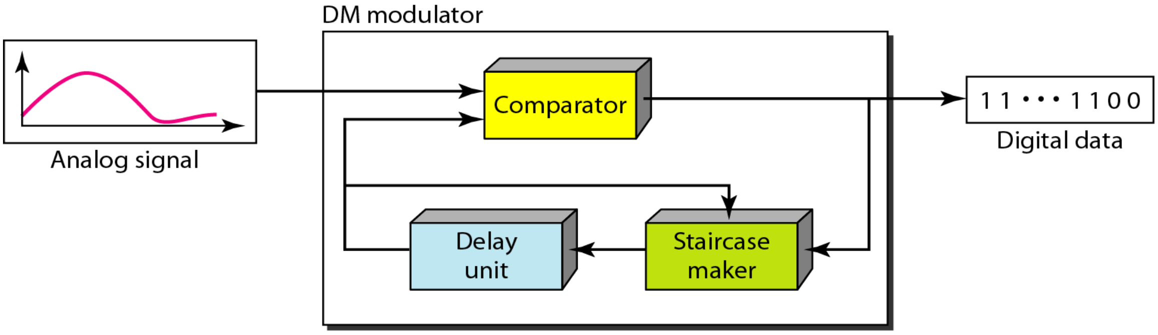


Figure 6.2:components of Delta modulation

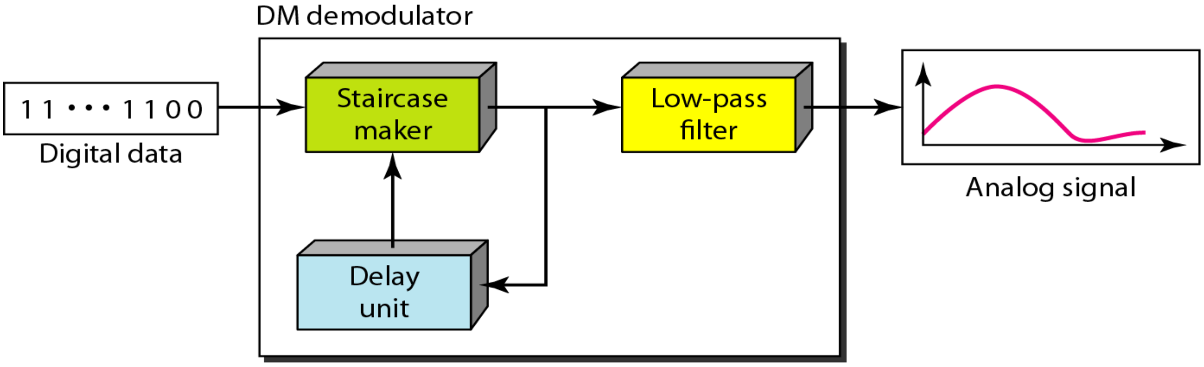


Figure 6.3: components ofDelta demodulation

MATLAB Program:

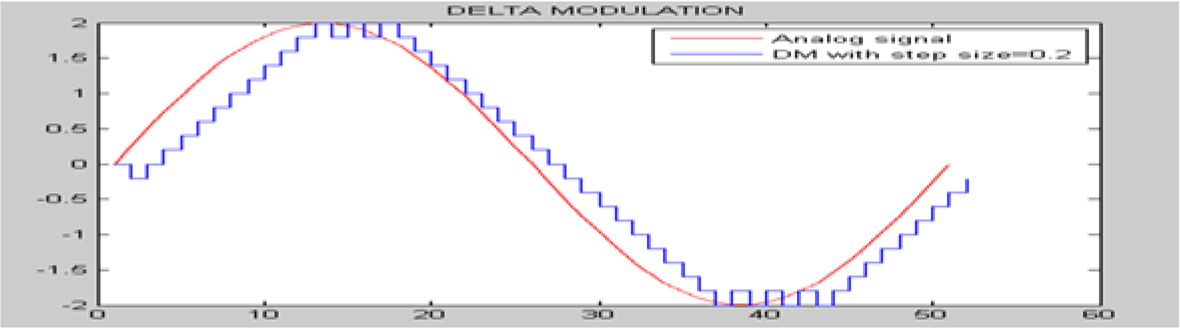
Main Program:

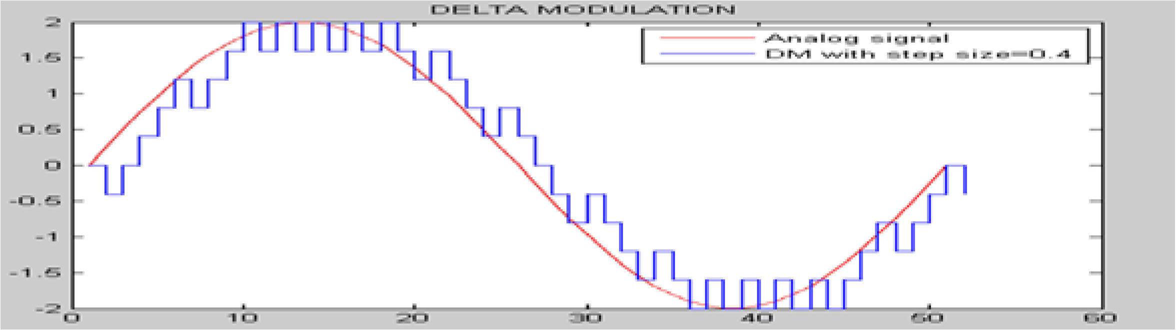
```
clc;
        clear all;
        close all;
        a=2;
        t=0:2*pi/50:2*pi; % Signal Generation
        x=a*sin(t);
        l=length(x);
        plot(x,'r');
        delta=0.2;
        %delta1=2*delta;%Apply delta modulation with doubling the step size
        %delta2=3*delta;
        hold on
        xn=0;
        for i=1:l;
        if x(i)>xn(i)
        d(i)=1;
        xn(i+1)=xn(i)+delta;
        else
        d(i)=0; xn(i+1)=xn(i)-delta;
```

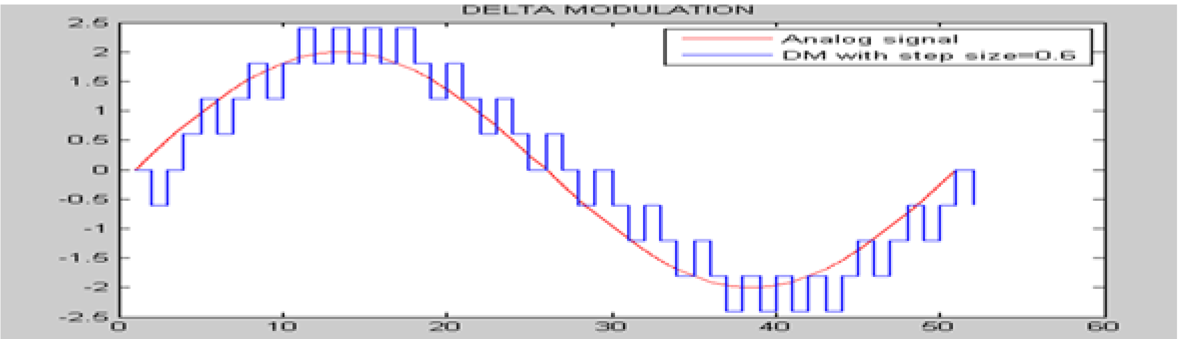
end

```
end
stairs(xn)
hold on
legend('Analog signal','DM with step size=0.2')
title('DELTA MODULATION')
```

Expected Output Waveform:







Observation:

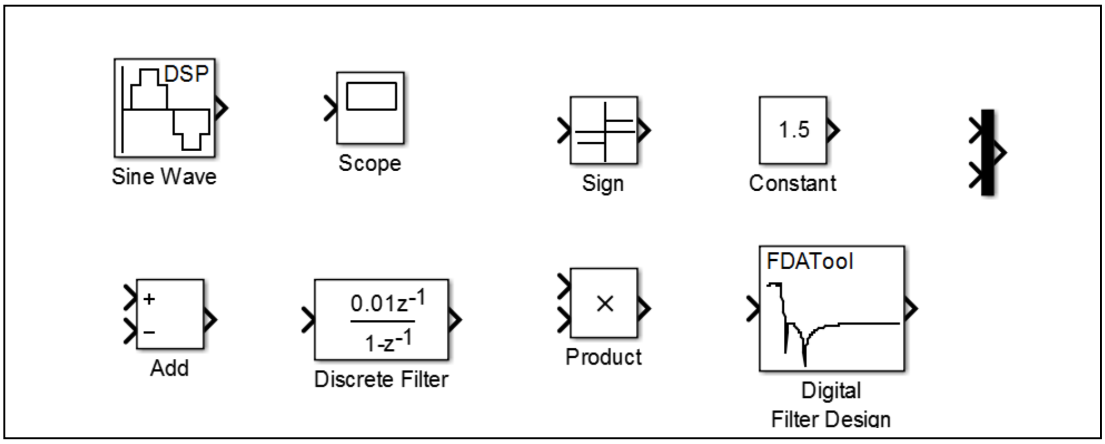
Simulink:

Procedure:

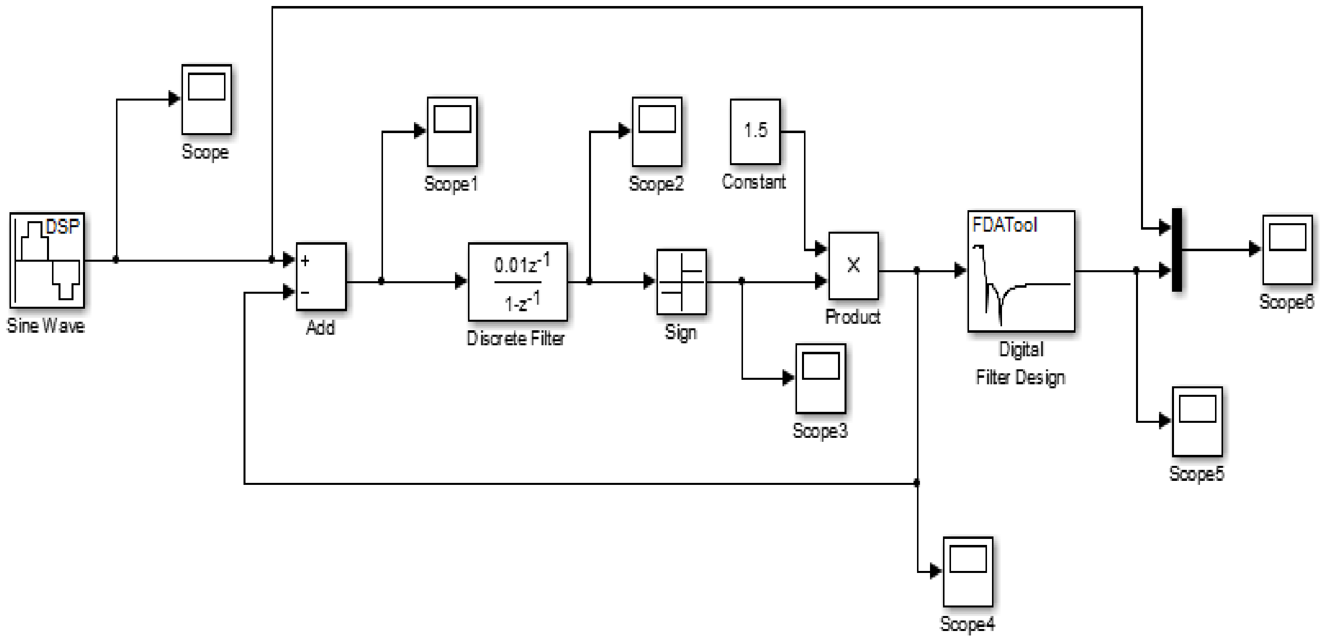
- Type **Simulink** in the Matlab Command Window to open a Simulink Library Browser
- Go to Signal Processing Blockset -> Signal Processing Sources
- Choose Sine wave as source. Right click on it and select add to untitled.

File -> save as -> give file name and save it with .mdl extension For ex: DM_codec.mdl

In this experiment simulink is used to generate a whole DM waveform (Transmitter and receiver).Thebelow shown blocks could be found in Simulink Library Browser:



DM Block (sampling,quantization, encoding)
Transmitter & Receiver:
Construct the model as shown in the figure below:



The Input parameters are:

(a) Sine wave
Amplitude: 1 V
Frequency: 6/(2*3.142)Hz % 6radians
Phase offset: 0 radians
Sample mode: Discrete
Output complexity: Real
Computation method: Trigonometric fcn
Sample time: 1/100
Samples per frame: 1
Resetting states when re-enabled: at time zero

(b) **Add**
List of signs: +-

(c) **Discrete Filter**
Block parametersshown below

(d) **Sign**
Enable zero-crossing detection
Sample time: -1

(e) **Constant**
Constant value: 1.5

(f) **Product**

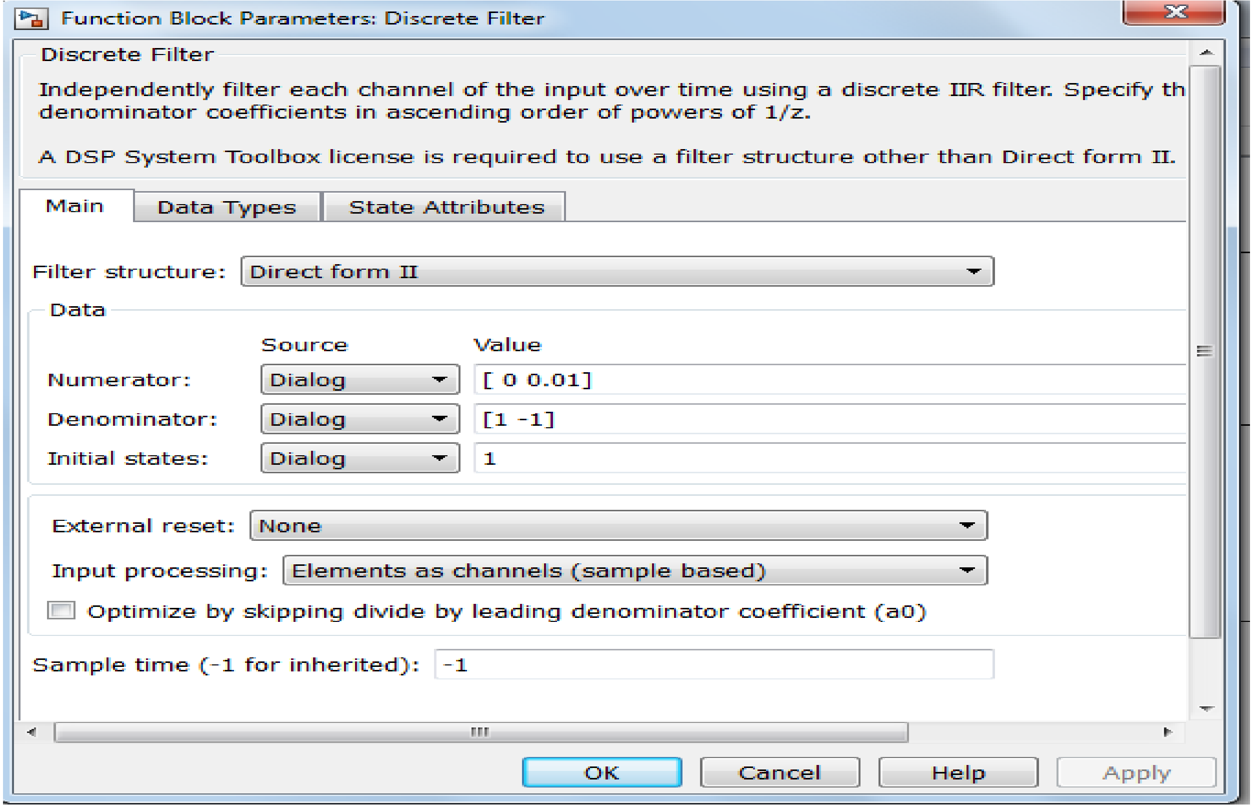
Number of inputs: 2

Multiplication: Element -wise

Sample time: -1

(g) Digital Filter Design

Block parameters shown below



Function Block Parameters: Discrete Filter

Discrete Filter

Independently filter each channel of the input over time using a discrete IIR filter. Specify th denominator coefficients in ascending order of powers of 1/z.

A DSP System Toolbox license is required to use a filter structure other than Direct form II.

Filter structure: Direct form II

Data

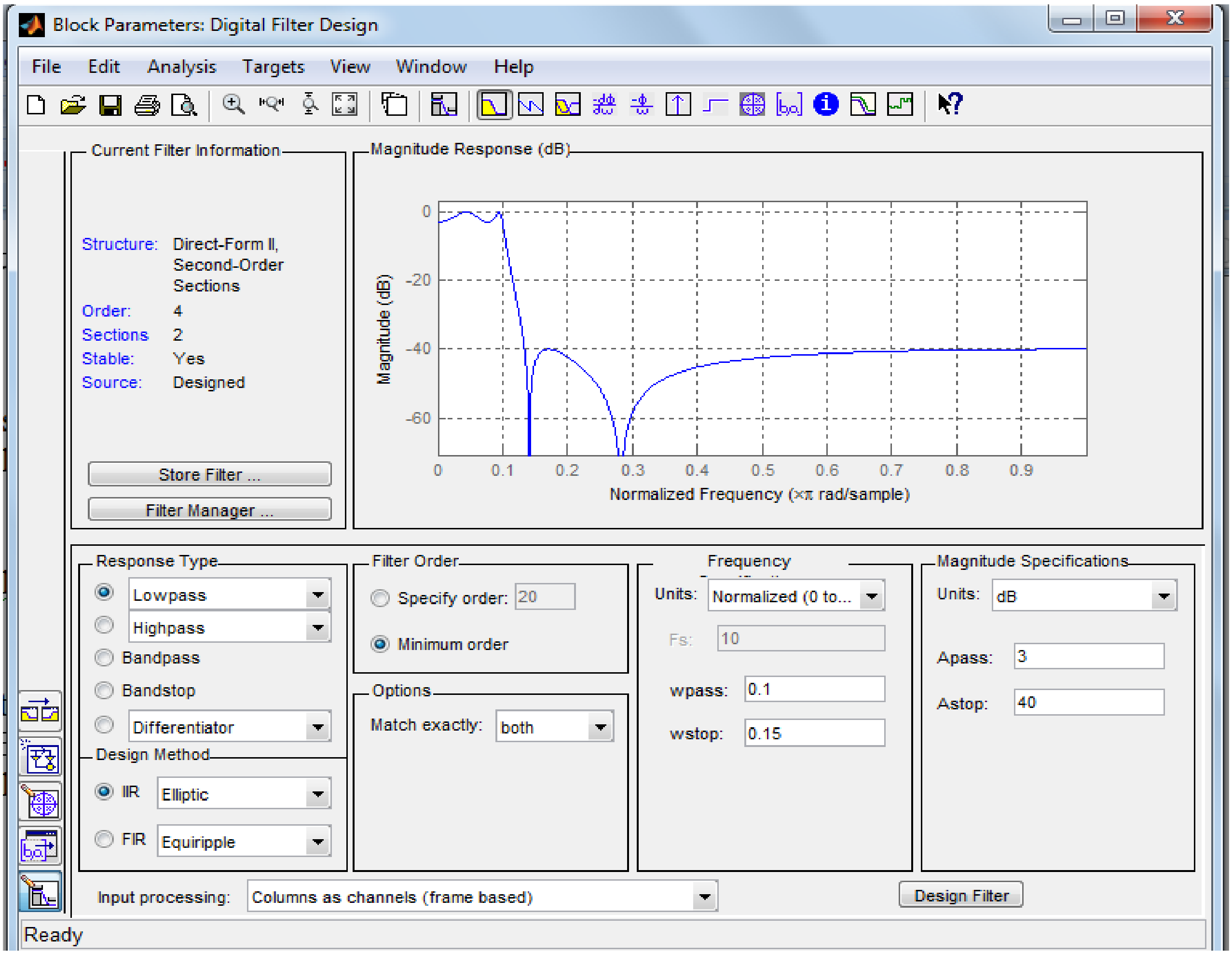
	Source	Value
Numerator:	Dialog	[0 0.01]
Denominator:	Dialog	[1 -1]
Initial states:	Dialog	1

External reset: None

Input processing: Elements as channels (sample based)

□ Optimize by skipping divide by leading denominator coefficient (a0)

Sample time (-1 for inherited): -1



Structure: Direct-Form II, Second-Order Sections
Order: 4
Sections 2
Stable: Yes
Source: Designed

Response Type: ● Lowpass ○ Highpass ○ Bandpass ○ Bandstop ○ Differentiator

Design Method: ● IIR Elliptic ○ FIR Equiripple

Filter Order: ○ Specify order: 20 ● Minimum order

Options: Match exactly: both

Frequency: Units: Normalized (0 to... Fs: 10 wpass: 0.1 wstop: 0.15

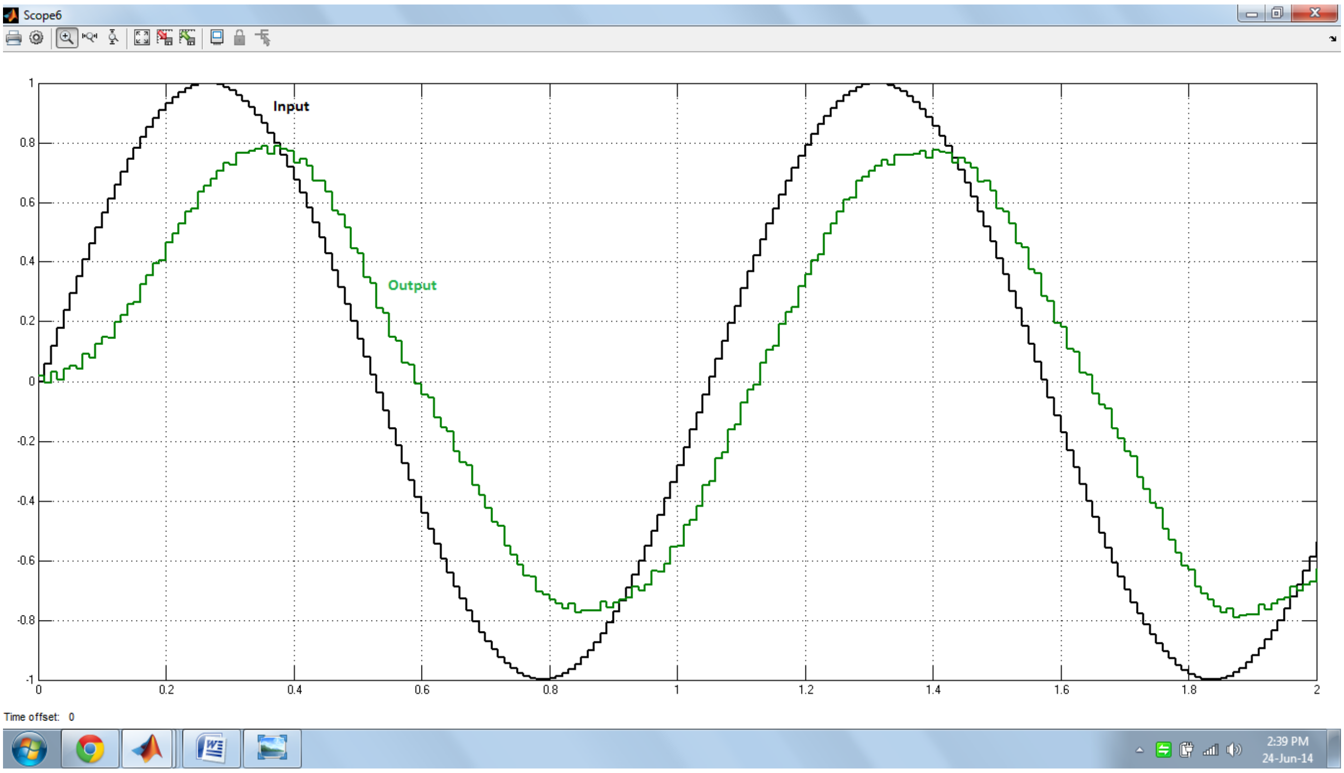
Magnitude Specifications: Units: dB Apass: 3 Astop: 40

Input processing: Columns as channels (frame based)

The Digital Filter Design block now represents a IIR lowpass filter. The filter passes all frequencies up to 10% of the Nyquist frequency (half the sampling frequency), and stops frequencies greater than or equal to 15% of the Nyquist frequency as defined by the **wpass** and **wstop** parameters.

Elliptic IIR Lowpass Filter
Minimum order
wp=0.1, ws=0.15 (Normalized)
Ap=3dB, As=40dB

Expected Output Waveform:



Observation:

Sl. No	**Criteria**	**Max Marks**	**Marks obtained**
Data sheet			
A	Problem statement	10	
B	Design & specifications	10	
C	Expected output	10	
Record			
D	Simulation/ Conduction of the experiment	15	
E	Analysis of the result	15	
	Viva	40	
	Total	100	
Scale down to 10 marks			

Experiment 9

Generation of Noise and studying its properties

Aim:

To simulate Additive white Gaussian noise and characterize its Properties.

Theory:

Additive white Gaussian Noise and Noise properties

In signal processing, noise is defined as an undesired signal. Noise sources often are characterized by their probability density function (pdf). The pdf describes the distribution of probability in terms of integrals. The pdf function is nonnegative everywhere, and its integral from negative infinity to positive infinity is 1, which means that the probability of the event occurring between this range is certain.

Gaussian noise is statistical noise having a probability density function (PDF) equal to that of the normal distribution, which is also known as the Gaussian distribution. In other words, the values that the noise can take on are Gaussian-distributed.The probability density function p of a Gaussian random variable z is given by:

$$p_G(z) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(z-\mu)^2}{2\sigma^2}}$$

where μ is the mean value and σ is the standard deviation.

A special case is white Gaussian noise, in which the values at any pair of times are identically distributed andstatistically independent (and hence uncorrelated). In communication channel testing and modeling, Gaussian noise is used as additive white noise to generate additive white Gaussian noise.Additive white Gaussian noise (AWGN) is a basic noise model used in Information theory to mimic the effect of many random processes that occur in nature. This noise denotes specific characteristics:

a) 'Additive' because it is added to any noise that might be intrinsic to the information system.

b) 'White' refers to idea that it has uniform power across the frequency band for the information system. It is an analogy to the color white which has uniform emissions at all frequencies in the visible spectrum.

c) 'Gaussian' because it has a normal distribution in the time domain with an average time domain value of zero.

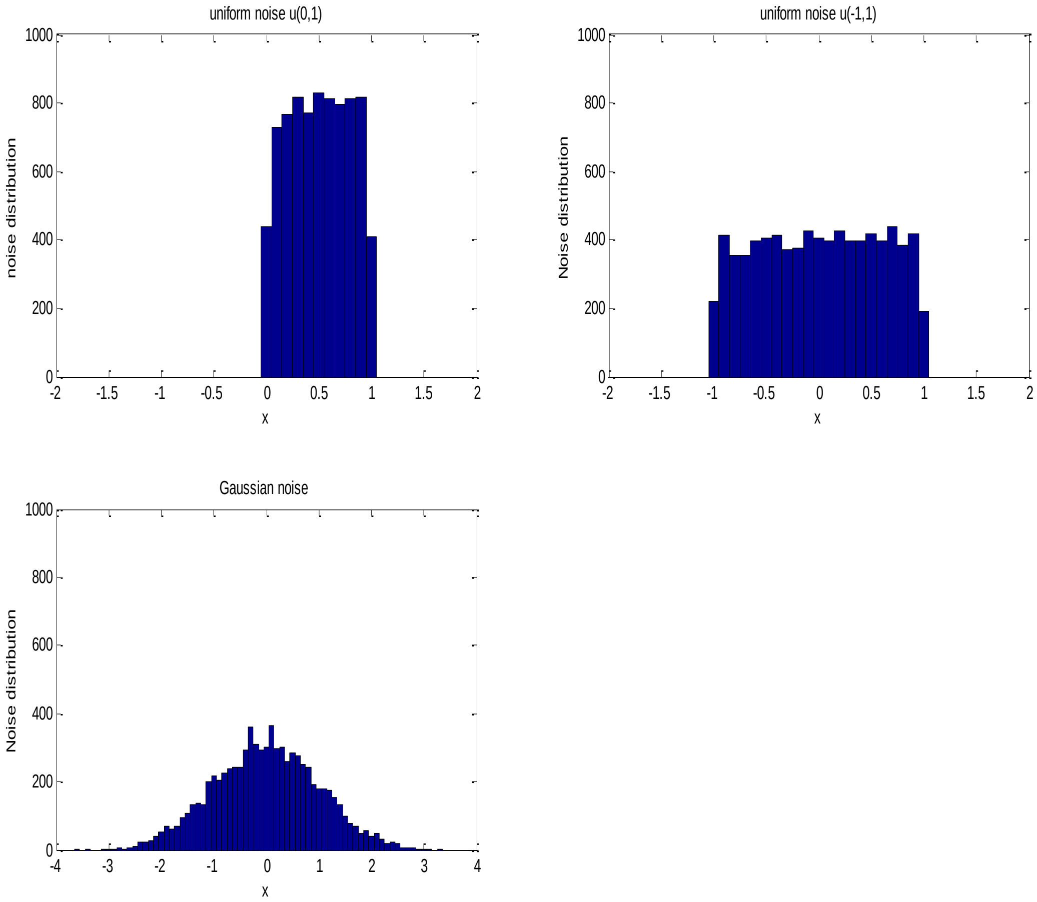
a) Different Types of Noise

Program:

```
% uniform Noise
clear all;close all;clc;
x=-4:0.1:4
%uniform noise with u(0,1)
```

```
        u_noise=rand(1,8000)
        subplot(221);
        hist(u_noise,x);

        %Uniform noise with u(-1,1)
        u_noise1=2*u_noise-1;
        subplot(222);
        hist(u_noise1,x);

        %Gaussian Noise
x=-4:0.1:4
        g_noise=randn(1,8000);
        subplot(223);
        hist(g_noise,x);
```

Expected Output Waveform:



b) Noise properties

Program:

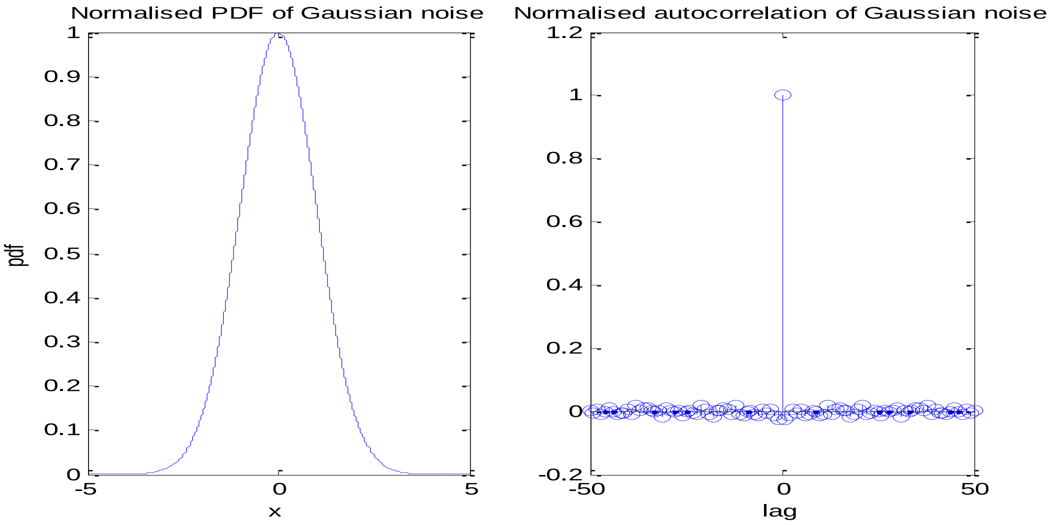
```
t=-10:0.01:10;
L=length(t);
n=randn(1,L);
nmean=mean(n);
disp('mean=');disp(nmean);
nmeansquare=sum(n.^2)/length(n);
disp('mean square=');disp(nmeansquare);
nstd=std(n);
disp('std=');disp(nstd);
nvar=var(n);
disp('var=');disp(nvar);
```

PDF and Autocorrelation of Gaussian Noise

Program:

```
clear;
clc;
%pdf of noise using Signal to noise ratio
snr=0.5;
es1=1;es2=0;
n0=es1^2/snr;
sigma=sqrt(n0/2);
m=0;
x=-5:0.001:5;
gpdf=(1/(sigma*(sqrt(2*pi)))).*exp((-(x-m).^2)/(2*(sigma)^2));
figure(1);
subplot(1,2,1);
plot(x,gpdf/max(gpdf));
title('Normalised PDF of Gaussian noise');
xlabel('x');
ylabel('pdf');
 l=50;
x=-4:0.1:4
g_noise=randn(1,8000);
subplot(1,2,2);
[acor,lags]=xcorr(g_noise,l);
stem(lags,acor/max(acor));
title('Normalised autocorrelation of Gaussian noise');
xlabel('lag');
```

Expected Output Waveform:



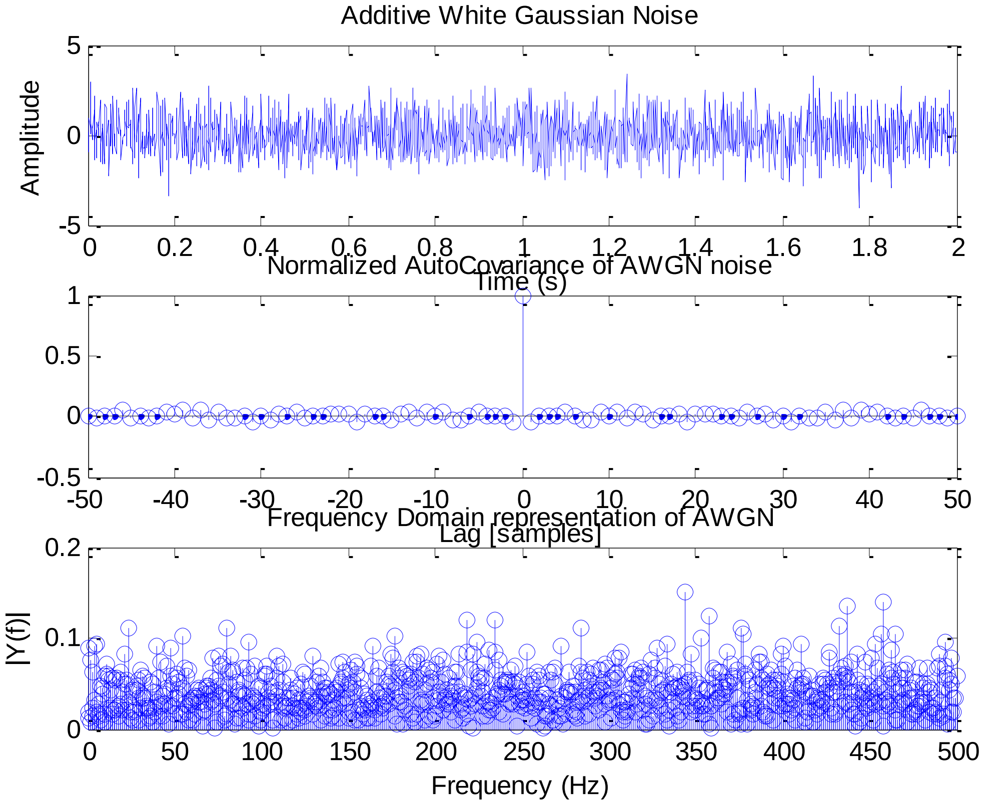
Covariance of AWGN

Program:

```
L=50 ;% Number of samples used in autocovariance calculation
clc;
Fs=1000; %sampling rate
Fc=10;   % carrier frequency for the dummy signal
t=0:1/Fs:2; %time base
variance = 1; %variance of white noise
noise=1;
signal=5*sin(2*pi*Fc*t);
% Generate Gaussian White Noise with zero mean and unit %variance
whiteNoise=sqrt(variance)*randn(1,length(signal));

%Calculate auto Covariance of the generated white noise
% L is the number of samples used in autocovariance calculation
[whiteNoiseCov,lags] = xcov(whiteNoise,L);
%Frequency domain representation of noise
noise=whiteNoise;
NFFT = 2^nextpow2(length(noise));
whiteNoiseSpectrum = fft(noise,NFFT)/length(noise);
f = Fs/2*linspace(0,1,NFFT/2+1);

%plotting commands
figure(1); subplot(3,1,1);plot(t,whiteNoise); title('Additive White Gaussian Noise');
xlabel('Time (s)');ylabel('Amplitude');

subplot(3,1,2);
stem(lags,whiteNoiseCov/max(whiteNoiseCov)); title('Normalized AutoCovariance
of AWGN noise');
xlabel('Lag [samples]');
subplot(3,1,3);
```

```
stem(f,2*abs(whiteNoiseSpectrum(1:NFFT/2+1))) ; title('Frequency Domain
representation of AWGN');
xlabel('Frequency (Hz)');   ylabel('|Y(f)|')
```

Expected Output Waveform:



Observation:

Sl. No	Criteria	Max Marks	Marks obtained
Data sheet			
A	Problem statement	10	
B	Design & specifications	10	
C	Expected output	10	
Record			
D	Simulation/ Conduction of the experiment	15	
E	Analysis of the result	15	
	Viva	40	
	Total	100	
Scale down to 10 marks			

Experiment 10

Line codes

Aim:

Simulate the different line codes for a sequence (Unipolar NRZ, Polar NRZ, Bipolar NRZ, Manchester) and plot the power spectral density and the probability of error for the same.

Theory:

The terminology **line coding** originated in telephony with the need to transmit digital information across a copper telephone *line;* more specifically, binary data over a digital repeatered line. The concept of line coding, however, readily applies to any transmission line or channel. In a digital communication system, there exists a known set of symbols to be transmitted.

It is commonly accepted that the dominant considerations effecting the choice of a line code are:

1. ***Timing:*** The waveform produced by a line code should contain enough timing information such that the receiver can synchronize with the transmitter and decode the received signal properly. The timing content should be relatively independent of source statistics, i.e., a long string of **1**s or **0** s should not result in loss of timing or jitter at the receiver.
2. ***DC content:*** Since the repeaters used in telephony are AC coupled, it is desirable to have zero DC in the waveform produced by a given line code. If a signal with significant DC content is used in AC coupled lines, it will cause **DC wander** in the received waveform. That is, the received signal baseline will vary with time. Telephone lines do not pass DC due to AC coupling with transformers and capacitors to eliminate DC ground loops. Because of this, the telephone channel causes a drop in constant signals. This causes DC wander. It can be eliminated by DC restoration circuits, feedback systems, or with specially designed line codes.
3. ***Power spectrum:*** The power spectrum and bandwidth of the transmitted signal should be matched to the frequency response of the channel to avoid significant distortion. Also, the power spectrum should be such that most of the energy is contained in as small bandwidth as possible. The smaller the bandwidth, the higher the transmission efficiency.
4. ***Performance monitoring****:* It is very desirable to detect errors caused by a noisy transmission channel. The error detection capability in turn allows performance monitoring while the channel is in use (i.e., without elaborate testing procedures that require suspending use of the channel).
5. ***Probability of error****:* The average error probability should be as small as possible for a given transmitter power. This reflects the reliability of the line code.

6. ***Transparency:*** A line code should allow all the possible patterns of **1**s and **0** s. If a certain pattern is undesirable due to other considerations, it should be mapped to a unique alternative pattern.

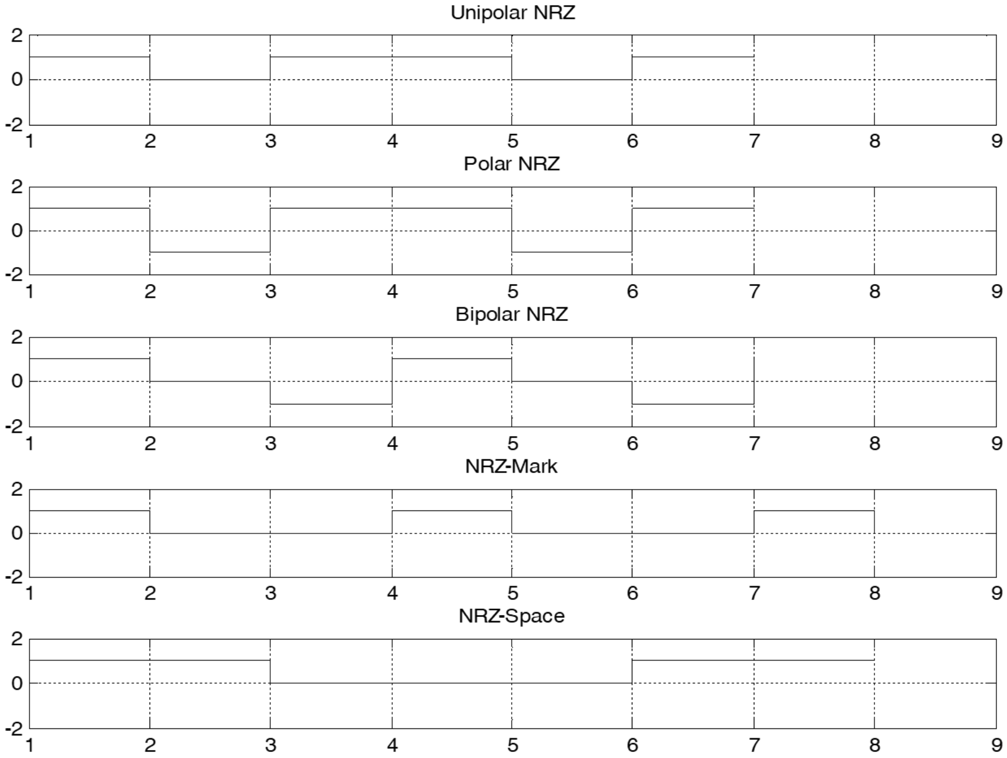
MATLAB Program:

```
%Input parameters
N= 10; % Number of input bits
a=floor(2*rand (1,N)) % generates random 1's and zero's and displays

A=5; % Pulse amplitude
Tb=1;  %bit period
fs=100; % Number of samples (even number) taken in a bitperiod

%Unipolar NRZ
U=[];
for k=1:N;
U = [U A*a(k)*ones(1,fs)];
end

%Unipolar RZ
U_rz=[];
for k=1:N;
    c = ones(1,fs/2);
    b = zeros(1,fs/2);
    p = [c b];
    U_rz = [U_rz A*a(k)*p];
end

%Polar NRZ
P=[];
 for k=1:N
 P = [P ((-1)^(a(k) + 1))*A*ones(1,fs)];
 end

 %Polar RZ
 P_rz=[];
for k = 1:N
   c = ones(1,fs/2);
   b = zeros(1,fs/2);
   p = [c b];
  P_rz = [P_rz ((-1)^(a(k)+1))*A*p];
end

%Bipolar NRZ
B=[];
count=-1;
for k= 1:N
   if a(k)==1
      if count==-1
         B = [B A*a(k)*ones(1,fs)];
         count=1;
      else
          B = [B -A*a(k)*ones(1,fs)];
```

```
            count=-1;
        end
    else
     B = [B A*a(k)*ones(1,fs)];
    end
end


%Bipolar RZ / AMI RZ
B_rz=[];
count=-1;
for k= 1:N
    if a(k)==1
        if count==-1
           B_rz = [B_rz A*a(k)*ones(1,fs/2) zeros(1,fs/2)];
           count=1;
        else
            B_rz = [B_rz -A*a(k)*ones(1,fs/2) zeros(1,fs/2)];
            count=-1;
        end
    else
     B_rz = [B_rz A*a(k)*ones(1,fs)];
    end
end

 %Split-phase or Manchester code
M=[];
        for k = 1:N
            c = ones(1,fs/2);
            b = -1*ones(1,fs/2);
            p = [c b];
            M = [M ((-1)^(a(k)+1))*A*p];
        end

T = linspace(0,N*Tb, length(U));% Time vector  % Lengths of all codes are same

figure(1)
 subplot(4, 1, 1); plot(T,U,'LineWidth',2)
axis([0 N*Tb -6 6])
title('Unipolar NRZ')
grid on
subplot(4, 1, 2); plot(T,U_rz,'LineWidth',2)
axis([0 N*Tb -6 6])
title('Unipolar RZ')
grid on
subplot(4, 1, 3); plot(T,P,'LineWidth',2)
axis([0 N*Tb -6 6])
title('Polar NRZ')
grid on
subplot(4, 1, 4); plot(T,P_rz,'LineWidth',2)
axis([0 N*Tb -6 6])
title('Polar RZ')
grid on

figure(2)
```

```
subplot(3, 1, 1); plot(T,B,'LineWidth',2)
axis([0 N*Tb -6 6])
title('Bipolar NRZ')
grid on
subplot(3, 1, 2); plot(T,B_rz,'LineWidth',2)
axis([0 N*Tb -6 6])
title('Bipolar RZ / RZ-AMI')
grid on
subplot(3, 1, 3); plot(T,M,'LineWidth',2)
axis([0 N*Tb -6 6])
title('Split-phase or Manchester code')
grid on
```

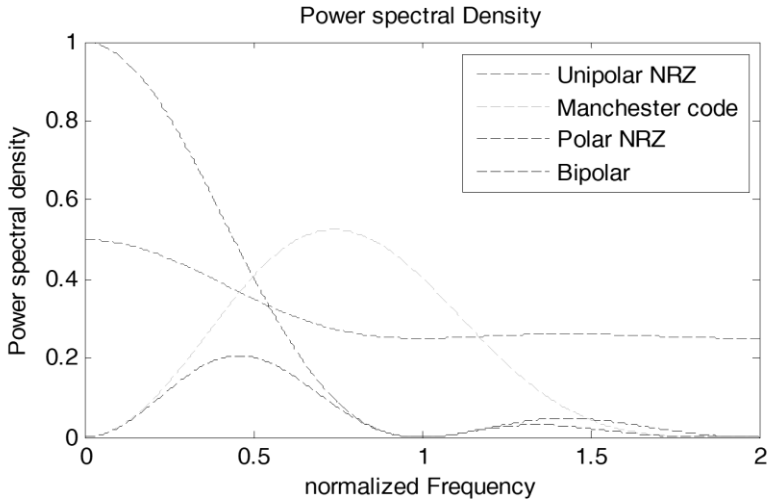
Expected Output Waveform:



MAT LAB Program to find PSD:

```
v=1;  % voltage level of a bit
R=1;  % Bitrate
T=1/R; % Bit period
f=0:0.001*R:2*R;  % frequency vector in terms of bit rate
f= f+1e-10; % Otherwise, sin(0)/0 is undefined
% PSD curves are plotted for Bitrate=1bps and Pulse amplitude=1V
%Unipolar NRZ
s=((v^2*T/4).*(sin(pi.*f*T)./(pi.*f*T)).^2);
s(1)=s(1)+(v^2/4);% corresponds to an impulse function of weight v^2/4 at f=0 added to s(f)
at f=0;
ff=0;
stem(ff,s(1),'*r','LineWidth',4)% sketching an impulse at f=0
hold on;
plot(f,s,'-r','LineWidth',2);
hold on;
%Manchester code
s=(v.^2.*T).*((sin(pi.*f*T/2)./(pi.*f*T/2)).^2).*(sin(pi.*f*T/2).^2);
plot(f,s,'--g','LineWidth',2);
hold on;
%Polar NRZ
s=((v^2*T).*(sin(pi.*f*T)./(pi.*f*T)).^2);
plot(f,s,'--b','LineWidth',2);
hold on;
%Bipolar RZ
s=(v.^2.*T/4).*((sin(pi.*f*T/2)./(pi.*f*T/2)).^2).*(sin(pi.*f*T).^2);
plot(f,s,'--k','LineWidth',2);
legend('Unipolar NRZ: impulse at at f=0','Unipolar NRZ',
'Manchestercode','PolarNRZ','Bipolar RZ/ RZ-AMI');
xlabel('Normalised frequency)');
ylabel('Power spectral density');
```

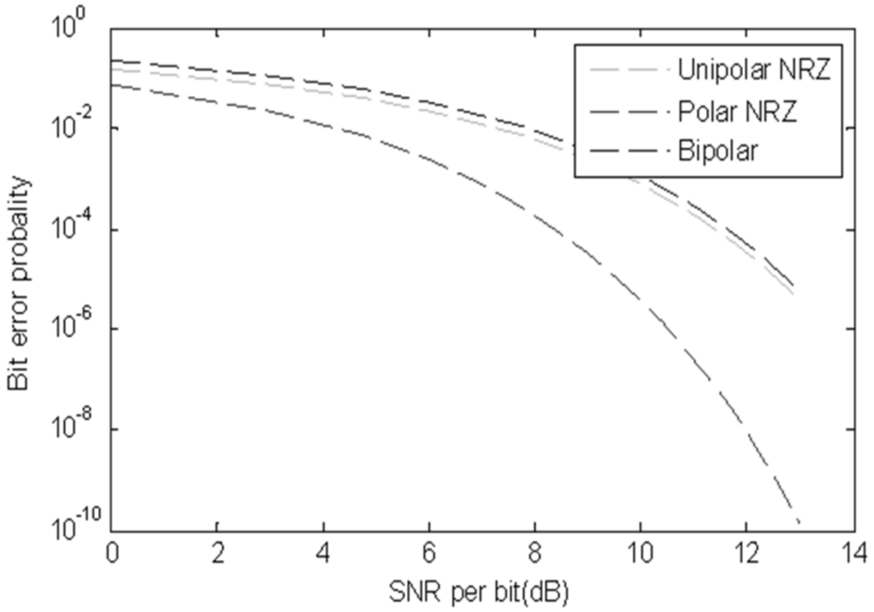
Expected output(Power spectral density):



MATLAB Program to find Probability of error of line codes:

```
%The probability of error, for equally likely data, with additive white Gaussian noise (AWGN) and  matched filter

%Unipolar NRZ
E=[0:1:25]; % Eb/N0=SNR of the recieved signal

%Unipolar NRZ
P1=(1/2)*erfc(sqrt(E/2));

%polar NRZ and Manchester code has same Pe for equiprobable 1's and 0's
P2=(1/2)*erfc(sqrt(E));

%Bipolar RZ/ RZ-AMI
P3=(3/4)*erfc(sqrt(E/2));

E=10*log10(E); % SNR in dB
semilogy(E,P1,'-k',E,P2,'-r',E,P3,'-b','LineWidth',2)
legend('Unipolar NRZ','Polar NRZ and Manchester','Bipolar RZ/ RZ-AMI','Location','best');
xlabel('SNR per bit, Eb/No(dB)');
ylabel('Bit error probality Pe');
```

Expected output(Probability of error):



Observation:

Sl. No	Criteria	Max Marks	Marks obtained
Data sheet			
A	Problem statement	10	
B	Design & specifications	10	
C	Expected output	10	
Record			
D	Simulation/ Conduction of the experiment	15	
E	Analysis of the result	15	
	Viva	40	
	Total	100	
Scale down to 10 marks			

Appendix-A

Simulink

Simulink, developed by MathWorks, is a data flow graphical programming language tool for modeling, simulating and analyzing multi-domain dynamic systems. Its primary interface is a graphical block diagramming tool and a customizable set of block libraries. It offers tight integration with the rest of the MATLAB environment and can either drive MATLAB or be scripted from it. Simulink is widely used in control theory and digital signal processing for multi-domain simulation and Model-Based Design.

Simulink® is a block diagram environment for multi domain simulation and Model-Based Design. It supports system-level design, simulation, automatic code generation, and continuous test and verification of embedded systems. Simulink provides a graphical editor, customizable block libraries, and solvers for modeling and simulating dynamic systems. It is integrated with MATLAB®, enabling you to incorporate MATLAB algorithms into models and export simulation results to MATLAB for further analysis.

A number of MathWorks and third-party hardware and software products are available for use with Simulink .It can automatically generate C source code for real-time implementation of systems. As the efficiency and flexibility of the code improves, this is becoming more widely adopted for production systems,[4][5] in addition to being a popular tool for embedded system design work because of its flexibility and capacity for quick iteration. Embedded Coder creates code efficient enough for use in embedded systems.[6][7][8]

Matlab:

MATLAB (matrix laboratory) is a numerical computing environment and fourth-generation programming language. Developed by MathWorks, MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages, including C, C++, Java, and Fortran.

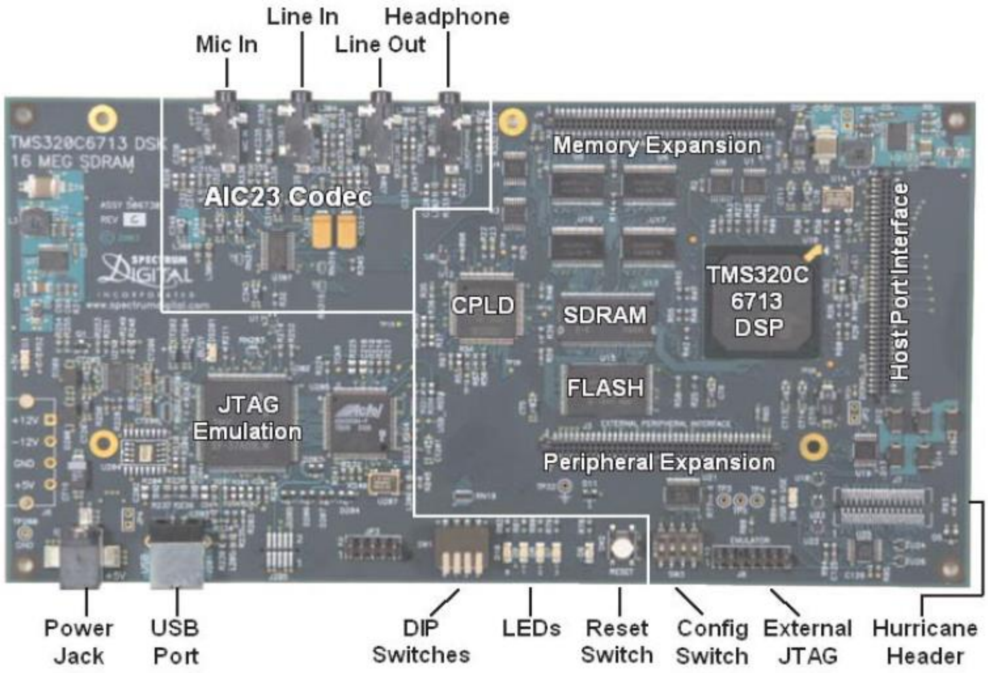
Relation between Matlab and Simulink:

Although MATLAB is intended primarily for numerical computing, an additional package, Simulink, adds graphical multi-domain simulation and Model-Based Design for dynamic and embedded systems. MATLAB is the programming environment, we need to program in the command window or m files. SIMULINK is used to do simulations, it has many blocks , you just need to drag and connect them as you need. Simulink is largely a controls oriented solution. It graphically depicts math like products, sums, integrals, etc. However, it's conditional logic facility is lacking. Matlab and simulink , both environments developed by MathWorks. It is used in academic and research institutions as well as industrial enterprises.

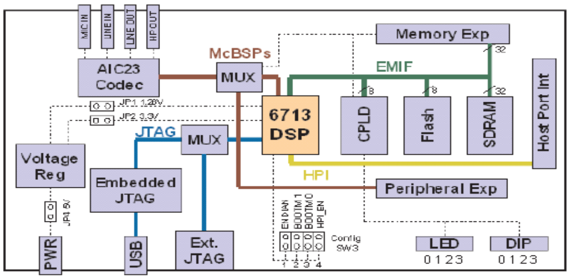
Appendix-B

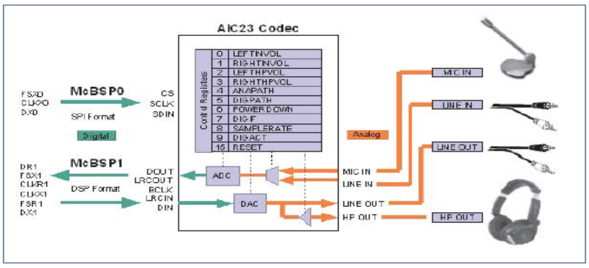
DSK6713

Board interfaces and pin details

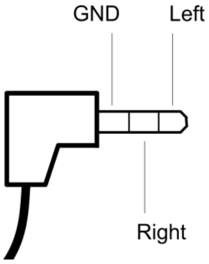


DSK 6713 & AIC23 Codec Schematic Diagram



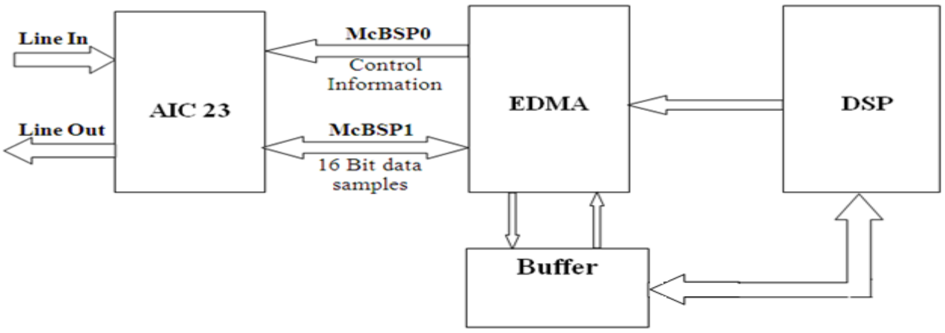


Stereo pin Configuration



Interconnection between AIC 23 and DSP

The figure below describes interconnection between Analog Interface Circuit 23(AIC 23) and DSP.



Audio data is transferred back and forth from the codec through McBSP1 (Multi Channel Buffered Serial port), a bidirectional serial port. The EDMA (Enhanced Direct Memory Access) is configured to take every 16-bit signed audio sample arriving on McBSP1 and store it in a buffer in memory until it can be processed. The DSP process the data in the buffer. Once it has been processed, it is sent back out through McBSP1 to the codec for output. One EDMA channel is used to transmit data to the codec while another is used to receive data from the codec. The McBSP0 is used to control the internal configuration registers of AIC 23. The scheme uses EDMA to relieve the DSP from the duty of data transfer. The EDMA controller takes incoming audio data directly from McBSP1 and places it in a memory buffer. It also takes data from a memory buffer and sends it to McBSP1 to generate the audio output. Separate EDMA channels are used to transmit and receive audio data.

The audio data is a series of 16-bit signed integers representing the amplitude of the input waveform at a particular point in time. Since the AIC23 is a stereo codec, the audio(or line) input consists of both left and right audio channels. Data is received in a frame consisting of two elements, one 16-bit sample from the left channel followed by one 16-bit sample from the right channel. Line In and Line out are used to input and take out the signal respectively.

Board support Library (BSL) Functions

The BSL provides a C-language interface for configuring and controlling all on-board devices. The library consists of discrete modules that are built and archived into a library file.Each module represents an individual API (Application Programming Interface) and is referred to simply as an API module. There will be separate APIs for different on board components and these APIs are used to access he respective hardware components. Note that an API is nothing but a function.

BSL Functions used in the program

1. void DSK6713_init()

Sets all CPLD registers to their power-on states and initializes internal BSL data structures. Must be called before any other BSL functions.

2.DSK6713_AIC23_CodecHandle DSK6713_AIC23_openCodec(int id, DSK6713_AIC23_Config *Config)

Parameters: id – Specifies which codec to use, on the DSK6713, id = 0.

Config – Pointer to structure containing codec register values. The function will initialize the codec registers with the values in the structure.

Return Value: Handle for the codec, used by the other codec module functions. The return type is integer value. The DSK6713_AIC23_CodecHandle is name given to integer data type.(refer dsk6713_aic23.h)

This function initializes all McBSP registers. If initialization is successful, this function returns TRUE value (Note that in C program there is no Boolean TRUE or FALSE. TRUE is defined as 1 and FALSE is defined as 0 in header files). This value act as handle for the Codec. Other functions check this value before doing there operation.

DSK6713_AIC23_Config is a structure which is defined as follows.
typedef struct DSK6713_AIC23_Config {
int regs[DSK6713_AIC23_NUMREGS];
} DSK6713_AIC23_Config;

Note that, DSK6713_AIC23_NUMREGS=10;(no. of registers in AIC 23 Codec)

3.Int16 DSK6713_AIC23_read(DSK6713_AIC23_CodecHandle hCodec, Int32 *val);
Parameters: hCodec – Codec handle.
val – Address of 16 bit signed variable to receive codec data.
Return Value: TRUE – Data read successfully.
FALSE – Data port is busy.

Read 32 bits of codec data, loop to retry if data port is busy
while(!DSK6713_AIC23_read(hCodec, &data));
4.Int16 DSK6713_AIC23_write(DSK6713_AIC23_CodecHandle hCodec, Int32 val);
Parameters: hCodec – Codec handle.
val – 32 bit value signed to write to codec.
Return Value :TRUE – Data written successfully.
FALSE – Data port is busy.

Write 32-bit to the codec, loop to retry if data port is busy
while(!DSK6713_AIC23_write(hCodec, data));

5. void DSK6713_AIC23_setFreq(DSK6713_AIC23_CodecHandle hCodec, Uint32 freq);
Parameters: hCodec – Codec handle.
freq – Sample rate of the codec clock.
The default sampling rate is 48 kHz. This can be done by loading value 0x0081 in to sample rate control register of AIC 23. This default value can be changed by using above function. In this program, we are setting sampling rate to 8 kHz by passing argument freq=1 to the API module.

6.void DSK6713_AIC23_closeCodec(DSK6713_AIC23_CodecHandle hCodec)
Parameters: hCodec – Codec handle.
Close codec module.

To support all BSL functions, dsk6713.h and dsk6713_aic23.h header files and libraryfile dsk6713bsl.lib are required.