Laboratory Manual

for

EC620 - Digital Signal Processing

B. Tech.

SEM. VI (EC)

Department of Electronics & Communication
Faculty of Technology
Dharmsinh Desai University, Nadiad

TABLE OF CONTENT

EXPERIMENT – 1

DISCRETE TIME SIGNALS

OBJECTIVES:

- To generate different discrete time (DT) signals
- To generate & plot ODD & EVEN parts of a discrete time signal.
- To calculate Energy of a given signal.
- To generate an interpolated signal using zero, step and linear interpolation method.

THEORY:

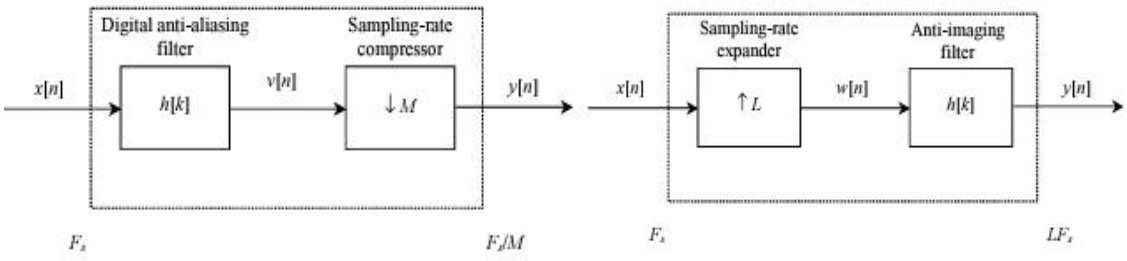
Signals are broadly classified into analog & discrete signals. An analog signal will be denoted by x(t), in which the variable t can represent any physical quantity. A discrete signal will be denoted by x(n), in which the variable n is integer valued and represents discrete instances in time. Therefore it is called a discrete time signal, which is a number sequence and will be denoted by one of the following notations:

$$x(n)=\{\ldots,x(-1),x(0),x(1),\ldots.\}$$

For various applications, Energy & Power of a signal is a useful measure.

E =n=-n=|x[n|2

Decimation & Interpolation are used to change the rate of a discrete signal. They are used in multirate signal processing.



1. (2)

Fig. Block diagram (1) Decimation, (2) Interpolation

SAMPLE PROGRAM:

1. **Signal Generation (Review)**

```
clc;
clear;
xdel(winsid());
//Impulse Signal
t1=-10:10;
x1=[zeros(1,10) 1 zeros(1,10)];
//scf();
subplot(133);
plot(t1,x1,'cya+','marker','d','markerfac','green','markeredg','red');
//Step Signal
n1=1,n0=50,n2=100;
```

```
if((n0<n1)|(n0>n2)|(n1>n2))
   error('arugument incorrect');
end
n=[n1:n2];
x2=[(n-n0)>=0,1];
subplot(132);
plot(n,x2(n1:n2),'cya+','marker','d','markerfac','green','markeredg','red');
//Ramp Signal
t3=0:10;
x3=t3;
//scf();
subplot(234);
plot(t3,x3,'cya+','marker','d','markerfac','green','markeredg','red');
```

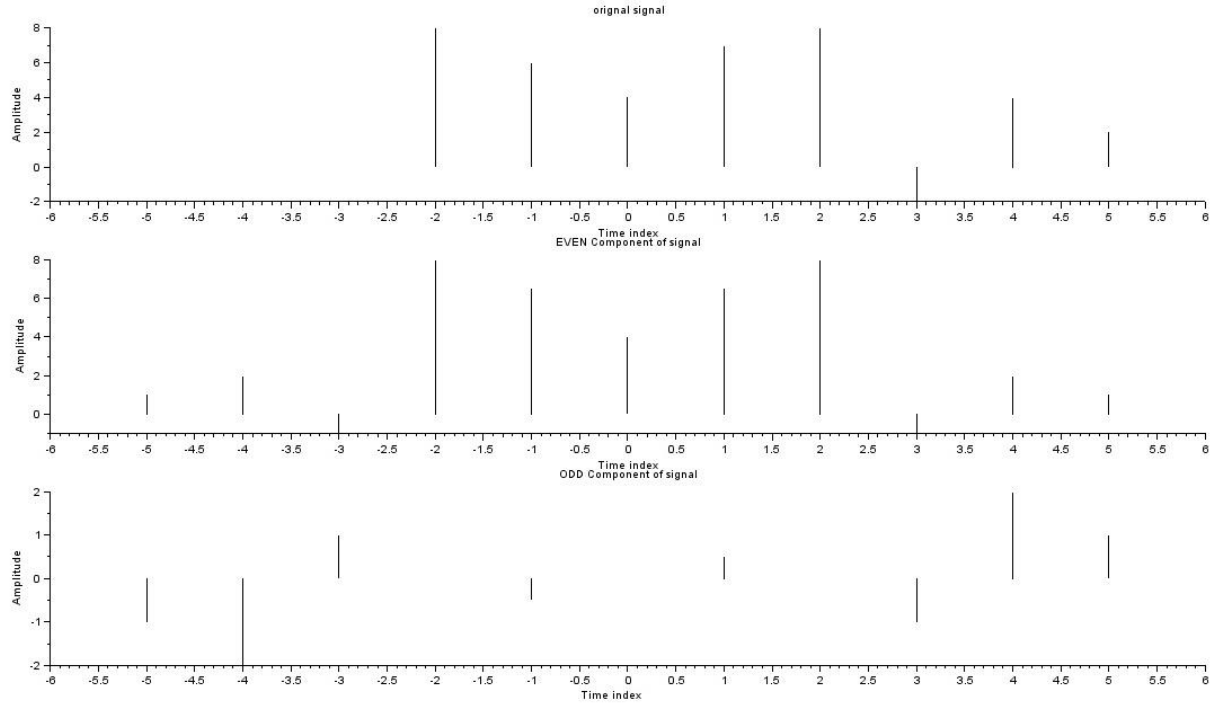
OUTPUT:

2. To separate EVEN & ODD Part of Discrete Signal

```
clc;
close;
clear;
x=input("enter array:") // [8 6 4 7 8 -2 4 2]
N=input("enter index:") // [-2 -1 0 1 2 3 4 5]
[r,ind]=find(N==0);
disp(ind)// Position of Origin of the signal
leng=int((length(x)/2))

if leng+1>ind then
   z=[zeros(1,abs(2*ind-length(x))+1),x]
end
if leng<ind then
   z=[x,zeros(1,int(ind-length(x)/2))]
end
n=-int(length(z)/2):1:int(length(z)/2)
y=flipdim(z,2)
xe=(z+y)/2
x0=(z-y)/2
subplot(3,1,1)
plot2d3(n,z(n+int(length(z)/2)+1))
xtitle( 'orignal signal', 'Time index', 'Amplitude')
subplot(3,1,2)
plot2d3(n,xe(n+int(length(z)/2)+1))
xtitle( 'even part of signal', 'Time index', 'Amplitude')
subplot(3,1,3)
plot2d3(n,x0(n+int(length(z)/2)+1))
xtitle( 'odd part of signal', 'Time index', 'Amplitude')
```

OUTPUT:



3. To calculate Energy of a given signal.

```
clc
clear
close
x=input("Enter The Signal: ")  // [8 6 4 7 8 -2 4 2]
N=length(x);
for i=1:N
   E=E+(x(i)*x(i));
disp(“The Energy of Signal =”,E)
```

4. Method of Interpolation : ZERO, STEP, LINEAR

```
clc
clear
close
x_n=input("Enter Signal: ")  // [8 6 4 7 8 -2 4 2]
i_f=input("interpolation factor = ")  // 3
disp("Select Interpolation Method") // 3
disp("for ZERO Interpolation, type:1")
disp("for STEP Interpolation, type:2 ")
disp("for LINEAR Interpolation, type:3")
choice=input("Enter your Choice:")
select choice
case 1 then
//zero interpolation
x_z=[ ]
for i=1:1:length(x_n)
```

```
    x_z($+1)=x_n(i)
    for j=1:1:(i_f-1)
        x_z($+1)=0
    end
end
x_z=x_z'
disp("ZERO Interpolation:")
disp(x_z)
case 2 then
//step interpolation
x_s=[]
for i=1:1:length(x_n)
    x_s($+1)=x_n(i)
    for j=1:1:(i_f-1)
        x_s($+1)=x_n(i)
    end
end
x_s=x_s'
disp("STEP Interpolation:")
disp(x_s)
case 3 then
//linear interpolation
x_n($+1)=0
x_l=[]

for i=1:1:(length(x_n)-1)
    x_l($+1)=x_n(i)
    step=(x_n(i+1)-x_n(i))/i_f
    for j=1:1:(i_f-1)
        x_l($+1)=x_l($)+step
    end
end
x_l=x_l'
disp("LINEAR Interpolation:")
disp(x_l)
end
```

OUTPUT:

LINEAR Interpolation:

column 1 to 7

8. 7.3333333 6.6666667 6. 5.3333333 4.6666667 4.

column 8 to 14

5. 6. 7. 7.3333333 7.6666667 8. 4.6666667

column 15 to 22

1.3333333 - 2. 0. 2. 4. 3.3333333 2.6666667 2.

column 23 to 24

1.3333333 0.6666667

MODIFICATIONS:

1. **Generate Discrete Exponential, Sine wave and random signal.**
2. **Write a program to perform decimation operation on the discrete signal.**

CONCLUSION:

EXERCISES:

1. Generate Sine waves with 50 Hz, 75 Hz and100 Hz and Addition of both for 0.1 sec.
2. Generate rect(n/2N) and for N=5.

EXPERIMENT – 2

Discrete Time Systems

OBJECTIVES:

- **To Find output of Discrete Time(DT) system represented by given difference equation**
- **To implement DT system to achieve desired response.**

THEORY:

Discrete time systems are categorized as Infinite Impulse response (IIR) and Finite impulse response(FIR) filters. The linear time invariant (LTI)filters can be represented in following three ways:

1. Difference equation:

FIR:

$y(n) = B_0x(n) + B_1x(n-1) + B_2x(n-2) + \ldots + B_Mx(n-M)$

IIR :

$y(n) + A_1y(n-1) + A_2y(n-2) + \ldots A_Ny(n-N) = B_0x(n) + B_1x(n-1) + \ldots B_Mx(n-M)$

2. Impulse response:

$h(n) + A_1h(n-1) + A_2h(n-2) + \ldots A_Nh(n-N) = B_0\delta(n) + B_1\delta(n-1) + \ldots B_M\delta(n-M)$

3. Transfer function:

$$H(z) = \frac{B_0 + B_1z^{-1} + \ldots B_Mz^{-M}}{1 + A_1z^{-1} + A_2z^{-2} + \ldots + A_Nz^{-N}}$$

In most cases the desired response may be achieved by either FIR or IIIR filter by finding appropriate values of coefficients.

SAMPLE PROGRAM - 1:

```
clc
clear
y(1)=10

for n=2:20
   x(n)=0.4^(n-2)
    y(n)=x(n)+0.6*y(n-1);
end
n=[2:20]
y1=-2*(0.4)^(n-2)+9*(0.6)^(n-2)
disp(y)
disp(y1')
```

Output:

SAMPLE PROGRAM - 2:

```
y(1)=12
y(2)=0
x=ones(1,20)
for n=3:20
    y(n)=4*x(n)+(1/6)*y(n-1)+(1/6)*y(n-2);
end
n=[3:20]
y1=-1.2*(0.5)^(n-3)+1.2*(-1/3)^(n-3)+6
disp(y)
```

Output:

SAMPLE PROGRAM - 3:

```
x=[0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0]
y(1)=0
y(2)=0
for n=1:16
   y(n+2)= x(n+2)+0.166*y(n+1)+0.166*y(n)


end
disp(y)
plot(y)
```

Output:

Conclusion:

EXPERIMENT – 3

CORRELATION & APPLICATIONS

OBJECTIVES:

- To perform correlation of two discrete sequences.
- To detect the object's presence in RADAR using correlation.
- To detect transmitted binary symbol at receiver.

THEORY:

Correlation operation between two discrete sequences:
Correlation is a measure of similarity between two signals and is found using a process similar to convolution. The discrete correlation (denoted **) of x[n] and h[n] is defined by

$$r_{xy} = x[n] ** h[n] = \sum_{k=-\infty}^{\infty} x(k) * h(k - n)$$

Correlation is equivalent to performing the convolution of x[n] with flipped signal h[-n].

RADAR Signal Processing:
Correlation is used to detect object in space. The transmitter continuously sends signals. If the object is present there is reflection. The system continuously performs correlation of transmitted signal and received signal. If no object is present, there is only noise as received signal. If an object is present we will get finite correlation.

Binary Symbol Detection at Receiver:
Due to finite possible symbols in digital communication, the receiver correlates the received signal with all this possibilities. The receiver will decided upon the symbol which gives the maximum correlation.

SAMPLE PROGRAM:

[1] Correlation of Two Sequences:

```
clc
close
clear
x=[2 5 0 4 5 9 4 8 6];
h=[2 1 3 5 6];
M=length(x);
N=length(h);
//disp(M,N);
y=zeros(N,M+N-1);
h1=mtlb_fliplr(h);
for i=1:N
   disp(i)
   y(i,i:i+M-1)=h1(i).*x
end
disp(y);
x_corr=zeros(1,M+N-1);
for j=1:M+N-1
```

```
   x_corr(j)=sum(y(:,j));
end
disp("The Correlation of Discrete Sequence :")
disp(x_corr);
```

OUTPUT:

The Correlation of Discrete Sequence:

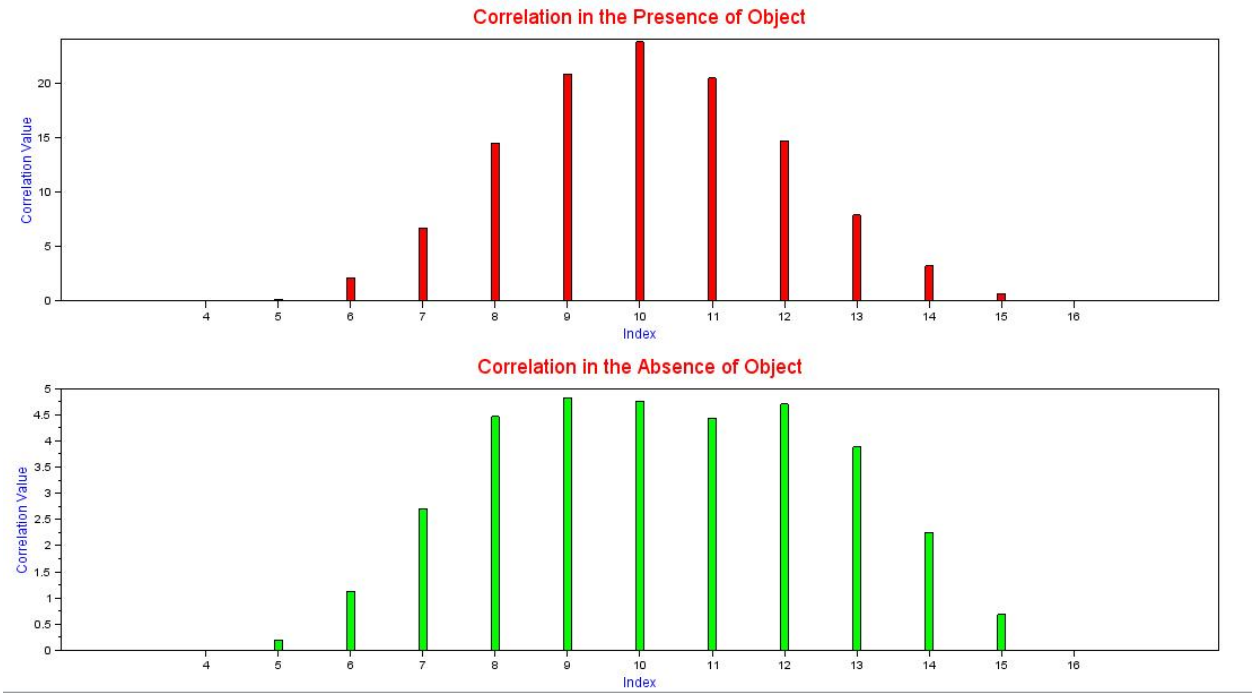
6. 17. 9. 22. 19. 13. 10.

[2] RADAR Signal Processing

```
clc;
clear;
xdel(winsid());
x=[0 1 2 3 2 1 0];          //Triangle pulse transmitted by radar
n=[-3 -2 -1 0 1 2 3];       // Index of Triangular Pulse
D=10;                       // Delay amount
nd=n+D;                     // Index of Delayed Signal
y=x;                        // Delayed Signal
scf();
subplot(2,1,1);
bar(n,x,0.1,'red');
title('Original Transmitted Signal','color','red','fontsize', 4);
xlabel("Index", "fontsize", 2,"color", "blue");
ylabel("Amplitude", "fontsize", 2, "color", "blue");
subplot(2,1,2);
bar(nd,y,0.1,'yellow');
title('Delayed Signal','color','red','fontsize', 4);
xlabel("Index", "fontsize", 2,"color", "blue");
ylabel("Amplitude", "fontsize", 2, "color", "blue");
w=rand(1,length(x));      // Noise Generation
nw=nd;
scf();
bar(nw,w,0.1,'red');
title('Noisy Signal','color','red','fontsize', 4);
xlabel("Index", "fontsize", 2,"color", "blue");
ylabel("Amplitude", "fontsize", 2, "color", "blue");
// If object is present we receive the signal R(n) = x(n-D) + Noise
R=y+w;       // Original Signal+Noise
nr=nw;       // Index of received signal at RADAR
nr_fold=mtlb_fliplr(nr);
R_fold=mtlb_fliplr(R);
nmin=min(n)+min(nr_fold);        // Lowest index of y(n)
nmax=max(n)+max(nr_fold);        // Highest index of y(n)
n_received=nmin:nmax;
Received_Presence=conv(x,R_fold); // Convolution of Original signal and Received Signal in
the Presence of Object (Equvalent to Correlation)//
scf();
subplot(2,1,1);
bar(n_received,Received_Presence,0.1,'red');
title('Correlation in the Presence of Object','color','red','fontsize', 4);
xlabel("Index", "fontsize", 2,"color", "blue");
```

```
ylabel("Correlation Value", "fontsize", 2, "color", "blue");
// If object is not present we receive the signal R(n) = Noise
R=w;                        // only Noise Signal
nr=nw;
nr_fold=mtlb_fliplr(nr);
R_fold=mtlb_fliplr(R);
nmin=min(n)+min(nr_fold);       // Lowest index of y(n)
nmax=max(n)+max(nr_fold);       // Highest index of y(n)
n_received=nmin:nmax;
Received_Absence=conv(x,R_fold); // Convolution of Original transmitted signal and
Received Signal in the Absence of Object  (Equvalent to Correlation)//
subplot(2,1,2);
bar(n_received,Received_Absence,0.1,'Green');
title('Correlation in the Absence of Object','color','red','fontsize', 4);
xlabel("Index", "fontsize", 2,"color", "blue");
ylabel("Correlation Value", "fontsize", 2, "color", "blue");
```

OUTPUT:

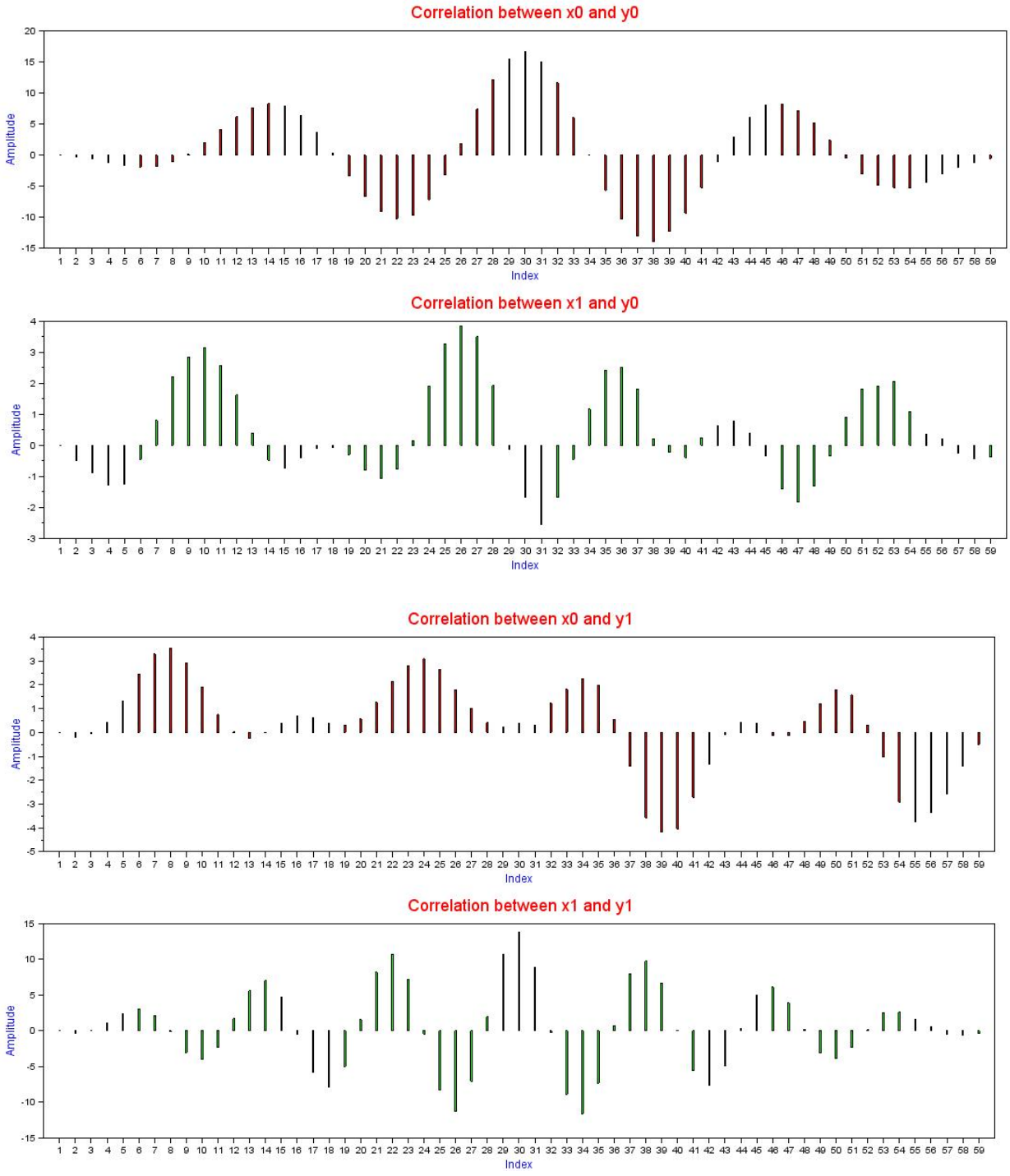


[3] Binary Symbol Detection at Receiver

```
clc;
clear;
xdel(winsid());
N=30;                   // Length of the signal
n=0:N-1;
pi=3.14;
x0=sin(n*pi/8);         // For '0' transmission
x1=sin(n*pi/4);         // For '1' transmission
scf();
subplot(2,1,1);
```

```
bar(n,x0,0.1,'red');
title('Sin(n*pi/8) to Represent Binary 0','color','red','fontsize', 4);
xlabel("Index", "fontsize", 2,"color", "blue");
ylabel("Amplitude", "fontsize", 2, "color", "blue");
subplot(2,1,2);
bar(n,x1,0.1,'yellow');
title('Sin(n*pi/4) to Represent Binary 1','color','red','fontsize', 4);
xlabel("Index", "fontsize", 2,"color", "blue");
ylabel("Amplitude", "fontsize", 2, "color", "blue");
w=rand(1,N);  // Noise Signal
y0=x0+w;      // Original Signal + Noise Signal
y1=x1+w;      // Original Signal + Noise Signal
// If received signal is y0(n)
rx0y0=xcorr(x0,y0);       // crosscorrelation between x0(n) and y0(n)
rx1y0=xcorr(x1,y0);       // crosscorrelation between x1(n) and y0(n)
scf();
subplot(2,1,1);
bar(rx0y0,0.1,'red');
title('Correlation between x0 and y0','color','red','fontsize', 4);
xlabel("Index", "fontsize", 2,"color", "blue");
ylabel("Amplitude", "fontsize", 2, "color", "blue");
subplot(2,1,2);
bar(rx1y0,0.1,'green');
title('Correlation between x1 and y0','color','red','fontsize', 4);
xlabel("Index", "fontsize", 2,"color", "blue");
ylabel("Amplitude", "fontsize", 2, "color", "blue");
// If received signal is y1(n)
rx0y1=xcorr(x0,y1);       // crosscorrelation between x0(n) and y1(n)
rx1y1=xcorr(x1,y1);       // crosscorrelation between x1(n) and y1(n)
scf();
subplot(2,1,1);
bar(rx0y1,0.1,'red');
title('Correlation between x0 and y1','color','red','fontsize', 4);
xlabel("Index", "fontsize", 2,"color", "blue");
ylabel("Amplitude", "fontsize", 2, "color", "blue");
subplot(2,1,2);
bar(rx1y1,0.1,'green');
title('Correlation between x1 and y1','color','red','fontsize', 4);
xlabel("Index", "fontsize", 2,"color", "blue");
ylabel("Amplitude", "fontsize", 2, "color", "blue");
```

OUTPUT:



MODIFICATION:

Modify the above Program to find cross correlation of two Pulse Amplitude Modulated signals.

CONCLUSION:

EXERCISE:

1. Show that correlation operation is not a linear operation.

EXPERIMENT - 4

APPLICATIONS OF DSP

OBJECTIVES:

- To generate DTMF signal for the given mobile number.
- To study the design of Echo Filter sound processing using scilab.
- To generate ECG wave for biomedical signal processing.
- Musical tone generation [sa re ga ma pa dh ni sa]

THEORY:

a. **DUAL-TONE MULTI-FREQUENCY SIGNALING** (**DTMF**):

DTMF is used for telecommunication signaling over analog telephone lines in the voice-frequency band between telephone handsets and other communications devices and the switching center. The version of DTMF that is used in push-button telephones for tone dialing is known as Touch-Tone.

AT&Ts Compatibility Bulletin described the product of DTFT as,
"A *method for pushbutton signaling from customer stations using the voice transmission path*."

Multi-frequency signaling is a group of signaling methods that use a mixture of two pure tone (pure sine wave) sounds. The earliest of these were for in-band signaling between switching centers, where long-distance telephone operators used a 12-digit keypad to input the next portion of the destination telephone number in order to contact the next downstream long-distance telephone operator. This semi-automated signaling and switching proved successful in both speed and cost effectiveness. *Dual-tone multi-frequency* (DTMF) signaling was developed for the consumer to signal their own telephone-call's destination telephone number instead of talking to a telephone operator.

USAGE:
DTMF tones were used at first in the telephone handsets and other communication devices to dial any numbers.
DTMF tones were used by cable television broadcasters to indicate the start and stop times of local commercial insertion points during station breaks for the benefit of cable companies.
DTMF tones are also used by some cable television networks and radio networks to signal the local cable company/network station to insert a local advertisement or station identification.
DTMF signaling tones can also be heard at the start or end of some VHS (Video Home System) cassette tapes. Information on the master version of the video tape is encoded in the DTMF tone.

Generating DTMF Tones:
The DTMF system uses seven different frequency signals transmitted in pairs to represent 12 different numbers, symbols and letters. The DTMF keypad is laid out in a 4×3 matrix, with each row representing a *low* frequency, and each column representing a *high* frequency.
Pressing a single key (such as '1') will send a sinusoidal tone for each of the two frequencies (697 and 1209 hertz (Hz)). The original keypads had levers inside, so each button activated two contacts. The multiple tones are the reason for calling the system multi frequency.

These tones are then decoded by the switching center to determine which key was pressed. These frequencies were chosen to prevent any harmonics from being incorrectly detected by the receiver as some other DTMF frequency. The frequencies allocated to the push-buttons of the telephone pad are shown below:

Table7.1 DTMF Row & Column Frequency

FREQUENCIES	1209 Hz	1336 Hz	1477 Hz
697 Hz	1	2	3
770 Hz	4	5	6
852 Hz	7	8	9
941 Hz	*	0	#

	1209 Hz	**1336 Hz**	**1477 Hz**
697 Hz	1	2	3
770 Hz	4	5	6
852 Hz	7	8	9
941 Hz	*	0	#

Fig.7.1 DTMF Key board

Special Tone Frequencies:

The tone frequencies, as defined by the Precise Tone Plan, are selected such that harmonics and intermodulation products will not cause an unreliable signal. No frequency is a multiple of another, the difference between any two frequencies does not equal any of the frequencies, and the sum of any two frequencies does not equal any of the frequencies. The frequencies were initially designed with a ratio of 21/19, which is slightly less than a whole tone. The frequencies may not vary more than ±1.8% from their nominal frequency, or the switching center will ignore the signal. The high frequencies may be the same volume as – or louder than – the low frequencies when sent across the line. The loudness difference between the high and low frequencies can be as large as 3 decibels (dB) and is referred to as "twist."

Each DTMF can be represented as: $y(t) = A (\sin(\omega_1 t) + \sin(\omega_2 t))$

Where ω_1 and ω_2 are the 2 frequencies that are to be added.

SAMPLE PROGRAM: Generation of DTMF signal for given Mobile no.

```
clc;
close;
clear;
row_f1=[700 770 850 941];            // Row Frequency
colum_f1=[1220 1350 1490];           // Column Frequency
fs=8000;                             // Sampling Frequency
N=1:4000;                            // Total No. of Sample
mobile=[9 9 7 8 3 7 4 2 5 3];        // Mobile Number
temp=[];                             // Array that Contain total signal for each Digit
```

```
figure;
for i=1:length(mobile)
   select mobile(i)
   case 1
      row_f=1;
      colum_f=1;
   case 2
      row_f=1;
      colum_f=2;
   case 3
      row_f=1;
      colum_f=3;
   case 4
      row_f=2;
      colum_f=1;
   case 5
      row_f=2;
      colum_f=2;
   case 6
      row_f=2;
      colum_f=3;
    case 7
      row_f=3;
      colum_f=1;
   case 8
      row_f=3;
      colum_f=2;
   case 9
      row_f=3;
      colum_f=3;
    case 0
      row_f=4;
      colum_f=2;
   else
      row_f=4;
      colum_f=1;
   end
    y=sin(2*3.14*(row_f1(row_f)/fs)*N)+sin(2*3.14*(colum_f1(colum_f)/fs)*N);
    temp=[temp y zeros(1,4000)]; // Append the Signal + zeros After each Number
   end
   plot(temp);
   sound(temp,fs);
```

OUTPUT:

MODIFICATION:

1. Decode mobile no. from DTMF Signal.

b. To design Echo Generation Filter:

Sound Processing:

Digital signal processing (DSP) technology and its advancements have dramatically impacted our modern society everywhere. Without DSP, we would not have digital/Internet audio or video; digital recording; CD, DVD, and MP3 players; digital cameras; digital and cellular telephones; digital satellite and TV; or wire and wireless networks. DSP's application in field of sound and biomedical is discussed below.

The digital signal processing of audio signals often involves digital filter to create various special effects such as echo and reverb for compensation for the acoustics of a listening environment, which are the most important for modern-day studios processing audio signals. In listening space, what we hear from an audio source consists not only of direct sound, but also early echoes reflected directly from the walls and other structures. The direct sound provides clues to the location of the source, the early echoes provide an indication of the physical size of the listening space. The amplitude of the echoes decays exponentially with time. An echo filter has the form of equation given by: $Y[n] = X[n] + \alpha * X[n - D]$

This describes FIR filter whose output Y[n] equals to the input X[n] and its delayed (by D samples) and attenuated (by α) replica of X[n] (the echo term). This filter is also called a **Comb filter**.

SAMPLE PROGRAM 1: Echo Generation

```
b = [1 zeros(1,7999) 0.5];      % numerator coefficient correspond to 1 sec delay and 50%
                                attenuation
a = [1];                        %denominator coefficient
[x,Fs] = wavread('e.wav')       %read speech file
Y =filter(b,a,x);               %pass speech file through filter
wavwrite( Y,'ringecho1.wav')    % generate another file with delay
```

Filter function:

The filter function filters a data sequence using a digital filter which works for both real and complex inputs. The filter is a direct form II transposed implementation of the standard difference equation. **y = filter(b,a,x)** filters the data in vector 'x' with the filter described by numerator coefficient vector b and denominator coefficient vector a. Here from Direct form II representation of vector '**b**' we get the numerator in z-transform as

$$N(z) = 1 + 0.5z^{-8000}$$

This shows the delay of 1 second with the power of 'z' that is 8000 and the attenuation is given by its co-efficient 0.5 showing 50% attenuation.

$$D(z) = 1$$

Here vector '**a**' is denominator co-efficient, which is always 1, otherwise it is made 1 by the filter normalizes its co-efficient to 1. If '**a**' is 0 then the filter function shows an error.

Wavread function**:**[y, Fs] = wavread(filename) loads a WAVE file specified by 'Filename', returning the sampled datain y.

Wavwrite function**:** wavwrite(y,filename) writes the data stored in the variable y to a WAVE file called filename. The filename input is a string enclosed in single quotes. The data has a sample rate of 8000 Hz and is assumed to be 16-bit.

Rate conversion of sound files: DSP can be useful in the field of sound processing. Various sound effects can be created using Scilab. Scilab can read and write sound files in the .wav format. In many areas of digital signal processing (DSP) applications—such as communications, speech, and audio processing—rising or lowering of a sampling rate is required. Speed of the audio signal can be altered with the help of sampling rate. The speed of a sound signal is increased by down sampling and that reduced by up-sampling the signal.

Down sampling is the process of reducing the sampling rate of a signal. This is usually done to reduce the data rate or the size of the data. The down sampling factor is usually an integer or a rational fraction greater than unity. This factor multiplies the sampling time or, equivalently, divides the sampling rate. For example, if an audio signal at 44,100 Hz is down sampled to 22,050 Hz , the bit rate is reduced in half, from 1,411,200 bit/s to 705,600 bit/s. The audio was therefore down sampled by a factor of 2.

The following code illustrates the down sampling of an audio signal. Here, the speed of audio signal increases.

```
%Down sampling ("speed up")
%clear all;
[x, Fs] = wavread('e.wav');
y = intdec(x,2);
sound(y,Fs)
wavwrite(y,44100,'downsamp_ee.wav')
```

Up-sampling is the process of increasing the sampling rate of a signal. It is just opposite to down sampling. Here, the up-sampling factor multiplies the sampling rate or, equivalently, divides the sampling period. Thus speed of an audio signal can be reduced by increasing the sampling rate. Up-sampling is illustrated in the following Scilab code.

```
% Upsampling ("slow down")
[x, Fs] = wavread('e.wav');
z = intdec(x,0.5);
sound(z,44100)
wavwrite(z,44100,'upsamp.wav')
```

Filtering of sound files:

In sound processing, the audio signal has to be controlled for various frequency components present. The frequency response of a sound system is corrected so that the frequency balance of the music as heard through speakers, better matches the original performance picked up by a microphone. This is done by use of low-cut (high pass) or high-cut (low pass) filter.

A low-pass filter is an electronic filter that passes low-frequency signals but attenuates (reduces the amplitude of) signals with frequencies higher than the cutoff frequency. The following Scilab code is used to attenuate the high frequency components of the desired audio file. Here, a fourth order low pass elliptic filter is designed with a normalized pass band edge frequency 0.15, 5 dB of ripple in pass band and 80 dB of attenuation in the stop band.

SAMPLE PROGRAM :

Lowpass Filtering

```
N = 4;Rp = 5;Rs = 80;
Wn = .15;
[x, Fs] = wavread('e.wav');
[B,A] = ellip(N,Rp,Rs,Wn);
yl= filter(B,A,x);
sound(yl,44100)
wavwrite(yl,44100,'lpf.wav')
```

OUTPUT:

A high pass filter is just opposite to a low pass filter. A high pass filter, also called a low-cut filter, or bass-cut filter when used in audio application, passes high frequency signals but attenuates signals with lower frequency. For elimination of low frequency sounds the following code is demonstrated in which a high pass fourth order elliptic filter is designed with a normalized pass band edge frequency 0.15, 5 dB of ripple in pass band and 80 dB of attenuation in the stop band.

MODIFICATION:

1. Modify above code to perform Highpass Filtering.

OUTPUT:

c. Generation of ECG signal (Use of DSP in biomedical signal processing):

Medical instruments would be less efficient or unable to provide useful information for precise diagnoses if there were no digital electrocardiography (ECG) analyzers or digital x-rays and medical image systems.
Hum noise created by poor power supplies, transformers, or electromagnetic interference sourced by a main power supply is characterized by a frequency of 50 Hz and its harmonics. If this noise interferes with a desired audio or biomedical signal (e.g., in electrocardiography [ECG]), the desired signal could be corrupted. The corrupted signal is useless without signal processing. It is sufficient to eliminate the 50-Hz hum frequency with its second and third harmonics in most practical applications.

SAMPLE PROGRAM:

Generation of ECG signals.

```
clc;
clear all;
close all;
cycle=zeros(1,500);
y=[01 2 3 4 5 6 7 8 9 10 9 8 7 6 5 4 3 2 1 0]/4;   % Triangle pulse segment
t=[0:1:40];                                         % Local time axis
a=(.05*(t-20).*(t-20)-20)/50;                       % Parabolic recovery segment
cycle(1:61)=2*[y a];                                % Place pulse in 1sec. cycle
cyc=filter([1 1 1 1 1], [1], cycle);                % Smooth the corners
x=[cyc cyc cyc cyc cyc];                            % Repetition used for 5 cycle of trace
[idum, nsize]=size(x);
t=[0:1:nsize-1]/500;                                % Sampling frequency 500Hz
plot(t,x);
xlabel('Time(sec)');
ylabel('Amplitude');
title('ECG signal');
```

OUTPUT:

The ECG signals interfered by some noise signals, can be generated using Scilab. The 'randn' function of Scilab can be used to add noise to the signal .the following code demonstrates the same.

MODIFICATION:

1. Generation of Noisy ECG signals.

d. Musical tone generation [sa re ga ma pa dh ni sa] with each tone has time duration 0.5 sec**.**

SAMPLE PROGRAM:

```
clc;
close;
clear;
frequency=[240 254 302 320 358.5 380 451 470];  // Corresponding Frequency
fs=8000;                          // Sampling Frequency
no=8;
N=1:4000;                          // Total No. of Samples for Each tone


temp=[];
for i=1:no
   y=sin(2*3.14*(frequency(i)/fs)*N);
   temp=[temp y];
end
length(temp);
sound(temp,fs);
```

OUTPUT:

CONCLUSION:

EXERCISE:

1. Design moving average filter.

EXPERIMENT – 5

DISCRETE FOURIER TRANSFORM

OBJECTIVES:

- To plot the spectrum of a given DT signal using DFT.
- To verify different properties of DFT.

THEORY:

Fourier Transform is used to find spectrum of analog signals. If the signal is Discrete Time, Discrete Time Fourier Transform (DTFT) is applicable. DTFT gives the spectrum which is continuous in frequency domain. To process the discrete time signals in frequency domain using processors, we need its discrete version. This can be obtained by another transform known is Discrete Fourier Transform (DFT). DFT computes N equally spaced frequency samples of the DTFT. The inverse transform is known as IDFT and the transform pair is given as x(n)↔X(k). Both the indices n and K are ranging from 0 to N-1.The integer *n* is known as time index since it denotes the time instant. The integer *k* denotes discrete frequency and is called frequency index. The expressions are as follows.

DFT: $X(k) = \sum_{n=0}^{N-1} \mathrm{x(n)}\, \mathrm{e}^{\frac{-\mathrm{j}2\pi \mathrm{kn}}{\mathrm{N}}}, k = 0,1...N-1$

IDFT: $x(n) = \sum_{n=0}^{N-1} \mathrm{X(k)}\, \mathrm{e}^{\frac{-\mathrm{j}2\pi \mathrm{kn}}{\mathrm{N}}}, n = 0,1...N-1$

Algorithms which reduce computations while evaluating above equations are known as Fast Fourier Transform (FFT) algorithms.

SAMPLE PROGRAM:

DFT using inbuilt function

```
clc;
x=[1 2 3 4];
y=fft(x);
stem(abs(y));
title('dft');
xlabel('time');
ylabel('amplitude');
```

SAMPLE PROGRAM:

Spectrum of the exponential function

```
clc;
close;
clear;
```

```
fs=1; //sampling rate
n=0:1/fs:10 //10s duration (Increasing the value and observe spectrum)
x=exp(-n);
plot2d3(x)
//x1=[x, zeros(1,100)]; //zero padding for better spectrum resolution
y=fft(x1);
plot2d3(abs(fftshift(y)
```

MODIFICATIONS:

1. Give circular shift to x by 2 and obtain DFT.
2. Find IDFT of y using inbuilt function.
3. Find DFT of signal for duration of 100s. Give your observations by comparing with the previous results of 10s
4. Find DFT of signal for same duration of 10s but increase the sampling frequency. Give your observations by comparing with the previous results of fs=1Hz

Verify circular convolution property of DFT for

x=[1 1 1 1 0 0 0 0] and h=[0.5 0.3 0.5].

A pulse signal of 1ms duration and 50% duty cycle is sampled at 8KHz. Plot the spectrum with required N-point DFT. Increase the value of N and plot the spectrum. Increase sampling rate to 16, 32 and 64 KHz and plot the spectrum. Write down your observations in each case.

CONCLUSION:

EXERCISE:

1. Write Scilab code to find DFT without using inbuilt function fft.
2. In above programs use variables which count number of multiplications and additions while determining fft/ifft and comment on it.

EXPERIMENT – 6

INTRODUCTION TO CODE COMPOSER STUDIO

OBJECTIVES:

- To create a simple project using C file.
- Implement simple programs such as Linear Convolution and DFT in CCS
- To study TI6x CPU architecture through help.

METHOD:

- To Study procedure to create and run project in CCS refer the steps in the Annexure.

- To study TI6x CPU architecture through help.

  Get help on the following CPU architectural features.

  i. CPU registers
  ii. Data cross path
  iii. Addressing modes
  iv. MV instruction
  v. Address bus and data bus

Program Statement:

Write a program to display a message “Hello” using .c program.

Sample Program 1:

Create a project and Write a program in CCS to print “Hello”

```
#include<stdio.h>
main()
{
printf("Hello");
}
```

OUTPUT:

Sample Program 2: Create a project and Write a program in CCS to print Factorial.

```
#include<stdio.h>
int main()
{
int n,fac,i;
printf("enter n:");
scanf("%d",&n);
fac=1;
for(i=n;i>1;i--)
{
fac=fac*n;
n--;
}
printf("%d",fac);
return 0;
```

```
}
```

CONCLUSION:

EXPERIMENT – 7

CONVOLUTION & DFT IN CCS

Objective: To implement and execute convolution and DFT programs in CCS

Convolution :

Sample program

```
#include<stdio.h>
int main(void) {
int a[5]={1,2,3,4,5};
int b[3]={1,2,3};
int ans[7]={0,0,0,0,0,0};
int i,j;
for(i=0;i<5;i++){
for(j=0;j<3;j++){
ans[i+j]=ans[i+j]+(a[i]*b[j]);
}
}
for(i=0;i<7;i++){
printf("%d\n",ans[i]);
}
return 0;
}
```

Discrete Fourier Transform (DFT)

Sample program:

```
#include<stdio.h>
#include<math.h>
void main(){
int a[4]={1,3,2,5};
float ans[8]={0,0,0,0,0,0,0,0};
int N=4,k,k1,i;
float w;
for(k=0;k<2*N;k=k+2){
for(i=0;i<N;i++){
k1=k/2;
w=2*3.14*k1*i/N;
ans[k] += a[i]*cos(-w);
ans[k+1] += a[i]*sin(-w);
}
}
for(i=0;i<(2*N);i=i+2){
printf(" %f + j%f \n",ans[i],ans[i+1]);
}
}
```

CONCLUSION:

EXPERIMENT – 8

SINEWAVE GENERATION IN CCS

Objective:

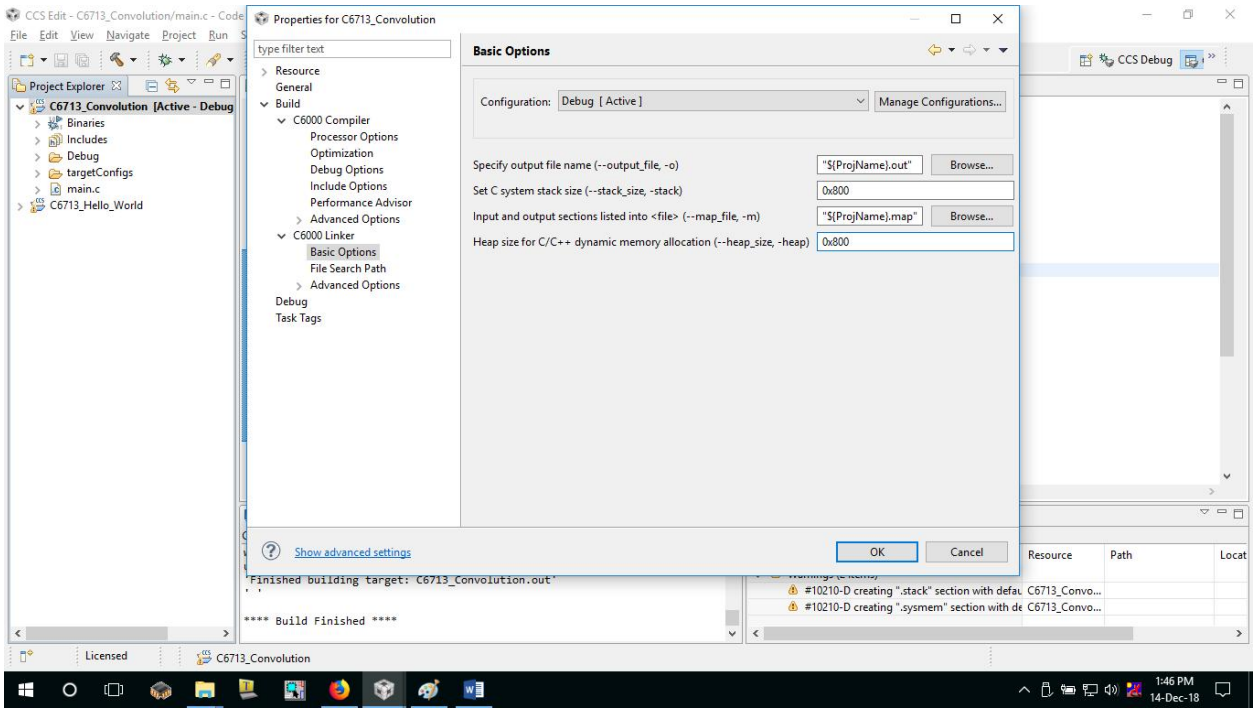
- Write a program for “plot sine wave in graph window” and execute on EPB_C6713 target board

Hardware Part List:

- PC
- Code Composer Studio v5.3
- +5v DC Power supply supplied with Kit
- EPB_C6713
- USB A to B Cable

Steps for creating new project:

Then open “project property” by “*right click-> properties*” and at C6000 Linker set the stack size as shown below in screenshot image.

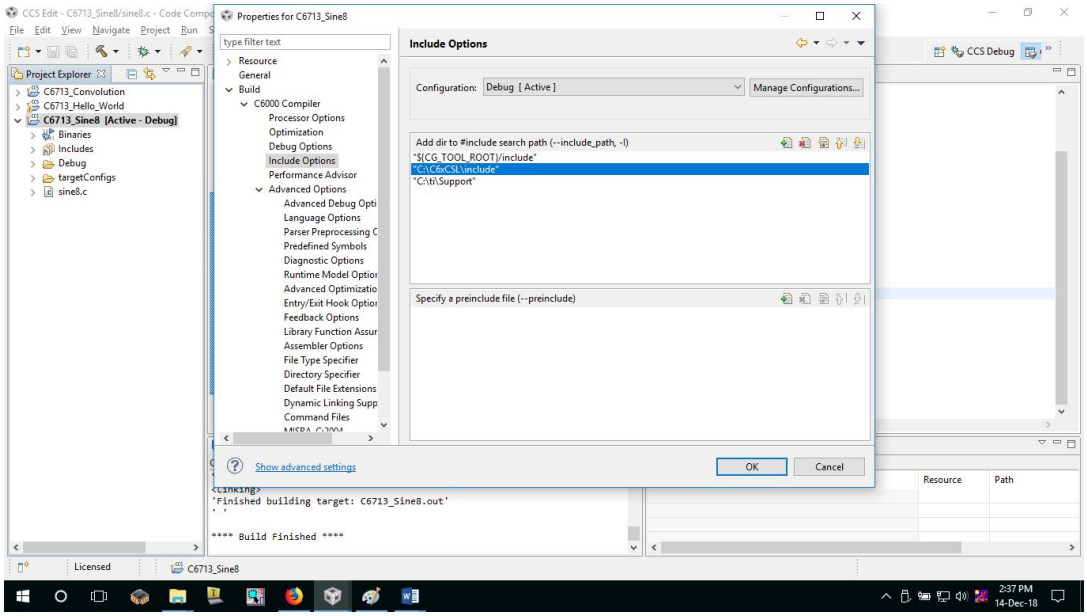


Then open “project property” by “*right click-> properties*” and at “**C6000 Compiler->include Option** “. Add the header file path as shown here.

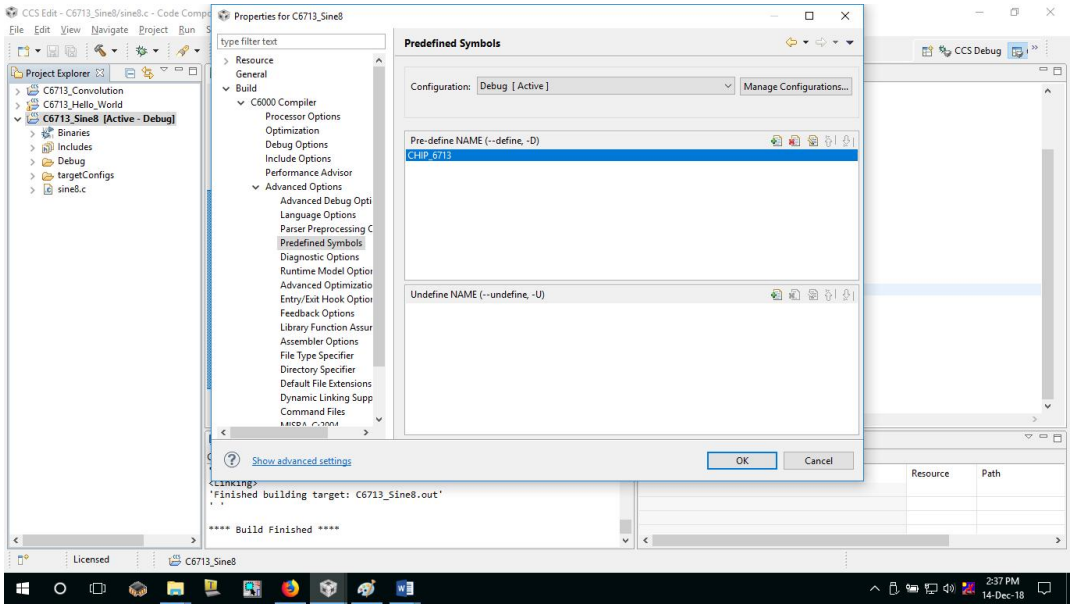
Files:

C:\C6xCSL\include

C:\ti\Support



Then open “project property” by “*right click-> properties*” and at “**C6000 Compiler->Predefined Symbol**“. Add the pre-defined name CHIP-6713 as shown.



METHOD:

SAMPLE PROGRAM:

//sine8 Sine generation. Output buffer plotted within CCS

```
#include <dsk6713_aic23.h>
        //support file for codec,DSK
Uint32 fs=DSK6713_AIC23_FREQ_16KHZ;
        //set sampling rate
int loop = 0;
        //table index
short gain = 10;
        //gain factor
short square_table[8]={-1000,-1000,1000,1000,-1000,-1000,1000,1000};//square
values
short sq_table[8]={1000,1000,200,200,1000,1000,200,200};//square values
short ramp_table[8]={0,500,1000,500,0,-500,-1000,-500};      //sine values
short sine_table[8]={0,707,1000,707,0,-707,-1000,-707};      //sine values
short out_buffer[256];                                       //output buffer
const short BUFFERLENGTH = 256;                                    //size of output
buffer
int i = 0;                                                   //for buffer count
int j;

void main()
{
        while(1)
        {                                       //infinite loop
                for(j=0;j<8;j++)
                {
                        /* Break Point*/
                        //out_buffer[i] = sq_table[j]*gain;            //output to buffer
                        out_buffer[i] = (gain*sine_table[loop])/1000;               //output to
buffer
                        i++;
        //increment buffer count
                        if(i==BUFFERLENGTH) i=0;        //if @ bottom reinit count
                        if (loop < 7) ++loop;       //check for end of table
                        else loop = 0;              //reinit table index
                }
        }                                                                   //init DSK, codec,
McBSP

}
```

CONCLUSION:

EXERCISE:

1. Create a project and Write a program in CCS to display half wave sinewave.

EXPERIMENT - 9

FIR FILTER DESIGN USING WINDOWING METHOD

OBJECTIVES:

- To design FIR Low-pass & Band-pass filter using rectangular window.

THEORY:

A digital Finite Impulse Response (FIR) filter is a type of digital filter used in digital signal processing. In an FIR filter, the filter's output settles down to zero after a finite number of samples. Some key characteristics and properties of digital FIR filters are Linear Phase (a desirable property in applications such as audio and image processing), Stability, No feedback, Design flexibility etc.

One of the method of to design FIR filter is using window function.

Design steps for Linear Phase FIR Filter (Fourier Series method):

1 Based on the desired frequency response specification $H_d(e^{j\omega})$ determine the corresponding unit sample response $h_d(n)$ is determined using the following relation

$$h_d(n) = \frac{1}{2\pi}\int_{-\pi}^{\pi} H_d(w)e^{jwn}\, dw$$

Where $H_d(w) = \sum_{-\infty}^{\infty} h_d\,(n)e^{-j\omega n}$

This truncation of $h_d(n)$ to length M-1 is same as multiplying $h_d(n)$ by the rectangular window defined as

$$w(n) = \begin{cases} 1 & 0 \le n \le M-1 \\ 0 & otherwise \end{cases}$$

Thus the unit sample response of the FIR filter becomes

$$\begin{aligned} h(n) &= h_d(n)\ w(n) \\ &= h_d(n) \qquad 0 \le n \le M\text{-}1 \\ &= 0 \qquad \text{otherwise} \end{aligned}$$

Name of window function $w(n)$	Mathematical definition
Rectangular	1
Hanning	$0.5-0.5\cos\left[\frac{2\pi n}{N-1}\right]$
Hamming	$0.54-0.46\cos\left[\frac{2\pi n}{N-1}\right]$
Blackman	$0.42-0.5\cos\left[\frac{2\pi n}{N-1}\right]+0.08\cos\left[\frac{2\pi n}{N-1}\right]$
Kaiser	$\frac{I_0\left[\beta\sqrt{1-\left(\frac{\lvert 2n-N+1\rvert}{N-1}\right)^2}\right]}{-I_o(\beta)}$ Where, $I_0(x)=\sum_{k=0}^{\infty}\left(\frac{x^k}{2^k k!}\right)^2$

SAMPLE PROGRAM:

1) Low-pass filter

```
//Caption : Program to Design FIR Low Pass Filter
clc ;
close ;
M = input ('Enter the odd Fileter length = ' );
// F i l t e r l e n g t h
Wc = input ('Enter the Digital Cutoff frequency = ');
// D i g i t a l C u t o f f f r e q u e n c y
Tuo = (M -1) /2 // C ent er Value
for n = 1:M
   if (n == Tuo +1)
      hd(n) = Wc/ %pi ;
   else
      hd(n) = sin(Wc *((n -1) -Tuo)) /(((n -1) -Tuo )*%pi);
   end
end
// R e c t an gul a r Window
for n = 1:M
   W(n) = 1;
end
// Windowing Fitler Coefficients
h = hd .*W;
disp (h, 'Filter Coefficientsare')
[hzm ,fr ]= frmag (h ,256) ;
hzm_dB = 20* log10 (hzm)./ max ( hzm );
subplot (2 ,1 ,1)
plot (2*fr , hzm )
xlabel('Normalized Digital Frequency W');
ylabel('Magnitude');
title ('Frequen cy Response of FIR LPF using Rectangular window')
xgrid (1)
subplot (2 ,1 ,2)
```

```
plot (2*fr , hzm_dB )
xlabel ('Normalized Digital Frequency W');
ylabel ('Magnitude in dB');
title ('Frequency Response of FIR LPF using Rectangular window')
xgrid (1)
```

2) Band-pass filter

```
// Caption : Program to Design FIR Band Pass Filter
clear ;
clc ;
close ;
M = input('Ent er the Odd Filter Length = ');
// Filterlength
// Digital Cut off frequency [ Lower Cut off , Upper Cut off]= [ %pi/4 , 3*%pi/ 4]
Wc = input('Enter the Digital Cut off freqency (in tuple) = ');
Wc2 = Wc(2)
Wc1 = Wc(1)
Tuo = (M -1) /2 // Center Value
hd = zeros (1,M);
W = zeros (1,M);
for n = 1:M
   if (n == Tuo +1)
      hd(n) = (Wc2 - Wc1 )/ %pi ;
   else
      n
      hd(n) = (sin(Wc2 *((n -1) -Tuo))-sin(Wc1*((n -1) -Tuo ))) /(((n -1) -Tuo )* %pi );
   end
   if(abs(hd(n)) <(0.00001) )
      hd(n) =0;
   end
end
hd;
// Rectangular Window
for n = 1:M
   W(n) = 1;
end
// Windowing Filter Coefficients
h = hd .*W;
disp (h, 'Filter Coefficients are')
[hzm ,fr ]= frmag (h ,256) ;
hzm_dB = 20* log10 (hzm)./ max ( hzm );
subplot (2 ,1 ,1)
plot (2*fr , hzm )
xlabel('Normalized Digital Frequency W');
ylabel('Magnitude');
title('Freq. Response of FIR BPF using Rectangular window')
xgrid (1)
subplot (2 ,1 ,2)
```

```
plot (2*fr , hzm_dB )
xlabel('Normalized Digital Frequency W');
ylabel ('Magnitude in dB');
title ('Frequency Response of FIR BPF using Rectangular window')
xgrid (1)
```

MODIFICATIONS:

1) Modify the above program to design low pass filter with Hann window.

EXERCISES:

1) Design High – Pass filter with rectangular window.

EXPERIMENT - 10

IIR FILTER DESIGN

OBJECTIVES:

1) To design Digital IIR Butterworth lowpass filter using bilinear transformation.
2) To design IIR High-pass filter using inbuilt function.

THEORY:

Digital IIR (Infinite Impulse Response) filters are a type of digital filter commonly used in signal processing applications. Unlike Finite Impulse Response (FIR) filters, IIR filters have feedback loops in their structure. There are many classes of analogue low-pass filter, such as the Butterworth, Chebyshev and Elliptic filters. The classes differ in their nature of their magnitude and phase responses. The design of analogue filters other than low-pass is based on frequency transformations, which produce an equivalent high-pass, band-pass, or band-stop filter from a prototype low-pass filter of the same class. The analogue IIR filter is then converted into a similar digital filter using a relevant transformation method. There are three main methods of transformation, the impulse invariant method, the backward difference method, and the bilinear z-transform.

Time domain representation of IIR filter.

$$y[n] = \sum_{l=1}^{N} a_l y[n-l] + \sum_{k=0}^{M} b_k x[n-k] .$$

Transfer function and Frequency response of IIR filter described as follows.

$$H(z) = \frac{\sum_{k=0}^{M-1} b_k z^{-k}}{1 + \sum_{k=1}^{N-1} a_k z^{-k}} \xrightarrow{z=e^{j\Omega}} H(\Omega) = \frac{\sum_{k=0}^{M-1} b_k e^{-j\Omega k}}{1 + \sum_{k=1}^{N-1} a_k e^{-j\Omega k}}$$

SAMPLE PROGRAM

Objective 1: To design Digital IIR Butterworth lowpass filter using bilinear transformation.

```
clc ;
```

```
clear ;
close ;
fp= input ('Enter the pass band edge (Hz) = ' );
fs= input ('Enter the stop band edge (Hz) =');
kp= input ('Enter the pass band attenuation(dB) = ');
ks= input ('Enter the stop band attenuation(dB) = ');
Fs= input ('Enter the sampling rate  samples/sec =');
d1 = 10^(kp/20) ;
d2 = 10^(ks/20) ;
d = sqrt ((1/(d2^2)) -1);
E = sqrt ((1/(d1^2)) -1);
// Digital filter specifications(rad/samples)
wp =2* %pi *fp *1/ Fs;
ws =2* %pi *fs *1/ Fs;
disp (wp ,'Digital Pass band edge freq in rad/samples wp= ')
disp (ws ,'Digital Stop band edge freq in rad/samples rws= ')
// Pre warping
op =2* Fs* tan (wp /2);
os =2* Fs* tan (ws /2);
disp (op ,' Analog Pass Band Edge Freq . in rad/sec op=')
disp (os ,'Analog Stop band Edge Freq . in rad/sec os= ')
N = log10(d/E)/ log10(os/op);
disp(N)
oc = op /((E**2)**(1/(2*N)));
N = ceil(N); // rounded to n e a r e s t i n t e g e r
disp (N,'IIR Filterorder N = ');
disp (oc ,'Cutoff Freuency in rad/seconds OC = ')
[pols ,gn] = zpbutt (N,oc);
disp (gn ,'Gain o f Analog IIR Butt erw orth LPF Gain =')
disp (pols ,'Polesof Analog IIR Butterworth LPF Poles = ')
HS = poly (gn ,'s' , 'coeff')/ real(poly(pols ,'s'));
disp (HS ,'Transfer function of Ananlog IIR Butterworth LPF H(S)= ')
z = poly (0, 'z')
Hz = horner (HS ,(2*Fs*(z -1)/(z +1) ))
num = coeff (Hz(2))
den = coeff (Hz(3))
Hz(2)= Hz(2)./den(3);
Hz(3) = Hz(3)./den(3) ;
disp (Hz ,'Transfer function of Digitl a IIR Butt erworth LPF H(Z)= ')
[Hw ,w] = frmag (Hz ,256) ;
figure (1)
plot (2*w*%pi ,20* log10 ( abs (Hw)));
xlabel ('Digital Frequency w−−−>' )
ylabel ('Magnitude in dB 20log|H(w)|=')
title ('Magnitude Response of IIR LPF')
xgrid (1)
```

```
//Example data
//Enter the pass band edge (Hz) = 1500
//Enter the stop band edge (Hz) =2000
//Enter the pass band attenuation(dB) = -1
//Enter the stop band attenuation(dB) = -10
//Enter the sampling rate  samples/sec =8000
```

Objective 2: Design Highpass filter using inbuilt function

```
clc ;
clear ;
close ;

fc= input ('enter the cutoff frequency fc=');
fs= input ('enter the sampling frequency fs=');
N= input ('enter the order of the filter N=');
fp =2* fc/fs;
//genrating Highpass Butterworth IIR filter

[Hz2]= iir(N, ' hp ' , ' butt ' ,[fp /2 ,0] ,[0 ,0]) ;
[Hw2 ,w2 ]= frmag (Hz2 ,256) ;
subplot (2 ,2 ,1);
plot2d3 (w2 , abs ( Hw2 ));
xlabel ('frequency');
ylabel ('magnitude');
title ('butterworth high pass IIR filter');

//Use the above program
//enter the cutoff frequency fc =1000

//enter the sampling frequency fs =10000
//enter the order of the filter N=3
```

MODIFICATIONS:

1. Modify the above program to design high pass filter with Hann window.

EXERCISES:

1. Design lowpass, bandpass & bandstop filters using inbuilt function.

DATA SHEET

TMS320C6713B
FLOATING-POINT DIGITAL SIGNAL PROCESSOR

SPRS294B – OCTOBER 2005 – REVISED JUNE 2006

- Highest-Performance Floating-Point Digital Signal Processor (DSP): TMS320C6713B
  - Eight 32-Bit Instructions/Cycle
  - 32/64-Bit Data Word
  - 300-, 225-, 200-MHz (GDP and ZDP), and 225-, 200-, 167-MHz (PYP) Clock Rates
  - 3.3-, 4.4-, 5-, 6-Instruction Cycle Times
  - 2400/1800, 1800/1350, 1600/1200, and 1336/1000 MIPS/MFLOPS
  - Rich Peripheral Set, Optimized for Audio
  - Highly Optimized C/C++ Compiler
  - Extended Temperature Devices Available
- Advanced Very Long Instruction Word (VLIW) TMS320C67x™ DSP Core
  - Eight Independent Functional Units:
    - 2 ALUs (Fixed-Point)
    - 4 ALUs (Floating-/Fixed-Point)
    - 2 Multipliers (Floating-/Fixed-Point)
  - Load-Store Architecture With 32 32-Bit General-Purpose Registers
  - Instruction Packing Reduces Code Size
  - All Instructions Conditional
- Instruction Set Features
  - Native Instructions for IEEE 754
    - Single- and Double-Precision
  - Byte-Addressable (8-, 16-, 32-Bit Data)
  - 8-Bit Overflow Protection
  - Saturation; Bit-Field Extract, Set, Clear; Bit-Counting; Normalization
- L1/L2 Memory Architecture
  - 4K-Byte L1P Program Cache (Direct-Mapped)
  - 4K-Byte L1D Data Cache (2-Way)
  - 256K-Byte L2 Memory Total: 64K-Byte L2 Unified Cache/Mapped RAM, and 192K-Byte Additional L2 Mapped RAM
- Device Configuration
  - Boot Mode: HPI, 8-, 16-, 32-Bit ROM Boot
  - Endianness: Little Endian, Big Endian
- 32-Bit External Memory Interface (EMIF)
  - Glueless Interface to SRAM, EPROM, Flash, SBSRAM, and SDRAM
  - 512M-Byte Total Addressable External Memory Space
- Enhanced Direct-Memory-Access (EDMA) Controller (16 Independent Channels)
- 16-Bit Host-Port Interface (HPI)
- Two McASPs
  - Two Independent Clock Zones Each (1 TX and 1 RX)
  - Eight Serial Data Pins Per Port: Individually Assignable to any of the Clock Zones
  - Each Clock Zone Includes:
    - Programmable Clock Generator
    - Programmable Frame Sync Generator
    - TDM Streams From 2-32 Time Slots
    - Support for Slot Size: 8, 12, 16, 20, 24, 28, 32 Bits
    - Data Formatter for Bit Manipulation
  - Wide Variety of I2S and Similar Bit Stream Formats
  - Integrated Digital Audio Interface Transmitter (DIT) Supports:
    - S/PDIF, IEC60958-1, AES-3, CP-430 Formats
    - Up to 16 transmit pins
    - Enhanced Channel Status/User Data
  - Extensive Error Checking and Recovery
- Two Inter-Integrated Circuit Bus (I^2C Bus™) Multi-Master and Slave Interfaces
- Two Multichannel Buffered Serial Ports:
  - Serial-Peripheral-Interface (SPI)
  - High-Speed TDM Interface
  - AC97 Interface
- Two 32-Bit General-Purpose Timers
- Dedicated GPIO Module With 16 pins (External Interrupt Capable)
- Flexible Phase-Locked-Loop (PLL) Based Clock Generator Module
- IEEE-1149.1 (JTAG†) Boundary-Scan-Compatible
- 208-Pin PowerPAD™ PQFP (PYP)
- 272-BGA Packages (GDP and ZDP)
- 0.13-µm/6-Level Copper Metal Process
  - CMOS Technology
- 3.3-V I/Os, 1.2‡-V Internal (GDP/ZDP/ PYP)
- 3.3-V I/Os, 1.4-V Internal (GDP/ZDP) [300 MHz]

TMS320C67x and PowerPAD are trademarks of Texas Instruments.
I^2C Bus is a trademark of Philips Electronics N.V. Corporation
All trademarks are the property of their respective owners.
† IEEE Standard 1149.1-1990 Standard-Test-Access Port and Boundary Scan Architecture.
‡ These values are compatible with existing 1.26-V designs.

TEXAS INSTRUMENTS
POST OFFICE BOX 1443 • HOUSTON, TEXAS 77251–1443

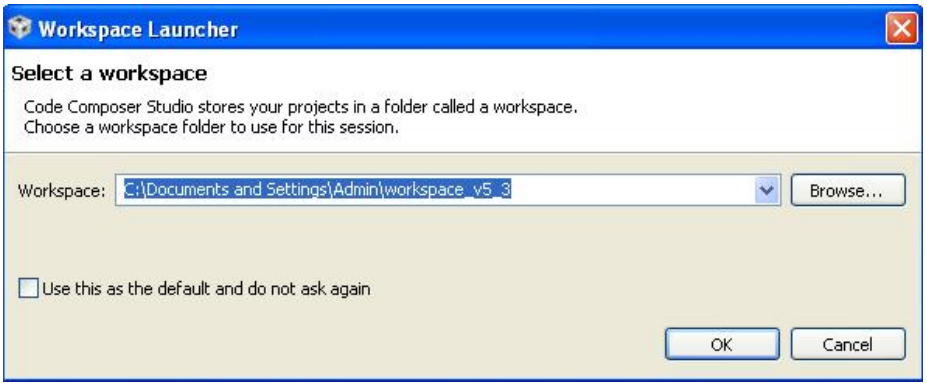
APPENDIX

Steps for creating new project:

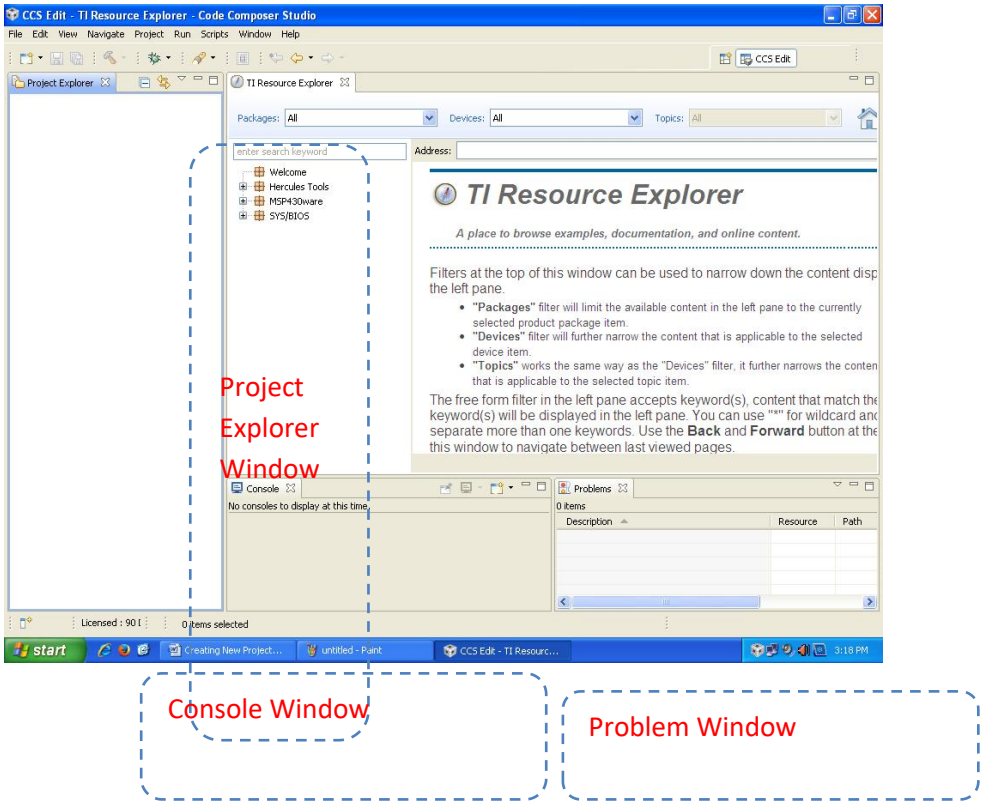
Open *CCS V5.3* from desktop shortcut

It will open default *CCS V5* screen.

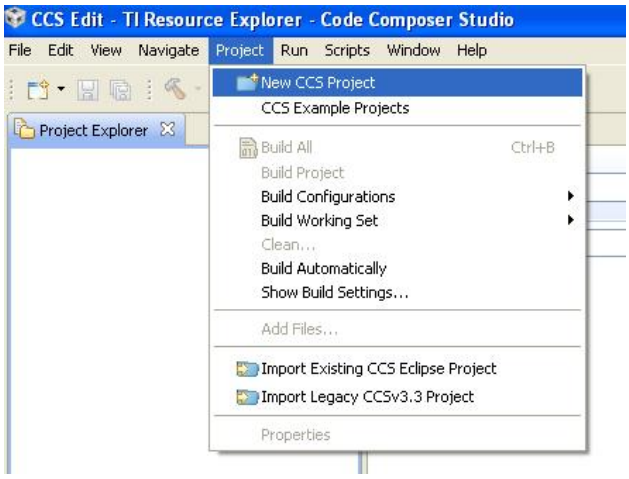
Then it will ask for workspace path Select path "***C:\workspace_v5_3_6713_DSP***" for windows



Then it will open Default CCS5 screen as shown below



Click "*Project -> New CCS Project*" menu.

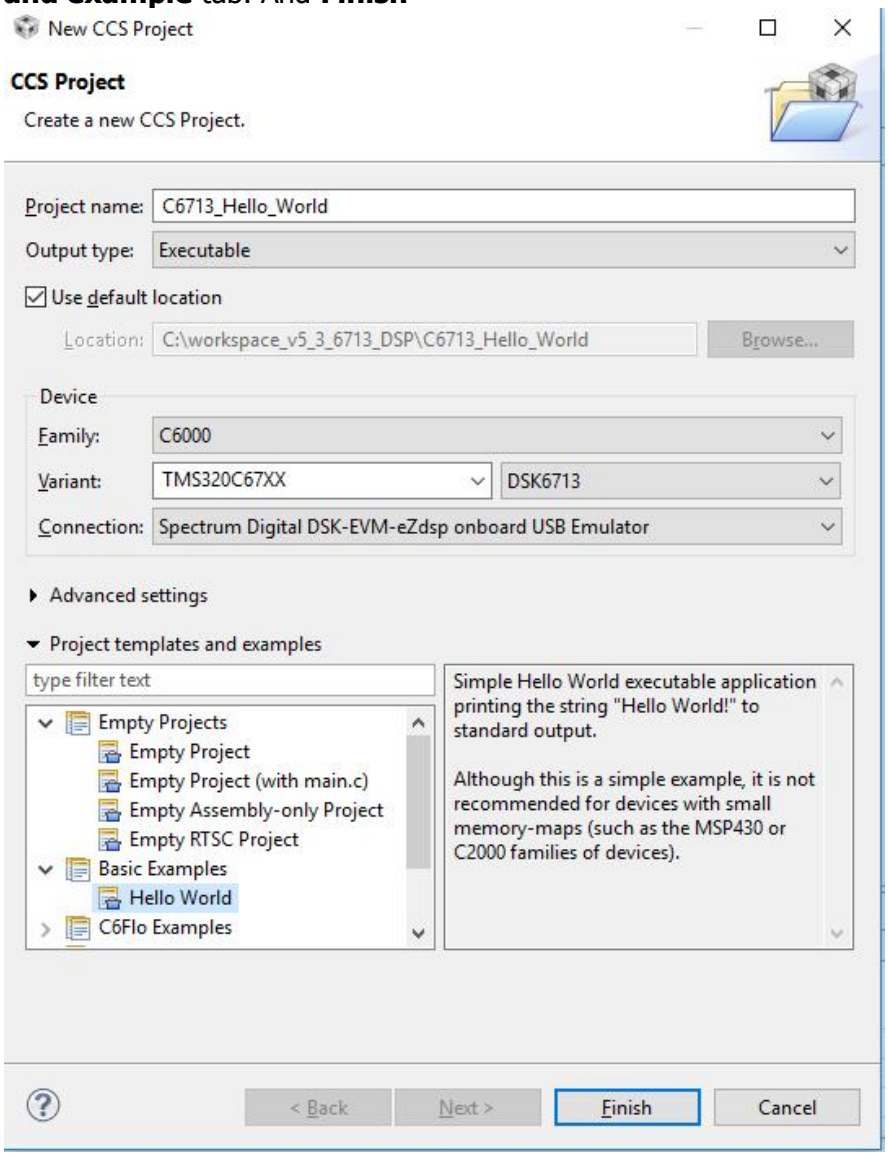


It will open following screen

Project name as desired, - e.g "C6713_Hello_World"
Output type: *Executable* as in figure.
And keep selected "***use default location***" so that project will be created in workspace with project name typed

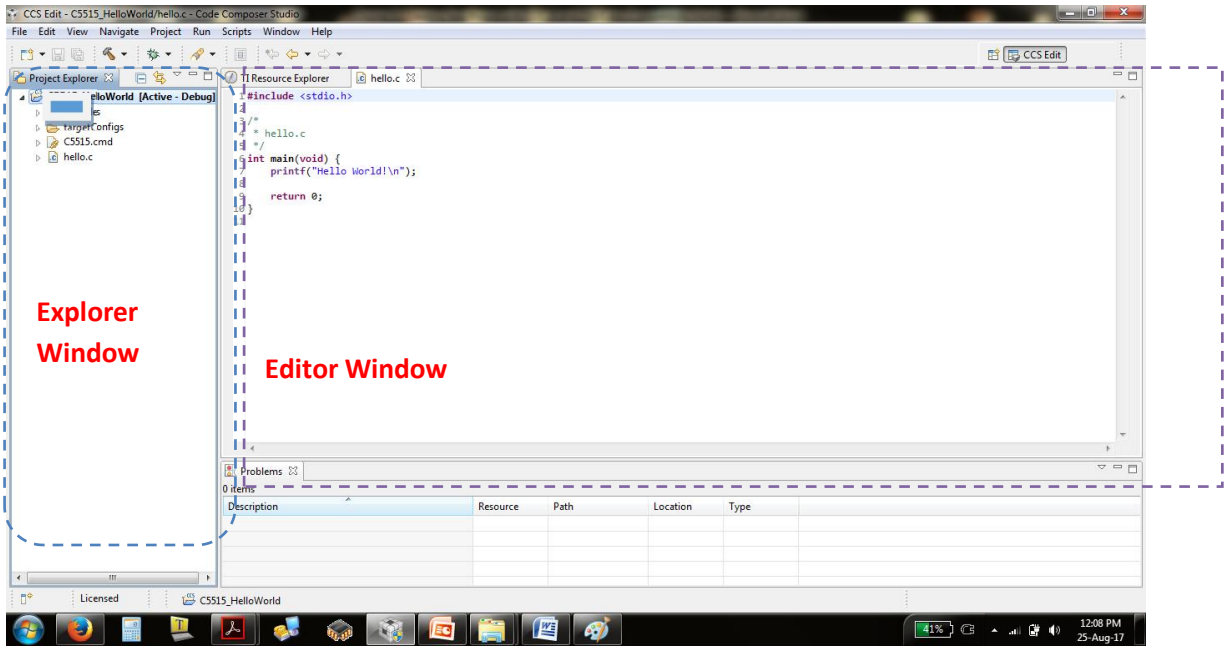
Select family: *C6700*,
Variant: TMS320C*C67xx*
Processor: *DSK6713*
And use **connection** type as *Spectrum Digital DSK-EVM-eZdsp onboard USB emulator*

Then at last select "*Hello World*" example from "*Basic Examples*" location in **Project Templates and example** tab. And **Finish**



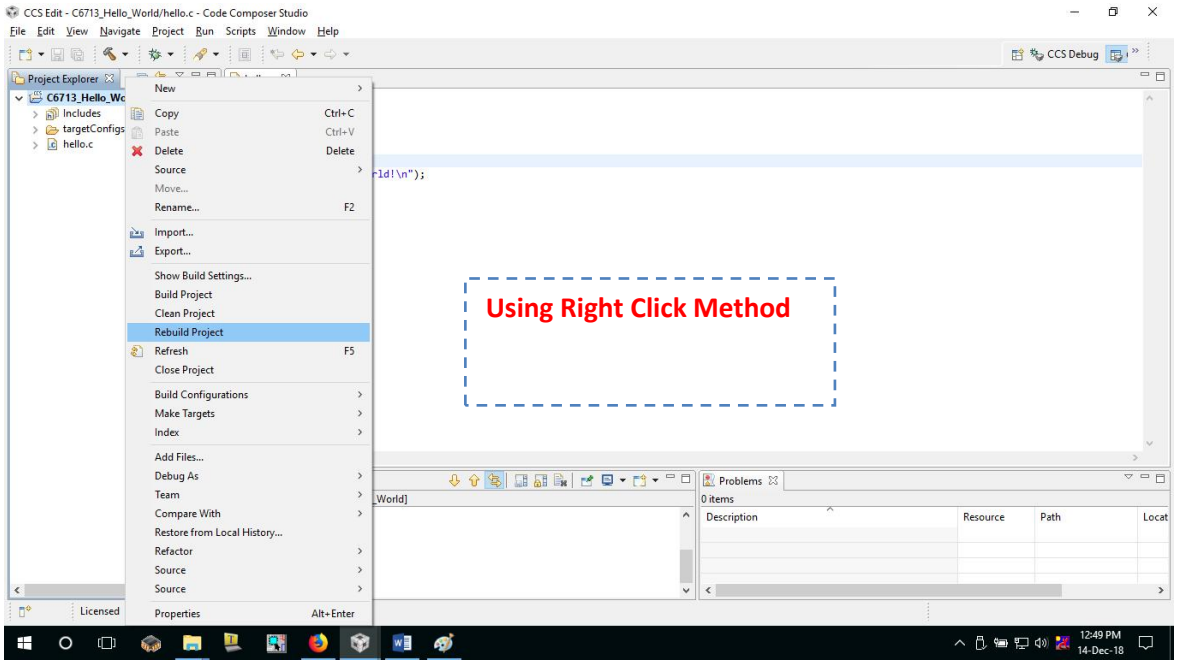
It will open screen as shown here. Here project is already created and it can be seen from "**project explorer**"

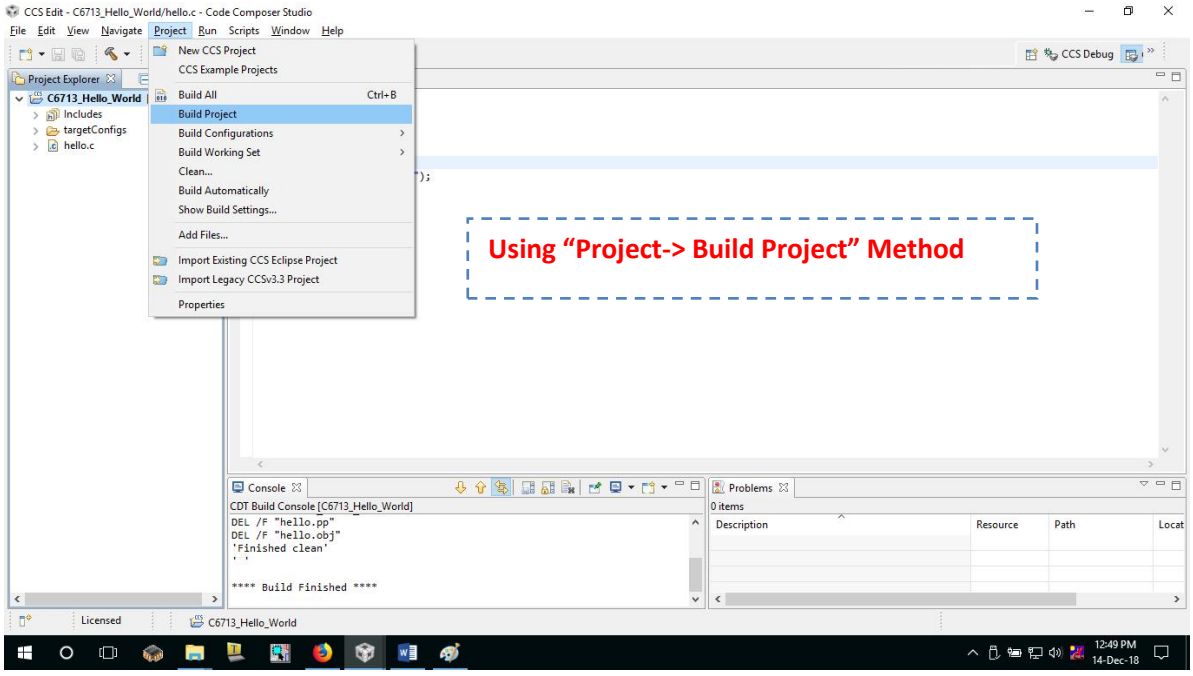
Editor window will show *hello.c* file which can be edited as per requirement.



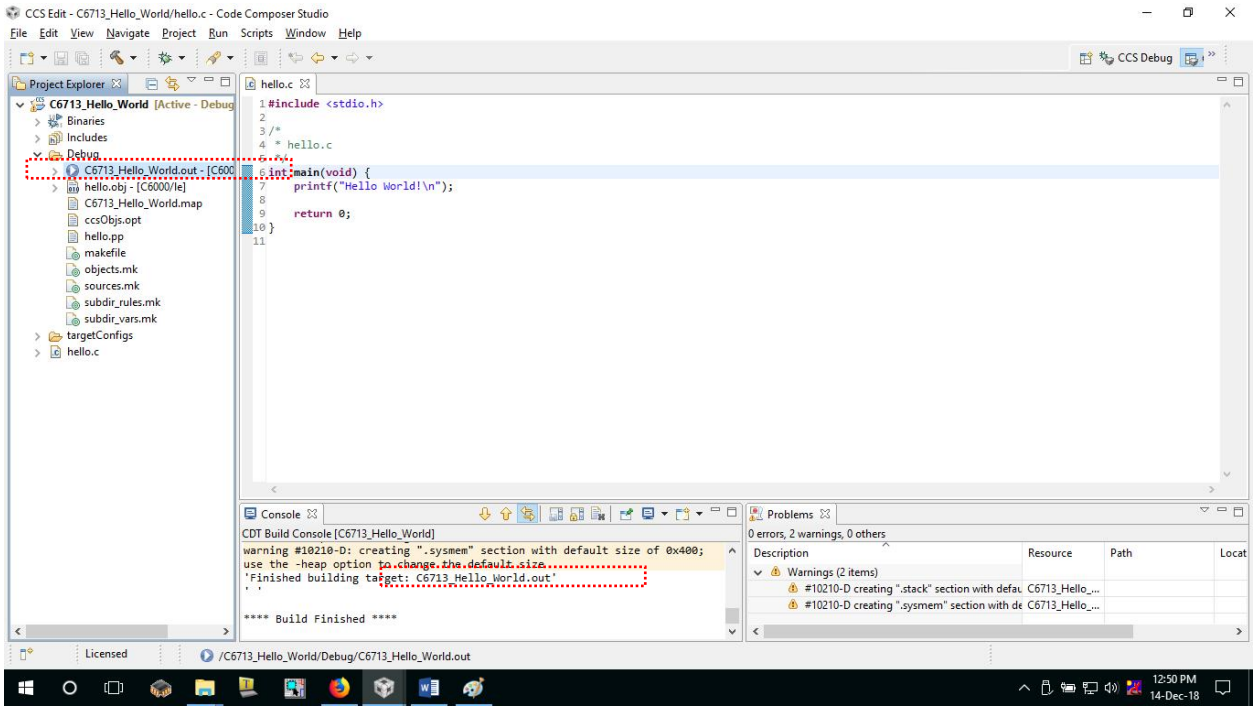
Steps to Build the project:

Compile the program by "*right click-> build project*" or "*right click-> rebuild project*" as shown.





It will generate *C6713_HelloWorld.out* file in "*debug*" folder.



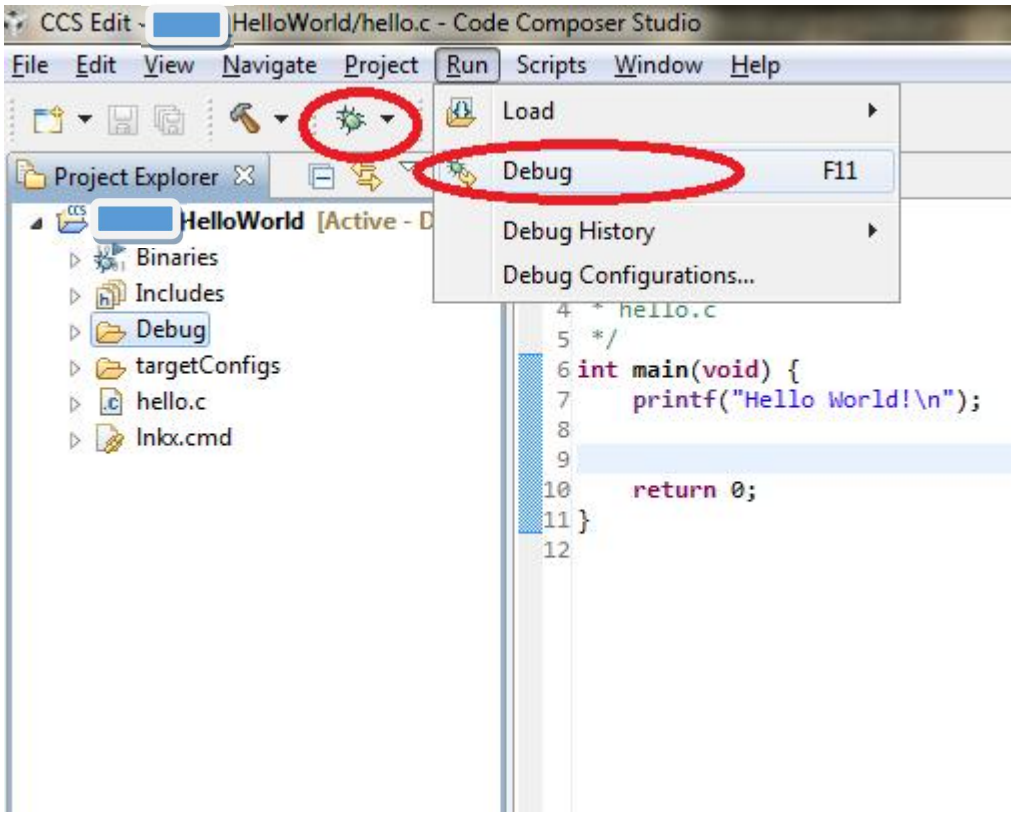
Steps to Run the project:

Steps for Hardware connection:

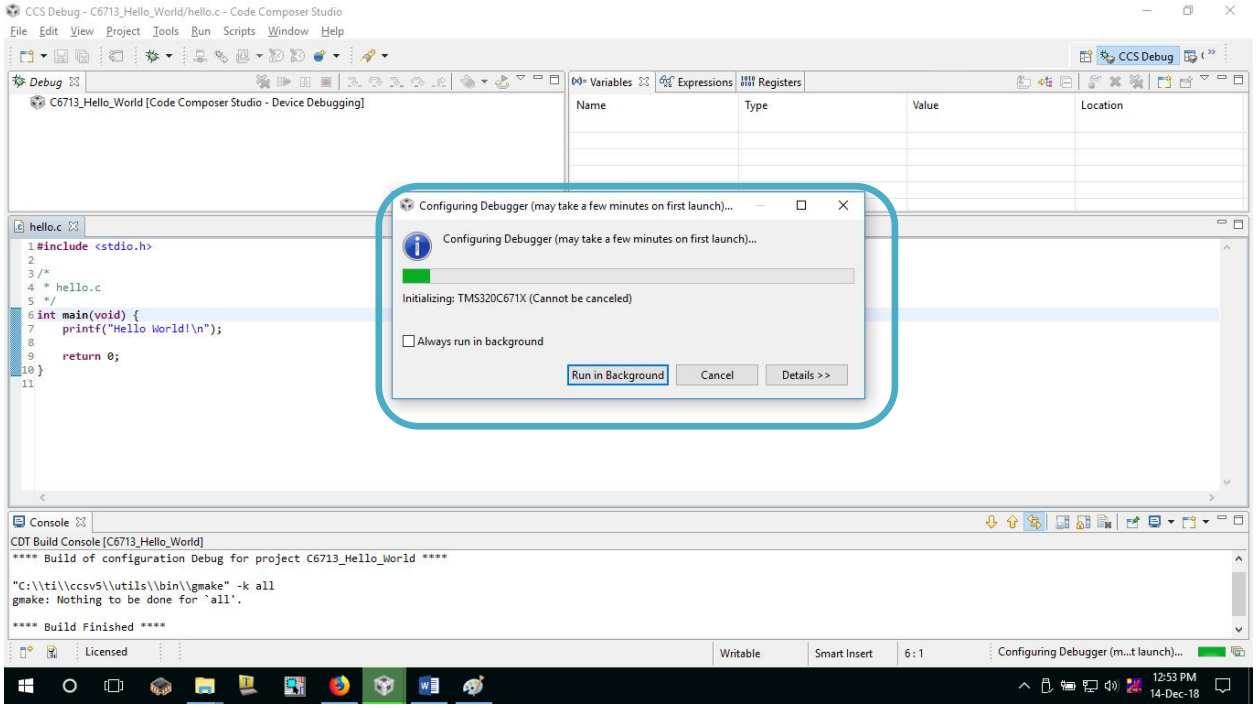
- Power on EPB_C6713 hardware using +5V Power supply
- Connect EPB_C6713 with PC using USB A-to-B cable.
- Reset CPU board DSK_6713

Steps to run/debug program:

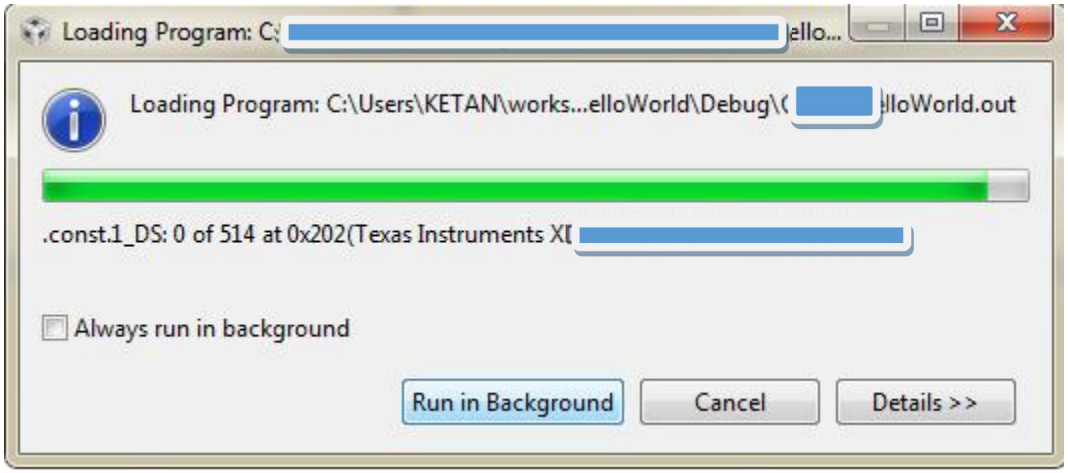
Now to debug the program click "**debug**" as shown in the screen from home screen icon **OR** from "**run->debug**" menu.



It will configure/connect DSK_C6713 kit with the CCSV5. It will be done automatically.

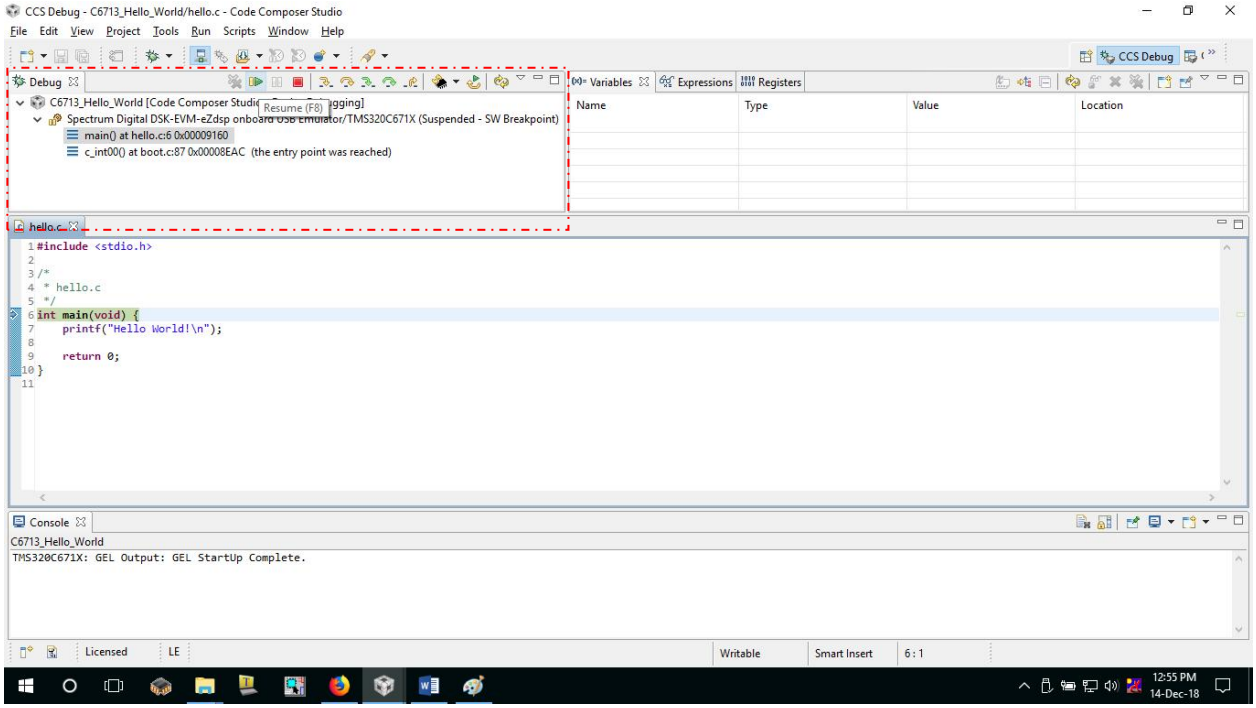


Once Configuration is over, it will start loading program into the CPU using JTAG emulator

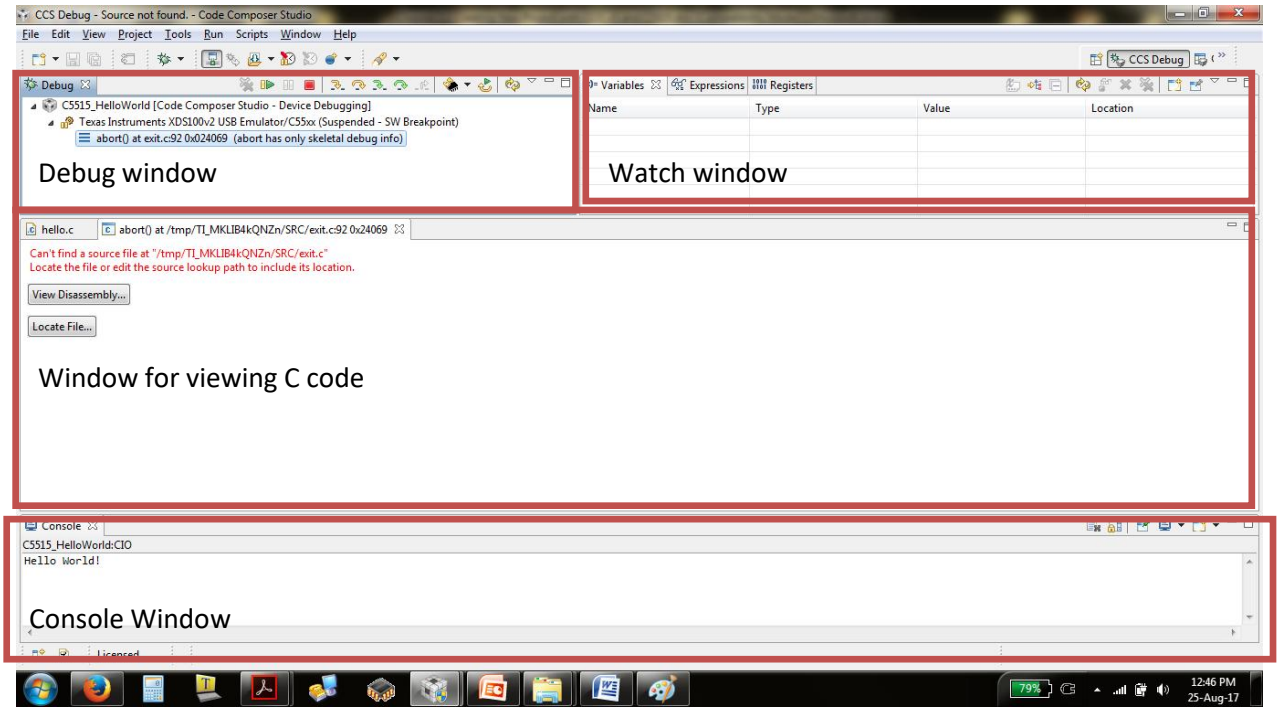


Time to Run/Execute the project:

Once program is loaded click "*resume*". It will run/execute the program and give output on consol window



Check output on output consol window



To stop the project use Halt/Terminate as shown here in dotted red colored highlight and it stops the running program and closes the active project.