TMS320C6748/OMAPL138 DEVELOPMENT KIT

USER MANUAL

StarCom Information Technology Limited

"Times Square", # 88, M.G. Road, Bangalore - 560001

Phone: +91-80-67650000

CONTENTS

OMAPL138/TMS320C6748 DSK

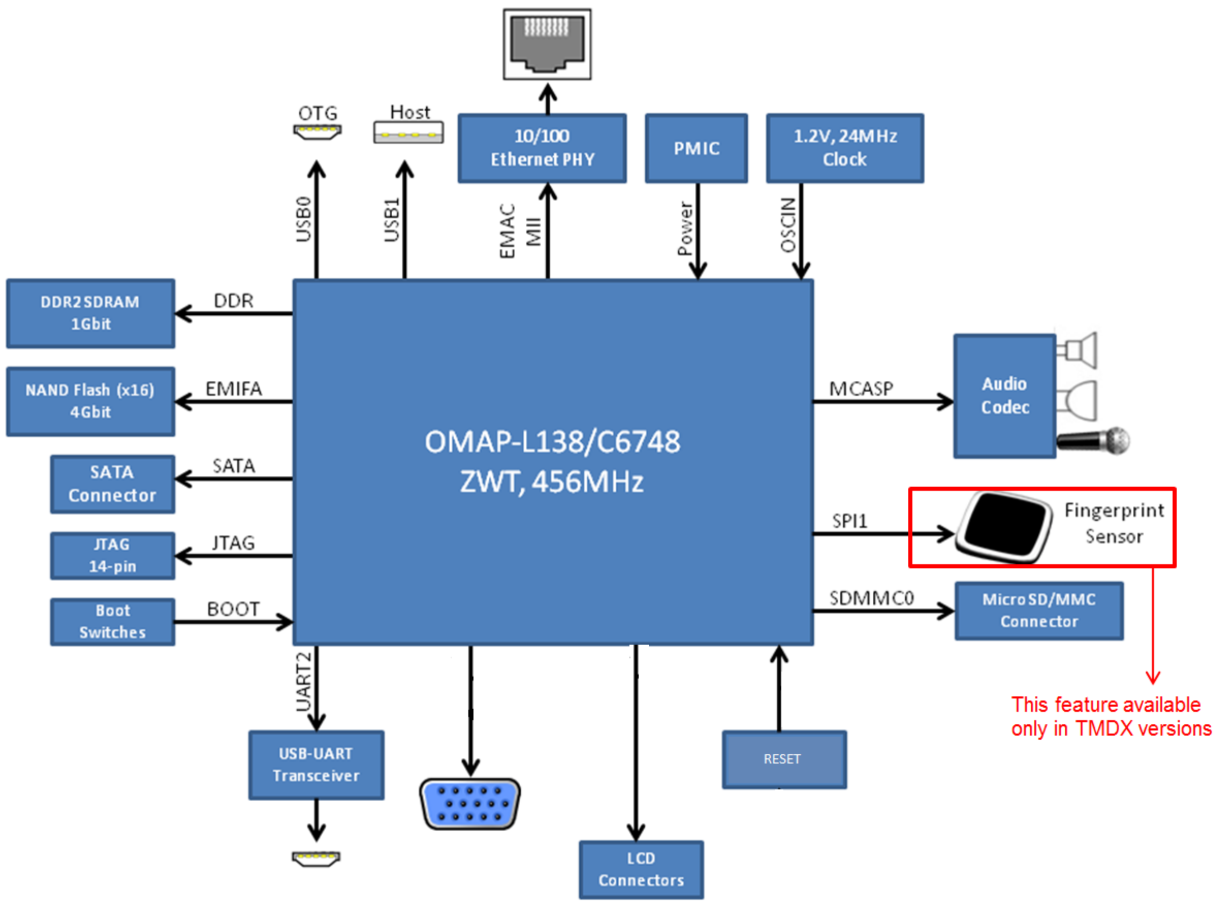
Kit Contents:

- OMAP L138/C6748 Hardware Board
- Power Card
- XDS 100v2 JTAG Emulator
- USB Cable
- User Manual

Description:

The TMS320C6748 DSP development kit (LCDK) is a scalable platform that breaks down development barriers for applications that require embedded analytics and real-time signal processing, including biometric analytics, communications and audio. The low-cost LCDK will also speed and ease your hardware development of real-time DSP applications. This new board reduces design work with freely downloadable and duplicable board schematics and design files. A wide variety of standard interfaces for connectivity and storage enable you to easily bring audio, video and other signals onto the board.

The LCDK does not have an onboard emulator. An external emulator from TI (such as the XDS100, XDS200,XDS510, XDS560) or a third-party will be required to start development.



The OMAP-L138 C6000 DSP+ARM processor is a low-power applications processor based on an ARM926EJ-S and a C674x DSP core. This processor provides significantly lower power than other members of the TMS320C6000™ platform of DSPs

The dual-core architecture of the device provides benefits of both DSP and reduced instruction set computer (RISC) technologies, incorporating a high-performance TMS320C674x DSP core and an ARM926EJ-S core.

The ARM926EJ-S is a 32-bit RISC processor core that performs 32-bit or 16-bit instructions and processes 32-bit, 16-bit, or 8-bit data. The core uses pipelining so that all parts of the processor and memory system can operate continuously

The device DSP core uses a 2-level cache-based architecture. The level 1 program cache (L1P) is a 32- KB direct mapped cache, and the level 1 data cache (L1D) is a 32-KB 2-way, set-associative cache. The level 2 program cache (L2P) consists of a 256-KB memory space that is shared between program and data space. L2 memory can be configured as mapped memory, cache, or combinations of the two. Although the DSP L2 is accessible by the ARM9 and other hosts in the system, an additional 128KB of RAM shared memory is available for use by other hosts without affecting DSP performance.

Processor

- TI TMS320C6748 DSP Application Processor
- 456-MHz C674x Fixed/Floating Point DSP
- 456-MHz ARM926EJ RISC CPU (OMAP-L138 only)
- On-Chip RTC

Memory

- 128 MByte DDR2 SDRAM running at 150MHz
- 128 MByte 16-bit wide NAND FLASH
- 1 Micro SD/MMC Slot

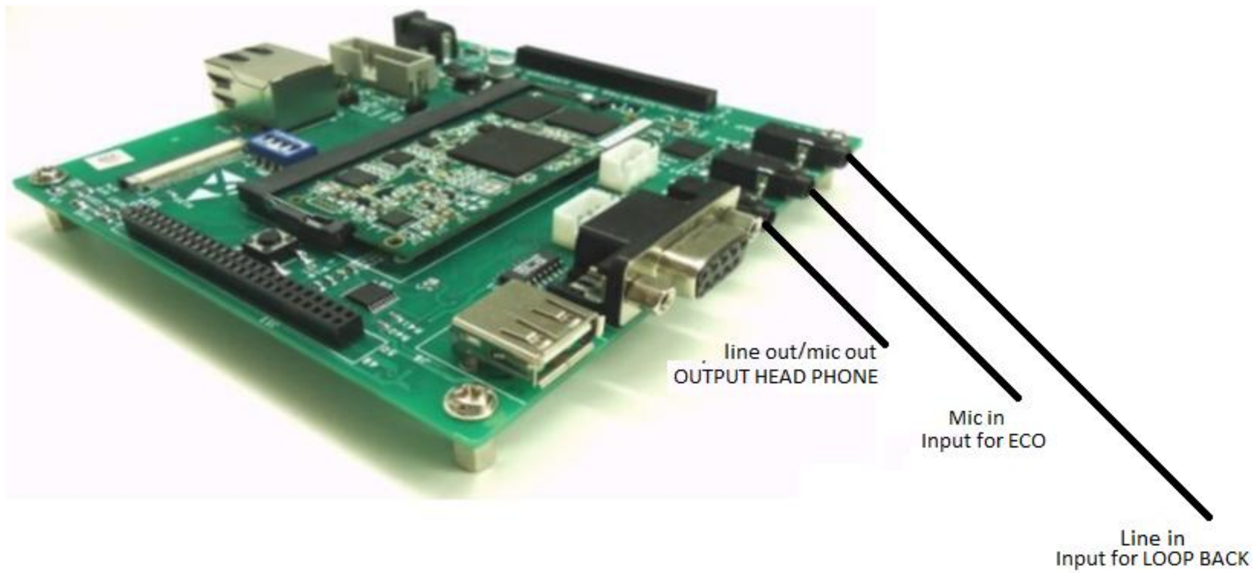
Interfaces

- One mini-USB Serial Port (on-board serial to USB)
- One Fast Ethernet Port (10/100 Mbps) with status LEDs
- One USB Host port (USB 1.1)
- One SATA Port (3Gbps)
- One LCD Port (Beagleboard XM connectors)
- One Leopard Imaging Camera Sensor Input (36-pin ZIP connector)
- Three AUDIO Ports (1 LINE IN-J55 & 1 LINE OUT-J56 & 1 MIC IN-J57)
- 14-pin JTAG header (No onboard emulator; external emulator is required)

TMS320C6748 DSP Features

- Highest-Performance Floating-Point Digital Signal Processor (DSP):
  - Eight 32-Bit Instructions/Cycle
  - 32/64-Bit Data Word
  - 375/456-MHz C674x Fixed/Floating-Point
  - Up to 3648/2746 C674x 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:
    - Two ALUs (Fixed-Point)
    - Four ALUs (Floating- and Fixed-Point)
    - Two Multipliers (Floating- and Fixed-Point)
  - Load-Store Architecture With 64 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
- 67x cache memory.
  - 32K-Byte L1P Program Cache (Direct-Mapped)
  - 32K-Byte L1D Data Cache (2-Way)
- 256K-Byte L2 unified Memory RAM\Cache.
- Real-Time Clock With 32 KHz Oscillator and Separate Power Rail**.**
- Three 64-Bit General-Purpose Timers
  - 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™) .
- 3 64-Bit General-Purpose Timers (each configurable as 2 32-bit timers)
- Flexible Phase-Locked-Loop (PLL) Based Clock Generator Module
- IEEE-1149.1 (JTAG †) Boundary-Scan-Compatible
- 3.3-V I/Os, 1.2 ‡ -V Internal (GDP & PYP)
- 3.3-V I/Os, 1.4-V Internal (GDP)(300 MHz only)
- Two Serial Peripheral Interfaces (SPI) Each With Multiple Chip-Selects
- Two Multimedia Card (MMC)/Secure Digital Card Interface with Secure Data I/O (SDIO) Interfaces
- One Multichannel Audio Serial Port.
- Two Multichannel Buffered Serial Ports



INSTALLATION

SYSTEM REQUIREMENTS:

	Recommended system requirements	Minimum system requirements
Processor	PENTIUM OR HIGHER	1.5GHz
RAM	4GB or above	1GB
Free Disk Space	4GB	4GB
Operating System	Microsoft windows 8/7/XP	-DO-

SOFTWARE INSTALLATION:

CCSv5 or CCSv6 (**http://processors.wiki.ti.com/index.php/Download_CCS**)

C64XPLUS-IMGLIB (**http://www.ti.com/tool/sprc264**)

HARDWARE REQUIREMENTS:

- Kit connected with 5V power supply
- USB cable connected with PC and JTAG emulator.
- Double sided stereo cable and speakers/ear phones (For Audio processing)
- Stereo cables, Function generator and CRO (For signal filtering applications)

IMPORTANT (MUST BE DONE BEFORE START WORKING WITH CCS):

COPY THE FILE: "C6748_LCDK.gel" FROM *C6748 STARCOM BOARD\LCDK folder\Supporting files*

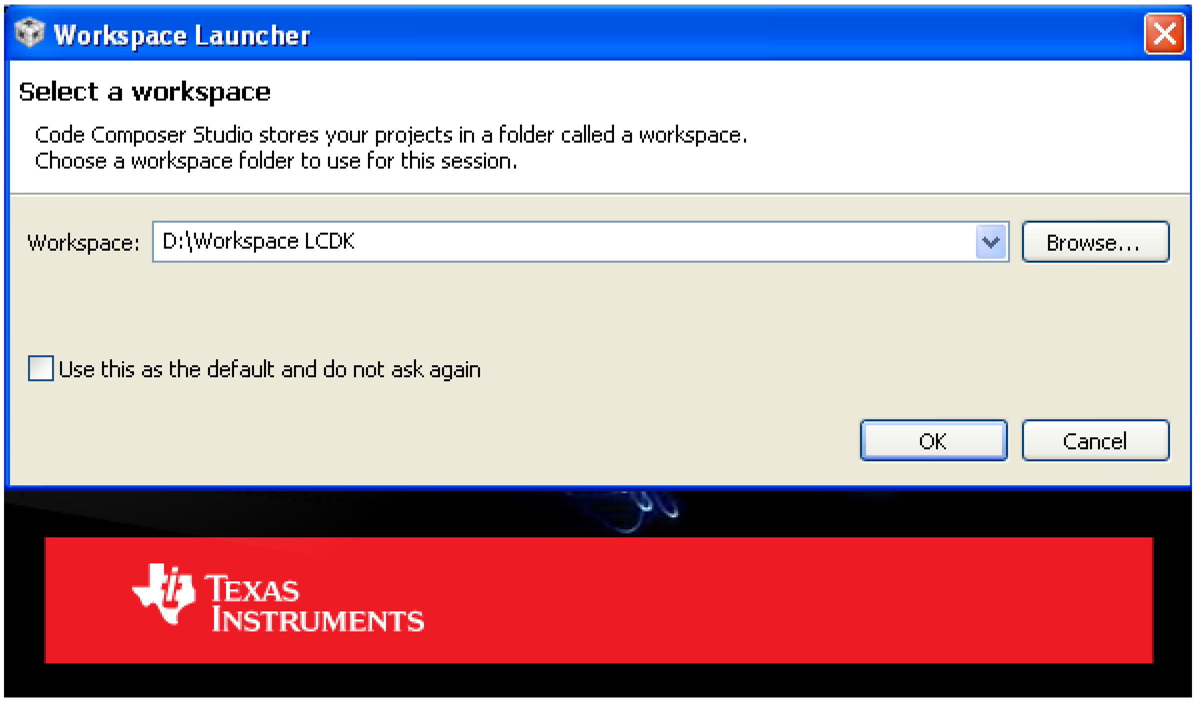
AND REPLACE WITH THE FILE IN *C:\ti\ccsv6\ccs_base\emulation\boards\lcdkc6748\gel\*

General Procedure to work on Code Composer Studio V6 for non-real time projects Using target:

NR1. Launch CCS

Launch the CCS v6 icon from the Desktop or goto **All Programs -> Texas Instruments -> CCSv6**

NR2. Choose the location for the workspace, where your project will be saved.



Workspace Launcher

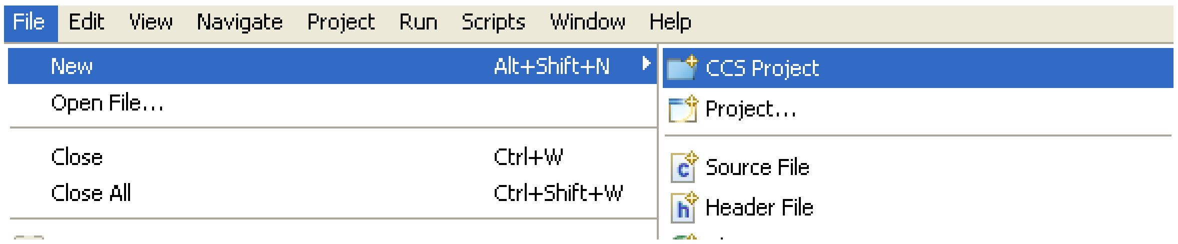
Select a workspace

Code Composer Studio stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: D:\Workspace LCDK

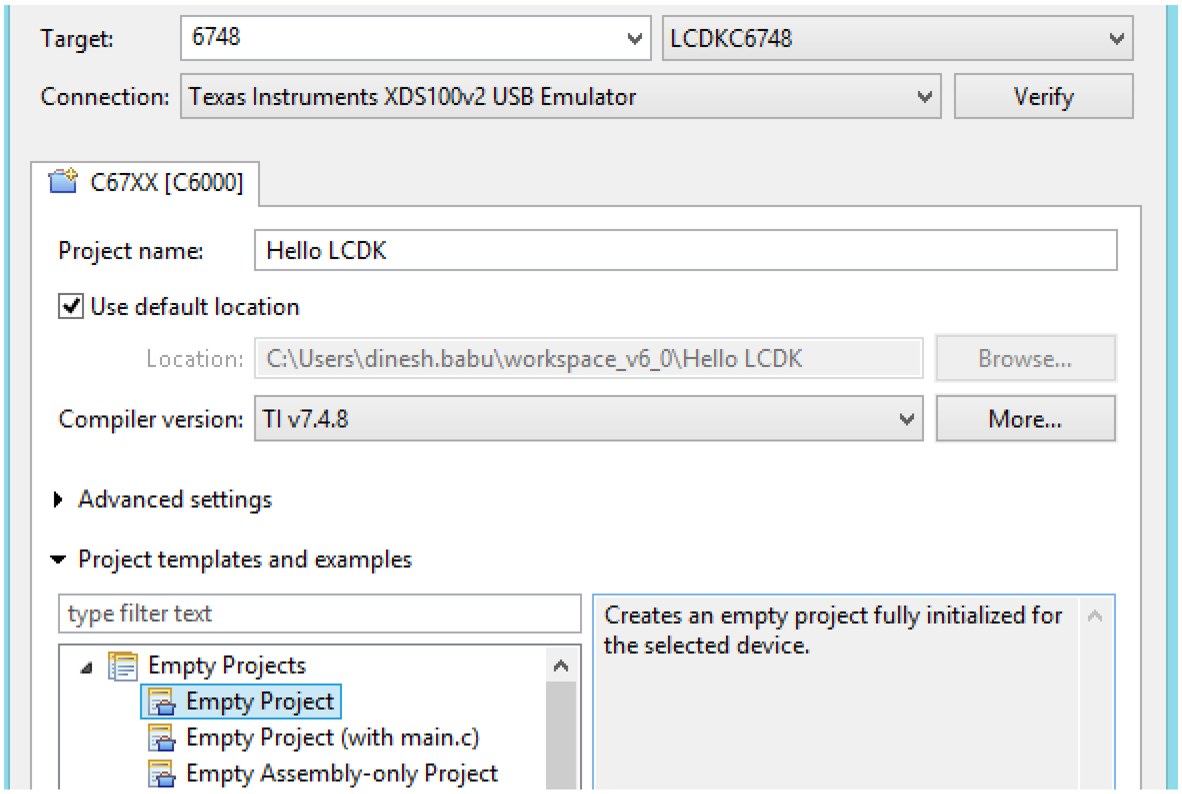
NR3. Creating a New Project:

➢ Go to File → New → CCS Project.



➢ Specify the name of the project in the space provided e g., Project Name: Hello LCDK.

Specify the “Device” properties as shown in the figure below and select an “Empty Project”.

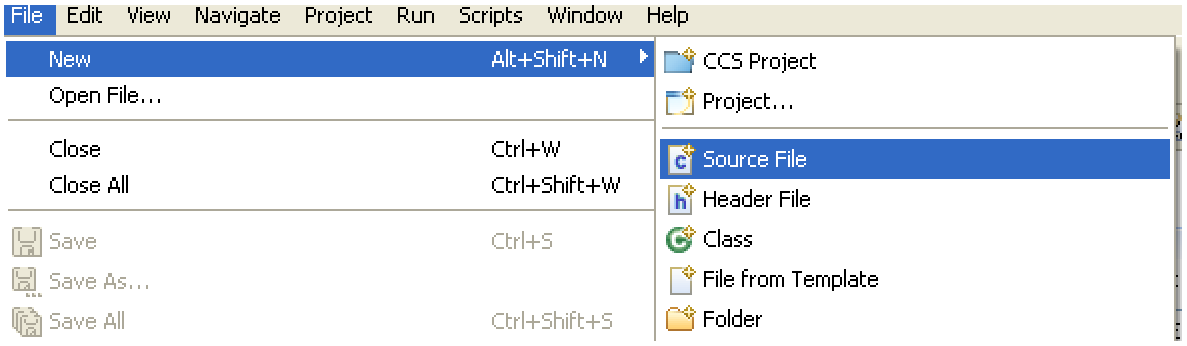


Target: 6748 LCDKC6748
Connection: Texas Instruments XDS100v2 USB Emulator
C67XX [C6000]
Project name: Hello LCDK
☑ Use default location
Location: C:\Users\dinesh.babu\workspace_v6_0\Hello LCDK
Compiler version: TI v7.4.8
Advanced settings
Project templates and examples
Empty Projects
Empty Project
Empty Project (with main.c)
Empty Assembly-only Project
Creates an empty project fully initialized for the selected device.

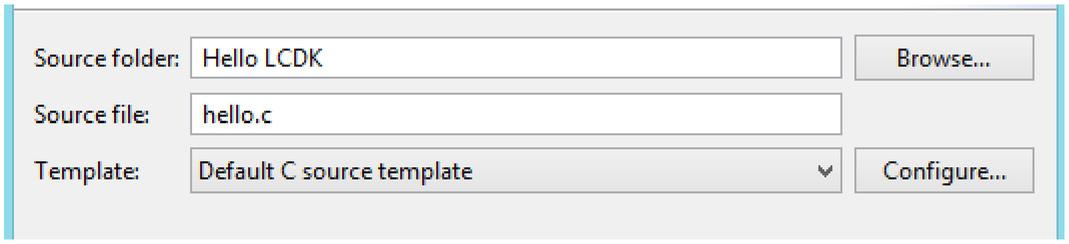
Click **Finish**

NR4. To write a program, select one new source file.

A. Go to File → New → Source File.

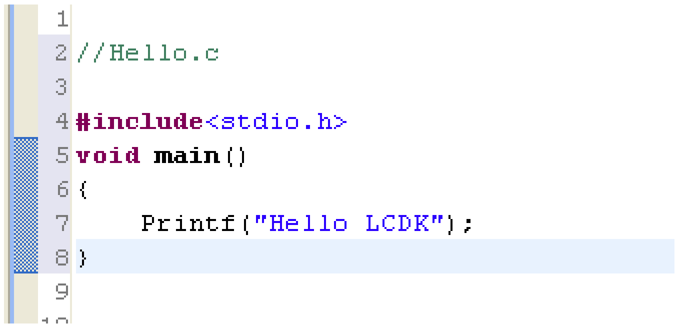


➢ Specify the arbitrary source file name with “.c” extension. It should be in the source folder (current project name).



Source folder:	Hello LCDK	Browse...
Source file:	hello.c	
Template:	Default C source template	Configure...

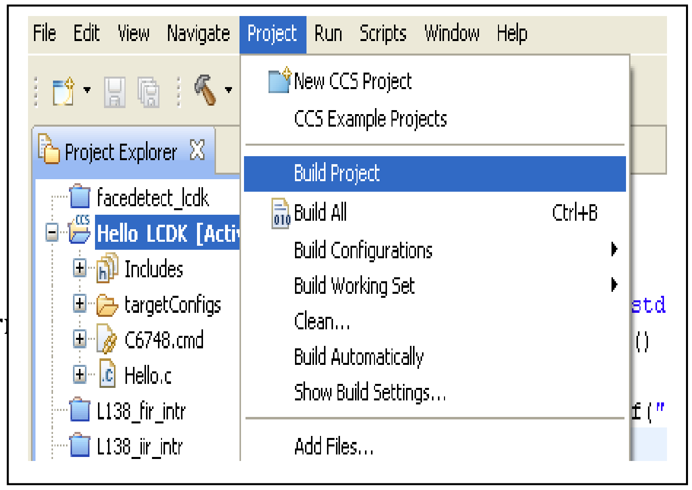
➢ Type your C Code and save.



```

//Hello.c

#include<stdio.h>
void main()
{
    Printf("Hello LCDK");
}

```

.

NR5. BUILD



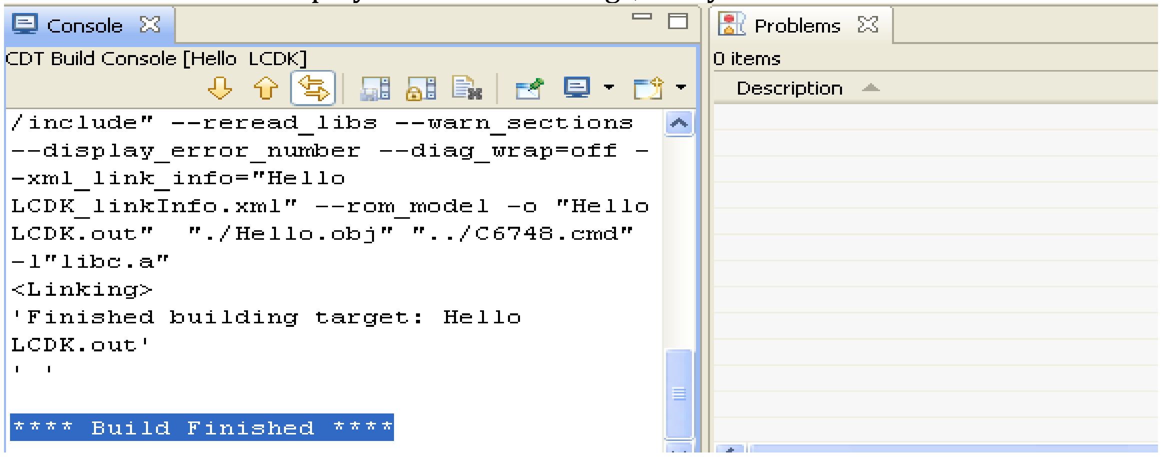
Build the program to check your code.

Go to

Project → Build project.

If your code doesn't have any errors and war
window that "**** Build Finished ****"

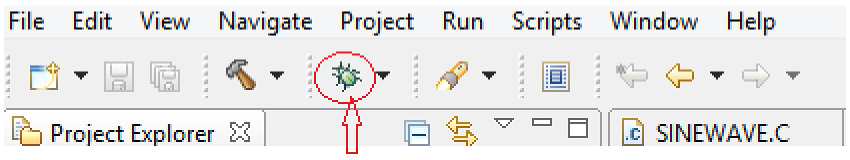
Problems window display errors or warnings, if any.



```
/include" --reread_libs --warn_sections
--display_error_number --diag_wrap=off -
-xml_link_info="Hello
LCDK_linkInfo.xml" --rom_model -o "Hello
LCDK.out"  "./Hello.obj" "../C6748.cmd"
-l"libc.a"
<Linking>
'Finished building target: Hello
LCDK.out'
' '

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

NR6. DEBUG

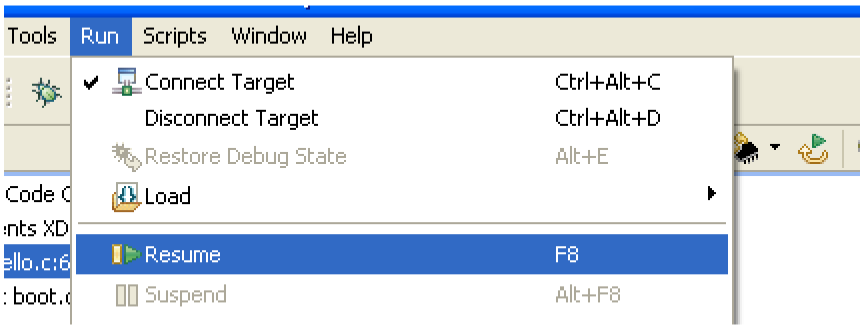
After successful Build, connect the kit with the system using the JTAG emulator and power the kit.
Click the **Debug** as shown in the below figure.



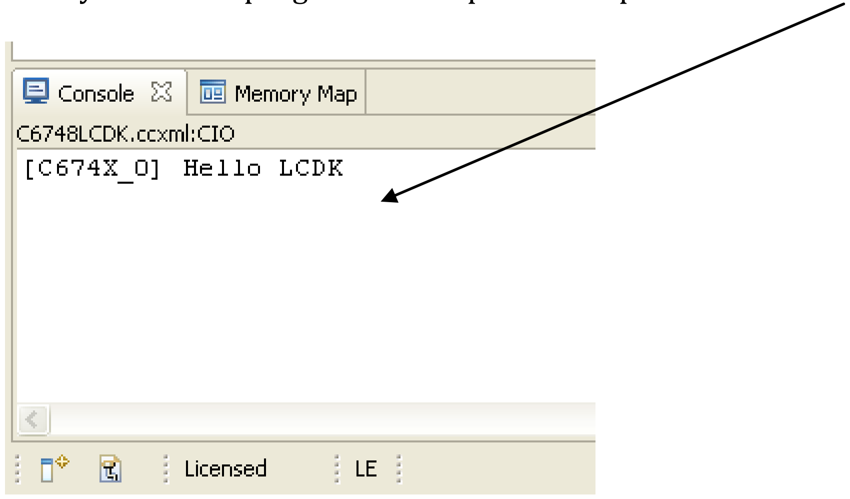
It will redirect to the Debug perspective automatically.

NR7. RUN

Wait until the program loaded to the hardware automatically.
Now you can run the code, by selecting Run-> **Resume**.


Tools
Run
Scripts
Window
Help
Connect Target Ctrl+Alt+C
Disconnect Target Ctrl+Alt+D
Restore Debug State Alt+E
Load
Resume F8
Suspend Alt+F8

Once you run the program the output will be printed in the Console Window.



```
C6748LCDK.ccxml:CIO
[C674X_0] Hello LCDK
```

We can also double click on the Console Window to view it on full screen.

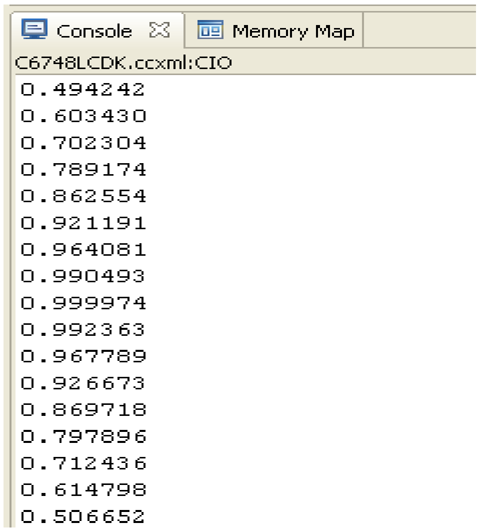
SINE WAVE GENERATION

Procedure to create a simple non-real time project - **Sine wave Generation:**

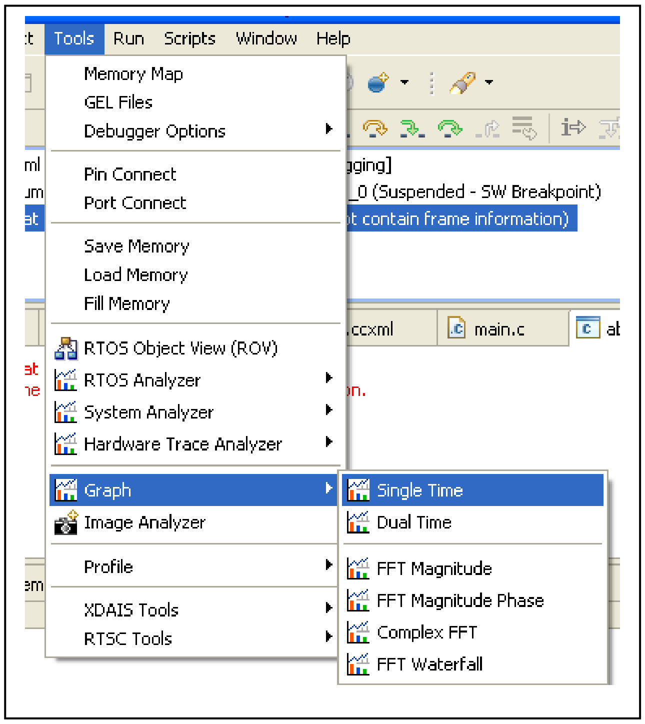
```
#include <stdio.h>
#include<math.h>
#define FREQ 500
float m[128];
main()
{
 int i=0;
  for(i=0;i<127;i++)
 {
  m[i]=sin(2*3.14*FREQ*i/24000);
  printf("%f\n",m[i]);
}
}
```

```
m[i]=sin(2*3.14*FREQ*i/24000);
printf("%f\n",m[i]); }}
```

Once you run the program, 128 no's of sine samples has been printed in the Console Window.



```
C6748LCDK.ccxml:CIO
0.494242
0.603430
0.702304
0.789174
0.862554
0.921191
0.964081
0.990493
0.999974
0.992363
0.967789
0.926673
0.869718
0.797896
0.712436
0.614798
0.506652
```



To view the values in graph
Go to

Tools → Graph →Single Time

- Set the Graph Properties as shown in the

Picture

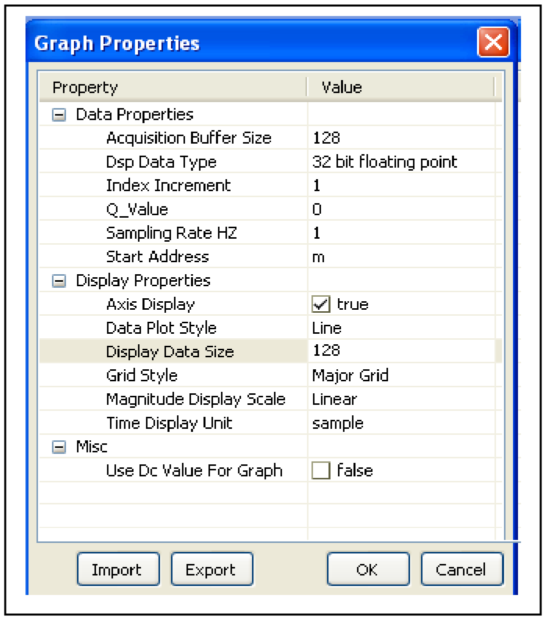
Buffer size: **128 (No. of samples)**

DSP Data Type: **32-Bit floating point**

Start Address: **m**

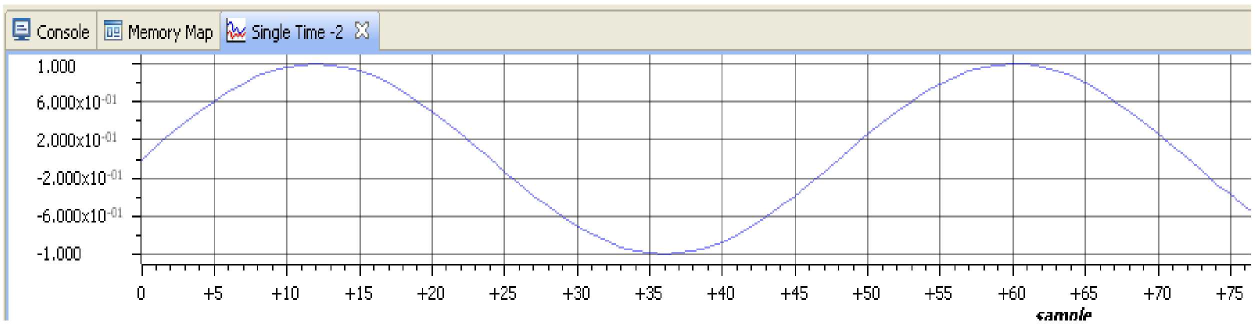
Display Data Size: **128**

Click: **Ok**



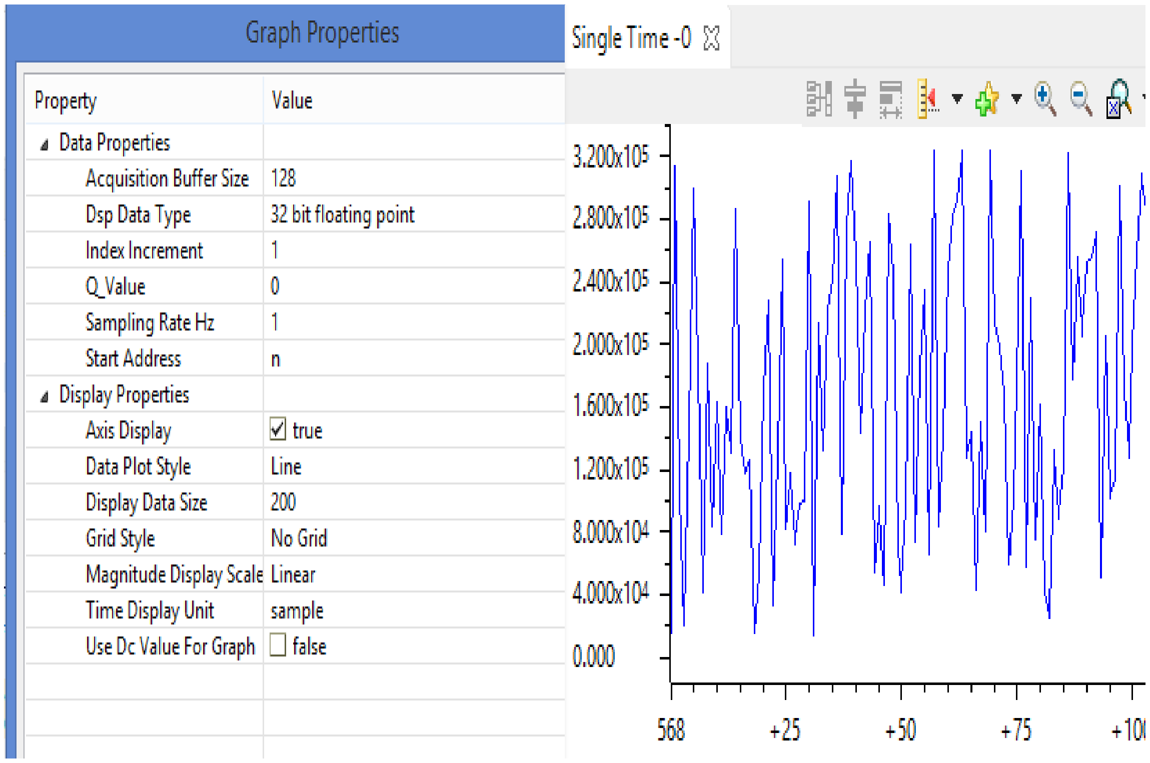
Property	Value
Data Properties	
Acquisition Buffer Size	128
Dsp Data Type	32 bit floating point
Index Increment	1
Q_Value	0
Sampling Rate HZ	1
Start Address	m
Display Properties	
Axis Display	✓ true
Data Plot Style	Line
Display Data Size	128
Grid Style	Major Grid
Magnitude Display Scale	Linear
Time Display Unit	sample
Misc	
Use Dc Value For Graph	false

Graph will be plotted as shown below:


Console
Memory Map
Single Time -2
1.000
6.000x10-01
2.000x10-01
-2.000x10-01
-6.000x10-01
-1.000
0
+5
+10
+15
+20
+25
+30
+35
+40
+45
+50
+55
+60
+65
+70
+75
sample

RANDOM WAVE GENERATION

```
#include<stdio.h>
#include<math.h>
#define PI 3.14
#define PTS 128
float x[PTS];
float y[PTS];
float z[PTS];
float n[PTS];

void main()
{
      int i,j;
      for (i = 0 ; i < PTS ; i++)
      {
            x[i] = sin(2*PI*i*20/128.0);
            y[i]=0.0;
      n[i]=x[i] + rand() * 10 ; //rand function to generate random signal
      }
}
```



Graph Properties

Property	Value
Data Properties	
Acquisition Buffer Size	128
Dsp Data Type	32 bit floating point
Index Increment	1
Q_Value	0
Sampling Rate Hz	1
Start Address	n
Display Properties	
Axis Display	☑ true
Data Plot Style	Line
Display Data Size	200
Grid Style	No Grid
Magnitude Display Scale	Linear
Time Display Unit	sample
Use Dc Value For Graph	☐ false

LINEAR CONVOLUTION

To Verify Linear Convolution:

Linear Convolution involves the following operations.

1. Folding
2. Multiplication
3. Addition
4. Shifting

These operations can be represented by a Mathematical Expression as follows:

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

x[ ]= Input signal Samples
h[ ]= Impulse response co-efficient.
y[ ]= Convolution output.
n = No. of Input samples
h = No. of Impulse response co-efficient.

Algorithm to implement 'C' or Assembly program for Convolution:

Eg: x[n] = {1, 2, 3, 4}
h[k] = {1, 2, 3, 4}

Where: n=4, k=4. ; Values of n & k should be a multiple of 4.
If n & k are not multiples of 4, pad with zero's to make Multiples of 4
r= n+k-1 ; Size of output sequence.
= 4+4-1
= 7.

r=	0	1	2	3	4	5	6
n= 0	x[0]h[0]	x[0]h[1]	x[0]h[2]	x[0]h[3]			
1		x[1]h[0]	x[1]h[1]	x[1]h[2]	x[1]h[3]		
2			x[2]h[0]	x[2]h[1]	x[2]h[2]	x[2]h[3]	
3				x[3]h[0]	x[3]h[1]	x[3]h[2]	x[3]h[3]

Output: y[r] = {1, 4, 10, 20, 25, 24, 16}.

NOTE: At the end of input sequences pad 'n' and 'k' no. of zero's

PROGRAM: LINEAR.C

```
#include<stdio.h>
#define LENGHT1 6 /*Lenght of i/p samples sequence*/
#define LENGHT2 4 /*Lenght of impulse response Co-efficients */
int x[2*LENGHT1-1]={1,2,3,4,5,6,0,0,0,0,0}; /*Input Signal Samples*/
int h[2*LENGHT1-1]={1,2,3,4,0,0,0,0,0,0,0}; /*Impulse Response Coefficients*/
int y[LENGHT1+LENGHT2-1];
main()
{
int i=0,j;
for(i=0;i<(LENGHT1+LENGHT2-1);i++)
{
y[i]=0;
for(j=0;j<=i;j++)
y[i]+=x[j]*h[i-j];
}
for(i=0;i<(LENGHT1+LENGHT2-1);i++)
printf("%d\n",y[i]);}
```

Procedure:

Follow steps 1-7.

To view the values in graph
Go to

Tools → Graph →Single Time

Set the Graph Properties as shown

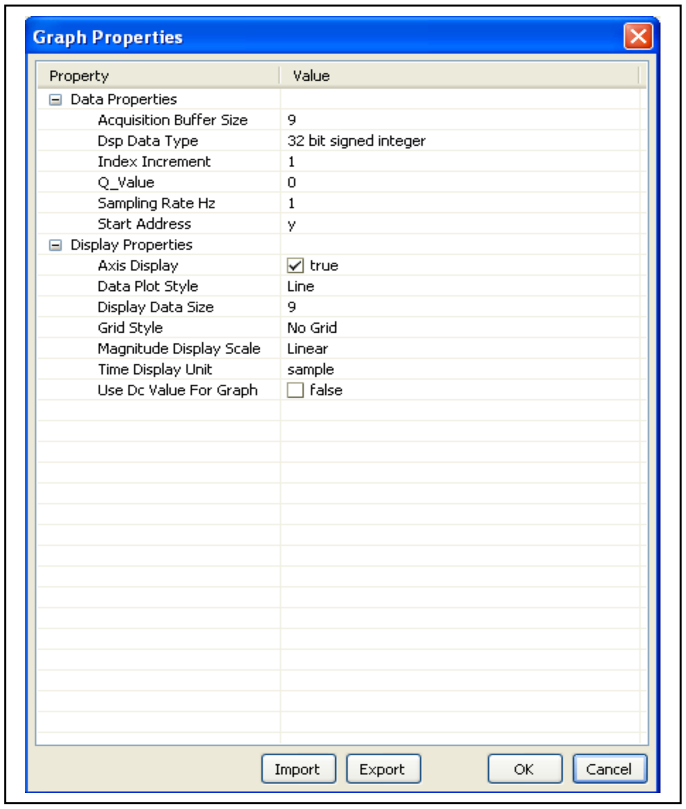
Buffer size : **9**

DSP Data Type : **32-Bit Signed Integer**

Start Address: **y**

Display Data Size **: 9**

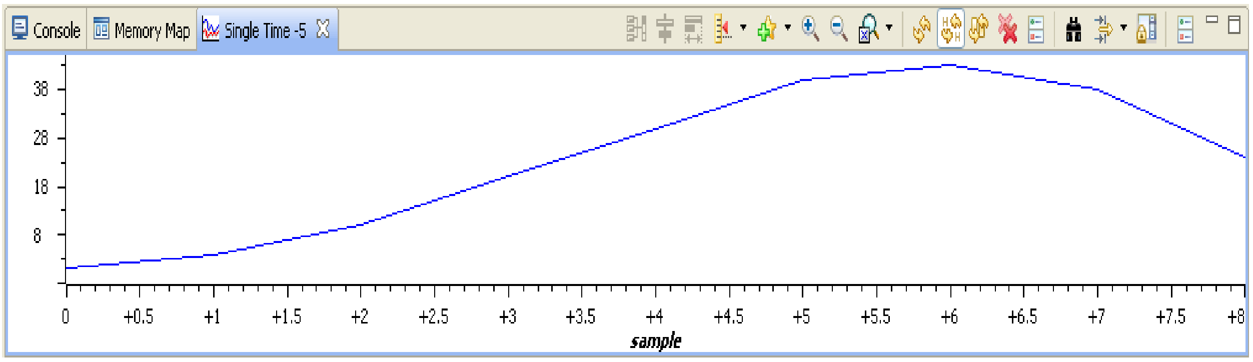
Click : **Ok**



Graph Properties

Property	Value
⊟ Data Properties	
Acquisition Buffer Size	9
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	Line
Display Data Size	9
Grid Style	No Grid
Magnitude Display Scale	Linear
Time Display Unit	sample
Use Dc Value For Graph	☐ false

The graph has been plotted as shown below.



DFT and IDFT

DFT of a discrete time signal transforms the signal in time domain into frequency domain signal.

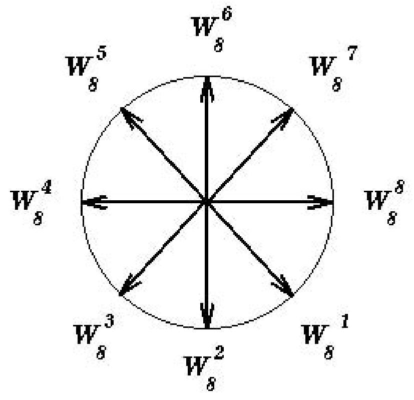
DFT EQUTION:

The sequence of *N* complex numbers $x_0, ..., x_{N-1}$ is transformed into another sequence of *N* complex numbers according to the DFT formula:

$$X(k) = \frac{1}{N} \sum_{n=0}^{N-1} x(n) W_N^{nk}$$

$$where \;\; W_N^{nk} = e^{-j\frac{2\pi nk}{N}} \; \text{[TWIDDLE FACTOR]}$$

where N = number of samples. If we take an 8 bit sample sequence we can represent the twiddle factor as a vector in the unit circle. e.g.



Note:

1 It is periodic. (i.e. it goes round and round the circle !!)
2 That the vectors are symmetric
3 The vectors are equally spaced around the circle

The **inverse discrete Fourier transform (IDFT)** is given by:

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k \cdot e^{+i2\pi \frac{k}{N} n}.$$

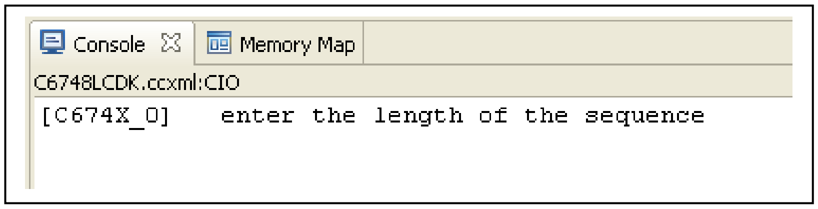
C. PROGRAM TO IMPLEMENT DFT.

```
#include<stdio.h>
#include<math.h>
int N,k,n,i;
float pi=3.1416,sumre=0, sumim=0,out_real[8]={0.0}, out_imag[8]={0.0};
int x[32];
void main(void)
{
printf("  enter the length of the sequence\n");
   scanf("%d",&N);
   printf("  enter the sequence\n");
   for(i=0;i<N;i++)
   scanf("%d",&x[i]);
for(k=0;k<N;k++)
{
sumre=0;
sumim=0;

for(n=0;n<N;n++)
{
sumre=sumre+x[n]* cos(2*pi*k*n/N);
sumim=sumim-x[n]* sin(2*pi*k*n/N);
}
out_real[k]=sumre;
out_imag[k]=sumim;
printf("X([%d])=\t%f\t+\t%fi\n",k,out_real[k],out_imag[k]);
}
}
```

PROCEDURE

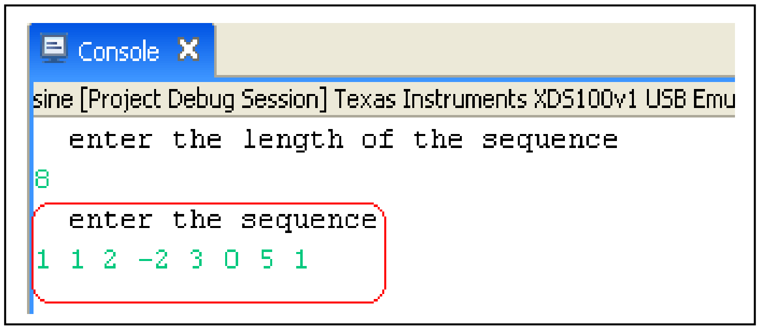
Follow steps 1-7.

Once you run the program. Console window will ask for length of sequence. Enter the length.



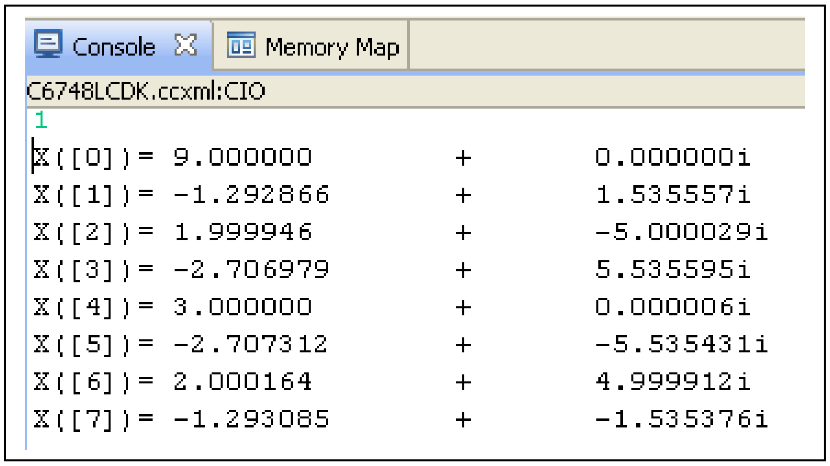
```
C6748LCDK.ccxml:CIO
[C674X_0]    enter the length of the sequence
```

Now it will ask for sequence. Enter the sequence.



```
sine [Project Debug Session] Texas Instruments XDS100v1 USB Emu
 enter the length of the sequence
8
 enter the sequence
1 1 2 -2 3 0 5 1
```

DFT of the given input will appear in console window as shown below.



```
C6748LCDK.ccxml:CIO
1
X([0])= 9.000000         +       0.000000i
X([1])= -1.292866        +       1.535557i
X([2])= 1.999946         +       -5.000029i
X([3])= -2.706979        +       5.535595i
X([4])= 3.000000         +       0.000006i
X([5])= -2.707312        +       -5.535431i
X([6])= 2.000164         +       4.999912i
X([7])= -1.293085        +       -1.535376i
```

FAST FOURIER TRANSFORMS (FFT)

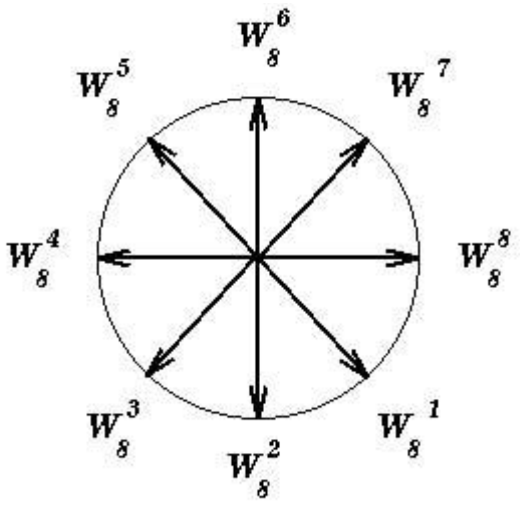
The DFT Equation

$$X(k) = \frac{1}{N}\sum_{n=0}^{N-1} x(n)W_N^{nk}$$

$$where \quad W_N^{nk} = e^{-j\frac{2\pi nk}{N}} \quad \text{[TWIDDLE FACTOR]}$$

Twiddle Factor

Where N = number of samples. If we take an 8 bit sample sequence we can represent the twiddle factor as a vector in the unit circle. e.g.



Note that

1 It is periodic. (i.e. it goes round and round the circle !!)
2 That the vectors are symmetric
3 The vectors are equally spaced around the circle.

Why the FFT?

If you look at the equation for the Discrete Fourier Transform you will see that it is quite complicated to work out as it involves many additions and multiplications involving complex numbers. Even a simple eight sample signal would require 49 complex multiplications and 56 complex additions to work out the DFT. At this level it is still manageable; however a realistic signal could have 1024 samples which requires over 20,000,000 complex multiplications and additions. As you can see the number of calculations required soon mounts up to unmanageable proportions.

The Fast Fourier Transform is a simply a method of laying out the computation, which is much faster for large values of N, where N is the number of samples in the sequence. It is an ingenious way of achieving rather than the DFT's clumsy P^2 timing.

The idea behind the FFT is the divide and conquer approach, to break up the original N point sample into two (N / 2) sequences. This is because a series of smaller problems is easier to solve than one large one. The DFT requires (N1)^2 complex multiplications and N(N1) complex additions as opposed to the FFT's approach of breaking it down into a series of 2 point samples which only require 1 multiplication and 2 additions and the recombination of the points which is minimal.

For example Seismic Data contains hundreds of thousands of samples and would take months to evaluate the DFT. Therefore we use the FFT.

FFT Algorithm

The FFT has a fairly easy algorithm to implement, and it is shown step by step in the list below. Thjis version of the FFT is the Decimation in Time Method

1. Pad input sequence, of N samples, with ZERO's until the number of samples is the nearest power of two.

   e.g. 500 samples are padded to 512 (2^9)

2. Bit reverses the input sequence.

   e.g. 3 = 011 goes to 110 = 6

3. Compute (N / 2) two sample DFT's from the shuffled inputs.

   See "Shuffled Inputs"

4. Compute (N / 4) four sample DFT's from the two sample DFT's.

   See "Shuffled Inputs"

5. Compute (N / 2) eight sample DFT's from the four sample DFT's.

See "Shuffled Inputs"

6. Until the all the samples combine into one N sample DFT

Shuffled Inputs

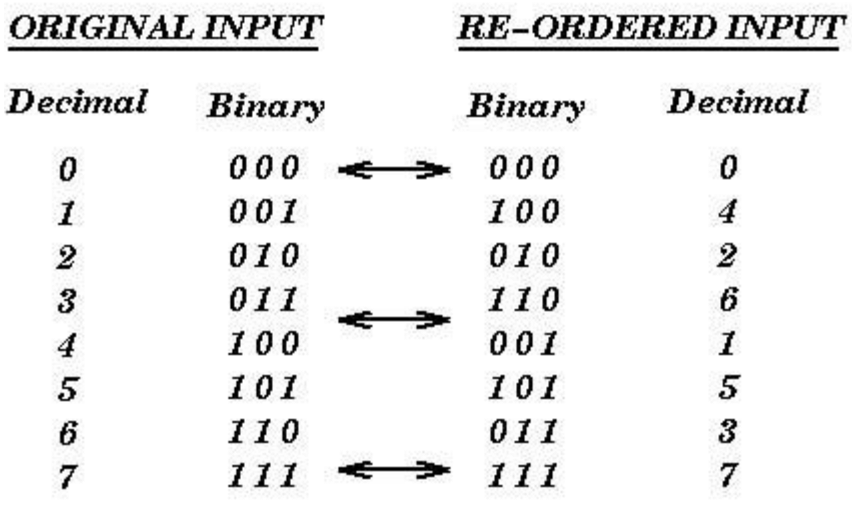
The process of decimating the signal in the time domain has caused the INPUT samples to be reordered. For an 8 point signal the original order of the samples is

0, 1, 2, 3, 4, 5, 6, 7

But after decimation the order is

0, 4, 2, 6, 1, 5, 3, 7

At first it may look as if there is no order to this new sequence, BUT if the numbers are



ORIGINAL INPUT			RE-ORDERED INPUT	
Decimal	Binary		Binary	Decimal
0	000	↔	000	0
1	001		100	4
2	010		010	2
3	011	↔	110	6
4	100		001	1
5	101		101	5
6	110		011	3
7	111	↔	111	7

What has happened is that the bit patterns representing the sample number has been reversed. This new sequence is the order that the samples enter the FFT.

C. PROGRAM TO IMPLEMENT FFT.

```
#include <math.h>
#define PTS 64          //# of points for FFT
#define PI 3.14159265358979
typedef struct {float real,imag;} COMPLEX;
void FFT(COMPLEX *Y, int n);      //FFT prototype
float iobuffer[PTS]; //as input and output buffer
float x1[PTS];          //intermediate buffer
short i;          //general purpose index variable
short buffercount=0;          //number of new samples in iobuffer
short flag = 0;            //set to 1 by ISR when iobuffer full
COMPLEX w[PTS];            //twiddle constants stored in w
COMPLEX samples[PTS];            //primary working buffer
main()
{
for (i = 0 ; i<PTS ; i++)          // set up twiddle constants in w
{
w[i].real = cos(2*PI*i/(PTS*2.0));      //Re component of twiddle
constants
w[i].imag =-sin(2*PI*i/(PTS*2.0));        //Im component of twiddle
constants
}
for (i = 0 ; i < PTS ; i++) //swap buffers
{
iobuffer[i] = sin(2*PI*10*i/64.0);        //*10- > freq, 64 -> sampling
freq*/
samples[i].real=0.0;
samples[i].imag=0.0;
}
for (i = 0 ; i < PTS ; i++) //swap buffers
{
samples[i].real=iobuffer[i]; //buffer with new data
}
for (i = 0 ; i < PTS ; i++)
samples[i].imag = 0.0; //imag components = 0
FFT(samples,PTS); //call function FFT.c
for (i = 0 ; i < PTS ; i++) //compute magnitude
{
x1[i]=sqrt(samples[i].real*samples[i].real +
samples[i].imag*samples[i].imag);
}
} //end of main

//FFT FUNCTION
void FFT(COMPLEX *Y, int N) //input sample array, # of points
{
```

```
int upper_leg, lower_leg; //index of upper/lower butterfly leg
int leg_diff; //difference between upper/lower leg
int num_stages = 0; //number of FFT stages (iterations)
int index, step; //index/step through twiddle constant
i = 1; //log(base2) of N points= # of stages
do
{
num_stages +=1;

i = i*2;
}while (i!=N);
leg_diff = N/2;  //difference between upper&lower legs
step = (PTS*2)/N;  //step between values in twiddle.h
for (i = 0;i < num_stages; i++)  //for N-point FFT
{
index = 0;
for (j = 0; j < leg_diff; j++)
{
for (upper_leg = j; upper_leg < N; upper_leg += (2*leg_diff))
{
lower_leg = upper_leg+leg_diff;
temp1.real = (Y[upper_leg]).real + (Y[lower_leg]).real;
temp1.imag = (Y[upper_leg]).imag + (Y[lower_leg]).imag;
temp2.real = (Y[upper_leg]).real - (Y[lower_leg]).real;
temp2.imag = (Y[upper_leg]).imag - (Y[lower_leg]).imag;
(Y[lower_leg]).real = temp2.real*(w[index]).real
-temp2.imag*(w[index]).imag

(Y[lower_leg]).imag = temp2.real*(w[index]).imag

+temp2.imag*(w[index]).real;
(Y[upper_leg]).real = temp1.real;
(Y[upper_leg]).imag = temp1.imag;
}
index += step;
}
leg_diff = leg_diff/2;
step *= 2;
}
j = 0;
for (i = 1; i < (N-1); i++)   //bit reversal for resequencing data
{
k = N/2;
while (k <= j)
{
j = j - k;
k = k/2;
}
j = j + k;
if (i<j)
{
temp1.real = (Y[j]).real;
temp1.imag = (Y[j]).imag;
(Y[j]).real = (Y[i]).real;
(Y[j]).imag = (Y[i]).imag;
(Y[i]).real = temp1.real;
(Y[i]).imag = temp1.imag;
}}
return;
}
```

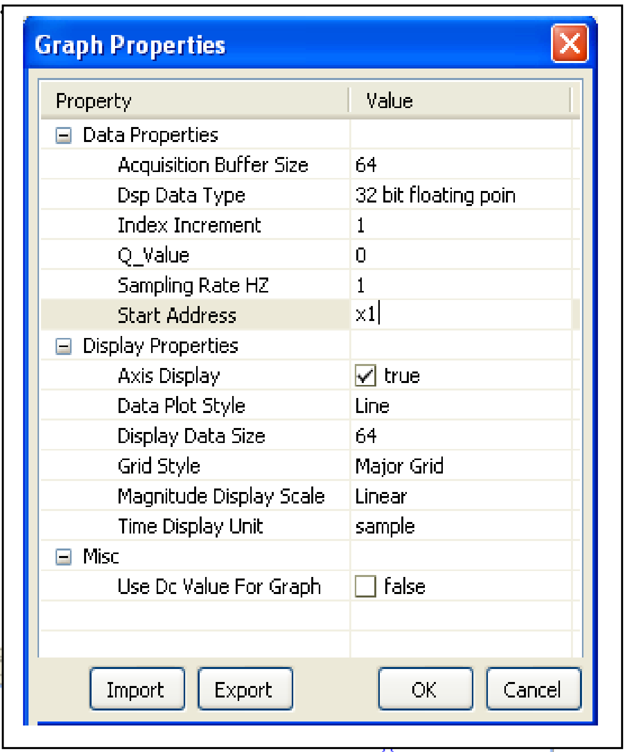
Procedure:

Follow steps 1-7.

To view results goto view->watch.

In watch window type the variable name to which you want to view results.

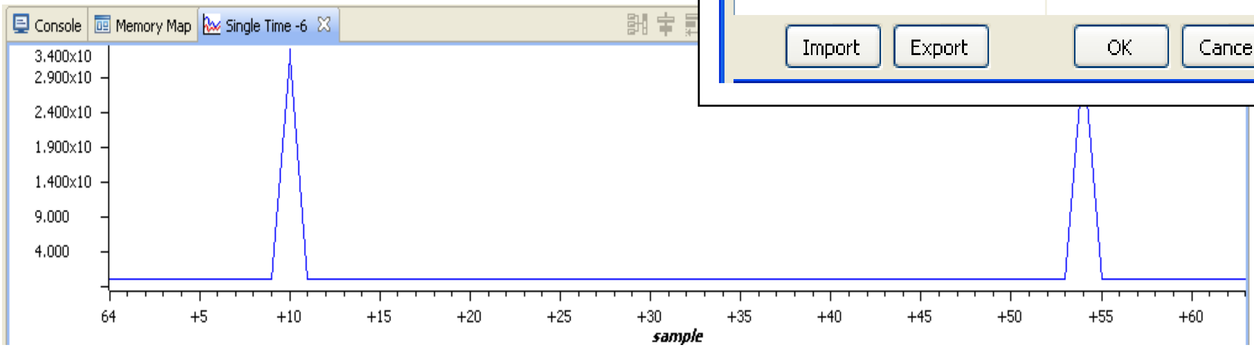
1. Set the graph properties as shown in figure.



Graph Properties

Property	Value
Data Properties	
Acquisition Buffer Size	64
Dsp Data Type	32 bit floating poin
Index Increment	1
Q_Value	0
Sampling Rate HZ	1
Start Address	x1
Display Properties	
Axis Display	✓ true
Data Plot Style	Line
Display Data Size	64
Grid Style	Major Grid
Magnitude Display Scale	Linear
Time Display Unit	sample
Misc	
Use Dc Value For Graph	false

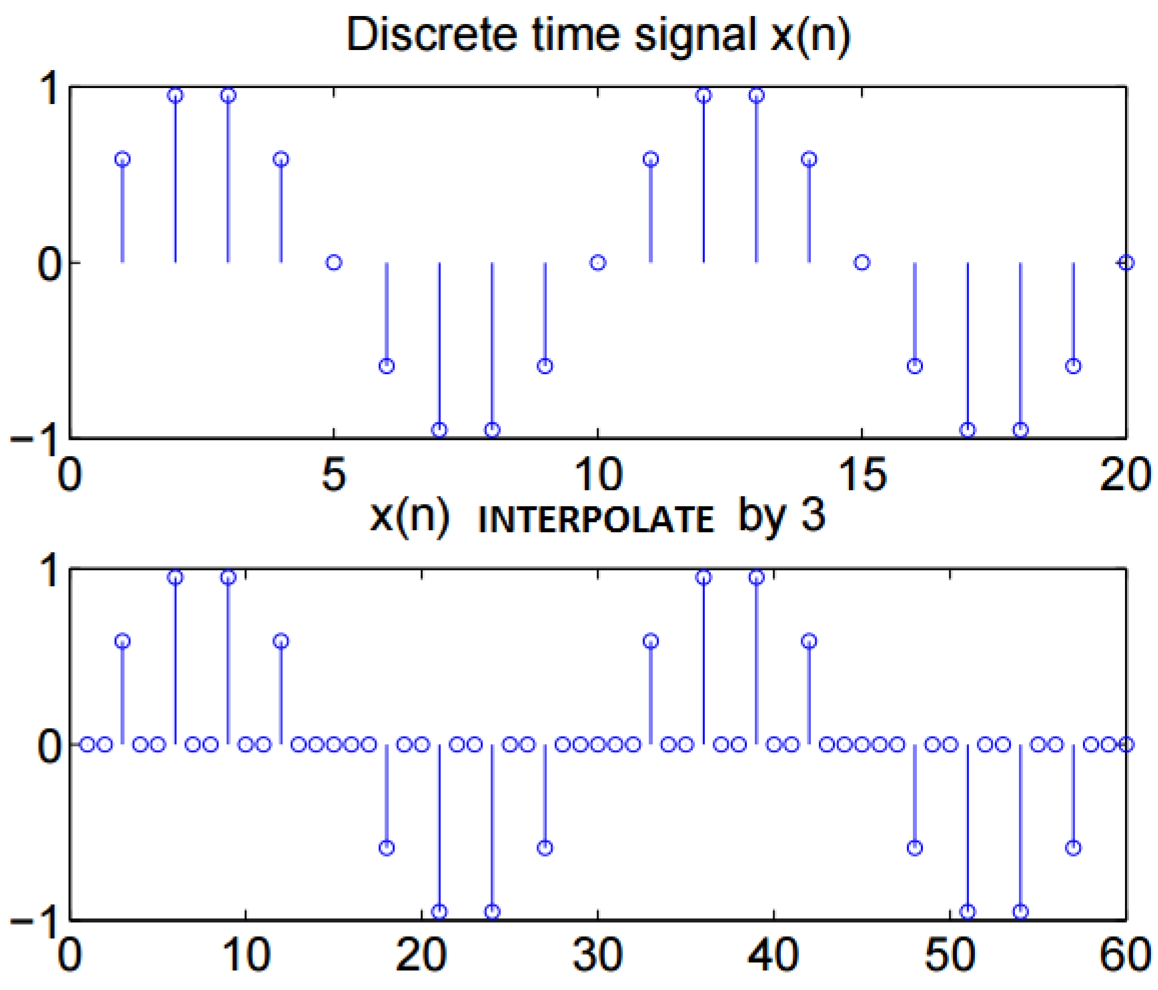
The graph has been plotted as shown below.



INTERPOLATION AND DECIMATION OF SINE WAVE

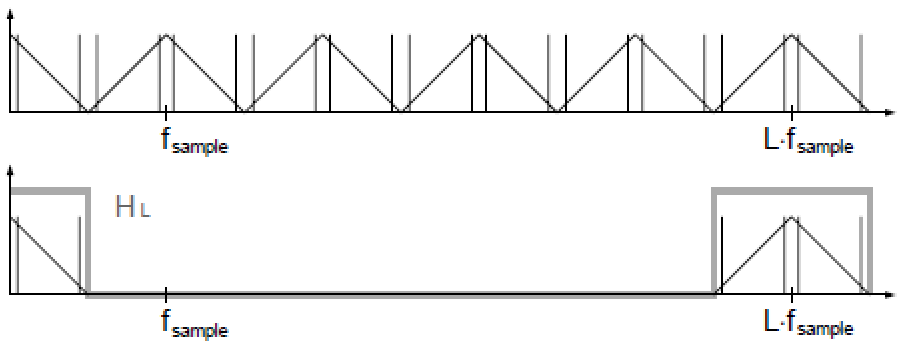
Interpolation

Increase the sampling rate of a discrete-time signal. Higher sampling rate preserves fidelity**.**



The primary reason to interpolate is simply to increase the sampling rate at the output of one system so that another system operating at a higher sampling rate can input the signal.

In this section the increase of the sample rate by an integer factor L is described. In the following we refer to this process as interpolation.

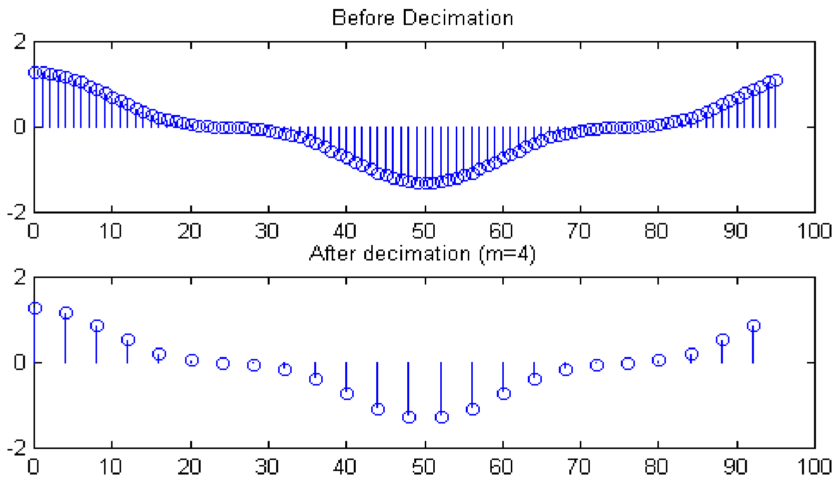


If we increase the sample rate of a signal we can preserve the full signal content according to the sampling theorem. After the interpolation the signal spectrum repeats only at multiples of the new sample rate L. fs. The interpolation is accomplished by inserting L-1 zeros between successive samples (zero-padding) and using an (ideal) lowpass filter with

$$H_L(f) = \begin{cases} L & \text{for } 0 < f < \frac{f_{sample}}{2} \\ 0 & \text{for } \frac{f_{sample}}{2} < f < \frac{L \cdot f_{sample}}{2} \end{cases}$$

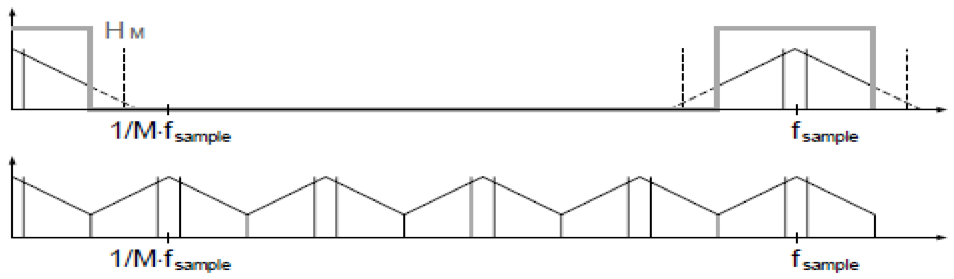
Decimation

Reduce the sampling rate of a discrete-time signal. Low sampling rate reduces storage and computation requirements.



The most immediate reason to decimate is simply to reduce the sampling rate at the output of one system so a system operating at a lower sampling rate can input the signal. But a much more common motivation for decimation is to reduce the*cost* of processing:
the *calculation* and/or *memory* required to implement a DSP system generally is proportional to the sampling rate, so the use of a lower sampling rate usually results in a cheaper implementation

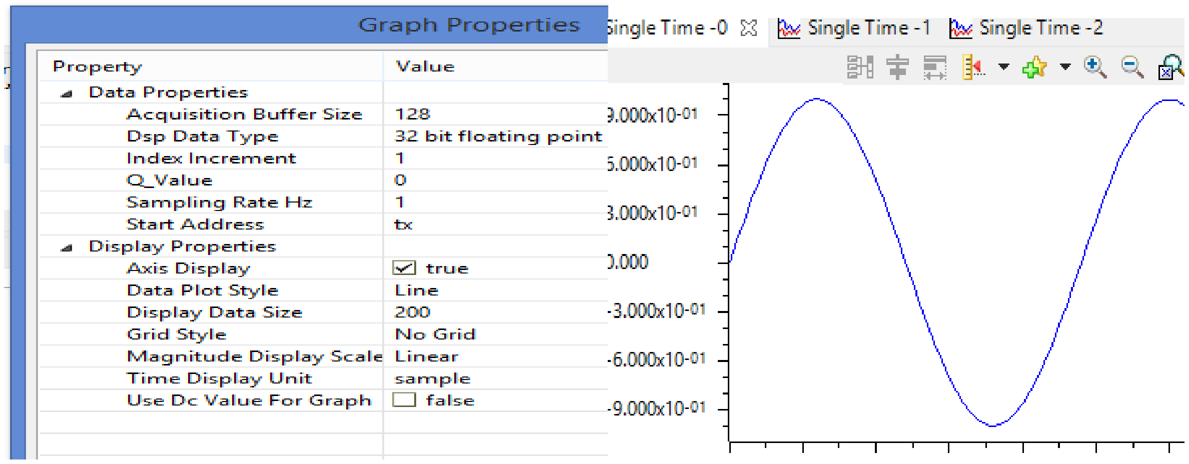
To decrease the sample rate by an integer factor M (decimation) we must first band-limit the signal to fs/(2.M) by the lowpass filter Hm to comply with sampling theorem and keep only every Mth sample. As a result, we loose all signal content above half the target sampling frequency fs/M.



$$H_M(f) = \begin{cases} 1 & \text{for } 0 < f < \frac{f_{sample}}{2\cdot M} \\ 0 & \text{for } \frac{f_{sample}}{2\cdot M} < f < \frac{f_{sample}}{2} \end{cases}$$

```
//interpolation & decimation .c

#include<stdio.h>
#include<math.h>
#define FREQ 500
int tb=2; //sampling rate multiplier for interpolated signal
float tx[128]; //to store sine(input) wave Signal
float ty[256]; // to store Interpolated signal
float x[256],y[256]; //to store decemated Signal
void main()
{
      int txx,ti,tc,td;
      int a,b,d;
      int i,j,z,g;
      int tj,tcc,tz;
      int ta=128;

      for(i=0;i<127;i++)
 {
  tx[i]=sin(2*3.14*FREQ*i/24000); //generating sine wave
  printf("%f\n",tx[i]);
 }
//Interpolation
tc = tb - 1;
            txx = 0;
            for (ti=1;ti<=ta;ti++)
            {
                  ty[ti+txx] = tx[ti];
                  tcc = ti+txx;

                  tz = ti;
                  for (tj = 1 ; tj<=tc ;tj++)
                  {
            ty[tcc+1] = 0; //adding zeros in between samples
                        ti = ti+1;
                        tcc = tcc+1;
                  }
                  txx = tcc-tz;
                  ti = ti-tc;
            }
            td = ta*tb; //Length of interpolated signal
            for(ti=1;ti<=td;ti++)
            {
      printf("\n The Value of output ty[%d]=%f",ti,ty[ti]);
}
//Decimation
b=2; //Sampling rate divider for decimated signal
             j=1;
            for (g=1;g<=128;g++)
               {
               y[g] = ty[j];
               j = j+b;
               printf("%f\n",y[g]);
                           }
}
```

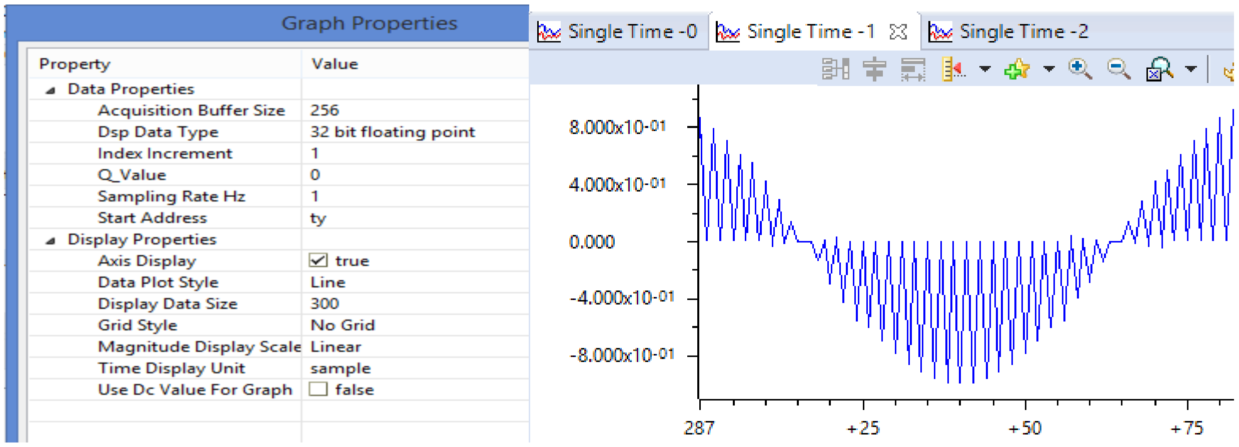
INPUT SINE WAVE



Graph Properties

Property	Value
◢ Data Properties	
Acquisition Buffer Size	128
Dsp Data Type	32 bit floating point
Index Increment	1
Q_Value	0
Sampling Rate Hz	1
Start Address	tx
◢ Display Properties	
Axis Display	✓ true
Data Plot Style	Line
Display Data Size	200
Grid Style	No Grid
Magnitude Display Scale	Linear
Time Display Unit	sample
Use Dc Value For Graph	☐ false

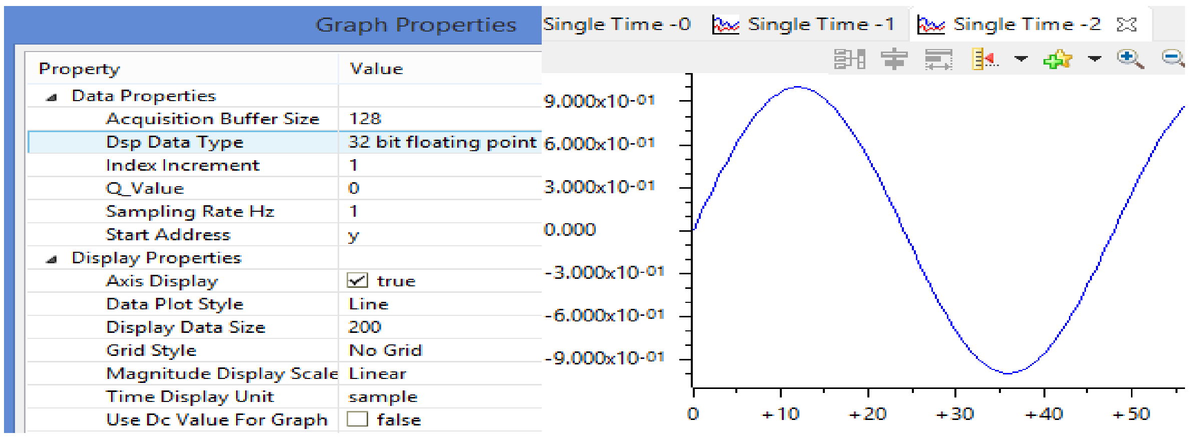
INTERPOLATION OUTPUT



Graph Properties

Property	Value
◢ Data Properties	
Acquisition Buffer Size	256
Dsp Data Type	32 bit floating point
Index Increment	1
Q_Value	0
Sampling Rate Hz	1
Start Address	ty
◢ Display Properties	
Axis Display	✓ true
Data Plot Style	Line
Display Data Size	300
Grid Style	No Grid
Magnitude Display Scale	Linear
Time Display Unit	sample
Use Dc Value For Graph	☐ false

DECEMATION OUTPUT

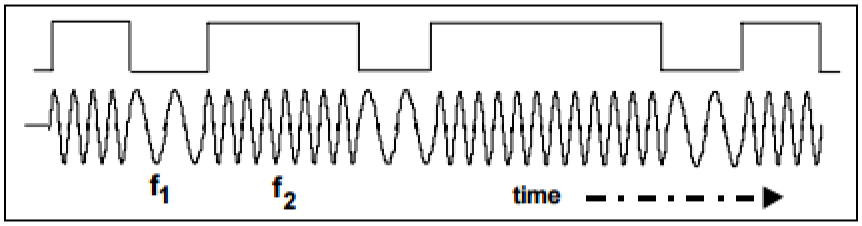


Graph Properties

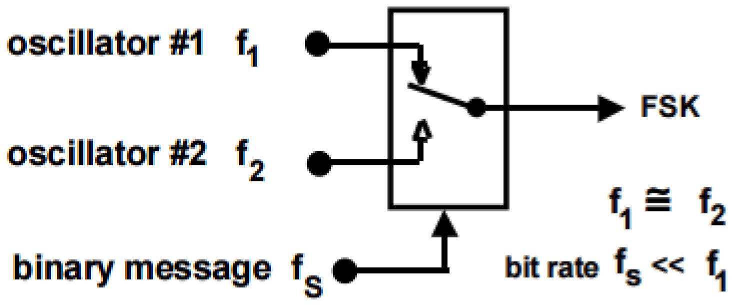
Property	Value
◢ Data Properties	
Acquisition Buffer Size	128
Dsp Data Type	32 bit floating point
Index Increment	1
Q_Value	0
Sampling Rate Hz	1
Start Address	y
◢ Display Properties	
Axis Display	✓ true
Data Plot Style	Line
Display Data Size	200
Grid Style	No Grid
Magnitude Display Scale	Linear
Time Display Unit	sample
Use Dc Value For Graph	☐ false

FSK

As its name suggests, a frequency shift keyed transmitter has its frequency shifted by the message. Although there could be more than two frequencies involved in an FSK signal, in this experiment the message will be a binary bit stream, and so only two frequencies will be involved. The word 'keyed' suggests that the message is of the 'on-off' (mark-space) variety, such as one (historically) generated by a morse key, or more likely in the present context, a binary sequence. The output from such a generator is illustrated in Figure 1 below



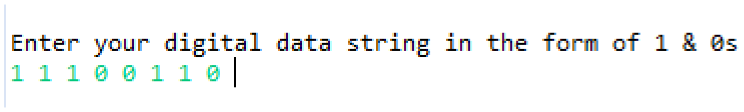
Conceptually, and in fact, the transmitter could consist of two oscillators (on frequencies f1 and f2), with only one being connected to the output at any one time. This is shown in block diagram form in Figure 2 below.



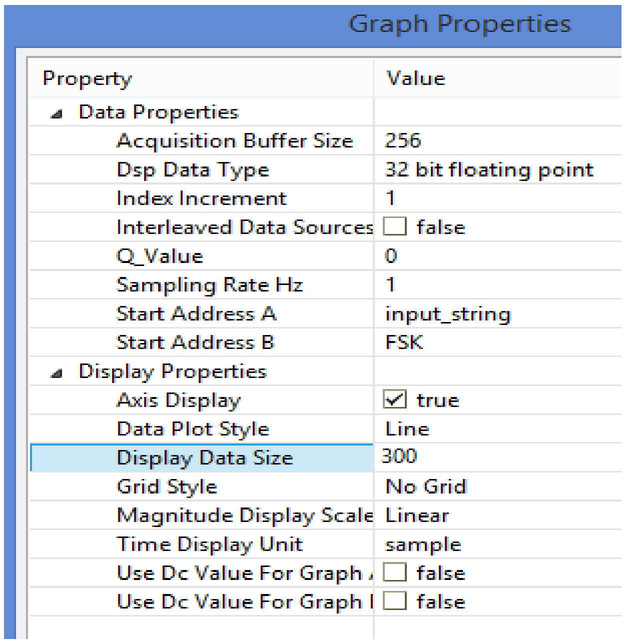
Unless there are special relationships between the two oscillator frequencies and the bit clock there will be abrupt phase discontinuities of the output waveform during transitions of the message.

```c
#include<stdio.h>
#include<math.h>
#define TIME 336 //length of FSK signal
#define PI 3.14


float fh[TIME],fl[TIME],FSK[TIME];
int input_string[8],scale_data[TIME];

void main()
{

      int i,j,k,l;
      printf("\nEnter your digital data string in the form of 1 & 0s\n");
      for(i=0;i<8;i++) //data in binary format
      scanf("%d",&input_string[i]);
            k=0;
      for(k=0;k<TIME;k++)
      {
            for(j=0;j<8;j++)
            {
                  for(i=0;i<21;i++)
                  {
            scale_data[k]=input_string[j]; // Scaling input data
                        k++;
                  }
            }
      }
      for(i=0;i<TIME;i++)
      {
            fh[i]=sin(2*PI*2000*i/10000);//high frequency
            fl[i]=sin(2*PI*1000*i/10000);//low frequency
      }

//assigning high frequency to bit 1
      k=0;
      for(l=0;k<TIME;l++)
      {
            for(i=0;i<8;i++)
            {
                  if(input_string[i]==1)
                  {
                        for(j=0;j<21;j++)
                        {
                              FSK[k]=fh[j];
                              k++;
                        }
                  }
                  else
//assigning low frequency to bit 0

                  {
                        if(input_string[i]==0)
                        {
                              for(j=0;j<21;j++)
                              {
                                    FSK[k]=fl[j];
                                    k++;
                              }}}}}}
```

INPUT SEQUENCE



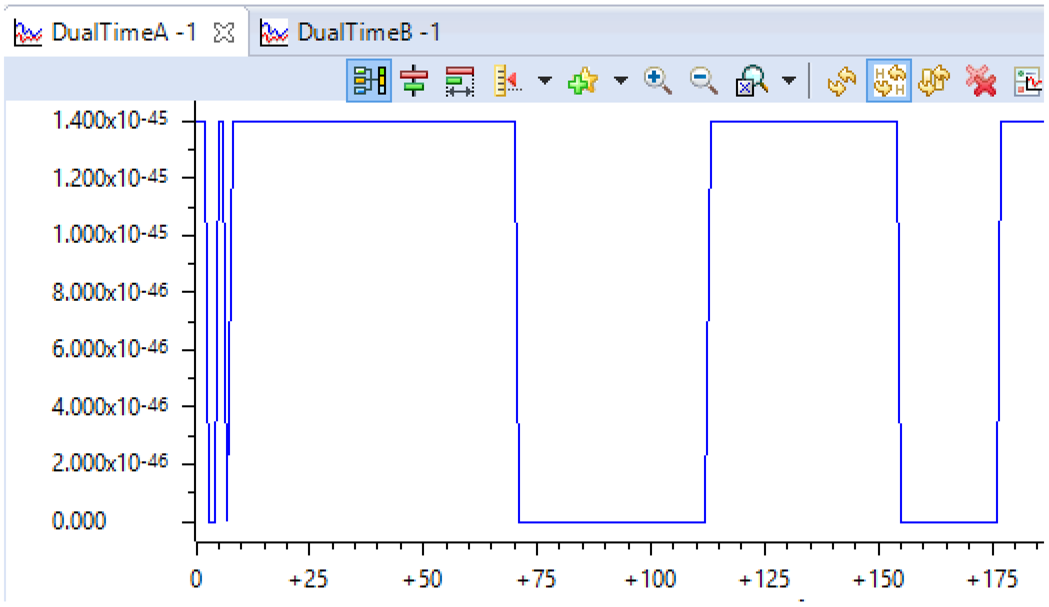
```
Enter your digital data string in the form of 1 & 0s
1 1 1 0 0 1 1 0
```



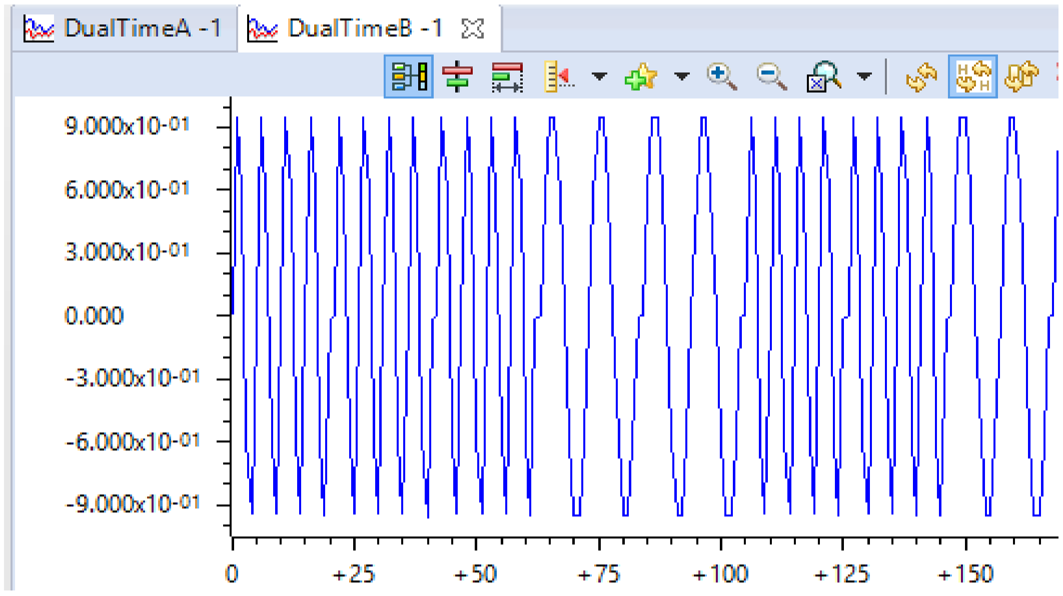
Graph Properties

Property	Value
◢ Data Properties	
Acquisition Buffer Size	256
Dsp Data Type	32 bit floating point
Index Increment	1
Interleaved Data Sources	☐ false
Q_Value	0
Sampling Rate Hz	1
Start Address A	input_string
Start Address B	FSK
◢ Display Properties	
Axis Display	☑ true
Data Plot Style	Line
Display Data Size	300
Grid Style	No Grid
Magnitude Display Scale	Linear
Time Display Unit	sample
Use Dc Value For Graph /	☐ false
Use Dc Value For Graph I	☐ false

INPUT SEQUENCE



FSK WAVE



CACULATION OF POWER AND ENERGY OF SAMPLES

Energy of Samples

The **energy** of a discrete-time signal is defined as

$$E_x \triangleq \sum_{n=-\infty}^{\infty} |x[n]|^2 .$$

```
#include<stdio.h>
int main()
{
  int num,i,j,x[32];
  long int sum=0;
  printf("\nEnter the number of samples: ");
  scanf("%d",&num);
  printf("\nEnter samples: ");
  for(j=0;j<num;j++)
     scanf("%d",&x[j]);
  for(i=0;i<=num;i++)
  {
            sum+=x[i]*x[i];

  }
  printf("\n the energy of above samples is\n %d",sum);
  return 0;
}
```

Power of Samples

The **average power** of a signal is defined as

$$P_x \triangleq \lim_{N\to\infty} \frac{1}{2N+1} \sum_{n=-N}^{N} |x[n]|^2$$

```
#include<stdio.h>
int main(){
  int num,i,j,x[32];
            float  num1;
  long int sum=0;
  printf("\nEnter the number of samples: ");
  scanf("%d",&num);
  printf("\nEnter samples: ");
  for(j=0;j<num;j++)
     scanf("%d",&x[j]);
  for(i=0;i<=num;i++)
  {
            sum+=x[i]*x[i];
  }
  num=num*2;
  num++;
  num1 = sum / (float) num;
  printf("\n the Average power of above samples is\n %.2f",num1);
  return 0;
}
```

POWER SPECTRUM

The total or the average power in a signal is often not of as great an interest. We are most often interested in the PSD or the Power Spectrum. We often want to see is how the input power has been redistributed by the channel and in this frequency-based redistribution of power is where most of the interesting information lies. The total area under the Power Spectrum or PSD is equal to the total avg. power of the signal. The PSD is an even function of frequency or in other words.

To Compute PSD:

The value of the auto-correlation function at zero-time equals the total power in the signal. To compute PSD We compute the auto-correlation of the signal and then take its FFT. The auto-correlation function and PSD are a Fourier transform pair. (Another estimation method called "period gram" uses sampled FFT to compute the PSD.)

E.g.: For a process *x(n)* correlation is defined as:

$$\boxed{\begin{aligned} R(\tau) &= E\{x(n)x(n+\tau)\} \\ &= \lim_{N\to\infty}\frac{1}{N}\sum_{n=1}^{N} x(n)x(n+\tau) \end{aligned}}$$

Power Spectral Density is a Fourier transform of the auto correlation.

$$\boxed{\begin{aligned} S(\omega) &= FT(R(\tau)) = \lim_{N\to\infty}\sum_{\tau=-N+1}^{N-1}\hat{R}(\tau)e^{-j\omega\tau} \\ R(\tau) &= FT^{-1}(S(\omega)) = \frac{1}{2\pi}\int S(\omega)e^{j\omega\tau}d\omega \end{aligned}}$$

ALGORITHM TO IMPLEMENT PSD:

- **Step 1 -** Select no. of points for FFT(Eg: 64)
- **Step 2 –** Generate a sine wave of frequency 'f '(eg: 10 Hz with a sampling rate = No. of Points of FFT (eg. 64)) using **math library function**.
- **Step 3 -** Compute the Auto Correlation of Sine wave

- **Step4** - Take output of auto correlation, apply FFT algorithm.
- **Step 4 -** Use Graph option to view the PSD.
- **Step 5 - Repeat Step-1 to 4** for different no. of points & frequencies.

‘C’ Program to Implement PSD:

```
PSD.c:

/**************************************************************
*

* FILENAME

* Non_real_time_PSD.c

* DESCRIPTION

* Program to Compute Non real time PSD

* using the TMS320C6711 DSK.

**************************************************************
**

* DESCRIPTION

* Number of points for FFT (PTS)

* x --> Sine Wave Co-Efficients

* iobuffer --> Output of Auto Correlation.

* x1 --> use in graph window to view PSD

/*============================================================
*/

#include <math.h>

#define PTS 128        //# of points for FFT

#define PI 3.14159265358979

typedef struct {float real,imag;} COMPLEX;

void FFT(COMPLEX *Y, int n);    //FFT prototype

float iobuffer[PTS];       //as input and output buffer

float x1[PTS],x[PTS];             //intermediate buffer

short i;                                  //general purpose
index variable

short buffercount = 0;    //number of new samples in iobuffer
short flag = 0;     //set to 1 by ISR when iobuffer full

float y[128];

COMPLEX w[PTS];       //twiddle constants stored in w

COMPLEX samples[PTS];     //primary working buffer

main()

{

float j,sum=0.0 ;

int n,k,i,a;

for (i = 0 ; i<PTS ; i++)     // set up twiddle constants in w

{

w[i].real = cos(2*PI*i/(PTS*2.0));

/*Re component of twiddle constants*/

w[i].imag =-sin(2*PI*i/(PTS*2.0));

/*Im component of twiddle constants*/

}

/****************Input Signal X(n) *************************/

for(i=0,j=0;i<PTS;i++)

{ x[i] = sin(2*PI*5*i/PTS);  // Signal x(Fs)=sin(2*pi*f*i/Fs);

samples[i].real=0.0;

samples[i].imag=0.0;

}

/********************Auto Correlation of X(n)=R(t) ***********/

for(n=0;n<PTS;n++)

{

sum=0;

for(k=0;k<PTS-n;k++)

{

sum=sum+(x[k]*x[n+k]);      // Auto Correlation R(t)

}

iobuffer[n] = sum;

}
```

```
/******************** FFT of R (t) ***********************/

for (i = 0 ; i < PTS ; i++)        //swap buffers

{

samples[i].real=iobuffer[i];    //buffer with new data

}

for (i = 0 ; i < PTS ; i++)

samples[i].imag = 0.0;          //imag components = 0

FFT(samples,PTS);               //call function FFT.c

/******************** PSD ********************/

for (i = 0 ; i < PTS ; i++)        //compute magnitude

{

x1[i] = sqrt(samples[i].real*samples[i].real samples[i].imag*samples[i].imag);

}

}

 //end of main

/*FFT*/

#define PTS 128                 //# of points for FFT

typedef struct {float real,imag;} COMPLEX;

extern COMPLEX w[PTS];                  //twiddle constants stored in w*/

void FFT(COMPLEX *Y, int N)              //input sample array, # of points

{

COMPLEX temp1,temp2;     //temporary storage variables

int i,j,k; //loop counter variables

int upper_leg, lower_leg;             //index of upper/lower butterfly leg

int leg_diff;                  //difference between upper/lower leg

int num_stages = 0;    //number of FFT stages (iterations)
int index, step;     //index/step through twiddle constant
i = 1;       //log (base2) of N points= # of stages
do
{
num_stages +=1;
i = i*2;
}while (i!=N);
leg_diff = N/2;      //difference between upper&lower legs
step = (PTS*2)/N;    //step between values in twiddle.h// 512
for (i = 0;i < num_stages; i++)       //for N-point FFT
{
index = 0;
for (j = 0; j < leg_diff; j++)
{
for (upper_leg = j; upper_leg < N; upper_leg += (2*leg_diff))
{
lower_leg = upper_leg+leg_diff;
temp1.real = (Y[upper_leg]).real + (Y[lower_leg]).real;
temp1.imag = (Y[upper_leg]).imag + (Y[lower_leg]).imag;
temp2.real = (Y[upper_leg]).real - (Y[lower_leg]).real;
temp2.imag = (Y[upper_leg]).imag - (Y[lower_leg]).imag;
(Y[lower_leg]).real = temp2.real*(w[index]).real
-temp2.imag*(w[index]).imag;
(Y[lower_leg]).imag = temp2.real*(w[index]).imag
+temp2.imag*(w[index]).real;
(Y[upper_leg]).real = temp1.real;
(Y[upper_leg]).imag = temp1.imag;}
index += step;
}
leg_diff = leg_diff/2;

step *= 2;}

j = 0;

for (i = 1; i < (N-1); i++)

//bit reversal for re sequencing data

{k = N/2;

while (k <= j)

{

j = j - k;

k = k/2;

}

j = j + k;

if (i<j)

{

temp1.real = (Y[j]).real;

temp1.imag = (Y[j]).imag;

(Y[j]).real = (Y[i]).real;

(Y[j]).imag = (Y[i]).imag;

(Y[i]).real = temp1.real;

(Y[i]).imag = temp1.imag;

}}

return;

}
```

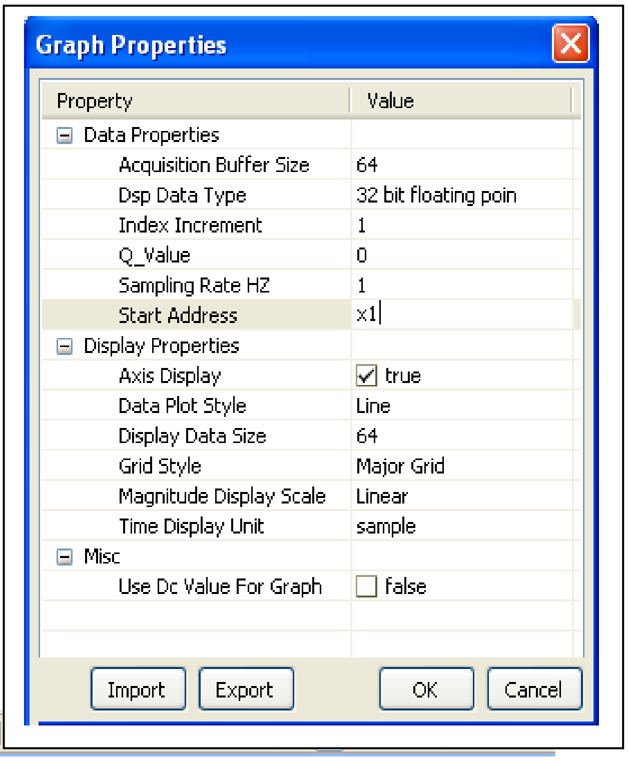
Procedure:

1. Follow steps 1-7.
2. Run the program to see the results.

   To view results goto view->watch.

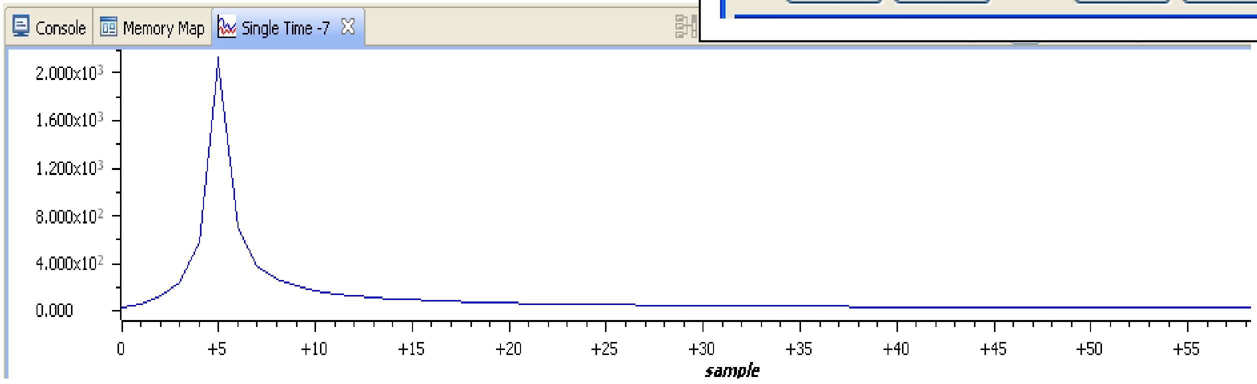
   In watch window type the variable name to which you want to view results.
3. Set the graph properties as shown in figure. To view the result use graph utility

Set the graph properties as shown in figure.



Property	Value
⊟ Data Properties	
Acquisition Buffer Size	64
Dsp Data Type	32 bit floating poin
Index Increment	1
Q_Value	0
Sampling Rate HZ	1
Start Address	x1
⊟ Display Properties	
Axis Display	✓ true
Data Plot Style	Line
Display Data Size	64
Grid Style	Major Grid
Magnitude Display Scale	Linear
Time Display Unit	sample
⊟ Misc	
Use Dc Value For Graph	☐ false

The graph has been plotted as shown below.

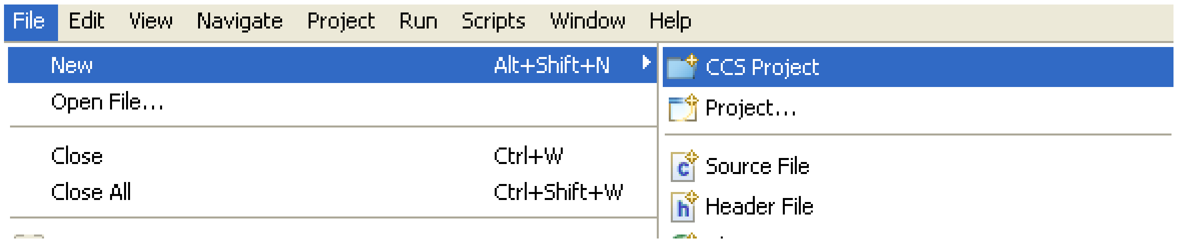


Procedure to work with NON REAL TIME USING SIMULATOR MODE IN V6:

- **TO MAKE CCS V6 TO WORK IN SIMULATOR MODE PLEASE GO THROUGH APPENDIX**

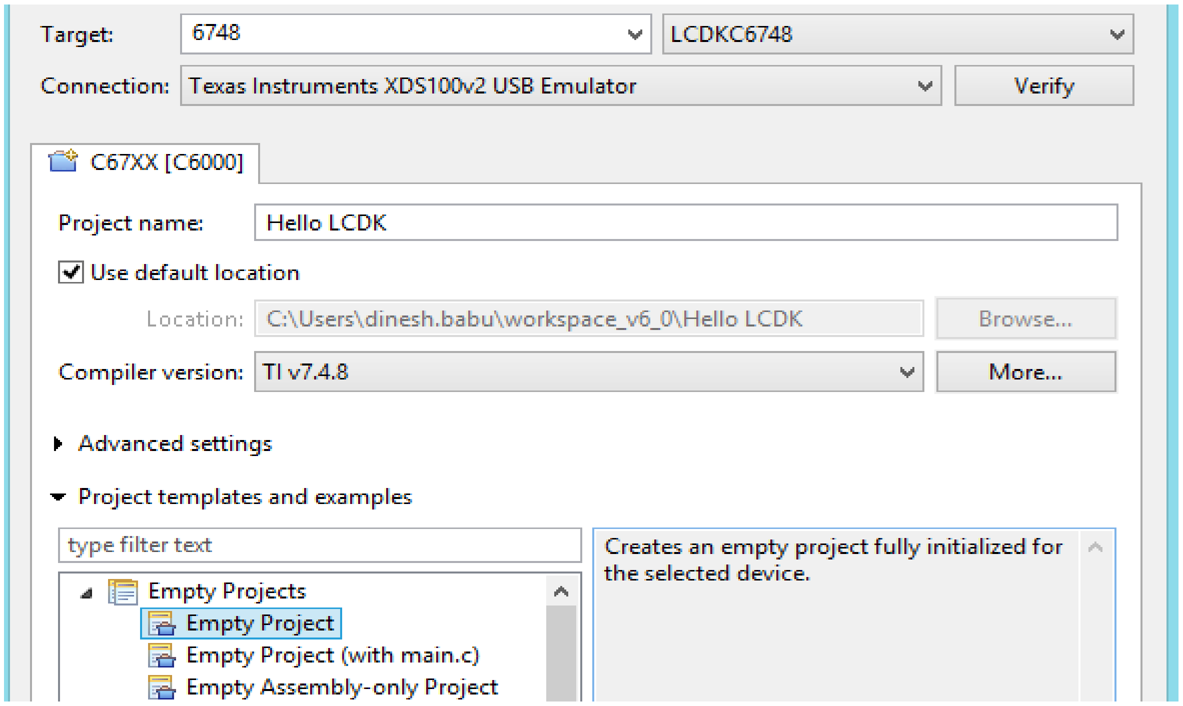
Creating a New Project:

- Go to File → New → CCS Project.



- Specify the name of the project in the space provided e g., Project Name: Hello LCDK.

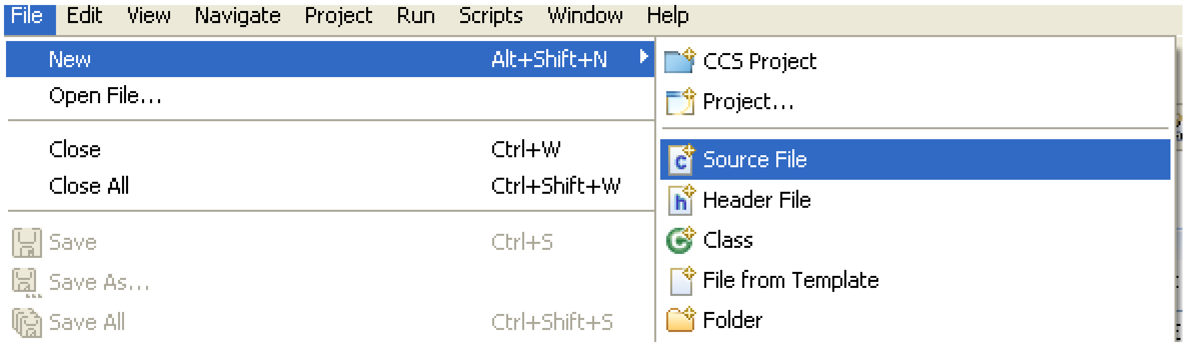
Specify the "Device" properties as shown in the figure below and select an "Empty Project".


Target: 6748 LCDKC6748
Connection: Texas Instruments XDS100v2 USB Emulator
C67XX [C6000]
Project name: Hello LCDK
Use default location ✓
Location: C:\Users\dinesh.babu\workspace_v6_0\Hello LCDK
Compiler version: TI v7.4.8
Creates an empty project fully initialized for the selected device.
Verify
Browse...
More...
Advanced settings
Project templates and examples
type filter text
Empty Projects
Empty Project
Empty Project (with main.c)
Empty Assembly-only Project

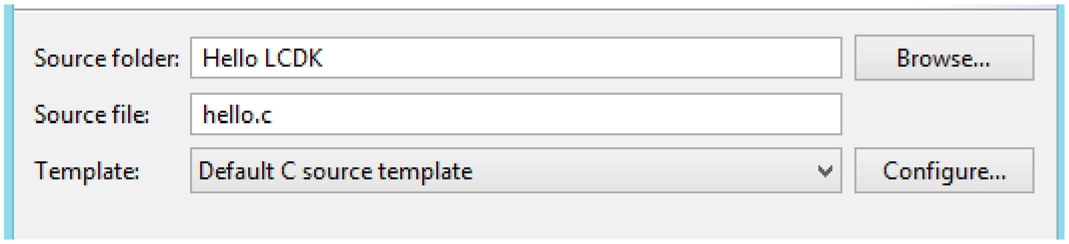
Click **Finish**

NR4. To write a program, select one new source file. (Do either A or B)

A. Go to File → New → Source File.



➢ Specify the arbitrary source file name with ".c" extension. It should be in the source folder (current project name).

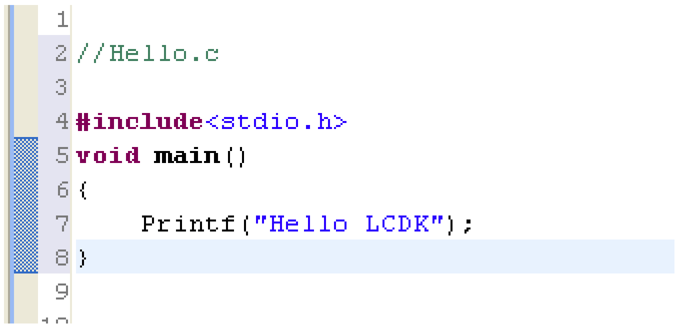


Source folder: Hello LCDK

Source file: hello.c

Template: Default C source template

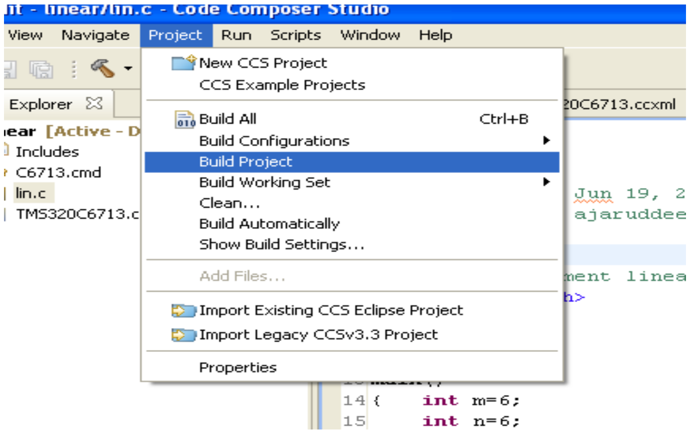
➢ Type your C Code and save.



```

//Hello.c

#include<stdio.h>
void main()
{
    Printf("Hello LCDK");
}

```

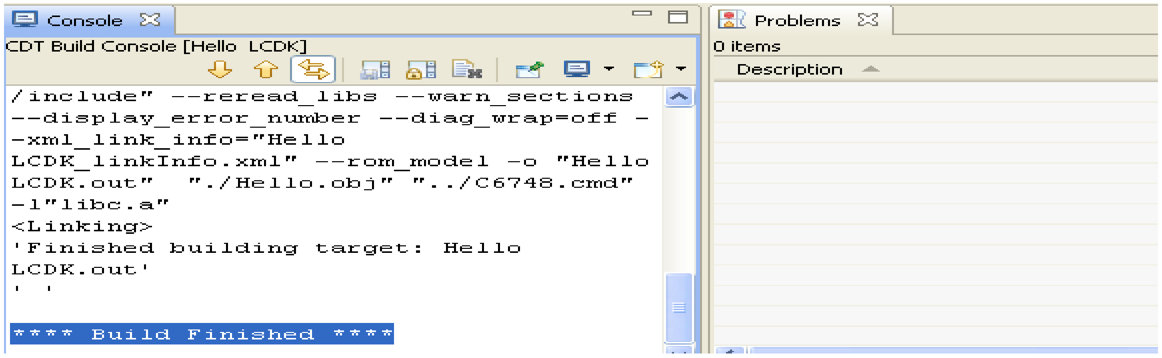
NR3: <u>Build the project</u>

Go to **Project→Build Project** as shown in Fig 11below.



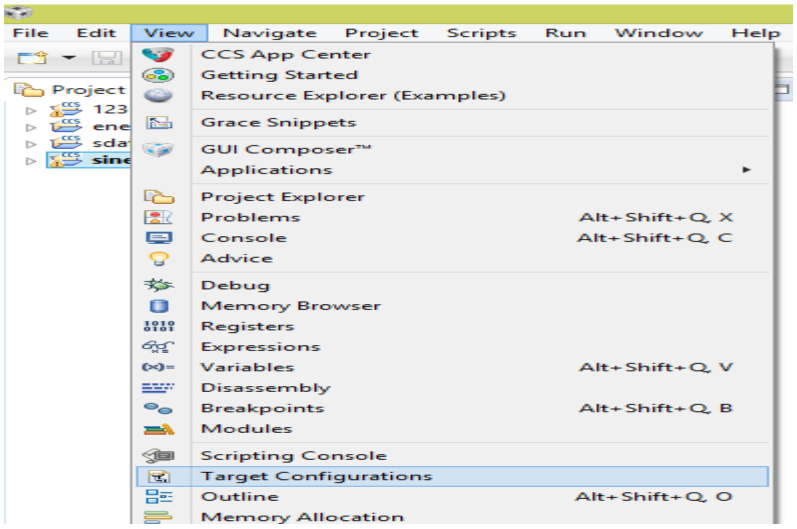
If your code doesn't have any errors and warnings, a message will be printed in the console window that "**** Build Finished ****"

Problems window display errors or warnings, if any.

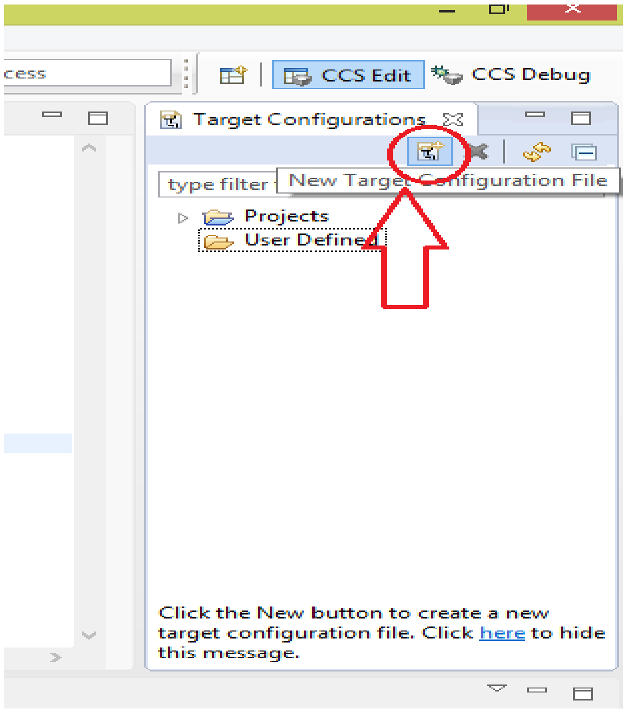


```
/include" --reread_libs --warn_sections
--display_error_number --diag_wrap=off -
-xml_link_info="Hello
LCDK_linkInfo.xml" --rom_model -o "Hello
LCDK.out"  "./Hello.obj" "../C6748.cmd"
-l"libc.a"
<Linking>
'Finished building target: Hello
LCDK.out'
' '

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

NR4: <u>Target Configuration</u>



A new window on right side will appear as shown below



Click the New button to create a new target configuration file. Click here to hide this message.

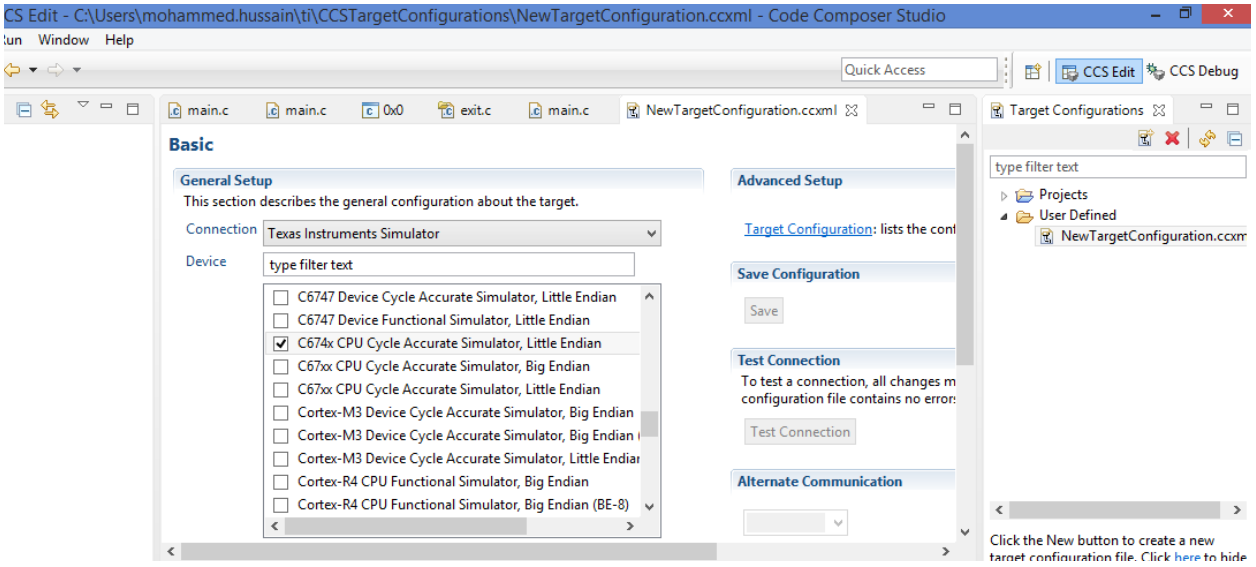
Click on NEW TARGET CONFIGURATION

A TARGET CONFIGURATION Window opens give any name with **.CCXML** Extension

After successful creation of TARGET FILE name click finish a window shown below appear

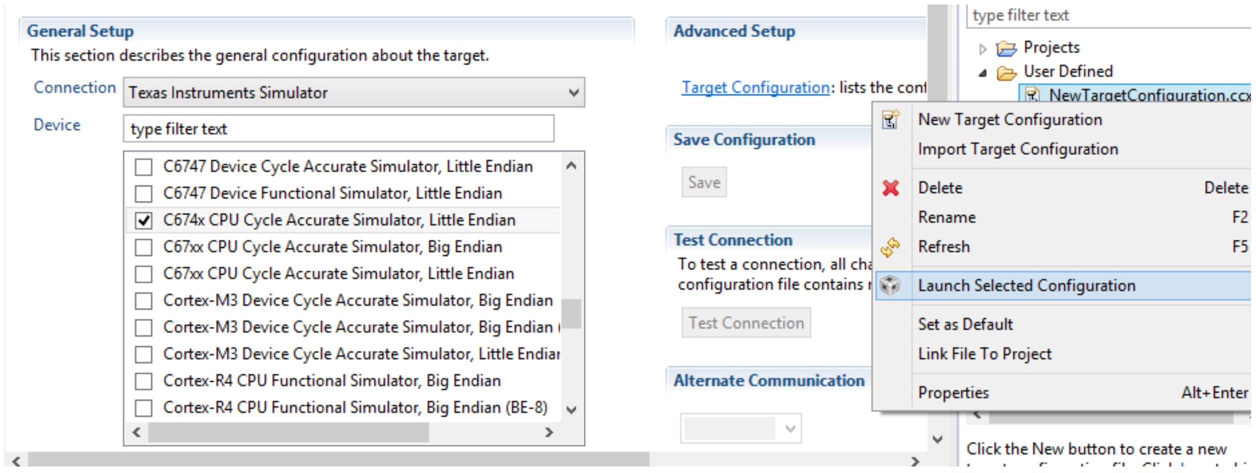
Select **Connection**: Texas Instrument Simulator

Target: C674x CPU Cycle Accurate Simulator, little Endian


CS Edit - C:\Users\mohammed.hussain\ti\CCSTargetConfigurations\NewTargetConfiguration.ccxml - Code Composer Studio
Basic
General Setup
This section describes the general configuration about the target.
Connection Texas Instruments Simulator
Device
C6747 Device Cycle Accurate Simulator, Little Endian
C6747 Device Functional Simulator, Little Endian
C674x CPU Cycle Accurate Simulator, Little Endian
C67xx CPU Cycle Accurate Simulator, Big Endian
C67xx CPU Cycle Accurate Simulator, Little Endian
Cortex-M3 Device Cycle Accurate Simulator, Big Endian
Advanced Setup
Save Configuration
Save
Test Connection
Alternate Communication
Target Configurations
Projects
User Defined
NewTargetConfiguration.ccxml

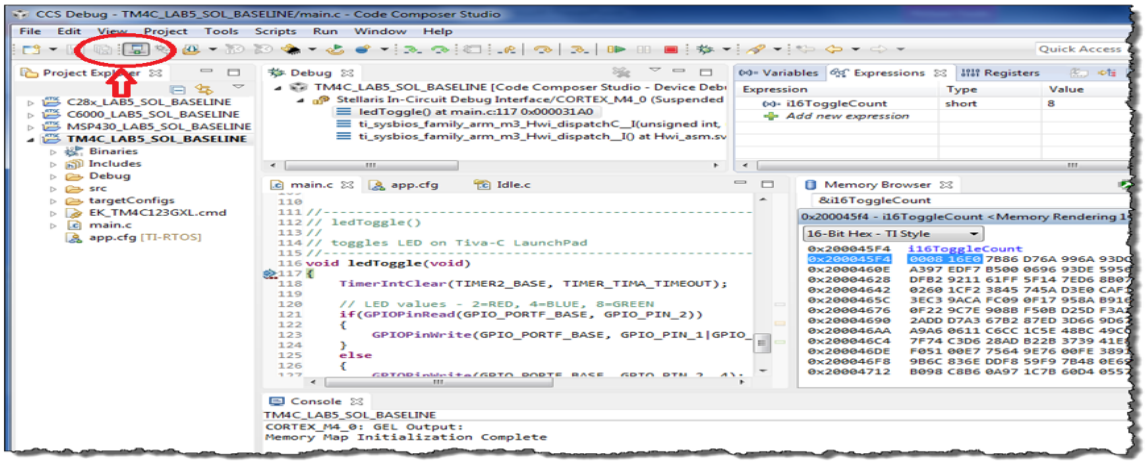
Save the changes

Right Click on Target configuration you created select **Launch Selected Configuration**


General Setup
Connection Texas Instruments Simulator
Device
C674x CPU Cycle Accurate Simulator, Little Endian
New Target Configuration
Import Target Configuration
Delete
Rename
Refresh
Launch Selected Configuration
Set as Default
Link File To Project
Properties

Then you will enter into DEBUG window

Click on Target Connection as shown below



```
TM4C_LAB5_SOL_BASELINE [Code Composer Studio - Device Debu
  Stellaris In-Circuit Debug Interface/CORTEX_M4_0 (Suspended
    ledToggle() at main.c:117 0x000031A0
    ti_sysbios_family_arm_m3_Hwi_dispatchC__I(unsigned int,
    ti_sysbios_family_arm_m3_Hwi_dispatch__I() at Hwi_asm.sv
```

Expression	Type	Value
i16ToggleCount	short	8
Add new expression		

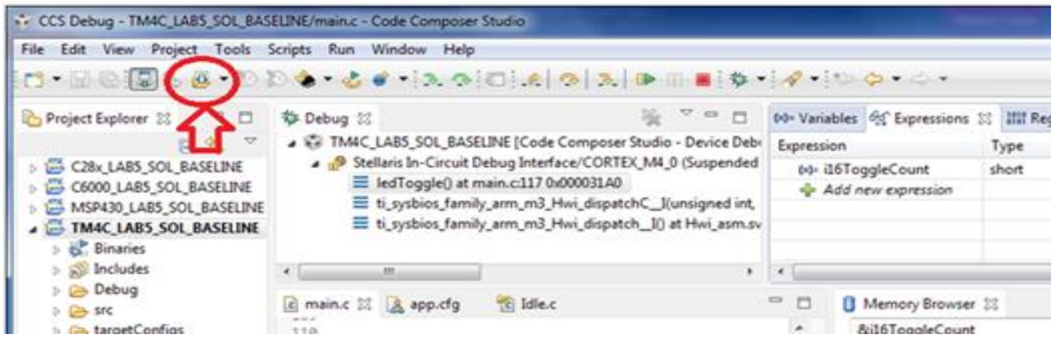
```
//---------------------------------------------------------------
// ledToggle()
//
// toggles LED on Tiva-C LaunchPad
//---------------------------------------------------------------
void ledToggle(void)
{
    TimerIntClear(TIMER2_BASE, TIMER_TIMA_TIMEOUT);

    // LED values - 2=RED, 4=BLUE, 8=GREEN
    if(GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_2))
    {
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_
    }
    else
    {
```

```
&i16ToggleCount
0x200045f4 - i16ToggleCount <Memory Rendering 1
16-Bit Hex - TI Style
0x200045F4    i16ToggleCount
0x200045F4    0008 16E0 7B86 D76A 996A 93DC
0x2000460E    A397 EDF7 B500 0696 93DE 5956
0x20004628    DFB2 9211 61FF 5F14 7ED6 8B07
0x20004642    0260 1CF2 3845 745A D3E0 CAF1
0x2000465C    3EC3 9ACA FC09 0F17 958A B916
0x20004676    0F22 9C7E 908B F50B D25D F3A1
0x20004690    2ADD D7A3 67B2 87ED 3D66 9D67
0x200046AA    A9A6 0611 C6CC 1C5E 48BC 49C0
0x200046C4    7F74 C3D6 28AD B22B 3739 41E8
0x200046DE    F051 00E7 7564 9E76 00FE 3891
0x200046F8    9B6C 836E DDF8 59F9 7B48 0E69
0x20004712    B098 C8B6 0A97 1C7B 60D4 0557
```

```
TM4C_LAB5_SOL_BASELINE
CORTEX_M4_0: GEL Output:
Memory Map Initialization Complete
```

After successful memory map initialization (shown in console window)

Load corresponding program

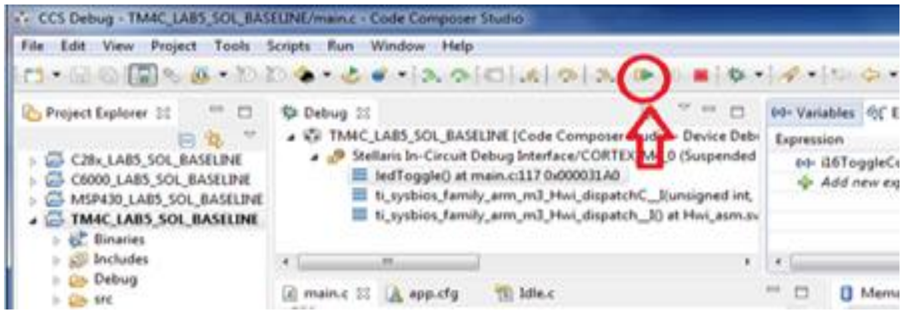

CCS Debug - TM4C_LAB5_SOL_BASELINE/main.c - Code Composer Studio
File Edit View Project Tools Scripts Run Window Help
Project Explorer
C28x_LAB5_SOL_BASELINE
C6000_LAB5_SOL_BASELINE
MSP430_LAB5_SOL_BASELINE
TM4C_LAB5_SOL_BASELINE
Binaries
Includes
Debug
src
targetConfigs
Debug
TM4C_LAB5_SOL_BASELINE [Code Composer Studio - Device Debu
Stellaris In-Circuit Debug Interface/CORTEX_M4_0 (Suspended
ledToggle() at main.c:117 0x000031A0
ti_sysbios_family_arm_m3_Hwi_dispatchC__I(unsigned int,
ti_sysbios_family_arm_m3_Hwi_dispatch__I() at Hwi_asm.sv
Variables
Expressions
Expression
Type
i16ToggleCount
short
Add new expression
main.c
app.cfg
Idle.c
Memory Browser
&i16ToggleCount

Click on **Browse Project** and select corresponding .OUT file from your

 project name(Hello) →**Debug**→**.OUT**

Hit OK

Now run Program


CCS Debug - TM4C_LAB5_SOL_BASELINE/main.c - Code Composer Studio
File Edit View Project Tools Scripts Run Window Help
Project Explorer
C28x_LAB5_SOL_BASELINE
C6000_LAB5_SOL_BASELINE
MSP430_LAB5_SOL_BASELINE
TM4C_LAB5_SOL_BASELINE
Binaries
Includes
Debug
src
Debug
TM4C_LAB5_SOL_BASELINE [Code Composer Studio - Device Debu
Stellaris In-Circuit Debug Interface/CORTEX_M4_0 (Suspended
ledToggle() at main.c:117 0x000031A0
ti_sysbios_family_arm_m3_Hwi_dispatchC__I(unsigned int,
ti_sysbios_family_arm_m3_Hwi_dispatch__I() at Hwi_asm.sv
Variables
Expression
i16ToggleC
Add new ex
main.c
app.cfg
Idle.c

OUTPUT can be observed in console window

To see graph same procedure can be followed as in NON REAL TIME WITH TARGET

GENERAL PROCEDURE FOR WORKING WITH THE REAL TIME PROJECTS USING C6748DSK:

AUDIO PLAYBACK

1.0 Unit Objective:

To configure the codec AIC3106 for a talk through program using the board support library.

2.0 Prerequisites

C6748LCDK, PC with Code Composer Studio, CRO, Audio Source, Speakers, headphone and Signal Generator.

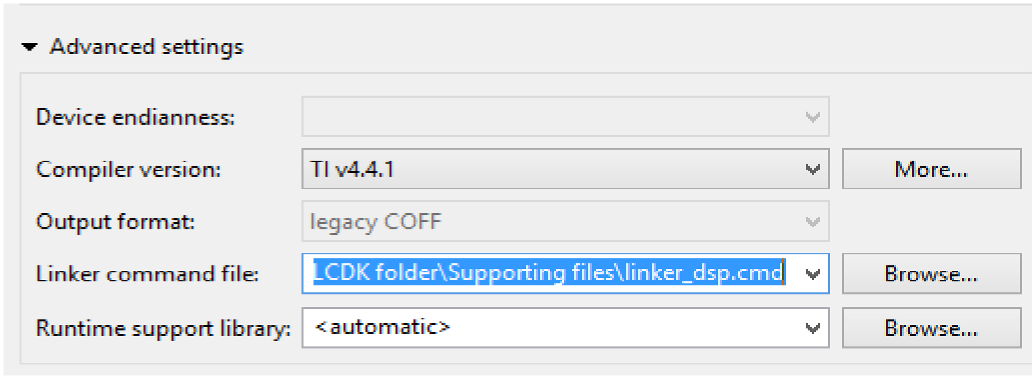
3.0 Procedure

- All the Real time implementations covered in the Implementations module follow code Configuration using board support library.
- The board Support Library (CSL) is a collection of functions, macros, and symbols used to configure and control on-chip peripherals.
- The goal is peripheral ease of use, shortened development time, portability, hardware abstraction, and some level of standardization and compatibility among TI devices.
- Connect one end of a stereo cable to the LINE IN(J55) of the kit and other end to PC (or function generator).
- Connect the headphone (or CRO) to the LINE OUT(J56) of the kit.
- Connect the power supply to the board.
- Connect the External JTAG XDS100v2 with board and PC.

<u>General Steps For Real Time Programs</u>:

R1. Create project.

- Goto File ->new->CCS project and enter the details as done in NR3 but only change is in "Advanced Settings" choose the Linker command file: as "**linker_dsp.cmd**" (Browse from Supporting files folder given at the time of installation)


Advanced settings

Device endianness:

Compiler version: TI v4.4.1

Output format: legacy COFF

Linker command file: LCDK folder\Supporting files\linker_dsp.cmd

Runtime support library: <automatic>

R2. Create source file to write code.

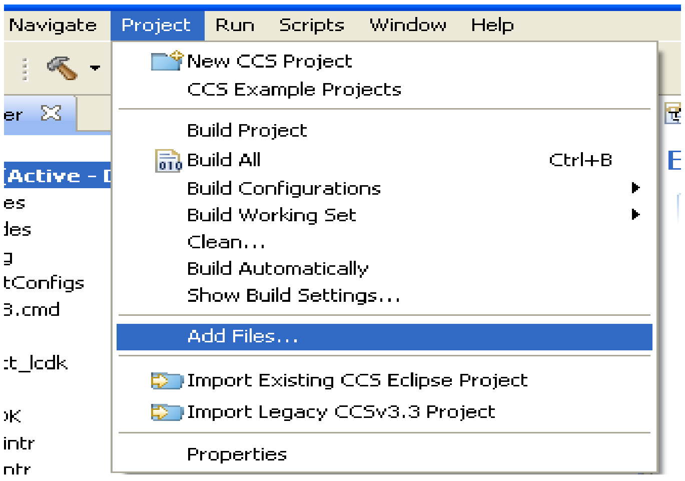
In source file type (or copy) the following code.

```
// L138_loop_intr.c
//

#include "L138_LCDK_aic3106_init.h"

interrupt void interrupt4(void)   // interrupt service routine
{
  uint32_t sample;

  sample = input_sample(); // read L + R samples from ADC
  output_sample(sample);   // write L + R samples to DAC
  return;
}

int main(void)
{
L138_initialise_intr(FS_48000_HZ,ADC_GAIN_0DB,DAC_ATTEN_0DB,LCDK_LINE_INPUT)
;
  while(1);
}
```

Once the program is written, save it.

R3. Now add the following library files and supporting files of codec to your project.

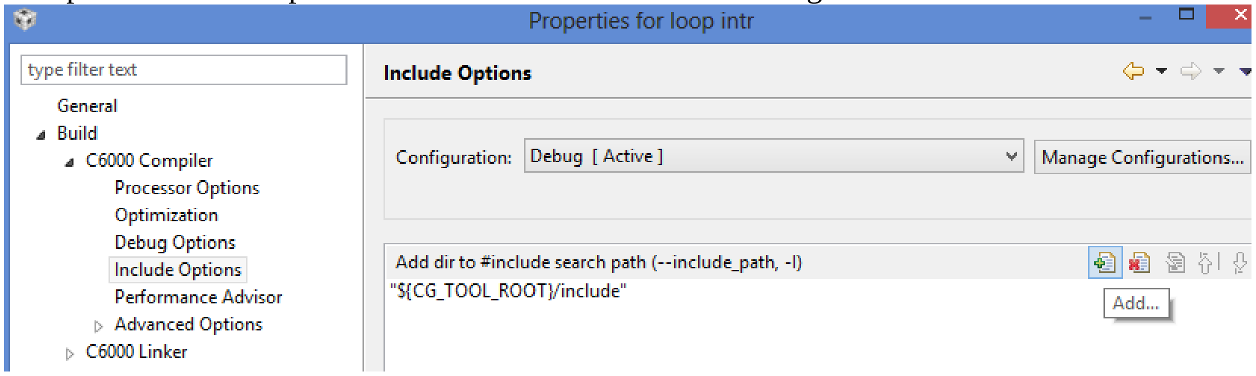
- To add files goto Project-> Add files.



- Follow the path given below to add the required files:

1. Supporting files**evmomapl138_bsl.lib**
2. Supporting files **L138_LCDK_aic3106_init.h**
3. Supporting files **L138_LCDK_aic3106_init.c**
4. Supporting files **vectors_intr.asm**

R4. Right click on the project name and choose "Show Build Settings.." -> Build-> C6000 Compiler-> Include Options -> Click "Add" as shown in the figure.

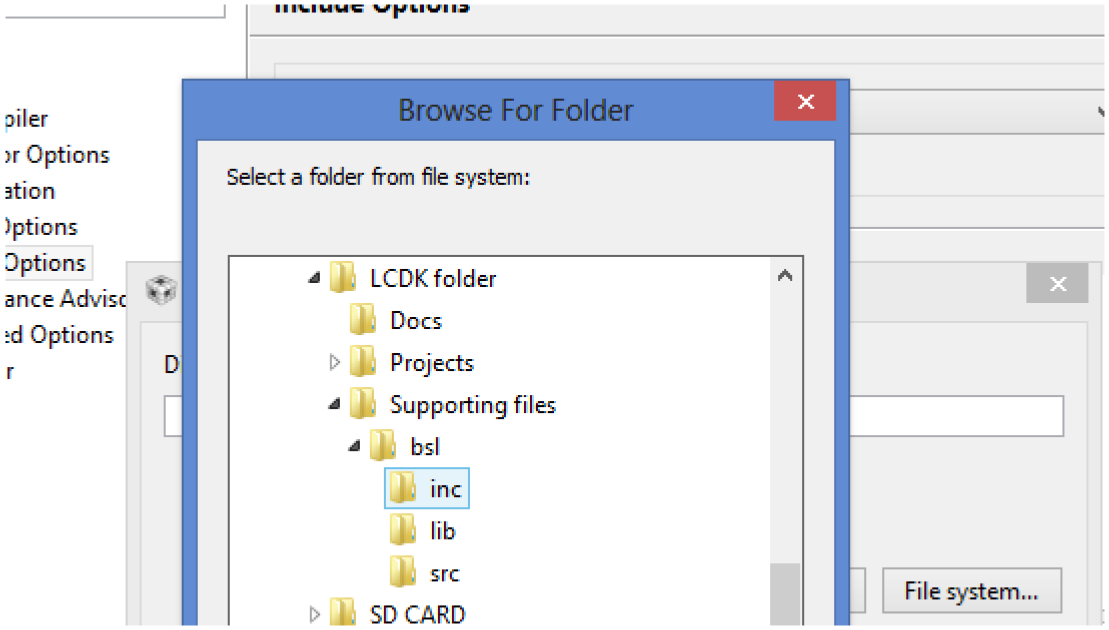


Configuration: Debug [Active]

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

"${CG_TOOL_ROOT}/include"

- Now Click File system -> supporting files->bsl->inc->OK.



R5. Build Project, Load and Resume the project.

After clicking the Resume option, play the music now. You can hear the song from the headphone without any noise.

Alternatively you can also connect function generator to the LINE IN and CRO to the LINE OUT. At this time you can observe same wave appearing on the CRO which is given in function generator. Thus the CODEC is verified.

You can also try changing the sampling frequency, gain of the codec.

Note: if you want to play loop signal only in right channel then change the 6th line to

output_right_sample(sample);

If you want to play loop signal only in left channel then change the 6th line to

output_left_sample(sample);

Creating ECO Effect

```c
// L138_echo_intr.c
//

#include "L138_LCDK_aic3106_init.h"
#define GAIN 0.6
#define BUF_SIZE 16000

int16_t input,output,delayed;
int16_t buffer[BUF_SIZE];
int i = 0;

interrupt void interrupt4(void) // interrupt service routine
{
  input = input_left_sample();
  delayed = buffer[i];
  output = delayed + input;
  buffer[i] = input + delayed*GAIN;
  i = (i+1)%BUF_SIZE;
  output_left_sample(output);
  return;
}

int main(void)
{
  int i;

  for (i=0 ; i<BUF_SIZE ; i++)
  {
    buffer[i] = 0;
  }
  L138_initialise_intr(FS_48000_HZ,ADC_GAIN_0DB,DAC_ATTEN_0DB,LCDK_MIC_INPUT);
  while(1);
}
```

Procedure:

- After successful generation of .out file connect the Target to Host PC run program
- Connect the MIC to MIC INPUT
- Connect loud speaker to LINE OUT By giving input to MIC observe the output in loud speaker

Creating DELAY

```
// L138_delay_intr.c
//

#include "L138_LCDK_aic3106_init.h"
#define BUF_SIZE 24000

uint16_t input,output,delayed;
uint16_t buffer[BUF_SIZE];
int i = 0;

interrupt void interrupt4(void) // interrupt service routine
{
  input = input_left_sample();
  delayed = buffer[i];
  output = delayed + input;
  buffer[i] = input;
  i = (i+1)%BUF_SIZE;
  output_left_sample(output);
  return;
}

int main(void)
{
  int i;

  for (i=0 ; i<BUF_SIZE ; i++)
  {
    buffer[i] = 0;
  }
  L138_initialise_intr(FS_48000_HZ,ADC_GAIN_0DB,DAC_ATTEN_0DB,LCDK_MIC_INPUT);
  while(1);
}
```

Procedure:

- Follow the same procedure as ECO program

Advance Discrete Time Filter Design (FIR)

Finite Impulse Response Filter

DESIGNING AN FIR FILTER:

Following are the steps to design linear phase FIR filters Using Windowing Method.

I. Clearly specify the filter specifications.

Eg: Order = 30;
Sampling Rate = 8000 samples/sec
Cut off Freq. = 400 Hz.

II. Compute the cut-off frequency W_c

Eg: $W_c = 2*pi* f_c / F_s$
$= 2*pi* 400/8000$
$= 0.1*pi$

III. Compute the desired Impulse Response $h_d(n)$ using particular Window

Eg: b_rect1=fir1(order, W_c, 'high',boxcar(31));

IV. Convolve input sequence with truncated Impulse Response x (n)*h (n)

USING MATLAB TO DETERMINE FILTER COEFFICIENTS :
Using FIR1 Function on Matlab

B = FIR1(N,Wn) designs an N'th order lowpass FIR digital filter and returns the filter coefficients in length N+1 vector B.

The cut-off frequency Wn must be between 0 < Wn < 1.0, with 1.0 corresponding to half the sample rate. The filter B is real and has linear phase, i.e., even symmetric coefficients obeying B(k) = B(N+2-k), k = 1,2,...,N+1.

If Wn is a two-element vector, Wn = [W1 W2], FIR1 returns an order N bandpass filter with passband W1 < W < W2.
B = FIR1(N,Wn,'high') designs a highpass filter.
B = FIR1(N,Wn,'stop') is a bandstop filter if Wn = [W1 W2].

If Wn is a multi-element vector,
    Wn = [W1 W2 W3 W4 W5 ... WN],
FIR1 returns an order N multiband filter with bands
0 < W < W1, W1 < W < W2, ..., WN < W < 1.
B = FIR1(N,Wn,'DC-1') makes the first band a passband.
B = FIR1(N,Wn,'DC-0') makes the first band a stopband.

For filters with a passband near Fs/2, e.g., highpass
and bandstop filters, N must be even.

By default FIR1 uses a Hamming window. Other available windows,
including Boxcar, Hanning, Bartlett, Blackman, Kaiser and Chebwin
can be specified with an optional trailing argument. For example,
B = FIR1(N,Wn,kaiser(N+1,4)) uses a Kaiser window with beta=4.
B = FIR1(N,Wn,'high',chebwin(N+1,R)) uses a Chebyshev window.

By default, the filter is scaled so the center of the first pass band
has magnitude exactly one after windowing. Use a trailing 'noscale'
argument to prevent this scaling, e.g. B = FIR1(N,Wn,'noscale'),
B = FIR1(N,Wn,'high','noscale'), B = FIR1(N,Wn,wind,'noscale').

<u>Matlab Program to generate 'FIR Filter-Low Pass' Coefficients using FIR1</u>

```
% FIR Low  pass filters using rectangular, triangular and kaiser windows
% sampling rate - 8000
order = 30;
cf=[500/4000,1000/4000,1500/4000];                    cf--> contains set of cut-off  frequencies[Wc]

% cutoff frequency - 500
b_rect1=fir1(order,cf(1),boxcar(31));  Rectangular
b_tri1=fir1(order,cf(1),bartlett(31));    Triangular
b_kai1=fir1(order,cf(1),kaiser(31,8)); Kaisar [Where 8-->Beta Co-efficient]


% cutoff frequency - 1000
b_rect2=fir1(order,cf(2),boxcar(31));
b_tri2=fir1(order,cf(2),bartlett(31));
b_kai2=fir1(order,cf(2),kaiser(31,8));


% cutoff frequency - 1500
b_rect3=fir1(order,cf(3),boxcar(31));
b_tri3=fir1(order,cf(3),bartlett(31));
b_kai3=fir1(order,cf(3),kaiser(31,8));


fid=fopen('FIR_lowpass_rectangular.txt','wt');
fprintf(fid,'\t\t\t\t\t\t%s\n','Cutoff -400Hz');
fprintf(fid,'\nfloat b_rect1[31]={');
fprintf(fid,'%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,\n',b_rect1);
fseek(fid,-1,0);
fprintf(fid,'};');

fprintf(fid,'\n\n\n\n');
fprintf(fid,'\t\t\t\t\t\t%s\n','Cutoff -800Hz');
fprintf(fid,'\nfloat b_rect2[31]={');
fprintf(fid,'%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,\n',b_rect2);
fseek(fid,-1,0);
fprintf(fid,'};');

fprintf(fid,'\n\n\n\n');
fprintf(fid,'\t\t\t\t\t\t%s\n','Cutoff -1200Hz');
fprintf(fid,'\nfloat b_rect3[31]={');
fprintf(fid,'%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,\n',b_rect3);
fseek(fid,-1,0);
fprintf(fid,'};');
fclose(fid);
winopen('FIR_lowpass_rectangular.txt');
```

T.1 : Matlab generated Coefficients for FIR Low Pass Kaiser filter:

Cutoff -500Hz
```
float b_kai1[31]={-0.000019,-0.000170,-0.000609,-0.001451,-0.002593,-0.003511,-
0.003150,0.000000,0.007551,0.020655,0.039383,0.062306,0.086494,0.108031,0.122944,
0.128279,0.122944,0.108031,0.086494,0.062306,0.039383,0.020655,0.007551,0.000000,
-0.003150,-0.003511,-0.002593,-0.001451,-0.000609,-0.000170,-0.000019};
```

Cutoff -1000Hz
```
float b_kai2[31]={-0.000035,-0.000234,-0.000454,0.000000,0.001933,0.004838,0.005671,
-0.000000,-0.013596,-0.028462,-0.029370,0.000000,0.064504,0.148863,0.221349,0.249983,
0.221349,0.148863,0.064504,0.000000,-0.029370,-0.028462,-0.013596,-0.000000,0.005671,
0.004838,0.001933,0.000000,-0.000454,-0.000234, -0.000035};
```

Cutoff -1500Hz
```
float b_kai3[31]={-0.000046,-0.000166,0.000246,0.001414,0.001046,-0.003421,-0.007410,
0.000000,0.017764,0.020126,-0.015895,-0.060710,-0.034909,0.105263,0.289209,0.374978,
0.289209,0.105263,-0.034909,-0.060710,-0.015895,0.020126,0.017764,0.000000,-0.007410,
-0.003421,0.001046,0.001414,0.000246,-0.000166, -0.000046};
```

filter

Cutoff -500Hz
```
float b_rect1[31]={-0.008982,-0.017782,-0.025020,-0.029339,-0.029569,-0.024895,
-0.014970,0.000000,0.019247,0.041491,0.065053,0.088016,0.108421,0.124473,0.134729,
0.138255,0.134729,0.124473,0.108421,0.088016,0.065053,0.041491,0.019247,0.000000,
-0.014970,-0.024895,-0.029569,-0.029339,-0.025020,-0.017782,-0.008982};
```

Cutoff -1000Hz
```
float b_rect2[31]={-0.015752,-0.023869,-0.018176,0.000000,0.021481,0.033416,0.026254,-0.000000,-
0.033755,-0.055693,-0.047257,0.000000,0.078762,0.167080,0.236286,0.262448,
0.236286,0.167080,0.078762,0.000000,-0.047257,-0.055693,-0.033755,-0.000000,0.026254,
0.033416,0.021481,0.000000,-0.018176,-0.023869,-0.015752};
```

Cutoff -1500Hz
```
float b_rect2[31]={-0.020203,-0.016567,0.009656,0.027335,0.011411,-0.023194,-0.033672,
0.000000,0.043293,0.038657,-0.025105,-0.082004,-0.041842,0.115971,0.303048,0.386435,
0.303048,0.115971,-0.041842,-0.082004,-0.025105,0.038657,0.043293,0.000000,-0.033672,
-0.023194,0.011411,0.027335,0.009656,-0.016567,-0.020203};
```

Cutoff -500Hz

```
float b_tri1[31]={0.000000,-0.001185,-0.003336,-0.005868,-0.007885,-0.008298,-0.005988,
0.000000,0.010265,0.024895,0.043368,0.064545,0.086737,0.107877,0.125747,0.138255,
0.125747,0.107877,0.086737,0.064545,0.043368,0.024895,0.010265,0.000000,-0.005988,
-0.008298,-0.007885,-0.005868,-0.003336,-0.001185,0.000000};
```

Cutoff -1000Hz

```
float b_tri2[31]={0.000000,-0.001591,-0.002423,0.000000,0.005728,0.011139,0.010502,
-0.000000,-0.018003,-0.033416,-0.031505,0.000000,0.063010,0.144802,0.220534,0.262448,
0.220534,0.144802,0.063010,0.000000,-0.031505,-0.033416,-0.018003,-0.000000,0.010502,
0.011139,0.005728,0.000000,-0.002423,-0.001591,0.000000};
```

Cutoff -1500Hz

```
float b_tri3[31]={0.000000,-0.001104,0.001287,0.005467,0.003043,-0.007731,-0.013469,
0.000000,0.023089,0.023194,-0.016737,-0.060136,-0.033474,0.100508,0.282844,0.386435,
0.282844,0.100508,-0.033474,-0.060136,-0.016737,0.023194,0.023089,0.000000,-0.013469,
-0.007731,0.003043,0.005467,0.001287,-0.001104,0.000000};
```

MATLAB Program to generate 'FIR Filter-High Pass' Coefficients using FIR1

```
% FIR High pass filters using rectangular, triangular and kaiser windows
% sampling rate - 8000
order = 30;

cf=[400/4000,800/4000,1200/4000];                ;cf--> contains set of cut-off frequencies[Wc]

% cutoff frequency - 400
b_rect1=fir1(order,cf(1),'high',boxcar(31));
b_tri1=fir1(order,cf(1),'high',bartlett(31));
b_kai1=fir1(order,cf(1),'high',kaiser(31,8)); Where Kaiser(31,8)--> '8'defines the value of 'beta'

% cutoff frequency - 800
b_rect2=fir1(order,cf(2),'high',boxcar(31));
b_tri2=fir1(order,cf(2),'high',bartlett(31));
b_kai2=fir1(order,cf(2),'high',kaiser(31,8));

% cutoff frequency - 1200
b_rect3=fir1(order,cf(3),'high',boxcar(31));
b_tri3=fir1(order,cf(3),'high',bartlett(31));
b_kai3=fir1(order,cf(3),'high',kaiser(31,8));

fid=fopen('FIR_highpass_rectangular.txt','wt');
fprintf(fid,'\t\t\t\t\t\t%s\n','Cutoff -400Hz');
fprintf(fid,'\nfloat b_rect1[31]={');
fprintf(fid,'%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,\n',b_rect1);
fseek(fid,-1,0);
fprintf(fid,'};');
fprintf(fid,'\n\n\n\n');
fprintf(fid,'\t\t\t\t\t\t%s\n','Cutoff -800Hz');
fprintf(fid,'\nfloat b_rect2[31]={');
fprintf(fid,'%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,\n',b_rect2);
fseek(fid,-1,0);
fprintf(fid,'};');

fprintf(fid,'\n\n\n\n');
fprintf(fid,'\t\t\t\t\t\t%s\n','Cutoff -1200Hz');
fprintf(fid,'\nfloat b_rect3[31]={');
fprintf(fid,'%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,\n',b_rect3);
fseek(fid,-1,0);
fprintf(fid,'};');
fclose(fid);
winopen('FIR_highpass_rectangular.txt');
```

T.1 : MATLAB generated Coefficients for FIR High Pass Kaiserfilter:

Cutoff -400Hz

```
float b_kai1[31]={0.000050,0.000223,0.000520,0.000831,0.000845,-0.000000,-0.002478,
-0.007437,-0.015556,-0.027071,-0.041538,-0.057742,-0.073805,-0.087505,-0.096739,
0.899998,-0.096739,-0.087505,-0.073805,-0.057742,-0.041538,-0.027071,-0.015556,
-0.007437,-0.002478,-0.000000,0.000845,0.000831,0.000520,0.000223,0.000050};
```

Cutoff -800Hz

```
float b_kai2[31]={0.000000,-0.000138,-0.000611,-0.001345,-0.001607,-0.000000,0.004714,
0.012033,0.018287,0.016731,0.000000,-0.035687,-0.086763,-0.141588,-0.184011,0.800005,
-0.184011,-0.141588,-0.086763,-0.035687,0.000000,0.016731,0.018287,0.012033,0.004714,
-0.000000,-0.001607,-0.001345,-0.000611,-0.000138,0.000000};
```

Cutoff -1200Hz

```
float b_kai3[31]={-0.000050,-0.000138,0.000198,0.001345,0.002212,-0.000000,-0.006489,
-0.012033,-0.005942,0.016731,0.041539,0.035687,-0.028191,-0.141589,-0.253270,0.700008,
-0.253270,-0.141589,-0.028191,0.035687,0.041539,0.016731,-0.005942,-0.012033,-0.006489,
-0.000000,0.002212,0.001345,0.000198,-0.000138,-0.000050};
```

T.2 :MATLAB generated Coefficients for FIR High Pass Rectangular filter

Cutoff -400Hz

```
float b_rect1[31]={0.021665,0.022076,0.020224,0.015918,0.009129,-0.000000,-0.011158,
-0.023877,-0.037558,-0.051511,-0.064994,-0.077266,-0.087636,-0.095507,-.100422,0.918834,
-0.100422,-0.095507,-0.087636,-0.077266,-0.064994,-0.051511,-0.037558,-0.023877,
-0.011158,-0.000000,0.009129,0.015918,0.020224,0.022076,0.021665};
```

Cutoff -800Hz

```
float b_rect2[31]={0.000000,-0.013457,-0.023448,-0.025402,-0.017127,-0.000000,0.020933,
0.038103,0.043547,0.031399,0.000000,-0.047098,-0.101609,-0.152414,-0.188394,0.805541,
-0.188394,-0.152414,-0.101609,-0.047098,0.000000,0.031399,0.043547,0.038103,0.020933,
-0.000000,-0.017127,-0.025402,-0.023448,-0.013457,0.000000};
```

Cutoff -1200Hz

```
float b_rect3[31]={-0.020798,-0.013098,0.007416,0.024725,0.022944,-0.000000,-0.028043,
-0.037087,-0.013772,0.030562,0.062393,0.045842,-0.032134,-0.148349,-0.252386,0.686050,
-0.252386,-0.148349,-0.032134,0.045842,0.062393,0.030562,-0.013772,-0.037087,-0.028043,
-0.000000,0.022944,0.024725,0.007416,-0.013098,-0.020798};
```

T.3 : MATLAB generated Coefficients for FIR High Pass Triangular filter

Cutoff -400Hz

```
float b_tri1[31]={0.000000,0.001445,0.002648,0.003127,0.002391,-0.000000,-0.004383,
-0.010943,-0.019672,-0.030353,-0.042554,-0.055647,-0.068853,-0.081290,-0.092048,
0.902380,-0.092048,-0.081290,-0.068853,-0.055647,-0.042554,-0.030353,-0.019672,
-0.010943,-0.004383,-0.000000,0.002391,0.003127,0.002648,0.001445,0.000000};
```

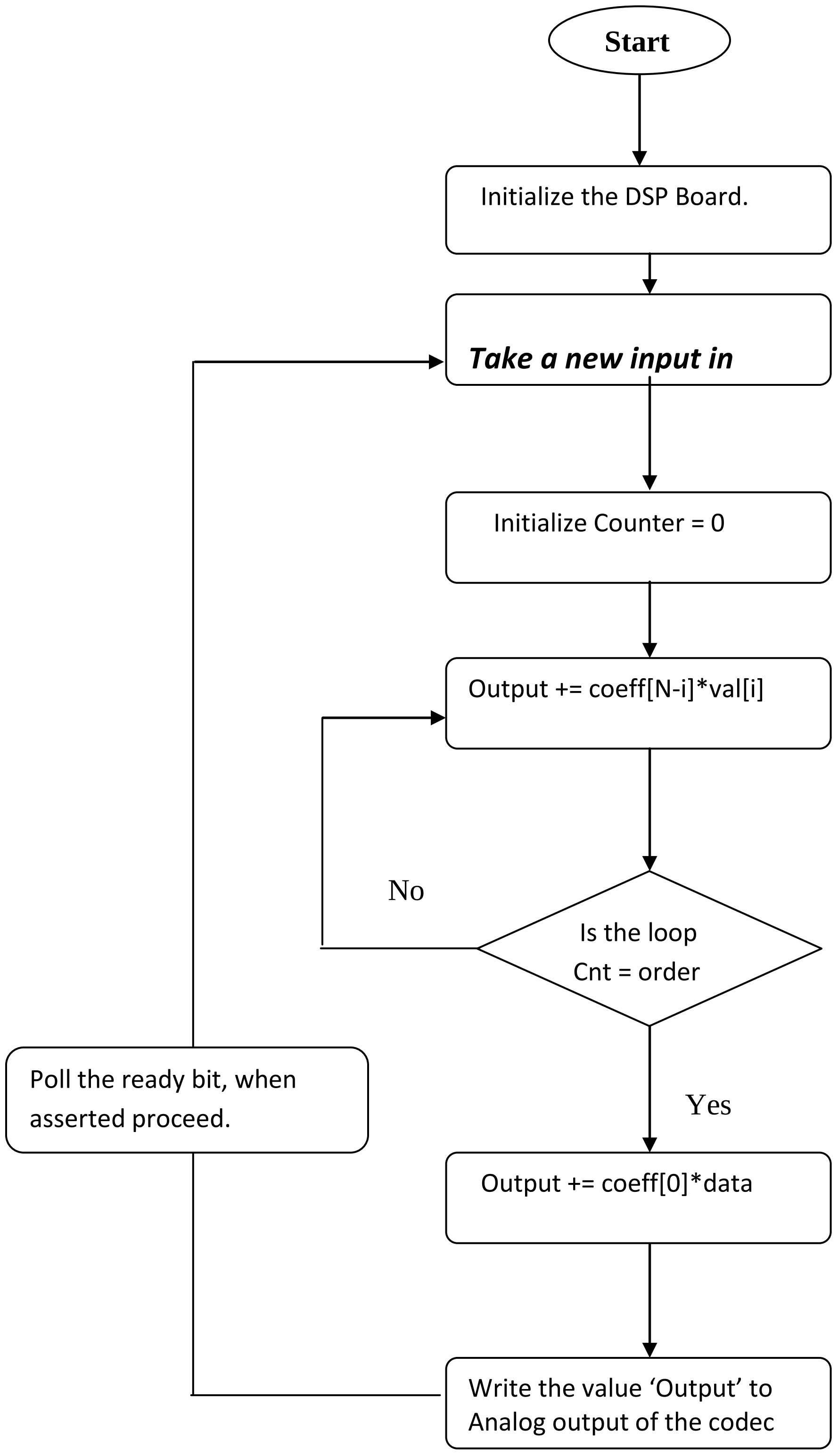
Cutoff -800Hz

```
float b_tri2[31]={0.000000,-0.000897,-0.003126,-0.005080,-0.004567,-0.000000,0.008373,
0.017782,0.023225,0.018839,0.000000,-0.034539,-0.081287,-0.132092,-0.175834,0.805541,
-0.175834,-0.132092,-0.081287,-0.034539,0.000000,0.018839,0.023225,0.017782,0.008373,
-0.000000,-0.004567,-0.005080,-0.003126,-0.000897,0.000000};
```

Cutoff -1200Hz

```
float b_tri3[31]={0.000000,-0.000901,0.001021,0.005105,0.006317,-0.000000,-0.011581,
-0.017868,-0.007583,0.018931,0.042944,0.034707,-0.026541,-0.132736,-0.243196,0.708287,
-0.243196,-0.132736,-0.026541,0.034707,0.042944,0.018931,-0.007583,-0.017868,-0.011581,
-0.000000,0.006317,0.005105,0.001021,-0.000901,0.000000};
```

FLOW CHART TO IMPLEMENT FIR FILTER:


Start
Initialize the DSP Board.
Take a new input in
Initialize Counter = 0
Output += coeff[N-i]*val[i]
No
Is the loop Cnt = order
Yes
Output += coeff[0]*data
Write the value 'Output' to Analog output of the codec
Poll the ready bit, when asserted proceed.

C PROGRAM TO IMPLEMENT FIR FILTER:

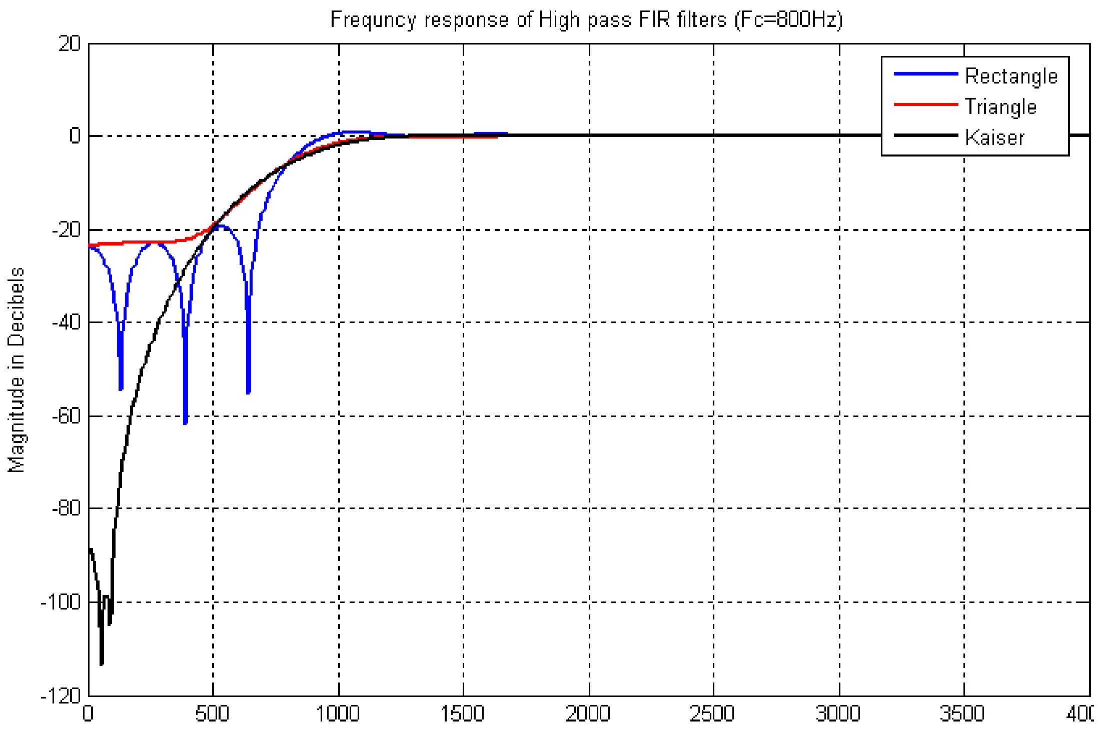
```
// L138_fir_intr.c

#include "L138_LCDK_aic3106_init.h"
#define N 5
float h[N] = {
2.0000E-001,2.0000E-001,2.0000E-001,2.0000E-001,2.0000E-001
};
float x[N]; // filter delay line
interrupt void interrupt4(void)
{
  short i;
  float yn = 0.0;
  x[0] = (float)(input_left_sample()); // input from ADC
  for (i=0 ; i<N ; i++)                // compute filter output
    yn += h[i]*x[i];
  for (i=(N-1) ; i>0 ; i--)            // shift delay line
    x[i] = x[i-1];
  output_left_sample((uint16_t)(yn));  // output to DAC
  return;
}
int main(void)
{
L138_initialise_intr(FS_8000_HZ,ADC_GAIN_0DB,DAC_ATTEN_0DB,LCDK_LI
NE_INPUT);
  while(1);
}
```

PROCEDURE:

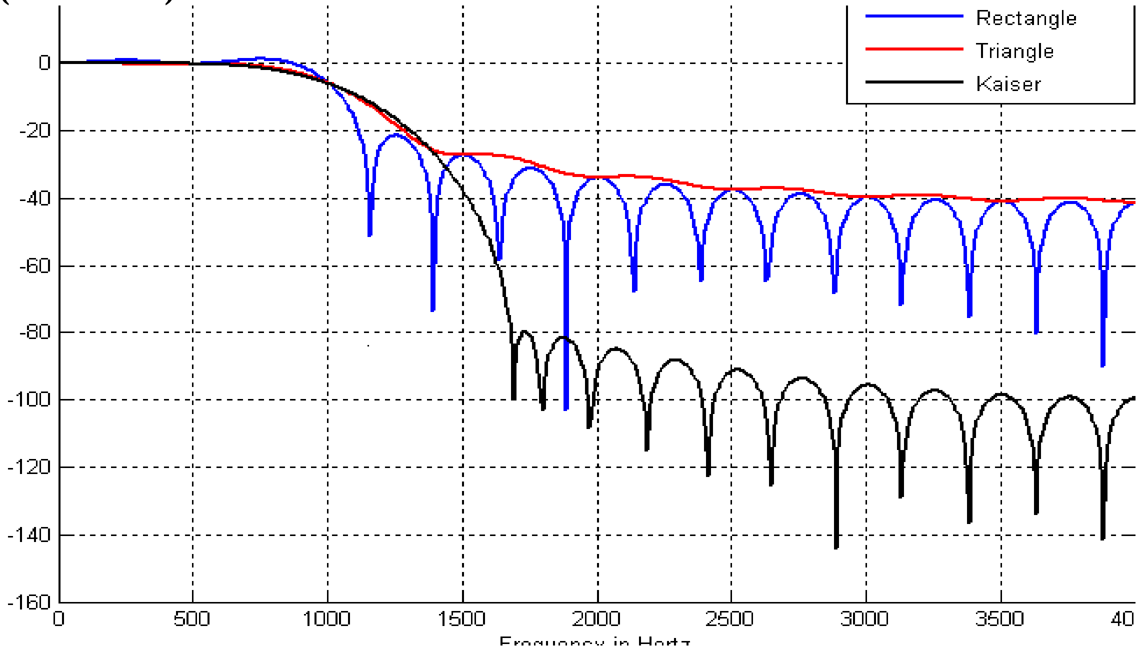
- After successful generation of .out file connect target to host PC
- Connect signal generator to LINE INPUT with 2Vp-p Sine Wave frequency is set according to filter co-efficient
- Connect CRO to LINE OUT with volt/div at 1 and time/div at 5msec

MATLAB GENERATED FREQUENCY RESPONSE

High Pass FIR filter(Fc= 800Hz).



Low Pass FIR filter (Fc=1000Hz)



ADVANCE DISCRETE TIME FILTER DESIGN (IIR)

IIR filter Designing Experiments

GENERAL CONSIDERATIONS:

In the design of frequency – selective filters, the desired filter characteristics are specified in the frequency domain in terms of the desired magnitude and phase response of the filter. In the filter design process, we determine the coefficients of a causal IIR filter that closely approximates the desired frequency response specifications.

IMPLEMENTATION OF DISCRETE-TIME SYSTEMS:

Discrete time Linear Time-Invariant (LTI) systems can be described completely by constant coefficient linear difference equations. Representing a system in terms of constant coefficient linear difference equation is it's time domain characterization. In the design of a simple frequency–selective filter, we would take help of some basic implementation methods for realizations of LTI systems described by linear constant coefficient difference equation.

UNIT OBJECTIVE:

The aim of this laboratory exercise is to design and implement a Digital IIR Filter & observe its frequency response. In this experiment we design a simple IIR filter so as to stop or attenuate required band of frequencies components and pass the frequency components which are outside the required band.

BACKGROUND CONCEPTS:

An Infinite impulse response (IIR) filter possesses an output response to an impulse which is of an infinite duration. The impulse response is "infinite" since there is feedback in the filter, which is if you put in an impulse, then its output must produced for infinite duration of time.

PREREQUISITES:

Ω Concept of discrete time signal processing.
Ω Analog filter design concepts.
Ω TMS320C6748 Architecture and instruction set.

EQUIPMENTS NEEDED:

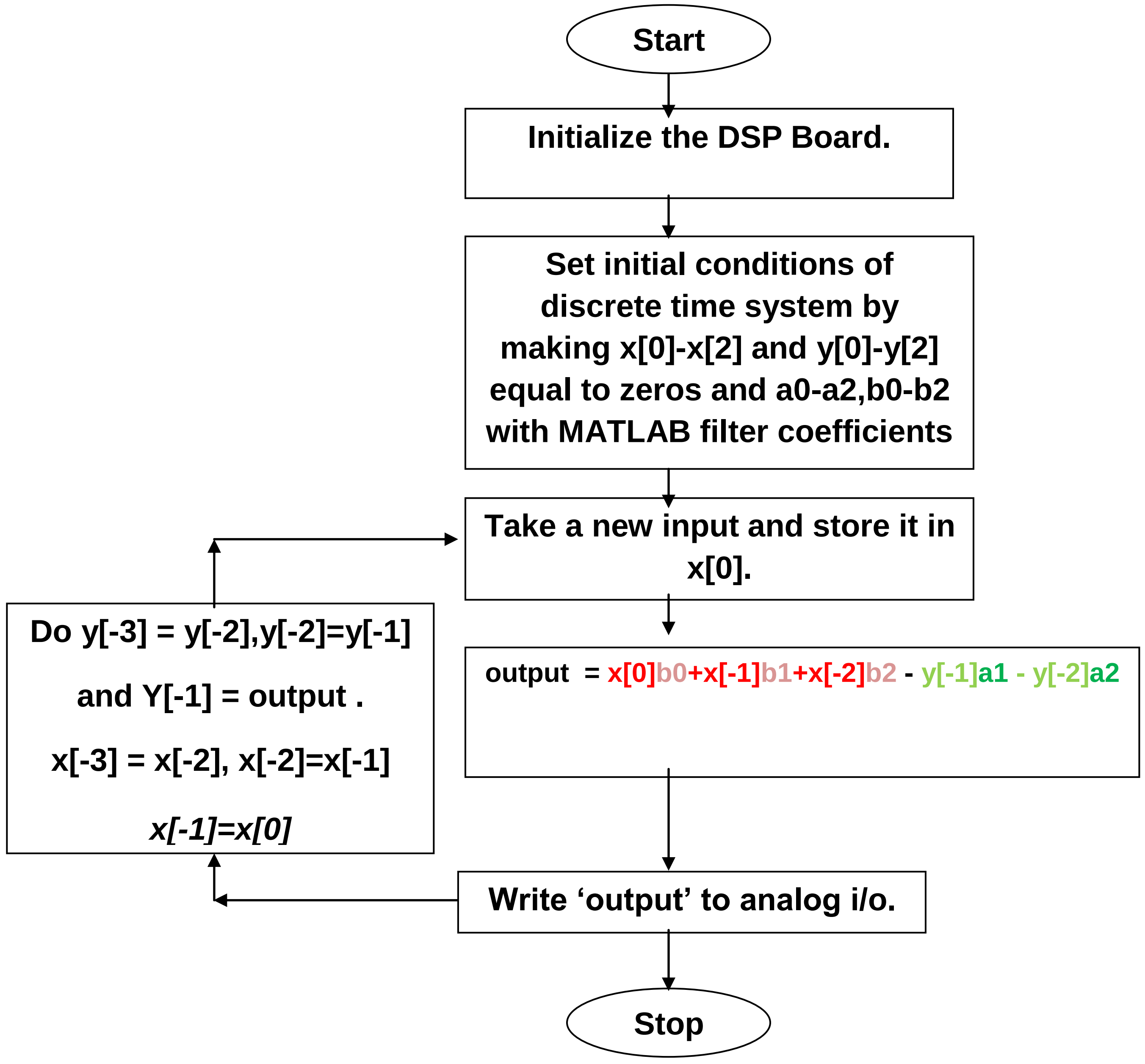
Ω Host (PC).
Ω TMS320C6748 DSP Kit.
Ω Oscilloscope and Function generator.

ALGORITHM TO IMPLEMENT:

We need to realize the Butter worth band pass IIR filter by implementing the difference equation $y[n] = b_0x[n] + b_1x[n-1]+b_2x[n-2]-a_1y[n-1]-a_2y[n-2]$ where $b_0 - b_2$, a_0-a_2 are feed forward and feedback word coefficients respectively [Assume 2nd order of filter].These coefficients are calculated using MATLAB.A direct **form I** implementation approach is taken.

- **Step 1 - Initialize** the McASP, the DSP board and the on board codec.
- **Step 2 -** Initialize the discrete time system, that is, specify the initial conditions. Generally zero initial conditions are assumed.
- **Step 3 -** Take sampled data from codec while input is fed to DSP kit from the signal generator. Since Codec is stereo, take average of input data read from left and right channel. Store sampled data at a memory location.
- **Step 4** - Perform filter operation using above said difference equation and store filter Output at a memory location.
- **Step 5 -** Output the value to codec (left channel and right channel) and view the output at Oscilloscope.
- **Step 6 -** Go to step 3.

FLOWCHART FOR IIR IMPLEMENTATION:


Start
Initialize the DSP Board.
Set initial conditions of discrete time system by making x[0]-x[2] and y[0]-y[2] equal to zeros and a0-a2,b0-b2 with MATLAB filter coefficients
Take a new input and store it in x[0].
output = x[0]b0+x[-1]b1+x[-2]b2 - y[-1]a1 - y[-2]a2
Write 'output' to analog i/o.
Do y[-3] = y[-2],y[-2]=y[-1] and Y[-1] = output . x[-3] = x[-2], x[-2]=x[-1] x[-1]=x[0]
Stop

MATLAB PROGRAM TO GENRATE FILTER CO-EFFICIENTS

```
% IIR Low pass Butterworth and Chebyshev filters
% sampling rate - 24000

order = 2;
cf=[2500/12000,8000/12000,1600/12000];

% cutoff frequency - 2500
[num_bw1,den_bw1]=butter(order,cf(1));
[num_cb1,den_cb1]=cheby1(order,3,cf(1));

% cutoff frequency - 8000
[num_bw2,den_bw2]=butter(order,cf(2));
[num_cb2,den_cb2]=cheby1(order,3,cf(2));

fid=fopen('IIR_LP_BW.txt','wt');
fprintf(fid,'\t\t-----------Pass band range: 0-2500Hz----------\n');
fprintf(fid,'\t\t-----------Magnitude response: Monotonic-----\n\n\');
fprintf(fid,'\n float num_bw1[9]={');
fprintf(fid,'%f,%f,%f,%f,%f,\n%f,%f,%f,%f};\n',num_bw1);
fprintf(fid,'\nfloat den_bw1[9]={');
fprintf(fid,'%f,%f,%f,%f,%f,\n%f,%f,%f,%f};\n',den_bw1);

fprintf(fid,'\n\n\n\t\t-----------Pass band range: 0-8000Hz----------\n');
fprintf(fid,'\t\t-----------Magnitude response: Monotonic-----\n\n');
fprintf(fid,'\nfloat num_bw2[9]={');
fprintf(fid,'%f,%f,%f,%f,%f,\n%f,%f,%f,%f};\n',num_bw2);
fprintf(fid,'\nfloat den_bw2[9]={');
fprintf(fid,'%f,%f,%f,%f,%f,\n%f,%f,%f,%f};\n',den_bw2);

fclose(fid);
winopen('IIR_LP_BW.txt');

fid=fopen('IIR_LP_CHEB Type1.txt','wt');
fprintf(fid,'\t\t-----------Pass band range: 2500Hz----------\n');
fprintf(fid,'\t\t-----------Magnitude response: Rippled (3dB) -----\n\n\');
fprintf(fid,'\nfloat num_cb1[9]={');
fprintf(fid,'%f,%f,%f,%f,%f,\n%f,%f,%f,%f};\n',num_cb1);
fprintf(fid,'\nfloat den_cb1[9]={');
fprintf(fid,'%f,%f,%f,%f,%f,\n%f,%f,%f,%f};\n',den_cb1);
fprintf(fid,'\n\n\n\t\t-----------Pass band range: 8000Hz----------\n');
fprintf(fid,'\t\t-----------Magnitude response: Rippled (3dB)-----\n\n');
fprintf(fid,'\nfloat num_cb2[9]={');
fprintf(fid,'%f,%f,%f,%f,%f,\n%f,%f,%f,%f};\n',num_cb2);
fprintf(fid,'\nfloat den_cb2[9]={');
fprintf(fid,'%f,%f,%f,%f,%f,\n%f,%f,%f,%f};\n',den_cb2);
fclose(fid);

winopen('IIR_LP_CHEB Type1.txt');

%%%%%%%%%%%%%%%%%%
figure(1);
[h,w]=freqz(num_bw1,den_bw1);
w=(w/max(w))*12000;
plot(w,20*log10(abs(h)),'linewidth',2)
hold on
[h,w]=freqz(num_cb1,den_cb1);
w=(w/max(w))*12000;
plot(w,20*log10(abs(h)),'linewidth',2,'color','r')
grid on
legend('Butterworth','Chebyshev Type-1');
xlabel('Frequency in Hertz');
ylabel('Magnitude in Decibels');
title('Magnitude response of Low pass IIR filters (Fc=2500Hz)');


figure(2);
[h,w]=freqz(num_bw2,den_bw2);
w=(w/max(w))*12000;
plot(w,20*log10(abs(h)),'linewidth',2)
hold on
[h,w]=freqz(num_cb2,den_cb2);
w=(w/max(w))*12000;
plot(w,20*log10(abs(h)),'linewidth',2,'color','r')
grid on
legend('Butterworth','Chebyshev Type-1 (Ripple: 3dB)');
xlabel('Frequency in Hertz');
ylabel('Magnitude in Decibels');
title('Magnitude response in the passband');
axis([0 12000 -20 20]);
```

IIR_CHEB_LP FILTER CO-EFFICIENTS:

Co-Efficients	Fc=2500Hz		Fc=800Hz		Fc=8000Hz	
	Floating Point Values	Fixed Point Values(Q15)	Floating Point Values	Fixed Point Values(Q15)	Floating Point Values	Fixed Point Values(Q15)
B0	0.044408	1455	0.005147	168	0.354544	11617
B1	0.088815	1455[B1/2]	0.010295	168[B1/2]	0.709088	11617[B1/2]
B2	0.044408	1455	0.005147	168	0.354544	11617
A0	1.000000	32767	1.000000	32767	1.000000	32767
A1	-1.412427	-23140[A1/2]	-1.844881	-30225[A1/2]	0.530009	8683[A1/2]
A2	0.663336	21735	0.873965	28637	0.473218	15506

Note: We have Multiplied Floating Point Values with 32767(2^{15}) to get Fixed Point Values.

IIR_BUTTERWORTH_LP FILTER CO-EFFICIENTS:

Co-Efficients	Fc=2500Hz		Fc=800Hz		Fc=8000Hz	
	Floating Point Values	Fixed Point Values(Q15)	Floating Point Values	Fixed Point Values(Q15)	Floating Point Values	Fixed Point Values(Q15)
B0	0.072231	2366	0.009526	312	0.465153	15241
B1	0.144462	2366[B1/2]	0.019052	312[B1/2]	0.930306	15241[B1/2]
B2	0.072231	2366	0.009526	312	0.465153	15241
A0	1.000000	32767	1.000000	32767	1.000000	32767
A1	-1.109229	-18179[A1/2]	-1.705552,	-27943[A1/2]	0.620204	10161[A1/2]
A2	0.398152	13046	0.743655	24367	0.240408	7877

Note: We have Multiplied Floating Point Values with 32767(2^{15}) to get Fixed Point Values.

IIR_CHEB_HP FILTER CO-EFFICIENTS:

Co-Efficients	Fc=2500Hz		Fc=4000Hz		Fc=7000Hz	
	Floating Point Values	Fixed Point Values(Q15)	Floating Point Values	Fixed Point Values(Q15)	Floating Point Values	Fixed Point Values(Q15)
B0	0.388513	12730	0.282850	9268	0.117279	3842
B1	-0.777027	-12730[B1/2]	-0.565700	-9268[B1/2]	-0.234557	-3842[B1/2]
B2	0.388513	12730	0.282850	9268	0.117279	3842
A0	1.000000	32767	1.000000	32767	1.000000	32767
A1	-1.118450	-18324[A1/2]	-0.451410	-7395[A1/2]	0.754476	12360[A1/2]
A2	0.645091	21137	0.560534	18367	0.588691	19289

Note: We have Multiplied Floating Point Values with 32767(2^{15}) to get Fixed Point Values.

IIR_BUTTERWORTH_HP FILTER CO-EFFICIENTS:

Co-Efficients	Fc=2500Hz		Fc=4000Hz		Fc=7000Hz	
	Floating Point Values	Fixed Point Values(Q15)	Floating Point Values	Fixed Point Values(Q15)	Floating Point Values	Fixed Point Values(Q15)
B0	0.626845	20539	0.465153	15241	0.220195	7215
B1	-1.253691	-20539[B1/2]	-0.930306	-15241[B1/2]	-0.440389	-7215[B1/2]
B2	0.626845	20539	0.465153	15241	0.220195	7215
A0	1.000000	32767	1.000000	32767	1.000000	32767
A1	-1.109229	-18173[A1/2]	-0.620204	-10161[A1/2]	0.307566	5039[A1/2}
A2	0.398152	13046	0.240408	7877	0.188345	6171

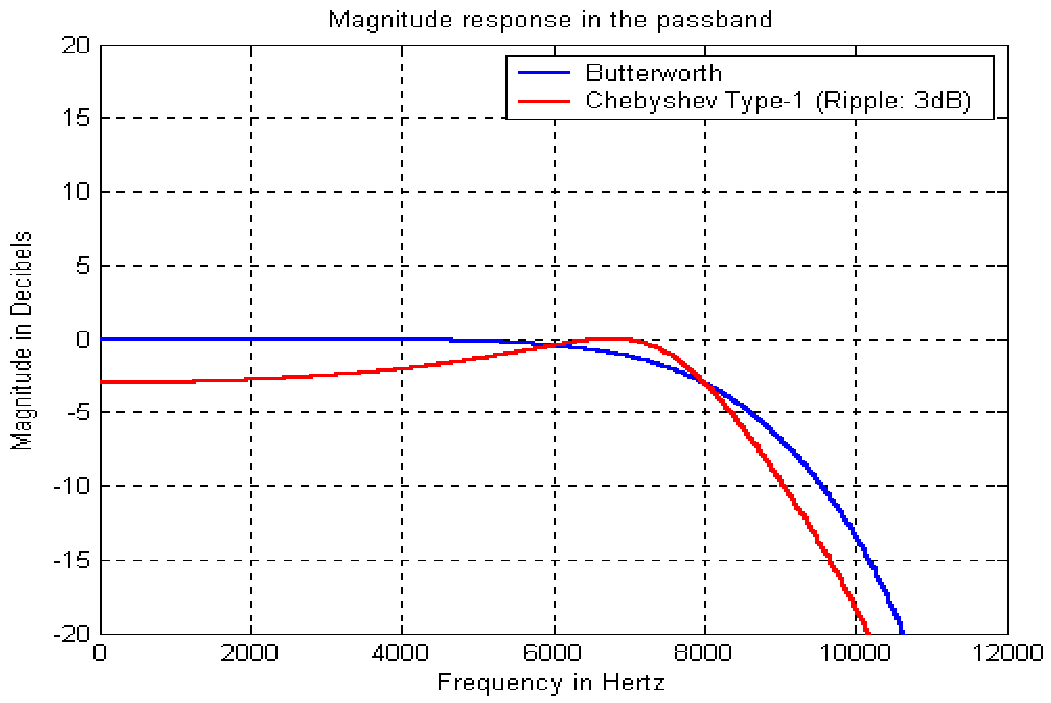
Note: We have Multiplied Floating Point Values with 32767(2^{15}) to get Fixed Point Values.

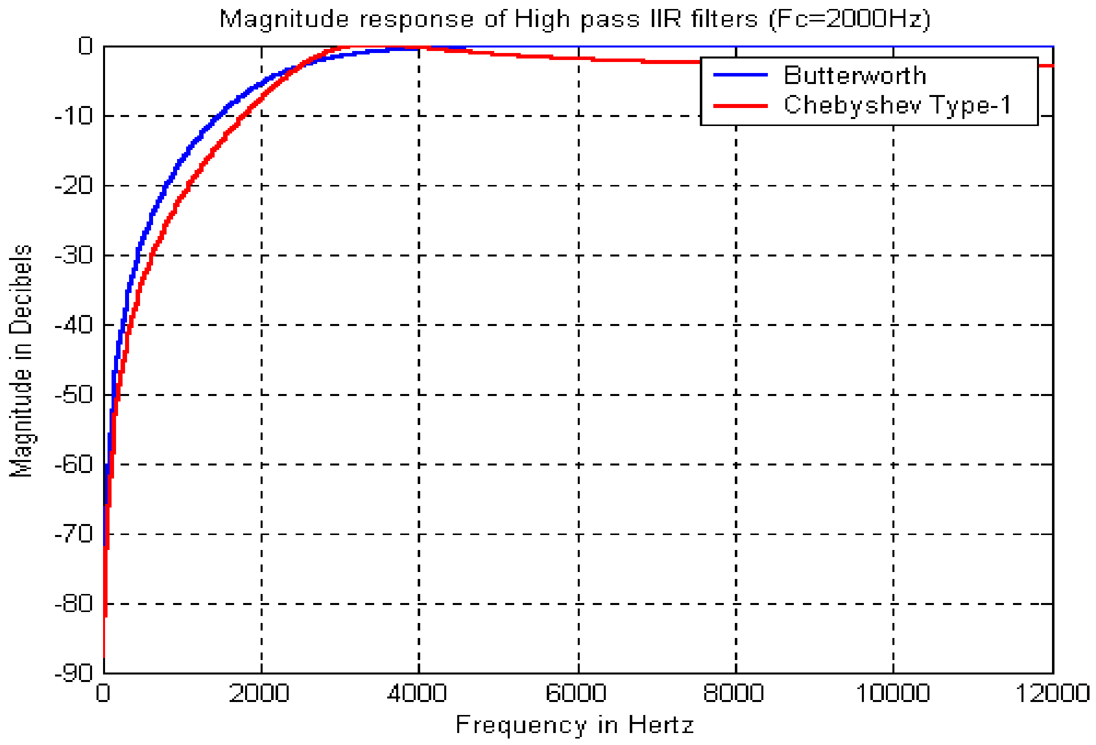
‘C’ PROGRAM TO IMPLEMENT IIR FILTER

```
// L138_iir_intr.c
// IIR filter implemented using second order sections
// integer coefficients read from file
#include "L138_LCDK_aic3106_init.h"
#include "bs1800int.cof"
int w[NUM_SECTIONS][2] = {0};
interrupt void interrupt4()    //interrupt service routine
{
  int section;   // index for section number
  int input;     // input to each section
  int wn,yn;       // intermediate and output values in each stage
  input = input_left_sample();

//  input = (int)prbs();
  for (section=0 ; section< NUM_SECTIONS ; section++)
  {
//
    wn = input - ((a[section][1]*w[section][0])>>15) -
((a[section][2]*w[section][1])>>15);
//
    yn = ((b[section][0]*wn)>>15) +
((b[section][1]*w[section][0])>>15) +
((b[section][2]*w[section][1])>>15);
    w[section][1] = w[section][0];
    w[section][0] = wn;
    input = yn;              // output of current section will be
input to next
  }
  output_left_sample((int16_t)(yn)); // before writing to codec
  return;                         //return from ISR
}

int main(void)
{

L138_initialise_intr(FS_8000_HZ,ADC_GAIN_0DB,DAC_ATTEN_0DB,LCDK_LI
NE_INPUT);
  while(1);
}  // end of main()
```

PROCEDURE:

- Add **bs1800int.cof** file that contains filter co-efficient (will be in IIR filter folder)
- After successful generation of .out file connect target to host PC
- Connect signal generator to LINE INPUT with 2Vp-p Sine Wave frequency is set according to filter co-efficient
- Connect CRO to LINE OUT with volt/div at 1 and time/div at 5msec


Magnitude response in the passband
Butterworth
Chebyshev Type-1 (Ripple: 3dB)
Magnitude in Decibels
Frequency in Hertz
20
15
10
5
0
-5
-10
-15
-20
0
2000
4000
6000
8000
10000
12000

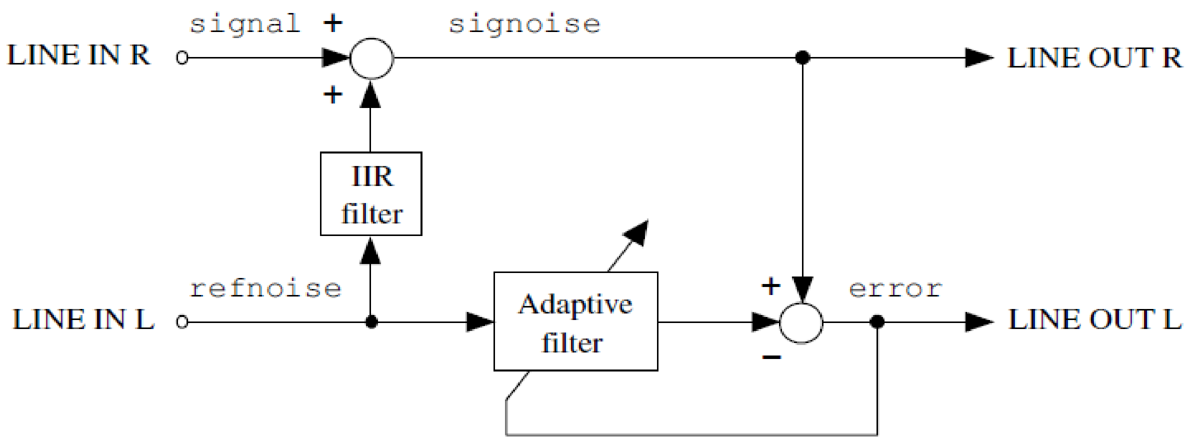

Magnitude response of High pass IIR filters (Fc=2000Hz)
Butterworth
Chebyshev Type-1
Magnitude in Decibels
Frequency in Hertz
0
-10
-20
-30
-40
-50
-60
-70
-80
-90
0
2000
4000
6000
8000
10000
12000

Adaptive Filters

An adaptive filter is a system with a linear filter that has a transfer function controlled by variable parameters and a means to adjust those parameters according to an optimization algorithm. Because of the complexity of the optimization algorithms, almost all adaptive filters are digital filters. Adaptive filters are required for some applications because some parameters of the desired processing operation (for instance, the locations of reflective surfaces in a reverberant space) are not known in advance or are changing. The closed loop adaptive filter uses feedback in the form of an error signal to refine its transfer function.

Generally speaking, the closed loop adaptive process involves the use of a cost function, which is a criterion for optimum performance of the filter, to feed an algorithm, which determines how to modify filter transfer function to minimize the cost on the next iteration. The most common cost function is the mean square of the error signal.

As the power of digital signal processors has increased, adaptive filters have become much more common and are now routinely used in devices such as mobile phones and other communication devices, camcorders and digital cameras, and medical monitoring equipment.



This example cancel an undesired noise signal using external inputs that requires two external inputs, a desired signal and a reference noise signal to be input to left and right channels, respectively. Stereo 3.5mmjack plug to dual RCA jack plug cable is useful for implementing this example using two different signal sources. Alternatively, this may be played through a sound card and input to the experimenter via a stereo 3.5 mm jack plug to 3.5 mm jack plug cable. Speechnoise.wav comprises pseudorandom noise on the left channel and speech on the right channel.

Figure shows the program in block diagram form. Within the program, a primary noise signal correlated with the reference noise signal input on the left channel is formed by passing the reference noise through an IIR filter. The primary noise signal is added to the desired signal (speech) input on the right channel.

Build and run the program and test it using file speechnoise.wav. As adaptation takes place, the output on the left channel of LINE OUT should gradually change from speech plus noise to speech only. You may need to adjust the volume at which you play the file speechnoise.wav. If the input signals are too quiet, then the adaptation may be very slow.

```
// L138_adaptnoise_2IN_iir_intr.c
//

#include "L138_LCDK_aic3106_init.h"
#include "bilinear.cof"

#define beta 1E-12        // learning rate
#define N 128             // number of adaptive filter weights

AIC31_data_type codec_data;

float weights[N];             // adaptive filter weights
float x[N];                   // adaptive filter delay line

float w[NUM_SECTIONS][2];

interrupt void interrupt4(void) // interrupt service routine
{
  short i;
  float input, refnoise, signal, signoise, wn, yn, error;
  int section;

  codec_data.uint = input_sample();
  refnoise =(codec_data.channel[LEFT]); // reference sensor
  signal = (codec_data.channel[RIGHT]); // primary sensor

  input = refnoise;
  for (section=0 ; section<NUM_SECTIONS ; section++)
  {
    wn = input - a[section][1]*w[section][0]
         - a[section][2]*w[section][1];
    yn = b[section][0]*wn + b[section][1]*w[section][0]
       + b[section][2]*w[section][1];
    w[section][1] = w[section][0];
    w[section][0] = wn;
    input = yn;
  }
  signoise = yn + signal;

  yn=0.0;
  x[0] = refnoise;
  for (i = 0; i < N; i++) // compute adaptive filter output
      yn += (weights[i] * x[i]);
error = (signoise) - yn;    // compute error
  for (i = N-1; i >= 0; i--) // update weights and delay line
  {
    weights[i] = weights[i] + beta*error*x[i];
    x[i] = x[i-1];
  }
  codec_data.channel[LEFT]= ((uint16_t)(error));
  codec_data.channel[RIGHT]= ((uint16_t)(error));
  output_sample(codec_data.uint);

  return;
}

int main()
{
  int i;

  for (i = 0; i < N; i++) // initialise weights and delay line
  {
    weights[i] = 0.0;
    x[i] = 0.0;
  }
  for (i = 0 ; i<NUM_SECTIONS ; i++)
  {
    w[i][0]=0.0;
    w[i][1]=0.0;
  }

L138_initialise_intr(FS_8000_HZ,ADC_GAIN_0DB,DAC_ATTEN_0DB,LCDK_LINE_INPUT);
  while(1);
}
```

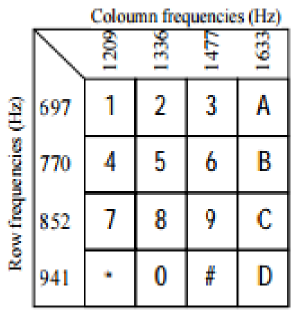
Procedure

- Add **bilinear.cof** hedder file from Adapative folder that contains co-efficent built project to generate .out file
- Connect corrupted signal to LINE INPUT and connect LINE OUT TO Speakers
- Connect target to host PC run the program observe the error free output in speakers

DTMF

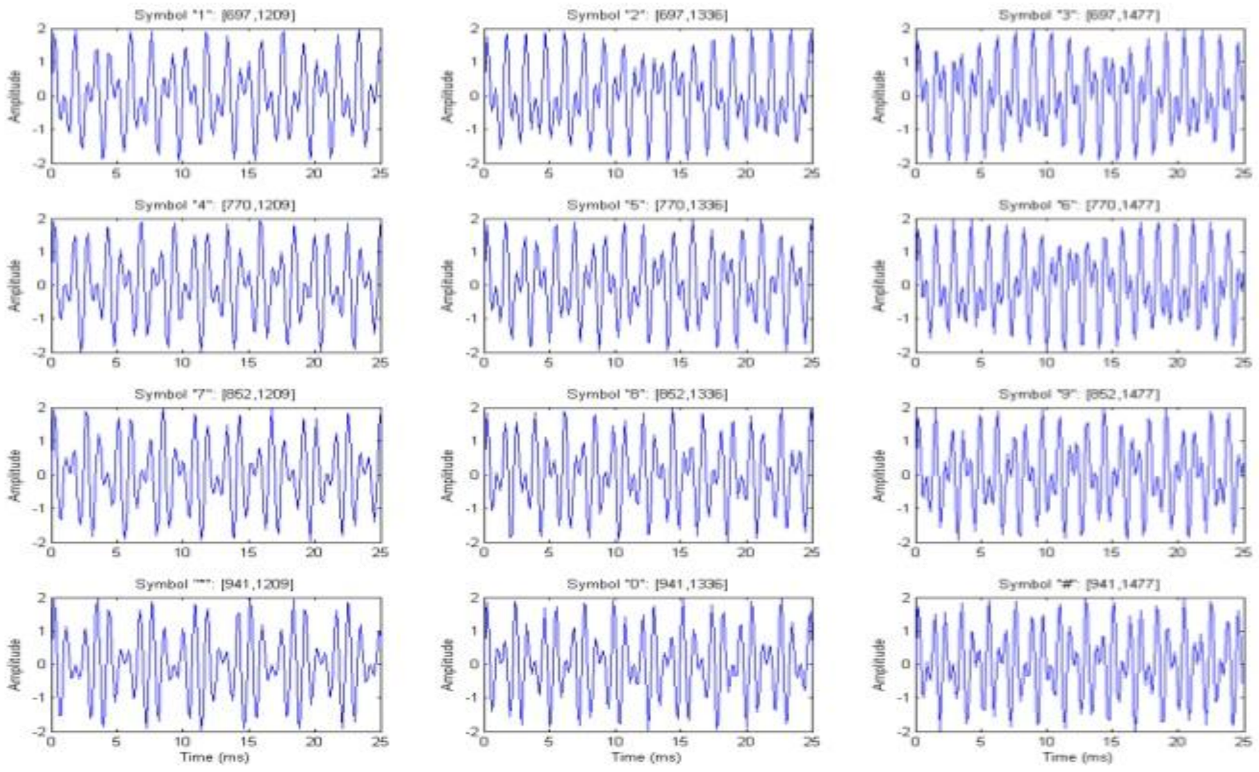
Dual-tone multi-frequency (DTMF) is an international signaling standard for telephone digits. These signals are used in touch-tone telephone call signaling as well as many other areas such as interactive control applications, telephone banking, and pager systems.

A DTMF signal consists of two superimposed sinusoidal waveforms whose frequencies are chosen from a set of eight standardized frequencies. These frequencies were chosen in Bell Laboratories, where DTMF signaling system was originally proposed as an alternative to pulse dialing system in telephony.



	Coloumn frequencies (Hz)			
Row frequencies (Hz)	1209	1336	1477	1633
697	1	2	3	A
770	4	5	6	B
852	7	8	9	C
941	*	0	#	D

WAVEFORMS OF DTMF DIGITS



```c
// L138_sineDTMF_intr.c
//
#include "L138_LCDK_aic3106_init.h"
#include <math.h>
#define PI 3.14159265358979
#define TABLESIZE 512        // size of look up table
#define SAMPLING_FREQ 16000
#define STEP_770  (float)(770 * TABLESIZE)/SAMPLING_FREQ
#define STEP_1336  (float)(1336 * TABLESIZE)/SAMPLING_FREQ
#define STEP_697  (float)(697 * TABLESIZE)/SAMPLING_FREQ
#define STEP_852  (float)(852 * TABLESIZE)/SAMPLING_FREQ
#define STEP_941  (float)(941 * TABLESIZE)/SAMPLING_FREQ
#define STEP_1209  (float)(1209 * TABLESIZE)/SAMPLING_FREQ
#define STEP_1477  (float)(1477 * TABLESIZE)/SAMPLING_FREQ
#define STEP_1633  (float)(1633 * TABLESIZE)/SAMPLING_FREQ
int16_t sine_table[TABLESIZE];
float loopindexlow = 0.0;
float loopindexhigh = 0.0;
int16_t i;
interrupt void interrupt4(void)  // interrupt service routine
{
output_left_sample(sine_table[(int16_t)loopindexlow] +
sine_table[(int16_t)loopindexhigh]);
  loopindexlow += STEP_770;
  if (loopindexlow > (float)TABLESIZE)
    loopindexlow -= (float)TABLESIZE;
  loopindexhigh += STEP_1477;
  if (loopindexhigh > (float)TABLESIZE)
 loopindexhigh -= (float)TABLESIZE;
  return;
}
int main(void)
{

L138_initialise_intr(FS_16000_HZ,ADC_GAIN_0DB,DAC_ATTEN_0DB,LCDK_LINE_INPUT
);
  for (i=0 ; i< TABLESIZE ; i++)
    sine_table[i] = (short)(10000.0*sin(2*PI*i/TABLESIZE));
  while(1);
}
```

Procedure:

- After successful generation of .out file connect the kit
- Connect mobile phone to LINE INPUT using 3.5mm stereo Jack
- Connect CRO to LINE OUT
- Press any key in phone and observe corresponding frequency in CRO

Generation of real time Signals (Sine, Square & ramp)

Sine wave

```
// L138_sine_intr.c
//

#include "L138_LCDK_aic3106_init.h"
#include "math.h"

#define SAMPLING_FREQ 8000
#define PI 3.14159265358979

float frequency = 1000.0;
float amplitude = 20000.0;
float theta_increment;
float theta = 0.0;

interrupt void interrupt4(void) // interrupt service routine
{
  theta_increment = 2*PI*frequency/SAMPLING_FREQ;
  theta += theta_increment;
  if (theta > 2*PI) theta -= 2*PI;
  output_left_sample((int16_t)(amplitude*sin(theta)));
  return;
}

int main(void)
{

L138_initialise_intr(FS_8000_HZ,ADC_GAIN_0DB,DAC_ATTEN_0DB,LCDK_LINE_INPUT);
  while(1);
}
```

Square wave Generation

```
// L138_squarewave_intr.c
//

#include "L138_LCDK_aic3106_init.h"
#define LOOPLENGTH 64

int16_t square_table[LOOPLENGTH] =
{10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
10000, 10000, 10000, 10000, 10000, 10000, 10000, 10000,
-10000,-10000,-10000,-10000,-10000,-10000,-10000,-10000,
-10000,-10000,-10000,-10000,-10000,-10000,-10000,-10000,
-10000,-10000,-10000,-10000,-10000,-10000,-10000,-10000,
-10000,-10000,-10000,-10000,-10000,-10000,-10000,-10000};

int16_t loopindex = 0;

interrupt void interrupt4(void) // interrupt service routine
{
  output_left_sample(square_table[loopindex++]);
  if (loopindex >= LOOPLENGTH)
    loopindex = 0;
  return;
}

int main(void)
{

L138_initialise_intr(FS_8000_HZ,ADC_GAIN_0DB,DAC_ATTEN_0DB,LCDK_LINE_INPUT);
  while(1);
}
```

Ramp Wave Generation

```
// L138_ramp_intr.c
//

#include "L138_LCDK_aic3106_init.h"
#define LOOPLENGTH 64

int16_t output = 0;

interrupt void interrupt4(void) // interrupt service routine
{
  output_left_sample(output);    // output to L DAC
  output += 2000;                // increment output value
  if (output >= 30000)           // if peak is reached
  output = -30000;               // reinitialize
  return;
}

int main(void)
{

L138_initialise_intr(FS_8000_HZ,ADC_GAIN_0DB,DAC_ATTEN_0DB,LCDK_LINE_INPUT);
  while(1);
}
```

Procedure

- After successful generation of .out file connect kit to host PC and run program
- Connect Line out to CRO to observe Waves

✓ To vary frequency and amplitude in sine wave change the value of Frequency and amplitude

✓ To vary amplitude of Square wave change the value of lookup table and to vary frequency change sampling frequency

✓ To vary amplitude of Ramp wave change the value of peak amplitude and to vary frequency change the value of sampling frequency

Generation of AM signal

```
// L138_am_poll.c
//

#include "L138_LCDK_aic3106_init.h"

short amp = 20;    //index for modulation

int main(void)
{
 int16_t baseband[20]={1000,951,809,587,309,0,-309,
                       -587,-809,-951,-1000,-951,-809,
                     -587,-309,0,309,587,809,951}; // base band signal
 int16_t carrier[20] ={1000,0,-1000,0,1000,0,-1000,
                        0,1000,0,-1000,0,1000,0,-1000,
                        0,1000,0,-1000,0}; // 2 kHz
 int16_t output[20];
 int16_t k;


L138_initialise_poll(FS_8000_HZ,ADC_GAIN_0DB,DAC_ATTEN_0DB,LCDK_LINE_INPUT);

 while(1)
 {
  for  (k=0;  k<20;  k++)
  {
   output[k]= carrier[k]  +  ((amp*baseband[k]*carrier[k]/10)>>12);
   output_left_sample(20*output[k]);
  }
 }
}
```

Procedure:

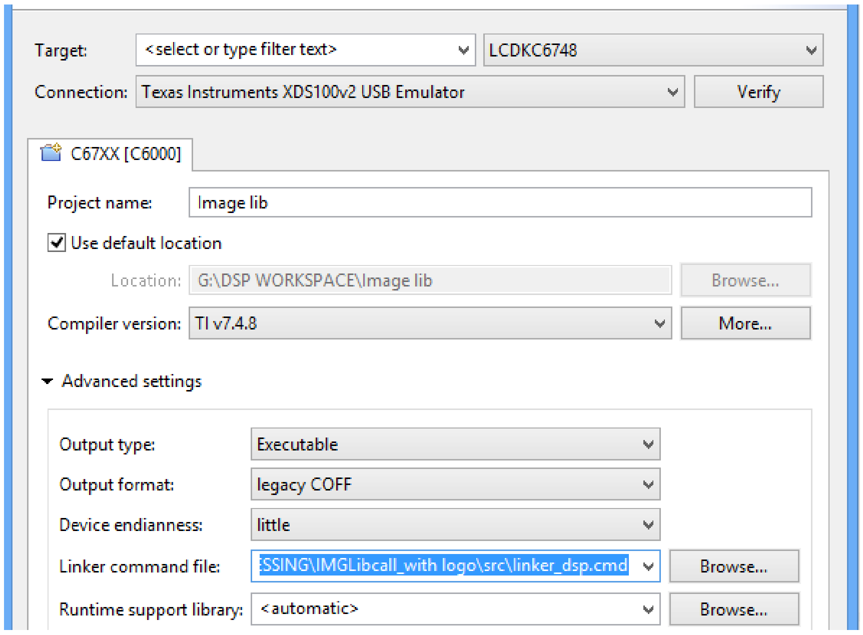
- Hear ISR is executing through Polling method so add vectors_poll.asm instead of vectors_intr.asm
- After successful generation of .out file connect target to Host PC and run program
- Connect LINE OUT to CRO to observe AM wave

✓ Carrier frequency: Carrier holds 20 samples of 5 cycles of a sinusoidal carrier signal with a frequency of 5 Fs/20 = 2 KHz

IMAGE PROCESSING WITH LIBRARY FUNCTIONS:

Create a project:

Goto File ->new->CCS project and enter the details as below.In “Advanced Settings” choose the Linker command file: as “**linker_dsp.cmd**” (Browse from ‘IMGLibcall_with logo\src’ folder given at the time of installation)



Target: <select or type filter text> LCDKC6748
Connection: Texas Instruments XDS100v2 USB Emulator
C67XX [C6000]
Project name: Image lib
☑ Use default location
Location: G:\DSP WORKSPACE\Image lib
Compiler version: TI v7.4.8
Advanced settings
Output type: Executable
Output format: legacy COFF
Device endianness: little
Linker command file: :SSING\IMGLibcall_with logo\src\linker_dsp.cmd
Runtime support library: <automatic>

Adding the files:

- Right click on project name-> Add files..-> Add ‘IMAGEMAIN.c’ and ‘STARCOMLOGO.c’ from ‘src’ folder.
- Add all the files from ‘src\lib files’.
- Add ‘wolf.c’ from ‘bitmaps’ folder.

3. Build and debug the project.

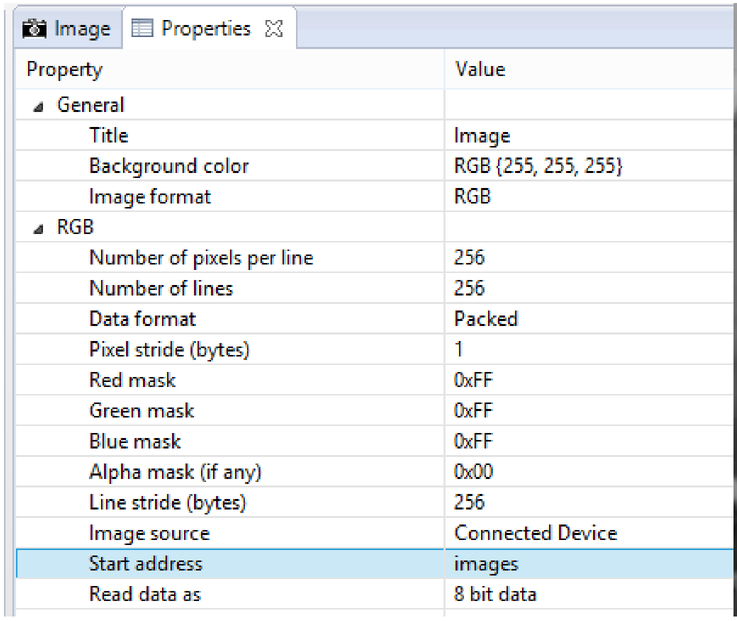
4. Before running, create breakpoints after each line starting from 78th line.

5. Every time you change the breakpoint, restart the program and Resume it.

6. Also refresh the image to see the change.

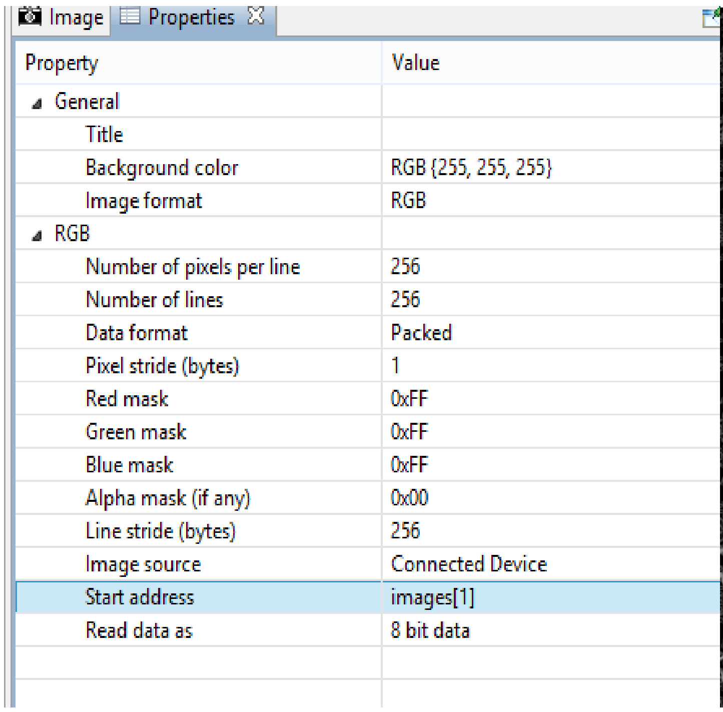
7. Plot image by Tools-> Image Analyzer and follow the below details:

Plot for input image:

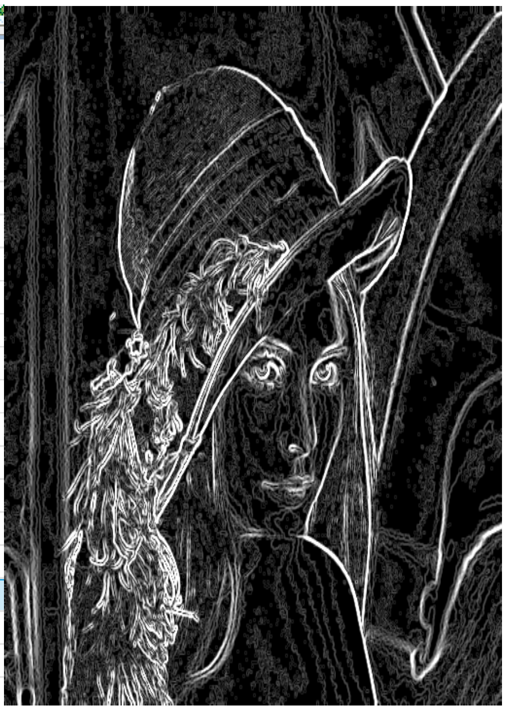


Property	Value
◢ General	
Title	Image
Background color	RGB {255, 255, 255}
Image format	RGB
◢ RGB	
Number of pixels per line	256
Number of lines	256
Data format	Packed
Pixel stride (bytes)	1
Red mask	0xFF
Green mask	0xFF
Blue mask	0xFF
Alpha mask (if any)	0x00
Line stride (bytes)	256
Image source	Connected Device
Start address	images
Read data as	8 bit data

Plot for output image:



Property	Value
◢ General	
Title	
Background color	RGB {255, 255, 255}
Image format	RGB
◢ RGB	
Number of pixels per line	256
Number of lines	256
Data format	Packed
Pixel stride (bytes)	1
Red mask	0xFF
Green mask	0xFF
Blue mask	0xFF
Alpha mask (if any)	0x00
Line stride (bytes)	256
Image source	Connected Device
Start address	images[1]
Read data as	8 bit data

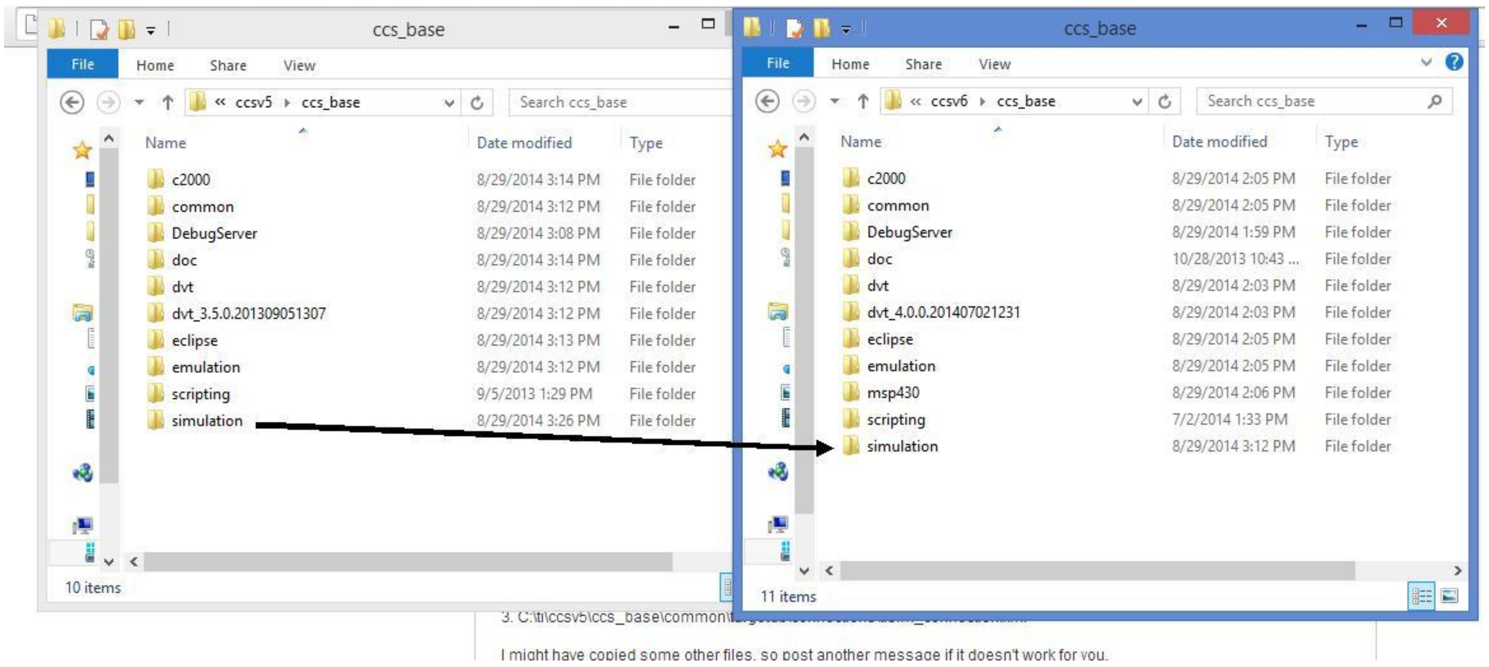


APPENDIX

SIMULATOR IN CCS V6

- Install CCS V6 and CCS V5
- The installer automatically places them in a folder named “ti” in your C drive
- Copy the following files from CCSV5 to CCS V6

C:\ti\ccsv5\ccs_base\simulation<--- complete directory!



ccs_base — « ccsv5 › ccs_base

Name	Date modified	Type
c2000	8/29/2014 3:14 PM	File folder
common	8/29/2014 3:12 PM	File folder
DebugServer	8/29/2014 3:08 PM	File folder
doc	8/29/2014 3:14 PM	File folder
dvt	8/29/2014 3:12 PM	File folder
dvt_3.5.0.201309051307	8/29/2014 3:12 PM	File folder
eclipse	8/29/2014 3:13 PM	File folder
emulation	8/29/2014 3:12 PM	File folder
scripting	9/5/2013 1:29 PM	File folder
simulation	8/29/2014 3:26 PM	File folder

10 items

ccs_base — « ccsv6 › ccs_base

Name	Date modified	Type
c2000	8/29/2014 2:05 PM	File folder
common	8/29/2014 2:05 PM	File folder
DebugServer	8/29/2014 1:59 PM	File folder
doc	10/28/2013 10:43 ...	File folder
dvt	8/29/2014 2:03 PM	File folder
dvt_4.0.0.201407021231	8/29/2014 2:03 PM	File folder
eclipse	8/29/2014 2:05 PM	File folder
emulation	8/29/2014 2:05 PM	File folder
msp430	8/29/2014 2:06 PM	File folder
scripting	7/2/2014 1:33 PM	File folder
simulation	8/29/2014 3:12 PM	File folder

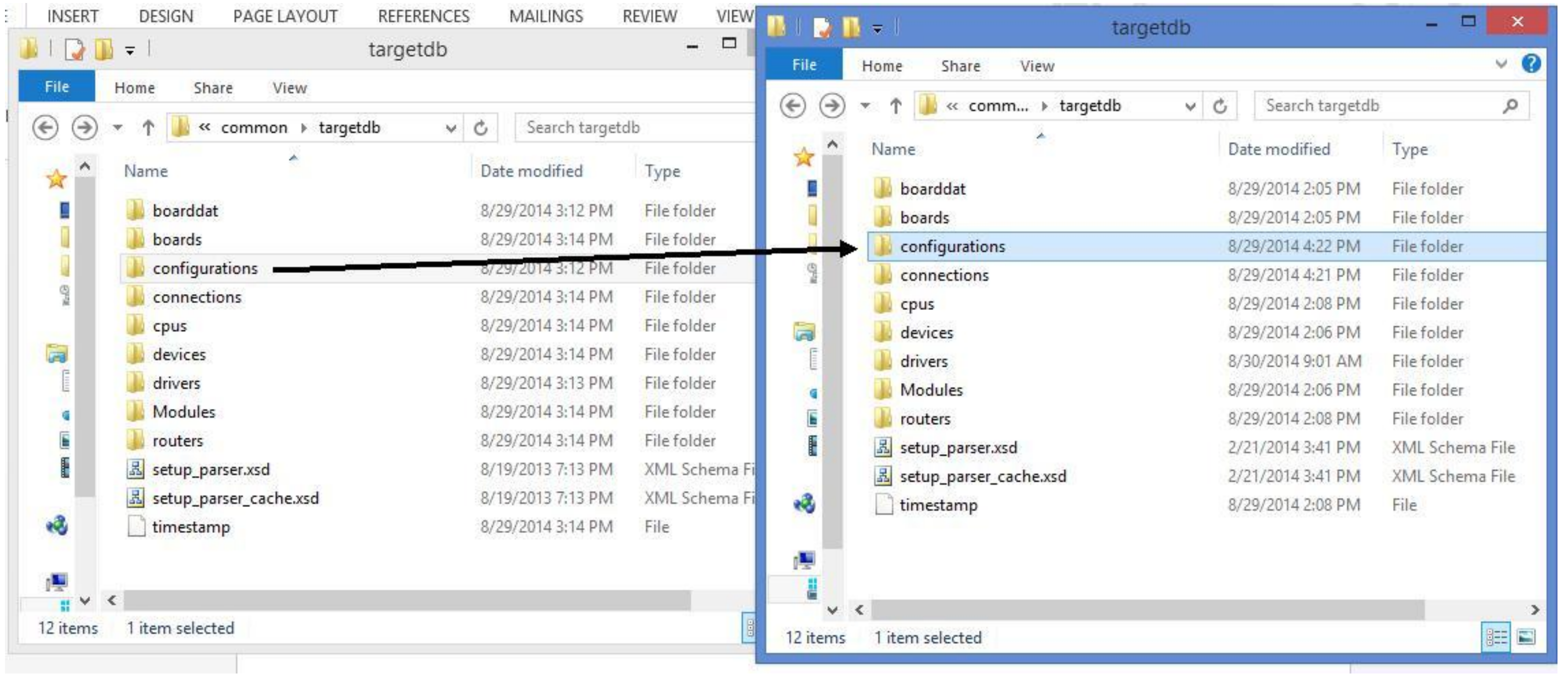
11 items

3. C:\ti\ccsv5\ccs_base\common\

I might have copied some other files, so post another message if it doesn't work for you.

Copy over the Configurations file as well

C:\ti\ccsv5\ccs_base\common\targetdb\configurations



targetdb — « common › targetdb

Name	Date modified	Type
boarddat	8/29/2014 3:12 PM	File folder
boards	8/29/2014 3:14 PM	File folder
configurations	8/29/2014 3:12 PM	File folder
connections	8/29/2014 3:14 PM	File folder
cpus	8/29/2014 3:14 PM	File folder
devices	8/29/2014 3:14 PM	File folder
drivers	8/29/2014 3:13 PM	File folder
Modules	8/29/2014 3:14 PM	File folder
routers	8/29/2014 3:14 PM	File folder
setup_parser.xsd	8/19/2013 7:13 PM	XML Schema Fi
setup_parser_cache.xsd	8/19/2013 7:13 PM	XML Schema Fi
timestamp	8/29/2014 3:14 PM	File

12 items 1 item selected

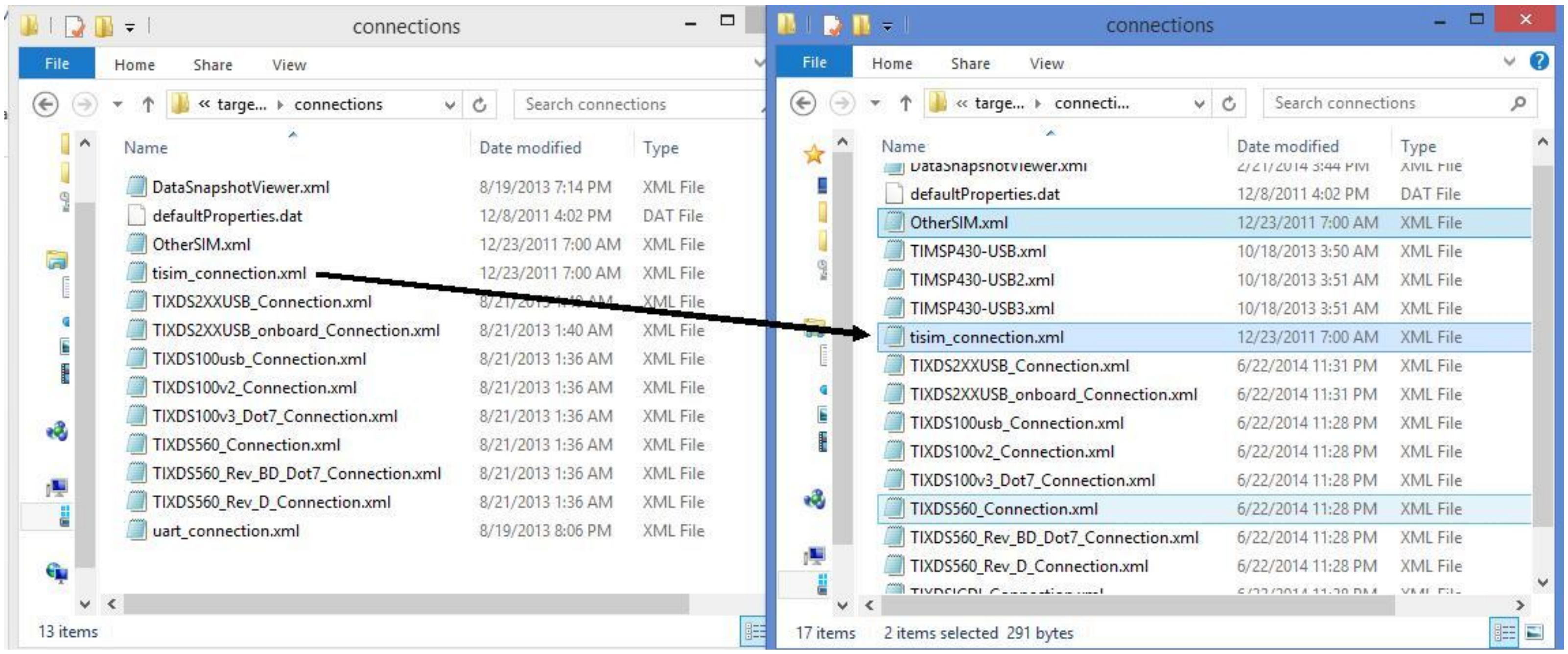
targetdb — « comm... › targetdb

Name	Date modified	Type
boarddat	8/29/2014 2:05 PM	File folder
boards	8/29/2014 2:05 PM	File folder
configurations	8/29/2014 4:22 PM	File folder
connections	8/29/2014 4:21 PM	File folder
cpus	8/29/2014 2:08 PM	File folder
devices	8/29/2014 2:06 PM	File folder
drivers	8/30/2014 9:01 AM	File folder
Modules	8/29/2014 2:06 PM	File folder
routers	8/29/2014 2:08 PM	File folder
setup_parser.xsd	2/21/2014 3:41 PM	XML Schema File
setup_parser_cache.xsd	2/21/2014 3:41 PM	XML Schema File
timestamp	8/29/2014 2:08 PM	File

12 items 1 item selected

Copy over the v5 simulator connection file into the v6 connections file

C:\ti\ccsv5\ccs_base\common\targetdb\connections\tisim_connection.xml



connections — « targe... › connections

Name	Date modified	Type
DataSnapshotViewer.xml	8/19/2013 7:14 PM	XML File
defaultProperties.dat	12/8/2011 4:02 PM	DAT File
OtherSIM.xml	12/23/2011 7:00 AM	XML File
tisim_connection.xml	12/23/2011 7:00 AM	XML File
TIXDS2XXUSB_Connection.xml	8/21/2013 1:40 AM	XML File
TIXDS2XXUSB_onboard_Connection.xml	8/21/2013 1:40 AM	XML File
TIXDS100usb_Connection.xml	8/21/2013 1:36 AM	XML File
TIXDS100v2_Connection.xml	8/21/2013 1:36 AM	XML File
TIXDS100v3_Dot7_Connection.xml	8/21/2013 1:36 AM	XML File
TIXDS560_Connection.xml	8/21/2013 1:36 AM	XML File
TIXDS560_Rev_BD_Dot7_Connection.xml	8/21/2013 1:36 AM	XML File
TIXDS560_Rev_D_Connection.xml	8/21/2013 1:36 AM	XML File
uart_connection.xml	8/19/2013 8:06 PM	XML File

13 items

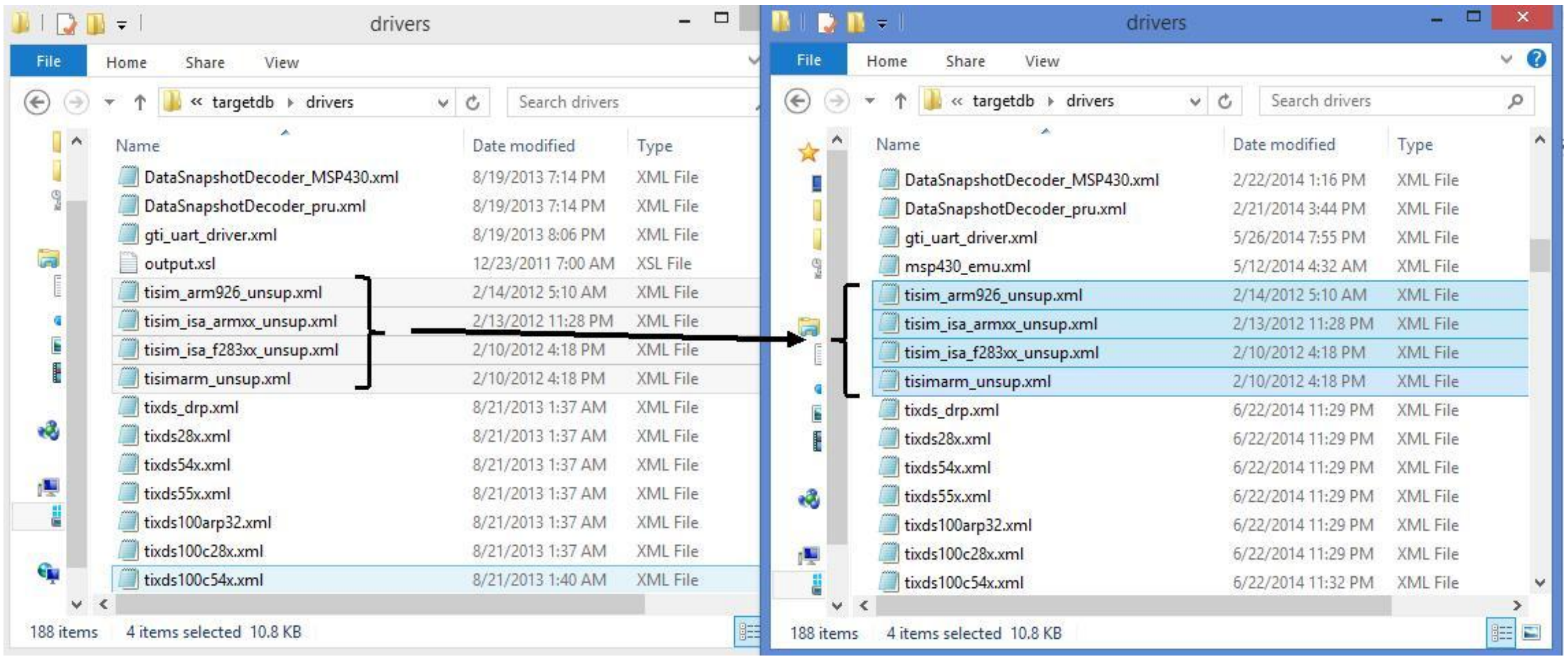
connections — « targe... › connecti...

Name	Date modified	Type
DataSnapshotViewer.xml	2/21/2014 3:44 PM	XML File
defaultProperties.dat	12/8/2011 4:02 PM	DAT File
OtherSIM.xml	12/23/2011 7:00 AM	XML File
TIMSP430-USB.xml	10/18/2013 3:50 AM	XML File
TIMSP430-USB2.xml	10/18/2013 3:51 AM	XML File
TIMSP430-USB3.xml	10/18/2013 3:51 AM	XML File
tisim_connection.xml	12/23/2011 7:00 AM	XML File
TIXDS2XXUSB_Connection.xml	6/22/2014 11:31 PM	XML File
TIXDS2XXUSB_onboard_Connection.xml	6/22/2014 11:31 PM	XML File
TIXDS100usb_Connection.xml	6/22/2014 11:28 PM	XML File
TIXDS100v2_Connection.xml	6/22/2014 11:28 PM	XML File
TIXDS100v3_Dot7_Connection.xml	6/22/2014 11:28 PM	XML File
TIXDS560_Connection.xml	6/22/2014 11:28 PM	XML File
TIXDS560_Rev_BD_Dot7_Connection.xml	6/22/2014 11:28 PM	XML File
TIXDS560_Rev_D_Connection.xml	6/22/2014 11:28 PM	XML File

17 items 2 items selected 291 bytes

The last files to copy over are the simulation drivers needed to run the TI Simulator

C:\ti\ccsv5\ccs_base\common\targetdb\drivers



drivers — « targetdb › drivers

Name	Date modified	Type
DataSnapshotDecoder_MSP430.xml	8/19/2013 7:14 PM	XML File
DataSnapshotDecoder_pru.xml	8/19/2013 7:14 PM	XML File
gti_uart_driver.xml	8/19/2013 8:06 PM	XML File
output.xsl	12/23/2011 7:00 AM	XSL File
tisim_arm926_unsup.xml	2/14/2012 5:10 AM	XML File
tisim_isa_armxx_unsup.xml	2/13/2012 11:28 PM	XML File
tisim_isa_f283xx_unsup.xml	2/10/2012 4:18 PM	XML File
tisimarm_unsup.xml	2/10/2012 4:18 PM	XML File
tixds_drp.xml	8/21/2013 1:37 AM	XML File
tixds28x.xml	8/21/2013 1:37 AM	XML File
tixds54x.xml	8/21/2013 1:37 AM	XML File
tixds55x.xml	8/21/2013 1:37 AM	XML File
tixds100arp32.xml	8/21/2013 1:37 AM	XML File
tixds100c28x.xml	8/21/2013 1:37 AM	XML File
tixds100c54x.xml	8/21/2013 1:40 AM	XML File

188 items 4 items selected 10.8 KB

drivers — « targetdb › drivers

Name	Date modified	Type
DataSnapshotDecoder_MSP430.xml	2/22/2014 1:16 PM	XML File
DataSnapshotDecoder_pru.xml	2/21/2014 3:44 PM	XML File
gti_uart_driver.xml	5/26/2014 7:55 PM	XML File
msp430_emu.xml	5/12/2014 4:32 AM	XML File
tisim_arm926_unsup.xml	2/14/2012 5:10 AM	XML File
tisim_isa_armxx_unsup.xml	2/13/2012 11:28 PM	XML File
tisim_isa_f283xx_unsup.xml	2/10/2012 4:18 PM	XML File
tisimarm_unsup.xml	2/10/2012 4:18 PM	XML File
tixds_drp.xml	6/22/2014 11:29 PM	XML File
tixds28x.xml	6/22/2014 11:29 PM	XML File
tixds54x.xml	6/22/2014 11:29 PM	XML File
tixds55x.xml	6/22/2014 11:29 PM	XML File
tixds100arp32.xml	6/22/2014 11:29 PM	XML File
tixds100c28x.xml	6/22/2014 11:29 PM	XML File
tixds100c54x.xml	6/22/2014 11:32 PM	XML File

188 items 4 items selected 10.8 KB

Now that all of the necessary files are in the v6 directory, run CCSv6.