**Department of Electrical Engineering**

|  |  |
| --- | --- |
| **Faculty Member:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** | **Dated: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** |
|  |  |
| **Course/Section:BEE8** | **Semester: Spring 2019** |
|  |  |

**EE-330 Digital Signal Processing**

**Lab 5: Audio Processing using TMS 320C6713 DSK**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | **PLO4** | | **PLO5** | **PLO8** | **PLO9** |
| **Name** | **Reg. No** | **Viva / Quiz / Lab Performance** | **Analysis of data in Lab Report** | **Modern Tool Usage** | **Ethics and Safety** | **Individual and Team Work** |
|  |  | **5 Marks** | **5 Marks** | **5 Marks** | **5 Marks** | **5 Marks** |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

# Lab5: Audio Processing using TMS 320C6713 DSK

**Objectives**

The objective of this lab is to explore some more features of Code Composer Studio (CCS) that is time domain and frequency domain plots of audio signal in addition to that we will also do real time processing of audio input.

* Time domain and frequency Domain Plots in CCS
* Real Time processing of Audio Signal
* Working with basic sinusoids on DSP Kit

**Lab Instructions**

**Lab Instructions**

* The students should perform and demonstrate each lab task separately for step-wise evaluation (please ensure that course instructor/lab engineer has signed each step after ascertaining its functional verification)
* Each group shall submit one lab report on LMS within 6 days after lab is conducted. Lab report submitted via email will not be graded.
* . Students are however encouraged to practice on their own in spare time for enhancing their skills.

**Lab Report Instructions**

All questions should be answered precisely to get maximum credit. Lab report must ensure following items:

* Lab objectives
* C codes
* Results (graphs/tables) duly commented and discussed
* Conclusion

## Introduction

In this Lab we will explore some more features of Code Composer Studio that is time domain and frequency domain plots of audio signal in addition to that we will also learn real time processing of audio input. We will also see the details of support files that we used in previous Lab.

## SUPPORT FILES

The support files *c6713dskinit.c* ,*vectors\_intr.asm* or *vectors\_poll.asm* , and *c6713dsk.cmd* are used by nearly all of the examples .

### Initialization/Communication File ( *c6713dskinit.c*)

Source file *c6713dskinit.c* contains the definitions of a number of functions used to initialize the DSK. Calls are made from these functions to lower level functions provided with CCS in the board support library (BSL) and chip support library (CSL) files dsk6713bsl.lib and csl6713.lib . Functions *comm\_intr()* and *comm\_poll()* initialize communications between the C6713 processor and the AIC23 codec for either interrupt - driven or polling - based input and output. In the case of interrupt - driven input and output, interrupt #11 (INT11), generated by the codec via the serial port (McBSP), is configured and enabled (selected). The nonmaskable interrupt bit must be enabled as well as the global interrupt enable (GIE) bit.Functions*input\_sample()* , *input\_left\_sample()* , *input\_right\_sample()* , *output\_sample()* , *output\_left\_sample()* , and *input\_right\_sample()* are used to read and write data to and from the codec. In the case of polling – based input and output, these functions wait until the next sampling instant (determinedby the codec) before reading or writing, using the lower level functions MCBSP\_read() or MCBSP\_write() . They do this by polling (testing) the receive ready (RRDY) or transmit ready (XRDY) bits of the McBSP control register (SPCR). In the case of interrupt - driven input and output, the processor is interrupted by the codec at each sampling instant and when either *input\_sample()* or *output\_sample()* is called from within the interrupt service routine, reading or writing proceeds without RRDY or XRDY being tested. Header File ( *c6713dskinit.h*)The corresponding header support file *c6713dskinit.h* contains function prototypes as well as initial settings for the control registers of the AIC23 codec. Nearly all of the example programs in use the same AIC23 control register settings. However, two codec parameters — namely, sampling frequency and selectionof ADC input (LINE IN or MIC IN) — are changed more often, from one program example to another, and for that reason the following mechanism has been adopted. During initialization of the DSK (in function dsk\_init() , defined in file c6713dskinit.c ), the AIC23 codec control registers are initialized using the DSK6713\_AIC23\_Config type data structure config defined in header file c6713dskinit.h . Immediately following this initialization, two functions DSK6713\_AIC23\_setFreq() and DSK6713\_AIC23\_rset() are called and these set the sampling frequency and select the input source according to the values of the variablesfs and inputsource . These values are set in the first few lines of every top level source file; for example,

Uint32 fs = DSK6713\_AIC23\_FREQ\_8KHZ; //set sampling rate

#define DSK6713\_AIC23\_INPUT\_MIC 0x0015

#define DSK6713\_AIC23\_INPUT\_LINE 0x0011

Uint16 inputsource=DSK6713\_AIC23\_INPUT\_MIC; //select input source

In this way, the sampling frequency and input source can be changed without having to edit either c6713sdkinit.h or c6713dskinit.c .

**Function defined in these files**

*void c6713\_dsk\_init();*

*voidcomm\_poll();*

*voidcomm\_intr();*

*voidoutput\_sample(int);*

*voidoutput\_left\_sample(short);*

*voidoutput\_right\_sample(short);*

*Uint32 input\_sample();*

*shortinput\_left\_sample();*

*shortinput\_right\_sample();*

### Vector Files ( vectors\_intr.asm, vectors\_poll.asm )

To make use of interrupt INT11, a branch instruction (jump) to the interrupt service routine (ISR) c\_int11() defined in a C program, for example, sine8\_buf.c , must be placed at the appropriate point in the interrupt service table (IST). Assembly language file vectors\_intr.asm , which sets up the IST, Note the underscore preceding the name of the routine or function being called. By convention, this indicates a C function. For a polling - based program, file vectors\_poll.asm is used, in place of vectors\_intr.asm . The main difference between these files is that there is no branch to c\_int11() in the IST set up by vectors\_poll.asm . Common to both files is a branch to c\_int00() , the start of a C program, associated with reset.

### Linker Command File ( c6713dsk.cmd)

Linker command file *C6713dsk.cmd* specifies the memory configuration of the internal and external memory available on the DSK and the mapping of sections of code and data to absolute addresses in that memory. For example, the *.text* section, produced by the C compiler, is mapped into IRAM, that is, the internal memory of the C6713 digital signal processor, starting at address 0x00000220 . The section .vectors created by vectors\_intr.asm or by vectors\_poll.asm is mapped into IVECS, that is, internal memory starting at address

***Generation of Sinusoid and Plotting with CCS ( sine8\_buf )***

This example generates a sinusoidal analog output signal using eight precalculated and prestored sample values. However, it differs fundamentally from sine8\_LED in that its operation is based on the use of interrupts. In addition, it uses a buffer to store the BUFFERLENGTH most recent output samples. It is used to illustrate the capabilities of CCS for plotting data in both time and frequency domains. All the files necessary to build and run an executable file sine8\_BUF.out are stored in folder sine8\_buf . Program file*sine8\_buf.c* is listed in Figure 3.12 . Because a project file sine8\_buf.pjt is supplied, there is no need to create a new

*//****sine8\_buf.c*** *sine generation with output stored in buffer*

*#include "DSK6713\_AIC23.h" //codec support*

*Uint32 fs=DSK6713\_AIC23\_FREQ\_8KHZ; //set sampling rate*

*#define DSK6713\_AIC23\_INPUT\_MIC 0x0015*

*#define DSK6713\_AIC23\_INPUT\_LINE 0x0011*

*Uint16 inputsource=DSK6713\_AIC23\_INPUT\_MIC; // select input*

*#define LOOPLENGTH 8*

*#define BUFFERLENGTH 256*

*intloopindex = 0; //table index*

*intbufindex = 0; //buffer index*

*shortsine\_table[LOOPLENGTH]={0,707,1000,707,0,-707,-1000,-707};*

*intout\_buffer[BUFFERLENGTH]; //output buffer*

*short gain = 10;*

*interrupt void c\_int11() //interrupt service routine*

*shortout\_sample;*

*out\_sample = sine\_table[loopindex++]\*gain;*

*output\_left\_sample(out\_sample); //output sample value*

*out\_buffer[bufindex++] = out\_sample; //store in buffer*

*if (loopindex>= LOOPLENGTH) loopindex = 0; //check end table*

*if (bufindex>= BUFFERLENGTH) bufindex = 0; //check end buffer*

*return;*

*} //return from interrupt*

*void main()*

*{*

*comm\_intr(); //initialise DSK*

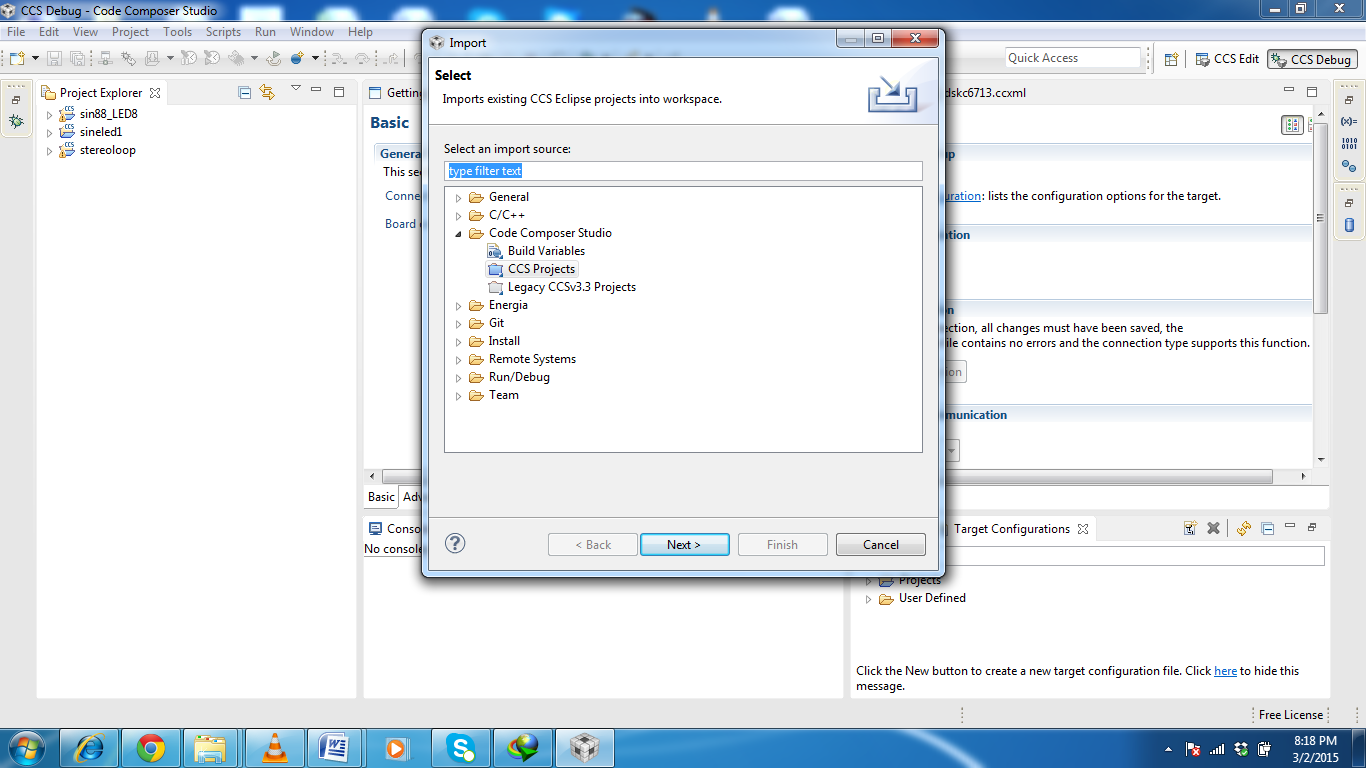
*while(1); //infinite loop*

*}*

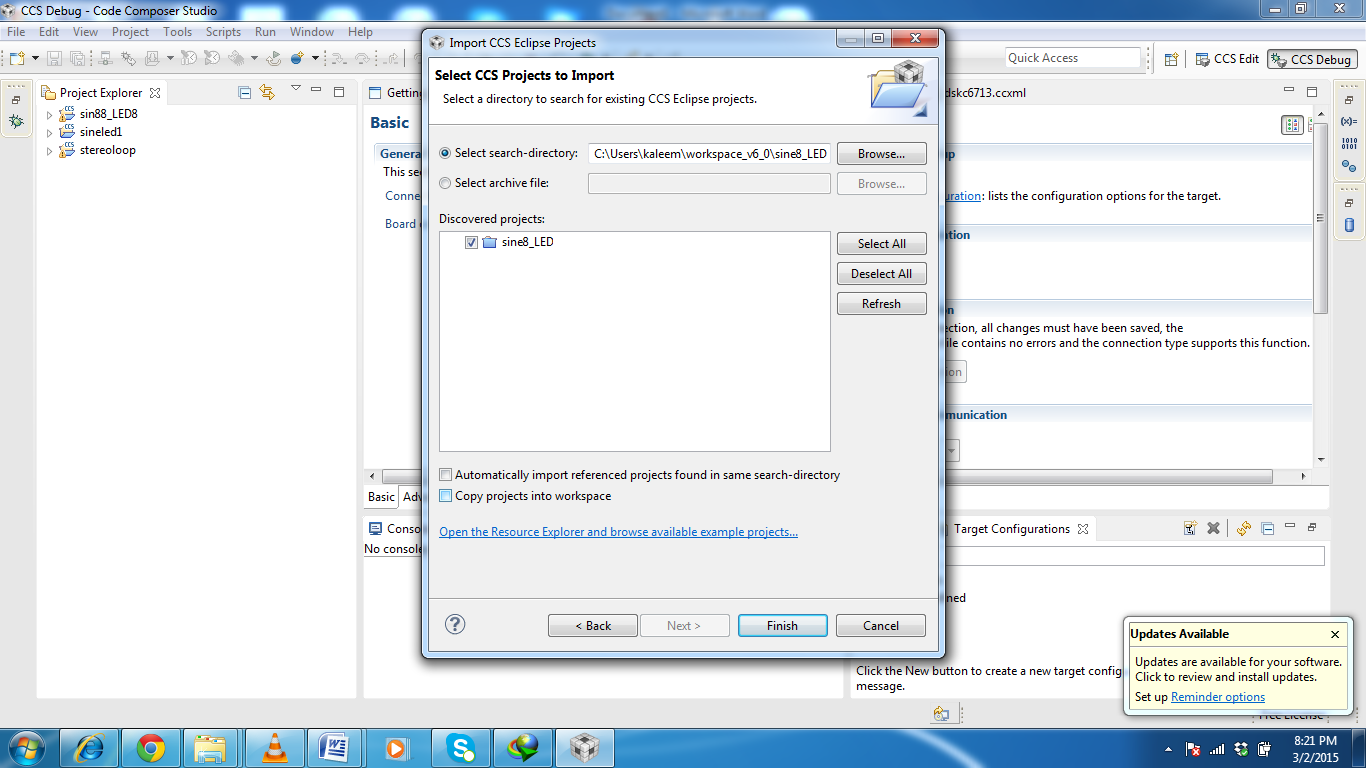
**FIGURE 3.12.** Listing of program sine8\_buf.c .project file, add files to it, or alter compiler and linker build options. In order tobuild, download and run program sine8\_buf.c .

### How to import existing project CCS Project:

1. Close any open projects in CCS.
2. Go to file 🡪 import and you will see following figure



1. Select CCS projects and click next
2. Click browse and you will see following figure. Select the root folder of the which is sine8\_buff in our case.

****

**5.**Click finish and you should be able to see the and relevant files of the prohect in project view.

**Sine8\_buff project description**

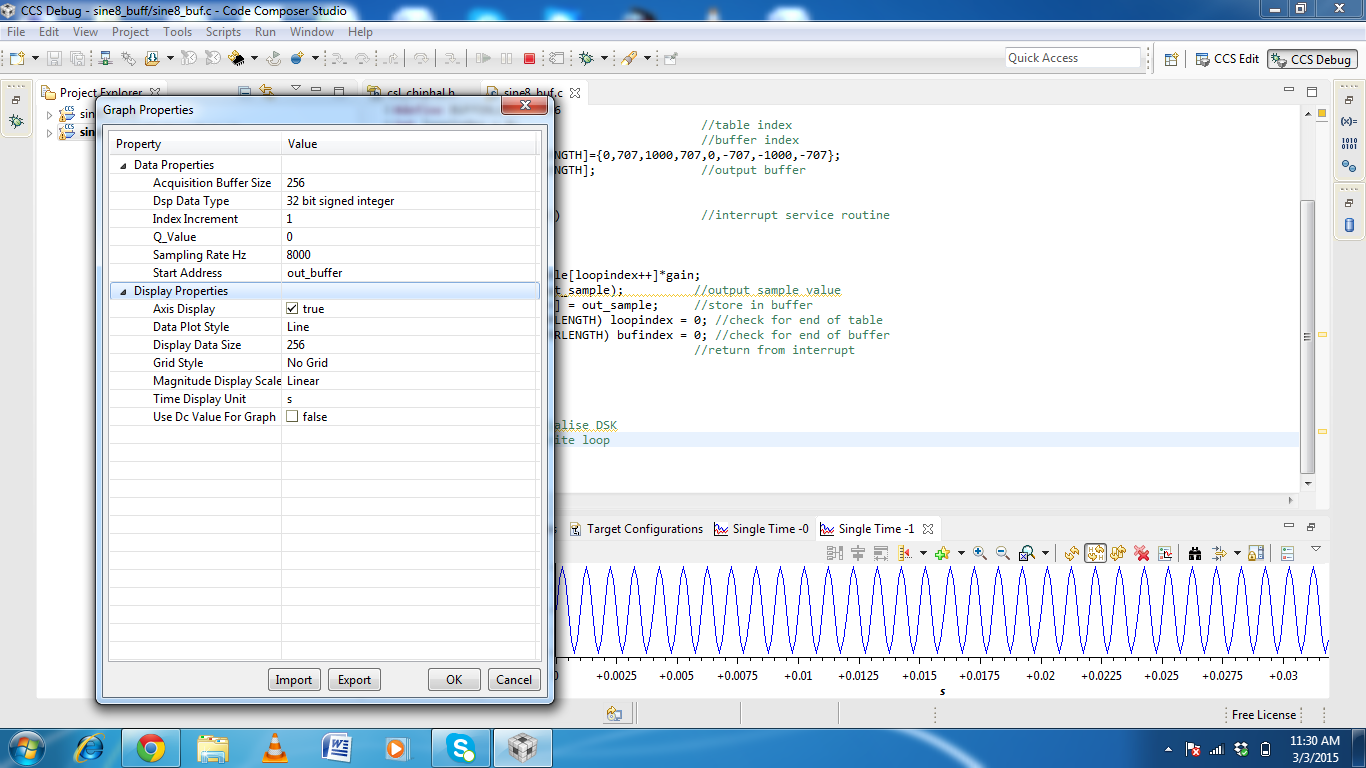
Because this program uses interrupt - driven input/output rather than polling, the file vectors\_intr.asm is used in place of vectors\_poll.asm . The interrupt service table specified in vectors\_intr.asm associates the interrupt service routine c\_int11() with hardware interrupt INT11, which is asserted by the AIC23 codec on the DSK at each sampling instant. Within function main() , function comm\_intr() is used in place of comm\_poll().Essentially, it initializes the DSK hardware, including the AIC23 codec, such that the codec sampling rate is set according to the value of the variable fs and the codec interrupts the processor at every sampling instant. The statement while(1) in function main() creates an infinite loop, during which the processor waits for interrupts. On interrupt, execution proceeds to the interrupt service routine (ISR) c\_int11() , which reads a new sample value from the array sine\_table and writes it both to the array out\_buffer and to the DAC using function output\_left\_sample() . Interrupts are discussed in more detail in Chapter 3 .Build this project as **sine8\_buf** . Load and run the executable file*sine8\_buf. out*and verify that a 1 - kHz sinusoid is generated at the LINE OUT and HEADPHONE sockets

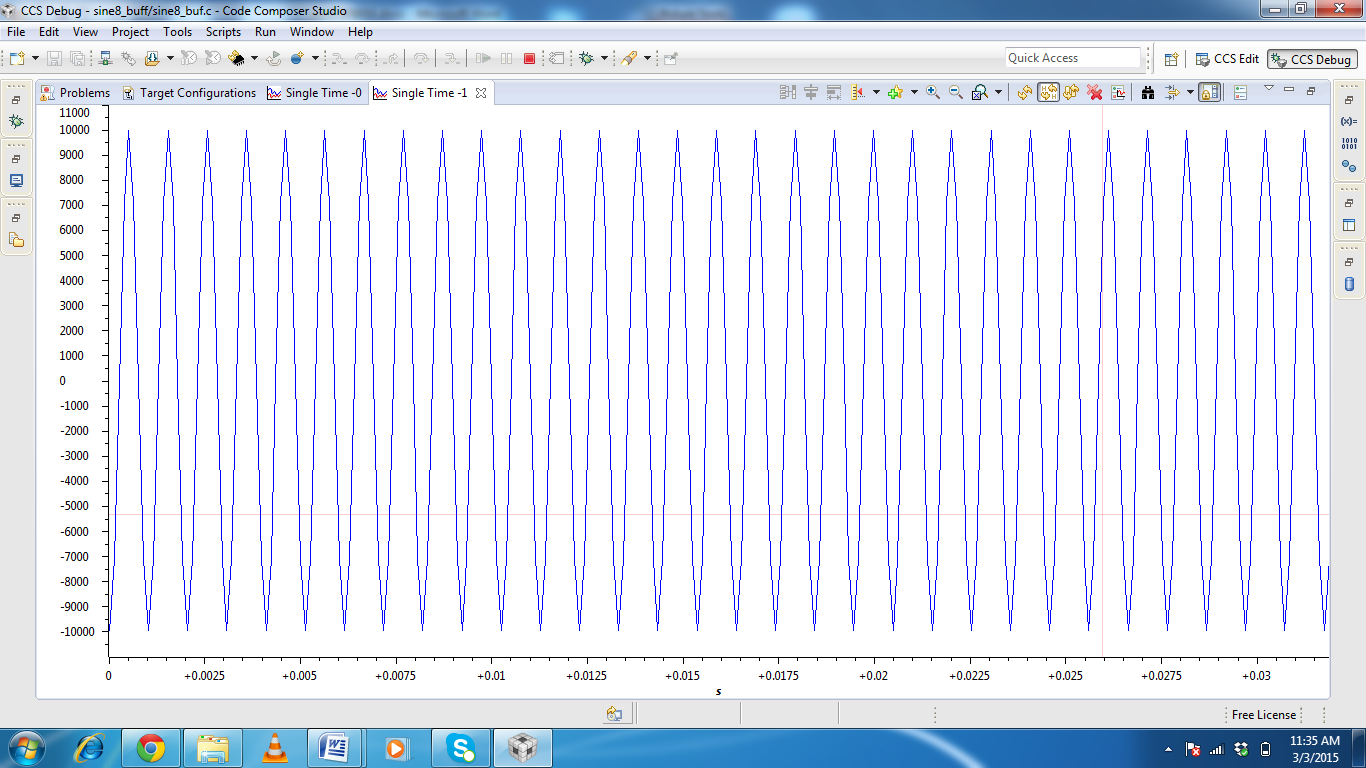
### Graphical Displays in CCS

The array out\_buffer is used to store the BUFFERLENGTH most recently output sample values. Once program execution has been halted, the data stored in out\_ buffer can be displayed graphically in CCS.

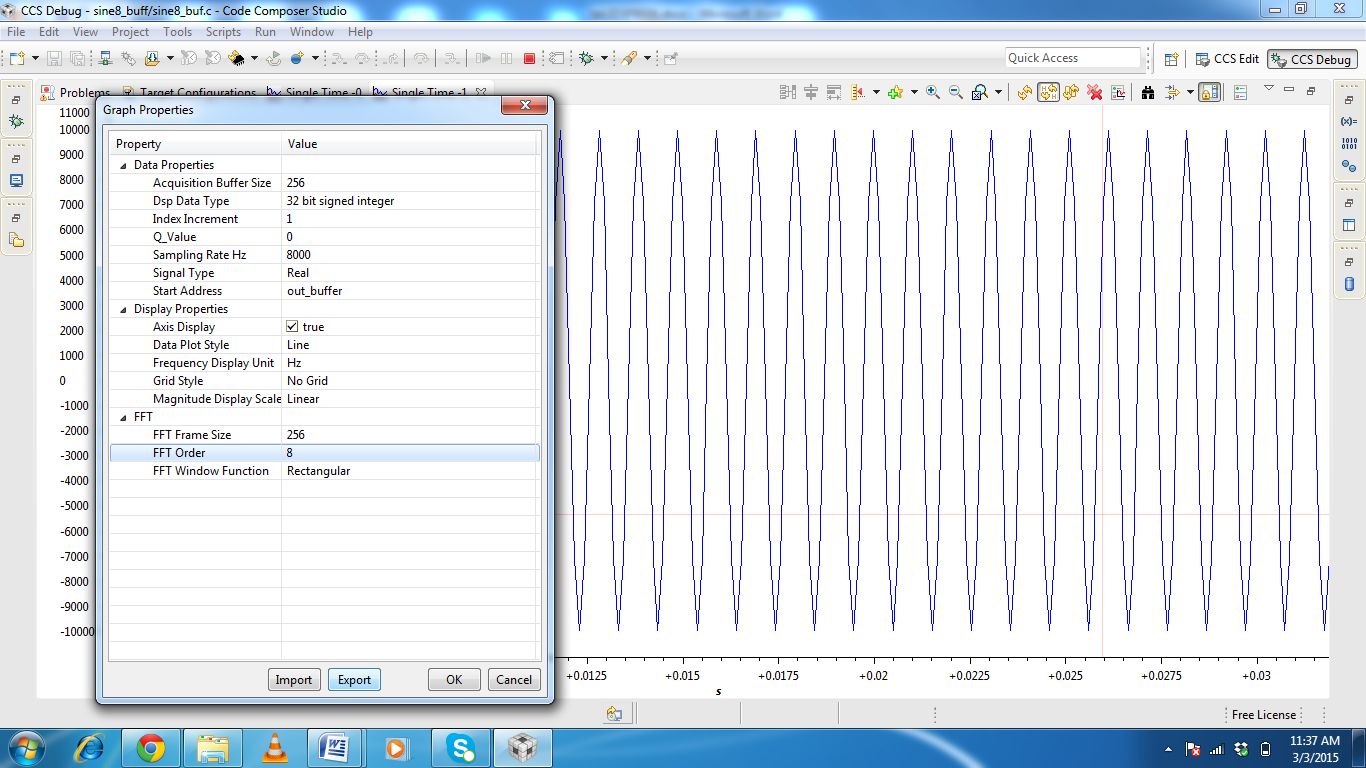
**1.** Select *Tool🡪Graph 🡪Time signal* and set the *Graph Property Dialog* properties as shown in Figure 3.13 a. Figure 3.13 b shows the resultant *Graphical Display* window.

**2.** Figure 3.14 a shows the *Graph Property Dialog* window that corresponds to the frequency domain representation of the contents of out\_buffershown in Figure 3.14 b. The spike at 1 kHz represents the frequency of the sinusoid generated by program sine8\_buf.c.





**FIGURE 3.13.** (a) *Graph Property* window and (b) Time domain plot of data stored inout\_buffer.

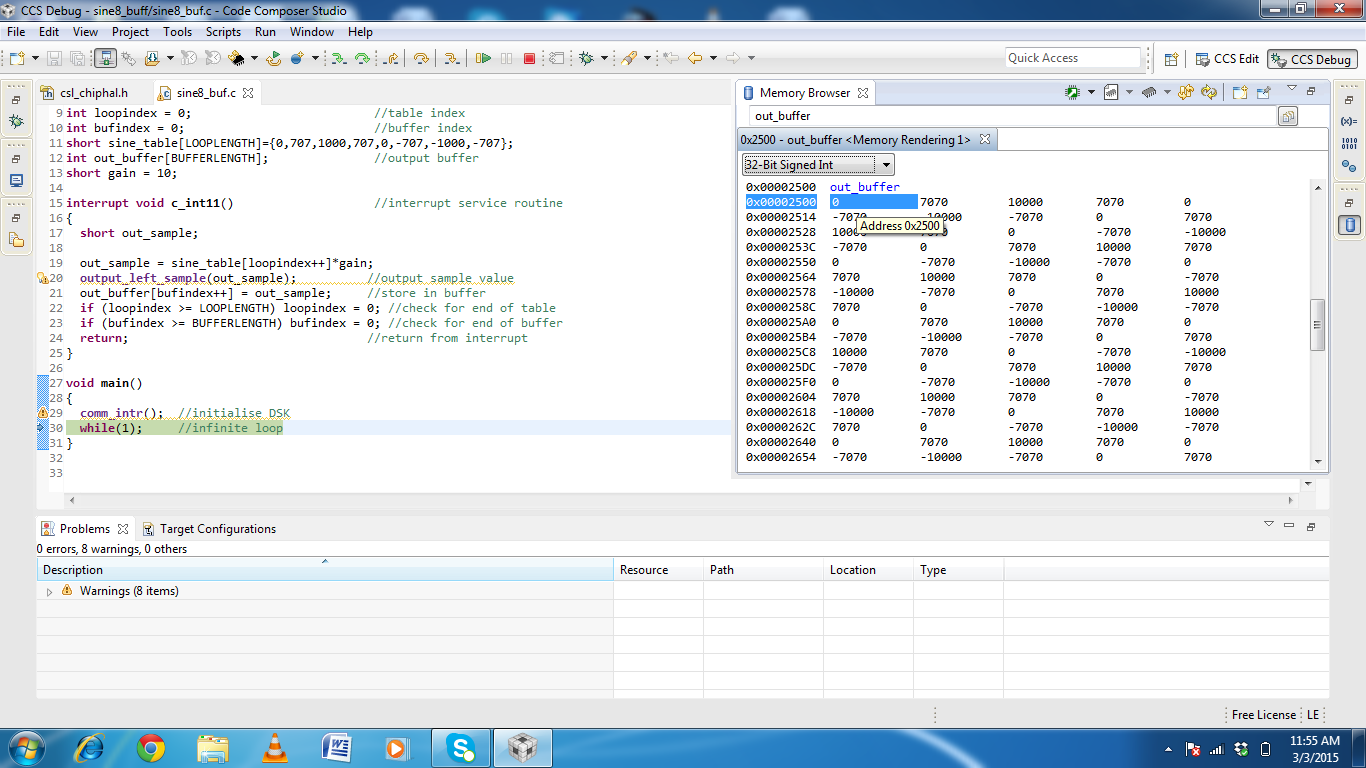




**FIGURE 3.14.** (a) *Graph Property* window and (b) Frequency domain plot of data storedin out\_buffer.

### Viewing and Saving Data from Memory into File

To view the contents of out\_buffer , select *View 🡪MemoryBrowser*. Specify out\_buffer as the *Address* and select *32 - bit Signed Integer* as the *Format* , as shown in Figure 3.15 a.

****

**FIGURE 3.15.** (a) *Memory* window view of data stored inout\_buffer.

//sine8\_buf.c sine generation with output stored in buffer

**#include** "DSK6713\_AIC23.h" //codec support

Uint32 fs=DSK6713\_AIC23\_FREQ\_24KHZ; //set sampling rate

**#define** DSK6713\_AIC23\_INPUT\_MIC 0x0015

**#define** DSK6713\_AIC23\_INPUT\_LINE 0x0011

Uint16 inputsource=DSK6713\_AIC23\_INPUT\_MIC; // select input

**#define** LOOPLENGTH 8

**#define** BUFFERLENGTH 256

**int** loopindex = 0; //table index

**int** bufindex = 0; //buffer index

**short** sine\_table[LOOPLENGTH]={0,707,1000,707,0,-707,-1000,-707};

**int** out\_buffer[BUFFERLENGTH]; //output buffer

**short** gain = 10;

**interrupt** **void** **c\_int11**() //interrupt service routine

{

**short** out\_sample;

out\_sample = sine\_table[loopindex++]\*gain;

output\_left\_sample(out\_sample); //output sample value

out\_buffer[bufindex++] = out\_sample; //store in buffer

**if** (loopindex >= LOOPLENGTH) loopindex = 0; //check for end of table

**if** (bufindex >= BUFFERLENGTH) bufindex = 0; //check for end of buffer

**return**; //return from interrupt

}

**void** **main**()

{

comm\_intr(); //initialise DSK

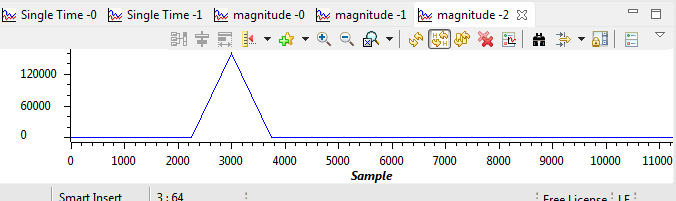
**while**(1); //infinite loop

}

## LAB TASK 1

Modify program sine8\_buf.c to generate a sine wave with a frequency of 3000 Hz. Verify your result using an oscilloscope connected to the LINE OUT socket on the DSK as well as using Code Composer to plot the 32 most recently output samples in both the time and frequency domains.

Uint32 fs=DSK6713\_AIC23\_FREQ\_24KHZ; //set sampling rate



## LAB TASK 2

Write polling - based program such that when DIP switch #3 is pressed down, LED #3 turns on and a 500 - Hz cosine wave is generated for 5 seconds.

//sine8\_LED.c sine generation with DIP switch control

**#include** "dsk6713\_aic23.h" //codec support

Uint32 fs = DSK6713\_AIC23\_FREQ\_8KHZ; //set sampling rate

**#define** DSK6713\_AIC23\_INPUT\_MIC 0x0015

**#define** DSK6713\_AIC23\_INPUT\_LINE 0x0011

Uint16 inputsource=DSK6713\_AIC23\_INPUT\_MIC;//select input

**#define** LOOPLENGTH 16

**short** loopindex = 0; //table index

**short** gain = 100; //gain factor

**short** sine\_table[LOOPLENGTH]={1, 0.92, 0.707, 0.38, 0, -0.38, -0.707, -0.92, -1, -0.92, -0.707, -0.38, 0, 0.38, 0.707, 0.92}; //sine values

**void** **main**()

{

comm\_poll(); //init DSK,codec,McBSP

DSK6713\_LED\_init(); //init LED from BSL

DSK6713\_DIP\_init(); //init DIP from BSL

/\*while(1) //infinite loop

{

if(DSK6713\_DIP\_get(0)==0) //=0 if DIP switch #0 pressed

{

DSK6713\_LED\_on(); //turn LED #0 ON

output\_left\_sample(sine\_table[loopindex++]\*gain); //output sample

if (loopindex >= LOOPLENGTH) loopindex = 0; //reset table index

}

else DSK6713\_LED\_off(0); //turn LED off if not pressed

} \*/ //end of while(1) infinite loop

**int** i;

**while** (1)

{

**if**(DSK6713\_DIP\_get(0)==0)

{

**for** (i = 0; i < 40000; i++)

{

DSK6713\_LED\_on(); //turn LED #0 ON

output\_left\_sample(sine\_table[loopindex++]\*gain); //output sample

**if** (loopindex >= LOOPLENGTH) loopindex = 0; //reset table index

}

}

**else**

DSK6713\_LED\_off(0); //turn LED off if not pressed

}

} //end of main

## LAB TASK 3

Write an interrupt - driven program that maintains a buffer containing the 128 most recent input samples read at a sampling frequency of 16 kHz from the AIC23 codec, using the MIC IN socket on the DSK. Halt the program andplot the buffer contents using Code Composer.