**Lab3: Introduction to DSP Kit TMS 320C6713 DSK**

**Objectives**

The objective of this lab is to introduce the DSP Starter Kit C6713

* Getting started with DSP Kit
* Getting Started with Code Composer Studio
* Basic Code Compilation on DSP Kit
* Working with basic sinusoids on DSP Kit

**Lab Instructions**

* This lab activity comprises of three parts: Pre-lab, Lab Exercises, and Post-Lab Viva session.
* The lab report shall be uploaded on LMS three days before next scheduled lab. The Pre-lab tasks should be completed before coming to the lab and hard copy of Pre-lab session should be deposited with teacher/lab engineer at start of the lab for necessary evaluation. Alternatively, the reports can be submitted in PDF format on LMS.
* 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)
* Only those tasks that completed during the allocated lab time will be credited to the students. 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
* MATLAB codes
* Results (graphs/tables) duly commented and discussed
* Conclusion

# Pre Lab

### Introduction

The hardware experiments in the DSP lab are carried out on the Texas Instruments TMS320C6713 DSP Starter Kit (DSK), based on the TMS320C6713 floating point DSP running at 225 MHz. The basic clock cycle instruction time is 1/(225 MHz)= 4.44 nanoseconds. During each clock cycle, up to eight instructions can be carried out in parallel, achieving up to 8×225 = 1800 million instructions per second (MIPS). The C6713 processor has 256KB of internal memory, and can potentially address 4GB of external memory. The DSK board includes a 16MB SDRAM memory and a 512KB Flash ROM. It has an on-board 16-bit audio stereo codec (the Texas Instruments AIC23B) that serves both as an A/D and a D/A converter. There are four 3.5 mm audio jacks for microphone and stereo line input, and speaker and head-phone outputs. The AIC23 codec can be programmed to sample audio inputs at the following sampling rates:

fs = 8, 16, 24, 32, 44.1, 48, 96 kHz

The DSK also has four user-programmable DIP switches and four LEDs that can be used to control and monitor programs running on the DSP. All features of the DSK are managed by the CCS, which is a complete integrated development environment (IDE) that includes an optimizing C/C++ compiler, assembler, linker, debugger, and program loader. The CCS communicates with the DSK via a USB connection to a PC. In addition to facilitating all programming aspects of the C6713 DSP, the CCS can also read signals stored on the DSP’s memory, or the SDRAM, and plot them in the time or frequency domains.

### TMS 320 C 6713 Digital Signal Processor

The TMS320C6713 (C6713) is based on the very long instruction word (VLIW) architecture, which is very well suited for numerically intensive algorithms. The internal program memory is structured so that a total of eight instructions can be fetched every cycle. For example, with a clock rate of 225 MHz, the C6713 is capable of fetching eight 32 - bit instructions every 1/(225 MHz) or 4.44 ns. Features of the C6713 include 264 kB of internal memory (8 kB as L1P and L1D Cache and 256 kB as L2 memory shared between program and data space), eight functional or execution units composed of six ALUs and two multiplier units, a 32 - bit address bus to address 4 GB (gigabytes), and two sets of 32 - bit general – purpose registers. The C67xx processors (such as the C6701, C6711, and C6713) belong to the family of the C6x floating - point processors; whereas the C62xx and C64xx belong to the family of the C6x fixed - point processors. The C6713 is capable of both fixed and floating point processing. 



### CODE COMPOSER STUDIO

Code Composer Studio (CCS) provides an integrated development environment (IDE) for real - time digital signal processing applications based on the C programming language. It incorporates a C compiler, an assembler, and a linker. It has graphical capabilities and supports real - time debugging. The C compiler compiles a C source program with extension .c to produce an assembly source file with extension *.asm* . The assembler assembles an *.asm* source file to produce a machine language object file with extension *.obj* . The linker combines object files and object libraries as input to produce an executable file with extension *.out* . This executable file represents a linked common object file format (COFF), popular in Unix - based systems and adopted by several makers of digital signal processors [44] . This executable file can be loaded and run directly on the digital signal processor.

A Code Composer Studio project comprises all of the files (or links to all of the files) required in order to generate an executable file. A variety of options enabling files of different types to be added to or removed from a project are provided. In addition, a Code Composer Studio project contains information about exactly how files are to be used in order to generate an executable file. Compiler/linker options can be specified. A number of debugging features are available, including setting breakpoints and watching variables, viewing memory, registers, and mixed C and assembly code, graphing results, and monitoring execution time. One can step through a program in different ways (step into, or over, or out). Real - time analysis can be performed using CCS’ s real - time data exchange (RTDX) facility. This allows for data exchange between the host PC and the target DSK as well as analysis in real - time without halting the target.

### File Types

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

**1.** file.c : C source program.

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

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

**4.** file.h : header support file.

**5.** file.lib : library file, such as the run - time support library file rts6700.lib .

**6.** file.cmd : linker command file that maps sections to memory.

**7.** file.obj : object file created by the assembler.

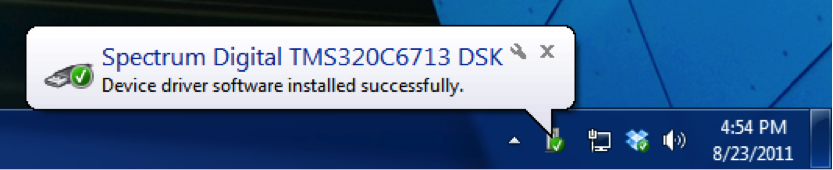
**8.** file.out : executable file created by the linker to be loaded and run on theC6713 or C6416 processor.

**9.** file.cdb : configuration file when using DSP/BIOS.

## Warmup

### QUICK TESTS OF THE DSK ( ON POWER ON AND USING CCS )

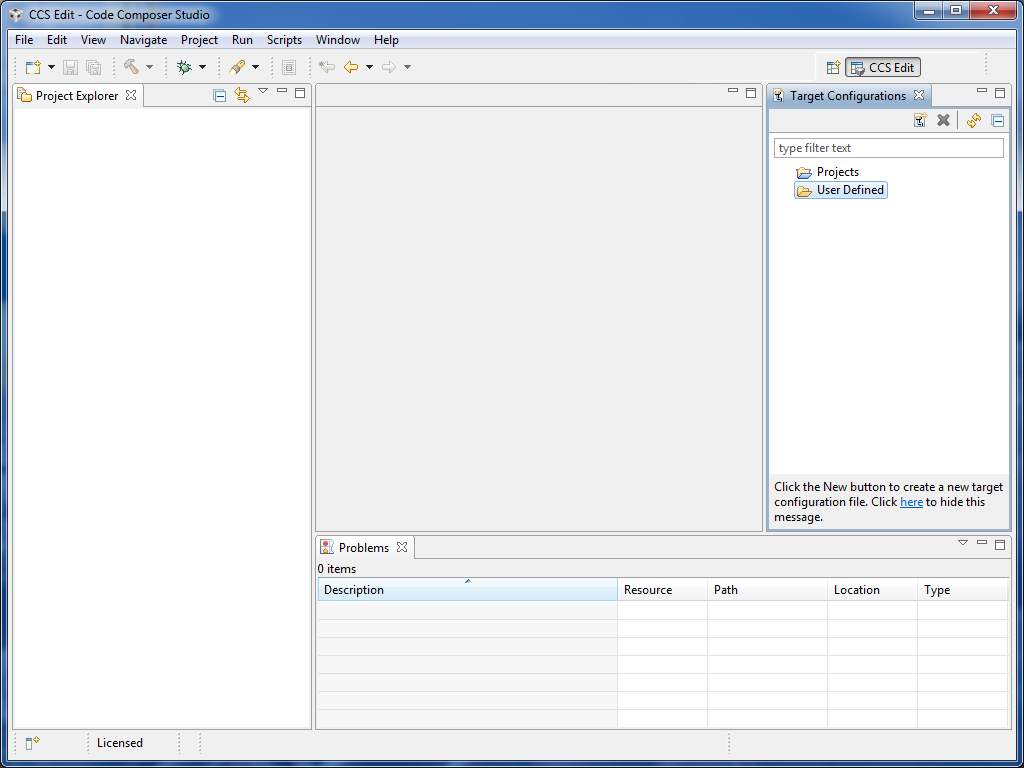
Connect the USB cable from the DSK6713 to the computer and then connect power to the DSK6713. If this is the first time the DSK has been connected to the computer, you may see Windows install the device driver software:

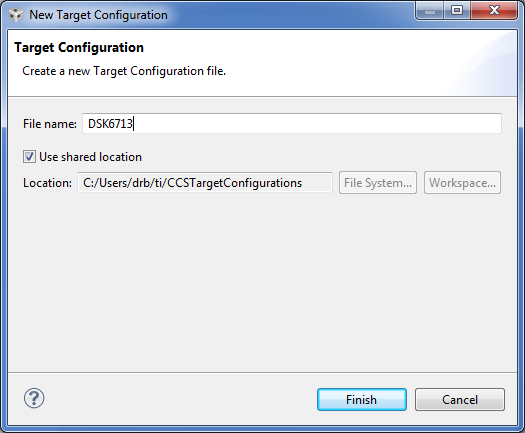
   
You can also check that the DSK is recognized by going into the Device Manager and looking for "SD USB Based Debug Tools". You should see a "Spectrum Digital TMS320C6713 DSK" listed as a correctly installed device. If not, then CCS v6 has probably not been installed correctly.

Now start CCS v6 by double-clicking on the desktop icon (or navigating through the start menu). You may be prompted to select a workspace. If so, choose a directory that you would like to use as your workspace and click "OK". If the TI Resource Explorer window comes up, just close it. Eventually, you should get to the main CCS v6 window, which should look something like this

This is the "normal" edit view of CCS v6 where you will manage projects, build and debug your code, and interact with the DSK. Since this is the first time we are using CCS v6, we must set up a "target configuration" for the DSK6713.

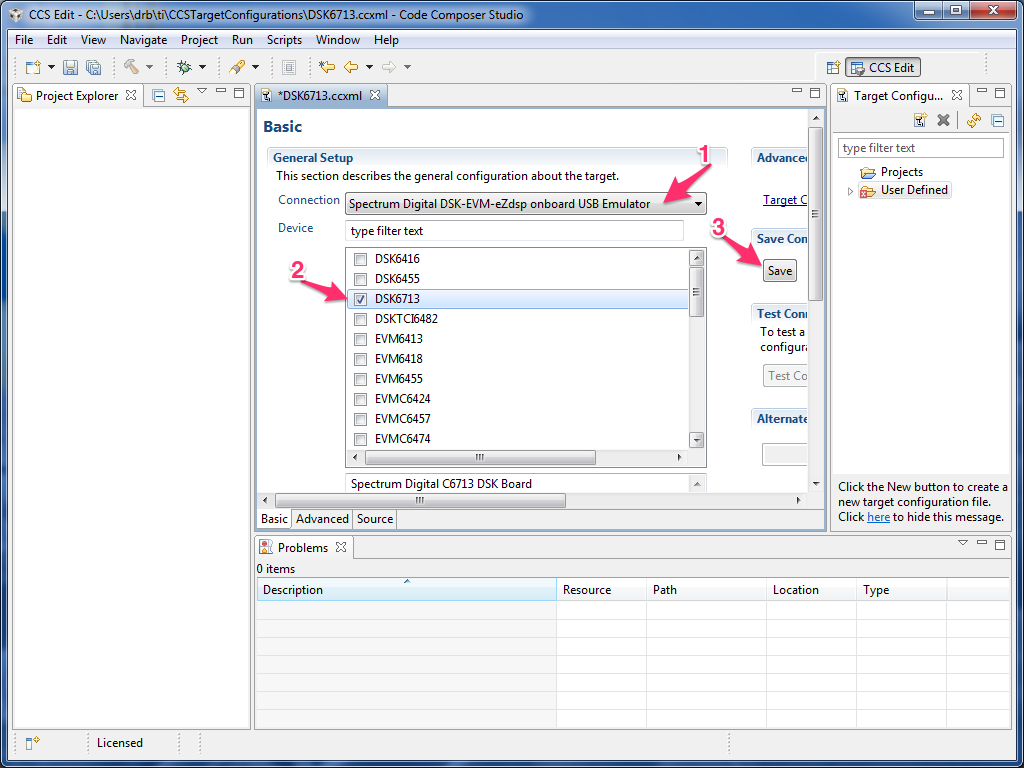
**Part 2: Setting up a DSK6713 Target Configuration**

From the normal view of CCS v6, click on "Window->Show View->Target Configurations" to expose the target configurations panel.   
When you first use CCS v6, there won't be any target configurations. So we will set one up for the DSK6713. Note that you only need to do this once. Right click on the "User Defined" folder in the target configurations panel and select "New Configuration".

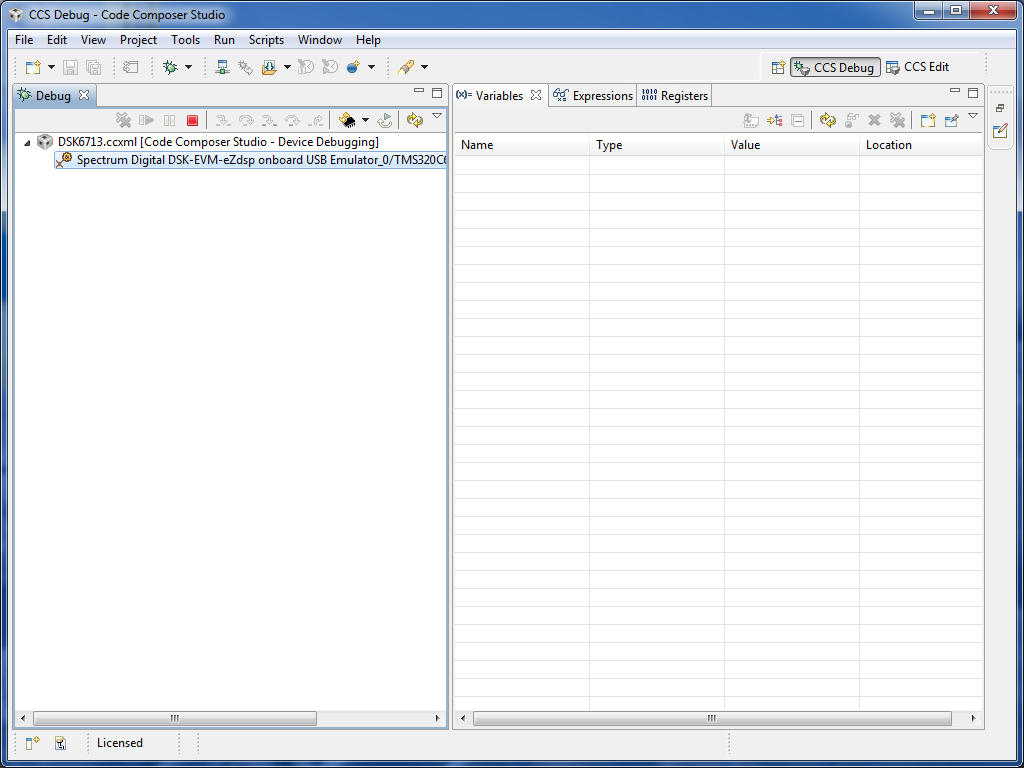
 

You can name the new target configuration anything you want, but it is a good idea to pick something descriptive. I've named the target configuration DSK6713.ccxml (the extension will be added automatically) and stored it in the default shared location. Click "Finish".

A new panel should appear where you set some options for this DSK6713.ccxml target configuration. In the connection pull down menu, select "Spectrum Digital DSK-EVM-eZdsp onboard USB Emulator". In the device selector, select "DSK6713"

   
Click "Save".

Now expand the "User Defined" target configurations folder by clicking on the little triangle next to the folder. You should see the new DSK6713.ccxml target configuration in there. Right click on it and select "Launch Selected Configuration". You should see some USB enumeration as CCS v6 talks with the DSK and then you should see something like this window.

   
If you get an error, then either CCS v6 wasn't installed correctly or you haven't set up the target configuration correctly.

Now click on "Run->Connect Target" (or press Ctrl+Alt+C). You should then see a window something like this The message "GEL StartUp Complete" in the Console means that CCS v6 is successfully talking with the DSK.

### Sine Wave Generation Using Eight Points with DIP Switch Control ( sine8\_LED )

This example generates a sinusoidal analog output waveform using a table – lookup method. More importantly, it illustrates some of the features of CCS for editing source files, building a project, accessing the code generation tools, and running a program on the C6713 processor. The C source file *sine8\_LED.c* listed in Figure 1.2 is included in the folder *sine8\_LED* .

### Program Description

The operation of program *sine8\_LED.c* is as follows. An array, sine\_table , of eight 16 - bit signed integers is declared and initialized to contain eight samples of exactly one cycle of a sinusoid. The value of sine\_table[i] is equal to

*1000sin(2πi/8) for i 🡪1, 2, 3, . . . , 7*

Within function *main()* , calls to functions *comm\_poll()* , *DSK6713\_LED\_init()* , and *DSK6713\_DIP\_init()* initialize the DSK, the AIC23 codec onboard the DSK, and the two multichannel buffered serial ports (McBSPs) on the C6713 processor. Function *comm\_poll()* is defined in the file *c6713dskinit.c* , and functions *DSK6713\_LED\_init()* and *DSK6713\_DIP\_init()* are supplied in the board support library (BSL) file dsk6713bsl.lib .

The program statement while(1) within the function *main()* creates an infinite loop. Within that loop, the state of DIP switch #0 is tested and if it is pressed down, LED #0 is switched on and a sample from the lookup table is output. If DIP switch #0 is not pressed down then LED #0 is switched off. As long as DIP switch #0 is pressed down, sample values read from the array sine\_table will be output and a sinusoidal analog output waveform will be generated via the left - hand channel of the AIC23 codec and the LINE OUT and HEADPHONE sockets. Each time a sample value is read from the array sine\_table , multiplied by the value of the variable gain , and written to the codec, the index, loopindex , into the array is incremented and when its value exceeds the allowable range for the array( LOOPLENGTH - 1 ), it is reset to zero.

*//****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 8*

*short loopindex = 0; //table index*

*short gain = 10; //gain factor*

*short sine\_table[LOOPLENGTH]=*

*{0,707,1000,707,0,-707,-1000,-707}; //sine values*

*void main()*

*{*

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

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

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

*while(1) //infinite loop*

*{*

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

*{*

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

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

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

*}*

*else DSK6713\_LED\_off(0); //else turn LED #0 OFF*

*} //end of while(1)*

*} //end of main*

**FIGURE 1.2.** Sine wave generation program using eight points with DIP switch control ( sine8\_LED.c ).

Each time the function output\_left\_sample() , defined in source file *C6713dskinit.c* , is called to output a sample value, it waits until the codec, initialized by the function comm\_poll() to output samples at a rate of 8 kHz, is ready for the next sample. In this way, once DIP switch #0 has been pressed down it will be tested at a rate of 8 kHz. The sampling rate at which the codec operates is set by the program statement

*Uint32 fs = DSK6713\_AIC23\_FREQ\_8KHZ;*

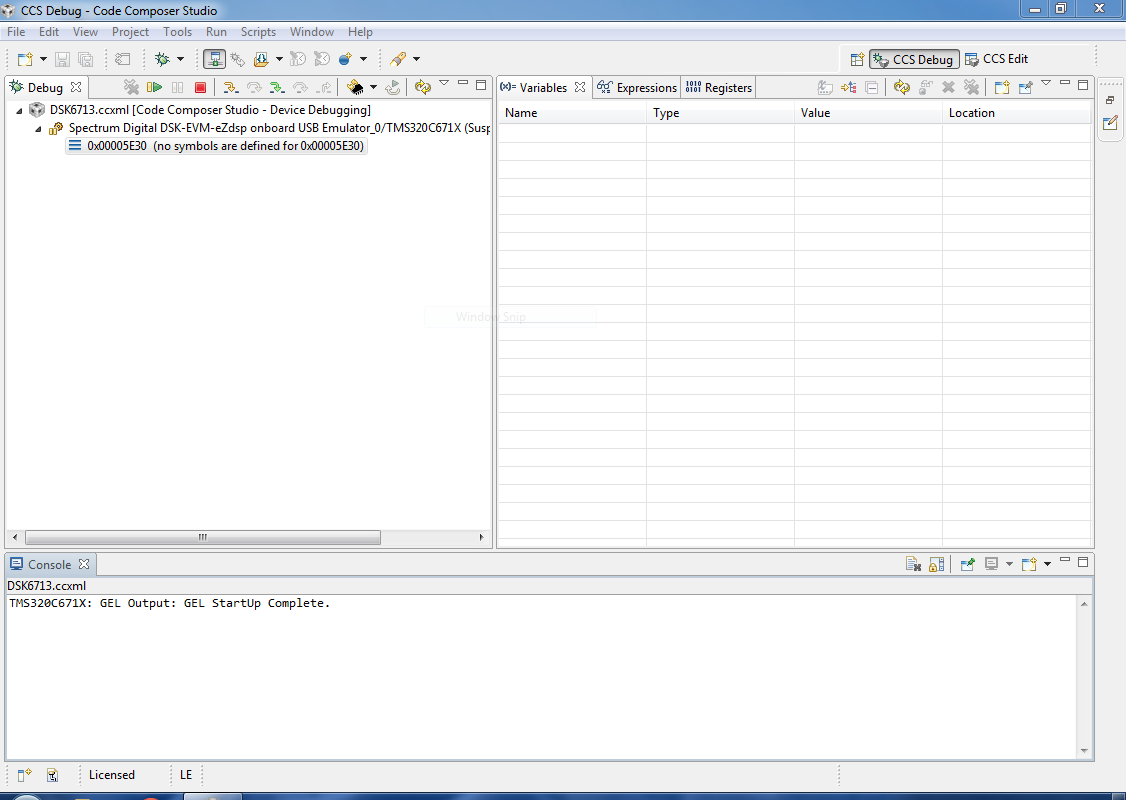
One cycle of the sinusoidal analog output waveform corresponds to eight output samples and hence the frequency of the sinusoidal analog output waveform is equal to the codec sampling rate (8 kHz) divided by eight, that is, 1 kHz.

## Lab Task 1

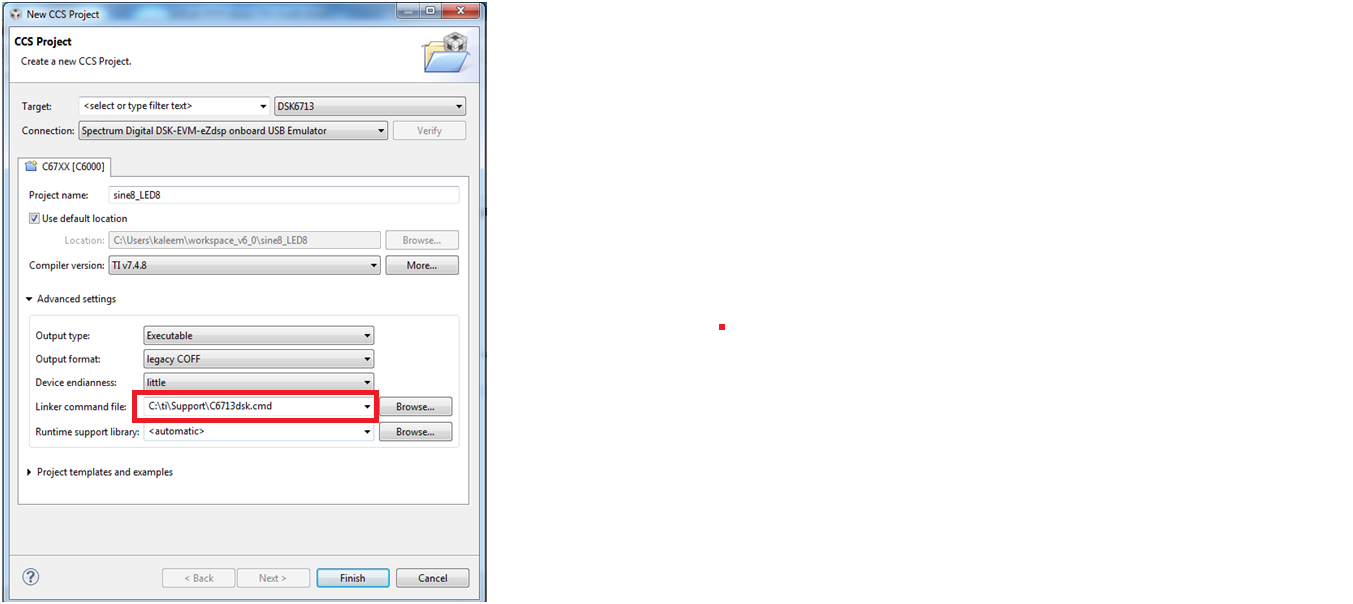
### Creating a Project

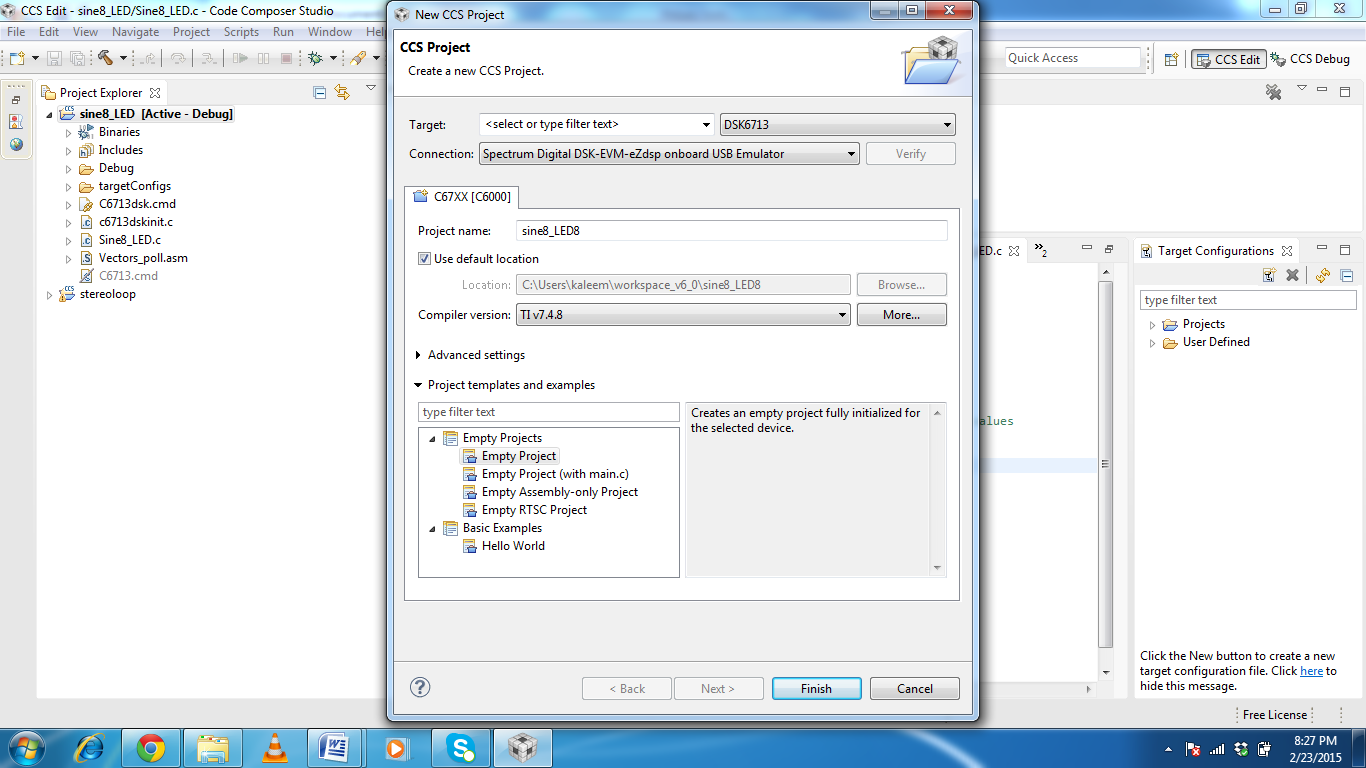
This project will require following files: the main source code ([sine8\_LED.c](http://spinlab.wpi.edu/courses/ece4703_2013/stereoloop/stereoloop.c)), a linker command file (C6713.cmd), c6713dskinit.c, c6713dskinit.h and an ASM file containing the necessary interrupt vectors (vectors\_poll.asm). You can download these files from lms now and add them to the project in a later

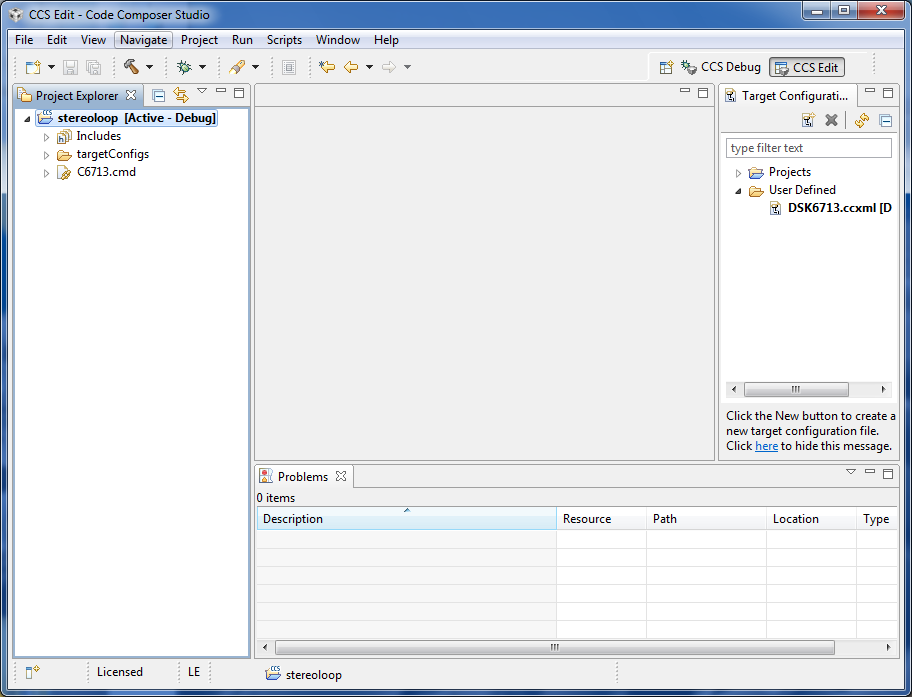
**Part 2: Create the CCS v6 project container**

These steps assume that you have correctly installed and configured CCS v6. You should be able to connect to the DSK and see something like the following window before proceeding.

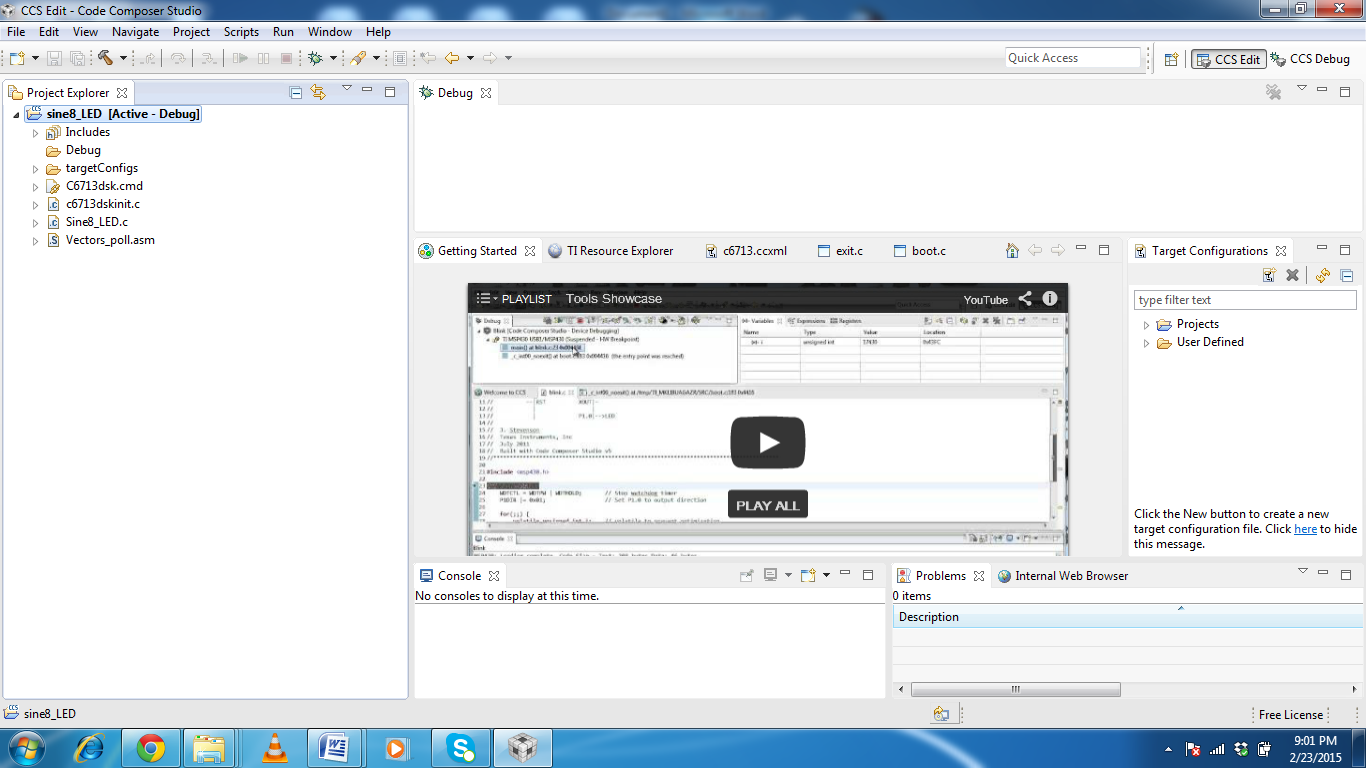
Click on "File->New->CCS Project". If this is the first time using CCS v6, you may have to click "File->New->Other" and then select CCS project. The following window should appear and set values according to figure.

   
This is where you name and configure your project. Here we named the project "sine8\_LED", told CCS to use the default location (you may want a different location), set the device variant to DSK6713 and told CCS how to connect to it (this is very important). We've also opened the advanced settings and told CCS to use our C6713.cmd linker command file (if you miss this step, you can always replace the linker command file later).”C6713.cmd” should be available on the path C:\ti\Support\C6713dsk.cmd .Under project templates and examples, select "Empty Project" and then click "Finish".

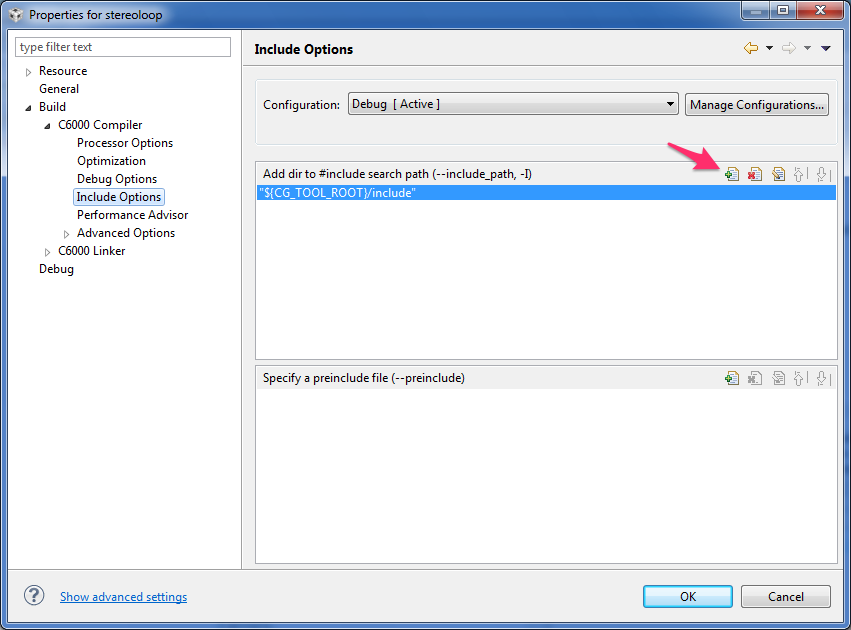


 This is what it should look like after you've set up the project container.   
You might have some other panels visible, but the Project Explorer panel should show the sin8\_LED project as "Active" and should show the C6713.cmd file.

**Part 3: Add source code to your project**

So far, we just have an empty project container with no code. Now you can copy or move sin8\_LED.c a, vectors\_poll.asm and c6713dskinit.c into your project directory using Project->add files option.these files are available on SSC/LMS.Here is what CCS v6 should look like after it recognizes that you moved the two source files into the project directory.  

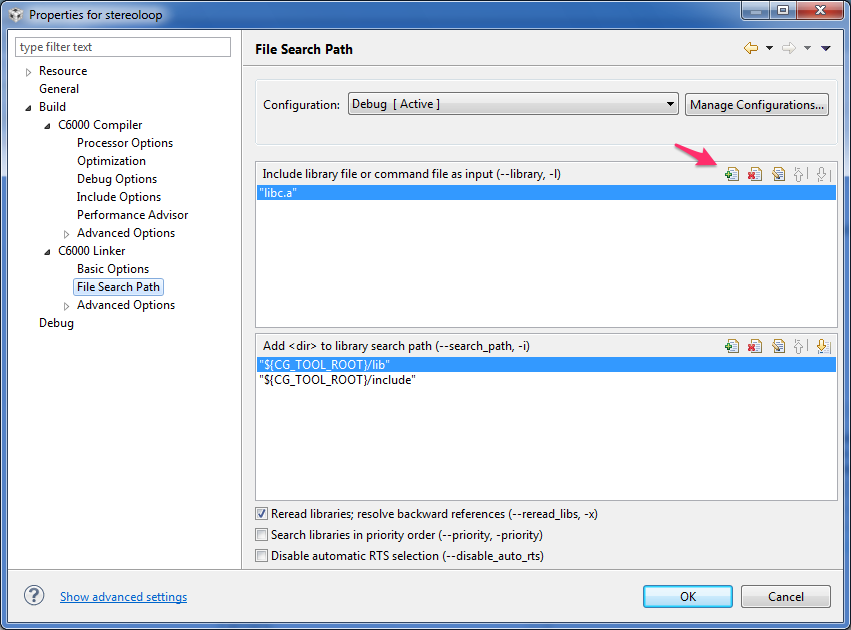
**Part 4: Tell CCS v6 where to find the necessary header files and libraries**

If we try to build the project now, we will get some errors because (among other things) CCS v6 doesn't know how to find the CSL and BSL header files and libraries. We'll first tell CCS v6 where to find the header files. Click on "Project->Properties". Drill down to "Build / C6000 Compiler / Include Options" in the left sidebar:   
This is where we will configure CCS v6 to find the appropriate header files. Note that we can add directories to the #include search path here by pressing the button with the green plus sign add following paths.

C:\ti\Support

C:\ti\DSK6713\DSK6713\c6000\dsk6713\include

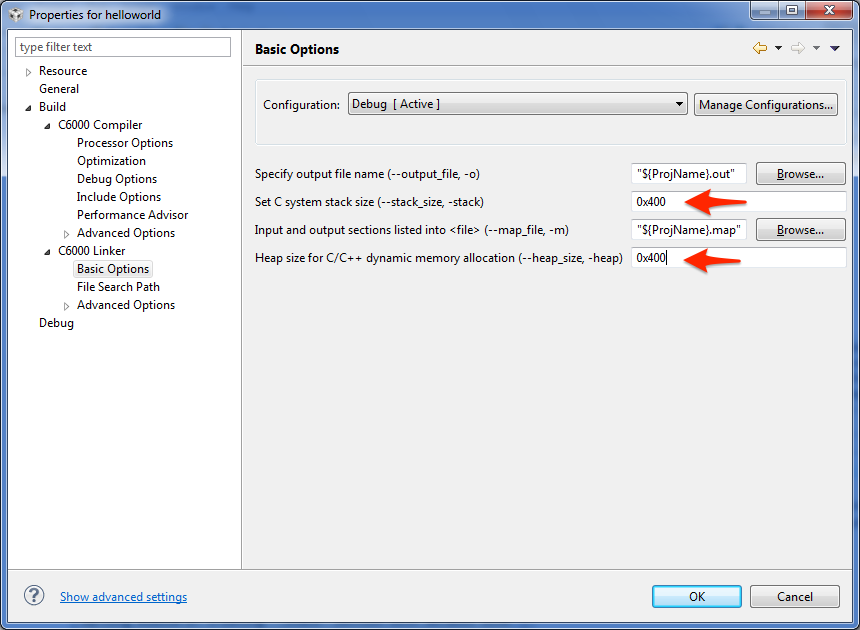
C:\ti\C6xCSL\include.

We now need to tell CCS v6 to include the CSL and BSL library files in the project. Drill down to "Build / C6000 Linker / File Search Path" in the left sidebar:   
Note that we can include library files directly in the project by pressing the button with the green plus sign.

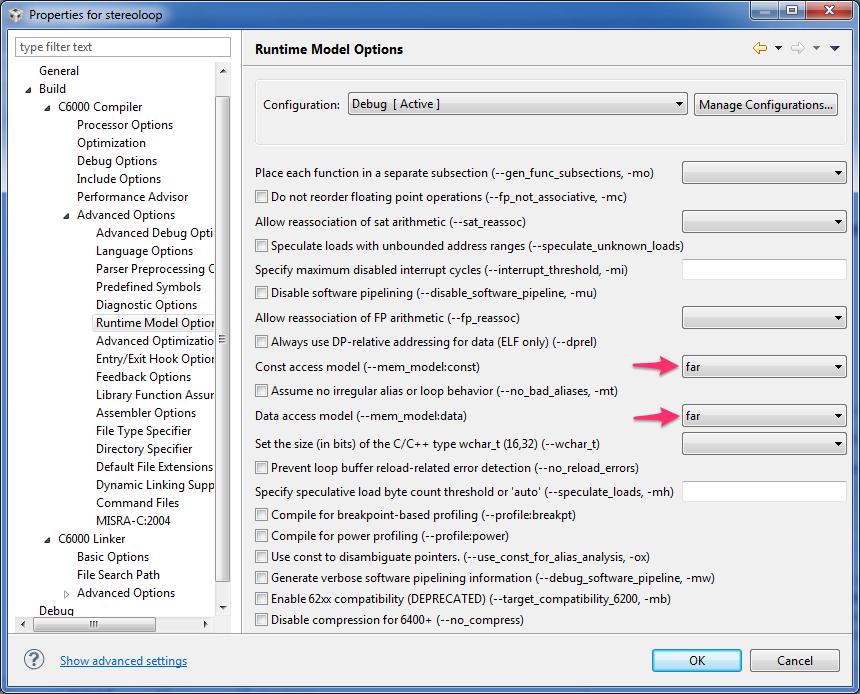
Press the button with the green plus sign to add the CSL library   
Usually, the CSL library is in C:\ti\C6xCSL\lib\_3x\csl6713.lib. Click "OK".

Press the button with the green plus sign to add the BSL library   
Usually, the BSL library is in C:\ti\DSK6713\c6000\dsk6713\lib\dsk6713bsl.lib. Click "OK".

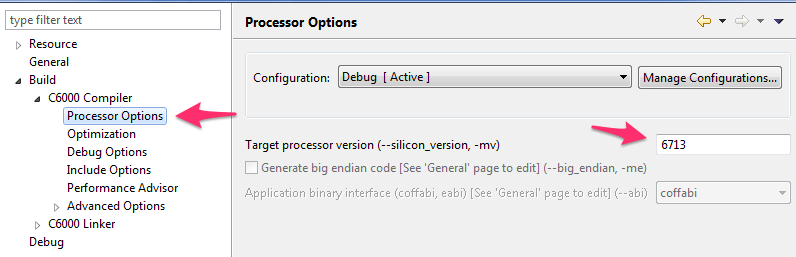
**Part 5: Tell CCS what to use for stack and heap sizes**

You could now compile the project, but you would get some warnings about not explicitly setting the stack and heap sizes (CCS v6 will just go with some default values). We would like to get rid of these warnings. To do this, drill down to "Build / C6000 Linker / Basic Options" in the left sidebar and put in values of 0x400 for stack size and 0x400 for heap size 

**Part 6: Tell CCS to use a "far" memory model**

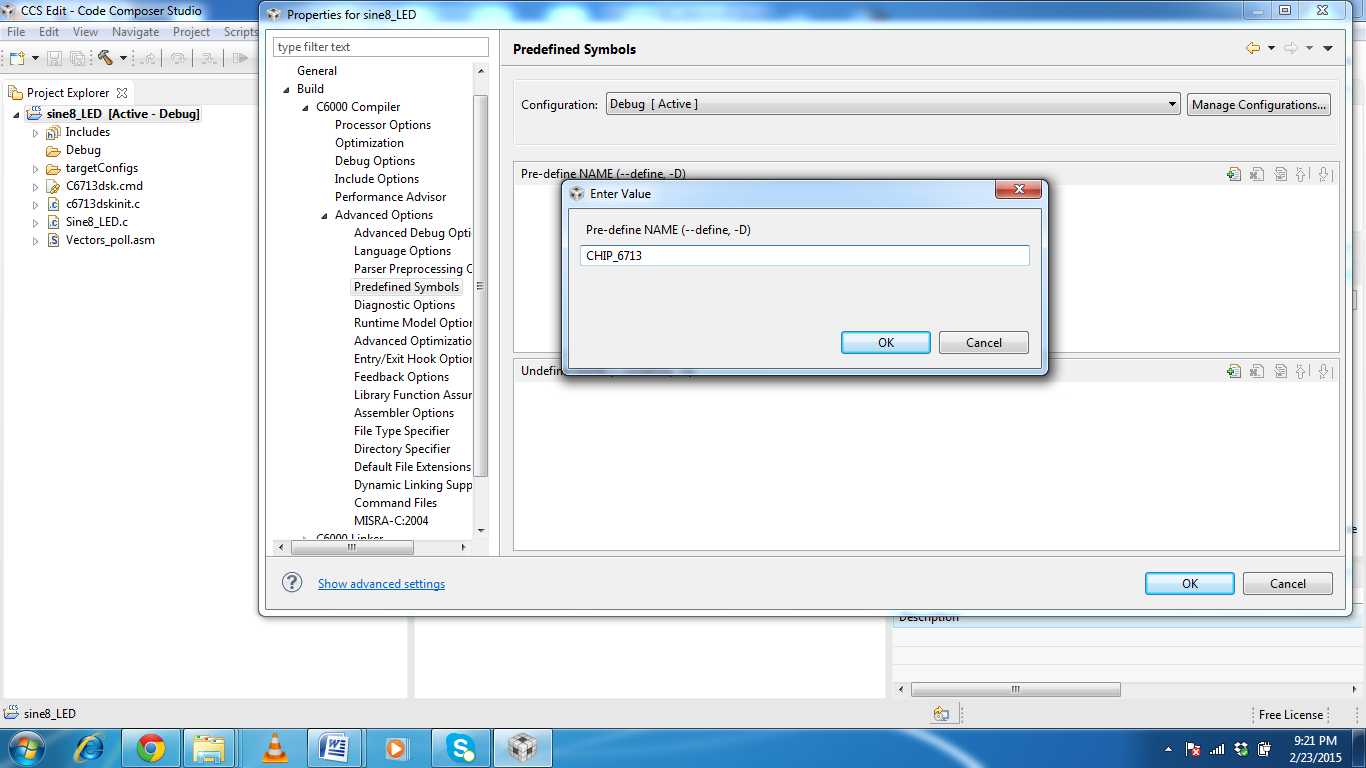
There is still one more thing to fix before CCS v6 will cleanly compile the project. Drill down to "Build / C6000 Compiler / Advanced Options / Runtime Model Options" in the left sidebar and set the "Constant access model" and "Data access model" both to "Far":

**Part 7: Tell CCS we are using a 6713 chip**

While the project will compile with the current settings, the compiler will not use instructions optimized for the TMS320C6713 chip unless we tell it to do so. Drill down to "Build / C6000 Compiler / Processor Options" in the left sidebar and set the "Target processor version" to "6713" (no quotes):   
Now click "OK".

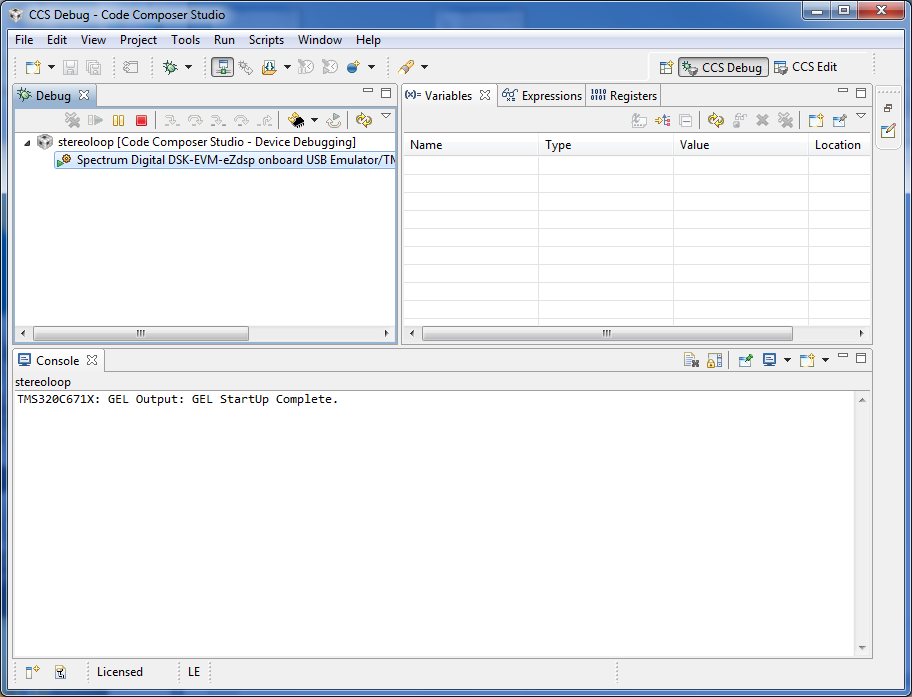
**Part 8 define Chip symbol that is CHIP\_6713 in our case**

Drill down to "Build / C6000 Compiler / Advance options/predefined symbols" and press the button with the green plus sign add type in CHIP\_6713 and press OK as show in figure below

****

**Part 9: Build and run the project**

If you've followed all of the steps above, you should be able to build the project without errors or warnings. Click on the hammer button or "Project->Build All". You should see something like this:

Now click the green bug icon (this launches the debugger). This will cause the executable output of the build to be loaded onto the DSK and put the instruction pointer of the DSP at the start of your code. You may be asked to terminate an existing debug session. If so, click "Yes". You may also see some USB enumeration as CCS connects with the DSK. After a few seconds, you should see something like this upon entering the debugger:   
In the Debug window, press the Run/Resume button. Try it. You can halt the code anytime by pressing the Suspend button (looks like pause). When the DSP is running, you should hear music. When the DSP is halted, you should not hear music.

## LAB Task 2

### Changing the Frequency of the Generated Sinusoid

There are several different ways in which the frequency of the sinusoid generated by program sine8\_LED.c can be altered.

**1.** Change the AIC23 codec sampling frequency from 8 kHz to 16 kHz by changing the line that reads

Uint32 fs = DSK6713\_AIC23\_FREQ\_8KHZ; to read Uint32 fs = DSK6713\_AIC23\_FREQ\_16KHZ;

Rebuild (use incremental build) the project, load and run the new executable file, and verify that the frequency of the generated sinusoid is 2 kHz. The frequencies supported by the AIC23 codec are 8, 16, 24, 32, 44.1, 48,and 96 kHz.

**2.** Change the number of samples stored in the lookup table to four. By changing the lines that read

#define LOOPLENGTH 8

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

to read

#define LOOPLENGTH 4

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

Verify that the frequency of the sinusoid generated is 2 kHz (assuming an 8-kHz sampling frequency).Remember that the sinusoid is no longer generated if the DIP switch #0 is not pressed down. A different DIP switch can be used to control whether or not a sinusoid is generated by changing the value of the parameter passed to the functions *DSK6713\_DIP\_get(), DSK6713\_LED\_on(), and DSK6713\_ LED\_off()* . Suitable values are 0, 1, 2, and 3. Two sliders can readily be used, one to change the gain and the other to change the frequency. A different signal frequency can be generated, by changing the incremental changes applied to the value of loopindex within the C program (e.g., stepping through every two points in the table). When you exit CCS after you build a project, all changes made to the project can be saved.

**REFERENCES**

1. R. Chassaing , *DSP Applications Using C and the TMS320C6x DSK* , Wiley , Hoboken, NJ , 2002 .

2. R. Chassaing , *Digital Signal Processing Laboratory Experiments Using C and the TMS320C31 DSK* , Wiley , Hoboken, NJ , 1999 .

3. R. Chassaing , *Digital Signal Processing with C and the TMS320C30* , Wiley , Hoboken NJ , 1992 .

4. R. Chassaing and D. W. Horning , *Digital Signal Processing with the TMS320C25* , Wiley , Hoboken NJ , 1990 .

5. N. Kehtarnavaz and M. Keramat , *DSP System Design Using the TMS320C6000* , Prentice Hall , Upper Saddle River, NJ , 2001 .

6. N. Kehtarnavaz and B. Simsek , *C 6x - Based Digital Signal Processing* , Prentice Hall , Upper Saddle River, NJ , 2000 .

7. N. Dahnoun , *DSP Implementation Using the TMS320C6x Processors* , Prentice Hall , Upper Saddle River, NJ , 2000 .

8. Steven A. Tretter , *Communication System Design Using DSP Algorithms With Laboratory Experiments for the TMS320C6701 and TMS320C6711* , Kluwer Academic , New York , 2003 .

9. J. H. McClellan , R. W. Schafer , and M. A. Yoder , *DSP First: A Multimedia Approach* , Prentice Hall , Upper Saddle River, NJ , 1998 .

10. C. Marven and G. Ewers , *A Simple Approach to Digital Signal Processing* , Wiley , Hoboken NJ , 1996 .

11. J. Chen and H. V. Sorensen , *A Digital Signal Processing Laboratory Using the TMS320C30* , Prentice Hall , Upper Saddle River, NJ , 1997 .

12. S. A. Tretter , *Communication System Design Using DSP Algorithms* , Plenum Press , New York , 1995 .

13. A. Bateman and W. Yates , *Digital Signal Processing Design* , Computer Science Press , New York , 1991 .

14. Y. Dote , *Servo Motor and Motion Control Using Digital Signal Processors* , Prentice Hall , Upper Saddle River, NJ , 1990 .

15. J. Eyre , The newest breed trade off speed, energy consumption, and cost to vie for an ever bigger piece of the action, *IEEE Spectrum* , June 2001 .

16. J. M. Rabaey , Ed., VLSI design and implementation fuels the signal - processing revolution, *IEEE Signal Processing* , Jan. 1998 .

17. P. Lapsley , J. Bier , A. Shoham , and E. Lee , *DSP Processor Fundamentals: Architectures and Features* , Berkeley Design Technology , Berkeley, CA , 1996 .

18. R. M. Piedra and A. Fritsh , Digital signal processing comes of age, *IEEE Spectrum* , May 1996 .

19. R. Chassaing , The need for a laboratory component in DSP education: a personal glimpse, *Digital Signal Processing* , Jan. 1993 .

20. R. Chassaing , W. Anakwa , and A. Richardson , Real - time digital signal processing in education , *Proceedings of the 1993 International Conference on Acoustics, Speech and Signal Processing (ICASSP)* , Apr. 1993 .

21. S. H. Leibson , DSP development software, *EDN Magazine* , Nov. 8, 1990 . 22. D. W. Horning , An undergraduate digital signal processing laboratory , *Proceedings of the 1987 ASEE Annual Conference* , June 1987 .

23. *TMS320C6000 Programmer ’ s Guide* , SPRU198G, Texas Instruments, Dallas, TX, 2002 .

24. *TMS320C6211 Fixed - Point Digital Signal Processor – TMS320C6711 Floating – Point Digital Signal Processor* , SPRS073C, Texas Instruments, Dallas, TX, 2000 .

25. *TMS320C6000 CPU and Instruction Set Reference Guide* , SPRU189F, Texas Instruments, Dallas, TX, 2000 .

26. *TMS320C6000 Assembly Language Tools User ’ s Guide* , SPRU186K, Texas Instruments,

Dallas, TX, 2002 .

27. *TMS320C6000 Peripherals Reference Guide* , SPRU190D, Texas Instruments, Dallas, TX, 2001 .

28. *TMS320C6000 Optimizing C Compiler User ’ s Guide* , SPRU187K, Texas Instruments, Dallas, TX, 2002 .

29. *TMS320C6000 Technical Brief* , SPRU197D, Texas Instruments, Dallas, TX, 1999 .

30. *TMS320C64x Technical Overview* , SPRU395, Texas Instruments, Dallas, TX, 2000 .

31. *TMS320C6x Peripheral Support Library Programmer ’ s Reference* , SPRU273B, Texas Instruments, Dallas, TX, 1998 .

32. *Code Composer Studio User ’ s Guide* , SPRU328B, Texas Instruments, Dallas, TX, 2000 .

33. *Code Composer Studio Getting Started Guide* , SPRU509, Texas Instruments, Dallas, TX, 2001 .

34. *TMS320C6000 Code Composer Studio Tutorial* , SPRU301C, Texas Instruments, Dallas, TX, 2000 .

35. *TLC320AD535C/I Data Manual Dual Channel Voice/Data Codec* , SLAS202A, Texas Instruments, Dallas, TX, 1999 .

36. *TMS320C6713 Floating - Point Digital Signal Processor* , SPRS186L, 2005 .

37. *TMS320C6414T, TMS320C6415T, TMS320C6416T Fixed - Point Digital Signal Processors (Rev. J)* , SPRS226J, 2006 .

38. TLV320AIC23 Stereo Audio Codec, 8 - to 96 - kHz, with Integrated Headphone Amplifier Data Manual, SLWS106G, 2003

39. TMS320C6000 DSP Phase - Locked Loop (PLL) Controller Peripheral Reference Guide, SPRU233.

40. *Migrating from TMS320C6211/C6711 to TMS320C6713* , SPRA851.

41. *How to Begin Development Today with the TMS320C6713 Floating - Point DSP* , SPRA809.

42. TMS320C6000 DSP/BIOS User ’ s Guide, SPRU423, 2002 .

43. TMS320C6000 Optimizing C Compiler Tutorial, SPRU425A, 2002 .

44. G. R. Gircys , *Understanding and Using COFF* , O ’ Reilly & Associates , Newton, MA , 1988 .