Department of Electrical and Electronic Engineering
Imperial College London

EE3-19

Real Time Digital Signal Processing

Course homepage: http://learn.imperial.ac.uk

Lab 2 - Learning C and Sine Wave Generation

Paul D. Mitcheson
paul.mitcheson@imperial.ac.uk
Room 1112, EEE

Objectives

- Learn how to create and build a project.
- Learn how to use functions in C.
- Learn how to generate a sinewave on the DSP board using table lookup.

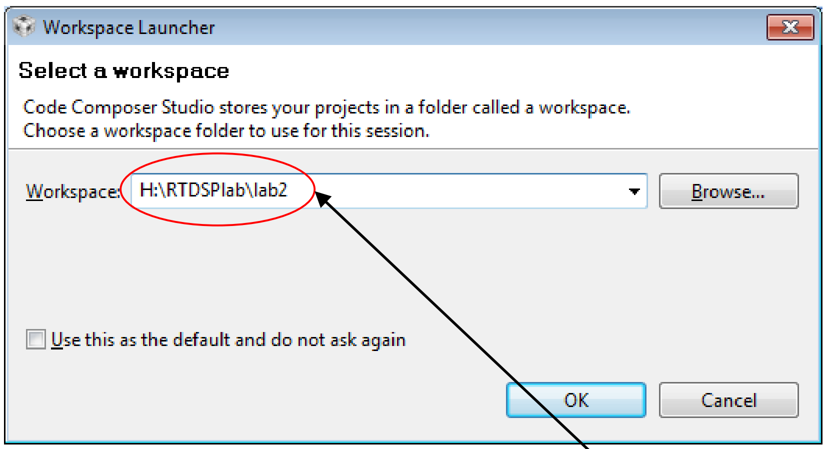
This lab does not count for marks towards your final grade in RTDSP. However, there are mark allocations given and if you submit the lab, feedback will be given. This should allow you to calibrate your style and reports with expectations ready for future labs which will be marked and will count for your final grade. Work in pairs.

Creating a Workspace for lab 2

As mentioned in lab 1 we have decided that you should create a workspace for each individual lab. Once you have created a workspace for this lab, the folder will provide a template for all subsequent labs. All you will have to do is copy and paste the folder and change the C file. This will save the majority of the tedious setting up of properties and dialogue boxes (there will only be a few slight tweaks in future labs). As a consequence you must get this lab set up properly in order to carry out subsequent labs.

- Start CCS by double clicking the icon on the desktop (or using the start menu).

- You are asked to select (or create a new) workspace. Create a new workspace called ***lab2*** under your ***RTDSPlab*** folder. Ensure the **Use this as the default and do not ask again** check box is **NOT** checked. The dialogue should look like figure 1.


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: H:\RTDSPlab\lab2

☐ Use this as the default and do not ask again

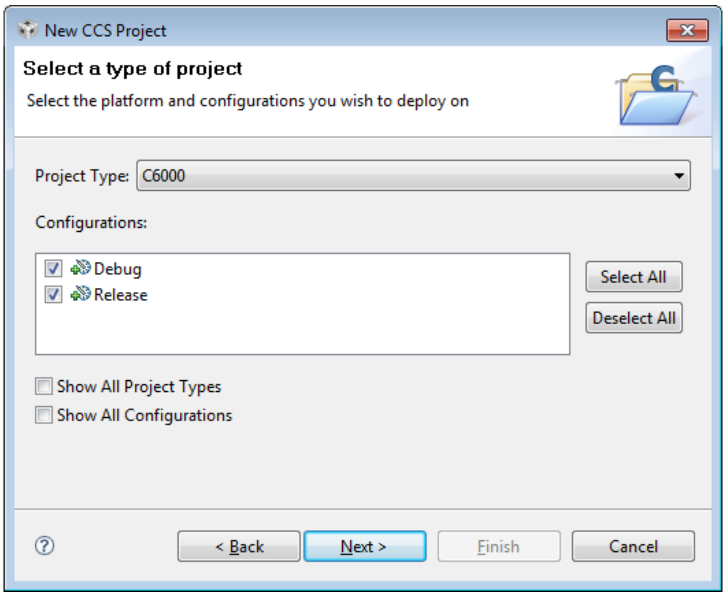
Ensure that any paths given here use the drive letter (e.g. h:\path\) and **not** the UNC path (e.g. icfs1.cc.ic.ac.uk\path\)

Figure 1: Creating a new workspace for lab2

- Hit **Ok** when done.
- Close the welcome screen that opens (by hitting the x on the tab).

Creating a project

- Create a project by using **File→New→CCS Project,** call it ***RTDSP*** by typing in the **Project Name** box (this also creates a ***RTDSP*** subfolder below ***RTDSPlab\lab2***)>
- Ensure that **Use default location** is checked
- Hit **Next**
- Select **C6000** for the **Project Type** (see figure 2)


New CCS Project

Select a type of project

Select the platform and configurations you wish to deploy on

Project Type: C6000

Configurations:

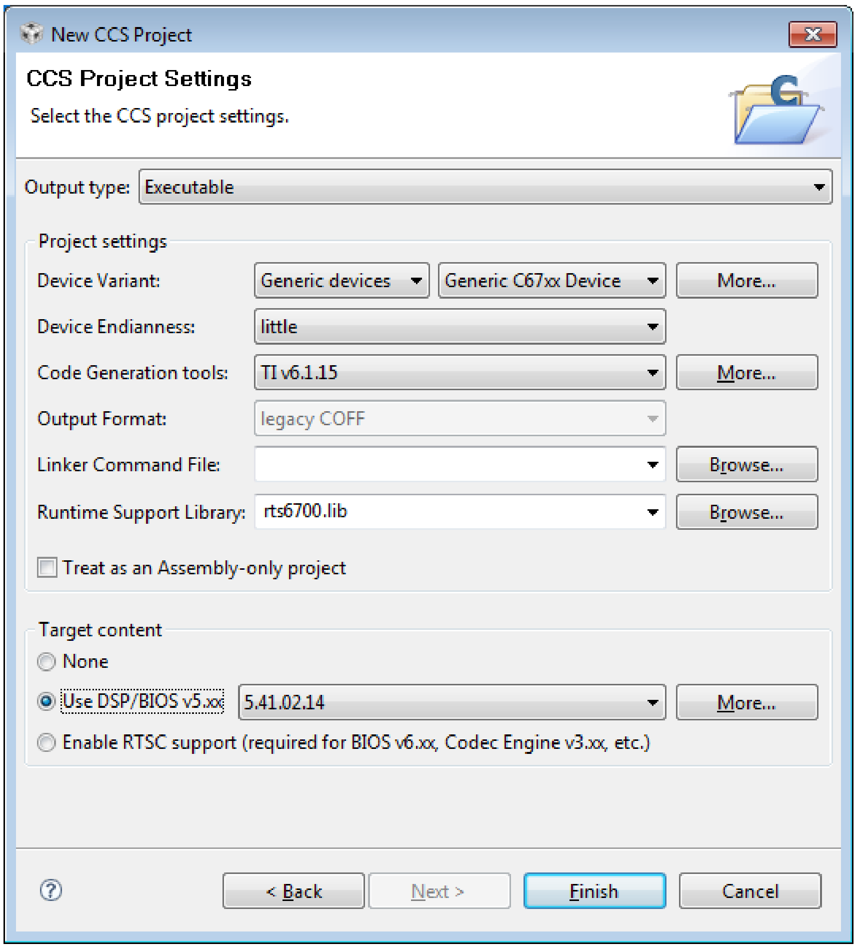
- [✓] Debug
- [✓] Release

- [ ] Show All Project Types
- [ ] Show All Configurations

Figure 2: Setting up a new project

- Hit **Next** twice (i.e. ignore the dialogue regarding **Additional Project Settings**.) Set up the dialogue as shown in figure 3. An important change from lab 1 is that the **Use DSP/BIOS** radio button is selected. This is a significant change. For this lab and subsequent ones you will be using the DSP/BIOS configuration tool to simplify some of the tasks required for creating working programs. The configuration tool is a graphical environment for modularly adding the software components you require for a particular project. For this lab we are using the configuration tool to automatically handle memory mapping but we will extend the use of the configuration tool in later labs – to assign interrupts). [1]

[1] If you cast your mind back to lab1 you will notice there were no configuration files, but there was a **vectors.asm** file and also one called **volume.cmd** that were added into the project. The latter file defines the memory mapping, and the former file defines where in memory the code should begin execution. By using the configuration tool these tasks are managed automatically.


New CCS Project

CCS Project Settings

Select the CCS project settings.

Output type: Executable

Project settings

Device Variant:	Generic devices	Generic C67xx Device	More...
Device Endianness:	little		
Code Generation tools:	TI v6.1.15		More...
Output Format:	legacy COFF		
Linker Command File:			Browse...
Runtime Support Library:	rts6700.lib		Browse...

[ ] Treat as an Assembly-only project

Target content

( ) None

(●) Use DSP/BIOS v5.xx — 5.41.02.14 — More...

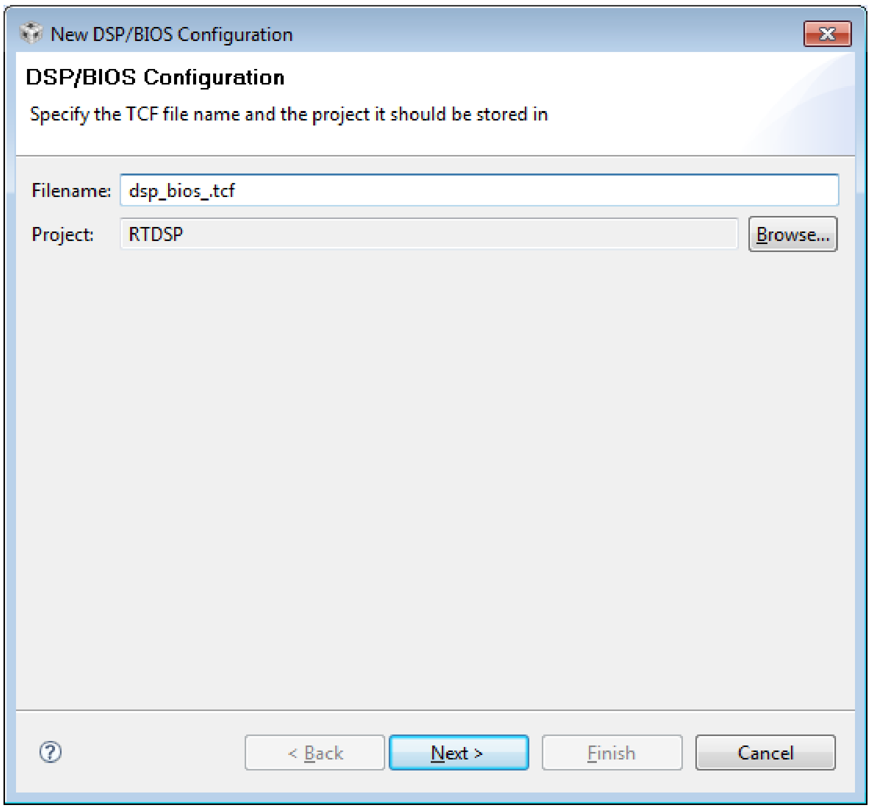
( ) Enable RTSC support (required for BIOS v6.xx, Codec Engine v3.xx, etc.)

Figure 3: Final dialogue to set up project

- Hit **Finish.**
- Download the ***lab2.zip*** folder from Blackboard. Unzip the contents into ***H:\RTDSPlab\Lab2\RTDSP\*** folder.
- In order to navigate and see what files are held in your project you need to use the **C/C++ Projects** window. Back in the main window hit the following icon on the bottom left hand side of the screen
- In the pop up choose **C/C++ Projects**. This creates another icon on the bottom left of the screen that will allow you to reach **C/C++projects** window quickly.

Setting up DSP/BIOS

- Select the menu **File→new→DSP/BIOS v5.xx Configuration File**
- In the **Filename** box type dsp_bios_.tcf so that it looks like figure 4. Note there is an underscore between BIOS and tcf



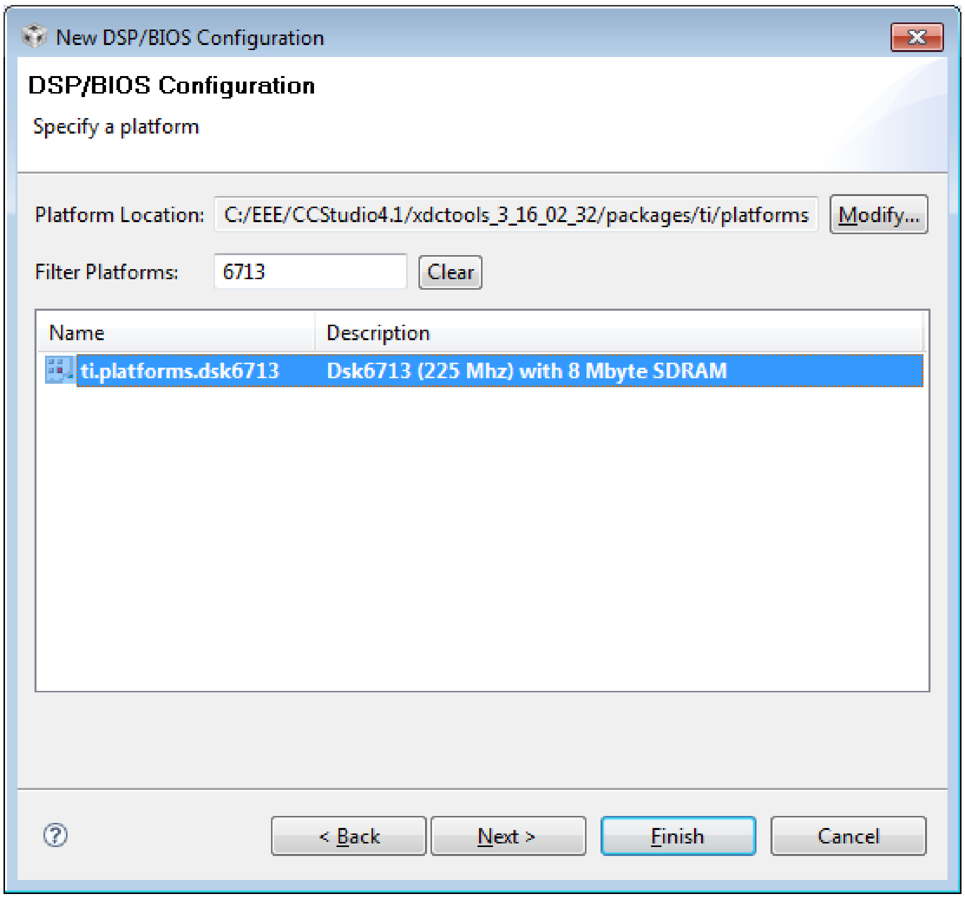
Specify the TCF file name and the project it should be stored in

Filename: dsp_bios_.tcf

Project: RTDSP

Figure 4: Setting up DSP/BIOS

- Make sure you have spelled the filename correctly or the project will not compile and you will have to start over again.
- Hit **Next**


New DSP/BIOS Configuration

DSP/BIOS Configuration

Specify a platform

Platform Location: C:/EEE/CCStudio4.1/xdctools_3_16_02_32/packages/ti/platforms

Filter Platforms: 6713

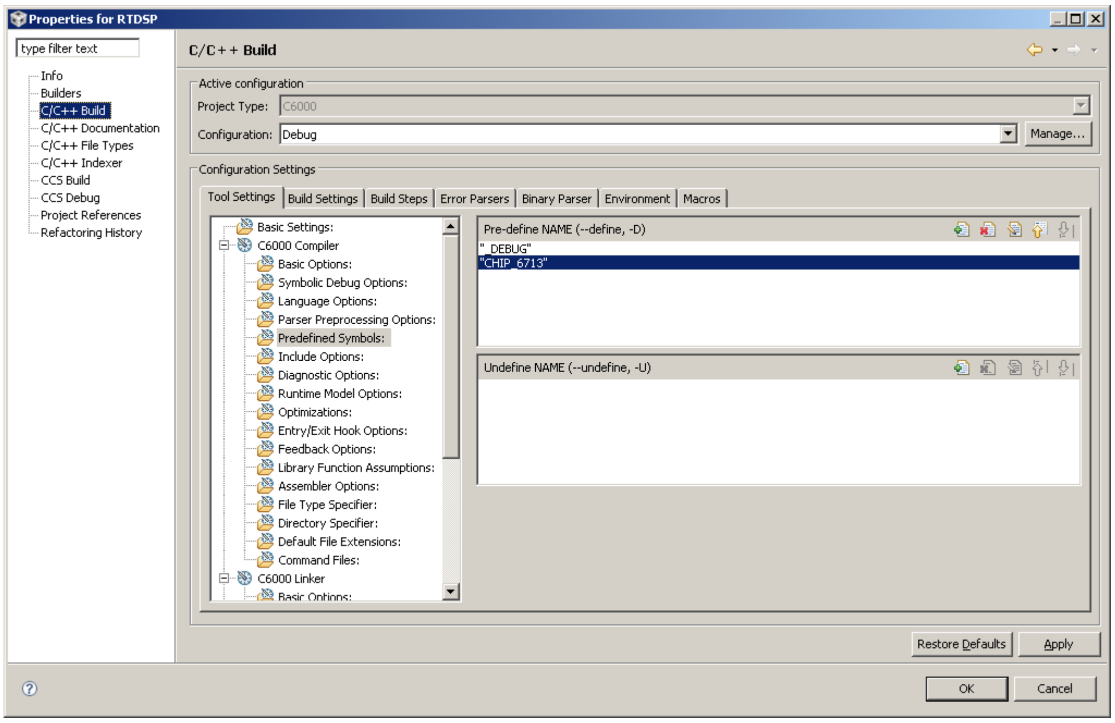
Name	Description
ti.platforms.dsk6713	Dsk6713 (225 Mhz) with 8 Mbyte SDRAM

Figure 5: DSP/BIOS

- Figure 5 shows the necessary settings for the next window. Make sure the line with **Dsk6713 (255 Mhz) with 8 Mbyte SDRAM** is selected and hit **Next.**
- Ignore the next window by hitting **finish** (but leave everything selected).
- After a few moments the DSP/BIOS Configuration tool will open. You will be modifying settings in this window in later labs. For this lab, everything is ok so close the **Configuration Tool** window.

Setting up links to code Libraries

- Open up a window with the code for ***sine.c*** by double clicking on ***sine.c*** in the C/C++ Projects window.
- Ensure your cursor is somewhere in the code window.
- Select the menu **Project→Properties**
- Select the **C/C++Build** option in the tree structure on the left hand side of the **Properties** dialogue.
- In the **C6000 Compiler** section select the **Predefined Symbols** option
- add "_DEBUG" and "CHIP_6713" so that the window looks like figure 6. (A text file ***compiler-linker.txt*** was included in the lab zip file so that you can copy and paste the text if you wish)



Project Type: C6000

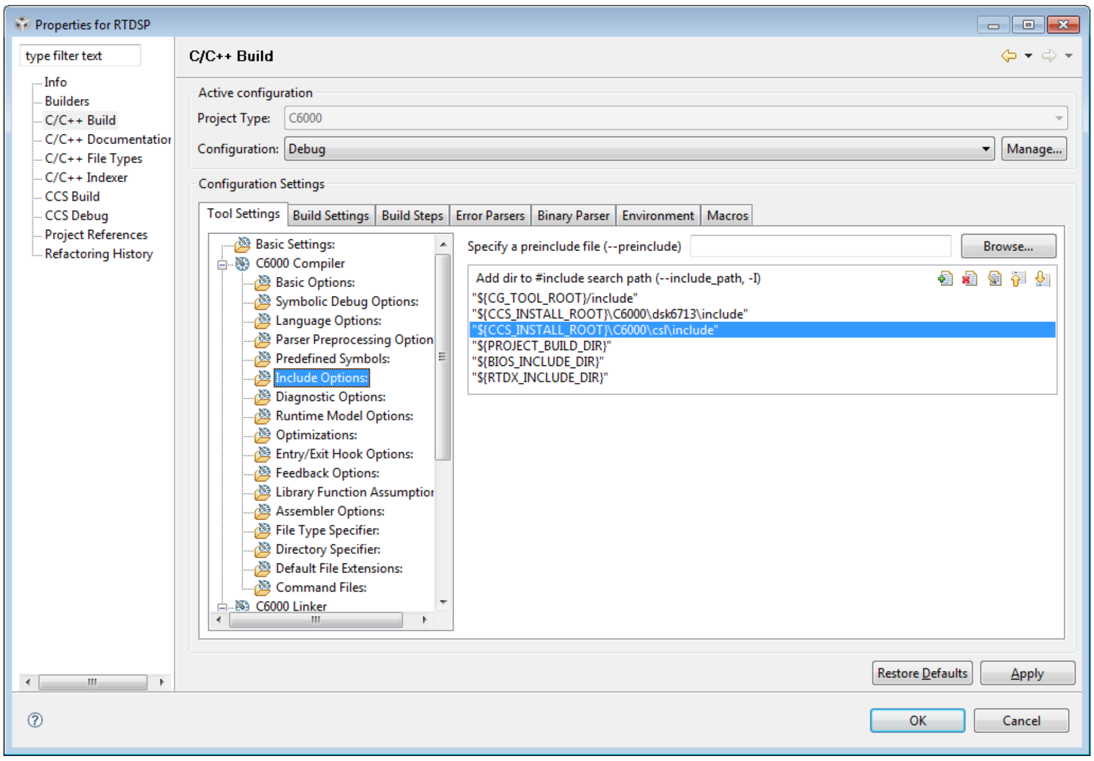
Configuration: Debug

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

"_DEBUG"
"CHIP_6713"

Figure 6: Adding some symbols

- Move down to the **include options:** menu in the **C6000 Compiler** section.
- Add the 2 paths shown in figure 6 into the pre-include window. (A text file ***compiler-linker.txt*** was included in the lab zip file so that you can copy and paste the text if you wish)



C/C++ Build

Active configuration

Project Type: C6000

Configuration: Debug

Configuration Settings

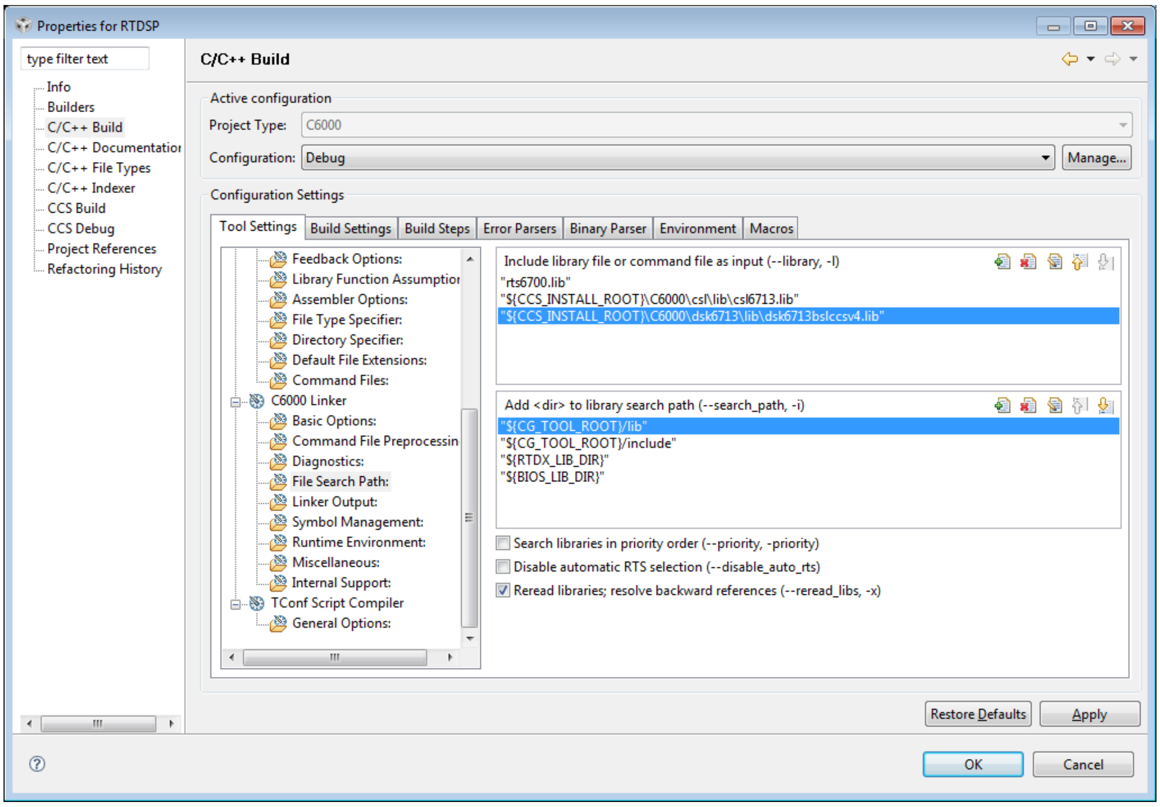
Specify a preinclude file (--preinclude)

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

```
"${CG_TOOL_ROOT}/include"
"${CCS_INSTALL_ROOT}\C6000\dsk6713\include"
"${CCS_INSTALL_ROOT}\C6000\csl\include"
"${PROJECT_BUILD_DIR}"
"${BIOS_INCLUDE_DIR}"
"${RTDX_INCLUDE_DIR}"
```

Figure 6: Compiler includes

- Move down to the **File Search Path:** menu in the **C6000 Linker** section.
- Add the 2 paths shown in figure 7 into the include library file window. (A text file ***compiler-linker.txt*** was included in the lab zip file so that you can copy and paste the text if you wish)


Properties for RTDSP

C/C++ Build

Active configuration
Project Type: C6000
Configuration: Debug

Configuration Settings

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

```
"rts6700.lib"
"${CCS_INSTALL_ROOT}\C6000\csl\lib\csl6713.lib"
"${CCS_INSTALL_ROOT}\C6000\dsk6713\lib\dsk6713bslccsv4.lib"
```

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

```
"${CG_TOOL_ROOT}/lib"
"${CG_TOOL_ROOT}/include"
"${RTDX_LIB_DIR}"
"${BIOS_LIB_DIR}"
```

- [ ] Search libraries in priority order (--priority, -priority)
- [ ] Disable automatic RTS selection (--disable_auto_rts)
- [x] Reread libraries; resolve backward references (--reread_libs, -x)

Figure 7: Linker includes

- Finally hit **Apply** and then **Ok**

Before attempting compile the project take some time to study the code. This is covered in the next section.

The sine wave generation program

Take a look at the code for **sine.c** in CCS. Scroll the window down to the main() function. The main program does the following:

- Initializes the audio I/O ports using the init_hardware() function which is defined below main(). Init_hardware() makes use of predefined library functions; we will look at these functions in more detail in a later lab.
- A while loop runs forever, writing out data calculated in the subroutine sinegen(). Currently this subroutine simply outputs a sine wave of fixed frequency (the same data is sent to both left and right audio channels). In the reminder of the lab you will write a routine that outputs a sine wave of a user-defined frequency (using a look up table).

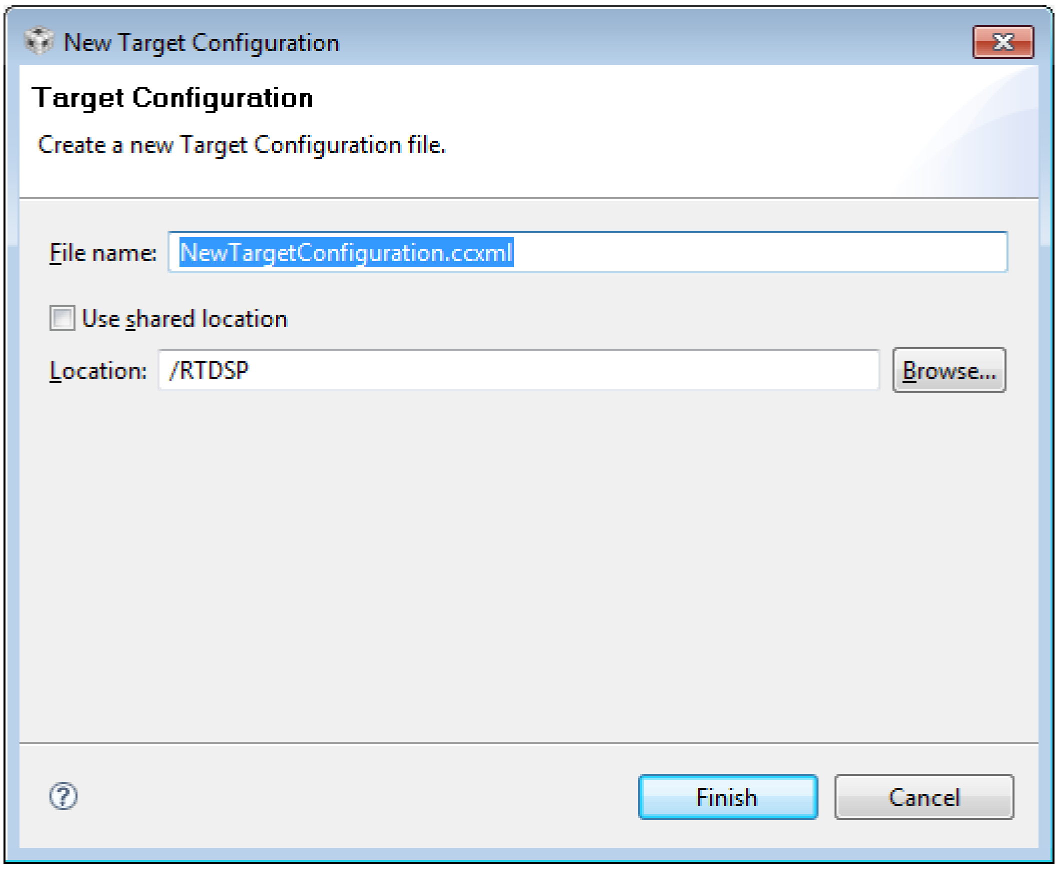
Building the project

- Now build the project **Project→Rebuild Active Project.** A successful build results in the **problems** pane reporting: **0 errors, 0 warnings, 0 info's**. Once the project has successfully compiled and linked, it will create a file ***sine.out*** that is the DSK executable file. If you do not get a successful build read the errors and fix the problem before moving on to the next step.

The Target Configuration

Having built the program it can be downloaded to the DSK and run. Before you can do this the project needs to know about the target (i.e. the hardware) and it does this via the target configuration file.

- Hit **File→New→Target Configuration File**
- Leave the default filename selected and ensure that **Use shared location** is **NOT** selected, set the location to ***/RTDSP*** see figure 8.
- Hit **Finish**


New Target Configuration

Target Configuration

Create a new Target Configuration file.

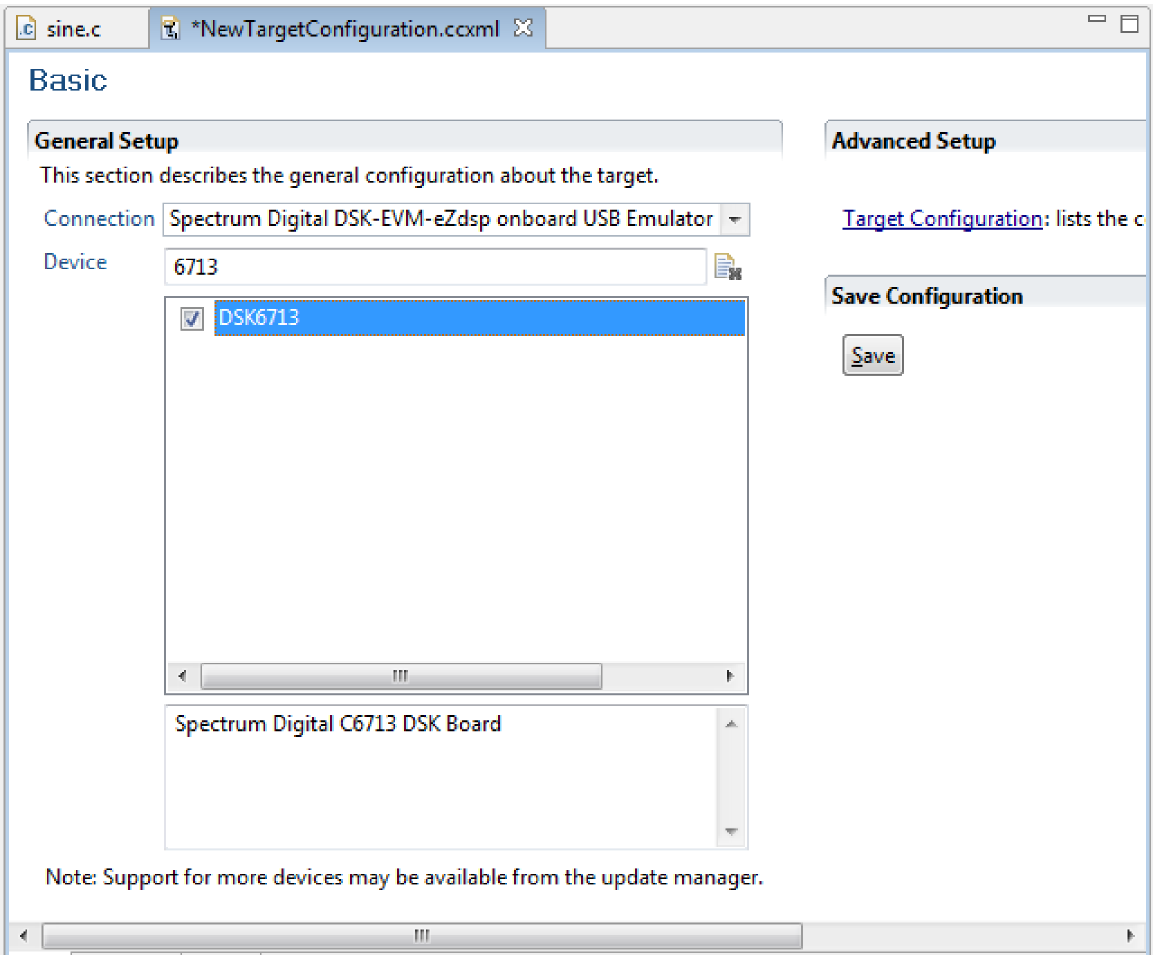
File name: NewTargetConfiguration.ccxml

☐ Use shared location

Location: /RTDSP

Figure 8: Target Configuration

- Figure 9 shows the settings to ensure that you can talk to the 6713 hardware. After you have completed these settings hit the **Save** button.
- You can close the Target configuration window if you wish.



sine.c | *NewTargetConfiguration.ccxml

Basic

General Setup

This section describes the general configuration about the target.

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

Device: 6713

☑ DSK6713

Spectrum Digital C6713 DSK Board

Note: Support for more devices may be available from the update manager.

Advanced Setup

Target Configuration: lists the c

Save Configuration

Figure 9: Attaching to the hardware

Load and run the program

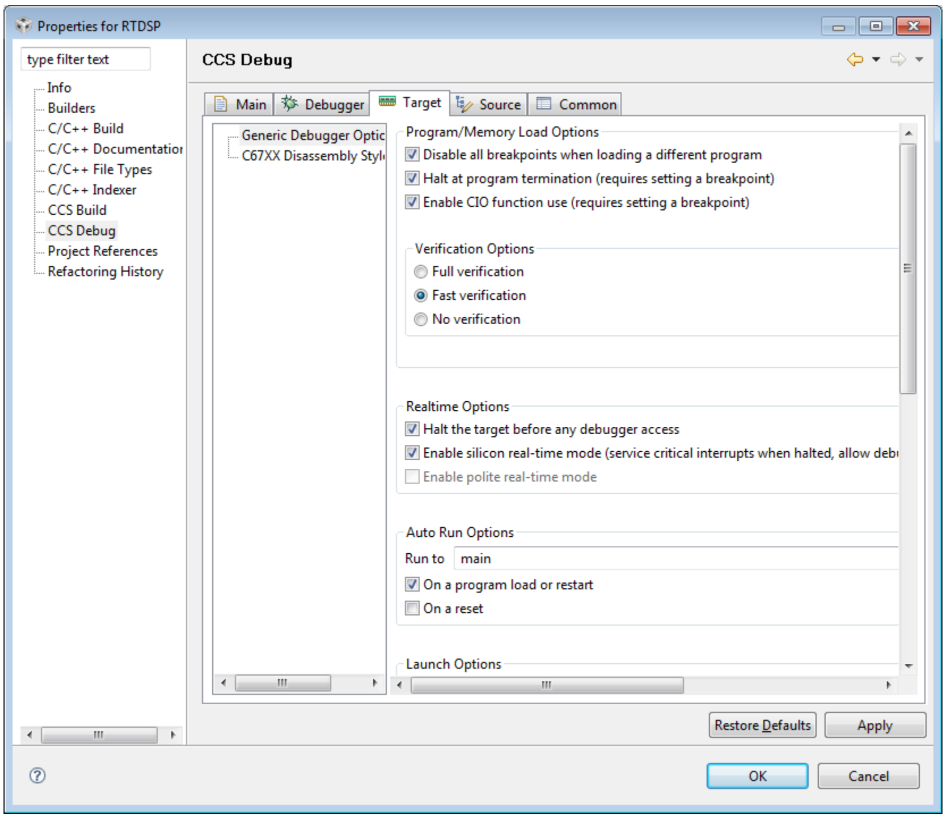
- Re-cycle the power on the DSK board. Ensure the USB cable is linking the DSK and the PC.
- Hit **Target→Debug Active Project** This will put CCS in debug mode, connect CCS to the target and also send the executable file to the DSK board.
- Connect an oscilloscope to either left or right line out on the DSK board, (please use the coax connector on the aluminium strip to save the min-jacks from wear and tear) and then run the program using **Target→Run.** Ensure that the output is a sinewave.
- Halt the program using **Target→Halt**.

Whenever the program is modified and rebuilt it can be reloaded to the DSK by using **Target→Reload Program**.

Enabling real-time modification of variables

A final tweak is required to enable the updating of watch window variables on the fly.

- Place the cursor somewhere in the code window of ***sine.c***
- Hit **Project→Properties**
- Choose **CCS Debug** from the tree diagram
- Choose the **Target** tab
- For the real- time options, ensure the two boxes shown in figure 10 are checked.
- Hit **Apply** then **Ok**



CCS Debug

Program/Memory Load Options
- [✓] Disable all breakpoints when loading a different program
- [✓] Halt at program termination (requires setting a breakpoint)
- [✓] Enable CIO function use (requires setting a breakpoint)

Verification Options
- ○ Full verification
- ● Fast verification
- ○ No verification

Realtime Options
- [✓] Halt the target before any debugger access
- [✓] Enable silicon real-time mode (service critical interrupts when halted, allow debu
- [ ] Enable polite real-time mode

Auto Run Options

Run to main
- [✓] On a program load or restart
- [ ] On a reset

Launch Options

Figure 10: Enabling modification of variables

A Note about code writing and style

In the software industry programmers generally work in teams and are expected to follow style conventions and communicate their ideas via reasonable commenting and variable naming. Additionally, programs should be easy to update and modify through the use of symbols.

We expect you to follow a style similar to the code examples given to you in the code templates.

Some hints:

- Use whitespace to help readability
- Use symbolic constants so the constant can be modified in one place
- Use tabs appropriately around structures such as if/for/while – tab in again for nested structures
- Use comments to give an abstract, higher level view of what the code is doing
- Place repeated sections of code into procedures/functions
- Use sensible names for variables

Marks are available for well written and presented code in your reports.

Generating a sine wave

Currently, the function sinegen() generates a sine wave with fixed frequency of 1 kHz (For a sampling frequency of 8kHz). This method uses an IIR Filter. In the remainder of the lab you will write a new function so that the frequency that is output is determined by the value of the global variable sine_freq. The sine wave generation will use an alternative method called "using a look up table".

For this lab we do not wish you to spend any time on understanding IIR filters, this will come in a later lab. Make sure you understand how this IIR filter generates the values by hand calculating a series of values. For anyone who is interested and has the time the appendix contains a derivation of the IIR filter (this section will not be assessed).

Questions

1. **Provide a trace table of Sinegen for several loops of the code. How many samples does it have to generate to complete a whole cycle?**

[2]

2. **Can you see why the output of the sinewave is currently fixed at 1 kHz? Why does the program not output samples as fast as it can? What hardware throttles it to 1 kHz? (If you are having problems working this out try changing the sampling frequency[2] by changing sampling_freq).**

[4]

3. **By reading through the code can you work out the number of bits used to encode each sample that is sent to the audio port?**

[1]

The simplest way of generating a sine wave signal is by using a sine lookup table. We will define a lookup table in memory, and then modify sinegen() so that it uses this table to output a sine wave of the desired frequency.

1. Define a global symbolic constant SINE_TABLE_SIZE that defines how many values are in the table, and assign it a value of 256. (See the definition of PI for how to declare global symbolic constants using #define.)
2. Define a global variable called table that is an array of floats containing SINE_TABLE_SIZE elements. This is done by inserting the following line in the section of the program designated as being for global variables:

   ```
   float table [SINE_TABLE_SIZE];
   ```

3. Notice that the definition for y defines an array of 3 floats, and assigns some initial values. The above definition for table does not assign initial values. In C, arrays are indexed starting from 0, so an array of 10 values would be indexed 0 through 9. Individual elements of an array are accessed by using the variable name followed by the index in square brackets. For example, to assign the zeroth element of table to 0.2, you would use:

   ```
   table [0] = 0.2;
   ```

4. Having defined how big the table is and where it will be stored in memory, we now need to create a function that will fill it with the appropriate sine values. Create a function sine_init() that will do this. It will have no parameters and return no value; its purpose is

[2] You can set the variable to one of the supported sampling frequencies: 8000, 16000, 24000, 32000, 44100, 48000 or 96000. Note that commercial music CD’s use 44100.

simply to initialize the global variable table. This will involve defining the function prototype (look at the definition of init_hardware() at the top of ***sine.c*** for how to do this). Then you will need to write the implementation of this function. It should consist of a *for loop* (do a google search for a suitable C programming guide for the syntax) that fills table with the appropriate number of points within one period of a sine wave. In the math.h library file (which is included in the shell program), there is a function sin() that returns the sine of its argument (in radians). A global symbolic constant PI is assigned a value of 3.141592653589793 at the top of sine.c for you to use in your calculations.

5. Add a call to sine_init() in the main program, just before the while loop. Rebuild the project and download it to the DSP. Set a breakpoint just after the call to sine_init(), and run the program. To check that your function is successful, look at the memory starting at table (use **View→Memory**)[3] and confirm that 256 points within a single period of a sine wave are stored there. You could also do this by using the graph function (**Tools→Graph→Single Time**)[3]. Ensure that sine_init() is working correctly before proceeding.
6. Now rewrite the function sinegen() so that it reads the appropriate value from the sine table and returns this value to the main routine. You will need to use the global constant SINE_TABLE_SIZE that you declared earlier and the global variable sampling_freq which has been pre-defined for you.
7. Since the frequency of the sine wave is determined by the global variable sine_freq, you can easily change the output frequency by changing the value of this variable. Open a watch window (**View→Watch**) and insert sine_freq. By changing the value of sine_freq within the watch window you will be able to update the frequency of the sine wave without requiring you to recompile the code[4]. Also try setting different sampling frequencies by adding the variable sampling_freq to the watch window and changing it to one of the supported sampling frequencies: 8000, 16000, 24000, 32000, 44100, 48000, or 96000.
8. In order to gain good marks for this part of the assignment you must have succeeded in all of the following:
   - Your calculation should account for the change in sampling frequency by outputting the same sine frequency sine regardless of the current sampling frequency[4].
   - Ensure that the sine frequency output is accurate (as compared to a scope measurement) and the sine has no distortions or harmonics for all applicable frequencies.
   - Ensure the algorithm will support frequencies from around 10Hz up to just below the Nyquist Frequency.
   - Is your code easy to read and modify? Ensure your code is properly commented and any variable names are sensible. Make sure you have used symbolic constants appropriately.

Comments

Within the existing function sinegen(), the variable wave is a local variable. Any variable that is defined within a function is local, and it is undefined outside that particular function. Moreover, when the function is exited the local variable is destroyed. If it is necessary for a variable to keep its value even after a local function is exited, or from one function call to the next, it should be defined as a global variable, by inserting its definition before main (as you just did when defining table) or define the variable as local to the main program and pass the variable from function to function. You will probably need to define one more variable as a global variable, and use it to keep track of where you are within the sine table from one call of sinegen() to the next.

[3] Remember to set the correct data type!

[4] The sine frequency should not be set above sampling frequency / 2.

Deliverables

You **do not** have to write a formal report including abstract, conclusions etc for this lab. You are, however, required to write a tidy short write-up which provides program listings and evidence (e.g. graphs) that you have done the exercises. Ensure you cover the points made below and make your report easy to read.

The lab is marked out of [29]. Marks are awarded as follows:

- The 3 questions asked above in **Arial Bold Text**

  [7]

- Paragraph explaining the operation of your code, using code snippets if required. Think about how to increase the resolution of the output without using a larger lookup table.

  [7]

- Scope traces showing your code operates as expected

  [3]

- Discuss the limitations on upper and lower bounds of frequencies that can be generated on this system. What do you observe happening to the output as you approach what you consider to be the upper and lower limits of operation? Why?

  [8]

- Full readable code listing with comments (as an appendix to your report)

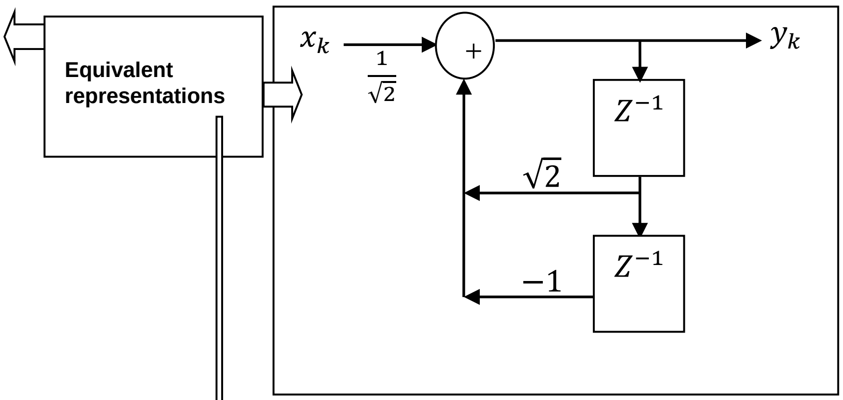
  [4]

Please submit your assignment via the Blackboard upload page for lab 2.

Appendix 1 – explanation of original sinegen function

Difference equation

$y_k = \sqrt{2}y_{k-1} - y_{k-2} + \frac{1}{\sqrt{2}}x_k$



Code from loop

```
float wave;
// represents square root of 2
float a0 = 1.4142;
float b0= 0.707;
y[0] = a0*y[1]  - y[2] + b0*x[0];
y[2] = y[1];   // rotate buffer
y[1] = y[0];
x[0] = 0; // remove input
wave = y[0];
return(wave);
```

Take Z transforms

$$\mathcal{Z}\{y_k\} = \sqrt{2}\mathcal{Z}\{y_{k-1}\} - \mathcal{Z}\{y_{k-2}\} + \frac{1}{\sqrt{2}}\mathcal{Z}\{x_k\}$$

Shift theorem

$$Y(z) = \sqrt{2}Y(z)z^{-1} - Y(z)z^{-2} + \frac{1}{\sqrt{2}}X(z)$$

Collect Y(z) terms

$$Y(z) - \sqrt{2}Y(z)z^{-1} + Y(z)z^{-2} = \frac{1}{\sqrt{2}}X(z)$$

Take out Y(z) factor

$$Y(z)(1 - \sqrt{2}z^{-1} + z^{-2}) = \frac{1}{\sqrt{2}}X(z)$$

Find transfer function

$$\frac{Y(z)}{X(z)} = \frac{\frac{1}{\sqrt{2}}}{1-\sqrt{2}z^{-1}+z^{-2}}$$

Compare to the general form of an IIR filter below.
This is a 2nd order IIR Filter.

$$\frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1} + \cdots + b_n z^{-n}}{1 + a_1 z^{-1} + \cdots + a_n z^{-n}}$$

Now multiply by $\frac{z^2}{z^2}$ this gives the transfer function

$$\frac{Y(z)}{X(z)} = \frac{\frac{1}{\sqrt{2}}z^2}{z^2-\sqrt{2}z+1}$$

This is in a form very similar to that of the z transform for a sine function (If we set ωT to $\pi/4$), where:

$$\frac{Y(z)}{X(z)} = \frac{\mathrm{z}\sin\emptyset}{z^2 - 2\,\mathrm{z}\cos\emptyset + 1} \quad where\ \emptyset = \omega T$$

The only difference is the extra Z multiplier in the numerator, which is just a 1 sample advance.

Finding the Poles

Back to the transfer function

$$\frac{Y(z)}{X(z)} = \frac{\frac{1}{\sqrt{2}}z^2}{z^2-\sqrt{2}z+1}$$

Complete the square

$$\frac{Y(z)}{X(z)} = \frac{\frac{1}{\sqrt{2}}z^2}{\left(z-\frac{\sqrt{2}}{2}\right)^2 - \frac{1}{2} + 1}$$

Tidy up

$$\frac{Y(z)}{X(z)} = \frac{\frac{1}{\sqrt{2}}z^2}{\left(z-\frac{1}{\sqrt{2}}\right)^2 + \frac{1}{2}}$$

Poles occur when denominator is zero i.e.

$$\left(z-\frac{1}{\sqrt{2}}\right)^2 + \frac{1}{2} = 0$$

Manipulate

$$\left(z-\frac{1}{\sqrt{2}}\right)^2 = -\frac{1}{2}$$

Square root of minus number yields a complex number

$$z - \frac{1}{\sqrt{2}} = {}^{+}/{-}\ j\frac{1}{\sqrt{2}}$$

Finally find poles. Notice there are 2 poles which are conjugate symmetric

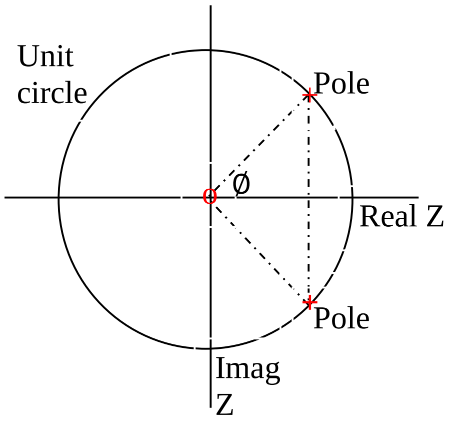
$$z = \frac{1}{\sqrt{2}} +/- \, j\frac{1}{\sqrt{2}}$$

Also notice when finding the modulus of the poles they both have a length of 1 so they lie on the unit circle

$$|z| = \sqrt{\left(\frac{1}{\sqrt{2}}\right)^2 + \left(\frac{1}{\sqrt{2}}\right)^2} \qquad \emptyset = \arctan\left(\frac{1}{\sqrt{2}} \Big/ \frac{1}{\sqrt{2}}\right) = \frac{\pi}{4}$$

$$|z| = \sqrt{\frac{1}{2} + \frac{1}{2}}$$

$$|z| = 1$$

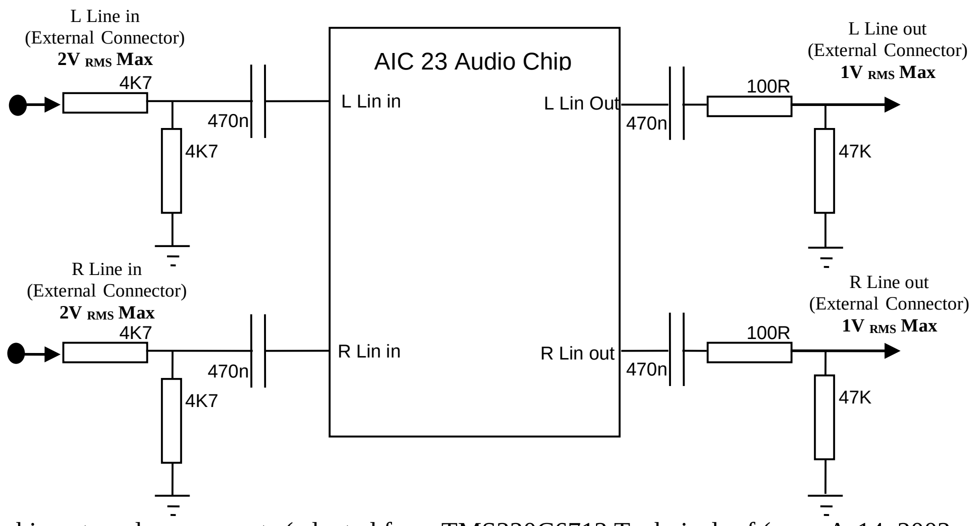


These poles can be expressed in polar form as:

$z = 1e^{\frac{j\pi}{4}}$ and $z = 1e^{-\frac{j\pi}{4}}$

As can now be seen, the system has two complex conjugate poles, which are on the unit circle. These poles therefore exhibit marginal stability, in other words, if the system is excited, it will oscillate forever at frequencies set by those poles, without the oscillation growing or decaying.

Appendix 2 –Analogue IO configuration on the 6713 DSK



AIC23 Audio chip external components (adapted from TMS320C6713 Technical ref (page A-14, 2003 revision A)

Revision History

1st Feb 2010 – Add mark allocation and tidy up appendix
1st Oct 2010 – Update to CCS v4
3rd January 2012 – Add appendix 2
20th Jan 2014 – lab no longer counts towards final grade for course