Assignment #1

Due Monday February 10, 2014

Make note of the following:

- Each team of two will turn in documentation for the assigned problem(s), that is, assembly or C source code as appropriate.
- Turn in a hard copy of hand calculations, with answers in hex format if appropriate to verify the running program results.
- For this first assignment I would like to see turned the profiling results, in a simple report format, that you obtain under various conditions, as explained under problem 4.
- I will ask for a demo of the working program(s) and the usage of Code Composer Studio features during the lab session.

Problems:

0. If required re-flash the board using the instructions handed out in class. This is a very important first step, so be very careful to follow the instructions. This step can be omitted unless you hear otherwise from your instructor.
1. Work through text Appendix A (`App_CCS_5_1_omapl138.pdf`), Code Composer Studio (CCS): A Brief Tutorial for the OMAP-L138. When the code is up and running verify that audio from a music source entering the *line in* jack can be heard via the PC Speakers found at each lab station.
   If desired take a look at the waveform on the scope. It will appear to be noisy untless youj are using a lowpass filter cable. See the Appendix of this document for an explaination. In a later experiment you will drill down on the codec interface (ADC/DAC) interface and its characteristics.
   You will use CCS 5.5 for this exercise. The instructions for CCS 5.1 still apply.
2. Starting from the codebase for the project `sine8_buf_files` (found in `set1.zip`), discussed in lecture, and captured on video, repeat some of the same experiments on your own. In particular (see the appendix for more details):
   a.) Set break points
   b.) Add watch variables
   c.) Memory view
   d.) Graphs, both time domain and frequency domain. See the Appendix for more details on CCS Graphs.
   e.) Observe the output waveform via the PC Speakers and on the scope. Is it a 1 kHz sinusoid?
   f.) Experiment with the general extension language (GEL) gain slider. See the appendix of this assignment sheet for more details on GEL files.
   g.) In `Codec_ISR()` observe what happens if you uncomment `switches = ReadSwitches()` or `WriteLEDs(LedMask)`.

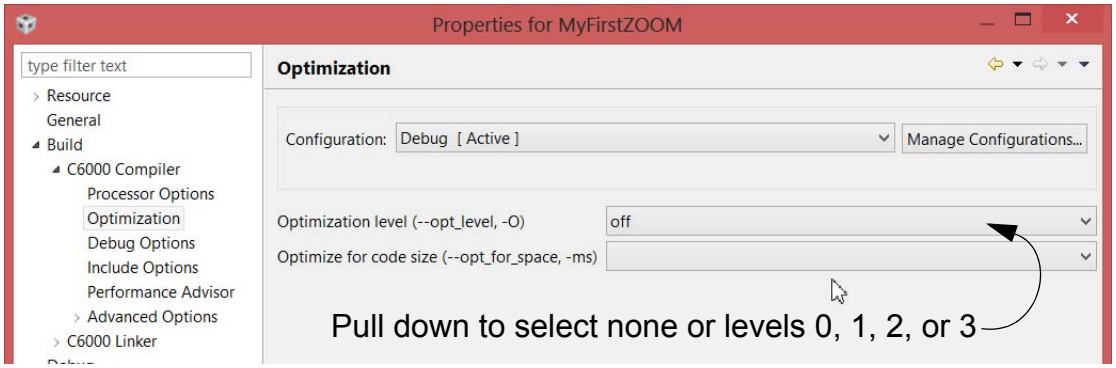
h.) Change the codec sampling rate from 8 kHz to 24 kHz by commenting and uncommenting lines in the header file `DSP_Config.h`. Can you predict the new frequency for the sinusoid? What is it from a scope measurement?

```
// uncomment just the line for the sample rate when using the OMAP-L138
//#define SampleRateSetting AIC3106Fs48kHz// 48kHz sample rate
//#define SampleRateSetting AIC3106Fs96kHz// 96kHz sample rate
//#define SampleRateSetting AIC3106Fs32kHz// 32kHz sample rate
//#define SampleRateSetting AIC3106Fs24kHz// 24kHz sample rate
//#define SampleRateSetting AIC3106Fs16kHz// 16kHz sample rate
//#define SampleRateSetting AIC3106Fs12kHz// 12kHz sample rate
#define SampleRateSetting AIC3106Fs8kHz//  8kHz sample rate
```

3. Starting from the codebase for the project `dot4p` (files found in `set1.zip`), discussed in lecture, and captured on video, and eventually documented in the course notes, repeat some of the same experiments on your own. In particular:

   a.) Use the OmapL138 target configuration file, use the simple (profile) clock under the `Run` menu, `Clock` menu fly out, time the `dotp()` function under the compiler optimizations `none`, `o0`, `o1`, and `o2`. Note you cannot set break points when `o3` is invoked.

   Under the Project menu you will find the Properties item (or alt+enter). This brings up a dialog box that allows you to change compiler optimization settings:



   b.) Remove the OmapL138 target configure file and create a new target configuration file for the simulator `C674x CPU Cycle Accurate Simulator, Little Endian`. Repeat the cycle timings from part (a) using the simple clock.

   c.) Repeat part (b), except now you will configure the Profiler. Note that the Profiler is not available for c6x targets other than the simulator.

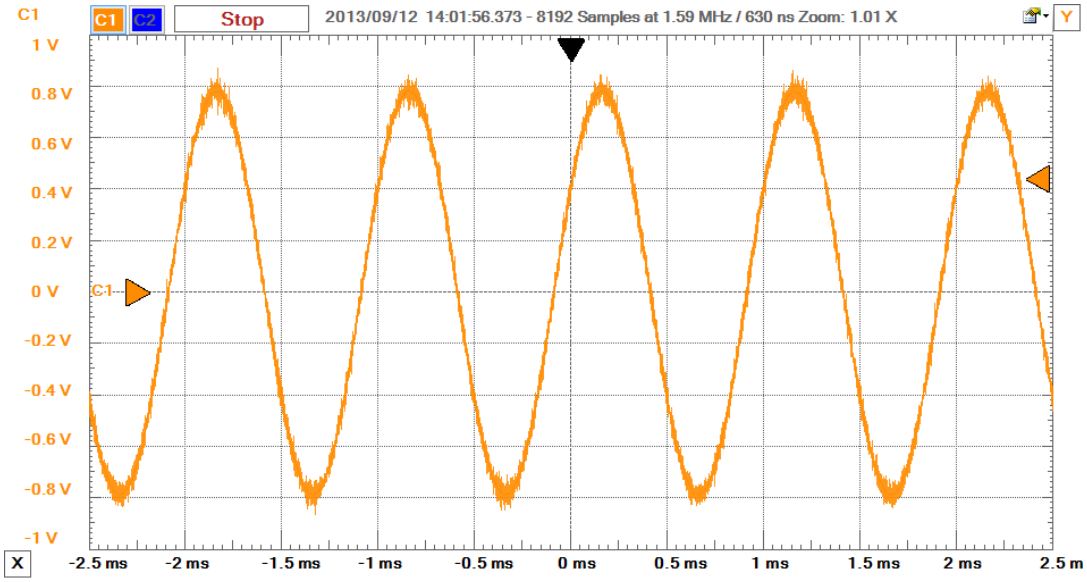
4. In this problem you will modify the table-lookup sinusoid generation technique of `sin8_buf`. We desire a sinusoid at 4 kHz. Pick a sample rate from the header file and construct an appropriate table, `sine_table`. You will demo this in lab to the instructor.

Appendix

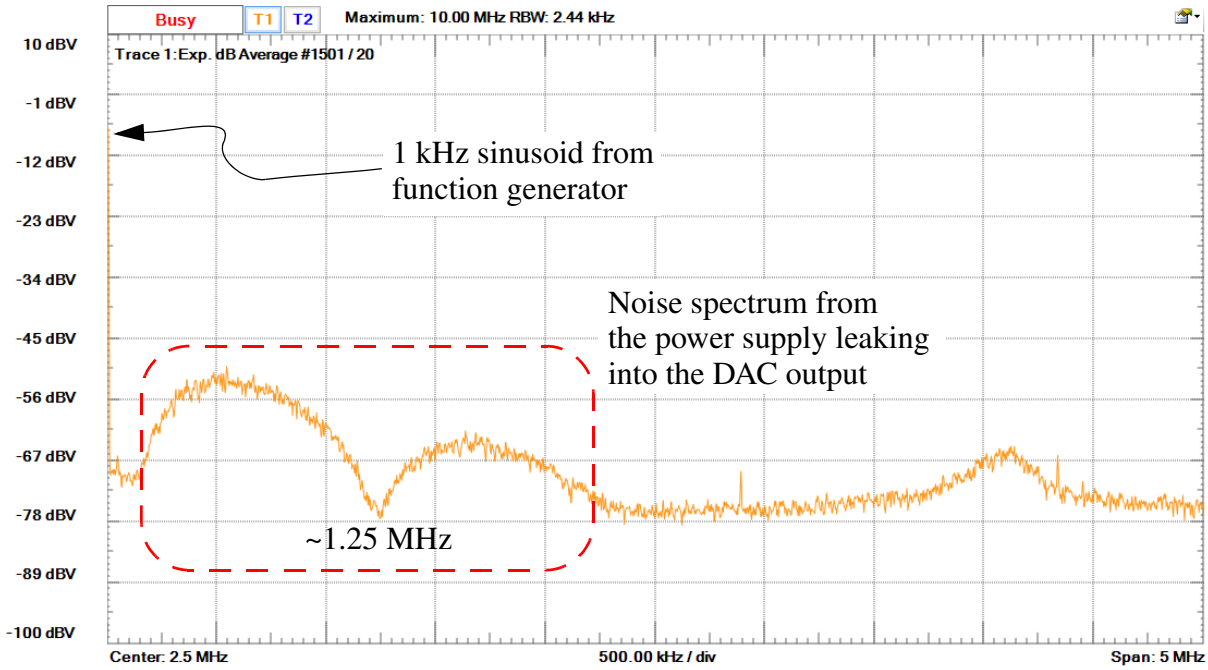
Noise on the Audio Codec Output

The waveform will not be appear clean and sharp as you might expect. The digital-to-analog converter (DAC) contains a small leakage signal from sigma-delta DAC switching waveforms and perhaps from a switching power supply (LDO). The DAC lowpass reconstruction

filter does its job of converting the samples to a continuous-time waveform, however the leakage signal enters the output port through a board design issue. See the DAC output for a 1 kHz input in the following figure, captured using the Analog Discovery[1].



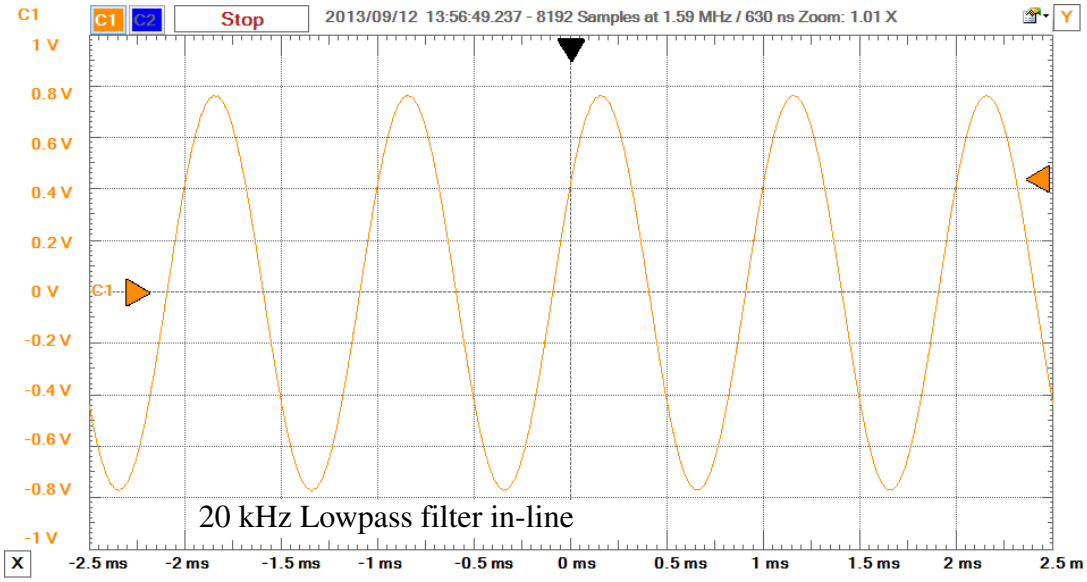
When you listen to this signal through ear buds or the PC speakers on the lab bench you do not hear the noise because it is at a frequency range above your hearing. A spectrum analyzer plot shows the offending portions of the DAC output frequency spectrum.



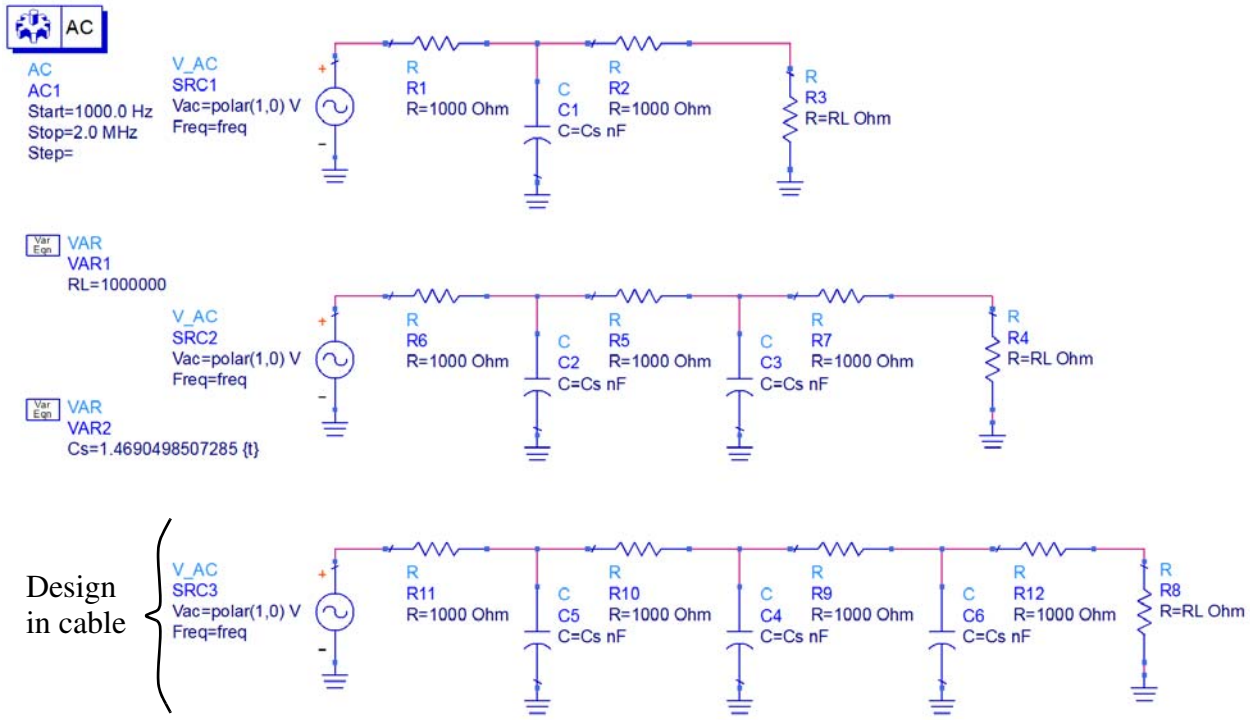
To allow you to view a clean waveform on the scope, a stereo 3.5 mm cable is available in the lab which contains a three section passive lowpass filter (LPF) to remove the high frequency noise

1. http://www.digilentinc.com/Products/Detail.cfm?Prod=ANALOG-DISCOVERY

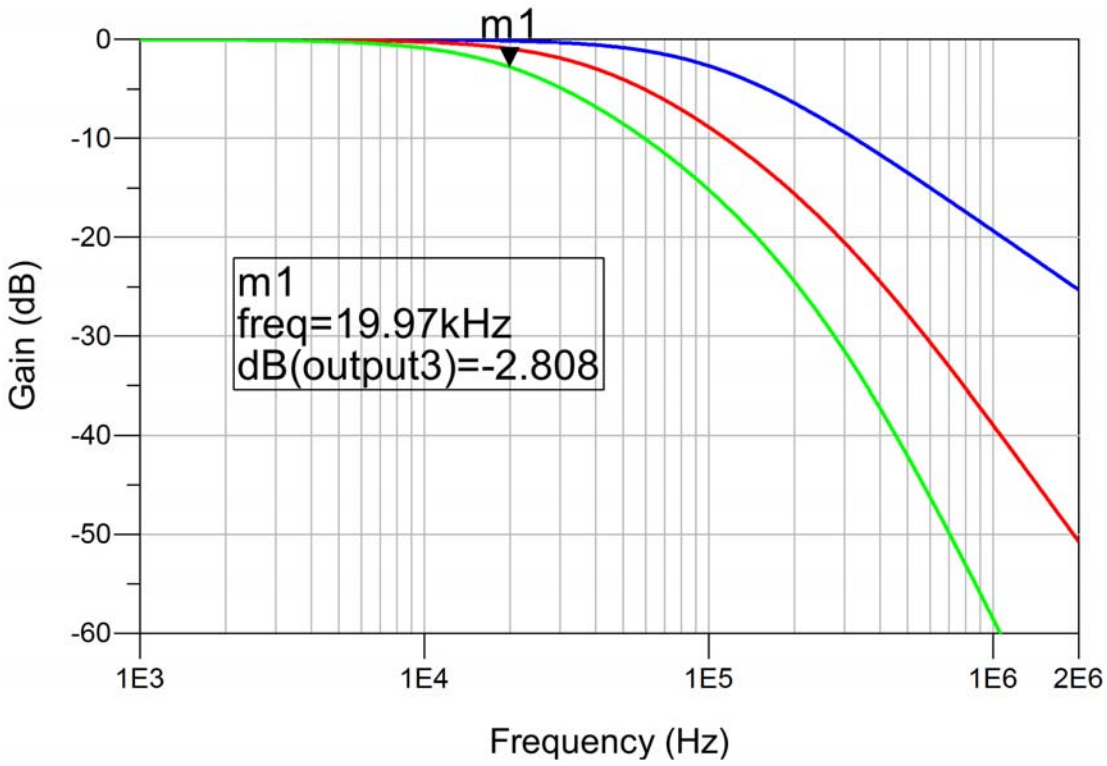
above 20 kHz. I used Agilent ADS to design a LPF that is placed in an audio cable. When this cable is used to connect the codec output to the scope, the trace is very clear as seen in the figure below.



Passive Lowpass Filter Design for DAC Noise Reduction: Consider three passive filter designs build around 1000 ohm series resistors.



- I used the ADS Tuner to find a shunt C value that will produce a 3 dB frequency of about 20 kHz.

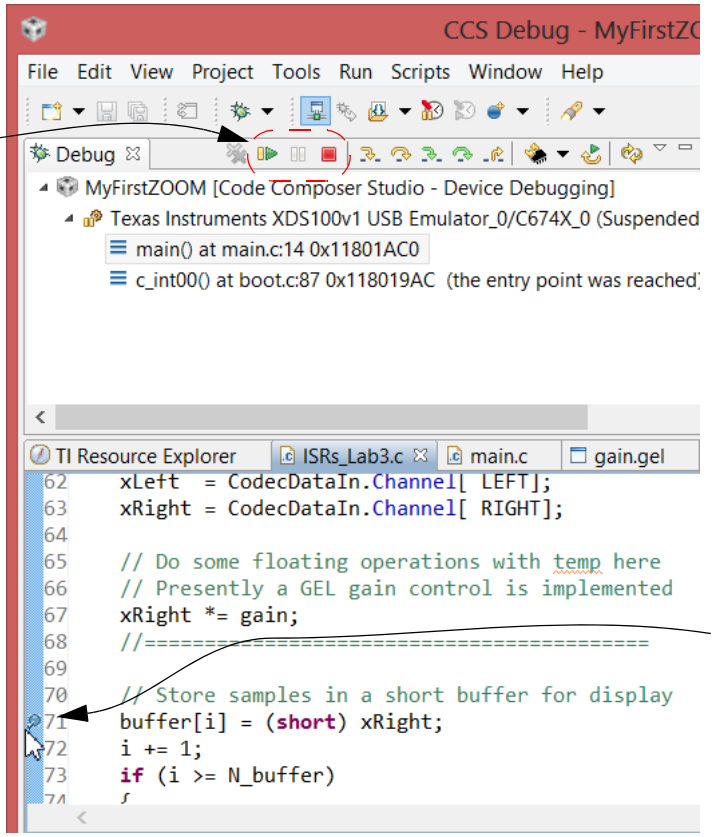

m1
0
-10
-20
-30
-40
-50
-60
Gain (dB)
m1
freq=19.97kHz
dB(output3)=-2.808
1E3
1E4
1E5
1E6
2E6
Frequency (Hz)

Debugging

Suspend the program (`alt-F8`) and set a break-point in the code as shown below:

Run/Resume *Suspend* & *Halt* debugging buttons

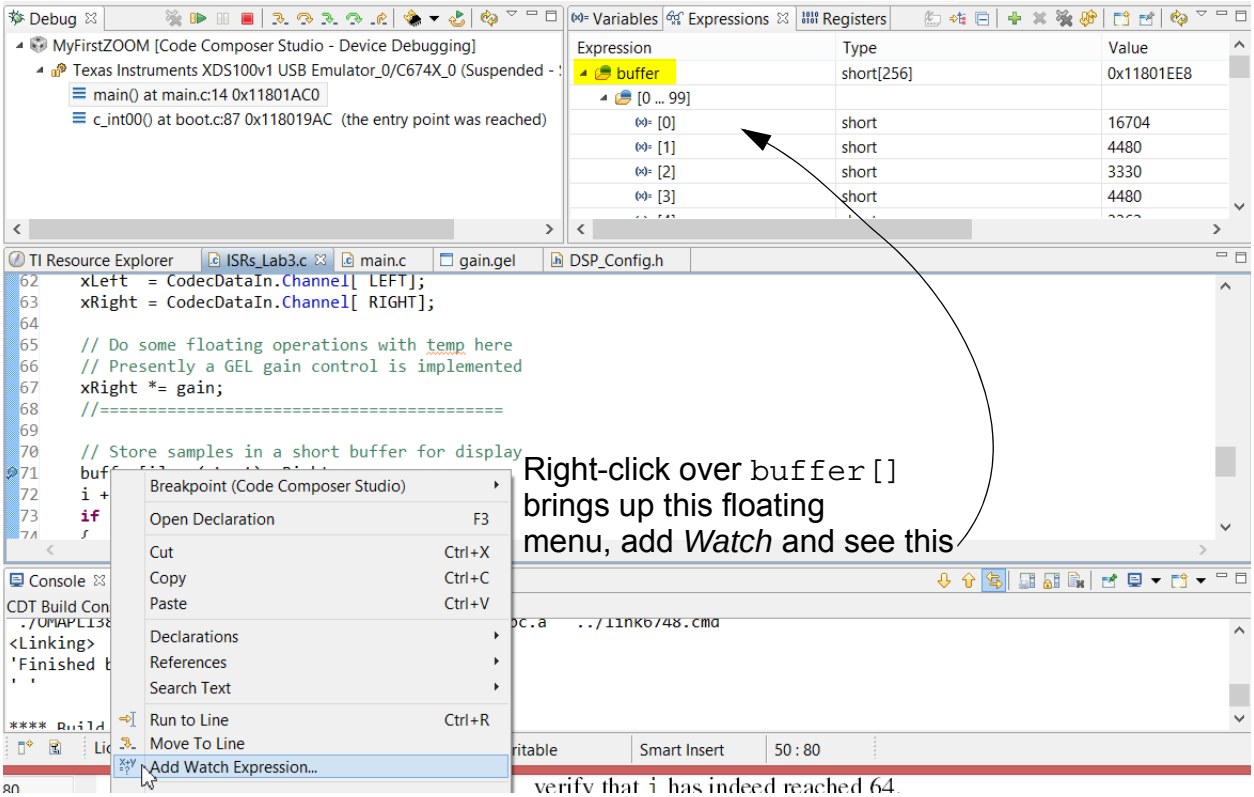
Note: You can *suspend* and modify, then rebuild code without stopping the debugger; you just need to change from the *debug perspective* to the *edit perspective*



Double-click to place breakpoint; click the *resume* button

– Run the program again and see that it will now stop at the break-point you just set.

– Now, right-click over the array variable `buffer`, and select add to watch window.



As you can see in the above screen shot, the watch window opens and you can expand the array variable to see each element of the array `buffer`. Practice setting breakpoints by setting one inside the `if` clause. When the program halts verify that `i` has indeed reached 256.

- When a program is halted you can also *hover* the mouse pointer over any variable and see its value and/or its memory address
- Note also that when you place a variable in the watch window you can also change it by double-clicking it; when you run again, execution continues with the variable change.

Note: This is the eclipse environment for TI programmable devices, which means it also can be used to develop MSP430 code.

GEL Sliders

The general extension language (GEL) capability of CCS allows you to create a GUI slider control that can manipulate variables on code running on the OMAP-L138. In this lab you use it to vary gain of sine output waveform. As an example suppose a global variable in the ISRs `c` file is `gain`:

```
float gain = 1.0; // gain variable controlled by gain.gel
```

At present this variable has been fixed at 1.0, since there has been no outside control enabled to change the value. With a GEL file the value of this variable can be manipulated while the program is running under the control of CCS. A GEL file that performs this function is

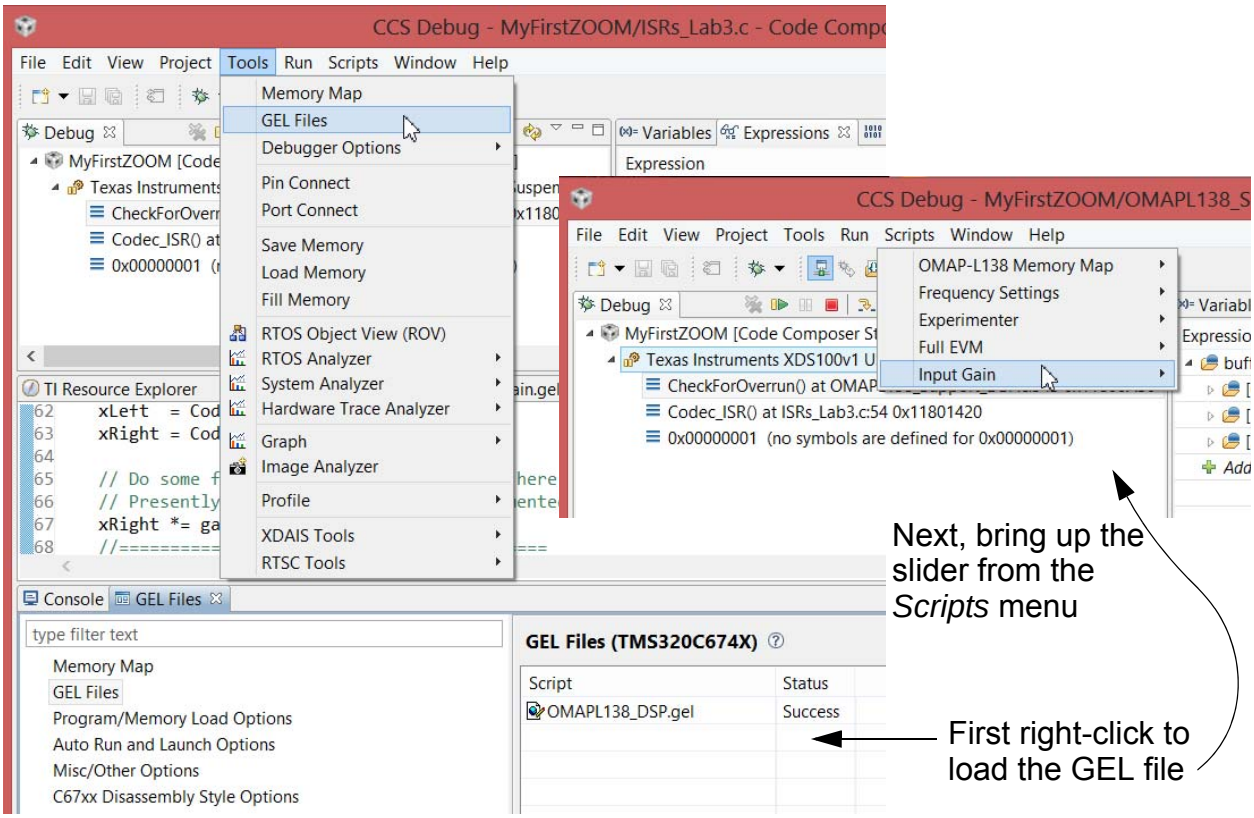
listed below.

```
/*gain.gel GEL slider to vary amplitude of a signal */

menuitem "Input Gain"

slider Gain(0,50,4,1,gain_parameter) /*incr by 4, up to 50*/
{
    /*vary gain_parameter over 0-50 then scale by 10 */
    /* Range of values for gain is thus [0,50]/10 = [0,5.0] */
    gain = (float) gain_parameter/10.0;
}
```

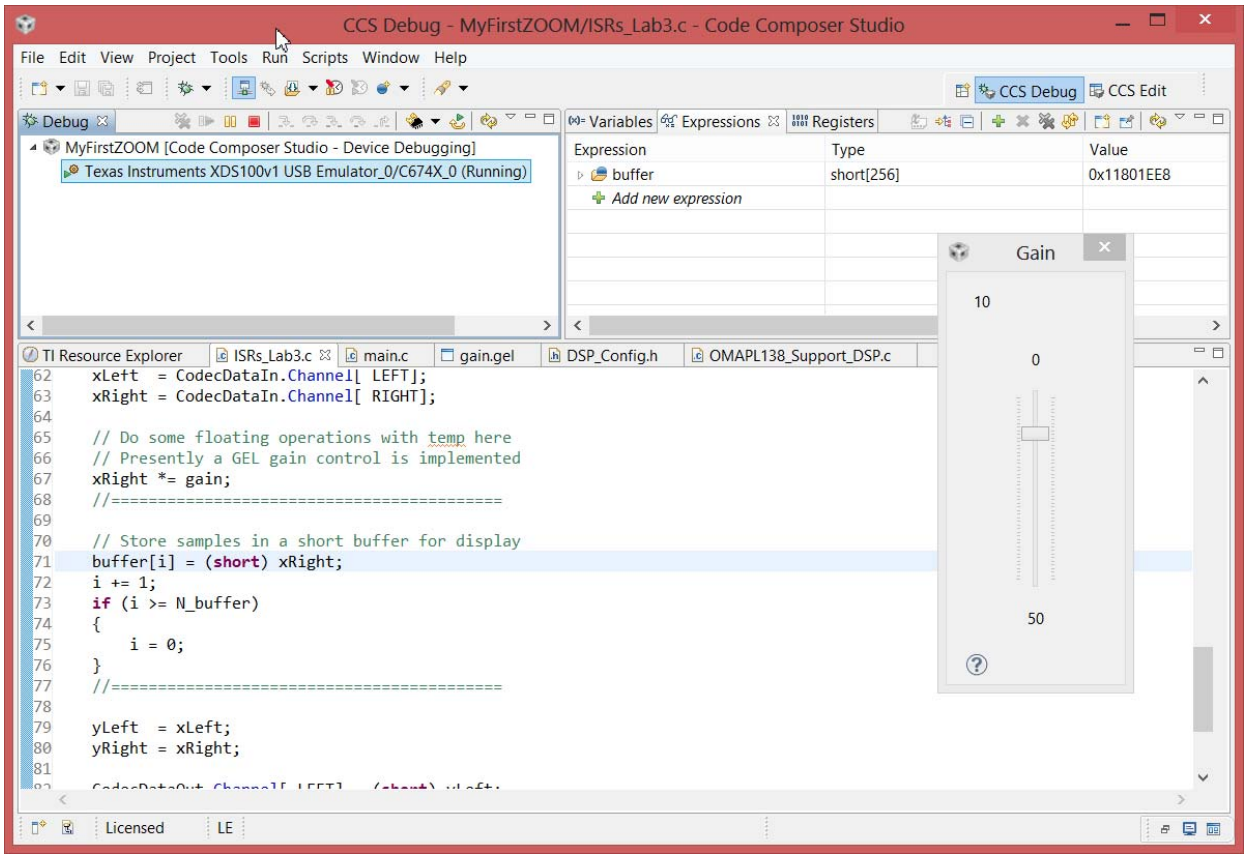
To load the GEL into the current project you first need to include the file in your project. Next start the debugger and from the debugger perspective you will be able access the GEL Files menu item under Tools. then from the File menu:



Once the file is loaded it will show up in the project workspace under the Scripts menu. Clicking this menu reveals the Input Gain item which contains a single slider *gain*.

- In the GEL file listing notice that we created a menu item called "`Input Gain`"
- Under this menu item we have scripted a GUI slider element names Gain, which has range (integer steps) from 0 to 50, page-up/page-down steps of 1, and arrow left/arrow right of 1.
- This slider control then places its output variable, `gain_parameter` into the float

variable `gain`, which is mapped to the matching variable `gain` in ISRs_Lab3.c.

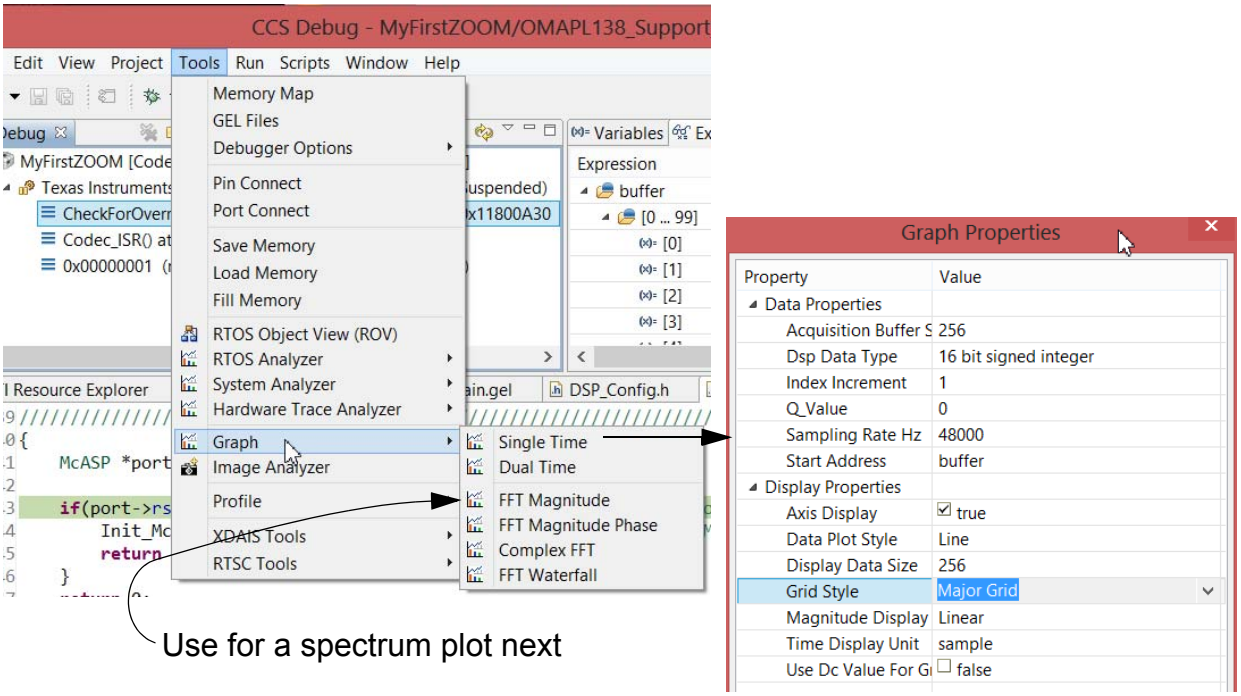


To see the slider in action run the program (move the function generator frequency back to 1 kHz) and notice as you move the slider up and down the signal amplitude will change; best if you use the keyboard (page-up/down or arrow left/right), otherwise the action does not apply until you let go of the slider with the mouse

CCS Graphs

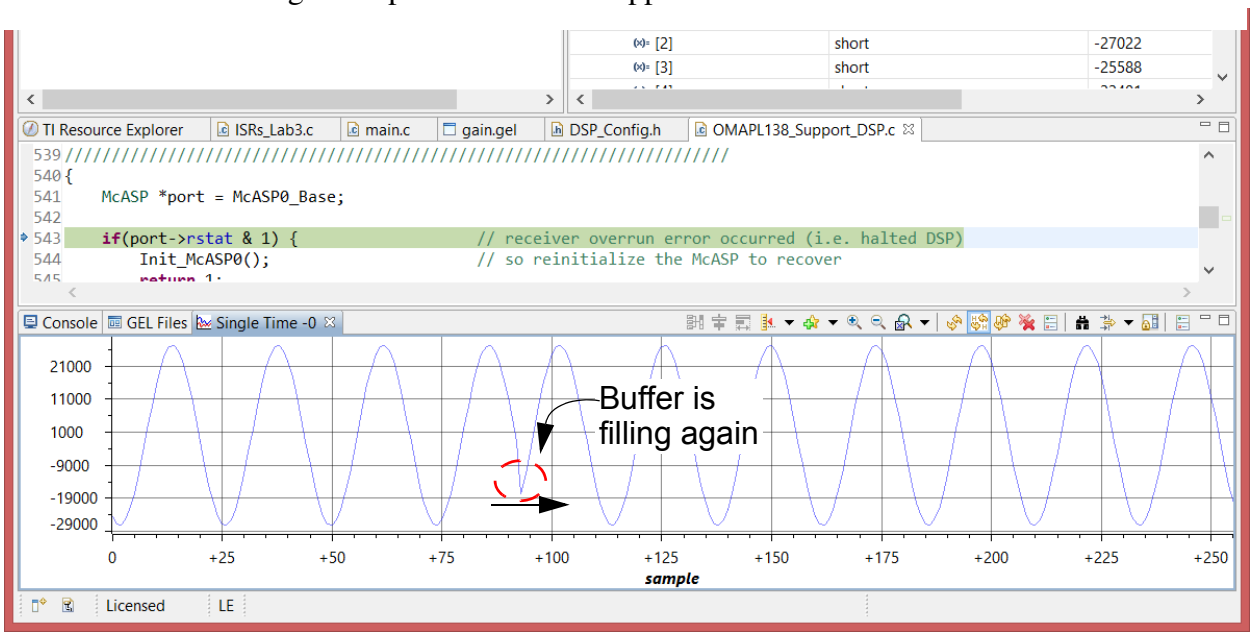
Suppose you have created a buffer that contains a 256 sample record of the most recently processed samples. These samples continuously write over old samples, so the buffer actually contains a discontinuity any time you halt the processor to examine the buffer. The watch wind lets you look at the contents of an array, but with CCS graphical displays, you can plot the contents of a buffer and perform operations such as the FFT, to display the buffer in the frequency domain.

With the processor halted go to the View menu and select Graph:Time/Frequency...

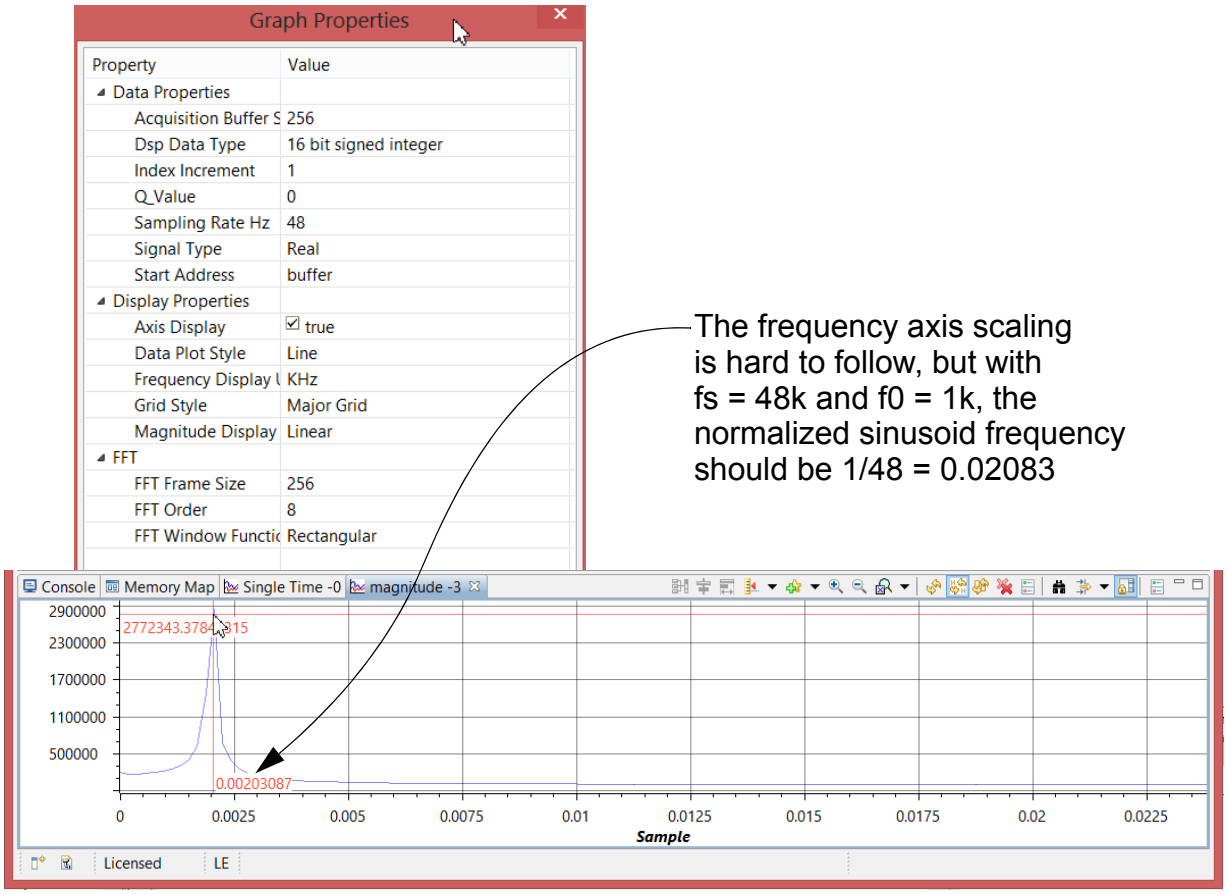


You will then see the Graph Property dialog.

- Configure the start address to the variable name `buffer` (why is this a valid address?).
- Configure the acquisition buffer size and display data size both to 256 (`N_buffer`).
- Configure the DSP data type to `short` (16-bit signed integer).
- Finally configure the sampling rate to 48,000 Hz; this agrees with the setting in the `DSK_Config.h file` (see `#define SamplingRateSetting`).
- OK the dialog and a plot window will appear as shown below:



- At this point you can take measurement from the waveform by dragging the cursor around and reading the amplitude and time locations (time in seconds according to the entered sampling rate value is ignored by CCS for some reason).
- A second plot window can be opened to display the FFT magnitude of the data set
- Under Display Type, the top listed property, choose display FFT magnitude, then OK the dialog and observe the spectrum of a 256-point windowed sinusoid as shown below:



In the screen shot above you can see that the function generator input was at 1 kHz. Change the input frequency to 9 kHz, run (`F5`) and then suspend the processor (`alt-F8`), then verify with graph cursor that the frequency of the sinusoid is 9 kHz in the strange CCS scaling.

FYI: Dr. Wickert's Home Set-Up Using the Analog Discovery