Assignment #3

Due March ~19, 2013

Make Note of the Following:

- This assignment will be written up more like a lab report
- Include the observations made from the scope etc., also include a block diagram of how the test equipment is configured around the DSK

Analog I/O Using the AIC3106 Onboard Audio Codec

1. In this problem you will perform a number of experiments using a modifed version of Morrow's `ISRs.c` as found in the `myFirstProject` folder of Appendix A. We denote this ISR file as `ISRs_Plus.c`. This project is available pre-built on the course Web Site as ZIP package, see below. The file listing

Downloading CCS 5.1 Workspace Projects from Web Site
(can be imported into your workspace)

Sample Code

- Norm Squared ZIP package (C, ASM, SA, and PJTs updated sp2009)
- myFirstProject_Plus (OMAP-L138), VC5505 advanced polling, and C5515 advanced polling examples

← Click here

Directory Listing in: /wickert/ece5655/code/codecs_lab/

File Name	Type	File Size	Last Modified	
C5515_AIC_3204_polling.zip	Compressed (zipped) Folder	41007	3/11/2012 5:05:29 PM	For problem #7
goldwave.zip	Compressed (zipped) Folder	651993	2/24/2004 2:05:17 PM	
loop_stereo.zip	Compressed (zipped) Folder	28482	2/17/2009 11:32:49 AM	
myFirstProjectPlus.zip	Compressed (zipped) Folder	66157	3/6/2012 7:44:10 PM	For problem #1 this package
Thumbs.db	Data Base File	17920	2/17/2010 11:12:57 AM	
tlv320aic3204.pdf	PDF File	2475567	2/16/2010 4:18:40 PM	
vc5505_AIC_3204_polling.zip	Compressed (zipped) Folder	37941	3/11/2012 5:05:40 PM	For problem #7

is:

```
// Welch, Wright, & Morrow,
// Real-time Digital Signal Processing, 2011
// Modified by Mark Wickert February 2012

////////////////////////////////////////////////////////////////////////
// Filename: ISRs.c
//
// Synopsis: Interrupt service routine for codec data transmit/receive
//
////////////////////////////////////////////////////////////////////////

#include "DSP_Config.h"
```

```
// Function Prototypes
long int rand_int(void);

// Data is received as 2 16-bit words (left/right) packed into one
// 32-bit word.  The union allows the data to be accessed as a single
// entity when transferring to and from the serial port, but still be
// able to manipulate the left and right channels independently.

#define LEFT  0
#define RIGHT 1

volatile union {
       Uint32 UINT;
       Int16 Channel[2];
} CodecDataIn, CodecDataOut;


/* add any global variables here */


interrupt void Codec_ISR()
////////////////////////////////////////////////////////////////////////
// Purpose:   Codec interface interrupt service routine
//
// Input:     None
//
// Returns:   Nothing
//
// Calls:     CheckForOverrun, ReadCodecData, WriteCodecData
//
// Notes:     None
////////////////////////////////////////////////////////////////////////
{
       /* add any local variables here */
       WriteDigitalOutputs(1); // Write to GPIO J15, pin 6; begin ISR timing pulse
       float xLeft, xRight, yLeft, yRight;
       short k;


       if(CheckForOverrun()) // overrun error occurred (i.e. halted DSP)
              return;        // so serial port is reset to recover

       CodecDataIn.UINT = ReadCodecData();// get input data samples

       /* add your code starting here */
       // this example simply copies sample data from in to out
       xLeft  = CodecDataIn.Channel[ LEFT];
       xRight = CodecDataIn.Channel[ RIGHT];
       //*********** Input Noise Testing ***************
       //Generate left and right noise samples
       //xLeft = ((short)rand_int())>>2;
```

```
        //xRight = ((short)rand_int())>>2;
        //*********************************************

        yLeft  = xLeft;
        yRight = xRight;

        CodecDataOut.Channel[ LEFT] = yLeft;
        CodecDataOut.Channel[RIGHT] = yRight;

        for(k=0; k<100; k++)
        {
               // Idle loop to load processor inside ISR
        }
        /* end your code here */

        WriteCodecData(CodecDataOut.UINT);// send output data to  port
        WriteDigitalOutputs(0); // Write to GPIO J15, pin 6; end ISR timing pulse
}

//White noise generator for filter noise testing
long int rand_int(void)
{
        static long int a = 100001;

        a = (a*125) % 2796203;
        return a;
}
```

The sampling rate can be changed via the line of code

```
// 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
#endif
```

For this part of the exercise use `Fs = 24 ksps`.

a.) Set the AIC3106 codec up to pass the input samples directly to the output. Input a sinusoid from an Agilent 33250A function generator. Determine the maximum input level of the input A/D and the maximum output level of the D/A. This will likely require that you perform some scaling experiments as you pass samples read from the A/D directly over the D/A. In `ISRs_Plus.c` you find inside the interrupt routine `Codec_ISR()` the two lines

```
yLeft  = xLeft;
yRight = xRight;
```

The left and right input samples from the A/D that have been placed in the `float` work-

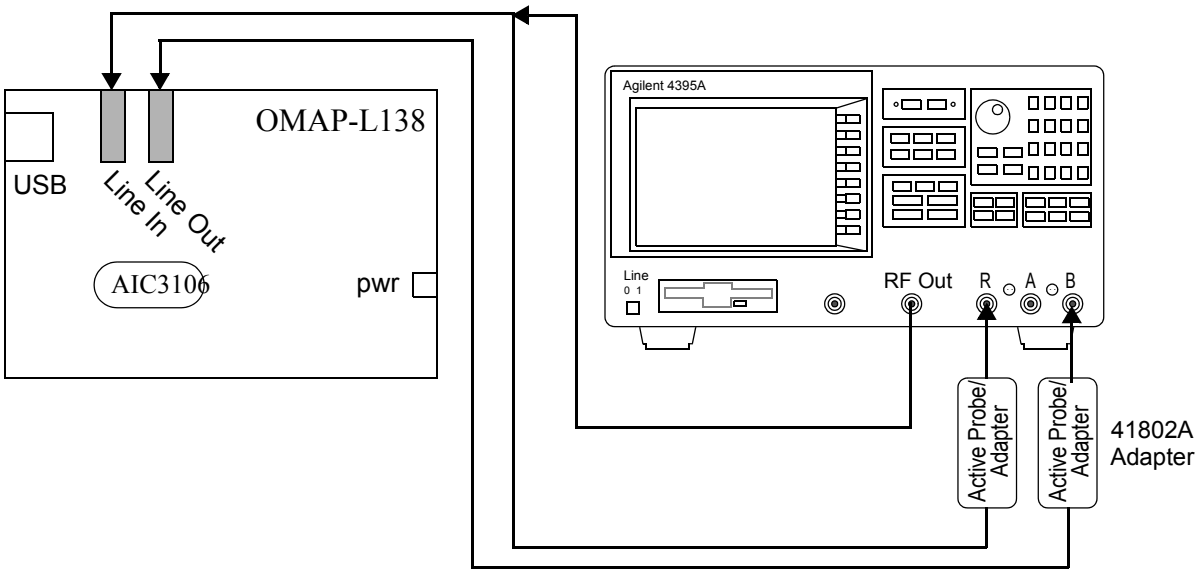

Figure 1: Agilent 4395A vector network analyzer interface to the AIC3106.

ing variables `xLeft` and `xRight`, are directly copied into the output left and right float working variables `yLeft` and `yRight`, to ultimately be sent to the D/A. To understand amplitude limiting issues, some form of scaling will also be needed. Do not alter the default codec gain settings for this experiment. Note these settings are hidden away in the codec library, but you do have access to it. The idea is to determine the exact dynamic range values for the default settings for future reference. I realize that these values can change if the codec attenuator settings are altered. Using a buffer to store `xLeft` or `xRight` samples will allow you to view signals samples inside the CCS environment, e.g. a plot window or variable listing

Scope plots can be obtained by capturing the data directly into MATLAB using the HP-IB data bus. Helper files can be found at `http://eceweb.uccs.edu/wickert/ece4670/`.

b.) Using the same configuration as in part (a), now apply a squarewave at the input and measure the rising edge time delay in going from the DSK analog input to the analog output. Be sure to use a long enough period so that additional cycles do not creep in should the time delay be longer than you anticipate.

c.) Using the vector network analyzer measure the frequency response magnitude in dB as the analog output over the input (say using analyzer ports B/R). The frequency range you sweep over should be consistent with the sampling frequency. Use what you learned in part (a) to properly set the input level to the A/D. A good safe level for the input is 80-90% of full-scale.

Plots from the 4395A can be obtained in several ways. One approach is to bring the captured data directly into MATLAB using the HP-IB data bus. Helper files can be found at `http://eceweb.uccs.edu/wickert/ece4670/`. A second approach is to use the instruments capability to save an Excel data file or a tiff image file to the 4395A

floppy drive. Thirdly there is a 4395A PC screen capture utility described on the ECE4670 Web Site.

Comment on the locations of the critical frequencies, e.g., lower and upper band edges as appropriate, with respect to known A/D antialiasing filter characteristics in combination with the known reconstruction filter characteristics. Try to verify the passband ripple that we know exits in both the A/D and D/A signal paths.

An approximate model of the system can be created in MATLAB using filter design functions. A good starting point is to use a high-order linear phase FIR filter designed using `fir1()` or better still `firpm()`. You can obtain basic details about the filters from the AIC3106 data sheet and also the course notes. You cannot create an exact model since there is not enough information given. You should still able to create an approximate frequency response magnitude of the cascade connection of the A/D – D/A shaping filters with MATLAB's `freqz()` operating on the filter coefficients.

2. In this problem you will again work with the AIC3106 codec, but this time you will use a uniform white noise generator as a wideband signal source. The output of the D/A will then be spectrum analyzed using MATLAB via postprocessing of a `.wav` audio file. You may need to scale the output of the uniform random number generator before passing it to the codec output function. Inside the `Codec_ISR()` you will find:

```
//xLeft  = CodecDataIn.Channel[ LEFT];
//xRight = CodecDataIn.Channel[ RIGHT];
//********** Input Noise Testing ***************
//Generate left and right noise samples
xLeft = ((short)rand_int())>>2;
xRight = ((short)rand_int())>>2;
//*********************************************
```

To ignore the A/D samples and use the noise generator samples instead, comment and uncomment lines as shown above. When this program is run the noise generator is output to the D/A which includes the a lowpass reconstruction filter (part of the delta-sigma D/A). In the discrete-time domain we know that the power spectrum will be flat (white). Since here the noise generator output is connected directly to the D/A, the PSD of the output should have shape corresponding to the magnitude squared of the D/A lowpass filter frequency response, in cascade with perhaps other analog circuit poles and zeros.

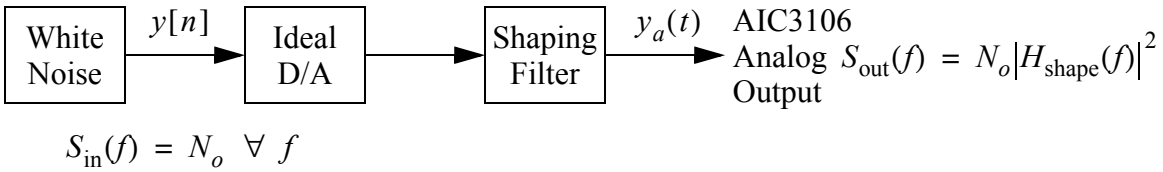

Figure 2: Model for the noise spectrum produced at the codec output.

Rather than using the analog spectrum analyzer as in Problem 1, this time we will use the PC SoundBlaster digital audio system to capture a finite record of D/A output. The software capture tool that is useful here is the shareware program *GoldWave* (`goldwave.zip` on Web site). A 30 second capture at 44.1 kHz seems to work well. Since the sampling rate is

only 24 ksps.

Once a record is captured in GoldWave it can be saved as a `.wav` file. The `.wav` file can then be loaded into MATLAB using the `wavread()` function. MATLAB 7 has the ability to import data files directly from the *Import Data...* option under the *file* menu. Note, GoldWave can directly display waveforms and their corresponding spectra, but higher quality spectral analysis can be performed using the MATLAB signal processing toolbox. The spectral analysis function to be used in MATLAB is `simpleSA()`, which implements Welch's method of averaged periodograms. Suppose that the `.wav` file is saved as `test1.wav`, then a plot of

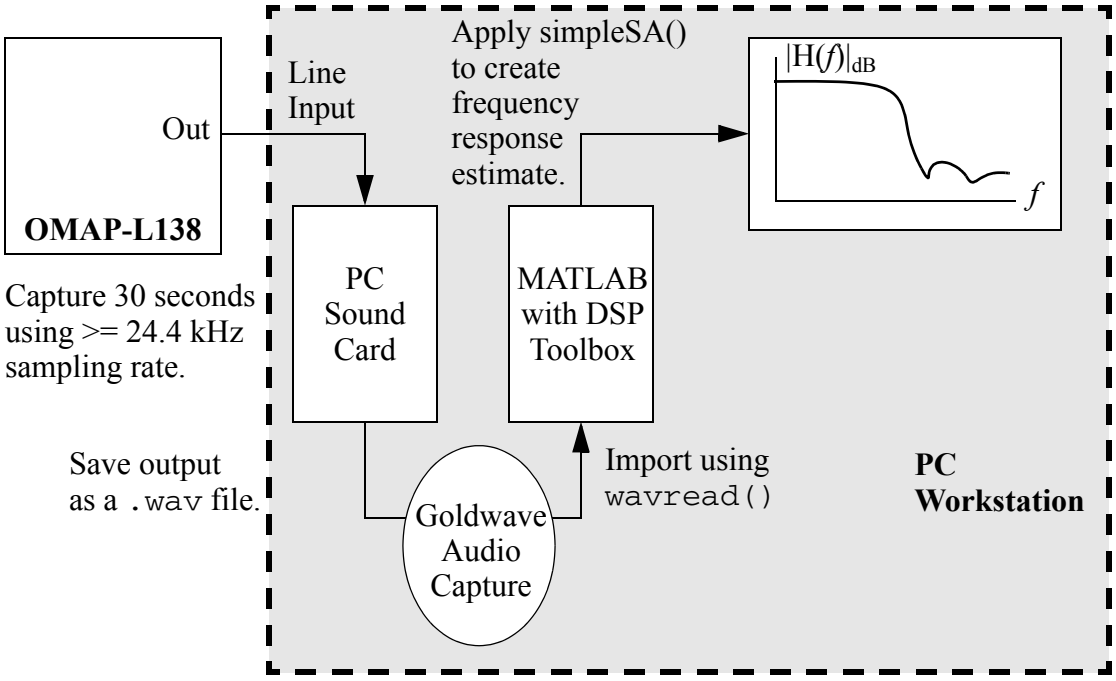

Figure 3: Waveform capture using the PC sound card.

the frequency response would be created as follows:

```
» [x,Fs] = wavread('test1.wav'); %Get x and sampling rate
» % Use 2048 pt. FFT and plot with proper fs value
» simpleSA(detrend(x(:,2)),2048,44100); %detrend() removes DC offsets
» % Rescale plot as needed and overlay other plots if desired.
```

One condition to watch out for is overloading of the PC sound card line input. Goldwave has level indicators to help with this however. Use the sound card line input found on the back of the PC Workstation. The software mixer application has a separate set of mixer gain settings just for recording. You will need to adjust the Line In volume and watch actual input levels on Goldwave.

In summary, using the procedure described above obtain a MATLAB plot of the frequency response magnitude in dB, of the AIC3106 D/A output channel.

3. Using the sound card capture technique of Problem 2 modify the code to again pass signal samples from input to output. Place a 1 kHz sinusoid at the input, at about 80-90% of full

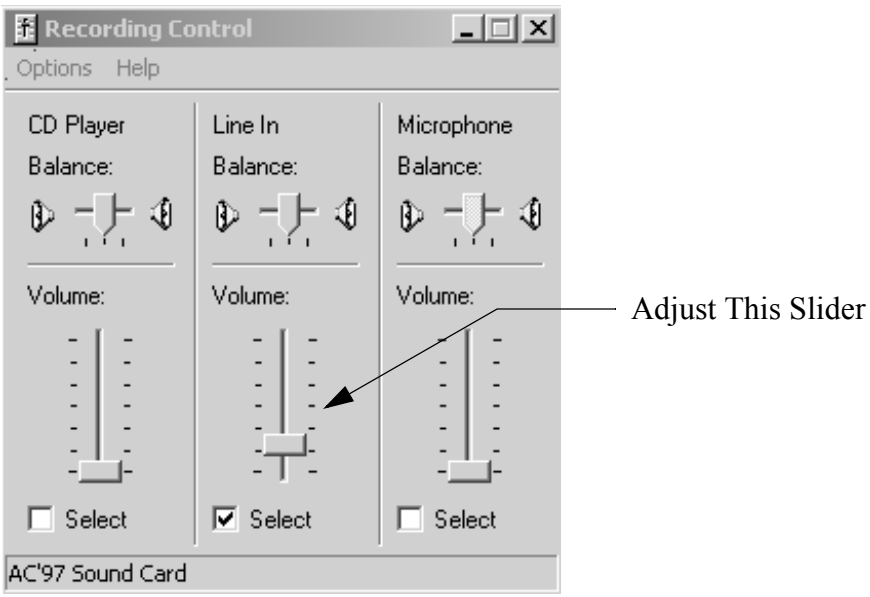

Figure 4: Controlling the input level to the PC sound card.

scale. Capture the sinusoid at the output as a 30 second wave file. Perform spectral analysis as in the noise case of Problem 2. Comment on the spectral purity and the presence of any distortion terms. To understand the PC sound card ambient noise environment better, make a recording with no input to the sound card to better characterize its limitations on dynamic range and spectral purity. This reference plot should be made last, after you have properly set the line-in level to avoid sound card input overload.

4. Using the OMAP-L138 with AIC3106, and sampling rate of 48 kHz, generate a sinusoidal oscillator that uses *GEL file*[1] to control the frequency of oscillation from 100 Hz to 20 kHz in 100 Hz steps. The frequency determining variable will actually be a float variable, but the GEL file itself will set the absolute frequency range and step size. The approach to generating the variable frequency oscillator will be via the equation

$$s[n] = s(nT) = \sin(2\pi f_0 nT) = \sin\left(2\pi\frac{f_0}{f_s}n\right) = \sin(n\Delta)$$

where T is the sample spacing, $f_s = 1/T$, and f_0 is the analog oscillation frequency. The angular change between samples is denoted by $\Delta = 2\pi(f_0/f_s)$. If we let $\theta[n] = n\Delta$, then we can write that

$$\theta[n+1] = (n+1)\Delta = n\Delta + \Delta = \theta[n] + \Delta$$

For more information on this technique see Chapter 5 of the text. In C code we can build a float version of this using the library `<math.h>` for the trig function (this will not be particularly efficient in terms of CPU cycles, but will be interesting to investigate).

```
. . .
#define pi 3.141592653589
//set sampling rate to 48 ksps
```

1. See the appendix at the end of this assignment doc.

```
float s = 0.0; //initial value for the signal s
float f0 = 1000.0; //initial value for oscillation frequency
float delta =  0.1308996938995747;  //2*pi*1000/48000.
float twopi = 2.0*pi;
float angle = 0.;
. . .
{
. . .
//inside interrupt service routine (use sinf() for float vs sin() double)
s = 15000.0*sinf(angle); //scale to half of full scale for short values
angle += delta; //increment angle by proper delta value
if( angle >= twopi) angle -= twopi; // prevent float overflow by wrapping the phase
. . .
CodecDataOut.Channel[LEFT] = (short) s; // cast float to short
}
. . .
```

The A/D inputs will not be used for anything in this example. The output only needs to appear on one channel, left or right.

5. This problem explores real-time generation of a *dual tone multifrequency* (DTMF) tone. The row and column frequencies used to generate the telephone 'touch tones' are given in Table 1.

Table 1: DTMF frequency pairs

Frequencies	1209 Hz	1336 Hz	1477 Hz
697 Hz	1	2	3
770 Hz	4	5	6
852 Hz	7	8	9
941 Hz	*	0	#

Using the frequency generation technique of Problem 4, a look-up table, or some other method that works, to write a C program to serve as a DTMF signal generator. Use GEL files to control modes of operation. The sampling rate will be $f_s = 8000$ Hz. Additional requirements are as follows:

- The program will have a test mode, via two GEL sliders, so that a given row column frequency pair can be generated for an indefinite duration. You might want to try using GUI composer for this instead.
- When not in the test mode, the program will dial a seven digit phone number, holding each digit (frequency pair) for D ms. The value D will be set via a GEL slider too, over the range of 10ms to 1s. The phone number is to be held in an array or as a character string, your choice. The bottom line is it needs to be changeable during the demo.
- When dialing a phone number, the sequence will be repeated every 30s.

a.) Characterize the performance of the generator by capturing on the spectrum analyzer, in continuous tone pair test mode (bullet #1), the tone spectrum at the code output.

b.) Configuration the program to dial my office phone number, 255-3500, and capture via the PC sound recorder, at least one cycle of dialing. Import the audio capture into MATLAB and display the results using `specgram(x,NFFT,fsamp)`, which is MATLAB's spectrogram function. From the spectrogram identify that the proper frequency pair sequence is being generated.

6. Using the OMAP-L138 with AIC3106, and sampling rate of 48 kHz, write sampling loop code that performs the following:

$$y[n] = (-1)^n x[n]$$

Your code need only implement this operation for one of the two I/O channels. Explain using DSP math what is happening in the frequency domain. Explain what you hear when you sweep a sinusoid across the input and listen to the result at the output. I will expect a demo of this in the lab.

VC5505 eZdsp™ or TMS320C5515 eZdsp™ Codec Basics

7. Sample CCS 5.1 projects are available for both boards on the course Web Site. The codec routines are identical, since both boards use the TI AIC3204 chip.

Directory Listing in: /wickert/ece5655/code/codecs_lab/

File Name	Type	File Size	Last Modified	
C5515_AIC_3204_polling.zip	Compressed (zipped) Folder	41007	3/11/2012 5:05:29 PM	For problem #7
goldwave.zip	Compressed (zipped) Folder	651993	2/24/2004 2:05:17 PM	
loop_stereo.zip	Compressed (zipped) Folder	28482	2/17/2009 11:32:49 AM	
myFirstProjectPlus.zip	Compressed (zipped) Folder	66157	3/6/2012 7:44:10 PM	
Thumbs.db	Data Base File	17920	2/17/2010 11:12:57 AM	
tlv320aic3204.pdf	PDF File	2475567	2/16/2010 4:18:40 PM	
vc5505_AIC_3204_polling.zip	Compressed (zipped) Folder	37941	3/11/2012 5:05:40 PM	For problem #7

The AIC3204 sample code currently sets the codec up for a sampling rate of 48 kHz. In this problem we desire programmable sampling rates, say by changing a `#define` in a header file. Desired sampling rates include 8, 44.1, and 48 KHz. Other values would be a bonus. The results from the Spring 2010 effort along these lines is what is available on the Web Site. The Spring 2011 students made no progress in this area. The challenge here is to create a true interrupt scheme for sample-by-sample processing. The present code is as shown below:

```
void main(void)
{
    //Define working variables
    short i = 0;
    short l_chan, r_chan;
    short buffer[128];

    // Initialize Polling
    comm_poll();
      while(1)
```

```
        {
                // Get sample using inputs
                //input_sample(&l_chan, &r_chan);

                // Get random sample
                r_chan = ((short) rand_int())>>3;

            /* Fill buffer */
            buffer[i] = r_chan;
            i += 1;
            if (i == 128) i = 0;

                /* Write Digital audio input */
                output_sample(l_chan, r_chan);
        }
    }
```

The polling nature is evidenced when you look at the functions `input_sample()` and `output_sample()` found in `aic3204.c`.

```
void input_sample(short *l_chan, short *r_chan)
{
       /* Read Digital audio input */
         while((RcvR & I2S0_IR) == 0);  // Wait for receive interrupt to be pending
       *l_chan = I2S0_W0_MSW_R;
       *r_chan = I2S0_W1_MSW_R;
}
void output_sample(short l_chan, short r_chan)
{
       while((XmitR & I2S0_IR) == 0); // Wait for transmit interrupt to be pending
        I2S0_W0_MSW_W = l_chan;  // 16 bit left channel transmit audio data
        I2S0_W1_MSW_W = r_chan;  // 16 bit right channel transmit audio data
    }
```

Appendix: GEL (general purpose evaluation language) Files

```
/*freq.gel GEL slider to vary the frequency of a sinewave*/
/*myFirstProjectPlus*/

menuitem "Frequency"
/* 100 = min, 20000 = max, 10 = arrow key up/down step size, 100 = page up/down step size */
slider Frequency(100,20000,10,100,freq_parameter) /*incr by 100, up to 100*/
{
     delta = freq_parameter*0.0001308996938995747;       /*vary frequency of sine*/
}
```

To load a GEL file see the CCS help and search for *load GEL.*