88/100

Timothy Sjoquist (0822403)

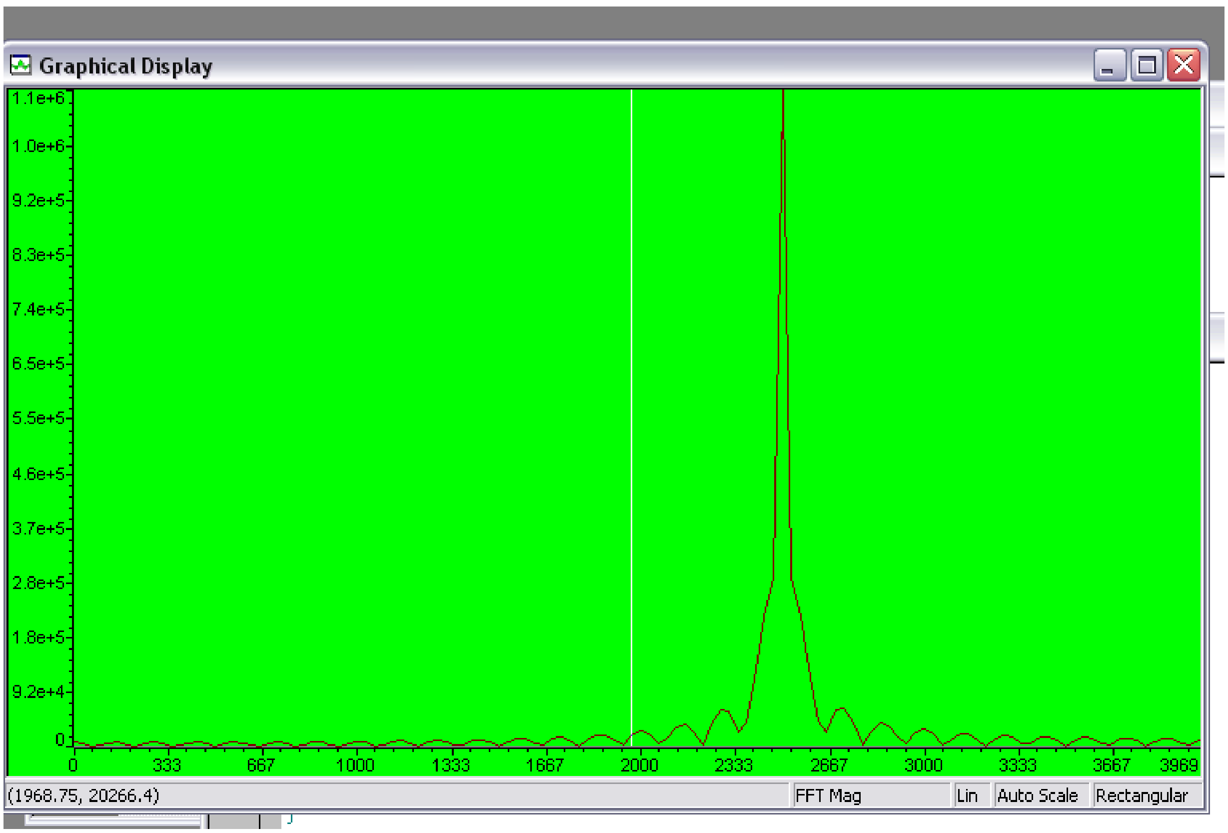
Yen-Ting Chen (1063219)

Lab 2

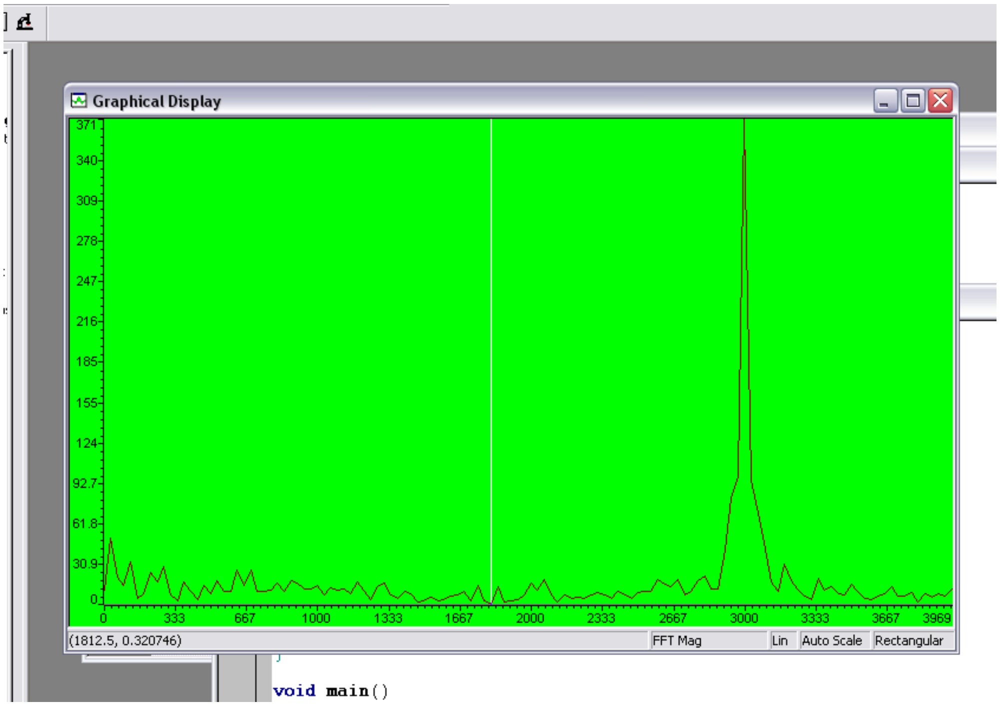
9/15

Problem 1

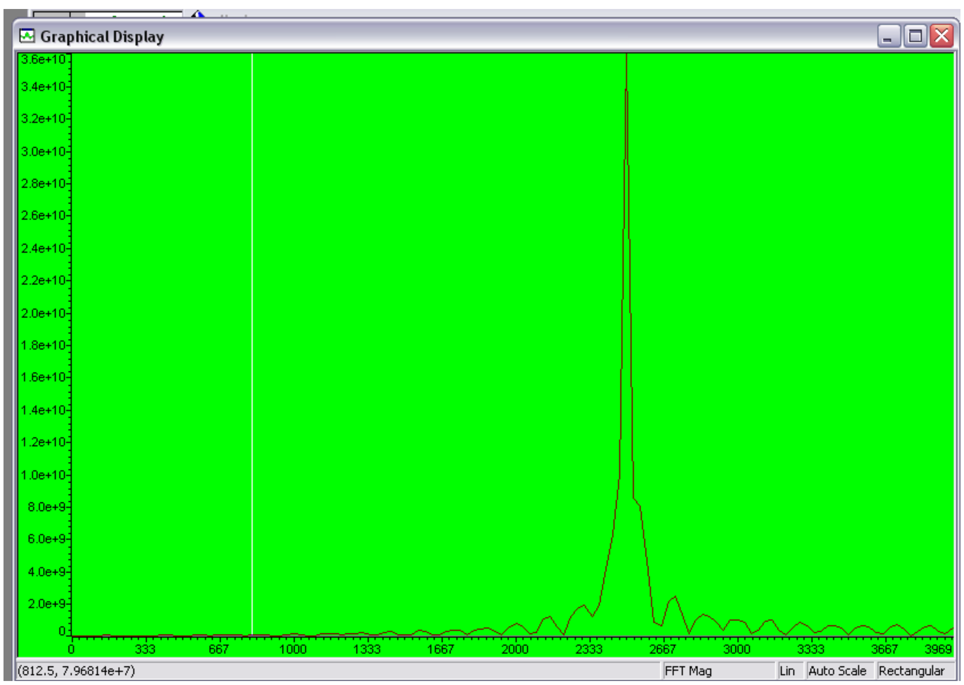
For problem one, we created two sinusoids, one with a frequency of 2500Hz and the other with a frequency of 5000Hz. We looked at their frequency response which are shown below. The response is normal for the 2500Hz signal, but because of aliasing the second signal appears to have a frequency of 3000Hz. This is so because it is “folding over” meaning as we would increase the frequency of the signal past 4000Hz the frequency response looks like “actual frequency-4000Hz”. Since we put in 5000Hz, the frequency response appears to be 3000Hz. From this part of the lab we learned that aliasing is something that needs to be dealt with when dealing with analog to digital transformations.



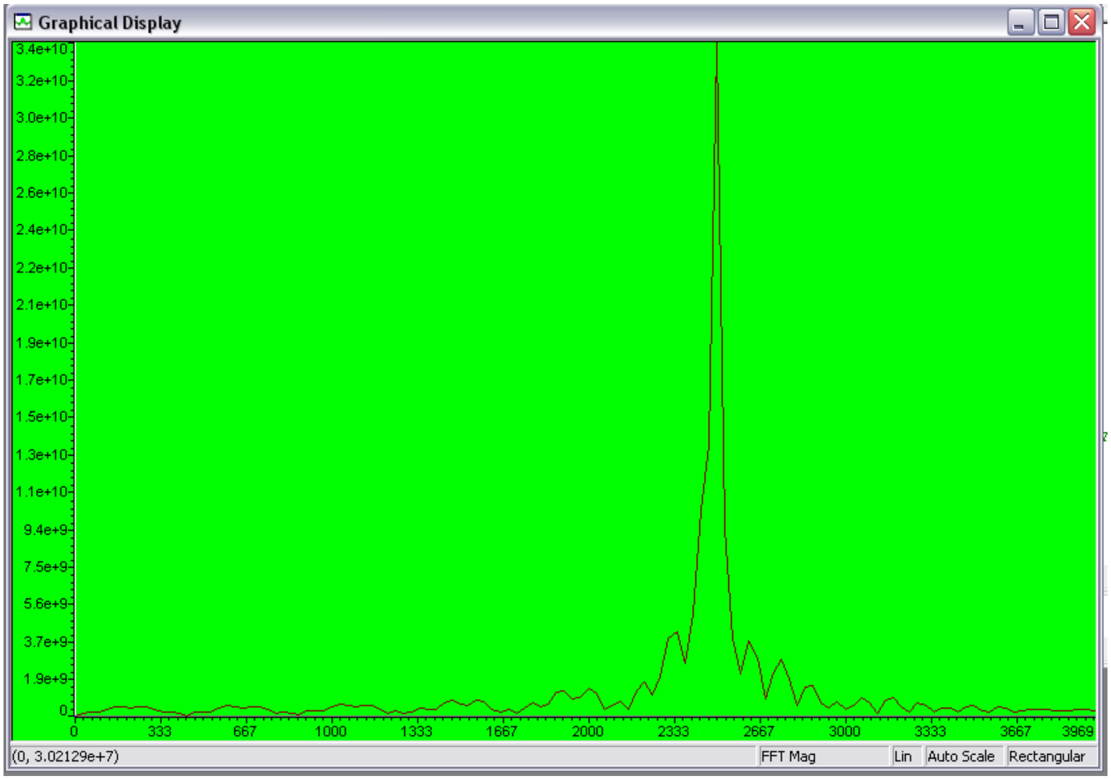
Frequency response of 2500 Hz, first part



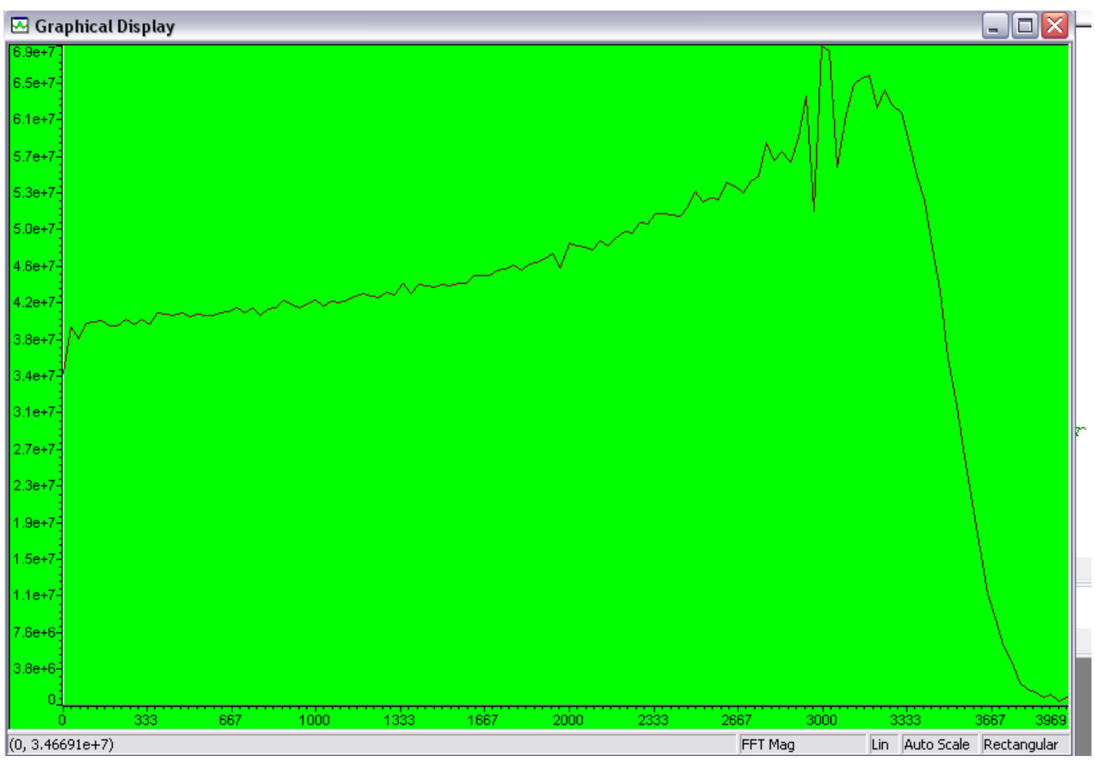
Frequency response of 5000 Hz, first part



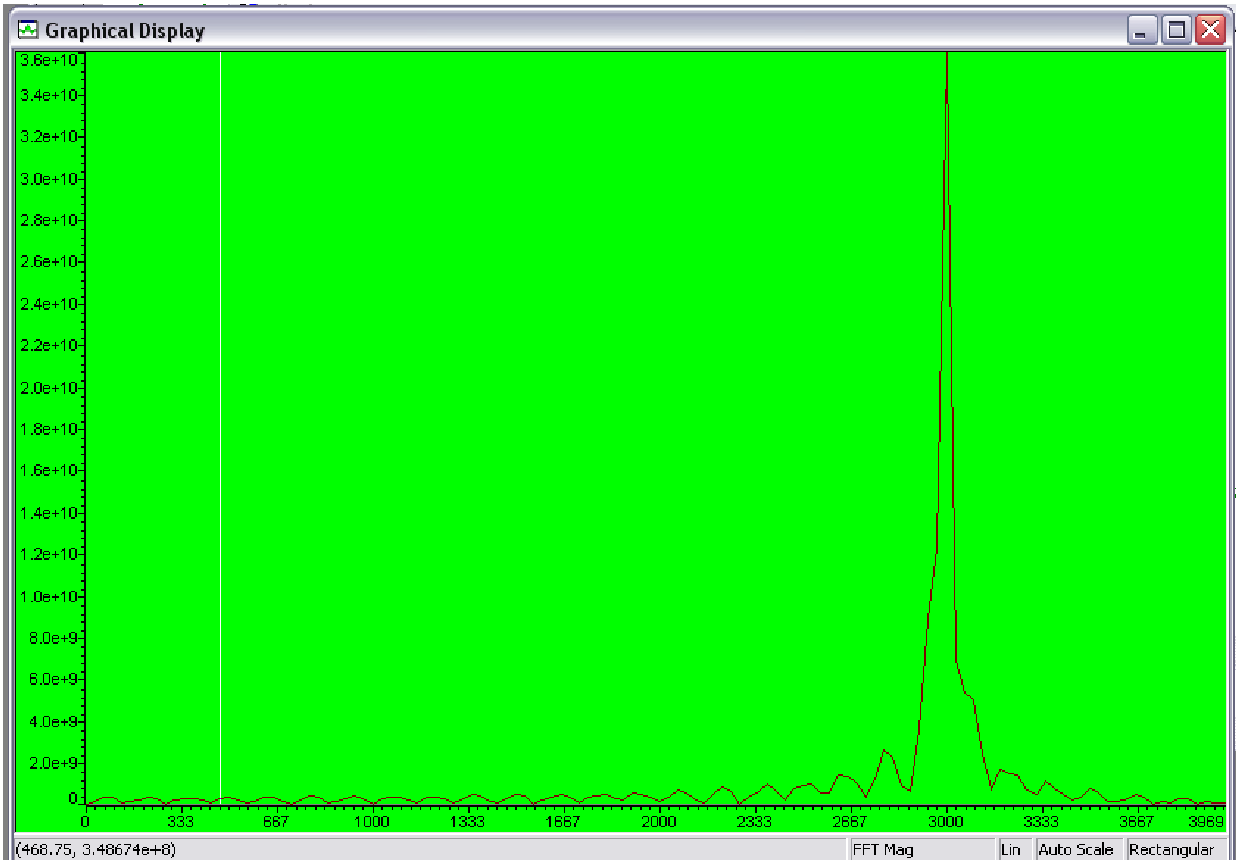
Frequency response of 2500 Hz with no anti-aliasing filter



Frequency response of 2500 Hz with anti-aliasing filter



Frequency response of 5000 Hz with no anti-aliasing filter



Frequency response of 5000 Hz with anti-aliasing filter

15/15

Problem 2

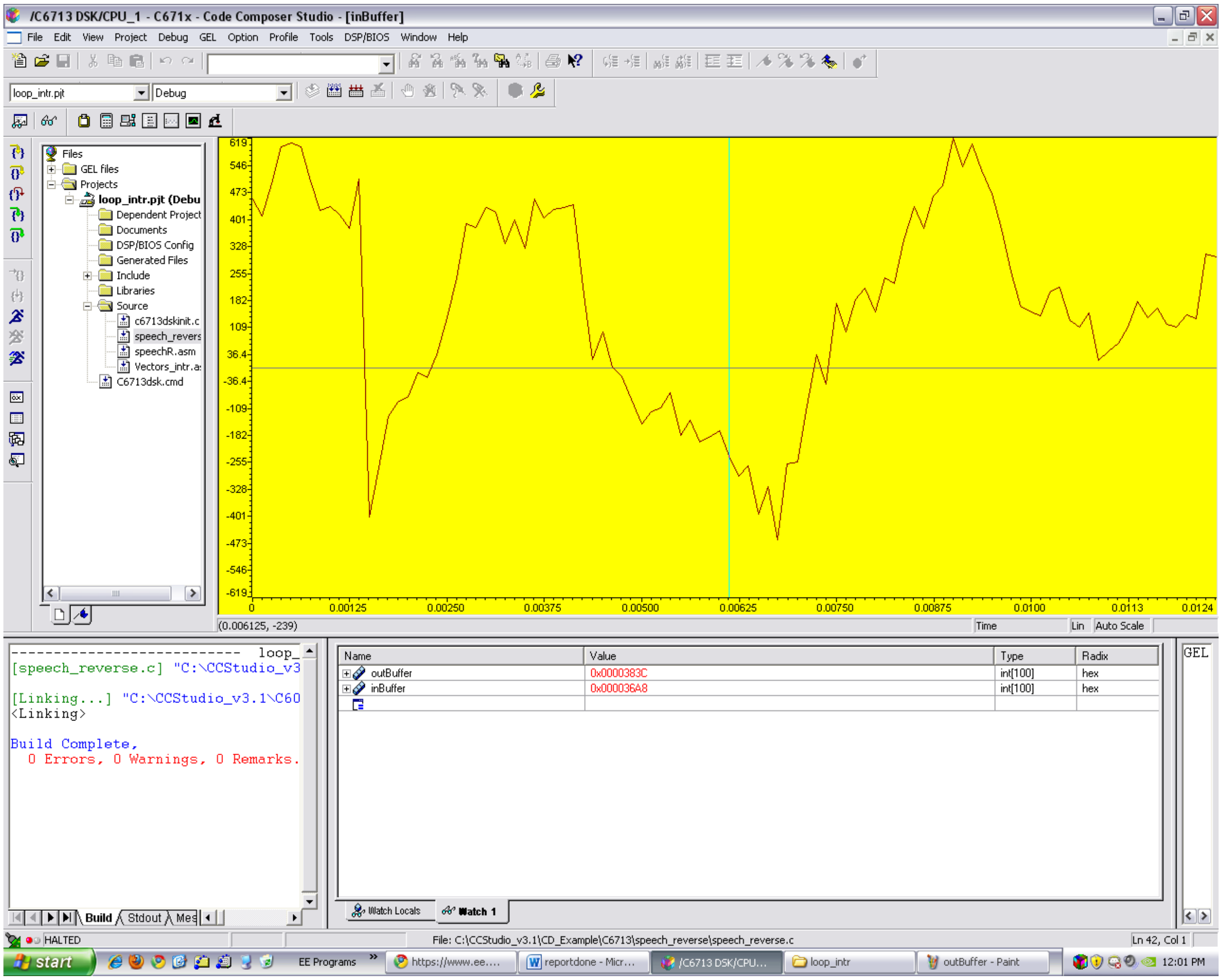
For problem 2 we were to have a line input playing music from the PC sampled at 32KHz and add delay functionality to it. We needed to create sliders to control both gain and the amount of delay for both the left and right channel. To accomplish this we needed to use union(Uint32 uint; short channel[2]), which would allow us to get different data for both the left and right channels. Then we got the data for either the left or the right, and added it to the correct buffer, corresponding to the correct side which allows us to delay the signal. We then outputted the most recent input data added to the data from the buffer to create the delay sound. We could change the amount of delay by changing the bufferlength, a larger buffer means longer delay. To change the gain we simply create a variable gain which is multiplied by the delay factor to change how loud the delay will be. No graphs or tables were required. From this portion of the lab, we learned how to add delay to an input signal for stereo output.

17/20

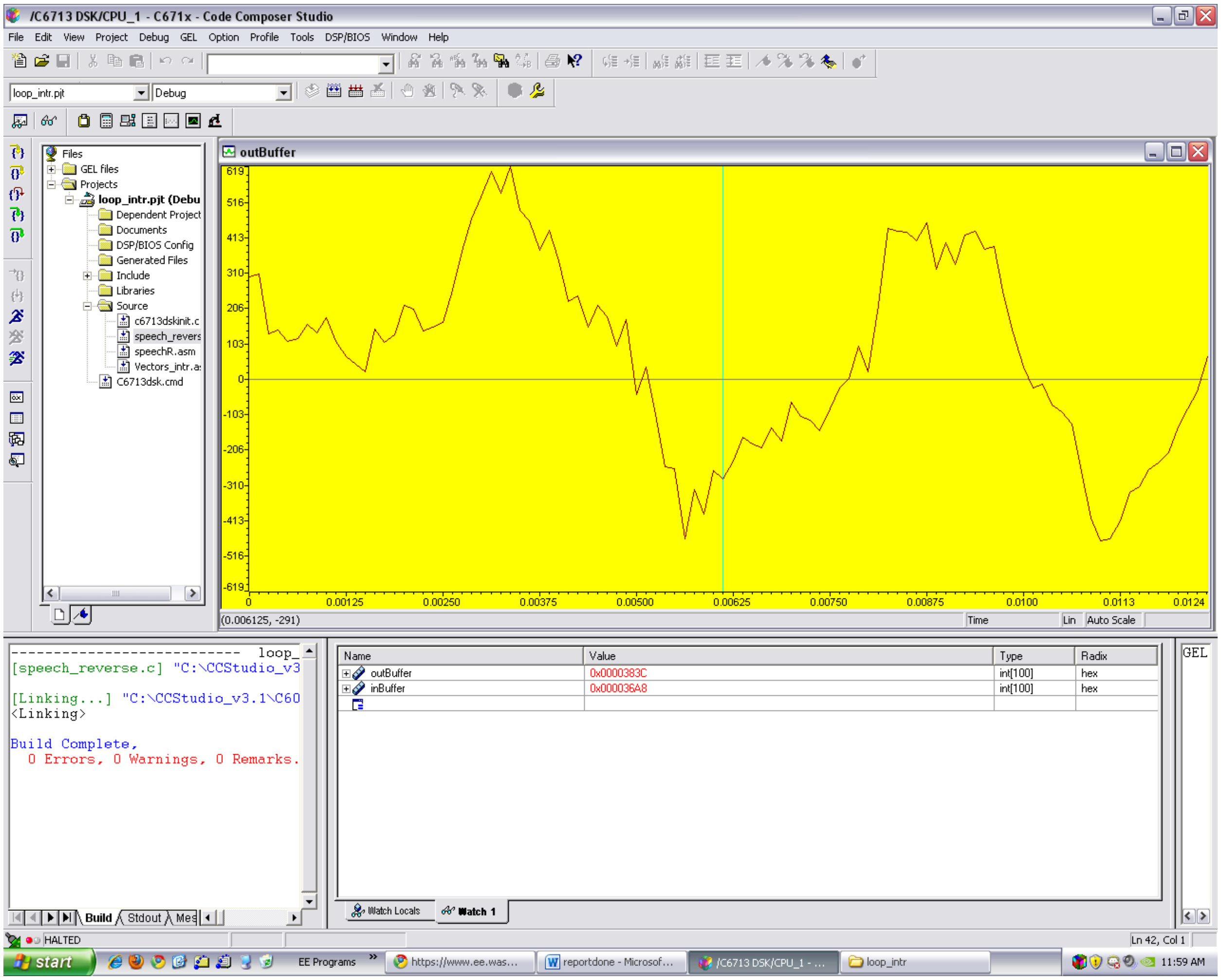
Problem3

For problem 3 we are using a C-callable assembly code to reverse an array of input samples. The c code for this problem should be able to call the reverse assembly on 100 of the input samples and skip another 100 that won't be reversed. We can look at the time domain data as below to see the reversed relationship between them(Certain samples in the input buffer don't have reversed relationship with output buffer samples, since the program has a switching function.)

Input buffer:



Output buffer:



15/15

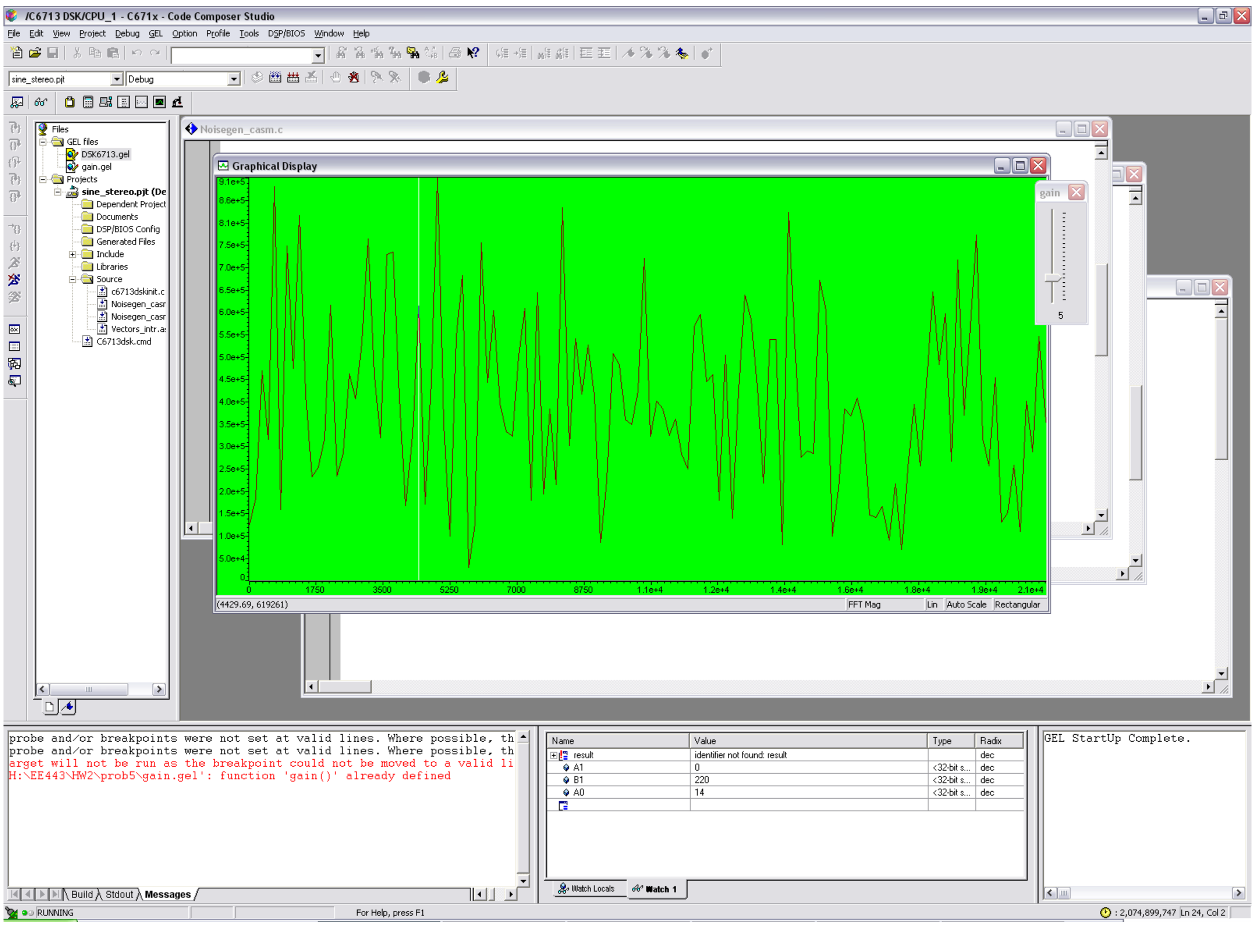
Problem 4

For problem 4 we were to take in stereo music from the PC and either output stereo mix or output the stereo signal without any processing depending on if the switch number one is up or down. To accomplish this, we used the union from Problem 2, so we could output different data for both the Left and Right channels. After we took the input samples from both input_left and input_right, if the switch was up we simply outputted the left data on the left channel and the right data on the right channel. However, if the switch was down, we created a stereo mix by subtracting the left sample data from the right sample data and outputting it on the right channel. For the left channel we added the data from both the right and left inputs and outputted them together. No tables or graphs were required. From this portion of the lab we learned how to create different mixes of output data depending on switch one.

15/15

Problem 5

For problem 5 we were to take a stereo music input from the PC and either output it normally or add pseudorandom noise depending on if switch 1 was up or down. In the C program we called an assembly function to generate random noise. Also, we had to implement a slider to control the amplitude of the random noise. The FFT of the random noise in shown below. From the graph we see that there is no predominant frequency in the “random noise” but lots of different values to make up the noise. From this part of the lab, we learned how to introduce random noise and add it to the input signal when needed.



FFT of random noise

17/20

Problem 6

For this part of the lab we were required to implement an autocorrelation function of an array of size 20 given as:

$$R(k) = \sum_{n=0}^{19-k} x(n)x(n+k)$$

We were to implement this in two different versions. First, only using the C language and second using C to call assembly to finish it. Programming this in C simply required a nested for loop accumulating the data into an int. For the next part, we called the assembly function in C to get the correct data in the array. In the assembly function it simply multiplied and added up the values to get the desired result. The results are shown below with the time difference as well. Both sets of code got the correct results, but the C calling assembly version was a lot faster, roughly five times faster. From this portion of the lab we learned that writing optimized assembly code is much faster than having the computer compile the C code for you.

C only (runtime 8911)	C called assembly (runtime 1738)
2870	2870
2660	2660
2451	2451
2244	2244
2040	2040
1840	1840
1645	1645
1456	1456
1274	1274
1100	1100
935	935
780	780
636	636
504	504
385	385
280	280
190	190
116	116
59	59
20	20

Problem 1 Code a;

```
//loop_buf.c loop program with storage
#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_LINE; // select input
#define BUFSIZE 512

int buffer[BUFSIZE];
int buf_ptr = 0;

interrupt void c_int11()         // interrupt service routine
{
  int sample_data;

  sample_data = input_left_sample();    // read input sample
  buffer[buf_ptr] = sample_data;        // store in buffer
  if(++buf_ptr >= BUFSIZE) buf_ptr = 0; // update buffer index
  output_left_sample(sample_data);      // write output sample
  return;
}

void main()
{
  comm_intr();                  // init DSK, codec, McBSP
  while(1);                     // infinite loop
}
```

Problem 1 code B:

```
//aliasing.c illustration of downsampling, aliasing, upsampling

#include "DSK6713_AIC23.h"          //codec support
Uint32 fs=DSK6713_AIC23_FREQ_16KHZ;  //set sampling rate
#include "lp6545.cof"                //filter coefficients

#define DSK6713_AIC23_INPUT_MIC 0x0015
#define DSK6713_AIC23_INPUT_LINE 0x0011
Uint16 inputsource=DSK6713_AIC23_INPUT_LINE; // select input

#define DISCARD 0
#define SAMPLE 1
```

```
short buffer[512];
short flag = DISCARD;          //toggles for 2x down-sampling
short indly[N],outdly[N];      //antialias and reconst delay lines
float yn; int i;               //filter output, index
short antialiasing = 1;        //init for no antialiasing filter
int bufl;

interrupt void c_int11()            //ISR
{
 indly[0]=(float)(input_left_sample());//new sample to antialias filter
 yn = 0.0;                   //initialize downsampled value
 if (flag == DISCARD) flag = SAMPLE; //don't discard at next sampling
 else
  {
   if (antialiasing == 1)       //if antialiasing filter desired
    {                          //compute downsampled value
     for (i = 0 ; i < N ; i++)    //using LP filter coeffs
       yn += (h[i]*indly[i]);    //filter is implemented using float
    }
   else                        //if filter is bypassed
       yn = indly[0];          //downsampled value is input value
   flag = DISCARD;             //next input value will be discarded
  }
 for (i = N-1; i > 0; i--)
                        indly[i] = indly[i-1];         //update input buffer

 outdly[0] = (yn);                                     //input to reconst filter
 yn = 0.0;                    //8 kHz sample values and zeros
 for (i = 0 ; i < N ; i++)       //are filtered at 16 kHz rate
     yn += (h[i]*outdly[i]);     //by reconstruction lowpass filter

 for (i = N-1; i > 0; i--)
     outdly[i] = outdly[i-1];    //update delays

 buffer[bufl++]=(short)yn;
 if(bufl>=512){
 bufl=0;
 }
 output_left_sample((short)yn);   //16 kHz rate sample
 return;                       //return from interrupt
}

void main()
{
 comm_intr();                  //init DSK, codec, McBSP
 while(1);                     //infinite loop
}
```

Problem 2 Code:

```
//echo_control.c echo with variable delay and feedback
#include "DSK6713_AIC23.h"                    // codec support
Uint32 fs=DSK6713_AIC23_FREQ_32KHZ;                  // set sampling rate
#define DSK6713_AIC23_INPUT_MIC 0x0015
#define DSK6713_AIC23_INPUT_LINE 0x0011
Uint16 inputsource=DSK6713_AIC23_INPUT_LINE; // select input

#define MAX_BUF_SIZE 8000       // this sets maximum length of delay
#define LEFT  0
#define RIGHT 1
union {Uint32 uint; short channel[2];} AIC23_data;

float gain = 0.5;
float gain2 = 0.5;
short buflength = 1000;
short buflength2 = 1000;
short buffer[MAX_BUF_SIZE];     // storage for previous samples
short buffer2[MAX_BUF_SIZE];
short input,output,delayed,delayed2,input2;
int i = 0;                    // index into buffer

interrupt void c_int11()     // interrupt service routine
{
  input = input_right_sample();
  input2= input_left_sample();  // read new input sample from ADC
  delayed = buffer[i];         // read delayed value from buffer
  delayed2=buffer2[i];
  AIC23_data.channel[RIGHT]=input+delayed;
  AIC23_data.channel[LEFT]=input2+delayed2;     // output sum of input and delayed values
  output_sample(AIC23_data.uint);
  buffer[i] = input + delayed*gain; // store new input and a fraction
  buffer2[i]= input2 + delayed2*gain2;                          // of the delayed value in buffer
  if(++i >= MAX_BUF_SIZE)       // test for end of buffer
    i = MAX_BUF_SIZE - buflength;
  return;                     // return from ISR
}

void main()
{
  for(i=0 ; i<MAX_BUF_SIZE ; i++)   // clear buffer
    buffer[i] = 0;
  comm_intr();                // init DSK, codec, McBSP
  while(1);                   //infinite loop
}
```

Problem 3 Code:
Assembly:

```
            .def _speechR
_speechR:

            ;A4 - inBuffer input
            ;B4 - outBuffer input
            ;A6 - 100

            MV      .L1     A6,A1; buffer data as bytes.
            MPY     .M1     A6,4,A6
            MV       A4,A7
            ADD     .S2 A6,B4,B4
            LDW .D2 *B4--,A0        ;shift A7 to point to the end of inBuffer.
            NOP 4

loop
                    LDW *A7++,A8
                    NOP 4
                    STW A8,*B4
                    NOP 4
            [A1]    B       .S2     loop
                    LDW     *B4--,A0
                    SUB     .S1 A1,1,A1
                    NOP 3
                    B       B3
                    NOP     5
            .end
```

C code:

```
//loop_intr.c loop program using interrupts
#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
#define bufSize 100
Uint16 inputsource=DSK6713_AIC23_INPUT_MIC; // select input

int inBuffer[bufSize];
int inBufIndex = 0;
int outBuffer[bufSize];
int outBufIndex = 0;
int isReversed = 0;
int* outPtr;

interrupt void c_int11()        //interrupt service routine
```

```
{

  short sample_data;
  sample_data = input_sample(); //input dataL
  inBuffer[inBufIndex++] = sample_data;
  output_sample(*outPtr++);

  if(outBufIndex >= bufSize){outBufIndex = 0;}
  if(inBufIndex >= bufSize){
        if(isReversed == 0){
              speechR(inBuffer,outBuffer,bufSize);
              outPtr = &outBuffer[outBufIndex];
        }else{
              outPtr = &inBuffer[inBufIndex];
        }
        inBufIndex = 0;
  }

  return;
}

void main()
{
 outPtr = &inBuffer[inBufIndex];
 comm_intr();          //init DSK, codec, McBSP
 while(1);             //infinite loop
}
```

Problem 4 Code:

```
//sine_stereo Sine generation to both LEFT and RIGHT channels

#include "dsk6713_aic23.h"              //codec support
Uint32 fs=DSK6713_AIC23_FREQ_48KHZ;           //set sampling rate
#define DSK6713_AIC23_INPUT_MIC 0x0015
#define DSK6713_AIC23_INPUT_LINE 0x0011
Uint16 inputsource=DSK6713_AIC23_INPUT_LINE; // select input

#define LEFT  0
#define RIGHT 1
union {Uint32 uint; short channel[2];} AIC23_data;

#define LOOPLENGTH 8       // size of look up table
short sample_dataL;
short sample_dataR;
//short loopindex = 0;       // look up table index

interrupt void c_int11() //interrupt service routine
{
        sample_dataL = input_left_sample(); //input data
        sample_dataR = input_right_sample();

        if(DSK6713_DIP_get(0)==0){
        AIC23_data.channel[RIGHT]=sample_dataR-sample_dataL; //for right channel;
        AIC23_data.channel[LEFT]=sample_dataR+sample_dataL; //for leftchannel;
        }else{
        AIC23_data.channel[RIGHT]=sample_dataR;
        AIC23_data.channel[LEFT]=sample_dataL;
        }
        output_sample(AIC23_data.uint); //output to both channels
  //if (++loopindex >= LOOPLENGTH)
   // loopindex = 0; // check for end of look up table
  return;
}

void main()
{
        comm_intr();             //init DSK,codec,McBSP
        while(1) ;                                          //infinite loop
}
```

Problem 5 C Code:

```
//Noisegen_casm.c Pseudo-random noise generation calling ASM function

#include "dsk6713_aic23.h"                    //codec-DSK support file
Uint32 fs=DSK6713_AIC23_FREQ_32KHZ; //set sampling rate
#define DSK6713_AIC23_INPUT_MIC 0x0015
#define DSK6713_AIC23_INPUT_LINE 0x0011
Uint16 inputsource=DSK6713_AIC23_INPUT_LINE; // select mic in

#define LEFT  0
#define RIGHT 1
union {Uint32 uint; short channel[2];} AIC23_data;

short sample_dataL;
short sample_dataR;
int previous_seed;
short gain;
int bufindex;
int x;
int out_buffer[1024];

short pos=32000, neg=-32000;           //scaling noise level

interrupt void c_int11()
{
	previous_seed = noisefunc(previous_seed); //call ASM function
	sample_dataL = input_left_sample(); //input data
	sample_dataR = input_right_sample();


	AIC23_data.channel[RIGHT]=sample_dataR;
	AIC23_data.channel[LEFT]=sample_dataL;

	if(DSK6713_DIP_get(0)==1){
	output_sample(AIC23_data.uint);
	}else{
		if(previous_seed & 0x01){
			x=pos;
		}else{
			x=neg;
		}
```

```
            AIC23_data.channel[RIGHT]=sample_dataR+(x*gain)/(350);
            AIC23_data.channel[LEFT]=sample_dataL + (x*gain)/(350);
            output_sample(AIC23_data.uint);
        }
        out_buffer[bufindex++]=x;
        if(bufindex>=1024) bufindex=0;
}

void main ()
{       bufindex=0;
        gain=0;
        comm_intr();                                                    //init
DSK, codec, McBSP
        previous_seed = noisefunc(0x7E521603);  //call ASM function

        while (1);                                              //infinite loop
}
```

Problem 5 assembly code:

```
;Noisegen_casmfunc.asm Noise generation C-called function

            .def    _noisefunc  ;ASM function called from C
_noisefunc  ZERO    A2          ;init A2 for seed manipulation
            MV      A4,A1       ;seed in A1
            SHR     A1,17,A1    ;shift right 17->bit 17 to LSB
            ADD     A1,A2,A2    ;add A1 to A2 => A2
            SHR     A1,11,A1    ;shift right 11->bit 28 to LSB
            ADD     A1,A2,A2    ;add again
            SHR     A1,2,A1;shift right 2->bit 30 to LSB
            ADD     A1,A2,A2    ;
            SHR     A1,1,A1;shift right 1->bit 31 to LSB
            ADD     A1,A2,A2    ;
            AND     A2,1,A2;Mask LSB of A2
            SHL     A4,1,A4;shift seed left 1
            OR      A2,A4,A4    ;Put A2 into LSB of A4
            B       B3          ;return to calling function
            NOP     5           ;5 delays for branch
```

Problem 6 C code:

```
#include <stdio.h>
#define count 20
short x[count]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20};
short y[count]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
short result2[count]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};


main(){
int i,j,k;

for(i=0;i<count;i++){
        for(j=0;j<count-i;j++){
        y[i]=x[j]*x[i+j]+y[i];
        }
        }

printf("hello2");
try(x[0],count,&result2);
printf("hello2");
for(k=0;k<count;k++){
        printf("Y= %d\n",y[k]);
        }
}
```

Problem 6 assembly code:

```
;Factfunc.asm  Assembly function called from C to find factorial

        .def    _try                ;asm function called from C
_try:           Mv  A4,A7
                        Mv  A6,A4
                        LDH *A6,A1
                        SUB B7,B7,B7
                        Mv B4,B1
Outer:
                        Mv B4,B0
                        Mv A7, A9
                        Mv A7, B9
                        LDB *B9++[B7],A0
LOOP:
                        SUB B0,1,B0

  [B0]  B   LOOP
                MPY A9,B9,A3
                LDB     *A9++,A0
                LDB     *B9++,A0
                ADD A3,A1,A1
                NOP     1

                SUB B1,1,B1
        [B1]   B Outer
                STH A1,*A6++
                SUB B4,1,B4
                SUB A1,A1,A1
                ADD B7,1,B7
                NOP 1

      B         B3                      ;return to calling routine
      NOP       5                       ;five NOPs for delay slots
      .end
```