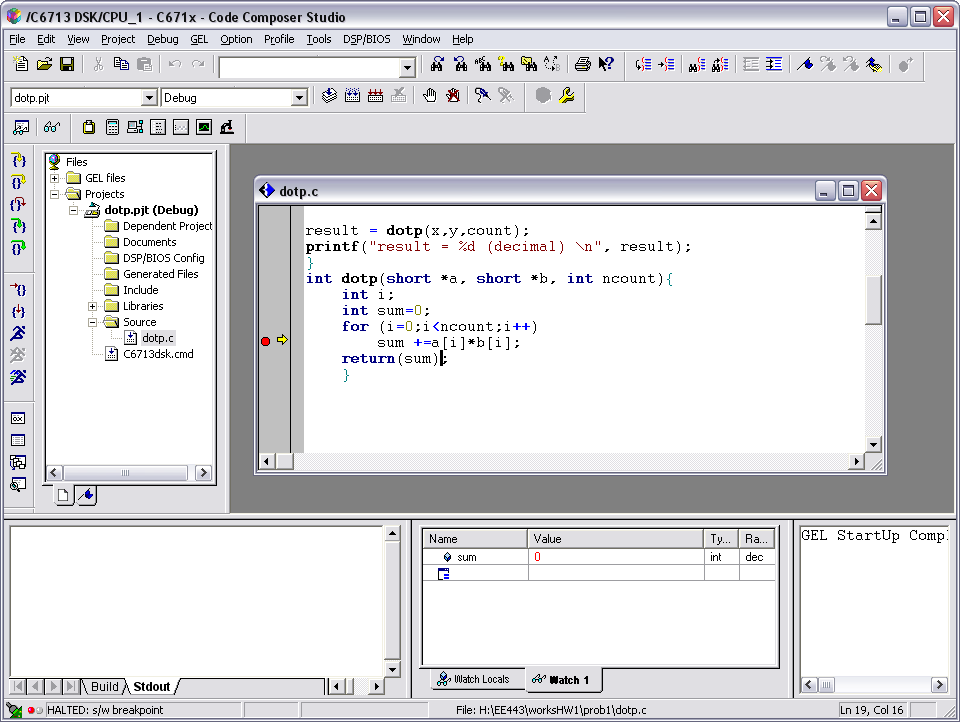
Timothy Sjoquist (0822403)

Yen-Ting Chen (1063219)

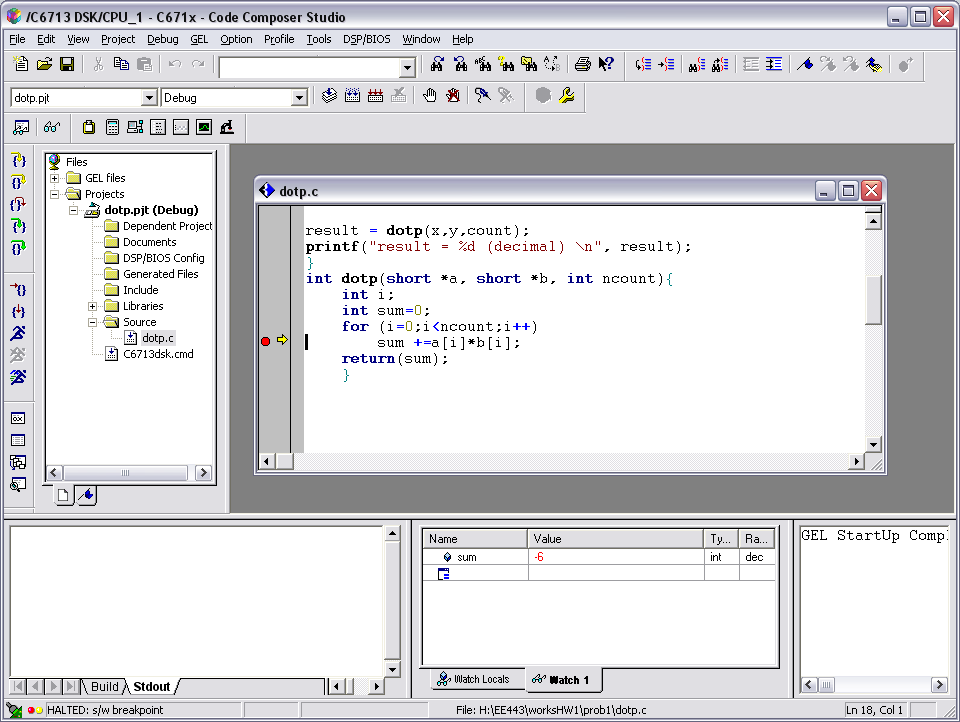
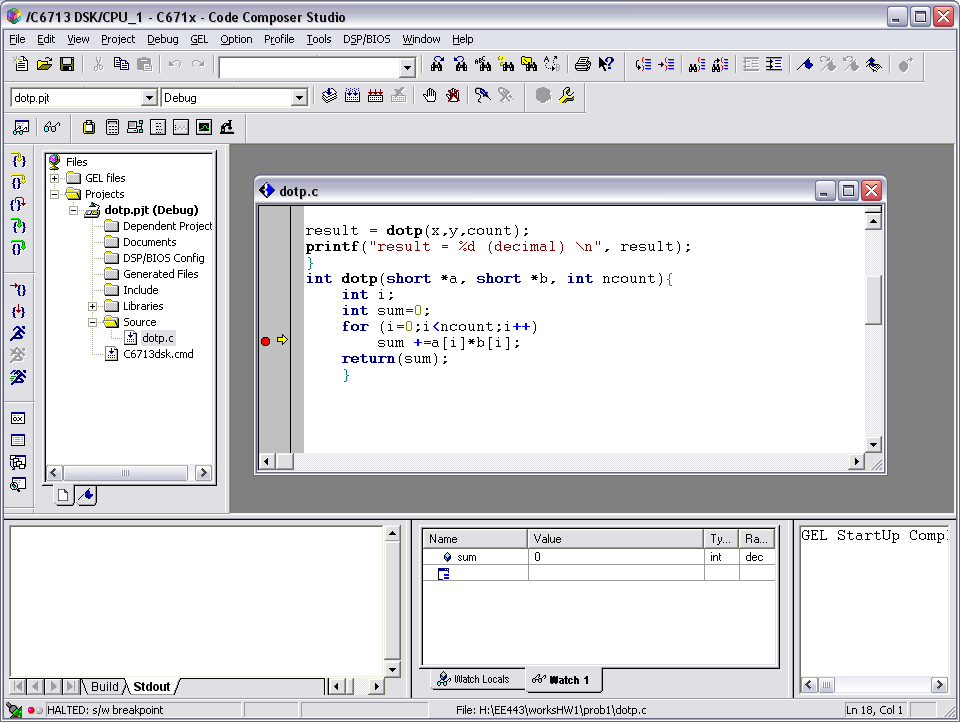
Homework 1

Problem 1:

For problem 1 we were given to vectors with a length of five and were required to take to dot product of them. The vectors we were told to multiply together were <1,-3,5,-7,9> and <0,2,-4,6,-8>. To accomplish this task, we created the two vectors and passed in pointers to the beginning of the vector as well as th number of elements in the vectors into a function. This function then went through a for loop however many elements were present, multiplying the two specific elements in either vector and adding it to the sum variable. Then we returned the sum variable and that was our result. The following screenshots show how the sum variable increases each time through the loop.

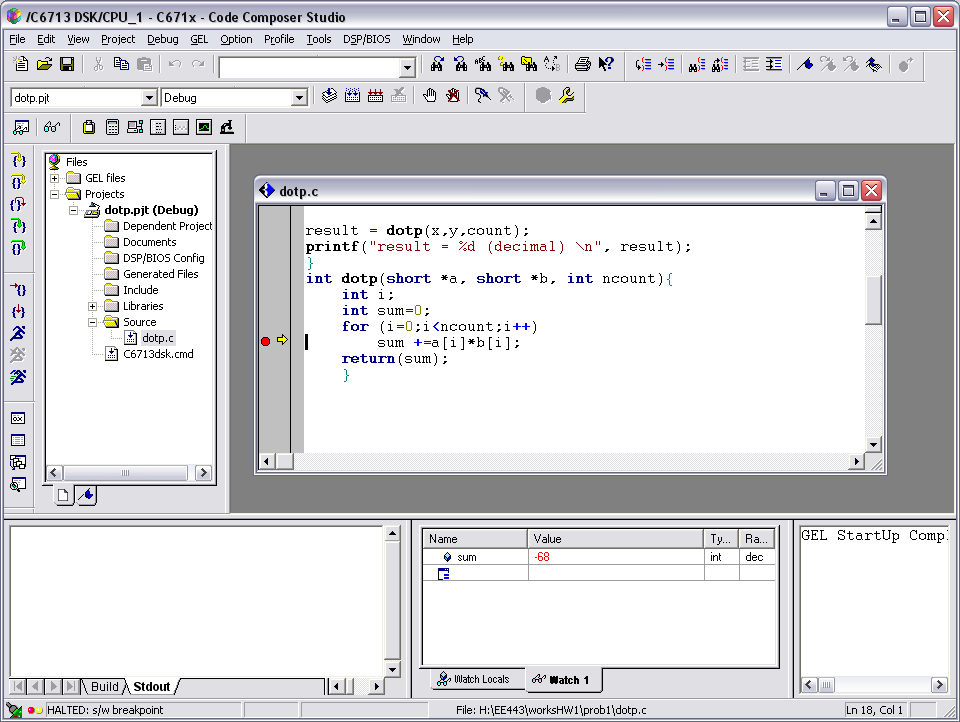
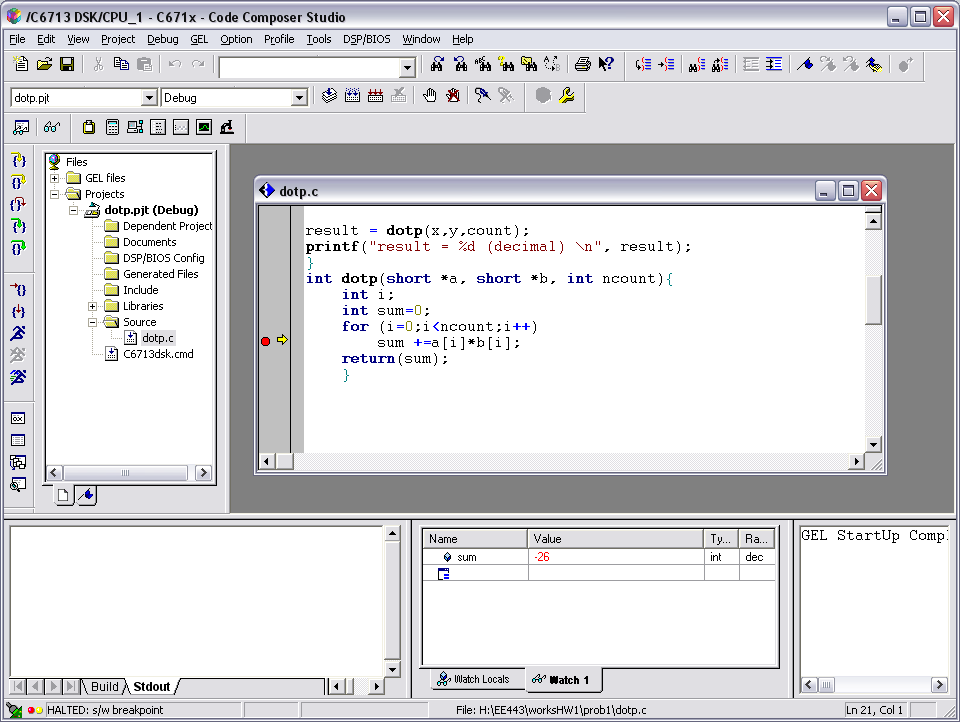


Initially Sum =0



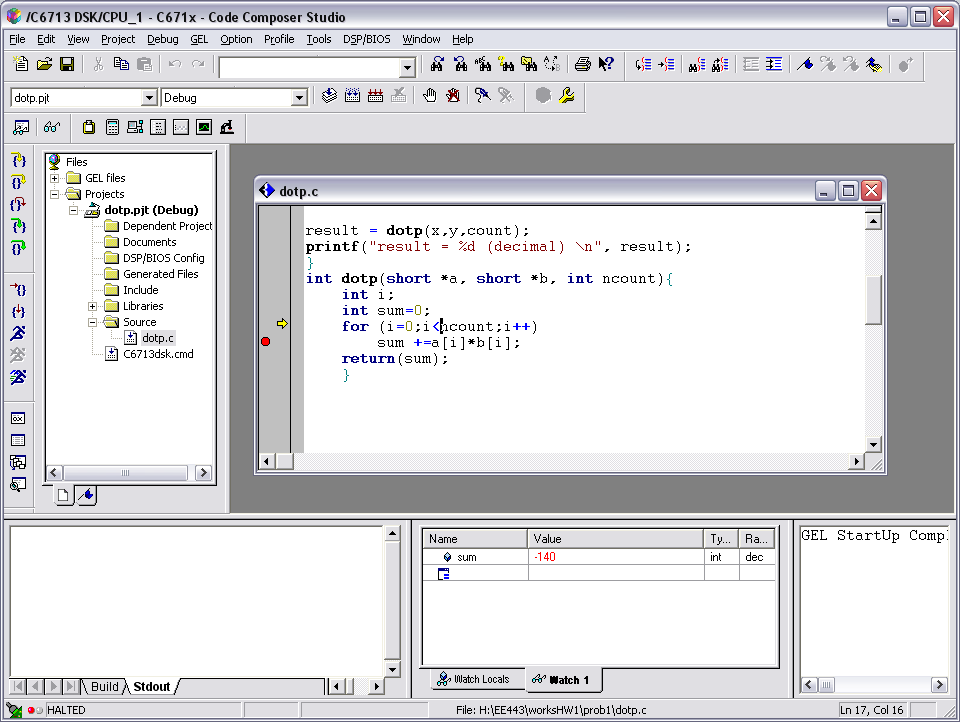
Multiplying 1\*0 and adding to sum

Multiplying -3\*2 and adding to sum



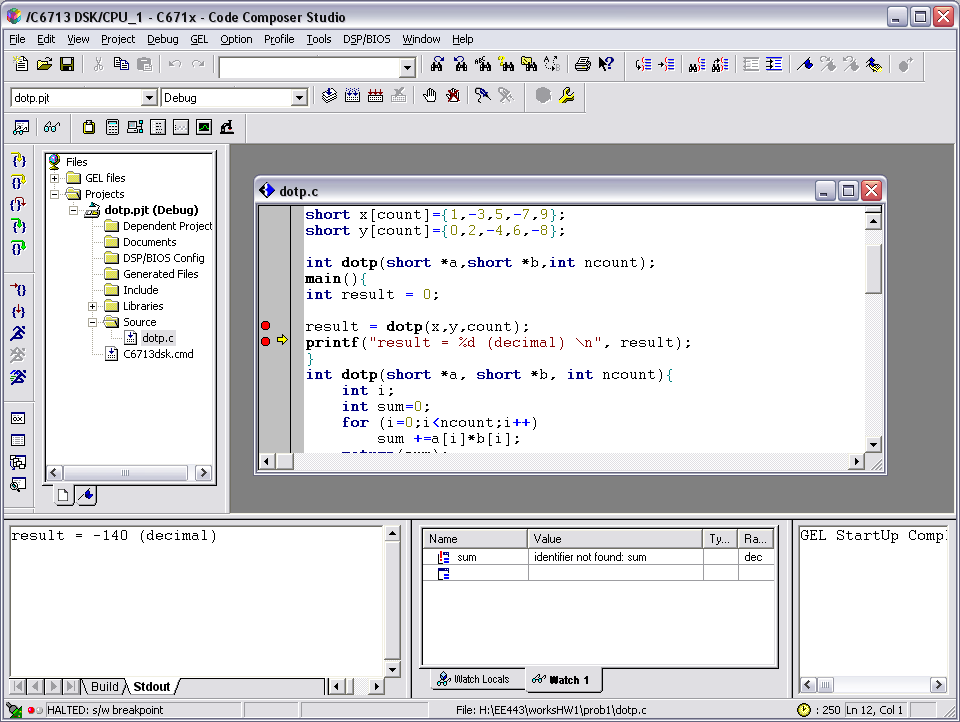
Multiplying 5\*-4 and adding to sum

Multiplying -7\*6 and adding to sum



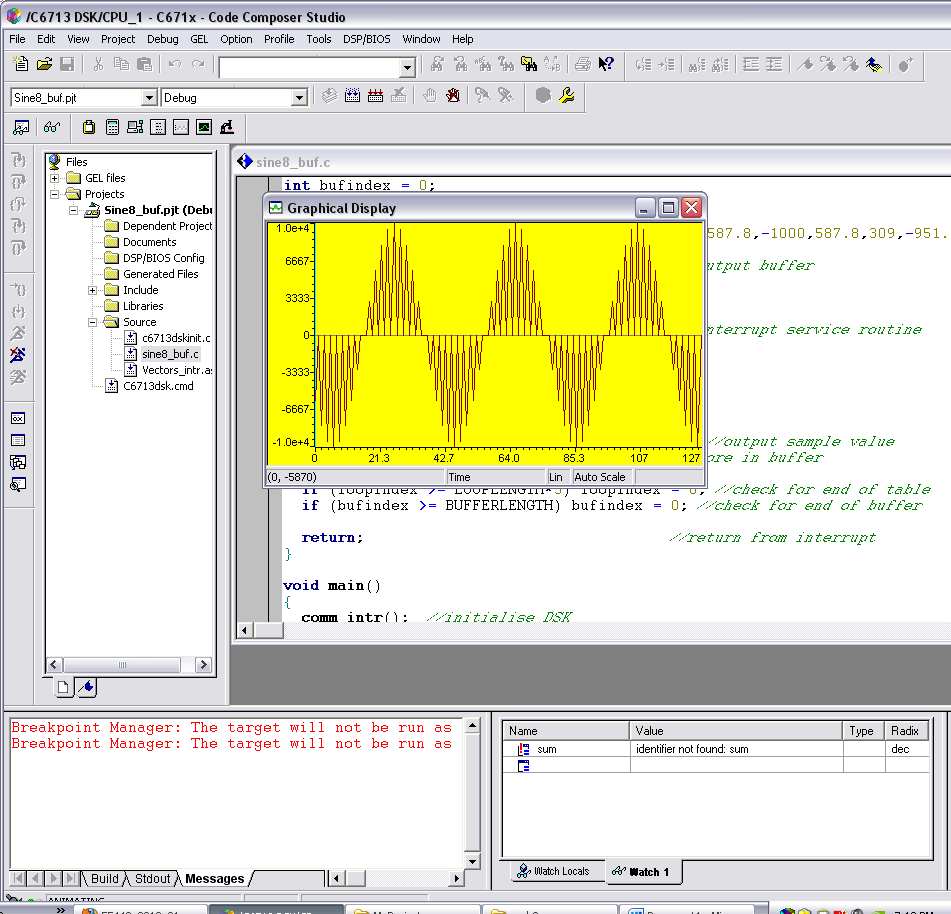
Multiplying 9\*-8 and adding to sum

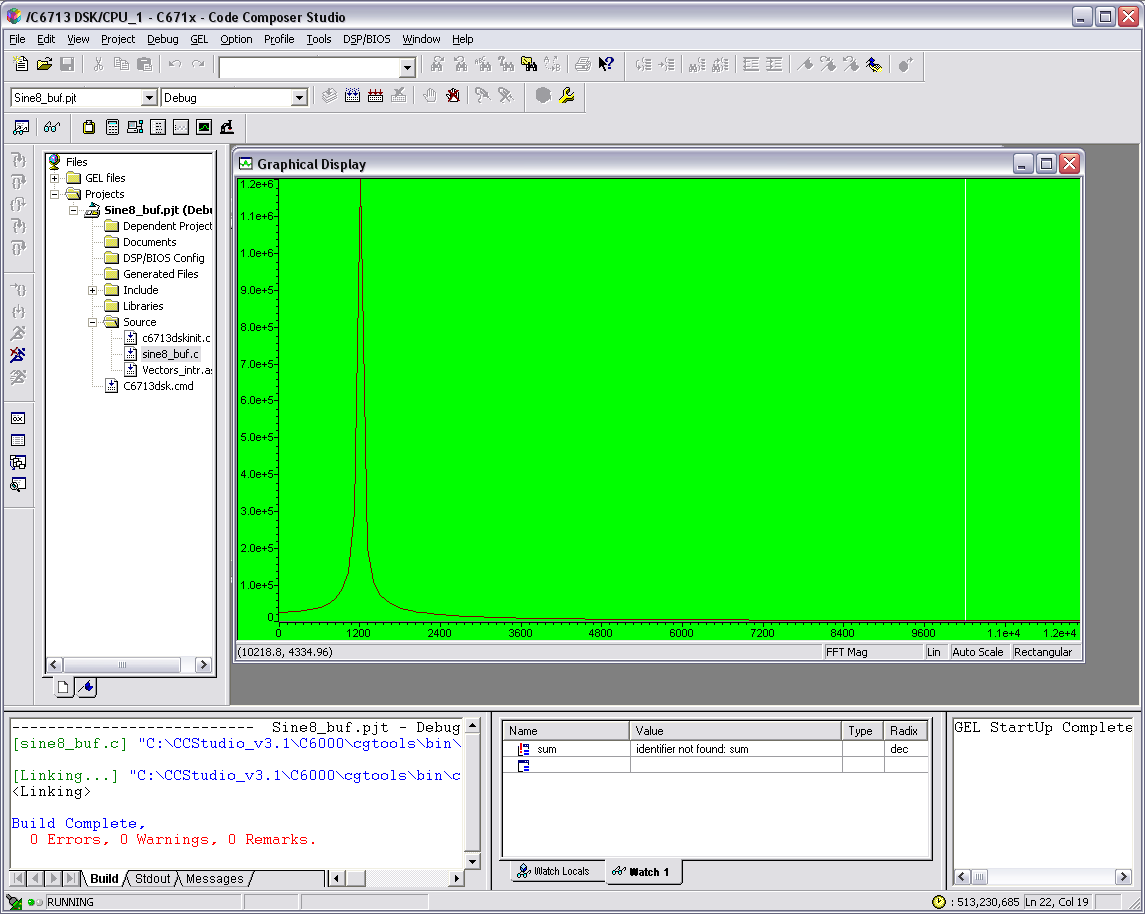
Also for this procedure we were required to calculate the running time of the code and it was 250 execution cycles as shown by the next screen shot.



From this procedure we learn how to take the dot product of two vectors.

Procedure 2:

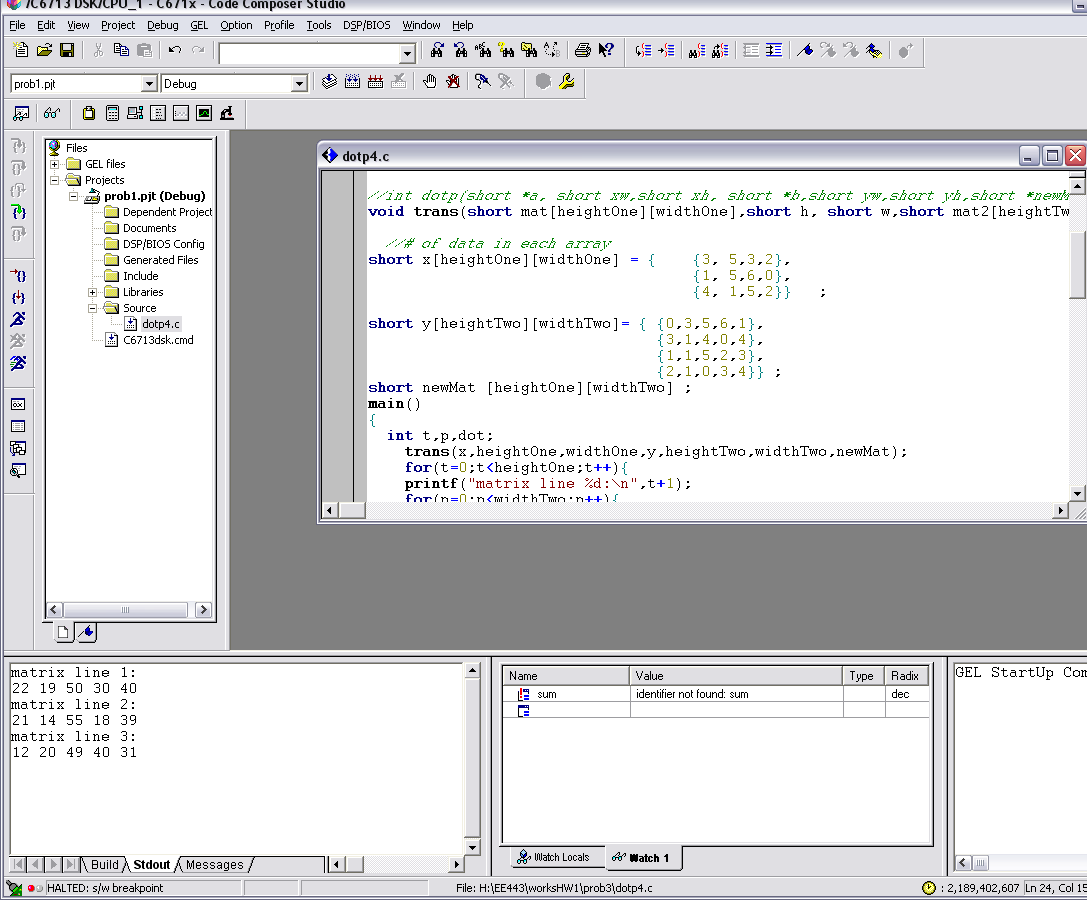
For this procedure we were to create a sine wave with a frequency of 1200Hz with a sampling rate of 8000Hz. To calculate the correct number of points to use we would do 8000/x=1200. Solving for x we get 6.67. We want a whole number so we multiply it by 3 to get 20, since we multiplied the x value by 3 to get 20, we also need to multiply the 8000Hz by 3 to get 24,000 Hz. This will give us the correct sine wave. We then could calculate the 20 samples needed to create the wave and were able to graph the data as shown in the screenshot below. The second screen shot is a shot of the FFT of the sine wave.



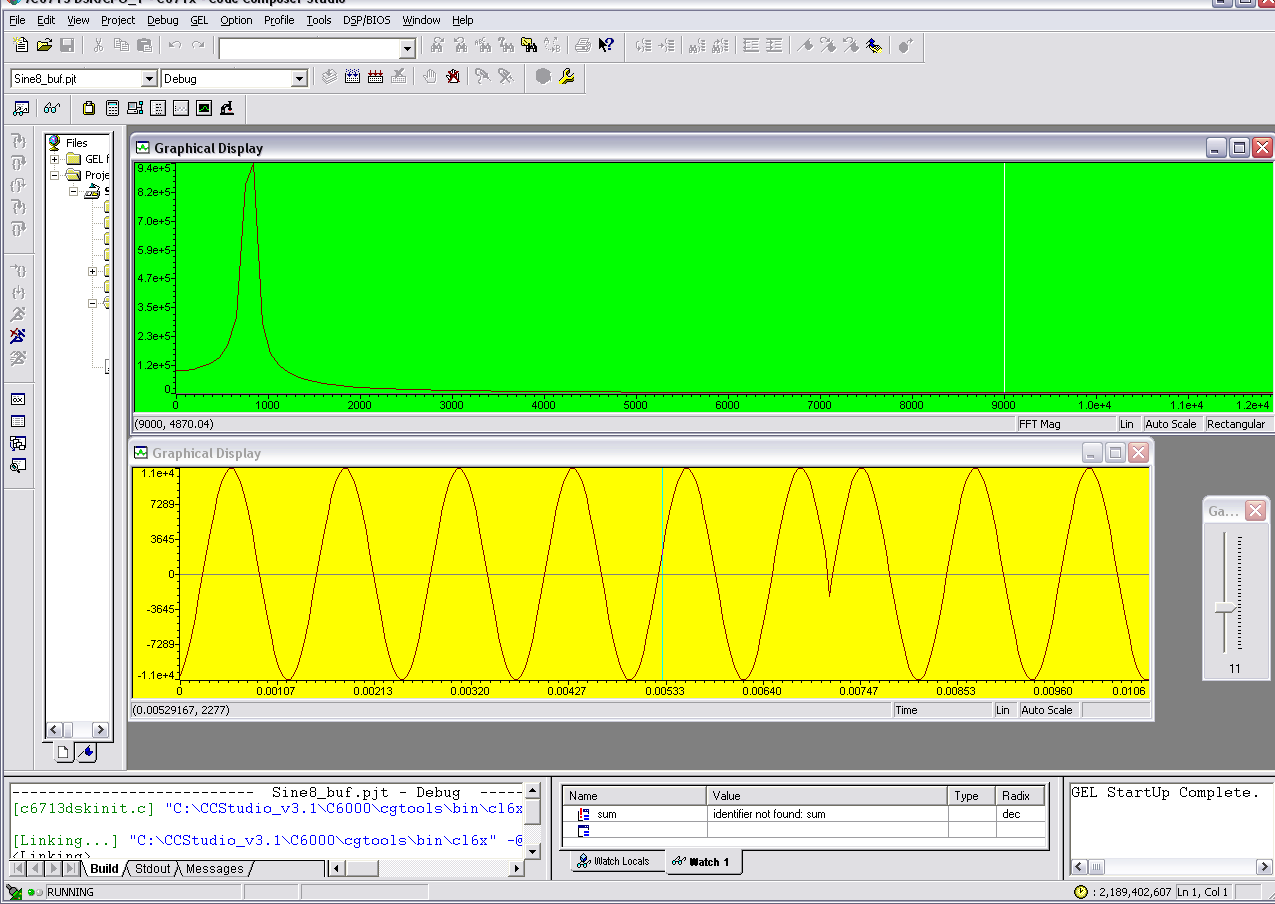
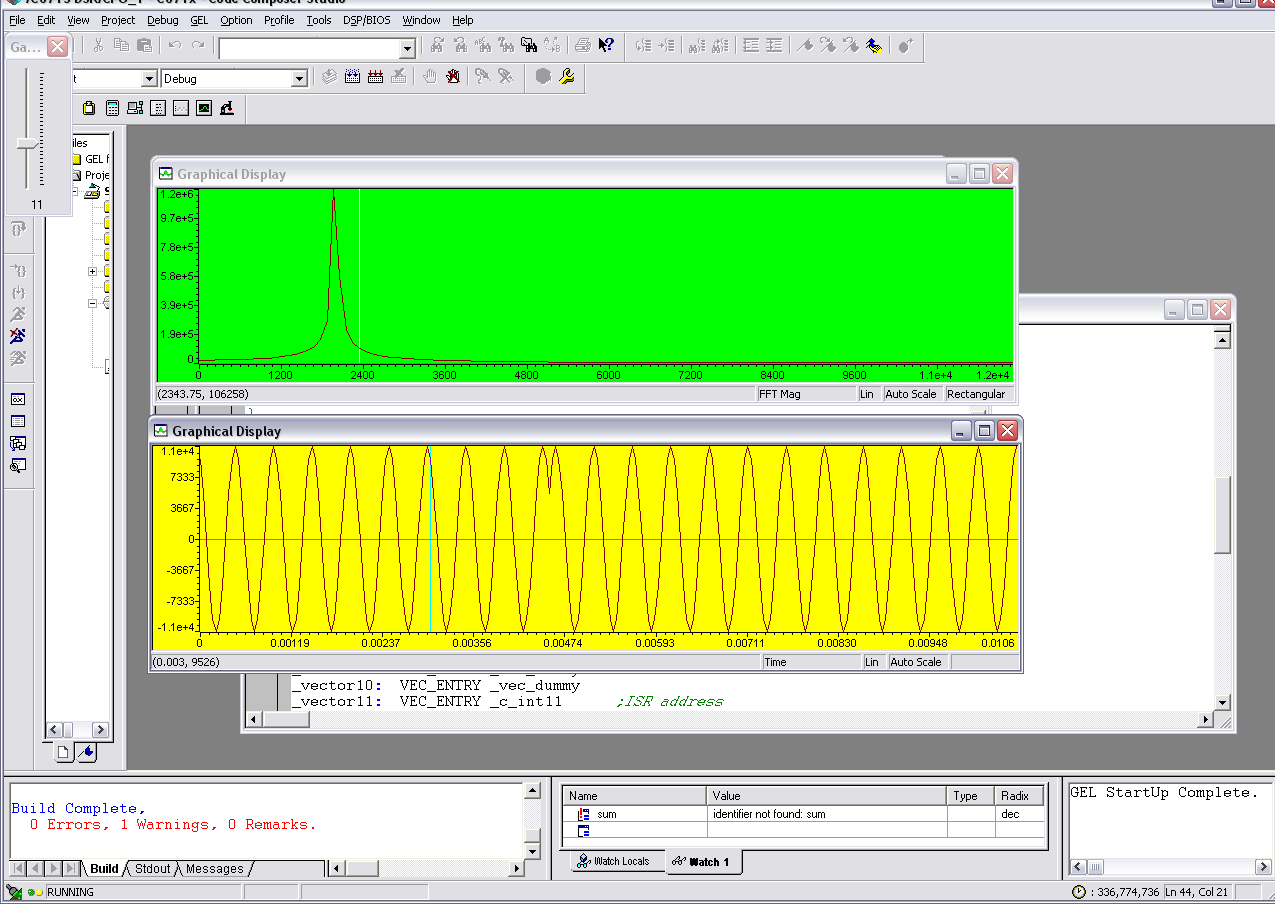
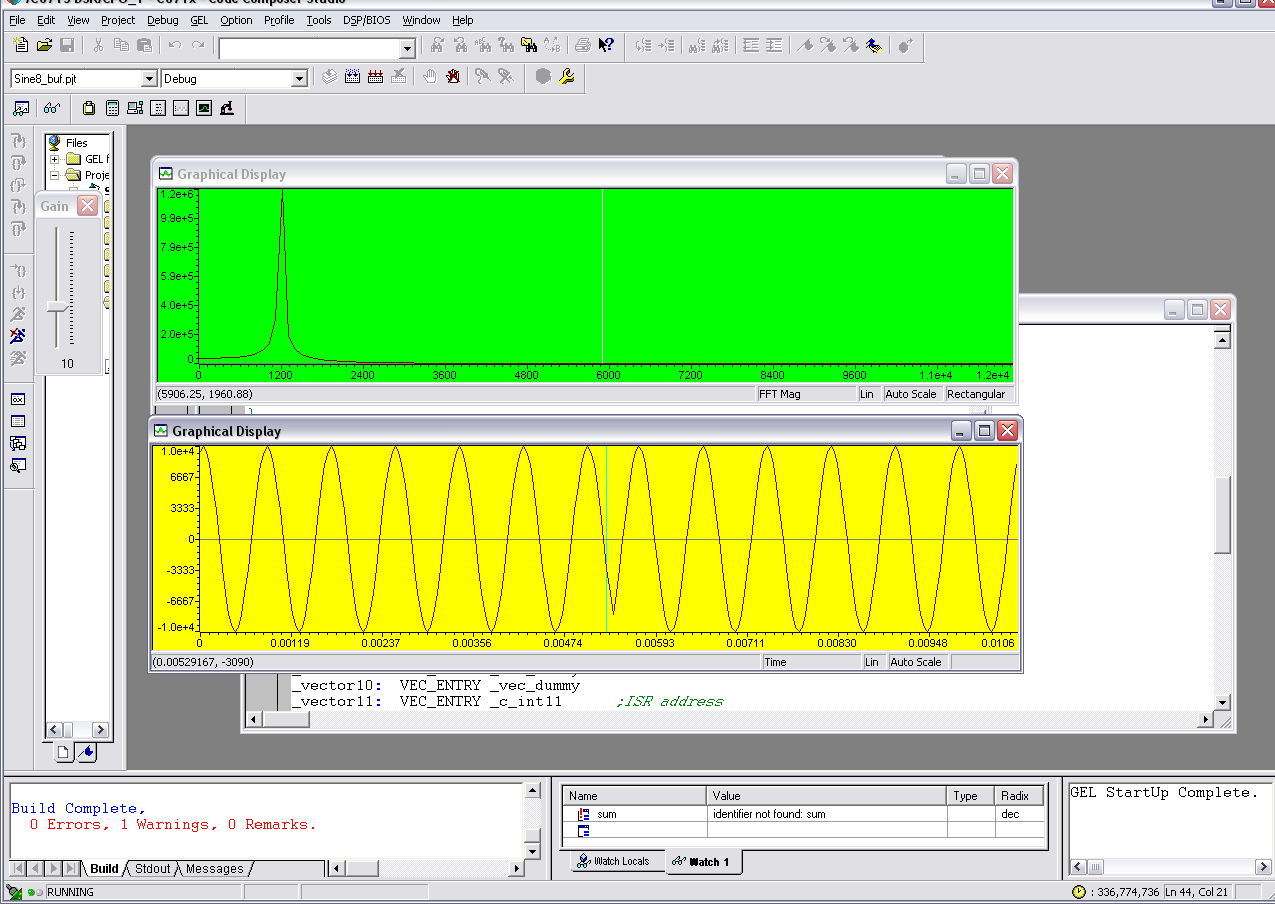
From this procedure we learn that if we want to create a sine wave with a frequency that does not divide evenly into the sampling frequency, we simply multiply the fraction by an interger to get a even number. This will allow to create a sine wave with any particular frequency

Procedure 3:

In this procedure we were required to take the dot product of two different matrices. We assume that the dimensions are correct in that the matrix’s can be multiplied. To achieve this goal, we simply had three for loops so that each element in a matrix is multiplied by the correct element in the other matrix and stores the value in a new matrix. This approach works for any size matrix as long as the two matrix’s dimensions match. The result of the given problem of multiplying the 3\*4 and a 4\*5 matrix is given in screenshot which created a 3\*5 result matrix. From this procedure we learned how to multiply two matrixes together.



Procedure 4:

For this procedure, we were required to create a multi component signal with adjustable sliders. The three components we were required to add together had a frequency of 800Hz, 1200Hz, and 2000Hz. We achieved this by finding the correct number of samples to generate those sine waves like in procedure 2. We created the sliders by writing gel files which would allow us to adjust the gain for each particular signal. In our code, we decided to have each gain set originally to zero so you could create the signal how you desired. The FFT graph and time domain graphs for each frequency are shown below. From this procedure we learn how to add different signals together and create sliders. 

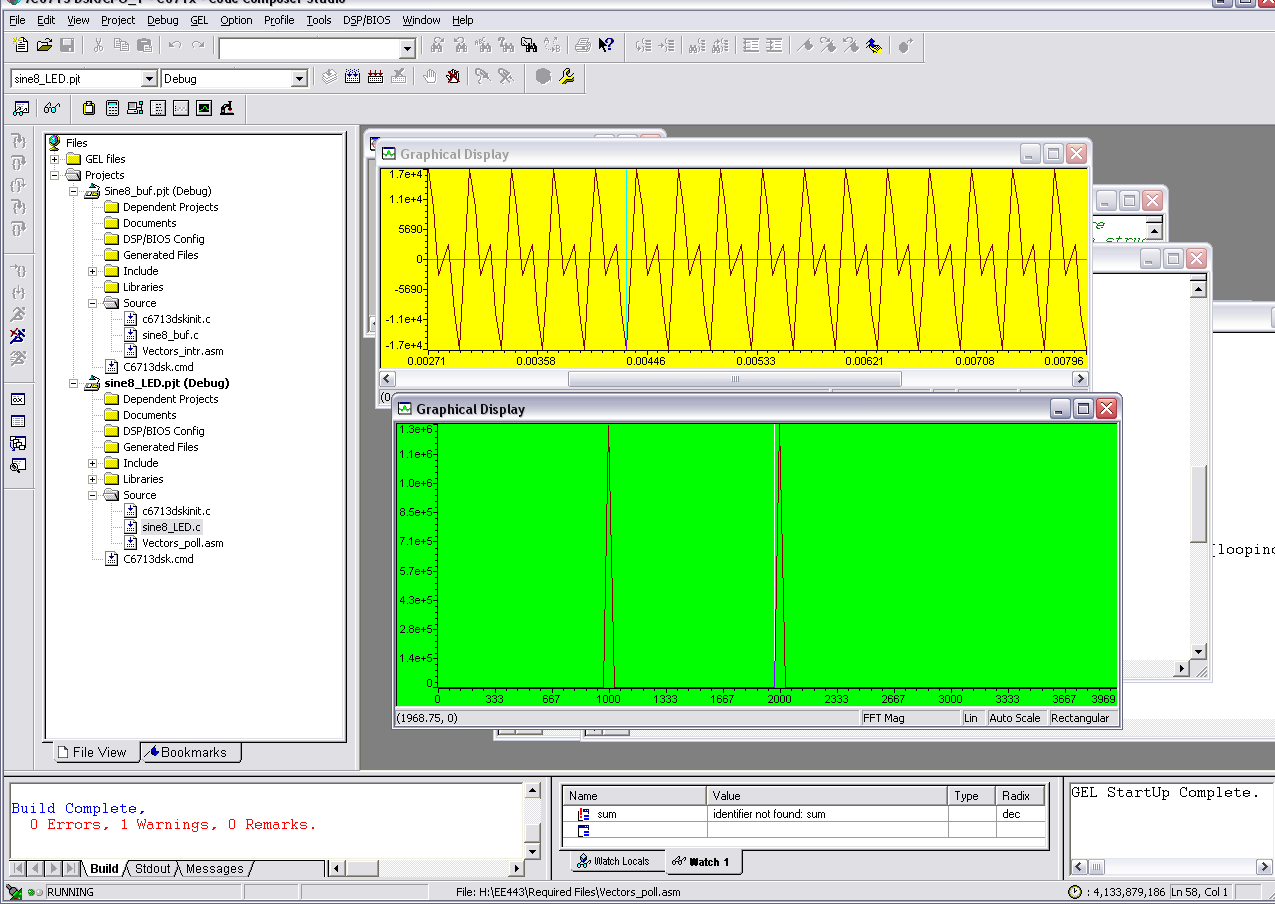
Frequency=800Hz

Frequency=1200Hz

Frequency=2000Hz

Procedure 5:

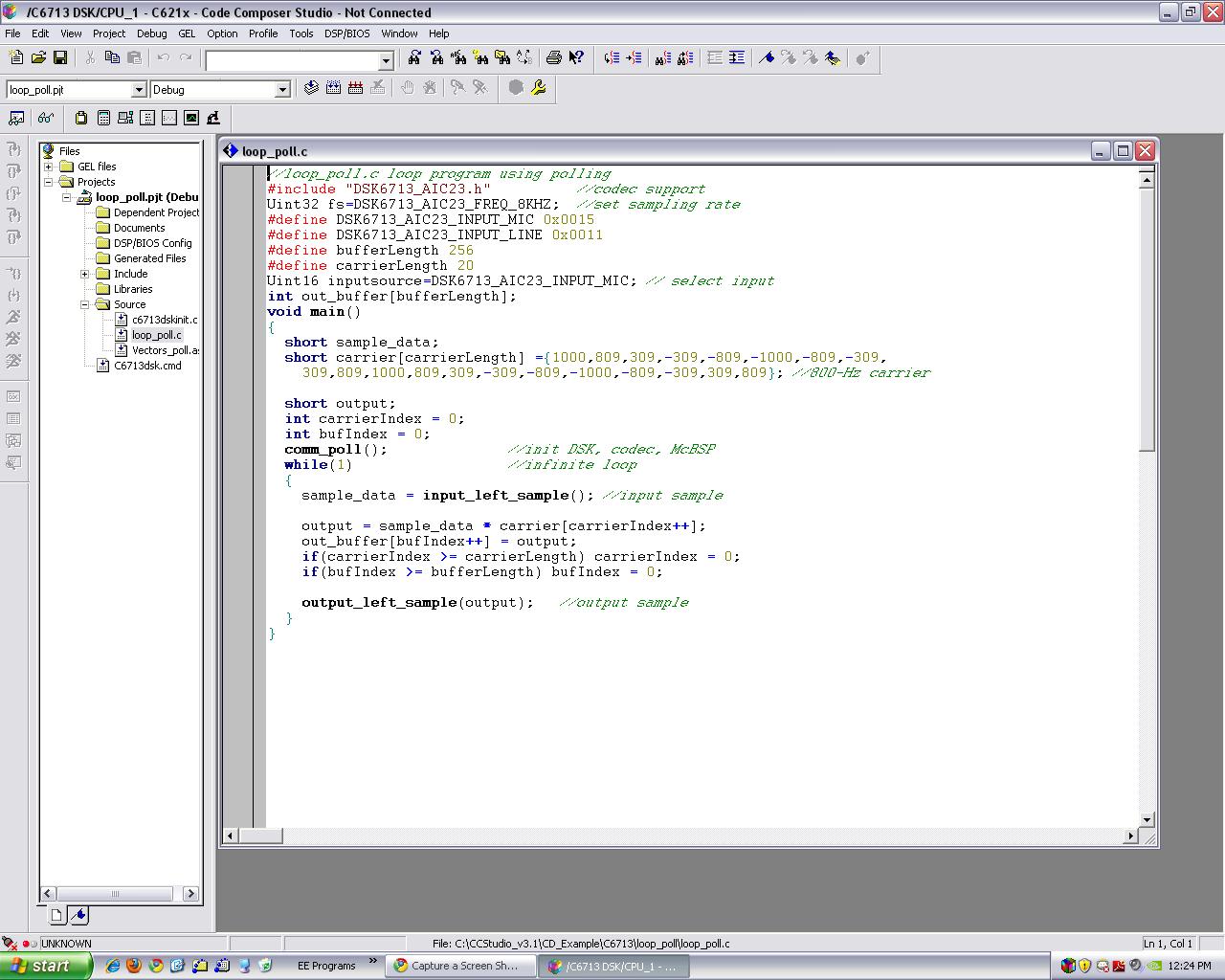
For this procedure we were required to have different outputs depending on switch 1. If the switch was down, the output was supposed to alternate every second between a sine signal of 1000Hz and a signal of 2000Hz. If the switch was up, LED was supposed to be turned on and the output graph was to be the addition of the two sine signals. To create the two sine signals, we simply found out how many points we needed to create the signal and then calculated the correct values of the points. To do the alternating, we originally planned to use the clock library in C, but that took too much time to compute and so our sine wave output was not clear. To solve this problem we simply created a counter that would change the outputs every 20,000 times through the code which came out very close to one second. The output of the added sine signal is given below in the screenshot.



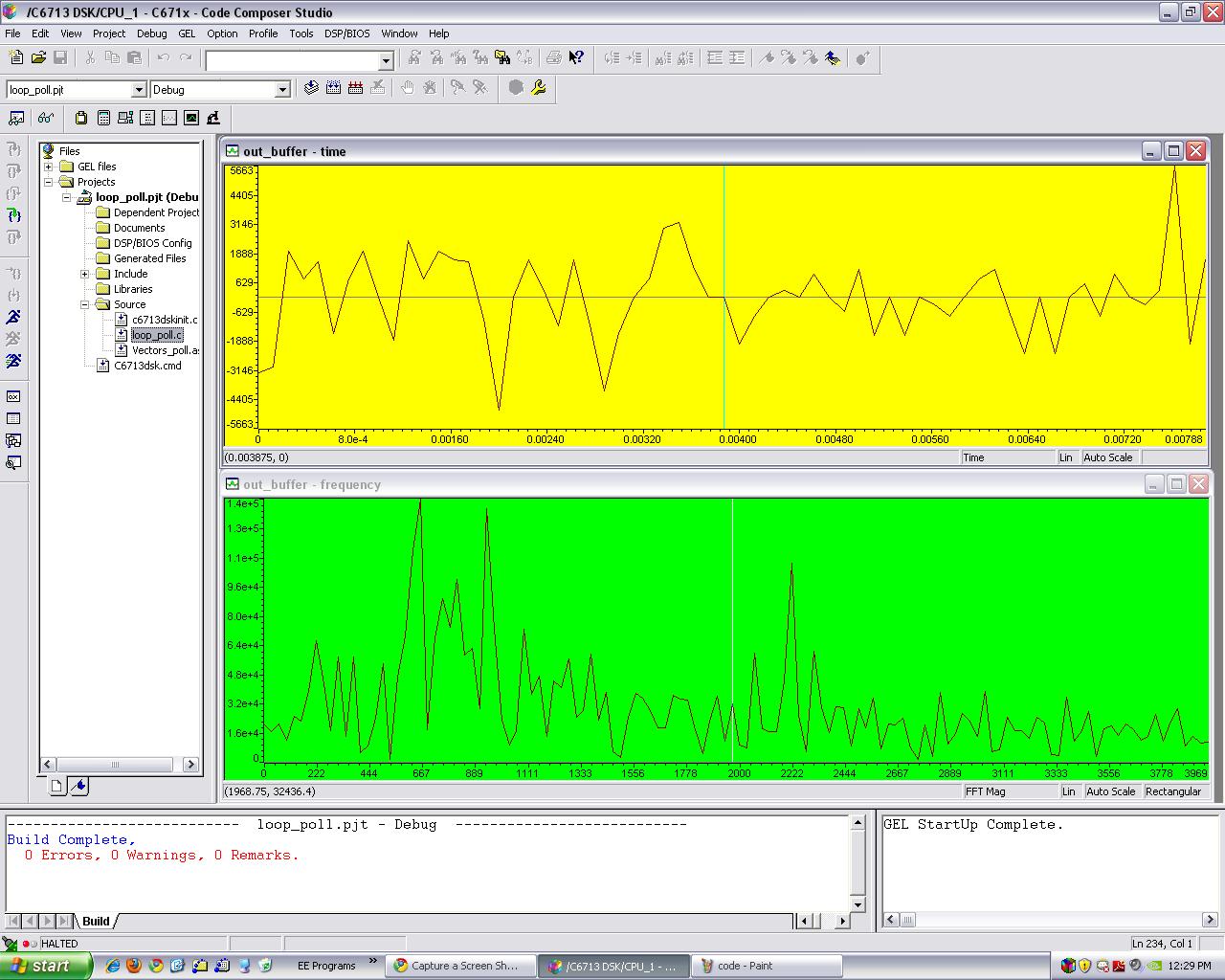
Added sine signals with 1000Hz and 2000Hz.

Procedure 6:

For this procedure we were required to implement a suppressed carrier amplitude modulation output. We were to create an 800 Hz sinusoidal carrier signal and multiply that by the input signal from the microphone. By using the polling method, we are getting data at the 8000 frequency sampling rate. The results are shown below.



Time domain and frequency domain graph:



The human speech has a range of frequencies between 300 to 3000 Hz. So by humming at the microphone around the 600 Hz area I can see a higher magnitude reflecting around that region of frequencies.

Code for Procedure 1

#include <stdio.h>

#define count 5

short x[count]={1,-3,5,-7,9};

short y[count]={0,2,-4,6,-8};

int dotp(short \*a,short \*b,int ncount);

main(){

int result = 0;

result = dotp(x,y,count);

printf("result = %d (decimal) \n", result);

}

int dotp(short \*a, short \*b, int ncount){

int i;

int sum=0;

for (i=0;i<ncount;i++)

sum +=a[i]\*b[i];

return(sum);

}

Code for problem 2

//sine8\_buf.c sine generation with output stored in buffer

#include "DSK6713\_AIC23.h" //codec support

Uint32 fs=DSK6713\_AIC23\_FREQ\_24KHZ; //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 20

#define BUFFERLENGTH 256

int loopindex = 0; //table index

int bufindex = 0;

short sine\_table[LOOPLENGTH]={0,809,-951.1,309,587.8,-1000,587.8,309,-951.1,809,0,-809,951.1,-309,-587.8,1000,-587.8,-309,951.1,-809};

int out\_buffer[BUFFERLENGTH]; //output buffer

short gain = 10;

interrupt void c\_int11() //interrupt service routine

{

short out\_sample;

out\_sample = sine\_table[loopindex%20]\*gain;

loopindex+=3;

output\_right\_sample(out\_sample);

out\_buffer[bufindex++] = out\_sample; //store in buffer

//out\_buffer2[bufindex2++]=out\_sample;

if (loopindex >= LOOPLENGTH\*3) loopindex = 0; //check for end of table

if (bufindex >= BUFFERLENGTH) bufindex = 0; //check for end of buffer

return; //return from interrupt

}

void main()

{

comm\_intr(); //initialise DSK

while(1); //infinite loop

}

Code for problem 3

//dotp4.c dot product of two vectors

#include <stdio.h> //for printf

#define widthOne 4

#define heightOne 3

#define widthTwo 5

#define heightTwo 4

//int dotp(short \*a, short xw,short xh, short \*b,short yw,short yh,short \*newMat); //function prototype

void trans(short mat[heightOne][widthOne],short h, short w,short mat2[heightTwo][widthTwo],short h2, short w2,short ans[heightOne][widthTwo]);

//# of data in each array

short x[heightOne][widthOne] = { {3, 5,3,2},

{1, 5,6,0},

{4, 1,5,2}} ;

short y[heightTwo][widthTwo]= { {0,3,5,6,1},

{3,1,4,0,4},

{1,1,5,2,3},

{2,1,0,3,4}} ;

short newMat [heightOne][widthTwo] ;

main()

{

int t,p,dot;

trans(x,heightOne,widthOne,y,heightTwo,widthTwo,newMat);

for(t=0;t<heightOne;t++){

printf("matrix line %d:\n",t+1);

for(p=0;p<widthTwo;p++){

dot=newMat[t][p];

printf("%d ",dot);

}

printf("\n");

}

}

void trans(short a[heightOne][widthOne],short h1 ,short w1,short b[heightTwo][widthTwo],short h2,short w2,short final[heightOne][widthTwo])

{

int i,j,k,temp;

int t,p;

int dot;

int check,check2;

for(t=0;t<h1;t++){

for(p=0;p<w2;p++){

final[t][p]=0;

}

}

for(i=0;i<h1;i++){

for(j=0;j<w1;j++)

{

temp=a[i][j];

for(k=0;k<w2;k++){

check2=b[j][k];

dot=temp\*check2;

final[i][k]=final[i][k]+dot;

}

}

}

}

Code for problem 4

//sine8\_buf.c sine generation with output stored in buffer

#include "DSK6713\_AIC23.h" //codec support

Uint32 fs=DSK6713\_AIC23\_FREQ\_24KHZ; //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 20

#define LOOPLENGTH2 20

#define LOOPLENGTH3 20

#define BUFFERLENGTH 256

int loopindex = 0;

int loopindex2=0;

int loopindex3=0; //table index

int bufindex = 0;

int bufindex2=0;

int bufindex3=0;

short sine\_table800[30] = {0,207.9,406.7,587.8,743.1,866,951.1,994.5,994.5,951.1,866,743.1,587.8,406.7,207.9,0,-207.9,-406.7,-587.8,-743.1,-866,-951.1,-994.5,-994.5,-951.1,-866,-743.1,-587.8,-406.7,-207.9};

//buffer index

short sine\_table[LOOPLENGTH]={0,809,-951.1,309,587.8,-1000,587.8,309,-951.1,809,0,-809,951.1,-309,-587.8,1000,-587.8,-309,951.1,-809};

short sine\_table2k[12]={0, 500, 866, 1000,866,500,0,-500,-866,-1000,-866,-500};

int out\_buffer[BUFFERLENGTH]; //output buffer

int out\_buffer2[BUFFERLENGTH];

short gain1 = 0;

short gain2 = 0;

short gain3 = 0;

interrupt void c\_int11() //interrupt service routine

{

short out\_sample;

short out\_sample2;

short out\_sample3;

short alpha=0;

out\_sample3=sine\_table800[loopindex3]\*gain3;

loopindex3++;

out\_sample2= sine\_table2k[loopindex2]\*gain2;

loopindex2++;

out\_sample = sine\_table[loopindex%20]\*gain1;

loopindex+=3;

output\_right\_sample(out\_sample+out\_sample2+out\_sample3);

//output\_left\_sample(out\_sample2); //output sample value

out\_buffer[bufindex++] = (out\_sample3+out\_sample2+out\_sample); //store in buffer

//out\_buffer2[bufindex2++]=out\_sample;

if (loopindex >= LOOPLENGTH\*3) loopindex = 0; //check for end of table

if (bufindex >= BUFFERLENGTH) bufindex = 0; //check for end of buffer

if (bufindex2>= BUFFERLENGTH) bufindex2=0;

if (loopindex2>=12)loopindex2=0;

if (loopindex3>=30)loopindex3=0;

return; //return from interrupt

}

void main()

{

comm\_intr(); //initialise DSK

while(1); //infinite loop

}

Code for problem 5

//sine8\_LED.c sine generation with DIP switch control

#include "dsk6713\_aic23.h"

#include <time.h> //codec support

Uint32 fs = DSK6713\_AIC23\_FREQ\_8KHZ; //set samplling rate

#define DSK6713\_AIC23\_INPUT\_MIC 0x0015

#define DSK6713\_AIC23\_INPUT\_LINE 0x0011

Uint16 inputsource=DSK6713\_AIC23\_INPUT\_MIC; //select input

#define LOOPLENGTH 8

int count(int what);

int out\_buffer[256];

short loopindex = 0;

short loopindex2 = 0; //table index

short gain = 10; //gain factor

short sine\_table[LOOPLENGTH] =

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

short sine\_table2k[4]=

{0,1000,0,-1000};

void main()

{

short temp;

short prev;

short outsample;

int bufindex = 0;

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

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

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

temp=0;

prev=0;

while(1)

{

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

{

DSK6713\_LED\_off(0); //turn LED #0 on

if(prev==1){

prev=0;

temp=0;

}

if(temp<10000){

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

temp++;

if(loopindex >= 8) loopindex=0;

}else if (temp<20000){

output\_left\_sample(sine\_table2k[loopindex2++]\*gain);

if(loopindex2>=4)loopindex2=0;

temp++;

}else{

temp=0;

}

}

else{

DSK6713\_LED\_on(0); //else turn LED #0 off

if(prev==0){

loopindex2=0;

loopindex=0;

prev=1;

}

outsample=sine\_table2k[loopindex2++]\*gain+sine\_table[loopindex++]\*gain;

output\_left\_sample(outsample);

if(loopindex2>=4)loopindex2=0;

if(loopindex>=8)loopindex=0;

out\_buffer[bufindex++] = outsample;

if (bufindex >= 256) bufindex = 0;

}

} //end of while(1)

} //end of main

Code for problem 6

//loop\_poll.c loop program using polling

#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 bufferLength 256

#define carrierLength 20

Uint16 inputsource=DSK6713\_AIC23\_INPUT\_MIC; // select input

int out\_buffer[bufferLength];

void main()

{

short sample\_data;

short carrier[carrierLength] ={1000,809,309,-309,-809,-1000,-809,-309,

309,809,1000,809,309,-309,-809,-1000,-809,-309,309,809}; //800-Hz carrier

short output;

int carrierIndex = 0;

int bufIndex = 0;

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

while(1) //infinite loop

{

sample\_data = input\_left\_sample(); //input sample

output = sample\_data \* carrier[carrierIndex++];

out\_buffer[bufIndex++] = output;

if(carrierIndex >= carrierLength) carrierIndex = 0;

if(bufIndex >= bufferLength) bufIndex = 0;

output\_left\_sample(output); //output sample

}

}