65/100

Timothy Sjoquist (0822403)

Yen-Ting Chen(1063219)

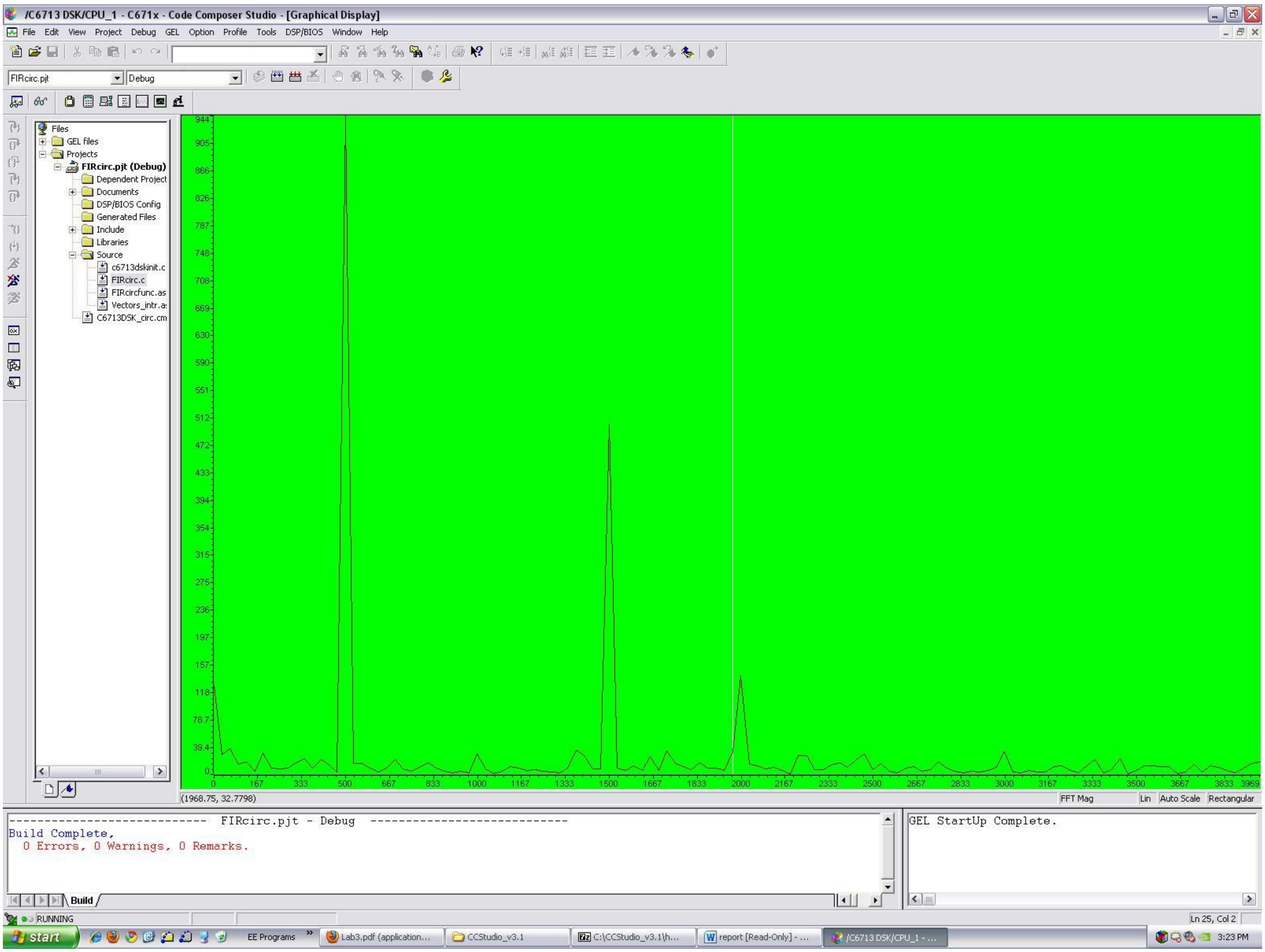
Lab 3

10/10

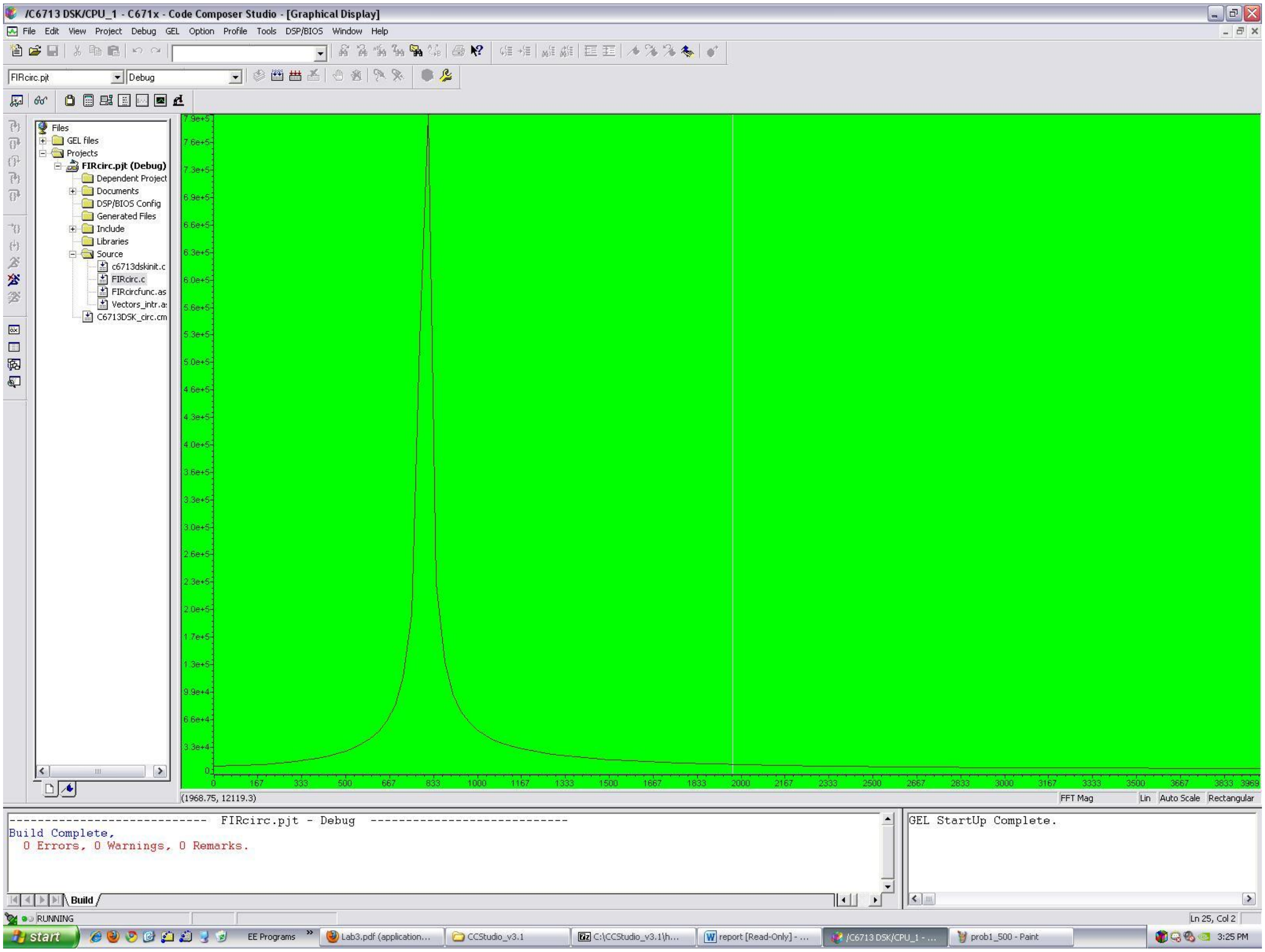
Problem 1:

For problem 1 we are designing a band pass filter according to the specification given with Fs1=0.8KHz, Fp1=1.6KHz, weight_pass=1.0, Fp2=2.4KHz, Fs2=3.4 KHz, weight_stop1=1.5, weight_stop2=2.0. We showed frequency domain values by giving different frequency values inputted by the function generator.

500 Hz

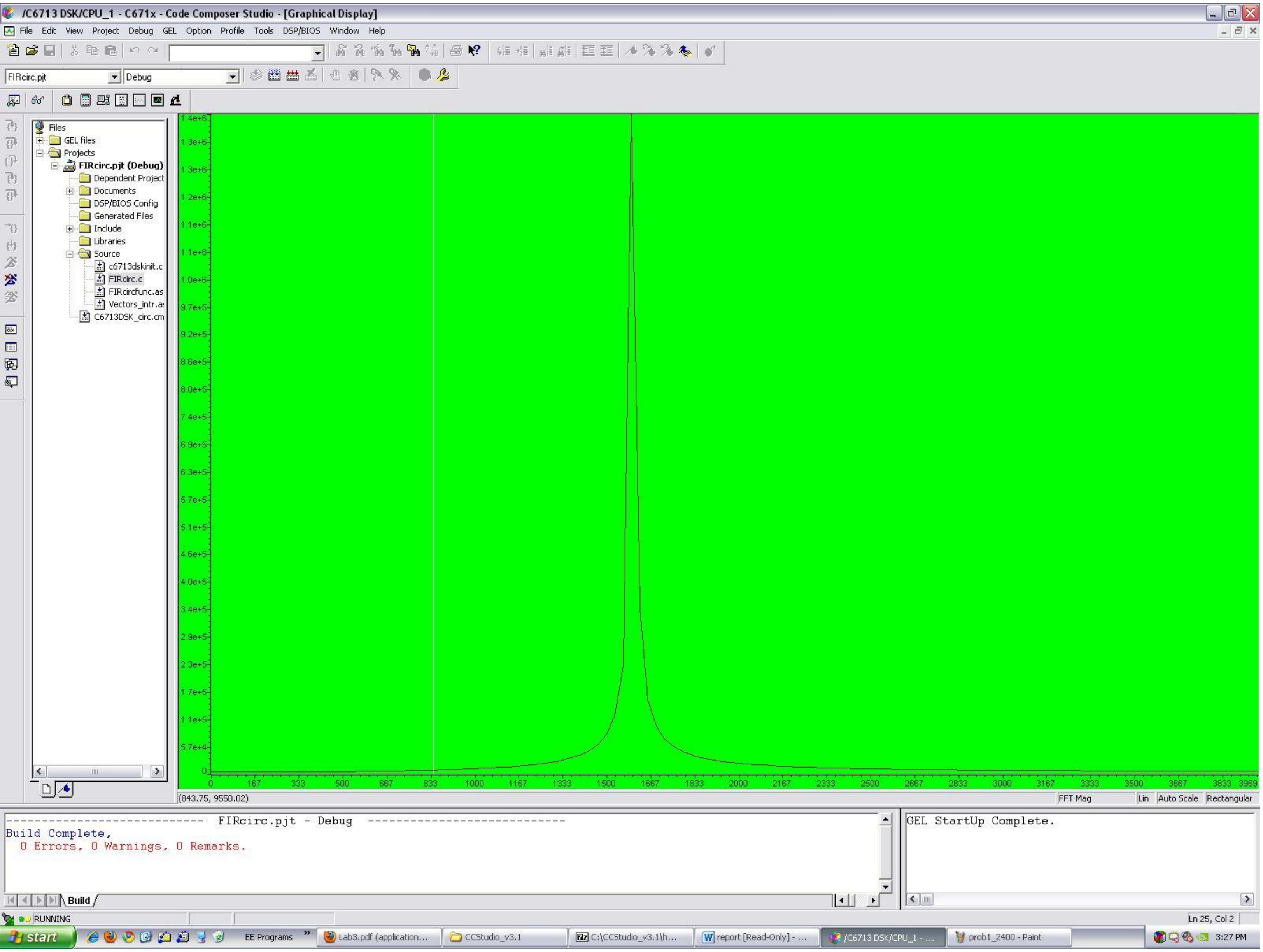


800 Hz

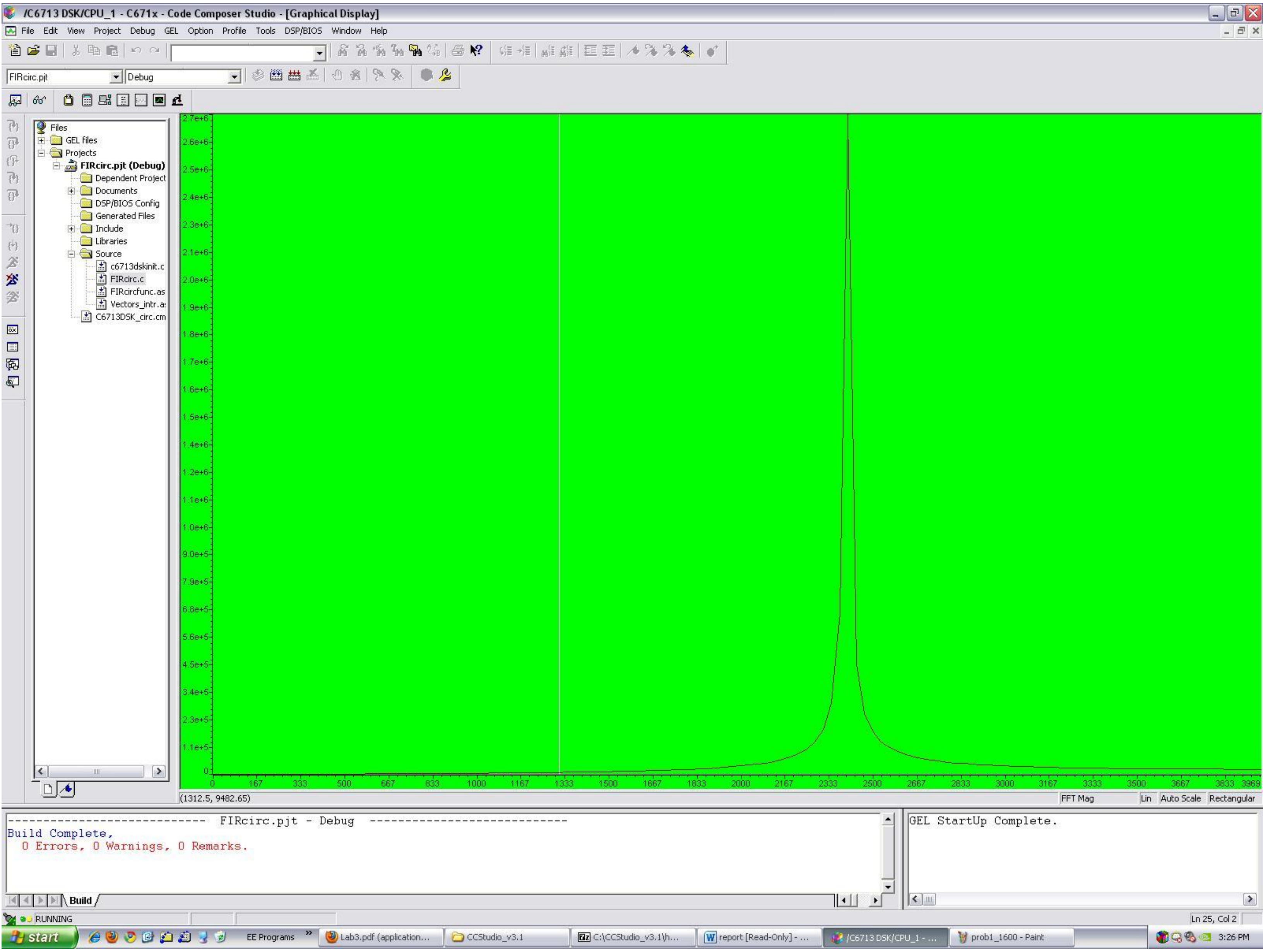


/C6713 DSK/CPU_1 - C671x - Code Composer Studio - [Graphical Display]
File Edit View Project Debug GEL Option Profile Tools DSP/BIOS Window Help
FIRcirc.pjt
Debug
Files
GEL files
Projects
FIRcirc.pjt (Debug)
Dependent Project
Documents
DSP/BIOS Config
Generated Files
Include
Libraries
Source
c6713dskinit.c
FIRcirc.c
FIRcircfunc.as
Vectors_intr.a:
C6713DSK_circ.cm
(1968.75, 12119.3)
FFT Mag
Lin
Auto Scale
Rectangular
---------------------------- FIRcirc.pjt - Debug ----------------------------
Build Complete,
0 Errors, 0 Warnings, 0 Remarks.
Build
GEL StartUp Complete.
RUNNING
Ln 25, Col 2
start
EE Programs
Lab3.pdf (application...
CCStudio_v3.1
report [Read-Only] - ...
prob1_500 - Paint
3:25 PM

1600 Hz

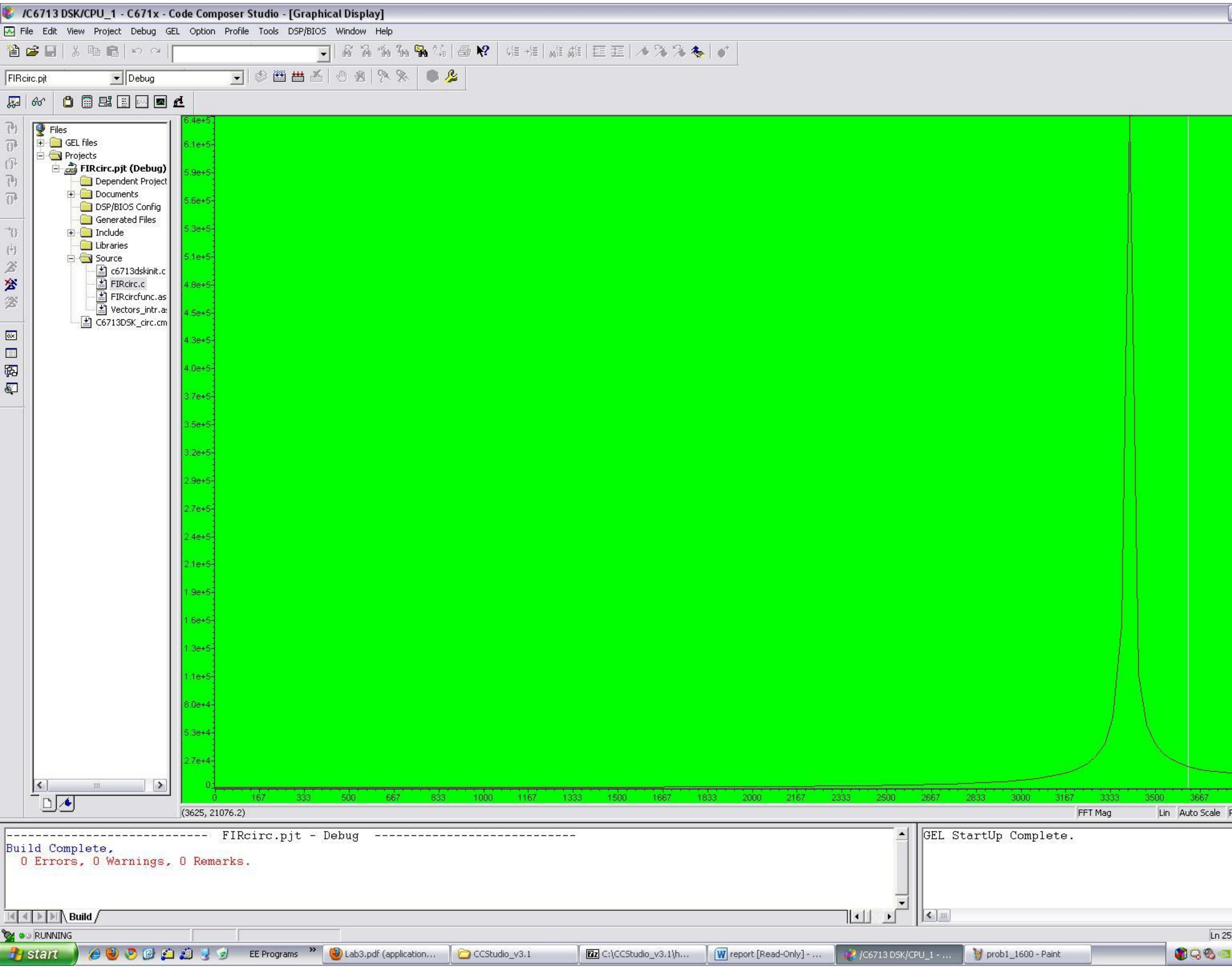

/C6713 DSK/CPU_1 - C671x - Code Composer Studio - [Graphical Display]
File Edit View Project Debug GEL Option Profile Tools DSP/BIOS Window Help
FIRcirc.pjt
Debug
Files
GEL files
Projects
FIRcirc.pjt (Debug)
Dependent Project
Documents
DSP/BIOS Config
Generated Files
Include
Libraries
Source
c6713dskinit.c
FIRcirc.c
FIRcircfunc.as
Vectors_intr.a
C6713DSK_circ.cm
(843.75, 9550.02)
FFT Mag
Lin
Auto Scale
Rectangular
FIRcirc.pjt - Debug
Build Complete,
0 Errors, 0 Warnings, 0 Remarks.
GEL StartUp Complete.
Build
RUNNING
Ln 25, Col 2
start
EE Programs
Lab3.pdf (application...
CCStudio_v3.1
C:\CCStudio_v3.1\h...
report [Read-Only] - ...
/C6713 DSK/CPU_1 - ...
prob1_2400 - Paint
3:27 PM

2400 Hz



/C6713 DSK/CPU_1 - C671x - Code Composer Studio - [Graphical Display]
File Edit View Project Debug GEL Option Profile Tools DSP/BIOS Window Help
FIRcirc.pjt
Debug
Files
GEL files
Projects
FIRcirc.pjt (Debug)
Dependent Project
Documents
DSP/BIOS Config
Generated Files
Include
Libraries
Source
c6713dskinit.c
FIRcirc.c
FIRcircfunc.as
Vectors_intr.a:
C6713DSK_circ.cm
(1312.5, 9482.65)
FFT Mag
Lin
Auto Scale
Rectangular
--------------------------- FIRcirc.pjt - Debug ----------------------------
Build Complete,
0 Errors, 0 Warnings, 0 Remarks.
GEL StartUp Complete.
Build
RUNNING
Ln 25, Col 2
start
EE Programs
Lab3.pdf (application...
CCStudio_v3.1
C:\CCStudio_v3.1\h...
report [Read-Only] - ...
/C6713 DSK/CPU_1 - ...
prob1_1600 - Paint
3:26 PM

3400 Hz


/C6713 DSK/CPU_1 - C671x - Code Composer Studio - [Graphical Display]
File Edit View Project Debug GEL Option Profile Tools DSP/BIOS Window Help
FIRcirc.pjt
Debug
Files
GEL files
Projects
FIRcirc.pjt (Debug)
Dependent Project
Documents
DSP/BIOS Config
Generated Files
Include
Libraries
Source
c6713dskinit.c
FIRcirc.c
FIRcircfunc.as
Vectors_intr.a:
C6713DSK_circ.cm
(3625, 21076.2)
FFT Mag
Lin
Auto Scale
FIRcirc.pjt - Debug
Build Complete,
0 Errors, 0 Warnings, 0 Remarks.
Build
GEL StartUp Complete.
RUNNING

10/10

Problem 2:

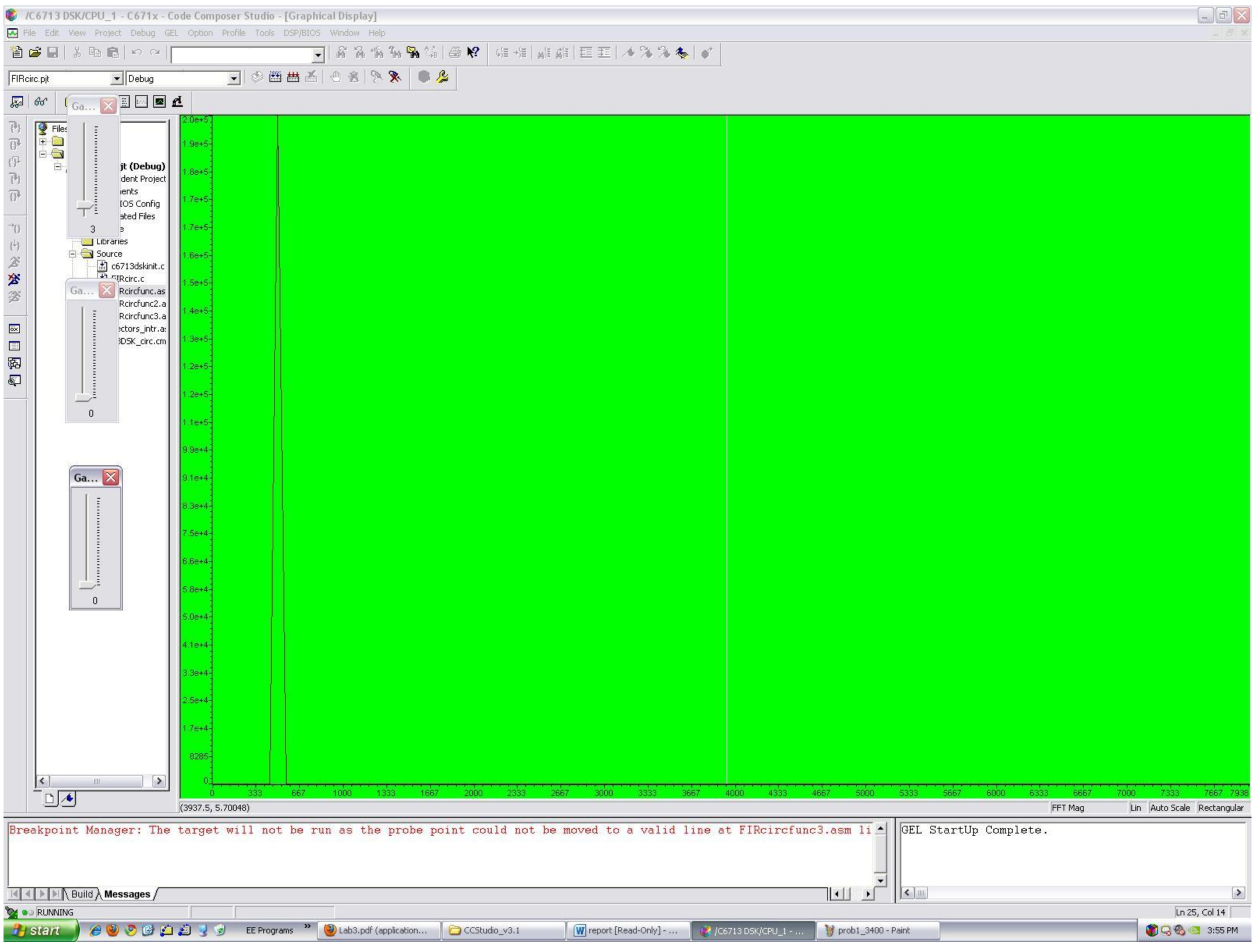
For problem 2 we designed an alarm generator based on the IIR sinusoidal generator. We used a random number generator to alternate between a 2.5 kHz signal and a 1.2kHz signal. Also, the 2.5kHz tone is played for T seconds while the 1.2kHz tone is placed for 2T seconds where T is dependent on the DIP switches. The value of T is shown in the table below for the various switches. We implemented this by DIP switch 1 having a modifier value of 1600, DIP switch 2 of 3200, and DIP switch 3 of 6400. We then added up the values and divided it by the sampling frequency (8000Hz) to get the total time. For example, if all the switches were pressed we would get a value of 12800. Dividing this by 8000 we get 1.6 seconds. To make the signals change we created a counter to count up to the computed value and grab a new random number to see if a high or low frequency would be emitted.

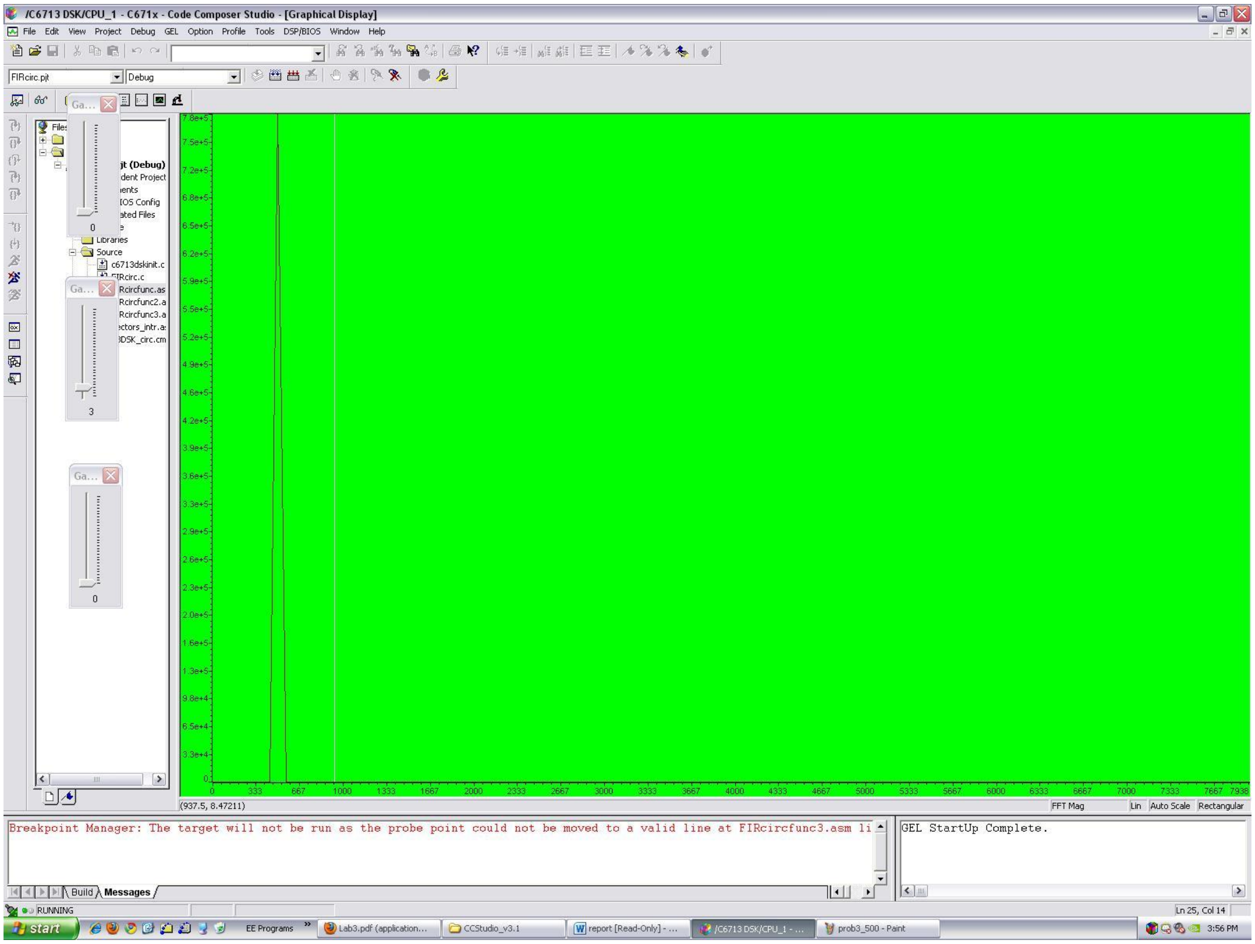
User_SW3	User_SW2	User_SW1	Value	T(second)
0	0	0	0	0.2
0	0	1	1	0.4
0	1	0	2	0.6
0	1	1	3	0.8
1	0	0	4	1.0
1	0	1	5	1.2
1	1	0	6	1.4
1	1	1	7	1.6

4/10

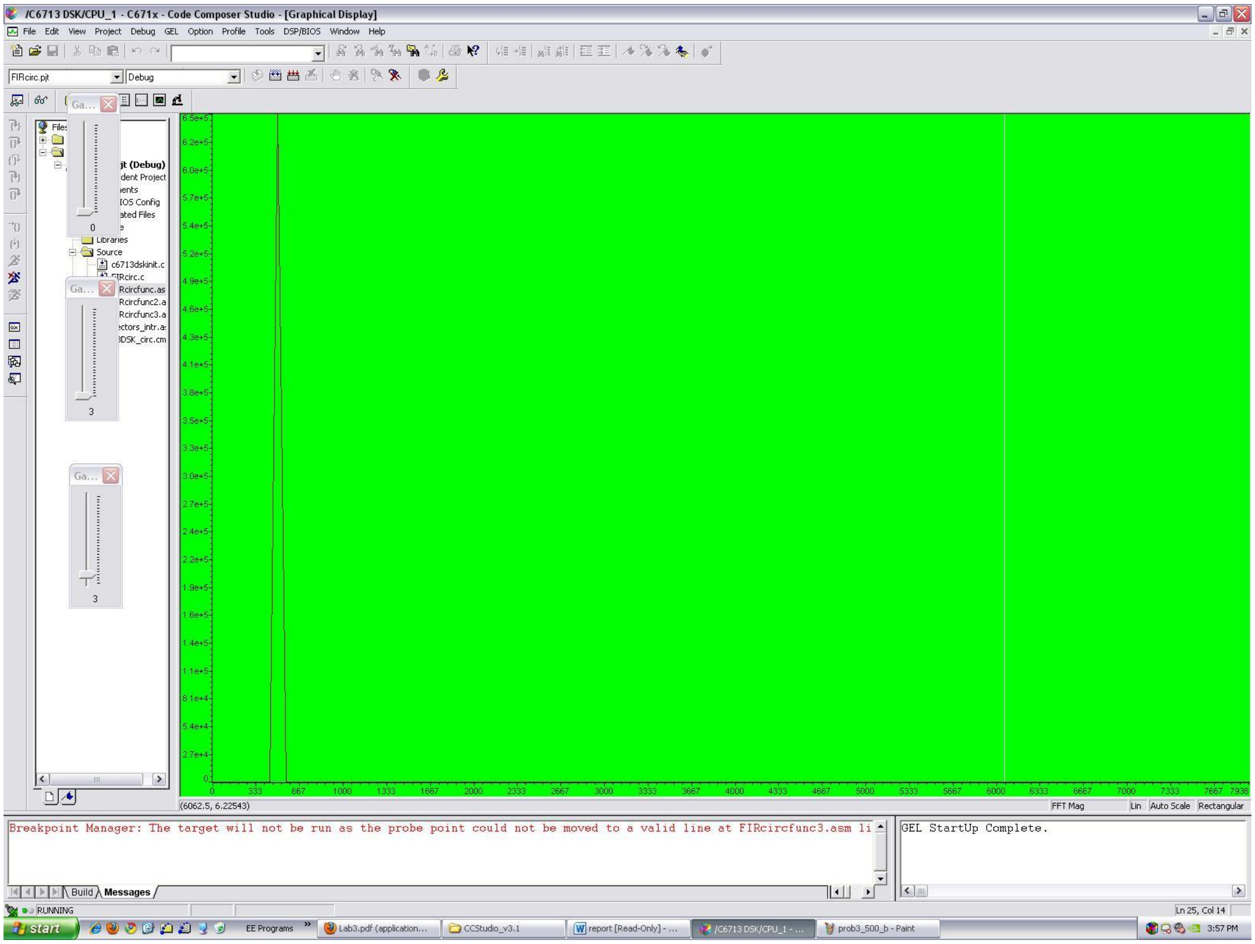
Problem 3:

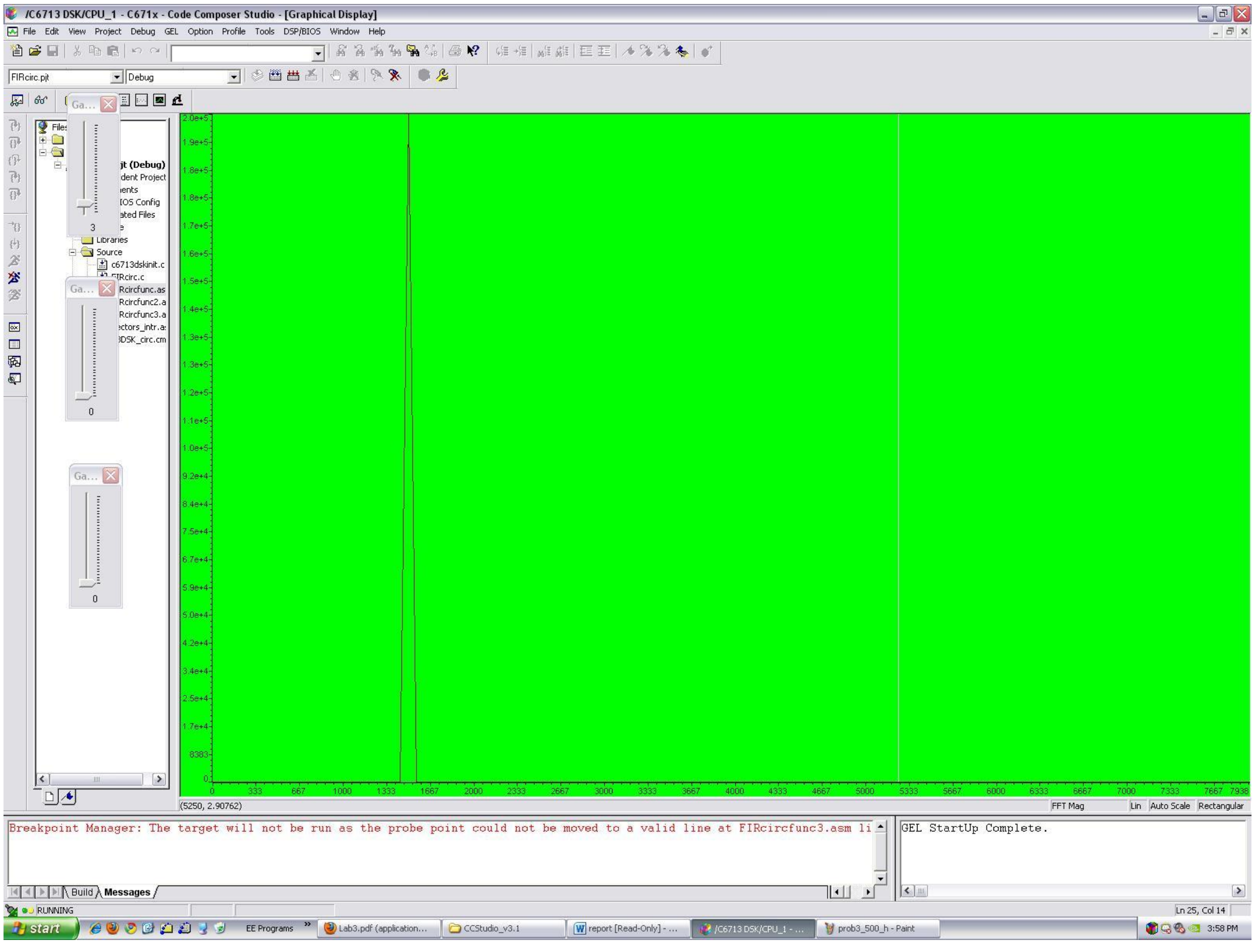
For problem 3 we want to create 3 sliders to control 3 different filters. After generating the coefficient files with Matlab we input these coefficients into three different Assembly codes. The resulting frequency domain graph shows the filtering effects of the combined filters. Our codes however do not produce any filtering effects.

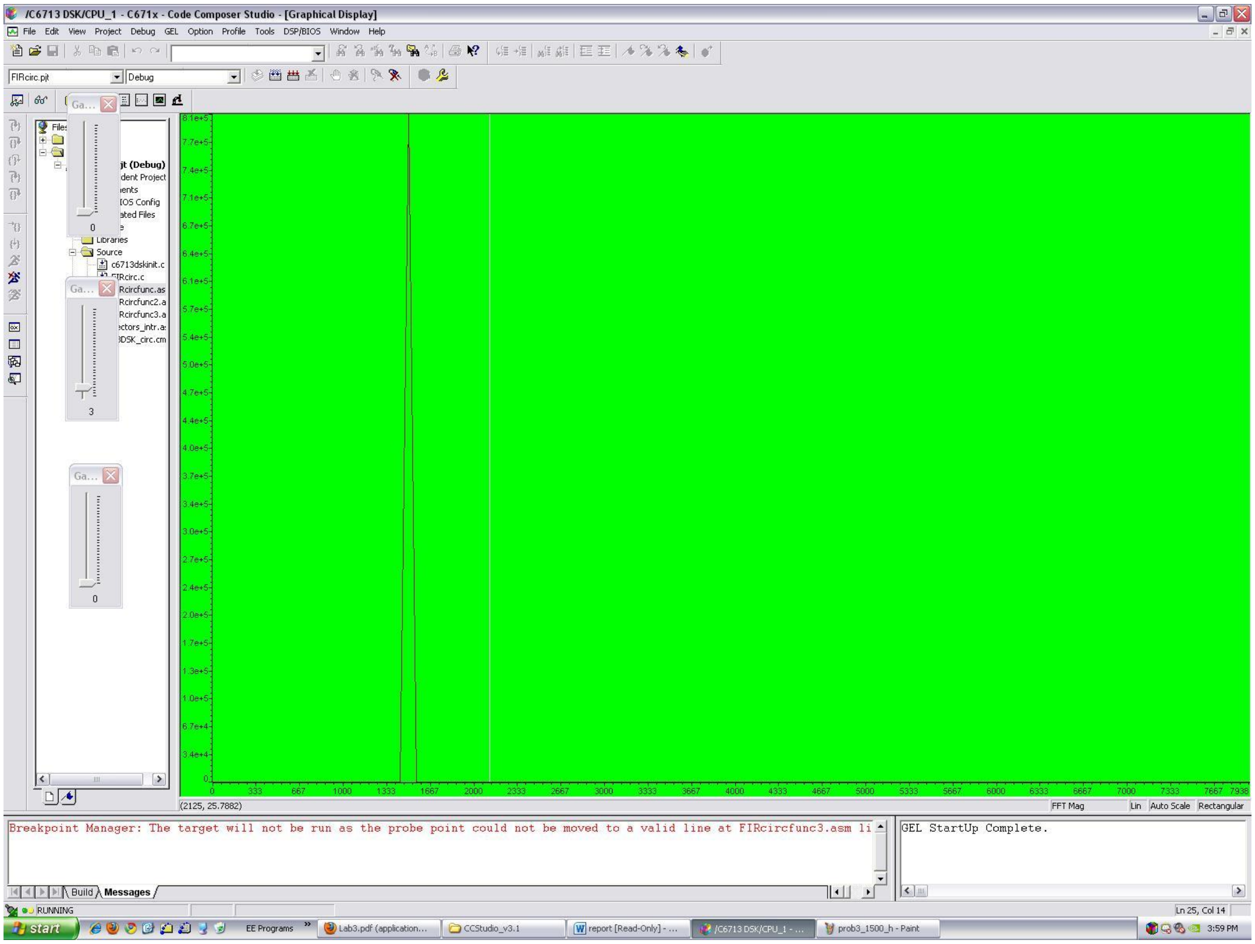




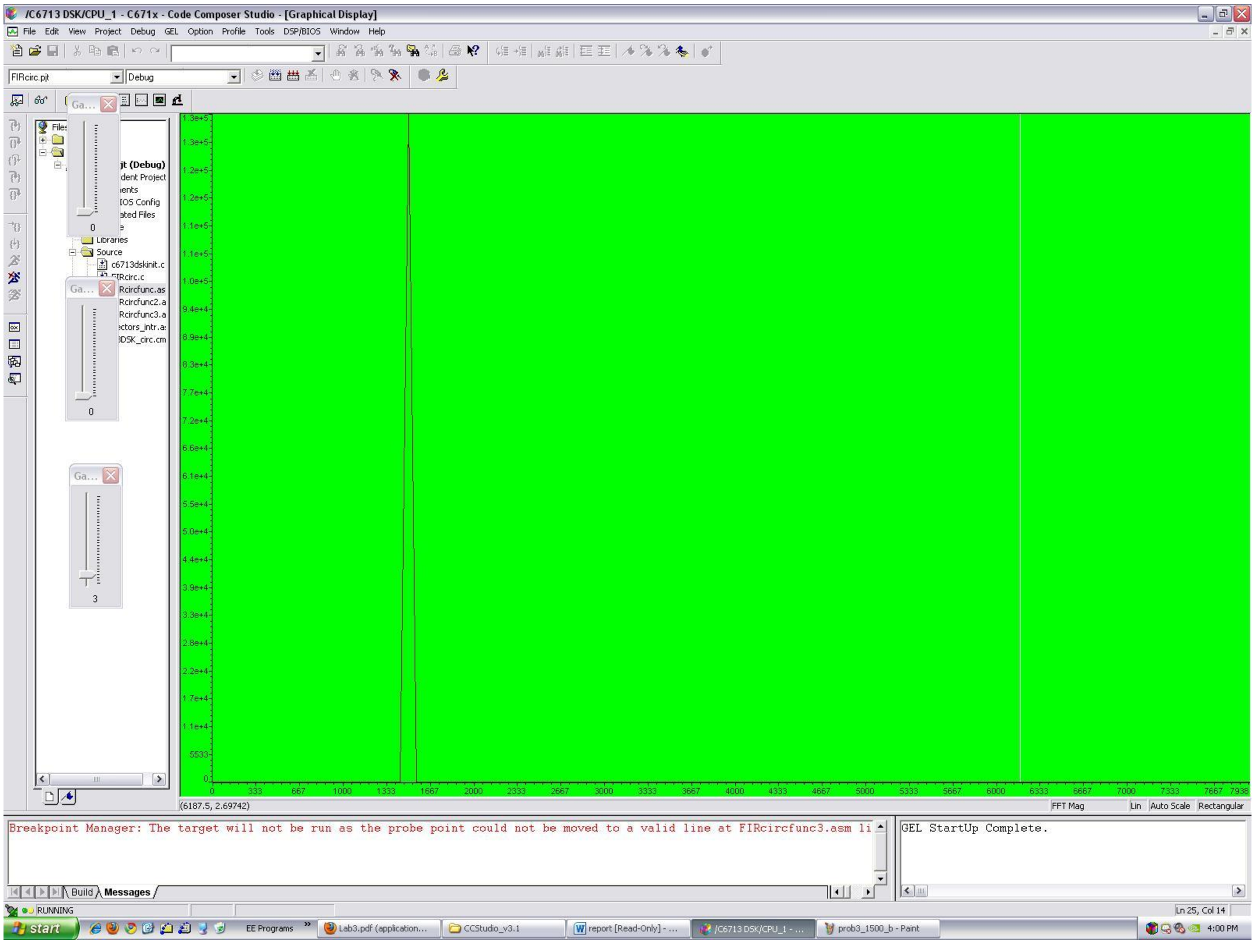
/C6713 DSK/CPU_1 - C671x - Code Composer Studio - [Graphical Display]
File Edit View Project Debug GEL Option Profile Tools DSP/BIOS Window Help
FIRcirc.pjt
Debug
(937.5, 8.47211)
FFT Mag
Lin
Auto Scale
Rectangular
Breakpoint Manager: The target will not be run as the probe point could not be moved to a valid line at FIRcircfunc3.asm li
GEL StartUp Complete.
Build
Messages
RUNNING
Ln 25, Col 14
start
EE Programs
3:56 PM

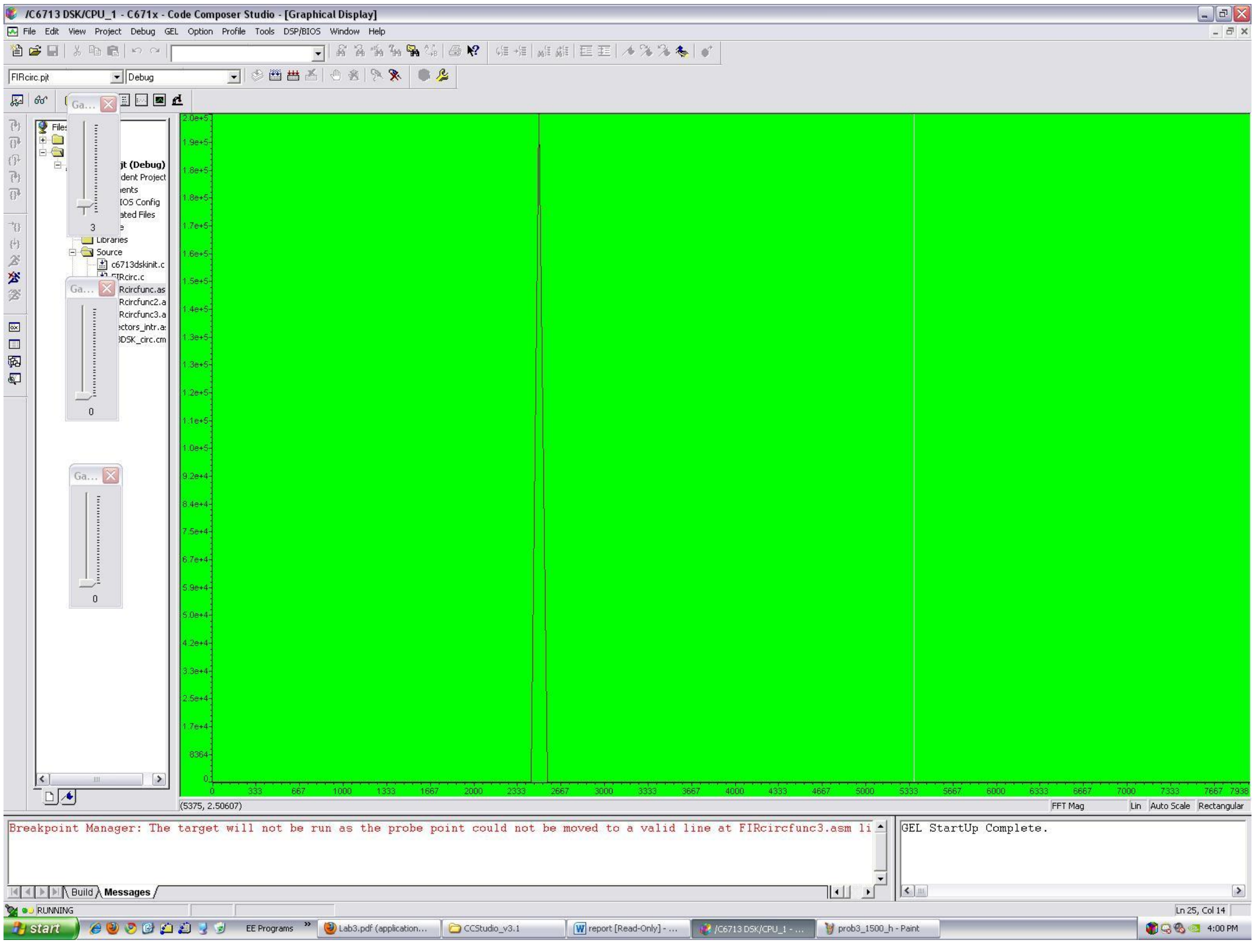

/C6713 DSK/CPU_1 - C671x - Code Composer Studio - [Graphical Display]
File Edit View Project Debug GEL Option Profile Tools DSP/BIOS Window Help
FIRcirc.pjt
Debug
(6062.5, 6.22543)
FFT Mag
Lin
Auto Scale
Rectangular
Breakpoint Manager: The target will not be run as the probe point could not be moved to a valid line at FIRcircfunc3.asm li
GEL StartUp Complete.
Build
Messages
RUNNING
Ln 25, Col 14
start
EE Programs
Lab3.pdf (application...
CCStudio_v3.1
report [Read-Only] - ...
/C6713 DSK/CPU_1 - ...
prob3_500_b - Paint
3:57 PM


/C6713 DSK/CPU_1 - C671x - Code Composer Studio - [Graphical Display]
File Edit View Project Debug GEL Option Profile Tools DSP/BIOS Window Help
FIRcirc.pjt
Debug
(5250, 2.90762)
FFT Mag
Lin
Auto Scale
Rectangular
Breakpoint Manager: The target will not be run as the probe point could not be moved to a valid line at FIRcircfunc3.asm li
GEL StartUp Complete.
Build
Messages
RUNNING
Ln 25, Col 14
start
EE Programs
Lab3.pdf (application...
CCStudio_v3.1
report [Read-Only] - ...
/C6713 DSK/CPU_1 - ...
prob3_500_h - Paint
3:58 PM

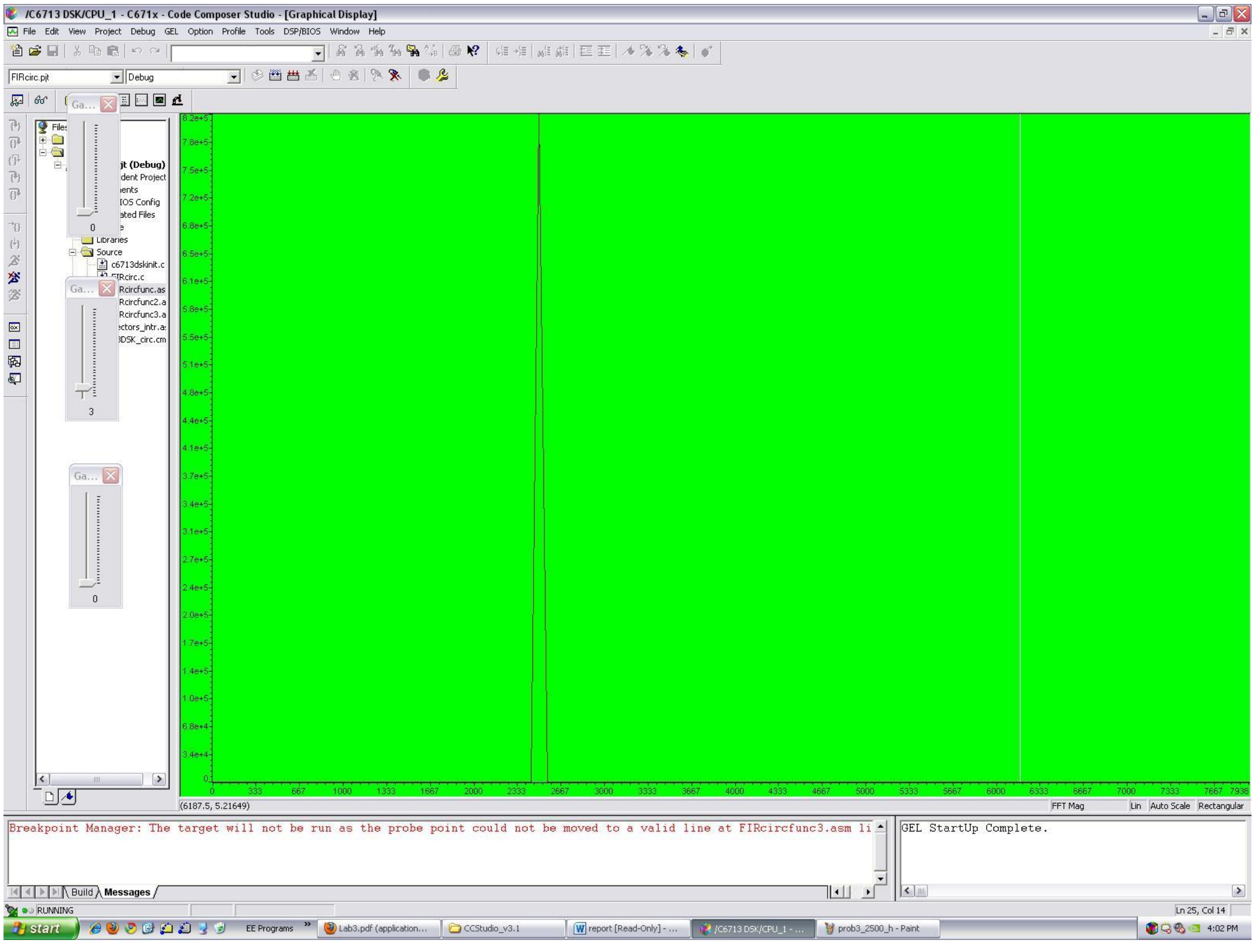


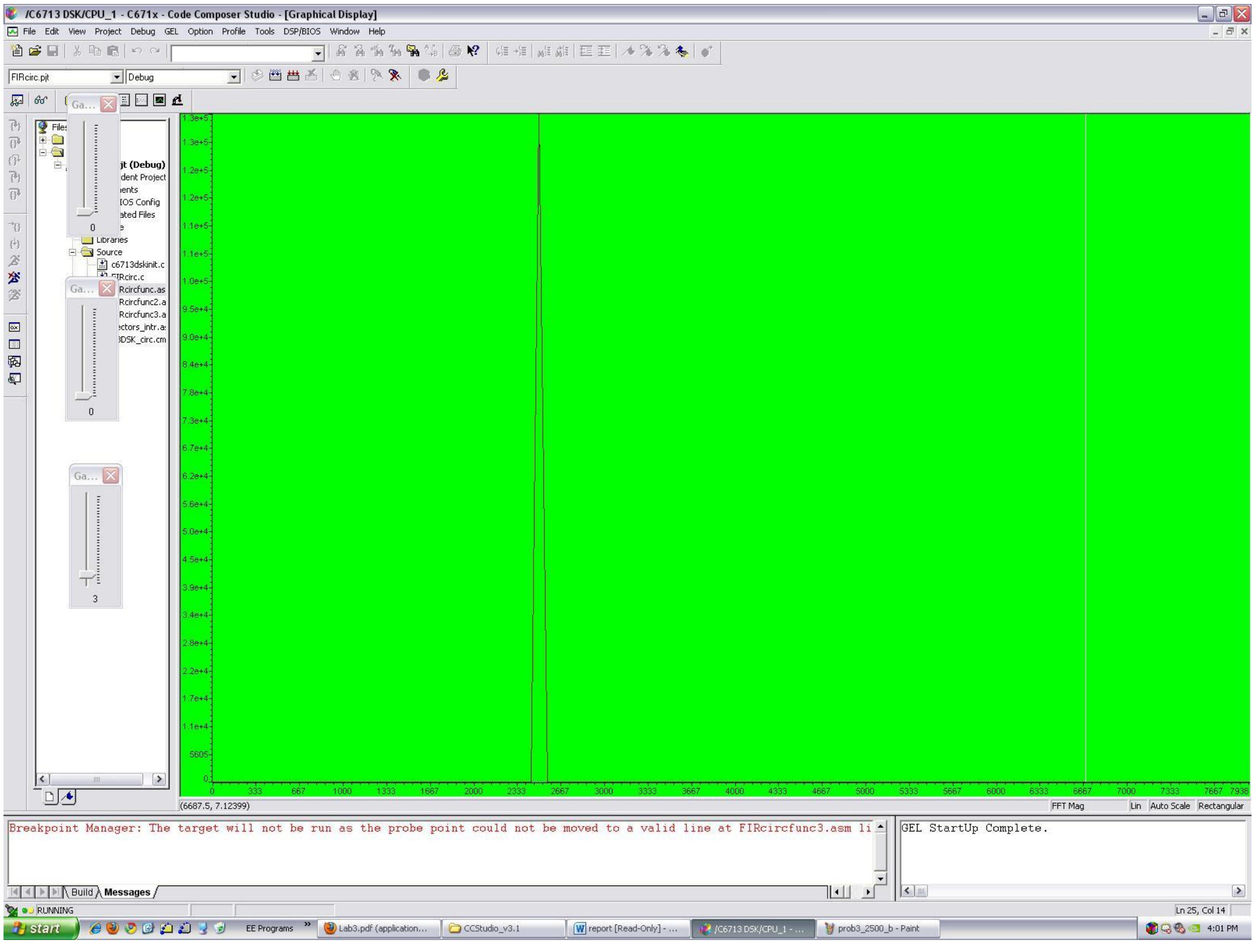
/C6713 DSK/CPU_1 - C671x - Code Composer Studio - [Graphical Display]
File Edit View Project Debug GEL Option Profile Tools DSP/BIOS Window Help
FIRcirc.pjt
Debug
(2125, 25.7882)
FFT Mag
Lin
Auto Scale
Rectangular
Breakpoint Manager: The target will not be run as the probe point could not be moved to a valid line at FIRcircfunc3.asm li
GEL StartUp Complete.
Build
Messages
RUNNING
Ln 25, Col 14
start
EE Programs
Lab3.pdf (application...
CCStudio_v3.1
report [Read-Only] - ...
/C6713 DSK/CPU_1 - ...
prob3_1500_h - Paint
3:59 PM

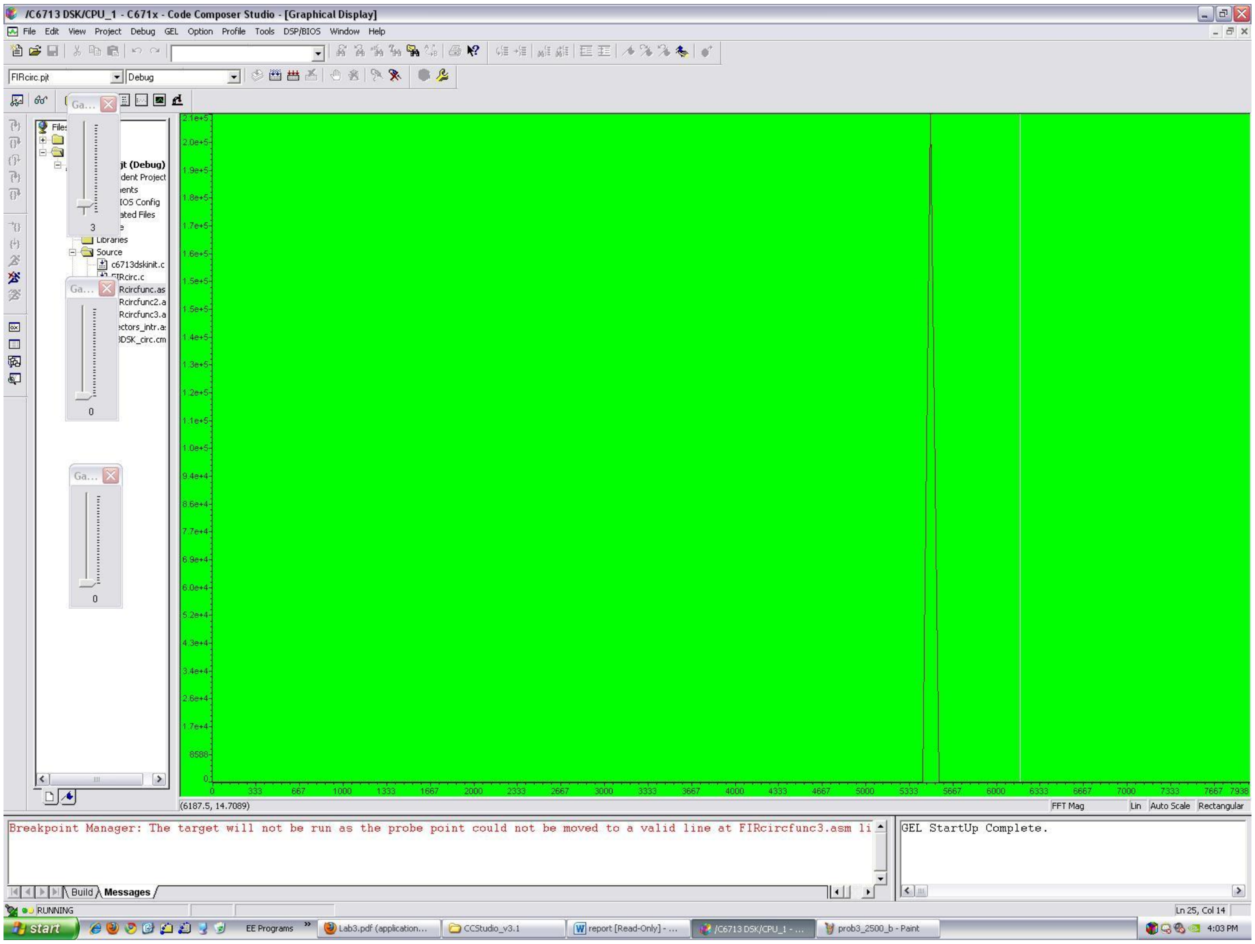

/C6713 DSK/CPU_1 - C671x - Code Composer Studio - [Graphical Display]
File Edit View Project Debug GEL Option Profile Tools DSP/BIOS Window Help
FIRcirc.pjt
Debug
(6187.5, 2.69742)
FFT Mag
Lin
Auto Scale
Rectangular
Breakpoint Manager: The target will not be run as the probe point could not be moved to a valid line at FIRcircfunc3.asm li
GEL StartUp Complete.
Build
Messages
RUNNING
Ln 25, Col 14
start
EE Programs
Lab3.pdf (application...
CCStudio_v3.1
report [Read-Only] - ...
/C6713 DSK/CPU_1 - ...
prob3_1500_b - Paint
4:00 PM



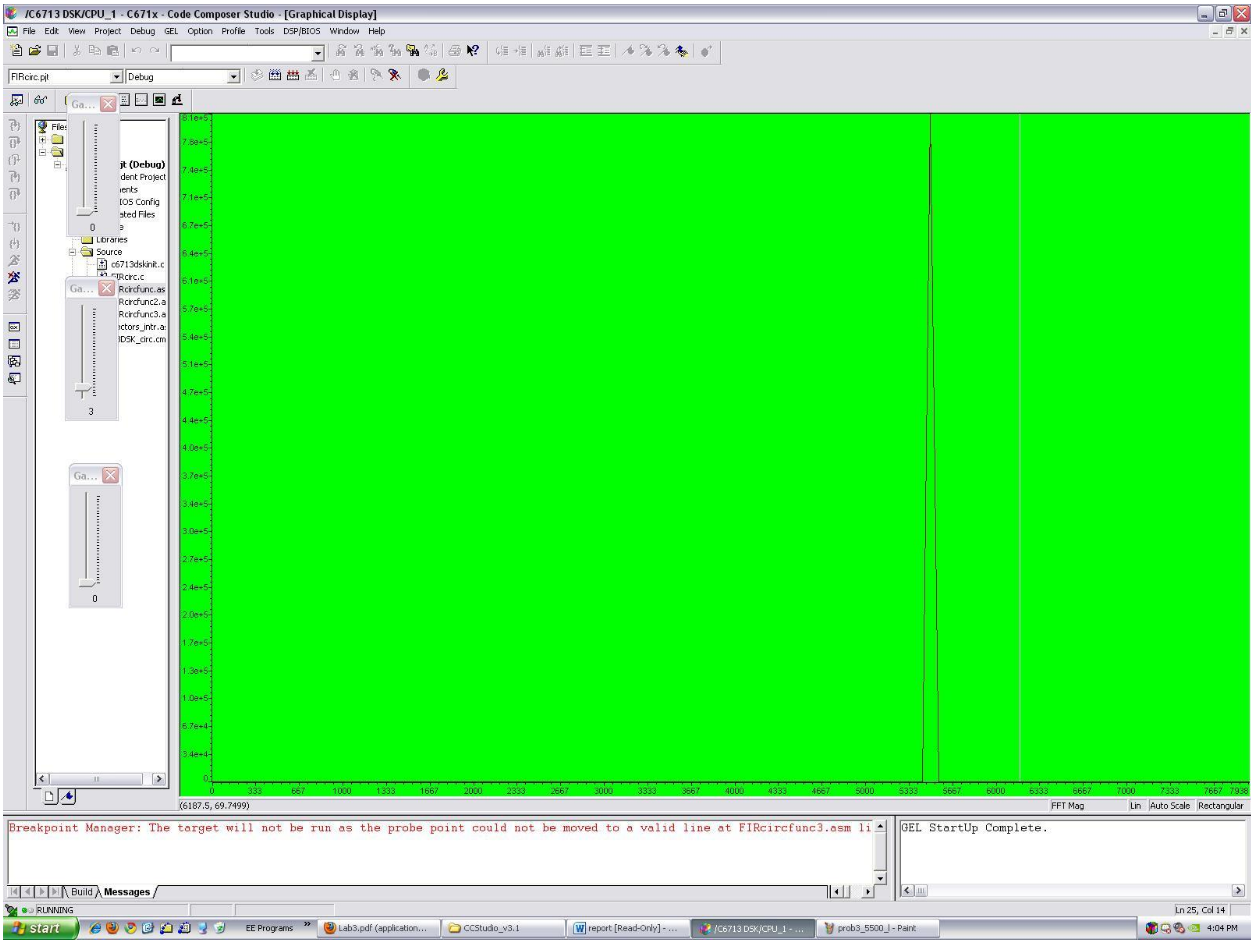
/C6713 DSK/CPU_1 - C671x - Code Composer Studio - [Graphical Display]
File Edit View Project Debug GEL Option Profile Tools DSP/BIOS Window Help
FIRcirc.pjt
Debug
Source
c6713dskinit.c
FIRcirc.c
(5375, 2.50607)
FFT Mag
Lin
Auto Scale
Rectangular
Breakpoint Manager: The target will not be run as the probe point could not be moved to a valid line at FIRcircfunc3.asm li
GEL StartUp Complete.
Build
Messages
RUNNING
Ln 25, Col 14
start
EE Programs
Lab3.pdf (application...
CCStudio_v3.1
report [Read-Only] - ...
prob3_1500_h - Paint
4:00 PM

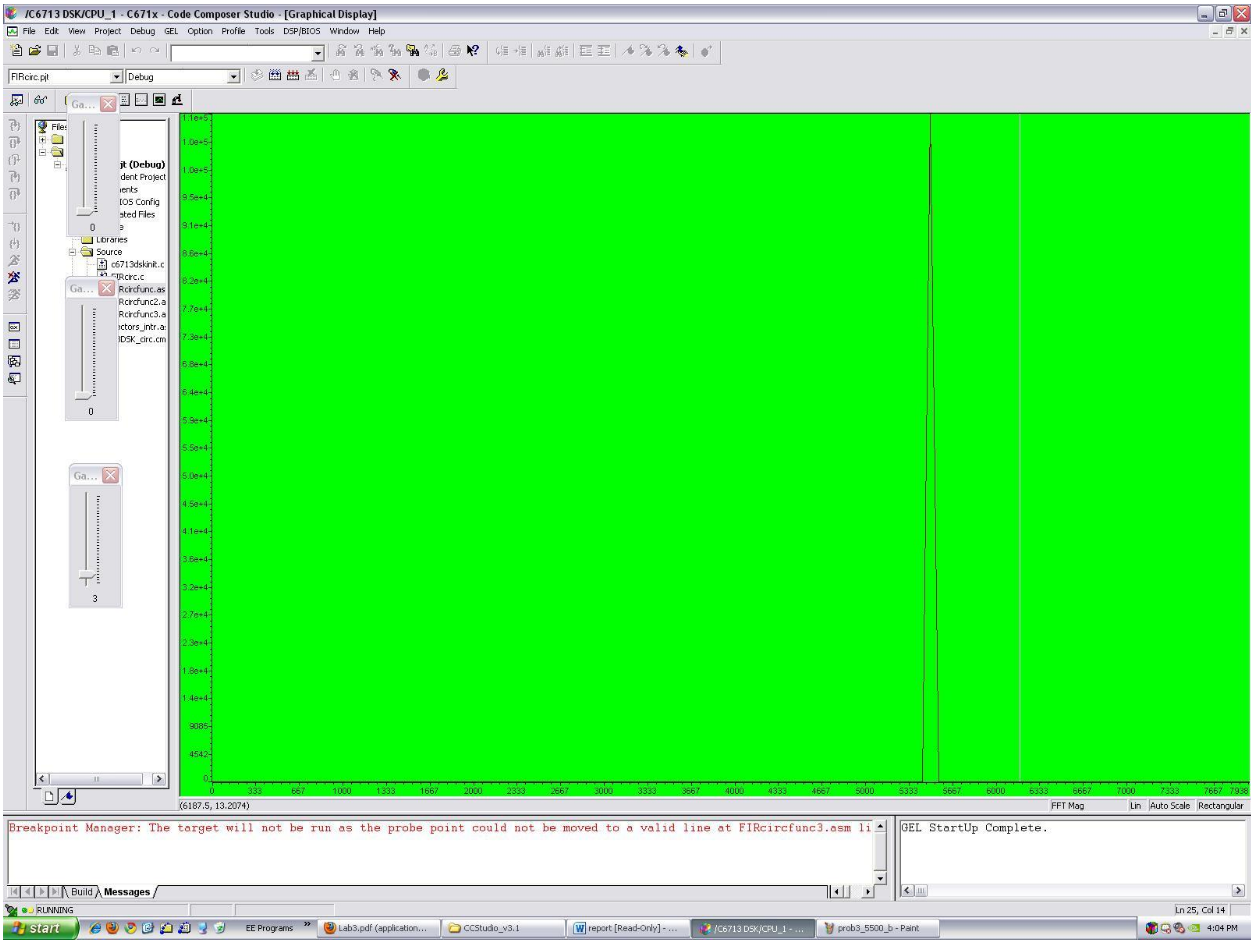

/C6713 DSK/CPU_1 - C671x - Code Composer Studio - [Graphical Display]
File Edit View Project Debug GEL Option Profile Tools DSP/BIOS Window Help
FIRcirc.pjt
Debug
(6187.5, 5.21649)
FFT Mag
Lin
Auto Scale
Rectangular
Breakpoint Manager: The target will not be run as the probe point could not be moved to a valid line at FIRcircfunc3.asm li
GEL StartUp Complete.
Build
Messages
RUNNING
Ln 25, Col 14
start
EE Programs
4:02 PM


/C6713 DSK/CPU_1 - C671x - Code Composer Studio - [Graphical Display]
File Edit View Project Debug GEL Option Profile Tools DSP/BIOS Window Help
FIRcirc.pjt
Debug
(6687.5, 7.12399)
FFT Mag
Lin
Auto Scale
Rectangular
Breakpoint Manager: The target will not be run as the probe point could not be moved to a valid line at FIRcircfunc3.asm li
GEL StartUp Complete.
Build
Messages
RUNNING
Ln 25, Col 14
start
EE Programs
4:01 PM



/C6713 DSK/CPU_1 - C671x - Code Composer Studio - [Graphical Display]
File Edit View Project Debug GEL Option Profile Tools DSP/BIOS Window Help
FIRcirc.pjt
Debug
FFT Mag
Lin
Auto Scale
Rectangular
(6187.5, 14.7089)
Breakpoint Manager: The target will not be run as the probe point could not be moved to a valid line at FIRcircfunc3.asm li
GEL StartUp Complete.
Build
Messages
RUNNING
Ln 25, Col 14
start
4:03 PM

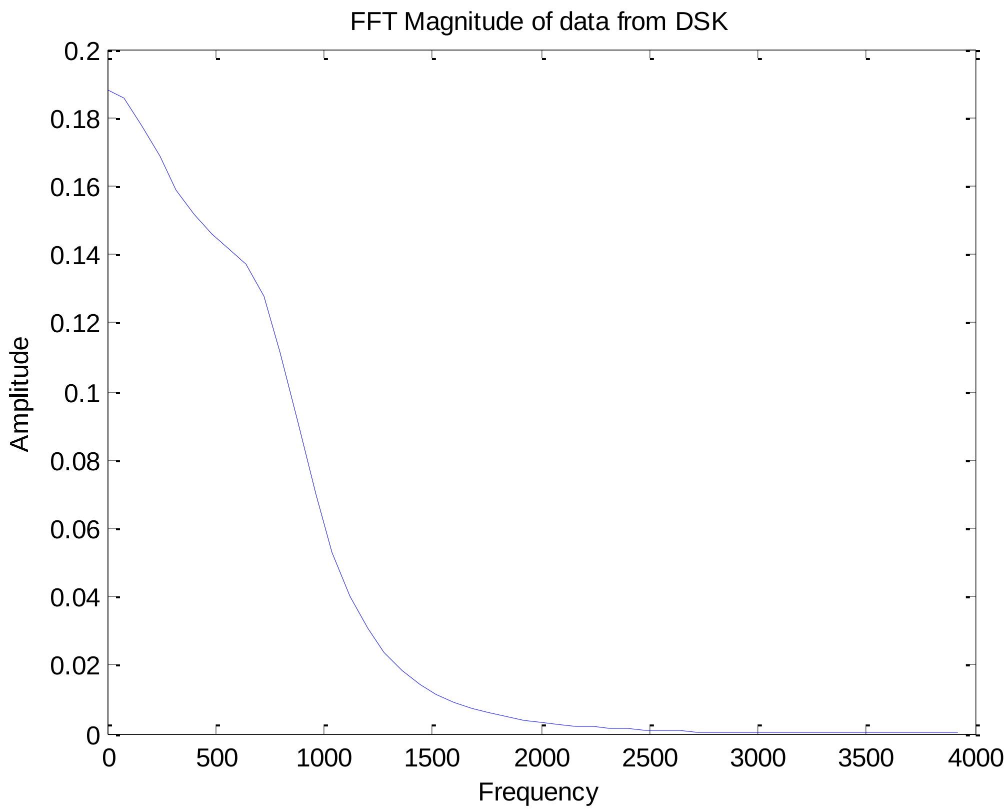

/C6713 DSK/CPU_1 - C671x - Code Composer Studio - [Graphical Display]
File Edit View Project Debug GEL Option Profile Tools DSP/BIOS Window Help
FIRcirc.pjt
Debug
(6187.5, 69.7499)
FFT Mag
Lin
Auto Scale
Rectangular
Breakpoint Manager: The target will not be run as the probe point could not be moved to a valid line at FIRcircfunc3.asm li
GEL StartUp Complete.
Build
Messages
RUNNING
Ln 25, Col 14
start
EE Programs
Lab3.pdf (application...
CCStudio_v3.1
report [Read-Only] - ...
/C6713 DSK/CPU_1 - ...
prob3_5500_l - Paint
4:04 PM



12/15

Problem 4:

For problem 4 we created a FIR adaptive filter to system identify a IIR lowpass filter we designed. After using Matlab to create the filter, we modified the given code to use the coefficients from our newly designed filter. This C program would then use RDTX to send the filter coefficients back to Matlab and plot the frequency response of it. The identified filter is shown below.



10/10

Problem 5

For problem 5 we used the overlap-and-add plut FFT to perform frame-based linear convolution using the 32 point bandpass filter designed in problem 1. To use the correct filter we simply changed the coefficient file to be the one from our designed filter. Then we changed the input to be from the mic and used convolution to apply the filter and remove part of the real time input signal.

9/15

Problem 6

For this problem we created an adaptive filter to remove an error signal as time went on. When first writing this program we created two different signals: one was the correct signal and the second the noise. As time went on, the filter would remove the noise until only the correct signal was left. Then we hooked it up to the mic and used it to filter the noise from our speech. We could also change the noise corruption using a GEL slider as well as the beta value which would change how fast the noise would be removed.

5/15

Problem 7

For Problem 7 we are programming a DTMF touch tone recognizer using the Goertzel Algorithm. The assembly code outputs the DFT values corresponding to different frequencies. We can recognize the dial tone number by observing the two frequencies that contains the largest DFT values. The program however needs to be work upon.

5/15

Problem 8

For Problem 8 we simply had to reimplement problem 3 using GUIs from Matlab instead of sliders from CCS. However, we couldn't get it working.

Problem 1 code:

```
//FIRcirc.c  C program calling ASM function using circular buffer

#include "DSK6713_AIC23.h"      //codec-DSK support file
Uint32 fs=DSK6713_AIC23_FREQ_8KHZ; //set sampling rate
#define DSK6713_AIC23_INPUT_MIC 0x0015
#define DSK6713_AIC23_INPUT_LINE 0x0011
#define bufSize 256
Uint16 inputsource=DSK6713_AIC23_INPUT_LINE; // select input source
#include "bp2000_11.cof"                    //BP at 1750 Hz coeff file
int yn = 0;                                 //init filter's output
int buffer[bufSize];
int bufIndex = 0;
interrupt void c_int11()           //ISR
{
 short sample_data;
 short output;
 sample_data = (input_left_sample());     //newest input sample data
 yn = fircircfunc(sample_data,h,N); //ASM func passing to A4,B4,A6
 output = (short)(yn>>15);
 output_left_sample(output);          //filter's output
 buffer[bufIndex] = (int)output;
 bufIndex++;
 if(bufIndex >= bufSize){bufIndex = 0;}
 return;                              //return to calling function
}

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

Problem 2 code:

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

#include "dsk6713_aic23.h"                          //codec-DSK support file
Uint32 fs=DSK6713_AIC23_FREQ_8KHZ; //set sampling rate
#define DSK6713_AIC23_INPUT_MIC 0x0015
#define DSK6713_AIC23_INPUT_LINE 0x0011
#include <math.h>
#define FREQ 2500
#define FREQ2 1200
#define SAMPLING_FREQ 8000
#define AMPLITUDE 10000
#define PI 3.14159265358979
Uint16 inputsource=DSK6713_AIC23_INPUT_LINE;

//Uint16 inputsource=DSK6713_AIC23_INPUT_MIC; // select mic in
int previous_seed;
int timer=0;
int change=1;
int sw3=0;
int sw2=0;
int sw1=0;
int temp=1;
float y[3] = {0.0, 0.0, 0.0};
float x[3] = {0.0, 0.0, 0.0};
float a1;
float b1;

interrupt void c_int11(){
        previous_seed = noisefunc(previous_seed);
        y[0] =(y[1]*a1)-y[2];
        y[2] = y[1];                    //update y1(n-2)
        y[1] = y[0];
        x[0] =(x[1]*b1)-x[2];
        x[2] = x[1];                    //update y1(n-2)
        x[1] = x[0];                    //update y1(n-1)

        if(DSK6713_DIP_get(3)==0){
        sw3=1;
        }
        else{
        sw3=0;
        }
        if(DSK6713_DIP_get(2)==0){
        sw2=1;
        }
        else{
        sw2=0;
```

```
	}
	if(DSK6713_DIP_get(1)==0){
	sw1=1;
	}
	else{
	sw1=0;
	}
	change=sw3*6400+sw2*3200+sw1*1600+1600;
	if(timer>=(change*temp)){
		if(previous_seed & 0x01){
			temp=1;
		}else{
			temp=2;
		}
		timer=0;
	}
	if (temp==1){
	output_left_sample((short)(y[0]*AMPLITUDE));//positive scaling
	timer++;
	}
	else{

	output_left_sample((short)(x[0]*AMPLITUDE));
	//negative scaling
	timer++;
	}

	return;
}

void main ()
{
	y[1] = sin(2.0*PI*FREQ/SAMPLING_FREQ);
	a1 = 2.0*cos(2.0*PI*FREQ/SAMPLING_FREQ);
	x[1] = sin(2.0*PI*FREQ2/SAMPLING_FREQ);
	b1 = 2.0*cos(2.0*PI*FREQ2/SAMPLING_FREQ);
	comm_intr();                                                   //init
DSK, codec, McBSP
	previous_seed = noisefunc(0x7E521603);  //call ASM function
  while (1);                                         //infinite loop
	//previous_seed = noisefunc(previous_seed); //call ASM function







}
```

Problem 3 code:

For the assembly code FIRcircfunc2 and FIRcircfunc3 I deleted the definition of last_addr and delays since they are defined in FIRcircfunc.

```
//FIRcirc.c  C program calling ASM function using circular buffer

#include "DSK6713_AIC23.h"     //codec-DSK support file
Uint32 fs=DSK6713_AIC23_FREQ_16KHZ;          //set sampling rate
#define DSK6713_AIC23_INPUT_MIC 0x0015
#define DSK6713_AIC23_INPUT_LINE 0x0011
#define bufSize 256
Uint16 inputsource=DSK6713_AIC23_INPUT_LINE; // select input source
#include "lp1500.cof"
#include "hp4500.cof"            //BP at 1750 Hz coeff file
#include "bp3000.cof"
int gL = 0;
int gB = 0;
int gH = 0;
int y1 = 0;
int y2 = 0;
int y3 = 0;

int yn = 0;                                        //init filter's output
int buffer[bufSize];
int bufIndex = 0;


interrupt void c_int11()         //ISR
{
 short sample_data;
 short output;
 sample_data = (input_left_sample());     //newest input sample data
 y1 = gL*fircircfunc(sample_data,h1,N1); //ASM func passing to A4,B4,A6
 y2 = gB*fircircfunc2(sample_data,h2,N2);
 y3 = gH*fircircfunc3(sample_data,h3,N3);
 yn = y1+y2+y3;
 output = (short)(yn>>15);
 output_left_sample(output);         //filter's output
 buffer[bufIndex] = (int)output;
 bufIndex++;
 if(bufIndex >= bufSize){bufIndex = 0;}
 return;                             //return to calling function
}

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

Assembly:

```
;FIRcircfunc.asm ASM function called from C using circular addressing
;A4=newest sample, B4=coefficient address, A6=filter order
;Delay samples organized: x[n-(N-1)]...x[n]; coeff as h(0)...h[N-1]

          .def  _fircircfunc2
                .sect "circdata"   ;circular data section
          .align 256        ;align delay buffer 256-byte boundary
delays      .space 256        ;init 256-byte buffer with 0's
last_addr   .int   last_addr-1 ;point to bottom of delays buffer
          .text                        ;code section
_fircircfunc2:                          ;FIR function using circ addr
          MV    A6,A1          ;setup loop count
                MPY   A6,2,A6      ;since dly buffer data as byte
                ZERO  A8            ;init A8 for accumulation
                ADD   A6,B4,B4          ;since coeff buffer data as bytes
                SUB   B4,1,B4      ;B4=bottom coeff array h[N-1]

          MVKL 0x00070040,B6 ;select A7 as pointer and BK0
          MVKH 0x00070040,B6 ;BK0 for 256 bytes (128 shorts)

          MVC   B6,AMR       ;set address mode register AMR

                MVKL  last_addr,A9  ;A9=last circ addr(lower 16 bits)
                MVKH last_addr,A9  ;last circ addr (higher 16 bits)
          LDW   *A9,A7       ;A7=last circ addr
          NOP   4
          STH   A4,*A7++     ;newest sample-->last address

loop:                                        ;begin FIR loop
          LDH   *A7++,A2     ;A2=x[n-(N-1)+i] i=0,1,...,N-1
   ||     LDH   *B4--,B2     ;B2=h[N-1-i] i=0,1,...,N-1
          SUB   A1,1,A1      ;decrement count
   [A1]   B     loop          ;branch to loop if count # 0
          NOP   2
                MPY   A2,B2,A6     ;A6=x[n-(N-1)+i]*h[N-1+i]
          NOP
          ADD   A6,A8,A8     ;accumulate in A8

          STW   A7,*A9       ;store last circ addr to last_addr
          B     B3          ;return addr to calling routine
          MV    A8,A4              ;result returned in A4
          NOP   4
```

Problem 4 code:

```
// iirsosadapt.c generic iir filter using cascaded second order sections
// characteristic identified experimentally using adaptive FIR filter
// float coefficients read from included .cof file

#include "DSK6713_AIC23.h"      //codec-DSK interface support
#include <rtdx.h>
Uint32 fs=DSK6713_AIC23_FREQ_8KHZ; //set sampling rate
RTDX_CreateOutputChannel(ochan);
#define DSK6713_AIC23_INPUT_MIC 0x0015
#define DSK6713_AIC23_INPUT_LINE 0x0011
Uint16 inputsource=DSK6713_AIC23_INPUT_LINE;

#include "noise_gen.h"
#include "cheb.cof" // contains a and b coefficient values and defines
                    // NUM_SECTIONS

float w[NUM_SECTIONS][2] = {0};

#define beta 1.0E-11                  // learning rate
#define WLENGTH 256                   //# of coeff for adaptive FIR
float h[WLENGTH+1]={0.0};                 //buffer coeff for adaptive FIR
float dly_adapt[WLENGTH+1]={0.0};            //buffer samples of adaptive FIR
short input_data[100]={0};
float input_data2[100]={0};
short k=0;
short fb;                                            //feedback variable
shift_reg sreg;

short prn(void)                               //pseudorandom noise generation
{
 short prnseq;                   //for pseudorandom sequence

 if(sreg.bt.b0)                 //sequence {1,-1}
        prnseq = -4000;                       //scaled negative noise level
 else
        prnseq = 4000;           //scaled positive noise level
 fb  =(sreg.bt.b0)^(sreg.bt.b1);   //XOR bits 0,1
 fb ^=(sreg.bt.b11)^(sreg.bt.b13); //with bits 11,13 ->fb
 sreg.regval<<=1;                //shift register 1 bit to left
 sreg.bt.b0 = fb;                     //close feedback path

 return prnseq;                              //return sequence
}



void main()
```

```
{
 //int i=0;
// int k=0;
 short i;
 short count=0;
 for(i=0;i<NUM_SECTIONS;i++){
             w[i][0]=0;
             w[i][1]=0;
             }
             i=0;
 comm_poll();
 IRQ_globalEnable();
 IRQ_nmiEnable();
 while(!RTDX_isOutputEnabled(&ochan))
             puts("\n\n Waiting... ");

 for (i = 0; i < WLENGTH; i++)
  {
   h[i] = 0.0;                                                  //init coeff of adaptive FIR
   dly_adapt[i] = 0.0;              //init samples of adaptive FIR
  }
  sreg.regval = 0xFFFF;       //shift register to nominal values
  fb = 1;                    //initial feedback value
  //comm_poll();               //init DSK, codec, McBSP
  while(1){
  int section=0;   // index for section number
  float input;   // input to each section
  float wn,yn;   // intermediate and output values in each stage
  int i,temp1,temp2;

  float adaptfir_out=0.0;                             //init output of adaptive FIR
  float E;                                                       //error signal
temp1=0;
temp2=0;
  input = (float)(prn()); // input to first iir section is PRBS value

  w[section][0]=0;
  dly_adapt[0] = input;   // copy input value to fir

  for (section=0 ; section< NUM_SECTIONS ; section++)
  {
    //w[section][0]=temp1;
//          w[section][1] =temp2;
    wn = input - a[section][0]*w[section][0] - a[section][1]*w[section][1];
    //temp1=wn;
    yn = b[section][0]*wn + b[section][1]*w[section][0] + b[section][2]*w[section][1];
    w[section][1] = w[section][0];
             //temp2=w[section][0];
```

```
  w[section][0] = wn;
  input = yn;              // output of current section will be input to next
 }

 //yn = yn * 10000;    // finally, scale output by codec attenuation factor

 for (i = 0; i < WLENGTH; i++) adaptfir_out +=(h[i]*dly_adapt[i]); //output of adaptive FIR
 E = yn - adaptfir_out;                  //error as difference of outputs
 for (i = WLENGTH-1; i >= 0; i--)
 {
  h[i] = h[i]+(beta*E*dly_adapt[i]);  //update weights of adaptive FIR
  dly_adapt[i+1] = dly_adapt[i];      //update samples of adaptive FIR
 }
        i=0;
        count++;
        if(count>=100){
        while(i<100){
        input_data2[i]=h[i];
        input_data[i]=(double)h[i];
        k++;
        //i++;
        output_left_sample(input_data[i++]);
        }
 RTDX_write(&ochan,input_data2,sizeof(input_data2));
 count=0;
 }
 //output_left_sample((short)adaptfir_out);
            //return from ISR
}
 }
```

Problem 5 code
//fastconv.c

```
#include "DSK6713_AIC23.h"      //codec-DSK interface 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_MIC; // select input source

#include "prob5.cof"

#include <math.h>
#include "fft.h"
#define PI 3.14159265358979
#define PTS 256

short buffercount = 0;    // index into frames
short bufferfull=0;
COMPLEX A[PTS], B[PTS], C[PTS];   //three buffers used
COMPLEX twiddle[PTS];             //twiddle factors
COMPLEX coeffs[PTS];              //zero padded freq domain filter coeffs
COMPLEX *input_ptr, *output_ptr, *process_ptr, *temp_ptr; //pointers to frames
float a,b;                        //variables used in complex multiply

interrupt void c_int11(void)      //ISR
{
  output_sample((short)((output_ptr + buffercount)->real)/10);
  (input_ptr + buffercount)->real = (float)(input_left_sample());
  (input_ptr + buffercount++)->imag = 0.0;
  if (buffercount >= PTS/2)
  {
    bufferfull = 1;
    buffercount = 0;
  }
}

void main()
{
  int n,i;
  for (n=0 ; n<PTS ; n++)         //set up twiddle factors in array w
  {
    twiddle[n].real = cos(PI*n/PTS);
    twiddle[n].imag = -sin(PI*n/PTS);
  }
  for (n=0 ; n<PTS ; n++)         //set up complex freq domain filter coeffs
  {
    coeffs[n].real = 0.0;
    coeffs[n].imag = 0.0;
```

```
  }
  for (n=0 ; n<N ; n++)
  {
    coeffs[n].real = h[n];
  }
  fft(coeffs,PTS,twiddle);             //transform filter coeffs to freq domain
  input_ptr = A;  //initialise pointers to frames/buffers
  process_ptr = B;
  output_ptr = C;
  comm_intr();


  while(1)                      //frame processing loop
  {
    while (bufferfull == 0);          //wait for iobuffer full
    bufferfull = 0;
    temp_ptr = process_ptr;
    process_ptr = input_ptr;
    input_ptr = output_ptr;
    output_ptr = temp_ptr;

    for (i=0 ; i< PTS ; i++) (process_ptr + i)->imag = 0.0;
    for (i=PTS/2 ; i< PTS ; i++) (process_ptr + i)->real = 0.0;

    fft(process_ptr,PTS,twiddle);          //transform samples into frequency domain

    for (i=0 ; i<PTS ; i++)      //filter frequency domain representation
    {                          //i.e. complex multiply samples by coeffs
      a = (process_ptr + i)->real;
      b = (process_ptr + i)->imag;
      (process_ptr + i)->real = coeffs[i].real*a - coeffs[i].imag*b;
      (process_ptr + i)->imag = -(coeffs[i].real*b + coeffs[i].imag*a);
    }
    fft(process_ptr,PTS,twiddle);
    for (i=0 ; i<PTS ; i++)
    {
      (process_ptr + i)->real /= PTS;
      (process_ptr + i)->imag /= -PTS; //if result is real, do we need this?
    }
    for (i=0 ; i<PTS/2 ; i++) //overlap add (real part only!!)
    {
      (process_ptr + i)->real += (output_ptr + i + PTS/2)->real;
    }

  } // end of while
}                              //end of main()
```

Problem 6 code:

```
//Adaptnoise_2IN.c  Adaptive FIR for sinusoidal noise interference
#include "DSK6713_AIC23.h"                  //codec-DSK support file
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_MIC;
#define beta 1E-12          //rate of convergence
#include <math.h>
#define N 30               //# of weights (coefficients)
#define LEFT 0
//#define betad 1                                       //left channel
#define RIGHT 1                                     //right channel
#define FREQ 1500
#define FREQ2 2000
#define SAMPLING_FREQ 8000
#define PI 3.14159265358979
float w[N];                //weights for adapt filter
float delay[N];              //input buffer to adapt filter
short output;
int gain;
int noisegain;                          //overall output
short out_type = 1;           //output type for slider
float y[3] = {0.0, 0.0, 0.0};
float x[3] = {0.0, 0.0, 0.0};
float a1;
float b1;
short betad=0;
short o=0;
short amplitude=0;
//volatile union{unsigned int uint; short channel[2];}AIC23_data;
interrupt void c_int11()                //ISR
 {
  short i;
  float a;
  float yn=0, E=0, dplusn=0, desired=0, noise=0;
  y[0] =(y[1]*a1)-y[2];
        y[2] = y[1];                        //update y1(n-2)
        y[1] = y[0];
        x[0] =(x[1]*b1)-x[2];
        x[2] = x[1];                        //update y1(n-2)
        x[1] = x[0];
        gain=pow(10,betad);
        noisegain = pow(10,amplitude);
  //AIC23_data.uint = input_sample(); //input 32-bit from both channels
  //desired =(AIC23_data.channel[LEFT]);//input left channel
  //noise = (AIC23_data.channel[RIGHT]); //input right channel
```

```
  a=input_left_sample();
  desired = (float)(input_left_sample());
  //desired = y[0];
  noise = x[0];
  if(o>=3){
          o=0;
          }
  dplusn = a + noise*amplitude;        //desired+noise
  delay[0] = noise*amplitude;                          //noise as input to adapt FIR

  for (i = 0; i < N; i++)         //to calculate out of adapt FIR
          yn += (w[i] * delay[i]);      //output of adaptive filter
  E = (a + noise*amplitude) - yn;      //"error" signal=(d+n)-yn
  for (i = N-1; i >= 0; i--)      //to update weights and delays
   {
          w[i] = w[i] + beta*E*delay[i]*gain; //update weights
          delay[i] = delay[i-1];      //update delay samples
   }
  if(out_type == 1)                                        //if slider in position 1
          output=((short)E);          //error signal as overall output
  else if(out_type==2)  //if slider in position 2
          output=((short)dplusn);//output (desired+noise)
  output_left_sample((short)(E*1000));               //overall output result
  return;
}

 void main()
{
  short T=0;
  float test=0.0;
  test=FREQ;
  test=FREQ/SAMPLING_FREQ;
  test=test*2;
  test=test*PI;
  y[1] = sin(2.0*PI*FREQ/SAMPLING_FREQ);
          a1 = 2.0*cos(2.0*PI*FREQ/SAMPLING_FREQ);
          x[1] = sin(2.0*PI*FREQ2/SAMPLING_FREQ);
          b1 = 2.0*cos(2.0*PI*FREQ2/SAMPLING_FREQ);
  for (T = 0; T < 30; T++)
   {
          w[T] = 0;                                          //init buffer for weights
          delay[T] = 0;                              //init buffer for delay samples
   }
  comm_intr();                 //init DSK, codec, McBSP
  while(1);                    //infinite loop
}
```

Problem 7 code:

```
//Goertzel Implementation

#include "DSK6713_AIC23.h"                  //codec-DSK support file
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;
#define bufSize 256
#define coeffSize 8
float buffer[bufSize];
float coeffs[coeffSize] = {1.703275,1.635585,1.562297,1.482867,1.163138, 1.008835, 0.790074,
0.559454};
float mags[coeffSize];
int freqs[coeffSize] = {697, 770, 852, 941, 1209, 1336, 1477, 1633};
int gzing = 1;
int bufIndex = 0;
int coeffIndex = 0;
float gz(float* ,float, int);

interrupt void c_int11()        //interrupt service routine
{

        buffer[bufIndex] =  (float) input_left_sample();
        bufIndex++;
        if(bufIndex>=bufSize){
                if(gzing ==1){
                        mags[coeffIndex] = gz(buffer,coeffs[coeffIndex],bufSize);
                        coeffIndex++;
                        gzing = 0;
                        if(coeffIndex >= coeffSize){coeffIndex = 0;}
                }else{gzing = 1;}
                bufIndex = 0;
        }
        return;
}

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

Assembly:

```
        .def _gz
                ; A1 count
                ; A4 input          B4 coeff
                ; A3 temporary input x
```

```
            ; A5 delay1       B5 delay2
            ; A7 prod1        B7 prod2         A9 prod3 temporary
            ; A8 sum1         B8 sum2 temporary
            ; B6 final Q(N) value
_gz:

            ZERO    A5
            ZERO    B5
            MV A6,A1 ;Use A1 as the main counter.
            MV A4,A0          ;Use A0 to iterate through the buffer

LOOP:   SUB A1,1,A1
            LDW     *A0++,A3          ;A11 is just a temp space to store values.
            NOP 4
            MPYSP A5, B4, A7 ; prod1
            NOP 3
            SUBSP A3, B5, A8 ; sum1
            NOP 3
            MV A5, B5
        [A1] B   LOOP
            ADDSP A8, A7, A5  ; delay1
            NOP 4

            MPYSP B5, B5, B7
            NOP 3
            MPYSP A5, A5, A7
            NOP 3
            ADDSP A7, B7, A8
            NOP 3
            MPYSP A5, B4, A9
            NOP 3
            MPYSP A9, B5, A9
            NOP 3
            SUBSP A8, A9, A8
            NOP 3
            MV A8, A4
        .end
```

Problem 8 code:

```
//FIR3LP_RTDX.c FIR of 3 LP with different BWs using RTDX with MATLAB

#include "lp600.cof"                        //coeff file LP @ 600 Hz
#include <rtdx.h>
#include <stdio.h>
#include "target.h"
#include "dsk6713_aic23.h"
Uint32 fs=DSK6713_AIC23_FREQ_16KHZ;
#define DSK6713_AIC23_INPUT_MIC 0x0015
#define DSK6713_AIC23_INPUT_LINE 0x0011
Uint16 inputsource=DSK6713_AIC23_INPUT_LINE;
#include "lp1500.cof"
#include "hp4500.cof"          //BP at 1750 Hz coeff file
#include "bp3000.cof"
#define bufSize 256
int yn = 0;                              //initialize filter's output
short dly[N];         //delay samples
short h[N];                         //filter characteristics 1xN
short loop = 0;
short sine_table[32] = {0,195,383,556,707,831,924,981,1000,
                                981,924,831,707,556,383,195,0,
                            -195,-383,-556,-707,-831,-924,-981,-1000,
                            -981,-924,-831,-707,-556,-383,-195};
short amplitude = 10;
#define BUFFER_SIZE 256
int buffer[BUFFER_SIZE];
int inputsample;
int outputsample;
short j = 0;
short temp=0;
int gL = 0;
int gB = 0;
int gH = 0;
int y1 = 0;
int y2 = 0;
int y3 = 0;
int buffer2[bufSize];
int bufIndex = 0;

RTDX_CreateInputChannel(ichan);                //create input channel
RTDX_CreateOutputChannel(ochan);               //create output channel

interrupt void c_int11()         //ISR
{
 short sample_data;
 short output;
 sample_data = (input_left_sample());     //newest input sample data
```

```
 y1 = gL*fircircfunc(sample_data,h1,N1); //ASM func passing to A4,B4,A6
 y2 = gB*fircircfunc(sample_data,h2,N2);
 y3 = gH*fircircfunc(sample_data,h3,N3);
 yn = y1+y2+y3;
 output = (short)(yn>>15);
 output_left_sample(output);          //filter's output
 buffer[bufIndex] = (int)output;
 bufIndex++;
 if(bufIndex >= bufSize){bufIndex = 0;}
 return;                              //return to calling function
}

void main()
{

 short i;
comm_intr();
 TARGET_INITIALIZE();
 RTDX_enableInput(&ichan);                  //enable RTDX channel
 RTDX_enableOutput(&ochan);                 //enable RTDX channel
 for (i=0; i<N; i++)
  {
        dly[i] = 0;             //init buffer
        h[i] = hlp600[i];                   //start addr of LP600 coeff
  }
 while(temp<256){
        buffer[temp]=sine_table[temp%32];
        temp++;
        }
 while(1)                                   //infinite loop
  {
   inputsample=rand()+amplitude*(sine_table[loop]);//gen input
   if (loop < 31) ++loop;
   else loop = 0;
   //FIR filter section
   dly[0] = inputsample;                    //newest input @ top of buffer
   yn = 0;                  //initialize filter output
   if (!RTDX_channelBusy(&ichan))  {
                RTDX_readNB(&ichan,&h[0],N*sizeof(short));}
   j++;
   if (j==BUFFER_SIZE) {
        j = 0;
          while (RTDX_writing != NULL) {}  //waiting for rtdx write to complete
                RTDX_write( &ochan, &buffer[0], BUFFER_SIZE*sizeof(int) );
   }

  }
}
```