Appendix: Software Source Code

I. GPIB Control Interface (C++ source code)

1) globals.h

```
//Contains all global structures, error codes, etc.


#ifndef __GLOBALS_H__
#define __GLOBALS_H__



struct LC9400Descriptor
{
	float FVGain;				//fixed vertical gain
	float VVGain;				//variable vertical gain
	float VOffset;				//vertical offset
	int ICoupling;			//input channel coupling at time of acquisition
	BOOL BWLimit;				//bandwidth limit at aquisition
	float TBase;				//timebase at acquisition
	float SInterval;			//8-bit sampling interval at acquisition
	UINT NDiv;					//number of measured data points per
division
	UINT PType;				//data processing of this record
};


#endif
```

2) control.cpp

```
// Contains routines to handle all interaction with GPIB Devices
//
// Author:  Noah W. Cushing
//
// Revision History:
//                      9/17/97 -- File created


#include "stdafx.h"
#include "LC9400.h"
#include "globals.h"
#include "FetchProgressDlg.h"
#include "math.h"


UINT Control(CEdit *WaveName, CEdit *CurAngle, CEdit *Message, CProgressCtrl *WriteProgress,
char *OutputFileName, UINT ChanOrMem, int Start, int Inc)
{
        //initialize variables
        UINT i;
        int temp = 0;
        static int cur_angle = Start;
        static int num = 0;
        static char OrigOutputName[128];
        unsigned char Desc[154];
        unsigned char Data[25004];
        float Voltage[25004];
        float SampleInterval;                   //time between samples

        FILE *OutputFile;
        FILE *DescOut;

        UINT Sizes[2] = {153, 25001};//set sizes of our arrays   {descriptor, max_data_size}

        //char DevName[] = {"DSO9400"};
        char DevName[] = {"GPIB1"};
        char MessageString[512];
        char DescOutName[128];
        char Temp[128];
        char Temp2[128];

        CLC9400 LC9400(ChanOrMem);
        LC9400Descriptor *WDesc;


        //////// END DATA DECLARATION ////////


        //Check for angle and name reset command//
        if(Start == -1)
        {
                WaveName->SetWindowText("c:\\data\\wave.dat");
                Message->SetWindowText("Resetting Variables...");

                strcpy(OrigOutputName, "c:\\data\\wave");
```

```
		num = 0;

		return(TRUE);
}

for(i=0; i < Sizes[0]; i++)
		Desc[i] = 0;

for(i=0; i < Sizes[1]; i++)
		Data[i] = 0;


//get waveform descriptor and data

LC9400.GetDD(Desc, Data, Sizes, 1);

//parse waveform descriptor
WDesc = LC9400.GetWDesc();


if(num > 0)
{
		strcpy(Temp, OrigOutputName);

		strcat(Temp, itoa(num+1, Temp2, 10));

		strcpy(OutputFileName, strcat(Temp, ".dat"));

		WaveName->SetWindowText(OutputFileName);
}


else
{
		temp = strlen(OutputFileName) - 4;
		strncpy(OrigOutputName, OutputFileName, temp);
}


if(Inc == 0)
		num = 0;


char AngleBuffer[20];

if(num == 0)
		cur_angle = Start;
else
		cur_angle = cur_angle + Inc;


CurAngle->SetWindowText(itoa(cur_angle, AngleBuffer, 10));

////////// Open Files //////////
sprintf(DescOutName, "%s.desc", OutputFileName);
```

```
	if((DescOut = fopen(DescOutName, "w")) <= 0)
	{
		sprintf(MessageString, "Unable to open file <%s> for writing!    Aborting run.",
DescOutName);
		MessageBox(GetActiveWindow(), MessageString, "Error", MB_OK);
		return(FALSE);
	}


	if((OutputFile = fopen(OutputFileName, "w")) <= 0)
	{
		fclose(DescOut);
		sprintf(MessageString, "Unable to open file <%s> for writing!    Aborting run.",
OutputFileName);
		MessageBox(GetActiveWindow(), MessageString, "Error", MB_OK);
		return(FALSE);
	}


	//// Output Descriptor ////
	SampleInterval = (float)WDesc->TBase / (float)WDesc->NDiv;

	fprintf(DescOut, "%f\n%d\n%d\n", log10(SampleInterval), WDesc->NDiv, cur_angle);


	//// Output Wave ////
	for(i = 0; i < WDesc->NDiv * 10; i++)
	{
		Voltage[i] = WDesc->FVGain * ((Data[i] - 128.0)/32.0 - (WDesc->VOffset -
200.0)/25.0) * 200.0/(WDesc->VVGain + 80.0);

		fprintf(OutputFile, "%f\n", Voltage[i]);

		WriteProgress->SetPos((int)(100 * ((float)i / (float)(WDesc->NDiv * 10))));
	}

	fclose(DescOut);
	fclose(OutputFile);

	Message->SetWindowText("Data Acquisition Complete!");

	int result = 1;			//dummy

	num++;

	return(TRUE);
}


////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
UINT SControl(CEdit *WaveName, CEdit *Message, CProgressCtrl *WriteProgress, char
*OutputFileName, UINT NumSweeps, int Start)
{
	//initialize variables
```

```
UINT i, count = 0;
int temp = 0;
unsigned char Desc[154];
unsigned char Data[32000];
float Voltage[25004];
FILE *OutputFile;
FILE *DescOut;
static char OrigOutputName[128];

static int num = 0;

//set sizes of our arrays   {descriptor, max_data_size}
UINT Sizes[2] = {153, 32000};

//char DevName[] = {"DSO9400"};
char DevName[] = {"GPIB1"};
char MessageString[512];
char DescOutName[128];

char Temp[128];
char Temp2[128];

float SampleInterval;
int cur_angle = 0;

CLC9400 LC9400;
LC9400Descriptor *WDesc;


/////// END DATA DECLARATION ////////

for(i=0; i < Sizes[0]; i++)
        Desc[i] = 0;

for(i=0; i < Sizes[1]; i++)
        Data[i] = 0;



//if(NumSweeps == 31)
//        Sizes[1] = 625;
//else
//        Sizes[1] = 500;                         //we care about the number of samples in a sweep



//Check for angle and name reset command//
if(Start == -1)
{
        WaveName->SetWindowText("c:\\data\\wave.dat");
        Message->SetWindowText("Resetting Variables...");

        strcpy(OrigOutputName, "c:\\data\\wave");
```

```
		num = 0;

		return(TRUE);
	}



	//append numbers to end of file name
	if(num > 0)
	{
		strcpy(Temp, OrigOutputName);

		strcat(Temp, itoa(num+1, Temp2, 10));

		strcpy(OutputFileName, strcat(Temp, ".dat"));

		WaveName->SetWindowText(OutputFileName);
	}


	else
	{

		temp = strlen(OutputFileName) - 4;
		strncpy(OrigOutputName, OutputFileName, temp);

		//temp = strlen(OutputFileName) - 4;
//		strcpy(OutputFileName, "c:\\data\\wave.dat");
//		strcpy(OrigOutputName, "c:\\data\\wave");
	}




	LC9400.GetSDD(Desc, Data, Sizes, NumSweeps, 1);


	//parse waveform descriptor
	WDesc = LC9400.GetWDesc();


	//// Open Files ////
	sprintf(DescOutName, "%s.desc", OutputFileName);

	if((DescOut = fopen(DescOutName, "w")) <= 0)
	{
		sprintf(MessageString, "Unable to open file <%s> for writing!    Aborting run.",
DescOutName);
		MessageBox(GetActiveWindow(), MessageString, "Error", MB_OK);
		return(FALSE);
	}
```

```
	//// Output Descriptor ////
	SampleInterval = (float)WDesc->TBase / (float)WDesc->NDiv;

	fprintf(DescOut, "%f\n%d\n%d\n", log10(SampleInterval), WDesc->NDiv, cur_angle);


	if((OutputFile = fopen(OutputFileName, "w")) <= 0)
	{
		fclose(DescOut);
		sprintf(MessageString, "Unable to open file <%s> for writing!    Aborting run.",
OutputFileName);
		MessageBox(GetActiveWindow(), MessageString, "Error", MB_OK);
		return(FALSE);
	}

	for(i = 0; i < (Sizes[1] - 5) * NumSweeps; i++)
	{
		Voltage[i] = WDesc->FVGain * ((Data[i] - 128.0)/32.0 - (WDesc->VOffset -
200.0)/25.0) * 200.0/(WDesc->VVGain + 80.0);

		fprintf(OutputFile, "%f\n", Voltage[i]);

		WriteProgress->SetPos((int)(100 * ((float)i / (float)((Sizes[1] - 5) * NumSweeps))));
	}

	fclose(DescOut);
	fclose(OutputFile);


	num++;

	int result = 1;			//dummy
	return(TRUE);
}
////////////////////////////////////////////////////////////////////////////
```

3) interface.h

```
// Contains definitions, prototypes, etc. for interface.cpp
//

#ifndef __INTERFACE__
#define __INTERFACE__


#define IBDEVCONN	100
#define IBFINDCONN	101
#define DEVOPEN			200
#define DEVCLOSED	201

struct DeviceData {
	UINT BoardIndex;
	UINT PrimaryAddress;
```

```
        UINT SecondaryAddress;
        UINT Timeout;
        UINT EOIMode;
        UINT EOSMode;
};



class CInterface
{
public:
        CInterface();                                           //constructor
        CInterface(UINT, UINT, UINT,
                        UINT, UINT, UINT);  //constructor, accepts device address
        CInterface(char *);                                     //constructor, accpets device name

        ~CInterface();                                          //destructor

        UINT OpenInterface();                           //attempt to get device descriptor for
interface
        UINT CloseInterface();                          //close device descriptor

        UINT WriteData(char *);                         //write data to device
        UINT ReadData(unsigned char *, UINT);   //read data from device and place in passed value


        UINT GetStatus() {return Status;}                               //get whether interface is
open or not

private:
        DeviceData *DevData;                            //contains information for use with ibdev
function

        char DevName[128];

        int DevDesc;                                            //device descriptor
        int BoardDesc;                                          //board descriptor

        UINT Status;                                            //status of connection
        UINT Type;                                                      //type of connection, set
during construction
};

#endif
```

4) interface.cpp

```
// Contains routines to handle interfacing with GPIB devices
// Author:  Noah W. Cushing
//
// Revision History
//                      9/17/97 -- Created file
//                                      Added the following functions
//                                                      -constructors
//                                                      -destructor
//                                                      -OpenInterface()
//                                                      -CloseInterface()
//                                                      -WriteData()
//                                                      -ReadData()
//

#include "stdafx.h"
#include "interface.h"
#include "decl-32.h"

#define IBOPENERROR 1001

////////////////////////////////////////////////////////////
//Class Constructors

CInterface::CInterface()
{
        DevDesc = -1;
        Status = DEVCLOSED;
}


CInterface::CInterface(UINT BoardIndex, UINT PrimaryAddress, UINT SecondaryAddress,
                                                UINT Timeout, UINT EOIMode, UINT EOSMode)
{
        DevData = new(DeviceData);

        DevData->BoardIndex = BoardIndex;
        DevData->PrimaryAddress = PrimaryAddress;
        DevData->SecondaryAddress = SecondaryAddress;
        DevData->Timeout = Timeout;
        DevData->EOIMode = EOIMode;
        DevData->EOSMode = EOSMode;

        Type = IBDEVCONN;
        Status = DEVCLOSED;
        DevDesc = -1;
}


CInterface::CInterface(char _DevName[])
{
        strcpy(DevName, _DevName);

        Type = IBFINDCONN;
        Status = DEVCLOSED;
        DevDesc = -1;
```

```
}

//////////////////////////////////////////////////////////
// Class Destructor
CInterface::~CInterface()
{
	if(Type == IBDEVCONN)
		delete(DevData);

	if(Status == DEVOPEN)
		CloseInterface();
}


//////////////////////////////////////////////////////////
// Open the GPIB interface
UINT CInterface::OpenInterface()
{
	if(Type == IBDEVCONN)
	{
		if((DevDesc = ibdev(DevData->BoardIndex, DevData->PrimaryAddress,
					DevData->SecondaryAddress, DevData->Timeout,
					 DevData->EOIMode, DevData->EOSMode)) < 0)

			return(IBOPENERROR);
	}

	else
	{
		if((DevDesc = ibfind(DevName)) < 0)
			return(IBOPENERROR);

	}


	BoardDesc = ibfind("GPIB1");

	Status  = DEVOPEN;

	int result_s = ibsic(DevDesc);

	int result = ibclr(DevDesc);

	int error = result & ERR;

	return(TRUE);
}
//////////////////////////////////////////////////////////
//Close the GPIB Interface
UINT CInterface::CloseInterface()
{
	ibloc(DevDesc); //was DevDesc

	if(Status == DEVOPEN) //was DevDesc
		ibonl(DevDesc, 0);
```

```
	return(TRUE);
}


////////////////////////////////////////////////////////
//Write Data to GPIB Interface
UINT CInterface::WriteData(char *Data)
{
	int result;
	int error;

	ibcmd(BoardDesc,"?_@$",4);
	result = ibwrt(DevDesc, Data, strlen(Data));

	error = result & ERR;

	return(TRUE);
}


UINT CInterface::ReadData(unsigned char *Data, UINT Size)
{
	int result;
	int error;

	ibcmd(BoardDesc,"?_ D",4);			//not sure about this
	result = ibrd(DevDesc, Data, Size);

	error = result & ERR;
	return(TRUE);
}
```

5) LC9400.h -- handles data received from LC9400

```
//			-- includes formatting of data, etc.


#ifndef __LC9400__
#define __LC9400__

#define START_STRING1			"CALIBRATE OFF"
#define START_STRING2			"COMM_TRAILER OFF"

#define END_STRING1			"CALIBRATE ON"
#define END_STRING2			"COMM_TRAILER ON"

#define FORMAT_STRING			"CFMT,A,BYTE"

#define READ_DESCRIPTOR1	"RD,C1.DE"
#define READ_DESCRIPTOR2	"RD,C2.DE"
```

```
#define READ_DESCRIPTORC    "RD,MC.DE"
#define READ_DESCRIPTORE    "RD,FE.DE"

#define READ_DATA1                  "RD,C1.DA"
#define READ_DATA2                  "RD,C2.DA"
#define READ_DATAC                  "RD,MC.DA"
#define READ_DATAE                  "RD,FE.DA"
#include "interface.h"
#include "globals.h"


class CLC9400 {
public:
        CLC9400();                                                  //class constructor
        CLC9400(UINT);                                              //class constructor specify memory
or channel
        ~CLC9400();                                                 //class destructor

        UINT GetDD(unsigned char *, unsigned char *, UINT *, UINT);         //get both sets of data,
waveform..data
                                //desc array     data array    sizes  channel


        UINT GetSDD( unsigned char *, unsigned char *, UINT *, UINT, UINT); //get sequence data
                                //desc array          data array                  sizes   sweeps channel

        LC9400Descriptor *GetWDesc() {return &WDesc;} //return waveform descriptor

private:
        void AlignSData(unsigned char *, UINT *, UINT);
        UINT GetDescriptor(unsigned char *, UINT, UINT); //get waveform descriptor for specified
channel
        UINT GetData(unsigned char *, UINT, UINT);                  //get waveform data for
specified channel
        UINT GetSData(unsigned char *, UINT, UINT, UINT);                   //get waveform
data for specified channel


        CInterface Interface;                       //interface pointing to LC9400
        LC9400Descriptor WDesc;                         //parsed waveform descriptor

        UINT SendStart();                               //initialize (send Start Strings 1 and 2 and
format)
        UINT SendEnd();                                 //send ending strings
        void AlignStrings(unsigned char *, unsigned char *, UINT *);
                                                                    //aligns waveform desc.
and data strings to start at 0byte
        void AlignSStrings(unsigned char *, unsigned char *, UINT *, UINT);

        UINT ParseDescriptor(unsigned char *);
                                                                    //parse values in
waveform descriptor
        void AlignDescriptor(unsigned char *, UINT *);

        UINT Memory;                                    //whether we're reading memory or a
channel
```

```
};


#endif
```

6) LC9400.c -- handles routines for interfacing with the LC9400
// digital oscilloscope

```
//
// Author:  Noah W. Cushing, Yurong Sun
//
// Revision History
//                      11/24/97 -- created file and added initial functions
//                      10/01/99 – modified the function of reading and writing Function E of
                                    LC9400



#include "stdafx.h"
#include "LC9400.h"
#include "decl-32.h"

//////////////////////////////////////////////////////////////////
// Class Constructors
CLC9400::CLC9400()
{
        int error = 0;
        int result = 0;

        UINT BoardIndex = 1;
        UINT PrimaryAddress = 4;
        UINT SecondaryAddress = 0;
        UINT Timeout = 10;
        UINT EOIMode = 1;
        UINT EOSMode = 1;

        //CInterface _Interface("DSO9400");
        CInterface _Interface("GPIB1");

        Interface = _Interface;

        result = Interface.OpenInterface();

        error = result & ERR;
```

```
	Memory = FALSE;
}


CLC9400::CLC9400(UINT ChanOrMem)
{
	int error = 0;
	int result = 0;

	if(ChanOrMem == 0)
		Memory = FALSE;
	else
		Memory = TRUE;

//	CInterface _Interface("DSO9400");
	CInterface _Interface("GPIB1");

	Interface = _Interface;

	result = Interface.OpenInterface();

	error = result & ERR;
}


////////////////////////////////////////////////////////////
// Class Destructor
CLC9400::~CLC9400()
{
	int error = 0;
	int result = 0;

	result = Interface.CloseInterface();

	error = result & ERR;

}



////////////////////////////////////////////////////////////
// Communications with LC9400
UINT CLC9400::SendStart()
{
	Interface.WriteData(START_STRING1);
	Interface.WriteData(START_STRING2);
	Interface.WriteData(FORMAT_STRING);

	return(TRUE);
}

UINT CLC9400::SendEnd()
{
	Interface.WriteData(END_STRING1);
	Interface.WriteData(END_STRING2);
```

```
	return(TRUE);
}

UINT CLC9400::GetDD(unsigned char *Desc, unsigned char *Data, UINT *Size, UINT Channel)
{
	if(Channel <= 0 || Channel > 2)
		return(FALSE);

	SendStart();
	GetDescriptor(Desc, Size[0], Channel);
	GetData(Data, Size[1], Channel);
	SendEnd();

	AlignStrings(Desc, Data, Size);
	ParseDescriptor(Desc);

	return(TRUE);
}

UINT CLC9400::GetSDD(unsigned char *Desc, unsigned char *Data, UINT *Size, UINT NumSweeps,
UINT Channel)
{
	if(Channel <= 0 || Channel > 2)
		return(FALSE);

	SendStart();
	GetDescriptor(Desc, Size[0], Channel);

	AlignDescriptor(Desc, Size);
	ParseDescriptor(Desc);

	Size[1] = WDesc.NDiv * 10;

	GetSData(Data, Size[1], NumSweeps, Channel);
	AlignSData(Data, Size, NumSweeps);


	SendEnd();

	return(TRUE);
}


UINT CLC9400::GetDescriptor(unsigned char *Desc, UINT Size, UINT Channel)
{
	int result = 0;

	/*if(Memory)
		result = Interface.WriteData(READ_DESCRIPTORC);
	else
	{
		if(Channel == 1)
			result = Interface.WriteData(READ_DESCRIPTOR1);
		else
			result = Interface.WriteData(READ_DESCRIPTOR2);
```

```
	}
	*/
	// Here we specify we read and write data to Function E
	result = Interface.WriteData(READ_DESCRIPTORE);

	result = Interface.ReadData(Desc, Size);

	result = result & ERR;

	return(TRUE);
}


UINT CLC9400::GetData(unsigned char *Data, UINT Size, UINT Channel)
{
	int result = 0;


	/*if(Memory)
		result = Interface.WriteData(READ_DATAC);
	else
	{
		if(Channel == 1)
			result = Interface.WriteData(READ_DATA1);
		else
			result = Interface.WriteData(READ_DATA2);
	}*/

	result = Interface.WriteData(READ_DATAE);

	result = Interface.ReadData(Data, Size);

	result = result & ERR;

	return(TRUE);
}


UINT CLC9400::GetSData(unsigned char *Data, UINT Size, UINT NumSweeps, UINT Channel)
{
	int result = 0;
	UINT i = 0, j = 0;
	char XmitString[32];
	unsigned char TempData[1000];

	for(i = 0; i < NumSweeps; i++)
	{
		sprintf(XmitString, "RD, C1.DA,,,,%d", i+1);

		if(Channel == 1)
			result = Interface.WriteData(XmitString);
		else
			result = Interface.WriteData(XmitString);

	//for(i = 0; i < 10; i++)
	//{
```

```
                result = Interface.ReadData(TempData, Size);

                result = result & ERR;


                for(j = 0; j < Size; j ++)
                {
                        Data[Size*i + j] = TempData[j];
                }
        }

        return(TRUE);
}


////////////////////////////////////////////////////////////
// Align and Parse Data
void CLC9400::AlignStrings(unsigned char *Desc, unsigned char *Data, UINT *Size)
{
        UINT i;

        for(i = 0; i < Size[0]; i++)
                Desc[i] = Desc[i + 4];

        for(i = 0; i < Size[1]; i++)
                Data[i] = Data[i + 5];
}


void CLC9400::AlignSStrings(unsigned char *Desc, unsigned char *Data, UINT *Size, UINT
NumSweeps)
{


        AlignDescriptor(Desc, Size);

        AlignSData(Data, Size, NumSweeps);
}



void CLC9400::AlignDescriptor(unsigned char *Desc, UINT *Size)
{
        UINT i;

        for(i = 0; i < Size[0]; i++)
                Desc[i] = Desc[i + 4];
}



void CLC9400::AlignSData(unsigned char *Data, UINT *Size, UINT NumSweeps)
{
        UINT i, j;

        for(j = 0; j < NumSweeps; j++)
```

```
                for(i = 0; i < Size[1] - 5; i++)
                        Data[(Size[1] - 5) * j + i] = Data[Size[1] * j + i + 5];
}


UINT CLC9400::ParseDescriptor(unsigned char *Desc)
{
        //get the fixed vertical gain
        switch((int)Desc[0])
        {
        case 22:
                WDesc.FVGain = (float)0.005;
                break;
        case 23:
                WDesc.FVGain = (float)0.01;
                break;
        case 24:
                WDesc.FVGain = (float)0.02;
                break;
        case 25:
                WDesc.FVGain = (float)0.05;
                break;
        case 26:
                WDesc.FVGain = (float)0.1;
                break;
        case 27:
                WDesc.FVGain = (float)0.2;
                break;
        case 28:
                WDesc.FVGain = (float)0.5;
                break;
        case 29:
                WDesc.FVGain = (float)1;
                break;
        case 30:
                WDesc.FVGain = (float)2.0;
                break;
        case 31:
                WDesc.FVGain = (float)5.0;
                break;

        default:
                break;                  //add some error checking later
        }



        //get the variable vertical gain
        //WDesc.VVGain = 0.4 + (0.005 * (float)Desc[1]);
        WDesc.VVGain = Desc[1];

        //get the vertical offset
        int value = (int)(Desc[5] | (Desc[4] << 8));
        //          WDesc.VOffset = -8.0 + (0.04 * (float)value);

        WDesc.VOffset = (float)(int)(Desc[5] | (Desc[4] << 8));  //added float 11/19
```

```
//get the input channel coupling
WDesc.ICoupling = (int)Desc[6];

//get whether BW limit on or off
if((int)Desc[8] == 0)
        WDesc.BWLimit = FALSE;
else
        WDesc.BWLimit = TRUE;

//get the timebase setting

switch((int)Desc[9])
{
case 4:
        WDesc.TBase = (float)2E-9;
        break;
case 5:
        WDesc.TBase = (float)5E-9;
        break;
case 6:
        WDesc.TBase = (float)10E-9;
        break;
case 7:
        WDesc.TBase = (float)20E-9;
        break;
case 8:
        WDesc.TBase = (float)50E-9;
        break;
case 9:
        WDesc.TBase = (float)0.1E-6;
        break;
case 10:
        WDesc.TBase = (float)0.2E-6;
        break;
case 11:
        WDesc.TBase = (float)0.5E-6;
        break;
case 12:
        WDesc.TBase = (float)1E-6;
        break;
case 13:
        WDesc.TBase = (float)2E-6;
        break;
case 14:
        WDesc.TBase = (float)5E-6;
        break;
case 15:
        WDesc.TBase = (float)10E-6;
        break;
case 16:
        WDesc.TBase = (float)20E-6;
        break;
case 17:
        WDesc.TBase = (float)50E-6;
        break;
```

```
case 18:
        WDesc.TBase = (float)0.1E-3;
        break;
case 19:
        WDesc.TBase = (float)0.2E-3;
        break;
case 20:
        WDesc.TBase = (float)0.5E-3;
        break;
case 21:
        WDesc.TBase = (float)1E-3;
        break;
case 22:
        WDesc.TBase = (float)2E-3;
        break;
case 23:
        WDesc.TBase = (float)5E-3;
        break;
case 24:
        WDesc.TBase = (float)10E-3;
        break;
case 25:
        WDesc.TBase = (float)20E-3;
        break;
case 26:
        WDesc.TBase = (float)50E-3;
        break;
case 27:
        WDesc.TBase = (float)0.1;
        break;
case 28:
        WDesc.TBase = (float)0.2;
        break;
case 29:
        WDesc.TBase = (float)0.5;
        break;
case 30:
        WDesc.TBase = (float)1.0;
        break;
case 31:
        WDesc.TBase = (float)2.0;
        break;
case 32:
        WDesc.TBase = (float)5.0;
        break;
case 33:
        WDesc.TBase = (float)10.0;
        break;
case 34:
        WDesc.TBase = (float)20.0;
        break;
case 35:
        WDesc.TBase = (float)50.0;
        break;
case 36:
        WDesc.TBase = (float)100.0;
```

```
        break;
default:
        break;          //error checking later
}


//find out the sampling interval at acquisition (only check for non-interleaved cases)
switch((int)Desc[10])
{
case 16:
        WDesc.SInterval = (float)10E-9;
        break;
case 17:
        WDesc.SInterval = (float)20E-9;
        break;
case 18:
        WDesc.SInterval = (float)40E-9;
        break;
case 19:
        WDesc.SInterval = (float)80E-9;
        break;
case 20:
        WDesc.SInterval = (float)200E-9;
        break;
case 21:
        WDesc.SInterval = (float)400E-9;
        break;
case 22:
        WDesc.SInterval = (float)800E-9;
        break;
case 23:
        WDesc.SInterval = (float)2E-6;
        break;
case 24:
        WDesc.SInterval = (float)4E-6;
        break;
case 25:
        WDesc.SInterval = (float)8E-6;
        break;
case 26:
        WDesc.SInterval = (float)20E-6;
        break;
case 27:
        WDesc.SInterval = (float)40E-6;
        break;
case 28:
        WDesc.SInterval = (float)80E-6;
        break;
case 29:
        WDesc.SInterval = (float)200E-6;
        break;
case 30:
        WDesc.SInterval = (float)400E-6;
        break;
case 31:
        WDesc.SInterval = (float)800E-6;
```

```
                break;
        case 32:
                WDesc.SInterval = (float)2E-3;
                break;
        case 33:
                WDesc.SInterval = (float)4E-3;
                break;
        case 34:
                WDesc.SInterval = (float)8E-3;
                break;
        case 35:
                WDesc.SInterval = (float)20E-3;
                break;
        case 36:
                WDesc.SInterval = (float)40E-3;
                break;
        default:
                break;
        }


        //get number of points per division
        value = (int)((int)Desc[23] |  ((int)Desc[22] << 8));
        WDesc.NDiv = (UINT)value;

        //get whether data processed or not
        WDesc.PType = (int)Desc[34];

        return(TRUE);
}
```

II. Signal Processing Program (Matlab source code)

```
1) function out=agetdata(filename)
%----------------------==========================================================
% AGETATA(FILENAME)
%
%     Reads in a file that contains ascii data and returns an array
%        containing the data
%
%Error Checking is performed, returns 2 element array on failure
%        with -1 in the first (1) position
%
%     Examples:   agetdata('c:\data\data.dat');
%
%        test = agetdata(String)
%            if test(1)==-1
%              failure
%----------------------==========================================================


DataFP = fopen(filename, 'r');

if DataFP == -1
   String = ['<!! AGETDATA...Error opening file: ' filename ' !!>'];
   disp(String);

   out = [-1 0];
else
   String = ['<...AGETDATA...Reading data from ' filename '...>'];
   disp(String);
   [out, count] = fscanf(DataFP, '%f');
   fclose(DataFP);

   if count < 1
      disp('<!! AGETDATA...Error reading file:  File contains no data.
!!>');
      out = [-1 0];
   end

end
```

```
2) function out = analysis2()

%=========================================================================
===
   %  Create Graphical User Interface Screen Functions

%=========================================================================
===

  figNumber=figure( ...
      'Name','Ultrasound Analysis System', ...
      'NumberTitle','off', ...
      'Tag', 'Fig', ...
      'Visible','off', ...
      'Position',[100 300 800 400]);

     axes( ...
         'Units','normalized', ...
         'Position',[0.07 0.12 0.7 0.65]);

   %====================================
   %Console and Button Information
   labelColor=[0.8 0.8 0.8];
   top=0.95;
   bottom=0.05;
   yInitLabelPos=0.90;
   labelWid=0.20;
   labelHt=0.05;
   btnWid=0.10;
   btnHt=0.05;
   left = 1 - (btnWid+0.05);

   % Spacing between the label and the button for the same command
   btnOffset=0.003;
   % Spacing between the button and the next command's label
   spacing=0.05;

   %====================================
   % The CONSOLE frame
   frmBorder=0.02;
   yPos=0.05-frmBorder;
   frmPos=[left-frmBorder yPos btnWid+2*frmBorder 0.9+2*frmBorder];
   h=uicontrol( ...
      'Style','frame', ...
      'Units','normalized', ...
      'Position',frmPos, ...
      'BackgroundColor',[0.50 0.50 0.50]);


   value = 0;

   cwave_popup = uicontrol( ...
      'Style', 'pushbutton', ...
      'String', 'wave', ...
      'Value', 1, ...
      'Tag', 'Wave', ...
      'Units', 'Normalized', ...
```

```
   'Position', [0.05 0.92 0.06 0.05], ...
   'CallBack' , [...
      'UD = get(findobj(''Tag'', ''Wave''), ''UserData'');', ...
      'val = get(findobj(''Tag'', ''Xcorr''), ''UserData'');',...
      'aplot(UD, val(1), ''Measured Wave'');']);


cbase_popup = uicontrol( ...
   'Style', 'pushbutton', ...
   'String', 'base', ...
   'Value', 1, ...
   'Tag', 'Base', ...
   'Units', 'Normalized', ...
   'Position', [0.13 0.92 0.06 0.05], ...
   'CallBack' , [...
      'UD = get(findobj(''Tag'', ''Base''), ''UserData'');', ...
      'val = get(findobj(''Tag'', ''Xcorr''), ''UserData'');',...
      'aplot(UD, val(1), '''');']);

cwwave_popup = uicontrol( ...
   'Style', 'pushbutton', ...
   'String', 'wwave', ...
   'Value', 1, ...
   'Tag', 'Wwave', ...
   'Units', 'Normalized', ...
   'Position', [0.21 0.92 0.06 0.05], ...
   'CallBack' , [...
      'UD = get(findobj(''Tag'', ''Wwave''), ''UserData'');', ...
      'val = get(findobj(''Tag'', ''Xcorr''), ''UserData'');',...
      'aplot(UD, val(1), ''Windowed Wave'');']);

cwbase_popup = uicontrol( ...
   'Style', 'pushbutton', ...
   'String', 'wbase', ...
   'Value', 1, ...
   'Tag', 'Wbase', ...
   'Units', 'Normalized', ...
   'Position', [0.29 0.92 0.06 0.05], ...
   'CallBack' , [...
      'UD = get(findobj(''Tag'', ''Wbase''), ''UserData'');', ...
      'val = get(findobj(''Tag'', ''Xcorr''), ''UserData'');',...
      'aplot(UD, val(1), ''Windowed Base'');']);

   ccwwave_popup = uicontrol( ...
   'Style', 'pushbutton', ...
   'String', 'cwwave', ...
   'Value', 1, ...
   'Tag', 'Cwwave', ...
   'Units', 'Normalized', ...
   'Position', [0.37 0.92 0.06 0.05], ...
   'CallBack' , [...
      'UD = get(findobj(''Tag'', ''Cwwave''), ''UserData'');', ...
      'val = get(findobj(''Tag'', ''Xcorr''), ''UserData'');',...
      'aplot(UD, val(1), ''Downsampled, Windowed Wave'');']);


ccwbase_popup = uicontrol( ...
```

```
      'Style', 'pushbutton', ...
      'String', 'cwbase', ...
      'Value', 1, ...
      'Tag', 'Cwbase', ...
      'Units', 'Normalized', ...
      'Position', [0.45 0.92 0.06 0.05], ...
      'CallBack' , [...
         'UD = get(findobj(''Tag'', ''Cwbase''), ''UserData'');', ...
         'val = get(findobj(''Tag'', ''Xcorr''), ''UserData'');',...
         'aplot(UD, val(1), ''Downsampled, Windowed Reference'');']);


   cwmag_popup = uicontrol( ...
      'Style', 'pushbutton', ...
      'String', 'wmag', ...
      'Value', 1, ...
      'Tag', 'Wmag', ...
      'Units', 'Normalized', ...
      'Position', [0.53 0.92 0.06 0.05], ...
      'CallBack' , [...
         'UD = get(findobj(''Tag'', ''Wmag''), ''UserData'');', ...
         'val = get(findobj(''Tag'', ''Xcorr''), ''UserData'');',...
         'aplot(UD, val(4), ''Wave Magnitude Spectrum'');']);


   cbmag_popup = uicontrol( ...
      'Style', 'pushbutton', ...
      'String', 'bmag', ...
      'Value', 1, ...
      'Tag', 'Bmag', ...
      'Units', 'Normalized', ...
      'Position', [0.61 0.92 0.06 0.05], ...
      'CallBack' , [...
         'UD = get(findobj(''Tag'', ''Bmag''), ''UserData'');', ...
         'val = get(findobj(''Tag'', ''Xcorr''), ''UserData'');',...
         'aplot(UD, val(4), ''Reference Magnitude Spectrum'');']);

   cmag_popup = uicontrol(...
      'Style','pushbutton', ...
      'Units', 'Normalized', ...
      'Position', [0.69 0.92 0.06 0.05], ...
      'Value', 1, ...
      'String','Rmag', ...
      'Tag', 'Mag', ...
       'Callback', [...
         'UD = get(findobj(''Tag'', ''Mag''), ''UserData'');',...
         'val = get(findobj(''Tag'', ''Xcorr''), ''UserData'');',...
         'aplot(UD, val(4), ''Magnitude Difference: Rmag - Wmag'');',
...
         'axis([0 10^6 val(7) val(8)]);']);


   cplot_popup = uicontrol(...
      'Style','pushbutton', ...
      'Units', 'Normalized', ...
      'Position', [0.09 0.85 0.06 0.05], ...
      'Value', 1, ...
```

```
   'String','PlotFile', ...
   'Tag', 'Plot', ...
   'Callback', 'plotfile');


cviewseq_popup = uicontrol(...
   'Style','pushbutton', ...
   'Units', 'Normalized', ...
   'Position', [0.17 0.85 0.06 0.05], ...
   'Value', 1, ...
   'String','ViewSeq', ...
   'Tag', 'ViewS', ...
   'Callback', [...
      'UD = viewseq(0);'...
      'set(findobj(''Tag'', ''LastS''), ''UserData'', UD)']);


clastseq_popup = uicontrol(...
   'Style','pushbutton', ...
   'Units', 'Normalized', ...
   'Position', [0.25 0.85 0.06 0.05], ...
   'Value', 1, ...
   'String','LastSeq', ...
   'Tag', 'LastS', ...
   'Callback', [...
      'UD = get(findobj(''Tag'', ''LastS''), ''UserData'');'...
      'viewseq(UD)']);

uicontrol( ...
   'Style','pushbutton', ...
   'Units', 'Normalized', ...
   'Position',[0.33 0.85 0.06 0.05], ...
   'String','IAS', ...
   'Callback', 'threeias(1, 1, 1)');

uicontrol( ...
   'Style','pushbutton', ...
   'Units', 'Normalized', ...
   'Position',[0.41 0.85 0.06 0.05], ...
   'String','ACORR', ...
   'Tag', 'Acorr', ...
   'Callback', 'acorr(1, 1, 1)');

  uicontrol( ...
   'Style','pushbutton', ...
   'Units', 'Normalized', ...
   'Position',[0.49 0.85 0.06 0.05], ...
   'String','CCorr', ...
   'Callback', [...
      'UD1 = get(findobj(''Tag'', ''Base''), ''UserData'');', ...
      'UD2 = get(findobj(''Tag'', ''Wave''), ''UserData'');', ...
      'UD3 = get(findobj(''Tag'', ''Xcorr''), ''UserData'');', ...
      'attcorrcorr(UD1, UD2, UD3)']);
```

```
    %====================================
    %Close button

    uicontrol( ...
       'Style','pushbutton', ...
       'Units','normalized', ...
       'Position',[left bottom btnWid 2*btnHt], ...
       'String','Close', ...
       'Callback', 'close(gcf)');

    buttonb = bottom + 2*btnHt + 0.02;

    uicontrol( ...
       'Style','pushbutton', ...
       'Units','normalized', ...
       'Position',[left buttonb btnWid 2*btnHt], ...
       'String','Select Files', ...
       'Callback', 'analysis(10);');

    buttonb = top - 2*btnHt - 0.02 - frmBorder;

    xcorr_button = uicontrol( ...
       'Style','pushbutton', ...
       'Units','normalized', ...
       'Position',[left buttonb btnWid 2*btnHt], ...
       'String','XCORR', ...
       'Tag', 'Xcorr', ...
       'Callback', [...
          'UD1 = get(findobj(''Tag'', ''Wbase''), ''UserData'');', ...
          'UD2 = get(findobj(''Tag'', ''Wwave''), ''UserData'');', ...
          'UD3 = get(findobj(''Tag'', ''Xcorr''), ''UserData'');', ...
          'correlation(UD1, UD2, UD3)']);



    buttonb = buttonb - 2*btnHt - 0.02;

    bua_button = uicontrol( ...
       'Style','pushbutton', ...
       'Units','normalized', ...
       'Position',[left buttonb btnWid 2*btnHt], ...
       'String','BUA', ...
       'Tag', 'Bua',...
       'Callback', [...
          'UD = get(findobj(''Tag'', ''Mag''), ''UserData'');',...
          'val = get(findobj(''Tag'', ''Xcorr''), ''UserData'');',...
          'aplot(UD, val(4), ''Select BUA region...'');', ...
          'text(600000, max(UD)/2, ''Select BUA region'', ''color'',
''red'');', ...
          'axis([0 10^6 val(7) val(8)]);',...
          'UD1 = get(findobj(''Tag'', ''Bua''), ''UserData'');',...
          'UD2 = get(findobj(''Tag'', ''Xcorr''), ''UserData'');',...
          'bua(UD1, UD2);']);

    buttonb = buttonb - 2*btnHt - 0.02;
```

```
    ibs_button = uicontrol(...
        'Style','pushbutton', ...
        'Units','normalized', ...
        'Position',[left buttonb btnWid 2*btnHt], ...
        'String','IBS', ...
        'Tag', 'Ibs', ...
        'Callback', [...
            'UD1 = get(findobj(''Tag'', ''Ibs''), ''UserData'');',...
            'UD2 = get(findobj(''Tag'', ''Acorr''), ''UserData'');', ...
            'ibs(UD2, UD1)']);

    buttonb = buttonb - 2*btnHt - 0.02;

    uicontrol(...
        'Style','pushbutton', ...
        'Units','normalized', ...
        'Position',[left buttonb btnWid 2*btnHt], ...
        'String','Group Velocity', ...
        'Callback', [...
            'UD1 = get(findobj(''Tag'', ''Cwwave''), ''UserData'');',...
            'UD2 = get(findobj(''Tag'', ''Cwbase''), ''UserData'');',...
            'UD3 = get(findobj(''Tag'', ''Xcorr''), ''UserData'');',...
            'groupvel(UD1, UD2, UD3);']);



    % Now uncover the figure
    set(figNumber,'Visible','on');


%=========================================================================
===
%  End Screen Functions
%=========================================================================
===
```

```
3) function out = analysis(mode, base, data)
%----------------------==========================================================
% ANALYSIS(MODE, BASE, DATA)
%
%     Values for MODE:  10   --> Use file selection dialog boxes.
%            BASE and DATA not required
%
%                       1   --> Perform analysis on files in c:\data
%
%     BASE refers to a measurement of the water filled tank WITHOUT a
%     sample in place.
%
%     DATA refers to a measurement of the water filled tank WITH a
%     sample in place.
%
%
%     ** Note:  BASE and DATA measurments must be taken with the same
sampling rate. **
%
%     Examples:   analysis(10);
%                analysis(1, 'base', 'wave');
%
%----------------------==========================================================

if mode == 10
   [dfname, dpname] = uigetfile('*.dat', 'Select Data File');

   if dfname == 0    %the user pressed cancel
      disp('<...ANALYSIS...User Abort on data file select...>');
      return;     %abort and return to window
   end

   DescFile = [dpname dfname '.desc'];

elseif mode == 1
   dfname = [data '.dat'];
   dpname = ['c:\data\'];
   DescFile = [dpname dfname '.desc'];

elseif mode == 100 | mode == 200
   dfname = data;
   DescFile = [dfname '.desc'];
end



%read in and parse descriptor file
Param = parse_descriptor(DescFile);
if(Param(1) == -1)         %unable to read file
   DispString = sprintf('<...ANALYSIS...Unable to get descriptor from
file %s ..>', DescFile);
   disp(DispString);
   out = [-1 0];
   return;
```

```
end


TimePerSample = Param(1);
NumSamples = Param(2)
SampRate = Param(3);
HzPerSample = Param(4);



if mode == 10
   [bfname, bpname] = uigetfile('*.dat', 'Select Base (water only)
File');
   if bfname == 0
      disp('<...ANALYSIS...User Abort on reference file select...>');
      return;     %abort and return to window
   end
elseif mode == 1
   bfname = [base '.dat'];
   bpname = ['c:\data\'];
elseif mode == 100 | mode == 200
   bfname = base;
   bpname = ['c:\data\'];
end


%load the data sets
if mode == 100 | mode == 200
   DataFile = data;
   BaseFile = base;
else
   DataFile = [dpname dfname]
   BaseFile = [bpname bfname]
end

wave = agetdata(DataFile);
base = agetdata(BaseFile);

fbase = 1:NumSamples;
for i = 1:NumSamples
   fbase(i) = i * TimePerSample;
end

plot (fbase, wave);
y = max(wave);
text(fbase(NumSamples)/3, y, 'Select Window Region', 'color', 'red');
xlabel('Time (s)')
ylabel('Voltage (v)')
points = getclicks_t (gca, TimePerSample, NumSamples);

%Points returned window start, end time value
wstart = points(1)
wend = points(2)

% Adjust to points index according to TimeBase
w2start = round (wstart/TimePerSample)
w2end   = round (wend/TimePerSample)
```

```
% Some error checking
if w2start > NumSamples
   w2start = NumSamples;
elseif w2start <= 0
   w2start = 1;
end

if w2end > NumSamples
   w2end = NumSamples;
elseif w2end <= 0
   w2end = 1;
end



%set some parameters for windowing, downsampling, and zero padding
%WindowTime = 20*10^-6;                %window length
WindowTime = wend - wstart               %window length
WindowRes = 10^4;                    %frequency resolution of final,
zero padded signal in Hz/point
DesSampRate = 10*10^6;                %desired sampling rate after
downsampling
PaddedTime = 100^-6;                 %time length of zero padded signal

%We want a time range of 100us to provide 10kHz frequency resolution
Param(6) = WindowTime / Param(1);    %number of points included in
window
%   Param(4) = WindowRes;

disp(Param);




%Window the data for bua/etc.
IBSWave = ibs_window_wave(wave, Param(1));


OrigTimePerSample = Param(1);

%Window the data
wbase = window_wave (base, 2000);

wwave = Window_waves (wave, w2start, w2end);

%Remove any DC Components

IBSWave = removeDC(IBSWave);
wwave = removeDC(wwave);
wbase = removeDC(wbase);

if mode == 100
%   out = [correlation(wbase, wwave) encorrelation(wbase, wwave,
OrigTimePerSample) Param(9)];
```

```
   out = [correlation(wbase, wwave) attcorrcorr(base, wave,
OrigTimePerSample) Param(9)];

return;
elseif mode == 200
   out = [ibs(wwave, OrigTimePerSample) Param(9)];
   return
end


%Now Downsample to desired rate
disp('<ANALYSIS...Downsampling...>');
if DesSampRate < Param(3)                          %only downsample if
needed
   factor = round(Param(3)/ DesSampRate);          %downsample by a factor
of 'factor'
   cwwave = compress(wwave, factor);
   cwbase = compress(wbase, factor);
   TimePerSample = Param(1) * factor;
else
   cwwave = wwave;
   cwbase = wbase;
   TimePerSample = Param(1);
end

SampRate = 1/TimePerSample;
Param(3) = SampRate;


%Pad the waveforms out to desired time
cwwave = zero_pad(cwwave, DesSampRate);
cwbase = zero_pad(cwbase, DesSampRate);

%Calculate ffts
disp('<ANALYSIS...Calculating FFTs...>');
base_mag = 20*log10(abs(fft(cwbase)))-7;

wave_mag = 20*log10(abs(fft(cwwave)))-47;

result_mag = abs(base_mag - wave_mag);
result_size = size(result_mag);

Param(5) = HzPerSample;                %store old value for plotting
HzPerSample = Param(3)/result_size(2);

Param(4) = HzPerSample;
Param(6) = result_size(2);
HzPerSample
result_size(2)

fbase = 1:result_size(2);
for i = 1:result_size(2);
   fbase(i) = i * HzPerSample;         %set frequency scale
end
```

```
ymin = min(result_mag);
ymax = max(result_mag);

Param(7) = ymin;
Param(8) = ymax;

set(findobj('Tag', 'LastS'), 'UserData', 0);
set(findobj('Tag', 'Wave'), 'UserData', wave);
set(findobj('Tag', 'Base'), 'UserData', base);
set(findobj('Tag', 'Wwave'), 'UserData', wwave);
set(findobj('Tag', 'Wbase'), 'UserData', wbase);
set(findobj('Tag', 'Cwwave'), 'UserData', cwwave);
set(findobj('Tag', 'Cwbase'), 'UserData', cwbase);
set(findobj('Tag', 'Wmag'), 'UserData', wave_mag);
set(findobj('Tag', 'Bmag'), 'UserData', base_mag);
set(findobj('Tag', 'Mag'), 'UserData', result_mag);
set(findobj('Tag', 'Xcorr'), 'UserData', Param);
set(findobj('Tag', 'Bua'), 'UserData', result_mag);
set(findobj('Tag', 'Ibs'), 'UserData', OrigTimePerSample);
set(findobj('Tag', 'Acorr'), 'UserData', IBSWave);
```

4)angle_corr.m

```
% An independent program to calculate the angular correlation function
% When it starts, the user will be prompted to select the reference
signal, and select the signal in the sequence

clear all
close all

[bfname, bpname] = uigetfile('*.dat', 'Select reference File');
if bfname == 0
      disp('<...ANALYSIS...User Abort on reference file select...>');
      return;     %abort and return to window
end

BaseFile = [bpname bfname]

base = agetdata(BaseFile);
wbase = base';

wave = zeros(10000, 15);
for c=1:15
   [dfname, dpname] = uigetfile('*.dat', 'Select Data File');

   if dfname == 0    %the user pressed cancel
      disp('<...ANALYSIS...User Abort on data file select...>');
      return;     %abort and return to window
   end

   DescFile = [dpname dfname '.desc'];

   %read in and parse descriptor file
   Param = parse_descriptor(DescFile);
   if(Param(1) == -1)         %unable to read file
      DispString = sprintf('<...ANALYSIS...Unable to get descriptor
from file %s ..>', DescFile);
      disp(DispString);
      out = [-1 0];
      return;
   end

   DataFile = [dpname dfname]

   wave(:,c) = agetdata(DataFile);

end

TimePerSample = Param(1);
NumSamples = Param(2);
SampRate = Param(3);
HzPerSample = Param(4);

DesSampRate = 10*10^6;                %desired sampling rate after
downsampling
```

```
%windowing, downsampling, zeropadding
factor = round(Param(3)/ DesSampRate);       %downsample by a factor of
'factor'

if DesSampRate < Param(3)                          %only downsample if
needed
   cwbase = compress(wbase, factor);
   TimePerSample = Param(1) * factor;
else
   cwbase = wbase;
   TimePerSample = Param(1);
end

cwbase = 0.4467*cwbase;                      %Think about attenuation

%Zero Padding from 50us to 100us
cbase = [zeros(1, 500) cwbase zeros(1, 500)];
size_cbase = size(cbase)

base_energy = energy(cwbase, 1)

z = zeros(15, 1000);
zzz = zeros(15, 2*size_cbase(2) - 1);
zz = zeros(15, size_cbase(2));

temp = 0;
for c = 1:15
   %wwave = Window_wave (wave(:,c), Param(6));
   wwave = wave(:,c)';

   %Remove DC component
   %wwave = removeDC(wwave);

   %Now Downsample to desired rate
   disp('<ANALYSIS...Downsampling...>');
   cwwave = compress(wwave, factor);

   %Consider of Magnification of Signal: Wave 47dB, Base 7dB
   cwwave = 0.0044668*cwwave;

   %Pad the waveforms out to desired time
   cwave = [zeros(1, 500) cwwave zeros(1, 500)];

   z(c,:) = cwave(501:1500);
   max_wave = max(z(c,:));

   if (temp < max_wave)
      temp = max_wave;
   end


   corr1 = xcorr(cbase, cwave, 'coeff');
   corr_size = size(corr1)
   fcorr1 = fft(corr1);
```

```
   for i = (corr_size(2)+1)/2:corr_size(2)
        fcorr1(i) = 0;
   end

   for i = 1:(corr_size(2)-1)/2
        fcorr1(i) = 2*fcorr1(i);
     end

   correlation = ifft(fcorr1);

   size(correlation)
   zzz(c,:) = abs(correlation);% ./ (D));
   zz(c,:) = zzz(c,1001:3000);

   max_value(c) = max(zz(c,:));

end

max_wave = temp

t = (1:1000)*10^(-7);
theta = -35:5:35;


figure;
for c= 2:2:14
   plot(t, z(c,:)/(max_wave) + c)
   hold on;
end
axis([0 1e-4 0 16]);
xlabel('Time (s)'), ylabel('pulse number')
hold off


figure;
plot(theta, max_value)
xlabel('Angle (Degree)'), ylabel('Normalized Correlation Peak Value')

max_value'
```

```
5) function out = aplot(data, val, string)


data_size = size(data);

if data_size(1) > data_size(2)
   ds = data_size(1);
else
   ds = data_size(2);
end

for i = 1:ds
   fbase(i) = i*val;
end




plot(fbase, data);
y = max(data)/2;
text(fbase(ds)/2, y, string, 'color', 'red');
```

```
6)function slope = bestfit(base, data)
%slope  -- calculates slope of line of best fit
%BESTFIT Return the slope of the line of best fit for the given data
%        (independent variable) and base (dependent variable)
%Author:  Noah W. Cushing  11/1997


%initialize variables
base_sum = 0;
data_sum = 0;
s_xy = 0;
s_xx = 0;

base_size = size(base);

%calculate averages
for i = 1:base_size(2)
   base_sum = base_sum + base(i);
   data_sum = data_sum + data(i);
end

base_avg = base_sum / base_size(2);
data_avg = data_sum / base_size(2);

for i = 1:base_size(2)
   s_xy = s_xy + ((base(i) - base_avg)*(data(i) - data_avg));
   s_xx = s_xx + (base(i) - base_avg)^2;
```

```
end


slope(1) =  s_xy / s_xx;
slope(2) = data_avg - (slope(1) * base_avg);
```

7)

```
function bob = bua(result_mag, Param)

HzPerSample = Param(4);
SampleRate = Param(3);
NumPoints = Param(6);
EndSample = Param(5);

points = getclicks(gca, HzPerSample, NumPoints);

%points returns frequency values
fstart = points(1);
fend = points(2);

%set the frequency base
fbase = 1:NumPoints;
bua_region = 1:NumPoints;

%set bua range in terms of array indices
bua2start = round(fstart/HzPerSample);
bua2end = round(fend/HzPerSample);


%some error checking -- not sure if it's necessary
if bua2start > NumPoints
   bua2start = NumPoints;
elseif bua2start <= 0
   bua2start = 1;
end

if bua2end > NumPoints
   bua2end = NumPoints;
end


bua_region2 = 1:(bua2end - bua2start);

fbase2 = bua_region2;

temp = size(fbase);

for i = 1:temp(2)
   fbase(i) = (i - 1) * HzPerSample;
end
```

```
sum=0;
for i = bua2start:bua2end
   fbase2(i - bua2start + 1) = fbase(i);
   bua_region2(i - bua2start + 1) = result_mag(i);
%calculate the average value of the bua

sum=sum+result_mag(i);

%avg=sum/i_size;
end


sum=0;
for i = bua2start:bua2end
  % n=length(i)
  result_mag(i)
  sum = sum+(result_mag(i))
   % avg=sum(result_mag(i))/n;
 %result=result_mag(i)

end
i = bua2start:bua2end
ssize=size(i)
s=length(i)
sum
sum2=sum/s
avg=sum2;


slope2 = bestfit(fbase2, bua_region2);

line2 = fbase2;   %ensure line and fbase have same length

for i = bua2start:bua2end
   line2(i - bua2start + 1) = (slope2(1) * fbase2(i - bua2start + 1)) +
slope2(2);
end;

%calculate the average value of the bua

%sum = 0;
%for i = bua2start:bua2end


%wavesize = size(wave);

%for i = 1:wavesize(2)
 %  sum = sum + wave(i);
%end
%avg = sum / wavesize(2);



plot(fbase2, bua_region2, 'r-', fbase2, line2, 'b:');
%plot(fbase2,sum, 'r-');
```

```
%plot(fbase2, bua_region2, 'r-', fbase2, line2, 'b:', fbase2, sum, 'g-
.');

d = date;
bua = slope2(1) * 10^6

%open a new window for graphing

%set labels
xlabel('Frequency (Hz)');
ylabel('Magnitude (dB)');
%title([d,'  c:\\data\\', 'SPAM', '.dat  Bua: ', num2str(bua), '
dB/MHz']);

title([d,' average : 'num2str(avg),'  c:\\data\\', 'SPAM', '.dat  Bua:
', num2str(bua), ' dB/MHz']);
```

8)
```
function out = compress(data, factor)

temp = size(data);

for i = 1:fix(temp(2) / factor)
   temp2(i) = data(factor * i);
end

out = temp2;
```

9)
```
function out = energy(data, TimePerSample)

temp = size(data);

if temp(1) > temp(2)
   sztemp = temp(1)
else
   sztemp = temp(2)
end



temp2 = 0;

%calculate energy (using rectangle approximation)
for i = 1:sztemp
   temp2 = temp2 + ((data(i) * data(i)) * TimePerSample);
```

```
end

out = temp2;
```

```
10) function out=formattext(string)
% formattext(string) takes a text input (such as a filename) and
formats it with
%  additional /'s for proper printing

sz = size(string);

index = 1;

for i = 1:sz(2)
   if string(i) == '\'
      out(index) = '\';
      index = index+1;
      out(index) = '\';
   else
      out(index) = string(i);
   end

   index = index+1;
end
```

```
11) function out = getclicks_t(gca, TimePerSample, NumSamples)

click = 0;
done = 0;
fstart = -1;                    %set fstart to one left of origin
fend = (TimePerSample * NumSamples) + 1;     %set fend one to one right
of end of graph

while ~done
   waitforbuttonpress;

   currPt=get(gca,'CurrentPoint');   % value depends on settings of
axes

   if((currPt(1) > 0) & (currPt(1) < (NumSamples * TimePerSample)))
      text(currPt(1), currPt(3), 'X', 'color', 'red');
      drawnow
```

```
      if fstart < 0,
         fstart = currPt(1);
      elseif fend > (NumSamples * TimePerSample)
         fend = currPt(1);
      end;
     if fstart > fend               %make sure fstart < fend
        temp = fstart;
        fstart = fend;
        fend = temp;
     end;

     click = click + 1;
  end;

  if click > 1
     done = 1;
  end;

end;
out = ([fstart fend]);
```

```
12) function out=parse_descriptor(filename)
%----------------------=============================================------
% PARSE_DESCRIPTOR(FILENAME)
%
%  Parses descriptor files written by ultra.exe
%
%  Internal function for analysis.m
%
%  Returns [-1 0] on failure
%----------------------=============================================------

Desc = agetdata(filename);

DescSize = size(Desc);

if Desc(1) == -1
   disp('<...PARSE_DESCRIPTOR...Descriptor file not found!...>');
   out(1) = -1;
elseif DescSize(1) < 2
   disp('<...PARSE_DESCRIPTOR...Descriptor file too short!...>');
   out(1) = -1;
else
   out(1) = 10^Desc(1);             %Calculate time per sample
   out(2) = 10 * Desc(2);           %Calculate number of samples
   out(3) = 1 / out(1);          %Calculate sampling rate
   out(4) = out(3)/out(2);       %Number of Hz per sample
   out(5) = 0;                      %NOT USED
   out(4) = round(out(4));       %round to the nearest integer
   if DescSize(1) > 2
      out(9) = Desc(3);             %current angle
   else
```

```
      disp('<...PARSE_DESCRIPTOR...Warning:  No angle data in
Descriptor file...>');
   end
end
```

13)
```
function out = plotfile()

 [dfname, dpname] = uigetfile('*.dat', 'Select Data File');

 if dfname == 0
    return
 end


 newname = [strtok(dfname, '.') '.angles.dat'];


 abase = agetdata([dpname newname]);
 temp = agetdata([dpname dfname]);

 asize = size(abase);
 tsize = size(temp);


if asize(1) == -1 | asize(1) == tsize(1)
    temp = agetdata([dpname dfname]);
    plot(abase, temp);
    asize = size(abase);
    y2 = max(temp);
    y1 = min(temp);
    x2 = abase(1);
    x1 = abase(asize(1));
    axis([x1 x2 y1 y2]);
    s = date;
    string = formattext([dpname dfname]);
    title([string '    ' s]);
```

```
    grid on;
    zoom on;
 else
    temp = agetdata([dpname dfname]);
    plot(temp);
    s = date;
    string = formattext([dpname dfname]);
    title([string '    ' s]);
    grid on;
    zoom on;
 end
```

14) `function out = removeDC(wave)`

```
disp('<REMOVEDC...Removing DC Components...>');
%remove DC component from windowed waveform
sum = 0;
wavesize = size(wave);

for i = 1:wavesize(2)
   sum = sum + wave(i);
end

avg = sum / wavesize(2);

for i = 1:wavesize(2)
   out(i) = wave(i) - avg;
end
```

15) velocity: calculate wave velocity.

```
clear all;
close all;


%% input wave waveform
wave = zeros(10000, 19);
for c=1:19
   [dfname, dpname] = uigetfile('*.dat', 'Select Data File');

   if dfname == 0    %the user pressed cancel
      disp('<...ANALYSIS...User Abort on data file select...>');
      return;     %abort and return to window
   end
```

```
   DescFile = [dpname dfname '.desc'];

   %read in and parse descriptor file
   Param = parse_descriptor(DescFile);
   if(Param(1) == -1)         %unable to read file
      DispString = sprintf('<...ANALYSIS...Unable to get descriptor
from file %s ..>', DescFile);
      disp(DispString);
      out = [-1 0];
      return;
   end

   DataFile = [dpname dfname]

   wave(:,c) = agetdata(DataFile);
end

TimePerSample = Param(1);
NumSamples = Param(2);
SampRate = Param(3);
HzPerSample = Param(4);

twave = 1:NumSamples;
for i = 1:NumSamples
   twave(i) = i*TimePerSample;
end

Cwater = 1481;
Ds = 0.03765;
for c = 1:19
   figure;
   plot(twave, wave(:,c));
   y = max(wave(:,c));
   text(twave(NumSamples)/3, y, 'Select Window Region', 'color',
'red');
   xlabel('Time(s)')
   ylabel('Voltage(v)')
   points = getclicks_t (gca, TimePerSample, NumSamples);

   wstart = points(1);
   wend = points(2);
   delta_t = wend - wstart

   Velocity (c) = Cwater*Ds/(Ds - Cwater*delta_t);


end

figure;
theta = 0:10:180;
plot(theta, Velocity)
xlabel('Angle (Degree)'), ylabel('Fast Wave Velocity Peak Value')
Velocity
```

```
16)function out = viewseq(file)

if file == 0
   [dfname, dpname] = uigetfile('*.dat', 'Select Start File');
   out = [dpname dfname];
   DescFile = [dpname dfname '.desc'];
else
   out = file;
   DescFile = [file '.desc'];
end



if dfname == 0
   return
end



szname = size(dfname);
sfname = size(file);

numcount = 1;

if file ~= 0
   for i =1:sfname(2) - 4
      newname(i) = file(i);
   end
else
   for i = 1:szname(2) - 4
      newname(i) = dfname(i);
   end
end



done = 0;


while ~done
   numcount = numcount+1;

   if file == 0
      data = agetdata([dpname dfname]);
      Desc = parse_descriptor(DescFile);
      dfname = [newname num2str(numcount) '.dat'];
      DescFile = [dpname dfname '.desc'];

   else
      data = agetdata(file);
      Desc = parse_descriptor(DescFile);
      file = [newname num2str(numcount) '.dat'];
      DescFile = [file '.desc'];

   end
```

```
   if data(1) == -1 & data(2) == 0
      done = 1;
   else
      plot(data);
      y = round(max(data));
      angle = sprintf('%d', Desc(9));
      string = ['Angle:  ' angle];
      text(100, y, string);
      drawnow;
      pause(1);
   end
end


disp('<...VIEWSEQ...Loaded last file of set...>');
```

17) waterfallA.m: calculate the impulse response as a function of rotation angle of the coral sample

```
clear all;
close all;

%% input base waveform
[bfname, bpname] = uigetfile('*.dat', 'Select Base (water only) File');
if bfname == 0
      disp('<...ANALYSIS...User Abort on reference file select...>');
      return;     %abort and return to window
end

BaseFile = [bpname bfname]

base = agetdata(BaseFile);

%% input wave waveform
wave = zeros(10000, 10);
for c=1:10
   [dfname, dpname] = uigetfile('*.dat', 'Select Data File');

   if dfname == 0    %the user pressed cancel
      disp('<...ANALYSIS...User Abort on data file select...>');
      return;     %abort and return to window
   end

   DescFile = [dpname dfname '.desc'];

   %read in and parse descriptor file
   Param = parse_descriptor(DescFile);
   if(Param(1) == -1)         %unable to read file
```

```
      DispString = sprintf('<...ANALYSIS...Unable to get descriptor
from file %s ..>', DescFile);
      disp(DispString);
      out = [-1 0];
      return;
   end

   DataFile = [dpname dfname]

   wave(:,c) = agetdata(DataFile);
end

TimePerSample = Param(1);
NumSamples = Param(2);
SampRate = Param(3);
HzPerSample = Param(4);

twave = 1:NumSamples;
for i = 1:NumSamples
   twave(i) = i*TimePerSample;
end

%set some parameters for windowing, downsampling, and zero padding
WindowTime = 20*10^-6;                %window length
WindowRes = 10^4;                   %frequency resolution of final,
zero padded signal in Hz/point
DesSampRate = 10*10^6;                %desired sampling rate after
downsampling
PaddedTime = 100^-6;                %time length of zero padded signal

%We want a time range of 100us to provide 10kHz frequency resolution
Param(6) = WindowTime / Param(1);   %number of points included in
window

%disp(Param);
OrigTimePerSample = Param(1);

%windowing, downsampling, zeropadding
factor = round(Param(3)/ DesSampRate);       %downsample by a factor of
'factor'

wbase = window_wave (base, Param(6));
%wbase = removeDC(wbase);

if DesSampRate < Param(3)                        %only downsample if
needed
   cwbase = compress(wbase, factor);
   TimePerSample = Param(1) * factor;
else
   cwbase = wbase;
   TimePerSample = Param(1);
end

cwbase = 0.4467*cwbase;

tcwbase = (1:1000)*10^(-7); %TimePerSample;
```

```
%zeropadding
cbase = [zeros(1, 1800) cwbase zeros(1, 2800)];

SampRate = 1/TimePerSample;

cwbase_size = size(cbase);

fbase = fft(cbase);
for i = 400:cwbase_size(2)-400
   fbase(i) = 0;
end

delta = .007*max(abs(fbase));

hbase = (1:1000)*10^(-7);

z = zeros(10, 500);
zz = zeros(10, 1000);

for c = 1:10
   %wwave = Window_wave (wave(:,c), Param(6));
   h = figure;
   plot(twave, wave(:,c));
   y = max(wave(:,c));
   text(twave(NumSamples)/3, y, 'Select Window Region', 'color',
'red');
   xlabel('Time(s)')
   ylabel('Voltage(v)')
   points = getclicks_t (gca, TimePerSample, NumSamples);

   wstart = points(1);
   wend = points(2);
   w2start = round (wstart/OrigTimePerSample);
   w2end = round(wend/OrigTimePerSample);

   %some error checking
   if w2start > NumSamples
      w2start = NumSamples;
   elseif w2start <= 0
      w2start = 1;
   end

   if w2end > NumSamples
      w2end = NumSamples;
   elseif w2end <= 0
      w2end = 1;
   end

   wwave = Window_waves(wave(:,c), w2start, w2end);

   %Remove DC component
   %wwave = removeDC(wwave);

   %Now Downsample to desired rate
   disp('<ANALYSIS...Downsampling...>');
   if DesSampRate < Param(3)                        %only downsample if
needed
```

```
      cwwave = compress(wwave, factor);
   else
      cwwave = wwave;
   end

   %Consider of Magnification of Signal: Wave 47dB, Base 7dB
   cwwave = 0.0044668*cwwave;
   wave_max = max(cwwave)

   size_cwwave = size(cwwave)
   %Pad the waveforms out to desired time
   cwave = [zeros(1, 2000) cwwave zeros(1, 3000-size_cwwave(2))];

   z(c,:) = cwave(2001:2500);

   %Fourier Transform of wave and base
   fwave = fft(cwave);
   for i = 400:cwbase_size(2)-400
      fwave(i) = 0;
   end


   wf = abs(fbase.^2)./(abs(fbase.^2).*fbase+delta);

   fh = fwave.*wf;

   h = ifft(fh);
   for i = 1001:5000
      h(i) = 0;
   end

   analy_h = abs(hilbert(h(1:1000)));
   IRF_MAX = max(analy_h)
   wave2 = conv( h, cbase);

   zz(c,:) = analy_h;


end


h1=figure;
theta = 0:20:180;
[X,Y] = meshgrid( tcwbase(1:500), theta);
x_size = size(X)
y_size = size(Y)
z_size = size(z)
waterfall(X, Y, z), %title('downsample waveform of wave in rotating the
sample approach')
xlabel('Time (s)'), ylabel('Angle'), zlabel('Amplitude(v)')
view(-10, 30);


h2 = figure;
[X,Y] = meshgrid( hbase, theta);
x_size = size(X)
y_size = size(Y)
```

```
z_size = size(zz)
waterfall(X, Y, zz), %title('Analytical Signal of Impulse Response
Function in Rotating the Sample Approach')
xlabel('Time (s)'), ylabel('Angle(degree)'), zlabel('Amplitude')
%view(-10, 30);
```

18) waterfallA3_t.m: calculate the impulse response as a function of rotation angle of the receiving transducer

```
clear all;
close all;

%% input base waveform
[bfname, bpname] = uigetfile('*.dat', 'Select Base (water only) File');
if bfname == 0
      disp('<...ANALYSIS...User Abort on reference file select...>');
      return;     %abort and return to window
end

BaseFile = [bpname bfname]

base = agetdata(BaseFile);

%% input wave waveform
wave = zeros(10000, 10);
for c=1:10
   [dfname, dpname] = uigetfile('*.dat', 'Select Data File');

   if dfname == 0    %the user pressed cancel
      disp('<...ANALYSIS...User Abort on data file select...>');
      return;     %abort and return to window
   end

   DescFile = [dpname dfname '.desc'];

   %read in and parse descriptor file
   Param = parse_descriptor(DescFile);
   if(Param(1) == -1)         %unable to read file
      DispString = sprintf('<...ANALYSIS...Unable to get descriptor
from file %s ..>', DescFile);
      disp(DispString);
      out = [-1 0];
      return;
   end

   DataFile = [dpname dfname]
```

```
   wave(:,c) = agetdata(DataFile);
end

TimePerSample = Param(1);
NumSamples = Param(2);
SampRate = Param(3);
HzPerSample = Param(4);

twave = 1:NumSamples;
for i = 1:NumSamples
   twave(i) = i*TimePerSample;
end

%set some parameters for windowing, downsampling, and zero padding
WindowTime = 20*10^-6;                 %window length
WindowRes = 10^4;                    %frequency resolution of final,
zero padded signal in Hz/point
DesSampRate = 10*10^6;                 %desired sampling rate after
downsampling
PaddedTime = 100^-6;                 %time length of zero padded signal

%We want a time range of 100us to provide 10kHz frequency resolution
Param(6) = WindowTime / Param(1);   %number of points included in
window

%disp(Param);
OrigTimePerSample = Param(1);

%windowing, downsampling, zeropadding
factor = round(Param(3)/ DesSampRate);       %downsample by a factor of
'factor'

wbase = window_wave (base, Param(6));
%wbase = removeDC(wbase);

if DesSampRate < Param(3)                        %only downsample if
needed
   cwbase = compress(wbase, factor);
   TimePerSample = Param(1) * factor;
else
   cwbase = wbase;
   TimePerSample = Param(1);
end

cwbase = 0.4467*cwbase;

tcwbase = (1:1000)*10^(-7); %TimePerSample;

%zeropadding
cbase = [zeros(1, 1800) cwbase zeros(1, 2800)];

SampRate = 1/TimePerSample;

cwbase_size = size(cbase);

fbase = fft(cbase);
for i = 400:cwbase_size(2)-400
```

```
   fbase(i) = 0;
end

delta = .007*max(abs(fbase));

hbase = (1:1000)*10^(-7);

z = zeros(10, 500);
zz = zeros(10, 1000);

for c = 1:10
   %wwave = Window_wave (wave(:,c), Param(6));
   h = figure;
   plot(twave, wave(:,c));
   y = max(wave(:,c));
   text(twave(NumSamples)/3, y, 'Select Window Region', 'color',
'red');
   xlabel('Time(s)')
   ylabel('Voltage(v)')
   points = getclicks_t (gca, TimePerSample, NumSamples);

   wstart = points(1);
   wend = points(2);
   w2start = round (wstart/OrigTimePerSample);
   w2end = round(wend/OrigTimePerSample);

   %some error checking
   if w2start > NumSamples
      w2start = NumSamples;
   elseif w2start <= 0
      w2start = 1;
   end

   if w2end > NumSamples
      w2end = NumSamples;
   elseif w2end <= 0
      w2end = 1;
   end

   wwave = Window_waves(wave(:,c), w2start, w2end);

   %Remove DC component
   %wwave = removeDC(wwave);

   %Now Downsample to desired rate
   disp('<ANALYSIS...Downsampling...>');
   if DesSampRate < Param(3)                        %only downsample if
needed
      cwwave = compress(wwave, factor);
   else
      cwwave = wwave;
   end

   %Consider of Magnification of Signal: Wave 47dB, Base 7dB
   cwwave = 0.0044668*cwwave;
   wave_max = max(cwwave)
```

```
    size_cwwave = size(cwwave)
    %Pad the waveforms out to desired time
    cwave = [zeros(1, 2000) cwwave zeros(1, 3000-size_cwwave(2))];

    z(c,:) = cwave(2001:2500);

    %Fourier Transform of wave and base
    fwave = fft(cwave);
    for i = 400:cwbase_size(2)-400
       fwave(i) = 0;
    end


    wf = abs(fbase.^2)./(abs(fbase.^2).*fbase+delta);

    fh = fwave.*wf;

    h = ifft(fh);
    for i = 1001:5000
       h(i) = 0;
    end

    analy_h = abs(hilbert(h(1:1000)));
    IRF_MAX = max(analy_h)
    wave2 = conv( h, cbase);

    zz(c,:) = analy_h;


end


h1=figure;
theta = 0:20:180;
[X,Y] = meshgrid( tcwbase(1:500), theta);
x_size = size(X)
y_size = size(Y)
z_size = size(z)
waterfall(X, Y, z), %title('downsample waveform of wave in rotating the
sample approach')
xlabel('Time (s)'), ylabel('Angle'), zlabel('Amplitude(v)')
view(-10, 30);


h2 = figure;
[X,Y] = meshgrid( hbase, theta);
x_size = size(X)
y_size = size(Y)
z_size = size(zz)
waterfall(X, Y, zz), %title('Analytical Signal of Impulse Response
Function in Rotating the Sample Approach')
xlabel('Time (s)'), ylabel('Angle(degree)'), zlabel('Amplitude')
%view(-10, 30);
```

19)
```
function out = window_wave(wave, numpoints)

numpoints

MaxWaveVal = max(wave);

WaveMaxPoint2 = find(wave == MaxWaveVal);

WaveMaxPoint = WaveMaxPoint2(1);


%window the data
disp('<WINDOW_WAVE...Windowing Data...>');

WaveStartPoint = WaveMaxPoint - round(numpoints);
WaveEndPoint = WaveStartPoint + round(2*numpoints);
%WaveStartPoint = round(numpoints/6);
%WaveEndPoint = WaveStartPoint + round(numpoints/2);

WaveSize = size(wave);

%some error checking
if WaveStartPoint < 1
   disp('<WINDOW_WAVE...Waveform window too close to start of
data...Adjusting>');
   WaveStartPoint = 1;
end


if WaveEndPoint > WaveSize(1)
   disp('<WINDOW_WAVE...Waveform window too close to end of
data...Adjusting>');
   WaveEndPoint = WaveSize(1) - 1;
end


%Window waveforms
for i = WaveStartPoint:WaveEndPoint
   out((i - WaveStartPoint) + 1) = wave(i);      %wwave = windowed wave
end
```

20)
```
function out = window_waves (wave, WaveStartPoint, WaveEndPoint)

% window waveforms
for i = WaveStartPoint:WaveEndPoint
   out((i-WaveStartPoint) + 1) = wave(i);
end
```

```
21) function out = zero_pad(wave, DesSampRate)

wavesize = size(wave);

disp('<ZERO_PAD...Zero Padding...>');
TimeWave = wavesize(2) * (1/DesSampRate);
TimeToAdd = 100*10^-6 - TimeWave;               %amount of time to zero
pad


for i=1:wavesize(2)
   out(i) = wave(i);
end

for i=wavesize(2):(TimeToAdd*DesSampRate + wavesize(2))
   out(i) = 0;
end
```