Scanning Measurement Testbed for

Advanced Nondestructive Evaluation

by

Michael R. Horne

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

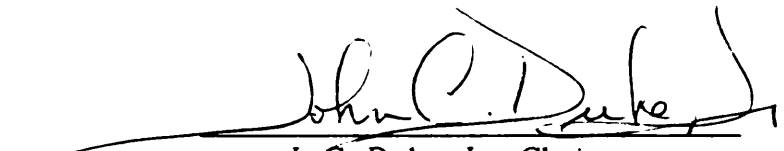
in partial fulfillment of the requirements for the degree of

Master of Science

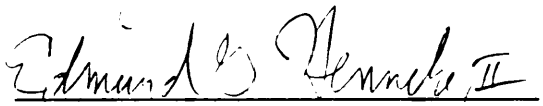
in

Engineering Mechanics

APPROVED:



J. C. Duke, Jr., Chairman



E. G. Henneke, II

O. H. Griffin

March 1990

Blacksburg, Virginia

c.2

LD
5655
V855
1990
H662
c.2

Scanning Measurement Testbed for

Advanced Nondestructive Evaluation

by

Michael R. Horne

J. C. Duke, Jr., Chairman

Engineering Mechanics

(ABSTRACT)

New materials and manufacturing processes, and the quest for economy and user safety, have necessitated the development of nondestructive testing methods to quantify the life and reliability of a product during manufacture and service. Described herein, is a testbed to be used in the research and development of these testing methods. A brief motivation for using ultrasonics applied to nondestructive evaluation is followed by a chapter on the feasibility of using a unique testing method and animated data presentation on advanced composite materials. This testing method, conceived by the author, utilizes oblique injection of ultrasound into the specimen. Several cycles of the ultrasonic waveform radiated from the specimen downstream of the injection area is digitized and recorded. The data has three independent dimensions; cartesion location and time. The time variable is the key to the presentation of the data as an animated two dimensional image. It was this work that illustrated the need for a flexible scanning imaging research testbed, not only for the discussed method, of which it is an integral part, but for advanced development of other techniques. Software development and integration of off-the-shelf parts into a unified computer controlled testing facility is the contribution by the author in the second phase of this research. Chapters on the description of the system, an example showing the capabilities of the system analogous to traditional ultasonic C-scanning, accomplishments, and a look to the future conclude this thesis. The appendices include listings of the programs developed for the system, a manufacturer address list. A videotape of the animation data presentation is included as a second volume of this thesis.

Acknowledgements

I would like to acknowledge:

Deep Appreciation of my parents, for many things, but most recently for their understanding and patience (they can now confidently tell people that I have gotten my degree); Cindy, for being my sister (one is enough!...only kidding), and for not having kids yet (I'm not sure I'm ready to be an uncle, Hear that, Mark?);

Thanks to Dr. John Duke, Jr., as my committee chairman, for his "gentle" persuasion techniques to help me to finish this thesis (I want him for my Ph.D. committee!?); Thanks to Dr. E. G. Henneke, II and Dr. O. H. Griffin for serving on my committee;

Much gratitude to all the folks in the MCIS at NASA Langley for their wonderful support while I was there;

Scott Bartlett, for his friendship and his "computer-ese"; Paul Grosskopf for continually bringing up entrepreneurial ideas; Laura Perry, for using my computer, thereby forcing me to figure out how to do things that I didn't know, and also for keeping my TV; Roz Rutstein, (a non-engineering person, formally) for actually wanting to put herself through the torture (she says adventure) of reading my introduction;

All the people who would prefer that I do their work instead of mine;

The University for continuing to confirm my belief that a body of knowledge is a body of knowledge, nothing more, nothing less. Knowledge, in itself, does not create problems, nor does it solve them. Creative application of knowledge is born, not of adept remembrance and regurgitation, but of going beyond the pursuit of details to the pursuit of concepts. One thread does not know the tapestry.

Table of Contents

Volume I

1 Purpose 1

1.1 Introduction 1

1.2 Direct Objective 2

1.3 Indirect Objective 4

2 Feasability Study of Ultrasonic Evaluation of Material Performance 7

2.1 Introduction 7

2.2 Initial Approach 10

2.3 Probe Fixture Design 12

2.4 Equipment Setup 14

2.5 Final Specimen Design and Test Configuration 16

2.6 Analytic Signal Analysis 23

2.7 Video Format Presentation 24

2.8 Specific Description of Volume II Movie 27

3 Modular System Description 28

3.1 Introduction 28

3.2 Hardware 32

3.2.1 Requirements 32

3.2.2 Controller 34

3.2.3 Controller I/O 35

3.2.4 Signal Excitation 37

3.2.5 Motion Control 39
3.2.6 Signal Detection and Conditioning 40
3.2.7 Data Acquisition 41
3.3 Software Modules 44
3.3.1 Requirements 44
3.3.2 Stepper Motor Drivers 48
3.3.3 Data Acquisition Drivers 48
3.3.4 GPIB Device Drivers 49
3.3.5 Video Drivers 50
3.3.6 Data Acquisition Tasks 51
3.3.7 Stepper Motor Control Tasks 52
3.3.8 Scanning Tasks 53
3.3.9 GPIB Tasks 55
3.3.10 Menu 55
3.3.11 Main 57
3.4 External Software for Analysis and Presentation 58
4 Illustrative Example 59
4.1 Introduction 59
4.2 Equipment Configuration 59
4.3 Results 59
4.4 Comparisons 60
5 Accomplishments 64
5.1 Feasibility Study 64
5.2 Testbed Development 65
6 Future Work 66
6.1 Hardware 66

6.2 Analysis 67

6.3 Applications 68

References 69

Appendix A. Program Listings 73

Appendix B. Equipment Suppliers List 108

Vita 110

Volume II

Movie (See Section 2.8 for Description) ..... Video-cassette

List of Illustrations

Figure 1.1 Block Diagram of an Analog Testing System 3

Figure 2.1 Various Probes Used for Ultrasonic Probing of Composite Materials 11

Figure 2.2 Ultrasonic Scanning Fixture 13

Figure 2.3 NASA Equipment Setup 15

Figure 2.4 Fixture Used to Impact Composite Specimens 17

Figure 2.5 Scanning Grid Pattern 19

Figure 2.6 Slowness Surfaces: Lucite-Unidirectional Graphite/Epoxy Interface 21

Figure 2.7 Energy Flux and Particle Displacement Deviation for Unidirectional Graphite/Epoxy 22

Figure 2.8 Typical Waveform and the Magnitude of the Analytic Function of that Waveform 25

Figure 3.1 Scanning System Function Diagram 30

Figure 3.2 Hardware Schematic 33

Figure 3.3 Software Hierarchy Diagram 45

Figure 3.4 Setup Screen for Data Acquisition Parameters 54

Figure 4.1 Scan of a Quarter: 200 pnt. x 200 pnt. x .005 in./pnt. 61

Figure 4.2 Scan of a Quarter: 500 pnt. x 500 pnt. x .002 in./pnt. 62

1 Purpose

1.1 Introduction

Before discussing the how and what of this research, let us look at the why; the philosophy that lays the foundation for this research. Why do we use ultrasonics to interrogate and evaluate mechanical response of a material when in many instances other nondestructive methods such as x-ray radiography can give far better resolution of the "flaws" [1.1]. The key words are "mechanical response". One of the parameters of the deformation aspect of mechanical response is the velocity at which the deformation propagates. Therefore, as it has been noted by Kolsky [1.2], all external loads on a structure do not affect all parts of the structure simultaneously. These loads are distributed through the material by dynamic deformation or stress waves. For example, consider an elastic bar of length L, with one end against a rigid wall while the other end is being slowly compressed and then held. Since every part of the bar will be in compression at equilibrium, the contention is that the "front" of a stress wave with a wavelength of 4L travels along the bar redistributing the displacement. Let us assume that this stress wave propagates by that same mechanism that a short wavelength (compared to the dimensions of the bar), high frequency ultrasonic stress wave propagates, i.e., changes in interatomic forces due to changes in interatomic distances and the drive to reduce those forces. Then, on a larger scale, the

patterns of energy distribution of ultrasonic waves due to differences in the material should be related to the redistribution of the stresses due to loads on the structure. Hence, using ultrasonic nondestructive evaluation it may be possible to quantify the similarities and relate them to the capability of the structure to withstand loading.

1.2 Direct Objective

The primary goal of this thesis was to design and construct a working, flexible, computer controlled, scanning measurement testbed for research of advanced nondestructive testing and evaluation of advanced materials. The resulting configuration utilizes, but is not limited to, ultrasonic interrogation of these materials. Realization of this goal is made possible by the advent of powerful and portable digital electronics technology, in the last decade, for use in data acquisition and machine control. In the past, techniques which utilized analog circuitry that could discriminate, filter, and modulate analog signals (for example, as in Fig. 1.1 [1.3]), have been used for nondestructive testing of materials. Such techniques are still useful in some situations because they can be fast and portable, once the quantity to be detected and the method of detection is known. However, in a research environment where development of new techniques is the objective, flexibility is of paramount importance. Interestingly, digital technology is advancing to the point where inexpensive, fast, and portable test equipment, with flexibility implemented in software, is going into the field and displacing traditional testing equipment. Chapter 4 presents some results, from the system developed, which are equivalent to the output of a traditional analog C-scan ultrasonic scanning

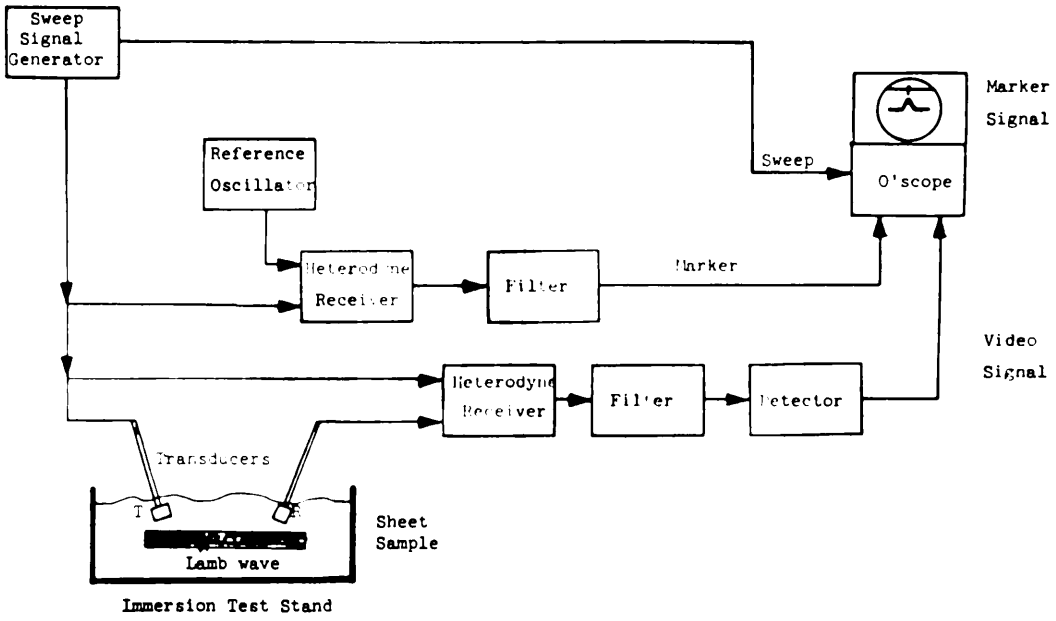


Figure 1.1 Block Diagram of an Analog Testing System

unit, even though the system has greater potential.

The system developed has the ability to easily change the type of data analysis, the scanning configuration and the presentation of the results due to the careful attention paid to the modularity of the controlling program. This was accomplished by writing libraries of primitives (a primitive is a small subroutine that only does one very basic task), for the mechanical control of the stepper motors, control of the data acquisition, and input/output control, from which modules for scanning, peripheral control, and data acquisition and analysis were created. A detailed discussion of this system is presented in Chapter 3.

1.3 Indirect Objective

At one time materials were simple and, subsequently, testing methods to determine the design parameters were simple. Designs were built with large factors of safety, but, failures still occurred. Design disciplines, failure modeling, and material rejection criteria were developed around the concept of critical parameters such as fatigue loading, stress concentrations and flaw size [1.4, 1.5, 1.6, 1.7]. Testing methods that focused on the flaw sizes and microstructural properties of the material became necessary for safe designs that minimized the use of resources [1.8, 1.9, 1.10]. Light weight, high strength and toughness have become desirable, if not the primary parameters for selection of materials used in all load bearing engineering designs. Super materials, composites [1.11], ceramics, and advanced production methods including net shape parts that require minimal or no machining before use, and the increasing sophistication and proliferation of products that would be

injurious to life and limb if they failed, have necessitated the development of nondestructive testing methods that can easily quantify the life or reliability of a part during production and service.

Ultrasound was found, historically, to be a noninvasive and safe way of "looking" into materials for flaws and defects. Indeed, demands for safety caused ultrasonic interrogation to become very important in the testing of the most sophisticated design: man. But, the lack of clarity of quantifying internal defects and inhomogeneities, due to the dispersion and diffraction of mechanical waves propagating in a material, has hindered the use of ultrasonics in "critical flaw size" imaging of engineering materials.

The secondary goal of this study, with an eye toward further research, was the development of an ultrasonic scanning and imaging technique that could be used to investigate the physics of the local mechanical energy dispersion properties of a load bearing member. Preliminary research with such an objective in mind was conducted by the author, at the NASA Langley Research Center, as outlined in Chapter 2 of this discussion.

Regarding the secondary nature of this goal, it is not within the scope of this thesis to present an extensive treatise on wave propagation in solids, especially in composites, nor the latest advancements in ultrasonic evaluation of composites. For that, the reader is urged to peruse the following references [1.12, 1.13, 1.14, 1.15].

The results of this preliminary research suggest that this technique could then be used in the nondestructive evaluation of the life and performance characteristics of advanced composite materials, which are not well represented by traditional nondestructive testing and evaluation

techniques. A consequence of this research, also, was the recognition that further development would require access to a scanning measurement testbed. The project described by this thesis has provided such a testbed.

2 Feasibility Study of Ultrasonic Evaluation of Material Performance

2.1 Introduction

The work described in this section was conducted at the Materials Characterization and Instrumentation Section of the General Research Instrumentation Branch at the NASA Langley Research Center as a result of the author being funded by a NASA fellowship. This research lead the author to the conclusion that a flexible research testbed was needed for the completion of his Masters Degree and the pursuit of his Doctoral Degree.

The purpose behind this feasibility study was to establish the features of an ultrasonic scanning and imaging technique that could characterize the local mechanical energy dispersion properties of a load bearing member.

The goal was to answer the following questions. Is it possible to excite and maximize "quasi-longitudinal" stress wave propagation and/or particle displacement along a primary structural loading direction from a different direction? With such excitation, is it possible to detect energy scattered from areas of low energy transmissibility? If that detection is possible, can the scattered energy be related to the ability of that area to carry load?

The reasoning behind these questions is as follows. One should attempt to maximize the wave energy propagation and the particle displacement in the loading direction, e.g., in the plane of the plate for plate structures. This

should result in a technique that is more sensitive to variations in the material, parallel to loading direction, which relate to load transfer around "weak" areas that would typically fail under loading. Thus, the technique would not describe an imperfection, but, would characterize the load carrying properties of that imperfection, hence, leading to an "image" that could inherently quantify life of the part. A simple example would be an isolated fiber with a break. The break, itself, would not be able to carry a tensile load and could be able to carry a compressive load. Mechanical wave propagation across the break would be similarly affected. The part of the wave that would open the break would not be transmitted across the break. The part of the wave that would close the break would be transmitted, possibly at reduced amplitude, across the break. Therefore, if one were able to image the local displacements at the break, one would see a change in the characteristics and energy of the wave propagating downstream from the break. However, since a fiber does not behave in a composite independently, the surrounding matrix is affected by the break. If the matrix is well bonded to the fiber, hence maintaining continuity of displacement across the interface, shear stresses transfer the load around the break to the other part of the fiber and other fibers. Local stiffness variations also affect the local velocity and direction of wave propagation in that area. Thus, the energy is scattered in other directions, subsequently affecting the displacement of the surface of the plate in the area of the propagating wave. It also follows that any "defect" that would affect stress transfer and hence load carrying capability, such as fiber/matrix disbond, porosity, matrix cracks, fiber breaks, and local variations in the volume fraction and mechanical properties of the fibers and matrix would affect the wave propagation and subsequently the

plate surface displacement.

Criteria were established for the development of a technique that could lead to answers to the above questions and are reported in the following paragraphs.

Non-contact should be maintained for the sensing element of the scanning probe. Contact could significantly change the boundary conditions at the surface of the plate and would affect the wave propagation in that area.

It is not necessary to maintain non-contact for the wave initiation element of the scanning probe, as long as the appropriate wave is developed and allowed to propagate freely in a small section of the plate. This idea was generated by the concept of the superposition of partial waves [2.1] which, stated simply, says that one can derive the equations for plate waves by superimposing the solutions for all possible waves traveling down the plate while reflecting off the surfaces. In actuality, one need only start with an internal longitudinal wave incident on the surface. At the first reflection, mode conversion creates both longitudinal and shear waves which give rise to more longitudinal and shear waves upon subsequent reflection. This process continues until a saturated state is reached and a fully developed plate wave propagates. In an anisotropic material the situation is even more complicated. For an incident quasi-longitudinal wave, the potential exists for three reflected waves; a quasi-longitudinal and two transverse or quasi-transverse modes.

The sensing element should have an active diameter smaller than half of the wavelength of the propagating wave. Elements larger than the wavelength average the displacement over the area of the element, thus giving

false indications of the total energy received by that element. This problem of phase sensitivity has been investigated by Heyman [2.2] who proposed and developed phase insensitive transducers. However, if one would like to more accurately map the displacement of a surface it would make more sense to use one small element and measure at more locations. Either way, this is an essential criterion for any technique that analyzes the energy in the waveform.

2.2 Initial Approach

To facilitate this phase of the research, an empirical approach was taken. Significant effort was devoted to developing an empirical understanding of wave propagation in composite materials, without doing an intensive parametric investigation of all the variables. Much experimentation was done with various hand-held rudimentary probe configurations, an analog oscilloscope, and several types of graphite/epoxy specimens. The specimens had various types of "damage" such as artificial porosity, cut fibers, and impact damage. Several of the probe fixtures are shown in Fig. 2.1. By doing simple experiments, such as

- sweeping the probe fixture over areas of damage while watching the waveform change on the oscilloscope,
- probing the edge of the plate with the receiving transducer while changing the incident angle of the sending transducer,
- propagating waves with and across the fiber direction,
- varying the distance between excitation and receiving transducers

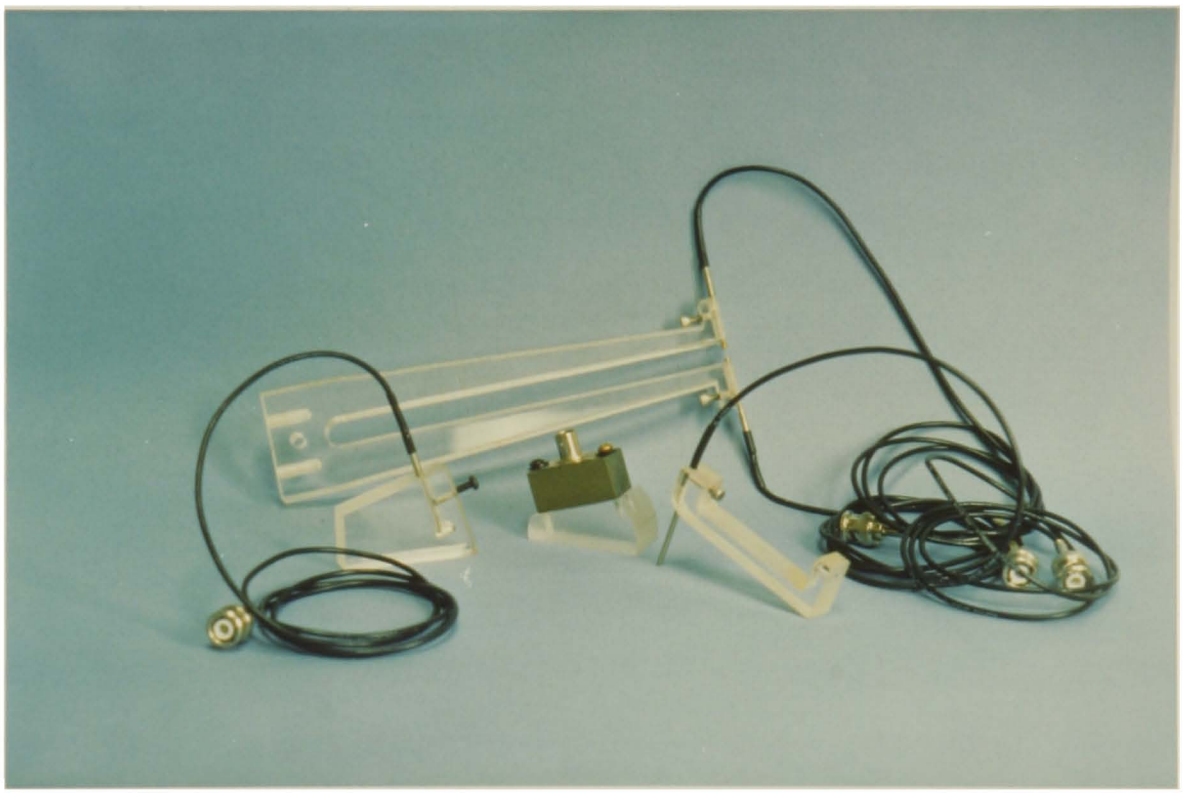


Figure 2.1 Various Probes Used for Ultrasonic Probing of Composite Materials

while keeping one stationary,

- looking at the displacements on opposite sides of the plate with one sender and two pointducer receivers (very small diameter undamped piezoelectric transducers),
- and changing such parameters as size of the transducers and frequency of the input signal,

it was discovered that the criteria outlined in section 2.1 were feasible and that the following testing approach was reasonable to pursue.

2.3 Probe Fixture Design

Following the criteria above, a probe fixture, as shown in Fig. 2.2, was developed. Using piezoelectric transducers for both the sensing and wave initiation elements of the scanning probe allowed flexibility in the type of excitation produced, which is a function of the electronics driving the transducer. The 1 Mhz center frequency sending transducer was attached to a plastic (acrylic) variable angle block that allowed the incident angle, as measured from the vertical, to be set from 0 to 80 degrees. The angle block was held in sliding contact with the plate by the weight of the angle block and transducer. The arms attached to the angle block held the receiving pointducer (1 Mhz, .030 inch diameter transducer) at a distance of approximately 2.25 inches measured from the vertical centerline of the angle block. The pointducer could be positioned at any angle incident to the plate and also any distance, up to .5 inches from the plate. The fixture and specimen were completely submerged in water, which coupled the transducers to the surface of the plate and maintained constant boundary conditions on all

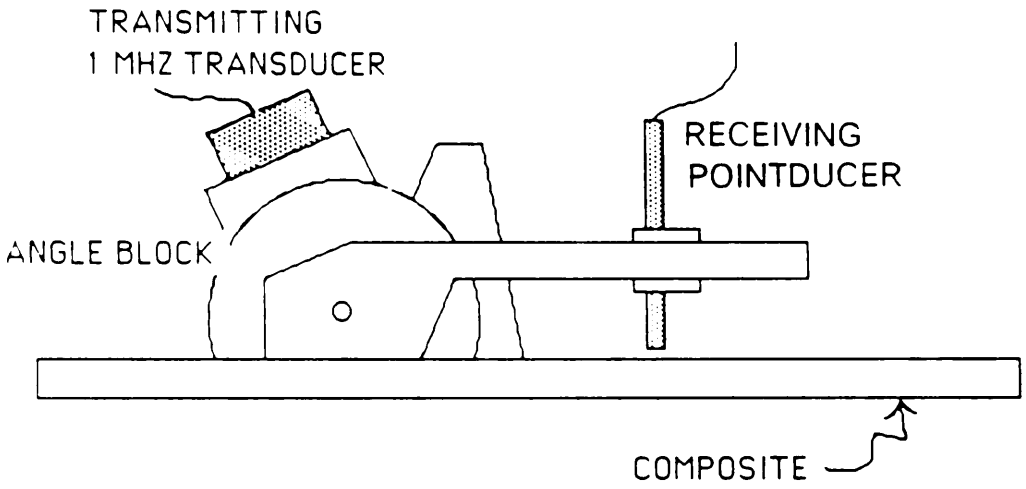


Figure 2.2 Ultrasonic Scanning Fixture

surfaces of the plate. The fixture was attached to the bridge of a typical ultrasonic testing tank. This allowed a rastered type of probe movement over the surface of the plate, similar to the motion used when performing a standard ultrasonic C-scan.

2.4 Equipment Setup

The equipment was setup in the manner shown in Fig. 2.3. The modularity shown in the diagram is real. The equipment was either stand-alone or plugged into a rack mounted backplane. Another feature was the IEEE 488 (GPIB or HPIB) interface capabilities of much of the stand-alone equipment. The mainframe VAX computer in that section also had IEEE 488 capabilities via an RS232/488 converter. Hence, the laboratory was modular with relatively consistent communication between modules, and had computing power to support it, thus allowing flexibility in test design. Appendix B contains a list of the equipment that was used.

The primary signal sources were pulsers that could output a voltage spike and function generators that could output continuous or tone burst (gated) sine, triangle or square waves. Depending on the source, the signal could be fed into a power amplifier to boost the amplitude to several hundred volts before exciting the transducer. This increases the excitation in the specimen which increases the signal to noise ratio at the receiving transducer. However, some amplification is usually still necessary before the received signal is fed into the recording digital oscilloscope or digital voltmeter. Note, that until this point the signal had been analog; analog at the source, analog transducers, and the analog wave propagation at the scale we are investigating.

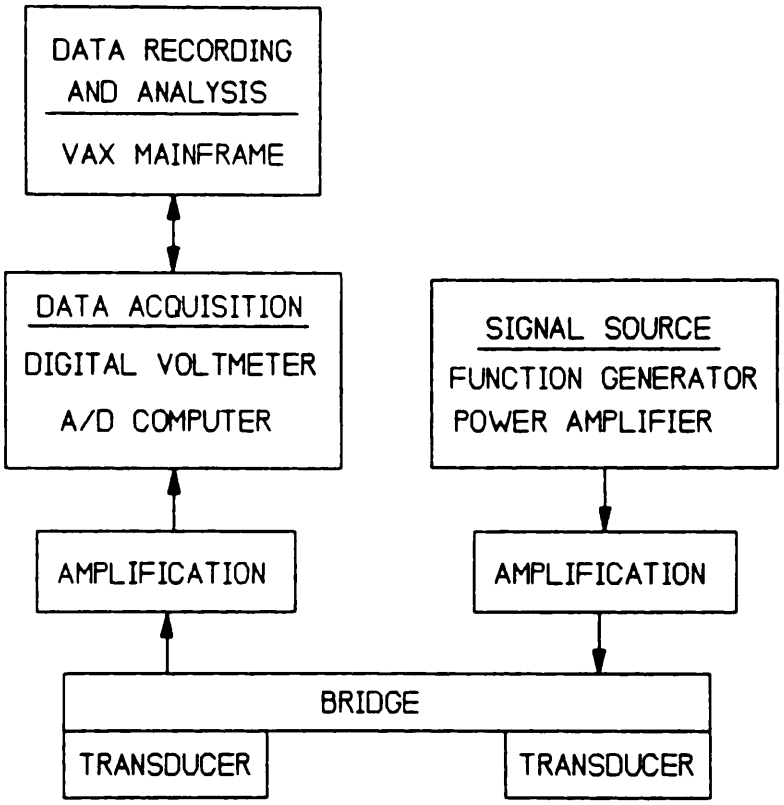


Figure 2.3 NASA Equipment Setup

Now, the digital oscilloscope could digitize portions of the waveform and send it to a computer via the IEEE 488 bus for analysis or storage. The digital voltmeter could also communicate over the IEEE 488 bus with the computer. The computer, in addition to data storage and analysis, controlled and synchronized all the equipment and the electronics which controlled the motion of the scanning bridge. Here is where the flexibility of the system breaks down. The bridge electronics could only understand commands from the computer to start, stop, step (forward or back in the X or Y direction). The step size could only be changed by manually reentering an assembly language controller program with new parameters. There were no output capabilities to flag the computer whether the bridge was moving or not and the motor speed could not be changed. These points are critical to the purpose of this thesis, which is the development of a flexible measurement testbed. The appropriate amount of pause time had to be imbedded in the main computer program so that the data acquisition would stay in synchronization with the bridge controller. Also, one could not do adaptive scanning where the scanning parameters could be changed in critical areas during the scan.

2.5 Final Specimen Design and Test Configuration

Since this was a feasibility study it was appropriate that a simplistic approach was taken in specimen design and that one test configuration was followed to final data analysis and presentation.

Three small areas each, of four thicknesses of unidirectional 5208/T300 graphite/epoxy composite plate, of 8, 16, 32, and 64 plies, were constrained to prevent end-to-end splitting using a fixture as shown in Fig. 2.4, and impacted

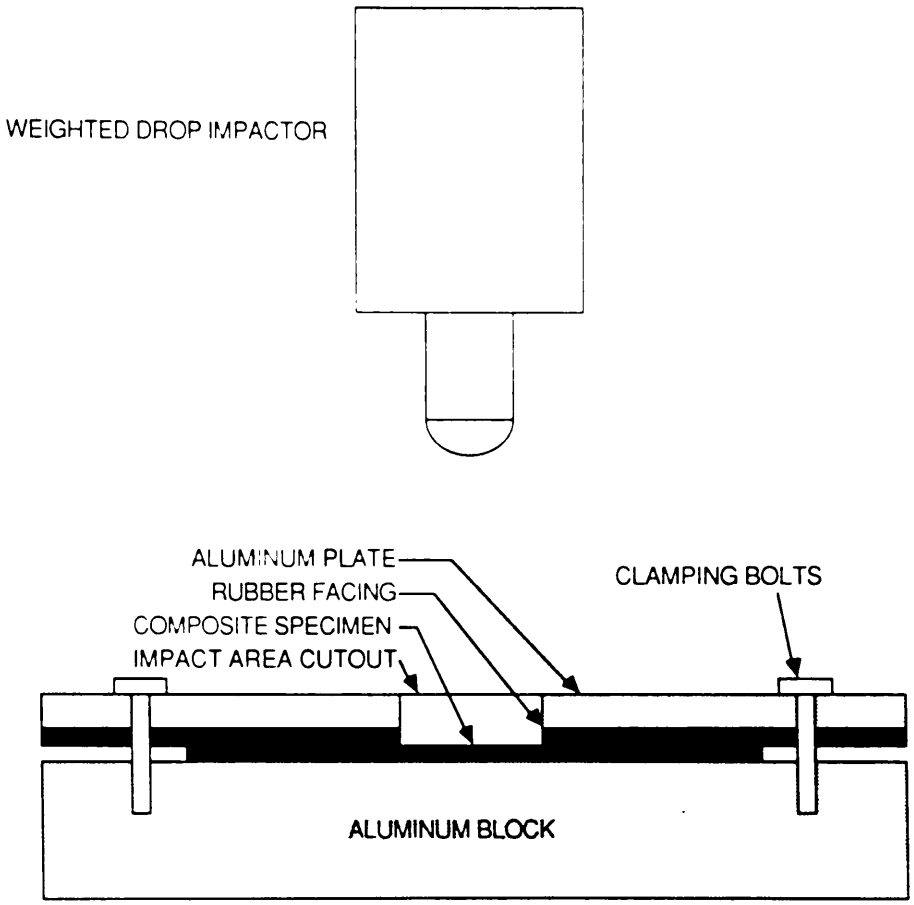


Figure 2.4 Fixture Used to Impact Composite Specimens

in each area at one of three energy levels; 1, 4, and 7 ft-lb. An area of 1.5 in. by 1.5 in. centered around each impact zone was scanned in a 50 by 50 grid pattern with a .76 mm resolution as shown in Fig. 2.5, using the probe fixture described above. The sending transducer excitation was a single cycle tone burst of 1 MHz. Use of a single cycle, instead of multiple cycles, is the result of a phenomenon which is related to the one noted in Section 2.7, i.e., distinguishable features in the image degrade at a certain point on the time axis of the signal. It was discovered that when more cycles were added to the tone burst the very front of the wave (approximately 1.5 cycles) did not change in amplitude or character. Hence, there was no need for more cycles.

During the exploratory stage of testing it was discovered that the maximum signal amplitude in the plane of the plate occurred when the angle of incidence of the launched wave corresponded to the "critical" angle [1.2] for a longitudinal wave in lucite (acrylic) launched at the interface of an isotropic material that has properties equivalent to the properties in the fiber direction of graphite/epoxy. This angle, as calculated from Snell's Law, the following equation, is approximately 18°.

$$\sin\theta = \frac{\upsilon_{\text{lucite}}}{\upsilon_{\text{UNI GR/EP II fiber}}}$$

where: $\upsilon \equiv$ velocity of longitudinal wave

$\theta \equiv$ angle measured from the normal

Theoretically, at the critical angle for an isotropic material, the predominant mode propagated down the plate would be a surface wave with a minimal depth of penetration into the plate. However, the graphite/epoxy is

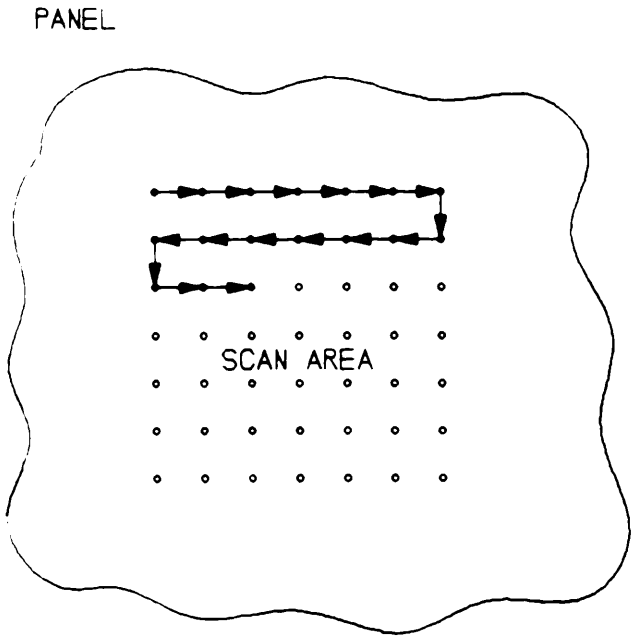


Figure 2.5 Scanning Grid Pattern

anisotropic with noncircular slowness surfaces, as can be seen in Fig. 2.6 (borrowed from [2.3] and added to for lucite), which is representative of the wave propagation of this testing configuration. Note that the normal to a slowness surface is the energy flux propagation direction for the resulting refracted wave [1.12] as indicated by the arrows on the plot. It can be seen that for a small range of incident angles near the calculated "critical" angle, there is a range of quasi-longitudinal waves refracted, which are not surface waves, but have energy propagation directions predominantly parallel to the surface of the plate. Also, Stiffler [2.4] and Kriz [2.5] show that the particle displacement deviation (measured from the wave propagation direction, as is energy flux deviation) is approximately the same as the energy flux deviation as indicated in Figure 2.7; hence, the particle displacements are predominantly parallel to the surface of the plate. Note that θ in the figure as denoted by Stiffler and Kriz refers to an angle measured from the fiber direction in the plane of the plate. However, due to the transverse isotropy of unidirectional graphite/epoxy, it does not matter if θ measured from the fiber direction is parallel or perpendicular to the plate. Also, note that θ in the figure is 90° minus the incident angle. Further note, that the other waves produced, transverse and quasi-transverse (which are slower than the quasi-longitudinal), are also propagating at steeper angles to the surface (both the wave propagation and the energy flux directions) and therefore have an even slower progression along the plate than the quasi-longitudinal. Hence, the very front of the wave will not have a wavefront (the constant phase plane normal to the wave propagation direction) normal to the plate center plane but should have energy flux and particle displacement parallel to the center plane. This is an important feature that will be discussed again in section 2.7. Since real

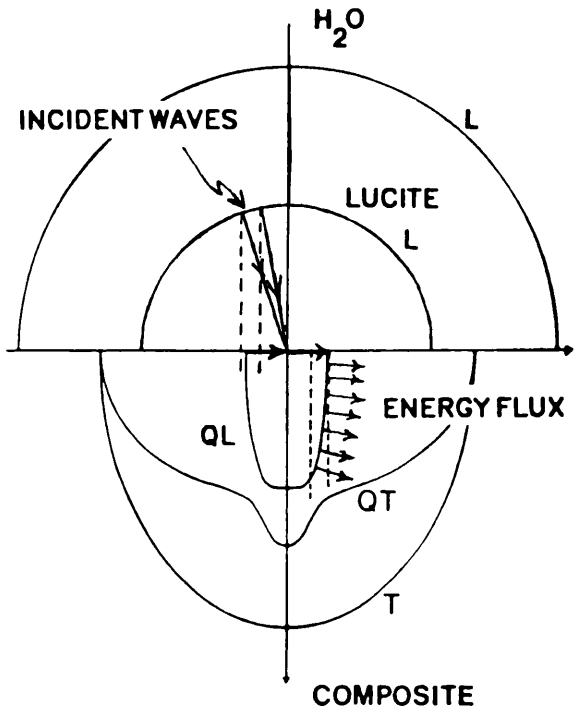
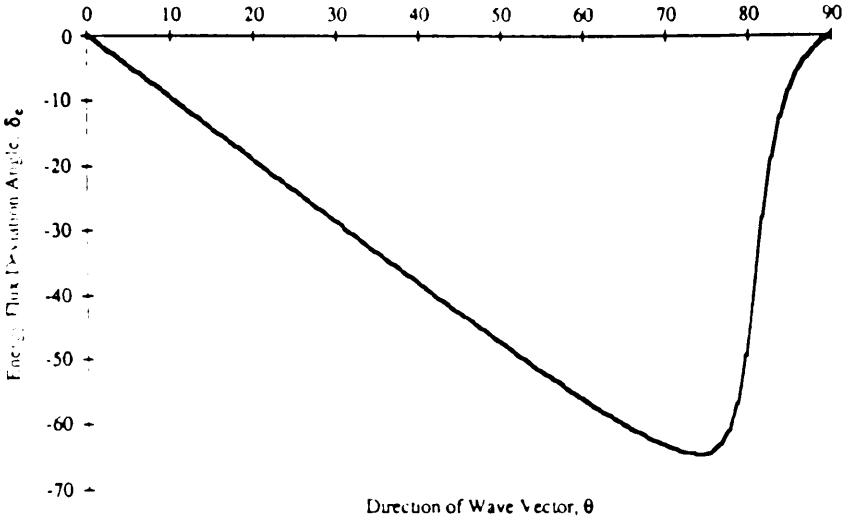
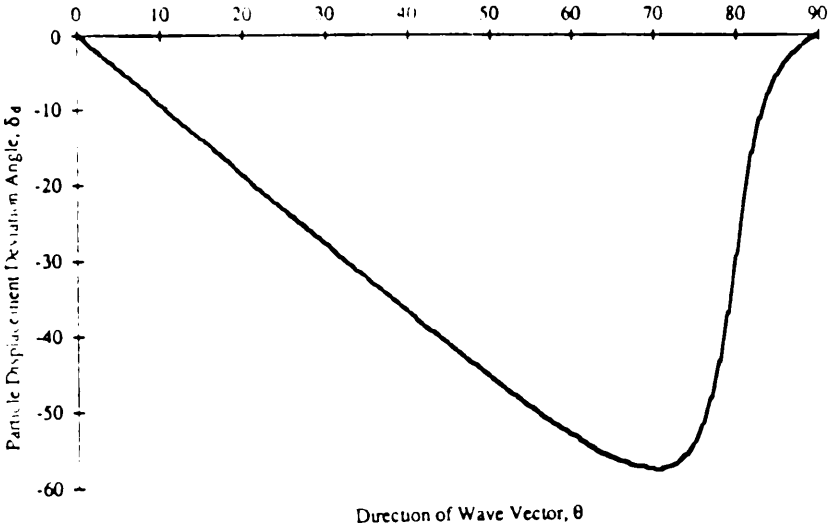


Figure 2.6 Slowness Surfaces: Lucite-Unidirectional Graphite/Epoxy Interface



Energy flux deviation angle for quasi-extensional wave in unidirectional graphite epoxy laminate.



Particle displacement deviation angle for quasi-extensional wave in unidirectional graphite epoxy laminate.

Figure 2.7 Energy Flux and Particle Displacement Deviation for Unidirectional Graphite/Epoxy

transducers create Gaussian and not perfectly plane wave profiles (hence waves are impinging on the graphite/epoxy surface through a range of incident angles) it is becomes clearer how one can measure a large amplitude in the plane of the plate ("quasi-bulk" wave?). This "waveguide" effect can be exploited as will be commented on in Chapter 6.

The receiving pointducer was adjusted to an incident angle of 0° and offset from the plate approximately 2 mm. Waveforms measured at that distance and greater were essentially the same except for amplitude, whereas, closer to the plate the waveform changed character, probably due to shear motion that had not yet died.

2.6 Analytic Signal Analysis

The initial attempt to glean information from a complicated waveform might be to rectify and filter the data so that the result would be an envelope of the waveform from which one could get an approximation of when and where the bulk of the wave or possibly a different mode of the propagation existed. However, a better method exists which does not alter or lose information embedded in the wavetrain. An enlightening discussion of the analytic function of a signal, which is the basis for this method, has been presented by Bracewell [2.6]. In short, the analytic function of the signal is a complex function, where the real part is the original waveform and the imaginary part is the Hilbert transform of the original waveform. One feature of this derivation is that the magnitude of the analytic function is a envelope of the waveform and has been shown by Gammel [2.7] to be equivalent to the rate of energy arrival and related to the total energy density of the wave.

Figure 2.8 shows a typical waveform encountered during this research and the magnitude of the analytic function of that waveform.

2.7 Video Format Presentation

A significant portion of the received waveform was digitized and stored for each scan location. This produces an immense amount of data, the analysis of which posed the question; how does one present such data in a meaningful format that is intuitively and quickly digested. For instance, a scan grid of 50 by 50 locations with 512 samples taken at each location creates over 1.2 million values. An analogous problem existed when going from A-scan data collected at one spatial location to C-scan data which is collected at many locations and should be presented simultaneously to easily visualize the differences between each location. The solution for presenting C-scan data was to present the data in an image format; essentially a "picture" of the mechanical behavior of the inside of the plate.

The data collected in this research was not only dependent upon spatial location, like C-scan data, but also upon time. Hence, the data is four dimensional (time, x location, y location and amplitude) which is most difficult to present simultaneously. The key is that this data vary with time. The solution is to use an existing format where three dimensional image data (intensity, x location, y location) vary with time: animation. A software package at NASA could take standard pulse-echo C-scan waveforms and create a "movie" which could then be output to a video recorder [2.8]. Because one frame of such a movie of a pulse-echo waveform is a slice in time, hence an image of a plane at a certain depth in the material, the movie is essentially a

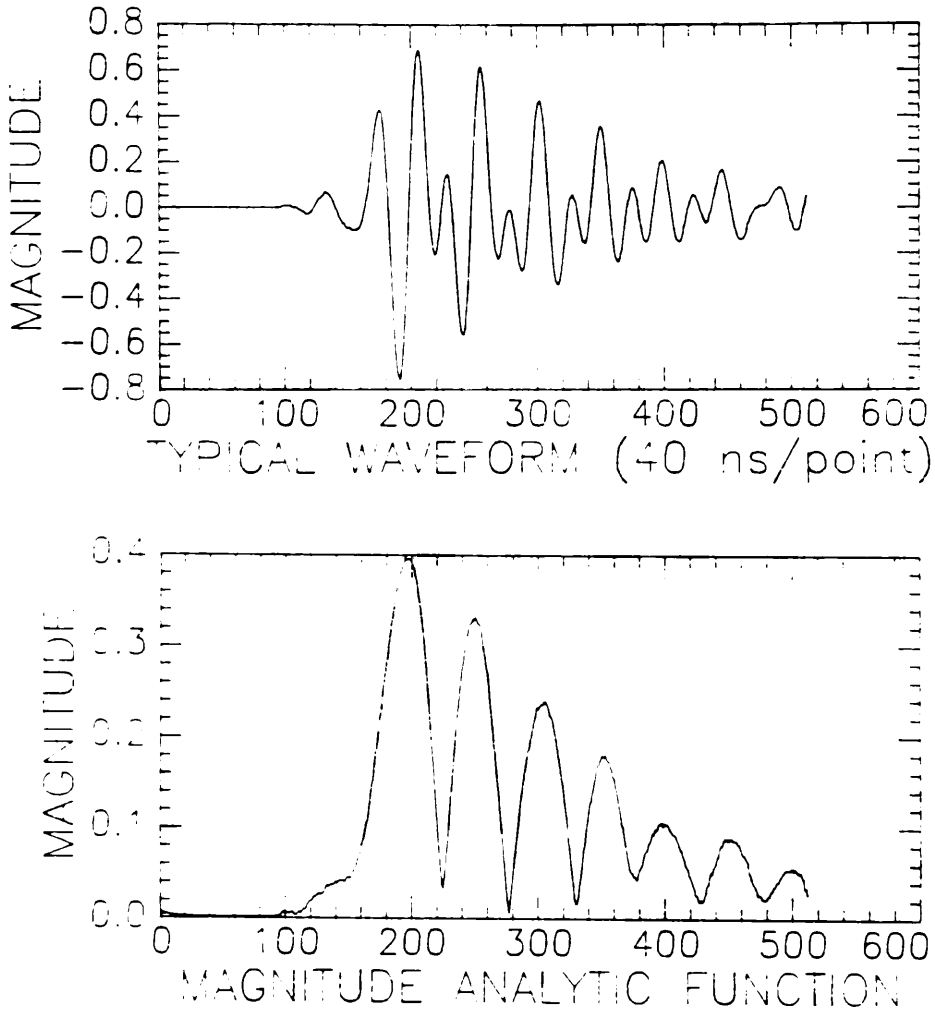


Figure 2.8 Typical Waveform and the Magnitude of the Analytic Function of that Waveform

flight through the material from the front surface to the back surface. This package was used to present the data collected for this chapter.

However, the movie created may not be interpreted as the movie described above. At first, one might think the movie was the result of watching the wavefront pass from one side of the image to the other due to the propagation of the wavefront along the plane of the plate. Actually, it could be viewed (unrealistically but helpfully) as if a wavefront were passing across all scan locations, simultaneously, without affecting any other location, and the surface is watched with a full-field sensor as this "wavefield" passes. In essence, at every point on any one frame of the movie the phase of the wavefront is the same. Therefore, each frame (slice in time) is a simultaneous display of how the surface of the plate at every location is reacting to a "wavefield" with the same properties (phase, amplitude, and particle displacement) at each location as it passes that location.

Interestingly, only the first few cycles of the passing waveform hold any discernable information about the state of the material it passed through. The first few frames at the very beginning of the wavetrain show random noise, possibly due to particle displacement realignment, where all the locations along the wavefront are starting to be, but are not quite, in synchronization with each other. The next several hundred frames, which represent approximately 14 microseconds of the wavetrain, have differences in the surface displacement which correlate well with the location of the damage in the plate. The image then degenerates into what looks like random noise with no discernable features. This correlates well with the conclusion, in section 2.5, that the early part of the wave is "clean", i.e., energy flux and particle

displacement parallel to the plate with very little displacement normal to the plate except for areas which alter the wave propagation, followed by a "wake" of reflecting and refracting quasi-transverse waves or possibly the fully developed zone of a Lamb wave.

2.8 Specific Description of Volume II Movie

The two movies on the videotape are of the data recorded from the scan of the 7 ft-lb impact zone on the 32 ply plate where the transducer fixture was opposite the impact surface. An area 1.5 inches square was scanned at a step size of .76 mm (50 by 50 points). The scanning proceeded from the top left of the screen to the bottom right. The fixture was situated such that the ultrasound traveled in the direction left to the right as viewed. The data was collected at 25 Mhz, hence, the time between frames is 40 nsec. Noting Figure 2.8, the movies represent the frames between 100 and 450, hence, spanning approximately 14 μsec of time. The difference between the two movies is the manner in which the magnitude data was matched to a greyscale. In the first version, the data was scaled and normalized using the maximum and minimum of all the data within the 350 frames. The cycling, between light and dark, was attributed to the reverberations of the receiving pointducer. In the second version, each frame was scaled and normalized using the maximum and minimum of the data in that frame. In both versions, lighter means a larger value of the analytic function magnitude data.

3 Modular System Description

3.1 Introduction

This section will outline the general concepts of a modular testing and evaluation system. The remainder of the chapter will then discuss in greater detail the items selected to fulfill these concepts. The second section details the hardware and the third section details the software.

There are a few basic functions, in a generic sense, of any nondestructive test and evaluation. The control function initiates, maintains and synchronizes the other tasks and evaluates the results. Another important function is probing of a specimen by directly or indirectly generating some kind of excitation in the specimen and detecting how the specimen is affected by or affects the excitation, hopefully gaining some knowledge about the specimen that was not known in the absence of the excitation. Another possible function is movement of the probe or the specimen relative to the other. This might be necessary for physical differences between the probe's "field of vision" and the specimen. A good analogy for this process might be human vision with the brain (controller), the eye (probe), and the hand (motion).

For our purposes, these generic functions are implemented in a modular automated system, which is to be a research testbed. If a system is truly modular, the end result is the flexibility to do many different tests with the

existing system, and also to easily incorporate changes to the system. The emphasis on modularity is in the hardware and also the software implementation.

A functional diagram of our instrumented, automated nondestructive testing system is shown in Fig 3.1. The three generic functions outlined above have been reevaluated, and in the case of the probing function subdivided into the following; control, signal excitation, signal detection, and mechanical motion. The arrows denote communication paths between the functions and the dotted line denotes the demarcation between the hardware and software subfunctions of the controller.

The "action" of the controller function in our system is primarily software execution and hardware interface operation. This action can be broken down into several specific tasks. The controller receives and acts on input from the user and returns results to the user. It controls the signal excitation and the signal detection electronics of the "probing" function and the mechanics responsible for the relative motion between the probe and the specimen. It also analyzes the data received from the test and possibly modifies the testing in view of the results.

The signal excitation function consists of signal creation and possibly amplification to a useful level before sending the signal to the probe sending transducer. How tightly this function is bound to the controller is dependent on the signal source option. In the most rudimentary form there is no direct link between the controller and the signal source. The more sophisticated options have direct software communication links with the controller.

The mechanical function consists of a bridge which moves in two

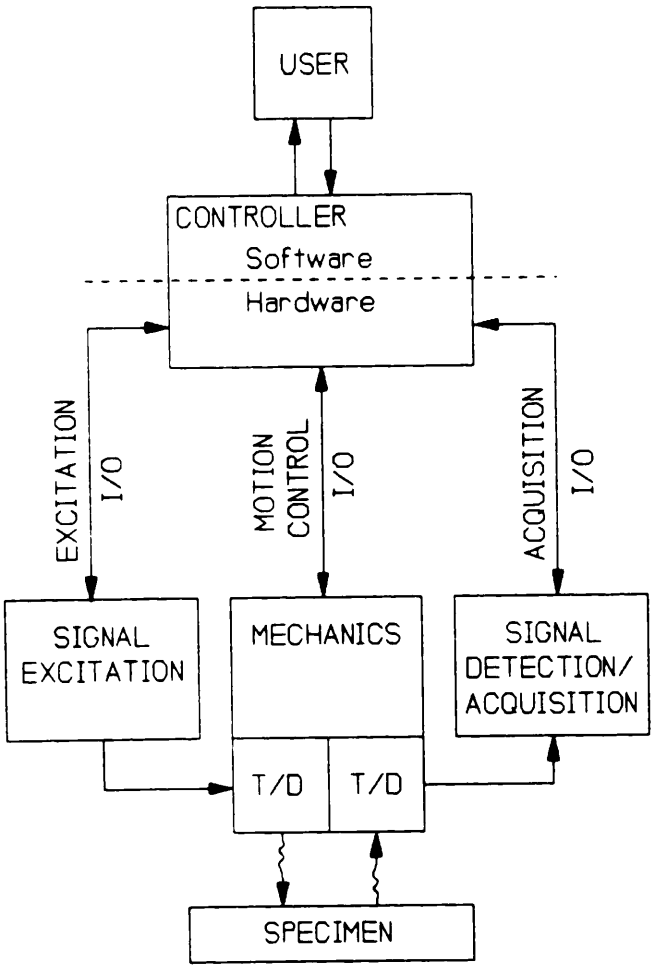


Figure 3.1 Scanning System Function Diagram

directions, the control of the bridge movement, and the probe fixture which is typically attached to the bridge. The probe fixture can be as simple as a single transducer for pulse-echo scanning or as complex as the fixture described in chapter 2.

The signal acquisition function is the heart and possibly the most complex part of the system. It includes preamplification of the received signal, possibly some filtering, and most importantly the digitization of the receiver signal. Without the ability to convert the analog signal into an array of discrete samples that are usable by a computer this system could be custom built out of analog electronics but would lack the flexibility available from digital computational analysis.

3.2 Hardware

3.2.1 Requirements

The schematic in Fig. 3.2 shows the hardware configuration of the system. It should be noted that this schematic is essentially a detailed version, specific to our system, of the schematic in Fig. 3.1 below the dotted line. The discussion of the hardware will be broken into sections based on the system functions and each section will detail the instrumentation required for that function.

To accomplish the long term objectives discussed in chapter 1, it was determined that the hardware should be modular and flexible. The approach considered best uses a controller that communicates with each piece of test equipment and handles all task scheduling, general computation (not the computation internal to some kinds of test equipment), and data handling requirements of the system.

The test equipment should not only be able to listen to the controller, but should also be able to tell what stage the process is in to the controller. For the system to have flexibility the equipment should be interchangeable with comparable devices.

Not too many years ago this approach would have been nearly impossible. Because this equipment would most likely come from many different vendors, the above system developement is practical only if the hardware configurations and communication capabilities are industry standard. The proliferation of digital electronics and the advent of the microcomputer has forced industry to create various mechanical and electronic standards and

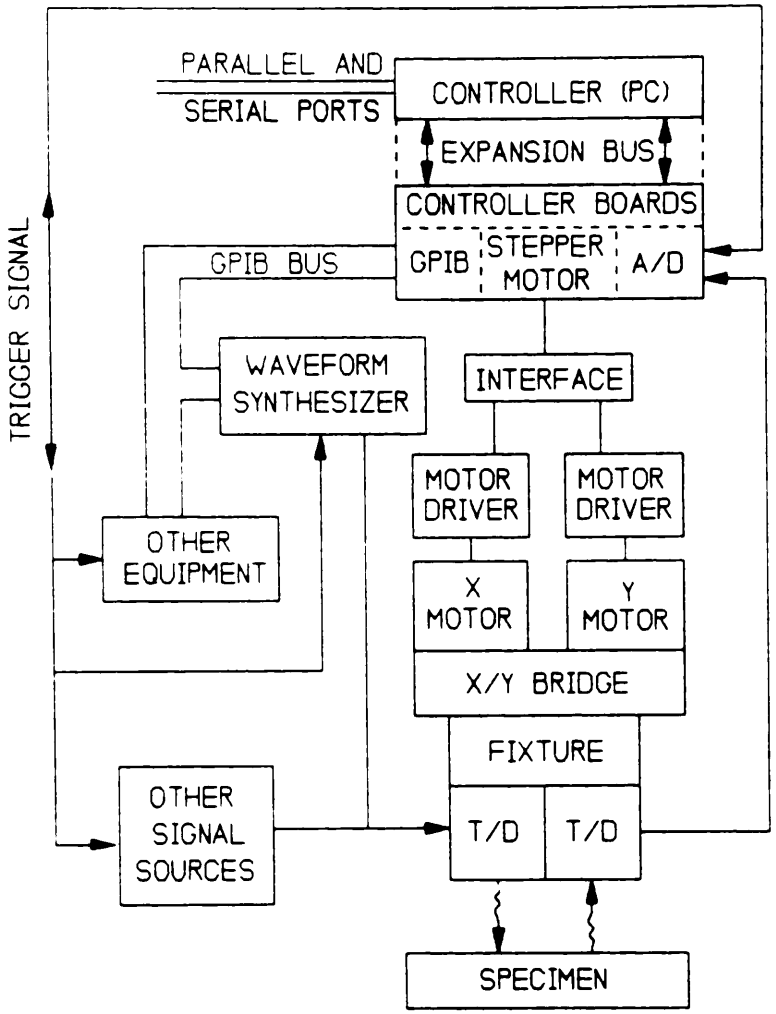


Figure 3.2 Hardware Schematic

has made this approach possible. Digital electronics has generalized and increased the capabilities in areas of instrumentation that were traditionally bastions of analog electronics. The microcomputer has added stand-alone computing and control capabilities to those areas.

An IBM personal computer or compatable with the standard AT bus (not the Microchannel Bus) was used as the foundation for the system. Many manufacturers support this computer architecture in the form of clones of the computer itself, enhancements to the original specifications, and expansion cards for all kinds of research, computing, and interface capabilities.

An effort was made to acquire the fundamental functions of the system on expansion cards rather than stand-alone or rack mounted equipment. Expansion cards have complete access to the resources of the computer without having to go through an external I/O port. Because of this access, expansion cards can communicate faster with the computer than stand-alone equipment and tend to have greater flexibility as regards their control. The particular expansion cards that are presently part of our system have the following capabilities. One controls the stepper motors which move the bridge. One handles the data acquisition. Another card adds an input/output (I/O) port that conforms to IEEE488 standards (originally Hewlett Packard Interface Bus, now General Purpose Interface or GPIB). Another card increases the memory capacity of the computer.

3.2.2 Controller

From the hardware perspective the controller is the personal computer (PC), or to be more exact, the microprocessor chip or central processing unit of the personal computer. The PC processes software, subsequently,

controlling the stepper motor controller, analog to digital data acquisition, and the GPIB interface boards over the computer expansion or input/output (I/O) bus. It can control peripheral equipment and instrumentation via the serial, parallel, and GPIB I/O ports. The PC that is the controller for our system is a clone of the IBM PC AT. It has an Intel 80286 microprocessor and an Intel 80287 math coprocessor to handle the computational chores. It has a fully populated base memory of 640 kilobytes accessable by the operating system (MSDOS), 384 kilobytes extended memory which can be used primarily as a disk cache or virtual disk, and 2 megabytes of extended/expanded memory via expansion card which can be used for temporary waveform storage. For introductory and advanced information about this type of computer the following references are recommended [3.1, 3.2, 3.3].

3.2.3 Controller I/O

There are three basic methods of interaction between the microprocessor and the rest of the circuitry in the computer; memory access, port I/O, and interrupts [3.1]. The working memory is the part of the computer where data and instructions are stored during program execution. It can be considered a map of storage units each with a unique address by which access to that unit is obtained. I/O ports are also accessed through a map of unique addresses which is independent of the memory map, and which does not have storage (in the same sense as memory) capabilities. Interrupts are the means by which the microprocessor knows that external circuitry needs to be serviced. The entire family of IBM personal computers and compatables have a relatively standardized memory, I/O port, and interrupt usage [3.4]. The operating system, normal system resources, and even some types of expansion

cards have specific sections of the memory, I/O ports, and interrupts designated for use by specific functions, leaving the other unused areas open for user expansion.

The remainder of the circuitry includes the expansion bus. The expansion bus is essentially a doorway into the same communications bus and interaction methods that all the system resources use. Because of that access, any card installed physically and "mentally" in the expansion bus becomes part of the computer. Hence, it is conceptually more useful to label as "controller I/O", not the I/O between the computer and the card, but, the I/O between the expansion card and the outside world.

The I/O capabilities of two of the expansion boards in our system are not discussed in the following sections and, therefore, will be touched upon here.

The GPIB interface board, as was mentioned previously, provides a IEEE 488 or GPIB compatible port [3.5]. The GPIB is a parallel bus that allows transfer of data and commands between up to 15 different devices somewhat similar to the computer expansion bus. The most apparent difference is that the physical connection for the GPIB is a cable. This allows stand-alone instrumentation to communicate with each other and the computer if they all have GPIB ports.

Originally, the video adapter and monitor used had to be compatible with the IBM Color Graphics Adapter standard (CGA) due to memory contention between the stepper motor board and the IBM Enhanced Graphics Adapter standard (EGA). This limited the graphics capabilities to two displayable colors at 640 by 200 pixels screen resolution or four displayable

colors for the lower resolution of 320 by 200 pixels. The memory contention problem has recently been solved by the manufacturer of the stepper motor board and the hardware has been upgraded to EGA compatibility. This provides us graphics capabilities of 16 color 640 by 350 pixel resolution maximum.

3.2.4 Signal Excitation

The excitation portion of the system is the portion that produces the signal to be sent to the scanning probe. Its rudimentary configuration consists of hardware with only minimal software to either initialize or control it. There are also several options for the hardware and the subsequent signal that is produced.

- Ultrasonic pulsers
- Function generators
- Waveform synthesizers

The pulser is specifically designed for ultrasonic nondestructive testing. It produces a short rise time, high voltage spike on the order of a few hundred volts amplitude and a few nanoseconds duration. The spike is designed to approximate a delta function (unit-area impulse), which has zero rise time and infinite amplitude and has a frequency spectrum that contains equal amplitudes of all frequencies [2.6]. These pulsers are either separate units that plug into a rack mounted bus or are packaged with receivers and amplifiers in a stand-alone piece of test equipment.

Function generators typically produce low voltage waveforms with a maximum amplitude of tens of volts. There are usually several options for the type of signal produced. These signals can be sine, triangle, or square

waves. They can be single frequency continuous waves or they can be gated waves with a finite number of cycles (tone bursts). They can also have their shape parameters changed. For example, triangle waves can be turned into sawtooth waves. Many models, such as the one available at Virginia Tech, can also do frequency sweeps or "chirps" which are signals that vary in frequency from beginning to end.

While the above two signal sources are generally analog, waveform synthesizers are digital. In other words, even though the output signal is analog as are the other sources, it is the result of the processing of digital data. Digital waveforms are created internally by several different calculation methods. The flexibility of the calculations are the same as the flexibility of calculation on a personal computer. These waveforms are then sent to a digital to analog converter circuit (D/A) which creates analog output from digital input. Hence, there are a multitude of the types of signals that can be produced. The particular synthesizer used here has three modes of operation. One mode is very similar to a function generator. A few "standard" types of waveforms are available and the shape parameters have the same variability as those of a function generator. Another mode is programmable: it has an equation language from which polynomial, transcendental and discontinuous functions can be calculated and output. The last mode is completely arbitrary: it uses digital arrays of data that are created by an external computer or similar means and downloaded to the synthesizer through input/output ports to create an output waveform equivalent to the input array. The last two modes not only increase the numbers of waves that can be produced, but also provide a mechanism to do adaptive signal generation

where the output signal can be changed due to feedback from external events.

It is desirable to have a high amplitude ultrasonic signal to excite the material being tested. Unlike pulsers, which are designed for ultrasonic work, function generators and waveform synthesizers do not have the voltage output required to produce a substantial excitation in the specimen. A nice solution is to use a power amplifier, with a broad frequency response that will not distort the signal, to boost the amplitude of the signal being sent to the probe. The use of the power amplifier therefore allows the use of many different types of signal sources that are too weak for ultrasonic interrogation of lossy materials.

3.2.5 Motion Control

The mechanical heart of the system is the device which automates the relative motion between the specimen and the probe, allowing rapid, systematic, and consistent interrogation of the specimen. At present, we have a two axis bridge that can move to virtually any point in a plane that is parallel to the surface of the water in the tank that the bridge spans. Two stepper motors control the motion, one for each axis. The two high speed drivers that drive each stepper motor are controlled by a stepper motor controller card installed in the personal computer dedicated to this system. This controller card has the capacity to control more axes with the addition of daughter-boards. The card also contains registers that keep track of the run status of the motors. The resolution of the incremental motion is nominally one thousandth of an inch, but, can be changed to one half thousandth. For this discussion a "step" is defined as one increment and a "jump" is defined as many steps.

3.2.6 Signal Detection and Conditioning

Before it can be analyzed, the ultrasonic signal that passed through the material must impinge on the receiving transducer in the probe, where it is converted into an electrical signal. Most likely, the signal will need to be conditioned before going to the data acquisition board. This section will describe that process and the associated equipment.

The received signal usually needs to be amplified before it can be used by the rest of the system. Typically, received signals have microvolt amplitudes. The data acquisition board currently used has a conversion resolution of approximately 4 millivolts, hence, amplification is needed to fully utilize the dynamic range of the board.

Different types of preamplifiers are available to provide the amplification. They have different power sources, frequency ranges, and gains. The DC preamplifiers are battery powered and therefore do not introduce noise from the power source into the signal as can the AC powered preamplifiers. Even though the preamplifiers do have a broadband frequency response they still have their limitations. The gain is constant throughout the reported frequency band but drops off outside the range effectively filtering out those frequencies. Therefore, one should make sure that the signal and the preamplifiers are matched so as not to lose any information.

If the waveform synthesizer or the function generator is used as a signal source the amplified signal from the preamplifier usually goes directly to the data acquisition board. However, if the ultrasonic pulser is being used the preamplifier feeds directly into a receiver section of the pulser. Receiver sections usually contain their own gain and attenuation controls for further

tailoring of the signal amplitude. Ultrasonic pulser/receiver units are designed to work in either a pulse-echo mode where one transducer is used as transmitter and receiver, or in a pitch-catch mode that uses two transducers. Because of this, the receiver circuitry is designed to handle the voltage of the initial pulse without being damaged.

There is also other signal conditioning hardware, such as gates and peak detectors, that can stand alone or is integral to many ultrasonic pulser/receivers. Gating circuitry can be used to blank out unwanted portions of the waveform. Inside the gate, which can be moved in time and changed in size, the signal is unchanged, while outside the gate the signal is reduced to zero. Gating can also be combined with peak detection circuitry which will output a voltage that is proportional to the peak amplitude detected in the gate.

3.2.7 Data Acquisition

The main purpose of the data acquisition hardware is to convert an analog input signal into a representation that is usable by digital electronics. This is called the analog to digital (A/D) conversion or the digitization of the signal. This conversion is accomplished by sampling the analog signal at discrete time intervals and outputting a binary integer representation of the amplitude of each sample. Hence, the analog input signal becomes an output sequential array of numerical values.

The initiation of the A/D conversion is dependent upon a trigger signal which is a square wave signal. Every time the trigger changes in one direction it starts a sampling sweep where the A/D converter takes multiple samples. The number of samples taken depends upon the size of the buffer

holding the samples. After each sweep, which fills the buffer, the buffer is either cleared or averaged with the next sweep.

The A/D conversion process has some details, if not properly addressed, that can adversely affect the accuracy of the representation of the analog input signal by the digital output signal. First, the binary representation of the amplitude of the sample is integer valued. The A/D converter has a finite range for input voltage and it represents this range by a finite number of equal steps. Any variation in the input signal smaller than the difference between steps is output as the integer value which represents the closest step. Hence, very small signals are not represented accurately. Another problem area is the discrete sampling in time. If the sampling frequency is not at least twice the frequency of the highest frequency component of the analog signal there is misrepresentation of the frequencies above half the sampling rate [3.6].

For example, the current A/D board has an input range of $\pm$ .5 volts which is represented by 256 steps, hence the resolution of the representation of the amplitude is 1/256 or approximately 4 millivolts [3.7]. It can sample transient waveforms at a rate of 25 MHz and equivalent time sample repetitive waveforms at rates up to 200 MHz. The equivalent time sampling is where, even though the A/D converter can sample at a particular rate, a time delay of increasing duration per trigger cycle, (that is smaller than the sampling cycle time) is inserted after the trigger for each sampling sweep before the signal is resampled. Hence, if a series of sampling sweeps are overlaid, the signal has essentially been sampled many times between each sample of a standard sampling sweep, thereby increasing the effective sampling rate. The

board has two channels of input and an onboard memory buffer for temporary storage of waveforms between each sampling sweep. The board can average several waveforms together before outputting the resultant waveform and it has the ability to do the triggering or be triggered externally.

3.3 Software Modules

3.3.1 Requirements

A generalized flowchart of the software is shown in Fig. 3.3. The solid line boxes represent either a subroutine or a library that controls either a system function or a specific piece of hardware. The dotted lines represent actual software modules. As one moves from the top of the diagram to the bottom, the routines relate less to the user and more to the hardware. The bottom row primarily represents the hardware driver subroutine library software provided by the manufacturers. The details of these modules will be discussed in more detail in the following sections. Unless otherwise indicated, in this chapter the word "module" has a physical meaning: a section of code that has been separately compiled and must be linked to the other modules to form the program. The code was written so that the word "module" also has a corresponding conceptual meaning: a section of code that contains logically related functions.

Ideally, the software drivers, that do the low level interfacing to the hardware, should be provided as separate libraries of subroutines that can be linked into user created code. It would also be useful if the manufacturer provided diagnostic software or demonstration source code that can be used to exercise the hardware. These packages are very useful for testing and debugging each piece of hardware separately from the rest of the system.

The programming should be modular or structured in appearance, and also in practical terms, so as to allow changing modules with a minimum of effort. For example, it may be desirable to change the scanning pattern to a pattern that is not a traditional rectangular grid. A new module should be

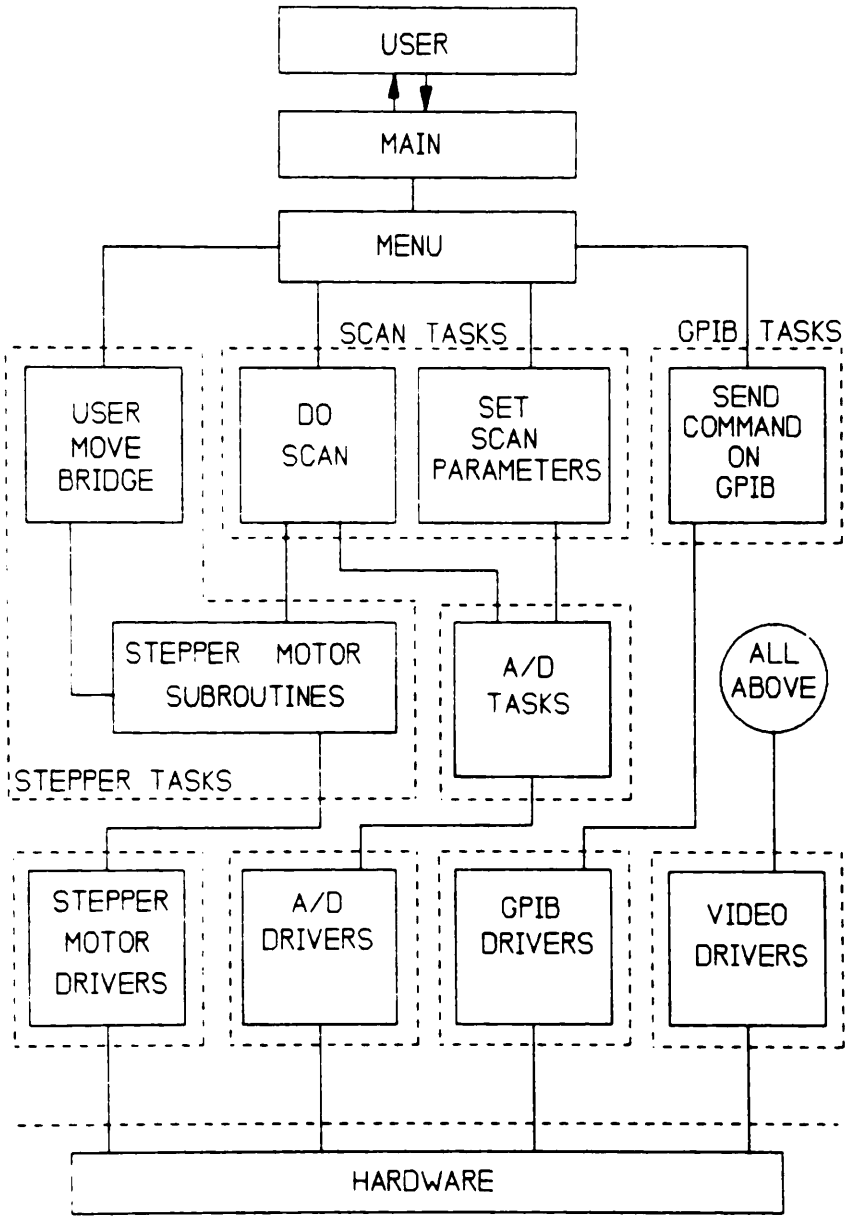


Figure 3.3 Software Hierarchy Diagram

able to be easily programmed from a library of primitives (simple subroutines that do one basic task each) that isolates the programmer from the hardware interface details of the stepping motor controller. It should be implemented with minimal impact on the other modules and easily switched with the existing module or added to a menu of available modules.

"C" a widely supported and popular language, was chosen for the programming [3.8, 3.9]. "C" allows the above mentioned programming capabilities because it is small, compiled, structured, low level to interface with the hardware, high level to do advanced mathematical analysis, and extendable by libraries of functions. Smallness implies that a language has a sufficient but minimum number of logic control structures required for programming without a lot of extraneous structures with overlapping abilities. A language, when compiled, typically executes faster than when interpreted, because there is no interpretation overhead. Structure in a language can enhance modular programming, but too much rigidity can increase bookkeeping. A high level capability can be programming mathematical manipulations in the manner that we write equations, algebraically. A low level capability is explicit programming of the computational hardware. If one needs even more functions, the language should be extensible in a modular fashion that allows one to incorporate needed functions without carrying along the overhead of unneeded ones.

It should be noted that the modules of Fig. 3.3 exist entirely above the dotted line in Fig. 3.1. They are the interface between the user and the system hardware. Since the system functions can be implemented in many different ways by the software, a short discussion of the software

requirements for each system function follows, after which are presented the detailed discussions of each of the software modules beginning with the drivers and progressing up through the modules.

In the signal excitation function each of the options for signal sources has different requirements for control by software. For the pulser, the only control required is a trigger signal to synchronize the output with the operation of the rest of the system. If that trigger signal comes from hardware in the system that is under software control no special code needs to be written for the pulser. However, the function generator has a GPIB I/O port that allows an external computer to initialize and set the parameters that one would normally do from the front panel. There are no data coming back from the function generator, therefore software is only needed that will initialize parameters before the test and possibly change parameters during the test. The waveform synthesizer also has a GPIB I/O port, but has many more capabilities then the function generator and hence a greater capacity for control. The adaptive scanning module mentioned above does not yet exist, hence it will be discussed in the chapter on future work. The power amplifier, required by the function generator and the waveform synthesizer, is a nonadjustable piece of equipment and has no options for control except the on/off switch.

The next level of complexity exists in the motion control function software. One can send commands to the controller, and through the use of status registers on the board the software can tell when the stepper motors are moving or stopped.

The most complex function is the signal detection and acquisition

function. The preamplifiers and the ultrasonic receivers have no communication capabilities. They do not require any software to control them, however, the operation of the data acquisition board is entirely software controlled. A portion of the software is interactive and emulates a digital oscilloscope with a real time image of the signal and a menu for changing the parameters. Once everything is set and the scanning is started, the software automatically handles the board.

3.3.2 Stepper Motor Drivers

The motion is controlled by two subroutine libraries of software drivers, provided by the vendor [3.10], that cause the board to set a step speed, step at that speed in the positive and negative directions for each axis for a specific number of steps, or accelerate to a set speed based on a ramp function. One library contains the actual drivers for the hardware functions of the board and is never directly called by any of the system programming. It was delivered as object code (already compiled) and therefore can not and should not be modified. Any call to this library is made by the other library which contains the code for interfacing to specific languages. It was delivered as assembly source code and required modification to be called by large memory model Microsoft C programs.

3.3.3 Data Acquisition Drivers

The data acquisition board is controlled by an extensive library of 39 software drivers. These drivers are supplied, by the vendor [3.7], as assembly language source code which after compilation can be called directly by C programs. They were written for the large memory model of Microsoft C and did not require any modifications.

The drivers communicate almost entirely with the board through eight I/O ports and can be divided into groups by their function. Because there can be multiple A/D boards in the system, one group relates to the board environment. These drivers handle such functions as setting the number of boards in the system, setting which board is active, setting the port addresses, and setting where and how the memory is accessed. The largest group is used to set and change all the data acquisition parameters. This group handles control of the trigger, pretrigger, active channel, threshold and other related parameters. Two routines are for general port I/O. Other functions such as arming and firing the board are used during data collection and the final group manipulates the data after it is collected.

3.3.4 GPIB Device Driver

Since the GPIB protocol is an IEEE standardized method of controlling a myriad of functionally different instruments, modification of this device driver would be disastrous, hence this device driver is delivered by the vendor [3.5] as an executable file only. Included with the driver are utilities for interactive control of the GPIB and monitoring of application programs. These are very useful for developing and debugging application programs.

The GPIB, as was mentioned previously, is a communications bus that allows the transfer of data and device specific commands between devices and controllers. Hence, the driver functions control the bus "handshaking", data transfer, and other bus management functions.

There are two different means of classifying the driver functions, the simplest being a differentiation into high level and low level functions. The low level or board functions perform a basic single GPIB operation and require

some knowledge of GPIB protocol, to allow solution of specific bus management problems. The high level or device functions execute command sequences and automatically direct bus management, hence freeing the programmer from GPIB protocol, but are limited in scope and flexibility.

The other means of classification is differentiation into six groups based on the type of application to which they are useful. The first three groups consist of primarily high level functions while the other three are primarily low level. Many instrument control applications can be written out of the first group which are for communication between the controller and one simple device at a time. They open, read from, and write to devices that do not require special services such as polling or triggering. The second group is for polling, triggering, clearing, and placing into local mode the devices on the GPIB. The third group is for asynchronous operation, I/O to a file, controller authority and timeout value changes, and reinitialization of the GPIB device driver itself. Group four is for broadcasting messages, interdevice communication, parallel polling, and control of specific handshake and management GPIB lines. Group five is for the situation where the GPIB interface board is not the controller in charge (CIC) of the bus and group six is for changing configuration parameters such as GPIB addresses and transmission terminators.

3.3.5 Video Drivers

The basic functioning of a video display is included in the base computer as a standardized interaction between the operating system, the BIOS (Basic Input/Output System, the kernel that drives all the system hardware) and the hardware itself [3.2]. The display is memory mapped and the

appropriate section of memory is automatically accessed and displayed without any intervention from the user.

It should be noted that there is a difference between text video modes and graphics video modes. One frame (a complete scan from the top left to bottom right corner) is displayed in one mode because the video memory map is interpreted differently for each display mode. This makes displaying "text" in a graphics mode or displaying "graphics" in a text mode either not easy or not very comprehensive.

The vendor of the data acquisition board has created a library of graphics routines appropriate for the operation of the board. It functions in a graphics mode and is able to display an input signal in real time. Included are routines which create grids and display alphanumeric characters.

Also being used are other video libraries, as will be discussed in section 3.3.10 and Chapter 4, for thier specialization in certain areas.

3.3.6 Data Acquisition Tasks

This module is a simple set of functions for the basic operation of the data acquisition board. These functions are slightly modified versions of some demonstration code provided by the vendor.

One function [stinit] reads all the setup parameters from a file of values that correspond to one option for each parameter out of a set of options. It will then initialize all the I/O ports, the memory map, and the board parameter settings. Another function [setall] can be used for resetting all the parameters to the default values after they have been changed. The last function [strbuff] arms and fires one or both channels and moves the data

from the high speed buffer on the board to an array in conventional memory where the rest of the software has access to it.

3.3.7 Stepper Motor Control Tasks

In general, there are two ways to move the bridge a certain distance. One way involves putting a call to move a single step inside a software loop that executes a set number of times, while the other involves a single call to do a jump. The first method forces the stepper motors to start and stop for each step, and the second method does a "smooth scroll" through the steps. The first method can produce very rough motion if the time between steps is very small. It should also be noted that the board is essentially self contained. Once it receives a command it can carry out the execution of that command regardless of what the computer is doing.

A library of subroutines has been written that combines calls to the driver subroutines to simplify and enhance the use of the board. There are four subroutines, one to initialize, one for interactive motion of the bridge, and two stepping subroutines for different types of scanning. The initialization subroutine [startpcm] contains the code that will initialize the board, sets the step speed that was determined to be optimum for our bridge, and clears up unwanted motion that sometimes occurs when the board is initialized. The interactive bridge motion subroutine [movbridge] allows the user to position the bridge via the keyboard in a manner similar to joystick control. The subroutine is written such that the user can change the length of a "jump" from one step to any number desired and the execution of the jump is controlled by the keyboard arrow keys which correspond to the direction of the jump. Both stepping subroutines combine subroutine calls to

allow the specification of the number of steps and the direction of motion in one call statement. The difference is that one subroutine [step] halts program execution until the stepping is finished while the other [fly] allows program execution to continue and returns a status code indicating whether the stepping has finished or not.

3.3.8 Scanning Tasks

This is the largest module in the entire program. The section which displays a waveform and parameter menu was inspired by some software originally written by the vendor, but was highly modified to streamline it for this purpose. The remainder of the module is the scanning routine itself which does the automated interrogation of the specimen.

The function that does all the data acquisition parameter changes [setparm] was written to simulate a digital oscilloscope. The input signal is displayed real time with a menu of parameters and current settings. The interface presented to the user looks like the image in Fig. 3.4. The functions for changing the sampling rate, the scale of the display, the size and location of the gate, and the threshold and trigger control have been assigned to function keys. For the functions which have many options, repeated typing of the function key will cycle through the options. The arrow keys are used to scroll the waveform on the display. A few alphanumeric keys are used for one-shot functions which clear the screen, home to the beginning of the signal, rewrite the signal without erasure, rectify the signal, and quit.

The scanning function [scan] integrates the operation of the data acquisition board and the stepper motor controller. Once all the parameters have been set with the previously described function this function is called to

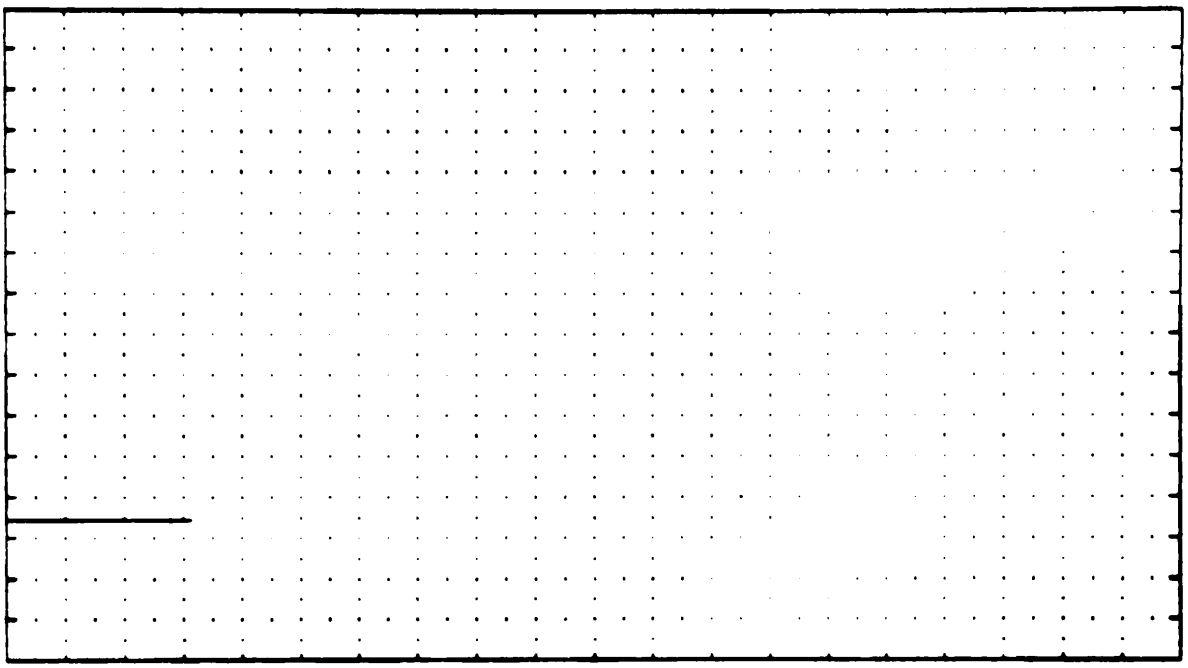


Figure 3.4 Setup Screen for Data Acquisition Parameters

do the test itself. This function does a raster scan test where the probe (attached to the bridge) does an on-the-fly sweep in the y direction, while the data acquisition board is simultaneously taking data. When it reaches the end of the sweep, data acquisition stops and the bridge steps in the x direction. The probe is then swept in the negative y direction as data is being taken until it reaches the end of the sweep. When the original y coordinate is reached, everything stops, then a step is made in the x direction and the process starts all over. The overall size of the scan and the step size between sweep lines is controlled by the values interactively input by the user when the function is called.

3.3.9 GPIB Tasks

This module is a simple set of functions to only send commands to one device on the GPIB. The primary function opens and reads a user created file of device specific commands. While it sends these commands to the device, the error flags are monitored. If an error occurs from opening the device, from communicating with the device, or from the device itself during execution of the commands, the appropriate error handling function is called. Currently, the error handlers only report the error and the command file is written for the waveform synthesizer.

3.3.10 Menu

The user interface is the "how" and the "what" of the user communication with the system. The most important functions of an interface usually occur at the beginning and the end of the test sequence. At the beginning the user needs to set all the correct options and parameters required for the test. At the end the program presents the results to the user in a

manner hopefully useful to the user. During the test procedure intervention by the user should only be to change or stop the test. The user should not have to intervene to keep the test going. Also the program should let the user know where in the process the test is and what it is doing. The effectiveness and efficiency of the system depends on whether all the functions of the user interface are obscure or intuitive.

The primary method of control over this program is through the menu interface. The menu currently has as separate functions the following:

- Move the bridge to any desired location before scanning,
- Set the data acquisition parameters,
- Initialize any instruments on the IEEE 488 bus,
- Set the step size and grid size, and start the scan,
- Exit the program.

The part of the program that controls the execution of the above modules on the menu is realized in a library of menu and video subroutines [3.11]. The menu library handles displaying the menu and controlling the user selection of menu items. Displaying the menu requires functions which save and restore the portion of the video screen where the menu is to be located. User selection requires functions that redisplay the menu with the highlighted selection and functions that "execute" the chosen selection. Scrolling through the menu with the up and down arrow keys highlights the current selection. Selections are executed by hitting the enter key which chooses the highlighted selection or by hitting the letter key which corresponds to the indicated letter of a menu item, regardless of whether the item is highlighted. However, the menu library does not actually call the module corresponding to

the selected menu item. It merely indicates to the main program which module was selected. For the sake of modularity and also because the different modules do not have many overlapping needs for video output, each module takes over control of the video input/output when that module is being executed.

3.3.11 Main

The main program is very simple, consisting primarily of calls to initialize the hardware, to initialize the display mode, and to execute the various modules as they are needed. The most important call is to the menu module which creates and runs the menu interface, hence controlling the calls to the other modules. If the menu has to be changed, all the necessary changes to the menu itself are done in this module which are then passed to the menu module during execution. There is no need to change the menu module.

3.4 External Software for Analysis and Presentation

Analysis of the data into meaningful results is predominantly a software function. Also, wave propagation and its relation to the state of the material is a very complex subject. Hence, there are many options available for analysis. Also, because the area of research is so complex, presentation of the data and results is intimately tied to the type of data to be presented.

Currently, a few options exist for dealing with the data. One is to send the raw data directly to a file. Hence, the data can be processed and displayed in any manner an end user wishes without changing the existing program. One such program written by Bartlett [3.12] can be used with a little modification, to calculate the narrow band moments of the frequency spectrum of the data. Presently, the only internal analysis the program does is a peak detect within a user definable gate. It stores those results to a file which can be displayed in a traditional C-scan format by another program, written by the author, which is discussed in Chapter 4. However, once the analysis and display concepts needed for the next phase of research are solidified, it will be very easy to add capabilities, such as the magnitude of the analytic signal as discussed in Chapter 2.

4 Illustrative Example

4.1 Introduction

This chapter will present an example which will illustrate the capabilities of this system, and relate these capabilities to an analog C-scan and the scans discussed in Chapter 2.

4.2 Equipment Configuration

The system was configured to do a standard C-scan, using a Panametrics pulser/receiver (broadband spike excitation) and a 15 MHz, .25 in. dia., 1.5 in. focal length, focused underwater piezoelectric transducer in a pulse-echo mode. The specimen scanned was the surface of the ubiquitous quarter. The software was configured (at each scan grid point) to grab the waveform inside the user adjustable gate, scan that data for the peak amplitude, and write that value to a file. The gate was approximately 3.5 μsec in length and was adjusted around the first returned pulse so that when the focal point was not on the quarter (hence shifted in time) the signal in the gate was just the baseline noise. The size of the gate was such that, when the focal point was on the quarter, the entire pulse (no discrimination between back, front and intermediate surface reflections) was in the gate.

4.3 Results

The peak values written to the file were plotted to the screen by a

separate program [scanplot]. The program scales input data into a monochromatic gray-scale. A graphics library [4.1] was incorporated which had routines that could change the colors of the standard EGA palette. Hence, the scale has been expanded from four grays (in the standard palette) to seven true (not dithered) shades (toward black) and tints (toward white) of blue: from black, which represents the lowest amplitude, to white, which represents the highest amplitude. If more differentiation is required, which means more bins to fit the amplitude values into, the program can be easily adjusted to produce a cacophony of 16 colors. The program allows the user to scale small images to fill the display, and will clip any part of the image that goes beyond the screen limits. It also allows the user to adjust the thresholds for the separate bins which correspond to each color displayed. Thus, the number of pixels plotted for each color, and hence, the quality of the image is controllable.

Figures 4.1 and 4.2 are color plots of the results of two scans done at different step sizes. The first one is a 200 x 200 point, .005 in. step size scan and the second is a 500 x 500 point, .002 in. stepsize scan. The stand off distance of the transducer (approximately 1.5 in.) was adjusted between the first and the second scan, by maximizing the amplitude of the reflected pulse.

4.4 Comparisons

Comparing this scan with those discussed in Chapter 2 and with standard C-scans would seem like comparing apples and oranges, due to the different probe configurations. However, one can compare some of the limitations of each system.

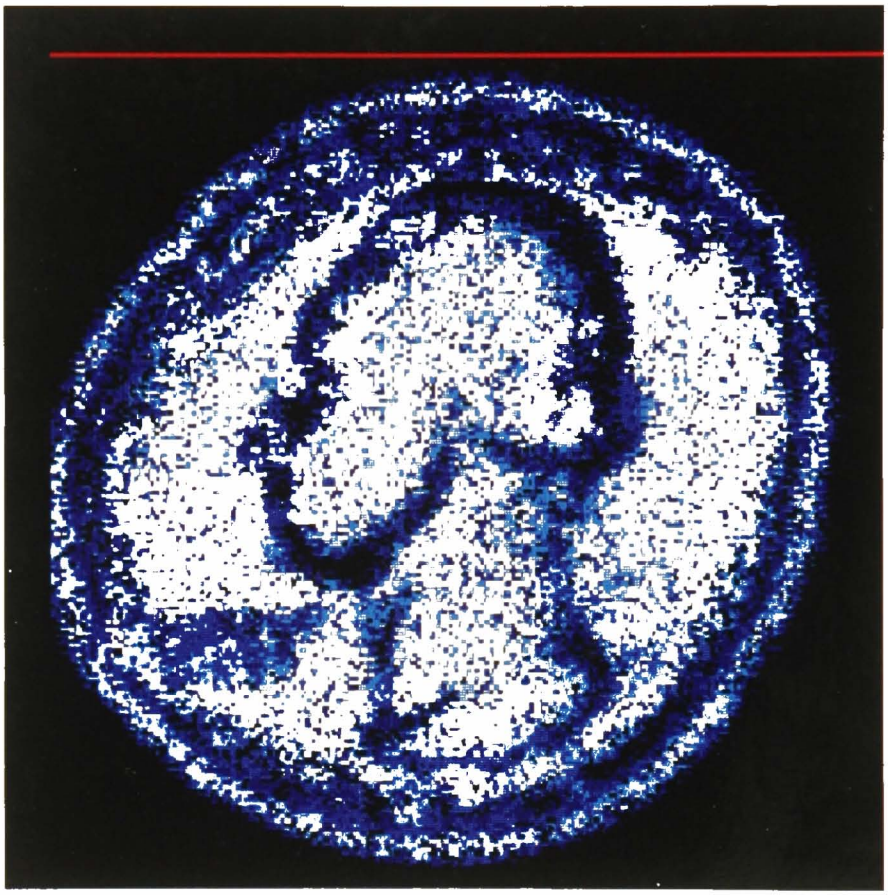


Figure 4.1 Scan of a Quarter: 200 pnt. x 200 pnt. x .005 in./pnt.

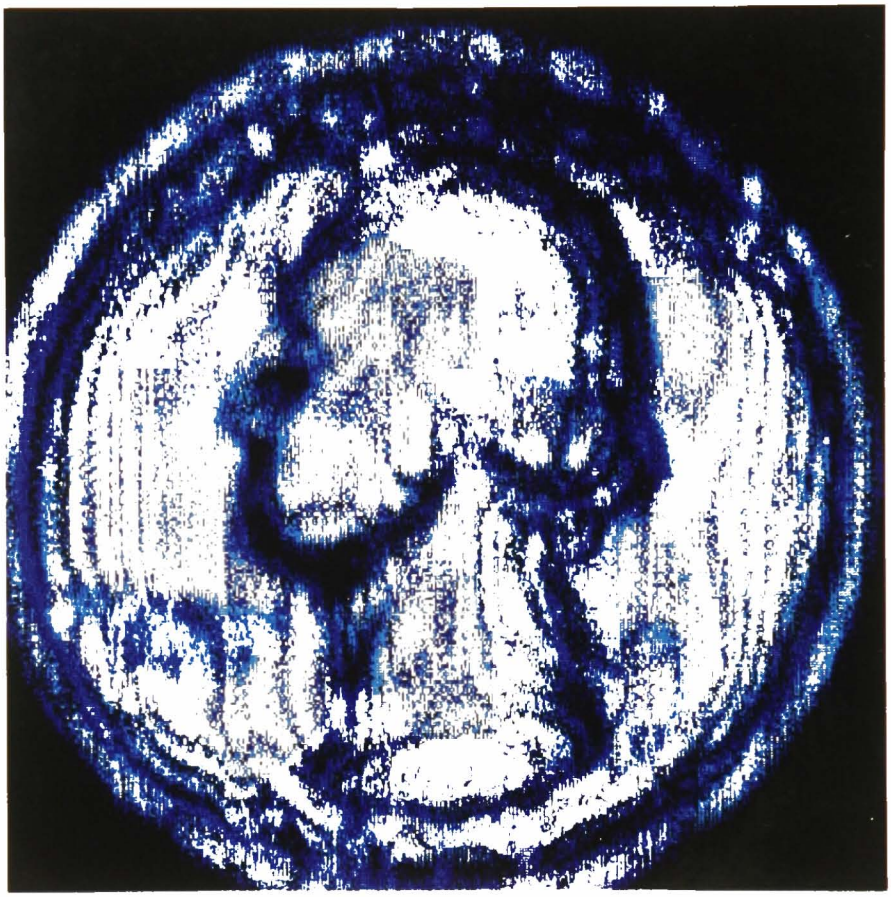


Figure 4.2 Scan of a Quarter: 500 pnt. x 500 pnt. x .002 in./pnt.

The times required for the scans of a quarter were approximately 45 minutes for the first one and 5 hours for the second one. The large difference is due to a larger number of data points and also wave averaging for 16 cycles to reduce some of the noise. Obviously, if all that is needed is a traditional C-scan (the above size and resolution would have required less than 30 minutes), then the traditional C-scan tank should be used. Comparison to the scans in Chapter 2 (50 x 50 scan took approximately an hour) is also legitimate, because both scans grabbed an entire section of a wavetrain. One scan (NASA) stored the data while the other (quarter) filtered the data for the peak. Hence, if all scanning parameters are equal this system shows an improvement in speed over the NASA scans. Also, note that for this system the amount of wave averaging and the pause time between steps can be easily changed.

At the time, the scans at NASA were limited to black-to-white gray scale output. Traditional C-scans are also only gray scale. The human eye can only resolve 30 variations in a gray scale but can distinguish many more variations in color. Thus, color can convey more information if it is needed. At present, our system is limited to 16 colors, but an easy upgrade to the VGA standard would increase, to 256, the number of colors available for display on the screen at one time.

5 Accomplishments

5.1 Feasibility Study

Several questions where addressed by this study. Is it possible to excite and maximize stress wave propagation along a primary structural loading direction from a different direction? With that excitation, is it possible to detect energy scattered from areas of low energy transmissibility? If that detection is possible, can the scattered energy be related to the ability of that area to carry load. This phase of the research has shown that:

- excitation of particle displacement primarily in the plane of the plate, from the surface of the plate, is feasible for some materials, if certain features of the representative slowness surfaces are exploited,
- changes in material quality affect the surface displacement of the plate, when the above mentioned excitation passes through that area, significantly enough to be used as an imaging technique,
- and continuation of this research would be enhanced by a flexible computer controlled scanning testbed.

The last question concerning load carrying ability of these detected low transmissibility areas was only partially answered. If one can assume that the impact damage in the specimens significantly affects that area of the plate's

ability to carry load, then this technique has detected that ability. However, at this time no mechanical testing has been done to those specimens.

5.2 Testbed Development

Separate pieces of off-the-shelf controlling, computing and data acquisition hardware have been physically combined into a scanning controller system. Software has been written that combines the separate drivers for each of the hardware functions into a single controlling program. The result is an integrated and working testbed that has greater potential for significant research than currently existing scanning systems (which typically do not store entire waveforms). The system is continually being improved, however, it has reached the point where it can be used as a research tool for new techniques.

6 Future Work

6.1 Hardware

A user's and programmer's manual is next on the agenda. It will document the useful details related to our implementation of the hardware (which are missing or incomprehensible in the existing documentation), the procedures for customizing the software, and other tips and hints regarding operation of the system.

Future plans also include integrating into the system a robot arm, which has been purchased for complex testing procedures and/or complex specimen geometry.

Since the EGA standard and the IBM Video Graphics Array (VGA) graphics standard are memory compatible (hence, one does not have to worry about memory contention with other accessory boards when upgrading) there is an easy upgrade path to the VGA's 16 color 640 by 480 or 256 color 320 by 200 graphics modes. Because the VGA modes are very close to the visual appearance of broadcast television there are VGA adapters on the market that have standard television NTSC video I/O. Hence, production of a movie such as described in the Chapter 2, will be available to the present system.

An interesting point to make is that the system is not limited to ultrasonic scanning with piezoelectric transducers. Other excitation sources can be used such as laser generation of ultrasound. One could even abandon

ultrasound altogether and, for example, do scanning with eddy current probing equipment. The GPIB interface, which is an industry standard for connecting digital instumentation, has opened up the system up to a wide range of instrumentation.

6.2 Analysis

Because one can digitize and store any portion of a waveform, the data are available for many different kinds of analysis and there is a myriad of possibilities for future software development for this system.

The first objective of the author's doctoral research is implementation of properties of the analytic function of a waveform, which, as mentioned in section 2.3, is related to the energy density of the wave. A detailed explanation of this relation has been given by Heyser [6.1, 6.2, 6.3]. In short, the magnitude of the analytic function is proportional to the energy rate-of-arrival, and therefore, is an indication of the total energy density. The phase of the analytic function is an indication of the instantaneous partition of the total energy into potential energy (related to particle displacement) and kinetic energy (related to particle velocity). Furthermore, the potential energy density and the kinetic energy density can be derived from the signal by convolution with the impulse response and doublet response (Hilbert transform of the impulse response), respectively. It would be very interesting to make images of the specimen based on such properties of the signal. Granted, such images would not be constant in time, but if viewed in a movie format, these images might prove valuable for examining energy transmissibility.

Another exciting idea is adaptive signal analysis and generation. If a

waveform synthesizer is used as the signal source, a feedback loop can be created using the arbitrary wave and download capabilities of the synthesizer. The received signal can be used as input to some "judgement" or "analysis" as to what the excitation signal should look like. This prototype excitation signal can then be downloaded to the synthesizer and output using the arbitrary wave mode. Depending on the extent of the analysis, such a concept could be performed in conjunction with a scanning regime. It is plausible that this procedure will be needed for complex geometry scanning as implemented by the robot arm mentioned above.

6.3 Applications

The application of composites in a practical design is typified by complex geometry, where an effort is made to increase the percentage of fibers in critical directions. It seems plausible that the local slowness surfaces for these critical areas would be similar to the ones discussed in chapter 2. Hence, if one were to exploit the discussed scanning technique (which would require some fixture modifications for complex geometries) and the waveguide effect, it is possible that an image can be made that isolates these critical areas. If a correlation can be made between the image and the load carrying ability of the critical area, a valuable technique has been advanced. Teaming this concept with the ideas of analysis presented above, it might be possible to examine only the major load carrying component of composite structures: the fibers. Hence, further research in this area would seem prudent.

References

1.1 *Nondestructive Inspection and Quality Control, Metals Handbook,* 8th ed., Vol. 11, Am. Soc. Metals, Metals Park, Ohio (1976).

1.2 H. Kolsky, *Stress Waves in Solids*, Dover Publications, Inc., New York NY (1963).

1.3 A. Wieczorek and G. Schmitz, "Development of Nondestructive Testing Methods for the Evaluation of Thin and Ultrathin Sheet Materials," *AFML-TR-65-320*, (December 1965).

1.4 *Prevention of the Failure of Metals Under Repeated Stress*, John Wiley and Sons, Inc., New York NY (1941).

1.5 S. Timoshenko, *History of the Strength of Materials,* McGraw-Hill Book Co., New York NY (1953).

1.6 S. Timoshenko, *Strength of Materials,* Part II, 3rd ed., D. Van Nostrand Co., Princeton NJ (1956).

1.7 H. Liebowitz, Ed., *Fracture*, Vols. 1-7, Academic Press, New York NY (1968-1972).

1.8 J. Branger and F. Berger, Eds., *Problems with Fatigue in Aircraft*, Proceedings of Eighth ICAF Symposium, ICAF Doc. NO. 801, Lausanne, Switzerland (1975).

1.9 H. Liebowitz, Ed., *Fracture Mechanics of Aircraft Structures*, AGARD-AG-176, National Technical Information Service, Springfield VA (1974).

1.10 E. T. Wessel, W. G. Clark, Jr., and W. H. Pryle, *Fracture Mechancs Technology Applied to Heavy Section Steel Structures*, Paper 72, Fracture 1969, P. L. Pratt, Ed., Chapman and Hall Ltd., London (1969).

1.11 M. M. Schwartz, *Composite Materials Handbook*, McGraw-Hill Book Co., New York NY (1984).

1.12 E. G. Henneke, II, "Reflection-Refraction of a Stress Wave at a Plane Boundary Between Anisotropic Media", *J. Acoust. Soc. Am.*, Vol. 51, pp. 210-217 (1972).

1.13 R. D. Kriz and W. W. Stinchcomb, "Elastic Moduli of Transversely Isotropic Graphite Fibers and Their Composites", *J. Exp. Mechanics*, Vol. 19, pp. 41-49 (November 1987).

1.14 R. A. Kline, "Wave Propagation in Fiber Reinforced Composites for Oblique Incidence", *J. Composite Materials*, Vol. 22, pp. 287-303 (1988).

1.15 M. T. Kiernan, "A Physical Model for the Acousto-Ultrasonic Method", PhD. Dissertation, College of Engineering, Virginia Polytechnic Institute and State University, Blacksburg VA (December 1989).

2.1 B. A. Auld, *Acoustic Fields and Waves in Solids*, Vol. II, John Wiley and Sons, Inc., New York NY (1973).

2.2 J. S. Heyman and J. H. Cantrell, Jr., "Effects of Material Inhomogeneities on Ultrasonic Measurements: The Problem and a Solution," *Nondestructive Evaluation and Flaw Criticality for Composite Materials, ASTM STP 696,* R. B. Pipes, Ed., Amer. Soc. for Test. and Mater., Philadelphia PA (1979).

2.3 R. A. Kline and Z. T. Chen, "Ultrasonic Technique for Global Anisotropic Property Measurement in Composite Materials", *Materials Evaluation*, Vol. 46, Amer. Soc. Nondestruct. Test. (June 1988)

2.4 R. C. Stiffler, "Wave Propagation in Composite Plates", *Proceedings of the Fourth Annual Review*, Virginia Tech Center for Composite Materials and Structures, Blacksburg VA (1987).

2.5 R. D. Kriz and H. M. Ledbetter, "Elastic Representation Surfaces of Unidirectional Graphite/Epoxy Composites", *Recent Advances in Composites in the United States and Japan, ASTM STP 864*, J. R. Vinson and M. Taya, Eds., Amer. Soc. for Test. of Mater., Philadelphia PA (1985).

2.6 R. N. Bracewell, *The Fourier Transform and Its Applications*, 2nd ed. rev., McGraw-Hill Book Co., New York NY (1986).

2.7 P. M. Gammell, "Coherent Processing of the Full Analytic Signal Information of Ultrasonic Waveforms", *International Advances in Non-Destructive Testing 1984*, Vol. 10, W. McGonnagle, Ed., Gordon and Breach (1984).

2.8 B. T. Smith and A. M. Buoncristiani, "Digital Signal Processing Methods for Ultrasonic Backscattered Waves in Composite Materials", *Ultrasonics Symposium* (1986).

3.1 P. Norton, *Inside the IBM PC*, Brady Books, Prentice Hall Press, New York NY (1986).

3.2 P. Norton, *The Peter Norton Programmer's Guide to the IBM PC*, Microsoft Press, Redmond WA (1985).

3.3 S. Mueller, *Upgrading and Repairing PCs*, Que Corporation, Carmel IN (1988).

3.4 T. Hogan, *The Programmer's PC Sourcebook*, Microsoft Press, Redmond WA (1988).

3.5 *GPIB-PC Users Manual for the IBM Personal Computer and Compatables,* National Instruments Corp., Austin TX (1988).

3.6 A. V. Oppenheim and R. W. Schafer, *Digital Signal Processing,* Prentice-Hall, Inc., Englewood Cliffs NJ (1975).

3.7 *STR825 Analog to Digital Converter User's Manual,* Sonotek, Inc., Springfield VA (1987).

3.8 S. P. Harbison and G. L. Steele, Jr., *C: A Reference Manual*, 2nd ed., Prentice-Hall, Englewood Cliffs NJ (1987).

3.9 B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, 2nd ed., Prentice-Hall, Englewood Cliffs NJ (1978).

3.10 *I 432 I/O PCMotion and PCMotion Plus Boards Users Manual,* Roger's Laboratories, Inc., Santa Ana CA (1987).

3.11 M. Goodwin, *User Interfaces in C*, Management Information Source Press, Portland OR (1989).

3.12 S. Bartlett, *Nondestructive Evaluation of Complex Geometry Advanced Material Components*, Masters Thesis, Virginia Polytechnic Institute and State University, Blacksburg VA (1989).

4.1 R. F. Ferraro, *Programmer's Guide to the EGA and VGA Cards*, Addison-Wesley Publishing Company, New York NY (1988).

6.1 R. C. Heyser, "Determination of Loudspeaker Signal Arrival Times, Part I," *J. of the Audio Engineering Soc.*, Vol. 19, No. 9, pp. 734-743 (October 1971).

6.2 R. C. Heyser, "Determination of Loudspeaker Signal Arrival Times, Part II," *J. of the Audio Engineering Soc.*, Vol. 19, No. 10, pp. 829-834 (November 1971).

6.3 R. C. Heyser, "Determination of Loudspeaker Signal Arrival Times, Part III," *J. of the Audio Engineering Soc.*, Vol. 19, No. 11, pp. 902-905 (December 1971).

The following is a general reference list for gaining an understanding about IBM compatible computer architecture, MS-DOS programming, and "C" and assembler programming in that environment. Note that some of these are repeats of specific references.

R. Duncan, *Advanced MS-DOS*, Microsoft Press, Redmond WA (1986).

R. F. Ferraro, *Programmer's Guide to the EGA and VGA Cards*, Addison-Wesley Publishing Company, New York NY (1988).

M. Goodwin, *User Interfaces in C*, Management Information Source Press, Portland OR (1989).

S. P. Harbison and G. L. Steele, Jr., *C: A Reference Manual*, 2nd ed., Prentice-Hall, Inc., Englewood Cliffs NJ (1987).

T. Hogan, *The Programmer's PC Sourcebook*, Microsoft Press, Redmond WA (1988).

R. Jourdain, *Programmer's Problem Solver for the IBM PC, XT & AT*, Brady Books, Prentice Hall Press, New York NY (1986).

R. Jourdain, *C Express*, Brady Books, Prentice Hall Press, New York NY (1989).

B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, 2nd ed., Prentice-Hall, Englewood Cliffs NJ (1978).

B. D. Kliewer, *EGA/VGA: A Programmer's Reference Guide*, Intertext Publications / McGraw-Hill Book Company, New York NY (1988).

S. Mueller, *Upgrading and Repairing PCs*, Que Corporation, Carmel IN (1988).

P. Norton, *Inside the IBM PC, Brady Books*, Prentice Hall Press, New York NY (1986).

P. Norton, *The Peter Norton Programmer's Guide to the IBM PC*, Microsoft Press, Redmond WA (1985).

W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes in C*, Cambridge University Press, New York NY (1988).

H. Schildt, *C: The Complete Reference*, Osborne McGraw-Hill, Berkeley CA (1987).

H. Schildt, *C: Power User's Guide*, Osborne McGraw-Hill, Berkeley CA (1988).

H. Schildt, *Advanced C*, 2nd ed., Osborne McGraw-Hill, Berkeley CA (1988).

R. T. Stevens, *Graphics Programming in C*, M & T Books, Redwood City CA (1988).

Appendix A. Program Listings

The source listings for the scanning program (testcscn.exe) and the plotting program (scanplot.exe) are followed by lists of the subroutines, with short descriptions, of the hardware and video driver libraries (excluding the standard libraries of Microsoft C).

```
/*****************************************************************
*  Main program for the FLEXSCAN system
*   Michael R. Horne 2/5/90
*/

#include <dos.h>
#include <stdlib.h>
#include <stdio.h>
#include "windows.h"

void helpfile(void);

extern movbridg();
extern setparm();
extern scanfly();
extern scanstep();

static MENU menu[]={
    {"Bridge: move and initialize origin location",  0,movbridg,helpfile},
    {"GPIB: startup equipment on IEEE 488 bus",      0,helpfile,helpfile},
    {"Parameters: set A/D board parameter settings", 0,setparm,helpfile},
    {"Flyscan: set scan area, step size and start test",0,scanfly,helpfile},
    {"Stepscan: set scan area, step size and start test",0,scanstep,helpfile},
    {"Exit program"}};

/*****************************************************************
*  Controlling menu: Inits equipment and calls the subroutines
*/
main()
{
    _menu_att=0x30;         /* Menu color, highlight, and hotkey codes */
    _menu_highlight=0x47;
    _menu_hotkey=0x34;
    save_initial_video();
    startpcm();      /*inits PCMotion stepper motor board*/
    startstr();      /*inits Sonotek STR_825 a/d board*/
    clearscreen(1,1,25,80,0);
    while(!popup(6,menu,10,40));
    exit(0);
}

/*****************************************************************
* Menu helpfile
*/
void helpfile(void)
{
    display_error("Helpfile");
}
```

```
/*****************************************************************
* cscntask.c
*  library of cscan functions
*  by Michael R. Horne loosely based on SONOTEK, INC teste program
*  2/5/90
*/

#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "str_plx.h"                    /*plotting structure*/
#include "str_str.h"                    /*str board structures*/
#include "cscntask.h"
#include "windows.h"
#define NCOM 27
#define VMOD 16

/*the following arrays store data from  str boards chan a and chan b*/
    char buff[2][4096] = {0};       /*stores digitized points for adc*/
    char temp[512];

/*global variables*/
    int mov=2;
    int movit[4]={1,32,128,512};    /*amount of cursor movement*/
    int ireg;                       /*palette register number*/
    int ipal[16]={0,1,2,3,4,5,7,7,8,9,10,11,12,13,14,15};
    unsigned char status[16];       /*a/d board status*/
    unsigned char iout,inn;
    unsigned int ick;
    int adoff=0;                    /*a-d rearm switch (if 1 dont rearm)*/
    int w=0;                        /*active plot number 0 or 1 */
    int iscroll=0;
    int tdel[2]={0,0};
    int vmod=VMOD;
    float rate[11]={.1953125,.390625,.78125,1.5625,3.125,6.25,12.5,
                    25.,50.,100.,200.};
    int ipm[2]={2,2};               /* initialize plotting modes */

/*Structures*/
    struct str sa=
{1,7,0,0,4096,3,0,0,1,0,0,128,128,32768,0xd400,0x250,512,1,0,0,0,2,0,25,1.,1.};
    struct plot sp[2]={{1,1,640,0, 222, 94,349,2,0,-1,2,0},
                       {1,1,640,0, 222, 94,349,2,0,-1,2,0}};
    struct gatevars{
        int gstart;
        int glen;
        int pkamp[2];
        int pkpos[2];
    }pg;
    struct gatevars pg={0,100,0,0};
```

```
/*****************************************************************
* scanfly():does on-the-fly scanning in a rasterized motion, stepper motors
*          move bridge while the data acquisition continues
* NOTE: this prog only stores channel 0 data
*/
int scanfly()
{
    long int wait;
    int no_x,no_y,home_x,home_y;
    int x,y_steps,x_steps,size_step,dir,nwaves;
    float size_x,size_y;
    int oldmode;
    int ichar;
    FILE *openfile(),*filepnt;
    char far *mot1flag;
    char far *mot3flag;

    oldmode=getmode();                    /* get the video mode */
    mot1flag=(char far *) 0x98400092L;    /* motor run flag addresses */
    mot3flag=(char far *) 0x984001beL;

    printf("\rInput x size(in), y size(in),integer x increment(.001 in)\n");
    scanf("%f %f %i",&size_x,&size_y,&size_step);
    no_y=(int)(size_y*1000.);
    no_x=(int)(size_x*1000./size_step);
    printf("\rNumber of x steps: %i \n",no_x);
    printf("\rNumber of y steps: %i \n",no_y);
    filepnt=openfile();
    fwrite(&no_x,sizeof(int),1,filepnt);
    fwrite(&no_y,sizeof(int),1,filepnt);
    for(x=0;x<=no_x;x++){
        if(kbhit()){                      /* check for a keyboard hit */
            ichar=getkey();               /* do this if hit key */
            if(ichar=='q' || ichar=='Q'){
                mode(oldmode);
                clearscreen(1,1,25,80,0);
                return(0);
            }
        }
        dir=(x%2);
        for( wait=0; wait<=40000; ++wait);
        fly('y',dir,no_y);
        while(*mot3flag){
            strbuf(buff);
            peakgate();
            fputc(pg.pkamp[0],filepnt);
        }
        for( wait=0; wait<=40000; ++wait);
        if( x < no_x) step( 'x', 0, size_step);
        fputc( '\n', filepnt);
    }
    home_x=size_x*1000;
```

```
    step('x',1,home_x);
    for( wait=0; wait<=20000; ++wait);
    if(no_x%2 == 0){
        home_y=size_y*1000;
        step('y',1,home_y);
    }
    fclose(filepnt);
    clearscreen(1,1,25,80,0);
    return(0);
}

/*********************************************************************
* scanstep(): does scanning in a grid pattern with the data aquisition
*           occuring when bridge is not moving
* NOTE this prog only stores channel 0 data
*/
int scanstep()
{
    FILE *openfile(),*filepnt;
    int oldmode;
    long int wait=8000;
    int i;
    int x,y,no_xstep,no_ystep,no_xdata,no_ydata,home_x,home_y;
    int x_step,y_step,dir;
    float x_size,y_size;
    int ichar;

    oldmode=getmode();        /* get the video mode */
    printf("\rInput x and y size(in),x and y stepsize(.001 in),wait(8000)\n");
    scanf("%f %f %i %i %li",&x_size,&y_size,&x_step,&y_step,&wait);
/* calc  no. of steps */
    no_ystep=(int)(y_size*1000./y_step);
    no_xstep=(int)(x_size*1000./x_step);
    printf("\rNo. x steps (+1 for no. pnts): %i \n",no_xstep);
    printf("\rNo. y steps (+1 for no. pnts): %i \n",no_ystep);
    filepnt=openfile();
    no_xdata=no_xstep+1;
    no_ydata=no_ystep+1;
    fwrite(&no_xdata,sizeof(int),1,filepnt);
    fwrite(&no_ydata,sizeof(int),1,filepnt);
    printf("SCANNING on line\n ");
    for(x=1;x<=no_xstep;x++){
        printf("\r %i",x);
        if(x>1)step('x',0,x_step);
        for( i=0; i<=wait; i++);
        strbuf(buff);
        peakgate();
        fputc(pg.pkamp[0],filepnt);
        for(y=1;y<=no_ystep;y++){
            if(kbhit()){        /*check for a keyboard hit*/
                ichar=getkey();      /*do this if hit key*/
                if(ichar=='q' || ichar=='Q'){
```

```
                mode(oldmode);
                clearscreen(1,1,25,80,0);
                return(0);
            }
        }
        dir=!(x%2);
        step('y',dir,y_step);
        for( i=0; i<=wait; i++);
        strbuf(buff);
        peakgate();
        fputc(pg.pkamp[0],filepnt);
    }
    fputc('\n',filepnt);
}
home_x=no_xstep*x_step;
step('x',1,home_x);
if(no_xstep%2 == 1){
    home_y=no_ystep*y_step;
    step('y',1,home_y);
}
fclose(filepnt);
clearscreen(1,1,25,80,0);
return(0);
}

/*********************************************************
* setparm(): changes the board parameters based on
*           the menu changes made by the user
*/
int setparm()
{
    int ichar,statchar,i,ier;          /*local variables*/
    int itemp;
    FILE *fptr;
    int oldmode;

/* global */
    sp[0].type=sa.type;
    sp[1].type=sa.type;

/* get the video mode */
    oldmode=getmode();

/*initialize for plotting*/
    mode(16);                              /*set video mode 640 * 200*/
    vigen(16);                            /*generate video lookup table*/
    attrib(&sp[0]);
    attrib(&sp[1]);
    grid(0,20,94,16,1,5);                 /*plot coord. grid */
    dspcom();
```

```
/*** update display ***/
    for(i=0;i<NCOM;i++)showpar(i);
    plotgate(1);

/*data collection and plotting loop*/
    while(1){
        if(kbhit()){        /*check for a keyboard hit*/
            ichar=getkey();      /*do this if hit key*/
            plotgate(0);
            statchar=setdsp(ichar);
            plotgate(1);
            if(statchar=='q' || statchar=='Q'){
                mode(oldmode);
                clearscreen(1,1,25,80,0);
                return(0);
            }
        }
        ier=strbuf(buff);
        itemp=sp[0].idel;
        sp[0].idel=0;
        if(ier == 0)  fplot(&ipm[0],&sp[0],buff[0]);
        sp[0].idel=itemp;
        if(sa.nchan == 2){     /*plot channel 2 data*/
            itemp=sp[1].idel;
            sp[1].idel=0;
            if(ier == 0)  fplot(&ipm[1],&sp[1],buff[1]);
            sp[1].idel=itemp;
        }
       /* if(ier==0)peakgate(); */
    }
}

/*****************************************************************
* setdsp(): check which key was struck and adjust
*           parameter display
*/
int setdsp(ichar)
int ichar;
{
    int f1=0x3b00,f2=0x3c00,f3=0x3d00,f4=0x3e00,f5=0x3f00;
    int f6=0x4000,f7=0x4100,f8=0x4200,f9=0x4300,f10=0x4400;
    int af1=0x6800,af2=0x6900,af3=0x6a00,af4=0x6b00,af5=0x6c00;
    int af6=0x6d00,af7=0x6e00,af8=0x6f00,af9=0x7000,af10=0x7100;
    int iret=0x0d,movl=0x4b00,movr=0x4d00,movup=0x4800,movdn=0x5000;
    int esc=0x1b,home=0x4700,iend=0x4f00, pageup=0x4900,pagedn=0x5100;
    int ins=0x5200,del=0x5300,alt1=0x7800,alt2=0x7900,alt3=0x7a00;
    int alt4=0x7b00,alt5=0x7c00,alt6=0x7d00,alt7=0x7e00,alt8=0x7f00;
    int alt9=0x8000,alt0=0x8100;
    int isave,i;
    unsigned int ti0,ti1,max,min;
    long int ll=0,hl=65535;
    long int lstart,llen;
```

```
    lstart=sa.start;
    llen=sa.len;

/*** do required function for the character hit ***/
    if(ichar == 'q' || ichar == 'Q'){          /*exit to DOS*/
        return(ichar);
    }
    else if(ichar == home){                    /*set delay to zero*/
        sp[w].idel = 0;
        sa.pdel[w]=0;
        tdel[w]=0;
    }
    else if(ichar == 'a' || ichar == 'A'){  /*accumulate screen*/
        if(ipm[w] != 3)ipm[w]=3;
        else if(ipm[w] == 3)ipm[w]=1;
    }
    else if(ichar == 'c' || ichar == 'C'){          /*refresh the screen*/
        setmode(vmod);                     /*set video mode*/
        grid(0,20,94,16,1,5);
        ipm[0]=2;
        ipm[1]=2;
        dspcom(0,0);
        for(i=0;i<NCOM;i++)showpar(i);
    }
    else if((ichar>>8) >= 120 && (ichar>>8) <= 131){   /*change palette*/
        ireg=(ichar>>8)-120;                           /*use alt-i keys*/
        if(ichar == 0x8100)ireg=0;
        ipal[ireg]=ipal[ireg]+1;
        if(ipal[ireg] >63)ipal[ireg]=0;
        setpal(ireg,ipal[ireg]);
    }
    else if(ichar==movl){   /*left arrow key waveform scroll*/
        incdec(1,&sp[w].idel,movit[mov],0,16384-(640*sp[w].iscale));
        sa.pdel[w]=sp[w].idel;
    }
    else if(ichar==movr){   /*right arrow key waveform scroll*/
        incdec(-1,&sp[w].idel,movit[mov],0,16384-(640*sp[w].iscale));
        sa.pdel[w]=sp[w].idel;
    }
    else if(ichar==movup){ /*up arrow key waveform scroll*/
        incdec(1,&sp[w].yoff,movit[mov],72,200);
        attrib(&sp[w]);
    }
    else if(ichar==movdn){  /*down arrow key waveform scroll*/
        incdec(-1,&sp[w].yoff,movit[mov],72,200);
        attrib(&sp[w]);
    }
    else if(ichar== f1  ){          /* vertical resolution*/
        inc(&sp[w].ishift,1,-6,8);
        attrib(&sp[w]);
    }
    else if(ichar== f2 ){    /*horiz resolution*/
```

```
        inc(&sp[w].iscale,1, 1,4);
        if(sp[w].idel + 512*sp[w].iscale > 16384){
            sp[w].idel=16384-512*sp[w].iscale;
            sa.pdel[w]=sp[w].idel;
        }
    }
/*    else if(ichar=='l' || ichar== 'L'){    active channel
        inc(&w,1,0,1);
    }
    else if(ichar== f4 ){        no. of channels
        int dummy;
        inc(&sa.nchan,1,1,2);
        if(sa.nchan == 2 ){
            w=1;
            ipm[1]=2;
        }
        else if(sa.nchan == 1){
            w=0;
            ipm[1]=0;
            fplot(&ipm[1],&sp[1],&dummy);
        }
    }
*/
    else if(ichar== f5 ){      /*sample rate*/
        inc(&sa.isr,   1, 0,10);
        sampra(sa.isr);
    }
    else if(ichar== f6 ){    /*move increment*/
        inc(&mov,1,0,3);
    }
    else if(ichar== f7 ){    /*gstart<-*/
        if(pg.gstart >= movit[mov]) pg.gstart=pg.gstart-movit[mov];
    }
    else if(ichar== f8 ){    /*gstart->*/
        if(pg.gstart <= 639-movit[mov]) pg.gstart=pg.gstart+movit[mov];
    }
    else if(ichar== f9 ){    /* -gate length*/
        if(pg.glen >= movit[mov]) pg.glen=pg.glen-movit[mov];
    }
    else if(ichar== f10 ){    /* +gate length*/
        if(pg.glen <= 639-movit[mov]) pg.glen=pg.glen+movit[mov];
    }
    else if(ichar== 'r' || ichar== 'R'){    /*rectify*/
        inc(&sp[w].irec,2,-1,1);
        attrib(&sp[w]);
    }
    else if(ichar==af1 ){      /*threshold control*/
        inc(&sa.ctlthr, 1,0,31);
        thrctl(sa.ctlthr);
    }
    else if(ichar==af2 ){      /*buffer length*/
        uinc(&sa.nsiz,movit[mov],32,4096);
```

```
    }
    else if(ichar==af3 ){      /*ch1 threshold*/
        inc(&sa.thrch1,movit[mov],0,255);
        thrsh1(sa.thrch1);
    }
    else if(ichar==af4 ){      /*ch2 threshold*/
        inc(&sa.thrch2,movit[mov],0,255);
        thrsh2(sa.thrch2);
    }
    else if(ichar== af6 ){     /*trigger source*/
        inc(&sa.srctr, 1,0,3);
        trgsrc(sa.srctr);
        thrsh1(sa.thrch1);
        thrsh2(sa.thrch2);
    }
    else if(ichar== af7 ){      /* trigger phase*/
        inc(&sa.phastr,1,0,1);
        phatrg(sa.phastr);
    }
    else if(ichar== af8){       /*no. of waves averaged*/
        inc(&sa.navg,1,0,7);
    }
    else if(ichar== f8 ){
        if(sa.pretr == 0)min=0;
        else min=-16384;
        inc(&sa.pdel[w],movit[mov],min,16384);
    }
    else if(ichar==af1 ){
         inc(&sa.pretr, 1,0,1);
         pretrg(sa.pretr);
    }
    for(i=0;i<NCOM;i++)showpar(i);
}

/*****************************************************************
* column locations
*/

     int ix[NCOM]={0,208,416,0,208,416,0,208,416,0,208,416,0,208,416,
                   0,208,416,0,208,416,0,208,416,0,208,416};
     int iy[NCOM]={64,64,64,56,56,56,48,48,48,40,40,40,32,32,32,
                   24,24,24,16,16,16,8,8,8,0,0,0};

/* display command labels at bottom*/
    char com[NCOM][18]=
    {"F1  VertRes(V) :","F2  HorRes(us) :","Display delay  :",
     "              :","              :","Vert Offset    :",
     "F5  Sample Rate:","F6  Move Increm:","                ",
     "F7  <Gate Start:","F8  >Gate Start:","  <arrows>Scroll",
     "F9  -GateLength:","F10 +GateLength:","  <C>lear       ",
     "AF1 Thresh Cont:","AF2 Buff Length:","  <Home>        ",
     "AF3 Ch1 Thresh :","AF4 Ch2 Thresh :","  <A>ccumulate  ",
```

```
    "              :","AF6 Trig Source:","  <R>ectify      ",
    "AF7 Trig Phase :","AF8 # Waves Avg:","  <Q>uit         "};
   static  int iz[NCOM]={136,344,552,136,344,552,136,344,552,
                         136,344,552,136,344,552,136,344,552,
                         136,344,552,136,344,552,136,344,552};
   static char strig[4][8]={"Intern","Thresh","Extern-","Extern+"};
   static char swtch[2][8]={"On ","Off"};
   static char achan[2][8]={"A","B"};

/*****************************************************************
* dspcom(): plot commands on screen
*/
void dspcom()
{
    int i,inv=26;
    for(i=0;i<NCOM;i++) chrplt(com[i],ix[i],iy[i],2);
}

/*****************************************************************
* showpar(): change parameters on screen
*/
void showpar(icom)
int icom;
{

    char pstr[NCOM][8];
    float rat,vscale(),hscale();
    rat=rate[sa.isr];

    if     (icom ==  0)sprintf(pstr[icom],"%-7.3f",vscale());
    else if(icom ==  1)sprintf(pstr[icom],"%-7.3f",hscale());
    else if(icom ==  2)sprintf(pstr[icom],"%-7.3f",sp[w].idel/rat);
    else if(icom ==  3)sprintf(pstr[icom],"       ");
    else if(icom ==  4)sprintf(pstr[icom],"       ");
    else if(icom ==  5)sprintf(pstr[icom],"%-7i",sp[w].yoff);
    else if(icom ==  6)sprintf(pstr[icom],"%-7.3f",rate[sa.isr]);
    else if(icom ==  7)sprintf(pstr[icom],"%-7i",movit[mov]);
    else if(icom ==  8)sprintf(pstr[icom],"       ");
    else if(icom ==  9)sprintf(pstr[icom],"%-7i",pg.gstart);
    else if(icom == 10)sprintf(pstr[icom],"       ");
    else if(icom == 11)sprintf(pstr[icom],"       ");
    else if(icom == 12)sprintf(pstr[icom],"%-7i",pg.glen);
    else if(icom == 13)sprintf(pstr[icom],"       ");
    else if(icom == 14)sprintf(pstr[icom],"       ");
    else if(icom == 15)sprintf(pstr[icom],"%-7i",sa.ctlthr);
    else if(icom == 16)sprintf(pstr[icom],"%-7u",sa.nsiz);
    else if(icom == 17)sprintf(pstr[icom],"       ");
    else if(icom == 18)sprintf(pstr[icom],"%-7i",sa.thrch1-128);
    else if(icom == 19)sprintf(pstr[icom],"%-7i",sa.thrch2-128);
    else if(icom == 20)sprintf(pstr[icom],"       ");
    else if(icom == 21)sprintf(pstr[icom],"       ");
    else if(icom == 22)sprintf(pstr[icom],"%-7s",strig[sa.srctr]);
```

```
    else if(icom == 23)sprintf(pstr[icom],"        ");
    else if(icom == 24)sprintf(pstr[icom],"%-7i",sa.phastr);
    else if(icom == 25)sprintf(pstr[icom],"%-7i",1<<sa.navg);
    else if(icom == 26)sprintf(pstr[icom],"        ");
    chrplt(pstr[icom],iz[icom],iy[icom],14);
}

/*********************************************************
* vscale(): changes vertical scale factor
*/
float vscale()
{
    if(sp[w].ishift >= 0) return (.0625 * (1 << sp[w].ishift));
    else  return (.0625 / (1 << -sp[w].ishift));
}

/*********************************************************
* hscale(): changes horizontal scale factor
*/
float hscale()
{
    return (sp[w].iscale *32./rate[sa.isr]);
}

/*********************************************
* inc(): for incrementing parameters
*/
void inc(value,inc,lolim,hilim)
int *value,inc;
int lolim,hilim;
{
    *value=*value+inc;
    if(*value > hilim) *value=lolim;
}

/*********************************************
* uinc(): for incrementing unsigned parameters
*/
void uinc(value,inc,lolim,hilim)
unsigned int *value;
int inc,lolim,hilim;
{
    *value=*value+inc;
    if(*value > hilim) *value=lolim;
}

/*********************************************
* incdec(): for incrementing and decrementing parameters
*/
void incdec(dir,value,inc,lolim,hilim)
int *value,dir,inc;
int lolim,hilim;
```

```
{
    if (dir == 1){
        *value=*value+inc;
        if(*value > hilim) *value=hilim;
    }
    if (dir == -1){
        *value=*value-inc;
        if(*value < lolim) *value=lolim;
    }
}

/*************************************************************
* startstr(): initializes str825 board
*/
void startstr()
{
    int ichar;
    int vmod=VMOD;
/* inits str825 board */
    stinit();
    clearscreen(1,1,25,80,0);
}

/*************************************************************
* *openfile(): opens a file and returns the file variable
*/
FILE *openfile()
{
    static char *prevfn;
    char *fname=NULL;
    FILE *filevar,*fopen();
/* open file streams */
/*  printf("\r\nPrevious file was : %s",prevfn);   */
    printf("\r\nInput new filename: ");
    scanf("%s",fname);
    if((filevar=fopen(fname,"wb"))==NULL){
      printf("\r\nfile not open, try again");
    }
/*  while(*prevfn++ = *fname++);  */
    return(filevar);
}

/*************************************************************
* wrtphead(): writes header to peak data file
*/
void wrtphead(filevar,nox,noy)
FILE *filevar;
int nox,noy;
{
    fseek(filevar,0L,SEEK_SET);
    fwrite(&nox,sizeof(int),1,filevar);
    fwrite(&noy,sizeof(int),1,filevar);
```

```
}

/*****************************************************************
* plotgate(): plot a gate on the screen (called only when gate changes)
*    0=turn gate off, 1=turn gate on
*/
void plotgate(int funct)
{
    int xpnt,ypnt=200,color;
    if(funct==0){
        for(xpnt = pg.gstart; xpnt <= pg.gstart+pg.glen; xpnt++){
            color=readpixel(xpnt,ypnt);
            color=color^15;
            plotpixel(xpnt,ypnt,color);
        }
    }
    if(funct==1){
        for(xpnt = pg.gstart; xpnt <= pg.gstart+pg.glen; xpnt++){
            color=readpixel(xpnt,ypnt);
            color=color^15;
            plotpixel(xpnt,ypnt,color);
        }
    }
}

/*****************************************************************
* peakgate(): find the peak value in the gate
*/
void peakgate()
{
    int i;
    pg.pkamp[0]=0;
    pg.pkpos[0]=0;
    for(i = pg.gstart;i <= pg.gstart+pg.glen; i++){
        if(buff[0][i] >= pg.pkamp[0]) {
            pg.pkamp[0]=buff[0][i];
            pg.pkpos[0]=i;
        }
    }
}

/*************************************************
* plotpixel(): plot a pixel in graphics mode
* from GRAPHICS PROGRAMMING IN C by R. T. Stevens, pge.62
* EGA only
*/
void plotpixel(int x, int y, int color)
{
    #define seq_out(index,val) {outp(0x3c4,index);outp(0x3c5,val);}
    #define graph_out(index,val) {outp(0x3ce,index);outp(0x3cf,val);}

    unsigned int offset;
```

```
    int dummy,mask;
    char far * mem_address;

    offset=(long)y*80l+((long)x/8l);
    mem_address=(char far *) 0xa0000000l+offset;
    mask=0x80>>(x%8);
    graph_out(8,mask);
    seq_out(2,0x0f);
    dummy=*mem_address;
    *mem_address=0;
    seq_out(2,color);
    *mem_address=0xff;
    seq_out(2,0x0f);
    graph_out(3,0);
    graph_out(8,0xff);
}

/*********************************************************
* readpixel(): reads pixel in graphics mode
* from GRAPHICS PROGRAMMING IN C by R. T. Stevens, pge.62
* EGA only
*/
int readpixel(int x,int y)
{
    #define DISPLAY_OUT(index,val) {outp(0x3ce,index);outp(0x3cf,val);}

    int i,j,color=0;
    unsigned char mask,exist_color;
    char far *base;
    base=(char far *) (0xa0000000l+((long)y*80l+((long)x/8l)));
    mask=0x80>>(x%8);
    for(i=0;i<4;i++)
    {
        DISPLAY_OUT(4,i);
        DISPLAY_OUT(5,0);
        exist_color=*base&mask;
        if(exist_color!=0)color|=0x01<<i;
    }
    return color;
}
```

```
/*****************************************************************
* Header file for cscntask.c library
*  Michael R. Horne 2/5/90
*/

int scanfly();
int scanstep();
int setparm();
int setdsp(int ichar);
void dspcom();
void showpar(int icom);
float vscale();
float hscale();
void inc(int *value,int inc,int lolim,int hilim);
void uinc(unsigned int *value,int inc,int lolim,int hilim);
void incdec(int dir,int *value,int inc,int lolim,int hilim);
void startstr();
FILE *openfile();
void wrtphead(FILE *filevar,int nox,int noy);
void plotgate(int cpnt);
void peakgate();
void plotpixel(int,int,int);
int readpixel(int,int);
```

```
/*****************************************************************
* Scanning bridge task library, steptask.c
* by Michael R. Horne 2/5/90
*/

#include <stdio.h>
#include "steptask.h"
#include "windows.h"

/* stepper motor routines */
    extern INIT1();
    extern INIT3();
    extern FREQ1N2();
    extern FREQ3N4();
    extern MOT1();
    extern MOT3();
    extern MOTX1();
    extern MOTY1();
    extern FLAG1();
    extern FLAG3();

/*****************************************************************
* For moving bridge other than during scanning
*/
int movbridg()
{
  long wait;
  int ichar;
  float stepno;
  int movl=0x4b00,movr=0x4d00,movup=0x4800,
    movdn=0x5000,esc=0x1b,home=0x4700,iend=0x4f00,
    plus=0x4e00,minus=0x4a00 ;
  int invfreq=2000;
  int stepsize=1;

  clearscreen(1,1,25,80,0);

  FREQ1N2(invfreq);
  FREQ3N4(invfreq);

  printf("\rUse arrow keys to move bridge\n\rp & m changes stepsize\n\r");
  while(1){
  stepno=stepsize/1000.;
    printf("\rstepsize(in.): %5.3f",stepno);
    ichar=waitkey();                          /*check for a keyboard hit*/
    if(ichar == 'q' || ichar == 'Q'){         /*exit to menu*/
      clearscreen(1,1,25,80,0);
      return(0);
    }
    else if(ichar=='p'){
      if(stepsize==1)stepsize=stepsize+99;
      if(stepsize>1)stepsize=stepsize+100;
```

```
      if(stepsize>=5000)stepsize=5000;
    }
    else if(ichar=='m'){
      if(stepsize>1)stepsize=stepsize-100;
      if(stepsize<=1)stepsize=1;
    }
    else if(ichar=='4'){
      step('y',1,stepsize);
    }
    else if(ichar=='6'){
      step('y',0,stepsize);
    }
    else if(ichar=='8'){
      step('x',1,stepsize);
    }
    else if(ichar=='2'){
      step('x',0,stepsize);
    }
  }
}

/*****************************************************************
* Initializes pcmotion board
*/
void startpcm()
{
/* pcmotion init fixes unwanted motion when brd is inited*/
  long int wait;
  INIT1();
  INIT3();
  FREQ1N2(5950);
  FREQ3N4(5950);
  MOT1(1,0);
  MOT3(1,0);
  for( wait=0; wait<=20000; ++wait);
  MOT1(1,1);
  MOT3(1,1);
}

/*****************************************************************
* Step control of the scanner bridge by axis direction and
*  no. of steps.  Does nothing until the motion is complete
*/
void step(char axis,int dir,int no)
{
  int invfreq=14875;
  FREQ1N2(invfreq);
  FREQ3N4(invfreq);
  switch(axis){
    case 'x':
      MOT1(no,dir);
      FLAG1();
```

```
      break;
    case 'y':
      MOT3(no,dir);
      FLAG3();
      break;
    }
}

/*************************************************************************
* Does on-the-fly control of the bridge.  Program execution can
*  continue after board initiates motion
*/
void fly(char axis,int dir,int no)
{
  int invfreq=14875;
  FREQ1N2(invfreq);
  FREQ3N4(invfreq);
  switch(axis){
    case 'x':
      MOT1(no,dir);
      break;
    case 'y':
      MOT3(no,dir);
      break;
    }
}

/*************************************************************************
* Header file for steptask.c library
*  Michael R. Horne 2/5/90
*/

void startpcm();
void step(char axis,int dir,int no);
void fly(char axis,int dir,int no);
int movbridg();
```

```
/*****************************************************************
* STR825 board task library (A/D), strtask.c
*  SONOTEK, INC supplied subroutines with minor mods
*/

#define  MEMSIZ 4096
#define  NCHAN  2
#include <stdio.h>
#include <str_str.h>

int stinit();
void setall();
int strbuf(unsigned char buff[NCHAN][MEMSIZ]);

static int ibuff[NCHAN][MEMSIZ];

/* stinit reads 1) the number of boards in sys, and 2) setup paramters
   for each board, then initializes each board. This routine must be run
   before calling any other str825 drivers for proper operation*/

extern struct str sa;

/*****************************************************************
*  Initalizes STR board
*/
int stinit()
{
  float dummy=3.8;
  int i,nbs;
  FILE *fptr;
/* get a/d board setup parameters from setup file if it exists*/
  fptr=fopen("setup.str","r");
  if(fptr != 0){
    fscanf(fptr,"%d",&nbs);      /*get number of boards in system*/
    fscanf(fptr,"%i %i %i %u %u %i %i %i %i %i",
              &sa.on,         &sa.isr,
              &sa.clock,      &sa.start,
              &sa.len,        &sa.wprot,
              &sa.contr,      &sa.pretr,
              &sa.phastr,     &sa.srctr);
    fscanf(fptr,"%i %i %i %u %x %x %u %i %i %i",
              &sa.ctlthr,     &sa.thrch1,
              &sa.thrch2,     &sa.memsiz,
              &sa.memmap,     &sa.port,
              &sa.nsiz,       &sa.etsdel,
              &sa.pdel[0],    &sa.pdel[1]);
    fscanf(fptr,"%i %i %i %i %f %f\n",
              &sa.navg,       &sa.nchan,
              &sa.type,       &sa.brate,
              &sa.vrange[0], &sa.vrange[1]);
  }
  fclose(fptr);
```

```
/*initialize str-825 a/d boards ports and memory maps */

        nbsys(nbs);                     /*set # boards in system*/
        setmap(0,sa.port,sa.memmap,sa.memsiz);

/* initialize boards by setting all parameters (for all boards) */
            boards(0);                  /*set current board*/
            sampra(sa.isr);               /*set sample rate*/
            clksrc(sa.clock);             /*set clock source*/
            contrg(sa.contr);              /*set continuous trigger switch*/
            pretrg(sa.pretr);             /*set pretrigger switch*/
            phatrg(sa.phastr);             /*set trigger phase*/
            mempro(sa.wprot);               /*set a/d memory write protect*/
            thrctl(sa.ctlthr);            /*set threshold control*/
            trgsrc(sa.srctr);             /*set trigger source*/
            thrsh1(sa.thrch1);             /*set thresholds*/
            thrsh2(sa.thrch2);

        boards(0);                      /*make board 0 current board*/
        return(nbs);
}


/*****************************************************************
* Set all of the STR825 board parameters
*/
void setall()
{
  sampra(sa.isr);               /*set sample rate*/
  clksrc(sa.clock);             /*set clock source*/
  contrg(sa.contr);              /*set continuous trigger switch*/
  pretrg(sa.pretr);             /*set pretrigger switch*/
  phatrg(sa.phastr);             /*set trigger phase*/
  mempro(sa.wprot);               /*set a/d memory write protect*/
  if(sa.thrch1 >127)thrctl(9);
  else thrctl(10);
  thrsh1(sa.thrch1);
  trgsrc(sa.srctr);             /*set trigger source*/
  thrsh1(sa.thrch1);             /*set thresholds*/
  thrsh2(sa.thrch2);
}

/*****************************************************************
* take a shot with str*825 board and fill conventional
*  mem buffer "buff"
*/
int strbuf(buff)
unsigned char buff[][MEMSIZ];
{
  int ptlen,pdel[2],nsiz,ier,i,nav,j,len;
```

```
  if(sa.nsiz > MEMSIZ) sa.nsiz = MEMSIZ;
  nav= 1 << sa.navg;
  if(sa.pretr == 0 && sa.pdel[0] < 0)sa.pdel[0] = 0;
  if(sa.pretr == 0 && sa.pdel[1] < 0)sa.pdel[1] = 0;
  if(sa.pretr == 1)len=sa.memsiz/2;
  else len=sa.len;
/* averaging loop */
  for(j=0; j<nav; j++){
/* for 25 Mhz and below*/
        if(sa.isr < 8){
            setsta(sa.start);          /* else ,set start to n*/
            setlen(len);               /*this arms bothchannels*/
            ier=tochk(2);              /*check for record/idle*/
            if(ier != 0)return(-1);
            setcha(0);                 /*set board for ch 1 */
            if(sa.pretr == 1)premov(sa.pdel[0],sa.nsiz,buff[0]);
            else bufmov(sa.pdel[0],sa.nsiz,buff[0]);     /*normal move*/
            if(sa.nchan == 2){
                setcha(1);             /*set board for ch 2*/
                if(sa.pretr == 1)premov(sa.pdel[1],sa.nsiz,buff[1]);
                else bufmov(sa.pdel[1],sa.nsiz,buff[1]);
                }
            }

/*for 50, 100, and 200 Mhz*/

        else{
            nsiz = sa.nsiz >> (sa.isr - 7);
            setcha(0);                 /*set board for ch 1 */
            ier=ets(sa.isr,sa.len,sa.pdel[0],nsiz,buff[0],sa.etsdel);
            if(sa.nchan == 2){
                setcha(1);             /*set board for ch 2*/
                ier=ets(sa.isr,sa.len,sa.pdel[1],nsiz,buff[1],sa.etsdel);
                }
            if(ier != 0)return(-1);
            }

        if(nav > 1){
            if(j == 0){
                accum(0,buff[0],ibuff[0],0,sa.nsiz);
                if(sa.nchan == 2)accum(0,buff[1],ibuff[1],0,sa.nsiz);
                }
            else{
                accum(1,buff[0],ibuff[0],0,sa.nsiz);
                if(sa.nchan == 2)accum(1,buff[1],ibuff[1],0,sa.nsiz);
                }
            }
        }

    if(nav > 1){
        shift(ibuff[0],buff[0],sa.navg,sa.nsiz);
        if(sa.nchan == 2)shift(ibuff[1],buff[1],sa.navg,sa.nsiz);
```

```
        }


/* convert to 2's compliment*/
    if(sa.type == 2){
        twocmp(buff[0],buff[0],sa.nsiz);
        if(sa.nchan == 2)twocmp(buff[1],buff[1],sa.nsiz);
        }

  return(0);
}
```

```
/*************************************************************
*  scanplot.c
*    This program takes peak data as stored by the testcscn
*    program and plots it to the screen in a blue gray-scale.
*    Input: scaling factor (1 -1 pixel/point, 2 -2 pixel/point,
*           and 8 threshold values for partitioning the data
*    Michael R. Horne 2/90
*/

#include <stdio.h>
#include <dos.h>
FILE *infile,*outfile;
int th1,th2,th3,th4,th5,th6,th7,th8;
int bin1=0,bin2=0,bin3=0,bin4=0,bin5=0,bin6=0,bin7=0,bin8=0,bin9=0;

void scaleplt(int,int,int,int);
void dataplot(int,int,int);
void pattnplt(int,int,int);
void colscale();

main(argc,argv)
int argc;
char *argv[];
{
    union inkey{
        char ch[2];
        int i;
    }c;
    register short x,y;
    int no_xdata,no_ydata,ycoord,scale,ichar;
    char dpnt,i,state[4];

/*initialize for plotting*/
    GetStat(state);
    SetPage(0);
/* open file streams */
    infile=fopen(argv[1],"rb");
while(1){
    SetMod(3);
    printf("QUIT y/n\n");

    while(!bioskey(1));
    c.i=bioskey(0);
    if(c.ch[0]=='y' || c.ch[0]=='Y') goto stop;

    printf("Input scale & 8 partition levels(-/+128)\n");
    printf("Bins:B %i %i %i %i %i %i %i %i W\n",
           bin1,bin2,bin3,bin4,bin5,bin6,bin7,bin8,bin9);
    printf("Thresh:B %i %i %i %i %i %i %i %i W\n",
           th1,th2,th3,th4,th5,th6,th7,th8);
    scanf("%i %i %i %i %i %i %i %i %i",
          &scale,&th1,&th2,&th3,&th4,&th5,&th6,&th7,&th8);
```

```
    bin1=0;bin2=0;bin3=0;bin4=0;bin5=0;bin6=0;bin7=0;bin8=0;bin9=0

    SetMod(16);    /*set video mode 640 * 350*/
/*    SetPage(1);    */
/*    printf("\n\n\n\n\n\n\nCALCULATING");  */
    OffVideo();

    ycoord=0;
/** get data recording parameters */
    fseek(infile,0L,SEEK_SET);
    fread(&no_xdata,sizeof(int),1,infile);
    fread(&no_ydata,sizeof(int),1,infile);
    colscale();
/* get channel data */
    for(x=0;x<no_xdata;x++){
        for(y=0;y<no_ydata;y++){
            dpnt=fgetc(infile);
            if(!(x%2)){
                scaleplt(scale,x,y,dpnt);
            }
            else{
                ycoord=no_ydata-1-y;
                scaleplt(scale,x,ycoord,dpnt);
            }
        }
        dpnt=fgetc(infile);  /* gets \n */
    }
/*    SetPage(0);  */
    OnVideo();

    while(!bioskey(1));
    c.i=bioskey(0);
    if(c.ch[0]=='y' || c.ch[0]=='Y') goto stop;
}
stop:
fclose(infile);
SetMod(3);
}
/*************************************************************/
void scaleplt(int scale,int xcoord,int ycoord,int data)
{
    register short i,j;
    int newx,newy,smooth=0;
    xcoord=xcoord*scale;
    ycoord=(int)(350.-(ycoord*scale*350./480.));
    for(j=0;j<scale+smooth;j++){
        newy=ycoord+j;
        for(i=0;i<scale+smooth;i++){
            newx=xcoord+i;
            if(newy<=350 && newx<=640)dataplot(newx,newy,data);
        }
```

```
    }
}
/*************************************************************/
void dataplot(int xcoord, int ycoord, int data)
{

    if(data >= th8){
        pattnplt(xcoord,ycoord,0x00d);
        bin9++;
    }
    if(data >= th7 && data < th8){
        pattnplt(xcoord,ycoord,0x006);
        bin8++;
    }
    if(data >= th6 && data < th7){
        pattnplt(xcoord,ycoord,0x005);
        bin7++;
    }
    if(data >= th5 && data < th6){
        pattnplt(xcoord,ycoord,0x004);
        bin6++;
    }
    if(data >= th4 && data < th5){
        pattnplt(xcoord,ycoord,0x003);
        bin5++;
    }
    if(data >= th3 && data < th4){
        pattnplt(xcoord,ycoord,0x002);
        bin4++;
    }
    if(data >= th2 && data < th3){
        pattnplt(xcoord,ycoord,0x001);
        bin3++;
    }
    if(data >= th1 && data < th2){
        pattnplt(xcoord,ycoord,0x000);
        bin2++;
    }
    if(data >= -128 && data < th1){
        pattnplt(xcoord,ycoord,0x00c);
        bin1++;
    }
    return;
}
/*****************************************************
* pattern_plot() - plots points alternating between
*                  two colors according to a specified
*                  pattern.
* input color as 0xABC where: A=fill no. (0-3),B=background,C=foreground
*/
void pattnplt(int x, int y, int color)
{
```

```
#include <dos.h>

unsigned int mask, pattern[4][8];
int i,foreground,background,fill;

for (i=0; i<8; i++)
{
pattern[0][i] = 0xFFFF;
if (i%2 == 0)
{
pattern[2][i] = 0x4444;
            pattern[3][i] = 0xAAAA;
                }
else
{
pattern[2][i] = 0x1111;
            pattern[3][i] = 0x5555;
}
}
        pattern[1][0] = 0x2020;
        pattern[1][1] = 0x0202;
        pattern[1][2] = 0x8080;
        pattern[1][3] = 0x0808;
        pattern[1][4] = 0x2020;
        pattern[1][5] = 0x0202;
        pattern[1][6] = 0x8080;
        pattern[1][7] = 0x0808;
foreground = color & 0x0F;
background = (color & 0xF0) >> 4;
fill = (color & 0xF00) >> 8;
mask = 0x8000 >> (x % 16);
    if (pattern[fill][y%8] & mask){
        DrawPnt(0,x,y,foreground);
    }
    else{
        DrawPnt(0,x,y,background);
    }
}
/****************load a color scale into pallete registers*********/
void colscale()
{
    char black,blue1,blue2,blue3,blue4,blue5,white;
    black=EnCol(0,0,0);
    blue1=EnCol(0,0,1);
    blue2=EnCol(0,0,2);
    blue3=EnCol(0,0,3);
    blue4=EnCol(1,1,3);
    blue5=EnCol(2,2,3);
    white=EnCol(3,3,3);
    WrPal(0,black);
    WrPal(1,blue1);
    WrPal(2,blue2);
```

```
    WrPal(3,blue3);
    WrPal(4,blue4);
    WrPal(5,blue5);
    WrPal(6,white);
}
/************************/
bioskey(c)
int c;
{
    switch(c) {
        case 0: return get_key();
        case 1: return kbhit();
    }
}
/********************/
get_key()
{
    union REGS r;
    r.h.ah=0;
    return int86(0x16, &r, &r);
}
```

FSTPLT.LIB: Sonotek graphics library

_attrib	Sets plotting attributes
_CHRPLT	Plots character string in graphics mode
_cursor	Positions cursor
_DSPLAF	Displays character string in text mode
_EGINIT	Sets EGA for plotting points
_EGREST	Restore EGA for writing text
_FILMEM	Block store data to addressable memory
_FILSCR	Block fill screen with a character
_FPLOT	Plot waveforms to screen
_getkey	Waits until key is struck and returns its value
_grid	Plots oscilloscope type grid on screen
_mode	Sets video display mode
_PKFIND	Finds peak amplitude and position in data array
_PNTPLT	Plots a single pixel in graphics mode
_vigen	Generates video lookup table for the plotting functions
_XFER	Block move data around in addressable memory

LWINDOWS.LIB: M. Goodwin (*User Interfaces In C*) text menu library

_close_window	Close previously opened window
_cursoroff	Turn cursor off
_cursoron	Turn cursor on
_dialog_menu	Displays a dialog box menu
_display_error	Displays custom error messages
_drawbox	Draws border around text window
_draw_window	Displays a window
_error_handler	Traps hardware errors and displays message
_fillone	Displays one character/attribute pair
_fillscreen	Fills text window with one character/attribute pair
_getcurpos	Gets cursor position
_horizontal_bar	Displays horizontal scroll bar in text window
_hotstring	Displays character string with a hotkey
_open_window	Opens text window
_popup	Displays popup menu
_printcenter	Displays character string centered around column
_printone	Displays one character
_printstring	Displays character string
_pulldown	Implement multiple pull-down menus
_pulldown_bar	Display pull-down menu bar
_restorescreen	Restores saved text window
_savescreen	Saves contents of text window
_save_initial_video	Saves operating environment
_scroll_window	Scrolls contents of text window
_setattrib	Sets text window character attributes to one setting
_setcurpos	Moves cursor
_setcursor	Changes cursor height
_setone	Changes character attribute at one location
_settext80	Initializes LWINDOWS operating environment
_vertical_bar	Displays vertical scroll bar
_waitkey	Waits for key-press and returns value
__menu_att	Global variable: display attribute for menu items
__menu_highlight	Global variable: display attribute for highlighted items
__menu_hotkey	Global variable: display attribute for hotkey characters

MCIBL.LIB: **National Instruments IEEE 488 driver library**

_ibbna	Assign board n to be access board for specified device
_ibcac	Opposite of ibgts
_ibclr	Clears a device
_ibcmd	Command specified board to write commands to GPIB
_ibcmda	Asynchronous ibcmd
_ibdma	Selects DMA I/O for board
_ibeos	Assigns end of string character for rd's and wrt's
_ibeot	Enable/disable sending EOI message
_ibfind	Returns unit descriptor associated with name of device
_ibgts	Causes board to go from active to standby controller
_ibist	Clears board parallel poll flag
_ibloc	Places device in local mode
_ibonl	Reinitialize device and cancel asynchronous I/O
_ibpad	Set primary device address
_ibpct	Passes controller authority to specified device
_ibppc	Remotely configure device for parallel poll
_ibrd	Reads specified number of bytes from a device
_ibrda	Asynchronous ibrd
_ibrdf	File ibrd
_ibrpp	Conduct parallel poll and return result
_ibrsc	Request/release system control authority
_ibrsp	Serially polls a device and returns status response
_ibrsv	Request service from controller
_ibsad	Set secondary device address
_ibsic	Initialize GPIB
_ibsre	Set or clear remote enable line
_ibstop	Stops any asynchronous device operation
_ibtmo	Changes time limit for device operations to complete
_ibtrg	Triggers a device
_ibwait	Waits for an event
_ibwrt	Writes a specified number of bytes to a device
_ibwrta	Asynchronous ibwrt
_ibwrtf	File ibwrt

NEWMSCE1.LIB: Rogers Labs Stepper motor driver library (EGA compatible)

_BINR1	Read single byte from one data port
_BINW1	Write single byte to data port
_FLAG1	Suspend program until motor 1 is finished
_FLAG2	Suspend program until motor 2 is finished
_FLAG3	Suspend program until motor 3 is finished
_FLAG4	Suspend program until motor 4 is finished
_FREQ1N2	Set stepping rate for nonramped logical motors 1 and 2
_FREQ3N4	Set stepping rate for nonramped logical motors 3 and 4
_INIT1	Initialize 1 and 2 timers
_INIT3	Initialize 3 and 4 timers
_MOT1	Run nonramped logical motor 1
_MOT2	Run nonramped logical motor 2
_MOT3	Run nonramped logical motor 3
_MOT4	Run nonramped logical motor 4
_MOTA	Run ramped logical motor A
_MOTB	Run ramped logical motor B
_MOTC	Run ramped logical motor C
_MOTD	Run ramped logical motor D
_MOTX1	Run ramped logical motor X for specified stepping mode
_MOTY1	Run ramped logical motor Y for specified stepping mode

PGEGAVGA.LIB: R. Ferraro (*Programmers Guide to the EGA and VGA Cards*) graphics mode library

_BitMask	Set bit mask register
_Circle	Draw a circle
_CLIPPER	Clip a line to a graphics window
_ClrLine	Clear a line of text
_ClrPage	Clear a page
_ClrWin	Clear a window
_CURSOR	Move cursor position
_DecCol	Decode a color from palette register code to RGB
_DecCols	Decode 16 colors from palette register code to RGB
_DEFAULTP	Enable/disable VGA mode set loading
_DEFCURS	Define cursor shape
_DEMUX	Convert video data from bit-packed to pixels
_DRAWPNT	Draw a pixel
_EnCol	Encode a color from RGB to palette register code
_EnCols	Encode 16 colors from RGB to palette register code
_ErasePal	Erase palette registers from screen (see ShowPal)
_ESetRes	Enable set/reset register
_F25ROW	Set 25 row display
_F43ROW	Set 43 row display
_F50ROW	Set 50 row display
_FONTINFO	Read font information
_GETSTAT	Read display status
_GRAPHCHR	Write a character w/o using BIOS
_GROM8X14	Write 8x14 font in graphics mode
_GROM8X16	Write 8x16 font in graphics mode
_GROM8X8	Write 8x8 font in graphics mode
_GUSER8X8	Write user 8x8 font in graphics mode
_GUSERSET	Write user font in graphics mode
_HLine	Draw a horizontal line
_HLine13	Draw horizontal line in mode x013
_Ital14	Make 8x14 italic font
_Line	Draw a line in 16 color mode
_Line13	Draw a line in mode x013
_LineComp	Load line compare register
_MapMask	Set the map mask register
_MaxLine	Load the maximum rows per character
_ModeReg	Set the mode register
_MUX	Convert video data from pixel to bit-packed
_OFFVIDEO	Turn off the VGA video
_ONVIDEO	Turn on the VGA video
_PELPAN	Load the panning register
_PreRow	Load preset row scan register
_RdBlock	Read active block
_RDCHAR	Read text character/attribute pair
_RDCHARS	Read string of text character/attribute pairs
_RDCOLOR	Read VGA color register
_RDCOLORS	Read multiple VGA color registers
_RDCURS	Read cursor position
_RDFONT	Read font in text mode

_RDINFO	Read VGA state information
_RDPAL	Read from a palette register
_RDPALS	Read from 16 palette registers
_RdWin	Read a window in any video mode
_ReadMap	Read map register
_READPNT	Read a pixel
_ROM8X14	Write 8x14 font in text mode
_ROM8X16	Write 8x16 font in text mode
_ROM8X8	Write 8x8 font in text mode
_SCROLLDW	Scroll text window down
_SCROLLUP	Scroll text window up
_SETLINES	Set lines/page in text mode
_SETMOD	Set the video display mode
_SETPAGE	Set the display page
_SetRes	Set the set/reset register
_ShowPal	Show palette registers on screen
_StartAdr	Load starting address registers
_SUMGRAY	Sum VGA colors to a gray scale
_USERSET	Write user font in text mode
_VIDSTATE	Read VGA video state
_VLine	Draw a vertical line
_VLine13	Draw vertical line in mode x013
_WrBlock	Write active block
_WRCHAR	Write text character/attribute pair
_WRCHARS	Write string of text character/attribute pairs
_WRCOLOR	Write to a VGA color register
_WRCOLORS	Write to many VGA color registers
_WRFONT	Write font in text mode
_WRPAL	Write to a palette register
_WRPALS	Write to 16 palette registers
_WRSTRA	Write string of character/attributes
_WRSTRC	Write string, one attribute
_WRSTRING	Write string of characters only
_WrWin	Write a window in any video mode

STR825.LIB: Sonotek A/D driver library

_ACCUM	Accumulate data (for averaging)
_ARM	Arm specified board for a start position and length
_BANK	Set memory bank number
_BLKMOV	Move 1K blocks from A/D to PC memory
_BOARD	Set current A/D board (does not disable others)
_BOARDS	Set current A/D board (disable all others)
_BUFMOV	Move arbitrary amount from A/D to PC memory
_CHECK	Check if A/D done infinite loop (key press gets out)
_CLKSRC	Set for internal or external clock source
_CONTRG	Set continuous trigger on/off
_CVTBIN	Convert 8-bit character from 2's compliment to binary
_ETS	Enable Equivalent Time Sampling
_FIRE	Fire the board (internal trigger mode)
_FISTAT	Reset finish status bit
_GETRD	Get value read from port
_GETWRT	Get value written to port
_INTSON	Enable interrupts on all A/D boards except current
_MEMPRO	Set A/D memory write protect on/off
_MEMWRT	Switch between bus or A/D access to STR825 memory
_NBSYS	Set the number of A/D boards in system
_PHATRG	Set output trigger phase +/-
_PHSDEL	Set the phase delay (equivalent time sampling)
_PKDET	Enable/disable hardware peak detect (not recommended)
_PREMOV	Move arbitrary amount from A/D to PC memory (pretrig)
_PRETRG	Set pretrigger mode on/off
_RESET	Reset record mode
_SAMPRA	Set the sampling rate
_SETCHA	Set channel number for reading
_SETIEN	Set interrupt enable bit on/off
_SETLEN	Set length of waveform to grab (also arms the board)
_SETMAP	Set port addresses, memory map and effective memory
_SETSTA	Set at what position to start filling memory
_SHIFT	Shift integer to the right (bitwise, divides by 2)
_THRCTL	Set threshold trigger control
_THRSH1	Set channel 1 threshold value
_THRSH2	Set channel 2 threshold value
_TOCHK	Check if A/D done loop (limited time before quitting)
_TRGSRC	Set trigger source internal/external
_TWOCMP	convert 8-bit character from binary to 2's compliment

Appendix B. Equipment and Manufacturers

B.1 Chapter 2 Equipment and Settings

Sending Side:

Hewlett-Packard 3314A Function Generator: 950 Khz, .25 V ampl., 1 cycle sinewave, 45.2 msec rep. rate tone burst output pulse

ENI 2100L Power Amplifier: 50 dB, 10 kHz-12 MHz

Harasonics AB peizoelectric transducer: square .5 in x .5 in, 1.0 Mhz

Harasonics Angle Block: designated for steel

Receiving Side:

Harasonics peizoelectric pointducer: 1 MHz, .030 in dia.

Tektronix AM 502 Differential Amplifier: DC to 1 MHz passband (3 dB points), Gain range 100-200

Data Precision DATA 6000 digitizing oscilloscope:

Timebase- 128 points, 100 nsec/point, 18-20 μsec delay

Input- $\pm$ 1.2 V range

Trigger- Channel 1 Normal, external 160 mV

B.2 Chapter 3 General List of Equipment

Hewlett-Packard 3314A Function Generator

Panametrics Ultrasonic Analyzer Model 5052UA

Panametrics Ultrasonic Pre-Amps

Accu-tron Ultrasonic Pre-Amps Model 3030

Win 286 Computer: IBM AT compatable, 6-10 Mhz

Sonotek STR*825 Analog to Digital Convertor board

Rogers Labs I 432 I/O, PCMotion Stepper Driver Board

National Instruments GPIB Controller Board

Data Precision Model 2020 Polynomial Waveform Synthesizer

ENI Model 240L RF Power Amplifier

Transducers-large selection

B.3 Chapter 4 Specific List of Equipment

Panametrics Ultrasonic Analyzer Model 5052UA

Accu-tron Ultrasonic Pre-Amp Model 3030

Panametrics transducer A313R, 15 MHz, .25 in. dia., 1.5 in. focal length

Win 286 Computer: IBM AT compatable, 6-10 Mhz

Sonotek STR*825 Analog to Digital Convertor board

Rogers Labs I 432 I/O, PCMotion Stepper Driver Board

B.4 Equipment Sources

Accu-tron, Inc.

75 Spring Street

Millis MA 0205 (617) 376-4671

Data Precision Division of Analogic Corp.

Electronics Avenue

Danvers MA 01923 (617) 246-1600

ENI, Inc.

100 Highpower Road

Rochester NY 14623-3498

Hewlett-Packard

Everywhere-check local listings for dealer

National Instruments

12109 Technology Boulevard

Austin, Texas 78727-6204 (512) 250-9119

Panametrics, Inc.

225 Crescent Street

Waltham MA 02254 (617) 376-4671

Rogers Laboratories, Inc.

2727 South Croddy Way Suite E

Santa Ana CA 92704

Sonotek, Inc.

8700 Morrissette Drive

Springfield VA 22152 (703) 440-0222

Vita

Michael R. Horne was born September 28, 1958 in the United States. He graduated from Boonsboro High School, Boonsboro, Maryland in 1976, while concurrently taking classes at Hagerstown Junior College, Hagerstown, Maryland. Upon graduation from the Engineering Transfer curriculum at the Junior College with an Associate of Arts Degree in 1978, he transferred to the Department of Engineering Science and Mechanics at Virginia Polytechnic Institute and State University in Blacksburg, Virginia. He graduated with a Bachelor of Science Degree in 1980. Between his graduation and his return in 1985 to V.P.I. to work on a Master of Science Degree in Engineering Mechanics, he worked as a research engineer at Arctec, Inc. in Columbia, Maryland, an Engineering Computer Support engineer at Grove Manufacturing, Shady Grove, Pennsylvania and as a part-time teacher in Engineering at Hagerstown Junior College.

Michael R. Horne