CALIFORNIA STATE UNIVERSITY, NORTHRIDGE

AN ULTRASONIC IMAGING SYSTEM

A project submitted in partial satisfaction of the requirements for the degree of Master of Science in Computer Science

by

George Vaclav Krestyn

May, 1983
The project of George Vaclav Krestyn is approved:

Ichiro Hashimoto, PhD.

Diane L. Schwartz, PhD.

Philip Gilbert, PhD. (Chairman)

California State University, Northridge
Acknowledgments

I am grateful to my committee members: Dr. Hashimoto, Dr. Schwartz and particularly Dr. Gilbert, for their advice and direction.

Special thanks go to my associates at Jet Propulsion Laboratory: Dr. John Hestenes, Dr. James Rooney and Dr. Ross Larkin for their insights and generous support, and to Dorothy Crawford for her outstanding clerical assistance.

And, I am indebted to my wife, Marya, and children, George and Anna, for their limitless patience.
# Table of Contents

Acknowledgments ........................................................................................................ iii
List of Tables .................................................................................................................. vi
List of Figures ................................................................................................................ vii
Credits ............................................................................................................................ ix
Abstract ......................................................................................................................... x

## Chapter

1. **Introduction and Summary** ...................................................................................... 1

2. **Ultrasonic Imaging** ................................................................................................ 10
   2.1 Fundamental Physics of Ultrasound .................................................................. 10
   2.2 Transducers ......................................................................................................... 13
   2.3 The Pulse-echo Technique ................................................................................ 13
   2.4 Digitization and Quantization .......................................................................... 14
   2.5 Displaying the Data. .......................................................................................... 16
   2.6 Time Delay Spectrometry .................................................................................. 17

3. **Digital Video Image Formation** ............................................................................ 26
   3.1 Digital Images ..................................................................................................... 26
   3.2 Video Display Technology ................................................................................ 27
   3.2.1 Raster Scanning ............................................................................................. 27
   3.2.2 Flicker ............................................................................................................. 28
   3.2.3 Interlace .......................................................................................................... 29
   3.2.4 Video Standards ............................................................................................. 30
   3.3 Frame Buffer Architecture ................................................................................ 31

4. **Image Enhancement** ............................................................................................ 39
   4.1 Definition of Image Enhancement ...................................................................... 39
   4.2 The Histogram .................................................................................................... 39
   4.3 Point operations .................................................................................................. 40
   4.4 Algebraic Operations ......................................................................................... 42

5. **System Hardware Configuration** .......................................................................... 45
   5.1 Kernel ................................................................................................................ 45
   5.1.1 Overview ......................................................................................................... 45
   5.1.2 Multifunction Board ...................................................................................... 45
   5.1.3 512 X 480 X 1 Frame Buffer ......................................................................... 46
   5.1.4 512 X 480 X 8 Frame Buffer ......................................................................... 46
   5.1.5 GPIB Board .................................................................................................... 48
   5.2 Peripherals .......................................................................................................... 48
   5.4 TDS System ........................................................................................................ 48
   5.4 Scanners .............................................................................................................. 48
   5.5 Spectrum Analyzer ............................................................................................. 49
# Table of Contents (continued)

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6. <strong>Software Requirements Analysis</strong></td>
<td>56</td>
</tr>
<tr>
<td>6.1 User Requirements</td>
<td>56</td>
</tr>
<tr>
<td>6.2 User-Computer Software Interface</td>
<td>57</td>
</tr>
<tr>
<td>6.3 Enhancement of User Requirements</td>
<td>59</td>
</tr>
<tr>
<td>6.4 Functional Specification</td>
<td>61</td>
</tr>
<tr>
<td>7. <strong>Software Preliminary Design</strong></td>
<td>63</td>
</tr>
<tr>
<td>7.1 General Design Principles</td>
<td>63</td>
</tr>
<tr>
<td>7.1.1 Modularization</td>
<td>63</td>
</tr>
<tr>
<td>7.1.2 Hierarchical Representation</td>
<td>64</td>
</tr>
<tr>
<td>7.1.3 Module Independence</td>
<td>65</td>
</tr>
<tr>
<td>7.2 Automatic File Naming</td>
<td>65</td>
</tr>
<tr>
<td>7.3 Software Architecture</td>
<td>67</td>
</tr>
<tr>
<td>8. <strong>Software Detailed Design</strong></td>
<td>81</td>
</tr>
<tr>
<td>8.1 Using Different CRT Terminals</td>
<td>81</td>
</tr>
<tr>
<td>8.2 Acquiring Data</td>
<td>83</td>
</tr>
<tr>
<td>8.3 Transforming Data</td>
<td>84</td>
</tr>
<tr>
<td>8.4 Image Display</td>
<td>85</td>
</tr>
<tr>
<td>8.4.1 Clear CAT Frame Buffer</td>
<td>85</td>
</tr>
<tr>
<td>8.4.2 Identity Map</td>
<td>86</td>
</tr>
<tr>
<td>8.4.3 Image Stretching</td>
<td>87</td>
</tr>
<tr>
<td>9. <strong>Software Implementation</strong></td>
<td>90</td>
</tr>
<tr>
<td>9.1 Implementation Language Considerations</td>
<td>90</td>
</tr>
<tr>
<td>9.1.1 Selection Criteria</td>
<td>90</td>
</tr>
<tr>
<td>9.1.2 Comparison of Pascal and C</td>
<td>91</td>
</tr>
<tr>
<td>9.1.2.1 Type Philosophy</td>
<td>91</td>
</tr>
<tr>
<td>9.1.2.2 Pointers and Machine Access</td>
<td>92</td>
</tr>
<tr>
<td>9.1.2.3 Variables and Information Hiding</td>
<td>94</td>
</tr>
<tr>
<td>9.1.2.4 I/O</td>
<td>94</td>
</tr>
<tr>
<td>9.1.3 Compiler and Support Environment</td>
<td>95</td>
</tr>
<tr>
<td>9.2 Packaging</td>
<td>95</td>
</tr>
</tbody>
</table>

**References** | 100

**Appendices**

- A The GPIB (General Purpose Interface Bus) | 105
- B GPIB Handler Source Listing | 114
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 Acoustic Reflection Coefficients — Absolute Values</td>
<td>19</td>
</tr>
<tr>
<td>9.1 Summary of Syntax</td>
<td>97</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>1.1</td>
<td>Artery Wall</td>
</tr>
<tr>
<td>1.2</td>
<td>A section of a porcine carotid artery</td>
</tr>
<tr>
<td>1.3</td>
<td>A longitudinal ultrasound image of a section of a porcine carotid artery</td>
</tr>
<tr>
<td>2.1</td>
<td>The behavior of a wave incident on the boundary between two media</td>
</tr>
<tr>
<td>2.2</td>
<td>Low-frequency sinusoid results from the undersampling that causes aliasing</td>
</tr>
<tr>
<td>2.3</td>
<td>The A-scan and B-scan displays</td>
</tr>
<tr>
<td>2.4</td>
<td>Isometric A-scan composite views of a section of carotid artery</td>
</tr>
<tr>
<td>2.5</td>
<td>Linearly swept TDS signals as a function of time</td>
</tr>
<tr>
<td>2.6</td>
<td>Block diagram of the TDS front-end</td>
</tr>
<tr>
<td>3.1</td>
<td>The scanning process and the video signal</td>
</tr>
<tr>
<td>3.2</td>
<td>Critical fusion frequency as a function of the display luminance</td>
</tr>
<tr>
<td>3.3</td>
<td>A black-and-white frame buffer with one bit-plane</td>
</tr>
<tr>
<td>3.4</td>
<td>A black-and-white frame buffer with N bit-planes</td>
</tr>
<tr>
<td>3.5</td>
<td>A black-and-white frame buffer with video look-up table</td>
</tr>
<tr>
<td>4.1</td>
<td>The linear point operation</td>
</tr>
<tr>
<td>4.2</td>
<td>Contrast stretching: (a) before stretching; (b) after stretching</td>
</tr>
<tr>
<td>5.1</td>
<td>Ultrasound imaging-system block diagram</td>
</tr>
<tr>
<td>5.2</td>
<td>Imaging-system kernel block diagram</td>
</tr>
<tr>
<td>5.3</td>
<td>Imaging-system frame buffer architecture</td>
</tr>
</tbody>
</table>
## List of Figures (continued)

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.4</td>
<td>Frame buffer windowing technique</td>
<td>54</td>
</tr>
<tr>
<td>5.5</td>
<td>Mirror scanning system</td>
<td>55</td>
</tr>
<tr>
<td>7.1</td>
<td>Error curve for normal problem solving</td>
<td>69</td>
</tr>
<tr>
<td>7.2</td>
<td>State diagram for root menu</td>
<td>70</td>
</tr>
<tr>
<td>7.3</td>
<td>State diagram for data-acquisition-routines menu</td>
<td>71</td>
</tr>
<tr>
<td>7.4</td>
<td>State diagram for data-analysis-routines menu</td>
<td>72</td>
</tr>
<tr>
<td>7.5</td>
<td>State diagram for image-routines menu</td>
<td>73</td>
</tr>
<tr>
<td>7.6</td>
<td>Structure chart for acquire-data module</td>
<td>74</td>
</tr>
<tr>
<td>7.7</td>
<td>Structure chart for start-acquisition module</td>
<td>75</td>
</tr>
<tr>
<td>7.8</td>
<td>Structure chart for grab-data module</td>
<td>76</td>
</tr>
<tr>
<td>7.9</td>
<td>Structure chart for end-acquisition module</td>
<td>77</td>
</tr>
<tr>
<td>7.10</td>
<td>Structure chart for transform-data module</td>
<td>78</td>
</tr>
<tr>
<td>7.11</td>
<td>Structure chart for get-input module</td>
<td>79</td>
</tr>
<tr>
<td>7.12</td>
<td>Structure chart for transform-N-data-lines module</td>
<td>80</td>
</tr>
<tr>
<td>9.1</td>
<td>Binary search routine in C</td>
<td>98</td>
</tr>
<tr>
<td>9.2</td>
<td>Binary search routine in Pascal</td>
<td>99</td>
</tr>
<tr>
<td>A.1</td>
<td>An example of the GPIB connected to an S-100 system</td>
<td>110</td>
</tr>
<tr>
<td>A.2</td>
<td>GPIB pin assignments</td>
<td>111</td>
</tr>
<tr>
<td>A.3</td>
<td>GPIB signal lines</td>
<td>112</td>
</tr>
<tr>
<td>A.4</td>
<td>GPIB handshake protocol</td>
<td>113</td>
</tr>
</tbody>
</table>
Credits

Figure

1.1 [DEVE78]
2.1 [WELL77]
2.2 [BUCK81]
2.3 [DEVE78]
3.1 [THOM81]
3.2 [CONR80]
3.3 [SIGG81] (adapted from)
3.4 [SIGG81] (adapted from)
3.5 [SIGG81] (adapted from)
4.1 [CAST79] (adapted from)
5.3 [SIGG81] (adapted from)
7.1 [YOUR79]
9.1 [FEUE82] (adapted from)
9.2 [FEUE82] (adapted from)
A.1 [NATI82] (adapted from)
A.2 [NATI82]
A.3 [LOUG75]
A.4 [MOTO]

Table

2.1 [ERIK74]
9.1 [FEUE82]
ABSTRACT

AN ULTRASONIC IMAGING SYSTEM

by

George Vaclav Krestyn

Master of Science in Computer Science

Ultrasonic imaging is a technique which uses ultrasound waves to allow objects not naturally visible (such as internal anatomical structures) to be visualized. It is an important diagnostic method in medicine. One application is the detection of atherosclerosis, a disease characterized by the progressive narrowing of the lumen (cavity) of an artery by the deposition of various materials on or in the intima (innermost arterial wall). Ultrasonic imaging allows reliable, noninvasive assessment of atherosclerotic disease. A more traditional technique, angiography, requires the injection of a radioactive dye and subsequent acquisition of an X-ray picture, and thus includes risks of infarction, hematoma, infection and X-ray exposure. It is also uncomfortable and relatively costly. On the other hand, ultrasonic imaging is painless (since it does not involve cutting or puncturing the body); entails no known risks, not even from exposure to ultrasound (unlike ionizing radiation such as X-rays); is fast; and is
relatively inexpensive. In fact, there is evidence that ultrasound images can be superior to angiograms in showing fine detail.

This project entails the design and implementation of software for an ultrasonic imaging system which involves computer controlled acquisition of data derived by scanning an object with an ultrasound transducer, followed by digitization, transformation and manipulation of that data to generate a two-dimensional video image.
CHAPTER ONE
Introduction and Summary

Ultrasonic imaging is a technique which uses ultrasound waves to allow objects not naturally visible (such as internal anatomical structures) to be visualized. It is an important diagnostic method in medicine. One application is the detection of atherosclerosis, a disease characterized by the progressive narrowing of the lumen (cavity) of an artery by the deposition of various materials on or in the intima (innermost arterial wall—see Figure 1.1). Ultrasonic imaging allows reliable, noninvasive assessment of atherosclerotic disease. A more traditional technique, angiography, requires the injection of a radioactive dye and subsequent acquisition of an X-ray picture, and thus includes risks of infarction, hematoma, infection and X-ray exposure. It is also uncomfortable and relatively costly. On the other hand, ultrasonic imaging is painless (since it does not involve cutting or puncturing the body); entails no known risks, not even from exposure to ultrasound (unlike ionizing radiation such as X-rays); is fast; and is relatively inexpensive. In fact, there is evidence that ultrasound images can be superior to angiograms in showing fine detail [EVAN78] [DEVE78] [HIRS73] [GREE77].

This project entails the design and implementation of software for an ultrasonic imaging system which involves computer controlled acquisition of data derived by scanning an object with an ultrasound transducer, followed by digitization, transformation and manipulation of
that data to generate a two-dimensional video image.

The imaging system software was successfully implemented by the author according to the functional specifications given in Chapter 6. It is being used routinely for research by the Medical Ultrasound Group at the Jet Propulsion Laboratory (California Institute of Technology). Figure 1.2 shows a section of a porcine carotid artery which was scanned to obtain the digital video image of Figure 1.3, using this software.

Chapters 2, 3 and 4 of this report concern the principles and techniques used in ultrasonic imaging, to the extent that they relate to this project. Chapter 2 reviews the physics of ultrasound, as well as the steps in transforming ultrasound waves into images. In medical applications, ultrasonic imaging is usually accomplished by a pulse-echo technique. Digital ultrasound images are obtained (using techniques similar to those first developed for radar and sonar) by transmitting a pulse of ultrahigh-frequency sound at an object, capturing the returning echoes in the form of numerical quantities, mathematically manipulating that data, and then using a display system to generate an image from the transformed data on a video monitor.

Chapter 3 treats some details of forming a digital video image from the requisite data. The system of this project uses an image formation technique called video gray-scale display. An image is conceptually partitioned into a two-dimensional array whose row and column indices identify a point in the image, such that the corresponding array element, or pixel (picture element), contains the gray level (amount of brightness) of the image at that point. This digital representation of an image is stored in a frame buffer, a
special solid-state RAM arranged as a two-dimensional array, which places the pixels onto a video monitor, such that the frame buffer elements have a one-to-one correspondence with the pixel locations on the screen, thus forming a two-dimensional image on the screen.

Chapter 4 discusses ways of improving the visual quality of an image. After an image has been stored in the computer in the form of gray levels, mathematical operations can be carried out on that data to improve the visual quality of the image by, for instance, increasing contrast, reducing blur or removing noise. For example, an image may contain a noise pattern which may mask important information. Subtracting out that noise pattern would make that information more easily visible. Enhancement techniques may be used to suppress selected features in an image, or to emphasize certain features at the expense of others. Thus, image enhancement can be considered as a selective suppression and emphasis of information in an image in order to increase its usefulness to a user.

Chapter 5 gives the hardware configuration for the specific system used for this project.

Chapters 6, 7, 8 and 9 deal with the imaging-system software, including the requirements analysis, preliminary design, detailed design and implementation of the software. Chapter 6 discusses the user's needs, user-computer interaction issues, and the functional specifications of the imaging-system software. The software is meant for use in an ultrasound research laboratory populated primarily by physicists who use computers when they must in order to accomplish their goals. The computer is one of several tools used in the study of the interaction of ultrasound with tissue. It is important that
the use of a tool does not interfere with the main scientific or engineering pursuit. Accordingly, special sensitivity to the user's needs was a major consideration in the software design.

A key word in user-computer interaction is "user-friendliness." That is, the computer system should be accommodating to the user, taking into account the many limitations of humans and maximizing ease of use by being helpful and forgiving.

A menu-driven software system was designed to facilitate operator interaction for novice and casual users; the computer presents the user with a list of possible actions to be performed by the computer which are appropriate at the time, and prompts him to select one of the options. With many options, a hierarchy of menus is usually best, where selecting an option may result in the display of another menu.

The requirements conveyed by the user were enhanced and supplemented in order to (1) provide additional capabilities likely to be appreciated or later requested by the user, and (2) construct a user-friendly system.

The software is divided into three major subsystems: data acquisition, data analysis, and image display and manipulation. The functional specification of each of these parts is given, taking into account both the user's stated needs and the supplemental requirements.

Chapter 7 describes the software architecture which was obtained using the general design principles of modularization, hierarchical representation and module independence.

In Chapter 8, the procedural description for a few selected structural elements of special importance or interest are presented in a
pseudo-code based on PASCAL and C.

Chapter 9 discusses implementation language considerations and packaging of the software. The implementation language system used for an application has a major impact on the application software's effectiveness and on life-cycle software costs, and so should be selected with careful attention to management and technical considerations instead of such unacceptable criteria as hesitancy in learning a new language or arbitrarily imposed requirements. For this project, a modern, high-level language appropriate for interactive data acquisition was sought. Furthermore, the language had to be one supported under the CP/M operating system with an 8085 CPU as the target processor. Thus, many languages (assembly, Basic, Fortran, Cobol and the like) were easily dismissed, and the choice was narrowed down to either Pascal or C. The primary reasons for finally choosing C are presented in this chapter.

After the conceptual, modular structure of a program is determined, the software must be packaged, that is, assembled into load units (distinct physical units) for execution on a specific machine, with attention to execution speed and memory constraints. Future directions for the software system presented in this report include several possible enhancements. First, execution speed could be optimized by (1) coding critical modules in assembly language, especially image display routines (an image could probably be put up on the screen five to ten times faster in this way), and (2) transferring data from the GenRad to the computer over a 16-bit parallel pathway using interrupt-driven or DMA techniques.

Second, an optional user-computer interaction mechanism not re-
quiring menus could be included for the benefit of the sophisticated user who might find menus unnecessary most of the time.

Third, image display could be improved by automatically overlaying the image with such ASCII notation as date and time. Also, the capability for the user to optionally overlay the image with notation specified by him could be included.

Finally, a three-dimensional reconstruction of an object could be displayed with the capability to view the object from any angle.
Figure 1.1

The artery wall

ARTERY

ENDOTHELium

INTIMA

INTERNAL ELASTIC LAMINA

MEDIA

ADVENTITIA

ELASTIC FIBERS

SMOOTH+MUSCLE CELLS

COLLAGEN FIBERS AND TISSUE CARBOHYDRATES

BLOOD VESSELS
Figure 1.2

A section of a porcine carotid artery
Figure 1.3

A longitudinal ultrasound image of a section of a porcine carotid artery
CHAPTER TWO
Principles of Ultrasonic Imaging

In medical applications, ultrasonic imaging is usually accomplished by a pulse-echo technique. Digital ultrasound images are obtained (using techniques similar to those first developed for radar and sonar) by transmitting a pulse of ultrahigh-frequency sound at an object, capturing the returning echoes in the form of numerical quantities, mathematically manipulating that data, and then using a display system to generate an image from the transformed data on a video monitor.

2.1 Fundamental Physics of Ultrasound

The frequency range of sound that is audible to humans is 20 Hz (Hertz or cycles per second) to 20,000 Hz. Ultrasound has a lower bound of 20,000 Hz [AIUM80]; in medical ultrasound applications the commonly used range lies between 1 MHz and 15 MHz (mega Hertz) [CARL75].

In general, ultrasound waves obey the same laws of optics as do audible sound waves -- they are reflected and refracted in the same way. (The interaction is more complicated when the ultrasonic wavelength is comparable with or greater than the dimensions of the reflecting object.) Thus, when a wave strikes an interface between two materials, or media, of different characteristic impedances, it is partially reflected at the same velocity with which it approached the
interface and partially transmitted at a velocity corresponding to propagation in the adjacent medium (Figure 2.1). Characteristic impedance \( Z \) is an attribute of a material defined as the product of the material's density \( p \) and the velocity of sound \( v \) through it. That is, \( Z = pv \).

Furthermore, the magnitude of reflection, known as the reflection coefficient, is defined as

\[
R = \frac{(Z_2 \cos \theta_i - Z_1 \cos \theta_t)}{(Z_2 \cos \theta_i + Z_1 \cos \theta_t)}
\]

where \( Z_2 \) and \( Z_1 \) are the characteristic impedances of medium 1 and medium 2, respectively, \( \theta_i \) is the angle of incidence, and \( \theta_t \) is the angle of refraction (as shown in Figure 2.1) [WELL77]. When the direction of wave propagation is perpendicular to the interface, \( \theta_i = \theta_t = 0 \), so that the above equation becomes

\[
R = \frac{(Z_2 - Z_1)}{(Z_2 + Z_1)}.
\]

Therefore, if \( Z_2 = Z_1 \), then \( R = 0 \) and there is no reflection. In other words, the essence of these equations is that the magnitude of reflection is a function of the characteristic impedance mismatch. Table 2.1 shows the reflection coefficients for some medically significant interfaces.

This is very significant for diagnostic purposes. The difference between the characteristic impedance of one soft tissue and another is relatively small, so that the reflection of ultrasound at the boundaries between them is also relatively small. For instance, only 0.5 percent of the incident energy is reflected at the boundary between kidney and fat tissue. Although faint, this echo is large enough to
be detected by a sensitive receiver. Moreover, since most of the ultrasound is transmitted, it can continue to penetrate more deeply into the body, and thus provide information at various depths. A further implication of the principle of the reflection coefficient is that the interface between soft tissue and bone or gas returns a strong echo due to the large impedance mismatch. These reflections tend to mask the weaker reflections from soft tissue boundaries, thus limiting the application of ultrasound diagnosis in these areas.

Another factor which must be taken into account is attenuation, in this case the decrease in ultrasound intensity due to ultrasound wave diffraction, scattering and absorption. In general, soft tissue attenuation is one decibel per centimeter per MHz. A decibel (dB) is a logarithmic measure of the ratio of pairs of wave amplitudes, or wave intensities, where the level of one of these is taken as a reference for comparison with others. Since ultrasound waves are usually generated and detected electrically, relative wave amplitudes can be expressed as ratios of voltages. Expressing these ratios as logarithms is expedient because it is easy in this way to express numbers extending over many orders of magnitude, and because arithmetic manipulation of logarithms is simple (e.g., the product of two quantities is obtained by addition of their logarithms). A decibel is defined as

\[ dB = 10 \log \left( \frac{P_2}{P_1} \right) = 20 \log \left( \frac{A_1}{A_2} \right) \]

where \( P_1 \) and \( P_2 \) are the voltages, and \( A_1 \) and \( A_2 \) are the corresponding wave amplitudes [WELL77] [NEWB80].

Attenuation of echoes increases with distance and depends on the
type of material. This suggests two things. First, the electronic signals received from returning echoes must be amplified to compensate for attenuation. One way to partially achieve this is to apply time-gain compensation, whereby the gain of the receiver is increased with time so that echoes from deeper structures are amplified more than those originating closer to the transducer [WELL77]. Second, the dynamic range, or ratio between highest and lowest signal, is increased. The significance of this will be discussed later.

2.2 Transducers

Ultrasound waves are generated and received by a transducer (a device which converts one form of energy to another). Of the several types of transducers, those used in diagnostic ultrasound are based on the piezoelectric effect, which is reversible, so that the same transducer can be used both for generation and detection. This phenomenon, discovered by Pierre and Jacques Curie in 1880, is that when a mechanical stress is applied to certain substances, an electrical field observable as a voltage will be induced, and visa versa [BLIT71] [WELL77] [CARL75]. The piezoelectric effect is manifested by several materials, the most commonly used being barium titanate, lithium sulfate and lead zirconate titanate. Recently, a strong piezoelectric effect has been demonstrated in elongated polarized polymer films, such as polyvinylidene fluoride (PVDF).

2.3 The Pulse-echo Technique

A typical pulse-echo imaging system uses a single piezoelectric
transducer which is coupled to transmitter and receiver electronics. The transmitter circuit applies a burst of voltage to the transducer piezoelectric element, causing it to oscillate and thus to produce ultrasound pulses. This pulse travels through the subject until an interface is encountered, at which time it is reflected as an echo. The strength of the echo depends on the characteristic impedance mismatch, attenuation and angle of incidence of the transmitted ultrasound wave (the greater the angle of incidence, the greater the reflection. Specular reflection is most intense, resulting from an angle of 90 degrees to the interface). While the pulse is traversing the subject, the imaging system is in receiving mode to respond to returning echoes. When an echo arrives at the transducer, it compresses the transducer, so that an electrical potential is generated. This is detected and processed by the receiver electronics, and presented as a radio-frequency (rf) signal. After amplification, this analog signal is digitized (converted to digital form) for subsequent ingestion by computer [CARL75].

2.4 Digitization and Quantization

Digitization is comprised of two basic steps, sampling and quantization. First, the continuous analog signal must be sampled. That is, a finite number of points must be selected in order to represent the analog signal in the noncontinuous digital domain. A sample-and-hold circuit samples the analog input and provides an output which it holds fixed until it can be converted to a discrete quantity by an analog-to-digital converter (A/D converter).
The fundamental sampling theorem states that in order to reconstruct a signal from its discrete representation, the original signal must be sampled at a rate equal to or greater than twice its maximum frequency component. That is, if the highest frequency present in the signal is \( f \), then the signal must be sampled at \( 2f \) samples/sec or more. If this is not observed, attempts to recover the original signal result in aliasing (Figure 2.2). To determine the minimum sampling rate, the maximum frequency component can be determined by a Fourier analysis of the signal. An example of aliasing taken from common experience is the phenomenon of stroboscopic stop action demonstrated in Western movies by the wheels of a wagon moving in the forward direction. The wheels sometimes seem to move forwards, sometimes seem to stand still, and sometimes seem to move backwards. The movie camera samples the scene being photographed by opening its shutter 24 times per second. When these 24 frames/sec are projected on a screen, our eye-brain system integrates the frames, thus giving the appearance of motion. Suppose the wheel has \( N \) spokes. If the wheel turns exactly \( 1/N \) revolutions between frames, it stands still in our perception. If it turns more than \( 1/N \) revolutions between frames, it appears to go forward, and if it turns less than \( 1/N \) revolutions, it seems to go backwards.

After sampling, the sample-points of the signal must be quantized (converted to an integer value) by an A/D converter. Once this is done, the resulting numbers can be presented to a computer for a various kinds of processing [BUCK81] [SHOR81] [FOST81] [FINK75] [BRIG75] [BOUR81] [AUSL81] [CAST79].
2.5 Displaying the Data

The final step is displaying the data. Of the several kinds of display, the single-line A-scan, isometric A-scan composite, B-scan and digital video displays are of prime interest here. In the A-scan (amplitude modulated) presentation, the horizontal axis represents the time required for an echo to travel from an interface to transducer, and the vertical axis represents echo amplitude (strength of signal). The isometric A-scan composite display is achieved by plotting several single-line A-scans isometrically, with hidden-line removal. Figure 2.3 shows two isometric A-scan views of a section of a carotid artery. The B-scan (brightness modulated) display represents echoes as spots of light, such that the intensity of the spot represents the echo amplitude, and the position of the spot corresponds to the position of the interface reflecting the echo, as in the A-scan display (Figure 2.4). A digital video image, an example of which was given in Figure 1.3, may be formed by a process explained in Chapter 3.

Because of the great variation in anatomical structure geometry, interface impedance mismatches and depths of imaging, the dynamic range of echoes is fairly large – on the order of 100 dB (10,000 : 1). Yet, the dynamic range of most display devices is approximately 20 dB (20 : 1) [Koss76] [Griff75]. Therefore, the received signal is subjected to logarithmic processing, so that the resulting signal is effectively compressed in a way that is commensurate with the display device [Erik74] [Heus78].
2.6 Time Delay Spectrometry

The system of this project uses a relatively new technique known as Time Delay Spectrometry (TDS). Although the same fundamental principles and techniques apply both to pulse-echo imaging and to TDS imaging, TDS is a swept-frequency technique which can provide either spectral information or improved time (range) resolution. The mathematical details of TDS are provided in a paper by its inventor [HEYS74]. A TDS transducer emits a linearly swept range of frequencies covering 1 to 10 MHz in a time period of 20 milliseconds, and after a 5 millisecond reset time, repeats the sweep. The received signal is coherently mixed with the transmitted signal in a balanced mixer. Then, it is low-pass filtered to remove unwanted high frequency components. Because of the long-duration, continuous transmission of ultrasound energy, echo signals from deep-lying tissues arrive at the transducer while it is still transmitting. To separate these signals, TDS uses a tracking filter which is tuned to accept time-delayed echo returns while rejecting all others. The time delay with which the received signals arrive at the transducer depend on the path length and the propagation velocity, and is equivalent to a frequency offset, since the transmitter frequency is swept. Hence, any equivalent time interval can be selected by selecting an appropriate offset frequency and bandwidth for the tracking filter. For the currently used linear sweep, the discrimination of arrival times is shown schematically in Figure 2.5 and is related to the filter bandwidth by

\[ \text{delta } T = \frac{B}{(df/dt)} \]
where \( \delta T \) is the time discrimination; \( B \) is the receiver bandwidth; and \( df/dt \) is the frequency sweep rate. In Figure 2.5, (1) is the frequency of the transmitted signal as a function of time, (2) is the frequency versus time curve of the desired signal, (3) represents the same function for a spurious reflection, and (4) represents the frequencies to which the receiver is sensitive.

The filtered signal is down-converted to baseband so that the closest image point corresponds to 0 MHz. These baseband signals are converted by fast Fourier transform (FFT) from frequency values into equivalent digital range values with twelve bits of significance. Finally, these numbers are digitally transformed into eight bit data which is appropriate for the formation of a digital video image, as is discussed in the next chapter. Figure 2.6 is a block diagram of the TDS front end.
Table 2.1

Acoustic Reflection Coefficients — Absolute Values

<table>
<thead>
<tr>
<th></th>
<th>Water</th>
<th>Fat</th>
<th>Muscle</th>
<th>Skin</th>
<th>Brain</th>
<th>Liver</th>
<th>Blood</th>
<th>Skull-Bone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Water</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fat</td>
<td>.047</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Muscle</td>
<td>.02</td>
<td>.067</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Skin</td>
<td>.029</td>
<td>.076</td>
<td>.009</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Brain</td>
<td>.007</td>
<td>.054</td>
<td>.013</td>
<td>.022</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Liver</td>
<td>.035</td>
<td>.049</td>
<td>.015</td>
<td>.0061</td>
<td>.028</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Blood</td>
<td>.007</td>
<td>.047</td>
<td>.02</td>
<td>.029</td>
<td>.00</td>
<td>.028</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Skull-bone</td>
<td>.57</td>
<td>.61</td>
<td>.56</td>
<td>.57</td>
<td>.55</td>
<td>.57</td>
<td>.57</td>
<td></td>
</tr>
<tr>
<td>Lucite</td>
<td>.35</td>
<td>.39</td>
<td>.33</td>
<td>.32</td>
<td>.34</td>
<td>.32</td>
<td>.35</td>
<td>.29</td>
</tr>
</tbody>
</table>
The behavior of a wave incident on the boundary between two media
Low-frequency sinusoid results from the undersampling that causes aliasing. Note that the sampling pulses define this low-frequency waveform as well as the higher frequency undersampled sinusoid.
ECHOES OF PULSES ARE DISPLAYED on a cathode-ray tube, yielding information about the internal structure of the body. Two different types of display are shown. The upper one is the A-scan display, which is a graph of peaks showing the echoes received with respect to time. Strength of each echo is indicated by height of its peak. The first echo received is from the abdominal wall, the second is from the near side of the organ, the third is from the far side of the organ and the fourth is from the vertebra. Echo from the far side of the organ is fainter than the one from the near side because the ultrasound is attenuated as it penetrates deeper into the body. The echo from the vertebra is quite strong, however, because bone reflects a large fraction of the ultrasound energy and soft tissues reflect only a small fraction of it. The lower display is the B-scan display, which represents the echoes on the cathode-ray tube as a series of bright spots. Again the position of each spot represents the time required for the echo to return, but the strength of each echo received is represented by the brightness of its spot.

Figure 2.3

The A-scan and B-scan displays
Figure 2.4

Isometric A-scan composite views of a section of carotid artery
Figure 2.5
Linearly swept TDS signals as a function of time

\[ \Delta T = \frac{\partial \phi}{\partial t} \]
Figure 2.6

Block diagram of the TDS front-end
CHAPTER THREE

Digital Video Image Formation

The system of this project uses an image formation technique called video gray-scale display, which is reviewed in this chapter.

3.1 Digital Images

The only attribute of monochromatic ("one color") light such as that seen on a black and white video monitor is intensity (amount of brightness). A monochrome image can be described as a two-dimensional light intensity function $f(x,y)$, where $x$ and $y$ denote spatial coordinates and the value of $f$ at any point $(x,y)$ is proportional to the brightness of the image at that point [GONZ77].

The intensity of a monochrome image $f$ at coordinates $(x,y)$ is called the gray level ($L$) of the image at that point, where $L$ lies in the range

$$L_{\text{min}} \leq L \leq L_{\text{max}}$$

The interval $[L_{\text{min}}, L_{\text{max}}]$ is called the gray scale. Typically, this interval is numerically equivalenced to the interval $[0, L_{\text{max}}]$, where $L = 0$ is considered full black and $L = L_{\text{max}}$ is considered full white. Thus, the gray scale is the continuous gradation of shades of gray from black to white [GONZ77].

A digital monochrome image (referred to simply as image hereafter) is an image $f(x,y)$ whose spatial coordinates and brightness levels
Digitization of the spatial coordinates is referred to as image sampling, and digitization of the brightness level is termed gray-level quantization (the gray scale is divided into a number of discrete gray levels). Such an image can be conceptually partitioned into a two-dimensional array whose row and column indices identify a point in the image, such that the corresponding array element contains the gray level of the image at that point. These small regions of an image, or digital image array elements, are called pixels (picture elements). In digital image processing, the number of gray levels is an integer power of two. That is, a pixel N bits wide can have any one of \(2^N\) levels of gray; thus, an 8 bit pixel has 256 possible gray levels.

3.2 Video Display Technology

3.2.1 Raster Scanning

The image is formed on a black-and-white video monitor, such as a television. A video monitor has two principal components — a cathode ray tube (CRT) and beam positioning circuitry. The CRT has a phosphor coated screen, electron gun (cathode), focusing system and beam deflection plates. When the electron gun filament is heated, it emits a ray of electrons which is focussed into a sharp beam and directed by means of the horizontal and vertical deflection plates onto a specific point on the screen. As soon as the beam strikes one of the phosphor dots that make up the coating on the inside of the screen, it causes the dot to glow in proportion to the strength of the beam. A video signal, which is a continuously varying voltage, governs the electron
gun so that it modulates the electron beam as it is being scanned, controlling how many electrons it shoots and when. This determines the brightness (intensity) of each scan line along its length. The electron beam is scanned from left to right, top to bottom, beginning with the first scan line of the raster. At the end of each horizontal scan from left to right, a horizontal synchronization pulse causes the beam intensity to be decreased to an invisible level (blanked) and returned to the left at an accelerated rate; this return path is termed the horizontal retrace. Similarly, when the beam reaches the bottom of the screen, a vertical synchronization pulse causes it to be returned to the top; this return path is the vertical retrace [KIVE73]. Figure 3.1 diagrams the video signal (for two lines) and the scanning process.

Each line of pixels is called a scan line, and a set of equi­
distant, parallel scan lines comprising the image is known as a raster (from the Latin meaning "rake") [CAST79].

3.2.2 Flicker

These scan lines are volatile. That is, they must be redrawn or they disappear. If the image scanned out on the picture tube, called a frame, is to appear constant and steady to the viewer, rather than to flicker, the refresh rate (number of times per second that a frame is redrawn) must be commensurate with a phenomenon called persistence of vision. This is a property of the eye, not yet completely understood, which is manifested when a visual image remains for awhile after its stimulus is removed, partly because the light-initiated
chemical reactions involved in visual perception decay at a determinate rate. As a result, there exits a temporal frequency for a given light source at which the eye no longer perceives any temporal changes in light level. Although this critical fusion frequency (CFF), also known as critical flicker frequency, is a function of luminance (where luminance is the light source intensity per unit area), as Figure 3.2 shows, it is about 50 Hz for typical viewing luminance. Therefore, if the CRT refresh rate at least matches the CFF, no flicker will be discernible, since the eye will be able to temporally integrate the separate frames into a perception of continuity [KIVE73] [CONR80] [HALL79]. Years ago, the U.S. electric power industry decided to adopt an AC line-voltage frequency of 60 Hz in order to prevent incandescent lights from flickering. Consequently, a television frame rate of 60 times per second was chosen to reduce the effects of interference from the AC power line [CONR80].

A situation analogous to television is a motion picture. Each second, 24 individual frames move through the projector. Because flicker is apparent at this rate, the image of each frame is flashed twice at the screen so that each picture in a frame is actually seen twice. Thus, the effective frame rate is 48 frames per second, resulting in a flicker-free perception of motion, even though the actual frame rate of 24 per second necessitates only half the length of film (one half the "bandwidth") which would be needed for an actual frame rate of 48 per second [KIVE73].

3.2.3 Interlace

In video display, similarly, to reduce the very large frequency
bandwidth required to scan at a rate of 60 new frames per second, a technique known as interlaced scanning is used, which maintains an effective rate of 60 frames per second but an actual rate of 30 per second. This is accomplished by displaying all odd-numbered scan lines in 1/60 of a second, and then all even-numbered lines in 1/60 of a second, so that adjacent lines are displayed 1/60 of a second apart, even though each individual line is scanned every 1/30 of a second. At a normal viewing distance, the eye cannot typically distinguish the adjacent lines, resulting in no perceivable flicker. An example of an exception is the case of a single white horizontal line displayed against a black background. The line is refreshed 30 times a second and is entirely different, thus distinguishable, from its adjacent lines; flicker is then evident. In the case of two adjacent white horizontal lines against a black background, flicker is not perceived since each line is being displayed in 1/60 of a second, and the eye merges the two lines so that the mind thinks it is seeing the same line every 1/60 second of a second [FOLE82].

3.2.4 Video Standards

In the United States, the RETMA (Radio-Electronics-Television Manufacturing Association) standards for scanning specify 525 scan lines per frame. This means that each field contains 262.5 lines. Furthermore, performance standards have been set by the Electronic Industries Association (EIA) for monochrome display systems. This RS-170 standard includes specifications for video signal timing. The vertical sync pulse, which signals the start of the next field, occurs
every 1/60 second. The vertical retrace time, the time required for the beam to travel from the bottom the current field to the top of the next field, is about 1.3 msecs. The horizontal sync pulse signals the start of a new scan line and takes 63.5 usecs:

\[(1 \text{ frame}/525 \text{ lines}) \times (1 \text{ second}/30 \text{ frames}) = 63.5 \text{ usecs/line}\]

The horizontal retrace time, the time required for the beam to travel from the end of the current scan line to the start of the next one, is about 10 usecs. The first 21 lines of each field are not displayed because of the blanking during vertical retrace (vr):

\[(1.3 \text{ msec/vr}) \times (1 \text{ line}/63.5 \text{ usecs}) = 21 \text{ lines/vr}\]

Therefore, 42 lines per frame are invisible, leaving only 483 lines actually visible out of the total of 525 lines per frame [CONR80] [KIVE73] [FOLE82] [CAST].

3.3 Frame Buffer Architecture

The image which is displayed on the screen at the 30 Hz frame rate is stored in a frame buffer, a special solid-state RAM store arranged as a two-dimensional array, the elements of which have a one-to-one correspondence with the pixel locations on the screen. Each frame buffer element contains the gray level of the pixel [FOLE82] [NEWM79].

The frame buffer is conceptually organized as a set of one or more bit planes, where a bit plane is a matrix of 1 bit per element. Each bit plane element at a particular x,y position contributes one bit to the total number of bits representing the pixel at that position. Bit depth refers to the number of bit planes, or equivalently, to the
number of bits per pixel. The simplest frame buffer, diagrammed in Figure 3.3, is one bit deep and is known as a bit map. Two gray levels, usually full black and full white, are possible. With a bit depth of \( N \), \( 2^{N} \) gray levels are possible. Multiple planes can be logically arranged combinatorially. For example, an 8 bit deep frame buffer with 512 rows and 512 columns can represent a single 512 x 512 image of 8 bits of intensity precision, two images of 4 bits precision, eight images of 1 bit precision, 1 image of 4 bits precision and 2 images of 2 bits precision each, and so on. In any case, the total size of this frame buffer is 512 x 512 x 8 bits or 256 K bytes [NEWM79].

As Figure 3.4 shows, the combined digital output of the bit planes (the gray level of the pixel) is transformed by a display controller via a digital-to-analog converter (DAC) into an analog voltage, along with the appropriate video timing signals, which drives the CRT electron gun. The display controller cycles pixel by pixel through the entire frame buffer 30 times per second, synchronizing the frame buffer memory reference with the screen location being currently scanned [FOLE82] [NEWM79].

A feature of many frame buffer systems is the programmable video look-up table, which has one entry for each possible gray level. That is, a frame buffer of bit depth \( N \) requires a table with \( 2^{N} \) elements; an 8 bit deep frame buffer system would have a 255 element table. Instead of feeding the pixel value directly into the intensity DAC, it is used as an index into the look-up table. The table's value at that index is then used to generate the video signal voltage at that point (Figure 3.5). Thus, if the pixel value is 150, the
contents of table location 150 would be sent to the DAC. If the table has a bit width of W, then $2^W$ intensity levels are displayable. Since all frame buffer pixels are filtered through the look-up table which can be easily, dynamically altered, this technique provides a very fast means of changing the appearance of an image without changing the contents of the frame buffer. For instance, if all pixels with values of 200 or above were to be displayed as pure white, table locations 200 through 255 would be set to a value of 255, given that 255 defines a gray level of pure white. This is much more efficiently done than if all the pixel values in question were directly changed; in the case of a 512 x 512 x 8 bit frame buffer, the worst case would involve changing all 262,144 pixels rather than just 56 table locations to achieve the same effect [FOLE82] [CONR80] [NEWM79].
Figure 3.1

The scanning process and the video signal
Figure 3.2

Critical fusion frequency as a function of the display luminance
Figure 3.3

A black-and-white frame buffer with one bit-plane
Figure 3.4

A black-and-white frame buffer with \( N \) bit-planes
Figure 3.5

A black-and-white frame buffer with video look-up table
CHAPTER FOUR
Image Enhancement

4.1 Definition of Image Enhancement

After an image has been stored in the computer in the form of gray levels, mathematical operations can be carried out on that data to improve the visual quality of the image by, for instance, increasing contrast, reducing blur or removing noise. For example, an image may contain a noise pattern which may mask important information. Subtracting out that noise pattern would make that information more easily visible. Enhancement techniques may be used to suppress selected features in an image, or to emphasize certain features at the expense of others. Thus, image enhancement can be considered as a selective suppression and emphasis of information in an image in order to increase its usefulness to a user [ROZE76] [ROZE79] [CANN81] [CAST79].

4.2 The Histogram

A fundamental tool of digital image processing is the gray level histogram, which graphically presents the gray level content of an image. It is a plot of gray levels as a function of the number of pixels in the image that have that gray level, where the abscissa is the gray level and the ordinate is the frequency of occurrence. The histogram can be useful in determining how to enhance the image.
4.3 Point Operations

Point operations are an important set of image enhancement techniques which modify the gray scale of an image. A point operation is a mapping of input pixels to output pixels such that each input pixel corresponds directly to each output pixel, without reference to the surrounding pixels as is the case in a local operation in which a group of neighboring pixels determines the output pixel's gray level. Also, each output pixel corresponds directly to the input pixel that has the same coordinates.

One type of point operation is called gray scale transformation. It is used to change the gray scale throughout an image, or some region of the image, in order to make details of the image more easily visible by increasing contrast. Normally, a digital image should consist of gray levels that occupy most of the dynamic range of the gray scale available, since adjacent gray levels are usually not distinguishable from one another. This gives the impression of a continuously varying gray scale even though the gray scale is in fact quantized. If the gray levels are spread apart, the contrast of the image is "stretched," and detail that was not sharply demarcated is brought out more. For example, an image which only makes use of a portion of the gray scale will have low contrast and so will look "underexposed." The appearance of such an image can be improved by stretching it to compensate for the "underexposure." Figure 4.2(a) shows a low contrast image and its histogram below it. It is clear from the histogram that the gray levels do indeed occupy only a part of the full gray scale range of 256 levels; the range between the
darkest pixel and the lightest pixel is less than 256. Stretching the image increases the contrast, resulting in the image shown in Figure 4.2(b); its histogram reflects the greater spread of gray levels.

On the other hand, even if an image occupies the whole gray scale, it may be advantageous to stretch the contrast in part of the range, at the cost of compressing other regions, in order to emphasize a region of particular interest in the image.

Another point operation is complementing, or inverting, an image (reversing its polarity). This results in an image in which the gray scale is reversed; thus, for example, white becomes black and black becomes white.

The mathematical expression for a point operation, in which the output gray level is a linear function of the input gray level, is

\[ y = f(x) = kx + b \]

where the input point with gray level \( x \) is transformed into the output point with gray level \( y \) (Figure 4.1). Contrast is determined by \( k \): if \( k \) is greater than 1, the contrast of the output image is increased, and if \( k \) is less than 1, the contrast is reduced. Brightness is controlled by \( b \): if \( k = 1 \) and \( b \) is non-zero, the gray levels of all pixels are scaled up or down, making the whole image appear to be darker or lighter. Furthermore, if \( k \) is negative, the image is complemented [CAST79].

Suppose that the allowable range of gray levels in both the output gray scale and the input gray scale is the same, that is, \( n \leq x \leq N \) and \( n \leq y \leq N \). Suppose also that an image occupies a portion of this range, such that \( m \leq x \leq M \), where \([m, M]\) is a subinterval of \([n, N]\).
Then, the image can be stretched \[\text{[ROZE76]} \text{[HALL79]}\] to occupy the full range \([n, N]\) by

\[
y = \left(\frac{N - n}{M - m}\right) (x - m) + n
\]

A piecewise linear transformation may be appropriate if most of the gray levels of the input image are in the subrange \([m, M]\):

\[
y = \begin{cases} 
\left(\frac{N - n}{M - m}\right) (x - m) + n & \text{for } m \leq x \leq M \\
n & \text{for } x < m \\
N & \text{for } x > M 
\end{cases}
\]

This stretches the interval \([m, M]\) of the input gray scale, and compresses the intervals \([n, m]\) and \([M, N]\). Such a transformation is applicable if there are so few points in the the compressed intervals that the information lost in the process is negligible \[\text{[ROZE76]}\].

4.4 Algebraic Operations

Algebraic operations form another class of techniques. They involve the production of an image by the pixel-by-pixel sum, difference, product, or quotient of two input images. Image subtraction, for example, is useful in removing a pattern of background noise from the image. A background image is first acquired. This is then subtracted from subsequent images which have information of interest, so that the subsequent images display only the desired information, and not the background pattern.
Figure 4.1

The linear point operation

\[ f(x) = kx + b \]
Figure 4.2

Contrast stretching: (a) before stretching; (b) after stretching
CHAPTER FIVE

System Hardware Configuration

The overall block diagram of the ultrasound imaging system hardware used for this project is given in Figure 5.1.

5.1 Kernel

5.1.1 Overview

The kernel is a microcomputer system (based on the S-100 bus) for controlling both image acquisition and display; its block diagram is shown in Figure 5.2. Its components are:

1. 5 MHz 8085 CPU
2. 64K of 12 MHz static RAM
3. DMA floppy disk controller
4. 1 Mb RAM disk
5. dual channel serial I/O interface
6. multifunction board
7. parallel port interface
8. 512 X 480 X 1 frame buffer
9. 512 X 480 X 8 dual-port frame buffer
10. IEEE 488 (GPIB) interface

The most important components relevant to this project software are discussed below.

5.1.2 Multifunction Board

The I/O mapped multifunction board includes a crystal-controlled calendar clock with a battery back-up capability. It provides the date and time of day, which are available as BCD digits that can be
randomly read. Either 12 or 24 hour mode can be specified by software. Once the date and time are set on the chip, it will keep both current (even when computer power is turned off, if battery back-up is used) [GODB81].

5.1.3 512 X 480 X 1 Frame Buffer

This I/O mapped bit-map contains an independent Z80 CPU, 32K bytes RAM, and up to 8K bytes EPROM. It also has 6K ROM of resident graphics software, which has two components: one which emulates a text-only "dumb terminal" interface, and the other which includes graphics commands for such operations as point and vector generation. Once the system CPU sends a command to the bit-map over its ports, the CPU can continue with its own processing while the bit-map carries out display generation in parallel. Of its 32K bytes address space, 30K is display memory, which is mapped directly onto the display screen. That is, each of the 245,760 bits in this range appears as a single pixel on the video monitor. A good application of this frame buffer is the display of histograms and plotting data [SCI081].

5.1.4 512 X 480 X 8 Frame Buffer

The architecture type of this dual-port frame buffer, called the CAT (Computer-aided TV), is diagrammed in Figure 5.3. Although the CAT has the capability to display color images, for the purposes of this project it was treated as a gray scale display device by only connecting the output of one color map (video look-up table) to a monochrome video monitor. The interface between host computer and frame buffer
consists of eleven registers aboard the frame buffer which are accessed as I/O ports by the host. Eight of these registers are used by host software to control all activities of the frame buffer, and three are status registers which allow the host to determine what processes are occurring in the frame buffer. The frame buffer memory size is 256K bytes (512 * 480 * 1 byte * 8 bits/byte = 256K bytes). However, it is accessed by the host through a windowing technique diagrammed in Figure 5.4. A fraction of the 64 Kbyte address space of the S-100 bus, either 2K or 8K, is selected as a window into the frame buffer address space. With a 2 K window, the frame buffer memory is divided into 128 pages (128 * 2 K = 256 K). Software specifies the page to be accessed through the window by setting the frame buffer page address in one of its control registers. Thus, to write a value into all 256 K locations of the frame buffer from beginning to end, the software first would have to select page 0, write into its 2 K locations, then select page 1, write into its 2 K locations, and so on until all 128 page had been selected and each of their 2 K locations written into. This method minimizes the amount of system memory occupied by the frame buffer. The look-up tables in the frame buffer are likewise accessed through a window. It is possible to designate the same block of system memory as the window space for both frame buffer memory and look-up tables, since software can select which it will access at any given time. Due to the frame buffer memory's dual-port design, the video monitor is being refreshed via one port even as the host is accessing the buffer contents via the other port [DIGI79].
5.1.5 GPIB Board

This board handles communication between the host CPU and the IEEE 488 bus, otherwise known as the General Purpose Interface Bus (GPIB), via seven write registers and six read registers that are mapped into eight ports. In this system, the GPIB is used to pass digitized data from the spectrum analyzer to the kernel. The board implements all functions specified in the IEEE 488-1978 interface bus standard. A synopsis of the GPIB is given in Appendix I.

5.2 Peripherals

Peripherals on the system include a high-resolution monochrome video monitor, CRT terminal, 26-Mbyte Winchester hard disk, two dual-density 8-inch floppy disks, a video hard-copy unit which records a video monitor image on dry-silver paper, a video printer which provides photographs of the video monitor image (using 35mm, 4 X 5 or SX-70 film), and a VHS-format video cassette recorder.

5.3 TDS System

The TDS system used here was described in Chapter 2.

5.4 Scanners

Either of two different types of scanners can be used in this system. One is a linear scanner which can move the transducer in any of four planar directions (left, right, up or down). Scanning is accomplished by one stepper motor for vertical motion and one stepper
for horizontal motion. Typically, the transducer is scanned horizontally across an object, starting from a zero position set by the user. This requires the user to indicate to the software the number of scan lines to acquire, the number of steps between scan lines, and the stepper delay time (the stepper needs time to settle down after moving).

The other scanner is a mirror system, diagrammed in Figure 5.5. The ultrasound waves are reflected off the mirror, which is moved rotationally to scan the ultrasound across an object. An absolute center position is calibrated for the mirror. Then, for scanning, a secondary scan center relative to the absolute center can be specified by the user, along with the starting point expressed as the angular distance from the relative center, the ending angle (in degrees), and the step size of the mirror (in degrees).

5.5 Spectrum Analyzer

The one-channel spectrum analyzer (GenRad) is controlled by an internal microprocessor operated by software residing in ROM. All acquisition and processing parameters can be controlled by pushbutton selection from menus presented on a raster-scan display, which also provides graphical output of specified operations on the input signal. The input signal is operated on in time segments referred to as frames, such that 512 samples of the signal constitute an input frame. The input frame is digitized by a 10-bit analog-to-digital converter which produces a signed two’s complement binary number for each sample, putting the values into one of two 1024-word by 16-bit input buffers. When a complete frame has been digitized, the GenRad's
microprocessor is notified by means of an interrupt. At this point, a
Fast Fourier Transform (FFT) is performed in-place on that input frame
to produce a 400-line power spectrum of the frequency components that
constitute the input signal, where each element of the spectrum is a
double-precision (32 bit) number. (These spectral elements are trans­
ferred to the imaging-system host software to be transformed into gray
levels.) Concurrently with this operation, another input frame is
being filled into the second input buffer. Input frames can be aver­
aged in one of four modes; for instance, the subtractive mode can be
used to remove background noise.

All the processing parameters, which can be entered either manu­
ally from the front control panel on the instrument or loaded remotely
via software, are stored in a 1024-bit RAM area of 256 4-bit words,
one for each parameter. These can also be read out from the memory by
software.

Remote control via host computer software and data transfer from
and to the GenRad can be done either through a 16-bit parallel pathway
or by means of the GPIB, in binary or ASCII. Data can be transferred
in a number of formats, depending on the data in question, such as 16­
bit raw integer, 32-bit raw integer, or floating point [GENR80].
Figure 5.1

Ultrasound imaging-system block diagram
Figure 5.2

Imaging-system kernel block diagram
Figure 5.3

Imaging-system frame buffer architecture
Figure 5.4

Frame buffer windowing technique
Figure 5.5

Mirror scanning system
CHAPTER SIX
Software Requirements Analysis

This chapter discusses the user's needs, user-computer interaction issues, and the functional specifications of the imaging-system software.

6.1 User Requirements

The user's stated needs were:

* the linear scanner is to be controlled as directed by the user, according to the parameters: number of lines to acquire, number of scanner steps between lines, and stepper delay time;

* the mirror scanner is to be controlled as directed by the user, according to the parameters: relative center in degrees, low limit in degrees, high limit in degrees, and step size in degrees;

* the software should provide a capability of acquiring a calibration spectrum for optional spectral subtraction to remove background noise, as well as subsequent image subtraction;

* the user should be able to select whether transformation of raw data into image data takes place during acquisition or after acquisition;

* provision should be made for plotting of raw and image data after acquisition;

* both raw and pixel data should be saved on disk;

* the image data should be contrast stretched after being formed on the video monitor; and

* the user should be able to specify the number of lines to be averaged by the GenRad.
all command and data transfers involving the GenRad are to be over the GPIB.

6.2 User-Computer Software Interface

The software is to be used in an ultrasound research laboratory populated primarily by physicists who use computers when they must in order to accomplish their goals. The computer is one of several tools used in the study of the interaction of ultrasound with tissue. It is important that the use of a tool does not interfere with the main scientific or engineering pursuit. Accordingly, special sensitivity to the user's needs was a major consideration in the software design.

A key word in user-computer interaction is "user-friendliness." That is, the computer system should be accommodating to the user, taking into account the many limitations of humans and maximizing ease of use by being helpful and forgiving.

For novice and casual users, a menu-driven software system can facilitate operator interaction; the computer presents the user with a list of possible actions to be performed by the computer which are appropriate at the time, and prompts him to select one of the options. With many options, a hierarchy of menus is usually best, where selecting an option may result in the display of another menu. A simple exit from the menu should be provided, as well as an easy return to either a previous menu or the root menu. This system relieves the user of having to remember what the options are, and enables the user to specify an action by means of a single keystroke.

Simplicity is important. This entails the menu structure, prompts, error messages, display layout and keyboard input. Simpli-
city would dictate keeping the terminal screen uncluttered and as free of extraneous material as possible; clearing the screen before putting up a menu is a help.

The user's interaction with the computer should be minimized -- let the computer do the work. This includes prompting only for absolutely necessary information. For example, if the computer can calculate a parameter based on some input, it should do so, rather than prompting the user for it. Also, a default input should be provided. The default selection should be displayed as part of the prompt, and should be activated by means of an easily struck key, such as the space bar or carriage return. Furthermore, although the default should be set to an initial value, in some cases it is appropriate to let the default associated with a prompt assume the last value input by the user in response to that prompt. Such reusability of former input is appropriate when the user wishes to repeat some action or set of actions with the same or nearly the same set of parameters.

Feedback to the user's input should be as complete as possible. This includes an indication of what is happening if the response time is long. If parameters are input, they should be echoed as they are entered, and just before any action takes place which is determined by them. In the case of a sequence of menus, each menu should have a title displayed which describes its sphere of activity. The end of a long process should be indicated auditorily as well as visually -- this allows the user to undertake some activity in parallel in the vicinity of the computer system, without requiring him to stare at the terminal for a message informing him that the process has been completed.

Spatial consistency should be observed. Menu items in the same
relative position within a menu should appear in the same location on
the terminal screen.

Error handling should include meaningful error messages (accompa-
nied by an audible indication) and graceful error recovery [MART73]
[SHNE79] [SHNE80] [SNEE78] [WASS80].

6.3 Enhancement of User Requirements

The requirements conveyed by the user were enhanced and supple-
mented in order to (1) provide additional capabilities likely to be
appreciated or later requested by the user, and (2) construct a user-
friendly system. These additional requirements are:

After a scan is completed, the ultrasound scanner will be reposi-
tioned automatically to the origin of the scan.

Any single-character response to a menu prompt will involve a
single key stroke, instead of the character followed by a carriage-
return.

Files will be named automatically, when possible, with optional
override by the user.

Parameters associated with data acquisition will be saved on disk.

Any of the terminals available in the laboratory should be compa-
tible with the software. This is desirable because three different
terminals are attached to different computer systems throughout the
laboratory, and any given terminal might be on any of the systems on a
given day. The terminals are: (1) the DEC VT100, (2) the TeleVideo
950, and (3) the Hazeltine 1520.

Furthermore, the software will provide the capability to:

* save and restore the GenRad parameters to and from disk;
* send commands to the GenRad interactively from the terminal;
* test scanner operation interactively from the terminal;
* dump the raw data and image data values;
* generate a test image via the CAT frame buffer;
* stretch an image automatically or by explicit input of end-points;
* restore a stretched image to its original form ("linear stretch");
* invert an image;
* generate a histogram, and print it out, display it, save it to disk, or restore it from disk;
* display any part of an image (window) in any part of the screen;
* clear the CAT display;
* generate a test image via the bit-map;
* clear and reset the bit-map;
* take a "quick look" scan of the object, showing A-scans without grabbing data;
* plot any part of a line of data in any area of the screen.

When sending commands interactively to the GenRad, the user should be prompted for a line of one or more commands. According to Genrad command syntax, commands must be terminated by a semicolon. If the user forgets to type this terminating semicolon, he should be notified of this. Furthermore, if he types in an illegal character, the user should be made aware of this, and the illegal character (or the first one if more than one has been typed) should be pointed out. For example:
6.4 Functional Specification

The software can be divided into three major subsystems: data acquisition, data analysis, and image display and manipulation. Each of these parts is specified separately below, taking into account both the user's stated needs and the additional requirements given in the previous section:

Data Acquisition.

A. scan object with ultrasound transducer

1. control scanner according to the following parameters input by user: number of lines to acquire, number of steps between lines, and stepper delay time.

2. reposition scanner automatically to zero-point after scanning.

3. save acquisition parameters on disk.

4. provide for testing of scanner.

5. provide "quick look" scan.

B. transmission of data between GenRad and host computer

1. send commands and data over via GPIB.

2. provide capability of saving and restoring Genrad parameters.

3. provide capability of sending commands to GenRad interactively.

4. provide capability of acquiring a calibration spectrum for optional spectral subtraction.
Data Analysis.
A. transform raw data into image data.
B. allow dump of raw data and image data.
C. allow plotting of raw data and image data.

Image Display and Manipulation.
A. display any part of an image in any part of the video monitor.
B. provide for generation of test image via CAT and bit map.
C. provide following image manipulation functions:
   1. inversion.
   2. automatic stretch.
   3. manual stretch.
   4. linear stretch.
   5. histogram operations
      i. generation.
      ii. print-out.
      iii. display.
      iv. save to disk.
      v. restore from disk.
D. provide ability to clear frame buffers.

Overall
At any time, the user will have the capability of specifying which terminal (DEC VT100, TeleVideo 950 or Hazeltine 1520) is to be used currently with the software.
CHAPTER SEVEN
Software Preliminary Design

In this chapter, some general points are made concerning methodology used for this project. Then, the imaging-system software architecture is presented.

7.1 General Design Principles

Three universal means for designing a well-structured system with minimal complexity are: "1) partitioning the system into parts having identifiable and understandable boundaries, 2) representing the system as a hierarchy, and 3) maximizing the independence among the parts of the system" [MYER78].

7.1.1 Modularization

As Miller has noted [MILL56], the amount and complexity of information that the human mind is able to immediately understand can possibly be increased by recursively presenting the information as a sequence of chunks (groups of items that go together), such that each chunk contains no more than about seven items or chunks. To look at it another way (Figure 7.1), as the complexity of a problem (i.e., a number of parts that must be dealt with simultaneously) increases, the number of errors generated in solving the problem increases in a linear fashion until about $7+2$ entities are involved. For more enti-
ties, the number of errors generated rises sharply and non-linearly, partly because of interactions between elements, which must be dealt with in addition to the individual elements themselves [YOUR79].

Thus, each software task should be partitioned into separately named and addressable subtasks, or modules, which are as independent of each other as possible, where the number of subtasks or modules is \(7^{+}2\). This modularization is done repeatedly until the software is intellectually manageable. As for module size, a heuristic (a specific rule of thumb that usually works but is not guaranteed to do so) is that the optimal range is ten to one hundred statements [GILB80] [YOUR79].

7.1.2 Hierarchical Representation

Abstraction and information hiding are concepts integral to modularization. To abstract something is to disassociate its essential properties from its specific existence. Applying this concept to software design, levels of abstraction can be defined in order to provide a description of the software at certain levels of detail, ranging from the most general formulation appropriate to the most specific (generation of source code). So, to begin with, a very high-level description of the software is considered without reference to its detailed substructure. When this is well understood, the initially-irrelevant substructure is then considered at a lower level of abstraction, again disregarding underlying details irrelevant at this point. The process of stepwise refinement, of greater and greater elaboration of detail, is pursued until the level of source code is reached [SCHN81] [PRES82] [WIRT76] [DIJK68].
7.1.3 Module Independence

Information hiding is a principle which dictates that a module be constructed in such a way that as little as possible about its inner workings is known by the other modules making up the software system. The result is a set of independent modules that share with one another only that internal procedural or data structure information that is essential for them to function together. An important advantage of information hiding is that, since modules are more self-contained, they are more easily modified and thus more easily maintained [GILB80] [GOMA82] [PRES82] [PARN72].

7.2 Automatic File-naming

Before presenting the imaging-system software architecture (a hierarchical description of the overall software structure which identifies modules and the relationships between them), the mechanism for automatic file naming must be considered. Generating a file name automatically for a user spares him the otherwise necessary burden of multiply typing the file name during the course of a program, and of devising a file name each time. Each time that data is acquired, for example, at least three files are required: one each for (1) raw data, (2) acquisition parameters, and (3) GenRad parameters. In addition, if data is taken for image subtraction, another file is needed to store the calibration spectrum. Thus, automatic creation of file names makes use much easier.

A program can be related to the number of times it is used in a
given time frame (year and month). For the imaging system, the file naming convention is:

1. a file name consists of two parts: a six-character primary name field and a three-character extension. Each part is separated by a period.

2. the primary name is divided into three two-character sub-fields:
   i. the first (leftmost) two characters identify the current year.
   ii. the center two characters identify numerically the current month (01..12).
   iii. the rightmost two characters indicate the nth time the program has been used in the current month (00..99). (The user has indicated that the acquisition software would not be used more than 100 times per month.)

3. the extension indicates what kind of information the file contains:
   i. "raw" for raw data.
   ii. "cal" for calibration data.
   iii. "app" for acquisition program parameters.
   iv. "gpt" for GenRad parameter table.
   v. "img" for transformed (pixel) data.

Thus, if the program were to be run for the first time in May of 1983, the file name for raw data would be "830500.raw," the file name for the acquisition parameters would be "830500.app," and so on.

This scheme can be realized by using the calendar clock on the multifunction board, and by keeping available to the software a file containing the last used primary file name. The clock can be read by software to determine the current date, and the primary-file-name file can be read to determine the number of times the program has been used.
so far during the current month; this information is sufficient to
correctly form the primary file name when the program is executed.
The acquisition program must update the primary-file-name file. For
instance, suppose the last used primary file name is "830500," and
the program is run again in May of 1983. The program will read the
clock, ascertain that the current month and year have not changed
since the last time it was run, and form a new primary file name of
"830501." Unless the program is aborted or the user decides not to
save the acquired data, the primary-file-name file is updated to
contain "830501." However, the user must have the opportunity to
override this automatic file naming.

In addition, since the clock has to be initialized, and may need
to be changed, software clock routines must be provided to enable the
user to do this.

7.3 Software Architecture

Parnas [PARN69] suggested the use of state transition diagrams to
describe the user interface in an interactive computer system. This
approach can be applied in the preliminary design of a menu-driven
system to derive the highest level software architecture. That is, the
menus and the relationships between them are defined by state transi-
tion diagrams to the point at which routines are invoked to perform an
action chosen by the user. Each menu represented in the state transi-
tion diagram becomes a top-level module in the software structure.
This will be demonstrated below.

The state diagram notation used in this report has been modified
from standard notation in two ways. First, instead of using a separate arc to represent input to a node and a separate arc for output, a single double-pointed arc is used, labelled to indicate input and output. If an output direction is not labelled, it should be interpreted to mean that that direction is taken at the completion of processing in the node from which it emanates. Second, round nodes represent either a menu (menu module) or the final state (program termination), while square nodes indicate a software module which is activated by means of a menu selection (process module).

Figures 7.2 through 7.5 present the state diagrams describing the first two levels of the menu structure. The menus form a hierarchical system, with a root menu that calls other menus, which in turn either call further menus, or invoke a routine to fulfill a user-selected request. Each single character label accompanying an arc represents an actual key used to select a menu option. For example, in Figure 7.2, the arc connecting the root-menu node to the data-analysis-routines node has two labels: "2->" and "<-P." The label "2->" means that the root-menu has an option that the user selects by typing "2" on the keyboard. The "->" indicates that this selection enters the data-analysis-routines menu. Similarly, if the user is at the data-analysis-routines menu, and he types "P" in response to the menu prompt, the root-menu is invoked.

The software structure can be further defined at this point by decomposing the process modules into their components. The structure charts for two of the largest process modules are given in Figures 7.6 and 7.12. For the sake of brevity, this decomposition has been depicted to the third level at the deepest, showing only the major modules.
Errors generally becomes unmanageable at $7 \pm 2$.

Number of elements of the problem that must be dealt with simultaneously

Figure 7.1

Error curve for normal problem solving
Figure 7.2

State diagram for root menu
Figure 7.3

State diagram for data-acquisition-routines menu
Figure 7.4

State diagram for data-analysis-routines menu
Figure 7.5

State diagram for image-routines menu
Figure 7.6
Structure chart for acquire-data module
Figure 7.7

Structure chart for start-acquisition module
Figure 7.8
Structure chart for grab-data module
Figure 7.9

Structure chart for end-acquisition module
Figure 7.10

Structure chart for transform-data module
Figure 7.11

Structure chart for get-input module
Figure 7.12

Structure chart for transform-N-data-lines module
CHAPTER EIGHT
Software Detailed Design

The procedural description for only a few selected structural elements of special importance or interest will be presented. The notation used is a pseudo-code based on PASCAL and C.

8.1 Using different CRT terminals

Every CRT terminal (CRT) has certain screen functions which are controlled by code sequences that (except for a few standard ASCII codes such as BEL) are particular to that CRT. The three terminals supported by this project software, according to specifications, are the DEC VT100, the TeleVideo 950 (TLV950) and the Hazeltine 1520 (HZTN). One screen function implemented by all three terminals is "screen erase;" if the CRT is sent the proper code sequence by software, the whole CRT screen will be erased. For the VT100, the code sequence is "ESC [2J"; for the TLV950 it is "ESC *"; and for the Hazeltine it is "ESC FS." (ESC and FS are ASCII code mnemonics.)

Sending the proper code sequences can be taken care of by a CRT handler module (as is done for this imaging system), which contains a separate routine for each screen function required by the imaging software, such as erase functions and cursor movement functions. The routine that is called from the menus to set the CRT type presents a list of the CRT types supported and asks the user to select one, thereby setting a variable which is local to the CRT handler (known
globally to all routines within the handler but unknown by any software external to the handler). This variable determines which code sequence is sent to the terminal for each screen-function routine invoked:

LOCAL variable CRT-Type;

procedure Set-CRT-Type
  clear CRT;
  display CRT selection menu;
  repeat
    input selection;
    case selection of
      valid selection : set CRT-Type;
      default : error;
    endcase;
  until (no error)
end Set-CRT-Type

procedure erase(extent)
  case CRT-Type of
    VT100 : case extent of
       WHOLELINE : send ESC [2K ; (* to the CRT *)
       SCREEN : send ESC [2J ;
      endcase;
    TLV950 : case extent of
       WHOLELINE : send CR ESC t ;
       SCREEN : send ESC * ;
      endcase;
    HZTN : case extent of
       WHOLELINE : send CR ESC SI ;
       SCREEN : send ESC FS ;
      endcase;
  endcase;
end erase
procedure cursor(where)
    case CRT-Type of
        VT100 : case where of
            HOME : send ESC [H ;
            endcase ;

        TLV950 : case where of
            HOME : send RS ;
            endcase ;

        HZTN : case where of
            HOME : ESC DC2 ;
            endcase ;
        endcase ;
    end cursor

procedure ClearScreen
    erase(SCREEN) ;
    cursor(HOME) ;
end ClearScreen

procedure ResetScreen
    ClearScreen ;
    send 3 CR-LF's to CRT ; (* Carriage Return + Line Feed *)
end ResetScreen

procedure bell(max)
    for beep := 1 to max do
        send BEL to terminal ;
    endfor
end bell

8.2 Acquiring Data

As Figure 7.6 shows, the module for acquiring data has three main submodules: (1) "start acquisition," which prompts the user for parameters to control scanning and acquires a calibration spectrum if the user chooses to do so; (2) "grab data," which drives the scanner according to the user-input parameters to grab as many lines of data
...and (3) "end-acquisition," which either removes all files generated if the user does not want to keep data just acquired, or saves: (a) all files generated by acquisition; (b) the GenRad parameter table; and (c) the acquisition parameters.

The procedure for grabbing data is:

```plaintext
procedure GrabData
  open file for raw data ;
  ask user: "ready to acquire data ?" ;
  wait until user responds affirmatively ;
  for line := 1 to number-of-lines-to-acquire do
    echo line number to terminal
    grab a line of data ;
    write line of data to file ;
    move scanner 1 position according to parameters ;
  endfor
end GrabData
```

To grab a single line of data:

```plaintext
procedure GrabLine
  tell GenRad to start averaging data ;
  repeat
    transfer from GenRad number of averages done so far ;
    until (number averages done = NumToAvg) ;
    grab 400 averaged 16-bit values (binary) from Genrad ;
  end GrabLine
```

8.3 Transforming Data

The raw data as grabbed from the GenRad are all negative, 16-bit numbers. Although the numbers are in fixed point format (with one sign bit, eight integer bits and seven fraction bits), they can be treated as 15-bit integers with 1 sign-bit. These numbers must be transformed into 8-bit pixel data, which can be done line-by-line according to the following algorithm:
Here, an adjustable positive offset is added to the negative raw data to make it positive. The larger the offset, the greater the range of raw data that is made positive. If, after adding the offset, the resulting number is still negative, it is forced to equal 0, which is full black in the gray scale. Thus, the offset controls the brightness of the image.

After this operation, the data are shifted to the right N times, where N is adjustable. If, after being shifted, the number is still larger than eight bits (255), it is forced to equal 255, which is full white in the gray scale. This step has a bearing on the contrast of the image, because it affects how many raw numbers are brought within the pixel range of 0-255 and how many are forced to the upper limit of 255. As more numbers are brought into the pixel range, a greater probability exists that a wider area within the pixel range is represented, that is, that a higher contrast results.
0 to all its locations (*addr means "contents of address pointed to by addr"):

```plaintext
procedure ClearCAT
    for i := 1 to (256K / window_size) do
        select page i ;
        addr := CAT_image_window_address ;
        for j := 1 to window_size do
            *addr := 0 ;
            addr := addr + 1 ;
        endfor
    endfor
end ClearCAT
```

The CAT window is either 2K or 8K bytes in size, and the total CAT buffer size is 256K bytes. So, if the window size is 2K, for example, then the software must cycle through 128 pages, each 2K bytes large, stuffing the value 0 into each of the 2048 locations of the window memory space.

8.4.2 Identity Map

A useful routine is one which fills the video lookup table (LUT), or color map, with values such that the frame buffer contents are passed through to the video monitor unchanged [BASS81]:

```plaintext
procedure ramp
    enable color window ;
    select color map ;
    addr := CAT_map_window_address ;
    for i := 0 to (map_size - 1) do
        *addr := i ;
        addr := addr + 1 ;
    endfor
end ramp
```

The color window must first be enabled to tell the CAT that the following data is meant for the color map, not the frame buffer.
Although one of three color maps can be selected, since this imaging software deals only with gray scale images, only one color map need be used, the output of which is directed to the video monitor. The size of the color map is 256 locations (bytes). This routine puts 0 into the 0th map location, 1 into the 1st map location, 2 into the 2nd map location, and so on. As a result, each pixel value passing through the map emerges unchanged. For instance, a pixel value of 45 is routed through the 45th location of the map. The contents of this map location have been set by the above routine to 45, and this value is sent to the video monitor via the video DAC's.

8.4.3 Image Stretching

Once a histogram for an image has been generated, that image can be stretched in several ways. A "manual" stretch, in which the user specifies the lower and upper limits of the stretch, evenly spreads the gray scale between the two limits passed as parameters to the stretch routine:

```
procedure stretch(lower, upper)
    enable color window;
    select color map;
    addr := CAT_map_window_address;
    for i := 0 to 256 do
        if i <= lower              (* lower limit *)
        then
            *addr := 0;
        else
            if i >= upper
            then
                *addr := 255;
            else
                *addr := (256 * (i - lower)) / (upper - lower);
            endif
        endif
    addr := addr + 1;
```
Any value less than the lower limit is forced to 0, any value greater than the upper limit is clipped to the upper limit. The values between the limits are used to calculate a new LUT which stretches the values evenly between the gray scale bounds of 0 and 255; this spreading increases the contrast for the stretched region.

An "automatic" stretch routine calculates the limits between which the image will be stretched. To determine the lower limit, the histogram is integrated from the position corresponding to the gray scale value of 1 (the extreme values of 0 and 255 are avoided) until some arbitrary sum, determined experimentally to yield satisfactory results (100 works well), has been achieved. The upper limit is determined analogously, starting from the histogram position corresponding to the gray scale value of 255:

```
SumUpper := 0;
SumLower := 0;
UpperLimit := 0;
LowerLimit := 0;

i := 1;
repeat
  SumLower := SumLower + histogram[i];
  i := i + 1;
until (i > 254) OR (SumLower > 100);

LowerLimit := i - 1;

i := 254;
repeat
  SumUpper := SumUpper + histogram[i];
  i := i - 1;
until (i < 1) OR (SumUpper > 100);

UpperLimit := i + 1;
```
Once these limits have been calculated, the stretch algorithm given above is used.
CHAPTER NINE
Software Implementation

9.1 Implementation Language Considerations

9.1.1 Selection Criteria

The implementation language system used for an application has a major impact on the application software's effectiveness and on life-cycle software costs, and so should be selected with careful attention to management and technical considerations instead of such unacceptable criteria as hesitancy in learning a new language or arbitrarily imposed requirements. Valid general criteria include [ANDE82]:

* development time and cost
* life-cycle maintenance
* data representation
* control structures
* systems programming features
* support environment
* target CPU transportability
* extent of use
* learnability
* documentation
* time efficiency
* space efficiency
* assembly language linkage
* readability
* ROMable object code
* reentrancy and recursion
* compile-time efficiency

For this project, a modern, high-level language appropriate for interactive data acquisition was sought. Furthermore, the language had to be one supported under the CP/M operating system with an 8085 CPU as the target processor. Thus, many languages (assembly, Basic, Fortran,
Cobol and the like) were easily dismissed, and the choice was narrowed down to either Pascal or C.

9.1.2 Comparison of Pascal and C

Both C and Pascal are relatively small languages designed circa 1970. Pascal is intended to allow systematic and reliable program development by means of restrictions meant to enforce a disciplined structure. C is meant to be a relatively small, highly flexible general purpose language, on a low enough level to allow doing the sorts of things usually only possible in assembly language and efficient enough to remove the compulsion to use assembly language instead [KERN78].

A syntax comparison of most of the similar facilities of the two languages is given in Table 9.1. Figures 9.1 and 9.2 present a binary search routine written in C and Pascal, respectively.

Some major differences between C and Pascal, are discussed below [FEUE82] [MATE80]:

9.1.2.1 Type Philosophy

Pascal conforms almost perfectly to the definition of a strongly typed language, in which (1) every object belongs to exactly one type, and (2) type conversion takes place by converting a value from one type to another (type aliasing, where the representation of a value can be viewed as a different type, is not permitted). It is possible in Pascal to violate this concept of strong typing by such means as the variant record.
On the other hand, C is typed, although not strongly typed. It was consciously designed to allow easy manipulation of data type representations through unions, pointers, parameter passing and structure component names.

A great advantage of Pascal's strong typing is that the compiler can detect common errors which are often difficult for a programmer to pinpoint quickly, such as those caused by incompatible types. Runtime checking of argument types and array subscripts is provided. But, a serious disadvantage is that, since the dimension of an array is considered to be part of the type of the array, routines cannot be written which accept arrays of different sizes. Thus, a generic string routine or array sorting routine could not be implemented. In C, this can be done.

9.1.2.2 Pointers and Machine Access

Another significant difference is the handling of pointers, which are variables that contain the address of another variable. In Pascal, the application of pointers is restricted to objects in the heap, that is, variables of a specific type created dynamically by the procedure, new. Thus, given the declaration

```pascal
var p : integer ;
```

`new(p)` results in `p` being set to point to a newly allocated variable of type integer. Since the pointer is strictly bound to the type of the object it is declared to point to, full type checking is possible. A prime application of pointers in Pascal is in the definition and manipulation of recursive data structures such as linked lists and
However, the flexibility and power of pointers are so important in C that "its integration of pointers, arrays and address arithmetic is one of the major strengths of the language" [KERN78]. A pointer can point to any object of a particular type (including functions). Also, the arithmetic operations of comparison, addition and subtraction on pointers are valid; this arithmetic is performed in units of the storage size of the pointer's base type. Therefore, to stuff the value contained in "num" into location 500 and location 501 in a byte-addressable memory, the following code would suffice (the ++ increment operator is a more concise and efficient way of adding one to a variable than "x = x + 1"):

```c
BYTE *p; /* declare p to be a pointer to a byte. */
p = 500; /* p now points to location 500. */
*p = num; /* the contents of loc 500 are set to num. */
*p++ = num; /* p is incremented by 1 so that it points to */
/* location 501, whose contents are then set */
/* to num. */
```

So, unlike in Pascal, the underlying machine is much more accessible, but at the expense of more danger in programming.

Moreover, C has bit-wise logical operators to enable manipulation of bits within a word:

```
& AND
| OR
^ XOR
<< left shift
>> right shift
~ one's complement (unary).
```
9.1.2.3 Variables and Information Hiding

Although C does not allow procedure and function nesting as in Pascal, it encourages the localization of variables by permitting them to be declared at the start of any block, such that their scope is that block. Furthermore, C provides a storage class called "static," either internal or external. Internal static variables are local to a particular block and remain in existence for the duration of the program rather than coming and going each time the block is activated. Pascal must resort to global variables or parameters to achieve this permanence; global variables increase unreliability and unreadability. External static variables and functions are known only within the remainder of the physical source file in which they are declared. This enables a way to implement information hiding. A collection of routines can be put into a separate physical file such that only those that are to be used by the outside world are seen by the outside world, while data objects private to those routines (and other functions which only need be used within the scope of the file) are inaccessible to the outside.

9.1.2.4 I/O

Even though I/O routines are not specified in the C language, they are provided in a standard library. These include routines to read and write data in blocks of arbitrary size, and to read and write formatted text. The format specified in an I/O routine references strings of arbitrary form, and since it is itself a string, can be generated dynamically and passed as a parameter.
Pascal's built-in I/O routines are far more clumsy and inflexible; its capability for formatted output is confined to specifying the field width for printed values, and it has no mechanism for formatted input. Also, unlike C, Pascal provides no standard, straightforward way to handle non-printing characters, such as control-C (ASCII ETX, i.e., 3), because the character set recognized by Pascal is implementation dependent, and so these control characters may be excluded from that character set.

The essence of these points is that C is far more suitable for interactive programs.

### 9.1.3 Compiler and Support Environment

For the above reasons, and others, C was chosen as the implementation language. Specifically, Whitesmith's cross-compiler running under RT-11 on an LSI-11/23 was used. Of all the versions of C investigated, Whitesmith's not only has C for the 8080 and 8085 CPU's and CP/M operating system (which was used for this imaging system), but for several other CPU's and operating systems, making it the most portable. It also comes with a good library, and generates fairly fast code.

### 9.2 Packaging

After the conceptual, modular structure of a program is determined, the software must be assembled into load units (distinct physical units) for execution on a specific machine, with attention to execution speed and memory constraints; the load units can be overlays
or separate programs, for example. Effective packaging can be achieved by using procedural analysis, which consists of a set of criteria to determine which modules should be in the same load unit. These criteria include degree of repetition, frequency of access and time interval between calls. Sometimes, the module type or the structure itself of the system may need to be modified in order to satisfy execution speed requirements. For instance, if the overhead of the subroutine-calling mechanism is unacceptable, the subroutine module can be put in-line in the form of a macro or as lexically included code [YOUR79].

Instead of one program as designed, the 5500 lines or so of software written for this project had to be packaged into several separate programs, with some rearrangement of modules. This was necessary because the size of the executable code was too large to fit into the 64K byte memory space available. Overlays were not an option because the Whitesmith's C linker does not support overlays, nor was another linker available that did. Also, some display routines were made into macros in order to make image formation and manipulation faster.

The routines were packaged into physical files in as logical a way as possible. Thus, for example, all the functions dealing with the GPIB form a separate GPIB handler file, with information-hiding implemented by means of static variables and functions.

To give the flavor of an entire file of modules written in C, the GPIB handler is presented in Appendix II.
<table>
<thead>
<tr>
<th>Facility</th>
<th>C</th>
<th>PASCAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declarations</td>
<td></td>
<td></td>
</tr>
<tr>
<td>constant</td>
<td>#define name = value</td>
<td>const name = value</td>
</tr>
<tr>
<td>type</td>
<td>typedef type name;</td>
<td>type name = type</td>
</tr>
<tr>
<td>variable</td>
<td>[storage-class] type names;</td>
<td>var variables: type</td>
</tr>
<tr>
<td>array</td>
<td>[storage-class] element-type name [size]…;</td>
<td>var name: array[index-type,…] of element-type</td>
</tr>
<tr>
<td>record</td>
<td>[storage-class] struct name (fields);</td>
<td>var name: record</td>
</tr>
<tr>
<td></td>
<td></td>
<td>fixed-part; variant-part end</td>
</tr>
<tr>
<td>pointer</td>
<td>[storage-class] type *name;</td>
<td>var name: type</td>
</tr>
<tr>
<td>routine</td>
<td>[storage-class] result-type name</td>
<td>function name</td>
</tr>
<tr>
<td></td>
<td>([formal-parameters]);</td>
<td>[(formal-parameters)]: result-type;</td>
</tr>
<tr>
<td></td>
<td>[parameter-declarations]</td>
<td>local-declarations;</td>
</tr>
<tr>
<td></td>
<td>compound-statement</td>
<td>compound-statement</td>
</tr>
<tr>
<td></td>
<td></td>
<td>procedure name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[(formal-parameters)];</td>
</tr>
<tr>
<td></td>
<td></td>
<td>local-declarations;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>compound-statement</td>
</tr>
<tr>
<td>Referencing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>array element</td>
<td>array-name [i]…[i]</td>
<td>array-name [i,…,i]</td>
</tr>
<tr>
<td>record field</td>
<td>record-name.field-name</td>
<td>record-name.field-name</td>
</tr>
<tr>
<td>pointer object</td>
<td>*p</td>
<td>p†</td>
</tr>
<tr>
<td>Statements</td>
<td></td>
<td></td>
</tr>
<tr>
<td>assignment</td>
<td>variable = expression;</td>
<td>variable := expression</td>
</tr>
<tr>
<td>compound</td>
<td>(variable-declarations statements)</td>
<td>begin statements end</td>
</tr>
<tr>
<td>conditional</td>
<td>if(expression) statement;</td>
<td>if Boolean then statement;</td>
</tr>
<tr>
<td></td>
<td>else statements</td>
<td>else statements</td>
</tr>
<tr>
<td>selection</td>
<td>switch(expression) (</td>
<td>case expression of</td>
</tr>
<tr>
<td></td>
<td>labeled-statements</td>
<td>labeled-statements</td>
</tr>
<tr>
<td></td>
<td>)</td>
<td>end</td>
</tr>
<tr>
<td>while loop</td>
<td>while(expression) statement</td>
<td>while Boolean do statement</td>
</tr>
<tr>
<td>repeat loop</td>
<td>do statement while (expression)</td>
<td>repeat statements until Boolean</td>
</tr>
<tr>
<td>for loop</td>
<td>for(initialization; expression; reinitialization)</td>
<td>for variable := initial-value</td>
</tr>
<tr>
<td></td>
<td>statement</td>
<td>to final-value do statement</td>
</tr>
<tr>
<td></td>
<td></td>
<td>for variable := initial-value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>downto final-value do statement</td>
</tr>
</tbody>
</table>
```c
#include <stdlib.h>

#define FAILURE (-1)

typedef string *char ;
typedef struct

    { int key ;
    string value ;

    } item

;

/* either return index of item A with key k, */
/* or return FAILURE if k is not in A. */

BinarySearch(A, n, k)
    item A[] ;
    int n, k ;
{
    int low = 0, high = n -1, mid ;

    while(low <= high)
        {
            mid = (low + high) / 2 ;
            if(k < A[mid].key)
                high = mid - 1 ;
            else
                if(k > A[mid].key)
                    low = mid + 1 ;
                else
                    return(mid) ;
            } return(FAILURE) ;
        }

Figure 9.1

Binary search routine in C
const max = 100;  \{max size of the array\}
    failure = -1;

type index = 1..max;
    string = packed array[1..20] of char;
    item = record
        key: integer;
        value: string;
    end;
    data = array[1..max] of item;

{ either return index of item in A with key k, }
{ or return failure if k is not in A. }

function BinarySearch (var A : data; n : index; k : integer) : integer;
    var low, mid, high : index;
    result : integer;
begin
    low := 1;
    high := n;
    result := failure;
    while \{low <= high\} and \{result = failure\} do
        begin
            mid := \{high + low\} div 2;
            if k < A[mid].key
                then high := mid - 1
                else
                    if k > A[mid].key
                        then low := mid + 1
                        else result := mid
            BinarySearch := result;
        end;
end;

Figure 9.2

Binary search routine in Pascal
References


[DIGI79] CAT-100C Technical Specifications. Digital Graphic 

[DIJK68] Dijkstra, E.W. The Structure of the THE-Multiprogramming 

[ERIK74] Erikson, K.R.; Fry, F.J.; and Jones, J.P. 
Ultrasound in Medicine -- A Review. IEEE Transactions on 
Sonics and Ultrasonics 21/3 (July 1974).

[EVAN78] Evans, T.C.; Green, P.S.; and Greenleaf, J.F. Development 
of High-Resolution Ultrasonic Imaging Techniques for 
Detection and Clinical Assessment of Cardiovascular 
Disease. National Institute of Health Report, No. NIH-

[FAIR81] Microprocessor-GPIB Interfacing with the 96LS488. Fairchild 
camera and Instrument Corp. Mountain View, California 
(June, 1981).

Programming Languages C and PASCAL. ACM Computing Surveys, 

[FINK75] Finkel, J. Computer-Aided Experimentation: Interfacing to 

[FOLE82] Foley, J.D., and Van Dam, A. Fundamentals of Interactive 

Reading: Addison-Wesley, 1981.

GenRad, Inc. Santa Clara, California (1980).

[GILB80] Gilbert, P. Notes on Program Design Techniques. Reseda: 
Ridgeview, 1980.

Park, California (1981).

[GOMA82] Gomaa, H.; Lui, J.; and Woo, P. The Software Engineering 
of a Microcomputer Application System. Software Practice 

Reading: Addison-Wesley, 1977.

Suarez, J.R.; Marich, K.W.; Evans, T.C.; Sandok, A.; and 
Greenleaf, J.F. A Real-Time Ultrasonic Imaging System for 


The GPIB (General Purpose Interface Bus)

The GPIB, designed by Hewlett-Packard and adopted as a standard by IEEE in 1975, is fully described in IEEE STD 488-1978, Standard Digital Interface for Programmable Instrumentation [IEEE78]. It sets forth electrical specifications (voltage and current levels for transfer of signals), mechanical specifications (cables, connectors, etc.) and functional specifications (definition of all signal lines, the protocol and timing relationships for using the lines, and the repertoire of messages that may be exchanged) for a byte-serial, bit-parallel, bidirectional interface system designed to allow up to fifteen devices within a localized area to communicate with each other over a common bus. This provides the capability for interfacing a number of devices by means of only one interface card instead of requiring one interface card per device. As a result, rather than taking up fifteen slots in the computer mainframe for fifteen different devices, just one slot is occupied. Another reason it is advantageous to reduce the number of interface cards in a system is that each card introduces additional power consumption and dissipation. An example of a GPIB system connected to an S-100 bus is diagrammed in Figure A.1.

There are several defined constraints for this interface system:

* no more than fifteen devices can be interconnected by a single contiguous bus;
* maximum cable length is 20 meters, or 2 meters times the
  number of devices, whichever is less;

* the data rate through any signal line must be no greater
  than 1 Mbyte per second; and

* all data transfer is to be digital.

Two classes of information (messages) are carried on the bus:
(i) interface messages, used for bus management, and (ii) device-
dependent messages, which are passed through the bus for programming
and data transfer between the devices on the bus.

The functionality of the devices on the bus is divided into three
types: that of listener, talker, and controller. A listener can
receive data from other devices. A talker can send data to other
devices. A controller manages communication over the GPIB; for exam-
ple, it determines which device can talk or listen, initializes the
bus, responds to a request for service from a device, and so on. A
given device may have any or all of these capabilities. If more than
one controller is connected to the bus, one is designated as the
system controller. In order to enable it to be addressed as a talker
or listener, each device is assigned a unique five-bit address, which
can be set by an external DIP switch on the device, or via software.

The bus structure is comprised of 24 lines (as shown in the
connector pin assignments in Figure A.2), of which 16 are signal lines
organized into three sets (Figure A.3):

* 8 data input/output lines

* 3 data transfer control (handshake) lines

* 5 general interface management lines

The data lines, D101–D108, carry all interface messages and
device-dependent messages.

The handshake lines, NRFD, DAV and NDAC, are used to coordinate the asynchronous transfer of information on the data lines between devices generating data (sources) and devices receiving that data (acceptors), according to the logic illustrated in Figure A.4. This process achieves three goals. First, more than one device is able to accept data at a time. Second, since new data is only sent when each listener has received the last byte and is ready for the next, devices with different operating speeds can be interconnected, and the data transfer rate is adjusted to the slowest device. Third, data can be transferred asynchronously. Although any number of devices may be listeners (which control NRFD and NDAC), only one may be a talker (which controls DAV). The NRFD (Not Ready For Data) line is true when one or more listeners are not ready for data and false when all listeners are ready for data. The DAV (Data Valid) line indicates that the data on the data lines is valid and available for use on the bus. The NDAC (Not Data Accepted) line is true when one or more listeners have not accepted the data.

The five bus management lines are ATN, IFC, REN, SRQ and EOI. ATN and IFC must be monitored by all devices on the bus; the others need not be. Their functions are:

**ATN** (Attention). When set true, devices interpret information on the data lines as interface messages; otherwise, data on the data lines are considered to be device-dependent messages.

**IFC** (Interface Clear). When asserted (for a minimum of 100 microseconds, the bus is initialized by placing all devices in a known quiescent state.)
REN (Remote Enable). When asserted, devices can be programmed either by their local controls, such as front panel switches, or by information sent over the bus.

SRQ (Service Request). Enables a device to asynchronously request service from the controller.

EOI (End or Identify). When ATN is true, the controller may send the Identify message by asserting EOI to initiate a parallel poll of all devices with parallel poll capability. If ATN is false, an addressed talker may send the END message to indicate the end of its data transmission by asserting EOI at the same time it places the last data byte on the data lines.

Bus operation is conceptually partitioned into device functions and interface functions. Device functions are the unique capabilities of a device defined by the device designer. Interface functions are bus management functions which establish the environment in which device functions can be accomplished. Several interface functions are defined in the standard according to a specific protocol in terms of one or more groups of interconnected, mutually exclusive states. For example, the Source Handshake (SH) interface function controls the initiation of, terminations of, and transfer of a multiline message byte.

The two general categories of messages are uniline and multiline messages. Uniline messages are those that are sent over a single line from the five bus management lines; more than one uniline message can be sent concurrently. Multiline messages are sent over the eight data lines. Unlike uniline messages, only one multiline message at a time can be sent. Multiline interface messages are subdivided into those that are effective whether or not a listener has been designated (universal commands), and those that are only recognized when a device
has been addressed as a listener (addressed commands). An example of a universal command is telling a device to be a listener, called MLA (My Listen Address). To accomplish this, the controller places the device's listen address on the data lines and then performs the Source Handshake interface function. As soon as ATN is asserted, all devices hear the controller's commands. In this case, the information on the data lines is interpreted as a listen address. When a device recognizes its listen address on the bus, it enters the Listener Addressed State (LADS), and will become an active listener as soon as the controller releases the ATN line [CARR81] [FAIR81] [NATI82] [TEXA79] [LOUG75].
Figure A.1

An example of the GPIB connected to an S-100 system
Figure A.2

GPIB pin assignments
Figure A.3

GPIB signal lines
Figure A.4

GPIB handshake protocol
Appendix B

GPIB Handler Source Listing

/*

File name : GPIB.C
Author : George V. Krestyn
Creation date : 01-Oct-81
Last update : 14-Feb-83
Description : GPIB handler
Functions :

  waitbo[]
  waitbi[]
  ClearGPIB[]
  OutGPIB[datum]
  GRRead[dev, array, len]
  GBRead[dev, array, len]
  GCWrite[dev, array, len]
  GDWrite[dev, array, len]

Parameters:
  dev  IEEE 488 device number: 0 < dev < 31
  array Name of string [character array] whose
           address is passed.
  len  Length of string — tested on received
       strings.

Documentation :

IEEE488 bus code for DYLON Model 1020A S-100/488 interface board, using
the Texas Instruments TMS 9914 GPIB Adapter chip.

Pertinent documentation :
1. Dylon Installation Manual for Model 1020A S-100/488 Interface Board,
   referred to as Dylon in this program's comments.
2. Texas Instruments TMS 9914 GPIB Adapter Preliminary Data Manual,
   referred to as TMS in this program's comments.
3. IEEE STD 488-1975, referred to as IEEE in this program's comments,
4. The C Programming Language by Kernighan and Ritchie, referred to as
   K&R in this program's comments.

*/
#include <std.h>

/\*-----------------------------------------------\*/

#define base 0x80 /* base address */
#define insts0 base+0 /* int status 0 */
#define insts1 base+1 /* int status 1 */
#define addsts base+2 /* address status */
#define bussts base+3 /* bus status */
#define addswt base+4 /* address switch 1 */
#define cmdpas base+6 /* cmd pass thrgh */
#define datin base+7 /* data in */

#define intmk0 base+0 /* int mask 0 */
#define intmk1 base+1 /* int mask 1 */
#define auxcmd base+3 /* auxiliary cmd */
#define address base+4 /* address reg */
#define serpol base+5 /* serial poll */
#define parpol base+6 /* parallel poll */
#define datout base+7 /* data out */

/\*-----------------------------------------------\*/

#define swrstc 0x00 /* software reset clear */
#define swrstc 0x80 /* software reset set */
#define dacrc 0x01 /* release DAC holdoff clear */
#define dacrc 0x81 /* release DAC holdoff set */
#define rhdf 0x02 /* release RFD holdoff */
#define hdfac 0x03 /* holdoff on all data clear */
#define hdfac 0x83 /* holdoff on all data set */
#define hdfec 0x04 /* holdoff on end clear */
#define hdfec 0x84 /* holdoff on end set */
#define nbaf 0x05 /* set new byte available false */
#define fgetc 0x06 /* force group execute trigger clear */
#define fgetc 0x86 /* force group execute trigger set */
#define rtlc 0x07 /* return to Local clear */
#define rtlc 0x87 /* return to Local set */
#define feoi 0x08 /* force end or identify */
#define lonc 0x09 /* listen only clear */
#define lonc 0x89 /* listen only set */
```
#define tone 0x0A /* talk only [ clear ] */
#define tons 0x8A /* talk only [ set ] */
#define gts 0x0B /* go to standby */
#define tcs 0x9C /* take control asynchronously */
#define tcs 0x0D /* take control synchronously */
#define rppc 0x0E /* request parallel poll [ clear ] */
#define rpps 0x0E /* request parallel poll [ set ] */
#define sicc 0x0F /* send interface clear [ clear ] */
#define siccs 0x0F /* send interface clear [ set ] */
#define srec 0x10 /* send remote enable [ clear ] */
#define sres 0x10 /* send remote enable [ set ] */
#define rqc 0x11 /* request control */
#define rlc 0x12 /* release control */
#define daic 0x13 /* disable all interrupts [ clear ] */
#define daics 0x13 /* disable all interrupts [ set ] */
#define stdlc 0x14 /* set T1 delay [ clear ] */
#define stdls 0x15 /* set T1 delay [ set ] */
#define shdwc 0x16 /* shadow handshake [ clear ] */
#define shdws 0x16 /* shadow handshake [ set ] */

define del 0x14 /* device clear */
define gtl 0x01 /* go to local */
define sdc 0x04 /* selected device clear */
define unl 0x3f /* unlisten */
define unt 0x5f /* untalk */
```

/* Universal Interface Messages [a subset] */
I*

waitbo() --------------------------------------------------

Wait until last output done.

With the exception of INTO and INT1, each interrupt bit in the interrupt
status register is set when the corresponding event occurs, regardless of the
state of the respective mask bit. The action of reading each interrupt status
register also clears it.

BO, which indicates that the TMS 9914 is ready to accept for output the next
(or first) data byte, is the name of bit 4 of Interrupt Status Register 0
corresponding to position D3 of the MPU bus in the table on page 5 of TMS.
MAC is the name of bit 0, the LSB. See page 5 & 6 of TMS.

/*

#define waitbo() while((in(intstO) & 0x10) is 0) {} ;

*/

I*

waitbi() --------------------------------------------------

Wait until last input done.

BI, which indicates that a data byte has been received, is the name of bit 5
of Interrupt Status Register 0 corresponding to position D2 of the MPU bus in
the table on page 5 of TMS. MAC is the name of bit 0, the LSB.

BYTE waitbi()
{
    BYTE a ;

    while(((a = in(intstO)) & 0x20) is 0) {} ;
    return(a) ;
}

*/

ClearGPIB() --------------------------------------------------

ClearGPIB()
{
    FAST UCOUNT i ;

    out[auxcmd, swrste] ; /* do a software reset */
    out[auxcmd, swrsto] ; /* stop doing the software reset */
    out[auxcmd, sica] ; /* send an IFC */

    for(i = 0; i < 100; i++) ; /* must wait minimum of 100 microseconds. Page 17 TMS */
    out[auxcmd, sicc] ; /* stop sending IFC */
    out[auxcmd, sres] ; /* send remote enable */
    OutGPIB(dcl) ;
}

I*
OutSPIB(datum)
BYTE datum;
{
/*
* If : ATN is asserted, and if bit 1 of addsts is set, then TADS [talker
addressed state] is indicated;
else : if ATN is not asserted, and if bit 1 of addsts is set, then TACS
[talker active state] is indicated.
See pp. 8 & 19 TMS.
Note : if bit 5 of addsts is set, ATN is asserted.
*/

if((in(addsts) & Ox22) is 0) /* ATN is not asserted */
{
    out(auxcmd,tcs); /* Assert ATN by taking control synchronously */
in(datin);
}
else /* ATN is asserted */
{
    if((in(addsts) & Ox20) is 0) /* bit 5 -> ATN */
    {
        waitbo(); /* wait until bus is clear of data */
        out(auxcmd,tca); /* Assert ATN by TCA */
    }
}
out(auxcmd,tonc);
out(auxcmd,lonc);
waitbo(); /* wait until bus is clear */
out[datout, datum]; /* and send datum */
}
GARead(dev, array, Len) /* Read ASCII data — LF or CR is end-of-line */

BYTE dev;
FAST BYTE *array;
FAST UCOUNT Len;
{
    FAST UCOUNT i;
    BYTE status, waitbi();
    BYTE inchar;

    OutGPIB('!_'); /* UNTalk command
    OutGPIB(dev | 0x40); /* tell talker to talk

    if((in[addsts] & 0x20) != 0) /* BIT 5 -> ATN */
    {
        waitbo(); /* wait for bus to clear
        out(auxcmd, lone); /* listen-only set
        out(auxcmd, gts); /* go-to-standby set — ATN released
    }

    for(i = 0; i < len; i++)
    {
        status = waitbi(); /* wait for byte to come in
        inchar = in(datin) & 0x7F; /* input byte & mask out parity bit

        /* quit if current input byte is an EOS character [End-Of-String is
        /* either LF or CR]; else, stuff into array:

        if((inchar is LF) || (inchar is CR))
            return(i); /* quit inputting data & return # bytes read */
        else
            *array++ = inchar; /* stuff current input byte into array */

        if((status & 0x08) != 0) /* if END [bit 3] is true, it's
            return(i+1); /* time to quit and return # bytes read */
    }

    return(i); /* return # bytes read */
I*==================
GBRead(dev, array, Len)
BYTE dev ;
FAST BYTE *array ;
FAST UCOUNT Len ;
{
    FAST UCOUNT i ;
    BYTE status, waitbi[] ;
    OutGPIB('T' ) ;
    OutGPIB(dev | 0x40) ;
    if((in(addsts) & 0x20) I= 0)
    {
        waitbi[] ;
        out(auxcmd, lons) ;
        out(auxcmd, gts) ;
    }
    for (i = 0; i < Len; i++)
    {
        status = waitbi[] ;
        *array++ = in(datin) ;
        if((status & 0x0B) I= 0)
        {
            return[++i] ;
        }
    }
    return[i] ;
}
void GCWrite(dev, array, len)

BYTE dev;
BYTE *array;
UCOUNT Len;

[FAST UCOUNT i;

OutGPIB('?'); /* UNListen all devices */
OutGPIB(dev | Ox20); /* tell listener to listen */

/* If ; ATN is asserted, and if bit 1 of addsts is set, then TADS [talker addressed state] is indicated;
   else ; if ATN is not asserted, and if bit 1 of addsts is set, then TACS [talker active state] is indicated.
   If ; bit 5 of addsts is set, ATN is asserted.
See pp.8 & 19 TMS.
*/

if((in[addsts] & Ox22) is 0)
{
   out[auxcmd,tcs]; /* Assert ATN by taking control synchronously */
in[datin];
}
else
{
   if[(in[addsts] & Ox20) is 0] /* bit 5 -> ATN */
   {
      waitbo(); /* wait until bus is clear of data */
      out[auxcmd,tca]; /* Assert ATN by TCA */
   }
}
out[auxcmd,tonc];
out[auxcmd,lonc];

for[1 = 0; i < len; i++]
{
   waitbo(); /* wait for last data byte to clear bus */
   out[datout, *array++]; /* send byte out */
}
GDWrite(dev, array, len) /* Write data */

BYTE dev;
BYTE *array;
UCOUNT Len;

{ 
FAST UCOUNT i;

OutGPIB['?']; /* UNListen all devices */
OutGPIB[dev | 0x20]; /* tell listener to listen */

if((in[addsts] & 0x20) != 0) /* bit 5 -> ATN */
  
    out[auxcmd, tons]; /* ATN is asserted */
    out[auxcmd, gts ]; /* talk only set */
  
if(i is (Len - 1)) out[auxcmd, fec1]; /* indicate EOI */
out[datout, *array++]; /* send byte out */
}

/* Write data */
/* UNListen all devices */
/* tell listener to listen */
/* bit 5 -> ATN */
/* ATN is asserted */
/* talk only set */
/* go-to-standby set - ATN released */
/* wait for last data byte to clear bus */
/* indicate EOI */
/* send byte out */