A COMPUTERIZED ULTRASOUND EXPOSIMETRY SYSTEM

BY

FELICE CHU

B.S., University of Illinois, 1990
B.S., University of Illinois, 1990

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1991

Urbana, Illinois

DEDICATION

This thesis is dedicated to my parents, Yung Fan and Teh Hwa Chu, whose love and moral support have given me the strength to strive for my goals.

ACKNOWLEDGMENTS

I am indebted to my research advisor, Professor William D. O'Brien, Jr., for his help and encouragement during this project and for his faith and patience in me during my college career. Special thanks are offered to Ilmar Hein for his many hours of technical advice and for being by my side as a great friend. My appreciation is extended to Nadine Smith for her advice during the software development. Thanks are given to Billy McNeill for designing and building the water tank. I thank Wanda Elliott and Robert Cicone for their administrative assistance. My deepest appreciation goes to Michael Lightstone for his support and encouragement. Also, thanks to those who cheered me on: Beth Neundorfer, Melissa Tessendorf, Shu-Wen Chen, Zhian Xu, Jian-Biao Zheng, Kathi Annous, Eric Chen, Scott Ellis, Tim Peck, and many others who are not mentioned but not forgotten.

This work was supported by a subcontract from the University of California, Davis, through a grant from the National Institutes of Health, grant NIH HD25528, for research being conducted by Dr. Alice Tarantal.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Ultrasound is a commonly used imaging tool in today's medicine and can be found in most medical specialties. Moreover, at least 50% of all fetuses are exposed to one ultrasound scan [1]. Yet, how do we know that diagnostic ultrasound exposure is safe and does not have long term biological effects? Is there a limit to the exposure time or strength? An understanding of the output of diagnostic ultrasound is needed before biological effects and safety can be evaluated. But, how is the output energy measured? What is the method of expressing the energy? What does it mean??

Many people are concerned about the safety of ultrasound and if there are possibly harmful effects under different conditions. At the California Primate Research Center, University of California, Davis, Dr. Alice Tarantal is researching in the field of ultrasonic bioeffect and tarantology. Her studies focus on prenatal exposure to ultrasound in nonhuman primates, specifically, Macaque monkeys. The fetuses are exposed for extended periods of time during specified days of the fetal development and are monitored prenatally and postnatally. These studies 1) evaluate the safety of diagnostic ultrasound (scan

mode) and pulsed Doppler exposure in an *in vivo* system similar to humans and 2) compare changes in animals exposed prenatally to diagnostic ultrasound and pulsed Doppler. To pursue the mechanisms for and enhance the understanding of her results, the output of her clinical device needs to be calibrated prior to each experiment. This would allow any possible biological effects to be linked to a quantifiable measure of ultrasound. In other words, if there are biological effects from diagnostic ultrasound, at what level of ultrasonic output are these biological effects occurring. In order to calibrate her diagnostic equipment frequently, an exposimetry system was commissioned to be designed and built specifically to meet her needs.

This thesis is the description of the exposimetry system that was built for the California Primate Research Center. Chapter 2 defines the acoustical terms and presents the measurement theory that is implemented in the system. Chapter 3 describes the hardware components. Chapter 4 describes the software developed to run the hardware and make the intensity calculations. Chapter 5 introduces possible improvements on the system due to the uncertainties of the system described. Chapter 6 concludes and summarizes the exposimetry system which was delivered.

CHAPTER 2

MEASUREMENT THEORY

Ultrasound is similar to the sound in the audible frequency range except that it is at a higher frequency so that humans cannot hear it. This sound is projected into the body where it bounces off at different interfaces. These ultrasonic echoes are then used to create an image of the body; this is called diagnostic ultrasound.

There are different modes in which ultrasound can operate, continuous wave (CW) mode and pulsed mode. When a transducer is driven electrically, it creates a mechanical motion back and forth, thus vibrating and creating a wave. If this wave is continuous without any interruptions, the transducer is said to be operating in continuous wave mode. Figure 1 is representative of a continuous wave. The frequency of this wave is the number of cycles per second, and the period is the reciprocal of the frequency.

When the transducer is driven in bursts of identical pulses, it is operating in pulsed mode. Each pulse is only a few cycles long. In diagnostic ultrasound, these pulses tend to have different amplitudes for each half cycle. A variation of the pulsed mode is a Doppler pulse where the half cycles of the pulse

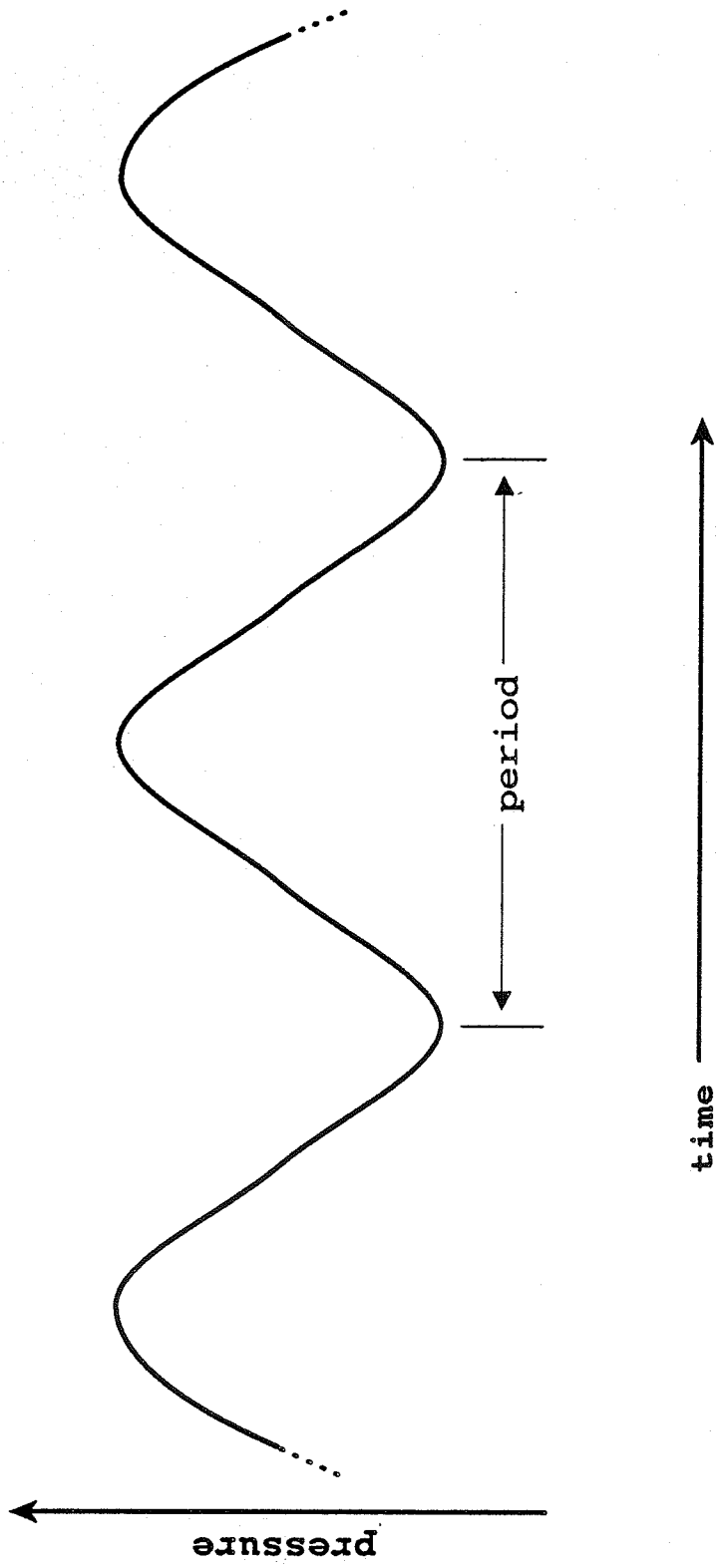


Figure 1 - Continuous ultrasonic wave.

have similar amplitudes and are a few cycles longer than in diagnostic ultrasound. Figure 2 shows the differences between the two pulses. As in continuous mode, the transducer frequency is the frequency at which that the transducer is driven and the frequency within the pulse. In pulse mode, additional parameters are used to define the signal. The pulse duration (pd) is the time from the beginning of the pulse to the end of the pulse. The pulse repetition frequency (prf) is the number of pulses in one second. The pulse repetition period (prp) is the time between consecutive pulses, as measured at the same point in the pulse, and is also the reciprocal of the pulse repetition frequency. To represent the fraction of time at which the sound is actually on during one complete cycle, the duty cycle is the ratio of the pulse duration to the pulse repetition period.

Both continuous mode and pulsed mode can have focused and unfocused sound beams. As represented in Figure 3a), the unfocused sound beam has a uniform energy output at different distances from the transducer, given no attenuation by the medium. The focused sound beam, see Figure 3b), concentrates the energy at a specified distance from the transducer.

Although total power is an important acoustical parameter, more often the specifications of acoustical output of diagnostic ultrasound are in terms of intensity parameters derived from acoustical pressure measured by a miniature hydrophone. Intensity, or sometimes called power density, is the spatial concentration of power. However, measuring the intensity can be complicated. There are two components of intensity, spatial pattern and

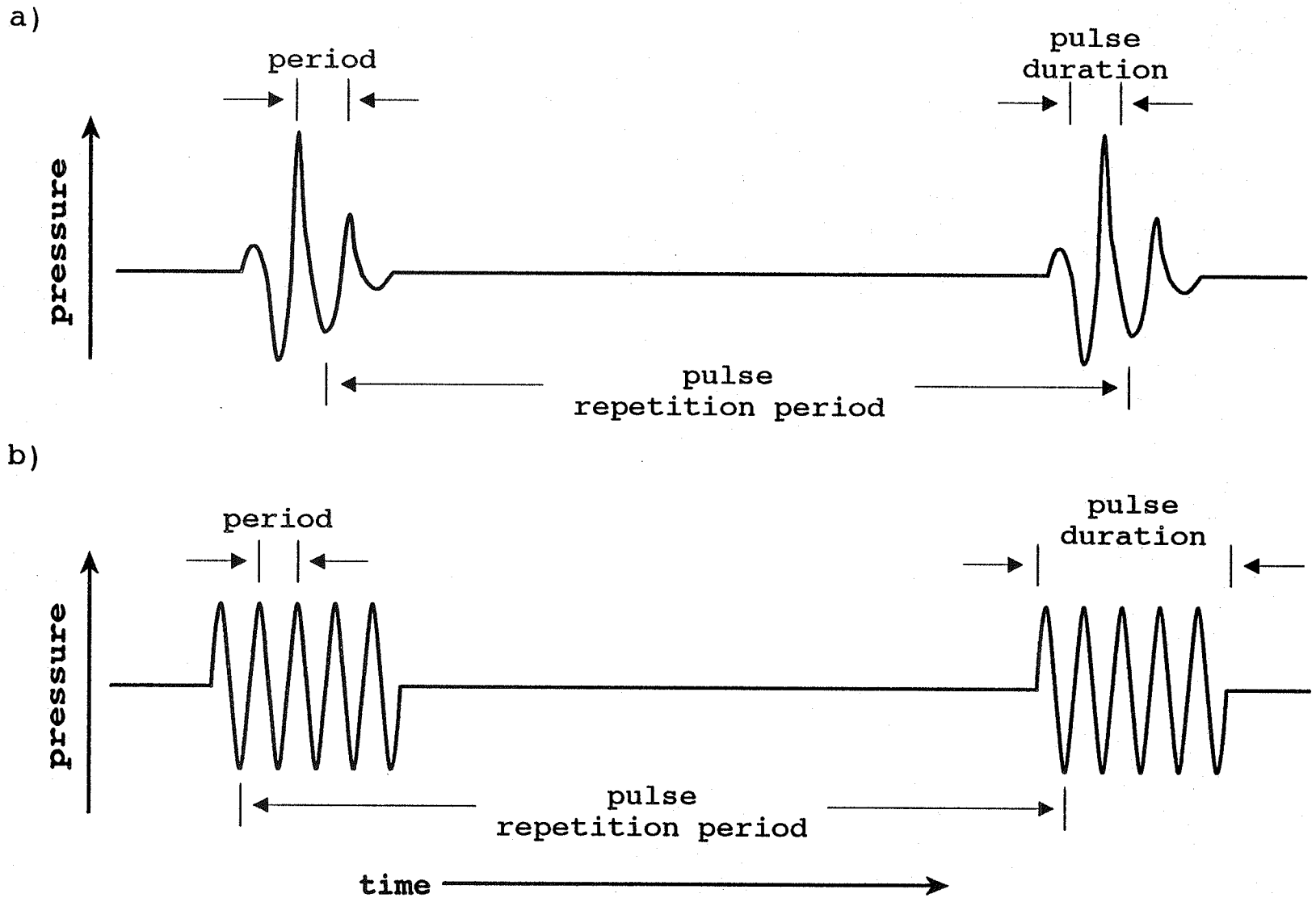


Figure 2 - a) Ultrasonic scan pulse. b) Ultrasonic Doppler pulse.

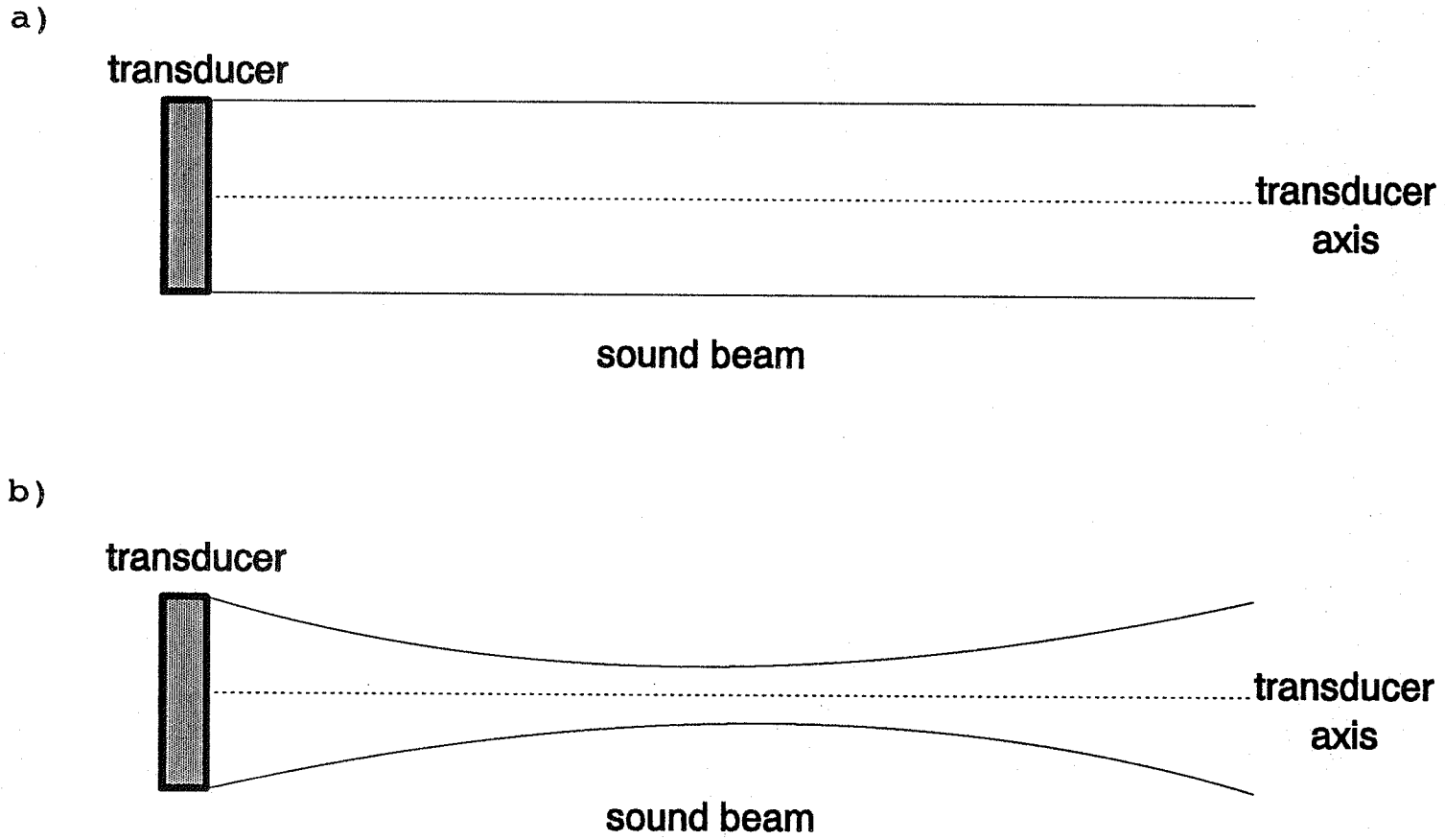


Figure 3 - a) Unfocused ultrasonic beam in the near field. b) Focused ultrasonic beam.

temporal pattern. Each point in both time and space has an instantaneous intensity. For continuous wave ultrasonic signals, only the spatial component is of concern because there is a constant energy output. The spatial peak (SP) intensity is the highest value of intensity in the sound beam, usually along the axis of the beam; and for focused beams, the spatial peak is in the focal zone. More frequently, the spatial average (SA) intensity is measured by dividing the total power output by a specified area of the beam. However, the area of the beam is arbitrary and depends on the specific definition used. For an unfocused beam, the area of the transducer face is often used but not necessarily. For both focused and unfocused beams, the beam width can be taken -3 dB, -6 dB, -10 dB or -20 dB down from the axial spatial peak intensity. This is the ambiguity of the spatial average intensity measurement.

Defining the intensity for a pulsed ultrasound signal is more involved than for a continuous mode signal because the pulsed signal varies both spatially and temporally. The spatial peak and spatial average are defined the same for pulsed mode as for continuous mode. The temporal pattern is also separated as temporal peak and temporal average. By choosing different combinations of these patterns, the intensities are defined as follows: 1) the spatial-average, temporal-average intensity (I_{SATA}) is the average power divided by the specified beam area (spatial average for continuous mode); 2) the spatial-average, temporal-peak intensity (I_{SATP}) is the spatial average value of the intensity during a pulse; 3) the spatial-peak, pulse-average intensity

(I_{SPPA}) is the intensity at the spatial peak in the sound beam over the pulse duration; 4) the spatial-peak, temporal-average intensity (I_{SPTA}) is the intensity at the spatial peak over the pulse repetition period; 5)the spatial-peak, temporal-peak intensity (I_{SPTP}) is the largest instantaneous intensity value during the pulse at the spatial peak.

How do you calculate these intensities? First, a pressure signal is captured from the transducer. For this purpose, a hydrophone receives the ultrasonic signal. However, the actual signal received is a voltage signal which is proportional to the pressure. According to AIUM standards [2], the center frequency (f_c) is defined as

$$f_c = \frac{f_1 + f_2}{2} \qquad (1)$$

where f_1 and f_2 are the frequencies at which the transmitted acoustic pressure spectrum is 71% (-3 dB) of its maximum value. To determine the center frequency, the hydrophone/amplifier system must be located at the point of the spatial peak. The hydrophone's output voltage shall be used to generate a frequency spectrum display. Either of two methods can be used for generating this frequency spectrum:

a) an automatic spectrum analyzer with an operating range encompassing the frequency envelope of the detected signal.

b) A computational system capable of performing Fourier transforms and either displaying the spectrum or calculating the desired values.

Determining the center frequency is important because hydrophone sensitivities are based on the center frequency.

The sensitivity of the hydrophone is often given as the end-of-cable open-circuit sensitivity (M_c), which is the sensitivity of the hydrophone at the end of the cable when it is not connected to any devices, i.e., no electrical load. When the hydrophone is connected to an electrical load, such as an amplifier, the sensitivity changes to what is called the end-of-cable loaded sensitivity (M_L). Consider the hydrophone as a two-terminal network with a complex impedance Z, such that when loaded, the hydrophone sees an impedance Z_L. The end-of-cable loaded sensitivity, M_L, is related to the end-of-cable open-circuit sensitivity, M_c, in the following method [3].

$$M_L = M_c \left[\frac{Re(Z_L)^2 + Im(Z_L)^2}{[Re(Z_L) + Re(Z)]^2 + [Im(Z_L) + Im(Z)]^2} \right]^{1/2} \quad (2)$$

where Re and Im are the real and imaginary parts of the relevant complex impedance, respectively. If the electrical load can be assumed to be a parallel combination of a resistance R_L and capacitance C_L, then $Re(Z_L)$ and $Im(Z_L)$ are given by

$$Re(Z_L) = \frac{R_L}{1 + \omega^2 C_L^2 R_L^2} \quad (3)$$

and

$$Im(Z_L) = \frac{-\omega C_L R_L^2}{1 + \omega^2 C_L^2 R_L^2} \qquad (4)$$

where ω is the angular frequency. If both the hydrophone and the load can be assumed to be capacitative, then Equation (4) can be further reduced to

$$M_L = M_C \left[\frac{C}{C + C_L} \right] \qquad (5)$$

where C is the end-of-cable capacitance of the hydrophone, including any integral cable and connector, and can be approximated by

$$C = \frac{-1}{\omega Im(Z)} . \qquad (6)$$

The voltage (v) signal can now be converted to a pressure (p) signal, according to

$$p = \frac{V}{M_L(f_c)} \qquad (Pa) \qquad (7)$$

where $M_L(f_c)$ is the end-of-cable loaded sensitivity. The peak compressional pressure, p_c, is the temporal peak compressional pressure amplitude, and the peak rarefactional pressure, p_r, is the temporal peak rarefactional pressure amplitude at a specified spatial point; these are also calculated as in Equation (7).

Once the signal is representative of the pressure, the instantaneous intensities are calculated,

$$I_{instant} = \frac{p^2}{10^4 \rho c} \qquad (W/cm^2), \tag{8}$$

where p is the pressure, ρ is the density of water, and c is the speed of sound in water. The pulse intensity integral (PII) is then calculated as [2]

$$PII = \frac{\int V^2(t)\ dt}{10^4 \rho c\ M_L^2(f_c)} \qquad (J/cm^2) \tag{9}$$

where v is voltage, ρ is the density of water, c is the speed of sound in water, and $M_L(f_c)$ is the end-of-cable loaded sensitivity. If the signal is stored digitally, the integral is treated as a summation of the instantaneous intensities multiplied by the sampling period (Δt).

$$PII = \frac{\sum v^2(i)\ \Delta t}{10^4 \rho c\ M_L(f_c)} \qquad (J/cm^2) \tag{10}$$

The pulse intensity integral is used to calculate the pulse duration (pd) and is a good starting point for calculating intensities. The pulse duration is

$$pd = (1.25)\ \tau \qquad (sec) \tag{11}$$

where τ is the time between the 10% and 90% values of the pulse intensity integral.

In this system, the spatial average parameters are not calculated because of the difficulty in determining the cross-sectional area of the beam width. The spatial peak parameter is determined by moving the hydrophone along the x-, y-, and z-axes until the largest pressure pulse is received. Once the spatial peak is determined from the field, the following intensities calculated are spatial peak values. From the pulse intensity integral and the pulse duration, the spatial-peak, pulse average intensity, I_{SPPA}, is calculated as follows

$$I_{SPPA} = \frac{PII}{pd} \qquad (W/cm^2) . \tag{12}$$

From the I_{SPPA}, the spatial peak temporal average intensity, I_{SPTA}, is calculated as follows

$$I_{SPTA} = (I_{SPPA})\,(duty\ cycle) \qquad (W/cm^2) \tag{13}$$

where the duty cycle is the ratio of the pulse duration over the pulse repetition period. Equivalently, the spatial-peak, temporal average intensity can be calculated using the pulse intensity integral directly:

$$I_{SPTA} = (PII(\max))\,(prf) \qquad (W/cm^2) . \tag{14}$$

The last intensity calculated with this system is the spatial-peak, temporal peak intensity, I_{SPTP}. The temporal peak intensity is simply the greatest instantaneous intensity value in the pulse. The spatial-peak, temporal-peak intensity is the highest intensity of these three intensities.

CHAPTER 3

HARDWARE

This chapter describes the hardware used in the exposimetry system. A 386sx pc compatible computer interfaces with three computer boards. A digitizing board captures the acoustic signal received by the hydrophone; an inport/outport board interfaces with a programmable attenuator which changes the hydrophone signal to fit the dynamic range of the digitizing board; a custom-designed and custom-built board calculates the pulse repetition period from the trigger signal of an inductor coil. A block diagram of these interfaces is shown in Figure 4. The exposimetry system also includes a custom-built water tank. Figure 5 shows the general setup of the system including the diagnostic ultrasound unit. The hardware specifications meet the criteria set by the AIUM standards [2].

3.1 Computer

The system computer had to be self-contained, moveable and capable of making calculations in real time. The desire for the system to be used for subsequent years meant that the system had to be expandable and capable of handling upgrades. A CompuAdd 316s computer was selected for this system.

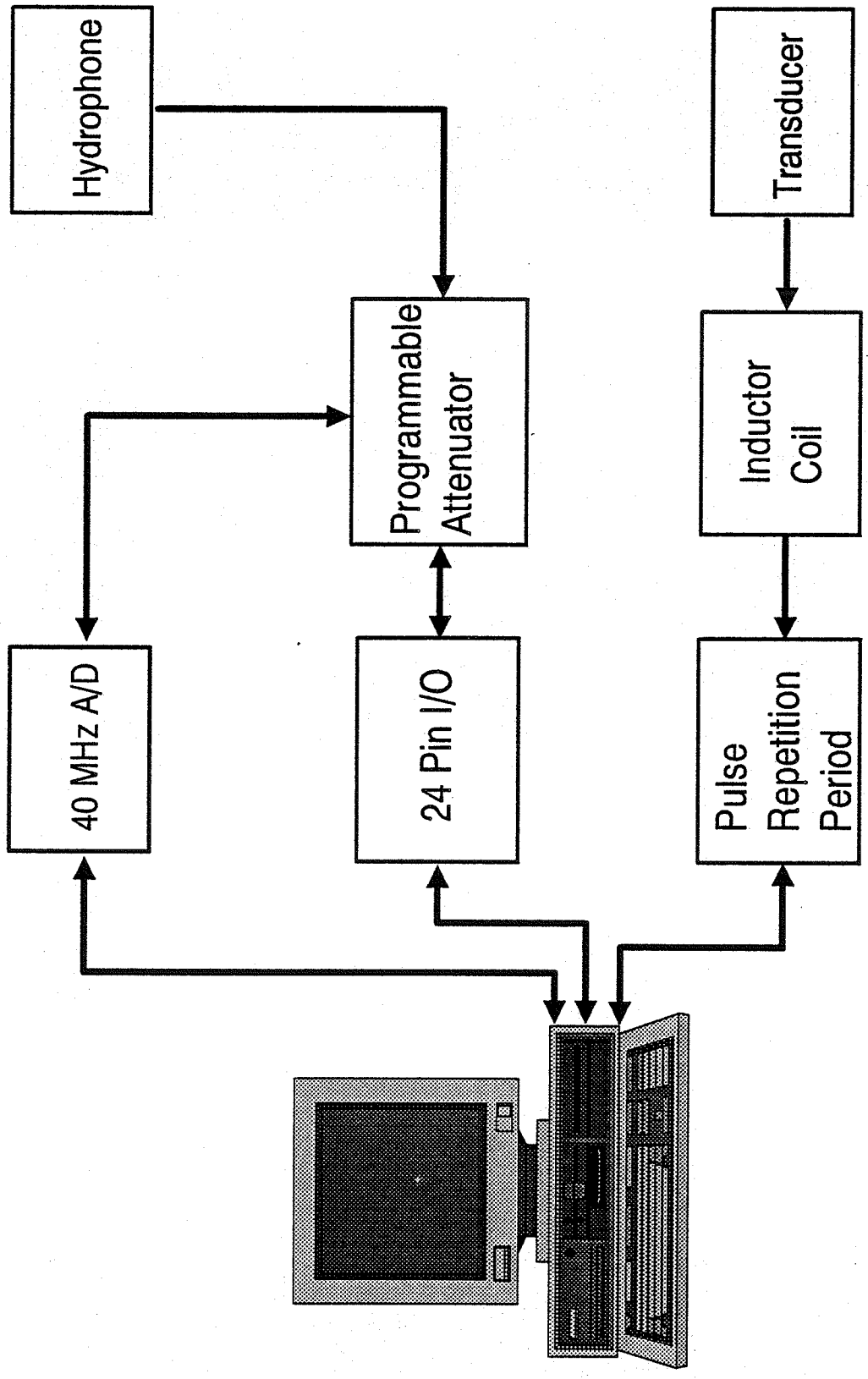


Figure 4 - Block diagram of hardware interfaces.

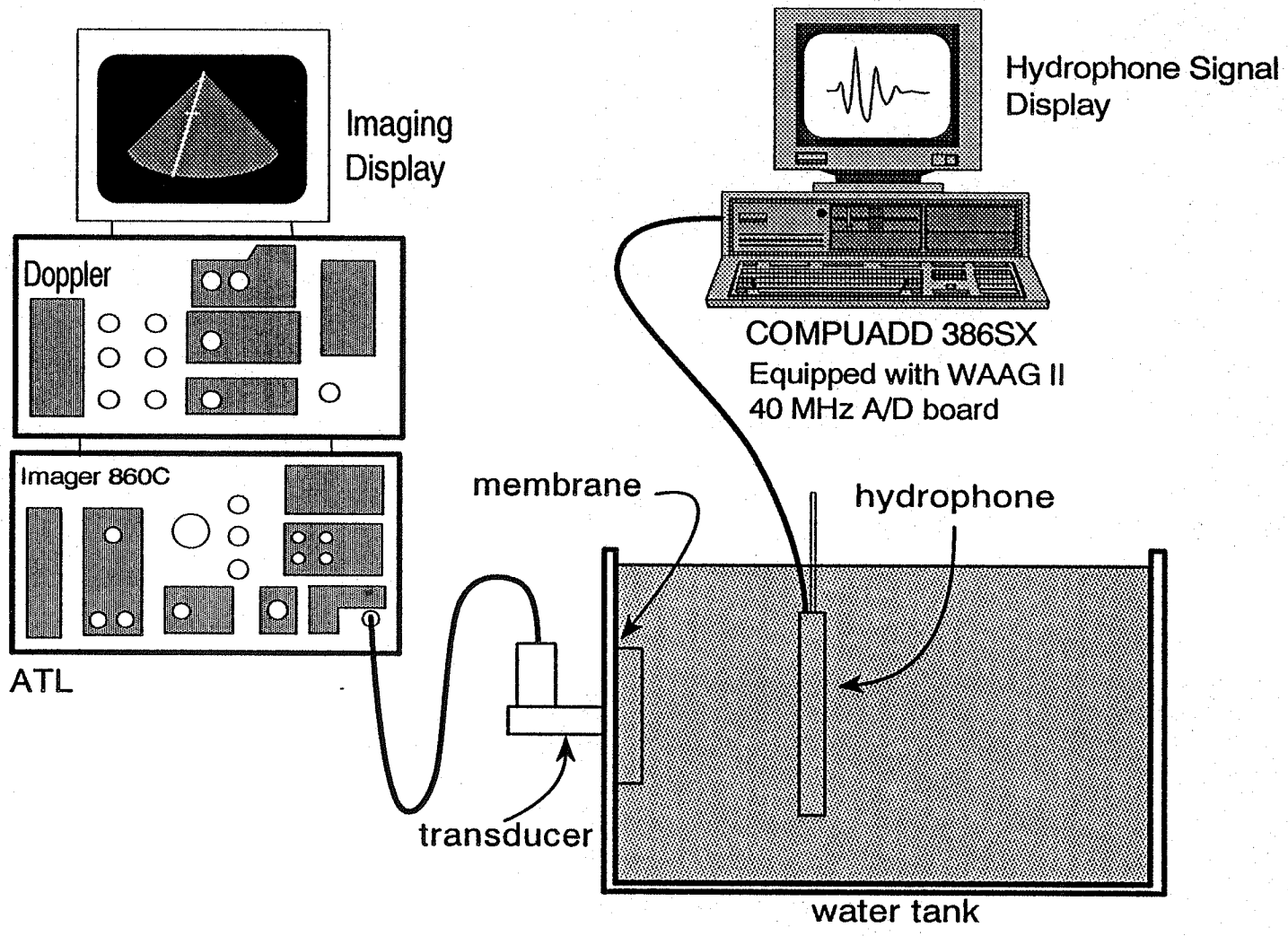


Figure 5 - Basic equipment setup during calibration.

The 316s computer features a 386sx microprocessor running at 16 MHz. It also includes a 387sx math coprocessor to enhance the speed of floating-point operations and provides 32 bit precision. The 316s has 2 Mbytes of 0 wait state page-mode memory and a 40 Mbyte hard drive for running and storing programs and data. This particular model includes one parallel port and two serial ports. Also, three full-size 16-bit and 2 half-size 8-bit expansion slots are in a low profile casing. In addition, two diskette drives were installed, a 5.25" 1.2 Mbyte and a 3.5" 1.44 Mbyte. This system includes a CompuAdd VGA monitor and graphic adapter and a 24-pin Panasonic kx-p1124 dot matrix printer. More information on the CompuAdd 316s computer can be found in the references [4].

3.2 Marconi PVDF Hydrophone

Typically, for diagnostic equipment a polymer polyvinylidene diflouride miniature hydrophone is selected. In most systems and this one, the membrane coplanar shielded type with an active element of diameter 0.5 mm is used. The advantage of the hydrophone chosen is the flat frequency response in the low mega Hertz range which extends to 40 MHz [5]. Also, the membrane hydrophone minimizes field deformations, caused by its insertion [6].

Originally, a Sonic Technology 0.4 mm coplanar PVDF membrane hydrophone was purchased for this system. However, due to the poor signal to noise ratio, the Sonic Technology hydrophone could not be used. A Marconi 0.5 mm bilaminar PVDF membrane hydrophone

was available. The calibration by National Physical Laboratory of the Marconi hydrophone is shown in Table 1.

Table 1: Calibration listing of the Marconi 0.5 mm bilaminar PVDF membrane hydrophone (Serial No. IP041)
Water Temperature: 20.7 ± 0.5 °C

Frequency	End-of-cable open-circuit sensitivity	Impedance	
		Re(z)	Im(z)
MHz	μV/Pa	Ω	Ω
1	0.060	140	-1910
2	0.060	80	-1000
3	0.062	60	-680
4	0.063	44	-520
5	0.066	34	-419
6	0.067	29	-351
7	0.068	26	-302
8	0.071	23	-264
9	0.073	21	-235
10	0.077	19	-211
11	0.079	18	-191
12	0.084	17	-174
13	0.087	16	-161
14	0.092	16	-148
15	0.098	14	-137

Table 1 lists the calibration as the end-of-cable open-circuit, M_c. The end-of-cable loaded sensitivity, M_L, is determined as described by Equation (5) in Chapter 2 -- Measurement Theory. This calculation is done in software.

3.3 WAAG II Digitizing Board

The analog signal had to be acquired in accordance with the Nyquist criterion, that is, two times the highest frequency possibly output from the clinical system. At the time, a Waveform Acquisition and Arbitrary Generator board (WAAG II) made by Markenrich Corporation was available. The hardware of the WAAG II includes data acquisition and storage, time base, trigger circuit,

signal conditioning and generator output [7]. Through hardware and software, the WAAG II is used in single-channel 40 MHz mode and internally triggered to acquire and display a waveform. The WAAG II's memory is 32 kbyte, high speed (45 ns), static RAM (random access memory). Sampling at 40 MHz with a minimum signal frequency of 0.5 MHz, only 1 kbyte of RAM is needed to store the waveform. This provides resolution of 8 bits. The trigger source is selected by a slide switch located on the board. It has four registers: two 16-bit address registers, one 16-bit control register, and one 8-bit status register. The full scale input signal is hardware selected to be +0.635 to -0.640 V. More information is provided in [7].

3.4 Amplifier-Attenuator-Amplifier Series

The hydrophone signal is in the range of 30 to 50 mV when in the spatial peak of the sound beam. When the hydrophone is moved out of the spatial peak, the signal is much smaller. Thus, the signal needs to be amplified to make it robust within the limitations of the WAAG II board. A combination of amplifier-attenuator-amplifier is used to fit the hydrophone signal to the dynamic range of the WAAGII board. The combined amplification of the amplifiers, when the attenuator is set to zero, needed to have a gain of 40 dB, such that a 10 mV signal would be amplified to 1 V. Since the amplifiers have set gains, a variable attenuator allowed the amplification to be changed. The attenuator needed to attenuate from 0 dB to 40 dB, at least to the amplified amount in order to keep the WAAG II from saturation. The attenuator also

needed to be computer controlled to automate the amplification of the signal. All three components require flat frequency responses in the range of 0.5 MHz to 10 MHz.

A 17 dB Marconi Hydrophone Amplifier [8] amplifies the signal directly from the hydrophone cable because a Marconi hydrophone is used. This particular hydrophone was calibrated by National Physical Laboratory as listed in Table 2.

Table 2: Calibration listing of the Marconi Hydrophone Amplifier. Serial Number Y-33-9724 IP134. RMS input voltage to device was 100 mV.

Frequency MHz	Gain dB
0.5	16.7
1.0	16.8
2.0	16.8
3.0	16.8
4.0	16.8
5.0	16.7
6.0	16.7
7.0	16.6
8.0	16.5
9.0	16.5
10.0	16.4
11.0	16.3
12.0	16.2

The Marconi Hydrophone Amplifier also changes the phase of the signal by 180°.

The signal is attenuated by the Wavetek P1506 attenuator [9] (impedance of 50 Ω) after being amplified by the Marconi Hydrophone Amplifier. This attenuator attenuates from 0 to 63 dB in 1 dB increments (± 0.1%) with a switching speed of 6 msec. The frequency range is from dc to 1500 MHz. The attenuator is TTL controlled and interfaces with the PIO24 inport/outport computer

board. The PIO24 is a basic inport/outport board with a 24 pin "D" connector. Though the attenuator only uses 15 pins, the PIO24 provided extra connections that may be needed in the future.

Directly connected to the other side of the attenuator is a MiniCircuits ZFL500LN broadband linear power amplifier with a frequency range of 0.1 to 500 MHz. The MiniCircuits amplifier is measured to be 26 dB gain. More detailed specifications are in the references [10]. Both amplifiers are powered by a ± 15 V power supply.

3.5 Custom-built Pulse Repetition Period Board

After installing the WAAG II board and running simple signals, it became apparent that the pulse repetition period (prp) could not be calculated with the WAAG II. Given the slowest pulse repetition frequency (prf) of 0.5 kHz and sampling at 40 MHz, 32 kbyte of memory could not store two pulses. Without storing at least two pulses the pulse repetition period could not be calculated. Thus, a separate computer board was designed and built to calculate the pulse repetition frequency. The method using an inductor coil next to the transducer head seemed easiest to implement to provide an appropriate trigger. When the transducer pulses, the change in electromagnetic field creates a current in the coil as described the Biot-Sarvat Law,

$$\vec{B} = \frac{\mu_o}{4\pi}\int_{spiral} \frac{Id\vec{l} \times \hat{i}_R}{R^2}, \qquad (15)$$

where B is the magnetic flux density, R is the distance from the current element to the point of change in magnetic flux density, i_R is the unit vector along the line joining the current element and the point in the flux density, I is the current, dl is the element length and μ_o is the permeability of free space. By empirical means, an inductor coil of fifteen turns in a flat spiral provided a strong signal, approximately 1 V. Figure 6 illustrates Equation (15). This coil is embedded in black epoxy to provide protection from handling.

The coil is taped to the side of the transducer and is connected to the Vector Electronic 4617 Plugbord by a coaxial cable and BNC connector. This analog signal is amplified and triggers a one-shot CMOS chip. This combination has changed the analog signal to a digital pulse. The digital pulse, designed to be 10 μsec, latches the value of a continuously running 1 MHz clock and stores the value in memory. Another analog signal creates another digital pulse and latches the new value of the clock. Refer to Appendix A for the circuit diagram. The difference between these two values is the pulse repetition period. The calculation of the prp is done in software.

3.6 Water Tank

Finally, the system is only complete with a water tank, which was custom designed for this system and is shown in Figure 7. It is constructed of 0.75" Plexiglas for stability and with dimensions: length - 35.5 cm; width - 27.1 cm, and depth - 25.9 cm. There is a front opening in which a plastic membrane is held

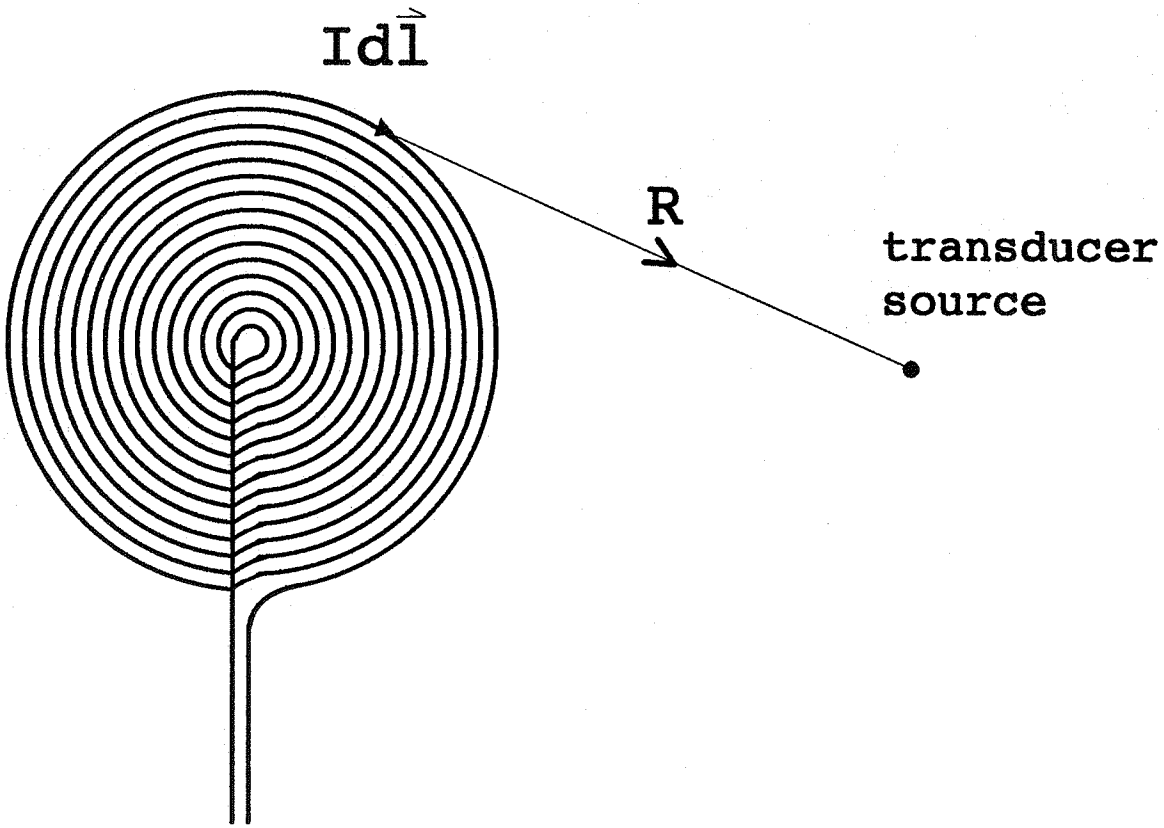


Figure 6 - Current induced in an inductor coil in a flat spiral with fifteen turns.

in place by a 3" diameter aluminium anodized ring. This membrane can be secured from the inside or the outside of the tank, depending on the needs of the user. The transducer is coupled to the outside of the tank through this membrane with ultrasonic gel. It is held in place with adjustable clamps, which are designed to accommodate various configurations of commercial transducers. The tank is also equipped with vertical and lateral coarse movements, on which a Daedal XYZ high precision positional apparatus is attached. The Daedal XYZ system travels in all axial directions of 25 mm and a repeatability of 0.000050" for fine positioning of the Marconi hydrophone, which is mounted in the system.

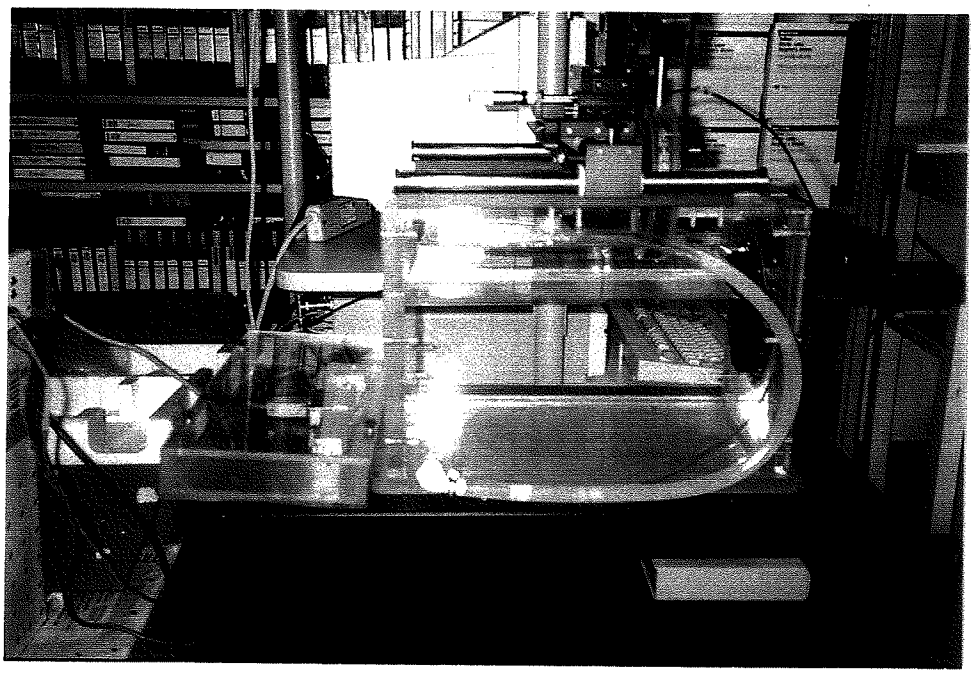


Figure 7 - Water tank on which Daedal XYZ positioning system is mounted. Amplifier series left of tank.

CHAPTER 4

SOFTWARE

This chapter describes the software that controls the three main hardware components, the WAAG II board, the PIO24 I/O board to the programmable attenuator, and the custom-built pulse repetition period board. Figure 4 is a block diagram of the necessary interfaces. Once, a hydrophone signal is digitized, the software performs all of the calculations to determine the intensities. These calculations include determining the center frequency, calculating the hydrophone sensitivity, converting the hydrophone voltage signal to the corresponding pressure signal, and calculating the different intensities. All calculations are based on the AIUM standards [2].

4.1 Controlling Hardware

The computing language "C" was chosen for writing all the software because it has the capabilities of a high level language and the necessary hardware interaction. A Microsoft C Optimizing compiler [11] was used because it produces efficient machine code for the CompuAdd 316s and compatible machines. Markenrich Corp. provided the "bare bones" C program to run the WAAG II board to acquire a signal [7]. However, the Markenrich Corp. software for

plotting the waveform was inadequate. A program to arm and acquire a signal and to display the signal to the screen was developed for another project [12]. This seed program became the body of the current main program (see Appendix B). As the project developed, needed subroutines were written and called from the main program. This maintained a working basis upon which to rely. The following sections describe the programs and subroutines created to control the hardware. The actual C coding is listed in the Appendices.

4.1.1 Controlling the WAAG II board

As mentioned, Markenrich Corp. provided the essential coding for the WAAG II board [7]. The WAAG II port locations are defined in the beginning of the main program. The default memory locations are used because they do not interfere with other boards or software used in this computer system. To acquire a signal, the main program calls subroutine "sample" (see Appendix C).

"Sample" arms the WAAG II board and waits for a trigger. It then stores 1 kbyte of points in the WAAG memory. For this system to display the beginning of the waveform, it is necessary to display the presampled data, the data sampled before the trigger. However, not knowing when the board would be armed relative to the signal, the amount of presampled data changed for each pass of the acquiring loop in the program. To synchronize the presampled data with the trigger and guarantee the amount of presampled data stored, the board is immediately armed again after one signal has been acquired. This ensures that a sufficient number of

presampled data points and that the entire signal after the trigger are stored. The second signal is the signal displayed on the screen and stored.

4.1.2 Controlling the programmable attenuator

An amplifier-attenuator-amplifier series is connected between the hydrophone and WAAG II board. This combination allows the signal to fit the dynamic range of the board and a wider range of the sound beam to be detected and displayed. The two amplifiers have set amplifications as described in Chapter 3 -- Hardware. Because the attenuator is programmable, the exact amplification is known so as to meet the needs of the acoustic signal. The subroutine "attenuation" (see Appendix D) controls the setting of the attenuator through the PIO24 board. The port locations used by the PIO24 computer board are set at the beginning of the main program. To minimize the setting time, three major increment/decrement steps are used, 10, 5, and 1 dB. Knowing that every sampled signal would not have exactly the same amplitude, a window of 0.5 to 0.6 V was selected as the acceptable signal amplitude to keep the attenuator from constantly resetting. The attenuator is limited from 0 to 63 dB. However, is it possible to write larger numbers or negative numbers to the attenuator with the attenuator only registering the last 6 bits. This created a slight problem in decrementing and incrementing where it is crucial to keep track of the attenuation value. Conditions to check for exceeding 63 dB (signal too large) and underwriting 0 dB (signal too small) were made and corrected back to either 63 or 0

dB. This made writing the code more difficult because all possible cases of error had to be confronted. If not, it would be possible to lose the exact attenuation value or hang the program.

Once the initial "attenuation" subroutine was implemented, another problem arose. The attenuator is set at 60 dB at the beginning of the main program to prevent a possibly large signal from damaging the WAAG II board. In order to change the setting on the attenuator, a sampled signal must be stored by the WAAG II and checked for its maximum amplitude. However, more frequently, the signal is too small so that "sample" never receives a trigger; "sample" waits for a trigger indefinitely and, thus, hangs the computer. A new subroutine "atten_sample" (see Appendix E) was created to be called exclusively when setting the attenuator. "Atten_sample" essentially is the same as "sample." However, "atten_sample" limits the waiting time for a trigger to 2 seconds. If the trigger is not set within 2 seconds, the attenuator is decremented 10 dB. If the attenuator is decremented below 0 dB, the attenuator is reset to 0 dB, and a message is written to the screen along with an audible beep while waiting for a signal. The other feature in "atten-sample" is that the user can quit the program by typing "q."

After the triggering problem was solved, "attenuation" still required fine tuning. Appropriate voltage windows were made for the decrementing and incrementing schemes. For 10 dB, the window is 0.2 V to 0.6 V. This provided a window slightly smaller than 10 dB. The software checks that the attenuator is not setting the voltages constantly outside the window. The window for the 5 dB

incrementing is 0.4 V to 0.6 V. Again, this window is slightly smaller than 5 dB, but after 13 passes automatically continues to 1 dB increments. For 1 dB increments, 0.5 to 0.6 V is the window. This window is larger than a 1 dB voltage change so that a signal slightly larger (smaller) than the window will be decremented (incremented) into the window. These decrementing and incrementing schemes made setting the attenuator more efficient.

4.1.3 Controlling the custom-built pulse repetition period board

The third piece of hardware which is computer controlled is the custom-designed board used to calculate the pulse repetition period. As described in Chapter 3 -- Hardware, the board has a continuously running clock which is latched each time the transducer pulses. Subroutine "read_usec" (see Appendix F) reads the memory location in which the clock's latched value is stored and puts these values in an array of 300 elements. This scheme is computer dependent; the rate at which the memory is read and stored is dependent on the computer's access time and bus. However, by storing 300 elements, a 0.5 kHz pulse repetition frequency can still be measured. Subroutine "dprp" (see Appendix F) calls "read_usec" and returns the pulse repetition period value by searching the array for the first change in clock values and subtracting the values. The uncertainty of the measurement is just under ± 10% at 30 kHz and decreases as the pulse repetition frequency decreases.

4.2 Calculations

Once the signal is within the limits of the WAAG II board, calculations are made. The WAAG II digitizes a signal and stores it in WAAG II memory. This signal is then written to the computer memory and stored as an array of relative voltage amplitudes. The relative voltages stored are of the signal that the WAAG II sees. However, this signal has been amplified. Because the amplifier/attenuator combination was used, the exact amplification is known, and the relative voltages directly from the WAAG II board are converted to their respective unamplified hydrophone voltages. With the unamplified hydrophone signal, all calculations are made following AIUM standards [2].

First, the center frequency is calculated by performing a Fourier transform of the voltage signal in subroutine "dtft" (see Appendix G). The actual Fourier transform of the signal is performed by a professional subroutine "realft" as written in Numerical Recipes in C [13]. From the largest magnitude peak of the signal in the frequency domain, the two 3 dB points are determined as described in Chapter 2 -- Measurement Theory. Given the bandwidth, the center frequency is determined as described by Equation (1). Other coding in "dtft" is for plotting the frequency response.

The center frequency is used to determine the end-of-cable loaded sensitivity of the hydrophone (M_L) (see Appendix H). The Marconi hydrophone is calibrated by the National Physical Laboratory with the calibration listing given in Table 1 in Chapter 3 -- Hardware. This listing only reports the end-of-cable

open-circuit sensitivity and the real and imaginary parts of the hydrophone impedance. The end-of-cable loaded sensitivity is determined as described by Equations (5) and (6). Because the calibration is only for integer frequencies, linear regressions are performed between each integer frequency. Therefore, if the center frequency is not exactly an integer value, the calibration constants can be calculated, and thus, the end-of-cable loaded sensitivity can be calculated too.

Given the end-of-cable sensitivity of the hydrophone, the digitized voltage signal is changed to a digitized pressure signal, according to Equation (7). This calculation is done in subroutine "press" (see Appendix I). In order to plot the pressure signal, another array stores the corresponding time value. Each data point is acquired at 40 MHz such that the time interval is 25 nsec. Other coding in "press" is for plotting the pressure pulse.

Finally, with the pressure signal, the intensity values are calculated in subroutine "int_time_ave" (see Appendix J). The instantaneous intensities are calculated, using Equation (8). The largest instantaneous intensity value in the pulse is called the temporal peak intensity, I_{TP}. The pulse intensity integral, PII, is calculated, as described by Equation (10), so that the pulse duration and other intensity values can be calculated. The maximum pulse intensity integral is stored, and pulse average intensity, I_{PA}, calculated (see Equation (12)). From here, the duty cycle is calculated as the ratio of the pulse duration to the pulse repetition period, which is determined in subroutine "dprp"

(see Appendix F). The temporal average intensity, I_{TA}, is computed as in Equation (13). The spatial peak values of these three intensity values is when the Daedal system (see Hardware Chapter) positions the hydrophone in the spatial peak of the sound beam. Then these values become the spatial peak temporal peak (I_{SPTP}), spatial peak pulse average (I_{SPPA}), and spatial peak temporal average (I_{SPTA}) intensities.

The last subroutine is "done" (see Appendix K). All values need to recalculate intensity values are saved. Other important values saved are the peak compressional pressure, p_c, and the peak rarefactional pressure, p_r. "Done" simply displays all the important values to the screen and stores them in a file.

Other subroutines listed in Appendix L are to aid in these calculations. For example, "findmax" is a simple subroutine that searches the y array for the maximum value. Because searching the y array is done more than once, it is useful to write a separate subroutine. These subroutines include "load_trigger", "absolute_max", "findmax", and "findmin".

CHAPTER 5

UNCERTAINTY AND IMPROVEMENTS

This chapter presents the uncertainty of measurements by the exposimetry system that was developed. Included in this chapter are the accuracy and precision of the equipment. A brief discussion of possible improvements are concluded from these uncertainties.

5.1 Uncertainties

In the exposimetry system, each component contributes to the accuracy and precision of the measurement. Different components of the hardware have criteria to be met which decrease the amount of error in the measurement.

5.1.1 Computer

The CompuAdd 316s computer has 32-bit precision and runs at 16 MHz [4]. All calculations are performed by the computer; its precision influences the uncertainty of the measurement minimally because the uncertainty of other instruments is much greater.

5.1.2 WAAG II board

The WAAG II digitizes at 40 MHz with 8-bit precision [7]. The input voltage is set for +0.635 V to -0.640 V, but is measured

to be +0.638 V to -0.648 V. This input voltage is represented as a character from 0 to 255 where 0.0 V is measured to be character 129. Thus, each bit represents a range of 0.00502353 V such that the uncertainty is ± 0.002511765 V. For a voltage signal near 0.0 V, the percentage of error is great; for a voltage near 0.6 V, the percentage of error is much less. The error is minimized when determining the pulse intensity integral because the values near 0.0 V contribute least to the integral. The error of measurement is also minimized in this system because the hydrophone signal is amplified such that the peaks are within 0.5 V to 0.6 V. For the worse case, where the peak is at 0.5 V, the uncertainty is ± 0.5%. This is consistent with the accuracy reported by Markenrich [7].

5.1.3 Hydrophone

The National Physical Laboratory reports the temperature dependency of the hydrophone sensitivity as in Table 3 [3].

Table 3: Temperature Dependance of Hydrophone Sensitivity From 15 °C to 25 °C.

2 MHz (% per °C)	10 MHz (% per °C)
0.1	0.4

The calibration uncertainties are also reported as dependent on frequency range for a 95% confidence level as shown in Table 4 [3].

Table 4: Overall Hydrophone Uncertainties at 95% Confidence Level

1 to 4 MHz	± 6%
5 to 8 MHz	± 7%
9 to 12 MHz	± 8%
13 to 15 MHz	± 9%

5.1.4 Amplifier-attenuator-amplifier series

The amplifiers and attenuator were each measured separately and together. GEC - Marconi Research Center reports that the hydrophone amplifier's gain linearity changes less than 0.1 dB for RMS input levels of 20 mV to 500 mV for 1, 5, 10, and 20 MHz [8]. Appendix M includes a copy of the gain vs. frequency response for the particular hydrophone amplifier used. The accuracy of the amplifier is listed in Table 5.

Table 5: Uncertainty of Gain Measurements in ± dB (and in ± %) as a Function of Frequency and Input Level

Input (mV)	1 MHz	5 MHz	10 MHz
10	0.22 (2.6)	0.66 (7.9)	1.43 (18.0)
100	0.13 (1.5)	0.37 (4.4)	0.67 (8.0)

Wavetek reports that the programmable attenuator has an incremental accuracy of ± 0.5 dB or 2 % [9]. It is measured to attenuate for the amount set. MiniCircuits states that the gain linearity is ± 0.3 dB from 0.1 MHz to 500 MHz [10]. It was measured to have a gain of 26 dB for a 50 mV input (zero to peak) at 5 MHz. Using an HP Network Analyzer, HP3577B, the two amplifiers with the attenuator set at 0 dB are measured to be 44 dB over the range of 0.5 MHz to 10 MHz (see Appendix M).

5.1.5 Custom-built pulse repetition period board

The custom-built pulse repetition period board is measured to have an uncertainty of ± 10 %. For an input signal at 30 kHz, the board would calculate it to be in the range of 33 kHz to 27 kHz. When the input signal is at 1 kHz, the result would be in the range of 1.1 kHz to 0.9 kHz. The error is seemingly less at smaller frequencies, but the percentage of error is the same.

5.2 Improvements

This section includes both hardware and software improvements.

5.2.1 Custom-built pulse repetition period board

To improve the uncertainty of the custom-built pulse repetition period board, a faster clock could be substituted for the currently used clock which is running at 1 MHz.

As mentioned in Chapter 4 -- Software, the pulse repetition period calculation is computer-dependent. The dependency is based on the amount of time the computer uses to read a memory location and to write the value to an element in an array, which is determined by the bus speed. The solution to the problem is to make the pulse repetition period calculation completely in hardware. A simple subtraction circuit on the computer board could make the calculation after the two clock values are latched. This would eliminate the need for speed in reading the memory and writing the value.

5.2.2 Software

Currently, the acoustic signal is being digitized at 40 MHz by the WAAG II board. According to the Nyquist criterion, the WAAG II can at most digitize a 20 MHz signal for reconstruction. However, acoustic signals are nonlinear as shown in Figure 8 by the different shapes of the half cycles. Though the center frequency may only be 7.5 MHz, the nonlinearities of the wave and the media produce higher frequency harmonics. These higher frequencies may exceed the Nyquist criterion, and thus, the acoustic signal may not be fully reproduced. This means that a signal at 3.5 MHz will be represented by 11 points per cycle; a signal at 5 MHz has 8 points per cycle; and, a signal at 7.5 MHz will only have 5 points per cycle. At lower frequencies, the signal is represented by more points, and a larger number of the higher harmonics (up to 20 MHz) are represented. As the frequency increases, the number of points representing the signal decreases, and the higher harmonics are more likely to exceed 20 MHz. The fewer the points in the signal, the more difficult it is to determine the peaks of the signal as seen in Figure 8.

There are a number of solutions to this problem, but not all are applicable. The easiest and most reliable solution is to digitize the signal at a faster rate. Only by this method can all of the acoustic signal be reconstructed. However, this is not feasible with our system because the WAAG II is already set to digitize as fast as it can. Otherwise, a faster digitizing board has to be purchased.

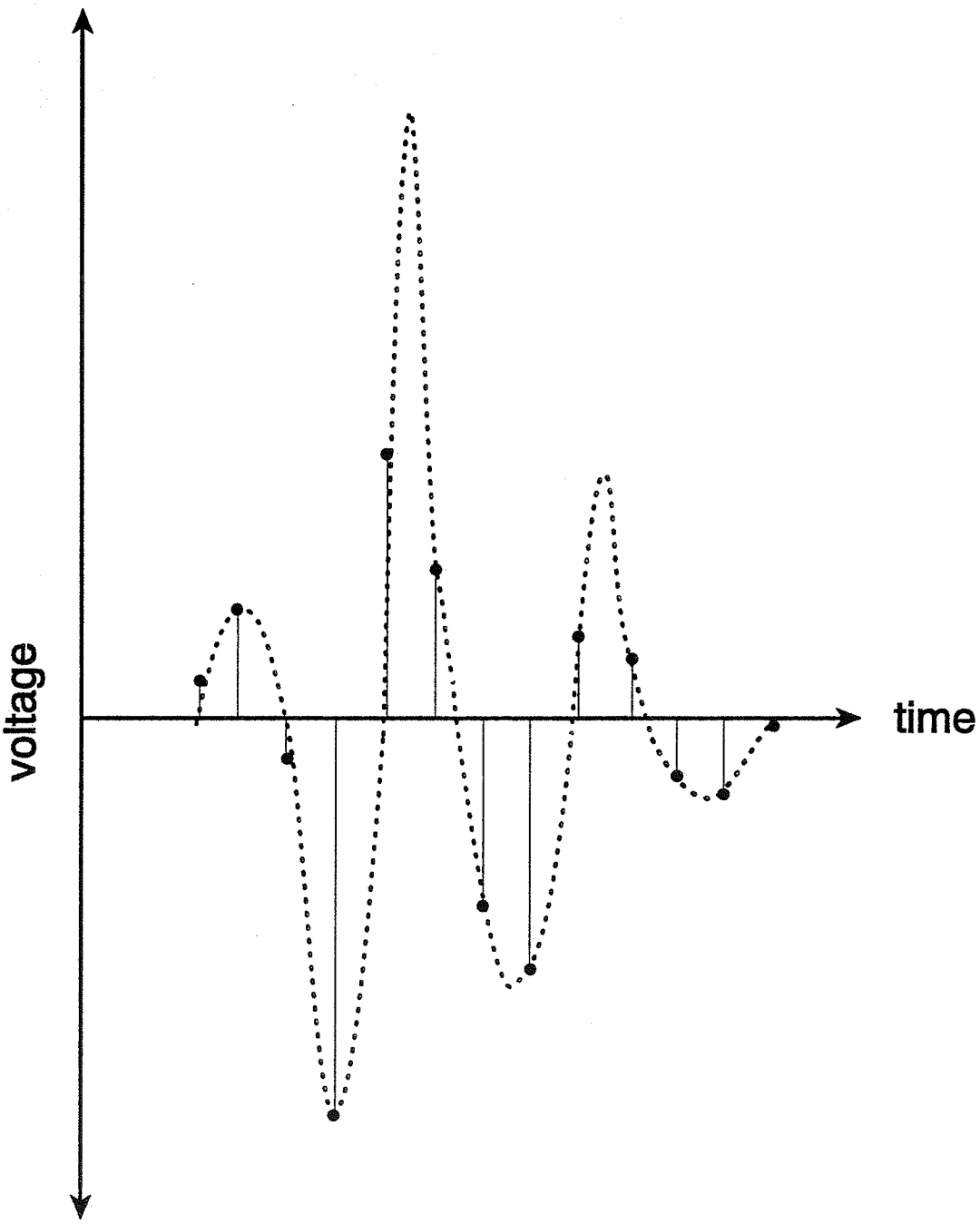


Figure 8 - Example of digitized pulse with 4 points per cycle.

Another, solution would be to have a mask so that the digitized signal would have another signal in which to compare. The peaks could then be determined from a correlation between the two signals. A large number of wave shapes, e.g., 10000 waveforms, would have to be sampled and averaged to create the mask. This mask would be stored and compared as each new echo was measured. This method presupposes the wave shape and assumes that the echo never changes. It also is dependent on the speed of the computer to sample and average the waveforms; but this only has to be done once.

A final solution presented here is to extrapolate the peaks. Because the compressional half cycles and the rarefactional half cycles are shaped differently, the extrapolations are different. The compressional half cycles are more spiked in shape than the rarefactional half cycles. The peak compressional pressure can be extrapolated by first finding the largest positive voltage point digitized (as compressional peaks are positive when measured with the Marconi hydrophone). A line with the two points before the point is determined; a line with the two points after the point is determined. The intersection of these two lines is the extrapolated peak compressional pressure. Because the rarefactional peaks are more rounded, a parabolic curve fit can be used to fit the 3 most negative points. The peak of the parabola is the extrapolated peak rarefactional pressure. Both methods can be used for the appropriate half cycles such that the entire waveform is reconstructed. However, reconstructing the entire waveform may not be necessary.

5.3 Summary

The computer performs all of the calculations with 32-bit precision so that its uncertainty is minimal. The hydrophone sensitivity uncertainty in the worst case is ± 7 %. For the frequency range in which the system is to be used, the uncertainty of the amplifier-attenuator-amplifier series is ± 1 %. Both the center frequency and the bandwidth calculation are determined by the uncertainty of the WAAG II and the amplifiers; thus, they each have an uncertainty of ± 1.1 %. The peak compressional pressure and peak rarefactional pressure are influenced by the hydrophone, the amplifiers and the WAAG II; therefore, the pressure value uncertainty is ± 7.1 %. Because the instantaneous intensity values are proportional to the square of the pressure values, the uncertainties are calculated to be ± 9.3 %. The pulse repetition period and the pulse repetition frequency have an uncertainty of ± 10 %. The pulse duration is calculated from the pulse intensity integral (uncertainty is ± 9.3 %) with an uncertainty of ± 13.2 %. The duty factor has an uncertainty of ± 16.1 % because of the pulse duration and the pulse repetition period. The uncertainties of the pulse average intensity (I_{PA}), the time average intensity (I_{TA}), and the time peak intensity (I_{TP}) are ± 16.1 %, ± 19%, and ± 9.3%, respectively.

CHAPTER 6

CONCLUSIONS

An exposimetry system was proposed to quantify the intensity output of the clinical ultrasonic device used at the California Primate Research Center. After studying the environment in which the system was to be used, a semi-automated exposimetry system was designed. The system was to be self-contained and maneuverable around the animal room. The system employs a CompuAdd 316s computer with the WAAG II, the PIO24, and the custom-built pulse repetition period boards installed. An amplified signal from the Marconi bilaminar hydrophone is digitized at 40 MHz by the WAAG II. The hydrophone is mounted in a manual XYZ positioning device which is part of a water tank. The clinical device is coupled to the membrane side of the water tank with ultrasonic gel. Figure 9 is a photograph of the system.

All calculations are made automatically by the computer after the hydrophone is in the desired position. These calculations include the following: center frequency, bandwidth, hydrophone loaded sensitivity, peak compressional pressure, peak rarefactional pressure, pulse repetition period, pulse repetition frequency, pulse duration, duty factor, pulse intensity integral, pulse average intensity (I_{PA}), temporal average intensity (I_{TA}), and

temporal peak intensity (I_{TP}). If desired, these values can be stored in a file for later reference. Graphs of the frequency response and pressure signal and the calculated values can be printed.

Figure 9 - Picture of system. Computer in cart to the right. Water tank with hydrophone to the left.

REFERENCES

[1] *Diagnostic Ultrasound Imaging in Pregnancy. Report of a Consensus Development Conference*, National Institutes of Health, NIH Publication #84-667, US Government Printing Office, Washington, DC, 1984.

[2] *Acoustic Output Measurement and Labeling Standard for Diagnostic Ultrasound Equipment*, AIUM, Bethesda, Maryland, 1988.

[3] *Certificate of Calibration (Serial No. IP041)*, National Physical Laboratory, Teddington, Middlesex, England, 1987.

[4] *System Manual for the CompuAdd 316s*, CompuAdd, Austin, Texas, 1990.

[5] R.C. Preston, "The NPL ultrasound measurement service and comments on the measurement and specification of the acoustic output of medical ultrasonic equipment," *Ultrasonics International 85 Conference Proceedings*, pp. 269-274, London, United Kingdom, July 1985.

[6] F.A. Duck and H.C. Starritt, "Acoustic shock generation by ultrasonic imaging equipment," *The British Journal of Radiology*, vol. 57, pp. 231-240, 1984.

[7] *WAAG II User Guide*, Markenrich Corporation, Duarte, California, 1988.

[8] *Membrane Hydrophone Amplifier Characteristics (Serial No. IP134)*, GEC-Marconi Research Center, Chelmsford, England, 1988.

[9] *RF and Microwave Components Catalog*, Wavetek RF Products, Inc., Indianapolis, Indiana, 1990.

[10] *RF/IF Signal Processing Guide*, Mini-Circuits, Brooklyn, New York, 1989-1990.

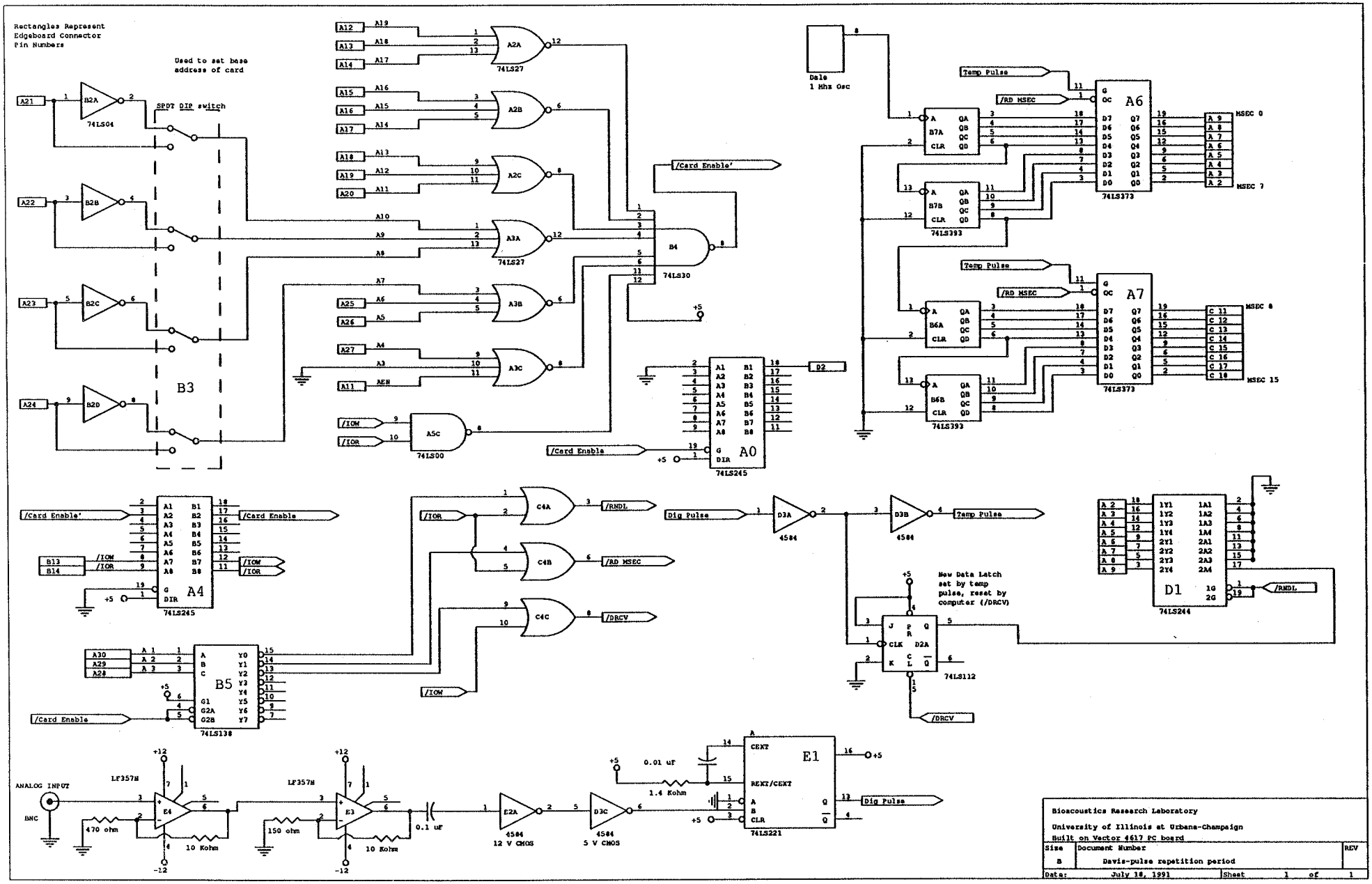
[11] *Microsoft C 5.1 Optimizing Compiler User's Guide*, Microsoft Corporation, Redmond, Washington, 1987.

[12] Personal correspondance with I.A. Hein, Ph.D.

[13] W.H. Press, B.P. Flannery, S.A. Teukolsky and W.T. Vetterling, *Numerical Recipes in C. The Art of Scientific Computing*. New York: Cambridge Press, 1988.

APPENDIX A

PULSE REPETITION PERIOD COMPUTER BOARD SCHEMATICS


Rectangles Represent
Edgeboard Connector
Pin Numbers
Used to set base
address of card
SPDT DIP switch
B3
74LS04
74LS27
74LS30
74LS00
74LS245
74LS373
74LS393
74LS138
74LS244
74LS112
74LS221
4584
LF357N
Dale
1 Mhz Osc
Temp Pulse
/RD MSEC
/Card Enable'
/Card Enable
/IOW
/IOR
/RNDL
/DRCV
Dig Pulse
New Data Latch
set by temp
pulse, reset by
computer (/DRCV)
ANALOG INPUT
BNC
470 ohm
10 Kohm
150 ohm
0.1 uF
0.01 uF
1.4 Kohm
12 V CMOS
5 V CMOS
MSEC 0
MSEC 7
MSEC 8
MSEC 15
Bioacoustics Research Laboratory
University of Illinois at Urbana-Champaign
Built on Vector 4617 PC board
Size
B
Document Number
Davis-pulse repetition period
REV
Date:
July 18, 1991
Sheet
1
of
1

APPENDIX B

EXPOSIMETRY SYSTEM MAIN PROGRAM

```
/*****************************************************************************/
/*  Program calibrat.c reads an analog signal input to the WAAG II board.  The*/
/*  WAAG II board digitizes the signal at 40 MHz.  It waits for an internal   */
/*  trigger and then will store the next 1024 points (1 kbyte of memory).     */
/*  This memory can store up to 32 kbytes.  When a signal triggers the board, */
/*  the program calls the attenuator subroutine, which sets the attenuator    */
/*  such that the signal is in the dynamic range of the board.  Each time the */
/*  signal is in the dynamic range the board, it is plotted to the screen.    */
/*  The user then has 3 options:  1)  continue to calculations, 2)  shift     */
/*  the signal, and 3)  change the trigger level.  Shifting the signal and    */
/*  changing the trigger level keeps the program in the "while" loop which    */
/*  looks for a signal.  Continuing to the calculations breaks out of the     */
/*  "while" loop and continues through the program.  The signal is stored as  */
/*  arrays, x and y.  Then, the signal is changed to its actual voltage       */
/*  values.  This program then calls the subroutine "dtft" to calculate the   */
/*  center frequency.  Then the subroutine "marc" is called to determine the  */
/*  sensitivity of the hydrophone given the center frequency.  The signal is  */
/*  then changed to the proportional pressure signal when "pressure" is       */
/*  called.  A separate program to calculate the pulse repetition period is   */
/*  called.  From the pressure signal, the intensities are calculated with    */
/*  "int_time_ave".  Finally, all calculated values are printed to the screen */
/*  and stored in a file when "done" is called.                               */
/*****************************************************************************/

#include <graphms.h>
#include <graph.h>
#include <dos.h>
#include <stdlib.h>
#include <stdio.h>
#include "nr.h"
#include "nrutil.h"
#include <math.h>
#include <process.h>
#include <conio.h>
#include <string.h>
#include <time.h>

/*-------------------------- WAAG2 locations -------------------------------*/

#define    WAAG     0xD0000000  /* WAAG memory */
#define    PORT0    0x178       /* default setting      */
#define    PORT1    0x179       /* all switches are off */
#define    PORT2    0x17A
#define    PORT3    0x17B

#define LENGTH 1024        /* Echo Length */
#define COUNT -1279        /* Value to load WAAG counter is -(LENGTH + 0xFF ) */
#define CONTROL 0xA00C     /* Set acquisition at 40 Mhz */

/*------------------------ PDMA 16 addresses -------------------------------*/

#define PDMA_A 0x300           /* address of port a */
#define PDMA_B 0x301           /* address of bort b */

#define PDMA_DMA 0x302         /* DMA control register */
#define PDMA_INT 0x303         /* Interrupt control register */
#define STP _settextposition   /* moves curser to a position on the screen */
```

```
/*---------------------- calculation constants ----------------------------*/

#define CLS     "\x1B[2J"          /*  clear screen              */
#define PI      3.141592654
#define SOS     1481               /*  speed of sound in water   */
#define CEL     4.8                /*  elect. load cap. for Marconi amplifier = 4.8 pF +/- 10%  */
#define RHO     998                /*  density of water          */
#define Zo      1.48e6             /*  char.impedance in water (Pa*sec/m)  */

FILE *fptr,                        /*  pointer to file           */
     *output;                      /*  pointer to output file    */

/*---------------- ports for the programmable attenuator -------------------*/

int     port_a = 0x308,       /* address of port a   */
        port_b = 0x309,       /* address of port b   */
        port_c = 0x30A        /* address of port c   */
        ;

/*-------- variables for reading and storing the signal from the waag --------*/

unsigned char  far     *waag          /* pointer to WAAG memory       */
                       ;

int     trigger,              /*  the trigger address on the waag                    */
        shift = 0,            /*  a value added to the address to reading the board  */
        atten,                /*  attenuation value of the attenuator                */
        control = 0xA00c      /*  control parameters for WAAG board -- set acquisition at 40 MHz  */
        ;

char
     trigger_level = 0x80,          /* trigger level                    */
     txt[80],                       /* dummy for printing text to screen   */
     filename[12]                   /* dummy for printing a file name      */
     ;

float   x[1024],              /*  current values of x axis values corresponding to y array    */
        y[1025],              /*  current values of y axis values (signal), i.e.  amplitude   */
        freq_axis[1024],      /*  values of the frequency axis of original signal             */
        echo[1024],           /*  original voltage signal from the WAAG II                    */
        intenwm[1024],        /*  signal converted to intensity values                        */
        cent_freq,            /*  center frequency                                            */
        bandwidth,            /*  bandwidth of signal                                         */
        ml,                   /*  end-of-cable sensitivity for the Marconi 0.5 mm bilaminar   */
        pd,                   /*  pulse duration                                              */
        dutycyc,              /*  duty cycle                                                  */
        pc,                   /*  peak compressional pressure                                 */
        pr,                   /*  peak rarefractional pressure                                */
        piimax,               /*  maximum value from the pulse intensity integral             */
        ita,                  /*  intensity -- temporal average                               */
        ipa,                  /*  intensity -- pulse average                                  */
        itp                   /*  intensity -- temporal peak                                  */
        ;

double  real[512],                 /* real values of dtft of y array       */
        img[512],                  /* imaginary values of dtft of y array  */
        temp,                      /* dummy variable                       */
        prp                        /* pulse repetition period      */
        ;

/*----------------------------- subroutines --------------------------------*/

void absolute_max(),    /* sub to calculate absolute value maximum value in y array */
     dtft(),            /* sub to calculate center frequency and bandwidth      */
     press(),           /* sub to calculate pressure values in y array          */
     caldata(),         /* sub to determine end-of-cable sensitivity -- ml      */
     int_time_ave(),    /* sub to calculate intensity values in y array         */
     done()             /* sub to print data on screen and in file              */
     ;

int  sample(),          /* sub to arm WAAG and wait for trigger after attenuator is set  */
```

```
     attenuation(),     /* sub to set programmable attenuator          */
     atten_sample()     /* sub to arm WAAG and wait for trigger while setting attenuator */
     ;

float findmax(),        /* sub to calculate maximum value in y array   */
      findmin(),        /* sub to calculate minimum value in y array   */
      marc()            /* sub to calculate end-of-cable sensitivity   */
      ;

double dprp();          /* sub to calculate pulse repetition period    */



/*============================== MAIN PROGRAM ================================*/


main()
{
     unsigned char far
                      *index          /* waag + c + trigger     */
                      ;

     char
          dummy,      /* dummy variable                                  */
          key,        /* character when keyboard is hit                  */
          txt[50],    /* string of character to be printed to screen     */
          blank[50]   /* string of blanks to be printed to screen to "erase" text  */
          ;

     int
          c,          /* dummy counter  */
          i,
          counter,
          cont = 1
          ;

     void
         load_trigger_offset()        /* loads trigger level */
         ;


  int
      g_driver,                      /* Turbo-C graphics initialization */
      g_mode,                        /* variables                       */
      g_error
      ;


     _setvideomode(_VRES16COLOR);              /* set video mode to graphics   */
     _clearscreen( _GCLEARSCREEN );


     atten = 60;                               /* set attenuator to 60 dB      */
     outp(port_a, atten);

     waag = (char far *) WAAG;


     /*------------- draw signal clipping limits and menu --------------------*/
     _setcolor(9);
     _moveto(0,99);
     _lineto(600,99);
     _moveto(0, 100+256);
     _lineto(600,100+256);

     STP(1,24);                                  /* print title on screen */
     sprintf(txt," 40 Mhz RF Acquisition    \n" );
     printf("%s", txt);

     STP(4,5);                                             /* print menu */
```

```
strcpy(txt, "OPTIONS:");
printf("%s", txt);

STP(5,5);
strcpy(txt, "[c]  continue to do calculations");
printf("%s", txt);

STP(6,5);
strcpy(txt, "[s]  shift echo (+ or -)");
printf("%s", txt);

STP(5,50);
strcpy(txt, "[t]  set trigger level");
printf("%s", txt);

STP(6,50);
strcpy(txt, "[q]  quit");
printf("%s", txt);

STP(24,5);                    /* print attenuation value in lower left */
strcpy(txt,"Attenuation is set at ");
printf("%s", txt);

STP(24,50);           /* print relative voltage values in lower right */
strcpy(txt, "Maximum voltage is");
printf("%s", txt);

STP(25,50);
strcpy(txt, "Minimum voltage is");
printf("%s", txt);


load_trigger_offset (trigger_level);       /* set trigger level at 0x80 */


while( cont )   /* plot until button pushed then acquire data */

  {
   attenuation();                          /* set programmable attenuator */
   trigger = sample();                     /* data acquisition */
   trigger <<= 1;                          /* address = trigger*2  */
   STP(1,1);


   _setcolor(13);                                    /* plot RF echo */
   c=0;
   index = (char far *)(waag + c + trigger + shift);
   _moveto( c, (int)*( index ) + 100 );

   for (c=1; c<600; c++)
       {
        index = (char far *)(waag + c + trigger + shift);
       _lineto( c, (int)*( index ) + 100 );
       }


   STP(24,28);                            /*  print attenuation value  */
   printf("%2d dB", atten);

   STP(24,69);                            /*  print relative voltages  */
   printf("%f", findmax(&counter,1024));

   STP(25,69);
   printf("%f", findmin(&counter,1024));


   _setcolor(0);                                    /* erase echo */
   c=0;
   index = (char far *)(waag + c + trigger + shift);
   _moveto( c, (int)*( index ) + 100 );
   for (c=1; c<600; c++)
```

```
    {
     index = (char far *)(waag + c + trigger + shift);
     _lineto( c, (int)*( index ) + 100 );
    }


 if( kbhit() )          /*  options menu if keyboard is hit  */
  {
    key = getch();

    switch (key)
       {
        /*------------------ change trigger level ---------------------*/
        case 't':
            STP(24,5);
            strcpy(txt,"Enter trigger level (0-255): ");
            printf("%s", txt);
            scanf("%d", &dummy );
            trigger_level = (char)dummy;
            load_trigger_offset( trigger_level );
            STP(25,5);
            sprintf(txt,"Trigger level is now --> %d", (int)trigger_level );
            printf("%s", txt);
            STP(24,5);
            strcpy(blank,"                                        ");
            printf("%s", blank);
            STP(25,5);
            printf("%s", blank);
            STP(24,5);
            strcpy(txt,"Attenuation is set at ");
            printf("%s", txt);
            break;

        /*--------------------------- shift echo ---------------------*/
        case 's':
            STP(25,5);
            strcpy(txt,"Enter shift (+ or -):  ");
            printf("%s", txt);
            scanf("%d", &shift );
            STP(25,5);
            strcpy(blank,"                                  ");
            printf("%s", blank);
            break;

        /*-------------- exit while loop and continue program ---------*/
        case 'c':
            cont = 0;
            printf(CLS);
            _setvideomode(_TEXTC80);
            break;

        /*-------------------------- exit program --------------------*/
        case 'q':
            _setvideomode(_TEXTC80);
            exit(0);

        /*----------------------- other keys:  beep ------------------*/
        default:
            printf("%c",7);

       } /* switch statement     */
  }      /* if statement         */

} /* main while loop */


trigger = sample();                     /* data acquisition     */
trigger <<= 1;                          /* address = trigger*2  */

for (c=0; c<1024; c++)                  /* store original signal */
   {
```

```
        echo[c] = (float)( (int)*(waag + c + trigger + shift) - 129 ) * 0.00502353;
        x[c] = c;
       }

    for (c=0; c<1024; c++)          /* change relative voltage value to real voltage value */
       {
        echo[c] /= (float)pow( 10.0,(double)(44-atten)/20.0 );
        y[c+1] = echo[c];                   /* array shift for realft() in dtft()   */
       }

    dtft();                         /* calculate center frequency and bandwidth     */

    mi = marc(&cent_freq);          /* calculate end-of-cable sensitivity           */

    press(mi);                      /* calculate pressure signal                    */

    counter = 0;
    pc = findmin(&counter,1024);    /* minimum value of y is peak compressional pressure     */
    counter = 0;
    pr = findmax(&counter,1024);    /* maximum value of y is peak rarefractional pressure    */

    prp = dprp();                   /* calculate pulse repetition period            */

    int_time_ave();                 /* calculate intensities                        */

    done();                         /* print values to screen and store in file     */

    _setvideomode(_TEXTC80);        /* set video to color text mode                 */

}
```

APPENDIX C

SAMPLE SUBROUTINE

```
/*****************************************************************************/
/*  Subroutine "sample" initializes and arms the waag board to store a       */
/*  signal.  Once armed the while loop continues to wait for a signal until  */
/*  the trigger bit is set.  Because the board may be initialized at any     */
/*  time during or between signals, it is necessary to ensure the entire     */
/*  signal is stored.  So, the waag board is then immediately initialized and */
/*  armed again to wait for another signal.  This synchronizes the acquiring */
/*  of the new signal with the past signal, ensure that there is enough      */
/*  presampled data and that the entire signal is stored.                    */
/*****************************************************************************/

int  sample()
 {
    int
        busy = 1,                       /*  while loop flag     */
        count
        ;

    char
         key
         ;

    outpw(PORT2,0xA06f);                /* initialize */
    outpw(PORT0,0xffff);                /* clear counter */
    outpw(PORT0,0xffff);
    outpw(PORT0,COUNT);                 /* load count */
    outpw(PORT0,COUNT);
    outpw(PORT2,CONTROL & 0xbffc);      /* start the counter */


    while (busy)                        /* wait for sampling to be completed */
      {
        if(inp(PORT2) & 01) busy = 0;   /* sampling is done */
      }

    /* look for trigger again --                                          */
    /* this synchronizes the presample data with the arming of the board */

    outpw(PORT2,0xA06f);                /* initialize */
    outpw(PORT0,0xffff);                /* clear counter */
    outpw(PORT0,0xffff);
    outpw(PORT0,COUNT);                 /* load count */
    outpw(PORT0,COUNT);
    outpw(PORT2,CONTROL & 0xbffc);      /* start the counter */

    busy = 1;
    while (busy)                        /* wait for sampling to be completed */
      {
        if(inp(PORT2) & 01) busy = 0;   /* sampling is done */
      }
    outpw(PORT2,0xA06f);                /* enabled WAAGII ram */
    return(inpw(PORT0));                /* read trigger address */
 }
```

APPENDIX D

ATTENUATION SUBROUTINE

```
/*****************************************************************************/
/*  Subroutine "attenuation" sets the WaveTek P1506 programmable attenuator.  */
/*  In the main program, the attenuator is initially set at 60 dB.  The       */
/*  attenuator is set each time the signal is plotted to the screen (each     */
/*  pass of the main while loop).  The attenuator is set so the signal is     */
/*  in the dynamic range of the board, between 0.5 V and 0.6 V.  There are    */
/*  3 major sections of this program, decrementing and incrementing by        */
/*  10, 5 and 1 dB.  Different windows of voltage are used to determine       */
/*  which decrementing and incrementing values should be used and to make     */
/*  the setting of the attenuator more efficient.                             */
/*****************************************************************************/

int attenuation()
{

  int int_control = 0x308;

  int   not_dynamic_range,
        counter,
        c
        ;

  double abs_max,       /* absolute maximum -- largest positive peak    */
         abs_min,       /* absolute minimum -- largest negative peak    */
         min,
         max,
         limit_indicator,
         check
         ;

  outp ( int_control, 0xff );

  not_dynamic_range = 1;        /* not within the dynamic range */
  counter = 0;

  trigger = atten_sample();     /* get signal   */
  trigger <<= 1;
  STP(24,28);
  printf("%2d dB", atten);

  for (c=0; c<1024; c++)                /* store signal in y array      */
       y[c] = (float)((int)*(waag + c + trigger + shift) -129) * 0.00502353;

/************************** 10 dB Increments ********************************/

  while ( not_dynamic_range )           /* while the attenuator is not set */
        {
         STP(24,28);
         printf("%2d", atten);
         counter++;
         absolute_max(&min, &max, 1024);
         abs_min = fabs(min);   /* take absolute value of the minimum value  */
         abs_max = fabs(max);   /* take absolute value of the maximum value  */
         limit_indicator = (float)max(abs_min, abs_max);        /* find larger peak     */
         check = fabs(limit_indicator - 0.6);   /* how far away is the value from 0.6 V  */

      if( (check > 0.4) && (atten > 0) && (counter != 8) )
        {                                               /* if the signal is 0.4 V or smaller plus      */
```

```
        if(limit_indicator <= 0.2)                    /* the attenuator is not set at 0 dB and this  */
            {                                         /* is not the 8th pass, then....               */
             outp(port_a, atten -= 10);
             trigger = atten_sample();                /* if the signal is 0.2 V or smaller, decrease */
             trigger <<= 1;                           /* the attenuation by 10 dB                    */

             for (c=0; c<1024; c++)
                  y[c] = (float)((int)*(waag + c + trigger + shift) -129) * 0.00502353;
            }
        else if (limit_indicator > 0.6)               /* if the signal is larger than 0.6 V, then... */
                  if (atten >= 54)                    /* if the attenuator is set at 54 dB or larger */
                       not_dynamic_range = 0;         /* then the dynamic range is set and jump out  */
                  else                                /* of while loop.  Else, increase attenuation  */
                      {                               /* by 10 dB.                                   */
                       outp(port_a, atten += 10);
                       trigger = atten_sample();
                       trigger <<= 1;

                       for (c=0; c<1024; c++)
                            y[c] = (float)((int)*(waag + c + trigger + shift) -129) * 0.00502353;
                      }
       }
      else if (atten <= 0)                            /* if the attenuator is less than or equal to  */
             {                                        /* 0 dB, then set the attenuator to 0 dB       */
              atten = 0;
              outp(port_a, atten);
              if (counter == 8)                       /* if the counter is 8, then jump out of       */
                  {                                   /* while loop                                  */
                   printf("\nERROR!!  Signal too small");
                   not_dynamic_range = 0;
                  }
             }
        else
           not_dynamic_range = 0;            /* check < 0.4 */
       }                                     /* while loop */

  not_dynamic_range = 1;                /* reinitialize flag and counter        */
  counter = 0;

/***************************** 5 dB Increments ********************************/

  while (not_dynamic_range)
        {
         STP(24,28);
         printf("%2d dB", atten);
         counter++;
         absolute_max(&min, &max, 1024);
         abs_min = fabs(min);            /* take absolute value of the minimum value  */
         abs_max = fabs(max);            /* take absolute value of the maximum value  */
         limit_indicator = (float)max(abs_min, abs_max);        /* find larger peak     */
         check = fabs(limit_indicator - 0.6);           /* windowing    */

      if( (check > 0.2) && (atten > 0) && (counter != 13) )
        {                                             /* if window is greater than 0.2 V, and        */
         if(limit_indicator <= 0.4)                   /* attenuation is greater than 0 dB, and it has */
             {                                        /* not passed through 13 times, then....       */
              outp(port_a, atten -= 5);
              trigger = atten_sample();               /* if signal is smaller than 0.4 V, then       */
              trigger <<= 1;                          /* decrease attenuation by 5 dB                */

              for (c=0; c<1024; c++)
                   y[c] = (float)((int)*(waag + c + trigger + shift) -129) * 0.00502353;
             }
         else if (limit_indicator > 0.6)              /* if signal is larger than 0.6 V and          */
                   if (atten >= 59)                   /* attenuator is 59 dB or larger, then jump out */
                        not_dynamic_range = 0;        /* of while loop                               */
                   else                               /* otherwise, attenuator is less than 59 dB so  */
                        {                             /* add 5 dB                                    */
                         outp (port_a, atten += 5);
                         trigger = atten_sample();
                         trigger <<= 1;
```

```
                    for (c=0; c<1024; c++)
                         y[c] = (float)((int)*(waag + c + trigger + shift) -129) * 0.00502353;
                   }
      }
     else if (atten <= 0)                    /* so attenuation is less than or equal to 0 dB */
              {                              /* set attenuation to 0 dB                      */
               atten = 0;
               outp(port_a, atten);
               if (counter == 13)            /* check for number of passes;  if 13, then     */
                   {                         /* jump out of while loop                       */
                    not_dynamic_range = 0;
                    printf ("\nERROR!!  Cannot change by 5 dB");
                   }
              }
         else
              not_dynamic_range = 0;         /* check < 0.2 */
      }                                      /* while loop */

  not_dynamic_range = 1;                     /* reinitialize flag and counter        */
  counter = 0;

/***************************** 1 dB Increments ******************************/

  while (not_dynamic_range)
        {
         STP(24,28);
         printf("%2d dB", atten);
         counter++;
         absolute_max(&min, &max, 1024);
         abs_min = fabs(min);                /* get min and max values       */
         abs_max = fabs(max);
         limit_indicator = (float)max(abs_min, abs_max);
         check = fabs(limit_indicator - 0.6);          /* windowing    */

      if ( (limit_indicator > 0.5) && (limit_indicator < 0.6) )      /* if signal is between 0.5 and 0.6 V,
*/
        not_dynamic_range = 0;                                       /* then stop while loop on next pass
*/

      if( (check > 0.01) && (atten >= 0) && (counter != 65) )
        {                                          /* if signal is not within 0.01 of 0.6 V  */
         if(limit_indicator < 0.5)                 /* and attenuation is larger than 0 dB    */
             {                                     /* and 65 passes have not occurred, then..*/
              outp(port_a, atten -= 1);            /* if signal is lower than 0.5 V, then    */
              trigger = atten_sample();            /* decrease attenuation by 1 dB           */
              trigger <<= 1;

              for (c=0; c<1024; c++)
                    y[c] = (float)((int)*(waag + c + trigger + shift) -129) * 0.00502353;
             }
         else if (limit_indicator > 0.6)           /* if signal is greater than 0.6 V and if */
                   if (atten >= 63)                /* attenuator is already 63 dB, then set  */
                        {                          /* attenuator to 63 dB, print message to  */
                         atten = 63;               /* screen, and exit main program          */
                         outp(port_a, atten);
                         printf ("\nERROR!!  Signal too large");
                         exit(0);
                        }
                   else                           /* otherwise increase attenuator by 1 dB  */
                        {
                         outp (port_a, atten += 1);
                         trigger = atten_sample();
                         trigger <<= 1;

                         for (c=0; c<1024; c++)
                              y[c] = (float)((int)*(waag + c + trigger + shift) -129) * 0.00502353;
                        }
        }
      else if ( (atten <= 0) && (not_dynamic_range == 1) )
              {                                    /* if attenuator is already 0 dB or less and not */
               atten = 0;                          /* in dynamic range, then set attenuator to 0 dB*/
```

```
                outp(port_a, atten);                    /* if 65 passes, then print message to screen   */
                if (counter == 65)                      /* exit main program                            */
                    {
                     printf("\nERROR!!  Signal too small");
                     exit(0);
                    }
               }
           else if (atten >= 63)                        /* if attenuation is larger than 63 dB, then    */
                    {                                   /* set attenuation to 63 dB and exit main program */
                     atten = 63;
                     outp(port_a, atten);
                     printf("\nERROR!!  Signal too big");
                     exit(0);
                    }
                else
                     not_dynamic_range = 0;          /* check < 0.01 */

        }       /* while loop */

  return(0);

}
```

APPENDIX E

ATTEN_SAMPLE SUBROUTINE

```
/*****************************************************************************/
/*  Subroutine "atten_sample" is essentially the same as subroutine "sample." */
/*  It is called from subroutine "attenuation" and is only used while setting */
/*  the programmable attenuator.  This keeps the program from hanging in an   */
/*  infinite while loop (i.e. if subroutine "sample was called in Appendix C).*/
/*  It initializes and arms the waag board and waits 2 seconds for a signal.  */
/*  If there is no signal and 2 seconds have passed, the attenuator is        */
/*  decreased by 10 dB.  If the attenuator is already 0 dB, a message that    */
/*  the signal is too small is printed to the screen.  The subroutine will    */
/*  continue to wait for a signal (again 2 seconds) and printing the message  */
/*  until a large enough signal is found or when the "q" key is hit.  Only    */
/*  when "q" is hit will the program stop waiting for a signal and exit the   */
/*  main program.  All other keys will only cause the computer to beep.       */
/*****************************************************************************/

int  atten_sample()
 {
    int
        busy = 1,                       /* while loop flag */
        not_data_acquired = 1,          /* while loop flag */
        count
        ;

    time_t    start_time,               /* time markers */
              current_time
              ;

    long      elapsed_time = 0
              ;

    char
         key,
         blank[30]
         ;

  while (not_data_acquired)
  {
    outpw(PORT2,0xA06f);                /* initialize */
    outpw(PORT0,0xffff);                /* clear counter */
    outpw(PORT0,0xffff);
    outpw(PORT0,COUNT);                 /* load count */
    outpw(PORT0,COUNT);
    outpw(PORT2,CONTROL & 0xbffc);      /* start the counter */

    time(&start_time);
    elapsed_time = 0;
    busy = 1;

    while (elapsed_time < 2)            /* wait for sampling to be completed */
      {
        time(&current_time);
        elapsed_time = current_time - start_time;       /* calculate elapsed time */
        if(inp(PORT2) & 01)
             {
              busy = 0;                 /* signal exists:  sampling is done */
              elapsed_time = 5;
             }
      }
```

```
    if ( (busy == 1) && (atten > 0) )          /*  if there is no signal and the programmable  */
        {                                      /*  attenuator is set about 0 dB, decrease the  */
         atten -= 10;                          /*  attenuator by 10 dB                         */
         outp(port_a, atten);
         STP(24,28);
         printf("%2d dB", atten);
        }
    else if ( (busy == 1) && (atten <= 0) )    /*  if there is no signal and the programmable  */
        {                                      /*  attenuator is less than or equal to 0 dB,   */
         atten = 0;                            /*  set attenuator to 0 dB and print message    */
         outp(port_a, atten);

         STP(12,35);
         strcpy(txt, "Signal too small");
         printf("%s", txt);
         STP(13,35);
         strcpy(txt, "Waiting for larger signal");
         printf("%s", txt);

         if ( kbhit() )                        /*  if "q" is hit on keyboard, exit program     */
              {                                /*  otherwise, beep                             */
                key = getch();
                if (key == 'q')
                       {
                        _setvideomode(_TEXTC80);
                        exit(0);
                       }
                else
                        printf("%c",7);
              }
         else printf("%c", 7);


         STP(12,35);                           /*  erase text that has been written            */
         strcpy(blank,"                        ");
         printf("%s",blank);
         STP(13,35);
         printf("%s",blank);

         not_data_acquired = 1;          /*  no data acquired  */
        }
    else
        not_data_acquired = 0;          /*  busy = 0 so proceed to acquire echo */

  }


  outpw(PORT2,0xA06f);                   /* enabled WAAGII ram */
  return(inpw(PORT0));                   /* read trigger address */
 }
}
```

APPENDIX F

PULSE REPETITION PERIOD SUBROUTINES

```
#include <MATH.H>
#include <TIME.H>
#include <CONIO.H>

#define RD_USEC 0x302          /* memory location of clock's latched value   */

/*****************************************************************************/
/* Subroutine "dprp" returns the pulse repetition period value.  It works in  */
/* conjunction with a signal from an inductor coil going to a home designed   */
/* board.  "Dprp" calls subroutine "read_usec", which actually stores the     */
/* clock values from the board to an array.  The array is scanned for the     */
/* first change in values.  The first value is subtracted from the second     */
/* value.  This value is multiplied by 1 microsecond because the clock runs   */
/* at 1 MHz.  This, then, is the pulse repetition period.                     */
/*****************************************************************************/

double dprp()
{

int     read_usec(),           /* subroutine to read clock values      */
        count1,                /* first clock value                    */
        count2,                /* second clock value                   */
        change,                /* flag marking clock values changed    */
        c                      /* counter                              */
        ;

unsigned int     usec_array[300];       /* array storing clock values   */

double  period,                /* pulse repetition period              */
        frequency              /* pulse repetition frequency           */
        ;

        for (c=0; c<300; c++)                   /* clear array  */
               usec_array[c] = 0;

        read_usec(usec_array);

        for (c=0; c<300; c++)          /* search for first change in values */
               {
               if (c == 0)
                  {
                  count1 = abs(usec_array[c]);         /* first value */
                  change = 0;
                  }
               if ( (count1 != abs(usec_array[c])) && (change == 0) )
                  {
                  count2 = abs(usec_array[c]);         /* second value */
                  change = 1;
                  }
               }

        period = (double)abs(count2-count1) * 1.0e-6;
        frequency = 1.0/period;

        return(period);
}
```

```
/*****************************************************************************/
/*  Subroutine "read_usec" stores values from RD_USEC into an array.         */
/*  Returns 0 is no signal; 1 if signal present.                             */
/*                                                                           */
/*  RD_USEC - address of millisecond latch.    Whenever a signal pulse       */
/*            arrives, the value of a free-running 1 MHz counter is          */
/*            latched into this latch, and the New Data latch is set.        */
/*            Software then reads RD_USEC.  Consecutive values from          */
/*            RD_USEC will give the period in microseconds of the            */
/*            incoming transducer pulses, from which the pulse repetition    */
/*            period can be calculated.                                      */
/*****************************************************************************/

int read_usec( usec_array )

unsigned int *usec_array;

{

   int     c;

   unsigned int  newdata;

        for (c=0; c<300; c++)
            usec_array[c]  = inpw( RD_USEC );


 return(1);

}
```

APPENDIX G

DTFT SUBROUTINE

```
/*****************************************************************************/
/* Subroutine "dtft" ultimately calculates the center frequency and bandwidth.*/
/* It uses the subroutine "realft" from Numerical Recipes in C to perform    */
/* a discrete time Fourier transform on the y array.  "Realft" stores the    */
/* dtft in the y array, alternating real and imaginary components.  The      */
/* magnitude of the signal in frequency domain is then calculated and        */
/* restored in the y array.  The maximum is then determined and thus the 3 dB */
/* points are determined.  The corresponding frequency to each element in the */
/* y array is calculated and stored in the array freq_axis.  From the low    */
/* frequency 3 dB point to the high frequency 3 dB point is the bandwidth.   */
/* One-half of the bandwidth is the center frequency.                        */
/*****************************************************************************/

void dtft()


{

int     wavehex,                        /* a character type integer     */
        i,                              /* index                        */
        j,                              /* index of returned realft array       */
        c,
        nxdiv,
        nydiv,
        npts,
        dex,                            /* max array numeral            */
        posdb,                          /* index for 3dB pt at high frequency   */
        mindb                           /* index for 3dB pt at low frequency    */
        ;

char    title[30],
        key
        ;

float   maxy,
        desired_freq,
        f1,                             /* 3dB pt. at lower frequency   */
        f2,                             /* 3dB pt. at higher frequency  */
        bw                              /* bandwidth                    */
        ;



void realft();

float findmax();

strcpy(title,"FREQUENCY RESPONSE");

        printf(CLS);
        printf(" Calculates the center frequency\n");

        for (c=0; c<1024; c++)
                freq_axis[c] = ( (float)c )*( 1e-6/(1024*2.5e-8) );

        realft(y,512,1);        /* canned routine from "Numerical Recipes in C" to do dtft */
```

```
        j=1;
        for(i=0;i<512;i++)          /* insert real #'s real array  */
                {
                real[i]=(double)y[j];
                j=j+2;
                }
        j=2;
        for(i=0;i<512;i++)          /* insert img #'s img array  */
                {
                img[i]=(double)y[j];
                j=j+2;
                }
        for(i=0;i<512;i++)          /* calculate magnitude in frequency domain */
                {
                temp=img[i]*img[i]+real[i]*real[i];
                y[i]=sqrt(temp);
                }

/************************ find max and 3db points ****************************/

        dex = 0;
        maxy = findmax(&dex,512);       /* find maximum in y array      */

        printf("\n max-y[%d]= %f",dex,y[dex]);
        printf("  max-freq_axis[%d]= %f",dex,freq_axis[dex]);

        i = dex;
        while ( (y[i] > 0.707*maxy) && (i < 512) )      /* 3 dB point   */
                i++;
        posdb = i;              /* index of high frequency 3 dB point   */

         printf("\n plus 3db : y[%d]=%f",posdb,y[posdb]);
         printf("   freq_axis[%d]=%f  ",posdb,freq_axis[posdb]);
         f2=freq_axis[posdb];

         i = dex;
         while ( (y[i] > 0.707*maxy) && (i > 0) )       /* 3 dB point   */
                i--;
         mindb = i;             /* index of low frequency 3 dB point    */

         printf("\n  neg 3db : y[%d]=%f",mindb,y[mindb]);
         printf("   freq_axis[%d]=%f  ",mindb,freq_axis[mindb]);
         f1=freq_axis[mindb];

         cent_freq = (f2+f1)/2;                 /* center frequency         */
         bandwidth = f2-f1;                     /* bandwidth                */

        printf("\n\n Center Frequency = %f Mhz ",cent_freq);
        printf("\n Bandwidth        =  %f Mhz ",bandwidth);

        printf("\n\n Do you wish to plot the frequency response?? [n]  ");
        key = getch();
        if (key == 'y' || key == 'Y')
       {
        printf("\n\n Enter highest frequency to plot  ");
        printf("\n (Range of 0 to 20 MHz) [return]: ");
        scanf("%f",&desired_freq);
        npts = (int)(512.0*desired_freq/20.0);

/************************* DTFT Plotting part ********************************/
/***************** From Scientific Endeavors Corp. "GraphiC 4.1" **************/

        bgnplot(1,'g',"dtft.plt");
        startplot(1);
        font(3,"simplex.fnt",'\310',"triplex.fnt",'\311',"complex.fnt",'\312',"",'2');
        page(9.0,6.855);
        area2d(8.0,5.5);
        grid(2);
        upright(1);
        nxdiv=11;
        nydiv=11;
```

```
        color(10);
        scales(nxdiv,nydiv,freq_axis,y,npts);
        color(10);
        xname("Frequency (MHz)");
        heading(title);
        yname("magn. of F(exp(jw))");
        color(14);
        curve(freq_axis,y,npts,0);
        endplot();
        stopplot();
      }

}
```

APPENDIX H

HYDROPHONE SENSITIVITY SUBROUTINES

```
/*****************************************************************************/
/* Subroutine "marc" returns the end-of-cable loaded sensitivity value (ml).  */
/* Because Marconi only lists its calibration in discrete integer frequencies,*/
/* a linear regression is performed for frequencies between 2 integer         */
/* frequencies to determine appropriate end-of-cable loaded sensitivity       */
/* values.  Subroutine "caldata" is called to obtain all necessary variables  */
/* to calculate the end-of-cable loaded sensitivity for integer frequencies.  */
/* If no value is determined, the regression is used.                         */
/*****************************************************************************/

float marc(center_frequency)

float *center_frequency;
{

float *fc;

float
      round_up,      /* for preforming a linear regression          */
      slope_mc,
      b_mc,
      slope_rez,
      b_rez,
      slope_imz,
      b_imz,
      fcx[3],
      mcy[3],
      rezy[3],
      imzy[3],
      freq,
      freqhz,
      ml,            /* End-of-cable loaded sensitivity of hydrophone */
      mc,            /* end-of-cable open circuit-given (uV/Pa)       */
      rez,           /* Re(Z); real part impedance in ohms            */
      imz,           /* Im(Z); imaginary part impedance in ohms       */
      cap            /* C: end-of-cable capac. of the hydrophone      */
                     /*  combined C of the cable and scope            */
      ;


       *fc = *center_frequency;
       freqhz=*fc*1e6;

       /*  printf("\n\n mod ans = %f ",fmod(*fc,1) );   */
       /*  if not a integer,  do a regression         */

       if (fmod((double)*fc,1.0) == 0.0 )
          {                                  /* calibration variables determined for integer frequencies
*/
          caldata(fc, &mc, &rez, &imz);
          printf("\n\n   Direct from NPL Table" );
          }

       else            /* otherwise frequency is no an integer, so do regression      */

          {
          round_up=*fc;               /* find the values for the nearest low integer frequency       */
          freq=*fc;
```

```
            fcx[0]=(float)floor((double)*fc);
            *fc=(float)floor((double)*fc);
            caldata(fc,&mc,&rez,&imz);
            mcy[0]=mc;
            rezy[0]=rez;
            imzy[0]=imz;


            fcx[1]=(float)ceil((double)round_up); /* find the values for the nearest high integer frequency
*/
            *fc=(float)ceil((double)round_up);
            caldata(fc,&mc,&rez,&imz);
            mcy[1]=mc;
            rezy[1]=rez;
            imzy[1]=imz;

            slope_mc=(mcy[0]-mcy[1])/(fcx[0]-fcx[1]);           /* determine slope      */
            slope_rez=(rezy[0]-rezy[1])/(fcx[0]-fcx[1]);
            slope_imz=(imzy[0]-imzy[1])/(fcx[0]-fcx[1]);

            b_mc=(fcx[0]*mcy[1]-fcx[1]*mcy[0])/(fcx[0]-fcx[1]);
            b_rez=(fcx[0]*rezy[1]-fcx[1]*rezy[0])/(fcx[0]-fcx[1]);
            b_imz=(fcx[0]*imzy[1]-fcx[1]*imzy[0])/(fcx[0]-fcx[1]);

            mc=slope_mc*freq+b_mc;                      /* do regression        */
            rez=slope_rez*freq+b_rez;
            imz=slope_imz*freq+b_imz;


            }

        cap=(-1e12)/(2*PI*(freqhz)*(imz));      /*  capacitance   */
        ml=mc*(cap/(cap+CEL));                  /*  sensitivity */

        return (ml);

}
```

```
/*****************************************************************************/
/* Subroutine "caldata"is called from "marc".  It is a listing of the        */
/* calibration variables determined by Marconi for the Marconi for the 0.5 mm */
/* bilaminar hydrophone (serial no. IP041) for integer frequencies           */
/*****************************************************************************/

void caldata(fc, mc, rez, imz)

           /* all measurements taken in water at 20.7 +/- 0.5 degree C */
float   *fc,            /* center frequency                                */
        *mc,            /* end-of-cable open-circuit sensitivity (uV/Pa) */
        *rez,           /* impedance:  real component (ohms)               */
        *imz            /* impedance:  imaginary component (ohms)          */
        ;

{
         if ( *fc >= 0.99 && *fc <=  1.01 )
                {  *mc=0.060;
                   *rez=140.0;
                   *imz=-1910.0;        }
         else if ( *fc >= 1.99 && *fc <=  2.01 )
                {  *mc=0.060;
                   *rez=80.0;
                   *imz=-1000.0;         }
         else if (*fc >= 2.99 && *fc <=  3.01 )
                {  *mc=0.062;
                   *rez=60.0;
                   *imz=-680.0;         }
         else if ( *fc >= 3.99 && *fc <=  4.01 )
                {  *mc=0.063;
                   *rez=44.0;
                   *imz=-520.0;         }
         else if ( *fc >= 4.99 && *fc <=  5.01 )
                {  *mc=0.066;
                   *rez=34.0;
                   *imz=-419.0;         }
         else if ( *fc >= 5.99 && *fc <=  6.01 )
                {  *mc=0.067;
                   *rez=29.0;
                   *imz=-351.0;         }
         else if ( *fc >= 6.99 && *fc <=  7.01 )
                {  *mc=0.068;
                   *rez=26.0;
                   *imz=-302.0;         }
         else if ( *fc >= 7.99 && *fc <=  8.01 )
                {  *mc=0.071;
                   *rez=23.0;
                   *imz=-264.0;         }
         else if ( *fc >= 8.99 && *fc <=  9.01 )
                {  *mc=0.073;
                   *rez=21.0;
                   *imz=-235.0;         }
         else if ( *fc >= 9.99 && *fc <=  10.01 )
                {  *mc=0.077;
                   *rez=19.0;
                   *imz=-211.0;         }
         else if ( *fc >= 10.99 && *fc <=  11.01 )
                {  *mc=0.079;
                   *rez=18.0;
                   *imz=-191.0;         }
         else if ( *fc >= 11.99 && *fc <=  12.01 )
                {  *mc=0.084;
                   *rez=17.0;
                   *imz=-174.0;         }
         else if ( *fc >= 12.99 && *fc <=  13.01 )
                {  *mc=0.087;
                   *rez=16.0;
                   *imz=-161.0;         }
         else if ( *fc >= 13.99 && *fc <=  14.01 )
                {  *mc=0.092;
                   *rez=16.0;
```

```
                *imz=-148.0;          }
        else if ( *fc >= 14.99 && *fc <=  15.01 )
                {  *mc=0.098;
                   *rez=14.0;
                   *imz=-137.0;          }
        else
              printf("\n passes through");

}
```

APPENDIX I

PRESS SUBROUTINE

```
/*****************************************************************************/
/* Subroutine "press" converts the voltage signal to the pressure signal.    */
/* Given the end-of-cable sensitivity, calculated from subroutine "marc",    */
/* pressure = [ (voltage)(1e6) ]/ ml   {Pascal}.                             */
/*****************************************************************************/

void press(ml)

float   ml;                     /* end-of-cable sensitivity (uV/Pa) */
{

char    key,
        filename[30],
        xbuff[20],                /*  x - axis labeling buffer  */
        ybuff[20]                 /*  y - axis labeling buffer  */
        ;

int i=0,
    c,
    counter,
    npts;                     /* number of points for "findmax" */


float xtc,                    /* x-time calc. scaling factor        */
      ytc,                    /* y-time calc. scaling factor        */
      maximum,                /* y max value                        */
      minimum,                /* y min value                        */
      ymult,                  /* y scaling factor                   */
      xmult,                  /* x scaling factor                   */
      xzero,                  /* x min or zero start                */
      xincr,                  /* x increment                        */
      xste,                   /*   change in x  : for plotting  */
      xmax,                   /*   max x        : for plotting  */
      yori,                   /*   left hand y  : for plotting  */
      yste,                   /*   change in y  : for plotting  */
      ymax;                   /*   max y        : for plotting  */


strcpy(xbuff,"Time");
strcpy(ybuff,"Pressure (Pa)");

     for(c=0; c<1024; c++)            /* copy original voltage signal to y array      */
        y[c] = echo[c];

/***************************** Y SCALING FOR PLOTTING ******************************/

        maximum = findmax(&counter,1024);       /* find max and min for plotting purposes       */
        minimum = findmin(&counter,1024);


        yori = ((minimum*1.0e6)/ml)+0.15*((minimum*1.0e6)/ml);          /* minimum y value      */
        ymax = ((maximum*1.0e6)/ml)+0.15*((maximum*1.0e6)/ml);          /* maximum y value      */
        yste = ((ymax - yori)/10.0);                                   /* y step size to have 10 increments
*/

/******************************** fill arrays x & y ****************************/

        xincr = 2.5e-8;                 /* each x value as determined by 40 MHz */
```

```
        x[0] = 0.0;                     /* assume start time is zero           */
        y[0] = (y[0]*1.0e6)/ml;

        for ( i = 1; i < 1024; i++ )            /* change voltage signal to a pressure signal   */
                {                               /* need to multiply by 1e6 to change voltage to */
                 x[i] = (x[i-1]+xincr);         /* micro volts for ml (uV/Pa)                   */
                 y[i] = (y[i]*1.0e6)/ml;
                }

/***************************** X SCALING FOR PLOTTING ************************/

        xmax = ( 400*xincr );   /* x maximum is currently set to show only 400 points -- can be changed */

        if ( xmax >= 1 )                /*  xmax > 1 sec   */
                {
                xtc=1;
                strcpy(xbuff,"Time (sec)");
                }

        if (  xmax < 1 &&  xmax >= 0.001 )
                {
                xtc=1000;
                strcpy(xbuff,"Time (msec)");
                }

        if (  xmax < 0.001 && xmax >= 0.000001 )
                {
                xtc=1000000;
                strcpy(xbuff,"Time (usec)");
                }

        else
                {
                xtc=1e9;
                strcpy(xbuff,"Time (nsec)");
                }

        for ( i = 0; i < 1024; i++ )    /* change x values to read integer given known axis label       */
                x[i] *= xtc;


        xmax *= xtc;
        xste = (xmax - x[0])/10.0;      /* x step size to give 10 divisions


   printf("\n\n Do you wish to plot the pressure?? [n]  ");
   key = getch();
   if (key == 'y' || key == 'Y')
       {
        printf("\n Please enter filename [8 letters]:   ");
        scanf("%s", filename);
        strcat(filename, ".plt");               /* file is default with extension .plt  */

/****************************** plotting part ********************************/

        bgnplot(1,'g', filename);
        startplot(1);
        font(3,"simplex.fnt",'\310',"triplex.fnt",'\311',"complex.fnt",'\312',"",'2');
        page(9.0,6.855);
        area2d(8.0,5.5);
        grid(2);
        upright(1);
        color(10);
        numht(0.11);
        graf("%2.2f",x[0],xste,xmax,"%4.3f",yori,yste,ymax);
        color(10);
        xname(xbuff);
        heading(filename);
        yname(ybuff);
        color(14);
```

```
        curve(x,y,400,0);       /* must change 400 to change number of points to plot */
        endplot();
        stopplot();
       }

}
```

APPENDIX J

INT_TIME_AVE SUBROUTINE

```
/*****************************************************************************/
/* Subroutine "int_time_ave" calculates instantaneous intensities, intensity */
/* temporal peak, intensity pulse average and intensity time average.  It    */
/* assumes that the y array is the pressure signal.  It changes it to the    */
/* intensity values using equation I = (p**2)/[ (Zo)(1e4) ].  The Itp is the */
/* absolute maximum intensity value.  Then the peak intensity integral is    */
/* calculated, pii = (sum) (y[i]*2.5e-8).  The maximum value of the pii is    */
/* needed to calculate Ipa.  The time between the 10% and 90% of the pii     */
/* is tau.  The pulse duration is 1.25 * tau (AIUM standard).  So,           */
/* Ipa = piimax/pd.  The dutycycle is the pulse duration divided by the pulse */
/* repetition period.  So, Ita = Ipa * dutycycle.                            */
/*****************************************************************************/

void int_time_ave()

{
int     dex,
        index,
        ninety,
        ten,
        c,
        i=0;

float   xincr
        ;

/************************** create intensity values **************************/

        xincr = 2.5e-8;
        x[0] = 0.0;
        intenwm[0] = ( (y[0]*y[0])/Zo )/10000;
           y[0] = intenwm[0];

        for (i=1; i<1024; i++)
                {
                 x[i] = (x[i-1]+xincr);
                 intenwm[i] = ( (y[i]*y[i])/Zo )/10000;
                 y[i] = intenwm[i];
                }

/************************** intensity temporal peak **************************/

         itp = findmax(&dex,1024);

/************************** pulse intensity integral *************************/

        y[0]=0;
        for (i=1; i<1024; i++)
                y[i] = (y[i] * 2.5e-8) + y[i-1];        /*  J/cm**2   */

/******************** 10% and 90% of pii and pulse duration ******************/

        piimax = findmax(&dex,1024);
        c = dex;

        for(i=0; i<1024; i++)
                if ( y[i] >= (0.9*piimax) )
                   {    ninety = i;
```

```
                    break;
                }

        for(i=0; i<1024; i++)
                if ( y[i] >= (0.1*piimax) )
                {   ten=i;
                    break;
                }

        pd = (float)fabs( (double)x[ninety] - (double)x[ten] ) * 1.25;

        dutycyc=pd/(float)prp;

/**************** intensity pulse average and temporal average ****************/

        ipa=piimax/pd;
        ita=ipa*dutycyc;

}
```

APPENDIX K

DONE SUBROUTINE

```
/*****************************************************************************/
/* Subroutine "done" prints to the screen all the necessary variables.  It   */
/* asks to save the values to a file.                                        */
/*****************************************************************************/

void done()
{

char    key;

int     c;

printf(CLS);
printf("\n  The following is the data just acquired:  \n");
printf("\n  Center frequency          =  %f  MHz  ", cent_freq);
printf("\n  Bandwidth                 =  %f  MHz  \n", bandwidth);
printf("\n  Sensitivity (Ml) O.C.     =  %f  uV/Pa  \n", ml);
printf("\n  Peak compressional pressure (fc)   =  %f  kPa  ", pc/1000);
printf("\n  Peak rarefractional pressure (fc)  =  %f  kPa  \n", pr/1000);
printf("\n  Pulse repetition period     =  %f  msec  ", prp * 1000);
printf("\n  Pulse repetition frequency  =  %f  kHz  ", 1/(prp * 1000) );
printf("\n  Pulse duration              =  %f  usec  ", pd * 1e6);
printf("\n  Dutycycle                   =  %f  percent\n", dutycyc * 100);
printf("\n  Pulse Intensity Integral (PII)     =  %f  uJ/cm**2  ", piimax * 1e6);
printf("\n  Intensity Pulse Average (Ipa)      =  %f  W/cm**2  ", ipa);
printf("\n  Intensity Time Average (Ita)       =  %f  mW/cm**2  ", ita * 1000);
printf("\n  Intensity Time Peak (Itp)          =  %f  W/cm**2  \n", itp);

printf("\n\n  Do you wish to save data??     ");

key = getch();
if (key == 'y' || key == 'Y')
        {
         printf("\n  Please enter filename [8 letters(dot)3 letter extension]:  ");
         printf("\n  ---> ");
         scanf("%s", filename);

         if( (output = fopen(filename, "w") ) == NULL)
               {
                printf("\nHelp... can't open %s!!\n", filename);
                exit(0);
               }

         printf("\n  Storing data in %s\n", filename);

         fprintf(output, "%s\n", filename);

         fprintf(output, "%f  MHz  \n", cent_freq);
         fprintf(output, "%f  MHz  \n", bandwidth);
         fprintf(output, "%f  uV/Pa  \n", ml);
         fprintf(output, "%f  Pa  \n", pc);
         fprintf(output, "%f  Pa  \n", pr);
         fprintf(output, "%e  sec  \n", prp);
         fprintf(output, "%f  kHz  \n", 1/(prp * 1000) );
         fprintf(output, "%e  sec  \n", pd);
         fprintf(output, "%e  sec  \n", dutycyc);
         fprintf(output, "%f  J/cm**2  \n", piimax);
         fprintf(output, "%f  W/cm**2  \n", ipa);
```

```
        fprintf(output, "%f  W/cm**2  \n", ita);
        fprintf(output, "%f  W/cm**2  \n", itp);

        for(c=0; c<1024; c++)
              fprintf(output, "%e     %e     %f\n", x[c], echo[c], intenwm[c]);

        fclose(output);
        puts("\n  Done writing!\n");
       }

else
       puts("\n  No data saved!\n");

}
```

APPENDIX L

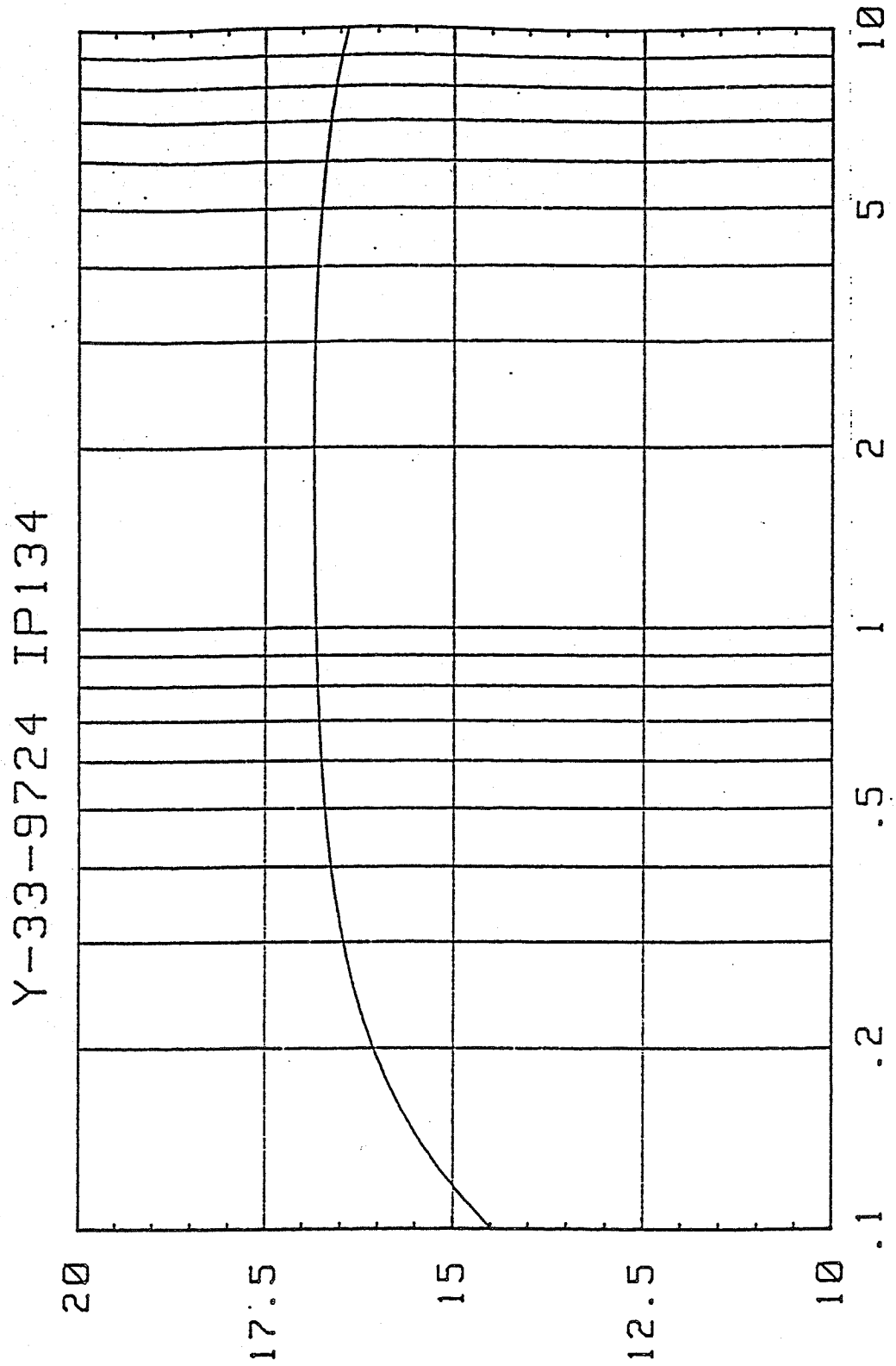
REMAINING SUBROUTINES

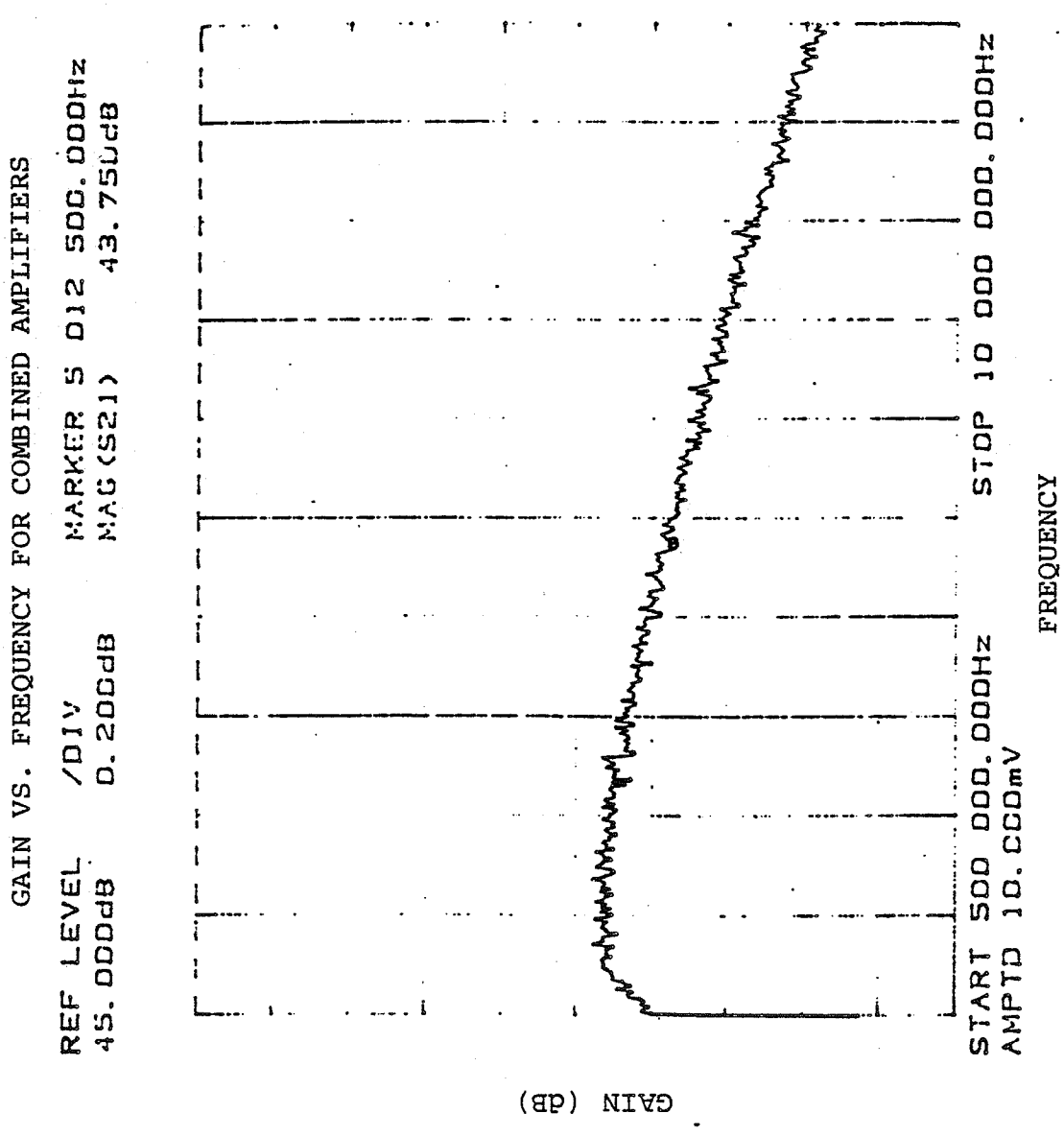
```
/*****************************************************************************/
/*  Subroutine "load_trigger" changes the trigger level to the "x" value.    */
/*  The values 0 to 256 represent 0.635 v to -0.640 v.  The trigger is set   */
/*  when there is a change at the chosen value.                              */
/*****************************************************************************/

void  load_trigger_offset(x)
      char x;
  {
     char far *waag;
     waag = ( char far *) WAAG;

     outpw(PORT2,0xc06f);      /* enabled trigger offset */
     *waag = x;

     outpw(PORT2,0xA06f);
  }



/*****************************************************************************/
/* Subroutine "absolute_max" displays to the screen the maximum and minimum  */
/* value in the y array.                                                     */
/*****************************************************************************/

void absolute_max(min, max, number_pts)

int number_pts;
double *min,
       *max;

{
 int i;

 *max = -1000000.0;
 *min = 1000000.0;

 for (i=0;  i<number_pts;  i++)
       {
        if (*max < (double)y[i])
              *max = (double)y[i];
        else if (*min > (double)y[i])
              *min = (double)y[i];
       }

       STP(24,69);
       printf("%f",*max);
       STP(25,69);
       printf("%f",*min);
}
```

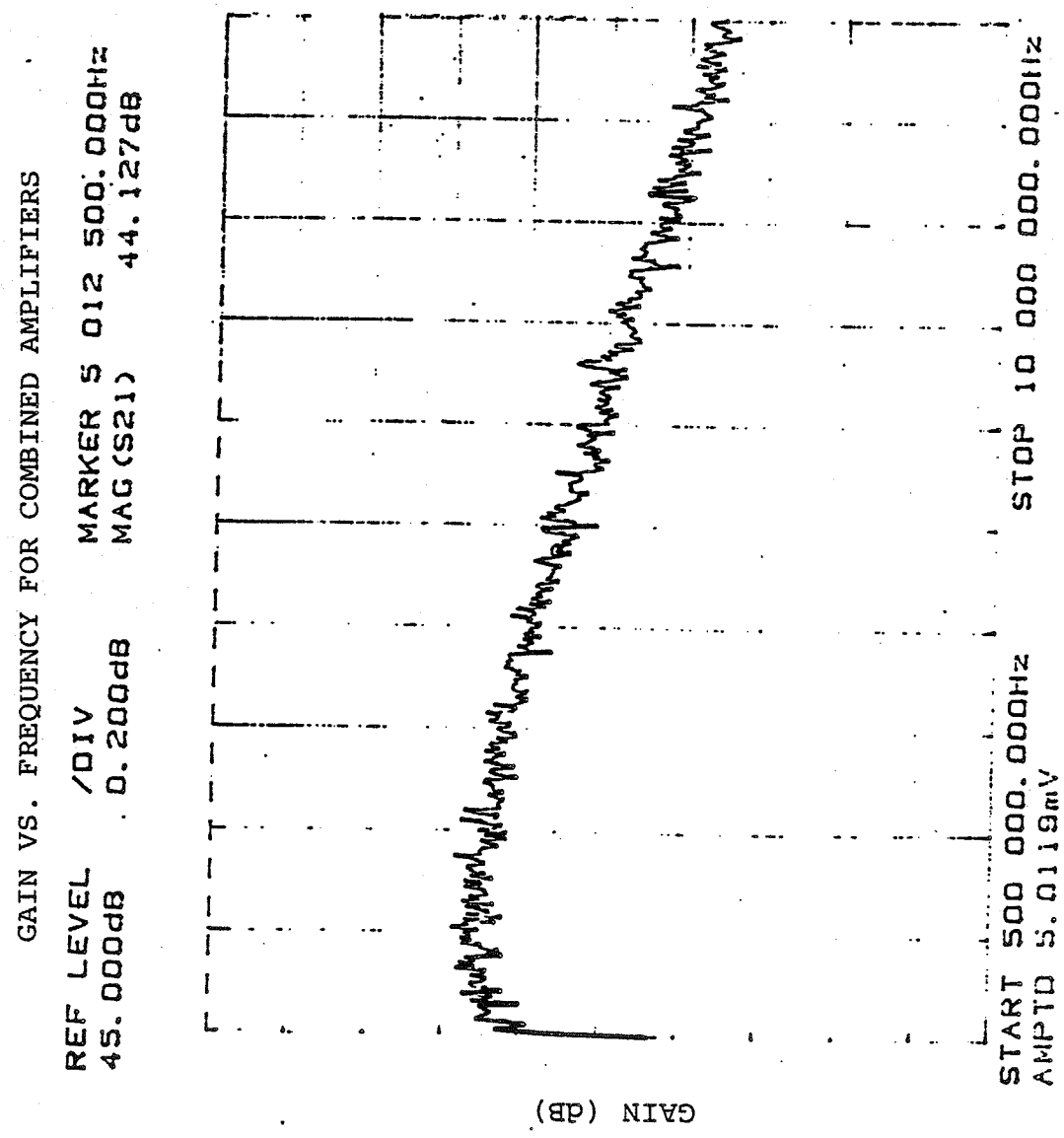
```
/*****************************************************************************/
/* Subroutine "findmax" returns the maximum value in the y array.  If        */
/* necessary, the index for the minimum value in the array is also returned. */
/*****************************************************************************/

float findmax(index,number_pts)

int     *index,
        number_pts
        ;
{

int     i
        ;

float   max
        ;


        max = -1000000.0;

        for (i=0; i<number_pts; i++)
                if (max < y[i])
                   {
                    max = y[i];
                    *index = i;
                   }
        return(max);
}



/*****************************************************************************/
/* Subroutine "findmin" returns the minimum value in the y array.  If        */
/* necessary, the index in the array for the minimum value is also returned. */
/*****************************************************************************/

float findmin(index,number_pts)

int     *index,
        number_pts
        ;

{

int     i;

float   min;

        min = 1000000.0;

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

                if (min > y[i])
                   {
                    min = y[i];
                    *index = min;
                   }

        return(min);
```

APPENDIX M

FREQUENCY RESPONSE OF MARCONI HYDROPHONE AMPLIFIER AND MINICIRCUIT ZFL500LN POWER AMPLIFER


Y-33-9724 IP134
Gain (dB)
20
17.5
15
12.5
10
.1
.2
.5
1
2
5
10
Frequency (MHz)
Run performed by S.SOLANKI on 8.8.88 using
HP4192A Impedance Analyser (TE no.10656) - oscillator level 100 mV.
© GEC-Marconi Limited 1988


GAIN VS. FREQUENCY FOR COMBINED AMPLIFIERS
REF LEVEL
45.000dB
/DIV
0.200dB
MARKER 5 012 500.000Hz
MAG(S21) 43.750dB
START 500 000.000Hz
AMPTD 10.000mV
STOP 10 000 000.000Hz
FREQUENCY
GAIN (dB)


GAIN VS. FREQUENCY FOR COMBINED AMPLIFIERS
REF LEVEL
45.000dB
/DIV
0.200dB
MARKER 5 012 500.000Hz
MAG(S21) 44.127dB
START 500 000.000Hz
AMPTD 5.0119mV
STOP 10 000 000.000Hz
GAIN (dB)
FREQUENCY

GAIN VS. FREQUENCY FOR COMBINED AMPLIFIERS