

Louisiana State University

LSU Scholarly Repository

LSU Historical Dissertations and Theses

Graduate School

12-1990

A Data Acquisition and Processing System for High Frequency Repetitive Waveforms

Daran Lynn Rehmeyer

Louisiana State University and Agricultural and Mechanical College

Follow this and additional works at: https://repository.lsu.edu/gradschool_disstheses

Recommended Citation

Rehmeyer, Daran Lynn, "A Data Acquisition and Processing System for High Frequency Repetitive Waveforms" (1990). *LSU Historical Dissertations and Theses*. 8299.

https://repository.lsu.edu/gradschool_disstheses/8299

This Thesis is brought to you for free and open access by the Graduate School at LSU Scholarly Repository. It has been accepted for inclusion in LSU Historical Dissertations and Theses by an authorized administrator of LSU Scholarly Repository. For more information, please contact gradetd@lsu.edu.

A DATA ACQUISITION AND PROCESSING SYSTEM
FOR HIGH FREQUENCY REPETITIVE WAVEFORMS

A Thesis

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering
in
The Department of Electrical Engineering

by
Daran Lynn ~~Rehmeyer~~
B.S., Virginia Polytechnic Institute and State University, 1982
December 1990

C.2

MANUSCRIPT THESES

Unpublished theses submitted for the Master's and Doctor's Degrees and deposited in the Louisiana State University Libraries are available for inspection. Use of any thesis is limited by the rights of the author. Bibliographical references may be noted, but passages may not be copied unless the author has given permission. Credit must be given in subsequent written or published work.

A library which borrows this thesis for use by its clientele is expected to make sure that the borrower is aware of the above restrictions.

LOUISIANA STATE UNIVERSITY LIBRARIES

378.76

L930

1990

c. 2

ACKNOWLEDGEMENT

My thanks go to Jorge Aravena, who adopted me when no one else would have me. I appreciated the counsel and advice. I would also like to thank Powsiri Klinkhachorn for helping organize and clarify the concept for this thesis in the beginning. Thanks also go to my committee Ahmed El-Amawy and Suresh Rai for critiquing my presentation and thesis. And finally, my thanks to my wife, Teresa, for tolerating me while I worked on this.

TABLE OF CONTENTS

Acknowledgment	ii
List of Figures	v
Abstract	vii
1. Introduction	1
2. Signal Acquisition	4
2.1 Requirements	4
2.2 Equivalent Time Sampling	5
3. Hardware Design and Construction	10
3.1 Data Acquisition Subsystem	10
3.2 Transmitter Trigger	13
3.3 Timing Subsystem	14
3.4 Integration of the Subsystems	19
3.5 Evolution of the Timing Circuit Design	23
3.6 Construction	27
4. Testing of the Hardware	30
4.1 Hardware Testing Software	30
4.1.1 Subsystem Testing Software	30
4.1.2 Signal Acquisition Software	33
4.2 Hardware System Test	34
4.2.1 Individual Subsystems	34
4.2.1.1 Timing Subsystem	34
4.2.1.2 Transmitter Trigger Subsystem	35
4.2.1.3 Data Acquisition Subsystem	35

LIST OF FIGURES

Figure 1.	System Block Diagram.	1
Figure 2.	Real-Time Sampling at 1 GHz.	6
Figure 3.	Repeating Signal.	7
Figure 4.	Equivalent Time Sampling.	8
Figure 5.	Data Acquisition, Timing, and Trigger Block Diagram.	10
Figure 6.	Data Acquisition Schematic.	13
Figure 7.	Transmitter Trigger Schematic.	13
Figure 8.	Timing Subsystem Block Diagram.	15
Figure 9.	Timing Subsystem Schematic.	17
Figure 10.	Effect of Feedback on the Comparator.	18
Figure 11.	Integrated System Schematic.	
Figure 11-A.	Timing and Data Acquisition Subsystems Schematic.	20
Figure 11-B.	Address Decode/Chip Select and Transmitter Trigger Schematic.	21
Figure 12.	Acquisition Timing Diagram.	23
Figure 13.	Ramp Circuit - First Try.	24
Figure 14.	Ramp Circuit - Second Try.	25
Figure 15.	Ramp Circuit - Third Try.	26
Figure 16.	Fabricated System Board.	28
Figure 17.	System Configuration for Evaluation and Test.	36
Figure 18.	System Recorded Signal.	37
Figure 19.	Oscilloscope Recorded Signal.	37

Figure 20.	System Recorded Internal Ramp Signal.	39
Figure 21.	Statistical Analysis of Ramp at 25% Point.	40
Figure 22.	Statistical Analysis of Ramp at 50% Point.	41
Figure 23.	Statistical Analysis of Ramp at 75% Point.	42
Figure 24.	System Recorded Signal for Statistical Analysis.	44
Figure 25.	Statistical Analysis of Signal at 25% Point.	45
Figure 26.	Statistical Analysis of Signal at 50% Point.	46
Figure 27.	Statistical Analysis of Signal at 75% Point.	47
Figure 28.	FFT of Recorded Signal.	51
Figure 29.	20 th Order Butterworth Filter.	52
Figure 30.	10 th Order Elliptic Filter.	53
Figure 31.	20 th Order Chebychev Type 1 Filter.	53
Figure 32.	20 th Order Chebychev Type 2 Filter.	54
Figure 33.	$w_c=0.30$ Butterworth Filter.	54
Figure 34.	$w_c=0.30$ Elliptic Filter.	55
Figure 35.	$w_c=0.30$ Chebychev Type 1 Filter.	55
Figure 36.	$w_c=0.30$ Chebychev Type 2 Filter.	56
Figure 37.	Deconvolution of Time Jitter.	57
Figure 38.	Data Interpretation Criteria.	58
Figure 39.	Processing Software - Midpoint Selection.	60
Figure 40.	Data Processing and Display Software Menu Tree.	61, 62

Abstract

The purpose of this thesis is two-fold: to develop a computerized data acquisition system to replace a system using an oscilloscope-mounted Polaroid camera for recording waveforms, and secondly, to develop the processing software to supplement and ultimately replace the manual interpretations of the recorded signals.

The impetus for upgrading the original acquisition system is provided by two main factors. The first is the volume and cost of the film required to record an entire waveform using the Polaroid camera and second, the time consuming method of recording and splicing the individual pictures together to form a complete waveform. The current system requires 10 minutes to record and 10 minutes to splice a single waveform. This excessive time precludes continuous profiling and immediate interpretation. Further time could be saved by changing from the manual interpretation technique to one performed by software. Greater consistency in interpretation results could also be achieved.

To meet and exceed the resolution of the Polaroid-recorded waveform, the data acquisition system would require a sampling rate of 1 GHz. Sampling in real time at 1 GHz is possible. Systems are available that meet and exceed this sampling rate. However, the cost of these systems is prohibitive. This thesis

covers the development of a low cost acquisition system that fills the 1 GHz sampling rate requirement.

The analysis of the data is *ad hoc* hence the development of specialized software is also necessary. The development of the critical processing software will be covered.

1. INTRODUCTION

This thesis describes the development of a very high speed data acquisition system and the supporting acquisition and processing software. The equipment is part of a proprietary soil analyzer shown in block diagram form in Figure 1.

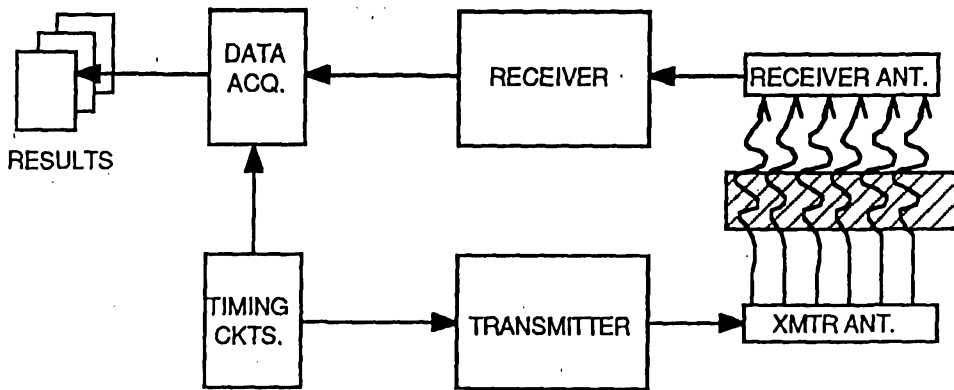


Figure 1. System Block Diagram.

The transmitter sends an electromagnetic pulse through the material under test. The receiving antennae collects the response and conditions it for the data acquisition system.

The current equipment configuration uses an oscilloscope-mounted Polaroid camera for data collection. Data preparation is carried out by hand splicing individual pictures to form a complete waveform. A complete waveform is composed of eight spliced pictures. The length of the recorded waveform is approximately 4 microseconds. Current interpretation of the recorded signals is based on visual analysis of the wave shape

and amplitude.

The subsystems selected for computerization are the Data Acquisition System, Transmitter Trigger Circuit, and the Timing Circuits. The transmitter and receiver will be left as is. The greatest return on investment (both time and money) will only come in eliminating the use of the Polaroid film and automating the interpretation for consistency and speed.

The operation of the system is as follows: The timing circuit generates a trigger pulse to the transmitter. The transmitter fires and sends a signal to its antennae. The signal travels through the material of interest and is picked up by the receiver antennae. The receiver amplifies and filters the incoming signal (all analog) and passes the signal to the data acquisition system.

The timing circuit is also responsible for starting the data acquisition procedure. The timing is set such that just before the signal arrives at the input to the data acquisition system, a signal is sent from the timing circuit which starts the acquisition of the incoming signal.

Of the three subsystems selected for computerization, the Data Acquisition System presents the greatest technical challenge. To duplicate and exceed the resolution of the Polaroid-recorded waveform in the proposed Data Acquisition System, the digital sampling rate was set to 1 GHz with the capability of collecting 4000 points per recorded waveform. The

method chosen to accomplish this is described in section 3.1.

The processing software is developed separately. It is required that the computer interpretation be consistent with the manual interpretation while also eliminating the inconsistencies between different operators. The processing software is written to be independent of the acquisition procedure so that it could stand alone regardless of the method of acquisition (given data input constraints).

The thesis is organized as follows: Chapter 2 is a description of the hardware requirements and the solutions and methods picked to satisfy each. The actual hardware design and construction is described in chapter 3. Each of the completed subsystems and the final integrated system is covered. Chapter 3 also includes a description of specific hurdles and design problems encountered in the Timing Circuit subsystem design. The test and evaluation of the hardware system is described in chapter 4. The software developed to test each subsystem, the physical test methods, and evaluation of the system are covered. Chapter 5 describes the data processing and display software. The signal conditioning, data interpretation, and data display software are described. Finally, chapter 6 includes comments and conclusions on this design project.

2. SIGNAL ACQUISITION

This chapter describes the basic system requirements and the approach taken to design the data acquisition system. The generic IBM-PC bus was selected to support the new hardware. The PC was selected because the system bus is easily accessible through the backplane expansion slots. Pascal was chosen as the high-level programming language. Turbo-Pascal was specifically selected because of its availability and the ability to incorporate imbedded machine code in the program.

2.1 Requirements

In the original system utilizing the oscilloscope-mounted Polaroid camera, the time scale on the oscilloscope was set to 0.05 usec/div and a maximum of eight frames would be shot. The composite of these eight frames would result in roughly 4 microseconds of recorded waveform. To duplicate (and exceed) this resolution digitally, the sampling rate was set at 1 GHz (1 nanosecond between samples) with again, 4 microseconds of recorded waveform.

The output of the receiver has a maximum voltage swing of 16 volts nominal. A 12-bit converter was chosen to record the waveform with adequate resolution.

The overall system requirements are:

- interface to IBM-PC bus
- record 4 microseconds of waveform

- 1 GHz sampling rate
- 12-bit resolution from A/D converter
- generate all necessary timing signals and trigger pulses.

The fastest commercial PC-based real-time data acquisition system found is marketed by Rapid Systems. It has a 100 MHz sampling rate with an 8-bit resolution. This is still an order of magnitude lower than the required sampling rate and also has a lower resolution than required. Additionally, the input of this system is constrained to 2 volts. The principle of *equivalent time sampling* was used to satisfy the 1 GHz sampling rate requirement. Equivalent time sampling is covered in the next section.

2.2 Equivalent Time Sampling

All of the requirements listed previously are straightforward except the 1 GHz sampling rate. The first approach to meet the required sampling rate is to try *real-time* sampling. Real-time sampling is defined as digitizing sequentially from the same signal each sample point at the required sampling rate. For a 1 GHz sampling rate, the time between samples is the reciprocal of the sampling frequency or 1 nanosecond. Figure 2 depicts graphically the principle of real-time sampling. Sampling at 1 GHz in real-time requires ultra-high speed components and necessitates very accurate timing

constraints and therefore more complex design strategies which, because of component cost and availability, put this approach out of the reach of this project.

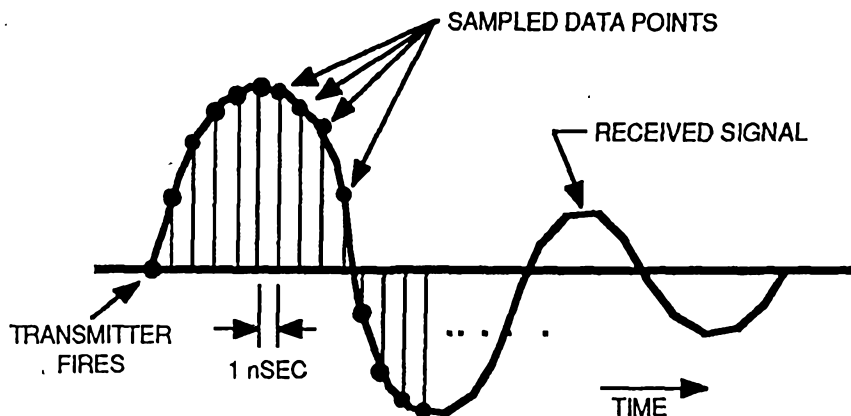


Figure 2. Real-Time Sampling at 1 GHz.

Since sampling at 1 GHz in real-time is an unreasonable goal for this project, an alternative method had to be found. The following discussion places the groundwork and justifies the decision to use equivalent time sampling as the alternative method.

For each firing of the transmitter, an identical signal is received. The signal lasts from 4 to 12 microseconds before returning to the zero-signal baseline. The received signals can be assumed to be identical under the following conditions: the transmitter and receiver antennas are not moved while sampling and the signal from the transmitter is identical for each firing. Both of these conditions are realistic expectations that appear to be justified from the observed data. Outside interference

from other signal sources has not been shown to be a problem from inspecting waveforms recorded on film. However, removal of unwanted noise (eg. from other transmitted signals) could be accomplished using active filters on the receiver input without affecting the repeatability of the signal. Additionally, because of the short duration of the transmitted and received pulses, external influences from temperature changes or other environmental variations are negligible and have not been observed. Figure 3 graphically depicts a repetitive signal obtained by successive firings of the transmitter.

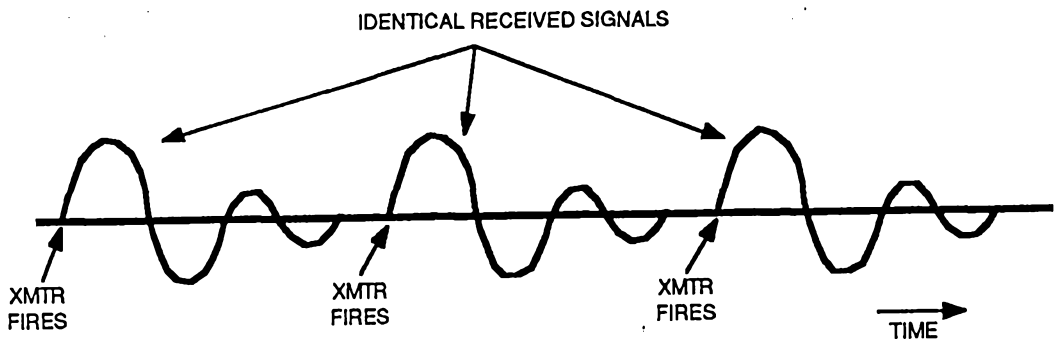


Figure 3. Repeating Signal.

Using equivalent time sampling the waveform can be stored. Equivalent time sampling is a technique which allows digitization of high frequency, repetitive signals. With each "pass" of the signal, one point on the waveform is digitized. For a 1 GHz sampling rate, this one point is 1 nanosecond away in relative-time from the previous point that was digitized. Figure 4 graphically depicts the principle of equivalent time sampling.

In order to collect n sequential values of a repetitive waveform, the transmitter must be triggered n times. Let h be the required relative-time interval between the sequential values (1 nanosecond for this system). If T_k and T_{k+1} are the starting times of the k and $k+1$ pulses, then the real time distance between the k and $k+1$ samples is $T_{k+1} - T_k + h$. Notice that as long as repeatability is assured, the difference $T_{k+1} - T_k$ may be allowed to vary. The critical value in the equation is h . The repeatability of h determines the accuracy of the equivalent time sampling system.

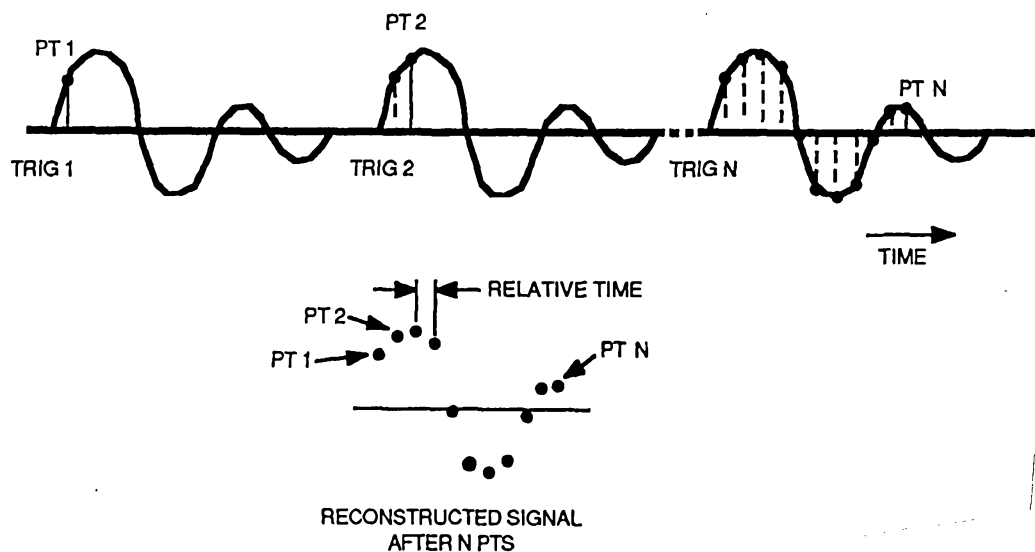


Figure 4. Equivalent Time Sampling.

The challenge then is to design a timing circuit capable of accurately triggering an A/D conversion with an equivalent time sampling rate of 1 GHz.

The major error introduced in equivalent-time waveforms is due to *time jitter*. Time jitter is defined as the error introduced in the recorded waveform when the relative-time between samples varies from the specified amount (here 1 nanosecond). Computational methods (deconvolution techniques) have been developed to compensate for and reduce the effects of time jitter. However, the literature recommends that the time jitter be reduced to a minimum by hardware design then use computational methods if still required. [7][10][12]

3. HARDWARE DESIGN AND CONSTRUCTION

A block diagram of the system to be designed is shown in Figure 5.

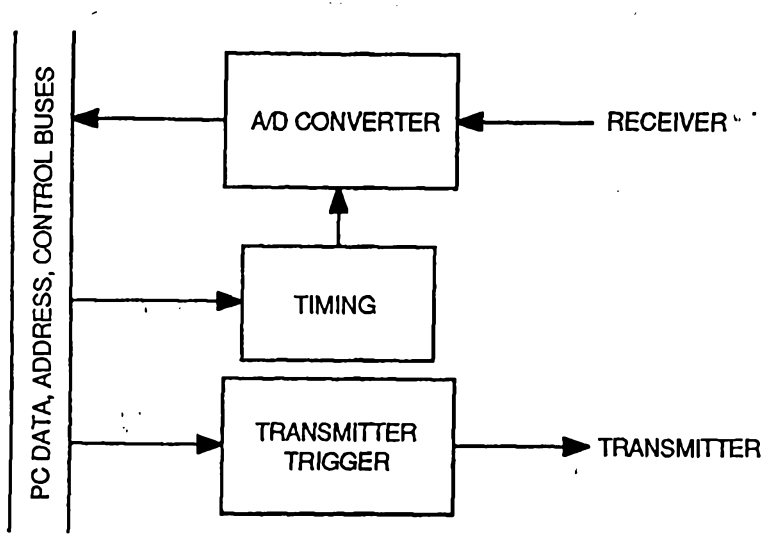


Figure 5. Data Acquisition, Timing, and Trigger Block Diagram.

The design will be handled as three separate subsystems and then integrated. The IBM-PC leaves I/O addresses 300H-31FH available for prototype boards. Addresses in this range will be used to control the Data Acquisition, Trigger, and Timing circuits.

3.1 Data Acquisition Subsystem

Since the maximum conversion time between samples is dependent on the transmitter firing rate and that rate would be controlled by the transmitter firing circuit and software, a

slower and cheaper A/D converter could be used. However, because of the high frequency content of the received signal, a high speed Track/Hold amplifier would be needed on the input of the A/D converter. The requirements set for the A/D converter were:

- 12-bit conversion
- microprocessor/TTL compatible
- 8-bit bus interface
- modest conversion time
- modest price.

The ADC674A from Burr Brown was selected. The conversion time of the ADC674A is 12 microseconds (nominal). This conversion time is comparable to the maximum time it takes for the received signal to return to the zero-signal baseline. Thus, a minimum amount of time is wasted before the transmitter can be triggered and the next sample taken.

Because of the high frequency of the signal to be digitized, a high speed track/hold amplifier would be needed to hold each sample point stable until the converter can complete the conversion. Because of the high frequency content of the received signal, the fastest possible track/hold amplifier aperture time and shortest possible aperture jitter would be needed. ECL track/hold amplifiers provide the fastest aperture time (2 nanoseconds) and shortest aperture jitter (2 picoseconds). However, because of the cost and incompatibility with the TTL circuitry, the ECL chips could not be used. The

Analog Devices HTC-0300A was selected for its high speed (aperture time of 6 nanoseconds, aperture jitter 50 picoseconds) and its compatibility with the A/D converter and TTL control circuitry.

The nominal 6 nanosecond aperture time of the Track/Hold amplifier is the delay from application of the Hold command to the Track/Hold amplifier to when the Track/Hold amplifier actually "holds" the input signal constant at its output. This effectively represents a "constant" time delay applied to all Hold commands. What determines the accuracy of the acquisition is the aperture jitter or uncertainty. This is the range over which the "constant" delay may vary from one Hold command to the next. The 50 picosecond aperture jitter is well within the specified accuracy of 1 nanosecond.

Figure 6 shows the basic data acquisition subsystem schematic. The op amp on the input of the Track/Hold amplifier serves two purposes. First, the maximum allowable input voltage to the amplifier is plus or minus 15 volts and the output range of the amplifier is plus or minus 10 volts. The op amp limits the maximum input voltage to the amplifier to roughly plus or minus 12 volts. This protects against overvoltage damage to the amplifier. Secondly, the op amp provides a DC offset for the input signal to bring it into the Track/Hold amplifier and A/D converter input ranges.

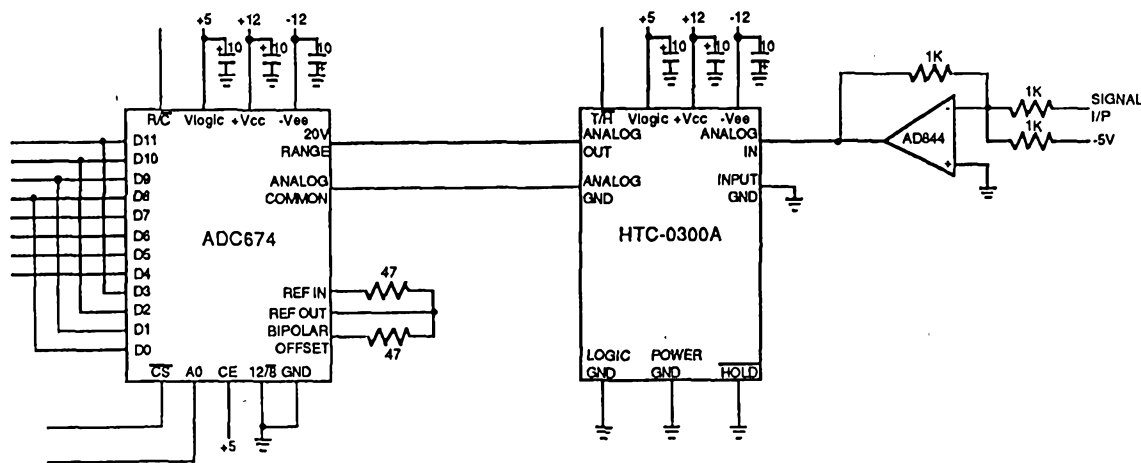


Figure 6. Data Acquisition Schematic.

3.2 Transmitter Trigger

The trigger input of the transmitter has a nominal input impedance of 50 ohms and requires a minimum amplitude pulse of 10 volts to trigger an output pulse. A monostable multivibrator, the 74LS122, was selected to generate the trigger pulse. Figure 7 shows the trigger circuit schematic.

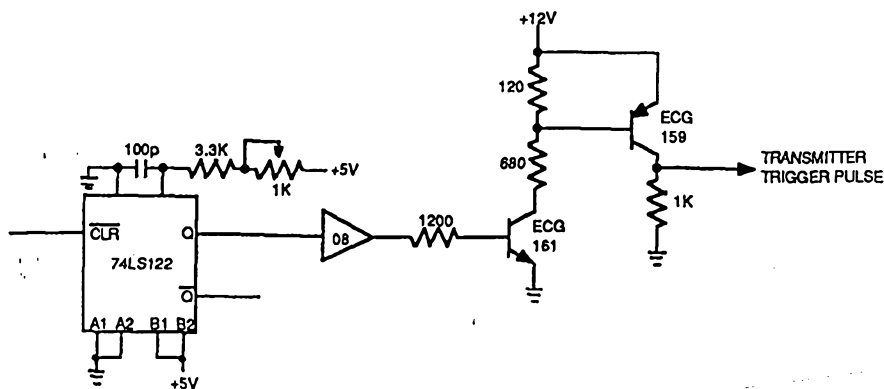


Figure 7. Transmitter Trigger Schematic.

By adjusting an external resistor and capacitor, the trigger pulse length can be controlled. The A1, A2 and B1, B2 inputs are connected to ground and Vcc respectively. These connections cause the positive going edge of a pulse on the CLEAR line to trigger the trigger pulse. The CLEAR input pulse is generated from the address decode logic for address 304H. The output pulse from the 74LS122 must be converted into a 10 volt minimum amplitude pulse for input to the 50 ohm impedance of the transmitter trigger. An amplifier circuit was designed to convert the TTL pulse to a nominal 12 volt pulse for input to the 50 ohm impedance transmitter trigger input.

3.3 Timing Subsystem

The design of the timing subsystem is the most critical aspect in setting the accuracy of the entire data acquisition system. The accuracy of the recorded signal depends upon the ability of the timing circuit to generate the HOLD signal for the Track/Hold amplifier for each sample point as close to 1 nanosecond from the previous point in relative-time as possible. To meet the resolution requirement of 1 nanosecond, the difference between the sample points in relative-time should be as close to 1 nanosecond as possible. The difference between the actual time and the specified 1 nanosecond is called *timing noise* or *jitter* and optimally should be much less than the 1

nanosecond. Ideally, if available, a counter operating at 1 GHz which could count from the transmitter trigger pulse to the desired sample point and trigger the Track/Hold amplifier would fill this requirement. However, no counters operating at this frequency and TTL compatible were found. A second approach was investigated. A programmable digital delay line with a one nanosecond step between delays could be used. To record a waveform four microseconds long, the delay line would require a minimum of 4000 programmable steps. A 12-bit delay line with 4096 steps would suffice. Again, however, no 12-bit delay lines were found.

An alternative timing circuit was developed. Using a linear ramp generating circuit, a digital-to-analog converter (DAC), and a differential comparator, an accurate timing circuit could be designed. Figure 8 shows the circuit in block diagram form.

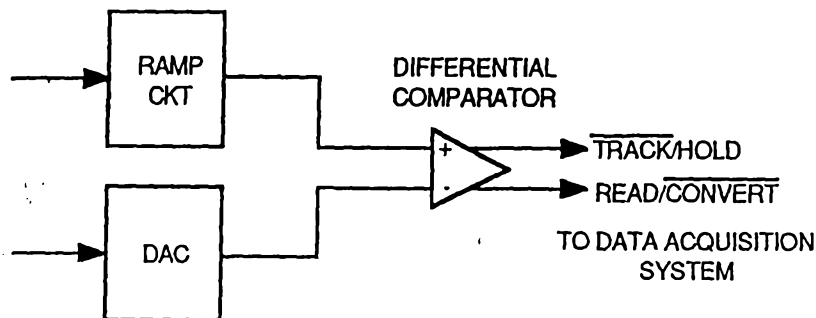


Figure 8. Timing Subsystem Block Diagram.

Operation of the Timing Subsystem is as follows: The DAC

output is set to a particular voltage. The ramp is triggered and starting from zero, the ramp voltage increases in a linear fashion. When the ramp voltage reaches the level of the DAC output voltage, the comparator fires, activating the HOLD for the Track/Hold amplifier. To store the next sample point, the DAC output voltage is increased by 1 least-significant-bit, the ramp reset, and then triggered again. The process repeats for each sample.

4 microseconds of waveform must be recorded. This length requires a minimum ramp length of 4 microseconds. The fastest TTL-output differential comparators have a maximum allowable differential input voltage of 5 volts nominal. This condition limits the range of the ramp from 0 to 5 volts. Given a 0 to 5 volt, 4 microsecond ramp, 4000 discrete levels must be recognizable. This will determine the resolution required from the DAC. A 12-bit DAC will provide 4096 discrete voltage levels, which for a 0-5 volt range will be 1.22 millivolts per step. Using a DAC with a maximum output voltage range of 0-5 volts will also ensure that the differential input voltage limit to the comparator is not exceeded. The AD667 from Analog Devices was selected because it fits all the above requirements with the addition of interfacing to an 8-bit data bus. Figure 9 shows the timing subsystem schematic.

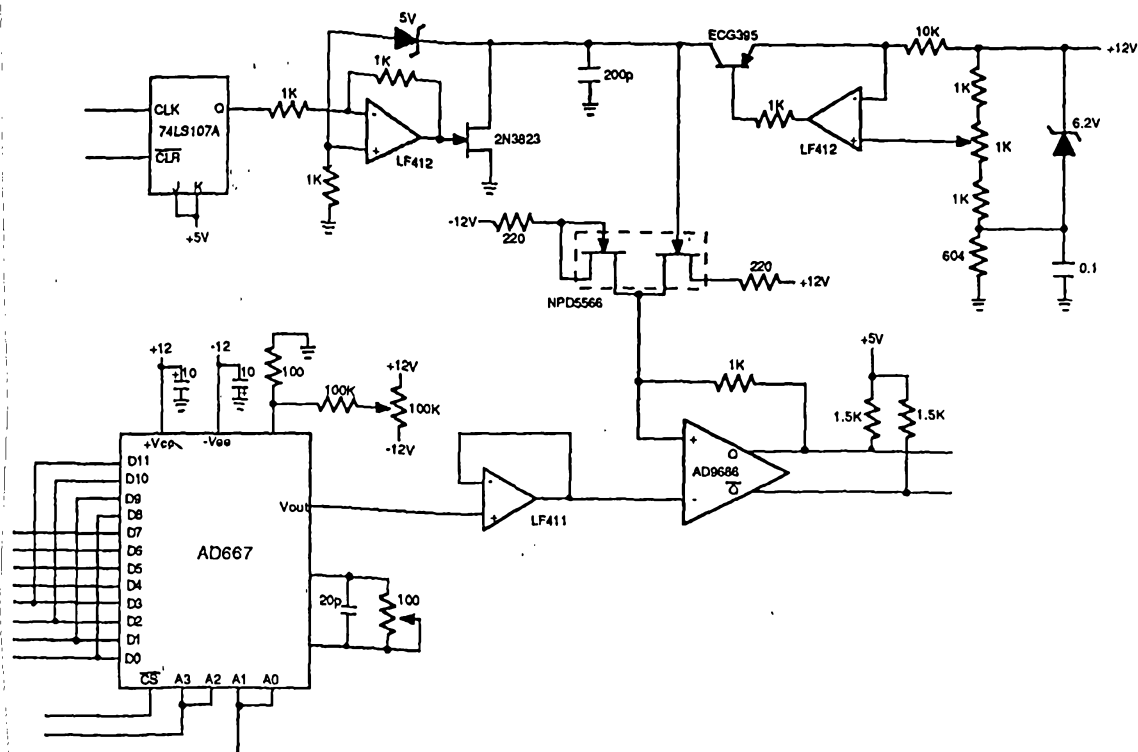


Figure 9. Timing Subsystem Schematic.

The ramp circuit design is covered in detail in section 3.5, an overview is given here. The 74LS107A is used to switch the ramp on and off by a pulse on the CLK input. The Q output switches the output of the LF412 op amp from 0 to -5 volts. This change switches the 2N3823 FET on and off respectively. When the FET is turned off, the current from the constant current source (collector of PNP transistor ECG395) flows into the 200 pf capacitor. A linear voltage increase (ramp) results. The combination of the two FET's in the NPD5566 isolates the ramp

circuit from the comparator to remove any non-linearities caused by the loading of the comparator and feedback circuits.

The AD9686 differential comparator is a high-speed TTL voltage comparator. This comparator has a nominal 7 nanosecond propagation delay from the input to the output switching to the high output level. The 1K ohm resistor from the Q output to the positive input of the comparator latches the output of the comparator high on the first transition of the Q output. Figure 10 illustrates the comparator response with and without feedback.

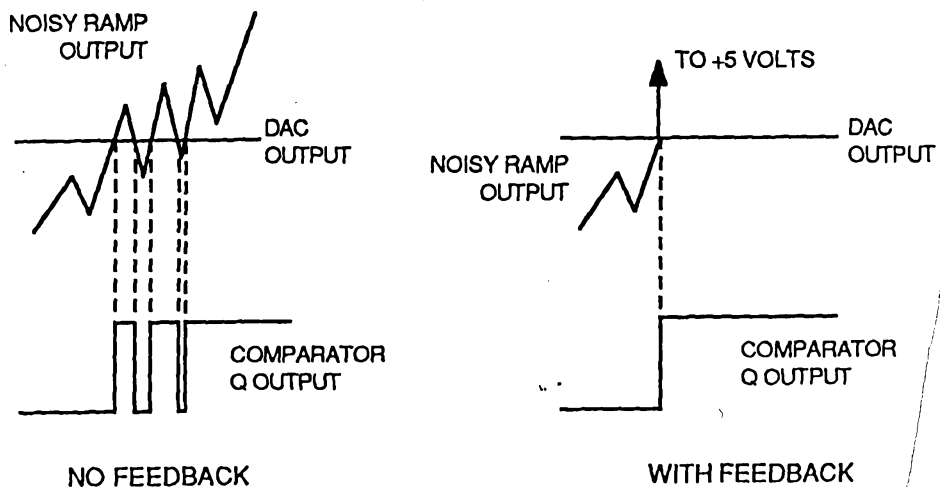


Figure 10. Effect of Feedback on the Comparator.

If noise is present on the ramp output or DAC output, with no feedback, the comparator may switch numerous times before settling to the final level. With feedback, the input to the comparator is pulled to +5 volts on the first transition of the

output. The feedback keeps the output from switching more than once. Multiple level transitions disrupt the analog-to-digital conversion.

3.4 Integration of the Subsystems

Integrating the three subsystems requires determining what timing relationships exist among the various signals that must be passed between the subsystems. Specific logic connections and relationships had to be determined also. Finally, what timing relationships should be controlled by software and/or what should be specifically controlled by hardware had to be determined and incorporated into the hardware or conversely left until the operating software was written.

Figures 11-A and 11-B show the entire system with all interconnections.

The more involved connections are explained below.

The complementary outputs of the differential comparator provide the complementary TRACK/HOLD and READ/CONVERT signals. The TRACK/HOLD signal to the track/hold amplifier must precede the READ/CONVERT signal to the A/D converter by roughly 120 nanoseconds. This ensures that the output of the track/hold amplifier is stable before the conversion is started. An AMDM-150 TTL delay line from Rhombus Industries was inserted into the READ/CONVERT line to provide the necessary delay.

Initially, both the transmitter trigger pulse and the ramp-

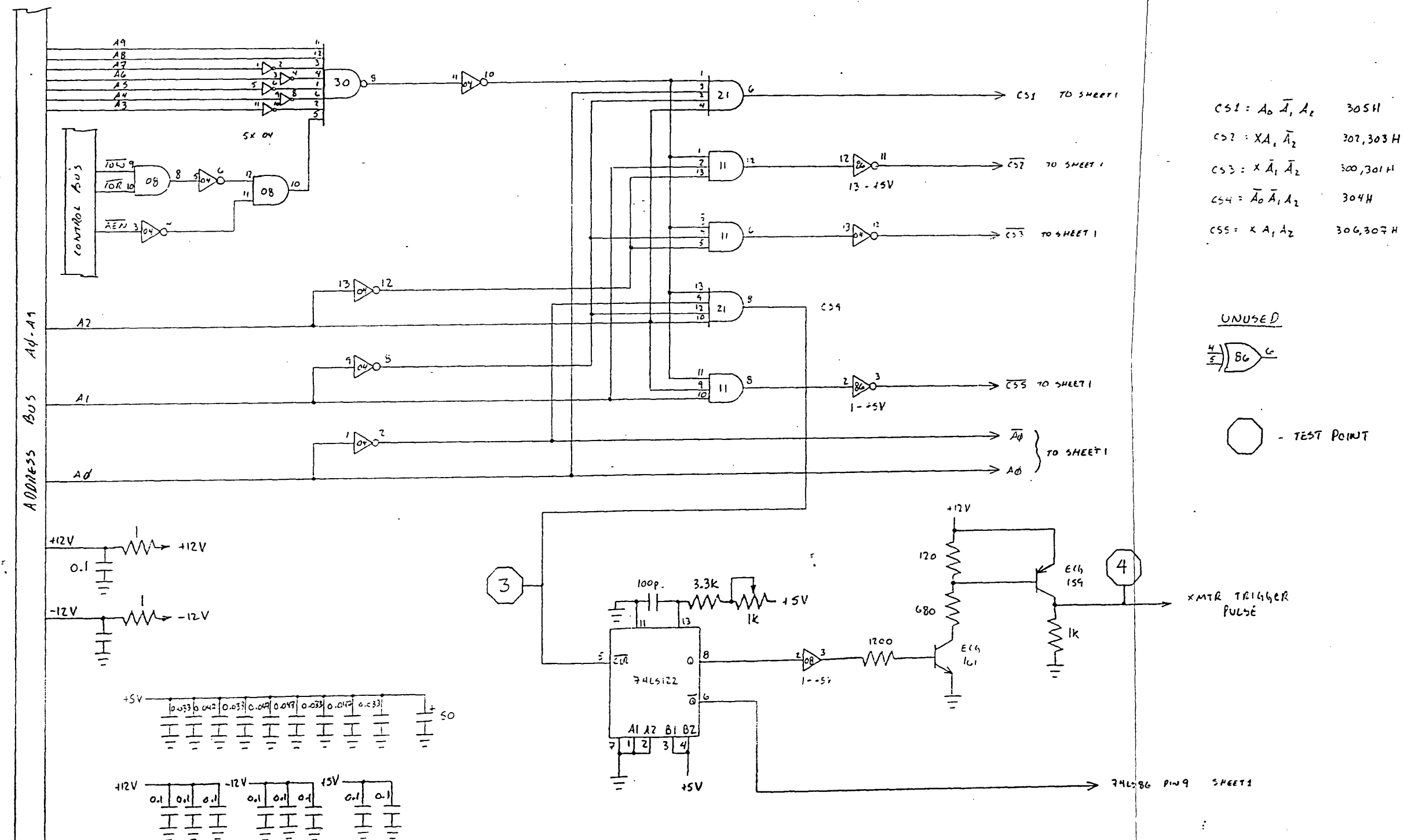


Figure 11-B. Chip Select/Address Decode and Transmitter Trigger Schematic.

start pulse were generated by the operating software. Although the code which generated the two pulses was purely sequential, in actuality, the time between these two pulses was not constant. The variation of the actual time between the two pulses from one sample to the next was too large and was not acceptable. The large variation introduced excessive jitter in the recorded signal. A line was connected from the 74LS122 (the transmitter trigger pulse source) to one side of an EXCLUSIVE-OR gate. The other side of the EXCLUSIVE-OR gate was connected to the software controlled ramp-start control circuit. This way, the ramp would be started from a transmitter trigger pulse with an acceptable amount of variation from sample to sample, but could still be controlled by software (eg., to turn the ramp off).

The manufacturer of the transmitter specifies the transmitter output pulse time jitter as no greater than 5 nanoseconds. This means the time delay from the transmitter trigger pulse on the input of the transmitter to the output pulse from the transmitter will vary by no more than 5 nanoseconds between transmitter firings. Because the acquisition is started by the transmitter trigger pulse and not the actual transmitter output pulse, this variation could add significant time jitter error to the recorded waveform. This error was deemed acceptable for the current equipment configuration and design.

The timing of the trigger, conversion, and A/D converter data read cycles are shown in Figure 12. Software initiated

versus hardware initiated signals are shown.

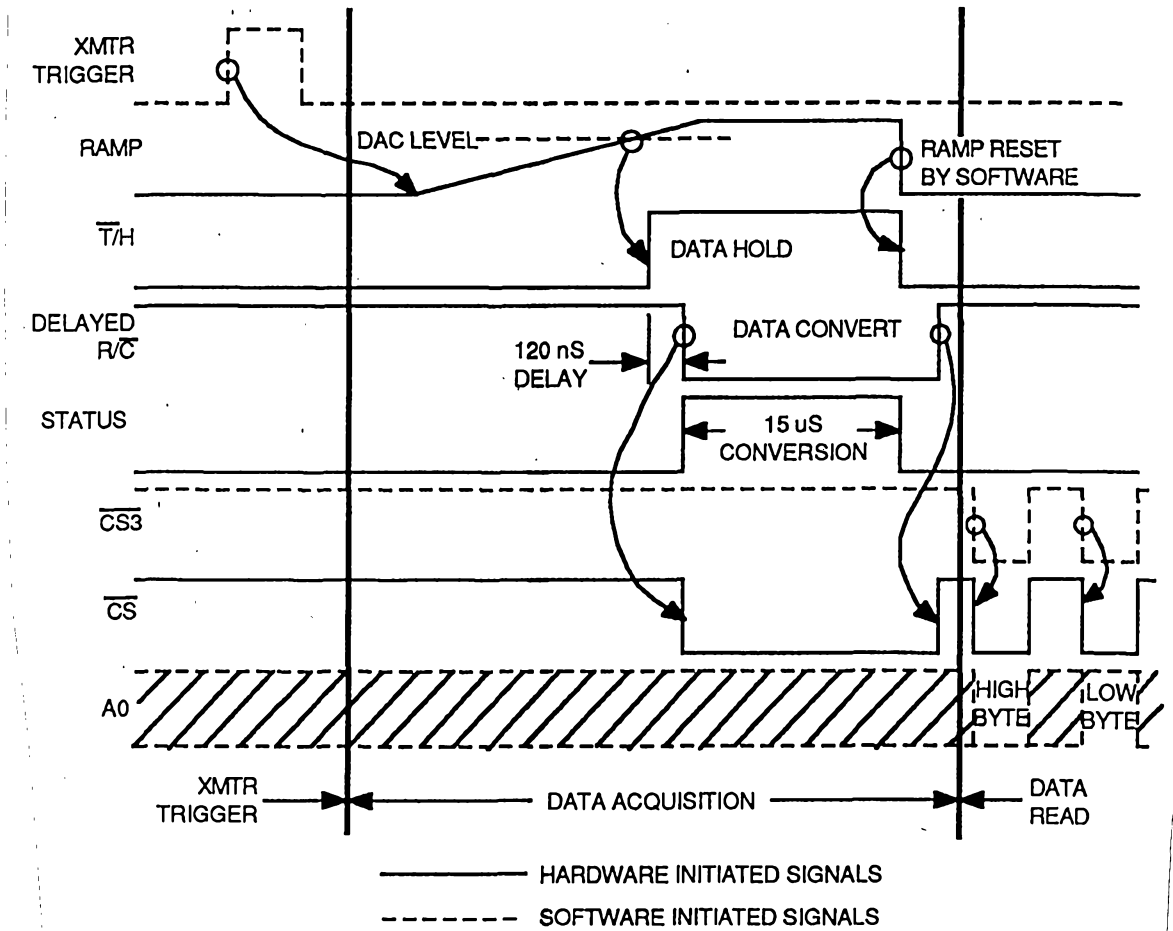


Figure 12. Acquisition Timing Diagram.

3.5 Evolution of the Timing Circuit Design

The timing circuit design proved to be the greatest challenge. Device specific limitations became readily apparent. In low speed/accuracy applications, these limitations would not

exhibit the same degrading effects on the circuit. The timing ramp circuit went through three major design iterations. Each iteration will be covered in sequence.

Figure 13 shows the first design iteration.

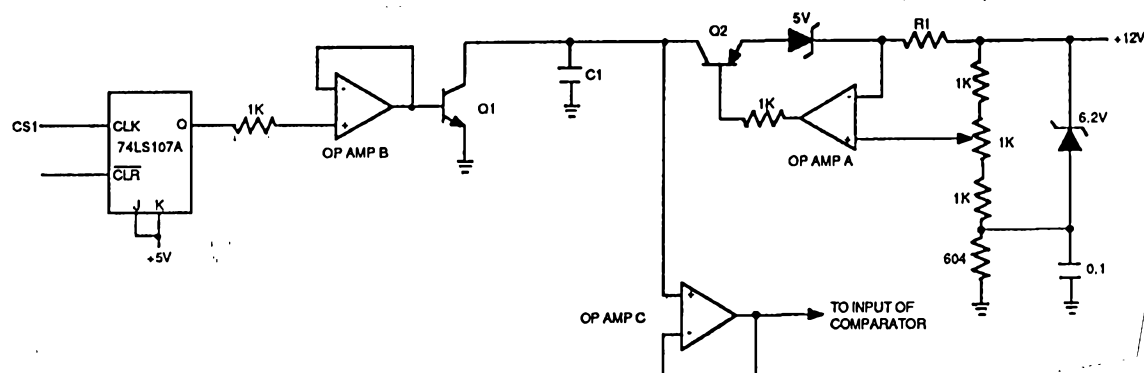


Figure 13. Ramp Circuit - First Try.

Operation is as follows: Assume initially the Q output of the 74LS107 is high and Q1 is turned on. Op amp A maintains a constant voltage drop across resistor R1. Since the input to an op amp (ideally) draws no current, a constant current flows through the 5 volt zener diode and out the collector of Q2. Since Q1 is turned on, this current is dumped directly to ground. A pulse on CS1 toggles the Q output, setting it low. This turns off Q1. The current now charges C1. The constant current into C1 causes a linear voltage increase which is output to the comparator by op amp C. When the maximum length of the ramp is

setting C1, R1, and the 1K ohm adjustable resistor. The problems which surfaced now were more subtle. Noise and jitter on the ramp were having an affect on the recorded signal.

Several additional steps were taken to increase the stability and decrease the noise in the circuit. Figure 15 shows the third iteration.

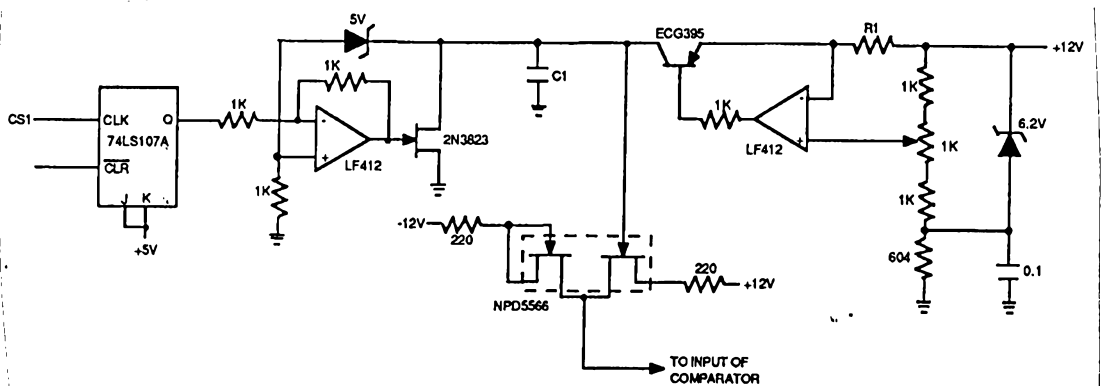


Figure 15. Ramp Circuit - Third Try.

A FET was substituted for the NPN bipolar transistor used to shunt the charging capacitor to ground. The FET provides a more linear transition into saturation or cutoff causing the voltage developed on C1 to be more linear. The original Op Amp B had a very slow slew rate (0.5 V/uSec). When the Q output of the 74LS107 would toggle, it would take roughly 2.5 microseconds for the op amp to switch the transistor Q1 fully on or off. An op amp with a slew rate of 10 V/uSec was substituted for this op amp. The FET could now be switched in roughly 0.5 microsecond. The 5 volt zener diode which limits the ramp height was also

moved to reduce extraneous noise on the output voltage as C1 charged.

This circuit represents the last design iteration and is incorporated in the Timing subsystem. This circuit meets the requirements of a linear 0 - 5 volt, 4 microsecond ramp.

3.6 Construction

There are two possible areas to concentrate on to eliminate noise and jitter from the system. The first is by optimizing the system hardware design while the second is by applying signal processing techniques to the recorded signal to remove or reduce any noise or jitter. This section describes the hardware construction to reduce noise. Section 5.1 describes signal processing techniques to reduce the noise.

The original data acquisition system was built on a proto-board type PC board. This board had obvious deficiencies. Primarily, this type of board is not intended to hold analog circuits, especially circuits requiring high speeds and high accuracy. The data acquisition system was redesigned onto a custom printed circuit board. This board was designed specifically to reduce noise and increase the accuracy of the high speed analog circuits. Additionally, more care could be taken in separating the analog and digital circuits to reduce cross talk. This board showed great improvement over the original. Figure 16 is a photograph of the final circuit board.

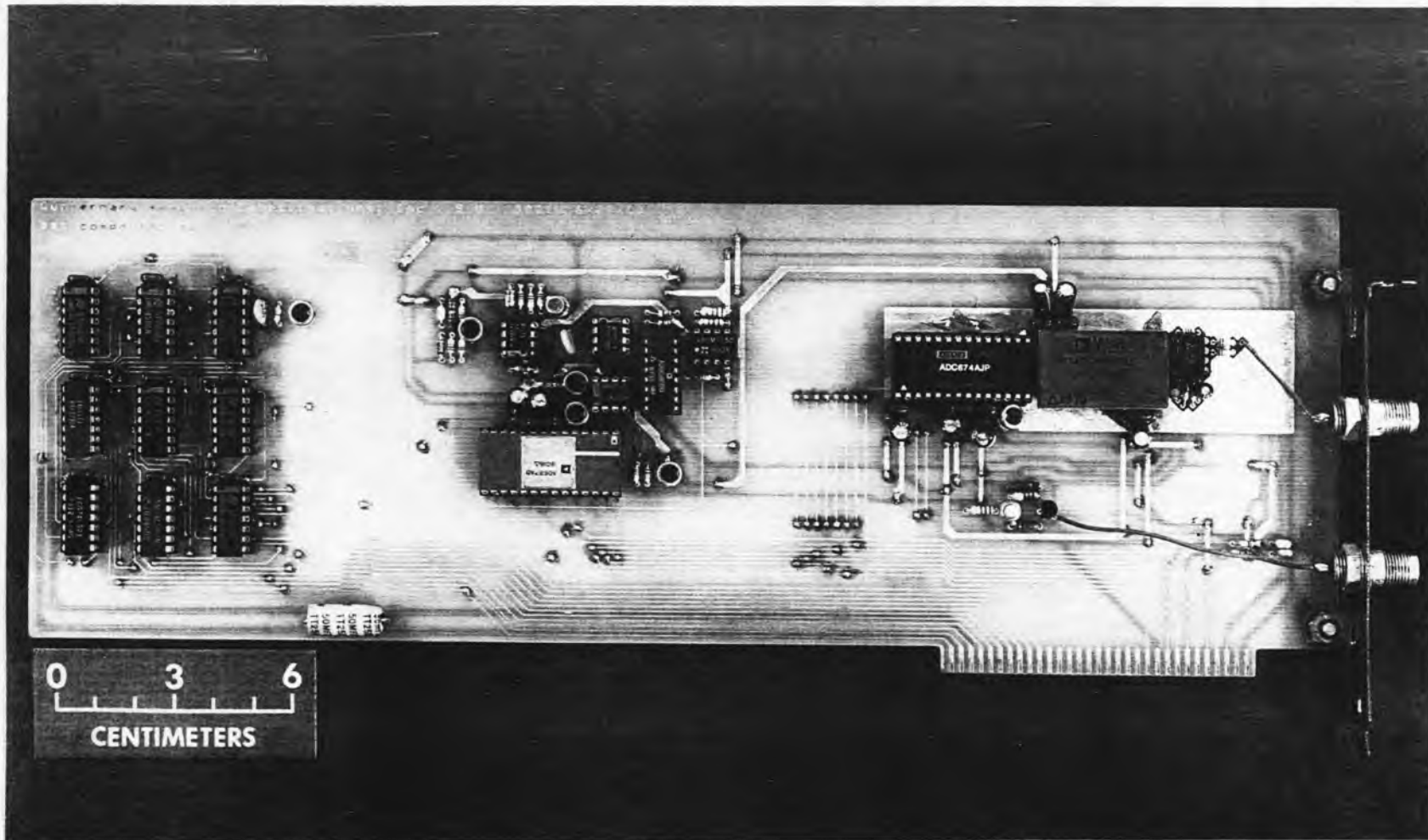


Figure 16. Fabricated System Circuit Board.

As the board was already designed and built, further design changes in the hardware would be costly in both time and money. For example, a known contributor to the time jitter in the recorded signal is from the jitter of the transmitter output (see Section 3.4). By changing the data conversion trigger to the actual transmitter output pulse versus the transmitter trigger pulse would eliminate the source of jitter. However, the cost would be too high with limited gain. Section 5.1 describes the application of signal processing techniques to improve the signal at minimal cost and with greater flexibility.

4. TESTING OF THE HARDWARE

To test the hardware system, numerous software modules were required. Each module was written to test a specific subsystem of the hardware. This chapter describes the software modules that were developed, the physical test methods for the hardware, and the results of the tests. Both quantitative and qualitative measurements are made of the system accuracy.

4.1 Hardware Testing Software

The hardware system software could be divided into two distinct areas. Software was developed to test each of the functional areas of the hardware system. Secondly, software was written for the actual waveform acquisition and storage. Each area is covered in the following sections.

4.1.1 Subsystem Testing Software

Software was written to test specific hardware subsystems. The software would excite the subsystem under test. An oscilloscope is used to monitor the circuit and where applicable the results of the operation are displayed on the computer screen. Programs were written to test the following subsystems: ramp circuit, DAC, trigger circuit, and the A/D converter.

The different subsystems are controlled by logic pulses generated by addressing specific I/O addresses. The DAC and A/D converter circuits utilize the data bus from the computer to

write to and read from the converters respectively. The other subsystems are controlled solely by pulses at the required addresses. The data lines are not utilized (data output while the address lines are activated are ignored). The following table lists the I/O address, the subsystem affected, and the result of activating that address.

<u>I/O ADDRESS</u>	<u>SUBSYSTEM</u>	<u>RESULT</u>
300 H	A/D Converter	Reads high order byte of 12-bit converter (D4-D11).
301 H	A/D Converter	Reads 12-bit converter lower 4 bits (D0-D3) shifted to high end of byte. Low end of byte is 0 filled.
302 H	DAC	Writes 4 high order bits (D8-D11) shifted to low end of byte to converter.
303 H	DAC	Writes low order byte (D0-D7) to 12-bit converter.
304 H	Trigger Circuit	Activates 74LS122 astable multivibrator. The rising edge of the pulse triggers the 74LS122. Duration of the output pulse is controlled by external

		resistor and capacitor combination.
305 H	Ramp Circuit	Toggles Q output of 74LS107A. Turns ramp ON to OFF or OFF to ON (Q high turns ramp on).
306 H	Ramp Circuit	Connected to CLEAR of 74LS107A. Sets Q low (turns ramp off).

A total of eleven testing programs were written. These could be grouped into three types. The first were individual programs which performed each step of the data acquisition process for one sample point. These programs are:

RESET - turns ramp off.

TOGGLE - switches ramp ON to OFF or OFF to ON.

SET_DAC - set DAC to specified voltage.

TRIGGER - initiates trigger output pulse.

CONVERT - start A/D converter conversion.

READ - read from A/D converter.

CONVERT2 - performs entire acquisition procedure for one sample.

The second type were programs written to perform selective subsystem tests. These tests would cycle numerous times to allow viewing of the logic waveforms to verify proper circuit

operation. These programs were:

TOGGLE2 - switches ramp ON to OFF/OFF to ON 5000 times.

READ2 - reads A/D converter 1000 times.

TRIG_TST - initiates trigger output pulse 1000 times.

The last program type was for calibrating the DAC. This program, called DAC_TST, provides the required digital inputs to the DAC to perform the zero and gain adjusts on the DAC. These adjustments ensure the DAC output operates over the proper range (0 - 5 volts).

Listings for each program are in Appendix A.

4.1.2 Signal Acquisition Software

The signal acquisition software is responsible for the actual waveform acquisition and storage. The individual programs which performed each step of a single data point acquisition, as written for the subsystem tests, could be incorporated as procedures in the waveform acquisition program. A total of 4 microseconds of waveform were to be stored as set forth in the original system requirements. The ramp was designed for a 0 to 5 volt rise in 4 microseconds. Since the DAC output was set for 0 to 5 volts also, all 12 bits (4096 discrete voltage levels) had to be used to record the entire 4 microseconds. The addition of these 96 samples changes the theoretical absolute maximum equivalent-time sampling resolution to 0.98 nanoseconds.

As most of the required acquisition procedures had been

developed during the testing stage, the majority of the signal acquisition software development involved writing user interfaces and developing waveform display, storage, and print routines.

A listing for the 1 GHz signal acquisition program is in Appendix A. This program is titled T1G12B. This is the program designed to meet the original data acquisition speed and accuracy requirements.

4.2 Hardware System Test

This section covers the individual hardware system tests, the operation of the system as a whole, and evaluation of the system operation and accuracy.

4.2.1 Individual Subsystems

Each subsystem was tested using the Subsystem Testing Software as described in Section 4.1.1. Actual testing is described in the following sections. Refer to Figure 11A and 11B for the test points referenced in the following sections.

4.2.1.1 Timing Subsystem

The ramp circuit is tested by placing an oscilloscope probe (Channel A) at Test Point (TP) 1 and a second probe (Channel B) at TP2. The program TOGGLE2 is executed. Channel A shows the pulse generated from the address decode logic for I/O address 305H. Channel B shows the actual ramp voltage signal. The ramp

turns on or off with each pulse on Channel A.

The DAC circuit is tested by placing a voltmeter probe at TP5. Executing the program SET_DAC allows the operator to specify the DAC output voltage. This voltage can be verified on the voltmeter.

The operation of the comparator is verified by placing an oscilloscope probe (Channel B) at TP6. Channel A's probe is moved to TP2. The DAC is set to a midrange value (using SET_DAC). TOGGLE2 is executed. Triggering on Channel A, the ramp is shown increasing. When the ramp reaches the DAC output level, the comparator will switch (Channel B). If these tests are successful, the entire Timing Subsystem is operational.

4.2.1.2 Transmitter Trigger Subsystem

The Transmitter Trigger Circuit is tested by attaching an oscilloscope probe (Channel A) at TP3 and a second probe (Channel B) at TP4. The program TRIG_TST is executed. This program addresses the transmitter trigger circuit repeatedly at I/O address 304H. Channel A will show a TTL pulse from the address decode logic. Channel B will show a 12 volt pulse approximately 8 microseconds long.

4.2.1.3 Data Acquisition Subsystem

The data acquisition subsystem is checked by applying known voltages to the signal input and performing the conversion.

Specifically, 0, +5, and +12 volts are applied to the input (from the system power supplies). The program CONVERT2 is executed with the result displayed on the terminal.

When all three subsystems are checked operational, the system is checked as a unit.

4.2.2 Integrated System Test

The entire system is tested by connecting all the components as shown in Figure 17.

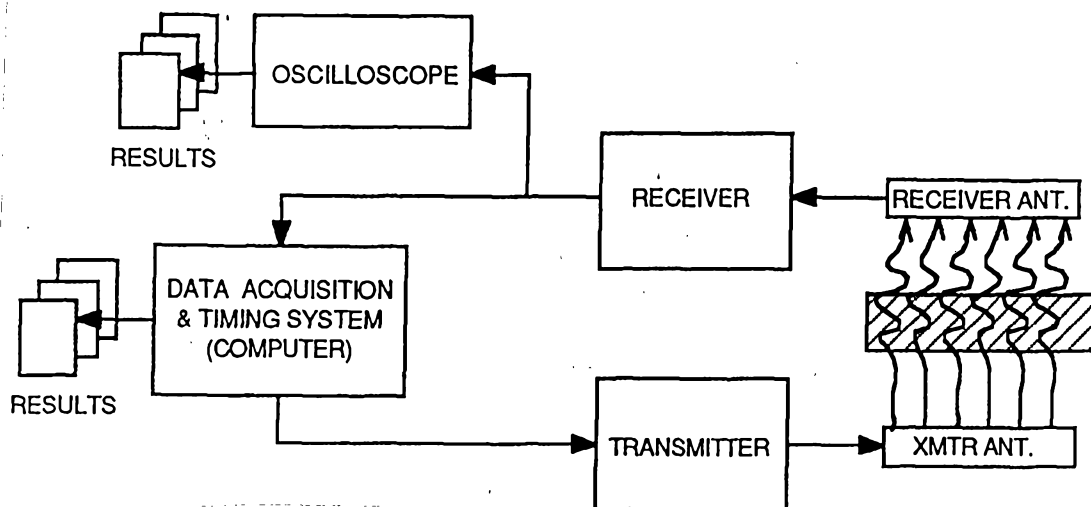


Figure 17. System Configuration for Evaluation and Test.

A real signal is recorded with the system and compared to its equivalent from an oscilloscope. See Figures 18 and 19. Note that the signals compare favorably.

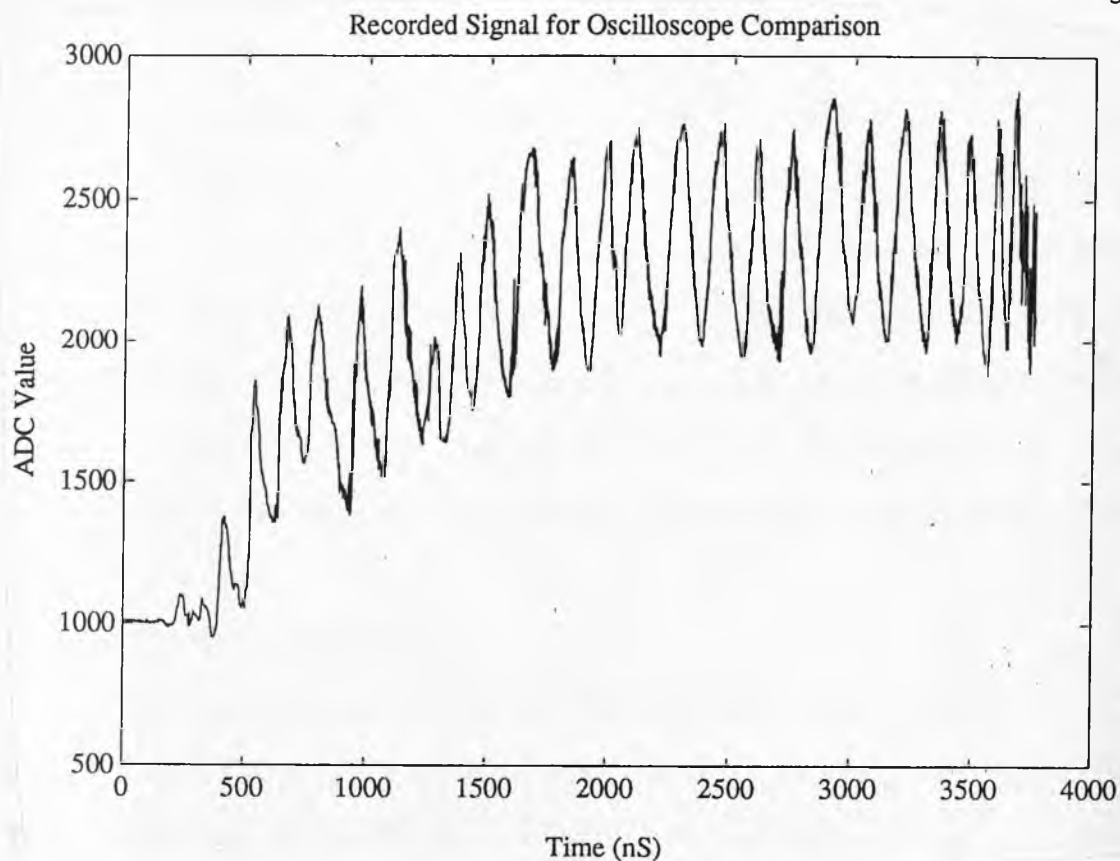


Figure 18. System Recorded Signal.

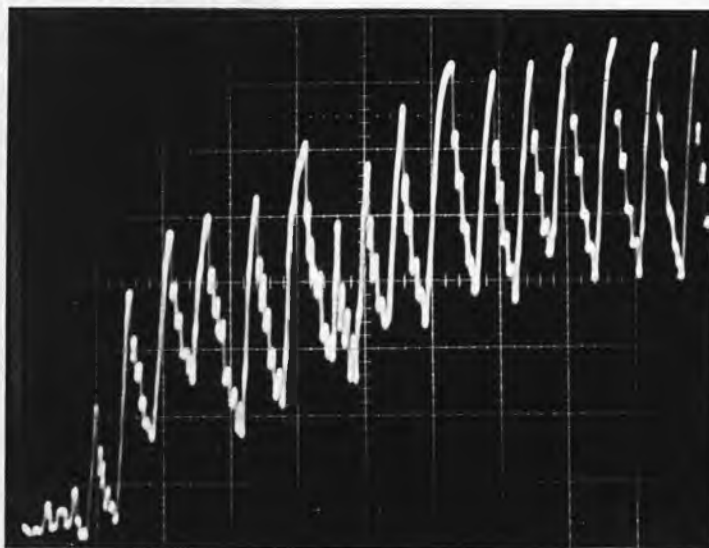


Figure 19. Oscilloscope Recorded Signal.

4.2.3 Evaluating System Operation and Accuracy

A quantitative measure of the accuracy of the data acquisition system is made by measuring the noise and jitter of the ramp used in the timing circuit. A qualitative measure of the accuracy of the entire system is made by sampling a real signal numerous times and checking for repeatability. A quantitative measure of the entire system accuracy is made also.

4.2.3.1 Ramp Evaluation

Measuring the accuracy of the internal ramp circuit of the data acquisition system shows the base accuracy of the system. The system can not be more accurate than the ramp circuit itself. The ramp accuracy was measured using two methods. By the first method, the ramp was recorded by the designed data acquisition system itself. The ramp signal at Test Point 2 was connected to the input of the system. Figure 20 shows the actual recorded ramp.

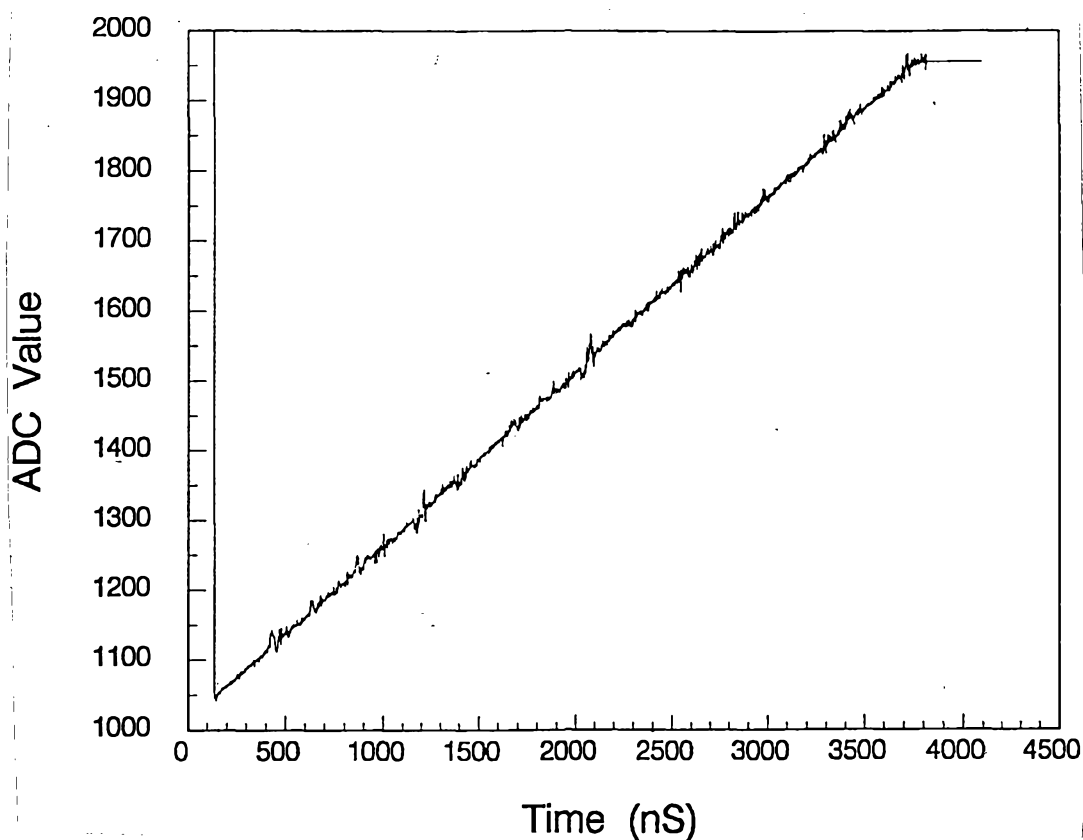


Figure 20. System Recording Internal Ramp Signal.

Comparing the recorded ramp to an ideal ramp, the following statistical values were found:

Standard deviation: 5.054,

Mean: 1.1015.

This standard deviation is 0.26% of full scale and is an acceptable value.

Keeping the same connection, the second method samples the ramp at the 25, 50, and 75 percent point of the ramp (in time) for one complete sample period (4096 data points) each. As the

sample should be taken from the exact same point on the ramp for each sample point in each set, the quality of the ramp can be derived at each point. Figures 21, 22, and 23 show the data collected for the 25, 50, and 75 percent points respectively. The 'Typical Data Set' in each figure represents a single data set of 4096 points which has had no filtering or averaging applied. The 'Averaged Data Set (4)' in each figure represents an average of four different single data sets, each with 4096 points.

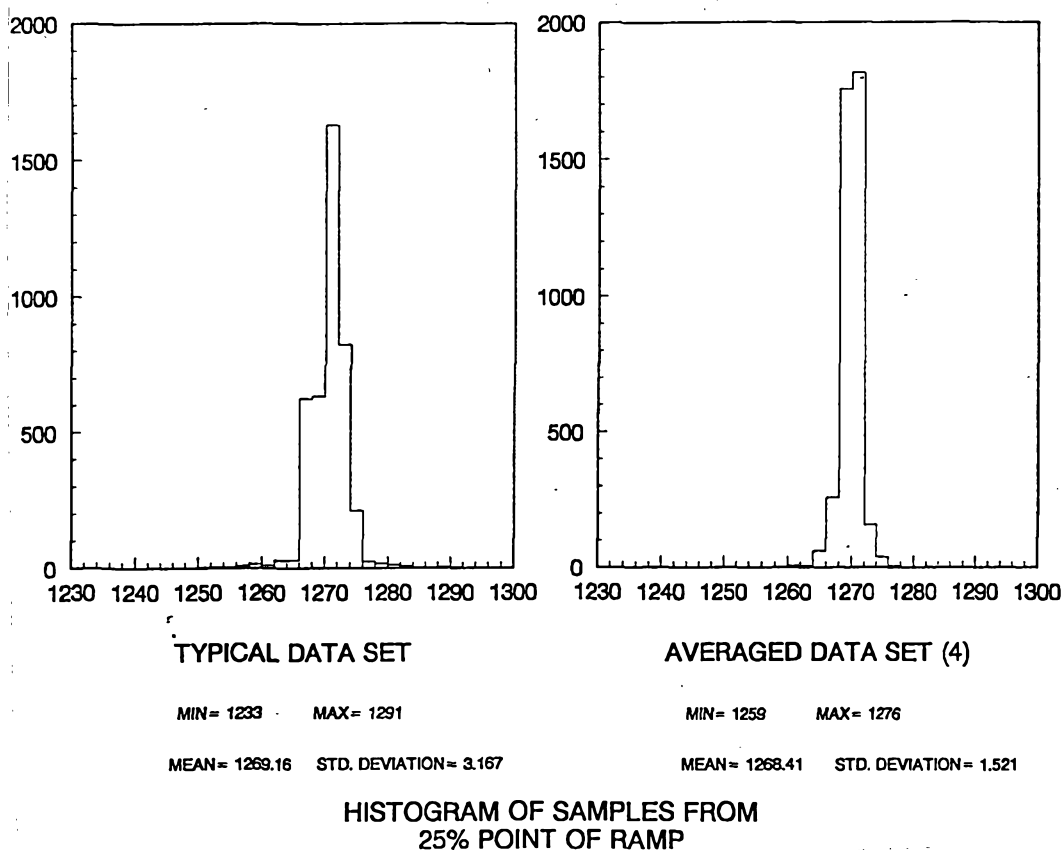


Figure 21. Statistical Analysis of Ramp at 25% Point.

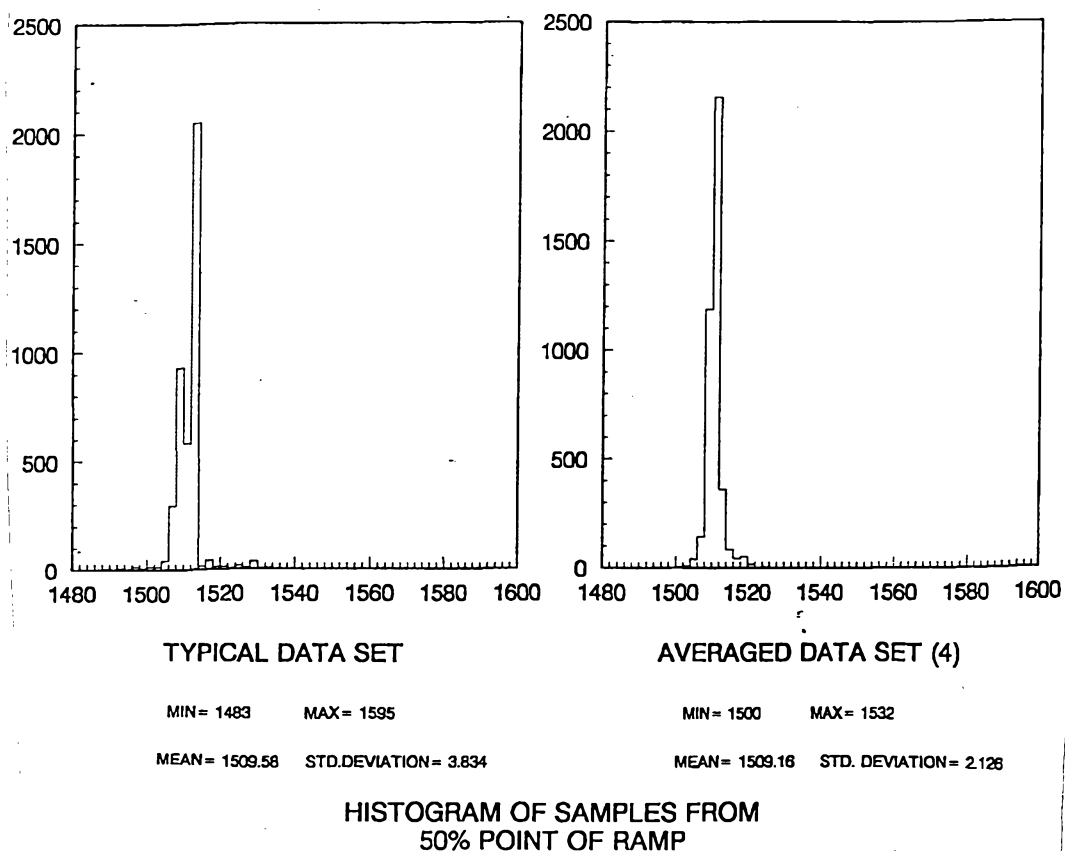


Figure 22. Statistical Analysis of Ramp at 50% Point.

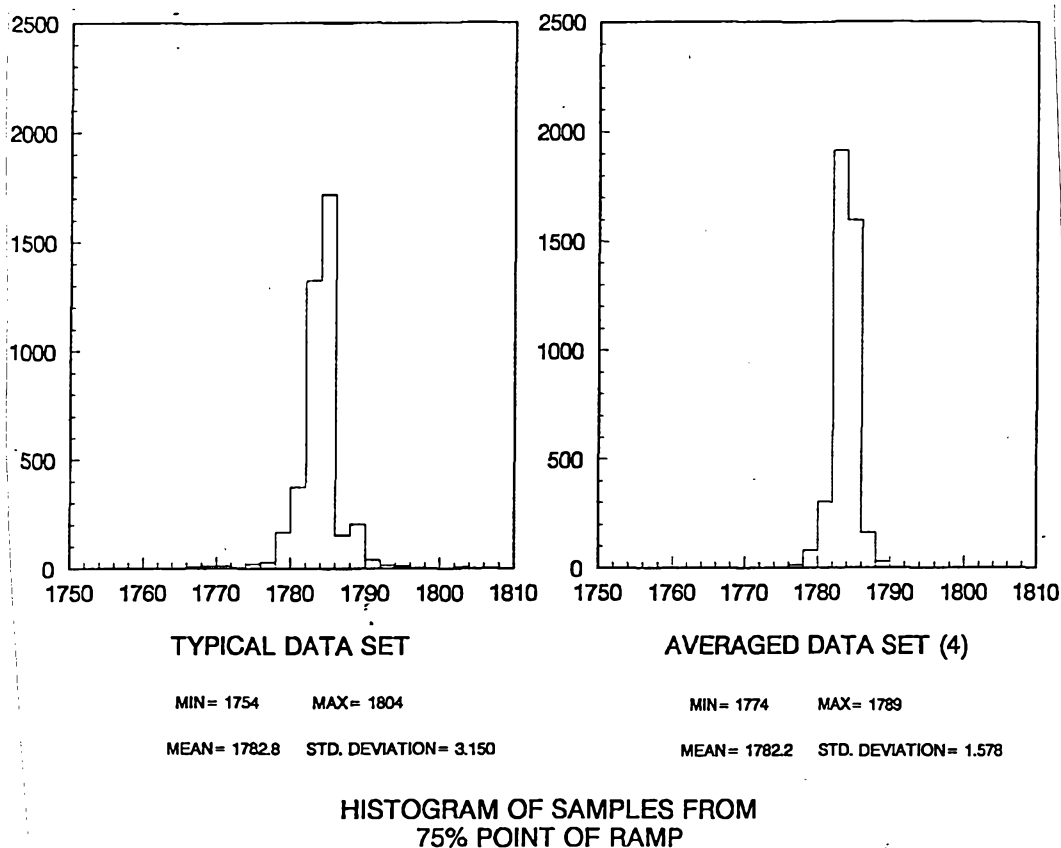


Figure 23. Statistical Analysis of Ramp at 75% Point.

These figures show that the ramp accuracy is within an acceptable range. Note also, that this is actually a worst case measurement of the ramp accuracy. Routing the ramp to the input of the data acquisition system places a load on the ramp and

comparator circuits which in normal operation would not be there. This loading actually degrades the recorded ramp and the accuracy of the timing system.

4.2.3.2 Real Signal Evaluation

The real signal evaluation actually measures the accuracy of the entire system. Inaccuracies in timing and the transmitter will be measured now. A qualitative measurement is made by comparing Figures 18 and 19. The signal is repeatable and compares very favorably with the oscilloscope record. A quantitative measurement is made by taking samples at the 25, 50, and 75 percent points on the signal as was performed on the ramp. Figure 24 shows the signal and location the data points were recorded from.

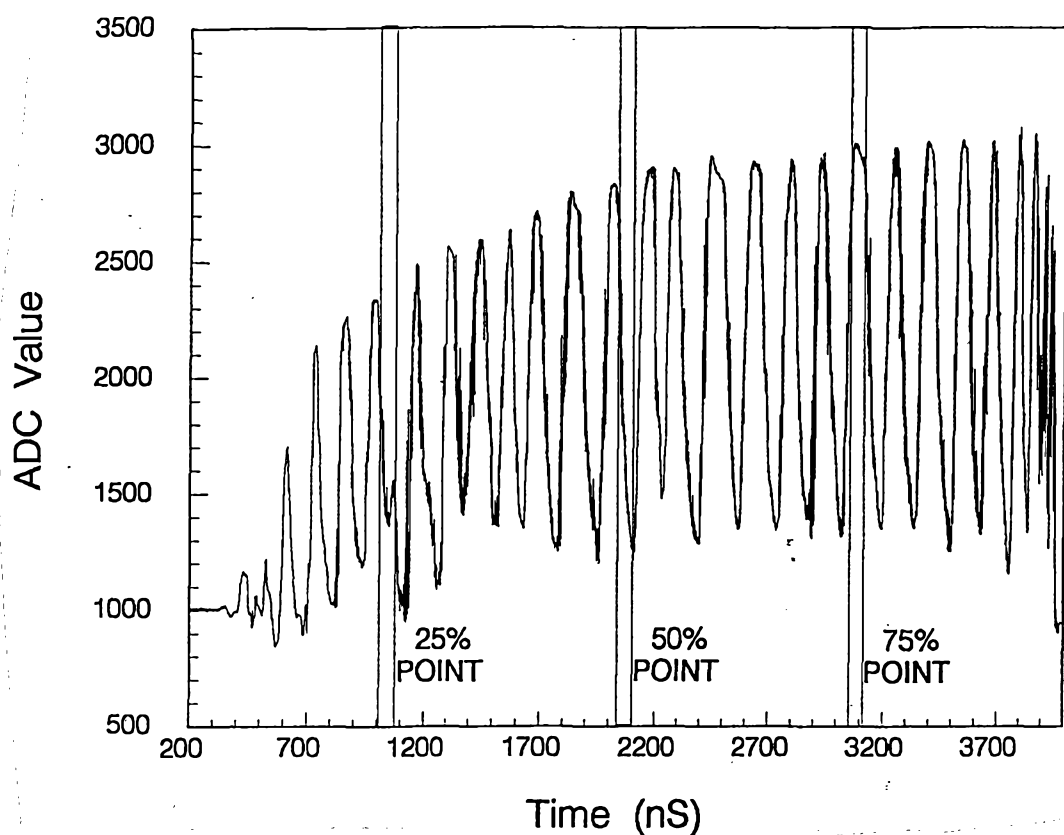


Figure 24. System Recording Signal for Statistical Analysis.

Figures 25, 26, and 27 show the data collected for the 25, 50, and 75 percent points respectively. Again, the 'Typical Data Set' in each figure represents a single data set of 4096 points which has had no filtering or averaging applied. The 'Averaged Data Set (4)' in each figure represents an average of four different single data sets, each with 4096 points.

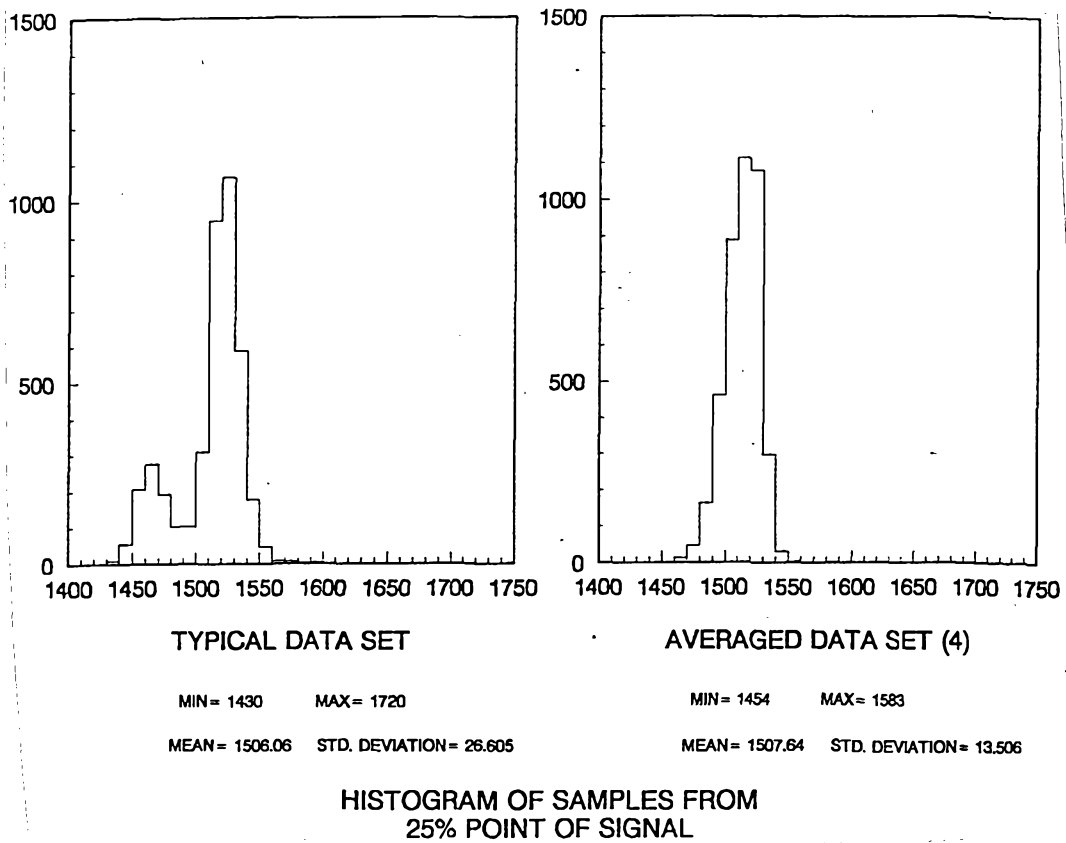


Figure 25. Statistical Analysis of Signal at 25% Point.

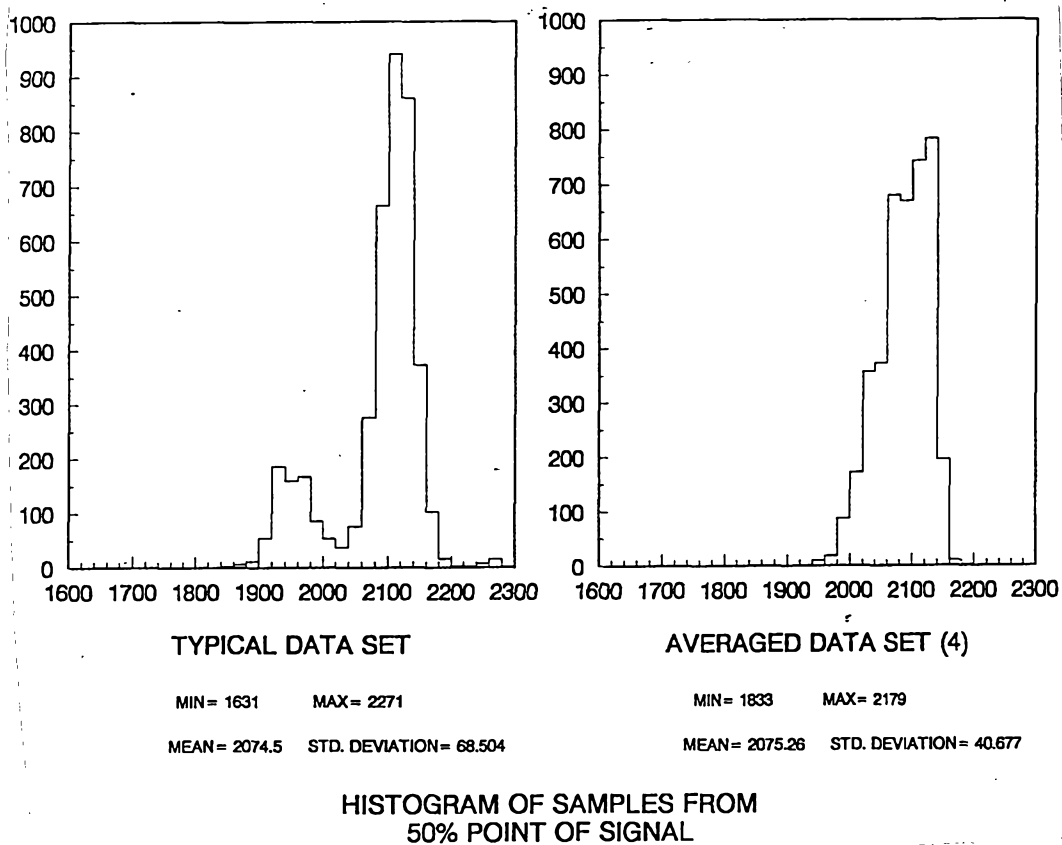


Figure 26. Statistical Analysis of Signal at 50% Point.

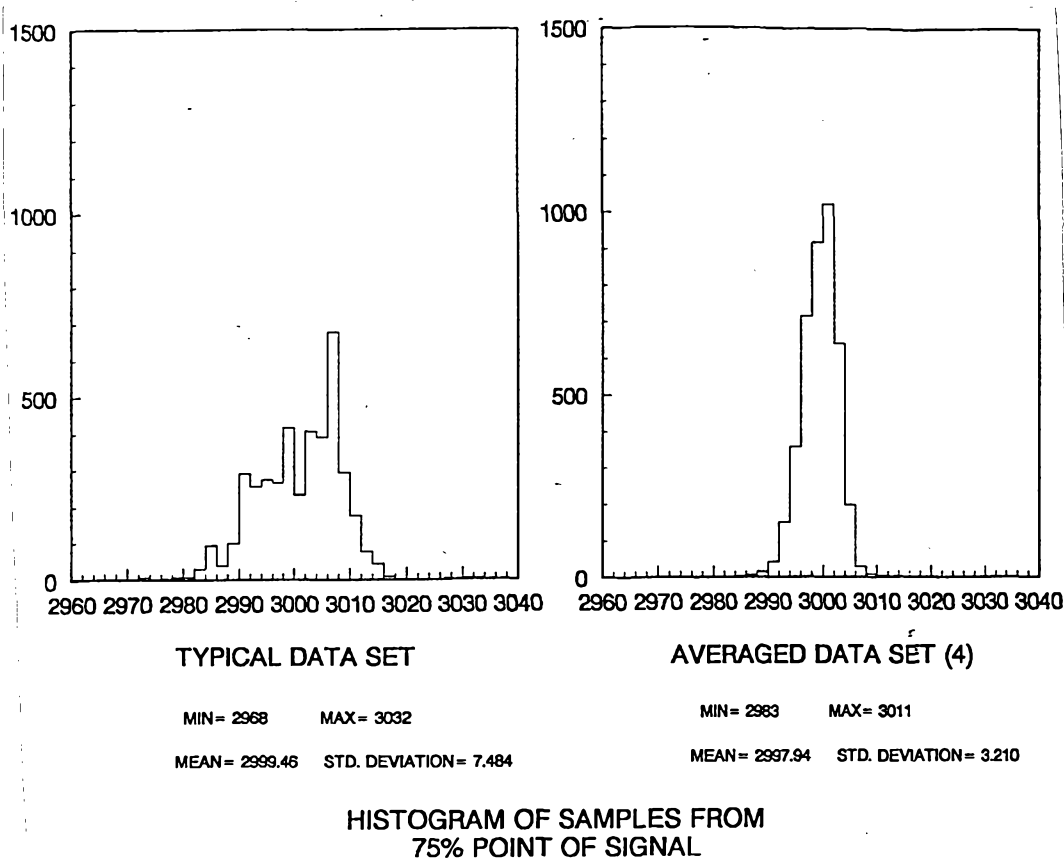


Figure 27. Statistical Analysis of Signal at 75% Point.

The entire system accuracy is within an acceptable range. As was expected however, the system accuracy is lower than the ramp accuracy. This can be attributed to the jitter inherent in the transmitter. Note also, that the accuracy at the 25 and 75 percent point look very good compared to the 50 percent point.

The signal is relatively flat where the system recorded the data at the 25 and 75 percent points. Data recorded in these regions will not show the true effects of time jitter. Data recorded at the 50 percent point is recorded on a portion of the signal which has a very large slope. The effects of time jitter are very evident by the larger standard deviation.

5. DATA PROCESSING AND DISPLAY SOFTWARE

The software for processing and performing computer interpretation was developed separately from the hardware system software. This software is also written to be independent of the actual hardware acquisition system. The only requirement placed on the data is that it meet the input requirements of the program. Two methods are provided to input the data. The data format and input requirements for each method are as follows:

Input Method 1

- 4096 data points/waveform
- 4 waveforms/station
- 4 waveforms stored on disk as single text file
- each set of 4096 points concatenated to previous set
- data values range from 0 to 4095.

Input Method 2

- 4096 data points/waveform
- 1 waveform/station
- 1 waveform stored on disk as single text file
- 1 set of 4096 points
- data values range from 0 to 4095.

The vast majority of the code written for the data processing and display software is related to user interfaces and data presentation. It will be described in subsequent sections. The specialized digital signal processing and application specific (actual interpretation) code will also be described.

The entire program is listed in Appendix B.

5.1 Signal Conditioning Software

Three methods are incorporated into the software to reduce the effects of noise on the recorded signal. These are: amplitude averaging, smoothing, and filtering. Averaging is automatically performed when a waveform is input via Input Method 1 described previously. Each waveform is read from the disk file and the corresponding data points added. Amplitude averaging is performed: the result of adding the corresponding data points from each waveform is divided by four. Smoothing is an optional procedure that averages data points that exceed a threshold limit with their neighbors. The threshold is set such that if the difference between two points is greater than the difference between 90% of the remaining points, that point is averaged with its neighbors.

Filtering is also an optional procedure. The following discussion describes the various filter types which were examined and the final choice of filter to incorporate in the software and apply to the recorded waveform.

The Fast Fourier Transform (FFT) of the signal in Figure 24 is shown in Figure 28. The FFT was performed on the waveform data from 1000 nanoseconds to 3047 nanoseconds.

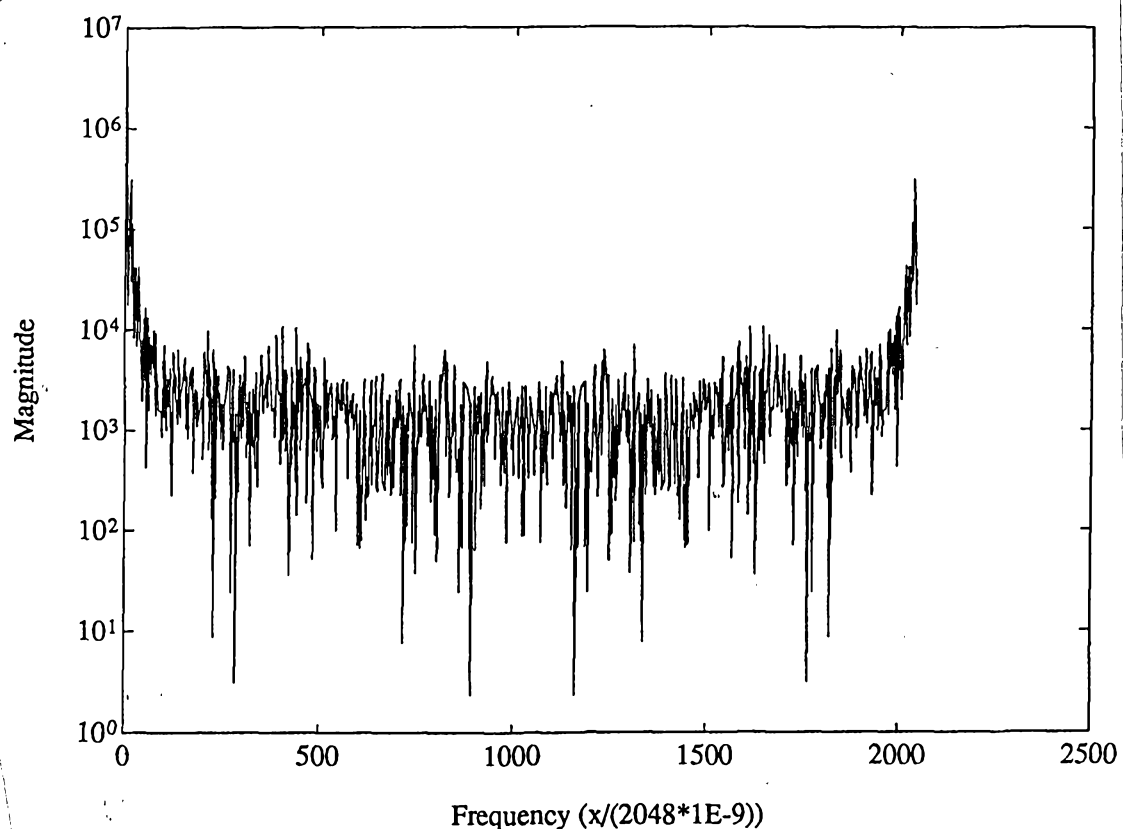


Figure 28. FFT of Recorded Signal.

The FFT shows that the majority of the spectrum is at and below 25 percent of the sample frequency. Several filter types were designed with varying order numbers and cut-off frequency. Figures 29 through 36 display representative results of several tests using different classes of filtering algorithms and cutoff frequencies. The figures show that setting too low a cut-off frequency may cause loss of significant information. As an illustrative example, Figure 29 shows the effect of reducing the

bandwidth of a 20th order Butterworth filter. It is apparent that setting $w_c = 0.2$ produces a signal that has little resemblance to the oscilloscope record (Figure 19). Conversely, setting $w_c = 0.5$ leaves too much corruptive noise.

The selection of the filtering algorithm also has a significant effect on the resulting signal.

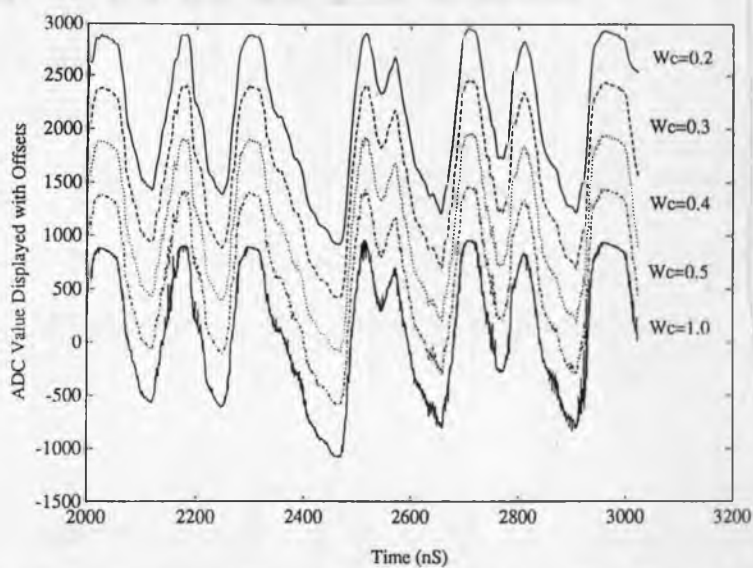


Figure 29. 20th Order Butterworth Filter.

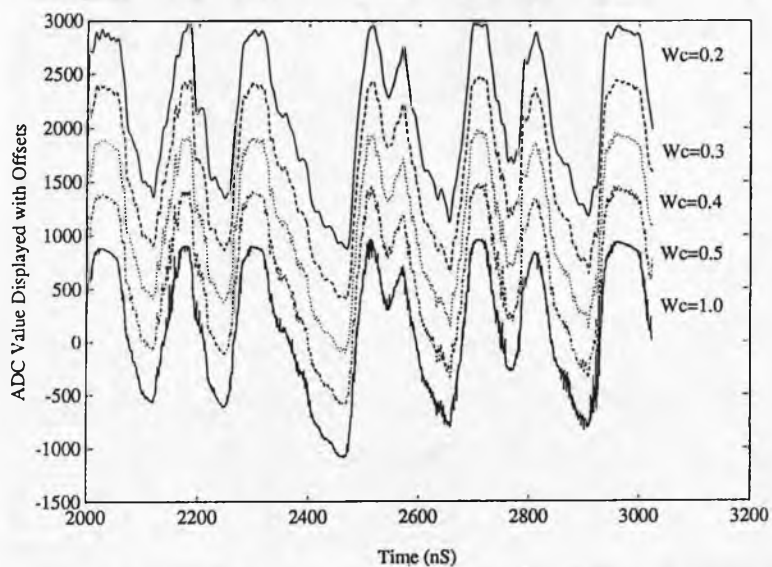


Figure 30. 10th Order Elliptic Filter.

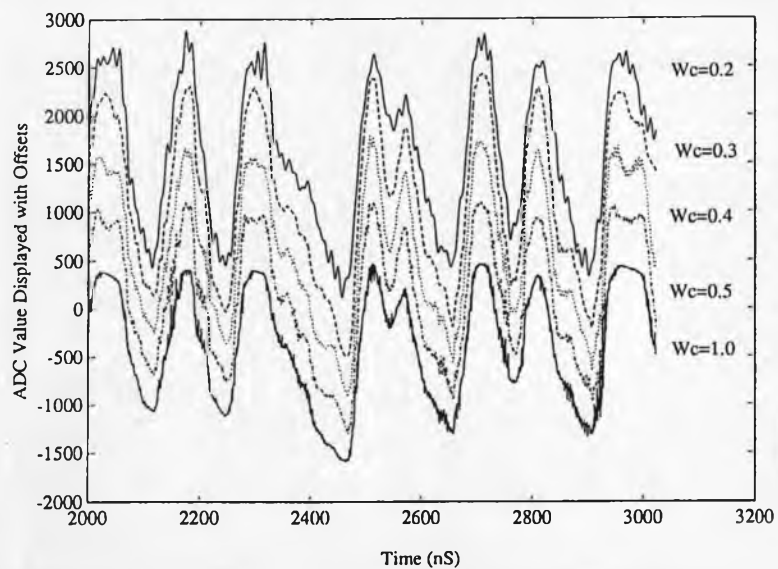


Figure 31. 20th Order Chebyshev Type 1 Filter.

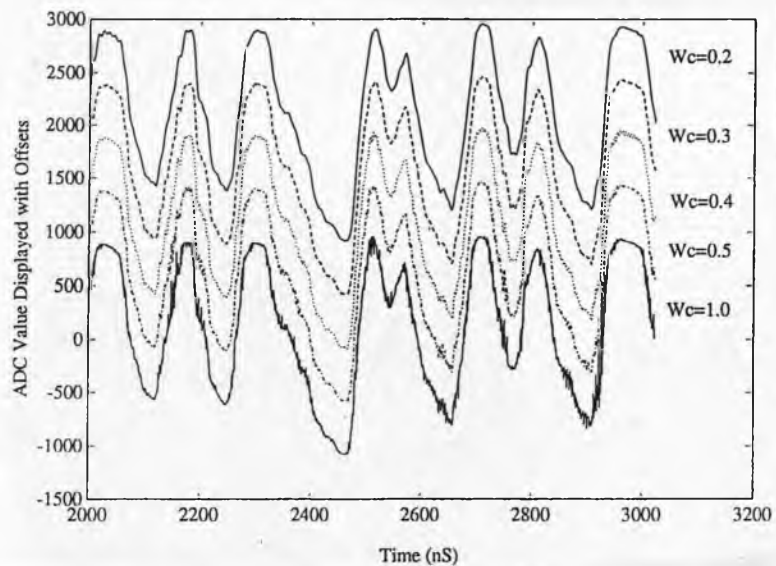


Figure 32. 20th Order Chebyshev Type 2 Filter.

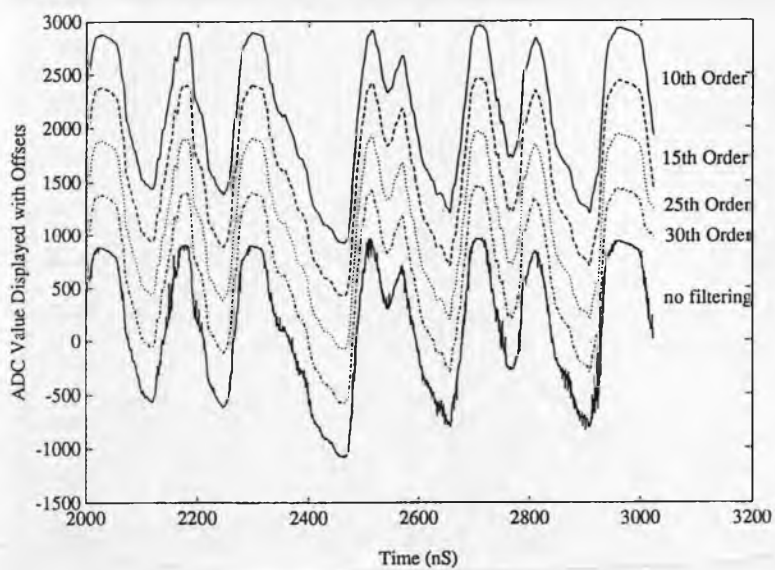


Figure 33. $w_c=0.30$ Butterworth Filter.

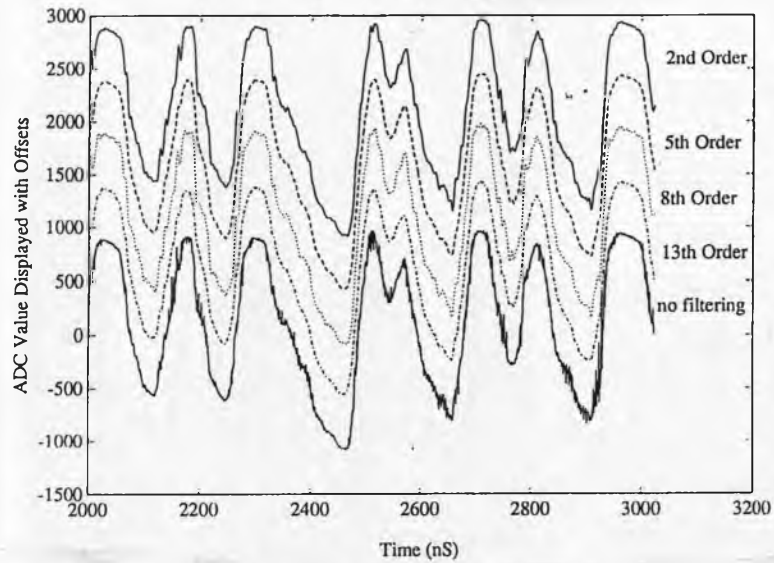


Figure 34. $w_c=0.30$ Elliptic Filter.

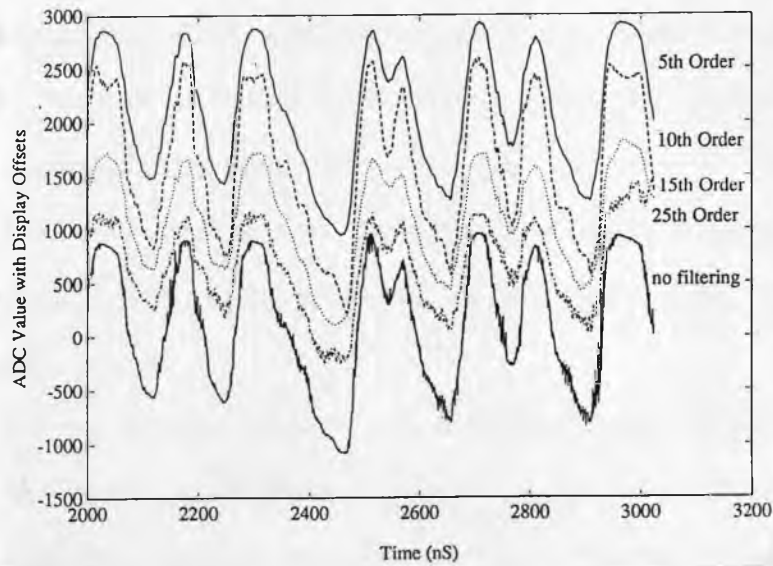


Figure 35. $w_c=0.30$ Chebyshev Type 1 Filter.

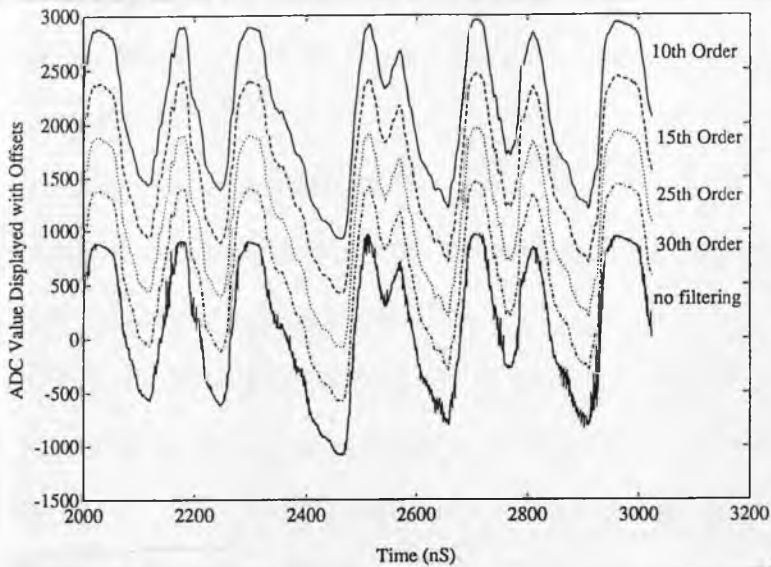


Figure 36. $w_c=0.30$ Chebychev Type 2 Filter.

The final selection of filtering algorithm has been made by visual comparison with the scope record. The complexity of a high order filter is also considered a strong disadvantage for single precision implementation in the software. Therefore, a 2nd order Elliptic filter was selected. Elliptic filters provide the same filtering with a lower order than the other filter types. [14]

Since the final result is highly subjective, the final implementation has a default cutoff frequency of 0.3. However, the user has the option of varying this frequency in a limited range.

Another processing technique was investigated also. Deconvolution of the time jitter from the recorded waveform was performed [7][10][12]. The time jitter was assumed to have a Gaussian pdf. The FFT of the pdf has the same form as the pdf:

$$\text{pdf} = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{t}{\sigma}\right)^2}$$

$$\text{Fourier Transform(pdf)} = e^{-\frac{1}{2}(2\pi f\sigma)^2}$$

The deconvolution procedure is described in the frequency domain by the following equation:

$$Y(f) = X(f)P(f),$$

where: $Y(f)$ = system response,

$X(f)$ = input waveform,

$P(f)$ = Fourier Transform of gaussian time jitter pdf.

Figure 37 shows the recorded waveform after applying the deconvolution technique with different standard deviation values.

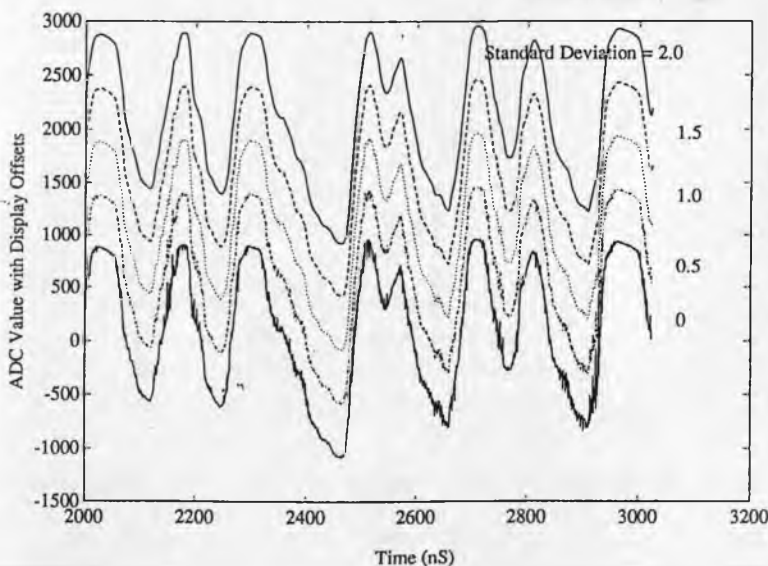


Figure 37. Deconvolution of Time Jitter.

The deconvolution technique did not provide an increase in resolution over that of filtering described previously. Also, the computing expense is much greater to perform deconvolution versus filtering in the time domain.

5.2 Data Interpretation Software

Interpretation of the recorded signal is generally straightforward. The midpoint of the waveform between each maximum and minimum is marked. The time (from a zero point at the beginning of the waveform) is recorded for each mark and then compiled for interpretation. Figure 38 illustrates this.

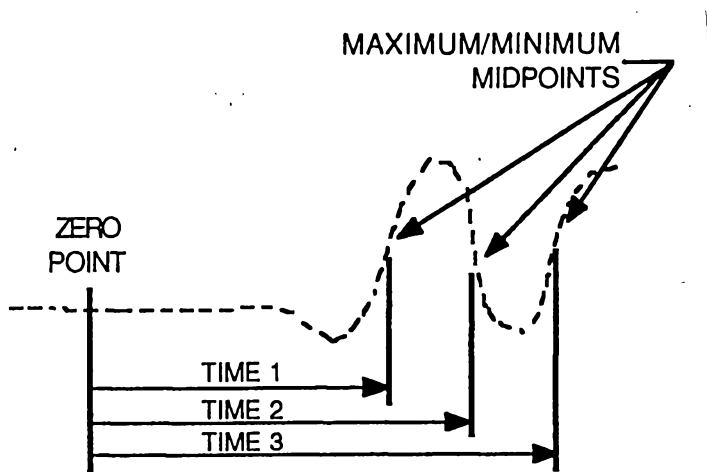


Figure 38. Data Interpretation Criteria.

The manual interpretation technique introduces significant

error. This error is due to variations in the initial center point picks and variations in the time measurements from the zero point.

The critical processing software, i.e., the software responsible for the center point pick, is written as an exchangeable procedure. The 4096-point array holding the filtered/averaged sample points is passed to the procedure. The procedure passes back a linked-list holding the position in the array of each center point pick. The procedure was written to be exchangeable so that future interpretation techniques, methods, or criterion could be written as procedures and easily substituted for the current procedure.

The technique used for the procedure incorporated in this version of the program finds each maximum and corresponding minimum. The midpoint value is then found between the maximum and minimum and added to the linked-list. The maximum or minimum is found by forming an average of a small section of the waveform and comparing this average to an average obtained by moving out the waveform. When the average peaks, a maximum or minimum has been encountered. This technique is illustrated in Figure 39.

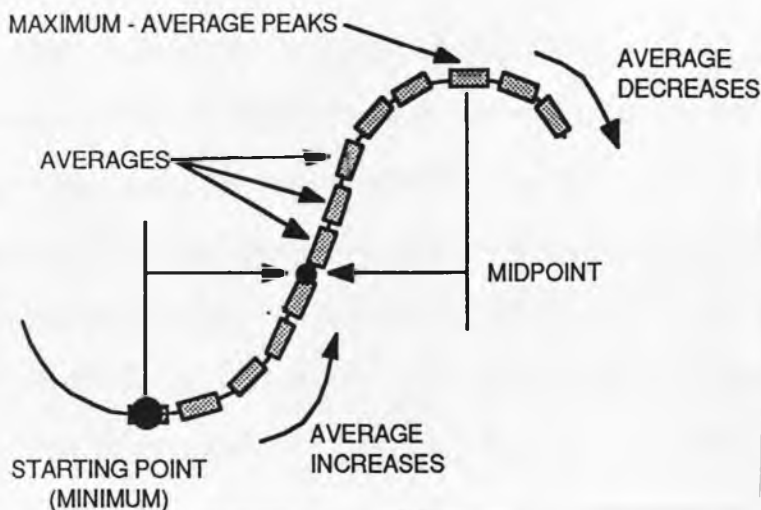


Figure 39. Processing Software - Midpoint Selection.

The midpoint between the maximum and minimum is taken as the waveform midpoint and added to the linked-list which is returned to the main program.

5.3 Data Display Software

The display software generates the graphical display of the waveform being processed. A graphical display of the computer generated interpretation can be viewed also. The display software provides the user with the facilities to edit the computer generated interpretation and see the results of the edit

immediately. The user can 'zoom' in on any selected section of the waveform to facilitate editing the interpretation.

5.4 The User Interface

Since the system is likely to be operated by occasional users, a menu driven interface was considered desirable. The overall structure of the menu tree is illustrated in Figure 40. The user manual for the data display and processing software is included in Appendix C. For convenience, the user manual can also be retrieved from within the program by selecting the 'Help' option from the first menu.

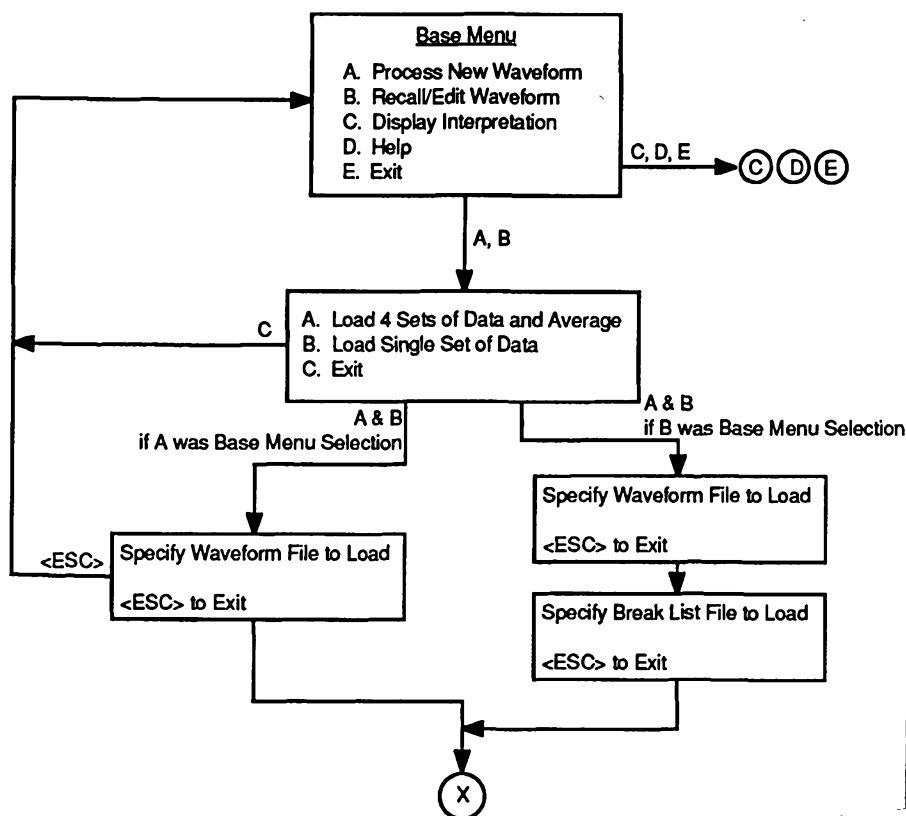


Figure 40. Data Processing and Display Software Menu Tree.

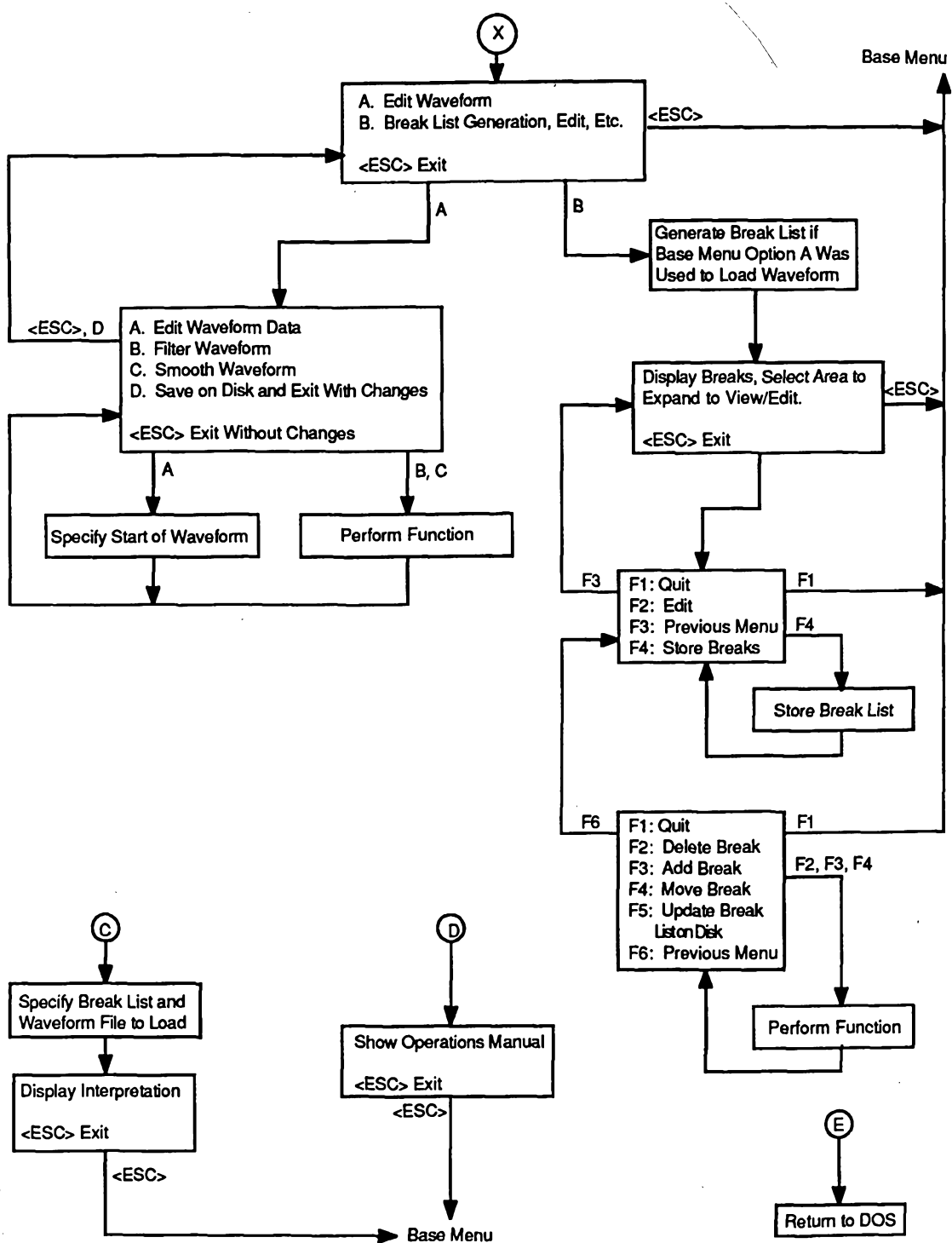


Figure 40. Data Processing and Display Software Menu Tree.

6. COMMENTS AND CONCLUSIONS

It has been successfully demonstrated that a data acquisition system for repetitive high frequency waveforms can be designed and constructed inexpensively. The accuracy of the constructed system meets the original requirements. While this system was designed to solve a specific problem (to eliminate an oscilloscope and Polaroid camera), the system should theoretically be able to record any repetitive waveform with the proper input and trigger/timing constraints.

The data display and processing software complements the hardware system by reducing noise on the signal. Averaging, smoothing, and filtering are all used to improve the signal quality. The use of a software algorithm to perform the interpretation removes the inconsistencies inherent in the manual method. The software could be adapted to other signal processing problems with the replacement or exchange of the interpretation specific procedures. The display and user interface sections could remain unchanged.

REFERENCES

- 1 P.R.Rigg, J.E.Carroll, C.Eng, "Low-Cost Computer-Based Time-Domain Microwave Network Analyser," IEE Proc., vol. 127, pt. H, no. 2, pp. 107-111, April 1980.
- 2 N.S.Nahman, "The Measurement of Baseband Pulse Rise Times of Less than 10^{-9} Second," Proc. IEEE, vol. 55, no. 6, pp. 855-864, June 1967.
- 3 N.S.Nahman, "Picosecond-Domain Waveform Measurements," Proc. IEEE, vol. 66, no. 4, pp. 441-454, April 1978.
- 4 N.S.Nahman, "Picosecond-Domain Waveform Measurement: Status and Future Directions," IEEE Trans. Instrum. Meas., vol. IM-32, no. 1, pp. 117-124, March 1983.
- 5 H.M. Cronson, P.G.Mitchell, "Time-Domain Measurements of Microwave Components," IEEE Trans. Instrum. Meas., vol. IM-22, no. 4, pp. 320-325, December 1973.
- 6 A.M.Nicolson, G.F.Ross, "Measurement of the Intrinsic Properties of Materials by Time-Domain Techniques," IEEE Trans. Instrum. Meas., vol. IM-19, no. 4, pp. 377-382, November 1970.
- 7 W.L.Gans, "The Measurement and Deconvolution of Time Jitter in Equivalent-Time Waveform Samplers," IEEE Trans. Instrum. Meas., vol IM-32, no. 1, pp. 126-133, March 1983.

- 8 B.J.Elliott, "System for Precise Observations of Repetitive Picosecond Pulse Waveforms," IEEE Trans. Instrum. Meas., vol. IM-19, no. 4, pp. 391-395, November 1970.
- 9 H.M.Cronson, A.M.Nicolson, P.G.Mitchell, "Extensions of Time Domain Metrology Above 10 GHz to Materials Measurements," IEEE Trans. Instrum. Meas., vol. IM-23, no. 4, pp. 463-468, December 1974.
- 10 W.L.Gans, J.R.Andrews, "Time Domain Automatic Network Analyzer for Measurement of RF and Microwave Components," NBS Tech. Note 672, September 1975.
- 11 "Technical Reference - Personal Computer," IBM Personal Computer Hardware Reference Library, April 1984.
- 12 N.S.Nahman, M.E. Guillaume, "Deconvolution of Time Domain Waveforms in the Presence of Noise," NBS Tech. Note 1047, October 1981.
- 13 E.O.Brigham, 'The Fast Fourier Transform and its Applications', Prentice-Hall, Englewood Cliffs, New Jersey, 1988.
- 14 L.R.Rabiner, B.Gold, 'Theory and Application of Digital Signal Processing', Prentice-Hall, Englewood Cliffs, New Jersey, 1975.

APPENDIX A

DATA ACQUISITION SOFTWARE LISTING

```

{
{          CONVERT
{
{          begin adc conversion
{
{          addresses location 300h (i/o) to start conversion
{
{

```

```
program CONVERT;
```

```
begin
    inline($ba/$00/$03/      { mov dx, 300h          }
        $ee);                { out dx, al            }
end.
```

```

{                                     }
{             CONVERT2               }
{                                     }
{             Test program for adc chip      }
{                                     }
{             performs all steps for one data point acquisition }
{                                     }

```

program CONVERT2;

```

var
    voltagep,           { actual adc word      }
    loop                : word;                { counter variable  }
    voltage              : real;                { actual adc voltage }

begin { TEST program convert2 }

    { reset ramp }

    inline ($ba/$06/$03/ { mov dx, 306h }
    $ee);               { out dx, al }

    { set dac }

    inline ($ba/$03/$03/ { mov dx, 303h }
    $b8/$08/$00/        { mov ax, 800h }
    $ee/                { out dx, al }
    $86/$c4/            { xchg al, ah }
    $4a/                { dec dx }
    $ee);               { out dx, al }

    { trigger ramp }

    inline ($ba/$05/$03/ { mov dx, 305h }
    $ee);               { out dx, al }

    { wait }

    for loop := 1 to 40 do
        inline ($90);    { nop }

        { reset ramp }

        inline ($ba/$05/$03/ { mov dx, 305h }
        $ee);               { out dx, al }

        { wait }

        for loop := 1 to 20 do
            inline ($90);    { nop }

            { read conversion }

            inline ($ba/$00/$03/ { mov dx, 300h }
            $ec/                { in al, dx }
            $86/$c4/            { xchg al, ah }
            $42/                { inc dx }
            $ec/                { in al, dx }
            $b1/$04/            { mov cl, 04h }
            $d3/$e8/            { shr ax, cl }
            $a3/voltagep);      { mov voltagep, ax }

```

```
        voltage := voltagep * 4.88e-3;  
        writeln (voltage:10:6);  
end. { TEST program convert2 }
```



```

readln (ans);
writeln ('*****');
if upcase (ans) = 'Y' then
  repeat
    inline ($ba/$03/$03/           { mov dx, 303h           }
            $b8/$00/$00);          { mov ax, 0000h        }
    for loop := $0000 to $0fff do
      inline ($ee/                  { out dx, al           }
              $86/$c4/              { xchg al, ah          }
              $4a/                  { dec dx               }
              $ee/                  { out dx, al           }
              $86/$c4/              { xchg al, ah          }
              $40/                  { inc ax               }
              $42);                { inc dx               }
    write ('Repeat DAC test? y/n ');
    readln (ans);
    writeln ('*****');
  until upcase (ans) = 'N'
end. { TEST program dac_tst }

```



```

{                                     }
{           READ                     }
{                                     }
{           Test program to read adc }
{                                     }
{           addresses locations 300h and 301h for hi and lo }
{           bytes                    }
{                                     }

```

```
program READ;
```

```
var
    voltage : real;           { a/d converter value }
```

```
function adc:word;
    inline ($ba/$00/$03/      { mov dx, 300h      }
           $ec/              { in al, dx    }
           $86/$c4/          { xchg al, ah  }
           $42/              { inc dx      }
           $ec/              { in al, dx    }
           $b1/$04/          { mov cl, 04h  }
           $d3/$e8);         { shr ax, cl   }
```

```
begin
    voltage := adc * 4.88e-3;
    writeln (voltage:10:6);
end.
```

```

{                                     }
{         READ2                       }
{                                     }
{         Test program to read adc    }
{                                     }
{         addresses locations 300h and 301h for hi and lo }
{         bytes                       }
{                                     }
{         reads converter 1000 times  }
{                                     }

```

```
program READ2;
```

```

var
  loop      : integer;           { counter variable   }
  voltage    : real;             { a/d converter value }

```

```

function adc:word;
  inline ($ba/$00/$03/           { mov dx, 300h       }
         $ec/                   { in al, dx         }
         $86/$c4/               { xchg al, ah       }
         $42/                   { inc dx            }
         $ec/                   { in al, dx         }
         $b1/$04/               { mov cl, 04h       }
         $d3/$e8);              { shr ax, cl        }

```

```

begin
  for loop := 1 to 1000 do
    begin
      voltage := adc * 4.88e-3;
      writeln (voltage:10:6);
    end;
  end.

```

```

{                                     }
{           RESET                     }
{                                     }
{   reset ramp program               }
{                                     }
{   addresses location 306h (i/o) to reset ramp circuit }
{                                     }
{   turns ramp OFF                   }
{                                     }

```

```
program RESET;
```

```

begin
    inline($ba/$06/$03/             { mov dx, 306h      }
        $ee);                       { out dx, al       }
end.

```

```

{
{
    SET_DAC
{
    set dac output voltage
{
    address location 302h, 303h (i/o) for low and high
{
    byte. user inputs desired voltage (0 to +5 volts)
{
}
}
}
}
}
}

```

```

program SET_DAC;

```

```

var
    voltage      : real;           {selected o/p voltage }
    out_voltage  : word;           {word output to dac  }

begin
    repeat
        writeln ('Input the voltage level for the DAC to output: ');
        writeln;
        readln (VOLTAGE);
        if (voltage > 4.996) or (voltage < 0) then
            writeln ('Input voltage is out of range');
        until (voltage >= 0) and (voltage <= 4.996);
        out_voltage := round(voltage / 1.22e-3);
        inline($ba/$03/$03/           { mov dx, 303h           }
               $ee/                    { out dx, al           }
               $86/$c4/                { xchg al, ah          }
               $4a/                    { dec dx             }
               $ee);                   { out dx, al           }
        writeln (out_voltage:10);
    end.

```



```

{
{          TOGGLE2
{
{          toggles ramp:  program cycles 10000 times
{
{          addresses location 305h (i/o) to toggle ramp circuit
{
}
}
}
}
}

program TOGGLE2;

var
  loop,                { counter variable      }
  loop1  :  integer;    { counter variable      }

begin
  for loop := 1 to 10000 do
    begin
      inline($ba/$05/$03/      { mov dx, 305h      }
        $ee);                 { out dx, al       }
      for loop1 := 1 to 1000 do
        inline($90);           { nop              }
      end;
    end;
  end.

```

```

{
{
    TRIG_TST
}
{
    Test program for trigger circuit
}
{
    contents of 'al' register are unchanged for this
    operation and are not important as address is used as
    strobe. data lines are not utilized.
}
{
    Oscilloscope connections for test: Test-point #3
    provides trigger, Test-point #4 is 'trigger pulse'.
}
{
    Loop repeats 1000 times. User prompted for continue y/n
}
}

```

```
program TRIG_TST;
```

```

var
    loop      : integer;           { counter variable   }
    ans       : char;              { keyboard prompt  }

begin { TEST program trig_tst }
    repeat
        for loop := 1 to 1000 do
            begin { for }
                writeln (loop:5);
                inline ($52/
                        $ba/$04/$03/
                        $ee/
                        $5a);
                                { push dx           }
                                { mov dx, 304h        }
                                { out dx, al          }
                                { pop dx             }
            end; { for }
            read (ans);
            until upcase (ans) = 'Y'
        end. { TEST program trig_tst }

```



```

{
{
    program for data acquisition
{
    1 GHz      12 BIT
{
}
}
}

program TlG12B;
uses dos, crt, graph;

type
    single      = array[1..4,1..4096] of word;
    job_string   = string[20];

var
    store,
    finished     : boolean;           { store session variable }
    ans          : char;              { continue? variable }
    waveform     : single;            { continue? answer }
    job          : job_string;        { waveform array }
    shot,
    loop         : integer;           { job name }
                                       { counter variable }

procedure reset;
    inline($ba/$06/$03/           { mov dx, 306h }
    $ee);                         { out dx, al }

procedure set_dac (voltage:word);
    inline($58/                   { pop ax }
    $ba/$03/$03/                 { mov dx, 303h }
    $ee/                         { out dx, al }
    $86/$c4/                     { xchg al, ah }
    $4a/                         { dec dx }
    $ee);                       { out dx, al }

procedure trigger;
    inline($ba/$04/$03/           { mov dx, 304h }
    $ee);                       { out dx, al }

procedure micro_delay (m_delay:integer);

var
    loop        : integer;          { counter variable }

begin
    for loop := 1 to m_delay do
        inline($90);               { nop }
    end;

function adc:word;

```

```

inline($ba/$00/$03/      { mov dx, 300h      }
    Sec/                  { in al, dx        }
    $86/$c4/              { xchg al, ah      }
    $42/                  { inc dx          }
    Sec/                  { in al, dx        }
    $b1/$04/              { mov cl, 04h      }
    $d3/$e8);             { shr ax, cl      }

```

```

procedure acquire_waveform (var waveform:single; shot:integer);

```

```

var
    count      : integer;      { counter variable      }
    voltage    : word;         { actual input voltage  }
    finished   : boolean;      { entire signal recorded?}

```

```

begin {procedure acquire_waveform}
    writeln;
    count := 0;
    finished := false;
    voltage := $0;
    repeat                                { until finished      }
        inc (count);
        set_dac (voltage);
        inc (voltage);
        trigger;
        micro_delay(40);
        reset;
        micro_delay(20);
        waveform[shot,count] := adc;
        if (count > 200) and (waveform[shot,count] = 4095) then
            begin
                count := 0;
                voltage := 0;
                write ('*');
            end;
        if count = 4096 then
            finished := true;
    until finished;
    writeln;
    set_dac ($0);
end; {procedure acquire_waveform}

```

```

procedure store_on_disk (var waveform:single; job:job_string);

```

```

var
    loop1,
    loop      : integer;      { counter variable      }
    waveform_file : text;     { file to store waveform}
    file_name,  : text;       { name of file to store }
    path        : job_string; { path name              }

```

```

begin {procedure store_on_disk}
    path := 'a:\' + job + '\';
    write ('Enter name of file to store waveform under: ', path);
    readln (file_name);
    file_name := path + file_name;
    writeln (file_name);

```

```

assign (waveform_file, file_name);
rewrite (waveform_file);
for loop1 := 1 to 4 do
  for loop := 1 to 4096 do
    write (waveform_file, waveform[loop1,loop]:6);
  close (waveform_file);
end; {procedure store_on_disk}

```

```

procedure show_waveform (var waveform:single);

```

```

var
  graphdriver,
  graphmode,
  window      : integer;
  in_char     : char;
  avg,
  finished    : boolean;

```

```

procedure next(var window:integer; var waveform:single);

```

```

var
  loop1,
  loop      : integer;

```

```

begin
  clrscr;
  outtextxy (1,2,' E:exit A:average P:print file ->:next
                                window <=:previous window');
  case window of
    1,2,3,4,5 : for loop1 := 1 to 4 do
                  for loop := 0 to 639 do
                    putpixel(loop, waveform[loop1,(loop+1)
                                + (window*640)] div 12, red);
                  6 : for loop1 := 1 to 4 do
                        for loop := 0 to 255 do
                          putpixel(loop, waveform[loop1,(loop+1)
                                                  + (window*640)] div 12, red);
                        end;
                  inc(window);
  end;
end;

```

```

procedure previous(var window:integer; var waveform:single);

```

```

var
  loop1,
  loop      : integer;

```

```

begin
  clrscr;
  outtextxy (1,2,' E:exit A:average P:print file ->:next
                                window <=:previous window');
  for loop1 := 1 to 4 do
    for loop := 0 to 639 do
      putpixel(loop, waveform[loop1,(loop+1)+((window-2)*640)]
                                div 12, red);
    dec(window);
  end;
end;

```

```

procedure average (var waveform:single);

```

```

var
  loop,
  loop1      : integer;

begin
  for loop := 1 to 4096 do
    begin
      for loop1 := 2 to 4 do
        waveform[1, loop] := waveform[1, loop] + waveform[loop1, loop];
        waveform[1, loop] := waveform[1, loop] div 4;
        for loop1 := 2 to 4 do
          waveform[loop1, loop] := waveform[1, loop];
        end;
      end;
    end;

  procedure print_waveform (var waveform:single; avg:boolean);

  var
    print_file    : text;
    loop,
    loop1,
    loop2,
    loop3,
    exp_count,
    data_print,
    min,
    max           : integer;
    found         : boolean;

    function exponent (y,x:integer):integer;

    var
      temp,
      loop      : integer;

    begin
      temp := 1;
      for loop := 1 to x do
        temp := temp * y;
      exponent := temp;
    end;

  begin
    if not avg then
      assign (print_file, 'tlgl2b.sig')
    else
      assign (print_file, 'tlgl2b.ave');
    rewrite (print_file);
    write (print_file, chr(27), chr(13), 'P', chr(27), '@',
          chr(27), 'x', chr(0), chr(13));

    for loop := 0 to 4095 do
      if (loop mod 8) = 0 then
        begin
          max := 0;
          min := 684;
          for loop1 := 1 to 4 do
            for loop2 := loop to (loop + 7) do
              begin
                if max < (waveform[loop1, loop2 + 1] div 6) then
                  max := waveform[loop1, loop2 + 1] div 6;
                if min > (waveform[loop1, loop2 + 1] div 6) then
                  min := waveform[loop1, loop2 + 1] div 6;
              end;
            end;
          end;
        end;
      end;
    end;
  end;

```

```

write (print_file, chr(27), 'K', chr(175), chr(2), chr(0));
for loop1 := (max + 1) to 683 do
  write (print_file, chr(0));
for loop1 := max downto min do
  begin
    data_print := 0;
    exp_count := 0;
    for loop2 := (loop + 7) downto loop do
      begin
        found := false;
        for loop3 := 1 to 4 do
          if ((waveform[loop3, loop2 + 1] div 6) =
              loop1) and (not found) then
            begin
              found := true;
              data_print := data_print + exponent(2, exp_count);
            end;
            exp_count := exp_count + 1;
          end;
        if data_print = 26 then
          data_print := 25
        else if data_print = 9 then
          data_print := 10;
        write (print_file, chr(data_print));
      end;
    for loop1 := 0 to (min + 1) do
      write (print_file, chr(0));
    write (print_file, chr(0), chr(0), chr(27), 'J', chr(23), chr(27), '<');
  end;
  close (print_file);
end;

begin
  graphdriver := ega64;
  graphmode := egah1;
  initgraph (graphdriver, graphmode, 'c:\tpas\');
  clrscr;
  window := 2;
  avg := false;
  previous (window, waveform);
  finished := false;
  repeat
    in_char := readkey;
    case ord(in_char) of
      77 : case window of
            1,2,3,4,5,6 : next(window, waveform);
            7 : ;
          end;
      75 : case window of
            1 : ;
            2,3,4,5,6,7 : previous(window, waveform);
          end;
      69,101 : finished := true;
      65, 97 : begin
        average (waveform);
        avg := true;
        case window of
          1 : begin
              window := 2;
              previous(window, waveform);
            end;
          2,3,4,5,6,7 : begin
              window := window - 1;

```

```

                                next(window, waveform);
                                end;
                                end;
                                end;
80,112 : print_waveform (waveform, avg);
end;
until finished;
closegraph;
clrscr;
end;

begin { main program }
    clrscr;
    write ('Will this session be stored on disk? Y/N ');
    ans := readkey;
    writeln;
    if upcase(ans) = 'Y' then
        begin
            store := true;
            write ('Enter name of job (name length cannot exceed 5 characters): ');
            readln(job);
            writeln;
            writeln ('Install formatted disk in drive "a:".');
            writeln;
            writeln ('Hit any key to continue');
            ans := readkey;
            writeln;
            mkdir('a:\' + job);
        end
    else
        store := false;
        writeln;
        inline ($fa;                                     ) { cli                                     }
        finished := false;
        while not finished do
            begin
                shot := 1;
                repeat                                     { until shot > 4                                     }
                    set_dac ($0);
                    reset;
                    acquire_waveform (waveform, shot);
                    write ('Do you want to view this waveform? (raw data) Y/N');
                    ans := readkey;
                    if upcase(ans) <> 'N' then
                        for loop := 1 to 4096 do
                            begin
                                write (waveform[shot,loop]:6);
                                if (loop mod 10) = 0 then
                                    writeln;
                            end;
                        end;
                    writeln;
                    write ('Is this waveform recorded correctly? Y/N');
                    ans := readkey;
                    writeln;
                    if upcase(ans) = 'Y' then
                        begin
                            writeln ('Waveform Accepted -- Shot #: ', shot:3);
                            inc(shot);
                        end
                    else

```

```

        writeln ('Waveform Rejected -- Shot #: ', shot:3, ' next. ');
until shot > 4;
write ('Do you want to view this waveform? (graphically) Y/N ');
ans := readkey;
if upcase(ans) <> 'N' then
    show_waveform (waveform);
writeln;
if store then
    begin
        write ('Do you want to store this waveform? Y/N ');
        ans := readkey;
        writeln;
        if upcase(ans) = 'Y' then
            store_on_disk (waveform, job);
        end;
        write ('Do you want to acquire another waveform? Y/N ');
        ans := readkey;
        writeln;
        if upcase(ans) = 'N' then
            finished := true;
        end;
    inline ($fb);
end. { main program }

```

APPENDIX B

POST PROCESSING SOFTWARE LISTING

{ This Listing is Arranged as follows:

1. Procedure List Showing Procedure Nesting.
2. Procedure List Showing Calling Procedures, Procedures Called, and Procedure Function.
3. Program Listing.

1. Procedure Nesting

PROCESS

```

MENU (menu_choices:choice_array; var selection:integer)
PATH_TO_FILE (var filename:name_string)
    ENTER_PATH (var filename:name_string; var in_char:char)
SELECT_LOAD_OPTION(var load_option:char)
LOAD_WAVEFORM (var signal_array:waveform; filename:name_string);
PROCESS_4_SIGNALS (var averaged_array:waveform; filename:name_string)
INSERT_BREAK (s_break:integer; s_positive:boolean; var
                break_list:listpointer)
PICK_BREAKS (var signal_array:waveform; var break_list:listpointer)
NEW_WAVEFORM (var signal_array:waveform; var filename:name_string)
DISPLAY_BREAKS (break_list:listpointer; var signal_array:waveform)
DISPLAY_WAVEFORM (var signal_array:waveform; filename:name_string)
STORE_BREAK_LIST (filename:name_string; break_list:listpointer)
DISPLAY_EXPANDED_WAVEFORM (start, stop:integer; break_list:listpointer;
                            mode:option; var signal_array:waveform)
EDIT_BREAKS (var finished:boolean; filename:name_string; var
                break_list:listpointer; var signal_array:waveform; start,x:integer)
    DISPLAY_START_END (var displaystart, displayend, start,stop:integer)
    BREAK_SELECT (start, stop, move:integer; var
                    signal_array:waveform; var break_list:listpointer)
    DELETE_BREAK (start, stop, move:integer; var
                    success:boolean; var break_list:listpointer)
    ADD_BREAK (start, stop, move:integer; var break_list:listpointer)
    MOVE_BREAK (var start, stop, move:integer; var
                    break_list:listpointer; var signal_array:waveform)
EXPAND_WAVEFORM (var signal_array:waveform; var break_list:listpointer;
                var finished:boolean; filename:name_string)
EDIT_WAVEFORM_DATA (var temp_array:waveform)
FILTERING (var temp_array:waveform)
SMOOTHING (var temp_array:waveform)
EDIT_WAVEFORM (var signal_array:waveform;filename:name_string)
DISPLAY_EXPAND (var signal_array:waveform;filename:name_string;
                var break_list:listpointer;breaks:boolean)
DISPLAY_INTERPRETATION (var signal_array:waveform;
                var filename:name_string;var breaks_list:listpointer)
EDIT_FROM_DISK_DISPLAY_INTERPRETATION (var signal_array:waveform;
                var filename,b_filename:name_string; var break_list:listpointer;
                var break_list:listpointer)

```

2. Procedure Function Listing

```

program PROCESS;
uses graph,dos,crt,drivers;

const
  array_length = 4096;

type
  choice_array = array[1..5] of string[40];
  waveform = array[1..array_length] of integer;
  name_string = string[40];
  option = string[10];
  break_rec = record
    break      : integer;
    positive   : boolean;
  end;
  listpointer = ^listnode;
  listnode = record
    break      : integer;
    positive   : boolean;
    next       : listpointer;
  end;

const
  opening_menu: choice_array = (
    'A.  PROCESS NEW WAVEFORM',
    'B.  RECALL/EDIT WAVEFORM',
    'C.  DISPLAY INTERPRETATION',
    'D.  HELP ',
    'E.  EXIT');

var
  signal_array      : waveform;
  break_record      : break_rec;
  break_file        : file of break_rec;  {global to whole program}

  path_filename1,
  path_filename     : name_string;
  break_list        : listpointer;
  graphdriver,
  selection,
  graphmode         : integer;
  breaks            : boolean;

  {*****}
  {*****}

procedure MENU (menu_choices:choice_array; var selection:integer);

var
  YY,
  loop              : integer;
  in_char           : char;
  p                 : pointer;
  size              : word;

Calls:          none
Called by:     Main Program

```

Function: Returns selection from menu display to the calling program. An integer (1-5) is returned in the variable "selection". Menu displayed is passed

to MENU procedure in "menu_choices": an array which can hold up to 5 character strings. Each string must be prefixed with a letter (ie. A - E).

```
{*****}
{*****}
```

procedure PATH_TO_FILE (var filename:name_string);

```
var
  x,                                { global inside path_to_file}
  loop          : integer;
  size          : word;
  q             : pointer;        { global inside path_to_file}
  in_char       : char;
  dirinfo       : searchrec;
```

Calls: ENTER_PATH (internal)
Called by: NEW_WAVEFORM
 EDIT_FROM_DISK_DISPLAY_INTERPRETATION

Function: Facilitates entry of path and filename of file to retrieve. Procedure checks that path and filename are valid but does not check that file is a valid file type. Returns path and filename to calling program in "filename" variable. Also facilitates display of specified directories.

```
{*****}
```

procedure ENTER_PATH (var filename:name_string; var in_char:char);

```
var
  loop          : integer;
  temp          : name_string;
```

Calls: none
Called by: PATH_TO_FILE (internal)

Function: Actual keyboard entry of filename or directory path. Returns path in "filename" variable and last keyboard entry in "in_char" variable.

```
{*****}
{*****}
```

procedure SELECT_LOAD_OPTION (var load_option:char);

Calls: none
Called by: NEW_WAVEFORM
 EDIT_FROM_DISK_DISPLAY_INTERPRETATION

Function: Facilitates user selection to read single data set or from concatenated data set to then perform averaging.

```
{*****}
{*****}
```

procedure LOAD_WAVEFORM (var signal_array:waveform; filename:name_string);

```
var
  value,
  loop          : integer;
  disk_file     : text;
```

Calls: none
Called by: NEW_WAVEFORM

EDIT_FROM_DISK_DISPLAY_INTERPRETATION

Function: Reads a single data set into the computer from a disk file. Reads 4096 data points - does no averaging.

```
{*****}
{*****}
```

```
procedure PROCESS_4_SIGNALS (var averaged_array:waveform; filename:name_string);
```

```
var
  value,
  loop,
  loop1      : integer;
  disk_file  : text;
```

Calls: none

Called by: NEW_WAVEFORM
EDIT_FROM_DISK_DISPLAY_INTERPRETATION

Function: Reads data from disk file specified in "filename" variable sent from calling program. Expects four concatenated sets of data. Reads all four, adds respective data points and divides by 4 to compute average. Resulting average is returned to calling program in "averaged_array" variable.

```
{*****}
{*****}
```

```
procedure INSERT_BREAK (s_break:integer; s_positive:boolean; var
                        break_list:listpointer);
```

```
var
  temp,
  traverse      : listpointer;
```

Calls: none

Called by: PICK_BREAKS
EDIT_FROM_DISK_DISPLAY_INTERPRETATION

Function: Adds break points sent to procedure in "s_break" and "s_positive" variables to end of current break list. "break_list" variable is returned to calling program.

```
{*****}
{*****}
```

```
procedure PICK_BREAKS (var signal_array:waveform; var break_list:listpointer);
```

```
const
  ave_length = 20;
  overlap    = 0;
  diff       = 5;
```

```
var
  av1,
  av2,
  av3,
  inflection1,
  inflection2,
  midpoint,
  start      : integer;
  peak,
  valley     : boolean;
```

```
function AVERAGE (var signal_array:waveform; start:integer):integer;
```

Calls: INSERT_BREAK
Called by: DISPLAY_EXPAND

Function: Picks breaks from waveform based on the following method: three averages are computed on three partially overlapping sections of the waveform. When a peak or valley is detected (by comparing the three averages) the break is chosen as the midpoint between the last valley (or peak) and the present peak (or valley). The variable "signal_array" and "break_list" are returned to the calling program.

```
{*****}
{*****}
```

```
procedure NEW_WAVEFORM (var signal_array:waveform; var filename:name_string);
```

```
var
  old_fillpattern : word;
  option          : char;
```

Calls: PATH_TO_FILE
PROCESS_4 SIGNALS
SELECT_LOAD_OPTION
LOAD_WAVEFROM
Called by: Main Program

Function: Inputs raw waveform data. Returns "signal_array" and "filename" variables to calling program.

```
{*****}
```

```
procedure DISPLAY_BREAKS (break_list:listpointer; var signal_array:waveform);
```

```
const
  ydiv = 28;
  yadd = -20;
```

```
var
  loop,
  x      : integer;
  temp   : listpointer;
  t      : pointer;
  size   : word;
```

Calls: none
Called by: EXPAND_WAVEFORM

Function: Displays breaks passed to it in "break_list" variable on waveform displayed by DISPLAY_WAVEFORM procedure. "signal_array" is passed back to calling program.

```
{*****}
{*****}
```

```
procedure DISPLAY_WAVEFORM (var signal_array:waveform; filename:name_string);
```

```
const
  ydiv = 28;
  yadd = -20;
```

```
var
```

```

last,
current,
loop1,
space,
x,
loop      : integer;
dummy    : char;

```

Calls: none
Called by: EDIT_WAVEFORM
EXPAND_WAVEFORM
DISPLAY_EXPAND

Function: Displays entire waveform across top of screen. "signal-array" variable is passed back to calling program.

```

{*****}
{*****}

```

```

procedure STORE_BREAK_LIST (filename:name_string; break_list:listpointer);

```

```

var
loop,
x      : integer;
in_char : char;
p      : pointer;
size   : word;
temp_string,
break_filename : name_string;
temp_pointer  : listpointer;

```

Calls: none
Called by: EDIT_BREAKS
EXPAND_WAVEFORM

Function: Stores break list (as passed to it in "break_list" variable) on to file on disk. This is not an ASCII file! "filename" variable is passed to procedure and holds name of currently shown waveform file.

```

{*****}
{*****}

```

```

procedure DISPLAY_EXPANDED_WAVEFORM (start, stop:integer; break_list:listpointer;
mode:option; var signal_array:waveform);

```

```

const
yadd = -40;
ydiv = 14;

```

```

var
temp      : listpointer;
t         : pointer;
size      : word;
loop,
xex,
length,
displaystart,
displayend : integer;

```

Calls: none
Called by: EXPAND_WAVEFORM
BREAK_SELECT

Function: Displays waveform from "start" to "stop" across the lower half of the screen. "start" and "stop" are selected and passed by calling program. Breaks are displayed. "mode" specifies 'view' or 'edit'. "signal_array" is passed back to calling program.

```
{*****}
{*****}
```

```
procedure EDIT_BREAKS (var finished:boolean; filename:name_string;
    var break_list:listpointer; var signal_array:waveform;
    start, x:integer);
```

```
const
    yadd      = -40;          {global to Edit Breaks      }
    ydiv      = 14;          {global to Edit Breaks      }
```

```
var
    in_char   : char;
    move      : integer;
    success   : boolean;
```

```
function ARRAY_POSITION (displayend, displaystart,
    point:integer):integer;
```

Calls: BREAK_SELECT (internal)
 DELETE_BREAK (internal)
 ADD_BREAK (internal)
 MOVE_BREAK (internal)
 STORE_BREAK_LIST

Called by: EXPAND_WAVEFORM

Function: Facilitates edit of breaks on waveform. Moving, deleting, and adding breaks is facilitated. An updated break list can be stored on disk. "signal_array" is passed back to calling program. "finished" and "mode" are also returned to the calling program.

```
{*****}
```

```
procedure DISPLAY_START_END (var displaystart, displayend, start, stop:integer);
```

```
var
    length    : integer;
```

Calls: none
Called by: BREAK_SELECT
 DELETE_BREAK
 ADD_BREAK

Function: Determines actual array values that begin and end displayed waveform from DISPLAY_EXPAND_WAVEFORM procedure. These values are returned in "displaystart" and "displayend" variables respectively. "start" and "stop" are also returned to calling program.

```
{*****}
```

```
procedure BREAK_SELECT (start, stop, move:integer; var
    signal_array:waveform; var break_list:listpointer);
```

```
var
    current_position,
    displaystart,
    displayend : integer;
```

Calls: DISPLAY_START_END
 DISPLAY_EXPANDED_WAVEFORM
Called by: MOVE_BREAK
 EDIT_BREAKS (internal)

Function: Displays break select box on waveform. Passes "signal_array" variable back to calling program.

{*****}

procedure DELETE_BREAK (start, stop, move:integer; var
 success:boolean; var break_list:listpointer);

var
 delete_position,
 tolerance,
 displaystart,
 displayend : integer;
 temp1,
 temp : listpointer;

Calls: DISPLAY_START_END
Called by: MOVE_BREAK
 EDIT_BREAKS (internal)

Function: Deletes selected break from break list. Updates "break_list" and passes variable back to calling program.

{*****}

procedure ADD_BREAK (start, stop, move:integer; var
 break_list:listpointer);

var
 add_position,
 displaystart,
 displayend : integer;
 temp1,
 temp : listpointer;

Calls: DISPLAY_START_END
Called by: MOVE_BREAK
 EDIT_BREAKS (internal)

Function: Adds break to break list. Updates "break_list" and passes variable back to calling program.

{*****}

procedure MOVE_BREAK (var start, stop, move:integer; var
 break_list:listpointer; var signal_array:waveform);

var
 move1 : integer;
 in_char : char;
 success : boolean;

Calls: BREAK_SELECT
 DELETE_BREAK
 ADD_BREAK
Called by: EDIT_BREAKS (internal)

Function: Moves selected break the specified amount. Updates "break_list"

variable and passes it back to the calling program. Also passes "start", "stop", "move", "last_move", and "signal_array" variables back to calling program.

$$\left\{ \begin{array}{l} \text{*****} \\ \text{*****} \end{array} \right.$$

```
procedure EXPAND_WAVEFORM (var signal_array:waveform; var break_list:listpointer;  
                           var finished:boolean; filename:name_string);
```

```
const
    yadd      = -40;
    ydiv      = 14;
```

```
var
    in_char      : char;
    q            : pointer;
    size         : word;
    start,
    x            : integer;
```

Calls: DISPLAY_EXPANDED_WAVEFORM
 STORE_BREAK_LIST
 EDIT_BREAKS
 DISPLAY_BREAKS
 DISPLAY_WAVEFORM

Called by: DISPLAY EXPAND

Function: Displays instructions and facilitates selection of segment of waveform to display in expanded form. Returns "signal_array" and "finished" variables to calling program.

$$\left\{ \begin{array}{l} \text{*****} \\ \text{*****} \end{array} \right.$$

```
procedure EDIT_WAVEFORM_DATA (var temp_array:waveform);
```

```
var
    size    word;
    q       : pointer;
    x,
    loop    : integer;
    in char : char;
```

Calls: none

Called by: EDIT_WAVEFORM

Function: Allows user to select start of good data on waveform. Waveform is shifted to start of good data, end is filled with last value. Returns updated "temporary" waveform file.

{*****}

```
procedure FILTERING (var temp_array:waveform);
```

```

type
  second_order_elliptic = array[1..11,1..3] of real;
  real_waveform = array[1..array length] of real;

```

[illegible]

```

(1,-1.2199,0.6341),      {0.26}
(1,-1.1423,0.6155),      {0.28}
(1,-1.0628,0.5983),      {0.30}
(1,-0.9818,0.5824),      {0.32}
(1,-0.8991,0.5678),      {0.34}
(1,-0.8149,0.5545),      {0.36}
(1,-0.7293,0.5426),      {0.38}
(1,-0.6424,0.5320));    {0.40}

```

```

coeffic_b : second_order_elliptic = ((0.1750,-0.0932,0.1750), {0.20}
(0.1840,-0.0621,0.1840), {0.22}
(0.1937,-0.0291,0.1937), {0.24}
(0.2043,0.0056,0.2043), {0.26}
(0.2157,0.0419,0.2157), {0.28}
(0.2278,0.0798,0.2278), {0.30}
(0.2407,0.1192,0.2407), {0.32}
(0.2543,0.1601,0.2543), {0.34}
(0.2686,0.2025,0.2686), {0.36}
(0.2835,0.2462,0.2835), {0.38}
(0.2992,0.2912,0.2992)); {0.40}

```

```

var
  temp      : real_waveform;
  Wc,
  loop      : integer;
  in_char   : char;

```

Calls: none
Called by: EDIT_WAVEFORM

Function: Performs second order elliptic filtering on waveform in time domain. Allows user to specify cutoff frequency between 0.2 and 0.4 the sampling frequency. Returns updated "temporary" waveform file.

```

{*****}
{*****}

```

```

procedure SMOOTHING (var temp_array:waveform);

```

```

var
  max_difference,
  above_threshold,
  below_threshold,
  loop : integer;
  finished : boolean;
  count : real;

```

Calls: none
Called by: EDIT_WAVEFORM

Function: Finds top 10% magnitude differences from point to point. Performs smoothing across these points. Returns updated "temporary" waveform file.

```

{*****}
{*****}

```

```

procedure EDIT_WAVEFORM (var signal_array:waveform; filename_name_string);

```

```

var
  temp_array : waveform;
  loop : integer;
  in_char : char;
  temp_string : name_string;

```

```
disk_file    : text;
```

```
Calls:      EDIT_WAVEFORM_DATA
            FILTERING
            SMOOTHING
            DISPLAY_WAVEFORM
```

```
Called by:  DISPLAY_EXPAND
```

Function: Gives options for editing/processing actual waveform data. Returns "signal_array" to calling program.

```
{*****}
{*****}
```

```
procedure DISPLAY_EXPAND (var signal_array:waveform;
                          filename:name_string; var break_list:listpointer;breaks:boolean);
```

```
var
  finished    : boolean;
  in_char     : char;
```

```
Calls:      DISPLAY_WAVEFORM
            EXPAND_WAVEFORM
            PICK_BREAKS
```

```
Called by:  EDIT_WAVEFORM
            EDIT_FROM_DISK
            Main Program
```

Function: Calls DISPLAY_WAVEFORM procedure to display entire waveform and EXPAND_WAVEFORM procedure to facilitate expanding and displaying the selected section of the waveform. Calls EDIT_WAVEFORM procedure to facilitate waveform edit procedures. Returns "signal_array" and "break_list" variables to the calling program.

```
{*****}
{*****}
```

```
procedure DISPLAY_INTERPRETATION (var signal_array:waveform;
                                   var filename:name_string; var break_list:listpointer);
```

```
var
  temp        : listpointer;
  x1,
  x2          : integer;
  in_char     : char;
```

```
Calls:      DISPLAY_WAVEFORM
            DISPLAY_BREAKS
```

```
Called by:  EDIT_FROM_DISK_DISPLAY_INTERPRETATION
```

Function: Displays computer generated interpretation of waveform from specified waveform and pre-generated break list.

```
{*****}
{*****}
```

```
procedure EDIT_FROM_DISK_DISPLAY_INTERPRETATION (var signal_array:waveform;
                                                  var filename,b_filename:name_string; var break_list:listpointer;edit:boolean);
```

```
var
  option      : char;
```

```
Calls:      PATH_TO_FILE
```

```

PROCESS_4_SIGNALS
INSERT_BREAK
DISPLAY_EXPAND
LOAD_WAVEFORM
DISPLAY_INTERPRETATION

```

Called by: Main Program

Function: Recalls waveform and previously stored break list from disk file. Returns "signal_array", "filename", "b_filename", and "break_list" variables back to calling program if called from Main Program Option B. If Main Program OptionC was selected, the computer interpretaion is generated.

```

{*****}
{*****}

```

MAIN PROGRAM

Calls:

```

MENU
NEW_WAVEFORM
DISPLAY_EXPAND
EDIT_FROM_DISK_DISPLAY_INTERPRETATION

```

Function: Operates from base menu:

- 1- New Waveform View and Edit
- 2- Recall Waveform and Break List from Disk for View and Edit
- 3- Recall Waveform and Break List from Disk for Interpretation Display
- 4- Display Help File
- 5- Exit to DOS

3. Program Listing }

```

{$M 40000,0,25000}
program PROCESS;
uses graph,dos,crt,drivers;

const
    array_length = 4096;

type
    choice_array = array[1..5] of string[40];
    waveform = array[1..array_length] of integer;
    name_string = string[40];
    help_string = string[80];
    option = string[10];
    break_rec = record
        break      : integer;
        positive   : boolean;
    end;
    listpointer = ^listnode;
    listnode = record
        break      : integer;
        positive   : boolean;
        next       : listpointer;
    end;

const
    opening_menu: choice_array = (
        'A.  PROCESS NEW WAVEFORM',

```

```

        'B. RECALL/EDIT WAVEFORM',
        'C. DISPLAY INTERPRETATION',
        'D. HELP ',
        'E. EXIT');

var
    signal_array      : waveform;
    break_record      : break_rec;
    break_file        : file of break_rec;      (global to whole program )
    path_filename1,
    path_filename     : name_string;
    break_list        : listpointer;
    graphdriver,
    selection,
    graphmode         : integer;
    breaks            : boolean;

    {*****}
    {*****}

procedure MENU (menu_choices:choice_array; var selection:integer);

var
    yy,
    loop              : integer;
    in_char           : char;
    p                 : pointer;
    size              : word;

begin
    size := imagesize(1,1,16,9);
    getmem(p,size);
    setcolor(green);
    rectangle(20,20,getmaxx - 20,getmaxy - 20);
    setfillstyle(solidfill, green);
    floodfill(0,0,green);
    setfillstyle(solidfill, white);
    setcolor(white);
    for loop:=1 to 5 do
        outtextxy(70,40+(loop-1)*10,menu_choices[loop]);
    outtextxy(70,120,'Enter Choice or Use Arrow keys and Hit <RETURN>');
    yy:=39;
    getimage(1,1,16,9,p^);
    putimage(68,yy,p^,xorput);
    selection:=0;
    repeat
        in_char := upcase(readkey);
        case ord(in_char) of
            80 : begin
                    putimage(68,yy,p^,xorput);
                    if yy=79 then
                        yy:=39
                    else
                        yy:=yy+10;
                    putimage(68,yy,p^,xorput);
                end;
            72 : begin
                    putimage(68,yy,p^,xorput);
                    if yy=39 then
                        yy:=79
                    else
                        yy:=yy-10;
                    putimage(68,yy,p^,xorput);
                end;
        end;
    until selection <> 0;
end;

```

```

        end;
        65,66,67,68,69 : selection := ord(in_char) - 64;
        13 : selection := (yy div 10) - 2;
    end;
    until selection <> 0;
    freemem(p, size);
end;

{*****}
{*****}

procedure PATH_TO_FILE (var filename:name_string);

var
    x,                                     { global inside path_to_file}
    loop                                : integer;
    size                                : word;
    q                                    : pointer;                               { global inside path_to_file}
    in_char                             : char;
    dirinfo                             : searchrec;

    {*****}

procedure ENTER_PATH (var filename:name_string; var in_char:char);

var
    loop                                : integer;
    temp                                : name_string;

begin
    x := 150;
    for loop := 1 to 40 do
        putimage(150 + (loop-1)*9,60,q^,copyput);
    in_char := readkey;
    filename := '';
    for loop := 1 to 73 do
        putimage(9 + (loop-1)*8,85,q^,copyput);
    while (ord(in_char) <> 13) and (ord(in_char) <> 27)
        and (ord(in_char) <> 59) do
        begin
            if (x < 509) and (ord(in_char) <> 8) then
                begin
                    filename := filename + in_char;
                    outtextxy(x,60,in_char);
                    if x < 510 then
                        x := x + 9;
                    end
                end
            else if ord(in_char) = 8 then
                begin
                    if x > 150 then
                        x := x - 9;
                    putimage(x,60,q^,copyput);
                    temp := '';
                    for loop := 1 to ((x-149)div 9) do
                        temp := temp + filename[loop];
                    filename := temp;
                end;
            in_char := readkey;
        end;
    end;

    {*****}

```

```

begin
  size := imagesize(1,1,8,8);
  getmem(q,size);
  setviewport(21,21,getmaxx - 21, getmaxy - 21,clapon);
  clearviewport;
  getimage(21,21,28,28,q^);
  rectangle(148,56,509,69);
  repeat
    setviewport(21,61,getmaxx - 21,68,clapon);
    clearviewport;
    setviewport(21,120,getmaxx - 21,130,clapon);
    clearviewport;
    setviewport(21,21,getmaxx - 21, getmaxy - 21,clapon);
    outtextxy(60,40,'Enter Path and Filename for File to Retrieve:');
    outtextxy(60,101,'<RETURN> to Continue    <F1> for Directory    <ESC> to
                                                                Exit');

    enter_path (filename, in_char);
    if ord(in_char) = 13 then
      begin
        findfirst(filename,anyfile,dirinfo);
        case doserror of
          0 : clrscr;
          3 : outtextxy(10,85,'Path Not Found (Dos Error = 3): ' + filename);
          18 : outtextxy(10,85,'No More Files (Dos Error = 18): ' + filename);
        end;
      end
    else if ord(in_char) = 59 then
      begin
        setviewport(21,61,getmaxx - 21,68,clapon);
        clearviewport;
        setviewport(21,120,getmaxx - 21,130,clapon);
        clearviewport;
        setviewport(21,21,getmaxx - 21, getmaxy - 21,clapon);
        outtextxy(60,40,'Enter Directory to Show:');
        outtextxy(100,101,'<RETURN> to Continue    <ESC> to Exit');
        enter_path (filename, in_char);
        if ord(in_char) <> 27 then
          begin
            setviewport(21,131,getmaxx - 21, getmaxy - 21, clapon);
            clearviewport;
            setviewport(21,21,getmaxx - 21, getmaxy - 21,clapon);
            setcolor(red);
            line(0,111,getmaxx,111);
            line(0,125,getmaxx,125);
            line(99,125,99,getmaxy);
            line(198,125,198,getmaxy);
            line(297,125,297,getmaxy);
            line(396,125,396,getmaxy);
            line(495,125,495,getmaxy);
            setcolor(yellow);
            outtextxy(90,115,'Directory Shown is: ' + filename);
            findfirst(filename, archive, dirinfo);
            loop := 128;
            x := 0;
            while doserror = 0 do
              begin
                outtextxy(2 + x,loop,dirinfo.name);
                findnext (dirinfo);
                loop := loop + 9;
                if loop = 308 then
                  begin
                    loop := 128;
                    x := x + 99;

```

```

        end;
    end;
    setcolor(white);
end
else
    filename := '';
end
else
    filename := '';
until (doserror = 0) or (ord(in_char) = 27);
freemem(q, size);
end;

```

```

{*****}
{*****}

```

```

procedure SELECT_LOAD_OPTION (var load_option:char);

```

```

begin
    setviewport(21,21,getmaxx - 21, getmaxy - 21, clipon);
    clearviewport;
    outtextxy(10,10,'You Have Two Choices:');
    outtextxy(10,28,'- You can load a file holding raw waveform data. The file');
    outtextxy(10,37,' must hold four sets of concatenated data, 4096 points each.'
    );
    outtextxy(10,46,' The four sets are averaged. ');
    outtextxy(30,64,' PRESS <A>');
    outtextxy(10,82,'- You can load a file holding a single waveform data set of');
    outtextxy(10,91,' 4096 points. NO modification is done to the data set:');
    outtextxy(10,100,' It is displayed as it is recorded. ');
    outtextxy(30,118,' PRESS <B>');
    outtextxy(20,145,' PRESS <C> to EXIT');
    load_option := readkey;
    while not (ord(load_option) in [65,97,66,98,67,99]) do
        load_option := readkey;
    end;
end;

```

```

{*****}
{*****}

```

```

procedure LOAD_WAVEFORM (var signal_array:waveform;filename:name_string);

```

```

var
    value,
    loop      : integer;
    disk_file : text;
begin
    assign(disk_file,filename);
    reset(disk_file);
    outtextxy(200,100,'Retrieving File');
    for loop := 1 to array_length do
        begin;
            read(disk_file,value);
            signal_array[loop] := 4095 - value;
        end;
    close(disk_file);
end;

```

```

{*****}
{*****}

```

```

procedure PROCESS_4_SIGNALS (var averaged_array:waveform;filename:name_string);

```



```

var
  value,
  loop,
  loop1      : integer;
  disk_file  : text;

begin
  assign(disk_file,filename);
  reset(disk_file);
  outtextxy(200,100,'Retrieving File');
  for loop := 1 to array_length do
    read(disk_file,averaged_array[loop]);
    for loop1 := 1 to 3 do
      for loop := 1 to array_length do
        begin
          read(disk_file,value);
          averaged_array[loop] := averaged_array[loop] + value;
        end;
      close(disk_file);
      outtextxy(210,110,'Averaging Waveforms');
      for loop := 1 to array_length do
        averaged_array[loop] := 4095 - (averaged_array[loop] div 4);
      end;

      {*****}
      {*****}

procedure INSERT_BREAK (s_break:integer; s_positive:boolean;
                        var break_list:listpointer);

var
  temp,
  traverse      : listpointer;

begin
  new(temp);
  temp^.break := s_break;
  temp^.next := nil;
  temp^.positive := s_positive;
  if break_list = nil then
    break_list := temp
  else
    begin
      traverse := break_list;
      while traverse^.next <> nil do
        traverse := traverse^.next;
      traverse^.next := temp;
    end;
  end;

end;

      {*****}
      {*****}

procedure PICK_BREAKS (var signal_array:waveform;var break_list:listpointer);

const
  ave_length = 20;
  overlap    = 0;
  diff       = 5;

var
  av1,

```

```

av2,
av3,
inflection1,
inflection2,
midpoint,
start      : integer;
peak,
valley     : boolean;

{*****}

function AVERAGE (var signal_array:waveform; start:integer):integer;

var
    sum,
    loop      : integer;

begin
    sum := 0;
    for loop := start to (start + ave_length) do
        sum := sum + signal_array[loop];
    average := sum div ave_length;
end;

{*****}

begin
    start := 1;
    av1 := average(signal_array, start);
    start := start + 1 + ave_length - overlap;
    av2 := average(signal_array, start);
    start := start + 1 + ave_length - overlap;
    av3 := average(signal_array, start);
    peak := false;
    valley := false;
    repeat
        (until peak or valley or at end)
        if ((av1<av2) and (av2>av3)) or ((av1<av2) and (abs(av2-av3) < diff)) or
            ((abs(av1-av2) < diff) and (av2>av3)) then
            peak := true
        else if ((av1>av2) and (av2<av3)) then
            valley := true
        else
            begin
                av1 := av2;
                av2 := av3;
                start := start + 1 + ave_length - overlap;
                av3 := average(signal_array, start);
            end;
        until peak or valley or ((start + 1 + ave_length - overlap)>array_length);
        inflection1 := start - (ave_length div 2) + overlap;
        av1 := av2;
        av2 := av3;
        start := start + 1 + ave_length - overlap;
        av3 := average(signal_array, start);
        repeat
            (until end)
            repeat
                (until peak or valley or end)
                if ((av1<av2) and (av2>av3)) or ((av1<av2) and (abs(av2-av3) < diff)) or
                    ((abs(av1-av2) < diff) and (av2>av3)) then
                    begin
                        peak := true;
                        valley := false;
                    end
                else if ((av1>av2) and (av2<av3)) then

```

```

begin
    valley := true;
    peak := false;
end
else
begin
    av1 := av2;
    av2 := av3;
    start := start + 1 + ave_length;
    av3 := average(signal_array, start);
end;
until peak or valley or ((start + 1 + ave_length - overlap) > array_length);
inflection2 := start - (ave_length div 2) + overlap;
midpoint := inflection2 - (inflection2 - inflection1) div 2;
insert_break (midpoint, peak, break_list);
inflection1 := inflection2;
peak := false;
valley := false;
av1 := av2;
av2 := av3;
start := start + 1 + ave_length;
av3 := average(signal_array, start);
until (start + 1 + ave_length - overlap) > array_length;
end;

```

```

{*****
*****
*****
***** NEW WAVEFORM PROCEDURE *****
*****
*****
*****}

```

```

procedure NEW_WAVEFORM (var signal_array:waveform; var filename:name_string);

```

```

var
    old_fillpattern : word;
    option          : char;
begin
    break_list := nil;
    setfillstyle(solidfill,black);
    bar(150,5,350,16);
    outtextxy(160,7,'Retrieve Waveform File');
    select_load_option(option);
    if not (ord(option) in [67,99]) then
        begin
            path_to_file(path_filename);
            if path_filename <> '' then
                if ord(option) in [65,97] then
                    process_4_signals(signal_array,path_filename)
                else
                    load_waveform(signal_array, path_filename);
            end
        else
            path_filename := '';
    end;
end;

```

```

{*****
*****
*****
***** DISPLAY BREAKS *****
*****
*****}

```

```

*****
*****}

procedure DISPLAY_BREAKS (break_list:listpointer; var signal_array:waveform);

const
  ydiv      = 28;
  yadd      = -20;

var
  loop,
  x          : integer;
  temp       : listpointer;
  t          : pointer;
  size       : word;

begin
  for loop := 1 to 10 do
    putpixel(1,loop,lightmagenta);
    size := imagesize(1,1,1,10);
    getmem(t,size);
    getimage(1,1,1,10,t^);
    putimage(1,1,t^,xorput);
    temp := break_list;
    while temp^.next <> nil do
      begin
        x := ((temp^.break div 7) + 1) + 19 + (((temp^.break mod 7) - 1) * 2)
                                     div 7;
        putimage(x,signal_array[temp^.break]div ydiv + round(1.25 * yadd),t^,orput);
        temp := temp^.next;
      end;
      freemem(t, size);
    end;
  end;

  { *****
  *****
  *****
  ***** DISPLAY WAVEFORM *****
  *****
  *****
  ***** }

procedure DISPLAY_WAVEFORM (var signal_array:waveform; filename:name_string);

const
  ydiv      = 28;
  yadd      = -20;

var
  last,
  current,
  loop1,
  space,
  x,
  loop      : integer;
  dummy     : char;

begin
  setviewport (0,0,getmaxx,getmaxy,true);
  clearviewport;
  settxtstyle(smallfont,horizdir,4);
  outtextxy(10,getmaxy - 10,filename);
  setviewport(0,0,639,110,clipoff);

```

```

x := 20;
last := (signal_array[1] div ydiv) + yadd;
putpixel(x, last, white);
for loop := 8 to array_length do
  if ((loop-1) mod 7) = 0 then
    begin
      x := x + 1;
      current := (signal_array[loop] div ydiv) + yadd;
      putpixel(x, current, white);
      space := abs(last - current);
      if (space > 1) and (current > last) then
        for loop1 := 1 to (space div 2) do
          begin
            putpixel(x, current - loop1, white);
            putpixel(x - 1, last + loop1, white);
          end
        else if (space > 1) and (current < last) then
          for loop1 := 1 to (space div 2) do
            begin
              putpixel(x, current + loop1, white);
              putpixel(x - 1, last - loop1, white);
            end;
          last := current;
        end;
      settxtstyle(defaultfont,horizdir,1);
      setviewport (0,0,getmaxx,getmaxy,true);
    end;

```

```

{*****
*****
***** STORE BREAK LIST *****
*****
*****}

```

```

procedure STORE_BREAK_LIST (filename:name_string; break_list:listpointer);

```

```

var
  loop,
  x          : integer;
  in_char    : char;
  p          : pointer;
  size       : word;
  temp_string,
  break_filename : name_string;
  temp_pointer : listpointer;

begin
  size := imagesize(1,1,8,8);
  getmem(p,size);
  getimage(1,1,8,8,p^);
  setviewport(160,120,480,180,clipon);
  clearviewport;
  setviewport(0,0,getmaxx,getmaxy,clipoff);
  rectangle(160,120,480,180);
  rectangle(165,122,475,178);
  floodfill(161,121,white);
  outtextxy(169,127,'Displayed Waveform Filename:');
  outtextxy(174,136,filename);
  outtextxy(169,148,'Name of File to Store Breaks:');
  x := 174;
  in_char := readkey;

```

```

break_filename := '';
while (ord(in_char) <> 13) and (ord(in_char) <> 27) do
begin
  if (x < 464) and (ord(in_char) <> 8) then
  begin
    break_filename := break_filename + in_char;
    outtextxy(x,157,in_char);
    if x < 465 then
      x := x + 9;
    end
  else if ord(in_char) = 8 then
  begin
    if x > 174 then
      x := x - 9;
    putimage(x,157,p^,andput);
    temp_string := '';
    for loop := 1 to ((x-173)div 9) do
      temp_string := temp_string + break_filename[loop];
    break_filename := temp_string;
    end;
    in_char := readkey;
  end;
  temp_pointer := break_list;
  assign(break_file,break_filename);
  rewrite(break_file);
  break_record.break := temp_pointer^.break;
  break_record.positive := temp_pointer^.positive;
  write(break_file,break_record);
  repeat
    temp_pointer := temp_pointer^.next;
    break_record.break := temp_pointer^.break;
    break_record.positive := temp_pointer^.positive;
    write(break_file,break_record);
  until temp_pointer^.next = nil;
  close(break_file);
  freemem(p, size);
end;

{ *****
*****
*****
***** DISPLAY EXPANDED WAVEFORM *****
*****
*****
*****}

procedure DISPLAY_EXPANDED_WAVEFORM (start, stop:integer; break_list:listpointer;
mode:option; var signal_array:waveform);

const
  yadd      = -40;
  ydiv      = 14;

var
  temp      : listpointer;
  t         : pointer;
  size      : word;
  loop,
  xex,
  length,
  displaystart,
  displayend : integer;

```

```

begin
  setfillstyle(emptyfill,black);
  rectangle(0,110,639,338);
  setviewport(1,111,638,337,clipon);
  clearviewport;
  xex := 0;
  length := 7 * (stop - start);
  if length > getmaxx then
    begin
      displaystart := 7 * (start - 20) - (getmaxx - (length mod getmaxx))
                                                    div 2;

      if displaystart < 0 then
        displaystart := 0;
      displayend := displaystart + ((length div getmaxx) + 1) * getmaxx;
      if displayend > array_length then
        begin
          displaystart := displaystart - (displayend - array_length);
          if displaystart < 0 then
            begin
              displaystart := 0;
            end
          displayend := ((length div getmaxx) + 1) * getmaxx;
        end
      else
        displayend := array_length;
      end;
      for loop := displaystart to displayend do
        if (loop mod ((length div getmaxx) + 1)) = 0 then
          begin
            putpixel(xex,(signal_array[loop] div ydiv) + yadd,white);
            xex := xex + 1;
          end;
        end
      end
    else
      {length <= getmaxx}
      begin
        displaystart := 7 * (start - 20) - (getmaxx - length) div 2;
        if displaystart < 0 then
          displaystart := 0;
        displayend := displaystart + getmaxx;
        if displayend > array_length then
          begin
            displaystart := displaystart - (displayend - array_length);
            displayend := array_length;
          end;
        for loop := displaystart to displayend do
          begin
            putpixel(xex,(signal_array[loop] div ydiv) + yadd,white);
            xex := xex + 1;
          end;
        end;
      end;
      setviewport(0,0,getmaxx,getmaxy,clipoff);
      rectangle(displaystart div 7 + 20,0,displayend div 7 + 20,110);
      setfillstyle(solidfill,red);
      floodfill(displaystart div 7 + 25,2,white);
      floodfill(displaystart div 7 + 25,108,white);
      floodfill(displayend div 7 + 15,2,white);
      floodfill(displayend div 7 + 15,108,white);
      setviewport(1,111,638,339,clipoff);
      rectangle(0,0,45,13);
      setfillstyle(solidfill,blue);
      floodfill(2,2,white);
      outtextxy(7,3,mode);
      for loop := 1 to 20 do
        putpixel(1,loop,cyan);
      end
    end
  end
end

```

```

size := imagesize(1,1,1,20);
getmem(t,size);
getimage(1,1,1,20,t^);
putimage(1,1,t^,xorput);
temp := break_list;
while (temp^.break < displaystart) and (temp^.next <> nil) do
  temp := temp^.next;
if temp^.next <> nil then
  while (temp^.break <= displayend) and (temp^.next <> nil) do
    begin
      xex := (temp^.break - displaystart) div ((displayend-displaystart + 1)
                                                div getmaxx);
      putimage(xex,signal_array[temp^.break]div ydiv + round(1.25 *
                                                                yadd),t^,orput);
      temp := temp^.next;
    end;
  setviewport(0,0,getmaxx,getmaxy,clipoff);
  freemem(t, size);
end;

{*****
*****
***** EDIT BREAKS *****
*****
*****}

procedure EDIT_BREAKS (var finished:boolean; filename:name_string;
  var break_list:listpointer; var signal_array:waveform; start, x:integer);

const
  yadd      = -40;          (global to Edit Breaks   )
  ydiv      = 14;          (global to Edit Breaks   )

var
  in_char   : char;
  move      : integer;
  success   : boolean;

  {*****}

function ARRAY_POSITION (displayend, displaystart, point:integer):integer;

begin
  array_position := displaystart + point * ((displayend -
                                              displaystart + 1)div getmaxx);
end;

{*****}

procedure DISPLAY_START_END (var displaystart, displayend, start, stop:integer);

var
  length    : integer;

begin
  length := 7 * (stop - start);
  if length > getmaxx then
    begin
      displaystart := 7 * (start - 20) - (getmaxx - (length mod getmaxx))
                                                div 2;
      if displaystart < 0 then

```



```

    displaystart := 0;
    displayend := displaystart + ((length div getmaxx) + 1) * getmaxx;
    if displayend > array_length then
        begin
            displaystart := displaystart - (displayend - array_length);
            if displaystart < 0 then
                begin
                    displaystart := 0;
                    displayend := ((length div getmaxx) + 1) * getmaxx;
                end
            else
                displayend := array_length;
            end;
        end
    else
        {length <= getmaxx}
        begin
            displaystart := 7 * (start - 20) - (getmaxx - length) div 2;
            if displaystart < 0 then
                displaystart := 0;
            displayend := displaystart + getmaxx;
            if displayend > array_length then
                begin
                    displaystart := displaystart - (displayend - array_length);
                    displayend := array_length;
                end;
            end;
        end;
end;

{*****}

procedure BREAK_SELECT (start, stop, move:integer; var signal_array:waveform;
                        var break_list:listpointer);

var
    current_position,
    displaystart,
    displayend : integer;

begin
    display_start_end(displaystart, displayend, start, stop);
    current_position := 320 + move;
    if current_position < 4 then
        current_position := 4
    else if current_position > (getmaxx - 5) then
        current_position := getmaxx - 5;
    display_expanded_waveform(start, stop, break_list, 'EDIT', signal_array);
    setviewport(1,111,638,339,clipon);
    setcolor(green);
    rectangle(current_position - 4, signal_array[array_position(displayend,
        displaystart, current_position)]div ydiv + round(1.32 * yadd),
        current_position + 4, signal_array[array_position(displayend,
        displaystart, current_position)]div ydiv + round(1.32 * yadd) + 24);
    setcolor(white);
    setviewport(0,0,getmaxx, getmaxy, clipoff);
end;

{*****}

procedure DELETE_BREAK (start, stop, move:integer; var success:boolean;
                        var break_list:listpointer);

var
    delete_position,

```

```

tolerance,
displaystart,
displayend    : integer;
templ,
temp          : listpointer;

begin
  display_start_end(displaystart, displayend, start, stop);
  tolerance := (displayend-displaystart)div 640;
  delete_position := array_position(displayend, displaystart, 320 + move);
  temp := break_list;
  templ := temp;
  while (temp^.break < (delete_position-tolerance)) and (temp^.next <> nil) do
    begin
      templ := temp;
      temp := temp^.next;
    end;
  if (temp^.next <> nil) and (temp^.break >= (delete_position-tolerance)) and
    (temp^.break <= (delete_position+tolerance)) then
    begin
      if temp = break_list then
        break_list := temp^.next
      else
        templ^.next := temp^.next;
        temp^.next := nil;
        dispose(temp);
        success := true;
      end
    else
      success := false;
    end;
end;

{*****}

procedure ADD_BREAK (start, stop, move:integer; var break_list:listpointer);

var
  add_position,
  displaystart,
  displayend    : integer;
  templ,
  temp          : listpointer;

begin
  display_start_end(displaystart, displayend, start, stop);
  add_position := array_position(displayend, displaystart, 320 + move);
  temp := break_list;
  while (temp^.next <> nil) and (temp^.next^.break < add_position) do
    temp := temp^.next;
  new(templ);
  templ^.break := add_position;
  templ^.positive := true;
  if (temp = break_list) and (temp^.break > add_position) then
    begin
      templ^.next := temp;
      break_list := templ;
    end
  else
    begin
      templ^.next := temp^.next;
      temp^.next := templ;
    end;
end;
end;

```

```

{*****}

procedure MOVE_BREAK (var start, stop, move:integer; var break_list:listpointer;
                      var signal_array:waveform);

var
  move1      : integer;
  in_char    : char;
  success    : boolean;

begin
  setviewport(10,getmaxy - 20,633,getmaxy - 12, clipoff);
  clearviewport;
  setviewport(0,0,getmaxx,getmaxy, clipoff);
  move1 := move;
  repeat
    outtextxy(10,getmaxy - 20, '<- -> CTRL+-> -> X 20 CTRL+<- -');
    outtextxy(317,getmaxy - 20, ' <- X 20 F4 - Return To Previous Menu');
    in_char := readkey;
    case ord(in_char) of
      116 : begin
        delete_break(start, stop, move1, success, break_list);
        move1 := move1 + 20;
      end;
      115 : begin
        delete_break(start, stop, move1, success, break_list);
        move1 := move1 - 20;
      end;
      75 : begin
        delete_break(start, stop, move1, success, break_list);
        move1 := move1 - 1;
      end;
      77 : begin
        delete_break(start, stop, move1, success, break_list);
        move1 := move1 + 1;
      end;
      62 : move := move1;
    end;
    if ord(in_char) in [116,115,75,77] then
      begin
        if success then
          add_break(start, stop, move1, break_list);
          break_select(start, stop, move1, signal_array, break_list);
        end;
        until ord(in_char) = 62;
      end;
  end;

{*****}

begin
  move := 0;
  setviewport(1,111,46,124,clipoff);
  clearviewport;
  rectangle(0,0,45,13);
  floodfill(2,2,white);
  outtextxy(7,3,'EDIT');
  setviewport(0,0,getmaxx, getmaxy, clipoff);
  break_select(start, x, move, signal_array, break_list);
  outtextxy(10,getmaxy - 20,'F1 - Quit F2 - Delete F3 - Add');
  outtextxy(305,getmaxy - 20,'F4 - Move F5 - Update F6 - Prev. Menu');
  in_char := readkey;
  repeat
    (until F1 or F6)

```

```

in_char := readkey;
case ord(in_char) of
  116: move := move + 20;
  115: move := move - 20;
  75: move := move - 1;
  77: move := move + 1;
  59: finished := true;
  60: delete_break(start, x, move, success, break_list);
  61: add_break(start, x, move, break_list);
  62: move_break(start, x, move, break_list, signal_array);
  63: store_break_list(filename, break_list);
  64:;
end;
if ord(in_char) in [116,115,75,77,60,61,62,63] then
begin
  break_select(start, x, move, signal_array, break_list);
  outtextxy(10,getmaxy - 20,'F1 - Quit   F2 - Delete   F3 - Add');
  outtextxy(305,getmaxy - 20,'F4 - Move   F5 - Update   F6 - Prev. Menu');
end;
until ord(in_char) in [59, 64];
end;

{*****
*****
*****
***** EXPAND WAVEFORM *****
*****
*****}

procedure EXPAND_WAVEFORM (var signal_array:waveform; var break_list:listpointer;
                           var finished:boolean; filename:name_string);

const
  yadd      = -40;
  ydiv      = 14;

var
  in_char   : char;
  q         : pointer;
  size      : word;
  start,
  x         : integer;

begin
  line(20,2,20,108);
  size := imagesize(20,2,20,108);
  getmem(q, size);
  getimage(20,2,20,108,q^);
  setviewport(1,111,638,337,clipoff);
  clearviewport;
  setviewport(0,0,getmaxx,getmaxy,clipon);
  repeat
    rectangle(1,111,45,124);
    setfillstyle(solidfill,blue);
    floodfill(2,112,white);
    outtextxy(7,114,'VIEW');
    outtextxy(25,getmaxy - 100,'To Expand the Waveform:');
    outtextxy(25,getmaxy - 91,' 1. Use arrow keys to move the marker right or left
                                to');

    outtextxy(454,getmaxy - 91,' the beginning');
    outtextxy(25,getmaxy - 82,'   of the waveform segment to be expanded, hit');
    outtextxy(401,getmaxy - 82,' <RETURN>');
  until finished;
end;

```

```

outtextxy(25,getmaxy - 73,' 2. Use arrow keys to specify the area to be
                                                    expanded');
outtextxy(449,getmaxy - 73,', hit <RETURN>.');
outtextxy(25,getmaxy - 55,'<CTRL + Arrow key> combination facilitates faster
                                                    marker movement.');
```

```

outtextxy(75,getmaxy - 35,'<ESC> to Exit');
display_breaks(break_list, signal_array);
x := 20;
repeat
    in_char := readkey;
    case ord(in_char) of
        116 : begin
            putimage(x,2,q^,xorput);
            x:= x + 20;
            if x >= (getmaxx - 35) then
                x := getmaxx - 35;
            putimage(x,2,q^,xorput);
        end;
        115 : begin
            putimage(x,2,q^,xorput);
            x:= x - 20;
            if x <= 20 then
                x := 20;
            putimage(x,2,q^,xorput);
        end;
        75 : begin
            putimage(x,2,q^,xorput);
            x:= x - 1;
            if x <= 20 then
                x := 20;
            putimage(x,2,q^,xorput);
        end;
        77 : begin
            putimage(x,2,q^,xorput);
            x:= x + 1;
            if x >= (getmaxx - 35) then
                x := getmaxx - 35;
            putimage(x,2,q^,xorput);
        end;
    end;
until ord(in_char) in [13, 27];
start := x;
if ord(in_char) <> 27 then
    repeat
        in_char := readkey;
        case ord(in_char) of
            116 : begin
                if x <> start then
                    putimage(x,2,q^,xorput);
                    x:= x + 20;
                    if x >= (getmaxx - 35) then
                        x := getmaxx - 35;
                    putimage(x,2,q^,xorput);
                end;
            115 : begin
                if x <> start then
                    putimage(x,2,q^,xorput);
                    x:= x - 20;
                    if x > start then
                        putimage(x,2,q^,xorput);
                    else
                        x := start;
                    end;
            end;
        end;
    until ord(in_char) = return;
end;

```

```

75 : begin
    if x <> start then
        putimage(x,2,q^,xorput);
        x:= x - 1;
        if x > start then
            putimage(x,2,q^,xorput)
        else
            x := start;
        end;
    end;
77 : begin
    if x <> start then
        putimage(x,2,q^,xorput);
        x:= x + 1;
        if x >= (getmaxx - 35) then
            x := getmaxx - 35;
            putimage(x,2,q^,xorput);
        end;
    end;
until ((ord(in_char) = 13) and (x <> start)) or (ord(in_char) = 27);
if ord(in_char) <> 27 then
begin
    display_expanded_waveform(start, x, break_list, 'VIEW', signal_array);
    outtextxy(80,getmaxy - 20,'F1 - Quit   F2 - Edit   F3 - Previous Menu');
    outtextxy(440,getmaxy - 20,'F4 - Store Breaks');
    repeat
        in_char := readkey;
        if ord(in_char) = 62 then
            begin
                store_break_list(filename, break_list);
                display_expanded_waveform(start, x, break_list, 'VIEW',
                    signal_array);
                outtextxy(80,getmaxy - 20,'F1 - Quit   F2 - Edit   F3 - Previous
                    Menu');
                outtextxy(440,getmaxy - 20,'F4 - Store Breaks');
            end;
        until ord(in_char) in [59,60,61];
        case ord(in_char) of
            59 : finished := true;
            60,61 : begin
                if ord(in_char) = 60 then
                    edit_breaks(finished, filename, break_list, signal_array,
                        start, x);
                    clearviewport;
                    display_waveform(signal_array, filename);
                    putimage(20,2,q^,xorput);
                end;
            end;
        end
    else
        finished := true;
    until finished;
    freemem(q, size);
end;

{ *****
*****
***** EDIT WAVEFORM DATA *****
*****
*****}

procedure EDIT_WAVEFORM_DATA (var temp_array:waveform);

```

```

var
  size      : word;
  q         : pointer;
  x,
  loop      : integer;
  in_char   : char;

begin
  setviewport(1,150,638,337,clipoff);
  clearviewport;
  setviewport(0,0,getmaxx,getmaxy,clipon);
  outtextxy(25,getmaxy - 109,'Select Start of Waveform "good" Data:');
  outtextxy(25,getmaxy - 100,' Use arrow keys and <CTRL + arrow keys> to
                                position');
  outtextxy(25,getmaxy - 91,' marker to start of good data on the waveform. ');
  outtextxy(25,getmaxy - 82,' Hit <RETURN> to select point, ');
  outtextxy(25,getmaxy - 73,'      <ESC> to exit without changes. ');
  line(20,2,20,108);
  size := imagesize(20,2,20,108);
  getmem(q, size);
  getimage(20,2,20,108,q^);
  x := 20;
  repeat                                     {until in_char = return or esc}
    in_char := readkey;
    case ord(in_char) of
      116 : begin
        putimage(x,2,q^,xorput);
        x:= x + 20;
        if x >= (getmaxx - 35) then
          x := getmaxx - 35;
          putimage(x,2,q^,xorput);
        end;
      115 : begin
        putimage(x,2,q^,xorput);
        x:= x - 20;
        if x <= 20 then
          x := 20;
          putimage(x,2,q^,xorput);
        end;
      75 : begin
        putimage(x,2,q^,xorput);
        x:= x - 1;
        if x <= 20 then
          x := 20;
          putimage(x,2,q^,xorput);
        end;
      77 : begin
        putimage(x,2,q^,xorput);
        x:= x + 1;
        if x >= (getmaxx - 35) then
          x := getmaxx - 35;
          putimage(x,2,q^,xorput);
        end;
    end;
  until ord(in_char) in [13, 27];
  if ord(in_char) <> 27 then
    begin
      x := 7*(x-20);
      for loop := x to array_length do
        temp_array[loop - x + 1] := temp_array[loop];
      for loop := (array_length - x + 2) to array_length do
        temp_array[loop] := temp_array[array_length];
    end;
  end;

```

```

    end;
    freemem(q, size);
end;

{ *****
*****
*****
***** FILTERING *****
*****
*****
*****}

procedure FILTERING (var temp_array:waveform);

type
    second_order_elliptic = array[1..11,1..3] of real;
    real_waveform = array[1..array_length] of real;

const
    coeffic_a : second_order_elliptic = ((1,-1.4410,0.6978),      {0.20}
                                           (1,-1.3694,0.6753),      {0.22}
                                           (1,-1.2957,0.6540),      {0.24}
                                           (1,-1.2199,0.6341),      {0.26}
                                           (1,-1.1423,0.6155),      {0.28}
                                           (1,-1.0628,0.5983),      {0.30}
                                           (1,-0.9818,0.5824),      {0.32}
                                           (1,-0.8991,0.5678),      {0.34}
                                           (1,-0.8149,0.5545),      {0.36}
                                           (1,-0.7293,0.5426),      {0.38}
                                           (1,-0.6424,0.5320));      {0.40}

    coeffic_b : second_order_elliptic = ((0.1750,-0.0932,0.1750), {0.20}
                                           (0.1840,-0.0621,0.1840), {0.22}
                                           (0.1937,-0.0291,0.1937), {0.24}
                                           (0.2043,0.0056,0.2043), {0.26}
                                           (0.2157,0.0419,0.2157), {0.28}
                                           (0.2278,0.0798,0.2278), {0.30}
                                           (0.2407,0.1192,0.2407), {0.32}
                                           (0.2543,0.1601,0.2543), {0.34}
                                           (0.2686,0.2025,0.2686), {0.36}
                                           (0.2835,0.2462,0.2835), {0.38}
                                           (0.2992,0.2912,0.2992)); {0.40}

var
    temp      : real_waveform;
    Wc,
    loop      : integer;
    in_char   : char;

begin
    setviewport(1,150,638,337,clipoff);
    clearviewport;
    setviewport(0,0,getmaxx,getmaxy,clapon);
    outtextxy(25,getmaxy - 111,'Select Filter Cut-Off Frequency:');
    outtextxy(25,getmaxy - 100,' A.  0.20                G.  0.32');
    outtextxy(25,getmaxy - 91,' B.  0.22                H.  0.34');
    outtextxy(25,getmaxy - 82,' C.  0.24                I.  0.36');
    outtextxy(25,getmaxy - 73,' D.  0.26                J.  0.38');
    outtextxy(25,getmaxy - 64,' E.  0.28                K.  0.40');
    outtextxy(25,getmaxy - 55,' F.  0.30 -RECOMMENDED!');
    outtextxy(25,getmaxy - 44,'      <ESC> to exit with no filtering');
    repeat
        in_char := readkey;

```



```

Wc := 0;
case ord(in_char) of
  65,97 : Wc := 1;
  66,98 : Wc := 2;
  67,99 : Wc := 3;
  68,100: Wc := 4;
  69,101: Wc := 5;
  70,102: Wc := 6;
  71,103: Wc := 7;
  72,104: Wc := 8;
  73,105: Wc := 9;
  74,106: Wc := 10;
  75,107: Wc := 11;
end;
until (Wc <> 0) or (ord(in_char) = 27);
if ord(in_char) <> 27 then
begin
  outtextxy(380,getmaxy-80,'WAIT - FILTERING IN PROGRESS');
  temp[1] := coeffic_b[Wc,1]*temp_array[1];
  temp[2] := coeffic_b[Wc,1]*temp_array[2] + coeffic_b[Wc,2]*temp_array[1] -
    coeffic_a[Wc,2]*temp[1];
  for loop := 3 to array_length do
    temp[loop] := coeffic_b[Wc,1]*temp_array[loop] +
      coeffic_b[Wc,2]*temp_array[loop-1] +
      coeffic_b[Wc,3]*temp_array[loop-2] -
      coeffic_a[Wc,2]*temp[loop-1] -
      coeffic_a[Wc,3]*temp[loop-2];
  for loop := 1 to array_length do
    temp_array[loop] := round(temp[loop]);
  end;
end;

{*****
*****
***** SMOOTHING *****
*****
*****}

procedure SMOOTHING (var temp_array:waveform);

var
  max_difference,
  above_threshold,
  below_threshold,
  loop           : integer;
  finished       : boolean;
  count          : real;

begin
  outtextxy(380,getmaxy-80,'WAIT - SMOOTHING IN PROGRESS');
  max_difference := 0;
  count := 0;
  finished := false;
  for loop := 1 to (array_length - 1) do
    if (abs(temp_array[loop] - temp_array[loop + 1])) > max_difference then
      max_difference := abs(temp_array[loop] - temp_array[loop + 1]);
    repeat
      { until finished }
      above_threshold := 0;
      below_threshold := 0;
      for loop := 1 to (array_length - 1) do
        if (abs(temp_array[loop] - temp_array[loop + 1])) >= (max_difference * count)

```

```

                                then
        above_threshold := above_threshold + 1
    else
        below_threshold := below_threshold + 1;
    if (above_threshold / (above_threshold + below_threshold)) <= 0.1 then
        finished := true
    else
        count := count + 0.001;
    until finished;
    for loop := 1 to (array_length - 2) do
        if (abs(temp_array[loop] - temp_array[loop + 1])) >= (max_difference * count)
                                then
            temp_array[loop + 1] := (temp_array[loop] + temp_array[loop + 2]) div 2;
    end;

{*****
*****
*****
***** EDIT WAVEFORM *****
*****
*****
*****}

procedure EDIT_WAVEFORM (var signal_array:waveform; var filename:name_string);

var
    temp_array : waveform;
    in_char    : char;
    loop       : integer;
    temp_string: name_string;
    disk_file  : text;

begin
    for loop := 1 to array_length do
        temp_array[loop] := signal_array[loop];
    repeat
        setviewport(1,150,638,337,clipoff);
        clearviewport;
        setviewport(0,0,getmaxx,getmaxy,clipon);
        outtextxy(25,getmaxy - 127,'To Edit the Waveform:');
        outtextxy(25,getmaxy - 118,' <A> - Edit Waveform Data');
        outtextxy(25,getmaxy - 109,' <B> - Perform Filtering');
        outtextxy(25,getmaxy - 100,' <C> - Perform Smoothing');
        outtextxy(25,getmaxy - 91,' <D> - Save Waveform and Exit WITH Changes');
        outtextxy(25,getmaxy - 73,' <ESC> - Exit Without Changes');
        in_char := readkey;
        while not (ord(in_char) in [65,66,67,68,97,98,99,100,27]) do
            in_char := readkey;
        case ord(in_char) of
            65,97 : edit_waveform_data(temp_array);
            66,98 : filtering(temp_array);
            67,99 : smoothing(temp_array);
            68,100: begin
                    loop := 1;
                    temp_string := '';
                    while filename[loop] <> '.' do
                        begin
                            temp_string := temp_string + filename[loop];
                            loop := loop + 1;
                        end;
                    temp_string := temp_string + '.';
                    filename := temp_string + 'EDT';
                    assign(disk_file, filename);

```

```

        rewrite(disk_file);
        for loop := 1 to array_length do
            begin
                write(disk_file, 4095 - temp_array[loop], ' ');
                signal_array[loop] := temp_array[loop];
            end;
        end;
    end;
    if not (ord(in_char) in [27,69,101]) then
        display_waveform(temp_array, filename);
    until ord(in_char) in [27,68,100];
end;

{*****
*****
***** DISPLAY_EXPAND *****
*****
*****}

procedure DISPLAY_EXPAND (var signal_array:waveform; filename:name_string;
                        var break_list:listpointer; breaks:boolean);

var
    finished    : boolean;
    in_char     : char;

begin
    finished := false;
    if breaks then
        pick_breaks(signal_array, break_list);
    repeat
        display_waveform(signal_array, filename);
        outtextxy(25,getmaxy - 110,'Select:');
        outtextxy(25,getmaxy - 91,' <A> - To edit waveform by filtering,
                                truncation,');
        outtextxy(431,getmaxy - 91,' or smoothing. ');
        outtextxy(25,getmaxy - 73,' <B> - To proceed with break-list generation,');
        outtextxy(404,getmaxy - 73,'view, store and edit options. ');
        outtextxy(75,getmaxy - 35,'<ESC> to Exit');
        repeat
            in_char := readkey;
            case ord(in_char) of
                27 : finished := true;
                65,97 : edit_waveform(signal_array, filename);
                66,98 : expand_waveform(signal_array, break_list, finished, filename);
            end;
        until ord(in_char) in [27,65,66,97,98];
    until finished;
end;

{*****
*****
***** DISPLAY INTERPRETATION *****
*****
*****}

procedure DISPLAY_INTERPRETATION (var signal_array:waveform;
                                var filename:name_string; var break_list:listpointer);

```

```

var
    temp          : listpointer;
    x1,
    x2            : integer;
    in_char       : char;

begin
    display_waveform(signal_array, filename);
    display_breaks(break_list, signal_array);
    temp := break_list;
    while temp <> nil do
        begin
            x1 := ((temp^.break div 7) + 1) + 19 + (((temp^.break mod 7) - 1) * 2)
                                                    div 7;

            if temp^.next <> nil then
                x2 := ((temp^.next^.break div 7) + 1) + 19 + (((temp^.next^.break mod 7)
                                                                - 1) * 2) div 7

            else
                x2 := getmaxx - 35;
                setfillstyle(solidfill,green);
                bar(x1,getmaxy-200,x2,getmaxy-150);
                temp := temp^.next;
                if temp <> nil then
                    begin
                        x1 := ((temp^.break div 7) + 1) + 19 + (((temp^.break mod 7) - 1)
                                                                * 2) div 7;

                        if temp^.next <> nil then
                            x2 := ((temp^.next^.break div 7) + 1) + 19 + (((temp^.next^.break
                                                                mod 7) - 1) * 2) div 7

                        else
                            x2 := getmaxx - 35;
                            setfillstyle(solidfill,red);
                            bar(x1,getmaxy-200,x2,getmaxy-150);
                            temp := temp^.next;
                        end;
                    end;
                outtextxy(50,getmaxy-30,'Hit any key to continue');
                in_char := readkey;
            end;
        end;

        {*****
        *****
        *****
        ***** EDIT FROM DISK *****
        *****
        *****}

    procedure EDIT_FROM_DISK_DISPLAY_INTERPRETATION (var signal_array:waveform;
        var filename,b_filename:name_string; var break_list:listpointer;
        edit:boolean);

    var
        option      : char;

    begin
        setfillstyle(solidfill,black);
        bar(150,5,350,16);
        outtextxy(160,7,'Retrieve Waveform File');
        select_load_option(option);
        if not (ord(option) in [67,99]) then
            begin
                path_to_file(filename);

```



```

                                path_filename, break_list, true);
3 :                                {DISPLAY INTERPRÉTATION}
    edit_from_disk_display_interpretation(signal_array, path_filename1,
                                path_filename, break_list, false);

4 : begin                                                                {HELP}
    closegraph;
    swapvectors;
    exec('\command.com','/c type HELP.DOC|more');
    swapvectors;
    initgraph(graphdriver,graphmode,'');
    end;
    5 : {ok_to_exit};
end;
setgraphmode(graphmode);
until selection = 5;
clrscr;
closegraph;
end.

```

APPENDIX C

Data Processing and Display Software Operations Manual

This manual describes the options available for this program and the acceptable input for each option.

Five options are available from the base menu.

Select B to: input an edited waveform and
 previously generated interpretation.
 Requires input of waveform and break
 list file names.

Select D to: display this manual.

Select E to: exit to DOS.

Selections C, D, and E have no other options available. Selections A and B are described below. A and B both facilitate inputting a waveform to the system. Selection A will be covered first.

Base Menu Selection A

The type of file format the waveform is stored in must be specified:

Input Method A

- 4096 data points/waveform
- 4 waveforms/station
- 4 waveforms stored on disk as single text file
- each set of 4096 points concatenated to previous set
- data values range from 0 to 4095.

Input Method B

- 4096 data points/waveform
- 1 waveform/station
- 1 waveform stored on disk as single text file
- 1 set of 4096 points
- data values range from 0 to 4095.

The program prompts for the complete path and filename of the waveform to retrieve. To display a directory listing,

select 'F1', then specify the complete path of the directory. A file extension must be specified. For example, to list all files in a directory type '*..*'.

After the file has been loaded successfully, two options are displayed:

- A. Edit Waveform,
- B. Break List Generation.

Selecting A facilitates various waveform edit procedures. These are:

1. waveform edit: specify start of valid data.
2. filtering: apply filtering algorithm.
3. smoothing: apply smoothing algorithm.

Any changes to the waveform can be saved or discarded by selecting the appropriate exit option ('D' or <ESC> respectively). Exiting this menu returns to the previous menu.

Selecting B generates the break list and displays the breaks on the waveform. Using the arrow keys and the CTRL+arrow key combinations, a cursor can be moved to the beginning of any section of the waveform to be expanded. Place the first cursor and hit <Return>. Again, using the arrow keys, place the second cursor at the end of the section to expand. Hit <Return>. The selected area will be

expanded to fill the lower half of the terminal screen.

Two options are available:

1. editing the displayed breaks, and
2. storing the in a disk file for later retrieval.

Storing the break list simply requires entering the filename to store the list under. If 'F2-EDIT' is selected, four new options are displayed. A rectangular box is displayed midscreen on the waveform. The arrow and CTRL+arrow keys control the placement of the box.

To delete a break, it must be centered in the box. Adding a break places a break at the center of the box. Selecting 'F4-MOVE' allows movement of an existing break that is centered in the box. Selecting 'F5" facilitates storage of the displayed break list onto the disk.

Base Menu Selection B

Selection B provides all the same options as selection A except as noted below. In addition to specifying the waveform file to enter, a break list file must be specified also. This break list file should have been generated from the specified waveform BUT does not have to be.

After the waveform and break list have been loaded DO NOT edit the waveform data by specifying a new start point,

filtering, or smoothing. A new break list will NOT be generated. The list loaded from disk will be kept. To edit a waveform, always use Base Menu Selection A to load the waveform. All other options are identical to the Base Menu Selection A.

To display the computer generated interpretation after completing all waveform and break list edit functions, the files must be stored onto the disk. Return to the Base menu and select option C.

VITA

Daran Rehmeier was born in 1960 in Baltimore, Maryland. He graduated in 1982 with a B.S.E.E. from Virginia Polytechnic Institute and State University in Blacksburg, Virginia. He worked for Vector Automation, Inc. in Baltimore as a technical writer and for Schlumberger Offshore Services as a Senior Field Engineer. He began studies at Louisiana State University, Baton Rouge, Louisiana in 1987 to pursue a M.S.E.E. Since returning to school, he has been employed at Quaternary Resource Investigations, Inc. in Baton Rouge as a Senior Engineer, responsible for development of a ground penetrating radar system. He is a member of the IEEE and the IEEE Geoscience and Remote Sensing Society, the NSPE, and Eta Kappa Nu.

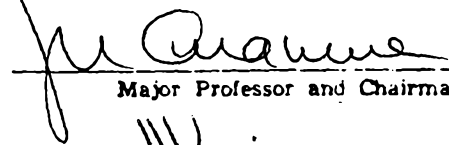
MASTER'S EXAMINATION AND THESIS REPORT


Candidate: Daran Lynn Rehmeier

Major Field: Electrical Engineering

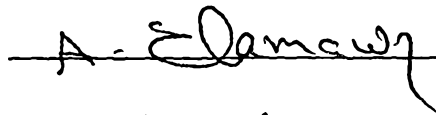
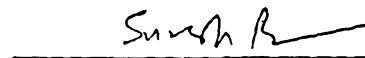
Title of Thesis: A Data Acquisition and Processing System for High Frequency Repetitive Waveforms

Approved:


Major Professor and Chairman


Dean of the Graduate School

EXAMINING COMMITTEE:

Date of Examination:

Oct 18, 1990