Second Generation In-Service Bridge Monitoring System

by

Harry W. Shenton III
Daniel A. Howell

Department of Civil and Environmental Engineering
University of Delaware

October 2003

DELAWARE CENTER FOR TRANSPORTATION

University of Delaware
355 DuPont Hall
Newark, Delaware 19716
(302) 831-1446

Second Generation In-Service Bridge Monitoring System

by

HARRY W. SHENTON III
DANIEL A. HOWELL

Department of Civil and Environmental Engineering
University of Delaware
Newark, Delaware 19716

DELAWARE CENTER FOR TRANSPORTATION
University of Delaware
Newark, Delaware 19716

This work was sponsored by the Delaware Center for Transportation and was prepared in cooperation with the Delaware Department of Transportation. The contents of this report reflect the views of the authors who are responsible for the facts and accuracy of the data presented herein. The contents do not necessarily reflect the official views of the Delaware Center for Transportation or the Delaware Department of Transportation at the time of publication. This report does not constitute a standard, specification, or regulation.

The Delaware Center for Transportation is a university-wide multi-disciplinary research unit reporting to the Chair of the Department of Civil and Environmental Engineering, and is co-sponsored by the University of Delaware and the Delaware Department of Transportation.

DCT Staff

Ardeshir Faghri
Director

Jerome Lewis
Associate Director

Wanda L. Taylor
Assistant to the Director

Lawrence H. Klepner
T^2 Program Coordinator

Sandi Wolfe
Secretary

DCT Policy Council

Carolann Wicks, Co-Chair
Chief Engineer, Delaware Department of Transportation

Eric Kaler, Co-Chair
Dean, College of Engineering

Timothy K. Barnekov
Dean, College of Human Resources, Education and Public Policy

The Honorable Timothy Boulden
Chair, Delaware House of Representatives Transportation Committee

Michael J. Chajes
Chair, Civil and Environmental Engineering

Kevin Coyle
Representative of the Secretary of the Delaware Department of Natural Resources and Environmental Control

The Honorable Tony DeLuca
Chair, Delaware Senate Transportation Committee

Raymond C. Miller
Director, Delaware Transit Corporation

Donna Murray
Representative of the Director of the Delaware Development Office

Ralph A. Reeb
Director of Planning, Delaware Department of Transportation

Delaware Center for Transportation
University of Delaware
Newark, DE 19716
(302) 831-1446

Second Generation In-Service Bridge Monitoring System

by

Harry W. Shenton III
Daniel A. Howell

Department of Civil and Environmental Engineering
University of Delaware
Newark, Delaware 19716

Executive Summary

A second generation In-Service Bridge Monitoring System (ISBMS) was recently developed by the authors. The system measures live load strains in a bridge due to site-specific traffic. This report documents the system hardware, software, laboratory tests and field tests of the ISBMS.

The original ISBMS was developed in 1999 by Shenton and Holloway at the University of Delaware. The system was designed to measure the peak strain in a bridge girder or slab due to heavy vehicles crossing the bridge. It was small, lightweight, battery powered, portable, and rapidly deployable. The first generation ISBMS was capable of operating, under attended, for up to three weeks. It proved to be a very useful tool for bridge diagnostics, but was limited by the following features: (1) data could only be retrieved by physically connecting to the system with a laptop computer in the field, (2) it was limited to capturing only peak strains and could not be reprogrammed , and (3) used a full bridge strain transducer and would not accept a quarter-bridge strain gage as input.

Building off of the success of the first generation system, an effort was made to enhance the capabilities of the ISBMS, make it user-friendlier, and simplify the setup and operation of the system. The new second generation system would incorporate the following improvements: (1) would be remotely accessible via wireless communication, (2) would be accessed through a user friendly web site, (3) would accept a quarter-bridge strain gage, as well as a full bridge strain transducer, (4) would provide three different modes of data collection, and (5) would be programmable.

The second generation ISBMS was designed using off-the-shelf components. Custom fabrication and integration of the components was completed at the University. At the heart of the system is a small but powerful single board computer (SBC), which has a 12 bit analog-to-digital converter, multiple analog input channels, and several digital I/O channels. The other major component of the ISBMS is a full duplex Cellular Digital Packet Data (CDPD) modem for wireless communication over the existing cellular phone network. Inputs are provided for a full bridge transducer and a quarter-bridge foil strain gage; signal conditioning and bridge completion is provided for the gages. The system is powered by two rechargeable batteries: one 12-volt battery powers both the modem and the computer, while a 2-volt battery powers the sensor. The batteries are sufficient to provide power for up to four weeks in the field. The system is housed in a NEMA4 rated enclosure. It weighs 22.4 lbs and has overall dimensions of 11.6 x 9.4 x 6.6 inches. Strong magnets on the back of the enclosure make for easy mounting to a steel member in the field.

A program was developed for the SBC to handle all of the data acquisition and control functions of the ISBMS. The program was written in TFBASIC, a modified version of the well-known BASIC programming language. The system has three different modes of capturing strain data: peaks, time history, and rainflow. In the peaks mode, the system waits for a strain "pulse" to exceed a user specified trigger threshold. Once the trigger is exceeded the system measures and stores the peak strain of the measured waveform. The date and time of the peak event are also stored. In the time history mode, the system waits for a user specified trigger threshold to be exceeded. Once the trigger is exceeded the ISBMS captures and stores the complete strain waveform, according to user specified parameters. Finally, in the rainflow mode, the

ISBMS counts cycles of randomly varying strain using the rainflow algorithm. The program handles all of the book keeping and data storage tasks. It also controls the power to the CDPD modem, according to user specified parameters, so that it can be turned on and off at specified times during the day to save power.

The second generation ISBMS has been designed to be accessed remotely using the cellular data network. To use the remote capabilities of the system requires a data account with a cellular provider (Verizon Wireless was used in the development of the system). With a data account, the user is charged by the amount of data that is transferred over the network, not by airtime minutes.

There are two options for accessing the system remotely, one uses a more rudimentary terminal based interface, and the other is a user friendly web-based interface. With the former, the operator interacts with the system through the Windows based program, HyperAccess. This mode of operation is useful for the more experienced operator of the ISBMS and provides access to some lower level functionality that is not available with the web interface. It is also useful to have in case the web is not available to the user. The web-interface is designed to be user friendly and intuitive and greatly simplifies the setup, control and data retrieval tasks of the system. Computer science graduate students from the University of Delaware developed the web-interface. The web-interface is intended to bring the ISBMS to the desktop of the bridge management engineer. Using either option, the user has complete control over the system from their office computer (provided there is Internet access). The user can select the mode of operation, set all of the necessary parameters for that particular mode, change the modem on-off parameters, set the gage sensitivities, download data, erase the data stored on the system, and set the date and

time of the system. Operating through the web-interface, this is accomplished by completing and submitting web forms; in the terminal based mode, this is accomplished by inputting parameters at the command prompt.

The second generation ISBMS was tested extensively in the laboratory. Tests to quantify the noise level in the strain measurements revealed a standard deviation of 0.5 με for the strain transducer and from 1.6 to 9.4 με for the foil strain gage. The latter would be expected since a quarter-bridge circuit is inherently more susceptible to noise by its design. In addition to the noise testing, the accuracy of both gages was tested. All of the tests revealed accuracy satisfactory for field-testing of a bridge.

Finally, the ISBMS was tested in the field on bridge 1-704, which carries southbound I-95 over the Christina Creek in Newark, DE. The bridge has a high Average Daily Truck Traffic (ADTT), and is composed of three simply supported spans. The main span is 62.5 feet long, with two approach spans each measuring 24.5 feet. The bridge consists of four 12-foot travel lanes, a 12-foot shoulder, and an exit lane. A full bridge transducer was mounted at a center span distance of 14 feet on the first approach span. The gage was mounted to the bottom flange of girder G5 to measure live load strain. The peak program was tested with a sample rate of 50 samples per second; a maximum tensile strain of 85 με was recorder over a period of 27 minutes. The time history program was tested at 50 and 100 samples per second. The recorded waveforms clearly show the axles of the truck as they travel across the bridge. A maximum tensile strain of 79 με was recorded. Finally, the rainflow program was tested at 50 samples per second and recorded several cycle ranges, the largest of which was 65 με

The second generation ISBMS has the potential to become a very important and effective tool for bridge management and maintenance. It is small, easy to deploy and simple to operate. It was developed for use on ordinary bridges, where the periodic measurement of strain due to live loads can be of tremendous benefit in assessing the condition of the structure. It does not require an experienced test engineer to install and operate the system. It was designed to be portable and easily moved from one bridge to another. In that way, quantitative data about an ordinary bridge's response can be obtained in a quick and easy manner. The data obtained from the system can be used for the following purposes: (1) measurement the stress in a bridge due to a permit vehicle, (2) fatigue assessment of critical details, (3) experimental load rating based on in-service data, and (4) short- or long-term health monitoring of a bridge. The full potential of the ISBMS will not be realized unless it is used in conjunction with a regular bridge inspection program.

ACKNOWLEDGMENTS

Funding for this work was provided by the Delaware Department of Transportation and National Science Foundation (Grant CMS-9874774). The authors would like to thank the sponsors for their generous support. The authors would also like to acknowledge the contributions of Messer's Michael Davidson, Douglas Baker, Gary Wenczel and Danny Richardson to the project.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 Background

Many of the bridge in the United States are approaching the end of their life cycle. State Departments of Transportation are responsible for inspecting bridges in their inventory every two years to determine the condition of their bridges. Deciding which bridges will be rehabilitated or replaced is not an easy task, because these projects can be quite costly and funding is limited.

Evaluating bridges can be accomplished by several means: visual inspection, non destructive tests, and strain monitoring all provide valuable information about the bridge. Using these techniques, local problems in a bridge member, as well as the overall condition of the bridge can be investigated.

Strain monitoring can be used to measure the overall response of a bridge, or the detailed state of stress in a very local region of a bridge. There are two types of strain monitoring: long-term and short-term strain monitoring. Long-term strain monitoring involves a permanent installation on the structure and is used to record changes over time. This type of monitoring typically starts with an initial reading, commonly referred to as a baseline measurement, which successive measurements are referenced to. Short-term, or "in-service" monitoring, is a temporary installation on the structure. The data obtained from short-term monitoring is useful in load-rating, as well as fatigue analysis of the structure.

1.2.2 Long-Term Strain Monitoring Systems

1.2.2.1 New York Long-Term Monitoring Systems

A long-term monitoring system has been developed in New York as a prototype for possible future use. The monitoring system is intended for continuous monitoring of a bridge for at least four seasons of in-service data to capture variations in temperature and weather conditions. As noted by Alampalli and Fu, the system uses a variety of different sensors including accelerometers, strain gages, and inclinometers as well as a 9600 baud modem for transferring data from the field (Alampalli and Fu, 1994). A land-line AC power supply supports the system; a ten-year lithium battery is also included to save key system configurations in case a power outage is encountered. The system is capable of recording five records for each of its 16 channels for a total of 80 records.

The system was tested on two bridges owned by Consolidated Rail Corporation (Conrail) carrying I-490 East and I-490 West rail traffic through Rochester, New York. The bridges were both built in 1963, are roughly 76 meters in length, and have three spans of steel girders. The bridges were to be rehabilitated including replacement of corroded diaphragms, repair of the concrete decks with a concrete overlay, installation of new joint systems at the approaches, and replacement of the steel rocker bearings with elastomeric bearings with load plates.

The remote bridge-monitoring system (RBMS) was installed prior to rehabilitation to capture key structural responses. As illustrated by Alampalli and Fu, the system is intended to measure certain structural indexes for assessment and monitoring including modal properties (natural frequencies, damping ratios, and mode shapes) and their derivatives (Alampalli and Fu, 1994). By measuring these structural indexes, the system is able to show deterioration or damage over time.

the system and data retrieval. The system is programmed to reset itself in the event of a power failure.

The Magazine Ditch system monitors data in two modes—slowly and rapidly. Data that is recorded slowly is referred to as "monitor" data. Recording data in this manner is intended to capture any gradual changes in the bridge due to sustained loads and environmental factors. This "monitor" data is a snap shot of all the sensor outputs taken at one hour intervals. Data is read at 100 samples per second for 5 seconds; the average, maximum, minimum and standard deviation are then calculated and stored. Data that is recorded quickly is referred to as "event" data, and is recorded any time a heavy vehicle crosses the bridge and exceeds a 50 microstrain trigger. The system records 9 seconds of data and sorts out the peak value. Shenton, et al.(2000a) have shown that the system is reliable, robust, and operates in a stand-alone manner, with little to no user intervention.

Early results from the system add some incite into the seasonal variation of loads on the bridge. Shenton et al. illustrate in terms of temperature data, both the longitudinal and transverse strain decrease with increasing temperature (Shenton et al., 2000a). The daily maximum longitudinal strain occurs at the same time as the daily low temperature, while the maximum transverse strain occurs after the daily low. The average daily stresses increase as the temperature decreases. The average strain varies up to 300 με from summer to winter months.

Bridge 1-351, completed in 1998, replaced a single span, concrete slab bridge with an advanced polymer composite structure. The goals for the monitoring system were similar to that of the Magazine Ditch bridge, to be reliable, robust, and operate in a stand alone manner. Unlike the Magazine Ditch project, the 1-351 bridge did not have a convenient place to house the monitoring equipment, so an

retrieved by pointing a radio frequency tag reader at the embedded system (Harris, 2002).

1.3 First Generation In-Service Bridge Monitoring System (ISBMS)

1.3.1 Background of ISBMS

The first generation in-service bridge monitoring system (ISBMS) was developed at the University of Delaware by Holloway and Shenton in 1999 (Holloway, 1999). The goals for the system were that it be small, lightweight, battery powered and could operate in a stand alone manner. The resulting system met the before mentioned requirements and is capable of operating in the field for up to 2-3 weeks unattended. It uses a full-bridge strain transducer in conjunction with a proprietary data acquisition system to capture peak live load strains due to site specific traffic. The data is stored in non-volatile memory and retrieved by physically driving to the bridge site and downloading the information through a hard wire serial connection.

The ISMBS is composed of a digital data acquisition system, a full-bridge strain transducer, battery pack, and an environmental enclosure. The data acquisition system is powered by a single non-rechargeable 9-volt battery, while the strain transducer is powered by 6 non-rechargeable D cell batteries. The system is housed in a National Electrical Manufacturers Association (NEMA) 4 rated enclosure. A NEMA 4 enclosure is intended to protect the system from rain, sleet, snow, windblown dust, or the formation of ice on the enclosure.

The data acquisition system is a modified Snap Shock Plus™ (SSPM4), manufactured by Instrumented Sensor Technology. The SSPM4 is a small, battery operated instrument with an on-board microprocessor, 16 kilobyte EEPROM memory,

from Philadelphia, Pennsylvania to Atlantic City, New Jersey. The structure, a steel through girder design, had a low load rating controlled by the flexural capacity of the steel deck trough. Typical rail loads were attributed to a locomotive and three passenger cars.

The bridge was instrumented with conventional strain gages as well as with the in-service system. Sixteen strain transducers were mounted to the bridge girders and trough to measure the in-situ strain experienced by the bridge. In addition to the diagnostic load test, the first generation ISBMS was placed on the bridge to record the peak live load strains for one week. The system captured data for one week's time from 163 trains with an associated 1456 axles (Chajes, et al., 2001). The conclusion from the diagnostic and in-service data suggested that the bridge should indeed have a higher load rating, and the speed restriction was lifted. This conclusion was based on the fact that the response of the bridge to each train was similar due to the reproducibility of the loads.

1.3.3 Limitations of the System

While the first generation ISBMS was a step forward in many respects toward simple, cost-effective strain monitoring, there were some noted drawbacks. The ISBMS requires on-site setup and data retrieval; there is no remote access to the system. Also, data can only be collected in one manner, i.e., capturing peak live load strains. There is no flexibility to tailor data acquisition functions to the user's preference. Other issues that were cause for concern are the non-rechargeable batteries for the system and the environmental enclosure that houses the system. The top of the NEMA enclosure is secured by four flat-head screws that are quite inconvenient for

Chapter 2

SYSTEM DESIGN

2.1 System Overview

The second generation ISBMS is based on a small but powerful single board computer, the TFX-11. In addition, the system contains a full duplex Cellular Digital Packet Data (CDPD) modem for wireless communication over the existing cellular phone network. Inputs are provided for a full bridge transducer and a quarter-bridge foil strain gage; bridge completion is provided for the quarter bridge gage. The system is powered by two rechargeable batteries, which are sufficient to provide power for up to four weeks in the field. One 12-volt battery powers both the modem and the data acquisition system, while a 2-volt battery powers the sensor being used. The system weighs 22.4 lbs and has overall dimensions of 11.6 x 9.4 x 6.6 inches. The system is shown in Figure 2.1 below.

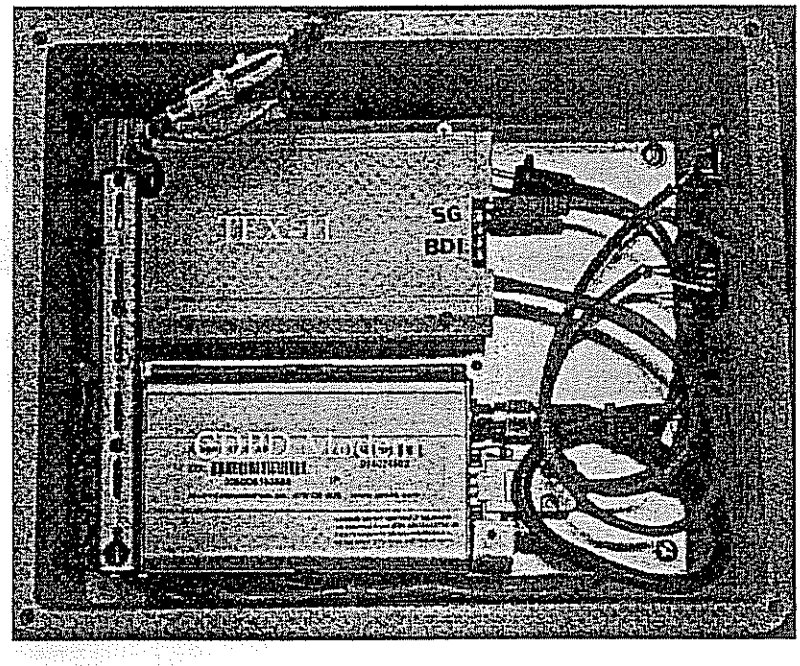


Figure 2.2 System without Battery Harness

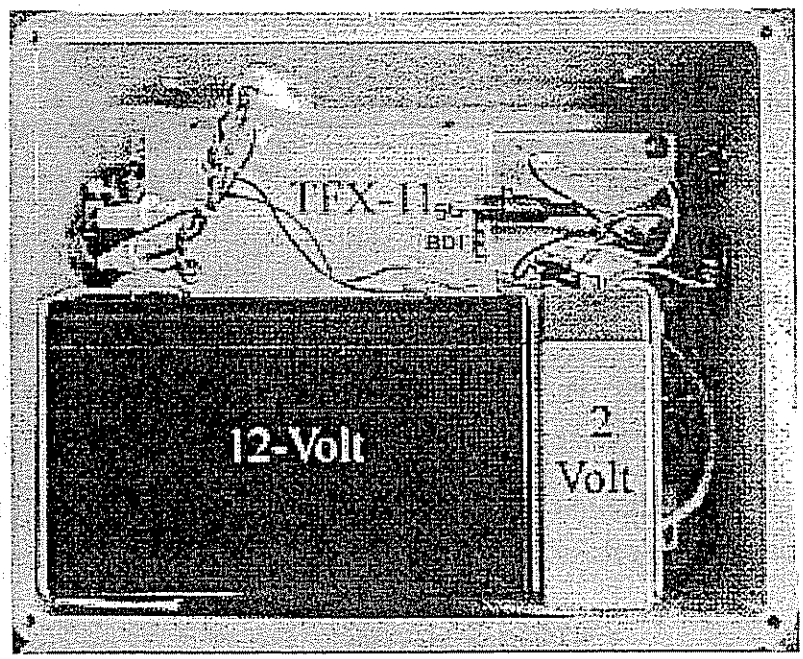


Figure 2.3 System with Battery Harness

parameters including which sub-program is running in the field, the date the program was last loaded, the gage used, and any set strain triggers. In addition, the user can download the current data file and change current modem parameters. Details of the hardware and software are presented in the sections to follow.

2.2 System Hardware

2.2.1 TFX-11

The main hardware component of the system—the TFX-11 data logger/controller engine manufactured by Onset Computer Corporation—is a small, state-of-the-art single board computer (SBC). The system is quite small, with overall dimensions measuring 2.4 x 3.2 x 0.5 inches. The TFX-11 was selected over other SBC because of its small size, low cost and power requirements. It was specifically developed, and has been used over the years, for stand-alone data logging in remote locations. The TFX-11 has 468K memory and runs on two Motorola processors, a 68HC11F1 and a PIC16C62. The memory is divided into three main areas: 21K is reserved for the operating system, 43K for the program, and 413K for data. The system contains 11 analog input channels and a 12-bit analog-to-digital converter with an input range of 0 to 5 volts. The system also has 16 TTL Digital I/O pins. The TFX-11 contains two communication ports, a parallel port and a serial port connection shown in Figure 2.5.

size, low power consumption, and overall communication capabilities. The modem is a small hardware component, with overall dimensions measuring 3 x 5.1 x 1 inches—Figure 2.6 shows the relative size of the modem. To guard against environmental factors, the Raven II is built with a rugged outer case and a removable antenna. In addition, a mounting bracket, ideal for field installations, is included with the modem.

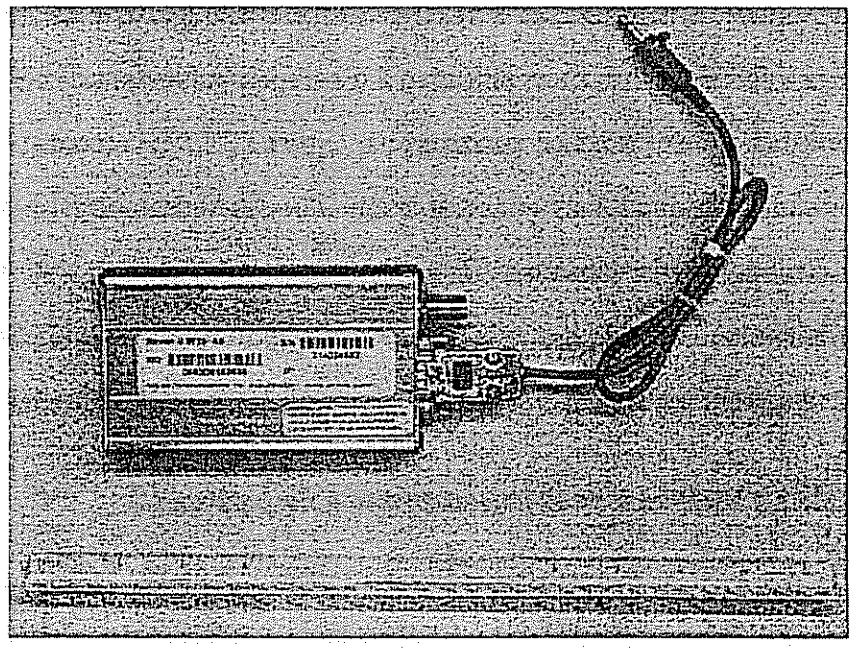


Figure 2.6 Raven II CDPD Modem

The outer case of the Raven II also contains Light Emitting Diodes (LEDs) that show the status of the modem. The LEDs indicate key parameters about the modem such as power, transmition errors, signal strength, and connection to the local network. The LEDs are visible during field deployment from a small mirror, shown in Figure 2.7, mounted inside the enclosure.

2.2.3 Other Hardware

The enclosure for the second generation ISBMS is a NEMA 4X rated enclosure manufactured by the Adalet Products Company. The enclosure is shown in Figure 2.8 with the full bridge strain transducer and modem antenna. The enclosure is a 10 x 8 x 6 inch raised cover fiberglass housing with lockable quick release latches. The NEMA 4X rating is intended to protect the system from rain, sleet, snow, windblown dust, splashing water, hose-directed water, corrosion, and the formation of ice on the enclosure. Mounted to the bottom of the enclosure are four industrial strength magnets, two 65 lb and two 120 lb capacity magnets. The magnets, shown in Figure 2.9, will allow the system to be placed on a steel bridge girder in the field.

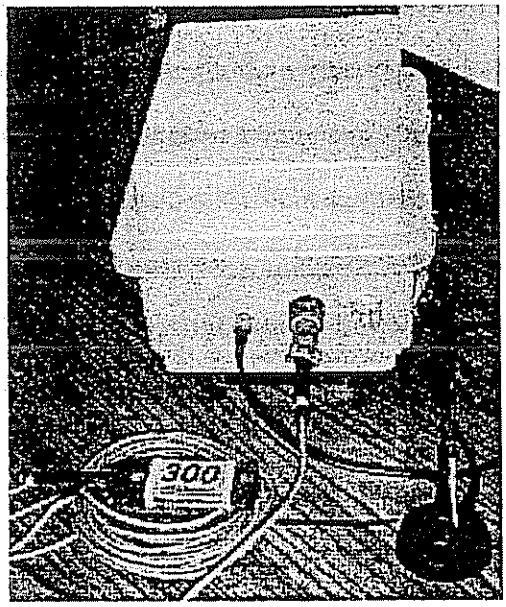


Figure 2.8 System Enclosure with Strain Transducer and Modem Antenna

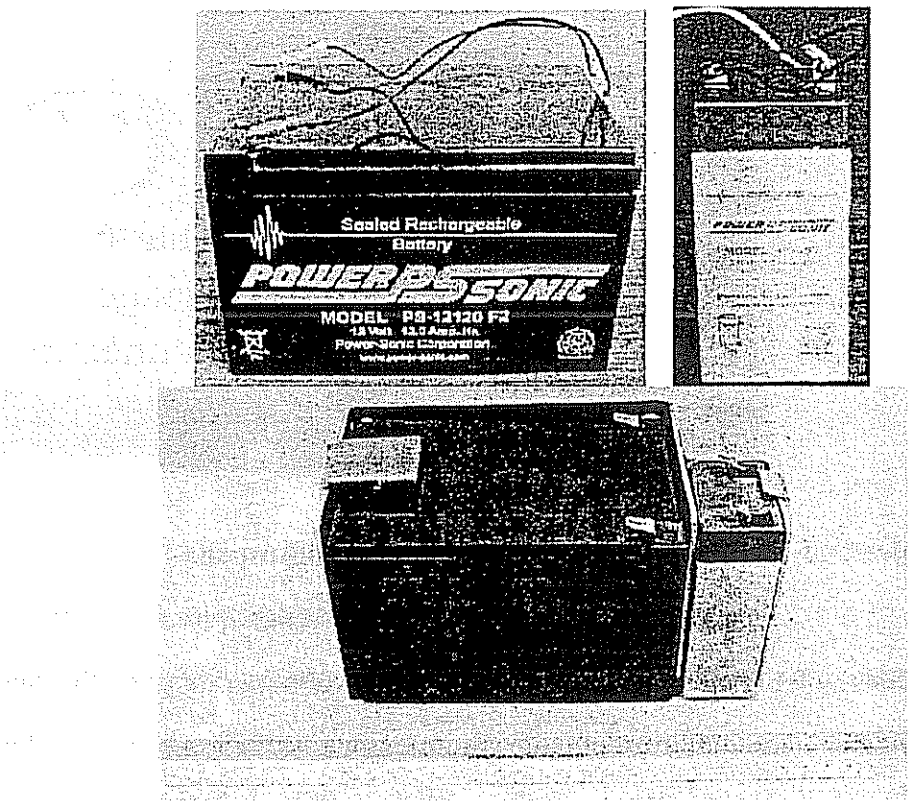


Figure 2.10 12-Volt and 2-Volt Batteries

The ISBMS is designed to measure strain using either a full bridge strain transducer or a quarter bridge strain gage. Pile Dynamics Inc. manufactures the full bridge strain transducer, which features a full Wheatstone bridge with four active 350 Ω gages. The foil gage is a quarter bridge strain gage. Any 350 Ω foil gage will work with the system; Micro Measurements manufacture the gages typically used at the University of Delaware. A model MR1-350-130 bridge completion module, manufactured by Micro Measurements, provides bridge completion for the quarter-bridge gage. Circuitry is provided for balancing the strain transducer or strain gage outputs after it is installed in the field. A close-up view of the terminals for the two gages and the connection for the modem's antenna is shown in Figure 2.11.

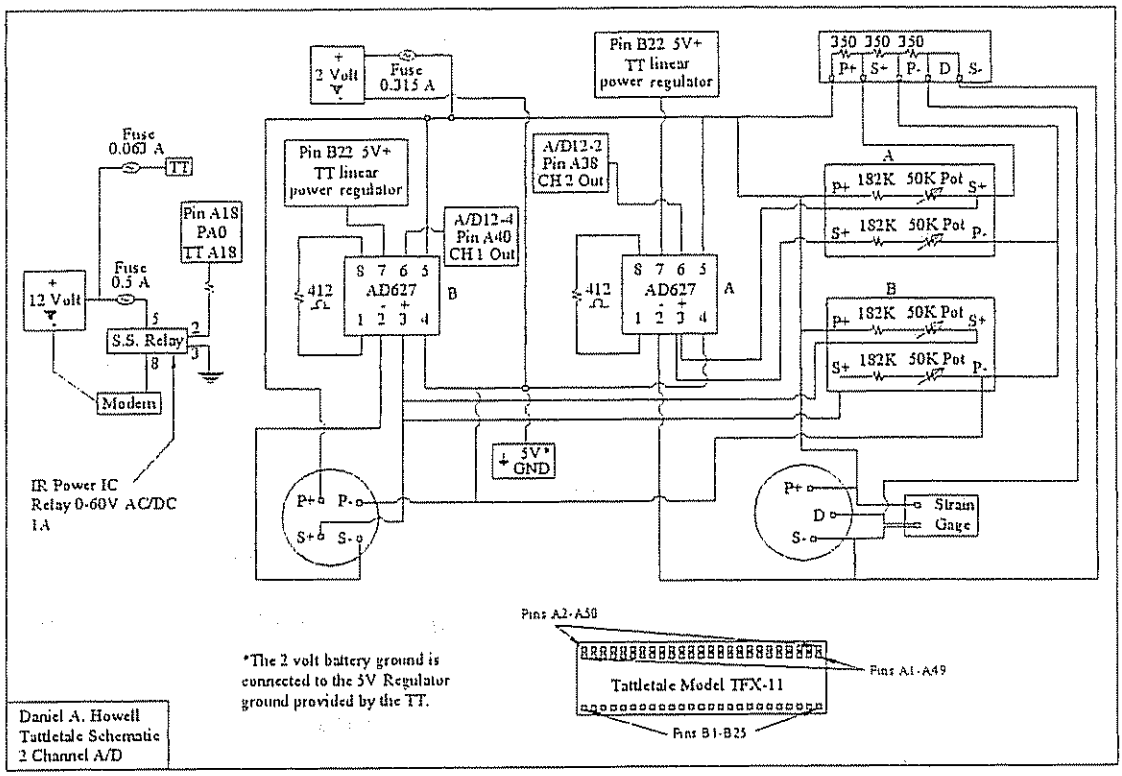


Figure 2.12 Schematic Diagram of ISBMS

The strain transducer and strain gage outputs are amplified before they are input to the TFX-11. The Op-Amp chosen for this project—model AD627-N manufactured by Analog Devices, Inc—is an integrated, micro power, instrumentation amplifier that delivers rail-to-rail output swing on single and dual (+2.2V to +/- 18 V) supplies. The AD627-N was chosen for its low power drain of 60-85 μA; direct current errors are minimized with low voltage offset, offset drift, gain error, and gain drift. The internal gain for the Op-Amp is set by an external resistor with a gain range from 5 to 1000.

The gain required for the Op-Amps was selected based on the nominal sensitivity of the full bridge transducer of 250 με/mV (an output of 0.004 mV/με) based on a gage excitation of 2 Volts, the maximum expected strain in the field and

so that power to the modem could be switched on and off when needed, for example to download data files or change system parameters. The relay chosen is a one Amp, 0-60 Volt AC/DC Microelectronic Power IC Relay manufactured by International Rectifier. The relay is controlled by a digital output on the TFX-11. The solid-state relay is engaged by setting PIN A18 on the TFX-11 to a logic high (+5 Volts), thus powering both the TFX-11 and the modem. Once PIN A18 is set to a logic low, the relay is disengaged, effectively turning off the modem.

Three fuses are included in the electrical circuitry to protect the main system components. A 0.63 Amp fuse is in place from the 12-volt battery to the TFX-11. A second 0.5 Amp fuse is also in place from the 12-volt battery to the solid-state relay that in turn powers the modem. A 0.315 Amp fuse links the 2-volt battery to each of the two Op-Amps used by the full bridge strain transducer and the quarter bridge foil strain gage. These fuses are intended to safeguard the system in the event of a short circuit or other unforeseen problem.

2.3 System Software

The second generation ISBMS requires three software platforms to operate: software associated with the TFX-11, the Raven II CDPD modem, and the software related to the web-server.

2.3.1 TFX-11 Associated Software

Programs for the TFX-11 are developed in the TFTools environment. TFBASIC files are written, compiled, and loaded onto the TFX-11 from within TFTools. TFTools contains a syntax checker that facilitates debugging. In addition, TFTools can download data files from the TFX-11 through a hard-wired serial or

selecting the new COM port, the virtual modem allows TFTools to communicate directly over the internet, with no phone lines. For our application, the new COM port directs all serial communication to the IP address of the modem. In this manner, the TFTools software is fully functional.

2.3.2 Modem Software

The Raven II CDPD modem came equipped with the software package Wireless Ace, to assist the user with many components of the modem's operation. Wireless Ace is a Window's based software package that can be used to change certain modem parameters. For example, the first time the modem was powered on, Wireless Ace was used to set the IP address of the modem, as well as the side preference, baud rate, and parity. Wireless Ace also shows the relative signal strength of the modem in relation to the nearest cellular tower. Wireless Ace contains a copy of the LED information that is on the outer case of the Raven II, giving information about power, transmitting errors, and data being received or transmitted. Wireless ACE can be used to check the status of the modem when it is in the field at a remote location.

2.3.3 World Wide Web Software

The software needed to link the system to the World Wide Web includes a web-server and other intermediate software. The web server runs on a computer in The Department of Civil and Environmental Engineering at the University of Delaware. The server runs on Internet Information Server (IIS 5.0) and requires a Windows XP platform (the server will not run on a Linux machine).

The second software platform required for web access to the system is HyperACCESS, manufactured by Hilgraeve Inc. HyperACCESS is based on an

A glossary of the functions used in the code for the DOS and Web-based files, as well as the files themselves can be found in Appendix A.

The DOS-based version of the main program contains a series of user prompted questions and answers; it is very interactive and structured giving the user a solid understanding of key system parameters. The Web-based version of the main program is very fast and requires no user interaction beyond the web interface. It is essentially in the same format as the DOS-based version but instead of having the user prompted questions printed to the screen, the questions are implicitly asked. The answers to these questions are sent as a series of numbers, through HyperACCESS to the TFX-11. A flow chart showing the questions that are implicitly asked, typical answers to the questions, and the format of typical answers are shown in Figure 2.13.

program uses specific information about the strain versus time curve, an example of which is shown in Figure 2.14, to capture the peak strain in an efficient manner. Figure 2.15 shows a typical lower trigger event.

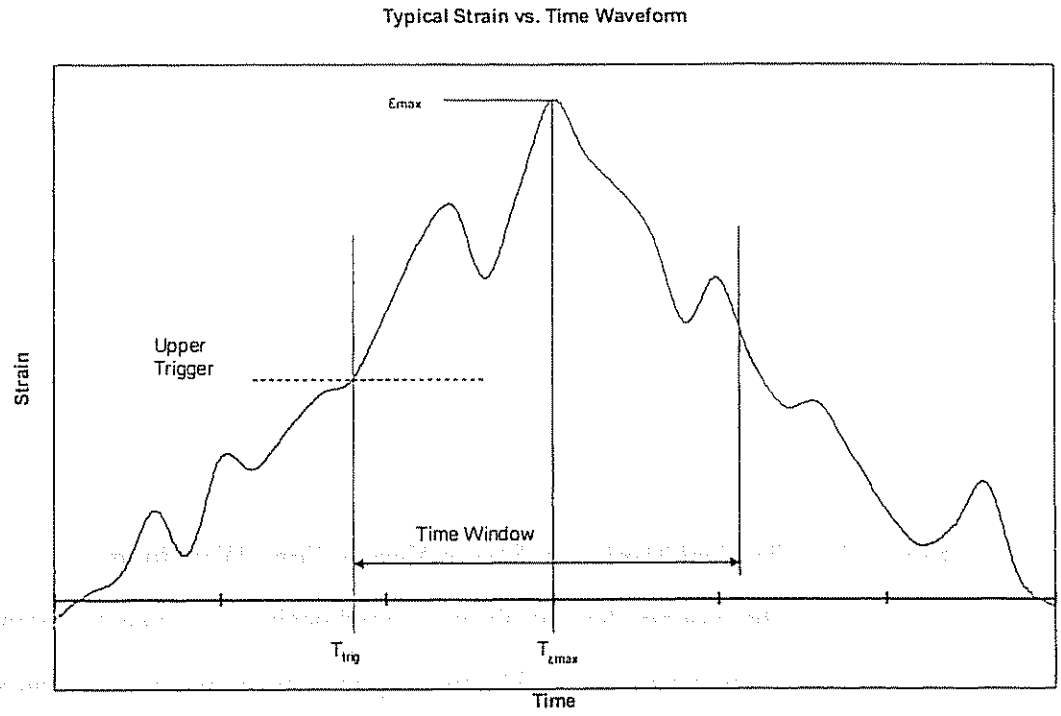


Figure 2.14 Typical Maximum Strain Versus Time Waveform

recording low amplitude compressive strains. Correspondingly, if the user intends to primarily capture minimum strain values, the upper threshold should be set high, to prevent recording low amplitude tensile strains.

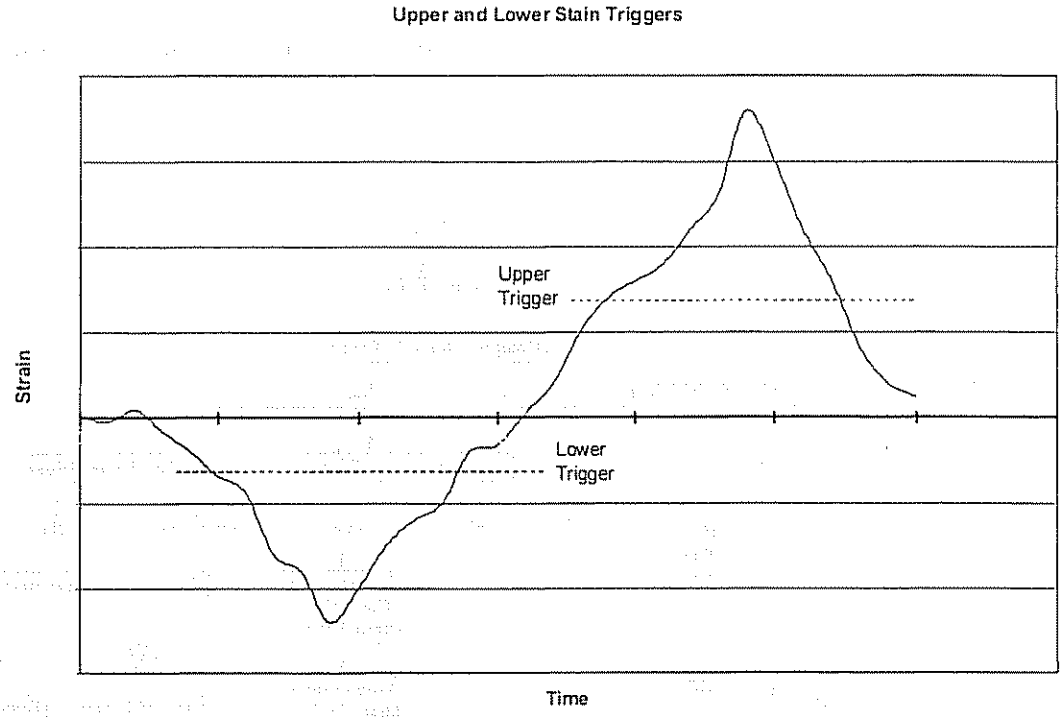


Figure 2.16 Upper and Lower Triggers

The user is required to input several key parameters for the peak program. The program asks the user to enter the upper and lower trigger values in microstrain (Figure 2.16). In addition, the user must enter the sample rate of 25, 50, or 100 samples per second. Finally, the system asks the user to enter the time window in $1/100^{th}$ of a second to capture an event (a response of 100 would equal 1 second).

The program sorts to find the maximum or minimum value in a precise manner. For example, if 50 points are sorted to find a maximum value, the program will assign the maximum value to point 1 at the start of the sorting algorithm. It then compares point 2 to the current maximum, point 1. If point 2 is smaller than point 1, it is discarded and the program compares point 3 to the current maximum value. If point 3 is larger than the current maximum (still point 1), then it is assigned the current maximum value. The program continues comparing successive points to the current maximum value, until all the data points have been sorted, and the current maximum is stored as the global maximum value. A similar routine is followed for a minimum value.

If an upper trigger is experienced, followed by a sudden compressive strain, as shown in Figure 2.18, the program will record point A, the largest positive or tensile strain. That is, if the upper trigger is exceeded, the system will locate the maximum positive strain, regardless if there is a larger negative value. The same would hold true if a lower trigger is experienced followed by a large tensile strain, shown in Figure 2.19. The system would again record point A, the largest negative or compressive strain. Again, if the lower trigger is exceeded, the system will locate the maximum negative strain, regardless if there is a larger positive value.

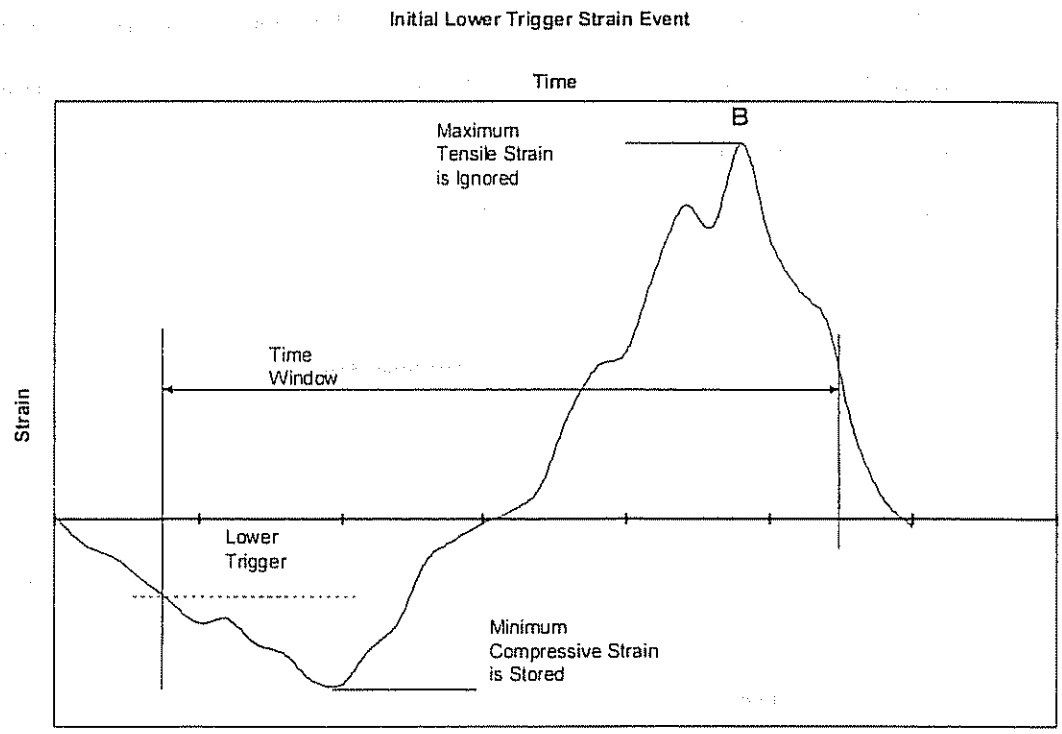


Figure 2.19 Initial Lower Trigger Strain Event

The date and time are stored along with the maximum or minimum strain in a data file on the TFX-11, the rest of the waveform is discarded. Due to the nature of the peak program, the date and time are stored once the trigger is exceeded, not when a maximum or minimum strain is actually experienced. The system automatically resets itself after an event is recorded and begins monitoring for the next strain event. The system can store 46,973 peak strain events with the available memory.

Alternate methods for capturing the maximum or minimum strain were explored, but not adopted for reasons of speed and efficiency. The algorithm adopted may not capture the true peak strain if the time window is too short. For example, the waveform in Figure 2.20 shows a peak strain event. If the current sorting algorithm is

For this example,

$$Time = \frac{100\,feet}{70mph * (5280\,ft / 1mile) * (1hr / 3600\,\text{sec})}$$

$$T = 0.974 \text{ Seconds}$$

Therefore, with a sample rate of 100 points per second, and a time of 0.974 seconds, a peak will be captured for this vehicle axle if 100 points are sorted, in other words, if a one second time window is specified. The key point is that the time window must be greater than the time it takes for all vehicle axles to cross the bridge.

Another sorting algorithm for peak strains would be to store strain values once the upper trigger is exceeded and continue storing values until the readings fall below the upper trigger, ensuring that the peak value is indeed captured. This method would work for minimum strain values as well. However, using this approach, the TFX-11 must constantly compare the current strain value with the trigger value, which requires more computational effort and would slow the system down.

Finally, another sorting algorithm would be to compare adjacent points x_i and x_{i+1} to confirm that the waveform is increasing or decreasing. This routine has two drawbacks. The first is that the TFX-11 would have to compare individual data points, involving extra computing effort and generally slowing the system down. The second drawback relates to the jagged shape of the strain versus time curve attributed to measurement noise. If the before mentioned routine was used, it would only capture *local* maximum/minimum values.

trigger that is stored. The pre and post-trigger time window allows variability with waveform size.

The time history program operates on a principle similar to that of the peak program. A flow chart illustrating the program's operation is shown in Figure 2.22. The program constantly reads the strain from either the full bridge transducer or the foil strain gage until the upper or lower threshold is exceeded. Data is read into an array that is constantly overwritten so that once a threshold is exceeded, the pre-trigger data is automatically recorded from the array. The program then records a predetermined number of data points constituting the post-trigger data. The time and date that the threshold is exceeded is stored. The system then re-arms itself after an event is captured and waits for the next trigger. Based on a typical waveform of 100 points, the system is capable of storing 2032 strain events.

2. If there are less than three points, go to Step 1. Form ranges X and Y using the three most recent peaks and valleys that have not been discarded
3. Compare the absolute values of ranges X and Y
    a. If X<Y, go to Step 1
    b. If X>Y or X=Y, go to Step 4
1. If range Y contains the starting point S, go to Step 5; otherwise, count range Y as one cycle; discard the peak and valley of Y; and go to Step 2
2. Count range Y as one-half cycle; discard the first point (peak or valley) in range Y; move the starting point to the second point in range Y; and go to Step 2
3. Count each range that has not been previously counted as one-half cycle.

This process is illustrated by the following load history:

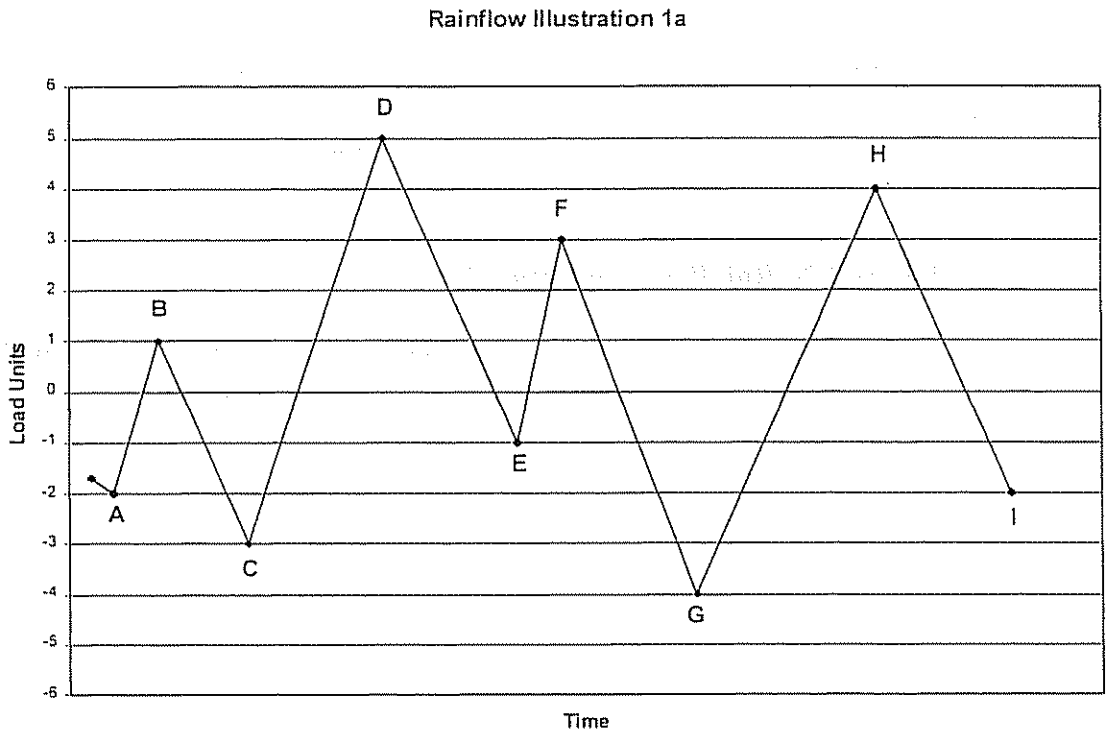


Figure 2.23 Rainflow Illustration 1a

1. S = A; Y = |A-B|; X = |B-C|; X > Y. Y contains S, that is, point A. Count |A-B| as one-half cycle and discard point A; S = B as shown below:

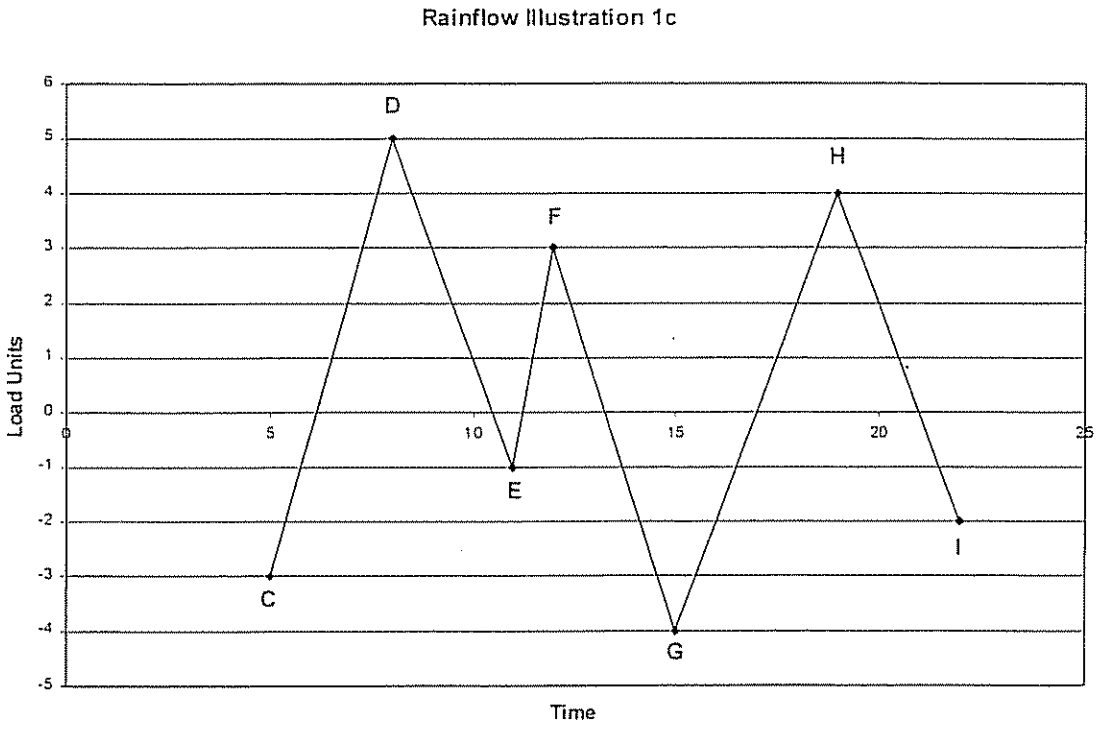


Figure 2.25 Rainflow Illustration 1c

3. Y = |C-D|; X = |D-E|; X < Y
4. Y = |D-E|; X = |E-F|; X < Y
5. Y = |E-F|; X = |F-G|; X > Y. Count |E-F| as one cycle and discard points E and F. Note that a cycle is formed by pairing range E-F and a portion of range F-G as shown below

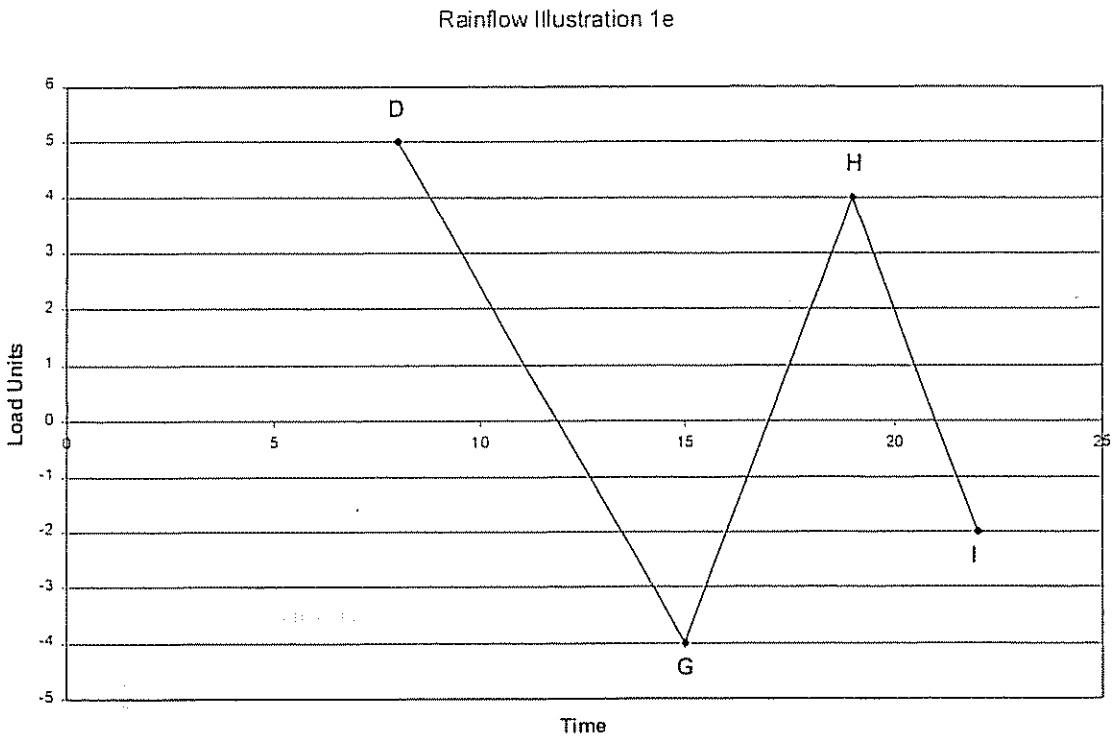
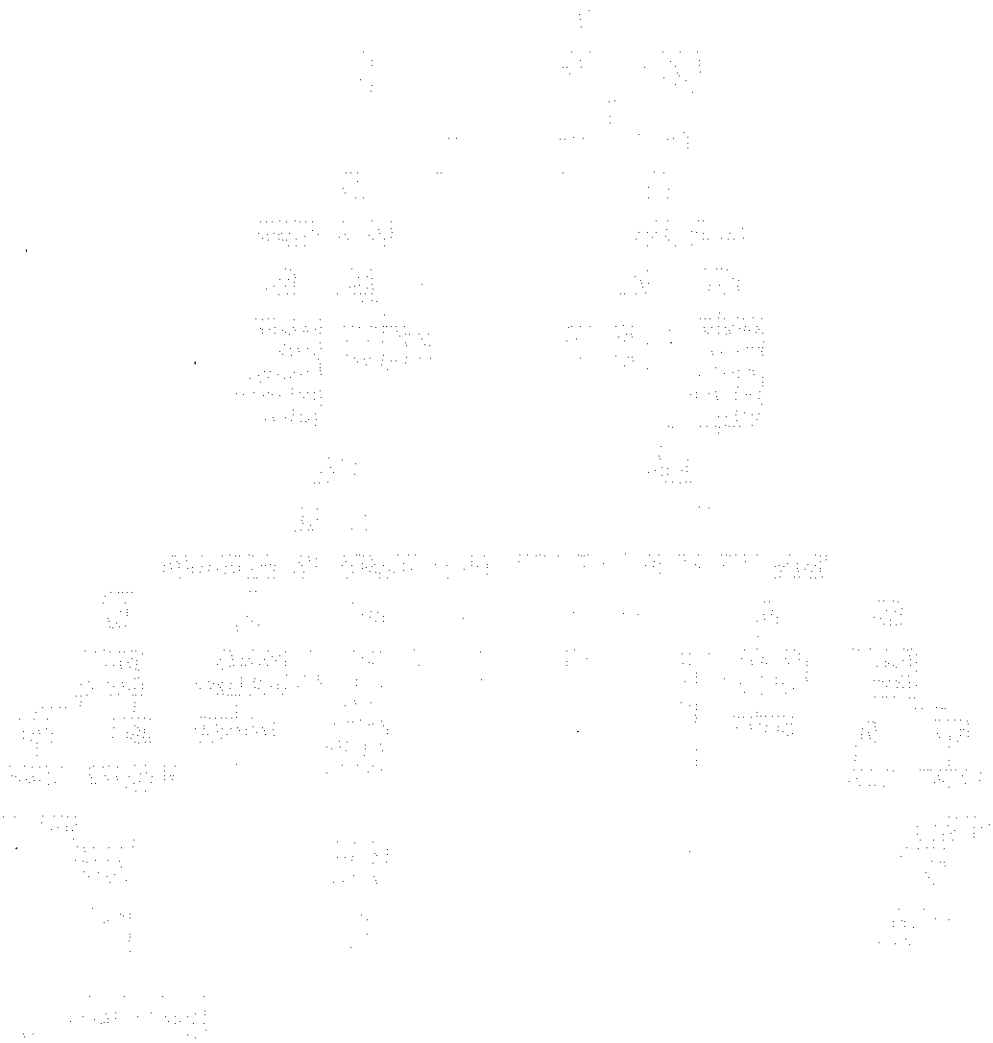


Figure 2.27 Rainflow Illustration 1e

7. Y = |D-G|; X = |G-H|; X < Y.
8. Y = |G-H|; X = |H-I|; X < Y. End of data.
9. Count |D-G| as one-half cycle, |G-H| as one-half cycle, and |H-I| as one-half cycle.
10. End of counting.

The sequence produced Table 2.2 and the histogram shown in Figure 2.28.

The rainflow program modified from the ASTM standard for the TFX-11 is very efficient in terms of the data stored. A flow chart illustrating the program's operation is shown in Figure 2.29. The program stores the time and date with each cycle count of 0.5 or 1.0 and the cycle range.



The rainflow program written for the TFX-11 utilizes a user supplied positive and negative noise threshold so that low amplitude strains are initially ignored. The user is required to enter an upper and lower noise threshold as well as a resolution strain level. These three items are required because of the way the rainflow program sorts maximum and minimum values in the field. The maximum and minimum strain values, shown in Figure 2.30, may be local, not absolute values, so the zero strain reading could not be used for a reference. This situation was overcome with a constantly updated maximum or minimum value as shown in Figure 2.31.

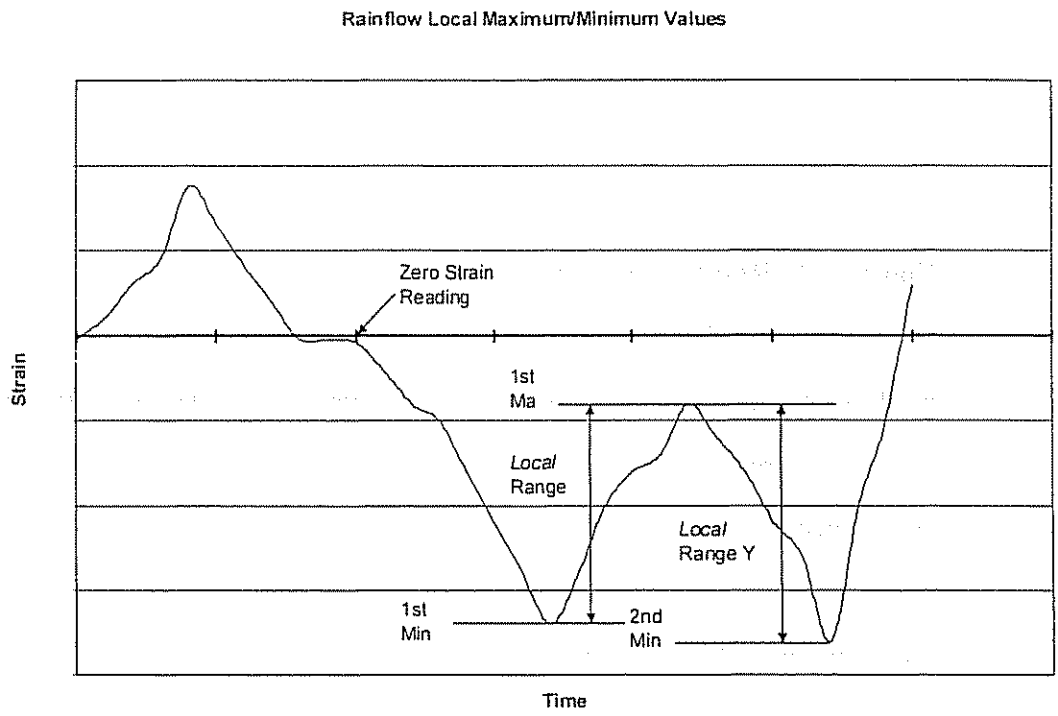


Figure 2.30 Rainflow Local Maximum/Minimum Values

minimum value that is 5 με lower than the last upper trigger. For this algorithm, the program will always find a sequence of max-min-max or min-max-min.

2.5 Field Setup

Presented in this section are details on setting up the system in the field for a typical test.

2.5.1 Gage Setup

The first step in setting up the system for field use is to mount the transducer or foil strain gage to the bridge. The location of the transducer/gage will depend on the purpose of the test. If the purpose of the test were to measure the overall global response of the bridge, the transducer/gage would normally be mounted near mid-span in a single span bridge. If the purpose of the test is to gather information about a fatigue critical detail, the gage may be mounted on, or in the vicinity of the detail.

The full bridge transducer can be mounted to the bottom flange of a steel girder with two C-clamps. Clamping should be sufficient to hold the gage on the girder but not tight enough to introduce bending in the transducer. Mounting the transducer to non-metallic girders is accomplished with a quick-setting epoxy and special mounting tabs.

The foil strain gage is quite small and is ideal for use in areas that the BDI transducer is not able to access. Special attention should be given to mounting the foil strain gages. The gages are either bonded or welded to the surface of a bridge component after the surface has been properly cleaned. Lead wires are then soldered to the gages and connected to the system enclosure. The gage being used should be

program burned into memory cannot be changed on site, without the use of a labtop and a parallel cable.

Alternatively, a program can be loaded onto the TFX-11 using the *RUN* command. Unlike the *LAUNCH* command, the *RUN* command sends a program to the TFX-11's RAM, but is lost if power to the system is cut. While the *LAUNCH* command is sent through a parallel connection, the *RUN* command is sent through a serial connection. The advantage of this method is that different programs can be easily interchanged in the field with a labtop and a serial cable (for example, if the DOS-based program was to be executed and not the web-based program). The disadvantage of this method is that the program will be lost if power to the system is cut.

To RUN or LAUNCH the program for onsite or laboratory setup, the user will need the ISBMS, a portable lab top or desktop PC and either a DB9 to 3.5mm stereo phone jack Serial cable (to RUN) or a DB25 to 9-pin mini-DIN Parallel cable (to LAUNCH). The computer being used should have TFTools software and the Web version of the main program. With the serial or parallel cable connected from the computer to the TFX-11, the TFTools environment should be up and running. The 12-volt and 2-volt batteries should next be connected.

The TFTools environment is used to download the main program to the TFX-11 and to set key system parameters. In TFTools, there are two main windows, a Terminal window and an Editor window. Figure 2.32 shows an active Terminal window in front of the Editor window containing the DOS version of the main program. The Terminal window is displayed when TFTools is first started and displays characters that are received by the host computer serial port. There can only

11. Moving the mouse to the File menu, located in the upper left hand corner of the TFTools environment, the user must select the '*OPEN*' command and select either the web or DOS-based file; the file will then be shown in the Editor window.

To 'RUN' the current program, the cursor must be in the Editor window of that particular program (TFTools is capable of multiple Editor windows). The user must then go to the *Tattletale* pull-down menu, and select '*RUN*' to send the program to the TFX-11. Selecting the '*RUN*' command will load the program into RAM on the TFX-11, and begin executing. Note, if the '*RUN*' command is selected while the terminal window is active, i.e. the cursor is in the Terminal window, it will execute the program currently loaded, regardless if the current program was loaded using the '*RUN*' or '*LAUNCH*' command.

To '*LAUNCH*' the current program the user must go to the *Tattletale* pull-down menu, and select '*LAUNCH*'. Selecting the *LAUNCH* command will cause the program to be tokenized and burned into the TFX-11's flash storage. While the *RUN* command is executed from a serial port, the *LAUNCH* command is executed through the parallel port. When the *LAUNCH* command is executed, the data file is erased and the data file pointer is reset to 0. If power is removed from the system, the program will not be lost. When the power is reconnected to the system, it will begin running the last program that was launched into its flash storage.

2.5.2.1 Terminal-Based Interaction

The DOS based program is a good tool to illustrate what is required of the user for changing system parameters if the web environment is not available. It is used here to demonstrate what parameters are needed for field setup of the system. Recall that TFTools is completely functional with the use of COM/IP middleware, in

The next parameter the user must address is the modem setup. Because the modem uses quite a bit of power, the system has been designed so that it can be turned on and off at specified times during the day, thereby saving power. The modem can be turned on at the top of any hour for a specific number of minutes; for example, it could be turned on at 2:00 am for 15 minutes in order to download the data file or change system parameters. The modem can also be left on all the time if so desired.

The user can view/change the modem parameters by entering '2' from the main menu. The program then prints:

```
There are two modes of operation for the modem
1. The modem is constantly on
2. The modem is on periodically
Enter which mode you would like to enter now:
```

If the user enters '1', the system will turn the modem on and return to the main menu. If the user enters '2', the program then prints:

```
Currently, the modem will be powered on starting at 13 hundred hours for 18
minutes. To change this scenario, press 2 now. To maintain it, press 1 now:
```

If the user enters '1', the system will return to the main menu; a '2' response will prompt the user to enter the hour (in military time) to turn the modem on. The program then asks the user to enter the amount of time in minutes the modem is to be kept on; the user is then sent back to the main menu.

The user must next input the parameters for the full bridge transducer or the foil strain gage. A '3' response from the main menu will print the following sequence:

calibration factor, the modem configurations, and the program currently running. The remaining header information is unique to the three sub-programs based on the parameters needed for each specific program.

Once the program is running, the user must disconnect the serial cable from the TFX-11, insert the serial cable from the Raven II CDPD modem into the TFX-11, and shut the enclosure. The user must then mount the enclosure to the bridge or some secure location and mount the modem antenna to the bridge girder.

2.5.2.2 Web-Based Interaction

The web-based program is similar to the DOS-based program except that the questions are not printed to the screen: responses are sent by selecting the appropriate radio buttons from a form-based web page. A figure of the web environment is shown in Figure 2.33.

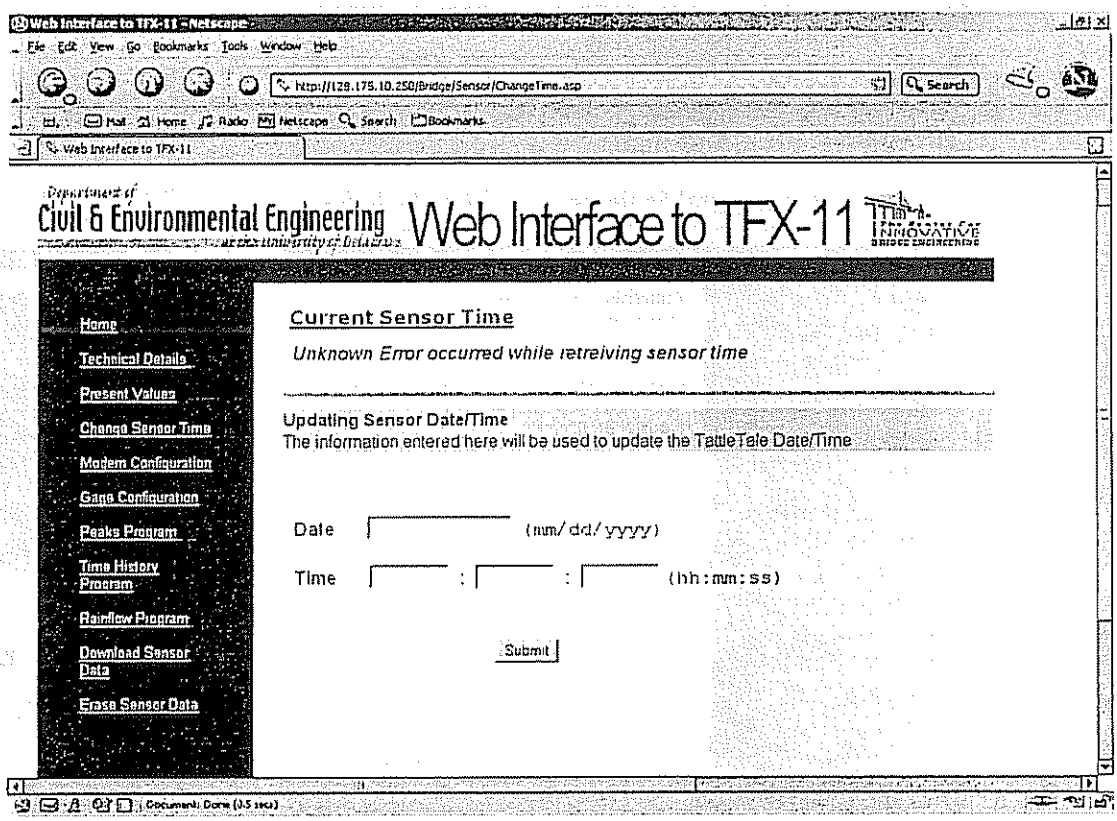


Figure 2.34 Change System Time Web Page

The modem parameters are accessed from the web page by selecting the 'Modem Configuration' page from the main menu. This will display a form, shown in Figure 2.35, for powering the modem on constantly or periodically. If the modem is to be powered periodically, the form requires the hour to turn on the modem as well as the time window to keep the modem on. The user must select the submit button at the bottom of the page to send the data to the TFX-11.

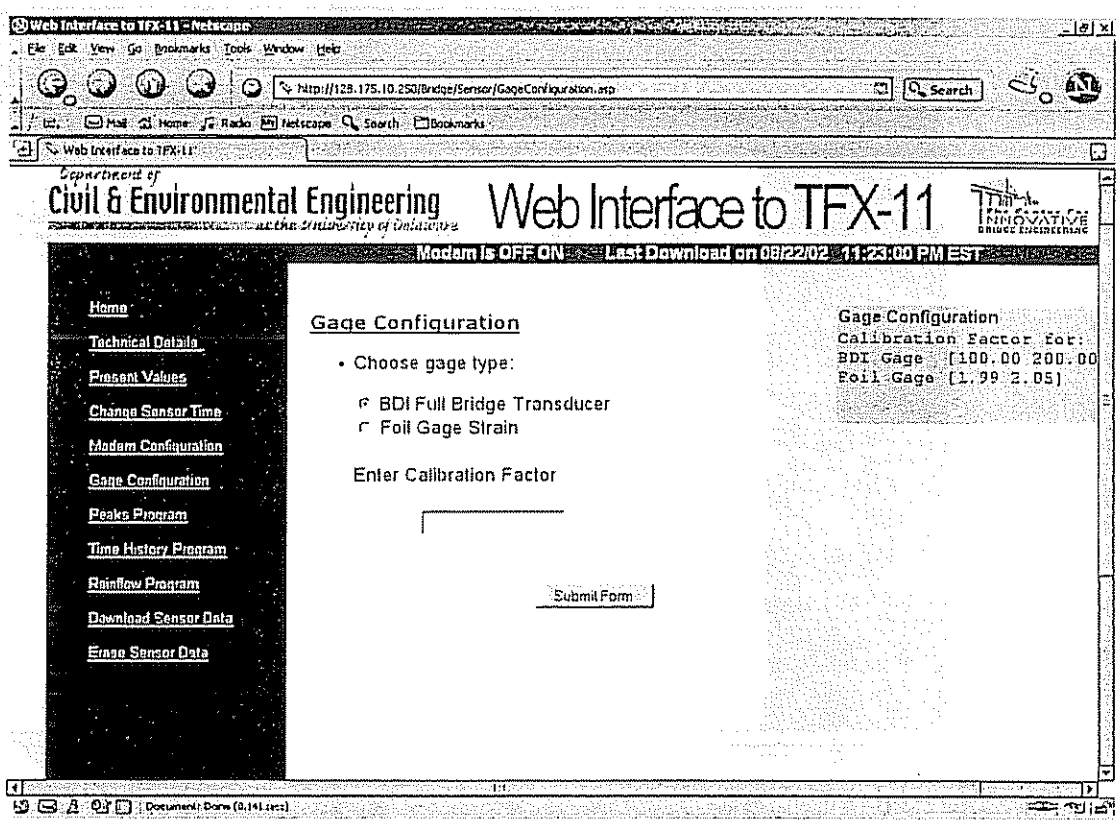


Figure 2.36 Gage Configurations Web Page

The three programs are each selected from the main menu. The three pages are shown in Figures 2.37, 2.38, and 2.39 respectively. The peak and time history web pages display a form requiring the upper and lower trigger ranges, the sample rate, and the time window(s) in $1/100^{th}$ of a second. They both feature a submit button for sending the data to the TFX-11. The rainflow program displays a form requiring an upper and lower trigger range, and a strain resolution value. The page also contains a submit button for sending data to the TFX-11.

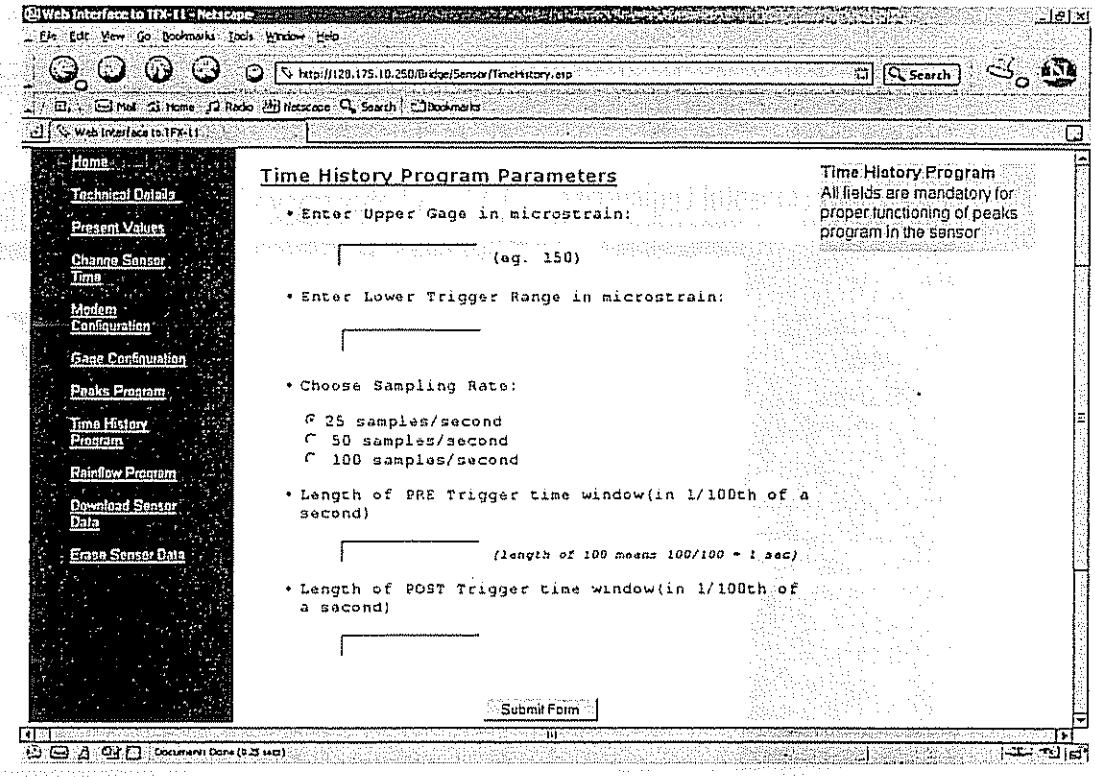


Figure 2.38 Time History Program Web Page

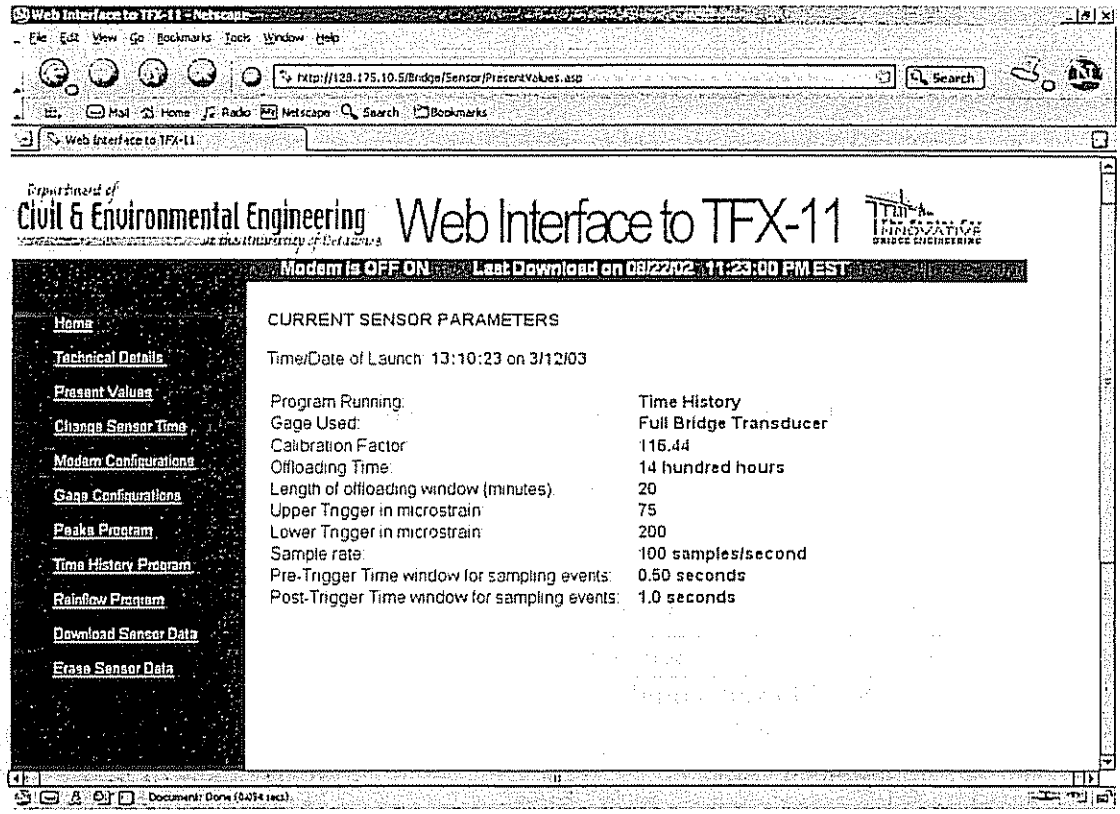


Figure 2.40 Current Parameters Web Page

2.5.3 Offloading Data from the TFX-11

Offloading data from the TFX-11 involves the use of a particular File Transfer Protocol (FTP). The TFX-11 uses a variation of XMODEM called XMODEM Cyclic Redundancy Check (CRC), which sends data in 128-byte packets. XMODEM CRC operates in the following manner: the sender sends one 128-byte packet to the receiver. The receiver then echoes the size of the packet back to the sender. If the size is the same as that originally sent by the sender, another packet is sent. If the size of the packets is not the same, the protocol attempts to resend the packet again. This sequence repeats itself until all the data is received.

To off-load the datafile using serial port commands

Step	*Host PC Action*	*TFX-II Response*
1	Send Ctrl-O	Echo Ctrl-O
2	Sent Ctrl-T	Responds with total size of datafile as 8 hex characters followed by CR/LF. Bytes are send MSB first.
3	Send Ctrl-N	Responds current datafile pointer as 8 Hex character followed by CR/LF. Sends MSB first.
4	Send number of bytes to off-load as 8 hex characters followed by CR/LF. Sends MSB first.	
5		Responds Ctrl-Z (query for Block 0)
6	Send Ctrl-B to request Block 0 or any other character for NO. There is a < 1 second time-out for this.	Echo character
7	Start XMODEM Checksum in HyperACCESS	Respond to XMODEM checksum

An example illustrating how to download the data file from HyperACCESS is shown here. Figure 2.41 shows a typical HyperACCESS window with some characters shown below the '#' prompt. The first character visible after the '#' prompt is a small box corresponding to the Ctrl+O character sequence (step 1). A Ctrl+T sequence is then sent to the TFX-11, and it echoes back the total size of the data file as 00075000 (step 2). A Ctrl+N character is then sent to the TFX-11 resulting in an echo of the current data file pointer in hexadecimal form as 00000093 (step 3). The 00000093 number is entered by the user a second time followed by a carriage return/line feed (1 press of the enter button) and sent to the TFX-11 (step 4). Because of the 1-second timeout for the next step, it was performed immediately after the

show the types of XMODEM available to the user. From this menu, shown in Figure 2.43 the user must select the radio button for XMODEM Checksum. The pull down menus for *Seconds to wait to receive for each packet (1-60)* and *Seconds to wait to receive for each byte (1-60)* should be set to 1 second. The remaining pull down menu for *Attempts to send each packet (1-25)* should be set to 25.

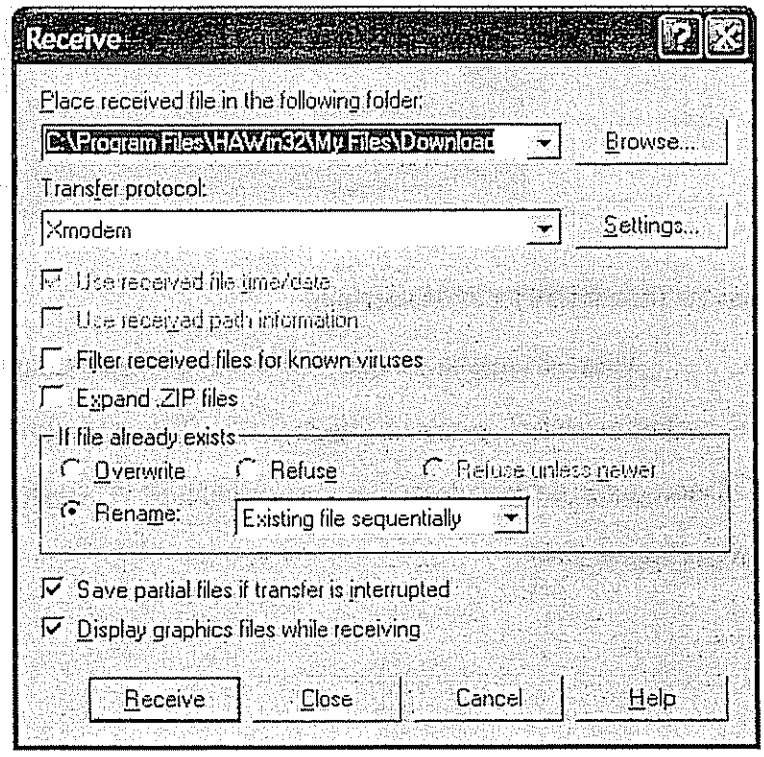


Figure 2.42 HyperACCESS File Received Settings

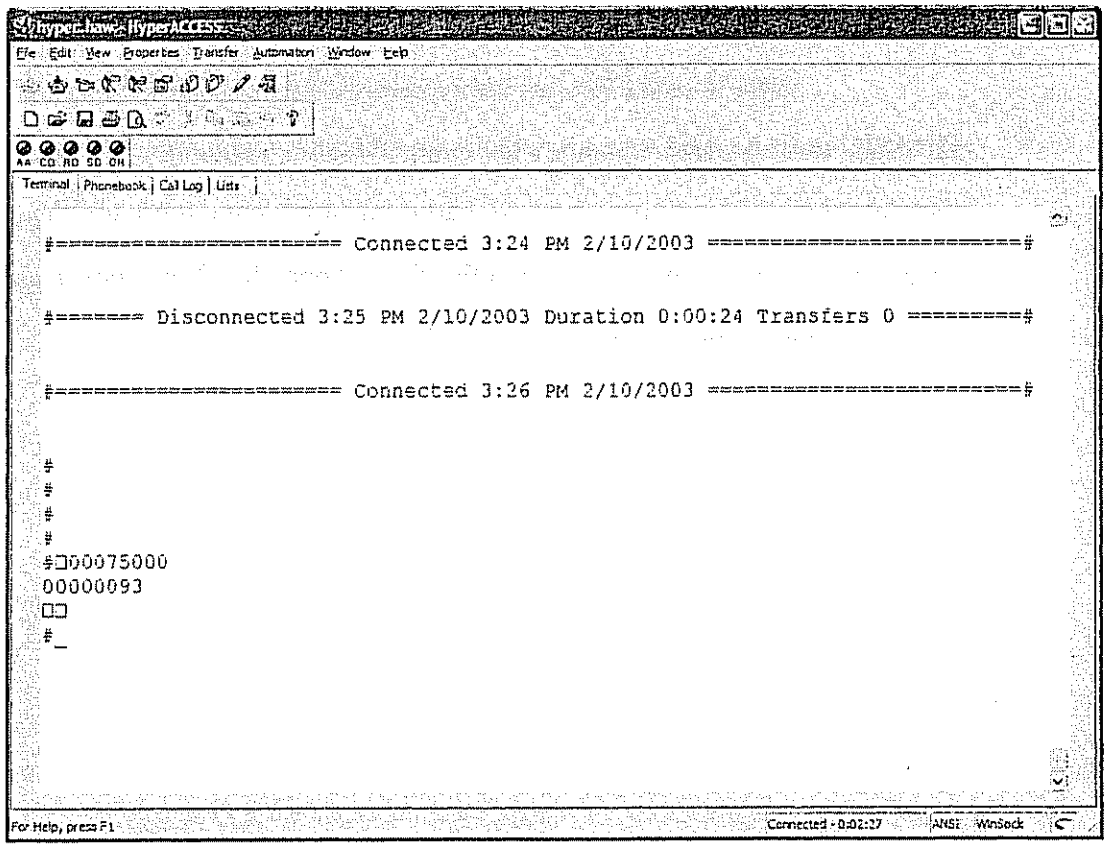


Figure 2.44 Evidence of Complete Download

2.5.3.2 Web-Based Offloading

Downloading the data file from the web page is quite easy. From the main menu page, the user must select the 'Download System Data' page and simply select the submit button, no further action is required. The data is downloaded from the TFX-11 and stored in the folder location specified in figure 2.42. After downloading the data file, the user must then erase the data file by selecting the 'Erase System Data' page and selecting the submit button.

Chapter 3

LABORATORY AND FIELD TESTING

3.1 Introduction

To verify the operation and accuracy of the second generation ISBMS, several tests were performed on the system both in the laboratory and in the field. Tests conducted in the laboratory investigated how external noise and interference affected the readings of the full bridge transducer and the foil strain gage. In addition, the system was tested in the field on a bridge.

3.2 Noise Testing

Several tests were performed to measure the signal noise in the strain gage and the BDI strain transducer readings from the ISBMS. A sample program was written for the TFX-11 to record the strain from the full bridge strain transducer and the foil strain gage for 5 minutes with no load on the gages; the ISBMS was detached from any outside source (labtop, PC, etc.) during the test. Results for the full bridge transducer showed little variation in strain; the standard deviation for the full bridge was 0.5 με.

The foil strain gage is more sensitive to interference than the full bridge transducer because only one resistor in the Wheatstone bridge is active (i.e., it is a quarter-bridge sensor). Full bridge sensors are inherently more immune to noise than quarter- or half-bridge sensors. In addition, the foil gage does not have a hard rugged

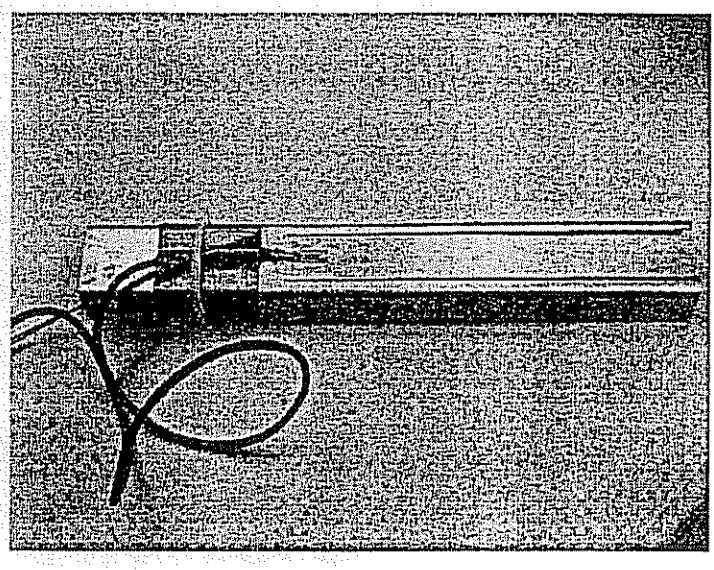


Figure 3.1 Strain Bar

To measure the strains accurately, a Micro Measurements P-3500 Strain Indicator, shown in Figure 3.2, was used. The test was performed by first attaching one of the strain gages from the strain bar to the P-3500 Strain Indicator. The gage was then calibrated using the P-3500 shunt calibration circuit. To do this, the P-3500 shunts the active gage with a precision resistor that simulates 5000 με. The gage factor is then adjusted on the P-3500 until a reading of 5000 με is achieved. This gage factor was then used to calculate the sensitivity factor for the ISBMS as described in Appendix B.

Once the P-3500 was calibrated, an object was inserted into the strain bar to induce a certain strain. Figure 3.3 shows the strain bar attached to the P-3500 Strain Indicator, with the bar flexed and held constant.

strain); this is shown in Figure 3.4. The ISBMS then recorded the strain from the same gage and the two values were compared.

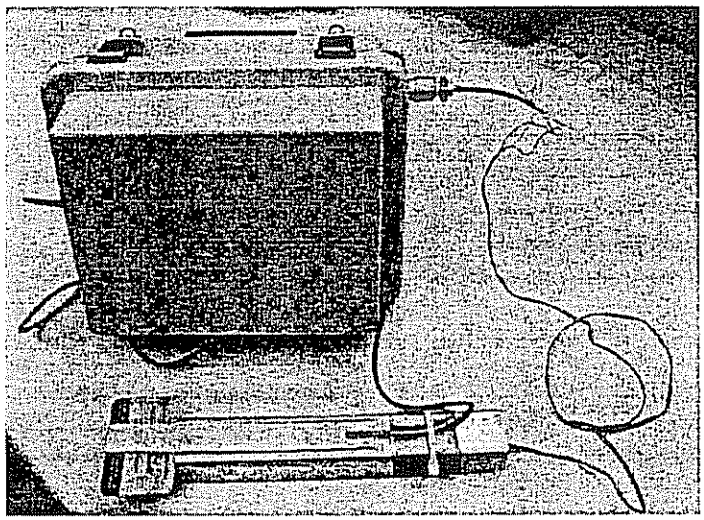


Figure 3.4 Strain Bar Attached to ISBMS

3.3.1.2 Test Results

The error in strain as recorded by the ISBMS is generally less than 1%, with the greatest error just under 3%. The tests showed satisfactory agreement between the P-3500 reading and the ISBMS reading. The results of the test are shown in Table 3.1; a typical plot of the strain comparison is shown in Figure 3.5. If the P-3500 Strain Indicator were used as the absolute measurement of strain, the ISBMS calibration factor for the foil gage could be adjusted so that the two systems closely matched each other. This would result in more accurate results.

3.3.2 Full Bridge Strain Transducer Accuracy Test

3.3.2.1 Test Setup

The accuracy of the second generation ISBMS and the full bridge transducer was tested using a hollow tube (Figure 3.6) in tension.

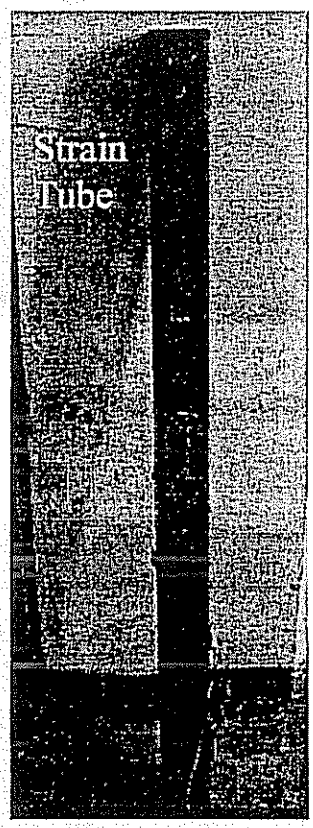


Figure 3.6 Hollow Closed Tube

The tube was instrumented with foil strain gages on each side, as shown in Figure 3.7. The full bridge transducer was then placed over the foil gage on one side and attached to the tube with two C-clamps, as shown in Figure 3.8.

The tube was then placed in a Tinius-Olsen testing machine and tested in tension. The foil strain gages on the tube were recorded using the System 5000 data acquisition system, while the full bridge transducer was recorded using the ISBMS. The test setup is shown in Figure 3.9.

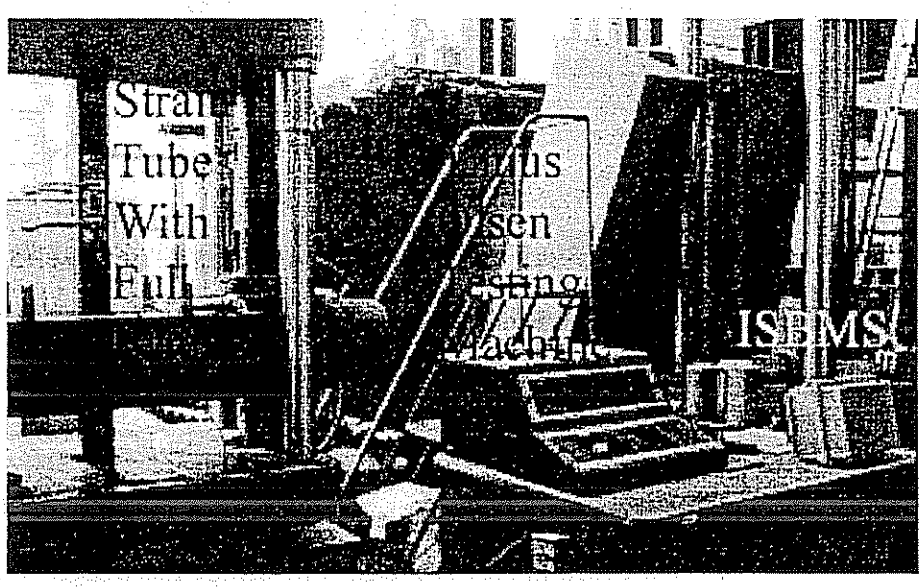


Figure 3.9 Full Bridge Transducer Test Setup

3.3.2.2 Test Results

The results for the tests are shown in Table 3.2. The percent error is somewhat high, but this may be due to noise affecting the foil gages or slight bending of the tube.

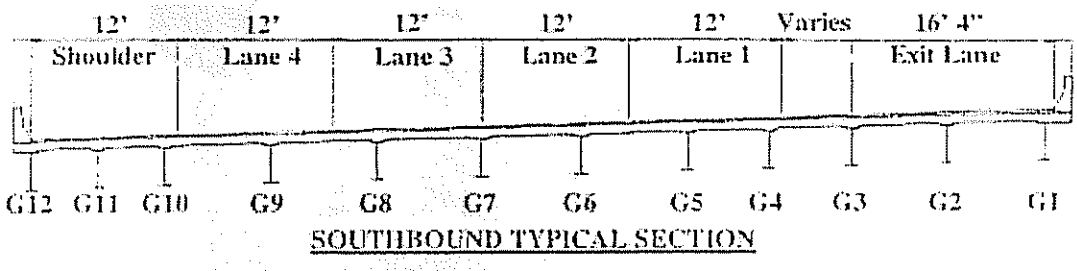


Figure 3.10 Girder Layout for Southbound Approach Span of Bridge 1-704

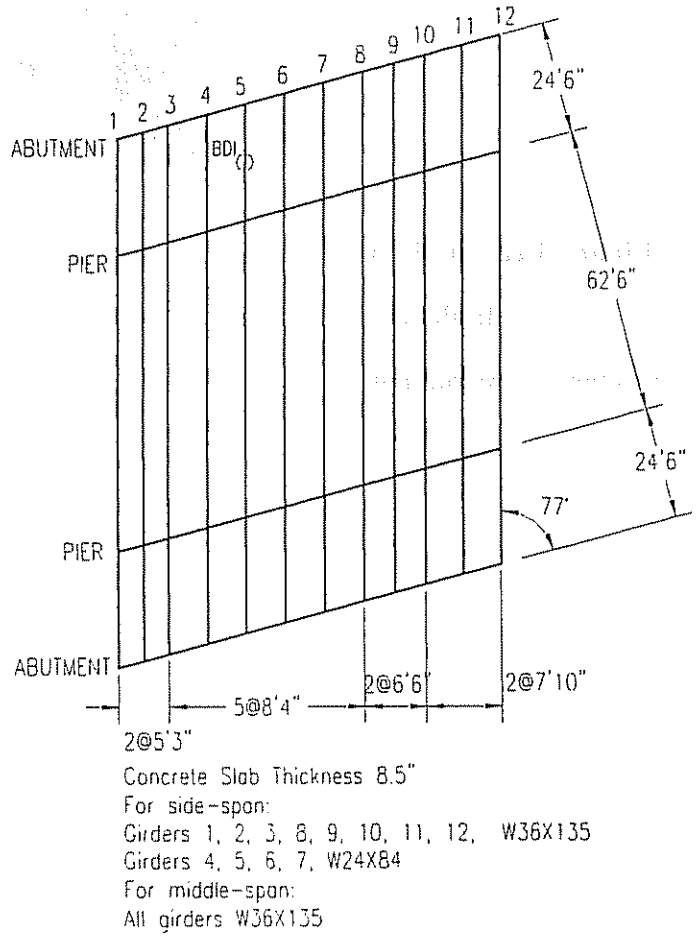


Figure 3.11 Plan View of Bridge 1-704 Showing Location of Full Bridge Transducer on Girder G5

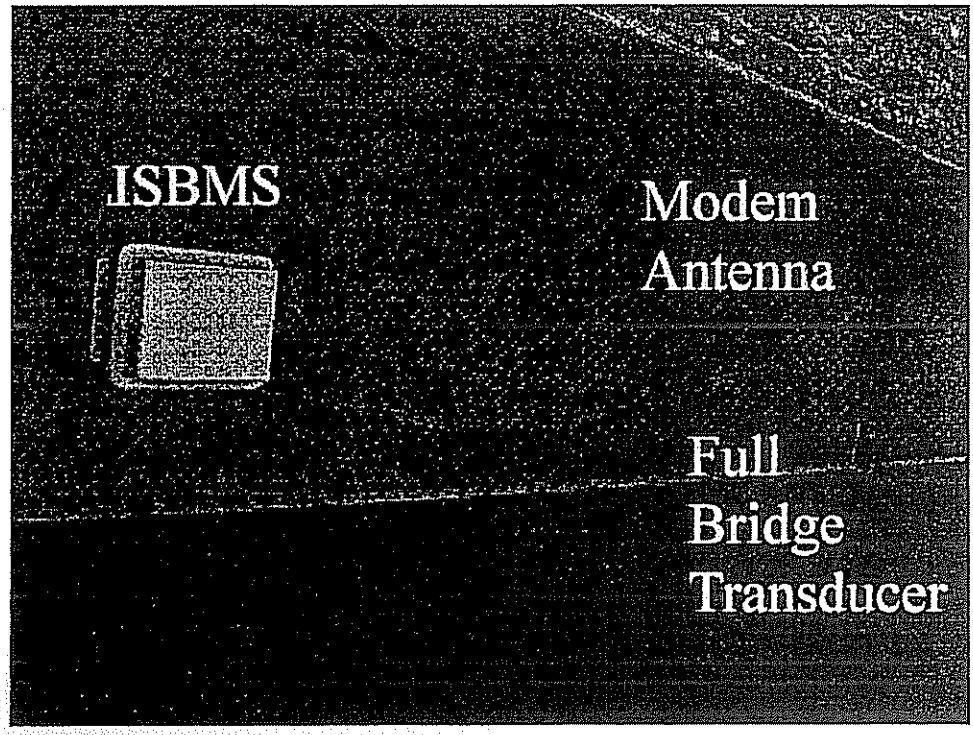


Figure 3.13 Typical Field Setup of ISBMS

3.4.1 Test Setup

The full bridge transducer was positioned at the mid-span location for testing the three sub-programs. For each series of tests, the overall response of the bridge was measured.

3.4.2 Peak Program

The peak program was set to capture data at a sample rate of 50 samples per second with a 1.0-second time window. The gage was mounted at mid-span to capture the overall response of the bridge due to large live loads. While compressive strains would be expected on the bottom flange of the simply supported girder, the compressive strain threshold was set high so that any compressive strains would not be

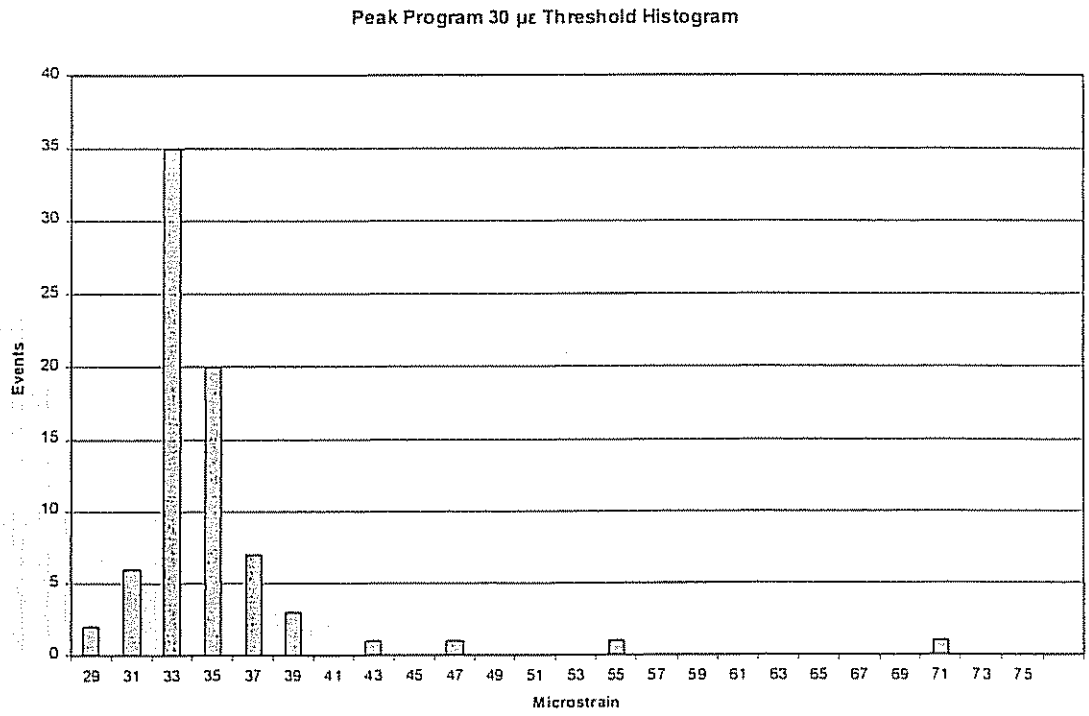


Figure 3.15 Histogram for 30 με Threshold

Due to the large number of events, the tensile strain threshold was reset to 50 με. The test stored 40 events over a 27 minute period, with a maximum tensile strain of 85 με. The results for this test are shown in Figure 3.16; a histogram of the results is shown in Figure 3.17.

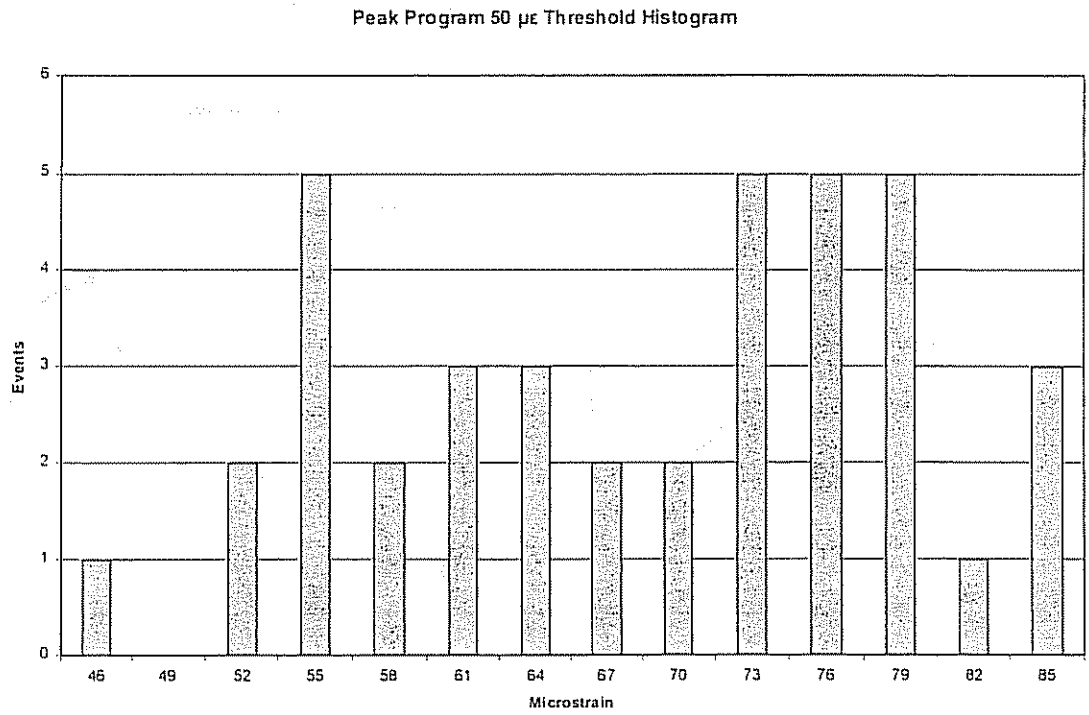


Figure 3.17 Histogram for 50 με Threshold

3.4.3 Time History Program

The sample rate for the time history program was also set to 50 samples per second; the program was set to capture 12 pre-trigger data points for 0.25 seconds and 25 post-trigger data points for 0.50 seconds, bringing the total time window for capturing the event to 0.75 seconds. The pre and post-trigger events are shown in Figure 3.18 from a typical event.

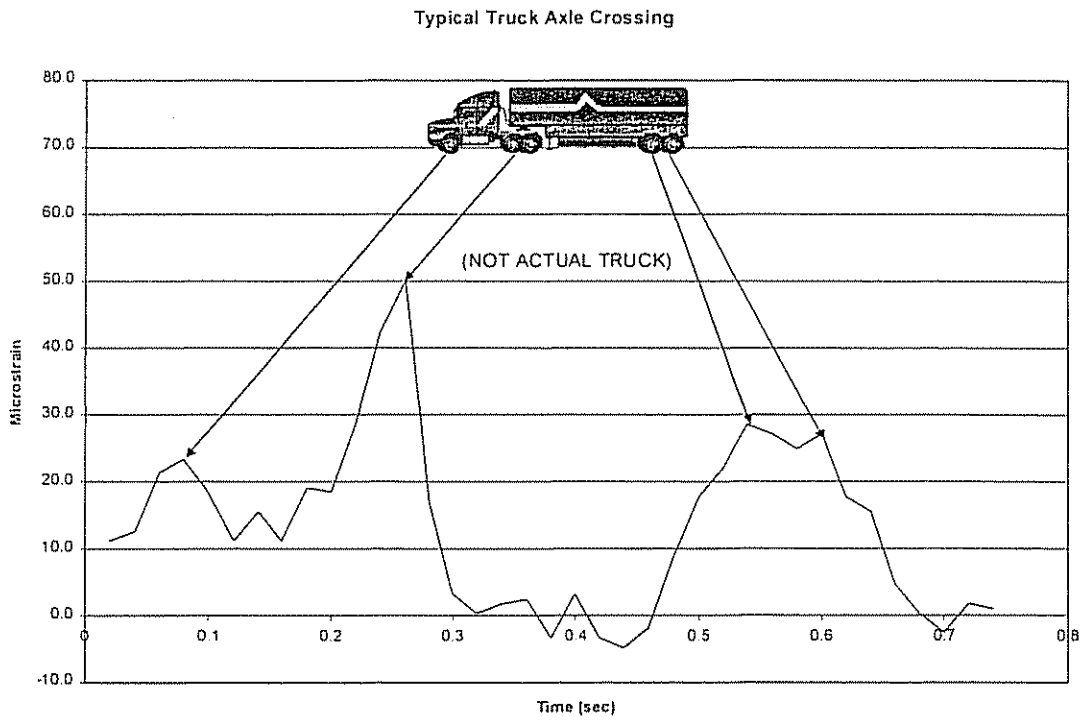


Figure 3.19 Typical Truck Axle Crossing

The test revealed the weight distribution for various trucks; some trucks exhibited a large strain from the rear axles, while others showed the largest strains from the front axles. The test also shows the relative length between axles.

A second set of tests were performed for the time history program with a sample rate of 100 samples per second and longer pre and post-trigger time windows. The pre-trigger time window was increased from 0.25 to 0.5 seconds, resulting in 50 stored points. The post-trigger time was increased from 0.75 to 2.0 seconds, resulting in 200 stored points. Plots displaying several events are shown in Figure 3.20; results clearly define the peak axle crossings over the approach span.

3.4.4 Rainflow Program

The rainflow program was setup much like that of the previous two programs. The sample rate was set to 50 samples per second; the tensile strain threshold was set to 50 με while the compressive strain threshold was set to 30 με. The results shown below represent a time window of 22 minutes. The resulting cycle counts are shown in Figure 3.21.

Table 3.3 Rainflow Program Cycle Counts and Ranges

Range (με)	Cycle Count
41	1
45	1
47	1.5
49	1
50	1
51	2
56	0.5
65	0.5

Chapter 4

SUMMARY AND CONCLUSIONS

A second-generation in-service bridge monitoring system has been developed that takes advantage of the latest advances in remote telemetry. The system measures live load strains due to site-specific traffic. The system is small, rapidly deployable, and is intended to run under its own battery power unattended for up to four weeks. It is based on a single-board-computer with a 12 bit A-to-D converter, analog inputs and digital inputs and outputs. The system has a CDPD modem to communicate over the existing cellular network to a host PC. The system can be used with either a full bridge strain transducer or a quarter bridge foil strain gage. The system contains three sub-programs; the peak program records the peak live load strains, the time history program captures a dynamic waveform of the live load strain, and the rainflow program counts varying amplitude strain cycles to investigate fatigue. A web environment has been produced for the system, allowing bridge owners or maintenance engineers access to strain data from the web.

The ISBMS is housed in a NEMA 4X enclosure that protects against rain, sleet, snow, windblown dust, splashing water, hose-directed water, corrosion, and the formation of ice on the enclosure. Four industrial strength magnets, two 65 lb and two 120 lb capacity, are attached to the bottom of the enclosure for mounting the system to a typical steel bridge girder. Power for the system is provided by two rechargeable batteries. A 12-volt, 12 Amp hour, sealed lead-acid battery provides power to the TFX-11 single-board-computer and the Raven II CDPD modem. A 2-volt, 6 Amp

the full bridge was 0.5 με. The foil strain gage, on the other hand, which is more susceptible to noise because it is only a quarter bridge sensor, showed standard deviations that ranged from a low of 1.6 to a high of 9.4 με. The foil strain gage was also tested with the ISBMS attached to a labtop computer and the deviation in strain increased to 23.1 με. This sudden increase was attributed to a grounding problem with the labtop.

In addition to the noise testing, the accuracy of both gages were tested. The foil strain gage was tested using a strain bar, in conjunction with a P-3500 Strain Indicator. Tests revealed an strain error of less than 1% with the greatest error just under 3%. The tests showed satisfactory agreement between the P-3500 and the ISBMS. Next, the full bridge transducer was tested using a hollow tube placed in tension. Strains from the ISBMS were compared to the System 5000 data acquisition system and the relative error was slightly higher than expected. This may be attributed to noise affecting the foil gages or slight bending of the tube

After laboratory tests, the system was tested in the field on bridge 1-704, a heavily traveled steel girder bridge carrying southbound I-95 traffic through northern Delaware. The peak program was tested with a sample rate of 50 samples per second; a maximum tensile strain of 85 με was recorder over a period of 27 minutes. The time history program was tested at 50 and 100 samples per second. The recorded waveforms clearly show the axles of the truck as they travel across the bridge. A maximum tensile strain of 79 με was recorded. Finally, the rainflow program was tested at 50 samples per second and recorded several cycle ranges, the largest of which was 65 με.

REFERENCES

Alampalli, S., and Fu, G. (1994). "Instrumentation for Remote and Continuous Monitoring of Structure Conditions." *Transportation Research Record*, 1432, 59-67.

Bernard, K. J., Culmo, M. P., and DeWolf, J. T. (1997). "Strain Monitoring to Evaluate Steel Bridge Connections." *Building to Last: Proceedings of Structures Congress XV*, 2, 919-923.

Chajes, M. J., Shenton, H. W., and Finch, W. W. (2001). "Diagnostic and In-Service Testing of a Transit Railway Bridge." *Transportation Research Record*, 1770.

Fu, G., Moosa, A. G., and Peng, J. (1999). "Noncontact Nondestructive Testing for Structure Health Monitoring." In R. R. Avent and M. Alawady (Eds.). *Structural Engineering in the 21st Century: Proceedings of the 1999 Structures Congress*, 340-343.

Harris, C. A. (2002). "Structural Health Monitors Go Wireless." *Civil Engineering Magazine*, ASCE, 72(8), 19.

Holloway, E. S. (1999). "A Long-Term Monitoring System for Highway Bridges." Masters Thesis, University of Delaware, Newark, DE.

Kirkpatrick, T. C., Peterson, D. O., Rossi, P. J., Ray, L. R., Livingston, R. (1999). "A Preliminary Study to Facilitate Smart Structures Systems in Bridge Girders." In S. C. Liu (Eds.). *Smart Structures and Materials: Proceedings of The International Society for Optical Engineering 1999 Conference*, 152-160.

Li, D. (2003). "Application of Load and Resistance Factor Reading Using Site Specific Data." Ph.D. Dissertation, University of Delaware, Newark, DE.

Miller, T. C. (2000). "The Rehabilitation of Steel Bridge Girders Using Advanced Composite Materials." Masters Thesis, University of Delaware, Newark, DE.

Sartor, R. R., Culmo, M. P., and DeWolf, J. T. (1999). "Short-Term Strain Monitoring of Bridge Structures." *Journal of Bridge Engineering*, ASCE, 4(3), 157-164.

Appendix A

GLOSSARY OF TFBASIC FUNCTIONS USED, WEB AND DOS-BASED FILES

11. STORE [#n], [exp]

The STORE command places bytes, words, double words or strings into the next free section of the SFLASH. The formats for n=1,2, or 4 specify whether the data is to be stored as one, two, or four bytes. The exp portion of the STORE command assigns what value is to be stored in the SFLASH.

```
//


//
IF biganswer = 2
        GOTO modems
ENDIF
//
// A '3' response from the main menu will send the user to the gage configuration
// section
//
IF biganswer = 3
        GOTO gages
ENDIF
//
// A '4' response from the main menu will send the user to the programs section
//
IF biganswer = 4
        GOTO programs
ENDIF
//
// The 'timedate' label is used at the start of the time/date section.  Once at the
// 'timedate' label, the program will print the current time and date
//
timedate:
//
// The 'rtime' function below reads the local software real time clock, so that each
// time through this section will result in an updated time and date
//
rtime
//
// The date and time are stored in the variables ?(2), ?(1), ?(0), ?(4), ?(3), ?(5).
// The hour corresponds to variable ?(2), the minute to variable ?(1), the second
// to variable ?(0), the month to variable ?(4), the day to variable ?(3), and the
// year to variable ?(5)
//
PRINT "The time is now: "
PRINT  ?(2),":",?(1),":",?(0);
PRINT " on ", ?(4),"/",?(3),"/",?(5)
//
//
// The program then asks the user if the information printed to the screen is correct
// A '1' response simply skip this section, while a '2' response will require the user
// to enter the present time and date
//
INPUT "If this is correct, press 1, to change press 2 now " chtime
IF chtime = 2
        INPUT "The year is (1980 to 2040) " ?(5)
        INPUT "the month is (1 to 12) " ?(4)
```

```
        PRINT "Currently, powering the modem will start at ", stoffload
        PRINT " hundred hours(military time), for ", endoffload, " minutes "
        INPUT "To change this scenario, press 2 now. To maintain it press 1 now "      offloadtime
//

//
// If a '2' response is sent, the program will ask the user to enter the hour to start
// powering the modem (in 24 hour time) followed by the time window (in minutes)
// to keep the modem on.  The user is then sent back to the main menu
//
        IF offloadtime = 2
                PRINT "Now enter what hour you would like to start offloading data "
                INPUT "Note: enter 1:00 pm as 13 ie, use military time " stoffload
                PRINT "Now enter for how many minutes you would like to leave
        this"
                INPUT "window for download open for ie, 10, 20, etc " endoffload
        ELSE
        ENDIF
ENDIF
stime
setrtc
GOTO bigmenu
//
// The 'gages' label is used at the start of the gage configuration section
//
gages:
//
// Once at the 'gages' label, the program will print that there are two gages
// available for use, and the user must select:
// 1) for the full bridge strain transducer, or
// 2) for the foil strain gage
// After entering the appropriate gage, the user will be prompted to enter the
// calibration factor for the gage.
// Typical calibration factors for the full bridge strain transducer are 100-120 ms/mv
// while typical calibration factors for the foil gage are 1.99-2.03 ms/mv
// The user is then sent back to the main menu
//
PRINT "The two gages available for this application are: "
PRINT " 1) full bridge strain transducer "
PRINT " 2) foil strain gage "
INPUT "Enter now which gage you will be using for your application " curgage
IF curgage = 1
        PRINT "Now enter the BDI Calibration Factor in the following manner: "
        INPUT "microstrain/millivolt (typical value range 100-120 ms/mv) " bdicf!
ELSE
        PRINT "Now enter the Foil Gage Calibration Factor in the following   manner:"
        INPUT "microstrain/millivolt (typical value range 1.99-2.03 ms/mv) " fgcf!
ENDIF
GOTO bigmenu
```

```
// in microstrain.  The program then prints the value that was entered and
// asks the user if this is correct.  A '1' response means yes, while a '2'
// response means no, and sends the user back to the initial question.
// This series of questioning is used throughout the remainder of the code,
// and it is helpful if a errant number is entered by accident
//

//
urange:
INPUT "Enter the upper trigger range in microstrains " utrig
PRINT "Your upper trigger is ", utrig
INPUT "Keep this value(y=1/n=2)? " uanswer
IF  uanswer = 2
        GOTO urange
ENDIF
//
// The program then stores the gage and calibration factor to the data file.  The
// variable 'curgage' equals '1' for the full bridge transducer and equals '2' for
// the foil strain gage. The variable 'bdicf!' is used for the calibration factor for
// the full bridge transducer while the variable 'fgcf!' is used as the calibration
// factor for the foil strain gage
//
IF curgage = 1
        STORE "Gage=BDI", 0
        STORE "Gage Calibration Factor = ", 0, bdicf!
ELSE
        STORE "Gage=Foil", 0
        STORE "Gage Calibration Factor = ", 0, fgcf!
ENDIF
//
// The program then stores the upper trigger range to the data file
//
STORE "UTrigger= ", 0, #2, utrig
//
// The program next prints that the user must enter the lower trigger
// in microstrain.  Again, the program will reproduce the answer to the
// question and ask the user if it is correct
//
lrange:
INPUT "Enter the lower trigger range in microstrains " ltrig
PRINT "Your lower trigger is ", ltrig
INPUT "Keep this value(y=1/n=2)? " lanswer
IF lanswer = 2
        GOTO lrange
ENDIF
//
// If the information is correct, the program then stores the lower trigger value to the
// data file
//
```

```
//
// The variable 'npoints' is used later in the program as the number of points
// to sort through if a trigger value has been exceeded.
// The program will ask the user if the information is correct.  If not, it will loop
// back so the user has the option of entering the time window a second time
//
npoints=AINT(FLOAT(timestp)*(FLOAT(tpoints)/100))
fakvar!=FLOAT(tpoints)/100
//
//
PRINT "You will record ", npoints , " points for ",fakvar!, " seconds "
INPUT "keep this value(y=1/n=2)? " ranswer
IF ranswer=2
        GOTO ratestep
ENDIF
//
// Once the user has agreed upon the time window, it is stored to the data file
//
STORE "Time=", 0, fakvar!
//
// Once all of the header information is complete, the program will print the
// header information to the screen (this is used more for the web-based program)
//
// Print the program we are in
//
Print progchoice
//
// Print the current time/date
//
PRINT  ?(2),?(1),?(0),?(4),?(3),?(5)
//
// Print the gage and calibration factor
//
PRINT curgage
IF curgage =1
        PRINT bdicf!
ELSE
        PRINT fgcf!
ENDIF
//
// Print the offloading time if there is one. If the modem is on continuously, print '99'
//
if modmode=1
        PRINT "99"
ELSE
        PRINT stoffload
        PRINT endoffload
ENDIF
//
```

```
ELSE
        utrigger = AINT(FLOAT(utrig)/(.07778*fgcf!)) + abszero
        ltrigger = abszero - AINT(FLOAT(ltrig)/(.07778*fgcf!))
ENDIF
//
```

```
//
// The flag 'staymod' is used when the modem is powered on.  For example, if the
// user intends to leave the modem on for 30 minutes to change certain parameters
// and it only takes 7 minutes to change them, the user has the option of turning
// off the modem after the last set of changes are complete, thus saving power.
// If the option to power the modem off after the last change is sent, then the flag
// 'staymod' will equal '2' and the command 'PCLR(0)' will power down the modem
//
IF staymod = 2
        PCLR(0)
        sleep 200
ENDIF
//
// Two arrays are used to find the peak strain values
// The 'larray' array constantly monitors strain values, and is overwritten
// continuously; the counter for this array is 'e'
// The 'sort' array is only used during the sorting process, once a trigger is
// exceeded; the counter for this array is 'f'
//
DIM larray(400)   // counter is 'e'
DIM sort(200)     // counter is 'f'
prestart1:
//
// Throughout the code, a series of flags are used to direct the program to certain
// areas.  The label 'prestart1' is used to reset many of the program's flags
//
flair = 0
absflair = 0
pthresh = 0
vthresh = 0
f = 1
prestart2:
//
// If the counter for the large array ever gets to 399, then the program will reset
// it equal to zero from the 'prestart2' label
//
e = 0
//
```

```
        v=chan(2)
ENDIF
//
// The variable 'v' is assigned the strain value from the appropriate gage.
// This variable is then assigned a value in the 'larray' array, shown below
//
larray(e) = v
//
// The flag 'absflair' is equal to '1' if either the upper or lower thresholds have
// been exceeded previously.  The system will NOT compare the current
// strain to either of the two triggers if the flag 'absflair' equals '1'
//

//
IF absflair = 1
        GOTO jump
ENDIF
//
// If the 'absflair' flag has not been tripped, the program compares the current
// strain first to the lower trigger value, denoted by the variable 'ltrigger' It then
// compares the current strain to the upper trigger value, denoted by the
// variable 'utrigger'
//
IF v <= ltrigger
        STORE #1, ?(2), #1, ?(1), #1, ?(0), #1, ?(4), #1, ?(3), #2, ?(5)
//
// If the lower trigger is exceeded, the system stores the time and
// date that the trigger was exceeded.
//
// The flag 'pthresh' is equal to '1' if an upper trigger has been exceeded.  If this has
// occurred, the program sets the flag 'vthresh' equal to '0'.  If the upper trigger has
// NOT been exceeded, then the program assigns flag 'vthresh' equal to '1' and the
// flag 'pthresh' equal to '0'
//
        IF pthresh = 1
                vthresh = 0
        ELSE
                vthresh = 1
                pthresh = 0
        ENDIF
//
// The first time the lower trigger has been exceeded, the current strain
// value denoted by variable 'v' is assigned as the upper trigger.  The flag 'absflair'
// is then set to '1' indicating that a trigger has been exceeded.
//
        absflair = 1
        v = utrigger
ENDIF
//
```

```
// For example, if the counter 'e' equals 193, and the program is sorting through 75
// points, then the variable 'tempe' would equal 193, and the variable 'target' would
// equal 193 + 75 = 268
//
        tempe = e
        target = tempe + npoints
ENDIF
//
// The counter 'e' is incremented by 1 regardless if a trigger was encountered
// or the program is simply sorting data to be sorted.  As mentioned previously, if
// the counter 'e' equals 399, the program is directed to the label 'prestart2'
// After updating the counter 'e', the program returns to the label 'start'
//


//
e = e + 1
IF e = 399
        GOTO prestart2
ENDIF
GOTO start
//
// If the program has encountered either an upper or lower trigger, it will be
// directed, through the flags 'absflair' and 'flair' to the 'skip' label below
//
skip:
//
// The program will continue to gather data to be sorted, as long as the
// counter 'tempe' is less than the counter 'target'
//
IF tempe < target
//
// If the counter 'tempe' is less than the counter 'target' the current strain
// values are assigned from the array 'larray' to the array 'sort'.  The counter
// for the 'sort' array is 'f'
//
        sort(f) = larray(e)
//
// Once the current strain values are assigned to the sorting array, the counters
// 'tempe', 'f', and 'e' are incremented and the program returns to the 'start' label
//
        f = f + 1
        tempe = tempe + 1
        e = e + 1
//
// For every increment of the counter 'e', the IF statement below must follow, to
// prevent the program from incrementing 'larray' more than 400 data points
//
        IF e = 399
```

```
// the global maximum is found.  The maximum is then stored to the data file.
//
        max = sort(g)
        FOR m = 2 to npoints
                IF sort(g-1) >= sort(g)
                        max = sort(g-1)
                ENDIF
        g = g - 1
        NEXT m
        STORE #2, max
        PRINT "peak is ", max
ENDIF
//
// After the maximum or minimum is stored, the program is directed to the label
// 'prestart1', which resets the many flags needed for operation
//

//
GOTO prestart1
//
////////////////////////////////////
// Time History Program //
////////////////////////////////////
//
phistprog:
//
// Once the program reaches the 'phistprog' label, the program will store the time
// and date, and the program currently entered (time history)
//
rtime
STORE "Time=", 0, #1, ?(2), #1, ?(1), #1, ?(0)
STORE "on", 0, #1, ?(4), #1, ?(3), #2, ?(5)
STORE "Program=History", 0
//
// The program then enters into a series of questions and answers for the
// header information regarding the gage being used, any trigger information, etc.
// This information is stored in a header located at the top of each data file.
//
// The program first prints that the user must enter the upper trigger range
// in microstrain.  The program then prints the value that was entered and
// asks the user if this is correct.  A '1' response means yes, while a '2'
// response means no, and will send the user back to the initial question.
//
urange2:
INPUT "Enter the upper trigger range in microstrains " utrig2
PRINT "Your upper trigger is ", utrig2
INPUT "Keep this value(y=1/n=2)? " uanswer2
IF uanswer2 = 2
        GOTO urange2
```

```
ENDIF
//
// After entering the values, the program correlates this to the sleep command.
// The 'sleep(x)' command sends the TFX-11 into a precise timing sequence with
// x being entered in 1/100th of a second.  A response of 'sleep 100' would cause a 1.0
// second delay.  Therefore, if the sample rate is for 100 samples/second, this
// corresponds to a 'sleep 1'.  A sample rate of 50 samples/second corresponds to
// a 'sleep 2', and a sample rate of 25 samples/second corresponds to a 'sleep 4'
//
IF timestp2>99
        slptime2=1
ELSE
        If timestp2>49
                slptime2=2
        ELSE
                slptime2=4
        ENDIF
ENDIF
//
//
// The sample rate is then stored to the data file
//
STORE "Sample_rate=", 0, #2, timestp2
//
// The next section deals with the time window for capturing events.
// The user is asked to enter the time window in 1/100ths of a second
// Again, a response of 100 would equal a 1 second time window
//
ratestep2:
PRINT "Enter the pre-trigger length of time you would like to capture events, "
INPUT "in 1/100th of a second " stpoints
PRINT "Enter the post-trigger length of time you would like to capture events, "
INPUT "in 1/100th of a second " ctpoints
//
// After entering the sample rate and the time window, the program will calculate
// the number of pre and post-trigger points to be sorted through and the
// time window  for the pre and post trigger events
// The variable 'spoints' is used later in the program as the number of pre-trigger
// points to sort through if a trigger value has been exceeded.
// The variable 'cpoints' is used later in the program as the number of post-trigger
// points to sort through if a trigger value has been exceeded.
//
spoints=AINT(FLOAT(timestp2)*(FLOAT(stpoints)/100))
cpoints=AINT(FLOAT(timestp2)*(FLOAT(ctpoints)/100))
sfakvar!=FLOAT(stpoints)/100
cfakvar!=FLOAT(ctpoints)/100
//
// The program will ask the user if the information is correct.  If not, it will loop
// back so the user has the option of entering the time windows a second time
```

```
PRINT spoints
PRINT sfakvar!
PRINT cpoints
PRINT cfakvar!
//
sleep 0
//
// The line 'sleep 0' is used as a reference for the sleep command
// As outlined in the TFX-11 manual, the sleep command is actually a countdown
// timer, so for example, if a 'sleep 100' command is issued, the program
// will search for an already running sleep timer from a previous command
// to see if it has expired.  The 'sleep 0' command resets the sleep timer.
//
// Now that the header information has been printed, the system
// will take readings for 10 seconds to zero the gage being used
//




//
test1b = 0
test2b = 0
FOR i = 1 to 1000
        IF curgage = 1
                test1b=chan(4)
        ELSE
                test1b=chan(2)
        ENDIF
        test2b = test2b + test1b
        sleep 1
NEXT i
abszero2 = AINT(FLOAT(test2b/1000))
zero2! = ((FLOAT(abszero2/16))/4096)*5.0
//
// The program will then print the relative zero point first as a voltage,
// then as a reading from the 'chan(x)' command—a 5 digit integer, and store the
// zero reading in the data file
//
PRINT "our absolute zero point is at a voltage of ", zero2!
PRINT "and a strain of ", abszero2
STORE "Zero_Point=", 0, #2, abszero2
//
// The upper and lower trigger ranges use the 10 second zero strain reading as
// a reference, shown below
//
IF curgage=1
        utrigger2 = AINT(FLOAT(utrig2)/(.0003889*bdicf!)) + abszero2
```

```
// For example, if the program encounters a trigger value, it will usually loop back
// to the 'start2' label
//
start2:
//
// In order to check if the time to turn the modem on has been reached, it will be
// checked each time we loop through the starting labels
// If the time for offloading has been reached, the program will automatically
// set Pin 0 to a logic high (set it to +5 Volts) therefore engaging the solid state
// relay, and thus powering the modem. The program is then directed to the
// 'olwindow' label, at the end of this code, where the program will loop until
// the user defined time window for powering the modem on has been reached.
// After this time limit has passed, the program is directed back to whichever of the
// three sub-programs it was previously in. This is accomplished by the 'resume2'
// label, shown below. In this case, it would loop back to the time history program.
//

//
rtime
IF ?(2) = stoffload
        IF ?(1) < endoffload
                prog = 2
                PCLR(0)
                PSET(0)
                PRINT "we have set Pin 0 to high state "
                GOTO olwindow
        ENDIF
ENDIF
resume2:
//
// The 'sleep slptime2' is used as the sample rate mechanism. Recall the user was
// required to enter the sample rate as 25, 50, or 100 samples/second. The 'slptime2'
// variable contains the sample rate mentioned previously
//
sleep slptime2
//
// The program identifies which gage to read based on the 'curgage' variable
// If curgage = 1, then the full bridge strain transducer has been selected as the current
// gage, while if curgage = 2, the foil strain gage has been selected
//
IF curgage = 1
        v2=chan(4)
ELSE
```

```
//
IF flair2 = 1
        GOTO continu
ENDIF
//
// If the current strain value has not exceeded the lower trigger, then the system
// will investigate if it has exceeded the upper trigger.  If the current strain is above
// the upper threshold value, or the lower trigger has been exceeded, the program
// will enter the following sequence.
//
IF v2 >= utrigger2
//
// If the program has encountered a lower trigger, or the actual strain is above
// the upper trigger, the program will enter this sequence
//
        IF absflair2 = 1
                v2 = fakev
//
// If the flag 'absflair2' equals '1', the program has previously encountered a
// lower trigger. The program then assigns the variable 'v2' the value of 'fakev'.
// In essence the program has tricked the system into seeing a upper trigger value,
// but once it sees the 'absflair2' flag, it will change the current strain value, 'v2' to
// the previous value, denoted by the variable 'fakev'
//

//
        ELSE
//
// If the flag 'absflair2' equals '0', then the program has not encountered a lower
// trigger so it stores the time and date the UPPER trigger has been encountered
//
                Store #1, ?(2), #1, ?(1), #1, ?(0), #1, ?(4), #1, ?(3), #2, ?(5)
        ENDIF
//
// After storing the date, the program will set the flags 'flair2' and 'absflair2' equal
// to '1', in order to skip the comparison of strain values to any triggers
// The counter 'p' is assigned the value of the counter 't' from the large array
// 'sortarray'
//
        p = t
        flair2 = 1
        absflair2 = 1
//
// The program seeks to capture the pre-trigger data in an efficient manner.
// By using the array 'sortarray', the pre-trigger data is always available for storing.
// One area of concern, from a programming point of view, is if the counter 't'
// for the 'sortarray' is less than the number of pre-trigger data points.  For example,
// if the counter for the 'sortarray', 't' has recently reset, and is at data point
// number 22 (t = 22), and the program attempts to store 40 pre-trigger points, then the
```

```
                    r = r + 1
                next m
//
// Once the data points at the end of the array have been stored, the program
// will store the remaining data points, if any, at the beginning of the array.
// The counter 'n' is set to zero, just at the counter 't' is when the program starts
// in order to keep an accurate count.  For the example, the counter 'p' and the
// counter 't' are the same value, so the program will store t = p = 12 data points
//
                FOR n = 0 to p-1
                    ievent(r) = sortarray(n)
                    store #2, ievent(r)
                    PRINT ievent(r)
                    r = r + 1
                next n
        ENDIF
//
// The variable 'tempt' is used for the post-trigger storing sequence in conjunction
// with the 'targett' variable
//
tempt = t - 1
targett = tempt + cpoints
GOTO continu
ENDIF
//
// The counter 't' is incremented by one integer, regardless if a trigger was
// encountered or the program is simply storing data.
//
t = t + 1
//
// If the counter 't' gets to 999, the program will be sent to the 'prestart1b' label
//
IF t = 999
        GOTO prestart1b
ENDIF
//
// Once the program has reached this point, it is sent back to the 'start2' label
//
GOTO start2
//
// The 'continu' label is used to simply store the post-trigger data points
//
continu:
//
// As long as the counter 'tempt' is less than 'targett', the program will store
// the elements of 'sortarray'
//
IF tempt < targett
        ievent(r) = sortarray(t)
```

```
PRINT "Your upper trigger is ", unoise
INPUT "Keep this value(y=1/n=2)? " uanswer3
IF  uanswer3 = 2
        GOTO nurange
ENDIF
//
// The program then stores the gage and calibration factor to the data file.  The
// variable 'curgage' equals '1' for the full bridge transducer and equals '2' for
// the foil strain gage.  The variable 'bdicf!' is used as the calibration factor for
// the full bridge transducer while the variable 'fgcf!' is used as the calibration
// factor for the foil strain gage
//
If curgage = 1
        STORE "Gage=BDI", 0
        STORE "Calibration_factor=", 0, bdicf!
ELSE
        STORE "Gage=Foil", 0
        STORE "Calibration_factor=", 0, fgcf!

ENDIF
//
// The program then stores the upper trigger range to the data file
//
STORE "UNoise= ", 0, #2, unoise
//
// The program then prints that the user must enter the lower trigger
// in microstrain.  Again, the program will reproduce the answer to the
// question and ask the user if it is correct
//
//
nlrange:
INPUT "Enter the lower trigger range in microstrains " lnoise
PRINT "Your lower trigger is ", lnoise
INPUT "Keep this value(y=1/n=2)? " lanswer3
IF lanswer3 = 2
        GOTO nlrange
ENDIF
//
// If the information is correct, the program then stores the lower trigger value to the
// data file
//
STORE "LNoise= ", 0, #2, lnoise
//
// The program then prompts the user to enter the strain resolution in microstrain.
// Again, the program will reproduce the answer to the question and ask the user
// if it is correct
//
resolution:
INPUT "Enter the strain resolution in microstrain " res
```

```
check1 = 0
check2 = 0
sleep 0
//
// The line 'sleep 0' is used as a reference for the sleep command
// As outlined in the TFX-11 manual, the sleep command is actually a countdown
// timer, so for example, if a 'sleep 100' command is issued, the program
// will search for an already running sleep timer from a previous command
// to see if it has expired.  The 'sleep 0' command resets the sleep timer.
//
FOR i = 1 to 1000
        IF curgage = 1
                check1=chan(4)
        ELSE
                check1=chan(2)
        ENDIF
        check2 = check2 + check1
        sleep 1
NEXT i
relzero = AINT(FLOAT(check2/1000))
azero! = ((FLOAT(relzero/16))/4096)*5.0
//
// The program will then print the relative zero point first as a voltage,
// then as a reading from the 'chan(x)' command—a 5 digit integer, and store the
// zero reading in the data file
//
PRINT "our absolute zero point is at a voltage of ", azero!
PRINT "and at a strain of ", relzero
STORE "Zero_point=", 0, #2, relzero
//

//
// The upper, lower, and resolution strains ranges use the 10 second zero strain
// reading as a reference, as shown below.
//
IF curgage = 1
        uchattrig = relzero + aint((unoise/(.00038889*bdicf!)))  //upper noise trigger in microstrain
        lchattrig = relzero - aint((lnoise/(.00038889*bdicf!))) // lower noise trigger in microstrain
        localtrig = aint((res/(.00038889*bdicf!)))  // strain resolution
ELSE
        uchattrig = relzero + aint((unoise/(.07778*fgcf!)))     //upper noise trigger in microstrain
        lchattrig = relzero - aint((lnoise/(.07778*fgcf!)))  //lower noise trigger in microstrain
        localtrig = aint((res/(.07778*fgcf!)))     //strain resolution
ENDIF
//
// The flag 'staymod' is used when the modem is powered on.  For example, if the
// user intends to leave the modem on for 30 minutes to change certain parameters
// and it only takes 7 minutes to change them, the user has the option of turning
// off the modem after the last set of changes are complete, thus saving power.
```

```
//
x1 = 0
x2 = 0
y1 = 0
y2 = 0
x1trig = 0
y1trig = 0
j = 0
k = 0
//
// The 'begin' label is used for ordinary operation, and is referenced later on in the
// code
//
begin:
//
// In order to check if the time to turn the modem on has been reached, it will be
// checked each time we loop through the starting labels
// If the time for offloading has been reached, the program will automatically
// set Pin 0 to a logic high (set it to +5 Volts) therefore engaging the solid state
// relay, and thus powering the modem.  The program is then directed to the
// 'olwindow' label, at the end of this code, where the program will loop until
// the user defined time window for powering the modem on has been reached.
// After this time limit has passed, the program is directed back to whichever of the
// three sub-programs it was previously in.  This is accomplished by the 'resume3'
// label, shown below.  In this case, it would loop back to the rainflow program
//
```

```
//
rtime
IF ?(2) = stoffload
        IF ?(1) < endoffload
                prog = 3
                PCLR(0)
                PSET(0)
                PRINT "we have set Pin 0 to high state "
                GOTO olwindow
        ENDIF
ENDIF
resume3:
//
////////////////////////////////
// Running Average  //
////////////////////////////////
```

```
// 'psort' equal to '1'  and enters the 'psorter' lable, which finds a peak strain value
//
IF c <= lchattrig
        vsort = 1
        GOTO vsorter
ELSE
        psort = 1
ENDIF
//
//////////////////////
// Peak Sorter //
//////////////////////
//
psorter:
//
// The first time the program enters the peak sorter, the flag 'x1trig' is set to '0'
// and the program assigns the variables 'x1' and 'x2' the current strain values.
// These variables are used later as the moving average calculation for the rainflow
// program. The current strain value is also assigned a max value and the flag
// 'x1trig' is set equal to '1' so that the variables 'x1' and 'x2' are only assigned the
// FIRST time the program enters the peak sorter routine.  After assigning the
// variables, the program is sent to the 'begin' label.
//
IF x1trig = 1
        GOTO continue1
ELSE
        x1 = c
        x2 = c
        max1 = c
        x1trig = 1
        GOTO begin
ENDIF
//
// Once the variables 'x1' and 'x2' are set, the label 'continue1' is used to divert
// the program from assigning the variables a second time
//
continue1:
//
// The flag 'vpos' is set to '1' if the program has previously encountered
// a valley that is BELOW the LOWER trigger.  For this case, the peak value to
// be sorted MAY be a local value (below the lower trigger), fall between the noise
// triggers, or exceed the UPPER trigger.  If the strain exceeds the UPPER trigger,
// the program will set the flag 'vneg' below to '1' and be directed to the 'negvalley'
// label.
//
IF vpos = 1
        GOTO posvalley
ENDIF
//
```

```
        max1 = c
//
// Once the current strain has exceeded the resolution value, the variables 'x1' and
// 'x2' are reset
//
        x1 = max1 + localtrig
        x2 = max1
ELSE
//
// If the current strain value is ABOVE the LOWER trigger, then the program
// assigns the flag 'smalldeal' equal to '1', assigns the current strain value
// to the variable 'x3'.  The strain is now in the 'noise zone'
//
        IF c > lchattrig
                smalldeal = 1
                x3 = c
                GOTO begin
        ENDIF
ENDIF
//
// The program then loops to the 'begin' label
//
GOTO begin
//
// The label 'negvalley' is used to sort a peak value that is ABOVE the UPPER
// trigger; this is the simplest sorting algorithm for a peak strain and corresponds
// to 'vpos' equal to '1'
//
negvalley:
//
// If the current strain value is greater than the resolution, it is assigned the variable
// 'max1' and the current resolution value is incremented
//
IF c > x2
        max1= c
        x2 = c
ENDIF
//
// If the current strain value is greater than the 'x1' + resolution value, then
// it is assigned the max value.  The variables 'x1' and 'x2' are updated, and the
// program loops to the 'begin' label
//
IF c > x1 + localtrig
        max1=c
        x1= max1 + localtrig
        x2= max1
        GOTO begin
ELSE
//
```

```
// The first scenario is investigated here.  If the strain is still above the lower trigger,
// then the strain is still in the 'noise zone'
//
IF c > lchattrig
//
// If the strain is still in the 'noise zone', then the program investigates if it is larger
// than the resolution trigger 'localtrig' If it is, then the current strain value is assigned
// the maximum value, 'max1'
//
        IF c > x3 + localtrig
                max1 = c
                x3 = c
        ENDIF
//
// The second scenario, if the strain exceeds the UPPER trigger, is investigated here.
// If the strain is above the upper trigger, then it is out of the 'noise zone' and the
// program sets new values for the variables 'x1' and 'x2'.  In addition, the program
// sets the flag 'vneg' equal to '1' which will direct the program into finding a peak
// that is ABOVE the UPPER trigger.  Lastly, the flag 'smalldeal' is set to '0'
//
        IF c > uchattrig
                x1 = c
                x2 = c
                vpos = 0
                vneg = 1
                smalldeal = 0
        ENDIF
//
ELSE
//
// If the strain is not above the lower trigger, then it is below it, and a peak value
// must have occurred while in the 'noise zone'.  In this scenario, the peak value
// is stored, the flag 'vsort' is set to '1'—implying that the next value to be found
// is a valley
//




//
        IF c < lchattrig
                peak(h) = max1
                PRINT "our peak is ", max1
                vsort = 1
                psort = 0
//
// The flags 'x1trig' and 'x2trig' are set to '0', the flag 'pneg' is set to '1' to
// show that the peak value was found BELOW the UPPER trigger
//
```

```
//
// Once the variables 'y1' and 'y2' are set, the label 'continue2' is used to divert
// the program from assigning the variables a second time
//
continue2:
//
// The flag 'pneg' is set to '1' if the program has previously encountered
// a local peak.  For this case, the valley value will be BELOW the LOWER trigger.
// This is the simplest of the sorting algorithms.
//
IF pneg = 1
        GOTO negpeak
ENDIF
//
// The label 'negpeak' is used to sort a valley value BELOW the LOWER trigger.
//
// The flag 'ppos' is set to '1' if the program has previously encountered a peak
// that is ABOVE the UPPER trigger.  For this case, the valley
// value MAY be local, fall into the 'noise zone', or fall below the lower trigger
//
IF ppos = 1
        GOTO pospeak
ENDIF
//
// The label 'pospeak' is used to find where the valley value will end up—either
// local value (above the upper trigger), fall between the noise triggers, or
// exceed the LOWER trigger
//
pospeak:
//
// In this algorithm, we are looking for a valley that is initially ABOVE the UPPER
// trigger. This value MAY be a local valley, or it may enter the noise zone.
//
// The flag 'bigdeal' is used with the label 'bgdeal' if the current strain enters the
// 'noise zone'
//
IF bigdeal = 1
        GOTO bgdeal
ENDIF
//

//
// If the current strain is above the upper trigger and less than the variable 'y2'
// then it is assigned the minimum value, 'min1'
//
IF c > uchattrig & c < y2
        min1 = c
ENDIF
//
```

```
// If the strain is above the variable 'y1' then a minimum has occurred, and it
// is stored. The flag 'psort' is set to '1', implying the next value will be a peak
// value. The flags 'x1trig' and 'y1trig' are set to '0'
//
        IF c > y1
                valley(i) = min1
                PRINT "our valley is ", min1
                psort = 1
                vsort = 0
                x1trig = 0
                y1trig = 0
//
// The flag 'vpos' is set to '1' implying the valley stored was NOT a local value.
// The flag 'bigdea' is set to '0', and the counter for the valley sequence is
// incremented by one integer.
//
                vpos = 1
                vneg = 0
                bigdeal = 0
                i = i + 1
//
// If the sum of the counters for the peak and valley sorters is equal to '3', then the
// program has 2 ranges that can be compared
//
                IF i+h = 3
                        GOTO vach
                ENDIF
                GOTO begin
        ENDIF
ENDIF
//
GOTO begin
//
// The label 'bgdeal' is used when the strain falls into the 'noise zone'
//
bgdeal:
//
// If the strain is below the upper trigger, then it is still in the 'noise zone'
//
IF c < uchattrig
//


//
// In this case, the program checks to see if the current strain is below the strain
// resolution, 'localtrig'. If it is, the current strain value is assigned the minimum
// value, 'min1' and the variable 'y3' is updated
//
        IF c < y3 - localtrig
```

```
// If the sum of the valley and peak counters equals '3' then the program has
// two ranges to compare, and the program is directed to the 'vach' label
//
                IF i+h = 3
                        GOTO vach
                ENDIF
        ENDIF
ENDIF
GOTO begin
//
vach:
//
// If the program has 3 points, it must consist of either 2 peaks and 1 valley or
// 1 peak and 2 valleys
//
PRINT "the value of h + i is ", c2
//
// For the 2 peaks, 1 valley sequence, the counter 'i' equals '1' and the counter
// 'h' equals '2'.  This is the peak-valley-peak sequence
//
IF i=1 & h=2
//
// The first time through the sequence, the flag 'trigger' is set to '0'
// It is used to show if the event "range x is less than y" has occurred
// previously.  The first time through the sequence, this is ignored.
//
        If trigger = 1
//
// If the flag 'trigger' equals '1', then the event "range x is less than y" HAS
// occurred and the variable 's' is assigned the variable 'stemp'.  The variable
// 's2' is assigned the value of 's3' This sequence is used to keep the starting
// point 's' at the beginning of the comparison.  If the ranges are successfully
// smaller, then the starting point will always be retained, until the event
// x > y occurs.  Refer to the ASTM documentation for further information
// on this point.
//
                s = stemp
                s2 = s3
        ELSE

//
// If the event "range x is less than y" has NOT occurred, the program then assigns
// the first peak (peak(0)) to the variable 's'; it will become the starting point
// 's' for the beginning of the comparison
//
//
                s = peak(0)
        ENDIF
//
```

```
// 'stemp' and the first valley to the variable 's3'.
//
                        key = 2
                        stemp = peak(0)
                        s3 = valley(0)
//
// The second peak (peak(1)) is then assigned the value of the first peak.  The flag
// 'trigger' is set to '1', indicating that the event "range x is less than y" has
// indeed occurred.  The program will then loop to the beginning to find a valley.
// It will therefore enter the second sequence with 1 PEAK/2 VALLEYS
//
                        peak(0) = peak(1)
                        trigger = 1
                ENDIF
//
                h = 1
                i = 1
                GOTO prebegin
        ELSE
//
// If the range x is not less than y, the program enters this sequence.  Again,
// the flag 'trigger' is checked to see if its value is '1'
//
                IF trigger = 1
//
// If the flag 'trigger' is equal to '1', the program looks toward a second flag, 'key'.
//
// If the flag 'key' is equal to '1', then the program will overwrite the current
// variable 's2' with the current peak, peak(1).  This will create a situation where
// the starting point 's' is a valley, the second point, 's2' is now a  peak, so then
// the program would then need a valley to complete the three points
// needed to make a comparison.  This is accomplished by setting the peak counter
// 'h' equal to '1'.  The program will store the current range and then find a second
// valley value and be sent to the 1 PEAKS/2 VALLEY sorter
//
                        IF key = 1
                                s2 = peak(1)
                                h = 1
                        ELSE
//
// If the flag 'key' is not equal to '1', then the first point, 's' is a peak.  The second
// point, 's2' is a valley, and the third point will be peak(1), and the program will
// loop once again through this sequence to make a comparison.
//
                                h = 2
                        ENDIF
                        i = 1
//
// The current range y is stored as 1.0 cycle
```

```
// For the 1 peak, 2 valley sequence, the counter 'i' equals '2' and the counter
// 'h' equals '1'.  This is the valley-peak-valley sequence.
//
IF h=1 & i=2
//
// The first time through the sequence, the flag 'trigger' is set to '0'
// The 'trigger' flag is used to show the event "range x is less than y" has occurred
// previously.  The first time through the sequence, this is ignored.
//
        IF trigger = 1
//
// If the flag 'trigger' equals '1', then the event "range x is less than y" has
// occurred and the variable 's' is assigned the variable 'stemp'.  The variable
// 's2' is assigned the value of 's3' This sequence is used to keep the starting
// point 's' at the beginning of the comparison.  If the ranges are successfully
// smaller, then the starting point will always be retained, until the event
// x > y occurs. Refer to the ASTM documentation for further information
// on this point.
//
                s = stemp
                s2 = s3
        ELSE
//
// If the event "range x is less than y" has NOT occurred, the program then assigns
// the first valley (valley(0)) to the variable 's'. It will become the starting point 's'
// at the beginning of the comparison.
//
                s = valley(0)
        ENDIF
//
// If the flag 'trigger' equals '2', then the event  "range x is less than y" has
// occurred, but it has not occurred with the starting point 's'.  The program then
// compares the current peak/valley sequence with the starting point 's'
//
        IF trigger = 2
//
// The range y now begins at starting point 's' followed by the next point denoted
// by the variable 's2'
//
                s = stemp
                y = s2 – s
//
// The range x begins at the second point 's2' and ends at the second peak, valley(1)
//
                x = s2 - valley(1)
//
// The array values valley(0) and peak(0) are updated so they coincide with the
// starting point 's' and 's3'
//
```

```
//
//
// If the range x is not less than y, then the program enters this sequence.  Again, the
// flag 'trigger' is checked to see if it equals '1'
//
                    IF trigger = 1
//
// If the flag 'trigger' is indeed equal to '1', the program looks toward a second
// flag, 'key'
//
// If the flag 'key' is equal to '2', then the program will overwrite the current
// variable 's2' with the current valley, valley(1).  This will create a situation where
// the first starting point 's' is a peak, the second point, 's2' is now a valley, so the
// program would then need a peak to complete the three points needed to make
// a comparison.  This is accomplished by setting the counter 'i' for the valley array
// equal to '1'.  The program will store the current range and then find a second
// peak and enter the 2 PEAKS/1 VALLEY sorter
//
                              IF key = 2
                                        s2 = valley(1)
                                        i = 1
                              ELSE
//
// If the flag 'key' is equal to '2', then the first point 's' is a valley.  The second
// point, 's2' is a peak, and the third point will be valley(1).  The program will
// loop once again through this sequence to make a comparison.
//
                              i = 2
//
// The current range y is stored as 1.0 cycle
//
                              ENDIF
                              h = 1
                              range(j) = y
                              store #2, range(j)
                              PRINT "trigger equaled 1 previously, y = ", y
                              cycle(k) = 1.0
                              store #2, cycle(k)
                              PRINT cycle(k)
//
// The counters 'j' and 'k are incremented by one integer
//
                              j = j + 1
                              k = k + 1
//
// The flag 'trigger' is set to '2' indicating that the next comparison of ranges will
// include the first points 's' and 's2'.  Depending on the value of the flag 'key',
// the program will either find a third point and be sent to the first sorting sequence,
// or it will simply loop back into the current sequence and do the comparison there.
```

```
//
// The variable '?(1)' is the minute variable that is assigned to the variable
// 'endoffload'. For example, if the time window is set for 14 minutes, the variable
// 'endoffload' will equal 14. The program continually loops until the local time
// variable '?(1)' equals 14. Once this occurs, the program will power down the
// modem and be directed to the program previously loaded by the 'progcontin'
// label
//
IF ?(1) = endoffload
//
// Once the local time clock has reached the minute for leaving the modem on, the
// program will power the modem off using the variable '?(0)'. This time variable
// corresponds to the seconds, so that the modem is powered off in the first second
// of the minute. For example, at 14 minutes and 1 second past the hour, the modem
// will be powered off. If the variable '?(0)' is not used, the program would attempt
// to power down the modem 60 times until the next minute (15) has occured
//
        IF ?(0) = 0
                sleep 0
                PCLR(0)
                PRINT "we have set Pin 1 to low state "
                sleep 200
                GOTO progcontin
        ENDIF
ELSE
//
// If the minute for offloading has not been reached, the program will simply loop
// back to the label 'olwindow'
//
                GOTO olwindow
ENDIF
//
// The label 'modemoff' is in place if the user intends to change system parameters
// and power the modem off after they have been changed. Otherwise, the modem
// would stay on for extra time, using extra battery power that is not needed. For
// example, if the user intends to leave the modem on for 25 minutes and it only
// takes 3 minutes to download the datafile, and no changes to the program are
// required, then the user has the option to issue a 'Ctrl-C' character and simply
// power down the modem after the three minutes
//
modemoff:
//
// The flag 'staymod' is used to power down the modem in each of the three
// sub-programs. If it is set to '2', then the modem will power down once the user
// has entered a particular program
//
```

```
        GOTO resume2
ENDIF
IF prog = 3
        GOTO resume3
ENDIF
Web-Based File
//
// MASTER PROGRAM
//
bigmenu:
CBREAK
INPUT " " biganswer
IF biganswer = 1
        GOTO timedate
ENDIF
IF biganswer = 2
        GOTO modems
ENDIF
IF biganswer = 3
        GOTO gages
ENDIF
IF biganswer = 4
        GOTO programs
ENDIF
timedate:
rtime
PRINT "The time is now: "
PRINT  ?(2),":",?(1),":",?(0);
PRINT " on ", ?(4),"/",?(3),"/",?(5)
INPUT " " chtime
IF chtime = 2
         INPUT " " ?(5)
         INPUT " " ?(4)
         INPUT " " ?(3)
         INPUT " " ?(2)
         INPUT " " ?(1)
         INPUT " " ?(0)
ELSE
ENDIF
stime
setrtc
GOTO bigmenu
modems:
INPUT " " modmode
IF modmode = 1
         PCLR(0)
         PSET(0)
ELSE
         INPUT " " offloadtime
```

```
timestep:
INPUT " " timestp
IF timestp>99
        slptime=1
ELSE
        if timestp>49
                slptime=2
        ELSE
                slptime=4
        ENDIF
ENDIF
STORE "Sample_Rate= ", 0, #2, timestp
ratestep:
INPUT " " tpoints
npoints=AINT(FLOAT(timestp)*(FLOAT(tpoints)/100))
fakvar!=FLOAT(tpoints)/100
STORE "Time=", 0, fakvar!
//print the program we are in
Print progchoice
//print the current time/date
PRINT  ?(2),?(1),?(0),?(4),?(3),?(5)
//print the gage and calibration factor
PRINT curgage
IF curgage =1
        PRINT bdicf!
ELSE
        PRINT fgcf!
ENDIF
//print the offloading time if there is one
if modmode=1
        PRINT "99"
ELSE
        PRINT stoffload
        PRINT endoffload
ENDIF
//print upper/lower trigger
PRINT utrig
PRINT ltrig
PRINT timestp
PRINT npoints
PRINT fakvar!
sleep 0
```

```
                GOTO olwindow
        ENDIF
ENDIF
resume1:
sleep slptime



IF curgage = 1
        v=chan(4)
ELSE
        v=chan(2)
ENDIF
larray(e) = v
IF absflair = 1
        GOTO jump
ENDIF
IF v <= ltrigger
        IF pthresh = 1
                vthresh = 0
        ELSE
                vthresh = 1
                pthresh = 0
        ENDIF
        absflair = 1
        v = utrigger
ENDIF
jump:
IF flair = 1
        GOTO skip
ENDIF
IF v >= utrigger
        IF vthresh = 1
                pthresh = 0
        ELSE
                pthresh = 1
                vthresh = 0
        ENDIF
        flair = 1
        absflair = 1
        tempe = e
        target = tempe + npoints
ENDIF
e = e + 1
IF e = 399
        GOTO prestart2
ENDIF
GOTO start
```

```
        STORE "Gage=BDI", 0
        STORE "Gage Calibration Factor = ", 0, bdicf!
ELSE
        STORE "Gage=Foil", 0
        STORE "Gage Calibration Factor = ", 0, fgcf!
ENDIF
STORE "Utrigger=", 0, #2, utrig2
lrange2:
INPUT " " ltrig2
STORE "Ltrigger=", 0, #2, ltrig2

timestep2:
INPUT " " timestp2
STORE "Sample_rate=", 0, #2, timestp2
IF timestp2>99
        slptime2=1
ELSE
        If timestp2>49
                slptime2=2
        ELSE
                slptime2=4
        ENDIF
ENDIF
ratestep2:
INPUT " " stpoints
INPUT " " ctpoints
spoints=AINT(FLOAT(timestp2)*(FLOAT(stpoints)/100))
cpoints=AINT(FLOAT(timestp2)*(FLOAT(ctpoints)/100))
sfakvar!=FLOAT(stpoints)/100
cfakvar!=FLOAT(ctpoints)/100
STORE "Time=", 0, sfakvar!, cfakvar!
//print the program we are in
Print progchoice
//print the current time/date
PRINT  ?(2),?(1),?(0),?(4),?(3),?(5)
//print the gage and calibration factor
PRINT curgage
IF curgage =1
        PRINT bdicf!
ELSE
        PRINT fgcf!
ENDIF
//print the offloading time if there is one
if modmode=1
        PRINT "99"
ELSE
        PRINT stoffload
        PRINT endoffload
ENDIF
```

```
                PCLR(0)
                PSET(0)
                GOTO olwindow
        ENDIF
ENDIF
resume2:
sleep slptime2
IF curgage = 1
        v2=chan(4)
ELSE
        v2=chan(2)
ENDIF
sortarray(t) = v2
IF absflair2 = 1
        GOTO hop
ENDIF
IF v2 <= ltrigger2
        absflair2 = 1
        fakev = v2
        v2 = utrigger2
ENDIF
hop:
IF flair2 = 1
        GOTO continu
ENDIF
IF v2 >= utrigger2
        IF absflair2 = 1
                v2 = fakev
        ENDIF
        p = t
        flair2 = 1
        absflair2 = 1
        IF p-cpoints > 0
                n = cpoints
                FOR m = 1 to cpoints
                        ievent(r) = sortarray(p-n)
                        store #2, ievent(r)
                        n = n - 1
                        r = r + 1
                next m
         ELSE
                k = 0
                FOR m = 1 to p
                        ievent(r)= sortarray(k)
                        store #2, ievent(r)
                        k = k + 1
                        r = r + 1
                next m
         ENDIF
```

```
        PRINT fgcf!
ENDIF
//print the offloading time if there is one
if modmode=1
        PRINT "99"
ELSE
        PRINT stoffload
        PRINT endoffload
ENDIF
check1 = 0
check2 = 0
sleep 0



FOR i = 1 to 1000
        IF curgage = 1
                check1=chan(4)
        ELSE
                check1=chan(2)
        ENDIF
        check2 = check2 + check1
        sleep 1
NEXT i
relzero = AINT(FLOAT(check2/1000))
azero! = ((FLOAT(relzero/16))/4096)*5.0
STORE "Zero_point=", 0, #2, relzero
trigger = 0
If curgage = 1
        chattrig = 2571
        localtrig = 130
ELSE
        chattrig = 643
        localtrig = 32
ENDIF
STORE "Local_trigger=", 0, #2, localtrig
STORE "Chatter_trigger=", 0, #2, chattrig
psort = 1
vsort = 0
crash:
h = 0
i = 0
prebegin:
x1 = 0
x2 = 0
x1trig = 0
x2trig = 0
j = 0
k = 0
```

```
                vsort = 1
                psort = 0
                x1trig = 0
                h = h + 1
                GOTO vach
        ENDIF
ENDIF
GOTO begin
vsorter:
IF c >= relzero - chattrig & v <= relzero + chattrig
        GOTO begin
ENDIF



IF x2trig = 1
        GOTO continue2
ELSE
        x2 = c
        x2trig = 1
ENDIF
continue2:
IF c < x2 - localtrig
        x2 = c
ELSE
        IF c <= x2
                min1 = c
        ELSE
                valley(i) = min1
                psort = 1
                vsort = 0
                x2trig = 0
                i = i + 1
                GOTO vach
        ENDIF
ENDIF
GOTO begin
vach:
c2 = h + i
IF i+h < 3
        GOTO prebegin
ENDIF
IF i=1 & h=2 //one valley, two peaks
        If trigger = 1
                s = stemp
                s2 = s3
        ELSE
```

```
                ELSE
                        range(j) = y
                        store #2, range(j)
                        cycle(k) = 0.5
                        store #2, cycle(k)
                        peak(0) = peak(1)
                        j = j + 1
                        k = k + 1
                        i = 1
                        h = 1
                        trigger = 0
                ENDIF
        ENDIF
GOTO prebegin
ENDIF



IF h=1 & i=2 //one peak, two valleys
        IF trigger = 1
                s = stemp
                s2 = s3
        ELSE
                s = valley(0)
        ENDIF
        IF trigger = 2
                s = stemp
                y = s2 - s
                x = s2 - valley(1)
                valley(0) = stemp
                peak(0) = s2
        ELSE
                y = peak(0)-valley(0)
                x = peak(0)-valley(1)
        ENDIF
        IF x < y
                IF trigger = 1
                        stemp  = s
                        s2 = s3
                ELSE
                        key = 1
                        stemp = valley(0)
                        s3 = peak(0)
                        trigger = 1
                        valley(0) = valley(1)
                ENDIF
                h = 1
                i= 1
```

```
CBREAK
sleep 0
PCLR(0)
sleep 200
IF prog = 1
        GOTO resume1
ENDIF
IF prog = 2
        GOTO resume2
ENDIF

IF prog = 3
        GOTO resume3
ENDIF
```

B.1 Foil Strain Gage

$$\varepsilon = \frac{\left[(GageFactor)(1000)(Chan(x) - zero)\right]}{\frac{(V_{ext})(2^{16})}{5}} \frac{1000}{relativegain}$$

Where:

ε = Strain

Gage Factor = number entered by user (typical values range from 1.95-2.05) and specified by the gage manufacturer

Chan(x) = TFTools channel command

zero = baseline zero strain reading recorded by the TFX-11when a program is first executed

V_{ext} = Excitation voltage, 2 Volts

$2^{16} = 65536$

relative gain = 490.44

This simplifies to:

$$\varepsilon = (.07778)(GageFactor)(Chan(x) - zero)$$

In order to calculate the trigger values from a user entered strain value to the TFX-11 Chan(x) command, we simply modify the above equation, so that an upper or lower trigger would simply be:

$$uppertriggvalue = \frac{(uppertrigger(microstrain))}{(.07778)(GageFactor)} + zeroreading$$

It holds then, that our new strain is:

$$\varepsilon = 0.005097(GageFactor)[0.07629(Chan(x)\text{-}zero)]$$

$$\varepsilon = 0.00038889(GageFactor)(Chan(x)\text{-}zero)$$

In order to calculate the trigger values from a user entered strain value to the TFX-11 Chan(x) command, we simply modify the above equation, so that an upper or lower trigger would simply be:

$$uppertriggvalue = \frac{(uppertrigger(microstrain))}{(.00038889)(GageFactor)} + zeroreading$$

For a Gage Factor of 116.44, an upper trigger value of 50 με, and a zero reading of 32100,

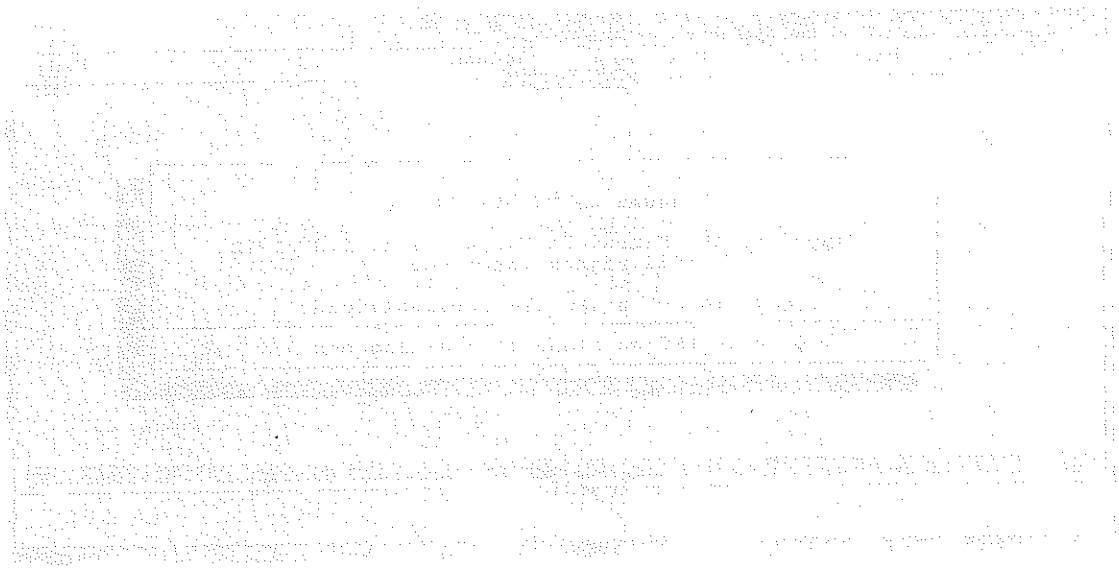
$uppertriggvalue = 33204$

Appendix C

PRINTING SEQUENCE

Appendix D

QBASIC BINARY TO ASCII DATA FILE CONVERSION



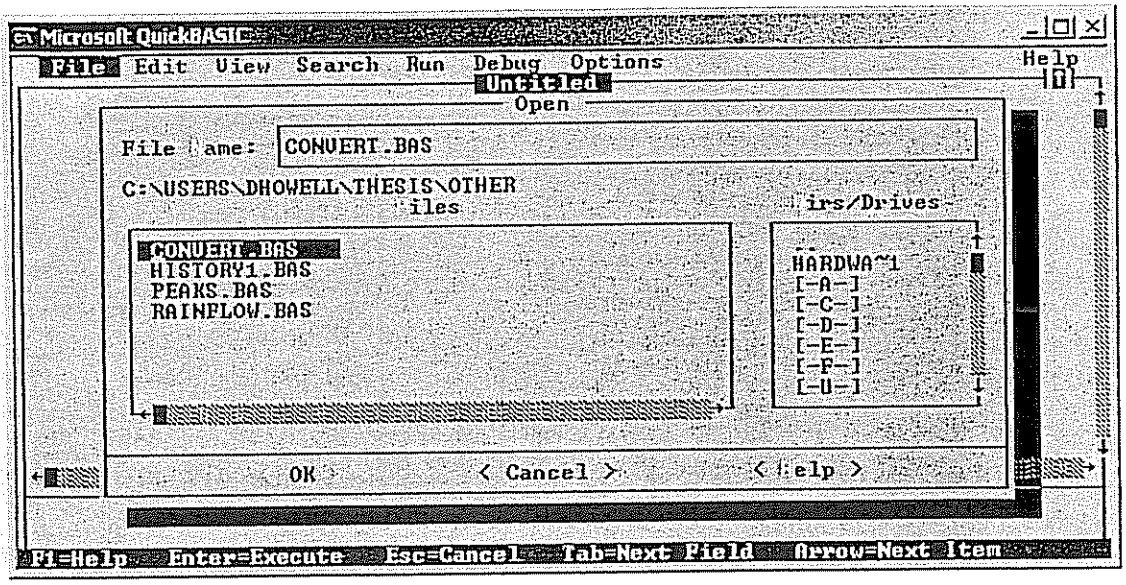


Figure D.2 QBASIC Open Dialog Box

To convert a file that has been downloaded from the TFX-11, it must be in the same directory as the QBASIC working directory. This location is shown above in Figure D.2 as C:\USERS\DHOWELL\THESIS\OTHER. The file to be converted must be in this folder.

The user must select the file PEAKS.BAS to convert the Peak Program's binary file. Once the file has been loaded, the user must modify several parameters in the file itself. The user must scroll through the file until the words 'Main Program' are visible, as shown in Figure D.3. The user must enter the name of the binary file from the TFX-11 shown in Figure D.3 by the line:

CALL OpenFile("pk821a.dat", 1)

This will open the binary file for conversion. The user must also enter the name of the converted ASCII file in the line denoted by:

CALL OpenTextFile("pk821a.Txt", 2)

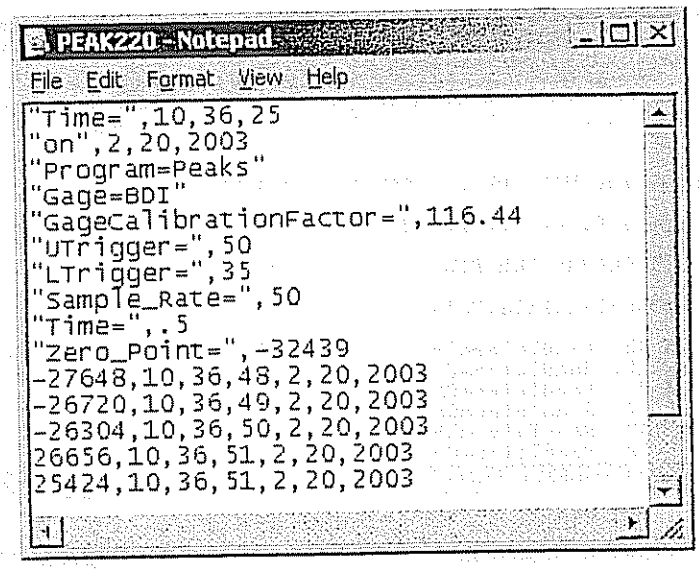


Figure D.4 Typical Peak Program ASCII File

To select the Time History Program, the user must follow the same procedure outlined above. The user must select the HISTORY1.BAS file from the Open Dialog Box. Again, the user must make sure the file from the TFX-11 is in the same folder as the QBASIC program. Once the program is loaded, the user must enter the name of the TFX-11 binary file, shown in Figure D.5, denoted by the line:

CALL OpenFile("ht1127b.dat", 1)

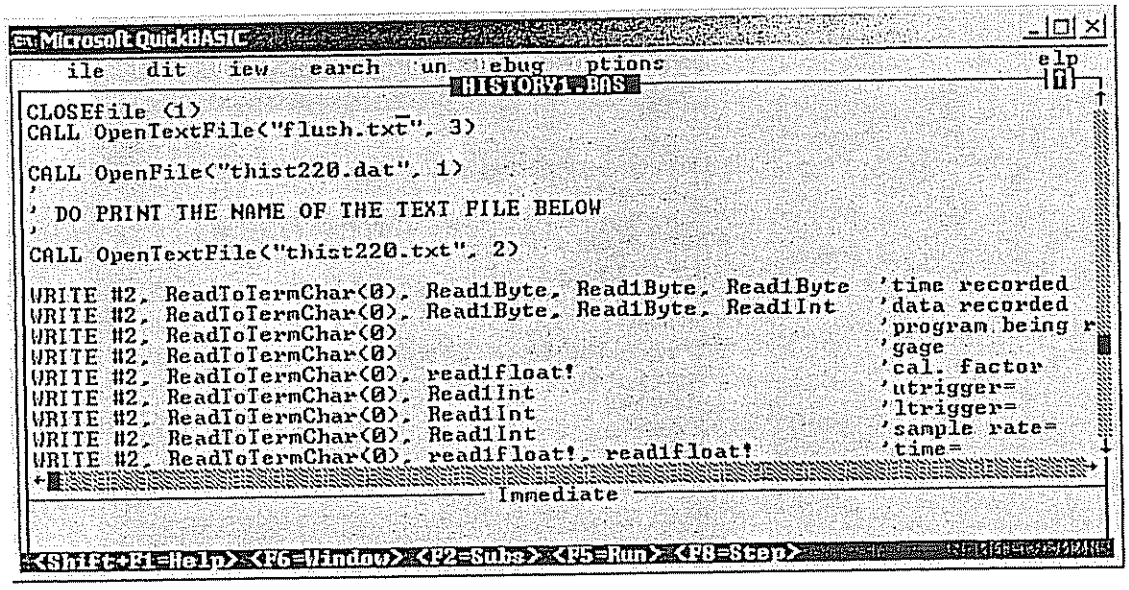


Figure D.6 QBASIC Time History ASCII File Input

Under the line stating 'DO PRINT THE NAME OF THE TEXT FILE BELOW', the user must input the name of the ASCII file to be converted. To run the program, the user must select the *Run* pull-down menu and select *Start*. This will complete the conversion process and result in an ASCII readable file shown in Figure D.7.

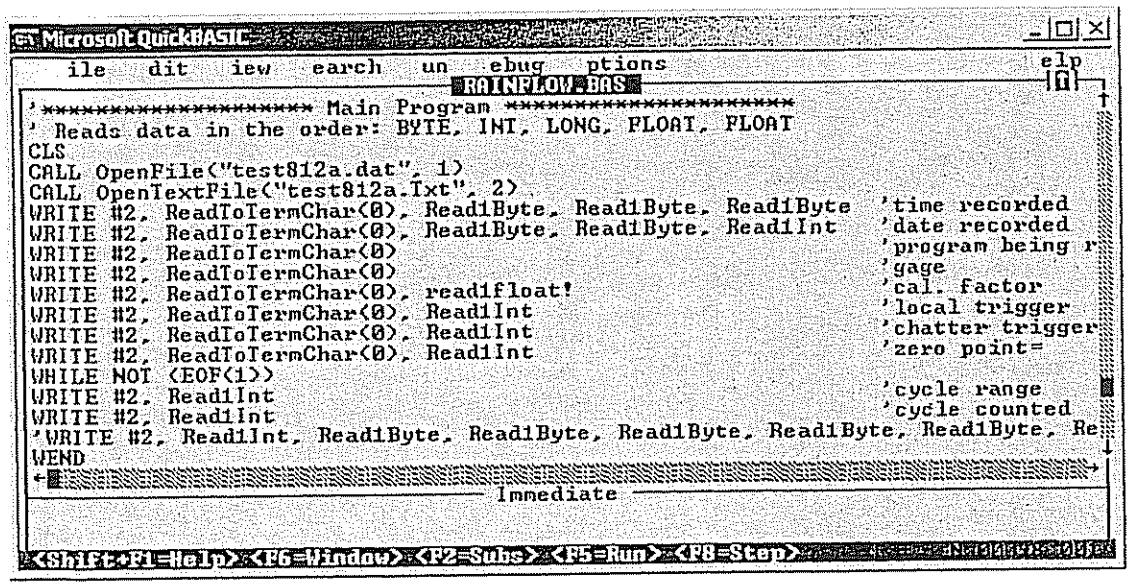


Figure D.8 QBASIC Rainflow Binary/ASCII File Input

Once the file names have been entered, the program is executed by selecting the *Run* pull-down menu and selecting *Start*. An example of the converted ASCII file is shown in Figure D.9.

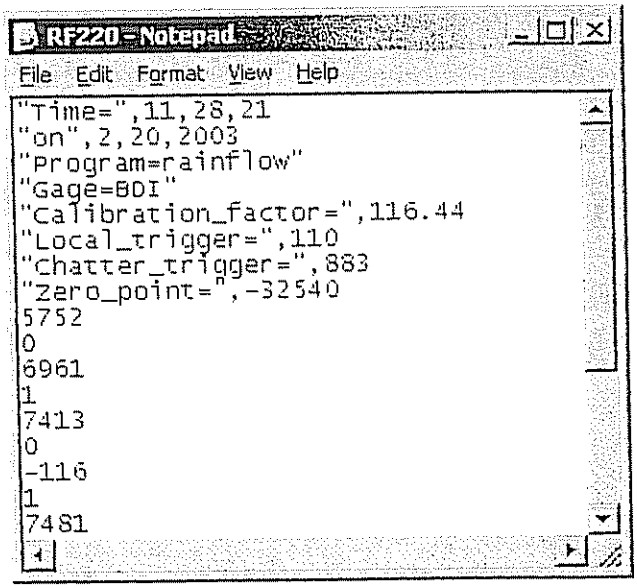


Figure D.9 Typical Rainflow ASCII File

Delaware Center for Transportation
University of Delaware
Newark, Delaware 19716