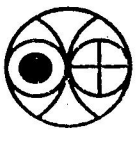
PRL TN-79-2001

PRL Technical Note

Remote Controlling of Tektronix DSO by IBM-PC using GPIB Interface

VISHAL N. SHAH, VINOD NAMBOODIRI,
K.P. SUBRAMANIAN AND A.P. GOHIL

JANUARY 2001



Physical Research Laboratory
Navrangpura, Ahmedabad

PRL Technical Note

Remote Controlling of Tektronix DSO by IBM-PC using GPIB Interface[s]

Vishal N. Shah[†], Vinod Namboodiri[†], K.P. Subramanian and A.P. Gohil
Physical Research Laboratory, Navrangpura, Ahmedabad - 380009

[†]B.E. (Instrumentation and Control) Govt. Engineering College, Gandhinagar, Gujarat.

January 2001

Physical Research Laboratory
Navrangpura, Ahmedabad

Remote Controlling of Tektronix DSO by IBM-PC using GPIB Interface[§]

Vishal N. Shah[†], Vinod Namboodiri[†], K.P. Subramanian and A.P. Gohil
Physical Research Laboratory, Navrangpura, Ahmedabad - 380009

[†]B.E. (Instrumentation and Control), Govt. Engineering College, Gandhinagar, Gujarat.

Abstract:

Using the General Purpose Interface Bus (GPIB), the Tektronix TDS 540 digital storage oscilloscope has been interfaced with a personal computer. A software has been developed for setting the oscilloscope in remote mode. Some of the most commonly-used front panel controls are developed using the graphics features available in C-programming. After the acquisition, useful parameters set in the DSO as well as the waveform data are transferred to the master controller PC.

Key words : GPIB, IEEE 488.2, Tektronix DSO, C-programming

[§]A project undertaken done in PAR-AM group, PRL during 1999-2000.

Acknowledgement

We thank Prof. Vijay Kumar for kindly permitting us to carry out the project work in his laboratory. We are also grateful to him for the interest shown in the progress of the work and extending constant support and encouragement.

Our sincere thanks are due to Dr. V. Sivakumaran for spending a lot of time for explaining the details of instruments and also for defining the project. His expertise in the subject has helped us during the course of the work.

We thank Dr. Prabhakar Sharma, Head, Academic Services for his interest and promotion of the work.

Vishal Shah and Vinod Namboodiri express their thanks to Prof. N.V. Gunchala, Head, Instrumentation & Control, Govt. Engineering College, Gandhinagar for guiding and imparting requisite knowledge for the successful completion of this work.

January, 2001

Vishal Shah, Vinod Namboodiri,
K.P. Subramanian and A.P. Gohil

Preface

Laboratory equipments, either control or measurement devices, when used in stand-alone mode require basic minimum external controls. These controls can be performed by a sequence of operations through the front panel of the instrument. These instruments also are generally been provided with either serial or IEEE ports (or both) for data transfer and for setting up the parameters required for the experiment using a computer.

In complex experiments (or processes), where many such instruments are being used, the front panel control and setting of each equipment becomes extremely cumbersome. Yet in some other cases, where the setting of one instrument depends on the status of another parameter of a module in the cluster, manual operation of the equipments becomes inefficient or even impractical. Remote controlling of equipments is also essential where experiments are done in hazardous environment, such as nuclear reactors or process controls in chemical industry dealing with toxic reactants. These situations warrants interfacing of the all instruments in a cluster using a common bus which allow data flow across the modules in conformation with a standard protocol. The data exchanged through the bus could be instructions (commands), parameters (set values, addresses of modules, etc.), status (checking of a module) and the measured data as well.

In such a network of equipments, it is imperative to have one module acting as the controller of the process. The controller will act as node and will be the In-Charge of the bus. It will also have all the address specifications of the devices in the network. By the address specifications, it controls the data flow in the bus.

National Instruments, USA has come up with controller card, known as General Purpose Interface Bus (GPIB) card, which can be inserted to a personal computer. With proper software loaded in the PC, this card can work as the In-Charge of interfaced laboratory equipments. The equipments that can be added to this network necessarily should conform to IEEE 488 standard. The PC acting as the controller provides necessary support to view the status of the process on its consol and storage of data using user developed programs.

In Laboratory Astrophysics group of PRL, an attempt is initiated to network the laboratory instruments using GPIB protocol. The equipments conform with GPIB protocol are excimer laser (LPX 110E), Tektronix digital storage oscilloscope (TDS 540), UV spectrograph (HR460, ISA-JY Spex), CCD (Spectrum One, Horiba, ISA-JY Spex), Turbo MCS (EG&G, Ortec), laser power meter (RM-6600, Laser Probe) and CAMAC (Model No. 6700, Bira system) and its modules. The interfacing of all these instruments will enhance the efficiency and usefullness of data handling system and also will simplify the complexity of experiments considerably.

This technical note discusses the interfacing of Tektronix TDS 540 DSO to the master controller. The following sections deal with the communication, setting up and data transfer of the DSO under GPIB protocol. The codes are written C programming language, developed under Borland C++ (with MS-DOS 6.2 as OS). The work presented in this report offers future scope to develop software for emulating additional features in data acquisition using DSO and also for achieving remarkable synchronization of the processes down to nanosecond time scales. It would be worth mentioning here that the work described here would lead to the interfacing of instruments in graphic programming language, G, using LabVIEW software.

Ahmedabad
January 8, 2001

Vishal Shah, Vinod Namboodiri,
K.P. Subramanian, A.P. Gohil

Index

List of Figures

List of Tables

1. Introduction

1.1 GPIB Overview

GPIB (General Purpose Interface Bus) describes a standard interface for communication between instruments and controllers from various vendors. It contains information about electrical, mechanical, and functional specifications. The GPIB is a digital, 8-bit parallel communications interface with data transfer rates of 1 Mbytes/s and above. The bus supports one System Controller, usually a computer and up to 32 additional instruments. The ANSI/IEEE Standard 488.2-1987 extends IEEE 488.1 by defining a bus communication protocol, a common set of data codes and formats, and a generic set of common device commands. GPIB devices can be Talkers, Listeners, or Controllers. A Talker sends out data messages and issues commands to the devices. Listeners receive data messages. The Controllers manages the flow of information on the bus. It defines the communication links and sends GPIB commands to devices. Devices are usually connected with a cable assembly consisting of shielded 24-conductor cable with both a plug and receptacle connector at each end. With this design, one can link devices in a linear configuration, a star configuration, or a combination of the two.

To achieve the high data transfer rate, that the GPIB was designed for, one must limit the physical distance between devices (maximum total cable length of 20 meter, average separation of two meters over the entire bus, etc.) and the number of devices on the bus. If one want to exceed these limitations, one can use bus extenders to increase the cable length or expanders to increase the number of device loads.

In this report, we present the details of interfacing laboratory equipment with master controller (IBM-PC with GPIB card) under the GPIB protocol. NI 488.2 driver software for GPIB supports communication languages Basica, Qbasic, C and C++, of which C was the language chosen for developing the source code. The instrument chosen was Tektronix storage oscilloscope (TDS 540). The software developed could simulate most of the front panel control of the oscilloscope such as setting up parameters, acquiring data and transferring to the controller. Basic graphics also were developed to make the software user-friendly.

2. Tektronix TDS 540 Overview

This Digital Storage Oscilloscope (or TEK as hearafter be called) has 500 MHz maximum analog bandwidth, 1 Gigasample/second maximum digitizing rate, four-channel acquisition capacity, eight-bit digitizers, up to 15,000 record length per channel capacity, full GPIB software programmability, complete measurement and documentation capability, on-line help at the touch of a button etc. It consists of various sub-systems. Following 4 are the sub-systems with which we are concerned more.

1. Triggering System

Triggers determine when the digital storage oscilloscope starts acquiring and displaying a waveform. The trigger event establishes the time-zero point in the waveform record. Once a trigger is recognized, the digital storage oscilloscope will not accept another trigger until the acquisition is complete. The basic trigger is the edge trigger.

Basically three trigger sources are available, which are as follows.

* Input Channels
* AC Line Voltage
* Auxiliary Trigger

2. Acquisition System

Acquisition is the process of sampling the analog input signal, converting it into digital data, and assembling it into a waveform record at regular time intervals defined by time base.

The five in-built acquisition modes of the DSO are:

1. Sample : In Sample mode, the oscilloscope creates a record point by saving the first sample during each acquisition interval. This is the default mode.
2. Peak Detect : Peak Detect mode alternates between saving the highest sample in one acquisition interval and lowest sample in the next acquisition interval. This mode is useful to reveal aliasing and for glitch detection.
3. Hi Res. : In Hi Res mode, the digitizing oscilloscope averages all samples taken during an acquisition interval to create a record point. That average results in a higher resolution, lower-bandwidth waveform.
4. Envelope : Envelope mode lets one acquire and display a waveform record that shows the extremes in peak variation over several acquisitions .One specify the number of acquisitions over which to accumulate the data .

5. Average : Average mode lets one acquire and display a waveform record that is the averaged result of several acquisitions. This mode reduces random noise .The oscilloscope acquires data after each trigger event using sample mode. It then averages the record point from the current acquisition with those stored from previous acquisitions.

3. Scaling and Positioning Waveform System

Scaling and Positioning Waveforms means increasing or decreasing their displayed size and moving them up, down, right, and left on the display. This system in turn controls the vertical and horizontal systems used for varying the horizontal and vertical parameters.

4. Measurement System

The measurement system provides numeric information on the displayed waveforms. This is used to provide numeric readouts of parameters like area, fall time, peak-to-peak voltage, pulse width etc.

TEK has control buttons on its front panel through which one can set different parameters, acquire waveform, store waveform in REF and MATH channels for calculations and for future use. On-line help at the touch of a button is available so that the user can get necessary tips while working. But, in situations where experiments have to employ more than one equipment concurrentl/ in coordination with the TEK, front-end controls become extremely cumbersome and in some cases become impractical as well. Similar problems also will arise, where a measured parameter acquired in one module has to be passed on to other instrument during the process of experiment, or sequencing processes in industrial operations. GPIB interfacing provides a viable solution to the above problems. The following section describes the details of interfacing TDS 540 Tektronix oscilloscope with the master controller, including the design consideration and technical details of the software.

3. Implementation of SPAW Software

Basic task of this software was to establish communication between the TEK and the master controller (IBM-PC) through GPIB. Basic requirements for an instrument to communicate mutually with the connected devices (including master controller) are listed below.

- The master controller (IBM-PC) must have the GPIB card and NI488.2 software.
- Every device connected in this network must be compatible with IEEE standard.
- Each device should have a unique primary address.
- Each device is to be connected to the master controller using the 24-pin connector (GPIB cable).

The GPIB cable has 24-pin, of which 8-pins are data lines, 8-pins are shielded ground line, 5-pins are handshake line and 3-pins are interface management line. (Refer Appendix B for pin configuration.)

The primary address is in the range of 0 to 30. Using this address, while sending or receiving data the GPIB controller set the 5th or 6th bit of address byte to make that device as Listener or Talker respectively. (Refer Appendix B for byte address configuration.)

Once these requirements are fulfilled, using the unique primary address of particular device communication with that instrument can be started.

Any communication programme for GPIB communication should follow the sequence given below. For this, the basic GPIB functions and routines to be used are given in the parenthesis.

1) Find the device (ibfind, FindLstn)
2) Open the device (ibonl)
3) Perform the task (ibwrt, ibrd,ibwait)
4) Close the device (ibonl)

3.1. Task of Software

As this software basically deals with Setting up Parameters and Acquireing Waveform points, we call it as **SPAW** software. Following are the tasks performed by this software.

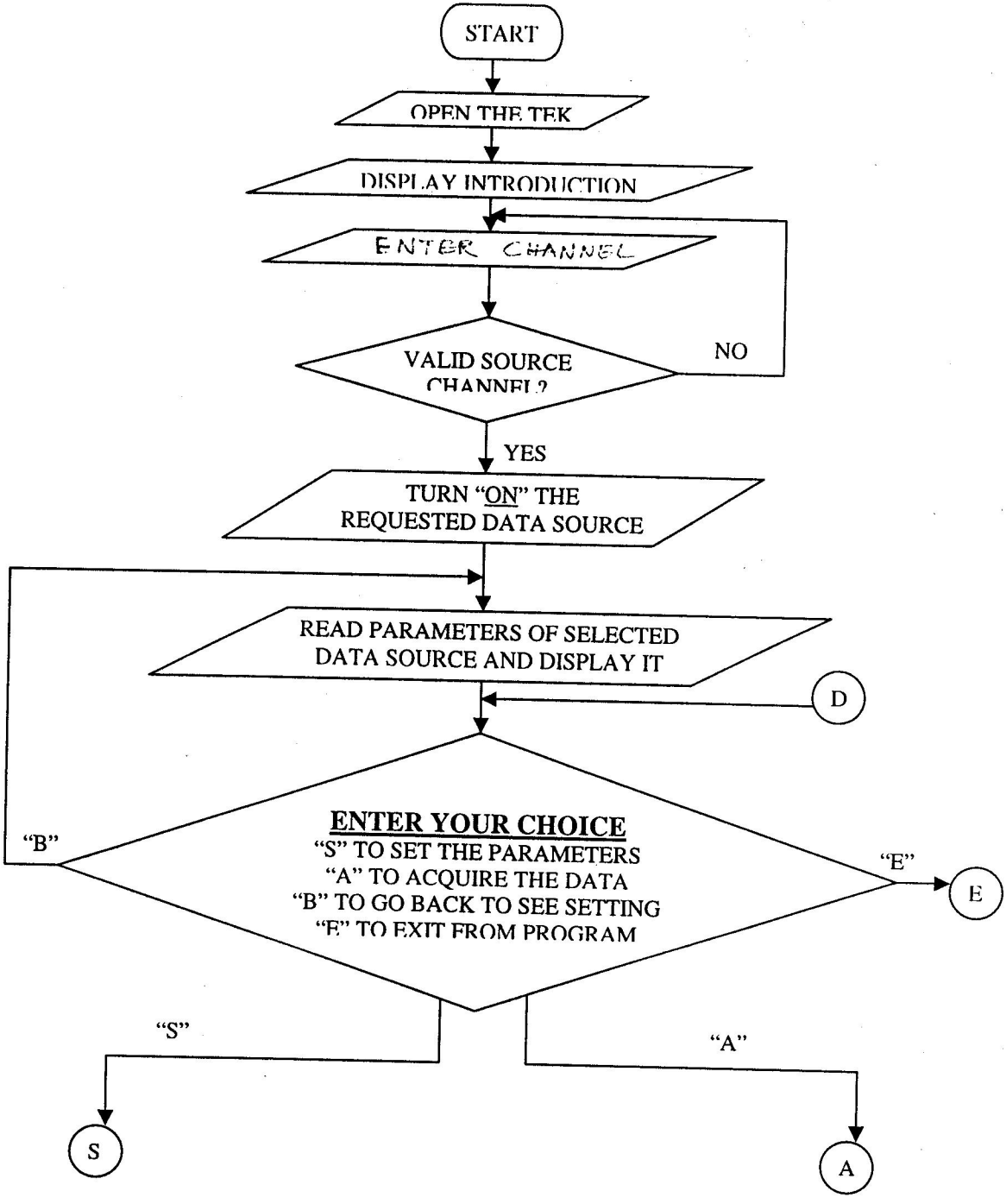
* Setup any of the various horizontal, vertical or trigger parameters.
* Setup different acquisition modes.

* Acquire waveform
* Save the acquired waveform points in a file with all parameters on demand.

The setup screen also display the corresponding current settings of all the parameters so that one can be setup accordingly.

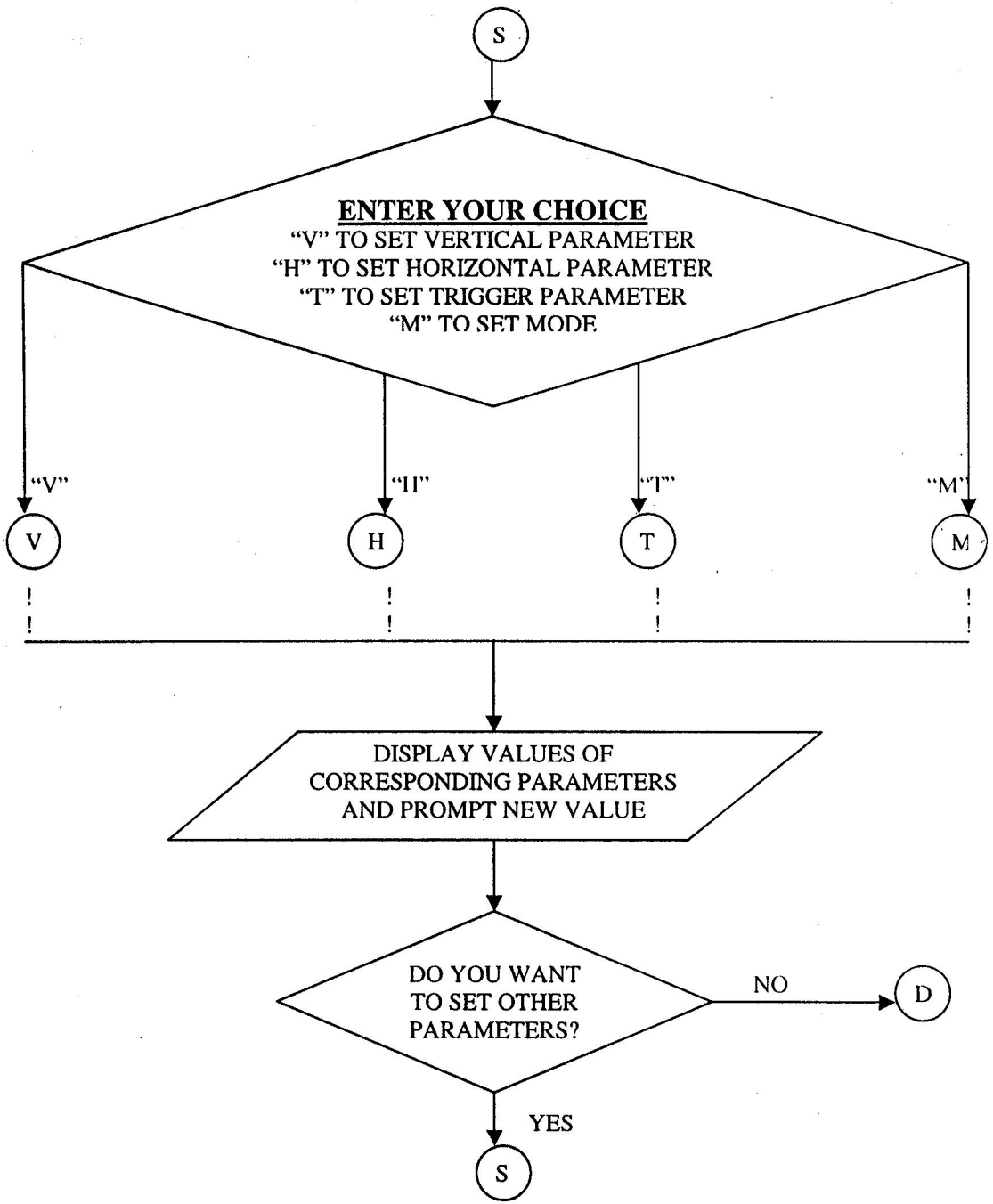
3. 11 Details of Flow chart and tree:

After giving introduction of the programme, user would be asked to enter data source of the TEK. Once data source is selected software read all the parameters of that particular channel source and display on the screen. Next pop-up menu ask user to enter either of " S, A, B or E ". Here "E" singinfice to exit give facility to exit from the programme. Here "B" significe back give facility to see the current parameter setting once again if require. Here "S" significe setup use to set different parameters. They are Vertical (V), Horizontal (H), Trigger (T) and Mode (M).On pressing either of "V, H , T or M " parameter setting menu of Vertical, Horizontal, Trigger and Mode will be appeared respectively with its current value. User have to enter the values where parameters will be set. On pressing " A " Acquire menu will open where user has to set acquire mode, start and stop no. etc. On completion user can store data points in some file or can see it on the screen. On pressing " E " user will be come out of the programme. Graph-tree of the programme given below make one to understand the flow chart quickly. Following page contain Graphics of the programme which make this programme more user-friendly.



(Continued on next page...)

(*Continued from previous page...*)



(*Continued on next page...*)

(Continued from previous page...)

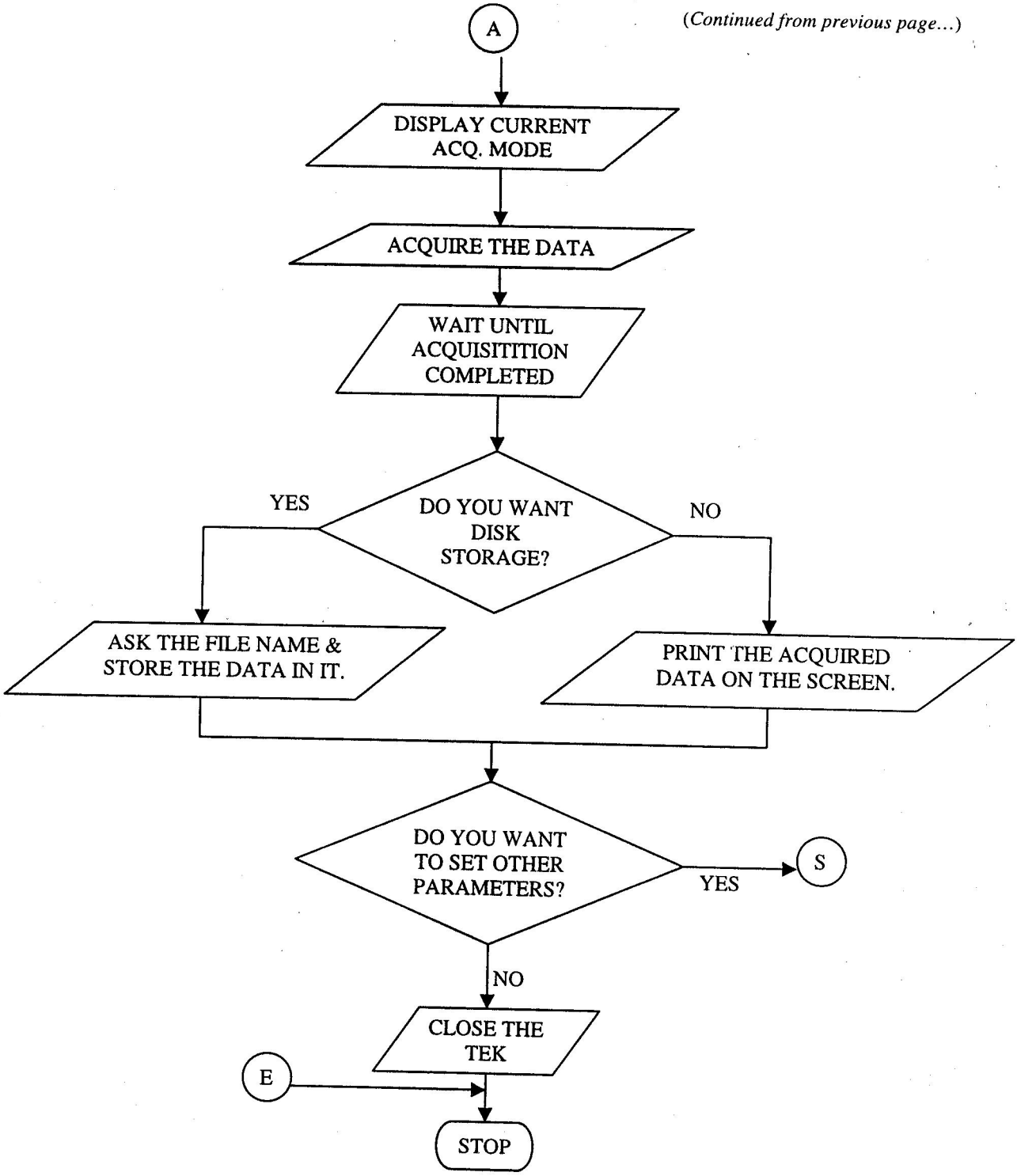


Fig 1 : Flow chart of the SPAW

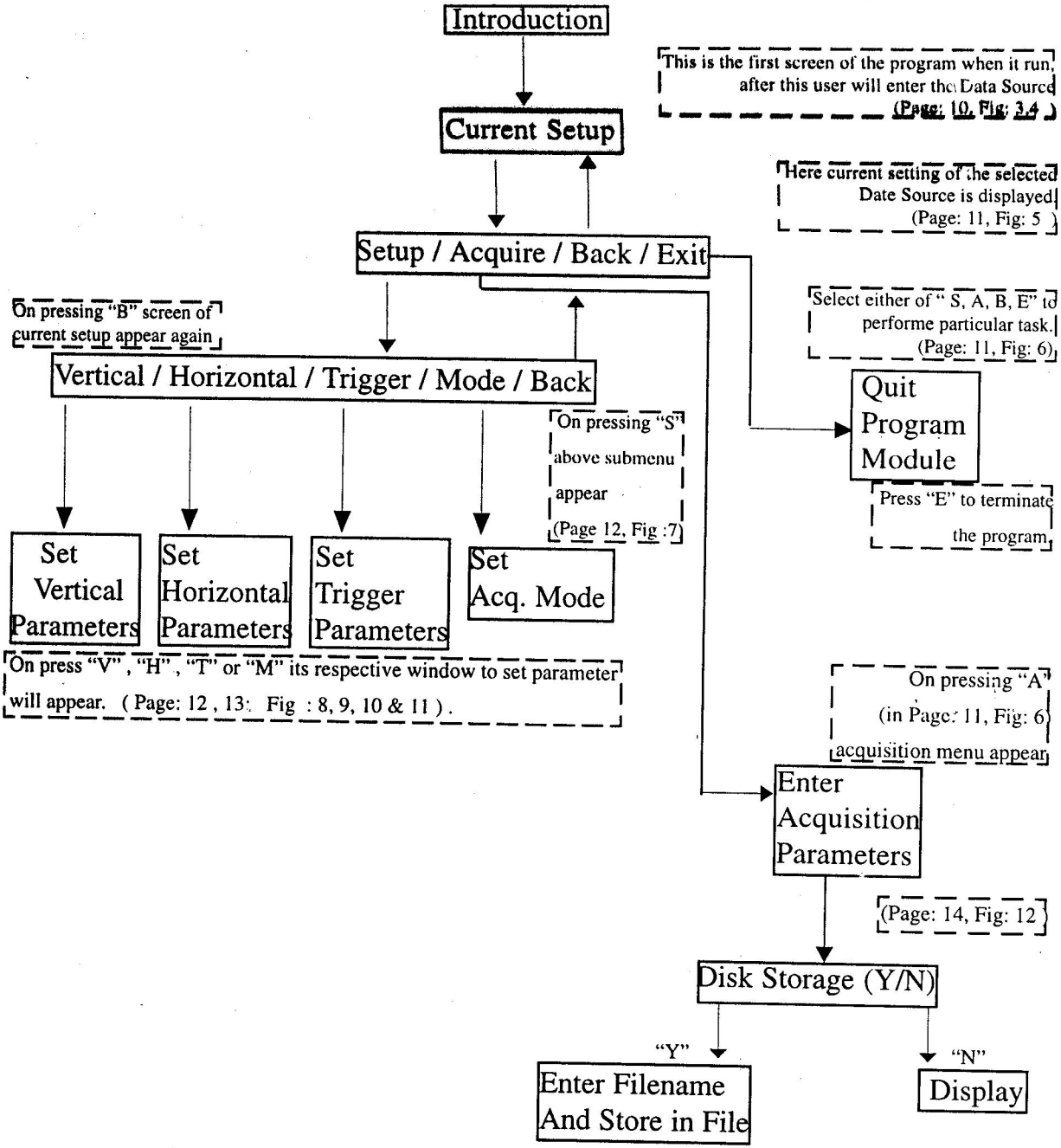


Fig 2. : Graph tree of the SPAW software

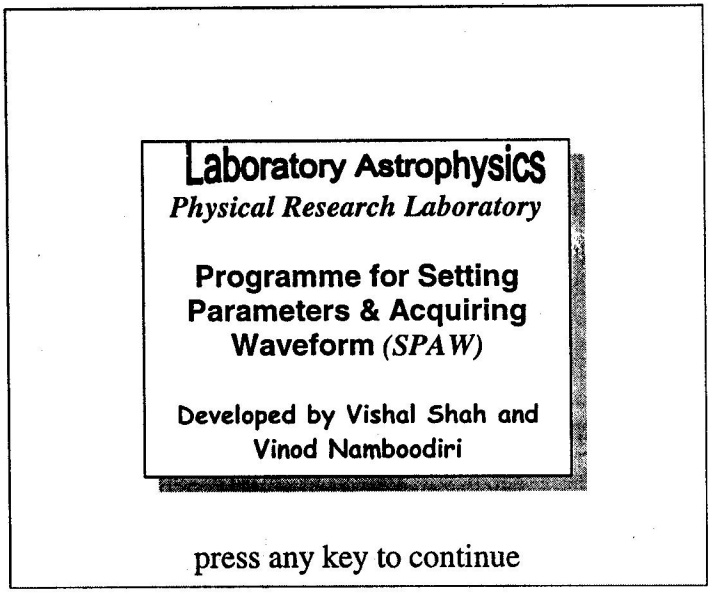


Fig 3 : The first screen of the program

This is the first screen display when the programme run. After this the screen shown below will appear. In which user has to enter correct data source.

AVAILABLE DATA SOURCES :-

For setting up the different parameters and acquiring choose any of the following.	"CH1","CH2","CH3","CH4"
To read the available data choose any of the following.	"CH1","CH2","CH3","CH4" "REF1","REF2","REF3","REF4" "MATH1","MATH2","MATH3"

ENTER DATA SOURCE : "CH3"

Fig 4 : Second screen where user has to choose the data source

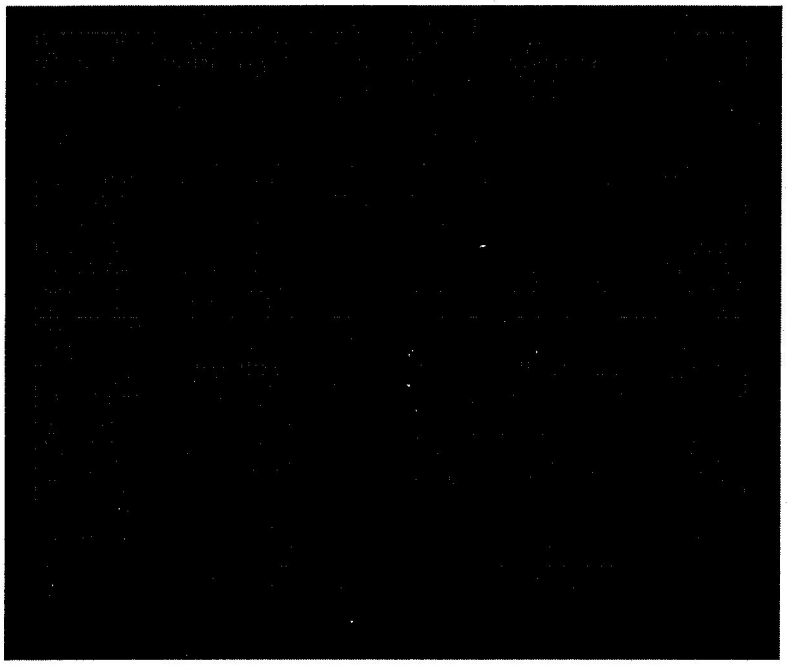


Fig 5 : Screen showing the current setup of selected data source

On entering the correct data source, above screen will appear with current readings of various parameters of the entered data source. On pressing any key, the screen shown below will appear.

The below shown screen ask user to select the task he want to do. When " S " is pressed the screen shown next will be appeared, which ask user which parameter user want to set. As shown in this screen Back option is available through which user can go to the above shown screen. On pressing "V,H,T, or M" parameter setting screen will be appeared.

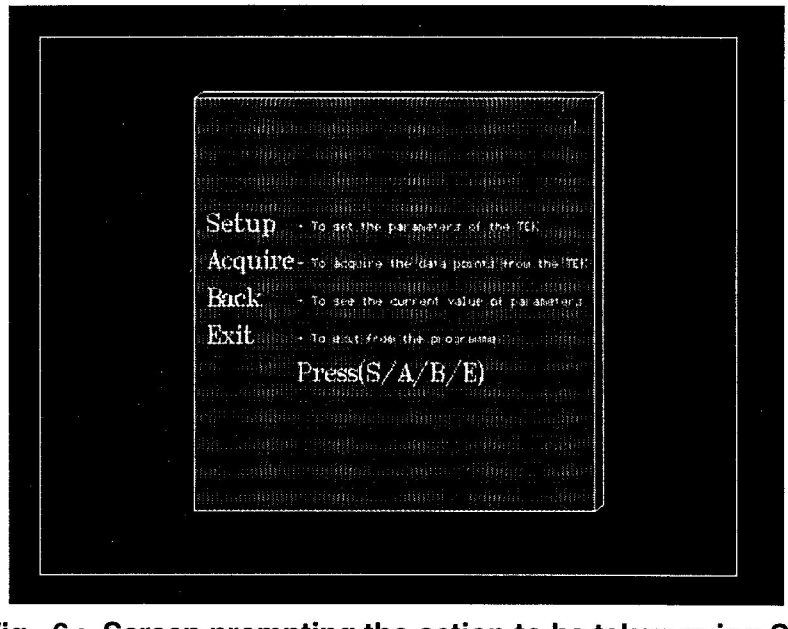


Fig 6 : Screen prompting the action to be taken using SPAW

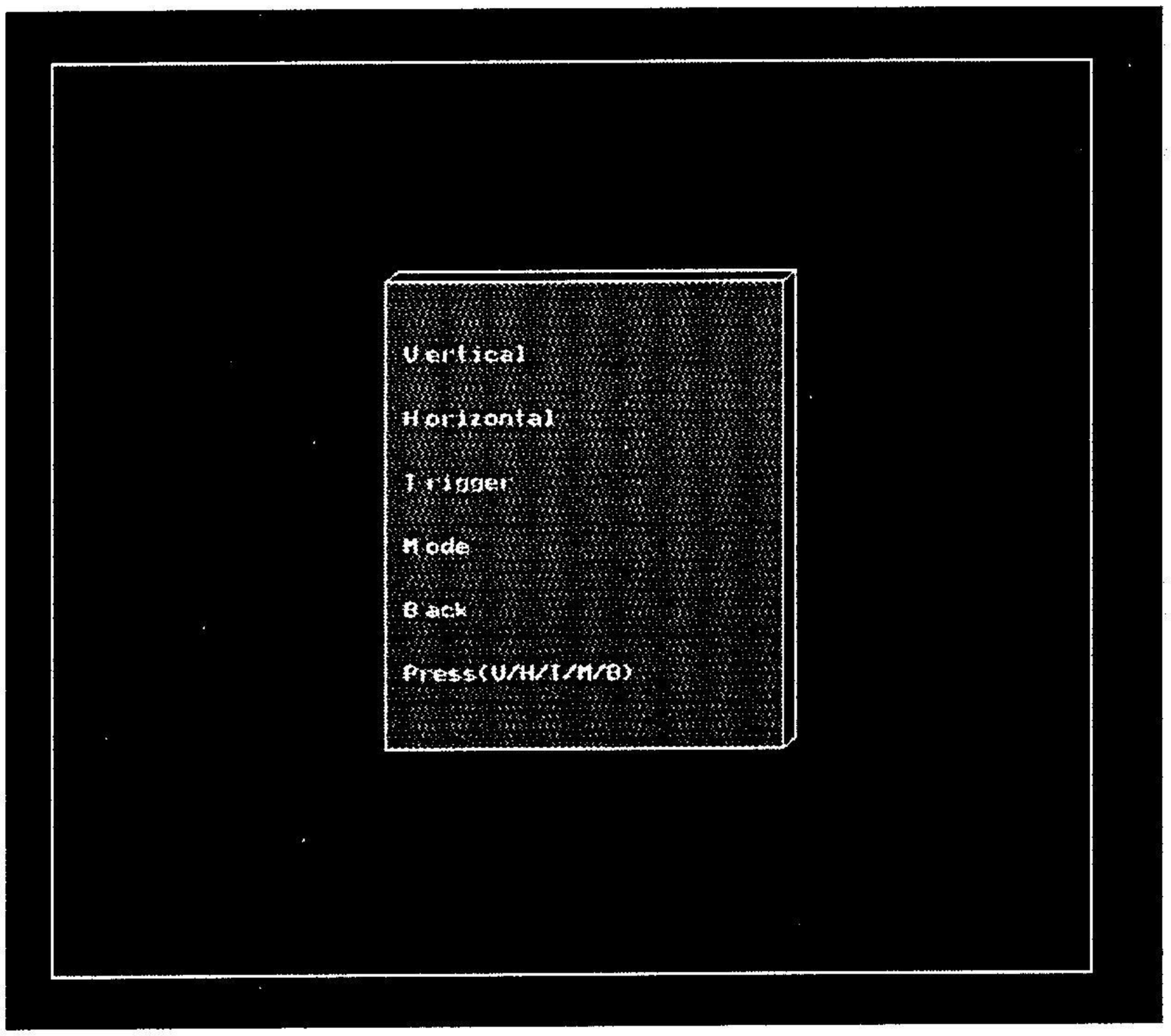


Fig 7 : Sub-level menu to change parameters of system

****

* VERTICAL *

****

	CURRENT VALUE	ENTER NEW VALUE
	****************	******************
1) SCALE	: 100E-01	" 200e-01 "
2) POSITION	: 0	" 0 "
3) OFFSET	: 0	" 0 "

DO YOU WANT TO SET OTHER PARAMETER (Y/N): " Y "

Fig 8 : Screen for setting Vertical Parameters

On pressing " V " in previous screen above screen will be displayed. Like this when "H" , "T" or "M" is pressed respective screen are opened to set their respective parameters. As shown here current values of parameters are displayed with this so there is no need to go back to the setting the parameter screen to see the current values. After filling all the parameter values user has choice to set another parameter or go back to the screen shown in Fig 7 and Fig 8 respectively by simply pressing "Y" or "N".

```
**********
* HORIZONTAL *
**********

CURRENT VALUE          ENTER NEW VALUE
**********             **********

1) RECORD LENGTH :  500              " 1000 "

2) SAMP. INTERVAL  :  200E-06        " 100E-06 "

3) TRIG. POSITION  :  40             " 10 "

DO YOU WANT TO SET OTHER PARAMETER (Y/N): " Y "
```

Fig 9 : Screen for setting Horizontal Parameters

```
**********
*  TRIGGER  *
**********

CURRENT VALUE          ENTER NEW VALUE
**********             **********

1) LEVEL    :  34E-03        " 40E-03 "

2) SLOPE    :  RIS           " LOW "

3) SOURCE   :  CH3           "CH20 "

DO YOU WANT TO SET OTHER PARAMETER (Y/N): " Y "
```

Fig 10 : Screen for setting Trigger Parameters

```
**********
*  MODE  *
**********

CURRENT VALUE          ENTER NEW VALUE
**********             **********

1) MODE     :  SAM           " AVG "

ENTER NO. ACQUISTION :  "  15  "

DO YOU WANT TO SET OTHER PARAMETER (Y/N): " Y "
```

Fig 11 : Screen for setting Acquisition Mode.

If the user press "A" in the screen shown in Fig 6 the below shown screen will be displayed. In this user have to enter " Start No" and "Stop No.". Then the user has a choice either to store the acquire data in the file or see it on the screen.

```
                    **************
                    *  ACQUIRE   *
                    **************

THE CURRENT MODE OF ACQUISITION IS "AVG".

                CURRENT VALUE          ENTER NEW VALUE
                 ***************       ******************

    1) START    : 1                     " 1 "
    2) STOP     : 500                   " 500 "
 DO YOU WANT TO STORE IN DISK (Y/N)     :      " Y "
 ENTER THE FILE NAME      :      " DATA "
ENTER YOUR MESSAGE : " TESTING OF A PROGRAMME "
WAVEFORM POINT ARE STORED IN FILE " DATA ".
```

Fig 12 : Screen for Acquiring Data Points

On complition of acquiring data ,the user will be prompted with two choices to view the data points. The first one is to store the data in a disk file and the second one is to type them out on the screen. It will be very useful to acquire a set of waveform parameters such as sampling interval, vertical scale, acquisition mode etc. for meaningful interpretation and off-line analysis of the acquired data. The following header parameters (as given in table 1) are acquired followed by the data points acquired as a string of ASCII values with comma as the delimiter.

MESSAGE	*(Here, the user can key-in comments which specify the expt.)*
DATE	Date (System Calls from PC) of the experiment
TIME	Time (System Calls from PC) of the experiment
FILE NAME	Name of the disk file in PC
RECORD LENGTH	The chosen record length, R
START NO.	Start value of the block transferred to PC, S (0 <S <E)
STOP NO.	End value of the block transferred to PC, E (S < E < R)
ACQ MODE	The mode of acquisition.
SAMPLING INT	Time base (Main), t (500 ps < t < 10 sec.)
TRIGGER POSI	The value (in %) of the displayed pre-trigger waveform
VERT. SCALE/P	Volts/Div for the Y-scale, v (1 mV < v < 10 V)
VERTICAL POS.	The channel position of the W/F corresponding to the baseline
VERT. ZER. OFST	The Y-offset in the W/F baseline.
AREA	Measured area under the W/F (volt. sec.)
PEAK	The peak value (positive) appearing in the W/F.
WIDTH	The F.W.H.M. of the W/F (if only a stable W/F)
FALL TIME	Fall time of the W/F followed by the peak.
TRIGGER LEVEL	The voltage level set to trigger an event.

(Followed by a string of ASCII data starting from START NO. to STOP NO. delimited by commas)

Table 1 : The list of header parameters appearing in the file (or on the screen) on completion of acquisition of the data points.

Appendix-A : GPIB Protocols

NI-488 Functions

The NI-488 functions are the "de facto" industry standard for GPIB programming and are fully IEEE 488.2 compatible. The high-level device functions are the best option for the majority of users. They automatically handle the GPIB communication protocol needed to manage devices on the bus. The low-level board functions provide flexibility to handle unusual GPIB situations. The following sections list several NI-488 functions and a short description of each. They are grouped according to functionality.

NI-488 Functions	Description
ibrd	Read data to buffer
ibrda	Read data asynchronously to buffer
ibrdf	Read data to file
ibwrt	Write data from buffer
ibwrta	Write data asynchronously from buffer
ibwrtf	Write data from file

Table A-1 *GPIB I/O Functions*

The GPIB I/O functions read from and write to GPIB devices.

NI-488 Function	Description
ibdev	Open and initialize a device descriptor
ibtrg	Trigger device
ibclr	Clear specified devices
ibrsp	Return serial poll byte from device
ibrpp	Conduct a parallel poll
ibppc	Configure a device for parallel poll
ibpct	Pass control
ibonl	Place device on line/ offline

Table A-2 *GPIB Device Control Functions*

The GPIB Device Control functions direct bus management and polling instructions to devices.

NI-488 Function	Description
ibfind	Open a board descriptor
ibsic	Send interface clear (IFC)
ibsre	Set/clear remote enable (REN) line
ibcac	Become Active Controller
ibln	Check for presence of GPIB device
ibgts	Go from Active Control to standby
iblines	Get status of GPIB bus management lines
ibloc	Go to local
ibstop	Abort asynchronous I/O
ibwait	Wait for GPIB event

Table A-3 *Bus Management Functions*

The Bus Management functions perform system-wide action of provide system-wide status.

NI-488 Function	Description
ibcmd	Send GPIB command from buffer
ibcmda	Send GPIB command asynchronously from buffer

Table A-4 *Low-Level I/O Functions*

The Low-Level I/O functions send more detailed information for use in unusual GPIB situations.

NI-488 Function	Description
ibask	Return driver configuration information
ibbna	Change access board of device
ibconfig	Configure the driver
ibdma	Enable/disable DMA
ibeos	Change/disable end-of-string (EOS) mode
ibeot	Enable/disable END message
ibpad	Change primary address
ibrsc	Request/release system control
ibsad	Change secondary address
ibtmo	Change/disable time limit

Table A-5 *Configuration Functions*

The Configuration functions set and retrieve NI-488.2 driver configuration information.

NI-488 Function	Description
ibist	Set/clear individual status bit for parallel poll
ibrsv	Request service

Table A-6 *Talker/Listener Functions*

The Talker/Listener functions are used in situations where the GPIB interface is not a Controller.

NI-488.2 Routines

The NI-488.2 routines implement the Controller sequences and protocols defined in the IEEE 488.2 standard. They accept a single device address or list of device addresses as an input parameter, so routines can easily address several instruments. The following sections list several NI 488.2 routines and a short description of each. The routines are grouped according to functionality.

NI-488.2 Routine	Description
Send	Send data bytes to a single GPIB device
Receive	Read data bytes from a GPIB device

Table A-7 *Simple Device I/O*

The Simple Device I/O routines can read and write to individual GPIB devices.

NI-488.2 Routine	Description
Send	Send data bytes to a single GPIB device
Receive	Read data bytes from a GPIB device

Table A-8 *Multiple Device I/O*

The Multiple Device I/O routines write the same message to several Listeners with a single message transmission.

NI-488.2 Routine	Description
Trigger	Trigger a single device
DevClear	Clear a single device
ReadStatusByte	Serial poll a single device to get its status byte
PPoll	Perform a parallel poll
PPollConfig	Configure a device for parallel polls
PPollUncoinfg	Unconfigure devices for parallel polls
PassControl	Pass control to another device with Controller capability

Table A-9 *Simple Device Control*

The Simple Device Control routines direct various bus management and polling instructions to individual devices.

NI-488.2 Routine	Description
TriggerList	Trigger multiple devices
DevClearList	Clear multiple devices
EnableRemote	Enable remote GPIB programming of devices
EnableLocal	Enable operations from decice's front panels
FindRQS	Determine which device is requesting service
AllSpoll	Serial poll all devices

Table A-10 *Multiple Device Control*

The Multiple Device Control routines direct bus management and polling instructions to several devices in the same messages.

NI-488.2 Routine	Description
ResetSys	Initialize an IEEE 488.2 system
SendIFC	Clear GPIB interface functions with IFC
FindLstn	Find all Listeners
TestSRQ	Determine the current state of the SRQ line
WaitSRQ	Wait until a device asserts Service Request
TestSys	Cause devices to conduct self-tests
SendLLO	Send the local lockout message to all devices
SetRWLS	Place devices in the Remote with Lockout state

Table A-11 *Bus Management*

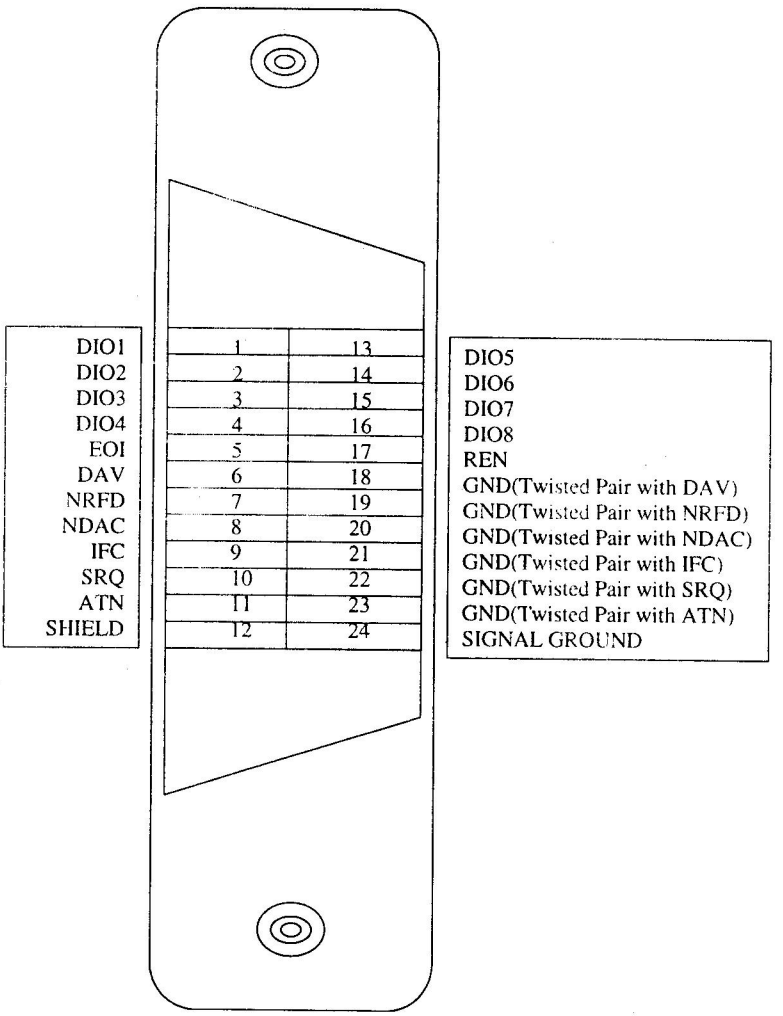
The Bus Management routines perform system-wide functions or provide system-wide status.

NI-488.2 Routine	Description
SendCmds	Send GPIB command bytes
SendDataBytes	Send data bytes to already addressed devices
SendSetup	Address devices as Listeners, GPIB board as Talker
RcvResMsg	Read data bytes from already addressed devices
ReceiveSetup	Address a device as Talker, GPIB board as Listener

Table A-12 *Low-Level I/O*

The Low-Level I/O routines break down a higher-level routine into more detailed instructions for use in unusual situation

Appendix-B : Pin configuration diagram (B.1) and bit position (B.2) of the GPIB Connector



(B.1)

Bit	7	6	5	4	3	2	1	0
Position	0	TA	LA	GPIB Primary Address Range(0 to 30)				

(B.2)

Appendix-C : Source Code Developed

Software for Setting up Parameters and Acquiring Waveforms (*SPAW*)

```
#include <process.h>
#include <errno.h>
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include "decl.h"
#include <time.h>

   int gpibWrite( int scope, char *cmd);          /*gpib write function*/
   int gpibRead( int scope, char *cmd, int cnt);  /*gpib read function*/
   int gpibWaitCom( int scope, int delay );       /*gpib wait function*/
   void gpiberr(char *msg);                       /* gpib error function */

        char mode[50];                            /* array for acquisition mode*/
        char dts[15];                             /* array for storing data source*/
        char stt[15],stt1[15];                    /* array for storing start num*/
        char stp[15],stp1[15];                    /* array for storing stop num*/
        char num[5];                              /* array for storing start num*/
        char msg[60];                             /* array for storing message*/
        int scope;                                /* address of scope */
        int brd;                                  /* handle for gpib board: GPIB0 */
        int rslt;                                 /* error return variable */
        char si[15],si1[15];                      /*array for sampling interval*/
        char tp1[5],tp[5],vo[15];                 /*array for trig. pos.& ver.off.*/
        char tl[9],tl1[9];                        /*array for trigger level "ECL", "TTL", or "<NR>">*/
        char vp[15],vs[15],am[9];                 /*arrays for ver.pos.,ver.scale &acq.mode*/
        float vp1,vs1,vo1;                        /*arrays for querying ver.pos.,ver.scale &acq.mode*/
        int i,len,x;                              /* loop index */
        char cmd[80];                             /* array for storing commands to write to the device*/
        char c;                                   /* Getting response for disk storage*/
        char rl[5],rl1[5],ts[5],ts1[5];           /* arrays for record length and trigger slope*/
        char tsrc[10],tsrc1[10];                  /*arrays for trigger source*/
        int midx,midy;                            /*variables for graphic coordinates*/
        char test[4][15];                         /*array for handling msmt. parameters*/
        char fl_name[30];                         /*array for storing filename when prompted*/
        char peak[25];                            /*array for peak-to-peak voltage*/
        char fall[25];                            /*array for fall-time*/
        char width[25];                           /*array for pulse-width*/
        char area[50];                            /*array for area*/
        FILE *outfile;                            /*output file handle */
        char na[9];                               /*array for storing n.o of acquisitions*/
        int k,v;
        static char type[9][9]={"AREA","MINI","NWIDTH","FALL"};/*msmt. parameters*/

   void display();              /*function for displaying current setup*/
   void menu();                 /*function for showing S/A/B/E options*/
   void setup();                /*function for showing V/H/T/B options*/
   void acquire();              +/*function for acquiring*/
```

```
  void graphics();              /*function for graphic commands*/
  void call();                  /*function for going back to display()*/
  void set();                   /*function for setting other parameter(Y/N) */
  void vertical();              /*function for Vertical parameters*/
  void horizontal();            /*function for Horizontal parameters*/
  void trigger();               /*function for Trigger parameters*/
  void amode();                 /*function for Mode setting*/
  void readver();               /*function for reading vertical parameters*/
  void readhori();              /*function for reading horizontal parameters*/
  void readtrig();              /*function for reading trigger parameters*/
  void readacq();               /*function for reading acquisition parameters*/

void main()
  {
        char msg1[180],msg2[180],msg3[180];/*parameter for trigger level */
        int left,top,right,bottom;          /*variables for coordinates*/
        char b[1],c[1],d[1];                /*variables for getting char.*/

/* request auto detection */
        graphics();
        left = getmaxx() / 2;
        setfillstyle(SOLID_FILL,1);
        bar3d(0,0,getmaxx(),getmaxy(),0,0);

        gotoxy(13,13);
        settextstyle(SMALL_FONT, HORIZ_DIR,6);

        outtextxy(80,75," Work that can be done with this program :-");
        outtextxy(90,84,"------------------------------------------- ");
        outtextxy(10,135," Set the parameters of the Tek.");
        outtextxy(10,165," Acquire data points from the Tek.");
        outtextxy(10,195," Run the Tek in ADD mode.");

        setcolor(14);
        outtextxy ((left-9),250,"BY");
        outtextxy ((left-60),300,"VISHAL SHAH");
        outtextxy ((left-5),320,"&");
        outtextxy ((left-80),340,"VINOD NAMBOODIRI");

/* clean up */
 getch();
 closegraph();

       printf("\nENTER MESSAGE : ");
        gets(msg);

        again:  /*Come back if invalid data source*/
        printf("\nENTER DATA SOURCE : ");
        gets(dts);
        len=strlen(dts);
/*
 * check whether the entered data source is valid or not
 */

 switch(len)
    {
        case 4:
```

```
        if((strcmp(dts,"ref1")==0)||(strcmp(dts,"REF1")==0)||
                (strcmp(dts,"ref2")==0)||(strcmp(dts,"REF2")==0)||
                (strcmp(dts,"ref3")==0)||(strcmp(dts,"REF3")==0)||
                (strcmp(dts,"ref4")==0)||(strcmp(dts,"REF4")==0))
                {
                break;
                }
        else
                {
                printf("\nInvalid Data Source");
                goto again;
                }
        case 3:

        if((strcmp(dts,"ch1")==0)||(strcmp(dts,"CH1")==0)||
                (strcmp(dts,"ch2")==0)||(strcmp(dts,"CH2")==0)||
                (strcmp(dts,"ch3")==0)||(strcmp(dts,"CH3")==0)||
                (strcmp(dts,"ch4")==0)||(strcmp(dts,"CH4")==0))
                {
                break;
                }
        else
                {
                printf("\n Invalid Data Source");
                goto again;
                }
        case 5:

        if((strcmp(dts,"math1")==0)||(strcmp(dts,"MATH1")==0)||
                (strcmp(dts,"math2")==0)||(strcmp(dts,"MATH2")==0)||
                (strcmp(dts,"math3")==0)||(strcmp(dts,"MATH3")==0)||
                (strcmp(dts,"math4")==0)||(strcmp(dts,"MATH4")==0))
                {
                break;
                }
        else
                {
                printf("\n Invalid Data Source");
                goto again;
                }
        default:
                printf("\n Invalid Data Source");
                goto again;
}

        memset( cmd, 0, 80 );   /* initialize command string to all nulls */

/*
 * Assign unique identifiers to the device DEVA store it in the
 * variable "scope" and check for errors. If DEVA is not defined,
 * ibfind returns -1.
 */

        if( ((scope = ibfind("DEVA")) < 0) ||
         ((brd = ibfind("GPIB0")) < 0))
         {
```

```
                gpiberr("ibfind Error: Unable to find device/board!");
                exit(0);
        }
/*
 * Clear the device and check for errors.
 */

         if ((ibclr(scope) < 0) ||
         (ibsre(brd,0) < 0))
         {
         gpiberr("ibclr/ibsre Error: Unable to clear device/board!");
         exit(0);
         }
         clrscr();

/*
 * Prepare to read waveform data.
 */

 sprintf( cmd, "DATA:SOURCE %s",dts);
        if(gpibWrite( scope, cmd ) < 0)
        {
                gpiberr("sprintf1 error");
                exit(1);
        }

 sprintf(cmd,"SEL:%s ON",dts);
        if(gpibWrite(scope,cmd)< 0)
        {
                gpiberr("sprintf2 error");
                exit(1);
        }

        if(gpibWrite(scope,"DATA:ENCDG ASCII" ) < 0)
        {
                gpiberr("sprintf3 error");
                exit(1);
        }

        if(gpibWrite( scope, "HEADER OFF") < 0)
        {
                gpiberr("write3 error");
                exit(1);
        }

  readacq();
  readhori();
  readtrig();
  readver();

  display();
  menu();
  return;
    }
                /*****************************************************************************/
                /****************************** DISPLAY ************************************/
```

```
/*****************************************************************************/
void display()
  {
        int left,right,top,bottom;
        graphics();

        setfillstyle(SOLID_FILL,1);
        bar3d(0,0,getmaxx(),getmaxy(),0,0);

        setfillstyle(INTERLEAVE_FILL,3);
        bar3d(25,20,610,440,0,0);

        char s1[31];
        sprintf(s1,"Current setup of channel : %s", dts);
        outtextxy(200,10,s1);
        line(getmaxx()/ 2,20,getmaxx () / 2,(getmaxy()-40));
        line(25,(getmaxy()/2-10),getmaxx()-30,(getmaxy() / 2-10));
        outtextxy(28,450,"Press any key to continue");
        setcolor(14);
        settextstyle(SMALL_FONT, HORIZ_DIR,5);
        outtextxy(130,40,"HORIZONTAL:");
        outtextxy(420,40,"VERTICAL:");
        outtextxy(130,260,"ACQUISITION:");
        outtextxy(425,260,"TRIGGER:");

        setcolor(11);
        settextstyle(SMALL_FONT, HORIZ_DIR,5);

        memset(cmd,0,80);
        sprintf(cmd,"1) RECORD LENGTH : %s",rl1);
        outtextxy(40,65,cmd);

        memset(cmd,0,80);
        sprintf(cmd,"2) SAMP. INTERVAL: %s",si1);
        outtextxy(40,95,cmd);

        memset(cmd,0,80);
        sprintf(cmd,"3) TRIG. POSITION: %s",tp1);
        outtextxy(40,125,cmd);

        memset(cmd,0,80);
        sprintf(cmd,"1) SCALE : %e",vs1);
        outtextxy(330,65,cmd);

        memset(cmd,0,80);
        sprintf(cmd,"2) POSITION : %e",vp1);
        outtextxy(330,95,cmd);

        memset(cmd,0,80);
        sprintf(cmd,"3) OFFSET : %e",vo1);
        outtextxy(330,125,cmd);

        memset(cmd,0,80);
        sprintf(cmd,"1) ACQ. MODE : %s",mode);
        outtextxy(40,285,cmd);
        memset(cmd,0,80);
```

```
        sprintf(cmd,"2) START NO. : %s",stt1);
        outtextxy(40,315,cmd);

        memset(cmd,0,80);
        sprintf(cmd,"3) STOP NO. : %s",stp1);
        outtextxy(40,345,cmd);

        memset(cmd,0,80);
        sprintf(cmd,"1) LEVEL : %s",tl1);
        outtextxy(330,285,cmd);

        memset(cmd,0,80);
        sprintf(cmd,"2) SLOPE : %s",ts1);
        outtextxy(330,315,cmd);

        memset(cmd,0,80);
        sprintf(cmd,"3) SOURCE : %s",tsrc1);
        outtextxy(330,345,cmd);

/* clean up */
        getch();
    }
        /*****************************************************************************/
        /****************************** MENU *************************************/
        /*****************************************************************************/
 void menu()
    {
        char d,e;

        midx=getmaxx()/2;
        midy=getmaxy()/2;

        graphics();
        setfillstyle(SOLID_FILL,1);
        bar3d(25,25,600,455,0,0);

        setfillstyle(INTERLEAVE_FILL,3);
        bar3d(midx-165,midy-165, midx+165, midy+165,7, 7);

        settextstyle(TRIPLEX_FONT,HORIZ_DIR,2);
        setcolor(15);
        outtextxy(165,160,"S");
        outtextxy(165,190,"A");
        outtextxy(165,220,"B");
        outtextxy(165,250,"E");
        outtextxy(240,280,"Press(S/A/B/E)");

        setcolor(11);
        outtextxy(177,160,"etup");
        outtextxy(177,190,"cquire");
        outtextxy(177,220,"ack");
        outtextxy(177,250,"xit");
        settextstyle(SMALL_FONT,HORIZ_DIR,4);
        outtextxy(240,170,"- To set the parameters of the TEK");
        outtextxy(240,200,"- To acquire the data points from the TEK");
```

```
        outtextxy(240,230,"- To see the current value of parameters");
        outtextxy(240,260,"- To exit from the program");

gotoxy(50,19);
d=getche();

switch(d)
  {
        case 'S':
        case 's':
                closegraph();
                setup();
                break;
        case 'A':
        case 'a':
                closegraph();
                acquire();
                break;
        case 'B':
        case 'b':
                closegraph();
                call();
                break;
        case 'E':
        case 'e':
                closegraph();
                exit(0);
                break;
        default:

                closegraph();
                menu();

                }
 getch();
   }

              /*****************************************************************************/
              /******************************** CALL ***********************************/
              /*****************************************************************************/

 void call()
   {
          display();
          getchar();
          menu();
          return;
   }
              /*****************************************************************************/
              /******************************** SETUP **********************************/
              /*****************************************************************************/

 void setup()
   {
          char p;
```

```
            graphics();
            setfillstyle(SOLID_FILL,1);
            bar3d(25,25,600,455,0,0);

            setfillstyle(INTERLEAVE_FILL,3);
            bar3d(midx-110,midy-110, midx+110, midy+110,7, 7);

            settextstyle(TRIPLEX_FONT,HORIZ_DIR,2);
            setcolor(15);
            outtextxy(220,160,"V");
            outtextxy(220,190,"H");
            outtextxy(220,220,"T");
            outtextxy(220,250,"M");
            outtextxy(220,280,"B");
            outtextxy(220,310,"Press(V/H/T/M/B)");

            setcolor(11);
            outtextxy(232,160,"ertical");
            outtextxy(232,190,"orizontal");
            outtextxy(232,220,"rigger");
            outtextxy(232,250,"ode");
            outtextxy(232,280,"ack");

gotoxy(51,19);
p=getche();

switch(p)
   {
        case 'v':
        case 'V':
                    closegraph();
                    vertical();
                    break;
        case 'h':
        case 'H':
                    closegraph();
                    horizontal();
                    break;
        case 't':
        case 'T':
                    closegraph();
                    trigger();
                    break;
        case 'm':
        case 'M':
                    closegraph();
                    amode();
                    break;

        case 'b':
        case 'B':
                   closegraph();
                   menu();
                   break;
        default:
                   closegraph();
```

```
                setup();
            }
    }

            /*****************************************************************************/
            /******************************** ACQUIRE ********************************/
            /*****************************************************************************/
void acquire()
    {
        char time[12],date[12];
        char p;

        printf ("\n\t THE ACQ> MODE IS %s" mode);
        printf ("\n\t DO YOU WANT TO CHANGE IT (Y/N) : ");
       p=getchar();
       switch(p)
       {
       case 'y':
       case 'Y':
                amode();
               break;
       case 'n':
       case 'N':
                break;
       default:
              printf("PLEASE ENTER (Y/N) : ");
            p=getchar();
       }

        gotoxy(42,8);
        scanf("%s",am);
        printf("\n\t2) START NO. : %s\t\t",stt1);
        gotoxy(42,10);
        scanf("%s",stt);
        printf("\n\t3) STOP NO. : %s\t\t",stp1);
        gotoxy(42,12);
        scanf("%s",stp);

    memset(cmd,0,80);
    sprintf(cmd,"DATA:ENCDG ASCII;START %s;STOP %s",&stt,&stp);
    if(gpibWrite(scope,cmd ) < 0)
          {
          gpiberr("sprintf3 error");
          exit(1);
          }


        char wfm[2540];                         /* array for raw scope input */

/*
 * Read screen image data.
 */

        if(gpibWrite( scope, "CURVE?") < 0)
```

```
                {
                    gpiberr("ibwrt Error: CURVE?");
                    exit(1);
                }
/*
 * Read in the header information. The header includes #<x><yyy>.
 */

          if(gpibRead(scope, wfm,2500)<0)      /* read in string length of 2500 bytes to transfer */
        {
                gpiberr("ibrd error:CURVE?");
                exit(1);
        }
       wfm[ibcnt]='\0';  /* force a null terminated string */


for (x=0;x<=3;x++)
   {
         memset(cmd,0,80);
         sprintf(cmd,"MEASU:MEAS%d:SOURCE1 %s;TYPE %s;STATE ON",(x+1),dts,type[x]);
         if(gpibWrite(scope,cmd)<0)
             {
                 gpiberr ("area measurement error");
                 exit(1);
             }
         if(gpibWrite(scope,"*WAI")<0)
             {
                 gpiberr ("wait error");
                 exit(1);
             }
         memset(cmd,0,80);
         sprintf (cmd,"MEASU:IMMED:TYPE %s;VAL?",type[x]);
         if (gpibWrite(scope,cmd)<0)
             {
                 gpiberr("area read error");
                 exit(1);
             }
         if(gpibRead(scope,test[x],25)<0)
             {
                 gpiberr("area measurement read error");
                 exit(1);
             }
        test[x][ibcnt]='\0';
   }

_strdate (date);
_strtime (time);
    readver();
    readhori();
    readtrig();
    readacq();

         read:
         gotoxy(6,17);
         printf("\nDISK STORAGE(y/n) : ");
         c=getchar();
         c=getchar();
```

```
switch(c)
   {
   case 'Y':
   case 'y':
         {
/*
 * Open the output file
 */

         printf("\nENTER FILENAME :");
         scanf("%s",fl_name);

         if ((outfile = fopen(fl_name, "wb" )) == NULL)
                  {
                  fprintf(stderr, "can't open output file %s\n",fl_name);
                  exit(1);
                  }
                  printf("\nENTER MESSAGE :");
                  scanf("%s",msg);
                  fprintf(outfile,"\nMESSAGE : %s",msg);
                  fprintf(outfile,"\nDATE : %s",date);
                  fprintf(outfile,"\nTIME : %s",time);
                  fprintf(outfile,"\nFILE NAME : %s",fl_name);
                  fprintf(outfile,"\nSTART NO. : %s",stt);
                  fprintf(outfile,"\nSTOP NO. : %s",stp);
                  fprintf(outfile,"\nRECORD LENGTH: %s",rl1);
                  fprintf(outfile,"\nACQ MODE : %s",mode);
                  fprintf(outfile,"\nSAMPLING INT : %s",si1);
                  fprintf(outfile,"\nTRIGGER POSI : %s",tp1);
                  fprintf(outfile,"\nVERT.SCALE/P : %e",vs1);
                  fprintf(outfile,"\nVERTICAL POS.: %e",vp1);
                  fprintf(outfile,"\nVERT.ZER.OFST: %e\n",vo1);
                  fprintf(outfile,"AREA : %s",test[0]);
                  fprintf(outfile,"PEAK : %s",test[1]);
                  fprintf(outfile,"WIDTH : %s",test[2]);
                  fprintf(outfile,"FALL TIME : %s",test[3]);
                  fprintf(outfile,"TRIGGER LEVEL: %s",tl1);

                  fprintf(outfile,"\r\n%s",wfm);
                  fprintf(outfile,"\r OVER");

                  printf("\n\t Waveform voltage points are stored in");
                  printf(" file %s\n\n",fl_name);
         break;
         }
         case 'n':
         case 'N':
         {
/*
 * Show o/p on screen
 */
                  printf("\n\t Press any key to see the waveform on screen");
                  getch();

                  printf("MESSAGE : %s",msg);
                  printf("\rDATE : %s",date);
```

```
                printf("\rTIME : %s",time);
                printf("\rFILE NAME : %s",fl_name);
                printf("\rSTART NO. : %s",stt);
                printf("\rSTOP NO. : %s",stp);
                printf("\rRECORD LENGTH: %s",rl1);
                printf("\rACQ MODE : %s",mode);
                printf("\rSAMPLING INT.: %s",si1);
                printf("\rTRIGGER POSI : %s",tp1);
                printf("\rVER. SCALE/P : %e",vs1);
                printf("\rVERTICAL POS.: %e",vp1);
                printf("\rVER.ZER.OFST : %e",vo1);
                printf("\rAREA : %s",test[0]);
                printf("\rPEAK : %s",test[1]);
                printf("\rWIDTH : %s",test[2]);
                printf("\rFALL TIME : %s",test[3]);
                printf("\rTRIGGER LEVEL: %s",tl1);

                printf("\r\n%s",wfm);
                printf("\r OVER");
        break;
        }
        default:
                printf("Character entered not y or n");
                goto read;
    }

/*
 * Cleanup time
 */
 fclose(outfile);
 exit (0);
    }

            /*****************************************************************************/
            /****************************** READ ACQUIRE ************************/
            /*****************************************************************************/
void readacq()
   {
        if(gpibWrite(scope,"DATA:START?" ) < 0)
            {
                gpiberr("sprintf3 error");
                exit(1);
            }
         if(gpibRead(scope,stt1,15)<0)
            {
                gpiberr("start stop read error");
                exit(1);
            }
        stt1[ibcnt]='\0';

        if(gpibWrite(scope,"DATA:STOP?" ) < 0)
            {
                gpiberr("sprintf3 error");
                exit(1);
```

```
        }
    if(gpibRead(scope,stp1,15)<0)
        {
            gpiberr("start stop read error");
            exit(1);
        }
    stp1[ibcnt]='\0';

    if(gpibWrite(scope,"ACQ:MODE?")<0)
        {
          gpiberr("acquisition write error");
          exit(1);
        }

    if(gpibRead(scope,mode,15)<0)
        {
            gpiberr("acquisition read error");
            exit(1);
        }
    mode[ibcnt]='\0';
}

        /*************************************************************************/
        /********************************* VERTICAL ******************************/
        /*************************************************************************/

void vertical()
  {
      char p;
      printf ("\t\t\t ************** ");
      printf ("\n\t\t\t * VERTICAL * ");
      printf ("\n\t\t\t ************** ");
      printf ("\n\n\t\t CURRENT SETUP \t PARAMETER SETTING ");
      printf ("\n\t\t ************* \t ***************** ");
      printf("\n\t1) SCALE : %e\t",vs1);
      gotoxy(50,7);
      scanf("%s",vs);
      printf("\n\t2) POSITION : %e\t",vp1);
      gotoxy(50,9);
      scanf("%s",vp);
      printf("\n\t3) OFFSET : %e\t",vo1);
      gotoxy(50,11);
      scanf("%s",vo);

      memset(cmd,0,80);
      sprintf(cmd,"%s:OFFS %s;POS %s;SCAL %s",dts,vo,vp,vs);
     if(gpibWrite(scope,cmd ) < 0)
        {
          gpiberr("vertical setting error");
          exit(1);
        }
      readver();
      set();
  }
```

```
/*****************************************************************************/
/************************** READ VERTICAL ************************************/
/*****************************************************************************/
void readver()
  {
        memset(cmd,0,80);
        sprintf(cmd,"%s:OFFS?;POS?;SCAL?",dts);
       if(gpibWrite(scope,cmd) < 0)
           {
               gpiberr("Vertical parameter error");
               exit(1);
           }
       if(gpibRead(scope,cmd,80)<0)
           {
               gpiberr("Vertical parameter read error");
               exit(1);
           }
        cmd[ibcnt]='\0';
        sscanf(cmd,"%e;%e;%e",&vo1,&vp1,&vs1);
  }

/*****************************************************************************/
/************************** HORIZONTAL ***************************************/
/*****************************************************************************/
void horizontal()
  {
        char p;
        printf ("\t\t\t ********** ");
        printf ("\n\t\t\t * HORIZONTAL * ");
        printf ("\n\t\t\t ********** ");
        printf ("\n\n\t\t\tCURRENT SETUP \t PARAMETER SETTING ");
        printf ("\n\t\t\t********** \t ********** ");

        gotoxy(6,7);
        printf("1) RECORD LENGTH: %s",rl1);
        gotoxy(50,7);
       scanf("%s",rl);
       gotoxy(6,9);
       printf("2) SAM. INTERVAL: %s",si1);
       gotoxy(50,9);
       scanf("%s",si);
        gotoxy(6,11);
        printf("3) TRI. POSITION: %s",tp1);
        gotoxy(50,11);
        scanf("%s",tp);

        memset(cmd,0,80);
        sprintf(cmd,"HOR:MAI:SCA %s;:HOR:TRIG:POS %s",si,tp);
       if(gpibWrite(scope,cmd ) < 0)
           {
               gpiberr("horizontal setting error");
               exit(1);
           }
```

```
        memset(cmd,0,80);
        sprintf(cmd,"HOR:RECO %s",rl);
       if(gpibWrite (scope,cmd)<0)
            {
                gpiberr("reco error");
                exit(1);
            }

        if(gpibWrite(scope,"ACQ:STATE ON")<0)
            {
                gpiberr("acquisition write error");
                exit(1);
            }

       if(gpibWrite(scope,"FORCE TRIGGER")<0)
            {
                gpiberr("Reco trigger set error");
                exit(1);
            }
       if(gpibWrite(scope,"ACQ:STATE OFF")<0)
            {
                gpiberr("acquisition write1 error");
                exit(1);
            }

        readhori();
        set();
   }

            /*****************************************************************************/
            /************************** READ HORIZONTAL ************************/
            /*****************************************************************************/
void readhori()
   {
       if(gpibWrite (scope,"HOR:RECO?")<0)
            {
                gpiberr("reco error");
                exit(1);
            }
       iif(gpibRead(scope,rl1,25)<0)
            {
                gpiberr("reco read error");
                exit(1);
            }
rl1[ibcnt]='\0';

       memset(cmd,0,80);
       if(gpibWrite (scope,"HOR:MAI:SCA?")<0)
           {
                gpiberr("horizontal error");
                exit(1);
           }
       if(gpibRead(scope,si1,15)<0)
           {
                gpiberr("Sampling Interval read error");
```

```
                exit(1);
            }
        si1[ibcnt]='\0';

        if(gpibWrite (scope,"HOR:TRIG:POS?")<0)
            {
                gpiberr("horizontal trig. pos. error");
                 exit(1);
            }
        if(gpibRead(scope,tp1,15)<0)
            {
                gpiberr("trig. pos. read error");
                exit(1);
            }
        tp1[ibcnt]='\0';
    }

            /*****************************************************************************/
            /****************************** TRIGGER ******************************/
            /*****************************************************************************/

void trigger()
    {
        char p;
        printf ("\t\t\t ************* ");
        printf ("\n\t\t\t* TRIGGER * ");
        printf ("\n\t\t\t************* ");
        printf ("\n\n\t\t\tCURRENT SETUP \t PARAMETER SETTING ");
        printf ("\n\t\t\t************* \t ***************** ");

        gotoxy(6,7);
        printf("1) TRIG. LEVEL : %s",tl1);
        gotoxy(50,7);
        scanf("%s",tl);
        gotoxy(6,9);
        printf("2) TRIG. SLOPE : %s",ts1);
        gotoxy(50,9);
        scanf("%s",ts);
        gotoxy(6,11);
        printf("3) TRIG. SOURCE: %s",tsrc1);
        gotoxy(50,11);
        scanf("%s",tsrc);
        memset(cmd,0,80);
        sprintf(cmd,"TRIGGER:MAIN:LEVEL %s",tl);
        if (gpibWrite(scope,cmd)<0)
            {
              gpiberr("triggerlevel error");
              exit(1);
            }
        sprintf(cmd,"TRIGGER:MAIN:EDGE:SLOPE %s",ts);
        if (gpibWrite(scope,cmd)<0)
            {
                gpiberr("trigger slope error");
                exit(1);
            }
        sprintf(cmd,"TRIGGER:MAIN:EDGE:SOURCE %s",tsrc);
```

```
        if (gpibWrite(scope,cmd)<0)
            {
                gpiberr("trigger source error");
                exit(1);
            }
    readtrig();
    set();
    }

                /*****************************************************************************/
                /************************** READ TRIGGER ***************************/
                /*****************************************************************************/
void readtrig()
    {
        if (gpibWrite(scope,"TRIGGER:MAIN:LEVEL?")<0)
            {
                gpiberr("triggerlevel error");
                exit(1);
            }
        if (gpibRead(scope,tl1,15)<0)
            {
                gpiberr("trigger level read error");
                exit(1);
            }
        tl1[ibcnt]='\0';

        if (gpibWrite(scope,"TRIGGER:MAIN:EDGE:SLOPE?")<0)
            {
                gpiberr("trigger slope error");
                exit(1);
            }

        if (gpibRead(scope,ts1,15)<0)
            {
                gpiberr("trigger slope read error");
                exit(1);
            }
         ts1[ibcnt]='\0';

         if (gpibWrite(scope,"TRIGGER:MAIN:EDGE:SOURCE?")<0)
              {
                gpiberr("trigger source error");
                exit(1);
              }
         if (gpibRead(scope,tsrc1,15)<0)
              {
                  gpiberr("trigger source read error");
                  exit(1);
              }
          tsrc1[ibcnt]='\0';
    }
```

```
/*****************************************************************************/
/***************************** ACQ. MODE *****************************/
/*****************************************************************************/
void amode()
  {
        char p;
        printf ("\t\t\t ************ ");
        printf ("\n\t\t\t* ACQ. MODE ");
        printf ("\n\t\t\t************ ");
        printf ("\n\n\t\t\tCURRENT SETUP \t PARAMETER SETTING ");
        printf ("\n\t\t\t************ \t ********** ");

       gotoxy(6,7);
       printf("1) ACQ. MODE  : %s",mode);
       gotoxy(50,7);
       scanf("%s",am);

       memset(cmd,0,80);
        sprintf(cmd,"ACQ:MODE %s",am);
       if(gpibWrite(scope,cmd)<0)
            {
                 gpiberr("acquisition write error");
                 exit(1);
            }
  if((strcmp(am,"ave")==0)||(strcmp(am,"average")==0)||
  (strcmp(am,"AVE")==0)||(strcmp(am,"AVERAGE")==0))
       {
        gotoxy(6,15);
        printf("\nENTER NO. OF ACQ.:");
        scanf ("%s",na);

   memset(cmd,0,80);
   sprintf(cmd,"ACQ:NUMAV %s",na);
  if(gpibWrite(scope,cmd ) < 0)
       {
        gpiberr("No. of Acq. error");
        exit(1);
       }

  memset(cmd,0,80);
   sprintf(cmd,"TRIG:MAIN:EDGE:SOURCE %s",dts);
   if(gpibWrite(scope,cmd)<0)
       {
        gpiberr("Trigger source set error");
        exit(1);
       }

   if(gpibWrite(scope,"ACQ:STOPA SEQ" ) < 0)
       {
        gpiberr("run error");
        exit(1);
       }
  if(gpibWrite(scope,"ACQ:STATE RUN" ) < 0)
       {
        gpiberr("run error");
        exit(1);
```

```
else if((strcmp(am,"env")==0)||(strcmp(am,"envelope")==0)||
(strcmp(am,"ENV")==0)||(strcmp(am,"ENVELOPE")==0))
      {
       gotoxy(6,15);
       printf("\nENTER NO. OF ACQ.:");
       scanf("%s",na);

       memset(cmd,0,80);
       sprintf(cmd,"ACQ:NUMEN %s",na);
       if(gpibWrite(scope,cmd ) < 0)
           {
                gpiberr("No. of Acq. error");
                exit(1);
           }

       memset(cmd,0,80);
       sprintf(cmd,"TRIG:MAIN:EDGE:SOURCE %s",dts);
       if(gpibWrite(scope,cmd)<0)
           {
                gpiberr("Trigger source set error");
                exit(1);
           }

      if(gpibWrite(scope,"ACQ:STOPA SEQ" ) < 0)
           {
                gpiberr("run error");
                exit(1);
           }
       if(gpibWrite(scope,"ACQ:STATE RUN" ) < 0)
           {
                gpiberr("run error");
                exit(1);
           }
}

else
 {

      memset(cmd,0,80);
      sprintf(cmd,"TRIG:MAIN:EDGE:SOURCE %s",tsrc);
      if(gpibWrite(scope,cmd)<0)
           {
                gpiberr("Trigger source set error");
                exit(1);
           }

      if(gpibWrite(scope,"ACQ:STOPA RUNST" ) < 0)
         {
                gpiberr("run error");
                exit(1);
         }

      if(gpibWrite(scope,"ACQ:STATE RUN" ) < 0)
         {
```

```
                gpiberr("run error");
                exit(1);
            }

        if(gpibWrite(scope,"TRIG FORCE" ) < 0)
            {
                gpiberr("trigger force error");
                exit(1);
            }

        if(gpibWrite(scope,"ACQ:STATE OFF" ) < 0)
            {
                gpiberr("off error");
                exit(1);
            }
    }
        set();
        }
            /*****************************************************************************/
            /********************************* SET ***************************************/
            /*****************************************************************************/

void set()
    {
        char p;
        printf("\nDO YOU WANT TO SET OTHER PARAMETER (Y/N):");
        p=getche();

switch(p)
    {
    case 'Y':
    case 'y':
                closegraph();
                setup();
                break;
    case 'N':
    case 'n':
                closegraph();
                menu();
                break;
    default:
                set();
    }
        return;
    }

            /*****************************************************************************/
            /*************************** GRAPHICS ****************************************/
            /*****************************************************************************/

void graphics()
    {
        int gdriver = DETECT, gmode, errorcode;
        int left, top, right, bottom,color,maxcolor;
```

```
        char msg1[50],msg2[50];                 /* initialize graphics and local variables */
        initgraph(&gdriver, &gmode, "");

                                                /* read result of initialization */
        errorcode = graphresult();
        if (errorcode != grOk)                  /* an error occurred */
  {
        printf("Graphics error: %s\n",grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1);                                /* terminate with an error code */
  }
        return;
  }

        /*************************************************************************
         * gpibRead - read into the string from the device and wait for the
         * read to finish.
         *************************************************************************/

int gpibRead( int dev, char *resp, int cnt )
   {
/*
 * set the timeout for 10 seconds, send the command, and
 * wait for the scope to finish processing the command.
 */
        ibtmo(dev,13);
        ibrd(dev,resp,cnt);
/*
 * If ibrd was successful, wait for scope completion.
 */
        if(ibsta >=0)
                ibwait(dev,STOPend);

       return(ibsta);
   }

        /*************************************************************************
         * gpibWait - wait for a gpib command to finish by doing a query
         * and reading its results; wait only as long as the delay value
         *************************************************************************/

int gpibWaitCom(int dev, int delay)
   {
        char rd[6];
        gpibWrite(dev,"*OPC?");
        if(ibsta >= 0)
                gpibRead(dev,rd, strlen(rd));

        return(ibsta);
   }
```

```
/*****************************************************************************
 * gpibWrite - send the contents of the string to the device and wait
 * for the write to finish.
 *****************************************************************************/

int gpibWrite( int dev, char *cmd)
   {
        int cmd_len;
        cmd_len = strlen(cmd);
/*
* set the timeout for 10 seconds, send the command
* wait for the scope to finish processing the command.
*/
        ibtmo(dev,13);
        ibwrt (dev,cmd,cmd_len);
/*
* If ibwrt was successful, wait for scope completion.
*/
        if(ibsta >=0)
                ibwait(dev,STOPend);

        return(ibsta);
    }

        /*****************************************************************************
         * gpiberr - display error from defined error codes based on what
         * is contained in ibsta. This routine would notify you that an IB
         * call failed.
         *****************************************************************************/

void gpiberr(char *msg)
   {
        printf ("%s\n", msg);
        printf ( "ibsta=&H%x <", ibsta);

        if (ibsta & ERR ) printf (" ERR");
        if (ibsta & TIMO) printf (" TIMO");
        if (ibsta & END ) printf (" END");
        if (ibsta & SRQI) printf (" SRQI");
        if (ibsta & RQS ) printf (" RQS");
        if (ibsta & CMPL) printf (" CMPL");
        if (ibsta & LOK ) printf (" LOK");
        if (ibsta & REM ) printf (" REM");
        if (ibsta & CIC ) printf (" CIC");
        if (ibsta & ATN ) printf (" ATN");
        if (ibsta & TACS) printf (" TACS");
        if (ibsta & LACS) printf (" LACS");
        if (ibsta & DTAS) printf (" DTAS");
        if (ibsta & DCAS) printf (" DCAS");
        printf (" >\n");

        printf ("iberr= %d", iberr);
        if (iberr == EDVR) printf (" EDVR <DOS Error>\n");
```

```
if (iberr == ECIC) printf (" ECIC <Not CIC>\n");
if (iberr == ENOL) printf (" ENOL <No Listener>\n");
if (iberr == EADR) printf (" EADR <Address error>\n");
if (iberr == EARG) printf (" EARG <Invalid argument>\n");
if (iberr == ESAC) printf (" ESAC <Not Sys Ctrlr>\n");
if (iberr == EABO) printf (" EABO <Op. aborted>\n");

if (iberr == ENEB) printf (" ENEB <No GPIB board>\n");
if (iberr == EOIP) printf (" EOIP <Async I/O in prg>\n");
if (iberr == ECAP) printf (" ECAP <No capability>\n");
if (iberr == EFSO) printf (" EFSO <File sys. error>\n");
if (iberr == EBUS) printf (" EBUS <Command error>\n");
if (iberr == ESTB) printf (" ESTB <Status byte lost>\n");
if (iberr == ESRQ) printf (" ESRQ <SRQ stuck on>\n");
if (iberr == ETAB) printf (" ETAB <Table Overflow>\n");
```