



HANDSHAKE

NEWSLETTER OF SIGNAL PROCESSING AND INSTRUMENT CONTROL



**Data Logging...
Capturing the Unpredictable**

Table of contents

Data Logging...

Capturing the unpredictable

Part I—First steps to data logging	3
Part II—Event detection	9
Part III—Retrieving logged data	15
TEK SPS BASIC routine for single-key program selection	20
Getting the most out of TEK BASIC graphics	22
Four new programmable digitizer systems available	24
1360P/1360S provides GPIB control of 129 signal paths	25
New Real Time Clock ROM Pack—	26
7612D improves resolution on low duty cycle pulse capture	27
Literature available from Tektronix	28

Managing Editor: Bob Ramirez

Edited by: Bob Ramirez

Graphics by: Joann Crook

HANDSHAKE is published quarterly by the HANDSHAKE Group. Permission to reprint material in this publication may be obtained by writing to:

HANDSHAKE Editor
Group 157 (94-384)
Tektronix, Inc.
P.O. Box 500
Beaverton, Oregon 97077

HANDSHAKE is provided free of charge by Tektronix, Inc., as a forum for people interested in programmable instrumentation and digital signal processing. As a free forum, statements and opinions of the authors should be taken as just that—statements and opinions of the individual authors. Material published in HANDSHAKE should not be taken as or interpreted as statement of Tektronix policy or opinion unless specifically stated to be such.

Also, neither HANDSHAKE nor Tektronix, Inc., can be held responsible for errors in HANDSHAKE or the effects of these errors. The material in HANDSHAKE comes from a wide variety of sources, and, although the material is edited and believed to be correct, the accuracy of the source material cannot be fully guaranteed. Nor can HANDSHAKE or Tektronix, Inc., make guarantees against typographical or human errors, and accordingly no responsibility is assumed to any person using the material published in HANDSHAKE.


Data logging... Capturing the unpredictable

Since its seismic stirrings began in March of 1980, Washington's Mount St. Helens has been a spectacular example of modern data logging needs and capabilities. Hundreds of monitoring stations make up the Washington and Oregon seismic network. Most stations continuously monitor the mountain...discarding unwanted activity...waiting for something significant to happen. And plenty has happened, including the devastating eruption shown on our cover. There's been, literally, a mountain of data to capture, transmit, and analyze.

But Mount St. Helens is just one example of data logging. Less spectacular, but every bit as important, are the numerous uses of data logging techniques in physics, biomedicine, chemistry, nuclear research, vibration mechanics, anywhere that events, trends, or changes occur unpredictably, in rapid succession or just occasionally.

In almost every case, whether you are monitoring a mountain or a machine bearing, data logging comprises three major operations:

1. Capturing and storing sensor output
2. Event detection
3. Retrieval of stored event data

This issue of HANDSHAKE covers all three aspects, starting with an article on simple data logging methods using the TEKTRONIX 7612D Programmable Digitizer and TEK SPS BASIC software. A second article continues with an examination of ways to distinguish events wanted for storage from unwanted base-level activity (e.g., noise). And, finally, a third article looks at retrieving the stored data. 

Data Logging...

Capturing the unpredictable

Part I—First steps to data logging

The violent eruption of Mount St. Helens on May 18, 1980 stunned people in the northwest United States and around the world. But for months before and after the eruption, scientists have carefully monitored the mountain's every shudder and quake. There are many days—even weeks—when little or nothing happens. But they can't stop watching because the subtle hints of what's happening inside the mountain come in short bursts of data buried in hours of silence.

In the past, this kind of long-term monitoring often required low-speed recording on paper or film. If events had to be detected in real-time, someone had to stay glued to a chart recorder or oscilloscope for hours on end. Today, much of the monitoring, event detection, and recording is done electronically. For example, most of the data from St. Helens is collected by seismometers placed around the mountain. These devices detect tremors and either transmit the data to a remote recording site or record data locally on magnetic tape.

This process of monitoring and recording data is generally referred to as "data logging" and it is used in a variety of fields, from earthquake seismology to monitoring high-speed data

communication channels. (See "Monitoring Data Communication Channels with the 7612D", HANDSHAKE Fall 1980). This, the first part of a three-part article, explores some simple approaches to data logging with TEK SPS BASIC and the 7612D Programmable Digitizer. The techniques described here can (with minor adaptation) be applied to most any data logging application for signals up to the bandwidth of the acquisition instrument (90 MHz, for the 7612D).

Digital data logging

A typical seismic data logging system based on the 7612D is shown in Fig. 1. The seismometer(s) sense vibration from earthquakes, machinery, or other vibration sources, and convert it to electrical signals. These signals are fed to a 7612D equipped with 7A16P Programmable Amplifiers. The 7612D samples the input signals and converts them to digital information for processing and/or storage. Digitized data from the 7612D is sent to the computer over the IEEE-488 bus (GPIB). The computer may simply transfer the data to mass storage device (magnetic disk or tape), or it may perform some event detection or processing before writing the data out.

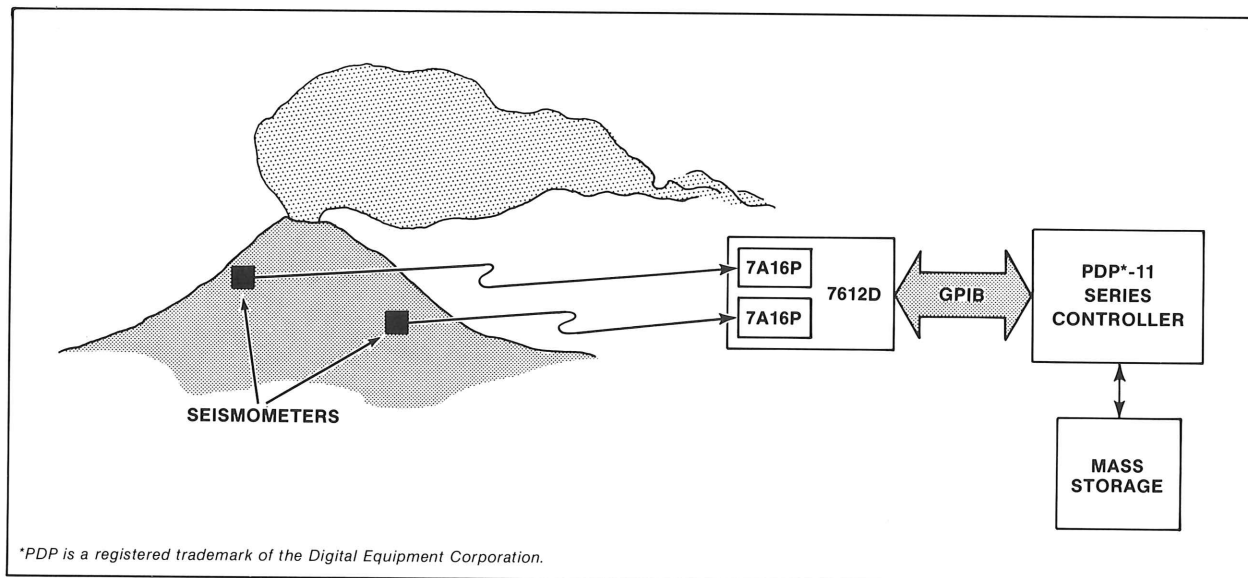


Fig. 1. A typical seismic data logging system using a 7612D and a PDP-11 series controller running TEK SPS BASIC. The seismometers can monitor earthquake activity or vibration from machinery or other vibration sources.

Data logging...Part I

In the simplest case, the computer just acts as a pipeline for data between the 7612D and the mass storage device. The object is to acquire the data and write it to the mass storage device as fast as possible. Little or no processing is done, and the 7612D's triggering is all that is used for event detection.

Figure 2 shows a program that logs data in this simple form, with no scale factors or other information. It takes advantage of the DLOG command that is part of the optional 7612D Commands Package for TEK SPS BASIC V02 and V02XM.

```
10 LOAD "INS
20 ATTACH #1 AS INS1,0:WITH 1,2 @0
30 PRINT "ENTER FILENAME FOR DATA LOGGING OUTPUT: ";
40 INPUT FS
50 OPEN #2 AS DLOG:FS FOR WRITE
60 PRINT "ENTER ACQUISITION MODE (ALT N OR REP N,C)"
70 PRINT " WHERE N=NO. OF REPETITIONS, C=CHANNEL(S) USED ";
80 INPUT MDS
90 DLOG #2 FROM #1,MDS
100 DETACH #1
110 CLOSE #2
```

Fig. 2. A simple TEK SPS BASIC data logging program for the 7612D. The program uses the DLOG command, which is part of the optional 7612D Commands Package for TEK SPS BASIC V02 and V02XM.

The program starts by loading the high-level instrument driver (INS) and attaching the instrument. It assumes that the 7612D's primary address is set to one and the mainframe secondary address is set to zero. It also assumes that a 7A16P plug-in is installed in both compartments.

Next, the program asks for the name of the file you want data stored in. Line 80 opens the output file for write. The default for the OPEN command allocates one-half the largest free space on the disk to the file. Unused space is returned to free when the file is closed.

The DLOG command writes data in integer array format. As a result, each block of space on the disk can hold 256 samples. A few extra blocks are required for file overhead. Keep this restriction in mind when you specify the number of waveforms to be logged, because trying to log too many waveforms into a file will cause a P-11 error (output file full). If this becomes a problem, cancel files to free some space, specify fewer waveforms for logging, or use the INTO option on the OPEN command to allocate more space to the file. (Refer to the TEK SPS BASIC System Software Manual for more information).

Lines 50-70 get the command string that will be sent to the 7612D to initiate acquisition. The

response must be a valid ALT or REP command. The syntax of the ALT and REP commands is—

ALT n

where: **n** is the number of times to execute the alternate sequence and

REP n,c

where: **n** is the number of times to execute the repeat sequence, **c** is the channel(s) used (A or B or A,B or B,A).

In ALT mode, the 7612D alternately acquires data from each channel. Data is read from one channel while the other channel is acquiring data. In REP mode, the instrument repetitively arms and reads the specified channels. In each case, you specify the number of waveforms acquired in the command string. The choice of which mode to use depends mostly on the timing of the signals to be logged. More information is provided in the box "To repeat or alternate—that is the question."

With all the preparations made, line 90 initiates the data logging. The DLOG command checks the syntax of the command string you entered and, if it is valid, sends it to the 7612D. Then the instrument is addressed to talk. As acquisitions complete, data is sent across the GPIB, through the computer, and is written on the disk. Direct Memory Access (DMA) is used for data transfer from the 7612D to the computer and from the computer to the hard disk (DK or DL). DMA is not used for floppy disks (DX) or magnetic tape (MT) output.

When the data logging is complete, the instrument is detached and the output file closed.

Simplest isn't always best

This is data logging in its simplest form. The problem with this approach is that a month later, when you go back to retrieve and analyze the data, several things are missing. For example: What were the scale factors? Where was the zero reference? At what time was the waveform acquired?

With a fully programmable digitizer linked to a powerful minicomputer and software, the solution is simple—write a program to acquire the instrument settings, zero reference, and acquisition time and write this information on the disk along with the logged data. Figure 3 shows a listing of such a program.


```

10 DIM CHS(1),PS(1)
20 CHS(0)="A" CHS(1)="B" PS(0)="L" PS(1)="R"
30 SA=0:BL=0
40 PRINT "PLEASE ENTER DATE (DAY-MON-YR) ";
50 INPUT DS:SETDATE DS
60 PRINT "PLEASE ENTER TIME (HH:MM:SS) ";
70 INPUT TS:SETTIME TS
80 PAGE
90 DELETE SF$,SF,VAS,ZR
100 REM ** GET MODE, CHANNELS, NO. OF WAVEFORMS AND FILENAME **
110 PRINT 'DO YOU WANT TO USE "ALT" OR "REP" MODE TO ACQUIRE DATA?';
120 INPUT MDS
130 IF MDS<>"ALT" THEN IF MDS<>"REP" THEN 110
140 IF MDS="ALT" THEN 180
150 PRINT "ENTER CHANNEL(S) TO ACQUIRE DATA FROM EA OR B OR A,B: ";
160 INPUT CS
170 IF CS<>"A" THEN IF CS<>"B" THEN IF CS<>"A,B" THEN 180
180 PRINT "ENTER NO. OF WAVEFORMS (MUST BE EVEN FOR ALT OR REP A,B) ";
190 INPUT NU
200 IF NU<1 THEN 180
210 IF MDS="ALT" THEN IF ITP(NU/2)<NU/2 THEN 180
220 IF MDS="REP" THEN IF CS="A,B" THEN IF ITP(NU/2)<NU/2 THEN 180
230 PRINT "ENTER FILE NAME FOR DATA LOGGING OUTPUT: ";
240 INPUT FS
250 CANCEL DLO:FS
260 LOAD "INS
270 INTEGER NB(1),NR(1),RL(1)
280 ATTACH $1 AS IN$1,0:UITH 1,2 @0
290 REM ** GET NO. OF RECORDS, RECORD LENGTH AND BREAKPOINTS **
300 IF MDS="REP" THEN IF CS="B" THEN 360
310 GET NR(0),RL(0) FROM $1,"THBS A;REC?"
320 GET NB(0) FROM $1,"NBPT?"
330 DELETE BA:DIM BA(NB(0)*2-1)
340 GET BA FROM $1,"SBPT?"
350 IF MDS="REP" THEN IF CS="A" THEN 410
360 GET NR(1),RL(1) FROM $1,"THBS B;REC?"
370 GET NB(1) FROM $1,"NBPT?"
380 DELETE BB:DIM BB(NB(1)*2-1)
390 GET BB FROM $1,"SBPT?"
400 REM ** CALCULATE FILE SIZE AND OPEN FILE **
410 NI=NU
420 IF MDS="REP" THEN IF CS<>"A,B" THEN 440
430 NI=NU/2
440 BL=(NIX(RL(0)*NR(0)+RL(1)*NR(1))/256)+4
450 OPEN $2 AS DLO:FS FOR WRITE INTO BL
460 REM ** WRITE NO. OF WAVEFORMS, MODE, CHANNEL(S) **
470 REM ** NO. OF RECORDS, RECORD LENGTH, AND BREAKPOINTS **
480 WRITE $2,NU,MDS,CS
490 IF MDS="REP" THEN IF CS="B" THEN 510
500 WRITE $2,NR(0),RL(0),NB(0),BA
510 IF MDS="REP" THEN IF CS="A" THEN 540
520 WRITE $2,NR(1),RL(1),NB(1),BB
530 REM ** ACQUIRE ZERO REFERENCE AND SCALE FACTORS **
540 B=0:E=1
550 IF MDS="REP" THEN IF CS="A" THEN E=0
560 IF MDS="REP" THEN IF CS="B" THEN B=1
570 FOR J=B TO E
580 GET CP$ FROM $1,J:SA+1,"CPL?"
590 PUT "CPL GND" INTO $1,J:SA+1
600 PUT "ARM ",CHS(J) INTO $1
610 DELETE A
620 FOR I=1 TO NR(J)
630 WAIT 25
640 PUT "INTRIG" INTO $1
650 NEXT I
660 JB="INTRIG;READ "&CHS(J)
670 GET A FROM $1,JB
680 ZR=REA(A)
690 PUT CP$ INTO $1,J:SA+1
700 JB="US"&PS(J)&"1?"
710 GET SF$,SF,VAS FROM $1,JB
720 VAS=SEQ(VAS,1,1)
730 WRITE $2,ZR,SF,VAS
740 NEXT J
750 REM ** GET TIME AND DATE AND BEGIN DATA LOGGING **
760 DELETE A,NR,RL,NB,ZR,SF,BA,BB
770 DATE DS:TIME TS
780 WRITE $2,DS,TS
790 RELEASE AUTO
800 IF MDS="REP" THEN LS="REP "&STR(NI)&","&CS
810 IF MDS="ALT" THEN LS="ALT "&STR(NI)
820 DLOG $2 FROM $1,LS
830 CLOSE $2:DETACH $1
840 GOTO 80

```

Fig. 3. Adding program lines to acquire the zero reference, scale factors, instrument settings, time, and date makes the simple data logging program of Fig. 2 more useful.

The program gets under way in lines 10-80 by initializing some variables, setting the current time and date, and paging (erasing) the terminal screen.

Line 100 asks which mode (ALTErnatE or REPeat) you want to use to log data. If the response is not ALT or REP, line 120 branches back to the question and it is asked again. If REP mode is selected, line 140 asks which channel(s) you want to acquire data from. The choices are A, B, or A,B. Line 160 checks the response for validity. If ALT mode was selected, lines 140-160 are skipped, since no channel specification is required.

Next, line 170 asks how many waveforms you want to log. If you selected ALT mode or REP A,B, you must enter an even number, because the waveforms are acquired in pairs (one from each channel). Line 180 gets the response and lines 190-210 check it for validity. Then, line 220 asks what file name you want the data stored under. If a file of the same name already exists, it is cancelled.

Lines 250-270 load the high-level instrument driver (INS), dimension integer arrays, and attach the 7612D and plug-ins as ILUN (Instrument Logical Unit Number) 1.

To avoid the problem of overflowing the data file, this program calculates the amount of space required to log the requested number of waveforms. The amount of space depends on the number of waveforms, number of records, and record lengths. If ALT or REP A,B mode was selected, the number of records and record length of both channels will be acquired. Otherwise, the program only acquires the number and length of records for the selected channel. The number of breakpoints, breakpoint locations, and sampling intervals are also acquired during this step.

Line 290 checks the mode and channel responses to see if channel A will be used. If not (REP B was specified), the queries for channel A are skipped. Otherwise, line 300 gets the number and length of records.

Line 310 gets the number of breakpoints and line 320 dimensions an array to hold the breakpoint locations and sampling intervals. The SBPT? query returns two values for each breakpoint—a breakpoint location (sample number) and a sampling interval, so the array is dimensioned to two times the number of breakpoints.

From the number and length of records information, line 430 calculates the minimum number of blocks required to store the data. Since each block can hold 256 data points in integer array format, the number of blocks is:

$$\text{# of Blocks} = \frac{\text{No. of Wfms} * (\text{A Total Rec. Length} + \text{B Total Rec. Length})}{256} + 4$$

$$\text{Total Rec. Length} = \text{Number of Records} * \text{Record Length}$$

The number of waveforms is divided by two in line 420 if REP A,B or ALT mode is specified since each

To repeat or alternate—that is the question

The 7612D offers two acquisition modes designed to increase throughput during data logging or other repetitive acquisitions. One mode—the REPEAT mode—causes the 7612D to repeat an ARM and READ sequence for a specified number of times (or indefinitely, until a device clear message is received). In the REP mode, data is acquired from the specified channels (A, B, or A and B) and read out after each acquisition. If REP A,B is used, both channels are acquired before reading any data out. Then both channels of data are transmitted.

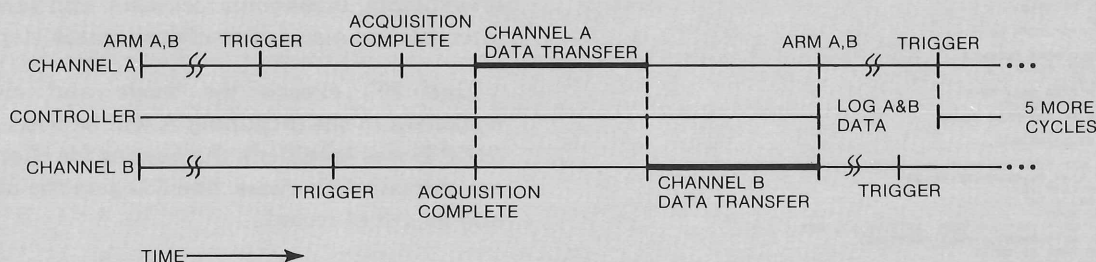
The other mode (ALTERNATE) causes the 7612D to repeat an ARM and READ sequence for both channels, and to transmit data from one channel while acquiring data in the other. When data acquisition is completed in channel A, channel B begins its acquisition and data from channel A is

transmitted on the bus. When channel B completes its acquisition, channel A starts another acquisition and channel B data is transmitted.

The question is—which mode is best for a particular application? To answer that, a clear understanding of the timing in both modes is important. Figure 1a illustrates the timing of a REP 6,A,B command.

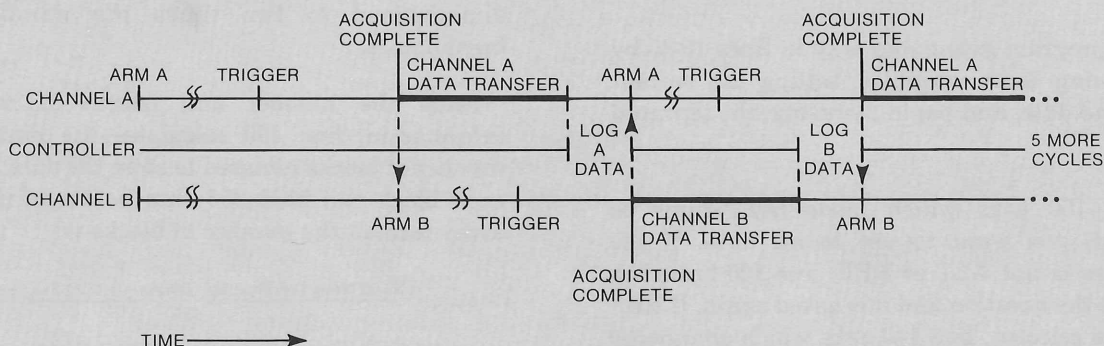
Notice that in each acquisition channels A and B are acquired with minimal delay. They can even be set-up with post-trigger mode to be concatenated into a single extended-length record. However, at the end of the acquisition, both channels of data are transmitted and acquisition stops until the data transmission completes and

REP 6,A,B



a. The REPEAT command waits until acquisition in both channels completes before transmitting data. If a single channel is used (REP A or REP B), data is transmitted as soon as the acquisition is complete—the unused channel is not affected.

ALT 6



b. The ALTERNATE command acquires data in one channel while transmitting data from the other. If the acquisition time is longer than the data transfer and logging time, digitizing can be nearly continuous.

Fig. 1. ALT and REP timing.

the data is logged to a peripheral device. Also, if pre-trigger mode is used (even when set for PRE 0), a full record of pre-trigger data is acquired before the channel is ready to be triggered.

Figure 1b shows the timing for an equivalent ALT 6 command. Notice that the same trigger and acquisition restrictions apply, but when channel B begins its acquisition, channel A's data can be read out. When channel B finishes, data from channel A can be read out. If the acquisition time is long compared to the time required to transfer and log the data (i.e. a slow sampling interval is used), digitizing can be nearly continuous. It's important to remember that the time required to write the data on the peripheral device must also be included, since the controller can't read more data until the previous data is written out.

In applications where continuous digitizing is important, ALT mode may be your best choice. But remember that continuous digitizing requires

that the total acquisition time be longer than the data transfer and logging time. Also, remember that trigger restrictions still apply. Faster controllers and peripherals allow continuous digitizing at a higher rate.

If the application requires long records (up to 4096 points) or logging of two time-related signals, REP mode is probably the best choice. Again, remember the restrictions of pre-trigger acquisition and triggering.

Using the flexible triggering of the 7612D, you can set-up the channels for most any type of acquisition. For example, to acquire two time-related signals, both channels should be set for the same mode (PRE or POST). They may be offset in time by choosing different amounts of pre- or post-trigger. To append the two channels, set one for POST 8 and the other for POST 2048 (assuming a record length of 2048) and use the same trigger.



continued from page 5

cycle acquires two waveforms—one from channel A and one from channel B. If REP A or REP B modes are specified, the number of waveforms is unchanged.

Four extra blocks are added to the calculated number of blocks for file overhead and the settings and zero reference information that is also written in the file. Line 440 opens the output file as PLUN (Peripheral Logical Unit Number) #2 into the calculated number of blocks using the name you specified.

The first thing written to the output file is the number of waveforms, the acquisition mode (REP or ALT), and the channel(s) used. This is followed in line 490 and 510 by the number of records, record length, number of breakpoints and the breakpoint locations and sampling intervals for the selected channel(s).

After the settings information is written out, the zero reference is acquired for each channel. Figure 4 shows a flow chart of the process. First, the current plug-in coupling is saved in CP\$. Then, the coupling is set to ground and the time base is armed. One Manual TRIGger (MTRIG) command is sent for each record in the channel, since each record requires a separate trigger. Appropriate WAITs are also included to allow each record to complete before sending the next MTRIG.

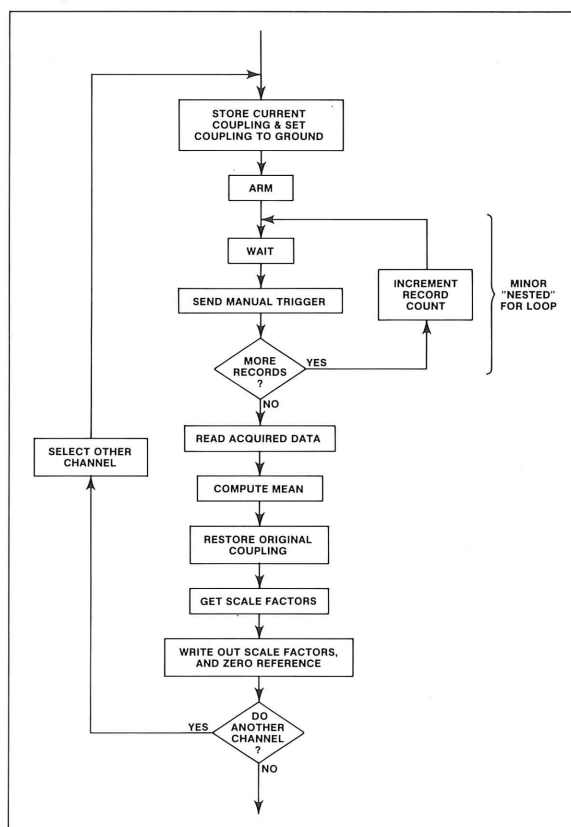


Fig. 4. Acquiring the zero reference uses two nested FOR loops. The major loop is executed once for each channel used. The minor loop is executed once for each record in the channel.

Data logging...Part I

When the acquisition is complete, the data is read into the controller, averaged, and the result is stored in ZR. Then, the plug-in coupling is returned to its previous setting.

Two nested FOR loops in lines 560-730 implement this process. The major FOR loop is executed once for each channel used. In ALT or REP A,B modes, two passes are executed. In REP A or REP B modes, a single pass is executed. Lines 530-550 set-up the beginning and ending index variables for the loop. Lines 570-590 store the current plug-in coupling, set the coupling to ground, and ARM the time base. Then, the minor FOR loop in lines 610-640 begins sending MTRIG commands—one for each record. After each MTRIG, a WAIT statement delays execution until the record is complete. Then another MTRIG is sent for the next record. (For long sampling intervals, the WAIT value may have to be increased since it takes longer to acquire each record.)

When the zero reference acquisition is complete, lines 650-660 read the data and line 670 finds the mean (average) of it. This mean value represents the zero reference for data acquired later.


For convenience, the vertical scale factor information is also acquired as part of this loop. Lines 690-710 get the scale factors and units, and line 720 writes the zero reference, scale factor, and vertical units to the output file.

Just before initiating the acquisition, the number of waveforms (NW) is divided by two if REP A,B or ALT mode is selected. Since these modes acquire two waveforms per iteration, the count is divided by two before being used as a command argument. Then, the unused variables are deleted, the beginning acquisition time is written in the output file, and all auto-loaded commands are released. This frees all available memory for faster execution of the DLOG command.

When data logging is complete, the file is closed, and the instrument is detached.

The largest part of this "improved" program involves acquiring the scale factors, zero reference, settings, and time information. The data logging itself is still just one statement. Data is passed from the 7612D, through the computer and on to the disk without intervention.

Noise may add more steps

The two programs described here illustrate the process of data logging with TEK SPS BASIC and the 7612D. They operate well in most environments where noise in the input signal is not a major problem and where the 7612D's flexible analog triggering system is sufficient to detect valid data. In some applications, noise is a more significant problem. Also, more sophisticated techniques may be required to differentiate between noise and valid data. The second part of this article series examines several approaches for detecting valid events and reducing logged noise. 

*By Mark Tilden,
HANDSHAKE Staff,
with grateful acknowledgment to:
Dr. Richard Couch,
Dr. Dale Bibee,
and Dr. Michael Fehler,
of the Geophysics Group,
School of Oceanography,
Oregon State University,
for providing valuable background
information and consultation.*

*We would also like to thank
Mr. Norm Smale,
of the Oregon Museum of Science & Industry,
for his suggestions and direction.*

Data Logging...

Capturing the unpredictable

Part II—Event detection

Part I of this article examined some simple approaches to data logging with the 7612D and TEK SPS BASIC. The programs presented in Part I logged data directly to mass storage (disk) using the optional DLOG command for TEK SPS BASIC. The controller simply passed the data from the GPIB to the disk as illustrated in Fig. 1. Data acquisition was initiated when the 7612D received a trigger either from its internal trigger circuit or an externally supplied trigger signal.

The 7612D's flexible triggering is adequate for many applications. However, in noisy environments or where data must be logged selectively, the analog trigger may not be sufficient. The instrument may trigger on noise or log a considerable amount of unwanted data. This article, Part II of the series, describes some solutions to this problem using the signal processing power of TEK SPS BASIC.

What is an event?

It's usually easy to pick out valid information from noise on an oscilloscope display—you know basically what you're looking for. But when the data logging system is on the side of an active volcano or in a room full of machinery, you can't always be there to pick out valid data from noise. And the analog trigger circuits on the digitizer may not be selective enough to do it either.

The solution? Acquire the data, read it into the controller, and let the signal processing power of

TEK SPS BASIC decide if it is valid data. If it is, log it. If not, ignore it and acquire more data.

This process of distinguishing valid data from noise or other unwanted signals is called "event detection." An "event" is any valid data captured in the process. In some cases, such as in earthquake seismology, an event may be a transient signal. In other applications, such as monitoring machinery vibration, the event may be a random or repetitive signal.

There are a multitude of techniques for digitally detecting events. The choice of which one to use and how to apply it depends on the type of data you're looking for. No technique is fool proof, but knowing some basic information about the input signal makes the choice easier. For example, what is the frequency range of the signal? What is its approximate amplitude range? Is the signal repetitive, transient, or random? Also, an estimate of the noise that will appear with the signal will help.

The techniques discussed here are just a few of the many that exist. Though they may not be suitable for every application, they provide a basis for developing an event detection routine that best suits your needs.

Averaging cleans up the data

If the input signal is repetitive, signal averaging may be the answer. Since noise sources are often

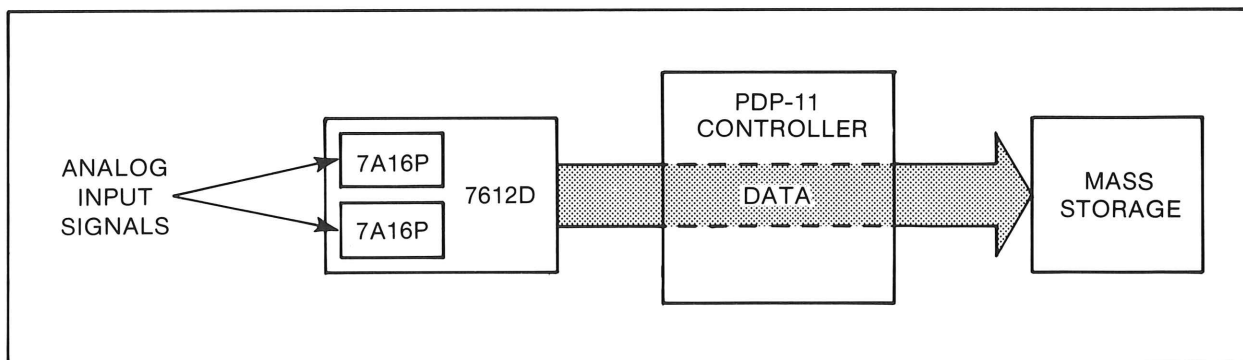


Fig. 1. In the simplest case, the controller simply acts as a pipeline between the 7612D and the mass storage device. The 7612D's analog trigger provides event detection.

Data logging...Part II

random with respect to the data, averaging can keep a single record of noise from being logged and can reduce the effect of noise on the logged data. As a result, a record of data triggered by a noise spike will not be logged, but averaged in with records of valid data (Fig. 2). The more averages per logged waveform, the less random noise that appears in the data.

The optional 7612D Commands Package for TEK SPS BASIC V02 or V02XM contains a DAVG command that automatically acquires and averages data from a 7612D. This command provides a convenient, fast way of acquiring and averaging the data in preparation for data logging. Figure 3 shows a simple program that uses DAVG to acquire, average, and log data on the disk.

After loading the high-level instrument driver and attaching the instrument, the program asks for the data file name. If an old file exists with the same name, line 50 cancels it. Line 60 opens the new file for write.

Then, lines 70-190 get the acquisition mode, channel(s), number of times to average each waveform, and the number of waveforms to log. Lines 180 and 190 build the REP or ALT command string for the 7612D from this information.

```
10 LOAD *INS
20 ATTACH #1 AS INS1,0:WITH 1,2,60
30 PRINT "ENTER FILENAME FOR DATA LOGGING OUTPUT: ";
40 INPUT FS
50 CANCEL DL0:FS
60 OPEN #2 AS DL0:FS FOR WRITE
70 PRINT "ENTER ACQUISITION MODE (ALT OR REP): ";
80 INPUT MDS
90 IF MDS<>"ALT" THEN IF MDS<>"REP" THEN 70
100 IF MDS="ALT" THEN 140
110 PRINT "ENTER CHANNEL(S) TO ACQUIRE DATA FROM (A OR B OR A,B): ";
120 INPUT CS
130 IF CS<>"A" THEN IF CS<>"B" THEN IF CS<>"A,B" THEN 110
140 PRINT "ENTER NO. OF TIMES TO AVERAGE EACH WAVEFORM: ";
150 INPUT AU
160 PRINT "ENTER NO. OF WAVEFORMS (MUST BE EVEN FOR REP A,B OR ALT): ";
170 INPUT NW
180 IF MDS="REP" THEN IF CS<>"A,B" THEN 210
190 IF NW/2<>ITP(NW/2) THEN 160
200 NW=NW/2
210 JS=MDS* " &STR(AU)
220 IF MDS="REP" THEN JS=JS* " &CS
230 FOR I=1 TO NW
240 DELETE A,B
250 IF MDS="REP" THEN IF MDS<>"A,B" THEN 280
260 DAVG A,B FROM #1,JS
270 GOTO 290
280 DAVG A FROM #1,JS
290 WRITE #2,A
300 IF MDS="REP" THEN IF CS<>"A,B" THEN 320
310 WRITE #2,B
320 NEXT I
330 DETACH #1\CLOSE #2
```

Fig. 3. A TEK SPS BASIC program to acquire, average, and log data from the 7612D. Signal averaging keeps random noise from being logged and reduces noise in logged data.

Lines 230-320 form a FOR loop that acquires and averages the waveforms and writes the averaged results on the disk. If REP A or REP B modes were selected, each pass through the loop writes one averaged waveform on the disk. If REP A,B or ALT modes were selected, line 200 divides the number of waveforms (NW) by two, since each pass through the loop writes two waveforms—one from channel A and one from channel B. When the specified number of averaged waveforms have been written, line 330 detaches the 7612D and

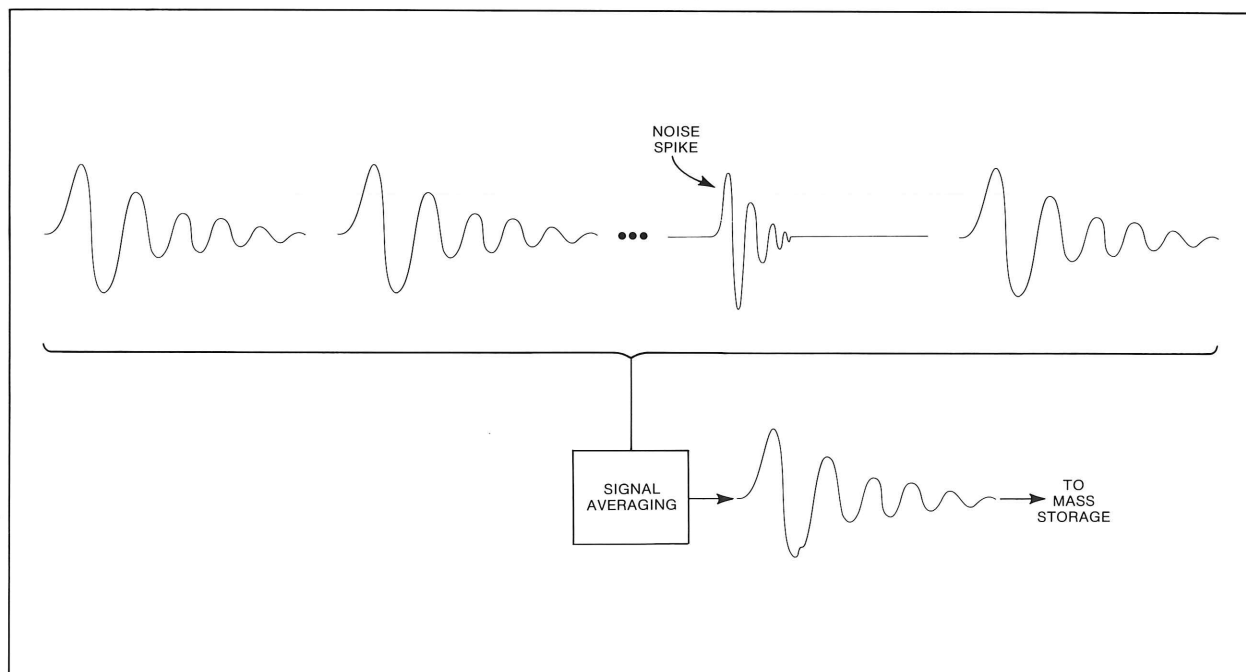


Fig. 2. Signal averaging repetitive events prevents random noise from being logged as an event. Instead, the noise is averaged along with valid data, reducing its effect.

closes the output file.

Looking for maximums

The signal averaging approach works well when you can afford the time required to acquire and average the data and when the input signal is repetitive. But what about the times when the signal occurs only once? Signal averaging is ruled out because a repetitive signal is required.

One solution is a simple amplitude check on the acquired data. If the amplitude of the noise is consistently lower than the signal amplitude, data can be discarded if its peak-to-peak amplitude is below a certain threshold. The MIN and MAX functions in TEK SPS BASIC can be used to compute the peak-to-peak amplitude. This technique only duplicates the analog trigger on the 7612D. In some environments, that's just not enough.

Unfortunately, noise spikes are often narrow high-amplitude pulses that will pass a simple amplitude test (or trigger an analog trigger circuit). To reduce the possibility of noise signals slipping through, the acquired waveform can be segmented into several parts and the peak-to-peak amplitude of each segment computed. If a certain number of the segments have amplitudes above the threshold, the waveform is considered valid

data. Alternately, the peak-to-peak amplitudes for all the segments can be averaged and the result compared to a threshold value (Fig. 4).

A program that uses the segment averaging technique is shown in Fig. 5. Lines 10-90 get user input and open the output file as in the previous programs. Line 110 defines the number of segments and the threshold value. Notice that the waveform is divided into equal segments, so the number of segments should be chosen to divide evenly into the record length. The threshold value

```
10 PRINT "WHICH CHANNEL DO YOU WANT TO USE (A OR B) ";
20 INPUT CS
30 IF CS<>"A" THEN IF CS<>"B" THEN 10
40 PRINT "ENTER FILE NAME FOR DATA LOGGING OUTPUT: ";
50 INPUT FS
60 CANCEL DL0:FS
70 OPEN #2 AS DL0:FS FOR WRITE
80 REM ** N=NUMBER OF SEGMENTS TH=THRESHOLD **
90 N=32\TH=30
100 DELETE PP\DIM PP(N-1)
110 LOAD "INS
120 ATTACH #1 AS INS1,0:WITH 1,2 @0
130 DELETE A
140 GET A FROM #1,"ARM "&CS&";READ "&CS
150 L=SIZ(A)
160 L=L\N
170 FOR I=0 TO N-1
180 B=I*L
190 E=((I+1)*L)-1
200 PP(I)=MAX(A(B:E))-MIN(A(B:E))
210 NEXT I
220 AU=MEA(PP)
230 IF AU>TH THEN WRITE #2,A
```

Fig. 5. This program divides the waveform into 32 equal segments and computes the peak-to-peak amplitude of each segment. These amplitudes are averaged and the result is compared to the threshold (TH). If it exceeds the threshold, the data is considered valid and is logged.

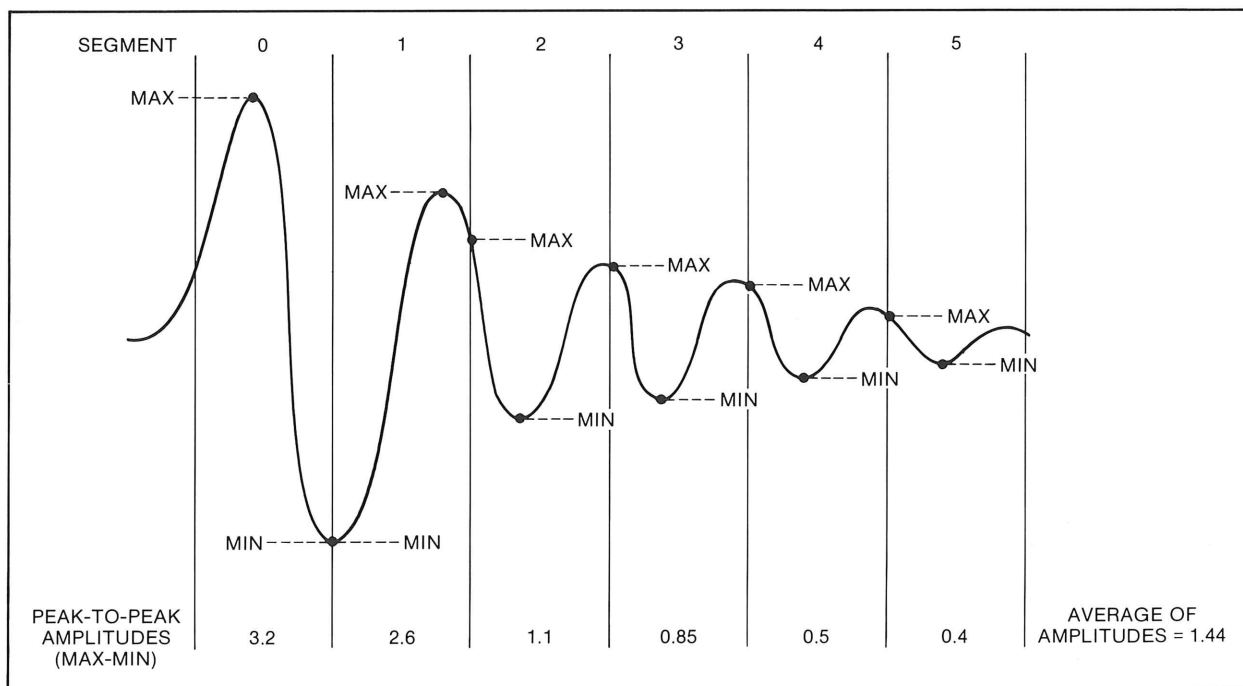


Fig. 4. Dividing the waveform into several equal segments and averaging the amplitudes from the segments reduces the possibility of noise spikes being detected as events.

Data logging...Part II

depends on the expected signal and noise levels.

Line 120 dimensions an array (PP) to hold the peak-to-peak amplitudes for each segment of the waveform. Then the instrument is attached and the waveform acquired from the specified channel. Lines 170 and 180 compute the size (number of samples) of each segment. The FOR loop, starting in line 190, computes the amplitude in each segment. First, line 200 computes the beginning position of the segment and line 210 computes the ending position of the segment. The amplitude is computed in line 220 by subtracting the minimum value in the segment from the maximum value in the segment.

The loop is repeated until the amplitude for each segment is computed. Then, the amplitudes are averaged and line 250 compares the result with the threshold value (TH). If the result is greater than the threshold, the waveform is considered valid data, and it is logged. Otherwise, it is ignored. The loop is repeated until the requested number of waveforms have been logged.

The number of segments and the threshold value can be adjusted to best fit the type of signals and noise expected. Increasing the number of segments tends to filter low-frequency noise by looking at amplitude variations in smaller time windows (segments). (It also takes more time, since more amplitude values are calculated and averaged.) Increasing the threshold requires a larger amplitude variation within each segment to detect an event.

More averages

A second averaging technique that can be used where noise is relatively constant is the long-term vs. short-term average. This technique compares a long-term average computed over several waveforms to a short-term average computed on the current waveform. The long-term average represents the average noise level, and the short-term average represents the level of the signal just acquired.

First, the waveforms are acquired and shifted to center around zero by subtracting the mean of the waveform from each point. Then, the absolute value of the waveform is computed and the results averaged. This average is proportional to the energy of the waveform—not just its instantaneous amplitude. As a result, it is usually a better indicator of valid events than the simple amplitude values used in the previous example. Unfortunately, it's also more time consuming.

The averages computed for each waveform are stored in a table of averages (Fig. 6). When a new waveform is acquired, the average of that waveform is computed and compared to a long-term average. This long-term average is the average of the averages stored in the table, and it represents the base noise level. If the short-term average exceeds the long-term average by some defined amount, the waveform is considered a valid event. If not, the average of this waveform is written in the next position in the average table. The average pointer points to this position.

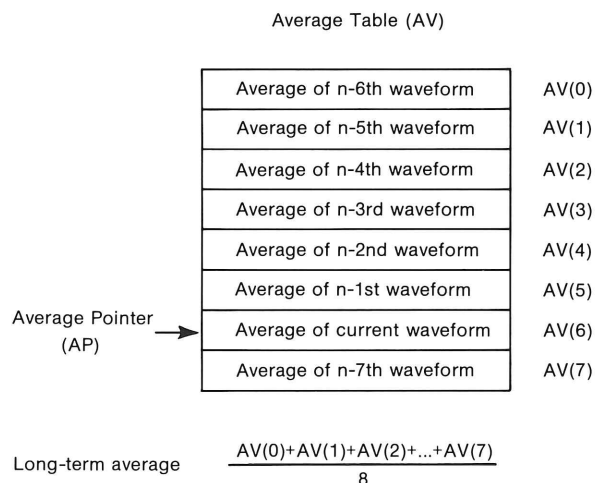


Fig. 6. The average table stores the averages of the last eight "non-event" waveforms. The long-term average is computed by averaging the averages in the table. The average pointer (AP) points to the location for the average of the current waveform.

Notice that once the table is filled the first time, old averages are overwritten by new ones. The average pointer is incremented each time, so that the table always contains the averages of the last eight non-event waveforms.

A TEK SPS BASIC program that implements this technique is shown in Fig. 7. The first 18 lines (10-180) perform the initialization and open the output file as before. Then, line 190 sends the 7612D an ALT 0 command. This tells the 7612D to continuously repeat the ALTeRnate sequence until it receives a device clear message. The GET statement in line 200 reads the waveform data from the 7612D into array A. The time and date are then stored in T\$ and D\$ respectively. This will be written in the file with the waveform data and reported on the terminal screen to identify when valid data was acquired.

```

10 LOAD *INS
20 ATTACH $1 AS INS1,0:WITH 1,2 @0
30 PRINT "ENTER THE FILE NAME FOR DATA LOGGING OUTPUT: ";
40 INPUT FS
50 PRINT "HOW MANY WAVEFORMS DO YOU WANT TO LOG (MUST BE EVEN NO.): ";
60 INPUT NU
70 IF ITP(NU/2)<NU/2 THEN 50
80 INTEGER NR(1),RL(1)
90 DIM AV(7)
100 AP=0:FL=0:TH=4:UC=1
110 GET NR(0),RL(0) FROM $1,"TMS A;REC?"
120 GET NR(1),RL(1) FROM $1,"TMS B;REC?"
130 IF RL(0)=RL(1) THEN 160
140 PRINT "CHANNEL A AND B RECORD LENGTHS MUST BE EQUAL"
150 GOTO 420
160 BL=(NU/2)*((RL(0)*NR(0))+(RL(1)*NR(1)))/128)+4
170 CANCEL DL0:FS
180 OPEN $2 AS DL0:FS FOR WRITE INTO BL
190 PUT "ALT 0" INTO $1
200 GET A FROM $1
210 TIME TS:DATE DS
220 A=(A-MEA(A))
230 DELETE B:DIM B(SIZ(A)-1)
240 B=ABS(A)
250 M0=MEA(B)
260 IF FL=0 THEN 290
270 LA=MEA(AV)
280 IF M0>LA*TH THEN 360:REM IF TRUE, WE HAVE AN EVENT!
290 AV(AP)=M0
300 AP=AP+1
310 IF AP<8 THEN 200
320 AP=0
330 IF FL=0 THEN PRINT "EVENT DETECTION ENABLED"
340 FL=1
350 GOTO 200
360 PRINT "EVENT NUMBER ";UC;" LOGGED AT ";DS;" ";TS
370 WRITE $2,DS,TS,A
380 UC=UC+1
390 IF UC<NU THEN 200
400 DEVICLEAR $1
410 CLOSE $2
420 END

```

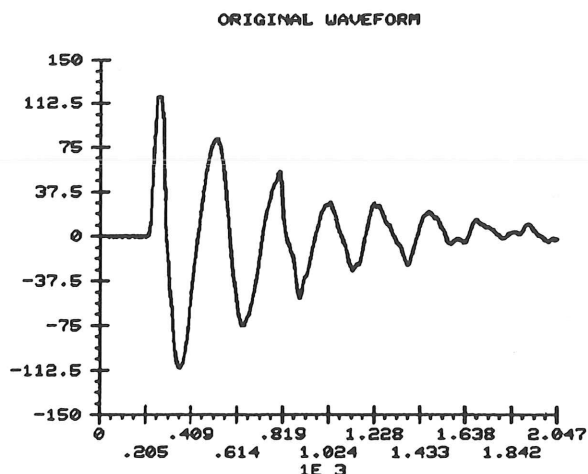
Fig. 7. A TEK SPS BASIC program that implements long-term vs. short-term average event detection.

Line 220 finds the mean of the waveform data and subtracts that value from each point to center the data around zero. If the DC level of the signal is important, a true zero reference should be acquired first and that value subtracted from the signal. (Part I of this article series discussed zero reference acquisition.)

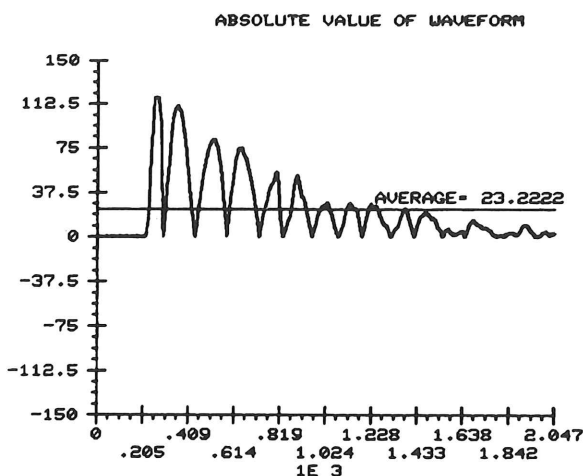
The computation performed from here on modifies the waveform data, so a separate floating-point array is dimensioned in line 230. All computations are performed in this second array, while the original data remains in array A. Line 250 begins the processing by computing the absolute value of the array. This effectively "folds" the negative half of the waveform data over and makes it positive. Then the mean of this "folded" waveform is computed, and the result is stored in M0 (Fig. 8).

Since this technique depends on comparing the average of a waveform to the average of the previous eight waveforms, the first eight waveforms acquired are not passed through the event detection routine. Instead, the first eight averages are just stored in the table. They will be averaged to compute the first long-term average. When the table is full, event detection begins. The flag variable, FL, is cleared (zero) until the table is full. Line 260 branches around the event detection when FL is equal to zero.

The average table is stored in array AV and the next available location in AV is pointed to by AP



a. The original waveform, shifted to center around zero.



b. The absolute value function "folds" the negative half of the waveform over and makes it positive. The average of the resulting waveform is proportional to the area under the curve.

Fig. 8. The program in Fig. 7 first shifts the acquired waveform to center around zero. Then the absolute value is computed and the result averaged. This average is a better indicator of the waveform's energy than the simple amplitude test.

(Average Pointer). Line 290 stores the average of the current waveform in AV, and line 300 increments the AP. When AP reaches eight, it is reset to zero (to point to the top of the table again). If FL is zero, the first eight acquisitions have just been completed, so line 330 rings the terminal bell and prints a message to tell the operator that event detection has been enabled. Line 340 sets the flag to one and line 350 branches back to get another waveform.

Data logging...Part II

When event detection is enabled, (FL=1) the condition in the IF statement of line 260 fails. Program control passes to the next line which computes the long-term average. This average is computed by taking the mean of the averages stored in the AV table. If the short-term average stored in M0 is greater than the short term average times the threshold (TH), the waveform is considered a valid event, and control is passed to line 360.

Line 360 prints a message on the terminal screen telling the operator that an event has been detected and the time at which it was acquired. Then, line 370 writes the time, date, and the waveform data into the output file. The variable WC counts the number of waveforms logged. When it reaches the specified number of waveforms, a device clear message is sent to the 7612D to abort the ALternate sequence, and the output file is closed.

Notice that averages of valid events are **not** entered into the average table (AV). This keeps the long-term average from growing too large during a long string of events, which could prematurely stop data logging. The updating of the average table begins again on the first non-event waveform.

Looking for a match

Each of the techniques discussed this far provides some level of event detection based on signal versus noise amplitude. Another more sophisticated technique is available with the signal processing power of TEK SPS BASIC. The correlation command (CORR) in TEK SPS BASIC compares two waveforms and produces an array that represents the degree of matching between the two. One of these waveforms is usually the standard or reference waveform. The other is the acquired signal. Correlation can detect similarity between the two signals, even when they are buried in significant amounts of noise.

For sophisticated event detection where high noise levels are a problem or data logging must be very selective, correlation may be a good choice. But, it doesn't come without cost. Correlation requires about six times the amount of memory required to store one waveform. Part of this memory is used to store a reference waveform that will be the standard for comparing acquired waveforms. The rest is used to hold the double-length result array and the scratch areas.


Correlation also requires a significant amount of execution time—about 20 seconds for a 512-point waveform on a PDP 11/03 or equivalent. Execution will be considerably faster on a PDP 11/23 or PDP 11/34.

Do you have the time?

The event detection techniques covered here are but a few of the many that exist for various applications. The choice of which to use depends on the type of data, the amount of noise, and how much time you have to decide whether a signal is a valid event. The more noise in the environment, and the more selective you want to be, the more sophisticated and time-consuming your event detection scheme will have to be.

You will also need to decide whether data logging must be continuous or not. If event detection takes too much time, the system may get bogged down with processing data, and the digitizer may have to wait with acquired data. As a result, gaps in sampling may occur.

If the extra mass storage space is available, you may want to consider logging data directly to the peripheral device and performing the event detection when the data is read back. If speed is critical, the cost of the extra mass storage can be made up by the significant time savings.

The moral of the story is: look carefully at your input signal and noise and then choose the event detection that best fits your needs, keeping in mind the time constraints. You may have to experiment a little to find the best technique for your application. The programs and ideas presented here will provide a starting point for that investigation. 

By Mark Tilden,
HANDSHAKE Staff,

with grateful acknowledgment to:
Dr. Dale Bibee and Dr. Michael Fehler,
of the Department of Geophysics,
School of Oceanography,
Oregon State University,
for their valuable assistance
and consultation.

Data Logging...

Capturing the unpredictable

Part III—Retrieving logged data

The first two parts of this article series examined several approaches to data logging and event detection. But data logging, no matter how sophisticated, is of little value if you can't retrieve the data after it is logged. Often, it is during this step that the major processing and analysis is done because there's usually plenty of time. You can read the data as slowly as you like and spend as much time as required processing each waveform.

This part of the series examines general techniques for retrieving logged data and provides some specific examples based on data logging routines from Part I of this article.

Know your data formats

The key to reading any logged data is a little foresight. It's important to know exactly how the data was written. For example, was only waveform data written, or was settings information also included? Was the time and date of acquisition written? In what order were these things written? Is the data written in ASCII strings, numeric arrays, or individual numeric variables?

The best tool for writing a program to read logged data is a listing of the program that wrote the data. It will provide specific information about the amount, type, and order of the data. Fortunately, some of the specifics are automatically handled with the advanced array processing and I/O capabilities of TEK SPS BASIC. In the simplest cases, you may not have to remember much of anything about the data.

For example, take the case of the simple data logging program described in Part I of this article series. A listing of that program is reproduced in Fig. 1a. This program logged only the data with no settings or other information added. Knowing only that the data is written in numeric array format by DLOG, a simple program, like the one shown in Fig. 1b can be written.

First, the program asks for the file name in which the data was stored. Line 30 opens the file for READ and assigns it to PLUN (Peripheral

```
10 LOAD *INS
20 ATTACH #1 AS INS1,0:WITH 1,2 @0
30 PRINT "ENTER FILENAME FOR DATA LOGGING OUTPUT: ";
40 INPUT FS
50 OPEN #2 AS DLO:FS FOR WRITE
60 PRINT "ENTER ACQUISITION MODE (ALT N OR REP N,C)"
70 PRINT " WHERE N=NO. OF REPETITIONS, C=CHANNEL(S) USED ";
80 INPUT MDS
90 DLOG #2 FROM #1,MDS
100 DETACH #1
110 CLOSE #2
```

a. A simple TEK SPS BASIC data logging program for the 7612D. The program uses the DLOG command, which is part of the optional 7612D Commands Package for TEK SPS BASIC V02 or V02XM.

```
10 PRINT "ENTER THE INPUT DATA FILE NAME: ";
20 INPUT FS
30 OPEN #2 AS DLO:FS FOR READ
40 EOF #2 GOTO 110
50 DELETE A
60 READ #2,A
70 PAGE
80 GRAPH A
90 SMOVE 0,0\PRINT "PRESS RETURN TO CONTINUE";\WAIT
100 GOTO 50
110 CLOSE #2
120 END
```

b. A simple program to read the data file produced by the data logging program of Fig. 1a.

Fig. 1. The most important tool for writing a program to read data logging files is the listing of the program that logged the data.

Logical Unit Number) 2. From this point on, the file is accessed by this PLUN rather than the file name. For the moment, skip over line 40, and we'll come back to it. Line 50 deletes array A for subsequent use by the READ command.

Next, the READ command takes over and reads the first waveform into array A. It isn't necessary to know exactly how many elements are in the array, because READ automatically takes care of that. READ uses some special information written in the file called "data descriptors" to determine how big the destination array needs to be to hold the data. Then it automatically dimensions the array to that size. These data descriptors are invisible to the user and are written in the file automatically by WRITE or DLOG so you don't have to worry about them.

When the data is read in, lines 70 and 80 page (erase) the terminal screen and graph the waveform. Then, line 90 moves the cursor to the bottom-left corner of the screen and prints a

Data logging...Part III

message. The program pauses until you press any key. (RETURN is usually used). When RETURN is pressed, execution continues, and line 100 sends the program back to line 50 to get another waveform.

This loop continues until the end of the data file. But then what? That's what line 40 is for. Since it's not known how many waveforms the file contains, line 40 sets up a condition that says "when the end of the file is reached (there are no more waveforms), go to line 110." When line 40 is executed, nothing much happens. But, when the last waveform is read, the end of file condition causes program control to be passed out of the loop to line 110, closing the input file and ending the program.

In this simple case, there's little left for you to remember. All you needed to know was that the file was written with DLOG or WRITE, and TEK SPS BASIC takes care of the rest automatically.

What you write is what you read

There is often a lot more than just waveform data written in a data logging file. There may be settings, time and date information, number of waveforms, or other information. When these are added, the situation gets a little more complex. TEK SPS BASIC still handles the mechanics of data transfer for you, but you have to remember the order in which things are written.

The second data logging program in Part I (Fig. 3, "Data Logging... Capturing the unpredictable, Part I") provides a good example. The format of the data file it produces is illustrated in Fig 2.

First, the number of waveforms in the file is written as a single numeric variable. Therefore, the program that reads this data file must read it into a numeric variable or array. Next comes the acquisition mode, written as a string, followed by the channel(s) used, also written as a string.

Then, the number of records, record length, number of breakpoints, and breakpoint locations are written for each channel. However, they are only written for the channel(s) used. For example, if the channel(s) used information says that only channel B was used, the information from channel A is not recorded. The program that reads this data file must look at the channel information and read only the appropriate data.

Number of waveforms	(N)	
Acquisition Mode	(S)	
Channel(s) Used	(S)	
Channel A No. of Records	(NA)	Omitted if Channel A is not used
Channel A Record Length	(NA)	
Channel A No. of Breakpoints	(NA)	
Channel A Breakpoints	(NA)	Omitted if Channel B is not used
Channel B No. of Records	(NA)	
Channel B Record Length	(NA)	
Channel B No. of Breakpoints	(NA)	Omitted if Channel A is not used
Channel B Breakpoints	(NA)	
Channel A Zero Reference	(NA)	
Channel A Vertical Scale Factors	(NA)	Omitted if Channel B is not used
Channel A Vertical Units	(NA)	
Channel B Zero Reference	(NA)	
Channel B Vertical Scale Factors	(NA)	Omitted if Channel B is not used
Channel B Vertical Units	(NA)	
Date	(S)	
Time	(S)	
Waveform Data	(NA)	

N = Numeric variable

S = String

NA = Numeric array

Fig. 2. Format of the data file produced by the second data logging program in Part I.

The zero reference, vertical scale factors, and vertical units for each channel used are written next, followed by the time and date of acquisition. Finally, the waveform data is written in numeric array format—one array per waveform.

Figure 3 shows the listing of a TEK SPS BASIC program to read the data from this file, scale the waveform data based on the scale factors and units recorded in the file, and graph the resulting data.

Getting under way, the program initializes some variables, sets the current time and date, and sets the viewport for the waveform graph. (VIEWPORT defines the physical screen limits of the graph—where it is placed on the screen and how big it is.) Then, line 130 asks for the name of the data file you want to read. Line 150 opens the file for read, and line 160 reads the first three entries in the file—the number of waveforms, acquisition mode, and channel(s) used.

Using the channel and acquisition mode information just read, line 170 checks to see if channel A was used in acquisition. If not, the statements that read the channel A settings are

```

10 DIM CHS(1)
20 CHS(0) = "A" CHS(1) = "B"
30 SA = 0 BL = 0 F = 0
40 VIEWPORT 420,000,250,800
50 PRINT "PLEASE ENTER DATE (DAY-MON-YR): ";
60 INPUT D$ SETDATE D$
70 PRINT "PLEASE ENTER TIME (HH:MM:SS): ";
80 INPUT T$ SETTIME T$
90 PAGE
100 DELETE SF$,SF,VAS,ZR
110 DIM ZR(1),SF(1),VAS(1)
120 INTEGER NR(1),RL(1),NB(1)
130 PRINT "ENTER DATA FILENAME: ";
140 INPUT F$
150 OPEN #2 AS DLOIFS FOR READ
160 READ #2,NR,NB,CS
170 IF NR$="REP" THEN IF CS="B" THEN 210
180 DELETE BA
190 READ #2,NR(0),RL(0),NB(0),BA
200 IF NR$="REP" THEN IF CS="A" THEN 230
210 DELETE BB
220 READ #2,NR(1),RL(1),NB(1),BB
230 MB=NBS
240 IF NR$="REP" THEN MB=NBS " &CS
250 PRINT "NO. OF WAVEFORMS IN FILE: ";NR;" ACQUIRED IN ";MB;" MODE"
260 PRINT
270 PRINT "ENTER THE NUMBER OF THE WAVEFORM YOU WANT TO GET FROM THE"
280 PRINT "FILE, OR ENTER 'ALL' TO GET ALL WAVEFORMS."
290 PRINT
300 PRINT "WHICH WAVEFORM DO YOU WANT TO GET ";
310 INPUT W$
320 I=NR
330 IF W$="ALL" THEN 370
340 I=VAL(W$)
350 IF I<0 THEN 300
360 IF I>NR THEN 300
370 B=0 V=1
380 IF NR$="REP" THEN IF CS="A" THEN E=0
390 IF NR$="REP" THEN IF CS="B" THEN B=1
400 FOR J=B TO E
410 READ #2,ZR(J),SF(J),VAS(J)
420 NEXT J
430 READ #2,DS,TS
440 K=0
450 IF NR$="REP" THEN IF CS="B" THEN K=1
460 FOR J=1 TO I
470 IF NR$="REP" THEN IF CS<>"A,B" THEN 500
480 K=0
490 IF ITP(J/2)=J/2 THEN K=1
500 DELETE A DIM A((NR(K)*RL(K))-1)
510 READ #2,A
520 IF W$<>"ALL" THEN IF J<I THEN 720
530 PAGE
540 PRINT "WAVEFORM NUMBER ";J;" FROM CHANNEL ";CHS(K)
550 SETGR VIEWPORT
560 A=(A-ZR(K))*SF(K)/32
570 GRAPH A
580 SMOVE 350,420 PRINT VAS(K)
590 SMOVE 0,600
600 PRINT "ACQUIRED ON: ";DS
610 PRINT "STARTING AT: ";TS PRINT
620 PRINT "DATA CONTAINS ";NR(K); " RECORD";
630 IF NR(K)>1 THEN PRINT "S";PRINT
640 PRINT RL(K); " POINTS PER RECORD" PRINT
650 PRINT "BREAKPOINTS"
660 PRINT "SAMPLE NO. SAMPLE INTERVAL" PRINT
670 FOR N=0 TO NB(K)*2-1 STEP 2
680 IF K=0 THEN PRINT TAB(2);BA(N);TAB(11);BA(N+1)
690 IF K=1 THEN PRINT TAB(2);BB(N);TAB(11);BB(N+1)
700 NEXT N
710 SMOVE 0,0 PRINT "PRESS RETURN TO CONTINUE";WAIT
720 NEXT J
730 CLOSE #2
740 GOTO 90

```

Fig. 3. This program reads the data file produced by the data logging program shown in Fig. 2, Part I. It also scales the data and produces a display on the terminal screen.

skipped, and control is passed to line 210. Otherwise, the number of records, record length, number of breakpoints, breakpoint locations, and sampling intervals for channel A are read by line 190. Lines 200-220 do the same thing for channel B.

Notice that the breakpoint location/sampling interval array (BA for channel A, BB for channel B) are deleted just before the READ statements in lines 190 and 220. The READ command automatically dimensions the array to the appropriate size using the data descriptors written in the file. But, it will only auto-dimension if the array is not already dimensioned (i.e. a simple numeric variable is specified). Thus, deleting the array before reading it allows READ to dimension the array to the correct size without knowing that size before hand.

Using the information read from the file so far, the following message is printed on the terminal screen:

```

NO. OF WAVEFORMS IN FILE: 16 ACQUIRED IN REP A,B MODE

ENTER THE NUMBER OF THE WAVEFORM YOU WANT TO GET FROM THE
FILE OR ENTER 'ALL' TO GET ALL WAVEFORMS.

WHICH WAVEFORM DO YOU WANT TO GET?

```

The number of waveforms, acquisition mode, and channel(s) used are filled in the first line from the data in the file. Line 300 asks which waveform you want to retrieve and graph. You can specify a number from 1 to the number of waveforms in the file or "ALL" to successively read and graph each waveform in the file.

Line 310 gets your response, and line 320 sets the default number to the number of waveforms (NR). Since the response may be a string ("ALL") or a waveform number, it is stored in a string. If the response was "ALL", line 490 branches around the numeric conversion. Otherwise, lines 340-360 convert the string to a numeric value and check to see that the value is not less than or equal to zero or greater than the number of waveforms in the file.

Next, a FOR loop is set up that will be executed once for each channel used. If both channels are used, the loop is executed twice. Otherwise, it is only executed once. Lines 370-390 set-up the beginning and ending index values for the loop. The loop consists of only three statements—the FOR and NEXT statements in lines 400 and 420, and the READ statement in line 410. This READ statement gets the zero reference, vertical scale factors and vertical units for the channel(s) used.

For example, if REP A,B mode was used, the IF conditions in lines 380 and 390 fail, and the statements are not executed. As a result, the default beginning index value (0) and ending index value (1) are used. On the first pass through the loop, line 410 reads the zero reference, scale factors, and vertical units for channel A into element zero of the arrays. On the second pass, the values for channel B are read into element 1 of the arrays. If REP A mode was used, line 380 sets the ending index to 0, so the loop is only executed for channel A. For REP B mode, the beginning index is set to 1 and the loop is executed for channel B only.

The remainder of the program forms another FOR loop that reads the waveform data, scales it using the zero reference and scale factors, and produces the output display.

Data logging...Part III

Just before starting the loop, line 430 reads the time and date of acquisition. Lines 440 and 450 set up a variable (K) that is used in the loop to point to the settings information for the channel each waveform was acquired from. When REP A,B or ALT mode is used, data is acquired and written into the output file first from channel A, then from channel B. As a result, the odd number waveforms (1,3,5,...) are acquired from channel A. Even number waveforms (2,4,6,...) are acquired from channel B (Fig. 4). When an odd number waveform is read, K is equal to zero to point to channel A settings. When an even number waveform is read, K is equal to one to point to channel B settings. If REP A mode is used, K is set to zero for all waveforms; if REP B mode is used, K is set to one for all waveforms.

	Waveform Number	Pointer Value (K)
Channel A Data	1	0
Channel B Data	2	1
Channel A Data	3	0
Channel B Data	4	1
Channel A Data	5	0
...		

Fig. 4. When data is acquired with REP A,B or ALT mode, channel A data is always written first. The variable K points to the zero reference, record length, and other settings for the corresponding channel.

Line 440 sets K to a default value of zero. Then, if REP B mode was used, line 450 sets K equal to one. Line 460 begins the main loop.

Lines 470-490 set the value of K if REP A,B or ALT modes were used, since K's value then depends on the number of the waveform currently being read. If REP A or REP B mode was used, the value of K is fixed, and line 470 sends control directly to line 500. Otherwise, line 480 sets K to zero and line 490 checks the index variable, J. If J is even, the expression in the IF statement is true, so K is set to one.

Line 500 deletes array A and dimensions it using the record length and number of records for this channel. Notice that READ doesn't auto-dimension the array in this case because it was

manually dimensioned as a floating-point array. If READ had auto-dimensioned it, an integer array would have been created, which would make the computation performed later slower and less precise.

Next, line 520 tests to see if this waveform should be displayed. If you asked for ALL waveforms, or if the current index is equal to the number of the requested waveform, the output routine is executed. Otherwise, the output routine is skipped, and control is passed directly to the NEXT J statement in line 720, which starts the next pass through the loop.

Line 530 begins the graphic output by paging the terminal screen. A short message identifying the waveform number and the channel it was acquired from is printed at the top of the screen. Then, the waveform is scaled using the scale factors and zero reference, and line 570 graphs the waveform in the area of the terminal screen specified by the VIEWPORT command at the beginning of the program. The remainder of the loop from line 580 to line 710 prints the vertical units, acquisition date and time, number and length of records, and breakpoints. A typical display is shown in Fig. 5.

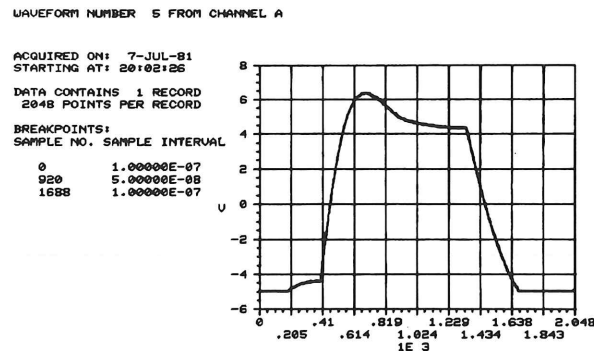


Fig. 5. A typical display produced by the program of Fig. 3.

The program pauses at this point to allow you to view or make a hard copy of the display. Execution continues when you press RETURN. When the requested waveform(s) have been displayed, the loop ends, the output file is closed, and control is passed back to ask you for another input file name.

This program demonstrates the process of reading a file with mixed data—waveforms,

settings, and other information. The key is all in knowing exactly how the file was written. The rest of the program depends on the application. The graphic output routine used in this example could easily be replaced or augmented with analysis or other data manipulation routines as the application requires.

A complete data logging/retrieval system

The data logging program shown in Fig. 3, Part I of this series and the program just discussed can be combined to form a complete data logging/retrieval system just by changing a few lines. Since the combined program would be too large to be completely memory resident in most controllers, it is broken into a small main program and two overlays—one for logging data and one for reading data. Figure 6 illustrates the overlays. The main program is shown in Fig. 7 and the two overlays are shown in Figs. 8 and 9.

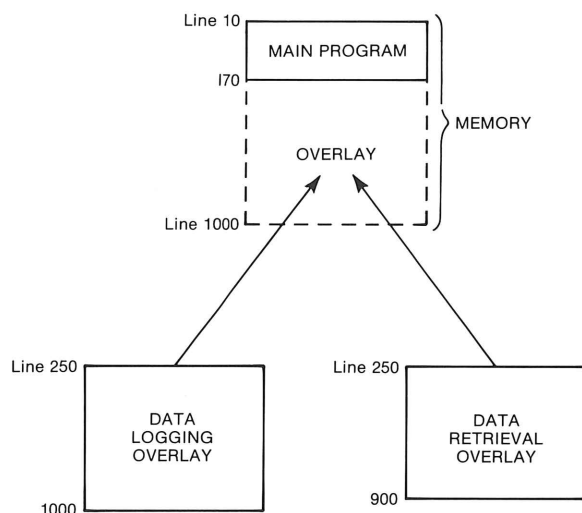


Fig. 6. The complete data logging/retrieval program is divided into a small main program that is always resident in memory and two overlays that are loaded as they are needed.

```

10 DIM CHS(1),PS(1)
20 CHS(0)="A" CHS(1)="B" PS(0)="L" PS(1)="R"
30 SA=0 BL=0 VF=0
40 UIEUPORT 420,960,250,680
50 PRINT "PLEASE ENTER DATE (DAY-MON-YR): ";
60 INPUT DS:SETDATE DS
70 PRINT "PLEASE ENTER TIME (HH:MM:SS): ";
80 INPUT TS:SETTIME TS
100 PAGE
110 PRINT "ENTER 'LOG' TO LOG DATA, 'READ' TO READ DATA OR 'QUIT' ";
120 INPUT FS
130 IF FS<>"LOG" THEN IF FS<>"READ" THEN IF FS<>"QUIT" THEN 110
140 IF FS="LOG" THEN IF F<1 THEN OVL0AD DL0:"76LOG.OV1"
150 IF FS="READ" THEN IF F<2 THEN OVL0AD DL0:"76LOG.OV2"
160 IF FS="QUIT" THEN END
170 DELETE SF,UF,VB,ZR

```

Fig. 7. The small main program is always resident. It loads the overlays (Figs. 8 and 9) as required.

```

250 F=1
260 REM ** GET MODE, CHANNELS, NO. OF WAVEFORMS AND FILENAME **
270 PRINT "DO YOU WANT TO USE 'ALT' OR 'REP' MODE TO ACQUIRE DATA?";
280 INPUT MD$
290 IF MD$<>"ALT" THEN IF MD$<>"REP" THEN 270
300 IF MD$="ALT" THEN 340
310 PRINT "ENTER CHANNEL(S) TO ACQUIRE DATA FROM CA OR B OR A,B: ";
320 INPUT CS
330 IF CS<>"A" THEN IF CS<>"B" THEN IF CS<>"A,B" THEN 340
340 PRINT "ENTER NO. OF WAVEFORMS (MUST BE EVEN FOR ALT OR REP A,B): ";
350 INPUT NU
360 IF NU<1 THEN 340
370 IF MD$="ALT" THEN IF ITP(NU/2)<NU/2 THEN 340
380 IF MD$="REP" THEN IF CS="A,B" THEN IF ITP(NU/2)<NU/2 THEN 340
390 PRINT "ENTER FILE NAME FOR DATA LOGGING OUTPUT: ";
400 INPUT FS
410 CANCEL DL0:FS
420 LOAD "INS
430 INTEGER NR(1),NR(1),RL(1)
440 ATTACH #1 AS INS1.0:WITH 1,2 #0
450 REM ** GET NO. OF RECORDS, RECORD LENGTH AND BREAKPOINTS **
460 IF MD$="REP" THEN IF CS="B" THEN 520
470 GET NR(0),RL(0) FROM #1,"TMS A;REC?"
480 GET NR(0) FROM #1,"NBPT?"
490 DELETE BA:DIM BA(NB(0)*2-1)
500 GET BA FROM #1,"SBPT?"
510 IF MD$="REP" THEN IF CS="A" THEN 570
520 GET NR(1),RL(1) FROM #1,"TMS B;REC?"
530 GET NR(1) FROM #1,"NBPT?"
540 DELETE BB:DIM BB(NB(1)*2-1)
550 GET BB FROM #1,"SBPT?"
560 REM ** CALCULATE FILE SIZE AND OPEN FILE **
570 BL=(NR(0)*NR(0)+(RL(1)*NR(1))/256+10
580 OPEN #2 AS DL0:FS FOR WRITE INTO BL
590 REM ** WRITE NO. OF WAVEFORMS, MODE, CHANNEL(S) **
600 REM ** NO. OF RECORDS, RECORD LENGTH, AND BREAKPOINTS **
610 WRITE #2,NU,MD$,CS
620 IF MD$="REP" THEN IF CS="B" THEN 640
630 WRITE #2,NR(0),RL(0),NB(0),BA
640 IF MD$="REP" THEN IF CS="A" THEN 670
650 WRITE #2,NR(1),RL(1),NB(1),BB
660 REM ** ACQUIRE ZERO REFERENCE AND SCALE FACTORS **
670 B=0 E=1
680 IF MD$="REP" THEN IF CS="A" THEN E=0
690 IF MD$="REP" THEN IF CS="B" THEN B=1
700 FOR J=B TO E
710 GET CPS FROM #1,J+SA+1,"CPL?"
720 PUT "CPL GND" INTO #1,J+SA+1
730 PUT "ARM " CHS(J) INTO #1
740 DELETE A
750 FOR I=1 TO NR(J)
760 WAIT 25
770 PUT "INTRIG" INTO #1
780 NEXT I
790 JS="INTRIG:READ "&CHS(J)
800 GET A FROM #1,JS
810 ZR=MEA(A)
820 PUT CPS INTO #1,J+SA+1
830 JS="US"&PS(J)&"1?"
840 GET SF,UF,VB FROM #1,JS
850 VAS=SEG(VAS,1,1)
860 WRITE #2,ZR,SF,VB
870 NEXT J
880 REM ** GET TIME AND DATE AND BEGIN DATA LOGGING **
890 IF MD$="REP" THEN IF CS<>"A,B" THEN 910
900 NU=NU/2
910 DELETE A,NR,RL,FS,NB,VB,CPS,ZR,SF,BA,BB
920 DATE DS:TIME TS
930 WRITE #2,DS,TS
940 RELEASE AUTO
950 IF MD$="REP" THEN LS="REP "&STR(NU)&","&CS
960 IF MD$="ALT" THEN LS="ALT "&STR(NU)
970 DLOG #2 FROM #1,LS
980 CLOSE #2:DETACH #1
990 GOTO 100

```

Fig. 8. This overlay, stored in the file "76LOG.OV1", handles data logging. With the exception of a few statements, it is identical to the data logging program shown in Fig. 3, Part I.

Except for the mechanics of handling the overlays, the program is functionally identical to the ones described earlier. It asks whether you want to log or read data and loads the appropriate overlay based on the response. The flag variable (F) in lines 140 and 150 of the main program, and line 250 of the overlays, indicates which overlay is currently resident, keeping the program from needlessly re-loading an overlay that is already resident if you ask for the same function twice.

Notice also that the overlays are stored in a pre-translated BASIC format. Normally, BASIC programs are stored on the disk as ASCII text and are translated into an internal binary format when loaded with the OLD command. This translation is somewhat time consuming, so the overlays for this program are stored on the disk in the internal binary format so that they can be directly loaded without translation.



```

250 F=2
260 PAGE
270 DIM ZR(1),SF(1),VAB(1)
280 INTEGER NR(1),RL(1),NB(1)
290 PRINT "ENTER DATA FILENAME: ";
300 INPUT F$
310 OPEN #2 AS DLOIFS FOR READ
320 READ #2,NV,NDS,CS
330 IF NDS="REP" THEN IF CS="B" THEN 370
340 DELETE BA
350 READ #2,NR(0),RL(0),NB(0),BA
360 IF NDS="REP" THEN IF CS="A" THEN 390
370 DELETE BB
380 READ #2,NR(1),RL(1),NB(1),BB
390 MS=NDS
400 IF NDS="REP" THEN MS=NDS$ " & CS
410 PRINT "NO. OF WAVEFORMS IN FILE: ";NV," ACQUIRED IN ";MS," MODE"
420 PRINT
430 PRINT "ENTER THE NUMBER OF THE WAVEFORM YOU WANT TO GET FROM THE"
440 PRINT "FILE, OR ENTER 'ALL' TO GET ALL WAVEFORMS."
450 PRINT
460 PRINT "WHICH WAVEFORM DO YOU WANT TO GET ";
470 INPUT UFS
480 I=NV
490 IF UFS="ALL" THEN 530
500 I=VAL(UFS)
510 IF I<0 THEN 460
520 IF I>NV THEN 460
530 B=0\B=1
540 IF NDS="REP" THEN IF CS="A" THEN E=0
550 IF NDS="REP" THEN IF CS="B" THEN B=1
560 FOR J=B TO E
570 READ #2,ZR(J),SF(J),VAB(J)
580 NEXT J
590 READ #2,DS,TS
600 K=0
610 IF NDS="REP" THEN IF CS="B" THEN K=1
620 FOR J=1 TO I
630 IF NDS="REP" THEN IF CS<>"A,B" THEN 660
640 K=0
650 IF ITP(J/2)=J/2 THEN K=1
660 DELETE A\DIM A((NR(K)*RL(K))-1)
670 READ #2,A
680 IF UFS<>"ALL" THEN IF J<>I THEN 800
690 PAGE
700 PRINT "WAVEFORM NUMBER ";J," FROM CHANNEL ";CHS(K)
710 SETGR VIEWPORT
720 A=(A-ZR(K))*SF(K)/32
730 GRAPH A
740 SMOVE 350,420\PRINT VAB(K)
750 SMOVE 0,690
760 PRINT "ACQUIRED ON: ";DS
770 PRINT "STARTING AT: ";TS\PRINT
780 PRINT "DATA CONTAINS ";NR(K)," RECORD";
790 IF NR(K)>1 THEN PRINT "S";\PRINT
800 PRINT RL(K)," POINTS PER RECORD"\PRINT
810 PRINT "BREAKPOINTS"
820 PRINT "SAMPLE NO. SAMPLE INTERVAL"\PRINT
830 FOR N=0 TO NB(K)*2-1 STEP 2
840 IF K=0 THEN PRINT TAB(2);BA(N);TAB(11);BA(N+1)
850 IF K=1 THEN PRINT TAB(2);BB(N);TAB(11);BB(N+1)
860 NEXT N
870 SMOVE 0,0\PRINT "PRESS RETURN TO CONTINUE";WAIT
880 NEXT J
890 CLOSE #2
900 GOTO 100

```

Fig. 9. This overlay, stored in the file "76LOG.OV2", handles reading the data files and producing the graphic output. It is functionally identical to the program shown in Fig. 3 of this article.

When an overlay is loaded, The statements in the overlay file replace statements with the same line number in memory. All other statements in memory are unaffected. As a result, after the overlay is loaded, control is passed to line 250, regardless of which overlay is loaded. If the data logging overlay is loaded, line 250 is the beginning of the logging routine. If the data retrieval routine is loaded, line 250 is the start of that routine.

This data logging/retrieval system probably won't exactly meet your specific needs, but it can be adapted and used as a base for building a program. It could even be further enhanced to contain a data logging overlay, an event detection overlay, and a data retrieval overlay. In any case, you should find the building blocks you need in the example programs to build a software package that best suits your needs. 

By Mark Tilden,
HANDSHAKE Staff.

TEK SPS BASIC routine for single-key program selection

Do you find yourself drawing on a library of standard programs for signal analysis? And do you find yourself doing directory listings to find file names every time you need another program? Or do you load several programs into different line number blocks, then wind up doing a listing to find the beginning line number each time you want to GOTO a different program?

If you find yourself in any of these situations, you can save yourself a lot of time and trouble by using the TEK SPS BASIC program listed in Fig. 1. This menu program allows you to call up as many as nine different routines or programs simply by pressing the terminal keys for numbers 1 through 9.

For example, let's say you frequently use programs for 1) acquiring a waveform, 2) storing it on a disk, 3) recalling it from a disk, 4) computing pulse parameters, 5) comparison to a standard waveform, 6) bandwidth determination, and so on up through nine different programs of your choice. In the menu routine listed in Fig. 1, all you have to do is install your programs in the line number blocks 1100-1199 for terminal key 1, 1200-1299 for 2, 1300-1399 for 3, etc. Or, for programs too long to fit in the assigned block of line numbers, you can simply use the block for a short routine to overlay the program from a disk. For this latter case, however, you need to make sure the program being

```

10 REM MENU ROUTINE
100 INPREQ CHAR,NOECHO GOSUB 1000
110 GOTO 110
1000 REM READ KEY
1005 INPUT I$
1010 ONERR ER GOTO 1055
1015 I=VAL(I$)
1020 ONERR
1025 INPREQ
1030 LOCKKB
1035 GOSUB I OF 1100,1200,1300,1400,1500,1600,1700,1800,1900
1040 LOCKKB OPEN
1045 INPREQ CHAR,NOECHO GOSUB 1000
1050 RETURN
1055 ONERR RETURN GOTO 1060
1060 PRINT "INVALID INPUT. USE KEYS 1-9 ONLY."
1065 RETURN
1100 PRINT "PROGRAM 1"
1110 RETURN
1200 PRINT "PROGRAM 2"
1210 RETURN
1300 PRINT "PROGRAM 3"
1310 RETURN
1400 PRINT "PROGRAM 4"
1410 RETURN
1500 PRINT "PROGRAM 5"
1510 RETURN
1600 PRINT "PROGRAM 6"
1610 RETURN
1700 PRINT "PROGRAM 7"
1710 RETURN
1800 PRINT "PROGRAM 8"
1810 RETURN
1900 PRINT "PROGRAM 9"
1910 RETURN

```

Fig. 1. Menu program written in TEK SPS BASIC V02-02 software.

overlayed is line-numbered 2000 and greater so that it doesn't overwrite the menu routine.

Once your library of programs is set up in the assigned line number blocks or set up for overlaying, all you need to do to call up a program is press the corresponding numeric key (1-9) on the terminal keyboard. As a memory jogger, small gummed labels with the program names can be stuck to your terminal above the number keys.

Beyond being a useful device for quickly calling up utility programs, the menu routine of Fig. 1 is also useful for demonstrating the INPREQ and ONERR commands of TEK SPS BASIC. To see how these commands function, let's go through the program beginning at line 100.


With TEK SPS BASIC V02-02 software, your terminal keyboard is live. That means you can enter data, commands, or program lines at any time, even while a program is running. The INPREQ command is simply a method of directing keyboard entries occurring while a program is running. In line 100 of Fig. 1, INPREQ enables the input request for a single character via the CHAR argument. Also, because of the optional NOECHO keyword, your input is not echoed (printed on the terminal). And finally, the GOSUB 1000 directs the program to go to the subroutine at line 1000 whenever there is keyboard input to process.

After INPREQ is activated, the program goes to line 110. This is simply a loop which keeps repeating itself while the program waits for input.

As soon as any character key is pressed on the terminal, INPREQ causes a branch from the loop at line 110 to the input routine at line 1000. At line 1005, the ASCII character entered from the keyboard is stored in the string variable I\$. Presumably, the character entered will be a single digit of 1 through 9. However, accidents can happen—an alpha character might be entered instead of a numeric. So an ONERR condition is set up at line 1010. The ONERR at 1010 essentially says, "if an error occurs, put the error message information into variable ER and go to line 1055 for error processing." In the next line, line 1015, the VAL function converts the ASCII character I\$ to its decimal value. If the character is not a number, a warning error occurs, and the ONERR condition branches the program to line 1055 which causes a message about input restrictions to be printed. Then the program returns to line 110 to wait for a correct input. However, if I\$ does contain a single-digit number, VAL converts it to floating point and stores the result in variable I.

In line 1020, the ONERR of line 1010 is disabled by the ONERR without keywords. Following that, INPREQ is disabled in line 1025 and the terminal keyboard is locked by LOCKKB in line 1030. This is done to keep further terminal input from interrupting the program being selected by the menu routine.

The single digit (1 through 9) entered from the keyboard and stored in I is used in line 1035 to branch to the subroutine line number in the list having a list position equal to I. If I equals 1, the branch is to the first line number in the list; if I equals 2, the branch is to the second line number in the list; and so on.

When the subroutine containing your utility program completes execution, the menu program's execution returns to line 1040. Here, the keyboard is unlocked to allow keyboard entry for selecting the next program. Then INPREQ is enabled again at line 1045 so that the keyboard input can be processed. And finally, line 1050 returns execution to the loop at line 110, where the program remains until it receives a keyboard input again. 

by Bob Ramirez,
HANDSHAKE Staff

Based on a program submitted
by Alan Jeddeloh, LDP Test Engineering,
Tektronix, Inc.

Getting the most out of TEK BASIC graphics—

For a different view, try isometric projection

Sometimes the best way to view an object is to pick it up in your hand, turn it around, and look at it from another angle. That may be easy if the object is sitting on the table in front of you. But what do you do when the object exists only as a computer simulation? You can use your imagination and guess what's hidden. Or, you can turn to sophisticated graphing techniques to simulate a three-dimensional image. Then, by specifying a viewing angle, you can get a glimpse of any surface.

This technique can also be helpful in viewing a series of related waveforms. In previous issues, programs have been given for obtaining and displaying multiple traces of slowly changing data (see "Orthogonal Projection of Sin X/X," Vol. 2 No. 2, Winter 76-77; "Studying Long Term Variations," Vol. 2 No. 3, Spring 1977; and "X-Y-Z Plotting Adds Another Dimension to Analysis Results," Vol. 4 No. 3, Spring/Summer 1979).

An isometric projection can be produced by tilting and rotating a set of waveforms on the X, Y, and Z axes. Figure 1 shows an isometric projection of long-term data variations resulting from ultrasonic measurements. Figure 2 shows several isometric projections of the data in Fig. 1 with a variety of different tilt and rotation angles.

Since the tilt, rotation, and hidden-line operations are done on a point-by-point basis, completing a plot does take a noticeable amount of time. But for the view you get, the wait is worth it!

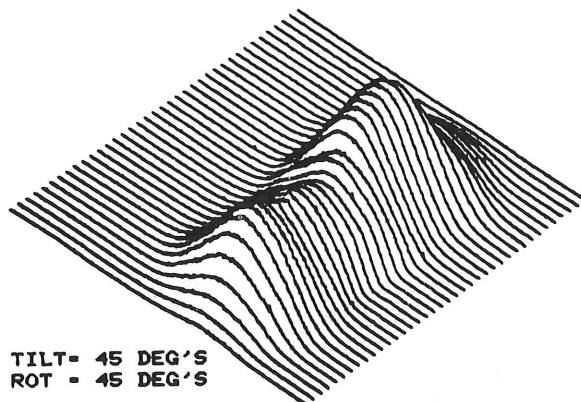
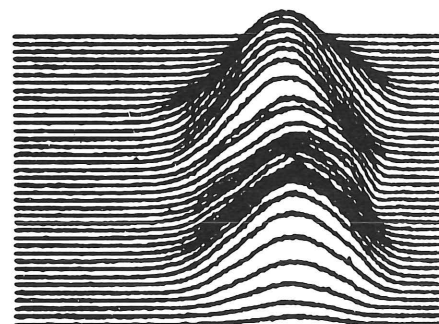
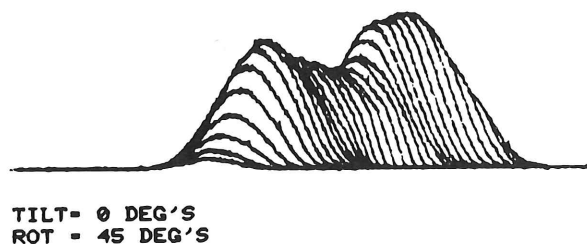
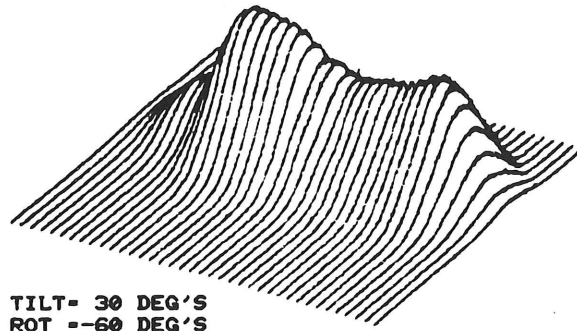
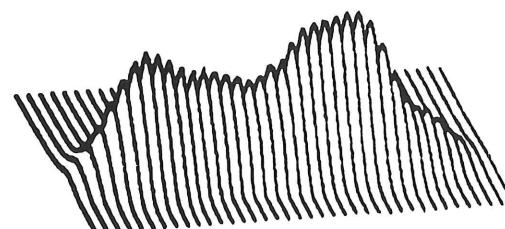


Fig. 1. Isometric projection of long-term data variations.



TILT= 45 DEG'S
ROT = 0 DEG'S



TILT= 20 DEG'S
ROT = 80 DEG'S

Fig. 2. Several different views of the isometric projection using different tilt and rotation angles.

Program Description

A TEK SPS BASIC program for producing an isometric display is shown in Fig. 3. Lines 10 through 170 set up the basic program. Data for the program is assumed to be stored on a flexible diskette in drive #1; it can be previously digitized data or data from a simulation routine. Line 120 allows the choice of finding the minimum and maximum values of the input data to

```

10 REM **** ISOMETRIC PROJECTION WITH HIDDEN LINE OPTION ****
20 REM **** FOR TEK SPS BASIC ****
30 PAGE
40 PRINT "INPUT NAME OF DATA FILE:"\INPUT NS
50 GS=NS&".DAT"
60 PRINT "INPUT DATE:"\INPUT DS
70 PAGE
80 WAVEFORM WA IS A(S11),SA,HAB,UAB
90 DIM H(S11),U(S11),P(S11),PP(S11)
100 CLOSE #2
110 OPEN #2 AS DX1:GS FOR READ
120 PRINT "DO YOU WISH TO FIND THE MAX AND MIN DATA VALUES (Y OR N)"
130 INPUT HS
140 IF HS="N" THEN GOTO 160
150 GOSUB 800
160 PAGE
170 PRINT "DO YOU WISH TO USE HIDDEN LINE PLOTTING (Y OR N)"\INPUT GS
180 PRINT
190 PRINT "INPUT ROTATION ANGLE (DEGREES)"\INPUT R
200 RR=(RX3.142/180)\SR-SIN(RR)\CR-COS(RR)
210 PRINT
220 PRINT "INPUT TILT ANGLE (DEGREES)"\INPUT T
230 TT=(TX3.142/180)\ST-SIN(TT)\CT-COS(TT)
240 REM - DISPLAY DATA IN FORM SELECTED
250 RESET #2
260 P=-1000000
270 PAGE
280 REM DEFINE Z-AXIS STEP SIZE
290 ZS=(DH-DL)/20
300 REM - COMPUTE X DELAY (NOTE: MUST BE INTEGER)
310 DE=INT(511*SR/(K-1)*CR)
320 REM - SET UP SIZE AND POSITION OF DISPLAY (NOTE: MUST BE SQUARE)
330 IF SR<0 THEN GOTO 360
340 VIEWPORT 0,500,350,850
350 GOTO 380
360 VIEWPORT 500,1000,200,700
370 REM - SET UP DATA WINDOW
380 DU=DH+ZS*(K-1)
390 WINDOW 0,S11,DL,DU
400 SMOVE 0,150\PRINT "TILT=";T;" DEG'S"
410 SMOVE 0,130\PRINT "ROT=";R;" DEG'S"
420 SMOVE 800,30\PRINT "C:";DS;"J"
430 SMOVE 150,30\PRINT "ISOMETRIC PROJECTION OF DATA FILE: ";GS
440 FOR II=1 TO K
450 READ #2,WA
460 X=0
470 XA=X+VA*(X)\ZA=ZS*(II-1)\GOSUB 920
480 MOVE H(0),U(0)
490 PP=P
500 REM - DRAW DISPLAY
510 FOR X=1 TO S11
520 XA=X+VA*(X)\ZA=ZS*(II-1)\GOSUB 920
530 IF GS="N" THEN GOTO 660
540 REM - ADJUST VIEWPORT IF ROTATIONAL ANGLE IS NEGATIVE
550 IF DE<0 THEN GOTO 580
560 IF X<S11-DE THEN GOTO 660
570 GOTO 590
580 IF X<-DE THEN GOTO 660
590 X=X-1
600 REM - TEST FOR HIDDEN LINE
610 IF U(XM)<P(DE+X) THEN GOTO 640
620 IF U(X)>P(DE+X) THEN GOTO 660
630 IF U(X)<P(DE+X) THEN GOTO 680
640 IF U(X)<P(DE+X) THEN GOTO 740
650 IF U(X)>P(DE+X) THEN GOTO 710
660 DRAW H(X),U(X)\PP(X)=U(X)
670 GOTO 750
680 GOSUB 990
690 DRAW XI,VI
700 GOTO 740
710 GOSUB 990
720 MOVE XI,VI\DRAW H(X),U(X)\PP(X)=U(X)
730 GOTO 750
740 PP(X)=P(X+DE)
750 NEXT X
760 P=PP
770 NEXT II
780 SMOVE 0,0
790 END
800 REM - SUBROUTINE TO FIND DATA MAX AND MIN VALUES
810 PRINT "DATA FILE: ";GS;" NOW BEING SEARCHED FOR MAX AND MIN VALUES"
820 DH=-1000000
830 DL=1000000
840 K=0
850 READ #2,WA
860 EOF #2 GOTO 910
870 IF DH<MAX(A) THEN DH=MAX(A)
880 IF DL>MIN(A) THEN DL=MIN(A)
890 K=K+1
900 GOTO 850
910 RETURN
920 REM - SUBROUTINE TO PERFORM TILT AND ROTATION
930 ZX=S11*ZA/(ZS*(K-1))
940 H(X)=XA*CR+ZX*SR
950 ZY=(DU-DL)*ZA/(ZS*(K-1))
960 XY=(DU-DL)*XA/S11
970 U(X)=YACT+(ZY*CR-XY*SR)*ST
980 RETURN
990 REM - SUBROUTINE TO FIND CROSSING POINTS
1000 S1=(P(X+DE)-P(XM+DE))/(X-XM)
1010 S2=(U(X)-U(XM))/(X-XM)
1020 XI=(P(XM+DE)-S1*(X-XM)-U(XM)+S2*(X-XM))/(S2-S1)
1030 VI=S2*(XI+U(XM))-S2*(X-XM)
1040 RETURN


```

Fig. 3. TEK SPS BASIC program listing to produce an isometric projection.

automatically set up the data window; this calculation is handled by the subroutine in lines 800-910. You should always answer Y (yes) the first time through the program; for subsequent plots of the same data, N (no) uses the values calculated the first time through the program to save time.

Line 170 allows choice of hidden-line plotting. Lines 190-230 ask you to input the rotation and tilt angle for the isometric projection. Rotation refers to the displacement angle from the Z axes in the XZ plane; tilt is the displacement angle from the Y axes in the YZ plane.

Data display begins at line 240. Line 310 computes the integer step for use with the hidden-line option. Lines 320-360 set up a square-sided viewport based upon the sign of the rotation angle; 370-390 set up the data window. Lines 400-440 provide labels for the graph.

The original data file is read again at line 450 to obtain the waveforms to be plotted by lines 460-590. Each point of each waveform is rotated and tilted using the subroutine in 920-980. If the hidden-line option was selected, each successive waveform is tested for hidden lines as it is plotted using the routine in lines 600-780 and the crossing point subroutine in lines 990-1040. 

by P.J. Highmore and B.S. Gray,
United Kingdom Atomic Energy Authority,
Risley Nuclear Power Development Laboratories,
K11/RD3
Risley, Warrington WA3 6AT
England

This application is condensed from an application report entitled "Graphical Representations of Ultrasonic Data Using a Digital Oscilloscope," ND-R-411(R). Copies of the report can be obtained by writing Dr. Highmore at the above address.

Four new programmable digitizer systems available



To meet varying needs in waveform digitizing and processing, Tektronix has introduced four new waveform processing systems—the WP2251, WP2252, WP3201, and WP3202. These systems are based on two different, but fully programmable, waveform digitizers and a mix of controllers and peripheral storage capacities.

State-of-the-art waveform capture

The WP2251 and WP2252 systems use the 7912AD Programmable Digitizer for waveform capture. The 7912AD is a ten-bit digitizer providing a waveform record length of 512 samples. It employs a scan converter for state-of-the-art digitizing speed and is the choice for high-bandwidth signal capture (500 MHz bandwidth with a 7A19 plug-in, 200 MHz with a 7A16P plug-in).

The WP3201 and WP3202 systems use the 7612D Programmable Digitizer for maximum flexibility in medium-speed signal capture. The 7612D is actually two digitizers in one. It uses two electron-bombarded-semiconductor (EBS) digitizer tubes for dual-channel, eight-bit digitizing with a 90 MHz bandwidth. Dual-channel flexibility is

further enhanced by long record lengths (2048 waveform samples per channel), pre- and post-trigger capabilities, and the ability to switch sample rates during digitizing.

Choice of controller power, storage capacity

The 7912AD in the WP2251 system and the 7612D in the WP3201 system are interfaced to identical signal processing systems. The processor is a TEKTRONIX CP1164X Option 31 Controller. This is a high-speed controller that is virtually identical to the DEC PDP-11/34A Minicomputer*. Beyond high speed, the primary features are 64K words (128K bytes) of MOS memory, memory management, bootstrap ROM, automatic self-test diagnostics, extended instruction set, and a DL11W Interface for a TEKTRONIX 4010-family display terminal. Available options for additional capabilities include another 64K of MOS memory, high-speed cache memory, and a floating-point processor.

Peripheral storage for the WP2251 and WP3201 systems is provided by DEC RL11-AK and RL01-AK Disk Drives. These provide random-access data storage using a high-density, single-disk cartridge. Each drive carries one cartridge with each being capable of storing five megabytes of data.

The WP2252 (7912AD digitizer) and WP3202 (7612D digitizer) systems provide instrument control and processing with a DEC PDP-11/23AA Controller. This controller features 64K words of MOS memory, memory management, bootstrap ROM, automatic self-test diagnostics, and the KEF11-AA Floating Point option. Also, a DLV11-F interface is provided for a TEKTRONIX 4010-family display terminal.

Peripheral storage for the WP2252 and WP3202 systems is provided by a DEC RX02 Double-Density, Flexible Diskette Drive. This is a random access dual diskette drive. Each diskette provides storage and retrieval for 512K eight-bit bytes of data.

GPIB compatibility plus


Each controller is complete with an IEEE 488 interface for control of up to 14 GPIB instruments.

**DEC and PDP are registered trademarks of the Digital Equipment Corporation.*

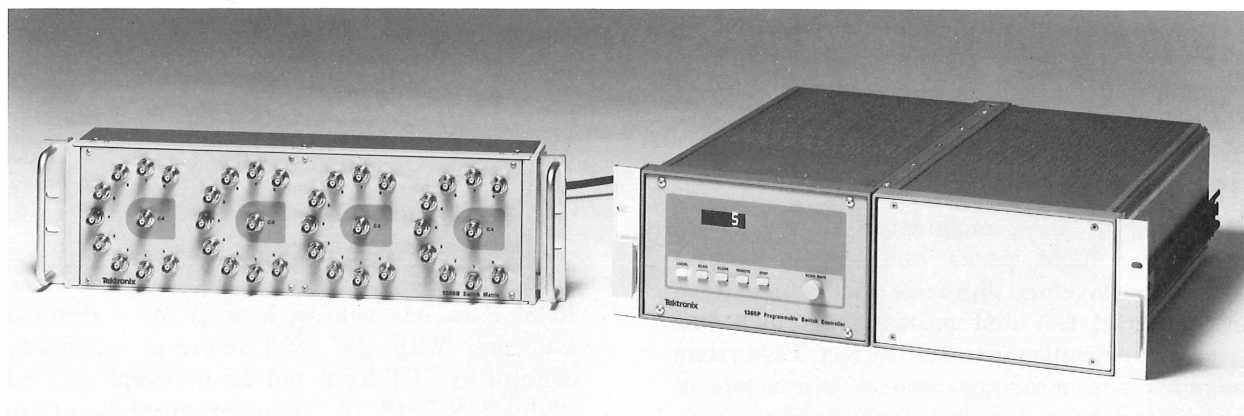
Additionally, the CP1164X controller has the potential capacity for three more IEEE 488 interfaces, allowing control of up to 56 bus-interfaced instruments.

The system software, TEK SPS BASIC V02 for 7912AD systems and an extended memory version (V02XM) for 7612D systems, enhances GPIB compatibility as well as provides full signal processing capabilities. The software includes a GPIB driver and a specialized commands module for the system digitizers (7912AD Driver for the WP2251 and WP2252 or 7612D Driver for the WP3201 and WP3202).

TEK Codes and Formats adds the final plus to these systems. TEK Codes and Formats is a Tektronix standard that ensures consistent IEEE 488 implementation, including standards for device commands and statement syntax. This makes instrument control programming easier, whether you use just the basic 7912AD or 7612D system or add on a full complement of GPIB instruments from Tektronix.

For more information on these systems and instruments, contact your local Tektronix Field Office or Tektronix Sales Representative, or use the reply card bound into this issue of HANDSHAKE. 

1360P/1360S provides GPIB control of 129 signal paths



Whether you need to send test signals to a number of test stations or monitor signals from those stations, the TEKTRONIX 1360P/1360S Programmable Signal Multiplexer offers an economical approach to signal distribution. Its 50-ohm coaxial switches have individual bandwidths of 250 MHz and capacities of 250 volts DC or 250 milliamperes (10 VA maximum). What's more, switching can be set up either manually or under program control over the GPIB.

There are actually two distinct units involved—the 1360P Switch Controller and the 1360S Switch Matrix.

The microprocessor-based 1360P contains a GPIB interface (IEEE 488-1978) and allows switch selection either manually or under program control. Additionally, the 1360P can be manually set to a scan mode, allowing the switches to be

progressively scanned at setable rates from 3 milliseconds to 10 seconds per switch.

The 1360S Switch Matrix unit provides 33 signal paths through four identical nine-pole coaxial switch assemblies. Strapping in the 1360P allows the matrix to be set for 1 output with 33 inputs, 2 ganged outputs with 17 inputs, or 4 ganged outputs with 8 inputs. Multiples of these combinations are possible with additional switching units. Each 1360P is capable of driving up to four 1360S switchers for a total of 129 signal paths that can be switched under program control.

For more information on this or any other Tektronix product, contact your local Tektronix Field Engineer or Sales Representative or use the reply card bound into this issue of HANDSHAKE.



New Real Time Clock ROM Pack — another measurement dimension for 4052 and 4054 systems



In automated test, the time when something happens can be as important as the event or measurement itself. That's why Tektronix, Inc., has developed and made available the 4052R09 Real Time Clock ROM Pack for 4052 and 4054 Graphic Computing Systems. Now you can add the dimensions of time and date, elapsed time, or vectored time interrupts to your measurements.

Time and date

A 4052 or 4054, augmented by the signal processing ROM packs and any of several Tektronix waveform digitizers, makes an extremely powerful test and measurement tool. One popular application is in quality test. The system acquires and measures product parameters in order to make a pass-fail decision. Sometimes, failure data is the only data kept and logged. In other cases, it's often necessary to qualify products by attaching test documentation to the product. In almost every case, it's at least handy if not an outright requirement to put the testing time and date on the test data. This is where the 4052R09 Real Time Clock ROM Pack comes in.

When you turn on your system, call the SETIME command to set the 4052R09 ROM clock to the desired time and date. Time is set in a 24-hour format in hours, minutes, and seconds.

Now, when you want to know the time, just call the RDTIME command to read the date and time into the string variable of your choice. This variable can then be logged as part of stored waveform or measurement data. Or you can print the variable on the viewing screen along with the associated test waveforms and data for hard copying.

Elapsed time

Besides the time-and-date clock, the 4052R09 Real Time Clock ROM Pack also has an elapsed time clock. This provides a stopwatch function that counts in 0.1 second increments.

To start the stopwatch, just call the STARTW command. The watch will start counting in tenths of a second. When you want to read the elapsed time, call the STOPIT command. The elapsed time in seconds will be read into the variable you specify with STOPIT.


If you want to measure the run time of one of your programs, use STARTW in the first line of the program and STOPIT in the last line. But more important applications occur in biophysical stimulus-response measurements, physics, chemistry, or any place where elapsed time is an important part of the data.

Vectored timed interrupts

Do you ever have to gather data on a timed basis? If so, vectored timed interrupts will let you do the job automatically.

The ONTIME command of the Real Time Clock ROM Pack lets you set a timer for a vectored interrupt. With the 4050 interrupt capability enabled by SET KEY and the interrupt interval set by "ONTIME", T, your program will branch to line 84 when the interval specified by T elapses. By some simple looping or subroutining, the ONTIME can be reinitiated each time T elapses. Thus, you have the capability of programming periodic vectored timed interrupts.

For example, you can set T to 300 (300 seconds) and set up your program to branch every T seconds to an instrument control and data collection routine for process monitoring. You can even use your 4052 or 4054 for main-task processing while waiting for the interrupt and return automatically to main-task processing after the process monitoring task completes.

In short, given the time—by a 4052R09 Real Time Clock ROM Pack of course—you can now schedule as well as automate your measurements. To find out more about the 4052R09 contact your local Tektronix Sales Engineer or Sales Representative. 

7612D improves resolution on low duty cycle pulse capture

Low duty cycle pulses crop up in a number of application areas. What probably comes to mind first are the traditional and well-developed areas of SONAR and RADAR. However, there are a variety of other areas where similar echo-ranging and identification techniques can be valuable test tools—tools for measuring distance, thickness, buried strata composition and arrangement, void size and location, and so on. The problem is that these applications are often not widespread. Indeed, some echo-return applications are still in the feasibility or research stages. As a result, instrumentation specialized for low duty cycle pulse or echo capture is limited, and the measurements often have to be made initially with general purpose instruments.

For some coarse echo-return measurements, standard oscilloscopes and counters can do the job. But detailed information is often difficult to get. As indicated in Fig. 1, low duty cycle pulses have long “dead” times between pulses. Adjusting an oscilloscope to view and measure these dead times precludes detailed observation of pulse structure. On the other hand, adjustment to observe detail on a pulse precludes measurement of the time between pulses. In some instances, this dichotomy can be eliminated by operating in a dual-channel alternate or chopped sweep mode with one channel adjusted for viewing pulse detail and the other adjusted for measuring the time between pulses. However, in some applications, the acquisition problem is further compounded by variation in the interpulse distance. There may even be substantial variations in the pulses themselves. In short, it's no longer a repetitive waveform situation, but a single-shot acquisition requirement.

For such random pulse trains, where both pulse detail and accurate interpulse time measurements are important, the TEKTRONIX 7612D Programmable Digitizer offers a simple measurement solution. The 7612D has two independent channels, each with its own sequential digitizer. Additionally, each channel's 2048-element record length can be partitioned into eight records of 256 elements each. This means that you can capture up to eight waveforms per channel in a single-shot mode. You can store eight



Fig. 1. A difficult acquisition situation posed by a pulse train with variations in individual pulses as well as interpulse time.

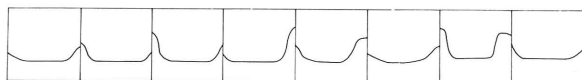


Fig. 2. A simple acquisition solution using the dual-channel, multiple-record features of the TEKTRONIX 7612D Programmable Digitizer.

pulses at high resolution in one channel and the dead times at a lower resolution in the other channel.

For example, let's say you need five nanoseconds per sample to get sufficient resolution on the pulses in the train and around 100 nanoseconds per sample for the dead times. Channel A can be set up for eight 256-element records with a five nanosecond sampling interval and triggering set for the positive slope. Channel B is set for eight records, also, with 100 nanoseconds per sample and set to trigger on the negative slope of each pulse.

This setup allows you to capture eight successive pulses in Channel A and eight successive dead times in Channel B as indicated in Fig. 2. The 7612D gives you the resolution you need for both the pulses and the dead times.



by Bob Ramirez,
HANDSHAKE Staff

Based on a memo from
Dean Turnbaugh,
Tektronix, Inc.,
Rockville Field Office

Literature available from Tektronix

The following literature can be ordered via the reply card bound into the center of this issue of HANDSHAKE.

Transducer Packages,

Catalog Page AX-3451-1.

This product sheet lists the various mechanical motion transducer packages available from Tektronix. Basic specifications are included. Also listed are the various mounting kits, cable assemblies, etc. for use with each transducer.

TEKTRONIX Codes and Formats for GPIB Instruments,

Application Note 99AX-4607.

The various concepts and philosophies followed by Tektronix in implementing the IEEE-488 standard instrument interface (GPIB) are discussed in this application note. Included is a discussion of **TEK Codes and Formats**, the additional standard used by Tektronix to assure consistent and friendly communication between Tektronix instruments and instrument controllers.

GPIB Communication with the 7854,

Application Note AX-4416.

This application note demonstrates interfacing of the 7854 over the GPIB to a 4052 Graphic Computing System and 4924 Tape Drive. The discussion is supported by a variety of programs for transferring data, waveforms, text, and programs between devices.

Introduction to 7854 Oscilloscope Measurement and Programming Techniques,

Application Note 42AX-4682.

The fundamentals of using the 7854 as a stand-alone tool for waveform digitizing and processing are introduced. Discussions include the basics of waveform storage, cursor measurements, and measurement programming with the Waveform Calculator. Automation of pulse analysis is used as the programming example.

Capture fast waveforms accurately with a 2-channel programmable digitizer,

Electronic Design reprint AX-4401.

This article reprint contains an in-depth discussion of the new 7612D Programmable Digitizer from Tektronix, Inc. The major features—dual-channel operation, variable record length, sample rate switching, and pre- and post-triggering—are described in terms of both operation and use. A full application example, testing a three-electrode gas lightning arrester, is also presented.

HANDSHAKE Application Library Catalog.

This catalog contains abstracts on a variety of signal processing programs that are available free as listings. Program listings are provided on an "as is" basis and are documented by embedded comments and available support literature.

HANDSHAKE
Group 157 (94-384)
Tektronix, Inc.
P.O. Box 500
Beaverton, Oregon 97077

BULK RATE U.S. POSTAGE PAID Tektronix, Inc.

Tektronix®
COMMITTED TO EXCELLENCE

45A-4858