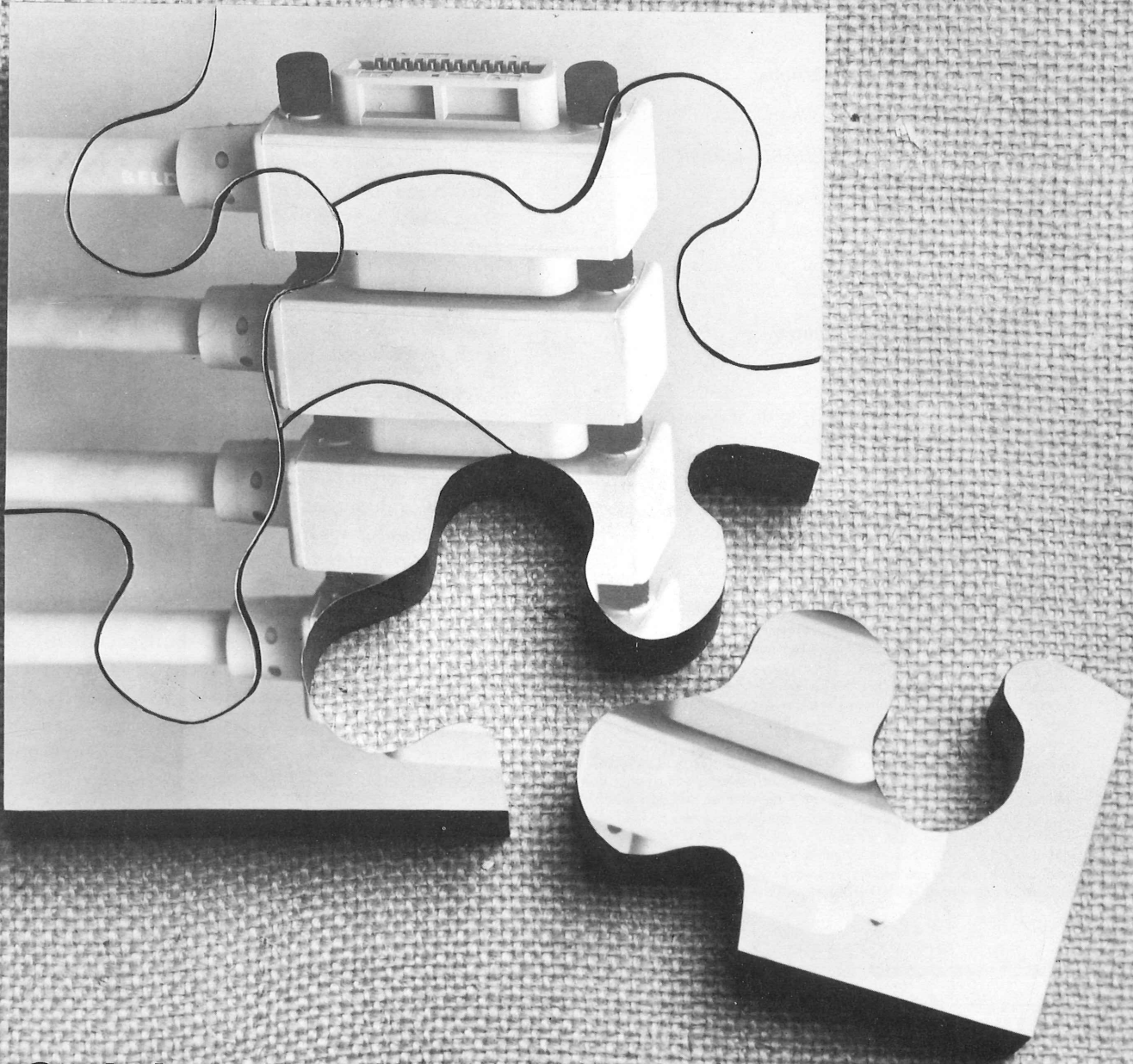


Tektronix®
COMMITTED TO EXCELLENCE



HANDSHAKE

Newsletter of Signal Processing and Instrument Control



Solving the GPIB puzzle

Table of contents

Solving the GPIB puzzle:

<i>Talking to the 7854 Oscilloscope with TEK SPS BASIC</i>	3
<i>Simple approaches to controlling the 7612D</i>	12
<i>CG551AP and TEK SPS BASIC make auto cal easy</i>	17
<i>Literature available from Tektronix</i>	22
<i>ASCII & IEEE (GPIB) Code Chart</i>	23
<i>Getting the most out of TEK BASIC graphics</i>	24
<i>Programming hints</i>	26

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 individual 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.

Solving the GPIB puzzle:

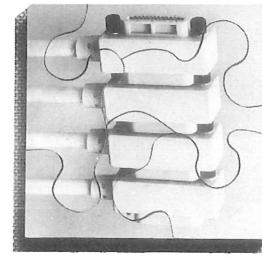
A preview

There are two ways to learn about the General Purpose Interface Bus (GPIB) and how it can be used to interface and operate test instrument systems. The first way is to spend a lot of time studying IEEE 488, the standard that defines the GPIB, until you are thoroughly familiar with it and the various ways it can be implemented. Unfortunately, most of us don't have the time for that. That leaves us the second path—learning enough from a few simple examples to get started. Then, as need dictates, this base of knowledge is added to. And finally, through several experiences, a reasonable operating familiarity with GPIB concepts and methods is developed.

This issue of HANDSHAKE offers you some guidance down that second path. It contains several examples of instrument control over the GPIB. The controller used in the examples is a Digital Equipment Corporation PDP*-11 series minicomputer with a Tektronix GPIB interface, and the instrument control and signal processing software used is the Tektronix developed TEK SPS BASIC package. The Tektronix instruments around which the examples are built are the 7612D Programmable Digitizer, the CG551AP Programmable Calibration Generator, and the 7854 Oscilloscope. Once you've seen how they are operated over the GPIB, you'll have the knowledge base needed to take that first step toward your own programs. 

*PDP is a trademark of Digital Equipment Corporation.

Solving the GPIB puzzle:



Talking to the 7854 Oscilloscope with TEK SPS BASIC

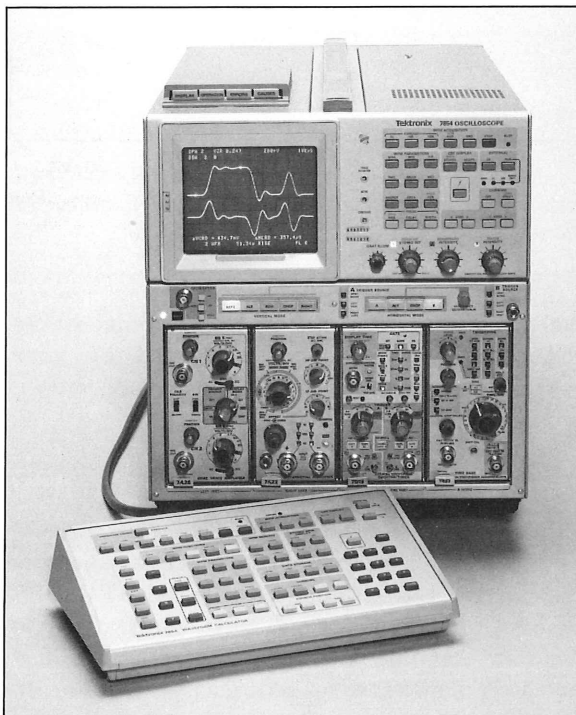


Fig. 1. The 7854 Oscilloscope with its attached Waveform Calculator keypad.

The new TEKTRONIX 7854 Oscilloscope goes far beyond what its name implies. It is much more than a standard oscilloscope. Waveforms acquired through its 400 MHz mainframe can be displayed in real time or digitized and stored in memory for later analysis. And a good share of that waveform analysis can be done via the 7854's internal microprocessor and firmware. Many individual functions—waveform maximum, minimum, RMS value, integration, differentiation, etc.—are provided as single keys on the attached Waveform Calculator (Fig. 1). Also, you can combine and store these keystroke functions as programs. In short, the 7854 is an oscilloscope enhanced by a small onboard computer.

Beyond being a powerful stand-alone measurement and analysis tool, the 7854 can also be used as a system component. It contains a GPIB interface conforming to IEEE Standard 488-1978. Thus, the 7854's capabilities and capacities can be further extended by adding a GPIB compatible desk-top computer or a minicomputer. The combination can be for any purpose, from simply providing more program and waveform storage space to providing additional computational power.

One such combination, offering an extraordinary variety of possibilities, is the pairing of the 7854 with a Digital Equipment Corporation PDP* 11 series minicomputer using a Tektronix GPIB interface and running with TEK SPS BASIC software. But before any of the possibilities can be realized, communication over the GPIB must be established. Here's how to do it with TEK SPS BASIC.

Getting plugged in

GPIB compatibility means, in the simplest sense, that you can plug your GPIB instrument and minicomputer or controller together without encountering any mechanical or electrical difficulties. There is a standard GPIB cable that matches the GPIB connectors on the instrument and controller, and the electrical levels and activities of the interface and bus lines are all standardized for compatibility. So you can plug things together as indicated in Fig. 2 without having to know anything about GPIB operation other than there must be a device load for every two meters of cable.

But just plugging things together doesn't mean they're going to work together!

**PDP is a trademark of Digital Equipment Corporation.*

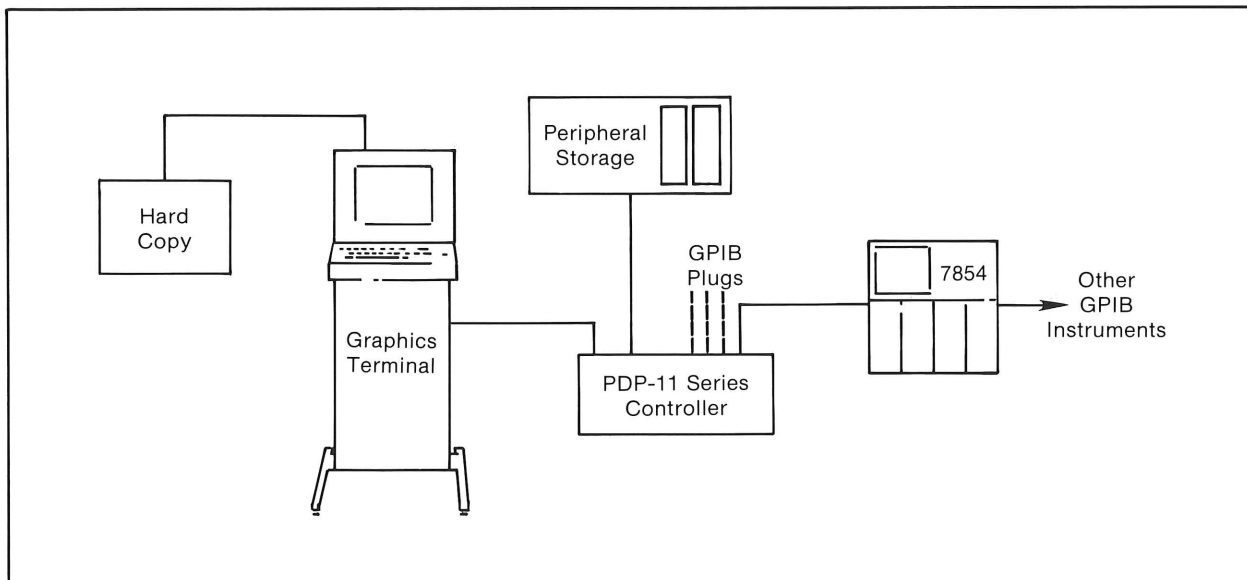


Fig. 2. The 7854 and PDP-11 based GPIB system. Backplane space permitting, SPS TEK BASIC can support up to four Tektronix GPIB interface cards. Each interface card and bus can support 15 devices (14 instruments plus controller) with one device load per two meters of cable.

GPIB compatibility also means there is a defined set of interface functions that must be used in governing bus operations and traffic. The rub is that not all of those functions have to be used to comply with the standard. In fact, most instruments and controllers implement only a subset of the available functions, and there are usually differences from one type of instrument to the next in what functions are implemented and how they are implemented. In other words, it's possible to come up with two devices, each complying with the standard while not being functionally compatible with each other.

As an example, consider message terminators. The GPIB has an EOI line (End Or Identify) which can be asserted with the last byte of a message as a message terminator. **EOI can be used as a message terminator, but it doesn't have to be.** In fact, there are three typical methods of message termination, all allowed by the standard, in use:

1. line feed
2. EOI
3. line feed and EOI

Now, should you have an instrument using line feed for message termination and a controller expecting EOI for message termination, you have a basic incompatibility.

Fortunately, most devices are strappable for various message terminations. But this does

mean, however, that you will have to determine the message terminator recognized by your controller and strap your instrument for compatibility.

In the case of PDP-11 series controllers using TEKTRONIX CP4100/IEEE 488 or CP1100/IEEE 488 Interface Boards, TEK SPS BASIC recognizes EOI as the message terminator. This means that, for compatibility with TEK SPS BASIC, your instruments must be strapped to generate EOI as the message terminator. The 7854 has a set of switches for that purpose (see Fig. 3). These switches also allow selection of a talk only, listen only, or talk/listen communication mode for the 7854. **To set the 7854 for GPIB operation with a PDP-11 and TEK SPS BASIC software, set switch 1 to 1 (ON LINE) and switches 2 and 3 to 0 (EOI, TALK/LISTEN communication mode).**

The remaining GPIB selection switches (4-8) are used to set the 7854's primary address. Possible selections for the primary address run from decimal 0 to 31. Which address you select depends upon several things. First of all, although address 31 is a possible switch setting, it is not a valid primary address. The instrument will essentially be off line if a primary address of 31 is used. So don't use address 31. Secondly, each device on a bus must have a different primary address. And finally, some controllers reserve an address for

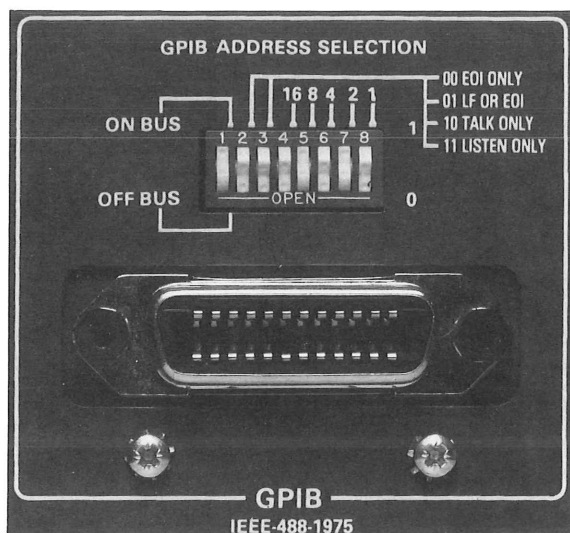


Fig. 3. 7854 Oscilloscope's GPIB connector and selection switches for setting primary address and communication mode.

themselves. This means that you cannot use that controller address for any other device on the bus. The TEKTRONIX 4050-Series Graphic Computing Systems, for example, reserve address 0. However, PDP-11 series controllers operating with Tektronix GPIB interface boards and TEK SPS BASIC software, assume no address. So you can use anything from 0 to 30 for your 7854. Generally, it's more convenient to use address 1 for the first or only instrument on the bus, address 2 for the second instrument, 3 for the third, etc.

For the purposes of this article, the 7854 primary address is set to 1. That means that 7854 GPIB selector switches 4 through 7 should be put in the 0 position and switch 8 in the 1 position.

Powering up with TEK SPS BASIC

Once your GPIB system is cabled and the message terminator and addresses selected, it is ready for power up. With Tektronix supplied systems operating under TEK SPS BASIC, the power-up sequence is not critical. However, for GPIB systems in general, it's good practice to power up the controller and its peripherals first, then load system software, and then power up the instruments on the bus. (For multiple instrument systems, more than half the bus-connected devices have to be powered up.)

When the 7854 is powered up, it goes through a self-test sequence. When the self test completes, the 7854 asserts an SRQ (service request). This

SRQ should be serviced by reading the status byte to make sure the instrument's power-up sequence completed successfully.

This preliminary activity—servicing the SRQ by reading the status—is done with the POLL statement of the TEK SPS BASIC Low-Level GPIB driver. To perform the poll, the driver must first be loaded. This is done by entering the following statement from your terminal.

```
LOAD "GPI.SPS"
```

The poll is then performed by entering a statement similar to the following.

```
POLL @0,S,ST,SS;65
```

In actual practice, some variation of this statement might be required. Any variation, however, will relate to the @0 and 65 used above.

The @0 used here refers to controller interface board number zero. As pointed out in Fig. 3, it is possible to support up to four GPIB interface boards (numbers 0-3) with a PDP-11 series controller and TEK SPS BASIC. So, in order to address an instrument on a particular bus, it is necessary to first address the controller interface board supporting that bus. In the example here, interface board number 0 is being addressed.

The 65 used in the example POLL statement refers to the talk address of the instrument having a primary address of 1. The talk address is obtained by adding 64 to the primary address. So, in cases where your instrument has a primary address other than 1, the talk address used will be different.

A talk address is used in the POLL statement because the poll is asking the instrument for information. In order for the instrument to send information (talk) to the controller, it has to be addressed to be a talker. If, on the other hand, it were being asked to receive information (listen) from the controller, as is the case for some other types of statements, it would need to be addressed as a listener (primary address + 32).

The variables S, ST, and SS will contain the information obtained by the POLL statement. S will be the value of the instrument status byte, ST will be the primary address of the instrument polled, and SS will be the secondary address. For example, after polling the 7854 power-up SRQ, the values of S, ST, and SS can be observed by using

Talking to the 7854...

the PRINT statement as follows.

```
POLL @0,S,ST,SS;65
PRINT S,ST,SS
65      65      0
```

The first number printed is the decimal value of the status byte. In this case it is 65, indicating power on for the instrument. (A full list of status byte meanings is provided in the 7854 Operators Manual.) The second value printed, again 65 in the example, is the talk address of the instrument serviced by the poll. The third output, zero in the example, is the secondary address of the serviced instrument. In the case of the 7854, secondary addressing is not used, hence the returned value of zero for SS.

It should be pointed out that the primary purpose of a poll is to service an SRQ to find out what the asserting device wants. If POLL is executed when an SRQ isn't being asserted, the routine returns zeros to its status and address variables. To get the status of an instrument regardless of whether it is asserting SRQ, use the GETSTA statement. For example,

```
GETSTA @0,S,65
```

gets the status byte of the instrument on interface 0 and having a primary talk address of 65. If that instrument happens to have SRQ asserted, reading its status clears the SRQ.

In general operation, an initialization routine is used to take care of power-up SRQs, addressing, checking status, etc. A very simple TEK SPS BASIC example is listed below.

```
10 REM INITIALIZATION ROUTINE
15 LOAD "GPI.SPS"
20 P1=1
25 L1=P1+32
30 T1=P1+64
35 POLL @0,S,ST,SS;T1
40 IF S=65 THEN 55
45 PAGE\PRINT S,ST,SS
50 PRINT "POWER-UP STATUS 65 NOT DETECTED"\GOTO 60
55 PAGE\PRINT "POWER UP OKAY"
60 END
```

This routine loads the GPIB low-level driver (line 15) and then sets variables for the primary, talk, and listen addresses (lines 20-30). Using address variables is a matter of convenience—mnemonically related variables are easier to remember than the numbers and the required increments, and changing addresses requires only changing the value of one variable (P1) rather than changing a numeric constant in each

program statement. The poll occurs in line 35, and the line following that determines action based on the status byte value. If the status is 65 (reserved in Tektronix GPIB instruments for valid power up), line 40 causes a branch to line 55. If the status is not 65, the routine prints the status and address values obtained by POLL and then prints a message indicating that the expected status was not detected. The printed value of the status byte gives an indication of what may be amiss.

Although this example initialization routine is quite simple and directed toward a single instrument, the same basic concept applies to multiple instrument systems. Multiple instrument systems just require more housekeeping tasks.

Simple instrument-controller dialog

Once your system is powered up and initialized, you can begin transferring commands and data back and forth between the 7854 and the controller. A good share of this is done with the PUT, RASCII, and WASCII statements of TEK SPS BASIC.

Learning to use these statements is best done by executing some simple operations in the immediate mode. In other words, sit down at your terminal and type in statements without line numbers so that they execute as soon as you press the return key. For example, if you'd like to put the 7854 into the SCOPE display mode, simply type

```
PUT "SCOPE" INTO @0,L1
```

To put the 7854 into the STORED display mode, type

```
PUT "STORED" INTO @0,L1
```

To put it into the BOTH display mode, type

```
PUT "BOTH" INTO @0,L1
```

Each of these examples could also be followed by a GETSTA or POLL to clear resulting SRQs. But, when operating the 7854 in the immediate mode with TEK SPS BASIC, you can just ignore the SRQs.

For the above three examples, it is assumed the 7854 is on the bus serviced by controller interface board number 0, hence the @0. L1 is the listen address and is used because the 7854 has to be in the listen mode to receive the message contained in the PUT statement.

The message is enclosed in quotes and is device dependent. Device dependent means it is specific to an instrument, in this case the 7854. The

message causes certain activities to take place within the instrument. Specifically, SCOPE, STORED, and BOTH cause the instrument to react exactly as if you'd physically pressed the SCOPE, STORED, or BOTH buttons on the 7854. SCOPE causes a real-time waveform from the plug-ins to be displayed, STORED causes a waveform from 7854 memory to be displayed, and BOTH causes simultaneous display of stored and real-time waveforms.

All of the labels above the keys on the 7854 measurement keyboards correspond to device-dependent messages that can be sent to the 7854 with a PUT statement. For example, the 7854 keystroke sequence for storing a waveform by signal averaging 100 times and then finding the peak-to-peak value can be executed by sending the following PUT statement sequence.

```
PUT "BOTH" INTO @0,L1
PUT "1 0 0 AVG" INTO @0,L1
PUT "P-P" INTO @0,L1
```

With regard to the second statement in the sequence, several subtle items of 7854 format should be noted. First, 1 0 0 AVG is the command sequence for signal averaging 100 times. Notice that the command sequence is given in Reverse Polish Notation; that is, the argument (100) precedes the command (AVG). Also, notice that each digit in the argument is separated by a space. This again is 7854 format, which requires that each keystroke be delimited by a space. Since numbers are entered one keystroke per digit, each digit must be separated by a space. And finally, more than one keystroke or device-dependent message can be included in a PUT statement. For example, the sequence could be reduced to the following.

```
PUT "BOTH 1 0 0 AVG P-P" INTO @0,L1
```

In either case, the result is the same—the 7854 signal averages a waveform 100 times and then computes its peak-to-peak value and stores it in the instrument's X register. An example of the resulting 7854 display is given in Fig. 4.

To transfer the X register contents (the peak-to-peak value in Fig. 4) out of the 7854 to the system controller, the 7854 must first be prepared for sending data from the X register. This can be done using the SENDX message as shown below.

```
PUT "SENDX" INTO @0,L1
```

Following this with

```
RASCII X FROM @0,T1
```

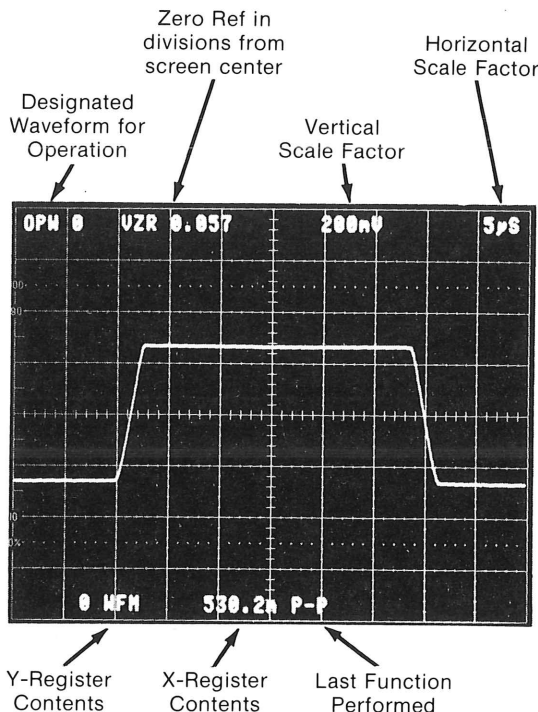


Fig. 4. Example of 7854 display showing a stored waveform with its peak-to-peak value in the X register.

causes the value in the 7854's X register to be transferred over the bus into variable X of TEK SPS BASIC.

Going the other way, you might rather read the contents of a TEK SPS BASIC variable into the 7854's X register. To do this, the 7854 must first be prepared to read data from the bus. This can be done using the READX message as follows.

```
PUT "READX" INTO @0,L1
```

The 7854 is now waiting for the data. To send the data, the contents of TEK SPS BASIC variable ZZ for example, use the following statement.

```
WASCII ZZ INTO @0,L1
```

This causes the contents of ZZ to be converted to ASCII and sent over the bus to listen address L1. The 7854, listen address L1, reads the ASCII data into its X register and also displays the value at the X register location on its screen.

Transferring bigger chunks— waveform data

Generally, the choice will be to process waveforms within the 7854. Eventually, however, there'll be a need to transfer waveforms into the system controller. This might be for the more extensive processing available with TEK SPS

BASIC, to archive waveforms in disk memory, or simply to use the more convenient hard copy capabilities for documenting waveforms. Whatever the case, there is a substantial amount of information to be handled in a waveform transfer and a variety of control tasks to be performed. Because of this increased complexity, successful transfer will be more likely when done under program control rather than from the terminal keyboard.

In programming GPIB communication with any instrument, there are a number of general items that should be taken into consideration.

First of all, there is the interface time-out period. Generally, GPIB controllers have a fixed or default interface time-out value. If a bus handshake cycle is not completed within that time, a time-out error occurs. In short, the activity is not completed in the allotted time, so the program is aborted. This prevents the bus from being tied up waiting for handshake completion should there ever be an instrument malfunction. However, there are also some normal operating cases where the handshake cycle can take longer than usual. This can occur with slow instruments or because a program asks for a response to a time consuming operation before the operation is complete.

In some cases, a time-out situation can be avoided by careful programming. In TEK SPS BASIC, however, time-out problems can easily be avoided by using the SIFTO (set interface time-out) command to select your own time-out value before communicating with an instrument.

For just getting your programs running, it is a good idea to put in a 1000-millisecond time-out value (actual time varies slightly with controller speed). This doesn't slow anything down or change anything other than just set up a condition where software will wait longer (1000 milliseconds) if it has to for handshake completion. Then, after the program is debugged and working as it should, you can reduce the time-out value. For simple programs this may not be necessary. But, for multiple instrument applications requiring fast completion, you may not want an instrument malfunction tying up the bus while a long time-out expires. In such cases, experimentally reduce the time-out value to the minimum required for successful program operation.

After time-out considerations, the next consideration is what you want your program to do while an instrument is busy. There are two general cases.

The first case is when a program needs information from an instrument before going any further. An example of this is telling an instrument to send a waveform to the controller for processing or storage. Naturally software cannot process or store the waveform until the instrument sends it. But it takes time for the instrument to decode the SENDX message and prepare for waveform transfer, and during that time the program goes on executing commands unless you make it wait for the instrument. This waiting is often accomplished with a program statement that loops on itself until the desired instrument status is achieved. Then the program picks up the information from the instrument and goes on to process it.

The second case is somewhat opposite of the first. It is where you set the instrument about some task, but want your program to continue normal execution until the instrument has completed the task and is ready with information. Instruments generally indicate completion of a task or readiness with information by asserting SRQ (service request). There are power-up SRQs, command-completion SRQs, and so forth. With TEK SPS BASIC, any SRQ can be recognized by a WHEN statement and given a higher priority than normal program execution. The WHEN statement is a way of telling the program, "when this particular condition or event occurs, stop what you are doing, take care of the situation, then return to what you were doing." Typically, WHEN statements are used in TEK SPS BASIC programs to branch to polling routines for servicing instrument SRQs as they occur.

Data format is the final major consideration in waveform transfers over the GPIB. The data format the instrument uses needs to be determined so that you can either preserve that format for easier transfers back to the instrument or modify it as needed for external processing.

For the 7854, the waveform data format is described in Fig. 5. The waveform data is sent in three major parts. There is an ASCII waveform preamble consisting of a header and then a string of descriptors giving pertinent information about waveform size, scaling, etc. This is followed by a separator which is either a carriage return or carriage return with line feed, depending on the instrument's message terminator setting (see Fig. 3). Following the separator is the curve header (CURVE) and then the curve data points, which are sent as ASCII-coded decimal numbers. All of this is sent as a single message terminated by EOI.

Waveform Data:

```
WFMPRE ENCDG:ASC,NR.PT:512,PT.FMT:Y,XZERO:0,XINCR:9.766E-06,
XUNIT:S,YZERO:2.704,YMULT:1,YUNIT:V;
CURVE 1.3779,1.3777,1.3778,1.3777,....,1.3777,-2.6953,-2.6955,-2.6955
```

Definitions:

WFMPRE ¹	waveform preamble (header)
ENCDG:ASC ¹	curve data encoded ASCII decimal
NR.PT:(P/W)	number of points/waveform
PT.FMT:Y ¹	point format (curve data in vert. div.)
XZERO:0 ¹	no horizontal offset
XINCR:[10*HSCL/(P/W)]	horizontal increment between points
XUNIT:S	horizontal scale factor units (S=seconds)
YZERO:[-(VSCL*VZR)]	vertical zero offset
YMULT:(VSCL)	vertical scale factor
YUNIT:V	vertical scale factor units (V=volts)
CURVE	waveform data header

¹Fixed value, cannot be changed.

Curve values are ASCII coded decimal numbers separated by commas. Each number represents a point on the waveform and is the vertical distance of that point above (+) or below (-) the center horizontal graticule line of the 7854 display.

Fig. 5. Data format for waveforms sent over the bus by the 7854 Oscilloscope. The waveform is sent as one message consisting of a waveform preamble, separator (carriage return for EOI or carriage return and line feed for LF OR EOI), and the curve data.

The above points—interface time-out, SRQ handling, status detection, and waveform message format—are taken into account variously in the sample TEK SPS BASIC programs of Figs. 6 and 7. Figure 6 lists two programs—one for reading a waveform out of a 7854 and onto a floppy disk and one for writing the waveform back into a 7854. The program in Fig. 7 is slightly different in that it only reads the waveform into the controller, where it is then converted to TEK SPS BASIC format for waveform processing.

Starting at line 105 of the program for transferring a waveform to a floppy disk (Fig. 6), the interface time-out value is set to 1000. This is followed in line 110 by setting the status variable, S, to zero. Then an array, ZW, is dimensioned to receive the curve data. In this case, it is presumed the curve contains 512 data values (points 0 through 511); however, the dimension will need to be changed for curves of other lengths. The next line, line 120, uses a WHEN statement to cause branching to the subroutine at line 1000 whenever an SRQ interrupt occurs. Note that the subroutine at line 1000 is simply a POLL to read the instrument status into S. This concludes the preliminary set up of conditions for transfer of a 7854 waveform.

The program goes on at line 125 to send a message telling the 7854 to put 0 WFM on the bus. At this point the program must wait (loop) until the 7854 is ready to send the waveform. Line 130 does this waiting by looping continuously while checking the value of S until S reaches 210. S does not reach 210 until the 7854 initiates the SENDX, at which time the 7854 sets its status to 210 (SENDX initiated) and asserts SRQ. The WHEN condition, set by line 120, recognizes the SRQ and causes a branch to service it (POLL at line 1000). S is set to 210 as a result of the POLL. Upon return from the POLL subroutine, the statement at line 130 finds S to be equal to 210 (waveform ready to send), and program execution moves to the next line, line 135.

Line 135 sets the termination character (STERMC) to a semicolon. In the next line, RASCII begins reading the waveform message into ZW\$ until a semicolon is reached, which denotes the end of the 7854 waveform preamble. At that point, it switches to reading the waveform data an element at a time into numeric array ZW. Since RASCII is reading into a numeric rather than string variable, it discards the data header (CURVE) and all other characters except numerics and +, -, ., and the letter E. The EOI, sent

Talking to the 7854...

```

100 REM STORE 7854 0 WFM ON FLOPPY
105 SIFTO @0,1000
110 S=0
115 DIM ZW(511)
120 WHEN @0 HAS "SRQ" AT 51 GOSUB 1000
125 PUT "0 WFM SENDX" INTO @0,L1
130 IF S<210 THEN 130
135 STERMC @0,";"
140 RASCII ZW$,ZW FROM @0,T1
145 CANCEL DX1:"WAF.1"
150 OPEN #1 AS DX1:"WAF.1" FOR WRITE
155 WRITE #1,ZW$,ZW
160 CLOSE #1
165 END
200 REM READ WAF.1 BACK TO 7854 0 WFM
205 OPEN #1 AS DX1:"WAF.1" FOR READ
210 READ #1,ZW$,ZW
215 CLOSE #1
220 SIFTO @0,1000
225 ZW$=ZW$&"CURVE "
230 S=0
235 WHEN @0 HAS "SRQ" AT 51 GOSUB 1000
240 PUT "0 WFM READX" INTO @0,L1
245 IF S<211 THEN 245
250 WASCII ZW$,ZW;INTO @0,L1
255 END
1000 POLL @0,S,ST,SS;T1
1005 RETURN

```

Fig. 6. *TEK SPS BASIC programs for transferring 7854 waveforms to a floppy disk (lines 100-165) and back to the 7854 (lines 200-255).*

by the 7854 with the last byte of its message, terminates the RASCII. At this point, the waveform exists in the controller as the preamble stored in ZW\$ and the data values stored in array ZW. The rest of the program uses standard procedure to write these variables out to a floppy disk.

The second program in Fig. 6 reads the waveform data back out to the 7854 by essentially just reversing the process. The disk file is opened and the information read into ZW\$ and array ZW. After setting the interface time-out, the program adds ;CURVE to ZW\$ since those characters were discarded by the RASCII in the preceding program. Then, after setting up for communication over the bus, the waveform is written as two parts, ZW\$ and ZW, back out to the 7854.

While the programs listed in Fig. 6 perform 7854 waveform transfers with a minimum of data format change, the program listed in Fig. 7 takes a different tack. Its purpose is to read a 7854 waveform into the controller and then convert it to the waveform processing format of TEK SPS BASIC.

The difference begins in line 330 where the termination character is set to a comma. This allows the RASCII in the next line to read each waveform preamble component, except the last one, into separate variables for individual use

```

300 REM GET WAVEFORM FROM 7854
305 SIFTO @0,1000
310 S=0
315 WHEN @0 HAS "SRQ" AT 51 GOSUB 1000
320 PUT "0 WFM SENDX" INTO @0,L1
325 IF S<210 THEN 325
330 STERMC @0,";"
335 RASCII Z1$,Z2,Z3$,Z4$,Z5,Z6$,Z7,Z8 FROM @0,T1
340 STERMC @0,";"
345 DELETE B
350 WAVEFORM WB IS B(Z2-1),HB,HB$,VB$
355 RASCII Z9$ FROM @0,T1
360 STERMC @0,""
365 RASCII B FROM @0,T1
370 B=B*Z8+Z7
375 HB=Z5\HB$=SEG(Z6$,7,LEN(Z6$))
380 VB$=SEG(Z9$,7,LEN(Z9$))
385 PAGE
390 GRAPH WB
395 END
1000 POLL @0,S,ST,SS;T1
1005 RETURN

```

Fig. 7. *TEK SPS BASIC program to get a waveform from the 7854 and convert it to TEK SPS BASIC format for signal processing.*

later. Once the preamble is read in, the termination character is set to a semicolon (line 340), and a TEK SPS BASIC WAVEFORM is specified using the points-per-waveform information (Z2) from the preamble. The last preamble element, terminated by a semicolon, is then read into Z9\$ by line 335. Line 360 sets the termination character to a null, indicating termination on EOI only. Then the remaining waveform points are read into array B of WAVEFORM WB.

Immediately following this, in line 370, the data points are converted from divisions to waveform values by multiplying by the scale factor (Z8, read from preamble in line 335) and offset by the appropriate amount (Z7). The next two lines deal with setting the WAVEFORM's digital increment and units variables. Some string processing is necessary to segment out the desired characters for the units. Line 380 completes formatting of the 7854 waveform data to the WAVEFORM format used by TEK SPS BASIC in signal processing. The waveform is now ready for fast Fourier transformation, convolution, correlation, or whatever TEK SPS BASIC capabilities you wish to bring to bear on the analysis.

Program transfers

The final form of communication you might want to set up is that of transferring 7854 programs back and forth between disk storage and the 7854. The uses of this vary from the simple one of providing permanent storage for 7854 programs to the more complex one of a multi-instrument, distributed processing system where

programs are down loaded from the host controller to various 7854 stations as needed. For either case, the basic idea of 7854 program transfer is embodied in the two TEK SPS BASIC programs listed in Fig. 8.

```

400 REM TRANSFER 7854 PROG. TO FLOPPY
405 SIFT0 @0,1000
410 S=0\FL=0
415 WHEN @0 HAS "SRQ" AT 51 GOSUB 1000
420 PUT "EXECUTE 0 GOTO PROGRAM SAVE" INTO @0,L1
425 IF S<208 THEN 425
430 CANCEL DX1:"P7854.PRO"
435 OPEN #1 AS DX1:"P7854.PRO" FOR WRITE
440 STERMC @0,CHR(13)
445 WHEN @0 HAS "EOI" GOSUB 475
450 RASCII PL$ FROM @0,T1
455 PRINT #1,PL$
460 IF FL=0 THEN 450
465 CLOSE #1
470 END
475 FL=1
480 RETURN
500 REM TRANSFER PROG. TO 7854
505 SIFT0 @0,1000
510 S=0
515 WHEN @0 HAS "SRQ" AT 51 GOSUB 1000
520 PUT "PROGRAM CLP NEXT" INTO @0,L1
525 IF S<66 THEN 525
530 OPEN #1 AS DX1:"P7854.PRO" FOR READ
535 EOF #1 GOTO 555
540 INPUT #1,PL$
545 WASCII PL$ INTO @0,L1
550 GOTO 540
555 CLOSE #1
560 END
1000 POLL @0,S,ST,SS;T1
1005 RETURN

```

Fig. 8. TEK SPS BASIC programs for transferring a 7854 program to floppy disk (lines 400-480) and back to the 7854 (lines 500-560).

Assuming you've developed a 7854 program and have it keyed into a 7854, the first TEK SPS BASIC program (Fig. 8, lines 400 to 408) allows transfer of that program from the 7854 to a floppy disk for storage. This program is quite similar in many respects to those used for waveform transfer. Interface time out is set in line 405, line 410 sets some variables for control use, and line 415 sets up a WHEN for the same reasons as discussed before. The EXECUTE 0 GOTO PROGRAM SAVE in line 420 is the command sequence to the 7854 for setting up transfer of its program, and the looping in the following line is set for exit on a status value of 208, which indicates initiation of the SAVE command.

Since the 7854 sends each of its program lines terminated by a carriage return, line 440 sets the termination character for RASCII to carriage return (ASCII decimal code 13). Each line of 7854 program is then read into PL\$ by RASCII (line


450) and printed to the disk (line 455). Line 460 causes a loop back to read in and print the next line, and so on until the end of the program is reached.

The 7854 asserts EOI at the end of the program. The EOI is detected by the WHEN set up by line 445. This results in the looping variable, FL, being set to a value of one so that an exit occurs after the last line of the program is read into PL\$ and printed to the disk. That completes transfer and storage of the 7854 program, and the file is closed at line 465.

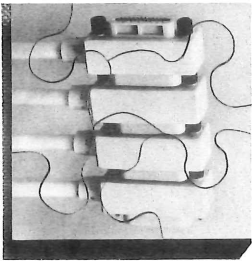
The second program in Fig. 8 (lines 500 through 560) reads the 7854 program from the disk and back into 7854 memory. This program is quite similar to the one for transfer to the disk. A loop is used to input the program a line at a time and write it out to the 7854 (lines 540 through 550). When the end of the file (EOF) is reached on the disk, line 535 causes a branch out of the loop, the file is closed, and the transfer program ended.

Taking the next step

All of the basic communication concepts and tools for building a GPIB system based on TEK SPS BASIC and the 7854 are embodied in the programming examples given here. Constants, commands, waveforms, and programs can all be easily transferred back and forth between the instrument and controller as needed.

The next step is to use these tools to build larger, more specific programs for your particular waveform capture, storage, and analysis needs. Tektronix maintains a network of Field Offices and overseas representatives that will be glad to assist you in defining those measurement needs and in selecting an instrumentation system to meet them. If you would like one of our field people to contact you, simply check the appropriate box on the reply card bound into this issue of HANDSHAKE. 

*By Bob Ramirez, HANDSHAKE Staff,
with grateful acknowledgment to
David Haworth, SID Scope Evaluation,
and Mark Tilden, HANDSHAKE Staff,
for their programming assistance.*



Solving the GPIB Puzzle:

Simple approaches to controlling the 7612D

Tektronix recently opened a new door in the field of high-speed sequential digitizers with the 7612D (see Spring/Summer 1980 HANDSHAKE). This dual-channel digitizer samples at up to 200 megahertz, has pre-trigger capability, sample-rate switching, and is fully programmable. Every instrument function can be controlled via the IEEE 488 bus (commonly referred to as the GPIB).

But taking full advantage of the 7612D's programmable features requires a powerful, versatile software package. TEK SPS BASIC provides the power and flexibility to make instrument control via the GPIB a simple task.

To get you started with your own programs, this article presents four simple routines for controlling and acquiring data from the 7612D. Each routine acquires a single record of data with no breakpoints (constant sampling rate). In all cases, it is assumed the 7612D has one 7A16 programmable plug-in in channel A, with the input signal connected to the upper connector (channel 1) on the plug-in. It is also assumed the instrument is connected to GPIB interface number 0 and that the primary address of the instrument is 1 and the secondary address is 0. (Refer to the instrument manuals for more information on setting the bus address). And, lastly, these programs only acquire data—they do not modify instrument settings. Programming instrument settings as well as the various modes of data acquisition will be covered in future articles.

Starting with the basics

The first program, shown in Fig. 1, uses the standard low-level GPIB driver (GPI) that is part of TEK SPS BASIC V02. This driver provides general purpose (independent of the type of instrument connected to the bus) line-level control for the GPIB.

```

10 REM THIS IS A SIMPLE 7612D ACQUISITION ROUTINE.
20 REM IT USES THE LOW-LEVEL GPIB DRIVER (GPI) FOR
30 REM TEK SPS BASIC V02. THE ROUTINE ASSUMES THAT
40 REM THE INSTRUMENT IS SET-UP TO ACQUIRE ONE RECORD
50 REM WITH A SINGLE SAMPLING INTERVAL. IT ALSO
60 REM ASSUMES THAT A SINGLE PROGRAMMABLE PLUG-IN
70 REM IS INSTALLED IN CHANNEL A OF THE INSTRUMENT.
80 REM
90 REM *** LOAD THE DRIVER AND SET-UP PARAMETERS ***
100 REM
110 LOAD "GPI.SPS"
120 IL=0
130 LA=32
140 TA=64
150 SA=96
160 SIFTO @IL,-1
170 REM *** GET RECORD LENGTH AND DIMENSION ARRAY ***
180 DELETE A,WA
190 PUT "REC?" INTO @IL,LA,SA
200 GET A$ FROM @IL,TA,SA
210 NP=VAL(SEG(A$,POS(A$," ",1)+1,LEN(A$)-1))
220 WAVEFORM WA IS A(NP-1),DA,HA$,VA$
230 REM *** ACQUIRE ZERO REFERENCE ***
240 PUT "CPL?" INTO @IL,LA,SA+1
250 GET C$ FROM @IL,TA,SA+1
260 PUT "CPL GND" INTO @IL,LA,SA+1
270 PUT "ARM A" INTO @IL,LA,SA
280 WAIT 100
290 PUT "MTRIG;READ A" INTO @IL,LA,SA
300 READBI A FROM @IL,TA,SA
310 ZR=MEA(A)
320 REM *** ACQUIRE WAVEFORM DATA ***
330 PUT C$ INTO @IL,LA,SA+1
340 PUT "ARM A;MTRIG;READ A" INTO @IL,LA,SA
350 READBI A FROM @IL,TA,SA
360 REM *** ACQUIRE SCALE FACTORS AND NORMALIZE ***
370 HA$="S"
380 PUT "VSL1?" INTO @IL,LA,SA
390 GET SF$ FROM @IL,TA,SA
400 PO=POS(SF$," ",1)
410 SF=VAL(SEG(SF$,POS(SF$," ",1)+1,PO-1))
420 VA$=SEG(SF$,PO+1,PO+1)
430 REM *** ASSUME ONE SAMPLING INTERVAL ***
440 PUT "SBPT?" INTO @IL,LA,SA
450 GET A$ FROM @IL,TA,SA
460 DA=VAL(SEG(A$,POS(A$," ",1)+1,LEN(A$)-1))
470 A=(A-ZR)*SF/32
480 PAGE
490 GRAPH WA
500 RELEASE "GPI.SPS"
510 END

```

Fig. 1.A Basic 7612D acquisition routine using the low-level GPIB driver (GPI).

The program starts by loading the GPI driver and initializing the interface and address variables. Then, line 160 sets the interface time-out value to infinity (-1). This causes the driver to wait indefinitely for the 7612D to respond. Next, the waveform data arrays are deleted and the program uses the REC? query to ask the 7612D for its record length. The response is stored in string variable A\$. The format of the REC? query response is:

```
REC n,y;
```

Where n is the number of records and y is the length (number of samples) of each record.

Line 210 extracts the record length value (y) from this string and assigns that value to the variable NP. This value determines the size of the destination array for the waveform data.

The WAVEFORM statement in line 220 declares a waveform named WA. This waveform consists of the A data array (which is dimensioned to NP-1 points), the sampling interval (DA), horizontal units (HA\$), and vertical units (VA\$). Figure 2 shows a typical waveform and identifies its components. The complete waveform, including data and scale factors, can be referenced using the waveform variable, WA.

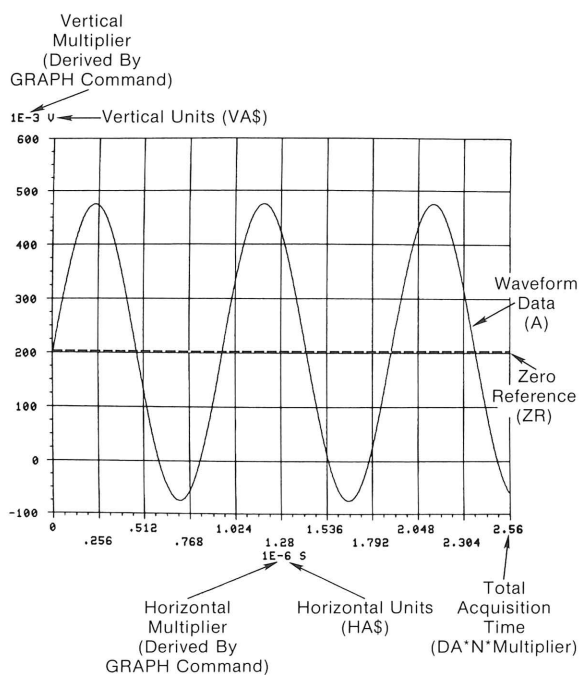


Fig. 2. A complete waveform consists of data, data sampling interval, horizontal units, and vertical units. The vertical multiplier and horizontal multiplier are derived by the GRAPH command.

With the preparations out of the way, data acquisition can begin. The first step is to acquire a zero reference with the plug-in input grounded. This reference level will later be subtracted from the waveform data to compensate for the Position control setting and yield a true DC level for the waveform. Line 240 reads the Coupling setting from the plug-in and stores this setting in C\$. Then Coupling is set to Ground (GND) and the 7612D is armed.

Since the 7612D may not trigger with its input grounded, the program sends a manual trigger (MTRIG) command to start the zero reference acquisition. However, the arm process may take many milliseconds to complete, so the WAIT statement in line 280 delays program execution for about 100 milliseconds to be sure that the arm process completes before the manual trigger command is sent.

Following MTRIG, the program also sends a READ command. The READ causes the 7612D to begin transmitting its data as soon as acquisition is complete. The READBI command in line 280 accepts the data and puts it into array A. If the acquisition is not complete when line 280 is executed, the program simply waits. The interface time-out value is set to -1, so the program will wait indefinitely. During this period, the GPIB waits for the 7612D to transmit data. As a result, no other device-dependent messages can be sent across the bus. This simplifies programming. But, for multiple instrument systems or when speed is important, another technique is demonstrated later.

For now however, when the zero reference acquisition is complete, line 290 sets the zero reference value, ZR, to the mean of the acquired data values. Next, the original setting for the plug-in coupling, stored in C\$, is sent back to the plug-in. The instrument is armed again, and the waveform data is acquired. Then, the data is read into array A. Again, if the acquisition is not complete when the READBI command is executed, the program waits.

The next step is to acquire the scale factor and units values that describe the waveform data. The horizontal units will be seconds, so an "S" is stored HA\$. The vertical scale factors are acquired using the VSL1? query (Vertical Scale factors for the Left plug-in, channel 1). The 7612D responds to this query with a message of the form:

```
VSL1 +5.E+00,V, CAL
```

Simple approaches...

Line 390 stores the query response in SF\$. The next two lines extract the volts/division value from the string and assign that value to SF. Then, line 420 assigns the units (V for volts) to VA\$.

The SBPT? query in line 440 gets the sampling interval (assuming a single sampling interval). Line 460 puts the sampling interval value extracted from the query response in DA.

Finally, the waveform data is normalized by subtracting the zero reference value from each data point, multiplying the result by the vertical scale factor, and dividing that result by 32 (the 7612D has 32 data levels per division on a standard 8-by-10 graticule).

Lines 480 and 490 page the terminal and graph the data. The GRAPH command automatically scales the vertical axis for best display. Then, it generates the graticule, labels the axis with scale factors, and graphs the data (see Fig. 2). When the graphing is complete, line 500 releases the GPI driver.

Interrupts free the bus

One limitation of the program presented in Fig. 1 is the "dead" time caused by waiting for the 7612D to complete its acquisition. Between the time that the READ command is sent and the acquisition is completed, the GPIB waits for the 7612D to transmit its data. While the bus is waiting, no device-dependent messages can be sent, which could be an important limitation in multiple-instrument systems.

The 7612D has a feature that can be used to free the bus for other traffic while an acquisition is in progress. When a channel completes its acquisition, the 7612D can generate an SRQ interrupt to tell the controller that the waveform data is ready for output. This interrupt is called a waveform readable interrupt (WRI). Adding a few lines to the program in Fig. 1 takes advantage of this feature.

The modified program is shown in Fig. 3 with the new lines in bold print. The program arms the instrument and waits for a waveform readable interrupt. During acquisition, the program waits, but the bus is free for other traffic. When the interrupt occurs, the program sends a read command to the instrument and gets the data.

Lines 220-330 get the record length, declare the waveform, store the plug-in coupling mode, and prepare for zero-reference acquisition just as in the first program. The difference starts at line 340. Instead of immediately sending the READ

```
10 REM THIS IS A SIMPLE INTERRUPT-DRIVEN 7612D
20 REM ACQUISITION ROUTINE. IT USES THE LOW-LEVEL GPIB
30 REM DRIVER FOR TEK SPS BASIC V02. THE ROUTINE ASSUMES
40 REM THAT THE INSTRUMENT IS SET-UP TO ACQUIRE ONE
50 REM RECORD WITH A SINGLE SAMPLING INTERVAL. IT ALSO
60 REM ASSUMES THAT A SINGLE PROGRAMMABLE PLUG-IN IS
70 REM INSTALLED IN CHANNEL A OF THE INSTRUMENT.
80 REM
90 REM *** LOAD THE DRIVER AND SET-UP PARAMETERS
100 REM
110 LOAD "GPI.SPS"
120 IL=0
130 LA=32
140 TA=64
150 SA=96
160 FL=0
170 SIFT0 @IL,200
180 REM *** ENABLE WAVEFORM READABLE INTERRUPT ***
190 REM *** AND SET-UP WHEN FOR SRQ ***
200 WHEN @IL HAS "SRQ" GOSUB 650
210 PUT "WRI ON" INTO @IL,LA,SA
220 DELETE A,WA
230 PUT "REC?" INTO @IL,LA,SA
240 GET AS FROM @IL,TA,SA
250 NP=VAL(SEG(AS,POS(AS," ",1)+1,LEN(AS)-1))
260 WAVEFORM WA IS A(NP-1),DA,HA$,VA$
270 REM *** ACQUIRE ZERO REFERENCE ***
280 PUT "CPL?" INTO @IL,LA,SA+1
290 GET CS FROM @IL,TA,SA+1
300 PUT "CPL GND" INTO @IL,LA,SA+1
310 PUT "ARM A" INTO @IL,LA,SA
320 WAIT 100
330 PUT "MTRIG" INTO @IL,LA,SA
340 REM *** WAIT FOR WAVEFORM READABLE SRQ ***
350 IF FL=0 THEN 350
360 FL=0
370 PUT "READ A" INTO @IL,LA,SA
380 READBI A FROM @IL,TA,SA
390 ZR=MEA(A)
400 PUT CS INTO @IL,LA,SA+1
410 REM *** ACQUIRE WAVEFORM DATA ***
420 PUT "ARM A" INTO @IL,LA,SA
430 REM *** WAIT FOR WAVEFORM READABLE SRQ ***
440 IF FL=0 THEN 440
450 FL=0
460 PUT "READ A" INTO @IL,LA,SA
470 READBI A FROM @IL,TA,SA
480 REM *** ACQUIRE SCALE FACTORS AND NORMALIZE ***
490 HA$="S"
500 PUT "VSL1?" INTO @IL,LA,SA
510 GET SF$ FROM @IL,TA,SA
520 PO=POS(SF$," ",1)
530 SF=VAL(SEG(SF$,POS(SF$," ",1)+1,PO-1))
540 VA$=SEG(SF$,PO+1,PO+1)
550 REM *** ASSUME ONE SAMPLING INTERVAL ***
560 PUT "SBPT?" INTO @IL,LA,SA
570 GET AS FROM @IL,TA,SA
580 DA=VAL(SEG(AS,POS(AS," ",1)+1,LEN(AS)-1))
590 A=(A-ZR)*SF/32
600 PAGE
610 GRAPH WA
620 RELEASE "GPI.SPS"
630 END
640 REM *** SRQ SERVICE ROUTINE ***
650 POLL @IL,ST,PP,SS;TA,SA;TA,SA+1;TA,SA+2
660 IF PP<>TA THEN 700
670 IF SS<>SA THEN 700
680 VARTST ST,"4",FL
690 GOTO 650
700 RETURN
```

Fig. 3. Adding a few lines to the program in Fig. 1 eliminates the "dead" time on the GPIB during waveform acquisition.

command and waiting for the data, the program loops at line 350 waiting for a flag (FL) to be set (i.e., FL=0).

FL is zero (cleared) until an SRQ interrupt is received. When the interrupt is received, the subroutine in lines 640-710 is executed. The subroutine starts by polling each device whose addresses are listed in the POLL command arguments. The polling process stops with the first device asserting SRQ. The status byte, primary talk address, and secondary address of this device are returned in ST, PP, and SS, respectively. If none of the devices listed in the poll command are asserting SRQ, the status byte, primary address, and secondary address variables are set to zero.

Lines 660 and 670 check the primary and secondary addresses to see that the status byte is from the 7612D mainframe. If the addresses do not match, the subroutine returns without setting the flag. The VARTST command in line 670 tests bit 3 of the 7612D's status byte. If this bit is set, channel A caused a waveform readable interrupt, so the VARTST command sets the flag (FL) to one. If bit 3 of the status byte is a zero, the flag is cleared. The GOTO statement in line 690 polls the devices again to be sure that all SRQ's are serviced before returning to the main program.

When the waveform readable interrupt is received and the flag set, the test in line 440 fails, the flag is reset in line 450, and the program proceeds. Lines 460 and 470 read the data from channel A. The data transfer begins immediately since the acquisition is complete.

The remainder of the program is identical to the program in Fig. 1.

The high-level driver makes it easier

Now let's take a look at the same program using the optional high-level instrument driver (INS) for TEK SPS BASIC V02/V02XM. This driver provides simple "high-level" communication with Tektronix instruments. It handles much of the mechanics of controlling an instrument automatically.

The program shown in Fig. 4 performs the same functions as the original version—it's just a lot simpler because it uses the high-level driver. For example, the original program shown in Fig. 1 uses three statements to get the record length value (lines 190-210). Compare this to the single statement used in line 120 of the new program (Fig. 4). The high-level driver allows you to send a query, get the response, and separate the response into its component parts, all in one statement!

```

10 REM THIS IS A SIMPLE 7612D ACQUISITION ROUTINE.
20 REM IT USES THE HIGH-LEVEL INSTRUMENT DRIVER (INS)
30 REM FOR TEK SPS BASIC V02-02/XM. THE ROUTINE ASSUMES
40 REM THAT THE INSTRUMENT IS SET-UP TO ACQUIRE ONE
50 REM RECORD WITH A SINGLE SAMPLING INTERVAL.
60 REM IT ALSO ASSUMES THAT A SINGLE PROGRAMMABLE PLUG-IN
70 REM IS INSTALLED IN CHANNEL A OF THE INSTRUMENT.
80 REM
90 LOAD "INS.SPS"
100 ATTACH #1 AS INS0,0:WITH 1 @0
110 DELETE A,WA
120 GET NR,NP FROM #1,"REC?"
130 WAVEFORM WA IS A(NP-1),DA,HA$,VA$
140 REM *** ACQUIRE ZERO REFERENCE ***
150 GET CS FROM #1;1,"CPL?"
160 PUT "CPL GND" INTO #1;1
170 PUT "ARM A" INTO #1
180 WAIT 100
190 GET A FROM #1,"MTRIG;READ A"
200 ZR=MEA(A)
210 PUT CS INTO #1;1
220 REM *** ACQUIRE WAVEFORM DATA ***
230 GET A FROM #1,"ARM A;READ A"
240 REM *** ACQUIRE SCALE FACTORS AND NORMALIZE ***
250 HA$="S"
260 GET SF$,SF,VA$ FROM #1,"VSL1?"
270 VA$=SEG(VA$,1,1)
280 REM *** ASSUME ONE SAMPLING INTERVAL ***
290 GET BP,DA FROM #1,"SBPT?"
300 A=(A-ZR)*SF/32
310 PAGE
320 GRAPH WA
330 DETACH #1
340 RELEASE "INS.SPS"
350 END

```

Fig. 4. *Converting the basic acquisition routine to use the optional high-level instrument driver (INS) simplifies the program considerably.*

The ATTACH statement in line 100 specifies an ILUN (instrument logical unit number) which will be used to reference the instrument. This statement links the ILUN to the primary and secondary addresses of the instrument, the secondary addresses of its plug-ins, and the interface number to which it is connected. After the ATTACH statement is executed, only the ILUN need be specified in a reference to the instrument; TEK SPS BASIC automatically addresses the instrument using the addresses specified in the ATTACH.

From here, the program performs the same basic functions as the previous two: It sets-up the waveform arrays, acquires the zero reference, data, and scale factors, normalizes the data, and graphs the results.

The best of both—interrupts and the high-level driver

Though the program in Fig. 4 is a lot simpler than its equivalent in Fig. 1, it still has the same waiting limitation—the bus is tied up while the 7612D completes acquisition. Fortunately, modifying the program to use the 7612D's

Simple approaches...

waveform readable interrupt feature is a simple task. Figure 5 shows the modified program using the high-level driver. The new parts are shown in bold print.

Again, the basic operations are the same as the program in Fig. 4. The program gets under way by initializing the FL flag, loading the driver, and attaching the instrument. Next, the automatic

```
10 REM THIS IS A SIMPLE INTERRUPT-DRIVEN 7612D
20 REM ACQUISITION ROUTINE. IT USES THE HIGH-LEVEL
30 REM INSTRUMENT DRIVER (INS) FOR TEK SPS BASIC V02-02/XM.
40 REM THE ROUTINE ASSUMES THAT THE INSTRUMENT IS SET-UP TO
50 REM ACQUIRE ONE RECORD WITH A SINGLE SAMPLING INTERVAL.
60 REM IT ALSO ASSUMES THAT A SINGLE PROGRAMMABLE PLUG-IN IS
70 REM INSTALLED IN CHANNEL A OF THE INSTRUMENT.
80 REM
90 REM *** LOAD THE DRIVER AND ATTACH THE INSTRUMENT ***
100 REM
110 FL=0
120 LOAD "INS.SPS"
130 ATTACH #1 AS INS0,0:WITH 1 @0
140 REM *** ENABLE WAVEFORM READABLE INTERRUPT ***
150 REM *** AND SET-UP WHEN FOR SRQ ***
160 LOCKSRQ
170 WHEN #1 HAS "307" GOSUB 550
180 WHEN #1 HAS "SRQ" GOSUB 560
190 SRQENABLE @0
200 PUT "WRI ON" INTO #1
210 DELETE A,WA
220 GET NR,NP FROM #1,"REC?"
230 WAVEFORM WA IS A(NP-1),DA,HA$,VA$
240 REM *** ACQUIRE ZERO REFERENCE ***
250 GET C$ FROM #1;1,"CPL?"
260 PUT "CPL GND" INTO #1;1
270 PUT "ARM A" INTO #1
280 WAIT 100
290 PUT "MTRIG" INTO #1
300 REM *** WAIT FOR WAVEFORM READABLE SRQ ***
310 IF FL=0 THEN 310
320 FL=0
330 GET A FROM #1,"READ A"
340 ZR=MEA(A)
350 PUT C$ INTO #1;1
360 REM *** ACQUIRE WAVEFORM DATA ***
370 PUT "ARM A" INTO #1
380 REM *** WAIT FOR WAVEFORM READABLE SRQ ***
390 IF FL=0 THEN 390
400 FL=0
410 GET A FROM #1,"READ A"
420 REM *** ACQUIRE SCALE FACTORS AND NORMALIZE ***
430 HA$="S"
440 GET SF$,SF,VA$ FROM #1,"VSLI?"
450 VA$=SEG(VA$,1,1)
460 REM *** ASSUME ONE SAMPLING INTERVAL ***
470 GET BP,DA FROM #1,"SBPT?"
480 A=(A-ZR)*SF/32
490 PAGE
500 GRAPH WA
510 DETACH #1
520 RELEASE "INS.SPS"
530 END
540 REM *** WRI SRQ SERVICE ROUTINE ***
550 FL=1
560 SRQENABLE @0
570 RETURN
```

Fig. 5. Adding interrupt handling to the high-level program uses the best of both worlds—the simplicity of the high-level driver and the efficiency of interrupt-driven acquisition.

SRQ re-enabling feature of the high-level driver is disabled with the LOCKSRQ statement. This statement keeps the driver from recognizing and processing an SRQ interrupt until the previous one has been serviced.

Then, two interrupt conditions are set-up—one for the specific SRQ of interest (waveform readable), and one for all others. The first WHEN statement in line 170 causes control to be transferred to line 550 when the 7612D reports a waveform readable interrupt from channel A. Line 550 sets the FL flag to 1 and SRQs are re-enabled in line 560. Finally, line 570 returns control to the main program at the point where it was interrupted by the waveform readable interrupt.

The second WHEN statement (line 180) handles all other interrupt conditions from instrument number 1. These other interrupts do not affect the main program, so the SRQ is simply re-enabled without setting FL.

Back in the main program, lines 240-290 set the instrument up to acquire the zero reference, arm the time base, and manually trigger it. Then, the program loops at line 310 waiting for FL to be set. When the interrupt occurs, execution of the main program is temporarily suspended, the interrupt is serviced, and FL is set. When the main program begins executing again, the IF statement at line 310 finds FL set, and program execution continues to line 320. The same procedure is used to acquire the waveform data in lines 370-410.

Choose your driver and build a program

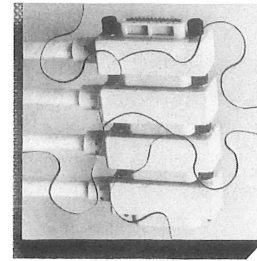
The routines in Fig. 1-5 handle only the simplest cases, but they do provide a basis for building a complete program suited to your needs. Just a word of caution—be careful in making your selection of which driver to use. The high-level driver is hard to beat in small systems of Tektronix instruments. But don't neglect the low-level driver! It still comes out on top for larger systems or systems with instruments from other manufacturers.

In any case, a good acquisition routine using the right driver can provide the foundation for a powerful, flexible automated measurement system.



By Mark Tilden,
HANDSHAKE Staff

Solving the GPIB Puzzle:



CG551AP and TEK SPS BASIC make auto cal easy

Until now, most of the examples presented in HANDSHAKE for controlling GPIB instruments with TEK SPS BASIC have involved data acquisition. But, TEK SPS BASIC is equally adept at controlling programmable signal sources for test stimuli or automatic calibration. A good example is controlling the new Tektronix CG551AP Programmable Calibration Generator.

Primarily designed as a programmable calibrator for oscilloscopes, the CG551AP generates reference signals for testing vertical gain, horizontal timing and gain, vertical bandwidth, transient response, probe accuracy and compensation, and calibrator output accuracy. In addition, it is fully programmable over the IEEE-488 bus (GPIB)—every front-panel setting can be remotely programmed. (For more

details see **A closer look at the CG551AP** in this issue.)

Full programmability and high accuracy also make the CG551AP a good choice for automated calibration of waveform digitizers. When the digitizer's settings are programmable, the calibration process can be completely automatic. Non-programmable instruments may require some manual interaction.

With the signal processing capabilities and instrument control power of TEK SPS BASIC, implementing a high-speed, fully automatic calibration system becomes a simple task. Using programmable digitizers, such as the Tektronix 7912AD or 7612D, the measurement system can be automatically configured for calibration, the tests

continued on page 19

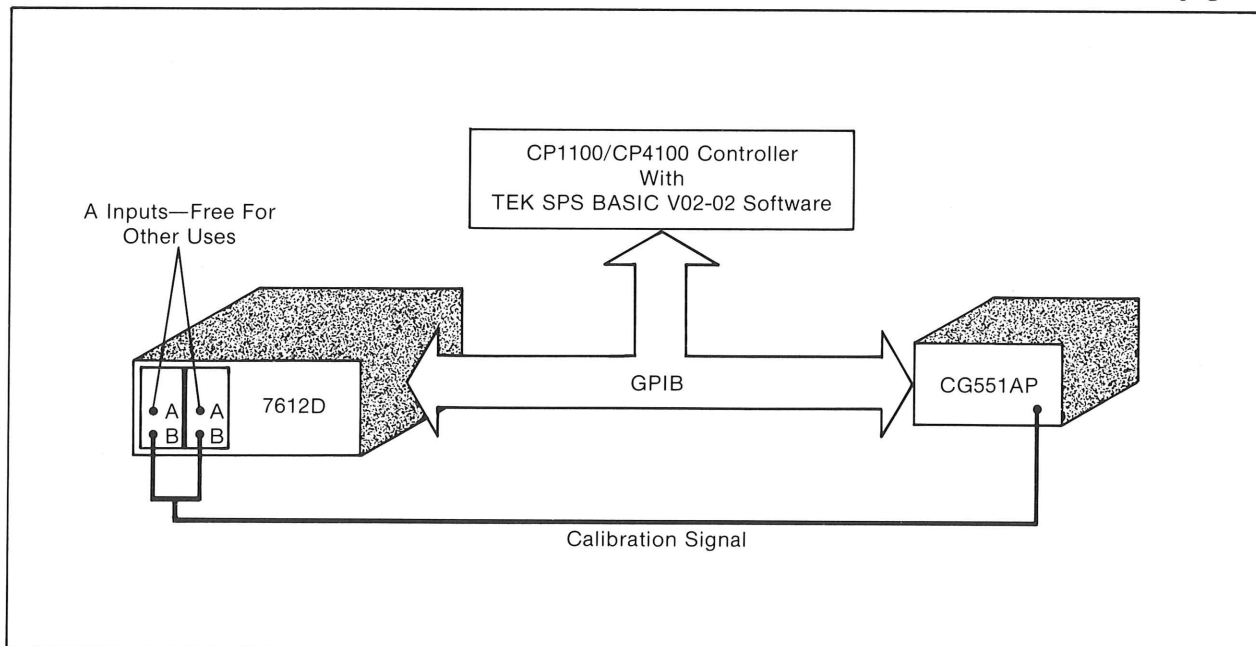


Fig. 1. A simple automatic calibration system for checking low-frequency vertical gain.

A closer look at the CG551AP

Interested in faster, more accurate oscilloscope calibration?

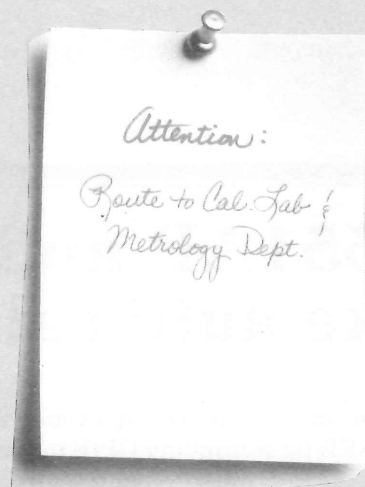
Then take a look at the TEKTRONIX CG551AP Programmable Calibration Generator. It has a complete list of oscilloscope calibration features including a 40-microvolt standard amplitude and slewed-edge timing markers for high-speed accuracy. Plus, it's a fully programmable, microprocessor-based generator designed to give you a choice of manual operation or fully automatic operation under computer control for calibration and verification of major oscilloscope parameters such as:

- Vertical gain
- Horizontal timing and gain
- Vertical bandwidth/pulse characteristics
- Probe accuracy and compensation
- Current probe accuracy
- Calibrator output accuracy

Why computer-aided calibration?

The CG551AP is GPIB compatible—it conforms to IEEE 488-1978. This means that it can be integrated into systems and used along with other GPIB instruments. A typical system is illustrated in the accompanying diagram and has the advantages of letting you program and automatically document your calibration procedures.

Faster calibration, fewer errors. Compared to conventional calibration set ups, the CG551AP under program control provides up to a fourfold increase in calibration speed. Not only can you increase throughput with the CG551AP, but you can make calibration more complete and consistent. Because the calibration is directed by computer program, complete with operator prompts if you wish, procedures are executed the same every time. This significantly reduces the chances of an operator overlooking a calibration step or inadvertently missetting a control.



Also, calibration error is automatically computed by the CG551AP's microprocessor. This removes another burden for the operator and ensures faster, more accurate, and more repeatable results. Additionally, limits of error can be included as part of the calibration program and used for pass-fail comparison with calibration results. Again, the operator is relieved of the task of continually comparing results to a specification sheet. Software can do it automatically.

Automatic documentation. Calibration results can also be documented automatically. Software can tabulate the calibration results and output them to a line printer or graphic terminal for conversion to a paper document. This eliminates time consuming hand recording of results as well as the transposition and omission errors common to hand prepared documentation.

The calibration results can also be archived on magnetic storage media, such as tape or disk. This provides a readily accessible data base for statistical analyses, instrument control, traceability, etc.

Building your program

There are a variety of ways to put the CG551AP to work. One is to design your own system and write the necessary software. Interfacing the system is simple since the CG551AP is GPIB compatible, so writing the software will be the greatest task.

If you would rather avoid the software design task, the TEKTRONIX SCPDA1 ScopeCal Procedure Development Aid is the answer. This software runs on a TEKTRONIX 4050-Series

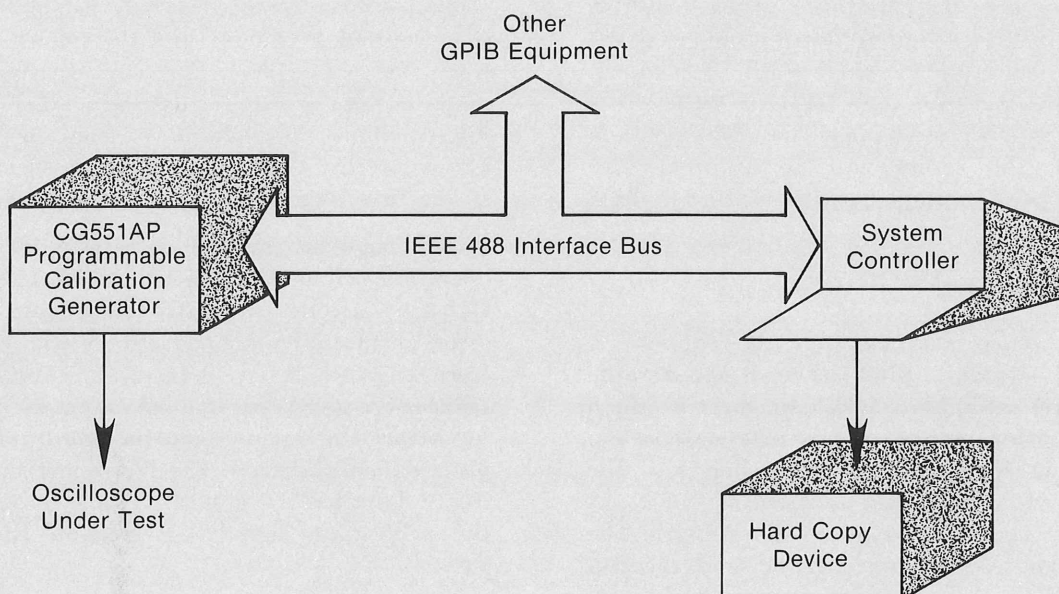



Fig. 1. Typical computer-controlled calibration system.

Graphic Computing System and provides a tool for developing your own calibration procedures. SCPDA1 guides you step-by-step through a calibration process, allowing you to enter test parameters, test limits, operator instructions, and control settings as you need them. It also automatically acquires and stores the CG551AP settings. As a result, you can develop automatic, interactive calibration procedures without any previous programming experience.

The F5165D3 Scope Calibration System is another way to get underway with a programmable calibration system. It provides everything required for setting up a programmable oscilloscope calibration system based on the CG551AP. The system includes the CG551AP Programmable Calibration Generator,

4052 Graphic Computing System, TM506 MOD JB Mainframe, SG502 Sine-Wave Generator, SG503 Sine-Wave Generator, SG504 Sine-Wave Generator, and the SCPDA1 ScopeCal Procedure Development Aid Software plus supporting accessories and operating instructions for the system.

Want more information?

This has only been a glimpse of what the CG551AP can do to automate your oscilloscope calibration procedures. For further information on the CG551AP or a demonstration at your calibration lab, check the appropriate box on the HANDSHAKE Reply Card. 

*By Dale Aufrecht,
HANDSHAKE Staff*

continued from page 17

performed, results logged, and the system returned to normal operation—all without any human intervention. The result is higher accuracy and dependable calibration without removing the instruments from their working environment and without time-consuming manual calibration.

The controller can even use the test results to automatically correct acquired data for calibration error!

To explore the auto cal possibility further, consider the process of vertical auto cal. Figure 1 shows a simple automatic calibration system using the CG551AP and a single 7612D Programmable Digitizer with 7A16P Programmable plug-ins. The calibration signal is applied to the B inputs of each plug-in. The A inputs are free for normal use. Since the 7A16P's A and B inputs are very well matched, the calibration signal follows essentially the same

path as a normal input signal. Errors introduced while acquiring the calibration signal will also show up in other acquired data. By comparing the acquired calibration signal with the known standard output from the CG551AP, these errors can be computed and stored. Then, when data is acquired, the errors can be automatically corrected with a simple routine written in TEK SPS BASIC.

Software tweaks the data

For example, assume that the channel A vertical system (plug-in and mainframe amplifiers) exhibits a +1.3% gain error on the 5 volts/division range. When the calibration routine is executed, the controller sets the CG551AP to generate six divisions of amplitude-calibrated square wave at 5 volts/division. The 7612D acquires this waveform and sends the data

to the controller memory. Then, the controller computes the difference between the amplitude of the acquired waveform and the known output from the CG551AP. For our example, the amplitude of the acquired data will be $1.3\% \pm 0.25\%$ higher than the output of the calibration generator (the CG551AP output is accurate to $\pm 0.25\%$ in voltage mode).

Data acquired through channel A in normal operation will also be affected by this 1.3% error. But, the controller can easily compensate for this error by multiplying each data value acquired from channel A by 0.987 ($1 - 1.3\%/100$). This reduces the gain error to a maximum of $\pm 0.25\%$ —far below the normal plug-in/mainframe gain accuracy specification. The flow chart shown in Fig. 2 describes the program required to acquire the calibration signal and compute the error values.

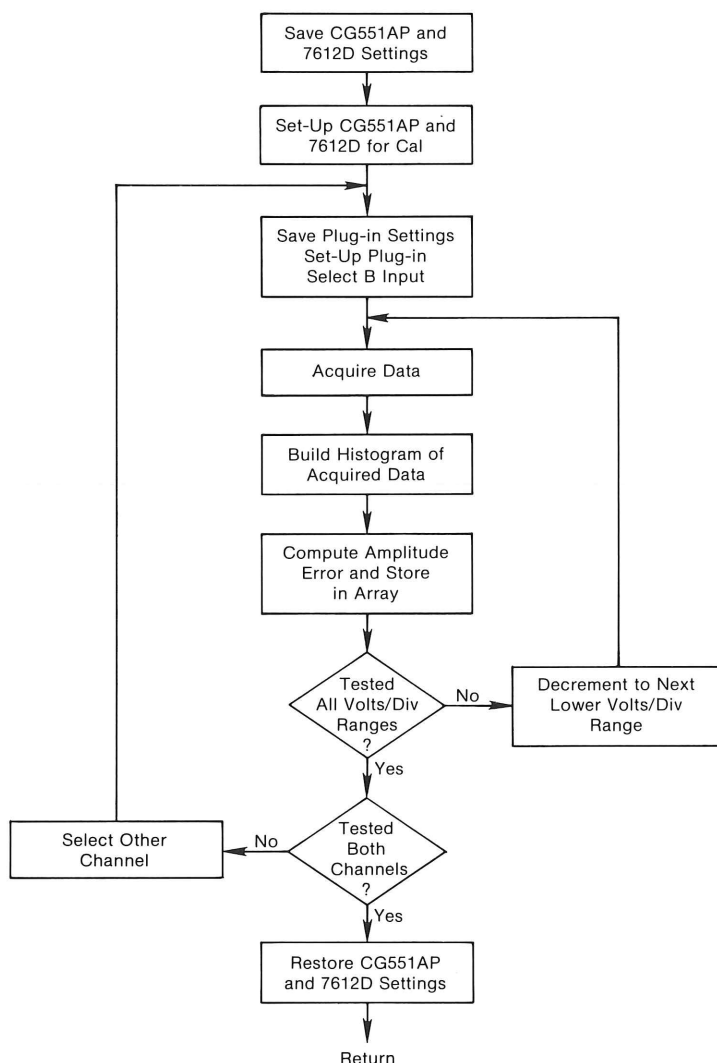


Fig. 2. Flow chart of a routine to test low-frequency gain calibration.

Getting down to the BASIC

With this basic auto-cal concept in mind, let's look at a TEK SPS BASIC program that uses the CG551AP to implement the simplest of automatic calibration procedures—a vertical gain check. The program shown in Fig. 3 checks the low-frequency vertical gain of a 7612D/7A16P system. The routine returns an array of error values, one value for each VOLTS/DIV setting on the 7A16P.

```

10 REM *****
20 REM CG551AP VERTICAL AUTO CAL ROUTINE FOR THE 7612D
30 REM *****
40 DELETE VD$,CS$,L$,QQ
50 DIM VD$(8),CS$(2),L$(2),PE(8,1),QQ(511)
60 VD$(0)="5"\VD$(1)="2"\VD$(2)="1"\VD$(3)="5"\VD$(4)="2"
70 VD$(5)="1"\VD$(6)="05"\VD$(7)="02"\VD$(8)="01"
80 CS$(1)="A"\CS$(2)="B"\L$(1)="L"\L$(2)="R"
90 LOAD "INS.SPS"
100 ATTACH #1 AS INS7:@0
110 ATTACH #2 AS INS0,0:WITH 1 @0
120 GET CS$ FROM #1,"SET?"
130 PUT "MODE V;MAG X1;MULT 6;CHOP ON;OUT ON;FXD" INTO #1
140 GET MS$ FROM #2,"SET?"
150 FOR C=1 TO 2
160 GET PS$ FROM #2,C,"SET?"
170 PUT "IMBS ";CS$(C);";WRI ON;CLK INT;BTA OFF;REC 1,512" INTO #2
180 PUT "SBPT 0,10E-6;LTC ";L$(C);";LEV 10" INTO #2
190 PUT "POS -3;POL NOR;VAR OFF;CPL DC;INP B" INTO #2;C
200 FOR I=0 TO 8
210 PUT "U/D "&VD$(I) INTO #1
220 PUT "V/D "&VD$(I) INTO #2;C
230 GET QQ FROM #2,"ARM "&CS$(C)&";READ "&CS$(C)
240 REM *****
250 REM GENERATE HISTOGRAM FROM ACQUIRED DATA.
260 REM COMPUTE AMPLITUDE AND PERCENT ERROR.
270 REM *****
280 VR=VAL(VD$(I))*6
290 DELETE B
300 DIM B(255)
310 FOR J=0 TO 255
320 B(QQ(J))=B(QQ(J))+1
330 NEXT J
340 CE=128
350 V1=CRS(B(0:CE),MAX(B(0:CE)))
360 V2=CRS(B(CE:255),MAX(B(CE:255)))
370 VM=(V2-V1)*VR/(6*32)
380 PE(I,C-1)=(VM-VR)*100/VR
390 NEXT I
400 PUT PS$ INTO #2;C
410 NEXT C
420 PUT MS$ INTO #2
430 PUT CS$ INTO #1
440 DETACH #1
450 DETACH #2
460 RELEASE "INS.SPS"
470 FOR C=1 TO 2
480 PRINT
490 PRINT "CHANNEL ";CS$(C)
500 FOR I=0 TO 8
510 PRINT "ERROR FOR ";VD$(I);" VOLTS/DIV RANGE IS: ";PE(I,C-1);"%
520 NEXT I
530 PRINT
540 NEXT C
550 END

```

Fig. 3. A simple automatic calibration program to check the low-frequency vertical gain of a 7612D/7A16P system.

This program uses the optional high-level instrument driver (INS) of SPS BASIC V02-02. This driver is designed to provide simple, "high

level" communication with Tektronix instruments. (For a complete discussion and comparison of the high-level and low-level drivers, see "Simple approaches to controlling the 7612D" in this issue). The driver handles SRQ interrupts automatically, provides a simple method of addressing, and a number of other features that simplify control of a GPIB-interfaced instrument.

Getting under way, the program initializes variables and loads the high-level driver into memory. String array VD\$ stores the volts/division settings for the plug-ins and CG551AP. Because the format for the plug-ins and calibration generator both conform to a Tektronix standard for GPIB message format, the same string can be used to set the range for both instruments.

Lines 100 and 110 "attach" the instruments. These commands tell the software what the bus address and interface number is for each instrument. (This program assumes address 7 for the CG551AP and 0 for the 7612D). After the instruments are attached, they can be referred to by the specified ILUN (Instrument Logical Unit Number)—#1 for the CG551AP and #2 for the 7612D. Since the instruments are probably set up to perform a specific measurement before the calibration routine is executed, lines 120, 150, and 160 get and store the current instrument settings. These settings are restored when the calibration routine is completed.

Line 130 sets up the CG551AP to generate a six-division amplitude-calibrated square wave. Then, line 140 begins a two-pass FOR loop. On the first pass, the routine checks the calibration of channel A of the 7612D. The second pass checks channel B. Lines 170 and 180 set up the 7612D to acquire the calibration waveform, and line 190 sets up the plug-in.

Next, the program begins another loop that successively sets the plug-in to each VOLTS/DIV range, and sets the CG551AP to generate a six-division square wave for each setting. On each pass, line 230 acquires the waveform, and lines 290 through 370 convert the data to a histogram. The histogram is used to find the upper and lower limits of the square wave while rejecting noise and overshoot. Finally, line 380 computes the difference between the known output from the generator and the amplitude of the acquired data. This quantity is converted to a percentage of error and stored in the array PE. The loop is repeated for each of the nine VOLTS/DIV ranges, and the

error values are stored in corresponding elements of the PE array.

When all nine ranges have been tested, the channel A settings are returned to their original values, and channel B is tested. Then, the CG551AP settings are restored, both instruments are detached, and the driver is released from memory. The loop in lines 480-550 prints and labels the error values. These errors need not be printed, since they can be used to automatically correct acquired data. Alternatively, the error values could be printed only when they exceed some specified limit.

Though this isn't a complete automatic calibration routine, it could be used as a subroutine in a complete auto-cal program or in an application program that requires high gain accuracy. The concept could also be extended to check and compensate for sampling inaccuracy, nonlinearity, and even bandwidth. With programmable instruments, like the CG551AP and 7612D, and powerful software, like TEK SPS BASIC, to link them together, the step up to full automatic calibration can be an easy one.



By Mark Tilden,
HANDSHAKE Staff

Literature available from Tektronix

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

GPIO Communication with the 7854, Application Note AX-4416.

This application note demonstrates interfacing of the 7854 over the GPIO 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.

Capture fast waveforms accurately with a 2-channel programmable digitizer, *Electronic Design* reprint, AX-4401.

This 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.

Spectrum Analysis Systems, AX-4011.

The Spectrum Analyzer has long been recognized as the most accurate and versatile instrument for making a wide variety of RF component measurements. But what about documentation of the results? What about the computations involved? This concept note describes how a Digitizing Oscilloscope system can be used to provide automatic spectrum acquisition, analysis, and report generation.

An Overview of Disk System Testing, TN-0005.

Waveform processing systems offer greater speed and repeatability for evaluating and testing magnetic disk systems. This technical note provides an overview of the system and some of the tests that can be made with a waveform processing system.

HANDSHAKE Applications Library Catalog.

The latest edition of the HANDSHAKE Applications Library Catalog is now available. Besides carrying a new name, this catalog lists ten new programs and includes a listing of Application Notes and Technical Notes that are applicable to signal processing and instrument control.

ASCII & GPIB CODE CHART

B7 B6 B5 BITS		$\begin{smallmatrix} \text{0} & \text{0} & \text{0} \\ \text{0} & \text{0} & \text{0} \end{smallmatrix}$		$\begin{smallmatrix} \text{0} & \text{0} & \text{1} \\ \text{0} & \text{0} & \text{1} \end{smallmatrix}$		$\begin{smallmatrix} \text{0} & \text{1} & \text{0} \\ \text{0} & \text{1} & \text{1} \end{smallmatrix}$		$\begin{smallmatrix} \text{1} & \text{0} & \text{0} \\ \text{1} & \text{0} & \text{1} \end{smallmatrix}$		$\begin{smallmatrix} \text{1} & \text{1} & \text{0} \\ \text{1} & \text{1} & \text{1} \end{smallmatrix}$		$\begin{smallmatrix} \text{1} & \text{1} & \text{1} \\ \text{1} & \text{1} & \text{1} \end{smallmatrix}$	
B4 B3 B2 B1		CONTROL		NUMBERS SYMBOLS		UPPER CASE		UPPER CASE		LOWER CASE		LOWER CASE	
$\begin{smallmatrix} \text{0} & \text{0} & \text{0} & \text{0} \\ \text{0} & \text{0} & \text{0} & \text{0} \end{smallmatrix}$		0 NUL	20 DLE	40 SP	60 0	16 @	0 P	140 ,	160 p				
$\begin{smallmatrix} \text{0} & \text{0} & \text{0} & \text{1} \\ \text{0} & \text{0} & \text{0} & \text{1} \end{smallmatrix}$		1 SOH	21 DC1	41 !	61 1	101 A	121 Q	141 a	161 q				
$\begin{smallmatrix} \text{0} & \text{0} & \text{1} & \text{0} \\ \text{0} & \text{0} & \text{1} & \text{0} \end{smallmatrix}$		2 STX	22 DC2	42 "	62 2	102 B	122 R	142 b	162 r				
$\begin{smallmatrix} \text{0} & \text{0} & \text{1} & \text{1} \\ \text{0} & \text{0} & \text{1} & \text{1} \end{smallmatrix}$		3 ETX	23 DC3	43 #	63 3	103 C	123 S	143 c	163 s				
$\begin{smallmatrix} \text{0} & \text{1} & \text{0} & \text{0} \\ \text{0} & \text{1} & \text{0} & \text{0} \end{smallmatrix}$		4 EOT	24 DC4	44 \$	64 4	104 D	124 T	144 d	164 t				
$\begin{smallmatrix} \text{0} & \text{1} & \text{0} & \text{1} \\ \text{0} & \text{1} & \text{0} & \text{1} \end{smallmatrix}$		5 ENQ	25 NAK	45 %	65 5	105 E	125 U	145 e	165 u				
$\begin{smallmatrix} \text{0} & \text{1} & \text{1} & \text{0} \\ \text{0} & \text{1} & \text{1} & \text{0} \end{smallmatrix}$		6 ACK	26 SYN	46 &	66 6	106 F	126 V	146 f	166 v				
$\begin{smallmatrix} \text{0} & \text{1} & \text{1} & \text{1} \\ \text{0} & \text{1} & \text{1} & \text{1} \end{smallmatrix}$		7 BEL	27 ETB	47 '	67 7	107 G	127 W	147 g	167 w				
$\begin{smallmatrix} \text{1} & \text{0} & \text{0} & \text{0} \\ \text{1} & \text{0} & \text{0} & \text{0} \end{smallmatrix}$		10 BS	30 CAN	50 (70 8	110 H	130 X	150 h	170 x				
$\begin{smallmatrix} \text{1} & \text{0} & \text{0} & \text{1} \\ \text{1} & \text{0} & \text{0} & \text{1} \end{smallmatrix}$		11 HT	31 EM	51)	71 9	111 I	131 Y	151 i	171 y				
$\begin{smallmatrix} \text{1} & \text{0} & \text{1} & \text{0} \\ \text{1} & \text{0} & \text{1} & \text{0} \end{smallmatrix}$		12 LF	32 SUB	52 *	72 :	112 J	132 Z	152 j	172 z				
$\begin{smallmatrix} \text{1} & \text{0} & \text{1} & \text{1} \\ \text{1} & \text{0} & \text{1} & \text{1} \end{smallmatrix}$		13 VT	33 ESC	53 +	73 ;	113 K	133 [153 k	173 {				
$\begin{smallmatrix} \text{1} & \text{1} & \text{0} & \text{0} \\ \text{1} & \text{1} & \text{0} & \text{0} \end{smallmatrix}$		14 FF	34 FS	54 ,	74 <	114 L	134 \	154 l	174 				
$\begin{smallmatrix} \text{1} & \text{1} & \text{0} & \text{1} \\ \text{1} & \text{1} & \text{0} & \text{1} \end{smallmatrix}$		15 CR	35 GS	55 -	75 =	115 M	135]	155 m	175 }				
$\begin{smallmatrix} \text{1} & \text{1} & \text{1} & \text{0} \\ \text{1} & \text{1} & \text{1} & \text{0} \end{smallmatrix}$		16 SO	36 RS	56 .	76 >	116 N	136 ^	156 n	176 ~				
$\begin{smallmatrix} \text{1} & \text{1} & \text{1} & \text{1} \\ \text{1} & \text{1} & \text{1} & \text{1} \end{smallmatrix}$		17 SI	37 US	57 /	77 ?	117 O	137 _	157 o	177 DEL (RUBOUT)				

KEY

octal	25	PPU	GIPIB code
	NAK		ASCII character
hex	15	21	decimal

Tektronix®
COMMITTED TO EXCELLENCE

REF: ANSI STD X3. 4-1977
IEEE STD 488-1978
ISO STD 646-1973

TEKTRONIX STD 062-5435-00 4 SEP 80
COPYRIGHT © 1979, 1980 TEKTRONIX, INC. ALL RIGHTS RESERVED.

Getting the most out of TEK BASIC graphics

Graphing data in log and log-log formats

Most array data, particularly waveform data, can be conveniently displayed in the linear format of the GRAPH command provided with TEK SPS BASIC software. However, there still remain some cases where a log or log-log display of data is more appropriate. This is particularly true in dealing with frequency-domain data where 3-dB points, relative attenuation, and roll-offs are of interest. For these cases, it is worthwhile to explore several steps leading up to a log-log display.

Figure 1 shows a time-domain display of pulse data in the linear format provided by the GRAPH command of TEK SPS BASIC. Here, the linear format is adequate for revealing the majority of detail associated with the data. The frequency-domain magnitude of this pulse, however, drops rapidly from its maximum to some low-level side lobes. As shown in Fig. 2, these side lobes are not entirely visible. For such cases, a log-amplitude display reveals more of the low-level detail, as can be seen in Fig. 3, while also expressing amplitude in decibels, which is a common requirement in frequency-domain analysis.

Such log-amplitude or decibel displays are easily achieved with TEK SPS BASIC software. For example, let's say that the frequency-domain magnitude data in Fig. 2 is stored as a waveform that has been defined by

```
WAVEFORM B IS Z(256),SB,HB$,VB$.
```

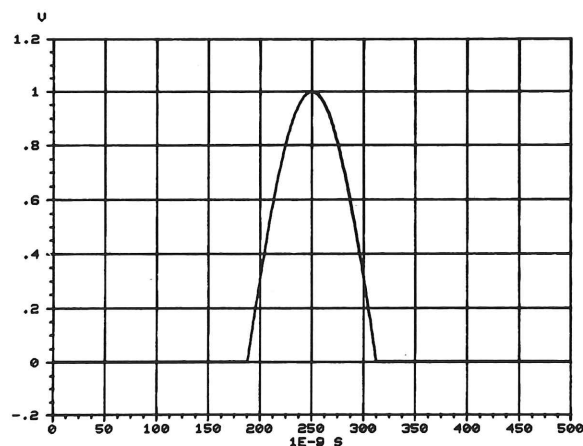


Fig. 1. Time-domain pulse graphed on a linear scale.

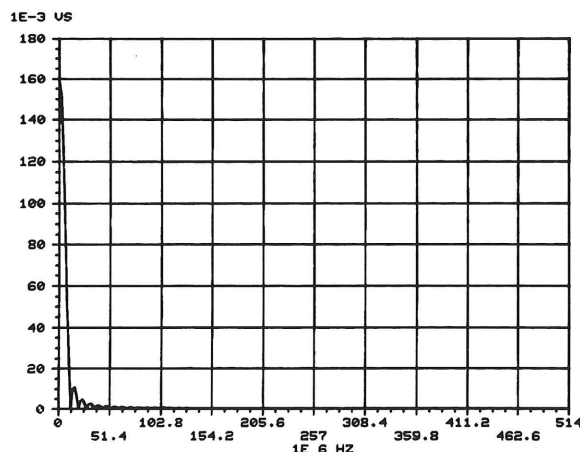


Fig. 2. Linear graph of frequency-domain magnitude of the pulse shown in Fig. 1.

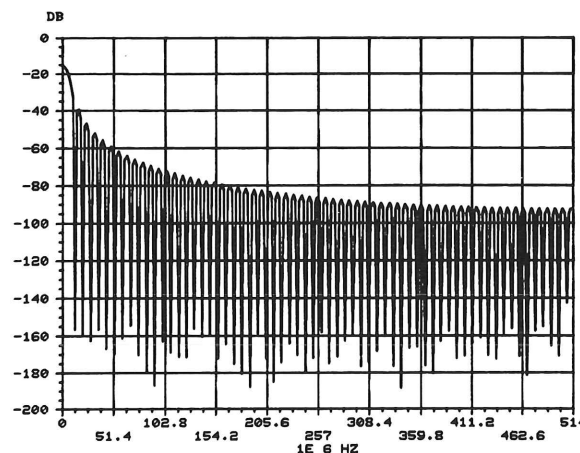


Fig. 3. Log-amplitude graph of frequency-domain magnitude.

WAVEFORM B can be converted to decibels with the following two TEK SPS BASIC statements:

```
Z=20*LOG(Z)/LOG(10)
VB$="DB"
```

The LOG function used in the first statement is the natural log, hence the need to convert to base ten by dividing by LOG(10). The second statement simply specifies DB (decibels) for the vertical units.

There is a point of caution that should be observed when converting data arrays to decibels: the LOG function is not valid for zero. If it is called to operate on a zero-valued array element, it will return a value of zero and cause a warning error to be issued. So it is wise to check arrays before decibel conversion for zero values and either change the elements to an acceptable nonzero value for conversion or ignore the conversion results for zero-valued elements.

After log amplitude, the next level of display is to convert the horizontal or frequency axis from a linear to a log scale. Doing this for the log-amplitude data of Fig. 3 results in the display shown in Fig. 4. This type of display is useful, for example, in documenting or examining roll-offs in terms of decibels per octave.

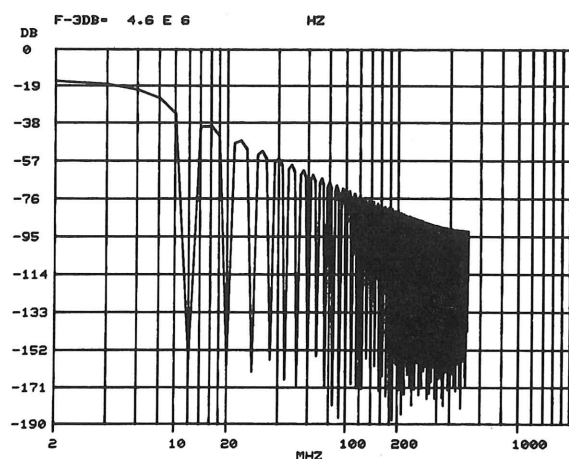


Fig. 4. Log-log plot of frequency-domain magnitude.

Unfortunately, converting to a log horizontal scale is not as straightforward as the decibel conversion. However, the task can be done with the program listed in Fig. 5. This program, in conjunction with the engineering notation program in Fig. 6, was used to produce the log-log display in Fig. 4.

The program in Fig. 5 assumes that the array of data to be plotted in log-log format has been defined as

WAVEFORM B IS Z(256),SB,HB\$,VB\$

and is the result of computing log magnitude after doing a 512-point fast Fourier transform. The routine scales the dB axis to the next higher value of ten, and the frequency axis spans three decades to accommodate the 257-point frequency function resulting from the 512-point transform.

```

2000 VIEWPORT 100,900,100,700
2010 PAGE
2020 T=MAX(Z)\REM *** FINDS MAX OF DB VS. FREQ. ARRAY ***
2030 L=ABS(MIN(Z))\REM *** FINDS ABS(MIN) OF SAME ***
2040 REM *** THE NEXT 2 STEPS ROUND UP TO THE NEAREST 10 ***
2050 T=ITP(T/10+.9)*10
2060 L=ITP(L/10+.9)*10
2070 L=L\REM *** NEED MINUS TO DEFINE BOTTOM OF WINDOW ***
2080 M=(T-L)/10\REM *** DIVIDES TOTAL RANGE INTO 10 PARTS ***
2090 WINDOW LOG(1)/LOG(10),3,L,T
2100 SMOVE 100,740\REM *** MOVE TO TOP OF GRAPH AREA ***
2110 PRINT "F-3DB= ";R1;" E";-N,HB$\REM * PRINT -3DB VALUE *
2120 K=1\REM *** SET UP FOR FIRST DECADE ***
2130 GOSUB 2250
2140 K=10\REM *** SET UP FOR SECOND DECADE ***
2150 GOSUB 2250
2160 K=100\REM *** SET UP FOR THIRD DECADE ***
2170 GOSUB 2250
2180 REM *** THE NEXT 3 STEPS DRAW HORIZONTAL LINES ***
2190 FOR I=0 TO 10
2200 MOVE 3,-M*I+T
2210 DRAW LOG(1)/LOG(10),-M*I+T
2220 NEXT I
2230 GOTO 2380
2240 REM *** THE NEXT 7 STEPS DRAW VERTICAL LINES ***
2250 FOR I=K TO 9*K STEP K
2260 REM *** THE NEXT STEP PROVIDES LONGER FIRST & FIFTH LINES ***
2270 GOTO I/K OF 2280,2300,2300,2300,2280,2300,2300,2300,2300
2280 L1=L-M/5
2290 GOTO 2310
2300 L1=L
2310 MOVE LOG(1)/LOG(10),T
2320 DRAW LOG(1)/LOG(10),L1
2330 NEXT I
2340 MOVE 3,T
2350 DRAW 3,L
2360 RETURN
2370 REM *** THE NEXT 3 STEPS PLOT DB VS. LOG FREQ. ***
2380 MOVE LOG(1)/LOG(10),Z(1)
2390 FOR I=2 TO 256
2400 DRAW LOG(1)/LOG(10),Z(I)
2410 NEXT I
2420 SMOVE 45,720\REM *** MOVE TO TOP LEFT ***
2430 PRINT "DB"
2440 REM *** THE NEXT 3 STEPS PRINT DB VALUES ON THE LEFT ***
2450 FOR I=0 TO 10
2460 SMOVE 35,692-I*60
2470 PRINT -I*M+T
2480 NEXT I
2490 K=1\REM *** SET UP FIRST DECADE ***
2500 GOSUB 2610
2510 K=10\REM *** SET UP SECOND DECADE ***
2520 GOSUB 2610
2530 K=100\REM *** SET UP THIRD DECADE ***
2540 GOSUB 2610
2550 SMOVE 480,40\REM *** MOVE TO BOTTOM CENTER ***
2560 PRINT "MHZ"
2570 RETURN
2580 REM *** THE NEXT 7 STEPS PRINT ROUNDED MHZ LABELS AT 1,5,10,
2590 REM *** 50,100,500 TIMES THE SAMPLE INTERVAL ACROSS THE
2600 REM *** BOTTOM OF THE GRAPH ***
2610 FOR I=K TO 10*K STEP K
2620 GOTO I/K OF 2630,2680,2680,2680,2630,2680,2680,2680,2680
2630 L=SB*I*1E-06
2640 L=(ITP(L*1000+.5)/1000)\REM * ROUNDED AND IN MHZ *
2650 MOVE LOG(1)/LOG(10),-10*M+T
2660 RSMOVE -20,-40
2670 PRINT L
2680 NEXT I
2690 RETURN

```

Fig. 5. Routine for producing log-log plots.

Generally, the REMarks in the listing explain the program's operation. It should be pointed out, however, that line 2110 prints the -3-dB frequency at the top of the graph and requires this value to be computed beforehand and converted to

TEK BASIC graphics...

```
1000 REM
1010 REM *** SPS BASIC ROUTINE ***
1020 REM *** FOR ENG. NOTATION ***
1030 REM *** 10/3/78 G.J. DEWITTE ***
1040 REM
1060 N=LOG(R)/LOG(10)
1070 IF N>0 THEN GOTO 1160
1080 REM
1090 REM *** ROUNDS UP TO NEXT HIGHER MULTIPLE OF 3 ***
1100 REM
1110 N=ITP(N/3-.999999)*3
1120 GOTO 1170
1130 REM
1140 REM *** ROUNDS DOWN TO NEXT LOWER MULTIPLE OF 3 ***
1150 REM
1160 N=ITP(N/3)*3
1170 N=-N
1180 REM
1190 REM *** GETS MANTISSA FOR ENG. NOTATION ***
1200 REM
1210 R1=R*10^N
1220 REM
1230 REM *** ROUNDS MANTISSA TO ONE DECIMAL PLACE ***
1240 REM
1250 R1=ITP(R1*10+.5)/10
1270 RETURN
```

Fig. 6. Routine to convert to engineering notation.

engineering notation. The -3dB frequency can be computed by the following TEK SPS BASIC statement

$R = \text{CRS}(Z, (\text{MAX}(Z) - 3)) * \text{SB}$

and the result can be converted to engineering notation and placed in R1 by the subroutine listed in Fig. 6.

With these preliminaries out of the way, the log-log plotting routine can be run. Normally its run time is adequate for most purposes. However, the program does use three nonresident commands—MOVE, DRAW, and SMOVE—and run time can be decreased by specifying when TEK SPS BASIC is loaded that there will be more than three nonresident commands resident at one time.



*By Gordon J. DeWitte,
Section Head,
High Speed Instrumentation,
EG&G, Inc., Los Alamos, NM*

By acceptance of this article, the publisher and/or recipient acknowledges the US Government's right to retain a nonexclusive, royalty-free license in and to any copyright covering this paper.

Programming hints

TEK SPS BASIC makes strings match—in any case

If you've written many programs using TEK SPS BASIC's powerful string capabilities, you probably have run into the problem of comparing equivalent upper- and lower-case strings. For example, one common technique is to ask a user to input a string selected from a menu of possibilities. The program usually accepts the string and compares it to a list of possible responses. If the user enters the correct string, but not in exactly the same case, the compare will fail. This problem can not only be irritating, but may cause errors because of incorrect input.

This handy little subroutine converts any lower case alphabetic characters in I\$ to upper case. Non-alphabetic characters are unchanged. The program simply loops through lines 30-70 as many times as there are characters in I\$. On each pass, line 40 extracts the next character from I\$ and stores it in a temporary string, T\$. Then, line 50 checks to see if the character in T\$ is a lower-case alpha. If it is, 32 is subtracted from the ASCII

value of the character, which converts it to the equivalent upper-case character. The resulting character is added to the end of the output string, O\$. Each loop increments I by one, causing line 40 to select the next character in the input string.

When all the characters are tested and mapped, line 80 prints the result. Lines 10, 80, and 90 may be removed when using this program as a subroutine in another program.

```
10 INPUT I$
20 O$=""
30 FOR I=1 TO LEN(I$)
40 T$=SEG(I$,I,I)
50 IF T$>="a" THEN IF T$<="z" THEN T$=CHR(ASC(T$)-32)
60 O$=O$&T$
70 NEXT I
80 PRINT O$
90 GOTO 10
```



*By Mark Tilden,
HANDSHAKE Staff*

Finding out more about Tektronix Signal Processing Systems

Tektronix, Inc.
3320 Holcomb Bridge Road
Peachtree Industrial Blvd.
Norcross, GA 30092
(404)449-4770

Tektronix, Inc.
3003 Bunker Hill Lane
Santa Clara, CA 95050
(408)496-0800

Tektronix, Inc.
P.O. Box 344529
Dallas, TX 75234
(214)233-7791

Tektronix, Inc.
2 Research Court
Rockville, MD 20850
(301)948-7151

Tektronix, Inc.
1258 Ortiz Drive, S.E.
Albuquerque, NM 87108
(505)265-5541

Tektronix, Inc.
24155 Drake Road
Farmington, MI 48024
(313)478-5200

Tektronix, Inc.
4660 Churchill Road
St. Paul, MN 55112
(612)484-8571

Tektronix, Inc.
482 Bedford St.
Lexington, MA 02173
(617)861-6800

Tektronix maintains Field Offices and Service Centers throughout the world. In the United States, the Field Offices listed below have Signal Processing Specialists ready to answer your questions and help you select the system that best suits your measurement needs. Outside of the United States, the Tektronix subsidiary or distributor in your country will be pleased to offer you the same services.

Tektronix
COMMITTED TO EXCELLENCE



HANDSHAKE

Newsletter of Signal Processing and Instrument Control

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

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