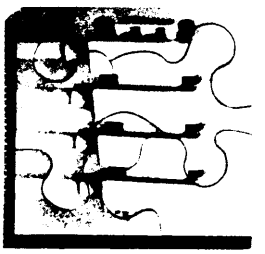
Solving the GPIB puzzle:



Talking to the 7854 Oscilloscope with TEK SPS BASIC

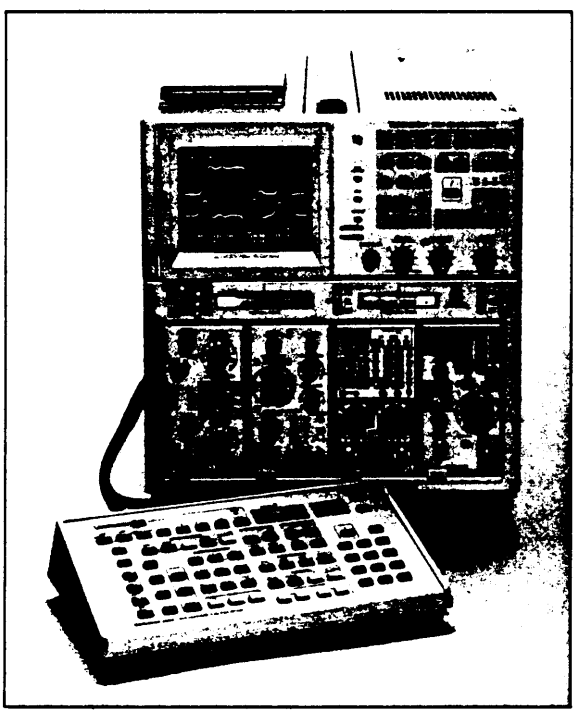


Fig. 1. *The 7854 Oscilloscope with its attached Waveform Calculator keypad.*

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.*

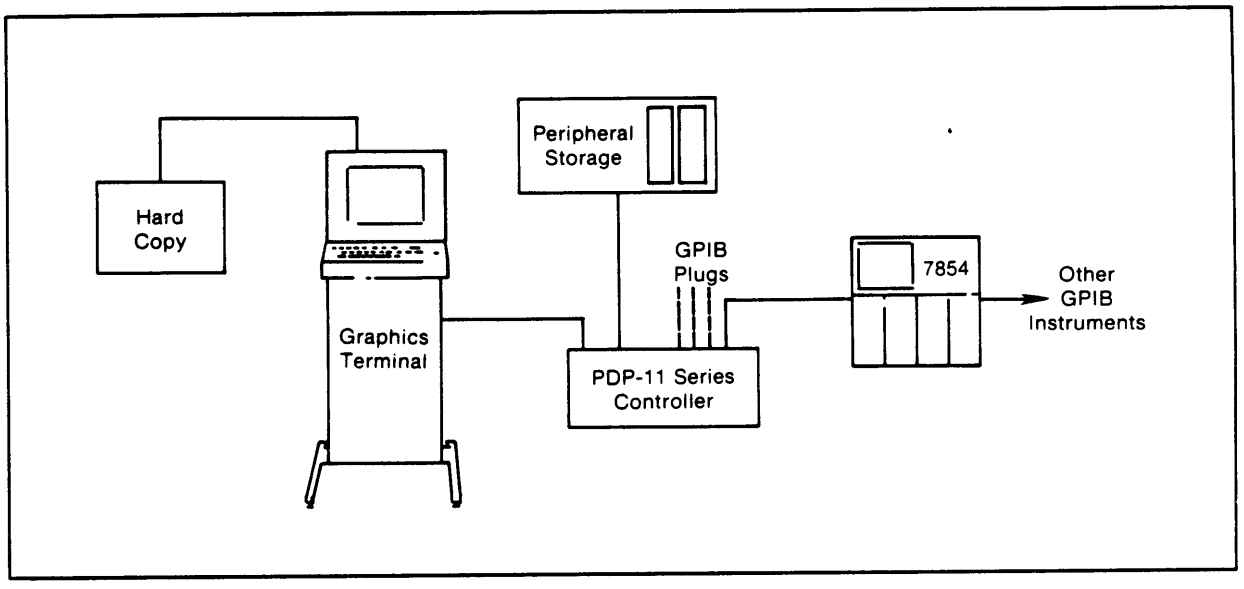


Fig. 2. *The 7854 and PDP-11 based GPIB system. Backplane space permitting, SPS TEK BASIC can support up to four Tektronix GPIB interface cards. Each interface card and bus can support 15 devices (14 instruments plus controller) with one device load per two meters of cable.*

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

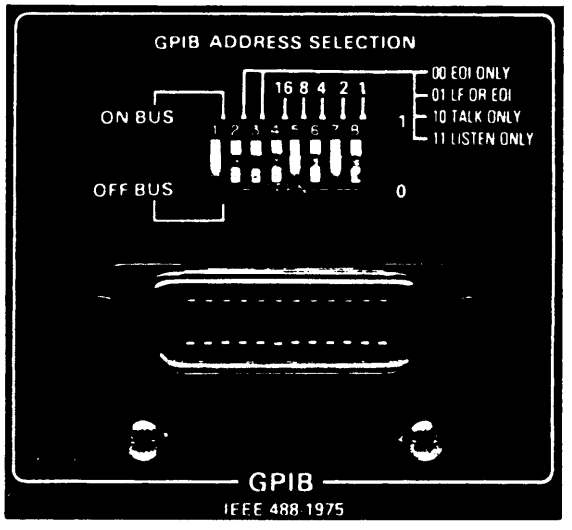


Fig. 3. *7854 Oscilloscope's GPIB connector and selection switches for setting primary address and communication mode.*

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 @Ø,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

the PRINT statement as follows.

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

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 @Ø,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.

```
1Ø REM INITIALIZATION ROUTINE
15 LOAD "GPI.SPS"
2Ø P1=1
25 L1=P1+32
3Ø T1=P1+64
35 POLL @Ø,S,ST,SS;T1
4Ø IF S=65 THEN 55
45 PAGE\PRINT S,ST,SS
5Ø PRINT "POWER-UP STATUS 65 NOT DETECTED"\GOTO 6Ø
55 PAGE\PRINT "POWER UP OKAY"
6Ø 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 intialized, 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 @Ø,L1
```

To put the 7854 into the STORED display mode, type

```
PUT "STORED" INTO @Ø,L1
```

To put it into the BOTH display mode, type

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

Following this with

```
RASCII X FROM @Ø,T1
```

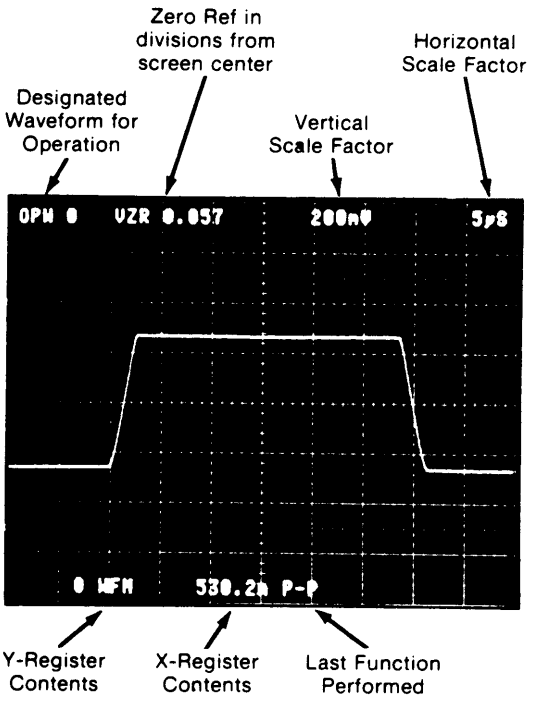


Fig. 4. *Example of 7854 display showing a stored waveform with its peak-to-peak value in the X register.*

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 @Ø,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 @Ø,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 alotted 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,-2.6954, -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

[1]Fixed 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

```
100 REM STORE 7854 @ WFM ON FLOPPY
105 SIFTO @@,1000
110 S=@
115 DIM ZW(511)
120 WHEN @@ HAS "SRQ" AT 51 GOSUB 1000
125 PUT "@ WFM SENDX" INTO @@,L1
130 IF S<>210 THEN 130
135 STERMC @@,";"
140 RASCII ZW$,ZW FROM @@,T1
145 CANCEL DX1:"WAF.1"
150 OPEN #1 AS DX1:"WAF.1" FOR WRITE
155 WRITE #1,ZW$,ZW
160 CLOSE #1
165 END
200 REM READ WAF.1 BACK TO 7854 @ WFM
205 OPEN #1 AS DX1:"WAF.1" FOR READ
210 READ #1,ZW$,ZW
215 CLOSE #1
220 SIFTO @@,1000
225 ZW$=ZW$&";CURVE "
230 S=@
235 WHEN @@ HAS "SRQ" AT 51 GOSUB 1000
240 PUT "@ WFM READX" INTO @@,L1
245 IF S<>211 THEN 245
250 WASCII ZW$,ZW;INTO @@,L1
255 END
1000 POLL @@,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 @@,1000
310 S=@
315 WHEN @@ HAS "SRQ" AT 51 GOSUB 1000
320 PUT "@ WFM SENDX" INTO @@,L1
325 IF S<>210 THEN 325
330 STERMC @@,","
335 RASCII Z1$,Z2,Z3$,Z4$,Z5,Z6$,Z7,Z8 FROM @@,T1
340 STERMC @@,";"
345 DELETE B
350 WAVEFORM WB IS B(Z2-1),HB,HB$,VB$
355 RASCII Z9$ FROM @@,T1
360 STERMC @@,""
365 RASCII B FROM @@,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 @@,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 SIFTO @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 SIFTO @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.