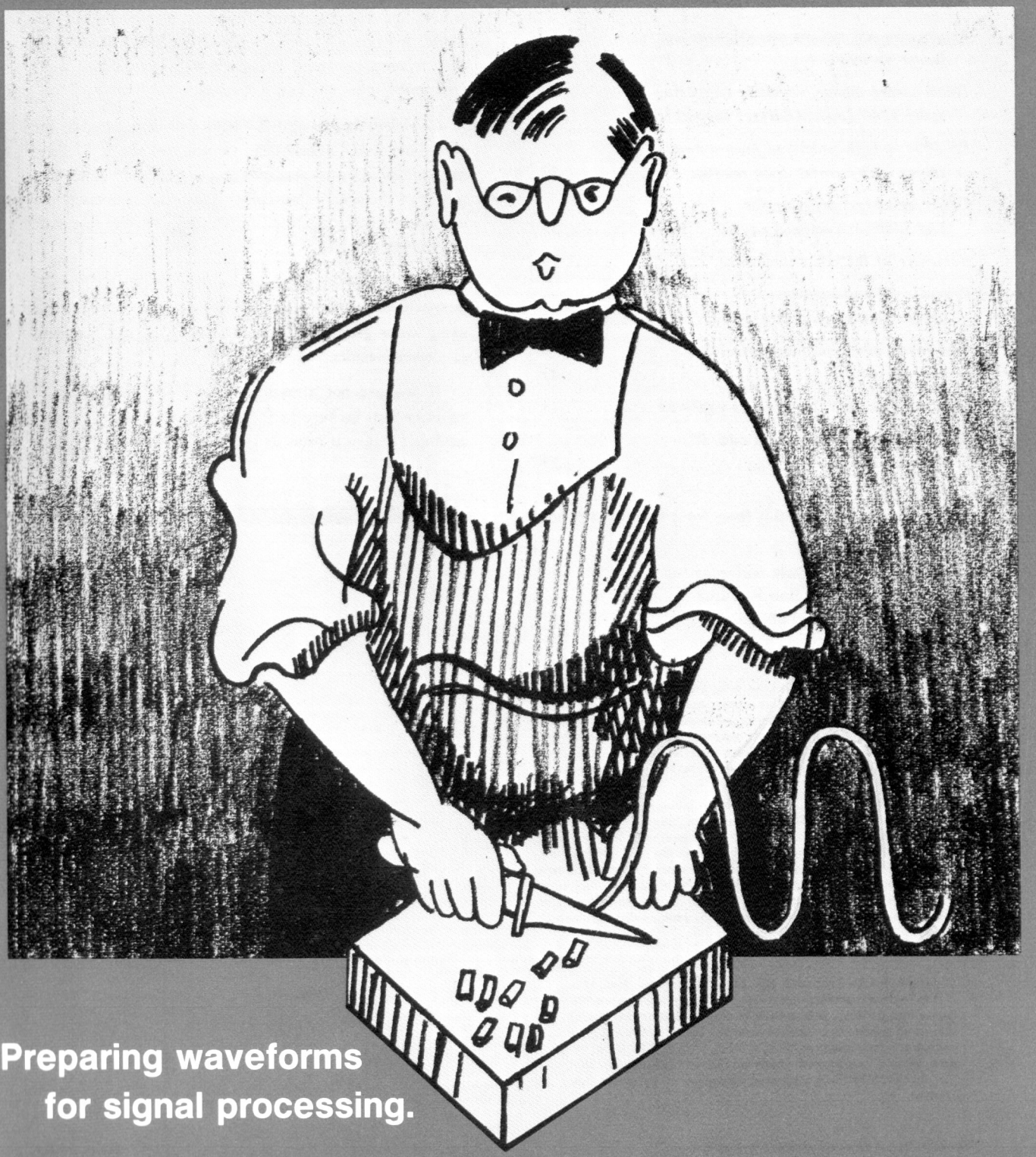
TEK APPLICATION INFORMATION

VOL. 9 NO. 1
SPRING/SUMMER 1984

HANDSHAKE

NEWSLETTER OF SIGNAL PROCESSING AND INSTRUMENT CONTROL



Preparing waveforms for signal processing.

Tektronix®
COMMITTED TO EXCELLENCE

Table of contents

Return the reply card in this issue for a free subscription to HANDSHAKE.

Managing Editor: A. Dale Aufrecht
Senior Writer/Editor: Bob Ramirez
Graphics: Bernard Chalumeau
Production: Anita Hicks

Late breaking news

As **HANDSHAKE** was going to press, we received notice of a new and exciting software package. This software is called EZ-TEST. It's designed to let you create extensive instrument control and measurement programs simply by making menu selections and following on-screen instructions. One of EZ-TEST's key features is that it supports IEEE Standard 488-1978 instruments from both Tektronix and other vendors.

Complete details on EZ-TEST will be covered in the next issue of **HANDSHAKE.** Or you can obtain preliminary information by contacting your Tektronix Sales Engineer or the sales representative for your country. Data sheets can also be obtained via the reply card in this issue of HANDSHAKE.

Also, watch the next issue of **HANDSHAKE** for announcements of new system configurations and details on using color graphics for enhancing your measurement processing results.

If you are not already on the **HANDSHAKE** subscription list, be sure to fill out a **HANDSHAKE** reply card and return it to us as soon as possible. 

Preparing waveforms for signal processing

A waveform digitizer's primary job, not surprisingly, is to digitize waveforms. Some additional functions, such as signal averaging or pretriggering, may be included too. But, by and large, these extras depend on first having the waveform digitized.

With waveforms in a digital format, many ancillary functions become possible, or at least easier to implement. Digitized waveforms are technologically easier to store and manipulate than analog data. As a result, dollar-for-dollar, more functions can be packed into a waveform digitizer than in a strictly analog instrument. And the digital format invites interfacing to any number of outside processing devices—from personal computers to large mainframes.

It is for this latter purpose—external processing—that the digital storage format of waveforms becomes of greatest concern. Instruments with built-in processing have built-in compatibility for their particular waveform format. But there are no guarantees of format compatibility when the waveform is transferred out of the instrument to that nebulous world of "external processing."

"External processing" is the point where detailed familiarity with waveform storage format is needed.

Basic waveform description

The traditional oscilloscope display is the essence of basic waveform description. As an example, look at the display in Fig. 1. There's the waveform, the familiar sinusoid. Then there's the crosshatch of graticule lines to help you count off amplitude levels or time intervals. And of course, at the top, there are scale factors for assigning amplitude and time values to each major graticule division.

That's the basic description. There's the waveform

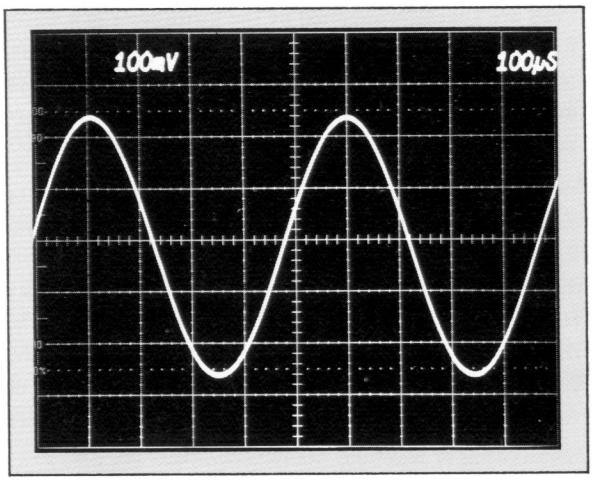


Fig. 1. *The standard oscilloscope display with readout contains all the basic information for describing a waveform.*

and a means of scaling or assigning values to it at any point along with units (e.g., volts or seconds).

For waveform processing, the digital description is not much different. However, instead of a waveform display, processing is done on an array of numbers representing the waveform. An example of a small waveform array is shown in Fig. 2.

The eight-point waveform array in Fig. 2 is certainly nowhere the size typically used in waveform processing. In practice, 256-point or larger arrays are used. However, the array in Fig. 2 does illustrate the essentials of digital waveform description. The waveform amplitude values, for a single cycle of a sine wave in this case, are stored in a time-ordered sequence starting from index 0. Index 0 is assumed to be time zero.

Index	0	1	2	3	4	5	6	7
value	0.00	0.707	1.00	0.707	0.00	−0.707	−1.00	−0.707

DSI = 1E−6
VU$ = "V"
HU$ = "S"

Fig. 2. *Standard format used in array descriptions of waveforms being set up for signal processing.*

While vertical scaling is intrinsic in the amplitude values stored, there is no corresponding or inherent horizontal scaling. Nor is there any inherent information about vertical or horizontal units. When waveforms are transferred from instrument to computer, scaling and units are generally supplied by the instrument in a waveform header. They aren't stored as part of the array, however, and must be parsed from the header and stored in individual variables. In Fig. 2, these variables are listed under the array as DSI, VU$, and HU$.

DSI stands for Digital Sampling Interval. This is the time interval between amplitude samples of the waveform. It is often referred to as "delta t" in signal processing literature, and its reciprocal is sampling frequency (the rate at which waveform samples were taken). In the case of Fig. 2, DSI is 1E–6 (one microsecond). For the eight samples in the array, this means that the array or record duration is 7*DSI, or 7E–6 (note that there are eight elements in the array, but only seven intervals).

The remaining items of description are vertical and horizontal units, which are stored in VU$ and HU$ in Fig. 2. With this final information, some very specific things can be said about the waveform stored in the array. For example in Fig. 2, the waveform value at index 3 is 0.707 volts and occurred 3 microseconds after digitizing started (element 0 or time zero).

With only minor exceptions, Fig. 2 is the format expected by most signal processing software packages. One exception you can expect is that some packages start array indexing from one rather than zero. Also, most packages do not automatically tie DSI and units to the waveform array. Taking care of these items as part of your application programming is a fairly straightforward process, however.

Getting into waveform format

For various reasons, few if any waveform digitizers output waveform data in the format of Fig. 3. Some output their data in raw form, leaving it to you to decide how best to format the data for your particular processing needs. Others, because of internal processing features, may modify the data slightly.

In any case, transfer of digitized waveforms from the digitizer to an external processor calls for routines to manipulate the date into the format of Fig. 2. These routines are specialized to the digitizer type and are usually discussed in the user's manual supplied with the digitizer. In the case of Tektronix Controllers and Digitizers, Controller Programming Guides and Instrument Interfacing Guides detail the waveform formats and transfer processes for specific instrument-controller combinations. These guides also provide example routines for typical transfer and formatting needs.

Invariably, the process reduces to three main steps—

- Setting up bus transfer of the data
- Reading waveform data into an array and parsing units and sampling interval information from the data header
- Converting the data to amplitude values

The last item, conversion to amplitude values, is necessary when the data is sent as digitizer levels or in screen units. Or, if the data is sent in binary, a conversion from binary to floating-point numbers becomes necessary.

Don't forget zero referencing

As part of any waveform acquisition, you'll also need to think about zero referencing. If you plan only relative measurements, such as peak-to-peak value, zero referencing isn't necessary. But most measurement and analysis requires absolute values. This means a zero reference has to be established for the waveform array before any processing is done. Failing to provide a zero reference generally results in an arbitrary level offset in the waveform data.

A zero-reference level is set up by switching the digitizer input coupling to ground. Some digitizers do an automatic zero-reference acquisition whenever ground coupling is selected. Others require selection of a referencing operation as well, such as pressing a GND button or issuing an ACQ GND command.

Fig. 3. *Most waveform digitizers have some means of zero referencing acquisitions. This can be by either issuing a command to the instrument or pressing a ground acquisition button (GND) with the input channel switched to ground.*

In any case, zero-reference acquisition generally consists of taking multiple samples of the ground signal and computing their average for a zero reference. This zero-reference value is then either stored as part of the waveform header or automatically subtracted from the waveform data to zero reference it. The exact zero-referencing procedure varies among digitizer models, so it's best to consult your digitizer's manual for specific details.

Basic processing considerations

With the waveform acquired, zero-referenced, vertically scaled, and placed in the array format of Fig. 2, actual processing can begin.

Processing is done element-by-element. For example, multiplying the array by a constant can be done in BASIC by a FOR loop as follows—

```
FOR I=0 TO 7
A(I)=4*A(I)
NEXT I
```

In this case, A is the waveform data array and 4 is the constant of multiplication. However, in software designed for signal processing, doing array operations by FOR loops is generally not necessary. Implied array operations are provided, and the same operation as above is performed (with the FOR loop implied) by the following simple statement—

```
A=4*A
```

In the same manner, two arrays can be multiplied by a statement such as—

```
C=B*A
```

where C is the array receiving the results of multiplying waveform data array B by waveform data array A. Again, the operation is element-by-element. The first element of B is multiplied by the first element of A. Then the second element of each is multiplied, and so on.

Continuing in the same element-by-element fashion, a variety of waveform operations can be provided. Mean and RMS values can be computed. Minimum, maximum, or any other level can be searched for. Waveform data can be integrated, differentiated, even FFTed (fast Fourier transformed). However, there are some special points to keep in mind throughout any array processing operation. Briefly, these are—

1. Operations are strictly numbers on numbers and assume nothing about the quality or orientation of the waveform.
2. Compatible array lengths are mandatory for multiple array operations.
3. Assumption of time coincidence is made in array combining operations.
4. Units processing generally is a separate operation.

These factors and how you view them or deal with them have a significant impact on waveform processing success. With that in mind, let's take a closer look at these factors and some other special considerations.

Waveform quality

Inevitably there is some noise associated with all waveforms. Typically it's a random component added to waveform amplitude and gets digitized right along with the waveform. In a plot of digitized waveform amplitude, the noise component makes the waveform look as if it has grown a crop of hair. Small amounts of noise appear as a short stubble, while larger amounts can make the waveform data plot appear quite bushy.

There are cases, too, where the noise is not random. Perhaps it's hum from a power supply or flourescent lighting. This type of additive noise is more apparent in low-frequency signals and makes the plotted waveform data appear to weave around its intended course.

Whatever the case, noise added to the signal is digitized along with the signal. That means the numbers put into the waveform data array are numbers for the signal amplitude plus noise amplitude at each sample point. If you search the data array for its maximum, using a MAX function for example, the number found will indeed be the maximum value in the data array. Whether that number actually represents the maximum of the waveform is another question! It could be the maximum of the noise riding on the signal.

The point is that any noise-polluted waveform that is digitized will be converted directly to noise-polluted numbers in the waveform data array. And those are precisely the numbers that the signal processing routines have to work on.

To avoid noise-polluted processing results, it's necessary to clean up the waveform. This can be done before digitizing by appropriate shielding or filtering. During acquisition and digitizing, repetitive waveforms can be cleaned up by signal averaging. Or, after digitizing, digital filtering or smoothing techniques can be used.

Many modern waveform digitizers have built-in signal averaging capabilities that can be applied for noise reduction. Invariably, fully integrated waveform processing systems provide signal averaging, either through instrument firmware or through system software routines. The other operations of digital filtering or smoothing are highly data and application dependent, and thus must

generally be implemented on an individual case-by-case basis. **Digital Filters**, by R.W. Hamming (1977, Prentice-Hall), is an excellent reference for filter construction and application.

Waveform orientation

In many cases, the orientation of the waveform data in the data window affects processing results. For example, consider computing the mean value of a waveform. Let's use a pure sine wave since it is symmetric in positive and negative amplitudes and will have a known mean of zero.

Figure 4 shows a sine wave acquired in a typical manner. Notice in particular that there is not an integer number of cycles displayed in the acquisition window. When the mean value of the data array for this sine wave is computed, the answer will not be the expected value of zero. This is because the sine wave is acquired with more data on positive going portions than negative going portions. The data has not been acquired to truly reflect the symmetry of the actual waveform.

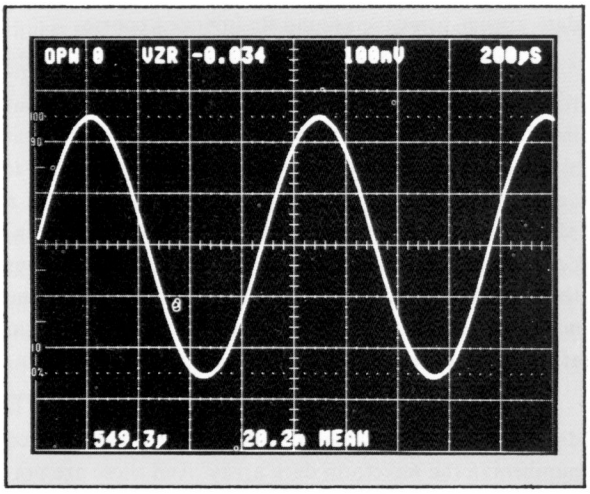


Fig. 4. *Example of operating on a noninteger number of cycles: The computed mean (bottom of display) is correct for the data displayed but it is not zero, as would be expected for a pure sinusoid.*

There are a number of solutions to the problem of maintaining symmetry for computations such as mean and RMS. The most obvious being to restrict either the data or the computations to an integer number of cycles.

Acquiring and digitizing precisely an integer number of cycles is usually not practical. However, a quick and simple way of reducing error associated with partial cycles is to acquire many cycles of the waveform. An additional fractional cycle out of 20 or 30 cycles of acquired waveform data contributes much less error to a mean or RMS computation than a fractional cycle out of one or two cycles.

Where maximum accuracy is required, the computations need to be restricted to an integer number of cycles. This requires going into the data array and searching it for the beginning and ending point of a cycle. Then mean or RMS computations can be applied to just that data segment. In software that provides data-zone specification for its operations, just specify the data zone of the array to be operated on. For packages that don't provide zone operations, you'll need to define another array and transfer the desired zone of integer cycle data to it for mean or RMS computation. This latter approach is detailed further in Fig. 5.

The right number of points

Another aspect of waveform orientation in the data window concerns getting the right number of points on the waveform. This has to do both with obtaining desired time resolution and maintaining array compatibility for certain types of processing.

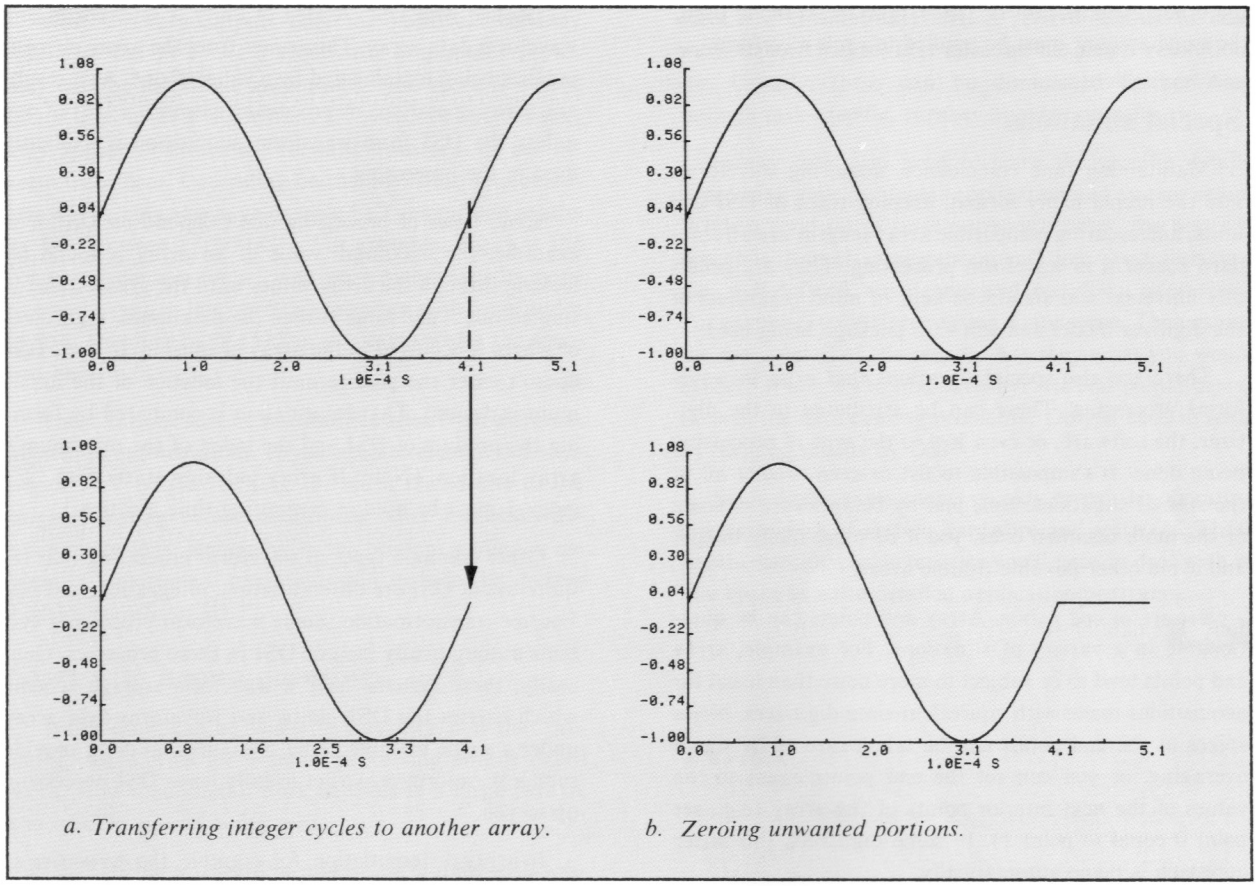
Depending on the type of digitizer used, you'll be selecting DSI (digital sampling interval) either directly or indirectly through a time-base setting. For indirect selection, DSI is figured by dividing record duration (the length of horizontal sweep, generally the product of the time base setting and the number of horizontal display divisions) by the record length (number of points being acquired into an array).

DSI is the time resolution of the data. If you want to compute pulse rise time to one nanosecond, for example, then a DSI of at least one nanosecond is a wise choice. Of course, values between samples can be interpolated, but this still depends on having enough points initially for adequate definition of waveform features.

In general, there should be no fewer than three points on a pulse's transition edge. Substantially more would be preferable.

There can be too much of a good thing, though. With fixed record length digitizers, concentrating too many points on a particular waveform feature can result in a record duration too short for capturing the rest of the waveform. A smaller DSI means a shorter record duration for a fixed record length. One advantage of variable record length digitizers is that you can compensate for a smaller DSI by selecting a longer record length.

Varying either DSI or record length needs to be done with regard for your overall processing plan. For example, most FFT routines require power-of-two record lengths (64, 128, 256, etc.). If you plan to use Fourier



a. Transferring integer cycles to another array.

b. Zeroing unwanted portions.

Fig. 5. *Two approaches for computations requiring limitation to an integer number of cycles.*

analysis, you'll need to select record lengths that are compatible with FFT needs. Also, DSI selection impacts how the waveform fits in the data window, which is another important consideration in FFT processing.

Multiple waveform processing

The right number of points is also a concern when two waveforms are combined mathematically. As a rule, each waveform data array must have the same number of points. This allows element-by-element computation of sums, products, etc., with standard statements such as LET A=B*C, where A is the results array and B and C are data arrays, all having the same length. The element-by-element operation is implied here by the variables being arrays.

If one array is longer than the other, element-by-element operation cannot be completed. The software comes to the end of the shorter array and still has unprocessed elements left in the longer array. This is an error situation for most signal processing software packages. But, if you find yourself in a situation with differing array lengths, possibly as a result of acquisitions from two different digitizer models, the operation can be done in a FOR loop. Array elements are explicitly called one at a time and operated on in the loop. Also, the results array must be the same length as the shortest array in the operation, and the number of loop iterations must be the same as the results array length.

Combining arrays element-by-element is not enough, however. The arrays are just numbers. Software happily operates on them in any order you specify, regardless of what the numbers really represent. So it's critical to remember that the array elements are supposed to represent waveform amplitudes at specific times. This means that multiplying or adding two waveform arrays produces valid results only when the array elements have been created while maintaining proper time coincidence. The DSI for each array must be the same. Also, the first element of each array must represent the same point in time, otherwise unwanted time shift appears in the results.

Generally, time coincidence is taken care of by acquiring the waveforms at the same time, with the same sampling rate, and while triggering one acquisition off of

the other. Also, if pre- or post-triggering is being used, the same amount must be applied to each waveform.

Special situations

Maintaining time coincidence, acquiring waveforms into the proper array format, keeping track of DSI and units, and assuring compatible array lengths are all standard concerns in waveform processing. They are generally universal and should be kept in mind regardless of the digitizer model and software package being used.

There are also special situations that arise in waveforms processing. These can be attributed to the digitizer, the software, or even just to the type of processing being done. It's impossible to list or even predict all of the special situations. But, just by being aware of some of the more common ones, you'll be more likely to spot and avoid other possible trouble areas.

Beware of end points. Array end points can be questionable in a variety of situations. For example, array end points tend to be subject to more noise than usual for acquisitions made with equivalent-time digitizers. Noise spikes at the end points can be taken care of by signal averaging, or you can set the end points equal to the values of the next interior points of the array (e.g., set point 0 equal to point 1). In some digitizers, this latter approach is taken automatically.

Array end points can also become questionable as a result of some types of processing. Typically, this occurs when the processing at a given point uses points to either side. Everything goes fine on the array's interior points. But, at the end points, there are no points to one side. The algorithm has to either extrapolate points or just use duplicates of the end points.

Three-point differentiation and ensemble averaging are two examples of processing that can create questionable end points. Some techniques of curve fitting and digital filtering can also create end-point problems. Whenever any of these types of processing are used, you should investigate their effect on end points and the implications for further processing. If the end-point data is not critical in the rest of your analysis, then you can ignore what happens at the end points. However, if the end points are critical in further calculations, such as frequency-domain analysis, something must be done about them.

Perhaps the easiest thing to do about questionable end points is to simply throw them away. For example, transfer the interior points to a shorter array, leaving out the questionable end points. Then interpolate the data back into an array of the desired length.

Dealing with DSI. A DSI of unity is inherent in any waveform data array. This comes from the array element indicies being incremented by a value of one. As a result, any array operations or processing assumes a DSI of one unless the DSI from waveform acquisition is explicitly used in the processing.

Some types of processing don't depend on DSI. For example, the maximum value of an array is found by looking through the array values until the greatest one is singled out. That value is then the maximum, regardless of where it occurred in the array or in time. In fact, DSI doesn't enter the picture until the location of the maximum is desired. That information is computed by forming the product of DSI and the index of the maximum's array location. (Note: If array indexing starts with "1", index-1 must be used in computing time location.)

Other common types of waveform processing that require use of DSI are differentiation, integration, and fast Fourier transformation. Some waveform processing systems automatically include DSI in these processes. Generally, these systems have a waveform storage scheme which carries the DSI, units, and waveform data array under a single variable name. Systems that don't provide such a waveform construct usually leave DSI processing up to you.

In array differentiation, for example, the derivative at a point is the difference in point values divided by the distance between points. In an array the distance between points is 1, which is the increment between indices. However, for waveform data, sample separation is given by DSI, and the numerical differentiation needs to be divided by DSI to correspond to dV/dt. Integration, on the other hand requires multiplication by DSI. And the fast Fourier transform (FFT) requires use of DSI to compute frequency sample interval in the results, which is—

$$\Delta f = 1/N*DSI$$

where N is the record length of the time waveform that was FFTed.

Before applying DSI for scaling as described above, you should experiement with your system to determine whether it automatically includes DSI in the processing or not.

The 1/N factor. In using the FFT to study frequency amplitudes of signal, additional scaling of FFT results may be necessary. The mathematics of the FFT produce frequency results that are split across both the positive and negative frequency domain (see Fig. 6). Half the amplitude of any given component is located in positive frequency and the other half in negative frequency.

Most software outputs only the positive frequency results of the FFT. As a result, the amplitudes shown may only be half-value amplitudes. Also there is a 1/N factor used in FFT algorithms (N is record length) that may appear in either the forward transform or the inverse transform. Depending on the algorithm implementation, you may also have to scale FFT magnitude results by 1/N or even 1/(N*DSI). For example, Fig. 6 is the frequency domain for a 1-volt peak sine wave. Its positive and negative line spectra are shown with amplitudes of 256. Dividing by N (512 in this case) scales the spectral lines to 0.5.

These various scaling possibilities arise from different algorithm optimizations. If you need to change the scaling of FFT output, be sure to return the results to the original scaling before performing other frequency-domain operations. Returning to the original scaling is especially important if operations will include the inverse Fourier transform (IFT).

Start with known quantities

If one thing can be said with certainty, it is that preparing waveforms for a signal processing entails detail work. How much detail work depends in a large part on the specific system being used.

Configuring a waveform processing system with software from one vendor, a computer from another, and a waveform digitizer from still another almost certainly ensures maximum detail work. A complete system from a single source typically eliminates the more onerous details of compatibility and format consistency. However, there are still the application related details to deal with—record length, DSI selection for proper resolution, etc. Usually, these can be diminished further only through task-specific, turnkey application software.

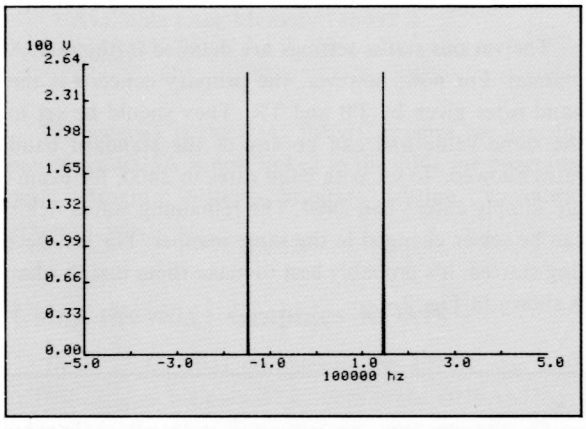


Fig. 6. *Positive and negative frequency domain output of a 512-point FFT for 75 acquired cycles of a sine wave of 1-volt peak amplitude. Note that, in this case, vertical scaling needs to be multiplied by 1/512 for absolute amplitude measurements.*

In any case, you need to learn the specific details concerning your particular application and system setup. While a good set of system manuals helps, the best approach is through some experimentation. Start with simple, well-defined waveforms and waveform operations. For example, try differentiating a sine wave. Then examine every detail of the results. Are they what you would expect? If not, try to find out why. Are end-point aberrations in the data causing problems? Do you need to scale by dividing results by DSI?

Such simple experiments quickly reveal the operating characteristics of system hardware and software. These are the specific characteristics you'll need to deal with as you begin to write detailed application programs.

By Bob Ramirez, HANDSHAKE Staff.

Setting up the 4025A with the 4041 System Controller

The Tektronix 4041 System controller can be used either in an execute-only mode or a program-development mode. The latter requires the optional Program Development ROM Packs and Keyboard. This is an easy, plug-it-in option that covers most standard programming needs.

There are, however, times when you'll need graphics capability as well. This is often the case when waveform processing operations are being developed. For such applications, there are 4041 Graphics and Plotting ROM Packs available too. But you need a graphics device, such as a 4025A Computer Display Terminal with the graphics option, to take advantage of them.

"Autold" adds terminal

Figure 1 lists a 4041 BASIC program that sets up communication with a 4025A Computer Display Terminal. This is an autoload program that needs to be loaded from DC-100 tape and executed each time you bring up the 4041/4025A combination.

Entering the program from the listing in Fig. 1 is done with the 4041 Program Development Keyboard. Then the program is saved to DC-100 tape with **Save "autold(ope=new,cli=yes)"**. The program can then be loaded and executed automatically any time by simply inserting the tape in the 4041 and pressing the 4041's AUTO LOAD button.

Some housekeeping first

However, before running the configuration program, some things need to be set in the 4025A. First, press SHIFT-STATUS. The STATUS key is the upper rightmost key on the 4025A keyboard. SHIFT-STATUS prints someting like the following on the 4025A screen—

U !2012

In the above 4025A message, "U" indicates that the 4025A is in the unbuffered mode. The "!" is the command character, and "2012" indicates memory blocks. The point of interest here is the command character. For 4025A operation with the 4041 Graphics and Plotting ROM packs, the command character needs to be ` , the grave accent or ASCII 96. To make this change, use the 4025A keyboard to enter **!COM`** followed by a carriage return. (Note that the COM must be preceded by the existing command character, ! in this example.)

```
100     Set console "frtp:"
110     Input prompt "baud rate:":b
120     S$="comm(bau="&str$(b)&",tim=5,edi=402):"
125     Set sysdev "tape(lon=yes):"
130     Set driver s$
140     Set console s$
150     Print "dd-mmm-yy hh:mm:ss"
160     On error(380) then gosub no_comm
170     Input da$
180     Da$=trim$(da$)
190     If len(da$)=0 then goto 150
200     On error(554) then call err
210     Set time da$
220     Print ask$("id")
230     Print "Available User Memory:";ask("memory",all)
240     Print "Ready"
250     Set console "comm:"
260     Delete line 1 to 65000
270 No_comm:     branch 100
280     End
300 Sub err
310     Branch 150
320     End
*_
```

Fig. 1. *The **autold** program used to configure the 4041 Controller for operations with a 4025A Computer Display Terminal.*

Now, enter ` **sys** from the 4025A keyboard. The 4025A will display a message similar to that shown in Fig. 2, which summarizes the current status of the 4025A.

The various status settings are detailed in the 4025A manual. For now, however, the primary concern is the baud rates given by TB and TR. They should be set to the same value and can be any of the standard baud rates allowed. To set both baud rates to 2400, for example, simply enter ` **bau 2400**. The remaining status items can be set or changed in the same manner. For just getting started, it's probably best to make them match what is shown in Fig. 2.

```
`sys
  TB= 2400  RB= 2400  DL=    0  LM= 1  RM=80  WL= 0  V#=3.0
  TS= 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  CC=`  FS=↓  PR=↑  EL=↓  RS=↓ ↓ ↓ ↓
  DU=F  BU=N  EC=R  FF=N  SN=N  KB=M  CM=M  PA=N
-
```

Fig. 2. *4025A system status message.*

Coming up with "autold"

Having dealt with the preliminary details of creating **autold** and selecting 4025A parameters, configuration of the 4025A with the 4041 can be completed. Connect the 4025A RS-232C cable to the RS-232C port (COMMO) on the 4041.

Then with both the 4041 and 4025A powered up, insert the autoload cassette into the 4041 tape drive. When the asterisk prompt appears on the 4041 display, press the 4041 **AUTO LOAD** button. The program will load and a **BAUD RATE=** prompt will appear on the 4041 display. Use the 4041 front-panel numeric keypad to key in the baud rate you selected for the 4025A. Then press **PROCEED** at the top right of the keypad.

As soon as you press **PROCEED,** a date and time entry prompt appear on the 4025A screen. This prompt appears as dd-mmm-yy hh:mm:ss. Using the 4025A keyboard, enter the day, month, year and military time in the order indicated by the prompt. Here's an example entry:

01-feb-84 09:50:01

There is a time-out on this entry. So, if you pause too long, the program reverts back to the stage of prompting you for a baud rate input from the 4041 keypad. If you find the time-out interval (5 seconds) too short for your liking, you can change the value of **tim=** in line 120 of the **autold** program to a higher value.

Once the date and time are successfully entered, the following message appears on the 4025A screen—

```
ID TEK/4041,V79.1,2.0
Available User Memory:146492.0
Ready
*
```

The asterisk is the 4041 BASIC prompt for user input. Your 4025A is now linked to the 4041 for programming, program listing, program editing, waveform graphics, whatever.

Using the 4041 Graphics ROMs

To set up for 4041 graphics on the 4025A, you need to first assign a logical unit number to the 4025A. This is done with the OPEN statement. For example, **Open #35:"comm0:"** assigns the RS-232C port as logical unit 35. Assuming the 4025A is connected to "comm0:", the 4025A can now be referred to as #35, or 35.

Reference by logical unit is necessary in the GINIT command, which is used to set up graphics communication between the 4041 and 4025A. Issuing **Ginit 35,4025,2** tells the 4041 that logical unit 35 is a 4025A terminal with option 2 installed. The 4041 knows about this terminal and opens up a graphics space over most of the 4025A's screen, leaving the bottom four lines for program lines or immediate mode commands. At this point, the 4401 Graphics ROM calls can be used to create graphic displays on the 4025A. To set the graphics area back to zero—i.e., return the screen to full size for program listings—enter **wor 0**.

You now have the basics of setting up a 4025A Computer Display Terminal for operation with a 4041 System controller. This same process, with some variations, can also be used to set up other types of terminals for operation with the 4041.

By Bob Ramirez, HANDSHAKE Staff, with appreciateion to Jerry Kryszek, also of the HANDSHAKE Staff, for providing the ***autold*** *program listing.*

Efficient data logging with the 4052A

Data logging is basically the process of writing captured data out to a storage device. The data can be a single value from a voltmeter, for example, and taken every hour. Or it can be multiple values following in quick succession, such as digitized amplitude samples of a waveform.

The reasons for writing such data to a storage device are various. Sometimes it's simply desirable to keep an archive record of measurements taken. In other cases, the measurement process requires collecting data at differing times or from different sources. For these applications, logging data to a storage device is a convenient and safe method of holding data until all of it is in for analysis.

The Tektronix 4052A Controller provides a variety of commands for meeting various data logging needs. By and large, choice of commands is a matter of convenience and desired data format. However, in time critical situations, such as capturing and logging a rapid succession of waveforms, special care must be taken to use the most efficient approach.

Some basic considerations

Efficient data logging is based primarily on two general requirements. These are:

1. The number of stored bytes should not be higher than the number of acquired bytes.
2. The data transfer operations between the data producing device and the storage device must be done as quickly as possible.

Basically, number 1 above says that fast logging programs should not add to or modify the data. Especially, they should not add bytes for control, header information, or anything else. The purpose at this point is to transfer data, not process it.

Number 2, above, is related in part to number 1. Limiting stored bytes to just those acquired, keeps the number of bytes dealt with at a minimum. The fewer the bytes, the faster the transfer and storage. However, the larger issue in Number 2 is that the transfer statements used be the most efficient ones and that they be combined in logging programs in the most efficient manner.

Meeting the above requirements demands in-depth knowledge of both the software being used and the acquisition instrument. As an example of the type of specifics needed, let's look at the 7612D Programmable Waveform Digitizer and how data from it can be logged by the 4052A.

The 7612D/4052A combination

The Tektronix 7612D is capable of outputting data at a very high rate. This makes it an interesting example for studying fast data logging.

7612D data is sent as binary data in 8-bit bytes, one byte per data point. The number of data points, or bytes, to be logged depends on the record-length setting of the 7612D. This can go as high as 2048 points.

The logging process, for optimum speed, should maintain the 7612D data format throughout. The process should act simply as a pipeline from the 7612D to the selected storage device. The storage device can be any of the 4050 peripherals—extended memory, internal tape, or an external GPIB device. The data path from the 7612D consists of two legs, a transfer from the 7612D to 4052A memory and then a transfer from 4052A memory to the storage device.

4052A transfer facilities

In the 4052A system, there are three pairs of data transferring facilities. These are:

INPUT and PRINT
RBYTE and WBYTE
READ and WRITE

Let's consider each of these and see which might best be used in a fast data logging situation.

INPUT-PRINT. This transfer pair is better suited for ASCII data. INPUT and PRINT can't be used for binary transfers because they ignore bit 8 of every byte, which means a data loss for binary data.

RBYTE-WBYTE. RBYTE is widely used to transfer data from the 7612D to the 4052A. However, this routine contains an implicit transformation to a floating-point data format. This means that every data byte from the 7612D is converted by RBYTE to 8 bytes within the 4052A system.

To optimize transfer speed from the 4052A to the storage device and to minimize storage requirements, the 8-byte floating-point format can be converted to a 1-byte ASCII character by using the CHR function. This must be done in a loop and is rather time consuming. But then

the data is in a format that can be efficiently transferred to a storage device with the WRITE command.

READ-WRITE. WRITE with a string source is the ideal routine for transferring bytes to a storage device. WRITE does not alter the number of bytes and it is fast. Of course, the absolute time of WRITE depends on the type of storage peripheral. For example, it takes a few milliseconds to write 2048 bytes to Extended Memory or as much as a second to write the same thing to internal tape. Regardless of the storage media, however, WRITE is always faster than any other command for transfer from the 4052A to storage.

Given the speed of WRITE, it's natural to ask: "Why not use READ for moving the acquired data from the 7612D to the 4052A?" This would bypass the building of floating-point numbers by RBYTE and therefore increase data logging speed.

There is one small problem with READ, however. It expects two leading format bytes that are normally produced by WRITE. Unfortunately, the 7612D does not offer such a format; it uses a binary block format. However, there is a way around the format problem. To see this, let's examine the formatting bytes at the bit level (see Fig. 1).

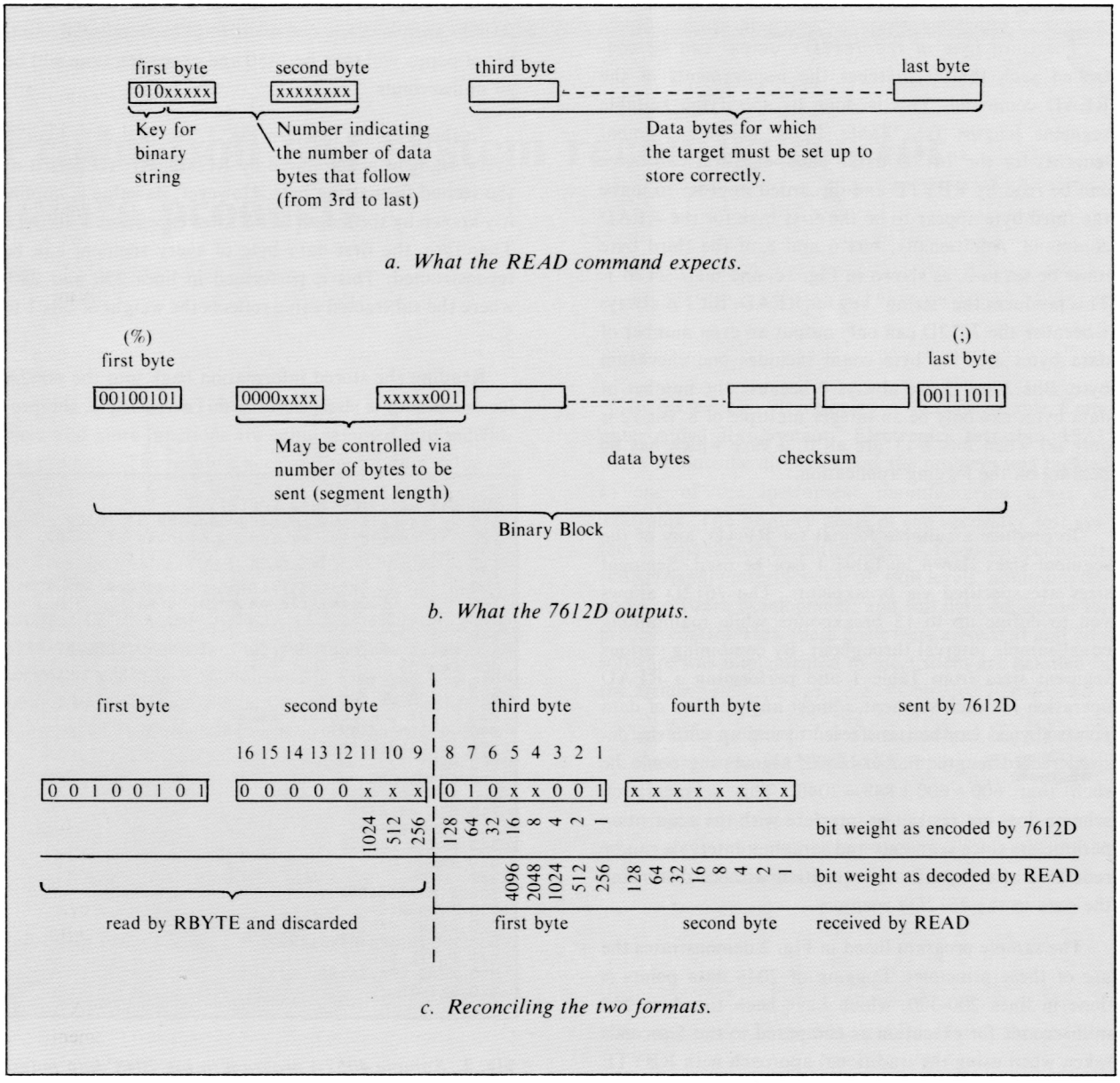


Fig. 1. *READing data from the 7612D into the 4052A.*

TABLE 1
ALLOWABLE 7612D WAVEFORM SEGMENTS

								: min. size of A$
64	----	---	---	----	----	----	----	: 511
72	328	584	840	1096	1352	1608	1864	: 2559
80	336	592	848	1104	1360	1616	1872	: 4607
88	344	600	856	1112	1368	1624	1880	: 6655

Making READ work with 7612D data

The third byte of the 7612D's output can be controlled such that it matches the requirements of the READ command. This is done by specifying suitable segment lengths (see Table 1 for possible segment lengths) for the 7612D data. Then, the first two bytes can be read by RBYTE and discarded in order to make the third byte appear to be the first byte for the READ command. Additionally, bits 6 and 8 of the third byte must be set to 0, as shown in Fig. 1c, and bit 7 set to 1. This produces the "string" key for READ. Bit 1 is always 1 because the 7612D can only output an even number of data bytes and the byte count includes one checksum byte. Bits 2 and 3 are always 0 because the number of data bytes can only be an integer multiple of 8. Bit 12 is only set when bits 2-11 are all 0, a case which has no bearing on the logging application.

To produce a suitable format for READ, any of the segment sizes shown in Table 1 can be used. Segment sizes are specified via breakpoints. The 7612D allows you to define up to 13 breakpoints while maintaining equal sample interval throughout. By combining various segment sizes from Table 1 and performing a READ operation for each segment, almost any number of data points (bytes) can be transferred to end up with the desired record length. For example, segmenting could be such that $600+600+848=2048$. This segmentation scheme does not restrict or interfere with the acquisition parameters since segments and sampling intervals can be redefined after waveform acquisition without disturbing the data in the 7612D's memory.

The sample program listed in Fig. 2 demonstrates the use of these principles. Logging of 2048 data points is done in lines 200-300, which have been timed at 215 milliseconds for execution as compared to the 5 seconds taken when using the traditional approach with RBYTE and CHR. Data logging time may be further decreased by modifying the program in Fig. 2 to contain only one READ operation instead of three. The maximum number of points will then be 1880 and execution time will be 90 milliseconds.

In the program listed in Fig. 2, the first data byte of every segment is not stored by READ. READ uses it as the second formatting byte. However, its value is implicitly known by the length of A$ after execution of READ. Therefore, the first data byte of every segment can be reconstructed. This is performed in lines 230 and 280, where the subtracted value reflects the weight of bits 1 to 5.

Reading the stored information back into the 4052A for processing is straightforward. For example, the pro-

```
100 DIM B$(1),C$(2048),D$(2)
110   !Pa=relative primary address of 7612D
120   !Sa=relative secondary address of 7612D
130   !
140   !     }
150   !     } Set acquisition parameters and arm.
160   !     } Open file on peripheral
170   !     }
180   !
190 PRINT @Pa,Sa:"CBPT;SBPT 600,5E-9,1200,5E-9";
200 C$=""
210 FOR I=0 TO 1
220    GOSUB 500
230    B$=CHR(LEN(A$)-6400)
240    DIM A$(599)
250    C$=C$&B$&A$
260 NEXT I
270 GOSUB 500
280 B$=CHR(LEN(A$)-4352)
290 DIM A$(847)
300 C$=C$&B$&A$
310 WRITE ...:C$
320 END
330   !
500 DIM A$(6655)
510 D$=STR(I)
520 PRINT @Pa,Sa:"REA A,0";D$;
530 WBYTE @Pa+64,Sa+96;
540 RBYTE A,A
550 READ @Pa,Sa:A$
570 RETURN
```

Fig. 2. *Sample 4052A program to log 2048 data points into 2048 (+2) stored bytes.*

gram could be set up as follows—

```
DIM A$(2048)
open file
READ. . . . . :A$
A=LEN(A$)
DIM Y(A)
FOR I=1 TO A
Y(I)=ASC(A$,I)
NEXT I
END
```

The programs and benchmarks discussed here are directly applicable to the 4052A. With minor modifications, they can also be used with a 4052. The concept of using READ for data transfers between an acquisition device with binary block format and the 4052A can be implemented whenever the device can output its memory content in segments. Thus, the Sony-Tektronix 390AD Programmable Waveform Digitizer is another candidate for this method.

By Peter Kellenberger,
Tektronix, Inc.,
Zug, Switzerland.

Production test system reconfigures for future products

With large scale integration and microprocessors, more and more functions are being jammed into individual products. The result is accelerating product value for the dollar. But, for manufacturers of the products, it also means soaring quality test complexity.

That the products must be tested with automated systems under software control goes without saying. On complex GPIB based products, such as a digital oscilloscope, manually testing all combinations of product functions would be a monumentally expensive task. But what do you do about the burden of test system software updating as new products with new functions come on line for testing? Test system reconfiguration can be an expensive process too.

In his article, "Developing a Reconfigurable Computer-Aided Test System," **Electronics Test,** Jan. 1983, Steve Jumonville discusses how the problem was tackled in one of the instrument manufacturing areas at Tektronix. The primary goals of the necessary test system are maximum reconfigurability, freed-up technician skills, friendly interfaces for all skill levels, minimum ongoing software development, and full data collection for process control. How these goals were arrived at and how software was implemented to meet them are detailed in the article.

To obtain a reprint of this article, check the appropriate square on the **HANDSHAKE** reply card.

Touch-screen menus, versatile triggering make 1240 Logic Analyzer logical

- Touch-screen menus
- Dual time base
- Supports 80186, 68000, F9450 and 15 other chips
- 72 channels
- 100 MHz asynchronous, 50 MHz synchronous
- Clock/qualifier lines
- Post-acquisition processing
- Glitch storage
- RS-232, GPIB, parallel printer port

The new Tektronix 1240 Logic Analyzer is a powerful system development and debugging tool. Application ROM packs, communication packs, and acquisition cards let you pick the features you need now and add others later. Yet, with all its power and flexibility, the 1240 is probably the easiest logic analyzer of all to use.

Dual time-base power

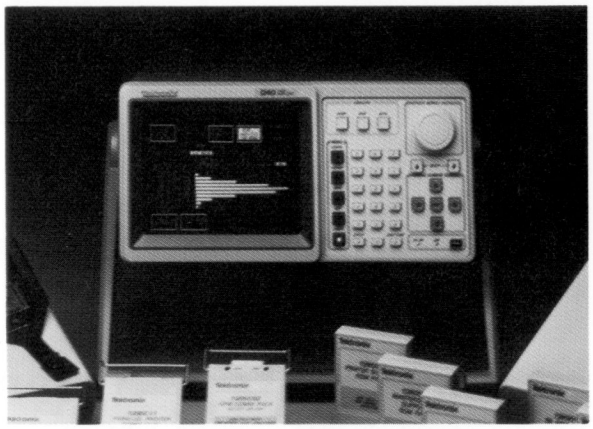
The 1240 has two separate time bases, making it unique among logic analyzers. The two time bases operate independently; however, they are time correlated for properly sequenced data acquisition. Triggers can be built from either time base. And triggering flexibility is provided by one global event recognizer and 14 sequential event recognizers with delay counters and timers, conditional branching, and duration filters.

After acquiring data, the 1240 can process it in various ways. The 1240 knows about ASCII, binary, octal, hex, and EBCDIC. In addition to the usual state tables and timing diagrams, ROM packs provide software for such processing as mnemonic disassembly and histogram displays.

With the GPIB or RS-232 COMM pack, you can set up communications with the 1240. The 1240 can be remotely controlled, and instrument setups and acquired data can be stored on a peripheral device.

Touch-screen ease

Powerful as it is, the 1240 is easy to use. It's almost entirely menu operated. It also provides four levels of operation so you don't have to deal with advanced features until you are ready for them. As a result, you can easily put the 1240 to work the day you receive it.



Operation is further simplified by a touch screen. To select an operation, simply touch the listed item on the screen. No more specialized and cryptic keyboards to learn!

For more information about the 1240 Logic Analyzer, contact your local Tektronix Sales Engineer or check the appropriate box on the reply card in this issue of **HANDSHAKE.**

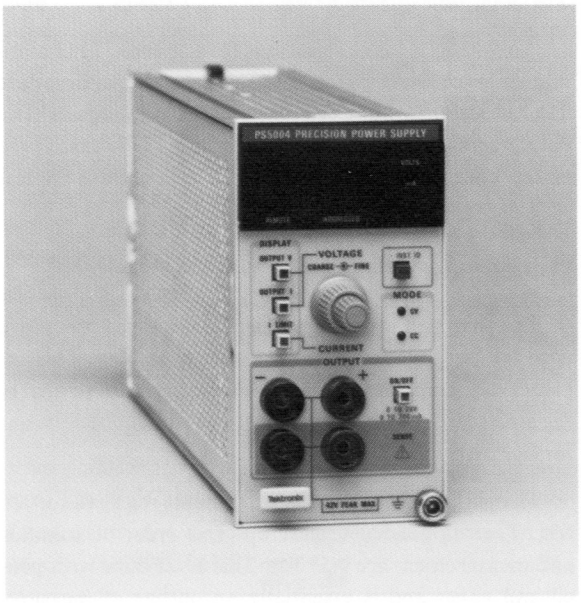
PS 5004, a high-precision programmable source with current draw readout

A new programmable power supply is available from Tektronix. It's the PS 5004 Precision Power Supply, and it offers a number of features that make it particularly useful in semiconductor and hybrid circuit applications.

A key feature is the high resolution and accuracy over its voltage range. Voltages from 0 to 20 VDC can be selected, either manually or under program control, to a resolution of 0.5 mV and an accuracy of ±0.01% +2mV. Current supplying capacity is 305 mA maximum, and a current limit can be set with 2.5 mA resolution. Output voltage, current limit, or current drawn from the supply can be observed on the PS 5004 display or output over the GPIB.

The high-impedance sense terminals are another key feature of the PS 5004 supply. Buffering reduces current flow and makes them significantly less sensitive to sense lead resistance. As a result the PS 5004 meets its output voltage specifications even when faced with as much as five ohms of sense lead resistance.

For full details on the PS 5004 Precision Power Supply, contact your local Tektronix Sales Engineer.



TPG creates test programs for TM 5000 instrument systems

TPG stands for Test Program Generator. This new software package runs on the Tektronix 4041 Controller and creates procedures for automated testing with TM 5000 series instruments.

All you do is make selections from menus. TPG automatically converts these selections to the appropriate TPG procedures for setup and control, data acquisition, data logging, and pass/fail limit testing. TPG takes care of the code generation so you can concentrate on test sequences.

How it works

The first step in setting up a measurement procedure is to lay out a step-by-step measurement plan. This includes any prompts that you want the program to give to the operator as either setup instructions or as requests for operator input of data.

The plan needs to list the instruments required for the test. Then the instrument setups and order of stimulus and measurement are laid out. This is all done with pencil and paper and is essentially an outline of measurement steps and their order. It's like a specification sheet that you would hand to a programmer. Except you hand it to the TM 5000 TPG instead, and the TPG programs it for you.

TPG knows your instruments

The TM 5000 TPG program exists on a DC-100 tape cartridge with several other supporting HELP files. Since the TPG is an AUTOLD program, all you do is insert the cartridge into the 4041 Controller's tape drive and press the AUTO LOAD button. In a few minutes, TPG will be loaded and ready to go.

TPG knows about the following TM 5000 instruments:

DC 5009 Programmable Universal Counter/Timer
DC 5010 Programmable Universal Counter/Timer
DM 5010 Programmable Digital Multimeter
FG 5010 Programmable 20 MHz Function Generator
PS 5010 Programmable Triple Power Supply
MI 5010 Multifunction Interface System and cards
SI 5010 Programmable Scanner
AA 5001 Programmable Distortion Analyzer
SG 5010 Programmable Oscillator
CG 5001 Programmable Calibration Generator
PS 5004 Programmable Precision Power Supply

You can have any or all of these instruments active on the GPIB during TPG operation. In fact, with the second GPIB interface option on the 4041 Controller, you can have up to 28 TM 5000 instruments configured and controlled under TPG operation.

The systemizing process is handled by the TM 5000 TPG program. It takes care of interrupt handling routines and drivers for the instruments. In the GPIB Initialization Process, the TPG prompts you to connect the desired instruments to the bus and power them up. Then you press the RETURN key on your terminal, and the TPG interrogates each instrument for its ID and builds a configuration table. The active devices on the bus are listed on the terminal so you can check the configuration. Press the RETURN key again and the TPG is ready to help code your test procedure.

Menu driven test program generation

The TM 5000 TPG is designed to let you build complete test procedures simply by making selections from menus and responding to prompts. Based on your menu selections and responses, TPG procedures are generated for the specified instruments and tasks.

The Main Menu, for example, offers:

OPERATOR PROMPT—allows you to generate multiline operator prompts of up to 300 characters for each prompt. You can create simple setup and connection instructions or interactive prompts asking for entry of data or pass/fail information.

SELECT INSTRUMENT—redefines the keyboard for instrument selection and enters the Instrument Selection Menu.

MEASUREMENT LOOP—generates procedure for iteractive stimulus and measurement operations by prompting you through stimulus/measurement instrument selection and setup.

EDIT PROCEDURE—redefines keys for editing and enters an Edit Menu.

FROM TAPE—retrieves a previously developed procedure from tape.

STORE>TAPE—stores current procedure on a user specified tape file.

GENERATE NEW PROCEDURE—clears current procedure from memory and reconfigures the GPIB.

TERMINATE PROGRAM—exits TPG and returns you to 4041 immediate mode.

HELP—retrieves Main Menu help information from a tape file and displays it on screen.

EXECUTE PROCEDURE—allows execution of a TPG developed test procedure using TM 5000 instruments.

The other menus branch down from the main menu as needed. For example, SELECT INSTRUMENT in the Main Menu leads to the Instrument Selection Menu, which, after you select an instrument, leads you to the Instrument Operating Mode Menu.

The Instrument Operating Mode Menu prompts you for instrument setup information. Basically, you set up the instrument front panel just as though you were doing manual testing. Then TPG learns the setup for that measurement step. You also have the option of entering instrument control code from the keyboard if you like. This allows you to take advantage of additional instrument features not available from the front panel.

Execution options abound

A Test Execution Menu offers numerous ways to execute TPG created procedures. A test procedure can be run in normal fashion, or it can be run in a step mode for checking program operating details. PLOT LOOPS is another option for systems using Tektronix graphics terminals. PLOT LOOPS allows you to plot data acquired in loop testing as acquired data versus stimulus or as acquired data versus loop iteration number. All of these options are useful in testing and debugging procedures before they are put on line.

The TM 5000 Test Program Generator is available in versions for systems using the Tektronix 4025A Graphics Terminal, the Tektronix 4105 Color Graphics Terminal, or an ANSI X3.64 standard computer display terminal. For price and ordering information contact one of the order points listed in the **Tektronix Instrumentation Software Library Catalog.** Copies of this catalog can be ordered, at no cost to you, by using the reply card in this issue of **HANDSHAKE.**

Special Notice

As **HANDSHAKE** was going to press, a new software package was announced. This package, called EZ-TEST, is based on the same concept as TPG but includes many significant extensions, including support of GPIB products from both Tektronix and other vendors. Watch the next issue of **HANDSHAKE** for a complete review of this new softward package.

Some useful BASIC expressions

Most BASICs have some fundamental math functions available, things like sine and cosine. But what do you do if you need to compute the arc hyperbolic secant of something?

Rather than riffling through dusty, old math books to find formulas for that odd operation that you may occasionally need, here's a list of expressions using standard BASIC functions. E, X, Y, E1, E2, P1, P2, M, and N used in these expressions may be constants, variables, arrays, or expressions.

Function	BASIC Expression
Log_x of E	LOG(E)/LOG(X)
LOG_x^{-1} of E	X↑E
Decibels	
(voltage or current)	20 * LOG(E/E2)/LOG(10)
(power)	10 * LOG(P1/P2)/LOG(10)
Rectangular to polar	
(X,Y to M∠N)	
(magnitude M)	SQR(X * X + Y * Y)
(angle N)	ATN(Y/X)
Polar to rectangular	
(M∠N to X,Y)	
(X coordinate)	M * COS(N)
(Y coordinate)	M * SIN(N)
Tangent of E	SINE(E)/COS(E)
Secant of E	1/COS(E)
Cosecant of E	1/SIN(E)
Cotangent of E	COS(E)/SIN(E)
Arc sine of E	ATN(E/SQR(1 − E * E))
Arc cosine of E	ATN(SQR(1 − E * E)/E)
Arc secant of E	ATN(SQR(E * E −1))
Arc cosecant of E	ATN(1/SQR(E * E − 1))
Arc cotangent of E	ATN(1/E)
Hyperbolic sine of E	(EXP(E) − EXP(−E))/2
Hyperbolic cosine of E	(EXP(E) + EXP(−E))/2
Hyperbolic tangent of E	(EXP(E) − EXP(−E))/ (EXP(E) + EXP(−E))
Hyperbolic secant of E	2/(EXP(E) + EXP(−E))
Hyperbolic cosecant of E	2/(EXP(E) − EXP(−E))
Hyperbolic cotangent of E	(EXP(E) + EXP(−E))/ (EXP(E) − EXP(−E))
Arc hyperbolic sine of E	LOG(E + SQR(E * E + 1))
Arc hyperbolic cosine of E	LOG(E + SQR(E * E − 1))
Arc hyperbolic tangent of E	(LOG(1 + E) − LOG (1 − E))/2
Arc hyperbolic secant of E	LOG(1/E +SQR(1/E * E − 1))
Arc hyperbolic cosecant of E	LOG(1/E +SQR(1/E * E + 1))
Arc hyperbolic cotangent of E	(LOG(E + 1) − LOG(E − 1))/2

Note: LOG in these expressions is the natural logarithm.

Friendly software creates programming efficiency

As IEEE-488 based instruments assume more programmable capabilities, test system software requirements become more complex. Maintaining a test and measurement system comes down to a choice of building a staff of super hackers or looking for test system software that's friendly enough for end users to program in.

That latter approach—friendlier software—would seem to be the most efficient and direct approach. Why make your research and engineering staff wait on another department for application software when they could just as well produce programs that would meet their needs more directly?

The question is: What is it that makes software friendly enough for end-user programming in the test and measurement environment?

Steve Peterson, Tektronix, Inc., explores that question in "Friendly Software for Test and Measurement," **Computer Design,** October 1983. The article looks at the increasing need for uniform codes and formats for instrument control—something beyond just the physical and electrical elements of IEEE 488. From here, the discussion extends to language type, learn mode capabilities, and required interrupt structures for processing both instrument events and error conditions.

The article provides a good background on IEEE-488 bus control of instruments and uses numerous programming examples to illustrate the major concepts. Here's just one example from the article, showing how a learn mode should operate to obtain settings from a DMM and store them for later automated instrument setup: "...the following sequence would be used to locate and read the settings of an instrument at primary address 16

```
100 PRINT #16:"SET?"
110 INPUT #16: SETDMM$
120 PRINT #"DMMSET":SETDMM$
```

In this case, the article illustrates "friendly" as three simple program lines to accomplish a very complex task.

For a reprint of this article, check the appropriate square on the **HANDSHAKE** reply card or call your local Tektronix Sales engineer and ask about 4041 BASIC for friendly test and measurement control.

We're interested in hearing from you

If you are a user of digital signal processing techniques, HANDSHAKE is your newsletter. We welcome your comments on information contained herein.

Our continuing goal is to make HANDSHAKE as meaningful as possible. The best way to do this is to know what your exact signal processing needs and applications are. Write and let us know what topics you would like to see discussed in future issues.

We will try to include articles on topics of general interest in future issues of HANDSHAKE. Better yet, why not write an article yourself! Tell us about your measurement problems and how you solved them. Articles accepted for publication in HANDSHAKE will carry full credit to author and company.

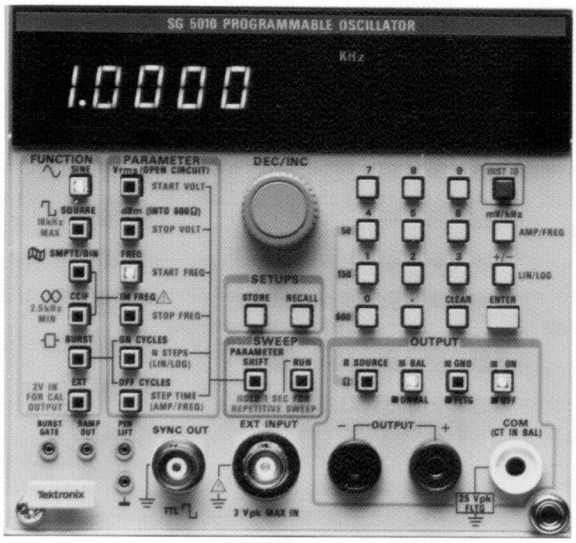
We would also like to invite you to send in programs or programming techniques and hints that you have found helpful.

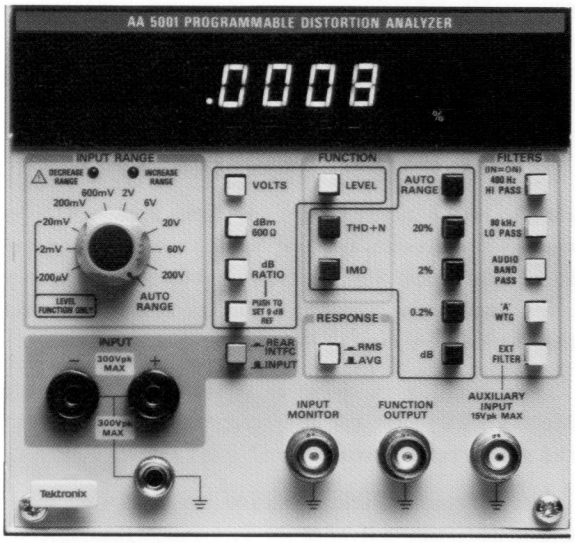
Address comments to:

HANDSHAKE Editor
Group 157 (54-016)
Tektronix, Inc.
P.O. Box 500
Beaverton, OR 97077

Use the attached postcard to order your HANDSHAKE subscription; to tell us if you're interested in more information on equipment, applications, and software mentioned in HANDSHAKE; or if you want to contribute to the newsletter. We'll be in touch with you promptly.

New instruments automate precision audio testing





Fully testing top-grade professional and consumer audio equipment used to be a painstakingly slow process, requiring the full attention of highly skilled technicians. Now, the whole process can be automated without sacrificing precision or accuracy. Test throughput and repeatability go up and technician labor decreases. Two new test instruments from Tektronix make it all possible. They are the SG 5010 Programmable Oscillator and the AA 5001 Programmable Distortion Analyzer.

The SG 5010 provides synthesized sine frequencies from 10.00 Hz to 163.80 kHz with 0.01% accuracy. Burst, square wave, and amplifier modes are included. Sweeping is also possible under program control or standalone. High-level output allows testing headroom and clipping thresholds of line level devices. Set up can be manual or programmed over the GPIB, and nonvolatile memory allows the instrument to store up to ten different instrument configurations.

The AA 5001 does all the standard audio tests—THD, IMD (SMPTE, DIN, CCIF difference tone), gain/loss, and signal-to-noise ratio. It can be operated either manually or under program control over the GPIB. The analyzer input is fully balanced, and fully programmable filter and detector selection is provided to meet a wide variety of measurement standards.

To find out more about the SG 5010 Programmable Oscillator and the AA 5001 Programmable Distortion Analyzer, use the convenient **HANDSHAKE** reply card in this issue. Or, for a faster response, contact your local Tektronix Field Office or the Tektronix Sales Representative for your country.

Audio test automation benefits explored

"If we use a five-to-one time savings to keep things conservative, the speed advantage of a system can cut 80 percent off of your present test labor costs. Using conservative labor and burden costs for skilled technicians, that computes to about $125,000 in annual savings for a moderate-volume manufacturer." This is just one of the examples of return on investment offered by Bob Metzler, Tektronix, Inc., in his article "Automated Audio Test Systems for Professional Audio Performance Requirements," **db**, March 1983.

The article also covers a number of intangible returns from using an automated test system. These geneally are in the area of product quality improvements. For example, studio operation and maintenance can be improved through faster, more thorough testing. Audio proofs can be run automatically every evening at shut down or every morning before start up.

For audio equipment manufacturers, system advantages, besides reduced labor cost, include more accurate and repeatable testing. The article also notes that automated systems, once in place, typically get used to perform more thorough and complex testing than ever would have been attempted with manual testing. The result is tighter quality and production control, which reduces to better products produced more efficiently.

And, for the audio equipment development laboratory, automated audio testing allows more avenues of exploration. Data collection, preparation, and presentation goes so much faster that engineers can test more design options. The result is better designs and higher design confidence.

If you'd like to find out more specifics on the benefits and cost justifications for automating your audio test capabilities, ask your local Tektronix Sales Engineer for a reprint of "Automated Audio Test Systems For Professional Audio Performance Requirements." Or you can use the **HANDSHAKE** reply card to obtain a reprint.

Speeding System Execution added to GPIB series

A little over a year ago, Tektronix made available reprints of a two-part article series. The articles originally appeared in **EDN** and were billed as "A Designer's Guide to GPIB instruments." The intent was to provide basic guidelines for selecting GPIB instruments and software and for getting a system up and running with minimum fuss.

The response to this article series has been gratifying. In fact, reprints had to be reordered to fill demand.

But now, the article series is even better. A third part, "Programming techniques speed IEEE-488 system execution," by Mark Tilden, Tektronix, Inc., has been added. For those planning a GPIB-based test system, this new addition shows you how to estimate system performance and discusses numerous performance factors to be aware of in selecting system components. And once you've configured a system, the article contains a wealth of information on setting up both programs and hardware for best system performance.

If you are planning a GPIB-based test system, or if you'd like to improve an existing system's performance, the information in this three-part article series will be a valuable aid. For reprints of "Designer's Guide to GPIB instruments, Parts I-III," contact your local Tektronix Sales Engineer, or request them with the reply card bound into this issue of **HANDSHAKE**.

HANDSHAKE
Group 157 (54-016)
Tektronix, Inc.
P.O. Box 500
Beaverton, Oregon 97077

Tektronix®
COMMITTED TO EXCELLENCE

45W-5753