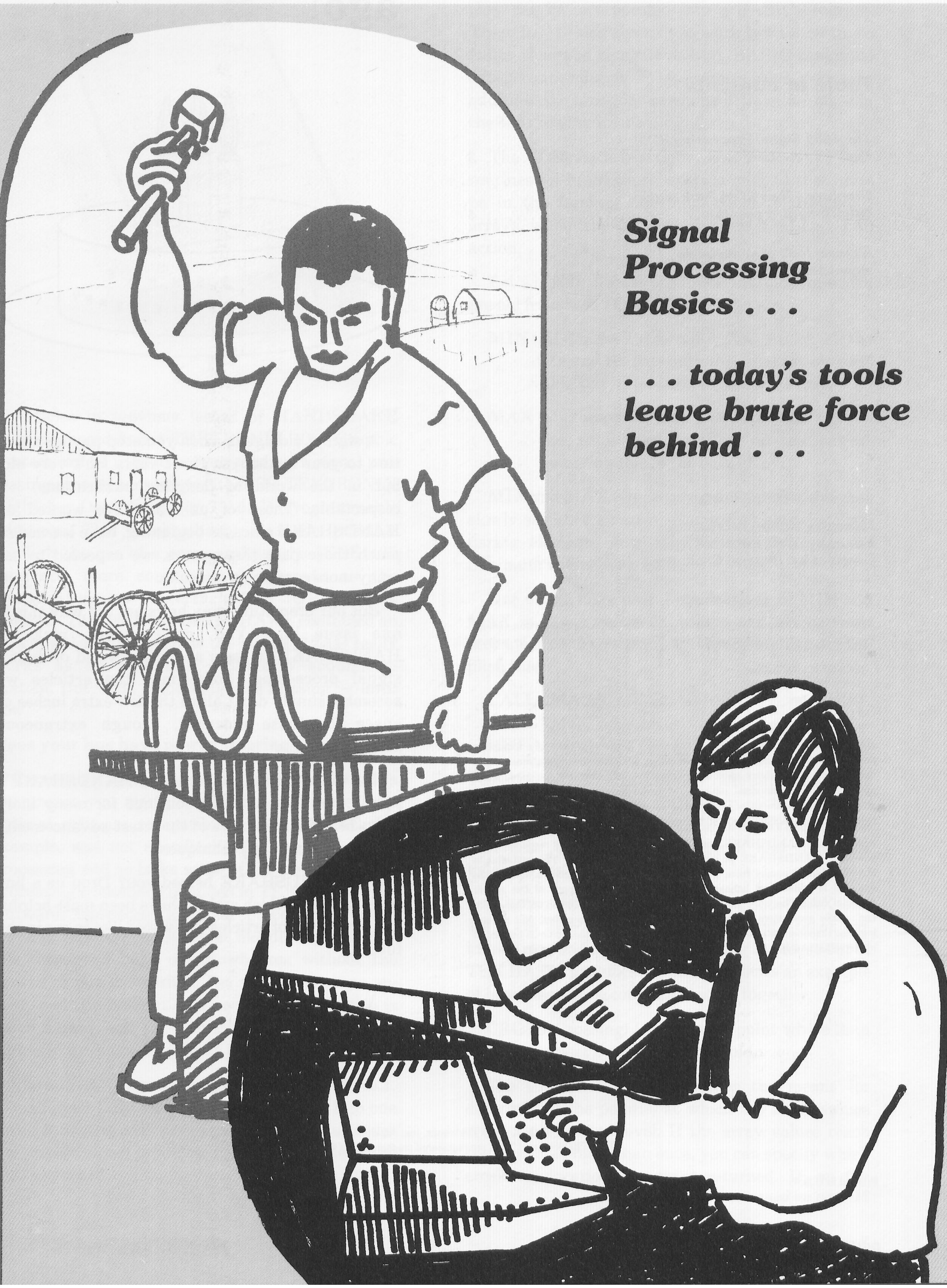
Vol. 4 No. 2 WINTER 78-79

Tektronix®
COMMITTED TO EXCELLENCE

HANDSHAKE

Newsletter of the Signal Processing Systems Users Group



Signal Processing Basics . . .

. . . today's tools leave brute force behind . . .

Tektronix® COMMITTED TO EXCELLENCE
HANDSHAKE
Newsletter of the Signal Processing Systems Users Group

Table of contents

The 4051 Signal Processing ROM Pack – something new *3*

Graphics – New ROM Pack makes quick graphing possible *4*

A closer look at some basic signal processing operations *6*

Managing Editor: Bob Ramirez
Edited by: Walt Robatzek
Graphics by: Bernard Chalumeau

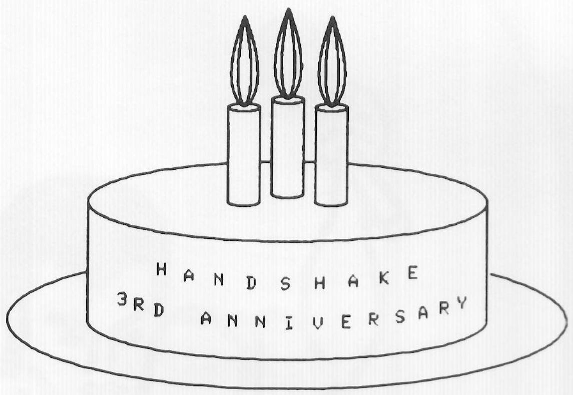
HANDSHAKE is published quarterly by the SPS Documentation Group. Permission to reprint material in this publication may be obtained by writing to:

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

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

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

Just three years ago!



A sign of old age is when you stop paying attention to your birthdays. Three years isn't very old, but in the world of fledgling publications, it's respectable. Those of us who have worked on HANDSHAKE since its beginning, have learned so much these past three years, we expected to see many more candles on our cake.

But our premature gray hairs are not why this is one issue late. You, our readers, pick up HANDSHAKE to learn about the world of digital signal processing. Sometimes the articles we assemble simply don't allow the few extra inches of space for these pleasant, though extraneous announcements.

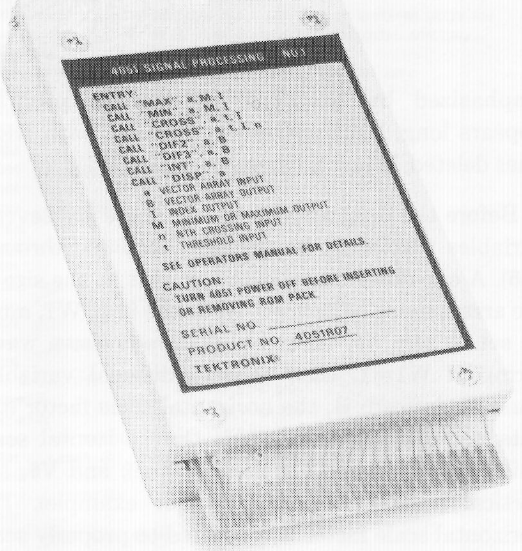
We welcome the beginning of HANDSHAKE's fourth year, and hope to continue for many more years bringing you news of the latest advancements in signal processing techniques.

Has HANDSHAKE helped you? Drop us a line to let us know which articles have been most helpful and what kind of articles you would like to see in the future.

Copyright © 1979 Tektronix, Inc. All rights reserved.
TEKTRONIX and TEK are registered trademarks of Tektronix, Inc.

Printed in U.S.A.

The 4051 Signal Processing ROM Pack — something new



Articles in previous issues of HANDSHAKE have discussed how a TEKTRONIX 4051 Graphic System can be used as a controller for a signal processing system containing either the TEKTRONIX Digitizing Oscilloscope or 7912AD Programmable Digitizer. This compatibility is made possible through the use of a common bus: the IEEE 488 interface, more commonly referred to as the General-Purpose Interface Bus (GPIB). The Summer, 1978 issue of HANDSHAKE contained an article which explored how the 4051/7912AD really works as an automatic waveform acquisition system. But, still, something was missing — until now.

Does your bus have power steering

Connecting instruments via the IEEE 488 Bus guarantees you many things, but not necessarily optimal software compatibility. 4051 BASIC, for example, was not specifically designed for signal processing and so lacks some of the powerful array processing commands included in TEK SPS BASIC software. Operating a signal processing system without these functions can be like driving a transit bus through a busy downtown area without the benefit of power steering. The scheduled stops are made and the destination is reached, but the trip is much slower and the driver is winded from the efforts of turning that wheel.

The 4051R07 Signal Processing ROM Pack No.1, a small Read-Only Memory device that fits into one of the 4051 backpack slots or into a ROM Expander slot, makes signal processing power available at the 4051 keyboard.

The seven new functions — MIN, MAX, CROSS, DIF2, DIF3, INT, DISP — can all be executed in an immediate mode directly from the 4051 keyboard, or they can be incorporated into a BASIC program. These ROM Pack algorithms work two to ten times faster than the equivalent 4051 BASIC programs (often containing slow, repetitious FOR loops). In addition, the functions consume little or no space in the 4051 read/write memory.

The ROM Pack functions, which are really subroutines set in firmware, operate on data that must be in the form of one-dimensional floating-point arrays when the subroutines are CALLed into action.

Let's briefly discuss the functions, most of which are old friends of TEK SPS BASIC users.

MIN (Minimum) Performs a fast search of the data and returns both the smallest numeric value and its location in the data array.

MAX (Maximum) Works like MIN but returns the value and location of the largest numeric value in the data array.

MIN and MAX replace program subroutines that slowly searched an array, element by element, comparing elements until, the desired maximum or minimum value was known.

One of the more useful applications of MIN and MAX is to normalize an array to values between zero and one. The following statements can accomplish this.

```
CALL "MAX", A,V1,I
CALL "MIN",A,V2,I
LET A =(A−V2)/(V1−V2)
```

The array, A, is normalized by first subtracting the minimum value of the array from each element in array A. Each resulting value is then divided by the difference between the maximum and minimum values of the original array.

When used with the DISP function, MAX and MIN are also useful for graphing data of an unknown numeric range. The "Getting the most out of TEK BASIC graphics" column presents an example of how these three commands work together.

CROSS (Crossing) locates each point where data meet a specified threshold.

The CROSS function provides the means for determining the point(s) at which the array values cross a designated level. If the array values reach the threshold more than once, you can specify which crossover location you want returned. If, on the

continued on page 16

Getting the most out of TEK BASIC graphics

New ROM Pack makes quick graphing possible

The TEKTRONIX 4051 Graphic System, outfitted with the 4051R07 Signal Processing ROM Pack makes a powerful controller for waveform processing. Three of the ROM Pack functions, MINimum, MAXimum, and DISPlay, which are actually firmware subroutines, are particularly useful in the graphics side of signal processing — when you want to see the acquired signal, in an informative way.

A fast, unlabeled graph of an array can be done in four 4051 BASIC program statements (Fig. 1, lines 45 through 60), even when the range of the data in the array is unknown. The sine wave used as an example is generated in lines 10 through 30. The VIEWPORT in line 35 overrides the default viewport (full-screen graphing), and places the graph in the upper right corner of the screen. Lines 45 and 50 use the MIN and MAX functions to determine the minimum and maximum data values. and place them in variables V1 and V2. The WINDOW statement in line 55 ensures that the entire array will be graphed; since, N is defined as the size of the array, and V1 and V2 are the array's minimum and maximum values. Once graphed, you can PRINT V1 and V2 to see the data range.

If, however, you want labeled graticules, and if you want to graph waveforms as well as arrays, you need the program in Fig. 2. This program provides a graph of an array, such as in Fig. 3, or a graph of a waveform, as in Fig. 4.

The listing in Fig. 2 is well documented so the flow of the program can be easily followed by reading the REMark lines (program segments are emphasized by boldfaced type). The program appears long, but the working portion, with REM lines deleted, is just 86 lines.

Before the program can be run there are several variables that must be defined (lines 100 through 145). A one-dimensional array, A, and N, the size of the array, must be defined. Also, the flag, W1, must be set to zero (for an array) or to one (for a waveform). If W1=1, then three additional variables must be defined: H, the horizontal scale factor (the data sampling interval); H$, the horizontal scale units (usually some division of time); and V$, the vertical scale units (volts in our example). The horizontal scale factor will be used to properly scale the horizontal axis.

The section of the program beginning with line 200 ensures a constant array or waveform (such as a DC signal) is properly graphed. Next, the graticule is sized and drawn beginning at line 250. Line 265 makes the horizontal dimension of the graticule equal to the size of the array and the vertical dimension equal to the difference of the array's minimum and maximum. Then, the AXIS statement in line 275 constructs the graticule from a FOR loop series of X-Y axes.

The expression, (M2−M1)/8*8+M1, in line 265 points out an interesting programming problem that deserves amplification. You might correctly expect the maximum Y value in the window statement to be written as, simply, M2; since the expression technically equates to M2. Notice, however,

```
LIST
10 DIM A(512),B(512)
15 LET A=1/512
20 CALL "INT",A,B
25 LET A=2*PI*B
30 LET A=SIN(A)
35 VIEWPORT 60,120,30,100
40 LET N=512
45 CALL "MAX",A,V2,I
50 CALL "MIN",A,V1,I
55 WINDOW 1,N,V1,V2
60 CALL "DISP",A

RUN
```

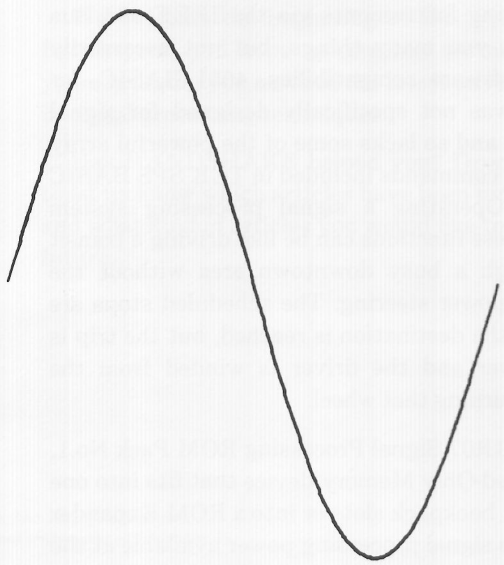


Fig. 1. *Sine wave generation and graphing using 4051 BASIC with the new ROM pack.*

```
1ØØ REM    ROUTINE TO GRAPH AN ARRAY OR WAVEFORM
1Ø5 REM  INPUTS:  A - ARRAY TO GRAPH
11Ø REM           N - SIZE OF THE ARRAY
115 REM           W1 - WAVEFORM FLAG: SET W1=1 IF WAVEFORM UNITS ARE
12Ø REM                TO BE PRINTED, AND FOR HORIZONTAL SCALING;
125 REM                SET W1=Ø IF NOT.
13Ø REM  IF W1=1 THEN THE FOLLOWING VARIABLES MUST BE SET;
135 REM           H - HORIZONTAL SCALE FACTOR
14Ø REM           H$ - HORIZONTAL SCALE UNITS
145 REM           V$ - VERTICAL SCALE UNITS
15Ø REM
155 REM  ROM PACK ROUTINES CALLED: MIN,MAX,DISP
16Ø REM
165 REM  PAGE SCREEN.
17Ø PAGE
175 REM    GET MINIMUM (M1) AND MAXIMUM (M2) OF ARRAY A.
18Ø REM
185 CALL "MIN",A,M1,I1
19Ø CALL "MAX",A,M2,I2
195 REM
2ØØ REM    CHECK FOR CONSTANT ARRAY.
2Ø5 REM
21Ø IF M2<>M1 THEN 235
215 M2=M2+4
22Ø M1=M1-4
225 REM    SET DEFAULT VALUE FOR NUMBER OF VERTICAL GRATICULE
23Ø REM    LINES (G), THEN CHECK ARRAY SIZE.
235 G=1Ø
24Ø IF N=>1Ø THEN 26Ø
245 G=N
25Ø REM    DRAW GRATICULE.
255 REM
26Ø VIEWPORT 2Ø,12Ø,18,9Ø
265 WINDOW Ø,N,M1,(M2-M1)/8*8+M1
27Ø FOR I=Ø TO G MAX 8
275 AXIS Ø,Ø,N/G*I,(M2-M1)/8*I+M1
28Ø NEXT I
285 REM    LABEL VERTICAL AXIS.
29Ø REM
295 REM       FIRST GET VERTICAL SCALE NORMALIZING FACTOR FROM SUBROUTINE.
3ØØ A1=ABS(M2) MAX ABS(M1)
3Ø5 GOSUB 77Ø
31Ø V1=K
315 VIEWPORT Ø,12Ø,18,9Ø
32Ø FOR I=Ø TO 8
325 A1=(M2-M1)/8*I+M1
33Ø REM
335 REM       NORMALIZE VALUE FOR LABELING, . . .
34Ø A1=A1/1Ø^V1
345 REM
35Ø REM       . . . AND ROUND TO THOUSANDTHS.
355 A1=INT(A1*1ØØØ+Ø.5)/1ØØØ
36Ø P$=STR(A1)
365 P=LEN(P$)
37Ø REM
375 REM       MOVE TO APPROPRIATE LEVEL ACCORDING TO WINDOW PARAMETERS,
38Ø REM       AND LENGTH OF NUMBER TO BE PRINTED.
385 MOVE N/12Ø*(21-P*1.79),A1*1Ø^V1
39Ø P$=SEG(P$,2,P)
395 PRINT P$
4ØØ NEXT I
4Ø5 REM    LABEL HORIZONTAL AXIS; SCALE FOR PROPER HORIZONTAL LABELING.
41Ø REM
415 IF W1<>Ø THEN 43Ø
42Ø H=1
425 REM       GET HORIZONTAL SCALE NORMALIZING FACTOR FROM SUBROUTINE.
43Ø A1=N*H
435 GOSUB 77Ø
44Ø H1=K
445 VIEWPORT 2Ø,12Ø,13,9Ø
45Ø FOR I=Ø TO G STEP 2
455 A1=N/G*I*H
46Ø REM
465 REM       NORMALIZE VALUE FOR LABELING,
47Ø A1=A1/1Ø^H1
475 REM
48Ø REM       AND ROUND TO THOUSANDTHS.
485 A1=INT(A1*1ØØØ+Ø.5)/1ØØØ
49Ø P$=STR(A1)
495 P=LEN(P$)
5ØØ REM
5Ø5 REM       MOVE TO APPROPRIATE POINT ACCORDING TO LENGTH OF NUMBER
51Ø REM       AND WINDOW PARAMETERS.
515 MOVE A1*1Ø^H1/H-P/2*1.79*N/1ØØ,M1
52Ø PRINT A1
525 NEXT I
53Ø REM    PRINT WAVEFORM UNITS IF APPLICABLE.
535 REM
54Ø IF W1=Ø THEN 585
545 VIEWPORT Ø,12Ø,18,98
55Ø REM
555 REM    PRINT VERTICAL SCALE UNITS, . . .
56Ø P=LEN(V$)
565 MOVE N/12Ø*(21-P*1.79),M2
57Ø PRINT V$
575 REM
58Ø REM    . . . AND VERTICAL NORMALIZATION FACTOR IF APPLICABLE.
585 IF V1=Ø THEN 632
59Ø VIEWPORT Ø,12Ø,18,95
595 U$=STR(V1)
6ØØ U$="(E"&U$
6Ø5 U$=U$&")"
61Ø P=LEN(U$)
615 MOVE N/12Ø*(21-P*1.79),M2
62Ø PRINT U$
625 REM
63Ø REM    PRINT HORIZONTAL SCALE UNITS, . . .
632 IF W1=Ø THEN 665
635 VIEWPORT 2Ø,12Ø,8,9Ø
64Ø P=LEN(H$)
645 MOVE N/1ØØ*(5Ø-P*1.79/2),M1
65Ø PRINT H$
655 REM
66Ø REM    . . . AND HORIZONTAL NORMALIZATION FACTOR, IF APPLICABLE.
665 IF H1=Ø THEN 72Ø
67Ø VIEWPORT 2Ø,12Ø,5,9Ø
675 U$=STR(H1)
68Ø U$="(E"&U$
685 U$=U$&")"
69Ø P=LEN(U$)
695 MOVE N/1ØØ*(5Ø-P*1.79/2),M1
7ØØ PRINT U$
7Ø5 REM
71Ø REM    DRAW GRAPH OF ARRAY.
715 REM
72Ø VIEWPORT 2Ø,12Ø,18,9Ø
725 WINDOW 1,N+1,M1,M2
73Ø CALL "DISP",A
735 PRINT
74Ø RETURN
745 REM    END OF 4Ø51 GRAPHICS ROUTINE.
75Ø REM
755 REM    SUBROUTINE TO NORMALIZE SCALING
76Ø REM  INPUTS:   A1 - NUMBER TO BE NORMALIZED
765 REM  OUTPUTS:  K - NORMALIZATION FACTOR (POWER OF 1Ø)
77Ø K=Ø
775 IF ABS(A1)<1ØØØ THEN 795
78Ø K=K+1
785 A1=A1/1ØØØ
79Ø GO TO 775
795 IF K<>Ø THEN 82Ø
8ØØ IF ABS(A1)=>1 THEN 82Ø
8Ø5 K=K-1
81Ø A1=A1*1ØØØ
815 GO TO 8ØØ
82Ø K=K*3
825 REM    K NOW EQUALS POWER OF TEN NORMALIZATION FACTOR.
```

Fig. 2. *4051 BASIC routine used to graph arrays and waveforms with graticules and scaling information.*

that in line 275, the AXIS statement employes the expression, (M2−M1)/8*I+M1, to construct the eight vertical graticule divisions (eight division lines plus a base line at M1). Because of the possible rounding or truncating during a math operation, the last time through the loop (I=8) might result in a value greater than M2. Using the same expression in the WINDOW statement as occurs the final time through the axis loop, therefore, ensures a maximum Y value for the graticule window that will always be equal to or greater than M2. Without this clever technique, we would, at times, get a graph minus a top graticule line.

The graph's labels are handled in a way which avoids printing long, difficult to read numbers. This

continued on page 15

A closer look at some basic signal processing operations

If scores were kept, false assumptions would probably come out the winner for generating measurement errors. This is particularly true in using digital signal processing systems since analog operations are being digitally executed and most of us are conditioned to analog rather than digital thinking. However, the course to preventing the resulting errors of illusion is relatively painless. In virtually all cases, it is certainly less painful than reverting to the old analog methods of solving analog problems. The course is to simply gain and maintain an organized and detailed awareness of what happens to a signal during digital processing.

To help you gain (if you are new to digital signal processing) and maintain (if you are an old hand at it) this organized and detailed awareness, let's take a digital look at some basic signal processing operations, starting with array storage and moving right into some specialized commands like MAX, MIN, CROSS, INT, and DIFF.

Getting the data straight

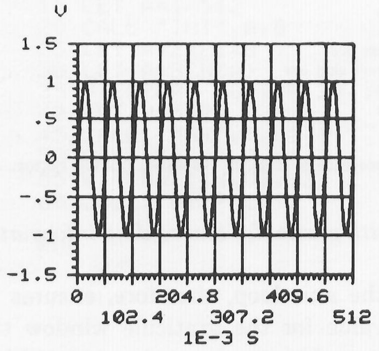
The first step in digitally analyzing an analog signal is to acquire and convert the signal to a digital format. Acquisition is done in a familiar analog manner, often with digital versions of analog instruments. Conversion takes place soon after with an analog-to-digital converter (ADC).

State-of-the-art ADCs are of such high resolution that a digitized signal may appear on a display as a continuum (Fig. 1a). But it isn't. It's a series of dots. The illusion is similar to that of newspaper photos. The printed picture appears to the unaided eye as a continous-tone photo but, on close inspection, turns out to be a series of dots—in effect, a digital version of the analog world. With displays of digitized signals, however, the illusion may even be further heightened by straight-line connection of the dots (Fig. 1b).

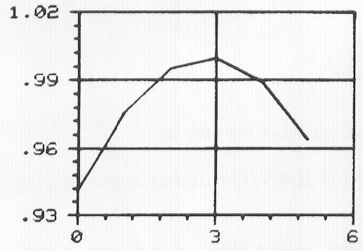
This illusion of continuity has its visual benefits. For one, it serves to remind us that we are analyzing an analog signal. But, at the same time, it is important to remember that the analysis is actually taking place digitally, on a digital or dot version of the analog signal.

It is also important to remember that this digital version of the signal is simply a string of numbers —an array—and not the actual signal. The numbers, themselves, represent vertical signal amplitudes at discrete sample points along the signal. The position of each number in the series represents its horizontal time location on the signal (Fig. 2). Memory areas, called arrays, are designated in the signal processing system for storing these strings of numbers, and they can be thought of as tables such as you might construct with pencil and paper. The big difference is that signal processing arrays or tables usually have many more entries—512 elements or more—than you would normally tabulate by hand.

The signal processing array is the ADC's view of the signal. The array arrangement, its scale factors, digital sampling interval, and the data contained in it are all that the signal processing software has to work on. What we might be tempted to assume about the signal's activity before sampling and digitizing begins, between sample points (array



a. The 512 samples of a digitized signal are close enough to each other to provide the appearance of a continuous signal.



b. However, a closer look at one of the peaks reveals that the digitized signal is a series of dots, which are often straight-line-connected for display.

Fig. 1. *Don't let the illusion of continuity cloud your thinking when you are dealing with digitized signals.*

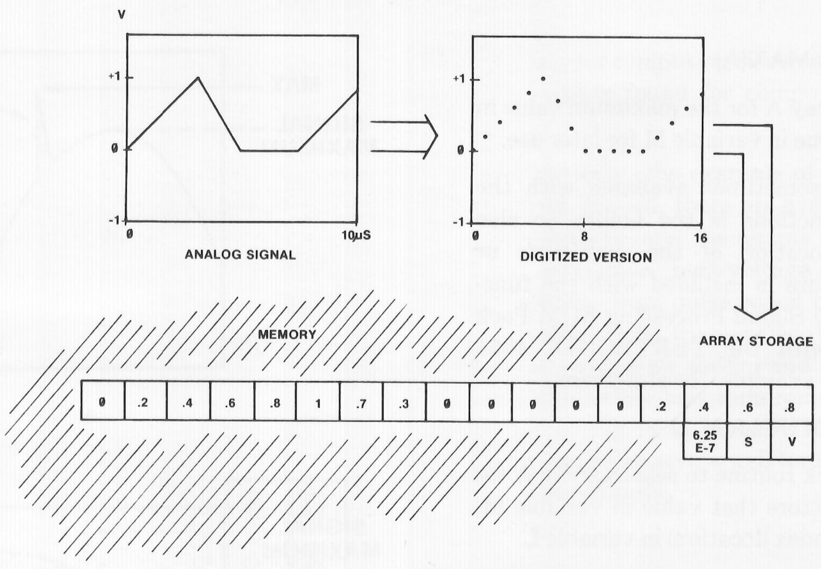


Fig. 2. *Before analog signals can be processed, they must be digitized and stored in the processing unit's memory.*

elements), or after sampling ends has no affect on the system. The system operates just on the numbers given it. Only in special cases does it make some limited assumptions about data possibilities beyond the array ends.

Some basic processing—by the numbers

Almost without exception, signal processing takes place element-by-element, starting with the first element in the array and progressing to the last element in the array.

For example, let's say you have acquired a waveform, and it has been digitized and transferred into a software defined array. Now maybe you want to add a constant to it: let's say the signal is stored in array A, it is a voltage, and you want to add a four volt bias to it. The signal processing BASIC statement for doing this is

LET A=A+4

When this executes, the first signal element in A has a value of four added to it. Then four is added to the second element, to the third, and so on until all of the array elements have been processed.

This same element-by-element process is also used in subtracting a constant from an array, multiplying an array by a constant, or dividing an array by a constant. It is also used in adding one array to another array, subtracting one array from another, multiplying one by another, or dividing one by another. For example, in

LET A=A/B

where A and B are arrays, the first element of A is divided by the first element of B, the second element of A is divided by the second element of B, and so on until each element of A has been divided by the corresponding element in B.

It all seems ho-hum simple. And it is, if you avoid the more common pitfalls by keeping the following DOs and DON'Ts in mind:

- DON'T attempt to combine (+, −, /, *) arrays of different lengths since the element-by-element processing will not complete.
- DON'T combine (+, −, /, *) arrays with different sampling intervals because the different time scaling may lead to erroneous or confusing results.
- DO be cautious of dividing by zero or very small numbers (such as occur at zero crossings on repetitive waveforms) since this can lead to uninterpretable results.
- DO know the dynamic range of your processing software and how it rounds or truncates numbers.

Organized searches

Most signal processing needs go beyond simple mathematical combinations of constants and waveforms. In many cases, the processing is done on a single waveform and amounts to searching it for specific points, such as a maximum or minimum value or some value in between. To help speed this type of waveform processing, signal processing software usually contains a MAX, a MIN, and a CROSS function.

The MAX and MIN functions are straightforward in operation. When invoked, they simply search the specified waveform array for the maximum or minimum value stored in the array. For example, in TEK SPS BASIC software, the statement

A closer look at some basic signal processing operations

LET M=MAX(A)

causes a search of array A for the maximum value in A and stores that value in variable M for later use.

Another feature sometimes available with the MAX and MIN functions is the ability to also return the array location of the maximum or minimum. This feature is included with the functions on the 4051R07 Signal Processing ROM Pack designed for use with the TEKTRONIX 4051 Graphic System. For example, the statement

CALL "MIN",A,M,L

causes the ROM Pack routine to search array A for its minimum value, store that value in variable M, and store the array index (location) in variable L.

In all cases, the MAX and MIN functions operate just on the values stored in the waveform array. If you do not take care in digitizing your signals, the values returned by the MAX or MIN may not exactly reflect what you might be expecting from the analog signal. Two cases where this can happen are illustrated in Fig. 3. Avoiding them is a matter of staying keenly aware of what your signals are doing and how they are being digitially represented.

Another more general search is provided by the CROSS function. Again, it operates on the array version of the signal, and the essentials of its operation are illustrated in Fig. 4. Special options are usually added in specific software packages to narrow the cross search. As examples, the CROSS function may optionally be able to begin at a specified point in the array, or it may be directed to search only a specific zone in an array, or it may be directed to look for a crossing beyond the first crossing.

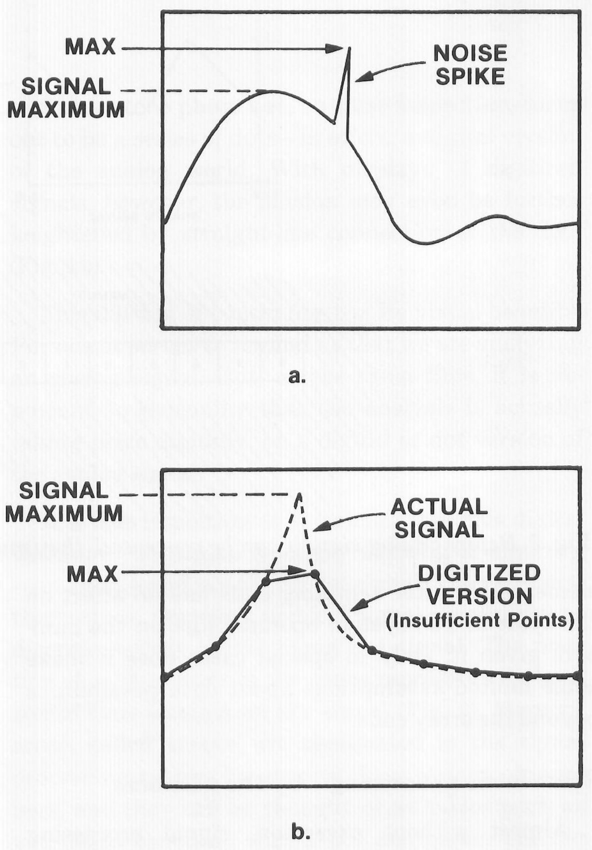


Fig. 3. *Most errors in using the MAX and MIN functions stem from "ideal" assumptions about the "real" waveform or its digitized representation. As examples, in "a" a noise spike causes a higher than expected value for the maximum, and in "b" insufficient sampling can cause a lower than expected value.*

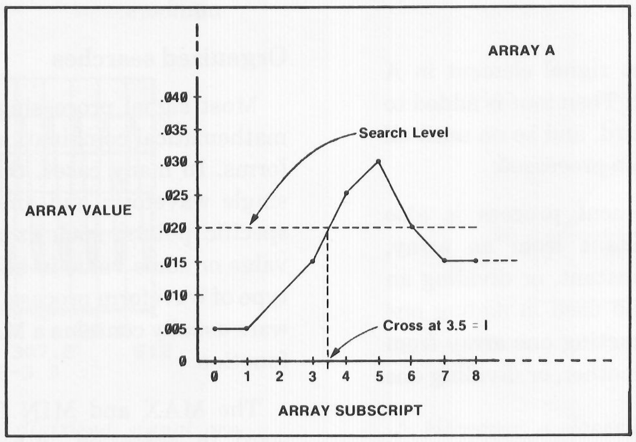
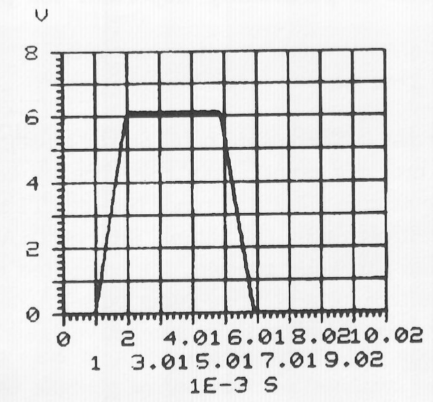


Fig. 4. *Searching array A with the TEK SPS BASIC LET I=CRS(A,.02) statement or the CALL "CROSS",A,.02,I statement of the 4051 Graphic System operating with the 4051R07 Signal Processing ROM Pack. The search starts with element zero and moves through the array looking for either a value equal to the search level or a set of values indicating that the search level has been crossed. In the latter case, the returned cross location is an interpolated value.*

Rise time, fall time, pulse width

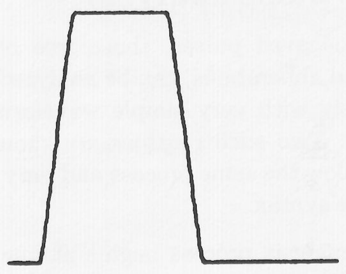
Combined in a program with the MAX and MIN functions, the CROSS function provides a variety of signal processing possibilities. Probably the most universal of these is pulse or transient analysis. Example programs and waveforms for a simple analysis sequence are shown in Fig. 5. The routines there, one in TEK SPS BASIC and the other in 4051 BASIC with the Signal Processing ROM Pack, search the waveform array to determine both the MAX and MIN values. These are, in turn, used to determine 10% and 90% levels on the pulse. These levels are then searched with the CROSS function to find data for computing rise and fall times. Also, the 50% level is found for computing the 50% pulse width.

Admittedly, the example of Fig. 5 is somewhat idealized. Indeed, there are even a variety of idealized pulses beyond the simple square pulse used—transients from high-voltage protection testing, pulse trains from pulse coded modulation systems, and pulsed radio frequencies, to name just a few. Each class has its special and often differing analysis approaches and requirements. But the pulse shown in Fig. 5 is probably the most commonly known type and serves well for illustrating the basic analysis concepts.



```
RISE TIME= 7.98985E-04
FALL TIME= 7.98986E-04
50% WIDTH= 3.91389E-03
```

```
100 REM TEK SPS BASIC PULSE ANALYSIS
105 LET A=A-MIN(A)
115 LET MA=MAX(A)
120 LET T9=CRS(A,.9*MA)
125 LET T1=CRS(A,.1*MA)
130 LET RT=(T9-T1)*HA
135 LET T9=CRS(A(T9+.5),.9*MA)
140 LET T1=CRS(A(T1+.5),.1*MA)
145 LET FT=(T1-T9)*HA
150 LET T5=CRS(A,.5*MA)
155 LET PW=(CRS(A(T5+.5),.5*MA)-T5)*HA
160 END
300 REM GRAPH PULSE ANALYSIS RESULTS
305 VIEWPORT 50,350,410,650
310 SETGR VIEW
315 GRAPH WA
320 SMOVE 0,300
325 PRINT "RISE TIME=";RT
330 PRINT "FALL TIME=";FT
335 PRINT "50% WIDTH=";PW
340 END
```



```
RISE TIME=7.989852414E-4
FALL TIME=7.989852414E-4
50% WIDTH=0.00391389432485
```

```
100 REM 4051 BASIC PULSE ANALYSIS
105 CALL "MIN",A,M1,I1
110 LET A=A-M1
115 CALL "MAX",A,M2,I2
120 CALL "CROSS",A,0.9*M2,T9
125 CALL "CROSS",A,0.1*M2,T1
130 LET R=(T9-T1)*H
135 CALL "CROSS",A,0.9*M2,T9,2
140 CALL "CROSS",A,0.1*M2,T1,2
145 LET F=(T1-T9)*H
150 CALL "CROSS",A,0.5*M2,T5
155 CALL "CROSS",A,0.5*M2,T6,2
160 LET W=(T6-T5)*H
165 END
300 REM GRAPH PULSE ANALYSIS RESULTS
305 PAGE
310 CALL "MIN",A,M1,I1
315 VIEWPORT 6.4,44.9,52.1,82.6
320 WINDOW 1,512,M1,M2
325 CALL "DISP",A
330 VIEWPORT 0,130,0,100
335 WINDOW 0,130,0,100
340 MOVE 0,38.5
345 PRINT "RISE TIME=";R
350 PRINT "FALL TIME=";F
355 PRINT "50% WIDTH=";W
360 END
```

Note: *The pulse is stored in array A and the sampling interval (△t) is stored in HA for the TEK SPS BASIC routine and H for the 4051 BASIC routine.*

Fig. 5. *Two simple routines, one in TEK SPS BASIC and the other in 4051 BASIC, for analyzing well-behaved pulses. Though the language syntax varies, the analysis concepts and processes remain the same. For more details, see the accompanying short article, "Basic pulse analysis."*

Also, in practice, pulses of any kind will often carry noise and may be distorted by overshoot, undershoot, and various degrees of ringing. When these conditions appear, you must take additional programming steps to ensure that the cross search, for example, does not find the 90% point on a noise spike or on part of the ringing instead of on the actual pulse edge. Also, remember that on noisy or ringing pulses, the maximums and minimums will be defined as the peaks of the noise or ringing, which may not be what you want. All of these things can be taken care of by additional programming steps, by using the zoning options provided in some software packages, or even by more sophisticated approaches to pulse analysis.

One approach, commonly used for distorted or noisy square pulses, uses a histogram routine for statistically defining the pulse top and bottom. Other approaches include various combinations of the maximum and minimum functions along with the histogram. These approaches and more will be discussed further in the next issue of HANDSHAKE. But, for now, we need to finish the groundwork by looking at two additional analysis operations, integration and differentiation.

The power of integration

Beyond rise time and fall time, it is often important to know the energy contained within a pulse. This is particularly important for people

Basic pulse analysis

Well-behaved pulses, those free of perceptible noise and abberations, can be analyzed quickly and accurately with very simple waveform processing routines. Two such routines are shown in Fig. 5. They follow the same process and vary only in their language syntax.

The analysis process begins at line 105. In the TEK SPS BASIC routine, line 105 finds the minimum value of the pulse and subtracts that value from the pulse array. The same thing is done in lines 105 and 110 of the 4051 BASIC routine, where line 105 calls the Signal Processing ROM Pack MIN function into action and then subtracts the minimum value (the minimum value is stored in M1 and its location in I1) from array A in line 110.

Finding the minimum and subtracting it is a very simple yet highly important step. Regardless of pulse position (positively or negatively biased) or polarity, subtracting the minimum forces it to the zero-level base line. By forcing the pulse to this "standard" position, a lot of extra manipulation required for special cases can be eliminated.

With the pulse shifted to the zero level, a determination of its parameters can begin. The first parameter found is pulse height since it is key to determining the rest of the parameters. And, since the pulse has previously had its minimum subtracted so that all its values are positive from zero to the pulse maximum, pulse height is simply the maximum value of the pulse. This is found in line 115 of each program.

The next parameter is rise time (fall time or negative rise time for a negative-going pulse). This is found by lines 120, 125, and 130 of each program. Line by line, these segments use the cross function to find T9 (the 90% crossing point on the pulse), T1 (the 10% crossing point), then rise time by computing the sample point difference between T1 and T9 and multiplying by the sample interval (HA or H). For this particular process, the computed rise time is positive for a positive-going pulse. A negative-going pulse is indicated by a negative value for the rise time.

Fall time is computed in the next three lines of the program by much the same technique. The difference, however, is that the cross searches are set to find crosses on the second transition rather than the first. In the case of the TEK SPS BASIC program, CRS (A(T9+.5),.9*MA) causes the cross search for 90% of MA to begin half an increment after the position of the first 90% crossing. The "CROSS",A,.9*M2,T9,2 operates slightly differently by essentially saying, "search array A along a level of .9*M2 until you cross the waveform, putting the location of that cross into variable T9, but don't stop the search until you have found the second crossing." The crossing points thus found for the 90% and 10% levels of the second transition are used in line 145 of each program to compute fall time. Again, the returned results will be positive to indicate a positive-going pulse or negative to indicate a negative-going pulse.

Then, after finding fall time, the final lines of the 100-series lines compute 50% pulse width in much the same manner. Of course, further lines could be added to compute other widths, proximal and distal times. Also, FOR loops and an incremented crossing variable could be used to perform the same analysis sequence on second and third pulses in a series of pulses.

dealing with lasers and for people involved in transient recording and analysis.

Most waveforms acquired for measurement are not directly representative of energy. Instead, they are usually only proportional to energy and must, therefore, be at least corrected by some multiplicative factor. Once this is done, the total energy in the waveform, a laser pulse for example, can be computed by integrating to obtain the area contained by the pulse.

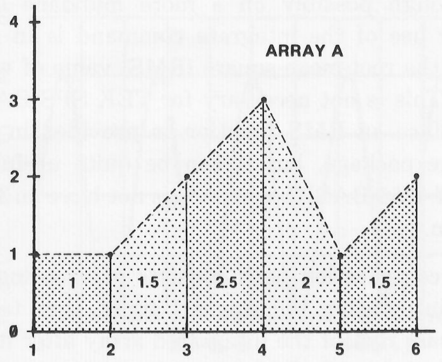
The specific integration routine used by TEK SPS BASIC and the 4051 Signal Processing ROM Pack operates by the following algorithm.

$$B(1)=0$$
$$B(n)=B(n-1)+.5*(A(n-1)+A(n)) \quad \text{for } n=2, 3, \ldots, N$$

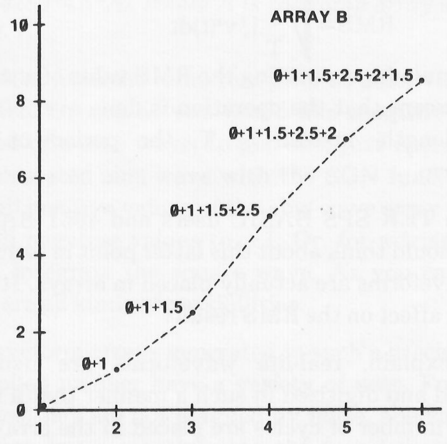
In this algorithm, A is the array being integrated, B is the array that will contain the results, n is the element number, and N is the last element of the array. This algorithm follows the trapazoidal rule for integration and amounts to computing and summing the incremental areas under the straight-line-connected data points of the array (see Fig. 6). In the case of a waveform array, the algorithm changes slightly to incorporate the time between samples ($\triangle t$) such that

$$B(n)=B(n-1)+(\triangle t/2)*(A(n-1)+A(n)) \quad \text{for } n=2, 3, \ldots, N$$

In either case, the total area under an array or waveform is the value contained in the last element of the resultant array, B(N). This is the value you will be interested in when computing pulse energy.



a. The six-element array with areas shown for integration.



b. The six-element array containing the point-by-point integral of a.

Fig. 6. *The software integration routine of TEK SPS BASIC and the 4051 BASIC Signal Processing ROM Pack produces a point-by-point sum of areas as the result. The total area under the waveform is contained in the last element of the resultant array.*

Basic pulse analysis

The final steps of the program get the data out. These programming steps, contained in the 300-series of lines in Fig. 5, consist primarily of graphics commands specialized to the systems being used. Because of their specialized nature, saying more about them wouldn't really add much conceptually to what is going on. They simply operate as a group, often interactively, to output the date in the format you choose. In the case of Fig. 5, they cause the pulse and its computed parameters to be output to a graphics terminal in the format shown.

For the results output in Fig. 5, it is interesting to note some slight differences. Most obvious, of course, is the fact that the TEK SPS BASIC routine plotted the pulse on a graticule while the 4051 BASIC routine did not. An additional 4051 BASIC routine, discussed in the "Getting the most out of TEK BASIC graphics" column in this issue, can be used to overlay a graticule on the pulse. Probably of more concern than the graphics, however, are the differences in the computed pulse parameters. First of all, notice that the differences are very slight. For example, the fall time output by the TEK SPS BASIC routine is 7.98986E-04 while that output by the 4051 BASIC routine is 7.989852414E-4. Since the same pulse generation routines were used for each analysis, the differences can be attributed to system differences. Different processors and software packages were used and slight variations in algorithm implementations, roundoff methods, and so forth are to be expected. In this case, a quick computation of percentage difference shows a figure orders of magnitude less than that normally tolerated in virtually any measurement situation.

A closer look at some basic signal processing operations

Although possibly on a more mundane level, another use of the integrate command is in computing the root-mean-square (RMS) value of waveforms. This is not necessary for TEK SPS BASIC users since an RMS function is provided in that software package, but it can be quite useful for users of 4051 BASIC, which does not have an RMS function.

The computation path consists of squaring the array, integrating the squared array, then taking the square root of the integrated array after it has been divided by the number of array elements. The last element of the resulting array is the RMS value of the original array and corresponds to

$$RMS=\sqrt{\frac{1}{T}\int_0^T v^2(t)dt}$$

the formula for computing the RMS value of a waveform, except that the operation is done over N, the array length, instead of T, the period of the waveform.

Both TEK SPS BASIC users and 4051 BASIC users should think about this latter point in terms of how waveforms are actually placed in arrays. It has a direct affect on the RMS result!

To explain, real-life waveforms are usually acquired and digitized in such a manner that a non-integer number of cycles are placed in the array. It may be 3.674 cycles or some other oddball number of cycles. Rarely do the periods of real-live waveforms correspond nicely enough to sweep speed for an exact integer number of cycles to be acquired into an array.

So, in real-life operation, N in no practical way corresponds to T. This means that if you blithely apply the RMS function of TEK SPS BASIC or use integration with the 4051 to compute the RMS value, you will get the RMS value of the array values but not necessarily the RMS value of the waveform being represented. Remember, the software works on the numbers you give it . . . all of them. So, to get the correct RMS value for a stored waveform, you must pick out of the array only those values representing an integer number of cycles for use in the computation.

For most types of waveforms, you can use the CROSS function to get the integer number of cycles. Begin with the first array value as the search level. Then search along this for the second repetition of the value, or whatever repetition represents a period of the waveform you are dealing with. Then note the array location where the repetition begins. Let's say, for example, that this occurs at element 277 of the 512-element array. To compute the RMS value of the waveform cycle with TEK SPS BASIC, simply use the zone feature with the RMS function —LET R=RMS(A(0:277)) for the example crossing. This causes the RMS function to be confined to elements 0 through 277 of the array, which we said for the sake of example would represent one cycle of the stored waveform.

In the case of 4051 BASIC, which does not provide the zone operation, you must confine your operation by other means to the region of the array defined to cover one cycle of the waveform. This is easily done by noting (via the CROSS command) the element marking the first repetition. Again, we'll say it is 277. Go ahead and square the entire array and integrate it. Now comes the change. Divide the squared and integrated array by the element number marking the end of the first repetition, 277 in this case. Take the square root of the divided array, then look at element 277 for the RMS value of the waveform.

Beyond pointing out the possibility of making grave errors in computing RMS values of waveforms, all of this serves to reiterate an important point made near the beginning of this article. Waveform processing software can be designed to operate on a variety of waveform arrays. But it can only operate on the numbers placed in those arrays. It cannot make assumptions from those numbers about what kind of waveform might be in the array or about its arrangement in the array. Therefore, successful measurements still hinge on your knowledge and skill. The software just removes the drudgery of repeated operations and the chance of error in highly complex and involved manipulations.

And, to continue to reiterate the importance of gaining and maintaining an organized and detailed awareness of what happens to a signal during digital processing, let's take a look at another aspect of the integration routine. There are some further benefits to be derived from a detailed awareness of what is done.

In particular, let's look at the property that, as the number of samples or array elements considered increases, trapazoidal integration comes closer to simply summing the sample values. A simple experiment in TEK SPS BASIC demonstrates this.

The experiment consists of using a small array, say ten elements, and setting the array to a value equal to the reciprocal of the number of elements, 1/10 in the case of the ten-element array. Integrating this ten-element array produces a final value of 0.9 for the area contained (there are 9 intervals, each containing an area of 1/10). On the other hand, summing the ten elements produces a value of one. Increase the array size to 100. Set it to 1/100, and repeat the process. The integrate routine returns an array with a final value of 0.989999, and summing the 100 elements of 1/100 still results in one. Try it with 300 elements. Then go to 512 elements, the standard array size for signal processing, and the integral result is 0.998047—very close to a value of one. In fact, it's within 0.2%.

So what good is this seemingly trivial information?

Well, for one, TEK SPS BASIC users might find it useful for summing the values of the elements in an array. It's faster than a FOR loop. Users of 4051 BASIC do have an advantage here, though, in that 4051 BASIC already has a SUM function for summing array elements.

But there is more to this than just summing. Let's look at the array resulting from integrating a 512-element array whose elements have been set to 1/512. Such an array is shown in Fig. 7. As seen there, the result is a linear ramp. In itself, such a ramp can find immediate use as a comparison standard in various linearity tests and measurements. A less obvious use, however, is as a seed for generating a variety of ideal waveforms.

As an example, let's generate 4.5 cycles of a sine wave. Array A is the generated ramp running from zero to nearly one. Multiply this ramp by 2*3.14159*4.5 (2π times the number of cycles desired), and you have a new ramp running from zero radians to the value in radians corresponding to the angular dimension of 4.5 cycles of a sine wave. Use the sine function, LET A=SIN(A), and the 2*3.14159* 4.5 radian ramp is turned into the sine wave shown in Fig. 8. Or maybe you would like to have the sine wave advanced or delayed by say $\pi/2$ radians. To do this, simply add or subtract

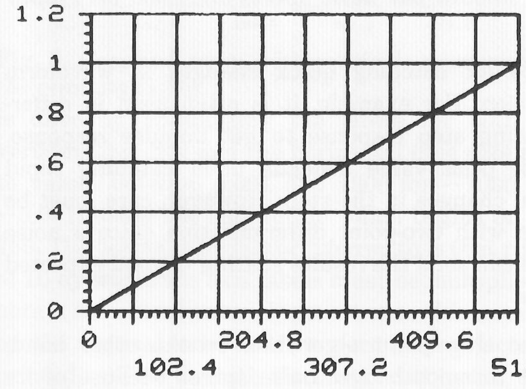


Fig. 7. *A linear ramp, running from zero to nearly one, obtained by integrating a 512-element array set to a constant value of 1/512.*

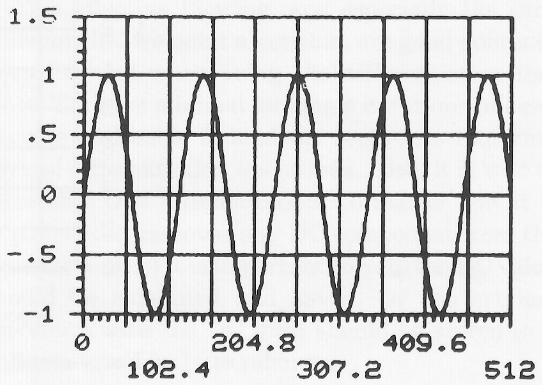


Fig. 8. *Sine wave array generated by LET A=SIN (2*3.14159*4.5*A), where A is the ramp array shown in Fig. 7.*

3.14159/2 from the ramp just prior to using the sine function. Or maybe you would like a square wave instead of a sine wave; to get one, just operate on the generated sine wave with the SGN function to turn all positive values of the sine wave array to +1 and all negative values to −1. Or, for a triangular wave, integrate the square wave. As you can see, there are all kinds of possibilities.

Waveform arrays generated in such a precise and controlled manner have a variety of uses. For one, because they are not plagued by noise, aberrations, etc., you know exactly what the waveform is and what its parameters are. Thus, you have an ideal learning aid for exploring the various functions and operations of software. Also, such known waveforms are excellent for use in simulations and as standards for checking out and debugging analysis routines.

Looking for change

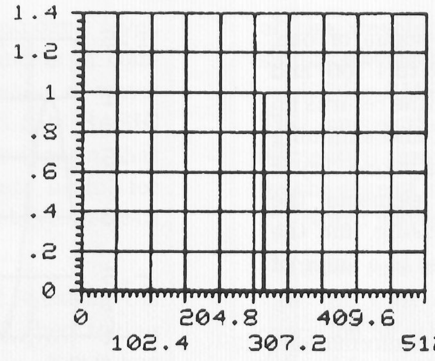
The opposite of integration is differentiation. And, in signal processing systems, this operation can be implemented in several ways. Both TEK SPS BASIC and the 4051 Signal Processing ROM Pack provide differentiation in two-point and three-point algorithms.

The two-point algorithm operates as follows

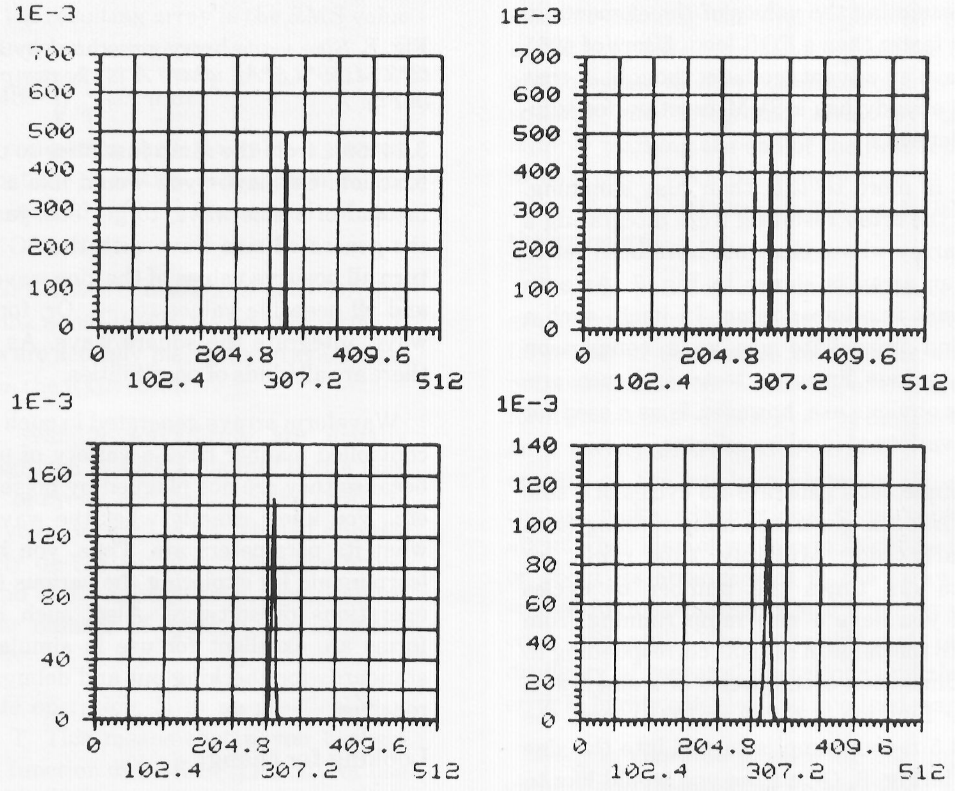
$$B(n)=A(n+1)-A(n) \qquad n=1, 2,..., N-1$$
$$B(N)=B(N-1)$$

where A is the array being differentiated, B is the array that contains the results, n is the array element being operated on, and N is the last element of the array. As can be seen from the above equations, the algorithm simply computes the incremental change (slope) on an element-by-element basis.

In the three-point derivative, the operation is a



a. Unity-amplitude, unity-width test spike.



b. Two-point derivative smoothing: top, one interation; bottom, 30 iterations.

c. Three-point derivative smoothing: top, one iteration; bottom, 30 iterations.

Fig. 9. *Alternately integrating and differentiating will tend to smooth out noise spikes and other very rapid transitions.*

little more complex, computing the slopes as follows

B(1)=(−3*A(1)+4*A(2)−A(3))/2
B(n)=(A(n+1)−A(n−1))/2
for n=2, 3,...,N−1
B(N)=(A(N-2)−4*A(N−1)+3*A(N))/2

Except for some special manipulation at end points, the algorithm obtains the change at each point by computing the average change between points one increment to either side.

In use, the two-point algorithm is more sensitive to change—able to turn the corners faster. It is the choice for catching quick changes in waveform direction. For example, it is often used in differentiating step response to get impulse response, whose peak value depends upon catching rapid initial changes in the step. However, care must be taken with two-point differentiation since a noise spike can send the results soaring beyond expected levels.

The three-point algorithm, on the other hand, tends to smooth over noise spikes because of its averaging. This makes it the choice for computing rates of rise or fall where you would prefer to suppress fast changes. In fact, the averaging property can be used to reduce or smooth (actually filter) out noise spikes on waveforms. This is done by alternately integrating and differentiating the waveform. Be sure the integration is done first, however, since differentiating removes any constant (DC) component from the waveform.

To get a better feel for the properties of the two differentiation algorithms, refer to Fig. 9. There, a spike is integrated then differentiated in a loop using first the two-point then the three-point algorithm.

Figure 9a shows the spike to be operated on. It is a single point of value one in an array of zeros. In Fig. 9b, the spike has been integrated and two-point differentiated once, then integrated and two-point differentiated 30 times. In Fig. 9c, the process was repeated with the three-point algorithm. In both cases, note that the spike is broadened and its amplitude reduced—smoothed—but more so for the three-point derivative. Also, note that two-point differentiation causes the spike to be shifted to the left, while three-point differentiation does not.

This effective filtering, and especially the time shift for the two-point algorithm, are good points to keep in mind when using the differentiation routines. They are minimal for single iterations but can become noticeable in analysis sequences requiring several differentiation operations. Also, it is well to remember that differentiation, no matter how it is implemented, removes any DC component from the waveform. So, if it is important to you, the DC value should be computed and saved. Or the original waveform with the DC term should be stored in a separate array for later reference.

In fact, in any waveform processing situation, it is a good idea to archive the original waveforms and do the processing on copies. This way, any unexpected turns in processing needn't be disastrous. You can always go back to the original data for another copy of the waveform. This provides a worthwhile measure of comfort when you are new to signal processing or trying out new routines, and it is certainly a mandatory practice for anyone dealing with transients or any other one-of-a-kind waveform.

By Bob Ramirez
HANDSHAKE Staff

continued from page 5

New ROM Pack makes quick graphing possible

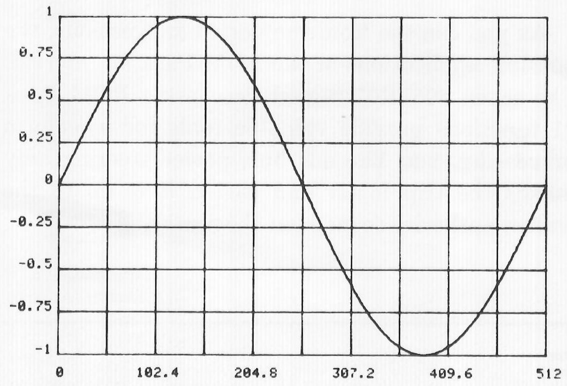


Fig. 3. *Graphed array with graticule overlayed by the program in Fig. 2.*

is done by normalizing the labels for both the vertical (line 295) and the horizontal (line 425) axes to values between 0.001 and 1000 (or −1000 and −0.001) using the subroutine which begins on line 755. The normalizing factor referred to is that power of 10 by which the axis labels must be multiplied to obtain the actual value. Figure 4 uses this notation on the vertical axis, and the normalizing factor is printed as E-69 beneath the units label (volts). (The negative 69th power of ten is expressed as E-69 in BASIC).

Once the graticule and all the labels are made, the

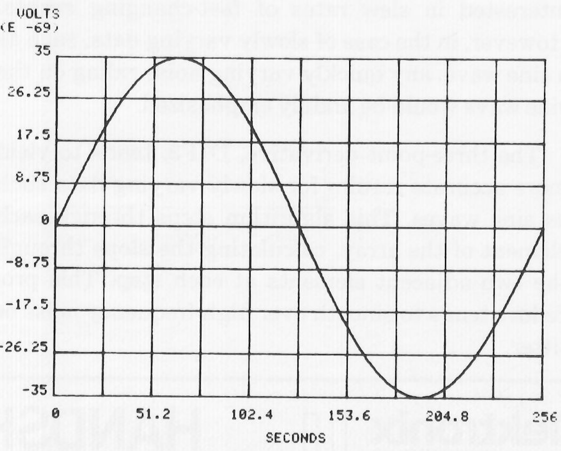


Fig. 4. *Graphed waveform, including scale factors, with graticule overlayed by the program in Fig. 2.*

data array is graphed. Line 725 sets the data window so the graphed array always occupies the entire graticule (the maximum X and Y axis values of the array equal the size of the graticule).

4051 users will appreciate the speed with which the graphics program executes — a courtesy of the 4051R07 Signal Processing ROM Pack. The MIN and MAX operations are 10 times faster than their 4051 BASIC program equivalents, DISP is at least twice as fast.

By Walt Robatzek, HANDSHAKE Staff,
program contributed by
Laurie DeWitt, SPS Software Engineer

continued from page 3

The 4051 Signal Processing ROM Pack — something new

other hand, the array values do not cross the threshold at all (or not the number of times you requested) a minus one (−1) is returned.

The CROSS command is useful for examining switching transistor parameters. CROSS can be used to find the 10% and the 90% levels of a pulse which in turn will yield delay time, rise time, fall time, etc. (see "How Quick Is Your Switch", HANDSHAKE, Vol.2 No.2 Winter 76-77). Some RF measurement systems find the CROSS function helpful. "Swept RF Measurements — A Realistic Approach", HANDSHAKE, Vol.3 No.2 Winter 77-78 uses a crossing function in a programmed search for sweep blanking intervals.

DIF2 (Two-point derivative) Provides a simple, forward two-point derivative (slope) of specified data.

DIF3 (Three-point derivative) Similar to DIF2, but determines the slope using a three-point algorithm.

DIF2 calculates slope by stepping through the array and determining the slope between each two array elements. This procedure results in accurate slope estimates of rapidly changing data: pulses, transients, square waves — a valuable tool if you're interested in slew rates of fast-changing events. However, in the case of slowly varying data, such as a sine wave, any quickly varying noise riding on the sine wave would be unduly emphasized.

The three-point derivative, DIF3, tends to yield more accurate results for slowly varying data such as sine waves. This algorithm steps through each element of the array, calculating the slope through the two adjacent elements at each step. This procedure tends to smooth over high-frequency noise or jitter.

INT (Integral) Determines the integral (area under the curve of specified data) by using a trapezoidal approximation to an ideal integral.

The INT function rapidly determines the energy contained in a pulse — a real time saver in laser and optical fiber measurements.

DIF2 or DIF3 and INT are inverse operations. In mechanical measurements, for instance, where your acquired data array might be graphed as distance versus time (feet vs seconds), differentiating the array will return velocity (feet/sec. vs time). Taking the second derivative yields an acceleration array (feet/sec^2 vs time). The integral of acceleration returns velocity, and likewise, the integral of velocity brings you back to distance vs time.

DISP (Display) Provides a graph of the raw data. No graticule or labels are provided.

Graphing is done in less than half the time required by an equivalent 4051 BASIC routine. If you cannot know in advance the numeric range of the data you wish graphed, possible clipping of the data extremes can be avoided by using the MAX and MIN ROM Pack functions. These can acquire data limits and assign them to variables used by the WINDOW command. Then, when DISP is used, a graph of all the data in the array results.

As you can see from the varied suggestions, the possible applications of the ROM Pack are myriad. The seven 4051R07 Signal Processing ROM Pack #1 functions provide valuable tools for waveform processing, but, like our bus, power steering only makes the trip faster and easier, it is up to the transit authority to map out the routes.

by Walt Robatzek,
HANDSHAKE Staff

Tektronix® COMMITTED TO EXCELLENCE

HANDSHAKE
Newsletter of the Signal Processing Systems Users Group

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

ADDRESS CORRECTION REQUESTED

AX-4116