SESSION 7 INTERACTIVE DISCUSSIONS/POSTERS

Chaired by Sarah Ransdell, New College

Tachistoscopic presentation and millisecond timing on the IBM PC/XT/AT and PS/2: A Turbo Pascal unit to provide general-purpose routines for CGA, Hercules, EGA, and VGA monitors

ROBERT E. HAUSSMANN
Yale University, New Haven, Connecticut

A review of the literature on adopting microcomputers for use in tachistoscopic research paradigms illustrates the need for a comprehensive package to effectively deal with the wide variety of video displays and microcomputers used in experimental settings. In addition, because of the large number of possible configurations created when video displays and computer systems are combined, there is a need to be able to efficiently drive any such combination without rewriting program code. A Turbo Pascal unit is presented to provide standard tachistoscopic routines compatible with monochrome, CGA, Hercules, EGA, and VGA video technology. Procedures that synchronize both text and graphics mode stimuli with the vertical retrace pulse are given, as are routines to provide access to a timer with better than millisecond precision. Pascal code is described that will run on the IBM PC, XT, AT, or PS/2, as well as on any compatible machine. The versatility of the unit allows the experimenter to use generic video and timer commands that will automatically adapt at run time to the system that is being used, without further experimenter intervention.

As microcomputer technology increases at an ever-growing rate, prices of powerful system configurations continue to drop. Complete workstations boasting clock speeds of 33 MHz, extended resolution monitors, and hard disk storage on the order of 200 MB are available for less than \$2,000. Yet although many researchers use such systems daily for word processing and statistical analysis, many do not utilize the machines' full capabilities. These systems can be employed as fully automated data collection devices, and the researcher who addresses a few of the limitations of the hardware can thus use the microcomputer in many behavioral research settings.

Controlling Stimulus Onset

The attempt to use a microcomputer to generate visual stimuli with millisecond precision raises several technical issues, the most important of which consists of the limitations of the video display hardware. VGA (video graphics array) displays are capable of presenting images at great resolution (up to $1,024 \times 640$ pixels). When one considers, however, how the information moves from the computer to the display, one discovers that several possible problems can arise in the attempt to accurately control short-duration stimuli.

A video display operates much as a common television set does. In the case of a color monitor, three electron beams rapidly scan the entire screen, from left to right on each of the (up to) 640 horizontal scan lines, one at a time. Three beams are needed to provide different colors, but the monitor can be thought of as using only one beam (as in the case of a monochrome monitor), because all three travel together. This electron beam is actually a stream of electrons that energizes the phosphor on the inner surface of the screen. Thus, in order for the entire

This material is based upon work supported under a National Science Foundation Graduate Fellowship. The author would like to thank Ed Domber and Janet Davis for guidance in the project from which this work is an offshoot. Special thanks also to Bethann Lefsky for comments on earlier drafts, and to the anonymous reviewers for their helpful suggestions on a preliminary version of this article. Correspondence should be addressed to Robert E. Haussmann, Department of Psychology, P.O. Box 11A Yale Station, New Haven, CT 06520-7447.

screen to be drawn, the beam must start in the upper left-hand corner, move horizontally to the right side, return to the left one scan line down, and repeat this process until it reaches the lower right-hand corner of the screen. After the screen is drawn in this manner, the electron beam returns to the upper left-hand corner (this process is known as the *vertical retrace*). This process is far from instantaneous.

When a computer program encounters a command to write to the video screen, the information generated by the program is sent to the video display adapter, which then stores the image in random access memory (video RAM). The display adapter continually reads the video RAM, while displaying its contents on the video monitor with the electron beam.

Video monitors are rated in terms of how fast they can present an entire screen. This measure is given in hertz, or cycles per second. Among displays today, measures from 60 to 70 Hz are common, although the actual range is 40–90 Hz (Woram, 1990, p. 452); 60–70 Hz means that the screen is redrawn 60 to 70 times each second. This translates into a screen refresh rate of once every 14.28 to 16.67 msec. (From this point on, a scan rate of 60 Hz is assumed, because this is most common.) The implication of this speed limitation is that with the use of a standard video display, stimuli can be presented only for durations that are multiples of the refresh rate, since it takes the entire 16.67 msec to draw the screen; no shorter display is possible. Even if the command to erase what was drawn was given immediately after presentation, it would take 16.67 msec for the electron beam to return to the stimulus location to begin erasing (the issue of phosphor decay also comes into play in examining how to “erase” stimuli, but for the sake of simplicity, this will be addressed later).

The position of the electron beam poses the second major problem. Software has no access to the current location of the beam during processing. The potential problem here is the following: if the position of the beam is essentially random, the program has no way of calculating how much time will pass between the time at which the display command is given and the time at which the image actually appears on the monitor. Suppose that the commands to present a stimulus (an “X” in the upper left-hand corner), to delay for 20.00 msec, and then to erase the stimulus are given at time t . Suppose further, that t_1 , t_2 , and t_3 represent the times of the three closest vertical retrace pulses. Let C be the constant time that it takes the monitor to scan enough lines to present the entire stimulus (this is generally very small; for a one-line stimulus, it is generally less than 1 msec). There are three possible outcomes to such a procedure.

First, assume that $(t_1 + C) < t < t_2 - 4$ msec, and that the system places the “X” into video memory at time t . Having missed the retrace at t_1 , the “X” will not appear on the video display until t_2 . By t_3 , the erase command will be in effect. Thus the stimulus is physically on the screen for one refresh cycle, or 16.67 msec.

However, if the command is given when $t = t_1$, the stimulus will be present on the screen beginning with the refresh cycle at t_1 . The 20-msec delay will last until past the point of t_2 , so the erase command will not be executed until the electron beam is beyond the stimulus, between t_2 and t_3 . In this case, the stimulus is actually on the screen for two refresh cycles, or 33.34 msec.

Finally, in the worst case scenario, the command to draw the “X” is given such that $t_1 < t < t_1 + C$. In this case, the bottom portion of the stimulus appears on the screen, and the entire stimulus is redrawn at t_2 . So the subject could end up viewing the bottom half of the “X” for 16.67 msec, followed by the entire “X” for another 16.67 msec, before the erase command takes effect at t_3 .

Thus, without a method of synchronization, large random errors (up to 33.34 msec or more, depending on the monitor frequency) will permeate any findings. A lack of synchronization also allows stimuli to be presented in pieces, further disrupting the intended effect.

Reed (1979) was among the first to point out that stimulus displays needed to be synchronized with the vertical retrace pulse. As a solution, he proposed a hardware modification for the Apple II microcomputer to allow software access to the retrace signal. Lincoln and Lane (1980) stressed the importance of three possible sources of error in using a video display: phosphor persistence, unsynchronized displays, and the time needed to create the stimulus.

The ideal solution to the problem of stimulus synchronization would be to bring the electron beam under software control. Although such a solution would require extensive hardware modifications, IBM-compatible video display adapters do support a *video refresh signal*. Through a software readable port, a program can determine when the electron beam has finished drawing the current screen and is traveling from the lower right to upper left corner (vertical retrace). Thus, it is relatively simple to write a procedure that waits to present the stimulus until the appropriate bit is set (indicating that a vertical retrace is pending). The port where this signal is present varies with the type of display adapter.

The BIOS memory location 40:63h holds the base address of the 6845 video chip. The base address is different, depending on whether the video mode is monochrome (3B4h) or color (3D4h). Regardless of mode, however, the display status port is offset 6 bytes from the base 6845 address (Sutty & Blair, 1990). For CGA, EGA, and VGA monitors, bit 3 of this display status port is high when a vertical retrace is occurring. Hercules cards, on the other hand, reset bit 7 during a retrace.

For cases in which the stimuli may be more complex than a few letters, it is a good practice to disable video output while the stimulus is being drawn. By doing this, the program can take as much time as it needs to create the stimulus. The stimulus can be placed in video RAM, but since the display is disabled, nothing appears on the monitor. Then, when the next vertical retrace occurs, the display is enabled. For CGA and Hercules monitors, the video can be disabled by resetting bit 3 of the appropri-

ate port (this port is offset 4 bytes from the base of the 6845 chip). Reenabling is simply a matter of setting this bit. In EGA monitors, the palette colors are changed to write the entire screen in black (a value of 10h written to port 3C0h will disable the video, while a value of 20h will reenable the video). VGA monitors are able to use a ROM BIOS call to disable or enable output (interrupt 10h, subfunction 36h is called, with AL=1 to disable and AL=0 to enable video). The following pseudocode shows the entire procedure:

```
DisableVideo;           { Turn off the display }
CreateStimulus;         { Write stimulus into RAM }
While not(retrace) do;   { Wait for a retrace }
EnableVideo;            { Enable the screen }
```

An alternate method is available for certain modes in the Hercules, EGA, and VGA systems. In these modes, various video "pages" are available, with any one being active (visible) at a given time. With such a configuration, a stimulus may be written into an inactive page (not interfering with the current display). The computer can then be instructed to switch between pages when a refresh pulse occurs. The page swap is done with a call to interrupt 10h, subfunction 5h. At the time of the call, register AL contains the page number to activate.

Many authors have published code to both present stimuli synchronized with the IBM vertical retrace and switch pages of video memory: Segalowitz (1987) provided a BASIC program (later rewritten in assembly language in 1988) that runs on a CGA system. Heathcote (1988) wrote a Turbo Pascal routine that handles CGA, Hercules, and EGA adapters, but is limited because the experimenter must first specify which type of display adapter is being used. Finley's (1989) routines provide synchronized displays and page swapping, but only for the Hercules card. Routines for page swapping in CGA, EGA, and VGA were combined in Dlhopsky's (1989) code, but here the Hercules card was not supported. More recently, Paredes, Miller, and Creeger (1990) concentrated on EGA and VGA monitors and their ability to utilize nonactive video pages to create more complex stimuli.

The procedures in this paper synthesize and expand all of these solutions: all types of monitors are supported (monochrome, CGA, Hercules, EGA, and VGA), in all possible video modes. Both synchronized display and page swap routines are also included, all within a single Turbo Pascal unit. The experimenter does not need to know ahead of time what type of system the code will be running on, because once the unit is initialized, it automatically detects and calibrates the video system.

Timing

With any type of IBM-compatible machine, another concern arises when tachistoscopic and reaction time programs are used. In its normal mode, the IBM system clock is only accurate to 55 msec, regardless of the speed of the processor. How, then, can one obtain millisecond timing accuracy?

The problem of achieving timing with millisecond accuracy from microcomputers has been around almost as long as the microcomputers themselves. Initial suggestions included employing an external timer connected to the microcomputer bus (Goldband, 1979; Segalowitz, 1987). Other routines have employed the serial port to provide constant timing frequencies (Emerson, 1988a).

In the IBM-compatible architecture, the Intel 8253 16-bit timer/counter (or upwardly compatible Intel 8254) is used, maintaining three independent counters, each with a fixed output rate of 1.1931817 MHz. Timer channel 0 is used to increment the system time-of-day (TOD) clock. In its unmodified state, an IBM-compatible machine generates interrupt 8h to update the TOD clock every time this counter counts down from its maximum value of FFFFh. After updating the clock, interrupt 1Ch is initiated, which usually contains a simple IRET (return from interrupt) command. At a frequency of 1.1931817 MHz, this occurs once every 54.925 msec. Thus, if a program were to use the TOD clock as its only indicator, each reading could be in error up to 55 msec.

The 8253/4 chip can operate in one of six different modes. The default, mode 3, generates square waves. To do so, it starts with a high output signal, decreases the residual counter value by decrements of two until reaching zero, lowers the output signal, reloads the counter, and begins the countdown again. In this mode, the residual value is not of much use to the experimenter, because the value returned could be that of the first or second phase of the countdown, with no way of distinguishing which one.

The other mode of interest to experimenters is mode 2, in which the chip operates as a rate generator. Here the chip repeatedly decreases the counter by decrements of one, and generates an interrupt when it reaches zero. In this mode, the residual value indicates how far into the countdown the chip actually is.

Graves and Bradley (1987) published one of the first timer software routines to give precision better than the TOD clock. Their paradigm, based on that of Smith and Puckett (1984), used timer mode 2 of the 8253 chip, but because it was written in a higher level language (BASIC), intertrial intervals were limited to a minimum precision of 35 msec. Graves and Bradley later (1988) modified their routine to increase the resolution to approximately 2 msec by using long integers rather than double precision variables.

Bührer, Sparrer, and Weitkunat (1987) proposed assembly language routines that changed the value from which the 8253 counter (mode 3) counted down in order to update the TOD clock. This caused the system to issue interrupt 8h (and subsequently 1Ch) every millisecond. The authors took advantage of this, using interrupt 1Ch to update a global timer variable. In doing so, they were able to obtain readings from a variable that was updated once every millisecond. This also created a negative side effect, however: the system clock ran 55 times its normal speed, possibly causing problems with other software. Both Heathcote (1988) and Brysbaert, Bovens, d'Ydewalle, and

Van Calster (1989) rewrote this code into Turbo Pascal; Heathcote also included a routine to reset the system time of day clock back to actual time. Dlhopsky (1988) made the same modifications in the C language. Creeger, Miller, and Paredes (1990) suggested emulating the standard system timer interrupt, so that the time of day clock would not run at a higher rate. Although this method accomplished what it set out to do, other software may not run properly because interrupt 8h is still being generated at 55 times its normal rate.

Emerson (1988b) used timer mode 2, but with commands available only on the 8524 chip. Such a routine was limited because it did not function correctly on unmodified PC or XT machines (as both use the 8523). Granaas (1989) modified this routine to operate in a wider range of C programming environments, but did not address the limitation that prevented the code from running on machines with an 8523 chip.

One of the first programs that achieved better than millisecond resolution with the use of the residual value of the 8523 in timer mode 2 was a Turbo Pascal program written by Crosbie (1989). Gabrielson and Jarvella (1990) later presented a Turbo C version. The Crosbie algorithm took over interrupt 1Ch and provided a count of the number of increments in the system time of day counter; Gabrielson and Jarvella read this value directly from the BIOS. To embellish the 55-msec accuracy of this clock, the residual timer count was used. Unfortunately, as Bovens and Brysbaert (1990) pointed out, it was possible that the system time of day counter could add an increment between reading the residual counter and TOD clock, thereby producing an error of 55 msec on certain trials.

In a review of timer routines, Graves and Bradley (1991) concluded that the Smith and Puckett (1984) algorithm, with a correction provided by Bovens and Brysbaert (1990), is the most accurate method. In this paradigm, the 8523 timer is initialized to mode 2, so that the residual will count down from a specified value (65,535 in this case) to 0 and then generate an interrupt to update the TOD clock. This counter value may be latched—that is, frozen for a later reading, without affecting the counting process. The residual value is then accessed by reading a port. Since the 8523 chip counts at a frequency of 1.193187 MHz, it can be calculated that the residual is decremented once every .000838095 msec (1/1.193187 MHz). Updates to the TOD clock (“ticks”) are made every time the residual counts down to 0, or 54.9254 msec. By taking the 10 most significant bits of the residual (after negating the count-down value), and combining them with the 22 least significant bits of the TOD “tick” count, we produce a signed long integer that gives us a resolution of .0536381 msec and can hold 2^{31} such “tocks” (a unit of time equal to 53.6381 μ sec; Emerson, 1988a), or slightly under 32 h. Since the counter value is reset to 0 at midnight anyway, this solution is ideal.

Once such a routine is installed, we have access to a global function timer that returns a long integer (longint) incremented once every .0536377 msec. For the sake of simplicity, a global constant, *tocks*, is defined to be the

conversion factor between a *timer* count and milliseconds ($\text{tocks} = 1/.053639 = 18.643469$). The ease with which this method can measure to within millisecond accuracy is shown in the following routine:

```

Display(Stimulus);      { Synchronized display stimulus }
t1:=timer;              { Get initial timer value }
while not(keypressed) do; { Do nothing until key press }
t2:=timer;              { Get timer after key pressed }
ch:=readkey;            { Clear key pressed }
Elapsed:=(t2-t1)/tocks; { Calculate milliseconds elapsed }

```

The timer is read immediately following the display of stimulus material; when a key is pressed, the timer is read again. The difference in the two values holds the elapsed tocks between the two events. Conversion to milliseconds is a matter of simple division.

As Graves and Bradley (1987) and Brysbaert (1990) pointed out, the use of Turbo Pascal’s *readkey* function alone can cause timing errors of 18.4–36.7 msec. When the *keypressed* Boolean value is employed to indicate a keyboard input, however, the accuracy improves, although it does not reach millisecond precision. Segalowitz and Graves (1990) have calculated that when a keyboard is the input device, reaction time measurements will be overstated by a mean of 18.5 msec and will be subject to an error of ± 7.5 msec. Although for most applications such an error term is negligible (as the 18.5-msec constant can be subtracted from all measurements), it is important to consider.

The TSCOPE Unit

TSCOPE is a Turbo Pascal 5.5 unit that addresses both the video and the timing concerns in a wide variety of IBM-compatible systems. CGA, EGA, Hercules, MCGA, and VGA video modes are supported on any machine that is based on an 8088, 80268, 80386, or 80486 processor, regardless of bus architecture. The unit is completely self-contained and needs only to be referenced in the program in which it is used with a *Uses Tscope* line below the program header. Such a reference provides all the necessary functions to the calling program to enable synchronized displays, video page swapping, and a millisecond accuracy timer (see Table 1).

A global variable, *Screen*, is used throughout the unit to provide information about the current hardware setup. *Screen* is actually a variant record, meaning that the number and nature of the fields contained within the record change on the basis of the value of one variable (in this case *Enable_Mode*). The fixed portion of the record contains the Boolean variable *Installed* (indicating whether or not the unit is in place and functioning properly), *Mode* (determining graphics vs. text), *MaxX* and *MaxY* (the maximum position at which stimuli may appear; in text mode, these are the number of characters, in graphics mode, the number of pixels), and *Frequency* (the vertical refresh frequency of the monitor, in hertz). Also found in *Screen* are the values *Video_Driver* and *Video_Mode* (representing the Turbo Pascal Driver and Mode values), their text equivalents (*DriverName* and *ModeName*), and the

Table 1
Summary of Global TSCOPE Procedures and Functions

Procedure	Parameters	Purpose
TScope_Unit	Screen,Command	Install, remove, test or provide information about the current setup; Command = (Install, remove, test, info)
ControlVideo	Screen,Command	Enable or disable video output; Command = (Enable, Disable)
Synchronize	Screen	Wait for vertical retrace
Display	Screen,Stimulus	Provide synchronized display
BiosWrite	Screen,Stimulus	Provide an alternative to Turbo Pascal's write for writing text mode stimuli
SetTextPage	Screen,Page	Switch to a current text page, after waiting for a vertical retrace; Page = any valid video page
CalcDuration	Stimulus	Calculates the millisecond duration from two timer counts
TScopeResult	(none)	Function returning any or all error codes
TScopeError	ResultCode	Provides English language version of error code
Timer	(none)	Function returning a longint representing current clock value
DelayMS	Milliseconds	Performs a delay for the specified number of milliseconds

information needed to recognize a video retrace: Refresh_Port, Refresh_Mask, and Refresh_Mode (indicating whether or not the specific bit is set or reset when a refresh is active). The variant portion of the record depends on the value of Enable_Mode, which specifies how the current video configuration enables and disables the screen, whether by using ports, registers, or the palette. When Enable_Mode = RegType, two sets of registers are defined: Regs[Enable] and Regs[Disable] (which, when passed to the appropriate BIOS routine, will enable or disable video output on a VGA monitor). In the case where Enable_Mode = PortType, video output is controlled by writing Enable_Byte[Enable] or Enable_Byte[Disable] to Enable_Port. Finally, if Enable_Mode = PaletteType (in the case of EGA), output is regulated by writing either Palette_Mask[Enable] or Palette_Mask[Disable] to Palette_Port.

A new type of StimulusType is defined, which the main program must make use of. All stimuli must be of this type to be passed to the Display, BiosWrite, or CalcDuration procedures. StimulusType is a record of nine properties. StimText contains the string to be presented. PageNum is the video page on which the presentation should occur, at coordinates Xpos and Ypos. Color determines the color of the stimulus. TimeOnScreen and TimeOffScreen are both of type longint and represent values of the timer counter. The former is set automatically in the Display procedure (when the vertical retrace occurs); the latter must be assigned by the main program. Finally, Duration is a real number variable that can be used to represent the actual stimulus duration in milliseconds. A call to the CalcDuration procedure will calculate this value

automatically, by converting the difference between the timer values into milliseconds.

The main workhorse of the unit is the procedure Tscope_Unit. This procedure is called by passing the global variable Screen and a command (one of Install, Remove, Test, or Info).

The Install command to the Tscope_Unit procedure sets up the initialization parameters for different types of monitors: CGA, EGA, Hercules, MCGA, and VGA. If the global variable Screen is passed with default values set, the procedure will automatically determine the best video configuration and initialize the graphics mode. The Install command also provides a minor self-test. Upon exit, all Screen fields will be set, and any error code may be returned by the TScopeResult function.

Prior to the Install, any of the following Screen variables may be specified: Mode (Text_Mode or Graph_Mode), Video_Driver (CGA, EGA, etc.), or Video_Mode (any appropriate video mode number), in which case the procedure will use the provided values. For example, if the call Tscope_Unit(Screen,Install) is given, and Screen.Video_Mode is preset to CGA, the procedure will initialize the system to CGA, in the highest graphics video mode available.

The Remove command simply resets the 8523 chip to its normal state and closes down the graphics display (if used). Screen.Installed is set to FALSE.

The TScope_Unit as a whole may be thoroughly tested (to assure system compatibility) with the Test command. Here a procedure is called that presents one "X" on each successive retrace at a random point on the screen. By

checking the elapsed time against constant range values, the system can determine whether it is operating within allowed bounds.

Finally, the Info command will produce a text screen of pertinent information, including the current TScope_Unit settings of refresh ports/registers, enable ports/registers, video modes, and screen frequency.

As mentioned previously, the TScopeResult function returns a value to indicate the current status of the unit. Any value other than 0 indicates an error state (bits 0–3 represent an error returned from Turbo Pascal, and bits beyond that each indicate a specific error). TScopeResult functions in much the same way as Turbo Pascal's IOResult: after it is called, the value is cleared to 0. The TScopeError procedure may be used to decipher an error code. For example, the call TScopeError(TScopeResult) will print (in text mode) the English-language version of any error codes and clear TScopeResult.

The procedure Display is responsible for presenting a synchronized stimulus display, operating in both text and graphics modes. First, if Stimulus.Disable is TRUE, video output is disabled. Stimulus.StimText is then written into video RAM (on page Screen.PageNum and in the color of Stimulus.Color), and the system loops until a vertical retrace is encountered. At this point, Stimulus.TimeOnScreen is given the current timer count, and the video output is reenabled if it has previously been disabled.

In the text mode, Display relies on the BiosWrite procedure. This code, written in machine language for optimum efficiency, places Stimulus.StimText with color attributes into video memory on page Stimulus.PageNum. It is better than the Turbo Pascal write, in that it can output text to any video page, and in any color, in a single statement.

Text pages of video memory may be swapped by using the SetTextPage procedure, which waits for a vertical retrace before making the switch. Turbo Pascal provides the analogous graphics mode commands with SetActivePage and SetVisualPage. These Turbo routines do not provide video synchronization, but the Synchronize procedure may be called at any time from the main program and will cause execution to pause until a retrace is pending. Similarly, the procedure ControlVideo may be used by the main program to enable or disable video output.

The last issue regarding the use of the video display is that of phosphor decay. If one examines how information is presented on the display, it becomes apparent that once the phosphor on the screen is excited by the electron beam, it remains in that state at least for the duration of one screen refresh. This is apparent because the image appears constant to an observer, although each point is only actually being drawn once every 16 msec or so. So when the computer is instructed to erase something on the display, it simply stops refreshing the appropriate phosphor. But it takes 16 msec or more before this phosphor loses its glow (in some monitors, phosphor persistence is on the order of 2 sec or more). To circumvent this problem, instead of a command's being sent to simply erase time-sensitive stimuli, a mask may be employed

as a function of Display. The Display procedure may be called to present a stimulus, wait for a specified time, and then replace this stimulus on the screen with a mask (such as a series of Xs or random letters). When this method is used, phosphor decay is no longer a concern. As an alternative, stimuli can be written in inverse video and written over with solid blocks of color to "erase" them.

The timing concerns are handled by the Timer function and the DelayMS procedure. The Install command of the TScope_Unit sets the 8523/8524 timer chip into mode 2. From that point on, we can call Timer to return a longint value that represents the number of times the timer has been incremented since the previous midnight. DelayMS makes use of the Timer function and will delay a specified number of milliseconds before returning control to the calling procedure.

Implementation

The sample Listing at the end of this article is ready to be compiled under Turbo Pascal Version 5.5. This simple program makes use of the aforementioned procedures and provides a basic implementation of a tachistoscopic display. The user is prompted for the desired prime stimulus, the length of time (in milliseconds) during which the prime is displayed, and the mask time (also in milliseconds). Due to hardware limitations, the video monitor can only display images for times that are multiples of 16.67 msec. To account for this, after the trial the actual prime duration is displayed.

Availability

The complete unit (including source code) may be obtained from the author via electronic mail. Direct requests to haurrobe@yalevm.bitnet or haussman@powerstrip.ai.cs.yale.edu. A floppy disk version is also available from the author for \$15. Please indicate disk size (5.25 or 3.5 in.) when ordering. Double-density disks will be used unless otherwise specified.

REFERENCES

- BOVENS, N., & BRYSAERT, M. (1990). IBM PC/XT/AT and PS/2 Turbo Pascal timing with extended resolution. *Behavior Research Methods, Instruments, & Computers*, 22, 332-334.
- BRYSAERT, M. (1990). A warning about millisecond timing in Turbo Pascal. *Behavior Research Methods, Instruments, & Computers*, 22, 344-345.
- BRYSAERT, M., BOVENS, N., D'YDEWALLE, G., & VAN CALSTER, J. (1989). Turbo Pascal timing routines for the IBM microcomputer family. *Behavior Research Methods, Instruments, & Computers*, 21, 73-83.
- BÜHRER, M., SPARRER, B., & WEITKUNAT, R. (1987). Interval timing routines for the IBM PC/XT/AT microcomputer family. *Behavior Research Methods, Instruments, & Computers*, 19, 327-334.
- CREEGER, C. P., MILLER, K. F., & PAREDES, D. R. (1990). Micro-managing time: Measuring and controlling timing errors in computer-controlled experiments. *Behavior Research Methods, Instruments, & Computers*, 22, 34-39.
- CROSBIE, J. (1989). A simple Turbo Pascal 4.0 program for millisecond timing on the IBM PC/XT/AT. *Behavior Research Methods, Instruments, & Computers*, 21, 408-413.
- DLHOPOLSKY, J. G. (1988). C language functions for millisecond tim-

- ing on the IBM PC. *Behavior Research Methods, Instruments, & Computers*, **20**, 560-565.
- DLHOPOLSKY, J. G. (1989). Synchronizing stimulus displays with millisecond timer software for the IBM PC. *Behavior Research Methods, Instruments, & Computers*, **21**, 441-446.
- EMERSON, P. L. (1988a). TIMEX: A simple IBM AT C language timer with extended resolution. *Behavior Research Methods, Instruments, & Computers*, **20**, 566-572.
- EMERSON, P. L. (1988b). Using serial interfaces and the C language for real-time experiments. *Behavior Research Methods, Instruments, & Computers*, **20**, 330-336.
- FINLEY, G. P. (1989). Tachistoscopic software for the Hercules display controller. *Behavior Research Methods, Instruments, & Computers*, **21**, 387-390.
- GABRIELSSON, J., & JARVELLA, R. J. (1990). How to obtain near-millisecond precision on the IBM PC with visual stimuli not exceeding one printed line. *Behavior Research Methods, Instruments, & Computers*, **22**, 520-525.
- GOLDBAND, S. (1979). Real-time clocks for microcomputers in behavioral research. *Behavior Research Methods & Instrumentation*, **11**, 111-114.
- GRANAAS, M. M. (1989). TIMEX2: A modified C-language timer for PC AT class machines. *Behavior Research Methods, Instruments, & Computers*, **21**, 619-622.
- GRAVES, R., & BRADLEY, R. (1987). Millisecond interval timer and auditory reaction time programs for the IBM PC. *Behavior Research Methods, Instruments, & Computers*, **19**, 30-35.
- GRAVES, R., & BRADLEY, R. (1988). More on millisecond timing and tachistoscope applications for the IBM PC. *Behavior Research Methods, Instruments, & Computers*, **20**, 408-412.
- GRAVES, R. E., & BRADLEY, R. (1991). Millisecond timing on the IBM PC/XT/AT and PS/2: A review of the options and corrections for the Graves and Bradley algorithm. *Behavior Research Methods, Instruments, & Computers*, **23**, 377-379.
- HEATHCOTE, A. (1988). Screen control and timing routines for the IBM microcomputer family using a high-level language. *Behavior Research Methods, Instruments, & Computers*, **20**, 289-297.
- LINCOLN, C. E., & LANE, D. M. (1980). Reaction time measurement errors resulting from the use of CRT displays. *Behavior Research Methods & Instrumentation*, **12**, 55-57.
- PAREDES, D. R., MILLER, K. F., & CREGER, C. (1990). Graphic precision: Controlling stimulus displays on IBM PC-compatible computers. *Behavior Research Methods, Instruments, & Computers*, **22**, 319-322.
- REED, A. V. (1979). Microcomputer display timing: Problems and solutions. *Behavior Research Methods & Instrumentation*, **11**, 572-576.
- SEGALOWITZ, S. J. (1987). IBM PC tachistoscope: Text stimuli. *Behavior Research Methods, Instruments, & Computers*, **19**, 383-388.
- SEGALOWITZ, S. J. (1988). IBM PC tachistoscope: II. Assembly language subroutines. *Behavior Research Methods, Instruments, & Computers*, **20**, 432.
- SEGALOWITZ, S. J., & GRAVES, R. E. (1990). Suitability of the IBM XT, AT, and PS/2 keyboard, mouse, and game port as response devices in reaction time paradigms. *Behavior Research Methods, Instruments, & Computers*, **22**, 283-289.
- SMITH, B., & PUCKETT, T. (1984, April). Life in the fast lane. *PC Tech Journal*, pp. 63-74.
- SUTTY, G., & BLAIR, S. (1990). *Advanced programmer's guide to Super-VGAs*. New York: Simon & Schuster.
- WORAM, J. (1990). *PC configuration handbook*. New York: Bantam.

LISTING

Sample Experimental Program

```

Program Demo (input,output);
Uses
  Crt, TScope;

Const
  MaskString='XXXXXXXXXX';
  MaskDuration=500;

Var
  Prime,Mask:StimulusType;
  PTime,MTime:integer;
  ErrorCode:Word;

Procedure Pause;
Var
  ch:char;
Begin
  writeln;
  writeln;
  writeln('Press any key to continue...');
  while not keypressed do;
    ch:=readkey
  end;

Procedure GetValues(Var Prime,Mask:StimulusType;
  Var PTime,MTime:integer);
Begin
  clrscr;
  Prime.PageNum:=0;
  Prime.Disable:=True;
  Write ('Enter stimulus to use: ');
  Readln(Prime.StimText);
  Write ('Enter x-coordinate (max. ',Screen.MaxX:2,'):');
  readln(Prime.XPos);
  write ('Enter y-coordinate (max. ',Screen.MaxY:2,'):');
  Readln(Prime.YPos);
  Prime.Color:=White;

```

LISTING (Continued)

```

    write ('Enter stimulus duration: ');
    readln (PTime);
    Mask:=Prime;
    Mask.Disable:=False;
    Mask.StimText:=Copy(MaskString,1,length(Prime.StimText));
    MTime:=MaskDuration
end;

Procedure DoTrial(Screen:ScreenType;Var Prime,Mask:StimulusType;
                  PTime,MTime:word);
Begin
    clrscr;

    Display(Screen,Prime);
    DelayMs(PTime);
    Display(Screen,Mask);
    DelayMs(MTime);
    Prime.TimeOffScreen:=Mask.TimeOnScreen;
    CalcDuration(Prime)
end;

Procedure Results (Stimulus:StimulusType;Code:Word);
Begin
    clrscr;
    writeln;
    write ('Actual prime duration: ',Stimulus.Duration:5:2);
    writeln (' milliseconds. ');
    writeln;
    writeln;
    TScopeError(Code);
    writeln;
    writeln;
    Pause
end;

Begin
    Screen.Mode:=Text_Mode;           { Use text mode      }
    TScope_Unit(Screen,Install);       { Install the unit   }
    ErrorCode:=TScopeResult;           { Check for error    }
    If ErrorCode<>0 then                 { If error, print text }
        TScopeError(ErrorCode)         { version            }
    else
        Begin
            GetValues(Prime,Mask,PTime,MTime);
            Pause;
            DoTrial(Screen,Prime,Mask,PTime,MTime); { If not,           }
            ErrorCode:=TScopeResult;           { do a trial       }
            Results(Prime,ErrorCode)
        end;
    TScope_Unit(Screen,Info);           { Display unit info  }
    TScope_Unit(Screen,Remove);         { Remove unit        }
    writeln;
    writeln;
    TScopeError(TScopeResult)          { Check for remaining error }
end.

```