Behavior Research Methods, Instruments, & Computers
1989, 21 (4), 441-446

COMPUTER TECHNOLOGY

Synchronizing stimulus displays with millisecond timer software for the IBM PC

JOSEPH G. DLHOPOLSKY
Radiation Dynamics, Inc., Melville, New York

A C language technique for synchronizing millisecond timer software to the appearance of the stimulus on the IBM PC's video monitor is described. Tachistoscopic programs that use the technique can correct the mean 8.3-msec bias normally found in reported response latencies and reduce the associated error variance.

Researchers considering the use of the IBM PC for tachistoscopic applications should know that the appearance of a stimulus on the video monitor is controlled by two asynchronous processes: the computer program and the video display circuitry. The program executes instructions that result in the storage of character data (or pixel images) in random access memory (RAM) that is dedicated to the video display. The display processing hardware recurrently reads video RAM and controls the intensity of the monitor's electron beam as it "draws" images. The result: the computer program—which also initiates the timing of response latencies—normally does not know the precise moment at which the display process causes the stimulus to appear on the screen.

The decoupling of the program and video processes has a number of effects of interest to researchers who require precise control over brief stimulus displays and millisecond interval timing. As the vertical dimension of the stimulus increases, it becomes more likely that the stimulus will be drawn on the screen in parts. For example, the electron beam might be aimed at the middle of the screen when the program transfers the stimulus data to video RAM. The bottom portion of the stimulus would then appear first, followed by an interval during which the electron beam would be scanning the bottom and then the top of the screen (both areas presumably empty of stimulus parts). Finally, the top portion of the stimulus would appear. The interval between the appearance of first- and last-drawn portions would be 16.7 msec (assuming a monitor with a 60-Hz refresh frequency).

The situation for response latencies produces both biased results and increased error variance. If the program starts a millisecond timer following the execution of the commands to display the stimulus, the timer will have been running an average of 8.3 msec before the stimulus appears on the screen. Thus the measured response latencies will be too large by an average of 8 msec. This is easy enough to correct by simply subtracting 8 msec from all the values. However, some of the latencies will approach an error of 16.7 msec, whereas others will have very small errors. Researchers who expect small independent variable effects would want to eliminate or at least reduce this source of error variance.

The decoupling of the program and display processes was recognized at an early point in the use of microcomputers for tachistoscopic laboratory applications (Dlhopolsky, 1982; Grice, 1981; Lincoln & Lane, 1980; Merikle, Cheesman, & Bray, 1982; Reed, 1979). Past solutions involved either hardware or software techniques in the Apple II and Commodore PET, and Models I and III of the TRS-80. However, similar techniques are not well known for the IBM PC. This article describes such a technique.

Method

I/O addresses 3D0h through 3DFh in the IBM PC, PC AT, and many compatibles are mapped to the color graphics adapter or CGA (3B0h-3BFh for the monochrome display adapter or MDA) (IBM, 1984; Tandy, 1984). The on/off state of bit 3 of address 3DAh reflects the state of the vertical retrace signal produced by the CGA video circuitry. This signal provides the ability to synchronize stimulus displays with timer software.

To grasp the usefulness of the vertical retrace signal, one must understand the operation of the CRT's electron beam as it "draws" an image. The beam is a stream of electrons that energizes phosphors on the inner surface of the monitor screen. Its "starting" position is at the top of the screen, from which it successively scans each of the 200 horizontal lines of the standard CGA monitor (more for MDA, EGA, and VGA). When it gets to the bottom, there is a pause during which it is turned off and reaimed back at the top. This pause is called the *vertical*

Correspondence may be addressed to Joseph G. Dlhopolsky, 27 Wilson Street, Port Jefferson, NY 11776.

retrace interval. The whole process is done 60 times a second on CGA monitors (50 Hz on TTL monitors and in Europe and in other countries; 70 Hz on some high-resolution monitors, e.g., EGA and VGA).

Bit 3 of port 3DAh is on during the retrace interval (it reads as 1). This state can be detected by having the program repeatedly input a byte from this port and perform a logical AND operation between it and 8 (2 raised to the third power). A TRUE result indicates that the electron beam is currently engaged in a retrace interval. A FALSE result indicates that the beam is energizing the screen phosphors.

A program that synchronizes timer onset with the synch bit of 3DAh should loop until it detects the onset of the retrace interval. At this point, it should execute the commands to store the stimulus data in video RAM. Assuming that the researcher has established the time it takes for the beam to get to the stimulus location, the millisecond timer could be started at the proper moment. Alternatively, the program could start the timer at the vertical retrace signal and then use it to pace stimulus intervals and response latencies from the first signal. This technique is used in the C language demonstration program listed in the Appendix and described below.

The program was written for the Turbo C 1.5 Compiler. Although other C compilers have similar capabilities, this one combines attributes of cost and versatility.

The program demonstrates the detection of the vertical retrace signal and its use in providing synchronized stimulus displays. It may also be used to determine the location of the electron beam at various intervals following the onset of the signal. This information is useful for determining the proper time to start a latency timer, or for making postexperiment corrections in latency data. For example, if the researcher establishes that the stimulus location is reached 5 msec after the retrace signal, he or she can adjust the measured response latencies by 5 msec.

The program's operation is straightforward. It requests the user to enter a delay value, the most informative values being between 0 and 16 msec. The computer then draws a scale that indicates the location of the 25 text lines. It then waits for a vertical retrace signal, upon which it pauses for the time entered by the user. At the end of this interval, it sends the values for a light blue vertical bar to video RAM. It pauses for the remainder of one screen scan (as 17 msec) and then sends blanks to the vertical bar's video RAM. It repeats this loop 100 times. The subjective effect is a rapidly flashing blue bar that has a relatively consistent top position and a consistent termination at the screen bottom.

Program Description

The lines in the program have been numbered to aid in the description. Lines 001 and 002 identify two header files that are included in the source module. The stdio.h file is universally included in C programs. The conio.h file is an ANSI Standard C header file that contains information specific to the handling of screen output.

Three macros are defined in lines 004-009. Their use makes later portions of the code more readable. The compiler substitutes the text in these macros for the macro name. For example, whenever the macro GET_MSEC appears in the source code, the compiler will replace it with the following text: (unsigned)peek(0, 0x046c).

The GET_MSEC macro is used to read the millisecond timer. The ZERO_TIMER macro sets the timer to 0. These macros, together with the two functions, set_timer( ) and reset_timer( ) (lines 012-026), have been described elsewhere (Dlhopolsky, 1988). They constitute the millisecond timer.

The VID_SYNC_OFF macro is used to test the state of the vertical retrace signal. Its construction, which is somewhat indirect, will become obvious when described below.

Consistent with C Language practice, the entry function in the program—main( )—appears at the end of the listing (lines 122-140). The first few lines of main( ) declare the functions called by main( ). Line 128 is the beginning of a *do* loop that continues until the user enters a negative delay value. In executing this loop, the computer requests a delay value, which should be in the range 0-16 msec. It reads this value from the keyboard as a string and converts it to an integer (line 132).

If the user enters a positive number, main( ) calls three functions that successively clear the screen, display a scale in the center of the screen, and flash the vertical bar. The clrscr( ) function (line 135) is the Turbo function that clears the screen. Being in the Turbo library, it does not appear in the listing. The other two functions are listed.

The draw_scale( ) function (lines 108-119) carries out a straightforward display of a scale that identifies the location of the 25 screen text lines. A printer output of this

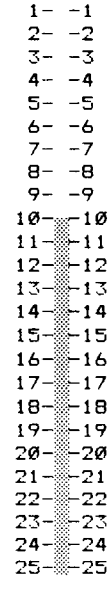

Figure 1. Sample time-delayed CRT electron beam locator scale. The numbers represent text lines. The central bar depicts the appearance of the screen with a delay of 7 msec on a Tandy 1000.

scale appears in Figure 1. Readers who are unfamiliar with the C language should refer to a textbook for a description of the printf( ) function in line 116.

The draw_vert_bar( ) function (lines 030-105) carries out the flashing of the vertical bar. The main( ) function passes to it the integer delay value that the user entered. This value is declared in the first line of the function in a format consistent with function prototyping provision of the forthcoming ANSI C Standard.

The first few lines of draw_vert_bar( ) contain declaration and initialization of variables. The integer variable, bar_chr, is the attribute and IBM PC character set code for a short vertical light blue bar. This is the building block of the full screen bar. The integer array, vidram, consists of the segment offsets of the video RAM addresses for the area of the screen on which the vertical bar will appear. The segment base for the IBM PC's video RAM is B800h for CGA graphics.

Line 041 starts the loop that carries out 100 displays and erasures of the vertical bar. The call to set_timer( ) in line 043 programs the IBM hardware timer to time in 1 msec increments, instead of the normal 55 msec. The computer then waits for a vertical retrace to occur, as programmed in line 044. As this line reads, the computer loops as long as the vertical retrace signal is off. The semicolon at the end of the *while* statement indicates that the loop is self-contained. When the vertical retrace signal goes on, the computer breaks out of the loop and executes the succeeding instructions.

The operation of the while loop in line 044 explains the somewhat cryptic definition of the VID_SYNC_OFF macro: The bitwise AND operation (&) between input from I/O address 3DAh and 8 returns FALSE during active screen scanning (bit 3 of port 3DAh is 0 and bit 3 of the digit 8 is 1). The logical negation operator (!) changes this to TRUE. Used in a *while* statement, the loop will continue as long as active screen scanning continues. As soon as the vertical retrace occurs, bit 3 becomes 1, the bitwise AND then returns TRUE, and the logical negation changes this to FALSE. Accordingly, the loop in line 044 breaks upon detecting this FALSE condition.

Line 045 sets the millisecond timer to 0. Line 046 causes the computer to pause for the chosen delay interval. The *while* loop will execute as long as the timer reading is less than the value of *delay*.

The 25 poke( ) function calls in lines 048-072 progressively store the vertical-bar-segment character code in the video RAM locations for inside the scale. The sequence is from the screen top to bottom in order to stay ahead of the advancing electron beam.

The program uses the sequence of pokes rather than a single poke within a loop, because a loop entails a count variable that must be incremented or decremented for each iteration. This adds processing overhead to the program's transfer of data to video RAM. Remember, this transfer is literally racing the electron beam, so compactness of code has been sacrificed to speed of operation. It is quite possible that a loop would have been fast enough, but the poke method used here emphasizes the need for speed. (See Usage Notes for more information on the poke method.)

There are several other alternatives to transferring the stimulus data to video RAM. Some compilers may be better than others at efficient use of the CPU registers, and the test program could be written to capitalize on this. The fastest approach, of course, would be to substitute in-line assembly code that would make maximal use of the registers.

Line 73 causes the computer to wait for 17 msec since the start of the vertical retrace interval. Then 25 poke( ) calls (lines 075-099) send blanks (' ') to the video RAM addresses for the vertical bar. The computer then pauses until 55 msec have passed since the first vertical retrace was detected. This takes it past the signal that would have been detected at 50 msec. Then the cycle is repeated another 99 times. The net effect is that a portion of the vertical bar will appear each time for 17 msec (actually 16.7 msec), followed by a 50-msec blank interval.

Results

The program was tested on a Tandy 1000, an IBM PC XT, and an IBM PC AT. The data for the Tandy 1000 are in Table 1. There was some scattering of the position of the top of the vertical bar. The scattering covered about 1.5 text lines—equivalent to about 1-msec variation—but it was easy to identify a line for which the bar was always solid for a particular delay value. Moreover, the 1-msec variation compares favorably with the 16.7-msec variation that would occur without the synchronization technique.

The scattering could be due to the fact that the routine executes every fourth refresh cycle, that is, every 66.667 msec. The first few vertical synch signals would arrive at 0, 66.667, 133.333, 200.000, 266.667, 333.333, ... msec from the first signal that was detected. Because the timer counts in milliseconds, it will not match the occurrence of the vertical synch signal precisely. Rather,

Table 1
Observed Location of Top of the Solid Portion of Vertical Bar After Various Delays from the Vertical Retrace Signal

Delay (msec)	Highest Solid Line
0	1
1	1
2	1
3	3
4	4
5	6
6	8
7	10
8	12
9	13
10	15
11	17
12	19
13	21
14	23
15	24
16	none

Note—These data were recorded from a Tandy 1000.

it will match the signal on one third of the cycles, but be off by .333 and .667 msec on the other two thirds. The net effect would be a varying visible starting point for the vertical bar with a variation on the order of a millisecond, which was found. This effect is amplified in the test program with its rapidly repeated 100 cycles. However, it could be circumvented with software implemented to run tachistoscopic experiments.

It is interesting to compare the synchronization technique with the test program running without it. To do this, the user could recompile and link the source code without the synchronization section (line 044). Alternatively, the program could be coded with a query to the user to set a flag that would determine whether or not to use the synchronization code. Either way, the results are dramatic: The nonsynchronized code chaotically draws the bar in random locations on the screen in contrast to the orderly synchronized process. One has only to imagine stimuli being presented in the two ways to ascertain the more appropriate method to use in research-grade software.

Usage Notes

The video synchronization technique described here will perform as described as long as the computer has no other active software that also reprograms the timer hardware. Such a program could be present without the user's knowing about it, if the computer loaded it as a memory-resident program during the execution of the AUTOEXEC.BAT file during boot-up. If uncertain about a line in the AUTOEXEC.BAT file, try comparing the performance of the test program with the current AUTOEXEC.BAT file and with an AUTOEXEC.BAT file without the questionable line (you must reboot the computer each time). If the questionable program degrades the performance of the test program, it should be removed from any computers that run tachistoscopic programs, even if the techniques in this article are not used.

Some peripherals also issue interrupts to the CPU. Among these are the keyboard and mouse interfaces. Although these devices only issue interrupts when they are used, and these interrupts will not interfere with the millisecond timer, operating them during the transfer of stimulus data to video memory could extend the transfer time beyond 16.7 msec. Of course, a computer that is capable of multitasking should not have other tasks ongoing.

The method that the program uses to display the vertical bar directly stores values in RAM, which is an approach that bypasses the IBM Basic Input/Output System resources (BIOS). The standard BIOS function calls and interrupt calls are, unfortunately, too slow—a previous version of the program, using BIOS calls, gave spurious results. The poke method, however, relies on video RAM located starting at B800h. Some PC-compatible computers do not use the same memory locations. The IBM EGA, VGA, and monochromatic display adapters also use different locations. The technique will work with this hardware if given the correct video RAM origin, which can be found in the technical manual for the adapter or computer.

The mapping of screen positions to millisecond latencies will be the same when the display is switched to a graphics mode of the same resolution in pixels. However, because there are more bytes involved, the problem of ensuring that a display is completed within a single beam scan becomes greater.

The direct high-speed pokes to video RAM will, with some adaptors, result in "snow" appearing on the screen. This by-product of the test program does not invalidate the results when applied to displays written through the BIOS calls.

While the technique described in this article will ensure that stimuli will be displayed synchronously with the video display's electron beam, researchers should ascertain that the stimulus data can be stored in video RAM quickly enough to stay ahead of the electron beam. The 16.7-msec time limit places constraints on the size of the stimulus: Computers with faster system clocks and faster RAM chips will be able to display larger stimuli. The storage time can be determined by embedding the stimulus display commands in a loop and then checking start and stop times with the millisecond timer functions used in this article.

Program Availability

A floppy disk containing the program and source code is available from the author for $10.

REFERENCES

DLHOPOLSKY, J. G. (1982). Software synchronizing of video displays and Z-80 processing in the Model III TRS-80. *Behavior Research Methods & Instrumentation*, **14**, 539-544.

DLHOPOLSKY, J. G. (1988). C language functions for millisecond timing on the IBM PC. *Behavior Research Methods, Instruments, & Computers*, **20**, 560-565.

GRICE, G. R. (1981). Accurate reaction time research with the TRS-80 microcomputer. *Behavior Research Methods & Instrumentation*, **13**, 674-676.

IBM CORP. (1984). *Technical Reference: Personal Computer AT* (Part 1502243). Boca Raton, FL: Author.

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.

MERIKLE, P. M., CHEESMAN, J., & BRAY, J. (1982). PET Flasher: A machine language subroutine for timing visual displays and response latencies. *Behavior Research Methods & Instrumentation*, **14**, 26-28.

REED, A. V. (1979). Microcomputer display timing: Problems and solutions. *Behavior Research Methods & Instrumentation*, **11**, 572-576.

TANDY CORP. (1984). *Tandy 1000 Technical Reference Manual*. Fort Worth, TX: Author.

APPENDIX

```
#include <stdio.h>                                                    001
#include <conio.h>                                                    002
                                                                      003
#define GET_MSEC       (unsigned)peek(0, 0x046c)                      004
#define VID_SYNC_OFF   !(inportb(0x3da) & 8)                          005
                      /* VID_SYNC_OFF returns 0 during retrace, which 006
                         is the signal to start transferring stimulus 007
                         data to video RAM */                         008
#define ZERO_TIMER     poke(0, 0x046c, 0)                             009
                                                                      010
                                                                      011
void set_timer()     /* sets timer 0 to time in milliseconds   */     012
{                                                                     013
   outportb (0x43,0x36);                                              014
   outportb (0x40,0xa9);                                              015
   outportb (0x40,0x04);                                              016
   return;                                                            017
}                                                                     018
                                                                      019
                                                                      020
void reset_timer()     /* Puts timer back to normal    */             021
{                                                                     022
   outportb (0x43,0x36);                                              023
   outportb (0x40,0);                                                 024
   outportb (0x40,0);                                                 025
   return;                                                            026
}                                                                     027
                                                                      028
                                                                      029
void draw_vert_bar(int delay)                                         030
{                                                                     031
    int i;                                                            032
    static int bar_chr = 0x0bdb; /* blue bar character */             033
      /* Video ram offset for vertical bar      */                    034
    static int vidram[25] = { 0x4e, 0xee, 0x18e, 0x22e, 0x2ce,        035
                              0x36e, 0x40e, 0x4ae, 0x54e, 0x5ee,      036
                              0x68e, 0x72e, 0x7ce, 0x86e, 0x90e,      037
                              0x9ae, 0xa4e, 0xaee, 0xb8e, 0xc2e,      038
                              0xcce, 0xd6e, 0xe0e, 0xeae, 0xf4e };    039
   set_timer();    /* set timer to time in msec */                    040
                                                                      041
   for (i = 0; i < 100; i++)    /* flash central bar 100 times */     042
   {                                                                  043
      while (VID_SYNC_OFF);  /* loop until vertical retrace signal */ 044
      ZERO_TIMER;                                                     045
      while (GET_MSEC < delay);   /* wait for programmed delay */     046
         /* send vertical bar characters to video RAM */              047
      poke (0xb800, vidram[0], bar_chr);                              048
      poke (0xb800, vidram[1], bar_chr);                              049
      poke (0xb800, vidram[2], bar_chr);                              050
      poke (0xb800, vidram[3], bar_chr);                              051
      poke (0xb800, vidram[4], bar_chr);                              052
      poke (0xb800, vidram[5], bar_chr);                              053
      poke (0xb800, vidram[6], bar_chr);                              054
      poke (0xb800, vidram[7], bar_chr);                              055
      poke (0xb800, vidram[8], bar_chr);                              056
      poke (0xb800, vidram[9], bar_chr);                              057
      poke (0xb800, vidram[10], bar_chr);                             058
      poke (0xb800, vidram[11], bar_chr);                             059
      poke (0xb800, vidram[12], bar_chr);                             060
      poke (0xb800, vidram[13], bar_chr);                             061
      poke (0xb800, vidram[14], bar_chr);                             062
      poke (0xb800, vidram[15], bar_chr);                             063
      poke (0xb800, vidram[16], bar_chr);                             064
      poke (0xb800, vidram[17], bar_chr);                             065
      poke (0xb800, vidram[18], bar_chr);                             066
      poke (0xb800, vidram[19], bar_chr);                             067
      poke (0xb800, vidram[20], bar_chr);                             068
      poke (0xb800, vidram[21], bar_chr);                             069
      poke (0xb800, vidram[22], bar_chr);                             070
      poke (0xb800, vidram[23], bar_chr);                             071
      poke (0xb800, vidram[24], bar_chr);                             072
      while (GET_MSEC < 17);   /* wait for end of screen scan  */     073
          /* then erase central vertical bar  */                      074
      poke (0xb800, vidram[0], ' ');                                  075
      poke (0xb800, vidram[1], ' ');                                  076
      poke (0xb800, vidram[2], ' ');                                  077
      poke (0xb800, vidram[3], ' ');                                  078
      poke (0xb800, vidram[4], ' ');                                  079
      poke (0xb800, vidram[5], ' ');                                  080
      poke (0xb800, vidram[6], ' ');                                  081
      poke (0xb800, vidram[7], ' ');                                  082
```

APPENDIX (Continued)

```
      poke (0xb800, vidram[8], ' ');                                        083
      poke (0xb800, vidram[9], ' ');                                        084
      poke (0xb800, vidram[10], ' ');                                       085
      poke (0xb800, vidram[11], ' ');                                       086
      poke (0xb800, vidram[12], ' ');                                       087
      poke (0xb800, vidram[13], ' ');                                       088
      poke (0xb800, vidram[14], ' ');                                       089
      poke (0xb800, vidram[15], ' ');                                       090
      poke (0xb800, vidram[16], ' ');                                       091
      poke (0xb800, vidram[17], ' ');                                       092
      poke (0xb800, vidram[18], ' ');                                       093
      poke (0xb800, vidram[19], ' ');                                       094
      poke (0xb800, vidram[20], ' ');                                       095
      poke (0xb800, vidram[21], ' ');                                       096
      poke (0xb800, vidram[22], ' ');                                       097
      poke (0xb800, vidram[23], ' ');                                       098
      poke (0xb800, vidram[24], ' ');                                       099
                                                                            100
      while (GET_MSEC < 55);   /*  55 msec gets past next two retraces */  101
   }                                                                        102
   reset_timer();  /* MUST be called for disk drives to operate  */        103
   return;                                                                  104
}                                                                           105
                                                                            106
                                                                            107
void draw_scale()    /* draws line locator scale */                         108
{                                                                           109
   void gotoxy();                                                           110
   int i, printf();                                                         111
                                                                            112
   for (i = 1; i <= 25; i++)                                                113
   {                                                                        114
      gotoxy(37, i);                                                        115
      printf("%2d%c %c%d", i, 0xc4, 0xc4, i);                               116
                                                                            117
   return;                                                                  118
}                                                                           119
                                                                            120
                                                                            121
main()    /* Program execution starts here   */                             122
{                                                                           123
   void clrscr(), draw_scale(), draw_vert_bar();                            124
   char *gets(), buffer[5];                                                 125
   int delay, atoi(), printf();                                             126
                                                                            127
   do                                                                       128
   {                                                                        129
      clrscr();                                                             130
      printf("Enter delay from video synch signal (-1 to quit): ");         131
      delay = atoi(gets(buffer));  /* get delay from user */                132
      if (delay >= 0)                                                       133
      {                                                                     134
         clrscr();                                                          135
         draw_scale();                                                      136
         draw_vert_bar(delay);                                              137
      }                                                                     138
   } while (delay >= 0);                                                    139
}                                                                           140
```

(Manuscript received January 27, 1989;
revision accepted for publication May 11, 1989.)