Behavior Research Methods, Instruments, & Computers
1989, 21 (1), 73-83

COMPUTER TECHNOLOGY

Turbo Pascal timing routines for the IBM microcomputer family

MARC BRYSBAERT, NOËL BOVENS, and GÉRY D'YDEWALLE
University of Leuven, Leuven, Belgium

and

JAN VAN CALSTER
Regaschool, Leuven, Belgium

Two Turbo Pascal (versions 3.0 and 4.0) software timers for the IBM microcomputer family are described: one with a 55-msec resolution and another with a 1-msec resolution. Both can be implemented without additional hardware requirements. The 55-msec timer makes use of the system-time-of-day clock; the 1-msec timer is a Turbo Pascal translation of the timer described by Bührer, Sparrer, and Weitkunat (1987). The logic of each timer is shown by a short demonstration program.

In this article, we present two Turbo Pascal timing routines for the IBM microcomputer family that cover the whole range from milliseconds to minutes. Because of the fact that at the moment we are in a transition phase between the old Turbo Pascal 3.0 and the new Turbo Pascal 4.0 versions (Borland Inc., 1987), programs for both are given.

THE SYSTEM-TIME-OF-DAY CLOCK

If long time intervals (minutes, seconds) are to be measured, the simplest strategy is to use the system-time-of-day clock. This clock contains four components: hours, minutes, seconds, and hundredths of seconds. The last component is misleading, however, because the IBM BIOS does not update time information every hundredth of a second, but only every 55 msec. So, successive time intervals for the same process are likely to alternate between two values, .05 or .06 sec apart.

Turbo Pascal 4.0 contains a "GetTime(var Hour, Minute, Second, Sec100 : word)" statement, which directly returns the values of hours, minutes, seconds, and hundredths. Turbo 3.0 does not contain this statement, but a procedure that gives the same information can easily be written (see Appendix A).

To measure a time interval, set one "GetTime" statement at the beginning of the interval and another at the end (see demonstration program in Appendix A). In between, other statements may be executed if they will not disrupt the clock or will not prohibit the end signal.

Correspondence may be sent to Marc Brysbaert, Department of Psychology, University of Leuven, B-3000 Leuven, Belgium.

A TURBO PASCAL MILLISECOND TIMER

To measure reaction times, which seldom exceed 1.5 sec, accuracy of 1 msec is required. Reports of several good millisecond timers for the IBM family recently have been published in this journal (see, e.g., Bührer, Sparrer, & Weitkunat, 1987; Graves & Bradley, 1987; Segalowitz, 1987), but all are assembly language routines to be called for by BASIC main programs. Turbo Pascal is another microcomputer language widely used in psychology laboratories. In this article, we present a translation of the Bührer et al. (1987) timer program that needs no additional hardware.

Bührer et al. (1987) have clearly explained the logic and the precautions (e.g., about the input frequency of the timer/counter chip) of their program. Because our version is a strict translation of theirs, it is not necessary for us to repeat their explanation. Both routines achieve millisecond accuracy by changing the timer interrupt of the 8253 timing chip from 18.2 Hz (55 msec) to 1000.152 Hz (0.999848 msec). The only differences are that our timer routine is not external, and that a slight modification is given to make the procedure accurate for IBM's new Personal System 2. The timer routine is not written in assembler because Turbo Pascal is fast enough and more comprehensible for most users. This becomes especially clear in the 4.0 version.

The accuracy of the program was tested by opposing the video screen refresh rate measured with the timer to the refresh rate measured with an HP (Hewlett-Packard) 5326B Timer/Counter/Digital Volt Meter. The delay of the vertical retrace of an IBM XT clone, measured with the HP timer, was 20.4 msec. Measured with the timing routine, the delay was 20.4 msec when determined after

10 vertical retraces, 20.39 msec when determined after 100 vertical retraces, and 20.388 msec when measured after 1000 vertical retraces (i.e., about 20 sec). Thus, satisfactory accuracy can be achieved, certainly when given the fact that the timer is known to accumulate a systematic underestimation of 152 nsec (Bührer et al. 1987). When this error was removed, the estimation after 1000 vertical retraces was 20.391 msec. Twenty consecutive trials never gave a deviation of more than .001 msec for the estimation after 1000 vertical retraces.

Because the counter is automatically augmented after each interrupt, the timer must be set to zero only at the beginning of the interval. During measurement, one can perform additional programming (e.g., to prepare the next stimulus). However, it should be stressed that because of the automatic interruption every millisecond, Turbo Pascal processes are seriously slowed down (about two times for the XT). After nearly a year of extensive use with the timer, we are convinced that this is not a serious drawback, as long as the timer is switched on and off before and after each time measurement. It is not a good strategy to start the timer at the beginning of a program and to stop it at the end, because this seriously slows down the programming that is often needed to bring the stimuli onto the screen.

A good millisecond timer, however, demands more than just a resolution of 1000 Hz. First, it is necessary to have the stimulus presented at once. An IBM XT takes a little bit more than 1.5 msec to display one character on the video screen with the Turbo "write" instruction. A word of more than 10 letters is likely to be presented during two display frames, and the first part of the word will be shown one frame longer than the second. This becomes more pertinent as more complex (graphical) stimuli are presented. Two solutions are possible. The first involves a temporally disactivation of the screen and is especially useful for the Hercules monochrome adaptor, which usually does not allow two text pages (the solution suggested by Bührer et al., 1987). The second one makes use of different screens and is especially useful when supplementary information (e.g., fixation dots, masks) is to be given before or after the stimulus, or when stimuli are presented in reverse video mode to circumvent screen persistence (Segalowitz, 1987). The disactivation solution, as suggested by Bührer et al., does not work with an Ega or Vga adaptor; so, screen swapping is indicated here.

An additional requisite for an accurate millisecond timer is synchronization between onset of stimulus and timer. Because a screen (depending on the system) is refreshed only every 14.3, 16.7, or 20.0 msec, it is necessary to wait for a new refreshment before starting the timer. If one does not wait for screen refreshment, supplementary variances of 17.01 msec for a 70-Hz system, 23.15 msec for a 60-Hz system, and 33.33 msec for a 50-Hz system must be accepted (Lincoln & Lane, 1980). Synchronization is not possible with an IBM monochrome card.

A factor not taken into account in this article, but undoubtedly important, is the time delay between the computer keyboard and the time counter. Graves and Bradley (1987) showed that this can amount to more than 36 msec with large interkeyboard differences and considerable standard deviations. Therefore, it is advisable not to use the keyboard as a response device, but to connect good external response buttons to the game or printer ports.

The Timing Routines

Both timing routines have been written for the IBM PC operating system. They probably will not work with other operating systems (e.g., CP/M). Efforts have been made to take into account the different video cards (Hercules mono, CGA, EGA, etc.) and systems (PC, XT, AT, PS), but we cannot be sure that the programs will run on each and every IBM-compatible configuration; however, reference manuals can help to make the necessary modifications.

Discussion of Procedures Described in Appendices

Appendix A starts with the "GetTime" procedure necessary to access the system-time-of-day clock in Turbo Pascal 3.0. (Turbo Pascal 4.0 has this procedure as a built-in DOS routine.) Thereafter, a demonstration program is given. A few lines of text state a problem, the solution of which is thought to take more than 1 min, if no calculator is used. So, errors associated with screen refreshment and lack of precision for the hundredths of seconds are likely to be a very small part of the total time delay. The time interval itself is measured by means of two "GetTimes": one before the subject starts the solution and one after he/she has successfully finished it (the "again" label ensures that the subject continues to search as long as mistakes are made).

If the subject gives the right answer, the first time measurement is subtracted from the second to get the response time. The program gives us an idea of some of the difficulties that might arise. We must, after all, be able to subtract such times as 23:23:59.16 from, say, 01:36:14.78. Another way to perform the subtraction is to convert the hours, minutes, and seconds into hundredths, and to compare both amounts of hundredths.

The last section of the program gives the subject feedback about his/her failure and prompts him/her to continue until the correct solution has been found. Time keeps running if a mistake is made, so that the subject cannot profit from guessing.

Appendix B gives the procedures needed for millisecond timing, and a demonstration program. The procedures "newint," "fast," "slow," "setint," and "resetint" are needed to change and restore the interrupt vector of the Intel 8253 timer/counter from 18.2 to 1000.1522 Hz, and vice versa (for more information, see Bührer et al., 1987), and to increase the counter (called "gtel") every time an interrupt occurs. Some of these procedures are built-in routines in Turbo Pascal 4.0.

The "setint" procedure saves the old 18.2-Hz interrupt vector and replaces it with a new interrupt vector, which, after execution of the "fast" procedure, has an

interrupt frequency of 1 kHz. The "slow" and "resetint" procedures restore the old situation. "Setint" and "resetint" are needed only in Turbo 3.0, because they can be replaced by the built-in procedures "getintvec" and "setintvec" in Turbo 4.0. The interrupts increase the counter by one in the procedure "newint." To find the counter, Turbo 3.0 needs the segment address of the counter stored in the variable "dsave."

As already mentioned, a timer can be tested by measuring the time interval needed for 1000 vertical retraces and comparing it with the expected interval (which can be checked with an external device or, to a certain extent, with the system-time-of-day clock). The result of this procedure was very satisfactory for the IBM PC, XT, and AT (see above), but not for the new Personal System 2. About 6% of the vertical retraces were missed when the timer ran, possibly because the BIOS loop in the PS 2 is too long. A very promising alternative for this system (and the other) is not to work with timer interrupt 1C (Bührer et al., 1987), but with timer interrupt 8, which is more directly accessible from the main program. Appendix B contains the necessary adaptations. We believe that the modification might be better than Bührer et al.'s original timer, but thus far we have little experience with it and we are not sure whether a change of interrupt 8 does involve a drawback.

The procedures "video_off" and "video_on" enable temporally disactivation of the screen, while the stimulus is written. This enables presentation of the whole stimulus within one display frame. Note that a vertical retrace is waited for, before the screen is reactivated. Because the bit where the vertical retrace is registered is not the same for the Hercules and the other cards, different instructions are needed. The "video_off" and "video_on" procedures do not work with Ega and Vga adaptors, probably because other memory addresses are involved. An alternative is to use the different pages the systems allow. Appendix B shows the procedures needed to write, show, and erase text stimuli (Borland offers software to work with different screens in graphics mode). The "color" procedure sometimes is necessary to make the colors of both pages equal. The Hercules card does not afford different text pages.

A rather painstaking problem in implementing the timer on the Hercules (monochrome) card was that the second bit of the display mode control port (03B8) did not stay on 1 when the screen was disactivated and reactivated in graphics mode. The only solution we found was to reset the bit every time the screen needed to be turned off and on in graphics mode. This may cause some inconvenience for those who want a fluent switching between text and graphics mode.

The second half of Appendix B gives a demonstration program. The word "hello!" is presented 10 times for an intended interval of 60 msec (actually, for a 50-Hz system, the interval will include 3 frames [60 msec]; for a 60-Hz system, it will include 4 frames [66.88 msec]; and for a 70-Hz system, 5 frames [71.40 msec]). The interstimulus interval varies between 2 and 5 sec after the subject has pressed a key on the previous trial. The last two sections calculate mean and standard deviation of the reaction times.

Availability

A great deal of information had to be put together in a few demonstration programs; therefore, we have written all possible combinations of Turbo versions and graphics cards on a floppy disk, together with procedures to turn off and on the cursor, to switch screens, and to test the timers on your system. The programs can be obtained by sending a 5.25-in. diskette in a returnable box to Marc Brysbaert, at his address at the University of Leuven. For administration costs, $10 must be included.

REFERENCES

BORLAND INTERNATIONAL, INC. (1987). *Turbo Pascal 4.0 IBM version.* Author.

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.

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.

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.

SEGALOWITZ, S. J. (1987). IBM PC tachistoscope: Text stimuli. *Behavior Research Methods, Instruments, & Computers*, **19**, 383-388.

APPENDIX A
55-msec Timer with Demonstration Program

```
program problem_solving_with_the_use_of_GetTime;


{only for Turbo 3.0

  procedure GetTime(var hour, minute, second, sec100: integer);
   type  regs = record
                   ax,bx,cx,dx,bp,di,si,ds,es,flags : integer
                end;
   var   registers : regs;
```

```
   begin
    with registers do
     begin
      ax := $2C00;
      msdos(registers);
      hour := hi(cx);
      minute := lo(cx);
      second := hi(dx);
      sec100 := lo(dx);
     end
   end;       }


{only for Turbo 4.0

  uses Crt, DOS;  }


var
  {a1,a2,b1,b2,c1,c2,d1,d2 : integer;      Turbo 3.0}
  {a1,a2,b1,b2,c1,c2,d1,d2 :    word;      Turbo 4.0}
  x: real;
  t: char;

label again;

begin
  clrscr;
  gotoxy(5,2);
  write('Problem 1: two timers with a difference of 152 nsec');
  gotoxy(5,5);
  write('Calculate how many days  (two digits precision) it takes before');
  gotoxy(5,7);
  write('two counters, one with a count every msec  and one with a count');
  gotoxy(5,9);
  write('every  .999848  msec have the same value, if you know that both');
  gotoxy(5,11);
  write('started at the same time and are reset to zero after 86,400,000');
  gotoxy(5,13);
  write('counts (= 1 day for the good timer).');
  gotoxy(1,16);
  write('n of days');
  gotoxy(5,18);

  GetTime(a1,b1,c1,d1);
  again:                                {repeat if wrong answer}

  read(x);
  GetTime(a2,b2,c2,d2);

  if x = 6578.95 then
   begin
    sound(1000);                        {feedback tone right}
    delay(150);
```

```
    nosound;
    gotoxy(5,20);
    writeln('Okay!');
    if d2 < d1 then                    {subtract gettime1 from gettime2}
     begin
      d2 := d2 + 100;
      c1 := c1 + 1;
     end;
    d2 := d2 - d1;
    if c2 < c1 then
     begin
      c2 := c2 + 60;
      b1 := b1 + 1;
     end;
    c2 := c2 - c1;
    if b2 < b1 then
     begin
      b2 := b2 + 60;
      a1 := a1 + 1;
     end;
    b2 := b2 - b1;
    if a2 < a1 then
     begin
      a2 := a2 + 24;
     end;
      a2 := a2 - a1;
    gotoxy(5,21);
    write('Your time is ',a2,' hours, ',b2,' mins, ',c2,' secs, and ',d2,
             ' hundredths');
   end
  else
   begin
    sound(400);                              {feedback tone wrong}
    delay(350);
    nosound;
    gotoxy(5,20);
    writeln('Not good! Press key and redo.');
    repeat until keypressed;
    {read(kbd,t);      Turbo 3.0     neutralize effect of keypressed}
    {t := readkey;     Turbo 4.0                                     }
    gotoxy(5,18);
    clreol;
    gotoxy(5,20);
    clreol;
    gotoxy(5,18);
    goto again;
   end;

end.
```

APPENDIX B
Millisecond Timer with Demonstration Program

```
program millisecond_timer_turbo_pascal;


(* only for Turbo 3.0

const  dsave : integer = 0;
          {a typed constant that will hold the value of DS.  It is
           located in the code segment so it is accessible by the
           interrupt service routine}

type  regs  = record
                case boolean of
                 true : (  ax,bx,cx,dx,bp,si,di,ds,es,flags : integer);
                 false : (al,ah,bl,bh,cl,ch,dl,dh : byte);
               end;

var r : regs;
    gtel,it : integer;
    segs,ofss:integer;

procedure newint;
begin
 inline($50/$1e/$2e/$ff/$36/dsave/$1f);   {push ax/push ds/push cs:dsave}
                                         {push address of dsave on stack}
 inline($ff/$06/gtel);                                  {increment gtel}
{inline($b0/$20/$e6/$20);                                  modification}
                      {mov al,eoi/out 020H,al ; end of interrupt to 8259}
 inline($1f/$58/$5c/$5d/$cf);          {pop ds/pop ax/pop sp/pop bp/iret}
                                                      {interrupt return}
end;

procedure setint;
begin
 with r do
  begin
   ah := $35;
{  al := $1c;                                  Bührer, Sparrer & Weitkunat}
{  al := $8;                                                modification}
  end;
 msdos(r);
 segs := r.es;
 ofss := r.bx;
 with r do
  begin
   ah := $25;
{  al := $1c;                                  Bührer, Sparrer & Weitkunat}
{  al := $8;                                                modification}
   ds := Cseg;
   dx := Ofs(newint);
  end;
 msdos(r);
end;
```

```
procedure resetint;
begin
 with r do
  begin

   ah := $25;
{  al := $1c;                                  Bührer, Sparrer & Weitkunat}
{  al := $8;                                                 modification}
   ds := segs;
   dx := ofss;
  end;
 msdos(r);
end;

*)



(* only for Turbo 4.0

uses Dos, Crt;

var  gtel : integer;

procedure newint;
interrupt;
begin
 inc(gtel);
{inline($b0/$20/$e6/$20);                                    modification}
end;

*)


procedure fast;
begin
 inline($fa);                                          {disable interrupts}
 port[$43] := $36;                                             {channel 3}
 port[$40] := $a9;                                           {new divisor}
 port[$40] := $04;
 inline($fb);                                           {enable interrupts}
end;

procedure slow;
begin
 inline($fa);                                          {disable interrupts}
 port[$43] := $36;                                             {channel 3}
 port[$40] := $00;                                           {new divisor}
 port[$40] := $00;
 inline($fb);                                           {enable interrupts}
end;
```

```
(* not for EGA and VGA adaptors

procedure video_off;                    {see BIOS listing, data segment $40}
var iw : byte;
    p : integer;
begin
 p := memw[$40:$63];                                  {get address of 6845}
 p := p+4;
 iw := mem[$40:$65];                                {get value of video-mode}
 iw := iw and $F7;
 port[p] := iw;                                                {turn video off}
{port[p] := iw or 2;                                 HercMono in graphics mode}
 end;

 procedure video_on;                    {see BIOS listing, data segment $40}
 var iw  : byte;
     p,q : integer;
 begin
  p := memw[$40:$63];                                  {get address pf 6845}
  p := p + 4;
  q := p + 2;
                                                    {wait for vertical retrace}
 {repeat until (port[q] and 8) =  0;
  repeat until (port[q] and 8) <> 0;                                    Cga}
 {repeat until (port[q] and $80) <> 0;
  repeat until (port[q] and $80) =  0;                                  HercMono}
  iw := mem[$40:$65];
  port[p] := iw;                                                {turn video on}
 {port[p] := iw or 2;                                 HercMono in graphics mode}
 end;

 *)


 (* not for Hercules adaptor

 type strin = string[80];

                                 { 0 ≤ page ≤ 3}

 {procedure color(number,page:integer);                                 Turbo 3.0}
 {procedure color(number,page:byte);                                    Turbo 4.0}
 var ikl : integer;
 begin
  for ikl := 1 to 2000 do
 { mem[$b800:2*ikl-1+4096*page] := number;                              Turbo 3.0}
 { mem[$b800:2*ikl-1+4096*page] := (mem[$b800:2*ikl-1
                           +4096*page] and 0) or number;                Turbo 4.0}
 end;

 procedure wrt(ok,ol:integer;word:strin;page:integer);
 var str1 : string[1];
 {   k    : integer;                                                    Turbo 3.0}
```

```
{   k    : byte;                                                  Turbo 4.0}
    ikl  : integer;
begin
 for ikl := 1 to length(word) do
  begin
   str1 := copy(word,ikl,1);                {split the word into letters}
{  k := integer(str1);                 {ASCII codes of letters  Turbo 3.0}
{  k := hi(integer(str1));                                       Turbo 4.0}
   mem[$b800:(160*(ol-1)+2*(ok-2+ikl)+4096*page] := k;
 end;
end;

procedure set_screen(page);
{type registers = record
                   ax,bx,cx,ds,bp,si,di,ds,es,flags:integer;
                  end;                                           Turbo 3.0}
var save : registers;
begin
 save.ax := $0500 + page;
 repeat until(port[memw[$40:$63]+6] and 8) = 0;   {wait for vert retrace}
 repeat until(port[memw[$40:$63]+6] and 8) <>0;
 intr($10,save);
end;

procedure clearscreen(page);
var ikl : integer;
begin
 for ikl := 1 to 25 do
 wrt(1,ikl,'                                          ' +
           '                                          ',page);
end;

*)



{* * * * * * * * * * * * * end of timer routines * * * * * * * * * * * * *}


var i, j : integer;
    ch : char;
    saved : array[1..10] of integer;
    mean, vari: real;
{   int1Csave : pointer;                                         Turbo 4.0}


begin
 clrscr;
{dsave := dseg;                                                  Turbo 3.0}
{color(15,0);
 color(15,1);            white letters on black background not Herc mono}
 for i := 1 to 10 do
```

```
  begin
{  video_off;                          blank video screen(not EGA, VGA)}
{  clearscreen(1);                                        blank screen 1
   set_screen(1);                                          not Herc mono}
{  gotoxy(37,10);
   write('hello!');                                        not EGA, VGA }
{  clearscreen(0);
   wrt(37,10,'hello!',0);                                  not Herc mono}
   delay(2000+random(3000));                 {interstimulus interval}
{  setint;                                                  Turbo 3.0 }
{  getintvec($1c,int1Csave);
   setintvec($1c,@newint);                 Turbo 4.0 ; Bührer, et al.}
{  getintvec($8,int1Csave);
   setintvec($8,@newint);                  Turbo 4.0 ; modification  }
   fast;
{  video_on;                  enable video screen; wait for vert. ret.
                                                        not EGA, VGA }
{  set_screen(0);                                       not Herc mono}
   gtel := 0;                                          {set timer to 0}
   repeat until gtel = 60;  {present stimulus for about 60 msec;see text}
{  video_off;                                           not EGA, VGA }
{  set_screen(1);                                       not Herc mono}
   repeat until keypressed;
   saved[i] := gtel;
   slow;
{  resetint;                                                Turbo 3.0}
{  setintvec($1c,Int1Csave);               Bührer, et al. ; Turbo 4.0}
{  setintvec($8,int1Csave);                modification   ; Turbo 4.0}
{  if keypressed then read(kbd,k);       neutralize keypress   Turbo 3.0}
{  if keypressed then ch := readkey;     neutralize keypress   Turbo 4.0}
{  gotoxy(37,10);
   clreol;
   video_on;                                            not EGA, VGA }
  end;
{clearscreen(0);
 set_screen(0);                                         not Herc mono}

 mean := 0;  vari := 0;           {calculate mean and standard deviation}
 for i := 1 to 10 do
  mean := mean + int(saved[i])/10;
 for i := 1 to 10 do
  vari := vari + sqr(int(saved[i])-mean)/9;

 clrscr;
 for i := 1 to 10 do
  begin
   gotoxy(10,3+i);
   write(saved[i]);
  end;
 gotoxy(10,14);
 writeln('-----------');
```

```
  gotoxy(3,15);
  writeln('mean:  ',mean:4:2);
  gotoxy(3, 16);
  vari := sqrt(vari);
  writeln('SD   :  ',vari:4:2);

end.
```

(Manuscript received September 26, 1988;
accepted for publication October 24, 1988.)