Behavior Research Methods, Instruments, & Computers
1990, 22 (3), 332-334

— PROGRAM ABSTRACTS/ALGORITHMS —

IBM PC/XT/AT and PS/2 Turbo Pascal timing with extended resolution

NOËL BOVENS and MARC BRYSBAERT
University of Leuven, Leuven, Belgium

A Turbo Pascal timer with accuracy beyond 1 msec is described. Unlike some previous timers, this one does not interfere with the time-of-day clock, and works well on an unmodified IBM PC/XT.

In earlier papers (Brysbaert, Bovens, d'Ydewalle, & Van Calster, 1989; Heathcote, 1988), Turbo Pascal timers for the IBM microcomputer have been described that achieve millisecond accuracy by changing the frequency of the time-of-day interrupt from 18.2 Hz (55 msec) to 1000.1522 Hz (about 1 msec). The procedure was initially proposed by Bührer, Sparrer, and Weitkunat (1987).

At about the same time, Graves and Bradley (1987, 1988) published assembly language routines to be called from BASIC main programs, which achieve extended resolution by setting the timer chip into Mode 2 and reading the BIOS timer fields and the Counter 0 residual counts. The main advantages of this procedure are that normal program execution is not interrupted as many times, and that BIOS time-of-day information is preserved. Emerson (1988) has proposed a program for the C language, which has the same advantages, but which works without compromises only on an AT.

Inspired by Graves and Bradley (1987, 1988) and Emerson (1988), we present (see Appendix A) a list of Turbo Pascal (Version 4.0 and 5.0) routines, to be included in the main program (see Appendix B for an example).

Timer.inc: The Timer Routines

The procedure timer_mod_2 waits until a time-of-day interrupt occurs and then changes the timer chip (from Mode 3) to Mode 2. This change is necessary to access the residual count (between two interrupts) of the timer-counter chip (whether an Intel 8253 or an Intel 8254).

The proceudre timer_mod_3 restores Mode 3 of the timer-counter chip.

The function timex is a Turbo Pascal translation of Emerson's (1988) C-language function. It has the same characteristics and returns the number of "tocks" (= a time unit of 53.6381 μsec; Emerson, 1988) since the last "midnight" (i.e., the moment at which the time-of-day clock is reset to zero). Insertion of two timex functions in the program, one before and one after the time interval to be measured, yields an estimate of the elapsed time through subtraction of the first timex function from the last. Dividing the difference by 18.6435 converts the "tocks" into milliseconds.

The time needed for the timex function itself can be estimated by measuring the time between two successive timex calls. On an IBM XT clone (with a 4.77-MHz processor clock), 5,000 replications gave a mean interval of 0.358 msec (minimum = 0.322 msec, maximum = 0.697 msec). Thus, millisecond accuracy is obtainable, and because most systems nowadays have a higher speed, the data presented here constitute a lower limit.

Compared with Graves and Bradley's (1987, 1988) routines, those in Appendix A have two major advantages. First, there is no need to use assembly routines, and second, interrupts during actual time reading are corrected for. The latter can be explained as follows: Because there is no way to access the BIOS timer fields and the Counter 0 residual counts exactly at the same time, a time-of-day interrupt can occur in the small time gap between disabling the interrupt and latching the timer data, which leads to an error of ±55 msec. The error can easily be demonstrated by making a tight loop of two consecutive time readings. Quite regularly, the elapsed time will amount to ±55 msec, instead of the normal 0.358 msec (see above). The problem is solved by a double reading of the BIOS timer fields in the timex routine and a correction if an interrupt has occurred between both readings. This correction is not present in the timer described by Graves and Bradley (1987, 1988) or in a recently published paper by Crosbie (1989), who presents timer routines quite comparable to ours. Because Crosbie uses a combination of two Turbo Pascal procedures to start and stop his timer, the time gap between registering the time-of-day calls and latching the timer counts is quite large (testing of Crosbie's timer with an XT clone [4.77 MHz processor clock] showed that the 55-msec error occurred once out of every 400 readings, on the average).

Compared with Emerson's (1988) C routines, those in Appendix A have one advantage: The timer can be run with an Intel 8253 timer-counter chip.

The routines in Appendix A are better than those previously proposed by Heathcote (1988) and Brysbaert et al. (1989) because they have a greater resolution, they do not interfere with the time-of-day clock, and they do not slow down normal program execution.

A Demonstration Program

Appendix B gives a short demonstration program. It measures the time between the start of a sound and the

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

—Accepted by previous editor, Joseph B. Sidowski

pressing of a key. At the beginning of the program, the timer must be initialized by the timer_mod_2 procedure. At the end of the program, the normal situation must be restored by using the timer_mod_3 procedure. Elapsed time is registered and shown to the subject.

REFERENCES

Brysbaert, 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.

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.

Emerson, P. L. (1988). TIMEX: A simple IBM/AT C language timer with extended resolution. *Behavior Research Methods, Instruments, & Computers*, **20**, 566-572.

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.

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.

APPENDIX A
Timer.inc: The Timer Routines

```
{Must be included in the main program by $i compiler command}


Const
  TIMER_0       = $40;
  TIMER_CTRL    = $43;
  SET_TIMER     = $34;
  RESET_TIMER   = $36;
  BIOS_SEG      = $40;
  TIMER_LOW     = $6c;
  TIMER_HIGH    = $6e;
  TIMER_LATCH   = $00;


Procedure timer_mod_2;
Var t:byte;
begin
  t:=mem[BIOS_SEG:TIMER_LOW];
  repeat until t <> mem[BIOS_SEG:TIMER_LOW];  {wait until counter interrupt}
  port[TIMER_CTRL]:=SET_TIMER;                    {set timer-chip in Mode 2}
  port[TIMER_0]:=0;
  port[TIMER_0]:=0;
end;

Procedure timer_mod_3;
Var t:byte;
begin
  t:=mem[BIOS_SEG:TIMER_LOW];
  repeat until t <> mem[BIOS_SEG:TIMER_LOW];  {wait until counter interrupt}
  port[TIMER_CTRL]:=RESET_TIMER;                  {set timer-chip in Mode 3}
  port[TIMER_0]:=0;
  port[TIMER_0]:=0;
end;

Function Timex:longint;
Var nticks1,nticks2:longint;
    lo,hi,ntocks:word;
begin
  inline($fa);                                           {disable interrupts}
  port[TIMER_CTRL]:=TIMER_LATCH;                       {save residual count}
  lo:= port[TIMER_0];
  hi:= port[TIMER_0];
  nticks1:=meml[BIOS_SEG:TIMER_LOW];                    {read timer counter}
  inline($fb);                                            {enable interrupts}
  ntocks:=(hi shl 2) or (lo shr 6);
  inline($fa);                                           {disable interrupts}
  nticks2:=meml[BIOS_SEG:TIMER_LOW];                    {read timer counter}
  inline($fb);                                            {enable interrupts}
  ntocks:=NOT(ntocks) and 1023;
  if (hi<>0) then timex:=(nticks2 shl 10) or ntocks  {calculate number of}
             else timex:=(nticks1 shl 10) or ntocks;                {tocks}
end;
```

APPENDIX B
Demonstration Program

```
Program demonstration;

uses Crt;

{$i timer.inc}                 {include the millisecond timing procedures}
                                                  {described in Appendix A}

var ch : char;
    t1,t2 : longint;

begin
 clrscr;
 randomize;
 gotoxy(12,2);
 write('Stop the sound by pressing any key.  Quit by ESC-key');
 timer_mod_2;                                           {initialize timer}
 repeat
  delay(2000+random(1000));                           {wait random delay}
                                           {neutralize possible type-ahead}
  while keypressed do ch := readkey;
  sound(880);                                     {give signal to the subject}
  t1 := timex;                     {get time at the beginning of the interval}
  repeat until keypressed;                                  {wait for key-press}
  t2 := timex;                            {get time at the end of the interval}
  nosound;                                                  {stop making noise}
  ch := readkey;                                          {neutralize keypress}
  gotoxy(25,10);
  writeln('Elapsed time was : ',(t2-t1)/18.6435:8:1,' msec');
                                              {give feedback to the subject}
 until ch = chr(27);
 timer_mod_3;                             {restore timer-chip in Mode 3}
end.
```

(Manuscript received May 15, 1989;
revision accepted for publication January 8, 1990.)