PROGRAM ABSTRACTS/ALGORITHMS

A simple Turbo Pascal 4.0 program for millisecond timing on the IBM PC/XT/AT

JOHN CROSBIE

Deakin University, Victoria, Australia

The IBM PC has become the de facto standard computer for use in psychology laboratories, and Turbo Pascal is a very popular language for this purpose. Unfortunately, no simple millisecond timing program is available in Turbo Pascal. Heathcote (1988) presented a Turbo Pascal program for this purpose, but it is complex and adversely affects the time of day (TOD) clock. Other programs are available only in BASIC and in assembly language, and therefore they cannot be incorporated directly into Turbo Pascal programs (e.g., see Bühner, Sparrer, & Weitkunat, 1987; Graves & Bradley, 1987, 1988). In the present paper, a program is described that performs millisecond interval timing for the IBM PC family of computers simply, neatly, efficiently, and accurately, entirely within Turbo Pascal 4.0.

Millisecond timing. On the IBM PC, XT, and AT, as well as compatibles, system timing (e.g., RAM refresh and updating the TOD clock) is performed by the Intel 8253 timer/counter provided on the motherboard. Regardless of the computer (i.e., PC, XT, or AT) and central processor speed (i.e., 4.77, 6, 8, 10, or 20 MHz), the 8253 timer/counter always operates at 1,193,182 Hz, with one count for every cycle. In normal operation, the TOD clock is updated every 65,536 counts of the 8253 timer—that is, every 54.925 msec. One way to obtain millisecond timing is to program the 8253 timer to update the TOD clock every 1,193 counts (i.e., approximately every millisecond) and then count the number of times the TOD clock is updated during the timing interval. However, because the BIOS assumes that a TOD update occurs every 55 msec, the TOD clock will run 55 times too fast (see Bühner et al., 1987, and Heathcote, 1988, for examples of this type of algorithm).

The 8253 is initialized with an integer between 1 and 65,536, then it repeatedly counts down from this number to 0. When the counter reaches 0, three things happen automatically: (1) hardware Interrupt 8 updates the TOD clock, (2) software Interrupt 28 (1C in Hexadecimal) is executed, and (3) the counter is reset to the initial number and the countdown continues. In normal operation, Interrupt 1C does nothing useful, but it can be modified

to execute a user-defined procedure when the interrupt is performed (i.e., when the TOD clock is updated; see Sargent & Shoemaker, 1984, for more information). Another way to obtain millisecond timing is to program the 8253 to count down from 65,536 (and therefore update the TOD clock appropriately), read the TOD clock and 8253 count at the beginning and end of the timing interval, and thereby calculate the number of 8253 counts during the interval. Reading the TOD clock in a high-level language such as Pascal or BASIC is somewhat slow, and using assembly language complicates the task (see Graves & Bradley, 1987, 1988, for examples of such an algorithm).

The present program. The program shown in the Appendix, MSECRT.PAS, employs elements of both of the algorithms described above to measure auditory reaction time with millisecond precision. The 8253 is programmed to count down from 65,536 so the TOD clock is not adversely affected (INSTALLTIMER), Interrupt 1C is modified to count the number of times the TOD clock is updated during the timing interval (INT1CHANDLER), and the 8253 count is recorded at both the beginning (STARTTIMER) and the end of the interval (STOPTIMER). Elapsed time is computed with the following formulas:

$$\text{Counts} = (\text{Updates} \times 65,536) + (\text{Count0} - \text{Count1})$$

$$\text{Time} = \text{Counts} \times \text{MsecsPerCount},$$

where Counts is the number of 8253 counts, Updates is the number of TOD updates, Count0 is the initial 8253 count, Count1 is the final 8253 count, Time is elapsed time in milliseconds, and MsecsPerCount is the number of milliseconds per 8253 count (0.0008380953). Similarly, microsecond precision can be obtained by multiplying the number of counts by the microsecond per count (0.8380953). When millisecond timing is no longer required, Interrupt 1C is returned to its original state (REMOVETIMER).

In addition to the simplicity of the timing method employed in MSECRT.PAS and the advantages discussed above, the technique is very accurate (times are within 0.21% of those obtained by an external timer), and it has only a small timing overhead. For example, on an Olivetti M24 running at 4.77 MHz, STARTTIMER takes 140 μsec , STOPTIMER takes 140 μsec , CLEARPREVIOUSRESPONSES takes 150 μsec , and the REPEAT UNTIL RESPONSEMADE loop takes 170 μsec . Therefore, the entire timing procedure takes less than 600 μsec .

MSECRT.PAS is merely an example of how the present timing procedure could be implemented in a simple auditory reaction time study. To simplify the demonstration,

Requests for reprints of this article and copies of the program should be sent to John Crosbie, Psychology Department, Deakin University, VIC 3217, Australia.

in this program keyboard responses have been employed, but, for serious scientific purposes, keyboard responses are far too slow (Graves & Bradley, 1987, report mean delays of 18.4 msec for an IBM PC keyboard and 36.7 msec for a clone keyboard), and more precise and faster response devices (e.g., good switches or mouse buttons) should be used instead.

Availability. A 5¼-in. diskette containing an ASCII version of MSECRT.PAS and MSECRT.INC (a file that can be included in any Turbo Pascal program to obtain millisecond timing) is available from the author for \$10 (Australian) to cover materials, postage, and handling.

REFERENCES

- 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.
- 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.
- SARGENT, M., III, & SHOEMAKER, R. L. (1984). *The IBM personal computer from the inside out*. Reading, MA: Addison-Wesley.

APPENDIX

Listing of MSECRT.PAS

```
PROGRAM MsecRt ;

{ This program employs the 8253 counter as a msec timer. }

USES
    Crt,
    Dos ;

CONST
    ScreenColor      = Blue ;
    MessageColor     = White ;
    PromptColor      = LightGreen ;
    ResultColor       = LightCyan ;
    ContColor        = Yellow ;
    CountsPerSec      = 1193182 ;
    CountsPerTodCall  = 65536 ;
    MsecsPerCount     = 0.0008380951 ;
    TimerCtl          = $43 ;
    Timer0            = $40 ;
    TimerSet          = $34 ;
    TimerLatch        = $00 ;

VAR
    TimerCountLo,
    TimerCountHi,
    StartCounts,
    StopCounts,
    RtCounts,
    RtMsecs,
    StopTodCalls,
    NTodCalls         : LONGINT ;
    Int1CSave         : POINTER ;
    Ch,
    Answer             : CHAR ;

{-----}

PROCEDURE DisableInterrupts ; INLINE ( $FA ) ;

{-----}

PROCEDURE EnableInterrupts ; INLINE ( $FB ) ;

{-----}

PROCEDURE Nop ; INLINE ( $90 ) ;

{-----}

PROCEDURE Int1CHandler (Flags, CS, IP, AX, BX, CX, DX,
    SI, DI, DS, ES, BP : WORD) ;

Interrupt ;

{ This procedure is invoked automatically (via Interrupt $1C)
  every time the Time of Day (TOD) clock is updated.
```

APPENDIX (Continued)

Therefore, this procedure keeps a count of the number of times the TOD clock is updated. }

```

BEGIN
    Inc (NTodCalls) ;

END ; { Of IntlCHandler }

{-----}

PROCEDURE InstallTimer ;

{ This Procedure initializes the 8253 to count down from
  65536, saves the address of the procedure normally executed
  when Interrupt $1C is serviced (IntlCSave), and replaces it
  with the address of the new procedure to be executed by
  Interrupt $1C (IntlCHandler). }

BEGIN
    Port [ TimerCtl ] := TimerSet ;
    NOP ;                { Time for 8253 to recover }
    NOP ;
    Port [ Timer0 ] := 0 ; { Load Lo Byte of count;
                           0 -> 65536 }
    NOP ;                { Time for 8253 to recover }
    NOP ;
    Port [ Timer0 ] := 0 ; { Load Hi Byte of count;
                           0 -> 65536 }

    GetIntVec ( $1C, IntlCSave ) ;
    SetIntVec ( $1C, @IntlCHandler ) ;

END ; { Of InstallTimer }

{-----}

PROCEDURE RemoveTimer ;

{ This procedure dismantles the msec timer by returning
  the $1C interrupt to its original condition and
  thereby removing the interrupt handler (IntlCHandler). }

BEGIN
    SetIntVec ( $1C, IntlCSave ) ;

END ; { Of RemoveTimer }

{-----}

PROCEDURE GetCounts ( VAR NumCounts : LONGINT ) ;

{ This procedure returns the current 8253 count. }

BEGIN
    DisableInterrupts ;      { Stop interrupts while
                              reading count }
    Port [ TimerCtl ] := TimerLatch ;
    NOP ;                    { Time for 8253 to recover }
    NOP ;
    TimerCountLo := Port [ Timer0 ] ;
    NOP ;                    { Time for 8253 to recover }
    NOP ;
    TimerCountHi := Port [ Timer0 ] ;

    EnableInterrupts ;      { Finished reading count;
                              resume interrupts }
    NumCounts := (TimerCountHi SHL 8)
                + TimerCountLo ;

END ; { Of GetCounts }

{-----}

PROCEDURE DisplayScreen ;

```

APPENDIX (Continued)

```

{ This procedure sets the background color, clears the screen,
  and produces messages on the screen. }

BEGIN
  TextMode ( C80 ) ;
  TextBackground ( ScreenColor ) ;
  ClrScr ;

  gotoxy ( 31, 4 ) ;
  TextColor ( MessageColor ) ;
  WRITE ( 'REACTION TIME TEST' ) ;
  gotoxy ( 22, 13 ) ;
  TextColor ( PromptColor ) ;
  WRITE ( 'Press any key when you hear the tone' ) ;
  gotoxy ( 80, 25 ) ;

END ; { Of DisplayScreen }

{-----}

PROCEDURE PresentStimulus ;

{ This procedure presents a 1000 HZ tone for 10 msec. }

BEGIN
  { Wait for a short randomly-determined period
    before stimulus presentation. }
  Delay ( 2000 + Random ( 3000 ) ) ;

  Sound ( 1000 ) ;
  Delay ( 10 ) ;
  NoSound ;

END ; { Of PresentStimulus }

{-----}

PROCEDURE StartTimer ;

{ This procedure initializes the number of TOD calls, and
  returns the 8253 count at the start of timing. }

BEGIN
  NTodCalls := 0 ;
  GetCounts ( StartCounts ) ;

END ; { Of StartTimer }

{-----}

PROCEDURE StopTimer ;

{ This procedure obtains the number of TOD calls since the
  start of timing, and returns the 8253 count
  at the end of timing. }

BEGIN
  StopTodCalls := NTodCalls ;
  GetCounts ( StopCounts ) ;

END ; { Of StopTimer }

{-----}

PROCEDURE ClearPreviousResponses ;

{ This procedure clears the response buffer so that responses
  made before stimulus offset will not be recorded. }

BEGIN
  WHILE KeyPressed DO
    ch := ReadKey ;

END ; { Of ClearPreviousResponses }

```

APPENDIX (Continued)

```

{-----}

FUNCTION ResponseMade : BOOLEAN ;

{ This function returns whether a response has been made
  since stimulus offset. }

BEGIN
  IF KeyPressed THEN
    BEGIN
      ResponseMade := true ;
      { Clear pressed key from buffer }
      WHILE KeyPressed DO
        ch := ReadKey ;
      END
    ELSE
      ResponseMade := false ;
    END ; { Of ResponseMade }

{-----}

PROCEDURE DisplayRt ;

{ This procedure calculates the number of 8253 counts during
  the timing period, then calculates and displays the msec
  that correspond to this number of counts. }

BEGIN
  RtCounts := (StopTodCalls * CountsPerTodCall)
    + (StartCounts - StopCounts) ;
  RtMsecs := ROUND (RtCounts * MsecsPerCount) ;

  gotoxy ( 35, 19 ) ;
  TextColor ( ResultColor ) ;
  WRITE ( RtMsecs:5, ' Msec' ) ;

END ; { Of DisplayRt }

{-----}

PROCEDURE AnyMoreTrials ;

{ This procedure asks the user whether more trials are
  required. }

VAR
  Col : INTEGER ;

BEGIN
  gotoxy ( 17, 22 ) ;
  TextColor ( ContColor ) ;
  WRITE ( 'Press the SPACE bar to continue, or S to STOP' ) ;
  gotoxy ( 80, 25 ) ;
  Answer := ReadKey ;
  IF NOT (Answer IN [ 'S', 's' ]) THEN
    BEGIN
      { Blank Result line }
      gotoxy ( 35, 19 ) ;
      WRITE ( ' ' ) ;
      { Blank Prompt line }
      FOR Col := 17 TO 75 DO
        BEGIN
          gotoxy ( Col, 22 ) ;
          WRITE ( ' ' ) ;
        END ;
      gotoxy ( 80, 25 ) ;
    END ;

END ; { Of AnyMoreTrials }

{-----}

```

APPENDIX (Continued)

```
{ Main Program }

BEGIN
  DisplayScreen ;
  InstallTimer ;

  REPEAT
    PresentStimulus ;
    StartTimer ;
    ClearPreviousResponses ;
    REPEAT
      UNTIL ResponseMade ;
    StopTimer ;
    DisplayRt ;
    AnyMoreTrials ;
  UNTIL Answer IN [ 'S', 's' ] ;
  RemoveTimer ;

END.
```

(Manuscript received September 26, 1988;
revision accepted for publication May 5, 1989.)