Notes on Graves and Bradley's timer algorithm

JÖRG BERINGER

Technische Hochschule Darmstadt, Darmstadt, Germany

This paper presents an extension of the Graves and Bradley (1987) millisecond timer, which avoids potential measurement errors of 54.9 msec. The synchronization of the system timer count with the BIOS timer counter must be taken into account to eliminate such errors.

Although there are many variants of language-specific implementations of millisecond timers (see, e.g., Brysbaert, Bovens, d'Ydewalle, & Van Calster, 1989; Creeger, Miller, & Paredes, 1990; Crosbie, 1989; Heathcote, 1988), there are only a few underlying algorithms for accessing high-resolution time on a standard IBM PC. Basically, such algorithms fall into two types: implementation of an interrupt-driven low-resolution software counter, or direct reading of the time out of the timer chip. Algorithms of the latter type are superior, because they produce no overhead for the CPU—in contrast with algorithms that implement interrupt-driven software counters triggered by system timers (Buehrer, Sparrer, & Weitkunat, 1987; Dlhopsky, 1988) or serial interfaces (Emerson, 1988b). A tradeoff between clock resolution and CPU processing overhead is always inherent in such interrupt-driven software counters.

The algorithms presented by Graves and Bradley (1987) and Emerson (1988a) read the timer chip directly. Whereas Graves and Bradley's (1987) algorithm reprograms Channel 0 of the timer chip into mode 0, Emerson uses the timer in the standard mode 3, in which the timer counts down twice with increments of 2. Using mode 3 has some caveats, in that the status bit of the timer has to be taken into account and this mode is also not applicable on an IBM XT (Emerson, 1988a).

Thus, the millisecond timer algorithm of Graves and Bradley (1987), which was originally introduced by Smith and Puckett (1984), is in principle the most efficient and precise timer algorithm available for the IBM PC. After some testing, however, the algorithm turned out to be unstable with respect to the measured time delays by as much as 54.9 msec. Presenting a Turbo Pascal implementation of Graves and Bradley's (1987) algorithm, Bovens and Brysbaert (1990) also reported the same problems when using the original algorithm. Within their Turbo Pascal routine, they presented a correction method similar to the one suggested in this paper.

The measurement error in Graves and Bradley's (1987) algorithm is not surprising from a technical point of view. The algorithm is based on reading the BIOS timer coun-

ter of the BIOS Data Area at 0040:006C together with the fast running system timer count (IBM, 1987). Concatenating both timer values results in a $16 + 32 = 48$ -bit integer value, which represents the current system time in units of system timer ticks. The algorithm relies on the assumption that the BIOS timer counter is updated via INT08 without any delay. Unfortunately, this is not the case when interrupts are disabled. Although interrupts are disabled, the system timer count continues running. When sending the read command to the timer chip while interrupts are disabled, a system timer overflow can occur without causing the update of the BIOS timer counter.

Empirical testing on a 20-MHz 80386 computer revealed that execution time between the CLI assembler statement and the read command takes about two system timer ticks. If the algorithm reads a system timer count of hex FFFE or greater, the BIOS timer counter will not have been updated yet. Interpreting the not-yet updated BIOS timer counter produces an error of an entire system timer cycle, which is 54.9 msec. The chance of producing invalid measurements in this example is only $2/65,535$, but it is easily reproduced when one uses the original algorithm published by Graves and Bradley (1987; see also Bovens & Brysbaert, 1990).

Consequently, the timer algorithm must be changed in a way that rules out the possibility of the reported error. A fast and efficient method is to read the BIOS timer counter again for verification if the system timer count is in the upper half of one cycle. On the second read, the BIOS timer counter is always updated because interrupts have previously been enabled. If the system timer counter value is in the lower half, no updating problems exist.

The listing gives an example of this extended verification method. The original version of Graves and Bradley's assembler language code is coded in Microsoft C (Version 6.0) in-line assembler code to provide access from other C routines.

A final comment for users of this kind of millisecond timer concerns the incompatibility of the AT&T timer chip in the AT&T 6300 and the Olivetti M24, which runs at the slightly different frequency of 1.2288 MHz (Olivetti, 1984). These XT-compatible models are old but apparently still in use (Crosbie, 1989). The conversion of AT&T timer ticks into milliseconds requires a correction factor of 0.0008138.

Correspondence should be addressed to Jörg Beringer, Institut für Psychologie, Technische Hochschule Darmstadt, Hochschulestraße 1, 6100 Darmstadt, Germany.

REFERENCES

- BOVENS, N., & BRYSHAERT, M. (1990). IBM PC/XT/AT and PS/2 Turbo Pascal timing with extended resolution. *Behavior Research Methods, Instruments, & Computers*, **22**, 332-334.
- BRYSHAERT, 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.
- BUEHRER, 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). Micromanaging time: Measuring and controlling timing errors in computer-controlled experiments. *Behavior Research Methods, Instruments, & Computers*, **22**, 34-79.
- 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 timing on the IBM PC. *Behavior Research Methods, Instruments, & Computers*, **20**, 560-565.
- 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.
- 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.
- 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.
- IBM (1987). *Personal System/2 and personal computer BIOS interface technical reference*. Portsmouth, U.K.: International Business Machines Corporation.
- OLIVETTI & Co. (1984). *Hardware architecture and function: Olivetti Personal Computer M24, M21*. Ivrea, Italy: Author.
- SMITH, B., & PUCKETT, T. (1984, April). Life in the fast lane. *PC Technical Journal*, 1984, pp. 63-74.

NOTE

1. Graves and Bradley recently published a corrected version of their assembler in this journal: "Millisecond timing on the IBM PC/XT/AT and PS/2: A review of the options and corrections for the Graves and Bradley algorithm" by R. E. Graves and R. Bradley, 1991, *Behavior Research Methods, Instruments, and Computers*, **23**, 377-379. (Editor's note)

LISTING

Function GETTIC() Written in Microsoft C Version 6.0

```

/* ===== */
/*Function gets current time in units of system ticks */
/*===== */

void gettic(unsigned int *TimerTick, unsigned long *BiosTick)
{
    __asm {
        push    si
        mov     bx, BiosTick          ; BX = pointer to BiosTick
        mov     ax, 40h
        mov     es, ax                ; ES = BDA segment
        mov     si, 6ch               ; SI = Offset of system timer counter location control
        xor     al, al                ; control byte 00h: latching, reading while counting
        cli
        out     43h, al               ; command to port 43h and
        mov     cx, word ptr es:[si]  ; get BIOS timer counter
        mov     dx, word ptr es:[si+2] ; near by with interrupt disabled
        in      al, 40h               ; read timer count channel 0
        xchg    ah, al
        in      al, 40h
        sti
        xchg    al, ah

        ; ----- new verification -----

        test    ah, 80h               ; NEW if timer count < 10000000b
        jz      read_ok               ; NEW no overflow problem exists
        cli
        ; NEW
        mov     cx, word ptr es:[si]  ; NEW get BIOS timer counter again
        mov     dx, word ptr es:[si+2] ; NEW for safety reasons
        sti
        ; NEW

        ; ----- end of verification -----
    }
}

```

```
read__ok:
    mov     [bx],cx
    mov     [bx+2],dx
    mov     bx,TimerTick          ; get pointer to TimerTick
    mov     [bx],ax              ; store timer count
    pop     si
}
```

(Manuscript received October 17, 1990;
revision accepted for publication November 5, 1991.)