Behavior Research Methods, Instruments, & Computers
1988, 20 (6), 560-565

C language functions for millisecond timing on the IBM PC

JOSEPH G. DLHOPOLSKY
Computer Associates International, Garden City, New York

This article describes four C language functions for programming the IBM PC and compatibles for timing with millisecond precision. The technique, which is based on a reprogramming of the PC's real time clock, requires no additional hardware, no assembly language code, and no programming of machine or software interrupts. One function restores the PC's time-of-day clock.

Solutions to timing with millisecond precision have been presented for a number of microcomputers over the years, from the early Apple II and Radio Shack TRS-8O to the current IBM PC and Apple Macintosh. Authors have generally relied on two approaches. One involves hardware additions or modifications together with supporting software (Emerson, 1988; Grice, 1981; Poltrock & Foltz, 1982; Rayfield, 1982). The other involves assembly language or compiled high level language software timers (Adams, 1985; Bührer, Sparrer, & Weitkunat, 1987; Dlhopolsky, 1983; Graves & Bradley, 1987, 1988; Hormann & Allen, 1987; Westall, Perkey, & Chute, 1986).

Emerson (1988) reported a C language timer that uses an IBM asynchronous serial interface, which is a common add-on board. It contains a hardware timer that can be programmed to issue an interrupt to the CPU (central processing unit) at the end of a programmed interval. Depending on the application, the timer's resolution can approach 1 msec. One characteristic of the technique, according to Emerson, is that the more frequent the interrupts, the more time the CPU takes in timekeeping tasks. In other words, the better the resolution, the less time the CPU has for other functions. For example, greater resolution may be programmed for an event-counter application. But screen intensive uses, such as the tachistoscopic display of large or complex stimuli, would involve a more restrictive tradeoff between timer resolution and stimulus size in bytes (see Usage Notes below). Emerson recommended a resolution of 10 msec as a suitable middle ground for general purposes.

Bührer et al. (1987) described an assembly language module that programs one of the hardware timers on the IBM PC's Intel 8253 Programmable Interval Timer to issue an interrupt every millisecond. The authors provided other assembly language modules to be called from BASIC programs to count the interrupts and read and reset the clock buffer. While BASIC is not a good choice for a programming language for real-time laboratory software, anyone familiar with 8086 Assembly Language could readily modify the routines to be called from a fast assembly language program. A by-product of Bührer et al. technique is the disruption of the time-of-day clock, which would be a concern for those who want accurate date and time stamps for their data files.

The timer functions described in this article program the 8253 timer as in Bührer et al., (1987), but they do not directly handle the interrupt, thereby saving some CPU processing overhead. Rather, they read and write data to the 0 segment random access memory (RAM) addresses 46c and 46d hex of the IBM PC, which contain the tally of timer ticks for the normally functioning time-of-day clock software. Like Emerson's timing software, the functions are written in the C language, which, when appropriately used, produces programs that approach the execution speed of assembly language programs. Unlike Emerson's (1988) code, however, the serial interface board is not required. The functions are modularly structured and three of them are 15 lines long or shorter, including blank and comment lines. Most professional compilers provide the capability of placing the functions in permanent libraries that can then be linked with any number of application programs.

The functions have in common with Emerson's (1988) and Bührer et al.'s (1987) approaches the feature of isolating the application software from the duty of maintaining timing loops. The program initially sets the timer, then reads and writes data to the clock buffer in one-word packets. Timing occurs in parallel with the execution of the program statements, which may then be devoted more fully to CPU-intensive operations such as the displaying of large stimuli. One of the provided functions also corrects the time-of-day clock.

In normal operation, the IBM PC's 8253 Timer 0 issues an interrupt after it counts down from 65535 to 0, paced by a 1.19318-MHz time base. This results in an interrupt every 54.9255-msec. The IBM PC uses this value as the time base for updating the time-of-day clock. It keeps a running total of timer ticks in memory addresses 46c and 46d hex.

By changing the Timer 0 clock event count from 65536 to 1193, the timer will produce an interrupt every 999.849 μsec and the result will be tallied in the same

This paper was not sponsored by Computer Associates International. The author's mailing address is 27 Wilson Street, Port Jefferson Station, New York 11776.

memory locations. Then, by storing zeroes in these addresses and reading the contents at various points in an experiment trial, a program can monitor intervals with close to millisecond accuracy. This makes the timing of intervals easier and faster because elapsed time can be compared directly with a terminal time instead of having to subtract the start time from the current time before the comparison can be made.

The functions keep track of the total elapsed milliseconds between the start of timing and its termination. This provides a value that may be used to restore the time-of-day clock. If desired, the .015% timing error incurred in the 999.849-μsec millisecond may be corrected by incorporating a correction equation in the software.

The timer software was compiled on the Aztec C86 C Compiler, Developer version (Manx Software Systems, Shrewsbury, NJ). However, C commands and standard library functions are portable and will compile on any compiler that conforms to the Kernighan and Ritchie (1978) standard, or the forthcoming ANSI standard (such as Lattice C, Turbo C, Microsoft C, and many others).

Aztec C86 also provides a library of IBM-specific functions that operate more flexibly and have greater utility on the IBM PC than the standard C library. Some of these have been used in the timer software. Other C compilers have the same capabilities, with only minor differences between the analogous functions. The differences should not dissuade the interested researcher from using other compilers, because the intense competition among software houses has resulted in C compilers that almost universally produce very fast and very compact machine language code.

The listing in the Appendix contains the source code for the timer and related functions. The numbers in the right-hand column of the listing are not normal to C code, but have been included here to facilitate the description of the code's operation (in fact, the program will not compile with the numbers). The code consists of the following functions:

set_timer()
fix_time_of_day()
reset_timer()
zero_timer()
main()

In addition, the timer is implemented with one macro: GET_MSEC.

The "#include <stdio.h>" on line 001 is the first line in all C programs. The stdio.h file contains standard input/output declarations. The second line is an instruction to load the Aztec C86 time.h header file. This contains a declaration of the time-of-day data structure that is used in the set_timer() and fix_time_of_day() functions. Lines 007 and 008 declare the global elapsed_msec variable and a global data structure of type tm: start_time.

The elapsed_msec variable counts milliseconds from the initiation of the timer. This allows the time-of-day clock to be restored when the program finishes its timing phase. An unsigned long integer (4 bytes) can tally milliseconds for 49.7 days before rolling over. This is considered more than ample for most laboratory applications.

The composition of the tm structure is declared in the time.h file (see line 002). Several elements of the start_time structure are used in various places in the code. They appear in the form: start_time.tm_x (where the "x" could be hour, min, sec, etc.). Note that the Aztec functions that read and set time of day operate through IBM PC BIOS (Basic Input/Output Services) calls and are not unique to this compiler.

The GET_MSEC macro (line 005)

Whenever the GET_MSEC macro appears in the source code, the compiler will replace it with the expression:

(unsigned int)peekw(0×46c,0)

The peekw() function works like the BASIC language PEEK command. It returns the 2-byte signed integer stored at the memory address indicated by the first argument at the segment offset indicated by the second argument. In this case, it reads the clock tick value at addresses 46c and 46d hex. The returned value is cast to an unsigned integer to prevent values greater than 32767 from being interpreted as negative numbers. Experiment intervals such as interstimulus intervals may be timed by enclosing the GET_MSEC macro in a while loop. Subject response latencies may be timed by issuing the GET_MSEC macro after a response is detected. The analogous peekw() function of other compilers may have a different name and format, but it will operate in a similar fashion.

set_timer() (lines 011-026)

This function reprograms Timer 0 of the 8253 chip to time in milliseconds by giving it a new loop constant of 1193 (4a9 hex). Line 014 declares two Aztec C86 functions: Outportb() and dostime(). Outportb() is used to output 1 byte of data to an I/O port. Dostime() is an Aztec function that reads the time-of-day clock. Both functions are common in other compilers and can also be implemented with direct BIOS calls. Lines 015 and 016 declare, respectively, for the set_timer() function, the global elapsed_time variable and the start_time structure. Lines 018 through 025 contain the commands that make up the function.

Elapsed_msec is set to 0 in line 018. The current time of day is read and stored in the start_time structure (line 019). The expression "&start_time" indicates to the dostime() function the address of the start_time structure into which the dostime() function stores the current time. (The fix_time_of_day() function restores the time-of-day clock by adding the elapsed milliseconds to the start time.)

In line 020, a control word (36 hex) is output to port 43 hex, which prepares the 8253 timer to receive a new loop constant for Timer 0. The new timer constant

(4a9 hex or 1193 decimal) is then sent over port 40 hex (lines 021 and 022). Line 023 sets the timer buffer to 0. The pokew( ) function in this line is an Aztec C86 function that stores a 2-byte word, the first argument, in the memory location and segment indicated in the second and third arguments, respectively. Other compilers provide an analogous capability.

When set_timer( ) returns, the clock is timing in approximate 1-msec intervals (999.849 μsec). The clock count at addresses 46c and 46d will recycle when it reaches 65535 (slightly over 1 min). Consequently, the global elapsed_msec variable should be updated at least every minute by calls to either the reset_timer( ) or the zero_timer( ) function. If the clock count is allowed to roll over without updating elapsed_msec, 65 sec will be lost from the time-of-day clock.

Rollover counts from addresses 46c and 46d are available at addresses 46e and 46f, but setting and reading these addresses would involve additional program statements. This would involve a second peekw( ) function call every time GET_MSEC was called, doubling the processing time for this call. Considering that GET_MSEC would likely be called repeatedly during while loops, this would add considerably to the CPU processing time and might affect the precision of the timer. The current configuration was designed to optimize tachistoscopic applications, where intervals are timed in multiples of screen refresh intervals (16.7 msec) and response latencies are lower than 1 min. In such applications, the zero_timer( ) function could be called during intertrial intervals to update the elapsed_msec variable. It is unlikely that experiments having intervals longer than a minute seriously require the precision of a millisecond timer. In such cases, the overhead involved in setting and reading addresses 46e and 46f would be negligible, and the code could be modified easily to accommodate this need.

Regardless of whether or not time of day is important, the reset_timer( ) function must be called before any disk access is performed and before the program is ended. The disk drives will simply not work with the 1193 constant in Timer 0.

fix_time_of_day( ) (lines 029-084)

This function is called by the reset_timer( ) function to restore the correct time of day. It decomposes the elapsed_msec variable into hundredths of a second, seconds, minutes, and hours (lines 038-044) and adds the results to the appropriate elements of the global start_time structure (lines 047-065). In the interest of brevity and speedy execution, the updating of the clock stops at hours; the day, month, and year are simply set back to their starting values (lines 066-071 and 079-081). Therefore, an experiment starting before midnight and continuing until after midnight will result in the time's being set back to 11:59:59 at its conclusion.

The bdos( ) function (lines 077 and 081) is the Aztec C86 function that calls the MS-DOS SET_DATE and SET_TIME functions with interrupt 21 hex. The "<< 8" expression (lines 075, 076, and 080) is the C syntax for a left shift of 8 bits. It has the effect of shifting a low-order byte of a word into the high-order position. (Multiplying the low order byte by 256 has the same effect, but it is a slower operation).

reset_timer( ) (lines 087-102)

This function sets the 8253 Timer 0 back to the normal timing mode. It therefore has the opposite effect of the set_timer( ) function. It also updates elapsed_msec (line 093) and calls fix_time_of_day( ). This function must be called before the disk drives are used.

zero_timer( ) (lines 105-114)

This function sets the timer ticks at addresses 46c and 46d hex to 0. Before it does so, however, it updates the elapsed_msec variable with a call to GET_MSEC.

main( ) (lines 117-141)

The main( ) function is the one from which every C program begins its execution. In this case, it performs a simple exercise of the millisecond timer functions to test them for proper operation. It consists of a while loop that continues until the ESC key is pressed. In the course of the loop, it displays the start time, counts milliseconds between key presses, and displays the corrected time at the end of the interval. This function was used to test the timer and time-of-day correction functions. Given the variance expected from rounding errors and observer reaction time, a comparison of the results with readings from a digital stopwatch illustrated that the functions operate as expected.

The dostime( ) function puts the current time in a structure of type tm, called buffer. The asctime( ) function returns a character string representation of the time that is passed to it in the buffer structure. Scr_getc( ) causes the computer to wait for a key press and returns the character (similar to the BASIC language INKEY command). This function is in the Aztec C86 library, but is implemented as a BIOS call. Other compilers offer a similar facility. The printf( ) function is a standard C formatted print command.

Usage Notes

The timer functions were tested on IBM XT, AT, and Tandy 1000 computers and functioned as expected. They have also been incorporated into one published laboratory software package (Dlhopolsky, 1988). They may not operate correctly on some IBM PC compatibles that lack hardware compatibility. Note that the CONFIG.SYS file on the boot disk must have a line that reads: DEVICE = ANSI.SYS, and the ANSI.SYS file must be present on the disk. Memory resident programs should not be installed during the use of the timers, especially those that issue interrupts (such as mouse driver software, or programs that provide so-called "hot keys"). Users should also refrain from random or rapid keyboard entries other than those intended by the program design, as the keyboard issues an interrupt every time a key is pressed. Finally, although used in the main( ) function in this paper, the printf( ) function should not be used in tachistoscopic applications because it is too slow. Direct pokes to video memory are more appropriate.

It should be noted that the maximum size of a stimulus (in bytes) in tachistoscopic applications is governed by the number of bytes that can be moved to video RAM within the span of one video refresh cycle, which is 16.7 msec on most American monitors. The speed of transfer is dependent on the storage speed of the computer's RAM chips and on its system clock frequency. The faster the RAM chips and clock frequency, the larger the maximum stimulus size. A 3-year-old Tandy 1000 with an 8088 chip operating at 4.77 MHz can move about 128 words to video RAM in 15 msec. More recently introduced computers with faster memory chips and 80286 or 80386 microprocessors operating at up to 25 MHz can do much better. However, the various system clock frequencies should have little impact on the timer described here, because IBM has announced that it will maintain the Timer 0 frequency of 1.19318 MHz in future products, regardless of the system clock frequency (IBM, 1984, p. 9-11).

Availability

Readers may acquire an MS-DOS compatible disk copy of the source code and executable program by sending $7 to the author at 27 Wilson Street, Port Jefferson Station, New York 11776, or by sending a formatted disk and $2 to cover postage.

REFERENCES

Adams, J. K. (1985). Visually presented verbal stimuli by assembly language on the Apple II computer. *Behavior Research Methods, Instruments, & Computers*, **17**, 489-502.

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.

Dlhopolsky, J. G. (1983). Machine language millisecond timers for the Z-80 microprocessor. *Behavior Research Methods & Instrumentation*, **15**, 511-520.

Dlhopolsky, J. G. (1988). *PC-HEMIP: Hemispheric information processing on the IBM PC.* Bayport, NY: Life Sciences Associates.

Emerson, P. L. (1988). 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.

Graves, R., & Bradley, R. (1988). More on millisecond timing and tachistoscope applications for the IBM PC. *Behavior Research Methods, Instruments, & Computers*, **20**, 408-412.

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

Hormann, C. A., & Allen, J. D. (1987). An accurate millisecond timer for the Commodore 64 or 128. *Behavior Research Methods, Instruments, & Computers*, **19**, 36-41.

IBM Corp. (1984). *Technical reference: Personal computer AT*, Part 1502243. Boca Raton, FL: Author.

Kernighan, B. W., & Ritchie, D. M. (1978). *The C Programming Language*. Englewood Cliffs, NJ: Prentice-Hall.

Poltrock, S. E., & Foltz, G. S. (1982). An experimental psychology laboratory system for the Apple II microcomputer. *Behavior Research Methods & Instrumentation*, **14**, 103-108.

Rayfield, F. (1982). Experimental control and data acquisition with BASIC in the Apple computer. *Behavior Research Methods & Instrumentation*, **14**, 409-411.

Westall, R., Perkey, M. N., & Chute, D. L. (1986). Accurate millisecond timing on Apple's Macintosh using Drexel's MilliTimer. *Behavior Research Methods, Instruments, & Computers*, **18**, 307-311.

APPENDIX
Listing of Program

```
#include <stdio.h>
#include <time.h>

#define ESC        0x1b      /* value of the escape key               */
#define GET_MSEC (unsigned int)peekw(0x046c,0)

unsigned long int elapsed_msec;
struct tm start_time;


void set_timer()      /* sets timer 0 to time in milliseconds         */
                      /* normal clock tick is 55 msec                 */
{
   int outportb(), dostime();
   extern unsigned long int elapsed_msec;
   extern struct tm start_time;

   elapsed_msec = 0;
   dostime(&start_time);            /* reads clock at start              */
   outportb (0x43,0x36);            /* 8253 timer control word           */
   outportb (0x40,0xa9);            /* new timer constant for 1.19318 MHz */
```

APPENDIX (Continued)

```
   outportb (0x40,0x04);          /* 4a9 hex = 1193                    */
   pokew(0x046c,0,0);             /* zero timer so elapsed msec = 0    */

   return;
}


void fix_time_of_day()
{
   int seconds, minutes, hours;
   unsigned int hour_min, second, month_day;
   unsigned long int hsec;     /* hundredths of a second             */
   extern unsigned long int elapsed_msec;
   extern struct tm start_time;

   /* Get elapsed hsec, seconds, minutes, hours */
   hsec = elapsed_msec / 10;
   hours = hsec / 360000;
   hsec %= 360000;                         /* %= is modulus operator   */
   minutes = hsec / 6000;
   hsec %= 6000;
   seconds = hsec / 100;
   hsec %= 100;

   /* Add elapsed time to start time  */
   start_time.tm_hsec += hsec;
   if (start_time.tm_hsec >= 100)
   {
      start_time.tm_sec++;
      start_time.tm_hsec -= 100;
   }
   start_time.tm_sec += seconds;
   if (start_time.tm_sec > 59)
   {
      start_time.tm_min++;
      start_time.tm_sec -= 60;
   }
   start_time.tm_min += minutes;
   if (start_time.tm_min > 59)
   {
      start_time.tm_hour++;
      start_time.tm_min -= 60;
   }
   start_time.tm_hour += hours;
   if (start_time.tm_hour > 23)    /* don't turn over to next day      */
   {
      start_time.tm_hour = 23;
      start_time.tm_min = 59;
      start_time.tm_sec = 59;
      start_time.tm_hsec = 99;
   }

   /* put new time value in clock  '<<' is left shift operator         */
   second = ((int)start_time.tm_sec << 8) + start_time.tm_hsec;
   hour_min = ((int)start_time.tm_hour << 8) + start_time.tm_min;
   bdos(0x2d, second, hour_min);  /* IBM function 2d sets time         */

   /* put start date back (in case it changed) */
   month_day  = (start_time.tm_mon << 8) + start_time.tm_mday;
   bdos(0x2b, month_day, start_time.tm_year);

   return;
}
```

APPENDIX (Continued)

```
void reset_timer()     /* Puts timer back to normal                            */
{
   void fix_time_of_day();
   int outportb();
   extern unsigned long int elapsed_msec;

   elapsed_msec += GET_MSEC;  /* tally last few milliseconds                    */

   /* Put timer back to normal */
   outportb (0x43,0x36);       /* 8253 control word                             */
   outportb (0x40,0);          /* old timer 0 constant = 0                      */
   outportb (0x40,0);
   fix_time_of_day();    /* Go to fix time of day                               */

   return;
}


void zero_timer()   /* set timer to 0 and sum elapsed milliseconds              */
{
   int pokew();
   extern unsigned long int elapsed_msec;

   elapsed_msec += GET_MSEC;  /* Keep track of elapsed milliseconds             */
   pokew(0x046c,0,0);

   return;
}


main()
{
   void set_timer(), reset_timer(), zero_timer();
   char *asctime(), scr_getc(), key_press;
   int printf(), dostime();
   unsigned int msec;
   struct tm buffer;

   while (key_press != ESC)    /* ESC key stops the test.                      */
   {
      printf("Press a key to start the timer; press again to stop.\n");
      key_press = scr_getc();          /* wait for key press to start          */
      set_timer();
      if (key_press == ESC) break;
      dostime(&buffer);
      printf("Start: %s\n", asctime(&buffer));
      key_press = scr_getc();          /* wait for key press to stop           */
      msec = GET_MSEC;
      printf("%u\n", msec);
      dostime(&buffer);
      printf("Stop:  %s\n\n", asctime(&buffer));
      reset_timer();
   }
   reset_timer();
}
```

(Manuscript received August 12, 1988;
revision accepted for publication October 15, 1988.)