*Behavior Research Methods, Instruments, & Computers*
*1986, 18 (3), 307-311*

# COMPUTER TECHNOLOGY

# Accurate millisecond timing on Apple's Macintosh using Drexel's MilliTimer

ROBERT WESTALL, M. NADINE PERKEY, and DOUGLAS L. CHUTE
*Drexel University, Philadelphia, Pennsylvania*

Many of the timed functions that concern psychologists, such as perceptual presentations and reaction time, are sensitive to a maximum variability in display timing caused by screen-refresh characteristics. For the Apple Macintosh, the screen operating speed is 60 Hz, which translates to an average of 8.33-msec variability. For microcomputers other than the Macintosh, a variety of hardware and software modifications to generate millisecond timing have become standard (e.g., Reed, 1979). Other than Reed College's (1985) implementation in Rascal, which requires the Rascal development language, there has been no method of which we were aware to synchronize experimental timing with display presentation on the Macintosh. This limitation in the usefulness of the Macintosh as an otherwise excellent research tool can be overcome using Drexel University's MilliTimer. The assembler code which follows should be considered in the public domain and can be readily adapted to any of the Macintosh-based languages.

Although in many respects the Apple Macintosh's high-resolution graphics, ease of editing, and convenient user interface make the Macintosh an ideal research tool, one major limitation from a psychologist's point of view has been the lack of millisecond timing. The essence of this problem has been described by Reed (1979), and his solutions for personal computers other than the Macintosh have become the standard. Essentially a synchronization is required between the timing in the experimental control program and the videoscan. Such synchronization removes an average error of 8.33 msec between the registration of a timed experimental event and the display of a stimulus on the screen. Unfortunately for the Macintosh, the fact that the development libraries provided by Apple do not allow direct access to a timer with millisecond accuracy seemed to preclude the use of this machine to run research-grade software. Reed College (1985) addressed the problem in their Rascal development language.

We have developed a MilliTimer routine (Perkey, 1986) that provides millisecond timing for a number of psychological applications (Chute & Daniel, 1985), including testing motor skills, testing pitch discrimination, measuring reaction time, testing hemispheric specialization of function, and solving problems (Chute, 1986).

MilliTimer is a small library that consists of two routines to assist Macintosh programmers in gaining access to a millisecond timer. The MilliTimer listed here is interrupt-driven off of timer no. 1 on the Synertek 6522 Versatile Interface Adapter (VIA) chip that is part of the standard Macintosh configuration. MilliTimer uses the 16-bit timer no. 1 in the free-run mode so that the timer will run continuously, generating a level-1 interrupt in the processor each time it expires.

The program listing in Appendix A was written and assembled using the Lisa cross-development system, Version 3.9 (created by Apple Computer, 1985). No matter which of the development systems for Macintosh is used, the fundamental 68000 code will be the same. Appendix B provides a listing for the Macintosh Development System (MDS). However, minor syntactic changes may be needed, depending upon the version or the Macintosh assembly development system used. The code can be easily modified to adapt to any of the Macintosh-based languages. The Pascal interface for the MilliTimer is given as:

```
Procedure MilliControl (CtrlFlag: Boolean);
Function MilliCount : LongInt;
```

The first of these routines, **MilliControl**, is used to turn the MilliTimer on and off. Passing **TRUE** as the parameter causes the MilliTimer to be installed into the system heap and starts it counting milliseconds, starting from zero. Normally this call is made once at the beginning of a program. Passing **FALSE** as the parameter turns off the MilliTimer and removes it from the system heap. This normally is called only once before the program is exited. The MilliTimer must be turned off before exiting, or it will continue to operate as long as no other process attempts to use the VIA timer no. 1.

Calling **MilliCount** returns the number of elapsed mil-

Software developments were supported in part by a grant from the Pew Memorial Trust to Drexel University and to Douglas Chute. The first two authors are with the Software Development Group, and the third author is with the Department of Neuropsychology, all at Drexel University. Send reprint requests to D. L. Chute, Department of Neuropsychology, Drexel University, Philadelphia, PA 19104.

 Copyright 1986 Psychonomic Society, Inc.

liseconds since **MilliControl** was last called. This function can be used like the toolbox call, **TickCount**. For example,

**MilliControl** (True); [Called at beginning of program]
MyCounter: = **MilliCount** ; [Called for as many timing events as experimentally required]
[Do other processing here]
MyCounter: = **MilliCount** - MyCounter;
**MilliControl** (False); [Called at exit of program]

Synchronizing the MilliTimer with the screen refresh can be accomplished by synchronizing to **TickCount**, which is automatically incremented during the vertical retrace interrupt. This is useful when using a visual cue to elicit a timed response. For example,

**MilliControl** (True);
MyLongInt: = **TickCount** + 1;
While MyLongInt > **TickCount** Do;
MyCounter: = **MilliCount** ; [Can be repeated as often as required for timing different events]
[Do other processing here]
MyLongInt: = **TickCount** + 1;
While MyLongInt >**TickCount** Do;
MyCounter: = **MilliCount** - MyCounter;
**MilliControl** (False);

Making continuous calls to **MilliControl** does not cause problems, since each call causes the previous MilliTimer to be removed from the system heap and a new one to be installed. This has the effect of resetting the counter.

It is important to remember that any other process or driver that uses the no. 1 timer on the 6522 chip (such as the sound driver) will step on the MilliTimer, making it necessary to reinitialize the counter with a call to **MilliControl**. Also, during timing of some user action, no disk access can take place during the timing process, as the time reported back by **MilliControl** is inflated by the disk access time. Testing has shown the accuracy of the MilliTimer to be approximately $\pm 1$ msec per 30 min. Appendix C provides a working demonstration applicable to Microsoft BASIC routines. Users of this routine are cautioned, however, that the processing times of interpreted languages are potentially a source of timing error in the millisecond range.

## REFERENCES


CHUTE, D. L. (1986). MacLaboratory for psychology: General experimental psychology with Apple's Macintosh. *Behavior Research Methods, Instruments, & Computers*, **18**, 205-209.

CHUTE, D. L., & DANIEL, R. S. (1985). *MacLaboratory for psychology*. Dubuque: Kendall Hunt.

PERKEY, M. N. (1986). The effect of a machine-rich environment on courseware development: The process and the product. *Behavior Research Methods, Instruments, & Computers*, **18**, 196-204.

REED, A. V. (1979). Microcomputer display timing: Problems and solutions. *Behavior Research Methods & Instrumentation*, **11**, 572-576.

REED COLLEGE. (1985). *Rascal user manual: Macintosh language for real time I/O oriented development*. Portland, OR: Metaresearch.


## APPENDIX A

```
.NOLIST
.INCLUDE    TIAsm/SysTraps.Text
.INCLUDE    TIAsm/SysEqu.Text
.LIST

.FUNC       MilliCount
MOVE.L      (SP)+,A1            ;Save return address in A1
LEA         Lv1IDT,A0           ;Load the address of the level 1 dispatch table into A0
MOVEA.L     24(A0),A0           ;Place the timer 1 interrupt vector into A0
MOVE.L      -8(A0),(SP)         ;Make the function result the millisecond count
JMP         (A1)                ;Return to the calling program

.PROC       MilliControl
MOVE.L      A2,-(SP)            ;Save A2 on the stack
MOVEA.L     VIA,A2              ;Move the base address of the VIA into A2
MOVE.B      #01000000B,VIER(A2) ;Turn off the timer 1 interrupts
MOVE.B      VT1C(A2),D0         ;Clear the VIFR bit for timer 1, just in case
MOVEA.L     SysZone,A0          ;Load the address of the system heap into A0
MOVEA.L     ApplZone,A1         ;Load the address of the application heap into A1

@0 LEA      MilliInterrupt,A2   ;Load the address of the interrupt routine into A2
   CMPA.L   A0,A1               ;Have we reached the end of the system heap?
   BLE.S    @1                  ;Yes, we're done searching for the interrupt
   CMPM.W   (A0)+,(A2)+         ;Compare a word in the heap with first word of the interrupt

   BNE.S    @0                  ;Not equal, keep searching
   MOVEQ    #<7-1>,D0           ;Load D0 for looping
@2 CMPM.L   (A0)+,(A2)+         ;Compare another longword in the heap with the interrupt
   DBNE     D0,@2               ;Leave if not equal otherwise loop 7 times
   BEQ.S    @3                  ;We found the interrupt - make a short branch
   SUBQ.L   #4,A0               ;Fix A0
   BRA.S    @0                  ;Go back and keep searching the heap
@3 SUBA.L   #<8+30>,A0          ;Set A0 to point to the interrupt in the system heap
   CLR.L    8(A0)               ;Wipe out the first 4 bytes of the interrupt
   _DisposPtr ,SYS              ;De-allocate the 38 bytes
```

## APPENDIX A (Continued)

```
@1 CMPI.W    #0,8(SP)                ;Test the boolean on the stack
   BEQ.S     @4                      ;If FALSE then branch around initialization code
   MOVEQ     #<30+8>,D0              ;We need 38 bytes of system heap space
   _NewPtr   ,SYS,CLEAR              ;Get some memory!
   TST.W     D0                      ;Everything O.K.?
   BNE.S     @4                      ;No, we can't do anything more
   ADDQ.L    #8,A0                   ;Leave 8 bytes for the millicounters
   MOVEA.L   VIA,A2                  ;Move the base address of the VIA into A2
   ORI.B     #01000000B,VACR(A2)     ;Set bit 6 of the VACR - timer 1 free run mode
   AND1.B    #01111111B,VACR(A2)     ;Clear bit 7 of the VACR - pulse disabled
   LEA       Lv1IDT,A1               ;Load the address of the level 1 dispatch table into A1
   MOVE.L    A0,24(A1)               ;Set up the new timer 1 interrupt vector
   MOVE.B    #$08,VT1C(A2)           ;Load low byte of counter into the timer 1 latch
   MOVE.B    #$03,VT1CH(A2)          ;Load high byte of timer 1 counter, start counting...
   MOVEQ     #<15-1>,D0              ;Load D0 for looping
   LEA       MilliInterrupt,A1       ;Load the address of the interrupt routine into A1
@5 MOVE.W    (A1)+,(A0)+             ;Move a word of the interrupt into the heap
   DBRA      D0,@5                   ;Loop 15 times
   MOVE.B    #11000000B,VIER(A2)     ;Turn on timer 1 interrupts
@4 MOVE.L    (SP)+,A2                ;Restore A2
   MOVE.L    (SP)+,A1                ;Place return address in A1
   ADDQ.L    #2,SP                   ;Move stack pointer above the boolean
   JMP       (A1)                    ;Return to the calling program

   MilliInterrupt

   MOVE.B    VT1C(A1),D0             ;Clear VIFR bit for timer 1 on the VIA
   LEA       *-12,A0                 ;Set A0 to point to the millisecond counter
   ADDQ.L    #1,(A0)+                ;Increment the millisecond counter by 1
   ADDQ.L    #1,(A0)                 ;Increment the background counter by 1
   CMPI.L    #1800,(A0)              ;Have 1800 milliseconds elapsed?
   BMI.S     @0                      ;No, jump around error correction code (next 2 instrs.)
   CLR.L     (A0)                    ;Clear the background timer
   SUBQ.L    #1,-4(A0)               ;Subtract 1 from the millisecond timer
@0 RTS                               ;Return to the caller

.END
```

## APPENDIX B

```
0000    ;******
0000    ;****** File: MilliTimerAsm.Txt
0000    ;****** Language: MDS Assembler v1.0
0000    ;****** Author: Robert Westall, Software Development Group,
0000    ;******  Drexel University
0000    ;******
0000
0000                          INCLUDE   MacTraps.D
0000                          INCLUDE   SysEquX.D
0000
0000                          XDEF      MilliCount
0000                          XDEF      MilliControl
0000
0000                    MilliCount
0000    226F 0004             MOVE.L    4(SP),A1
0004    41F8 0192             LEA       Lv1IDT,A0
0008    2068 0018             MOVEA.L   24(A0),A0
000C    32A8 FFFA             MOVE.W    -6(A0),(A1)
0010    2257                  MOVE.L    (SP),A1
0012    508F                  ADDQ.L    #8,SP
0014    4ED1                  JMP       (A1)
0016
0016                    MilliControl
0016    2F0A                  MOVE.L    A2,-(SP)
0018    2478 01D4             MOVEA.L   VIA,A2
001C    157C 0040 1C00        MOVE.B    #%01000000,VIER(A2)
0022    102A 0800             MOVE.B    VT1C(A2),D0
0026    2078 02A6             MOVEA.L   SysZone,A0
002A    2278 02AA             MOVEA.L   ApplZone,A1
```

## APPENDIX B (Continued)

```
002E
002E    45FA 0074       @0:  LEA       MilliInterrupt,A2
0032    B3C8                 CMPA.L    A0,A1
0036    6F1E                 BLE.S     @1
0036    B548                 CMPM.W    (A0)+,(A2)+
0038    66F4                 BNE.S     @0
003A    7006                 MOVEQ     #6,D0
003C    B588            @2:  CMPM.L    (A0)+,(A2)+
003E    56C8 FFFC            DBNE      D0,@2
0044    6704                 BEQ.S     @3
0044    5988                 SUBQ.L    #4,A0
0046    60E6                 BRA.S     @0
0048    91FC 0000 0026  @3:  SUBA.L    #38,A0
004E    42A8 0008            CLR.L     8(A0)
0052    A41F                 _DisposPtr ,SYS
0054
0054    0C6F 0000 0008  @1:  CMPI.W    #0,8(SP)
005C    6740                 BEQ.S     @4
005C    7026                 MOVEQ     #38,D0
005E    A51E                 _NewPtr   ,SYS,CLEAR
0060    4A40                 TST.W     D0
0064    6638                 BNE.S     @4
0064    5088                 ADDQ.L    #8,A0
0066    2478 01D4            MOVEA.L   VIA,A2
006A    002A 0040 1600       ORI.B     #%01000000,VACR(A2)
0070    022A 007F 1600       ANDI.B    #%01111111,VACR(A2)
0076    43F8 0192            LEA       Lvl1DT,A1
007A    2348 0018            MOVE.L    A0,24(A1)
007E    157C 000B 0800       MOVE.B    #$0B,VT1C(A2)
0084    157C 0003 0A00       MOVE.B    #$03,VT1CH(A2)
008A
008A    700E                 MOVEQ     #14,D0
008C    43FA 0016            LEA       MilliInterrupt,A1
0090    30D9            @5:  MOVE.W    (A1)+,(A0)+
0092    51C8 FFFC            DBRA      D0,@5
0096    157C 00C0 1C00       MOVE.B    #%11000000,VIER(A2)
009C    245F            @4:  MOVE.L    (SP)+,A2
009E    225F                 MOVE.L    (SP)+,A1
00A0    548F                 ADDQ.L    #2,SP
00A2    4ED1                 JMP       (A1)
00A4
00A4                      MilliInterrupt
00A4    1029 0800            MOVE.B    VT1C(A1),D0
00A8    41FA FFF2            LEA       *-12,A0
00AC    5298                 ADDQ.L    #1,(A0)+
00AE    5290                 ADDQ.L    #1,(A0)
00B0    0C90 0000 0708       CMPI.L    #1800,(A0)
00B8    6B06                 BMI.S     @6
00B8    4290                 CLR.L     (A0)
00BA    53A8 FFFC            SUBQ.L    #1,-4(A0)
00BE    4E75            @6:  RTS
```

## APPENDIX C

```
REM *** This program illustrates the use of the Drexel MilliTimer
REM *** Language:  Microsoft Basic v2.1
REM *** Author:  Robert Westall, Software Development Group
REM ***          Drexel University, Philadelphia, PA  19104
REM *** To avoid handling single-precision mathematics,  the MilliCount
REM *** function has been modified to return an integer (2 bytes).  Easy
REM *** modification could be made to allow the return of the full 4
REM *** bytes of the counter for single-precision or 8 bytes for double-
REM *** precision.

REM *** Install the MilliTimer routines
DIM CountCode%(11), ControlCode%(85)
True% = 1: False% = 0: MyTime% = 0: NewTime% = 0
CLS
```

**APPENDIX C (Continued)**

```
FOR Kount = 0 TO 10: READ CountCode% (Kount): NEXT Kount
FOR Kount = 0 TO 84: READ ControlCode% (Kount): NEXT Kount

REM *** Initialize the MilliTimer
  MilliControl = VARPTR (ControlCode%(0))
  CALL MilliControl (True%)

REM *** Time event
  MilliCount = VARPTR (CountCode%(0))
  CALL MilliCount (VARPTR(MyTime%))
  REM {***Do other processing here***}
  CALL MilliCount (VARPTR(NewTime%))
  PRINT "The elapsed time in milliseconds is "; NewTime%-MyTime%

REM *** Shutdown MilliTimer
  MilliControl = VARPTR (ControlCode%(0))
  CALL MilliControl (False%)

REM *** Machine code data for "MilliCount"
DATA &H226F,&H0004,&H41F8,&H0192,&H2068,&H0018,&H32A8,&HFFFA
DATA &H2257,&H508F,&H4ED1

REM ***Machine code data for "MilliControl"
DATA &H2F0A,&H2478,&H01D4,&H157C,&H0040,&H1C00,&H102A,&H0800
DATA &H2078,&H02A6,&H2278,&H02AA,&H45FA,&H0074,&HB3C8,&H6F1E
DATA &HB548,&H66F4,&H7006,&HB588,&H56C8,&HFFFC,&H6704,&H5988
DATA &H60E6,&H91FC,&H0000,&H0026,&H42A8,&H0008,&HA41F,&H0C6F
DATA &H0000,&H0008,&H6740,&H7026,&HA51E,&H4A40,&H6638,&H5088
DATA &H2478,&H01D4,&H002A,&H0040,&H1600,&H022A,&H007F,&H1600
DATA &H43F8,&H0192,&H2348,&H0018,&H157C,&H000B,&H0800,&H157C
DATA &H0003,&H0A00,&H700E,&H43FA,&H0016,&H30D9,&H51C8,&HFFFC
DATA &H157C,&H00C0,&H1C00,&H245F,&H225F,&H548F,&H4ED1,&H1029
DATA &H0800,&H41FA,&HFFF2,&H5298,&H5290,&H0C90,&H0000,&H0708
DATA &H6B06,&H4290,&H53A8,&HFFFC,&H4E75
```

(Manuscript received February 12, 1986;
revision accepted for publication May 1, 1986.)