SESSION VIII

MICROCOMPUTER RESEARCH TOOLS

Howard Kaplan, *Chair*

Software timing of events in cognitive psychology experiments

DAVID DORFMAN

Villanova University, Villanova, Pennsylvania

(Student Award for 1986)

Real-time systems for controlling cognitive psychology experiments typically use hardware clocks. However, systems using software clocks have comparable reliability, accuracy, and resolution, and the advantages of lower cost and hardware simplicity. Four types of software clocks are described and evaluated. Problems in implementing software clock systems are described, along with their solutions. Software clock routines for Apple II series microcomputers are presented.

An essential element of any microcomputer-based system for the real-time control of cognitive psychology experiments is some means of achieving precise timings of events such as display durations or subjects' response times. Because of the nature of current theorizing, it is desirable that such timings be accurate to the millisecond. There are two basic ways of achieving this end: (1) a *software clock*, that is, software routines ultimately dependent on the microcomputer's master clock, and (2) a *hardware clock*, that is, a dedicated device or chip.

The present paper is concerned with issues relevant to the use of software clock systems for controlling real-time cognitive psychology experiments. Although the paper focuses on the Apple II series computers, the underlying principles apply to systems implemented on other microcomputers.

Typically, systems for running cognitive psychology experiments on Apple II computers, for example, APT (Foltz & Poltrock, 1981) and ApplePsych (Osgood, 1984a, 1984b), have used a hardware clock. Even systems that do not in principle require a hardware clock, such as that described by Adams (1985), assume a hardware clock in their implementations. The only notable exception to this trend is a FORTH-based real-time system described by Watts (1986), which uses a software clock.

Appropriately implemented software clocks are comparable to hardware clocks in reliability, accuracy, and resolution. Since they cost nothing and do not increase hardware complexity, the use of software clocks would appear to be the preferable approach except in those circumstances requiring the special capabilities possible in a hardware clock system.

Besides considerations of cost and hardware simplicity, noted above, software clock systems avoid two problems specific to hardware clock systems. First, there is the possibility that the specific hardware needed for the system will not be readily available. For example, the Mountain clock, used on many systems for Apple II series microcomputers, is no longer manufactured. Second, a hardware clock cannot be used on some microcomputers, such as the Apple IIc, which has no expansion slots.

Systems using hardware clocks do have one advantage: If appropriately designed, they can be programmed to perform some additional task in parallel with the timed event. Systems using software clocks do not have this capability, although in some instances it is possible to program around this limitation.

Two points should be made concerning parallel programming capability. First, some systems using hardware clocks (e.g., the APT system) do not have this capability. Second, in terms of the procedures currently employed in cognitive research, the need for this capability is probably rare. DeSantis and Dorfman (1987) sampled 45 articles from the 1985 volumes of three important jour-

In preparing this paper I have benefited from numerous discussions with Brian Watts and from his detailed comments on several earlier drafts. I have also benefited from the comments of T. C. Toppino on an earlier draft. Reprints are available from David Dorfman, Department of Psychology, Villanova University, Villanova, PA 19085.

nals publishing experimental reports in cognition and related fields (15 articles were sampled from each of the following: *Journal of Experimental Psychology: Human Learning, Memory, & Cognition*; *Journal of Memory & Language*; *Memory & Cognition*). These articles reported a total of 136 experiments, 80 of which used computers. Of these 80 studies, 6 were excluded from consideration because their procedures did not lend themselves to a general-purpose microcomputer, irrespective of what type of clock was employed. Of the remaining 74 studies, all could have been performed with a system using a software clock.

METHODS FOR IMPLEMENTING SOFTWARE CLOCKS

A software clock has three basic components: (1) some piece of code that takes up a uniform amount of time and is repetitively executed (each repetition constitutes a "tick"); (2) a counter that keeps track of the number of ticks; and (3) a stopping condition. There are two basic types of stopping conditions: the ending of some predetermined interval (an *interval timer*) and the sensing of some event (an *event timer*). The two types of timers have obviously different functions in terms of experimental control. For example, an interval timer would be used to turn on a display for a duration determined by the experimenter; an event timer would be used to measure the time a subject takes to respond to a stimulus.

Interval timers and event timers have different programming logic. In pseudocode, an example of an interval timer would be the following:

```
set TICKOUNT = TO_BE_TIMED_INTERVAL
begin
  decrement TICKOUNT;
until TICKOUNT = 0;
```

An example of an event timer in pseudocode would be:

```
set TICKOUNT = 0;
begin
  increment TICKOUNT;
until EVENT = OVER;
set ELAPSED_TIME = TICKOUNT;
```

In each example above, the tick is the loop that changes the value of the counter, TICKOUNT. It should be noted that a tick does not necessarily equal some standard unit of time, such as a millisecond. Since each tick is uniform, the number of ticks is sufficient for data analysis (Price, 1979). However, the results of different experiments cannot readily be compared unless they are converted to real-time values by multiplying the number of ticks by the length of time occupied by each tick.

Four methods of programming software clocks lend themselves to the Apple II series. The fundamental difference between these methods is the manner in which the tick is created.

Method 1

A software clock can be created by executing a delay loop in a higher level language such as BASIC (Rosner, 1978). Although this method is convenient, it suffers problems in coarseness of resolution and timing variability and thus is of limited usefulness (Dihopolsky, 1983; Price, 1979). In general, it would appear to be difficult to write a precise software clock in a higher level language; Methods 2, 3, and 4 should be written in assembly language. However, once an assembly language clock routine is written, it can be used in programs written in a higher level language, as will be explained below. Thus it is not necessary for end-users of software clock routines to know about assembly language programming. The difficulty in using higher level languages is not a characteristic of software routines in particular; it is characteristic of real-time routines in general.

Method 2

This method uses the periodicity of the video display unit's scan as the tick. The video display is constantly being redrawn in a zigzag pattern from top to bottom. At the end of a scan, the electronic beam must be repositioned to the upper left-hand corner of the screen. The period during which the beam is being repositioned is called the *vertical blanking interval*; it occurs every 16.67 msec in real-time values and lasts for about 4 msec.

The method is easily implemented on the Apple IIe and IIC, since memory location 49177 (\$C019), generally referred to as VERTBLANK, can be examined to determine whether a vertical blanking interval is occurring.¹ Specifically, the value of VERTBLANK exceeds 127 only during a vertical blanking interval. A pseudocode example of an interval timer using this method would be:

```
set TICKOUNT = TO_BE_TIMED_INTERVAL
begin
  if VERTBLANK > 127
    wait until VERTBLANK ≤ 127;
    then decrement TICKOUNT;
until TICKOUNT = 0
```

Notice that it is necessary to wait for the end of the 4-msec vertical blanking interval before counting the tick. If this were not done, the routine would recheck VERTBLANK many times during a single vertical blanking interval, each time incorrectly decrementing TICKOUNT.

The method is accurate and reliable. It is particularly appropriate for timing display durations and has sufficient precision to allow the Apple to be used as a tachistoscope. It has one obvious limitation: since each tick is 16.67 msec, Method 2 does not have sufficient resolution to be used as the basis for an event timer, because the possible range of error is 0–33.33 msec. However, for interval timers, the limited resolution is relatively unimportant because it does not affect the precision of the timing. It simply means that the interval to be timed must be a multiple of 16.67 msec. In particular, it should

be emphasized that with respect to timing display durations, better resolution cannot be obtained with any other type of clock, since the limiting factor is not the resolution of the clock, but the periodicity of the vertical blanking interval. It is not possible to set up a display for an interval that is not some multiple of 16.67 msec.

It is not possible to determine when a vertical blanking interval is occurring on the earlier Apple II and II+ without a minor hardware modification. Such a modification is described by Reed (1979). With Reed's suggested modification, location 49251 (\$C063) would be substituted for VERTBLANK in the routine presented above. (Following then-current usage, Reed refers to this location as SW3 and as -16285.) An alternative modification described by Mitterer and Osborne (1982) results in parallel programming capability; however, their modification is more difficult to implement and use.

Cavanagh and Anstis (1980) describe routines for using the vertical blanking interval to control display durations. Their routines assume an Apple II+ with Reed's modification. To use their routines on the IIe and IIC, VERTBLANK should be substituted for SW3. A further example is presented in Appendix A. Additional information concerning the vertical blanking interval in particular, and the operation of Apple display routines in general, will be found in Chapter 7 of Little (1985).

Method 3

This method has sufficient resolution and accuracy to time subjects' responses. It makes use of the Monitor subroutine WAIT located at 64580 (\$FCA8), which, as its name implies, "waits" for a set period of time.² The length of the period depends on the value in the 6502 microprocessor's accumulator when the subroutine is called. Specifically, the subroutine waits for $(13 + 13.5A + 2.5A^2)/1023$ msec, where A equals the value in the accumulator. Thus it is possible to adjust the size of the tick from about .03 to 160 msec. Although the delay generated by WAIT gives a reasonable approximate value for the length of the tick, many users, for reasons explained below, will find it desirable to independently calibrate clocks using this method.

A pseudocode example of a routine designed to measure a subject's response time would be:

```
set TICKOUNT = 0;
begin
  call SUBROUTINE.WAIT;
  increment TICKOUNT;
until SUBROUTINE.CHECK_FOR_RESPONSE = TRUE;
set RESPONSE_TIME = TICKOUNT;
```

An interval timer using this method is shown in Appendix B; an event timer is shown in Appendix C. Price (1979, 1986) presents a clock conceptually identical to this method, although it does not make use of the Monitor WAIT subroutine. Further information on WAIT can be found in Scanlon (1982).

Method 4

The fourth method is based on the fact that any repetitively executed uniform piece of code can be used to create a tick. Thus a routine designed to accomplish a specific task, such as determining whether a subject has responded, can be programmed to time itself. This is illustrated by the following pseudocode example designed to measure a subject's response time:

```
set TICKOUNT = 0;
begin
  increment TICKOUNT;
until SUBROUTINE.CHECK_FOR_RESPONSE = TRUE;
set RESPONSE_TIME = TICKOUNT;
```

As can be seen, Method 4 is similar to Method 3, with SUBROUTINE.CHECK_FOR_RESPONSE performing the function of the WAIT subroutine. Like Method 3, Method 4 has sufficient accuracy and resolution to time subjects' responses. Method 3 has the advantage of greater flexibility in setting tick size. Also, because ticks under Method 4 tend to be brief (around .002-.005 msec), the counter may be slightly less convenient to program and may take up a bit more space. On the other hand, none of these considerations are major and under some circumstances this method might be more convenient.

COUNTERS

As has been indicated, some form of counter is an essential component of a software clock. A counter consists of one or more locations in memory that are incremented (or decremented) when a tick occurs. Each location in the Apple is 1 byte (8 bits); thus a counter occupying a single location cannot "count" higher than 255, because the capacity of a counter is $(2^B)^n - 1$, where B = the number of bits at each location and n = the number of locations. Although such a capacity is generally adequate for interval timers, it is not adequate for event timers, particularly under Method 4, which tends to have a large number of ticks. In general, 2 bytes suffice to record response times under Method 3, and 3 bytes suffice for Method 4. However, larger counters aid in calibration of the clock, as will be explained below. Thus Methods 3 and 4 can profitably use three and four counters, respectively.

INTERFACING ASSEMBLER ROUTINES WITH HIGHER LEVEL LANGUAGES

As indicated earlier, although software clocks should be programmed in assembly language, they can be utilized in programs written in a higher level language. A number of methods exist for accomplishing this (see Watts, 1986, for a discussion of this issue). One of the easiest (and most widely used) methods is to use a language having a command that calls a machine language

subroutine. Thus most of the program can be written in the higher level language, with the time-critical portions written in assembler and called as needed. It is, of course, not necessary in this context to understand the assembler programs to make effective use of them.

This approach readily lends itself to Applesoft BASIC and is used in a number of systems (e.g., Dorfman, 1984; Price, 1979). Subroutines are invoked by the CALL command. Parameters are passed to the subroutine by the POKE and retrieved by the PEEK command. The routines can be loaded by the DOS BLOAD command or by the Applesoft POKE command.

An example of using an interval timer with this method would be:

```
POKE NTICKS,DELAY
CALL INTERVAL_TIMER
```

In the above example, the variables are NTICKS, the address of the location holding the number of ticks the timer will run; DELAY, the number of ticks; and INTERVAL_TIMER, the address of the interval timer routine. An example of this procedure to time a subject's response would be:

```
CALL RESPONSE_MEASURE
TICK_TOTAL = PEEK*(LOCATION1)
+ 256*(PEEK(LOCATION2))
+ 65536*(PEEK(LOCATION3))
```

In the above example, the variables are TICK_TOTAL, elapsed ticks; RESPONSE_MEASURE, the address of the routine sensing and timing the subject's response; LOCATION1, the address of the location recording how many ones of ticks have elapsed; LOCATION2, the address of the location holding how many times 256 ticks have elapsed (i.e., how many times LOCATION1 has recycled); and LOCATION3, the address of how many times 65536 ticks have elapsed (i.e., how many times LOCATION2 has recycled). The routines can be loaded by the DOS BLOAD command or by the Applesoft POKE command. Price (1979) and Cavanagh and Anstis (1980) present further examples of this method. Some users may wish to substitute the ampersand command for the CALL command. The advantage is that while CALL takes around 3.5 msec to execute, ampersand takes about .3 msec. A clear explanation of how to use the ampersand command is given in Chapter 4 of Little (1985).

CALIBRATION

As previously indicated, although the number of ticks is sufficient for data analysis, each tick does not necessarily occupy a standard unit of time. Thus, if we wish to compare the results of different experiments, the number of ticks must be converted to real-time values. The procedure for determining the actual time value of a tick is called *calibration*.

The need to calibrate software clocks with Method 4 is self-evident. However, as noted above, many users will

find it desirable to calibrate subroutines under Method 3. The reason can be seen if we examine the pseudocode example of Method 3 and notice that there are portions of the routine not included in the WAIT subroutine. Although this *overhead time* is generally insignificant for a few ticks, it can cause significant deviations between the nominal time and actual time after the passage of a number of ticks. An arithmetic example will make this clear. Suppose the overhead time for a routine is 30 clock cycles on the Apple's master clock. For a single tick, the overhead time would be less than .3 msec; thus, the deviation is trivial. However, after 500 msec, the total deviation would be almost 15 msec; after 1,000 ticks, 29 msec. The same reasoning can be applied to Method 2. However, in Method 2 the deviation is much less, because there will be fewer ticks for any given interval of time.

There are two basic methods for calibrating a clock (or any other assembler routine). The first is *nominal* calibration, which is done by simply adding up the time each assembly command takes. The details of this method are explained by Price (1979). The necessary information concerning the commands is available in a number of standard sources, including Irwin (1984) and Little (1985).

The second method, *empirical* calibration, is illustrated by the following example. Assume we want to empirically calibrate a routine designed to measure how long it takes for subjects to respond with a buttonpress. To calibrate the routine, we would start it, wait for a predetermined time interval (e.g., 30 min), and then press the button. Suppose then the counters had recorded 1,882,341 ticks. This would indicate that each tick must have taken .9563 msec. Since we can run the routine for an arbitrary interval of time, we can empirically calibrate a routine to any desired degree of precision (given, of course, that the counter is large enough to accommodate the desired degree of precision). In many instances, empirical calibration is easier to perform, because it is sometimes difficult to nominally calibrate routines having complex control structures with numerous branches. However, the most desirable approach would be to perform both calibrations, using one as a check on the other.

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.
- CAVANAGH, P., & ANSTIS, S. M. (1980). Visual psychophysics on the Apple II: Getting started. *Behavior Research Methods & Instrumentation*, 12, 614-626.
- DESANTIS, G. M., & DORFMAN, D. (1987). *A survey of current research procedures in cognitive psychology*. Unpublished manuscript.
- DORFMAN, D. (1984). *Softsystem: A program for running cognitive psychology experiments on Apple II computers* [Unpublished computer program].
- DLHOPOLSKY, J. G. (1983). Limitations of high-level microcomputer languages in software designed for psychological experimentation. *Behavior Research Methods, Instruments, & Computers*, 15, 459-464.
- FOLTZ, G. S., & POLTROCK, S. E. (1981). A microcomputer system for the human psychology laboratory. *Western Educational Computer Conference*, 5, 128-130.

- IRWIN, P. (1984). *Apple programmer's handbook*. Indianapolis, IN: H. W. Sams.
- LITTLE, G. B. (1985). *Inside the Apple //e*. Bowie, MD: Brady Communications.
- MITTERER, J. O., & OSBORNE, B. (1982). Improved real-time control on the Apple by mimicking an on-board 60-Hz clock. *Behavior Research Methods, Instruments, & Computers*, 14, 412-413.
- OSGOOD, G. (1984a). *Hardware implementation and utility manual* [for ApplePsych System]. Available from G. Osgood, Department of Psychology, University of Oregon, Eugene, OR 97403.
- OSGOOD, G. (1984b). *Software program development guide* [for ApplePsych System]. Available from G. Osgood, Department of Psychology, University of Oregon, Eugene, OR 97403.
- PRICE, J. M. (1979). Software timing for 6500 series microcomputers. *Behavior Research Methods & Instrumentation*, 11, 568-571.
- PRICE, J. M. (1986). Relocating 65xx software timing routines. *Behavior Research Methods, Instruments, & Computers*, 18, 68-69.
- REED, A. V. (1979). Microcomputer display timing: Problems and solutions. *Behavior Research Methods & Instrumentation*, 11, 572-576.
- ROSNER, R. (1978). Daddy, is it a PET? *ROM*, 1, 26-32.
- SCANLON, L. (1982). *APPLE II assembly language exercises*. New York: Wiley.
- WATTS, B. H. (1986). FORTH, a software solution to real-time computing problems. *Behavior Research Methods, Instruments, & Computers*, 18, 228-235.

NOTES

1. Hexadecimal numbers will be prefixed by a \$. Where a generally accepted name exists for a location, as in the case of VERTBLANK, I shall refer to it by that name.

2. The Monitor WAIT routine should not be confused with the Applesoft function of the same name, which is different in both purpose and operation.

APPENDIX A

Interval Timer Using Method 2

	LDX NTICKS	;* Set number of ticks.
VBIOFF	LDA VERTBLANK	;* Is vertical blanking interval over?
	BMI VBIOFF	;* No! Check again.
VBION	LDA VERTBLANK	;* Yes! Is scan over?
	BPL VBION	;* No! Check again.
	DEX	;* Decrement tick counter. More ticks?
	BNE VBIOFF	;* Yes! Do another tick.
	RTS	;* No! Return to calling program.

Comments. In the above example, NTICKS is a location containing the number of ticks the counter is supposed to run. This value could be passed using the BASIC POKE as explained in the text.

Notice the use of the BMI (branch on result negative) and BPL (branch on result positive) instructions, which illustrates how to test whether some value is greater than or equal to 128 without using a CMP instruction. This alternative code takes advantage of the fact that loading the accumulator with values 128 or more sets the negative flag, while loading the accumulator with values less than 128 clears it. It is more compact and faster executing than code using the CMP instruction.

APPENDIX B

Interval Timer Using Method 3

	LDX NTICKS	;* Set number of ticks.
LTEN	LDA #\$C8	;* Prepare WAIT to
	JSR WAIT	;* tick for 100.4037 milliseconds.
	DEX	;* Decrement tick counter. More ticks?
	BNE LTEN	;* Yes! Do another tick.
	RTS	;* No! Return to calling program.

Comment. See Appendix A for discussion of NTICKS.

APPENDIX C
Event Timer to Measure Subject's Response Time

	LDA #\$00	;* Zero
	STA C1	;* Counter location 1
	STA C2	;* Counter location 2
	STA C3	;* Counter location 3.
START	LDA #\$12	;* Prepare WAIT to
	JSR WAIT	;* Tick for 1.042332 milliseconds.
	INC C1	;* Increment Counter 1. Has it recycled?
	BEQ T256	;* Yes! Goto T256.
	JMP CHECK	;* No! Then check buttons.
T256	INC C2	;* Increment Counter 2. Has it recycled?
	BEQ T256	;* Yes! Goto T65536.
	JMP CHECK	;* No! Then check buttons.
T65536	INC C3	;* Increment Counter 3.
CHECK	LDY #\$1	;* Y = button being checked.
	LDA B1	;* Is Button 1 being pressed?
	BMI PREST	;* Yes! Goto PREST.
	INY	;* No! Check Button 2.
	LDA B2	;* Is Button 2 being pressed?
	BMI PREST	;* Yes! Goto PREST.
	JMP START	;* No! Do another tick.
PREST	STY RESP	;* Store identity of button.
	RTS	;* Return to calling program.

Comments. In this example it is assumed that B1 corresponds to the OPEN-APPLE key and B2 to the CLOSED-APPLE key. The status of these keys can be determined by examining location 49249 (\$C061) for the OPEN-APPLE and 49250 (\$C062) for the CLOSED-APPLE. The value at these locations is greater than or equal to 128 only when a key is being pressed. The number of elapsed ticks could be retrieved using the BASIC PEEK command, as explained in the text. The identity of the key being pressed could be retrieved in the same fashion from location RESP. The counter in this example has sufficient capacity to time intervals of over 4 h and thus is adequate for empirical calibration.

This subroutine could be changed to Method 4 by eliminating the JSR WAIT and the two JMP CHECK instructions (instructions 6, 9, and 12) and by substituting a JMP CHECK instruction for the present LDA #\$12 instruction (instruction 5). A fourth location would have to be added to the counter due to the faster tick rate.