INSTRUMENTATION & TECHNIQUES

Screen display and timing synchronization on the Apple IIe and Apple IIgs

DOUGLAS B. EAMON

University of Wisconsin-Whitewater, Whitewater, Wisconsin

The logic of the vertical blanking interval (VBL) softswitch in the Apple IIgs is the reverse of the logic of the VBL in the Apple IIe. Programs written for the Apple IIe that synchronize the videoscans with the experiment control program timing by sensing the VBL pulse will therefore function incorrectly when run on an Apple IIgs. Procedures for modifying existing synchronization routines are discussed. A flexible alternative program that senses the type of computer on which it is running and accordingly adjusts the logic of the synchronization routine is described.

The Apple IIgs, a 16-bit upgrade of the Apple IIe, is faster and more versatile than its widely used predecessors, yet generally compatible with them in both software and hardware. This makes it an appropriate choice among psychologists who wish to move to more powerful computers without sacrificing substantial quantities of instructional and research programs they may have collected and used over several years.

Precise timing on the Apple II series (or on any computer with a raster scan display) requires that the stimulus presentation on the screen be synchronized with the timing routine (Reed, 1979). On the Apple IIe, this is accomplished with the monitoring of memory location \$C019, called RDVBLBAR, which holds the status of the vertical blanking interval (VBL).¹ Procedures for monitoring the VBL to synchronize displays on the Apple II+ and Apple IIe have become available (e.g., Adams, 1985; Cavanagh & Anstis, 1980; Dorfman, 1987; Reed, 1979; Sanmartin, 1987).

Programs may use RDVBLBAR in three ways: (1) A program may wait for the onset of a VBL before transferring characters from string array memory to text screen memory. Or, a program may wait for the onset of a VBL before drawing or altering a graphics display. The characters transferred or the changes in the graphics image will not become visible to subjects until a display cycle actually writes them on the screen, at which point a timer can be started. This use of the VBL assures the longest possible time in which to transfer information to screen

memory. Sanmartin (1987) employs this technique, which is also mentioned by Reed (1979). (2) Other programs (e.g., Diener & Smees, 1984) write the stimuli characters to a "hidden" screen memory area (a screen memory area other than the one currently displayed), then wait for the offset of a VBL before switching screen display pages and starting a timer. This also allows a timing routine to be synchronized with the onset of the display phase of the scan. (3) A third use of the VBL signal is to count its regular $\frac{1}{60}$ -sec "ticks" in a timing routine (e.g., Dorfman, 1987).

Although all of these techniques have been successfully employed on the Apple II and IIe, the Apple IIgs differs in an important way from the Apple IIe in its VBL soft switch signal. Although the Apple IIgs uses the same \$C019 location to store the VBL status, the logic of the switch is exactly the reverse of the logic in the Apple IIe. In the Apple IIe, RDVBLBAR is 0 (bit 7 clear) during a VBL (Apple Computer, 1986), but in the IIgs, RDVBLBAR is negative (bit 7 set) during a VBL (Lancaster, 1988; Little, 1989).² Therefore, Apple IIe-based programs that use RDVBLBAR to synchronize timing with screen displays will contain substantial errors when run on an Apple IIgs. Use of the VBL as a "tick" counter should be unaffected.

Two solutions to the VBL logic problem are discussed in this paper. The first is to modify the synchronization routine. This may be most suitable when the programs must remain the same length, or when they involve complicated sequences of display pages. The second solution is to substitute a new synchronization routine for the existing one. This approach may be warranted if the user is comfortable with making such modifications (i.e., is familiar with the programming language), or if new programs that need the synchronization routine are under de-

Requests for reprints should be addressed to Douglas B. Eamon, Department of Psychology, University of Wisconsin-Whitewater, Whitewater, WI 53190.

velopment. Ideally, the new routine should work correctly on either an Apple IIe or an Apple IIgs.

Modifying Existing Programs

Procedures for modifying existing code depend on where the synchronization routine is found and whether the user has access to the code itself. Obviously, this option is not available for copy-protected programs.

Applesoft BASIC. One solution to the problem is simply to reverse the logic of the Apple IIe synchronization routine when using an Apple IIgs. In routines that access RDVBLBAR from Applesoft BASIC, this change can usually be easily made. For example, Diener and Smee's (1984) Applesoft routine,

```
WAIT -16359, 128: WAIT -16359, 128, 128:
WAIT -16359, 128: ...
```

can be changed to

```
WAIT -16359, 128, 128: WAIT -16359, 128:
WAIT -16359, 128, 128: ...
```

for use on the Apple IIgs.

Cavanagh and Anstis (1980) provide a routine similar to Diener and Smee's (1984), but follow it with a short loop. Although one can attempt the same modification, problems will arise because the IIgs runs about 2.5 times faster than the IIe, and thus the loop will take less time on the IIgs than on the IIe. If the Apple IIgs is set to its "slow" speed (1 MHz), the delay should work as is; to use the programs at the IIgs's "fast" speed, multiplying the loop terminator by 2.5 may give a value close to the desired delay, but independent calibration would be required.

Machine language. If the location of the synchronization routine is known, machine language routines that sense the VBL can also be changed. For example, in the code below (suggested by Adams, 1985, to synchronize a page switch), it is only necessary to exchange the BMI (hex \$30) and the BPL (hex \$10).

```
AD 19 C0 WAITVBL LDA RDVBLBAR ;Read status
30 FB BMI WAITVBL ; location and
; wait for onset
; of VBL
AD 19 C0 WAITVBL1 LDA RDVBLBAR ;Then continue
10 FB BPL WAITVBL1 ; to wait
; for screen
; display
```

Some programs (e.g., Lancaster, 1984) use a BIT instruction rather than LDA. Since the BIT instruction also sets the negative flag on the 65C02, the effect is the same as LDA's, and the same modification will work.

These examples suggest that to correct the logic in routines designed to run on the IIe, so that they can run properly on a IIgs, it is only necessary to search the machine code for the occurrences of the sequences "19 C0 30" (RDVBLBAR followed by BMI), and "19 C0

10" (RDVBLBAR followed by BPL), and exchange the "30" with "10" (i.e., the BMI for BPL).

Assembly language. If the assembly source code is available, several procedures are available for modifying existing code to correct for the VBL logic incompatibility. The easiest and most obvious solution is to rewrite and reassemble the code, exchanging BMIs for BPLs, as suggested above for machine code. There are two main advantages to this approach: it can be done easily and quickly, and the resulting code will be the same length as the original. The second consideration might be important if PEEKs or POKEs are used to set up or recover information from the program.

Generally, however, a better solution is to rewrite the code so that it can sense the type of computer on which it is running, then correct the logic appropriately. This allows the program to be run on either type of computer without further concern about the synchronization. The Apple IIe and IIgs can easily be distinguished in software by setting the carry flag, then doing a JSR ("jump to subroutine") to location \$FE1F. On the Apple IIe, this location contains an RTS ("return from subroutine"); other code clears the carry flag in the IIgs (Apple Computer, 1987a). The following routine might be used to store a code for the computer type at the previously defined location COMPTYPE:

```
CHKTYPE LDA #$7F ;Decimal 127 will be code for
; IIe or earlier
STA COMPTYPE ;Assume IIe or earlier
;Now check the computer
; type
SEC ;Set carry flag
JSR $FE1F ;On the IIe, this is an RTS
; (return from subroutine)
;On the IIgs, other code
; clears the carry flag
BCS CONTINUE ;Carry is still set, so must
; have been a IIe
ASL COMPTYPE ;Multiply by 2, giving 254
; as code for IIgs
```

CONTINUE ...

After exiting this routine, location COMPTYPE will contain \$7F (decimal 127) if the computer is an Apple IIe (or earlier), or \$FE (254) if the computer is an Apple IIgs. Selection of these values as codes is not arbitrary, since they allow only a slight modification of a procedure used by Cavanagh and Anstis (1980) to provide synchronization for both computers. For example, the following routine could be used to wait for onset of the VBL before beginning a transfer of characters to screen memory:

```
LDA COMPTYPE ;Either 127 (IIe) or 254 (IIgs)
CHKVBL1 CMP RDVBLBAR ;If VBL is currently
; in progress
```

```

        BPL CHKVBL1      ; wait for it to end
CHKVBL2 CMP RDVBLBAR    ;Then wait for the next VBL
                        to begin
        BMI CHKVBL2      ; before writing to screen
                        memory

```

Onset of the VBL causes bit 7 of RDVBLBAR to be cleared to 0 on an Apple IIe, thus COMPTYPE (127 for a IIe) will be greater than RDVBLBAR if the VBL is on, and the program will remain in the first loop until the VBL is finished. When the VBL is over, bit 7 of RDVBLBAR is set; i.e., it will be greater than 127.³ The program will then fall into the second loop and remain in this loop until the display phase is finished and a new VBL starts. At that time, the entire VBL interval (about 4.45 msec) will be available for writing information to screen memory. Since COMPTYPE will be less than 128 on an Apple IIe and greater than 128 on an Apple IIgs, the routine will work in either computer.

These two modifications together add only 15 bytes to the length of compatible existing programs (assuming that a page zero location is used for COMPTYPE). Thus, they may usually be added to an existing program and stored in the same location. Care must be taken to adjust any CALLs, PEEKs, or POKEs if these are used to pass variables from or back to a BASIC program.

A General Purpose Stimulus Presentation and Timing Routine

When the location of the machine code is not known, or the assembly source program is not available, or the code is complex, altering existing routines may not be an appropriate solution. Unless they use a computer identification routine such as that above, even those who choose to alter the code will be left with programs that run correctly on the Apple IIgs, but not on an Apple IIe.

An alternate solution is to replace the code with a new routine. To facilitate installation of a new routine into existing programs (or into new programs), it should meet several criteria: (1) The code should distinguish the type of computer (Apple IIe or IIgs) on which it is running, and use the correct logic to synchronize field display and timing. (2) It should be flexible enough to substitute for a wide variety of routines currently in use; that is, it should be able to provide common functions available in other routines such as interval timing, response timing, stimulus presentation, and masking. The routine should also (3) be relocatable, so that other machine language routines can be left in place, (4) be easy to install as a substitute for timing routines in existing programs, and (5) allow a simple method for passing variables between the routine and BASIC.

GPSRT (general purpose stimulus presentation and response timer) is a flexible routine that meets these criteria and that may be suitable for many applications. In addition to synchronizing the video display with software

timing on either the Apple IIe or IIgs, it can optionally present a stimulus and/or a mask for a specified interval and obtain the response time to the stimulus presentation under user-defined conditions.

GPSRT is designed to be called from Applesoft BASIC, using either a CALL or the ampersand (&) command. The calling program must supply information regarding the horizontal and vertical position of the stimulus to be presented, the stimulus (or a null string), a mask (or a null string) that may overwrite the stimulus at the end of the presentation period, the stimulus display time (in milliseconds), the maximum time to wait for a response, a list of acceptable responses, and the names of variables in which to return the response and response time.

The routine uses the identification procedure described above to determine whether the program is running on an Apple IIgs or an Apple IIe, and adjusts the logic of the VBL sensing code appropriately. The code for the computer type is also stored in a page zero memory location. This information can be useful in other ways—for example, to allow optional access to the built-in clock on the IIgs.

GPSRT is completely relocatable, which allows it to be placed anywhere in memory that sufficient space (522 bytes) can be found. A program that automatically installs the routine as part of an Applesoft program is available.

If the ampersand vector has been set to point to the routine, stimulus presentation and timing can be initiated with the following syntax:

```

&HORIZPOS, VERTPOS, STIMULUS$, MASK$, DISPTIME,
MAXRESPTIME, ACCRESP$, RESPONSE$, RESPTIME

```

All of the parameters used to control the stimulus presentation may be constants (e.g., "12", "star"), simple variables (e.g., HP, ST\$), array variables [e.g., HP(I), ST\$(J)], or expressions [e.g., HP(I)/2, "The" + ST\$(J) + "is above the" ST\$(K)].

A stimulus and a mask may be displayed anywhere on the screen, or at the current (most recently used) cursor position. Through selection of appropriate values for DISPTIME and MAXRESPTIME, it is possible to display stimulus and mask strings under three timing conditions: (1) The STIMULUS\$ string can be displayed for a specified time while the program waits for an acceptable response. If an acceptable response is made within the time specified (or the time elapses without an acceptable response), the STIMULUS\$ string is replaced by the MASK\$ string, and the response and response time (or an error code, if no response was made) are returned to the calling program. (2) The STIMULUS\$ string can be displayed for a specified time during which no response is accepted. Following this, the MASK\$ is presented for an additional period during which responses are accepted. This allows an easy procedure for presenting stimuli in

same-different tasks, or for presenting a fixation point followed by a stimulus. (3) The stimulus string can be displayed for a specified time while the program waits for a response. If an acceptable response is made within the time specified (or the time elapses without an acceptable response), the program returns the response and the response time (or an error code). This is similar to (1), except that the STIMULUS\$ is not overwritten by the MASK\$.

The user can specify a set of acceptable responses; other responses will be ignored. The "open-Apple" and "closed-Apple" keys (game paddle buttons #0 and #1) may be included in the set of acceptable responses through the use of special codes. If the set of acceptable responses is null, any response will be accepted.

Although these specifications may appear lengthy, they result in a versatile program that is easy to use. The program automatically converts all alphabetic responses to uppercase before comparing them with the set of acceptable responses; this relieves the calling program of the task of checking for both upper- and lowercase responses. As a result, entering a lowercase character as the only "acceptable response" will force the routine to time out in every case, since after conversion to uppercase, no response will match the "acceptable response." This provides a flexible interval timer, which also may present information to the screen. GPSRPT reads values of the user-supplied Applesoft variables directly from memory and saves RESPONSE\$ and RESPTIME directly to memory, so no PEEKs or POKEs are necessary to pass variables to the routine or to recover its results. Because the user-supplied control parameters may be constants, simple variables, array variables, or expressions, the routine can easily be incorporated into a loop providing almost complete machine-language control of an entire experiment.

Limitations and cautions. Although the routine could easily be modified to allow page switching (e.g., for using the high-resolution graphics screens), the current version of GPSRPT is suitable for presentation of text stimuli on the 40-column screen only. Furthermore, the program will accept only single-key responses; users wishing to allow the input of multiple-key responses may choose to follow the machine-language call with an INPUT (or similar) command, and then concatenate the response returned by GPSRPT with the INPUT string.

In order to facilitate speed of processing, error checking has been minimized; it is up to the programmer to ensure that the values supplied to the routine fall within the ranges specified.

Availability. GPSRPT may be obtained from the author on a 5.25-in. disk (ProDOS and DOS 3.3) for \$10. The disk includes further documentation of the routine, Applesoft programming examples, and a file that will automatically load the routine, set up the ampersand vector, and attach the routine to the end of an Applesoft program so that it can be saved and loaded to/from a disk in a

"transparent" way (i.e., without POKEing data into memory or BLOADing the routine from disk; Eamon, 1988). A text file version of the source assembly program, written using the MicroSPARC Assembler (Floeter & Floeter, 1987), is also included.

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.
- APPLE COMPUTER. (1986). *Apple IIe technical reference manual*. Reading, MA: Addison-Wesley.
- APPLE COMPUTER. (1987a). *Apple IIgs firmware reference*. Reading, MA: Addison-Wesley.
- APPLE COMPUTER. (1987b). *Apple IIgs hardware reference*. Reading, MA: Addison-Wesley.
- CAVANAGH, P., & ANSTIS, S. M. (1980). Visual psychophysics on the Apple II: Getting started. *Behavior Research Methods & Instrumentation*, **12**, 614-626.
- DIENER, D., & SMEE, W. P. (1984). Apple tachistoscope. *Behavior Research Methods, Instruments, & Computers*, **16**, 540-544.
- DORFMAN, D. (1987). Softer timing of events in cognitive psychology experiments. *Behavior Research Methods, Instruments, & Computers*, **19**, 185-190.
- EAMON, D. B. (1988). Enhancing Applesoft using ampersand routines. *Behavior Research Methods, Instruments, & Computers*, **20**, 224-230.
- FLOETER, A., & FLOETER, V. (1987). *MicroSPARC Assembler* [Computer program]. Concord, MA: MicroSPARC, Inc.
- LANCASTER, D. (1984). *Enhancing your Apple II* (Vol. 1, 2nd ed.). Indianapolis, IN: Howard W. Sams.
- LANCASTER, D. (1988, August). Ask the guru. *Computer Shopper*, 338-345.
- LITTLE, G. B. (1985). *Inside the Apple IIe*. Bowie, MD: Brady Communications.
- LITTLE, G. B. (1989, February). Vertical blanking [Response to letter to the editor]. *A+*, 118.
- REED, A. V. (1979). Microcomputer display timing: Problems and solutions. *Behavior Research Methods & Instrumentation*, **11**, 572-576.
- SANMARTIN, J. (1987). TREC: A tool kit for programming cognitive experiments in Applesoft BASIC. *Behavior Research Methods, Instruments, & Computers*, **19**, 467-468.

NOTES

1. The Apple II+, discussed by Reed (1979), does not make this signal available at \$C019. One of Reed's contributions was to describe a simple hardware modification that made it possible to bring the VBL signal to another location from which it could be monitored. Since the VBL signal is readily available on the Apple IIe and IIgs, the routines described in this paper do not support this modification.

It should be observed that there is a slight (up to 7- μ sec) "jitter" in all of the synchronization routines discussed in this paper. The "jitter" is due to processing time of the assembly routine itself (see Lancaster, 1984, for details and a software fix).

2. There is some confusion about the value of RDVBLBAR during a VBL on the Apple IIgs. Much of this was created by Apple Computer, which has published incorrect information about the status of RDVBLBAR in at least two different sources (Apple Computer, 1987a, p. 277; 1987b, pp. 69, 260). Little (1985) also included incorrect information about the status of RDVBLBAR in his popular book on the Apple IIe. However, as manager of technical references and special resources for Apple Computer, Little later published a corrected statement (Little, 1989).

The value of RDVBLBAR during a VBL can be readily determined by setting up a loop to sample the proportion of time its bit 7 is set or clear over a short period, as in the following:

```

RDVBLBAR = 49177 :REM $C019 hex
FOR I = 1 TO 1000
X = X + ( PEEK (RDVBLBAR) > 127 )
:REM increment X if bit 7 set
NEXT I
PRINT X/(I-1)

```

The VBL pulse uses 70 of the 262 scan lines on the display (Apple Computer, 1986), or about 4.45 msec of the 16.67-msec total scan cycle. If $X/(I-1)$ is about .27 (70/262), then bit 7 of RDVBLBAR is set during the VBL; if $X/(I-1)$ is substantially higher (e.g., .73), then bit 7 of RDVBLBAR is set during the display cycle (clear during the VBL).

This technique can also be used to determine on which computer (Ile or Iigs) other software is running, if the delay is acceptable.

3. Hex digits with the high bit (bit 7) set are interpreted as negative numbers, so 128 (hex \$80) is treated as a negative. The comparison works anyway, however, since the negative values decrease in absolute value from 255 (hex \$FF, or -1 decimal) to 128 (hex \$80, or -127 decimal).

(Manuscript received March 7, 1989;
revision accepted for publication May 24, 1989.)