Appendix H **Controlling the Apple IIc Plus Accelerator**

This appendix describes how an assembly-language program can control the cache glue gate array (CGGA) chip in the Apple IIc Plus and provides code samples that you can incorporate into your program. This information is provided for the sake of completeness only. Any code that changes speed settings for ports or that disables the CGGA cannot be run on any machine other than the Apple IIc Plus. Any code that does not strictly adhere to the guidelines in this appendix is guaranteed not to work on future versions of the Apple IIc Plus. See the section "The Apple IIc Plus Cache Glue Gate Array (CGGA)" in Chapter 11 for a general description of the CGGA and for approved methods of controlling the speed of the Apple IIc Plus. ■

When the Apple IIc Plus is switched on or reset, ROM code for ports 1, 2, 5, and 6, and the code for the speaker and game paddles cannot be cached; this code is restricted to running at 1.023 MHz. The code for ports 3, 4, and 7 can be cached, and so can run at up to 4 MHz. These settings were chosen to allow code written for other Apple II computers to run on the Apple IIc Plus; we recommend that you never change these settings. Before you decide whether or not to change these default settings or disable the CGGA, consider the following points:

- Writing a 1 to any CGGA control word bits that are reserved can cause the system to crash. The Write command is described later in this appendix.
- Invalid data in a CGGA command can cause the system to crash.
- Executing a CGGA command changes the state of the DHiRes switch, altering the state of graphics screens. You must return the DHiRes switch to its original state when you are finished executing any CGGA command.
- If you speed up port 2, the Wait routine in firmware (see Appendix F) runs at 4 MHz rather than 1 MHz. Use the Wait routine in the code sample at the end of this appendix instead.
- If you speed up port 2, modem code might fail to work correctly.
- If you speed up ports 5 and 6, the disk drives no longer function.
- If the ROM isn't switched in before you execute a CGGA command, the system will crash. If RAM was switched in before you started, remember to return the RdLCRAM soft switch to its prior state before quitting.
- If you attempt to speed up the game paddles, they no longer function.
- Executing the Write command to the CGGA when the CGGA is disabled causes unpredictable results. You must be sure the CGGA is enabled before executing the Write command.
- There is no way to determine the state of the system's speed at any given time—many factors cause it to change frequently.
- If you execute a command to the CGGA on any machine other than the Apple IIc Plus, the system will crash.
- Making changes to the state of the CGGA can cause other applications to work incorrectly.
- Any or all of the above caveats may change with future revisions of the Apple IIc Plus hardware and firmware.

If your program makes any changes in CGGA settings, you *must* restore the CGGA to its original state before your program exits. Any changes in CGGA settings can prevent another application from running properly. Each time the system is reset, the Reset handler returns the CGGA to the default settings described at the beginning of this section; if your application quits by resetting the system, your program does not have to reset the CGGA.

CGGA commands

This section describes how to make calls to the CGGA to enable it, disable it, and change its mode of operation.

▲ **Warning** Modifying the accelerator registers without a full understanding of the CGGA and the Apple IIc Plus hardware and firmware can render the system inoperative, requiring the user to shut the machine off and turn it back on again to regain control. ▲

To send a command to the CGGA, you must first push the command parameters onto the stack, then execute a JSR instruction to the accelerator entry point, $C7C7. The parameters consist of a pointer to a buffer (when necessary) and a command number. For your convenience, sample code is provided with each command description showing the proper way to set up the parameters for that call.

The CGGA firmware pulls the parameters off the stack and checks the command number to determine if it corresponds to a valid command. If the command number is valid, the firmware performs the function specified by the command and returns to the calling routine with a value of $00 in the A register (accumulator). If the command number is not valid, the firmware returns the value $01 in the A register to indicate an error. In either case, the c (carry) flag is set. The calls themselves do not return errors. If the command number is valid, the firmware assumes that the parameters provided are also valid.

Before sending a command to the CGGA, you must be sure that the lower half of the ROM (the main ROM) is selected and that the ROM is switched in. To determine whether the main ROM is selected, check the contents of location $FCFF. If $FCFF contains a nonzero value, the main ROM is selected.

▲ **Warning** The system will crash if you send a command to the CGGA when the main ROM is not both selected and switched in. ▲

$01 Enable Accelerator

Description It is possible for an application program to disable the CGGA completely. The Enable Accelerator command reenables the CGGA.

Command number $01

Parameter list Command number

Example

```
lda     #$01                ;Enable Accelerator command
pha                         ;Command pushed on stack
jsr     Accelerator_Entry   ;Jump to the Accelerator
                            ;entry point
```

$02 Disable Accelerator

Description The Disable Accelerator command disables the CGGA completely. The Apple IIc Plus operates at 1 MHz as if there were no CGGA chip installed.

Command number $02

Parameter list Command number

Example

```
lda     #$02                ;Disable Accelerator command
pha                         ;Command pushed on stack
jsr     Accelerator_Entry   ;Jump to the Accelerator
                            ;entry point
```

$03 Lock Accelerator

Description The Lock Accelerator command locks the CGGA so that it cannot receive any commands except for the Unlock Accelerator command.

Command number $03

Parameter list Command number

Example

```
lda     #$03                ;Lock Accelerator command
pha                         ;Command pushed on stack
jsr     Accelerator_Entry   ;Jump to the Accelerator
                            ;entry point
```

$04 Unlock Accelerator

Description The Unlock Accelerator command reverses the effect of the Lock Accelerator command, making it possible for the CGGA to accept all commands.

Command number $04

Parameter list Command number

Example

```
lda     #$04                ;Unlock Accelerator command
pha                         ;Command pushed on stack
jsr     Accelerator_Entry   ;Jump to the Accelerator
                            ;entry point
```

$05 Read Accelerator

Description

The Read Accelerator command reads the CGGA registers, codes the state of the registers into a 2-byte word known as the *control word*, and places the control word in a buffer defined by the calling routine. The coding for the control word is shown in Table H-1.

■ **Table H-1** Accelerator control word

Bit	Meaning
Low byte	
7	Speaker speed (1 = fast)
6	Port 7 speed (1 = fast)
5	Port 6 speed (1 = fast)
4	Port 5 speed (1 = fast)
3	Port 4 speed (1 = fast)
2	Port 3 speed (1 = fast)
1	Port 2 speed (1 = fast)
0	Port 1 speed (1 = fast)
High byte	
7	Reserved
6	Paddle speed (1 = slow)
5	Reserved
4	Reserved
3	CGGA enable (1 = disabled)*
2	Reserved
1	Reserved
0	Reserved

* This bit is set and cleared by the Disable Accelerator and Enable Accelerator commands, not by the Write Accelerator command. For the Write Accelerator command, this bit is reserved.

Command number $05

Parameter list Command number
Pointer to buffer in which to store the control word

Example

```
lda     #<buffer            ;High byte of buffer address
pha                         ;Byte pushed on stack
lda     #>buffer            ;Low byte of buffer address
pha                         ;Byte pushed on stack
lda     #$05                ;Read Accelerator command
pha                         ;Command pushed on stack
jsr     Accelerator_Entry   ;Jump to the Accelerator
                            ;entry point
```

$06 Write Accelerator

Description The Write Accelerator command sends a control word to the CGGA, resetting the values in the CGGA's internal registers. You can use this command to control which ports in the Apple IIc run at 1 MHz and which run at 4 MHz. The CGGA control word is shown in Table H-1. Notice that you use the Enable Accelerator and Disable Accelerator commands to set or clear bit 3 of the high byte; do not write to this bit. All other bits are set by the Write Accelerator command.

△ **Important** Executing the Write command to the CGGA when the CGGA is disabled causes unpredictable results. Use the Read command—and then the Enable command if necessary—to make sure the CGGA is enabled before executing the Write command. △

Command number $06

Parameter list Command number
Pointer to buffer in which the control word is stored

Example

```
lda     #<buffer            ;High byte of buffer address
pha                         ;Byte pushed on stack
lda     #>buffer            ;Low byte of buffer address
pha                         ;Byte pushed on stack
lda     #$06                ;Write Accelerator command
pha                         ;Command pushed on stack
jsr     Accelerator_Entry   ;Jump to the Accelerator
                            ;entry point
```

Code sample

The following code sample constitutes a shell that goes around any calls you make to the CGGA. Several routines are provided; use only those that you need. The code sample includes the following routines:

- The main routine that calls the other subroutines.
- A routine that checks the ID bytes of the computer to determine if it is an Apple IIc Plus. If you are certain that the machine is an Apple IIc Plus, you don't have to check it again, but remember that any CGGA call causes any Apple II computer *other* than an Apple IIc Plus to crash.
- A routine that saves the states of the DHiRes and 80Col soft switches and turns off 80-column mode before you send any commands to the CGGA. If you can make your calls to the CGGA at the beginning of your program before setting the DHiRes switch and 80Col switch, then you do not have to use this routine. Just be sure to set the DHiRes and 80Col switches to the settings you want after you have finished sending commands to the CGGA. Remember also to switch in the main ROM before sending any commands to the CGGA, and to return the RdLCRAM soft switch to its prior state when you are finished.
- A routine that restores the saved state of the machine.
- A routine that unlocks the CGGA so that it can receive commands. You must use this routine before sending any commands to the CGGA.
- A routine that locks the CGGA so that no additional commands can be sent to it during normal system use. You must use this routine before quitting.
- A routine that you can use instead of the firmware version of the Wait routine (described in Appendix F) if your program speeds up port 2.

```
*******************************************************
*using the accelerator:
*
*This code implements any of the calls documented
*in this section that talk to the accelerator in the IIc Plus.
*
*               Entry:   ROM must be enabled
*
*               Exit:    C=0
*                        A=0
*                        X,Y undefined
*******************************************************
use.accel
accel.entry     equ      $C7C7              ;entry point to talk to accelerator

                jsr      check.id           ;must be at least a IIc Plus
                bcc      dont               ;won't work on any other machine
                jsr      save.state         ;save state if you don't know what it is

                jsr      unlock             ;must unlock it before anything else
*               .
*               .
*add your call(s) here.  See descriptions of calls for format
*               .
*               .
                jsr      lock               ;must lock it when all finished

                jsr      restore.state      ;must restore it if you saved it
dont            rts                         ;go back to the calling routine or user
```

```
*************************************************************
*check.id:      Calls to the accelerator can ONLY be made on
*              a IIc Plus.  If this call is made on any other
*              Apple II machine, the program WILL crash!
*              This routine checks for the current IIc Plus
*              and beyond (for future compatability).
*
*              Entry:   ROM must be enabled
*
*              Exit:    C=0 if not IIc Plus
*                       C=1 if IIc Plus
*                       A,X,Y undefined
*************************************************************
check.id
id1            equ      $FBB3
id2            equ      $FBC0
id3            equ      $FBBF

               lda      id1              ;should be $06
               cmp      #$06
               bne      no               ;if not IIc Plus then quit

               lda      id2
               bne      no               ;should be $00 -- if not then quit

               lda      id3              ;should be $≥ 5
               bmi      no               ;make sure it's not the orig IIc with #$FF
               cmp      #$05
               bcs      done             ;if c=0 then it failed-- make sure it does!
no
               clc                       ;clear carry means wrong machine
done           rts                       ;go back with status of test
```

```
*****************************************************************
*save/unsave state:      This routine saves and restores the states
*                        of the double hires and 80 column soft
*                        switches.  It turns off 80 column mode
*                        so that talking to the accelerator can't
*                        accidently turn on double hires.
*
*NOTE:                   If you put any calls to the accelerator
*                        BEFORE you set the state of the machine
*                        at the beginning of your program, then
*                        this routine will not be necessary.  Simply
*                        make sure your initialization routine
*                        sets the 80 column switch and the double
*                        hires switches to the state your program
*                        requires after making any calls to the
*                        accelerator.
*
*               Entry:   nothing
*
*               Exit:    A scrambled
*                        80Col firmware is turned off
*                        X,Y unchanged
*****************************************************************
save.state

rd80col         equ      $C01F           ;if bit 7 =1 then 80col is on
rddhires        equ      $C07F           ;if bit 7 = 0 then dhires is on
on.80col        equ      $C00D           ;writing turns on 80 col
off.80col       equ      $C00C           ;writing turns off 80 col
on.dhires       equ      $C05E           ;writing turns on dhires
off.dhires      equ      $C05F           ;writing turns off dhires

                lda      rd80col         ;see if 80 column is on
                asl      a               ;save state
                bcc      @1              ;if branch then it's off
                sta      off.80col       ;turn it off first
@1
                lda      rddhires        ;see if double hires on
                php                      ;save c and n flags on stack
                pla
                sta      temp            ;store result of both tests
                rts                      ;return to caller

unsave.state
                lda      temp            ;get back the status flags
                pha
                plp                      ;want status flags back in P reg
                bcc      @1              ;branch if 80Col should be off
                                         ;because it already is
                sta      on.80col        ;should be on - so turn it on
@1              bmi      @2              ;branch if dhires is off
                sta      on.dhires       ;restore dhires to on
                bpl      done2           ;branch unconditionally
@2              sta      off.dhires      ;restore dhires to off
done2           rts

temp            dfb      $00             ;used for temp storing of states
```

```
*******************************************************************
*unlock:         Unlocks the accelerator so it can be accessed.  This
*               must be done before anything else.
*
*NOTE:           Please note that the state of some soft switches
*               will be affected by this call.  (See save/restore.state)
*
*Entry:          nothing
*
*Exit:           C=0
*               A=0
*******************************************************************
unlock
                lda     #$04            ;cmd to unlock
                pha                     ;store it on stack
                jsr     accel.entry     ;call ROM
                rts                     ;back to caller
*******************************************************************
*lock:           Locks the accelerator so that normal system
*               use cannot affect it.  This must be the last
*               call made to the accelerator.
*
*Entry:          nothing
*
*Exit:           C=0
*               A=0
*******************************************************************
lock
                lda     #$03            ;cmd to lock accelerator
                pha                     ;store it on stack
                jsr     accel.entry     ;call ROM
                rts                     ;back to caller
```

```
*************************************************************************
*Wait.ram:       This replaces the ROM version of the wait routine
*                if your program speeds up port 2.
*
*NOTE:           Your program must call this routine and not the one in
*                ROM.  You may put this in the language card and disable
*                the ROM if you are not using any other ROM routines.
*
*NOTE:           This routine will run correctly at either fast or normal
*                speed.  It can also be run on any other version of the IIc
*                or IIe without harm.
*
*Entry:          A=$00 - $FF depending on the amount of time to wait:
*                Min delay = 1/2(50+25A+5A^2)+29
*                Delay is at least as great as caused by the wait routine in ROM,
*                and in most cases is exactly the same.
*                This routine has A-12 fewer cycles than the wait routine
*                in ROM.
*                X,Y,P undefined
*
*Exit:           X,Y unchanged
*                A=$00
*                P undefined
*************************************************************************
wait.ram

kbd             equ     $C000           ;address offset for port I/O

                phy                     ;save X and Y
                ldy     #$D0            ;$C0D0 is guaranteed 50 ms slow
                phx                     ;on IIc Plus but won't hurt other
                sec                     ;IIc or IIe's
                txa                     ;save wait value
@1
                lda     kbd,Y           ;this starts the slow down
                txa                     ;new version of wait routine
@2
                sbc     #$01            ;min delay = 1/2(50+25A+5A^2)+29
                bne     @2              ;timing is at least that of the old
                dex                     ;one and in most cases exact timing!
                bne     @1
                plx                     ;restore X and Y
                ply
                rts
```