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.



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

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

	\$02	Disable	Accelerator
--	------	---------	-------------

DescriptionThe 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

ExampleIda#\$02;Disable Accelerator command
phapha;Command pushed on stackjsrAccelerator_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	Comma	and number		
Example	lda	#\$03	;Lock Accelerator command	
	jsr	Accelerator_Entry	;Jump to the Accelerator ;entry point	

\$04	Unlock	Accelerator
Y • -		

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
 Ida
 #\$04
 ;Unlock Accelerator command

 pha
 ;Command pushed on stack

 jsr
 Accelerator_Entry
 ;Jump to the Accelerator

 ;entry point
 ;

Read Accelerator

Description

\$05

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 ;Byte pushed on stack pha ;Low byte of buffer address lda #>buffer ;Byte pushed on stack pha ¥\$05 ;Read Accelerator command lda ;Command pushed on stack ;Jump to the Accelerator pha Accelerator_Entry jsr ;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.

....

Command number \$06

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

. . . .

Example	Ida	# <buffer< th=""><th>;High byte of buffer address</th></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 \$C7C7 ;entry point to talk to accelerator accel.entry equ jsr check.id ;must be at least a IIc Plus ;won't work on any other machine bcc dont ;save state if you don't know what it is jsr save.state ;must unlock it before anything else jsr unlock * . *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 ;go back to the calling routine or user dont rts

*****	********	***********	*******
*check.id:	Calls to	the accelerator ca	an ONLY be made on
*	a IIC Plu	us. If this call :	is made on any other
*	Apple II	machine, the progr	ram WILL crash!
*	This rout	tine checks for the	e current IIc Plus
×	and beyo	nd (for future comp	patability).
*			
*	Entry:	ROM must be enable	ed
*			
*	Exit:	C=0 if not IIc Plu	s
*		C=1 if IIc Plus	
*		A,X,Y undefined	
*****	********	* * * * * * * * * * * * * * * * * * * *	*********
check,id			
id1	equ	\$FBB3	
id2	equ	\$FBC0	
id3	equ	ŞFBBF	
	lda	idl	;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 S> 5
	bmi	no	make sure it's not the orig IIc with #SEE
	Cmp	#S05	, ment toto ito o noo end orig ito with mort
	bcs	done	; if c=0 then it failed make sure it does!
no			, and a substantial make suit it does,
	clc		clear carry means wrong machine
done	rts		; go back with status of test
			, je seen nation statute of book

<pre>tave/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 * *NOTE: If you put any calls to the accelerator * *NOTE: If you put any calls to the accelerator * * *NOTE: * * * * * * * * * * * * *</pre>	*****	******	****	* * * * * * * * * * * * * * * * * * * *	
<pre>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 ** NOTE: If you put any calls to the accelerator ** ** NOTE: If you put any calls to the accelerator ** ** ** ** ** ** ** ** ** ** ** ** **</pre>	*save/unsave st	ate:	This routine saves and restores the states		
<pre>switches. It turns off 80 column mode so that talking to the accelerator can't accidently turn on double hires. *NOTE: *NOTE: *' *NOTE: *' *NOTE: *' *' *' *' *' *' *' *' *' *' *' *' *'</pre>	*		of the double hir	es and 80 column soft	
<pre>so that talking to the accelerator can't accidently turn on double hires. NOTE: NOTE: 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, V unchanged acve.state rd80col equ SCOIF ; if bit 7 = 1 then 80col is on rddhires equ SCOIF ; if bit 7 = 0 then dhires is on on.80col equ SCOSE ; writing turns on 80 col of.60col equ SCOSE ; writing turns off 80 col on.dhires equ SCOSE ; writing turns off dhires if dhires equ SCOSE ; writing turns off dhires if dhires equ SCOSE ; writing turns off dhires if da rd80col ; see if 80 column is on asl a ; save state ida rd80col ; see if 80 column is on asl a ; save state ida rd61. el unsave.state unsave.state ida rdhires ; see if double hires on php php bcc @1 ; if branch then it's off sta off.80col ; turn it off first unsave.state ida rd0hires ; see if double hires on sta temp ; store result of both tests ; return to caller unsave.state ida temp ; get back the status flags pha php bcc @1 ; branch if 80col should be off ; bccause it already is sta on.80col ; should be on -so turn it on e1 bcc @1 ; branch if dhires to off ; bccause it already is sta on.80col ; should be on -so turn it on e1 bcc %1 done2 ; branch unconditionally g2 sta off.dhires ; restore dhires to off done2 rts temp dfb S00 ; used for temp storing of states</pre>	*		switches. It tur	ns off 80 column mode	
<pre>* 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 * 80Col firmware is turned off * X,Y unchanged * 80Col equ SCOIF ; if bit 7 =1 then 80col is on rddhires equ SCOIF ; if bit 7 = 0 then dhires is on off.80col equ SCOOF ; writing turns on 80 col off.80col equ SCOOF ; writing turns off 80 col on.80col equ SCOSF ; writing turns off dhires off.dhires equ SCOSF ; writing turns off dhires is a save state ida rd80col ; see if 80 column is on asi a ; save state bcc @1 ; if bir 7 is of 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 vreturn to caller unsave.state unsave.state ida rdmires ; store result of both tests rts vreturn to caller unsave.state php ; 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 % bp branch if dhires to on pha ; branch if dhires to off done2 rts temp dfb \$00 ; used for temp storing of states</pre>	*		so that talking to the accolorator conit		
<pre>*NOTE: If you put any calls to the accelerator *NOTE: If you put any calls to 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 * * COSC ; writing turns off 80 col on.80col equ SCOTF ; if bit 7 = 1 then 80col is on rddhires equ SCOTF ; if bit 7 = 0 then dhires is on on.80col equ SCOTF ; if bit 7 = 0 then dhires is on on.80col equ SCOSE ; writing turns off 80 col on.dhires equ SCOSE ; writing turns off 80 col on.dhires equ SCOSF ; if bit 7 = 0 then dhires off.80col equ SCOSF ; writing turns off dhires off.dhoires equ SCOSF ; writing turns off dhires off.dhires equ SCOSF ; if bit 7 = 0 then thires off.dhires equ SCOSF ; writing turns off dhires equ SCOSF ; writing turns off the sta off.80col ; turn it off first @1</pre>	*		accidently turn o	a double hires	
<pre>*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 caccelerator. Entry: nothing Exit: A scrambled BOCol firmware is turned off X,Y unchanged accelerator. Reform a set of the first for 7 = 1 then 80col is on on.80col equ SCOIF ; if bit 7 = 1 then 80col is on on.80col equ SCOIF ; if bit 7 = 0 then dhires is on on.80col equ SCOC ; writing turns of 80 col on.80col equ SCOC ; writing turns of 60 col on.80col equ SCOSE ; writing turns of for set of the state off.dhires equ SCOSE ; writing turns of for thes</pre>	*		accidencity curn o	a double miles.	
at the beginning of your program, then at the beginning of your program, then at the beginning of your program, then sets the 80 column switch and the double hirs switches to the state your program requires after making any calls to the accelerator. exit: A scrambled 80Col firmware is turned off x, y unchanged save.state rd80col equ SCOIF ; if bit 7 =1 then 80col is on rd80col equ SCOFF ; if bit 7 = 0 then dhires is on on.80col equ SCOCC ; writing turns of 80 col off.80col equ SCOSE ; writing turns of 60 col off.80col equ SCOSE ; writing turns of 61 dhires off.80col equ SCOSE ; writing turns of 61 dhires off.80col i; turn it off first el id rd80col ; see if 80 column is on asl a ; save state bcc @1 ; if branch then it's off sta off.80col ; turn it of first @1 Ida rd80col ; see if double hires on php ; save c and n flags on stack pla ; save c and n flags on stack pla ; sta off.80col ; turn it of first @1 Ida rd0hires ; reetif double hires on php	*NOTE •		If you put any ca	ills to the accelerator	
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 Entry: nothing Exit: A scrambled 80Coll firmware is turned off X,Y unchanged save.state rd80col equ rd80col equ equ SCOIF ; if bit 7 =1 then 80col is on rddhires equ SCOIF ; if bit 7 =0 then dhires is on on.80col equ SCOIF ; if bit 7 =0 then dhires is on on.80col equ SCOOD ; writing turns on 80 coll off.80col equ SCOSE ; writing turns off dhires off.80col equ sta off.80col ; if bit 7 =1 then 80col is on rddhires if da rd80col ; writing turns off doi if bit 7 off.80 col if bi	*		BEFORE you sot th	o state of the machine	
act of beginning of you program, thisthis routhewill not be necessary. Simplymake sure your initialization routinesets the 80 column switch and the doublehires switches to the state your programrequires after making any calls to theaccelerator.Entry:nothingExit:A scrambled80Col firmware is turned offX,Y unchangedsave.staterddhiresequscoleyif bit 7 =1 then 80col is onrddhiresequscoleywriting turns on 80 coloff.80colequscoleywriting turns on 80 coloff.80colequscoleywriting turns off 80 colon.ahiresequscoleywriting turns off 80 colon.dhiresequscoleywriting turns off 80 coloff.dhiresequscoleywriting turns off 80 coloff.dhiresequscoleysave stateldard80colstaoff.80colstaoff.80colstaoff.80coliturn it off first@1ldardbiresysave cand n flags on stackplaysave cand n flags on stackplaysave cand n flags back in P regbcc@1ybpyeat status flags back in P regplpyeat status flags back in P regbcc@1plpyeat status flags back in P regbcc@1bcc%1 <t< td=""><td>*</td><td></td><td>at the beginning</td><td>of your program then</td></t<>	*		at the beginning	of your program then	
<pre>inits fold the will not be necessary. Simply instruction of the start over function of the start of the</pre>	*		at the beginning	of your program, chen	
make site 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 rddhires rddnires equ SCOIF ;if bit 7 =1 then 80col is on on.80col equ SCOIF ;if bit 7 = 0 then dhires is on on.80col equ equ SCOOD ;writing turns on 80 col off.80col equ scoSFF ;writing turns off 80 col on.dhires equ sco equ ida rd80col asl a asl asl ast off.80col ;save state bcc ell ida rd80col ;save state bcc ell ista off.80col ;save c and n flags on stac			This fourine will	itialization routing	
sets the 30 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 SCOIF ; if bit 7 =1 then 80col is on rddhires rd80col equ SCOIF ; if bit 7 =0 then dhires is on on.80col off.80col equ SCOOC ; writing turns off 80 col on.dhires off.80col equ SCOOC ; writing turns off dhires off.80col equ SCOOC ; writing turns off dhires off.80col equ SCOOF ; writing turns off dhires off.80col ; see if 80 column is on asl a ; save state bcc 01 ; fbranch then it's off sta ; save c and n flags on stack pla rsture result of both tests ; return to caller unsave.state ida temp ; get back the status flags pha ; sta on.80col ; should be on - so turn it on plp ; branch if 80col should be off ; branch if 80col should be off gla sta on.80col ; breatore dhires to on	- -		make sure your in	icialization routine	
<pre>nires suitches to the state your program requires after making any calls to the accelerator. Entry: nothing </pre>	<u>^</u>		sets the 80 colum	in switch and the double	
<pre>requires after making any calls to the accelerator. Entry: nothing Exit: A scrambled BOCol firmware is turned off X,Y unchanged ave.state rd80col equ SCOIF ; if bit 7 =1 then 80col is on rddhires equ SCOTF ; if bit 7 =0 then dhires is on on.80col equ SCOOD ; writing turns off 80 col on.dhires equ SCOSE ; writing turns off 80 col on.dhires equ SCOSF ; writing turns off dhires off.dhires equ SCOSF ; writing turns off dhires ida 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 da rddhires ; see if double hires on php ; save c and n flags on stack pla sta temp ; get back the status flags pha plp ; want status flags back in P reg bcc @1 ; branch if 80 col should be off ; branch if 80col should be off ; branch if dhires is off sta on.dhires ; restore dhires to on bpl done2 ; branch unconditionally 62 sta off.dhires ; restore dhires to off done2 rts</pre>	•		hires switches to the state your program		
<pre>accelerator. Entry: nothing Exit: A scrambled 80Col firmware is turned off X,Y unchanged aave.state rd8Dcol equ SCOIF ; if bit 7 =1 then 80col is on rddhires equ SCOTF ; if bit 7 =0 then dhires is on on.80col equ SCOD ; writing turns on 80 col of.60col equ SCOC ; writing turns of 80 col on.dhires equ SCOSE ; writing turns of dhires off.dhires equ SCOSF ; writing turns off dhires ida 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 equ scose ; writing turns on dhires ida rd80col ; 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 unsave.state ida temp ; get back the status flags pha pla ; want status flags back in P reg bcc @1 ; branch if dhires is off ; bccause it already is sta on.80col ; should be off ; bccause it already is sta on.80col ; branch if dhires to off bpl done2 ; branch unconditionally g2 sta off.dhires ; restore dhires to off done2 rts temp dfb \$00 ; used for temp storing of states </pre>	*		requires after ma	king any calls to the	
Entry: nothing Exit: A scrambled BCOL firmware is turned off X,Y unchanged Td80col equ SCOIF ; if bit 7 =1 then 80col is on rddhires equ SCOTF ; if bit 7 = 0 then dhires is on on.80col equ SCOOD ;writing turns on 80 col off.80col equ SCOCC ;writing turns of 80 col on.dhires equ SCOSE ;writing turns off 80 col on.dhires equ SCOSF ;writing turns off dhires off.80col equ SCOSF ;writing turns off dhires ida 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 rd8hires ;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 ;bcc @1 ;branch if 80col should be off ;because it already is sta on.80col ;should be off ;because it already is sta on.80col ;branch if dhires to off bd done2 ;rts temp dfb \$00 ;used for temp storing of states	*		accelerator.		
<pre>Entry: nothing Exit: A scrambled B0Col firmware is turned off X,Y unchanged Bayes State FrdBOcol equ SCOIF ; if bit 7 = 1 then 80col is on rddhires equ SCOIF ; if bit 7 = 0 then dhires is on on.80col equ SCOOD ; writing turns on 80 col off.80col equ SCOOC ; writing turns of 80 col on.dhires equ SCOSE ; writing turns off dhires off.dhires equ SCOSF ; writing turns off dhires Ida rd80col ; see if 80 column is on asl a ; save state Ida rd80col ; turn it off first el Ida 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 Ida 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 bpl done2 ; tras temp dfb \$00 ; used for temp storing of states </pre>	*	_			
Exit: A scrambled BOCol firmware is turned off X,Y unchanged save.state rd80col equ SCOIF ; if bit 7 =1 then 80col is on rddhires equ SCOTF ; if bit 7 =0 then dhires is on on.80col equ SCOOD ; writing turns of 80 col on.dhires equ SCOSE ; writing turns off 80 col on.dhires equ SCOSF ; writing turns off dhires off.dhires equ SCOSF ; writing turns off dhires ida 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 unsave.state unsave.state ida temp ; 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 ida temp ; get back the status flags pha ; bcc @1 ; branch if 80col should be off ; because it already is sta on.80col ; should be of - so turn it on §1 bmi @2 ; branch if dhires to on bpl done2 ; tts temp dfb \$00 ; used for temp storing of states	*	Entry:	nothing		
<pre>Exit: A scrambled</pre>	*				
 80Ccl firmware is turned off X,Y unchanged rd80col equ SCOIF ; if bit 7 =1 then 80col is on rddhires equ SCOOF ; if bit 7 =0 then dhires is on on.80col equ SCOOC ; writing turns of 80 col off.80col equ SCOSE ; writing turns off 80 col on.dhires equ SCOSE ; writing turns off dhires Ida rd80col ;see if 80 column is on asl a ;save state bcc @l ; if branch then it's off sta off.80col ;turn it off first Ida rd80col ;turn it off first g1 Ida rd80col ;turn it off first g2 g4 g5 g5 g6 g7 g8 g8 g9 <li< td=""><td>*</td><td>Exit:</td><td>A scrambled</td><td></td></li<>	*	Exit:	A scrambled		
<pre>* X,Y unchanged ************************************</pre>	*		80Col firmware is	turned off	
<pre>save.state rd80col equ SCOIF ;if bit 7 =1 then 80col is on rddhires equ SCOOF ;if bit 7 = 0 then dhires is on on.80col equ SCOOC ;writing turns on 80 col off.80col equ SCOOC ;writing turns off 80 col on.dhires equ SCOSE ;writing turns off dhires off.dhires equ SCOSF ;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 abould 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 off.dhires ;restore dhires to on bpl done2 ;branch if dhires to off done2 rts temp dfb \$00 ;used for temp storing of states</pre>	*		X,Y unchanged		
save.staterd80colequ\$CO1F; if bit 7 =1 then 80col is onrddhiresequ\$CO7F; if bit 7 = 0 then dhires is onon.80colequ\$C00D; writing turns on 80 coloff.80colequ\$C00C; writing turns on dhiresoff.dhiresequ\$C05E; writing turns off 80 colon.dhiresequ\$C05F; writing turns off dhiresoff.dhiresequ\$C05F; writing turns off dhiresidard80col; see if 80 column is onasla; save statebcc@1; if branch then it's offstaoff.80col; turn it off first@1ldarddhiresistatemp; store result of both testsplastatempstaplastaon.80col; branch if 80col should be offpbcc@1; branch if 80col should be off; bccause it already is; stastaon.dhiresgtastaon.80col; branch if dhires is offgtastaon.dhires; restore dhires to onbpldone2staoff.dhiresgtastaoff.dhires; restore dhires to off	*******	******	*****	******	
rd80col rddhires equ sc001 equ sc002 off.80col equ sc000 equ sc000 off.80col equ sc000 sc000 switing turns on 80 col switing turns on 80 col switing turns on 80 col switing turns of 80 col on.dhires equ sc005 sc005 switing turns off dhiresIda asl <	save.state				
Table 1equSCOTFfill bit 7 = 0 then address is onrddhiresequSCOOF;writing turns on 80 coloff.80colequSCOOC;writing turns off 80 colon.dhiresequSCOSE;writing turns of dhiresoff.dhiresequSCOSF;writing turns off dhiresldard80col;see if 80 column is onasla;see if 80 column is onasla;see statebcc01;if branch then it's offstaoff.80col;turn it off first01ldarddhiresisattemp;store result of both testsrtsrts;return to callerunsave.stateldatempldatemp;get back the status flagsphaplp;want status flags back in P regbcc01;branch if dhires is offistaon.80col;should be on - so turn it on01bmi02;branch if dhires is offgtstaon.dhires;restore dhires to onbcc01;branch if dhires is offgtastaon.dhires;restore dhires to offgtastaoff.dhires;restore dhires to offgtastaoff.dhires <t< td=""><td>xd00aa1</td><td></td><td>\$COLE</td><td>if hit 7 -1 than Quark is an</td></t<>	xd00aa1		\$COLE	if hit 7 -1 than Quark is an	
radningsequSCOTFill of the of the differs is onon.80colequSCOOD;writing turns on 80 coloff.80colequSCOSE;writing turns on dhiresoff.dhiresequSCOSE;writing turns off dhiresoff.dhiresequSCOSF;writing turns off dhiresidard80col;see if 80 column is onasla;save statebcc01;if branch then it's offstaoff.80col;turn it off first@1ldarddhiresidarddhires;see if double hires onphp;save c and n flags on stackplastatempstatemp;get back the status flagsphaplp;want status flags back in P regbcc01;branch if dhires is offglastaon.80colishould be on - so turn it on%1bmi%2%2staoff.dhires%2staoff.dhires%4staoff.dhires%4staoff.dhires%4staoff.dhires%4staoff.dhires%4staoff.dhires%4staoff.dhires%4staoff.dhires%4staoff.dhires%4staoff.dhires%4staoff.dhires%4staoff.dhires%4staoff.dhires%4staoff.dhires%4 </td <td>1080001</td> <td>equ</td> <td>\$COIF</td> <td>if his 7 - 0 then dotted is on</td>	1080001	equ	\$COIF	if his 7 - 0 then dotted is on	
off.80colequSCUUDFwiting turns of a 80 coloff.80colequSCOC;writing turns of a 80 colon.dhiresequSCOSE;writing turns of dhiresoff.dhiresequSCOSF;writing turns off dhiresldard80col;see if 80 column is onasla;save statebcc01;if branch then it's offstaoff.80col;turn it off first01ldarddhiresidarddhires;see if double hires onphp;save c and n flags on stackpla;statempstatemp;store result of both testsrts'yet back the status flagspha;bcc01plp;want status flags back in P regbcc01;should be on - so turn it on(1bmi02(2staon.dhiresistaon.dhiresistaoff.dhiresistaoff.dhiresistaoff.dhiresistaoff.dhiresistaoff.dhiresistaoff.dhiresistaoff.dhiresistaoff.dhiresistaoff.dhiresistaoff.dhiresistaoff.dhiresistaoff.dhiresistaoff.dhiresistaoff.dhiresistaoff.dhiresistaoff.dhiresistaoff.dhiresistaoff.dhiresistaoff.dhires	radnires	equ	\$COTF	; II bit / = 0 then dhires is on	
off.80colequSCUCC/Writing turns off 80 colon.dhiresequSCOSE;writing turns off dhiresoff.dhiresequSCOSF;writing turns off dhiresldard80col;see if 80 column is onasla;save statebcc@l;if branch then it's offstaoff.80col;turn it off first@lldarddhirespla;save c and n flags on stackplastatempstatemppla;store result of both testsrts;return to callerunsave.state;want status flags back in P regbcc@lpla;bccause it already isstaon.80col;bccause it already isstaon.dhiresstaon.dhiresgetstastaoff.dhiresgetsta </td <td>on.80C01</td> <td>equ</td> <td>\$0000</td> <td>Writing turns on 80 col</td>	on.80C01	equ	\$0000	Writing turns on 80 col	
on.dhiresequSCOSE;writing turns on dhiresoff.dhiresequSCOSF;writing turns off dhiresidard80col;see if 80 column is onasla;save statebcc@1;if branch then it's offstaoff.80col;turn it off first@1idarddhires;see if double hires onphp;save c and n flags on stackplastatempstatemp;store result of both testsrtsrts;return to callerunsave.stateidatempgl;want status flags back in P regbcc@1;branch if 80Col should be off;because it already isstastaon.80colstaon.dhiresstaon.dhiresytanch if dhires is offstaoff.dhiresstaoff.dhiresgldone2tempdfb\$00;used for temp storing of states	off.80col	equ	\$C00C	writing turns off 80 col	
off.dhiresequSCOSF;writing turns off dhiresIdard80col;see if 80 column is onasla;save statebcc@l;if branch then it's offstaoff.80col;turn it off first@lIdarddhiresldarddhires;see if double hires onphp;save c and n flags on stackplastatempstatemp;store result of both testsrts;return to callerunsave.stateidatemppha;want status flags back in P regbcc@l;branch if 80Col should be offpbc@l;branch if dhires is offstaon.80col;should be on - so turn it on@lbmi@2bpldone2;branch if dhires to onbpldone2;branch unconditionally@2staoff.dhirestempdfb\$00;used for temp storing of states	on dhires	equ	\$C05E	writing turns on dhires	
Idard80col;see if 80 column is on ;save statebcc@1;if branch then it's off ;turn it off first@1Idarddhires;see if double hires on ;save c and n flags on stack@1Idarddhires;see if double hires on ;save c and n flags on stackunsave.stateidatemp;store result of both tests ;return to callerunsave.stateidatemp;get back the status flags pha plpinsave.stateidatemp;get back the status flags pha plp(1)if 80col;bocolstatus flags back in P reg ;bcc(2)staon.80col;should be on - so turn it on plid(2)staon.dhires;restore dhires to on plid(2)staoff.dhires;restore dhires to off(2)staoff.dhires;restore dhires to off(2)staoff.dhires;restore dhires to off(3)done2;branch unconditionally(2)staoff.dhires(3)statestate(4)\$00;used for temp storing of states	off.dhires	equ	\$C05F	;writing turns off dhires	
asla;see if do binn is onasla;see statebcc@l;if branch then it's offstaoff.80col;turn it off first@lIdarddhires;see if double hires onphp;save c and n flags on stackplastatempstatemp;store result of both testsrts'return to callerunsave.stateidatempldatemp;get back the status flagsphaplp;branch if 80Col should be offbcc@l;branch if 80Col should be offstaon.80col;should be on - so turn it on@lbmi@2staon.dhires;restore dhires to onbpldone2;branch unconditionally@2staoff.dhirestempdfb\$00;used for temp storing of states		lda	rdBOcol	see if 80 column is on	
ds1d,save statebcc@l;if branch then it's offstaoff.80col;turn it off first@lldarddhires;see if double hires onphp;save c and n flags on stackplastatempstatemp;store result of both testsrts;return to callerunsave.stateidatempunsave.state;get back the status flagsbcc@l;branch if 80Col should be offjbp;want status flags back in P regbcc@l;branch if 80Col should be offstaon.80col;should be on - so turn it on@lbmi@2staoff.dhires;restore dhires to onbpldone2;branch unconditionally@2staoff.dhirestempdfb\$00;used for temp storing of states			1080001		
bcc@1;if branch then it's offg1staoff.80col;turn it off first@1idarddhires;see if double hires onphp;save c and n flags on stackplastatempstatemp;store result of both testsrts;return to callerunsave.stateidatempinsave.state;get back the status flagsbcc@1;branch if 80Col should be offbcc@1;branch if 80Col should be offbcc@1;branch if dhires is offstaon.80col;should be on - so turn it on@1bmi@2bpldone2;branch unconditionally@2staoff.dhirestempdfb\$00;used for temp storing of states		ası	a	Save State	
staoff.80col;turn it off first@1Idarddhires;see if double hires on ;save c and n flags on stack pla staplastatemp;store result of both tests ;return to callerunsave.stateIdatemp;get back the status flags pha plpunsave.stateidatemp;get back the status flags bccldatemp;get back the status flags bccin P reg ;branch if 80Col should be off ;branch if dhires is off sta on.dhires@1bmi@2;branch if dhires is off sta off.dhires@2staoff.dhires;restore dhires to on bpl done2tempdfb\$00;used for temp storing of states		bcc	61	; if branch then it's off	
@1Ida php pla starddhires (see if double hires on php) (save c and n flags on stack pla staunsave.stateida lda temp pha plp bcc(store result of both tests (return to caller)unsave.stateida lda temp plp bcc(get back the status flags (status flags back in P reg (bcc)(la lda plp bcc(el (branch if 80Col should be off (because it already is (status on.80col) (should be on - so turn it on bpl done2 (branch if dhires is off (status off.dhires) (restore dhires to on (bpl) done2 (rts)(el ldone2 temp(ff.dhires) (status) (status)(ff.dhires) temp(states)		sta	off.80col	;turn it off first	
Idarddnires;see if double hires on phpphp;save c and n flags on stackplastastatempits;store result of both testsrts;return to callerunsave.stateidaldatemppha;want status flags back in P regbcc@1bcc@1its;should be on - so turn it on@1bmibpidone2bpidone2its;restore dhires to onbpidone2tempdfb\$00;used for temp storing of states	@1				
<pre>php ;save c and n flags on stack pla sta temp ;store result of both tests rts ;return to caller unsave.state unsave.state unsave.state unsave.state ida 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</pre>		Ida	rddhires	;see if double hires on	
pla statemp;store result of both tests ;return to callerunsave.stateidatempidatemp;get back the status flags pha plpjpp;want status flags back in P reg bcc%want status flags back in P reg ;bcause it already is@1staon.80col@1bmi@2if staon.dhiresif staon.dhiresif staoff.dhiresif staoff.dhiresif done2rtstempdfb\$00;used for temp storing of states		php		;save c and n flags on stack	
statemp;store result of both testsunsave.stateIdatemp;return to callerunsave.stateIdatemp;get back the status flagsphaplp;want status flags back in P regbcc@1;branch if 80Col should be offstaon.80col;should be on - so turn it on@1bmi@2bpldone2;branch if dhires is offstaon.dhires;restore dhires to onbpldone2;branch unconditionally@2staoff.dhirestempdfb\$00		pla			
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		sta	temp	store result of both tests;	
unsave.stateIda pha plp bcctemp;get back the status flags pla ;want status flags back in P reg bcc%%%		rts		;return to caller	
unsave.stateIdatemp;get back the status flagsphaplp;want status flags back in P regbcc@1;branch if 80Col should be offbcc@1;branch if 80Col should be offstaon.80col;should be on - so turn it on@1bmi@2bpldone2;branch if dhires is offbpldone2;branch unconditionally@2staoff.dhiresdone2rts;used for temp storing of states					
Idatemp;get back the status flagsphaplp;want status flags back in P regbcc@1;branch if 80Col should be off;bcc@1;branch if 80Col should be offget backisstastaon.80col;should be on - so turn it on@1bmi@2;branch if dhires is offstaon.dhires;restore dhires to onbpldone2;branch unconditionally@2staoff.dhiresdone2rts;used for temp storing of states	unsave.state				
pha;want status flags back in P regbcc@1;bcause it already isstaon.80col;should be on - so turn it on@1bmibpldone2bpldone2clauseit offdone2rtstempdfb\$00;used for temp storing of states		Ida	temp	;get back the status flags	
plp;want status flags back in P reg ;branch if 80Col should be off ;because it already is should be on - so turn it on@1bmi@2;branch if dhires is off sta on.dhires ;restore dhires to on bpl@2staoff.dhires ;restore dhires to off@2staoff.dhires ;restore dhires to offdone2rtstempdfb\$00;used for temp storing of states		pha			
bcc@1; branch if 80Col should be off ; because it already isstaon.80col; should be on - so turn it on@1bmi@2; branch if dhires is offstaon.dhires; restore dhires to onbpldone2; branch unconditionally@2staoff.dhirestempdfb\$00; used for temp storing of states		plp		;want status flags back in P reg	
<pre>sta on.80col ; because it already is ; because it already is ; 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</pre>		bcc	01	;branch if 80Col should be off	
staon.80col; should be on - so turn it on@1bmi@2; branch if dhires is offstaon.dhires; restore dhires to onbpldone2; branch unconditionally@2staoff.dhiresdone2rtstempdfb\$00; used for temp storing of states				;because it already is	
@1bmi@2; branch if dhires is off stastaon.dhires; restore dhires to on bplbpldone2; branch unconditionally@2staoff.dhiresdone2rts		sta	on.80col	;should be on - so turn it on	
staon.dhires;restore dhires to onbpldone2;branch unconditionally@2staoff.dhires;restore dhires to offdone2rts;used for temp storing of states	01	bmi	@2	;branch if dhires is off	
bpldone2;branch unconditionally@2staoff.dhires;restore dhires to offdone2rts		sta	on.dhires	;restore dhires to on	
@2staoff.dhires; restore dhires to offdone2rtstempdfb\$00; used for temp storing of states		bpl	done2	;branch unconditionally	
done2rtstempdfb\$00;used for temp storing of states	02	sta	off.dhires	;restore dhires to off	
temp dfb \$00 ;used for temp storing of states	done2	rts			
temp dfb \$00 ;used for temp storing of states					
	temp	dfb	\$00	;used for temp storing of states	

*****	********	****	*****
*unlock:	Unlocks t	the accelerator so	it can be accessed. This
*	must be d	done before anythin	ng else.
*			
*NOTE:	Please no	ote that the state	of some soft switches
*	will be a	affected by this ca	all. (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	e accelerator so th	nat normal system
*	use cannot affect it. This must be the last		
*	call made	e to the accelerate	pr.
×			
*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

638 Apple IIc Technical Reference

* * * * * * * * * * * * * * *	*******	*****	******	
*Wait.ram: *	This replaces the ROM version of the wait routine			
*	ir your j	program speeds up	port z.	
*NOTE:	Your prod	gram must call thi	s routine and not the one in	
*	ROM. You	u may put this in	the language card and disable	
*	the ROM :	if you are not usi	ng any other ROM routines.	
×				
*NOTE:	This rout	tine will run corr	ectly at either fast or normal	
*	speed.	It can also be run	on any other version of the IIc	
*	or IIe w	ithout harm.		
*				
*Entry:	A=500 - SFF depending on the amount of time to wait:			
*	min detay = $1/2(50+25A+5A^2)+29$			
*	and in most cases is exactly the same.			
*	This rout	tine has A-12 fewe	r cycles than the wait routine	
*	in ROM.		- ofoice than the walt footing	
*	X,Y,P und	defined		
*				
*Exit:	X,Y uncha	anged		
*	A=\$00			
*	P undefined			
*****	*******	******	******	
wait.ram				
kbd	equ	\$C000	;address offset for port I/O	
	phy		;save X and Y	
	ldy	#\$D0	;\$CODO is guaranteed 50 ms slow	

	Tay	#\$D0	;\$CUDU is guaranteed 50 ms slow
	phx		;on IIc Plus but won't hurt other
	sec		;IIc or IIe's
	txa		;save wait value
01			
	lda	kbd,Y	;this starts the slow down
	txa		;new version of wait routine
@2			
	sòc	#\$01	;min delay = 1/2(50+25A+5A^2)+29
	bne	02	;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		

APPENDIX H Controlling the Apple IIc Plus Accelerator 639