Apple IIGS Computer Information

Controlling the 3.5 Drive Hardware on the Apple IIGS

http://wiretap.area.com/Gopher/Library/Techdoc/Hardware/iwmiigs.txt
22 January 2006

Finally!

Many months ago, I inquired about the level of interest in some technical information about various bits of Apple IIGS hardware. There was a very positive response, especially for the information on the 3.5 Drive, so I sat down and started writing. Unfortunately, I never really managed to finish it, and the project soon got shoved onto one of my numerous back burners, where it sat essentially untouched until recently.

Anyway, I finally decided I'd wasted enough time, so here it is. Enjoy. Please feel free to contact me if you have any questions, comments, corrections, etc...

Controlling the 3.5 Drive Hardware on the Apple IIGS
Part 1 of 2: Lotsa Really Confusing Information

By Neil Parker

First, the standard Dire Warnings:

The following article is based on information found in several publications (listed at the end of this article), my own disassemblies of the relevant Apple IIGS ROM routines, and on some experimentation. I can make no guarantees as to the accuracy of this information--it should probably be considered as a starting point for your own explorations rather than as an authoritative source.

Remember that when you use this information you're dealing directly with the Naked Hardware, and the myriad protective features of the firmware and operating system are not available. Should you be so foolish as to try out this information with a non-expendable disk in the drive, I won't be held responsible for any lost data.

(End of Dire Warnings.)

A note about machine code: All the sample routines in this article

```
      RTS
```

Note that in the write routine, the first byte is written differently than the rest--the STA Q7+1 activates write mode and writes the byte all in one step.

In actual practice, you would probably want to use a loop to read and store (or load and write) the data.

In addition to programming the IWM, it is also necessary to program the drive itself, which is somewhat "smarter" than the 5.25-inch drive (even though it's a "dumb" device).

The 3.5-inch drive contains several internal status bits which the user's program can examine, and several internal control switches which the user's program can use to control various functions of the drive. These status and control bits are accessed by the CA0...LSTRB switches mentioned above and by the SEL line (bit 7 of DISKREG). CA0...CA2 and SEL form a 16-way switch which selects the desired control or status function, and the LSTRB switch signals the drive to perform a control function. The IIGS ROM uses the following routine to select a status or control function (enter with desired function in A-reg):

```
SEL35 BIT CA0           ;set switches to known state
      BIT CA1+1
      BIT LSTRB
      BIT CA2
      LSR
      BCC SEL35A
      BIT CA2+1   ;if bit 0 on, turn on CA2
SEL35A      LSR
      PHA
      LDA DISKREG
      AND #$7F    ;if bit 1 off, turn off SEL
      BCC SEL35B
      ORA #$80    ;else turn on SEL
SEL35B      STA DISKREG
      PLA
      LSR
      BCC SEL35C
      BIT CA0+1   ;if bit 2 on, turn on CA0
SEL35C      LSR
      BCS SEL35D
      BIT CA1           ;if bit 3 off, turn off CA1
SEL35D      RTS
```

To read a status bit, turn Q6 off, Q7 on, and ENABLE on, configure CA0...CA2 and SEL for the desired function, and read the status bit from bit 7 of the IWM status register. The IIGS ROM uses the following code to accomplish this:

```
STAT35      JSR SEL35   ;select desired status bit
      BIT Q6+1
```

```
      BIT Q7            ;test status register
      RTS         ;(returns result in processor N-flag)
```

The status bits are as follows:

CA2	CA1	CA0	SEL	Param for STAT35	Function
off	off	off	off	$00	Step direction. 0=head set to step inward (toward higher-numbered tracks), 1=head set to step outward (toward lower-numbered tracks).
off	off	off	on	$02	Disk in place. 0=disk in drive, 1=drive is empty.
off	off	on	off	$04	Disk is stepping. 0=head is stepping between tracks, 1=head is not stepping.
off	off	on	on	$06	Disk locked. 0=disk is write protected, 1=disk is write-enabled.
off	on	off	off	$08	Motor on. 0=spindle motor is spinning, 1=motor is off.
off	on	off	on	$0A	Track 0. 0=head is at track 0, 1=head is at some other track. This bit becomes valid beginning 12 msec after the step that places the head at track 0.
off	on	on	off	$0C	*Disk switched? 0=user ejected disk by pressing the eject button, 1=disk not ejected.
off	on	on	on	$0E	Tachometer. 60 pulses per disk revolution
on	off	off	off	$01	Instantaneous data from lower head. Reading this bit configures the drive to do I/O with the lower head.
on	off	off	on	$03	Instantaneous data from upper head. Reading this bit configures the drive to do I/O with the upper head.
on	on	off	off	$09	Number of sides. 0=single-sided drive, 1=double-sided drive.
on	on	off	on	$0B	*Disk ready for reading? 0=ready, 1=not ready. I'm not too sure about this bit--the firmware waits for this bit to go low before trying to read a sector address field.
on	on	on	on	$0F	Drive installed. 0=drive is connected, 1=no drive is connected.

* Functions marked with an asterisk are used by the IIGS ROM but not documented in any publication available to me. I'm fairly sure about the function of status bit $0C (used by the firmware to test for disk-switched errors), but I'm rather uncertain about status bit $0B (if my programs neglect to test for it, the drive displays an annoying tendency to start reading while the head is still stepping).

Note the the settings of most of these bits are "backwards"--0 means yes and 1 means no.

To perform a control function, turn off LSTRB, configure CA0, CA1, and SEL for the desired function, set CA2 to the desired value (all control functions can be turned on or off), and then turn LSTRB on and back off.

The IIGS ROM uses the following code to accomplish this:

```
CONT35      JSR SEL35   ;select desired function
      BIT LSTRB+1 ;strobe on
      BIT LSTRB   ;strobe off
      RTS
```

The control functions are as follows:

CA1	CA0	SEL	CA2	Param for CONT35	Function
off	off	off	off	$00	Set step direction inward (toward higher-numbered tracks.)
off	off	off	on	$01	Set step direction outward (toward lower-numbered tracks.
off	off	on	on	$03	*Reset disk-switched flag? (The firmware uses this to clear disk-switched errors.)
off	on	off	off	$04	Step one track in current direction (takes about 12 msec).
on	off	off	off	$08	Turn spindle motor on.
on	off	off	on	$09	Turn spindle motor off.
on	on	off	on	$0D	Eject the disk. This takes about 1/2 sec to complete. The drive may not recognize further control commands until this operation is complete.

* Again, the asterisk marks a function used by the ROM but not documented in any publication available to me.

The following is a greatly simplified description of the steps that a simple program might take to I/O with the 3.5-inch drive.

```
     Save SLTROMSEL and CYAREG
     Switch in internal slot 6 and set fast speed
     Turn off disk I/O switches (to insure a "safe" state)
     Select the 3.5-inch drive (turn on bit 6 of DISKREG)
     Set IWM mode register to $0F
     Select drive 1 or 2 (access SELECT or SELECT+1)
     Turn on drive (access ENABLE+1)
     Turn on spindle motor (LDA #$08; JSR CONT35)
     IF we don't know what track we're currently on
     THEN Set step direction=out (LDA #$01; JSR CONT35)
        WHILE Not at track 0 (LDA #$0A; JSR STAT35; BPL ...)
        DO Step one track (LDA #$04; JSR CONT35)
           WHILE still stepping (LDA #$04; JSR STAT35; BPL ...)
           DO nothing
           END WHILE
        END WHILE
        Set current track=0
     END IF
     IF current track < desired track
     THEN Set step direction=in
```

```
      Set number of steps= desired track - current track
   ELSE IF current track > desired track
   THEN Set step direction=out
      Set number of steps= current track - desired track
   ELSE Set number of steps=0
   END IF
   WHILE number of steps > 0
   DO Step one track
    WHILE still stepping
    DO nothing
    END WHILE
    number of steps -= 1
   END WHILE
   Set current track= desired track
   Select desired side (LDA #$01 or LDA #$03; JSR STAT35)
   WHILE not ready to read (LDA #$0B; JSR STAT35; BMI ...)
   DO nothing
   END WHILE
   Read or write your data (this is the FUN part!)
   Turn off spindle motor (LDA #$09; JSR CONT35)
   Turn off drive (LDA ENABLE)
   Turn off CA0...LSTRB
   Set IWM mode register to $00
   Deselect 3.5 drive (turn off bit 6 of DISKREG)
   Restore slot and speed configuration
   Return to caller
```

You will probably notice that I glossed over the most important part--the "read or write your data" part. The basic method is to use routines like those listed above under the description of the IWM data register. Unfortunately, the data must undergo considerable preparation before writing and after reading. Those of you who are lucky enough to own a copy of _Beneath_Apple_DOS_ will understand the kind of work that is necessary. For those not so lucky, I must plead that a proper discussion would require another article every bit as long as this one. Rather than try to tackle that subject here, I will content myself with providing a sample program (with commented source code) which shows one way the above information can be put together to make a working program. I'll be posting it in a separate article a few minutes after this one.

REFERENCES:

Apple Computer, Inc., _Apple_IIGS_Firmware_Reference_. This contains a lengthy description of the SmartPort firmware, which includes some clues as to the functioning of the 3.5 Drive hardware and a diagram of the layout of an individual block of data. You will also need Apple IIGS Technical Note 25, which contains some error corrections.

Apple Computer, Inc., _Apple_IIGS_Hardware_Reference_. This contains a description of the disk interface register (DISKREG, $C031) and the internal registers of the IWM chip. You will also need Apple IIGS Technical Note 30, which corrects numerous errors in the IWM descriptions.

Apple Computer, Inc., _Inside_Macontosh,_Volume_III_. This contains a description of most of the 3.5 Drive status and control bits.

Apple Computer, Inc., _Macintosh_Family_Hardware_Reference_. The 3.5 Drive information from Inside Macintosh is also reprinted in this book (in several different locations).

Don Worth and Pieter Lechner, _Beneath_Apple_DOS_, Quality Software, Reseda, CA, 1981. This is THE classic reference for anything and everything having to do with DOS 3.3 and the 5.25 Drive hardware. Although the 3.5 Drive is a much more complex and powerful device, and uses a slightly different data format, much of the low-level information in this book is still quite relevant.

Don Worth and Pieter Lechner, _Beneath_Apple_ProDOS_, Reston Publishing Company, Reston, VA, 1984. This does for ProDOS what _Beneath_Apple_ _DOS_ did for DOS 3.3. It contains a somewhat abbreviated version of the previous volume's description of low-level formatting, and in addition offers some valuable information on the functioning of the disk interface hardware.

- Neil Parker

(Opinions are those of the author, and do not necessarily represent the opinions of anybody else anywhere.)

--
Neil Parker
parker@corona.uoregon.edu
parkern@jacobs.cs.orst.edu

No cute ASCII art...no cute quote...no cute disclaimer...no deposit, no return... (This space intentionally left blank:)

Controlling the 3.5 Drive Hardware on the Apple IIGS
Part 2 of 2: A real live working program!

By Neil Parker

The program listed below was written to illustrate the steps necessary to control the hardware of the 3.5 Drive from your own programs, without the use of the operating system or the firmware. It is essentially a 3.5-inch version of the DUMP program by Don Worth which was printed in "Beneath Apple DOS." It will read a track from a 3.5-inch disk into your Apple's memory, in its raw, encoded form.

Included below are a commented source code listing and a hex dump suitable for typing directly into the System Monitor (or capturing into a text file and EXECing).

Instructions:

First, boot DOS 3.3 or ProDOS 8. DUMP3.5 should be compatible with either operating system. If you booted ProDOS, get into BASIC.SYSTEM. When you see the] prompt, type "BLOAD DUMP3.5", and then "CALL-151". Store the number of the track you wish to examine in memory location 6, and the disk side you wish to examine (0 for the lower side, anything else for the upper side) in location 7. Put the disk to be examined in Drive 1, and type "900G". The raw track data will then be found in

memory locations $1000 through $7FFF (this buffer is much longer than an actual track, so the data will most likely be repeated several times in the buffer).

For example,

```
]BLOAD DUMP3.5      (Load the program)
]CALL-151           (Enter the Monitor)
*6:20               (Select track $20)
*7:1                (Select upper side)
*900G               (Run DUMP3.5 (don't forget to insert the disk first))
*1000.10FF          (Examine the first 256 bytes of the track)
```

The usual Dire Warnings apply: I make no guarantees whatsoever for this program. I have tested it, and it seems to work on my computer, but I recommend using it ONLY on expendable disks, and ONLY with the write-protect hole open. I assume no responsibility for any damage which may result from the use or misuse of this program.

Be especially careful if you enter either the assemby listing or the hex dump by hand--the slightest typographical error could turn a benign tool into a malevolent disk-eating monster.

I hope this program helps clarify the disk access process. If there is enough interest in an explanation of how to interpret what it accesses, it might be possible to talk me into writing up an explanation of the block encoding process. (I recommend first reading "Beneath Apple DOS" (if you can find a copy), and also the SmartPort chapter of the Firmware Reference.)

- Neil Parker

```
;*********************************************************************
; DUMP3.5--Dump a track of a 3.5-inch disk to memory.  (IIGS only)
;
; By Neil Parker--inspired by Don Worth's DUMP program from "Beneath
; Apple DOS"
;
; Inputs: $06 = Track to be dumped
;         $07 = Side to be dumped (0=lower side, non-0=upper side)
; Outputs: $1000-$7FFF = raw track data
;
; Example:
;     *6:20 1      (Select track $20, upper side)
;     *900G        (Run DUMP3.5)
;     *1000.10FF   (Examine part of the track)
;*********************************************************************
        ORG $900
TRACK     EQU 6           ;Track number
SIDE      EQU 7           ;Side number
PTR       EQU 8
BUFFER    EQU $1000       ;Start address for track data
SLTROMSEL EQU $C02D       ;Select internal/external ROMs for slots
DISKREG   EQU $C031       ;Select 3.5/5.25 drive, control SEL line
```

```
CYAREG    EQU $C036       ;System speed and motor-on-detect bits
CA0       EQU $C0E0       ;Phase 0, 3.5 drive control
CA1       EQU $C0E2       ;Phase 1, 3.5 drive control
CA2       EQU $C0E4       ;Phase 2, 3.5 drive control
LSTRB     EQU $C0E6       ;Phase 3, control strobe
ENABLE    EQU $C0E8       ;Turn drive off/on
SELECT    EQU $C0EA       ;Select drive 1/2
Q6        EQU $C0EC
Q7        EQU $C0EE
;
          LDA SLTROMSEL   ;Get slot 6 status,
          PHA             ;save it,
          AND #$BF        ;force internal ROM+I/O for Slot 6
          STA SLTROMSEL
          LDA CA0         ;Clear disk I/O latches
          LDA CA1
          LDA CA2
          LDA LSTRB
          LDA ENABLE      ;Insure that drive is off
          LDA SELECT      ;Select drive 1
          LDA Q6          ;Set IWM for reading (a "safe" state)
          LDA Q7
          LDA #$F         ;Configure IWM for 3.5 access
          JSR SELIWM
          LDA DISKREG     ;Save old DISKREG
          PHA
          ORA #$40        ;Select 3.5 drive
          STA DISKREG
          LDA ENABLE+1    ;Turn drive on
          LDA #2          ;Is there a disk in the drive?
          JSR SEL35
          JSR TEST35
          BPL THERE       ;If so, read
          JMP DONE        ;otherwise quit
THERE     LDA #8          ;Turn motor on
          JSR SEL35
          JSR TRIG35
          LDA #1          ;Set step direction=outward
          JSR SEL35
          JSR TRIG35
TSTTRK0   LDA #$A         ;Are we at track 0 yet?
          JSR SEL35
          JSR TEST35
          BPL ATTRK0      ;If so, go read
          LDA #4          ;otherwise do a step
          JSR SEL35
          JSR TRIG35
SEEKING0  JSR TEST35      ;Step still in progress?
          BPL SEEKING0    ;If so, loop until step done
          BMI TSTTRK0     ;otherwise go see if we're at track 0 yet
ATTRK0    LDX TRACK       ;What track did the user want?
          BEQ DUMP        ;If track 0, we're already there--go read
          LDA #0          ;else set step direction=inward
          JSR SEL35
```

```
          JSR TRIG35
SEEK      LDA #4          ;Do a step
          JSR SEL35
          JSR TRIG35
SEEKING   JSR TEST35      ;Step still in progress?
          BPL SEEKING     ;If so, loop until step done
          DEX             ;otherwise see if we've stepped enough yet
          BNE SEEK        ;If not, go step again
DUMP      LDA #$B         ;Disk ready for reading yet?
          JSR SEL35
READYT    JSR TEST35
          BMI READYT      ;Loop until disk ready
          LDA SIDE        ;What side did the user want?
          BEQ SIDE1       ;If 0, set lower side
          LDA #3          ;else set upper side
          BNE SETSIDE
SIDE1     LDA #1
SETSIDE   JSR SEL35
          JSR TEST35
          PHP             ;Save interrupt status
          SEI             ;Don't let anything interrupt us
          LDA CYAREG      ;Save old system speed
          PHA
          AND #$FB        ;Set speed=fast
          ORA #$80
          STA CYAREG
          LDA #<BUFFER    ;Initialize read buffer
          STA PTR         ;(NOTE:  For DOS TOOL KIT or EDASM assembler,
          LDA #>BUFFER    ;change #> to #< and #< to #>.)
          STA PTR+1
          LDY #0
DUMPLP    LDA Q6          ;Read a byte
          BPL DUMPLP      ;Loop until we have a valid byte
          STA (PTR),Y     ;Store byte in buffer
          INC PTR         ;Advance buffer pointer
          BNE DUMPLP
          INC PTR+1
          LDA PTR+1       ;Buffer full yet?
          CMP #$80
          BCC DUMPLP      ;If not, go read some more
          PLA             ;Done.  Restore system speed
          STA CYAREG
          PLP             ;Restore interrupt status
          LDA #9          ;Turn motor off
          JSR SEL35
          JSR TRIG35
DONE      LDA ENABLE      ;Turn drive off
          LDA CA0         ;Clear disk I/O latches
          LDA CA1
          LDA CA2
          LDA LSTRB
          PLA             ;Restore old DISKREG value
          STA DISKREG
        LDA #0          ;Configure IWM for 5.25 access
```

```
          JSR SELIWM
          PLA             ;Restore original slot configuration
          STA SLTROMSEL
          RTS             ;Amen.
;
;Subroutine to select 3.5 drive status/control registers
;Enter with accumulator=desired status:
;     Bit 0=CA2 status
;     Bit 1=SEL status
;     Bit 2=CA0 status
;     Bit 3=CA1 status
;
SEL35     BIT CA0
          BIT CA1+1
          BIT LSTRB
          BIT CA2
          LSR             ;If bit 0 set, turn on CA2
          BCC S35A
          BIT CA2+1
S35A      LSR             ;If bit 1 set, turn on SEL
          PHA
          LDA DISKREG
          AND #$7F
          BCC S35B
          ORA #$80
S35B      STA DISKREG
          PLA
          LSR             ;If bit 2 set, turn on CA0
          BCC S35C
          BIT CA0+1
S35C      LSR             ;If bit 3 set, turn on CA1
          BCS S35D
          BIT CA1
S35D      RTS
;
;Subroutine to read the status of the 3.5 drive
;First call SEL35 to select register to examine
;Result is in processor N (negative) flag
;
TEST35    BIT Q6+1
          BIT Q7
          RTS
;
;Subroutine to perform a 3.5 drive control function
;First call SEL35 to select function to be performed
;
TRIG35    BIT LSTRB+1
          BIT LSTRB
          RTS
;
;Subroutine to configure the IWM chip
;Before calling, make sure drive is OFF!
;Call with accumulator=desired Mode Register value
;     A=$00 for 5.25 drive
```

```
;     A=$0F for 3.5 drive
;
SELIWM    TAY
          BIT Q6+1        ;Prepare to access Mode & Status Regs.
          JMP SELIWM2     ;First see if it's already set like we want it
SELIWM1   TYA
          STA Q7+1        ;Try writing to Mode Reg.
SELIWM2   TYA
          EOR Q7          ;Compare input to Status Reg.
          AND #$1F
          BNE SELIWM1     ;If not the same, try writing again
          BIT Q6          ;else prepare IWM for data
          RTS
```

Here is the hext dump corresponding to the above assembler listing.
This can be entered by hand into the Monitor, or you can capture it into
a text file, put "CALL-151" at the beginning and "3D0G" and "BSAVE
DUMP3.5,A$900,L$145" at the end and EXEC it to create the program.

```
900:AD 2D C0 48 29 BF 8D 2D C0 AD E0 C0 AD E2 C0 AD
910:E4 C0 AD E6 C0 AD E8 C0 AD EA C0 AD EC C0 AD EE
920:C0 A9 0F 20 2E 0A AD 31 C0 48 09 40 8D 31 C0 AD
930:E9 C0 A9 02 20 F2 09 20 20 0A 10 03 4C D5 09 A9
940:08 20 F2 09 20 27 0A A9 01 20 F2 09 20 27 0A A9
950:0A 20 F2 09 20 20 0A 10 0F A9 04 20 F2 09 20 27
960:0A 20 20 0A 10 FB 30 E7 A6 06 F0 18 A9 00 20 F2
970:09 20 27 0A A9 04 20 F2 09 20 27 0A 20 20 0A 10
980:FB CA D0 F0 A9 0B 20 F2 09 20 20 0A 30 FB A5 07
990:F0 04 A9 03 D0 02 A9 01 20 F2 09 20 20 0A 08 78
9A0:AD 36 C0 48 29 FB 09 80 8D 36 C0 A9 00 85 08 A9
9B0:10 85 09 A0 00 AD EC C0 10 FB 91 08 E6 08 D0 F5
9C0:E6 09 A5 09 C9 80 90 ED 68 8D 36 C0 28 A9 09 20
9D0:F2 09 20 27 0A AD E8 C0 AD E0 C0 AD E2 C0 AD E4
9E0:C0 AD E6 C0 68 8D 31 C0 A9 00 20 2E 0A 68 8D 2D
9F0:C0 60 2C E0 C0 2C E3 C0 2C E6 C0 2C E4 C0 4A 90
A00:03 2C E5 C0 4A 48 AD 31 C0 29 7F 90 02 09 80 8D
A10:31 C0 68 4A 90 03 2C E1 C0 4A B0 03 2C E2 C0 60
A20:2C ED C0 2C EE C0 60 2C E7 C0 2C E6 C0 60 A8 2C
A30:ED C0 4C 39 0A 98 8D EF C0 98 4D EE C0 29 1F D0
A40:F4 2C EC C0 60
```

--
Neil Parker
parker@corona.uoregon.edu
parkern@jacobs.cs.orst.edu

No cute ASCII art...no cute quote...no cute
disclaimer...no deposit, no return...
(This space intentionally left blank:)