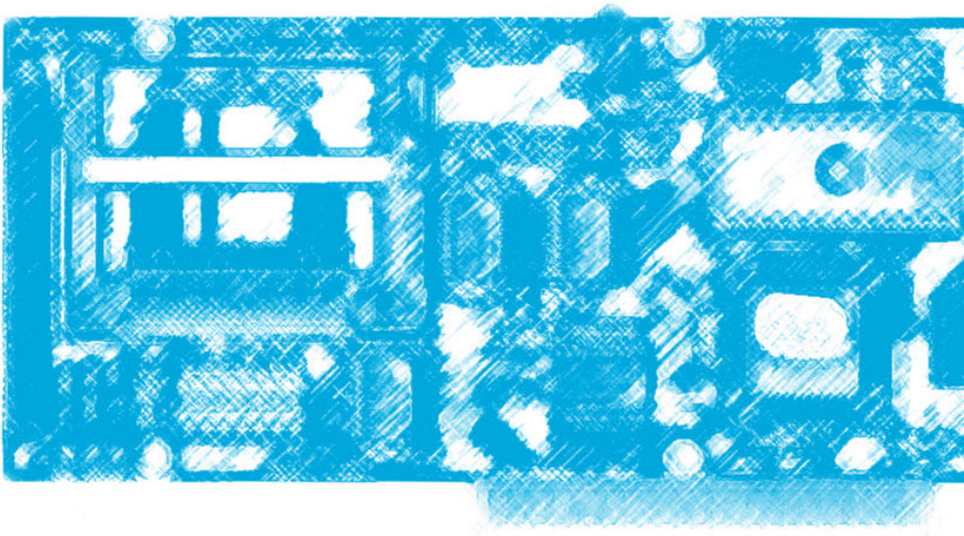
CFFA

Reference Manual



CompactFlash / IDE
Interface for Apple II

CFFA – CompactFlash Interface for Apple II

Manual for CFFA version 1.2 by Richard Dreher

R&D Automation - May 2002

Disclaimer of All Liability

Plain English Version:

Do not use this manual or the CFFA Interface card for any mission-critical applications, or for any purpose, in which a bug or failure could cause you a financial or material loss. This product was designed to enhance your Apple II computing experience, but may contain design flaws that could inhibit its proper operation, or result in a loss of the data recorded on the storage devices attached to it. When using this product you assume all risks associated with operation or data loss. If these terms are not acceptable, you may return the product for a refund.

Legalese Version:

Richard Dreher, doing business as R&D Automation, makes no warranties either express or implied with respect to this manual or with respect to the software or firmware described in this manual, its quality, performance, or fitness for any particular purpose. All software and firmware is sold or licensed “as is”. Any risk of incidental or consequential damages resulting from the use of the information in this manual or the software / firmware / hardware described herein, shall be assumed by the user or buyer or licensee. In no event will R&D Automation be liable for direct, indirect, incidental or consequential damages resulting from any defect in the software / firmware / hardware described in this manual.

R&D Automation reserves the right to make changes and improvements to the product described in this manual at any time and without notice.

Contents

Basic Information 6

Warranty and Return Information 7
Warnings 8
Quick Start Instructions 9
Installation Details 10
CFFA Partition Scheme 10
Apple II Boot Procedure with CFFA Interface Card Installed 11
CompactFlash Memory Cards 13
CompactFlash Socket 13
CF Advantages 13
CF Disadvantages 14
CF Removability 14
IDE Drives 15
IDE Drives Compatible with the CFFA 15
IDE Drive Connector 15
IDE Power Connector 15
Mounting Holes for 2.5” Drives 16
Preparing the Storage Device 17
Partition Scheme for Storage Devices 17
Formatting Storage Devices 17
Devices Compatible with CFFA 17
GS/OS Users 18

Advanced Information 20

Hardware 21
Altera CPLD 21
Altera CPLD Pinout 22
CPLD Logic Files 23
Firmware 25
Why a Static Partition Scheme? 25
Firmware Updates 25
Contributing Firmware to the CFFA Project 25
EPROM Firmware Select Jumpers 26
EPROM Layout 27
CFFA Hardware Memory Map 29
Marketing Megabytes 31
Contact Information 32
CFFA Web Site 32
Internet E-Mail 32

CFFA Message Web Forum 32
Acknowledgements 33
Appendix 1: Firmware Listing 34

Basic Information

Warranty and Return Information

You may return the CFFA Interface card for any reason within 90 days of receiving it. This should allow you enough time to evaluate the compatibility with your system. I guarantee your CFFA Interface card to be free of defects under normal usage for a period of one year from the date you receive the product. This means that if the card fails, and you have treated it properly, I will repair, replace, or refund your money at my discretion, to be determined by me on a case by case basis.

If you want to return the product under warranty, please contact me via E-mail to discuss return arrangements. Include your name and the serial number from the sticker on the back of the card. It is your responsibility to get the product you are returning back to my door. I will not be responsible for lost shipments. Please choose shipping methods and insurance as you deem necessary.

Warnings

You should avoid electrostatic discharge to the CFFA Interface card. Like all electronics devices, static “shock” can destroy or shorten the life span of the CFFA Interface card. Avoid touching the CFFA Interface card after you have walked across the room, especially over carpet, and especially in dry weather.

You should safely discharge yourself before you handle the CFFA Interface card. This can be done by momentarily coming into contact with a grounded piece of metal.

In all cases, please exercise common sense and observe all electrical warnings provided by the manufacturers of the equipment you are using.

Quick Start Instructions

Most Apple II users will probably not need instructions to install the CFFA Interface card, but I have included Quick Start instructions, below, as well as more detailed information in the next section. The information provided will give you insights into the behavior of the card, so you may better know what to expect.

1. Discharge yourself of excess static charge.
2. Open and remove the CFFA card from the anti-static bag.
3. Set the shorting-block jumpers for the desired firmware version. For most people the default jumper settings should work fine. See Table 2: Firmware Select Jumper Settings, page 26 for details.
4. Attach a storage device. Insert a CompactFlash card* or connect an IDE hard drive, but not both.

   ***IMPORTANT:** Many CF cards have a ridge or lip that may catch on the CFFA's top edge during insertion. In this case it may seem to require a large force to completely insert the card. Do NOT force it—simply lift the CF card's ridge over the top edge of the CFFA board and finish inserting the CF card.*
5. Turn off power to your Apple Computer.
6. Insert the CFFA Interface card into any empty Apple slot. Typically, slot 7 is used for mass storage devices.
7. Turn on your Apple computer. Assuming the attached storage device is not formatted yet, your computer will boot off another device.
8. The CFFA partition scheme is fixed so no user partitioning is necessary or possible. You simply need to format any or all drives on the device using any software that can format a ProDOS volume. I recommend Apple's System Utilities version 3.1 or later. Remember that every drive after the second drive will show up in another slot than the one the CFFA is actually using.
9. If you want to be able to boot from the device, after formatting the drive(s), ProDOS users will have to copy ProDOS and a startup program like BASIC.SYSTEM to the first drive. I recommend using ProDOS version 2.0.3. GS/OS will need to install Apple IIgs system software from floppies or another drive.
10. Before using the card for storing information that you would care not to lose, spend some time testing the CFFA card in your particular environment. In any case, always backup important data onto multiple sources.

Installation Details

The CFFA Interface card comes in an anti-static bag. This bag is a good place to store the card at times when the card is not installed in a computer. Before opening the zip-top bag, be sure you do not have a static charge built up in your body.

The CFFA Interface card can be installed in any Apple II slot. Depending on which Apple and what firmware or driver you are using, you may get varying degrees of functionality based on which slot you use. The card was designed to physically fit into any slot when using a CompactFlash device. If you mount a 2.5 inch hard drive onto the card (using standoffs and the drive mounting holes), I recommend using slot 7 because the hard drive will not interfere with the slots next to it.

CFFA Partition Scheme

The CFFA Interface card uses a fixed partition scheme described in Table 1 below. If you are using ProDOS 8 and Firmware #0 you will see up to four 32 MB drives. If you are using GS/OS and the COMPACTFLASH driver will see up to 6 drives: up to four 32 MB drives and up to two 1 GB drives.

Table 1: CFFA partition scheme

Drive # will exist for...	Device of size:
Drive 1 (up to 32 MB)	Any Size
Drive 2 (up to 32 MB)	> 32 MB *
Drive 3 (up to 32 MB) ProDOS 2.x or later	> 64 MB
Drive 4 (up to 32 MB) ProDOS 2.x or later	> 96 MB
Drive 5 (up to 1 GB) GS/OS with COMPACTFLASH driver	> 128 MB
Drive 6 (up to 1 GB) GS/OS with COMPACTFLASH driver	> 1152 MB

** Please note that the marketing departments of most device manufactures now define "MB" to mean 1,000,000 bytes instead of the more proper 1024 x 1024 = 1,048,576 bytes.*

The following are a few example configurations for different device sizes:

- A 16 MB CF card gives you 1 drive: Drive 1 = 16 MB.
- A 48 MB CF card gives you 2 drives: Drive 1 = 32 MB, Drive 2 = 16 MB.
- A 128 MB CF card gives you 4 drives: Drive 1 = 32 MB, Drive 2 = 32 MB, Drive 3 = 32 MB, Drive 4 = 32MB.

If you are using GS/OS, you will get the stated drive size configurations:

- A 1 GB IDE Hard Drive gives you 5 Drives: Drive 1 = 32MB, Drive 2 = 32 MB, Drive 3 = 32 MB, Drive 4 = 32 MB, Drive 5 = 896 MB
- An 1.6 GB IDE Hard Drive gives you 6 drives: Drive 1 = 32 MB, Drive 2 = 32 MB, Drive 3 = 32 MB, Drive 4 = 32 MB, Drive 5 = 1024 MB, Drive 6 = 486 MB

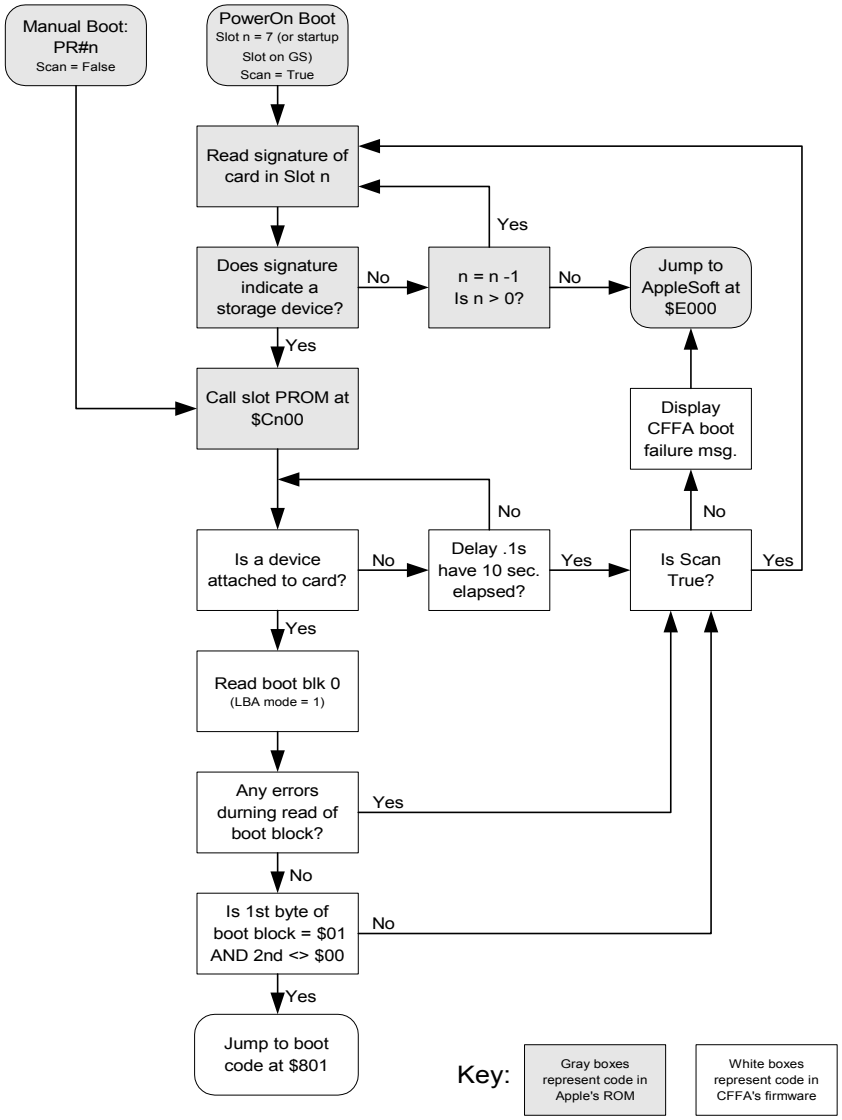
Apple II Boot Procedure with CFFA Interface Card Installed

When the Apple II boots, it begins a slot scan process, starting with slot 7, or Startup Slot on a IIgs, looking for a bootable device. If no bootable device is found in that slot it then proceeds to the next lower slot. For example, if you install your CFFA card into slot 7 with an unformatted device connected to it (as you might after having just received your card), the CFFA firmware will be called by the Apple's ROM because the CFFA Interface card will be recognized as a storage device.

After the card's firmware is called, it will check to see if a device is attached to the card. If a device is found, it will attempt to read the boot block off of the device into memory at location $800. The firmware will wait up to 10 seconds (on a 1MHz Apple) for a device to respond. If no device is found it will return control to the Apple's boot scan ROM routine and the next lower slot will be checked.

If a device was attached and the boot block was read into memory, the firmware will then check to see if the first byte of the boot block code is equal to $01 and the second byte is not equal to $00. If both conditions are met, the firmware will jump to the boot block code and the boot sequence will proceed under the boot code's control. If both conditions are not met, the firmware will return control to the Apple's boot scan ROM routine. See Figure 1 for a flow chart diagram of the boot sequence logic.

Figure 1: Apple II boot sequence logic with CFFA Interface card.



CompactFlash Memory Cards

This section provides detailed information specific to the use of CompactFlash (CF) memory cards with the CFFA Interface card.

CompactFlash Socket

Connector J3 labeled “CompactFlash Socket” is a Type II socket. This allows you to connect either Type I flash cards or Type II devices, such as the IBM Microdrive. The socket is hardwired to use the attached device’s “True IDE” compatibility mode. It is also hardwired to address the device as an IDE master.

When inserting a CF card into the CFFA, you should insert the card label side out. It should not take much effort to insert a card. After the socket pins start to engage the card, a little extra force is needed to fully mate the two.

IMPORTANT: *Many CF cards have a ridge or lip that may catch on the CFFA’s top edge during insertion. In this case it may seem to require a large force to completely insert the card. Do NOT force it—simply lift the CF card’s ridge over the top edge of the CFFA board and finish inserting the CF card.*

TIP: *To make CF card removal much easier you can fashion a handle by folding a piece of cellophane tape over the top of the CF card that extends up about an inch (2.5 cm).*

CF Advantages

CompactFlash cards have several advantages over traditional hard drives.

- CF cards are solid state memory devices which are completely silent and more reliable than IDE hard drives.
- CF cards use less power and generate less heat than IDE hard drives.
- CF cards have no seek delay times related to mechanical head movement. All data in a CF card is accessed at the same speed.

CF Disadvantages

CompactFlash cards do have a few disadvantages when compared to traditional hard drives.

- The cost per megabyte is higher for CF cards.
- Each sector on a CF card can only be written to a limited number of times. This is the write cycle endurance, and is a specification of the CF card itself, and not the CFFA Interface card. You can typically find endurance specifications for CF cards on the manufacturer's web site. For example, SanDisk Corporation specifies that their SDCFB-XX line of CF cards has an endurance of greater than or equal to 300,000 write cycles per block.

Because SanDisk CF cards can dynamically reorganize blocks that are causing errors, this effectively extends the useable life of their product.

If you are using the CFFA interface card in a system that is doing a very large number of write operations to the connected device, you may want use an IDE Hard drive instead of a CF card.

CF Removability

Although most CF cards are used in a "removable" sense, the CFFA interface card does not treat a CF card as a removable device. The card's firmware does not report to either ProDOS or GS/OS that it supports removable devices. You should not treat it like a removable device. In other words, if you want to remove the CF card from the CFFA interface card, shut down your computer first.

Removing the card with the computer's power on will not hurt the CF card, but if you plug the card back in, you will not be able to access the data until you do a complete power cycle of the computer or a reset* of the card. The reason the CF card is not "removable" is that it is being used in the "True IDE" mode and should be thought of as a normal hard drive. For the same reason you don't pull out your hard drive with the power on, you should not pull out a CF card with the power on.

**Reset of the card will occur when the Apple performs a reset, as long as the CFFA's reset enable jumper J2 is installed.*

IDE Drives

This section provides detailed information specific to the use of IDE hard drives with the CFFA Interface card.

IDE Drives Compatible with the CFFA

Most IDE drives should be compatible with the CFFA Interface Card. It is necessary for the IDE drive to support LBA (Logical Block Addressing) mode in order to work with the CFFA card. All IDE drives larger than 528 MB today support this mode. Most old IDE drives smaller than 528 MB did not support LBA and therefore will not work with the CFFA card. If in doubt, try your drive to see if it works.

IDE Drive Connector

Connector J5 labeled “IDE” allows you to connect a single IDE hard drive. Pin 1 of this connector is at the top of the board next to the label “J5”. A drive connected to this connector should be set to a “Master” drive and any CompactFlash card should be removed from the CompactFlash socket J3. The firmware shipped with the CFFA card doesn’t currently support drives set to “Slave”. If you set your drive to “Slave” you may be able to leave it connected while you are using a CF card, but you will not be able to access the drive until you set the drive back to “Master” and remove the CF card.

IDE Power Connector

Connector J1 labeled “IDE POWER” on the CFFA Interface card is provided to supply power to an IDE hard drive that you connect to J5. This connector provides access to the Apple’s power supplies: +5v, +12v, and Ground. It is up to you to ensure that the device you attach to this connector does not consume more power than the Apple’s power supply is capable of delivering. Remember to consider the load of all the other devices in your system.

J1 power connections are labeled as follows:

- +5v DC is labeled “5v-RED”
- +12v DC is labeled “12v-YEL”
- Ground is labeled “GND-BLK”

The labels RED, YEL, and BLK written on the CFFA PCB silkscreen show the standard wire colors used by most IDE drive connectors.

Because J1, the power connector, is a screw terminal type and requires you to connect wires, it is not fool proof. This connector can be miswired. Miswiring could cause the destruction of the IDE hard drive, the CFFA Interface card, and possibly your computer. Use caution, and observe polarities when connecting power to an IDE drive.

***IMPORTANT**: It is your responsibility to be sure that the device you attach to connector J1 is wired correctly, regardless of the wire colors involved.*

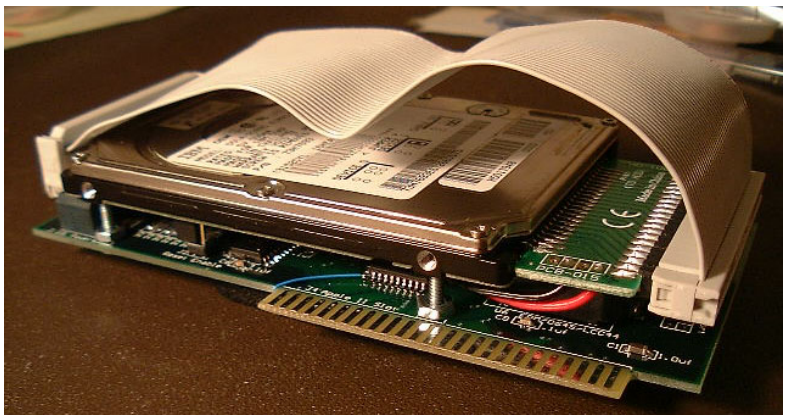
Mounting Holes for 2.5" Drives

The CFFA Interface card has four mounting holes for a standard 2.5 inch hard drive. The card has been designed so that a hard drive could be mounted on the top (component side) of the board using stand-offs or screws. By mounting the drive on the top of the board and placing it in slot 7 you will not have to sacrifice another slot or lay the hard drive on top of the computer's power supply. Figure 2 shows the drive mounting arrangement described above. The little circuit board with the "CE" label on it is a connector converter board that converts from the fine pitch pin spacing .05" of 2.5" hard drives to the standard .1" spacing used by IDE cables. This adapter is not provided with the CFFA Interface card, but can ordered from several places on the Internet.

When stand-offs are not available, common 4-40x1/2" machine screws will work. The threads of these screws do not match the thread pitch of the drive holes, but they are close enough to catch and hold the drive securely in place. Do not over tighten, just catch the threads. Use nuts to hold the screws in place, as shown in Figure 2.

***IMPORTANT:** When mounting a drive on the CFFA Interface card, be sure that no metal contacts, especially the "Reset Enable" header and the "Firmware Select" header, touch or short on the bottom of the drive. This could destroy the card and the drive.*

Figure 2



Preparing the Storage Device

Whether you use a CF card or an IDE drive you will need to prepare the storage device before you can use it to store your Apple II data.

Partition Scheme for Storage Devices

A “Partition Scheme” refers to the number and size of drives you will get on a specific storage device. Many users may be accustomed to deciding how the drive partitions are laid out. However, this is not possible with the CFFA card, which uses a static partition arrangement. Both the on-card firmware (assuming your are using Firmware #0) and the Dave Lyons’ GS/OS driver use the same static partition scheme. Therefore device partitioning is not possible or necessary with the CFFA Interface card.

Formatting Storage Devices

After you attach a storage device to the CFFA Interface card, if it is not already formatted, it must be formatted with the file system for the operating system you will be using. This would typically be ProDOS or HFS. It should be possible to use any software package that can format a ProDOS or HFS volume. I have had good luck using Apple System Utilities version 3.1 to format ProDOS volumes. I have had some problems using Copy II+ version 8.4. Occasionally the volumes formatted with Copy II+ report substantially fewer total blocks than they should.

Formatting is a high level OS format which does not perform any kind of media analysis or bad block checking. You may want to perform a disk verify using Apple System utilities or Copy II+ to see if your computer can read every block on the disk. This can be done before or after you format the volume. Note: A disk verify can take a very long time on a 32 MB drive.

Devices Compatible with CFFA

The CFFA Interface card was developed using SanDisk CompactFlash cards and 2.5” IBM hard drives. Many other devices should work with the card, but I can’t guarantee compatibility with anything else. To help determine which devices work with the CFFA card and which devices do not, I will maintain a compatibility list on my web site, http://dreher.net/CFforAppleII/Compatibility.html.

If you have information about the compatibility of a storage device with the CFFA Interface card, feel free to post a message to the Discussion Forum or E-mail me about it. I will update the Compatibility list as information becomes available.

GS/OS Users

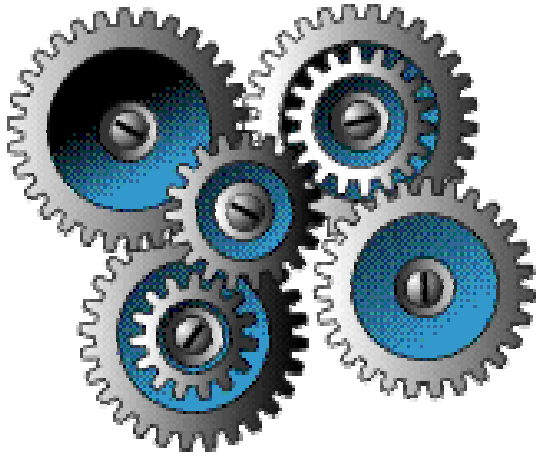
Dave Lyons has developed a GS/OS driver specifically for the CFFA Interface card. The driver is called: COMPACTFLASH. Once this driver is loaded it no longer uses the on-card firmware. It provides support for two additional drives, up to 1 GB each, and faster performance. Note: This driver should also work with most IDE hard drives. The driver can be downloaded via the Internet, from the CFFA web site in the downloads section.

After downloading the driver you should copy it into the DRIVERS folder inside your startup disk's SYSTEM folder. With the driver installed, you will see up to six partitions on your CFFA card, rather than the usual four. Be sure you have selected a firmware version that supports four drives, for example, Firmware #0 - all 3 jumpers installed. Selecting a firmware that supports a different number of drives, for example, Firmware #2 which supports 8 drives, will cause the GS/OS driver to not load.

Once loaded you can tell that the COMPACTFLASH driver is being used successfully from the device name of any partition on the CFFA. In the Finder, select a partition and type Apple-I (Icon Info). Then click the "Where" tab and note the device name. If GS/OS is using a "generated driver" that calls the CFFA's firmware, you will see a name like ".DEV3". If the COMPACTFLASH driver is being used, you will see something like ".CFFA.7A.SANDISK_SDCFB-16", where 7 is the slot number and the following letter indicates the partition (A through F).

The COMPACTFLASH driver controls up to two CFFA cards in your system. Normally, each slot containing a CFFA board will be set to “Your Card” in the Apple IIgs Control Panel. However, the COMPACTFLASH driver will find and use the card even if the slot is NOT set to “Your Card”. In this case, you can use the card while in GS/OS, but it will be invisible to ProDOS 8 making it impossible to boot from the CFFA.

Advanced Information



Hardware

This section gives detailed information about the hardware used on CFFA Interface card.

Altera CPLD

The chip U6 is an Altera EPM7064SLC44-10 CPLD (Complex Programmable Logic Device). It is flash based and can be reprogrammed in two ways. You can remove* the chip from the socket and reprogram it in a device programmer, or you can reprogram the chip without removing it from the socket using connector J7, labeled JTAG Port. For this you will need Altera's programming cable and development software that is compatible with the MAX 7000 series CPLDs. As this is beyond the scope of this manual, you can find more information at Altera's web site: http://www.altera.com/.

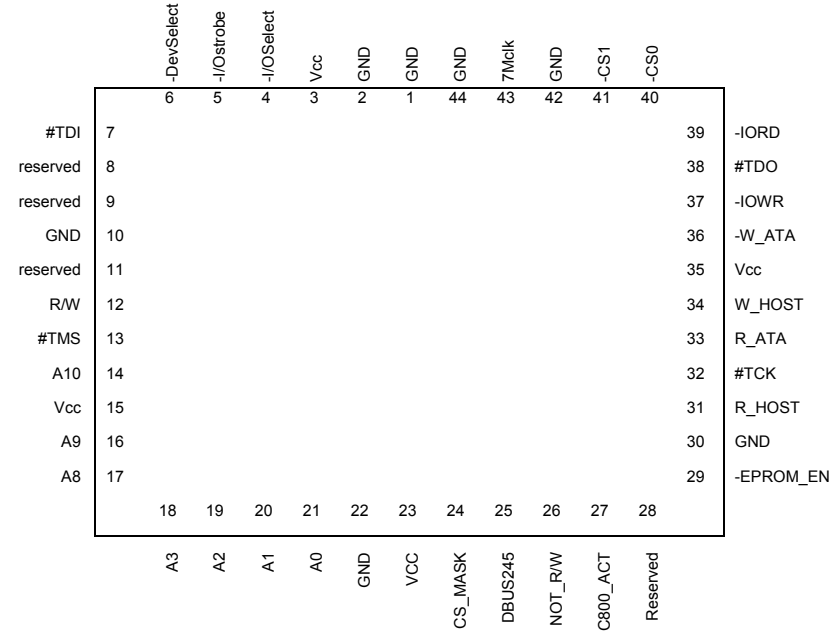
** To facilitate the removal of chip U6 from its socket, there is a hole in the bottom of the CFFA Interface card where you may insert a small tool to push the chip out of its socket. The tool should have a flat end.*

***IMPORTANT:** When inserting a tool, you should be able to insert it just over 7mm when it contacts the bottom of the chip. If your tool stops after inserting it only 3 or 4mm you are feeling the bottom of the socket! DO NOT APPLY PRESSURE TO THE SOCKET! Change your tool's alignment so your tool inserts the entire 7mm. Then apply a firm increasing pressure until the chip pops out. Note that pin 1 is on the side of the socket closest to C11. Pin 1 on the chip is marked by a small circle or dimple.*

Altera CPLD Pinout

Figure 3 provides the signal names for version 1.2 of the CFFA CPLD firmware (here “firmware” refers to the AHDL logic files used to program the CPLD). Different firmware versions could have a slightly different pinout.

Figure 3: Altera’s MAX Series EPM7064SLC44-10F CFFA-specific signal names



CPLD Logic Files

Listing 1 is the ADHL source file used to create the programmer-ready .pof file needed to program U6. Comments in the file start and end with the % character. This file was compiled using Altera's MAX+Plus II Baseline 10.1 development software. This software is free and can be downloaded from Altera's web site.

Listing 1: CPLD Logic - AHDL Source File Version 1.2

```
%------------------------------------------------------------------------
 CompactFlash/IDE Interface for the Apple II computer
 Project Home: http://dreher.net/CFforAppleII/
 Project Version 1.2   Feb 8, 2002

 Version 1.2  -  Two or more cards would not work in system at same time
                 because the DBUS245 select logic did not take into account
                 expansion ROM enable flipflop state.
              -  Moved all logic into appleidelogic.tdf file.
 Version 1.1  -  Add 7M clk & DFF to fix IDE drives /IORD & /IOWR timing
 Version 1.0  -  Initial Release

Note: Remember that the Apple II Bus does not have Phase 2!
------------------------------------------------------------------------%
FUNCTION nandltch (sn, rn) RETURNS (q);
FUNCTION DFF (D, CLK) RETURNS (Q);

SUBDESIGN AppleIDELogic
(
   A0, A1, A2, A3, A8, A9, A10             : INPUT;
   /RW, /DSEL, /IO_STRB, /IO_SEL, 7Mclk    : INPUT;

   /R_HOST, R_ATA, W_HOST, /W_ATA          : OUTPUT;
   /IOWR, /IORD, /CS0, /CS1                : OUTPUT;
   /DBUS245, C800_ACT, /EPROM_EN, NOT_RW   : OUTPUT;
   CS_MASK                                 : OUTPUT;
)
VARIABLE
SET_MASK, RESET_MASK, %CS_MASK,% DelayDSEL : NODE;
/CFXX, /C800_FF         : NODE;

BEGIN
   DEFAULTS
       CS_MASK = GND;
       /C800_FF = VCC;
   END DEFAULTS;


% Expansion Slot ROM enable Flip-flop. Active low signal %
/C800_FF = nandltch(/CFXX, /IO_SEL);
C800_ACT = !/C800_FF;  % For debug only, can be removed %

% Output for debug reworked PCB Version 1.2 Rev A  chip U2 was wired with A and B
busses reversed. %
NOT_RW = !/RW;  % Not needed for production %

% EPROM select. Active low signal %
/EPROM_EN = (/C800_FF # /IO_STRB) !$ /IO_SEL;

%------------------------------------------------------------------------%
% Fix for SanDisk Family of CompactFlash drives. True IDEmode is not quite     %
% True! The idea here is to mask the read cycle the preceeds all write cycles, %
% because the read cycle was confusing the Sandisk                             %

 SET_MASK = /DSEL # (A3 # A2 # A1 # !A0);
 RESET_MASK = /DSEL # (A3 # A2 # !A1 # A0);
 CS_MASK  = nandltch(SET_MASK, RESET_MASK);
%------------------------------------------------------------------------%

% ------ Delay /IORD and /IOWR approx 50ns using a D type FF and the Apple Bus %
% 7Mhz clock the A0-A3 signals are used to keep from generating these signals  %
```

```
% when accessing the latches %

 DelayDSEL = DFF(/DSEL, 7Mclk);
 /IOWR  = /DSEL # DelayDSEL # /RW  # !(A3 # A2 # A1 # A0);
 /IORD  = /DSEL # DelayDSEL # !/RW # !(A3 # A2 # A1 # A0);

% decode address range $CFxx for deselecting the onboard EPROM %
 /CFXX   = !(A8 & A9 & A10 & !/IO_STRB);

% Latch chip select logic %
 /R_HOST  = /DSEL # A3 # A2 # A1 # A0 # !/RW;
  R_ATA   = !/DSEL & (A3 # (A2 & A1)) & /RW;
  W_HOST  = !/DSEL & !(A3 # A2 # A1 # A0) & !/RW;
 /W_ATA   = /DSEL # !(A3 # (A2 & A1)) # /RW;

% device chip select logic %
 /CS0   = /DSEL # !A3 # (CS_MASK & /RW);
 /CS1   = /DSEL # (A3 # !(A1 & A2)) # (CS_MASK & /RW);
 /DBUS245 = /DSEL & /EPROM_EN & /IO_SEL;

END;
```

Firmware

This section gives detailed information about the EPROM based code (firmware) shipped with the CFFA Interface card.

Why a Static Partition Scheme?

The partition scheme is static because the CFFA card does not have onboard RAM, so there is no temporary place to store partition information. In order to implement a user-configurable partitioning scheme, the firmware/driver would have to read the partition table from the device on every read or write access, affecting performance in a negative way.

Firmware Updates

If new versions of firmware become available they may be downloaded from the CFFA Interface web site. If you have access to an EPROM programmer and eraser, you can simply download the new firmware files, and using the programmer-ready binary file, erase and reprogram your EPROM. After you have programmed your EPROM you can insert the EPROM back into the interface card and back into your Apple for testing.

Note: I recommend backing up the contents of the EPROM before you erase it for updating. Most EPROM programmers will let you read and save the data from an EPROM. Also, you can download all of the old versions of firmware from the CFFA web site.

Future versions of firmware may be offered on EPROM for a small fee.

Contributing Firmware to the CFFA Project

There are two ways in which you can contribute to the firmware portion of this project.

1. You can send me your ideas for improvements which I will consider integrating into a future firmware version.
2. Write your own firmware/driver for my hardware and send me the working source code and EPROM-ready binary, and I will post it on my web site under a contributors' section.

EPROM Firmware Select Jumpers

There are three pairs of pins on jumper J6 which can be used to assign a logic 1 or 0 to the top three address lines of the EPROM. These address lines are labeled A14, A13, and A12. When a jumper is in place you are connecting the corresponding address line to a logic 0 level. When the jumper is removed, the corresponding address line is pulled up to a logic 1 level. Since three address lines can represent 8 unique settings, you can select between 8 different 4K regions of EPROM which could each contain a unique firmware version. The CFFA Version 1.2 Rev B card comes with two main versions of firmware and two debug versions for a total of four versions programmed into the EPROM. See Table 2 for a list of those versions.

Table 2: Firmware Select - J6 Jumper Settings

A14	A13	A12	EPROM Offset	Firmware Selected
IN	IN	IN	$0000	#0: SmartPort / ProDOS 8 firmware – Supports four 32MB drives - **Default setting**. Apple IIgs users using GS/OS should use this setting with the COMPACTFLASH driver.
IN	IN	OUT	$1000	#1: Debug version of #0. Requires Apple's Super Serial Card in Slot 2 to use. Using without the SSC in slot 2 may cause unpredictable results.
IN	OUT	IN	$2000	#2: SmartPort / ProDOS 8 firmware – Supports eight 32MB drives. This setting would typically be used by ProDOS 8 only users who want 256MB of storage.
IN	OUT	OUT	$3000	#3: Debug version of #2. Requires Apple Super Serial Card in Slot 2 to use. Using without the SSC in slot 2 may cause unpredictable results.
OUT	IN	IN	$4000	#4: Empty
OUT	IN	OUT	$5000	#5: Empty
OUT	OUT	IN	$6000	#6: Empty
OUT	OUT	OUT	$7000	#7: Empty

OUT = Jumper block is not installed
IN = Jumper block is installed

The debug versions of firmware provide the same functionality as their non-debug counterparts with one addition. The debug version sends textual information out the serial port to another PC or terminal at 19200 baud. This allows the user to get an idea of what is happening as the driver code executes. The penalty for this additional information is a much slower execution speed of the firmware. Therefore it is recommended that you only use the debug versions when actually trying to find a problem with the firmware.

The empty space on the EPROM will allow for additional firmware versions. An example might be a 6502-only code driver for the Apple II+ or IIe.

EPROM Layout

The chip U5 located in the upper right corner of the board is an ST Microelectronics M27C256B-10F1 256Kbit EPROM. One of eight 4K sections of this 32KB EPROM is mapped into the Apple's address space starting at $C000 using jumpers J6 (A14, A13, A12). Therefore, it contains both the Peripheral Card ROM space and the Expansion ROM space. Because the EPROM is mapped over the entire I/O space, the EPROM has a separate space for each slot ROM. The firmware takes advantage of this setup by repeating substantially the same code for each $Cn00 slot space: $C100, $C200, ..., $C700. Note: Even though the base address of the EPROM is $C000, the EPROM is not enabled for addresses in the range of $C000 to $C0FF or $CF00 to $CFFF. Table 3 shows the relationship between an address on the Apple's bus and the EPROM's response.

Table 3: EPROM Offsets listed for when all 3 jumpers at J6 are installed

Address on Apple's Bus	EPROM Offset	EPROM's Response
$C000 to $C0FF	$0 to $FF	Not Used. EPROM never enabled in this range
$C100 to $C1FF	$100 to $1FF	Slot 1 ROM Space. Enabled when card is in slot 1.
$C200 to $C2FF	$200 to $2FF	Slot 2 ROM Space. Enabled when card is in slot 2.
$C300 to $C3FF	$300 to $3FF	Slot 3 ROM Space. Enabled when card is in slot 3.
$C400 to $C4FF	$400 to $4FF	Slot 4 ROM Space. Enabled when card is in slot 4.
$C500 to $C5FF	$500 to $5FF	Slot 5 ROM Space. Enabled when card is in slot 5.

Address on Apple's Bus	EPROM Offset	EPROM's Response
$C600 to $C6FF	$600 to $6FF	Slot 6 ROM Space. Enabled when card is in slot 6.
$C700 to $C7FF	$700 to $7FF	Slot 7 ROM Space. Enabled when card is in slot 7.
$C800 to $CEFF	$800 to $EFF	Expansion ROM space. Must be previously enabled by access to $CnXX. (n = slot)
$CF00 to $CFFF	$F00 to $FFF	EPROM never enabled in this range. Any access disables Expansion ROM space. But always use $CFFF to disable.

The layout shown in Table 3 is repeated 8 times, based on the setting of jumpers J6 labeled A12, A13, A14. See Table 2 for the EPROM offsets that would be accessed for each of the eight Firmware Select settings, set by J6.

CFFA Hardware Memory Map

Table 4 shows all of the slot-specific I/O addresses decoded by the CFFA Interface card. These addresses are used to interface a storage device's task register file to the Apple's bus. There is an extra register to allow the translation from the 16 bit ATA device to the Apple's 8 bit bus. Also, due to a bug in some CF card implementation of "TrueIDE" mode, there are two special soft switches used to inhibit CPU read cycles from confusing CF cards during block write routines.

Table 4: Slot specific I/O used by the CFFA Interface card

Apple Address (for Slot 7)	**Name Used in Source Code**	**Read/ Write**	**Description**
$C080+$n0 ($C0F0)	ATADataHigh	R/W	This register is used in combination with the ATADataLow register $C080+$n8. See $C080+$n8 description.
$C080+$n1 ($C0F1)	SetCSMask	R/W	Special soft switch to **disable** CS0 & CS1 signaling to attached device during 65C02 read cycles that always precede write cycles
$C080+$n2 ($C0F2)	ClearCSMask	R/W	Special soft switch to **enable** CS0 & CS1 signaling to attached device during 65C02 read cycles that always precede write cycles
$C080+$n3 ($C0F3)			Unused
$C080+$n4 ($C0F4)			Unused
$C080+$n5 ($C0F5)			Unused
$C080+$n6 ($C0F6)	ATADevCtrl	W	This register is used to control the device's interrupt request line and to issue an ATA soft reset to the device.
$C080+$n6 ($C0F6)	ATAAltStatus	R	This register returns the device status when read by the host. Reading the ATAAltStatus register does NOT clear a pending interrupt. (NOTE: CFFA does not use interrupts)
$C080+$n7 ($C0F7)			Unused

Apple Address (for Slot 7)	**Name Used in Source Code**	**Read/ Write**	**Description**
$C080+$n8 ($C0F8)	ATADataLow	R/W	This register is used to transfer data between the host and the attached device. It is used in combination with the ATADataHigh register to form a 16 bit data word. When reading words from the attached device, this register **must** be read first, and then the High byte can be read from the ATADataHigh register. When writing to the attached device, the ATADataHigh register **must** be written first, and then the ATADataLow register can be written.
$C080+$n9 ($C0F9)	ATAError	R	This register contains additional information about the source of an error when an error is indicated in bit 0 of the ATAStatus register.
$C080+$nA ($C0FA)	ATASectorCnt	R/W	This register contains the number of blocks of data requested to be transferred on a read or write operation between the host and the device. If the value in this register is zero, a count of 256 sectors is specified.
$C080+$nB ($C0FB)	ATASector	R/W	This register contains the starting sector number or bits 7-0 of the Logical Block Address (LBA) for any device access for the subsequent command.
$C080+$nC ($C0FC)	ATACylinder	R/W	This register contains the low order 8 bits of the starting cylinder address or bits 15-8 of the Logical Block Address.
$C080+$nD ($C0FD)	ATACylinderH	R/W	This register contains the high order bits of the starting cylinder address or bits 23-16 of the Logical Block Address.
$C080+$nE ($C0FE)	ATAHead	R/W	This register is used to select LBA addressing mode or cylinder/head/sector addressing mode. Also bits 27-24 of the Logical Block Address.
$C080+$nF ($C0FF)	ATACommand	W	A write to this register will issue an ATA command to the device.
$C080+$nF ($C0FF)	ATAStatus	R	This register returns the device status when read by the host. Reading the Status register does clear a pending interrupt. (NOTE: CFFA does not use interrupts)

n = the slot number in which the CFFA Interface card is installed.

Marketing Megabytes

At some point when storage device sizes became sufficiently large, some marketing genius decided that their products would appear more impressive if they used the standard SI definition of the prefix Mega, where 1 MB = 1,000,000 Bytes instead of the widely used computer-centric definition where 1 MB = 1024 x 1024 = 1,048,576 Bytes. It appears that after one company did this the rest were forced to follow suit.

Therefore, you may notice that the last drive on your storage device has a few less blocks than you may have expected. For example, one might expect that a SanDisk 64 MB CF card would provide space for two full 32 MB drives. However, this is not the case. It provides only 29.25 MB or 30.67 MB (marketing) for the second drive. Given that the SanDisk 64 MB card that has 125440 ($1EA00) blocks instead of the expected 131072 ($20000) blocks, and that the first drive consumes the first 65536 ($10000) blocks (ProDOS wastes 1 block), that leaves the second drive with only 59904 ($EA00) blocks. Which is a drive size of 29.25 MB or 30.67 MB (Marketing), but not 32 MB.

Note: *Block size is always 512 ($200) bytes.*

Contact Information

The following is a list of information resources for the CFFA Interface Card project. If you have questions, comments, or problems to report, please contact me using one of the methods listed below.

CFFA Web Site

The CFFA web site is located at: http://dreher.net/CFforAppleII/. There you will find any new firmware revisions, project revisions, and general project status information.

Internet E-Mail

I can be reached via E-mail at rich@dreher.net.

If you are reporting a problem, please use “CFFA problem“ or similar as your subject line. In your E-mail you should include the firmware version number, which can be determined by removing any CF cards or IDE drives from the CFFA Interface card and then booting your computer with the CFFA card installed. An error message appears and following the string “Ver:“ is the firmware version number. Also, if possible, describe the conditions necessary to cause the problem.

CFFA Message Web Forum

To post a message in the forum, simply point your web browser to: http://dreher.net/phpBB/. The CFFA message forum is a good place to post technical problems and solutions. Other users may have had similar problems and know of a possible solution. You will have to register before you can post to the forum the first time. This only takes a moment, and is free.

Acknowledgements

I would like to thank the following people for providing help on this project:

Sherry Dreher

Josh King

Dave Lyons

Jeff Pagel

Chris Schumann

Appendix 1: Firmware Listing

The following is a listing of the Firmware #0 located on the EPROM at offset $100, accessed when all three shorting blocks of jumper J6 are in place. The rest of the code on the EPROM is not shown in this listing. Please check for future firmware listings on the CFFA web site. A .lst list file is included in each firmware archive. Special thanks to Dave Lyons for doing the vast majority of the SmartPort firmware.

Listing 2: SmartPort Firmware Version 1.2 located at Firmware #0

```
ca65 V2.6.9 - (C) Copyright 1998-2000 Ullrich von Bassewitz
Main file   : SPDRV.S
Current file: SPDRV.S

000000r           ;--------------------------------------------------------------------------
000000r           ; ProDOS/SmartPort driver for CompactFlash/IDE Interface for Apple II computers
000000r           ; SPDRV.S  Version 1.2 - 04/12/2002
000000r           ;
000000r           ; Firmware Contributors:        Email:
000000r           ;  Chris Schumann                cschumann@twp-llc.com
000000r           ;  Rich Dreher                   rich@dreher.net
000000r           ;  Dave Lyons                    dlyons@lyons42.com
000000r           ;
000000r           ;
000000r           ; This code requires a 65C02 or 65C816 equipped machine.
000000r           ;
000000r           ; Tools used to build this driver: CA65: 6502 Cross Assembler
000000r           ; http://www.cc65.org/
000000r           ;
000000r           ; Here is the copyright from that tool using --version option
000000r           ; ca65 V2.6.9 - (C) Copyright 1998-2000 Ullrich von Bassewitz
000000r           ;
000000r           ; Example build instructions on an M$DOS based machine:
000000r           ;-------------------------------------------------------
000000r           ; Assumes you have installed the CC65 package and set your path, etc.
000000r           ;
000000r           ; 1) c:\firmware> ca65 -t apple2 --cpu 65C02 -l spdrv.s
000000r           ; 2) c:\firmware> ld65 -t apple2 spdrv.o -o spdrv.bin
000000r           ; 3) Because the EPROM can hold up to 8 user selectable version of firmware
000000r           ;    you must load spdrv.bin into your EPROM programmer with one of the
000000r           ;    following offsets:
000000r           ;    (Note: this offset has nothing to do with the card slot offsets)
000000r           ;
000000r           ;  for driver #0: use offset $0100.  Selected with: A14= IN, A13= IN, A12= IN
000000r           ;  for driver #1: use offset $1100.  Selected with: A14= IN, A13= IN, A12=OUT
000000r           ;  for driver #2: use offset $2100.  Selected with: A14= IN, A13=OUT, A12= IN
000000r           ;  for driver #3: use offset $3100.  Selected with: A14= IN, A13=OUT, A12=OUT
000000r           ;  for driver #4: use offset $4100.  Selected with: A14=OUT, A13= IN, A12= IN
000000r           ;  for driver #5: use offset $5100.  Selected with: A14=OUT, A13= IN, A12=OUT
000000r           ;  for driver #6: use offset $6100.  Selected with: A14=OUT, A13=OUT, A12= IN
000000r           ;  for driver #7: use offset $7100.  Selected with: A14=OUT, A13=OUT, A12=OUT
000000r           ;
000000r           ;  where IN = jumper shorted, and OUT = jumper open
000000r           ;  Driver #0 through #7 correspond to the user selectable
000000r           ;  jumpers: J6 (A14,A13,A12) on the interface card.
000000r           ;
000000r           ; 4) Load as many firmware versions, up to 8, into the EPROM programmer as you
000000r           ;    want. Remember that most programmers will, by default, clear all unused
000000r           ;    memory to $FF when you load a binary file. You will need to disable that
000000r           ;    feature after loading the first file.
000000r           ;
000000r           ; 5) Now you have an EPROM ready image to be programmed.
000000r           ;    Using a standard 27C256 EPROM or similar, program your EPROM.
000000r           ;
000000r           ; Firmware Version History
000000r           ;-------------------------
000000r           ; Version 1.2
000000r           ;   - Start of SmartPort driver. Based on Version 1.1 ProDOS driver
000000r           ;
000000r           ; Version 1.1
000000r           ;   - dynamically calculate drive sizes in GetStatus function
000000r           ;   - turn off interrupts in case boot code is called manually
000000r           ;   - add cardID/firmware revision bytes: CFFA$xx
000000r           ;   - added continuation of boot scan if device not present
000000r           ;   - added continuation of boot scan if boot block code looks invalid
000000r           ;   - reformatted this source file, removed tabs and added function headers
000000r           ;
000000r           ; Version 1.0
000000r           ;   - initial version for Prototype #2 with descrete latches
000000r           ;
```

```
000000r           ; PLD Logic Firmware Information
000000r           ;------------------------------
000000r           ; This version of firmware assumes you are using PLD logic of at
000000r           ; least version 1.2.  The source files for U6, the Altera PLD are:
000000r           ;       Appleideinterface.gdf
000000r           ;       Appleidelogic.tdf
000000r           ;
000000r           ; The programmer ready output file for the PLD logic is:
000000r           ;       Appleideinterface.pof
000000r           ;
000000r           ; These files are not included with this code.
000000r           ;
000000r           ; Acknowledgements
000000r           ;-----------------
000000r           ;Thanks to:
000000r           ; Chris Schuman - for his extensive initial development work
000000r           ; David Lyons   - for technical information, many improvement ideas and
000000r           ;                 SmartPort code development
000000r           ;
000000r
000000r           .define         EQU             =
000000r           .define         TRUE            1
000000r           .define         FALSE           0
000000r
000000r
000000r           ;
000000r           ; Firmware Version Information
000000r           ;
000000r           FIRMWARE_VER    EQU $12         ;Version 1.2 (Version of this code)
000000r           SPDRIVERVERSION EQU $1200       ;SmartPort version 1.2
000000r           GSOS_DRIVER     EQU $02         ;GS/OS driver will check this byte to see if it
000000r                                            ;is still compatible with this firmware.
000000r                                            ;Increment by one, when something changes that
000000r                                            ;would require a change in the GS/OS driver.
000000r                                            ;Otherwise only change the FIRMWARE_VER for all
000000r                                            ;other changes.
000000r                                            ;01 = ProDOS Driver supporting 2 drives
000000r                                            ;02 = SmartPort Driver supporting 4 drives
000000r                                            ;03 = SmartPort Driver supporting 8 drives
000000r           ;
000000r           ; Firmware Configuration Settings:
000000r           ;
000000r           SMARTPORT       EQU TRUE
000000r           BLOCKOFFSET     EQU 0           ;0..255: LBA of first block of first partition
000000r           PARTITIONS32MB  EQU 4           ;Number of 32MB Partitions supported
000000r                                            ;Remember, ProDOS only supports 13 total
000000r                                            ;volumes for all devices, floppies, SCSI drives,
000000r                                            ;RAM drives, etc.
000000r
000000r
000000r           ;-------------------------------------------------------------------------------
000000r           ; To enable debug output, set DEBUG = TRUE, only if you have a Apple Super
000000r           ; Serial card in slot 2. This will output one line of text for each request
000000r           ; made to the firmware, which can be seen on a computer or terminal attached to
000000r           ; the SSC.
000000r           ;
000000r           ; NOTE: If you use DEBUG=TRUE and you don't have an Apple Super Serial card in
000000r           ; slot 2, your computer might hang in the routine DSChar, waiting for
000000r           ; the SSC status bit to say it is okay to write to the 6551 UART.
000000r           ;
000000r           ; Set your terminal (software) at the remote end as follows:
000000r           ; BaudRate:      19200
000000r           ; Data Bits:     8
000000r           ; Parity:        None
000000r           ; Stop Bits:     1
000000r           ;
000000r           ; Example debug output at terminal from CAT command in ProDOS 8. Card is in
000000r           ; slot 6. ProDOS makes a ProDOS 8 call to the firmware to read block 2 from
000000r           ; unit: 60 into buffer memory at $DC00.
000000r           ;
000000r           ; P8: Rd B:0002 U:60 A$DC00 Chk$6711
000000r           ;
000000r           ; Rd = ProDOS Read ($01). Also could be Wr = write ($02), St = Status ($00)
000000r           ; U:60 = ProDOS 8 unit number $60  Slot 6, drive 1
000000r           ; A$DC00 = ProDOS buffer address
000000r           ; Chk$6711 = Simple block checksum used to visually check data integrity
000000r           ;
000000r           ; NOTE: When DEBUG is true, some zero-page locations are used. The data at
000000r           ; these locations are saved and restored and should not impact other programs.
000000r
000000r           DEBUG  = FALSE
000000r           ;DEBUG  = TRUE
000000r
000000r           ;-------------------------------------------------------------------------------
000000r           ; Driver constant definitions
000000r           ;
000000r           INITDONESIG     EQU $A5           ;Device init is done signature value
000000r           CR              EQU $0D
000000r           BELL            EQU $07
000000r
000000r           ; ProDOS request Constants
000000r           PRODOS_STATUS   EQU $00
000000r           PRODOS_READ     EQU $01
000000r           PRODOS_WRITE    EQU $02
000000r           PRODOS_FORMAT   EQU $03
```

```
000000r
000000r           ;ProDOS Return Codes
000000r           PRODOS_NO_ERROR      EQU $00    ;No error
000000r           PRODOS_BADCMD        EQU $01    ;Bad Command (not implemented)
000000r           PRODOS_IO_ERROR      EQU $27    ;I/O error
000000r           PRODOS_NO_DEVICE     EQU $28    ;No Device Connected
000000r           PRODOS_WRITE_PROTECT EQU $2B    ;Write Protected
000000r           PRODOS_BADBLOCK      EQU $2D    ;Invalid block number requested
000000r           PRODOS_OFFLINE       EQU $2F    ;Device off-line
000000r
000000r           ;SmartPort return codes
000000r           BAD_UNIT_NUMBER EQU $11
000000r
000000r           ; ATA Commands Codes
000000r           ATACRead        EQU $20
000000r           ATACWrite       EQU $30
000000r           ATAIdentify     EQU $EC
000000r
000000r           ;Constants for Wait
000000r           ; Constant = (Delay[in uS]/2.5 + 2.09)^.5 - 2.7
000000r
000000r           WAIT_100ms      EQU 197
000000r           WAIT_40ms       EQU 124
000000r           WAIT_100us      EQU 4
000000r           ;-------------------------------------------------------------------------------
000000r           ; Slot I/O definitions
000000r           ;
000000r           mslot           = $7F8          ;Apple defined location for the last active slot
000000r
000000r           IOBase          = $C080
000000r           ATADataHigh     = IOBase+0
000000r           SetCSMask       = IOBase+1      ;Two special strobe locations to set and clear
000000r                                           ; MASK bit that is used to disable CS0 line to
000000r                                           ; the CompactFlash during the CPU read cycles
000000r           ClearCSMask     = IOBase+2      ; that occur before every CPU write cycle.
000000r                                           ; The normally inoccuous read cycles were
000000r                                           ; causing the SanDisk CF to double increment
000000r                                           ; during sector writes commands.
000000r
000000r           ATADevCtrl      = IOBase+6      ;when writing
000000r           ATAAltStatus    = IOBase+6      ;when reading
000000r           ATADataLow      = IOBase+8
000000r           ATAError        = IOBase+9
000000r           ATASectorCnt    = IOBase+10
000000r           ATASector       = IOBase+11
000000r           ATACylinder     = IOBase+12
000000r           ATACylinderH    = IOBase+13
000000r           ATAHead         = IOBase+14
000000r           ATACommand      = IOBase+15     ; when writing
000000r           ATAStatus       = IOBase+15     ; when reading
000000r
000000r           ; Scratchpad RAM base addresses. Access using the Y register containg the slot #
000000r           ;
000000r           DriveResetDone  = $478          ;remember the device has been software reset
000000r           DriveNumber     = $4f8          ;normally 0 to 3 for four 32MB partitions
000000r           SerialInitDone  = $578          ;For debug: if $A5 then serial init is complete
000000r           DrvBlkCount0    = $5f8          ;low byte of usable block count
000000r                                           ; (excluding first BLOCKOFFSET blocks)
000000r           DrvBlkCount1    = $678          ;bits 8..15 of usable block count
000000r           DrvBlkCount2    = $6f8          ;bits 16..23 of usable block count
000000r           DrvMiscFlags    = $778          ;bit 7 = raw LBA block access
000000r           Available2      = $7f8          ;not currently used
000000r
000000r           ;-------------------------------------------------------------------------------
000000r           ; Zero-page RAM memory usage
000000r
000000r           .IF DEBUG                       ;data at these locations saved and restored
000000r           MsgPointerLow   = $EB
000000r           MsgPointerHi    = $EC
000000r           CheckSumLow     = $ED
000000r           CheckSumHigh    = $EE
000000r           .ENDIF
000000r
000000r           zpt1            = $EF           ;data at this location is saved/restored
000000r
000000r           StackBase       = $100
000000r
000000r           ; ProDOS block interface locations
000000r           pdCommandCode   = $42
000000r           pdUnitNumber    = $43
000000r           pdIOBuffer      = $44
000000r           pdIOBufferH     = $45
000000r           pdBlockNumber   = $46
000000r           pdBlockNumberH  = $47
000000r
000000r           ; Arbitrary locations for Smartport data,
000000r           ;  these locations are saved/restored before exit.
000000r           spCommandCode   = pdCommandCode
000000r
000000r           spParamList     = $48           ;2 bytes
000000r           spCmdList       = $4A           ;2 bytes
000000r           spCSCode        = $4C
000000r           spSlot          = $4D
000000r           spSlotX16       = $4E
000000r           spLastZP        = spSlotX16
```

```
000000r
000000r           spZeroPgArea    = pdCommandCode ; $42
000000r           spZeroPgSize    = spLastZP-spZeroPgArea+1
000000r
000000r
000000r           ;-------------------------------------------------------------------------------
000000r           ; Apple II ROM entry points
000000r           ;
000000r           ; We can use these at boot time, but not while handling a call through
000000r           ; our ProDOS or SmartPort entry points, as the ROM may not be available.
000000r           ;
000000r           SetVID          = $FE89
000000r           SetKBD          = $FE93
000000r           COUT            = $FDED
000000r           INIT            = $FB2F
000000r           HOME            = $FC58
000000r           ROMWAIT         = $FCA8
000000r           AppleSoft       = $E000
000000r
000000r           ;-------------------------------------------------------------------------------
000000r           ; Start of Peripheral Card ROM Space $Cn00 to $CnFF
000000r           ; A macro is used here so that this code can be easily duplicated for each slot
000000r           ; instead of by hand using the EPROM programmer. This is possible done because
000000r           ; the hardware does not overlay the C1xx address space at C2xx, C3xx, etc.
000000r           ; automatically. Instead the base address for the EPROM is $C000 but is enabled
000000r           ; only when a valid address in the range of $C100 to $CEFF is on the bus. This
000000r           ; allows for the development of slot specific behaviors in firmware, if desired.
000000r           ; Currently that is not being done, instead the same slot code is repeated for
000000r           ; every slot ROM space. Addresses $C000 to $C0FF are not decoded by the
000000r           ; hardware as it is Apple's internal I/O. Any access of $CF00 to $CFFF is
000000r           ; decoded by the card to reset the Expansion ROM flip-flop, but remember to
000000r           ; use always address $CFFF for that task.
000000r
000000r           .macro  CnXX    SLOTADDR, SLOTx16, SLOT
000000r              .local P8DriverEntry
000000r              .local P8Driver
000000r              .local SmartPortEntry
000000r              .local SPDriver
000000r              .local Boot
000000r              .local Error
000000r              .local NotScanning
000000r              .local wasteTime
000000r              .local ErrorMsgDisplay
000000r              .local msgLoop
000000r              .local msgDone
000000r              .local ErrorMessage
000000r
000000r              lda  #$20                  ;$20 is a signature for a drive to ProDOS
000000r              ldx  #$00                  ;$00 "
000000r              lda  #$03                  ;$03 "
000000r
000000r           .IF SMARTPORT
000000r              lda  #$00
000000r           .ELSE
000000r              lda  #$3c                  ;$3c "
000000r           .ENDIF
000000r              bra  Boot
000000r
000000r           ;------------------- Non-boot P8 driver entry point ---------------------
000000r           ;  The EPROM holding this code is decoded and mapped into $C100 to $CEFF,
000000r           ;  it is not nessecary to dynamically determine which slot we are in, as is
000000r           ;  common in card firmware. This code is in a MACRO and is located absolutely
000000r           ;  per slot. Any code in this MACRO that is not relocatable will have to be
000000r           ;  based on the MACROs parameters SLOTADDR, SLOTx16, SLOT.
000000r
000000r           P8DriverEntry:
000000r              jmp  P8Driver              ;located at $Cn0A for best compatibility
000000r           SmartPortEntry:               ;By definition, SmartPort entry is 3 bytes
000000r                                        ; after ProDOS entry
000000r              jmp  SPDriver
000000r
000000r           Boot:
000000r              ldy  #SLOT                 ;Y reg now has $0n for accessing scratchpad RAM
000000r              ldx  #SLOTx16              ;X reg now has $n0 for indexing I/O
000000r              lda  #>SLOTADDR            ;loads A with slot# we are in:$Cn
000000r              sta  mslot                 ;Apple defined location reserved for last slot
000000r                                        ; active. MSLOT needs the form of $Cn
000000r              bit  $cfff                 ;turn off expansion ROMs
000000r
000000r           ;
000000r           ; Need to wait here (before CheckDevice) in case the CFFA RESET jumper
000000r           ; is enabled, or a Delkin Devices CF card never becomes ready.
000000r           ;
000000r              ldy  #5
000000r           wasteTime:
000000r              lda  #WAIT_100ms
000000r              jsr  ROMWAIT
000000r              dey
000000r              bne  wasteTime
000000r              ldy  #SLOT
000000r
000000r              jsr  CheckDevice
000000r              bcs  Error
000000r
000000r              lda  #PRODOS_READ          ;Request: READ block
```

```
000000r               sta  pdCommandCode
000000r               stz  pdIOBuffer              ;Into Location $800
000000r               stz  pdBlockNumber           ;ProDOS block $0000 (the bootloader block)
000000r               stz  pdBlockNumberH
000000r               lda  #$08
000000r               sta  pdIOBufferH
000000r               stx  pdUnitNumber            ;From unit number: $n0 (where n=slot#),
000000r                                            ; so drive bit is always 0
000000r
000000r               jsr  P8Driver                ;Read bootloader from device's block 0 into
000000r                                            ; location $800
000000r               bcs  Error                   ;Check for error during bootblock read
000000r
000000r               lda  $800                    ;Check the first byte of boot loader code.
000000r               cmp  #$01                    ;If bootload code is there, this byte = $01
000000r               bne  Error
000000r               lda  $801                    ;If second byte is a 0, it's invalid
000000r                                            ; (we'd JMP to a BRK)
000000r               beq  Error
000000r
000000r               ldx  pdUnitNumber            ;X should contain the unit number when jumping
000000r                                            ; to the bootloader
000000r               jmp  $801                    ;No errors, jump to bootloader just read.
000000r
000000r
000000r            ; If any error occured, like drive not present, check to see if we are in a
000000r            ; boot scan, if so re-enter scan routine, else drop to Applesoft, aborting boot.
000000r            Error:
000000r               lda  $00
000000r               bne  NotScanning
000000r               lda  $01
000000r               cmp  mslot
000000r               bne  NotScanning
000000r               jmp  $FABA                   ;Re-enter Monitor's Autoscan Routine
000000r
000000r            ;The boot code must have been called manually because we are not in a slot scan.
000000r            NotScanning:
000000r               jsr  SetVID
000000r               jsr  SetKBD
000000r            ;
000000r            ;Display error message
000000r            ;
000000r               jsr  INIT             ;text mode, full screen, page 1
000000r               jsr  HOME
000000r               ldy  #0
000000r            msgLoop:
000000r               lda  ErrorMessage,y
000000r               beq  msgDone
000000r               ora  #$80
000000r               jsr  COUT
000000r               iny
000000r               bra  msgLoop
000000r            msgDone:
000000r               jmp  AppleSoft
000000r
000000r            ErrorMessage:
000000r               .byte CR,CR,CR,CR,CR
000000r               .byte "CFFA: Device missing, not formatted,",CR
000000r               .byte "or incompatible.",CR
000000r            ; "vX.Y" built automatically:
000000r               .byte "Ver:",$30+(FIRMWARE_VER/16),".",$30+(FIRMWARE_VER & $F)
000000r            ; Beep, then end-of-message:
000000r               .byte BELL,$00
000000r
000000r
000000r            ;------------------- Non-boot entry point for driver code -----------------
000000r            ;
000000r            ; Handle a ProDOS call
000000r            ;
000000r            ; Setup MSLOT, X and Y registers.
000000r            ; This must be done every time we enter this driver.
000000r            ;
000000r            P8Driver:
000000r               ldy  #SLOT                   ;Y reg now has $0n for accessing scratchpad RAM
000000r               ldx  #SLOTx16                ;X reg now has $n0 for indexing I/O
000000r               lda  #>SLOTADDR              ;loads A with slot# we are in:$Cn(where n=slot#)
000000r               sta  mslot                   ;Apple defined location reserved for last slot
000000r                                            ; active. MSLOT needs the form of $Cn
000000r               bit  $cfff                   ;turn off other ROM that might be on
000000r               bit  ClearCSMask,x           ;reset MASK bit in PLD for normal CS0 signaling
000000r               jmp  P8AuxROM
000000r
000000r            ;---------------------- SmartPort call handler code ----------------------
000000r            ;
000000r            ; Called from jmp at $Cn0D.
000000r            ; 1) Push a block of zero-page locations onto the stack, creating a work space.
000000r            ; 2) Get the request parameters that follow the call to this driver
000000r            ; 3) Using request parameters as pointers get request specific information
000000r            ;    and set up the P8 driver request registers $42-$47.
000000r            ; 4) Call P8Driver code
000000r            ; 5) On return from P8Driver code, restore zero page work space, and return to
000000r            ;    caller.
000000r            ;
000000r            SPDriver:
000000r               lda  #>SLOTADDR
```

```
000000r               sta  mslot
000000r               bit  $cfff
000000r
000000r               ldx  #SLOTx16               ;X reg now has $n0 for indexing I/O
000000r               bit  ClearCSMask,x          ;reset MASK bit in PLD for normal CS0 signaling
000000r               jmp  SPAuxROM
000000r
000000r
000000r               .RES   SLOTADDR+$F5-*       ;skip to $CnF5, where n is the slot#
000000r
000000r               .byte  GSOS_DRIVER          ;GS/OS driver compatibility byte. GS/OS driver
000000r                                           ; checks this byte to see if it is compatible
000000r                                           ; with this version of firmware. This way,
000000r                                           ; changes to firware versions, that do not
000000r                                           ; affect the GS/OS driver will not prevent the
000000r                                           ; GS/OS driver from loading and running. This
000000r                                           ; byte should be incremented each time a change
000000r                                           ; is made that would prevent the GS/OS driver
000000r                                           ; from working correctly. I.e. Partition layout
000000r                                           ; or something similar.
000000r
000000r               .byte "CFFA", FIRMWARE_VER  ;$CnF6..CnFA: Card Hardware ID,
000000r                                           ; non-standard scheme
000000r
000000r               .byte $0                    ;$CnFB: SmartPort status byte
000000r                                           ; Not Extended; not SCSI; not RAM card
000000r                                           ; Even if not supporting SmartPort, we need a
000000r                                           ; zero at $CnFB so Apple's RAMCard driver
000000r                                           ; doesn't mistake us for a "Slinky" memory
000000r                                           ; card.
000000r
000000r            ; Data table for ProDOS drive scan
000000r            ; $CnFC/FD = disk capacity, if zero use status command to determine
000000r            ; $CnFE = status bits (BAP p7-14)
000000r            ;  7 = medium is removable
000000r            ;  6 = device is interruptable
000000r            ;  5-4 = number of volumes (0..3 means 1..4)
000000r            ;  3 = device supports Format call
000000r            ;  2 = device can be written to
000000r            ;  1 = device can be read from (must be 1)
000000r            ;  0 = device status can be read (must be 1)
000000r            ;
000000r            ; $CnFF = LSB of block driver
000000r               .word $0000                 ;$CnFC-D: A zero here will cause prodos to
000000r                                           ; rely on the status command to determine
000000r                                           ; volume size
000000r
000000r               .byte $17                   ; $CnFE: support 2 ProDOS drives
000000r               .byte <P8DriverEntry        ; $CnFF: low-order offset to ProDOS entry point
000000r            .endmacro
000000r
000000r
000000r            .ORG $C100
00C100:A9 20 A2 00    CnXX $C100, $10, $01        ;Slot PROM code for slot 1
00C104:A9 03 A9 00
00C108:80 06 4C C3
00C200:
00C200:            .ORG $C200
00C200:A9 20 A2 00    CnXX $C200, $20, $02        ;Slot PROM code for slot 2
00C204:A9 03 A9 00
00C208:80 06 4C C3
00C300:
00C300:            .ORG $C300
00C300:A9 20 A2 00    CnXX $C300, $30, $03        ;Slot PROM code for slot 3
00C304:A9 03 A9 00
00C308:80 06 4C C3
00C400:
00C400:            .ORG $C400
00C400:A9 20 A2 00    CnXX $C400, $40, $04        ;Slot PROM code for slot 4
00C404:A9 03 A9 00
00C408:80 06 4C C3
00C500:
00C500:            .ORG $C500
00C500:A9 20 A2 00    CnXX $C500, $50, $05        ;Slot PROM code for slot 5
00C504:A9 03 A9 00
00C508:80 06 4C C3
00C600:
00C600:            .ORG $C600
00C600:A9 20 A2 00    CnXX $C600, $60, $06        ;Slot PROM code for slot 6
00C604:A9 03 A9 00
00C608:80 06 4C C3
00C700:
00C700:            .listbytes      unlimited      ;Show all of the code bytes for the 7th slot
00C700:            .ORG $C700
00C700:A9 20 A2 00    CnXX $C700, $70, $07        ;Slot PROM code for slot 7
00C704:A9 03 A9 00
00C708:80 06 4C C3
00C70C:C7 4C D5 C7
00C710:A0 07 A2 70
00C714:A9 C7 8D F8
00C718:07 2C FF CF
00C71C:A0 05 A9 C5
00C720:20 A8 FC 88
00C724:D0 F8 A0 07
00C728:20 82 CC B0
```

```
00C72C:26 A9 01 85
00C730:42 64 44 64
00C734:46 64 47 A9
00C738:08 85 45 86
00C73C:43 20 C3 C7
00C740:B0 11 AD 00
00C744:08 C9 01 D0
00C748:0A AD 01 08
00C74C:F0 05 A6 43
00C750:4C 01 08 A5
00C754:00 D0 0A A5
00C758:01 CD F8 07
00C75C:D0 03 4C BA
00C760:FA 20 89 FE
00C764:20 93 FE 20
00C768:2F FB 20 58
00C76C:FC A0 00 B9
00C770:7F C7 F0 08
00C774:09 80 20 ED
00C778:FD C8 80 F3
00C77C:4C 00 E0 0D
00C780:0D 0D 0D 0D
00C784:43 46 46 41
00C788:3A 20 44 65
00C78C:76 69 63 65
00C790:20 6D 69 73
00C794:73 69 6E 67
00C798:2C 20 6E 6F
00C79C:74 20 66 6F
00C7A0:72 6D 61 74
00C7A4:74 65 64 2C
00C7A8:0D 6F 72 20
00C7AC:69 6E 63 6F
00C7B0:6D 70 61 74
00C7B4:69 62 6C 65
00C7B8:2E 0D 56 65
00C7BC:72 3A 31 2E
00C7C0:32 07 00 A0
00C7C4:07 A2 70 A9
00C7C8:C7 8D F8 07
00C7CC:2C FF CF 3C
00C7D0:82 C0 4C 7D
00C7D4:CA A9 C7 8D
00C7D8:F8 07 2C FF
00C7DC:CF A2 70 3C
00C7E0:82 C0 4C 00
00C7E4:C8 xx xx xx
00C7E8:xx xx xx xx
00C7EC:xx xx xx xx
00C7F0:xx xx xx xx
00C7F4:xx 02 43 46
00C7F8:46 41 12 00
00C7FC:00 00 17 0A
00C800:            .listbytes      12              ;revert back to normal listing mode for the rest
00C800:
00C800:            ;-------------------- End of Peripheral Card ROM Space ----------------------
00C800:
00C800:
00C800:            ;--------------------- Start of I/O expansion ROM Space --------------------
00C800:            ; Code here starts at $c800
00C800:            ; This code area is absolute and can use absolute addressing
00C800:            .ORG   $C800                    ;note: .ORG does not cause this code to be
00C800:                                            ; placed at C800. The ".RES" statement above
00C800:                                            ; offsets this code
00C800:
00C800:
00C800:            ;-------------------------------------------------------------------------------
00C800:            ; SmartPort
00C800:            ;
00C800:            ;-------------------------------------------------------------------------------
00C800:            MagicRawBlocksUnit EQU 127      ;use unit 127 in ReadBlock/WriteBlock calls for
00C800:                                            ; raw LBA blocks.
00C800:
00C800:            SPAuxROM:
00C800:            ; save  ZP, fetch cmd + pointer from (PC)
00C800:A0 0C          ldy  #spZeroPgSize-1
00C802:            save:
00C802:B9 42 00       lda  spZeroPgArea,y
00C805:48             pha
00C806:88             dey
00C807:10 F9          bpl  save
00C809:BA             tsx
00C80A:BD 0E 01       lda  $101+spZeroPgSize,x
00C80D:85 48          sta  spParamList
00C80F:18             clc
00C810:69 03          adc  #3
00C812:9D 0E 01       sta  $101+spZeroPgSize,x
00C815:BD 0F 01       lda  $102+spZeroPgSize,x
00C818:85 49          sta  spParamList+1
00C81A:69 00          adc  #0
00C81C:9D 0F 01       sta  $102+spZeroPgSize,x
00C81F:
00C81F:            ; set up spSlot and spSlotX16
00C81F:AD F8 07       lda  mslot
00C822:29 0F          and  #$0f
```

```
00C824:A8            tay
00C825:0A            asl  a
00C826:0A            asl  a
00C827:0A            asl  a
00C828:0A            asl  a
00C829:AA            tax
00C82A:84 4D         sty  spSlot
00C82C:86 4E         stx  spSlotX16
00C82E:
00C82E:           ; clear DrvMiscFlags
00C82E:A9 00         lda  #0
00C830:99 78 07      sta  DrvMiscFlags,y          ;no special flags (such as raw block access)
00C833:
00C833:           ; reset the device if this is our first call
00C833:20 B1 CA      jsr  ResetDriveIfFirstTime   ;needs Y
00C836:
00C836:           .IF DEBUG
00C836:           ;warning this debug code trashes the Acc register
00C836:              jsr  DSString
00C836:              .byte "SP:",0
00C836:           .ENDIF
00C836:
00C836:           ; get command code from parameter list
00C836:A0 01         ldy  #1
00C838:B1 48         lda  (spParamList),y
00C83A:85 42         sta  spCommandCode
00C83C:C8            iny
00C83D:B1 48         lda  (spParamList),y
00C83F:AA            tax
00C840:C8            iny
00C841:B1 48         lda  (spParamList),y
00C843:85 49         sta  spParamList+1
00C845:86 48         stx  spParamList
00C847:
00C847:A9 01         lda  #PRODOS_BADCMD          ;anticipate bad command error
00C849:A6 42         ldx  spCommandCode
00C84B:E0 0A         cpx  #$09+1                  ;command too large
00C84D:B0 1E         bcs  out
00C84F:
00C84F:B2 48         lda  (spParamList)           ;parameter count
00C851:DD 8A C8      cmp  RequiredParamCounts,x   ;command number still in X
00C854:F0 04         beq  pCountOK
00C856:
00C856:A9 04         lda  #$04                    ;bad parameter count error
00C858:80 13         bra  out
00C85A:
00C85A:           pCountOK:
00C85A:A0 01         ldy  #1
00C85C:B1 48         lda  (spParamList),y
00C85E:A4 4D         ldy  spSlot
00C860:99 F8 04      sta  DriveNumber,y
00C863:
00C863:8A            txa                           ;X is still the command code
00C864:0A            asl  a
00C865:AA            tax
00C866:20 81 C8      jsr  JumpToSPCommand          ;Y is still spSlot
00C869:
00C869:B0 02         bcs  out
00C86B:A9 00         lda  #0
00C86D:
00C86D:           out:
00C86D:AA            tax                           ;error code in X
00C86E:
00C86E:           ; Restore zero page
00C86E:A0 00         ldy  #0
00C870:           restore:
00C870:68            pla
00C871:99 42 00      sta  spZeroPgArea,y
00C874:C8            iny
00C875:C0 0D         cpy  #spZeroPgSize
00C877:90 F7         bcc  restore
00C879:
00C879:8A            txa
00C87A:A0 02         ldy  #2                      ;high byte of # bytes transferred
00C87C:                                           ; (always (1) undefined, or
00C87C:                                           ; (2) #bytes transferred to host)
00C87C:A2 00         ldx  #0                      ;low byte of # bytes transferred
00C87E:C9 01         cmp  #1                      ;C=1 if error code nonzero
00C880:60            rts
00C881:
00C881:           JumpToSPCommand:
00C881:BD 95 C8      lda  spDispatch+1,x
00C884:48            pha
00C885:BD 94 C8      lda  spDispatch,x
00C888:48            pha
00C889:60            rts
00C88A:
00C88A:           RequiredParamCounts:
00C88A:03            .byte 3   ;0 = status
00C88B:03            .byte 3   ;1 = read
00C88C:03            .byte 3   ;2 = write
00C88D:01            .byte 1   ;3 = format
00C88E:03            .byte 3   ;4 = control
00C88F:01            .byte 1   ;5 = init
00C890:01            .byte 1   ;6 = open
```

```
00C891:01             .byte 1   ;7 = close
00C892:04             .byte 4   ;8 = read
00C893:04             .byte 4   ;9 = write
00C894:
00C894:            spDispatch:
00C894:A7 C8          .word spStatus-1
00C896:B6 C9          .word spReadBlock-1
00C898:C3 C9          .word spWriteBlock-1
00C89A:CF C9          .word spFormat-1
00C89C:D2 C9          .word spControl-1
00C89E:F6 C9          .word spInit-1
00C8A0:01 CA          .word spOpen-1
00C8A2:01 CA          .word spClose-1
00C8A4:05 CA          .word spReadChars-1
00C8A6:05 CA          .word spWriteChars-1
00C8A8:
00C8A8:            ;-------------------------------------------------------------------------------
00C8A8:            ; SmartPort STATUS call
00C8A8:            ;
00C8A8:            ; We support the standard calls, plus an "Identify" command
00C8A8:            ; for unit 0, which fills a buffer with the card's Identify
00C8A8:            ; data.
00C8A8:            ;
00C8A8:            spStatus:
00C8A8:20 0A CA       jsr  SPSetupControlOrStatus
00C8AB:B0 32          bcs  statOut
00C8AD:
00C8AD:A4 4D          ldy  spSlot
00C8AF:B9 F8 04       lda  DriveNumber,y
00C8B2:D0 33          bne  StatusForOneUnit
00C8B4:
00C8B4:            ; StatusCode = 0 && unit == 0: status for this entire SmartPort interface
00C8B4:A5 4C          lda  spCSCode
00C8B6:F0 0B          beq  Status00
00C8B8:
00C8B8:            ; Status for unit 0, subcode $49 = Identify (device-specific subcode)
00C8B8:C9 49          cmp  #$49
00C8BA:D0 03          bne  BadStatusCode
00C8BC:4C 48 C9       jmp  spStatusIdentify
00C8BF:
00C8BF:            ; Any other status code for unit 0 is an error.
00C8BF:            BadStatusCode:
00C8BF:A9 21          lda  #$21
00C8C1:38             sec
00C8C2:60             rts
00C8C3:
00C8C3:            Status00:
00C8C3:A6 4E          ldx  spSlotX16
00C8C5:A4 4D          ldy  spSlot
00C8C7:20 DA CA       jsr  GetStatus
00C8CA:B0 13          bcs  statOut
00C8CC:
00C8CC:A4 4D          ldy  spSlot
00C8CE:B9 F8 06       lda  DrvBlkCount2,y
00C8D1:1A             inc  a
00C8D2:92 4A          sta  (spCmdList)        ;byte +0 = number of drives
00C8D4:
00C8D4:A0 07          ldy  #7
00C8D6:            Stat00Loop:
00C8D6:B9 DF C8       lda  Stat00Data-1,y
00C8D9:91 4A          sta  (spCmdList),y
00C8DB:88             dey
00C8DC:D0 F8          bne  Stat00Loop
00C8DE:18             clc
00C8DF:            statOut:
00C8DF:60             rts
00C8E0:
00C8E0:            Stat00Data:
00C8E0:40             .byte $40                   ;Interrupt flag = no interrupts caused by this
00C8E1:                                           ; interface
00C8E1:
00C8E1:00 CC          .word $CC00                 ;Vendor ID assigned to Rich Dreher by
00C8E3:                                           ; www.syndicomm.com 3/16/2002
00C8E3:
00C8E3:00 12          .word SPDRIVERVERSION       ;Our version number
00C8E5:00             .byte $00                   ;Reserved byte
00C8E6:00             .byte $00                   ;Reserved byte
00C8E7:
00C8E7:            StatusForOneUnit:
00C8E7:3A             dec  a
00C8E8:99 F8 04       sta  DriveNumber,y
00C8EB:
00C8EB:A6 4C          ldx  spCSCode
00C8ED:F0 16          beq  Status0or3
00C8EF:CA             dex
00C8F0:F0 08          beq  StatusGetDCB
00C8F2:CA             dex
00C8F3:CA             dex
00C8F4:F0 0F          beq  Status0or3
00C8F6:A9 21          lda  #$21                   ;Bad status code
00C8F8:38             sec
00C8F9:60             rts
00C8FA:            ;
00C8FA:            ; We have no interesting data to return for the device control
00C8FA:            ; block, so return the shortest one allowed: a length byte of
```

```
00C8FA:            ; 1 followed by a data byte of 0.
00C8FA:            ;
00C8FA:            StatusGetDCB:
00C8FA:A9 01          lda  #1
00C8FC:92 4A          sta  (spCmdList)             ;Returned length = 1
00C8FE:A8             tay
00C8FF:A9 00          lda  #0
00C901:91 4A          sta  (spCmdList),y           ;Returned data = $00
00C903:18             clc
00C904:60             rts
00C905:
00C905:            ;
00C905:            ; Status code 0 and 3 start off the same; 3 returns extra data.
00C905:            ;
00C905:            ; 0: return device status (1 byte device status + 3-byte block count)
00C905:            ; 3: return Device Information Block
00C905:            ;
00C905:            Status0or3:
00C905:A9 F8          lda  #$F8                    ;Block device, write, read, format, online,
00C907:                                            ; not write-prot
00C907:92 4A          sta  (spCmdList)
00C909:A6 4E          ldx  spSlotX16
00C90B:A4 4D          ldy  spSlot
00C90D:20 DA CA       jsr  GetStatus               ;Returns block count in YX
00C910:B0 CD          bcs  statOut
00C912:
00C912:98             tya
00C913:A0 02          ldy  #2
00C915:91 4A          sta  (spCmdList),y           ;CmdList +2 = bits 8..15 of block count
00C917:88             dey
00C918:8A             txa                          ;CmdList +1 = bits 0..7 of block count
00C919:91 4A          sta  (spCmdList),y
00C91B:A0 03          ldy  #3
00C91D:A9 00          lda  #0
00C91F:91 4A          sta  (spCmdList),y           ;CmdList +3 = bits 16..23 of block count
00C921:
00C921:A5 4C          lda  spCSCode
00C923:F0 0C          beq  statDone
00C925:
00C925:            ; status code 3: return 21 more bytes of data
00C925:A0 04          ldy #4
00C927:            stat3Loop:
00C927:B9 2F C9       lda  stat3Data-4,y
00C92A:91 4A          sta  (spCmdList),y
00C92C:C8             iny
00C92D:C0 19          cpy  #21+4
00C92F:90 F6          bcc  stat3Loop
00C931:
00C931:            statDone:
00C931:18             clc
00C932:60             rts
00C933:
00C933:            stat3Data:
00C933:0D 43 4F 4D    .byte 13,"COMPACT FLASH   "  ;length byte + 16-byte ASCII, padded
00C937:50 41 43 54
00C93B:20 46 4C 41
00C944:02             .byte $02                    ;device type = hard disk
00C945:20             .byte $20                    ;subtype (no removable media, no extended,
00C946:                                            ; no disk switched)
00C946:00 12          .word SPDRIVERVERSION
00C948:
00C948:            ;--------------------------------------------------------------------------------
00C948:            ; Identify (Status subcode)
00C948:            ;
00C948:            ; The status list is a 512-byte buffer that we will fill with
00C948:            ; the drive's IDE "Identify" data.  We post-process the data
00C948:            ; by byte-swapping the model name and serial number strings
00C948:            ; so they make sense.
00C948:            ;
00C948:            spStatusIdentify:
00C948:A4 4D          ldy  spSlot
00C94A:A6 4E          ldx  spSlotX16
00C94C:20 82 CC       jsr  CheckDevice
00C94F:90 03          bcc  ident1
00C951:A9 28          lda  #PRODOS_NO_DEVICE
00C953:60             rts                          ;return error code in Acc with Carry set
00C954:
00C954:            ident1:
00C954:A9 00          lda  #0
00C956:9D 80 C0       sta  ATADataHigh,x
00C959:
00C959:A9 EC          lda  #ATAIdentify
00C95B:9D 8F C0       sta  ATACommand,x            ;Issue the read command to the drive
00C95E:20 78 CC       jsr  IDEWaitReady            ;Wait for BUSY flag to go away
00C961:
00C961:BD 8F C0       lda  ATAStatus,x             ;Check for error response
00C964:29 09          and  #$09
00C966:C9 01          cmp  #$01                    ;if DRQ=0 and ERR=1 an error occured
00C968:D0 04          bne  iCommandOK
00C96A:
00C96A:            iError:
00C96A:            .IF DEBUG
00C96A:            ;warning this debug code trashes the Acc register
00C96A:               jsr DSString
00C96A:               .byte "spIdfy Err:",0
```

```
00C96A:               lda  ATAError,x
00C96A:               jsr  DSByteCRLF
00C96A:            .ENDIF
00C96A:
00C96A:A9 27          lda  #PRODOS_IO_ERROR
00C96C:38             sec
00C96D:60             rts                           ;return error code in Acc with Carry set
00C96E:            ;
00C96E:            ; The "Identify" data is ready to read
00C96E:            ;
00C96E:            pageCount = spCommandCode        ;re-use a zero-page location
00C96E:
00C96E:            iCommandOK:
00C96E:A0 02          ldy  #2
00C970:84 42          sty  pageCount
00C972:A0 00          ldy  #0
00C974:            iLoop:
00C974:BD 8F C0       lda  ATAStatus,x              ;Note: not using IDEWaitReady, inline code
00C977:                                             ; instead
00C977:30 FB          bmi  iLoop                    ;Wait for BUSY (bit 7) to be zero
00C979:29 08          and  #$08                     ;get DRQ status bit
00C97B:F0 ED          beq  iError                   ;if off, didn't get enough data
00C97D:
00C97D:BD 88 C0       lda  ATADataLow,x
00C980:91 4A          sta  (spCmdList),y
00C982:C8             iny
00C983:BD 80 C0       lda  ATADataHigh,x
00C986:91 4A          sta  (spCmdList),y
00C988:C8             iny
00C989:D0 E9          bne  iLoop
00C98B:E6 4B          inc  spCmdList+1
00C98D:C6 42          dec  pageCount
00C98F:D0 E3          bne  iLoop
00C991:
00C991:            ; Swap ASCII text data of the Identify data to be more readable.
00C991:            ; These are both on the first page of the data.
00C991:
00C991:C6 4B          dec  spCmdList+1
00C993:C6 4B          dec  spCmdList+1              ;Point to beginning of buffer
00C995:
00C995:A0 2E          ldy  #23*2                    ;Start at word 23 (firmware rev, model number)
00C997:A2 18          ldx  #24                      ;24 words
00C999:20 A5 C9       jsr  SwapBytes
00C99C:
00C99C:A0 14          ldy  #10*2                    ;Start at word 10 (serial number)
00C99E:A2 0A          ldx  #10                      ;10 words
00C9A0:20 A5 C9       jsr  SwapBytes
00C9A3:
00C9A3:18             clc
00C9A4:60             rts
00C9A5:
00C9A5:            ;-------------------------------------------------------------------------------
00C9A5:            SwapBytes:
00C9A5:B1 4A          lda  (spCmdList),y
00C9A7:48             pha                           ;Save the 1st byte
00C9A8:C8             iny
00C9A9:B1 4A          lda  (spCmdList),y            ;Get the 2nd byte
00C9AB:88             dey
00C9AC:91 4A          sta  (spCmdList),y            ;Store it in position 1
00C9AE:C8             iny
00C9AF:68             pla                           ;Finally, retrieve the 1st byte
00C9B0:91 4A          sta  (spCmdList),y            ;Put it in position 2
00C9B2:C8             iny
00C9B3:CA             dex
00C9B4:D0 EF          bne  SwapBytes
00C9B6:60             rts
00C9B7:
00C9B7:            ;-------------------------------------------------------------------------------
00C9B7:            ; SmartPort READ BLOCK command
00C9B7:            ;
00C9B7:            spReadBlock:
00C9B7:20 1C CA       jsr  SPSetupReadWrite
00C9BA:B0 07          bcs  readDone
00C9BC:
00C9BC:A4 4D          ldy  spSlot
00C9BE:A6 4E          ldx  spSlotX16
00C9C0:4C 8F CB       jmp  ReadBlock
00C9C3:
00C9C3:            readDone:
00C9C3:            writeDone:
00C9C3:60             rts
00C9C4:
00C9C4:            ;-------------------------------------------------------------------------------
00C9C4:            ; SmartPort WRITE BLOCK command
00C9C4:            ;
00C9C4:            spWriteBlock:
00C9C4:20 1C CA       jsr SPSetupReadWrite
00C9C7:B0 FA          bcs writeDone
00C9C9:
00C9C9:A4 4D          ldy spSlot
00C9CB:A6 4E          ldx spSlotX16
00C9CD:4C EA CB       jmp WriteBlock
00C9D0:
00C9D0:            ;-------------------------------------------------------------------------------
00C9D0:            ; SmartPort FORMAT command
```

```
00C9D0:            ;
00C9D0:            ; We don't actually do anything beyond validating the unit number.
00C9D0:            ;
00C9D0:            spFormat:
00C9D0:4C 65 CA       jmp SPValidateUnitNumber
00C9D3:
00C9D3:            ;---------------------------------------------------------------------------
00C9D3:            ; SmartPort CONTROL command
00C9D3:            ;
00C9D3:            spControl:
00C9D3:20 0A CA       jsr SPSetupControlOrStatus
00C9D6:B0 18          bcs ctrlDone
00C9D8:
00C9D8:20 65 CA       jsr SPValidateUnitNumber
00C9DB:B0 13          bcs ctrlDone
00C9DD:
00C9DD:A6 4C          ldx spCSCode
00C9DF:F0 14          beq ctlReset                ;Control code 0 = Reset
00C9E1:CA             dex
00C9E2:F0 11          beq ctlSetDCB               ;Control code 1 = SetDCB
00C9E4:CA             dex
00C9E5:F0 06          beq ctlSetNewline           ;Control code 2 = SetNewline
00C9E7:CA             dex
00C9E8:F0 07          beq ctlServiceInterrupt     ;Control code 3 = ServiceInterrupt
00C9EA:CA             dex
00C9EB:F0 08          beq ctlEject                ;Control code 4 = Eject
00C9ED:
00C9ED:            ctlSetNewline:
00C9ED:A9 21          lda #$21                    ;Bad control code
00C9EF:38             sec
00C9F0:            ctrlDone:
00C9F0:60             rts
00C9F1:
00C9F1:            ctlServiceInterrupt:
00C9F1:A9 1F          lda #$1f                    ;Interrupt devices not supported
00C9F3:38             sec
00C9F4:60             rts
00C9F5:
00C9F5:            ctlReset:
00C9F5:            ctlSetDCB:
00C9F5:            ctlEject:
00C9F5:18             clc
00C9F6:60             rts
00C9F7:
00C9F7:            ;---------------------------------------------------------------------------
00C9F7:            ; SmartPort INIT command
00C9F7:            ;
00C9F7:            ; unit 0 = entire chain
00C9F7:            ;
00C9F7:            ;  SmartPort Technote #2 says you can't init an individual unit;
00C9F7:            ;  so return error if DriveNumber is nonzero.
00C9F7:            ;
00C9F7:            spInit:
00C9F7:B9 F8 04       lda DriveNumber,y
00C9FA:F0 04          beq initChain
00C9FC:A9 11          lda #BAD_UNIT_NUMBER
00C9FE:38             sec
00C9FF:60             rts
00CA00:
00CA00:            initChain:
00CA00:18             clc
00CA01:60             rts
00CA02:
00CA02:
00CA02:            ;---------------------------------------------------------------------------
00CA02:            ; SmartPort: Open and Close are for character devices only
00CA02:            ;
00CA02:            spOpen:
00CA02:            spClose:
00CA02:A9 01          lda #PRODOS_BADCMD
00CA04:38             sec
00CA05:60             rts
00CA06:
00CA06:            ;---------------------------------------------------------------------------
00CA06:            ; SmartPort: Read, Write
00CA06:            ;
00CA06:            ; We don't bother implementing Read and Write, although they are allowed
00CA06:            ; for block devices.  We would only support 512-byte transfers anyway.
00CA06:            ;
00CA06:            spReadChars:
00CA06:            spWriteChars:
00CA06:A9 27          lda #PRODOS_IO_ERROR
00CA08:38             sec
00CA09:60             rts
00CA0A:
00CA0A:
00CA0A:            ;---------------------------------------------------------------------------
00CA0A:            ; SPSetupControlOrStatus
00CA0A:            ;
00CA0A:            ; fetch from parameter block:
00CA0A:            ;    status/control list pointer (word)
00CA0A:            ;    status/control code (byte)
00CA0A:            ;
00CA0A:            SPSetupControlOrStatus:
00CA0A:A0 02          ldy #2
```

```
00CA0C:B1 48          lda (spParamList),y
00CA0E:85 4A          sta spCmdList
00CA10:C8             iny
00CA11:B1 48          lda (spParamList),y
00CA13:85 4B          sta spCmdList+1
00CA15:C8             iny
00CA16:B1 48          lda (spParamList),y
00CA18:85 4C          sta spCSCode
00CA1A:18             clc
00CA1B:60             rts
00CA1C:
00CA1C:            ;--------------------------------------------------------------------------------
00CA1C:            ; SPSetupReadWrite
00CA1C:            ;
00CA1C:            ; Input:
00CA1C:            ;    DriveNumber,y is already a copy of SmartPort unit number
00CA1C:            ;
00CA1C:            ; fetch from SP parameter list:
00CA1C:            ;   buffer pointer
00CA1C:            ;   block number
00CA1C:            ;
00CA1C:            ; Validate unit number:
00CA1C:            ;   127 = magic, allow any block number
00CA1C:            ;   1..N:  Validate block number: $0..FFFE
00CA1C:            ;
00CA1C:            ; DriveNumber,y: translated unit number in case unit was magic
00CA1C:            ; DrvMiscFlags: bit 7 set if we should not use BLOCKOFFSET
00CA1C:            ;
00CA1C:            SPSetupReadWrite:
00CA1C:            ; copy pdIOBuffer from parameter list
00CA1C:A0 02          ldy #2
00CA1E:B1 48          lda (spParamList),y
00CA20:85 44          sta pdIOBuffer
00CA22:C8             iny
00CA23:B1 48          lda (spParamList),y
00CA25:85 45          sta pdIOBuffer+1
00CA27:
00CA27:            ; copy pdBlockNumber from parameter list
00CA27:C8             iny
00CA28:B1 48          lda (spParamList),y
00CA2A:85 46          sta pdBlockNumber
00CA2C:C8             iny
00CA2D:B1 48          lda (spParamList),y
00CA2F:85 47          sta pdBlockNumber+1
00CA31:
00CA31:            ; Validate the unit number and block number.
00CA31:A4 4D          ldy spSlot
00CA33:B9 F8 04       lda DriveNumber,y
00CA36:C9 7F          cmp #MagicRawBlocksUnit
00CA38:F0 18          beq magicUnit
00CA3A:
00CA3A:20 65 CA       jsr SPValidateUnitNumber
00CA3D:B0 12          bcs srwOut
00CA3F:
00CA3F:A0 06          ldy #6
00CA41:B1 48          lda (spParamList),y          ;Bits 16..23 of block number must be $00
00CA43:D0 09          bne badBlockNum
00CA45:
00CA45:A5 47          lda pdBlockNumber+1          ;Block $FFFF is invalid
00CA47:25 46          and pdBlockNumber
00CA49:1A             inc a
00CA4A:F0 02          beq badBlockNum
00CA4C:18             clc
00CA4D:60             rts
00CA4E:
00CA4E:            badBlockNum:
00CA4E:A9 2D          lda #PRODOS_BADBLOCK         ;Bad block number
00CA50:38             sec
00CA51:            srwOut:
00CA51:60             rts
00CA52:            ;
00CA52:            ; For the "magic raw blocks" unit, allow a full 3-byte block number.
00CA52:            ;
00CA52:            magicUnit:
00CA52:A9 80          lda #$80
00CA54:19 78 07       ora DrvMiscFlags,y
00CA57:99 78 07       sta DrvMiscFlags,y           ;Raw block access
00CA5A:
00CA5A:A0 06          ldy #6
00CA5C:B1 48          lda (spParamList),y          ;Bits 16..23 of block number
00CA5E:A4 4D          ldy spSlot
00CA60:99 F8 04       sta DriveNumber,y
00CA63:18             clc
00CA64:60             rts
00CA65:
00CA65:            ;--------------------------------------------------------------------------------
00CA65:            ; SPValidateUnitNumber
00CA65:            ;
00CA65:            ; Validate that DriveNumber is from 1 to N.
00CA65:            ;
00CA65:            ; Input: DriveNumber
00CA65:            ;
00CA65:            ; Output: DriveNumber in range 0..N-1
00CA65:            ;
00CA65:            SPValidateUnitNumber:
```

```
00CA65:A4 4D          ldy spSlot
00CA67:B9 F8 04       lda DriveNumber,y
00CA6A:F0 0D          beq badUnit
00CA6C:3A             dec a
00CA6D:99 F8 04       sta DriveNumber,y
00CA70:D9 F8 06       cmp DrvBlkCount2,y
00CA73:F0 02          beq unitOK
00CA75:B0 02          bcs badUnit
00CA77:            unitOK:
00CA77:18             clc
00CA78:60             rts
00CA79:
00CA79:            badUnit:
00CA79:A9 11          lda #BAD_UNIT_NUMBER
00CA7B:38             sec
00CA7C:60             rts
00CA7D:
00CA7D:            ;---------------------------------------------------------------------------
00CA7D:            ; P8AuxROM - Handle a ProDOS call
00CA7D:            ;
00CA7D:            P8AuxROM:
00CA7D:
00CA7D:            ; If the ProDOS unit number doesn't match our slot number, add 2 to
00CA7D:            ; the drive number.
00CA7D:            ;
00CA7D:            ; This actually happens: If we're in slot 5, we get calls for slot 2
00CA7D:            ; that want to access our 3rd and 4th partitions.
00CA7D:8A             txa                        ; A = $n0
00CA7E:45 43          eor  pdUnitNumber
00CA80:29 70          and  #$70                  ;EOR the slot number
00CA82:F0 02          beq  OurSlot
00CA84:A9 02          lda  #2                    ;Point to drive 3 or 4
00CA86:
00CA86:            OurSlot:
00CA86:24 43          bit  pdUnitNumber
00CA88:10 01          bpl  DriveOne
00CA8A:1A             inc  a
00CA8B:
00CA8B:            DriveOne:
00CA8B:99 F8 04       sta  DriveNumber,y
00CA8E:A9 00          lda  #0
00CA90:99 78 07       sta  DrvMiscFlags,y        ;no special flags (such as raw block access)
00CA93:
00CA93:20 B1 CA       jsr  ResetDriveIfFirstTime
00CA96:
00CA96:            ; process the command code and jump to appropriate routine.
00CA96:
00CA96:            .IF DEBUG
00CA96:            ;warning this debug code trashes the Acc register
00CA96:               jsr  DSString
00CA96:               .byte "P8:",0
00CA96:            .ENDIF
00CA96:
00CA96:A5 42          lda  pdCommandCode
00CA98:C9 01          cmp  #PRODOS_READ
00CA9A:D0 03          bne  chk1
00CA9C:4C 8F CB       jmp  ReadBlock
00CA9F:
00CA9F:            chk1:
00CA9F:C9 02          cmp  #PRODOS_WRITE
00CAA1:D0 03          bne  chk2
00CAA3:4C EA CB       jmp  WriteBlock
00CAA6:
00CAA6:            chk2:
00CAA6:C9 00          cmp  #PRODOS_STATUS
00CAA8:D0 03          bne  chk3
00CAAA:4C DA CA       jmp  GetStatus
00CAAD:
00CAAD:            chk3:
00CAAD:            ; An invalid request # has been sent to this driver.
00CAAD:            ;
00CAAD:            .IF DEBUG
00CAAD:               pha
00CAAD:               jsr DSString
00CAAD:               .byte "CE",0
00CAAD:               pla
00CAAD:               jsr  DSByteCRLF
00CAAD:            .ENDIF
00CAAD:
00CAAD:A9 27          lda  #PRODOS_IO_ERROR
00CAAF:38             sec
00CAB0:60             rts                           ;return to caller. Should always be ProDOS
00CAB1:
00CAB1:            ;---------------------------------------------------------------------------
00CAB1:            ; ResetDriveIfFirstTime - Reset the drive once, the first time the driver
00CAB1:            ; is called ide_devctrl Bit 2 = Software Reset, Bit 1 = nIEN (enable
00CAB1:            ; assertion of INTRQ)
00CAB1:            ;
00CAB1:            ; Input:
00CAB1:            ;       X = requested slot number in form $n0 where n = slot 1 to 7
00CAB1:            ;       Y = $0n (n = slot#) for accessing scratchpad RAM;
00CAB1:            ;
00CAB1:            ; ZeroPage Usage:
00CAB1:            ;       None
00CAB1:            ;
```

```
00CAB1:            ; CPU Registers changed:  A, P
00CAB1:            ;
00CAB1:            ResetDriveIfFirstTime:
00CAB1:B9 78 04       lda  DriveResetDone,Y
00CAB4:C9 A5          cmp  #INITDONESIG
00CAB6:F0 21          beq  resetOK
00CAB8:            ;
00CAB8:            ; Reset the ATA device
00CAB8:            ;
00CAB8:A9 00          lda  #0
00CABA:9D 80 C0       sta  ATADataHigh,x           ;Clear high byte data latch
00CABD:
00CABD:A9 06          lda  #$06                    ;Reset bit=1, Disable INTRQ=1
00CABF:9D 86 C0       sta  ATADevCtrl,x
00CAC2:            ;
00CAC2:            ; Per ATA-6 spec, need to wait 5us minimum. Use a delay of 100us.
00CAC2:            ; Should cover accelerated Apples up to 20Mhz.
00CAC2:            ;
00CAC2:A9 04          lda  #WAIT_100us
00CAC4:20 A6 CC       jsr  Wait
00CAC7:
00CAC7:A9 02          lda  #$02                    ;Reset bit=0, Disable INTRQ=1
00CAC9:9D 86 C0       sta  ATADevCtrl,x
00CACC:            ;
00CACC:            ; Per ATA-6 spec, need to wait 2ms minimum. Use a delay of 40ms.
00CACC:            ; Should cover accelerated Apples up to 20Mhz.
00CACC:            ;
00CACC:A9 7C          lda  #WAIT_40ms
00CACE:20 A6 CC       jsr  Wait
00CAD1:            ;
00CAD1:            ; Per ATA-6 spec, wait for busy to clear, by calling IDEWaitReady
00CAD1:            ;
00CAD1:20 78 CC       jsr  IDEWaitReady
00CAD4:
00CAD4:A9 A5          lda  #INITDONESIG
00CAD6:99 78 04       sta  DriveResetDone,Y        ;Set the init done flag so init only happens
00CAD9:                                            ; once.
00CAD9:            resetOK:
00CAD9:60             rts
00CADA:
00CADA:            ;-------------------------------------------------------------------------
00CADA:            ; GetStatus - Called by ProDOS and SmartPort to get device status and size
00CADA:            ;
00CADA:            ; Input:
00CADA:            ;       DriveNumber,y (0 to 3)
00CADA:            ;       X = slot number in form $n0 where n = slot 1 to 7
00CADA:            ;       Y = $0n (n = slot#) for accessing scratchpad RAM;
00CADA:            ;
00CADA:            ; Output:
00CADA:            ;       A = ProDOS status return code
00CADA:            ;       X = drive size LSB
00CADA:            ;       Y = drive size MSB
00CADA:            ;       Carry flag: 0 = Okay, 1 = Error
00CADA:            ;       DrvBlkCount0..DrvBlkCount2 = usable blocks on device
00CADA:            ;
00CADA:            GetStatus:
00CADA:
00CADA:            .IF DEBUG
00CADA:            ;warning this debug code trashes the Acc register
00CADA:               jsr  DSString
00CADA:               .byte " St",0
00CADA:               jsr  DisplayParms
00CADA:            .ENDIF
00CADA:
00CADA:            ;
00CADA:            ; Determine if a drive/device is present.
00CADA:            ;
00CADA:20 82 CC       jsr  CheckDevice
00CADD:90 0A          bcc  sDriveOK
00CADF:
00CADF:A2 00          ldx  #$00
00CAE1:A0 00          ldy  #$00
00CAE3:A9 2F          lda  #PRODOS_OFFLINE
00CAE5:38             sec
00CAE6:4C 8E CB       jmp  sExit
00CAE9:
00CAE9:            ; Device is present
00CAE9:            sDriveOK:
00CAE9:A9 00          lda  #0
00CAEB:9D 80 C0       sta  ATADataHigh,x           ;clear high byte transfer latch
00CAEE:
00CAEE:20 78 CC       jsr  IDEWaitReady
00CAF1:A9 EC          lda  #ATAIdentify
00CAF3:9D 8F C0       sta  ATACommand,x            ;Issue the read command to the drive
00CAF6:20 78 CC       jsr  IDEWaitReady            ;Wait for BUSY flag to go away
00CAF9:
00CAF9:BD 8F C0       lda  ATAStatus,x             ;Check for error response
00CAFC:29 09          and  #$09
00CAFE:C9 01          cmp  #$01                     ;if DRQ=0 and ERR=1 an error occured
00CB00:D0 0A          bne  sValidATACommand
00CB02:
00CB02:            .IF DEBUG
00CB02:            ;warning this debug code trashes the Acc register
00CB02:               jsr DSString
00CB02:               .byte " Idfy Err:",0
```

```
00CB02:                  lda  ATAError,x
00CB02:                  jsr  DSByteCRLF
00CB02:               .ENDIF
00CB02:
00CB02:A2 00             ldx  #0
00CB04:A0 00             ldy  #0
00CB06:A9 27             lda  #PRODOS_IO_ERROR
00CB08:38                sec
00CB09:4C 8E CB          jmp  sExit                    ; Command Error occured, return error
00CB0C:
00CB0C:                  sValidATACommand:
00CB0C:5A                phy                           ;save Y, it holds the $0n sratchpad RAM offset
00CB0D:A0 00             ldy  #$00                     ;zero loop counter
00CB0F:
00CB0F:                  sPrefetchloop:
00CB0F:20 78 CC          jsr  IDEWaitReady             ;See if a word is ready
00CB12:B0 06             bcs  sWordRdy
00CB14:7A                ply
00CB15:A9 27             lda  #PRODOS_IO_ERROR
00CB17:4C 8E CB          jmp  sExit
00CB1A:
00CB1A:                  sWordRdy:
00CB1A:BD 88 C0          lda  ATADataLow,x             ;Read words 0 thru 56 but throw them away
00CB1D:C8                iny
00CB1E:C0 39             cpy  #57                      ;Number of the last word you want to throw away
00CB20:D0 ED             bne  sPrefetchloop
00CB22:7A                ply
00CB23:
00CB23:                  sPrefetchDone:
00CB23:BD 88 C0          lda  ATADataLow,x             ;Read the current capacity in sectors (LBA)
00CB26:38                sec
00CB27:E9 00             sbc  #BLOCKOFFSET
00CB29:99 F8 05          sta  DrvBlkCount0,y
00CB2C:BD 80 C0          lda  ATADataHigh,x
00CB2F:E9 00             sbc  #0
00CB31:99 78 06          sta  DrvBlkCount1,y
00CB34:BD 88 C0          lda  ATADataLow,x
00CB37:E9 00             sbc  #0
00CB39:99 F8 06          sta  DrvBlkCount2,y
00CB3C:
00CB3C:
00CB3C:               .IF DEBUG
00CB3C:                  jsr  DSString
00CB3C:                  .byte "Size:",0
00CB3C:
00CB3C:                  lda  ATADataHigh,x            ;get the high byte of high word just for display
00CB3C:                  jsr  DSByte
00CB3C:                  lda  DrvBlkCount2,y
00CB3C:                  jsr  DSByte
00CB3C:                  lda  DrvBlkCount1,y
00CB3C:                  jsr  DSByte
00CB3C:                  lda  DrvBlkCount0,y
00CB3C:                  jsr  DSByte
00CB3C:                  jsr  DSBlank
00CB3C:               .ENDIF
00CB3C:
00CB3C:B9 F8 06          lda  DrvBlkCount2,y
00CB3F:C9 04             cmp  #PARTITIONS32MB          ;max out at (#PARTITIONS32MB * $10000 + 00FFFF)
00CB41:                                                ; blocks
00CB41:B0 05             bcs  maxOutAtN
00CB43:
00CB43:BD 80 C0          lda  ATADataHigh,x
00CB46:F0 0D             beq  lessThan8GB
00CB48:                  maxOutAtN:
00CB48:A9 FF             lda  #$FF                     ;The device is truly huge! Just set our 3-byte
00CB4A:                                                ; block count to $03FFFF
00CB4A:99 F8 05          sta  DrvBlkCount0,y
00CB4D:99 78 06          sta  DrvBlkCount1,y
00CB50:A9 03             lda  #PARTITIONS32MB-1        ;Number of 32MB devices, set by the equate:
00CB52:                                                ; #PARTITIONS32MB
00CB52:99 F8 06          sta  DrvBlkCount2,y
00CB55:                  lessThan8GB:
00CB55:
00CB55:
00CB55:                  PostFetch:
00CB55:20 78 CC          jsr IDEWaitReady              ;read the rest of the words, until command ends
00CB58:90 05             bcc sReadComplete
00CB5A:BD 88 C0          lda ATADataLow,x
00CB5D:80 F6             bra PostFetch
00CB5F:                  sReadComplete:
00CB5F:
00CB5F:                  ; DrvBlkCount2 is the number of 32 MB partitions availiable - 1,
00CB5F:                  ; or the highest drive # supported (zero based).
00CB5F:                  ;
00CB5F:                  ; If DrvBlkCount2 > drive # then StatusSize = $FFFF
00CB5F:                  ; If DrvBlkCount2 = drive # then StatusSize = DrvBlkCount1,DrvBlkCount0
00CB5F:                  ; If DrvBlkCount2 < drive # then StatusSize = 0
00CB5F:                  ;
00CB5F:                  ; This scheme has a special case which must be handled because ProDOS
00CB5F:                  ;   partitions are not quite 32 meg in size but are only FFFF blocks in size.
00CB5F:                  ;   If a device is exactly: 32meg or 10000h blocks in size, it would appear
00CB5F:                  ;   as one drive of size FFFF and another drive of size 0000. To handle this
00CB5F:                  ;   case, we check for an exact size of 0000 and fall into the NoDrive code.
00CB5F:                  ;
00CB5F:B9 F8 04          lda DriveNumber,y
```

```
00CB62:D9 F8 06       cmp DrvBlkCount2,y
00CB65:F0 0B          beq ExactSize
00CB67:90 1E          bcc FullSize
00CB69:
00CB69:            NoDrive:
00CB69:A2 00          ldx  #0
00CB6B:A0 00          ldy  #0
00CB6D:A9 2F          lda  #PRODOS_OFFLINE
00CB6F:38             sec
00CB70:80 1C          bra  sExit
00CB72:
00CB72:            ExactSize:                    ;If equal, the DrvBlkCount1,DrvBlkCount0 is the
00CB72:                                          ; drive's exact size
00CB72:B9 F8 05       lda  DrvBlkCount0,y
00CB75:19 78 06       ora  DrvBlkCount1,y
00CB78:F0 EF          beq  NoDrive              ;can't have a 0-block device
00CB7A:
00CB7A:B9 F8 05       lda  DrvBlkCount0,y
00CB7D:AA             tax
00CB7E:B9 78 06       lda  DrvBlkCount1,y
00CB81:A8             tay
00CB82:A9 00          lda  #0
00CB84:18             clc                         ;no errors
00CB85:80 07          bra  sExit
00CB87:
00CB87:            FullSize:
00CB87:A2 FF          ldx  #$FF                   ;X gets low byte of size
00CB89:A0 FF          ldy  #$FF                   ;Y gets high byte of size
00CB8B:A9 00          lda  #0
00CB8D:18             clc                         ;no errors
00CB8E:
00CB8E:            sExit:
00CB8E:            .IF DEBUG
00CB8E:               php                         ;save the carry's state
00CB8E:               pha
00CB8E:               jsr  DSString
00CB8E:               .byte "Retd:",0
00CB8E:               tya
00CB8E:               jsr  DSByte
00CB8E:               txa
00CB8E:               jsr  DSByteCRLF
00CB8E:               pla
00CB8E:               plp                         ;recover the carry
00CB8E:            .ENDIF
00CB8E:
00CB8E:60             rts
00CB8F:
00CB8F:            ;----------------------------------------------------------------------------
00CB8F:            ; ReadBlock - Read a block from device into memory
00CB8F:            ;
00CB8F:            ; Input:
00CB8F:            ;       pd Command Block Data $42 - $47
00CB8F:            ;       X = requested slot number in form $n0 where n = slot 1 to 7
00CB8F:            ;
00CB8F:            ; Output:
00CB8F:            ;       A = ProDOS read return code
00CB8F:            ;       Carry flag: 0 = Okay, 1 = Error
00CB8F:            ;
00CB8F:            ; ZeroPage Usage:
00CB8F:            ;       $EF
00CB8F:            ;       w/DEBUG enabled: $EB, $EC, $ED, $EE
00CB8F:            ;       Note: location $EF is saved and restored before driver exits
00CB8F:            ;
00CB8F:            ReadBlock:
00CB8F:            .IF DEBUG
00CB8F:               jsr  DSString
00CB8F:               .byte " Rd",0
00CB8F:               jsr  DisplayParms
00CB8F:            .ENDIF
00CB8F:
00CB8F:A5 45          lda  pdIOBufferH
00CB91:48             pha
00CB92:A5 EF          lda  zpt1
00CB94:48             pha
00CB95:
00CB95:            .IF DEBUG
00CB95:               lda  CheckSumHigh
00CB95:               pha
00CB95:               lda  CheckSumLow
00CB95:               pha
00CB95:            .ENDIF
00CB95:
00CB95:20 9F CB       jsr ReadBlockCore
00CB98:
00CB98:            .IF DEBUG
00CB98:               ply
00CB98:               sty  CheckSumLow
00CB98:               ply
00CB98:               sty  CheckSumHigh
00CB98:            .ENDIF
00CB98:
00CB98:7A             ply
00CB99:84 EF          sty  zpt1
00CB9B:7A             ply
00CB9C:84 45          sty  pdIOBufferH
```

```
00CB9E:60            rts
00CB9F:
00CB9F:            ReadBlockCore:
00CB9F:            .IF DEBUG
00CB9F:               stz  CheckSumLow
00CB9F:               stz  CheckSumHigh
00CB9F:            .ENDIF
00CB9F:
00CB9F:20 78 CC      jsr  IDEWaitReady
00CBA2:20 4D CC      jsr  Block2LBA              ;Program the device's task file registers
00CBA5:                                          ; based on ProDOS address
00CBA5:
00CBA5:A9 00         lda  #0
00CBA7:9D 80 C0      sta  ATADataHigh,x
00CBAA:
00CBAA:A9 20         lda  #ATACRead
00CBAC:9D 8F C0      sta  ATACommand,x           ;Issue the read command to the drive
00CBAF:20 78 CC      jsr  IDEWaitReady           ;Wait for BUSY flag to clear
00CBB2:
00CBB2:BD 8F C0      lda  ATAStatus,x            ;Check for error response from device
00CBB5:29 09         and  #$09
00CBB7:C9 01         cmp  #$01                   ;If DRQ=0 and ERR=1 a device error occured
00CBB9:D0 04         bne  rCommandOK
00CBBB:
00CBBB:            .IF DEBUG
00CBBB:            ;warning this debug code trashes the Acc register
00CBBB:               jsr DSString
00CBBB:               .byte " Err!",0
00CBBB:               lda  ATAError,x
00CBBB:               jsr  DSByteCRLF
00CBBB:            .ENDIF
00CBBB:
00CBBB:            ;
00CBBB:            ; The drive has returned an error code. Just return I/O error code to PRODOS
00CBBB:            ;
00CBBB:A9 27         lda  #PRODOS_IO_ERROR
00CBBD:38            sec
00CBBE:60            rts
00CBBF:            ;
00CBBF:            ; Sector is ready to read
00CBBF:            ;
00CBBF:            rCommandOK:
00CBBF:A0 02         ldy  #2
00CBC1:84 EF         sty  zpt1
00CBC3:A0 00         ldy  #0
00CBC5:
00CBC5:            rLoop:
00CBC5:BD 8F C0      lda  ATAStatus,x            ;Note: not using IDEWaitReady, using inline code
00CBC8:30 FB         bmi  rLoop                  ;Wait for BUSY (bit 7) to be zero
00CBCA:29 08         and  #$08                   ;get DRQ status bit
00CBCC:F0 18         beq  rShort                 ;if off, didn't get enough data
00CBCE:
00CBCE:BD 88 C0      lda  ATADataLow,x
00CBD1:91 44         sta  (pdIOBuffer),y
00CBD3:C8            iny
00CBD4:
00CBD4:            .IF DEBUG
00CBD4:               clc
00CBD4:               adc  CheckSumLow
00CBD4:               sta  CheckSumLow
00CBD4:            .ENDIF
00CBD4:
00CBD4:BD 80 C0      lda  ATADataHigh,x
00CBD7:91 44         sta  (pdIOBuffer),y
00CBD9:
00CBD9:            .IF DEBUG
00CBD9:               adc CheckSumHigh
00CBD9:               sta CheckSumHigh
00CBD9:            .ENDIF
00CBD9:
00CBD9:C8            iny
00CBDA:D0 E9         bne  rLoop
00CBDC:E6 45         inc  pdIOBufferH
00CBDE:C6 EF         dec  zpt1
00CBE0:D0 E3         bne  rLoop
00CBE2:
00CBE2:            .IF DEBUG
00CBE2:               jsr  DSString
00CBE2:               .byte " Chk$",0
00CBE2:
00CBE2:               lda  CheckSumHigh
00CBE2:               jsr  DSByte
00CBE2:               lda  CheckSumLow
00CBE2:               jsr  DSByteCRLF
00CBE2:            .ENDIF
00CBE2:
00CBE2:A9 00         lda  #0
00CBE4:18            clc
00CBE5:60            rts
00CBE6:            ;
00CBE6:            ; The Block was short, return I/O error code to PRODOS
00CBE6:            ;
00CBE6:            rShort:
00CBE6:            .IF DEBUG
00CBE6:               jsr  DSString
```

```
00CBE6:               .byte " Short blk", 0
00CBE6:
00CBE6:               lda  zpt1
00CBE6:               jsr  DSByte
00CBE6:               tya
00CBE6:               jsr  DSByteCRLF
00CBE6:            .ENDIF
00CBE6:
00CBE6:A9 27          lda #PRODOS_IO_ERROR
00CBE8:38             sec
00CBE9:60             rts
00CBEA:
00CBEA:            ;--------------------------------------------------------------------------
00CBEA:            ; WriteBlock - Write a block in memory to device
00CBEA:            ;
00CBEA:            ; Input:
00CBEA:            ;       pd Command Block Data $42 - $47
00CBEA:            ;       X = requested slot number in form $n0 where n = slot 1 to 7
00CBEA:            ;
00CBEA:            ; Output:
00CBEA:            ;       A = ProDOS write return code
00CBEA:            ;       Carry flag: 0 = Okay, 1 = Error
00CBEA:            ;
00CBEA:            ; ZeroPage Usage:
00CBEA:            ;       $EF
00CBEA:            ;       w/DEBUG enabled: $EB, $EC, $ED, $EE
00CBEA:            ;       Note: location $EF is saved and restored before driver exits
00CBEA:            ;
00CBEA:            WriteBlock:
00CBEA:            .IF DEBUG
00CBEA:               jsr  DSString
00CBEA:               .byte " Wt",0
00CBEA:               jsr  DisplayParms
00CBEA:            .ENDIF
00CBEA:
00CBEA:A5 45          lda  pdIOBufferH
00CBEC:48             pha
00CBED:A5 EF          lda  zpt1
00CBEF:48             pha
00CBF0:
00CBF0:            .IF DEBUG
00CBF0:               lda  CheckSumHigh
00CBF0:               pha
00CBF0:               lda  CheckSumLow
00CBF0:               pha
00CBF0:            .ENDIF
00CBF0:
00CBF0:20 FA CB       jsr  WriteBlockCore
00CBF3:
00CBF3:            .IF DEBUG
00CBF3:               ply
00CBF3:               sty  CheckSumLow
00CBF3:               ply
00CBF3:               sty  CheckSumHigh
00CBF3:            .ENDIF
00CBF3:
00CBF3:7A             ply
00CBF4:84 EF          sty  zpt1
00CBF6:7A             ply
00CBF7:84 45          sty  pdIOBufferH
00CBF9:60             rts
00CBFA:
00CBFA:            WriteBlockCore:
00CBFA:            .IF DEBUG
00CBFA:               stz  CheckSumLow
00CBFA:               stz  CheckSumHigh
00CBFA:            .ENDIF
00CBFA:
00CBFA:A9 00          lda  #0
00CBFC:9D 80 C0       sta  ATADataHigh,x           ;Clear the high byte of the 16 bit interface
00CBFF:                                            ; data latch
00CBFF:
00CBFF:20 78 CC       jsr  IDEWaitReady
00CC02:20 4D CC       jsr  Block2LBA               ;program IDE task file
00CC05:
00CC05:            ; Write sector from RAM
00CC05:A9 30          lda  #ATACWrite
00CC07:9D 8F C0       sta  ATACommand,x
00CC0A:20 78 CC       jsr  IDEWaitReady
00CC0D:
00CC0D:BD 8F C0       lda  ATAStatus,x              ;Check for error response from writing command
00CC10:29 09          and  #$09                     ;if DRQ=0 and ERR=1 an error occured
00CC12:C9 01          cmp  #$01
00CC14:D0 04          bne  wCommandOK
00CC16:
00CC16:            .IF DEBUG
00CC16:            ;warning this debug code trashes the Acc register
00CC16:               jsr DSString
00CC16:               .byte " Err!:",0
00CC16:               lda  ATAError,x
00CC16:               jsr  DSByteCRLF
00CC16:            .ENDIF
00CC16:
00CC16:            ; The drive has returned an error code. Just return I/O error code to PRODOS
00CC16:            ;
```

```
00CC16:A9 27          lda  #PRODOS_IO_ERROR
00CC18:38             sec
00CC19:60             rts
00CC1A:            ;
00CC1A:            ; Sector is ready to write
00CC1A:            ;
00CC1A:            wCommandOK:
00CC1A:A0 02          ldy  #2
00CC1C:84 EF          sty  zpt1
00CC1E:A0 00          ldy  #0
00CC20:
00CC20:            wLoop:
00CC20:BD 8F C0       lda  ATAStatus,x             ;Note: not using IDEWaitReady, using inline code
00CC23:30 FB          bmi  wLoop                   ;Wait for BUSY (bit 7) to be zero
00CC25:29 08          and  #$08                    ;get DRQ status bit
00CC27:F0 20          beq  wShort                  ;if off, didn't get enough data
00CC29:
00CC29:B1 44          lda  (pdIOBuffer),y
00CC2B:48             pha
00CC2C:
00CC2C:            .IF DEBUG
00CC2C:               clc
00CC2C:               adc  CheckSumLow
00CC2C:               sta  CheckSumLow
00CC2C:            .ENDIF
00CC2C:
00CC2C:C8             iny
00CC2D:B1 44          lda  (pdIOBuffer),y
00CC2F:9D 80 C0       sta  ATADataHigh,x
00CC32:
00CC32:            .IF DEBUG
00CC32:               adc  CheckSumHigh
00CC32:               sta  CheckSumHigh
00CC32:            .ENDIF
00CC32:
00CC32:68             pla
00CC33:3C 81 C0       bit  SetCSMask,x             ;any access sets mask bit to block IDE -CS0 on
00CC36:                                            ; I/O read to drive
00CC36:9D 88 C0       sta  ATADataLow,x            ;Remember that all write cycles are
00CC39:                                            ; preceded by a read cycle on the 6502
00CC39:3C 82 C0       bit  ClearCSMask,x           ;Set back to normal, allow CS0 assertions on
00CC3C:                                            ; read cycles
00CC3C:
00CC3C:C8             iny
00CC3D:D0 E1          bne  wLoop
00CC3F:E6 45          inc  pdIOBufferH
00CC41:C6 EF          dec  zpt1
00CC43:D0 DB          bne  wLoop
00CC45:
00CC45:            .IF DEBUG
00CC45:            ; Display the Checksum
00CC45:            ; warning this debug code trashes the Acc register
00CC45:               jsr  DSString
00CC45:               .byte " Chk$",0
00CC45:               lda  CheckSumHigh
00CC45:               jsr  DSByte
00CC45:               lda  CheckSumLow
00CC45:               jsr  DSByteCRLF
00CC45:            .ENDIF
00CC45:
00CC45:A9 00          lda  #0
00CC47:18             clc
00CC48:60             rts
00CC49:            ;
00CC49:            ; The Block was short, return I/O error code to PRODOS
00CC49:            ;
00CC49:            wShort:
00CC49:            .IF DEBUG
00CC49:            ; Display "W:Short"
00CC49:               jsr  DSString
00CC49:               .byte " W:Shrt:", 0
00CC49:
00CC49:               lda  zpt1
00CC49:               jsr  DSByte
00CC49:               tya
00CC49:               jsr  DSByteCRLF
00CC49:            .ENDIF
00CC49:
00CC49:A9 27          lda  #PRODOS_IO_ERROR
00CC4B:38             sec
00CC4C:60             rts
00CC4D:
00CC4D:            ;---------------------------------------------------------------------------
00CC4D:            ; Block2LBA - Translates ProDOS block# into LBA and programs devices' task file
00CC4D:            ;             registers.
00CC4D:            ;
00CC4D:            ; Input:
00CC4D:            ;       pd Command Block Data $42 - $47
00CC4D:            ;       X = requested slot number in form $n0 where n = slot 1 to 7
00CC4D:            ;       Y = $0n (n = slot#) for accessing scratchpad RAM;
00CC4D:            ;
00CC4D:            ; Ouput:
00CC4D:            ;       None
00CC4D:            ;
00CC4D:            ; ZeroPage Usage:
```

```
00CC4D:            ;       None
00CC4D:            ;
00CC4D:            ; CPU Registers changed:  A, P
00CC4D:            ;
00CC4D:            ; This function translates the block number sent in the PRODOS request
00CC4D:            ; packet, into an ATA Logical Block Address (LBA).
00CC4D:            ; The least significant 16 bits becomes the ProDOS block#.
00CC4D:            ; The most significant 16 becomes the ProDOS Drive #
00CC4D:            ;
00CC4D:            ; A ProDOS block and a ATA sector are both 512 bytes.
00CC4D:            ;
00CC4D:            ; Logical Block Mode, the Logical Block Address is interpreted as follows:
00CC4D:            ; LBA07-LBA00: Sector Number Register D7-D0.
00CC4D:            ; LBA15-LBA08: Cylinder Low Register D7-D0.
00CC4D:            ; LBA23-LBA16: Cylinder High Register D7-D0.
00CC4D:            ; LBA27-LBA24: Drive/Head Register bits HS3-HS0.
00CC4D:
00CC4D:
00CC4D:            Block2LBA:
00CC4D:
00CC4D:A9 E0          lda  #$E0                    ;1, (LBA), 1, (Drive), LBA 27-24, where LBA=1,
00CC4F:                                            ; Drive=0
00CC4F:9D 8E C0       sta  ATAHead,x               ;Talk to the Master device and use LBA mode.
00CC52:                                            ; Remember that this write will seen by both
00CC52:                                            ; the master and slave devices.
00CC52:            ;
00CC52:            ; Add BLOCKOFFSET to the ProDOS block number to offset the first drive block we
00CC52:            ; use. This keeps the device's first BLOCKOFFSET blocks free, which usually
00CC52:            ; includes a MBR at block 0.
00CC52:            ;
00CC52:B9 78 07       lda  DrvMiscFlags,y           ; bit 7 = raw block access
00CC55:29 80          and  #$80
00CC57:49 80          eor  #$80
00CC59:F0 02          beq  rawBlocks
00CC5B:A9 00          lda  #BLOCKOFFSET
00CC5D:            rawBlocks:                      ; A = $00 or BLOCKOFFSET
00CC5D:18             clc
00CC5E:65 46          adc  pdBlockNumber
00CC60:9D 8B C0       sta  ATASector,x             ;store ProDOS Low block # into LBA 0-7
00CC63:
00CC63:A5 47          lda  pdBlockNumberH
00CC65:69 00          adc  #0                      ;account for any overflow in LBA 0-7
00CC67:9D 8C C0       sta  ATACylinder,x           ;store ProDOS High block # into LBA 15-8
00CC6A:
00CC6A:B9 F8 04       lda DriveNumber,y
00CC6D:69 00          adc #0                        ;account for overflow from LBA 8-15
00CC6F:9D 8D C0       sta ATACylinderH,x            ;store LBA bits 23-16
00CC72:
00CC72:A9 01          lda #1
00CC74:9D 8A C0       sta ATASectorCnt,x
00CC77:60             rts
00CC78:
00CC78:            ;-------------------------------------------------------------------------------
00CC78:            ; IDEWaitReady - Waits for BUSY flag to clear, and returns DRQ bit status
00CC78:            ;
00CC78:            ; Input:
00CC78:            ;       X = requested slot number in form $n0 where n = slot 1 to 7
00CC78:            ; Ouput:
00CC78:            ;       Carry flag = DRQ status bit
00CC78:            ;
00CC78:            ; ZeroPage Usage:
00CC78:            ;       None
00CC78:            ;
00CC78:            ; CPU Registers changed:  A, P
00CC78:            ;
00CC78:            IDEWaitReady:
00CC78:BD 8F C0       lda  ATAStatus,x
00CC7B:30 FB          bmi  IDEWaitReady            ;Wait for BUSY (bit 7) to be zero
00CC7D:6A             ror                          ;shift DRQ status bit into the Carry bit
00CC7E:6A             ror
00CC7F:6A             ror
00CC80:6A             ror
00CC81:60             rts
00CC82:
00CC82:            ;-------------------------------------------------------------------------------
00CC82:            ; CheckDevice - Check to see if a device is attached to the interface.
00CC82:            ; Input:
00CC82:            ;       X = requested slot number in form $n0 where n = slot 1 to 7
00CC82:            ; Output:
00CC82:            ;       Carry flag: 0 = Device Present, 1 = Device Missing
00CC82:            ;
00CC82:            ; CPU Registers changed:  A, P
00CC82:            ;
00CC82:            ;  Checks to see if the drive status register is readable and equal to $50
00CC82:            ;  If so, return with the Carry clear, otherwise return with the carry set.
00CC82:            ;  Waits up to 10sec on a standard 1Mhz Apple II for drive to become ready
00CC82:            ;
00CC82:            CheckDevice:
00CC82:5A             phy
00CC83:3C 82 C0       bit  ClearCSMask,x           ;reset MASK bit in PLD for normal CS0 signaling
00CC86:A9 E0          lda  #$E0                    ;$E0 = [1, LBA, 1, Drive, LBA 27-24]  where
00CC88:                                            ; LBA=1, Drive=0
00CC88:9D 8E C0       sta  ATAHead,x               ;Make sure ATA master drive is accessed
00CC8B:
00CC8B:A0 00          ldy  #0
```

```
00CC8D:            chkLoop:
00CC8D:BD 8F C0       lda  ATAStatus,x
00CC90:29 D0          and  #%11010000
00CC92:C9 50          cmp  #$50                  ;if BUSY= 0 and RDY=1 and DSC=1
00CC94:F0 0D          beq  DeviceFound
00CC96:A9 C5          lda  #WAIT_100ms
00CC98:20 A6 CC       jsr  Wait                  ;Wait 100ms for device to be ready
00CC9B:C8             iny
00CC9C:C0 64          cpy  #100                  ;Wait up to 10 seconds for drive to be ready
00CC9E:                                          ; This time may turn out to be much shorter on
00CC9E:                                          ; accelerated Apple IIs
00CC9E:D0 ED          bne chkLoop
00CCA0:
00CCA0:38             sec                        ;set c = 1 if drive is not attached
00CCA1:7A             ply
00CCA2:60             rts
00CCA3:
00CCA3:            DeviceFound:
00CCA3:18             clc                        ;set c = 0 if drive is attached
00CCA4:7A             ply
00CCA5:60             rts
00CCA6:
00CCA6:            ;--------------------------------------------------------------------------
00CCA6:            ;  Wait - Copy of Apple's wait routine. Can't use ROM based routine in case
00CCA6:            ;         ROM is not active when we need it.
00CCA6:            ;
00CCA6:            ; Input:
00CCA6:            ;       A = desired delay time, where Delay(us) = .5(5A^2 + 27A + 26)
00CCA6:            ;       or more usefully: A = (Delay[in uS]/2.5 + 2.09)^.5 - 2.7
00CCA6:            ;
00CCA6:            ; CPU Registers changed:  A, P
00CCA6:            ;
00CCA6:            Wait:
00CCA6:38             sec
00CCA7:
00CCA7:            Wait2:
00CCA7:48             pha
00CCA8:
00CCA8:            Wait3:
00CCA8:E9 01          sbc  #1
00CCAA:D0 FC          bne  Wait3
00CCAC:68             pla
00CCAD:E9 01          sbc  #1
00CCAF:D0 F6          bne  Wait2
00CCB1:60             rts
00CCB2:
00CCB2:
00CCB2:
00CCB2:
00CCB2:            .IF DEBUG
00CCB2:
00CCB2:            ;--------------------------------------------------------------------------
00CCB2:            ; DisplayParms - Display the parameters of the ProDOS request
00CCB2:            ; Input:
00CCB2:            ;       None
00CCB2:            ; Ouput:
00CCB2:            ;       None
00CCB2:            ;
00CCB2:            ; ZeroPage Usage:
00CCB2:            ;       None
00CCB2:            ;
00CCB2:            ; CPU Registers changed:  A, P
00CCB2:            ;
00CCB2:            DisplayParms:
00CCB2:               jsr  DSString
00CCB2:               .byte " B:",0
00CCB2:
00CCB2:               lda  pdBlockNumberH
00CCB2:               jsr  DSByte
00CCB2:               lda  pdBlockNumber
00CCB2:               jsr  DSByte
00CCB2:
00CCB2:               jsr  DSString
00CCB2:               .byte " U:",0
00CCB2:               lda  pdUnitNumber
00CCB2:               jsr  DSByte
00CCB2:
00CCB2:               jsr  DSString
00CCB2:               .byte " A$",0
00CCB2:
00CCB2:               lda  pdIOBufferH
00CCB2:               jsr  DSByte
00CCB2:
00CCB2:               lda  pdIOBuffer
00CCB2:               bra  DSByte
00CCB2:
00CCB2:
00CCB2:            ;--------------------------------------------------------------------------
00CCB2:            ; DSString - Sends a String to the Super Serial Card in Slot 2
00CCB2:            ; Input:
00CCB2:            ;       string must immediately follow the JSR to this function
00CCB2:            ;       and be terminated with zero byte.
00CCB2:            ; Ouput:
00CCB2:            ;       None
00CCB2:            ;
```

```
00CCB2:            ; ZeroPage Usage:
00CCB2:            ;       MsgPointerLow, MsgPointerHi
00CCB2:            ;
00CCB2:            ; CPU Registers changed:  A, P
00CCB2:            ;
00CCB2:            DSString:
00CCB2:               phx                        ;save the X reg
00CCB2:               tsx                        ;put the stack pointer in X
00CCB2:               lda  MsgPointerLow
00CCB2:               pha                        ;push zero page location on stack
00CCB2:               lda  MsgPointerHi
00CCB2:               pha                        ;push zero page location on stack
00CCB2:
00CCB2:               lda  StackBase+2,x         ;determine the location of message to display
00CCB2:               clc
00CCB2:               adc  #$01                  ;add 1 because JSR pushes the last byte of its
00CCB2:               sta  MsgPointerLow         ; destination address on the stack
00CCB2:
00CCB2:               lda  StackBase+3,x
00CCB2:               adc  #0
00CCB2:               sta  MsgPointerHi
00CCB2:
00CCB2:            dss1:
00CCB2:               lda  (MsgPointerLow)
00CCB2:               beq  dssend
00CCB2:               jsr  DSChar                ;display message
00CCB2:               inc  MsgPointerLow
00CCB2:               bne  dss1
00CCB2:               inc  MsgPointerHi
00CCB2:               bra  dss1
00CCB2:
00CCB2:            dssend:
00CCB2:
00CCB2:               lda  MsgPointerHi
00CCB2:               sta  StackBase+3,x
00CCB2:               lda  MsgPointerLow
00CCB2:               sta  StackBase+2,x         ;fix up the return address on the stack.
00CCB2:
00CCB2:               pla
00CCB2:               sta  MsgPointerHi          ;restore zero page location
00CCB2:               pla
00CCB2:               sta  MsgPointerLow         ;restore zero page location
00CCB2:               plx
00CCB2:               rts                        ;return to location after string's null.
00CCB2:
00CCB2:            ;-------------------------------------------------------------------------------
00CCB2:            ; DSByteCRLF - Sends a Hex byte followed by a CR LF to the Super Serial
00CCB2:            ; Card in Slot 2
00CCB2:            ;
00CCB2:            ; Input:
00CCB2:            ;       A = Hex number to display
00CCB2:            ; Ouput:
00CCB2:            ;       None
00CCB2:            ;
00CCB2:            ; CPU Registers changed:  A, P
00CCB2:            ;
00CCB2:            DSByteCRLF:
00CCB2:               jsr DSByte
00CCB2:            DSCRLF:
00CCB2:               lda  #$0D
00CCB2:               jsr  DSChar
00CCB2:               lda  #$0A
00CCB2:               bra  DSChar
00CCB2:
00CCB2:            ;-------------------------------------------------------------------------------
00CCB2:            ; DSByte - Sends a Hex byte to the Super Serial Card in Slot 2
00CCB2:            ; Input:
00CCB2:            ;       A = Hex number to display
00CCB2:            ; Ouput:
00CCB2:            ;       None
00CCB2:            ;
00CCB2:            ; CPU Registers changed:  A, P
00CCB2:            ;
00CCB2:            DSByte:
00CCB2:               pha
00CCB2:               lsr a
00CCB2:               lsr a
00CCB2:               lsr a
00CCB2:               lsr a
00CCB2:               jsr DSNibble
00CCB2:               pla
00CCB2:            DSNibble:
00CCB2:               and #$0F
00CCB2:               ora #$30
00CCB2:               cmp #$30+10
00CCB2:               bcc digit
00CCB2:               adc #6
00CCB2:            digit:
00CCB2:               bra DSChar
00CCB2:
00CCB2:            ;-------------------------------------------------------------------------------
00CCB2:            ; DSChar - Sends a char to the Super Serial Card in Slot 2
00CCB2:            ; Input:
00CCB2:            ;       A = Character to Send
00CCB2:            ; Ouput:
```

```
00CCB2:            ;       (data out serial port)
00CCB2:            ;
00CCB2:            ; ZeroPage Usage:
00CCB2:            ;       None
00CCB2:            ;
00CCB2:            ; CPU Registers changed:  P
00CCB2:            ;
00CCB2:            DSBlank:
00CCB2:               lda #$20
00CCB2:            DSChar:
00CCB2:               pha
00CCB2:               phy
00CCB2:               lda  mslot
00CCB2:               and  #$0f
00CCB2:               tay                           ;Y reg now has $0n for accessing scratchpad RAM
00CCB2:               lda  SerialInitDone,y
00CCB2:               cmp  #$A5
00CCB2:               beq  dsc0
00CCB2:
00CCB2:            ; Init the serial port if sig byte is not $A5.
00CCB2:            ; Set up serial port on the Apple Super Serial card. Always assume slot 2.
00CCB2:               lda  #$1f
00CCB2:               sta  $c0ab                    ;control register
00CCB2:               lda  #$0b
00CCB2:               sta  $c0aa                    ;format
00CCB2:               lda  $c0a9                    ;clear status
00CCB2:               lda  #$A5
00CCB2:               sta  SerialInitDone,y
00CCB2:
00CCB2:            dsc0:
00CCB2:               lda  $c0a9
00CCB2:               and  #%00010000               ;Transmit register empty?
00CCB2:               beq  dsc0                     ;If not, wait
00CCB2:
00CCB2:               ply
00CCB2:               pla                           ;get byte back
00CCB2:               sta  $c0a8                    ;send it
00CCB2:               rts
00CCB2:
00CCB2:            .ENDIF
00CCB2:
```